ACQ: A Data Acquisition Program for Cellular Neurophysiology Based on the MATLAB[1] Data Acquisition Toolbox[2]

Version 2.5 (November 15, 2006)

Paul B. Manis, Ph.D.[3]

Dept. of Otolaryngology/Head and Neck Surgery

Dept. of Cell and Molecular Physiology

Curriculum in Neurobiology

The University of North Carolina at Chapel Hill

[1] MATLAB is a registered trademark of The Mathworks Inc.

[2] This work was supported by NIDCD Grants R01DC00425 and R01DC04551 to Paul B. Manis, Ph.D.

[3] 1123 Bioinformatics Research Building Bldg, CB#7070, The University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-7070. e-mail: pmanis@med.unc.edu, http://www.med.unc.edu/∼pmanis

Contents

A INTRODUCTION

ACQ is a basic, extensible, data acquisition program for electrophysiology and biophysical studies of ion channels, based on the MATLAB (Mathworks) data acquisition toolbox. The program should serve the same purposes as pClamp (Axon Instruments), or a DOS program, DATAC. The program provides control of data acquisition (analog-to-digital conversion) and experimental protocols (stimulus waveforms delivered with digital-to-analog converters) through a set of m-files. Currently the program is focussed on acquiring data in response to current or voltage steps, pulse trains, alpha waveforms, or noise, and storing the data in MATLAB files on disk. Analysis is the domain of subsequent programs (such as the MATLAB version of "DATAC").

This document describes the operation of *ACQ*, as well as information necessary to write new stimulus modes or to add "macros".

A.1 Known bugs

1. Because the program operates in an event-driven mode, it is possible to initiate multiple actions that conflict with each other when using the GUI/mouse controls. Carefully click once to use the mouse-driven menu items. We have included more lock-out code to minimize problems, but by avoiding the temptation to click madly, the program's behavior will be more predictable.

2. Gapfree acquisition is not working. However, you can collect *very* long stretches of data.

3.

A.2 Document conventions

Program commands are given in `fixed-point type`. Optional arguments are enclosed with `[braces]`. Words used in a special context are *italicized* when first used.

B INSTALLATION

The program is installed by copying the '.m' files to their appropriate directories. The directory structure is simple. Under the top directory *(\mat_datac\acq)*, are the following: *AcqPar* (holds acquisition parameter files), *StmPar* (holds stimulus files), *Macros* (holds 'macro files'), *configs*, and *source* (all m-files that are accessible as commands must reside in the *source* directory). Under source are two directories: *private* and *utility*. These are the m-files that are not accessible as commands from the command box. m-files in the private directory are only accessible from the *source* directory, but not the *utility* directory. Routines placed in the *utility* directory need to be cognizant of this organization.

Once the files have been copied, a initial set of stimulus and acquisition files can be generated with the command `make_standard`. When this routine exits, a basic set of files can be found in the *StmPar* and *AcqPar* directories. It is recommended that a routine similar to this be created to allow you to automatically regenerate in a standard way any stimulus protocols you might use.

All of the routines necessary for the program are found ini the *source* directory and the directories below it. The MATLAB path is modified to include both the *source* and *source\utility*

directories when the program starts. To do this, you must go to the directory where *ACQ* is present, and run it, after which it should be sufficient to start it from the MATLAB command line. A good way to do this in new versions of MATLAB is to add a starting script to the MATLAB shortcuts list that appears just beneath the MATLAB toolbar.

The program requires a working copy of MATLAB, with the Data Acquisition and Signal Processing Toolboxes.

C CONFIGURATION

The basic configuration is controlled by files located in the config subdirectory. The choice of the configuration file is selected by a drop-down list when the program is started. The program is provided with a default file, called 'default.mat', which can be modified and renamed for each configuration that is desired. Thus, each user, or each user's experiment, can be organized by using the configurations, and appropriate subdirectories. Note that after a configuration file is created, it can only be loaded by exiting and restarting *ACQ*.

Details on configuring the program for specific amplifiers can be found in Section F.

D HARDWARE CONFIGURATION

ACQ uses the MATLAB Data Acquisition Toolbox. Any hardware supported by this toolbox can be used for acquisition; the program has been tested with 12 and 16 bit boards from National Instruments. In principle, the Windows sound card can also be used, although this is neither recommended nor fully implemented at present since it is only AC coupled and has limited input/output rates. We use NI6052E boards, with the BNC2090 chassis.

D.1 Connections

Configure NI boards as follows:

```
Triggering:

Connect PFI0/Trig1 to User 1 (via a short BNC cable)
       Connect DIO 1 to User 1 (use a wire jumper)
       Connect DIO 0 to PFI6 (use a wire jumper)
 OR:
       Connect PFI0/Trig1 to PFI16 and to DIO1.
```

The analog signals can be connected to any of the A-D input channels. Select NRSE mode for all channels by setting the appropriate switches on the BNC2090. The channels actually sampled and their order are set in the acquisition parameter file. However, if you are using an Axopatch amplifier, such as the Axopatch 200, 200A or 200B, from Axon Instruments (now owned by Molecular Devices), and wish to read the telegraphs, you **MUST** connect the telegraphs as follows:

`Amplifier MODE telegraph to A-D input 13.`

`Amplifier GAIN telegraph to A-D input 14.`

`Amplifier FILTER telegraph to A-D input 15.`

If you wish to control an external set of valves, you will need to connect lines from the digital IO port to the valve TTL input. The program sets DIO lines 5-8 to control up to 4 valves.

E OPERATION

E.1 Running ACQ

*ACQ*is run from within MATLAB. To run *ACQ*, first start MATLAB and type `acq` at the prompt. The program initializes its internal variables, sets paths to various files from a default configuration file (config\default.mat), allows the user to select a 'profile' (user-dependent directories and parameters) and registers the MATLAB functions (*.m files) present in the `.\source` directory as valid commands (these are the only commands recognized).

The program may be exited with the command `bye`; this should close all windows associated with the program. Alternatively, use File -¿ Quit from the menu.

E.2 Display

The program window is designed to take the full area of a 1024x768 display. It should not be necessary to view any other windows. (When using the Multiclamp Commander from Axon, it is convienent to position it over the parameter display area as described below). The display is divided into a four functionally distinct areas. First, down the left hand side there is a margin that holds several control buttons (described below). Second, the top of the next column holds the command line area (white box), the message area (grey box), and the status area (labeled blue boxes). Third, below this is the parameter display area, which shows the current settings for the selected parameter sets. Fourth, the right hand side of the display contains the graphics, and is divided into 3 areas. The top area has 2 vertically stacked graphs that display the output data waveforms for the two digital-to-analog convertor (DAC) channels. The default waveform for a stimulus generation (the one that is played out in "scope" mode) is shown in red, and the remainder are shown in white. Below this are two graphs that are used for on-line analysis. Currently, they are not programmable, and some stimulus generation routines may use these graphs to summarize the stimulus. Below these are two (or more) large graphs, which display the response data and the recorded stimulus waveforms. When the data is collected in "scope" mode, or is otherwise not being stored to disk, it is displayed with green lines and the axes are shown in gray. When data is being collected and stored on disk, the display lines are white, and the background axes and grid are blue.

E.3 Program Interaction

Commands may be given to the program either through the keyboard, or through selection of menu items and graphical interface elements. This section describes how to use the keyboard input. Commands entered at the keyboard are echoed in the field in the upper left hand corner of the screen (white area); when the window has focus, any characters entered at the keyboard should appear in that area. Currently, there is no 'text editor' per se, but a limited editing function is provided by the backspace and delete keys. Commands are entered one to a line, and terminated with the Enter key. A command buffer history is provided, which is accessed through the Control-U (up or back in the history) and Control-D (down, or forward in the history) keys. Commands may be entered while the program is acquiring, but this will stop the current acquisition. Stopping the acquisition with the space bar is recommended, as the first character entered at the keyboard is sometimes, but not always, lost during acquisition (this appears to be a problem with the relative execution times of various routines). There are no editable text fields on the screen that can be accessed with the mouse; all fields are "display only". Thus, if the window has focus, it should accept commands.

There are several buttons on the screen, appearing on the left hand margin. The `PV` button executes the preview function, which computes the stimulus waveforms and displays the results in the stimulus graphs. The `scope` button starts the program presenting the current stimulus waveform (the one indicated in red in the stimulus waveform window), but data is not stored. This is useful for setting up to patch or search for a cell, or exploring changes in stimulus paramters. The `stop` button ends scope mode, and also ends any data acquisition, including acquisition initiated by macros. The next button is `take 1`, which causes the program to present one stimulus and collect the data.

Below these are 4 buttons that are used mainly during initial patching and when switching amplfiier modes. The first is the `VC-S` button, which loads the s.mat protocol. This protocol, designed to be used in voltage-clamp, holds the electrode at 0 mV, with a brief step to -10 mV. The second is the `VC-I` button, which loads the i.mat protocol. The only difference here is that the pipette is held at -60 mV, and stepped to -70 mV. This is useful just prior to breaking into a cell, so that the break-in doesn't cause a large shift in membrane potential. Holding the pipette at a negative potential prior to break-in also sometimes helps improve the seal resistance. The next button is the `CCI` button. This is used in current-clamp mode to provide a short hyperpolarizing current pulse, to check input resistance and time constants once the cell has been ruptured and you are in whole-cell mode. Note that you have to switch the amplifier to current clamp before using this mode. The `SW` button located below the `CCI` button tries to coordinate a series of steps (turning off current injection, switching amplifier modes, loading the new protocol, and then enabling current or voltage commands) that minimize disruption of the cell during the transition between current and voltage clamp, as well as from voltage to current clamp.

E.4 A Typical Acquisition Session (command driven)

In this section, the operations of a typical acquisition session is described. Further details on commands are described below. A menu-driven mode is also incorporated, and is described later.

First, it is necessary to open a data acquisition file. In the white command entry box,

enter `aopen`. This opens a file to store the results of the acquired data. The file name is determined the same way as in *DATAC*: the name is formatted as ddmmmyyl.mat, where dd is the current date (leading zero), mmm is the 3-letter abbreviation for the current month, yy is a 2-digit year, and l is *a*-*z*, corresponding to the sequence of the file within the current date. It is recommended that this structure be adhered to. [4] The program will display a new window with several fields that should be filled in to describe the overall experiment. The last field is a line for the user (experimenter) to "sign". If this is the first acquisition of the day, then the fields will be filled with default values. However, if it is not the first file of the day, then the fields will be filled with the values from the previous file in the sequence and you should modify these as necessary. Click on `OK` to enter the dialog fields to the data file and close the box. You are now read to begin acquisition. Before proceeding further, you should make sure that the amplifier and the acquisition program are synchronized with respect to the operation mode - either voltage-clamp (VC) or current-clamp (CC). If the mode needs to be switched, use the `sw` command to detect the amplifier configuration and set the mode switch to the correct mode. Note that at present, this is only important for the Axopatch 200 amplifier. Other amplifiers cannot be read.

While searching for a cell, or beginning the formation of a patch, you will need to operate in a search mode. Typically, this will utilize a short current pulse (or voltage step). Load the stimulus configuration with the `g s` (this *G*ets the search file which is called `s.mat` and resides in the StmPar subdirectory; it also retrieves the acquisition parameters, which are contained in the same the file). Using the mouse, hit the "scope" button in the upper right hand corner of the display, or type `sco` in the command window. This begins a cycle of stimulus generation and display in an oscilloscope-style mode that can be used to monitor the electrode resistance or the formation of a patch. Once a seal is obtained, hit the "stop" button, and get the stimulus file to begin the intracellular acquisition (typically, `g i` (intracellular)). Once intracellular access is obtained as indicated by the capacitative charging transients and a change in the input resistance and holding current, one is ready for data acquisition.

A stimulus file/acquisition combination file can now be loaded with the `g filename` command. For example, a current-voltage relationship might be generated using a standard IV protocol, which is loaded with `g iv`. The command `seq` then will compute the stimuli and collect the data for an IV, using the parameters in the stimulus display - for an IV, cycling through the voltage steps. During the acquisition, the data will be displayed on the screen (assuming that the acquisition parameters are set to permit this), and the message box will show information about the progress of the acquisition. You may then get other protocols with the `g` command. Note that stimulus files may include acquisition information internally, or may externally reference a separate acquisition file.

You may also execute several stimulus protocols in sequence, using the `do xyz abc` command. The `do` command loads each protocol in order, makes sure the waveforms are current with the parameters, executes a `seq`, and when the acquisition is done, reloads the protocol that was in effect when the `do` command was entered, and goes into scope mode.

Notes regarding the experiment may be added with the `note` command. This brings up an

[4] I am considering adding to the date structure two "fields": one might describe the experiment type, and one the initials of the investigator running the experiment. This addition will prevent collisions in files collected on different rigs on the same day and is similar to the use of the investigator's initials for the file extension in our *DATAC* naming conventions. Comments on this are welcome

editing window, in which you may type a note of arbitrary length. The notes are meant to provide supplemental information, such as when valves are manually changed to apply drugs to a cell, or to indicate information about the status of the cell, that are either necessary or helpful for subsequent analysis. It is recommended that the notes be used generously; they form an important part of the experimental record and can really help untangle the acquisition session in data analysis that make take place years after the data was originally collected. Notes should also point out other files associated with this one, such as imaging data.

The data file is closed with the `aclose` command. Typically, there is only one cell to a data file (appropriate notes should be added if this is not true), and most cells will have only one data file, although exceptionally long protocols, program crashes, or lengthy recordings may result in several files holding data for one cell.

E.5 Switching Modes

Because the amplifier gains for current and voltage clamp commands are often different, the stimulus files are keyed to the acquisition mode. The telegraph outputs of the AxoPatch (but not the Multiclamp) series of amplifiers are interpreted by the program to insure that the modes of the amplifier and the intended stimuli are synchronized. If the modes were not synchronized, it would be possible to load a voltage-clamp stimulus file while doing current-clamp recording (or vice-versa), and this very likely would damage the cell. The next two paragraphs provide explicit instructions on how to change modes for the two different sets of amplifiers. You cannot change modes for the AxoProbe amplifier, since it is current-clamp only.

E.5.1 AxoPatch 200, 200A, 200B Amplifiers

To change the acquisition modes between voltage and current clamp, use the `sw` (*switch*) command; this command will wait for you to change the mode switch, and when it detects the change, it will load a default protocol for that mode (specified in the `config` structure). When setting the default modes, make sure that they have appropriate holding and step sizes for the configuration of the cell/patch that you want. After switching, which should leave you in a "safe" configuration, you will nearly always load new stimulus protocols associated with the new mode to collect additional data. To do this properly, you should switch to the I=0 setting and wait for the program to recognize the change (there will be an informative display message), before switching the the target mode.

E.5.2 All Other Amplifiers: MultiClamp

With other amplifiers, we cannot read the mode of the amplifier. Therefore, switching must be done differently. The following sequence of commands will ensure that you are presenting stimuli that are safe to the cell at all times:

- `STOP` the current stimulation
- Set the amplifier to I=0 (or disable the current injection).
- In *ACQ*, load a protocol appropriate for the mode you will go into next; e.g., `g i` if

going into voltage clamp next, or `g cci` if going into current clamp next.

- `START` the stimulus (and make sure it is going). This is an important step, so that the proper levels are going out the D-A converter.
- `NOW` switch the amplifier to the target mode (Voltage clamp or Current clamp).

Simply reverse this sequence in order to go back to the other mode.

E.6 Modifying the values in the data structures

Access to the data in the structures is accomplished via commands from the *ACQ* command window (the white box in the upper left corner of the main window). To be successful, this command must form the minimum unique sequence of letters that accesses the desired field, and does not collide with MATLAB command names that are registered with the program. The appropriate names are listed to the left of the data fields, and are case-insensitive. When these names have spaces, only the first part of the field is used.

Structure elements that take strings are simply entered (e.g., `Name ivx`). Structure elements that take single numbers can be entered in the same way. However elements that take more than one argument can be entered in two ways. One is to specify the index, followed by the value (e.g., `level 2 -100`, which changes the level of step number 2 to -100). The other is to specify the whole array, with a MATLAB-like syntax (e.g., `level [-60 0 -60 -120]`, which changes all of the levels and creates a 4-level step). Some care must be used when entering these numbers in either format, to ensure that valid numbers are entered; there is minimal checking in the program at present. It is also important to be sure that the number of events matches. For example, in steps, the number of levels and durations must be the same. In acquisition, the number of channels, gains, filter settings, scale factors, etc., must all match.

One exception the the entry format is that used to specify sequences. Sequences can be specified in several ways: as a MATLAB array; as a *DATAC*-style sequence; or as an m-file that returns appropriate values. Furthermore, the sequence may consist of several specified sequences (which in the case of the stimulus structure will be applied in a specific way to different control parameters), each using any of the above formats. Multiple sequences using the MATLAB array format in any of the individual cases must use an "&" sign between the sequences; this is not necessary if the sequence use *only* the *DATAC*-style. The resulting output represents the nested computation of the first sequence, cycled element by element against the second sequence, etc.

The command `seq` permits the rapid collection of parametric data once basic timing has been set. The use of this command can reduce the number of macros that one might maintain on disk as well as permits some flexibility during the experiment as to how the data will be collected. The syntax for "sequence" is as follows:¡p¿ valuelist is a sequence of the form a;b/xc that lists the values that the variable will take. The argument `c` is optional. `a;b/x` will step from a to b with increments of x; it is equivalent to the MATLAB statement [a:a+x:b]. The statement `a;b/xn` makes xsteps from a to b. `a;b/xl` makes x steps with logarithmic spacing. `a;b/xr` is the same as `a;b/xn` except that the results are in a randomized order. `a;b/xs` is the same as `a;b/xl` except that the results are in a randomized order. Floating point values are permitted. When random presentation is

done, the sequence is always the same (the same seed is always used). Similar sequences can be generated with MATLAB commands, for example `linspace or logspace`.

Sequences can be nested in another way also. Furthermore, if a file is specified for `addchannel` (e.g., for the second DAC output) or `superimpose` (summed with the primary DAC output), then the primary sequence will be repeated for each element of the sequences present in these stimulus blocks.

F SETTING STIMULUS AND RECORDING SCALING

Before commencing experiments, it is necessary to set up the stimulus parameter files. The way in which the program deals with the scaling of the data depends on these files, and there are several factors which interact. The ultimate indicator is the displayed data, which corresponds exactly to the numeric values of the data stored in the file.

F.1 Configuration parameter

In the configuration block, there is an entry for the amplifier type. The program recognizes several amplifiers from Axon Instruments, including the *AxoProbe* 1A (discontinued), the *Axopatch* 200 series, and the *Multiclamp* 700A. To get correct scaling for these amplifiers, you must enter the name of the amplifier in the field: `edit config; amplfier axopatch`, and save the configuration file. Additional amplfiers can be supported by editing the `acquire_one.m` file in the appropriate section.

F.2 Stimulus Scaling

There is no independent control of the stimulus scaling; it is set automatically according to the amplifier type and data collection mode (voltage or current clamp) in the `acquire_one.m` routine.

F.3 Recording Scaling

The recording scaling is a function of three different factors: the intrinsic gains of the amplifier, the setting of the gain selector on the output of the amplifier (for Axopatch and Multiclamp amplifiers), and the data collection mode. Recording scaling is controlled by the `sensor` parameter in the acquisition parameter block (`edit acquisition`). Typically these `sensor` values are large, rounded positive numbers, and there is a separate value for each channel collected. For example, to set the multiclamp commander, with the output gain set for 2 (corresponding to 1V/nA), the command:

```
sensor [40 2000]
```

will produce the correct data scaling. Note that in voltage clamp, the current channel is collected first, so this corresponds to a channel list (for example) of:

```
channel [11 3]
```

where the current output of the amplifier is connected to A/D input 11, and the voltage output is connected to A/D input 3.

The recording scaling is also affected by the amplifier gain. For the Axoprobe, this is fixed

in the software (10x for voltage, 1nA/V for the current), but for the Axopatch, the gain is read from the telegraph inputs, and is dependent on the mode (voltage or current clamp). For the Multiclamp, telegraphs are available, but we cannot read them into MATLAB yet, so it is treated as a fixed gain system, for which you will have to write down any variations from the standard gain settings.

In practice, setting the channel sensor factors is fairly easy, assuming that you have an independent way to verify the different outputs of the system.

Once the sensor values are set, it is a good idea to write a short MATLAB script to update these values, based on the `setcc` and `setvc` scripts found in the source directory. In this way, you add a command to the system that sets the gains automatically whenever you wish to design a new stimulus or acquisition protocol. A smart script would read the current configuration (`global CONFIG`), to get the amplifier type (in `CONFIG.Amplifier.v`), and set the gains accordingly.

G UTILITY COMMANDS

G.1 Index of the Data File

There are two routines for viewing the index of the data file. The first is `lif` (*List Index File*), which simply prints out to the MATLAB window the index file information. A more complete listing of the file can be obtained with the `pf` (*print file*) command. This lists the index, followed by information about each block including all of the note entries. You should consider making a hardcopy of this result and saving it for future reference.

G.2 Displaying data

The display settings for voltage and current can be modified. The command `vdis min max` sets the voltage display minimum and maximum values. Data outside this range is clipped and not displayed. The command `idis min max` works similarly for the current traces.

Data can be displayed after it has been collected using the `db` (*display block*) command. This command operates in one of two ways. If a file is currently open, then the command `db block#` will display ALL of the data in that block (there is no way to display individual records). If there is no file open, then the command `db block# filename` will display the requested data block from the specified file. The data is displayed into the data window using the current display settings.

The drop-down menu under `display` permits modification of display parameters for multiple channels.

G.3 Valves

Solution delivery can be commanded from the program by using the `valve` command. This requires a separate hardware interface, and is controlled through the digital IO lines of the acquisition hardware. See me for details.

H USING THE MENU

To simplify program operation, a standard drop-down menu list has been implemented, as can be seen on the top of the window. The menu provides the following functions:

H.1 File

This menu entry brings up a submenu largely related to file maintenance.

H.1.1 Open

This invokes the aopen command to open a data file for collection. The filename is automatically generated as described above.

H.1.2 Close

This invokes the aclose command to close a currently open data file. Closed data files cannot be re-opened.

H.1.3 Gather

This causes the program to update the current command list. Only commands that are 'registered' with the program can be executed from the command line. Normally this command is used only during program development when new commands are added. The 'gather' function is implemented automatically on program start-up and normally is not needed by the user.

H.1.4 Exit

Closes any open data file and exits the program to the MATLAB prompt.

H.2 Edit

This menu brings up a submenu controlling which parameter sets will be edited or saved.

H.2.1 Edit Acquisition

Selects the acquisition parameters for display and command-line editing.

H.2.2 Edit Stimulus

Selects the stimulus parameters for display and command-line editing.

H.2.3 Edit Stimulus2

Selects a secondary stimulus parameter set for display and modification by a GUI. Note that this only works for `pulse` stimuli at present. The GUI is designed to allow rapid modification of pulse trains (for example, used to stimulate afferent fiber systems) delivered on a second DAC channel, and updates the information in the primary protocol.

H.2.4 Edit Config

Selects the configuration parameters for display and command-line editing.

H.2.5 Save Acquisition

Saves the current acquisition parameters to disk, optionally overwriting or renaming the file.

H.2.6 Save Config

Saves the current acquisition configuration parameters to disk, optionally overwriting or renaming the file.

H.2.7 New

Each of the 'New' entries corresponds to the creation of a stimulus template for a different kind of stimulus. Use this when you are making new stimulus sets.

H.3 Protocol

This menu selection brings up a choice of commands relating to protocols, including a dynamically updated list of available protocols in the current *StmPar* directory.

H.3.1 Save Protocol

Saves the current protocol to disk.

H.3.2 Update Protocols

Reloads the current list of stimulus protocols from the selected directory and redisplays them in the menu.

H.3.3 Change Directory

This selection brings up a Windows directory browser to change the current stimulus protocol directory. A successful directory selection results in an updated list of protocols found in the new directory. The entry *StmPar* in the configuration parameter set is also updated to reflect the current choice (however, the configuration is not permanently updated, and must be saved manually if the new selection is to be made permanent).

Below this is a list of the recognized *StmPar* protocols in the selected directory. You may select one of these protocols from the list, and it will be loaded, recomputed if necessary, and is ready to use.

H.4 Macros

H.4.1 update list

reads the contents of the macros directory, and updates the drop-down list of available macros.

The remainder of the list allows selection of a macro from those a available.

H.5 Acquisition

H.5.1 sequence

causes the program to execute the current sequence, storing data if a file is open. This is how most parametric data is collected.

H.5.2 data

initiates collection of data without sequencing, and repeats until the stop button or menu item is selected.

H.5.3 scope

puts the program in scope mode (same as the button).

H.5.4 stop

stops ongoing data acquisition (same as the button).

H.5.5 switch

controls mode switching between voltage and current clamp with Axopatch 200 series amplifiers. This requires reading the telegraphs from the amplifier.

H.6 Display

H.6.1 Erase

forces a redraw of the screen

H.7 Help

H.7.1 Help

shows the list of current commands, with a short description of each one (taken from the m-file).

H.7.2 Show flags

shows the status of several of the control flags (mostly used for debugging).

H.7.3 Clear flags

resets the status of the flags, so that acquisition can run. This was necessary at one point to prevent the acquisition from stopping or failing to start. If the program seems to have stalled, or does not begin collecting data when requested, use this menu item to reset it to a known state.

I MACROS

One of the features of this program is the ability to write 'macros', or scripts that control data acquisition, using MATLAB commands and the routines/commands available to the program. The ability to use the power of MATLAB within a macro greatly extends the usefulness of the macros. Such macros are straightforward to create; however a few rules must be applied to allow the macros to participate successfully with the overall action of the program.

I.1 Writing Macros

Each macro should be defined as an m-file function, and stored in the *macros* directory as defined in the configuration. You must always be sure that the macro handles any errors gracefully, leaving the relevant flags in a proper state. To help with this, macros must use two supplied routines to insure proper operation.

At the beginning of the macro, you must check to see that it is ok to run the macro. The routine `ok_macro_run` does several things. First, it checks to see if the program is in scope mode, and if it is, it stops scope mode and prints a warning message; the macro is not run (you can start it again however, and it will run). Second, we make sure that no macro is currently running: nested macros are not allowed. With the GUI, which is an event-driven interface, it is possible to start several processes and not realize that other processes are already running, since once the macro starts, the menu is still active. Thus, we prevent starting a second macro. Third, we make sure that a file is open so we can store data. There is little point in running a large, complex data collection controlled by a macro if there is no file open.

Include this code as the first lines of the macro:

```
global IN_MACRO \% access the macro flag; this flag will be set
by the ok_macro_run routine if the macro can run.

if(~ok_macro_run)

    return

end;

}
```

Next, after every acquisition call (e.g., `seq` or `take`), there must be a call as follows:

```
if(check_macro_stop) return; end;
```

This allows the user to use the *stop* button to stop the macro, without having to stop each and every protocol that is attempted in the macro. It may be necessary also to reset the valves (`do_valve(manual)`), and turn off the "slow voltage clamp" (`sethold off`) if these are being used.

Finally, the last line in the macro should be:

```
IN_MACRO = 0;
```

This is to clear the macro flag when a stop is hit. *check_macro_stop* routine also performs this action when a macro is stopped from the button.

I.2 Example Macro

The following macro is somewhat complex, but illustrates the proper way to write a macro. Note that `try-catch` error handling is wrapped around the macro, so that if it fails because of a programming error, the system state is maintained.

This macro controls valves to challenge a cell with a drug, while running the slow-voltage clamp to keep the membrane potential constant, and monitors the responses both to single pulses (`ltp_base`) and to a parametric pulse protocol (`ap-hyp` protocol. Note also the inclusion of a test mode flag, which allows the macro to be tested for correctness in behavior and coding, without taking the full running time (50 minutes).

```
function hyp_drug(arg) function hyp_drug(arg)
% hyp_drug.m  -  protocol for to test effect of drugs on discharge pattern
% 7/18/2001
% Paul B. Manis, Ph.D.
% pmanis@med.unc.edu

% Required in all macros:

%-----------------------------------------
global IN_MACRO

if(~ok_macro_run) % function returns 0 if not ok to run the macro
   return;
end;

try % handle errors - note that this makes it hard to find
    % the errors, but keeps the program sync'd

    %-----------------------------------------
   % set up a test mode for quick testing to be sure macro works
   % and is error free
```

```
testmode = 0; % flag: 0 is normal full run, 1 is the 'test mode'

nsamp = 60 % number of samples to take...
if(testmode)
   nsamp = 6; % use for testing....
end;

pausetime = 1; % set up pause timer
if(nsamp == 6) % in test mode, we use a short wait.
   totpause = 5; % 10 second update
else
   totpause = 5*60; % total pause in seconds
end;


%-------------------------------------------------------
% initialize
 % we control the valves: tell user to switch valve control
do_valve('computer');

valve(1); % select valve 1.

% do we use 'slow voltage clamp' ?
% any argument on the input is same as 'hyp_drug_manual...'
% - i.e., we don't set holding ourselves
if(nargin == 0)
   sethold set   % make sure holding is locked
end;

g ltp_base  % baseline measurement of 'buildup' response

% insert a 5-minute pause before we start the data collection
% to allow the cell to stabilize.

for i = 1:floor(totpause/pausetime)
   QueMessage(sprintf('Pausing for stability: %d sec remaining', ...
      totpause - (i-1)*pausetime), 1);
   pause(pausetime);
   % a line like this is necessary after every command
   % to stop the macro completely.
   if(check_macro_stop)
      do_valve('manual');
      sethold off
      return;
   end;
end;

% -------------------------------------------------
```

```
% collect baseline data

take(nsamp);  % sets up for 5 minutes as 5 sec/trial (12/min)
if(check_macro_stop)
   do_valve('manual');
   sethold off
   return;
end;

% do hyp protocol before switching the valves
g ap-hyp
seq
if(check_macro_stop)
   do_valve('manual');
   sethold off
   return;
end;

% ------------------------------------------------
% now change the valve to the test solution
% and collect data during drug wash-in

QueMessage(sprintf('Switching to valve %d NOW!!! ', 2), 1);
valve(2)

g ltp_base % measure again with same parameters as baseline
take(nsamp*2); % 10 minutes worth here...
if(check_macro_stop)
   do_valve('manual');
   sethold off
   return;
end;

% return the valve to the normal solution
QueMessage(sprintf('Switching to valve %d NOW!!! ', 1), 1);
valve(1);

% ------------------------------------------------
% collect post-drug data
% the following loop does 2 things:
% we watch and we do parameteric measurements
% watch for 20 minutes or so, but keep doing
% the hyp protocol every 5 minutes.

for i = 1:4
   % do/verify hyp protocol
   g ap-hyp
   seq
```

```
   if(check_macro_stop)
      do_valve('manual');
      sethold off
      return;
   end;
   %
   g ltp_base % measure again with same parameters as baseline
   take(nsamp);  % take 5 minutes
   if(check_macro_stop)
      do_valve('manual');
      sethold off
      return;
   end;
end;

% ------------------------------------------------
% done - now just check cell properties at the end of the run

g ap-hyp % repeat the hyp protocol.
seq
if(check_macro_stop)
   do_valve('manual');
   sethold off
   return;
end;
%
% get cciv again and re-run with all parameters
g ap-iv % to confirm basic cell information
seq
if(check_macro_stop)
   do_valve('manual');
   sethold off
   return;
end;

 % ------------------------------------------------
% restore default conditions

do_valve('manual'); % tell user to return valves to manual control

sethold off % always turn off slow vclamp

%------ REQUIRED OF ALL MACROS::: successful exit
IN_MACRO = 0; % turn off macro flag.
return;

%*********
% handle matlab errors.
```

```
catch
   QueMessage('Macro hyp_drug: FATAL error detected (try/catch)',  1);
   acq_stop;
   do_valve('manual');
   sethold off;
   IN_MACRO = 0;
   return;
end;

return; % that's all
```

J PROGRAMMING NOTES

J.1 Extending commands

As mentioned above, adding new commands to the program is fairly simple. A command is equivalent to an m-file function that is located in the `source` directory, i.e., every m-file in that directory is a command. Functions that need to be hidden should reside either in the utility or private directories beneath the source directory. The m-files in the source directory are registered when the program starts. To register new commands, use the "gather" menu item under the "File" menu. Commands are interpreted as the minimum unique letter combination that will specify an m-file. For example, the command `l` is not unique, since there are several commands that begin with the letter "l". However, `lc` will list the configuration file structure to the MATLAB window (used for debugging purposes).

Command functions may accept parameters on the input line, but each function must parse and check these parameters on its own. Command functions should return error messages to the message window using the QueMessage function, to indicate their status or guide the user.

Command functions may access the data structures STIM, DFILE, and CONFIG by declaring them to be global variables. Any changes to these structures will be seen throughout the program, so this should be done carefully.

J.2 Adding Stimulus Protocols

Adding a new stimulus pattern (waveform) requires changes to 3 parts of the program. *First*, the routine new.m must be modified to generate the fields and tags for the new parameters needed. A look at new.m will make it very clear how this should be done, and how easy it is. *Second*, a *method* routine must be written to generate the stimulus waveforms. Examples of existing method routines are `steps.m`, `pulses.m`, `ramp.m`, `alpha.m` and `noise.m`; it is recommended that you start by copying one of these (`noise.m` and `steps.m` are presently the most recent and probably most cleanly coded stimulus methods). Some support routines are also available (see `noise.m` for how these are used); I expect to provide a small package of these in the future to simplify the process. Writing the method itself is usually the most

difficult part of the coding process, as it must handle the sequencing of stimulus parameters and generate appropriate arrays for output. The output arrays are held in the form of a cell array, so that each different stimulus in the output may have a different length or time base. Note that the command `pv` (*preview*) will generate the stimulus waveforms by calling the method routine. *Third*, the name of the routine must also be registered with the `chkfile` routine (found in the private directory), under "STIM", in order to allow the program to verify the correctness of the structure later on.

Note that stimulus generating protocols can call other protocols to either generate a second channel or to superimpose two different classes of waveforms generated by different methods on one channel. This is handled by the functions `combine.m` and `superimpose.m`. Examples of their use may be found in each of the existing stimulus method routines; it usually will suffice to copy the code from those routines into your new routine. Please send any new routines you develop to me at the e-mail address above, so I can incorporate them into the base code structure.

Although the stimulus protocols and their parameters are largely self-explanatory, a short text on how the parameters control the stimulus can be helpful (I haven't written such notes yet for the 5 implemented so far). However, by using the `pv` command, it should soon become clear how the parameters control the generated waveform.

K SUPPORTING FILES

K.1 Configuration

The configuration file describes the location of data, acquisition and stimulus files, the amplifier type in use, and the default stimulus files to be used when switching to current or voltage clamp. The configuration file is created by the command `new config`, and the named fields can be filed in. The configuration file is saved with the command `sc` (for `save config`). A different configuration file can be loaded with the command `gc` (for `get config`). If the configuration parameters are not current visible in the window, they can be brought forward either by selecting the `config` button, or by issuing the command `e config` (`edit config`).

When the configuration parameters are displayed, the names of the parameters, listed on the left side of the window, can be used as the commands to set the values on the right. Most of the parameters are self explanatory. However there are a few rules:

The BasePath is the path under which the various stimulus paramter, acquisition parameter and data files are stored. The stimulus, acquisition and data paths should be entered relative to this base path:

BasePath *c:\mat_datac\acq*

StmPath *StmPar* (Where the program expects to find the stimulus files. Note that this can be modified through the protocol menu.)

AcqPath *AcqPar* (Where the program expects to find the acquisition parameter files.)

DataPath *Data* (Where the program stores the data, and associated temporary data files.)

The configuration parameter *UserExt* defines a text string that is *prepended* to the filename. This is to allow the files to have unique identifiers in case the program is used to collect

data in an environment with more than one data collection occurring on a given day.

K.2 Stimulus files

Stimulus files are central to the data acquisition operation of the program. The stimulus files are really MATLAB structures that are polymorphic: each different kind of stimulus is generated by a different *method*, which is driven by the data from the associated structure. The stimulus files all have a common header region, and a variable tail. The header contains identification of the file type (stim), the name of the file, and, upon writing to the data file, the m-code for the method and a possibly a copy of the stimulus waveforms. The header region is not visible to the user, and cannot be changed except within the program. Thus, this structure contains everything about the stimulus that might be needed at a later point either to reconstruct the stimulus or to determine glitches in stimulus computation (which of course will never happen).

The variable region of the stimulus file contains the data that is used to drive the generation of the stimulus waveforms by the method routine. For example, the files for the *steps* method contains information about the step levels and durations, the stimulus sequence, scale factors, holding voltages/currents, which parameter types and which step elements are to be sequenced, the associated acquisition control structure/file, and an optional pointer to a stimulus file that can be superimposed on the primary file. Note that if the associated acquisition control file field is empty, then the current acquisition field will be stored along with the stimulus data. This is the preferred method.

The current stimulus waveform can be previewed with the `pv` command, and saved to disk with the `s [name]` command. If the name doesn't match the filename field in the structure, then the program will ask if the name should be changed. Stimulus files are restored from disk with the `g [name]` command; if no name is given a GUI file browser interface is provided to access the files. The contents of the current stimulus structure can be listed with the `ls` command from the MATLAB command line; and the structure itself can be returned to the base workspace with a command such as `s = ls;`; s will contain STIM (s.STIM), which is the stimulus structure data. It is useful to examine these structures and the `new.m` file when writing new stimulus methods and structures (Details of the specific default methods provided with the program are described below).

Stimulus files are stored on disk in the directory designated by the configuration. The files are stored as standard MATLAB.mat files, and contain a single structure corresponding to the stimulus. If no acquisition file is specified when the stimulus file is stored with `s`, then the presently loaded acquisition file is save along with the stimulus file; thus the file contains two structures, and retrieving the stimulus file then retrieves the acquisition information also.

K.3 Acquisition parameters

The control of data acquisition is separate from the generation of stimuli, in the sense that it is held in a different structure that can be reused or shared by different stimulus paradigms, or alternatively, stored with the stimulus paradigm. Acquisition structures are not polymorphic (there is only one kind of acquisition structure). Each acquisition structure consists of a fixed section, and a variable, user modifiable section. Acquisition

parameters include the channels to be collected, the sample rate, the number of points, the acquisition mode (voltage or current clamp), parameters that control the refresh of the display and points displayed, amplifier gains, acquisition hardware range settings and scale factors, filter settings, and potentially a junction potential offset. If not currently visible, the acquisition structure can be brought forward with the command `e acq`. The current structure can be saved with `sa [name]`, and retrieved with `ga [name]`. The data in the structure can be listed at the MATLAB command line with the `ld` command, or retrieved into the base workspace with the command `d = ld`. When the acquisition structure is currently displayed, the parameters can be edited using the same methods as described above for the stimulus parameters.

L DATA STRUCTURES

L.1 Stimulus Structure Definitions

(The following information is out of date, but serves as an example). The stim file consists of a base set of information and a variable section.

L.1.1 Base section

```
sfile.title='Stimulus Parameters';
sfile.NAME='STIM'; sfile.callback='paste';
sfile.frame = 'FStim'; % frame to associate window with
sfile.fhandles = []; % handles for data elements
sfile.method_code=[]; % holds the source file for the method.
sfile.waveform = []; % holds the actual stimulus command waveforms
sfile.start=1;

% the following are common to all stim types:

sfile.Method = create_element(method, SGL, 1, 'Method', '%s');
sfile.Name = create_element('IV', SGL, 2,'Name', '%s');
sfile.AcqFile = create_element('default', SGL, 3, 'AcqFile', '%s');
sfile.Cycle = create_element(1000, SGL, 3, 'Cycle(ms)', '%8.1f', 0, 50, 65000);
sfile.Repeats = create_element(1, SGL, 4, 'Repetitions(N)', '%d', 0, 1, 1000);
sfile.Stim_Repeat = create_element(1, SGL, 5, 'ProtocolReps(N)', '%d', 0, 1, 50);
sfile.Sample_Rate = create_element(1000, SGL, 6, 'SampleRate(us)', '%8.1f', 0, 1);
```

L.1.2 variable sections

The following determines the variable section:

```
steps: % Create a series of steps with duration and level, with a sequence
sfile.Sequence=create_element('-100;50/5', SGL, 8, 'Sequence', '%c');
sfile.SeqParList=create_element('Level', SGL, 9, 'SeqParameter', '%c');
sfile.SeqStepList=create_element(2, MULT, 10, 'SeqStepNo', '%d', 0, 1);
sfile.Duration=create_element([5,100,50], MULT, 11, 'Durations(ms)', '%8.1f', 0, 0, 30000);
```

```
sfile.Level=create_element([0,-100,0], MULT, 12, 'Level', '%8.1f');
sfile.Holding = create_element([0 0], MULT, 13, 'Holding', '%8.2f');
sfile.Superimpose = create_element('', SGL, 14, 'Superimpose', '%c');

ramp:
sfile.Durations=create_element([5,5,400], MULT, 11, 'Durations(ms)', '%8.1f', 0, 0, 30000);
sfile.Levels=create_element([-60,-100,0], MULT, 12, 'Levels(mV)', '%8.1f');
sfile.Holding = create_element([-60 0], MULT, 13, 'Holding', '%8.2f');

pulse:
sfile.Npulses=create_element(1, SGL, 8, 'NPulses', '%d', 0, 0, 65000);
sfile.Delay=create_element(5, SGL, 9, 'Delay(ms)', '%7.1f', 0, 0, 100000);
sfile.IPI=create_element(10, SGL, 10, 'IPI(ms)', '%8.2f', 0, 0.001, 65000);
sfile.Duration=create_element([0.1, 0.1], MULT, 11, 'Durations(ms)', '%8.2f', 0, 0, 30000);
sfile.Level=create_element([100,-100], MULT, 12, 'Levels(mV)', '%8.1f');
sfile.LevelFlag=create_element('absolute', MULT, 13, 'LevelFlag', '%s');
sfile.Scale=create_element(1, SGL, 14, 'Scale', '%8.3f', 0, -100000, 100000);
sfile.Offset=create_element(0, SGL, 15, 'Offset', '%8.3f', 0, -100000, 100000);
sfile.Sequence=create_element('1;100/25', SGL, 16, 'Sequence', '%c');
sfile.SeqParList=create_element('Level', MULT, 18, 'SeqParameter', '%c');
sfile.SeqStepList=create_element(1, MULT, 19, 'SeqStepNo', '%d', 0, 1);

alpha:
sfile.Npulses=create_element(1, SGL, 8, 'NPulses', '%d', 0, 0, 65000);
sfile.Delay=create_element(5, SGL, 9, 'Delay(ms)', '%7.1f', 0, 0, 100000);
sfile.IPI=create_element(10, SGL, 10, 'IPI(ms)', '%8.2f', 0, 0.001, 65000);
sfile.Alpha=create_element(0.1, SGL, 11, 'Alpha', '%8.2f', 0, 0, 30000);
sfile.Amplitude=create_element(1, SGL, 12, 'Amplitude', '%8.1f');
sfile.Scale=create_element(1, SGL, 14, 'Scale', '%8.3f', 0, -100000, 100000);
sfile.Offset=create_element(0, SGL, 15, 'Offset', '%8.3f', 0, -100000, 100000);
sfile.Sequence=create_element('1;100/25', SGL, 16, 'Sequence', '%c');
sfile.SeqParList=create_element('Level', MULT, 18, 'SeqParameter', '%c');
sfile.SeqLevelList=create_element(1, MULT, 19, 'SeqStepNo', '%d', 0, 1);
```

L.2 Data Acquisition Structure

```
dfile.title = 'Data Acquisition Parameters';
dfile.NAME = 'DFILE';
dfile.callback = 'paste';
dfile.frame = 'FDfile';
dfile.fhandles = [];
dfile.Z_Time=0;
dfile.F_Time=0; % file time
dfile.File_Mode=-1;
dfile.Block = 1; % internal block information
dfile.Record=1; % incremented internally
dfile.Actual_Rate = 20; % actual rate used by stim board
```

```
% definitions
SGL = 0; MULT = 1;

% the following parameters are adjustable...
dfile.start = 1;
dfile.Name = create_element('Default', SGL, 1, 'File', '%s'); % file
dfile.Record_Skip=create_element(4, SGL, 2, 'Display Skip (n)', '%d', 4, 1, 100);
dfile.Refresh=create_element(0, SGL, 2, 'Refresh (n)', '%d', 4, 1, 100);
dfile.Data_Mode = create_element('CC', SGL, 4, 'Acquisition mode', '%s');
dfile.Sample_Rate = create_element(20, SGL, 5, 'Sample Rate (us/pt)', '%d', 20, 1, 65000);
dfile.Points = create_element(5000, SGL, 6, 'Points per record', '%d', 2048, 256, 1000000);
dfile.Channels = create_element([0 1], MULT, 7, 'Channels to sample', '%d', 2, 1, 16);
dfile.Amplifier_Gain=create_element([0 0], MULT, 8, 'Amplifier Gains', '%8.1f', 1, 0.1, 100
dfile.AD_Range=create_element([5 5], MULT, 9, 'A-D Range (V)', '%8.2f', 5, 0.1, 10000);
dfile.Sensor_Range=create_element([200 20], MULT, 10, 'Sensor Ranges', '%8.2f', 5, 0.001, 1
dfile.Low_Pass=create_element([10.0 10.0], MULT, 11, 'LPF (kHz)', '%8.2f', 10.0, 0.01, 1000
dfile.High_pass=create_element([0 0], MULT, 12, 'HPF (kHz)', '%8.2f', 0, 0, 1000);
dfile.Junction_Potential = create_element(0, MULT, 13, 'JP (mV)', '%8.1f', 0, -200, 200);
dfile.end = 1;
```

L.3 Configuration Structure

```
cfile.title='Configuration Parameters';
cfile.NAME='CONFIG';
cfile.callback='paste';
cfile.frame = 'FConfig';
cfile.fhandles = [];
cfile.start=1;

cfile.Name = create_element('base', SGL, 1,'Name', '%s');
cfile.BasePath = create_element('BasePath', SGL, 3, 'BasePath', '%s');
cfile.StmPath = create_element('StmPar', SGL, 4, 'StmPath', '%s');
cfile.AcqPath = create_element('AcqPar', SGL,  5, 'AcqPath', '%s');
cfile.DataPath = create_element('Data', SGL, 6, 'DataPath', '%s');
cfile.Amplifier = create_element('Axopatch200', SGL, 7, 'Amplifier', '%s');
cfile.VC = create_element('VC_Default', SGL, 8, 'VCStim', '%s');
cfile.CC = create_element('CC_Default', SGL, 9, 'CCSTim', '%s');
cfile.end=1;
```

M DESIGN CONSIDERATIONS AND HISTORY

One of the general goals in writing this program was to include sufficient flexibility that the end user can readily customize the program. To this end we chose to let MATLAB do as much of the work as possible, and we chose to allow the user access to the capabilities of MATLAB rather than having the program take complete control. The program as provided and described herein provides a strong set of basic commands, protocols, and simple data storage. In addition, the end user can write "macros" (functions) in the MATLAB language to extend the program.

In *ACQ*, we have adopted the use of keyboard input rather than mouse-directed input, because it is generally easier and faster during an experiment to type short commands at the keyboard than it is to navigate a mouse through a set of menus. We have tried to implement the interaction system in a way to allow brief commands from the keyboard, as in the predecessor program, *DATAC*, because we have found that during an experiment this is the most expedient way to send commands to the computer. Nonetheless, a few graphical mouse items are necessary (e.g., buttons) because of the callback structure of MATLAB, which is used in *ACQ*, and the observation that MATLAB does not always respond to the keyboard when certain tasks are being performed, but it will respond to mouse clicks.

A second design decision in *ACQ* is that data *structures* should drive the program. All relevant information necessary to perform an acquisition protocol is held in data structures, and these structures are also made part of the archival data file. The structures invoke routines to generate stimuli, and drive the acquisition engine. From these structures, it should be possible to derive all features of the acquired data and stimulus. Presently, two structures (the stimulus and acquisition structures) completely determine how stimuli are generated and how the data is acquired. A third structure, the "note" structure, is used to store ancillary information about the experiment in a text form. A fourth structure informs the program about the general configuration of the program directories and the hardware, and has no information necessary for the analysis.

One might ask why we have chosen to use MATLAB? The answer to this question lies partially in the prior choices and experience with digital data acquisition for electrophysiology on a number of platforms (Data General, 1975-76; DEC PDP 11-40 (RT-11), 1977-1982; DEC PDP8 (LINK/FOCAL) and PDP 11-34 (TSX/RSX), 1982-1985, 8088, 80286, 80386, 80486, 80586/PC, 1985-1999, DOS 3.1-6.0, Windows 3.11, 95, 98). Our previous data acquisition program, *DATAC*, was written originally by Daniel Bertrand in C. This program had modules for array computation, data display and figure construction, data acquisition, and a macro facility. The program, originally written in 1985, ran only under DOS (although versions for the Mac and Sun systems using a graphical interface were generated). In the Manis laboratory, the program was extended to have a more powerful macro facility, extended array functions, and especially a powerful, fairly flexible, data acquisition engine, while remaining in the DOS environment. This program served well, was relatively free of bugs, and was easy to operate for data acquisition.

However, the program has neared the end of its useful lifetime for several reasons. DOS is no longer properly supported, and the supported data hardware (Axon Instruments DIGIDATA 1200 board, or the Manis-Bertrand board) uses the ISA bus, which has nearly disappeared from the current computer bus options. Rather than rewrite *DATAC* under Windows (a somewhat problematic task, given the direct interaction of the program with the hardware and the DMA controller in the PC), it was felt that a more general approach should be used. The approach should in particular provide a short development cycle, a strong and robust mathematical and graphic support facility, and make it easier for end users to extend the program for specific experiments. Originally, Origin (Microcal) was considered as a platform, since a GUI-based interface can be easily implemented and DLLs can written to control hardware. However the scripting language used by Origin, LabTalk, was found to be inconsistent and buggy, poorly typed, and overall both awkward and cumbersome. MATLAB provides all of the facilities of *DATAC*(including array processing, graphics or plotting functions) with few of the original limitations, and furthermore has a proper lan-

guage syntax. With the advent of the Data Acquisition Toolbox (or the wrapper programs for the NIDAQ library that are available), it became feasible to use MATLAB to replace nearly all of the functions of *DATAC* without having to write code for the basic engine. Furthermore, MATLAB runs well under Windows, Mac and Linux operating systems, so data analysis components should be platform independent (to a limited extent, this might be true for data acquisition between Windows and Mac; there is currently no complete library for Linux.). Thus, MATLAB seemed a logical choice, and perhaps is better than *DATAC* because of the extensive library of routines and the ability to rapidly extend the useful language with either C-coded mex files that act like MATLAB commands, or m-files.

The resulting program operates somewhat similarly to *DATAC* in that it has a display structure roughly based on the *DATAC* display, and has commands that are similar or identical to those in the *DATAC* acquisition engine. However, the program is more flexible and more easily modified than *DATAC* since it is not compiled, and it has significantly greater capability and flexibility in acquisition and stimulus generation. Furthermore, its development time was considerably shorter than that of *DATAC*: about 2 months of direct programming time, versus an estimate of 8 months overall for the acquisition engine in *DATAC*. The cost however is that the minimum computer requirements are currently a 400 Mhz or faster PC, running Windows 98. The resulting data files are also significantly larger, due to the use of single floating point format for storing data (as opposed to integer storage in *DATAC*), and the storage of a significantly larger body of ancillary information (such as stimulus waveforms) that ultimately should make analysis easier and more accurate.

Finally, the ultimate success of this program lies in its use. We have now used the program for *all* data collection in my lab for nearly a year. There have been and may still be a few bugs, and operation is not quite as smooth as I would like, but progress is being made. Although I initially disliked the GUI control (menus, etc), I find myself using it more, and plan to include GUI control for the stimulus parameters similar to that implemented for the pulse protocol for secondary channels. I am satisfied that the program serves its purpose as designed and that it does allow us to collect data in a much more flexible way than before. New stimulus protocols have been written (usually taking only a few hours for coding and testing). Routines have been written to access the data structure and pass data to my data analysis program, and the data has been correctly analyzed. New commands have been added with minimal work. Complex stimulus step protocols have been designed and are found to work correctly.