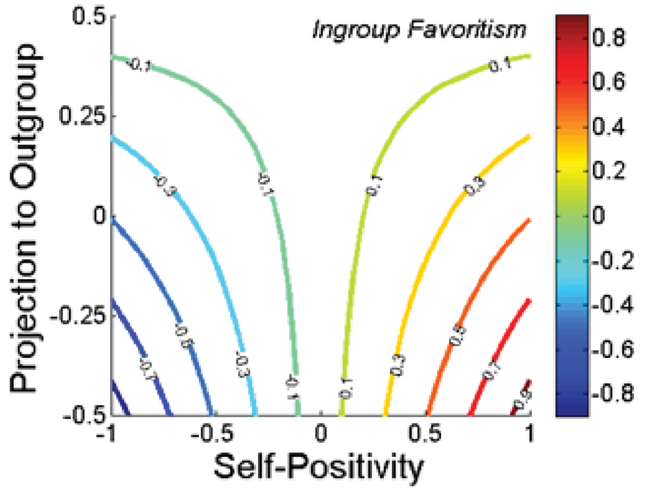
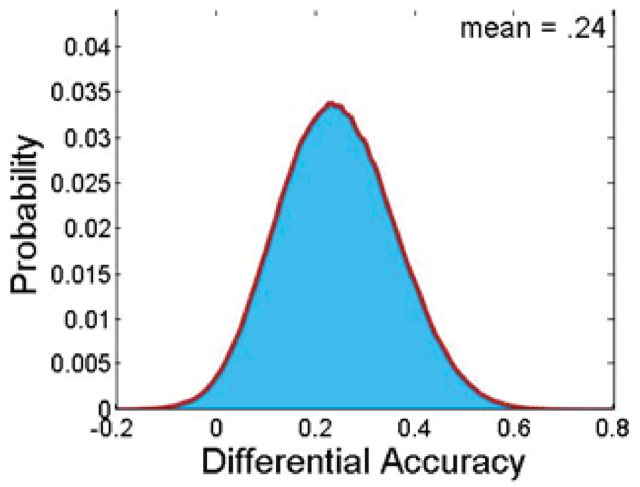
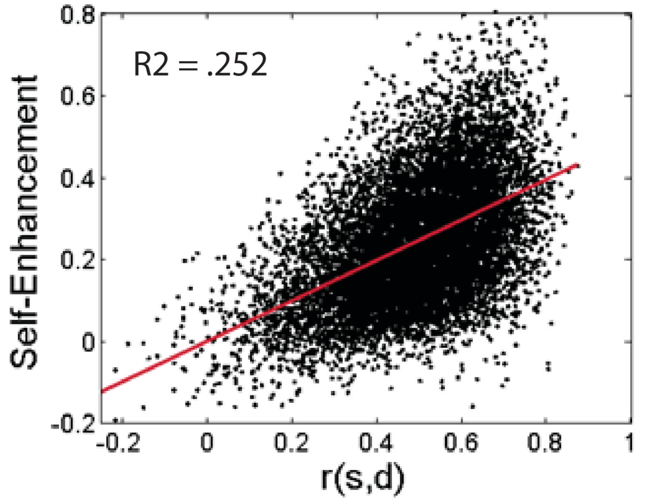
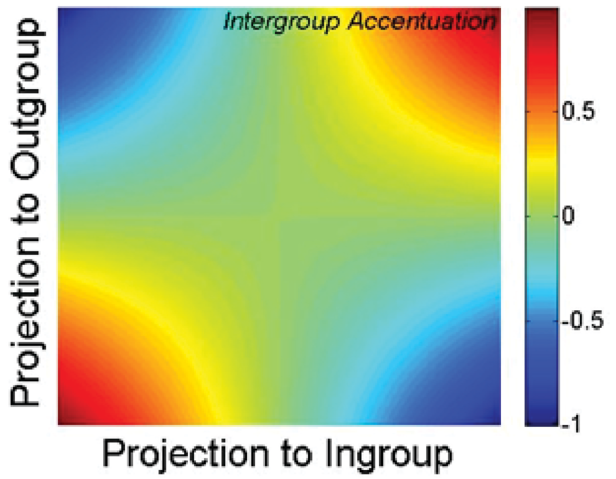
Manual

Inductive Reasoning Model (IRM) Simulator









Joachim I. Krueger
David M. Freestone
Patrick R. Heck

Table of Contents

1.0 Introduction

This manual describes a *Graphical User Interface* (GUI) developed for the exploration of the Inductive Reasoning Model (IRM) and its quantitative implications. The manual has two major sections. The first section is a starter's guide, or 'walkthrough.' The audience of this section is the regular user who is interested in applying the model to questions of interest. The walkthrough is therefore strictly concerned with helping users to discover the GUI's features so that its capability may be fully exploited. The second section, 'Under the interface hood,' was written for advanced users who are interested in the code underlying the GUI. These users will find useful information in this section that will allow them to modify the GUI so that they may pursue questions that currently lie outside of our exposition of the IRM. The manual closes with two brief sections displaying tables of methods and the example code used to generate each figure in the published manuscript.

Since the GUI emerged from our work on the IRM, it has become increasingly clear that the IRM is merely one possible set of psychological questions that the users may address with the GUI. As a tool, the GUI may find use for the exploration of quantitative relationships and constraints in a variety of substantive domains. Aside from statistical assumptions and conventions, the GUI is atheoretical. Its utility is to support and constrain the articulation and the testing theoretical questions in any empirical domain that is interested in the study of correlations among a small set of variables. If the GUI is atheoretical, it is also meta-theoretical. By mapping possible outcomes before empirical study, and by showing what kinds of outcomes are mathematically impossible, the GUI makes a general contribution to the epistemology of induction.

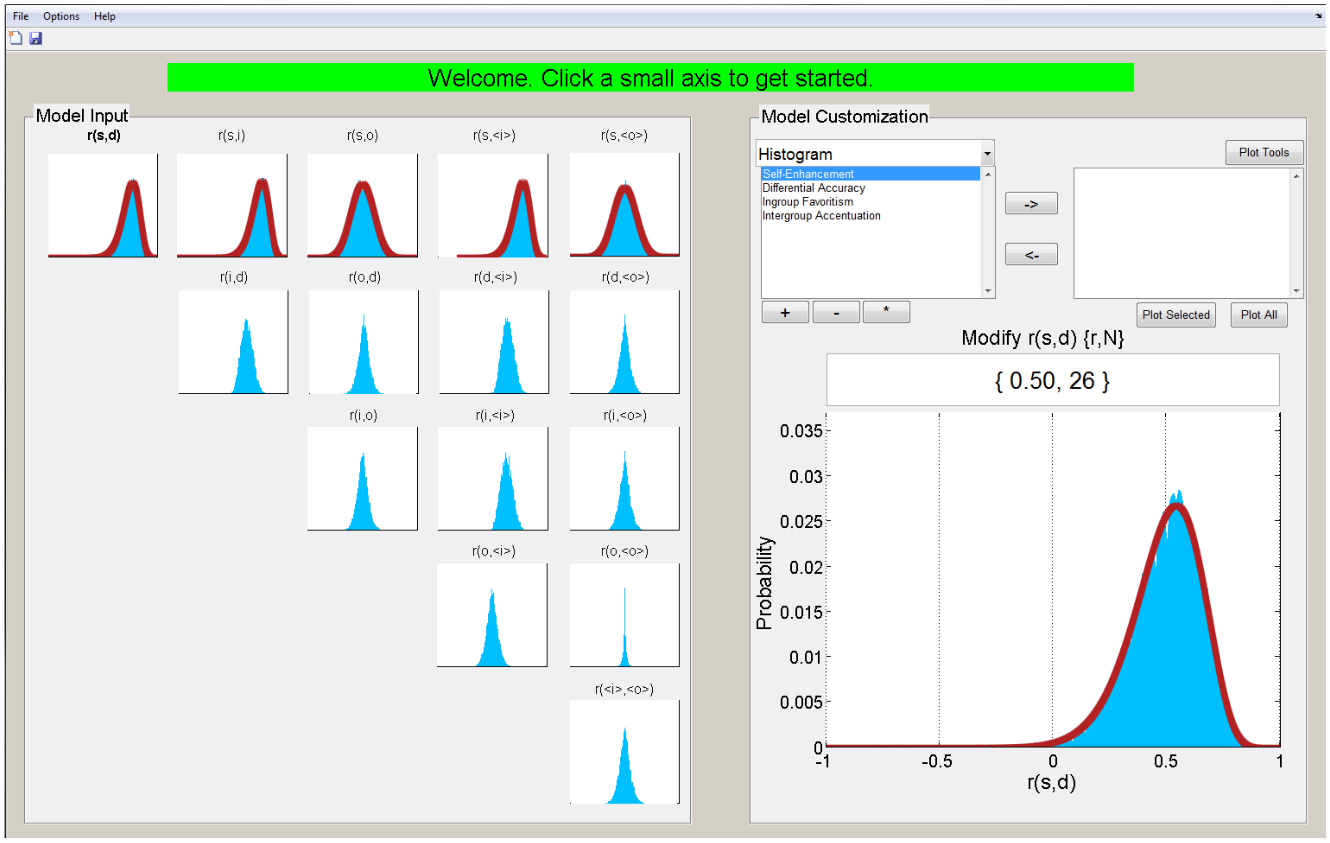
2.0 Starter's Guide (walkthrough)

This starter's guide will walk the user through several full simulations using the *IRM* simulation interface. The walkthrough begins with the simplest simulation (varying no parameters) and increases in complexity with each subsequent simulation (varying a single parameter, varying two parameters, and then calculating correlations between measures over varied parameters). The goal of this starter's guide is to introduce the basic functions of the program without technical terminology - all levels of simulation can be completed without any prior experience programming or running simulations.

2.1: Orienting the user to the interface window

Launching a new simulation will open the interface window. The first time you launch the program, it may take a while for the program to load - this is normal. The following section of the manual will identify and describe each part of the interface.

The interface:


Welcome. Click a small axis to get started.

Histogram

Self-Enhancement

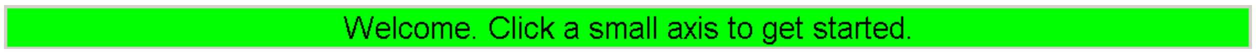
Modify r(s,d) {r,N}

{ 0.50, 26 }

The interface window is composed of several different sections, each with unique graphics, inputs, and functionality. These sections are described below.

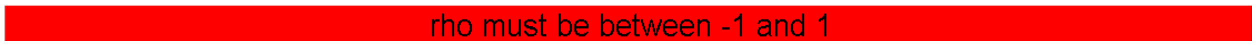
The **message center** is designed to guide the user and to display any error messages that may occur during simulation.

The message center:

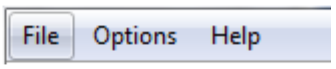

Welcome. Click a small axis to get started.

If an error occurs, the message center will turn red and display the error encountered.

Example of error display:


rho must be between -1 and 1

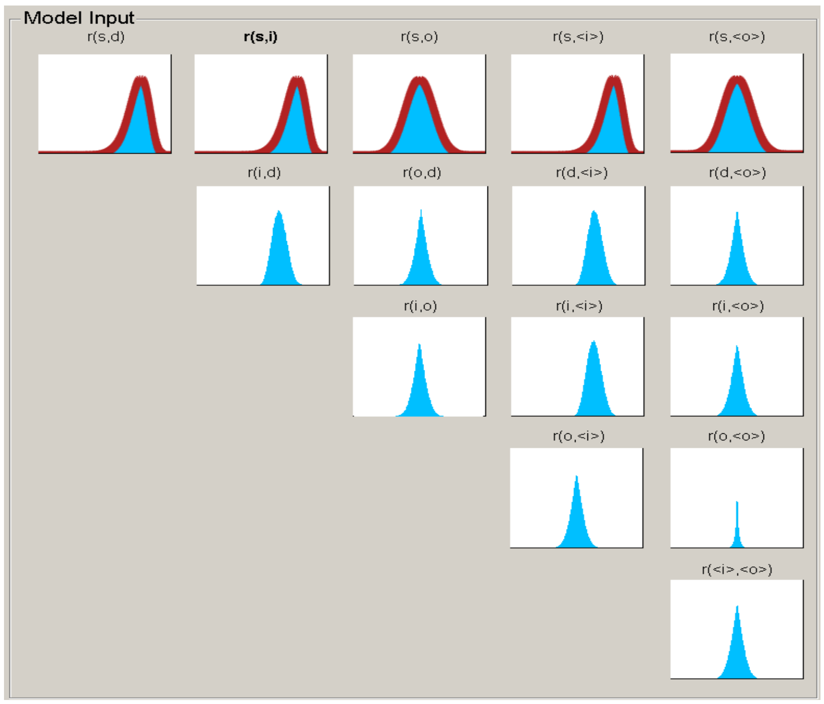
To the upper left of the **message center** is the **menu bar**. Here, the user can save or load a simulation, launch a new simulation, toggle simulation options, view the about screen, and access this manual. The details of the available menu options are explained at the end of the walkthrough.



Below the **menu section** is the **model input section.** The simulator displays the five baseline input correlations on the top row. Derived correlations are displayed below them. The area filled in blue represents the generated samples. The red line represents the theoretical distribution from which samples were drawn.

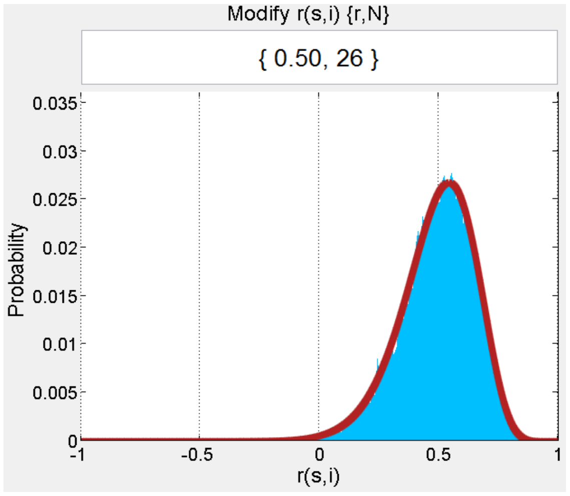
The Model Input section*:*

Clicking on another figure or measure will select it – click on r(s,i) now. Note that r(s,i) is now displayed in bold, indicating that it is currently selected. You can hover over an input to see that measure's label in the tooltip that appears.



The Modify Input section:

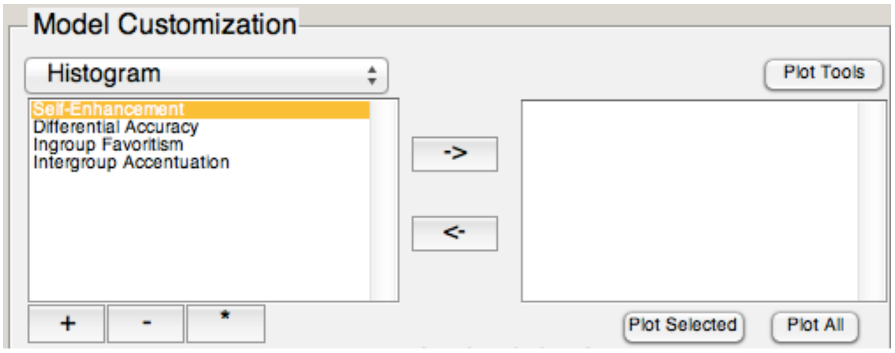
Once the user clicks on a figure in the **model input section,** the interface copies that figure into the larger figure in the bottom right corner. This is the **selected measure/modify input section**. Here, the currently selected measure is displayed as a histogram with its underlying parameters displayed above it. Currently, r(s,i) is selected, with ϱ = .5 and N = 26. We will walk through how to change input parameters in section 2.3



2.2: The model customization section

Just above the **selected measure/modify input section** is the **model customization section**. At default, it contains the four user-defined measures introduced in the manuscript.

Model customization section:



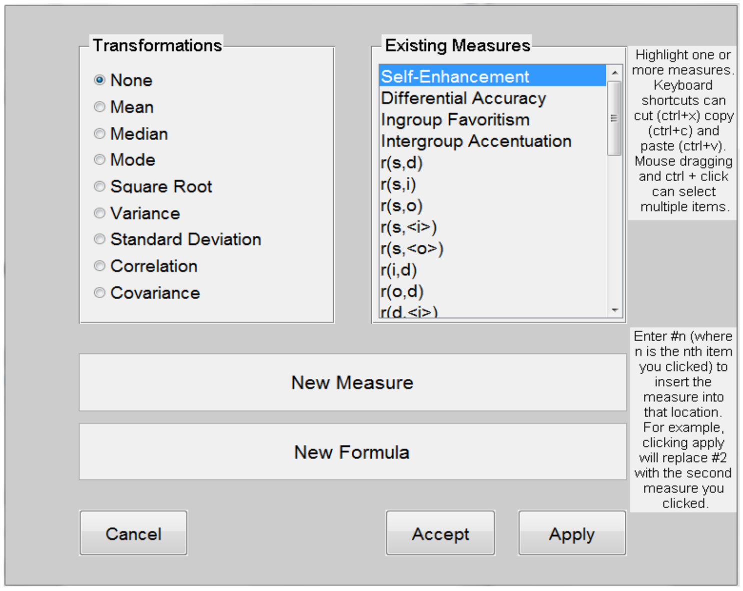
To add a new measure, click the plus button (+). A window will appear where you can enter your new measure.

You can call your new measure anything you would like, but the formula must be entered in terms recognized by the model. For now, add a new measure (type "Example Measure" into the name box) with the formula specified as:

r(s,d) * r(s,i) + r(s,o)

(type the formula into the formula box)

then click 'Accept.'


Transformations
None
Mean
Median
Mode
Square Root
Variance
Standard Deviation
Correlation
Covariance
Existing Measures
Self-Enhancement
Differential Accuracy
Ingroup Favoritism
Intergroup Accentuation
r(s,d)
r(s,i)
r(s,o)
r(s,<i>)
r(s,<o>)
r(i,d)
r(o,d)
r(d,<i>)
Highlight one or more measures. Keyboard shortcuts can cut (ctrl+x) copy (ctrl+c) and paste (ctrl+v). Mouse dragging and ctrl + click can select multiple items.
Enter #n (where n is the nth item you clicked) to insert the measure into that location. For example, clicking apply will replace #2 with the second measure you clicked.
New Measure
New Formula
Cancel
Accept
Apply

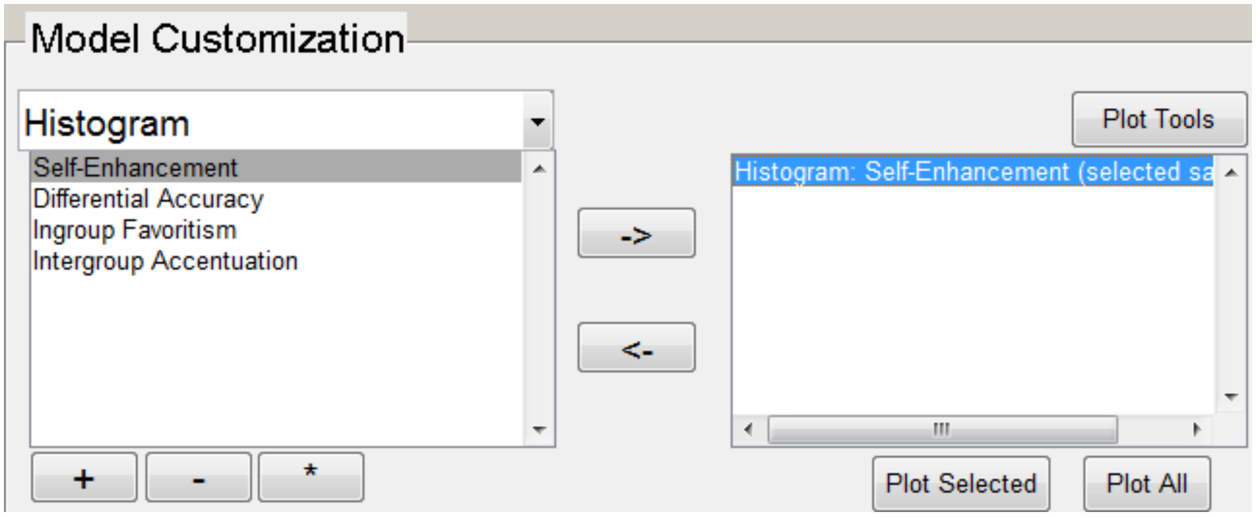
Note that the **model customization section** now contains your new example measure. Any time you would like to edit the name or formula of a custom measure, click the * button (*) to bring up the custom measure window.

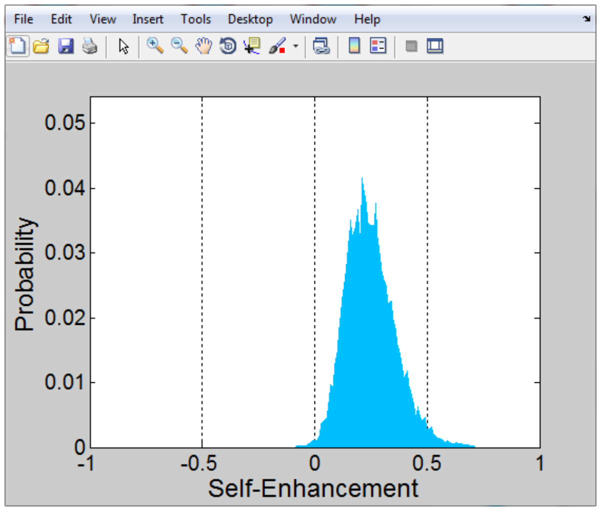
For full details on customizing measures, see section 2.9.

If you would like to remove a custom measure from the simulator, click the measure and then click the '-' button (-). Remember that when you remove a measure, you will no longer be able to use it until you create it again. For illustration, try removing 'Differential Accuracy.'

In order to plot a user-defined measure using the data from the current simulation, a custom measure and plot type combination must be selected. The currently selected plot type (visible in the box above the custom measure list) is 'Histogram.' Click on Self-Enhancement and, with 'Histogram' still selected in the drop down menu, click the arrow button facing toward the right (->) to add the currently selected measure and plot type combination to the plot list.

Plot list containing 'Histogram: Self-Enhancement":





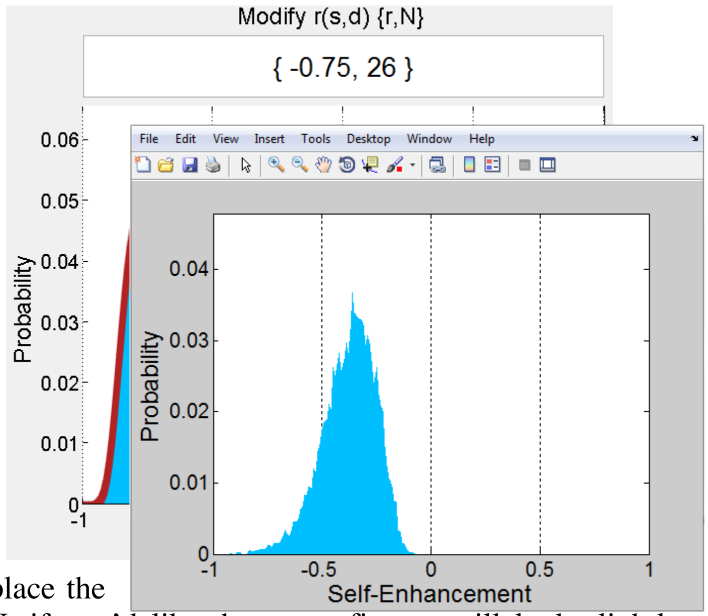
Now that a user-defined measure is ready for plotting, click the 'plot selected' button. This will create a new window containing the target figure. In this case, you can see that Self-Enhancement appears distributed around a mean value of just below 0.25.

You can add as many measure/plot type combinations to the plot list as you like. Clicking the 'plot all' button will create a new window and figure for every item in the plot list. (note: it may plot figure windows on top of each other. Move your figure windows to see all your plots).

To remove a measure from the plot list, simply click on the measure, then click on the left arrow button (<-).

Now that you are familiar with the interface, we will modify one of the input parameters and conduct a new simulation.

2.3: Modifying a single input parameter and plotting a measure.



The simulator launches with all parameters set to the default values specified in the paper. If you want to see the result of changing one of these baseline input measures, you can run a new simulation using your own specified inputs. In order to do this, you must first click on the input parameter you would like to change. Remember that only the top row is modifiable, since all other measures are derived from these five inputs.

For our example, click on r(s,d) and turn your attention to the **modify input section**. The default value for r(s,d) is 0.5, with N = 26. In order to see what happens when r(s,d) is strongly negative, replace the default value (0.5) with -0.75 (you can change N, if you'd like, but your figures will look slightly different than those to the right). After entering the value, press 'Enter' or click on another figure to run the simulation. Note that now the histogram of values for r(s,d) is centered around the mean we specified: -0.75. Any measures that are calculated using r(s,d) ((r(i,d), r(o,d), r(d,<i>), and r(d,<o>)) should have changed as well.

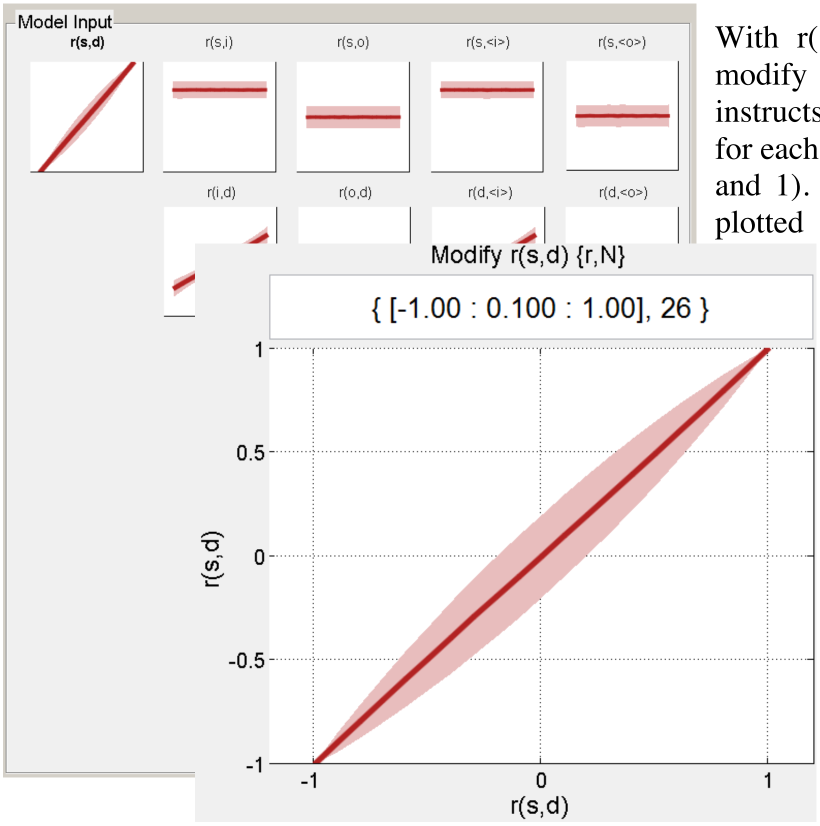
Now that r(s,d) has been modified, let's see what happened to Self-Enhancement. In the **model customization section**, click on Self-Enhancement, make sure that 'Histogram' is selected in the drop down menu, add the measure to the plot list, and click 'Plot Selected.' The resulting histogram confirms that decreasing individuals' positivity of the self-image (r(s,d)) caused the mean value of self-enhancement to become negative (around -0.3, as seen in the figure).

Using this method, it is possible to modify more (up to all) input parameters to reflect any desired user-specifications. Note that once a parameter has been altered it remains set at that new value until the user changes it again or launches a new simulation.

The IRM simulator is also capable of varying an input parameter over a number of steps – this method is explained in the next section.

2.4: Varying a single parameter and plotting a line

In order to view patterns in user-defined measures over variations in a single input measure, you must specify a minimum value, maximum value, and number of steps in which to vary an input. For example, you may want to observe the pattern of change in Self-Enhancement as self-positivity (r(s,d)) increases from negative to positive. To accomplish this, we will instruct the simulator to vary r(s,d) from -1 to +1 in steps of 0.1.



With r(s,d) still selected, enter the following into the modify input box and press enter: [-1 : 0.1 : 1]. This instructs the simulator to run 20 separate simulations, one for each step of the varied parameter (-1, -0.9, -0.8, … 0.9, and 1). In the large plot, the modified input measure is plotted against itself as a manipulation check – the resulting identity line represents r(s,d) plotted in both the x- and y-axes. You can also vary the input with specific values by entering them in brackets and separated by commas: [0, 0.33, 0.5, 0.66, 1]

Notice that the data displayed in the **model input section** are now represented by line plots. For line plots, the varied parameter is always plotted on the x-axis and the target measure is plotted on the y-axis. The thick red line represents the mean of the target measure and the light red shading represents the standard deviation around the mean at each step of the simulation.

Now that r(s,d) has been varied over several steps, we can see how Self-Enhancement varies as positivity of the self-image increases. In the **model customization section,** select Self-Enhancement. Because we are now plotting over variation in a single parameter, you should select ‘Line’ as the plot type (in order to view mean and standard deviation values over a series of simulations). Add this combination to the plot queue and click ‘Plot Selected.’ This will plot the mean value as a dark red line and the standard deviation as a lighter red shading for Self-Enhancement at each simulated step of r(s,d). The resulting figure is the *IRM’s* prediction for how Self-Enhancement increases with self-positivity.

The simulator is also able to vary up to two input parameters simultaneously. The following section explains how to vary a second parameter and plot the resulting simulation.

Note: Once a parameter has been varied, it will not reset to its default value until the user launches a new simulation or manually resets it. If you want to vary a different parameter than the one currently varied, you must reset the first parameter to a single value before doing so.

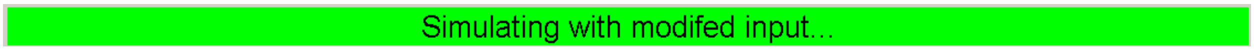
2.5: Varying two parameters and plotting in three dimensions

Now that r(s,d) has been varied from its most negative to most positive, let’s vary a second parameter and observe the effect on our target measure, Self-Enhancement.

Remember that when you click on another input measure, r(s,d) will remain varied unless you modify it again or launch a new simulation. Because Self-Enhancement is directly related to both self-positivity and projection to the ingroup, click on r(s,i). The modify input box should now display the default value for r(s,i): 0.5. Let’s see what happens as r(s,i) increases from 0 to 1. Type [0 : 0.1 : 1] into the modify input box and press enter. This will vary r(s,i) from 0 to 1, in 11 steps of 0.1.

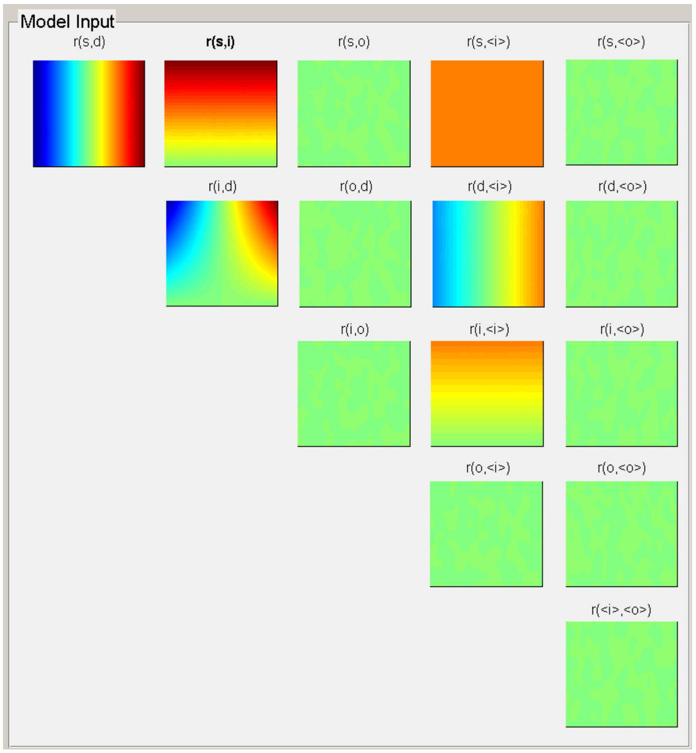
Note that any time you vary two parameters at once, the simulation time increases drastically. In this case, it should take between 15 and 60 seconds (depending on your computer). Increasing the number of points to simulate or the number of steps in which to vary input parameters will cause the simulation to take several minutes. The message center will update you on the progress of the simulation or tell you if there was an error.

Message center display during a complex simulation:



Simulating with modifed input...

Once the simulation is finished, notice that the **model input** section now displays all measures as heat maps. Here, the displayed values are normalized so that one color represents the same value in all figures. Because of this, some figures will be noisy and convey no meaningful information (such as r(s,o), in this case).



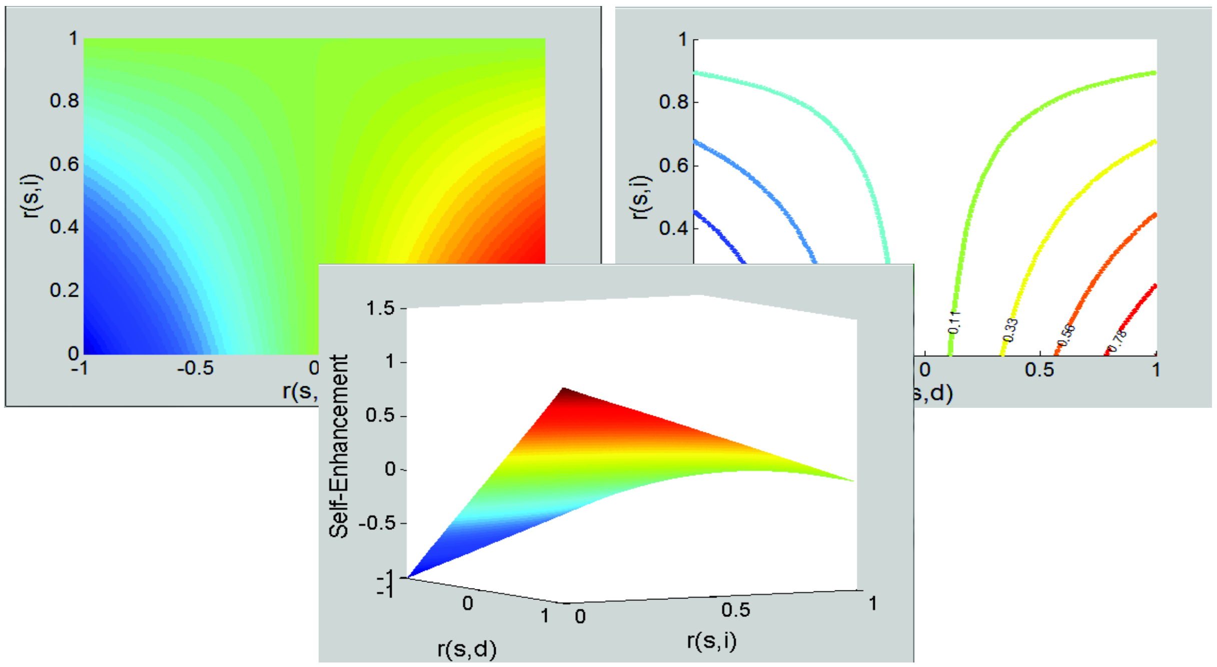
Now that two parameters have been varied, let's see what happens to our target measure, Self-Enhancement. Once two parameters have been varied, several three-dimensional plot types become available. For now, select 'heatmap' and plot Self-Enhancement. The plotted value increases with the warmth of the color. It appears that generally, Self-Enhancement increases (gets warmer) with higher self-positivity (plotted on the x-axis) and decreases (gets cooler) with higher projection to the ingroup (plotted on the y-axis).

The second 3D plot available to the user is the contour plot. This type of plot displays the same information as the heat map, but instead displays actual values on a series of contours. Create a contour plot of Self-Enhancement now and compare it to the heatmap.

Finally, measures can also be displayed using a surface plot. This plot is similar in display to the heatmap, but by toggling the "Rotate 3D" button (⟳) in the figure toolbar, you can manipulate the plot in three dimensions by clicking and dragging the surface. Create a surface plot, toggle the "Rotate 3D" option, and drag the surface of the plot to get a feel for the three-dimensional display.

For computational reasons, the IRM simulator can only vary two parameters at one time. This is the highest-level simulation the interface is capable of running. The final (and most complex) ability of the simulator described in the paper is to observe the correlation between two user-defined measures over two varying measures – this is described in the next section

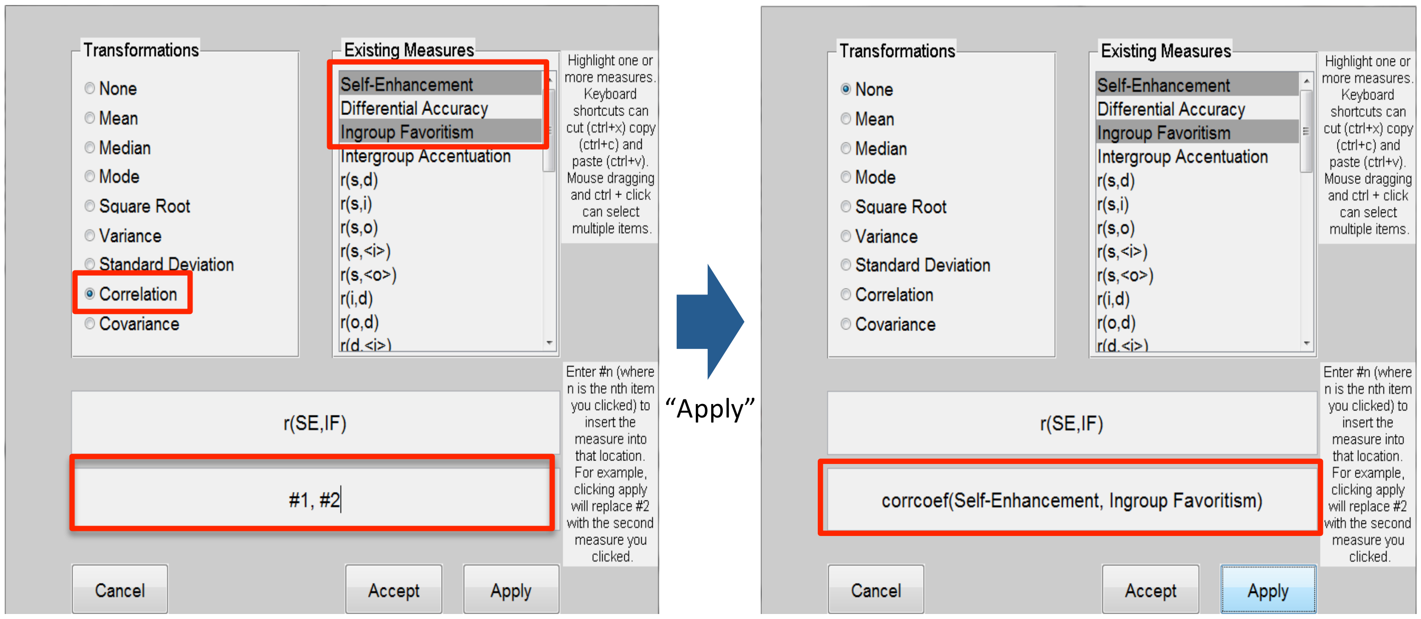
Three types of 3d plots: Heatmap, Contour Plot, and Surface Plot



2.6: Correlating and plotting two user-defined measures.

We have run a simulation where r(s,d) is varied from -1 to 1 and r(s,i) is varied from 0 to 1. As discussed in the paper, the relationship between two measures of theoretical interest can be computed and observed over varying input parameters. Because Self-Enhancement and Ingroup Favoritism are both calculated involving some combination of these two input measures, it may be useful to view the unique relationship between them as their underlying inputs vary.

To do this, you must first create a new user-defined measure specified as the correlation between Self-Enhancement and Ingroup Favoritism. In the **Model Customization Section**, click on the plus button to create a new measure. Name this new measure ‘r(SE,IF).’ For the formula, enter “#1, #2” and click on ‘Self-Enhancement’ and ‘Ingroup Favoritism.’ This will instruct the GUI to fill in #1 and #2 with the first and second measures you selected. Finally, select ‘correlation,’ and click apply (example below).



Transformations: None, Mean, Median, Mode, Square Root, Variance, Standard Deviation, [✓ Correlation], Covariance

Existing Measures: Self-Enhancement, Differential Accuracy, Ingroup Favoritism, Intergroup Accentuation, r(s,d), r(s,i), r(s,o), r(s,<i>), r(s,<o>), r(i,d), r(o,d), r(d,<i>)

Highlight one or more measures. Keyboard shortcuts can cut (ctrl+x) copy (ctrl+c) and paste (ctrl+v). Mouse dragging and ctrl + click can select multiple items.

r(SE,IF)

#1, #2

Enter #n (where n is the nth item you clicked) to insert the measure into that location. For example, clicking apply will replace #2 with the second measure you clicked.

“Apply”

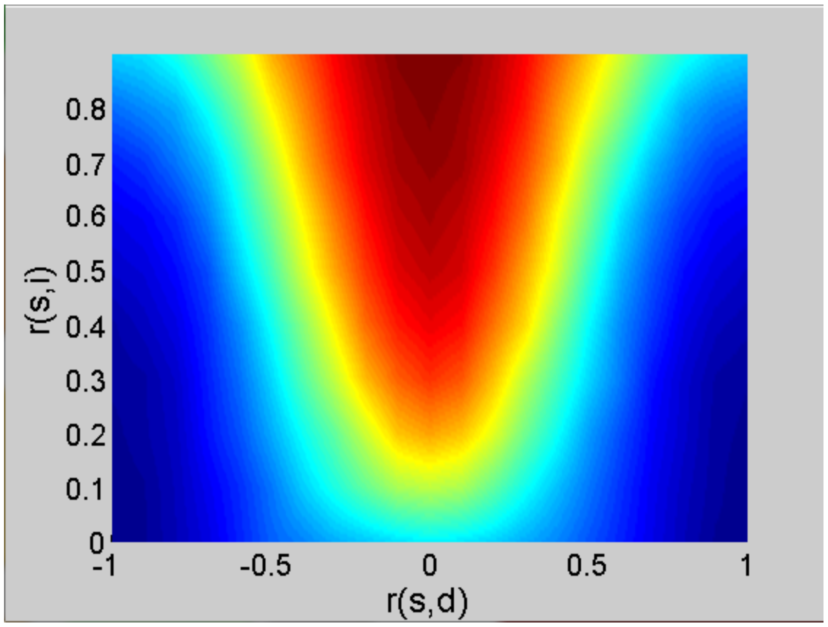
Transformations: [✓ None], Mean, Median, Mode, Square Root, Variance, Standard Deviation, Correlation, Covariance

r(SE,IF)

corrcoef(Self-Enhancement, Ingroup Favoritism)

Alternatively, you can simply type the function and measures you would like. In this case, you would type ‘corrcoef(Self-Enhancement, Ingroup Favoritism) into the formula box. **For more details on customizing measures, see section 2.9.**

Click accept and plot a heatmap of this measure. The displayed colors represent the Pearson’s *r* value for Self-Enhancement and Ingroup Favoritism at each step of the each varied parameter.



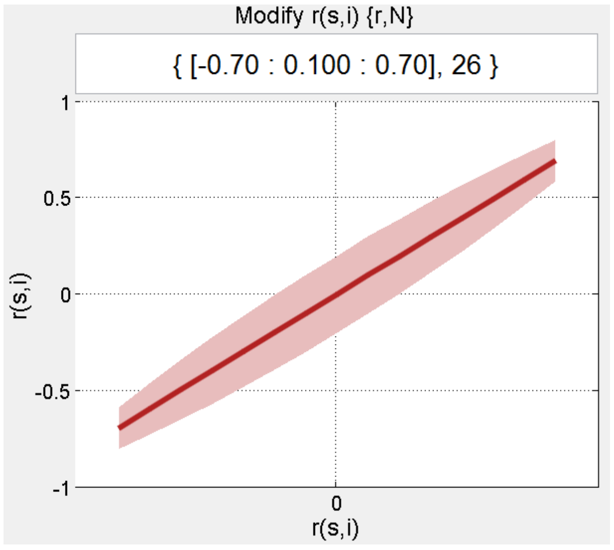
2.7: Plotting select data.

After generating a complex and time-consuming simulation, we have made it possible for the user to plot simpler graphs displaying a selection of data without having to recompute a simple simulation. This function allows you to display a histogram or line of data at any given step of a varied parameter. The ability to create simpler figures given complex simulations is described below.

Plotting a histogram when a single parameter has been varied

When a single parameter has been varied over a series of steps, the data are best visually represented as a line representing the mean and standard deviation calculated at each step of simulation. It is possible to view a histogram of the data at any single step of that variation, however, using the following technique. Note that for this example, r(s,i) has already been varied from -0.7 to 0.7 in steps of 0.1.

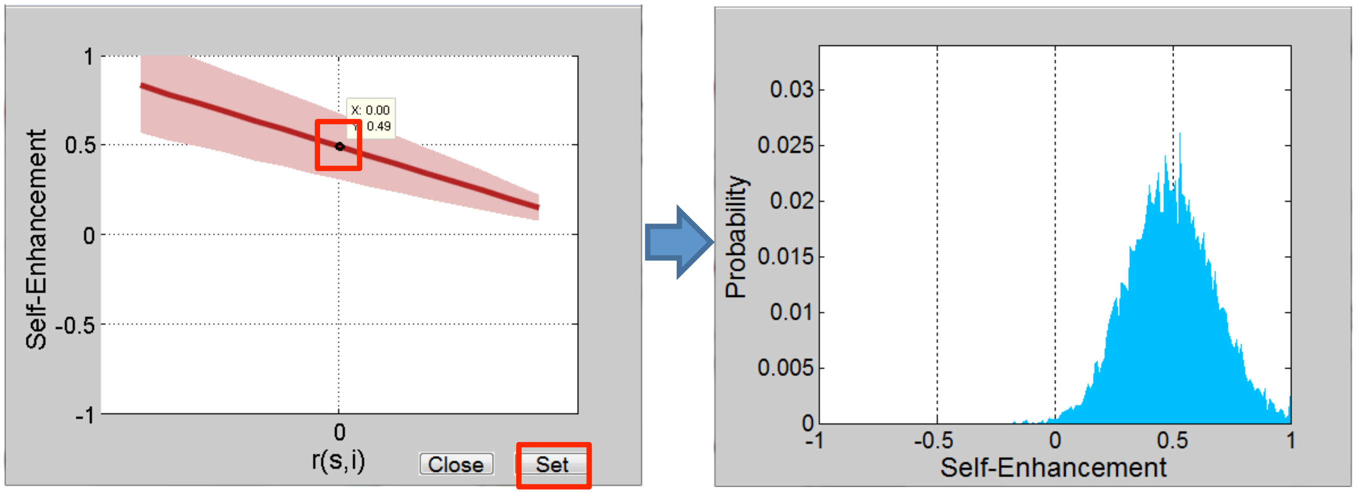
If you would like to view Self-Enhancement when r(s,i) has a mean of zero, add a histogram plot of Self-Enhancement to the list of plots and click ‘Plot Selected.’ Note that a line plot is generated.


Modify r(s,i) {r,N}

{ [-0.70 : 0.100 : 0.70], 26 }

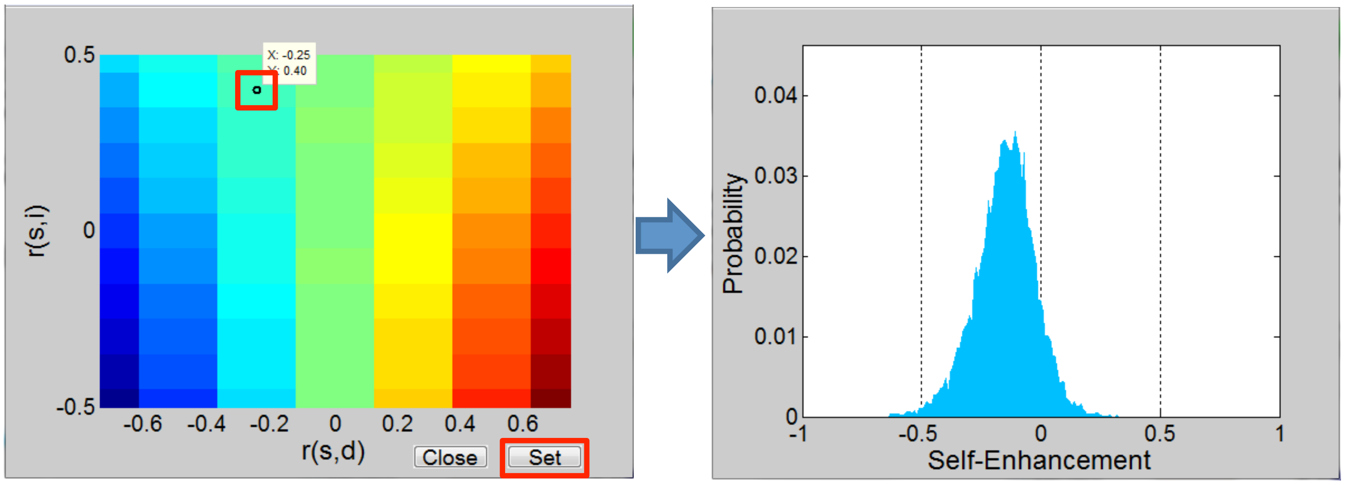
A line plot is displayed because a single histogram is impossible to generate for all simulations of a varied parameter. In order to view a histogram of Self-Enhancement values where r(s,i) is equal to zero, move your mouse over the red line where the x-axis displays zero. Note that tooltip box that follows your mouse: this displays the 'slice' of the simulation that will be plotted as a histogram. Click your mouse with the black dot placed where zero is marked on the x-axis, then click 'set.'

The resulting histogram represents the distribution of simulated Self-Enhancement data points for the simulation where r(s,i) is equal to zero. In essence, we have selected and plotted one of the fifteen simulations that was run (at - 0.7, -0.6, -0.5, -0.4, …, 0.5, 0.6, 0.7).



Plotting a histogram when two parameters have been varied

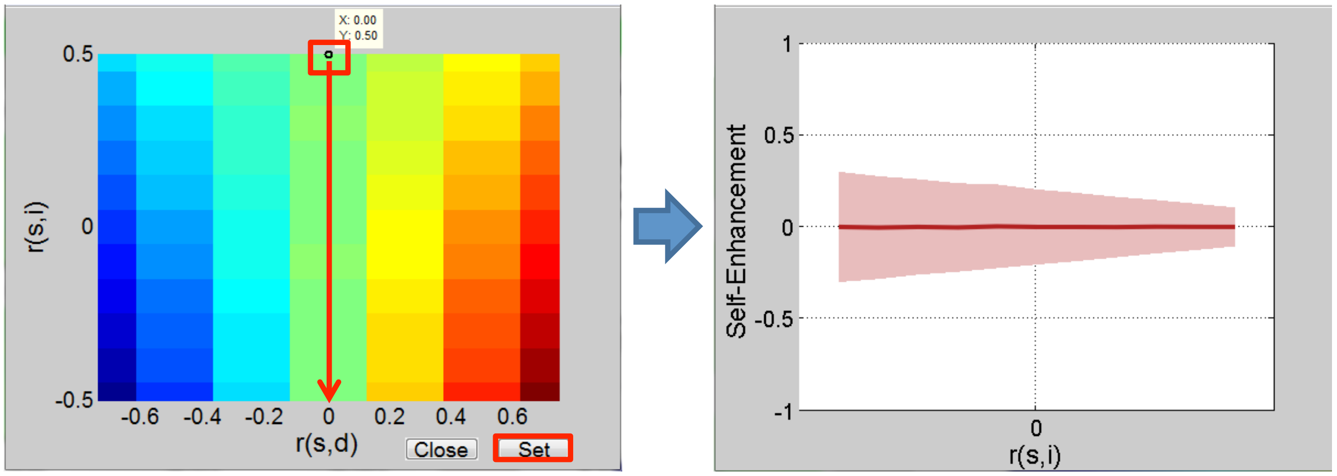
A similar technique is available when two parameters have been varied. This allows the user to plot a histogram at any particular point of each of two varied parameters. The user must simply click the mouse where s/he wishes to select and plot the distribution of a single measure. In the example below, r(s,d) has been varied from -0.75 to 0.75 in steps of 0.25, and r(s,i) has been varied from -0.5 to 0.5 in steps of 0.1. We will plot a histogram of Self-Enhancement, selecting a point where r(s,d) is simulated with a mean of -0.25 and r(s,i) is simulated with a mean of 0.4.



Plotting a line when two parameters have been varied

Finally, you may want to select a series of parameter variations to view when two parameters have been varied. This is equivalent to selecting a linear slice of a heatmap or contour plot. The same method is used to do this, but the mouse must be dragged along the series of variations you would like to view.

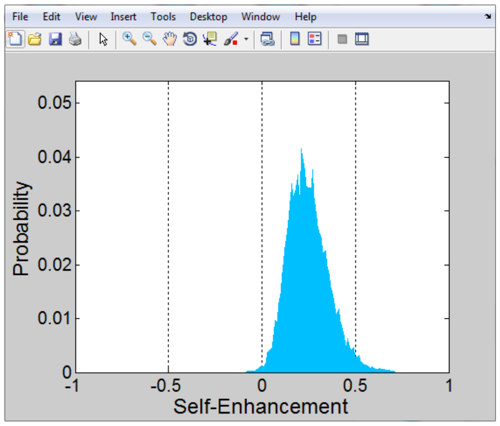
To view variation in Self-Enhancement as r(s,i) is varied where r(s,d) has a mean of zero, drag your mouse vertically down the center of the plot from top to bottom. When finished, click set. The resulting line plot represents Self-Enhancement at each varying step of r(s,i), with r(s,d) held constant at zero.



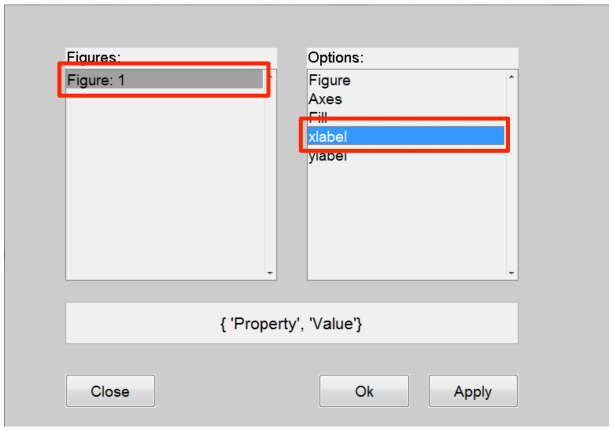
2.8: Modifying Plot Properties Using Plot Tools

We have also included in the interface the ability to modify properties of user-generated figures. The "Plot Tools" button in the **model customization section** will allow you to edit all modifiable properties of a figure that you create – a brief example follows.

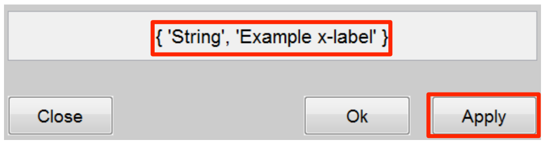
In order to modify the properties of a figure, you must have at least one figure open. With the default simulation inputs specified, create a histogram of Self-Enhancement. It should look similar to the default figure to the right. In this example, we will replace the default x-axis label to reflect our custom specification.



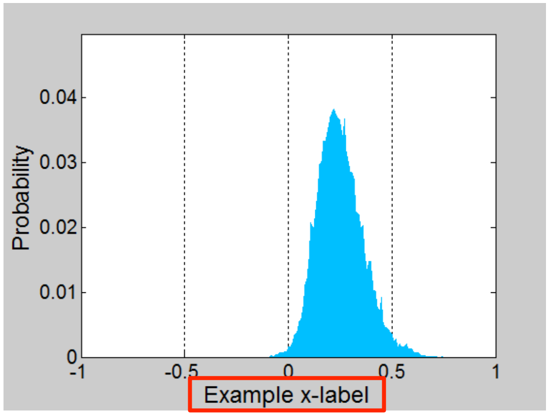
Now that you have an open figure, click on the button labeled "Plot Tools." This will open a new window. Click on 'Figure 1' to specify that you would like to edit the figure you just created. This should populate the empty box to the right with the objects available for modification. Now click on 'xlabel,' to specify that you would like to modify the text (string) value of the x-axis. Your plot tools window should look like the example to the right. Note that the property modification box displays {'Property', 'Value'}: these defaults instruct you to enter values in this format.



To change the x-axis label, we must instruct the simulator to modify the 'String' property of the 'xlabel' object. With 'xlabel' still selected, enter the following into the text box: { 'String', 'Example x-label' }. In all cases, the first element you enter will instruct the simulator to target that particular property. What you enter for 'Value' will be what you set the property to.



Clicking 'Apply' will apply your changes. Clicking 'OK' will apply your changes and close the plot tools window. Click either button and note that the x-label on our histogram of Self-Enhancement now reflects the change.



You can modify any property that isn't made read-only by Matlab. An exhaustive list of objects and their properties can be found here.

2.9: Menu Options

The **Menu Section** provides several options to the user. These options are detailed below.

File menu:

New: Closes the window of the current simulation and opens a window with a new simulation.

Open: Loads a previously saved Matlab workspace (.mat) file into the current simulation interface.

Save: Saves your current simulation in the form of a Matlab workspace file (.mat).

Save As: Allows you to rename the currently saved simulation.

Options menu:

Add notes: Opens a window where you can enter in any notes you wish to keep. These notes are stored in this window.

Number of data points: Allows you to increase or decrease the number of data points to draw for each simulation. The default is 10,000. Larger numbers will produce cleaner simulations but will take longer and consume more computational resources. The maximum is 1,000,000.

Use stored distributions: Toggling this option instructs the simulator to use previously generated sampling distributions if they exist. See the Simulation object section of the manual for more information. The default is set to 'on.'

Interpolate: Toggling this option instructs the simulator to apply a smoothing technique to heatmaps and contour plots. The default is set to 'on.'

Help menu:

About: Opens the about screen with contact and version information.

Manual: Opens the manual in PDF form.

2.10: Example of Detailed Measure Customization

For the ease of the user, we have created a measure customization interface for those interested in exploring measures that require either mathematical manipulations or references to other preexisting measures. The four ways to navigate the add/edit measures window are described below.

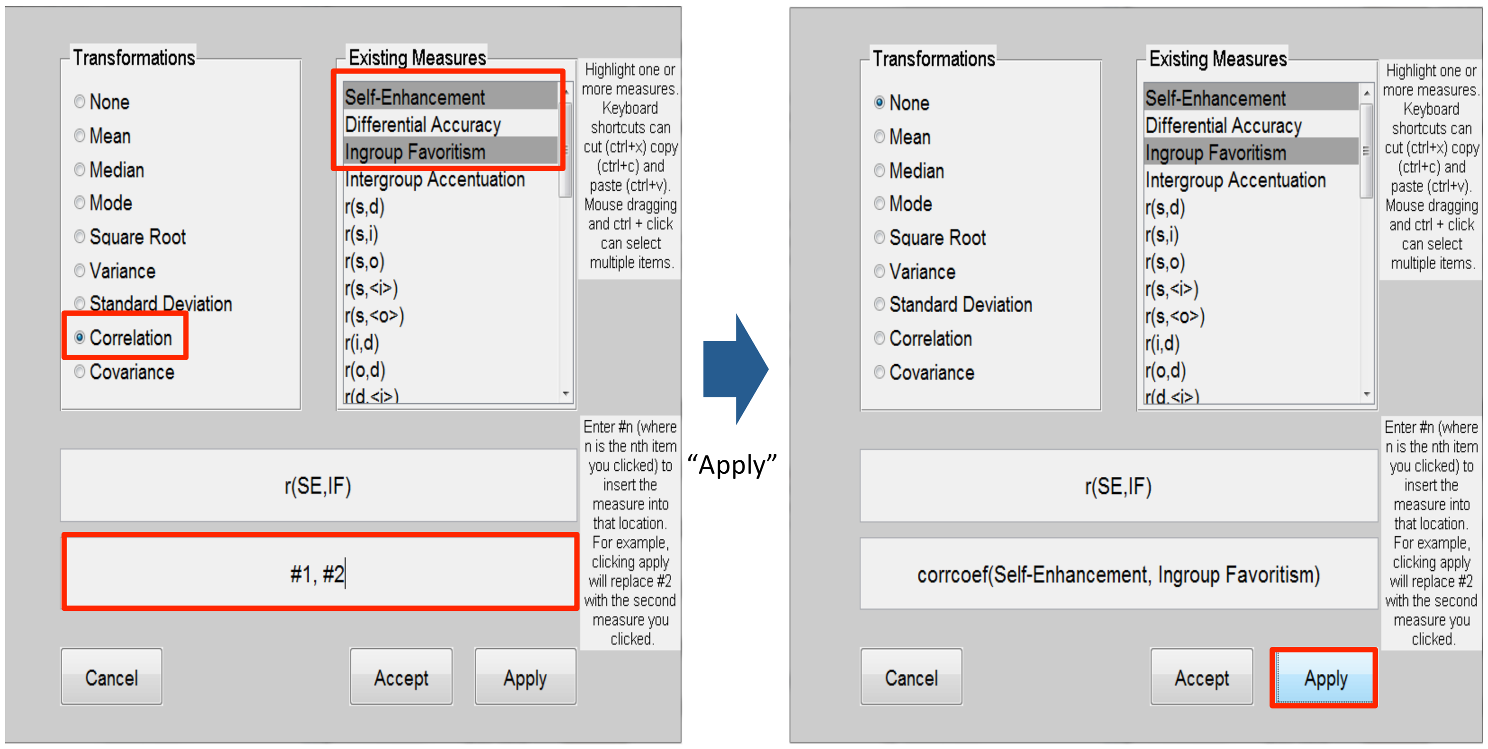
1.) The approach that gives the user the most control over measure customization is simply to type in the desired formula for any new measure using your keyboard. Remember that formulas must be entered using Matlab-recognized operators (for example, * for multiplication, or corrcoef(measure1,measure2) for a correlation between two measures).

2.) Users can also copy and paste measures into the formula box using standard copy/paste keyboard shortcuts. To copy a measure, select it from the 'Existing Measures' list and press Ctrl + C. To paste this measure, click on the formula box and press Ctrl + V.

3.) For users who prefer not to use keyboard shortcuts, you can enter the pound sign (#) followed by a number in place of any measure you would like to enter into the formula box. Clicking the 'Apply' button then replaces each # symbol with a highlighted measure in the order specified by the number following it. You can use ctrl + click or drag the mouse to select multiple measures. For example, entering into the formula box: [#1, #3, #2], selecting three measures, and clicking apply will combine the selected measures into a single formula representing the first, third, and second measures you clicked (in that order). For an example of this, refer to the illustrative figure following this section.

4.) For users who are unfamiliar with Matlab operators, the ‘Apply’ button will also prepend the formula entered in the formula box with an operator and a set of parentheses. To do this, select the radio button corresponding to the operator you would like to include and click the ‘Apply’ button. For example, clicking the ‘Apply’ button with “Self-Enhancement * Ingroup Favoritism” in the formula box, and ‘Mean’ selected in the transformations list, would enter “mean(Self-Enhancement * Ingroup Favoritism)” into the formula box.

A complete example of customization methods #3 and #4 can be seen in the figure below.



Transformations: None, Mean, Median, Mode, Square Root, Variance, Standard Deviation, [✓ Correlation], Covariance

Existing Measures: Self-Enhancement, Differential Accuracy, Ingroup Favoritism, Intergroup Accentuation, r(s,d), r(s,i), r(s,o), r(s,<i>), r(s,<o>), r(i,d), r(o,d), r(d,<i>)

Highlight one or more measures. Keyboard shortcuts can cut (ctrl+x) copy (ctrl+c) and paste (ctrl+v). Mouse dragging and ctrl + click can select multiple items.

Enter #n (where n is the nth item you clicked) to insert the measure into that location. For example, clicking apply will replace #2 with the second measure you clicked.

r(SE,IF)

#1, #2

Transformations: [✓ None], Mean, Median, Mode, Square Root, Variance, Standard Deviation, Correlation, Covariance

r(SE,IF)

corrcoef(Self-Enhancement, Ingroup Favoritism)

3.0 Under the Interface hood

The *IRM* Simulation package is built on the widely used and commercially available MATLAB(R) software package (website). Built with simplicity and generality in mind, the software is split into three components. The first is a simulation component designed to run the underlying model computations. The second is a graphics component designed to give the user the graphing tools we have found useful for exploring the *IRM*. The last is the user interface to make the process as friendly as possible.

Part of the usefulness in splitting the simulation package into components is its generality. Users familiar with MATLAB can take full advantage of it by downloading the software files. For practical reasons, the graphics and interface components add some restrictions on what can be done (more below). A proficient MATLAB user can skip these restrictions by using the simulation component directly.

Our goal in this section of the manual is to twofold. First, to give proficient MATLAB users the tools they need to go beyond the interface. The second is to give those unfamiliar with MATLAB the tools they need to begin to move beyond the interface. The interface includes an option to write your own MATLAB code. We provide the tools (with examples) for doing so below.

3.1 Simulation

The simulation object is responsible for computing, modifying, and storing simulation data. It stores an input matrix containing the six primary baseline correlation inputs in the *IRM*, any modifications made to these input measures, user-defined measures representing social-psychological phenomena (as predicted by the *IRM*), user-defined (custom) measures of interest, and samples generated through simulation. Several simple functions allow the user to view and modify any of these variables. The simulation object also has the ability to save simulations or load a previous one.

The following section walks the user through all available properties and methods. A summary table containing these properties and methods can be referenced at the end of the section (Table 1). The simulation component (called *object* in Matlab), has properties (variables), and methods that act on these variables. To full understand how to use the simulation object, you should become familiar with the properties and methods. They are described below.

simulate() (method)

This command initializes the simulation object, creating a new simulation. This simulation object stores both the variables and the methods used when running a simulation. This includes storing and manipulating the input matrix, adding or removing new measures, and generating and storing simulation data.

Use:

```
>> sim = simulate();
```

Note that the user can create as many simulation objects as desired, and each one can run a different simulation. Beware, however, that more extensive simulations with one simulation object, and having more than a single simulation object at a time, will require more computational resources. Very large simulations may require a large amount of system memory, CPU usage, or both.

Input (property)

The *IRM* has five base *input* measures. These measures specify the correlation between individual's ratings over traits. Each input measure is specified by two parameters: ϱ – the true underlying correlation between traits in the population, and *N* – the number of traits each individual rates. These two parameters specify the population distribution from which individuals are drawn (see Appendix A and the text for details).

The ***Input*** property stores these input measures as a matrix. The first column is the *name* of the measure (see main text, figure below). The second and third columns are the ϱ and *N* values. These values can be changed with the ***modify_input()*** method.

modify_input() (method)

Syntax:

```
sim.modify_input( measure )
```

Where measure is a valid measure to modify, specified in the cell format: {*name,* ρ , *N*}.

This function allows the user to modify the ***Input*** matrix. Modifications can be made to ρ or *N* for any baseline input measure. Input values can be set to a single point (e.g., 0.25) or several points. To input multiple specific values, enclose the values in brackets, and separate by commas: [0.25, 0.5, 0.75]. To input a range of values, specify the range and the distance between points. For example, the input [-1 : 0.1 : 1] sets the input to -1, -0.9, -0.8, -0.7, …, 1. Any modified input measure remains modified until the user changes it or initializes a new simulation. For computational reasons, we limit the number of simultaneously varied inputs to two.

Use:

```
>> sim.modify_input(input);
```

Example use:

```
>> %% Modify Input Example
>>
>> sim = simulate(); %create new simulation object
>>
>> sim.Input %view default input matrix

ans =

    'r(s,s)'      [     1]    [26]
    'r(s,d)'      [0.5000]    [26]
    'r(s,i)'      [0.5000]    [26]
    'r(s,o)'      [     0]    [26]
    'r(s,<i>)'    [0.5000]    [26]
    'r(s,<o>)'    [     0]    [26]

>> %modify a single measure
>> name = 'r(s,d)';          %Vary r(s,d)
>> rho = -1:.1:1;            %Vary from -1 to 1 in steps of .1
>> N = 26;                   %Do not vary N
>> input = {name, rho, N};   %Define input variable
>>
>> sim.modify_input(input);  %Execute function
>>
>> sim.Input %view modified input matrix where r(s,d) is now varied

ans =

    'r(s,s)'      [          1]    [26]
    'r(s,d)'      [1x21 double]    [26]
    'r(s,i)'      [     0.5000]    [26]
    'r(s,o)'      [          0]    [26]
    'r(s,<i>)'    [     0.5000]    [26]
    'r(s,<o>)'    [          0]    [26]
```

npoints (property)

This property specifies how many individuals to simulate. That is, it specifies how many random samples to draw for each of the input measures. Increasing this number results in more samples and a slower simulation. Decreasing this number results in fewer samples, but greatly speeds up the simulation. The default number of data points to sample when simulating each baseline measure is 10,000. Unless your computer has 8+ GB of RAM, we recommend a maximum of 1 million individuals.

Use:

```
>> sim.npoints = 100;
```

get_samples() (method)

Syntax:

`sim.get_samples();`

As the workhorse of the simulation, this function is responsible for generating simulation data. Individuals (specified by ***npoints***) are randomly sampled from the underlying distribution specified by ***Input***. These data points are then stored in ***randomsamples***.

Use:

```
>> sim.get_samples();
```

randomsamples (property)

After ***get_samples()*** is called, the resulting samples are stored in ***randomsamples***. When all the parameters in ***Input*** are scalar, the data is stored in a ***npoints*** x 6 double-precision matrix stored inside a 1x1 cell array.

When one or two parameters are varied, the resulting samples are stored as a **npoints** x 6 double precision matrix stored inside an n x m cell array (varying one input parameter results in a 1xm cell array, varying two input parameters results in an n x m cell array). Each cell stores the results from one set of input values. We illustrate this below.

Example use (next page):

```
>> %% Generate and Store Samples Example
>>
>> sim = simulate(); %create new simulation object
>> sim.randomsamples %randomsamples is empty

ans =

     {}

>> sim.get_samples(); %generate samples
>> sim.randomsamples  %samples are stored in a single matrix

ans =

    [10000x6 double]

>> %vary one input measure and get new samples
>> sim.modify_input( { 'r(s,d)', 0:.1:.3, 26 } ); %vary r(s,d) in four steps
>> sim.get_samples(); %get new samples (previous samples replaced)
>> sim.randomsamples  %samples are now stored in a vector of matrices

ans =

    [10000x6 double]    [10000x6 double]    [10000x6 double]    [10000x6 double]

>> %vary two measures and get new samples (note that r(s,d) remains varied)
>> sim.modify_input( { 'r(s,i)', [.2 .8], 26 } ); %also vary r(s,i) in two steps
>> sim.Input %view Input matrix with two varied inputs

ans =

    'r(s,s)'      [         1]    [26]
    'r(s,d)'      [1x4 double]    [26]
    'r(s,i)'      [1x2 double]    [26]
    'r(s,o)'      [         0]    [26]
    'r(s,<i>)'    [    0.5000]    [26]
    'r(s,<o>)'    [         0]    [26]

>> sim.get_samples(); %get new samples with two varying inputs
>> sim.randomsamples  %samples are now stored in a matrix of matrices

ans =

    [10000x6 double]    [10000x6 double]    [10000x6 double]    [10000x6 double]
    [10000x6 double]    [10000x6 double]    [10000x6 double]    [10000x6 double]
```

measures (property)

This property stores the name and formula of each user-defined measure. A measure represents any mathematical manipulation of the six baseline input correlations or any measure that results from them (see main text for details). When initializing a new simulation object, the measures variable is initialized to contain the four user-defined measures referenced in the paper (Self-Enhancement, Ingroup Favoritism, Differential Accuracy, and Intergroup Accentuation).

measures is a *n* x 2 cell matrix. The two columns store each measure's *name* and *formula* (respectively), while each row represents a unique user-defined measure. Users can *add*, *edit*, or *remove* a measure from this list.

add_measure() (method)

Syntax:

```
sim.add_measure( measure )
```

Where measure is a valid user-defined measure specified in the cell format: {*name, formula*}

This function adds a new measure to the list currently stored in the ***measures***. The user is able to create custom measures of varying scope and complexity in order to observe empirical relationships within (but not limited to) the *IRM*.

For any custom measure, the following conventions for name and formula must be followed: text must be in string format and formulas must boil down to one the five input measures. Any Matlab-recognized operators and many Matlab functions (mean, corrcoeff, etc…) can be used.

Use:

```
>> sim.add_measure(measure);
```

remove_measure() (method)

Syntax:

```
sim.remove_measure( measure )
```

Where measure is a valid user-defined measure specified in the cell format: {*name*} or the format {*name, formula*}.

If a user-defined measure appears redundant, cumbersome, or nonsensical, the user can use this function to remove a measure from ***measures***. The same {*name, formula*} syntax applies here, although the user can also shorten the input simply to {*name*}. Once removed, a user-defined measure is no longer available, however, it can be simply added again using ***add_measure().***

Use:

```
>> sim.remove_measure(measure);
```

edit_measure() (method)

Syntax:

`sim.remove_measure( measure )`

Where measure is a valid user-defined measure specified in the cell format: {*name*, *formula*}. This function allows the user to edit the name or formula of any current measure in ***measures***.
Use:

```
>> sim.edit_measure(measure);
```

Example use:

```
>> %% Add, Remove, and Edit User-Defined Measures Example
>>
>> sim = simulate(); %create new simulation object
>> sim.measures %view default user-defined measures

ans =

    'Self-Enhancement'          'r(s,d) - r(i,d)'
    'Differential Accuracy'     'r(i,<i>) - r(o,<o>)'
    'Ingroup Favoritism'        'r(i,d) - r(o,d)'
    'Over Accentuation'         'r(i,o) - r(<i>,<o>)'

>> %Add new user-defined measure
>>
>> name = 'Example Measure'; %name new measure
>> formula = 'r(s,d) + r(s,i) + r(s,o)'; %specify new measure formula
>> measure = {name, formula}; %define variable
>>
>> sim.add_measure(measure) %execute function
>> sim.measures %view user-defined measures including 'Example Measure'

ans =

    'Self-Enhancement'          'r(s,d) - r(i,d)'
    'Differential Accuracy'     'r(i,<i>) - r(o,<o>)'
    'Ingroup Favoritism'        'r(i,d) - r(o,d)'
    'Over Accentuation'         'r(i,o) - r(<i>,<o>)'
    'Example Measure'           'r(s,d) + r(s,i) + r(s,o)'

>> %Remove one existing user-defined measure and edit the formula of another
>>
>> sim.remove_measure( { 'Self-Enhancement' } ); %remove measure
>> sim.edit_measure( { 'Example Measure', 'mean(Ingroup Favoritism)' } ) %edit measure
>> sim.measures %view updated user-defined measures

ans =

    'Differential Accuracy'     'r(i,<i>) - r(o,<o>)'
    'Ingroup Favoritism'        'r(i,d) - r(o,d)'
    'Over Accentuation'         'r(i,o) - r(<i>,<o>)'
    'Example Measure'           'mean(Ingroup Favoritism)'
```

save() (method)

Syntax:

```
sim.save( overwrite )
```

Where overwrite is a logical value specifying whether or not to overwrite a previously saved file.

This function saves all properties of the current simulation object to a filename and location specified by the user. If overwrite is true, the save function overwrites the previously specified save file and path without first prompting the user to enter them.

```
>> sim.save( false )
```

load() (method)

Syntax:

```
sim.load(overwrite )
```

This function loads the properties of a previously saved simulation into the current simulation from a filename and location specified by the user.

```
>> sim.load();
```

r* and *dr (read-only properties)

Generating random samples from a non-standard distribution requires a well-defined and computable distribution to sample from. The correlation coefficient distribution cannot be computed analytically, but it can be approximated to arbitrary precision (see Appendix A). The precision with which we can compute this distribution, and therefore the precision with which we can generate unique samples, is determined by the discrete values over which we approximate the distribution. The more fine-grained these values become, the more accurate the simulation will be.

The property ***dr*** determines this precision; smaller values indicate better precision, but at the cost of computational resources. The default value is ***dr*** = 0.00001, which we have found to be sufficient (see Appendix A). The user is able to modify this variable in order to make these approximated distributions even finer in their discrete points for a more robust simulation, or coarser for a quicker simulation. The ***dr*** property can be adjusted using the ***sample_precision()*** method.

sample_precision() (method)

Syntax:

```
sim.sample_precision( dr )
```

Where dr is a scalar value.

Adjusts the sampling precision of the simulation by setting the ***dr*** and ***r*** properties.

Use:

```
>> % Change property dr to 0.001. Also sets property r.
>> sim.sample_precision( .001 );
```

rcdf (read-only property)

Approximating the correlation coefficient distribution and drawing samples from it takes time. As the distributions are computed, we (optionally) store the result in ***rcdf*** so that subsequent draws using the same parameters become faster. This is a read-only property, but its use can be modified by setting the ***ncdf*** and ***use_cdfs*** properties below.

Previously computed cumulative correlation coefficient distributions generated under some value of ρ are stored in a matrix where the first and second columns represent the values of ρ and N, respectively, and the remaining columns store the cdf. Each row is a new cdf, specified by its ρ and *N*.

ncdf (property)

Simulations that do not reference previously computed distributions will spend time and computer resources computing new distributions before sampling, but saving many distributions to ease computing requires a large amount of system memory. Thus the user can choose how many previously computed correlation coefficient distributions to save. The more distributions stored, the faster the simulation will be (especially for large simulations), but more system memory will be used. ***ncdf*** tells the simulation the maximum number of distributions to store. The default value is 200.

use_cdfs (property)

This property specifies whether or not the simulator should store the computed r-distributions when generating samples. The default value of this property is true (1). The user can set this value to false (0) if s/he wishes to run all simulations without referencing any previously computed distributions. This results in slower simulations, but less system memory is used.

Both of these properties can be set by simply treating them as fields in a structure:

```
sim.ncdf = 100;
sim.use_cdfs = false;
```

compute() (method)

Syntax:

```
sim.compute( measure )
```

Where measure is the name of a formula, or a formula for an undefined measure.

This function is responsible for applying computing the measure specified. Because formulas for custom measures are specified in string formats, it is necessary for the simulator to first deconstruct a measure formula into its constituent parts before performing a calculation. The simulation object does this explicitly.

For example, 'Self-Enhancement' is defined as 'r(s,d) – r(s,i).' ***compute()*** accesses the values for 'r(s,d)' and 'r(s,i)' (as stored in ***randomsamples***) and applies the 'minus' operator, yielding a value for 'Self-Enhancement' for each individual and simulation input parameters.

Note: the ***compute()*** function returns values in a cell array the same size as **randomsamples**, where each cell contains the result of the computation. It is difficult to plot numeric values stored in cell arrays, but cells can be converted to matrix format using the Matlab-native function cell2mat. We show its use in a full example below.

Example:

```
>> sim.compute( 'std(Ingroup Favoritism)' ); %compute the standard deviation for Ingroup Favoritism
```

compute_correlation() (method)

Syntax:

```
sim.compute_correlation( measure )
```

Where measure is specified in the cell format {*measure 1*, *measure 2, … , measure k*}.

The highest-level function of the *IRM* (and this simulator) is to predict and simulate relationships between user-defined measures of interest. Because of the complexity of these relationships, whether varying one or more input parameters, this function is included separately in order to perform the function of **compute*()*** in the specific case of observing correlations between measures. Correlations are returned in a 1 x *k* cell array. Inside each cell array is a double precision matrix of correlations the same size as randomsamples.

An incomplete list of some Matlab functions that may be useful to you follows this section. A comprehensive list of Matlab-native operators can be found here.

correlation_inputchk() (method)

Syntax:

```
is_correlation = sim.correlation_inputchk( measure )
```

Where measure is specified as a string. is_correlation is returned as a logical.

This function checks the formula for the input measure determined by the user. If the user requested the Matlab operators 'corrcoef' or 'cov', this function returns a logical specifying that the target measure formula includes a correlation. The measure can then be entered into **sim.compute**() or **sim.compute_correlation**().

3.3 Table of some useful Matlab functions

Function	**Purpose**
mean()	Computes the mean value
median()	Computes the median value
std()	Computes the standard deviation
sum()	Computes the sum
exp()	Computes the exponent
log()	Computes the natural log
sqrt()	Computes the square root
abs()	Computes the absolute value

3.4 Tables of methods and properties

Item	**Function**	**Implementation**
Properties		
Input	Stores baseline measures and values	*read only*
measures	Stores user-defined measures and formulas	*read only*
npoints	Number of data points per simulation	*read/write access*
randomsamples	Stores all simulated data points	*read only*
dr	Distance between possible simulation values	*read only*
rcdf	Matrix of possible values to sample	*read only*
ncdf	Maximum number of previous cdfs to store	*read/write access*
use_cdf	Determine whether to use and store cdfs	*read/write access*
Methods		
simulate	Create new simulation object, 'sim'	*sim = simulate()*
modify_input	Edit one or more parameters in the input matrix	*modify_input()*
get_samples	Generate random samples given input parameters	*get_samples()*
add_measure	Create and name a new user-defined measure	*add_measure()*
remove_measure	Remove a current user-defined measure	*remove_measure()*
edit_measure	Edit the name or formula of a current measure	*edit_measure()*
compute	Computes values of measures from simulation data	*compute()*
compute_correlation	Calculate correlations between user-defined measures	*compute_correlation()*
correlation_inputchk	Checks to see if target measure requires correlation	*correlation_inputchk()*
sample_precision	Modify property dx to adjust sampling precision	*sample_precision()*
load_variables	Load simulation data or stored cdfs	*load_variables()*
save	Save current simulation	*save()*
load	Load saved simulation	*load()*

3.2 Full working example in code

Following is an annotated full example of launching a new simulation, varying two baseline input measures, generating samples using the modified input, and plotting a user-defined measure. This example is comprised of the same steps that were taken to generate figure 1 in the paper. Note that the resulting figure is unedited.

```
>> %% Full Example (Generate Figure 1)
>>
>> sim = simulate(); %create a new simulation object
>>
>> %Modify the input matrix to vary both 'r(s,d)' and 'r(s,i)'
>> input = {
    'r(s,d)', 0 : .1 : 1   , 25
    'r(s,i)', [0.2 0.5 0.8], 25};
>> sim.modify_input( input );
>>
>> %Draw samples from the distributions specified by the input matrix
>> sim.get_samples();
>>
>> %Compute the dependent measure of interest
>> %Change data storage format from cell to matrix so values can be plotted
>> y = cell2mat( sim.compute( 'mean(Self-Enhancement)' ) );
>>
>> %Plot the measure of interest
>> p = plot( 0:.1:1, y );
```

4.0 Graphics

We have created a set of useful graphing tools to accompany the *IRM* simulation. Of course, these tools do not need to be used with the simulation object, but we have found them helpful. Novice users of Matlab may benefit from the cohesive organization the graphics objects provide, while advanced users may benefit from the transparency and accessibility of its structure.

The graphics object includes a set of plotting methods and stores the relevant Matlab *handles* for later modification. Several simple functions allow the user to view and modify these variables in order to change the way the plot looks. The graphics object also has the ability to save figures for later reference or for use in presentation and publication.

graph() (method)

A graphics object can be loaded via the ***graph()*** command. This initializes a graphics object, but does not plot anything. You can view its methods and properties in the command window

Use:

```
>> gr = graph();
```

plot() (method)

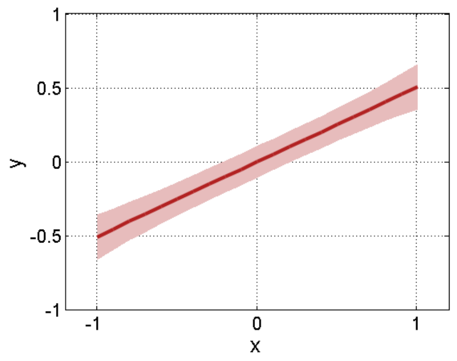
Syntax:

```
gr.plot( sim, measure, plot_type );
gr.plot( sim, measure );
```

Where sim is a valid simulation object, and measure is a valid measure. Optionally, plot_type can be ‘scatter’, ‘histogram’, ‘line’, ‘heat’, ‘contour’, or ‘surface.’ If no plot type argument is supplied, the plot method will choose one based on the simulation object supplied.

If no `plot_type` argument is supplied, the plot method will choose one based on the simulation object supplied. If there are no varied input parameters, the default is ‘histogram.’ For one varied parameter, the default is ‘line,’ and for two varied input parameters, the default is ‘heat. If the user is unsure of the best plot type for the current simulation, `plot_type` should not be supplied.

Example unedited plot:



We encourage you to use the graphics toolbox creatively. By using Matlab's native graphing tools GUI, the user can edit generated complex plots without ever having to write complex code.

Each Matlab plot is made up of several parts, called objects. Each datum is graphed on a specific *axes*, inside a specific *figure*. Each object has properties, and an identifying object handle. The axes object, for example, has tick marks, grids, font sizes, and others as properties. The figure has colormap, position, toolbar, and others as properties. The objects common to all plots you will create are: ***Figure, Axes, xlabel, ylabel,*** and ***zlabel*** (note that ***zlabel*** is specific to 3d graphics). Each data object plotted (like a line) also has its own properties, which we will discuss shortly. Any property can be changed using the **set_options()** method.

set_options() (method)

Syntax:

```
gr.set_options( options );
```

Where options is an nx2 cell array with two columns: the object to change (e.g., 'Axes') and the new property value (e.g., changing the 'FontSize'). The property modifications (second column) must also be in a cell array. Each row specifies a new object to modify. Many examples are given throughout this section.

Figure (property)

Figure stores the handle for the generated figure object. Any figure property can be changed using the ***set_options()*** method. These properties are set to default specifications any time a new figure is generated. A list of figure properties can be found here.

Use:

```
>> figOptions = {'Figure', {'Name', 'Example Fig'}}; %rename figure
>> gr.set_options(figOptions); %execute function
```

Axes (property)

Axes stores the handle for the axes contained in ***Figure***. As with ***Figure***, any axes property can be changed using the ***set_options()*** method and similar input. A list of axes properties can be found here.

```
>> axOptions = {'Axes', {'FontSize', 22}}; %change fontsize of Axes
>> gr.set_options(axOptions); %execute function
```

xlabel, ylabel, and zlabel (properties)

These objects are responsible for labeling the respective axes. The user is able to modify font size and type, as well as the displayed text itself. Note that the ***zlabel*** only becomes available when plotting in three dimensions (displaying data using a z-axes). A list of their properties can be found here.

```
>> %add xlabel and change font size
>> xlblOptions = {'xlabel', {'String', 'Example Label', 'FontSize', 24}};
>> gr.set_options(xlblOptions); %execute function
```

As noted, the user can modify multiple objects at once by creating an nx2 cell matrix. Each row contains the object, and the properties to modify*:*

```
Options = { ...
         'Figure', { 'Name', 'Figure 1' } %Change figure name
         'Axes'  , { 'XLim', [-1,1] } %Change x limits
         'xlabel', { 'FontSize', 22, 'FontName', 'Arial' } %label axes
         };
gr.set_options( Options ); %Set new figure properties
```

Note that subsequent calls to set_options() do not reset properties modified on previous calls.

Recall that the plot command is the over-arching plotting tool for the graphics toolbox. It is the smartest of the methods –it is able to choose the best plot for your data (**randomsamples**) and plot it with reasonable results. But for more direct control over plots, the user should use the lower-level graphing tools.

4.1 Functions for creating specific plots

Because output from the *IRM* can be displayed in several different formats, we have provided the user with the most useful plots available in Matlab to graphically represent the simulation data. The user should note that at minimum, all plot types take in some form of x- and y-input, as well as some additional (optional) arguments. These additional arguments vary based on the plot type: for example, all three-dimensional plots require a z-input. Similarly, line plots can, if requested, display the error measure above and below mean values.

plot_scatter() (method)

Syntax:

```
gr.plot_scatter ( x, y );
gr.plot_scatter ( x, y, options );
```

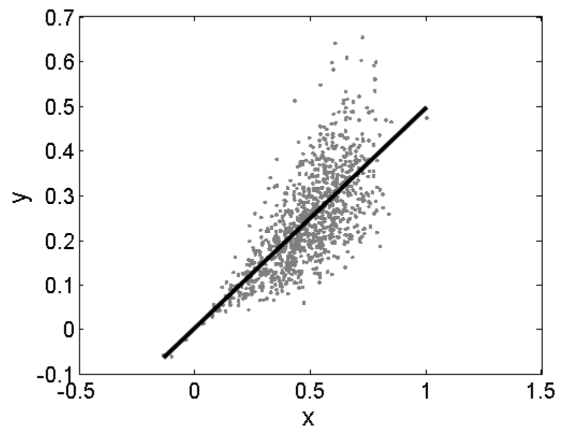
Where x and y are numeric vectors of equal length and (optionally) options is a cell vector of properties to modify when drawing the scatterplot.

If the user is interested in viewing how two measures covary, s/he can view simulation data in a scatterplot. There are two mandatory inputs: an x-vector (specifying each point's x-value) and a y-vector (specifying each y-value). Thus, each point will have x- and y-coordinates (this requires the x- and y-vectors to be the same size). Measures to plot can be any combination of the five input variables or any measure that boils down to them. The user may also enter property-modifying options in cell format.

Arguments to send in can be prepared by using the ***compute()*** function. This can also be done for both target measures at once, as long as the results are stored in separate x- and y-vectors (shown in the example). Note that this and all subsequent plotting functions require numeric values stored in matrix format (not cell arrays) – it may in some cases be necessary to convert stored data from cell to numeric format using Matlab's cell2mat command.

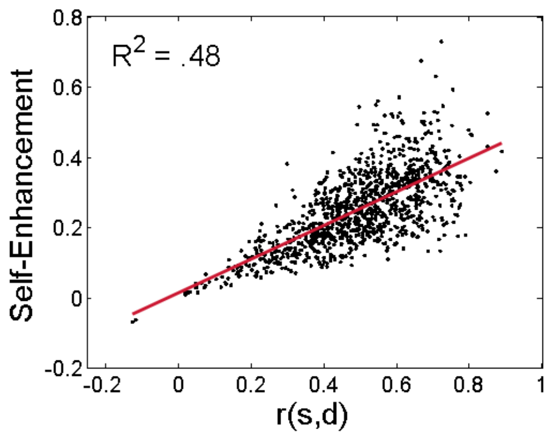
Use:

```
>> %% Scatterplot Example
>>
>> sim = simulate(); %create new simulation object
>> sim.npoints = 1000;
>> sim.get_samples(); %generate samples
>>
>> measure = '[r(s,d), Self-Enhancement]'; %define measures to plot
>>
>> xy = cell2mat( sim.compute( measure ) ); %convert target data to matrix
>> x = xy(:,1); %define r(s,d) as x
>> y = xy(:,2); %define Self-Enhancement as y
>>
>> gr = graph(); %create new graphics object
>>
>> gr.plot_scatter( x, y ); %create scatterplot of target measures
```



This is the default display for a scatterplot. We can modify properties of this figure or its containing objects to make it look differently.

```
>> %% Modifying Scatterplot Properties
>>
>> options = {
    'figure' , {'Name', 'Self-Positivity and Self-Enhancement'}
    'axes'   , {'xlim', [-.25 1], 'ylim', [-.2 .8]}
    'line'   , {'Color', [.8 .1 .2], 'LineWidth', 2} %change line color
    'scatter', {'Color', 'k'} %change scatter points color
    'xlabel' , {'FontSize', 22, 'FontName', 'Arial', 'String', 'r(s,d)'}
    'ylabel' , {'FontSize', 22, 'FontName', 'Arial', 'String', 'Self-Enhancement'}
    'text'   , { -.18,.7, 'R^2 = .48', 'FontSize', 20} %add fit value
    };
>> gr.plot_scatter( x, y, options );
```



plot_histogram() (method)

Syntax:

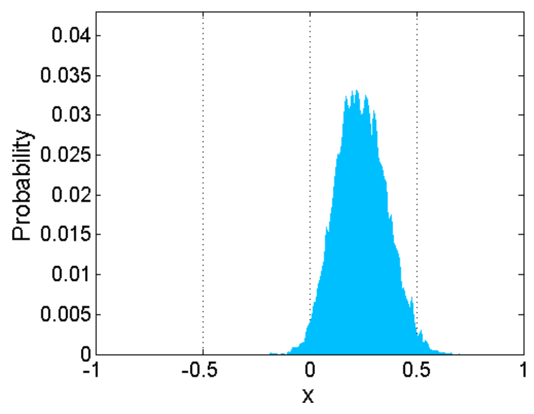
```
gr.plot_histogram( xfill, yfill )
gr.plot_histogram( [], [], xline, yline )
gr.plot_histogram( xfill, yfill, xline, yline )
gr.plot_histogram( xfill, yfill, xline, yline, options )
```

Where `xfill` or `xline` is a vector of bins, and `yfill` or `yline` is a vector of frequencies or probabilities. To draw a filled histogram, send in `xfill` and `yfill`. To view a line histogram, send in xline and yline. To view both, send in all four arguments (note the x and y values for the filled and line histograms can be different). `options` is an optional input composed of a cell vector of properties to modify when drawing the histogram (see **set_options()** above).

To plot a histogram, the user must first prepare the samples (e.g., call **compute()** or **compute_correlation()**). To do this, the user should define which measure to plot, specify a range and number of bins into which measure values can fall, and to calculate the number of values in each bin (done in the example below using the Matlab's native *hist* command). The ***plot_histogram()*** method will plot the distribution specified by input y over x. The output from the *hist* command is a frequency distribution of counts. If you wish to plot the probability distribution, you must first scale the frequency distribution by its sum (shown in the example below).

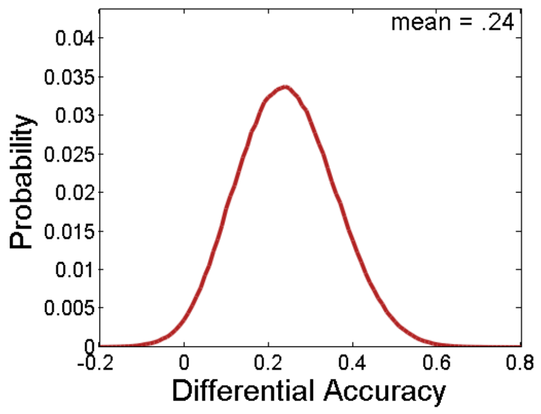
Use:

```
>> %% Histogram Example
>>
>> sim = simulate(); %create new simulation object
>> sim.npoints = 1e6; %increase number of points
>> sim.get_samples(); %generate samples
>>
>> measure = 'Differential Accuracy'; %define measure to plot
>>
>> bins = '-1:.01:1'; %specify bin size and range
>> func = sprintf( 'hist( %s, %s )', measure, bins ); %combine into 'hist' function
>>
>> x = eval(bins); %create vector of bins
>> y = cell2mat( sim.compute( func ) ); %compute measure and convert to matrix
>> y = y/sum(y); %convert frequency to proportion
>>
>> gr = graph(); %create new graphics object
>>
>> gr.plot_histogram( x, y, [], [] ); %create filled histogram
```



This is the default display for a filled histogram. See below for an example of how to modify several relevant properties and redraw.

```
>> %% Modifying Histogram Properties
>>
>> options = {
  'figure', {'Name', 'Probabilities of Differential Accuracy values'}
  'axes'   , {'xlim', [-.2 .8], 'xgrid', 'off', }
  'xlabel', {'FontSize', 22, 'String', 'Differential Accuracy'}
  'ylabel', {'FontSize', 22, 'String', 'Probability'}
  'line',    {'Linewidth', 3}
  'text',    {.49, .042, 'mean = .24', 'FontSize', 18}
  };
>> gr.plot_histogram( [],[], x, y, options ); %redraw histogram
```



plot_line() (method)

Syntax:

```
gr.plot_line( x, y )
gr.plot_line( x, y, e )
gr.plot_line( x, y, options )
gr.plot_line( x, y, e, options )
```

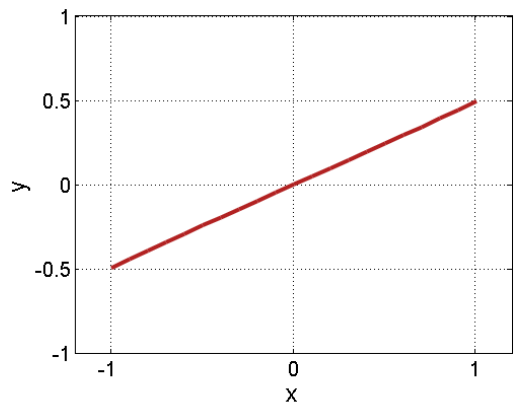
Where `x` is a vector of input values and `y` is a vector of derived values. `x` and `y` must be the same size. `options` is an optional input composed of a cell vector of properties to modify when drawing the line plot.

Line plots become available once one or more input parameters have been varied. This type of plot allows the user to display any baseline or user-defined measure values over steps of a varied input parameter. We have found that, most often, the line plotted is the mean of some measure (like Self-Enhancement) over the varied input parameter.

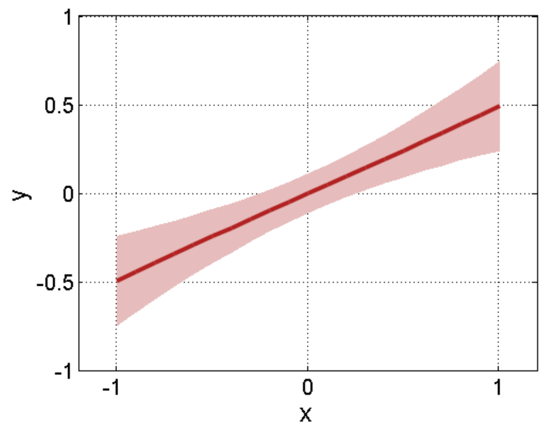
Unique to the line plot is the ability to display an error measure (typically the standard deviation) extending above and below the mean plotted value at each step of the varied input parameter. To do this, the user must compute and send in a vector containing the standard deviation of the target measure at each level of the varied parameter as argument `e`, following the required inputs *x* and *y*.

Use:

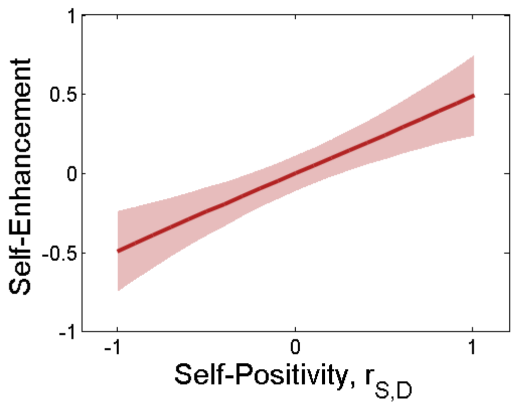
```
>> %% Line Plot Example
>>
>> sim = simulate(); %create new simulation object
>> input = { 'r(s,d)', -1 : .1 : 1, 26 }; %vary r(s,d)
>> sim.modify_input( input ); %modify input matrix
>>
>> sim.get_samples(); %generate samples
>>
>> measure = 'mean(Ingroup Favoritism)'; %define target measure
>> y = cell2mat( sim.compute( measure ) ); %compute measure
>> x = -1 : .1 : 1; %define x-axis values for plotting
>>
>> gr = graph(); %create new graphics object
>>
>> gr.plot_line( x, y ); %draw line plot
```



```
>> %% Adding Standard Deviation to Plot
>>
>> stDev = 'std(Ingroup Favoritism)'; %define standard deviation
>> stDev = cell2mat( sim.compute( stDev ) ); %compute measure
>>
>> gr.plot_line( x, y, stDev ); %redraw plot with standard deviation
```



```
>> %% Modifying Line Plot Properties
>>
>> options = ...
    {
    'figure', {'Name', 'Ingroup Favoritism varying with r(s,d)'}
    'axes'  , {'YLim', [-1 1], 'ygrid', 'off', 'xgrid', 'off' }
    'line'  , {'LineWidth', 3 }
    'xlabel', {'FontSize', 22, 'FontName', 'Arial', 'String', 'Self-Positivity, r_{S,D}'}
    'ylabel', {'FontSize', 22, 'FontName', 'Arial', 'String', 'Self-Enhancement'}
    };
>> gr.plot_line( x, y, stDev, options ); %redraw plot with new properties
```



We encourage you to use the graphics toolbox creatively. For example, if you want to look at the deviation of a histogram over several iterations of a simulation, use the **plot_line()** method instead of the **plot_histogram()** method to plot the histogram with an error measure.

4.2 3-dimensional plots

The three functions described below generate three-dimensional plots. 3d plots only become available once two input parameters have been varied. In all cases, varied input parameters are displayed on the x- and y-axes. The user is able to plot a baseline input measure, user-defined measure, or the correlation between any combination of these on the z-axis.

A 3d plot requires the user to send in a vector of steps for the x-axis, a vector of steps for the y-axis, and a matrix of data points to display for the z-axis. For the x- and y-axes, the user can either specify a range and number of steps or simply send in the steps specified when varying an input parameter. The z-axis must be sent in as a matrix of numeric values, which should be computed using the ***compute()*** function from the simulation object (or if the user wishes to plot the correlation between two measures, using the ***compute_correlation()*** function).

In order to highlight major properties of each type of plot, the provided example follows a single simulation plotted according to the three different 3d plot types. In this simulation r(s,d) is varied from -1 to 1 in 20 steps, r(s,o) is varied from -0.5 to 0.5 in 20 steps, and the measure to plot on the z-axis is the mean level of ingroup favoritism at each step of the simulation. For each of the three plot types, the user is walked through the initial (default) plot, how to modify universal properties, and properties/functions specific to each plot type.

Simulation example for all three plot types:

```
>> sim = simulate(); %create new simulation object
>>
>> x = linspace( -1, 1, 20 ); %specify range and number of steps
>> y = linspace( -0.5, 0.5, 20 ); %specify range and number of steps
>> sim.modify_input( {'r(s,d)', x, 26; 'r(s,o)', y, 26} ); %vary two inputs
>>
>> sim.get_samples(); %get samples
>>
>> measure = 'mean(Ingroup Favoritism)'; %define target measure
>>
>> z = cell2mat( sim.compute( measure )); %compute target measure for z-axis
```

plot_heat() (method)

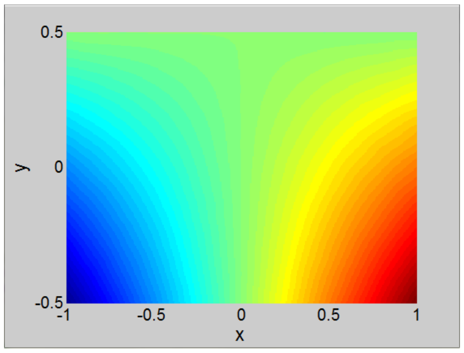
Syntax:

```
gr.plot_heat( x, y, z )
gr.plot_heat( x, y, z, options )
```

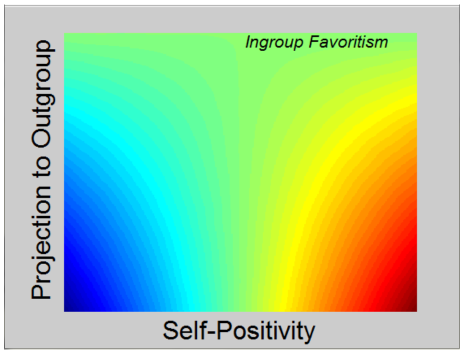
Where `x` and `y` are each a vector of input values and `z` is a matrix of derived values. `options` is an optional input composed of a cell vector of properties to modify when drawing the heatmap.

The heatmap displays a two-dimensional figure using colors to represent variation in values on a z-axis. This plot type allows the user to observe the general pattern of a target measure over two varying input measures.

```
>> %Plot heatmap
>> gr = graph(); %create new graphics object
>> gr.plot_heat( x, y, z ); %draw heat map; plot z
```

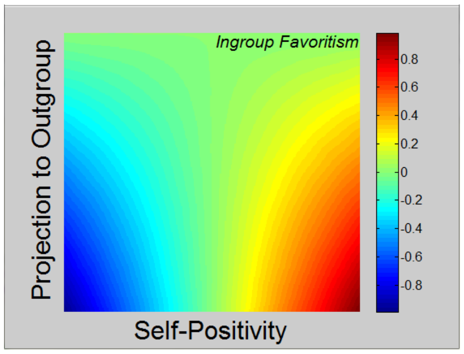


```
>> %Modify figure and axes properties
>> options = {
    'figure' , {'Name', 'Ingroup Favoritism with r(s,d) and r(s,o)'}
    'axes'   , {'xticklabel', [], 'yticklabel', []}
    'xlabel' , {'String', 'Self-Positivity' , 'FontSize', 24, 'FontName', 'Arial'}
    'ylabel' , {'String', 'Projection to Outgroup', 'FontSize', 24, 'FontName', 'Arial'}
    'text'   , {.03,.465, 'Ingroup Favoritism', 'FontAngle', 'italic', 'FontSize', 16}
    };
>> gr.plot_heat( x, y, z, options ); %redraw heatmap
```



It is often useful to the user to add a colorbar to the figure in order to provide reference for specific colors and values – this can be accomplished using Matlab’s *colorbar* command. And you can always change the colormap to something else as well.

```
>> %Execute commands specific to 3d plots
>>
>> colorbar( 'Location','EastOutside' ); %add colorbar
>> set(colorbar,'fontsize', 12); %change colorbar font size
>> colormap('jet'); %change colormap
```



plot_contour() (method)

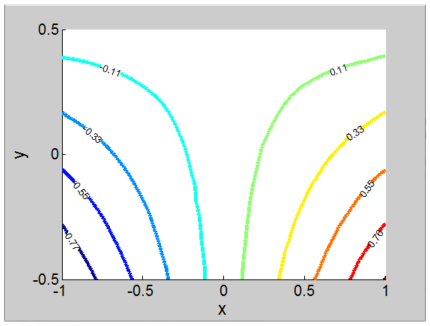
Syntax:

```
gr.plot_contour( x, y, z )
gr.plot_contour( x, y, z, options )
```

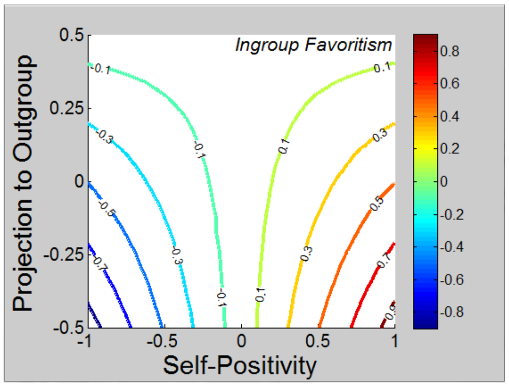
Where `x` and `y` are each a vector of input values and `z` is a matrix of derived values. `options` is an optional input composed of a cell vector of properties to modify when drawing the contour plot.

The contour plot displays a two-dimensional figure using different colors to represent variation in values on the z-axis. Instead of filling the entire figure (like the heatmap), this plot type instead displays discrete lines along a series of contours so that the user can observe the pattern of a particular target value over varying input parameters.

```
>> %% Contour Plot Example
>>
>> gr = graph(); %create new graphics object
>> gr.plot_contour( x, y, z ); %draw contour plot; plot z
```



```
>> %% Modifying Contour Plot Properties
>>
>> options = {
    'figure' , {'Name', 'Ingroup Favoritism with r(s,d) and r(s,o)'}
    'axes'   , {'ytick', linspace(min(y), max(y), 5)} %set tick marks on y axis
    'xlabel' , {'String', 'Self-Positivity' , 'FontSize', 22, 'FontName', 'Arial'}
    'ylabel' , {'String', 'Projection to Outgroup', 'FontSize', 22, 'FontName', 'Arial'}
    'image'  , {'LabelSpacing', 175, 'LevelList', -.9:.2:.9, 'LineWidth', 3,}
    'text'   , {-.04,.465,'Ingroup Favoritism', 'FontAngle', 'italic', 'FontSize', 16}
    };
>> gr.plot_contour( x, y, z, options ); %redraw contour plot
>>
>> colorbar( 'Location','EastOutside' ); %add colorbar
>> set(colorbar,'fontsize', 12); %change colorbar font size
>> colormap('jet'); %change colormap
```



plot_surface() (method)

Syntax:

```
gr.plot_surface( x, y, z )
gr.plot_surface( x, y, z, options )
```

Where `x` and `y` are each a vector of input values and `z` is a matrix of derived values. `options` is an optional input composed of a cell vector of properties to modify when drawing the surface plot.

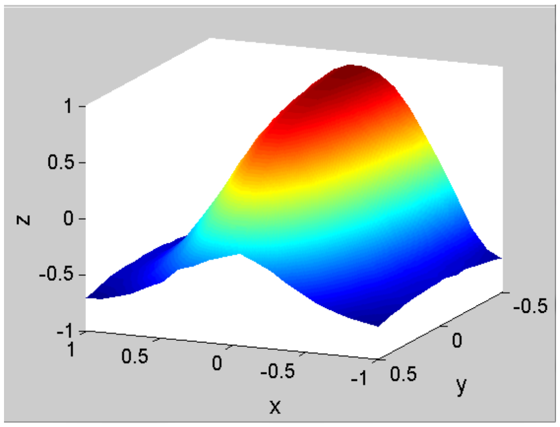
The surface plot displays the third dimension of data by allowing the user to rotate the axes within a figure. The target measure is plotted according to a color/value pairing (as in the heatmap and contour plot). Unique to the surface plot is the addition of a rotation component that allows the user to rotate the figure in order to view a 3-d representation in 2-d space.

Note that the surface plot will look identical to the heatmap, because its default view is from above. In order to view the surface in three dimensions, the user must toggle the ‘Rotate 3D’ option in the plot toolbar before clicking and dragging the axes itself.

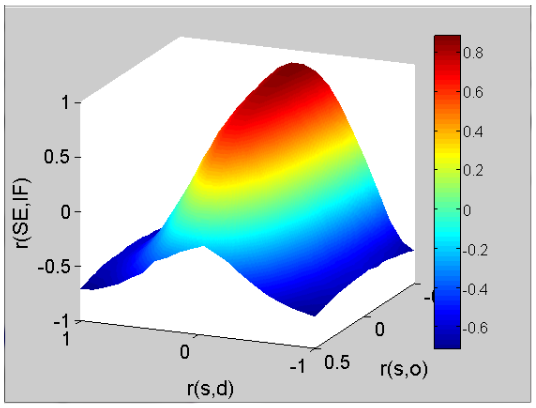
For the example, we have added a new measure in order to make the best use of a surface plot: the correlation between Ingroup Favoritism and Self-Enhancement.

```
>> %% Surface Plot Example
>> gr = graph(); %create new graphics object
>>
>> measure = 'corrcoef([Ingroup Favoritism, Self-Enhancement])'; %define target measure
>> z = cell2mat( sim.compute( measure )); %compute target measure for z-axis
>>
>> gr.plot_surface( x, y, z ); %draw surface plot; plot z
```

(click the 'Rotate 3D' button to activate 3D rotation)



```
>> options = {
     'figure' , {'Name', 'r(SE,IF) with r(s,d) and r(s,o)'}
     'xlabel' , {'String', 'r(s,d)' , 'FontSize', 16, 'FontName', 'Arial'}
     'ylabel' , {'String', 'r(s,o)', 'FontSize', 16, 'FontName', 'Arial'}
     'zlabel' , {'String', 'r(SE,IF)', 'FontSize', 16, 'FontName', 'Arial'}
     };
>> gr.plot_surface( x, y, z, options ); %redraw heatmap
>>
>> colorbar( 'Location','EastOutside' ); %add colorbar
>> set(colorbar,'fontsize', 12); %change colorbar font size
```



4.3 Tables of methods and properties.

Table 2: Properties and methods of the graphics object

Item	**Function**	**Implementation**
Methods		
graph	Creates a new graphics object, 'gr'	*gr = graph();*
plot	Chooses and returns a plot based on data entered	*plot();*
plot_scatter	Plots entered data in a scatterplot	*plot_scatter()*
plot_histogram	Plots entered data in a histogram	*plot_histogram()*
plot_line	Plots entered data in a line plot	*plot_line ()*
plot_heat	Plots entered data in a heatmap	*plot_heat()*
plot_contour	Plots entered data in a contour plot	*plot_contour()*
plot_surface	Plots entered data in a surface plot	*plot_surface()*
set_options	Sets current plot options	*set_options()*
Properties		
Figure	Modify figure properties	*see set_options()*
Axes	Change axes limits, ticks, tick labels, etc	*see set_options()*
Xlabel	Change x label text, font, and font size	*see set_options()*
Ylabel	Change y label text, font, and font size	*see set_options()*
Zlabel	Change z label text, font, and font size	*see set_options()*
Scatter	Change scatter point color, size, marker, etc	*see set_options()*
Fill	Change fill color, border line, etc	*see set_options()*
Line	Change line width, color, style, etc	*see set_options()*
Text	Add string of text to figure	*see set_options()*

5.0 Interface

The interface object is responsible for assigning callbacks to buttons, creating and storing embedded figures, and displaying simulations as they are computed. Because the code associated with the interface is read-only, callbacks and functions are not detailed. For the interested user, however, a summary table containing the interface methods can be referenced below (Table 3).

5.1 Tables of methods

Item	Function	Implementation
click()	Displays currently selected axes on large axes	
add_measure()	Opens a dialogue box to create a new measure	
edit_measure()	Edits a current user-defined measure	
remove_measure()	Deletes a current user-defined measure	
add_plot()	Queues a new measure & plot combination	
remove_plot()	Removes a current measure & plot combination	
plot_selected()	Plots the selected measure & plot combination	
plot_all()	Plots all queued measure & plot combinations	
modify_input()	Allows the user to modify the input matrix	
file_menu()	Executes options in the file menu	
options_menu()	Executes options in the options menu	
help_menu()	Executes options in the help menu	
create_ui_graphics()	Draws and redraws interface axes	
message_center()	Updates message center display	
populate_lists()	Fills menus and lists	
update_axes()	Sets properties of displayed axes	

6.0 Example Code

Code used to generate figures 1-4 in the paper:

```
function create_figures(fig)

switch fig
    case 1
        sim = simulate();

        %% Panel a

        x = 0 : .1 : 1;
        input = {
            'r(s,d)', x             , 26
            'r(s,i)', [0.2 0.5 0.8], 26};
        sim.modify_input( input );
        sim.npoints = 1e7;

        sim.get_samples();

        y = cell2mat( sim.compute( 'mean(Self-Enhancement)' ) )';

        gr = graph();

        colororder = repmat( [0.2 0.5 0.8]', [1,3] );
        options = ...
            {
            'figure', {'Name', 'Figure 1a'}
            'axes'    ,  {'YLim',  [0.0,  0.8],  'ColorOrder',  colororder,  'XGrid',
'off',...
            'YGrid', 'off', 'Xlim', [0 1], 'XTick', (0:.2:1) }
            'line'  , {'LineWidth', 4 }
            'xlabel',  {'FontSize',  22,  'FontName',  'Arial',  'String',  'Self-
Positivity, r_{S,D}'}
            'ylabel',  {'FontSize',  22,  'FontName',  'Arial',  'String',  'Self-
Enhancement'}
            'legend',   {'r_{S,I}   =   0.20','r_{S,I}   =   0.50','r_{S,I}   =
0.80','Location','NorthWest'}
            };

        gr.plot_line( x, y, options );

        %% Panel b
        input = {
            'r(s,d)', [0.2 0.5 0.8], 26
            'r(s,i)', 0 : .1 : 1   , 26};
        sim.modify_input( input );
        sim.npoints = 9e6;

        sim.get_samples();

        x = (0 : .1 : 1);
        y = cell2mat( sim.compute( 'mean(Self-Enhancement)' ) )';
        gr = graph();

        colororder = repmat( [0.2 0.5 0.8]', [1,3] );
        options = ...
            {
            'figure', {'Name', 'Figure 1a'}
            'axes'    ,  {'YLim',  [0.0,  0.8],  'ColorOrder',  colororder,  'XGrid',
'off',...
            'YGrid', 'off', 'Xlim', [0 1], 'XTick', (0:.2:1) }
            'line'  , {'LineWidth', 4 }
            'xlabel',  {'FontSize', 22, 'FontName', 'Arial', 'String', 'Projection
to Ingroup, r_{S,I}'}
            'ylabel',   {'FontSize',   22,   'FontName',   'Arial',   'String',   'Self-
Enhancement'}
            'legend',    {'r_{S,I}    =    0.20','r_{S,I}    =    0.50','r_{S,I}    =
0.80','Location','NorthEast'}
            };

        gr.plot_line( x, y, options );

    case 2
        sim = simulate();
        sim.npoints = 9e6;
        sim.get_samples();

        bins = '-1:.01:1';
        measures = '[r(s,o), r(s,i)]';

        func = sprintf( 'hist( %s, %s )', measures, bins );

        x = eval(bins);
        y = cell2mat( sim.compute( func ) );
        y = bsxfun( @rdivide, y, sum(y) );

        %

        colororder = repmat( [0.2 0.5 0.8]', [1,3] );
        options = ...
            {
            'figure', {'Name', 'Figure 2a', }
            'axes'   , {'ColorOrder', colororder, 'XTick', (-1:.5:1), 'XGrid', 'on',
'YGrid', 'off',...
            'XLim',   [-1   1],   'YLim',   [0   .035],   'YTick',   linspace(0,.035,4),
'YTickLabel', [], 'XTickLabel', [] }
            'line'  , {'LineWidth', 5, 'LineSmoothing', 'on' }
            'xlabel', {'FontSize', 22, 'FontName', 'Arial', 'String', []}
            'ylabel',    {'FontSize',    22,    'FontName',    'Arial',    'String',
'Probability'}
            'legend',   {'Projection   to   Outgroup',   'Projection   to   Ingroup',
'Location', 'NorthWest'}
            };

        gr = graph();
        gr.plot_line( x, y, options );
        set(gca, 'FontSize', 14)

        %% Panel b

        measures = '[r(o,d), r(i,o), r(i,d)]';

        func = sprintf( 'hist( %s, %s )', measures, bins );

        x = eval(bins);
        y = cell2mat( sim.compute( func ) );
        y = bsxfun( @rdivide, y, sum(y) );

        %
        colororder = repmat( [0.2 0.5 0.8]', [1,3] );

        options = ...
            {
            'figure', {'Name', 'Figure 2a', }
            'axes'   , {'ColorOrder', colororder, 'XTick', (-1:.5:1), 'XGrid', 'on',
'YGrid', 'off',...
            'XLim',   [-1   1],   'YLim',   [0   .065],   'YTick',   linspace(0,.065,4),
'YTickLabel', [], 'XTickLabel', [] }
            'line'   , {'LineWidth', 5, 'LineSmoothing', 'on' }
            'xlabel', {'FontSize', 22, 'FontName', 'Arial', 'String', []}
            'ylabel',    {'FontSize',    22,    'FontName',    'Arial',    'String',
'Probability'}
            'legend',  {'Outgroup  Positivity,  Intergroup  Accentuation',  'Ingroup
Positivity', 'Location', 'NorthWest'}
            };

        gr = graph();
        gr.plot_line( x, y, options );
        set(gca, 'FontSize', 14);

        %%
        measures = '[Ingroup Favoritism, Self-Enhancement, Differential Accuracy]';

        func = sprintf( 'hist( %s, %s )', measures, bins );

        x = eval(bins);
        y = cell2mat( sim.compute( func ) );
        y = bsxfun( @rdivide, y, sum(y) );

        %
        colororder = repmat( [0.2 0.5 0.8]', [1,3] );

        options = ...
            {
            'figure', {'Name', 'Figure 2a', }
            'axes'   , {'ColorOrder', colororder, 'XTick', (-1:.5:1), 'XGrid', 'on',
'YGrid', 'off',...
            'XLim',   [-1   1],   'YLim',   [0   .04],   'YTick',   linspace(0,.04,4),
'YTickLabel', [] }
            'line'   , {'LineWidth', 3, 'LineSmoothing', 'on' }
            'xlabel', {'FontSize', 22, 'FontName', 'Arial', 'String', 'Correlation
Coefficient'}
            'ylabel',   {'FontSize',    22,    'FontName',    'Arial',    'String',
'Probability'}
            'legend',   {'Ingroup   Favoritism',   'Self-Enhancement','Differential
Accuracy', 'Location', 'NorthWest'}
            };

        gr = graph();
        gr.plot_line( x, y, options );
        set(gca, 'FontSize', 14);

    case 3
        sim = simulate();
        %% Panel a

        sim.npoints = 5e5;
        x = linspace( 0, .999, 20 );
        sim.modify_input( {'r(s,d)', x, 26} );

        measure   =   'corrcoef([r(i,o),   Self-Enhancement,   Ingroup   Favoritism,
Differential Accuracy])';

        sim.get_samples();
        %
        y = cell2mat(sim.compute( measure ));

        % Don't plot the zero mean ones...
        mu = nanmean( y );
        f  = abs(mu) < 0.005; % reasonable threshold.
        y(:,f) = nan;

        colororder = repmat( linspace(0,.5,2)', [1,3] );
        linestyleorder = {'-', ':', '--' };
        options = ...
            {
            'figure', {'Name', 'Figure 3a', 'Color', 'white'}
            'axes'   , {'ColorOrder', colororder, 'LineStyleOrder', linestyleorder,
'XGrid',  'on',  'YGrid',  'off',  'XTick',0:.25:1,  'Xlim',  [0  1],  'YTick',  -
0.75:.25:0.75, 'YLim', [-0.76, 0.76]}
            'line'   , {'LineWidth', 5 }
            'xlabel',  {'FontSize',  22,  'FontName',  'Arial',  'String',  'Self-
Positivity, r_{S,D}'}
            'ylabel', {'FontSize', 22, 'FontName', 'Arial', 'String', 'Correlation
Coefficient'}
            'legend',   {'r_{IA,SE}*',   'r_{IA,IF}',   'r_{IA,DA}*',   'r_{SE,IF}',
'r_{SE,DA}', 'r_{IF,DA}', 'Location', 'NorthEastOutside'}
            };

        gr = graph();
        gr.plot_line( x, y, options );

        legend('boxon');

        %% Panel b
        sim.modify_input( {'r(s,i)', x, 26; 'r(s,d)', 0.5, 26} );

        measure    =   'corrcoef([r(i,o),   Self-Enhancement,   Ingroup   Favoritism,
Differential Accuracy])';

        sim.get_samples();
        %
        y = cell2mat(sim.compute( measure ));

        % Don't plot the zero mean ones...
        mu = nanmean( y );
        f  = abs(mu) < 0.001; % reasonable threshold.
        y(:,f) = nan;

        colororder = repmat( linspace(0,.5,2)', [1,3] );
        linestyleorder = {'-', ':', '--' };
        options = ...
            {
            'figure', {'Name', 'Figure 3b','Color', 'white'}
            'axes'   , {'ColorOrder', colororder, 'LineStyleOrder', linestyleorder,
'XGrid',   'on',   'YGrid',   'off',   'XTick',0:.25:1,'Xlim',   [0   1],   'YTick',   -
0.75:.25:0.75, 'YLim', [-0.76, 0.76] }
            'line'   , {'LineWidth', 5 }
            'xlabel',  {'FontSize',  22,  'FontName',  'Arial',  'String',  'Projection
to Ingroup, r_{S,I}'}
            'ylabel',  {'FontSize',  22,  'FontName',  'Arial',  'String',  'Correlation
Coefficient'}
            'legend',    {'r_{IA,IF}',    'r_{SE,IF}',    'r_{SE,DA}',    'r_{IF,DA}',
'Location', 'NorthEastOutside'}
            };

        gr = graph();
        gr.plot_line( x, y, options );

        legend('boxon');
        set(legend, 'visible', 'off')

        %% Panel c

        x = linspace( -0.5, 0.5, 20 );
        sim.modify_input( {'r(s,o)', x, 26;'r(s,i)', 0.5, 26; 'r(s,d)', 0.5, 26} );

        measure    =   'corrcoef([r(i,o),   Self-Enhancement,   Ingroup   Favoritism,
Differential Accuracy])';

        sim.get_samples();
        %
        y = cell2mat(sim.compute( measure ));

        colororder = repmat( linspace(0,.5,2)', [1,3] );
        linestyleorder = {'-', ':', '--' };
        options = ...
            {
```

```
            'figure', {'Name', 'Figure 3c','Color', 'white'}
            'axes'   , {'ColorOrder', colororder, 'LineStyleOrder', linestyleorder,
'XGrid', 'on', 'YGrid', 'off', 'XTick',-0.5:.25:0.5, 'Xlim', [-0.5 0.5], 'YTick', -
0.75:.25:0.75, 'YLim', [-0.76, 0.76] }
            'line'  , {'LineWidth', 5 }
            'xlabel', {'FontSize', 22, 'FontName', 'Arial', 'String', 'Projection
to Outgroup, r_{S,O}'}
            'ylabel', {'FontSize', 22, 'FontName', 'Arial', 'String', 'Correlation
Coefficient'}
            'legend',  {'r_{IA,SE}*',  'r_{IA,IF}',  'r_{IA,DA}*',  'r_{SE,IF}',
'r_{SE,DA}', 'r_{IF,DA}', 'Location', 'NorthEastOutside'}
            };

        gr = graph();
        gr.plot_line( x, y, options );

        legend('boxon');
        set(legend, 'visible', 'off')
    case 4
        %% Panel a

        sim = simulate();

        sim.npoints = 1e5;
        x = linspace( 0, 1, 30 );
        y = linspace( 0, 1, 30 );
        sim.modify_input( {'r(s,d)', y, 26; 'r(s,i)', x, 26} );

        measure  =  'corrcoef([Self-Enhancement,  Ingroup  Favoritism,  Differential
Accuracy])';

        sim.get_samples();

        z = (sim.compute( measure ));
        Za = z{1}(1);


        options = ...
            {
            'Figure' , {'Name', 'Figure 1a'}
            'Axes'    , {'Box', 'on', 'XTick', linspace(0,.99,5), 'XTickLabel', [0
.25 .50 .75 1], 'YTick', linspace(0,.99,5), 'YTickLabel', [0 .25 .50 .75 1]}
            'image'   , {'LabelSpacing',1000, 'LevelList', [-0.6, -.4, -.2, 0, .2,
.4, .6, .8], 'LineWidth', 4, 'LineColor', zeros(1,3)}
            'xlabel' , {'String', 'Projection to Ingroup, r_{S,I}'}
            'ylabel' , {'String', 'Self-Positivity, r_{S,D}'}
            'line'   , {'LineSmoothing','on'}
            };

        gr = graph();
        gr.plot_contour( x, y, Za{1}', options );
        %% Panel b

        Zb = z{1}(3);
        options = ...
            {
            'Figure' , {'Name', 'Figure 2b'}
            'Axes'    , {'Box', 'on', 'XTick', linspace(0,.99,5), 'XTickLabel', [0
.25 .50 .75 1], 'YTick', linspace(0,.99,5), 'YTickLabel', [0 .25 .50 .75 1]}
            'image'   , {'LabelSpacing', 1e3, 'LevelList', [ 0.025; 0.1; 0.2; 0.3;
0.4; 0.5; 0.60], 'LineWidth', 4, 'LineColor', zeros(1,3)}
            'xlabel' , {'String', 'Projection to Ingroup, r_{S,I}'}
            'ylabel' , {'String', 'Self-Positivity, r_{S,D}'}
            };

        gr = graph();
        gr.plot_contour( x, y, Zb{1}', options );

        %% Panel c

        x = linspace( 0, 1, 30 );
        y = linspace( -0.5, 0.5, 30 );

        sim.modify_input( {'r(s,d)', 0.5, 26; 'r(s,i)', x, 26; 'r(s,o)', y, 26} );

        measure  =  'corrcoef([Self-Enhancement,  Ingroup  Favoritism,  Differential
Accuracy])';

        sim.get_samples();

        z = (sim.compute( measure ));
        Zc = z{1}(3);

        options = ...
            {
            'Figure' , {'Name', 'Figure 2c'}
            'Axes'    , {'Box', 'on', 'XTick', linspace(0,1,5), 'XTickLabel', [0 .25
.50 .75 1], 'YTick', linspace(-.5,.5,5), 'YTickLabel', linspace(-.5,.5,5)}
            'image'    ,  {'LabelSpacing',  1e3,  'LevelList',  [0.1:0.1:0.5,  0.55],
'LineWidth', 4, 'LineColor', zeros(1,3)}
            'xlabel' , {'String', 'Projection to Ingroup, r_{S,I}'}
            'ylabel' , {'String', 'Projection to Outgroup, r_{S,O}'}
            };

        gr = graph();
        gr.plot_contour( x, y, Zc{1}, options );
    otherwise
        error;
end




end % create_figures
```