Subject Array and Wrapper Automation (SAWA) - version 1 -

SAWA is a toolbox that automatically wraps 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 component of SAWA is the subject array, a structural array that can contain subject-specific information to be utilized in wrapping functions. At its simplest, SAWA is an organizational tool that can maintain up-to-date information as well as record analyses and other notes. However, SAWA is built to feed information from the subject array to a wrapper for any batch, function, or command. As such, SAWA provides users the ability to perform complex analyses using subject data in 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 matlab's batch utility system, which allows users to directly choose variables that will be filled by the subject array or other functions/variables. The command line editor allows users to wrap command line functions while also selecting command line switches to use and set. Finally, the function wrapper utility provides users the ability to wrap matlab functions by setting input arguments and selecting output arguments to be returned.

As a way to record analyses that users have run, SAWA also prints inputs/outputs/errors. Users can add notes into the analysis records and save records for different subject arrays (e.g. different studies). Records are stored by analysis name and date.

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).

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).

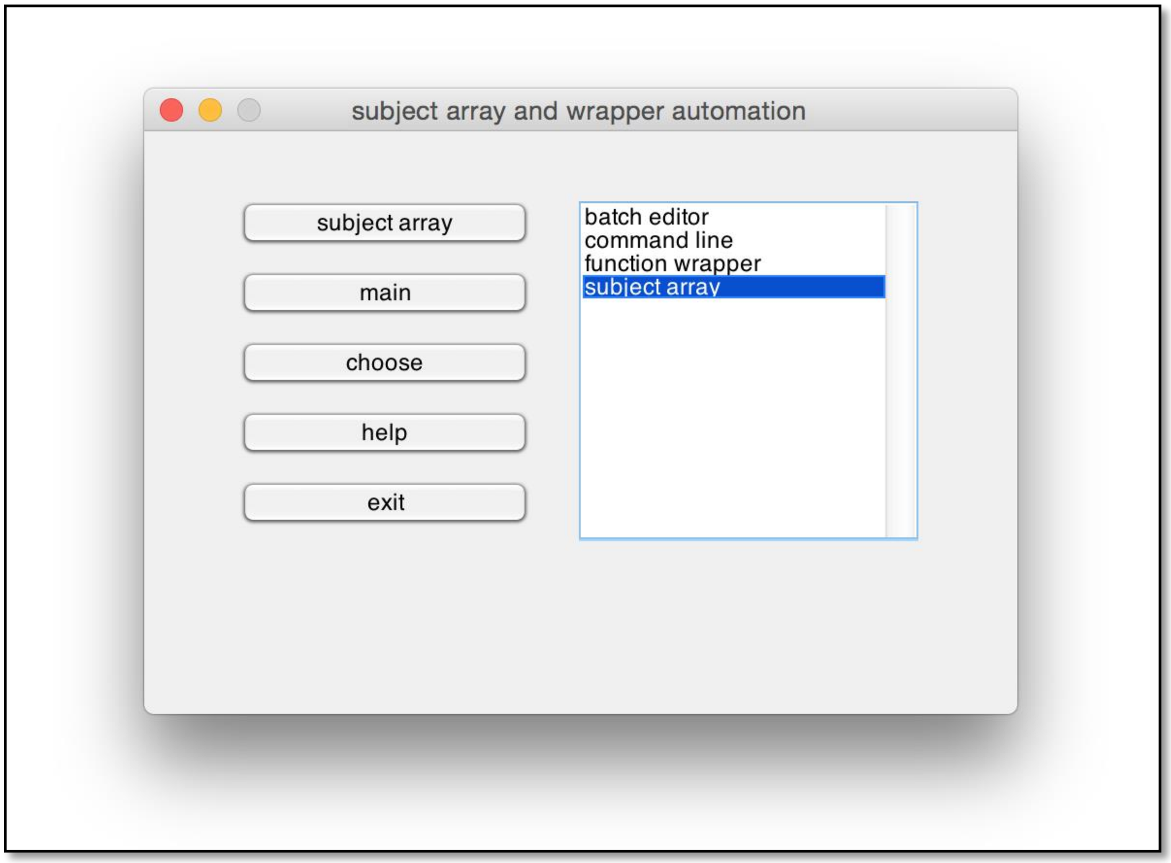


Figure 1. Main GUI page for SAWA includes folders in “sawa” folder.

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.

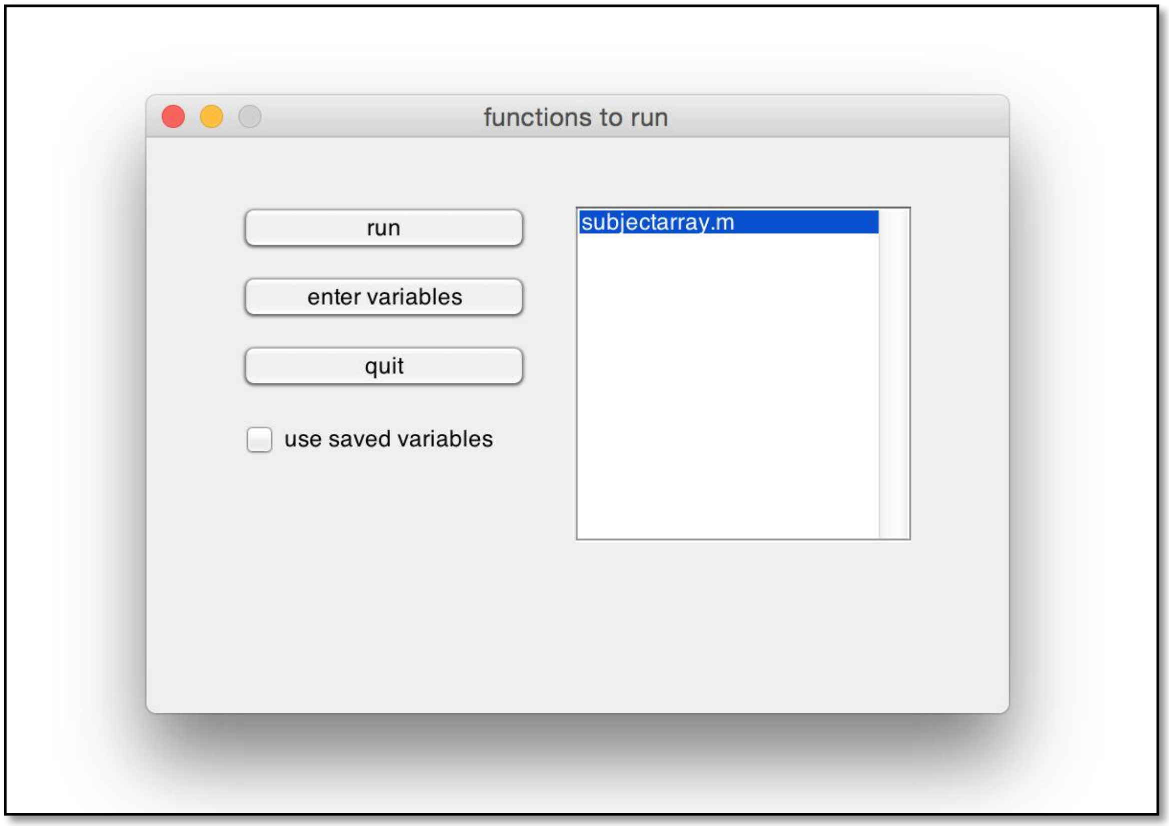


Figure 2. Functions to run GUI page for SAWA includes any files within the previously selected folder.

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.

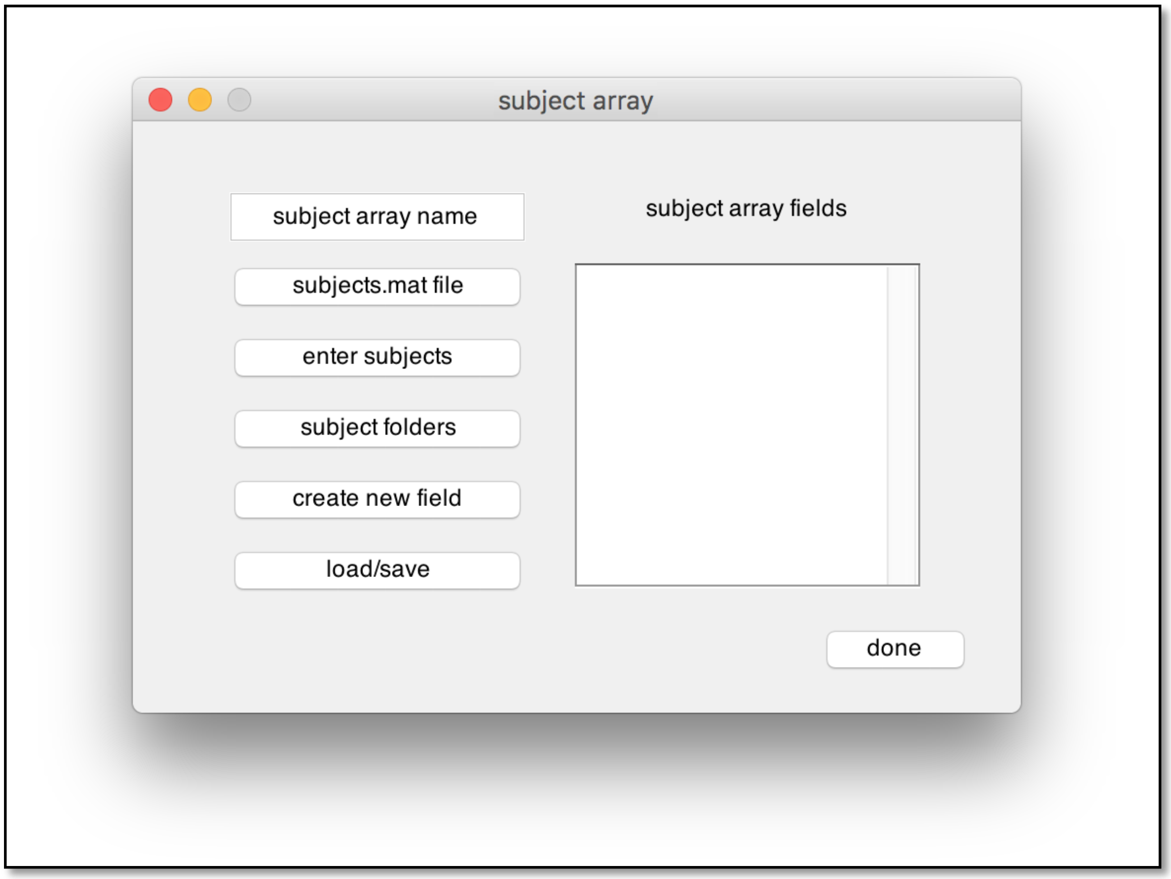
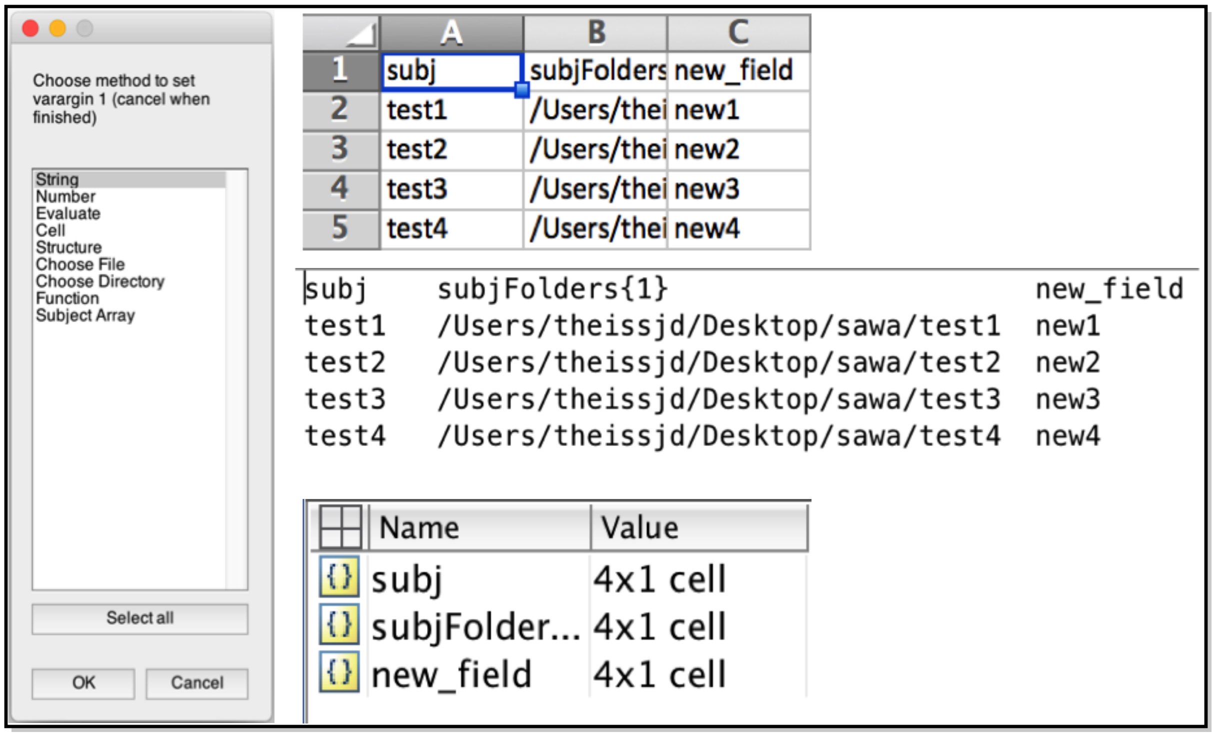


Figure 3. The subject array GUI allows users to create/load the subject array.

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.



Choose method to set varargin 1 (cancel when finished)

	A	B	C
1	subj	subjFolders	new_field
2	test1	/Users/thei	new1
3	test2	/Users/thei	new2
4	test3	/Users/thei	new3
5	test4	/Users/thei	new4

```
subj    subjFolders{1}                      new_field
test1   /Users/theissjd/Desktop/sawa/test1  new1
test2   /Users/theissjd/Desktop/sawa/test2  new2
test3   /Users/theissjd/Desktop/sawa/test3  new3
test4   /Users/theissjd/Desktop/sawa/test4  new4
```

Name	Value
subj	4x1 cell
subjFolder...	4x1 cell
new_field	4x1 cell

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. The first three methods (String, Number, and Evaluate) all involve an input box. Each row of the input box corresponds to a different subject. Furthermore, the 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”)). The “Cell” method will allow users to create cell variables. The “Structure” method allows users to create a structure array within the field (e.g., “Responses.Values”). When this method is selected, the user enters subfield(s) to create within the initial field. Then with each subfield the user will choose a method to set the value within the subfield. The “Choose File” and “Choose Directory” methods will open a file/directory selection window (the same as SPM uses). 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”. 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.

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.

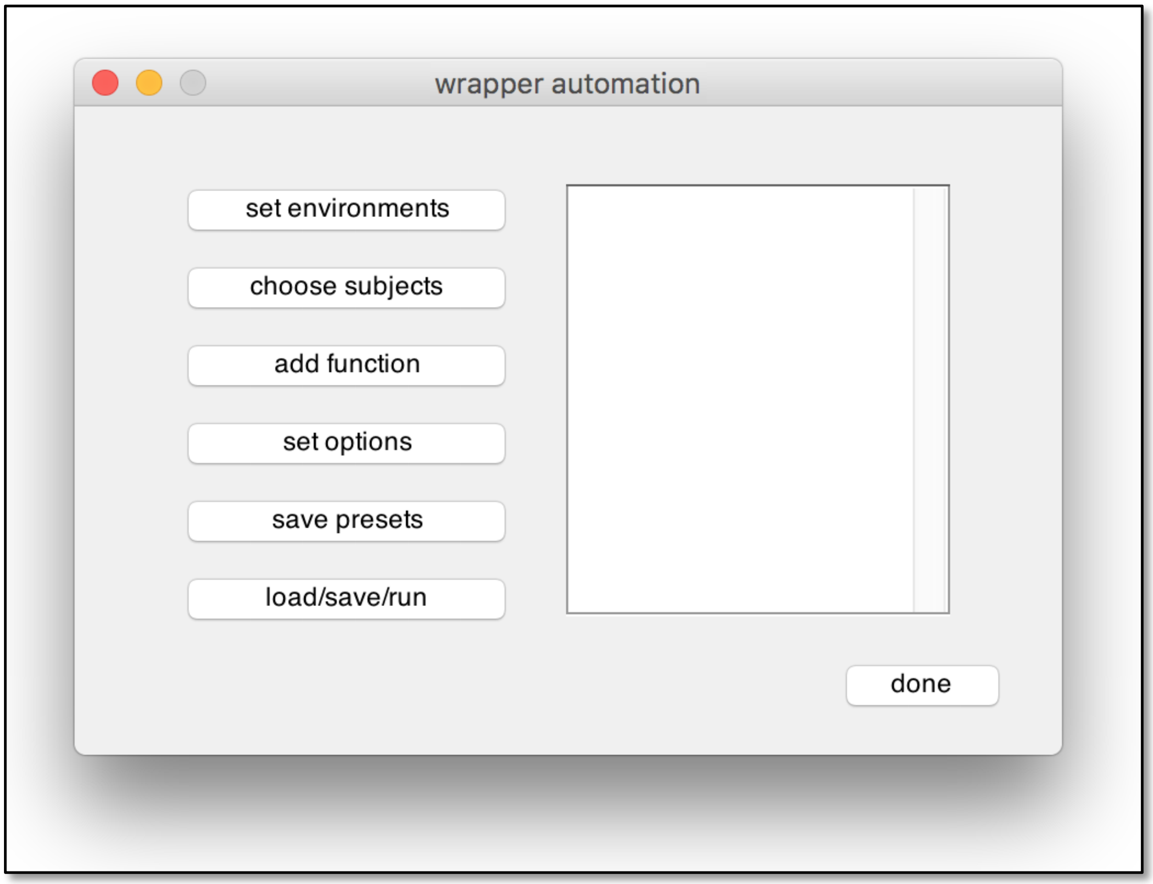


Figure 5. GUI for wrapper automation.

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, functions will automatically be assumed to be "per iterations" and the number of iterations will be matched to the number of variables input (e.g., if 12 files input, run 12 times). Similarly, if the number of iterations does not match the number of variables input, it will be assumed that those inputs should be included in each iteration (e.g., if 12 files input but two iterations chosen, run with 12 files for both iterations). 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, users can create matlab batches that pull information from the subject array and/or other functions/variables to complete jobs. Simply click "setup batch job" 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 pages 6 and 9 for overview).

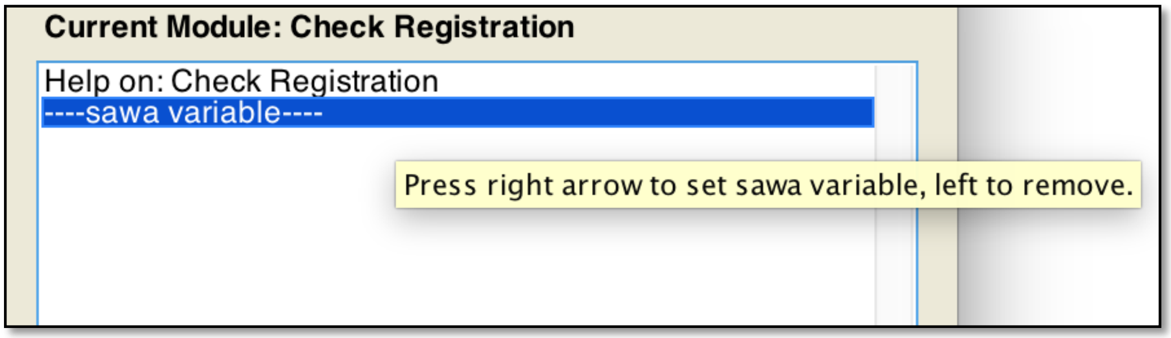
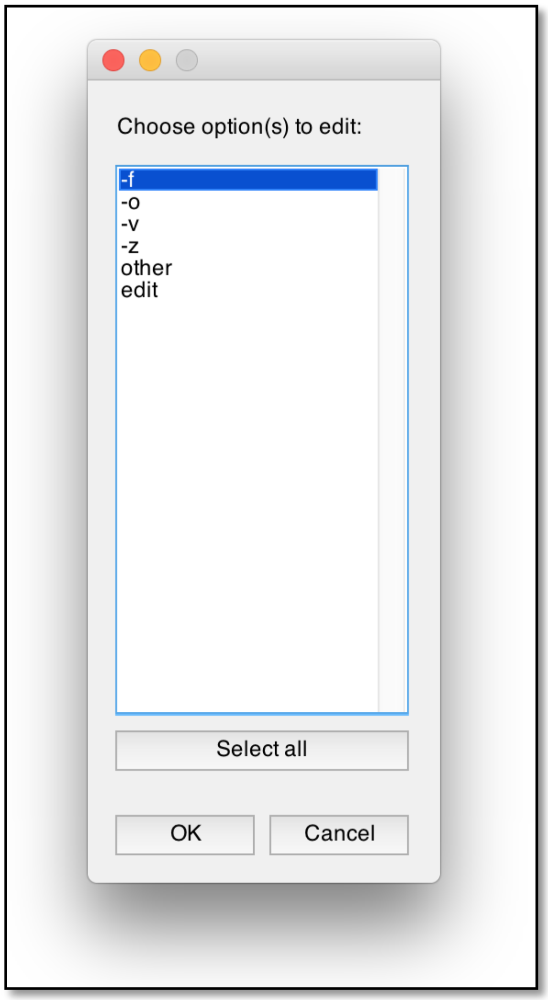

Current Module: Check Registration
Help on: Check Registration
----sawa variable----
Press right arrow to set sawa variable, left to remove.

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

Using the Command Line Utility

With the command line utility, users can easily set and wrap command line functions. 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 pages 6 and 9 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 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, set the input arguments, and select output arguments. As with the other utilities, enter the variables to use (see Figure 4 as well as pages 6 and 9 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). Finally, the output arguments that are selected will be displayed (if applicable) in the printed output, but also a variable called "output" will be created in the main workspace once the functions have completed.

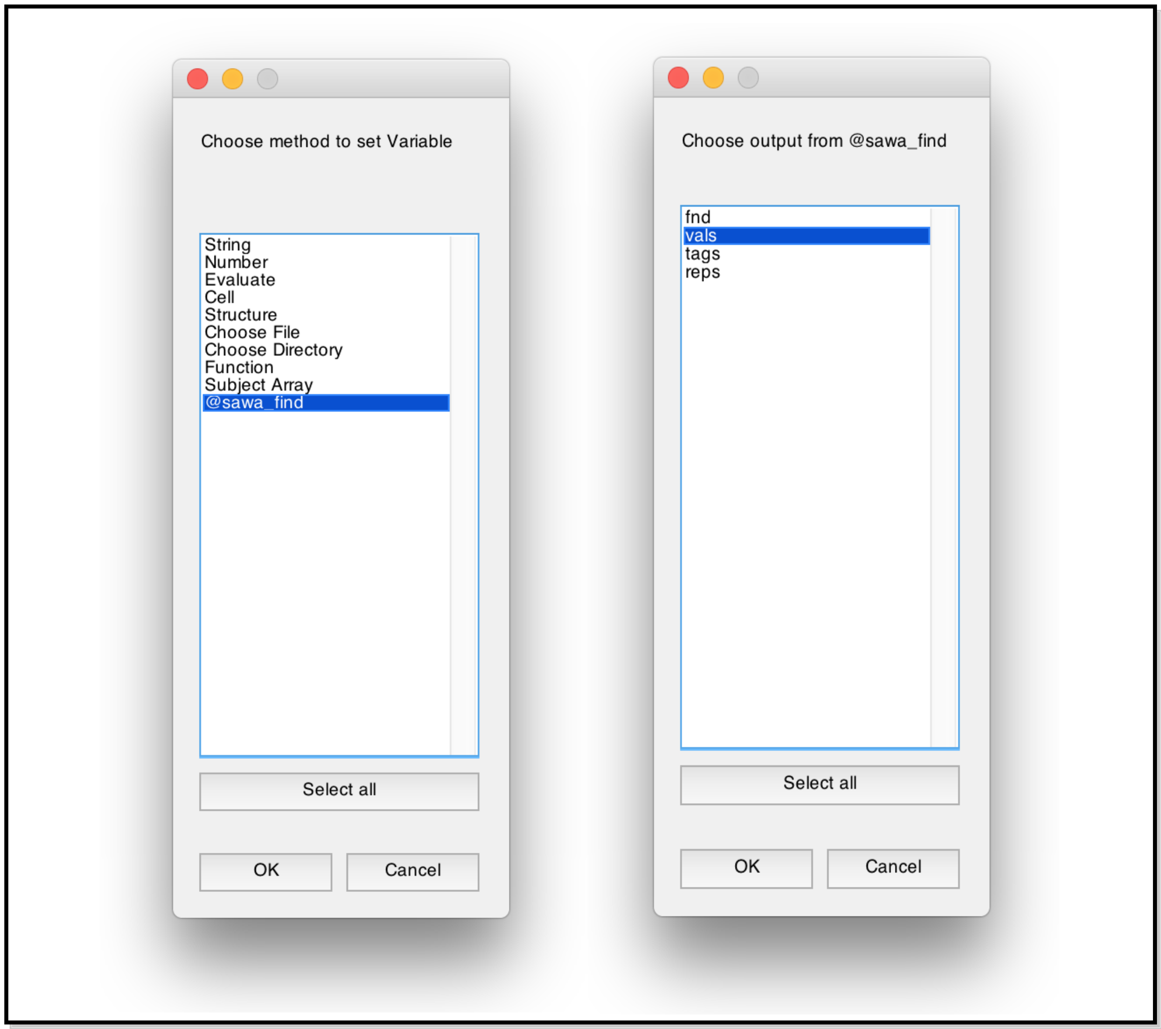


Figure 8. Using preceding function outputs as inputs.

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.

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

climb_branches.m

[val,tag,rep] = climb_branches(root,rootrep,lvl,idx)
This function will search through a structure as follows
root.mainbranch.lvl1(idx1).lvl2(idx2)

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)

common_str.m

str = commmon_str(strs)

convert_paths.m

sa = convert_paths(sa,task,fileName)
This function will convert the paths in the subject array sa 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

sawa(funcs, sv, savedvars)
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_evalchar.m

out = sawa_evalchar(str,expr)
evaluate strings using subject array (or evaluate expr within str)

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_find.m

[fnd,vals,tags,reps]=sawa_find(fun,search,varargin)
searches array or obj for search using function fun

sawa_getfield.m

[values, tags, reps] = sawa_getfield(A, irep, itag);
Gets values, tags, and reps (string representations) of structures or objects

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,fullfiles] = sawa_searchdir(fld, search)
search for files or folders within fld

sawa_searchfile.m

[files, pos] = sawa_searchfile(str,folder,filetype)
search for str within each .m file (default) in folder

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_setdeps(prebatch, matlabbatch)
this function will set dependencies for fields that change (e.g. if a dependency is set for Session 1 and new Sessions are added, new dependencies will be added to mirror the Sessions added). If however, the number available dependencies prior function-filling is greater than the number of dependencies set by user, then the user-set dependencies will be used instead.

sawa_setfield.m

structure = sawa_setfield(structure,idx,field,sub,varargin)
Set fields for structure indices

sawa_setupjob.m

[matlabbatch, itemidx, str] = sawa_setupjob(matlabbatch, itemidx)
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(fun,opts)
This function will run "system" but uses the wine function if running a .exe on mac.

sawa_xjview.m

sawa_xjview(varargin)
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)

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). 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 “No” to the dialog “Make subject folders?”. 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.

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, 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 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). 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. 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
Run "wrapper_automation" to bring up its GUI. Begin by clicking "choose subjects" then select the "testarray" and all three subjects. Choose "per subject" in the following dialog window.

Click "add function" and select "auto_function.m" in the "Choose program to use" window. Enter "isdir" in the following text box. You should then see "isdir" appear in the list on the right side of the GUI.

Press “set options” select “dirpath” in the following “Choose inputs to set” window. Next, select “result” in the “Choose output variables” window. You’ll notice that the help message for “isdir” appears in the command prompt window. Select “Subject Array” in the “Choose method to set dirpath” window and “Individual” in the “Choose group or individual” dialog window. This choice means that each iteration will evaluate the selected field using the current subject. Choosing “Group” allows users to set a variable to selected subjects. For example, in an SPM T-Test, users could select “Group” then “subjFolders” and enter ‘/Analysis/con*1.img’ in the subfield to select the first contrast image for selected subjects.

In the “Choose field(s) for dirpath” window, select “subjFolders” and “1” in the following “Choose subcells…” window. Enter “/Analysis” (or “\Analysis” for pc users) in the next text box. Make sure not to include a file separator at the end (e.g., “/” or “\”), as this would automatically create an “Analysis” folder, which we do not want to do.

At this point, the “Choose method to set dirpath” window should reappear. Click “Cancel” to set the dirpath variable. If we were selecting different folders for each of our three subjects, we could continue by reselecting “Subject Array” and enter a new folder to use. When doing so you would need to ensure that the number of input variables matches the number of subjects you intend to set (e.g., choosing “Subject Array” three times in this instance).

Next click “load/save/run” and select “save”. A new text box will appear with “wrapper_automation_savedvars.mat” as a default name to save the job as. Jobs are automatically saved in the /sawa/main/jobs folder for later use. Change the name to “isdir_savedvars.mat” and click “Ok”. Next click “done” in the wrapper automation GUI.

Now in the “functions to run” window, check the “use saved variables” box and select the “isdir_savedvars.mat” file we just saved. Click “run” and the following results text should appear in a new window:

```
wrapper_automation:
test1
isdir
dirpath
/Users/[username]/Desktop/test1/Analysis
result
0

test2
isdir
dirpath
/Users/[username]/Desktop/test2/Analysis
result
0
```

```
test3
isdir
dirpath
/Users/[username]/Desktop/test3/Analysis
result
0
```

Notes:

This displays the inputs and outputs for each iteration. Now go to your "workspace" in matlab and double-click on the "output" variable that was created. You should find that it has three cells (one for each subject), and an inner cell with the result from the isdir function. Each subject should have a 0 in the output variable if the subject folders do not exist and a 1 if they do. This output variable is created for any wrapper automation run. It consists of a cell array with cells for each iteration/subject. These iteration/subject cells contain a cell array with a size of functions by outputs.

3. Creating a pipeline with wrapper automation
Quit out of the functions to run GUI page, and reselect wrapper automation. This time we will run wrapper_automation to create a pipeline for Check Registration (you will need SPM for this portion of the tutorial). First, choose subjects and select only *one* subject this time.

Next, select "add function" and choose "auto_batch.m". This will load the batch editor (note, if you have multiple SPM versions, you will be asked to choose which version to use). Select SPM from the toolbar, choose "Util" then "Check Registration". Highlight "Images to Display" then click the right arrow to set that variable as an input to use. Pressing the left arrow will undo this. Now close out of the batch editor, which will open a file window. This window is used to choose a directory to save the resulting matlabbatch jobs. However, cancel this window and a new window will open with the choices being subject fields. Select "subjFolders" as the folder to save jobs into. This means that when this function is run, it's matlabbatch files will be saved within each chosen subject's main folder in a /jobs/[job name] folder. Change the job name to "Check Registration". Choose "Run" to the "Save jobs or run" window. The jobs will still be saved after running—the only difference is that if you choose "Save", the jobs will be saved but not run. Select "No" to the "Overwrite previous SPM files" window. For any matlab batch job that utilizes SPM, you will be asked if you want to overwrite SPM files. This is most useful when you are re-running first or second-level analyses that would require deleting the previous SPM files. If you have attempted this before, you would notice that SPM stops to ask if you are willing to overwrite the files. By selecting "Yes" to "Overwrite previous SPM files", sawa bypasses SPM's dialog window and automatically overwrites the files. On the other hand, if you select "No", sawa still bypasses this dialog window, but will skip to the next subject/iteration.

Select "add function" again, this time selecting "auto_function.m" and enter "sawa_setcoords". This is a built-in sawa function that allows users to set coordinates for a chosen figure containing axes (e.g., SPM display image figure or xjview). Select "add_function" again and again chose "auto_function.m" to enter "sawa_screencapture". This function will take a screenshot of a chosen figure and save it to a selected location.

Now select "Check Registration" from the functions list and press "set options". Choose "Images to Display" in the following window and select "Subject Array" as the method to set the variable. As before select "Individual" and "subjFolders" as the subfield. Next enter "/Structural/wStruc*.nii" in the text box. (Note you do not need to have this file in order to complete this step). The asterisk is included to demonstrate the wildcard ability. Sometimes, files can have subject names or other attachments so a wildcard allows sawa to choose a file based on its prefix/suffix in such instances.

Rather than cancelling when you return to the "Choose method…" window, this time select "Choose File". Assuming you have SPM, go to the /spm8/templates folder and select the "T1.nii" file from the file selector. Now when you return to the "Choose method…" window, click "Cancel". This will set both the "wStruc*.nii" and "T1.nii" files for each subject when the function is run, meaning both images will displayed.

Select "sawa_setcoords" from the functions list and press "set options". Select only "coords" from the "Choose inputs to set" window. If you look in the command prompt window, you'll notice that the default for hfig is the current figure, which means when it is run it will automatically select the "Check Registration" figure. Next select "k" as the output variable. Choose "Number" as the method to set for "coords" and enter coordinates to use (e.g., [0, 32, -4]). When sawa_setcoords run, the axes for both images will be moved to the chosen coordinate, which can be helpful for checking if registration worked properly. Again, cancel when you return to the "Choose method…" window.

Finally, select "sawa_screencapture" and "set options". This time select only "filename" as the input and "filename" as the output. Then select "Subject Array" as the method to set "filename" and choose "subjFolders". Here, enter "/Check Registration/Check Registration.png" in the text box. This means that the screenshot of the registration images (with axes at chosen coordinates) will be saved to each subject folder within a folder called "Check Registration" (which sawa_screencapture automatically creates).

Now we can save our newly created pipeline. Simply click "save presets" and choose a folder location for the file (e.g., /sawa/Check Registration) and give it a name (e.g., "Check_Registration"). Next, a text box will appear asking for you to enter a "help message". As you may have noticed earlier, on the main GUI page, there is a "help" button. When you have selected a function/pipeline in the listbox, pressing "help" will display any help information for that function/pipeline. This is basically the same as running help([function]) in the command prompt. Enter the following basic help message:

Check Registration
Runs check registration in SPM, sets coordinates to view, and saves screenshots.

Inputs:
Check Registration
- Images to Display - images to check registration
sawa_setcoords
- coords - coordinates to jump to
sawa_screencapture
- filename - output screen capture image

Example:
Check Registration
- Images to Display -
/Volumes/SPM/SUB001/Structural/wStructural.nii
/Applications/SPM8/templates/T1.nii
sawa_setcoords
- coords -
[0, 32, -2]
sawa_screencapture
- filename -
/Volumes/SPM/SUB001/Check Registration/Check_Reg.png

This basic help message helps users understand what a pipeline/function does, what inputs are required, and how it works. Although the help message is not required, it is very helpful… Now click "done" and go back into your main GUI page. Choose your "Check Registration" pipeline and click "help". You should see your help message appear in the command prompt window. If you have structural files, you can move them into the test folders and test out your new pipeline.

4. A command line example with wrapper automation
As a final example of wrapper automation, we will demonstrate the utility of wrapping command line functions. Run wrapper_automation and select your three subjects and choose "per subject". Next, choose "add function" then select "auto_cmd". Enter "fld=" as the command line function (this will allow us to set a variable "fld" for each iteration). Click "set options", choose "other", and leave the option blank. Next choose "subject array" as the method to set the variable, and select "Individual". Then choose the "subjFolders" field and leave the subfield blank. Click "cancel" to set the options and you should see "fld= sa(i).subjFolders{1}" in your command prompt.

Click "add function" and "auto_cmd" again, this time entering "ls" (or "dir" if no Windows). Choose "ls" in the functions list and select "set options". Next choose "edit" from the list of options. Move the cursor to the top of the edit box and enter "$fld". Click "OK" and you should see the following in your command prompt: "ls $fld". Here we are going to use the variable "fld", which we set to each subject's main folder earlier. Now click "load/save/run" and "run". If the subject folders do not exist, you should see the following in your print figure:

wrapper_automation:
test1

fld= “/Users/[username]/Desktop/test1”

test1

ls $fld

test2

fld= “/Users/[username]/Desktop/test2”

test2

ls $fld

test3

fld= “/Users/[username]/Desktop/test3”

test3

ls $fld

Notes: