November 19, 1993

A Proposal for a Framework for Mathematical Objects and Methods

R.M. Hjellming

1 Introduction

Some of the important classes in any object-oriented data processing system should be those based upon mathematical entities and operations. This was recognized early in the AIPS++ project and led to a prototyping emphasis on array-based classes and tables. However, the next layer of classes, and the way algorithms would be organized around mathematical classes, has not been defined with any completeness. In addition, there may be some additional attributes of vectors, matrices, and arrays that would be useful in a system where they are viewed as mathematical "components" of tables. Therefore we will discuss **mvector**, **mmatrix**, and **marray** classes that are almost identical to AIPS++ **vector**, **matrix**, and **array** classes, but will have one additional public attribute (dimnames) and additional methods to support the mathematical system being proposed. In addition, we will discuss a **mtable** class which represents a view of a table as part of a vector, matrix, etc., oriented system of mathematical processing, and which will have additonal methods reflecting its role relative to other data objects. The classes we propose represent a design in the context of a framework, a "larger building block" than classes (Firesmith 1993, Mössenböck 1992). This framework is designed for data processing using mathematical objects. This document is planned to be complete, but will stop short of specific astronomical applications using these mathematics based classes.

An important distinction should be made between the data objects and methods discussed in this document and the C++ classes, designed for power and efficiency, that they are based upon. This distinction deals with the data objects from the point of view of the scientist and not necessarily the programmer. We tacitly assume that some of these objects, and some of their methods, are intended solely to make programming and algorithm modification easy for astronomers who are not experts at the sublety and power of C++. Providing efficient classes is basic, but a secondary goal of providing classes expressing a scientist's point of view is probably essential to easy use of AIPS++ for programming by astronomers.

Some of the concepts to be discussed are derived from a study of the object-oriented data processing and visualization language S (Becker *et al.* 1988, Chambers and Hastie 1991), using the version released by AT & T in 1991, as implemented for workstations and PCs in the commercial implementation product S+ .

The issues that we should consider are not solely based upon a list of classes with associated methods. For this reason we will use a slightly different language to describe needed entities.

First of all, we will discuss fundamental data objects in a manner which seemingly separates these data objects from the methods that use, modify, and create basic data objects. It seems useful to distinguish between the user-invocation of a method, methods that represent specific algorithms using data objects, and the methods of classes that instantiate data objects.

Amongst the completely new data objects that we suggest as useful for AIPS++ are: **mlist**, **factor**, **grid**, and the abovementioned **mtable**. Minor augmentation of the attributes of AIPS++ Array-based classes, and addition of important methods for Array-based classes will be described for **marray**, **mvector**, and **mmatrix**. An **mtable** is a **table** when viewed as a collection of one or more vectors of numeric and/or string types, each with the same length, with character string labeling for rows and columns.

2 Basic Data Values and Objects

2.1 Data Values

From the point of view of the mathematics, there are six basic data types that are potential elements of all data objects:

- numeric (ordinary numbers, stored as integer, real, or double)
- logical (TRUE or FALSE with associated logical operators)
- complex (complex number numeric values)
- strings (sequences of characters)
- NA (a Not Available indicator for data values that plays a general role in mathematics to represent missing data, like the blank pixel concept in astronomy, IEEE NaNs, or the results of indeterminate computations like 0/0)
- NULL (a concrete return type for any data value; useful for positively returning no values or for establishing whether values were returned)

The first four types of data values (numeric, complex, logical, and strings) represent four "modes". Additional modes can be defined for higher level constructs. The use of fuzzy numbers that is being investigated for error propagation in AIPS++ will use constucts based upon vectors of numeric or complex type, with special rules for their arithmetic, but will not be discussed further in this document.

The use of NA and NULL at the basic level of data values seems to allow added flexibility in mathematical algorithms, with some methods using, or allowing, these data values, and some methods requiring that NA or NULL not be included in the data objects.

Now we can define a data object as an atomic or non-atomic collection of abovementioned data values. In atomic data objects all data values are of the same mode (numeric, complex, logical, and strings), whereas in non-atomic data objects there are mixtures of atomic data objects with different modes.

2.2 Data Objects

The basic data objects are defined by their attributes, with length, mode, dim (dimension), dimname, and class being most fundamental. The following table summarizes the basic mathematical data objects, their attributes, and their role.

Table 1 - Data Objects

Class	Atomic	Attributes					Role
Mvector	T	Length	Mode	Dim	Dimname		Most basic data object
Mmatrix	T	Length	Mode	Dims	Dimnames		Rows/columns of vectors
Marray	T	Length	Mode	Dims	Dimnames		N-dimensional array
Mlist	F	Length	Mode	Names			Ordered collection of data objects
Mtable	F	Length	Mode	Names	Row.Names		Generalized table with columns of numeric, logical or character data values
Factor	F	Length	Mode	Names	Levels		Qualitative identification and labeling of data
Grid	F	Length	Mode	Dims	Dimnames	Coords	N-dimensional array with even axis intervals

The first three data objects in Table 1 are augmentations of the classes already developed for AIPS++, with the specific addition of Dimname attributes for each dimension. These are named **mvector**, **mmatrix**, and **marray** to distinguish them from the AIPS++ classes that have already been implemented; dimnames allow a simple assignment expressions between these data objects and **mtables**, and allow all data objects to have vectorized selection/logic based upon key words.

The **mtable** data object is a specific view, or form, of an AIPS++ **table** that allows one to easily compose and decompose it from/to other data objects using methods related to **mvector**, **mmatrix**, **marray**, and other **mtable** data objects. As with an AIPS++ **table**, an **mtable** can contain columns of any of the other atomic data objects.

While many of these data objects will be small enough to fit into memory, with many cases of interest that will not be true; however, the use of buffered I/O is a prime example of

an implementation detail that should be hidden as part of the general data base manager for objects of all kinds.

The **mlist** data objects are associations of other data object components that are formed by a *mlist*(**o1**, **o2**, ..., **oN**) method, and which have a syntax allowing mathematical operations on component and sub-component data objects. The **mlist** data object allows simple association of related data objects resulting from methods or more complicated multi-object algorithms, without requiring construction of new kinds of data objects, since they are just different mlists of standard data objects. For example, one can map a AIPS FITS image into an **image** object with the method

mlist(labels=**labelvector**,values=**valuevector**,axes=**axesmatrix**, pixels=**imagearray**)

where labels is a vector of string-like header information, values a vector of the global numeric information for the image, axes is a matrix of numbers describing the values (and state) of the image coordinates, and pixels is the array of numbers which contain the image values. Because of the dimnames attributes of vectors and arrays, the keyword=value syntax of FITS images maps directly into vector, matrix, and array data objects. The data object components of observations, measurement sets, telescope models, etc., can be formed, referred to, and operated on with a combination of the syntax of *mlist* and *mtable* methods. The following is an example of a listing of contents of an **mlist** image data object derived from AIPS/FITS:

```
labels:
                    NAME
OBJECT           "NCYG92"
TELESCOP            "VLA"
INSTRUME            "VLA"
OBSERVER "R.M.HJELLMING"
   UNITS         "JY/BEAM"

values:
          VALUE
 NAXIS 4.00E+00
 EPOCH 1.95E+03
 SCALE 2.00E-04
OFFSET 0.00E+00
 BLANK 0.00E+00

axes:
```

```
         RA.SIN.DEG  DEC.SIN.DEG      FREQ.HZ
  DIM  1.750000E+02 1.750000E+02  1.00000E+00
CRBLC  1.600000E+02 1.600000E+02  1.00000E+00
CRVAL  3.072800E+02 5.246208E+01  2.24851E+10
CRINC -6.944444E-07 6.944444E-07 -5.00000E+07
CRREF -2.830000E+00 0.000000E+00  0.00000E+00

pixels:

[matrix of numbers]
```

Flexible *mlist* construction may be more of a UI-related operation because of the difficulties of implementation in C++.

Factor data objects allow useful identification of qualitative descriptions of data that can be utilized by logical operations in array-oriented or vectorizable algorithms. Each is essentially a vector of integers identifying levels, with an associated vector of names for each level. Constructs like this can be used for many things, e.g. data quality identification, weights, source/field identification, and so on. **Factors** are concrete classes that are a bridge between numeric arrays and keyword identification used in vectorized operations. An example of a **factor** object is the following, where the data in the object is a vector of integers identifying different "levels" in the object, and levels is a vector of strings indicating that each level identifies a source name:

values: 3,3,3,3,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1
levels: "2005+403","NCYG92","3C286"

where this vector matches, say, a row of source data so one can operate on the source data using selection methods attached to the **factor** data object.

The **grid** data object is one of the most interesting, and a major component of many applications, while retaining entirely mathematical characteristics and methods. In the astronomical context it has the mathematical essence of a time series, a spectrum, a gridded u-v array, and a regular image. In AIPS++ the work on GridTool and FFTtool has developed some of the methods for **grid** data objects. As a higher level component of the data objects in Table 1, **grid** is a **mvector**, **matrix**, or **array**, with the added attribute of coord = (begin, end, interval, units), and methods for extracting selected coordinate information and scaling coordinate values. Methods of the **grid** data objects will use linear equations involving the elements of the coord attribute and the range of indices for each axis.

It is possible that units for pixels should be additional **grid** attributes, since they affect the scales of stored numeric values, but it is clear that other astronomical content, like representation, reference frame, measure, etc., belongs to higher level classes. The dimnames attribute of

grid objects allows keyword identification of each dimension. This may be sufficient for including units since one can use dimnames with labels like "u.nanosecond". Later we will discuss a more extensive list of methods for **grid** data objects.

The evolution of the design and implementation of the framework of classes for images is where further isolation of a possible **grid** class should be examined, ensuring that it has methods that are generically mathematical and independent of the image-handling problem.

Reinforcing the view that a mathematically defined **grid** data object is a powerful construct for algorithms, it is possible to use FFT and related methods in S /S+ to produce, and operate on, their **matrix** and **time series** objects to represent, and make transformations between, gridded u-v data and images.

The emphasis on optimizing for mathematical operations wherever possible leads to methods like *ifelse*(logicalexpr, expr1, expr2) where a logical operation on an atomic data object results in choice between two expressions, expr1 and expr2, for the element-by-element operation involving that object, depending upon the result of the logical operation for each element. This type of mathematical operation (and the analogous *if*, *switch*, *all*, and *any* methods, cf. Table 2) is part of the reason for both **factor** data objects and the association of dimnames with vectors, matrices, and arrays, since names as keywords can then be used in logical operations. The current work on masked arrays is related to the development of these sorts of vectorize logic operations.

2.3 Formation, Testing, and Coercion of Data Objects

Formation of more complicated data objects from simpler ones, and vice versa, should be possible with simple syntax that hides the vector, matrix, etc., nature of the objects. Vector data objects can be formed by a *sequence* method, a *repetiton* method, or a *combine*(**o1**, ..., **oN**) method. Vectors should be formable into matrices with methods like *rbind*(**o1**, ..., **oN**) (for rows) and *cbind*(**o1**, ..., **oN**) (for columns). All data objects should be formable into **mlist** data objects by the *mlist* (**o1**, ..., **oN**) method, and atomic data objects formed into **mtable** objects with an *mtable* (**o1**, ..., **oN**) method. When implemented in C+ the first argument for each methods will be the number of elements in each object list. The reverse extraction of simpler data objects from more complicated ones is a question of extraction based upon some multi-level "subscript" notation.

Testing and coercion are useful concepts for mathematical handling of different, but relat-

able, data objects. Each data object can have a *is.objecttype*(**object**) method that tests for what is needed in some mathematical expression or algorithm, returning TRUE or FALSE, and a *as.objecttype*(**object**) method then returns a different type of data object that can be formed from the input object.. The testing method is useful in algorithms. The coercion method aids extraction of one type of data object (e.g. **mmatrix**) from others (e.g. **mtable**).

2.4 The Formula Class

The need for using arbitrary formulas or equations in the fitting or modeling of data is obvious, and a construct that could be useful for supplying formulas to methods, producing strings in the form of formulas in labels, and doing some symbolic algebra and symbolic evaluation, is the **formula** class. As a data object it takes as input string of characters identifying arithmetic operations, variable names, and parameters to be determined. It has basic symbolic algebra methods like *substitute*, *parse*, *expression*, *derivative*, *evaluate*, etc., that allow mathematical expression, decomposition, manipulation, and use in evaluation of symbolic expressions to return values for the quantities modeled by the formula(s).

2.5 Methods for Atomic Data Objects

All the ordinary operator-like operations involving vectors and matrices are assumed to be present, with * used for element by element multiplication as done with the AIPS++ Array class. Using a *crossprod* method for M x V and M x M with M and V (or M) as arguments, is reasonable. However, in a mathematical system there are distinctions based upon whether the vector or matrix is a transpose or not that can be checked at run time based upon a transpose of non-transpose identification, or left as a potential programmer error. Probably it is best to expect the application programmer to write tran(V) or tran(M) when mathematically required.

Table 2 lists methods for various atomic data objects, with V, M, A, and G indicating whether they apply to **mvector**, **mmatrix**, **marray**, and/or **grid** data objects, and N, C, and/or L indicating whether they are applicable to numeric, character, and/or logical data values.

Table 2

Methods of Atomic, Numeric Data Objects

combine	**V**	**NC**	**combine mlist of numbers into a vector**

rep	V	NC	form vector replicating mlist of numbers
sequence	V	N	form vector from vstart to vstop using optional step or length parameters
tran	VMAG	N	tranpose
diag	V	N	from diagonal matrix with input vector on diagonal
rbind	V	NCL	from matrix from mlist of vector objects with each vector becoming a row
cbind	V	NCL	from matrix from list of vector objects with each vector becoming a column
sort	V	NC	sort vector on elements
reverse	V	NC	reverse elements of vector (often after sorting)
order	V	NC	return integer vector containing the permutation that will sort teh input into ascending order
rank	V	N	returns a vector with ranks of the input vector
diff	VMAG	N	returns a VMAG with the differences between adjacent elements of the input data object
unique	V	NC	returns an object like the input but with repeated values deleted
duplicated	V	NC	returns a vector of logical values for an input object indicating whether elements are duplicated or not
sum	VMAG	N	returns the sum of all elements of input object
prod	V	N	returns the product of all elements of input object
max	VMAG	N	returns largest value in input object
min	VMAG	N	returns smallest value in input object
range	VMAG	N	returns vector of smallest and largest values
all	VMAG	L	returns TRUE if all elements of input logical expression(s) are TRUE, returns FALSE otherwise
any	VMAG	L	evaluates to TRUE if any elements of input logical expression are TRUE, returns FALSE otherwise
if	VMAG	L	evaluate an expression for each element if a logical expression for each element is true

ifelse	VMAG	L	depending upon logical expression evaluation for each element, performs onee of two operations on or with each element
switch	VMAG	N	depending upon the integer returned by an expression one of a series of expressions is used to used to return a value for each element of the input data object
apply	VMAG	N	Apply a function defined by a formula object to all elements of the input data object
outer	VMAG	N	Apply a function defined by a formula object to two input data objects with the same shape
mean	VMAG	N	returns mean of all elements of data object, optional trim parameter specifying range of values to be averaged
median	VMAG	N	returns median of all elements of data object, optional trim parameter specifying range of values to be considered
quantile	VMAG	N	returns vector of desired probability levels for a data object, as determined by optional input vector of desired probabilities
var	VM	N	returns variance of data object (for optionally specified range of values); if a matrix, columns represent variables and rows represent measurements
cor	M	N	return correlation matrix for optional range of values
cov	M	N	return covariance matrix for optional range of values
round	VMAG	N	return for each element the integer above or below value + 0.5
signif	VMAG	N	return for each element a value with rounding in the specified significant figure
cumsum	VMAG	N	returns an object for which each element is the sum of all elements to that point
cumprod	VMAG	N	returns an object for which each element

			is the product of all elements to that point
distrib	VMAG	N	returns for each element a value of a named probability distribution over an option range of values
fft	VMAG	N	transform a real or complex data object by a direct or inverse FFT
autocorr	VMAG	N	return autocorrelation function of data object
lag	VMAG	N	return same object with data lagged by specified intervals in one for one or more dimensions (mainly for case of a time-like dimension)
convolve	VMAG	N	convolve a function specified by a formula object with a specified span producing a smoothed version of the original data object
aggregate	VMAG	n	convolve, average, or smooth one or more dimensions to a data object with a reduced number of data points spanning the same range
subset	VMAG	N	return a subsection of a data object based upon a range specification for each dimension
coord	G	N	return coordinate values for specified elements

It is obvious from many of these methods that this puts considerable emphasis on vectorizable operations so one can express mathematics with operations that are accomplished as efficiently as possible by internal mechanisms hidden from the programmer.

The basic constructor methods (*mvector*, *mmatrix*, *grid*, etc.) have obvious use and syntax, and some of the operations in Table 2 reflect other ways of constructing these objects. The *is.class* for testing and *as.class* methods for testing classes and coercing classes important for specific use and decomposition of mathematical objects. Methods like *assigndim* and *assigndimnames* are needed as part of of the composition of data objects.

3 Linear and Non-Linear Algebra Methods and Objects

3.1 Scope

Interesting linear algebra methods begin when one considers inversions and decompositions of a matrix($\mathbf{A}$), or when one is solving linear equations like

$$\mathbf{Ax} = \mathbf{d} \tag{1}$$

where **d** is a vector of "measurements", and **A** is a "model" for how an vector of unknowns (**x**) reproduces the measurements. For non-linear models one is usually solving for a parameter set P for equations modeling measurements such as

$$O(P) = \mathbf{d} \tag{2}$$

where O is some non-linear operator. Classes and methods used to determine **x** for linear equations, and P for non-linear equations, are what we will call solvers, and to avoid confusion we will call the generic method for solving Equation (1) *linsolve* and the generic method for solving Equation (2) *nonlinsolve*.

The algorithms for inversion, decomposition, and *linsolve* with a single constraint equation are reasonably well understood and we will discuss the major methods and resulting data objects in the next section. However, there are two important areas where the algorithms are not standardized for either *linsolve* or *nonlinsolve*: multiple constraint equations on the same set of unknowns; and mixtures of "equality" and inequality constraint equations. A simultaneous solution for unknown parameters involving four polarization equations, and inequalities imposing positivity or some other range of parameters, are cases where this would be useful. We will not say more about this here, but experimentation with multiple constraint solvers is under way. However, we can proceed knowing that all such augmentations of this type are founded on the basic data objects and the standard linear and non-linear methods that we discuss in this document.

3.2 Inversion and Decompositon

Many algorithms are based upon decomposition (also called factorization) of a square or rectangular matrices into combinations of diagonal, upper triangular, lower triangular, or other matrices. In the We propose that all decompositions produce a **mlist** data object with components representing the vectors or matrices produced by the decomposition, and each of these has methods specific to using that decomposition, or generic methods, used for all decompositions. Table 3 lists the most important decomposition objects with some comments about their role.

Table 3

Matrix Decomposition Data Objects

Name	Role
LUdecomp	Fastest but least robust when used in solvers,
CHOLdecomp	Choleski decomposition, best for non-zero, square matrices, particularly the ''normal'' equation matrix (tran(A) A)
QRdecomp	Best compromise of speed and robustness when used in solvers, should be the default decomposition, using Householder transformations to perform the decomposition
SVdecomp	Singular value decomposition; slowest but most robust in solvers, particularly when allowing user interaction and modification of vector of singular values

Table 4 lists some of the methods that either use or support matrix decomposition objects.

Table 4

Methods Related to Matrix Decomposition Data Objects

Name	Role
backsolve	Takes an upper triangular matrix (R) and a vector (d) and solves for the vector (y) in R y = d
forwardsolve	Takes an lower triangular matrix (L) and a vector (d) and solves for the vector (x) in L x = d
invert	Takes either a decomposition object for a matrix A, or A and the name of an inversion or decomposition method(QR, LU, Choleski, SVD), and attempts to obtain Ainv such that crossprod(Ainv,tran(A)) = 1, using a default or supplied tolerance value; the default should be based on QR decomposition
determinant	Return a determinant of a matrix (or decomposition of of matrix) using a default or supplied method

The methods in Table 4 return a standard **mmatrix** data object and do not require creation of new data objects.

3.3 Linear Algebra Solvers and Error Analysis

The main use of matrix inversion and decomposition is in solvers with associated methods for analysis of the errors in the solution. Table 5 lists some of the data objects and methods that are basic for this form of data processing. We identify one of the matrix decomposition objects with the letter D, of any type, particularly those listed in Table 4. The letter F identifies a **formula** object supplied to a data fitting method.

Table 5

Methods Related to Linear Solvers and Error Analysis

Name	Objects	Role
linsolve	D	Returns the inverse of a matrix using methods associated with the decomposition object
linsolve	D,V	Solves for vector x using methods of the decomposition object, and the supplied ''measurement'' vector
linsolve	M	Returns the inverse of a matrix using a method specified by inversion or decompostion type
linsolve	D,V	Solves for vector x using a method specified by inversion or decomposition type, and the supplied ''measurement'' vector
eigen	D[,M}	returns a data object with eigenvalues and eigen vectors of matrix or previously computed matrix decomposition
lsfit	V,V[,V],F	Returns ''fit'' data object for given (linear) formula object used in a least squares fit to a ''measurement'' vector associated with a vector of independent variables, and an optional vector of weights, utilizing a specified decomposition method for the (tran(A) A) matrix
memfit	V,V[,V],F	Similar to lsfit except MEM is used with a constraint supplied with a formula object

In Table 5 we identify a **fit** data object that contains the solutions to $\mathbf{A}\ \mathbf{x} = \mathbf{D}$, and probably the residuals and the covariance matrix. The **fit** data objects will have additional methods return a list of traditional components of error analysis such as vectors of variance, standard deviation, correlation coefficients, etc. In the first sentence of this paragraph the word "probably" is used because it is unclear, when size of data sets becomes large, whether ancillary results should be automatically generated, or whether they should be generated from special methods using small **fit** data objects, the matrices (or decomposition objects), and the input data vectors.

There are a large number of useful fitting algorithms that could be listed in Table 5, and we are mentioning only a couple.

Please note that we are not implying that these linear algebra methods provide everything needed for AIPS++ solvers. Standard non-linear solvers, or specially coded equivalents appropriate to very large data sets are essential for many of the interesting cases; however many of these will use and operate upon basic data objects.

3.4 Number Generators

For data analysis one often needs to generate model data assuming values of the mean, standard deviation, and type of distribution (Uniform, Gaussion, Poisson, etc.). These are then used to compare real data with model distributions. Therefore part of the basic mathematics subsystem should be methods to fill VMAG data objects with model distributions of data, given specification of the desired distribution and its parameters. The basic versions already exist in AIPS++, but may need augmentation to work with the framework discussed in this document.

3.5 Statistical Analysis Methods

The statistical analysis of solutions, or fits, to data is a highly developed field for linear equations and data obeying well-understood statistics. A large number data objects and methods can be used for these standard statisical analysis methods. At first look these are the primary components of the statistics-oriented S+ . However, for the purpose of this document we eschew discussion of these approaches, assuming they will be developed as higher level applications.

3.6 Non-Linear Algebra Solvers and Error Analysis

In Table 6 we list two generic methods for solving non-linear equations. An initial version of *nonlinsolve* has already been protoyped in C++ using the Levenberg-Marquandt compromise for the Gauss-Newton method; however, it still needs cosmetic changes before being checked into the system. Solvers based on the Levenberg-Marquandt compromise are amongst the more robust nonlinear solvers. An early version of an iterative solver, based upon a sequence of assumed model equations supplied by formula objects, should also be prototyped early, since solvers of this type work well with most data from instruments like the VLA.

Table 6

Methods Related to Non-linear Solvers

nonlinsolve	V[,M]	Solve equations of the form O(P) = d where d is a supplied data vector, the form of the nonlinear function O is supplied with a formula object, and M is an optional matrix that is part of the nonlinear function definition
itersolve	V[,M]	solves using iteration, and supplied formula objects describing the first and subsequent formulas, to solve for a parameter set

4 Ancillary Methods for Basic Data Objects

While one can view data display as separate from data processing, there is great programming power in having a set of simple display, and display-interaction, methods operating on basic data objects. These ancillary methods are very important in allowing flexible prototyping of higher level display, editing, etc., methods.

Mathematical data processing is inseparable from interaction with, and display of, original, intermediate, and resulting data objects. Methods for displaying basic data objects like **mvector**, **mmatrix**, **marray**, and **grid**, and some capability to identify and locate the names or locations of data points in displays allows the programmer to provide powerful means for intervening in the data processing process. In Table 7 we list some of the useful, generic or special,

methods. There is a wide range of parameters controlling the displays and interactions that we will mostly not discuss in detail in this document. Some deal with parameters of each method, and some deal with general parameters of displays that control size, windowing, labeling, tic marks, axis label values, etc.

Table 7

Methods for Displaying and Interacting With Basic Data Objects

Name	Input Object(s)	Role
plot	V	Make x-y plot of vector elements vs its indices
plot	V1,V2	Make x-y plot of V2 vs V1
plot	M	Make x-y plot of matrix values in an x-y grid determined by matrix indices
plot	M,V1,V2	Make x-y plot of matrix values in an x-y grid determined by supplied ''coordinate'' vectors
hist	V[,V]	Plot a histogram for a vector as a function of is index or an a ''coordinate'' vector
picture	M	Make monochrome or color display of matrix in an x-y grid determined by matrix indices
picture	M,V2,V2	Make monochrome or color display of matrix in an x-y grid determined by supplied ''coordinate'' vectors
legend		Add ''legend'' to picture display indicated mapping of gray scale or color into pixel values
contour	M	Make contour display of matrix in an x-y grid determined by matrix indices
contour	M,V2,V2	Make contour display of matrix in an x-y grid determined by supplied ''coordinate'' vectors
perspec	M	Make perspective or ruled surface display of matrix in an x-y grid determined by matrix indices, with or without hidden line removal, with specified or default view angle parameters
perspec	M,V2,V2	Make perspective or ruled surface display of

	matrix in an x-y grid determined by supplied ''coordinate'' vectors
locate	For a specified plot or picture in a display window, starts a mouse and mouse-button driven procedure for clicking cursor (on/near) a data point allowing return of ''matrix'' value and x-y coordinates which are highlighted and added row by row to a matrix of identified value-x-y vectors; using one button for locating, and another button for removing a previously located data vector; requires display boxes in border area that one can click on to change size of locator box, exit from the locate method, or change other obvious parameters of the process
identify	Similar to locate, however one gets either or both row and column names from the matrix displayed next to the select data point
specdisp M[,V]	monochrome or color displays of rows and/or columns of vector/matrix data values as a function of row/column indices; with locate and identify methods invocable once the bands are displayed (allows TVFLAG-like display and interaction with basic data objects)
specdisp M[,V],V1	Same as above but supplied coordinate vector is used for the independent variable in a band-like display with monochrome or color coded data values as a function of an independent variable which is an index or a ''coordinate'' vector
rotate A	Initiate an interactive 3-d display of 3-D array values as points in a 3-D space with respect to displayed and labeled (from array axis names) axes in array index form, with pressing a mouse button

```
                    on boxes labeled right, left, circle, up, down,
                    causing the displayed data points and axes to
                    change their 3-D viewing angle
rotate  V1,V2,V3    same as above but x,y,z vectors are supplied
rotfind A           adds to a rotate method capabilities to locate and
                    identify data values as discussed above for the
                    locate and identify methods
rotfind V1,V2,V3    same as above but x,y,z vectors are supplied
reshow  C           redisplay previously saved display object on disk
movie   C           display movie sequence of previously saved
                    display objects stored on disk
panel   C           display sequence of objects on disk in paneled
                    sub-windows
```

The *reshow*, *movie*, and *panel* methods require saving display objects.

Each display windows should have buttons bringing up menus that

- Print plot in window to default printer
- Print plot in window to user specified printer
- Save plot in re-displayable form on disk
- Change properties of display, including (where appropriate)
  - monochrome vs color
  - select choice of color mapping from data value
  - initiating an interactive and separate window for interactive control of transfer function
  - zooming in and out

or one could have command line methods that carry out these functions; however, if one is in a window display domain, these functions are best handled with buttons/menus associated with these windows.

The *identify* and *locator* methods, or others allowing display and return of data coordinates can be used to build graphical editing capabilities for both basic data objects and higher level

displays of spectal, u-v, etc., data; however, the latter are beyond the level of things useful for basic data objects.

5 What's Next

Preliminary circulation of drafts of this document indicate that, as a proposal, there is considerable agreement that much of this should done in AIPS++. Most of the remaining questions relate to how different things would be implemented, what the priorities would be, and who would do what on what time scale. Therefore we leave this document in the current state, declared as finished, and distribute it as an AIPS++ note. With wider distribution useful suggestions will arise, and plans for implementation and prioritization can be developed.

6 References

Becker, R.A., Chambers, J.M., and Wilks, A.R. 1988, *The New S Language*,
(Wadsworth & Brooks/Cole, Pacific Grove)

Chambers, J.M., and Hastie, T.J. 1992, *Statistical Models in S*,
(Wadsworth & Brooks/Cole, Pacific Grove)

Firesmith, D.G. 1993, *J.O.O.P.*, **6**, No. 6, 6.

Mössenböck, M. 1992, *Object Oriented Programming in Oberon-2*, (Springer-Verlag, New York)