GPI Users Guide (Core Nodes)

Contents

Core Node Documentation

The following sections contain node documentation which can also be found in the 'about' widget of each node menu. The purpose of this document is to supply the user with a complete list of GPI nodes (and their descriptions) that are packaged as part of the 'core' library.

core.display.Audio

```
Converts numpy arrays into .wav format for reproduction on the audio
    system,
    writes to /tmp directory, then plays .wav file.

INPUT:
1 dimensional real-valued array to be converted to audio.
Data values are normalized internally.

WIDGETS:
Sample Rate (samp/sec) - D/A dwell time of waveform
# Loops - enter a number > 1 to play multiple times
Play - write waveform and play audio

KNOWN ISSUES:
None
```

core.display.CrossSection

```
Display image of 2D array. Dragging across image with left mouse button
    produces a graph of signal value along that line.

INPUT:
2D array

WIDGETS:
Viewport - displays image
  Double clicking on Viewport area brings up a scaling widget to change
     the image size, and change graphic overlay

L W F C - (hidden by default - double click on widget area to show
    sliders)
        Adjust value-to-pixel brightness mapping using Level/Window or
            Floor/Ceiling
Cross Section - hidden until line is drawn on image, then a graph of
    data values along line
```

core.display.DataQuery

```
A module for slicing, cropping and masking n-D numpy arrays.

INPUT - input array
```

OUTPUTS: output array – output is similar to center / width C/W option in reduce_GPI.py
Data values are displayed in the textbox

WIDGETS:
I/O info: – shows size of input, output arrays, print data values specified by the user
Dimension[i]
C/W – sliders select the center and width of cropping range along the ith dimension

Compute – generate sliced/cropped data and display output values in the textbox

core.display.dictionquery

Display contents of Dictionary–type data

INPUT
Data of type DICT

WIDGETS
Keys: comma–delimited list of dictionary elements to display
Range: allows inspection of a subset of large data arrays
C/W – sliders control center/width of range
B/E – sliders control begin/end of range
Slice – slider gives element to inspect
Pass – shows all elements

core.display.GLViewer

Prototype 3D GL Viewer
MOUSE EVENTS:
Left Button: translate object
Middle Button: move light source
Scroll wheel: zoom
Right Button: rotate (hold down shift key for different axis of rotation)

INPUT:
GL Object List

OUTPUT:
2D ARGB (stored as 3D array, with last dim of length 4, uint (byte) data for 0–255 per channel)
This is an image of what is rendered in the Viewport, useful for gluing together images to make movies

WIDGETS:
Viewport

Poly-Fill Line/Point - toggles between showing polygons, lines, or points (depends on how objects are created)
PolySmooth - for smoothing polygons, implemenation will vary
AntiAliasing - this option needs to be debugged further
Pixmap - shows the rendered pixmap. In this area, the scene is manipulated as follows:
Left Mouse Button Drag: translate object
Middle Mouse Button Drag: translate light source
Middle Mouse Wheel Scroll: Zoom
Right Mouse Button Drag: Rotate Object about 2 axes.
Hold down Shift key BEFORE selecting Right mouse button to change 2nd axis of rotation

Reset - resets rotation, translation, zoom to original values (x axis Horizontal, y axis vertical, z axis through-plane)

Compute/Nudge - sometimes needed to update renderer

Set Output - makes output port continually reflect image rendered in Viewport as ARGB data

KNOWN ISSUES:
Still needs to be tested on many platforms, bugs likely
On Linux platform, Z axis location occasionally seems to be improperly interpreted

core.display.ImageCompare

2D image Compare Module

INPUTS (must be the same size):
3D uint8 ARGB data (e.g. from ImageDisplay)
3D uint8 ARGB data (e.g. from ImageDisplay)

OUTPUT:
3D data of displayed image, last dimension has length 4 for ARGB byte (uint8) data

WIDGETS:
Transition: chooses how to transition between images of left and right ports
LeftRight: Toggles between left or right images
edge: slider to demarcate the line, or fading, between two images

core.display.ImageDisplay

2D image viewer for real or complex NPYarrays.

INPUT:
2D data, real or complex
3D uint8 ARGB data (e.g. output of another ImageDisplay node)

OUTPUT:
3D data of displayed image, last dimension has length 4 for ARGB byte (uint8) data

WIDGETS:
Complex Display – If data are complex, allows you to show Real, Imaginary, Magnitude, Phase, or "Complex" data.
If C (Complex) is chosen, then pixel brightness reflects value magnitude, while pixel color reflects value phase.
If input data are real–valued, this widget is hidden

Color Map – Chooses from a number of colormaps for real–valued data. Not available if Scalar Display is "Sign"

Edge Pixels – only visible for complex input data with Complex Display set to "C"
Setting this to N creates an N–pixel color ring around the image border illustrating the phase–to–color mapping

Black Pixels – only visible for complex input data with Complex Display set to "C"
Setting this to N creates an N–pixel black ring around the image border (but inside the Edge Pixels ring) to
separate the Edge pixel ring from the actual data image

Viewport – displays the image
Double clicking on Viewport area brings up a scaling widget to change the image size, and change graphic overlay

L W F C – (hidden by default – double click on widget area to show sliders)
Adjust value–to–pixel brightness mapping using Level/Window or Floor/Ceiling

Scalar Display – visible for real data, or complex data with "Complex Display" set to R, I, M, or P
Pass uses the real data in the data–to–pixel mapping
Mag uses the magnitude data in the data–to–pixel mapping (i.e. affects negative values only)
Sign will display positive values in green, and the absolutele value of negative values in magenta

Gamma – changes gamma of display function. Default value of 1 gives linear mapping of data to pixel value
pixel values refect value of data^gamma

Zero Ref – visible for real data, or complex data with "Complex Display" set to R, I, M, or P
also invisible if "Scalar Display" set to Sign

```
This is used for the data-to-pixel value mapping
--- maps the smallest value to black, and the largest value to white
0-> maps zero to black, and the largest value to white. All negative
    numbers are black
-0- maps zero to middle gray, with the largest magnitude set to black
    (if negative) or white (if positive)
<-0 maps zero to white, and the most negative value to black. All
    positive numbers are white

Fix Range
  If Auto-Range On, the data range for pixel value mapping is rescaled
    whenever new data appears at input
  If Fixed-Range On, the data range is fixed (and can be changed using
    Range Min and Range Max)

Range Min - shows minimum data value used for mapping to pixel values
  This value can be changed if (and only if) "Fix Range" is set to "
    Fixed-Ranged On"

Range Max - shows maximum data value used for mapping to pixel values
  This value can be changed if (and only if) "Fix Range" is set to "
    Fixed-Ranged On"
```

core.display.ImageRate

```
Module to compare images in a blinded and randomized manner.
Input Ports:
Input_List port requires a list of numpy arrays.
   Each numpy array should have the following dimensions:
      [different reconstructions to be compared (2 or 3); number of
         slices; x resolution; y resolution]
   To combine different numpy arrays to a list copy the following code
      into a custom module:
      in1 = self.getData('in1')
      in2 = self.getData('in2')
      # if in1 is a list, append in2 to the list, if in1 is not a list,
         combine them to a list
      if type(in1) is list:
        in1.append(in2)
        out = in1
      else:
        out = [in1, in2]
      self.setData_ofPort('out1', out)
previous_analysis_array port is an optional port to load a previously
   stored analysis array (numpy array)
   to continue the interrupted work.

Widgets:
image comparison: select whether the image above the button is much
   better, better, or the same as the other image.
```

Use much better for obviously better images.
Use better if you require the Toggle button to detect small differences.
Use same if the quality is the same even after using the Toggle button.
If image quality rating is disabled, then the next slice will be displaced after pressing a button.

image quality: select the image quality of the image above as either excellent, good, diagnostic, or non-diagnostic.
This widget may not be visible in case the "Enable image quality rating" button is set to "Off".

Toggle: Toggle button to switch the image on the left with the image on the right.

current slice: Shows how many slices have been analyzed so far.

out of slices: Shows the number of slices to analyze.

Undo button: Press this button to undo the last selection made.

Enable image quality rating: The image quality rating widget is only visible if this button is set to On.
This button is only visible if the current slice is 0 (zero).

Output Ports:

image_for_display port is a 2-dimensional numpy array that should be connected to an ImageDisplay module.

analysis_array: The numpy array with the results of the analsys.
Use SaveNPY to store the results. Make sure to store every so often as a backup of your work.
You can load the stored backup by using LoadNPY and connecting to previous_analysis_array input port.
Data are stored as follows:
[nr_slices, 4/7]
per slice up to 4/7 (number of reconstructions 2/3)4/7 (number of reconstructions 2/3)4/7 (number of reconstructions 2/3)4/7 (number of reconstructions 2/3) values are stored.
0: comparison reconstruction 0 and 1:
0: reconstruction 0 is much better than reconstruction 1
1: reconstruction 0 is better than reconstruction 1
3: reconstruction 0 is the same as reconstruction 1
4: reconstruction 1 is better than reconstruction 0
5: reconstruction 1 is much better than reconstruction 0
1: image quality of reconstruction 0 determined when comparing with reconstruction 1
1: excellent
2: good
3: diagnostic
4: non-diagnostic
2: image quality of reconstruction 1 determined when comparing with reconstruction 0
1: excellent
2: good
3: diagnostic

```
        4: non-diagnostic
    n.a./3: comparison reconstruction 0 and 2:
        0: reconstruction 0 is much better than reconstruction 2
        1: reconstruction 0 is better than reconstruction 2
        3: reconstruction 0 is the same as reconstruction 2
        4: reconstruction 2 is better than reconstruction 0
        5: reconstruction 2 is much better than reconstruction 0
    n.a./4: image quality of reconstruction 0 determined when
        comparing with reconstruction 2
        1: excellent
        2: good
        3: diagnostic
        4: non-diagnostic
    n.a./5: image quality of reconstruction 2 determined when
        comparing with reconstruction 0
        1: excellent
        2: good
        3: diagnostic
        4: non-diagnostic
    3/6 (last): patient number
```

core.display.Matplotlib

```
A Qt embedded plot window using the code from:
http://matplotlib.org/examples/user_interfaces/embedding_in_qt4_wtoolbar
    .html
keyboard shortcuts can be found here:
http://matplotlib.org/users/navigation_toolbar.html#navigation-keyboard-
    shortcuts

INPUTS
Up to 8 data sets can be plotted simultaneously
  1D real-valued data are plotted as graph
  2D data where the 2nd dimension is 2 will be plotted as X-Y parametric
      plot, otherwise
  all other 2D data are plotted as series of 1D plots
```

core.fileIO.ReadCSV

```
Reads comma separated values in ASCII text to an NPY array.
```

core.fileIO.ReadHDF5

```
Reads HDF5 files using the h5py project libs.

NOTE: This is a simple reader for getting 'dataset' objects from an HDF5
    file.
Since the files can be built in many different ways, this should be
    considered
a good starting point for writing code to read your specific format.
    More
```

general use cases will be added in future releases.

core.fileIO.ReadImage

Read images into GPI as numpy arrays

OUTPUT: Numpy array read from image file

WIDGETS: I/O Info – Gives info on data file and type
File Browser – button to launch file browser, and typein widget if the pathway is known
Gray Scale – button to flatten the last dimension of jpg and png images.
If toggle is on, output is a grey–scale float32 array with the same dimensions as the original image
If toggle is off, output is a uint8 array with the last dimension BGRA

core.fileIO.ReadNPY

Read arrays that were written as Numpy files

OUTPUT: Numpy array read from file

WIDGETS:
I/O Info – Gives info on data file and data type
File Browser – button to launch file browser, and typein widget if the pathway is known.
Squeeze – option for squeezing data, which removes all dimensions of length 1 (all data preserved)

core.fileIO.ReadPhilips

A module for reading Philips MR Scanner data from .data/.list, .lab/.raw/.sin, or .par/.xml/.rec files

OUTPUTS:
data: MR data from file
noise: noise measurement data
phc: phase data for EPI correction
user_parms: scan information written to .txt file – requires appropriate addition to methods code
header: information from Philips header files (.list, .sin, .lab, .par, .xml)

WIDGETS:
I/O Info – Gives info on selected file, data type, data dimensionality and dimension labels.
File Browser – button to launch file browswer, and typein widget if the pathway is known.
data/list only options:
ky chop – turns on/off ky chopping (multiplying data in even–numbered Y phase encodings by –1)

```
    kz chop - turns on/off kz chopping (multiplying data in even-
       numbered Z phase encodings by -1)
    PROPELLER TSE - select when reading PROPELLER TSE data sets
    PROPELLER GRASE - select when reading PROPELLER GRASE data sets
Re-scale Type: - used for par/rec files only
  FP = floating point
  DV = display value
  SV = stored value
Apply Corrections: - applies the following raw data corrections (lab/raw
   /sin only)
  Profile dependent amplification
  Random phase
  Measurement phase
  Not performed:
    DC offset correction
Execution Type - indicates if this is module is run as thread, process,
   or apploop
Read Param Only - set to read header only

NOTES:
    1) This currently reads .data/.list, .lab/.raw/.sin, and .par/.xml/.
       rec files.
    2) Tested acquisitions:
       For .data/.list
        Works:
         FFE-2D (includes 2 averages)
         FFE-MS
         FFE-3D
         FFE-MS-PartialEcho
         TFE-MS
         TSE-MS-Propeller (use PROPELLER TSE button)
         bTFE-MS-cardiacTriggered
         DWI-SEsshEPI-MS
         fMRI-FFEsshEPI-MS-10dynamics
         qFlow-T1TFE-2D-cardiacTriggered
         SE-MS
         SEsshEPI-MS
         SEmshEPI-MS
         T1FFE-2D-2echomDixon
         T1FFE-3D-2echomDixon
         TSE-MS (PROPELLER TSE button must be turned off)

        Doesn't work:
         bTFE-MS-cardiacRetroscpective (uses rtop dimension, and will
            require rebinning of data)

       All of the above work for .par/.xml/.rec
```

core.fileIO.ReadPhysioLog

Reads Physiology file generated by Philips Ingenia Scanner.
Separates text file into 11 channels:
0: v1raw – for ECG recording.
1: v2raw – for ECG recording
2: v1 – Smoothed ECG trace
3: v2
4: ppu – Pulse oximeter
5: resp – Respiratory Bellows
6: gx
7: gy
9: gz
10: mark – Scan markers. Includes scan start, scan stop, data acquisition markers, etc.

Each line in the physiology file represents 2 microseconds (500 Hz sampling rate).

NOTE: This is as simple case for reading physio–waveform files and should
be thought of as a start for implementing code specific to the needs of your application.

core.fileIO.ReadPickled

Provides an interface to the python pickle (cPickle) module for de–serializing py–objects from a file.

OUTPUT: Numpy array read from file

WIDGETS:
I/O Info – Gives info on data file and data type
File Browser – button to launch file browser, and typein widget if the pathway is known.

core.fileIO.ReadRaw

Reads raw data into several c–types: float, double, int, and char.
OUTPUT: Numpy array of data

WIDGETS:
I/O Info: information about file
File Browser: file browswer
Skip Bytes: number of bytes past which to start reading data (e.g. size of header)
<type>: type of data in file
ndim: Number of dimensions to fill (of output array)
Dimension N: number of elements in the Nth dimension
Compute: compute

core.fileIO.WriteCSV

A module for writing NPY arrays to a comma separated value list in ASCII text.

INPUT: Numpy array to write to file

WIDGETS:
File Browser – button to launch file browser, and typein widget if the pathway is known.
Write Mode – write at any event, or write only with new filename
Write Now – write right now

core.fileIO.WriteHDF5

Uses the HDF5 h5py project interface for writing arrays.

INPUT – numpy array to write

WIDGETS:
File Browser – button to launch file browser, and typein widget, to give pathname for output file
Write Mode – write at any event, or write only with new filename
Write Now – write right now

NOTE: This is a simple writer for writing 'dataset' objects from an HDF5 file.
Since the files can be built in many different ways, this should be considered
a good starting point for writing code to write your specific format. More
general use cases will be added in future releases.

core.fileIO.WriteImage

Uses the numpy save interface for writing arrays. File types are specified in the bottom of the file browser.

INPUT – Image array to write. This node is designed to write to file the output of the ImageDisplay node.
Image can be MxNx3 or MxNx4
If the shape is MxNx3, the array will store the RGB bands as the last dimension
If the shape is MxNx4, the array will store RGBA as the last dimension

WIDGETS:
File Browser – button to launch file browser, and typein widget, to give pathname for output file
***NOTE: If no extension (filetype) is specified, the input will be written as a .png file.
Automatically appending the filetype based on the chosen file filter (in the Save File dialog box)

has yet to be implemented.
Write Mode – write at any event, or write only with new filename
Write Now – write right now

core.fileIO.WriteNPY

Uses the numpy save interface for writing arrays.

INPUT – numpy array to write

WIDGETS:
File Browser – button to launch file browser, and typein widget, to give pathname for output file
Write Mode – write at any event, or write only with new filename
Write Now – write right now

core.fileIO.WritePickled

Implements the python pickle (cPickle) module for serializing py–objects and writing to a file. Uses HIGHEST_PROTOCOL.

INPUT: numpy array

WIDGETS:
File Browswer – Browse button to launch file browser, or typein widget for entering path for file to be written
Write Mode – write at any event, or write only with new filename
Write Now – write right now

core.fileIO.WriteRaw

A module for writing NPY arrays to a raw data file (no header)

INPUT: Numpy array to write to file

WIDGETS:
File Browser – button to launch file browser, and typein widget if the pathway is known.
Write Mode – write at any event, or write only with new filename
Write Now – write right now

core.generators.Bool

Specify a python boolean for use as node–data or widget–ports–parms.

OUTPUT – boolean value

WIDGETS:
bool – specifies whether boolean is True or False

core.generators.Float

Specifiy a python float for use as node-data or widget-ports-parms.

OUTPUT - float value

WIDGETS:
float - float value to output

core.generators.GLObjects

A basic module for generating GL object descriptions using GPI format.
These objects can be accumulated by creating a string of GLObjects nodes, with
the output of one node fed to the input of the next node. The final list of GL objects
can be viewed using the GLViewer

INPUTS:
GL Object List - optional input takes the output of another GLObject module, for concatenating all objects in a list
Crds - optional input of k-space coordinates, as a numpy array, for use with the "trajectory" GL Objects function.
The last dimension must be 3, corresponding to kx/ky/kz

OUTPUTS - GL Object List

WIDGETS:
GL Objects - type of GL Object to create
Self Evident, hook up to GLViewer to display. Clip Plane must be combined with a GL Object to observe
Color - changes hue of object
Subdiv - for sphere and cylinders, specifies how many planar surfaces to approximate curve
Pos X, Pos Y, Pos Z - X, Y, Z coordinates for center of object
Rot X, Rot Y, Rot Y - specifies rotation of object
Multiples - allows one to create more than one instance of object, at pseudo-random locations
IF GL Objects = Sphere:
Radius - Radius of Sphere
IF GL Objects = Cylinder:
Base, Top, Height - specifies lower and upper diameter of cylinder and its height
IF GL Objects = Axes:
Radius - length of axes
Tube Radius - Diameter of axes cylinders
IF GL Objects = Text:
Text - string to display
IF GL Objects = Trajectory or Random Trajectory:
Tube Radius - diameter of cylinders used to show trajectory
IF GL Objects = Clip Plane:
Plane No. - Specify which of 6 clipping planes are defined

core.generators.Integer

Specifiy a python integer for use as node-data or widget-ports-parms.

OUTPUT - integer value

WIDGETS:
int - integer value to output

core.generators.List

Add objects from the input port to a python list. Used for saving multiple serializable objects or for combining numpy arrays.

INPUT - any type: List concatenates to a growing python list

OUTPUT - python list

WIDGETS:
Clear List - clear python list at OUTPUT

core.generators.Shapes

Geometric function generator (used to make different shapes in 1, 2, or 3 dimensions).

INPUT: Optional input - data are not used, but the shape of this array (ndim and array shape) are used for the output array

OUTPUT: 1D, 2D, or 3D real-valued numpy array

WIDGETS: (some of these parameters change in special-use cases)
Dimensions - specify whether output data is 1D, 2D, or 3D
Function - type of Geometric function to create
Compute - turn off to change parameters without generating new data (turn on to generate data)
Dimension: i - for the ith dimesion:
size is the length of that dimension for the output data
width is (usually) the width of the Geometric Function, <= size

Note for some Functions, only Dimension: 1 is shown even for 2D and 3D dimensions,
implying that the output data will have the same size/width in all dimensions

Veclen: Size of last dimension, if > 1. In this case, the Geometric Function is copied along this dimension

Pass Value: Maximum value in function
Stop Value: Value in data outside of function

Window: allows one to taper function, creates a linear ramp between the pass and stop value.
Window units are in %, from 0 (creating a sharp edge) to 100 (tapering from the edge to the middle of the object)

Some Special Use Cases:
For Sinc function, width specifies the width of the main lobe. Window allows for additional tapering of the edges
For Poly, one can specify the 0th, 1st, and 2nd order coefficients of an N-dimensional polynomial
For Noise, one can specify the Noise standard deviation. Everytime the module is poked, it generates a new pseudo-random instance of noise

core.generators.SheppLogan

Generates a Shepp-Logan phantom of the designated size.

core.generators.String

Specifiy a python string for use as node-data or widget-ports-parms.

OUTPUT - string value

WIDGETS:
string - string value to output

core.gridding.DegridDFT

Inverse Gridding module for Post-Cartesian Data using DFT - works with 2 D data
This is to create exact k-space data corresponding to any image, often for simulation and testing

INPUTS:
image - k-space complex data - if not supplied, Grid uses "1" for all of its data
coords - k-space coordinates, normalized in units of "1/resolution", i.e. ranging from -0.5 to 0.5
Last dimension must be 2 or 3 (corresponding to kx/ky or kx/ky/kz, respectively)
Coordinates at the very edge of gridded k-space then have values of -/+ 0.5

OUTPUTS:
out - gridded data, same dimensions as coords

WIDGETS:
Eff Mtx - effective matrix of coords (specifies Nyquist distance)

core.gridding.Grid

```
Gridding module for Post-Cartesian Data - works with 2D and 3D data

INPUTS:
data - k-space complex data - if not supplied, Grid uses "1" for all of
    its data
coords - k-space coordinates, normalized in units of "1/resolution", i.e
    . ranging from -0.5 to 0.5
            Last dimension must be 1, 2 or 3 (corresponding to kx, kx/ky,
                or kx/ky/kz, respectively)
            Coordinates at the very edge of gridded k-space then have
                values of -/+ 0.5
weighting - sampling density correction - if not supplied, Grid uses "1"
params_in - optional dictionary from (e.g.) spiralcoords, to
    automatically specify the effective matrix

OUTPUTS:
gridded data, which is M+E dimensions, where
            M is 1, 2 or 3, depending on last dimension of coords
                input (i.e. 1D, 2D, or 3D)
              The size of these M dimensions is 1.5 times the given
                effective matrix
            E is (# input data dims) - (Dims per set widget value)
              E represents slices, coils, etc., which don't get
                gridded together.

WIDGETS:
Dims per Set - How many dimensions get gridded into the same space (and
    the space is determined by the last dim of coords)
              Remaining dimensions are independent, e.g. for slices,
                coils, etc.
Eff MTX XY - number of pixels in the final image (XY), nominally (
    without zero-padding) given by FOV/resolution
  Add 25% to matrix for "true resolution" for (e.g.) spiral
  Output data are gridded to a matrix 50% larger than this to mitigate
    gridding artifacts
Eff MTX Z - number of pixels in the final image (Z), nominally (without
    zero-padding) given by FOV/resolution
  Add 25% to matrix for "true resolution" for (e.g.) stack of cones,
    spherical distributed spiral, FLORET
  Output data are gridded to a matrix 50% larger than this to mitigate
    gridding artifacts
dx, dy, dz - for off-center FOV correction.  Specify number of pixels in
    each direction to shift (in image space)
            prior to gridding.

  Note on Input dimensions: If coords is N dimensions, with the last
    used for the 2D/3D information,
```

```
                                    1) weights must have N-1 dimensions, of
                                       the same shape as corresponding coords
                                    2) data can have N-1 or more dimensions (
                                       of correct shape)
                                    3) If data has extra dimension (the first
                                       dimensions), data from each index
                                         in these dimensions are gridded using
                                             the same coords and weights

  Examples: 2D Spiral Gridding, 4-channel coil, 22 slices, 32 arms,
     4056 points per interleaf
               data are complex, 4 x 22 x 32 x 4056
               coords are real,      22 x 32 x 4056 x 2
               weights are real,     22 x 32 x 4056
               Dims per Set widget is 2
               Eff Mtx XY widget is 200
               output data size is 4 x 22 x 300 x 300

            3D Distributed Spiral Gridding, 8 coils, 320 arms, 4056
               points per interleaf
               data are complex, 8 x 320 x 4056
               coords are real,      320 x 4056 x 3
               weights are real,     320 x 4056
               Dims per Set widget is 2
               Eff Mtx XY widget is 160
               Eff Mtx Z widget is 120
               output data size is 8 x 180 x 240 x 240
```

core.gridding.Rolloff

```
Implements rolloff correction in-house 2D gridding module written in C
   ++.
This module corrects the image shading created by the Grid module and
   crops the data by 1/3
in each Gridded dimension to produce an image of the right matrix size

INPUT: complex data - typically the output of Grid, Fourier Transformed
   to image space
       These data can represent 1D, 2D or 3D data, with optional extra
          dimensions representing (e.g.) coils or slices

OUTPUT: complex data after Rolloff Correction

WIDGETS:
Num Rolloff Dims - Set to 1, 2, or 3 corresponding to 1D, 2D, or 3D
   gridded data sets
                   Remaining dims are treated independently, e.g. as
                      slices, coils, etc.
Isotropic FOV - multiplies data by a circular/spherical mask for 2D/3D
   data
```

core.gridding.Rolloff_Comp

Implements rolloff correction in-house 2D gridding module written in C++.
This module corrects the image shading created by the Grid module and crops the data by 50%
in each Gridded dimension to produce an image of the right matrix size

INPUT: complex data - typically the output of Grid, Fourier Transformed to image space
These data can represent 1D, 2D or 3D data, with optional extra dimensions representing (e.g.) coils or slices

OUTPUT: complex data after Rolloff Correction

WIDGETS:
Multiple Sets - if this is selected, then the first dimension is presumed to be the (e.g.) coil or slice dimension,
so the rolloff correction (and 50% size reduction) are not performed.
Isotropic FOV - multiplies data by a circular/spherical mask for 2D/3D data

core.gridding.SDC

Computes Sampling Density Correction for 2D and 3D waveforms

INPUTS
crds - input coordinates, which range from -0.5 to +0.5.
the last dimension is 2 or 3, corresponding to 2D (kx/ky) or 3D (kx/ky/kz)
wates - optional relative weights, used to preferentially use some data over others in areas of overlap
params_in - optional dictionary from (e.g.) spiralcoords, to automatically specify the effective matrix

OUTPUTS - sampling density, same size as crds (minus the last dimension)

WIDGETS:
computenow - turn off to change effective matrix without computing the sampling density
Dims per Set - how many of the dimensions to compute together as 1 set of coordinates. Allows for extra
dimensions for (e.g.) slices, diffusion weightings, whatever
Iterations - times to iterate algorithm
Effective Matrix XY - FOV/resolution in X&Y, which indicates the width of data correlation in k-space
Effective Matrix Z - FOV/resolution in Z, which indicates the width of data correlation in k-space

```
Taper - taper the weights at the edge of k-space; value indicates the
    fraction of k-space radius to
        (linearly) taper from 1 to 0 (at the very edge): 0 gives no
            taper, 1 is "full" taper
```

core.interfaces.Alert

```
Warn the user when this node has received data. Either thru terminal
bell or OSX voice synth. The voice settings can be changed in the
    System Preferences.
```

core.interfaces.Custom

```
This node provides a simple code input interface and editor for
generating Python code to be executed in the node compute() on-the-fly.
The editor includes syntax highlighting. The four InPorts provided are
labeled: 'in1', 'in2', 'in3', and 'in4' (similarly for the OutPorts).
    This
code is run in a Python exec statement and therefore has associated
limitations (e.g. return statements). New widgets cannot be added to
    the
widget interface, however, packages that contain UI elements can be
    spawned
using the Execution-Type 'Main Loop'.
```

core.interfaces.Facebook_Twitter

```
A node for sharing data, code, and the joy of GPI with others.
-keep others up to date with step-by-step tweets of your exec-flow.
-ping your blogosphere with the latest data analysis.
```

core.interfaces.Template

```
A basic module for tutorial purposes. This module contains all
stock widgets and types so that the node-developer can preview and read
auto-doc information about each object.
```

core.interfaces.www

```
Browse the web for data, search your local subversion repo for code
hints, or update the GPI wiki.
```

core.iterate.AutoNum

```
A visual for-loop that produces floats and integers for a predefined
    limits and steps.
Module will output an number, wait for all other modules to run, then
    output the next integer
Also allows generation of uniformly distributed random numbers, and
    manually-entered numbers
A second "bound" mode is employed when input port is populated for
    spanning the bounds of a given dim.
```

INPUTS:
in_array: When populated, user specifies dimension, and can automatically or manually change index
in the range of that dimension

OUTPUTS:
int_out – integer output, given as round(float_out)
dict_for_reduce – passes the widget type necessary to hook IntegerLoop up to Reduce widget InPort
float_out – floating point output

WIDGETS:
Dimension – when in_array populated, lets user pick dimension to for step to span
Randomize – output value is either RANDOM or LINEAR, the latter = Minimum + step*(Step Size)
LOOP – when selected, step runs from 0 to "Number of Steps"–1
Minimum – value corresponding to step=0 for LINEAR mode, or minimum range of output for RANDOM mode
Maximum – value corresponding to last step for LINEAR mode, or maximum range of output for RANDOM mode
Step Size – change in value per step for LINEAR mode
Number of Steps – number of iterations between Minimum and Maximum
step – current index
Value – current value presented at output, either a random number between Minimum and Maximum for RANDOM mode
or equal to Minimum + step*(Maximum–Minimum)/("Number of Steps" – 1)

core.iterate.ConjugateGradient

A node that allows cyclic connections and provides a countdown for iterative algorithms using the conjugate gradient approach. The code tries to
match the nomenclature of the excellent article by Jonathan Richard Shewchuk,
An Introduction to the Conjugate Gradient Methods Without the Agonizing Pain
and in particular equations 45–49 on page 32 of that manuscript.

The node will automatically step through a specified number of iterations

So we are solving for Ax = b
INPUTS: The first four inputs are initialization, the last one is for iteration
d_in – initial guess for direction to move, equals b–Ax (Eq. 45)
r_in – initial guess, same as d_in, equals b–Ax (Eq. 45)
x_in – initial guess for x

Ad_in – matrix A times d_in
Adloop_in – used during iterations, this takes A multiplied by d_out (below)

OUTPUTS:
d_out – multiply this vector by A and send back to Adloop_in
r_out – not really used, just FYI
x_out – the final answer

WIDGETS:
Current Iteration – lets you know which iteration is being performed
Max Iteration – enter the desired number of iterations
Start/Stop – click on to start, click off to stop before iterations are done
Step – click to step through one more iteration
Reset – click to reset iteration count and begin with initial conditions

core.iterate.IntegerLoop

A visual for–loop that produces integers a predefined limits and steps.
Module will output an integer, wait for all other modules to run, then output the next integer
It will repeat this for the specified number of steps

OUTPUTS:
step_out – steps from the specified min to specified max over number of steps
rand_out – outputs a random integer between Minimum and Maximum
dict_for_reduce – passes the widget type necessary to hook IntegerLoop up to Reduce widget InPort

WIDGETS:
Minimum – starting value for step_out
Maximum – ending value for step_out
Number of Steps – number of iterations between Minimum and Maximum
Value – current value of step_out
Textbox – gives stepsize, value of step_out, and value of rand_out
ON/OFF – toggles starting and stopping the Looping process

core.iterate.Iter

A node that allows cyclic connections and provides a countdown for iterative algorithms.

INPUTS:
n+1 – input for iterations, typically data taken from output "n" and processed
n_0 – initial condition

OUTPUT:
n – value of data at nth iteration

WIDGETS:
Current Iteration – reports current iteration index
Max Iteration – enter number of desired iterations
Start/Stop – starts/stops iterations
Reset – resets iteration count to zero and data to initial condition (at input n_0)

core.math.Calc101

Integrate or Differentiate along a given axis

INPUT – numpy array to operate on

OUTPUT – numpy array after operation

WIDGETS:
Operation – select Integerate or Differentiate operation
Dimension – select axis along with to integerate or differentiate

core.math.Collapse

Collapse data along selected dimension

INPUT – input array

OUTPUTS:
out1 – output array, with 1 less dimension than input array (collapsed version)
out2 – single value from taking operation on entire array, active when "Collapse All" is selected

WIDGETS:
Status, Info – information boxes
Operation – selected method of collapse
Min, Max, Mean, Std. Dev, Sum, RMS, Prod, Median – self–evident
Energy – sum of squares along dimension
SWA – Energy/Sum
Max Val Index – index along dimension at which the maximum value occurs
Geo–Avg – Nth root of Prod, where N is the size of the collapse dimension
Dimension – dimension along with to collapse
Compute – compute
Span Entire Dimension – select whether to collapse along the entire span of specified dimension
Dimension Start_Index – if Span Entire Dimension is off, lets you pick index of where collapse starts
Dimension Stop_Index – if Span Entire Dimension is off, lets you pick index of where collapse ends
Collapse_All – when off, module collapses along single (specified) dimension, output array at out1

```
            when on, module collapses entire data set, output value at
               out2
```

core.math.Compare

```
    Node to get compare 2 numpy arrays

    INPUT - Two data sets to be compared - they must be the same size and
       type

    OUTPUT:
    diff - Difference image (L - R) after optional normalization

    WIDGETS:
    Info: gives RSS Difference and Dot Product after optional normalization
          define D = L - R
          define A* as an array with conjugate element values of the array A
          define A A* as the element-wise product of A and A*
          RSS Difference = sqrt(sum(D D*)), where the sum is over all
             elements
          Dot Product = sum(L R*), where the sum is over all elements

          When data are normalized, the Dot product is the correlation (
             assuming 0 mean)

    Normalize: Divide each array A by sqrt(sum(A A*)), where A* is the
       conjugate of A, and the sum is over all elements
          this effectively makes each array have "unit length"
```

core.math.DiffRMS

```
    Node to get compare RMS difference between data sets. Data set on right
        is scaled to minimize RMS diff

    INPUT - Two data sets to be compared - they must be the same size and
       type

    OUTPUT:
    diff - Difference image after scaling
    scaled_rhs - scaled version of right input data

    WIDGETS:
    Info: gives RMS Error and Scaling Factor
```

core.math.FFT_NUMPY

```
    Fast Fourier Transformation of N-dimensional data via scipy library.
    Cropping and zero-padding only work on transformed dimensions.

    INPUT - data to be transformed, can be real or complex. DC is assumed
       to be at index N/2 (starting at 0)
```

```
OUTPUT - transformed data, complex. DC is at index N/2 (starting at 0)

WIDGETS:
Dimension i - button turns off/on tranform in ith dimension
              factor and length are redundant parameters, length is the
                 output dimension size
              factor = length/(input dimension size)
              factors < 1 result in cropping data before transformation
              factors > 1 result in zero-padding before transformation
compute - compute
direction - select whether you want a Forward or Inverse FFT
```

core.math.FFTW

```
A module that implements the FFTW C++ package.
Cropping and zero-padding only work on transformed dimensions.

INPUT - data to be transformed, can be real or complex. DC is assumed
   to be at index N/2 (starting at 0)

OUTPUT - transformed data, complex. DC is at index N/2 (starting at 0)

WIDGETS:
Dimension i - button turns off/on tranform in ith dimension
              factor and length are redundant parameters, length is the
                 output dimension size
              factor = length/(input dimension size)
              factors < 1 result in cropping data before transformation
              factors > 1 result in zero-padding before transformation
compute - compute
direction - select whether you want a Forward or Inverse FFT
```

core.math.Float_Math

```
A node to do float math

INPUT: input float

OUTPUT: output float

WIDGETS:
For two inputs - math operations - add, subtract, multiply, divide, mod
   and power
For single input - math operations - add, subtract, multiply, divide,
   mod, power by scalar and do absolute, exponent, square root
```

core.math.Interpolate

```
A node to linearly interpolate in specified directions.
```

```
INPUT: input numpy array

OUTPUT: interpolated data in the specified directions.

WIDGETS:
Dimension[n]:  For each dimension,
  factor - (desired output length)/(input length)
  length - desired output length
Compute - compute
```

core.math.Math

```
Perform real or complex scalar operations on a per element basis.
Three Modes of Operation:
 1) Standard - basic Arithmetic and exponential operations.
 2) Trigonometric - basic Trigonometric operations.
 3) Comparison - returns maximum, minimum, or bit mask based on
                 comparison between inputs or against Scalar.

Operations which do not commute (e.g. divide) operate left to right, e.g
   .:
  output = (left port) / (right port)
```

core.math.phaseUnwrap

```
Unwraps the data of a numpy array containg phase along the specified
   dimension.

Widgets:
Dimension: dimension to unwrap along
Units:  choose degrees or radians
```

core.math.Regression

```
A module for doing least squares fitting of input data to a user
defined model.  Fitting is always done along the first dimension.

INPUTS:
dataIn - data to fit (i.e. "Y" values, assume "X" values are linear)
coordsIn - "X" values corresponding to "Y" values of dataIn, if they are
    not linear

OUTPUTS:
fit - the fit curve
coefficients - coefficients of the fit polynomial
residual - the input data minus the fit curve

WIDGETS:
Mode - choose type of fit
Info - some instructions for use
For Mode = Polynomial Fit
```

```
  Polynomial order - 0 fits to a constant, 1 fits to a line, 2 fits to a
      parabola, etc.
For Mode = Generalized Linear
  data are fit to a linear sum of functions_which_depend_on_x
  # Model Parameters - number of functions to fit (number of
     coefficients to calculate)
  fi(x) - the model for the ith function, written as a python function
     of x.  Examples include:
         1                  (i.e. just a constant)
         np.power(x,3)      (fit to x^3)
         np.sin(x)          (fit to sin of x)
For Mode = Generalized Non-Linear
  See Info box (with this Mode selected) for detailed information
```

core.math.RIMP

```
Output the Real, Imaginary, Magnitude, or Phase of a Complex array,
   element-wise
INPUT - complex data array

OUTPUT - real-valued array

WIDGETS:
R I M P - select Real, Imaginary, Magnitude, or Phase
Units - for Phase, select Degrees, Radians, or Cycles
Unwrap - for Phase, you can do simple phase unwrapping in one direction
Unwrap Dimension - for Phase, choose the dimension for phase unwrapping
```

core.math.Rotate

```
A node to rotate data about a plane specified by two axes

INPUT: input numpy array

OUTPUT: output numpy array

WIDGETS:
Theta is entered in degrees
Output can be reshaped to contain the input after rotation
Modes of rotation - constant, nearest, reflect and wrap can be chosen
Order for interpolation can be specified.
```

core.math.Statistics

```
Generate Statistics for numpy array

min,max,mean,and std are reported in the Data Statistics Text box, also
   via separate Outputs
```

core.math.ValueBounds

```
Clamp, Threshold, or Scale values
Pass does not operate on the data, but sets the min and max widgets to
    reflect data min and max
Clamp takes all values above(below) the max(min) and sets them to the
    max(min)
Threshold takes all values above(below) the max(min) and sets them to
    zero
Scale "Min" OR "Max" multiplies the data by a constant for the desired
    min/max
Scale "Min" AND "Max" multiplies the data by a constant and adds an
    offset for the desired min & max
For complex numbers, the operations work on the magnitude, while
    preserving phase
```

core.math.Zeropad

```
Fast Fourier Transformation of N-dimensional data via scipy library.


INPUT - data to be zeropadded or sinc-interpolated, can be real or
    complex.  DC is assumed to be at index N/2 (starting at 0)

OUTPUT - Return Sinc interpolated data - zero-padding or cropping is
    applied in the transform domain.

WIDGETS:
Dimension i - button turns off/on tranform in ith dimension
              factor and length are redundant parameters, length is the
                  output dimension size
              factor = length/(input dimension size)
              factors < 1 result in cropping data before transformation
              factors > 1 result in zero-padding before transformation

compute - return sinc interpolated data after FFT forward the input data
    -> zero-pad -> FFT backward
```

core.shapers.Combine

```
Combine two data sets along any dimension.
e.g. put two images side-by-side

Several restrictions are in place requiring arrays to be of the same
    size;
  this module should be updated to be more flexible in the future
```

core.shapers.Dimensions

```
Peform operations on an array that generally change its shape and order,
    but not so much the actual data
INPUT - input array
OUTPUT - output array
```

WIDGETS:
Info: - information on size, etc of input and output array - very useful to pay attention to this
Operation:
Reshape - give new dimensions to file (must be same total size), while data stay in same order in memory
Combine - combine 2 dimensions, output array will have one less dimension
Split - split a dimension into two dimensions, output array will have one extra dimension
Extend - add an extra dimension; if length of that dimension > 1, the data will be copied along that dimension
Transpose - transpose any dimensions
Flip - mirror the data along any dimension (can flip in multiple dimensions)
Shift - shift data along any dimension, filling in with zeros
CircShift - circularly shift data along any dimension, wrapping the data from the outgoing edge back around
Tile - Separate a dimension so that it is tiled in the other dimensions - may not work in all cases (useful for 3D data)

core.shapers.Extend

Inserts an extra dimension of length 1,
to the beginning of the shapes array.

core.shapers.Glue

Node to serially append input data to output data in specified direction

INPUT: input numpy array

OUTPUT: data containing glued-together pieces of input data

WIDGETS:
Autoadd: when true, any new incoming data automatically gets glued to output data
Add Now: when pressed, existing data at input port is glued to output data
Glue Dimension: dimension in which to glue input data together
A negative one here means a "new" dim 0, not the typical python -1 wrapped dim
Once the gluing starts, this cannot be changed until output data are cleared
Glue Dim Size: Reports back size along the gluing dimension
Clear Output: Press twice in a row to clear output data (must press twice!).

core.shapers.Recast

Change the NPY array data type (dtype) from any type to int<n>, uint<n>, float<n> and complex<n>, where 'n' represents the numbe of bits.

core.shapers.Reduce

A module for slicing, cropping and masking n-D numpy arrays.

INPUT - input array

OUTPUTS: note the "out" port is the data you typically want, while the "mask" port shows where that data came from
out - data after cropping or slicing - size may be different than input array
mask - copy of input array (same size), but data replaced with zeros wherever data were cropped/sliced

WIDGETS:
I/O info: - shows size of input, output arrays
Dimension[i]
C/W - sliders select the center and width of cropping range along the ith dimension
B/E - sliders select the beginning and end of cropping range along the ith dimension
Slice - slider selects the index to slice along the ith dimension
Pass - ith dimension is passed (not affected)
Mask - generate data for "mask" output - good to leave off for large data sets if not needed
Compute - generate sliced/cropped data

core.shapers.Slicer

A module for slicing through numpy arrays.
INPUT - input
OUTPUT - sliced data

WIDGETS:
I/O Info: - size of input, output arrays
Dimension# - dimension along which to slice
Slice# - index along that dimension to slice

core.shapers.ToComplex

Node to read either real and imaginary or magnitude and phase data and convert to complex data

INPUT:
inLeft: input numpy array
inRight: input numpy array

OUTPUT:
out: output numpy array

WIDGETS:
Operation: Determines what is done
If both input ports (L and R) are populated
L+iR – combines the two fields of same size to complex
data inLeft – Real
data inRight – Imaginary
L exp(iR) – combines the two fields of same size to complex
data inLeft – Magnitude
data inRight – Phase
If a single input port P is populated
Real – output complex data with input's real channel and all zeros in the imaginary channel
Imag – output complex data with input's imaginary channel and all zeros in the real channel
Phase – output complex data with magnitude = 1 and phase given by input
Vec2Cmplx – if the last dimension is 2, change this to R/I

Phase: pick degrees or radians when appropriate

OutType: single or double precision complex type

core.spinSim.Bloch

A Bloch simulator designed to work with the Spyn–node and optional RFwaveforms–node input. Bloch nodes can be cascaded to effectively build
segments of a pulse sequence.

The data are passed from the output of either the Spyn Module (the beginning) or
the output of a Bloch module to the left output (M_in) of the next Bloch module

INPUTS:
M_in – spin data
G_in – input gradient waveform – if 1 dimensional, this is used for all gradients
if 2 dimensional, with the 2nd dimension 3, this is used for separate Gx, Gy, Gz waveforms
Waveform is linearly interpolated to match the duration of the Bloch module
if not connected, gradients are all considered constant for Duration
RF_in – input RF waveform – if 1 dimensional, this is used for RF magnitude
if 2 dimensional, with the 2nd dimension 2, this is used as specified by the RF Waveform widget

Waveform is linearly interpolated to match the duration of the Bloch module
if not connected, rf is considered constant for Duration
Pars_in – accepts parameters to set other widgets (i.e. if external waveform is used

OUTPUTS – spin data

WIDGETS:
Duration: duration of this particular part of the sequence
Gx, Gy, Gz – amplitude of gradients. If an external waveform is supplied,
it is multiplied by these respective values
RF Mag (uT) – amplitude of RF pulse. If an external waveform is supplied,
it is multiplied by this value. If RF Flip is set, RF Mag is adjusted to
give that flip at the center frequency.
RF Flip (deg) – flip angle of RF pulse at the center frequency.
If RF Mag is set, RF Flip is adjusted accordingly.
RF Phase (deg) – phase of RF pulse
Crusher Tc (ms) – this is used to simulate a crusher, although not too realistically.
The (Mx,My) component of all spins decay with an additional exp(−t/Tc) during
this module. If Tc=0, Crusher is ignored.
RF Waveform: – specifies the mode of the input 2D RF waveform

core.spinSim.MRPSD1D

Reformats pulse sequence information from Bloch– (potentially cascaded Bloch–) node(s) for plotting in 1D viewer/plotter nodes.

core.spinSim.MRSig1D

Reformats magnetization profiles defined in Spyn and Bloch nodes for plotting in 1D viewer/plotter nodes.

INPUT: data from Bloch or Spyn node

OUTPUT: data to plot (typically via Matplotlib)

WIDGETS:
M display – select which values to display
(Mx, My, Mz, Mt=sqrt(mx^2+my^2), |M| = sqrt(mx^2+my^2+mz^2)
selecting multiple buttons will produce multiple graphs
Average Across: for each dimension (e.g. T1, T2, etc.), the magnetization can be averaged across that dimension
X Axis – the independent variable by which to plot on the abscissa
(cannot be a dimension that has been averaged across)
(T,T1,T2,FQ,...) Index –

for every dimension that has length > 1, is not averaged across, and is not used as the X Axis, there is an slider to indicate which index along that dimension to use.

core.spinSim.RFwaveforms

Generate RF waveforms from Philips source code, designed for use with Bloch module in GPI SpinSim

OUTPUT:
G_out – generated slice-select waveform to pass to accompany RF waveform in Bloch module
RF_out – generated RF waveform for Bloch module
Pars_out – relevant parameters for RF waveform and gradient to pass to Bloch module

WIDGETS:
Function – choose RF functionality, Excite, Refocus, or Invert
RF Shape – choose from list of Philips RF waveforms read from mgobjrfvarrcd.h
Time*BW – reports time-bandwidth of pulse
RF Flip (deg) – desired flip angle, in degrees (changes when new RF Mag is entered)
RF Mag (uT) – peak magntidue of RF pulse, in uT (changes when new flip angle is entered)
RF Dur (ms) – desired duration of RF Pulse
Slice Thickness (mm) – target slice thickness
GR Strength (mT/m) – reported slice-select gradient strength
Add rephase gradient – when on, an rephasing gradient is added corresponding to isocenter of RF

core.spinSim.SpinViz3D

Reformats spin magnetization profile data generated by Spyn and Bloch nodes for visualization in an OpenGL viewer node.

INPUT: MR Spin info from Bloch or Spyn module

OUTPUTS:
Spin Objects – graphical objects showing spins (send to GLViewer)
PS Waveform Objects – graphical objects showing Gradient and RF waveforms (send to GLViewer)

WIDGETS:
Time Index – select the time point to display. The duration per point is set by the Spyn module

*** The following modules affect the Spin Objects List
A/B/C Axis – select the variable to spread along the A/B/C axis
(A/B/C are the x', y', z' axes in the GLViewer, color coded as R/G/B, respectively)

```
A/B/C Axis stagger - select the distance between spins along the A/B/C
    axis
Show - select a few options (can select multiple buttons)
  Axes - show the actual A/B/C axes (color coded as R/G/B)
  Text - label each axis with the variable displayed on that axis
  Mx=0 - display spin vectors without the Mx component
  My=0 - display spin vectors without the My component
  Mz=0 - display spin vectors without the Mz component
Color Scheme - Select the color scheme for spin objects (spheres and
    cylinders)
  Off - white
  M3 - Mx/My/Mz fed into the R/G/B channel
  MG - color coded based on spin magnitude
  MT - color coded based on transverse magnetization
  PH - color coded basd on phase
  MTPH - same as PH, with brightness proportional to transverse
      magnetization
Tip Connect - draw lines between the tips of spin vectors in the chosen
    direction
Vector Radius - relative radius of cylinders used for spins
Sphere Radius - relative radius of spheres at base of spin vectors
Tracer History - enables drawing lines between locations of each spin-
    tips last n positions (sparkler effect)
Vector History - enables drawing last n vectors

*** The following modules affect the PS Waveform Objects List
Waveforms - Select which waveforms to view
Waveform Stretch - a kludgy way to adjust the aspect ratio of the PS
    Waveform plot (affects X-axis spacing)
```

core.spinSim.Spyn

```
Module to generate initial spin profile, for bloch simulations in GPI
  Output data are 11-dimensional, with data stored as follows:
  Dimension 0 corresponds to base information such  M0, T1, T2,
      velocities, Frequency,
              Diffusion (not yet implemented), and the value of dt
  Dimension 1 corresponds to time.
              The first index of Dimension 1 stores the above info, and
              the 2nd index stores the Spin magnetization, position,
                  time,
                                        gradient, rf, and spoiler info
  These two dimensions are illustrated below:

D  ####### Dim 0 ->>> ##############################
i  #     0  1  2  3  4  5  6  7  8  9  10 11 12
m  #
1  # 0   M0 T1 T2 Vx Vy Vz Fq Dx Dy Dz dt __ __
|  # 1   Mx My Mz X  Y  Z  T  Gx Gy Gz Rx Ry Sp
v  # 2   Mx My Mz X  Y  Z  T  Gx Gy Gz Rx Ry Sp
```

```
v  # 3  Mx My Mz X  Y  Z  T  Gx Gy Gz Rx Ry Sp
v  # 4  Mx My Mz X  Y  Z  T  Gx Gy Gz Rx Ry Sp
v  ##################################################
```

Dimensions 2-10 correspond to the widgets below, from T1, T2, etc. to Vy and Vz

OUTPUT - spin state, typically fed to Bloch module. No data present until "Starting Spins"
widget is set to something other than "OFF"

WIDGETS:
Starting Spins - starting state of all spins. When set to "OFF", no data are generated
T0 (ms) - starting time. Valuable for (e.g.) having velocities (which is not yet implemented)
dt (us) - time between time points which will be recorded by Bloch module and pasted along the first dimension
smaller numbers produce finer time resolution but larger data sets
Note the actual numerical simulations occur with time resolution of 1us regardless of this value
For the remaining widgets, the following is true:
Spins - length along that dimension, with values from the specified "Start" value to the specified "End" value
if # Spins is 1, the value of that property is given by the "Start" parameter
T1 (ms) - dimension 2 variable. A value of 0 is infinity (no T1 recovery), otherwise values are in ms
T2 (ms) - dimension 3 variable. A value of 0 is infinity (no T2 relaxation), otherwise values are in ms
Fq (Hz) - dimension 4 variable, off-resonance frequency
X0/Y0/Z0 - dimension 5-7 variables, locations of spins
Vx/Vy/Vz - dimension 8-10 variables, velocity of spins (not yet implemented)

core.spinSim.T1calculator

Determine T1 based on the signal of two measurements with different repetition times
and/or flip angles.

core.spinSim.VERSE

A basic module for tutorial purposes. This module contains all stock widgets and types so that the node-developer can preview and read auto-doc information about each object.

core.spiral.SpiralCoords

Module to generate the gradient waveforms for a desired k-space waveform. Uses the core file bnispiralgen.c

```
If using this module to generate k-space coordinates for collected data,
    it is crucial that the code be identical
with that used in the methods code (hence we keep bnispiralgen.c as a
   standalone file)

INPUT: dictionary from (e.g.) ReadPhilips with all of the parameters
   used to collect data.  If input data are present,
       parameters cannot be changed.  If not, parameters may be changed
          freely.

OUTPUTS:
crds_out - output coordinates: the last dimension is either 2 (kx/ky) or
    3 (kx/ky/kz) for 2D and 3D trajectories.  See
          discussion of "true resolution" below for more details about
             the scaling.  Dwell time specified by AD dwell time widget.
grd_out - gradient waveforms used to produce crds_out.  Fixed to dwell
   time of 6.4 us.

WIDGETS: most widgets self-explanatory, here are a few clarifications:
stype - ARCH is standard 2D spiral
        CYL DST is a cylindrical distributed spiral
        SPH DST is a cylindrical distributed spiral
        FLORET is FLORET...
# of spiral arms - allows floating point entry
                   for ARCH this is rounded to integer
                   for CYL and SPH DST and FLORET, which share arms
                      across ky and kz, this is the effective arms per
                      plane
                   and can be a floating point number (giving greater
                      flexibility in choosing the total number of arms vs
                      .
                   readout duration)
usamp st (0-1) - the relative value of kr at which undersampling begins
   (0 at the center, 1 at the edge of collected k-space)
                 the samples are collected at the nyquist limit (1/FOV X
                    -Y) prior to that
usamp end (0-1) - the relative value of kr at which undersampling ends
                  the samples are collected at the R times the nyquist
                     limit (1/FOV X-Y) prior to that
max underample - R, i.e. the undersampling relative to (1/FOV X-Y) after
    kr exceeds usamp end
ustype - determines the progression of radial undersampling from usamp
   st to usamp end as linear, quadratic, or hanning
Max G Freq - limits the maximum frequency of the gradient waveform
   during the spiral readout.  If set to 0, there is no limit
T2 Match - allows one to exponentially change the radial undersampling
   with time to match T2* decay, for optimal, T2*-matched, SNR
Sloppy Sp. Per - changes the characteristic of the sloppy spiral, if
   desired (see Miki Lustig's perturbed spiral paper).  0 turns off.
gtype - amount of gradient to output
```

```
    SPIRAL ONLY - the gradient waveform during the readout only
    G -> 0 - Same as above, but with gradient rampdown
    M0 -> 0 - Same as above, but with k-space rewinder
    M1 -> 0 - Same as above, but with gradient first moment compensation
spinout - controls spiral in, out, etc.
    OUT - generate spiral-out waveform
    IN  - generate spiral-in waveform
    INOUT 180R - generate single waveform with spiral-in followed by
       spiral-out on a trajectory rotated by 180 degrees
    INOUT SAME - generate single waveform with spiral-in followed by
       spiral-out on the same trajectory

****************************************
*** The concept of "true resolution" ***
k-space data are typically normalized, for the gridding process, to
   values between -0.5 and 0.5
For spiral data (which measure circular k-space) to have the same
   resolution as Cartesian (square k-space)
they must measure a diameter of 2/sqrt(pi) ~ 1.13 larger than
   conventional k-space limits (so that the area of the circle
equals the area of the square). k-space coordinates, therefore, are
   multiplied by 0.8, so that there range of
0.8*2/sqrt(pi) = 0.903, or -0.451 to +0.451, fits within the gridded
   space. The resulting image, with no further
zero-padding, will have pixels that are 0.8 times smaller than the
   requested resolution, with a matrix 25% larger in each
dimension. This is a semi-complicated way of making sure that this
   works routinely, and is referred to by the authors
as "true resolution".

A similar logic is used for measuring data in a sphere, but the diameter
    must be the cubed root of (6/pi), or 1.24, the
side of an equivalent cube for equal volumes. k-space coordinates then
   span 0.8*(6/pi)^(1/3) = 0.993 of the unit width of
gridded k-space.
****************************************
```