GKV User's Manual

Version 1.2

W.M. Nevins
LLNL

7/28/02

Table of Contents

1. Introduction

GKV is an interactive data analysis and visualization tool designed for use in analyzing output from plasma turbulence simulations. Developed as part of the *Plasma Microturbulence Project*, it is an object-oriented application built upon *IDL* (the Interactive Data Language, a product of Research Systems, Inc.). *GKV* imports turbulence simulation data from data files written by various plasma simulation codes. Data which has been imported into GKV is represented as a *GKVsd* object. Data analysis and visualization are then accomplished within *GKV* through *Methods* which act on the *GKVsd* objects to produce new *GKVsd* objects containing analyzed data (*e.g.*, the method *XCORR* acts on a *GKVsd* object and produces correlation functions, which are returned as a new *GKVsd* object). These methods enhance the power of the native *IDL* language, making it possible to perform significant amounts of data analysis and to produce presentation quality graphics with a relatively few lines of *IDL* script which can either be entered interactively at the *IDL* command line (for interactive use), or through the use of *IDL* scripts (for batch use).

We assume that the reader is familiar with the use of *IDL*. Object-oriented *IDL* differs slightly from what some *IDL* users may be familiar with in that

1) An objects data is only available through that objects methods — so you can only access the data within *GKVsd* objects through the methods provided by *GKV*.

2) All *methods* which act on objects within *IDL* are either *Procedures* or *Functions*. You invoke Procedure method as follows:

   Obj -> Procedure, args

   where *"Obj"* is an *IDL* object, *Procedure* is a *Procedure method* previously defined on this class of objects, and *args* are the various arguments expected by this procedure. The compound symbol "->" (a hyphen, "-", immediately followed by a greater than sign, ">") is used by *IDL* to invoke procedures.

3) *Function methods* are invoked as follows:

   result = Obj -> Function(args)

   Where *result* is the information returned by this Function Method, *Obj* is an *IDL* object, *Function* is a Function method previously defined on this class of objects, and *args* (now enclosed in parenthesis) are the arguments expected by this function..

2. Initializing GKV

You must first obtain a copy of the (hopefully current) *GKV* distribution folder. Within this folder you will find

1) *GKV.prj*, which is an *IDL* "project file". Project files are rather like *make* files, and are used to control the compilation order of the associated *IDL* source files. These project files are **not** backward compatible. If your version of *IDL* does not recognize *GKV.prj* as a valid project file, look for a file named *GKV_X.X.prj*, where "*X.X*" corresponds to the version of *IDL* which you are currently running.

2) *source*, which contains the *IDL* source files for *GKV*.

3) *NETCDF_pg3eq*, which contains the *IDL* source files for various routines which you will *only* require if you need to import data from the **.h.nc*, **.x.nc*, and **.c.nc* NetCDF files which are written only by the plasma micro turbulence code *pg3eq*.

4) *data*, which contains some sample data (from simulations of the *CYCLONE base case*) which you can use to practice with *GKV*.

2.1 IDL Version Requirements.

GKV is an *IDL* application, so the first step is to initialize *IDL*. This requires a license for IDL, which may be obtained from Research Systems, Inc.[1] I chose to write *GKV* as an *IDL* application because most fusion laboratories already have licenses for *IDL*. Research Systems periodically updates *IDL*, and I expect *GKV* users to see that the version of *IDL* available at their site is up-to-date. I currently support a version of *GKV* which operates under *IDL* Version 5.4; and expect to develop new procedures which require *IDL* Version 5.5 (particularly procedures for importing data from *HDF5* files) for full functionality.

On MACs, *IDL* can be initialized by simply "double-clicking" on the project file, *GKV.prj*, On UNIX systems I generally use the *IDL Development Environment* (a part of the normal *IDL* distribution), which is invoked by entering *IDLDE* on the command line. *IDL* is also available for *WINDOWS* systems. I've very little experience with using *IDL* under WINDOWS, but those users who have used *GKV* under WINDOWS have report that it works pretty much as advertised.

---

[1] *Research Systems, Inc.*
2995 Wilderness Place
Boulder, CO 80301
phone: (303)786-9900
fax: (303) 786-9909
e-mail: info@rsinc.com
http://www.rsinc.com

2,2 For pg3eq Users Only

If you are not planning to import data from NetCDF files produced by *pg3eq* in the current *GKV* session, skip this section. For those of you planing to import such data it is necessary to compile the source files in *NETCDF_pg3eq* directory before compiling *GKV*. This is accomplished by going to the *"file"* menu of *IDL*; choosing *"open"*; using the *"widget"* which pops up to open the *NETCDF_pg3eq* directory, and choosing the file *make_NETCDF_pg3eq.pro* within this directory. Now type

@make_NETCDF_pg3eq

on *IDL*'s command line (NOTE: *case* is important within *IDL* only when referring to files by name — this is, in exactly the context described here! Get those CAPS in the right place). *IDL* should respond by printing out a few pages of "module names" as they are compiled. It is good practice to close *make_NETCDF_pg3eq.pro* once *IDL* finishes compiling these modules.

If you need to compile the files in the *NETCDF_pg3eq* directory after compiling *GKV*, it will be necessary to recompile *GKV* after compiling *NETCDF_pg3eq* to avoid naming conflicts. This can be done from the *"Project"* menu as described in §2.3 below.

Finally, to avoid conflicts relating to color tables, etc. it is best to import data from *pg3eq* immediately after compiling *GKV*, and ***before*** running *GKV_INIT* (as described in §2.4 below).

2.3 Compiling GKV

After starting *IDL* (and compiling *NETCDF_pg3eq* if this is necessary for you) open the project file, *GKV.prj*. If you are on a MAC, and you initialized *IDL* by clicking on this project file, then it is already open. If you are on a MAC, but have initialized *IDL* without "double-clicking" on the *GKV.prj* project file, then you can open it by going to the *"file"* menu of *IDL*; choosing *"Open";* and using the *"widget"* which pops up to select *GKV.prj* within the *GKV* distribution folder. Once *GKV.prj* is open, *GKV* can be compiled by either selecting the *"Compile Modified Files"* button at the top of the project window, or by going to the *"Project"* menu and selecting *"Compile All Files"* (or, *"Compile Modified Files"* — it really doesn't make any difference here).

On UNIX systems the *GKV.prj* project file can be opened by going to the *"file"* menu of *IDL*; choosing *"Open Project …";* and using the *"widget"* which pops up to select *GKV.prj* from within the *GKV* distribution folder. Once the project file *GKV.prj* has been opened, go to the *"Project"* menu; choose *"Compile"* and then *"All Files"*.

IDL should respond to your *"Compile"* command by printing out several pages of

```
% Compiled module: COLOR_TRUE.
% Compiled module: COLOR_LIST.
% Compiled module: COLOR_INDEX.
% Compiled module: COLOR_INDEX_TEST.
% Compiled module: COLOR_IMAGE_BOTTOM.
% Compiled module: COLOR_IMAGE_NCOLORS.
% Compiled module: COLOR_IMAGE_RANGE.
% Compiled module: COLOR_IMAGE_SET.
% Compiled module: COLOR_SETUP.
```

etc.

If you get any compile errors, this is most likely due to a problem with the order in which routines have been compiled. If this is the case, it can be remedied by simply issuing the compile command again.

2.4 Initializing GKV

After having compiled *GKV* it must be initialized. This is accomplished by typing

```
GKV_INIT
```

on the *IDL* command line. Successful initialization will result in a window popping up containing a line plot (necessary as *IDL*'s graphics state is altered on many platforms when the first plot is drawn); a disclaimer being printed in the output window (necessary to keep the University of California's lawyers happy); and the *IDL* prompt changing from *"IDL"* to *"GKV"*.

If you're planning to import data from *pg3eq* it's best to do this ***before*** running *GKV_INIT*. Executing *pg3eq_Data* after running *GKV_INIT* leads to conflicts with color tables, etc.

2.5 Choosing Working and Source Directories

While it is not strictly necessary, it is very useful to begin your *GKV* session by defining two string variables: *work*, which should contain the full path to the directory containing the data files which you plan to analyze (and, where we will store our analyzed data); and *source*, which should contain the full path to the directory containing any *IDL* source files which you are using in you data analysis that may require debugging (I generally have *source* contain the full path to the *GKV* source directory within the *GKV* distribution folder). This is easily accomplished with the widget *GKV_ChangeDir*. Simply enter

source = GKV_ChangeDir()

or

work = GKV_ChangeDir()

on the *IDL* command line and use the widget which will pop up to choose the appropriate directory. You can verify that you have chosen defined the proper paths by examining the contents of the string variables, *source* and *work* within *IDL*'s Variable Watch window.

3. Importing Data from Simulation Code Output Files

GKV provides routines for importing data from output files created by various plasma turbulence simulation codes. These routines both read in the data and create GKV objects.

3.1 Importing Data from NETCDF Files in Standard Format

NETCDF_DATA

Usage:

result = NETCDF_DATA(keyword = value, ...)

Where, on return, *result* is an object array containing one GKV object for each dependent variable which was read from the NETCDF file.

NETCDF_DATA is the basic routine for reading data from NETCDF files in standard format (see below for a description of the standard NETCDF file format). Note that this routine should be able to read pretty much any NETCDF file. However, use of the 'standard' format is recommended because it allows you to transfer information about independent variables, titles, units, etc. automatically from your code to the GKV objects.

KEYWORDS:

FileName	Name of netCDF file. Must include either full path, or else the path from the current working directory. If FileName is not provided, DIALOG_PICKFILE will be called to allow the user to select an appropriate netCDF file. (Optional)
Path	Sets path to initial directory selected by DIALOG_PICKFILE (which will allow user to change to other directories...). If no path is provided, DIALOG_PICKFILE will default to current working directory. (Optional)
ArrayStyle	Set to the string 'c' if netCDF file was written by code (like C or PASCAL) which stores array data in 'column major' format. Default is 'fortran', appropriate when the netCDF file was written by code (like FORTRAN or IDL) which stores array data in 'row major' format. This issue is discussed in Chapter 5 of 'Building IDL Applications'. Defaults to 'fortran'.
Threshold	Size (in units of 10^6 elements) of threshold between 'large' and 'small' data sets. User will be prompted before reading a data array

of more than 'Threshold*10^6' elements from the netCDF file, while all small data sets will be read. Default is 1 (x10^6).

Species — A string containing the name (within the netCDF file) of the DIMENSION which indexes particle species. NetCDF_Data will break separate data corresponding to different species into separate GKV objects. The species index (incremented by 1, so that the first species will be '1', rather than '0') will be appended to the objects mnemonic and added (as a subscript) to the objects title.
Defaults to 'species'.

RI — A string containing the name (within the netCDF file) of the DIMENSION which is used to index the real and imaginary part of a complex variable. Within IDL's 'row major' format this MUST be the first (that is, most rapidly varying) dimension of the array. ***NOTE*** that NCDUMP (which is implemented as a C code) should show 'RI' as the LAST INDEX of the array within the netCDF file (consistent with C's use of 'column major' format).
Defaults to ‘ri’.

Debug — Set (i.e., put '/Debug' within the argument string) to turn off error trapping. Turning off error trapping is useful if you are debugging NetCDF_Data.
Default is Debug=0 (error trapping on)

3.2 Standard NetCDF File Format

The data to be analyzed and/or archived should be stored in one or more NetCDF files, each of which has the same basic format. For our purposes, we assume that the data is stored in rectangular arrays (which isn't always the case, see GTC_DATA for an example of reading data which is not stored in rectangular arrays). Each NetCDF file contains:

1) Information on the simulation code and the particular physics run stored as character-type GLOBAL ATTRIBUTE. The global attribute names which NETCDF_DATA searches for are:

CodeID	Identifies simulation code which originated this data
CodePI	Provides further information about this simulation code.
RunID	Identifies particular simulation run which generated this data.
RunInfo	Provides further information about this simulation run.

2) Dimensions and Independent Variables. When creating a NetCDF file, dimensions must be declared and given a symbolic name. Typically, dimensions correspond to independent variables (however, see below for some exceptions). When this is the case,

then a 1-D VARIABLE should be included in the NetCDF file with the ***same*** name as the corresponding dimension (having the same name for both a dimension and a variable is both legal and encourages in the NetCDF documentation).[2] This 1-D variable contains the values of the independent variable at the grid points. Information about these independent variables should be stored as VARIABLE ATTRIBUTES associated with this 1-D variable array. Possible variable attributes are:

mnemonic	a terse identifier in all ASCII characters
idl_name	in which you can use Hershey Vector Fonts to introduce Greek characters (see IDL manual).
pretty_name	Which is a synonym for 'idl_name'
title	Another synonym for 'idl_name
long_name	Yet another synonym for idl_name'
units	n which you give the units (DON'T make the units part of the name! Hershey fonts are OK here as well)
boundary	Information about the boundary conditions in this independent variable. Supported values of 'boundary' are 'periodic (open)', 'periodic (closed)', and 'open'. 'periodic (open)' is used to describe periodic systems in which only one boundary point is stored; while 'periodic (closed)' is used to describe systems in which the boundary values are repeated at both the beginning and end of the data arrays.

3) Auxiliary Dimensions. You may require some AUXILIARY DIMENSIONS (not corresponding to independent variables) to deal with multiple species (e.g., a 'species' dimension) and with the regrettable fact that NetCDF files don't support complex data (so use 'ri' to dimension the real and imaginary parts of your complex data). These dimension names are, in effect, keywords which NETCDF_DATA recognizes and treats specially. Currently supported 'auxiliary dimensions' are:

species	Dimension name for species index. NETCDF_DATA will make a separate GKV object for each species.
ri	Dimension name for complex data. Put real part of data at *ri=0*, and imaginary part of data at *ri=1*.

Note that the symbol corresponding to both of these auxiliary dimensions can be over ridden on the NETCDF_DATA command line.

[2] See http://www.unidata.ucar.edu/packages/netcdf/ for information about netCDF files and common netCDF file conventions.

4) Dependent Variables, like the electrostatic potential or the vector potential. This data is stored in arrays of appropriate dimensionality. NetCDF files allow a single 'unlimited' dimension which should be used to index 'time'. If an unlimited dimension is used, it must be the least rapidly varying index of the array. Information about each dependent variable is stored as VARIABLE ATTRIBUTES associates with the dependent variables. Note that it is legal for different variables to have attributes with the same name. NETCDF_DATA looks for the following VARIABLE ATTRIBUTES associated with each dependent variable:

mnemonic	a terse identifier in all ASCII characters
idl_name	in which you can use Hershey Vector Fonts to introduce Greek characters
pretty_name	Which is a synonym for 'idl_name'
title	Another synonym for 'idl_name'
long_name	Yet another synonym for idl_name'
units	in which you give the units--DON'T make the units part of the name! Hershey fonts are OK here as well

3.3 Importing Data from NetCDF Files in pg3eq Format

PG3EQ_DATA

Usage:

result =PG3EQ_DATA(keyword = value, …)

Where, on return, result is an anonymous structure with one tag for each data field selected using the widget which will pop p upon invoking PG3EQ_DATA. The value associated with each of these tags is a *GKVsd* object.

PG3EQ_DATA imports data from the netcdf files (*.x.nc, *.h.nc and *.c.nc) produced by the microturbulence simulation code *pg3eq*. These files should be in the same directory — presumably that whose path is contained in the string variable *work* (see §2.5 above). Generally, the data is chosen by "pointing and clicking" within the widgets which pop up when PG3EQ_DATA is invoked. PG3EQ_DATA must be invoked separately to access data from each of the three NetCDF files with suffixes **.h.nc* (which contains scalar data vs. time and also time histories of the amplitudes of selected Fourier modes), **.x.nc* (which contains flux-surface-averaged data as a function of the flux surface label, *x*, and time), and **.c.nc* (which contains scalar fields as a function of. two spatial co-ordinates — generally *x* and *y* and time).

KEYWORDS:

path	Set this keyword equal to a legal path to the directory at which you wish to begin your search for the appropriate NetCDF file (e.g., set *path=work*, where *work* was set as described in §2.5 above). Defaults to the current working directory. (Optional)
file	Set this keyword equal to the name of the appropriate NetCDF file. Default is to use DIALOG_PICKFILE to choose the appropriate NetCDF file. (Optional)
var	Set this equal to the name of the variable within the NetCDF file which you wish to import into *GKV*. Default is to us a pop up widget to select data from that which is contained in the selected NetCDF file. (Optional)

3.4 Importing Data from GTC

GTC writes data to (at least) three different types of files: *hist.out* files, *ShearEB.out* files, and families of *Piiii.ncd* files. Three different routines have been written to import data into *GKV* from each of these file types.

GTC_History

Which imports data from GTC's *History.out* files.

Usage:

result = GTC_HISTORY(keywords = value, …)

Where, on return, result is an anonymous structure with one tag for each piece of data contained within the *history.out* file. The value associated with each of these tags is a *GKVsd* object.

Keywords:

FileName	An optional 'string' argument containing the FULL name of the 'history.out' file. If this keyword is not set (or the requested file cannot be found) then a dialogue box will pop up to allow the user to select the '*history.out*' file. When in doubt, *****DON'T***** set this keyword! (Optional)

Path — An optional 'string' argument containing the path to the directory (folder on MACs) where the pop-up dialogue box will initialize the user's search for the '*history.out*' file. When in doubt, *****DO***** set this keyword using, for example, the string *work* initialized as described in §2.5 above.

Debug — Set this keyword (i.e, put *"/Debug"* on the command line) to turn off error trapping. Disabling error trapping is most useful while debugging.

GTC_ShearEB,

Which imports data from GTC's *ShearEB.out* files.

Usage:

result = GTC_ShearEB(keyword = value, …)

Where, on return, result contains an anonymous structure with one tag for each data set stored in the *ShearEB.out* file. The values associated with each tag are *GKVsd* objects.

Keywords:

FileName — An optional 'string' argument containing the FULL name of the '*ShearEB.out'* file. If this keyword is not set (or the requested file cannot be found) then a dialogue box will pop up to allow the user to select the *'ShearEB.out'* file. When in doubt, *****DON'T***** set this keyword.

Path — An optional 'string' argument containing the path to the directory (folder on MACs) where the pop-up dialogue box will initialize the user's search for the *'ShearEB.out'* file. When in doubt, ***DO*** set this keyword using, for example, the string *work* initialized as described in §2.5 above.

NT — An optional argument containing (on input) the number of time slices to be read from 'sheareb.out'. If set, NT ***MUST*** be less than or equal to the total number of time slices in *'ShearEB.out'*. If NT is not set, then GTC_SearEB reads all time slices in 'ShearEB.out' using READ_ASCII. It's best to set NT if you know how many time slices are present, as this speeds the reading process.

DT — An optional argument containing (on input) the interval between time slices (in units of 1/Omega_ci). Defaults to 20.

Debug — Set this keyword (i.e, "/Debug") to turn off error trapping. Disabling error trapping is most useful during debugging.

Verbose — Set this keyword (i.e., "/Verbose") so that READ_ASCI prints out a message upon reading ***EACH*** record in *'ShearEB.out'*; This *****REALLY***** slows the read to a crawl, so *****DON'T***** set 'Verbose' unless you know that *'ShearEB.out'* has less than about 1000 records (or so...), or you are really motivated to know how many records are in *'ShearEB.out'*.

GTC_DATA

Which imports data from the *RUNdimenID* file, and from GTC's sequence of *Piiii.ncd* files. The *RUNdimenID* file contains information about both the *q*-profile (and the *iota*-profile) and the structure of the data arrays within the *Piiii.ncd* files. The *Piiii.ncd* files contain scalar data as a function of three spatial variables and time (they also contain information about the grid metric which is not used by GTC_DATA).

Usage:

result = GTC_DATA(keywords = value, …)

Where, on return, *result* is an anonymous structure containing one tag for each data set imported. The values associated with these tags are *GKVsd* objects.

The present version will only allow ONE type of data (i.e., output resulting from a single keyword) from a given call to GTC_Data.

Keywords:

Info — If this keyword is set (e.g., /Info), GTC_Data prints information about the grid structure and the RETURNS. Selected information is PRINTed to the IDL output log, and more information will be found within the structure returned. On this call to GTC_Data. NO *GKVsd* data objects are returned. This is mainly useful for determining if you have a useful *RUNdimenID* file.

IsurfaceNormal — Set to an integer array containing indices to the flux surfaces on which data is desired vs. usual poloidal (theta) and toroidal (zeta) angle.

IsurfaceFieldLine	Set to an integer array containing indices to the flux surfaces on which data is desired vs. poloidal angle (theta) and the field-line label zeta_0 (the toroidal angle at which field line in question passes through the outer midplane).
IradialNormal	Set to a floating point array containing the values of the poloidal angle at which slices of the data vs. radial coordinate (r) and toroidal angle (zeta) are desired.
PoloidalAvg	Set this keyword (e.g., /PoloidalAvg) to get data averaged over fieldlines (over poloidal angles -π to π) vs. radial coordinate (r) and field-line label zeta_o
ZetaCut	Set this keyword to a floating point array of zeta values at which cuts of the data vs. radius and poloidal angle are desired.
Thetacut	Set this keyword to a floating point array of theta values at which cuts of the data vs. radius and toroidal angle are desired.
ToroidalAvg	Set this keyword (e.g., /ToroidalAvg) to get data averaged along field line (over one cycle of poloidal angle from -π to π) vs. radius and poloidal angle theta_0 (a field-line label).
IfluxLimits	Set to a two-element integer array containing the indices of the first and last flux surfaces to be included in forming IradialNormal or IradailFieldLIne data.
TimeSteps	Set this keyword to the number of timesteps which you desire in the output file. If not set, defaults to timeslice from selected Pxxxx file. If it is set, then you can only get ONE *GKVsd* object from this call to GTC_Data. If timeSteps is set to an integer array (and the second value of this array is less than the first element) then the first element determines the number of files to be read (in total) while the second element determines the interval between files (defaults to 1, or every file)
Dt	Set this keyword to the time interval between GTC data file files.
R_range	Set this keyword to a two-element floating-point array containing the range of 'r/q' values corresponding to the full range of 'iflux'. Default is [0.1, 0.9].

Qprofile	Set this keyword (e.g., /Qprofile) to obtain the q-profile as a *GKVsd* object.
IotaProfile	Set this keyword (e.g., /IotaProfile) to obtain the iota-profile as a *GKVsd* object.
Q_0	Set this keyword to the value of q on axis. (defaults to the range $0.5 < Q\_0 \leq 1.0$)
Iota_0	Alternatively, you can set this keyword to the value of iota on axis (defaults to the range $1 \leq \text{iota} < 2$).
Variable	Set this keyword to a STRING containing the name of the variable within the Pxxxx file desired. This defaults to 'potential'. If 'variable' is set to a non-zero integer, then will attempt to extract data corresponding to the (Variable+1)th variable within the Pxxxx file (set Variable = 0 to get first, and probably only variable in the Pxxxx file).
ncd	Set this keyword to the suffix of the sequence of netcdf files. Defaults to "ncd" (optional).
Path	Path to begin search for the GTC netcdf files. Defaults to current directory. (Optional)

3.5 Importing Data From TUBE

The microturbulence simulation code *TUBE* stores data in ASCII output files with names *hist.out, Phixy.out, Phixz.out, Phir.out,* etc. *TUBE_DATA* simultaneously imports data from all of these files.

Tube_Data,

Usage:

result = TUBE_DATA(keyword = value, …)

Where, on return, result is an anonymous structure containing a combination of *GKVsd* objects and other anonymous structures (which, in turn, contain a combination of *GKVsd* objects and other anonymous structures …).

Keywords:

Path	The path at which DIALOGUE_PICKFILE begins its search for the directory containing the TUBE output files. Defaults to the current working directory. (Optional)
TubePath	The full and correct path to the directory containing the TUBE output files. If this keyword is set, then DIALOGUE_PICKFILE will not be used. (Optional)
nt	Number of time steps the history file. This is only used if there is NOT a *'hist.out'* file in the selected directory.
Yang	Set this keyword (i.e., put "/Yang" on the command line) to interpret the third integer in the first record of the *phi**.out*, etc. files as the number of time slices in the file (instead of the interval between time slices in these files relative to the history files). Has the side effect of Changing default value for *"CodePi"* to Y. Chen. (Optional)
Codename	Set this keyword to the name of the code which generated the output files. Defaults to "TUBE". (Optional)
CodePi	Set this keyword to the name of the PI responsible for this simulation data. Defaults to "S.E. Parker" (unless the keyword 'Yang' is set, in which case *'CodePi'* defaults to "Y. Chen"). (Optional)
RunId	Set this keyword to a terse phrase describing this run. (Hershey vector font conventions can be employed). No default, so that the RunID field on output will be blank if this keyword is not set. (Optional)
FileId	Set this keyword to a terse phrase further describing this run. (Hershey vector font conventions can be employed). No default, so that the FileID field on output will be blank if this keyword is not set. (Optional)
Debug	Set this keyword to obtain relatively verbose behavior which may be useful in debugging this routine. (Optional)

Side Effects:

On return from *Tube_Data* the current working directory will be set to the directory containing the TUBE output files.

3.6 Importing Data from GYRO

The GYRO code writes NetCDF files in the standard format described in §3.2 above. Data from these files is imported using the routine NETCDF_DATA as described in §3.1. For example, the call

GyroData = NetCDF_Data()

will generate the structure *GyroData* containing data imported from a NetCDF file produced by Gyro. However, field quantities, like the electrostatic potential, are stored as a function of radius, *r*, and toroidal mode number, *n*. GYRO only provides fourier amplitudes for positive values of *n*. It is a consequence of the reality condition that the fourier amplitudes at negative values of *n* are simply the complex conjugate of those at the corresponding positive value of *n*.

You may wish to transform this data into a configuration-space *(r, ζ)* representation, where ζ is the toroidal angle.. This is accomplished by first applying the function method:

ALL_k

Usage:

result =GyroData.potential -> All_k(arg, keyword = value, …)

or

GyroData.potential -> All_k, arg, keyword = value, …

All_k can be called either as a function method or as a procedure method. When called as a function the result is stored in *result*, while when called as a procedure the "result" will overwrite the object which it acts on. The function method *All_k* expects that the object it acts on (*GyroData.potential* in this case) contains the non-negative Fourier coefficients (vs. the independent variable '*arg*') from a real data series. On return, *result* will contain a full set of both positive and negative Fourier coefficients, where the coefficients for negative values of '*arg*' are formed by taking the complex conjugate of those at positive values of '*arg'*.

Arguments:

arg	A valid axis ID. That is, the axis number or any valid axis mnemonic. Defaults to axis 1. (Optional)

Keywords:

'mnemonic'	Alternatively, the independent variable labeling the Fourier coefficients can be specified in the form *'mnemonic'=value*, where *'mnemonic'* is a valid axis mnemonic, and value is the maximum absolute value of this independent variable to be retained on return. If *'value'* is greater than the maximum value of this independent variable in *GyroData.potential*, then the additional Fourier coefficients will be set to zero. (Optional).
kmax	Set this keyword equal to the maximum absolute value of the Fourier transformed variable desired in 'self' on return.
NoAvg	Set this keyword (i.e., put '/NoAvg' on the command line) to remove the k=0 component of the Fourier transformed signal.
NoNegative	Set this keyword (i.e., put *'/NoNegative*' on the command line) to put zeros in the negative 'n' components. Default is to put the complex conjugate of the positive components in the negative 'n' components. (Optional).

Having completed the Fourier representation of the data contained within *GyroData.potential*, it can now be transformed into an configuration space (r,ζ) representation using the function *FFT* (see §4.11 below). Be sure to set the keyword *Inverse* as it requires an **inverse** transform to go from (r,n)-space to (r,ζ)-space.

A final note — the GKV convention for spatial fourier transforms [that the configuration space representation of a quantity is $A(\zeta)=A_n exp(+in\zeta)$] differs from that employed in the GYRO code [$A(\zeta)=A_n exp(-in\zeta)$]]. As a result, the positive sense of the toroidal angle in the GKV representation of the imported data is opposite to the positive sense of the toroidal angle within GYRO. Keep this in mind when working out which is the ion (electron) diamagentic direction!

4. Transforming the Representation of Data

Data is generally stored within microturbulence simulations in the most parsimonious possible manner because such a representation minimizes both the memory requirements and the computational effort. However, such parsimonious representations are often not the optimal representation for data analysis where memory is less of an issue. The methods described in this chapter allow conversion between several widely used data representations.

4.1 FFT

Usage:

result = object -> FFT(arg, keyword = values, …)

Transforms data between a configuration space and a Fourier representation in the specified independent variable. Normalization of the forward transform is chosen such that the signal *A*cos(2*pi*n*x/L)* will have an amplitude of *A* in the n^{th} Fourier harmonic (and zero elsewhere). *result* will contain both positive and negative *k*'s — a redundant representation for real data. For a more parsimonious Fourier representation, see *ktox* in §4.2 below.

Argument:	Any legal axis identifier. Defaults to the final axis. The independent variable may also be identified using an axis mnemonic. (Optional)

Input Keywords:

'mnemonic'	Where 'mnemonic' is the mnemonic for the independent variable over which the fourier transform is to be applied. 'Mnemonic' should be set equal to the range in this variable over which you wish to transform the data. Defaults to the final axis and current irange. (Optional)
Inverse	'Set' this keyword (i.e., put '/Inverse') to perform an inverse Fourier transform in. the indicated variable. Defaults to forward (i.e., configuration space to k-space) transform. (Optional)
Offset	Set this keyword to the (floating point) offset value which the specified independent variable will assume on completion of the inverse transform. Defaults to zero. (Optional)

4.2 FUNCTION XtoK

Usage:

result = Object -> XtoK()

This function method transforms the data in *Object* from a configuration space representation (like that output by *GTC, pg3eq,* and *TUBE*) into a Fourier space representation like that output by *GS2*, thus enabling convenient comparison with data from plasma micro turbulence simulation code which output data in a Fourier space representation (like *GS2*). If *Object* has three dimensions, then *XtoK a*ssumes that the first two dimensions correspond to configuration space values, while the third dimension is time. At each time-slice the configuration space representation of data in *Object* is converted to a k-space representation in *result.* This function method is the inverse of the function *KtoX* (see §4.3 below)

4.3 FUNCTION KtoX

Usage:

result = Object -> KtoX()

This function method transforms the data in *Object* from a Fourier space representation (like that output by *GS2*) into a configuration space representation, thus enabling convenient comparison with data from plasma micro turbulence simulation code which output data in a configuration space representation (like *GTC, pg3eq,* and *TUBE*). If *Object* has three dimensions, then *KtoX a*ssumes that the first two dimensions correspond to k-values, while the third dimension is time. At each time-slice the k-space representation of data in *Object* is converted to a configuration space representation in *result.* This function method is the inverse of the function method *XtoK* (see §4.2 above)

4.4 FUNCTION ROTATE

Usage:

result = Object -> Rotate(arg, keyword = value, …)

This function method rotates the data in 'self' through an angle specified by the argument 'arg'. If *Object* is a 3-D object, then the third independent variable is assumed to be time-like, and for each time-slice the data is rotated in the plane defined by the first two independent variables.

Argument: The argument to this function method is the desired (clockwise) angle of rotation in degrees.

Input Keywords:

x0	Value of first independent variable rotate about ('center of rotation'). Defaults to 0. (Optional)
'mnemonic1'_0	A synonym for x0 (see above), where 'mnemonic1' is the mnemonic for the first independent variable. (Optional)
y0	Value of second independent variable rotate about ('center of rotation'). Defaults to 0. (Optional)
'mnemonic2'_0	A synonym for y0 (see above), where 'mnemonic2' is the mnemonic for the first independent variable. (Optional)
title1	Set to an ascii string containing the title associated with the first independent variable. Defaults to 'x!D1!N' (that is, x_1 in Hershey vector font). (Optional)
mnemonic1	Set to an ascii string containing the mnemonic associated with the first independent variable. Defaults to 'x1'. (Optional)
units1	Set to an ascii string containing the units associated with the first independent variable. Defaults to input value of these units. (Optional)
title2	Set to an ascii string containing the title associated with the second independent variable. Defaults to 'x!D2!N' (that is, x_2 in Hershey vector font). (Optional)
mnemonic2	Set to an ascii string containing the mnemonic associated with the second independent variable. Defaults to 'x2'. (Optional)
units2	Set to an ascii string containing the units associated with the second independent variable. Defaults to input value of these units. (Optional)

Output KeyWords: None

5. Arithmetic on Objects

Once data has been inserted into IDL objects it is no longer possible to perform basic arithmetic operations on it using only IDL's command-line parser (*"only an objects methods can access an object's data"*—an edict which is strictly enforced by IDL in its object-oriented programming!). This functionality is recovered in GKV by providing METHODs to perform basic arithmetic operations. All four of these methods have common keywords as described below.

5.1 FUNCTION Plus

Usage:

result = object -> Plus(arg, keyword = values, …)

Defines basic arithmetic operation: addition. *Arg* can be a scalar; an array with the same dimensionality as *object*; another GKVsd object of the same dimensionality as *object* and having. the same independent variables; a 0-D GKVsd object; or a string containing the mnemonic of any of *object*'s independent variables. On return, *result* is a *GKVsd* object whose data values are formed by adding the data values of *object* to those of *arg*.

5.2 FUNCTION Minus

Usage:

result = object -> Minus(arg, keyword = values, …)

Defines basic arithmetic operation: subtraction. *Arg* can be a scalar; an array with the same dimensionality as *object*; another GKVsd object of the same dimensionality as *object* and having. the same independent variables; a 0-D GKVsd object; or a string containing the mnemonic of any of *object*'s independent variables. On return, *result* is a *GKVsd* object whose data values are formed by subtracting the data values of *arg* from those of *object*.

5.3 FUNCTION Times

Usage:

result = object -> Times(arg, keyword = values, …)

Defines basic arithmetic operation: multiplication. *Arg* can be a scalar; an array with the same dimensionality as *object*; another GKVsd object of the same dimensionality as *object* and having. the same independent variables; a 0-D GKVsd object; or a string containing the mnemonic of any of *object*'s independent variables. On return, *result* is a *GKVsd* object whose data values are formed by multiplying the data values of *object* by those of *arg*.

5.4 FUNCTION Over

Usage:

result = object -> Over(arg, keyword = values, …)

Defines basic arithmetic operation: division. *Arg* can be a scalar; an array with the same dimensionality as *object*; another GKVsd object of the same dimensionality as *object* and having. the same independent variables; a 0-D GKVsd object; or a string containing the mnemonic of any of *object*'s independent variables. On return, *result* is a *GKVsd* object whose data values are formed by dividing the data values of *object* by those of *arg*.

5.5 Arithmetic Keywords

All of these arithmetic operations share a common set of keywords which can be used to set the *title*, *mnemonic*, and *units* of the result.

Keywords:

Title	Set this keyword to a *string* containing the title of *result*. Hershey vector fonts can be used to get Greek or special characters. (Optional)
Mnemonic	Set this keyword to a *string* containing the mnemonic of *result*. Use only ascii characters (an nothing which *IDL* might interpret as a command delimiter). Be TERSE, as you may find yourself typing this *string* several times! (Optional)
Units	Set this keyword to a *string* containing the units of *result*. Hershey vector fonts can be used to get Greek or special characters. (Optional)

5.6 Derivatives of Objects — DbyD and D2byD

Usage:

result = Object -> DbyD(arg or keyword = value)

result = Object -> D2byD((arg or keyword = value)

The function methods *DbyD* (*D2byD)* returns a *GKVsd* object (of the same dimensionality as *object*) containing the first (second) partial derivative of *Object* with respect to the indicated independent variable. The algorithms employed provide a reasonable approximation to the partial derivative for BOTH uniform and non-uniform grids.

Argument: The (optional) argument is any legal axis identifier. That is, either an integer between 1 and nDims, or a *STRING* containing an axis mnemonic.

Keywords:

Axis — If no argument is provided, then this keyword may be used to identify independent variable with respect to which the partial derivative is to be taken. Set axis equal to any legal axis identifier (see discussion under *Argument* above).

mnemonic — Set the mnemonic of the selected axis equal to a two-element array, [min, max], to both identify the independent variable for which the (partial) derivative is to be taken, and to reset the signal window on this axis (before taking the derivative). This two-element array is interpreted as the desired *RANGE* in the independent variable, NOT the integer *'irange'*

irange — Set 'irange' to a two-element (integer) array to reset the signal window before taking the derivative *w.r.t.* the selected independent variable.

range — Set 'range' to a two-element (floating point) array to set the range in the independent variable over which the (partial) derivative is to be taken.

Side Effects:

If a 'range' or 'irange' is specified on the command line (either directly, or via 'mnemonic' = ...) then the *SignalWindow* method will be invoked on *'self'* and, on return, the signal window of the selected independent variable will have been modified.

5.7 Integrals of Objects — Int and CInt

Usage:

result = Object ->Int(arg or keyword = value)

result = Object ->CInt(arg or keyword = value)

The function method *Int* returns a *GKVsd* object with dimensionality one degree lower than that of *object'* containing the definite integral of 'self' over the selected independent variable; while the function method *CInt* returns a *GKVsd* object of the same dimensionality as *Object* containing the indefinite integral of *Object.* A "Simpson's Rule' algorithm is used which does not require that the data in *Object* be on a uniform grid in the selected independent variable.

Argument:	The (optional) argument is any legal axis identifier. That is, either an integer between 1 and nDims, or a STRING containing an axis mnemonic.

Keywords:

Axis	If no argument is provided, then this keyword may be used to identify independent variable to be integrated over. Set axis equal to any legal axis identifier (see above).
mnemonic	Set the mnemonic of the selected axis equal to a two-element array, [min, max], to both identify the independent variable to be integrated over, and to set the range of integration. This two-element array is interpreted as the desired RANGE in the independent variable, NOT the integer 'irange'
irange	Set 'irange' to a two-element (integer) array to set the range of integration through indices into the corresponding grid.values array.
range	Set 'range' to a two-element (floating point) array to set the range of integration.

Side Effects:

If a 'range' or 'irange' is specified on the command line (either directly, or via 'mnemonic' = ...) then the SignalWindow method will be invoked on 'self' and, on return, the signal window of the selected independent variable will have been modified.

5.8 Access to IDL Functions — GKVsd::Execute

Usage:

result = Object -> Execute(arg, arg1, arg2)

This method allows the user to apply any legal *IDL* function to the data contained in *Object*. It returns a *GKVsd* object of the same dimensionality as *Object* who data values are obtained by applying the specified function to the corresponding data value of *Object.*

Arguments:

arg	Accepts a *string* argument containing the name of an IDL function.
arg1, arg2	Up to two additional arguments can be passed to this IDL function by setting these (optional) arguments to *Execute*.

6. Basic Analysis I: Compressing Dimensionality

What computer visualization is mainly about is painting colors onto a 2-D screen. In plasma micro turbulence simulations we are blessed with (generally scalar) data as a function of four independent variables. Since it is not really convenient to display fields vs. more than two independent variables (I know that *Advanced Computer Visualization* is soon going to change all this, but bear with me …) it is generally necessary to compress the dimensionality before analyzing and/or displaying the data.

6.1 FUNCTION Slice

Usage:

result = object -> slice(keyword=value, …)

Object is sliced at the specified location, returning a *GKVsd* object of lower (by one) dimensionality.

Input Keywords:

Axis	Identifies the independent variable at the 'slice'. Set to a legal axis identifier—that is, an ascii string containing a valid axis mnemonic or an positive integer (no larger than the number of independent variables of this object). The axis can also be identified by a run-time keyword (see below). If no axis identifier is provided defaults to the final (usually 'time') axis. (Optional)
Value	Value of the independent variable at the 'slice'. This can also be set using run-time keywords (see below). Defaults to the first (presumably lowest) value of the selected independent variable. (Optional)
Index	Alternatively, the value of the independent variable at the 'slice' can be specified by providing an index into the array of grid values. Defaults to 0. (Optional)
Max	Set this keyword (that is, put */Max* on the command line) to 'slice' along the maxima of the dependent variable over the selected axis (implemented only for GKVs2D objects at present). Default is to slice at a selected value of the independent variable. (Optional).

MaxLocation — If the keyword Max (see above) has been set, then also setting MaxLocation (i.e., put */MaxLocation* on the command line) will cause *Slice* to return an anonymous structure with tags **slice** (corresponding to the *slice* from *object*) and **maxLocation** (corresponding to a GKVs1D object containing the location of the slice in the selected independent variable vs. the remaining independent variable). (Optional)

Run-time Keywords:

mnemonic — Put *mnemonic*=value on the command, where *mnemonic* is the mnemonic corresponding to one of the axes (independent variables) of *object* will cause *object* to be sliced on the corresponding axis at the specified value. (Optional).;

6.2 Function Avg

Usage:

result = Object -> Avg(arg)

or

result = Object -> Avg(keyword = value)

This function returns a *GKVsd* object with dimensionality one less than that of *Object* containing the average of *Object* over the selected independent variable.

Argument: The (optional) argument is any legal axis identifier. That is, either an integer between 1 and nDims (the dimensionality of *Object*), or a STRING containing an axis mnemonic

Keywords:

Axis — If no argument is provided, then this keyword may be used to identify independent variable over which the average is to be taken. Set axis equal to any legal axis identifier (see above).

mnemonic — Set the mnemonic of the selected axis equal to a two-element array, [min, max], to both identify the independent variable and the range in this variable over which the average is to be taken. This two-

element array is interpreted as the desired *RANGE* in the independent variable, NOT the integer *'irange'*. (Optional)

irange — Set 'irange' to a two-element (integer) array to reset the signal window before taking the average *w.r.t.* the selected independent variable. (Optional)

range — Set 'range' to a two-element (floating point) array to set the range in the independent variable over which the average is to be taken. (Optional)

Side Effects: If a *'range'* or *'irange'* is specified on the command line (either directly, or via *'mnemonic'* = ...) then the *SignalWindow* method will be invoked on *'Object'* and, on return, the signal window of the selected independent variable will have been modified.

6.3 FUNCTION Moments,

Usage:

result = object -> Moments(keyword = value, …)

Where *object* is a valid GKVsd object (of dimensionality 1 or greater) and *result* is a three-element object array. Default is to return the integral of the dependent variable over the selected axis in the 0^{th} element of the output array; the integral of the product of the dependent and the selected independent variable over this independent variable in the 1^{st} element of the output array; and the integral of the product of the dependent variable times the square of the deviations of the dependent variable from its mean in the 2^{nd} element of the output array. If the keyword *Avg* is set then these are replaced by the average value of the dependent variable; the mean value of the selected independent variable; and the standard deviation of the selected independent variable about this mean.

Keywords:

Axis — Identifies the independent variable over which the moments are to be taken. Set to a legal axis identifier—that is, an ascii string containing a valid axis mnemonic or a positive integer (no larger than the number of independent variables of this object). The axis can also be identified by a run-time keyword (see below). If no axis identifier is provided (either with this keyword, or by using a

run-time keyword) then an error message is printed and 0 is returned. (Optional)

Range — Set to a two-element floating point array (IDL will convert integer arrays to floating point) to set the range of the independent variable (in that variable's proper units) over which the moments are to be taken. Defaults to the current signal window for the selected independent variable. (Optional)

Irange — Alternatively, set *irange* to a two-element integer array to set the range of grid-indices for the independent variable over which the moments are to be taken. Defaults to the current signal window for the selected independent variable. (Optional)

Avg — Set this keyword (i.e., put */Avg* on the command line) to request this method to return the average over the selected axis, mean value of the selected dependent variable (using the dependent variable as the weighting function), and standard deviation of the independent variable about this mean instead of the first three moments. Default is to return first 0^{th} through 2^{nd} moments. (Optional)

Runtime Keywords:

mnemonic — Put *mnemonic=[xmin, xmax]* where *=[xmin, xmax]* is a two-element floating point array (IDL will automatically convert integer arrays) on the command line to simultaneously specify the axis over which the moments (or average) is to be taken and the range of the selected independent variable. See **Range** keyword above. (Optional)

6.4 FUNCTION Delta

Usage:

result = object -> delta(keywords)

Where *object* is a valid GKVsd object of dimensionality 1 or greater. The output (*result*) is a structure with two tags: *result .avg* contains the average of the dependent variable over the selected independent variable while *result .delta* contains the deviations from this average.

Keywords:

Axis	Identifies the independent variable over which the independent variable is to be decomposed into an average and deviations from its average. Set to a legal axis identifier—that is, an ascii string containing a valid axis mnemonic or a positive integer (no larger than the number of independent variables of this object). The axis can also be identified by a run-time keyword (see below). If no axis identifier is provided (either with this keyword, or by using a run-time keyword) then an error message is printed and 0 is returned. (Optional)
Range	Set to a two-element floating point array (IDL will convert integer arrays to floating point) to set the range of the independent variable (in that variable's proper units) over which the average and deviations are to be taken. Defaults to the current signal window for the selected independent variable. (Optional)
Irange	Alternatively, set *irange* to a two-element integer array to set the range of grid-indices for the independent variable over which the average and deviations are to be taken. Defaults to the current signal window for the selected independent variable. (Optional)

RunTime Keywords:

mnemonic	Put *mnemonic=[xmin, xmax]* where *[xmin, xmax]* is a two-element floating point array (IDL will automatically convert integer arrays) on the command line to simultaneously specify the axis over which the average and deviations are to be taken and the range of the selected independent variable. See **Range** keyword above to understand the usage for *[xmin, xmax]* . (Optional)

7. Basic Analysis II: Time Series Analysis

7.1 PROCEEDURE RMSNorm

Usage:

Object -> RMSNorm, keyword = value

This is a **proceedure** — that is it multiplies the dependent variable within *Object* by a constant such that RMS value of the dependent varialbe within the 'SignalWindow' of *Object* is equal to *rmsValue*.

Arguments: None

Keywords:

RMSValue Set this keyword equal to the desired rms value of signal. Defaults to 1.0. (Optional)

7.2 FUNCTION DeTrend

Usage:

Result = Object -> DeTrend(Order = iOrder)

This function acts on GKVs1D objects (only) and returns in a *Result* which contains the data from *Object* with the average, trend, etc removed. This is accomplished by making a least-squares-fit of the data in *Object* to a polynomial of order *iOrder*, and then subtracting this polynomial from the dependent variable in *Object* to obtain the dependent variable of *Result.*

7.3 FUNCTION XCORR

Usage:

result = object -> XCorr(keywords)

where *object* is a valid GKVsd object of dimensionality 1 through 3. Returns the auto (or cross) variance of the data. The auto (or cross) variance is computed using the

Blackman–Tukey fast Fourier transform algorithm. See any standard text on time series analysis for a discussion of data windows, etc.

Keywords:

Ref	Set this keyword equal to a valid GKV object of the same or lower dimensionality to compute the cross variance between the object being acted on and the reference object. If this keyword is not set, the autovariance will be computed.
No_Avg	Set this keyword (i.e., put */No_Avg* on the command line) to remove the average before computing the auto (or cross) variance.
Dw	Set this keyword to determine the length of the cosine roll-off in time (the data window) to be applied to the data. If DW is set to a real number, then it is interpreted the length of this roll off in the proper time units. If DW is set to an integer, then it is interpreted as the length in units of integer time-steps. Default data window is a cosine roll-off with a length equal to $1/10^{th}$ that of the input data.
Norm	Set this keyword to normalize the result such that the value of the auto-correlation function at zero lag is 1.0 — that is, to return the correlation function instead of the auto-variance. Note that this only works on auto-correlation functions (that is, when the keyword *Ref* has **not** been set!). Default is to compute the (auto-) variance. (Optional)
Hamming	Set this keyword (i.e., put */Hamming* on the command line) to specify a Hamming window as the data window.
Hanning	Set this keyword c to specify a Hanning window as the data window.
Double	Set this keyword (i.e., put */double* on the command line) to request double precision arithmetic.

7.4 FUNCTION XCORR0

Usage:

Result = Object -> XCORR0(Arg, keyword = value)

This function returns the auto or cross correlation between *Object* and the reference Object (see *Ref* keyword below) as a function of the specified independent variable of

Object' at zero lag (or displacement) in the remaining independent variable(s). The correlation function is obtained by averaging over the remaining variable(s), so the data should be homogeneous in these independent variables. This function is significantly faster than XCORR for large (generally *GKVs3D)* objects, and so is useful when only the correlation function at zero lag in all but one independent variable is required.

Argument:

Arg	Any legal axis identifier. This axis will become the independent variable of *Result.* Defaults to the first axis. The independent variable may also be identified using an Axis mnemonic as described below. (Optional)

Input Keywords:

'mnemonic'	Where 'mnemonic' is the mnemonic for the independent variable desired for the correlation function. 'Mnemonic' should be set equal to the reference value of this independent variable. Defaults to the initial value of this axis. (Optional)
Ref	Set this keyword to a GKVsd object of whose independent variable(s) are the same as the remaining independent variables of *Object* (that is , the independent variables **not** identified by *Arg* or by the *'mnemonic'* keyword. Defaults to first element of the data in *Object*. with respect to the selected independent variable. (Optional)
Norm	Set this keyword (i.e., put '/Norm' on the command line) to normalize the cross-correlation such that the maximum value is 1. Default is no normalization — that is, to return the cross-variance between *Object* and *Ref*. (Optional)

7.5 FUNCTION XSPECT

Usage:

result = object -> XSpect(keywords)

Where *object* is a valid GKVsd object of dimensionality 1 through 3. The output, *result*, is a GKVsd object of the same dimensionality containing the auto (or cross) spectrum. The auto (or cross) spectrum is computed using the Blackman–Tukey fast Fourier transform algorithm. See any standard text on time series analysis for a discussion of lag and data windows, etc.

Keywords:

Ref	Set this keyword equal to a valid GKV object of the same or lower dimensionality to compute the cross spectrum between the object being acted on and the reference object. If this keyword is not set, the autospectrum will be computed.
No_Avg	Set this keyword (i.e., put */No_Avg* on the command line) to remove the average and trend before computing the auto (or cross) spectrum.
Dw	Set this keyword to determine the length of the cosine roll-off in time (the data window) to be applied to the data. If DW is set to a real number, then it is interpreted the length of this roll off in the proper time units. If DW is set to an integer, then it is interpreted as the length in units of integer time-steps. Default data window is a cosine roll-off with a length equal to $1/10^{th}$ that of the input data.
LW	Set this keyword to determine the length of the cosine roll-off in time (the lag window) to be applied to the correlation function. If LW is set to a real number, then it is interpreted the length of this roll off in the proper time units. If LW is set to an integer, then it is interpreted as the length in units of integer time-steps. Default lag window is a cosine roll-off with a length equal to 1/2 of the correlation function (which is itself equal to the next power of 2 greater than the length of the data).
Hamming	Set this keyword (i.e., put */Hamming* on the command line) to specify a Hamming window for both lag and data windows.
Hanning	Set this keyword c to specify a Hanning window for both lag and data windows.
Double	Set this keyword (i.e., put */double* on the command line) to request double precision arithmetic.

7.6 Function Filter

Usage:

result = Object -> Filter(arg, Keyword = value, …)

This function returns a *GKVsd* object of the same dimensionality as *Object* containing the data values from *Object* acted on by a band-pass filter.

Arguments: Any legal axis identifier. Defaults to the final (generally time-like) axis. The independent variable may be identified using either an integer axis number, or by an Axis mnemonic. (Optional)

;Input Keywords:

'mnemonic' Where *'mnemonic*' is the mnemonic for the independent variable over which the filter is to be applied. *'Mnemonic'* should be set equal to the range in this variable over which you wish to have filtered data returned. Defaults to the final axis and the current *irange*. (Optional)

Omega_0 Central frequency of the digital filter. Defaults to zero. (Optional)

k_0 Central wavenumber of the digital filter. Defaults to zero. This keyword is really a synonym for 'Omega_0', and is included to maintain an intuitive interface when the axis in question corresponds to a spatial (rather than time-like) variable. (Optional).

dOmega Width of the filter in frequency. Defaults to 10/T, where 'T' is the length of the time interval over which data is available from *Object*. (Optional)

dk A synonym for dOmega--the width of the filter in wavenumber space which defaults to 10/L, where 'L is the length of the interval over which data is available from 'self'. (Optional)

dT An alternative means of specifying the filter width--if set, dT gives the width of the support of the

digital filter in the time-domain, corresponding to a frequency width of dOmega=2π/dT.

dL — A synonym for dT--the width of the support of the digital filter in the spatial-domain, corresponding to a frequency width of dk=2π/dT.

Output Keywords:

Filter — If this keyword is set equal to symbol on the command line then, on return, that symbol will refer to a *GKVs1D* object containing the frequency representation the filter applied.

7.7 Function BiSpect

Usage:

Output_Structure = Object -> Bispect(arg1, arg2, keyword = Value, …)

This function performs a bispectral analysis of the data in *Object*' (a GKVs1D object) with the data in arguments *'arg1'* and arg2'. It returns a structure (*Output_Structure)*)containing the biSpectrum; the biCoherence; the biCorrelation function; the spectral densities of *'Object'*, *'arg1'*, and 'arg2'; The auto-correlation functions of '*self*', '*arg1*', and '*arg2*'; and the length of the data and lag windows.

The algorithm used here is avoids being a memory hog by sequentially performing bispectral analyses over subsets of the data whose length is defined by the *Lag Window* (see description of the *LW* keyword below), and then averaging the result.

Arguments:

arg1 — A GKVs1D time series. Defaults to *'Object'*. (Optional)

arg2 — A GKVs1D time series. Defaults to *'Object'*. (Optional)

Keywords:

dw	The length of the data window in proper time units. Ideally, the data window should be chosen to shorter compared to the lag window, but longer than the characteristic period of oscillation of the data (Optional).
idw	The (integer) length of the data window in time steps. If 'idw' is set, its value over rides value given with 'dw'. Defaults to 1/100 of the length of the data series. (Optional).
lw	The length of the lagwindow in proper time units. The lag window should be chosen to be greater than the correlation time (which is easily determined using the function proceedure *XCORR*), but short compared to the length of the data set. It is important not to choose too long a lag window, as this controls the size of intermediate arrays and output objects. (Optional).
ilw	The (integer) length of the lag window. If 'ilw' is set, its value over rides the value given with 'lw'. Defaults to 1/10 of the length of the data series.
order	Order at which to detrend the data in *Object*, *arg1*, and *arg2* within each data subset before computing bicorrelations (0 to just remove average, 1 to remove average and trend, etc.). Defaults to 1. (Optional).
epsilon	Fractional decrement allowed in norm used to compute the bicoherence. Such a limit is required because the alternative leads to spurious divides by small numbers resulting from zeros in the fourier transform of the lag window. If uncorrected, this would result in unwanted, large-amplitude noise in the estimate of the bicoherence. Defaults to 0.0085, the ratio of the first subsidiary maximum in the FT of a Hanning window (used here as the lag window) to the maximum value. (Optional... and only experts should mess with this!).

Output:

The output structure contains the following tags:

Name	Ascii string used to form default directory name when *Output_Structure* is saved using *GKV_SaveStructure*. Defaults to 'Bispect'.
tauLags	A three-element, floating-point array containing the lag window specified for each of the three input objects, *Object, arg1, and arg2.* (note, it is possible, but **not** necessary to specify a separate lagwindow for each object).
tauData	A floating point scalar containing the length of the data window.
biSpectra	A *GKVs2D* object containing the biSpectrum of the input data.
biCoherence	A *GKVs2D* object containing the biCoherence among the input data.
biCorr	A *GKVs2D* object containing the bicorrelation function of the input data
spect0	The spectral density of *Object.*
spect1`	The Spectral density of *Arg1*.
spect2	The spectral density of *Arg2.*
corr0	The auto-correlation function of *Object.*
corr1	The auto-correlation function of *arg1*.
corr2	The auto-correlation function of *arg2.*

8. Analysis Protocols

While I am a great believer in interactive data analysis, it is often the case that you need to perform **exactly** the same analysis on data from many simulations runs — for example when comparing results of simulations from a parameter scan. This is facilitated by using *Analysis Protocols — GKV* scripts which perform a fixed suite of analysis routines, and store the result in an output structure. Several *Analysis Protocols* which I have found useful are described below.

8.1 Function TauCorrs

This Analysis Protocol is useful for analyzing 2-D plus time fluctuation data. Radial modes (zonal flows, etc) should be removed (see *delta* in §6.4) before applying this analysis protocol.

Usage:

result = Object -> TauCorrs(arg, keyword = values, …)

This function performs a correlation analysis of *Object* (which must be a *GKVs3D* object), and returns the results of this analysis in as an anonymous structure (the contents of the output structure are described below). This analysis assumes that there is one inhomogeneous independent variable (even in flux-tube micro turbulence simulation codes the radial coordinate should be considered as *inhomogeneous* due to the radial variations in the zonal flows) which must be identified (either with the optional argument, *arg*, or via a keyword). If the system is truly homogeneous, it is ***MUCH*** more efficient to simply compute the correlation function with Xcorr (using no reference object).

Input Argument:

TauCorr**s* will accept any legal axis identifier (that is, an integer between 1 and 3, or a valid axis mnemonic) as an argument. If no argument is provided, then ***TauCorrs will expect the axis corresponding to the inhomogeneous coordinate to be identified by keywords (see below). (Optional)

Input Keywords:

Axis

If no argument is provided, then this keyword may be used to identify the inhomogeneous coordinate. Set axis equal to any legal axis identifier (see above). (optional)

'mnemonic'	Set the *mnemonic* of the selected axis equal to a two-element array, [min, max], to both identify the selected independent variable, and reset the signal window on this axis. This two-element array is interpreted as the desired *RANGE* in the independent variable (i.e., it is interpreted in the units of the corresponding independent variable), NOT the integer *'irange'* (that is, NOT as an integer index into the grid.values array). (optional)
irange	Set 'irange' to a two-element (integer) array to reset the signal window of the selected independent variable. The value of irange is interpreted as an index into the grid.values array. 'irange' defaults to the current signal window of 'self. (optional)
range	Set 'range' to a two-element (floating point) array to set the range in the independent variable. (optional)
skip	The sampling interval for the specified inhomogeneous independent variable. Defaults to 1 (Optional).
localNorm	Set this keyword (i.e., put '/LocalNorm' on the command line) to normalize the data from 'self' (but not 'self' itself, whose data remains unaltered) such that the rms fluctuation at each value of the specified inhomogeneous independent variable is the same BEFORE computing the correlation time. (Optional)
debug	Set this keyword (i.e., put '/debug' on the command line) to print out intermediate information, allowing the user to keep track of the progress (or lack thereof...) of this function. (optional)
fraction	Normally ***TauCorrs*** returns the full width at half maximum of the correlation function vs. the specified axis. If this keyword is set, ***TauCorrs*** will instead compute the full width at 'fraction' of maximum vs. time. Defaults to 0.5 (Optional).

Output: ***TauCorrs*** returns a structure with the following tags. Additional tags can be included by setting

(optional) output keywords (see "Output Keywords" below).

TauCorr — A *GKVs1D* object containing the correlation time (full-width at half-maximum measured along the maximum of the local correlation function) vs. the specified inhomogeneous independent variable.

vPhase — A *GKVs1D* object containing the phase velocity (slope of the maximum in the local correlation function) vs. the inhomogeneous independent variable.

yCorr — A *GKVs1D* object containing the correlation length in the remaining independent variable (full-width at half maximum of the Hilbert-transform 'envelope' of the oscillatory local correlation function evaluated at zero time lag) vs. the inhomogeneous variable.

kAvg — A *GKVs1D* object containing the power-weighted average wave number in the remaining (presumably homogeneous) independent variable vs. the inhomogeneous variable. The error bars on kAvg represent the power-weighted standard deviation about this mean wave number.

kSpect — A *GKVs1D* object containing the average (over the inhomogeneous variable) of the local frequency-integrated spectrum vs. the wave number associated with the remaining independent variable. If the */LocalNorm* keyword is set, then this is a volume average, while if it is not set, then this is a power-weighted volume average.

CorrFcn — A *GKVs1D* object containing the (volume or power-weighted volume as with kSpect above) average over the inhomogeneous coordinate of the maximum of the local correlation function vs. the time lag.

yCorrFcn — A *GKVs1D* object containing the (volume or power-weighted volume as with *kSpect* above) average over the inhomogeneous coordinate of the local correlation function vs. the remaining independent variable evaluated at zero time lag.

Output Keywords:

CorrFcns	Set this keyword (i.e., put "*/CorrFcns*" on the command line) to add the tag "*TauCorrArr*" to the output structure. The corresponding value is an array of *GKVs1D* objects containing the local correlation function vs. tau for each location at which the correlation time is computed. (Optional)
yCorrFcns	Set this keyword (i.e., put "*/yCorrFcns*" on the command line) to add the tag "*yCorrArr*" to the output structure. The corresponding value is an array of *GKVs1D* objects containing the local correlation function vs. the homogeneous spatial coordinate at each radial location at which the correlation function in computed. (Optional)
kSpects	Set this keyword (i.e., put "*/kSpects*" on the command line) to add the tag "*kSpectArr*" to the output structure. The corresponding value is an array of *GKVs1D* objects containing the local k-spectrum (integrated over frequency) vs. the homogeneous spatial coordinate at each radial location at which the correlation function in computed. (Optional)
Side Effects:	Resets the signal window of *Object* to specified *'range'* or *'irange'* if one is provided with any of the keywords *'mnemonic'*, *irange*, or *range*.

8.2 AnalysisProtocol

This analysis protocol was designed for the analysis of TEM parameter scans from GS2. It assumes that there is a user present (and paying attention. It performs the analysis described in *TauCorrs* above (see §8.1) and also analyzes the radial modes (zonal flows and GAMs).

Usage:

Result = AnalysisProtocol(keyword = value, …)

In addition to all the keywords described in *TauCorrs* above, *AnalysisProtocol* accepts the following

Keywords

Data	Set this keyword to a structure containing the GS2 data (as returned by *NetCDF_DATA*). If this keyword is not present (or, does not point to a structure), then *NetCDF_DATA* will be initialized, and the user must choose the appropriate NetCDF file. (Optional)
Path	Set this keyword to a string variable containing the *Path* where *NetCDF_DATA* should begin its search for the appropriate GS2 data file. If not present, then the search will start at the current working directory. (Optional)
File	Set this keyword to a string variable containing the full specification of the NetCDF file containing the GS2 data. If not present, a widget will pop up so that the user can select the appropriate file. (Optional)
RunID	Set this keyword to a string varaible containing an appropriate *RunID* for the *NeteCDF* data file to be read. (Optional)
TRange	Set this keyword to a two-element floating-point array containing the time interval over which the data is to be analyzed. If not present, a widget will pop up, allowing the user to select an appropriate time interval. (Optional)
Save	Set this keyword (i.e., put *'/Save'* on the command line) to save the output of this Analysis Protocol to disk using GKV_SaveStructure (see §11.8 below).

8.3 LinearModes

This routine is useful for performing linear analysis from of data with one homogeneous varialbe (which will be Fourier transformed over) and one inhomogeneous variable. It expects data from the linear phase of the simulation in the input (GKVs3D) object. It returns a structure containing real frequency and growth rate of the of fastest growing mode for each Fourier harmonic, together with an object array containing the (best effort) at extracting the corresponding linear mode structures.

Usage:

result = Object -> LinearModes(argument, keyword = value, …)

Argument:	The argument is any legal axis identifier. It should identify the axis which is to be fourier transformed over to separate spatial structure of the linear eigenmodes. Defaults to axis 2.
Keywords:	
xRef	Set this keyword to an appropriate reference value of the inhomogeneous variable. This should be a location where the linear eigenmodes are expected to have a significant value. Defaults to 0. (Optional)
‘mnemonic’	Alternatively, this reference location can be set by using the syntax ‘*mnemonic’=value*, where ‘*mnemonic’* is the mnemonic for the inhomogeneous axis, and *value* is the desired reference value.

9. Outputting Results

9.1 View

Usage:

object -> view, arg0, arg1, …, keyword = value, …

VIEW is *GKV*'S basic tool for viewing objects and outputting results. Upon invoking *VIEW* a resizable graphics widget will pop up displaying a plot of "*object*". The menu allows the user to change the color table, or output the plot in a variety of graphics formats.

Arguments; *VIEW* only accepts arguments (as opposed to keywords) when *obj* is a *GKVs2D* object (that is, a 2-D *GKVsd* object). In this case, the a line drawing of a curve is displayed. Any arguments provided should be either *GKVs2D* objects or arrays of *GKVs2D* objects which will be displayed on the same frame as *obj* (but in different colors).

Keywords: Any of the keywords described under "*DRAW*" below, or any valid *IDL* graphics keyword can be employed with *VIEW*. In addition, the user may employ the keyword:

Shade_Surf For *GKVs2D* or *GKVs3D* objects the default plot is an *image* with colors chosen according the local data values (in *GKVs3D* objects only one time slice is shown at a time). If this keyword is set (i.e., */Shade_Surf* is put on the command line), then a surface plot is displayed instead of an *image* (essentially, a color contour plot).

9.2 GKVs1D::Draw

Usage:

object -> Draw, keywords

Where *object* is a valid GKVs1D object. "Draws" *object* in the currently open window, producing a line plot for 1-D objects.

Keywords:

In addition to the usual IDL graphics keywords (see IDL Reference manual for a discussion of IDL graphics keywords), there are the following:

Pretty	Set this keyword (i.e., put */Pretty* on the command line) to use Hershey vector fonts to make 'pretty' titles and labels. Default is to just use ascii characters.
Log	Set this keyword (i.e., put */Log* on the command line) to make a plot which is logarithmic in the dependent variable.
Indx1	When *Drawing* a GKVsd object of dimensionality greater than two set this keyword (and its companion, Indx2) to set values at which the remaining independent variables should be 'sliced'.

Runtime Keywords:

mnemonic	Set this keyword to a two-element floating point array (IDL will automatically convert integer arrays) to specify the plotting range in the specified independent variable.

9. 3 PRO GKVs1D::oPlot

Usage:

Object -> oplot, keywords

Where *Object* is a valid GKVs1D objects. Plots *Object* over the image on the currently open graphics window.

Keywords:

In addition to any IDL graphics keywords that are accepted by the IDL routine OPLOT, the following keyword is available.

RealOnly	Set this keyword (i.e., put */RealOnly* on the command line) to plot only the real part of a complex object. Default is to plot the real part as a solid line and the imaginary part as a dashed line. (Optional)

9.4 Pro GKVs2D::Draw

Usage:

Object -> Draw, keywords

Where *Object* is a valid GKVsd object of two or more dimensions. Produces an "image" plot (essentially a color contour plot) in the currently open graphics window.

Keywords:

In addition to the usual IDL graphics keywords (see IDL Reference manual for a discussion of IDL graphics keywords), there are the following:

Pretty	Set this keyword (i.e., put */Pretty* on the command line) to use Hershey vector fonts to make 'pretty' titles and labels. Default is to just use ascii characters. (Optional)
Polar	Set this keyword (i.e., put */Polar* on the command line) to make a polar plot. The first independent variable will be interpreted as the radius, while the second will be interpreted as the angle.
VRange	Set this keyword to a two-element floating point array (IDL will automatically convert integer arrays) to set the axis range for the dependent variable. (Optional)
Log	Set this keyword (i.e., put */Log* on the command line) to make a plot which is logarithmic in the dependent variable. (Optional)
Grid1	Set to any valid GKV axis identifier (an integer or an axis mnemonic) to specify the first ('x') axis for this plot. Defaults to the first independent variable. (Optional)
Grid2	Set to any valid GKV axis identifier (an integer or an axis mnemonic) to specify the second ('y') axis for this plot. Note that if Grid2 is specified, it MUST correspond to an axis whose (integer) identifier is greater than that of Grid1 (user the INFO method to determine integer axis identifiers). Defaults to the second independent variable. (Optional)
Indx1	Set this keyword (and its companion, Indx2) to set values at which the remaining independent variables to determine where the dependent variable should be 'sliced'. (Optional)

Runtime Keywords:

mnemonic Set this keyword to a two-element floating point array (IDL will automatically convert integer arrays) to specify the plotting range in the specified independent variable. (Optional)

OR

Set this keyword to a scalar to specify the value at which the third (or fourth) independent variable should be 'sliced' for this plot. (Optional)

9.5 Pro Shade_Surf

Usage:

Object ->Shade_Surf, keywords

Where *Object* is a valid *GKVsd* object of two or more dimensions. Produces a surface plot of the dependent variable in the currently open graphics window.

Keywords:

In addition to the usual IDL graphics keywords (see IDL Reference manual for a discussion of IDL graphics keywords), there are the following:

Pretty Set this keyword (i.e., put */Pretty* on the command line) to use Hershey vector fonts to make 'pretty' titles and labels. Default is to just use ascii characters. (Optional)

Polar Set this keyword (i.e., put */Polar* on the command line) to make a polar plot. The first independent variable will be interpreted as the radius, while the second will be interpreted as the angle.

PhaseShading For surface plots of complex data, unset this keyword (i.e., put *PhaseShading=0* on the command line) to turn off "phase shading" (the default for complex data). With phase shading off the surface is colored according to a scheme relating to the 'lighting', while the height is proportional to the real part of the dependent variable. When phase shading is turned on, the surface color is chosen according to the cosine of the phase of the complex dependent variable, while the height of the surface is proportional to its absolute value.

VRange — Set this keyword to a two-element floating point array (IDL will automatically convert integer arrays) to set the axis range for the dependent variable. (Optional)

Log — Set this keyword (i.e., put */Log* on the command line) to make a plot which is logarithmic in the dependent variable. (Optional)

Grid1 — Set to any valid GKV axis identifier (an integer or an axis mnemonic) to specify the first ('x') axis for this plot. Defaults to the first independent variable. (Optional)

Grid2 — Set to any valid GKV axis identifier (an integer or an axis mnemonic) to specify the second ('y') axis for this plot. Note that if Grid2 is specified, it MUST correspond to an axis whose (integer) identifier is greater than that of Grid1 (user the INFO method to determine integer axis identifiers). Defaults to the second independent variable. (Optional)

Indx1 — Set this keyword (and its companion, Indx2) to set values at which the remaining independent variables to determine where the dependent variable should be 'sliced'. (Optional)

Runtime Keywords:

mnemonic — Set this keyword to a two-element floating point array (IDL will automatically convert integer arrays) to specify the plotting range in the specified independent variable. (Optional)

OR

Set this keyword to a scalar to specify the value at which the third (or fourth) independent variable should be 'sliced' for this plot. (Optional)

9.6 Outputting Objects to a TIFF file

Many objects can be written into a TIFF file using the proceedure GKV_TiffOut.

Usage:

GKV_TiffOut, arg0, arg1, arg2, …, keyword = value, …

This routine accepts GKV objects, GKV object arrays, or structures containing GKV objects as input. It produces a Tiff output file which displays each of these GKV objects.

Arguments: GKV_TiffOut accepts up to 10 arguments, each of which is either a GKVsd object, an array of GKVsd objects, or a structure (any GKVsd objects in these structure will be displayed in the TIFF file produced by this command).

Keywords:

Pack — Setting this keyword (i.e., putting '/pack/ on the command line) will result in 4 plots per page in the output TIFF file. Default is one frame per page. (Optional) **NOT YET IMPLEMENTED**

FileName — Set this keyword to an ascii string containing the name of the desired output file. (Optional). Defaults:

If the argument is a single GKV object, then the fileName defaults to 'mnemonic'.tif, where 'mnemonic' is the mnemonic of the GKV object.

If the argument is a structure which includes the tag "Name", then the outfile name defaults to 'Name'.tif, where 'Name' is the (ascii) contents of the tag "Name".

If no "fileName keyword is supplied, and the structure argument does not have a tag "Name", then the FileName defaults to "GKV.tif".

Path — Set this keyword to the path to the directory where the TIFF file is to be stored. Defaults to the current working directory. (Optional)

Append — Set this keyword (i.e., put "/Append" on the command line) when calling GKV_TIffOut recursively to indicate that frames are to be added to an existing TIFF file.

xSize — x-dimension (in pixels) of resulting image. Defaults to 400. (Optional)

ySize — y-dimension (in pixels) of resulting image. Defaults to 400. (Optional)

10. Producing Animations

A convenient and rapid way of producing an animation is to use:

10.1 Movie

Usage:

obj -> Movie, arg1, arg2, …, keywords = value, …

This procedure creates and displays an animation of the data contained in *obj* using the *XINTERANIMATE* tool provided as part of *IDL*. This tool also optionally allows you to produce and save MPEG files containing this animation (NOTE: you must request the MPEG license from RSI when you upgrade to *IDL* Version 5.4. This license doesn't seem to cost extra, but only appears on request …).

Arguments:

MOVIE only accepts arguments (as opposed to keywords) when *obj* is a *GKVs2D* object (that is, a 2-D *GKVsd* object). In this case, the animation is a line drawing of a curve which will evolve in time. Any arguments provided should be either *GKVs2D* objects or arrays of *GKVs2D* objects which will be displayed (and evolve) on the same frame as *obj*.

KeyWords:

imin — Index within *'self'* to the first frame of the animation. Defaults to the first time-slice of 'self'. (Optional)

imax — Index within 'self' to the final frame of the animation. Defaults to the final time-slice of 'self'. (Optional)

'Mnemonic' — Alternatively, you can specify the time-interval for the animation in the format "Mnemonic = [start, end]", where "Mnemonic" is the mnemonic for the time-like (third) axis, while 'start' and 'end' are the values of the time-like coordinate at the stare and end of the animation.

skip — The animation uses only "iskip"th time-slice. Defaults to the smallest integer such that the total number of frames in the animation is ≤ 128. NOTE: on many (but not LINUX) platforms IDL appears to limit the total number of PIXMAP windows (one PIXMAP window is required for each frame of the animation) to 128, so the user's choice of

iskip is overridden if it would result in more than 128 frame in the animation. (Optional)

npixels — The number of pixels to be used in each frame of the animation. If npixels is set to a scalar, a square window of (npixels x npixels) is produced. If npixels is set to a two-element array, then the a (generally) rectangular window of (nPixels[0] x nPixels[1]) is produced. Defaults to 400x400. (Optional)

Shade_Surf — The default plot is an *image* with colors chosen according the local data values. If this keyword is set (i.e., */Shade_Surf* is put on the command line), then a surface plot is animated instead of an *image* (essentially, a color contour plot).

Graphics Keywords: Any additional keywords specified on the command line will forwarded to the plotting routines, allowing the user to customize his animation.

The *XINTERANIMATE* tool is limited on many platforms as to the maximum number of frames, and provides little control of the quality of the animation. Hence, there is a need of alternative means of producing animations.

10.2 MPeg,

Usage:

Object -> MPEG, keyword = value, …

;
This method makes an MPeg animation from a sequence of images from slices of 'self' at sequence of values the independent variable stored in self.Grid3 (this is generally the *unlimited*, or *time* variable). It creates an image of every iSkipth time slice of 'self', and stores the result into a file selected by the user. This can take awhile, so probably you'll want to get a nice cup of coffee while you're waiting for *IDL* to finish making the MGPEG animation. NOTE, this procedure will ONLY work with *IDL* Version 5.4 (or higher).
;
Input KeyWords:

iSkip — Interval between timeslices to be animated. Defaults to 1 (Optional).

trange	A two-element array specifying the first and last time-slices of this animation. Defaults to self.Grid3.range (Optional).
'mnemonic'	The mnemonic of self.Grid3 can be used in place of the keyword 'trange' with the same effect. Defaults to self.Grid3.range (Optional).
Shade_Surf	Set this KeyWord (i.e., '/Shade_Surf') to make a surface plot instead of an image. Defaults to making 'image' plots (Optional)
Path	Set to path to folder where the MPeg animation is to stored. Defaults to the current working directory (Optional).
FileName	Name of the file containing the MPeg animation. Defaults to 'self.mnemonic'.mpg. (Optional)
Xsize, Ysize	The size of the frame in 'device' pixels. Defaults to xsize = 500, ysize = 500. (Optional)
Quality	Set this keyword to an integer value between 0 (low quality) and 100 (high quality) inclusive to specify the quality at which the MPEG stream is to be stored. Higher quality values result in lower rates of time compression and less motion prediction which provide higher quality MPEGs but with substantially larger file size. Lower quality factors may result in longer MPEG generation times. The default is 50. (Optional)
ShowLoad	Set this keyword to view images as they are loaded into the MPeg file. Default is not to show images. (Optional)

10.3 Jpegs

Usage:

Object -> JPEGS, keyword = value, …

Utilities are available which can convert a sequency of images contained in separate files (this works MUCH better than trying to put all the images in the same file — believe me, as I tried that!) into animations. You are responsible for finding such a utility (e.g., the shareware utility GraphicCoverter for MACs). I’ve contented my self with providing convenient means within *GKV* of producing a directory full of image files.

Input KeyWords:

;iSkip	Interval between timeslices to be animated. Defaults to 1 (Optional).
trange	A two-element array specifying the first and last time-slices of this animation. Defaults to self.Grid3.range (Optional).
'mnemonic'	The mnemonic of self.Grid3 can be used in place of the keyword 'trange' with the same effect. Defaults to self.Grid3.range. (Optional).
Shade_Surf	Set this KeyWord (i.e., '/Shade_Surf') to make a surface plot instead of an image. Defaults to making 'image' plots (Optional)
Path	Set to path to folder where a new folder containing the sequence of JPEG files is to be stored. Defaults to the current working directory (Optional).
DirectoryName	Name of the new folder to be created to contain the sequence of JPEG files. Defaults to "'Self.mnemonic'_Animation". (Optional)
FileRoot	Sequence of JPEG files will have the names FileRoot00000.jpg, FILERoot00001.jpg, ... Defaults to self.mnemonic. (Optional)
Xsize, Ysize	The size of the frame in 'device' pixels. Defaults to xsize = 600, ysize = 700. (Optional)
Quality	Sets the quality index in the range of 0 ("terrible") to 100 ("excellent") of the JPEG images to be produced. The default is 80 to produce "pretty good" quality. Lower values of "Quality" produce higher compression ratios and smaller files. Default is Quality = 80. (Optional).
true	This keyword specifies the index, starting at 1, of the dimension over which the color is interleaved. For example, for an image that is pixel interleaved and has dimensions of (3, m, n), set TRUE to 1. Specify 2 for row-interleaved images (m, 3, n); and 3 for band-interleaved images (m, n, 3). If this keyword is not set, then WRITE_JPEG is called with true=1. You probably won't have to worry about this, as I believe *true* defaults to an appropriate value. (Optional)

ShowLoad	Set this keyword to view images as they are loaded into the jpeg file. Default is not to show images. (Optional)

10.4 TIFFs

Usage:

Object -> TIFFs, keyword = value, …

This routine is used to provide a sequence of TIFF images for an animation of 'self' over the independent variable stored in self.Grid3. It creates a TIFF image of every iSkipth timeslice of 'self', and stores the result into a sequence of files in a directory selected by the user. These files can be used to produce animations as described in *GKVs3D::JPEGS* above.

Input KeyWords:

iSkip	Interval between timeslices to be animated. Defaults to 1 (Optional).
trange	A two-element array specifying the first and last time-slices of this animation. Defaults to self.Grid3.range (Optional).
'mnemonic'	The mnemonic of self.Grid3 can be used in place of the keyword 'trange' with the same effect. Defaults to self.Grid3.range. (Optional).
Shade_Surf	Set this KeyWord (i.e., '/Shade_Surf') to make a surface plot instead of an image. Defaults to making 'image' plots (Optional)
Path	Set to path to folder where a new folder containing the sequence of TIFF files is to be stored. Defaults to the current working directory (Optional).
DirectoryName	Name of the new folder to be created to contain the sequence of TIFF files. Defaults to "TIFF_Animation". (Optional)
FileRoot	Sequence of TIFF files will have the names FileRoot00000.tif, FILERoot00001.tif, ... Defaults to self.mnemonic. (Optional)

Xsize, Ysize	The size of the frame in 'device' pixels. Defaults to xsize = 500, ysize = 500. (Optional)
ShowLoad	Set this keyword to view images as they are loaded into the TIFF file. Default is not to show images. (Optional)

11. Memory Management

11.1 FUNCTION MakeCopy

Usage:

result = object -> MakeCopy

Makes a 'deep copy' of *object* and places it into *result*. This differs from a simple assignment statement (*result = object*), which only gives a second name (*result*) to the data stored in *object*. Any action taken on *result* also modifies *object* because they both point to the **same** data. *MakeCopy* is an implementation of a 'deep copy' in which *result* points to an additional copy of the *object* data which has been created in memory. This allows you to act on *result* without acting on *object*.

11.2 PRO Trash

Usage:

object -> Trash

Deletes all data associated with *object*, thereby freeing up memory. Note that *Trash* does **not** remove *object* from IDL's symbol table (that is, from IDL's Variable Watch window). Once an object's data has been deleted with *Trash*, it can be removed from the symbol table with the IDL procedure **DELVAR** (Usage: *DELVAR, object* . See the IDL Reference manual for more info on DELVAR). Simply using DELVAR to remove an object from IDL's symbol table (without using *trash*) does not free up any of IDL's memory.

11.3 PRO GKVdelete, arg

Usage:

GKVdelete, arg

where *arg* is a variable, array (including arrays of GKVsd objects), or a structure (including structures which contain GKVsd objects). *GKVdelete* deletes all data associated with *arg* freeing up the memory. Like *Trash*, this will not remove *arg* from IDL's symbol table (which must be done with **DELVAR**).

11.4 Pro GKVsd::Save

Keywords:

FileName	The name of the file in which the GKVsd object is to be saved. Defaults to *mnemonic*.gkv, where *mnemonic* is the mnemonic of the GKVsd object being saved. (Optional)

Usage:

object -> Save, FileName=filename

Saves a GKVsd object to disk using IDL's XDR format (see **SAVE** in the IDL Reference manual for more information on XDR, but note that IDL's save routine is NOT a method on GKVsd objects and you cannot expect the keywords of IDL's save routine to work in this context).

11.5 FUNCTION GKV_RESTORE

Usage:

result = GKV_Restore(FileName=filename)

This procedure restores a GKVsd object which has previously been *Save*d (with *GKVsd::Save*) into *filename*. The restored GKVsd object is stored in *result*.

Keywords:

FileName	The name of the file in which a GKVsd was saved. If FileName is not provide, DIALOGUE_PICKFILE will allow the user to select an appropriate file. (Optional)

11.6 Procedure GKV_SaveArray

Usage:

GKV_SaveArray, ObjArr

This procedure saves the array of *GKVsd* objects specified by , *ObjArr* to disk. It does this by first creating a new directory ("Directoryname_arr") and then writing separate *.gkv* file (see ***GKVsd::Save***) with names FamilyName.*index*.gkv, where '*index*' is the

zero-based array index into *ObjArr*. Unfortunately, this procedure will only work with *IDL* Version 5.4 (or greater).

Input Argument: The (required) input argument, *ObjArr* must be an array of *GKVsd* objects, which will be saved to disk by this procedure.

Input Keywords

Path	Path to working directory in which the new directory, "DirectoryName_arr", containing the '.gkv' save files will be created. Defaults to current working directory. (Optional)
DirectoryName	Name of the new directory to be created within the directory specified by "Path". Defaults to the mnemonic of *ObjArr[0]* (or, "*GKVsd_arr*", if ObjArr[0] has no mnemonic). (Optional)
FamilyName	Family name of the sequence of .gkv files. Defaults to the mnemonic of *ObjArr[0]* (or, "*GKVsd*", if ObjArr[0] has no mnemonic). (Optional)
Debug	Set this keyword (i.e., put "*/Debug*" on the commandline) to print out intermediate information which may be useful in debugging this procedure

11.7 Function GKV_RestoreArray

Usage:

ObjArr = GKV_RestoreArray(keyword = value, …)

This function returns an array of GKVsd objects which have previously been saved to disk using GKV_SaveArray (see §11.6 above).

Input Keywords

Path	Path to working directory in which the new directory, "DirectoryName", containing the '.gkv' save files were created. Defaults to current working
DirectoryName	Name of the directory (within the working directory specified by 'Path') containing the '.gkv' save files.

If not specified the user will pick DirectoryName with "Dialog_Pickfile". (Optional)

FamilyName — The Save files should have a name of the form 'FamilyName'.'index'.gkv. If not specified *GKV* will extract 'FamilyName' from the first file returned by "FINDFILE". (Optional)

Debug — Set this keyword (i.e., put "/Debug" on the commandline) to print out intermediate information which may be useful in debugging this procedure

11.8 Procedure GKV_SaveStructure

Usage:

GKV_SaveStructure, structure

;This procedure save a structure (possibly containing *GKVsd* objects) onto to disk. It does this by first creating a new directory ("Directoryname_str") and then writing separate SAVE files using ***GKVsd::Sav***e, ***GKV_SaveArray***, and the native *IDL* SAVE routine as appropriate

;Input Argument: The (required) input argument, *structure* must bean anonymous structure which may contain *GKVsd* objects and/or *GKVsd* object arrays

Input Keywords

Path — Path to working directory in which the new directory, "DirectoryName_str", containing the '.gkv' save files will be created. Defaults to current working directory. (Optional)

DirectoryName — Name of the new directory to be created within the directory specified by "Path". Defaults to the contents of *structure.name* with *"_str"* appended if such a tag exists within *structure*, If no such tag exists, then default is *GKVsd_str*. (Optional)

Debug — Set this keyword (i.e., put "/Debug" on the commandline) to print out intermediate information which may be useful in debugging this procedure

11.9 FUNCTION GKV_RestoreStructure

Usage:

structure = GKV_RestoreStructure(keyword = values, …)

This function returns a structure (possibly containing *GKVsd* objects or arrays of *GKVsd* objects) which has previously been saved to disk using ***GKV_SaveStructure*** (see §11.8 above)

Input Keywords

Path	Path to working directory in which the new directory, "DirectoryName", containing the various save files and subdirectories was created. Defaults to current working directory. (Optional)
DirectoryName	Name of the directory within the working directory specified by 'Path' containing the various save files and subdirectories. If not specified, the user will pick DirectoryName with "Dialog_Pickfile". (Optional)
Debug	Set this keyword (i.e., put "/Debug" on the commandline) to print out intermediate information which may be useful in debugging this procedure

11. Basic GKVsd Object Management

11.1 Procedure Info

Usage:

object -> info

Prints out information about *object* to the IDL log window. This includes the object's mnemonic, title, indices, units, range of values, CodeName, CodePI, RunID, and FileID. For each of the object's independent variables the corresponding mnemonic, title, units, boundary condition, uniformity (of the grid), range, and signal window (irange) are printed out. All this can be helpful if you don't recall current state of a particular GKVsd object.

11.2 Function Cat

Usage:

result = Object -> CAT(arg1, arg2, arg3, …)

This function, which accepts up to 10 arguments, concatenates its argument(s) with *Object* and returns the result. *Object* is left unaltered. It returns a *GKVsd* object of the same dimensionality as *Object* in which the data and grid (independent variable) values from *'arg1' & Co*. have been concatenated behind those of 'self'. Concatenation is done *ONLY* in the last (generally, time-like) argument.

Input Arguments: Cat is generally called with at least one argument, which must be a *GKVsd* object of the same dimensionality as *Object*. If *CAT* is called with no argument, or with inappropriate arguments, then CAT returns a (deep) copy of *Object*.

11.3 FUNCTION SubSample

Usage:

Result = Object -> SubSample(Arg, keyword = value, …)

This function returns a GKVsd object of the same dimensionality as *Object* in which the data of *Object* has been 'subSampled' onto a uniform grid of 'nSteps' (see keyword descriptions below) over the selected independent varaible.

Argument:

Arg	The (optional) argument is any legal axis identifier. That is, either an integer between 1 and nDims, or a STRING containing an axis mnemonic.

Keywords:

Axis	If no argument is provided, then this keyword may be used to identify independent variable to be subsampled. Set axis equal to any legal axis identifier (see above).
mnemonic	Set the mnemonic of the selected axis equal to a two-element array, [min, max], to both identify the independent variable to be subsampled, and to reset the signal window on this axis (before subsampling). This two-element array is interpreted as the desired RANGE in the independent variable, NOT the integer 'irange'
irange	Set 'irange' to a two-element (integer) array to reset the signal window before subsampling on the selected independent variable.
range	Set 'range' to a two-element (floating point) array to set the range in the independent variable over subsampling is performed
nSteps	The desired number of gridpoint for the selected axis. Defaults to no change in number of grid points over *Object's* SignalWindow. (Optional).
DownBy	Decrease number of grid points by a factor of 'Downby'. Defaults to no change in number of grid points in *Object'* SignalWindow. (Optional).
Dt	Desired size of the uniform sampling interval in the output grid. Defaults to no change in number of grid points in *Object's* SignalWindow. (Optional).

11.4 GKVs1D::Squash

Usage:

result = Object -> Squash(arg)

This function method returns a *GKVs1D* object with the data of *Object* as the dependent variable, and the data of *arg* as the independent variable. Plots of *result* can be used to check for relationships between data objects.

Argument;	*arg* must be a *GKVsd* object of the same dimensionality as *Object*, with the same independent variables.

Input Keywords

Title	Should be set equal to a *String*, which becomes the *Title* for output object. Defaults to title of *Object*. (Optional)
Mnemonic	Should be set equal to a *String*, which becomes the *Mnemonic* for output object. Defaults to mnemonic of *Object*. (Optional)

11.5 PRO SignalWindow,

Keywords:

Axis	Set to any legal axis identifier—either an integer between 1 and the dimensionality of the object to be acted on, or the mnemonic of one of the independent variables.
range	Set to a two-element floating point array (IDL will automatically convert integer arrays) to set the range of the signal window for the selected independent variable in its proper units.
irange	Set to a two-element integer array to set the range of grid-indices of the selected independent variable corresponding to its signal window.

RunTime Keywords:

mnemonic	Set *mnemonic* to a two-element floating point array (IDL will automatically convert integer arrays) to set the range of the signal window for the independent variable identified by *mnemonic* in its proper units.

Usage:

object -> SignalWindow, keywords

The signal window is used to select a portion of the data for analysis. GKV analysis routines act only on data within the current signal window. For example, if you are interested in examining the linear phase from a non-linear simulation run you would set the signal window in time such that it covered just the initial transients before non-linear effects become important. Conversely, if you are interested in fully developed turbulence, set the signal window to exclude initial transients from further analysis.

11.6 PRO GET

Usage:

object -> GET, keyword=value, …

If you find that you are frustrated with the dictum *"Only an object's methods can access an object's data"*, use this "GET" (and the corresponding "SET") to work around it.

Keywords:

mnemonic	Mnemonic associated with GKVsd object
Title	Title of GKVsd object (used on 'pretty' plots)
Indices	An array used to properly label GKVsd plots...
units	Units of dependent variable
values	a POINTER to the GKVsd object's values
vrange	Range of dependent variable values for plots
ErrorBars	a POINTER to ErrorBars (of provided)
CodeName	Code in which data was created
CodePI	Additional info on this code

RunID	Information on particular code run
FileID	Additional information on code run

11.7 PRO SET

Keywords: See discussion under GKVsd

Usage:

Object -> Set, keyword=value

If you find that you are frustrated with the dictum *"Only an object's methods can access an object's data"*, use this "SET" (and the corresponding "GET") to work around it.

11.8 FUNCTION GetValues

Usage:

result = object -> GetValues

Returns a POINTER to an array containing those values of the dependent variable which lie within the object's SignalWindow. For 0-D GKVsd objects ONLY, it returns the (scalar) value rather than a pointer to this value.

*11.9 FUNCTION GKVs**N**D_Gen*

Usage:

result = GKVs1D(keywords)
result = GKVs2D(keywords)
…

Output is a valid GKVsd object of the specified dimensionality containing sample data with requested Fourier spectrum (and no correlations between Fourier modes), where ***N*** is an integer between 1 and 4 specifying the dimensionality of the object to be generated. The corresponding independent variables are (x, y, z, t).

Keywords:

Nx, Ny, Nz, Nt	Number of grid points in specified variable. (Optional)
Amplitude	The amplitude of the signal to be generated. (Optional)
kx, ky, kz, omega	The central wavenumber (frequency) in the specified independent variable of the signal to be generated. (Optional)
Del_k=bandwidth	The bandwidth of the signal to be generated. (Optional)