A Software Package for Interactive Multi-dimensional Image/Spectral Reconstruction, Processing and Analysis

USER'S GUIDE

Department of Electrical and Computer Engineering
National Center for Supercomputing Applications
Biomedical Magnetic Resonance Laboratory
University of Illinois at Urbana-Champaign

Contents

1 PRELIMINARIES **7**

1.1 Overview . . . 7
1.2 Obtaining V . . . 7
1.3 Installing V . . . 8
1.4 Contents . . . 8

2 THE V INTERFACE **11**

2.1 Register Stack and File I/O . . . 12
2.2 Data Manipulation . . . 12
2.2.1 Stack Manipulation Functions . . . 13
2.2.2 Register Manipulation Functions . . . 13
2.3 Graphical Display . . . 14

3 V FUNCTIONS **15**

3.1 REGISTER MANIPULATION . . . 16
3.1.1 CENTER . . . 16
3.1.2 CHUNK . . . 16
3.1.3 GROUP . . . 16
3.1.4 LABEL . . . 16
3.1.5 MERGE . . . 16
3.1.6 MIRROR . . . 16
3.1.7 POP . . . 16
3.1.8 PUSH . . . 17
3.1.9 REDIM . . . 17
3.1.10 REORDER . . . 17
3.1.11 SETTYPE . . . 17
3.1.12 SHOW_REG . . . 17
3.1.13 SPLIT . . . 17
3.1.14 SWAP . . . 18
3.1.15 XCHG . . . 18
3.2 DATA SYNTHESIS . . . 18
3.2.1 RANDOM . . . 18
3.2.2 UNIFFILL . . . 18
3.2.3 ZEROPAD . . . 18

3.3 INPUT/OUTPUT . . . 19
3.3.1 DUMP2TERM, DUMP2FILE . . . 19
3.3.2 INPUT . . . 19
3.3.3 OUTPUT . . . 20
3.4 GENERAL MATHEMATICS . . . 20
3.4.1 ABS . . . 20
3.4.2 ADD . . . 20
3.4.3 IMAG . . . 20
3.4.4 LINSCL . . . 21
3.4.5 REAL . . . 21
3.4.6 SUB . . . 21
3.5 VISUALIZATION . . . 21
3.5.1 DISPL . . . 21
3.5.2 GPLOT . . . 21
3.5.3 GPLOT2D . . . 22
3.6 TRANSFORMATION . . . 22
3.6.1 FT, FTnD . . . 22
3.6.2 PISHFT, PISHFTnD . . . 22
3.7 FILTERING . . . 23
3.7.1 RMDC . . . 23
3.7.2 WINDOW1D . . . 23
3.8 LINEAR PREDICTION . . . 24
3.8.1 HSVD . . . 24
3.8.2 LPSVD . . . 24
3.8.3 SYNFID . . . 24
3.8.4 VPNLS . . . 25
3.8.5 XTRP . . . 25
3.9 CONSTRAINED RECONSTRUCTION . . . 26
3.9.1 GSLIM . . . 26
3.9.2 RIGR . . . 26
3.9.3 SLIM . . . 26
3.10 MISCELLANEOUS . . . 27
3.10.1 HISTORY . . . 27
3.10.2 MAN . . . 27

4 V PROGRAMMING **29**
4.1 Directory Structure . . . 29
4.2 Contents . . . 29
4.3 A Recipe for the Development of New V Functions . . . 30
4.4 Library Function Development . . . 30
4.5 Calling Interface Development . . . 31
4.6 Documentation . . . 35
4.7 Modification of the V Source . . . 35

4.8 V Data Structure and Register Management 36

Chapter 1

PRELIMINARIES

1.1 Overview

Despite the recent surge of ideas for processing magnetic resonance data, very little effort has been made to make them accessible to the general MR community. Very often, using an algorithm to process data requires developing programs from scratch based on the literature or, at best, tailoring programs in their original form obtained from the author to apply them to specific problems. This process is made especially difficult by the multitude of operating systems and hardware used by researchers and the relative complexity of the newer reconstruction techniques. All too frequently, it seems, good ideas are lost in the literature because they are not readily applied.

V is our attempt to provide a general-purpose software system for magnetic resonance imaging and spectral reconstruction, processing, and analysis. It is intended not to be a comprehensive package including every known algorithm, but rather as a basic package including some of the popular reconstruction tools. Moreover, it is largely aimed at providing a means for distribution of functions among researchers in the MR community. V has been designed so that software development does not require an immense knowledge of the internal operation of the program. Rather, the programmer can develop individual functions indepently of V. After validation and testing, these functions can be *incorporated* into the V program with minimum modification.

1.2 Obtaining V

At the time of this publication, V is available via anonymous ftp to internet address v.brl.uiuc.edu (128.174.221.48). To obtain V from this location, use the ftp login name anonymous and your e-mail address as the password. A library of source code, manuals, and other developmental information will be maintained at this site in the /pub/V directory. Researchers are also encouraged to use this address to deposit any useful algorithms developed for V for easy release to other research groups.

For additional information on how to obtain V, please address correspondence to

Zhi-Pei Liang, Ph. D.
ATTN: V
4211 Beckman Institute
405 N. Mathews Avenue
University of Illinois at Urbana-Champaign M/C 251
Urbana, IL 61801

Questions regarding V should be addressed to V@mrel.brl.uiuc.edu. This address should also be used for subscriptions to a *very periodic* mailing list for information about new V releases, availability of V code developed by other researchers, and to report bugs. For up-to-date information on V release information, important updates, or other information, type finger v@mrel.brl.uiuc.edu on your host machine.

1.3 Installing V

For easy installation of V, the user is encouraged to download a version of V that includes binaries that have been compiled for a specific platform. At the time of this writing, there exist pre-compiled versions of V for SUN, SGI, HP, and IBM workstations. If a version in not available for a particular operating system or if the user is interested in tailoring the V source for his or her individual needs, then V can be rebuilt from scratch. Under normal circumstances, the installation should be performed by the system administrator.

After obtaining the desired version of the V source (see Section 1.2: Obtaining V), the following steps should be executed to install V. The distribution file should be in the directory *above* (the parent directory) of where the new V directory should be created[1].

1. Uncompress and untar[2] V by executing zcat V-1.0.tar.Z | tar xvf - This will take few moments to create the required directories and files.

2. If your distribution of V includes executable code, then you may skip to step 5. Otherwise, it is necessary to compile the V code.

3. At the shell prompt, typing cd V/src should place you in the V source directory. This directory and its children contains all of the V programs.

4. At the time of this writing, V has been compiled and tested on SUN, SGI, HP, and IBM workstations– makefiles for each platform should be located in the V/src directory. Since some V routines rely on Fortran functions, it is necessary to ensure that the correct path and the appropriate Fortran libraries are indicated in the F_LIBS line in the appropriate makefile for your system. After making any necessary modifications to the appropriate makefile, typing make -f MAKE.PLATFORM should begin the compilation process. This might take a few minutes.

5. The V executable code should now be in the V/bin directory. To complete the installation, one environment variable needs to be set, V_HOME. This can be done by adding the line setenv V_HOME path to your shell initialization script (eg .cshrc)– for example setenv V_HOME /usr/local/V. *Without this environment variable, V may not be able to locate manual pages and other important files.* Please consult your system documentation or contact your system administrator for additional information on defining environment variables. For direct access to V, the directory V_HOME/bin should be added to the working path.

1.4 Contents

This manual is organized into four chapters:

Chapter 2 describes the V interface. The basic data manipulation routines, input/output functions, and graphical display utilities are outlined. The chapter is intended to introduce the reader to V and to bring him/her up to speed as rapidly as possible.

Chapter 3 discusses the functions that are included in the basic V distribution. Generally, these routines fall into seven classes: register manipulation, transformation, linear prediction estimation, RIGR[3], SLIM[4] and

[1] In developing V, the programmers have attempted to make the code as portable as possible. However, some modifications might be necessary to the makefile (**V/src/MAKE.PLATFORM**) in order to make V compile properly.

[2] Some systems do not include the UNIX utility **zcat**. In this case, the user should uncompress the tar file by executing **uncompress V-1.0.tar.Z** and untarring the resulting file by executing **tar xvf V-1.0.tar**.

[3] Reduced-encoding Imaging by Generalized series Reconstruction.

[4] Spectral Localization by Imaging.

GSLIM[5], input/output, and math functions. This chapter should also serve as a reference once the reader is using V.

Chapter 4 , entitled *V Programming*, is included so that other scientists will be encouraged to incorporate their algorithms in V. The chapter deals with basic programming issues when developing code for V. The programmer, after browsing through this chapter, should be able to rapidly build new functions into V.

[5]Generalized SLIM.

Chapter 2

THE V INTERFACE

In general, MR data sets are multidimensional. Moreover, the newest reconstruction algorithms in imaging and spectroscopy use information from more than one experiment. As a result, any processing tool requires a convenient means for manipulating large, multidimentional data, whether they be real- or complex-valued. After reading this chapter, the user should be able to (1) read in and write out data, (2) reorder, extract, mirror, or otherwise prepare multidimensional data for processing, (3) manipulate data from separate experiments, and (4) display images. Apart from these basic instructions, the best way to learn V is, of course, by trying a few simple examples. Manual pages exist for on-line information about V functions. These pages can be accessed by typing -man command at the V prompt. For general information about V or how to obtain V, use -man without any arguments.

The V interface is easy to learn. All functions have the same format,

```
-command_name required_parameters optional_parameters
```

The command name will depend upon the type of processing that the user would like to execute. Associated with each command is a set of required parameters, variables that define the processing that will be done by the function. After these, many functions have a family of optional parameters which may or may not be used, depending on the type of operation desired by the user. Perhaps the best way to illustrate the command-line language is with an example. The -rigr command, described in the next chapter, requires two parameters and has three optional paramters:

```
-rigr 0 32 reg:0.1 filter:y xtrp:n pc:0
```

The last three arguments, reg:, filter:, and xtrp: are not required since there exist default values for these parameters. All parameters are separated by one or more spaces (or tabs). The order of the optional parameters is unimportant, but the required parameters must fall in a prescribed order. All optional arguments follow the convention variable:value, so that each denotes a variable that is to be defined by the given value within the function called by the user, in this example, the -rigr command. Though the optional paramters may fall in any order, there should be no space following the colon and the value, so that reg:0.1 is a valid optional parameter, but reg: 0.1 is not.

Access to data in V is via the register *stack*, a convention adopted from the popular multidimensional processing package viewit, written by Clinton Potter of the National Center of Supercomputing Applications[1].

[1] Viewit can be be obtained via anonymous ftp to **ftp.ncsa.uiuc.edu** (141.142.20.50). For additional information, the reader should contact *Clinton Potter, NCSA, ATTN: viewit, Beckman Institute, 405 N. Mathews Avenue, Urbana, IL 61801.* (email: cpotter@ncsa.uiuc.edu)

2.1 Register Stack and File I/O

The basic data processing backbone in V is the register. All data, whether read from a file or created synthetically, is stored in registers. Each register contains data from a series of experiments, from a single experiment, or specific data taken from one experiment. All registers are created dynamically, as they become necessary for processing. All are stored in a stack, the ubiquitous last-in first-out (LIFO) data structure. In this way, V is much like a reverse-polish notation calculator for multidimensional data. All processing is done on the registers at the top of the stack. The topmost register, referred to throughout as the *working register*, will usually contain the results of the operation.

Three functions are essential for basic manipulation of the register stack. First, the command -show_reg shows the contents of the register stack– how many registers exist, what type of data each holds, the dimensions of each register, etc. To create a new working register which contains the same data as the current working register, the -push command is used. After this operation, the top two registers in the stack will have identical contents, so that this operation is useful for making multiple copies of data before processing. To reverse the effect of a -push command, the -pop command should be called. This function has the effect of discarding the working register (and its contents!) and making the next register in the stack the working register.

At the time of this writing, V is capable of reading two file formats- NCSA Scientific Data Set HDF (Hierarchical Data Format) and ASCII. The default file format used by V is HDF[2], a format particularly well-suited to storing multidimensional data sets, relevant dimension information, and even the flexibility to include annotations, titles, and other useful, human-readable information. To load the working register with data from an HDF file, the command -i, for *input* is used. This will read the data (assumed complex-valued) into the working register and set the appropriate dimension information.

Data can also be read from a plain ASCII (text) file. Since such a file does not normally include necessary dimension information, the working register must first be set to the proper dimensions before the data can be read. To create a new working register (and destroy the contents of the old one), the *create* command is used. The calling syntax of this command is straightforward:

-create REAL/COMPLEX ndim dim1 [dim2 dim3 ...],

where REAL (or COMPLEX) is a required parameter indicating the type of data to be read, ndim is the dimensionality of the data (i.e., 2D, 3D, etc.), and remaining parameters dim1, etc. define the size of each dimension. For example, to prepare a register to hold a $256 \times 64 \times 4$ complex-valued set of data from an ASCII file, the appropriate create command is -create COMPLEX 3 256 64 4. The ASCII file is then read by issuing the command

-i *filename* format:ASCII,

where format: is an optional parameter indicating that an ASCII file (instead of an HDF file) is to be read.

Data from the working register is written to a file in much the same way that it is read from a file, only the *output* command, -o is used instead of -i. The calling syntax of the former is analogous to that of the latter:

-o *filename* [format:*format*].

The default file format is HDF, but the optional parameter format: can be set to ASCII for raw (text) files or PGM for portable gray map (PGM) files. The PGM format is only valid for real, two dimensional data (i.e., images).

2.2 Data Manipulation

The intrinsic functions for manipulating data in V fall into one of two categories. The first, *stack manipulation functions*, operate on the register stack and not on the contents of the individual registers. The second, *register*

[2]HDF was developed by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

manipulation functions operate on the data contained within an individual register (usually the working register). An good understanding of both of these types of functions is absolutely necessary to prepare data for processing by the reconstruction functions.

2.2.1 Stack Manipulation Functions

Aside from -push and -pop, there are only two important stack manipulation functions. First, the function -xchg (*exchange*) interchanges the top two registers in the stack, so that the working register becomes the second register in the stack and vice versa. Naturally, there must be at least two registers in the stack before this function can be called. Second, the -swap (*swap*) function is used to exchange the positions of multiple registers in the stack. The number of parameters required by this function is equal to the number of registers that exist. Each parameter is the number of an existing register (i.e., 1, 2, etc.) and their order is the desired order of the new stack. For example, if four registers exist (1, 2, 3, and 4) and the user wishes to swap the stack positions of the first two and the last two registers, the correct calling syntax would be -swap 2 1 4 3.

2.2.2 Register Manipulation Functions

More often than not, several dimensions in a multidimensional data set are processed separately from others. In a multislice imaging experiment, for example, each 2D slice in the 3D data set must undergo Fourier transformation. The register manipulation functions provide the means for redimensioning, reordering, extracting, or otherwise manipulating data within a particular register.

The simplest register manipulation functions define the type of data that a register contains. The -settype command changes the type of data that is stored in a register. Both real and complex data are stored in the same manner, so it is the responsibility of the user to ensure that the type is properly set for data before further processing. Many processing functions require data of one type or the other, and several will give different results depending on whether the data in a register a real or complex. The syntax of the command to set the type of a function is

-settype REAL/COMPLEX [register],

where REAL or COMPLEX defines the type to set and register is an optional parameter indicating the register whose type is to be changed. If this parameter is not given, then the command operates on the working register by default. The -split and -merge operate on complex and real registers, respectively. When the working register holds complex-valued data, the -split command has the effect of separating the real and imaginary parts of the data and placing them in register 1 and register 2 respectively. An analogous command undoes the effect of this operation– the -merge command takes the top two registers in the stack, which must be real and have the same dimensions, and combines them so that they become the real and imaginary parts of the working register.

By default, V functions operate on the leading dimension of the working register. Thus, if another dimension in the data must be processed instead, then it becomes necessary to *reorder* the data so that the desired dimension becomes the leading dimension. The reorder command is called as

-reorder dim1 dim2 [dim3 ...],

where the dim parameters indicate the desired order of the dimensions (0 is the first dimension). The reordering is always performed on the working register. For example, to make the second dimension the first and vice versa in the 2D data set present in the working register, the proper syntax is -reorder 1 0. The -reorder command requires the same number of parameters as dimensions present in the working register.

Instead of reordering dimensions, it is also possible to extract only the data required for processing. For example, in a multislice experiment, often only one slice of data is used to create an image. In some spectroscopy experiments, only a certain segment of data are valid. For these and similar situations in which only a subset of the data within a register is required, the -chunk command can be used to extract the desired data; the syntax is

-chunk off1 [off2 ...] np1 [np2 ...],

where the number of parameters required is equal to twice the dimensionality of the working register, *ndim*. The first *ndim* paramters indicate the offset into each dimension where the desired data to extract begin, and the last *ndim* parameters define the number of points to chunk out of each dimension. For example to extract the central 64×64 portion of the seond slice in a 256×4 data volume, the proper syntax for -chunk is -chunk 95 95 1 64 64 1.

Two less frequently used functions, -redim and -mirror are worth mentioning at this point for completeness. First, the -redim function is intended to provide the user a direct means of changing the dimension information. Data are stored in memory independently of the dimension information as a one-dimensional array. This dimension information can be changed, as long the redimensioning does not alter the overall cardinality of the data. For example, a 4×4 2D register can be redimensioned as a 16-point 1D register or a $2 \times 2 \times 2 \times 2$ 4D register. To redimension the working register, the -redim command is called as follows:

-sf redim ndim dim1 [dim2 ...],

where ndim is the dimensionality (number of dimensions of the data) of the redimensioned register. The remaining parameters (of which there must be ndim) define the sizes of each of the dimensions. To redimension a 2D 256×256 data set into a 3D $2 \times 2 \times 4$ one, -redim 3 2 2 4 is the proper calling syntax. A second function that mind find some use is the -mirror function, whose operation is to take a multidimensional set of data and flip a given dimension about its center. For example a 2D image is flipped about the vertical (horizontal) line going through its middle by issuing the command -mirror 0 (-mirror 1).

2.3 Graphical Display

V does not include its own image display utility for visualization. To display images, the program writes out a temporary PGM format file for display using xv[3] Thus, *to use the image display function in V,* xv *must be configured and be in the current path.* Data to be displayed must be in image format, i.e., must be real-valued and two-dimensional. For proper display, the image pixels should take values in the range 0 to 255. To take a real 2D set of data and linearly scale it so that all pixel values lie within this range, the command -linscl 0 255 can be issued. For 1D data sets, the function -gplot can be used to create a 2D plot from the 1D data. This command is called as

-gplot *width height*,

where *width* is the width and *height* is the height of the plot window in pixels. This function is particularly useful for plotting 1D spectra.

[3] xv, by John Bradley, can be obtained from many sites via anonymous ftp.

Chapter 3

V FUNCTIONS

The following sections should serve as a reference guide. Each section consists of the manual pages for functions that fall under the given categories. This information is also available directly when using V, by typing -man *commandname*. For a list of valid commands, the user can type -help[1].

[1]For a concise description of a particular command, the command **-help commandname** can be executed instead of **-man commandname**

3.1 REGISTER MANIPULATION

3.1.1 CENTER

NAME
center - data centering

SYNOPSIS
center

DESCRIPTION
-center translates each dimension of the data in the working register by one-half the number of elements in each dimension. If a dimension is odd in size, then the amount of translation is scaled down to the nearest integer value.

EXAMPLES
1. To center each dimension of the data in the working register, V> -center

RESTRICTIONS
A working register must be present for -center.

SEE ALSO
reorder, mirror, chunk

FILES
V_center.c

3.1.2 CHUNK

NAME
chunk - extract a subset of data from the working register

SYNOPSIS
chunk off0 [off1...] np0 [np1...]

DESCRIPTION
The -chunk command "chunks out" a segment of data. The offset into each dimension is given by the off0, off1, ... parameters (one for each dimension), and the number of points to extract in each dimension is given by the np0, np1, ... parameters (one for each dimension).

EXAMPLES
1. To extract the 64x64 central portion of the second slice in a 256x256x16 3D data set, the appropriate syntax is V> -chunk 95 95 1 64 64 1

SEE ALSO
reorder, mirror, redim

FILES
V_chunk.c

3.1.3 GROUP

NAME
group - create a 2D mosaic from a 3D data set

SYNOPSIS
group ncol

DESCRIPTION
The -group command uses the slices from a 3D data set to form a 2D data set by arranging the slices as a two-dimensional array. The number of columns in the resulting mosaic is given by the parameter ncol. This can be used, for example, to group a series of images for comparison.

EXAMPLES
1. To group a 256x256x16 3D data set into a 4x4 mosiac of 2D data sets, V> -group 4

RESTRICTIONS
The parameter ncol must take on values less than the number of slices in the 3D data set.

SEE ALSO
displ, reorder, chunk

FILES
V_group.c

3.1.4 LABEL

NAME
label - label the given register with ASCII text

SYNOPSIS
label register label

DESCRIPTION
The -label command labels the register given by the register number register (1, 2, ...) with the ASCII text given by label.

EXAMPLES
1. To label the second register with the label "Phophorus Image", V> -label 2 Phosphorus Image

RESTRICTIONS
The parameter register must correspond to an existing register.

SEE ALSO
settype

FILES
V_label.c

3.1.5 MERGE

NAME
merge - interlace two real data sets to form a complex data set

SYNOPSIS
merge

DESCRIPTION
The -merge command takes the data from the top two registers in the stack and uses them to create a complex-valued data set. The real part is taken from the working register, and the imaginary part is taken from the second register in the stack. The result is placed in the working register.

EXAMPLES
1. To concantenate the data in the first two registers to form the real and imaginary parts of a new data set to be placed in the working register, V> -merge

RESTRICTIONS
The -merge command requires that the top two registers in the stack be REAL and have the same dimensions.

SEE ALSO
split

FILES
V_merge.c

3.1.6 MIRROR

NAME
mirror - reflect the working register about the given dimension

SYNOPSIS
mirror dim

DESCRIPTION
The -mirror command reflects the data in the working register about the center of the given dimension dim, which may take on values 0,1,2 etc., depending on the dimension which is to be mirrored.

EXAMPLES
1. To flip an image about its vertical axis, V> -mirror 0

2. To flip an image about its horizontal axis, V> -mirror 1

RESTRICTIONS
Mirror must be called with one argument, whose value must correspond to an existing dimension in the working register.

SEE ALSO
reorder, redim, chunk

FILES
V_mirror.c

3.1.7 POP

NAME
pop - pop the working register

SYNOPSIS
pop

DESCRIPTION
The -pop command performs a "pop" operation on the register stack. The topmost register and its contents are discarded, and the remaining registers are copied from the bottom up to the preceding registers in the stack. Hence, a stack containing N registers before the call to -pop will contain N-1 registers after the call.

EXAMPLES
1. To pop the working register, V> -pop

RESTRICTIONS
The -pop command must be performed on a non-empty register stack.

SEE ALSO
push, xchg, swap

FILES
V_pop.c

3.1.8 PUSH

NAME
push - push the working register

SYNOPSIS
push

DESCRIPTION
The -push command performs a "push" operation on the register stack. The current registers are moved down the stack, and a copy of the old working register becomes the topmost register in the stack. Hence, a stack containing N registers before the call to -push will contain N+1 registers after the call, and registers 1 and 2 will hold identical copies of the same data.

EXAMPLES
1. To push the working register, V> -push

RESTRICTIONS
The -push operation must be performed on a non-empty register stack.

SEE ALSO
pop, xchg, swap

FILES
V_push.c

3.1.9 REDIM

NAME
redim - change the dimension information of the working register

SYNOPSIS
redim ndim dim0 [dim1 ...]

DESCRIPTION
The -redim redimensions the data in the working register. The number of dimensions is given by ndim, and the remaining parameters define the size of each dimension.

EXAMPLES
1. To transform a 4x4 2D data set into a 2x2x2x2 4D data set, V> -redim 4 2 2 2 2

RESTRICTIONS
1. The number of parameters required by -redim is equal to ndim + 1.

2. The parameters must preserve the cardinality of the data in the register.

SEE ALSO
reorder, mirror, chunk

FILES
V_redim.c

3.1.10 REORDER

NAME
reorder - rearrange array dimensions

SYNOPSIS
reorder dim0 [dim1 ...]

DESCRIPTION
The effect of -reorder is to internally rearrange data within the working register. The new order of the dimensions is given by the order in which the parameters occur in the call to reorder.

EXAMPLES
1. To tranpose the rows and columns of a two-dimensional data set, V> -reorder 1 0

RESTRICTIONS
The number of arguments in the call to -reorder must be the same as the number of dimensions of the data in the working register.

SEE ALSO
mirror, redim, chunk

FILES
V_reorder.c

3.1.11 SETTYPE

NAME
settype - change the type of the data stored in a register

SYNOPSIS
settype COMPLEX | REAL [register]

DESCRIPTION
The -settype command acts by default on the working register, unless the optional parameter register (1, 2, etc) is given. It has the effect of setting the data_type flag, not of converting from real to complex or vice versa. Hence calling -settype COMPLEX when the working register is real will half the size of the leading dimension. Similarly, calling -settype REAL when the working register is complex will double the size of the leading dimension.

To convert from complex to real data, the -real and -imag commands or the -split command should be used. To convert from complex to real, the -merge command should be used.

1. To change the type of the working register to real, V> -settype REAL

2. To change the type of the working register to complex, V> -settype COMPLEX

RESTRICTIONS
The optional parameter register , if given, must correspond to an existing register.

SEE ALSO
real, imag, redim, label

FILES
V_settype.c

3.1.12 SHOW_REG

NAME
show_reg - show the contents of the register stack

SYNOPSIS
show_reg

DESCRIPTION
The -show_reg command displays the contents of the register stack, including the data type, dimensions, and label for each register.

EXAMPLES
1. To display the current register stack, V -show_reg

SEE ALSO
displ

FILES
V_show_reg.c

3.1.13 SPLIT

NAME

split - de-interlace a complex data set

SYNOPSIS
split

DESCRIPTION
The -split command takes the data from a complex working register and separates it into real and imaginary parts. The real part is left in the working register, and the imaginary part is placed in the second register.

EXAMPLES
1. To return the real and imaginary parts of a complex in the first and second registers, respectively, V> -split

RESTRICTIONS
The -split command requires that the working register be COMPLEX.

SEE ALSO
merge, xchg

FILES
V_split.c

3.1.14 SWAP

NAME
swap - arrange the registers in the stack in a particular order

SYNOPSIS
swap reg1 [reg2 ...]

DESCRIPTION
The -swap command takes the registers already present in the stack and places them in the order given by the user.

EXAMPLES
1. To reverse the order of the registers in a stack which contains 4 registers, V> -swap 4 3 2 1

RESTRICTIONS
The number of arguments in the call to -swap must be the same as the number of registers present in the stack.

SEE ALSO
pop, push, xchg

FILES
V_swap.c

3.1.15 XCHG

NAME
xchg - exchange the top two elements in the register stack

SYNOPSIS
xchg

DESCRIPTION
The -xchg takes the top two registers in the stack and exchanges them, so that the new working register will contain the contents of the second register in the stack and vice versa.

EXAMPLES
1. To exchange the stack positions of the first two registers, V> -xchg

RESTRICTIONS
The stack must have at least two registers for the -xchg operation to be valid.

SEE ALSO
pop, push, swap

FILES
V_xchg.c

3.2 DATA SYNTHESIS

3.2.1 RANDOM

NAME
random - random number/noise generation

SYNOPSIS
random mean stdev noise_type

DESCRIPTION
The -random command fills the working register with either Gaussian- or uniformly-distributed random numbers with the given mean and standard deviation. The register must be dimensioned prior to the call to -random. The resulting data is placed in the working register.

NOTE: Random data uniformly distributed on [A,B] has mean=0.5*(A+B) and stdev=(B-A)

mean - mean of random data

stdev - standard deviation of random data (must be nonnegative)

noise_type UNIFORM or GAUSSIAN

EXAMPLES:
1. To generate noise uniformly distributed on [-5,3] V> -random -1 8 uniform

2. To generate Gaussian noise with mean 10 and standard deviation 5, V> -random 10 5 gaussian

RESTRICTIONS
The -random command requires a working register.

FILES
V_random, zufall.f

3.2.2 UNIFFILL

NAME
uniffill - uniform fill

SYNOPSIS
uniffill constant

DESCRIPTION
The -uniffill command fills the working register with the given floating point value constant. The register must be dimensioned prior to the call to uniffill. The result is placed in the working register.

EXAMPLES
1. To generate a REAL vector of 512 points with value 255, V> -create REAL 1 512 -uniffill 255

2. To generate a 256x256 COMPLEX array with real and imaginary parts equal to 1, V> -create COMPLEX 2 256 256 -uniffill 1

3. To generate a 4x4x4 COMPLEX array filled with the complex value 1-1i, V> -create REAL 3 4 4 4 -uniffill -1 -push -uniffill 1 -merge

RESTRICTIONS
The -uniffill command requires a pre-dimensioned working register.

SEE ALSO
zeropad

FILES
V_uniffill.c

3.2.3 ZEROPAD

NAME
zeropad - pad data with zeros

SYNOPSIS
zeropad np0 [np1 ...]

DESCRIPTION
The -zeropad command pads working register with zeros. This can be used, for example, in conjunction with the FFT routines to interpolate data. The register must already be dimensioned prior to the call to -zeropad. The result is placed in the working register.

EXAMPLES
1. To pad a 2D data set to 256x256 before (e.g.) Fourier transformation, V> -zeropad 256 256

RESTRICTIONS
The -zeropad command requires that a working register be present.

SEE ALSO
uniffill

FILES
V_zeropad.c

3.3 INPUT/OUTPUT

3.3.1 DUMP2TERM, DUMP2FILE

NAME
dump2term - dump the contents of the working register to the terminal

dump2file - dump the contents of the working register to a file

SYNOPSIS
dump2term [DATA|PARAMETER]

dump2file file [DATA|PARAMETER]

DESCRIPTION
-dump2term displays the contents of the working register. By default, the data in the register are printed on the terminal in vector form.

The optional parameter, [DATA|PARAMETER] allows the user to set the printing format. If the working register contains spectral parameters, by using PARAMETER the formatted output will be the data in a table with frequency, linewidth, peak area and phase.

-dump2file prints the contents of the working register to a file, where the filename is given by the file parameter.

file - output file name

EXAMPLES
1. To dump data in the working register to the console for inspection V> -dump2term

2. To dump parameters in the working register to a file named outfile V> -dump2file outfile parameter

RESTRICTIONS
When PARAMETER is used as the data format, the data in the working register must be a spectra parameter data set generated by lpsvd, hsvd or vpnls.

SEE ALSO
lpsvd, hsvd, vpnls, synfid

FILES
V_dump.c

3.3.2 INPUT

NAME
i - input data

SYNOPSIS
i filename [format:HDF|ASCII]

DESCRIPTION
The -i command reads data in from an HDF (default) or ASCII files. To read in data from ASCII files, the working register must be set to the dimensions and data type of the data to be read. This can be done using the -create command. For HDF files, no pre-dimensioning of the working register is necessary, but any data already in the working register will be overwritten.

For HDF files which were not written using V, the default data type is REAL. For complex data sets, it is necessary to use the -settype command to set the data type to COMPLEX after reading HDF data that has never been read by V. HDF files written using the -o command in V contain data type information and labels, so that -settype need only be used once on new data sets.

EXAMPLES
1. To read in 256x256 2D COMPLEX data from a file named infile.ascii, V> -create complex 2 256 256 -i infile format:ASCII

2. To read in a real HDF file named real.hdf, V> -i real.hdf

3. To read in a complex HDF file named complex.hdf which was written by V, V> -i complex.hdf

4. To read in a complex HDF file named complex.hdf which

was not written by V, V> -i complex.hdf -settype COMPLEX

RESTRICTIONS
ASCII input requires a pre-dimensioned working register.

SEE ALSO
o, dump2term, dump2file, create

FILES
V_i.c

3.3.3 OUTPUT

NAME
o - output data

SYNOPSIS
o filename [format:HDF | PBM | ASCII]

DESCRIPTION
The -o command writes data in the working register to a file filename in HDF (default), PGM (Portable Gray Map), or ASCII format.

The HDF file format will automatically store the data dimensions, type, and labels, so that the next time the data is read into V, there is no need to set the data type, dimensions, or labels.

RESTRICTIONS
PGM format is only valid for REAL, 2D working registers.

EXAMPLES
1. To save the data in the working register to an HDF file named outfile.hdf, V> -o outfile.hdf

2. To save the data in an ASCII file named outfile.ascii, V> -o outfile.ascii format:ASCII

3. To save an image in PGM format, the REAL, 2D data in the working register can be processed as V> -linscl 0 255 -o outfile.pgm format:PGM

SEE ALSO
i, dump2term, dump2file

FILES
V_o.c

3.4 GENERAL MATHEMATICS

3.4.1 ABS

NAME
abs - absolute value or modulus operation

SYNOPSIS
abs

DESCRIPTION
-abs operates on the working register. If the data is complex, abs will perform a magnitude (modulus) operation element-by-element on the given data. If the data is real, on the other hand, the effect of -abs is to take the absolute value of each element in the data set.

EXAMPLES
1. To take the absolute value of each element in a REAL working register, V> -abs

2. To take the modulus of each element in a COMPLEX working register, V> -abs

FILES
V_abs.c

3.4.2 ADD

NAME
add - piecewise add elements in first two registers

SYNOPSIS
add

DESCRIPTION
-add operates on the first two working registers. Each element in the first register is added to the corresponding element in the second register, and the sum is placed in the working register.

EXAMPLES
1. To add the contents of the first two registers, V> -add

RESTRICTIONS
The -add command requires that the dimensions of the first and second registers be identical.

SEE ALSO
sub

FILES
V_add.c

3.4.3 IMAG

NAME
imag - return the imaginary part of complex data

SYNOPSIS
imag

DESCRIPTION
The -imag command takes the imaginary part of the data in the working register, element-by-element. The result is placed in the working register, and will have the same dimensions as the complex data set from which it was derived.

EXAMPLES
1. To take the imaginary part of the data in the working register, V> -imag

RESTRICTIONS
The -imag command requires a complex working register

SEE ALSO
real, split, merge

FILES
V_imag.c

3.4.4 LINSCL

NAME
linscl - linearly scale data

SYNOPSIS
linscl min max

DESCRIPTION
The -linscl command scales the data in the working register so each data point falls between the values given by min and max. This function is useful, for example, for scaling a real 2D data set into 8-bit grayscale data.

EXAMPLES
1. To scale the data in the working register into 8-bit grayscale data (working register is assumed REAL), V> -linscl 0 255

RESTRICTIONS
The -linscl command operates only on real data

FILES
V_linscl.c

3.4.5 REAL

NAME
real - return the real part of complex data

SYNOPSIS
real

DESCRIPTION
The -real command takes the real part of the data in the working register, on a point-by-point basis. The result is placed is the working register, and will have the same dimensions as the complex data set from which it was derived.

EXAMPLES
1. To take the real part of the data in the working register, V> -real

RESTRICTIONS
The -real command requires a complex working register

SEE ALSO
imag, split, merge

FILES
V_real.c

3.4.6 SUB

NAME
sub - piecewise subtract elements in first two registers

SYNOPSIS
sub

DESCRIPTION
-sub operates on the first two working registers. Each element in the first register is subtracted from the corresponding element in the second register, and the difference is placed in the working register.

EXAMPLES
1. To subtract the contents of Register 1 from those of Register 2, V> -sub

RESTRICTIONS
The -sub command requires that the dimensions of the first and second registers be identical.

SEE ALSO
add

FILES
V_sub.c

3.5 VISUALIZATION

3.5.1 DISPL

NAME
displ - image display using xv

SYNOPSIS
displ

DESCRIPTION
The -displ command displays greyscale images in X windows environments. The data set should be scaled to have only values lying between 0 and 255. This can be done using the -linscl command. For systems which are not running X windows, or do not have xv installed, visualization must be performed outside of the V environment.

EXAMPLES
1. To display data in a REAL, 2D working register V> -linscl 0 255 -displ

RESTRICTIONS
1. -displ works only on X window systems where xv has been installed and is in the current path.

2. Data to be displayed must be REAL and two dimensional. One dimensional data can be plotted using the -gplot. command.

SEE ALSO
gplot, gplot2d, linscl, show_reg

FILES
V_displ.c

3.5.2 GPLOT

NAME
gplot - generate plot

SYNOPSIS
gplot wx wy

DESCRIPTION
The command -gplot generates 2D images from 1D subsets of the data in the working register. This can be used, for example, to generate x-y plots of 1D data, e.g. spectra, for visual inspection. The command requires two parameters:

wx - horizontal plot window size

wy - vertical plot window size

Data in the working register may have up to four dimensions. The data for all plots is taken along the first dimension. In each case, the resulting plots are arranged as follows:

1D A single plot is generated, where the graph of the data in the working register fills the entire plot window. The working register after the call will hold 2D data.

2D A mosaic image of plots is generated. The number of plots is given by the second dimension. The individual plots are arranged side by side horizontally in the resulting image. The working register after the call will hold 2D data.

3D A mosiac image of plots is generated. The number of plots is given by the product of the first and second dimensions. The individual plots are arranged horizonally for the data along the second dimension and vertically for the data along the third dimension. The working register after the call will hold 2D data.

4D A set of mosaic images of plots are generated. Each slice in the mosaic corresponds to one 3D "slice" of data. The individual plots are arranged horizonally for the data along the second dimension, vertically for the data along the third dimension. The working register after the call will hold 3D data. The -group or -chunk can be used to display the resulting plots.

For any number of dimensions, the wx and wy parameters

specify the size of each plot, not of the entire mosaic of plots.

EXAMPLES

1. To generate a plot for a 1D spectrum and display it, V> -i sample/fid1d.df -pishft -ft fwrd -abs -gplot 512 512 -displ

2. To generate an image consisting of plots of each spectrum in a 4D data set and display this image, V> -i sample/fid3d.df -pishft -ft fwrd -abs -group 4 -gplot 512 512 -displ

RESTRICTIONS

-gplot requires that data in the working register be REAL.

SEE ALSO

gplot2d, displ

FILES

V_gplot.c

3.5.3 GPLOT2D

NAME

gplot2d - generate stack plot

SYNOPSIS

gplot2d wx0 wy0 wx wy

DESCRIPTION

The command -gplot2d generates stack plots from 2D data sets in the working register. This can be used, for example, for visual comparison of peak locations and amplitudes of a series of spectra. Each curve in the stacked plot is offset from its predecessor by wx0 pixels in the horizontal dimension and wy0 pixels in the vertical dimension. The canvas for the entire image has horizontal and vertical size given by the wx and wy parameters, respectively. In summary, the required parameters are:

wx0 - horixontal offset between curves

wy0 - vertical offset between curves

wx - horizonal plot window size

wy - vertical plot window size

EXAMPLES

1. To generate and display a 512x512 stack plot for a series of spectra where the horizontal and vertical separations for each spectrum are 5 and 10, respectively, V> -gplot2d 10 10 512 512 -displ

RESTRICTIONS

-gplot2d requires that data in the working register be REAL and 2D.

SEE ALSO

gplot, displ

FILES

V_gplot2d.c

3.6 TRANSFORMATION

3.6.1 FT, FTnD

NAME

ft, ft1d, ft2d, ft3d, ft4d - Discrete Fourier Transform

SYNOPSIS

fft [FWRD|BACK] [tdim]

ft [FWRD|BACK] [tdim]

ft1d [FWRD|BACK]

ft2d [FWRD|BACK]

ft3d [FWRD|BACK]

ft4d [FWRD|BACK]

DESCRIPTION

These functions compute the Fourier transforms (DFT's) of the data in the working register. When the transform lengths are composites of 2 or products of prime factors, then the DFT's are computed using fast Fourier transform (FFT) algorithms. These functions operate on REAL or COMPLEX data, but in either case, the output data type is always COMPLEX. The default transform is forward, but the optional parameter FWRD/BACK can be used to manually set the transform direction.

-ft By default, performs M-D Fourier transform, where M is the dimension of the data set in the working register. The optional parameter tdim can be used to force a tdim-D FFT along only the leading tdim dimensions.

-ft1d 1D Fourier transform of data set along the leading dimension for each remaining dimension

-ft2d 2D Fourier transform of data set along first two dimensions for each remaining dimension

-ft3d 3D Fourier transform of data set along first three dimensions for each remaining dimension

-ft4d 4D Fourier transform of data set along first four dimensions for each remaining dimension

EXAMPLES

1. To compute the multidimension forward Fourier transform, V> -fft

2. To perform inverse 1D Fourier transformation along the leading dimension, V> -ft1d back

3. To perform 2D Fourier transform along the first two dimensions, V>-ft fwrd 2

RESTRICTIONS

The implied dimension of the Fourier transform operation using -fft, -ft, -ft1d, -ft2d, -ft3d, or -ft4d can not exceed the dimension of the working register.

SEE ALSO

center, pishft, pishft1d, pishft2d, pishft3d, pishft4d

FILES

V_pishft.c

3.6.2 PISHFT, PISHFTnD

NAME

pishft, pishft1d, pishft2d, pishft3d, pishft4d - PI shifting

SYNOPSIS

pishft

pishft1d

pishft2d

pishft3d

pishft4d

DESCRIPTION

These functions operate on COMPLEX working registers. They perform PI-shifting by multiplying the data set in the working register by appropriate powers of -1.

-pishftd M-D PI-shift, where M is the dimension of the data set

-pishft1d 1D PI-shift of data set along the leading dimension for each remaining dimension

-pishft2d 2D PI-shift of data set along first two dimensions for each remaining dimension

-pishft3d 3D PI-shift of data set along first three dimensions for each remaining dimension

-pishft4d 2D PI-shift of data set along first four dimensions for each remaining dimension

NOTE: The effect of V> -pishft2d -ft is identical to that of V> -ft -center

EXAMPLES
1. To perform 1D PI-shifting along the leading dimension of the working register, V> -pishft1d

RESTRICTIONS
The -pishft , -pishft1d , -pishft2d , -pishft3d , and -pishft4d commands require that the data in the working register be COMPLEX.

SEE ALSO
center, ft, ft1d, ft2d, ft3d, ft4d

FILES
V_pishft.c

3.7 FILTERING

3.7.1 RMDC

NAME
rmdc - remove DC offset

SYNOPSIS
rmdc np

DESCRIPTION
The -rmdc command uses the last np points of the (complex) data set in the working register to calculate an average value and subtracts this value from the entire data set. This is useful for removing the DC offset from spectroscopic data in which the final segment of the FID has decayed into the noise floor.

EXAMPLES
1. To subtract the the mean value from the data as computed from the final 32 points in each FID, V> -rmdc 32

RESTRICTIONS
The -rmdc command requires a COMPLEX working register.

FILES
V_rmdc.c

3.7.2 WINDOW1D

NAME
window1d - one dimensional windowing

SYNOPSIS
window1d pe_st pe_end window_type [slope: slope]

DESCRIPTION
The -window1d command multiplies the data in the working register with the chosen apodizing window, either hamming, hanning, bartlett, or ramp.

pe_st - index of first phase encoding

pe_end - index of final phase encoding

window_type - HAMMING, HANNING, BARTLETT, or RAMP

The one optional parameter is used only when window_type = RAMP:

slope - ramp function slope

When the working register contains multi-dimensional data, it is assumed that the data consists of multiple FID's along the first dimension. In this case, each FID is windowed independently.

EXAMPLES:
1. To multiply the data in the working register by a Hanning window, V> -window1d -16 48 HANNING

2. To multiply the data in the working register by a ramp function with slope 2.0, V> -window1d 0 32 RAMP slope:2.0

RESTRICTIONS
The number of data implied by the given first and final phase encoding indices must be equal to the number of points along the leading dimension of the working register.

FILES
V_window1d.c

3.8 LINEAR PREDICTION

3.8.1 HSVD

NAME
hsvd - state space linear prediction (LP) estimation

SYNOPSIS
hsvd np noff ncomp [maxlw: maxlw] [minlw: minlw]

DESCRIPTION
The -hsvd command uses the state-space method for LP estimation, known as HSVD in the NMR literature. This can be used to estimate NMR spectral parameters from measured FID (free-induction decay) data. A total of np points, beginning with an offset of noff points, are used to calculate the spectral linewidths, frequencies, and intensities from the data in the working register. The arguments for -hsvd are specified as follows:

np - number of samples (points) to use in estimation

noff - offset into the leading dimension of data to use

ncomp - number of spectral components to fit to data

Two optional parameters, maxlw and minlw allow the user to eliminate spectral peaks which are too wide or too narrow:

maxlw - discard peaks with linewidths greater than maxlw

minlw - discard peaks with linewidths less than minlw

When the working register contains multi-dimensional data, it is assumed that the data consists of multiple FID's, where the leading dimension is the FID time dimension, and the HSVD algorithm is performed on each one.

The HSVD parameters are returned in the working register.

EXAMPLES
1. To fit 7 components to the data in the working register using 32 points beginning with the second point in each FID, V> -hsvd 32 1 7

2. To fit 10 components to the data in the working register using the first 48 points in each FID, and to discard all peaks whose linewidths are less than 0.01 or greater than 0.9, V> -hsvd 48 0 10 0.01 0.9

RESTRICTIONS
1. -hsvd works only on COMPLEX data.

2. The number of points
np must be greater than or equal to twice the number of components, ncomp.

REFERENCES
[1] S. Y. Kung, K. S. Arun, and D. V. Bhaskar Rao, "State-space and singular-value decomposition-based approximation methods for the harmonic retrieval problem," J. Opt. Soc. Am., vol. 73, no. 12, pp. 1799-1811, December 1983.

SEE ALSO
lpsvd, synfid, vpnls, dump2term, dump2file

FILES
V_hsvd.c

3.8.2 LPSVD

NAME
lpsvd - Kumaresan-Tufts linear prediction (LP) estimation

SYNOPSIS
lpsvd np noff ncomp [maxlw: maxlw] [minlw: minlw]

DESCRIPTION
The -lpsvd uses the Kumaresan-Tufts method for LP estimation, known as LPSVD in the NMR literature. This can be used to estimate NMR spectral parameters from measured FID (free-induction decay) data. A total of np points, beginning with an offset of noff points, are used to calculate the spectral linewidths, frequencies, and intensities from the data in the working register. The arguments for .lpsvd are specfied as follows:

np - number of samples (points) to use in estimation

noff - offset into the leading dimension of data to use

ncomp - number of spectral components to fit to data

Two optional parameters, maxlw and minlw allow the user to eliminate spectral components that are too wide or too narrow:

maxlw - discard peaks with linewidths greater than maxlw

minlw - discard peaks with linewidths less than minlw

When the working register contains multi-dimensional data, it is assumed that the data consists of multiple FID's while the leading dimension is the FID time dimension, and the LPSVD algorithm is performed on each one.

The LPSVD parameters are returned in the working register.

EXAMPLES
1. To fit (up to) 16 components to the data in the working register using 32 data points (beginning with the second point in each FID) and an SVD truncation level of 10, V> -lpsvd 32 1 10

2. To fit (up to) 24 components to the data in the working register using the first 48 points in each FID using an SVD truncation level of 10 and to discard all peaks whose linewidths are less than 0.01 or greater than 0.9, V> -lpsvd 48 0 10 0.01 0.9

RESTRICTIONS
The -lpsvd command works only on COMPLEX data.

REFERENCES
[1] R. Kumaresan and D. W. Tufts, "Estimating the parameters of exponentially damped sinusoids and pole-zero modeling in noise," IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-30, no. 6, pp. 833-840, December 1982.
[2] H. Barkhuijsen et al., "Application of linear prediction and singular value decomposition (LPSVD) to determine NMR frequencies and intensities from the FID," Magnetic Resonance in Medicine, vol. 2, pp. 86-89, 1985.

SEE ALSO
hsvd, synfid, vpnls, dump2term, dump2file

FILES
V_lpsvd.c

3.8.3 SYNFID

NAME
synfid - Synthesis an FID signal from a spectra parameter data

SYNOPSIS
synfid np [noff]

DESCRIPTION
The -synfid command generates an FID of total np points using the spectral parameters in the working register. The optional parameter, noff allows the user to select the index of the first point (offset) of the output FID data:

np - number of points of the output FID data

noff - offset of the output FID data

When the working register contains multi-dimensional data, it is assumed that the data consists of an array of parameter sets, and the -synfid command is performed on each one and a multi-dimensional array of FID's will be generated.

The generated FIDs are returned in the working register.

EXAMPLES
1. To generate 256-point FID's from the parameters in the working register, where each FID begins with encoding index 10, V> -synfid 256 10

RESTRICTIONS
The -synfid command requires that the data in working register be spectral parameter data (as generated by hsvd, lpsvd, vpnls etc...).

SEE ALSO
lpsvd, hsvd, vpnls, dump2term, dump2file

FILES

V_synfid.c

3.8.4 VPNLS

NAME
vpnls - Variable Projection Nonlinear Least Square estimation

SYNOPSIS
vpnls np noff maxlw minlw itmx fxbrd

DESCRIPTION
The -vpnls command is a nonlinear optimization method in which the linear parameter vector is replaced by the linear least squares projection functional. Starting from a initial estimate (the V commands -hsvd or -lpsvd may be used to get this initial estimate), -vpnls iteratively calculates maximum-likelihood parameters (when the noise is white and Gaussian). This can be used, for example, to estimate NMR spectral parameters from measured FID (free-induction decay) data. A total of np points, beginning with an offset of noff, are used to refine the linewidths, frequencies, and intensities given by the spectral parameters in the second register, using the data in the working register.

np - number of samples (points) to use in estimation

noff - offset into the leading dimension of data to use

maxlw - discard peaks with linewidths greater than

minlw - constrain peak linewidth to be bigger than minlw

itmx - maximum number of iterations

fxbrd - flag == 1 to fix the frequencies and linewidths of the broad components with linewidths bigger than maxlw.

When the working register contains multi-dimensional data, it is assumed that the fastest dimension has the measured FID's, and the VPNLS algorithm is performed on each one. In this case, the initial parameter data in the second register can be either 1D or multi-dimensional. In the former case, the data serves as the initial data points for each FID's. If, on the other hand, the data is multi-dimensional, then it must have the same number of dimensions as the FID data.

The VPNLS parameters are returned in the working register.

EXAMPLES
1. To fit 7 peaks to the data in the working register using the first 64 data points and using HSVD as an initial estimate (with the sampled FID in the working register),

V> -hsvd 32 0 7 -xchg -vpnls 32 0 0.9 0.01 25 1

RESTRICTIONS
The -vpnls command requires the FID data in the working register be COMPLEX and the parameter data in the next register be in the parameter format. The parameter data either has one dimension or has the same number of dimensions as the FID data.

SEE ALSO
hsvd, lpsvd, synfid, dump2term, dump2file

FILES
V_vpnls.c

3.8.5 XTRP

NAME
xtrp - back-extrapolation of FID data given spectral parameters

SYNOPSIS
xtrp nmis nbrd [maxlw: maxlw][minlw: minlw]

DESCRIPTION
The -xtrp command back-extrapolates a FID data in the working register using the spectral parameters in the second register. The initial nbrd data points corrupted by broad components and the missing nmis data points caused by phase gradient pulse delay will be restored using the given spectral parameters.

nmis - number of missing points due to the phase gradient pulse delay.

nbrd - number of data points in the beginning of FID data corrupted by broad components and/or gradient pulse.

The optional parameters, maxlw, minlw, allow the user to discard the broad components or trivial spectral peak parameters (as estimated by hsvd, lpsvd, or vpnls , for example) before they are used in the back-extrapolation.

maxlw - discard peaks with linewidths greater than maxlw

minlw - constrain peak linewidth to be bigger than minlw

INPUT:
Register 1: Spectral parameters fit to data
Register 2: FID data

OUTPUT:
Register 1: Extrapolated data

When the registers contains multi-dimensional data, it is assumed that the parameters are in the leading dimension, and extrapolation is performed along the remaining dimensions.

The extrapolated FIDs are returned in the working register.

EXAMPLES
1. To remove the baseline distortion corrupted by broad components in the FID data in the second register, the parameter data in the working register is used to replace the first 12 data points in the FID (all peaks whose linewidths are less than 0.001 or greater than 0.05 are discarded) V> -xtrp 0 12 maxlw:0.05 minlw:0.001

RESTRICTIONS
The -xtrp command requires at least two existing registers. The working register should contain the (REAL) parameter data and the second register should hold the (COMPLEX) FID data. The registers must have the same number of dimensions.

SEE ALSO
lpsvd, hsvd, vpnls, synfid, dump2term, dump2file

FILES
V_xtrp.c

3.9 CONSTRAINED RECONSTRUCTION

3.9.1 GSLIM

NAME
gslim - Compartmental NMR spectra estimation method (short for Generalized SLIM)

SYNOPSIS
gslim operation [reg: regularization]

DESCRIPTION
The -gslim command uses the generalized series modeling concept to refine SLIM compartmental spectra estimates. The function can return two types of results: (1) metabolic maps for a given range in spectral frequency, or (2) the GSLIM average compartmental spectra.

The operation parameter can take on one of two values, depending on the operation to be performed:

M_MAP compute metabolic map
C_SPECTRA compute GSLIM average compartmental spectra

The optional parameter, regularization is the regularization value used in solving the block-Toplitz system (see [1]).

INPUT:
Register 1: SLIM compartmental spectra (REAL)
Register 2: k-space sampled spectra (COMPLEX)
Register 3: corresponding mask image

OUTPUT:
Register 1: metabolic map or GSLIM compartmental spectra.

EXAMPLES
1. Assuming that you have the SLIM compartmental spectra data in slimspec.df and baseline-corrected FID data in flatfid3d.df from 3D FID data sample/fid3d.df and 3D mask image data sample/mask3d.df (refering to example 2 of -slim man page), the following commands create the metabolic map for PCr and ATP concentrations, V> -i sample/mask3d.df -push -i flatfid3d.df -pishft1d -ft1d -chunk 112 0 0 0 9 4 4 4 -push -i slimspec.df -chunk 112 0 9 3 -gslim M_MAP reg:7000

REFERENCES
[1] Z.-P. Liang and P. C. Lauterbur, "A generalized series approach to MR spectroscopic imaging," IEEE Trans. Med. Imaging. v. MI-10, pp. 132-137, 1991.

SEE ALSO
slim

FILES
V_gslim.c

3.9.2 RIGR

NAME
rigr - Reduced-encoding Imaging by Generalized series Reconstruction

SYNOPSIS
rigr [reg: regularization] [filter: filter] [pc: pc]

DESCRIPTION
The -rigr command use the high-resolution ("reference") k-space data in the second register to reconstruct an image from the low-resolution ("dynamic") k-space data in the working register using the generalized series approach. This can be used for functional imaging, for example, where spatial information from a high-resolution pilot anatomical study can be used to in the reconstruction from subsequent high temporal resolution (low-spatial resolution) data.

The -rigr command accepts three optional parameters:

regularization - regularization value for Toeplitz solution

filter - flag = 1 if Hamming filtering is to be applied to the low-resolution data

pc - phase constraints mode (default = 0)

RIGR reconstruction is performed along the leading dimension only, so that it may be necessary to first transform along the remaining dimensions.

INPUT
Register 1: (COMPLEX) low-resolution k-space data
Register 2: (COMPLEX) high-resolution k-space data

OUTPUT
Register 1: (COMPLEX) RIGR reconstruction

EXAMPLES
1. With the (COMPLEX) high resolution data in Register 2 and dynamic (lores) data in Register 1 (already transformed along the frequency encoded dimension), to use -rigr with 0.1 regularization, V> -rigr reg:0.1

RESTRICTIONS
Reference and dynamic data must be 2D and have the same second dimension. Both data sets must be COMPLEX.

FILES
V_rigr.c

3.9.3 SLIM

NAME
slim - Spectral Localization by Imaging

SYNOPSIS
slim normal_flag [x_s x_o x_e ...]

DESCRIPTION
The -slim command uses structural information from pilot anatomical images to constrain the reconstruction of spectroscopic data. The k-space sampled FID (free-induction decay) data is in the working register. The corresponding compartmental mask image is in the second register. SLIM can return either the total FID signal for a given compartment or the signal normalized by the volume(3D)/area(2D) of the compartment. Only one parameter is required:

normal_flag - if none-zero, output normalized FID signal, otherwise, the total signal.

The optional parameters, x_s, x_o, x_e allow the user to set the k-space coverage of the FID data relative to the image data. The total number of dimensions must be identical to the FID k-space data and reference image mask data.

x_s - starting k-space gradiant location in k-x dimension

x_o - offset of the k-space gradiant in k-x dimension

x_e - final k-space gradiant location in k-x dimension

INPUT
Register 1: Measured FID data
Register 2: Compartmental mask image

OUTPUT
Register 1: SLIM compartmental amplitudes

EXAMPLES
1. To generate the total FID signal for each compartment outlined in the mask image, V> -i sample/mask3d.df -push -i sample/fid3d.df -slim 0

2. To generate the average FID signal for each compartment in the mask image sample/mask3d.df and correct for baseline distortion, first estimate the initial spectral parameters for narrow peaks from the central FID of a 4x4x4 3D fid data in sample/fid3d.df, and save it in the second register, V> -i sample/fid3d.df -chunk 0 2 2 2 256 1 1 1 -redim 1 256 -lpsvd 96 4 12 -dump2term parameter -xchg

Next, estimate the broad components for each FID and combine with the center narrow peaks to form initial estimates, V> -i sample/fid3d.df -lpsvd 4 0 2 -xchg -pop -pendpar -xchg -pop -xchg

Third, use the variable projection nonlinear least squares fitting routine -vpnls to estimate the parameters, and back-extrapolate the original 3D FID set by the parameters to remove the broad components (saving the resulting FID data as flatfid3d.df,) V> -i sample/fid3d.df -xchg -vpnls 128 0 0.05 0.001 20 1 -xchg -pop -xtrp 0 12 mxlw:0.05 mnlw:0.001 -o flat3d.df -xchg -pop -xchg

Finally, run SLIM, and save the compartmental spectra

to slimspec.df and display, V> -i sample/mask3d.df -xchg -slim 0 -xchg -pop -xchg -pop -pishft1d -ft1d -abs -o slimspec.df -gplot 512 128 -displ

RESTRICTIONS
The multi-dimensional FID data must have one more dimension than the corresponding mask image data. The working register must be COMPLEX, and the mask image in the second register must be REAL.

SEE ALSO
gslim

FILES
V_slim.c

3.10 MISCELLANEOUS

3.10.1 HISTORY

NAME
history, h - display the command history

SYNOPSIS
h

history

DESCRIPTION
The -history and -h commands display the previous V commands executed by the user in a numbered list. Those commands can be invoked again by simply typing in the command number.

EXAMPLES
1. To view the recent history of executed V commands, type V> -history and enter the number of the command to execute again.

FILES
V_history.c

3.10.2 MAN

NAME
man - display manual pages

SYNOPSIS
man [command_name]

DESCRIPTION
Displays the manual pages corresponding to command_name.

EXAMPLES
1. To display an overview of the V processing enviornment, V> -man

2. To see the man page for the command -lpsvd, V -man lpsvd

FILES
V_man.c

SEE ALSO
help

Chapter 4

V PROGRAMMING

The toolbox of the MR scientist is continually growing. Without the capacity to expand its library of functions, no program would be complete. V was created to facilitate the incorporation of newly-developed or often-used functions. With this capability, it is hoped that V will serve as not only as a research tool but also as a platform for distributing software among the MR research community. This section is intended as a springboard for further development. It is meant to familiarize the reader interested in developing V functions with only the necessary information; by enlarge the details about the internal operation of V have been omitted for brevity.

4.1 Directory Structure

The V directory structure has been tailored for the programmer. All the files nessary for V programming should be located in the src subdirectory of the main V directory (see Chapter 1, Section 3). In the src directory there exist several subdirectories, including

main — source code for the V kernel
include — include files
public — public domain code
reg_manip — register manipulation functions
io — input/output functions
math — math functions
transform — transformation functions
lp — linear prediction functions
nlls — variable projection functions
rigr — RIGR functions
slim — SLIM functions

New functions which easily fall into one of the above categories can be developed in the appropriate directory. Otherwise, a new directory can be added in the case that a completely new type of function is under development. In such a case, the makefile will have to edited to include files in the new directory.

4.2 Contents

Section 4 explains how to develop *V library functions*. These functions serve as the data processing engine of V, and can be developed essentially independently of V in any programming language.

Section 5 details the design of the *V calling interface*. If the library function is the heart of a new V function, then the *calling interface* is the brain. These programs act as intermediates between the V kernel (and

the user) and the V library functions.

Section 6 contains the procedure for creating documentation for your new V function.

Section 7 gives instructions for modifying the existing V source code in order to incorporate your new V function. This procedure was done with the novice programmer in mind, so that it can be done with very little difficulty.

Section 8 describes how V manages data. An understanding of the V data structure is essential for V programming. Additionally, the section touches on the issue of how V stores and manipulates registers. While a full understanding of the internal register management is not at all necessary for V programming, some familiarity with this material may be useful. The underlying motivation is to give the programmer enough rope to swing with, but not enough to hang himself on.

4.3 A Recipe for the Development of New V Functions

There are six steps that must be taken every time a new function is to be built into V:

- **Create the Library Function**
- **Create the Calling Interface**
- **Create the Manual Entry**
- **Edit the Makefile**
- **Edit src/main/commands.h**
- **Make V**

The remainder of the guide should serve to explain the purpose of each of these steps.

4.4 Library Function Development

The *library function* represents the heart of a new function. Data is the registers ends up being processed in one form or another by the library functions. Any C or FORTRAN 77 code can be easily made into a library function with some minor modifications. Hence, development of these functions should be pursued similarly to the development of a subroutine or function– that is, outside of the V framework. In writing the code for these functions, the programmer should focus on the technique or algorithm at hand rather than the details of obtaining parameters from the user, organizing the data, manipulating the registers, etc.

While actual processing required by any algorithm will differ widely from application to application, the programmer must include two elements in its coding in order to make it a V library function.

1. Every library function is (as rhw name might suggest) a function, and should return an integer. If an error occurs in the processing within the library function, then it should return the integer 1. Otherwise, it should return 0 to indicate an error-free exit. In this way, it is easy to test for errors while processing data. A good rule of thumb is to deal with the large majority of errors– invalid data, singularities, failed memory allocation, etc– that occur in processing should be dealt with outside of the library function.

2. When the library function returns a 1, then an error has occured. How should the error be dealt with? To a large degree, how an error is resolved depends upon what type of error occurs. It is the resposibility of the library function to provide some information as to what type of error has occurred. This should be

done by passing an integer error variable to the library function. On error (i.e., when, the library function returns 1), this variable should contain a code, interpretable by the calling interface, indicating the type of error that has occurred.

Perhaps the best way to illustrate the concept of the library function is by an example. The following is a library function which takes the magnitude of each element in a complex data set:

```
c-----------------------------------------------------------------------
c
c  getmag:  Return the magnitude of each element of a complex vector
c
c  V
c  University of Illinois at Urbana-Champaign
c
c  On Entry:
c             n - number of elements in given vector
c             z - n-dimensional complex vector
c  On Return:
c             getmag - 0 on success, 1 on error
c             mag    - result of operation
c             err    - always 0 for this simple operation
c
c-----------------------------------------------------------------------
c
function getmag(n,z,mag)

complex    z(*)
real       mag(*)
integer    n,getmag
c
getmag=0
c
do 10 i=1,n
   mag(i)=cabs(z(i))
 10 continue
c
return
end
```

Example Library Function

In order to get a better idea of the structure of more complex library functions, the reader is encouraged to explore the code for the library functions supplied with this distribution of V. Once the library function has been tested and debugged, the next step is to create a calling interface.

4.5 Calling Interface Development

The calling interface acts as the liason between the driver (the user) and the library function. It should organize the data as required before sending it to the library function, parse the the user input to obtain the required parameters for processing, put the result of the operation in the desired register, etc. Each V function should have its own calling interface. These interfaces must be written in C, and are named by convention as V_function() (i.e, V_abs()).

While the calling interface will vary significantly from function to function, *each should accept as arguments only three parameters*, argc, argv, *and* err. The argc and argv parameters function analogously to the global variables by the same name used in standard ANSI C. The parameter argc is an integer whose value equals the

number of parameters sent to the function. The variable argv is an array of strings containing the parameters from the user input. Finally, the variable err is a character pointer for storage of an error message.

As an example, consider the case when the user issues the command to linearly scale a set of data: -linscl 0 255. V will call the calling interface for the - linscl operation, namely V_linscl(). The parameters passed to V_linscl() will then have the following values:

argc (integer) 2
argv[0] (character) "0"
argv[1] (character) "255".

It is the reponsibility of the calling interface to use these parameters to determine exactly what operation to perform on the data. It must determine whether the passed parameters have suitable values, and should perform any necessary manipulation of the registers. Moreover, it should tell the V driver when any errors have occurred.

In general, the calling interface must include

1. **Basic usage** information;
2. **Parsing** of the input parameters to determine their validity and to use them appropriately;
3. **Testing** for the existence of necessary registers;
4. **Calling** the library function with the appropriate data and arguments; and
5. **Error messages** to return to the V driver to inform the user of any errors that have occurred in processing.

In order to accomplish these tasks, the routine will consist primarily of string manipulation functions to analyze the parameters in argv[] register manipulation functions. Most of thw string analysis (parsing) can be achieved using the standard C function strcmp(). A repertoire of register manipulation functions is included with the V source in an effort to make register manipulation transparent to the programmer. These intrinsic functions are covered in Section 5.

Every calling interface should include the file V.h by putting the text #include "V.h" at the beginning of each file in which calling interfaces are defined. Moreover, *two variables should almost always be declared at the beginning of every calling interface* V_reg *and* usage. While the former provides access to the data registers, the latter will include basic usage information that the function can return on error.

The usage information will be used by the intrinsic -help function and should be referenced when the first parameter sent in argv[] is a question mark. The code to do just this consists of only a few lines:

```
sprintf(usage,"USAGE: -function_name parameter_1 ... ");

if(argc)
  if(!strcmp(argv[0],"?")) {
    sprintf(err,"%s",usage);
    return(2);
  }
```

Notice that in this case, when the initial parameter is "?", the calling interface returns the value 2 to signify not that an error has not occurred, but rather that the string err contains usage information. If these few lines are not included in the calling interface, then the user will not have access to the brief help provided by the -help function command.

One operation that should be almost unanimously present in every calling interface is to check for the existence of the necessary registers. Data of specific types and dimensions may be required in one or more registers. To check for the existence of a working register (register 1), the following code may be used:

```
  if(!V_reg[0]) {
    sprintf(err,"Cannot operate on nonexistent data");
    return(1);
  }
```

To check for more than one register, the internal V function reg_count() may be employed. This function operates on the register stack as one might imagine: it returns an integer indicating the number of registers that are present in the stack. To check for the existence of 4 registers, for example, the following code may be substituted:

```
  if(reg_count()<4) {
    sprintf(err,"Cannot operate on nonexistent data");
    return(1);
  }
```

The reg_count() and other register manipulation functions are described in further detail later in this chapter.

The registers themselves can be reference directly by including the following global variable in the variable declaration section:

```
  extern struct V_struct **V_reg;
```

Upon including this line, registers may be accessed directly, the first (working) register as V_reg[0], the next as V_reg[1], etc. These references are actually pointers to the registers, so the programmer who wishes to directly access or modify the register contents must familiarize himself with the V_struct data structure. More on this later.

The calling interface must also be a function which returns one of three values. The (integer) values returned V_function have the following significance

0 no error has occurred

1 an error has occurred

2 the user has asked for the command usage

When an error has occurred, then an error message to be returned to the user must be present in the string variable err.

As an example, the calling interface for the -abs function is shown below. The reader is also encouraged to browse through the calling functions that have been distributed with V.

```
/* ---------------------------------------------------------------------
 * V_abs:  Calling interface for magnitude operation
 *
 * V
 * University of Illinois at Urbana-Champaign
 *
 * Subroutines called: L_cabs,scopy
 * --------------------------------------------------------------------- */

#include "V.h"
#include <malloc.h>
#include <string.h>

int V_abs(argc,argv,err)
int   argc;
char *argv[];
char *err;
{
```

```
extern struct V_struct   **V_reg;
char usage[MAX_STRING];

int     nelt,i,reg;
float  *work;             /* Workspace       */
char   *label;            /* Space for label */

  sprintf(usage,"USAGE: -abs");

/* ----------------------------
 * PARSING
 * ---------------------------- */

  if(argc)
    if(!strcmp(argv[0],"?")) {
      sprintf(err,"%s",usage);
      return(2);
    }

  if(!V_reg[0]) {
    sprintf(err,"No working register'');
    return(1);
  }

/* ----------------------------
 * OPERATION
 * ---------------------------- */

  reg =reg_top();
  nelt= reg_nelements(V_reg[reg]);

  if(V_reg[reg]->data_type == COMPLEX) {        /* COMPLEX  operation */

    work = (float *)malloc((unsigned)nelt*FWS);
    if (!work) {
      sprintf(err,"Out of memory");
      return(1);
    }

    if(L_ABS(&nelt,V_reg[reg]->data,work)) {  /* Call abs library */
      sprintf(err,"Magnitude operation failed");
      return(1);
    }

    if(V_reg[reg]->label) {
      label = (char *)calloc(strlen(V_reg[reg]->label), sizeof(char));
      strcpy(label, V_reg[reg]->label);
    } else label=NULL;

    if(reg_init(REAL,V_reg[reg]->ndim,V_reg[reg]->dim,err)) return(1);
    if(reg_load(work,err)) return(1);

    if(label) if(reg_label(reg_top(),label,err)) return(1);

    free (work);

  } else if(V_reg[reg]->data_type == REAL)      /* REAL operation */
    for(i=0; i<nelt; i++) V_reg[reg]->data[i] = abs(V_reg[reg]->data[i]);

  if(argc) {
```

```
    sprintf(err,"Extra parameters ignored\n\t%s",usage);
    return(1);
  }

  return(0);

}
```

Note also that the library function is called as L_ABS(). This is actually a macro definition, since the -abs library function is a FORTRAN subroutine. All such macros are should be included in the file **include/V.h**. The existing macro definitions in this file should serve as examples.

4.6 Documentation

No new function would be complete without some documentation to provide new users at least a rudimentary understanding of how to use it. Documentation should be written for any new function, so that new users can easily use the routine. The manual pages are written in nroff or troff, as is the case in the vast majority of operating systems. The V manual entries are located in the directory **man/man1**, in files named with a .1 suffix. For example, the manual entry for the -abs command is located in the file **man/man1/abs.1**.

The best way to write documentation for new routines is to use the existing manual entries as examples. In general, each entry should include sections entitled NAME, SYNOPSIS, DESCRIPTION, EXAMPLES, RESTRICTIONS, REFERENCES, SEE ALSO, FILES, AUTHORS, and COPYRIGHT. A good rule of thumb is to place all relevant information into the manual entry, so that no confusion arises when a user unfamiliar with your new function can easily apply it to his or her data.

4.7 Modification of the V Source

After the calling interface and the library function have been created, then the V source code must be modified to include the new V function. Upon completing these modifications, V can be rebuilt. Two files must be modified: **src/MAKE.PLATFORM** and **src/include/command.h**.

The file **src/MAKE.PLATFORM**[1] must be modified to tell the compilers and linker where the new source code is. A few lines into the makefile, the programmer will find an entry called **USRDIR**. On this line, type the path where the new code is located. Two more lines exist directly under the first, **USR_C** and **USR_F**. These lines are for new C and new FORTRAN-77 code, respectively. Each entry under these two lines should of the form $(USRDIR)/filename.

Once the makefile has been edited, then two lines must be added to the file **include/commands.h**. First, new C functions must have prototypes. The first part of **include/commands.h** includes the prototypes for all V functions. To include a new V function, add a function prototype in the section for your system. Next, towards the latter half of **include/commands.h** the programmer should find a definition of the *command table*. A new entry should be placed in alphabetical order in the command table for each new V function. Follow the instructions given in the file and use the other table entries as examples.

After these modifications, V can be rebuilt by typing make -f MAKE.PLATFORM in the **src** directory. The new V binary will replace the old one, so it may be wise to make a copy of the old V binary before attempting to build the new one.

[1] At the time of this writing, PLATFORM should be one of SUN4, IRIS4, HP, or IBM6000.

4.8 V Data Structure and Register Management

One feature that makes V unique is its emphasis on data *structure*. The same data structure is used internally and for disk storage. The underlying data structure in V is the V_struct:

```
typedef struct V_struct {
        char *label;
        float *data;
        int data_type;
        int ndim;
        int *dim;
} V_struct;
```

The large volume of data in MR makes it easy to forget how it has been processed. The V_struct uses a simple label identify a set of data. This label is accessed via the pointer *label. Data is also accessed via a pointer, *data. Multidimensional data are stored as floating point numbers in row-major order (final array index varies fastest). The size of each dimension is stored as an array, dim[]. The number of dimensions, ndim, is unlimited; five-dimensional data is stored in the same fashion as two-dimensional data. The data themselves are stored as floating point numbers. The data_type variable indicates the type of data that the structure contains, either real (data_type == REAL) or complex (data_type == COMPLEX). Complex data are stored as interleaved real and imaginary parts (two sequential floats for each complex number).

A library of register management functions exists for easy manipulation of data, but some knowledge of the internal storage of registers is useful to the programmer. The register storage can be thought of as an array. This array is global in scope and hence can be directly accessed by individual functions. As mentioned above, each calling interface should include the following variable declaration to ensure access to the registers

```
extern struct V_struct **V_reg;
```

Each element in the array is either a pointer register (if it exists) or is a null pointer (if no register exists). These pointers are stored in natural order in the array, so that the working ("top") register is stored as the first element in the array. Hence, if three registers exist, they are stored as

```
V_reg[0]  ⟼  Register 1
V_reg[1]  ⟼  Register 2
V_reg[2]  ⟼  Register 3
V_reg[3]  ⟼  (null)
.
.
.
```

It is the responsibility of the programmer to determine if data exists before using it.

Several intrinsic functions exist for easy manipulation of the registers and their contents:

- int reg_create(char *err)
  When a new register must be created by a function, the reg_create() routine should be used. The calling syntax for reg_create() is much like any other V function; the function also returns the value 0 upon a succesful return, just as most any other V function. If a new register could not be created, then 1 is returned, and err will contain an error message. The function has the effect of creating a new register pointer, **not** of creating space for data. Most functions need not call reg_create(), since processed data usually replaces the data that was in the working register. Since the working register existed prior to calling the function, there is no need to create a new register.

- int reg_push(char *err)
  Pushing the working register is similar to creating a new working register, but is not the same. Whereas

reg_create() creates a new register pointer, reg_push() copies all the data that already exists in the working register, including the data type, the dimension information, and the label, if any.

- int reg_pop(char *err)
  This function is the dual of reg_push(); calling it has the effect of freeing all memory allocated for the current working register. The next register in the register stack becomes the working register if it exists- if not, then the resulting stack is empty. Any errors are returned in the err string when the function returns a 1.

- int reg_init(int reg, int data_type, int ndim, int *dim, char *err)
  This initialization routine should be called immediately following reg_create(). Whereas the creation function creates space for a register, the reg_init function creates space for the data to be contained within the register. The reg parameter indicates the register that is to be initialized (0 for the working register, 1 for the next register in the stack, etc.). The data_type parameter may be either REAL or COMPLEX (both defined in V.h). Whereas ndim indicates the *number* of dimensions, the dim variable is an array of size ndim, each element of which denotes the size of the corresponding dimension. Finally, on error the err parameter contains an error string.

- int reg_redim(int reg, int data_type, int ndim, int *dim, char *err)
  The reg_redim() function has the same effect as the reg_init() function, only it preserves any existing data in the register. This function should be used prefentially to reg_init(), since most often the processed data can be deposited in the existing working register after using reg_redim() to change its size.

- int reg_load(int reg, float *data, char *err)
  This function copies the contents of the array data into the given register reg (0, 1, ...). Note that an error will result when the given register does not already exist. The number of data that will be copied is determined by the dimensions given in the (already initialized) register.

- int reg_unload(int reg, float **data, char *err)
  The reg_unload() function is the dual of the reg_load() function; data from the given register is copied into the array data. Note that the data array is passed as a pointer– this means that the memory allocation is done inside of reg_unload(). Hence, if the variable data is declared as a pointer to a float, i.e., float *data, the the function call would be reg_unload(reg,&data,err).

- int reg_label(int reg, char *label, char *err)
  To attach a string label to a register, reg_label() can be called. The parameter reg should hold the value of the register to be labelled. An errors are returned in the err string.

- int reg_settype(int reg, int data_type, char *err)
  This function sets the data type of the register reg. It *does not* make real data into complex data or vice versa, but rather sets the data_type variable for the register. Hence, if an existing register contains real data and reg_settype() is called with data_type = COMPLEX, then the leading dimension of the data dim[0] will be halved.

- int reg_count(void)
  No parameters should be passed to reg_count(). This function returns the number of registers currently present in the register stack. Hence reg_count() equals zero when there are no registers, one when only the working register has been allocated, etc. This function can be used to test for required registers before processing.