Subject Array and Wrapper Automation (*sawa*) - version 2 -

theissjd@berkeley.edu

sawa is a toolbox that automatically wraps and integrates any command line, matlab, or batch function using subject-specific or iterative variable inputs.

Contents:

Highly Suggested:
-SPM (for batch editor and spm_select): http://www.fil.ion.ucl.ac.uk/spm/software/

-xlwrite (for mac users): http://www.mathworks.com/matlabcentral/fileexchange/38591-xlwrite--generate-xls-x--files-without-excel-on-mac-linux-win

-findjobj (for horizontal scrollbar in notes):
http://www.mathworks.com/matlabcentral/fileexchange/14317-findjobj-find-java-handles-of-matlab-graphic-objects

Suggested Citation:

Theiss, J.D. (2015). Subject Array and Wrapper Automation (*sawa*) [Computer software]. https://github.com/jdtheiss/sawa.

Introduction

The main components of *sawa* are the subject array and wrapper automation functionalities. The subject array consists of a structural array that can contain subject-specific information to be utilized in wrapping functions, especially useful for maintaining information for separate studies. Meanwhile, wrapper automation allows users to create complex pipelines with unix/C/matlab/batch utility commands by passing variables from one function to the next. At its simplest, *sawa* is an organizational tool that can maintain up-to-date information as well as record inputs/outputs for analyses. However, *sawa* is built to feed information from the subject array to a pipeline of various types of functions. As such, *sawa* provides users the ability to perform complex analyses using subject data and pass variables between SPM, AFNI, FSL, etc., or any unix/C/matlab commands.

With wrapper automation for batch utility, command line, and matlab functions, users can build simple pipelines for repeat analyses and combine batch, matlab, or command line functions. The batch editor directly uses the matlab batch utility system (https://sourceforge.net/p/matlabbatch/wiki/Home/), which allows users to directly choose variables that will be filled by the subject array or other functions/variables. When using a command-line function, *sawa* automatically displays help information and provides potential command switches. When using matlab functions, sawa again displays help information and allows users to select inputs. Where applicable, users are able to select outputs to return from the function that can then be entered to a later function. The inputs and outputs for each iteration and function are then printed when the user runs the pipeline.

Installation

sawa is available for download and is updated on github (see citation above for link). After you have downloaded the zipped file folder and moved it to your desired location, go into the "sawa/main/functions" folder and open "sawa.m" in matlab. Simply run the function, and *sawa* will be automatically be added to your matlab path. At this point, you may close the "sawa.m" file and begin using *sawa* through its graphical user interface (GUI) or from the command prompt.

Getting Started

Once you have installed and run the "sawa.m" file, you may begin using any of the utilities that *sawa* provides. *sawa* uses a graphical user interface (GUI) for simple point-and-click usage, but each *sawa* function can also be run from the command prompt as well as within other functions. By typing sawa into the matlab command prompt, the main GUI opens (Figure 1).

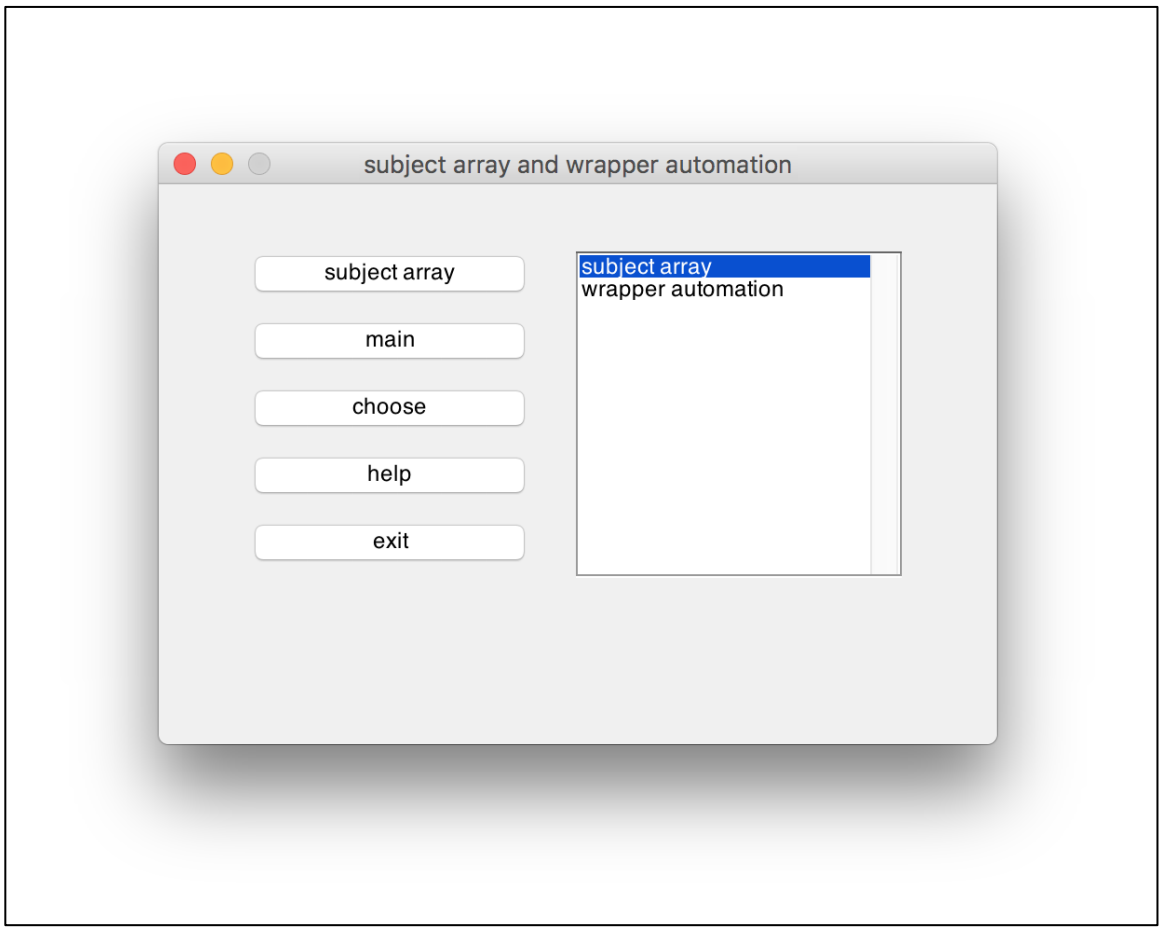


Figure 1. Main GUI page for *sawa* includes folders in "sawa" folder.

The list on the right side of the GUI displays the folders within the "sawa" folder. Users may then add other folders/functions to the same directory if they wish to use them in *sawa*. The "subject array" button allows users to choose a current subject array (a "subjects.mat" file that contains subject arrays). Initially, there will be no subject array to choose. The "main" button returns the user to the main folders whereas "choose" will reveal the contents of the selected folder in the list. After choosing a folder, the "choose" button will change to "add". Clicking "add" will load the function into the queue of functions to be run. When a function is selected in the list, pressing "help" on the main GUI page will display the function's help message in the command prompt. Finally, the "exit" button will quit *sawa*.

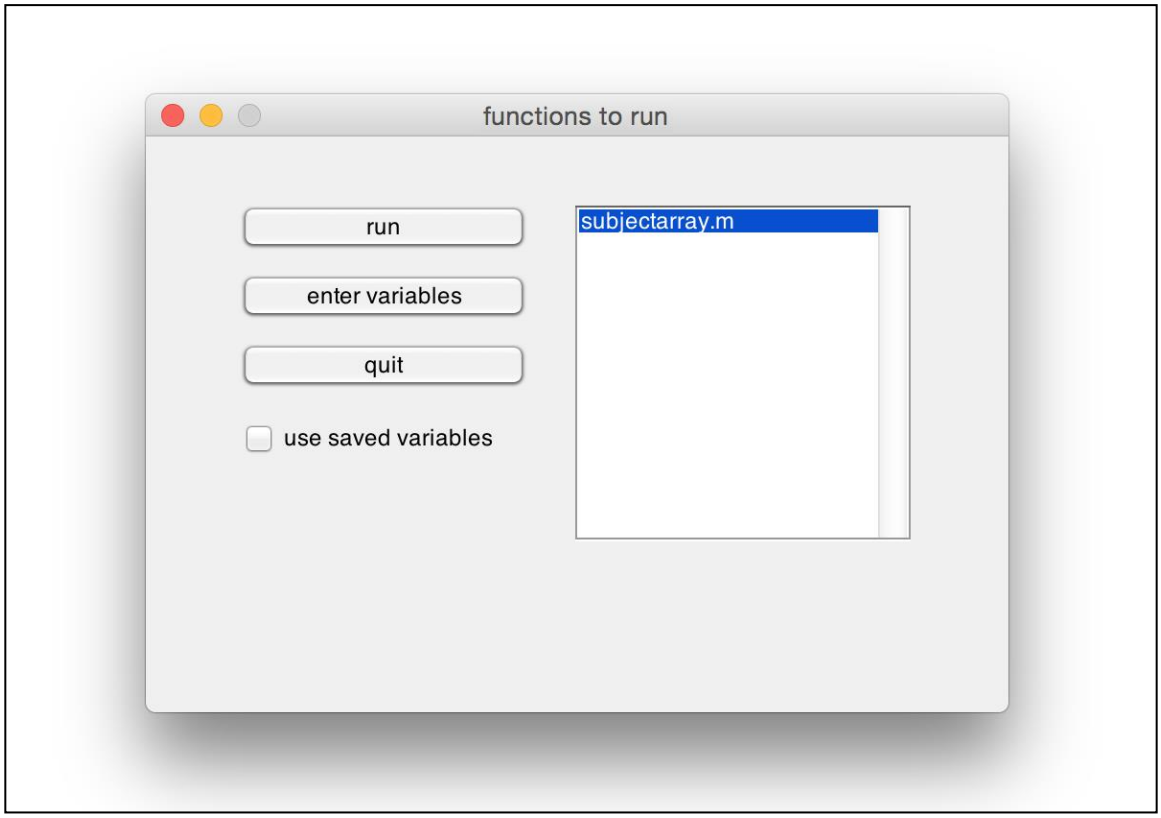


Figure 2. Functions to run GUI page for *sawa* includes any files within the previously selected folder.

After choosing a function to "add", the user is brought to the "functions to run" GUI page. The "functions to run" GUI page contains the list of functions that will be run, and new functions can by selecting them from the main GUI page and pressing "add". Pressing "run" will run each function in order. For the wrapper editors (i.e. batch editor, command line, and function wrapper), this will not automatically run since the user needs to input functions/variables to use. However, for functions that do not require variable input (i.e. a matlab script/function), the function will run immediately. The "enter variables" button allows users to enter the variables to input for each function prior to running. Once pressed, each function in the queue that can have its variables set will open for the user to input variables. This is useful for lengthy functions to avoid waiting for one function to finish before setting the next set of variables. The "use saved variables" checkbox allows users to choose previously saved variables (as "*_savedvars.mat") with the function. This is also how the "enter variables" button works in that each set of variables is saved to the "sawa/main/jobs" folder and automatically retrieved. Saved variables can also be set in the wrapper editors by selecting "save" after entering the desired functions and variables. Finally, the "quit" button will clear the queue of functions to run and close the "functions to run" GUI page.

Creating the Subject Array

Running the "subjectarray" function within the "subject array" folder, users are able to create or load a subject array (Figure 3). The subject array is a simple structural array for subject-specific information that is then used with the wrapper editors and other functions. Each subject array should have at least a "subj" and "subjFolders" field. The "subj" field contains the subject ID as a character array (e.g., 'Subject001'), and the "subjFolders" field contains the main folder location for each subject as a cellstring array (e.g., {'/Volumes/Subj001'}). Users may then add any other subject fields as they wish.

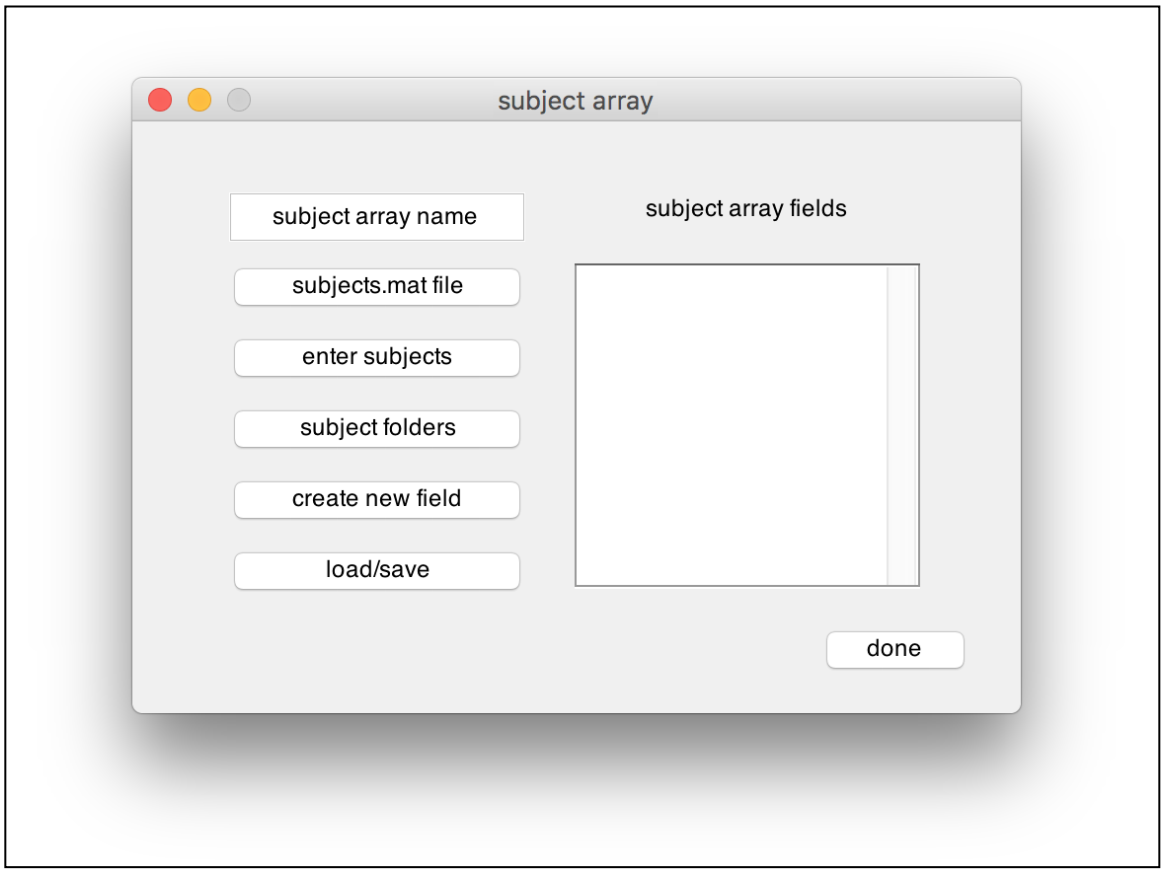


Figure 3. The subject array GUI allows users to create/load the subject array.

The "Subject Array Name" text box allows users to name the subject array (e.g., "ptsd" or "gonogo"). The subject array name is used to save the subject array in a "subjects.mat" file and is displayed when selecting a subject array to use. The "subjects.mat" file is simply the file that contains the subject array(s). As such, a user may choose to save a subject array to a previously created "subjects.mat" file or select a folder to save a new "subjects.mat" file. This is accomplished by pressing the "subjects.mat file" button. The "enter subjects" buttons allows users to add subjects to the subject array (and create the "subj" field). Pressing "subject folders" provides users the option to either choose a main folder that will be appended with each subject's subject ID or set manually. If choosing a main folder, the user can also choose to create each subject's main folder within the directory.

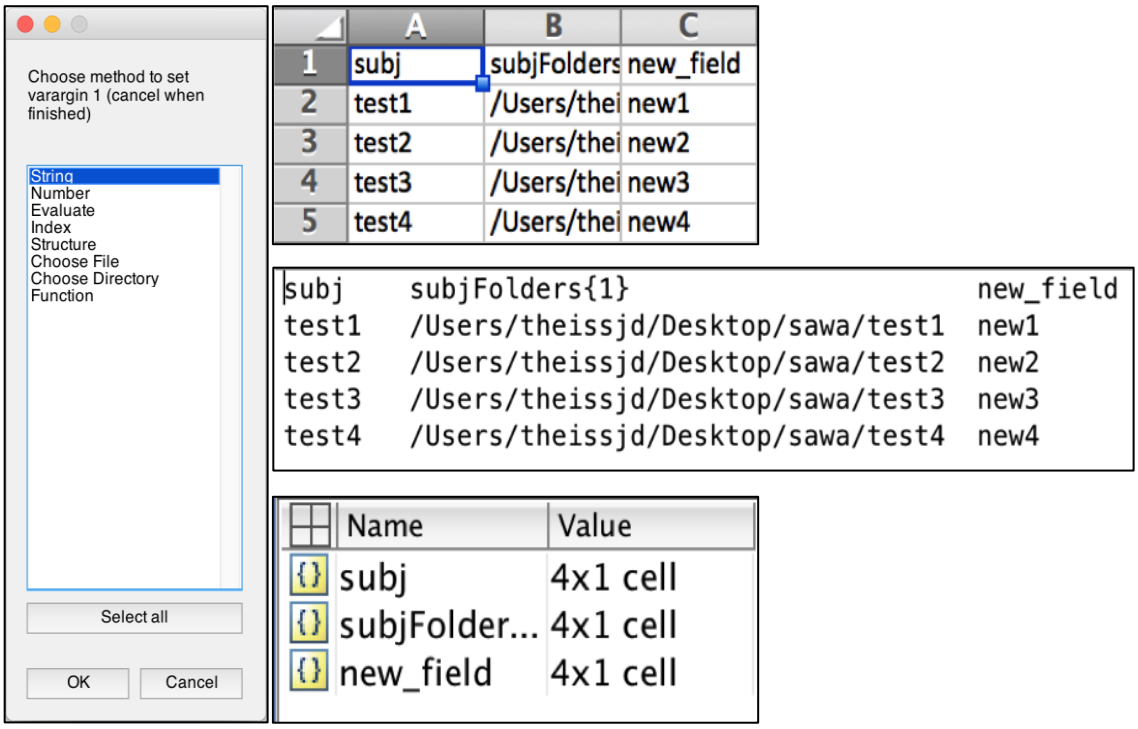


Figure 4. Methods to set subject arrays: manual (left), excel (right top), text (right middle), or .mat file (right bottom).

The "create new field" button allows users to enter a fieldname to add to the subject array and choose the values to be set for chosen subjects. Setting variables to a field involves choosing a method by which to set the variables (Figure 4). These methods are also the way in which users set variables for the wrapper editors as well. See the section "Creating Variables" on page 8 for more information. Once a subject array is created, an additional method called "Subject Array" is made available. With this method, users are able to select subject-specific variables to use to be set to the subject array. For example, one could easily set a "RestingState" field by selecting the "subjFolders" field then entering "/RestingState" as a subfolder.

To simplify the way in which a subject array is created, the "load/save" button can be used to load a subject array from any of the following files: excel, text, or .mat file. Additionally, a subject array can be loaded from a previously created "subjects.mat" file. For each file type, the way in which fields are set up is very similar. For excel files, the headers correspond to the fieldnames (Figure 4b, top). As such, if a field has a subfield/cell, it should be included in the name (e.g., "subjFolders{1}" no "subjFolders"). Each row below the headers then corresponds to a different subject. Similarly, a text file can be used as well (Figure 4b, middle). The same procedure is followed for text files and each field should be separated by a tab. Finally, a subject array can also be loaded from a .mat file with the fields of the .mat file corresponding to the fieldnames (Figure 4b, bottom). However, since .mat file fields cannot contain special characters (e.g., {1} or .subfield), the field should only contain the fieldname with the appropriate subfields reflected in the variable itself. For example, the "subjFolders" field would contain inner cells for each subject as below.

subjFolders = {{‘/Volumes/Subj001’};{‘/Volumes/Subj002’};{‘/Volumes/Subj003’}}

In addition to loading subject arrays, subject arrays can be written to excel/text/.mat files to continually update information in the subject array or the file. Once a subject array has been set up, simply choose “save” and then choose the type of file(s) to save the subject array to. At this point, the user can also create the “subjects.mat” file with the subject array. The location of the “subjects.mat” file will then be used for any further reference for the subject array.

Finally, users can edit/copy/delete subject array fields by clicking on the fieldname in the list on the right side of the subject array GUI. Note that deleting subject array fields will not remove the field from the subject array unless all subjects have empty field values. Otherwise, deleting for selected subjects will simply empty the field.

Creating Variables

When setting the input variable for any function, users will be asked to create the variable using one of several methods (Figure 4). For each variable, users can set multiple iterations or create different types of variables. Multiple iterations can be set per line (i.e. string/number/evaluate/choose file/choose directory methods), by index, by function, or by selecting a new method. The "Choose method…" window will reappear until the user has selected as many methods as desired.

String, Number, Evaluate

The first three methods ("String", "Number", and "Evaluate") all involve an input box. Each row of the input box corresponds to a different subject/iteration. Furthermore, each method corresponds to a different type of value. String values are made of characters (e.g., "Subject001"); number values are numeric (e.g., 12), and evaluated values are evaluated by matlab (e.g., {12} or strrep("Subject001","001","002")). Function handles (e.g., @(x)) can also be entered using the "Number" or "Evaluate" methods.

Index

The "Index" method allows users to create cell variables by entering the index (e.g., {1} or {2,1}) and then choosing a method to set for the index. This method is especially useful when entering multiple variables for a single iteration (e.g., four runs entered per subject in SPM preprocessing).

Structure

The "Structure" method allows users to create a structure array (e.g., Responses.Values). When this method is selected, the user choses the number of structures to create and enters subfield(s) to set. Then the user will choose a method to set the value within each subfield.

Choose File/Directory

The "Choose File" and "Choose Directory" methods will open a file/directory selection window (the same as SPM uses).

Function

The "Function" method allows users to create values based on the outputs of a matlab function. For example, the user could use the "strcat" function with inputs of "Subject" and "001", "002", "003" (one per row) to create "Subject001", "Subject002", "Subject003". Specifically, this method calls the auto_function utility.

Preceding Outputs

Finally, if previous functions in a pipeline have outputs, users can create a variable from one of those outputs. This will work for any function that can return an output. For example, if a user runs SPM preprocessing and wants to use the warped files (i.e. "w" files) in a new function, s/he can choose the specific output as the input for the later function. Note: if users are setting variables for consecutive batch functions, they should still use "set dependency" as output variables are only available after all of the sequential batch functions have completed.

Wrapper Automation

From the main GUI page, clicking "wrapper automation" and running "wrapper_automation.mat" will bring up the wrapper GUI (Figure 5). The same GUI is also loaded for any pipeline that is created. The right side of the GUI will contain the module/function names to be run. When module/function names are listed, the user may right-click on the name to copy/delete the module/function from the list. Modules/functions can be of three different types: batch, matlab, or command line functions. Specific information for each wrapper utility is described starting on page 11.

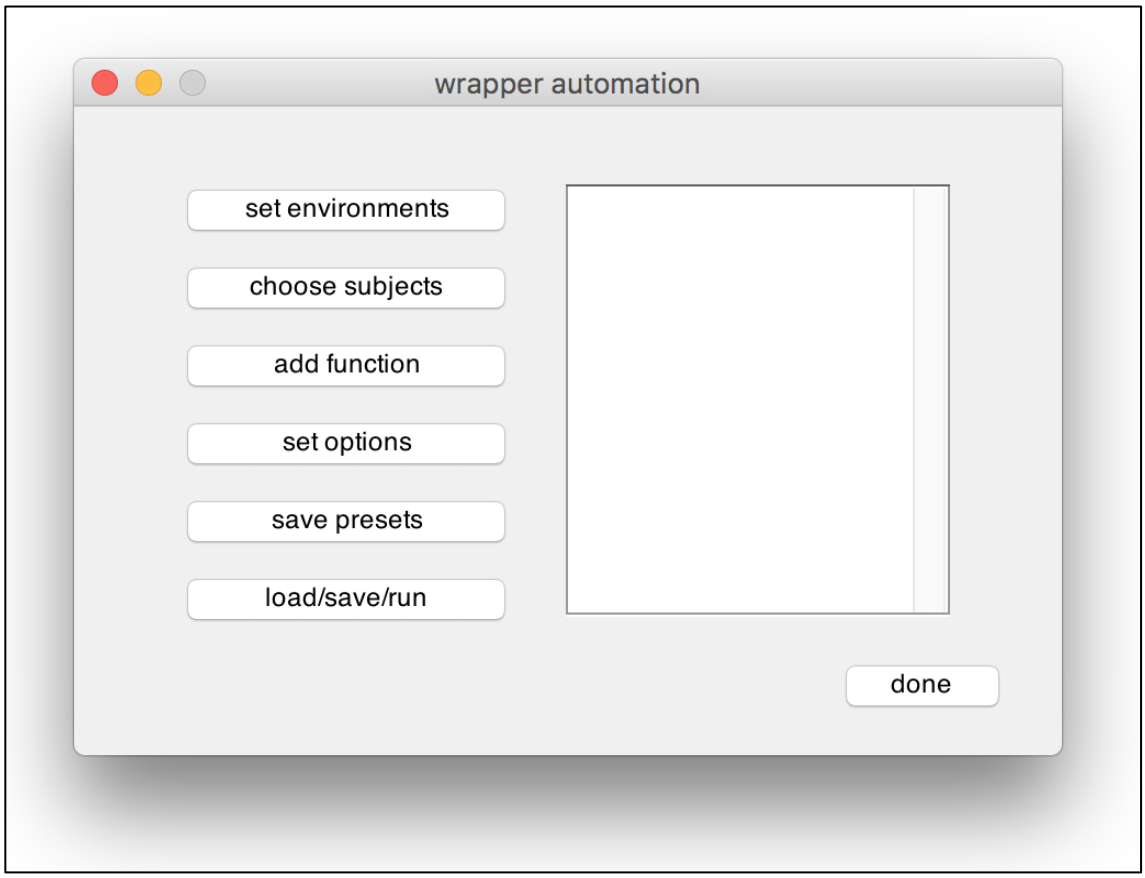


Figure 5. GUI for wrapper automation.

Set Environments

The first button, "set environments", is used for setting environments/paths for external programs/functions (e.g., FSL). Certain functions/programs, such as command line functions, have specific environments that need to be set in order to be run in matlab. This is also the way in which a user can add the path to a matlab function or java as well. If a set of functions is saved as a pipeline or job, the environments will automatically be set and unset each time it is run.

Choose Subjects

The "choose subjects" button allows users to select subjects to include and whether to run the functions as "per subject" or "per iteration". If iterations are chosen, the user will be asked to input the number of iterations. Note that if the "choose subjects" button is not

pressed, the number iterations will be assumed to be one. However, each function will still run for as many variables input (e.g., if 12 files input, run 12 times). Users may also choose subjects to use in the function while still selecting iterations. For example, one could run a two-sample t-test in SPM by selecting subjects that will be included then selecting "per iterations" and entering "1".

Add Function

The "add function" button will allow the user to add a function to the pipeline via one of three built-in editors (i.e., auto_batch, auto_cmd, auto_function). This allows the user to use batch, matlab, or command line functions in the same pipeline. For example, one could run the "Check Registration" function in SPM using the auto_batch editor followed by "sawa_screencapture" (a built-in sawa function) to save screenshots of check registration for chosen subjects. If users want to contribute their own wrapper editor, they should save the matlab function as "auto_[program]" within the /sawa/main/functions folder (see "Creating Wrapper Editor" for more information). This would then become an option to be chosen when clicking "add function". See information regarding each wrapper editor to learn more about how each works.

Set Options

The "set options" button allows users to set input variables (i.e. batch components, function inputs, command line options). When pressed, users will choose a method by which to set variables (See Figure 4 and page 6 for overview). It is important to note a few tips, however. When multiple subjects/iterations have been chosen (i.e. running the same function multiple times), there are two ways to set the variables for each subject/iteration. First, users may input one variable that will be applied to all subjects/iterations. For example, if one wanted to set the same value for ten iterations, choose "Number" then enter the value, click "Ok" to confirm the input, then cancel when the "Choose method" window reappears. This would set the variable for all ten iterations to the same value. Second, users may set the variable for each subject/iteration using a separate line or method. For example, the user could enter ten different values (one per line). Or the user could set each individually by choosing the "Number" method ten times. This method is most effective if a different method is used to set a variable for different iterations. Additionally, setting files/paths is another topic of note. Similar to the command line methods "dir" or "ls", a wildcard (i.e. *) can be used to choose file(s). For example, when choosing the method "Subject Array" and selecting a field such as "subjFolders", the user could then input a wildcard file path as the subfield e.g. '/Analysis/con_*.img'. This would select all files that start with "con_" and end with ".img" in each "Analysis" folder within each subject folder. Furthermore, directories can be created by placing a file separator (i.e. '/' or '\') at the end of the path. For example, users may create an "Analysis" folder for each subject while running first-level analysis by setting the variable to "subjFolders" with the subfield '/Analysis/'. If the directory already exists, the directory will not be remade. Finally, users may select specific volumes of images by adding a "," and the volumes to be selected after the image name in the subfield. For example, one could enter '/Run1/Run1.nii,inf' to select all 4D volumes or '/Run1/Run1.nii,1:50' to select only the first fifty volumes. See each editor utility for additional comments.

Save Presets

After functions/variables have been chosen, users can choose to save the preset functions and variables as a pipeline for later use. Pipelines are treated similarly to the wrapper editors in that the pipeline of functions will be automatically loaded into the GUI each time the user runs it. This is extremely helpful for analyses/steps that will be performed several times. For example, a pipeline can be created for SPM fMRI preprocessing or FSL's DTI analysis. The variables chosen can then be set each time the user runs the pipeline without having to reload the functions. Additionally, the variables that are set when creating the pipeline will become the defaults each time the pipeline is used.

Load/Save/Run

Finally, users can load, save, or run a pipeline of functions by clicking "load/save/run". Users may load previously saved variables (saved "sawa/main/jobs/*_savedvars.mat") and then edit the variables in the GUI. Similarly, the variables can be saved into the "jobs" folder for later use. This is helpful for specific jobs that do not necessarily need to be pipelines. For example, running several two-sample t-tests for SPM is easiest by reloading the previous saved variables and changing the contrast image, etc. Clicking "run" will automatically begin running the functions with the chosen variables. After the functions have completed, an "output" variable will be created in the base workspace that will contain the outputs per iteration and function.

Using the Batch Editor Utility

With the batch editor utility (auto_batch.m), users can create matlab batches that pull information from the subject array and/or other functions/variables to complete jobs. Simply click "add function" and choose "auto_batch.m" to load the matlab batch editor, then load or choose functions to use. Next, within the batch editor, choose variables to be filled out with *sawa* by pressing the right arrow on the keyboard when the item is highlighted (Figure 6). Similarly, to remove a variable, press the left arrow on the keyboard with the item highlighted. After choosing variables to be filled out with *sawa*, enter any other variables directly into the batch editor and close the editor. Finally, choose a folder (either through the file system or subject array) to save the JOBS/BatchName/BatchFile.mat and choose whether to run or save the jobs. Also, if the batch uses SPM directly, the user will be asked whether to overwrite previous files (i.e. SPM.mat files). Choosing to overwrite SPM files will automatically bypass the warning dialogs displayed when such an instance occurs. After choosing subjects/iterations, click on the module to edit and then click "enter variables". This will bring up the variables to be set for the chosen module. For each item, the user will be asked how they would like to set the variable (see Figure 4 as well as page 8 for overview).

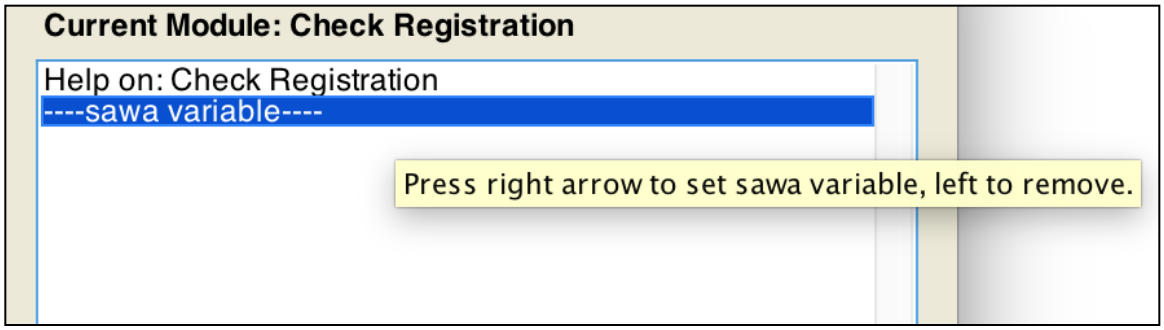
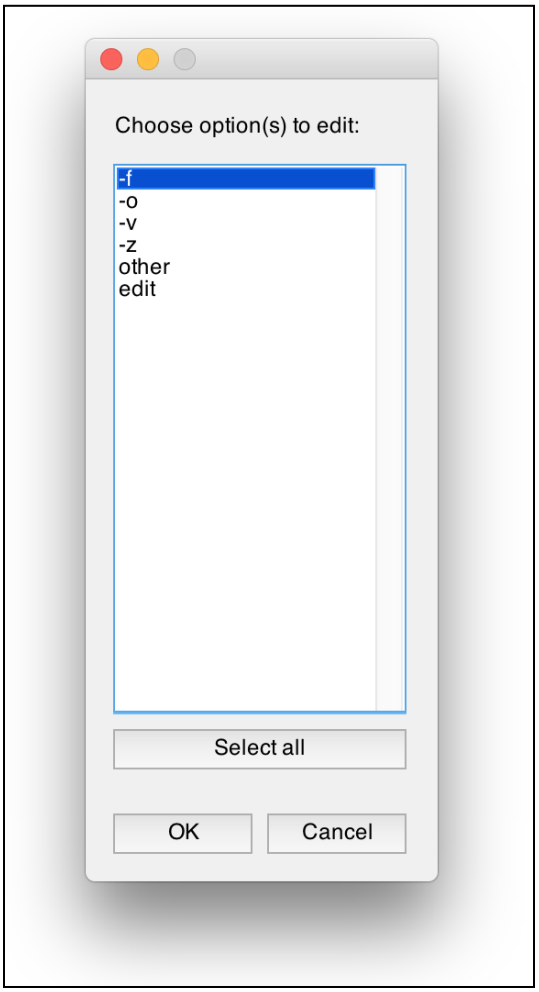


Figure 6. Selecting variables to be filled out by *sawa* in batch editor.

Using the Command Line Utility

With the command line utility (auto_cmd.m), users can easily set and wrap command line functions. Simply click “add function” and choose “auto_cmd.m” to add a command line function. Depending on the program being called, users will likely need to set the matlab path to the directory of the desired functions. Simply click “set environments”, choose “setenv”, and enter e.g. PATH and choose the folder containing the functions. (Note: if presets or jobs are saved, environments will automatically be set when using the pipeline).



Next, enter the functions to use. For each function, *sawa* will attempt to show the help information and extract command line switches (Figure 7). In addition to the command line switches, each function includes the options “other” and “edit”. The “other” option allows users to set an option that was not listed or to simply set no option (i.e. for functions that do have command line switches). The “edit” option allows users to edit the current command. This will bring up a text editor with each line corresponding to each subject/iteration. This is also how users can clear existing commands. Setting the command line switches/options is the same as setting any other variables (see Figure 4 as well as page 8 for overview). However, note that setting file variables with a wildcard will only work for single files. If multiple files would be returned from the file search, the original file path with the wildcard included is used.

Finally, users may also set a variable (e.g. for use as “$var”) by entering “var=” as the function and then enter the value as the option.

Figure 7. Command line switches are automatically extracted from help information.

Using the Matlab Function Utility

The Matlab function utility (auto_function.m) is very similar to the command line utility, with the only difference being the source of the commands. This utility allows users to wrap almost any matlab function. Note that these should be functions (i.e. with "function" at the top of the script in matlab editor). As with the command line utility, users may want to set certain script locations to the matlab path using the "set environments" button. Then, simply enter the functions to be called via the "add function" button and choosing "auto_function.m". As with the other utilities, enter the variables to use (see Figure 4 as well as page 8 for overview). For functions that have "varargin" (variable input arguments), the user will be asked if they want to add a new "varargin". Furthermore, outputs from preceding functions can be used as inputs for later functions. When setting options, preceding functions will be available in the choices of methods to set variables (Figure 8). For "varargout" outputs, users will be asked to select the index of varargout to choose.

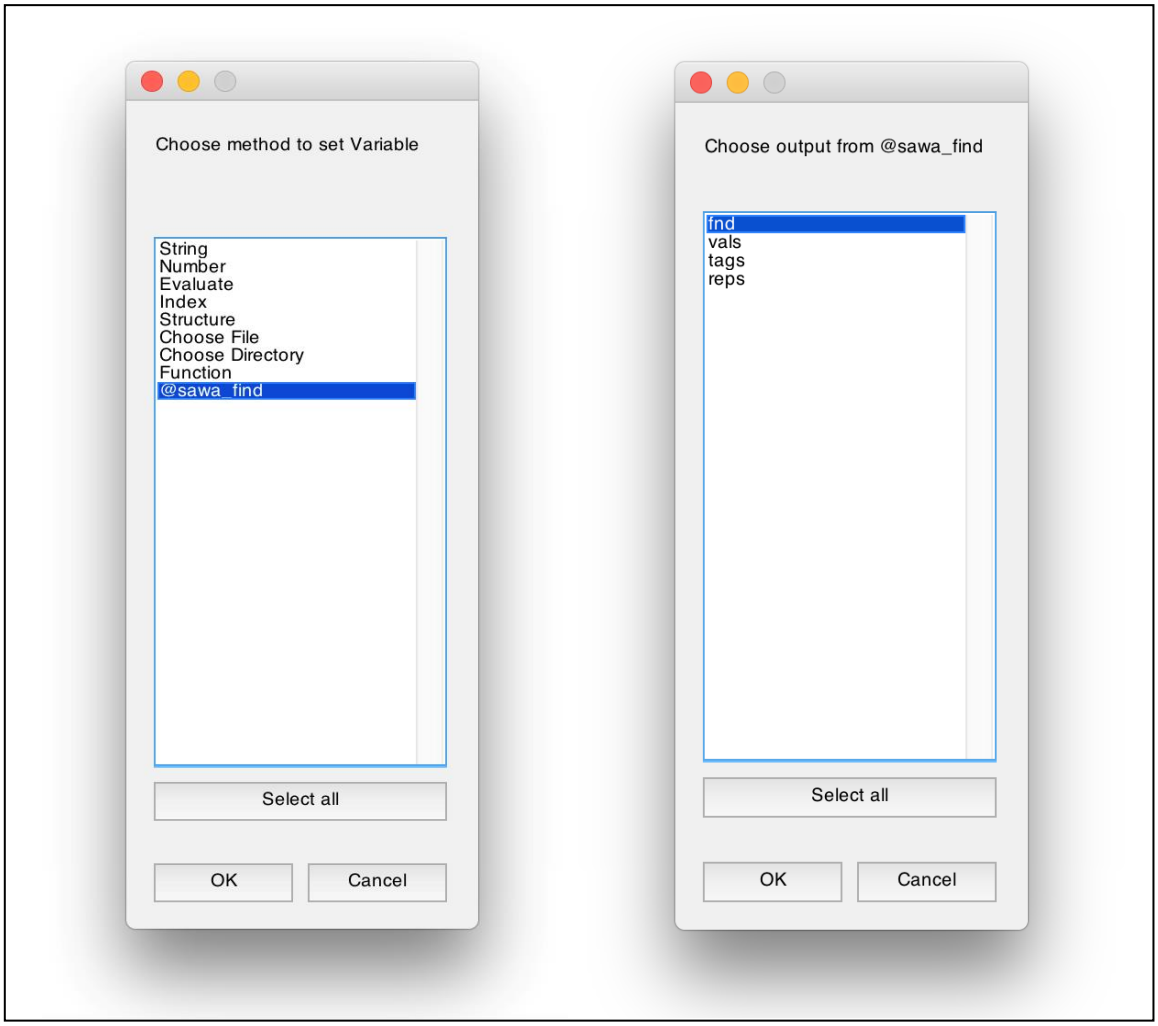


Figure 8. Using preceding function outputs as inputs.

Running *sawa* from Command Prompt

sawa.m

In addition to its GUIs, sawa can also be run via command prompt. This allows users to utilize sawa within their own scripts. First, sawa.m the main function can be run via command prompt with inputs functions to be run as well as any saved variables (both full path). For example, users could enter the following in the command prompt:

sawa('/Applications/sawa/wrapper automation/wrapper_automation.mat', 1, '/Applications/sawa/main/jobs/wrapper_automation_savedvars.mat')

Where the first input is the function to be run, the second is 0/1 as to whether savedvars will be utilized, and the final input is the savedvars file to use. If the second input is 0 instead, the user will be setting variables to save to the filename that is the third input (i.e. "_savedvars.mat" file).

sawa_editor.m

The sawa_editor function controls the wrapper automation utility. As such, it can be used from the command line to run a pipeline of functions. This function works primarily by passing a structure with the following main fields: funcs, options, program. The "funcs" field contains a cell array of the functions that will be used (e.g., "disp" or matlabbatch structure). The "options" field contains a cell array of the options that will be included (i.e. inputs for each function). This field should be a cell array with rows equal to the number of functions and columns equal to the inputs for each function. For example, if you have two functions, one with three inputs and the other with two, "options" would be a 2 x 3 cell array. Each cell of the options field should then contain a cell array of the inputs for each subject/iteration organized in rows. The "program" field contains the wrapper function to use (i.e., auto_batch, auto_cmd, or auto_function). This should be a cell array equal to the number of functions being run (e.g., {'auto_batch','auto_function','auto_function'}).

The sawa_editor function is used by the following command:

sawa_editor([command],[inputs])

This allows users to run sawa_editor's subfunctions with the aforementioned structure input. The subfunctions of sawa_editor include "load_sawafile", "set_environments", "choose_subjects", "add_function", "set_options", and "loadsaverun". You'll notice that most of these match the buttons from the wrapper automation GUI. This is exactly how you complete any of those actions. The "load_sawafile" subfunction will return the structure for a chosen "sawafile" (e.g., a pipeline that has been created). For example, users could enter the following to retrieve a structure (here named "fp") for a "Check_Registration" pipeline:

fp = sawa_editor('load_sawafile','/Applications/sawa/Check Registration/Check_Registration.mat')

In addition to the comprehensive sawa_editor function, users can run each "auto_" function in a similar fashion. Like the sawa_editor, the "auto_" functions utilize the same structure to pass information for functions, options, etc. (however, the program field would not be required when running an "auto_" function). Also, the "auto_" functions also contain the "add_function" and "set_options" subfunctions as well. However, each of the "auto_" functions also contains an "auto_run" subfunction that will wrap through the indicated functions and iterations/subjects. See explanations below for each "auto_" function.

auto_batch.m

The auto_batch function will run the wrapper for batch functions (e.g., SPM). Different from the other "auto_" functions, the contents of the funcs field for auto_batch is the actual matlabbatch structure, which is returned when saving a batch job. Furthermore, the auto_batch function uses the following fields as well: spmver, itemidx, rep, jobsavfld, jobname, saveorrun, and overwrite. Below is a short description of each:

spmver – spm version (e.g., 'spm8')
itemidx – cell array of indices of items to use as input variables (e.g., {[2,3], [4]})
rep – cell array of indices of items to repeat or 0 for no repeat (e.g., {[0],[2]})
jobsavfld – string of output folder to save jobs (e.g., 'sa(i).subjFolders{1}')
jobname – name for saving job (e.g., 'Check_Registration')
saveorrun – 'Save' or 'Run' choice
overwrite – 'Yes' or 'No' for overwriting e.g. previous SPM.mat folders

All of these fields are automatically set when running the "add_function" and "set_options" subfunctions (e.g., fp = auto_batch('add_function')).

auto_cmd.m

The auto_cmd function will run the wrapper for the command line functions (e.g., AFNI or FSL functions). The main difference for this function is the options field. For this function, the options field will still be a cell array with rows equal to the number of functions, however there will only be one column. This is because the options will be set as a single line (e.g., '-f /Applications/foo.nii –o /Applications/newfoo.nii'). Each row, however, should still correspond to a different subject/iteration (e.g., {1,1}{1}, {1,1}{2}, etc.).

auto_function.m

The auto_function function is the wrapper for any matlab function and is the simplest of the three. The only difference for auto_function is that users may select outputs from preceding functions as inputs using the following as an option:

'evalin('caller','output{i}{2,3}');'

This would evaluate the input for the chosen function to the output from the same subject/iteration for the 2nd function and 3rd output (note, this option would need to be an input for a function after the 2nd function).

Appendix A. Functions

The built-in functions can be found in the /sawa/main/functions folder (with the exception of subjectarray.m which is located in /sawa/subject array). In addition to the brief descriptions below, more help can be found by typing help([function name]) in the command prompt.

```
any2str.m
  out = any2str(varargin)
  This function will convert any class to its string representation (via
  @disp).
auto_batch.m
  varargout = auto_batch(cmd,varargin)
  This function will allow you to set arguments for the command line
  function funname and then run the function.
auto_cmd.m
  varargout = auto_cmd(cmd,varargin)
  This function will allow you to set arguments for the command line
  function funname and then run the function.
auto_function.m
  varargout = auto_function(cmd,varargin)
  This function will automatically create a wrapper to be used with
  chosen function and subjects.
cell2strtable.m
  strtable = cell2strtable(celltable,delim)
  Create string table from a cell table (different from matlab's table)
  with a specified delimiter separating columns.
choose_SubjectArray.m
  varargout = SubjectArray(fileName,task)
  holder for subjects.mat location
choose_fields.m
  flds = choose_fields(sa, subrun, msg)
  Choose string represntations of fields from subject array
choose_spm.m
  choose_spm
  This function will allow choosing between multiple SPM versions
closedlg.m
  closedlgs(figprops,btnprops)
  This function will set a timer object to wait until a certain
  dialog/message/object is found using findobj and will then perform a
  callback function based on button chosen.
collapse_array.m
  [C, idx] = collapse_array(A)
  This functon collapses consecutive identical components of an array.
common_str.m
  str = commmon_str(strs)
  This function will find the greatest common string (str) among a cell
  array of strings (strs)
convert_paths.m
  A = convert_paths(A,newpaths,type)
  This function will convert the paths in the array A from Mac to
  PC or vice versa.
funpass.m
  funpass(structure,vars)
  This function allows you to pass variable structures between functions
  easily. If only one input argument is entered, structure will be
  returned as well as variables created from the fields in structure.
  If two input arguments are entered, the structure is updated using
  the evaluated variables in vars.
getargs.m
  [outparams,inparams] = getargs(func)
  This function will retreive the out parameters and in parameters for a
  function.
```

```
make_gui.m
  data = make_gui(structure)
  This function will create a gui based on a "structure" of gui
  properties structure should be a cell array of structures
  corresponding to number of "pages" for the gui
match_string.m
  [gcstr,idx] = match_string(str)
  This function will find the greatest common string derivative of str
  that matches the greatest number of strings in cellstr
printres.m
  [hres,fres,outtxt] = printres(varargin)
  Create Results Figure
savesubjfile.m
  sa = savesubjfile(fileName, task, sa)
  Saves subjects.mat file and copies previous to backup folder
sawa.m
  subject array and wrapper automation (sawa)
  This toolbox will allow you to build pipelines for analysis by
  wrapping any command line, matlab, or batch functions with input
  variables from subject arrays or otherwise. The main component of
  this toolbox is the Subject Array (sometimes seen as "sa"), which
  contains the subject information (e.g., folders, demographic
  information, etc.) that can be loaded into different
  functions to automate data analysis. The toolbox comes with several
  functions highlighted by an editable batch editor, command line
  editor, function wrapper. Furthermore, users may add
  scripts/functions to the main folder to be used or build function
  pipelines using by saving presets in the aforementioned editors.
sawa_cat.m
  out = sawa_cat(dim,A1,A2,...)
  This function will force the directional concatenation of the set of
  inputs A1, A2, etc. by padding inconsistencies with cells.
sawa_createvars.m
  vars = sawa_createvars(varnam,msg,subrun,sa,varargin)
  Creates variables for specific use in auto_batch, auto_cmd,
  auto_function.
sawa_dlmread.m
  raw = sawa_dlmread(file,delim)
  This function will read csv/txt files and create a cell array based on
  delimiters.
sawa_editor.m
  sawa_editor(sawafile, sv, savedvars)
  Loads/runs sawafile functions (e.g., auto_batch, auto_cmd,
  auto_function) with make_gui.
sawa_eq.m
  C = sawa_eq(A,B)
  Checks if all components of A and B are equal.
sawa_evalvars.m
  valf = sawa_evalvars(val,opt)
  Evaluate variables created from sawa_createvars
sawa_fileparts.m
  outputs = sawa_fileparts(inptus, part, str2rep, repstr)
  function to get fileparts of multiple cells, remove parts of all
  strings
sawa_getfield.m
  [C,S,reps] = sawa_getfield(A,'property1','value1',...)
  Return values, substructs, and/or string representations from an
  object A.
sawa_insert.m
  output = sawa_insert(A, idx, B)
  This function allows users to insert components of arrays without
  error.
sawa_savebatchjob.m
  savepath = sawa_savebatchjob(savedir, jobname, matlabbatch)
```

```
  Save Batch Job
sawa_screencapture.m
  filename = sawa_screencapture(hfig,filename,ext)
  This function simply screencaptures a figure using the hgexport
  function.
sawa_searchdir.m
  files = sawa_searchdir(fld, search)
  search for files or folders within fld
sawa_searchfile.m
  [files, pos] = sawa_searchfile(expr,fld,filetype)
  search for expr within each .m file (default) in fld
sawa_setbatch.m
  [matlabbatch,sts] = sawa_setbatch(matlabbatch,val,idx,rep,m)
  iteratively set matlabbatch items for a single module (including
  repeat items)
sawa_setcoords.m
  k = sawa_setspmcoords(hfig,coords)
  This function will change the coordinates for the spm based axes in
  figure hfig using coordinates coords.
sawa_setdeps.m
  matlabbatch = sawa_setdeps2(matlabbatch, cjob)
  This function will set dependencies for matlabbatch jobs based on
  outputs from previous jobs.
sawa_setfield.m
  [C,S,reps] = sawa_setfield(A,'property1','value1',...)
  Set field/index for object A.
sawa_setupjob.m
  [matlabbatch, itemidx, str] = sawa_setupjob(matlabbatch, itemidx, m)
  Opens matlabbatch job using batch editor (cfg_ui) and records items to
  be used as sawa variables as well as the input user data.
sawa_setvars.m
  savedvars = setvars(mfil)
  Set variables for running multiple functions in GUI for either
  scripts or .mat functions
sawa_strjoin.m
  str = sawa_strjoin(C, delim)
  This function will concatenate any input as string with delimiter.
sawa_subrun.m
  [subrun,sa,task,fileName] = sawa_subrun(sa,subrun,isubrun)
  Choose fileName, task, subjects, and refine subjects based on subject
  fields
sawa_system.m
  [sts,msg] = sawa_system(funcs,opts,n)
  This function will run "system" but uses the wine function if running
  a .exe on mac.
sawa_xjview.m
  sawa_xjview('property1','value1','property2','value2',...)
  This function will allow for saving images and cluster details from
  multiple files using xjview
sawa_xlsread.m
  raw = sawa_xlsread(xfil)
  This function is mainly used since xlsread does not work consistently
  with .xlsx files on mac. However, this will not slow down the ability
  the functionality on pc or with .xls files. It also simplifies the
  usual [~,~,raw] = xlsread(xfil,s).
settimeleft.m
  hobj = settimeleft(varargin)
  Sets time left display
subidx.m
  out = subidx(item,idx)
  Index item as item(idx)
update_array.m
  sa = update_array(task)
  Used to update array (sa) with the latest data. Primarily used when
```

```
  savedvars is chosen, to ensure that the array is not going to save
  older data.
update_path.m
  update_path(fil,mfil)
  This function will update a filepath (fil) for the .m file entered
  (i.e. mfilename('fullpath'))
```

Appendix B. Examples
The following examples will demonstrate the basics of how sawa works. No datasets are needed to complete this basic tutorial.

1. Creating a test subject array
First open matlab and type "sawa" into the command prompt. Choose "subject array" and add "subjectarray" to your queue of functions to run. Press "run" to open the subjectarray GUI.

Enter "testarray" in the "subject array name" text edit. Press "subjects.mat file". Cancel to choose a folder in which to save the new subject array (e.g., /sawa/main/subjects). If you roll your mouse over "subjects.mat file", you should see your selected directory. Press "enter subjects" and "Yes" to the dialog "Add subjects?". Enter 3 subjects, then choose "String" as the method to set "subj". Enter "test1", "test2", "test3" each on a separate line, then click "ok". The "subj" field should now appear in the GUI to the right under "subject array fields".

Press "subject folders" and "Yes" to the dialog "Choose directory and use subject names as subject folders?". Choose a directory for the subject folders to exist (e.g., /Users/Desktop). Select "Yes" to the dialog "Make subject folders?", which will automatically create your subject folders in the selected directory. Now "subjFolders{1}" should appear below "subj" in the "subject array fields" list.

Press "create new field" and enter "group" into the "Edit field name" text box. Select subjects "test1" and "test3". Click "Cancel" to the dialog "Choose field to refine subject list by". This is utility is useful when more subject fields have been established. For example, users can select only subjects in a particular group or of a certain age. Click "Yes" to the "Continue?" dialog. Select "String" as the method to set "group". Enter "Control" into the text box and click "Ok". The "group" field will now appear in the "subject array fields" list. Click on it in the list and choose "Edit" in the dialog window. You'll notice that the command prompt will display the subjects and their respective groups as such:

test1: Control
test2:
test3: Control

Click "Ok" to maintain the same field name. This time select subject "test2", continue with the selected subject, and choose "String" to set "group". Enter "Patient" in the text box. Now if you click "group" in the list again (this time close the dialog window rather than selecting "Edit", "Copy", or "Delete") you will notice that the command prompt will show the updated groups (as below).

test1: Control
test2: Patient
test3: Control

Next click “load/save” then “save” in the following dialog window. Select “Yes” to “Save subject array testarray?” and choose “Excel” and “Text” in the following “Choose files(s) to save subject array to” window. For each, select subjects to output, choose a directory to save the file into, and enter a name to save (or keep the default, “testarray”).

A new window will appear with the information for the testarray and your selections. This is the “print results” window and appears when running subjectarray, wrapper_automation, and other built-in functions. The results should display the following:

subjectarray:
Subject Array name: testarray

Location: /Applications/sawa/main/subjects/subjects.mat

Subject Array testarray saved: Yes

Subject Array files saved: Excel
Text

Subject Array fields: subj
subjFolders
group

Subjects: test1
test2
test3

Press “save” at the bottom of the window, leave “subjectarray” as the filename, and select the newly created “testarray” in the following window. (You may have to select the subject array from the main GUI page via “subject array” button first). If you were to cancel when choosing the subject array, you will be able to save the file into a chosen directory. You can also click inside the results window and type any notes if needed.

If you go to the folder containing the “subjects.mat” file (e.g., /sawa/main/subjects), you should see a new folder, “Notes”. Within this folder will be a folder, “testarray”, with a subfolder containing the “subjectarray.txt” file. You’ll notice that this subfolder has today’s date—this is how the results files are organized. (If you saved the file to a specific directory instead of to the subject array, only the folder with today’s date will be created). Therefore, your results/notes will be saved by date and analysis type.

Next press “done” in the subjectarray GUI and “quit” in the “functions to run” GUI. This will return you to your main GUI page. Press “main” to return to the main set of sawa folders. This time select “wrapper automation” and add “wrapper_automation” to your functions to run.

2. Using wrapper automation

If you haven't already, make sure to click the "subject array" button in the main GUI page to select the "subjects.mat" file for "testarray". With "wrapper_automation" now in your "functions to run" GUI, click "run". First, click "choose subjects", select "testarray", and choose all subjects. Next, choose "group" from the "Choose field…", enter "strcmp" into the "Function" window, and enter "Control" into the "Search" window. Choose "Refine", then when the "Choose field…" window returns, select "subj". Again enter "strcmp" in the "Function" window, but then enter "test2" in the "Search" window. This time select "Add" and then cancel when returned to the "Choose field…" window. Select "Yes" to continue with your three subjects now in the following order: test1, test3, test2. Next choose "per subject" from the "Per Subject/Iterations" window. This allows you to choose whether you want the functions that you'll add to the list to be run "per subject" or for a number of iterations.

Now click "add function", choose "auto_cmd", and enter "val=" in the "Function" window. Now right-click in the list that displays "val=" and select "delete". Now follow the same instructions, but enter "VAL=" instead. Note: if you're using a PC, enter "set" as the auto_cmd function. Next, click "set options", choose "other", and leave the "option" blank. Select "Subject Array" from the "Choose method…" window, select "Individual" from the "Variable" window, and choose the "subjFolders" field and subcell "1". Leave the "Enter subfolder, files" window blank. The "Choose method…" window will then reappear.

This "Choose method…" window is the starting point for setting the input variables/options for any function. As you can see, users can choose the type of variable they want to input (e.g., string, number, structure, etc.). The "Subject Array" option allows you to input subject variables (either multiple variables by choosing "Group" or each subject's variable by choosing "Individual"). There are three ways to set the variables for each iteration of your current function: 1) enter multiple inputs (e.g., one per line in "String" or multiple files/directories), 2) choose a new method for each iteration, 3) select "Index" and manually enter the index for each iteration then select the method, or 4) select "Function" and create your variables using a matlab function. The "Index" option is good for when you want to enter multiple files for a single iteration. To do this, you would simply click "Index", enter "{1}" into the "____" window, and then select your method for iteration 1. The "Function" method is good for creating a large number of variables (e.g., num2cell(randi(1,4,1)) to create 4 random numbers). Keep in mind that each iteration is its own "cell", so when using the "Index" or "Function" methods you need to create cells for each iteration. Now click cancel in the "Choose method…" window.

In the command prompt, you should now see the following:

```
VAL=   sa(i).subjFolders{1}
```

For auto_cmd functions, you will be able to see the function and options you have set each time they are set. Don't worry about the space between "VAL=" and

“sa(i).subjFolders”; this will not affect its functionality. For other auto_cmd functions, you will see “help” messages, which will allow you to choose other command switches (in the window where you selected “other”).

Now click “add function” again, select “auto_batch”, and if you have multiple SPM versions, select spm8. The “Batch Editor” that you might use in SPM will appear. Under “BasicIO”, select “Make Directory” and then “Pass Output to Workspace”. To add variables that you want to set using sawa, select the variable and press the right arrow key. This will replace the item with “----sawa variable----“. Similarly, to remove a sawa variable, select the variable and press the left arrow key. In the “Make Directory” module, add the “Parent Directory” item using this method. Next, set “New Directory Name” to “new folder” in the Batch Editor. In the “Pass Output to Workspace” module, add “Output Variable Name” as a sawa variable and set “Output Item” to the Make Directory output using the “Dependency” button. Now close the Batch Editor. If you want to save the batch job for each iteration, you can select a directory to save into or cancel to select a field from the “testarray” to save the job. Cancel the choose directory window and select “subjFolders” field and subcell “1”. Change the job name from “wrapper_automation” to “Make Directory”. Select “Run” from the “Save or Run” window and the “Make Directory” dependency from the “Choose dependencies…” window.

You should now see each function in the GUI list. These functions can be edited, copied, deleted, or a new function can be inserted before a selected function. To do this, select a function in the list then right-click to choose the action you want to perform.

Next, click “set options” with “Make Directory” selected and choose “Parent Directory” as item to set. In the “Choose method…” window, you should now see “@VAL=” below “Subject Array”. For any function that can output a variable, you will be able to set that output as the input variable for a later function of any modality. Select “@VAL=” and select “VAL” from the “Choose output…” window and cancel the “Choose method…” window. Set the options for “Pass Output to Workspace” and choose “Output Variable Name” as item to set. You’ll see that “@Make Directory” is now also an option, but choose “String” as the method instead. Enter “var1”, “var2”, and “var3” each on a separate line.

Again, click “add function”, select “auto_function”, and enter “strrep”. This is a matlab function that replaces strings (characters; e.g. “this”) with other strings. Next, select “varargout” from the “output variables” window and leave “1” as the number of outputs to return. For functions that have variable outputs (i.e. returns a variable number of outputs), you will enter the expected number of outputs. This is also the case for most auto_cmd functions as well. Set options for “strrep” and select “varargin” as the input variable. Like “varargout”, “varargin” is used for variable inputs. When inputting variables for “varargin”, you will be asked if you want to add a new “varargin” after you’ve cancelled the “Choose method…” window. If you look at the command prompt again, you’ll see the help message for “strrep”. Any time you set options for a matlab function, the function’s help message will appear.

Now select “@Make Directory” and choose “Make Directory ‘new folder’” as the output. Cancel when the “Choose method…” window reappears, and click “Yes” in the “New varargin?” prompt. Select “Choose Directory” as the method for “varargin 2” and choose the directory where the subject folders are for “testarray”. Cancel again when “Choose method…” appears and again select “Yes” to the “New varargin?” prompt. Again select “Choose Directory” and choose any directory you want then choose “No” to the “New varargin?” prompt.

Now we are ready to run the functions. Click “load/save/run” then select “run”. You should see something similar to the following printout:

```
wrapper_automation:
test1
VAL= "/Users/theissjd/Desktop/test1"
VAL
"/Users/theissjd/Desktop/test1"

SPM version: spm8
Make Directory: [1x1 cfg_files]    [1x1 cfg_entry]
New Directory Name: 'new folder'
Pass Output to Workspace: [1x1 cfg_entry]    [1x1 cfg_entry]
Output Item: [1x1 cfg_dep]
Parent Directory: /Users/theissjd/Desktop/test1
Output Variable Name: var1
Make Directory 'new folder'
/Users/theissjd/Desktop/test1/new folder
Saving job to /Users/theissjd/Desktop/test1/jobs/Make Directory/Make
Directory_17-May-2016.mat

strrep
varargin
/Users/theissjd/Desktop/test1/new folder /Users/theissjd/Desktop/
/Users/theissjd/sawa/
varargout
/Users/theissjd/sawa/test1/new folder
--------------------------------------------------------------------------
test3
VAL= "/Users/theissjd/Desktop/test3"
VAL
"/Users/theissjd/Desktop/test3"

Parent Directory: /Users/theissjd/Desktop/test3
Output Variable Name: var3
Make Directory 'new folder'
/Users/theissjd/Desktop/test3/new folder
```

Saving job to /Users/theissjd/Desktop/test3/jobs/Make Directory/Make Directory_17-May-2016.mat

strrep
varargin
/Users/theissjd/Desktop/test3/new folder /Users/theissjd/Desktop/
/Users/theissjd/sawa/
varargout
/Users/theissjd/sawa/test3/new folder
---
test2
VAL= "/Users/theissjd/Desktop/test2"
VAL
"/Users/theissjd/Desktop/test2"

Parent Directory: /Users/theissjd/Desktop/test2
Output Variable Name: var2
Make Directory 'new folder'
/Users/theissjd/Desktop/test2/new folder
Saving job to /Users/theissjd/Desktop/test2/jobs/Make Directory/Make Directory_17-May-2016.mat

strrep
varargin
/Users/theissjd/Desktop/test2/new folder /Users/theissjd/Desktop/
/Users/theissjd/sawa/
varargout
/Users/theissjd/sawa/test2/new folder

Notes: