Contents

----

1) Technical Details:

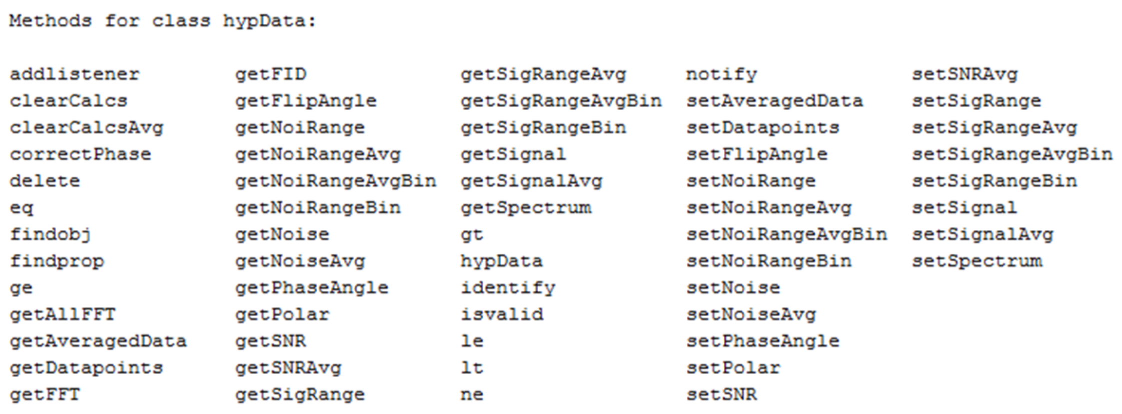
HypPo is a graphical user interface for the analysis of SPAR/SDAT spectroscopy files. It was programmed in Matlab 2012a with the use of GUIDE. Testing and development was performed primarily on a Windows 7 x64 laptop, but some testing in the Mac OSX operating system was performed.

2) What is the HypPo data tool?

HypPo is a spectroscopy tool for analyzing spar/sdat files. It can perform functions such as visualization, SNR calculations, filtering, phase correction, averaging, and normalizing. All data is stored in a HypPo object. Data can also be interacted with from the Matlab workspace after saving the data. See the User's Guide for graphical information on how to use the HypPo program.

3) What is a HypPo object?

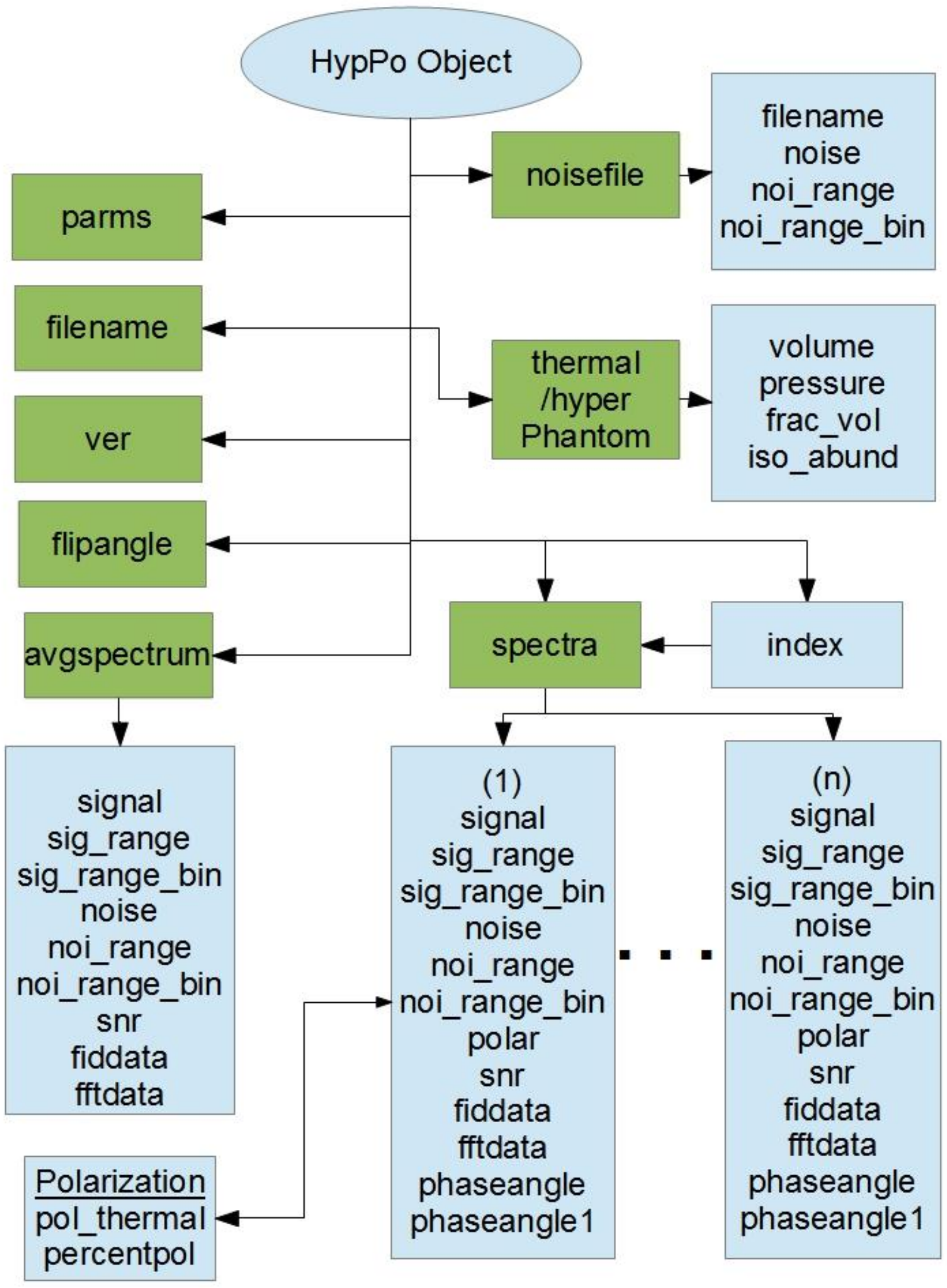
A HypPo object is a Matlab object that contains all the spectroscopy data, including data derived using the data tool. It also has a number of built in methods for interacting with fields in the object. This is the current list of built in functions. Refer to hypData.m for implementation details.



```
Methods for class hypData:

addlistener        getFID             getSigRangeAvg     notify             setSNRAvg
clearCalcs         getFlipAngle       getSigRangeAvgBin  setAveragedData    setSigRange
clearCalcsAvg      getNoiRange        getSigRangeBin     setDatapoints      setSigRangeAvg
correctPhase       getNoiRangeAvg     getSignal          setFlipAngle       setSigRangeAvgBin
delete             getNoiRangeAvgBin  getSignalAvg       setNoiRange        setSigRangeBin
eq                 getNoiRangeBin     getSpectrum        setNoiRangeAvg     setSignal
findobj            getNoise           gt                 setNoiRangeAvgBin  setSignalAvg
findprop           getNoiseAvg        hypData            setNoiRangeBin     setSpectrum
ge                 getPhaseAngle      identify           setNoise
getAllFFT          getPolar           isvalid            setNoiseAvg
getAveragedData    getSNR             le                 setPhaseAngle
getDatapoints      getSNRAvg          lt                 setPolar
getFFT             getSigRange        ne                 setSNR
```

The Hyppo object also has a large number of fields which can be represented by this diagram:



The HypPo object resides in the file hypData.m. For an example of a hypData object, see exampleHypObj.mat in the Documentation folder. HypPo objects can easily be extended through the addition of new attributes and methods. For example, we will add a field 'X' that is relevant to each spectrum in a data acquisition. X will store a floating point number for the purpose of this experiment. Opening hypData.m, the new attribute can be added in the top part of the file.

```
properties
  …

  X = -1.0;
  …

end
```

Next, to add a set up functions to interact with X we will go to the 'Get' and 'Set' portions of the file. `varargin` is used so that the value of a specific spectrum can be set/get, otherwise the current index (as found in obj.index) will be used.

```
methods
  …
     function x = getX(obj, varargin)
         if isempty(varargin)
             i = obj.index;
         else
             i = varargin{1};
         end
         x = obj.spectra(i).X;
     end
  …

    function obj = setX(obj, x, varargin)
         if isempty(varargin)
             i = obj.index;
         else
             i = varargin{1};
         end
         obj.spectra(i).X = x;
    end
  …

end
```

To interact with out new field we need an instantiation of our object, which we can get by reading a SPAR/SDAT file using getData(), or by calling it with no arguments which will create a dummy object. SPAR/SDAT complex data is automatically 4x zero padded, which can be changed in the hypData object.

Reading in a SPARS/SDAT file:

```
newObj = hypData(getData());
```

Creating a dummy object:

```
newObj = hypData();
```

Once a new object is created, X can be view by it 'getter' function.

```
X = newObj.getX();
```

If X is required for the nth spectrum in the data file, an index can be passed in

```
X = newObj.getX(n);
```

X can also be set in a similar fashion, where the index 'n' is dependent on whether you want to modify the current spectra or not

```
newObj = newObj.setX(newX, n)
```

HypData.m takes a structure as input and constructs the object from fields in the structure. The input structure is of the form:

```
datastruct = struct('data', spectroscopy_data, ...
                    'parms', file_parameters, ...
                    'filename',filename, ...
                    'ver', filetype, ...
                    'raw', 1);
```

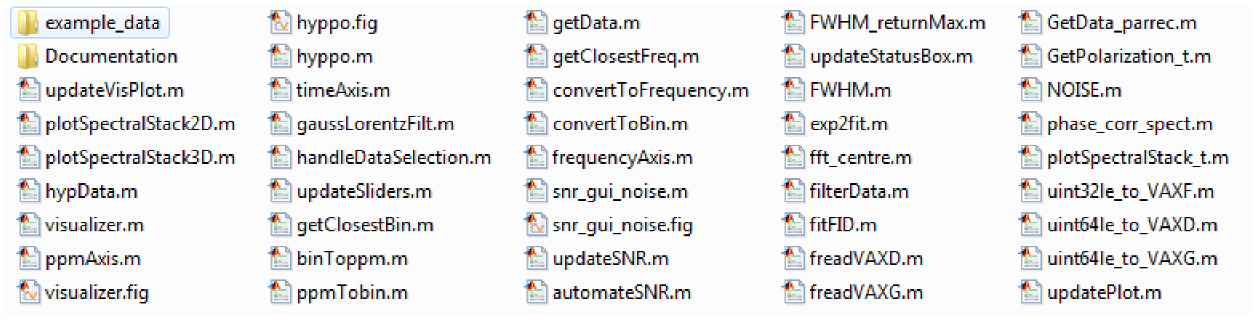
Where parms is a structure containing information about the scan (see GetData.m for the parms template), and also stores data added by hypData such as the zero padding factor and dimensions (number of acquisitions and number of samples per acquisition). The currently implemented fields of parms are:

parms.synthesizer_frequency	
parms.sample_frequency	
parms.padfactor	(default: 4)
parms.nucleus	(choices: `129Xe, 3He, 1H, 13C, 31P, 19F`)
parms.NSA	(number signal averages)
parms.samples	(number of datapoints per acquisition)
parms.rows	(number of acquisitions per experiment)

Data will contain an array of spectroscopy acquisitions. The raw field could be used to differentiate between versions of structures passed in (e.g. raw = 2 could be used for a file format other than SPAR/SDAT)

4) Modifying/Extending HypPo

In order to modify HypPo, it's important to know what functions are most important. Here is a list of all files associated with HypPo as of v0.8



example_data	hyppo.fig	getData.m	FWHM_returnMax.m	GetData_parrec.m
Documentation	hyppo.m	getClosestFreq.m	updateStatusBox.m	GetPolarization_t.m
updateVisPlot.m	timeAxis.m	convertToFrequency.m	FWHM.m	NOISE.m
plotSpectralStack2D.m	gaussLorentzFilt.m	convertToBin.m	exp2fit.m	phase_corr_spect.m
plotSpectralStack3D.m	handleDataSelection.m	frequencyAxis.m	fft_centre.m	plotSpectralStack_t.m
hypData.m	updateSliders.m	snr_gui_noise.m	filterData.m	uint32le_to_VAXF.m
visualizer.m	getClosestBin.m	snr_gui_noise.fig	fitFID.m	uint64le_to_VAXD.m
ppmAxis.m	binToppm.m	updateSNR.m	freadVAXD.m	uint64le_to_VAXG.m
visualizer.fig	ppmTobin.m	automateSNR.m	freadVAXG.m	updatePlot.m

In most cases you will be interacting with updateVisPlot, which deals with updating and storing the currentDpts as well as plotting them. currentDpts is a field that contains the currently plotted set of datapoints, and is the field that is used in all current calculations.

Calculations are done on the fly in updateVisPlot in a series of checks that cover the state of the program. The addition of new functions can be inserted in the checks at the appropriate point to ensure the correct calculations are performed, and hence the correct data is plotted.

For extending file formats that can be used with HypPo, model the output of your file reader after GetData() (more details in previous section).

The file handling interface is stored in hyppo.m/hyppo.fig. The main graphical interface functions of the visualizer are stored in visualizer.m, with the layout stored in visualizer.fig. Both interfaces were created in Matlab GUIDE.

5) Learning Resources

These are links to content that I found helpful when creating this program.

For Beginners:

http://blogs.mathworks.com/videos/category/matlab-basics/

http://www.mathworks.com/company/events/webinars/wbnr31351.html?id=31351&p1=961663781&p2=961663799 (Video, 1 hour) http://www.ck12.org/book/Engineering%253A-An-Introduction-to-Solving-Engineering-Problems-with-Matlab/

Object oriented programming in Matlab:

http://www.mathworks.com/discovery/object-oriented-programming.html

http://www.mathworks.com/company/events/webinars/wbnr33332.html?id=33332&p1=525981310&p2=525981328 (Video, 1 hour)

Matlab Webinars:

http://www.mathworks.com/products/matlab/webinars.html

Matlab GUIs:

http://blogs.mathworks.com/videos/category/gui-or-guide/

Good Reading:

http://www.matlabtips.com/matlab-and-blogs/

Also included is a script that can be used to find segments of code within the program. The scripts (findcode.py) is a python script and the directory to look in and the code to find can be modified from within the script. It will display what file and line number the code segment appears on and can be a useful script for familiarizing yourself with the program.