CalcPro's Manual

Version 1.3

Zynergy Apps, LLC

Table of Contexts

Table of Figures

Table of Tables

1 Introduction

Welcome to Zynergy Apps' CalcPro. This app can be used to perform simple calculations like a traditional calculator to complex calculations with scripting and plotting capabilities. You can perform computations without any cellular or Wi-Fi connection.

To get started quickly, please review the *Getting Started*, *Quick Tour*, *Demo Tour*, *Hello World Script*, and *Debugging Example* sub-sections. This will give you an overview of the app's capabilities within 30 minutes. The rest of this guide is designed to introduce general app capabilities to take you to the next level of calculating numbers of interest!

1.1 Getting Started

On the *Calculator* tab, enter an expression to calculate in the "Enter expression to calculate" field. For example, click in the field, enter `1 + 1,` and press Return to compute the value of `2`.

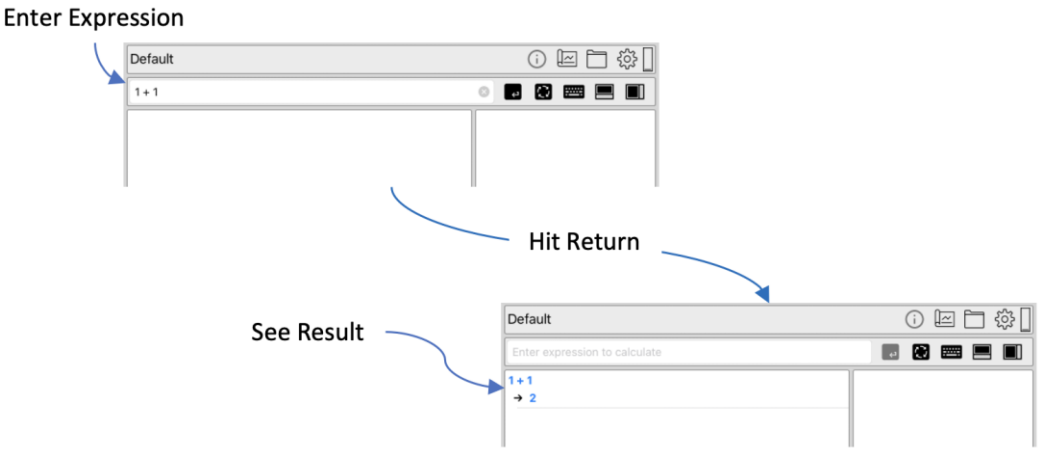

Figure 1 - Calculate 1 + 1

Naturally, you can compute more complicated calculations. For example, say your hourly rate at your place of employment is `$25.48,` and you work `40` hours a week with two weeks of vacation included. What is your annual salary? Enter `25.48 × 40 × 52,` and it will compute `52998.4,` representing your annual salary.

```
25.48 * 40 * 52
→ 52998.4
```

Figure 2 - Calculate Annual Salary

You can beautify the results using the `fmt` function by re-typing in `fmt("$%,7.2f", 25.48 × 40 × 52)` to output `$52,998.40`. Details are in the *Libraries* chapter in the *Convert* sub-section on the `fmt` function formatting options.

```
fmt("$%,7.2f",  25.48 * 40 * 52)
→ $52,998.40
```

Figure 3 - Calculate Annual Salary Alternate

1.2 Quick Tour

Let's review the general parts of the app and its basic functionalities.

1.2.1 Menu Bar

On each tab at the top of the screen, a menu bar is displayed to perform various actions by the app.

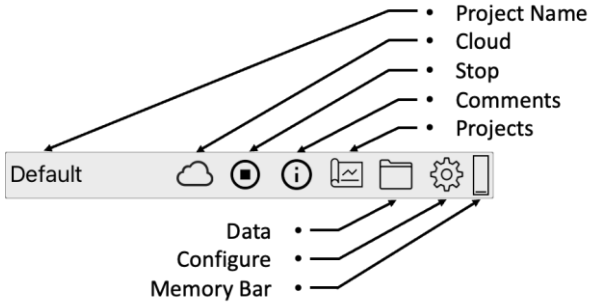

Figure 4 - Menu Bar

- The first part of the menu bar displays the name of the project currently in use.

  ⇒ NOTE: The "Default" project is always available for general-purpose calculations and cannot be renamed or deleted.

- The *Cloud* icon is an indicator the project is on the iCloud. Otherwise, no cloud icon is displayed for local project.

- The *Stop* button allows you to terminate a script currently running. Normally, this button is not visible unless the script has been running longer than 3 seconds.

- The *Comments* button allows you to add comments to your calculation to remind you of the purpose of the calculation. For example, let's add a comment to the previous calculation in the *Getting Started* section.

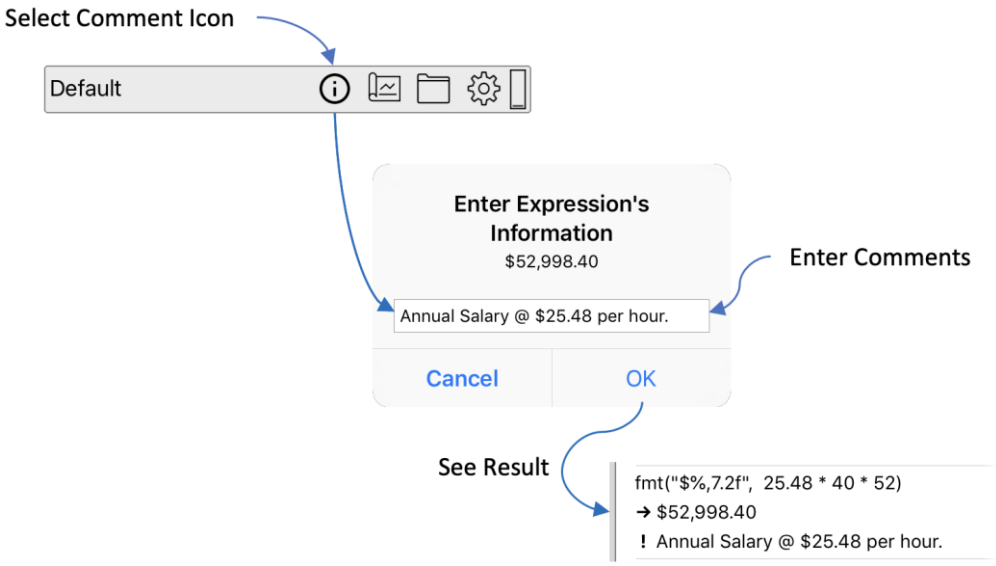

Figure 5 - Adding Calculate Annual Salary Comments

⇒ NOTE: Adding comments is applicable to *Calculator*'s Variable column, and the *Scripts* and *Graphs' Name* columns.

- The *Projects* button allows you to select another project. It also includes creating new projects, renaming projects, deleting, and archiving. You can create new projects to organize your working calculations, scripts, and graphs for various activities.

- The *Data* button allows importing text script files and project archive files, as well as reviewing created dataset files containing a collection of variables from your scripts for reuse. It also includes copying the data onto the clipboard, renaming, and deleting data files. Refer to additional details in the *Libraries* chapter in the *File* section on reading and writing these dataset files.

- The *Configure* button displays a series of dialogs to configure the app's environment. *Configuration* is the initial dialog displayed.

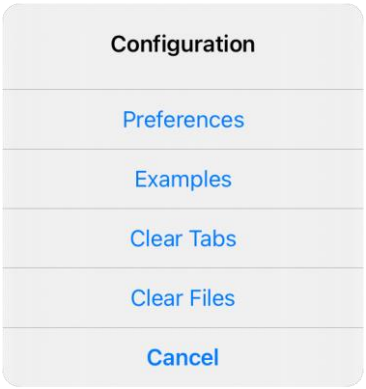

Figure 6 - Configuration Dialog

  - The *Preferences* option displays the app's parameter settings.

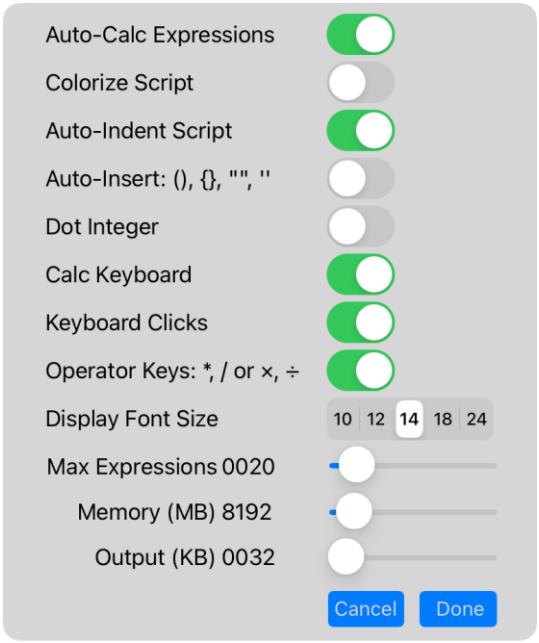

Figure 7 - Configuration's Settings Dialog

- *Auto-Calc Expressions* enables automatically recalculating your expressions containing variables that have changed. A similar mechanism is found in spreadsheet apps; if a cell is changed, it will trigger an update to other cells. Additional details are in the *Expressions* chapter in the *Variables* section.

- *Colorize Script* enables the script's text to be colorized to highlight different parts of the text's syntax; otherwise, the default text color is black.

- *Auto-Indent Script* enables the script's text to be automatically indented for each level of `if`/`else`/`for`'s sub-statements.

- *Auto-Insert: (), [], {}, " ", '* enables automatic text insertion of the matching character when entering an expression to calculate on the *Calculator* tab or the script's text on the *Scripts* tab.

- *Dot Integer* enables integer number results to embed a period character for every third integer digit to allow easy reading of large numbers of millions, billions, and trillions.

- *Calc Keyboard* enables app's custom keyboard panel for the *Calculator* and *Scripts* tab's keyboard entries rather than the Apple's built-in keyboard panel.

- *Keyboard Clicks* enable keyboard clicking sound when using *Calc Keyboard* mode.

- *Operator Keys: *, / or* ×, ÷ switches the classic multiplier and divisor keys for the *Calc Keyboard* panel to `*` and `/` rather than × and ÷ keys.

- *Display Font's Size* changes the text and field's font size between 10, 12, 14, 18, or 24 point-size.

- The *Max Expressions* slider sets the maximum number of expressions displayed on the *Calculator* tab before rolling off the *Expressions* column display. If the setting is set to zero, no expressions will be removed from the column's display.

- The *Memory (MB)* slider sets the maximum amount of memory that can be used in running the scripts.

    - The purpose of this setting is to avoid mistakes in your script requesting too much memory and causing the app either abnormally terminate or taking excessive memory resources from other apps, causing response issues.

    - MB means mega-bytes, which is 1,048,576 bytes (i.e., characters) of memory.

- The *Output (KB)* slider sets the maximum amount of memory that can be used to display on the *Outputs* tab before rolling off oldest message output.

    - The purpose of this setting is to avoid excessive messages being output by the scripts.

    - KB means kilobytes, which is 1,024 bytes (i.e., characters) of memory.

- *Examples* displays a collection of examples to be installed into the current project for demonstration purposes. There are three sets of examples to install: *Basic*, *Standard*, and *Advance*, comprising simple to complex examples. Likewise, you can uninstall these, too.

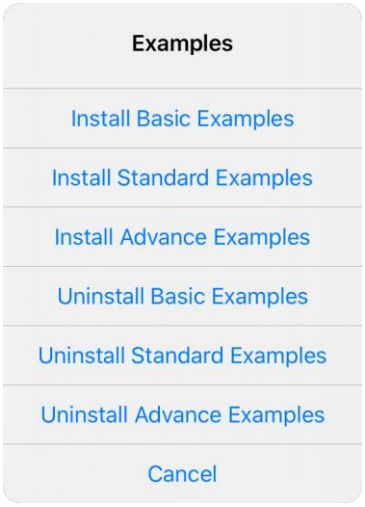

Figure 8 - Configuration's Examples Dialog

- *Clear Tabs* allows clearing the contents on the *Calculator*, *Scripts*, *Graphs*, and *Outputs* tabs. And there is *All* button to clear everything in the current project.

  ⇒ NOTE: The *Globals* are special saved global variables. Refer to the *Variables* sub-section on saving and managing global variables.

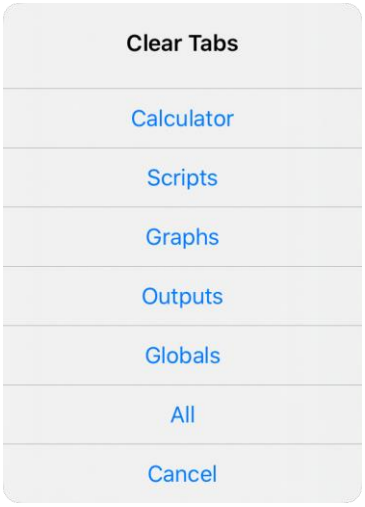

Figure 9 - Configuration's Clear Tabs Dialog

- *Clear Files* allows clearing *Comma Separated Values (CSV)*, *Dataset*, and *Global's* local and cloud data files.

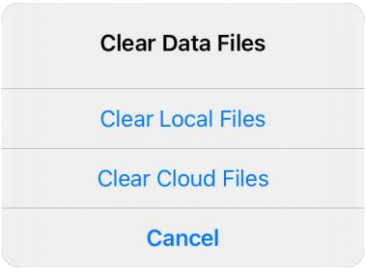

Figure 10 - Configuration's Clear Files Dialog

- The *Memory Bar* displays percentage memory used by the projects and running scripts. The memory bar will turn red if exceeds 80% of the memory settings in the *Configuration Preferences* dialog.

1.2.2 Calculator

The *Calculator* tab has a sub-menu bar and three sections. Section **A** is the *Expressions* column, displaying the results of the entered expressions. Section **B** is the *Variables* column, displaying the variables' assignments from the results of the entered expressions. Section **C** is the *Results* area, displaying output of the expressions' results.

You can resize the *Results* area by selecting the little black button on right side of the screen between sections **B** and **C**. This will allow more output viewing of the calculated results.

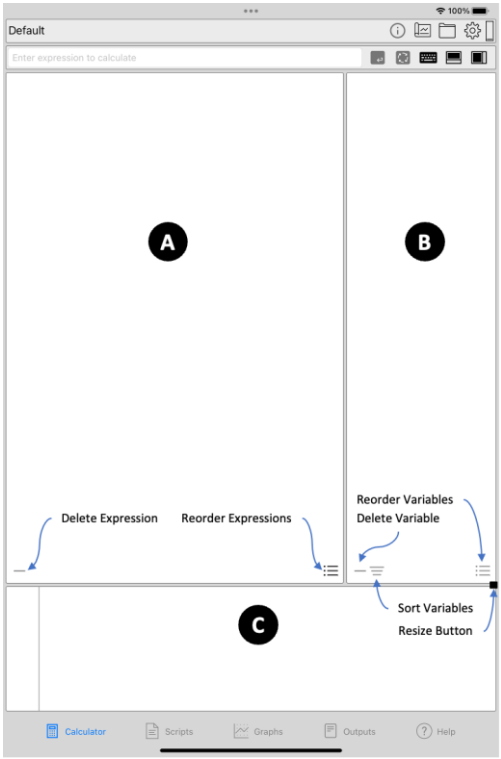

Figure 11 - Calculator Tab

- The first part of the menu bar displays the entry of the expression to calculate in its collection of commands.

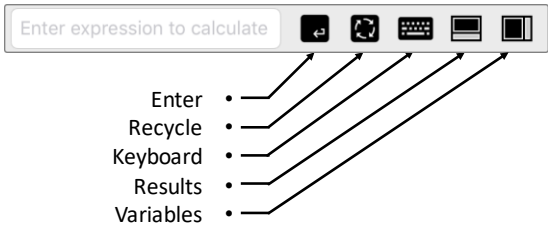

Figure 12 - Calculator Menu Bar

- The *Enter* button calculates the entered expression.
- *Recycle* will re-enter the previous expressions.

    ⇒ NOTE: Only saves the last 100 expression entries during app's session. This is reset whenever the app is restarted.

- *Keyboard* will show or hide the keyboard panel while entering an expression.
- *Results* will show or hide the *Results* area display output.
- *Variables* will show or hide the *Variables* column area.

1.2.2.1 Commands

Command	Description
Ctrl+R	Evaluate expression
Ctrl+C	Clear output results
Ctrl+Option+↓	Select next evaluated expression
Ctrl+Option+↑	Select previous evaluated expression
Ctrl+Option+←	Select top evaluated expression
Ctrl+Option+→	Select bottom evaluated expression
Ctrl+1	Switch to Calculator tab
Ctrl+2	Switch to Scripts tab
Ctrl+3	Switch to Graphs tab
Ctrl+4	Switch to Outputs tab
Ctrl+5	Switch to Help tab
—	The delete button allows removing an entry
⩧	The sort button allows sorting the column
⁝☰	The order button allows reordering the entries

Table 1 - Calculator Commands

1.2.3 Scripts

The *Scripts* tab has a sub-menu bar and two sections. Section **A** is the script text editing area to create and edit script code. Section **B** is the *Names* column, which displays a list of script files.

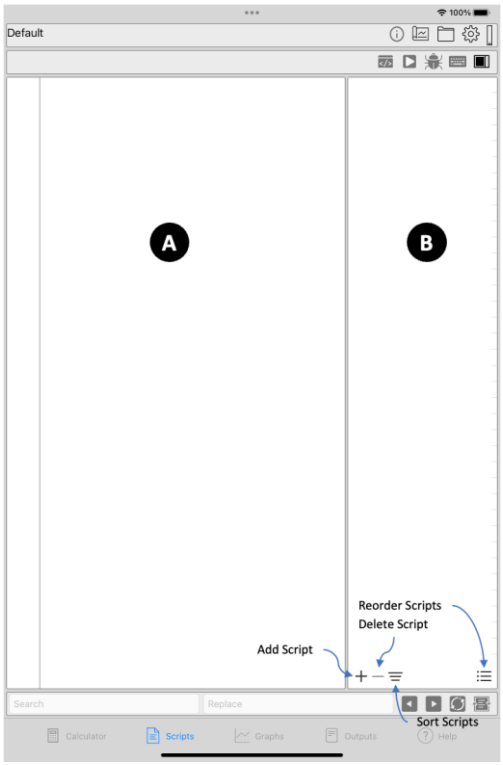

Figure 13 - Scripts Tab

The first part of the menu bar displays the name of the script in its collection of commands.

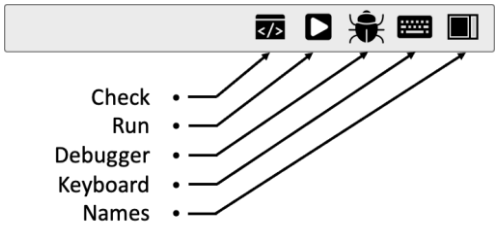

Figure 14 - Scripts Menu Bar

- *Check* will validate that the script's statements are syntactically correct.

    ⇒ NOTE: If an error is encountered within the script or it has been modified and has not been checked for correctness, the button color will be red.

- *Run* will execute your script and automatically add an expression entry on the *Calculator* tab.
- *Debugger* will execute your script in debugging mode by stepping through the script's code.
- *Keyboard* will show or hide the keyboard panel while editing the script's text code.
- *Names* will show or hide the *Names* column area.

The bottom part of the *Scripts* tab displays basic search-and-replace script text controls.

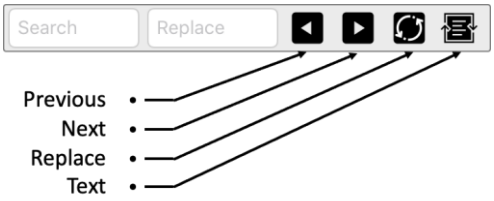

Figure 15 - Scripts Search-and-Replace Bar

- The *Search* text field is used to find matching text.
- The *Replace* text field is used to replace the matching text found.
- *Previous* searches for the previous matching text.

    ⇒ NOTE: If there is no text in the *Search* text field, it will scroll to the previous page of the script's text. If the button is double-tapped, it will scroll to the top of the script's text.

- *Next* searches for the next matching text.

    ⇒ NOTE: If there is no text in the *Search* text field, it will scroll to the next page of the script's text. If the button is double-tapped, it will scroll to the bottom of the script's text.

- *Replace* replaces the matching text.
- *Text* displays a *Text* dialog to change the selected text.

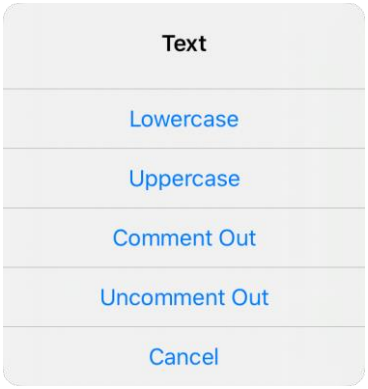

Figure 16 - Scripts Text Dialog

    - *Lowercase* will change the selected text to lowercase.
    - *Uppercase* will change the selected text to uppercase.
    - *Comment Out* will add the # character to comment out selected lines in the script's text.
    - *Uncomment Out* will remove the # character to uncomment out text from the selected script lines.

1.2.3.1 Commands

Command	Description
Ctrl+R	Run script
Ctrl+.	Stop running script
Ctrl+C	Validate script
Ctrl+S	Search for next matching text
Ctrl+Shift+S	Search for previous matching text
Ctrl+V	Replace matching text
Ctrl+A	Move insertion point to beginning of the line*
Ctrl+E	Move insertion point to the end of the line*
Ctrl+F	Move insertion point to next character*
Ctrl+B	Move insertion point to previous character*
Ctrl+T	Swap characters at insertion point*
Ctrl+N	Move insertion point to the next line*
Ctrl+P	Move insertion point to the previous line*
Ctrl+J	Insert a new line at insertion point*
Ctrl+D	Delete previous character at insertion point*
Ctrl+K	Delete characters to the end of line from insertion*
Ctrl+I	Insert a tab space at insertion point*
Ctrl+Shift+A	Select text from beginning of the line from insertion*
Ctrl+Shift+E	Select text from end of the line from insertion
Ctrl+Shift+F	Select next character from the end of selected text*
Ctrl+Shift+B	Select previous character from beginning of insertion*
Ctrl+Shift+N	Select next line from the end of selected text*
Ctrl+Shift+P	Select previous line from beginning of insertion*
Ctrl+Shift+↓	Scroll to next page
Ctrl+Shift+↑	Scroll to previous page
Ctrl+Shift+←	Scroll to the top of the script
Ctrl+Shift+→	Scroll to the bottom of the script
Ctrl+Option+↓	Select next script
Ctrl+Option+↑	Select previous script
Ctrl+Option+←	Select top script
Ctrl+Option+→	Select bottom script
Ctrl+1	Switch to Calculator tab
Ctrl+2	Switch to Scripts tab
Ctrl+3	Switch to Graphs tab
Ctrl+4	Switch to Outputs tab
Ctrl+5	Switch to Help tab
+	The new button allows creating a new script
−	The delete button allows removing a script
☰	The sort button allows sorting the column
⁝☰	The order button allows reordering the scripts
✎	The edit button allows script editing
✎	The read-only button disallows script editing

Table 2 - Scripts Commands

⇒ NOTE: * These are built-in system text shortcut commands.

1.2.4 Graphs

The *Graphs* tab has a sub-menu bar and two sections. Section **A** is the *Graphs* column, displaying plots created from script's execution. Section **B** is the *Names* column, displaying a list of graph plots.

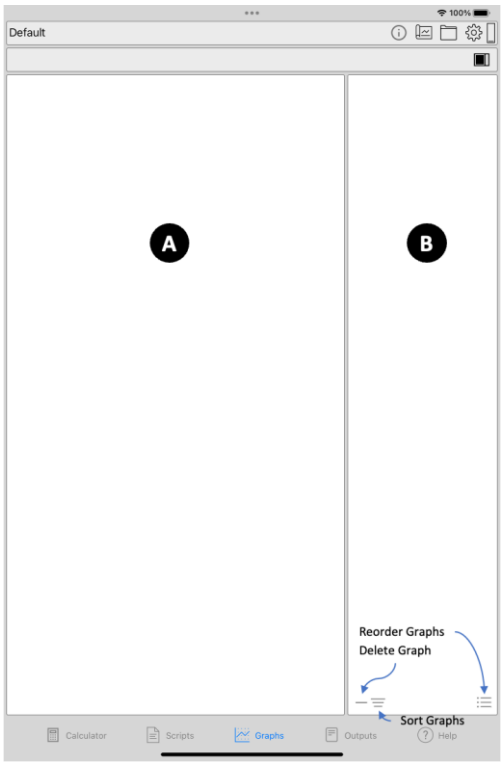

Figure 17 - Graphs Tab

The first part of the menu bar displays the name of the graph in its button commands.

Figure 18 - Graphs Menu Bar

- *Names* will show or hide the *Names* column area.

1.2.4.1 Commands

Command	Description
Ctrl+Option+↓	Select next graph
Ctrl+Option+↑	Select previous graph
Ctrl+Option+←	Select top graph
Ctrl+Option+→	Select bottom graph
Ctrl+1	Switch to Calculator tab
Ctrl+2	Switch to Scripts tab
Ctrl+3	Switch to Graphs tab

Command	Description
Ctrl+4	Switch to Outputs tab
Ctrl+5	Switch to Help tab
—	The delete button allows removing a graph
—	The sort button allows sorting the column
☰	The order button allows reordering the graphs

Table 3 - Graphs Commands

1.2.5 Outputs

The *Outputs* tab has an output area that displays results of the expressions entered on the *Calculator* tab and various informational, warning, and error messages.

⇒ NOTE: The output is limited and based on the *Getting Started* sub-section in the *Menu Bar* section *Configuration* on the *Outputs (KB)* slider setting.

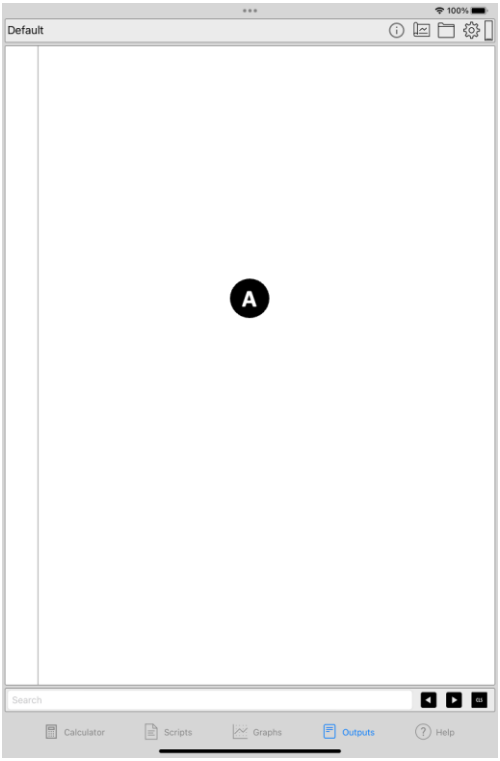

Figure 19 - Outputs Tab

The bottom part of the *Outputs* tab displays basic search-and-clear output results controls.

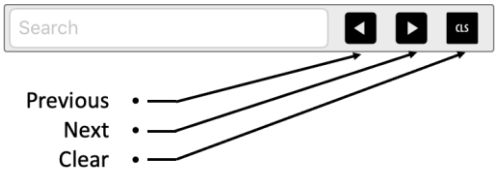

Figure 20 - Outputs Search-and-Clear Bar

- *Previous* searches for previous matching text.

    ⇒ NOTE: If there is no text in the *Search* text field, it will scroll to the previous page of the output results. If the button is double-tapped, it will scroll to the top of the results output.

- *Next* searches for next matching text.

    ⇒ NOTE: If there is no text in the *Search* text field then it will scroll to the next page of the output results. If the button is double-tapped, it will scroll to the bottom of the results output.

- *Clear* clears all output logging messages.

    ⇒ NOTE: If the button is double-tapped, the *Calculator* tab's *Results* area will also be cleared.

1.2.5.1 Commands

Command	Description
Ctrl+S	Search for next matching text
Ctrl+Shift+S	Search for previous matching text
Ctrl+C	Clear the outputted results
Ctrl+Shift+↓	Scroll to next page
Ctrl+Shift+↑	Scroll to previous page
Ctrl+Shift+←	Scroll to the top of the outputted results
Ctrl+Shift+→	Scroll to the bottom of the outputted results
Ctrl+1	Switch to Calculator tab
Ctrl+2	Switch to Scripts tab
Ctrl+3	Switch to Graphs tab
Ctrl+4	Switch to Outputs tab
Ctrl+5	Switch to Help tab

Table 4 - Outputs Commands

1.2.6 Help

The *Help* tab displays the app's built-in document for viewing purposes.

⇒ NOTE: You do not need a cellular or Wi-Fi connection to read the app's included documentation.

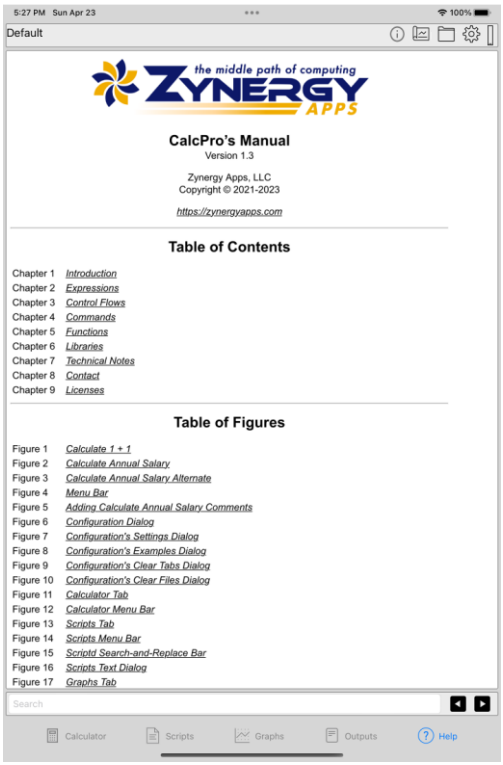

Figure 21 - Help Tab

The bottom part of the *Help* tab displays basic search controls.

Figure 22 - Help Search Bar

- *Previous* searches for previous matching text.

    ⇒ NOTE: If there is no text in the *Search* text field, it will scroll to the previous page of the documentation. If the button is double-tapped, it will scroll to the top of the documentation.

- *Next* searches for next matching text.

    ⇒ NOTE: If there is no text in the *Search* text field, it will scroll to the next page of the documentation. If the button is double-tapped, it will scroll to the bottom of the documentation.

1.2.6.1 Commands

Command	Description
Ctrl+S	Search for next matching text
Ctrl+Shift+S	Search for previous matching text
Ctrl+Shift+↓	Scroll to next page

Ctrl+Shift+↑	Scroll to previous page
Ctrl+Shift+←	Scroll to the top of the documentation
Ctrl+Shift+→	Scroll to the bottom of the documentation
Ctrl+1	Switch to Calculator tab
Ctrl+2	Switch to Scripts tab
Ctrl+3	Switch to Graphs tab
Ctrl+4	Switch to Outputs tab
Ctrl+5	Switch to Help tab

Table 5 - Help Commands

1.2.7 Projects

The *Projects* panel has two sections. Section **A** is the *Summary* column, displaying the project's expressions, variables, scripts, and graphs. Section **B** is the *Names* column, displaying a list of projects.

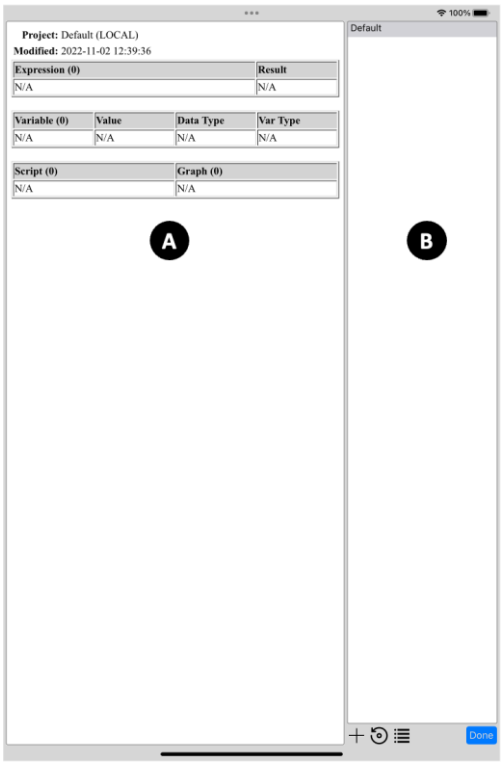

Figure 23 - Projects Dialog

The sub-menu bar at the bottom of the panel contains several button commands.

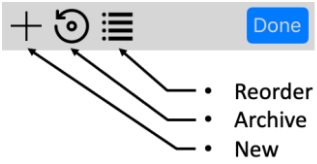

Figure 24 - Projects Menu Bar

- *Reorder* allows reordering the project's *Names* column.
- *Archive* displays *Archive* panel to restore previous saved projects or delete archived projects.

- *New* prompts you for new project name to be added.
- *Done* closes this panel and returns to the main screen of the app.

1.3 Demo Tour

This demonstration will tour the major components of the app to give you an overall understanding of its features and capabilities.

To create a *Demo* project, perform the following steps:

1.		Click the *Project* button in the top-level menu bar to display the *Project* panel.
2.	+	Click *New* at the bottom of the panel's sub-menu bar to create a new project.
3.		The *New Project* dialog will be displayed.
4.		Enter *Demo* in the *Name* text field.
5.		Save the project locally on Apple mobile devices by keeping the *Save project on iCloud* option deselected.
6.		Click *OK* to create the project; you will see *Demo* added to the *Names* column.
7.		Click *Done* at the bottom of the panel's sub-menu bar to return the main screen.

Table 6 - Create Demo Project

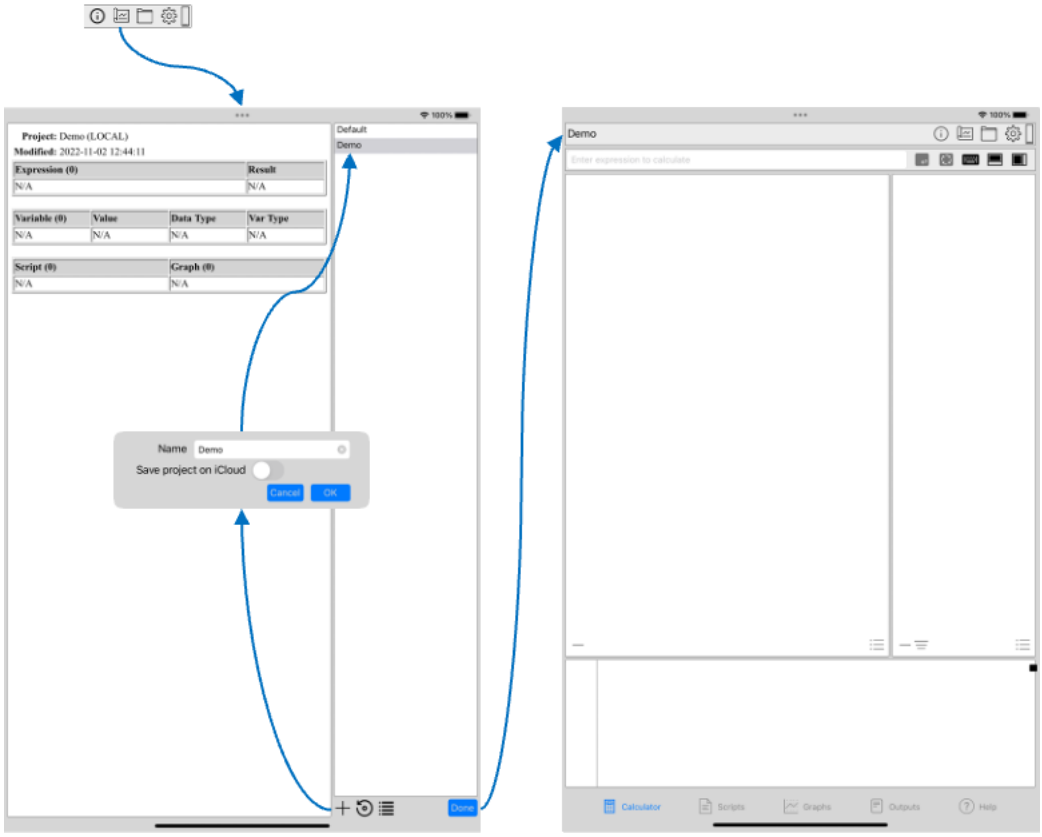

Figure 25 - Create Demo Project

To install advance scripts, perform the following steps:

1.	⚙	Click the *Configure* button in the top-level menu bar to display the *Configuration* dialog.

2.		Click *Examples* to display the *Examples* dialog.
3.		Click *Install Advance Examples* to display the *Install Advance Examples* dialog.
4.		Click *Yes* to populate the *Demo* project's collection of scripts on the *Scripts* tab for demonstration.

Table 7 - Install Advance Script Examples

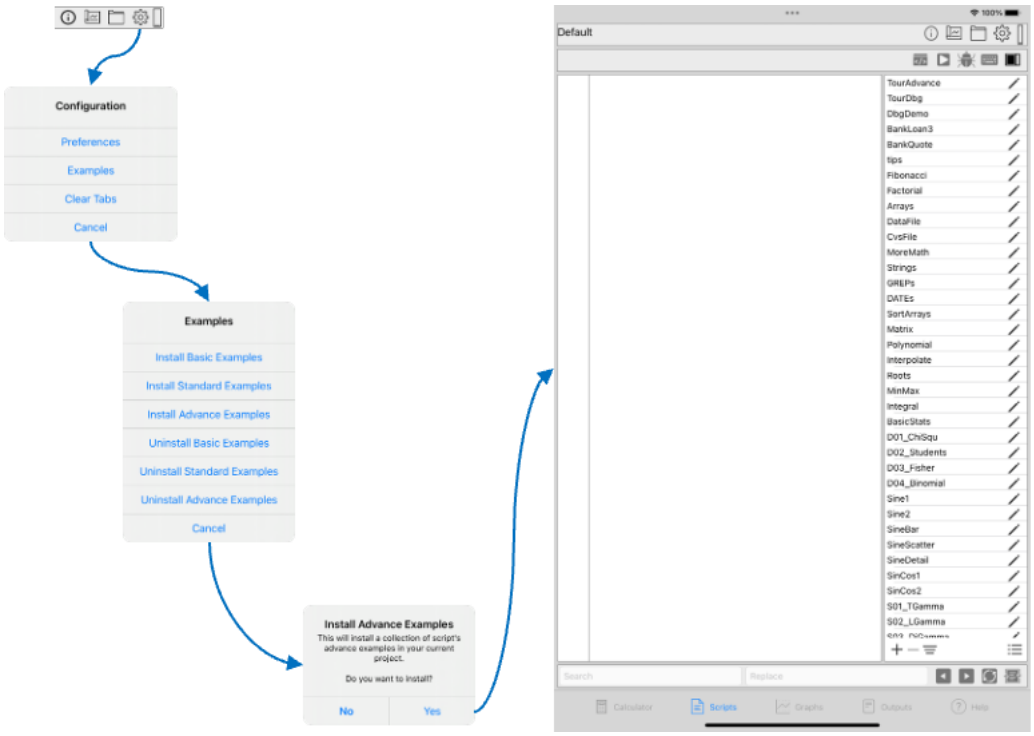

Figure 26 - Install Advance Script Examples

Run the `TourAdvance` script to populate plots on the *Graphs* tab by performing the following steps:

1.		Select the `TourAdvance` script entry on the *Scripts* tab's *Names* column.
2.		Click *Run* in the *Scripts* sub-menu bar to run the script.
3.		The *Stop* button will momentarily appear if the script takes longer than 3 seconds to allow terminating the script before finishing. In this case, let it finish.
4.		Upon completion of the script, it will automatically switch the tab to *Graphs* and select the *D25_Weibull* plot.
5.		Exercise: Select various plots of interest. The name of the graph matches the associated script on the *Scripts* tab.

Table 8 - Run Advance Scripts, Part 1

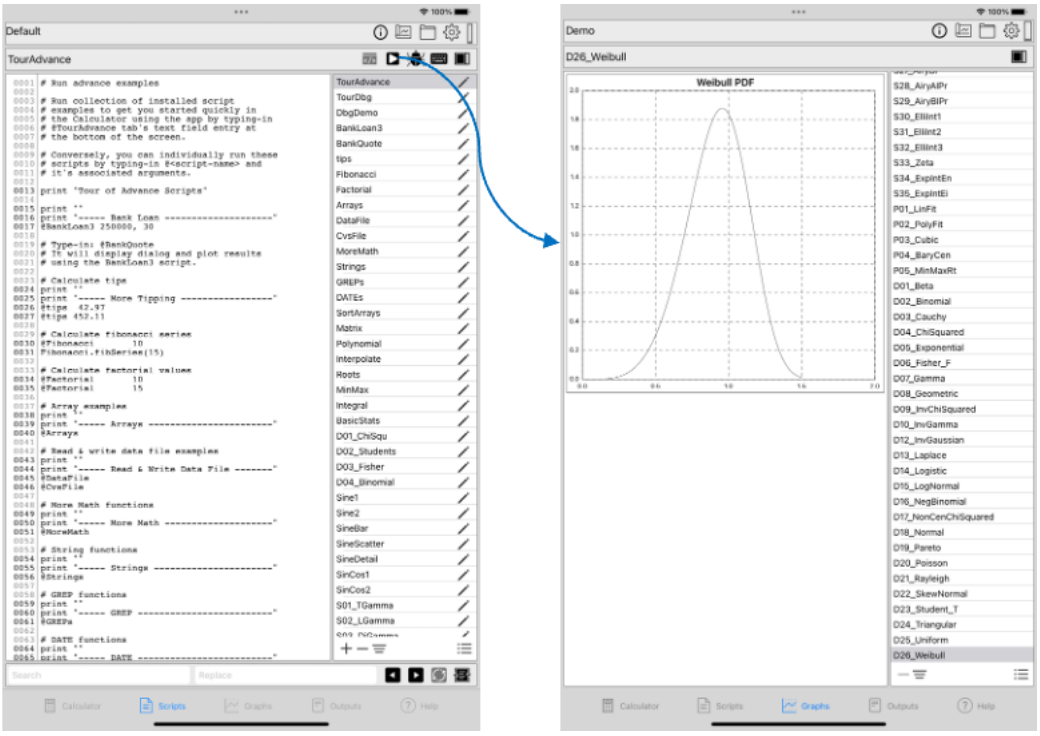

Figure 27 - Run Advance Scripts, Part 1

To add a comment to an expression entry on the *Calculator* tab, perform the following steps:

⇒ NOTE: These steps are applicable to the *Calculator* tab's *Variables* column, and *Scripts* and *Graphs* tabs' *Names* columns.

1.		Click the *Calculator* tab.
2.		Click the `@TourAdvance` entry in the *Expressions* column.
3.	ⓘ	Click *Comments* and the *Enter Expression's Information* dialog will be displayed.
4.		Enter `Advance Demonstration` in the dialog's field.
5.		Click *OK* and see the comment displayed in the *Expressions* column's entry.

Table 9 - Run Advance Scripts, Part 2

Figure 28 - Run Advance Scripts, Part 2

To review generated dataset files from the `TourAdvance` script, perform the following steps:

1.		Click *Data* in the top-level menu bar to display the *Files* dialog.
2.		Click *View Local Files* button in the *Files* dialog to display the *Files* panel.
3.		Click *dataset* entry in the *Names* column to display the contents of the file.
4.		Slide the *dataset* entry to the left to display the set of actions to perform.
5.		Click *Copy* to copy the *dataset*'s text onto the clipboard.
6.		Click *Rename* to rename the *dataset* file and the *Rename "dataset" File* dialog will be displayed. For this exercise, click *Cancel*.
7.		Click *Delete* to delete the *dataset* file and the *Delete File* dialog will be displayed. For this exercise, click *No*.
8.		Click *Done* to close *Files* panel to return to the main screen.

Table 10 - Display Created Dataset Files

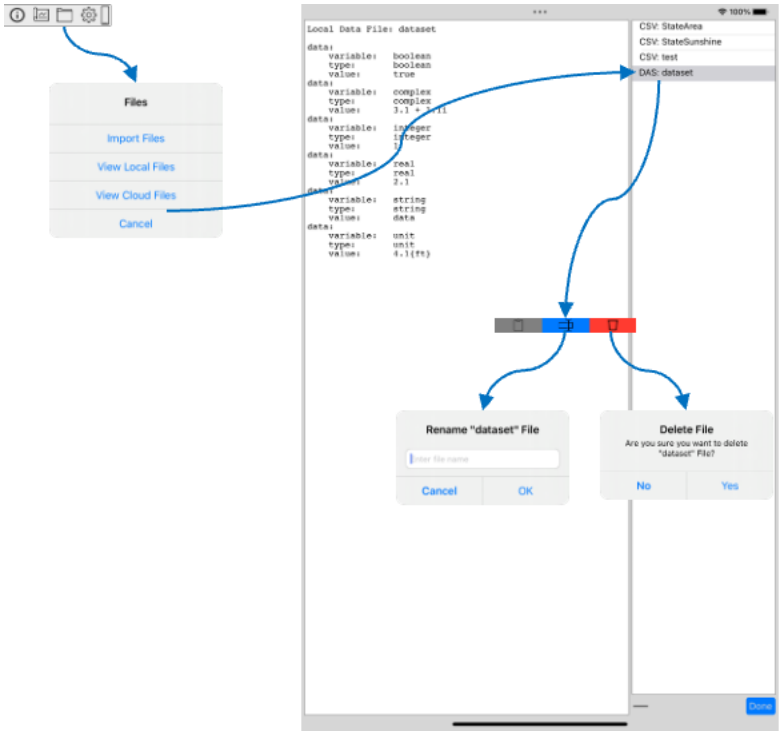

Figure 29 - Display Created Dataset Files

To manage projects, perform the following steps:

1.		Click *Project* in the top-level menu bar to display the *Project* panel.
2.		Click *Demo* project in the *Names* column and slide it to the left.
3.		*Mail* will e-mail the project file. This will invoke the iOS Mail app to e-mail the project file.
4.		Click *Rename* to rename the *Demo* project and the *Rename "Demo" File* dialog will be displayed. For this exercise, click *Cancel*.
5.		Click *Archive* to archive the *Demo* project and the dialog will be displayed.
6.		Enter `Demonstration Project` in the *Title*'s text field.
7.		Enter `Save Demo project for future use` in the *Comment*'s text field.
8.		Click *OK* to archive the *Demo* project.
9.		Click *Default* project in the *Names* column.
10.		Slide *Demo* entry to the left without selecting it. ⇒ NOTE: You can't delete a project if it's currently used.
11.		Click *Delete* to delete the *Demo* project and it will display *Delete Project* dialog.
12.		Click *Yes* to delete the *Demo* project.

Table 11 - Renaming, Archiving, and Deleting Projects

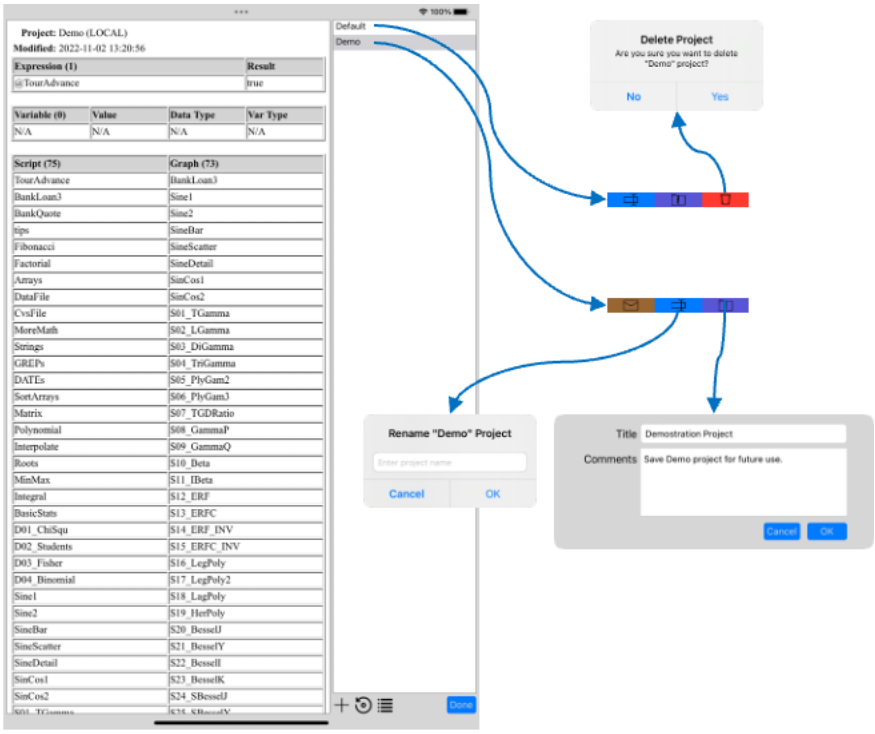

Figure 30 - Renaming, Archiving, and Deleting Projects

To restore or delete archives, perform the following steps:

1.		Click *Archive* in the sub-menu bar at the bottom right of the *Projects* panel to display the *Archive* panel.
2.		Select the second entry *Type* containing *Trash* and slide it to the left.
3.		Click *Trash* to delete the archive and the *Delete Archive* dialog will be displayed. For this exercise, select *No*.
4.		Select the first entry *Type* containing *Archive* and slide it to the left.
5.		Click *Archive* to restore the *Demo* project and the *Restore Archive* dialog will be displayed.
6.		Click *OK* to restore the *Demo* project.
7.		Click *Done* to return *Projects* panel.
8.		Click *Done* again to return the main screen.

Table 12 - Restoring and Removing Archived Projects

Figure 31 - Restoring and Removing Archived Projects

1.4 Hello World Script

To create a classic "Hello World" script, perform the following steps:

1.		Click the *Scripts* tab.
2.	+	Click *New* to create a new script and *Create Script* dialog will be displayed.
3.		Enter `Hello` in the text field.
4.		Click *OK* to create *Hello* script entry in the *Names* column.

5.		Enter the script as shown in the figure below:
		
		`# Simple Hello World Script`
		
		`millennium = 2000`
		`decade = 20`
		`year = millennium + decade + 3`
		
		`Print "Hello World ", year`

6.	</>	Click *Check* to validate the script's syntax for correctness. ⇒ NOTE: If there is an error in the script, a dialog will be displayed indicating the line in the script where syntax is not correct and the button will turn red.
7.	▶	Click *Run* to run the *Hello* script.
8.		Click the *Calculator* tab to see the results.

Table 13 - Create Hello World

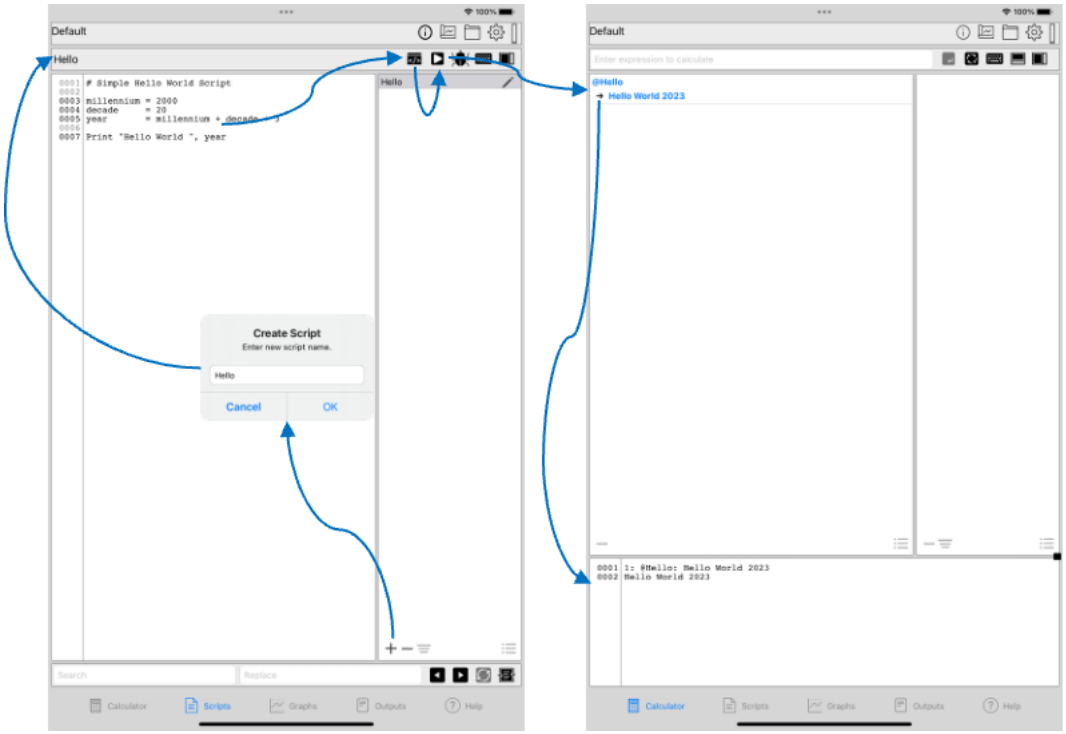

Figure 32 - Create Hello World Script

1.5 Debugging Example

1.5.1 Setting Up the Debugger

Let's create a new script to illustrate a debugging session with the following steps:

1.		Repeat steps 1 to 4 in *Hello World Script* sub-section above to create an another script.

2.		Type-in the debugging script example as shown in the figure below:
		
		`# Debugging Example`
		
		`primes = [ 1, 2, 3, 5, 7 ]`
		
		`day = 0`
		`date = Create.Date("01/01/2023")`
		
		`for i = 1:5`
		`  day = day + primes[i-1]`
		`  date.day = day`
		
		`  print i, ": date: ", date`
		`end`
		
		`print date.info`
		
		`date`

3.		Click *Debugger* to start the script's debugging session.
4.		The *Debugger* panel will be displayed as shown below.

Table 14 - Setting Up the Debugger

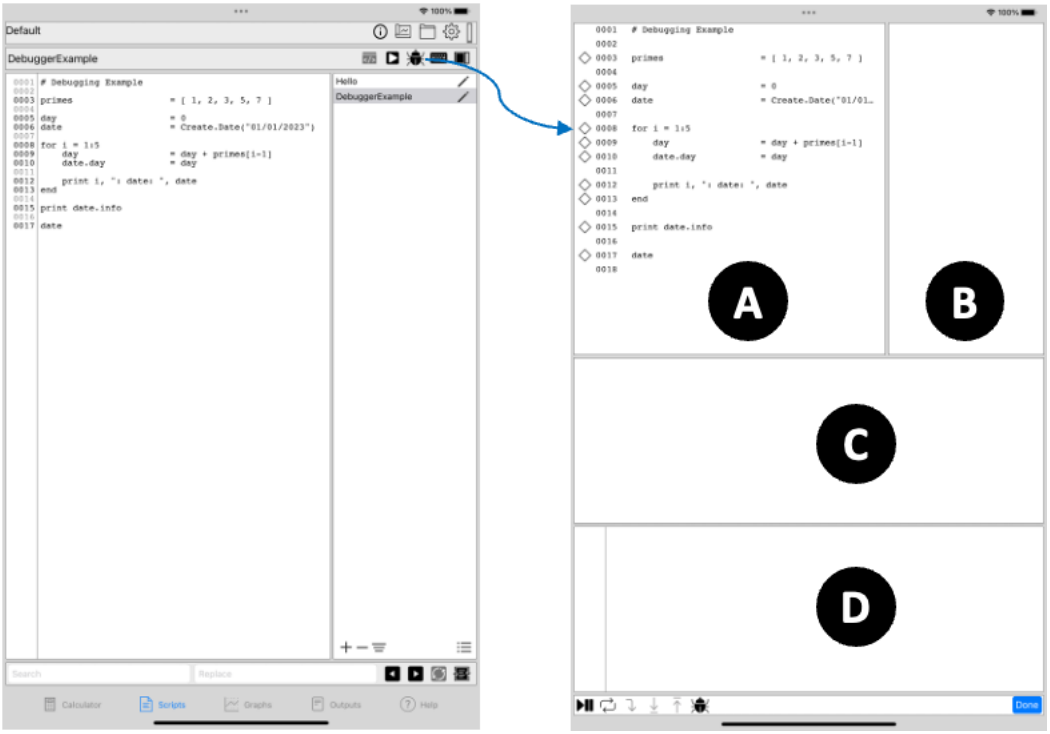

Figure 33 - Setting Up the Debugger

The *Debugger* panel has a sub-menu bar and four sections. Section **A** is the script code currently being debugged. Section **B** is the *Calling Stack*; a list of nested scripts currently being executed. Section **C** lists all *Variables* created within the script's execution flow. Section **D** is *Output*; shows results of the debugging session.

The sub-menu bar at the bottom of the panel contains several button commands.

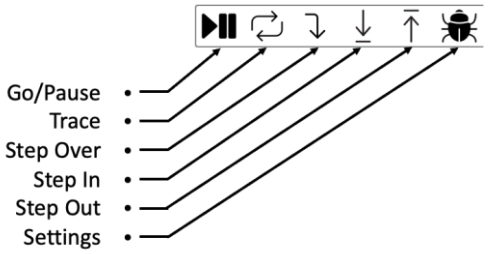

Figure 34 - Debugger Bar

- *Go/Pause* will run the script. Reselect and it will pause the script's execution.
- *Trace* will run the script, but will highlight each line of script code approximately ¼ of a second before preceding to the next line of code.
- *Step Over* will step over one line of code. It may be is useful to step over a script without tracing its lines of code.
- *Step In* will step into that script to execute its lines of code.

- *Step Out* will step out from the currently executing script without tracing its remaining lines of code. It will pause the calling script code.

- *Settings* will display a dialog to clear all break-points in the script.

The *Done* button will terminate the debugging session and return to the *Scripts* tab.

1.5.2 Setting a Break-Point and Running the Debugger

Setup a break-point during the *Debugger* session.

1.	◇	Click *Break-Point* icon in the script containing the line "`print date.info`" to display the *Break-Point* dialog that is used to pause execution of the script.
2.		Click *Set* button in dialog box to enable a break-point on script's line 15.
3.	◆	The script's line will display a highlighted break-point icon.
4.	⟲	Click *Trace* to start the *Debugger* execution of each line of code. ⇒ NOTE: The *Go* ▶⏸ button will transition to *Pause* ⏸ button.
5.	⏸	Click *Pause* to pause the *Debugger*.
6.	⟲	Click *Trace* again to resume debugging of the script.

Table 15 - Setting a Break-Point and Running the Debugger

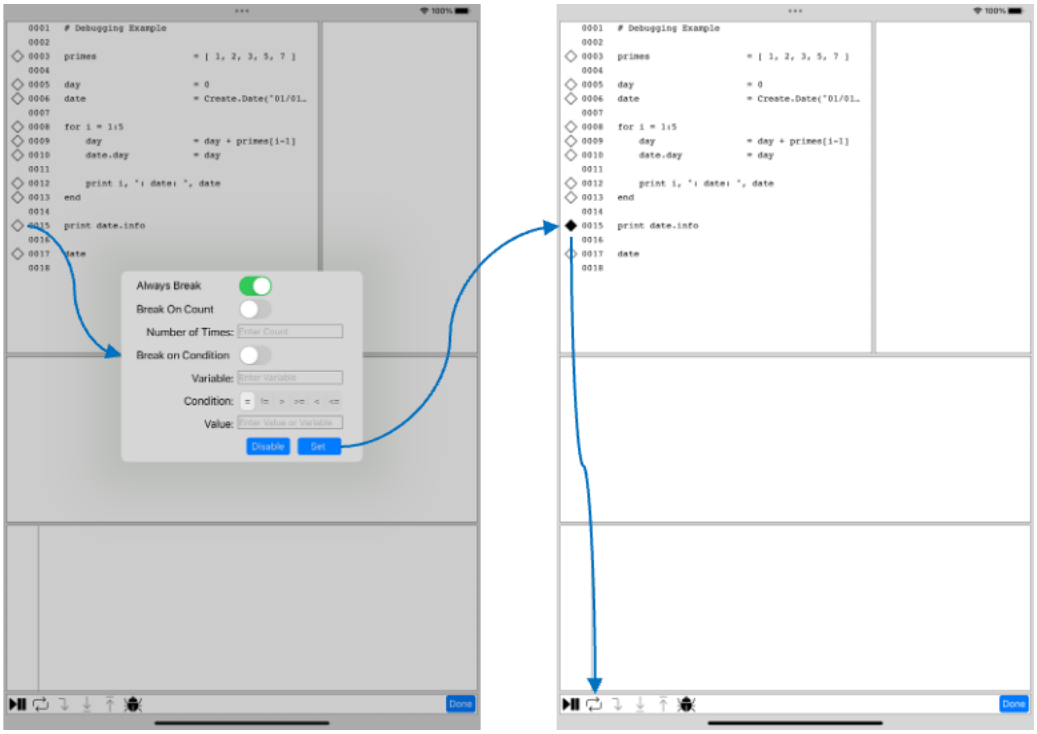

Figure 35 - Setting a Break-Point and Running the Debugger

The *Break-Point* dialog has three break-point options: Always, Count, or Conditional.

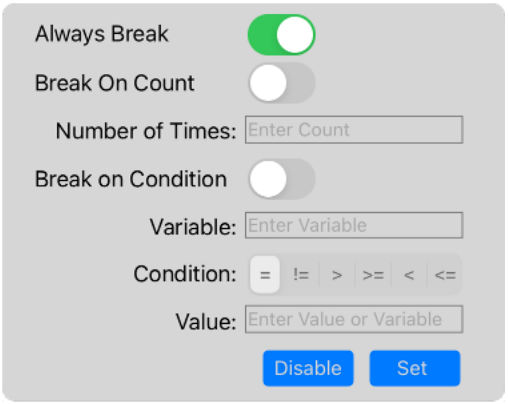

Figure 36 - Break-Point Dialog

- *Always Break* will pause the *Debugger* upon stepping onto the script's line.
- *Break On Count* will pause the *Debugger* up to a specified and thereafter *Number of Times* on the script's line occurrences.
- *Break on Condition* will conditionally pause the *Debugger* upon matching script's *Variable* value versus the specified *Value* of the different *Condition* types: **=** (equal), **!=** (not equal), **>** (greater than), **>=** (greater than or equal), **<** (less than), or **<=** (less than or equal).
- Click *Set* button to enable the break-point in the script. Conversely, click *Disable* button to remove or cancel the break-point setting.

1.5.3 Debugger at the Break-Point

As the *Debugger* traces through the script, it will encounter the break-point and pause the *Debugger*.

1.	⤵	Click *Step-Over* while at the break-point to execute, then pause on the next line.
2.	🔁	Click *Trace* again to resume debugging of the script.

Table 16 - Debugger at the Break-Point

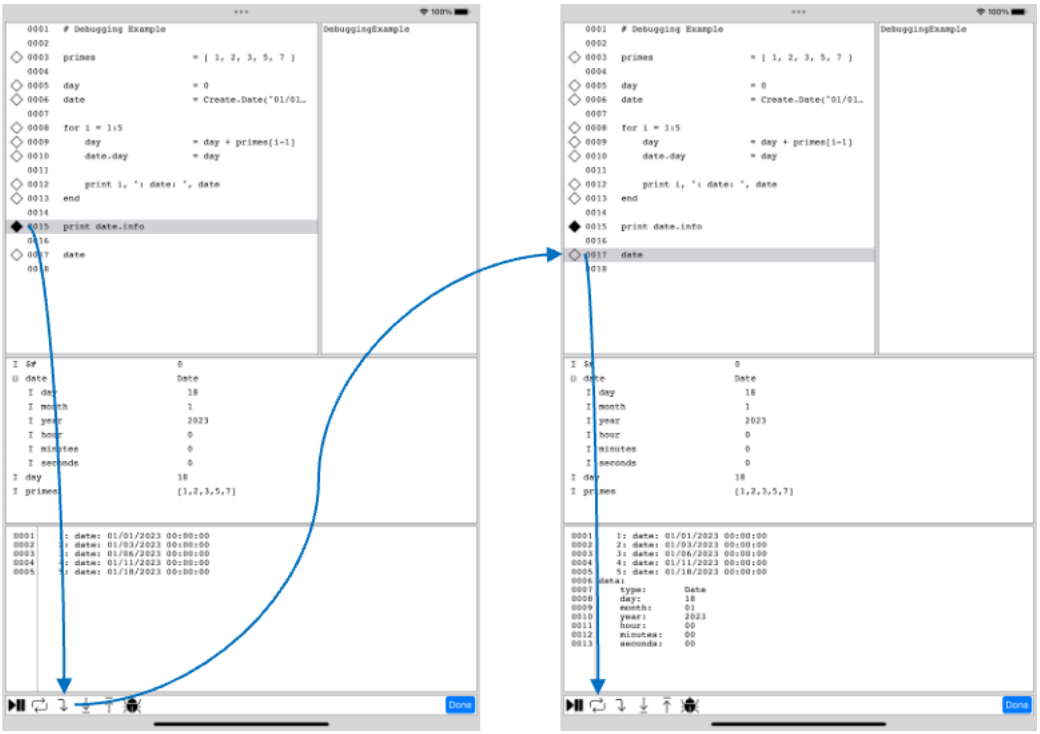

Figure 37 - Debugger at the Break-Point

The *Calling Stack* section will display the list of nested script execution. The top-most entry is the initial script (i.e., *DebuggerExample*). As you run scripts that have nested calls, the other scripts will be display in the *Calling Stack* order. When the *Debugger* is in a paused state, click on the script to view the current script's line execution and its associated *Variable* states.

⇒ NOTE: Refer to *Demo Tour*'s *TourAdvance* script example, but instead debug the *TourDbg* script to see an example of nested calling scripts.

The *Variables* section will display all the variables in the currently executing script. Double-click on the variable's value to enable editing of the current value for: **I**nteger, **R**eal, **C**omplex, **U**nit, **B**oolean, and **S**tring types.

The ⊡, ⊡, ⊡, ⊡, ⊡, and ⊡ buttons are 1 to 6 nested levels of sub-structures containing more variables. The ◆ button indicates seven or more sub-structures. Clicking of these buttons will expand or collapse the contents of its sub-structure.

1.5.4 Completion of the Debugger

Upon completion of the *Debugger* session or selection the *Done* button, the *Scripts* tab will be redisplayed along with all the break-point settings in the script during the *Debugger* session.

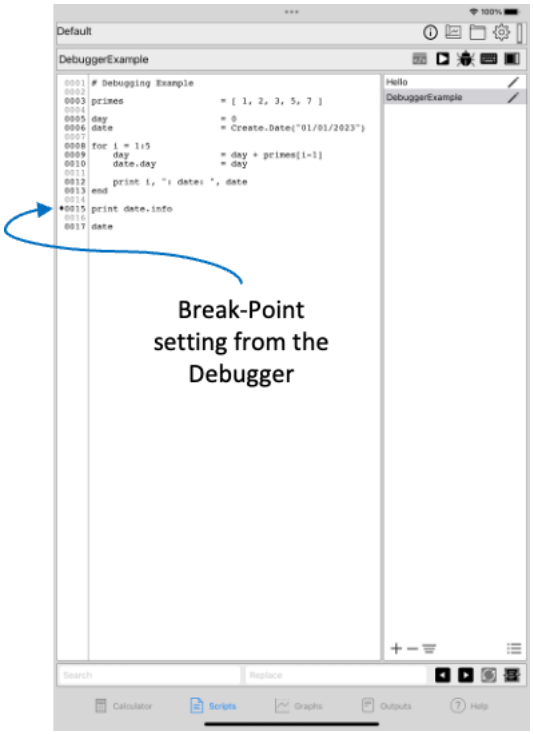

Figure 38 - Completion of the Debugger

1.6 Keyboard

The app has a built-in keyboard to augment the entry of expressions to calculate and script editing capabilities. The following figures show various combinations of keyboard's lower and upper shift keys for numeric (num), alphabet (abc), and built-in f(x) library functions.

Figure 39 - Numeric Lowercase Keyboard

Figure 40 - Numeric Uppercase Keyboard

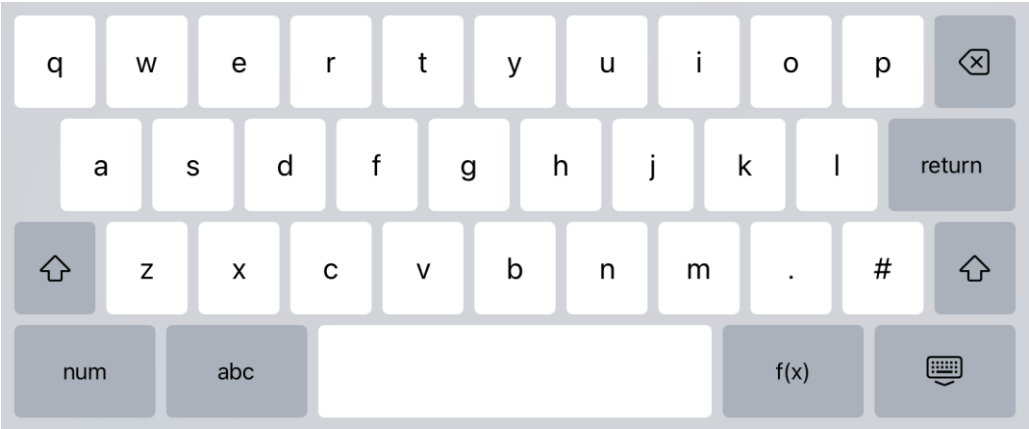

Figure 41 - Alphabet Lowercase Keyboard

Figure 42 - Alphabet Uppercase Keyboard

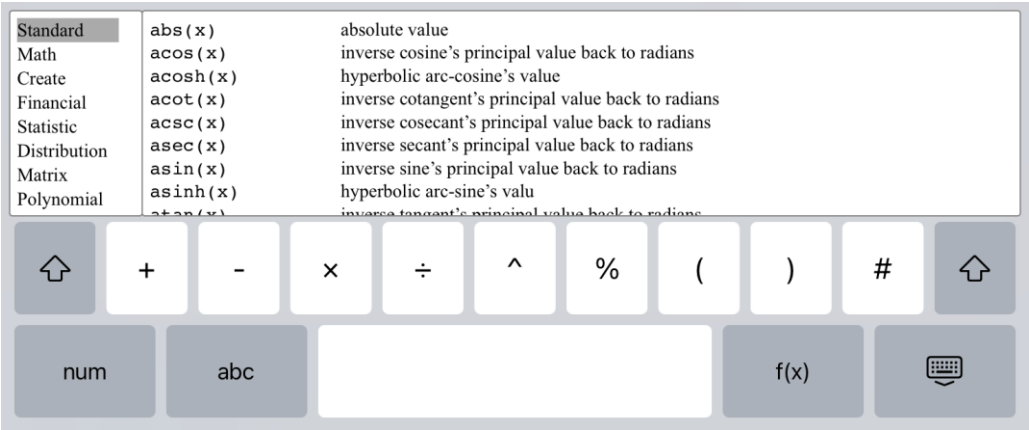

Figure 43 - Library Functions Lowercase Keyboard

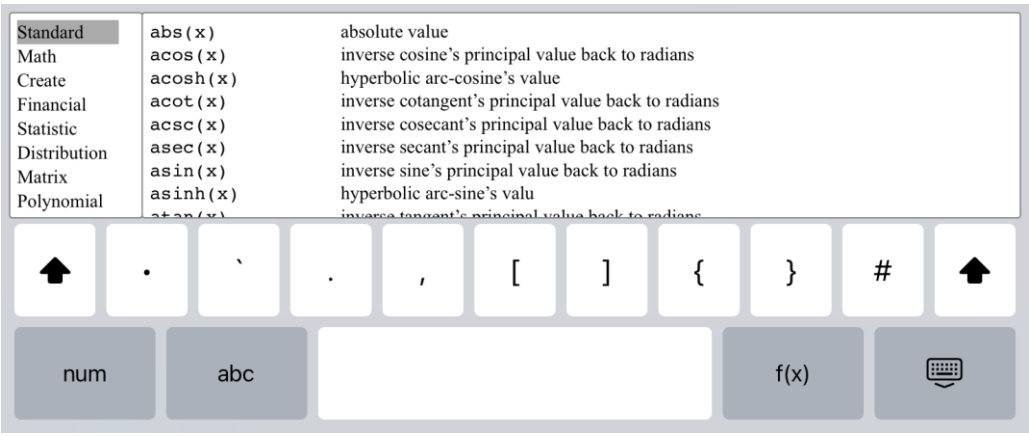

Figure 44 - Library Functions Uppercase Keyboard

⇒ NOTE:
- The *Library Functions Keyboard* is organized into several sub-library categories: *Math*, *Create*, *Financials*, *Statistics*, *Distributions*, *Matrix*, *Polynomial*, *Find*, *Root*, *Interpolate*, *Integration*, *Special*, *Plot*, *Graph*, *Sort*, *Type*, *File*, *Variable*, *Dialog*, *UI*, and *Unit*.
- The first entry, *Standard*, is equivalent to *Math* without its prefix.

2 Expressions

2.1 Variables

A variable name consists of letters (i.e., a-z and A-Z), digits (i.e., 0-9) and the underscore character. The first character must be a letter. It is case-sensitive, so variable `x` is different from variable `X`.

⇒ NOTE: There is no limit to how long the variable name can be. It is generally recommended to make the name descriptive of the purpose of the variable. Often, single characters such as `i`, `j`, `k`, `m`, and `n` are used for array indexing purposes; `x`, `y`, and `z` for plotting purposes.

Variables are locally scoped only, meaning that it's only valid within its script or function. There are no global variables spanning scripts, functions, and projects.

By default, variables will be locally scoped, meaning that it's only valid within its script or function. Global variables are accessible across scripts, functions, and projects.

There are two types of variables: dynamic and declared. The dynamic variables are not specifically declared as `integer`, `real`, `complex`, `unit`, `boolean`, or `string` types. It depends on the assignment to the variable that inherits the data type. Dynamic variable is the default type.

For example:

- `x = 1` — Variable `x` contains an integer value of `1`.
- `x = 1.5` — Contains a real number value of `1.5`.

Declared variables implicitly define the type of variable it will be.

For example:

- `Integer x = 1` — Variable `x` will always contain an integer value containing initialized value `1`.
- `x = 1.5` — Real value of `1.5` will be truncated into an integer value of `1`.

Global variable allows access between scripts, functions, and projects. Global variable is either transient or persistent. Global variable defaults to transient state, where it is only available within in the app's memory until it is restarted. Whereas, persistent is always available until it is explicitly deleted. Refer to *Quick Tour*'s *Menu Bar* section to select the *Data* icon button to bring up *View Local Files* dialog or *Variable.globaldelete* function to delete the global variable.

For example:

- `Global x` — Global variable `x` will contain a value either initialized from another script or function, or currently running script initializing it to zero.

⇒ NOTE: Call *Variable.globalsave* function to create a persistent global variable. Hence, subsequent global declaration will automatically use the data saved by the function.

2.2 Data Types

There are seven fundamental data types: `integer`, `real`, `complex`, `unit`, `boolean`, `string`, and `data`. The `unit` and `data` types are not typical data types found in other computing scripts or languages, such as C/C++, Java, Swift, and others; these are special types and will be covered in sub-sections below.

2.2.1 Integer

An integer number can be represented in several ways. The simplest of integer values are `0`, `141`, `314159`, `-27315`, `+662607015`, etc. There are other representations of integers to help with the readability of large numbers, as shown in the table below.

The integer can contain a period character on every third digit from right to left. For example, `1.000.000` is equivalent to `1,000,000` in traditional representation of 1 million. Refer to the table below for more examples.

If the integer is followed by `k`, `m`, `b`, `t`, or `q`, it's a multiplier of the value for shorthand notations. This can be lowercase or uppercase.

- `k` is thousands
- `m` is millions
- `b` is billions
- `t` is trillions
- `q` is quadrillions

Integer	Value	Comments
`1.000.000`	`1000000`	million
`1.000.000.000`	`1000000000`	billion
`1.000.000.000.000`	`1000000000000`	trillion
`1.000.000.000.000.000`	`1000000000000000`	quadrillion
`1k`	`1000`	thousand
`1.000k`	`1000000`	million
`1m`	`1000000`	million
`1.000m`	`1000000000`	billion
`1b`	`1000000000`	billion
`1.000b`	`1000000000000`	trillion
`1t`	`1000000000000`	trillion
`1.000t`	`1000000000000000`	quadrillion
`1q`	`1000000000000000`	quadrillion

Table 17 - Alternate Integer Representations

⇒ NOTE:
- If you use `1.00`, `1.000`, `1.00000`, etc., this is real number value of `1.0`.
- `1.000` is an exception of using a period with three subsequent digits defaulting to real number `1.0`, rather than `1,000`. This is because this is ambiguous between an integer or real number.

⇒ QUIRK: Dividing integer numbers can yield some unexpected results.

- If you enter the following:

```
Integer a = 1000
Integer b = 1000000
Integer c = a ÷ b
```

  The `c` variable will result to zero, because you are explicitly using integers.

- If you enter `1000 ÷ 1000000`, the result will be promoted to a real data type of `0.001`. Refer to the ***Mixed-Type Promotions*** sub-section on data type promotion rules.

2.2.2 Real

A floating-point number can be represented a couple of ways. The simplest real values are `0.1`, `1.41`, `3.14159`, `-273.15`, `+6.62607015`, etc.

For larger numbers, using powers of tens, the syntax is `[-+] <digits>.<factional> e[-+] <exponent>`.

For example:

- `6.62607015 × 10-34` is represented as `6.62607015e-34`, where `e` is the exponent character followed by numeric powers of tens.
- The exponent of `34` means moving the decimal point `34` digits to the right, which is typically zero, which is a large number. The maximum exponent is `304`, as shown in the ***Ranges*** sub-section below.

    ⇒ QUIRK: If you attempt to operate a large number with a small number, it can exhibit unexpected results, but it's correct in the floating-point realm. This is due to the device's computation limitations. This is normal in the computing world. There is wealth of science on this subject, which is beyond the scope of this manual. You just need to be aware of these obtuse conditions if your results are not what you expected. A key point to remember is that a floating-point number cannot hold an infinitely small or large value.

- If you enter `6.62607015e+34 + 1`, the result is `6.62607015e+34` because the precision is only `17` digits and power of ten makes up the rest of the floating-point value.
    - It's like saying, *`6626070150000000000000000000000000`* `+ 1 =` *`6626070150000000000000000000000001`*, but when the powers of tens are applied that `1` value is lost. Nothing is perfect.

2.2.3 Complex

A complex number is an interesting type of number used in various science and engineering disciplines, such as signal processing, electromagnetism, quantum mechanics, and others.

The complex number syntax is `[-+] <number> + <number> i`, where the first `<number>` is the real part of the complex number, and the second `<number>` is the imaginary part of the complex number. The number is internally represented as floating-point number.

Don't let the word "imaginary" lead you to think it's some mystical number. It's a mathematical way of describing aspects of the scientific description of the natural world.

For example:

- `1 + 2i` — Represented by a numerical value of `1.0` in the real part and `2.0` in the imaginary part of the complex number
- `1.2 + 3e10i` — `1.2` of the real part and `3.0 × 10`^{10} of the imaginary part

- `1.3e-20 + -2i` — 1.3×10^{-20} of the real part and `-2.0` of the imaginary part

2.2.4 Unit

A unit number is a real number associated with a unit description, such as length, time, and energy. Assigning and operating on these numbers will convert from one unit to another like converting feet to miles and vice-versa.

The unit number syntax is `[-+] <number> {<unit>}`, where `<number>` is the real number and `<unit>` is description of the units.

For example:

- `120{mi}` — `120` miles
- `2{h}` — `2` hours
- `60{mi/h}` — Velocity of `60` miles per hour
- `120{mi} ÷ 2{h}` — Computes `60{mi/h}`

When adding and subtracting units, they both must be the same type to allow the conversions. That is, the *Linear* category will allow converting between feet and miles.

For example:

- `5{mi} + 5280{ft}` — Computes `6{mi}`
- `5280{ft} + 5{mi}` — Computes `31680{ft}`

When multiplying and dividing units, the category's units add or subtract each unit numerator and denominator accordingly.

For example:

- `120{mi} ÷ (2{h} × 2{h})` — Units will compute `60{mi/h^2}`, meaning acceleration of `60` miles per hour square

2.2.4.1 Categories

There are 13 categories: *Linear*, *Time*, *Angle*, *Frequency*, *Temperature*, *Power*, *Voltage*, *Amperage*, *Farad*, *Energy*, *Light*, *Weight*, and *Liquid*. The table below lists all the unit combinations available.

Category	Acronym	Description
Linear	in	inch
	ft	foot
	kft	kilofoot
	yd	yard
	mi	mile
	nmi	nautical mile
	rod	rod
	mm	millimeter

Category	Acronym	Description
	cm	centimeter
	dm	decimeter
	m	meter
	km	kilometer
Time	ps	picosecond
	ns	nanosecond
	ms	millisecond
	s	second
	min	minute
	h	hour
	da	day
	wk	week
	mn	month
	yr	year
Angle	d	degree
	r	radian
	mr	milliradian
	gr	gradient
Frequency	hz	hertz
	khz	kilohertz
	mhz	megahertz
	ghz	gigahertz
Temperature	k	kelvin
	c	celsius
	f	fahrenheit
Power	w	watt
	uw	microwatt
	mw	milliwatt
	kw	kilowatt
Voltage	v	volt
	uv	microvolt
	mv	millivolt
	kv	kilovolt
Amperage	i	amperage
	ui	micro-amperage
	mia	milli-amperage
	ki	kilo-amperage
Farad	fa	farad
	pfa	pico-farad
	ufa	micro-farad
	mfa	milli-farad
	kfa	kilo-farad
Energy	erg	energy
	j	joule
	ev	electron volt
	kwh	kilowatt hour
Weight	oz	ounce

Category	Acronym	Description
	lb	pound
	g	gram
	mg	milligram
	kg	kilogram
Liquid	l	liter
	ga	gallon

Table 18 - Unit Categories

2.2.5 Boolean

Boolean is simple data type that can only be `true` or `false`.

For example:

- `a = true` Sets the variable value to `true`
- `a = false` Sets the variable value to `false`

2.2.6 String

String is a simple data type container of sequence of characters. You can perform simple concatenation of strings into a single string value.

For example:

- `"Hello"` A simple string containing the word Hello
- `'World'` Alternate way of defining a string using single-quote character
- `"Hello" + " " + "World"` Concatenates strings to create a new `"Hello World"` string

⇒ NOTE: If one is performing numerical arithmetic operation with the string, it will attempt to convert the string to a number before applying the operation; otherwise, it will convert the numerical value to string, then concatenate.

2.2.7 Data

The data type is special instance of a data container to perform specific operations. These types consist of *Array*, *Structure*, *Dictionary*, *Polynomial*, *Interpolate*, *Dataset*, *Plot*, *Graph*, *Distribution*, and *Dialog* created by the *Create* library functions. Refer to the library for details about using these data containers.

- *Array* container of single- to multi-dimension array of `integer`, `real`, `complex`, and other basic data types
- *Structure* container of *n*-level sub-fields of variables
- *Dictionary* container *n*-dimensional array of variables
- *Polynomial* container of polynomial coefficients
- *Interpolate* container of interpolating a point from set of defined points
- *Dataset* container is series of (x, y) data points with plotting parameters used by the *Plot* container
- *Plot* container of (x, y) data points of the *Dataset* along with other plotting parameters
- *Graph* container of *Plot* containers to plot single to multiple plots within its graph's chart
- *Distribution* container of standard statistical methodologies

- *Dialog* container of dialog parameters used to display a dialog requesting information needed by a script to preform computations based on the user's input.

2.2.8 Declarations

There are cases where you want variables to always be `integer`, `real`, or other types, rather than dynamically changing to the assigned value's data type.

> ⇒ NOTE: The declaration `integer`, `real`, `complex`, `unit`, `boolean`, and `string` keywords are case-insensitive.

For example:

- `integer i` Defines the variable `i` to always contain the `integer` data type
- `real r` Defines the variable `r` to always contain the floating-point `real` data type
- `complex c` Defines the variable `c` to always contain the `complex` data type
- `unit u` Defines the variable `u` to always contain the `unit` data type
- `boolean b` Defines the variable `b` to always contain the `boolean` data type
- `string s` Defines the variable `s` to always contain the `string` data type

2.2.9 Ranges

As always, everything has limits. The following table describes each of the numerical data type ranges of possible values.

Type	Minimum	Maximum	Description
integer	-9223372036854775808	+9223372036854775807	64 bits
real	2.2250738585072009 x 10^{-308}	1.7976931348623157 x 10^{+308}	64 bits
complex	1.1754943508 × 10^{-38}	3.4028234664 × 10^{38}	Real and imaginary components are 32-bits
unit	2.2250738585072009 x 10^{-308}	1.7976931348623157 x 10^{+308}	Similar to `real` with `unit` information
boolean	false	true	false = 0, true = 1

Table 19 - Type Ranges

2.2.10 Mixed-Type Promotions

The primary feature of the app is it not requiring the specifying of the data type. However, depending on how the calculations are performed, one may need to understand the various data type conversions that can affect the results.

The table below describes data type conversions. The symbol ⊗ is an operator for arithmetic operations: + (add), − (subtract), × (multiply), ÷ (divide), and % (modulo). The ⟸ is an operation result of the ⊗ operator.

Convert		Left		Right	Comments
integer, real	⟸	integer	+	integer	If addition exceeds maximum integer value, result is converted to real.
integer	⟸	integer	−	integer	

Convert		Left		Right	Comments
integer, real	⇐	integer	×	integer	If multiplication exceeds maximum integer value, result is converted to real.
real	⇐	integer	÷	integer	
real	⇐	integer	⊗	real	
complex	⇐	integer	⊗	complex	
unit	⇐	integer	⊗	unit	
	⇐	integer	⊗	boolean	Invalid operation
	⇐	integer	⊗	string	Invalid operation
real	⇐	real	⊗	integer	
real	⇐	real	⊗	real	
complex	⇐	real	⊗	complex	
unit	⇐	real	⊗	unit	
	⇐	real	⊗	boolean	Invalid operation
	⇐	real	⊗	string	Invalid operation
complex	⇐	complex	⊗	integer	
complex	⇐	complex	⊗	real	
complex	⇐	complex	⊗	complex	
unit	⇐	complex	⊗	unit	
	⇐	complex	⊗	boolean	Invalid operation
	⇐	complex	⊗	string	Invalid operation
unit	⇐	unit	⊗	integer	
unit	⇐	unit	⊗	real	
unit	⇐	unit	⊗	complex	
unit	⇐	unit	⊗	unit	
	⇐	unit	⊗	boolean	Invalid operation
	⇐	unit	⊗	string	Invalid operation
integer	⇐	boolean	⊗	integer	
real	⇐	boolean	⊗	real	
complex	⇐	boolean	⊗	complex	
unit	⇐	boolean	⊗	unit	
	⇐	boolean	⊗	boolean	Invalid operation
string	⇐	boolean	⊗	string	
integer	⇐	string	⊗	integer	If string is not a numerical value, it's zero.
real	⇐	string	⊗	real	If string is not a numerical value, it's zero.
complex	⇐	string	⊗	complex	If string is not a numerical value, it's zero.
unit	⇐	string	⊗	unit	If string is not a numerical value, it's zero.
boolean	⇐	string	⊗	boolean	If string is not a numerical value, it's zero.
string	⇐	string	+	string	The + add operator is only applicable for strings.

Table 20 - Data Conversions

2.3 Arrays

You can create one- or two-dimension arrays used to calculate a series of values typically applied to plotting graphs. The maximum number of dimensions is four.

The array indexing starts at zero to the length minus one. So, if the array length is `5`, the indexing into the array will go from `0` to `4`.

For example, creating an array:

- `[1,2,3]` Defines an integer one-dimension array
- `[[1,2,3], [4,5,6]]` Defines an integer two-dimension array

For example. declaring arrays:

- `integer i[5]` Defines single integer array length of `5` elements
- `real r[10,10]` Defines two-dimension floating-point array of `10 x 10`
- `complex c[20,20,20]` Defines three-dimension complex array of `20 x 20 x 20`
- `unit u[3,5,7,9` Defines four-dimension unit array of `3 x 5 x 7 x 9`
- `boolean b[100]` Defines single boolean array length of `100` elements
- `string s[1000]` Defines single string array length of `1000` elements

For example, initializing and printing one-dimension arrays:

```
integer a[3]
integer b[3] = [1,2,3]

a[0] = 1
a[1] = 2
b[2] = 3

v = [1, 2, 3]

print a[0], a[1], a[2]
print b[0], b[1], b[2]
print v[0], v[1], v[2]
```

For example, in multi-dimension arrays:

```
real a[2,3]

a[0,0] = 1.0
a[0,1] = 2.0
a[0,2] = 3.0
a[1,0] = 4.0
a[1,1] = 5.0
a[1,2] = 6.0

v = [[1.0,2.0,3.0], [4.0,5.0,6.0]]

print a[0,0], a[0,1], a[0,2], a[1,0], a[1,1], a[1,2]
print v[0,0], v[0,1], v[0,2], v[1,0], v[1,1], v[1,2]
```

2.4 Arithmetic Operators

The arithmetic expression used to calculate a standard set of operators is found on typical calculators consisting of +, −, ×, ÷, %, and ^ characters. The calculations ordering depends on the precedence order, meaning higher precedence is done first, down to the lowest, as shown in the table below. Use parentheses `(` and `)` to change the order of the precedence's computation.

For example:

- `1 + 2 × 3` — The multiply operator has higher precedence by first multiplying `2 × 3`, then adding 1, resulting in `7`.
- `(1 + 2) × 3` — Using the parentheses changes of the order of precedence by first adding `1 + 2`, then multiplying by `3`, and resulting in `6`.
- `22 % 4` — The modulo operator first takes `22`, divides by `4`, uses the whole-number portion of the result, and multiplies by `4`, then subtracts it from `22`, resulting in `2`. The resulting effect of modulo `4` ranges from `0` to `3`.
- `10^3` — The power operator will perform 10^3 (i.e., `10×10×10`), resulting in `1000`.

Operator	Precedence	Description
+	1	Add
−	1	Subtract
×	2	Multiply can use alternate character: *
÷	2	Divide can use alternate character: /
%	2	Modulo
^	3	Power of tens

Table 21 - Arithmetic Operators

2.5 Relational and Logical Operators

`If` and `While` statements, as described below in *Control Flows* chapter, uses a set of operators used determine if the condition is true to execute subsequent script's statements. As in the arithmetic calculations, conditional expression ordering depends on the precedence order within the logical expressions, meaning that higher precedence is done first, then on down to the lowest, as shown in the table below. Use parentheses `(` and `)` to change the order of the precedence's comparison.

The relational operators compare the expressions whether they are equal, not equal, less than, or greater than. The logical operator groups a series of relational expressions whether they are true using `&` and `|` (i.e., and, or, respectively) operators.

For example:

- `1 == 1` — Results in `true` condition
- `2 != 2` — Results in `false` condition
- `1 < 2` — Results in `true` condition
- `1 == 1 | 2 != 2` — Results in `true` condition
- `1 == 1 & 2 != 2` — Results in `false` condition

Operator	Precedence	Description
==	3	Relational: Equal
!=	3	Relational: Not equal
<	3	Relational: Less than
<=	3	Relational: Less than or equal
>	3	Relational: Greater than
>=	3	Relational: Greater than or equal
&	2	Logical: And
|	1	Logical: Or

Table 22 - Relational and Logical Operators

3 Control Flows

The control flows of the script's statements specify the order in which calculations are made. The intent here is not to teach how to program; there are many courses and books on programming. This chapter will attempt to quickly illustrate the script's programming language fundamentals for you to develop your own collections of scripts. Refer to ***Demo Tour*** section installing a collection of script's examples.

The control's `if`, `else`, `elseif`, `for`, `while`, `next`, `break`, and `end` keywords are case-insensitive.

3.1 If-Else

The `if-else` clause is used to calculate decision points to execute its sub-statements.

The syntax is as follows:

```
if expression
     statement
else
     statement
end
```

or

```
if expression
     statement
elseif expression
     statement
else
     statement
end
```

The *`expression`* is a conditional expression, as previously discussed in the ***Relational and Logical Operators*** sub-section. If the *`expression`* is `true`, it will proceed following one or more *`statements`*. If `false`, it will proceed to the `else`'s `statements`. The `else` clause is optional.

If there is an `elseif` clause, it will be evaluated if the previous expression is `false`. There can be multiple `elseif` clauses.

The *`statement`* can be one or more expressions to calculate something or additional nested clauses consisting of `if`, `else`, `for`, `while`, etc.

For example:

```
a = 1
if a < 3
     yup = true
else
     yup = false
end
```

The results of the variable's `yup` will be `true`.

3.2 For

The `for` clause is used to loop though executing its sub-statements.

The syntax is as follows:

```
for variable = list
     statement
end
```

The `variable` is current value looping though a `list` of values. The `list` is a series of numerical values or arithmetic expressions assigned to the `variable`.

The *`list`* syntax is as follows:

- *`expression:expression:expression`*, …
- *`expression`*, …

The `expression:expression:expression` uses the first `expression` as the starting value of the `variable` up to the maximum value in the second `expression`. The third `expression` is an optional incremental value; otherwise, the default incremental value is `1`. There can be a list of these separated by commas.

The alternative method is a series of numerical or arithmetic expressions, or another sub-list of values separated by commas.

Again, the `statement` can be one or more expressions to calculate something or additional nested clauses consisting of `if`, `else`, `for`, `while`, etc.

For example:

```
count = 0
for i = 1:6
     if i <= 3
          count = count + 1
     end
end
```

The results of the variable's `count` will be `3`.

Another example using an incremental value:

```
count = 0
for i = 1:6:2
     if i <= 3
          count = count + 1
     end
end
```

The results of the variable's `count` will be `2`.

Last example using a list of expressions:

```
count = 0
for i = 1+1, 2+2, 3+3
     if i <= 3
          count = count + 1
     end
end
```

The result of the variable's `count` will be `1`.

There are two other special statements: *`next`* and *`break`*. The `next` statement causes the execution flow to increment the *`variable`* to its next value, proceed executing its *`statement`*. The *`break`* statement causes the exit from its `for`'s loop execution.

This example will prematurely exit the loop once the `count` variable is `2`:

```
count = 0
for i = 1:6
     if i <= 3
          count = count + 1
          if count == 2
               break
          end
     end
end
```

The result of the variable's `count` will be `2`.

This example uses the *next* statement, which the variable's i will go though from 1 to 6:

```
count = 0
for i = 1:6
     if i > 3
          next
     end

     count = count + 1
end
```

The result of the variable's count will be 3.

3.3 While

The while clause is used to loop though executing its sub-statements.

The syntax is as follows:

```
while expression
     statement
end
```

The *expression* is a conditional expression, and if true, it will proceed following *statement*. The *next* and *break* statements are available as used in the for clause.

For example:

```
count = 0

i = 1
while i <= 6
     if i <= 3
          count = count + 1
     end

     i = i + 1
end
```

The result of the variable's count will be 3.

4 Commands

4.1 Loop

The loop command is combination of the for and expression statements.

The syntax is as follows:

```
loop variable = list @ expression
```

The *variable* is current value looping though a *list* of values. The *list* is a series of numerical values or arithmetic expressions assigned to the *variable*. Refer to the ***For*** section details about using the *variable* and *list* parts of the `loop` command.

The *expression* will be executed for each *variable*'s value in the `loop`.

For example:

```
loop i = 1:6 @ print "i: ", i
```

The result of the command will print out the variable's `i` value `6` times.

4.2 Print

The `print` command will print the list arguments on the statement.

The syntax is as follows:

```
print expression, …
```

The `print` command will print out the results of each *expression* in the list separated by commas. Refer to the ***Loop*** section for an example of using the `print` command.

4.3 Debugger

The `debugger` command is used to interact with the Scripts' debugger during execution of the script. Refer to the ***Debugging Example*** sub-section and installed *Advance Examples*' `DbgDemo` script example.

4.3.1 Pause

The `pause` option of the `debugger` command will pause the script's execution while running in the debugger mode.

For example:

```
debugger pause
```

The result of the command will pause the script debugger execution.

4.3.2 BreakPoint

The `breakpoint` option of the `debugger` command will pause the debugger at the specified script's line.

For example:

```
debugger breakpoint 10
```

The result of the command will pause the script debugger's execution when it encounters the script's line number `10`.

4.3.3 ClearPoint

The `clearpoint` option of the `debugger` command will clear any break-point setting at the specified script's line.

For example:

```
debugger clearpoint 10
```

The result of the command will clear the break-point at the script's line number `10`. Hence, the debugger will not pause upon executing the script's line number `10`.

⇒ NOTE: If the break-point line number is `0`, then it will clear all break-points within the script.

4.3.4 Dump

The `dump` option of the `debugger` command will display all the variables within currently executing script or function.

For example:

```
debugger dump
```

The result of the command will show current variables' values.

4.3.5 Abort

The `abort` option of the `debugger` command will terminate debugger and return to the *Scripts* tab.

For example:

```
debugger abort
```

The result of the command will terminate the script's debugging session.

4.4 Exit

The `exit` command terminates the execution of the script, including its nested script invocations.

5 Functions

Creating script functions can break up large computing chunks into smaller tasks to enable the reuse of common scripting code, manageable code organization, and general stability of the computations. A script function can accept input arguments used for its computation and return its results.

The syntax is as follows:

```
function name(arg1, arg2, …, argn)
     statement
end
```

The *name* is the function's name that can be called from the scripts and is case-sensitive. The function can have zero to several arguments passed to the function: *arg1*, *arg2*, etc. The *return* statement is optional and used to return the results to the caller of the function.

For example:

```
sum = addIt(1, 2)

function addIt(a, b)
     c = a + b

     return c
end
```

The result of the function call will be 3.

⇒ NOTE: The maximum number of function recursions is 256. This means calling the function within itself. This limit is employed to avoid infinite recursion conditions.

For example:

```
function fibonacci(n)
     if n == 0
          return 0
     elseif n == 1
          return 1
     else
          return fibonacci(n-1) + fibonacci(n-2)
     end
end
```

6 Libraries

The purpose of this chapter is to briefly describe all the functions without going into detail about trigonometry, matrices, etc. There are many venues in which to learn these mathematical concepts.

The functions are grouped into a collection of libraries, such as *Math*, *Matrix*, *Financials*, and others for your calculation needs. You will find many of these functions are commonly found in other programming languages,

such as C/C++, Java, FORTRAN, Python, and others. The primary differences are argument inputs that can be scalar or an array of `integer`, `real`, `complex`, `unit`, `boolean`, and `string` data types.

All functions and constants are case-insensitive.

⇒ NOTE: Many functions have graph examples that use the Demo's `Utils` script's `PlotIt`, `Plot2It`, `Plot3It`, `Graph3It`, `Graph4It`, `Graph5It`, and `ScatterIt` functions to minimize the amount of example code illustration in the function's usage. Refer to the ***Demo Tour*** section on installing the `Utils` script.

6.1 Math

6.1.1 Trigonometry

⇒ LINK: https://en.wikipedia.org/wiki/Trigonometry

6.1.1.1 Math.cos

`math.cos(x)` – Returns cosine angle of `x` radians

The `x` argument is the angle expressed in radians. The periodic range of `x` is $0 \leq x \leq 2\pi$. One radian is equivalent to `180 ÷ PI` degrees. Alternately, use `57.2958{d}` degrees, which is equivalent to `1{r}` radian.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `cos(x)`.

For example:

- `Math.cos(0)` Returns `1`
- `Math.cos(PI)` Returns `-1`

Plot example using radians - $0 \leq x \leq 2\pi$, where π is equivalent to 180°:

```
x = [ 0.0 : PI × 2 : (PI × 2) ÷ 360 ]
Utils.PlotIt("Cosine",                  \
             "Plt001_Cosine",           \
             Plot.CHART.LINE,           \
             x, Math.COS(x),            \
             0.0, PI × 2, -1,1,5,8)
```

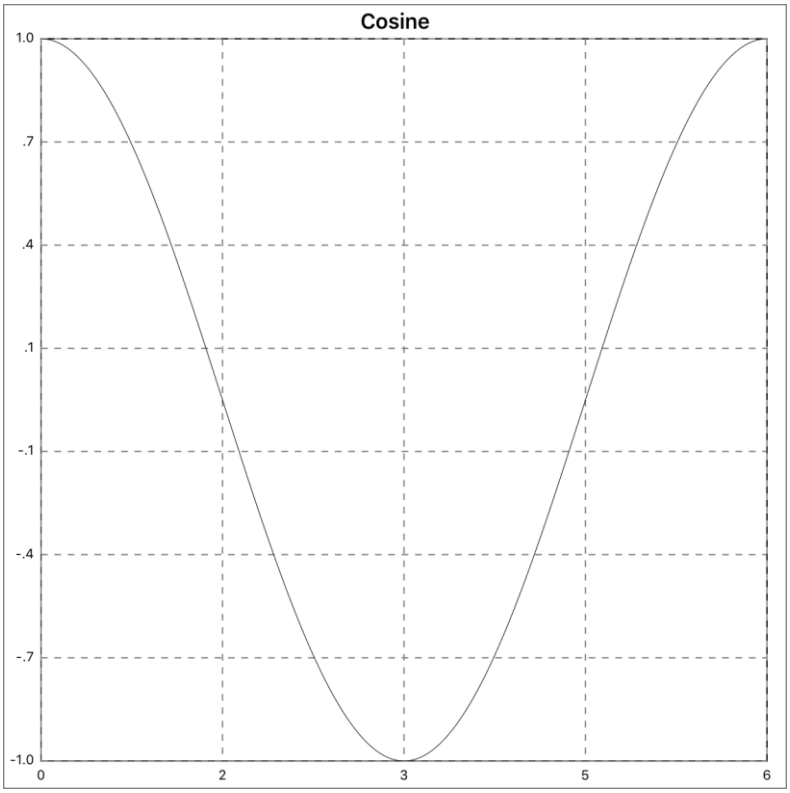

Figure 45 - Cosine Plot

6.1.1.2 Math.sin

`math.sin(x)` – Returns sine angle of `x` radians

The `x` argument is the angle expressed in radians. The periodic range of `x` is $0 \leq x \leq 2\pi$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `sin(x)`.

For example:

- `Math.sin(0)` Returns `0`
- `Math.sin(PI)` Returns `0`

Plot example using degrees - $0° \leq x \leq 360°$:

```
x = [ 0{d} : 360{d} : 1{d} ]
Utils.PlotIt("Sine",                    \
             "Plt002_Sine",             \
             Plot.CHART.LINE,           \
             x, Math.SIN(x),            \
             0{d}, 360{d}, -1,1, 5,8)
```

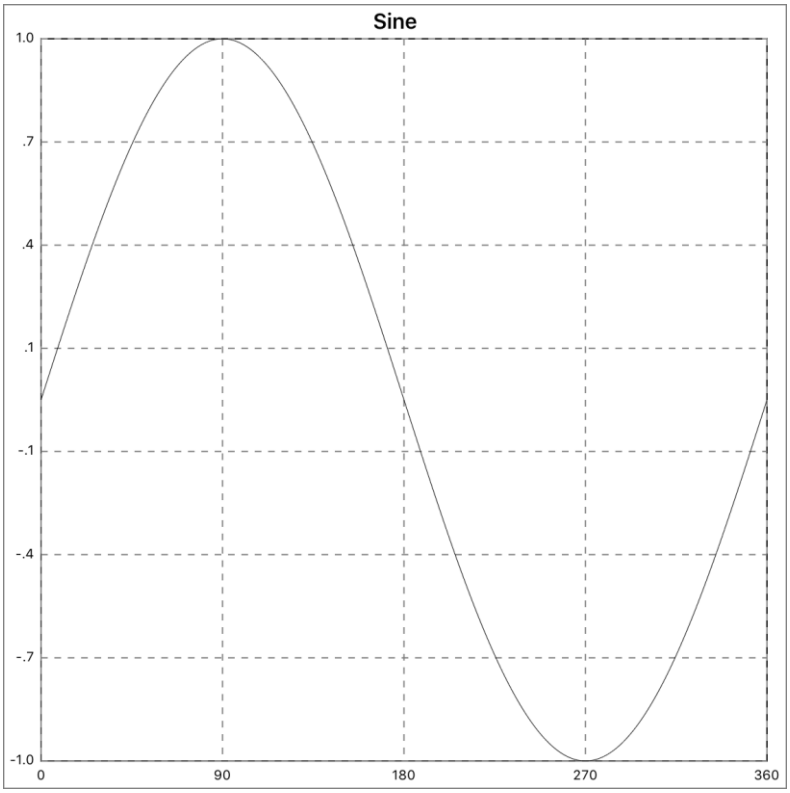

Figure 46 - Sine Plot

6.1.1.3 Math.tan

`math.tan(x)` – Returns tangent angle of `x` radians

The `x` argument is the angle expressed in radians. The periodic range of `x` is `0 ≤ x ≤` π.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `tan(x)`.

For example:

- `Math.tan(0)` Returns `0`
- `Math.tan(PI ÷ 4)` Returns `1`

Plot example using degrees - `0° ≤ x ≤ 360°`:

⇒ NOTE: The ranges, `0° ≤ x ≤ 360°`, must be broken up into three data sets to handle tangent's discontinuous boundaries on the π `÷ 2` periodical cycles.

```
x1 = [ 0{d}   : 89{d}  : 1{d} ]
x2 = [ 91{d}  : 269{d} : 1{d} ]
x3 = [ 271{d} : 360{d} : 1{d} ]
Utils.Plot3It("Tangent",                \
              "Plt003_Tanget",          \
              Plot.CHART.LINE,          \
              x1, Math.TAN(x1),         \
```

```
    x2, Math.TAN(x2),          \
    x3, Math.TAN(x3),          \
    0{d}, 360{d}, -5, 5, 5, 7)
```

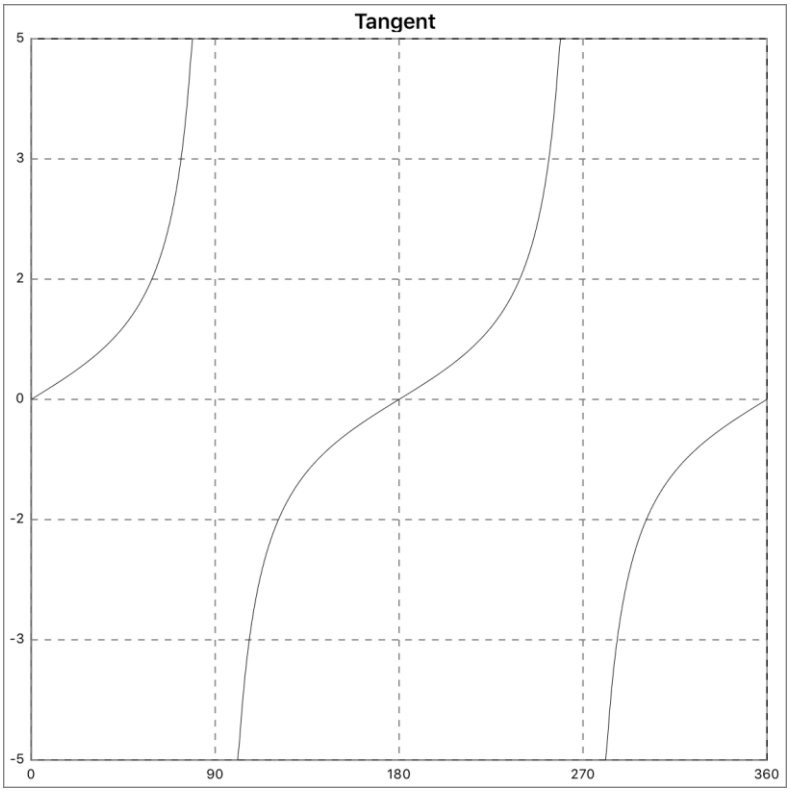

Figure 47 - Tangent Plot

6.1.1.4 Math.acos

`math.acos(x)` – Returns inverse cosine's principal value back to radians

The `x` argument is the principal value. The range of `x` is `-1 ≤ x ≤ 1`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `acos(x)`.

For example:

- `Math.acos(-1)` Returns `3.141593` (i.e., π)
- `Math.acos(1)` Returns `0`

Plot example using range - `-1 ≤ x ≤ 1`:

```
x = [ -1 : 1 : 2 ÷ 300 ]
Utils.PlotIt("Arc-Cosine",              \
             "Plt004_ArcCosine",        \
             Plot.CHART.LINE,           \
             x, Math.ACOS(x),           \
             -1, 1, -PI, PI, 5, 8)
```

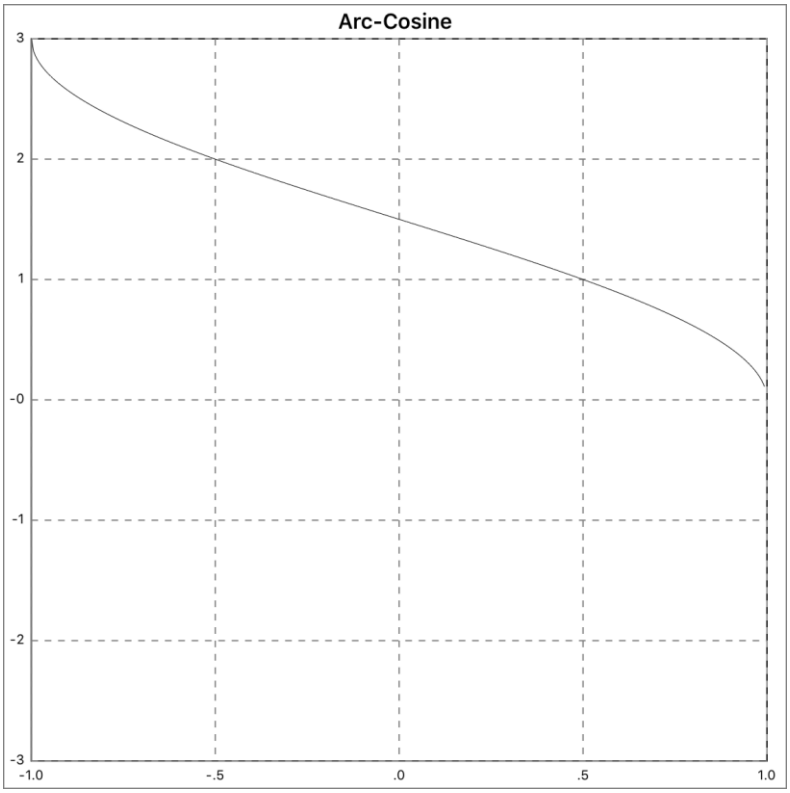

Figure 48 - Arc-Cosine Plot

6.1.1.5 Math.asin

`math.asin(x)` – Returns inverse sine's principal value back to radians

The `x` argument is the principal value. The range of `x` is $-1 \leq x \leq 1$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `asin(x)`.

For example:

- `Math.asin(-1)` Returns `-1.570796` (i.e., $\pi \div 2$)
- `Math.asin(1)` Returns `1.570796` (i.e., $\pi \div 2$)

Plot example using range - $-1 \leq x \leq 1$:

```
x = [ -1 : 1 : 2 ÷ 300 ]
Utils.PlotIt("Arc-Sine",                \
             "Plt005_ArcSinc",          \
             Plot.CHART.LINE,           \
             x, Math.ASIN(x),           \
             -1,1, -PI÷2,PI÷2, 5,7)
```

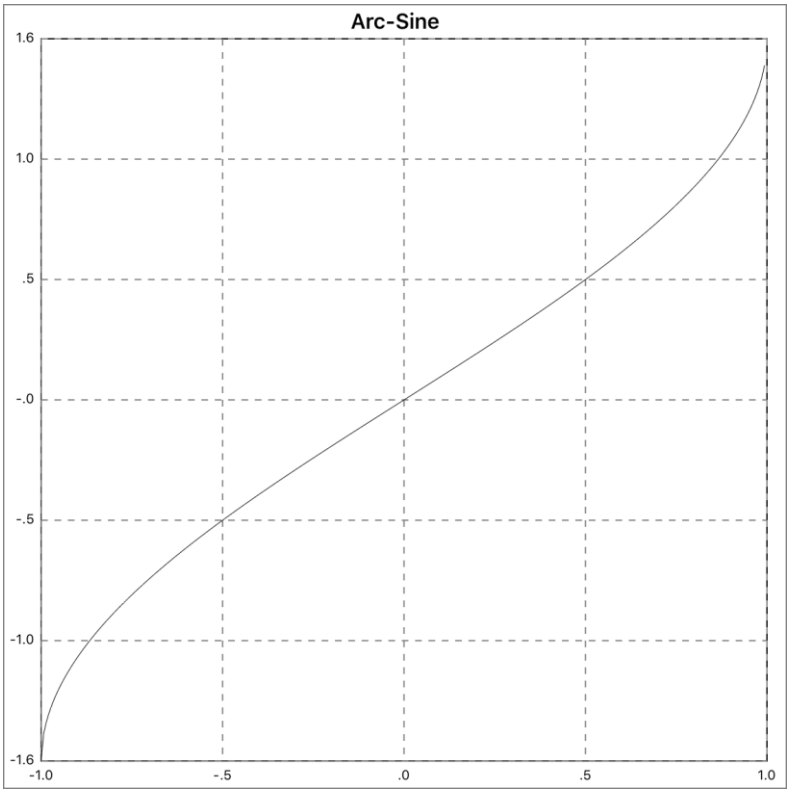

Figure 49 - Arc-Sine Plot

6.1.1.6 Math.atan

`math.atan(x)` – Returns inverse tangent's principal value back to radians

The `x` argument is the principal value. The range of `x` is $-\infty < x < \infty$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `atan(x)`.

For example:

- `Math.atan(-1000000)` Returns `-1.570795` (i.e., ~ $\pi \div 2$)
- `Math.atan(1000000)` Returns `1.570795` (i.e., ~ $\pi \div 2$)

Plot example using range - $-5 \leq x \leq 5$:

```
x = [ -5 : 5 : 10 ÷ 300 ]
Utils.PlotIt("Arc-Tangent",              \
             "Plt006_ArcTangent",        \
             Plot.CHART.LINE,            \
             x, Math.ATAN(x),            \
             -5,5, -PI÷2,PI÷2, 5,7)
```

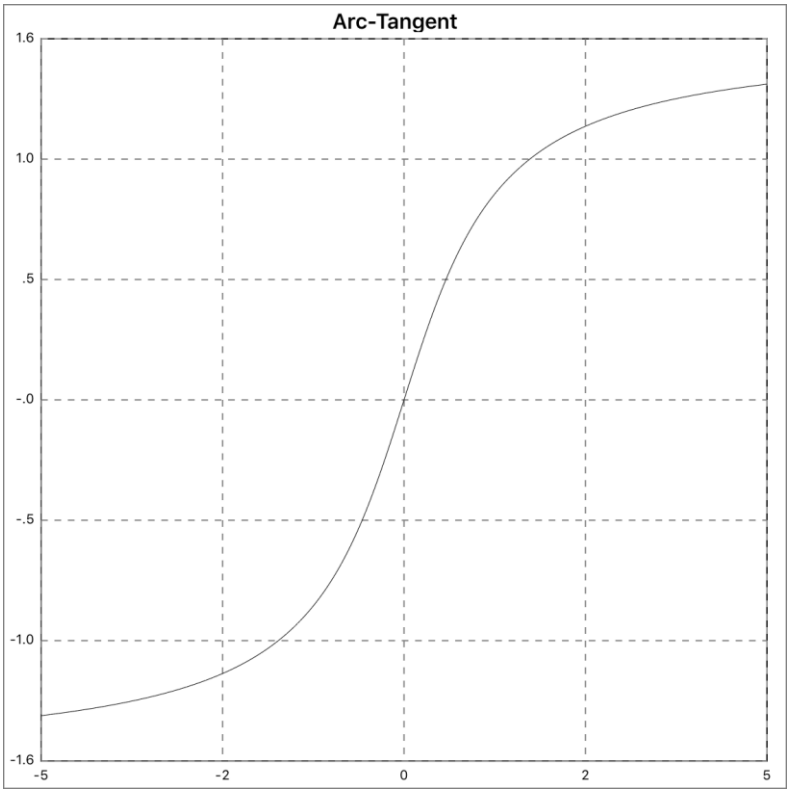

Figure 50 - Arc-Tangent Plot

6.1.1.7 Math.cosh

`math.cosh(x)` – Returns hyperbolic cosine's value

The range of `x` is `−∞ < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `cosh(x)`.

For example:

- `Math.cosh(-2)` Returns `3.762196`
- `Math.cosh(2)` Returns `3.762196`

Plot example using range - `−3 ≤ x ≤ 3`:

```
x = [ -3 : 3 : 6 ÷ 300 ]
Utils.PlotIt("Hyperbolic Cosine",       \
             "Plt007_HypCosine",        \
             Plot.CHART.LINE,           \
             x, Math.COSH(x),           \
             -3, 3, -5, 5, 5, 7)
```

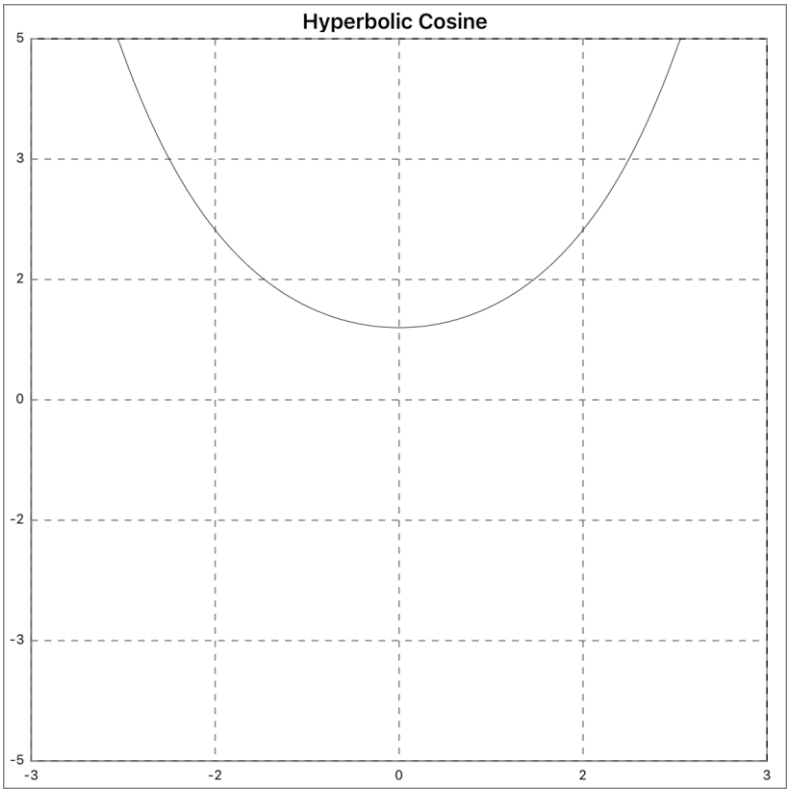

Figure 51 - Hyperbolic Cosine Plot

6.1.1.8 Math.sinh

`math.sinh(x)` – Returns hyperbolic sine's value

The range of `x` is $-\infty < x < \infty$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `sinh(x)`.

For example:

- `Math.sinh(-2)` Returns `-3.62686`
- `Math.sinh(2)` Returns `3.62686`

Plot example using range - $-3 \leq x \leq 3$:

```
x = [ -3 : 3 : 6 ÷ 300 ]
Utils.PlotIt("Hyperbolic Sine",          \
             "Plt008_HypSine",           \
             Plot.CHART.LINE,            \
             x, Math.SINH(x),            \
             -3, 3, -5, 5, 5, 7)
```

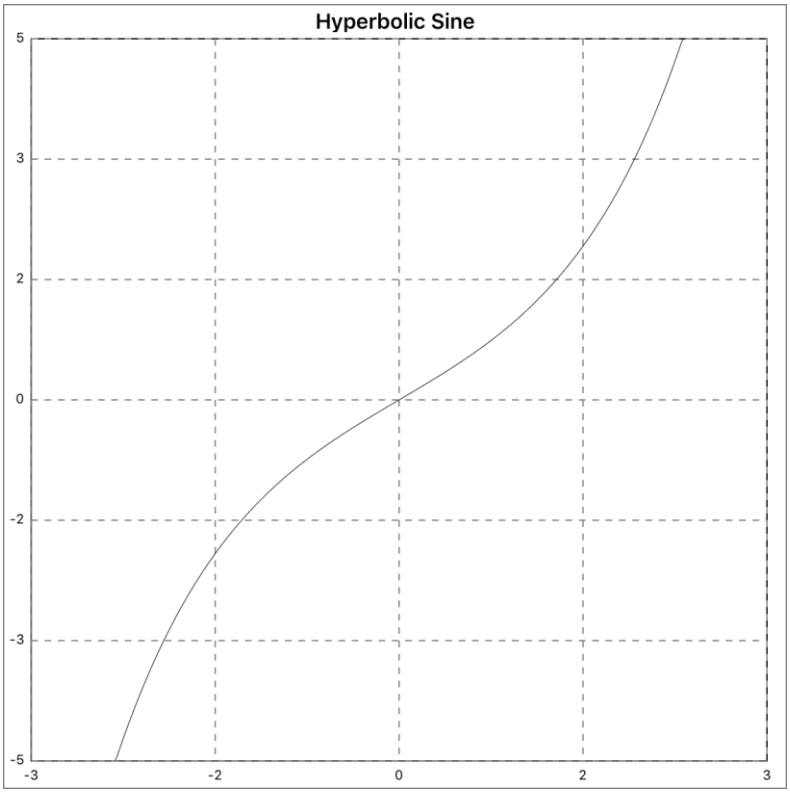

Figure 52 - Hyperbolic Sine Plot

6.1.1.9 Math.tanh

`math.tanh(x)` – Returns hyperbolic tangent's value

The range of `x` is `-∞ < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `tanh(x)`.

For example:

- `Math.tanh(-1.5)` Returns `-0.9051482536448664`
- `Math.tanh(1.5)` Returns `0.9051482536448664`

For example, using range - `-1.5 ≤ x ≤ 1.5`:

```
x = [ -1.5 : 1.5 : 3 ÷ 300 ]
Utils.PlotIt("Hyperbolic Tangent",      \
             "Plt009_HypTangent",       \
             Plot.CHART.LINE,           \
             x, Math.TANH(x),           \
             -1.5, 1.5, -1, 1, 5, 7)
```

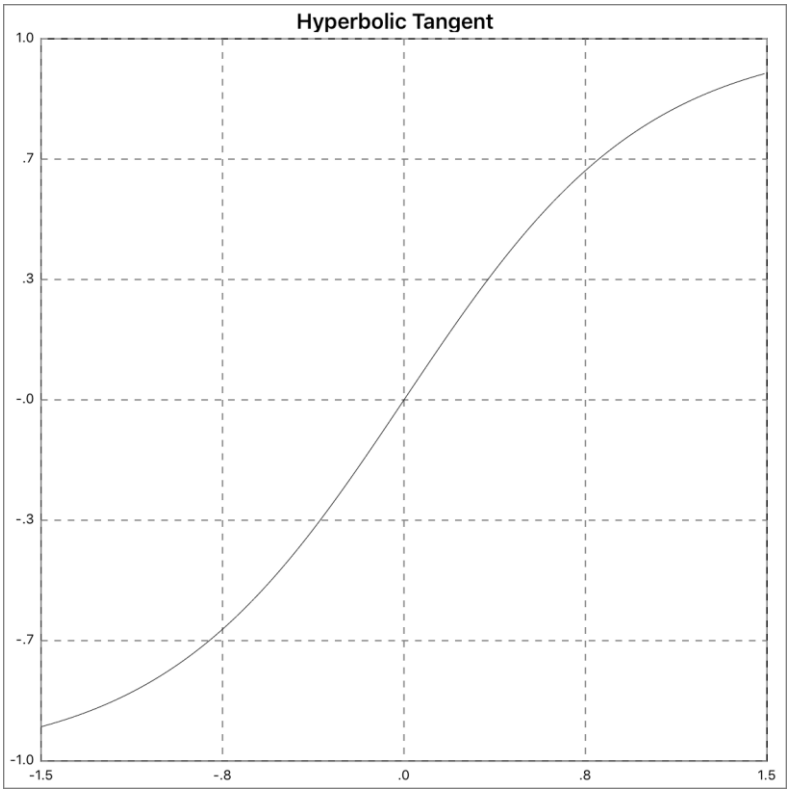

Figure 53 - Hyperbolic Tangent Plot

6.1.1.10 Math.acosh

`math.acosh(x)` – Returns hyperbolic arc-cosine's value

The range of `x` is `1 < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `acosh(x)`.

For example:

- `Math.acosh(1)` Returns `0`
- `Math.acosh(3)` Returns `1.762747`

Plot example using range - `1 ≤ x ≤ 3`:

```
x = [ 1 : 3 : 2 ÷ 300 ]
Utils.PlotIt("Hyperbolic Arc-Cosine",    \
             "Plt010_HypArcCosine",      \
             Plot.CHART.LINE,            \
             x, Math.ACOSH(x),           \
             -3, 3, -5, 5, 5, 7)
```

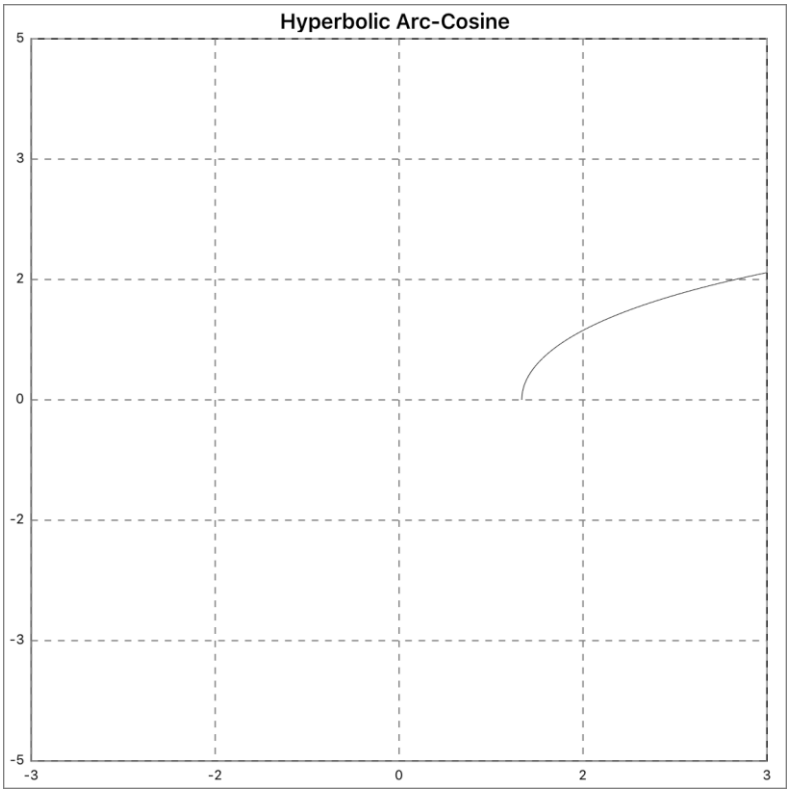

Figure 54 - Hyperbolic Arc-Cosine Plot

6.1.1.11 Math.asinh

`math.asinh(x)` – Returns hyperbolic arc-sine's value

The range of `x` is `-∞ < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `asinh(x)`.

For example:

- `Math.asinh(1)` Returns `-1.818446`
- `Math.asinh(3)` Returns `1.818446`

Plot example using range - `-3 ≤ x ≤ 3`:

```
x = [ -3 : 3 : 6 ÷ 300 ]
Utils.PlotIt("Hyperbolic Arc-Sine",     \
             "Plt011_HypArcSine",       \
             Plot.CHART.LINE,           \
             x, Math.ASINH(x),          \
             -3, 3, -5, 5, 5, 7)
```

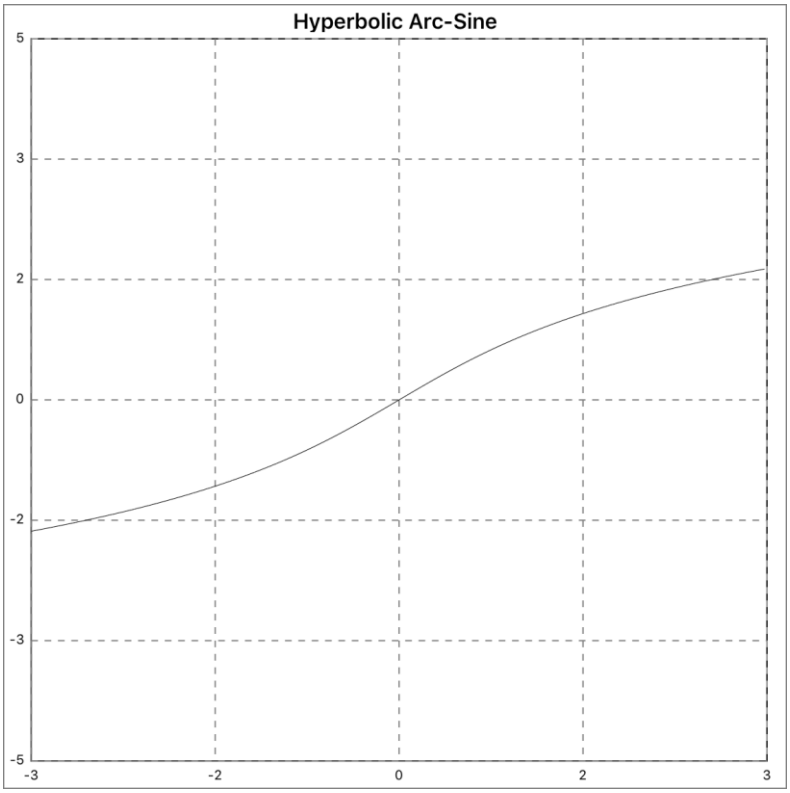

Figure 55 - Hyperbolic Arc-Sine Plot

6.1.1.12 Math.atanh

`math.atanh(x)` – Returns hyperbolic arc-tangent's value

The range of `x` is `-1 < x < 1`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `atanh(x)`.

For example:

- `Math.atanh(-0.5)` Returns `-0.54930614433405489`
- `Math.atanh(0.5)` Returns `0.54930614433405489`

Plot example using range - `-0.99 ≤ x ≤ 0.99`:

```
x = [ -0.99 : 0.99 : (2.0 - 0.01 × 2) ÷ 300 ]
Utils.PlotIt("Hyperbolic Arc-Tangent",  \
             "Plt012_HypArcTangent",    \
             Plot.CHART.LINE,           \
             x, Math.ATANH(x),          \
             -1, 1, -4, 4, 5, 7)
```

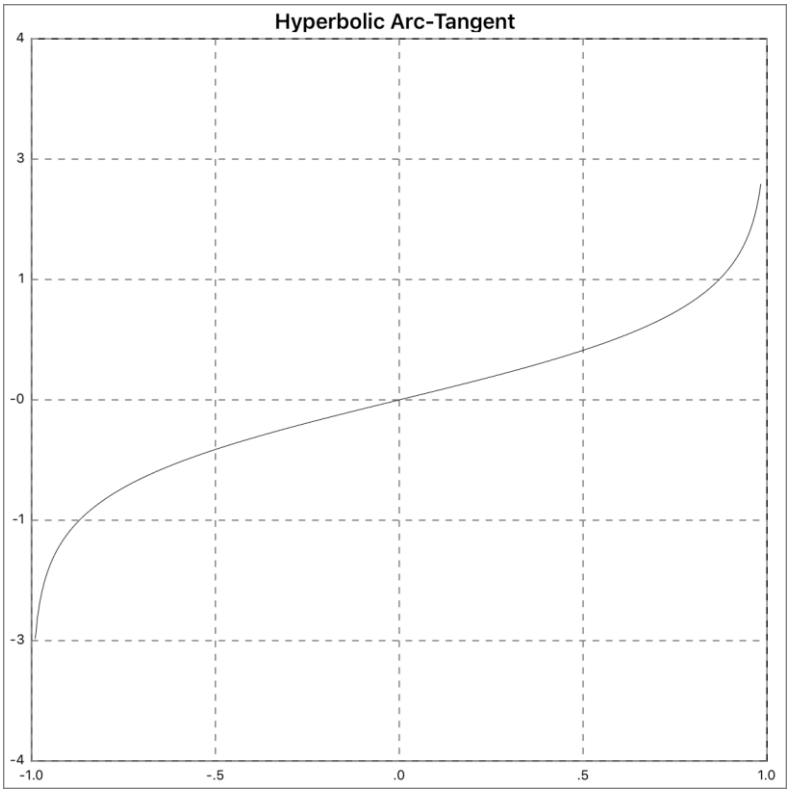

Figure 56 - Hyperbolic Arc-Tangent Plot

6.1.1.13 Math.sec

`math.sec(x)` – Returns secant angle of `x` radians

The `x` argument is the angle expressed in radians. The periodic range of `x` is $-\pi \div 2 < x < \pi \div 2$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `sec(x)`.

For example:

- `Math.sec(0)` Returns `1`
- `Math.sec(PI)` Returns `-1`

Plot example using range - $0° \leq x \leq 360°$:

⇒ NOTE: The ranges $0° \leq x \leq 360°$ must be broken up into three data sets to handle secant's discontinuous boundaries on the $\pi \div 2$ periodical cycles.

```
x1 = [ 0{d}   : 89{d}  : 1{d} ]
x2 = [ 91{d}  : 269{d} : 1{d} ]
x3 = [ 271{d} : 360{d} : 1{d} ]
Utils.Plot3It("Secant",                  \
              "Plt013_Secant",           \
              Plot.CHART.LINE,           \
              x1, Math.SEC(x1),          \
              x2, Math.SEC(x2),          \
```

```
x3, Math.SEC(x3),          \
0{d}, 360{d}, -5, 5, 5, 7)
```

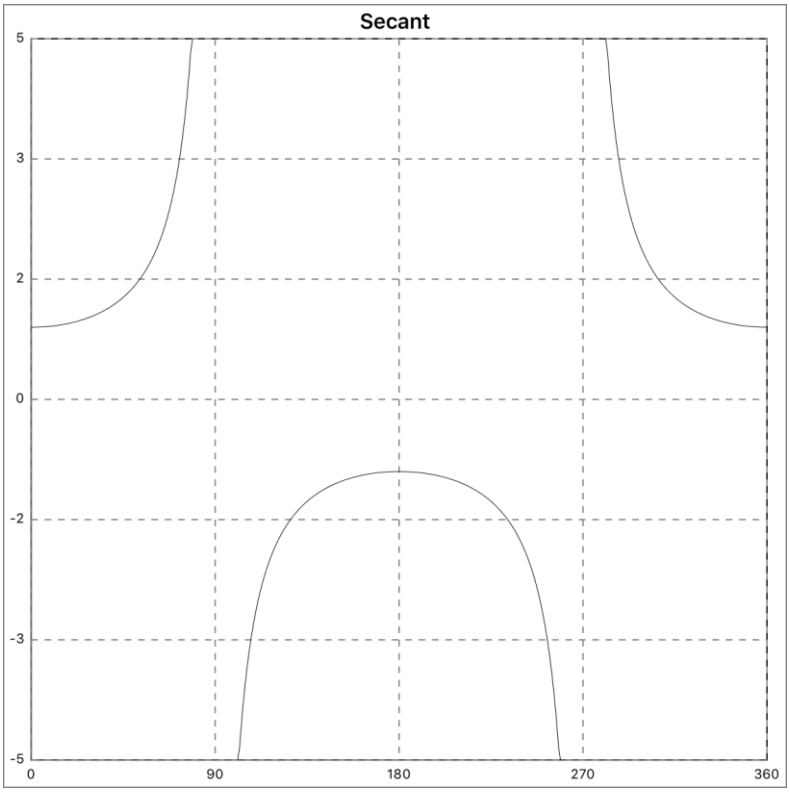

Figure 57 - Secant Plot

6.1.1.14 Math.csc

`math.csc(x)` – Returns cosecant angle of `x` radians

The `x` argument is the angle expressed in radians. The periodic range of `x` is $0 < x < \pi$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `csc(x)`.

For example:

- `Math.csc(PI ÷ 2)` Returns `1`
- `Math.csc(3 × PI ÷ 2)` Returns `−1`

Plot example using range - $0° \leq x \leq 360°$:

⇒ NOTE: The ranges $0° \leq x \leq 360°$ must be broken up into two data sets to handle cosecant's discontinuous boundaries on the π periodical cycles.

```
x1 = [ 1{d}   : 179{d} : 1{d} ]
x2 = [ 181{d} : 359{d} : 1{d} ]
Utils.Plot2It("Cosecant",                \
              "Plt014_Cosecant",         \
              Plot.CHART.LINE,           \
```

```
x1, Math.CSC(x1),          \
x2, Math.CSC(x2),          \
0{d}, 360{d}, -5, 5, 5, 7)
```

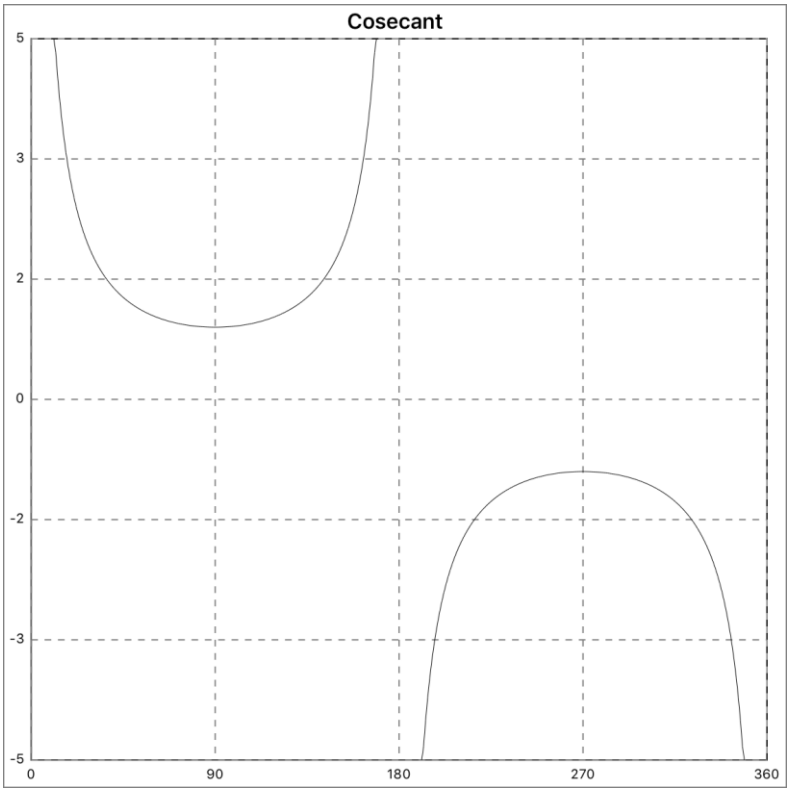

Figure 58 - Cosecant Plot

6.1.1.15 Math.cot

`math.cot(x)` – Returns cotangent angle of `x` radians

The `x` argument is the angle expressed in radians. The periodic range of `x` is $0 < x < \pi$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `cot(x)`.

For example:

- `Math.cot(PI ÷ 2)` Returns `0`
- `Math.cot(3 × PI ÷ 2)` Returns `0`

Plot example using range - $0° \leq x \leq 360°$:

⇒ NOTE: The ranges $0° \leq x \leq 360°$ must be broken up into two data sets to handle cotangent's discontinuous boundaries on the π periodical cycles.

```
x1 = [ 1{d}   : 179{d} : 1{d} ]
x2 = [ 181{d} : 359{d} : 1{d} ]
Utils.Plot2It("Cotangent",              \
              "Plt015_Cotangent",       \
```

```
Plot.CHART.LINE,          \
x1, Math.COT(x1),         \
x2, Math.COT(x2),         \
0{d}, 360{d}, -5, 5, 5, 7)
```

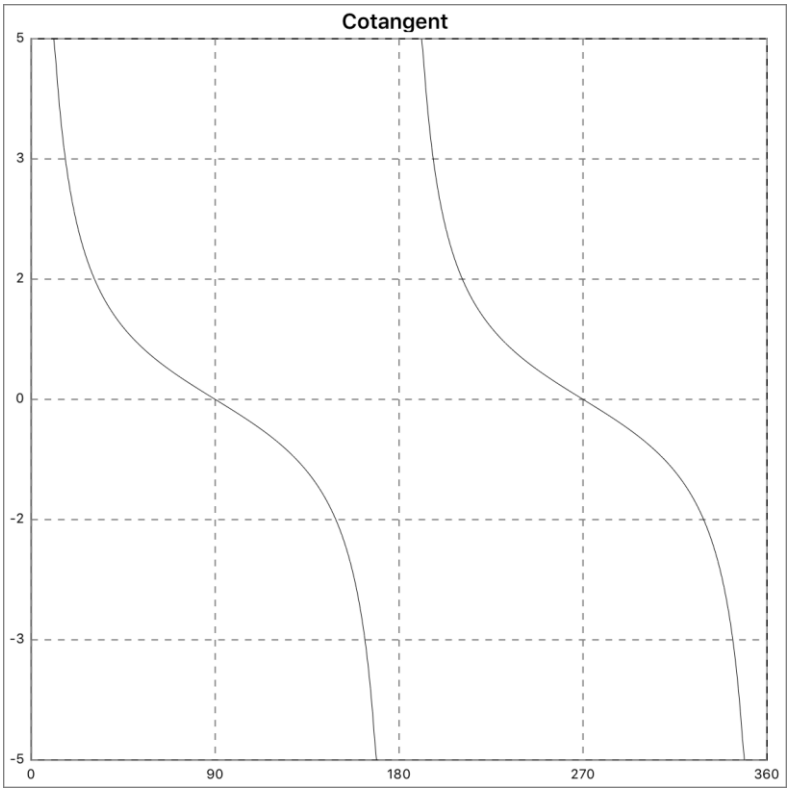

Figure 59 - Cotangent Plot

6.1.1.16 Math.asec

`math.asec(x)` – Returns inverse secant's principal value back to radians

The `x` argument is the principal value. The ranges of `x` are $-\infty < x \leq -1$ and $1 \leq x < \infty$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `asec(x)`.

For example:

- `Math.asec(-1)` Returns `3.141593` (i.e., π)
- `Math.asec(1)` Returns `0`

Plot example using range - $-8 \leq x \leq 8$:

⇒ NOTE: The ranges must be broken up into two data sets to handle arc-secant's discontinuous boundaries.

```
x1 = [ -5 : -1 : 8 ÷ 300 ]
x2 = [  1 :  5 : 8 ÷ 300 ]
Utils.Plot2It("Arc-Secant",          \
```

```
"Plt016_ArcSecant",        \
Plot.CHART.LINE,           \
x1, Math.ASEC(x1),         \
x2, Math.ASEC(x2),         \
-5, 5, -PI, PI, 5, 7)
```

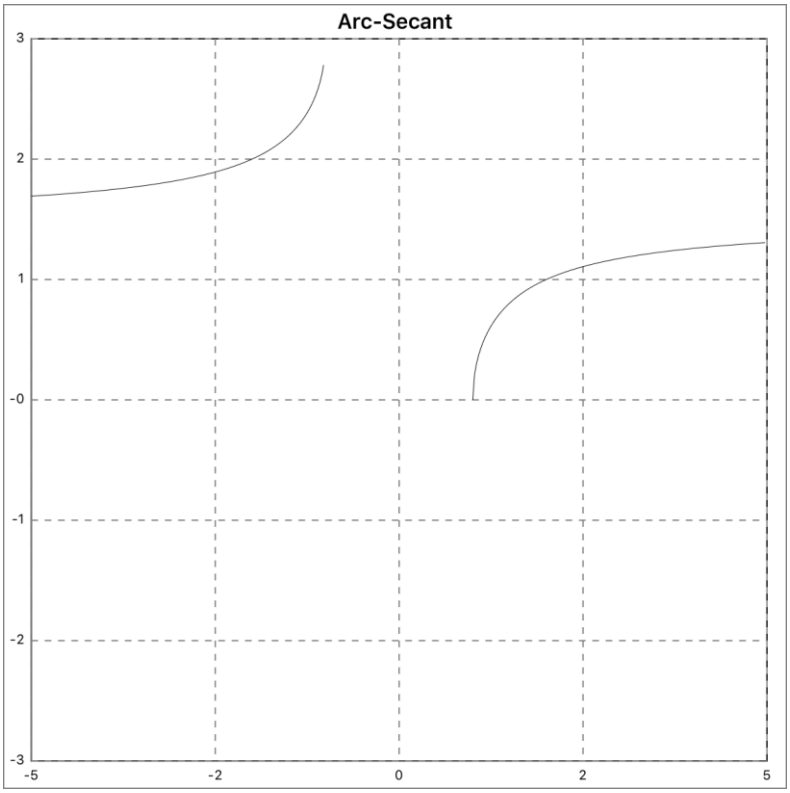

Figure 60 - Arc-Secant Plot

6.1.1.17 Math.acsc

`math.acsc(x)` – Returns inverse cosecant's principal value back to radians

The `x` argument is the principal value. The ranges of `x` are $-\infty < x \leq -1$ and $1 \leq x < \infty$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `acsc(x)`.

For example:

- `Math.acsc(-1)` Returns -1.570796 (i.e., $-\pi \div 2$)
- `Math.acsc(1)` Returns 1.570796 (i.e., $\pi \div 2$)

Plot example using range - $-8 \leq x \leq 8$:

⇒ NOTE: The ranges must be broken up into two data sets to handle arc-cosecant's discontinuous boundaries.

```
x1 = [ -5 : -1 : 8 ÷ 300 ]
x2 = [  1 :  5 : 8 ÷ 300 ]
```

```
Utils.Plot2It("Arc-Cosecant",           \
              "Plt017_ArcCosecant",     \
              Plot.CHART.LINE,          \
              x1, Math.ACSC(x1),        \
              x2, Math.ACSC(x2),        \
              -5, 5, -PI, PI, 5, 7)
```

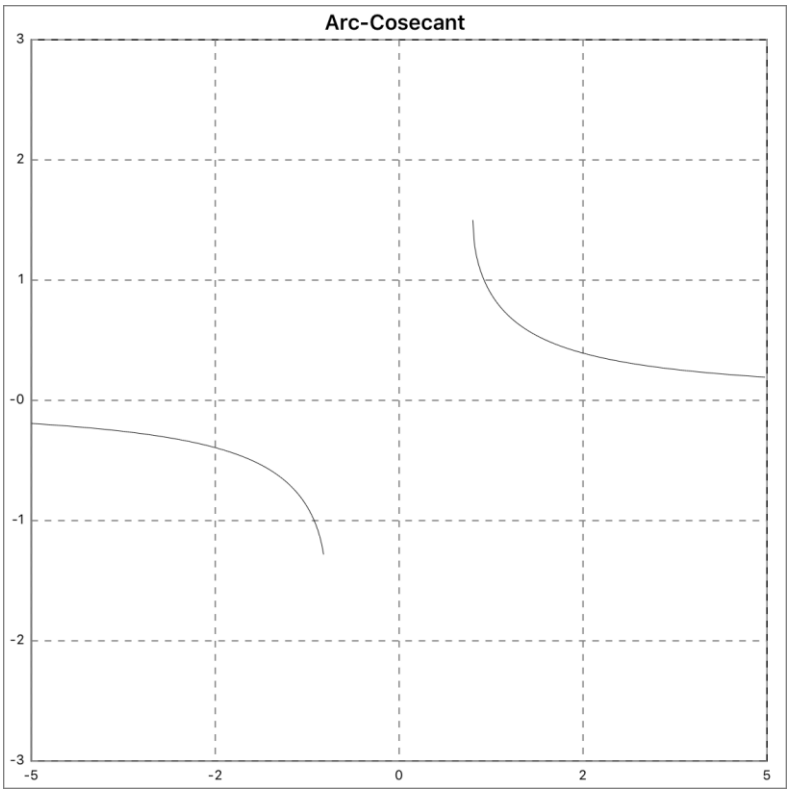

Figure 61 - Arc-Cosecant Plot

6.1.1.18 Math.acot

`math.acot(x)` – Returns inverse cotangent's principal value back to radians

The `x` argument is the principal value. The ranges of `x` are `-∞ < x < 0` and `0 < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `acot(x)`.

For example:

- `Math.acot(-5)` Returns `-0.19739555984988078`
- `Math.acot(5)` Returns `0.19739555984988078`

Plot example using range - `-5 ≤ x ≤ 5`:

⇒ NOTE: The ranges must be broken up into two data sets to handle arc-cotangent's discontinuous boundaries.

```
x1 = [ -5   : -0.1 : (10.0 - 0.1 × 2) ÷ 300 ]
x2 = [  0.1 :  5   : (10.0 - 0.1 × 2) ÷ 300 ]
Utils.Plot2It("Arc-Cotangent",          \
              "Plt018_ArcCotangent",    \
              Plot.CHART.LINE,          \
              x1, Math.ACOT(x1),        \
              x2, Math.ACOT(x2),        \
              -5, 5, -PI ÷ 2, PI ÷ 2, 5, 7)
```

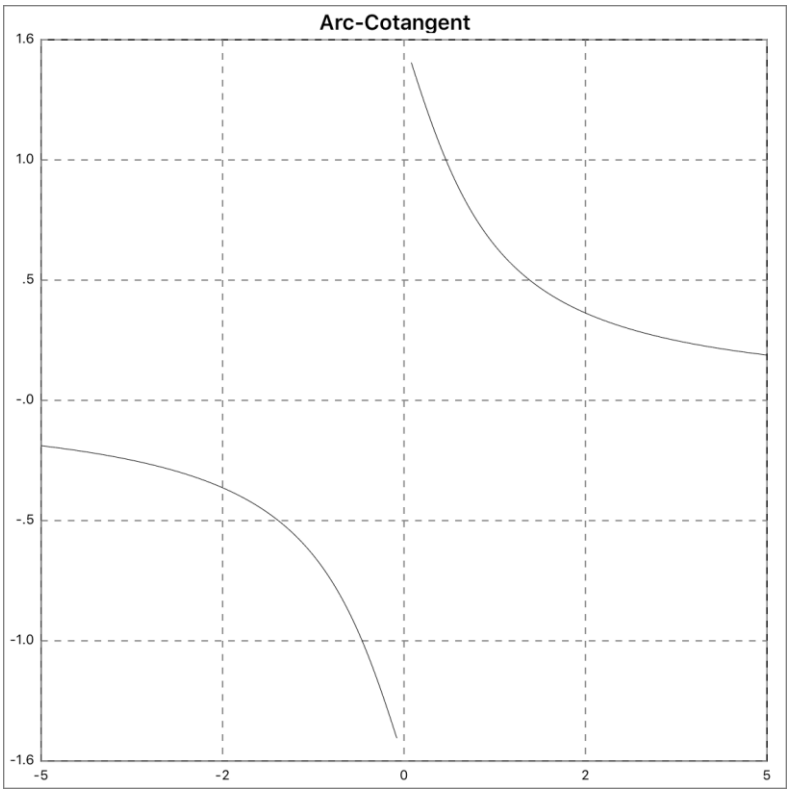

Figure 62 - Arc-Cotangent Plot

6.1.1.19 Math.sech

`math.sech(x)` – Returns hyperbolic secant's value

The range of `x` is `-∞ < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `sech(x)`.

For example:

- `Math.sech(-5)` Returns `0.01347528222130456`
- `Math.sech(5)` Returns `0.01347528222130456`

Plot example using range - `-5 ≤ x ≤ 5`:

```
x = [ -5   : 5  : 10 ÷ 300 ]
Utils.PlotIt("Hyperbolic Secant",       \
             "Plt019_HypSecant",        \
```

```
Plot.CHART.LINE,                \
x, Math.SECH(x),                \
-5, 5, -1, 1, 5, 7)
```

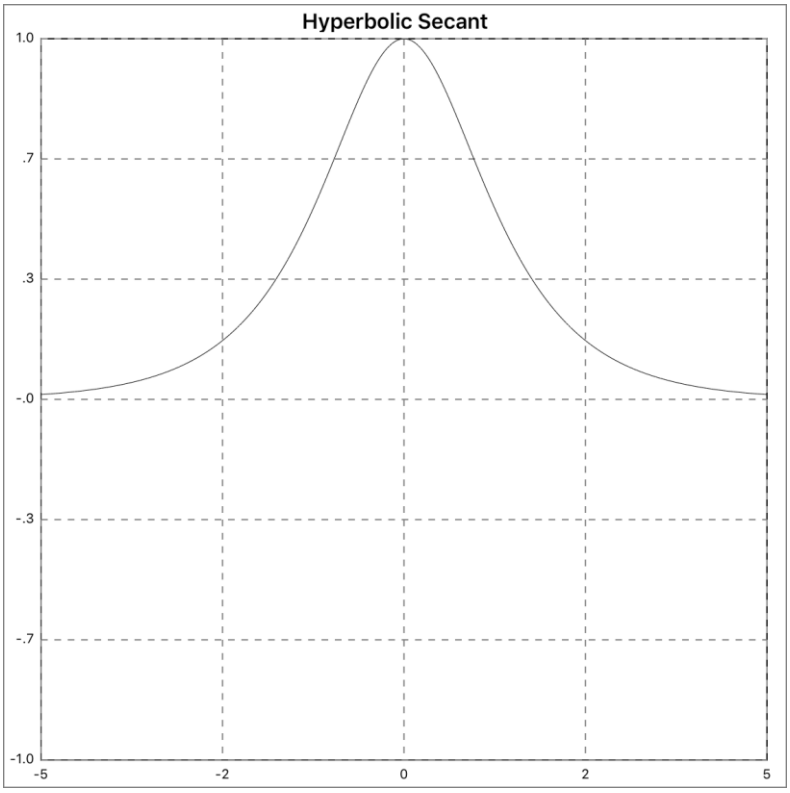

Figure 63 - Hyperbolic Secant Plot

6.1.1.20 Math.csch

`math.csch(x)` – Returns hyperbolic cosecant's value

The range of `x` is $-\infty < x < 0$ and $0 < x < \infty$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `csch(x)`.

For example:

- `Math.csch(-5)` Returns `-0.01347650583058909`
- `Math.csch(5)` Returns `0.01347650583058909`

Plot example using range - $-5 \leq x \leq 5$:

⇒ NOTE: The ranges must be broken up into two data sets to handle hyperbolic cosecant's discontinuous boundary at zero.

```
x1 = [ -5  : -0.1 : (10.0 - 0.1 × 2) ÷ 300 ]
x2 = [ 0.1 :  5   : (10.0 - 0.1 × 2) ÷ 300 ]
Utils.Plot2It("Hyperbolic Cosecant",    \
              "Plt020_HypCosecant",     \
              Plot.CHART.LINE,          \
```

```
x1, Math.CSCH(x1),          \
x2, Math.CSCH(x2),          \
-5, 5, -5, 5, 5, 7)
```

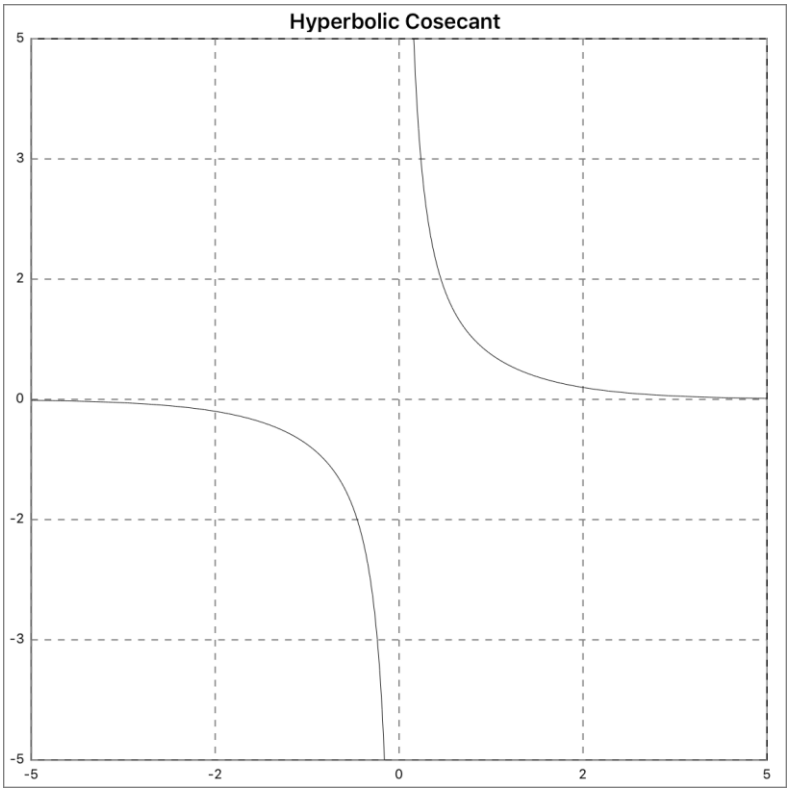

Figure 64 - Hyperbolic Cosecant Plot

6.1.1.21 Math.coth

`math.coth(x)` – Returns hyperbolic cotangent value

The range of `x` is `-∞ < x < 0` and `0 < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `coth(x)`.

For example:

- `Math.coth(-5)` Returns `-1.000091`
- `Math.coth(5)` Returns `1.000091`

Plot example using range - `-5 ≤ x ≤ 5`:

⇒ NOTE: The ranges must be broken up into two data sets to handle hyperbolic cotangent's discontinuous boundary at zero.

```
x1 = [ -5  : -0.1 : (10.0 - 0.1 × 2) ÷ 300 ]
x2 = [ 0.1 :  5   : (10.0 - 0.1 × 2) ÷ 300 ]
Utils.Plot2It("Hyperbolic Cotangent",   \
              "Plt021_HypContangent",   \
              Plot.CHART.LINE,          \
```

```
    x1, Math.COTH(x1),          \
    x2, Math.COTH(x2),          \
    -5, 5, -5, 5, 5, 7)
```

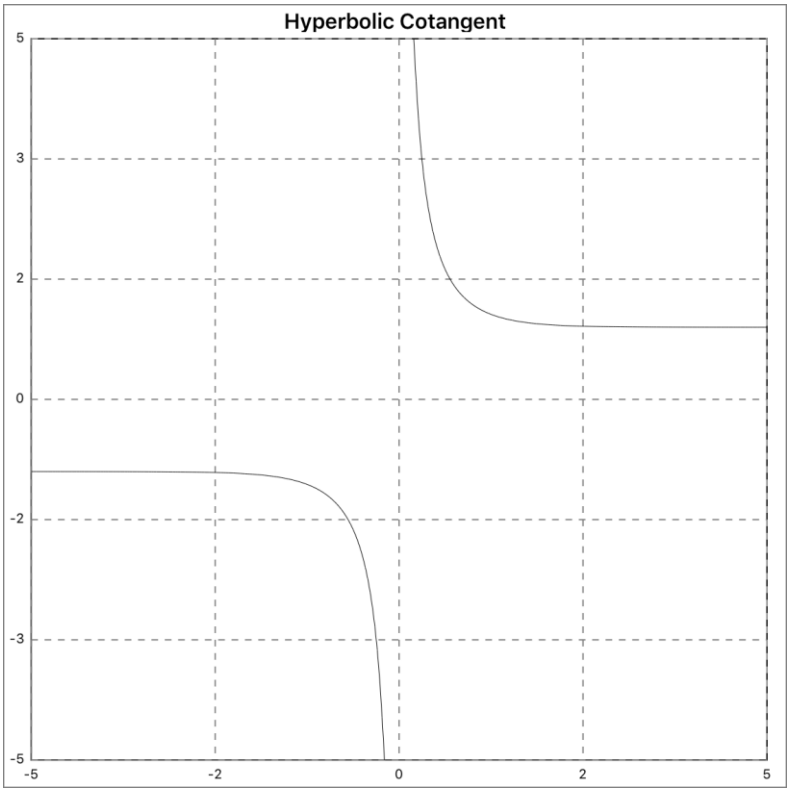

Figure 65 - Hyperbolic Cotangent Plot

6.1.2 Exponential

6.1.2.1 Math.exp

`math.exp(x)` – Returns natural exponential value

The range of `x` is $-\infty < x < \infty$.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `exp(x)`.

⇒ LINK: https://en.wikipedia.org/wiki/Exponential_function

For example:

- `Math.exp(-1)` Returns `0.36787944117144233`
- `Math.exp(0)` Returns `1`

Plot example using range - $-4 \leq x \leq 2$:

```
    x = [ -4 : 2 : 6 ÷ 300 ]
    Utils.PlotIt("Natural Exponential",      \
                 "Plt022_Exponential",       \
```

```
                Plot.CHART.LINE,            \
                x, Math.EXP(x),             \
                -4, 2, -1, 5, 0, 0)
```

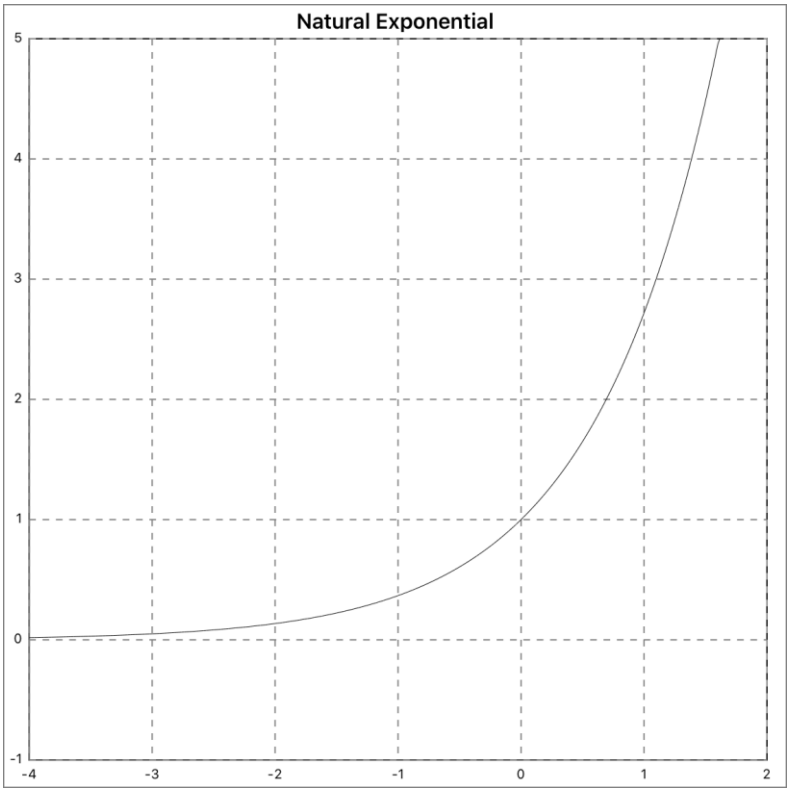

Figure 66 - Natural Exponential Plot

6.1.2.2 Math.log

`math.log(x)` – Returns logarithm base `2` value

The range of `x` is `0 < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `log(x)`.

⇒ LINK: https://en.wikipedia.org/wiki/Logarithm

For example:

- `Math.log(1)` Returns `0`
- `Math.log(2)` Returns `0.69314718055994529`

Plot example using range - `-0.1 ≤ x ≤ 5`:

```
x = [ 0.1 : 5 : (5.0 - 0.1) ÷ 300 ]
Utils.PlotIt("Logarithm 2",                 \
             "Plt023_Logarithm",            \
             Plot.CHART.LINE,               \
             x, Math.LOG(x),                \
             -5, 5, -2, 2, 0, 0)
```

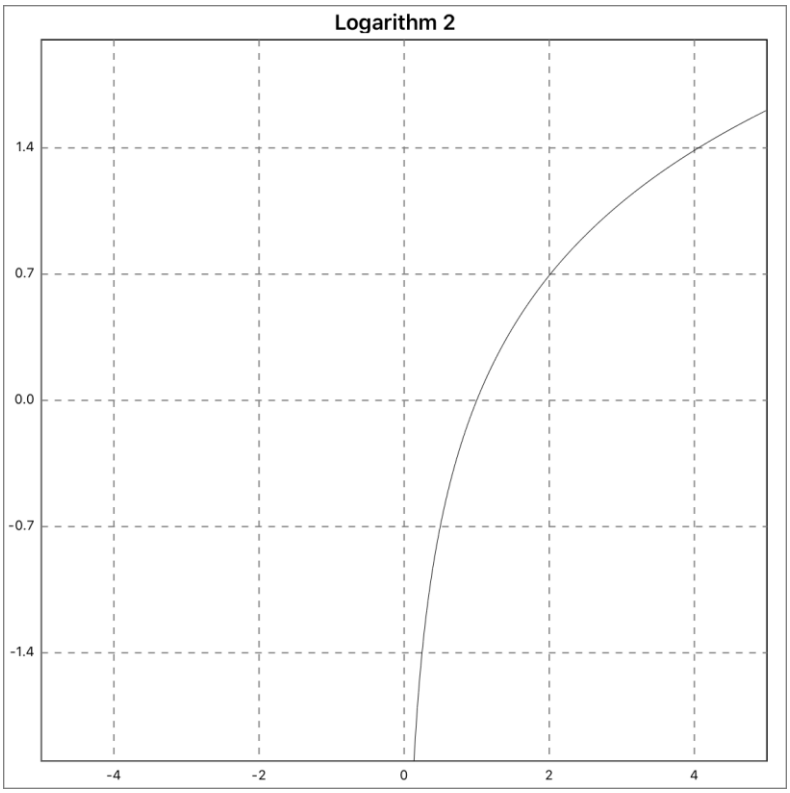

Figure 67 - Logarithm 2 Plot

6.1.2.3 Math.log10

`math.log10(x)` – Returns logarithm base `10` value

The range of `x` is `0 < x < ∞`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `log10(x)`.

For example:

- `Math.log10(1)` Returns `0`
- `Math.log10(2)` Returns `0.3010299956639812`

Plot example using range - `-0.1 ≤ x ≤ 5`:

```
x = [ 0.1 : 5 : (5.0 - 0.1) ÷ 300 ]
Utils.PlotIt("Logarithm 10",              \
             "Plt024_Logarithm10",        \
             Plot.CHART.LINE,             \
             x, Math.LOG10(x),            \
             -5, 5, -2, 2, 0, 0)
```

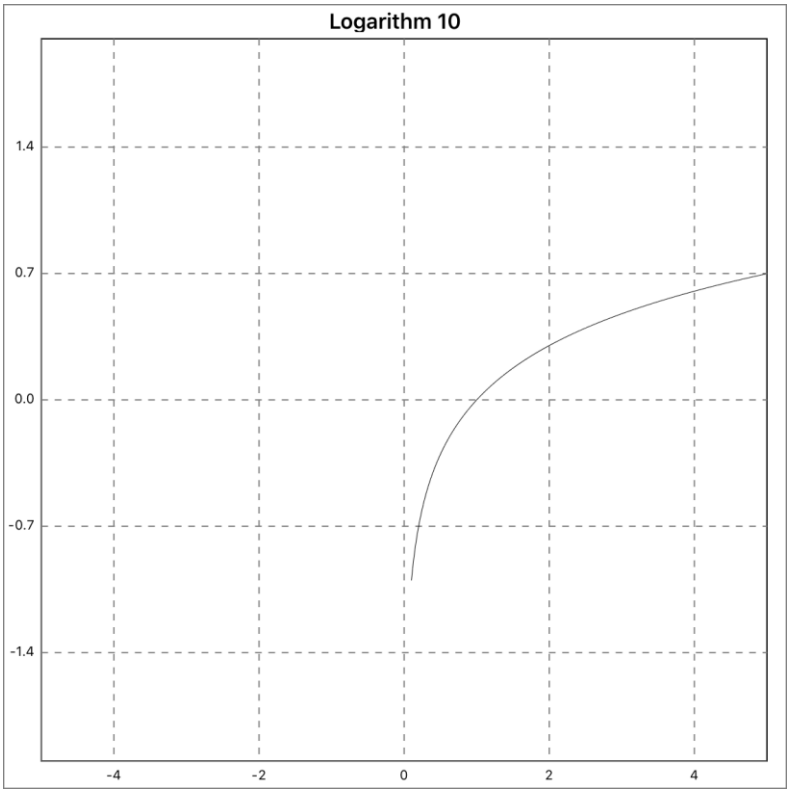

Figure 68 - Logarithm 10 Plot

6.1.3 Miscellaneous

6.1.3.1 Math.abs

`abs(x)` – Returns absolute value

The `x` argument is converted from negative value to positive value.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `abs(x)`.

For example:

- `Math.abs(-1)` Returns `1`
- `Math.abs(-1.5)` Returns `1.5`
- `Math.abs(-1.5{ft})` Returns `1.5{ft}`
- `Math.abs(-1.5 + 1.5i)` Returns `2.12132 + 0i`
- `Math.abs("-1.5")` Returns 1.5
- `Math.abs([-1.5,-2.5,-3.5])` Returns `1.5`, `2.5`, and `3.5`

6.1.3.2 Math.pow

`pow(x, p)` – Returns multiplicity of value to its power

The `x` argument is multiplied to the number of `p` argument.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `pow(x, p)`.

For example:

- `Math.pow(5, 2` Returns `25`
- `Math.pow(5.5, 2.5)` Returns `70.942538`
- `Math.pow([2, 4, 8], 2)` Returns `4`, `16` and `64`
- `Math.pow([2, 4, 8], 2.5)` Returns `5.65685`, `32`, and `181.019`
- `Math.pow([[2,4],[8,16]], 2)` Returns `4` and `16`; `64` and `256`

6.1.3.3 Math.sqrt

`sqrt(x)` – Returns square root value

The square root of the argument's `x` is calculated.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `sqrt(x)`.

For example:

- `Math.sqrt(2)` Returns `1.414214`
- `Math.sqrt([2, 3])` Returns `1.414214` and `1.73205`

6.1.3.4 Math.srand

`srand(x)` – Initializes random number generator

The `x` argument is random number generator seed used to initialize for repeatable results in using in subsequent `rand` function calls.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `srand(x)`.

For example:

- `Math.srand(31459)` Initialize random number generation for following `rand` function calls

6.1.3.5 Math.rand

`rand()` – Returns randomly generated number
`rand(i)` – Returns one dimension of randomly generated numbers
`rand(i, j)` – Returns two dimensions of randomly generated numbers
`rand(i, j, k)` – Returns three dimensions of randomly generated numbers
`rand(i, j, k, m)` – Returns four dimensions of randomly generated numbers

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `rand(x, p)`.

For example:

- `Math.srand(31459)` — Initialize random number generation for following `rand` function calls

- `Math.rand()` — Returns `528731413`
- `Math.rand(2)` — Returns `101527005` and `1262357317`
- `Math.rand(2, 2)` — Returns the following array:

```
[000,000]:          1448478106
[000,001]:          696905150
[001,000]:          509045312
[001,001]:          2097192783
```

- `Math.rand(2, 2, 2)` — Returns the following array:

```
[000,000,000]:      870005670
[000,000,001]:      2116626914
[000,001,000]:      1081931043
[000,001,001]:      1271000552
[001,000,000]:      686440755
[001,000,001]:      727617601
[001,001,000]:      1297133989
[001,001,001]:      1824452426
```

- `Math.rand(2, 2, 2, 2)` — Returns the following array:

```
[000,000,000,000]: 1800411916
[000,000,000,001]: 1478485982
[000,000,001,000]: 380620037
[000,000,001,001]: 1874661093
[000,001,000,000]: 1696404914
[000,001,000,001]: 1484492026
[000,001,001,000]: 392470136
[000,001,001,001]: 1323295815
[001,000,000,000]: 1292114373
[001,000,000,001]: 1211628547
[001,000,001,000]: 1401048575
[001,000,001,001]: 265210670
[001,001,000,000]: 1367163165
[001,001,000,001]: 1983774902
[001,001,001,000]: 1621158239
[001,001,001,001]: 1681493384
```

6.1.3.6 Math.infinity

`Math.infinity()` – Returns floating-point infinity value

Infinitely large floating-point number representation. Typically occurs on numerical computation overflow.

For example:

- `Math.infinity()` — Returns `INF`

6.1.3.7 Math.linear_spacing

`Math.linear_spacing(a, b)` – Returns vector of evenly spaced values
`Math.linear_spacing(a, b, n)`

The `a` and `b` arguments are the starting and ending ranges of the generated evenly spaced values. The `n` argument is optional number of spacing values to generate. The default value is `100`.

⇒ NOTE: You can use the alternate function name: `Math.linspace(a, b, n)`.

For example:

- `Math.linear_spacing(1, 2, 5)` Returns `1`, `1.25`, `1.5`, `1.75`, and `2`

6.1.4 Range

6.1.4.1 Math.ceil

`Math.ceil(x)` – Returns rounded-up integral value of x

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `ceil(x)`.

For example:

- `Math.ceil(2.1)` Returns `3`
- `Math.ceil(2.5)` Returns `3`
- `Math.ceil(2.9)` Returns `3`
- `Math.ceil(-2.1)` Returns `-2`
- `Math.ceil(-2.5)` Returns `-2`
- `Math.ceil(-2.9)` Returns `-2`
- `Math.ceil([2.1,2.5,2.9])` Returns `3`, `3`, and `3`

6.1.4.2 Math.floor

`Math.floor(x)` – Returns rounded-down integral value of x

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `floor(x)`.

For example:

- `Math.floor(2.1)` Returns `2`
- `Math.floor(2.5)` Returns `2`
- `Math.floor(2.9)` Returns `2`
- `Math.floor(-2.1)` Returns `-3`
- `Math.floor(-2.5)` Returns `-3`
- `Math.floor(-2.9)` Returns `-3`

- `Math.floor([-2.1,-2.5,-2.9])` Returns `-3`, `-3`, and `-3`

6.1.4.3 Math.round

`Math.round(x)` – Returns nearest rounded integral value of `x`

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `round(x)`.

For example:

- `Math.round(2.1)` Returns `2`
- `Math.round(2.5)` Returns `3`
- `Math.round(2.9)` Returns `3`
- `Math.round(-2.1)` Returns `-2`
- `Math.round(-2.5)` Returns `-3`
- `Math.round(-2.9)` Returns `-3`
- `Math.round([-2.1,-2.5,-2.9])` Returns `-2`, `-3`, and `-3`

6.1.4.4 Math.min

`Math.min(`x_1`, `x_2`, ... `x_n`)` – Returns maximum value of `x`'s

The minimum value of x_1, x_2, ... x_n is returned. If `x` is a one- or multi-dimension array, it returns the minimum array found in the list of arguments by finding its minimum value in each array to be compared against to other arrays.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `min(x)`.

For example:

- `Math.min(1, 2)` Returns `1`
- `Math.min(1, 2, 3)` Returns `1`
- `Math.min([1, 2, 3])` Returns 1
- `Math.min([[1, 2, 3], \` Returns `1`, `2`, and `3`
  `[4, 5, 6]])`

6.1.4.5 Math.max

`Math.max(`x_1`, `x_2`, ... `x_n`)` – Returns maximum value of `x`'s

The maximum value of x_1, x_2, ... x_n is returned. If `x` is a one- or multi-dimension array, it returns the maximum array found in the list of arguments by finding its maximum value in each array to be compared to other arrays.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `max(x)`.

For example:

- `Math.max(1, 2)` Returns `3`
- `Math.max(1, 2, 3)` Returns `3`
- `Math.max([1, 2, 3])` Returns `3`
- `Math.max([[1, 2, 3], \ [4, 5, 6]])` Returns `4`, `5`, and `6`

6.1.5 Complex

6.1.5.1 Math.conj

`Math.conj(x)` – Returns the conjugate of the complex value of `x`

The conjugate of a complex number changes the imaginary number's sign from positive to negative or negative to positive.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `conj(x)`.

For example:

- `Math.conj(2 + 5i)` Returns `2 + -5i`
- `Math.conj(2 + -5i)` Returns `2 + 5i`
- `Math.conj([2+5i, 2+-5i])` Returns `2 + -5i` and `2 + 5i`

6.1.5.2 Math.norm

`Math.norm(x)` – Returns the norm value of the complex value of `x`

The norm value of a complex number is squared magnitude of the real part squared plus imaginary part squared.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `norm(x)`.

For example:

- `Math.norm(2 + 2i)` Returns `8`
- `Math.norm(-2 + 2i)` Returns `8`
- `Math.norm(2 + -2i` Returns `8`
- `Math.norm(-2 + -2i)` Returns `8`
- `Math.norm([2+2i, 3+4i])` Returns 8 and `25`

6.1.5.3 Math.proj

`Math.proj(x)` – Returns the projection of the complex `x` value onto the Riemann sphere

Find the projection of complex number `x` argument onto the Riemann sphere.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `proj(x)`.

For example:

- `Math.proj(2 + 2i)` Returns `2 + 2i`
- `Math.proj(-2 + 2i)` Returns `2 + 2i`
- `Math.proj(2 + -2i)` Returns `2 + 2i`
- `Math.proj(-2 + -2i)` Returns `2 + 2i`
- `Math.proj([2 + 2i, 3 + 4i])` Returns `2 + 2i` and `3 + 4i`
- `Math.proj(toComplex(-Math.infinity(), 2))` Returns `INF + 0i`
- `Math.proj(toComplex(2, -Math.infinity()))` Returns `INF + -0i`

6.1.6 Summary

6.1.6.1 Math.sum

`sum(x)` – Returns summation of all the values

The `x` argument is either one or two dimensions of values to compute the addition of all values in the vector.

If it is a one-dimension array, all the values are summed up to a scalar value.

If it is a two-dimension array, each column will be summed up into a one-dimension array.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `sum(x)`.

For example:

- `Math.sum(1)` Returns `1`
- `Math.sum([1, 2, 3])` Returns `6`
- `Math.sum([[1,4], [2,5], [3,6]])` Returns `5`, `7`, and `8`

6.1.6.2 Math.diff

`diff(x)` – Returns the differences of all the values

The `x` argument is either one or two dimensions of values to compute the differences between values in the vector.

If it is a one-dimension array, the first value is subtracted by the second value, and the result of that subtraction is subtracted by the third value, etc. The result will return a scalar differences value.

If it is a two-dimension array, each column will compute the differences within that column, and the result will return a one-dimension array of differences.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `diff(x)`.

For example:

- `Math.diff(1)` Returns `1`

- `Math.diff([1, 2, 3])` Returns `-4`
- `Math.diff([[1,4], [2,5], [3,6]])` Returns `-4` and `-7`

6.1.6.3 Math.prod

`prod(x)` – Returns multiplication of all the values

The `x` argument is either one or two dimensions of values to compute the multiplication of all values in the vector.

If it is a one-dimension array, all the values are multiplied into a scalar value.

If it is a two-dimension array, each column will be multiplied into a one-dimension array.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `prod(x)`.

For example:

- `Math.prod(1)` Returns `1`
- `Math.prod([1, 2, 3])` Returns `6`
- `Math.prod([[1,4], [2,5], [3,6]])` Returns `4`, `10`, and `18`

6.1.6.4 Math.cummin

`Math.cummin(x)` – Returns the cumulative minimum values in the `x` array

Each element in the `x` array is populated with the current minimum value within the vector.

If it is a two-dimension array, each column vector will be filled with its cumulative minimum values.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `cummin(x)`.

For example:

- `Math.cummin([3, 5, 2])` Returns `3`, `3`, and `2`
- `Math.cummin([8,9,1,10,6,1,3,6,10,10])` Returns `8`, `8`, `1`, `1`, `1`, `1`, `1`, `1`, `1`, `1`
- `Math.cummin([[3,5,2],[1,6,3],[7,8,1]])` Returns `3`, `5`, `2`; `1`, `5`, 2; and `1`, `5`, `1`

6.1.6.5 Math.cummax

`Math.cummax(x)` – Returns the cumulative maximum values in the `x` array

Each element in the `x` array is populated with the current maximum value within the vector.

If it is a two-dimension array, each column vector will be filled with its cumulative maximum values.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `cummax(x)`.

For example:

- `Math.cummax([3, 1, 7` Returns `3`, `3`, and `7`
- `Math.cummax([9,10,2,10,7,1,3,6,10,10])` Returns `9`,`10`,`10`,`10`,`10`,`10`,`10`,`10`,`10`,`10`
- `Math.cummax([9,10,2,10,7,1,3,6,11,10])` Returns `9`,`10`,`10`,`10`,`10`,`10`,`10`,`10`,`11`,`11`
- `Math.cummax([9,10,2,10,7,1,3,6,11,15])` Returns `9`,`10`,`10`,`10`,`10`,`10`,`10`,`10`,`11`,`15`
- `Math.cummax([[3,5,2],[1,6,3],[7,8,1]])` Returns `3`, `5`, `2`; `3`, `6`, `3`; and `7`, `8`, `3`

6.1.6.6 Math.cumsum

`Math.cumsum(x)` – Returns the cumulative summation of values in the `x` array

Each element in the `x` array is populated with the summary of previous values within the vector.

If it is a two-dimension array, each column vector will be filled with its summary of previous values.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `cumsum(x)`.

For example:

- `Math.cumsum([1:5])` Returns `1`, `3`, `6`, `10`, and `15`
- `Math.cumsum([[1,4,7],[2,5,8],[3,6,9]])` Returns `1`, `4`, `7`; `3`, `9`, `15`; and `6`, `15`, `24`

6.1.6.7 Math.cumprod

`Math.cumprod(x)` – Returns the cumulative multiplications of values in the `x` array

Each element in the `x` array is populated with the multiplications of previous values within the vector.

If it is a two-dimension array, each column vector will be filled with its multiplications of previous values.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `cumprod(x)`.

For example:

- `Math.cumprod([1:5])` Returns `1`, `2`, `6`, `24`, and `120`
- `Math.cumprod([[1,4,7],[2,5,8],[3,6,9]])` Returns `1`,`4`,`7`; `2`,`20`,`56`; and `6`,`120`,`504`

6.1.7 Constants

6.1.7.1 Math.PI

`Math.PI` – π constant

The floating-point π constant is `3.14159265358979323846`.

⇒ NOTE: You can use the abbreviated function name without the `Math` prefix: `PI`.

6.1.7.2 Math.MIN.INTEGER

`Math.MIN.INTEGER` – Minimum integer constant

The minimum integer constant is `-9223372036854775808`.

6.1.7.3 Math.MAX.INTEGER

`Math.MAX.INTEGER` – Maximum integer constant

The maximum integer constant is `9223372036854775807`.

6.1.7.4 Math.MIN.REAL

`Math.MIN.REAL` – Minimum floating-point constant

The minimum floating-point constant is `2.22507e-308`.

6.1.7.5 Math.MAX.REAL

`Math.MAX.REAL` – Maximum floating-point constant

The maximum floating-point constant is `1.79769e+308`.

6.2 Matrix

The matrix operations extend the arithmetic set of operators to multiply and transpose matrices. The calculation's precedence order in the table below extends *Table 6 - Create Demo Project*.

Operator	Precedence	Description
•	2	Matrix can use alternate characters: **
`	3	Transpose

Table 23 - Matrix Operators

⇒ LINK: https://en.wikipedia.org/wiki/Matrix_(mathematics)

6.2.1 Matrix.create

`Matrix.create(n, m)` – Returns `n` × `m` matrix filled with zeros
`Matrix.create(n, m, v)`
`Matrix.create(n, m, v, t)`

The `n` and `m` arguments specify the number of `n` rows by `m` columns of the matrix filling in the initialized value of `v` argument of its specified `t` argument's data type.

If `v` argument is not specified, it defaults to zero.

If `t` argument is not specified, it defaults to `Matrix.REAL` data type.

For example:

- `Matrix.create(2, 3)` Returns matrix:

```
| 0, 0, 0 |
| 0, 0, 0 |
```

- `Matrix.create(2, 3, 2.0)` Returns matrix:

```
| 2, 2, 2 |
| 2, 2, 2 |
```

- `Matrix.create(2,3,2.0, \`
  `Matrix.COMPLEX)` Returns matrix:

```
| 0 + 0i, 0 + 0i, 0 + 0i |
| 0 + 0i, 0 + 0i, 0 + 0i |
```

6.2.2 Matrix.transpose

`Matrix.transpose(x)` – Returns transposed matrix

The `x` argument matrix is transposed by interchanging the rows and columns.

For example:

- `A = [[1.0, 2.0, 3.0], \`
  `[4.0, 5.0, 6.0]]` Returns matrix:

```
| 1, 2, 3 |
| 4, 5, 6 |
```

- `Matrix.transpose(A)` Returns matrix:

```
| 1, 4 |
| 2, 5 |
| 3, 6 |
```

- `A`` Returns matrix:

```
| 1, 4 |
| 2, 5 |
| 3, 6 |
```

6.2.3 Matrix.diag

`Matrix.diag(n)` – Returns diagonal `n` × `n` matrix filled with 1's
`Matrix.diag(n, x)`

A diagonal matrix is a `n × n` matrix if $a_{ij} = 0$ for `i ≠ j`. Hence, the `x` values off the main diagonal are all zero.

If `x` argument is not specified, it defaults to 1.

⇒ NOTE: You can use the alternate function name: `Matrix.diagonal`.

For example:

- `Matrix.diag(3)` Returns matrix:

```
| 1, 0, 0 |
| 0, 1, 0 |
| 0, 0, 1 |
```

- `Matrix.diag(3, 1.1)` Returns matrix:

```
| 1.1,   0,   0 |
|   0, 1.1,   0 |
|   0,   0, 1.1 |
```

6.2.4 Matrix.zero

`Matrix.zero(n, m)` – Returns zero filled `n × m` matrix
`Matrix.zero(n, m, t)`

A zero matrix is a `n × m` matrix if $a_{ij} = 0$. The `t` argument initializes the matrix's data type of `INTEGER`, `REAL`, `COMPLEX`, or `UNIT` type.

If `t` argument is not specified, it defaults to `Matrix.REAL` data type.

For example:

- `Matrix.zero(3, 3)` Returns matrix:

```
| 0, 0, 0 |
| 0, 0, 0 |
| 0, 0, 0 |
```

- `Matrix.zero(3, 3, \`
  `Matrix.COMPLEX)` Returns matrix:

```
| 0 + 0i, 0 + 0i, 0 + 0i |
| 0 + 0i, 0 + 0i, 0 + 0i |
| 0 + 0i, 0 + 0i, 0 + 0i |
```

6.2.5 Matrix.one

`Matrix.one(n, m)` – Returns one filled `n × m` matrix
`Matrix.one(n, m, t)`

A one matrix is a `n × m` matrix if $a_{ij} = 1$. The `t` argument initializes the matrix's data type of `INTEGER`, `REAL`, `COMPLEX`, or `UNIT` type.

If `t` argument is not specified, it defaults to `Matrix.REAL` data type.

For example:

- `Matrix.one(3, 3)` Returns matrix:

```
| 1, 1, 1 |
| 1, 1, 1 |
| 1, 1, 1 |
```

- `Matrix.one(3, 3, \ Matrix.COMPLEX)` Returns matrix:

```
| 1 + 0i, 1 + 0i, 1 + 0i |
| 1 + 0i, 1 + 0i, 1 + 0i |
| 1 + 0i, 1 + 0i, 1 + 0i |
```

6.2.6 Matrix.dot

`Matrix.dot(x, y)` – Returns dot product of two vectors

The two vectors from the `x` and `y` arguments are used to calculate the summation of the multiplied corresponding entries of the vectors to compute its dot product.

For example:

- `Matrix.dot([4.0, -1.0, 2.0], \ [2.0, -2.0, -1.0])` Returns `8`
- `Matrix.dot([1+1i, 1-1i, -1+1i], \ [3-4i, 6-2i, 1+2i])` Returns `8 + -10i`
- `Matrix.dot([1+1i, 1-1i, -1+1i, -1+-1i], \ [3-4i, 6-2i, 1+2i, 4+3i` Returns `7 + -17i`

6.2.7 Matrix.cross

`Matrix.cross(x, y)` – Returns cross-product of two vectors

The perpendicular coefficients of the cross-product are calculated from the `x` and `y` argument vectors.

For example:

- `Matrix.cross([4.0, -1.0, 2.0], \ [2.0, -2.0, -1.0])` Returns `5, 8, -6`

6.2.8 Matrix.det

`Matrix.det(x)` – Returns determinant value of the matrix

The `x` argument is a `n × n` matrix of $[a_{ij}]$. The determinant `|x|` is $\sum \pm a_{1}j_{2}, a_{2}j_{2} \ldots a_{nJn}$ where the summation is over all permutations $j_1, j_2, \ldots, j_n$ of the set `S = {1, 2, …, n}`. The sign ± is used as permutation of $j_1, j_2, \ldots, j_n$ is even or odd.

⇒ NOTE: You can use the alternate function name: `Matrix.determinant`.

For example:

- `Matrix.det([[ 1.0, -2.0, 4.0], \`
  `[-5.0, 2.0, 0.0], \`
  `[ 1.0, 0.0, 3.0]])` Returns `-32`

6.2.9 Matrix.inv

`Matrix.inv(x)` – Returns inverse matrix

The inverse matrix of the square matrix from the `x` argument. The matrix is invertible if there is a matrix `y` of the same size such that `x • y = y • x`, resulting in identity matrix.

⇒ NOTE: You can use the alternate function name: `Matrix.inverse`.

For example:

- `Matrix.inv([[ 1.0, -2.0, 4.0], \`
  `[-5.0, 2.0, 0.0], \`
  `[ 1.0, 0.0, 3.0]])` Returns inverse matrix:

```
| -0.1875, -0.1875,  0.25  |
| -0.46875, 0.03125, 0.625 |
|  0.0625,  0.0625,  0.25  |
```

6.2.10 Matrix.tri_lower

`Matrix.tri_lower(x)` – Returns lower triangle matrix
`Matrix.tri_lower(x, p)`

The `x` argument is `n` x `m` matrix of $[a_{ij}]$. The lower triangle matrix is $a_{ij} = 0$ for `i < j`.

If `p` argument is not specified, the pivot position defaults to `0`.

⇒ NOTE: You can use the alternate function name: `Matrix.triangle_lower`.

For example:

-
```
a = [[11, 12, 13], \
     [21, 22, 23], \
     [31, 32, 33]]
```

- `Matrix.tri_lower(a)` Returns matrix:

```
| 11,  0,  0 |
| 21, 22,  0 |
| 31, 32, 33 |
```

- `Matrix.tri_lower(a, 1)` Returns matrix:

```
| 11, 12,  0 |
| 21, 22, 23 |
| 31, 32, 33 |
```

- `Matrix.tri_lower(a, -1)` Returns matrix:

```
|  0,  0,  0 |
| 21,  0,  0 |
| 31, 32,  0 |
```

- `Matrix.tri_lower(a, -2)` Returns matrix:

```
|  0,  0,  0 |
|  0,  0,  0 |
| 31,  0,  0 |
```

6.2.11 Matrix.tri_upper

`Matrix.tri_upper(x)` – Returns upper triangle matrix
`Matrix.tri_upper(x, p)`

The `x` argument is `n x m` matrix of $[a_{ij}]$. The upper triangle matrix is $a_{ij} = 0$ for `i > j`.

If `p` argument is not specified, the pivot position defaults to `0`.

⇒ NOTE: You can use the alternate function name: `Matrix.triangle_upper`.

For example:

-
```
a = [[11, 12, 13], \
     [21, 22, 23], \
     [31, 32, 33]]
```

- `Matrix.tri_upper(a)` Returns matrix:

```
| 11, 12, 13 |
|  0, 22, 23 |
|  0,  0, 33 |
```

- `Matrix.tri_upper(a, 1)` Returns matrix:

```
|  0, 12, 13 |
```

- `Matrix.tri_upper(a, -1)` Returns matrix:

```
|  0,  0, 23 |
|  0,  0,  0 |
```

6.2.12 Matrix.rows

`Matrix.rows(x, r)` – Returns specified row(s) in the matrix
`Matrix.rows(x, r1, r2)`

The x argument is n x m matrix to extract r row argument or r1 to r2 row arguments to return its sub-matrix of x argument.

For example:

-
```
a = [[10, 11, 12, 13, 14, 15], \
     [20, 21, 22, 23, 24, 25], \
     [30, 31, 32, 33, 34, 35], \
     [40, 41, 42, 43, 44, 45], \
     [50, 51, 52, 53, 54, 55]]
```

- `Matrix.rows(a, 2)`

  Returns matrix:
```
| 30, 31, 32, 33, 34, 35 |
```

- `Matrix.rows(a, [0, 2, 4])`

  Returns matrix:
```
| 10, 11, 12, 13, 14, 15 |
| 30, 31, 32, 33, 34, 35 |
| 50, 51, 52, 53, 54, 55 |
```

- `Matrix.rows(a, 2, 3)`

  Returns matrix:
```
| 30, 31, 32, 33, 34, 35 |
| 40, 41, 42, 43, 44, 45 |
```

- `Matrix.rows(a, 3, 2)`

  Returns matrix:
```
| 30, 31, 32, 33, 34, 35 |
| 40, 41, 42, 43, 44, 45 |
```

- `Matrix.rows(a, 2, [3:4])`

  Returns matrix:
```
| 30, 31, 32, 33, 34, 35 |
| 40, 41, 42, 43, 44, 45 |
| 50, 51, 52, 53, 54, 55 |
```

- `Matrix.rows(a, [3:4], 2)`

  Returns matrix:
```
| 30, 31, 32, 33, 34, 35 |
| 40, 41, 42, 43, 44, 45 |
| 50, 51, 52, 53, 54, 55 |
```

6.2.13 Matrix.cols

`Matrix.cols(x, c)` – Returns specified column(s) in the matrix
`Matrix.cols(x, c1, c2)`

The x argument is `n x m` matrix to extract `c` column argument or `c1` to `c2` column arguments to return its sub-matrix of `x` argument.

For example:

-
```
a = [[10, 11, 12, 13, 14, 15], \
     [20, 21, 22, 23, 24, 25], \
```

```
        [30, 31, 32, 33, 34, 35], \
        [40, 41, 42, 43, 44, 45], \
        [50, 51, 52, 53, 54, 55]]
```

- `Matrix.cols(a, 2)`

  Returns matrix:

  ```
  | 12 |
  | 22 |
  | 32 |
  | 42 |
  | 52 |
  ```

- `Matrix.cols(a, [0, 2, 4])`

  Returns matrix:

  ```
  | 10, 12, 14 |
  | 20, 22, 24 |
  | 30, 32, 34 |
  | 40, 42, 44 |
  | 50, 52, 54 |
  ```

- `Matrix.cols(a, 2, 3)`

  Returns matrix:

  ```
  | 12, 13 |
  | 22, 23 |
  | 32, 33 |
  | 42, 43 |
  | 52, 53 |
  ```

- `Matrix.cols(a, 3, 2)`

  Returns matrix:

  ```
  | 12, 13 |
  | 22, 23 |
  | 32, 33 |
  | 42, 43 |
  | 52, 53 |
  ```

- `Matrix.cols(a, 2, [3:4])`

  Returns matrix:

  ```
  | 12, 13, 14 |
  | 22, 23, 24 |
  | 32, 33, 34 |
  | 42, 43, 44 |
  | 52, 53, 54 |
  ```

- `Matrix.cols(a, [3:4], 2)`

  Returns matrix:

  ```
  | 12, 13, 14 |
  | 22, 23, 24 |
  | 32, 33, 34 |
  | 42, 43, 44 |
  | 52, 53, 54 |
  ```

6.2.14 Matrix.range

`Matrix.range(x, s, e)` – Returns sub-matrix

The `x` argument is `n` x `m` matrix to extract `s` argument, starting with the upper-left's row and column index to `e` argument, ending the lower-right's row and column index.

For example:

```
a = [[10, 11, 12, 13, 14, 15], \
     [20, 21, 22, 23, 24, 25], \
     [30, 31, 32, 33, 34, 35], \
     [40, 41, 42, 43, 44, 45], \
     [50, 51, 52, 53, 54, 55]]
```

- `Matrix.range(a, [2,2], [2,2])` Returns `32`
- `Matrix.range(a, [0,0], [0,0])` Returns `10`
- `Matrix.range(a, [4,5], [4,5])` Returns `55`
- `Matrix.range(a, [2,2], [3,3])` Returns matrix:

```
| 32, 33 |
| 42, 43 |
```

- `Matrix.range(a, [1,1], [4,4])` Returns matrix:

```
| 21, 22, 23, 24 |
| 31, 32, 33, 34 |
| 41, 42, 43, 44 |
| 51, 52, 53, 54 |
```

- `Matrix.range(a, [1,1], [4,3])` Returns matrix:

```
| 21, 22, 23 |
| 31, 32, 33 |
| 41, 42, 43 |
| 51, 52, 53 |
```

- `Matrix.range(a, [0,0], [1,1])` Returns matrix:

```
| 10, 11 |
| 20, 21 |
```

- `Matrix.range(a, [3,4], [4,5])` Returns matrix:

```
| 44, 45 |
| 54, 55 |
```

6.2.15 Matrix.is_diag

`Matrix.is_diag(x)` – Returns a boolean indicator if it's a diagonal matrix

If `x` argument is a `n` × `n` matrix where $a_{ij} = 0$ for `i` ≠ `j` and the `x` values off the main diagonal are all zero.

⇒ NOTE: You can use the alternate function name: `Matrix.is_diagonal`.

For example:

- ```
  a =  [[ 1,  0,  0,  0], \
        [ 0,  1,  0,  0], \
        [ 0,  0,  1,  0], \
        [ 0,  0,  0,  1]]
  ```

- `Matrix.is_diag(a)` Returns `true`

6.2.16 Matrix.is_skew

`Matrix.is_skew(x)` – Returns a boolean indicator if it's a skew symmetric matrix

If the negated `x` argument is equivalent to the transpose matrix, it is a skew symmetric matrix. Hence, `x` ` = -x`.

For example:

- ```
  a =  [[ 0,  1,  2,  5], \
        [ 1,  0,  3,  4], \
        [ 2,  3,  0,  6], \
        [ 5,  4,  6,  0]]
  ```

- `Matrix.is_skew(a)` Returns `false`

- ```
  a =  [[ 0,  1, -2,  5], \
        [-1,  0,  3, -4], \
        [ 2, -3,  0,  6], \
        [-5,  4, -6,  0]]
  ```

- `Matrix.is_skew(a)` Returns `true`

6.2.17 Matrix.is_sym

`Matrix.is_sym(x)` – Returns a boolean indicator if it's a symmetric matrix

If the `x` argument is equivalent to the transpose matrix, it is a symmetric matrix. Hence, `x` ` = x`.

⇒ NOTE: You can use the alternate function name: `Matrix.is_symmetric`

For example:

- ```
  a =  [[ 0,  1,  2,  5], \
        [ 1,  0,  3,  4], \
        [ 2,  3,  0,  6], \
        [ 5,  4,  6,  0]]
  ```

- `Matrix.is_sym (a)` Returns `true`

- ```
  a =  [[ 0,  1, -2,  5], \
  ```
  ```

```
[-1,  0,  3, -4], \
[ 2, -3,  0,  6], \
[-5,  4, -6,  0]]
```

- `Matrix.is_sym (a)` Returns `false`

6.2.18 Matrix.is_tri_lower

`Matrix.is_tri_lower(x)` – Returns a boolean indicator if it's a lower triangle matrix
`Matrix.is_tri_lower(x, p)`

The `x` argument is `n` x `m` matrix of $[a_{ij}]$. The lower triangle matrix is $a_{ij} = 0$ for `i` < `j`.

If `p` argument is not specified, the pivot position defaults to `0`.

⇒ NOTE: You can use the alternate function name: `Matrix.is_triangle_lower`

For example:

- ```
  a = [[11,  0,  0], \
       [21, 22,  0], \
       [31, 32, 33]]
  ```
- `Matrix.is_tri_lower(a)` Returns `true`

6.2.19 Matrix.is_tri_upper

`Matrix.is_tri_upper(x)` – Returns a boolean indicator if it's an upper triangle matrix
`Matrix.is_tri_upper(x, p)`

The `x` argument is `n` x `m` matrix of $[a_{ij}]$. The upper triangle matrix is $a_{ij} = 0$ for `i` > `j`.

If `p` argument is not specified, the pivot position defaults to `0`.

⇒ NOTE: You can use the alternate function name: `Matrix.is_triangle_upper`

For example:

- ```
  a = [[11, 12, 13], \
       [0,  22, 23], \
       [0,   0, 33]]
  ```
- `Matrix.is_tri_upper(a)` Returns `true`

6.2.20 Constants

6.2.20.1 Matrix.INTEGER

`Matrix.INTEGER` – Integer matrix data type constant

6.2.20.2 Matrix.REAL

`Matrix.REAL` – Floating-point matrix data type constant

6.2.20.3 Matrix.COMPLEX

`Matrix.COMPLEX` – Complex matrix data type constant

6.2.20.4 Matrix.UNIT

`Matrix.UNIT` – Unit matrix data type

6.3 Type

The *Type*'s constants are used for *File* and *Create* library functions to define type of data to be used.

6.3.1 Constants

6.3.1.1 Type.INTEGER

`Type.INTEGER` – Integer data type constant

6.3.1.2 Type.REAL

`Type.REAL` – Floating-point data type constant

6.3.1.3 Type.COMPLEX

`Type.COMPLEX` – Complex data type constant

6.3.1.4 Type.UNIT

`Type.UNIT` – Unit data type

6.3.1.5 Type.BOOLEAN

`Type.BOOLEAN` – Boolean data type constant

6.3.1.6 Type.STRING

`Type.STRING` – String data type constant

6.4 Convert

6.4.1 toInteger

`toInteger(x)` – Returns value of `x` as integer

The `x` argument is converted and returns an integer(s) from `real, complex, boolean, unit,` or `string` data type. The fractional part of the floating-point value is truncated.

⇒ NOTE: If `x` argument is `integer` data type, it will return `x`. No conversion is performed.

For example:

- `toInteger(1.5)` Returns `1`
- `toInteger(1.5{ft})` Returns `1`
- `toInteger(1.5 + 1.5i)` Returns `2`
- `toInteger("1.5")` Returns `1`
- `toInteger([1.5,2.5,3.5])` Returns `1`, `2`, and `3`

6.4.2 toReal

`toReal(x)` – Returns value of `x` as floating-point

The `x` argument is converted and returns a floating-point(s) from `integer`, `complex`, `boolean`, `unit`, or `string` data type.

⇒ NOTE: If `x` argument is `real` data type, it will return `x`. No conversion is performed.

For example:

- `toReal(1)` Returns `1.0`
- `toReal(1{ft})` Returns `1.0`
- `toReal(1 + 1.5i)` Returns `1.802776`
- `toReal("1")` Returns `1.0`
- `toReal([1, 2, 3])` Returns `1.0`, `2.0`, and `3.0`

⇒ NOTE: Converting a `complex` number to `integer` or `real` formula is evaluating the value of real part of the complex number and squaring it plus imaginary part of the complex number and squaring it. Then take the square root of the summation of the squared parts: `1.802776` = √ of $1.0^2 + 1.5^2$.

6.4.3 toComplex

`toComplex(x)` – Returns value of `x` as complex

The `x` argument is converted and returns a complex(s) from `integer`, `real`, `boolean`, `unit` or `string` data type.

⇒ NOTE: If `x` argument is `complex` data type, it will return `x`. No conversion is performed.

For example:

- `toComplex(1.5)` Returns `1.5 + 0.0i`
- `toComplex(1.5{ft})` Returns `1.5 + 0.0i`
- `toComplex("1.5")` Returns `1.5 + 0.0i`
- `toComplex([1.5,2.5,3.5])` Returns `1.5+0.0i`, `2.5+0.0i`, and `3.5+0.0i`

6.4.4 toBoolean

`toBoolean(x)` – Returns value of `x` as boolean

The `x` argument is converted and returns a boolean(s) from `integer`, `real`, `complex`, `unit` or `string` data type. If the value is non-zero, it is converted to `true`; otherwise, it is `false`.

⇒ NOTE: If `x` argument is `boolean` data type, it will return `x`. No conversion is performed.

For example, variable `x` will contain value of `true` and `a` will contain an array of `true`, `true`, and `false`:

- `toBoolean(1.5)` Returns `true`
- `toBoolean(1.5{ft})` Returns `true`
- `toBoolean(1.5 + 1.5i)` Returns `true`
- `toBoolean("1.5")` Returns `true`
- `toBoolean("true")` Returns `true`
- `toBoolean("TRUE")` Returns `true`
- `toBoolean("T")` Returns `true`
- `toBoolean("t")` Returns `true`
- `toBoolean("1")` Returns `true`
- `toBoolean([1.5, 2.5, 0])` Returns `true`, `true`, and `false`

⇒ NOTE: If the string is `false`, `FALSE`, `f`, `F`, or any other sequence of characters, it is converted to `false` boolean value.

6.4.5 fmt

`fmt(f, x1, x2, ... xn)` – Returns formatted string of `x`'s values

The `f` argument string format is used to format the value of $x_1, x_2, \ldots x_n$ is returned as a formatted string. The table below will parse `f` argument's string formatting conversion tags. Any other character sequences are pass-though.

Conversion Tags	Description
`%d`	Convert x_i to integer data type
`%<n>d`	Convert x_i to integer data type with *n* number of spaces or digits:

	• If *n* is preceded with `0,` pre-pad with zeros up to the *n* number of zeros. • If minus sign character precedes the *n* attribute, it will format the outputted integer to be left-justified. • If comma character precedes the *n* attribute, it will format the outputted integer to contain comma character on every thousandth digit.
`%g`	Convert x_i to floating-point data type into either integer value or integral floating-point value, floating-point value with factional part, or floating-point with exponent value
`%f`	Convert x_i to floating-point data type
`%e`	Convert x_i to floating-point data type with exponent
`%<`*n.m*`>g, f, e`	Convert x_i to floating-point data type with *n* number of digits with *m* factional digits: • If comma character precedes the *n* attribute, it will format the outputted floating-point to contain comma character on every thousandth integral digit.
`%u`	Convert x_i to floating-point data type with associated unit specification
`%<`*n.m*`>u`	Convert x_i to floating-point data type with *n* number of digits with *m* factional digits with associated unit specification: • If comma character precedes the *n* attribute, it will format the outputted floating-point to contain comma character on every thousandth integral digit. • If exclamation character precedes the *n* attribute, it will translate the floating-point value to integer value.
`%p`	Convert x_i to percentage
`%<`*n.m*`>p`	Convert x_i to percentage with *n* number of digits with *m* factional digits: • If comma character precedes the *n* attribute, it will format the outputted floating-point to contain comma character on every thousandth integral digit.
`%s`	Convert to string
`%<`*n*`>s`	Convert the x_i to integer data type with *n* number of spaces or digits: • If minus sign character precedes the *n* attribute, it will format the outputted string to be left-justified.
`%%`	Pass though `%` character, rather than converting the input argument

For example:

- `fmt("Hello World %d...", 2020)` Returns `Hello World 2020...`
- `fmt("'%s'",  "4")` Returns `'4'`
- `fmt("'%s'",  4)` Returns `'4'`
- `fmt("'%s'",  4.5)` Returns `'4.5'`
- `fmt("'%s'",  4{d})` Returns `'4{d}'`
- `fmt("'%s'",  4.5 + 5.5i)` Returns `'4.5 + 5.5i'`
- `fmt("'%10s'",  "4")` Returns `'         4'`
- `fmt("'%-10s'",  "4")` Returns `'4         '`
- `fmt("'%10s'",  4)` Returns `'         4'`
- `fmt("'%10s'",  4.5)` Returns `'       4.5'`
- `fmt("'%10s'",  4{d})` Returns `'      4{d}'`
- `fmt("'%10s'",  4.5 + 5.5i)` Returns `'4.5 + 5.5i'`
- `fmt("'%d'",  4)` Returns `'4'`

- `fmt("'%10d'",    "4")` Returns `'         4'`
- `fmt("'%10d'",    4)` Returns `'         4'`
- `fmt("'%-10d'",   4)` Returns `'4         '`
- `fmt("'%+10d'",   4)` Returns `'        +4'`
- `fmt("'%-+10d'",  4)` Returns `'+4        '`
- `fmt("'%10d'",    4.5)` Returns `'         4'`
- `fmt("'%10d'",    4{d})` Returns `'         4'`
- `fmt("'%10d'",    4.5 + 5.5i)` Returns `'         7'`
- `fmt("'%g'",      4.5)` Returns `'4.5'`
- `fmt("'%10g'",    "4")` Returns `'         4'`
- `fmt("'%10g'",    4)` Returns `'         4'`
- `fmt("'%10g'",    4.5)` Returns `'       4.5'`
- `fmt("'%10g'",    4{d})` Returns `'         4'`
- `fmt("'%10g'",    4.5 + 5.5i)` Returns `'       4.5 +       5.5i'`
- `fmt("'%f'",      4.5)` Returns `'4.500000'`
- `fmt("'%10.4f'",  "4")` Returns `'    4.0000'`
- `fmt("'%10.4f'",  4)` Returns `'    4.0000'`
- `fmt("'%10.4f'",  4.5)` Returns `'    4.5000'`
- `fmt("'%10.4f'",  4{d})` Returns `'    4.0000'`
- `fmt("'%10.4f'",  4.5 + 5.5i)` Returns `'    4.5000 +    5.5000i'`
- `fmt("'%e'",      4.5)` Returns `'4.500000e+00'`
- `fmt("'%10.4e'",  "4")` Returns `'4.0000e+00'`
- `fmt("'%10.4e'",  4)` Returns `'4.0000e+00'`
- `fmt("'%10.4e'",  4.5)` Returns `'4.5000e+00'`
- `fmt("'%10.4e'",  4{d})` Returns `'4.0000e+00'`
- `fmt("'%10.4e'",  4.5 + 5.5i)` Returns `'4.5000e+00 + 5.5000e+00i'`
- `fmt("'%u'",      45.95)` Returns `'45.95'`
- `fmt("'%u'",      45.95{d})` Returns `'45.95{d}'`
- `fmt("'%5.2u'",   45.95)` Returns `'45.95'`
- `fmt("'%5.2u'",   45.95{d})` Returns `'45.95{d}'`
- `fmt("'%8.2u'",   45.95{d})` Returns `'45.95{d}'`
- `fmt("'%,12.2u'", 4500.95{d})` Returns `'  4,500.9{d}'`
- `fmt("'%,d'",     1000000)` Returns `'1,000,000'`
- `fmt("'%,7d'",    1000000)` Returns `'1,000,000'`
- `fmt("'%,9d'",    1000000)` Returns `'  1,000,000'`
- `fmt("'%,9.1f'",  1000000.1)` Returns `'1,000,000.1'`
- `fmt("'%p'",      4.5%)` Returns `'4%'`
- `fmt("'%5.2p'",   4.5%)` Returns `' 4.50%'`
- `fmt("'%8.3p'",   4000.5%)` Returns `'4000.500%'`
- `fmt("'%,8.0p'",  4000.5%)` Returns `'    4,001%'`
- `fmt("'%p'",      4000.5%)` Returns `'4001%'`

6.5 Sort

6.5.1 sort

`sort(x)` – Returns sorted array

The `x` argument is a one-dimension array to be sorted using quick sort method.

⇒ NOTE: You can use the alternate function name: `Sort.quick`

For example:

- `sort([3, 2, 1])` Returns `1`, `2`, and `3`
- `sort([3.5, 2.5, 1.5])` Returns `1.5`, `2.5`, and `3.5`
- `sort([3+1i, 2+1i, 1+1i])` Returns `1 + 1i`, `2 + 1i`, and `3 + 1i`
- `sort([3{ft},2{ft},1{ft}])` Returns `1{ft}`, `2{ft}`, and `3{ft}`
- `sort(["c", "b", "a"])` Returns `a`, `b`, and `c`

6.5.2 Sort.bubble

`sort.bubble(x)` – Returns sorted array

The `x` argument is a one-dimension array to be sorted using bubble `sort` method.

For example:

- `sort.bubble([3, 2, 1])` Returns `1`, `2`, and `3`
- `sort.bubble([3.5, 2.5, 1.5])` Returns `1.5`, `2.5`, and `3.5`
- `sort.bubble([3+1i, 2+1i, 1+1i])` Returns `1 + 1i`, `2 + 1i`, and `3 + 1i`
- `sort.bubble([3{ft},2{ft},1{ft}])` Returns `1{ft}`, `2{ft}`, and `3{ft}`
- `sort.bubble(["c", "b", "a"])` Returns `a`, `b`, and `c`

6.6 Financials

6.6.1 Financial.future_value

`Financial.future_value(i, n, v)` – Returns future value of periodic payments

The `i` argument is the interest rate of the periodic period, the `n` argument is the number of periods, and `v` is the periodic payments used to calculate the future value at the end of the periods.

Simple compound interest of `4%` for `15` years of `$500` monthly deposit:

- `Financial.future_value(4%÷12, 15×12, -500)` Returns `$12,3045.24`

6.6.2 Financial.periodic_ payment

`Financial.periodic_payment(i, n, v)` – Returns loan's periodic payments

The `i` argument is the interest rate of the periodic period, the `n` argument is the number of periods in the life of the loan, and `v` is the loan used to calculate the monthly payments.

Mortgage loan of `$250,000` at `4%` annual interest rate for `15` years example:

- `Financial.periodic_payment(4%÷12, 15×12, 250k)` Returns `$1,849.22`

6.6.3 Financial.interest_payment

`Financial.interest_payment(i, j, n, v)` – Returns interest portion of a periodic payment

The `i` argument is the interest rate of the periodic period, the `j` argument is the periodic period of the loan, the `n` argument is the number of periods in the life of the loan, and `v` is the loan used to calculate the interest payment portion of the period applied to the loan.

Mortgage loan interest of the first month of `$250,000` at `4%` annual interest rate for `15` years example:

- `Financial.interest_payment(4%÷12,1,15×12,250k)` Returns `$833.33`

6.6.4 Financial.principal_ payment

`Financial.principal_payment(i, j, n, v)` – Returns principal portion of a periodic payment

The `i` argument is the interest rate of the periodic period, the `j` argument is the periodic period of the loan, the `n` argument is the number of periods in the life of the loan, and `v` is the loan used to calculate the principal payment portion of the period applied to the loan.

Mortgage loan principle of the first month of `$250,000` at `4%` annual interest rate for `15` years example:

- `Financial.principal_payment(4%÷12,1,15×12,250k)` Returns `$1,015.89`

⇒ NOTE: The examples above show the interest plus the principal is equal to the loan's first monthly payment.

6.7 File

The *File* library allows you to save a collection of variables or data values for reuse in subsequent script execution. There are two types of data files this library can read and write the app's built-in format and popular comma-separated values (`CSV`) format used by a variety of apps.

6.7.1 File.open

`File.open(f)` – Open data file
`File.open(f, c)`

The `f` argument is the name of the data file stored locally on the app or on cloud server.

If `c` argument is not specified, the data file is saved or loaded locally on the device using `File.LOCAL` constant; otherwise, use `File.CLOUD` constant on the cloud server.

The following code example is used to set up the environment for subsequent `close`, `read`, `write`, `contains`, `delete`, `list`, and `dump` function examples.

```
f = "data set"
File.open(f)

File.write(f, "i", 1)
File.write(f, "r", 2.1)
File.write(f, "c", 3.1 + 1.1i)
File.write(f, "u", 4.1{ft})
File.write(f, "b", true)
File.write(f, "s", "data")
File.write(f, "a", [1, 2, 3])

File.close(f)
```

6.7.2 File.close

`File.close(f)` – Close data file

The `f` argument is the name of the data file to save and close the file.

For example:

- f = "data set"
- `File.open(f)`

- `File.close(f)` Results in closing the data file

6.7.3 File.read

`File.read(f, n)` – Reads data file's variable entry

The `f` argument is the name of the data file used to read the `n` argument's variable value.

For example:

- `f = "data set"`
- `File.open(f)`

- `File.read(f, "i")` Returns `1`
- `File.read(f, "r")` Returns `2.1`
- `File.read(f, "c")` Returns `3.1 + 1.1i`
- `File.read(f, "u")` Returns `4.1{ft}`
- `File.read(f, "b")` Returns `true`

- `File.read(f, "s")` Returns `data`
- `File.read(f, "a")` Returns `1`, `2`, and `3`

- `File.close(f)`

6.7.4 File.write

`File.write(f, n, x)` – Writes the variable entry into the data file

The `f` argument is name of the data file to save the `n` argument variable's `x` argument value.

For example:

- `f = "data set"`
- `File.open(f)`

- `File.write(f, "pi", PI)` Results in saving the `pi` variable in the data file

- `File.close(f)`

6.7.5 File.contains

`File.contains(f, n)` – Returns a boolean indicator if variable exists in the data file

The `f` argument is the name of the data file to check if the `n` argument variable exists in the data file.

For example:

- `f = "data set"`
- `File.open(f)`

- `File.contains(f, "i")` Returns `true`
- `File.contains(f, "r")` Returns `true`
- `File.contains(f, "c")` Returns `true`
- `File.contains(f, "u")` Returns `true`
- `File.contains(f, "b")` Returns `true`
- `File.contains(f, "s")` Returns `true`
- `File.contains(f, "a")` Returns `true`
- `File.contains(f, "x")` Returns `false`

- `File.close(f)`

6.7.6 File.delete

`File.delete(f, n)` – Removes the variable entry in the data file

The `f` argument is the name of the data file to remove the `n` argument variable entry from the data file.

For example:

- `f = "data set"`
- `File.open(f)`
- `File.delete(f, "i")` Results in removal from the data file
- `File.close(f)`

6.7.7 File.list

`File.list(f)` – Returns list of variable entries in the data file
`File.list(f, c)`

The `f` argument is the name of the data file.

If `c` argument is not specified, check if the data file is on the local device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

For example:

- `f = "data set"`
- `File.list(f)` Returns `a`, `b`, `c`, `r`, `s`, and `u`

⇒ NOTE: The results assume `File.delete(f, "i")` was invoked in the previous example.

6.7.8 File.dump

`File.dump(f)` – Outputs the data file contents
`File.dump(f, c)`

The `f` argument is name of the data file stored locally on the app or on cloud server.

If `c` argument is not specified, check if the data file is on the local device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

For example:

- `f = "data set"`
- `File.dump(f)` Returns the following outputs:

```
data:
    variable:   a
    type:       integer
    dimensions: 3
    [000]:      1
    [001]:      2
    [002]:      3
```

```
data:
    variable:   b
    type:       boolean
    value:      true

data:
    variable:   c
    type:       complex
    value:      3.1 + 1.1i

data:
    variable:   r
    type:       real
    value:      2.1

data:
    variable:   s
    type:       string
    value:      data

data:
    variable:   u
    type:       unit
    value:      4.1{ft}
```

⇒ NOTE: The results assume `File.delete(f, "i")` was invoked in the previous example.

6.7.9 File.export

`File.export(f, x)` – Exports one- or two-dimension array onto a `CSV` file
`File.export(f, x, c)`

The `f` argument is the name of the `CSV` file containing the outputted `x` argument array.

If `c` argument is not specified, the `CSV` file is saved locally on the device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

⇒ NOTE: The `.CSV` file extension is automatically added to the end of the `f` argument file name.

For example:

- ```
  a = [[ 1, 2, 3, 4, 5], \
       [ 6, 7, 8, 9,10], \
       [11,12,13,14,15], \
       [16,17,18,19,20]]
  ```
  ```

- `f = "data"`
- `File.export(f, a)` Results in exporting the array to `CSV` file

6.7.10 File.import

`File.import(f)` – Reads imported `CSV` file
`File.import(f, t, c)`

The `f` argument is the name of the `CSV` file imported into an array using the `t` argument data type.

If `t` argument is not specified, the data type defaults to `Type.REAL`.

If `c` argument is not specified, import the `CSV` file if it is on the local device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant look on the cloud server.

⇒ NOTE: The `.CSV` file extension is automatically added to the end of the `f` argument file name.

For example:

- `f = "data"`
- `File.Import(f, Type.INTEGER)` Returns:

```
|  1,  2,  3,  4 |
|  6,  7,  8,  9 |
| 11, 12, 13, 14 |
| 16, 17, 18, 19 |
```

⇒ NOTE: The results assume `File.export(f, a)` was invoked in the previous example.

6.7.11 File.exists

`File.exists(f)` – Returns boolean indicator if the data file exists
`File.exists(f, c)`

The `f` argument is the name of the data file to check if it exists.

If `c` argument is not specified, check if the data file is on the local device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

For example:

- `f = "data set"`
- `File.exists(f)` Returns `true`

⇒ NOTE: The results will be `true` if the previous examples were invoked.

6.7.12 File.copy

`File.copy(src, dst)` – Copies the data file
`File.copy(src, dst, srcCloud)`
`File.copy(src, dst, srcCld, dstCld)`

The `src` argument is the source data file name to be copied to the `dst` argument data file's destination.

If the `srcCld` argument is not specified, a copy of the data file is on the local device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

If the `dstCld` argument is not specified, the data file is copied locally on the device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

For example:

- `File.copy("myData set", "yourData")` Results in copying the data file

6.7.13 File.rename

`File.rename(src, dst)` – Renames the data file
`File.rename(src, dst, srcCloud)`
`File.rename(src, dst, srcCld, dstCld)`

The `src` argument is the source data file name to be renamed to the `dst` argument data file's destination.

If the `srcCld` argument is not specified, rename if the data file is on the local device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

If the `dstCld` argument is not specified, the data file is renamed locally on the device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

For example:

- `File.rename("myData", "yourData")` Results in renaming data file

6.7.14 File.remove

`File.remove(f)` – Removes the data file
`File.remove(f, c)`

The `f` argument is name of the data file to remove from local device or cloud server.

If `c` argument is not specified, remove if the data file is on the local device using the `File.LOCAL` constant; otherwise, use the `File.CLOUD` constant on the cloud server.

For example:

- `File.remove("myData")` Results in the data file being deleted

6.7.15 Constants

6.7.15.1 File.LOCAL

`File.LOCAL` – Read/write or import/export on the local device

6.7.15.2 File.CLOUD

`File.CLOUD` – Read/write or import/export on the cloud server

6.8 Create

The *Create* library functions will create special variables to perform specific tasks that are not the primitive data types such as integer, floating-point, complex, etc. These are special variables containing a collection of intrinsic functions and fields used to a perform tasks such as *Plot* variable holding the plotting data used to draw the graphs, how to the data is presented in the graph, etc. The following sections *Polynomial*, *Interpolate*, and others will describe how to use these special variables.

6.8.1 Create.array

`Create.array(v, i)` – Returns one-dimension array
`Create.array(v, i, j)` – Returns two-dimension array
`Create.array(v, i, j, k)` – Returns three-dimension array
`Create.array(v, i, j, k, m)` – Returns four-dimension array

The `v` argument is value initialized in the `i` argument's one-dimension array size. The `j`, `k`, and `m` arguments will create multi-dimension arrays accordingly. Conversely, the `v` argument can be an array used to initialize the created array.

For example:

- `v = Create.array(0, 3)` Returns integer array of `3` elements
- `v = Create.array(0.0, 3, 3)` Returns floating-point `3 × 3` array
- `v = Create.array(0 + 0i, 3, 3, 3)` Returns complex `3 × 3 × 3` array
- `v = Create.array(0{ft}, 3, 3, 3, 3)` Returns unit `3 × 3 × 3 × 3` array
- `v = Create.array(false, 3, 3)` Returns boolean `3 × 3` array
- `v = Create.array("", 3, 3)` Returns string `3 × 3` array

6.8.2 Create.structure

`Create.structure()` – Returns structure variable

The structure variable allows adding *n*-level sub-field names to organize data in an organized way.

For example:

- `v = Create.structure()`

- `v.dog.name = "Penny"`
- `v.dog.age  = 6{yr}`
- `v.dog.male = false`

- `v.cat.name = "Mack"`
- `v.cat.age  = 6{yr}`
- `v.cat.male = true`

6.8.3 Create.dictionary

`Create.dictionary()` – Returns dictionary variable

The dictionary variable allows *n*-dimensional indexing of organizing the data in an alternate way from the `Create.structure` variable type.

For example:

- `v = Create.dictionary()`

- `v["dog", "name"] = "Penny"`
- `v["dog", "age"]  = 6{yr}`
- `v["dog", "male"] = false`

- `v["cat", "name"] = "Mack"`
- `v["cat", "age"]  = 5{yr}`
- `v["cat", "male"] = true`

⇒ NOTE:
- The indexing parameters can be anything: `string`, `integer`, `real`, `complex`, `boolean`, or `unit` types. These will be automatically converted to a string for indexing into the dictionary.
- If the indexing parameter is an array, it will use the first array element as the index value.

6.8.4 Create.polynomial

`Create.polynomial(x)` – Returns a polynomial variable

The `x` argument is a coefficient array of the polynomial where the first element in the array is c_0, the second is c_1, etc. Refer to the ***Polynomial*** section for more details on using the polynomial variable.

For example:

- `v = Create.polynomial([10, -6, -4, 3])` Returns $3 \cdot x^3 - 4 \cdot x^2 - 6 \cdot x + 10$

- `v = Create.polynomial([-2.0, 1.0])` Returns `x - 2`

6.8.5 Create.interpolate

`Create.`interpolate`(t, y, i)` – Returns interpolate variable

The `t` argument is the type of interpolation to be performed in incrementally spaced `i` arguments of the `y` argument data array. Refer to the ***Interpolate*** section for more details using the interpolate variable.

For example:

- `i = 0.1`
- `y = sin([0:PI:inc])`

- `Create.interpolate(Interpolate.Cubic_B_Spline, x, i)` Results in interpolate variable

6.8.6 Create.dataset

`Create.dataset()` – Returns dataset variable for plotting
`Create.dataset(n)`

If the `n` argument is not specified, the dataset name is automatically generated. Refer to the ***Dataset*** section for more details using the dataset variable.

For example:

- `v = Create.dataset("myData")` Returns `myData`'s dataset variable

6.8.7 Create.plot

`Create.plot()` – Returns plot variable
`Create.plot(n)`

If the `n` argument is not specified, the plot name is automatically generated. Refer to the ***Plot*** section for more details using the plot variable.

For example:

- `v = Create.plot("myPlot")` Returns `myPlot`'s plot variable

6.8.8 Create.graph

`Create.graph()` – Returns graph variable
`Create.graph(n)`

If the `n` argument is not specified, the graph name is automatically generated. Refer to the ***Graph*** section for more details using the graph variable.

For example:

- `v = Create.graph("myGraph")` Returns `myGraph`'s graph variable

6.8.9 Create.distribution

`Create.distribution(d, ...)` – Returns statistical distribution model

The `d` argument is a type of statistical distribution data. The following arguments is dependent on the selected distribution. Refer to the ***Distributions*** section for more details on using the statistical distribution variable.

For example:

- `N = 100`
- `v = Create.distribution(Dist.CHI_SQUARED, N-1)` Returns *Chi-Square* distribution

6.8.10 Create.dialog

`Create.dialog()` – Returns dialog variable

Refer to the ***Dialog*** section for more details using the dialog variable.

For example:

- `v = Create.dialog()` Returns dialog variable

6.8.11 Create.grep

`Create.grep()` – Returns General Regular Expression Print (GREP) variable
`Create.grep(p)`

If the `p` argument is not specified, the `set` method must be called to set the string pattern matching.

The grep variable supports == and `!=` relational operators.

For example:

- `g1 = Create.grep()`
- `g2 = Create.grep("([0-9]+)[-]([0-9]+)[-]([0-9]+])")`

- `g1.set("([0-9]+)[/]([0-9]+)[/]([0-9]+])")`
- `g1 == "12/24/2022"` Returns `true`
- `g2 == "12-24-2020"` Returns `true`

- `g1 == g2` Returns `false`
- `g1 != g2` Returns `true`

6.8.12 Create.date

`Create.date()` – Returns date variable
`Create.date(t)`

If the `t` argument is not specified, the date-time is set to `January 1, 1900 00:00:00`.

The date variable supports ==, `!=`, >, >=, <, and <= relational operators.

For example:

- `d1 = Create.date()` Returns date variable
- `d2 = Create.date("12/24/2022 08:30")`

- `d1 == d2` Returns `false`
- `d1 != d2` Returns `true`
- `d1 > d2` Returns `false`
- `d1 >= d2` Returns `false`
- `d1 < d2` Returns `true`
- `d1 <= d2` Returns `true`

- `d1 == "12/24/2022 08:30"` Returns `false`
- `d1 != "12/24/2022 08:30"` Returns `true`
- `d1 > "12/24/2022 08:30"` Returns `false`
- `d1 >= "12/24/2022 08:30"` Returns `false`
- `d1 < "12/24/2022 08:30"` Returns `true`
- `d1 <= "12/24/2022 08:30"` Returns `true`

6.9 Polynomial

The *Polynomial* variable is created using the `Create.polynomial` function that contains an array of coefficients used for various calculations, such as evaluating the function of `x`, adding, subtracting, etc. The coefficients are ordered where the first element (i.e., zero index) in the array is c_0, the second is c_1, etc.

⇒ LINK: https://en.wikipedia.org/wiki/Polynomial

For example, the $2 \cdot x^2 + 3 \cdot x + 4$ polynomial will have the following coefficients:

- c_0 is 4
- c_1 is 3
- c_2 is 2

Generalized polynomial expression of the *n*th order is as follows:

$$c_n \cdot x^n + \ldots + c_4 \cdot x^4 + c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0$$

The following sub-section's intrinsic function examples will use the following coefficient arrays with its associated polynomial variables:

```
a  = [ 10, -6, -4, 3 ]
b  = [ -2.0, 1.0 ]

pA = Create.Polynomial(a)
pB = Create.Polynomial(b)
```

The polynomial variables supports `+`, `-`, `×`, and `÷`, and `%` arithmetic operators.

For example:

- `pA + pB` Results in $3 \cdot x^3 - 4 \cdot x^2 - 5 \cdot x + 8$
- `pA - pB` Results in $3 \cdot x^3 - 4 \cdot x^2 - 7 \cdot x + 12$
- `pA × pB` Results in $3 \cdot x^4 - 10 \cdot x^3 + 2 \cdot x^2 + 22 \cdot x - 20$
- `pA ÷ pB` Results in $3 \cdot x^2 + 2 \cdot x - 2$
- `pA % pB` Results in `6`

Likewise, polynomial variables support == and `!=` relational operators.

- `pA == pB` Results in `0`
- `pA != pB` Results in `1`

6.9.1 Poly.fit

`Poly.fit(x, y, n)` – Polynomial `(x, y)` data curve fitting

The `x` and `y` arguments curve data pairs derive the polynomial's coefficients to the `n` argument's degrees using the least-squares method.

⇒ NOTE: The maximum `n` argument's degrees in `1024`.

⇒ LINK: https://en.wikipedia.org/wiki/Least_squares

For example:

- `x = [0.0, 1.0, 2.0, 3.0, 4.0]`
- `y = [1.0, 1.8, 1.3, 2.5, 6.3]`

- `Poly.fit(x, y, 2)` Returns $0.55 \cdot x^2 - 1.07 \cdot x + 1.42$

6.9.2 Poly.linear

`Poly.linear(x, y)` – Linear `(x, y)` data line fitting

The `x` and `y` argument data pairs derive the simple line formula `y = m · x + b` using the least-squares method.

For example:

- `x = [50, 70, 100, 120]`
- `y = [12, 15,  21,  25]`

- `Poly.linear(x, y, 2)` Returns `0.18793·x + 2.27586`

6.9.3 Intrinsic

6.9.3.1 Functions

6.9.3.1.1 .x

`v.x(n)` – Returns polynomial's expression result

The `v` argument is the polynomial variable evaluating the `n` argument's value.

For example, evaluate the polynomial expression `x - 2`:

- pB.x(0) Returns `-2`
- pB.x(1) Returns `-1`
- pB.x(2) Returns `0`
- pB.x(3) Returns `1`

6.9.3.1.2 .fmt

`v.fmt()` – Returns formatted string of the polynomial variable

The `v` argument is the polynomial variable.

For example:

- `pA.fmt()` Returns $3 \cdot x^3 - 4 \cdot x^2 - 6 \cdot x + 10$
- `pB.fmt()` Returns `x - 2`

6.9.3.1.3 .add

`v.add(x)` – Returns addition of two polynomial variables

The `v` argument is the polynomial variable adding `x` argument's array of coefficients or polynomial variable.

For example:

- `pA.add(b)` Returns $3 \cdot x^3 - 4 \cdot x^2 - 5 \cdot x + 8$
- `pA.add(pB)` Returns $3 \cdot x^3 - 4 \cdot x^2 - 5 \cdot x + 8$

6.9.3.1.4 .sub

`v.sub(x)` – Returns subtraction of two polynomial variables

The `v` argument is the polynomial variable subtracting `x` argument's array of coefficients or polynomial variable.

⇒ NOTE: You can use the alternate intrinsic function name: `subtract`.

For example:

- `pA.sub(b)` Returns $3 \cdot x^3 - 4 \cdot x^2 - 7 \cdot x + 12$
- `pA.subtract(pB)` Returns $3 \cdot x^3 - 4 \cdot x^2 - 7 \cdot x + 12$

6.9.3.1.5 .mul

`v.mul(x)` – Returns multiplication of two polynomial variables

The `v` argument is the polynomial variable multiplying `x` argument's array of coefficients or polynomial variable.

⇒ NOTE: You can use the alternate intrinsic function name: `multiply`.

For example:

- `pA.mul(b)` Returns $3 \cdot x^4 - 10 \cdot x^3 + 2 \cdot x^2 + 22 \cdot x - 20$
- `pA.multiply(pB)` Returns $3 \cdot x^4 - 10 \cdot x^3 + 2 \cdot x^2 + 22 \cdot x - 20$

6.9.3.1.6 .div

`v.div(x)` – Returns division of two polynomial variables

The `v` argument is the polynomial variable dividing `x` argument's array of coefficients or polynomial variable.

⇒ NOTE: You can use the alternate intrinsic function name: `divide`.

For example:

- `pA.div(b)` Returns $3 \cdot x^2 + 2 \cdot x - 2$
- `pA.divide(pB)` Returns $3 \cdot x^2 + 2 \cdot x - 2$

6.9.3.1.7 .mod

`v.mod(x)` – Returns modulo of two polynomial variables

The `v` argument is the polynomial variable modulo `x` argument's array of coefficients or polynomial variable.

⇒ NOTE: You can use the alternate intrinsic function name: `modulo`.

For example:

- `pA.mod(b)` Returns `6`
- `pA.modulo(pB)` Returns `6`

6.9.3.1.8 .equals

`v.equals(x)` – Returns a boolean indicator if the two polynomials are equal to each other

The `v` argument is the polynomial variable equating the `x` argument's array of coefficients or polynomial variable.

For example:

- `pA.equals(a)` Returns `true`
- `pA.equals(pB)` Returns `false`

6.10 Find

Find's `minima` and `maxima` functions uses Brent's method root-finding algorithm, which uses a combination of the bi-section, secant, and inverse quadratic interpolation methods.

The following sub-section's function examples will use the following minimum and maximum variables with its associated sine's `(x, y)` data pairs used in the following plot examples:

```
xMin = 0
xMax = PI × 2

x1   = [xMin:xMax:0.01]
y1   = sin(x1)
```

⇒ LINK: https://en.wikipedia.org/wiki/Brent%27s_method

6.10.1 Find.minima

`Find.minima(f, min, max)` – Finds the function minimum value
`Find.minima(f, min, max, i)`

The `f` argument is the function used to find the minimum value within the minimum's `min` argument to maximum's `max` argument range along the x-axis of the function.

If the `i` argument is not specified, the maximum number of iterations will be performed in searching for the minimum value. The default maximum iteration is `20`.

Plot example:

```
x2   = find.minima(sine, PI, PI × 2)
```

```
y2    = sine(x2)

Utils.ScatterIt("Sine's Minimum",        \
                "Plt025_FindMinimum",    \
                x1, y1, x2, y2,          \
                xMin, xMax, -1.25, 1.25,\
                "Sine", "Minimum", 5, 11)
```

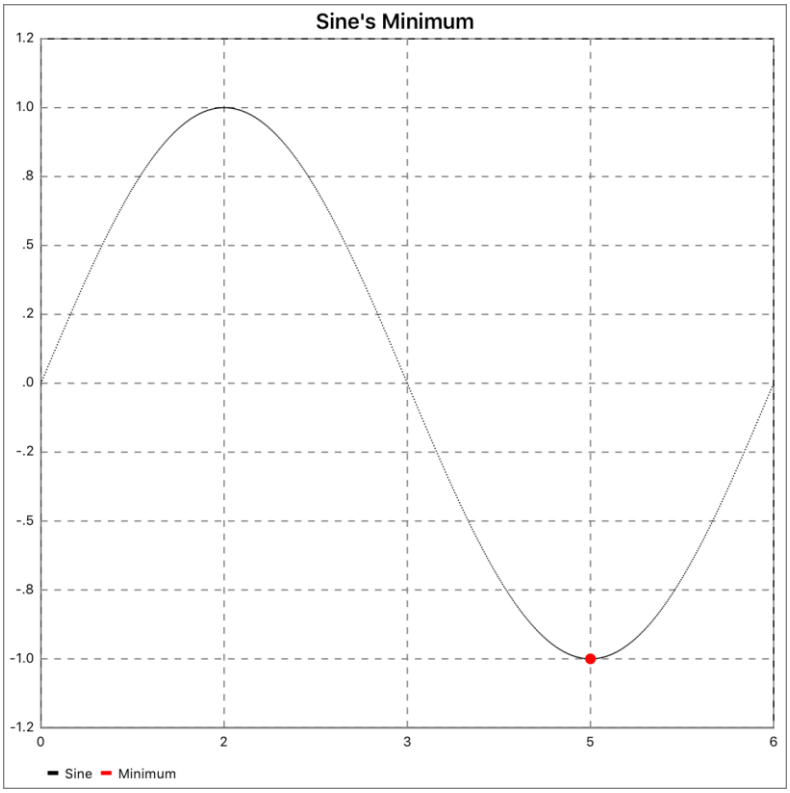

Figure 69 - Find Minimum Plot

6.10.2 Find.maxima

`Find.maxima(f, min, max)` – Finds the function maximum value
`Find.maxima(f, min, max, i)`

The `f` argument is the function used to find the maximum value within the maximum's `min` argument to maximum's `max` argument range along the x-axis of the function.

If the `i` argument is not specified, the maximum number of iterations will be performed in searching for the maximum value. The default maximum iteration is `20`.

Plot example:

```
x2    = find.maxima(sine, 0.0, PI)
y2    = sine(x2)

Utils.ScatterIt("Sine's Maximum",        \
                "Plt026_FindMaximum",    \
```

```
    x1, y1, x2, y2,          \
    xMin, xMax, -1.25, 1.25,\
"Sine", "Maximum", 5, 11)
```

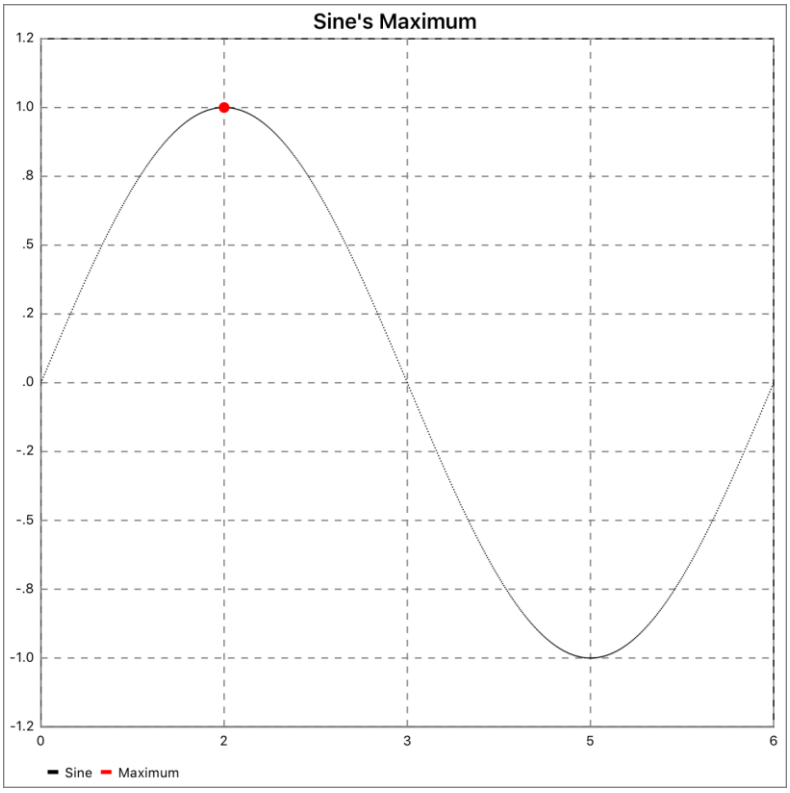

Figure 70 - Find Maximum Plot

6.11 Root

Root has a collection of different methods in finding the root (i.e., finding `x` value where `y` is zero) within its specified range.

The following sub-section's function examples will use the following minimum and maximum variables with its associated sine's `(x, y)` data pairs with function callback used in the following plot examples:

```
xMin = 0
xMax = PI × 2

x1   = [xMin:xMax:0.01]
y1   = sin(x1)

function sine(x)
    return sin(x)
end
```

6.11.1 root

`root(f, min, max)` – Finds the function root value

```
root(f, min, max, i)
```

The `f` argument is the function used to find the root value within the maximum's `min` argument to maximum's `max` argument range along the x-axis of the function.

If the `i` argument is not specified, the maximum number of iterations will be performed in searching for the maximum value. The default maximum iteration is `20`.

⇒ NOTE: You can use the alternate function name: `Root.bisection`.

Plot example:

```
x2   = [root(sine, -PI ÷ 4, PI ÷ 4),                    \
        root(sine, PI - PI ÷ 4, PI + PI ÷ 4),           \
        root(sine, PI × 2 - PI ÷ 4, PI × 2 + PI ÷ 4)]
y2   = sin(x2)

Utils.ScatterIt("Sine's Bi-Section Roots",     \
                "Plt027_RootBiSection",        \
                x1, y1, x2, y2,                \
                xMin, xMax, -1.25, 1.25,       \
                "Sine", "Roots", 5, 11)
```

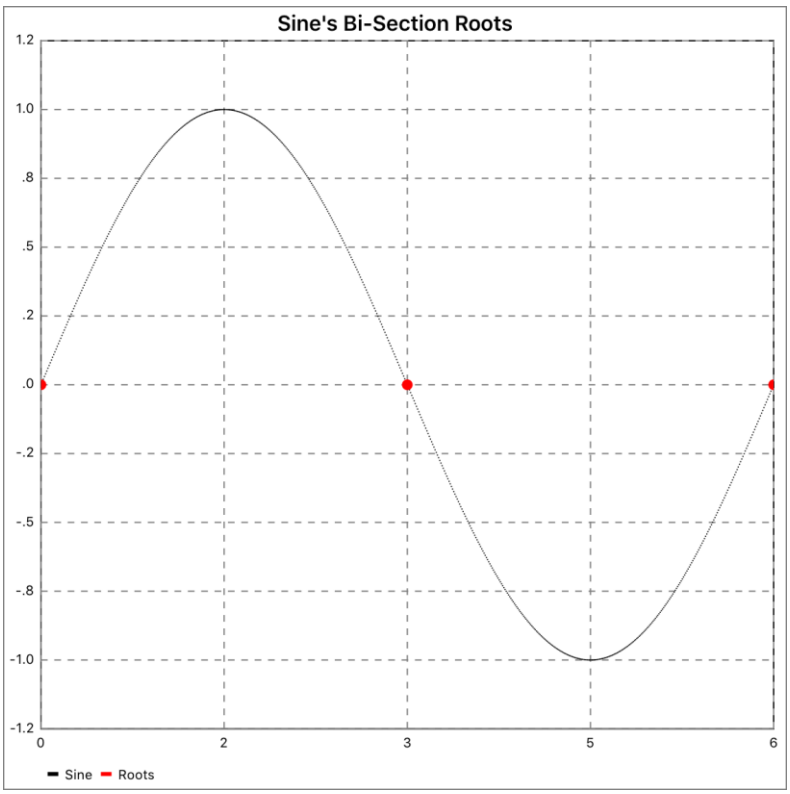

Figure 71 - Root Bi-Section Plot

6.11.2 Root.bracket

`Root.bracket(f, g, r)` – Finds the function root value
`Root.bracket(f, g, r, i)`

The `f` argument is the function used to find the root value starting at initial guess of the `g` argument whether it is increasing or decreasing the `r` argument.

If the `i` argument is not specified, the maximum number of iterations will be performed in searching for the maximum value. The default maximum iteration is `20`.

⇒ NOTE: The function must be monotonic where the initial guess must be half on the x-axis, meaning that the initial guess must be less than `x`'s root value.

Plot example:

```
x2   = root.bracket(sine, PI ÷ 2.0 + PI ÷ 4.0, false)
y2   = sine(x2)

Utils.ScatterIt("Sine's Bracket Roots", \
                "Plt028_RootBracket",   \
                x1, y1, x2, y2,         \
                xMin, xMax, -1.25, 1.25,\
                "Sine", "Roots", 5, 11)
```

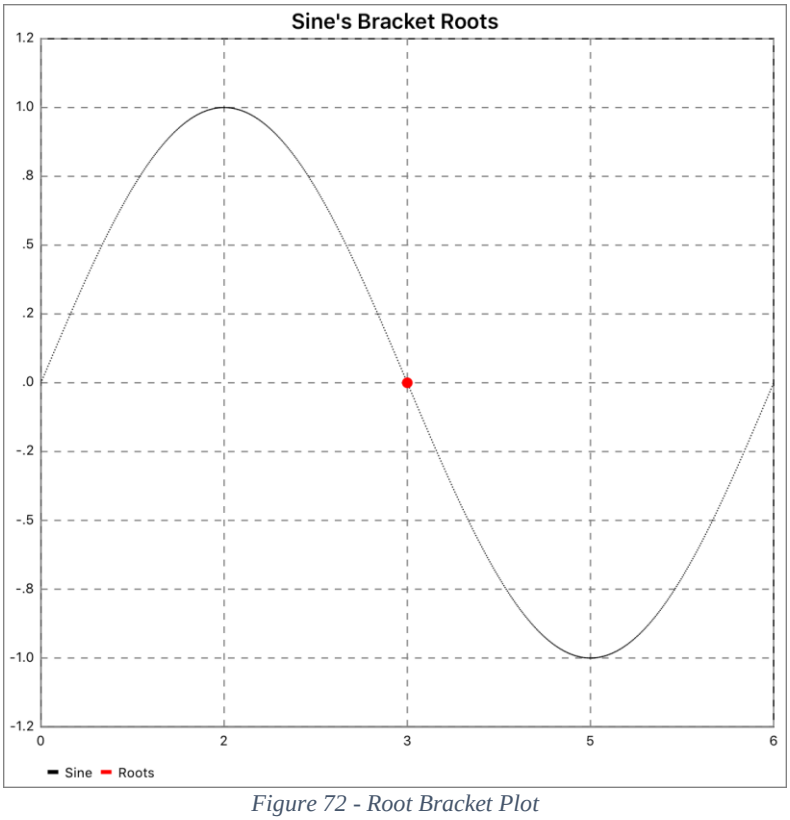

Figure 72 - Root Bracket Plot

6.11.3 Root.halley

`Root.halley(f, g, min, max)` – Finds the function root value
`Root.halley(f, g, min, max, i)`

The `f` argument includes first and second derivative functions used to find the root value starting at initial guess of the `g` argument within the maximum's `min` argument to maximum's `max` argument range along the x-axis of the function. The function needs to return tuple values of function value and its first and second derivative.

If the `i` argument is not specified, the maximum number of iterations will be performed in searching for the maximum value. The default maximum iteration is `20`.

⇒ LINK: https://en.wikipedia.org/wiki/Halley%27s_method

Plot example:

```
function sineD2(x)
    return [sin(x), cos(x), -sin(x)];
end

x2   = [root.halley(sineD2, -PI ÷ 8,          \
                            -PI ÷ 4,          \
                            PI ÷ 4),          \
        root.halley(sineD2, PI - PI ÷ 8,      \
                            PI - PI ÷ 4,      \
                            PI + PI ÷ 4),     \
        root.halley(sineD2, PI × 2 - PI ÷ 8, \
                            PI × 2 - PI ÷ 4, \
                            PI × 2 + PI ÷ 4)]
y2   = sin(x2)

Utils.ScatterIt("Sine's Halley Roots",        \
                "Plt029_Halley",              \
                x1, y1, x2, y2,               \
                xMin, xMax, -1.25, 1.25,      \
                "Sine", "Roots", 5, 11)
```

⇒ NOTE: The derivative of `sin(x)` is `cos(x)`, and second derivative is `-sin(x)`.

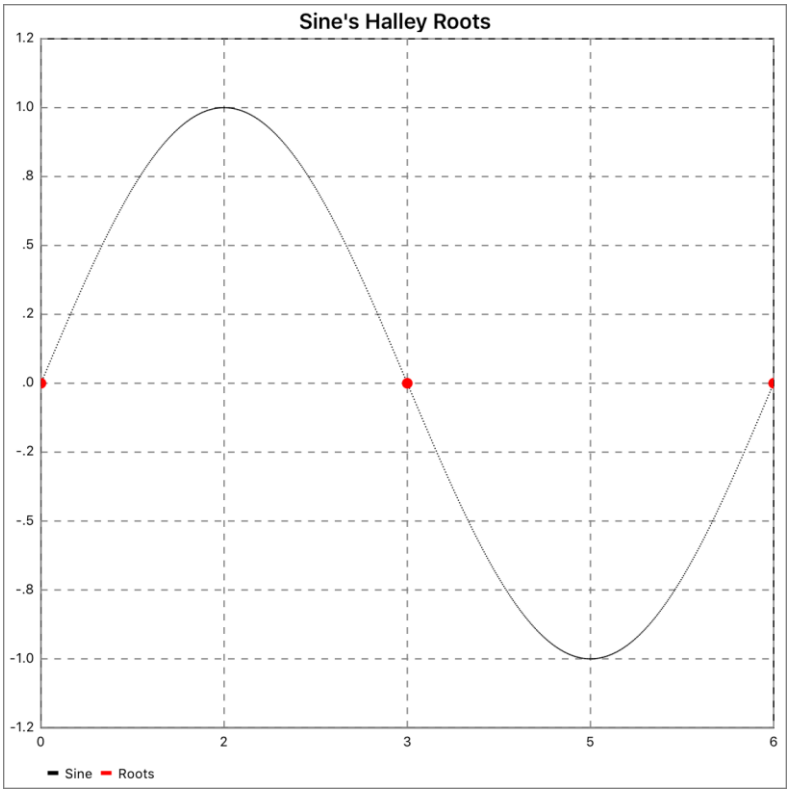

Figure 73 - Root Halley Plot

6.11.4 Root.newton_raphson

`Root.newton_raphson(f, g, min, max)` – Finds function root value
`Root.newton_raphson(f, g, min, max, i)`

The `f` argument includes the first derivative function used to find the root value starting at initial guess of the `g` argument within the maximum's `min` argument to maximum's `max` argument range along the x-axis of the function. The function needs to return pair values of function value and its first derivative.

If the `i` argument is not specified, the maximum number of iterations will be performed in searching for the maximum value. The default maximum iteration is `20`.

⇒ LINK: https://en.wikipedia.org/wiki/Newton%27s_method

Plot example:

```
function sineD(x)
    return [sin(x), cos(x)];
end

x2   = [root.newton_raphson(sineD, -PI ÷ 8,          \
                                   -PI ÷ 4,          \
                                   PI ÷ 4),          \
        root.newton_raphson(sineD, PI - PI ÷ 8,      \
                                   PI - PI ÷ 4,      \
```

```
                                        PI + PI ÷ 4),     \
        root.newton_raphson(sineD,  PI × 2 - PI ÷ 8,  \
                                        PI × 2 - PI ÷ 4,  \
                                        PI × 2 + PI ÷ 4)]
y2    = sin(x2)

Utils.ScatterIt("Sine's Newton-Raphson Roots",       \
                "Plt030_RootNewtonRaphson",          \
                x1, y1, x2, y2,                      \
                xMin, xMax, -1.25, 1.25,             \
                "Sine", "Roots", 5, 11)
```

⇒ NOTE: The derivative of `sin(x)` is `cos(x)`.

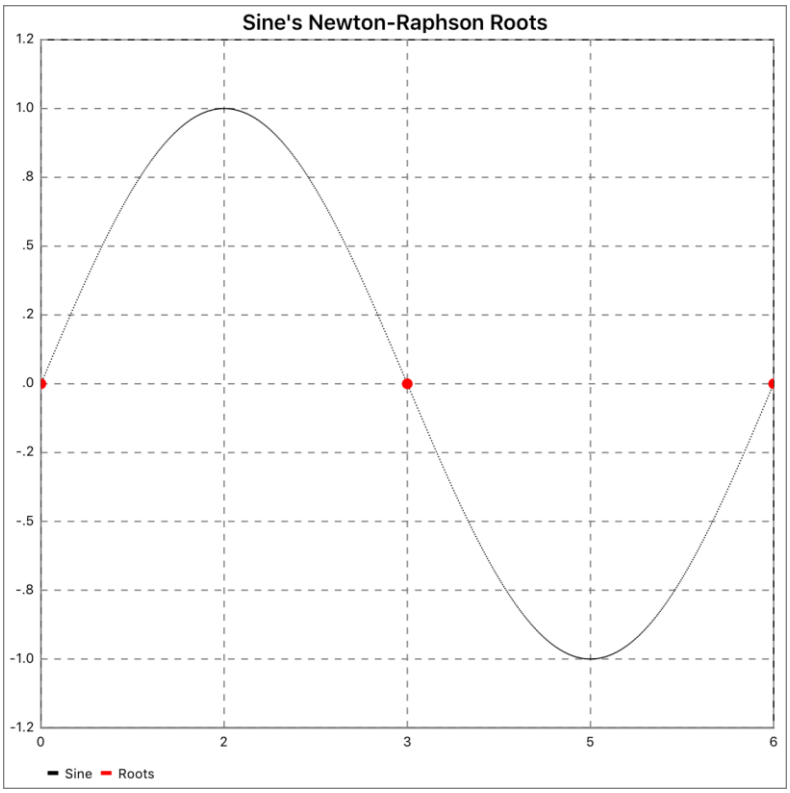

Figure 74 - Root Newton-Raphson Plot

6.12 Interpolate

Interpolating data is a method of deriving an approximate value between collection of points using Barycentric and Cubic B-Spline methods.

⇒ LINK: https://en.wikipedia.org/wiki/Interpolation

6.12.1 Intrinsic

6.12.1.1 Functions

6.12.1.1.1 .interp

`v.interp(x)` – Returns interpolated values

The `v` argument is the interpolate variable of `x` argument value.

⇒ NOTE: You can use the alternate intrinsic function name: `interpolate.`

Plot example:

```
inc  = 0.1

x    = [xMin:xMax:inc]    # sample data
y    = sin(x)

x1   = [xMin:xMax:0.01]   # high resolution
y1   = sin(x1)

fx   = Create.Interpolate(Interpolate.BARYCENTRIC, x, y)

x2   = [xMin:xMax:0.5]    # interpolate
y2   = fx.interpolate(x2)

Utils.ScatterIt("Barycentric Interpolation",    \
                "Plt031_Barycentric",           \
                x1, y1, x2, y2,                 \
                xMin, xMax, -1.25, 1.25,        \
                "Sine", "Interpolate", 5, 11)

fx   = Create.Interpolate(Interpolate.CUBIC_B_SPLINE, y, inc)

x2   = [xMin:xMax:0.5]    # interpolate
y2   = fx.interpolate(x2)

Utils.ScatterIt("Cubic B-Spline Interpolation", \
                "Plt032_ CubicBSpline ",        \
                 x1, y1, x2, y2,                \
                 xMin, xMax, -1.25, 1.25,       \
                 "Sine", "Interpolate", 5, 11)
```

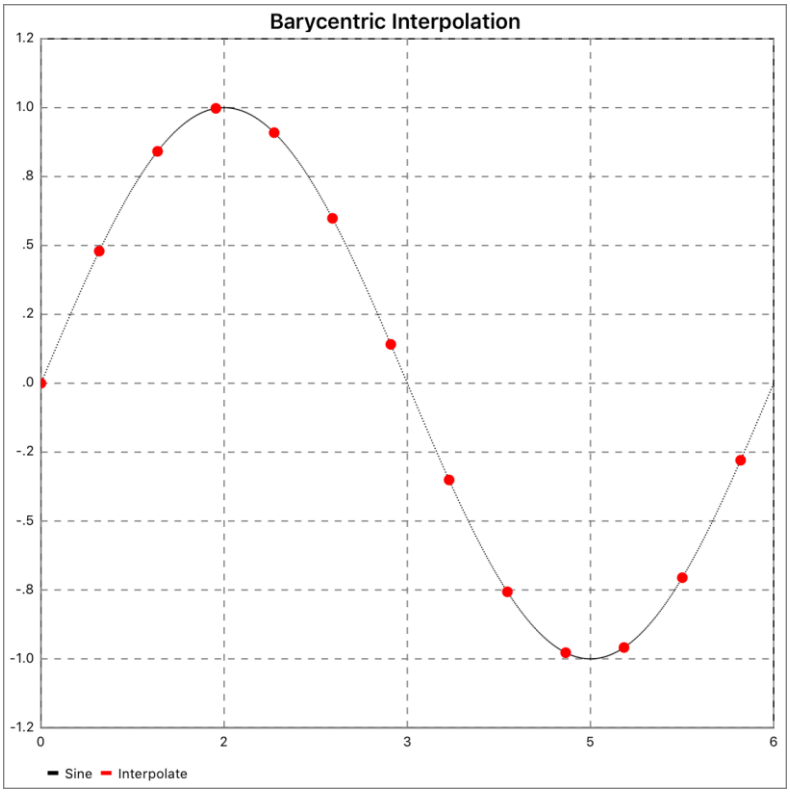

Figure 75 - Interpolate Barycentric Plot

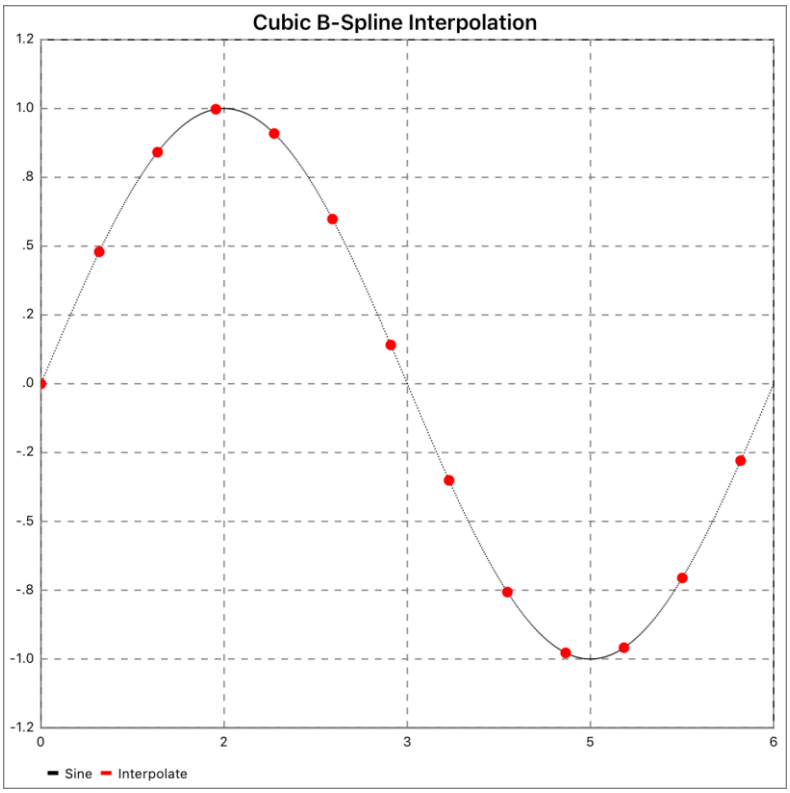

Figure 76 - Interpolate Cubic B-Spline Plot

6.12.2 Constants

6.12.2.1 Interpolate.BARYCENTRIC

`Interpolate.BARYCENTRIC` – Interpolates using the Barycentric method

6.12.2.2 Interpolate.CUBIC_B_SPLINE

`Interpolate.CUBIC_B_SPLINE` – Interpolates using the Cubic B-Spline method

6.13 Integration

The integrate `f(x)` function is the calculation of the area under the curve.

6.13.1 integrate

`integrate(f, min, max)` – Returns area under the curve
`integrate(f, min, max, t)`
`Integrate(f, min, max, t, r)`

The `f` argument is the function used to integrate (i.e., calculating the area under the curve) between minimum's `min` argument and maximum's `max` argument.

If the `t` argument is not specified, the tolerance of the step-wise integration of the function's default is `1.0E-6`.

If the `r` argument is not specified maximum, the number of refinements in the number of times the interval may be halved. The default maximum number of refinements is `12`.

⇒ NOTE: You can use the alternate function name: `Integrate.trapezoidal`.

⇒ LINK: https://en.wikipedia.org/wiki/Numerical_integration

For example:

- `integrate(sine, 0, PI)` Returns `2`
- `integrate(sine, 0, PI, 1.0E-6)` Returns `2`
- `integrate(sine, 0, PI, 1.0E-6,12)` Returns `2`

6.13.2 Integrate.romberg

`Integrate.romberg(f, min, max)` – Returns area under the curve
`Integrate.romberg(f, min, max, n)`

The `f` argument is the function used to integrate between minimum's `min` argument and maximum's `max` argument.

If the `n` argument is not specified, it will be divided $2^n + 1$ times between minimum and maximum range. The default maximum number of points is `5`. Maximum value is `100`.

⇒ LINK: https://en.wikipedia.org/wiki/Romberg%27s_method

For example:

- `Integrate.romberg(sine, 0, PI)` Returns `2`
- `Integrate.romberg(sine, 0, PI, 5)` Returns `2`

6.14 Color

The *Color*'s constants in the table below can be used for *Dataset*, *Plot*, and *Graph* section fields.

The `RGB` is represented by red, green, and blue colors ranging from `0.0` to `1.0`. The default color is black, whereas black's `RGB` is represented as `[0.0, 0.0, 0.0]` to white `[1.0, 1.0, 1.0]` of the chromatic color spectrum.

Constant	Red	Green	Blue
Color.BLACK	0.000	0.000	0.000
Color.WHITE	1.000	1.000	1.000
Color.RED	1.000	0.000	0.000
Color.GREEN	0.000	1.000	0.000
Color.BLUE	0.000	0.000	1.000
Color.SILVER	0.839	0.839	0.839
Color.GRAY	0.500	0.500	0.500
Color.LIGHT_GRAY	0.749	0.749	0.749
Color.YELLOW	1.000	1.000	0.000
Color.BROWN	0.820	0.410	0.536
Color.PINK	1.000	0.711	0.754
Color.PURPLE	0.500	0.000	0.500
Color.MAGENTA	1.000	0.000	1.000
Color.ORANGE	1.000	0.715	0.754

Table 24 - Color Constants

6.15 Dataset

The *Dataset* is created by the `Create.dataset` function to create a new variable to hold the plotting `(x, y)` data pairs. This is a container holding array of `x` and `y` values used for plotting.

The following sub-section examples will use the following dataset variable:

```
data = Create.dataset()

a = [[ 1,  2,  3],                                                    \
     [10, 11, 12]]

data.set(a)
```

6.15.1 Intrinsic

6.15.1.1 Functions

6.15.1.1.1 .set

`v.set(a)` – Sets dataset array containing `x` and `y` rows of data
`v.set(x, y)` – Sets dataset containing `x` and `y` vector data

The `a` argument is an array containing `x` data in the first row and `y` data in the second row; otherwise, the `x` and `y` arguments vectors are `(x, y)` data pairs.

For example, using an array:

- `d.set(a)` Results in setting array's `(x, y)` data pairs

For example, using `x` and `y` vectors:

- `data = Create.dataset()`

- `x = [0 : PI : PI ÷ 300]`
- `y = sin(x)`

- `data.set(x, y)` Results in setting two vectors as `(x, y)` data pairs

6.15.1.1.2 .point

`v.point(x, y)` – Sets dataset to single point `(x, y)` data pair

The `x` and `y` argument single `(x, y)` data pair to be saved into the dataset.

For example:

- `d.point(1, 10)` Results in setting single data point

6.15.1.1.3 .clear

`v.clear()` – Clears all `(x, y)` data pairs from the dataset

For example:

- `d.clear()` Results in clearing all `(x, y)` data pairs

6.15.1.2 Fields

6.15.1.2.1 .size

`v.size` – Returns number of `(x, y)` data pairs

For example:

- `d.size` Returns 3

6.15.1.2.2 .info

`v.info` – Returns textual information about the dataset

6.15.1.2.3 .name

`v.name` – Sets or gets the name of the `Dataset`

The default name is "`Dataset n`", where *n* is number of the dataset.

6.15.1.2.4 .color

`v.color` – Sets or gets dataset's `RGB` color

The default color is black (i.e., `Color.BLACK`).

⇒ NOTE: Use `color.red`, `color.green`, and `color.blue` fields to access sub-color values associated with the `color` field.

For example:

- `d.color = [0.5,0.5,0.5]` Results in gray color
- `d.color = Color.GRAY` Results in gray color

Alternatively:

- `d.color.red = 0.5` Results in gray color
  `d.color.green = 0.5`
  `d.color.blue = 0.5`

6.15.1.2.5 .thickness

`v.thickness` – Sets or gets thickness of the line plot

The default thickness is `0.5`.

6.15.1.2.6 .line.style

`v.line.style` – Sets or gets the style of plot to be drawn

The default style is `Plot.LINE.SOLID`. Other possible styles are `Plot.LINE.DASHES` and `Plot.LINE.DOTS`.

For example:

- `d.line.style = Plot.LINE.SOLID`
- `d.line.style = Plot.LINE.DASHES`
- `d.line.style = Plot.LINE.DOTS`

6.15.1.2.7 .line.display_values

`v.line.display_values` – Sets or gets boolean indicator of displaying value hovering over `(x, y)` data pair

The default value is `false`.

6.15.1.2.8 .line.display_points

`v.line.display_points` – Sets or gets boolean indicator display data point's `y` value

The default value is `false`.

6.15.1.2.9 .scatter.symbol

`v.scatter.symbol` – Sets or gets scatter type symbol used to display the `(x, y)` data point

The default symbol is `Plot.SCATTER.CIRCLE`. Other possible symbols are `Plot.SCATTER.CROSS`, `Plot.SCATTER.PLUS`, `Plot.SCATTER.SQUARE`, and `Plot.SCATTER.TRIANGLE`.

For example:

- `d.scatter.symbol = Plot.SCATTER.CIRCLE`
- `d.scatter.symbol = Plot.SCATTER.CROSS`
- `d.scatter.symbol = Plot.SCATTER.PLUS`
- `d.scatter.symbol = Plot.SCATTER.SQUARE`
- `d.scatter.symbol = Plot.SCATTER.TRIANGLE`

6.15.1.2.10 .scatter.display_values

`v.scatter.display_values` – Sets or gets boolean indicator of displaying value hovering over `(x, y)` data point

The default value is `false`.

6.15.1.2.11 .bar.display_values

`v.bar.display_values` – Sets or gets boolean indicator of displaying value hovering over bar

The default value is `false`.

6.16 Plot

The *Plot* is created by the *Create.plot* function to create a new plot variable to hold a collection of datasets created by the *Create.plot* function. This is used to plot one or more datasets.

The following sub-section examples will use the following dataset variable:

```
d1 = Create.dataset()

a  = [[ 1,  2,  3],                    \
      [10, 11, 12]]

d1.set(a)
d1.name = "myData"

b  = [[ 1,  2,  3,  4],                \
      [10, 11, 12, 13]]

d2 = Create.dataset()

d2.set(b)
d2.name = "myData"

p  = Create.plot()
```

6.16.1 Intrinsic

6.16.1.1 Functions

6.16.1.1.1 .add

`v.add(d)` – Adds dataset

For example:

- `p.add(d1)` Results in adding dataset to the plot variable

6.16.1.1.2 .get

`v.get(n)` – Returns named dataset

For example:

- `p.get("myData")` Returns dataset named `myData`

6.16.1.1.3 .set

`v.set(n, d1)` – Replaces named dataset

For example:

- `p.set("myData", d2)` Results in replacing a new dataset

6.16.1.1.4 .remove

`v.remove(n)` – Removes named dataset

For example:

- `p.remove("myData")` Results in removal of the `myData` dataset

6.16.1.1.5 .clear

`v.clear()` – Removes all datasets

For example:

- `p.clear()` Results in all datasets in the plot being removed

6.16.1.2 Fields

6.16.1.2.1 .size

`v.size` – Returns number of datasets in the plot

For example:

- `p.add(d1)` Results in adding dataset to the plot variable
- `p.size` Returns size of `3`

6.16.1.2.2 .info

`v.info` – Returns textual information about the plot

6.16.1.2.3 .name

`v.name` – Sets or gets the name of the plot

The default name is "`Plot` *`n`*", where *`n`* is number of the plot.

For example:

- `p.add(d1)` Results in adding the dataset to the plot variable

- `p.name` Returns `myData` string

6.16.1.2.4 .type

`v.type` – Sets or gets the type of plot to be drawn

The default type is `Plot.CHART.LINE`. Other possible types are `Plot.CHART.BAR` and `Plot.CHART.SCATTER`.

6.16.1.2.5 .title.name

`v.title.name` – Sets or gets the title displayed at the top section of the plot

6.16.1.2.6 .title.offset

`v.title.offset` – Sets or gets the title offset from the top

The default offset is `10`.

6.16.1.2.7 .title.position

`v.title.position` – Sets or gets the title's position

The default position is `Plot.POSITION.CENTER`. Other possible types are `Plot.POSITION.LEFT` or `Plot.POSITION.RIGHT`.

6.16.1.2.8 .title.color

`v.title.color` – Sets or gets title's `RGB` color

The default color is black (i.e., `Color.BLACK`).

⇒ NOTE: Use `title.color.red`, `title.color.green`, and `title.color.blue` fields to access sub-color values associated with the `title.color` field.

For example:

- `v.title.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.title.color = Color.GRAY` Results in gray color

Alternatively:

- `v.title.color.red = 0.5` Results in gray color
  `v.title.color.green = 0.5`
  `v.title.color.blue = 0.5`

6.16.1.2.9 .bg.color

`v.bg.color` – Sets or gets plot's `RGB` background color

The default color is black (i.e., `Color.SILVER`).

⇒ NOTE: Use `bg.color.red`, `bg.color.green`, and `bg.color.blue` fields to access sub-color values associated with the `bg.color` field.

For example:

- `v.bg.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.bg.color = Color.GRAY` Results in gray color

Alternatively:

- `v.bg.color.red = 0.5` Results in gray color
  `v.bg.color.green = 0.5`
  `v.bg.color.blue = 0.5`

6.16.1.2.10 .chart.bg.color

`v.chart.bg.color` – Sets or gets plot's drawing area `RGB` background color

The default color is black (i.e., `Color.LIGHT_GRAY`).

⇒ NOTE: Use `chart.bg.color.red`, `chart.bg.color.green`, and `chart.bg.color.blue` fields to access sub-color values associated with the `chart.bg.color` field.

For example:

- `v.chart.bg.color = [0.5,0.5,0.5]` Results in gray color
- `v.chart.bg.color = Color.GRAY` Results in gray color

Alternatively:

- `v.chart.bg.color.red = 0.5` Results in gray color
  `v.chart.bg.color.green = 0.5`
  `v.chart.bg.color.blue = 0.5`

6.16.1.2.11 .chart.fg.color

`v.chart.fg.color` – Sets or gets plot's drawing area `RGB` foreground color

The default color is black (i.e., `Color.LIGHT_GRAY`).

⇒ NOTE: Use `chart.fg.color.red`, `chart.fg.color.green`, and `chart.fg.color.blue` fields to access sub-color values associated with the `chart.fg.color` field.

For example:

- `v.chart_fg.color = [0.5,0.5,0.5]` Results in gray color
- `v.chart.fg.color = Color.GRAY` Results in gray color

Alternatively:

- `v.chart.fg.color.red = 0.5` Results in gray color
  `v.chart.fg.color.green = 0.5`
  `v.chart.fg.color.blue = 0.5`

6.16.1.2.12 .animate

`v.animate` – Animates displayed plot

The default value is `0.0`. Zero indicates that there is no drawing animation; otherwise, the value specifies how many seconds it will take to draw the plot.

6.16.1.2.13 .xaxis.grid.color

`v.xaxis.grid.color` – Sets or gets x-axis grid's `RGB` color

The default color is white (i.e., `Color.WHITE`).

⇒ NOTE: Use `xaxis.grid.color.red`, `xaxis.grid.color.green`, and `xaxis.grid.color.blue` fields to access sub-color values associated with the `xaxis.grid.color` field.

For example:

- `v.xaxis.grid.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.xaxis.grid.color = Color.GRAY` Results in gray color

Alternatively:

- `v.xaxis.grid.color.red = 0.5` Results in gray color
  `v.xaxis.grid.color.green = 0.5`
  `v.xaxis.grid.color.blue = 0.5`

6.16.1.2.14 .xaxis.grid.thickness

`v.xaxis.grid.thickness` – Sets or gets thickness of the x-axis grid line

The default thickness is `0.5`.

6.16.1.2.15 .xaxis.grid.style

`v.xaxis.grid.style` – Sets or gets x-axis grid line style

The default style is dashes (i.e., `Plot.LINE.DASHES`). Other possible styles are `Plot.LINE.DASHES` and `Plot.LINE.DOTS`.

6.16.1.2.16 .xaxis.range.min

`xaxis.range.min` – Sets or gets the x-axis minimum range

The default minimum value is calculated from the initial inputted datasets.

6.16.1.2.17 .xaxis.range.max

`xaxis.range.max` – Sets or gets the x-axis maximum range

The default maximum value is calculated from the initial inputted datasets.

6.16.1.2.18 .xaxis.labels.count

`xaxis.labels.count` – Sets or gets the x-axis number of labels along major grid points to be displayed

The default number of labels set to zero, indicating it will compute the best number of labels to be displayed on the x-axis.

6.16.1.2.19 .xaxis.labels.color

`v.xaxis.labels.color` – Sets or gets x-axis labeled number's `RGB` color

The default color is black (i.e., `Color.BLACK`).

⇒ NOTE: Use `xaxis.labels.color.red`, `xaxis.labels.color.green`, and `xaxis.labels.color.blue` fields to access sub-color values associated with the `xaxis.labels.color` field.

For example:

- `v.xaxis.labels.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.xaxis.labels.color = Color.GRAY` Results in gray color

Alternatively:

- ```
  v.xaxis.labels.color.red   = 0.5
  v.xaxis.labels.color.green = 0.5
  v.xaxis.labels.color.blue  = 0.5
  ```
  Results in gray color
  ```

6.16.1.2.20 .xaxis.title.name

`v.xaxis.title.name` – Sets or gets the x-axis title displayed at the bottom section of the plot

6.16.1.2.21 .xaxis.title.color

`v.xaxis.title.color` – Sets or gets x-axis title's `RGB` color

The default color is black (i.e., `Color.BLACK`).

⇒ NOTE: Use `xaxis.title.color.red`, `xaxis.title.color.green`, and `xaxis.title.color.blue` fields to access sub-color values associated with the `xaxis.title.color` field.

For example:

- `v.xaxis.title.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.xaxis.title.color = Color.GRAY` Results in gray color

Alternatively:

- ```
  v.xaxis.title.color.red   = 0.5
  v.xaxis.title.color.green = 0.5
  v.xaxis.title.color.blue  = 0.5
  ```
  Results in gray color

6.16.1.2.22 .xaxis.title.offset

`v.xaxis.title.offset` – Sets or gets the x-axis title offset from the bottom section of the plot

The default offset is `10`.

6.16.1.2.23 .xaxis.title.position

`v.xaxis.title.position` – Sets or gets the x-axis title's position

The default position is `Plot.POSITION.CENTER`. Other possible types are `Plot.POSITION.LEFT` or `Plot.POSITION.RIGHT`.

6.16.1.2.24 .yaxis.grid.color

`v.xaxis.grid.color` – Sets or gets y-axis grid's `RGB` color

The default color is white (i.e., `Color.WHITE`).

⇒ NOTE: Use `yaxis.grid.color.red`, `yaxis.grid.color.green`, and `yaxis.grid.color.blue` fields to access sub-color values associated with the `yaxis.grid.color` field.
  ```

For example:

- `v.yaxis.grid.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.yaxis.grid.color = Color.GRAY` Results in gray color

Alternatively:

- `v.yaxis.grid.color.red = 0.5`
  `v.yaxis.grid.color.green = 0.5`
  `v.yaxis.grid.color.blue = 0.5`
  Results in gray color

6.16.1.2.25 .yaxis.grid.thickness

`v.yaxis.grid.thickness` – Sets or gets thickness of the y-axis grid line

The default thickness is `0.5`.

6.16.1.2.26 .yaxis.grid.style

`v.yaxis.grid.style` – Sets or gets y-axis grid line style

The default style is dashes (i.e., `Plot.LINE.DASHES`). Other possible styles are `Plot.LINE.DASHES` and `Plot.LINE.DOTS`.

6.16.1.2.27 .yaxis.range.min

`yaxis.range.min` – Sets or gets the y-axis minimum range

The default minimum value is calculated from the initial inputted datasets.

6.16.1.2.28 .yaxis.range.max

`yaxis.range.max` – Sets or gets the y-axis maximum range

The default maximum value is calculated from the initial inputted datasets.

6.16.1.2.29 .yaxis.labels.count

`yaxis.labels.count` – Sets or gets the y-axis number of labels along major grid points to be displayed

The default number of labels set to zero, indicating it will compute the best number of labels to be displayed on the y-axis.

6.16.1.2.30 .yaxis.labels.color

`v.yaxis.labels.color` – Sets or gets y-axis labeled number's `RGB` color

The default color is black (i.e., `Color.BLACK`).

⇒ NOTE: Use `yaxis.labels.color.red`, `yaxis.labels.color.green`, and `yaxis.labels.color.blue` fields to access sub-color values associated with the `yaxis.labels.color` field.

For example:

- `v.yaxis.labels.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.yaxis.labels.color = Color.GRAY` Results in gray color

Alternatively:

- `v.yaxis.labels.color.red = 0.5`
  `v.yaxis.labels.color.green = 0.5`
  `v.yaxis.labels.color.blue = 0.5`
  Results in gray color

6.16.1.2.31 .yaxis.title.name

`v.xaxis.title.name` – Sets or gets the y-axis title displayed at the left section of the plot

6.16.1.2.32 .yaxis.title.color

`v.xaxis.title.color` – Sets or gets y-axis title's `RGB` color

The default color is black (i.e., `Color.BLACK`).

⇒ NOTE: Use `yaxis.title.color.red`, `yaxis.title.color.green`, and `yaxis.title.color.blue` fields to access sub-color values associated with the `yaxis.title.color` field.

For example:

- `v.yaxis.title.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.yaxis.title.color = Color.GRAY` Results in gray color

Alternatively:

- `v.yaxis.title.color.red = 0.5`
  `v.yaxis.title.color.green = 0.5`
  `v.yaxis.title.color.blue = 0.5`
  Results in gray color

6.16.1.2.33 .yaxis.title.offset

`v.yaxis.title.offset` – Sets or gets the y-axis title offset from the left section of the plot

The default offset is `10`.

6.16.1.2.34 .yaxis.title.position

`v.yaxis.title.position` – Sets or gets the y-axis title's position

The default position is `Plot.POSITION.CENTER`. Other possible types are `Plot.POSITION.TOP` or `Plot.POSITION.BOTTOM`.

6.16.1.2.35 .pad.top

`v.pad.top` – Sets or gets plotting area's padding offset from the top of the plot

The default is `0.0`.

6.16.1.2.36 .pad.bottom

`v.pad.bottom` – Sets or gets plotting area's padding offset from the bottom of the plot

The default is `0.0`.

6.16.1.2.37 .pad.left

`v.pad.left` – Sets or gets plotting area's padding offset from the left of the plot

The default is `0.0`.

6.16.1.2.38 .pad.right

`v.pad.right` – Sets or gets plotting area's padding offset from the right of the plot

The default is `0.0`.

6.16.1.2.39 .legend.show

`v.legend.show` – Sets or gets boolean indicator of showing or hiding the plotting data's legends

The default is `true`.

6.16.1.2.40 .legend.color

`v.legend.color` – Sets or gets legend's `RGB` color

The default color is black (i.e., `Color.BLACK`).

⇒ NOTE: Use `legend.color.red`, `legend.color.green`, and `legend.color.blue` fields to access sub-color values associated with the `legend.color` field.

For example:

- `v.legend.color = [0.5, 0.5, 0.5]` Results in gray color
- `v.legend.color = Color.GRAY` Results in gray color

Alternatively:

- `v.legend.color.red   = 0.5`
  `v.legend.color.green = 0.5`
  `v.legend.color.blue  = 0.5`
  Results in gray color

6.16.2 Constants

6.16.2.1 Plot.CHART.LINE

`Plot.CHART.LINE` – Displays line chart constant

6.16.2.2 Plot.CHART.BAR

`Plot.CHART.BAR` – Displays bar chart constant

6.16.2.3 Plot.CHART.SCATTER

`Plot.CHART.SCATTER` – Displays scatter chart constant

6.16.2.4 Plot.LINE.DASHES

`Plot.LINE.DASHES` – Draws dashes in the line constant

6.16.2.5 Plot.LINE.DOTS

`Plot.LINE.DOTS` – Draws dots in the line constant

6.16.2.6 Plot.LINE.SOLID

`Plot.LINE.SOLID` – Draws a solid line constant

6.16.2.7 Plot.SCATTER.CIRCLE

`Plot.SCATTER.CIRCLE` – Draws scatter point as a circle constant

6.16.2.8 Plot.SCATTER.CROSS

`Plot.SCATTER.CROSS` – Draws scatter point as a cross constant

6.16.2.9 Plot.SCATTER.PLUS

`Plot.SCATTER.PLUS` – Draws scatter point as a plus constant

6.16.2.10 Plot.SCATTER.SQUARE

`Plot.SCATTER.SQUARE` – Draws scatter point as a square constant

6.16.2.11 Plot.SCATTER.TRIANGLE

`Plot.SCATTER.TRIANGLE` – Draws scatter point as a triangle constant

6.16.2.12 Plot.POSITION.TOP

`Plot.POSITION.TOP` – Positions the title at the top constant

6.16.2.13 Plot.POSITION.BOTTOM

`Plot.POSITION.BOTTOM` – Positions the title at the bottom constant

6.16.2.14 Plot.POSITION.CENTER

`Plot.POSITION.CENTER` – Positions the title at the center constant

6.16.2.15 Plot.POSITION.LEFT

`Plot.POSITION.LEFT` – Positions the title at the left constant

6.16.2.16 Plot.POSITION.RIGHT

`Plot.POSITION.RIGHT` – Positions the title at the right constant

6.17 Graph

The *Graph* is created by the *Create.graph* function to create a new graph variable to hold a collection of plots created by the *Create.*graph*plot*Create.plot function. This is used to graph one or more plots.

The following sub-section examples will use the following graph variable:

```
d1 = Create.dataset()

a  = [[ 1,  2,  3],                    \
      [ 10, 11, 12]]

d1.set(a)
d1.name = "myData"

b  = [[ 1,  2,  3,  4],                \
      [ 10, 11, 12, 13]]
```

```
d2 = Create.dataset()

d2.set(b)
d2.name = "myData"

p1  = Create.plot()
p1.name = "myPlot"
p1.add(d1)

p2  = Create.plot()
p2.name = "myPlot"
p2.add(d2)

g = Create.graph()
```

6.17.1 Graph.list

`Graph.list()` – Returns a list of all graph names

6.17.2 Graph.delete

`Graph.delete(`g_1`, `g_2`, ..., `g_n`)` – Deletes named graphs

The g_1, g_2, ..., g_n arguments are lists of graph names to be removed.

6.17.3 Intrinsic

6.17.3.1 Functions

6.17.3.1.1 .add

`v.add(d)` – Adds plot

For example:

- `g.add(p1)` Results in adding plot to the graph variable

6.17.3.1.2 .get

`v.get(n)` – Returns named plot

For example:

- `g.get("myPlot")` Returns plot named `myPlot`

6.17.3.1.3 .set

`v.set(n, p)` – Replaces named plot

For example:

- `g.set("myPlot", p2)` Results in replacing a new plot

6.17.3.1.4 .remove

`v.remove(n)` – Removes named plot

For example:

- `g.remove("myPlot")` Results in the removal of `myPlot` plot

6.17.3.1.5 .plot

`v.plot(a)` – Plots array containing `x` and `y` rows of data
`v.plot(x, y1, y2, ..., yn)`

The `a` argument is an array containing `x` data in the first row and `y` data in the second row; otherwise, the `x` and `y` arguments vectors are `(x, y)` data pairs.

For example, using an array:

- `g = Create.graph()`
- `g.plot(a)` Results in a plot of the array's `(x, y)` data pairs

For example, using `x` and `y` vectors:

- `g = Create.graph()`
- `x = [0 : PI × 2 : (PI × 2) ÷ 300]`
- `y = sin(x)`
- `g.plot(x, y)` Results in a plot of two vectors as `(x, y)` data pairs

Another example using `x` and y_1 and y_2 vectors:

- `g = Create.graph()`
- `x = [0 : PI × 2 : (PI × 2) ÷ 300]`
- `g.plot(x, sin(x), cos(x))` Results in two plots of sine and cosine

6.17.3.1.6 .save

`v.save(n)` – Saves named graph

The `n` argument is the name graph saved in the Graphs tab.

For example:

- `g.save("myGraph")` Results in displaying `myGraph` in the Graphs tab

6.17.3.1.7 .clear

`v.clear()` – Removes all plots

For example:

- `g.clear()` Results in all plots in the graph being removed

6.17.3.2 Fields

6.17.3.2.1 .size

`v.size` – Returns number of datasets in the plot

For example:

- `g.add(p1)` Results in adding plot to the graph variable
- `g.size` Returns size of `1`

6.17.3.2.2 .info

`v.info` – Returns textual information about the graph

6.17.3.2.3 .name

`v.name` – Sets or gets the name of the `Graph`

For example:

- `g.name` Returns the name of the graph

6.18 String

6.18.1 Intrinsic

6.18.1.1 Functions

6.18.1.1.1 .lower

`v.lower()` – Returns string to lower case characters

For example:

- `s = "Jack and Jill"`
- `s.lower()` Returns `"jack and jill"`

- `s = ["Jack", "Jill"]`
- `s.lower()` Returns `["jack", "jill"]`

⇒ NOTE: The *String* library only works for ANIS 8-bit characters. It does not support Unicode Transformation Format (UTF) 16-bit characters.

6.18.1.1.2 .upper

`v.upper()` – Returns string to upper case characters

For example:

- `s = "Jack and Jill"`
- `s.upper()` Returns `"JACK AND JILL"`

- `s = ["Jack", "Jill"]`
- `s.upper()` Returns `["JACK", "JILL"]`

6.18.1.1.3 .concat

`v.concat(s)` – Returns concatenated string

The `s` argument is the string to be concatenated to the string's variable.

For example:

- `s = "Jack"`
- `s.concat("+Jill")` Returns `"Jack+Jill"`

- `s = ["Jack", "Jill"]`
- `s.concat("+1")` Returns `["Jack+1", "Jill+1"]`
- `s.concat(["+1","+2"])` Returns `["Jack+1", "Jill+2"]`

6.18.1.1.4 .prefix

`v.prefix(s)` – Returns a boolean indicator if matches beginning of the string

The `s` argument is the sub-string to find matching characters at the beginning of the string's variable.

For example:

- `s = "Jack and Jill"`
- `s.prefix("Jack")` Returns `true`

- `s.prefix("Jill")` Returns `false`

- `s = ["Jack", "Jill"]`
- `s.prefix("Jack")` Returns `[true, false]`
- `s.prefix("Jill")` Returns `[false, true]`

6.18.1.1.5 .postfix

`v.postfix(s)` – Returns a boolean indicator if matches end of the string

The `s` argument is the sub-string to find matching characters at the end of the string's variable.

For example:

- `s = "Jack and Jill"`
- `s.postfix("Jack")` Returns `false`
- `s.postfix("Jill")` Returns `true`

- `s = ["Jack", "Jill"]`
- `s.postfix("Jack")` Returns `[true, false]`
- `s.postfix("Jill")` Returns `[false, true]`

6.18.1.1.6 .find

`v.find(f)` – Returns character index of the first matching sub-string
`v.find(f, i)` – Returns character index of the matching sub-string starting at the i^{th} character

The `f` argument is the sub-string used to find matching characters in the string's variable.

If the `i` argument is not specified, the starting character index starts at the first character at zero index.

If the matching characters is not found in the string, then it will return `-1`.

For example:

- `s = "Jack and Jill"`
- `s.find("Jill")` Returns `9`
- `s.find("Jill", 4)` Returns `9`
- `s.find("Jill", 15)` Returns `-1`

- `s = ["Jack", "Jill"]`
- `s.find("Jack")` Returns `[0, -1]`

6.18.1.1.7 .findall

`v.findAll(f)` – Returns indexes of the all matching sub-strings

The `f` argument is the sub-string used to find all matching characters in the string's variable.

If the matching characters not found in the string, then it will return `-1`.

⇒ NOTE: If the `v` variable is an array, it will only find first instance for each element in the array.

For example:

```
p = "Jack and Jill went up the hill\n"      + \
    "To fetch a pail of water;\n"           + \
    "Jack fell down and broke his crown,\n" + \
    "and Jill came tumbling after.\n"

a = [ "Jack and Jill went up the hill",       \
      "To fetch a pail of water;",            \
      "Jack fell down and broke his crown,",  \
      "and Jill came tumbling after." ]
```

- `p.findAll("and")` Returns `[5, 72, 93]`
- `a.findAll("and")` Returns `[5, -1, 15, 0]`

6.18.1.1.8 .substr

`v.substr(i)` – Returns sub-string of the string
`v.substr(i, l)`

The `i` argument is the character index into the string's variable to extract number of characters from the `l` argument's length.

If the `l` argument is not specified, it will extract the sub-string from the character index to end of the string.

For example, using previous poem's variables:

- `p.substr(5, 3)` Returns `"and"`

- `i = p.find("and")` Returns `5`
- `p.substr(i, 3)` Returns `"and"`

- `i = a.find("and")` Returns `[5, -1, 15, 0]`
- `a.substr(i, 3)` Returns `["and", "", "and", "and"]`

6.18.1.1.9 .insert

`v.insert(s, i)` – Returns inserted sub-string into the string's variable

The `s` argument is the sub-string to be inserted at `i` argument's character index into the string's variable.

For example:

- `s = "Jack and Jill"`
- `i = s.find("Jack")` Returns `0`

- `s.insert("ie", i + 4)` Returns `"Jackie and Jill"`
- `s.insert(".", s.length)` Returns `"Jack and Jill."`

- `s = ["See Jack run up the hill.", "See Jack roll down the hill."]`

- `i = s.find("Jack")` Returns `[4, 4]`
- `s.insert(" & Jill", i+4)` Returns
  ```
  [ "See Jack & Jill run up the hill.",
    "See Jack & Jill roll down the hill." ]
  ```

6.18.1.1.10 .replace

`v.replace(s, r)` – Returns first replaced sub-string into the string's variable
`v.replace(s, r, n)` – Returns replaced n^{th} occurrence of the sub-string into the string's variable

The `s` argument is the sub-string to be replaced by `r` argument's sub-string at `n` argument's occurrence into the string's variable.

If the `n` argument is not specified, it will replace the first occurrence of the sub-string.

For example:

- `s = "Jack and Jill and the dog"`

- `s.replace("and", "&")` Returns `"Jack & Jill and the dog"`
- `s.replace("and", "&", 1)` Returns `"Jack and Jill & the dog"`

- `s = [ "Jack and Jill", "Tom and Jack" ]`

- `s.replace("Jack","Jerry")` Returns `[ "Jerry and Jill", "Tom and Jerry" ]`

6.18.1.1.11 .capitalize

`v.capitalize()` – Returns string to capitalize all the words

For example:

- `s = "jack and jill"`
- `s.capitalize()` Returns `"Jack And Jill"`

6.18.1.1.12 .sentence

`v.sentence()` – Returns string to capitalized all words beginning of a sentence

For example:

- `s = "jack and jill"`
- `s.sentence()` Returns `"Jack and jill"`

6.18.1.2 Fields

6.18.1.2.1 .length

`v.length` – Returns the length of the string

For example:

- `s = "Jack and Jill"`
- `s.length` Returns `13`

- `s = [ "Jack and Jill", "Tom & Jerry" ]`
- `s.length` Returns `[ 13, 11 ]`

6.19 Grep

General Regular Expression Print (GREP) is used to find occurrence of a string of characters that matches a specified pattern of characters sequence rather than simple sequence of character-by-character matching of a sub-string.

The regular expression meta characters used to control pattern matching are as follow: \ ^ $. [ ] | ( ) * + ?

Here is a summary of regular expressions used to find matching sub-strings:

Syntax	Description
c	Matches the non-meta character *c* (i.e., `a` to `z`, `A` to `Z`, `0` to `9`, and etc.)
`\`*c*	Matches literal character *c* (i.e., `\t` for tab, `\n` for newline, `\r` for return)
`^`	Matches the beginning of a string
$	Matches the end of string
`.`	Matches any single character
[abc…]	Matches any characters of *abc…*
[^abc…]	Matches any characters but *abc…*
r1 | r2	Matches any string containing *r1* or *r2* where as *r* is the syntax described above
(r)	Grouping of string match by *r*
*(r)**	Matches zero or more consecutive string matched by *r*
(r)+	Matches one or more consecutive string matched by *r*
(r)?	Matches the null string or one string matched by *r*

Table 25 - GREP Syntax

⇒ NOTE: The *Grep* library only works for ANIS 8-bit characters. It does not support Unicode Transformation Format (UTF) 16-bit characters.

Summary of macros that is expanded into the pattern string for sub-string matching:

Macro	Description	GREP
\w	clear	[\t]+
\a	alpha	[a-zA-Z]+
\s	symbol	[a-zA-Z\$_]+
\i	integer	[\t]*[-|+]?[0-9]+[lL]?[\t]*
\o	octal	[\t]*[0][0-7]+[\t]*
\h	hexadecimal	[\t]*[0][xX][a-fA-F0-9]+[\t]*
\d	decimal	[\t]*[0-9]*[.][0-9]+[\t]*
\f	real	[\t]*[-|+]?[0-9]+[.][0-9]*([e|E][-|+]?[0-9]+)?[\t]*| [\t]*[-|+]?[.][0-9]+([e|E][-|+]?[0-9]+)?[\t]*| [\t]*[-|+]?[0-9]+([e|E][-|+]?[0-9]+)?[\t]*
\c	complex	[([\t]*\f[\t]*[,][\t]*\f[\t]*[)]| [([\t]*\i[\t]*[,][\t]*\i[\t]*[)]| [([\t]*\f[\t]*[,][\t]*\i[\t]*[)]| [([\t]*\i[\t]*[,][\t]*\f[\t]*[)]

Table 26 - GREP Macros

6.19.1 Intrinsic

6.19.1.1 Functions

6.19.1.1.1 .set

`v.set(p)` – Returns a boolean indicator if it's compiles the pattern string's expression

The `p` argument's pattern string is used to compare matching string.

For example:

- `g = Create.Grep()` — Create grep variable
- `g.set("[0123456789]")` — Returns `true` to match any characters containing `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, or `9`
- `g.set("[0-9]")` — Returns `true` to match any characters containing `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, or `9`
- `g.set("[0-9]+")` — Returns `true` to match any one or more characters containing `0` to `9`
- `g.set("\i")` — Returns `true` using `\i` macro to match any one or more characters representing an integer containing `0` to `9` with optional prefixed and postfixed spaces along `+` or `−` prefix and postfixed `l` or `L` character indicating long integer.

6.19.1.1.2 .get

`v.get()` – Returns the pattern string's expression

For example:

- `g = Create.Grep()` Create grep variable
- `v.set("[0-9]+")` Returns `true` to match any one or more characters containing `0` to `9`
- `v.get()` Returns `"[0-9]+"` pattern string expression

6.19.1.1.3 .match

`v.match(s)` – Returns a boolean indicator if the pattern matches the string
`v.match(s, i)`

The `s` argument's string is used to match against the pattern's string expression.

If the `i` argument is not specified, it will start the pattern matching at the beginning of the string. Otherwise, it will start at the index position of the string to start the comparison.

For example:

- `g = Create.Grep("([0-9]+)[/]([0-9]+)[/]([0-9]+)")`
- `g.match("12/24/2022")` Returns `true`
- `g.match("12-24-2022")` Returns `false`

⇒ NOTE: The maximum number of groups can be defined is 10. Any henceforth groups are matched, but not accessible from the `group` and `indicies` methods.

This example will create 4 groups. The 1st group is the whole string, and the 2nd, 3rd, and 4th groups will match the month, day, and year sub-strings respectively.

6.19.1.1.4 .search

`v.search(s)` – Returns a boolean indicator if the pattern matches a sub-string in the string
`v.search(s, i)`

The `s` argument's string is used to find a sub-string match against the pattern's string expression.

If the `i` argument is not specified, it will start the pattern matching at the beginning of the string. Otherwise, it will start at the index position of the string to start the comparison.

For example:

- `g = Create.Grep("([0-9]+)[/]([0-9]+)[/]([0-9]+)")`
- `g.search("Date: 12/24/2022")` Returns `true`
- `g.search("Date: 12-24-2022")` Returns `false`

6.19.1.1.5 .group

`v.group(i)` – Returns nth sub-string found by the pattern string's expression

The `i` argument's index of the n[th] sub-string group found by the pattern string's expression. The `i` argument's range is from `0` to `9`. Maximum number of groups is `10`.

For example:

- `g = Create.Grep("([0-9]+)[/]([0-9]+)[/]([0-9]+)")`
- `g.match("12/24/2022")` Returns `true`
- `g.group(0)` Returns `"12/24/2022"`
- `g.group(1)` Returns `"12"`
- `g.group(2)` Returns `"24"`
- `g.group(3)` Returns `"2022"`

6.19.1.1.6 .indicies

`v.indices(i)` – Returns n[th] start and ending string positions of the sub-string found by the pattern string's expression

The `i` argument's index of the n[th] sub-string group found by the pattern string's expression. The `i` argument's range is from `0` to `9`. Maximum number of groups is `10`.

For example:

- `g = Create.Grep("([0-9]+)[/]([0-9]+)[/]([0-9]+)")`
- `g.match("12/24/2022")` Returns `true`
- `g.indices(0)` Returns `[0,9]`
- `g.indices(1)` Returns `[0 1]`
- `g.indices(2)` Returns `[3,4]`
- `g.indices(3)` Returns `[6,9]`

6.19.1.1.7 .clear

`v.clear()` – Clear the pattern string's expression and any previous string matching or searching results

6.19.1.2 Fields

6.19.1.2.1 .count

`v.count` – Returns number of sub-string groups found from the pattern string's expression

For example:

- `g = Create.Grep("([0-9]+)[/]([0-9]+)[/]([0-9]+)")`
- `g.match("12/24/2022")` Returns `true`
- `g.count` Returns `4`

6.19.1.2.2 .valid

`v.valid` – Returns a boolean indicator if the pattern string's expression is valid

For example:

- `g = Create.Grep()`
- `g.set("([0-9]+)[/]([0-9]+)[/]([0-9]+)")`
- `g.valid` Returns `true`

- `g.set("(0-9]+)[/]([0-9]+)[/]([0-9]+)")`
- `g.valid` Returns `false`

6.20 Date

6.20.1 Intrinsic

6.20.1.1 Functions

6.20.1.1.1 .set

`v.set(t)` – Returns formatted date and time string

The `t` argument's date-time string is parsed into numerical `day`, `month`, `year`, `hour`, `minutes`, and `seconds` fields.

The following date-time string formats are parsed. If any field is not provided then it will default to the base date-time `January 1, 1900 00:00:00`:

`MM/DD`	`MM` is month from `1` to `12` and `DD` is day from `1` to `31`
`MM/DD/YY`	`YY` is year from `0` to `99`
`MM/DD/YYYY`	`YYYY` is year from `0` to `9999`
`DD-MMM`	`MMM` is `JAN` to `DEC` or `JANUARY` to `DECEMBER`
`DD-MMM-YYYY`	
`MMM DD, YYYY`	
`HH:MM`	`HH` is hour from `0` to `12` and `MM` is minutes from `0` to `59`
`HH:MM:SS`	`SS` is seconds from `0` to `59`
`HH:MM:SS XX`	`XX` is `AM` or `PM`

⇒ NOTE: The date-time can contain just the date, time, or date-time combinations.

For example:

- `d = Create.Date()` Set initial time date-time: `01/01/1900 00:00:00`
- `d.set("12/24/2022 8:00")` Returns `"12/24/2022 08:00:00"`

⇒ NOTE: Each of the `day`, `month`, `year`, `hour`, `minutes`, and `seconds` fields will be set accordingly.

6.20.1.1.2 .current

`v.current()` – Returns current local time

For example:

- `d = Create.Date()` Set initial time date-time: `01/01/1900 00:00:00`
- `d.current()` Returns current local time

6.20.1.1.3 .clear

`v.clear()` – Returns initial date-time value

For example:

- `d = Create.Date("12/24/2022 8:00")`
- `d.clear()` Returns `"01/01/1900 00:00:00"`

6.20.1.2 Fields

6.20.1.2.1 .time

`v.time` – Returns date-time string

For example:

- `d = Create.Date("12/24/2022 8:00")`
- `d.time` Returns `"12/24/2022 08:00:00"`

6.20.1.2.2 .day

`v.day` – Sets or gets the date's `day` field

For example:

- `d = Create.Date("12/24/2022 8:00")`
- `d.day` Returns `24`
- `d.day = 25` Set day to the `25`th
- `d.time` Returns `"12/25/2022 08:00:00"`

6.20.1.2.3 .month

`v.month` – Sets or gets the date's `month` field

For example:

- `d = Create.Date("12/24/2022 8:00")`
- `d.month` Returns `12`
- `d.month = 11` Set month to November
- `d.time` Returns `"11/25/2022 08:00:00"`

6.20.1.2.4 .year

`v.year` – Sets or gets the date's `year` field

For example:

- `d = Create.Date("12/24/2022 8:00")`
- `d.year` Returns `2022`
- `d.year = 2021` Set the year to `2021`
- `d.time` Returns `"12/24/2021 08:00:00"`

6.20.1.2.5 .hour

`v.hour` – Sets or gets the time's `hour` field

For example:

- `d = Create.Date("12/24/2022 8:00")`
- `d.hour` Returns `8`
- `d.hour = 9` Set the hour to 9th hour
- `d.time` Returns `"12/24/2022 09:00:00"`

6.20.1.2.6 .minutes

`v.minutes` – Sets or gets the time's `minutes` field

For example:

- `d = Create.Date("12/24/2022 8:00")`
- `d.minutes` Returns `0`
- `d.minutes = 30` Set the minutes to ½ hour
- `d.time` Returns `"12/24/2022 08:30:00"`

6.20.1.2.7 .seconds

`v.seconds` – Sets or gets the time's `seconds` field

For example:

- `d = Create.Date("12/24/2022 8:00")`
- `d.seconds` Returns `0`
- `d.seconds = 15` Set the seconds to `15`

- `d.time` Returns `"12/24/2022 08:00:15"`

6.20.1.2.8 .leapyear

`v.leapyear` – Returns a boolean indicator if it is a leap year

For example:

- `d = Create.Date("02/29/2020")`
- `d.leapyear` Returns `true`

6.20.1.2.9 .dayofweek

`v.dayofweek` – Returns `day` of the week

The field will return one of the following values: `"sunday"`, `"monday"`, `"tuesday"`, `"wednesday"`, `"thursday"`, `"friday"`, or `"saturday"`

For example:

- `d = Create.Date("12/24/2021")`
- `d.dayofweek` Returns `"friday"`

6.20.1.2.10 .monthname

`v.monthname` – Returns `month`'s name

The field will return one of the following values: `"january"`, `"february"`, `"march"`, `"april"`, `"may"`, `"june"`, `"july"`, `"august"`, `"september"`, `"october"`, `"november"`, or `"december"`

For example:

- `d = Create.Date("12/24/2021")`
- `d.monthname` Returns `"december"`

6.21 Variable

6.21.1 Variable.locallist

`Variable.list()` – Returns a list of all local variable names

This will return a list of local variables with the currently executing script or function.

6.21.2 Variable.localdelete

`Variable.delete(`v_1`, `v_2`, ..., `v_n`)` – Deletes named local variables

The v_1, v_2, ..., v_n arguments are lists of local variable names to be removed within currently executing script or function.

6.21.3 Variable.globalsave

`Variable.globalsave(n, v)` – Saves variable for reuse
`Variable.globalsave(n, v, o)`

The `n` argument is the name of the variable to be saved for future restoration. The `v` argument is the variable to be saved. And, the `o` argument is a boolean for a transient for `true` or permanent storage `false` indicator.

If `o` argument is not specified, it defaults to `true`, meaning if the app is restarted then the variable is flushed from memory.

6.21.4 Variable.globalexists

`Variable.globalexists(n)` – Returns global variable exists indicator

The `n` argument is the name of the variable is `true` if it exists; otherwise, `false` indicator is returned.

6.21.5 Variable.globallist

`Variable.globalist()` – Returns list of all global variable names

This will return a list of global variables names.

6.21.6 Variable.globallookup

`Variable.globallookup(n)` – Returns global variable

The `n` argument is the name of the global variable to be restored.

6.21.7 Variable.globalflush

`Variable.globalflush(n)` – Flushes global variable

The `n` argument is the name of the global variable to be flushed from memory.

> ⇒ NOTE: If the global variable originally saved as transient mode, it's deleted; otherwise, it's flushed from memory and can be reloaded by calling the `Variable.globallookup` function since it's saved in the app.

6.21.8 Variable.globaldelete

`Variable.globaldelete(n)` – Deletes the global variable

The `n` argument is the name of the global variable to be deleted.

6.21.9 Intrinsic

6.21.9.1 Functions

6.21.9.1.1 .set

`v.set(i, j, v)` – Sets two-dimension indexed array's value

The `i` and `j` argument are indexes into two-dimension array to set value's `v` argument.

For example:

- `a = Create.array(0, 5, 5)` — Returns integer array of `5 × 5`
- `a.set(0, 0, 10)` — Results in `a[0, 0]` containing `10`
- `a.set(1, 1, 20)` — Results in `a[1, 1]` containing `20`

6.21.9.2 Fields

6.21.9.2.1 .size

`v.size` – Returns the number of elements

For example:

- `x = 1`
- `x.size` — Returns `1`

- `a = Create.array(0.0, 5, 5)`
- `a.size` — Returns `25`

6.21.9.2.2 .type

`v.type` – Returns data type

For example:

- `x = 1`
- `x.type` — Returns `Type.INTEGER`

- `a = Create.array(0.0, 5, 5)`
- `a.type` — Returns `Type.REAL`

6.21.9.2.3 .array

`v.array` – Returns the number of dimensions

For example:

- `x = 1`
- `x.array` Returns `0`

- `a = Create.array(0.0, 5, 5)`
- `a.array` Returns `2`

6.21.9.2.4 .dimensions

`v.dimensions` – Returns array's dimension sizes

For example:

- `x = 1`
- `x.dimensions` Returns `0`

- `a = Create.array(0.0, 5, 5)`
- `a. dimensions` Returns `5` and `5`

6.21.9.2.5 .info

`v.info` – Returns textual information about the variable

For example:

- `x = 1`
- `print x.info`

Results in:

```
variable:   x
type:       integer
value:      1
```

- `a = Create.array(0.0, 5, 5)`
- `print a. info`

Results in:

```
variable:   a
type:       real
matrix:     | 0, 0, 0, 0, 0 |
            | 0, 0, 0, 0, 0 |
            | 0, 0, 0, 0, 0 |
            | 0, 0, 0, 0, 0 |
            | 0, 0, 0, 0, 0 |
```

6.22 Distributions

The *Distribution* is a collection of statistical models consisting *Binomial*, *Chi-Square*, *Normal*, *Students-T* and many others.

6.22.1 Common Intrinsic

All the statistical methods have common intrinsic functions, such as `pdf`, `cdf`, `quantile`, etc.

The following sub-section's intrinsic function examples will use the following Binomial distribution using a coin's heads and tails statistics for best-of-three coin flips:

```
N = 3                                          Sets number of coin flips (i.e., trials)
p = 0.5                                        Sets probability to 50% for heads or tails
d = Create.Distribution(DIST.BINOMIAL, N, p)
```

6.22.1.1 Functions

6.22.1.1.1 .pdf

`v.pdf(x)` – Returns probability density function

⇒ LINK: https://en.wikipedia.org/wiki/Probability_density_function

For example:

- `d.pdf(1)` Returns `0.375`
- `d.pdf([0:N])` Returns `0.125`, `0.375`, `0.375`, and `0.125`

6.22.1.1.2 .cdf

`v.cdf(x)` – Returns cumulative density function

⇒ LINK: https://en.wikipedia.org/wiki/Cumulative_distribution_function

For example:

- `d.cdf(1)` Returns `0.5`
- `d.cdf(([0:N])` Returns `0.125`, `0.5`, `0.875`, and `1`

6.22.1.1.3 .cdfc

`v.cdfc(x)` – Returns cumulative density function complement

For example:

- `d.cdfc(1)` Returns `0.5`
- `d.cdfc(([0:N])` Returns `0.875`, `0.5`, `0.125`, and `0`

6.22.1.1.4 .quantile

`v.quantile(x)` – Returns the quantile

⇒ LINK: https://en.wikipedia.org/wiki/Quantile

For example:

- `d.quantile(1)` Returns `3`
- `d.quantile([0.25,0.50,0.75,1.00])` Returns `0`, `1`, `2`, and `3`

6.22.1.1.5 .quantilec

`v.quantilec(x)` – Returns the quantile from the complement of the probability

For example:

- `d.quantilec(1)` Returns `0`
- `d.quantilec([0.0,0.25,0.50,0.75])` Returns `3`, `2`, `1`, and `0`

6.22.1.1.6 .mean

`v.mean()` – Returns average value

⇒ LINK: https://en.wikipedia.org/wiki/Mean

For example:

- `d.mean()` Returns `1.5`

6.22.1.1.7 .variance

`v.variance()` – Returns variance of the standard deviation squared

⇒ LINK: https://en.wikipedia.org/wiki/Variance

For example:

- `d.variance()` Returns `0.75`

6.22.1.1.8 .stddev

`v.stddev()` – Returns measured amount of variation in a set of values

⇒ LINK: https://en.wikipedia.org/wiki/Standard_deviation

For example:

- `d.stddev()` Returns `0.8660254037844386`

6.22.1.1.9 .mode

`v.mode()` – Returns the value that is most likely to be sampled

⇒ LINK: https://en.wikipedia.org/wiki/Mode_(statistics)

For example:

- `d.mode()` Returns `2`

6.22.1.1.10 .skewness

`v.skewness()` – Returns a measured asymmetry of probability distribution of random value about its mean

⇒ LINK: https://en.wikipedia.org/wiki/Skewness

For example:

- `d.skewness()` Returns `0`

6.22.1.1.11 .median

`v.median()` – Returns the value between higher half from lower half of a data sample

⇒ LINK: https://en.wikipedia.org/wiki/Median

For example:

- `d.median()` Returns `1`

6.22.1.1.12 .kurtosis

`v.kurtosis()` – Returns the shape of a probability distribution

⇒ LINK: https://en.wikipedia.org/wiki/Kurtosis

For example:

- `d.kurtosis()` Returns `2.333333`

6.22.1.1.13 .kurtosis_excess

`v.kurtosis_excess()` – Returns a measure of the peaked-ness in a distribution

For example:

- `d.kurtosis_excess()` Returns `-0.6666666666666663`

6.22.2 Distributions Intrinsic

6.22.2.1 Bernoulli Functions

`Create.Distribution(DIST.BERNOULLI)` – Returns *Bernoulli* distribution model
`Create.Distribution(DIST.BERNOULLI, p)`

The *Bernoulli* distribution is a discrete distribution of the outcome of a single trial with a probability of success `p` argument resulting either `0` (i.e., failure) or `1` (i.e., success).

If the `p` argument is not specified, the success fraction value defaults to `0.5` (i.e., `50%` probability). The range of `p` is `0 ≤ p ≤ 1`.

⇒ LINK: https://en.wikipedia.org/wiki/Bernoulli_distribution

For example:

- `d = Create.Distribution(DIST.BERNOULLI)`
- `d.pdf([0, 1])` Returns `0.5` and `0.5`

6.22.2.1.1 .fraction

`v.fraction()` – Returns the probability of success fraction value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.BERNOULLI, 0.5)`
- `d.fraction()` Returns `0.5`

6.22.2.2 Beta Functions

`Create.Distribution(DIST.BETAD)` – Returns *Beta* distribution model
`Create.Distribution(DIST.BETAD, a)`
`Create.Distribution(DIST.BETAD, a, b)`

The *Beta* distribution is used as a prior distribution for binomial proportions in Bayesian analysis.

If the `a` argument is not specified, the alpha value defaults to 1. The range of `a` is `0 < a < ∞`.

If the `b` argument is not specified, the beta value defaults to 1. The range of `b` is `0 < a < ∞`.

If `a` and `b` equals `1`, it is space uniform distribution in the entire interval `0 ≤ x ≤ 1`. If `a` and `b` is less than `1`, the probability density function is U-shaped. If `a` is not equal to `b`, the shape is asymmetric.

⇒ LINK: https://en.wikipedia.org/wiki/Beta_distribution

Plot example:

```
d = Create.Distribution(DIST.BETAD, 2, 5)
x = [ 0 : 1 : 0.01 ]

Utils.PlotIt("Beta PDF",                    \
             "Plt033_Beta",                 \
             Plot.CHART.LINE,               \
             x, d.pdf(x),                   \
             0, 1, 0, 3, 11, 7)
```

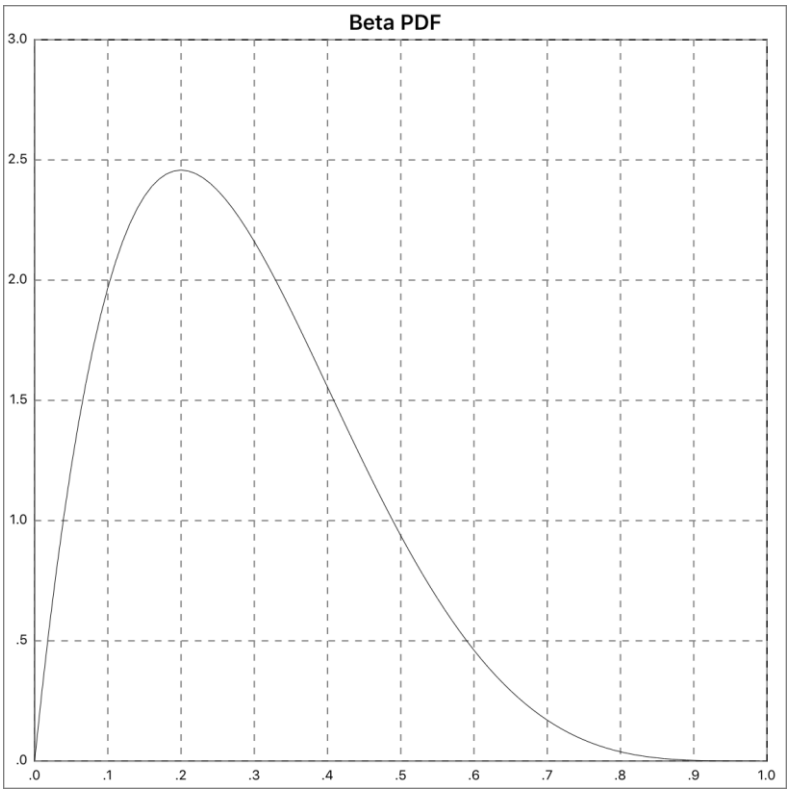

Figure 77 - Beta PDF Plot

6.22.2.2.1 .alpha

`v.alpha()` – Returns alpha's initialized value from the Beta distribution initialization

For example:

- `d = Create.Distribution(DIST.BETAD)`
- `d.alpha()` Returns `1`

6.22.2.2.2 .beta

`v.beta()` – Returns beta's initialized value from the Beta distribution initialization

For example:

- `d = Create.Distribution(DIST.BETAD)`

- `d.beta()` Returns `1`

6.22.2.3 Binomial Functions

`Create.Distribution(DIST.BINOMIAL)` – Returns *Binomial* distribution model
`Create.Distribution(DIST.BINOMIAL, t)`
`Create.Distribution(DIST.BINOMIAL, t, p)`

The *Binomial* distribution is used to derive the probability of observing the number of successes in `t` argument trials, with the `p` argument probability of success on a single trial.

If the `t` argument is not specified, the number of trials value defaults to `1`. The range of `t` is `0 ≤ t < ∞`.

If the `p` argument is not specified, the fraction value defaults to `0.5`. The range of `p` is `0 ≤ p ≤ 1`.

⇒ NOTE: Refer to the Demo script `D04_Binomial` function for detailed illustration of the *Binomial* distribution. Refer to the ***Demo Tour*** section for installing the script.

⇒ LINK: https://en.wikipedia.org/wiki/Binomial_distribution

Plot example:

```
d = Create.Distribution(DIST.BINOMIAL,10,0.5)
x = [ 0 : 10 : 0.1 ]

Utils.PlotIt("Binomial PDF",            \
             "Plt034_Binomial",         \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             0, 10, 0, 0.3, 11, 7)
```

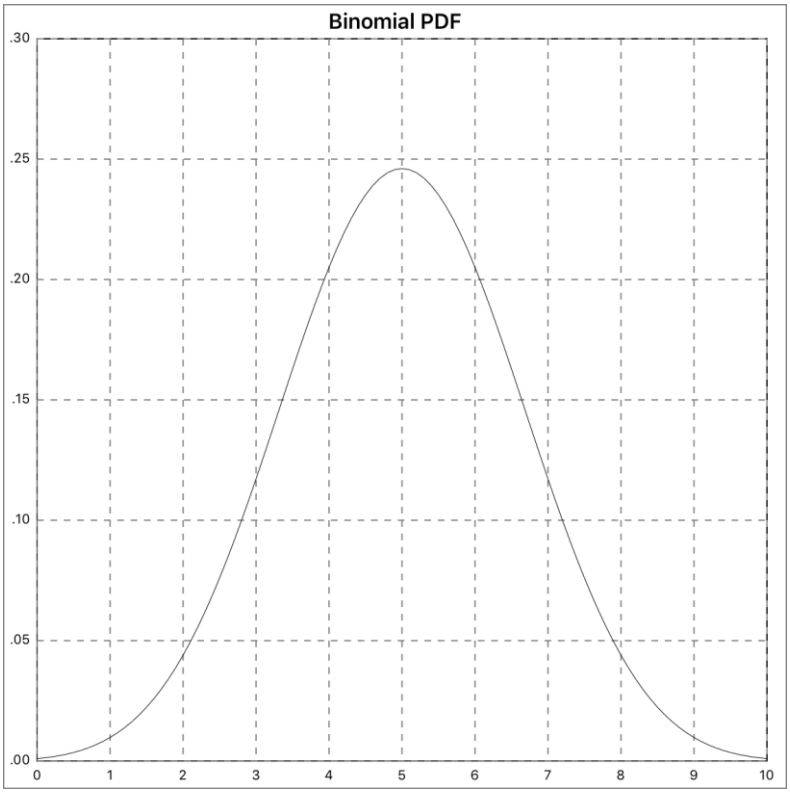

Figure 78 - Binomial PDF Plot

6.22.2.3.1 .trials

`v.trials()` – Returns the number of trials value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.BINOMIAL)`
- `d.trials()` Returns `1`

6.22.2.3.2 .fraction

`v.fraction()` – Returns the probability of success fraction value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.BINOMIAL)`
- `d.fraction()` Returns `0.5`

6.22.2.3.3 .find_lower_bound_on_p

`v.find_lower_bound_on_p(t, s, p)` – Returns lower-bound of successes
`v.find_lower_bound_on_p(t, s, p, m)`

The `t` argument is the number of trials with `s` argument's number of successes on the probability of `p` argument.

If the `m` argument is not specified, method's default interval is `Stat.CLOPPER_PEARSON_EXACT_INTERVAL`. The other interval option is `Stat.JEFFREYS_PRIOR_INTERVAL`.

For example:

- `d = Create.Distribution(DIST.BINOMIAL)`
- `d.find_lower_bound_on_p(d.trials(), \`
  `d.trials()÷2, 0.5)` Returns `0.163193985408`

6.22.2.3.4 .find_upper_bound_on_p

`v.find_upper_bound_on_p(t, s, p)` – Returns upper-bound of successes
`v.find_upper_bound_on_p(t, s, p, m)`

The `t` argument is the number of trials with `s` argument's number of successes on the probability of `p` argument.

If the `m` argument is not specified, method's default interval is `Stat.CLOPPER_PEARSON_EXACT_INTERVAL`. The other interval option is `Stat.JEFFREYS_PRIOR_INTERVAL`.

For example:

- `d = Create.Distribution(DIST.BINOMIAL)`
- `d.find_upper_bound_on_p(d.trials(), \`
  `d.trials()÷2, 0.5)` Returns `0.836806014591`

6.22.2.4 Cauchy Functions

`Create.Distribution(DIST.CAUCHY)` – Returns *Cauchy* distribution model
`Create.Distribution(DIST.CAUCHY, l)`
`Create.Distribution(DIST.CAUCHY, l, s)`

The *Cauchy* distribution can used as a solution to the differential equation describing forced resonance, while in spectroscopy, it is the description of the line shape of spectral lines.

If the `l` argument is not specified, the location value defaults to `0`. The range of `l` is `0 ≤ l < ∞`.

If the `s` argument is not specified, the scale value defaults to `1`. The range of `s` is `0 < s < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Cauchy_distribution

Plot example:

```
d = Create.Distribution(DIST.CAUCHY, 0, 1)
x = [ -10 : 10 : 0.2 ]

Utils.PlotIt("Cauchy PDF",              \
```

```
"Plt035_Cauchy",            \
Plot.CHART.LINE,            \
x, d.pdf(x),                \
-10, 10, 0, 0.4, 5, 5)
```

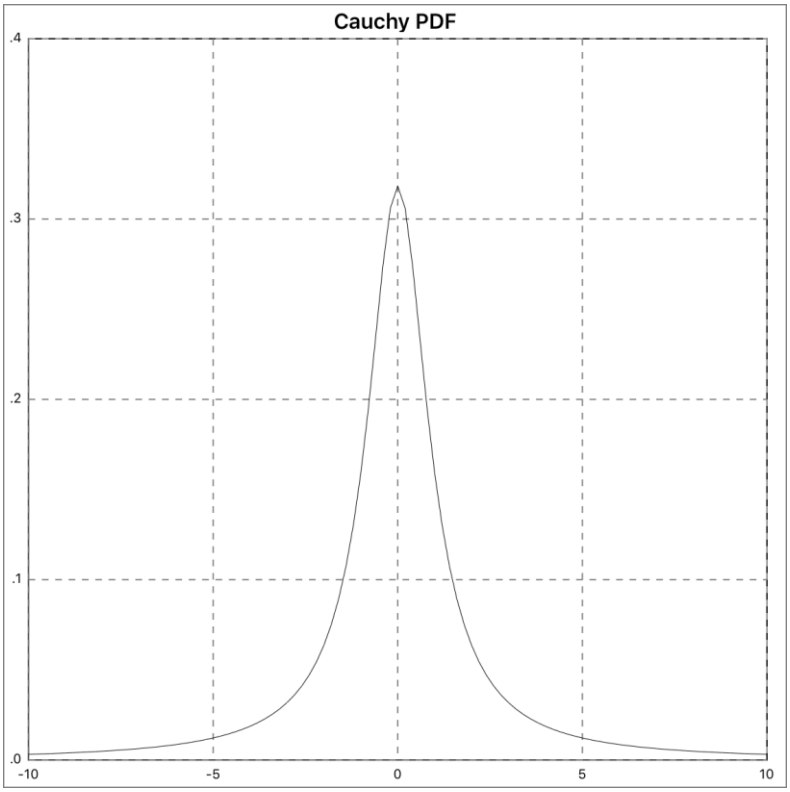

Figure 79 - Cauchy PDF Plot

6.22.2.4.1 .location

`v.location()` – Returns the location of the peak of the distribution value from the distribution initialization

- `d = Create.Distribution(DIST.CAUCHY)`
- `d.location()` Returns `0`

6.22.2.4.2 .scale

`v.scale()` – Returns the scale of half the interquartile range value from the distribution initialization

- `d = Create.Distribution(DIST.CAUCHY)`
- `d.scale()` Returns `1`

6.22.2.5 Chi-Squared Functions

`Create.Distribution(DIST.CHI_SQUARED, df)` – Returns *Chi-Squared* distribution model

The *Chi-Squared* distribution is a popular statistical model.

The `df` argument is degrees of freedom. The range of `df` is `0 < df < ∞`.

⇒ NOTE: Refer to the Demo script `D01_ChiSqu` function for detailed illustration of the *Chi-Squared* distribution. Refer to the ***Demo Tour*** section for installing the script.

⇒ LINK: https://en.wikipedia.org/wiki/Chi-square_distribution

Plot example:

```
d = Create.Distribution(DIST.CHI_SQUARED, 5)
x = [ 0 : 20 : 0.2 ]

Utils.PlotIt("Chi-Square PDF",          \
             "Plt036_ChiSquared",       \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             0, 20, 0, 0.2, 5, 5)
```

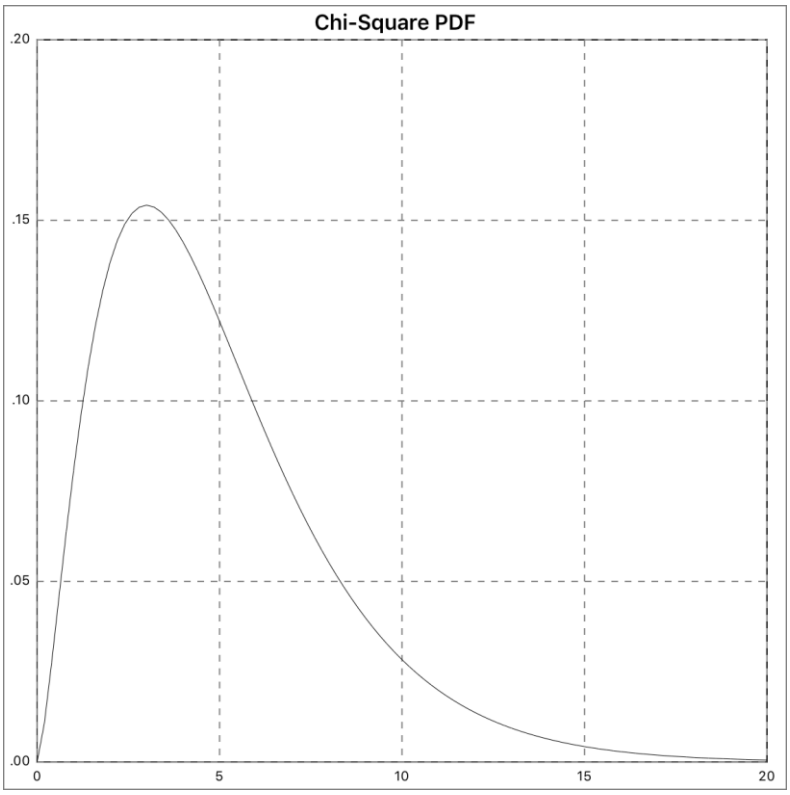

Figure 80 - Chi-Squared PDF Plot

6.22.2.5.1 .degrees_of_freedom

`v.degrees_of_freedom()` – Returns the degrees of freedom value from the distribution initialization

For example:

- `N = 100`
- `d = Create.Distribution(DIST.CHI_SQUARED, N - 1)`

- `d.degrees_of_freedom()` Returns `99`

6.22.2.5.2 .find_degrees_of_freedom

`v.find_degrees_of_freedom(d, a, b, v)` – Returns the degrees of freedom
`v.find_degrees_of_freedom(d, a, b, s, h)`

The `d` argument is the difference from the assumed nominal variance with maximum acceptable risk of rejecting the alpha's `a` argument along with the risk of falsely failing the beta's `b` argument using the variance's `v` argument to be tested.

If the `h` argument is not specified, the hint value defaults to `100`.

For example:

- `N        = 100` Sets sample size
- `stddev   = 0.6278908E-02` Sets sample standard deviation
- `alpha    = 0.05` Sets significance level
- `variance = 0.1` Sets variance

- `d = Create.Distribution(DIST.CHI_SQUARED, N - 1)`

- `d.find_degrees_of_freedom(\` Returns `50.351485`
  `variance - stddev,  \`
  `alpha, alpha, variance)`

6.22.2.6 Exponential Functions

`Create.Distribution(DIST.EXPONENTIAL)` – Returns *Exponential* distribution model
`Create.Distribution(DIST.EXPONENTIAL, l)`

The *Exponential* distribution is a continuous probability distribution.

If the `l` argument is not specified, the lambda value defaults to `1`. The range of `l` is `0 < l < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Exponential_distribution

Plot example:

```
d = Create.Distribution(DIST.EXPONENTIAL, 1)
x = [ 0 : 6 : 0.06 ]

Utils.PlotIt("Exponential PDF",          \
             "Plt037_Exponential",       \
             Plot.CHART.LINE,            \
             x, d.pdf(x),                \
             0, 6, 0, 1, 7, 11)
```

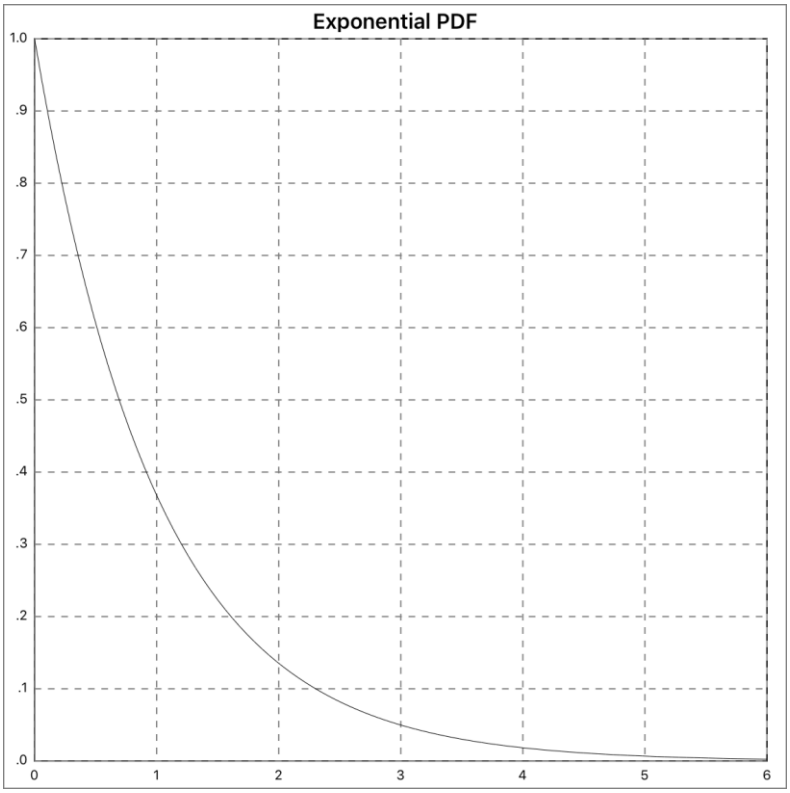

Figure 81 - Exponential PDF Plot

6.22.2.6.1 .lambda

`v.lambda()` – Returns lambda value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.EXPONENTIAL)`
- `d.lambda()` Returns `1`

6.22.2.7 Fisher's F Functions

`Create.Distribution(DIST.FISHER_F, df1, df2)` – Returns *Fisher's F* distribution model

The *Fisher's F* distribution is a continuous distribution for testing two samples having the same variance.

The `df1` and `df2` arguments are the numerator and denominator degrees of freedoms. The range of both degrees of freedom is `0 < df < ∞`.

⇒ NOTE: Refer to the Demo script `D03_Fisher` function for detailed illustration of the *Fisher's F* distribution. Refer to the ***Demo Tour*** section for installing the script.

⇒ LINK: https://en.wikipedia.org/wiki/F-distribution

Plot example:

```
d = Create.Distribution(DIST.FISHER_F,10,10)
```

```
x = [ 0 : 4 : 0.04 ]

Utils.PlotIt("Fisher's F PDF",          \
             "Plt038_Fisher_F",         \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             0, 4, 0, 1, 5, 11)
```

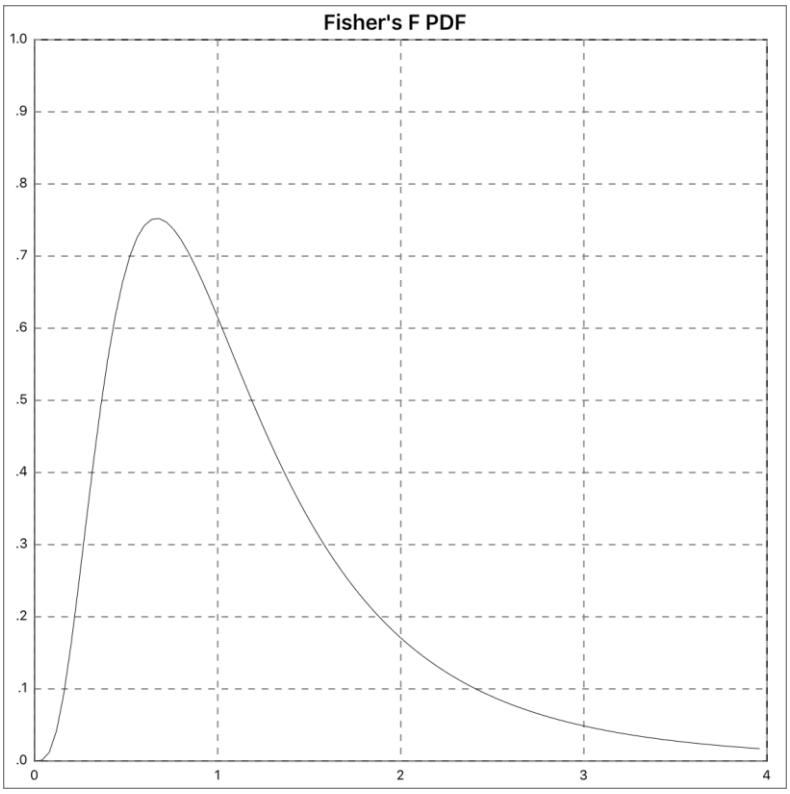

Figure 82 - Fisher's F PDF Plot

6.22.2.7.1 .degrees_of_freedom1

`v.degrees_of_freedom1()` – Returns numerator degrees of freedom first value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.FISHER_F, 4, 10)`
- `d.degrees_of_freedom1()` Returns `4`

6.22.2.7.2 .degrees_of_freedom2

`v.degrees_of_freedom2()` – Returns denominator degrees of freedom second value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.FISHER_F, 4, 10)`

- `d.degrees_of_freedom2()` Returns `10`

6.22.2.8 Gamma Functions

`Create.Distribution(DIST.GAMMA, sh)` – Returns *Gamma* distribution model
`Create.Distribution(DIST.GAMMA, sh, sc)`

The *Gamma* distribution is a continuous probability distribution.

The `sh` argument is the shape of the distribution. The range of `sh` is `0 < sh < ∞`.

If the `sc` argument is not specified, the scale value defaults to `1`. The range of `sc` is `0 < sc < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Gamma_distribution

Plot example:

```
d = Create.Distribution(DIST.GAMMA, 1, 1)
x = [ 0 : 5 : 0.05 ]

Utils.PlotIt("Gamma PDF",                \
             "Plt039_Gamma",             \
             Plot.CHART.LINE,            \
             x, d.pdf(x),                \
             0, 5, 0, 1, 6, 11)
```

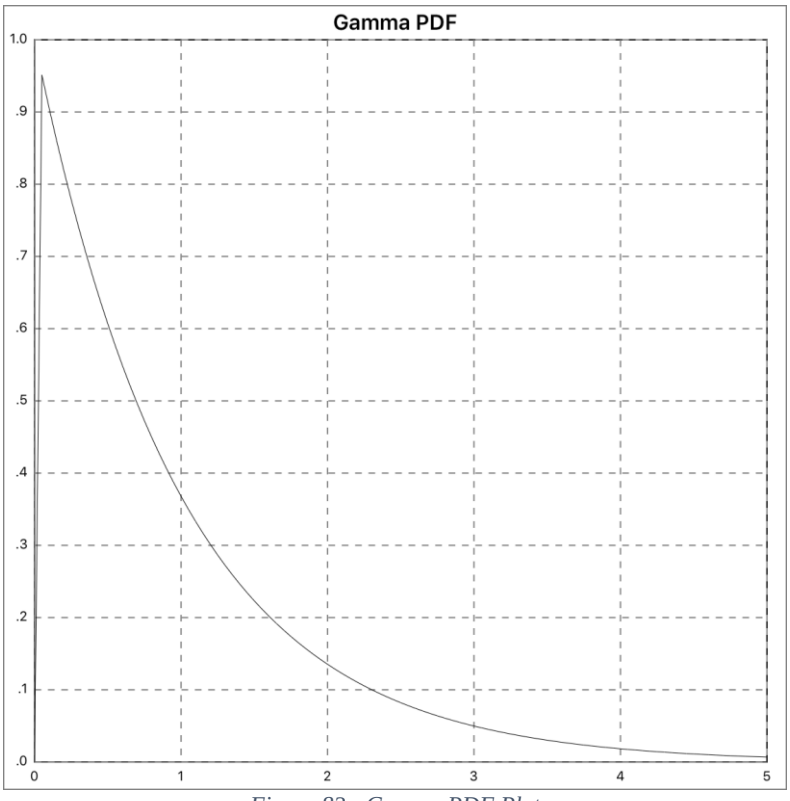

Figure 83 - Gamma PDF Plot

6.22.2.8.1 .shape

`v.shape()` – Returns shape value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.GAMMA, 1)`
- `d.shape()` Returns `1`

6.22.2.8.2 .scale

`v.scale()` – Returns scale value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.GAMMA, 1, 2)`
- `d.scale()` Returns `2`

6.22.2.9 Geometric Functions

`Create.Distribution(DIST.GEOMETRIC, p)` – Returns *Geometric* distribution model

The *Geometric* distribution is either of two discrete (i.e., success or failure, true or false) probability distributions in the `p` argument of the probability success fraction value.

The range of the `p` argument is $0 \leq p \leq 1$.

⇒ LINK: https://en.wikipedia.org/wiki/Geometric_distribution

Plot example:

```
d = Create.Distribution(DIST.GEOMETRIC, 0.5)
x = [ 0 : 8 : 0.08 ]

Utils.PlotIt("Geometric PDF",           \
             "Plt040_Geometirc",        \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             0, 8, 0, 0.5, 5, 6)
```

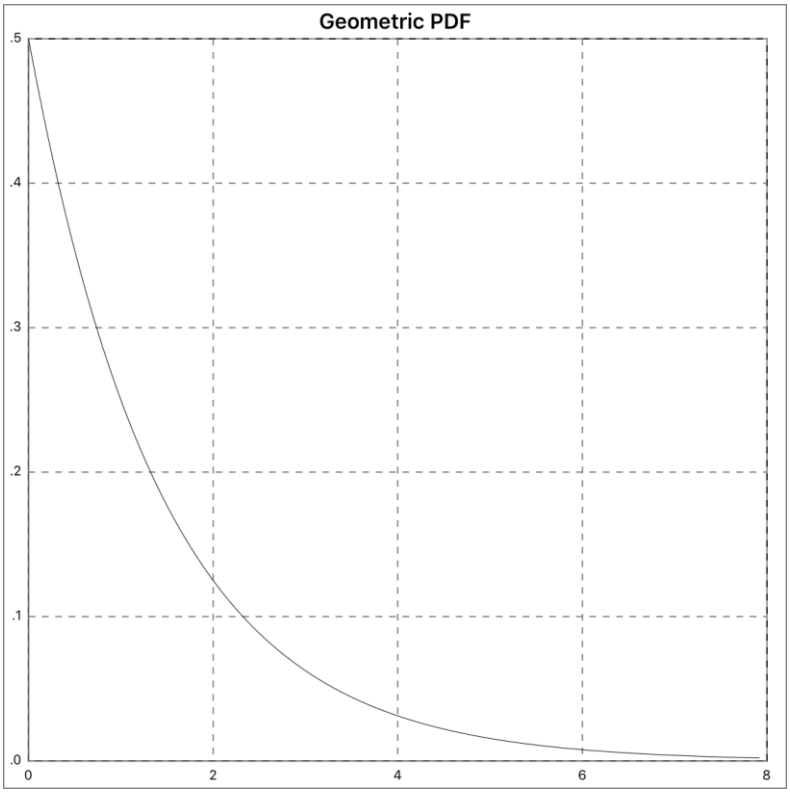

Figure 84 - Geometric PDF Plot

6.22.2.9.1 .fraction

`v.fraction()` – Returns the probability of success fraction value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.GEOMETRIC, 0.5)`
- `d.fraction()` Returns `0.5`

6.22.2.9.2 .successes

`v.successes()` – Returns number of successes

For example:

- `d = Create.Distribution(DIST.GEOMETRIC, 0.5)`
- `d.successes()` Returns `1`

6.22.2.9.3 .find_lower_bound_on_p

`v.find_lower_bound_on_p(t, p)` – Returns lower-bound of successes

The `t` argument is the number of trials on the probability of `p` argument.

For example:

- `d = Create.Distribution(DIST.GEOMETRIC, 0.5)`
- `d.find_lower_bound_on_p(d.successes(), 0.5)` Returns `0.5`

6.22.2.9.4 .find_upper_bound_on_p

`v.find_upper_bound_on_p(t, p)` – Returns upper-bound of successes

The `t` argument is the number of trials on the probability of `p` argument.

For example:

- `d = Create.Distribution(DIST.GEOMETRIC, 0.5)`
- `d.find_upper_bound_on_p(d.successes(), 0.5)` Returns `1`

6.22.2.10 Inverse Chi-Squared Functions

`Create.Distribution(DIST.INV_CHI_SQUARED, df)` – Returns *Inverse Chi-Squared* distribution
`Create.Distribution(DIST.INV_CHI_SQUARED, df, s)`

The *Inverse Chi-Squared* distribution is reciprocal of the variable distribution of the *Chi-Squared* distribution.

The `df` argument is degrees of freedom. The range of `df` is `0 < df < ∞`.

If the `s` argument is not specified, the scale value defaults to reciprocal of df. The range of `s` is `0 < s < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Inverse-chi-squared_distribution

Plot example:

```
d = Create.Distribution(DIST.INV_CHI_SQUARED, 3, 1)
x = [ 0 : 6 : 0.06 ]

Utils.PlotIt("Inverse Chi-Squared PDF", \
             "Plt041_InvChiSquared",    \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             0, 6, 0, 1, 7, 11)
```

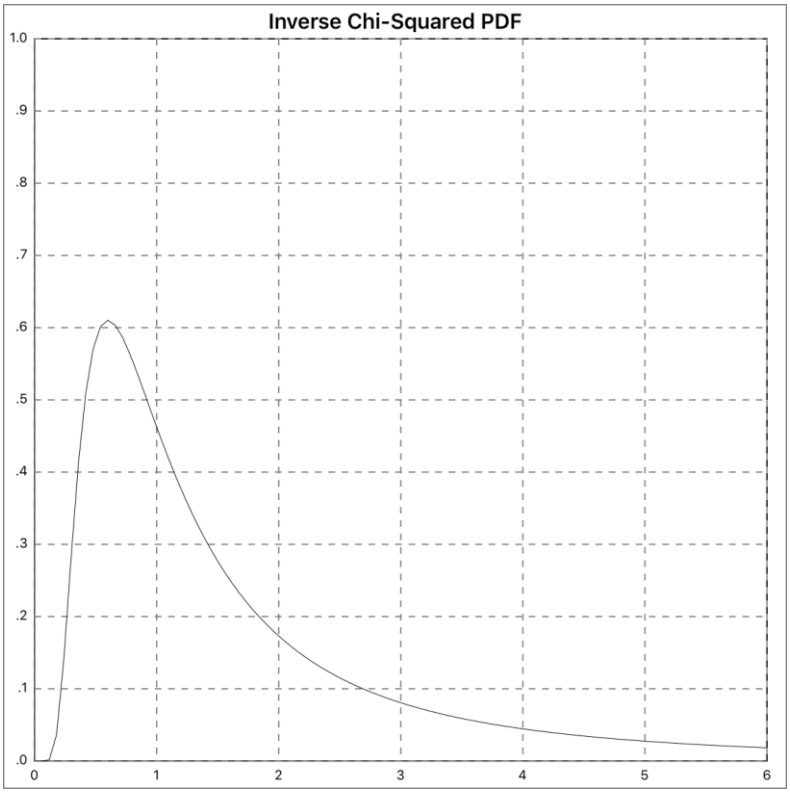

Figure 85 - Inverse Chi-Squared PDF Plot

6.22.2.10.1 .degrees_of_freedom

`v.degrees_of_freedom()` – Returns the degrees of freedom value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.INV_CHI_SQUARED, 3)`
- `d.degrees_of_freedom()` Returns `3`
- `d.scale()` Returns `0.33333333333333331`

6.22.2.10.2 .scale

`v.scale()` – Returns inverse Chi-Squared scale value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.INV_CHI_SQUARED, 3, 0.25)`
- `d.scale()` Returns `0.25`

6.22.2.11 Inverse Gamma Functions

`Create.Distribution(DIST.INV_GAMMA)` – Returns *Inverse Gamma* distribution model
`Create.Distribution(DIST.INV_GAMMA, sh)`
`Create.Distribution(DIST.INV_GAMMA, sh, sc)`

The *Inverse Gamma* distribution is reciprocal of the variable distribution of the *Gamma* distribution.

If the `sh` argument is not specified, the shape value defaults to `1`. The range of `sh` is `0 < sh < ∞`.

If the `sc` argument is not specified, the scale value defaults to `1`. The range of `sc` is `0 < sc < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Inverse-gamma_distribution

Plot example:

```
d = Create.Distribution(DIST.INV_GAMMA, 3, 2)
x = [ 0 : 2 : 0.02 ]

Utils.PlotIt("Inverse Gamma PDF",        \
             "Plt042_InvGamma",          \
             Plot.CHART.LINE,            \
             x, d.pdf(x),                \
             0, 2, 0, 3, 5, 7)
```

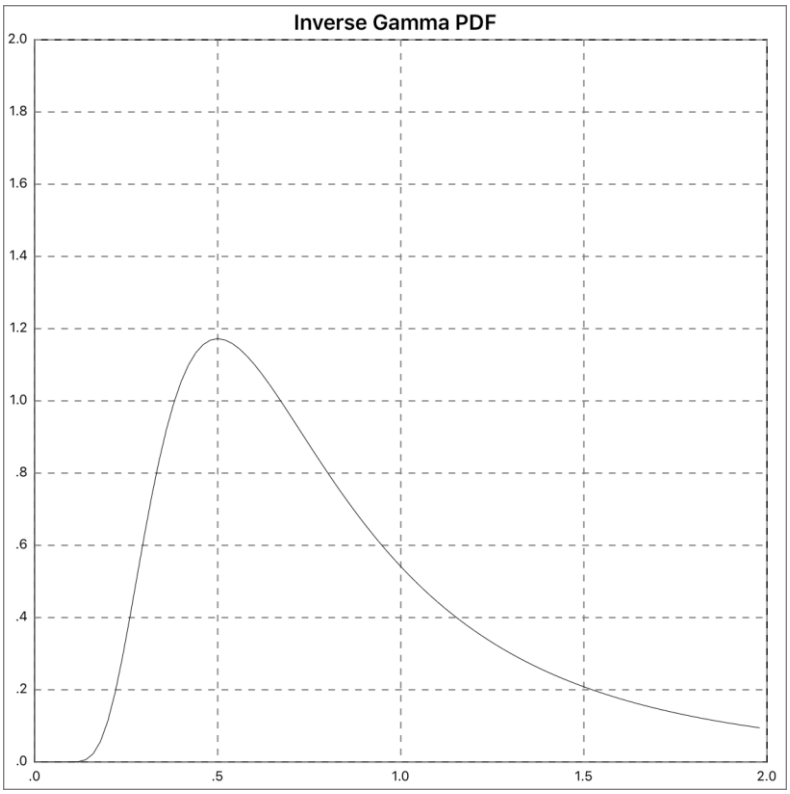

Figure 86 - Inverse Gamma PDF Plot

6.22.2.11.1 .shape

`v.shape()` – Returns shape value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.INV_GAMMA)`
- `d.shape()` Returns `1`

6.22.2.11.2 .scale

`v.scale()` – Returns scale value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.INV_GAMMA)`
- `d.scale()` Returns `1`

6.22.2.12 Inverse Gaussian Functions

`Create.Distribution(DIST.INV_GAUSSIAN)` – Returns *Inverse Gaussian* distribution model
`Create.Distribution(DIST.INV_GAUSSIAN, m)`
`Create.Distribution(DIST.INV_GAUSSIAN, m, s)`

The *Inverse Gaussian* distribution is continuous probability distribution.

If the `m` argument is not specified, the mean value defaults to `1`. The range of `m` is `0 ≤ m < ∞`.

If the `s` argument is not specified, the scale value defaults to `1`. The range of `s` is `0 < s < ∞`.

⇒ LINK : https://en.wikipedia.org/wiki/Inverse_Gaussian_distribution

Plot example:

```
D = Create.Distribution(DIST.INV_GAUSSIAN, 1, 1)
x = [ 0 : 4 : 0.04 ]

Utils.PlotIt("Inverse Gaussian PDF",    \
             "Plt043_InvGaussian",      \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             0, 4, 0, 2, 5, 11)
```

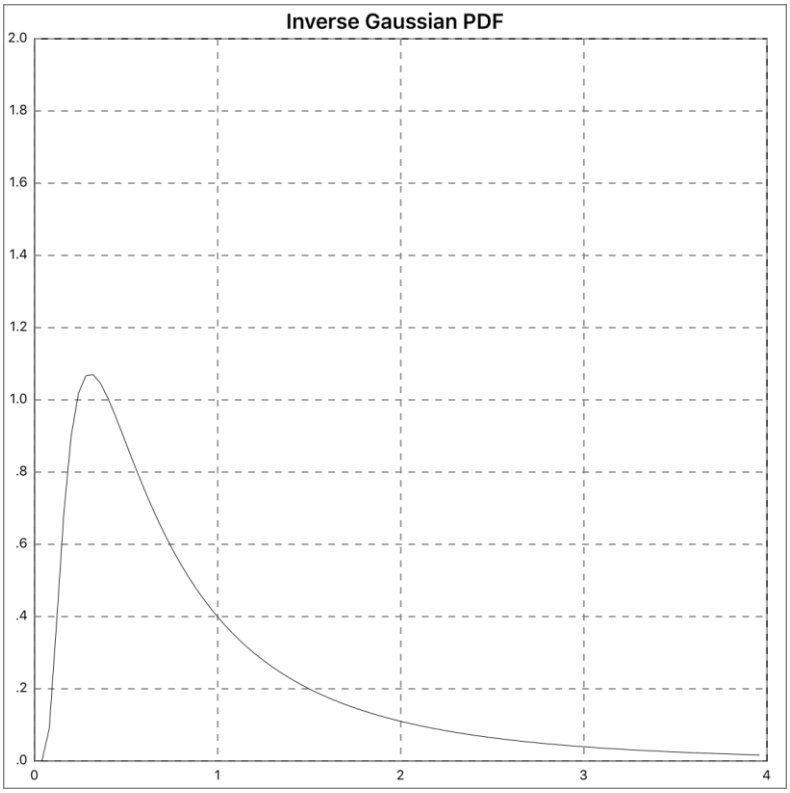

Figure 87 - Inverse Gaussian PDF Plot

6.22.2.12.1 .mean

`v.mean()` – Returns mean value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.INV_GAUSSIAN)`
- `d.mean()` Returns `1`

6.22.2.12.2 .scale

`v.scale()` – Returns scale value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.INV_GAUSSIAN)`
- `d.scale()` Returns `1`

6.22.2.13 Laplace Functions

`Create.Distribution(DIST.LAPLACE)` – Returns *Laplace* distribution model
`Create.Distribution(DIST.LAPLACE, l)`
`Create.Distribution(DIST.LAPLACE, l, s)`

The *Laplace* distribution is the distribution of differences between two independent variates with identical exponential distributions.

If the `l` argument is not specified, the location value defaults to `0`. The range of `l` is `0 < l < ∞`.

If the `s` argument is not specified, the scale value defaults to `1`. The range of `s` is `0 < s < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Laplace_distribution

Plot example:

```
d = Create.Distribution(DIST.LAPLACE, 0, 1)
x = [ -5 : 5 : 0.1 ]

Utils.PlotIt("Laplace PDF",                \
             "Plt044_Laplace",             \
             Plot.CHART.LINE,              \
             x, d.pdf(x),                  \
             -5, 5, 0, 0.5, 11, 6)
```

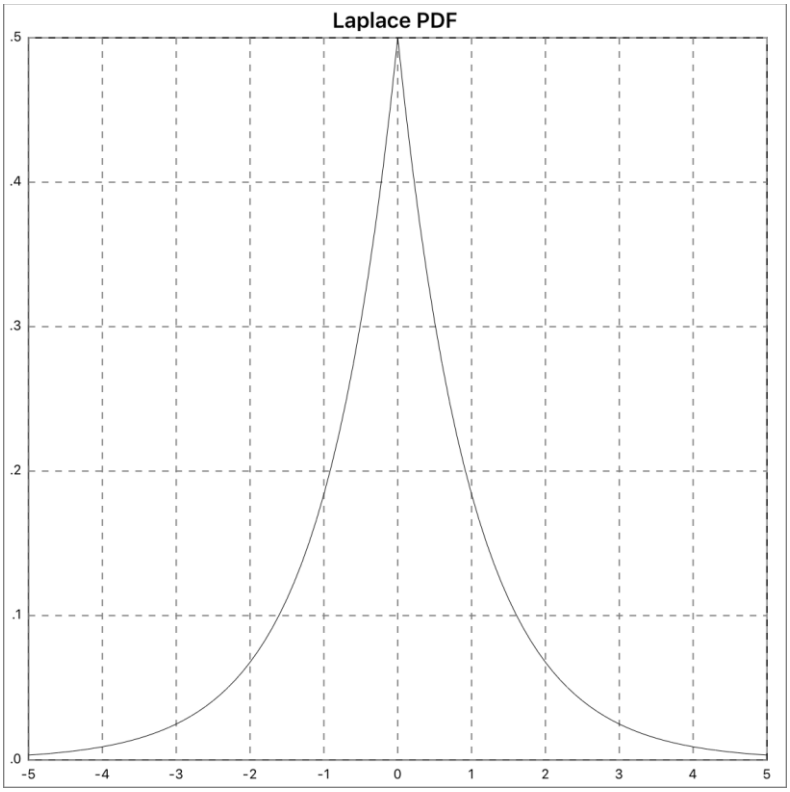

Figure 88 - Laplace PDF Plot

6.22.2.13.1 .location

`v.location()` – Returns location's mean of the random variate value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.LAPLACE)`

- `d.location()` Returns `0`

6.22.2.13.2 .scale

`v.scale()` – Returns scale's proportional to the standard deviation of the random variate value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.LAPLACE)`
- `d.scale()` Returns `1`

6.22.2.14 Logistic Functions

`Create.Distribution(DIST.LOGISTIC)` – Returns *Logistic* distribution model
`Create.Distribution(DIST.LOGISTIC, l)`
`Create.Distribution(DIST.LOGISTIC, l, s)`

The *Logistic* distribution is a continuous probability distribution.

If the `l` argument is not specified, the location value defaults to `0`. The range of `l` is `0 < l < ∞`.

If the `s` argument is not specified, the scale value defaults to `1`. The range of `s` is `0 < s < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Logistic_distribution

Plot example:

```
d = Create.Distribution(DIST.LOGISTIC, 0, 1)
x = [ -7 : 7 : 0.14 ]

Utils.PlotIt("Logistic PDF",            \
             "Plt045_logistic",         \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             -8, 8, 0, 0.3, 9, 7)
```

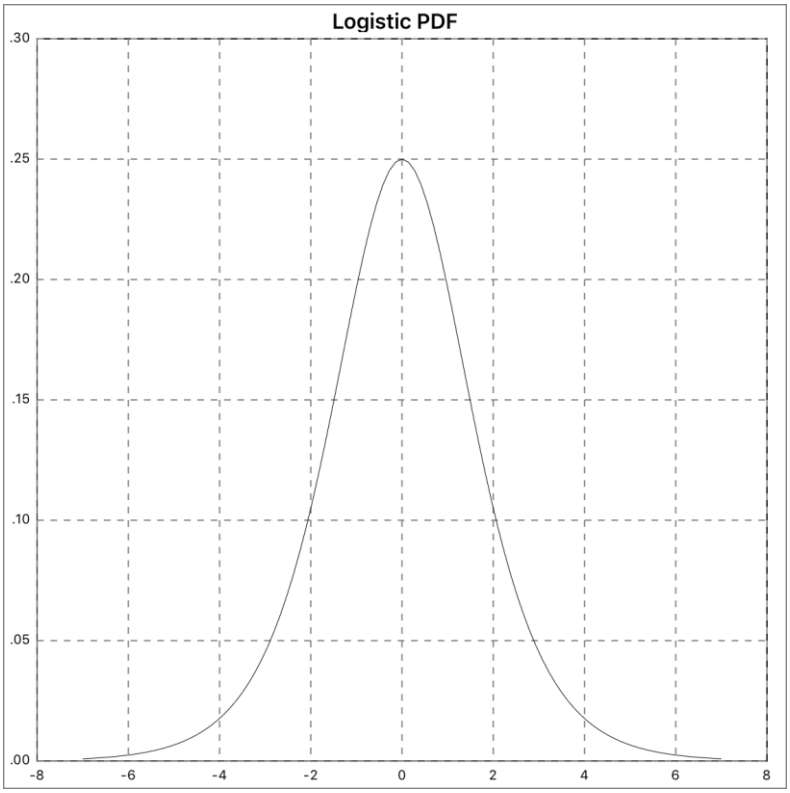

Figure 89 - Logistic PDF Plot

6.22.2.14.1 .location

`v.location()` – Returns the location value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.LOGISTIC)`
- `d.location()` Returns `0`

6.22.2.14.2 .scale

`v.scale()` – Returns the scale value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.LOGISTIC)`
- `d.scale()` Returns `1`

6.22.2.15 Log Normal Functions

`Create.Distribution(DIST.LOG_NORMAL)` – Returns *Log Normal* distribution model
`Create.Distribution(DIST.LOG_NORMAL, l)`
`Create.Distribution(DIST.LOG_NORMAL, l, s)`

If the `l` argument is not specified, the location value defaults to `0`. The range of `l` is `0 < l < ∞`.

If the `s` argument is not specified, the scale value defaults to `1`. The range of `s` is `0 < s < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Log-normal_distribution

Plot example:

```
d = Create.Distribution(DIST.LOG_NORMAL,0,1)
x = [ 0 : 7 : 0.07 ]

Utils.PlotIt("Log Normal PDF",          \
             "Plt046_LogNormal",        \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             0, 8, 0, 1, 9, 11)
```

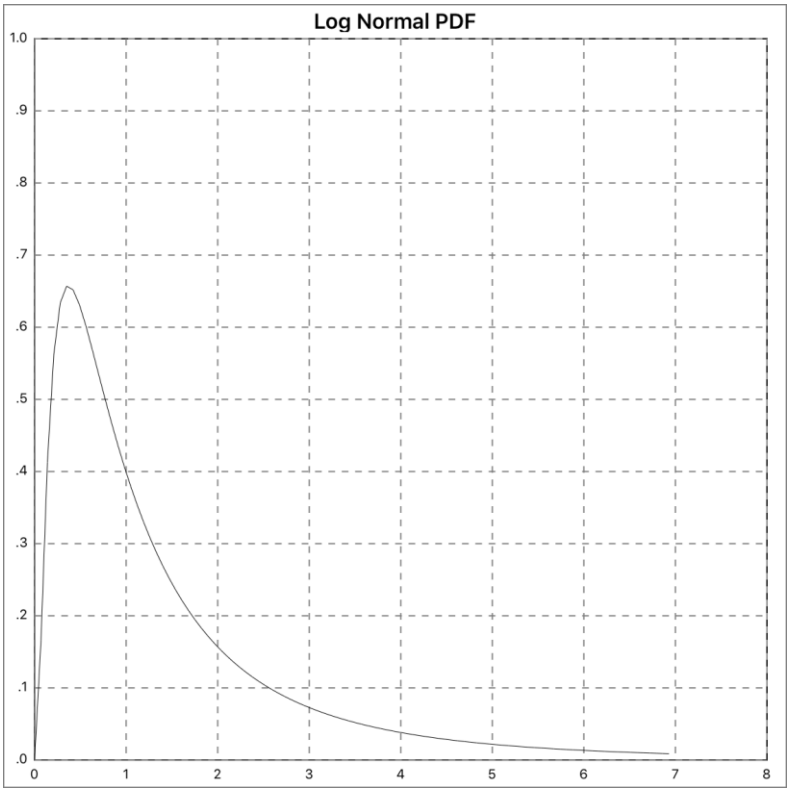

Figure 90 - Log Normal PDF Plot

6.22.2.15.1 .location

`v.location()` – Returns location's mean of the logarithm of the random variable value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.LOG_NORMAL)`
- `d.location()` Returns `0`

6.22.2.15.2 .scale

`v.scale()` – Returns scale's standard deviation of the logarithm of the random variable value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.LOG_NORMAL)`
- `d.scale()` Returns `1`

- `d = Create.Distribution(DIST.LOG_NORMAL)`
- `d.scale()` Returns `1`

6.22.2.16 Negative Binomial Functions

`Create.Distribution(DIST.NEG_BINOMIAL, s, p)` – Returns *Negative Binomial* distribution model

The *Negative Binomial* distribution is a discrete probability in the number of successes in the `s` argument with the `p` argument of the probability success fraction value.

The range of `s` argument is $0 < s < \infty$.

The range of `p` argument is $0 \leq p \leq 1$.

⇒ LINK : https://en.wikipedia.org/wiki/Negative_binomial_distribution

Plot example:

```
d = Create.Distribution(DIST.NEG_BINOMIAL, 20, 0.5)
x = [ 0 : 50 : 0.5 ]

Utils.PlotIt("Negative Binomial PDF",    \
             "Plt047_NegBinomial",       \
             Plot.CHART.LINE,            \
             x, d.pdf(x),                \
             0, 50, 0, 0.1, 6, 11)
```

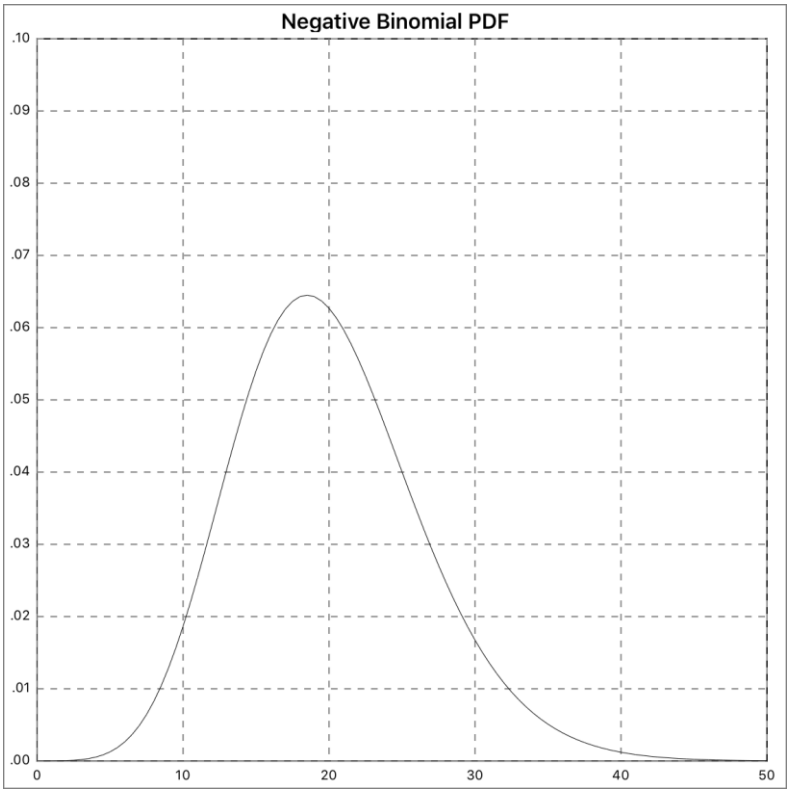

Figure 91 - Negative Binomial PDF Plot

6.22.2.16.1 .successes

`v.successes()` – Returns the number of successes value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.NEG_BINOMIAL, 10, 0.5)`
- `d.successes()` Returns `10`

6.22.2.16.2 .fraction

`v.fraction()` – Returns the probability of success fraction value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.NEG_BINOMIAL, 10, 0.5)`
- `d.fraction()` Returns `0.5`

6.22.2.16.3 .find_lower_bound_on_p

`v.find_lower_bound_on_p(t, s, p)` – Returns lower-bound of successes

The `t` argument is the number of trials with `s` argument's number of successes on the probability of `p` argument.

For example:

- `d = Create.Distribution(DIST.NEG_BINOMIAL, 10, 0.5)`
- `d.find_lower_bound_on_p(d.successes(), 5, 0.5)` Returns `0.4516941562236631`

6.22.2.16.4 .find_upper_bound_on_p

`v.find_upper_bound_on_p(t, s, p)` – Returns upper-bound of successes

The `t` argument is the number of trials with `s` argument's number of successes on the probability of `p` argument.

For example:

- `d = Create.Distribution(DIST.NEG_BINOMIAL, 10, 0.5)`
- `d.find_upper_bound_on_p(d.successes(), 5, 0.5)` Returns `0.4516941562236631`

6.22.2.17 Non-Central Chi-Squared Functions

`Create.Distribution(DIST.NON_CEN_CHI_SQUARED, df, l)` – Returns *Non-Central Chi-Squared* distribution model

The *Non-Central Chi-Squared* distribution is a generalization of the *Chi-Squared* distribution with the degrees of freedom in the `df` argument with the `l` argument's non-central lambda.

The range of `df` argument is `0 < df < ∞`.

The range of `l` argument is `0 < l < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Noncentral_chi-squared_distribution

Plot example:

```
d = Create.Distribution(DIST.NON_CEN_CHI_SQUARED, 20, 0)
x = [ 0 : 50 : 0.5 ]

Utils.PlotIt("Non-Central Chi-Squard PDF", \
             "Plt048_NonCenChiSquared",    \
             Plot.CHART.LINE,              \
             x, d.pdf(x),                  \
             0, 50, 0, 0.08, 6, 5)
```

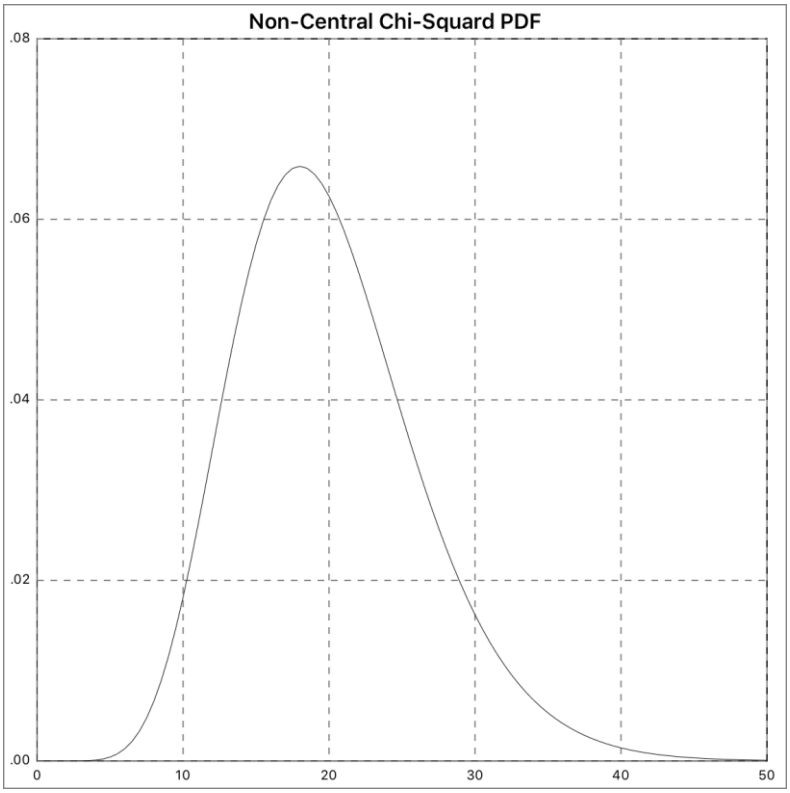

Figure 92 - Non-Central Chi-Squared PDF Plot

6.22.2.17.1 .degrees_of_freedom

`v.degrees_of_freedom()` – Returns the degrees of freedom value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.NON_CEN_CHI_SQUARED, 20, 0)`
- `d.degrees_of_freedom()` Returns `20`

6.22.2.17.2 .non_centrality

`v. non_centrality()` – Returns non-central lambda value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.NON_CEN_CHI_SQUARED, 20, 0)`
- `d. non_centrality()` Returns `0`

6.22.2.18 Normal Functions

`Create.Distribution(DIST.NORMAL)` – Returns *Normal* distribution model
`Create.Distribution(DIST.NORMAL, l)`
`Create.Distribution(DIST.NORMAL, l, s)`

The *Normal* distribution is the most popular statistical distribution with mean in the `l` argument with the `s` argument's scale.

If the `l` argument is not specified, the location's mean value defaults to `0`. The range of `l` is `0 ≤ l < ∞`.

If the `s` argument is not specified, the scale's standard deviation value defaults to `1`. The range of `s` is `0 < t < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Normal_distribution

Plot example:

```
d = Create.Distribution(DIST.NORMAL, 0, 0.5)
x = [ -2 : 2 : 0.04 ]

Utils.PlotIt("Normal PDF",                \
             "Plt049_Normal",             \
             Plot.CHART.LINE,             \
             x, d.pdf(x),                 \
             -2, 2, 0, 1, 5, 11)
```

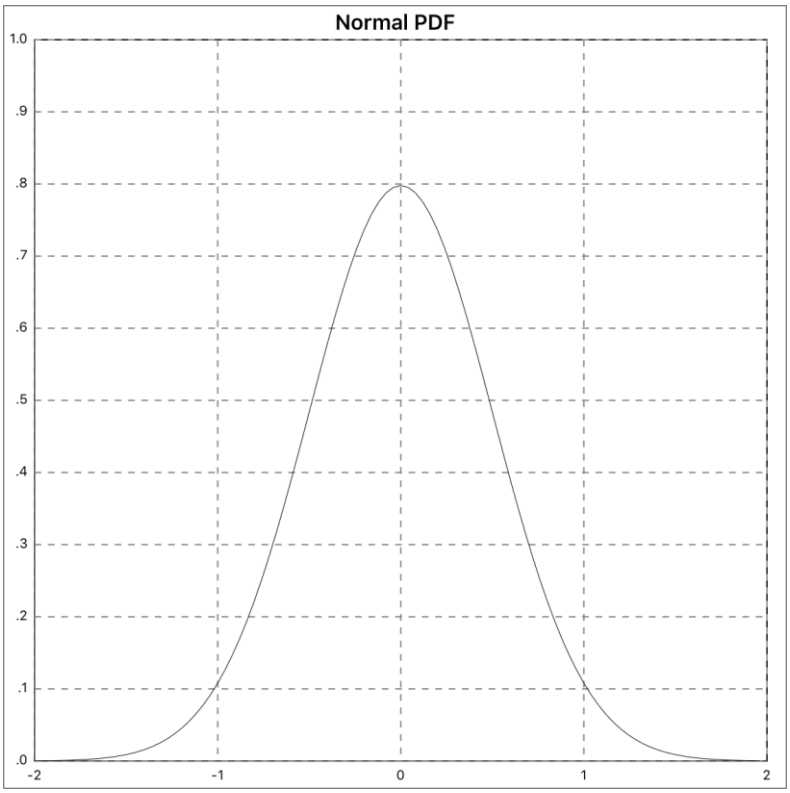

Figure 93 - Normal PDF Plot

6.22.2.18.1 .location

`v.location()` – Returns location's mean of the normal of the random variable value from the distribution initialization

For example:

- d = `Create.Distribution(DIST.NORMAL)`

- `d.location()` Returns `0`

6.22.2.18.2 .scale

`v.scale()` – Returns scale's standard deviation of the logarithm of the random variable value from the distribution initialization

For example:

- d = `Create.Distribution(DIST.NORMAL)`
- `d.scale()` Returns `1`

6.22.2.19 Pareto Functions

`Create.Distribution(DIST.PARETO)` – Returns *Pareto* distribution model
`Create.Distribution(DIST.PARETO, sc)`
`Create.Distribution(DIST.PARETO, sc, sh)`

The *Pareto* distribution is a continuous distribution.

If the `sc` argument is not specified, the scale value defaults to `1`. The range of `sc` is `0 < sc < ∞`.

If the `sh` argument is not specified, the shape value defaults to `1`. The range of `sh` is `0 < sh < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Pareto_distribution

Plot example:

```
d = Create.Distribution(DIST.PARETO, 1, 1)
x = [ 1 : 15 : 0.14 ]

Utils.PlotIt("Pareto PDF",              \
             "Plt050_Pareto",           \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             1, 16, 0, 1, 6, 11)
```

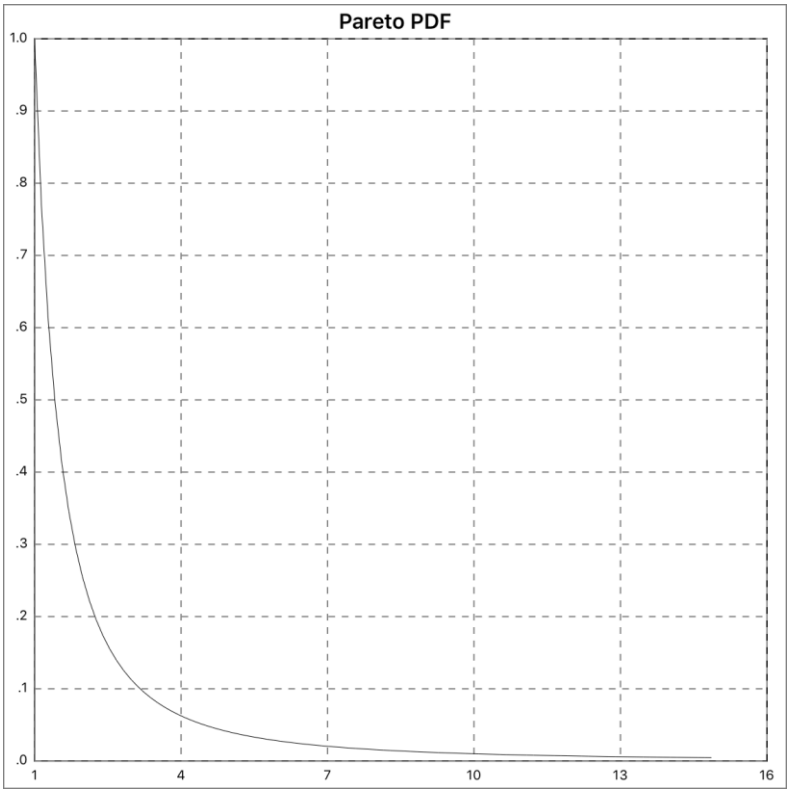

Figure 94 - Pareto PDF Plot

6.22.2.19.1 .scale

`v.scale()` – Returns scale value from the distribution initialization

For example:

- d = `Create.Distribution(DIST.PARETO)`
- `d.scale()` Returns `1`

6.22.2.19.2 .shape

`v.shape()` – Returns shape value from the distribution initialization

For example:

- d = `Create.Distribution(DIST.PARETO)`
- `d.shape()` Returns `1`

6.22.2.20 Poisson Functions

`Create.Distribution(DIST.POISSON)` – Returns *Poisson* distribution model
`Create.Distribution(DIST.POISSON, m)`

The *Poisson* distribution is a well-known statistical discrete distribution expressed in the probability of a number of events (or failures, arrivals, occurrences ...) over fixed period of time.

If the `m` argument is not specified, the mean value defaults to `1`. The range of `m` is `0 < m < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Poisson_distribution

Plot example:

```
d = Create.Distribution(DIST.POISSON, 5)
x = [ 0 : 10 : 0.1 ]

Utils.PlotIt("Poisson PDF",                \
             "Plt051_Poisson",             \
             Plot.CHART.LINE,              \
             x, d.pdf(x),                  \
             0, 10, 0, 0.2, 11, 5)
```

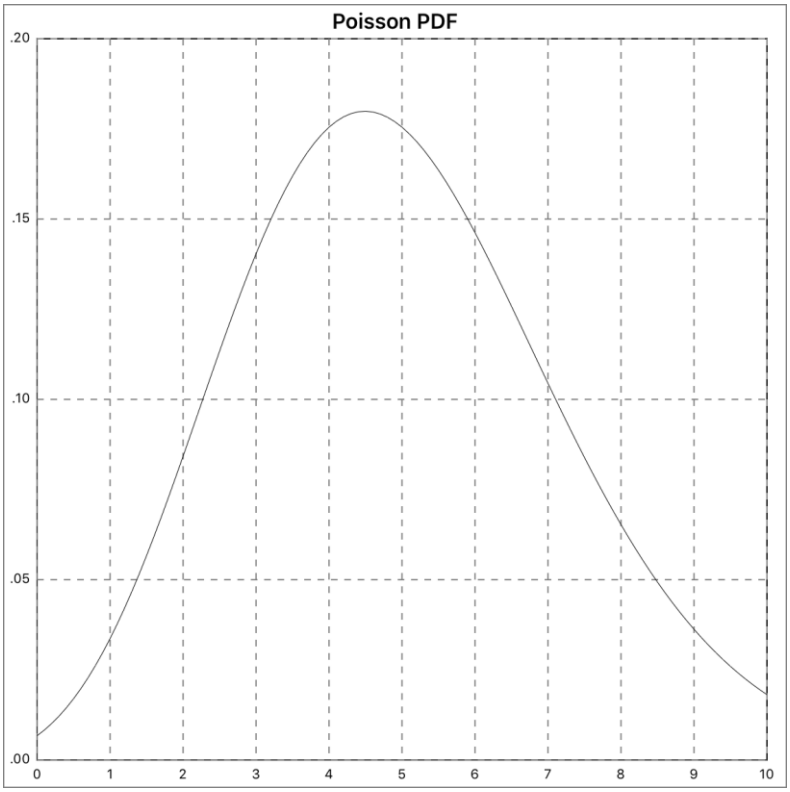

Figure 95 - Poisson PDF Plot

6.22.2.20.1 .mean

`v.mean()` – Returns mean value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.POISSON)`
- `d.mean()` Returns `1`

6.22.2.21 Rayleigh Functions

`Create.Distribution(DIST.RAYLEIGH)` – Returns *Rayleigh* distribution model
`Create.Distribution(DIST.RAYLEIGH, s)`

The *Rayleigh* distribution is a continuous distribution.

If the `s` argument is not specified, the sigma value defaults to `1`. The range of `s` is `0 < s < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Rayleigh_distribution

Plot example:

```
d = Create.Distribution(DIST.RAYLEIGH, 1)
x = [ 0 : 5 : 0.05 ]

Utils.PlotIt("Rayleigh PDF",              \
             "Plt052_Rayleigh",           \
             Plot.CHART.LINE,             \
             x, d.pdf(x),                 \
             0, 5, 0, 0.8, 6, 5)
```

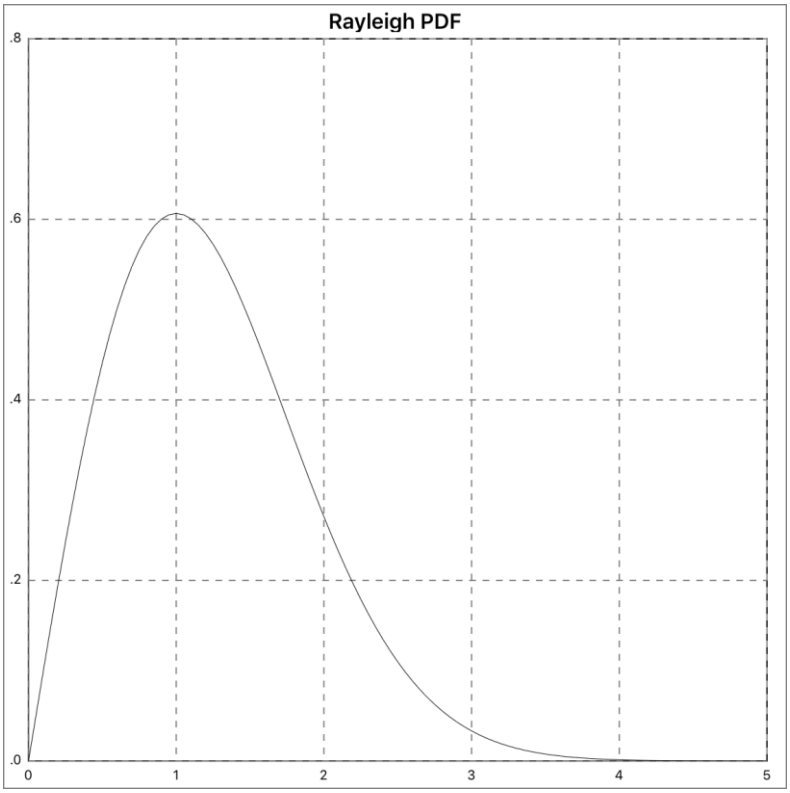

Figure 96 - Rayleigh PDF Plot

6.22.2.21.1 .sigma

`v.sigma()` – Returns sigma value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.RAYLEIGH)`
- `d.sigma()` Returns `1`

6.22.2.22 Skew Normal Functions

`Create.Distribution(DIST.SKEW_NORMAL)` – Returns *Skew Normal* distribution model
`Create.Distribution(DIST.SKEW_NORMAL, l)`
`Create.Distribution(DIST.SKEW_NORMAL, l, sc)`
`Create.Distribution(DIST.SKEW_NORMAL, l, sc, sh)`

The *Skew Normal* distribution is a variant of the *Gaussian* statistical distribution.

If the `l` argument is not specified, the location value defaults to `0`. The range of `l` is `0 ≤ l < ∞`.

If the `sc` argument is not specified, the scale value defaults to `1`. The range of `sc` is `0 < sc < ∞`.

If the `sh` argument is not specified, the shape value defaults to `0`. The range of `sh` is `0 < sh < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Skew_normal_distribution

Plot example:

```
d = Create.Distribution(DIST.SKEW_NORMAL, 0, 1, 0)
x = [ -3 : 3 : 0.06 ]

Utils.PlotIt("Skew Normal PDF",          \
             "Plt053_SkewNormal",        \
             Plot.CHART.LINE,            \
             x, d.pdf(x),                \
             -3, 3, 0, 0.5, 7, 6)
```

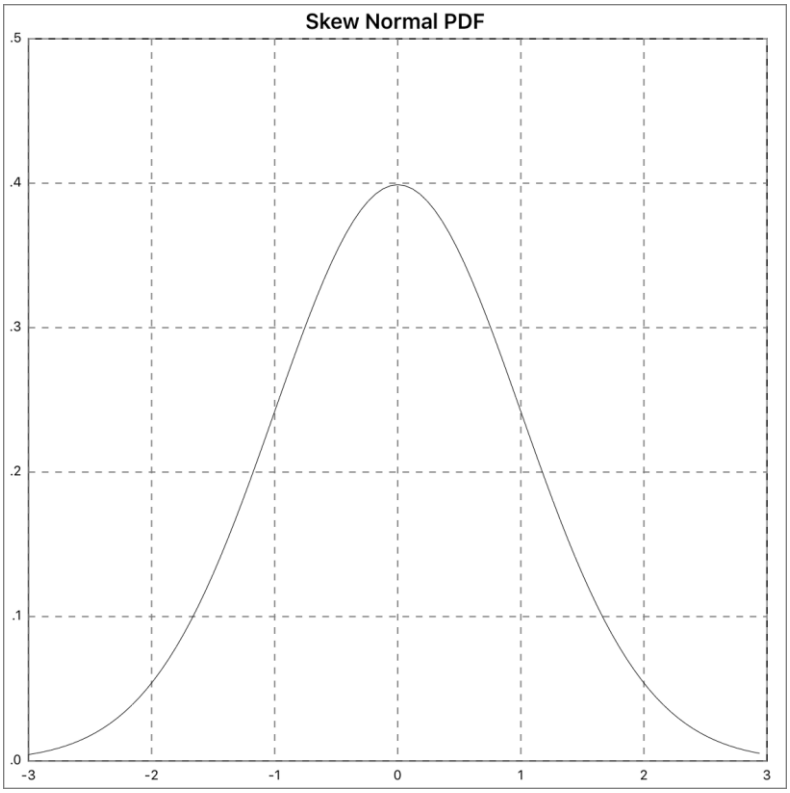

Figure 97 - Skew Normal PDF Plot

6.22.2.22.1 .location

`v.location()` – Returns location value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.SKEW_NORMAL)`
- `d.location()` Returns `0`

6.22.2.22.2 .scale

`v.scale()` – Returns scale value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.SKEW_NORMAL)`
- `d.scale()` Returns `1`

6.22.2.22.3 .shape

`v.shape()` – Returns shape value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.SKEW_NORMAL)`
- `d.shape()` Returns `0`

6.22.2.23 Student's T- distribution Functions

`Create.Distribution(DIST.STUDENTS_T, df)` – Returns *Student's T* distribution model

The *Student's T* distribution with `df` argument's number of degrees of freedom of the sample.

The range of `df` argument is `0 < df < ∞`.

⇒ NOTE: Refer to the Demo script `D02_Students` function for detailed illustration of the *Student's T* distribution. Refer to the *Demo Tour* section for installing the script.

⇒ LINK: https://en.wikipedia.org/wiki/Student%27s_t-distribution

Plot example:

```
d = Create.Distribution(DIST.STUDENTS_T, 1)
x = [ -4 : 4 : 0.08 ]

Utils.PlotIt("Student's T PDF",          \
             "Plt054_StudentT",          \
             Plot.CHART.LINE,            \
             x, d.pdf(x),                \
             -4, 4, 0, 0.4, 5, 5)
```

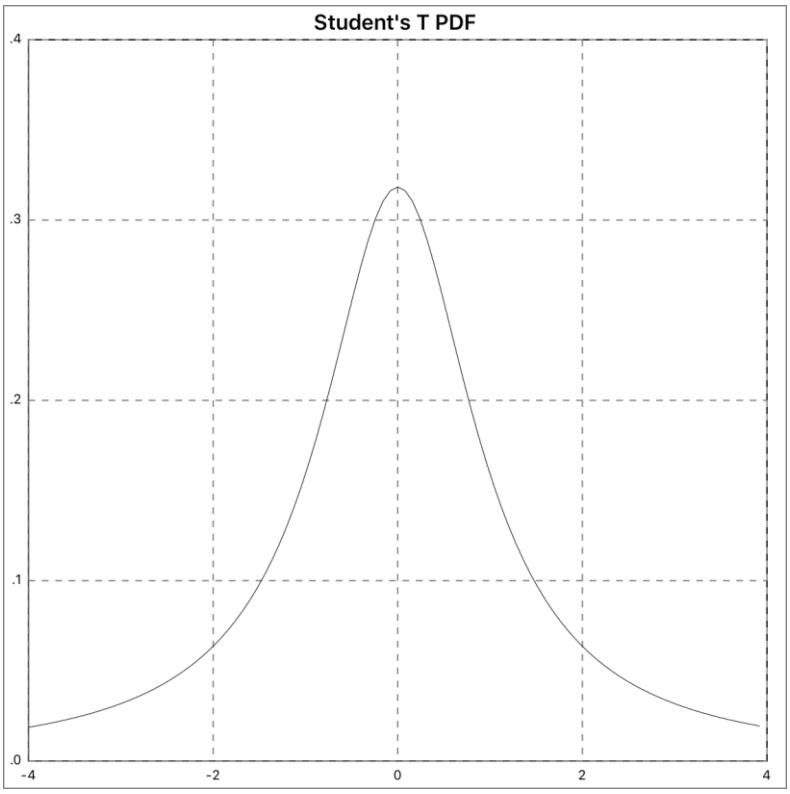

Figure 98 - Student's T PDF Plot

6.22.2.23.1 .degrees_of_freedom

`v.degrees_of_freedom()` – Returns the degrees of freedom value from the distribution initialization

For example:

- `N = 100` Sets sample size
- `d = Create.Distribution(DIST.STUDENTS_T, N - 1)`

- `d.degrees_of_freedom()` Returns `99`

6.22.2.23.2 .find_degrees_of_freedom

`v.find_degrees_of_freedom(m, a, b, s)` – Returns degrees of freedom
`v.find_degrees_of_freedom(m, a, b, s, h)`

The `d` argument is the differences from the assumed nominal variance with maximum acceptable risk of rejecting the alpha's `a` argument along the risk of falsely failing the beta's `b` argument using the variance's `v` argument to be tested.

If the `h` argument is not specified, the hint value defaults to `100`.

For example:

- `N = 100` Sets sample size
- `stddev = 0.6278908E-02` Sets sample standard deviation
- `alpha = 0.05` Sets significance level
- `variance = 0.1` Sets variance

- `d = Create.Distribution(DIST.STUDENTS_T, N - 1)`

- `d.find_degrees_of_freedom(\` Returns `13.24216`
  `variance - stddev, \`
  `alpha, alpha, variance)`

6.22.2.24 Triangular Functions

`Create.Distribution(DIST.TRIANGULAR)` – Returns *Triangular* distribution model
`Create.Distribution(DIST.TRIANGULAR, l)`
`Create.Distribution(DIST.TRIANGULAR, l, m)`
`Create.Distribution(DIST.TRIANGULAR, l, m, u)`

The *Triangular* distribution is a continuous probability distribution with a lower limit, mode, and upper limit.

If the `l` argument is not specified, the lower value defaults to `-1`. The range of `l` is `-∞ < l < ∞`.

If the `m` argument is not specified, the mode value defaults to `0`. The range of `m` argument is `-∞ < m < ∞`.

If the `u` argument is not specified, the upper value defaults to `1`. The range of `u` argument is `-∞ < u < ∞`.

The lower `l` argument is less than the mode `m` argument and mode is less than `u` argument.

⇒ LINK: https://en.wikipedia.org/wiki/Triangular_distribution

Plot example:

```
d = Create.Distribution(DIST.TRIANGULAR, -1, 0, 1)
x = [ -1 : 1 : 0.02 ]

Utils.PlotIt("Triangular PDF",          \
             "Plt055_Triangular",       \
             Plot.CHART.LINE,           \
             x, d.pdf(x),               \
             -1, 1, 0, 1, 5, 11)
```

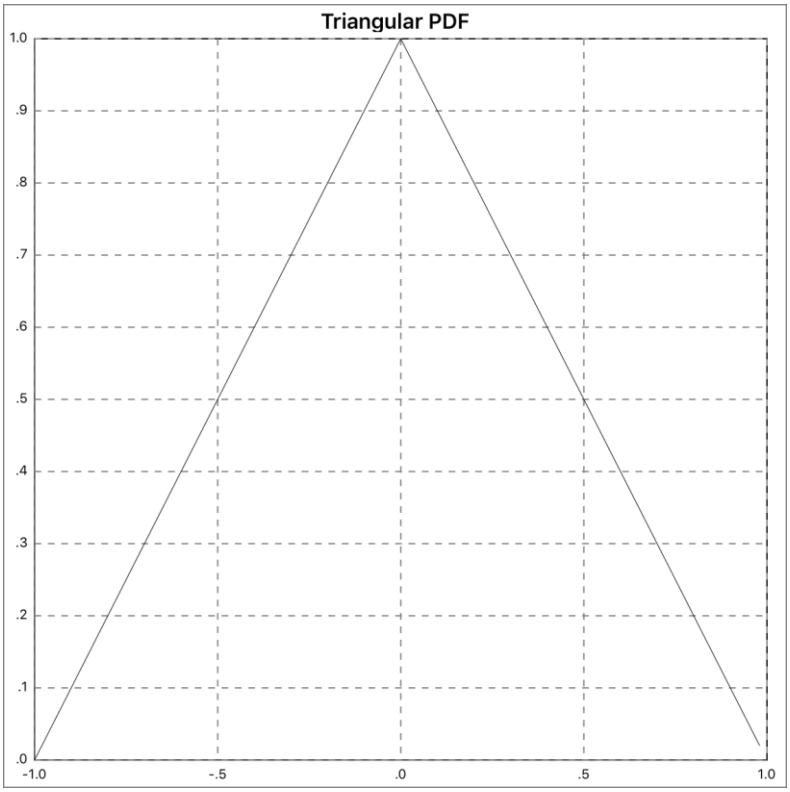

Figure 99 - Triangular PDF Plot

6.22.2.24.1 .lower

`v.lower()` – Returns lower value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.TRIANGULAR)`
- `d.lower()` Returns `-1`

6.22.2.24.2 .mode

`v.mode()` – Returns mode value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.TRIANGULAR)`
- `d.mode()` Returns `0`

6.22.2.24.3 .upper

`v.upper()` – Returns upper value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.TRIANGULAR)`
- `d.upper()` Returns `1`

6.22.2.25 Uniform Functions

`Create.Distribution(DIST.UNIFORM)` – Returns *Uniform* distribution model.
`Create.Distribution(DIST.UNIFORM, l)`
`Create.Distribution(DIST.UNIFORM, l, u)`

The *Uniform* distribution, also known as a rectangular distribution, is a probability distribution that has constant probability.

If the `l` argument is not specified, the lower value defaults to `-1`. The range of `l` is `-∞ < l < ∞`.

If the `u` argument is not specified, the upper value defaults to `1`. The range of `u` is `-∞ < u < ∞`.

The lower `l` argument must be less than the upper `u` argument.

⇒ LINK : https://en.wikipedia.org/wiki/Uniform_distribution_(continuous)

Plot example:

```
d = Create.Distribution(DIST.UNIFORM, -1, 1)
x = [ -2 : 2 : 0.04 ]

Utils.PlotIt("Uniform PDF",              \
             "Plt056_Uniform",           \
             Plot.CHART.LINE,            \
             x, d.pdf(x),                \
             -2, 2, 0, 1, 5, 11)
```

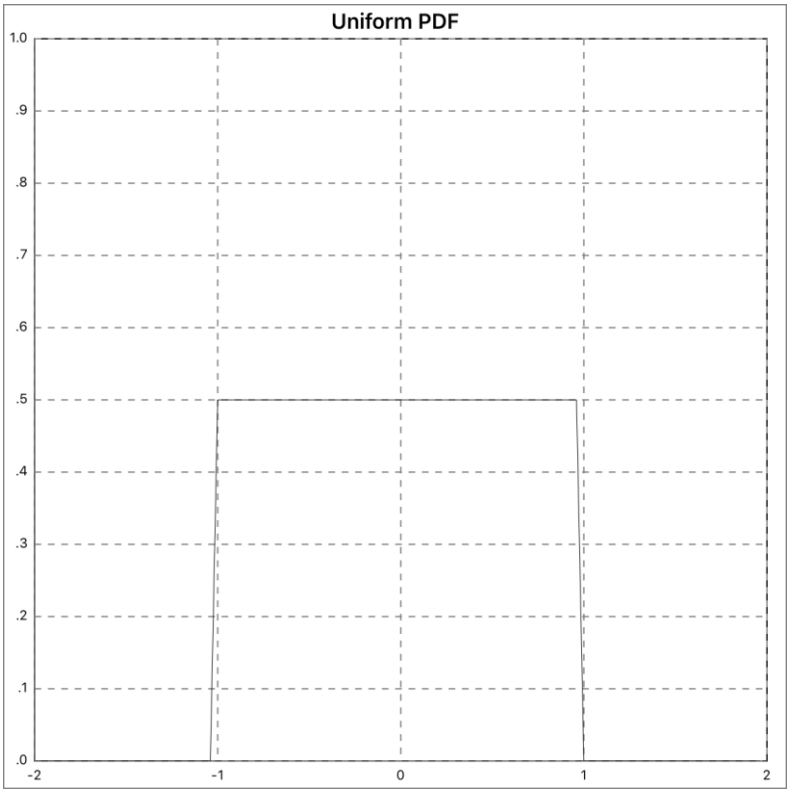

Figure 100 - Uniform PDF Plot

6.22.2.25.1 .lower

`v.lower()` – Returns lower value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.UNIFORM)`
- `d.lower()` Returns `-1`

6.22.2.25.2 .upper

`v.upper()` – Returns upper value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.UNIFORM)`
- `d.upper()` Returns `1`

`v.upper()` – Returns upper value from the distribution initialization

6.22.2.26 Weibull Functions

`Create.Distribution(DIST.WEIBULL, sc)` – Returns *Weibull* distribution model
`Create.Distribution(DIST.WEIBULL, sc, sh)`

The *Weibull* distribution is a continuous distribution.

The range of `sc` argument is `0 < sc < ∞`.

If the `sh` argument is not specified, the shape value defaults to `1`. The range of `sh` is `0 < sh < ∞`.

⇒ LINK: https://en.wikipedia.org/wiki/Weibull_distribution

Plot example:

```
d = Create.Distribution(DIST.WEIBULL, 5, 1)
x = [ 0 : 1.5 : 0.015 ]

Utils.PlotIt("Weibull PDF",              \
             "Plt057_Weibull",           \
             Plot.CHART.LINE,            \
             x, d.pdf(x),                \
             0, 2, 0, 2, 5, 11)
```

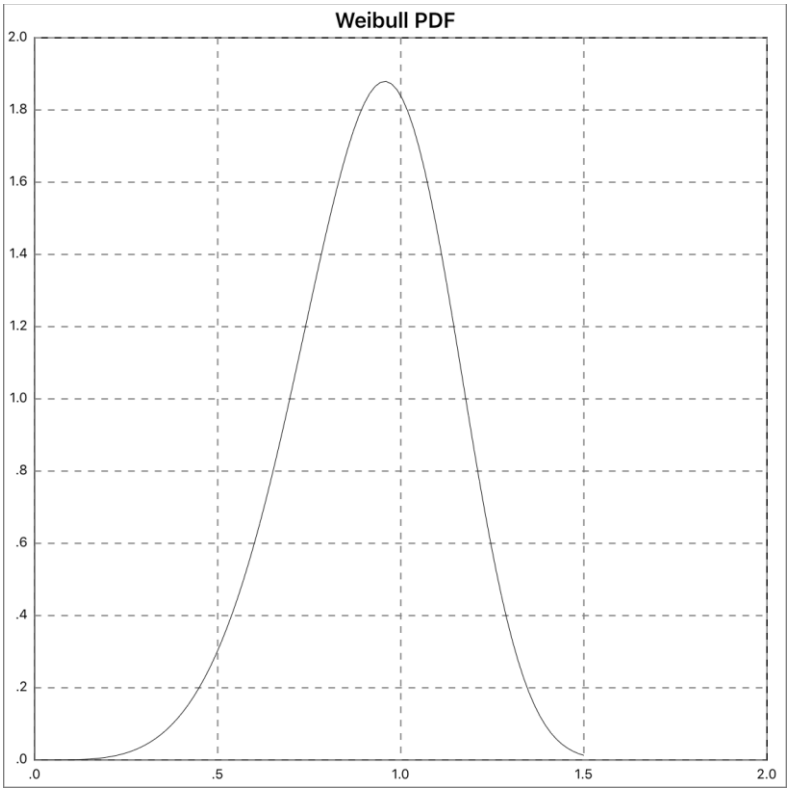

Figure 101 - Weibull PDF Plot

6.22.2.26.1 .shape

`v.shape()` – Returns shape value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.WEIBULL, 5, 1)`
- `d.shape()` Returns 5

6.22.2.26.2 .scale

`v.scale()` – Returns scale value from the distribution initialization

For example:

- `d = Create.Distribution(DIST.WEIBULL, 5, 1)`
- `d.scale()` Returns `1`

6.22.3 Constants

6.22.3.1 Dist.BERNOULLI

`Dist.BERNOULLI` – Bernoulli distribution type constant

6.22.3.2 Dist.BETAD

`Dist.BETAD` – Beta distribution type constant

6.22.3.3 Dist.BINOMIAL

Dist.BINOMIAL – Binomial distribution type constant

6.22.3.4 Dist.CAUCHY

Dist.CAUCHY – Cauchy distribution type constant

6.22.3.5 Dist.CHI_SQUARED

Dist.CHI_SQUARED – Chi-Squared distribution type constant

6.22.3.6 Dist.EXPONENTIAL

Dist.EXPONENTIAL – Exponential distribution type constant

6.22.3.7 Dist.FISHER_F

Dist.FISHER_F – Fisher's F distribution type constant

6.22.3.8 Dist.GAMMA

Dist.GAMMA – Gamma distribution type constant

6.22.3.9 Dist.GEOMETRIC

Dist.GEOMETRIC – Geometric distribution type constant

6.22.3.10 Dist.INV_CHI_SQUARED

Dist.INV_CHI_SQUARED – Inverse Chi-Squared distribution type constant

6.22.3.11 Dist.INV_GAMMA

Dist.INV_GAMMA – Inverse Gamma distribution type constant

6.22.3.12 Dist.INV_GAUSSIAN

Dist.INV_GAUSSIAN – Inverse Gaussian distribution type constant

6.22.3.13 Dist.LAPLACE

Dist.LAPLACE – Laplace distribution type constant

6.22.3.14 DIST.LOG_NORMAL

Dist.LOG_NORMAL – Logarithm Normal distribution type constant

6.22.3.15 Dist.LOGISTIC

Dist.LOGISTIC – Logistic distribution type constant

6.22.3.16 Dist.NEG_BINOMIAL

Dist.NEG_BINOMIAL – Negative Binomial distribution type constant

6.22.3.17 Dist.NON_CEN_CHI_SQUARED

Dist.NON_CEN_CHI_SQUARED – Non-Central Chi-Squared distribution type constant

6.22.3.18 Dist.NORMAL

Dist.NORMAL – Normal distribution type constant

6.22.3.19 Dist.PARETO

Dist.PARETO – Pareto distribution type constant

6.22.3.20 Dist.POISSON

Dist.POISSON – Poisson distribution type constant

6.22.3.21 Dist.RAYLEIGH

Dist.RAYLEIGH – Rayleigh distribution type constant

6.22.3.22 Dist.SKEW_NORMAL

Dist.SKEW_NORMAL – Skew Normal distribution type constant

6.22.3.23 Dist.STUDENTS_T

Dist.STUDENTS_T – Student's T distribution type constant

6.22.3.24 Dist.TRIANGULAR

Dist.TRIANGULAR – Triangular distribution type constant

6.22.3.25 Dist.UNIFORM

Dist.UNIFORM – Uniform distribution type constant

6.22.3.26 Dist.WEIBULL

Dist.WEIBULL – Weibull distribution type constant

6.23 Statistics

6.23.1 Stat.mean

`Stat.mean(x)` – Returns mean value

The `x` argument is used to calculate its mean.

If it is a one-dimension array, all the values are summed up to a scalar value.

If it is a two-dimension array, each column will be summed up into a one-dimension array.

⇒ LINK: https://en.wikipedia.org/wiki/Mean

For example:

- `Stat.mean([ 1.5, 2.5, 3.5])` Returns `2.5`
- `Stat.mean([[1.5, 2.5, 3.5], \`
  `[4.5, 5.5, 6.5]])` Returns `3`, `4`, and `5`

6.23.2 Stat.stddev

`Stat.stddev(x)` – Returns standard deviation value

The `x` argument is used to calculate its standard deviation.

If it is a one-dimension array, all the values are summed up to a scalar value.

If it is a two-dimension array, each column will be summed up into a one-dimension array.

⇒ LINK: https://en.wikipedia.org/wiki/Standard_deviation

For example:

- `Stat.stddev([ 1.5, 2.5, 3.5])` Returns `0.81649658092772603`
- `Stat.stddev([[1.5, 2.5, 3.5], \ [4.5, 5.5, 6.5]])` Returns `1.5`, `1.5`, and `1.5`

6.23.3 Stat.variance

`Stat.variance(x)` – Returns variance value

The `x` argument is used to calculate its variance.

If it is a one-dimension array, all the values are summed up to a scalar value.

If it is a two-dimension array, each column will be summed up into a one-dimension array.

⇒ LINK: https://en.wikipedia.org/wiki/Variance

For example:

- `Stat.variance([ 1.5, 2.5, 3.5])` Returns `0.66666666666666663`
- `Stat.variance([[1.5, 2.5, 3.5], \ [4.5, 5.5, 6.5]])` Returns `2.25`, `2.25`, and `2.25`

6.23.4 Stat.covariance

`Stat.covariance(x, y)` – Returns co-variance values

The `x` and `y` arguments are used to calculate its co-variances.

If it is a one-dimension array, all the values are summed up to a scalar value.

If it is a two-dimension array, each column will be summed up into a one-dimension array.

⇒ LINK: https://en.wikipedia.org/wiki/Covariance

For example:

- `Stat.covariance([[1.5, 2.5, 3.5], \ [4.5, 5.5, 6.5]])` Returns `0.66666666666666663`
- `Stat.covariance([[1.5, 2.5, 3.5], \ [4.5, 5.5, 6.5]],\ [[4.5, 5.5, 6.5], \ [7.5, 8.5, 9.5]])` Returns `2.25`, `2.25`, and `2.25`

6.23.5 Stat.corrcoef

`Stat.corrcoef(x, y)` – Returns correlation coefficient values

The `x` and `y` arguments are used to calculate its correlation coefficient.

If it is a one-dimension array, all the values are summed up to a scalar value.

If it is a two-dimension array, each column will be summed up into a one-dimension array.

⇒ LINK: https://en.wikipedia.org/wiki/Correlation_coefficient

For example:

- `Stat.corrcoef([[1.5, 2.5, 3.5], \ [4.5, 5.5, 6.5]])` Returns `3`
- `Stat.corrcoef([[1.5, 2.5, 3.5], \ [4.5, 5.5, 6.5]], \ [[4.5, 5.5, 6.5], \ [7.5, 8.5, 9.5]])` Returns `1`, `1`, and `1`

6.23.6 Stat.find_degrees_of_freedom

`Stat.find_degrees_of_freedom(d, ...)` – Returns distribution's degrees of freedom

The `d` argument is *Chi-Squared* or *Student's T* distribution type. The following arguments are dependent on the distribution type used to find the degrees of freedom.

Using *Chi-Squared* distribution:

```
Stat.find_degrees_of_freedom(Dist.CHI_SQUARED, f, a, b, v)
Stat.find_degrees_of_freedom(Dist.CHI_SQUARED, f, a, b, v, h)
```

The `f` argument is the difference from the assumed nominal variance that is to be detected.

The `a` argument is the maximum acceptable risk of rejecting the null hypothesis when it is, in fact, true.

The `b` argument is the maximum acceptable risk of falsely failing to reject the null hypothesis.

The `v` argument is the nominal variance being tested against.

If the `h` argument is not specified, it will use default hint for where to start looking for a result: the current sample size would be a good choice. The default value is `100`.

For example:

- `Stat.find_degrees_of_freedom(Dist.CHI_SQUARED, \ 55, 0.1, 0.1, 100)` Returns `69.143357`

For *Student's T* distribution:

```
Stat.find_degrees_of_freem(Dist.STUDENTS_T, f, a, b, s)
```

The `f` argument is the difference from true mean to detect.

The `a` argument is the maximum risk of falsely rejecting the null-hypothesis.

The `b` argument is the maximum risk of falsely failing to reject the null-hypothesis.

The `s` argument is the standard deviation.

For example:

- `Stat.find_degrees_of_freedom(Dist.STUDENTS_T, \ 1.3, 0.05, 0.1, 0.13)` Returns `0.811541302229`

6.23.7 Stat.find_lower_bound_on_p

`Stat.find_lower_bound_on_p(d, ...)` – Returns distribution's lower bounds

The `d` argument is *Binomial*, *Negative Binomial*, or *Geometric* distribution type. The following arguments are dependent on the distribution type used to find the lower bounds.

Using *Binomial* distribution:

```
Stat.find_lower_bound_on_p(d, t, s, p)
Stat.find_lower_bound_on_p(d, t, s, p, m)
```

The `t` argument is the number of trials.

The `s` argument is the number of successes.

The `p` argument is the maximum acceptable risk of falsely failing to reject the null hypothesis.

If the `m` argument is not specified, method's default interval is `Stat.CLOPPER_PEARSON_EXACT_INTERVAL`. The other interval option is `Stat.JEFFREYS_PRIOR_INTERVAL`.

For example:

- `Stat.find_lower_bound_on_p(Dist.BINOMIAL, \ 20, 4, 0.05 ÷ 2)` Returns `0.057333997050`

Using *Negative Binomial* distribution:

```
Stat.find_lower_bound_on_p(d, t, s, p)
```

The `t` argument is the number of trials.

The `s` argument is the number of successes.

The `p` argument is the maximum acceptable risk of falsely failing to reject the null hypothesis.

For example:

- `Stat.find_lower_bound_on_p(Dist.NEG_BINOMIAL, \`
  `20, 4, 0.05 ÷ 2)` Returns `0.057333997050`

Using *Geometric* distribution:

```
Stat.find_lower_bound_on_p(Dist.GEOMETRIC, t, p)
```

The `t` argument is the number of trials.

The `p` argument is the maximum acceptable risk of falsely failing to reject the null hypothesis.

For example:

- `Stat.find_lower_bound_on_p(Dist.GEOMETRIC, \`
  `100, 0.05 ÷ 2)` Returns `0.000253146033`

6.23.8 Stat.find_upper_bound_on_p

`Stat.find_upper_bound_on_p(d, ...)` – Returns distribution's upper bounds.

The `d` argument is Binomial, Negative Binomial, or Geometric distribution type. The following arguments are dependent on the distribution type used to find the upper bounds.

Using *Binomial* distribution:

```
Stat.find_upper_bound_on_p(d, t, s, p)
Stat.find_upper_bound_on_p(d, t, s, p, m)
```

The `t` argument is the number of trials.

The `s` argument is the number of successes.

The `p` argument is the maximum acceptable risk of falsely failing to reject the null hypothesis.

If the `m` argument is not specified, method's default interval is `Stat.CLOPPER_PEARSON_EXACT_INTERVAL`. The other interval option is `Stat.JEFFREYS_PRIOR_INTERVAL`.

For example:

- `Stat.find_upper_bound_on_p(Dist.BINOMIAL, 20, 4, 0.05 ÷ 2)` \ Returns `0.436614002997`

Using *Negative Binomial* distribution:

`Stat.find_upper_bound_on_p(d, t, s, p)`

The `t` argument is the number of trials.

The `s` argument is the number of successes.

The `p` argument is the maximum acceptable risk of falsely failing to reject the null hypothesis.

For example:

- `Stat.find_upper_bound_on_p(Dist.NEG_BINOMIAL, 20, 4, 0.05 ÷ 2)` \ Returns `0.395784551267`

Using *Geometric* distribution:

`Stat.find_upper_bound_on_p(Dist.GEOMETRIC, t, p)`

The `t` argument is the number of trials.

The `p` argument is the maximum acceptable risk of falsely failing to reject the null hypothesis.

For example:

- `Stat.find_upper_bound_on_p(Dist.GEOMETRIC, 100, 0.05 ÷ 2)` \ Returns `0.036575744983`

6.23.9 Constants

6.23.9.1 Stat.CLOPPER_PEARSON_EXACT_INTERVAL

`Stat.CLOPPER_PEARSON_EXACT_INTERVAL` – Binomial's interval produces an overly conservative coverage, sometimes much greater than the requested coverage level

6.23.9.2 Stat.JEFFREYS_PRIOR_INTERVAL

`Stat.JEFFREYS_PRIOR_INTERVAL` – Binomial's interval average coverage, typically very close to the requested coverage level

6.24 Special

6.24.1 Special.tgamma

`special.tgamma(x)` – Returns true *Gamma* of `x`

The `x` argument is expressed as $\int t^{x-1}e^{-t}dt$ over an integral from 0 to ∞.

⇒ LINK: https://en.wikipedia.org/wiki/Gamma_function

Plot example:

```
xMin    = -4.0
xMax    = 6.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y       = Special.tgamma(x)

Utils.PlotIt("T-Gamma",                  \
             "Plt058_t_gamma",           \
             Plot.CHART.LINE,            \
             x, y,                       \
             xMin, xMax, -100.0, 100.0, 0, 0)
```

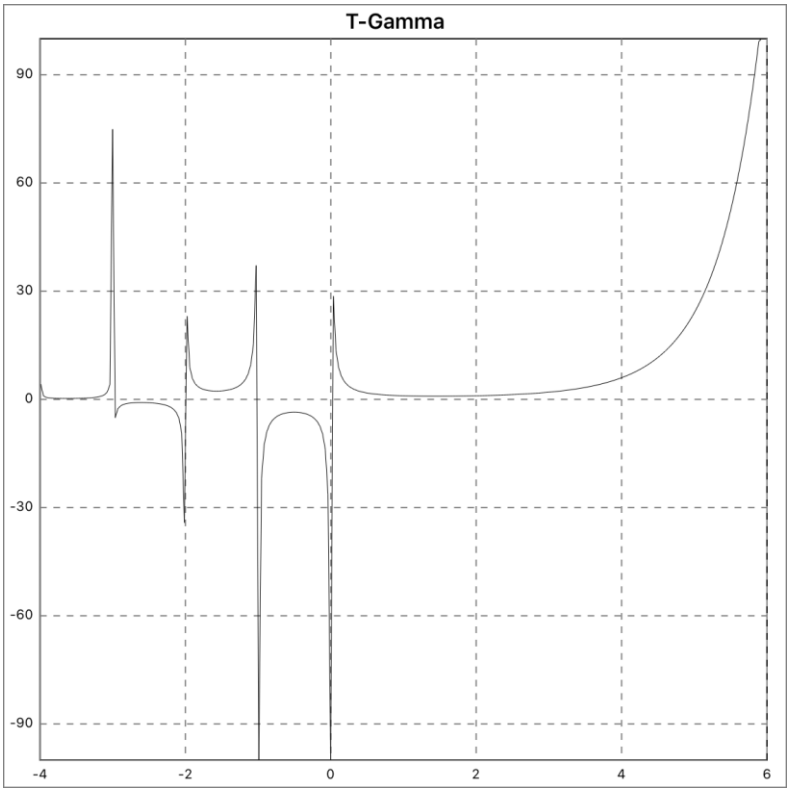

Figure 102 - T-Gamma Plot

6.24.2 Special.lgamma

`special.lgamma(x)` – Returns *Logarithm Gamma* of `x`

The `x` argument is expressed as $ln \mid \Gamma(x) \mid$.

Plot example:

```
xMin     = -5.0
xMax     = 10.0
inc      = 0.01

x        = Utils.PlotX(xMin + inc, xMax)
y        = Special.lgamma(x)

Utils.PlotIt("Log Gamma",                \
             "Plt059_gamma",             \
             Plot.CHART.LINE,            \
             x, y,                       \
             xMin, xMax, -3.0, 13.0, 0, 0)
```

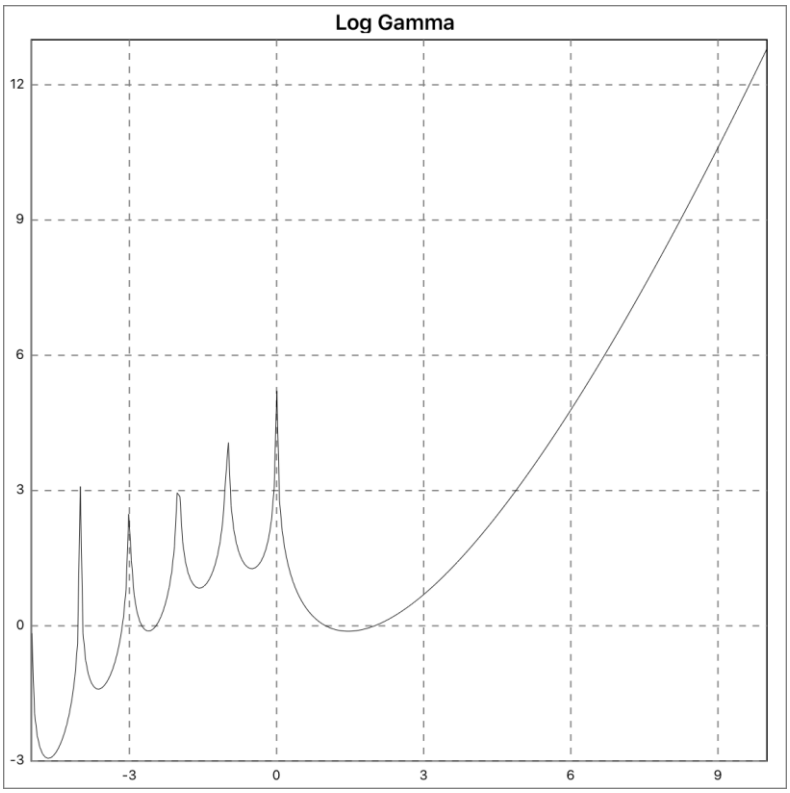

Figure 103 - Log Gamma Plot

6.24.3 Special.digamma

`special.digamma(x)` – Returns *Di-Gamma* of `x`

The `x` argument is logarithmic derivative of the Gamma function.

⇒ LINK: https://en.wikipedia.org/wiki/Digamma_function

Plot example:

```
xMin     = -5.0
xMax     = 10.0
inc      = 0.01
```

```
x       = Utils.PlotX(xMin + inc, xMax)
y       = Special.digamma(x)

Utils.PlotIt("Di-Gamma",                \
             "Plt060_di_gamma",         \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, -10.0, 10.0, 0, 0)
```

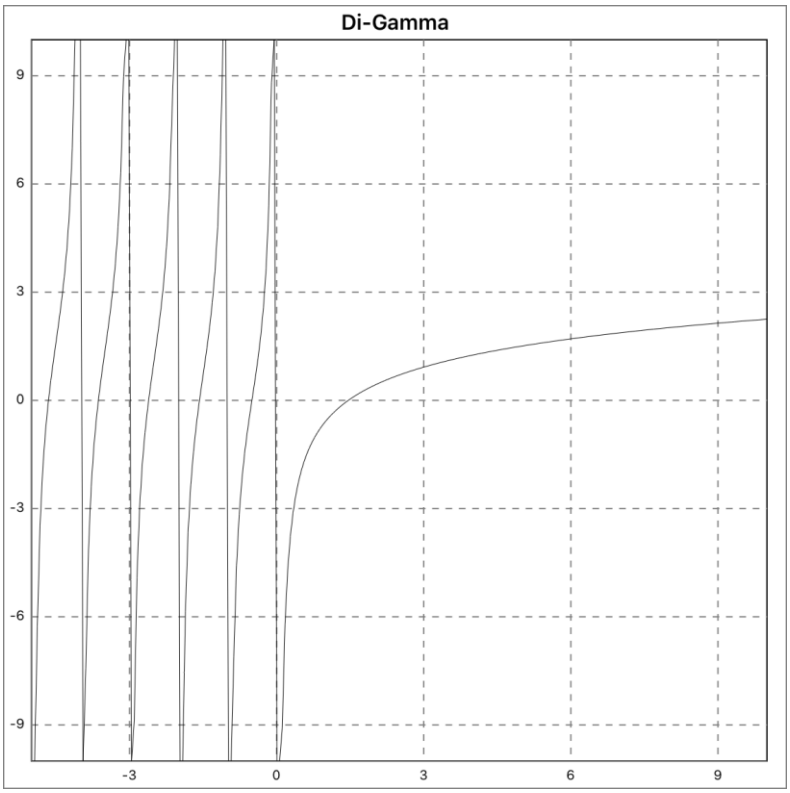

Figure 104 - Di-Gamma Plot

6.24.4 Special.trigamma

`special.trigamma(x)` – Returns *Tri-Gamma* of `x`

The `x` argument is the derivative of the *Di-Gamma* function.

Plot example:

```
xMin    = -5.0
xMax    = 5.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y       = Special.trigamma(x)

Utils.PlotIt("Tri-Gamma",               \
             "Plt061_tri_gamma",        \
             Plot.CHART.LINE,           \
```

```
                x, y,                           \
                xMin, xMax, 0.0, 400.0, 0, 0)
```

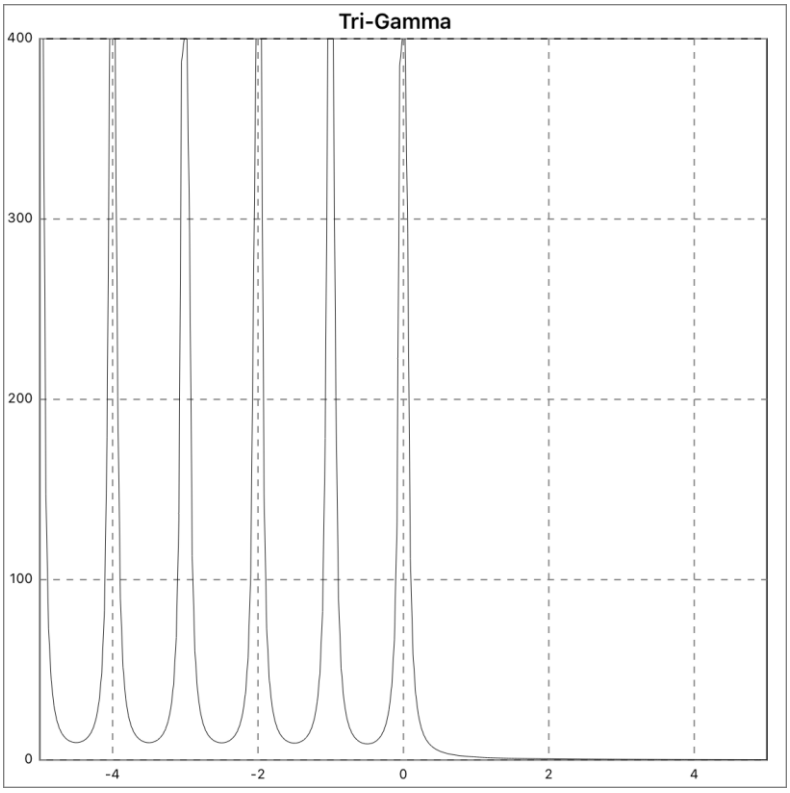

Figure 105 - Tri-Gamma Plot

6.24.5 Special.polygamma

`special.polygamma(x, n)` – Returns *Polynomial Gamma* of *n*th derivative of `x`

The `x` argument is defined as the `n` argument's derivative of the *Di-Gamma* function.

⇒ LINK: https://en.wikipedia.org/wiki/Polygamma_function

Plot example:

```
xMin    = -6.0
xMax    = 5.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y       = Special.polygamma(x, 2)

Utils.PlotIt("Poly Gamma(x, 2)",        \
             "Plt062_poly_gamma2",      \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, -50.0, 500.0, 0, 0)
```

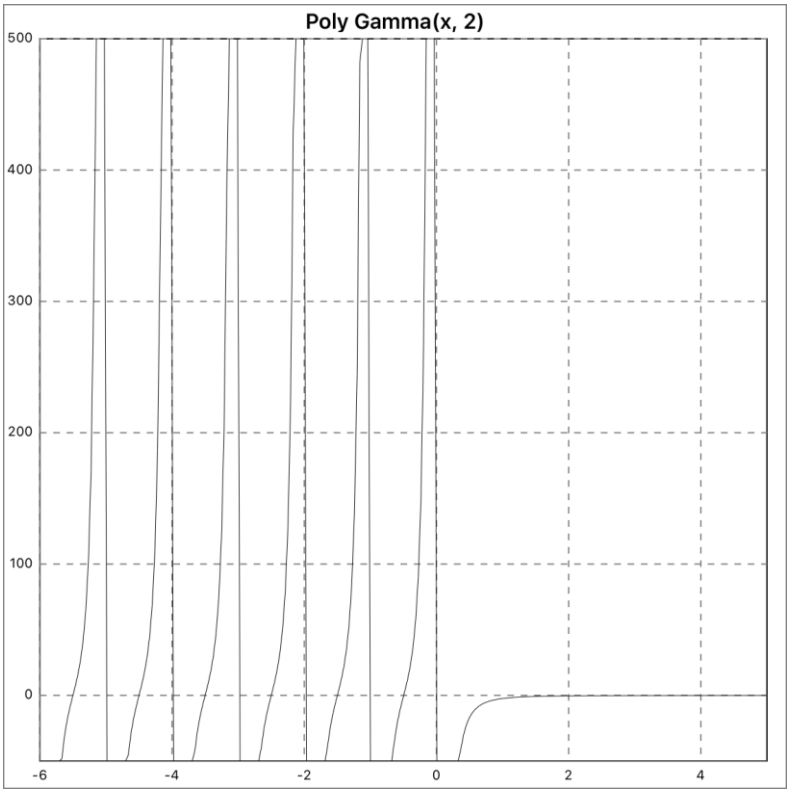

Figure 106 - Poly Gamma 2 Plot

Another plot example:

```
xMin    = -6.0
xMax    = 5.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y       = Special.polygamma(x, 3)

Utils.PlotIt("Poly Gamma(x, 3)",        \
             "Plt063_poly_gamma3",      \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, 0.0, 1000.0, 0, 0)
```

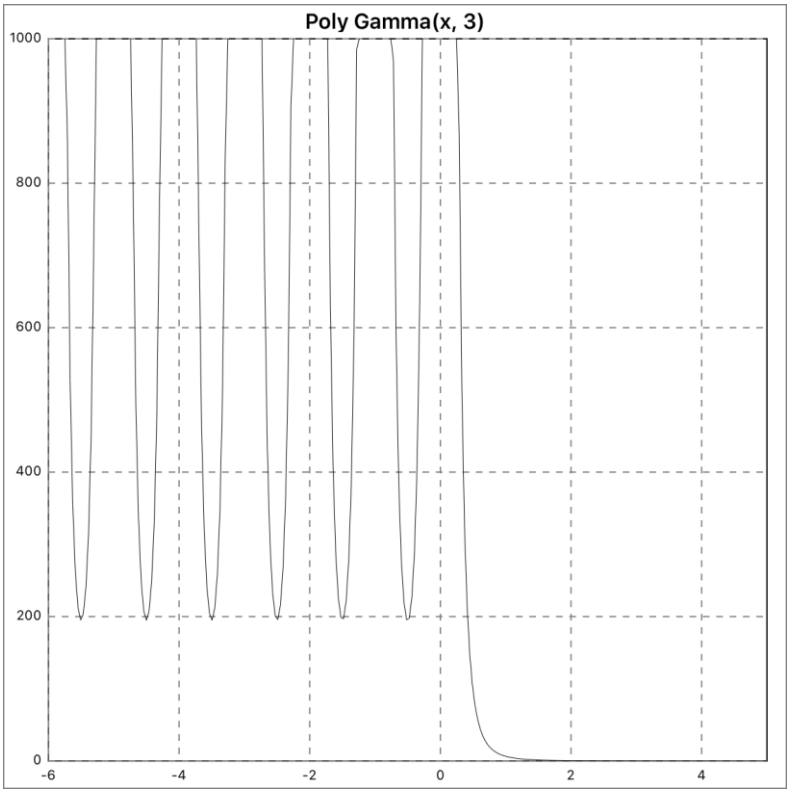

Figure 107 - Poly Gamma 3 Plot

6.24.6 Special.tgamma_delta

`special.tgamma_delta(x, a)` – Returns true *Gamma Delta* ratio pivot `a` of `x`

The `x` argument is expressed as $\Gamma(x) / \Gamma(x+a)$.

Plot example:

```
xMin    = 0.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.tgamma_delta(x, 1.0)
y2      = Special.tgamma_delta(x, 0.2)
y3      = Special.tgamma_delta(x, 0.5)
y4      = Special.tgamma_delta(x, -0.2)
y5      = Special.tgamma_delta(x, -0.)

Utils.Graph5It("T Gamma Delta",          \
               "Plt064_t_gamma_delta",   \
               Plot.CHART.LINE,          \
               x, y1, y2, y3, y4, y5,    \
               "delta = 1.0",            \
               "delta = 0.2",            \
               "delta = 0.5",            \
               "delta = -0.2",           \
               "delta = -0.5",           \
```

```
                xMin, xMax, 0.0, 6.0, 0, 0)
```

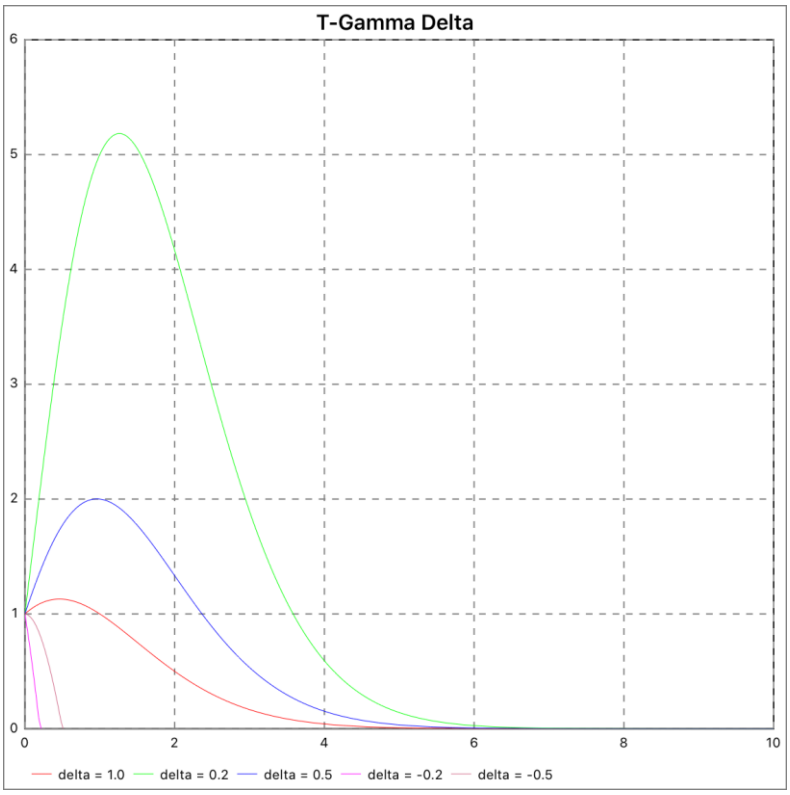

Figure 108 - T-Gamma Delta Plot

6.24.7 Special.tgamma_lower

`special.tgamma_lower(x, a)` – Returns full (non-normalized) lower incomplete *Gamma* function pivot `a` of `x`

The `x` and `a` arguments are expressed as $\int t^{a-1}e^{-t}dt$ over an integral from `0` to `x`.

Plot example:

```
xMin    = 0.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.tgamma_lower(x, 1.0)
y2      = special.tgamma_lower(x, 0.2)
y3      = special.tgamma_lower(x, 0.5)

Utils.Graph3It("T-Gamma Lower",             \
               "Plt066_t_gamma_lower",      \
               Plot.CHART.LINE,             \
               x, y1, y2, y3,               \
               "delta = 1.0",               \
               "delta = 0.2",               \
               "delta = 0.5",               \
               xMin, xMax, 0.0, 6.0, 0, 0)
```

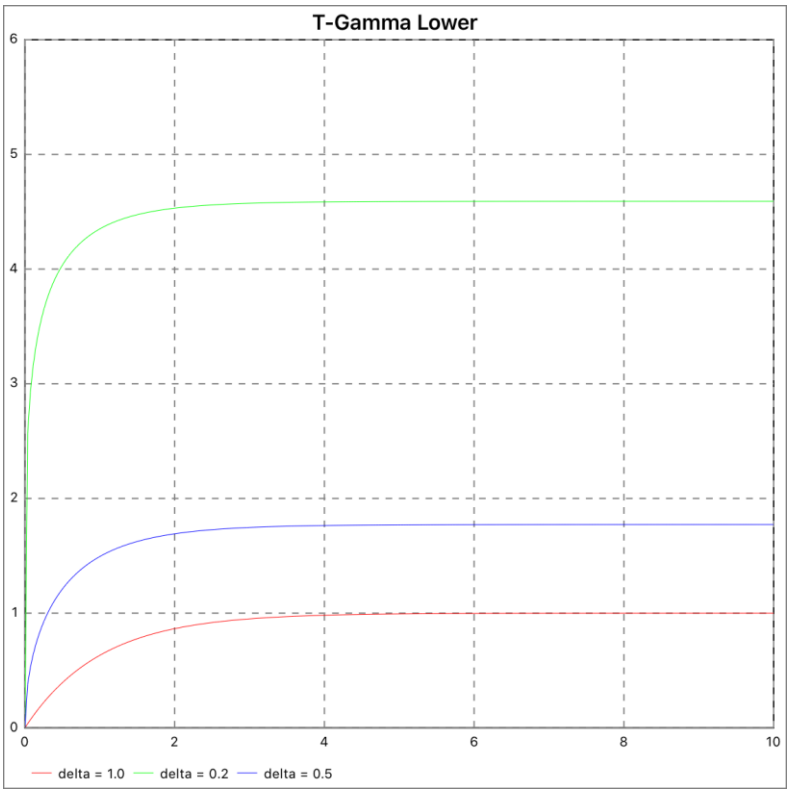

Figure 109 - T-Gamma Lower Plot

6.24.8 Special.tgamma_ratio

`special.tgamma_ratio(x, a)` – Returns true *Gamma Ratio* of `a` over `x`

The `x` and `a` arguments are expressed as $\Gamma(a) / \Gamma(x)$.

Plot example:

```
xMin    = 1.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.tgamma_ratio(x, 1.0)
y2      = special.tgamma_ratio(x, 0.2)
y3      = special.tgamma_ratio(x, 0.5)

Utils.Graph3It("T-Gamma Ratio",             \
               "Plt067_t_gamma_ratio",      \
               Plot.CHART.LINE,             \
               x, y1, y2, y3,               \
               "delta = 1.0",               \
               "delta = 0.2",               \
               "delta = 0.5",               \
               xMin, xMax, 0.0, 6.0, 0, 0)
```

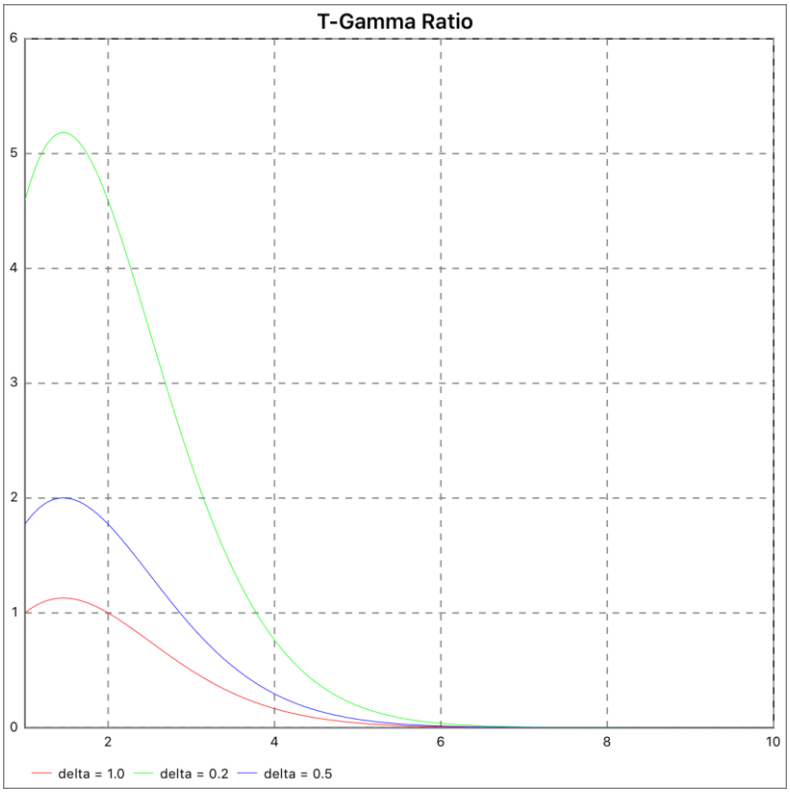

Figure 110 - T-Gamma Ratio Plot

6.24.9 Special.gamma_p

`special.gamma_p(x, a)` – Returns normalized lower incomplete *Gamma* function pivot `a` of `x`

The `x` and `a` arguments are expressed as $1 / \Gamma(a) \cdot \int t^{a-1} e^{-t} dt$ over an integral from `0` to `x`. The ranges of `a` > `0` and `x` ≥ `0`.

Plot example:

```
xMin    = 0.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.gamma_p(x, 0.5)
y2      = Special.gamma_p(x, 1.0)
y3      = Special.gamma_p(x, 5.0)
y4      = Special.gamma_p(x, 10.0)

Utils.Graph4It("Gamma P",                \
               "Plt068_gamma_p",         \
               Plot.CHART.LINE,          \
               x, y1, y2, y3, y4,        \
               "a = 0.5",                \
               "a = 1.0",                \
               "a = 5.0",                \
               "a = 10.0",               \
               xMin, xMax, 0.0, 1.0, 0, 0)
```

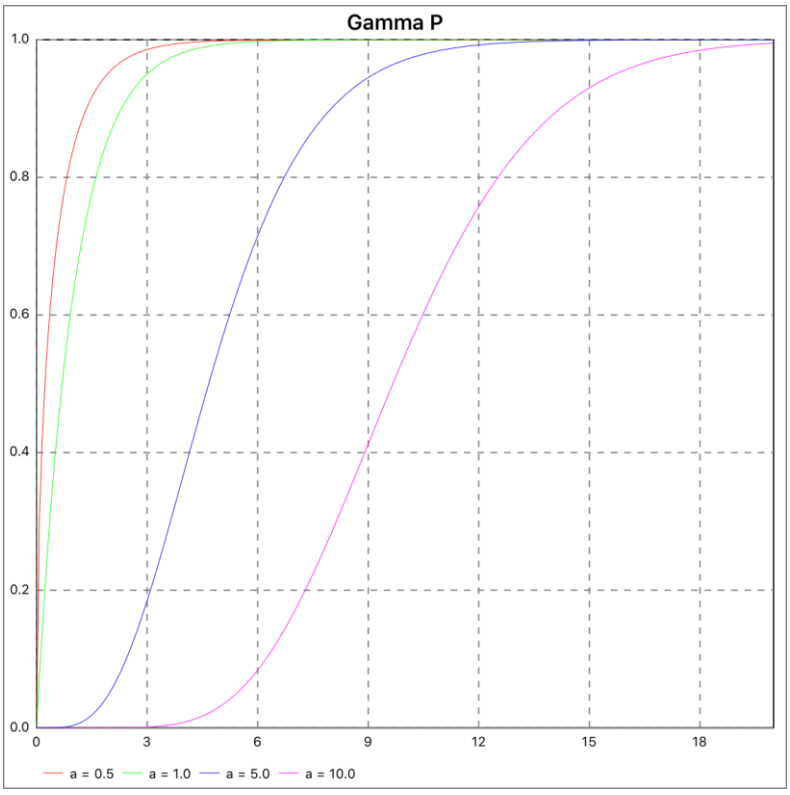

Figure 111 - Gamma P Plot

6.24.10 Special.gamma_p_der

`special.gamma_p_der(x, a)` – Returns partial derivative with respect to `x` of the incomplete *Gamma*

The `x` and `a` arguments are expressed as $e^{-x}x^{a-1}/\boldsymbol{\Gamma}(a)$.

Plot example:

```
xMin    = 0.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.gamma_p_der(x, 1.0)
y2      = special.gamma_p_der(x, 5.0)
y3      = special.gamma_p_der(x, 10.0)

Utils.Graph3It("Gamma P Derivative",        \
               "Plt069_gamma_p_der",        \
               Plot.CHART.LINE,             \
               x, y1, y2, y3,               \
               "a = 1.0",                   \
               "a = 5.0",                   \
               "a = 10.0",                  \
               xMin, xMax, 0.0, 1.0, 0, 0)
```

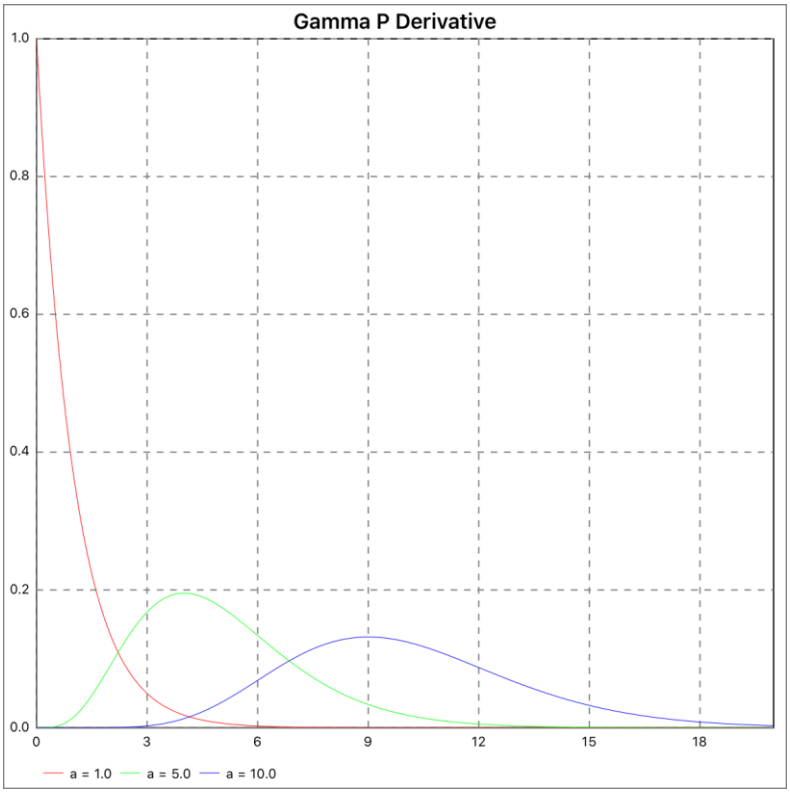

Figure 112 - Gamma P Derivative Plot

6.24.11 Special.gamma_p_inv

`special.gamma_p_inv(x, a)` – Returns inverse normalized lower incomplete *Gamma* function pivot `a` of `x`

The ranges of `a` > `0` and `x` > `0`.

Plot example:

```
xMin    = 0.1
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.gamma_p_inv(x, 0.5)
y2      = special.gamma_p_inv(x, 1.0)
y3      = special.gamma_p_inv(x, 5.0)
y4      = special.gamma_p_inv(x, 10.0)

Utils.Graph4It("Inverse Gamma P",           \
               "Plt070_inv_gamma_p",        \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4,           \
               "a = 0.5",                   \
               "a = 1.0",                   \
               "a = 5.0",                   \
               "a = 10.0",                  \
               xMin, xMax, 0.0, 20.0, 0, 0)
```

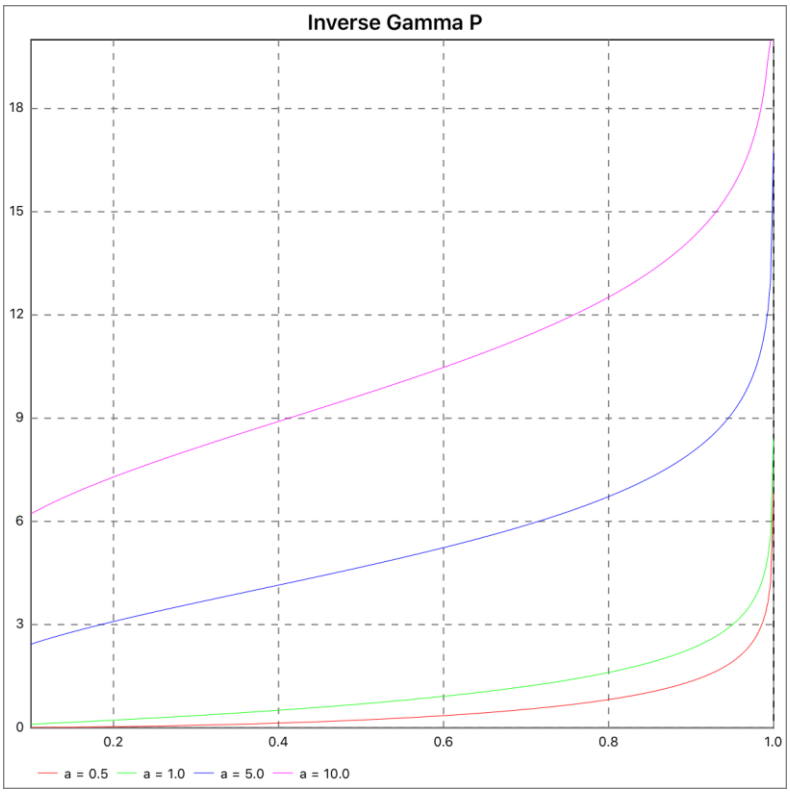

Figure 113 - Inverse Gamma P Plot

6.24.12 Special.gamma_p_inv_a

`special.gamma_p_inv_a(x, a)` – Returns inverse normalized lower incomplete *Gamma* function pivot `a` of `x`

The ranges of `a > 0` and `x > 0`.

Plot example:

```
xMin    = 0.1
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.gamma_p_inv_a(x, 0.5)
y2      = special.gamma_p_inv_a(x, 1.0)
y3      = special.gamma_p_inv_a(x, 5.0)
y4      = special.gamma_p_inv_a(x, 10.0)

Utils.Graph4It("Inverse Gamma P A",          \
               "Plt071_inv_gamma_p_a",       \
               Plot.CHART.LINE,              \
               x, y1, y2, y3, y4,            \
               "a = 0.5",                    \
               "a = 1.0",                    \
               "a = 5.0",                    \
               "a = 10.0",                   \
```

```
xMin, xMax, 0.0, 20.0, 0, 0)
```

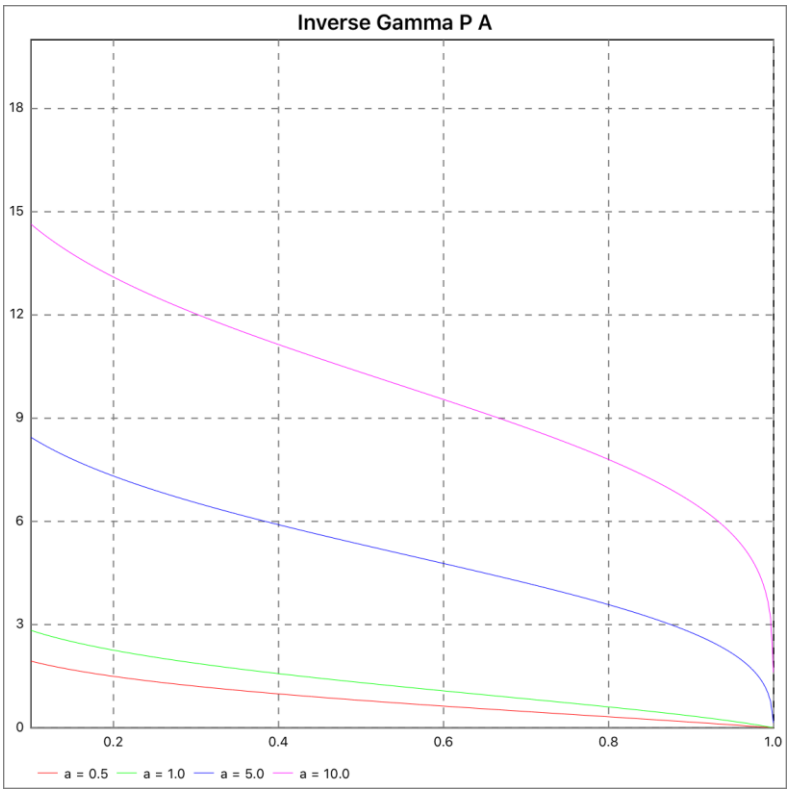

Figure 114 - Inverse Gamma P A Plot

6.24.13 Special.gamma_q

`special.gamma_q(x, a)` – Returns normalized upper incomplete *Gamma* function of `a` and `x`

The `x` and `a` arguments are expressed as $1 / \Gamma(a) \cdot \int t^{a-1}e^{-t}dt$ over an integral from `x` to ∞. The ranges of `a > 0` and `x ≥ 0`.

Plot example:

```
xMin    = 0.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.gamma_q(x, 0.5)
y2      = Special.gamma_q(x, 1.0)
y3      = Special.gamma_q(x, 5.0)
y4      = Special.gamma_q(x, 10.0)

Utils.Graph4It("Gamma Q",                \
               "Plt072_gamma_q",         \
               Plot.CHART.LINE,          \
               x, y1, y2, y3, y4,        \
               "a = 0.5",                \
               "a = 1.0",                \
               "a = 5.0",                \
```

```
                "a = 10.0",                 \
                xMin, xMax, 0.0, 1.0, 0, 0)
```

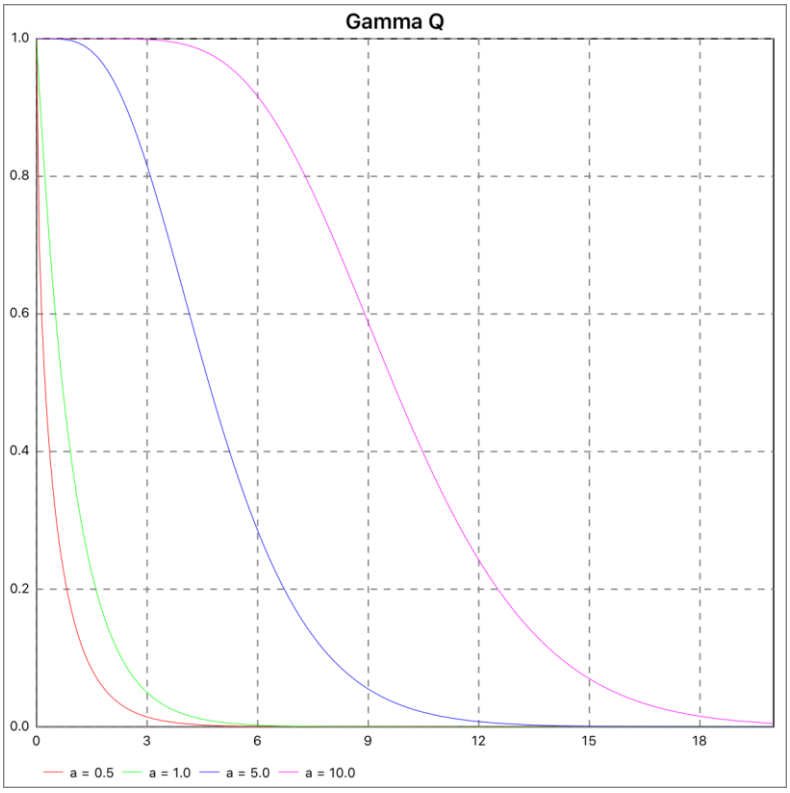

Figure 115 - Gamma Q Plot

6.24.14 Special.gamma_q_inv

`special.gamma_q_inv(x, a)` – Returns inverse normalized upper incomplete *Gamma* function of `a` and `x`

The ranges of `a` > `0` and `x` > `0`.

Plot example:

```
xMin    = 0.1
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.gamma_q_inv(x, 0.5)
y2      = special.gamma_q_inv(x, 1.0)
y3      = special.gamma_q_inv(x, 5.0)
y4      = special.gamma_q_inv(x, 10.0)

Utils.Graph4It("Inverse Gamma Q",           \
               "Plt073_inv_gamma_q",        \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4,           \
               "a = 0.5",                   \
               "a = 1.0",                   \
```

```
                "a = 5.0",                      \
                "a = 10.0",                     \
                xMin, xMax, 0.0, 20.0, 0, 0)
```

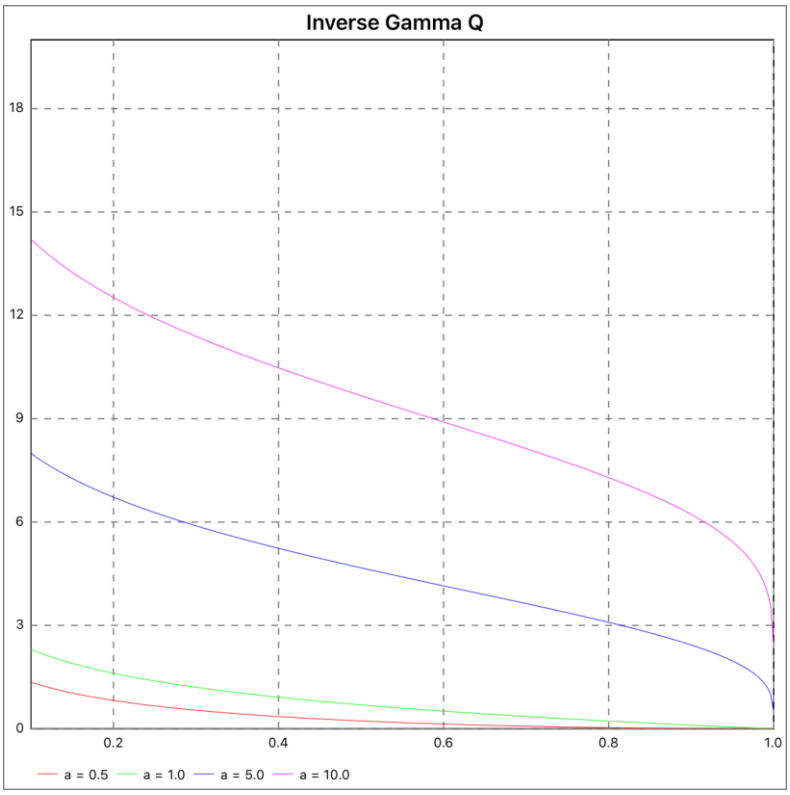

Figure 116 - Inverse Gamma Q Plot

6.24.15 Special.gamma_q_inv_a

`special.gamma_q_inv_a(x, a)` – Returns inverse normalized upper incomplete *Gamma* function of `a` and `x`

The ranges of `a` > `0` and `x` > `0`.

Plot example:

```
xMin    = 0.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.gamma_q_inv_a(x, 0.5)
y2      = special.gamma_q_inv_a(x, 1.0)
y3      = special.gamma_q_inv_a(x, 5.0)
y4      = special.gamma_q_inv_a(x, 10.0)

Utils.Graph4It("Inverse Gamma Q A",         \
               "Plt074_inv_gamma_q_a",      \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4,           \
               "a = 0.5",                   \
```

```
                "a = 1.0",                      \
                "a = 5.0",                      \
                "a = 10.0",                     \
                xMin, xMax, 0.0, 20.0, 0, 0)
```

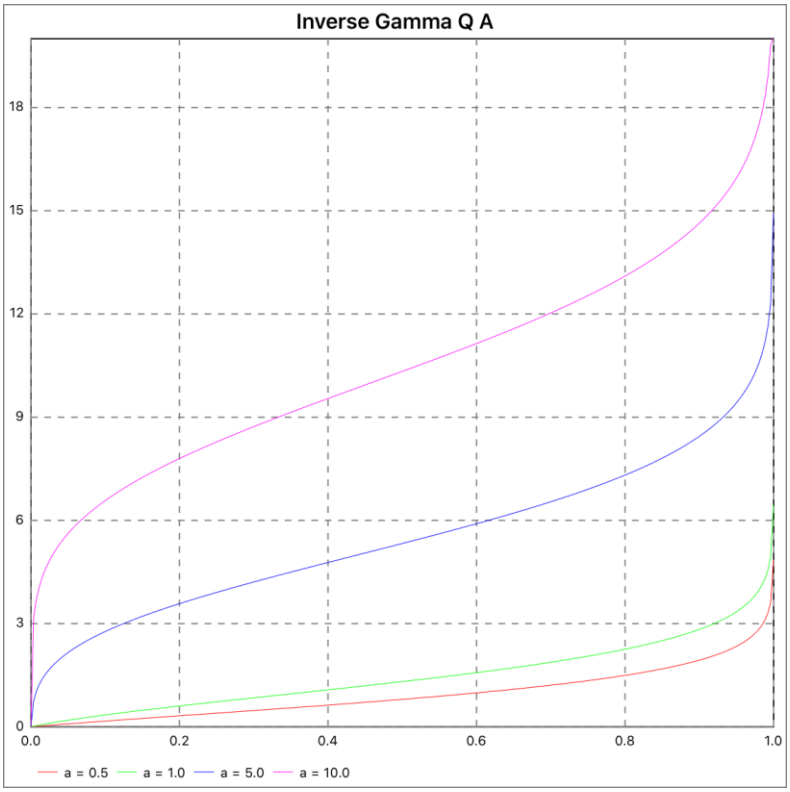

Figure 117 - Inverse Gamma Q A Plot

6.24.16 Special.beta

`special.beta(x, a)` – Returns *Beta* pivot `a` of `x`
`special.beta(x, a, b)`

The `a`, `b`, and `x` arguments are expressed as $\int t^{a-1}(1-t)^{b-1}dt$ over an integral from `0` to `x`. The ranges of `a`, `b` > `0`, and `0` ≤ `x` ≤ `1`.

⇒ LINK: https://en.wikipedia.org/wiki/Beta_function

Plot example:

```
xMin    = 0.0
xMax    = 5.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.beta(x, 0.5)
y2      = special.beta(x, 1.0)
y3      = special.beta(x, 5.0)
y4      = special.beta(x, 10.0)
```

```
Utils.Graph4It("Beta",                          \
               "Plt075_beta",                   \
               Plot.CHART.LINE,                 \
               x, y1, y2, y3, y4,               \
               "a = 0.5",                       \
               "a = 1.0",                       \
               "a = 5.0",                       \
               "a = 10.0",                      \
               xMin, xMax, 0.0, 10.0, 0, 0)
```

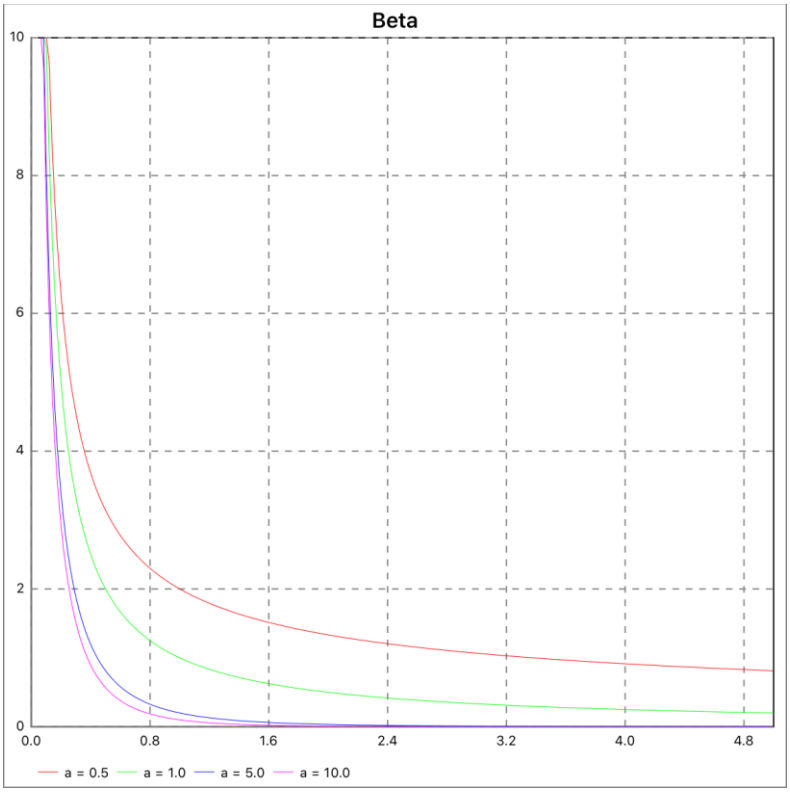

Figure 118 - Beta 1 Plot

Another plot example:

```
xMin    = 0.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.beta(x, 0.5, 1.0)
y2      = special.beta(x, 1.0, 1.5)
y3      = special.beta(x, 2.0, 2.0)
y4      = special.beta(x, 3.0, 2.5)

Utils.Graph4It("Beta",                          \
               "Plt076_beta",                   \
               Plot.CHART.LINE,                 \
               x, y1, y2, y3, y4,               \
```

```
                "a = 0.5, b = 1.0",           \
                "a = 1.0, b = 2.0",           \
                "a = 5.0, b = 5.0",           \
                "a = 10.0, b = 7.0",          \
                xMin, xMax, 0.0, 2.0, 0, 0)
```

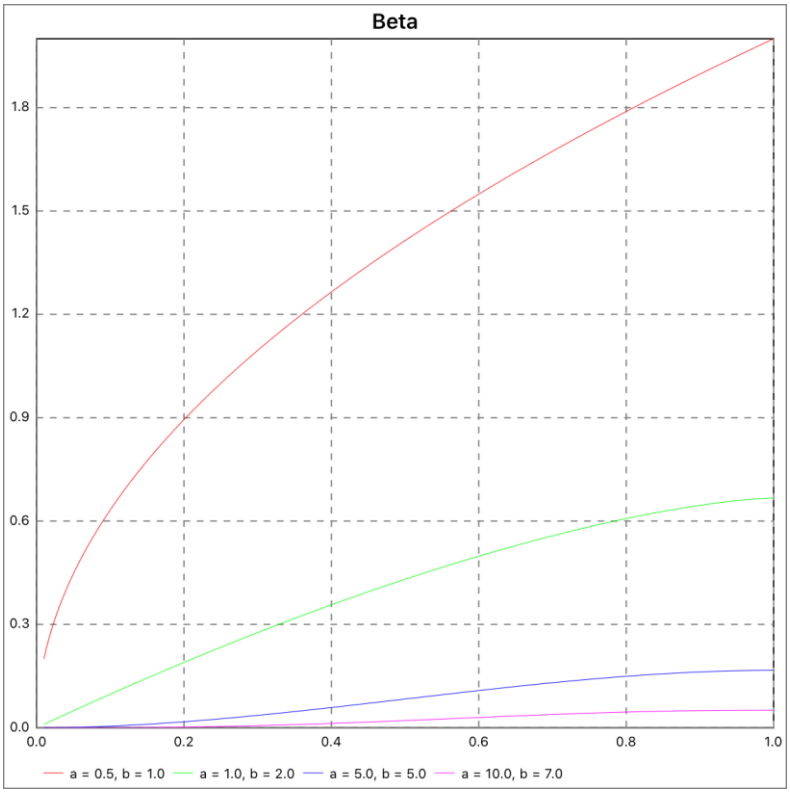

Figure 119 - Beta 2 Plot

6.24.17 Special.beta_c

`special.beta_c(x, a, b)` – Returns complement of *Beta* pivot `a` and `b` of `x`

The `a`, `b`, and `x` arguments are expressed as *beta(1-x, a)*. The ranges of `a` and `b` > `0`, and `0` ≤ `x` ≤ `1`.

Plot example:

```
xMin    = 0.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.beta_c(x, 9.0, 1.0)
y2      = special.beta_c(x, 7.0, 2.0)
y3      = special.beta_c(x, 5.0, 5.0)
y4      = special.beta_c(x, 2.0, 7.0)
y5      = special.beta_c(x, 1.0, 9.0)

Utils.Graph5It("Complement Beta",           \
               "Plt077_beta_c",             \
               Plot.CHART.LINE,             \
```

```
x, y1, y2, y3, y4, y5,          \
"a = 9, b = 1",                 \
"a = 7, b = 2",                 \
"a = 5, b = 5",                 \
"a = 2, b = 7",                 \
"a = 1, b = 9",                 \
xMin, xMax, 0.0, 0.15, 0, 0)
```

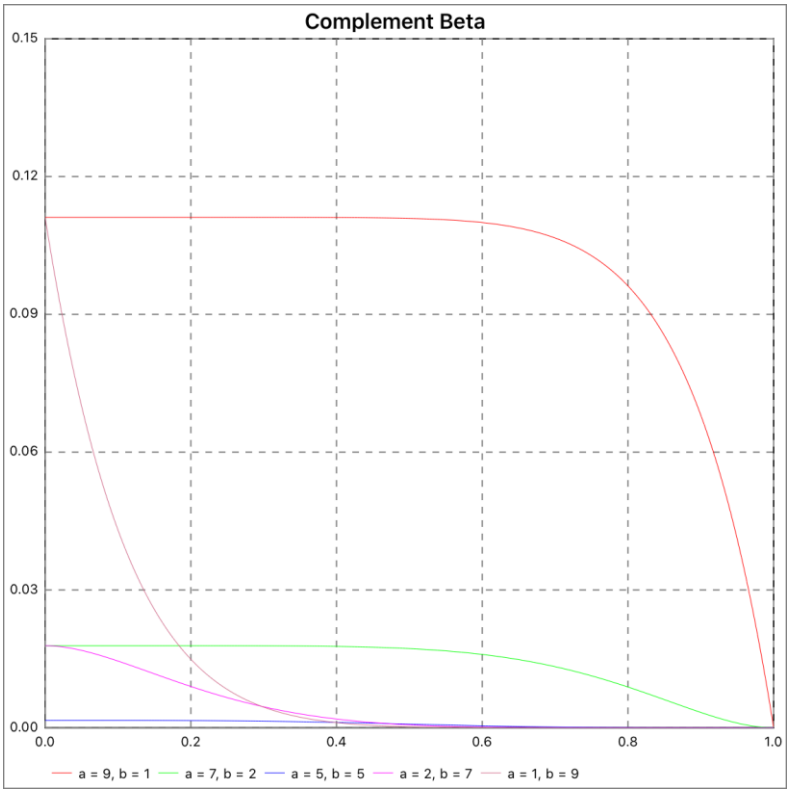

Figure 120 - Complement Beta Plot

6.24.18 Special.ibeta

`special.ibeta(x, a, b)` – Returns incomplete *Beta* pivot `a` and `b` of `x`

The `a`, `b`, and `x` arguments are expressed as $1/beta(x, a) \cdot \int t^{a-1}(1-t)^{b-1}dt$ over an integral from `0` to `x`. The ranges of `a` and `b` > `0`, and `0` ≤ `x` ≤ `1`.

Plot example:

```
xMin    = 0.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.ibeta(x, 9.0, 1.0)
y2      = special.ibeta(x, 7.0, 2.0)
y3      = special.ibeta(x, 5.0, 5.0)
y4      = special.ibeta(x, 2.0, 7.0)
y5      = special.ibeta(x, 1.0, 9.0)
```

```
Utils.Graph5It("Incomplete Beta",                \
               "Plt078_i_beta",                  \
               Plot.CHART.LINE,                  \
               x, y1, y2, y3, y4, y5,            \
               "a = 9, b = 1",                   \
               "a = 7, b = 2",                   \
               "a = 5, b = 5",                   \
               "a = 2, b = 7",                   \
               "a = 1, b = 9",                   \
               xMin, xMax, 0.0, 1.0, 0, 0)
```

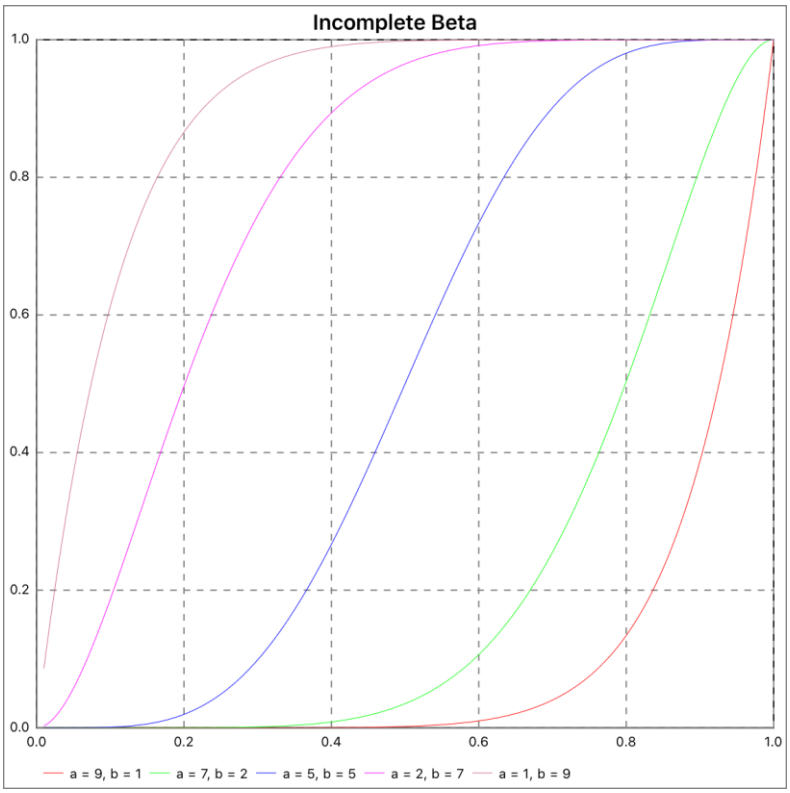

Figure 121 - I Beta A Plot

6.24.19 Special.ibeta_inv

`special.ibeta_inv(x, a, b)` – Returns inverse incomplete *Beta* pivot `a` and `b` of `x`

Plot example:

```
xMin    = 0.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.ibeta_inv(x, 9.0, 1.0)
y2      = special.ibeta_inv(x, 7.0, 2.0)
y3      = special.ibeta_inv(x, 5.0, 5.0)
y4      = special.ibeta_inv(x, 2.0, 7.0)
y5      = special.ibeta_inv(x, 1.0, 9.0)
```

```
Utils.Graph5It("Inverse Incomplete Beta",    \
               "Plt079_i_beta_inv",          \
               Plot.CHART.LINE,              \
               x, y1, y2, y3, y4, y5,        \
               "a = 9, b = 1",               \
               "a = 7, b = 2",               \
               "a = 5, b = 5",               \
               "a = 2, b = 7",               \
               "a = 1, b = 9",               \
               xMin, xMax, 0.0, 1.0, 0, 0)
```

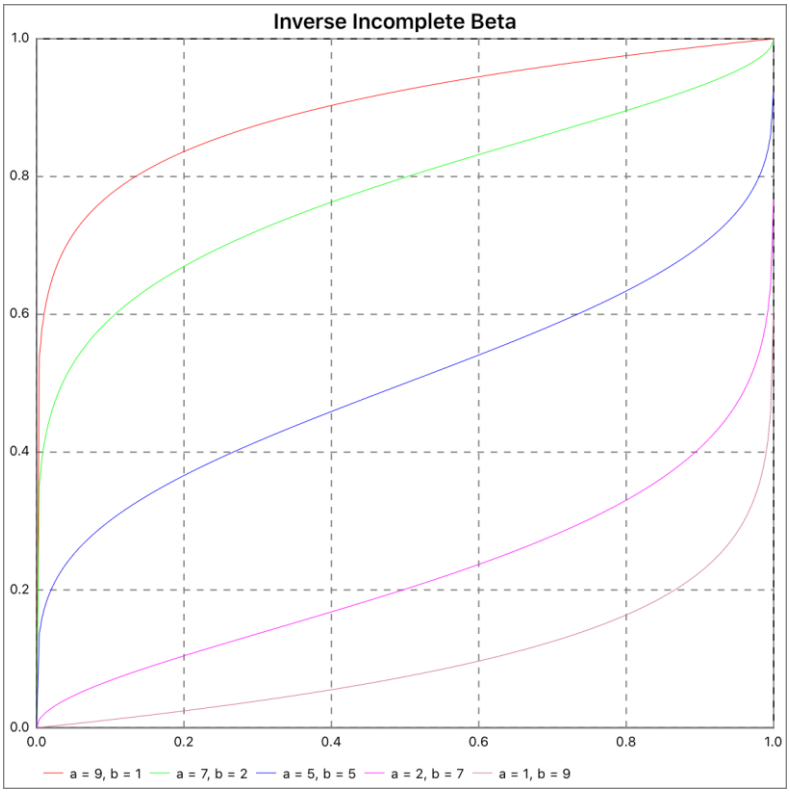

Figure 122 - Inverse Incomplete Beta Plot

6.24.20 Special.ibeta_inv_a

`special.ibeta_inv_a(x, a, p)` – Returns inverse incomplete *Beta* pivot `a` and `b` of `x`

Plot example:

```
xMin    = 0.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax - inc)

y1      = special.ibeta_inv_a(x, 9.0, 1.0)
y2      = special.ibeta_inv_a(x, 7.0, 2.0)
y3      = special.ibeta_inv_a(x, 5.0, 5.0)
```

```
y4      = special.ibeta_inv_a(x, 2.0, 7.0)
y5      = special.ibeta_inv_a(x, 1.0, 9.0)

Utils.Graph5It("Inverse Incomplete Beta A", \
               "Plt080_i_beta_inv_a",       \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "a = 9, b = 1",              \
               "a = 7, b = 2",              \
               "a = 5, b = 5",              \
               "a = 2, b = 7",              \
               "a = 1, b = 9",              \
               xMin, xMax, 0.0, 10.0, 0, 0)
```

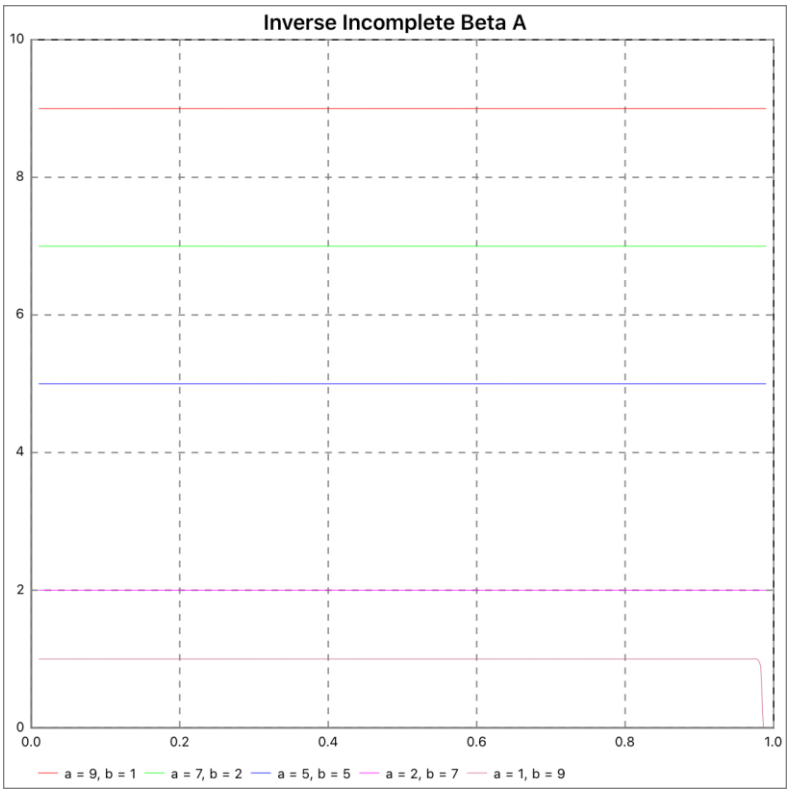

Figure 123 - Inverse Incomplete Beta A Plot

6.24.21 Special.ibeta_inv_b

`special.ibeta_inv_b(x, a, b)` – Returns inverse incomplete *Beta* pivot `a` and `b` of `x`

Plot example:

```
xMin    = 0.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.ibeta_inv_b(x, 9.0, 1.0)
y2      = special.ibeta_inv_b(x, 7.0, 2.0)
y3      = special.ibeta_inv_b(x, 5.0, 5.0)
```

```
Utils.Graph3It("Inverse Incomplete Beta B", \
               "Plt081_i_beta_inv_b",       \
               Plot.CHART.LINE,             \
               x, y1, y2, y3,               \
               "a = 9, b = 1",              \
               "a = 7, b = 2",              \
               "a = 5, b = 5",              \
               xMin, xMax, 0.0, 6.0, 0, 0)
```

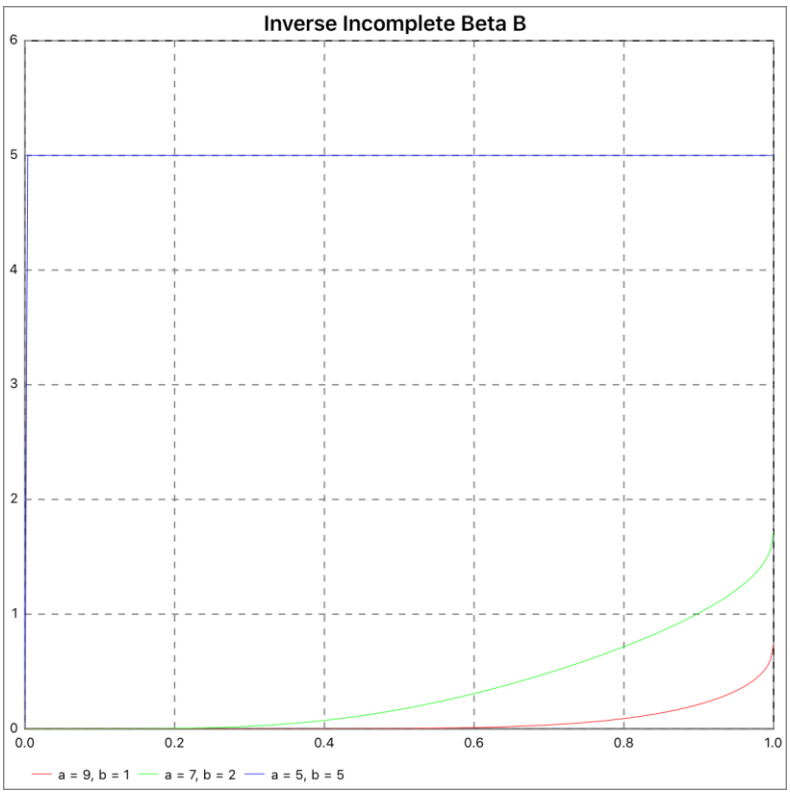

Figure 124 - Inverse Incomplete Beta B Plot

6.24.22 Special.ibeta_c

`special.ibeta_c(x, a, b)` – Returns complement of incomplete *Beta* pivot `a` and `b` of `x`

The `a`, `b`, and `x` arguments are expressed as *ibeta(1-x, a, b)*. The ranges of `a` and `b` > `0`, and `0` ≤ `x` ≤ `1`.

Plot example:

```
xMin    = 0.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.ibeta_c(x, 9.0, 1.0)
y2      = special.ibeta_c(x, 7.0, 2.0)
y3      = special.ibeta_c(x, 5.0, 5.0)
y4      = special.ibeta_c(x, 2.0, 7.0)
y5      = special.ibeta_c(x, 1.0, 9.0)
```

```
Utils.Graph5It("Complement Incomplete Beta",\
               "Plt082_i_beta_c",           \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "a = 9, b = 1",              \
               "a = 7, b = 2",              \
               "a = 5, b = 5",              \
               "a = 2, b = 7",              \
               "a = 1, b = 9",              \
               xMin, xMax, 0.0, 1.0, 0, 0)
```

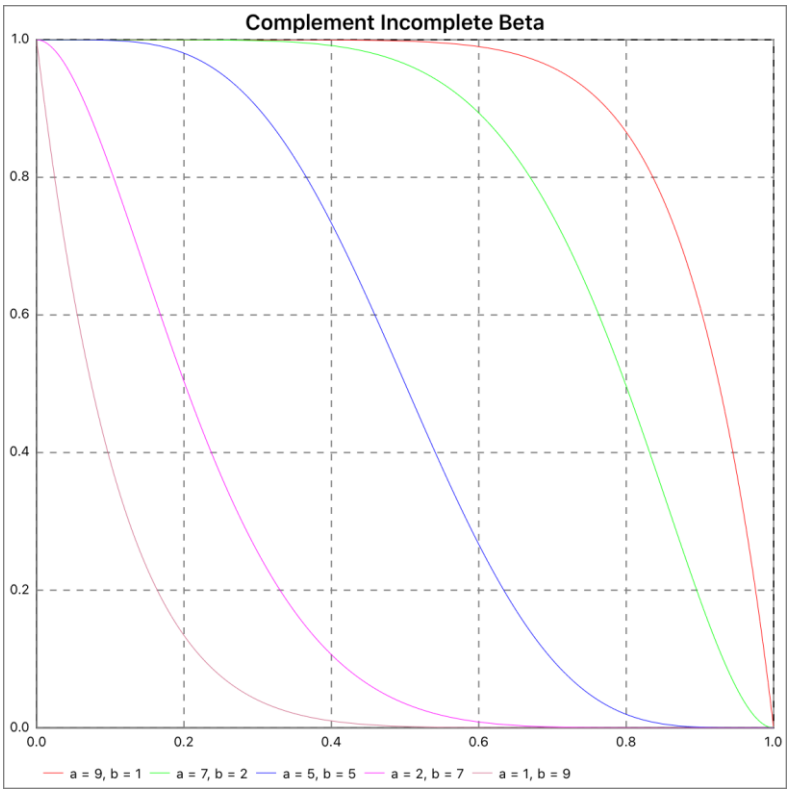

Figure 125 - Complement Incomplete Beta Plot

6.24.23 Special.ibeta_c_inv

`special.ibeta_c_inv(x, a, b)` – Returns inverse complement of incomplete *Beta* pivot `a` and `b` of `x`

Plot example:

```
xMin    = 0.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.ibeta_c_inv(x, 9.0, 1.0)
y2      = special.ibeta_c_inv(x, 7.0, 2.0)
y3      = special.ibeta_c_inv(x, 5.0, 5.0)
y4      = special.ibeta_c_inv(x, 2.0, 7.0)
y5      = special.ibeta_c_inv(x, 1.0, 9.0)
```

```
Utils.Graph5It("Inverse Complement Incomplete Beta", \
               "Plt083_i_beta_c_inv",          \
               Plot.CHART.LINE,                \
               x, y1, y2, y3, y4, y5,          \
               "a = 9, b = 1",                 \
               "a = 7, b = 2",                 \
               "a = 5, b = 5",                 \
               "a = 2, b = 7",                 \
               "a = 1, b = 9",                 \
               xMin, xMax, 0.0, 1.0, 0, 0)
```

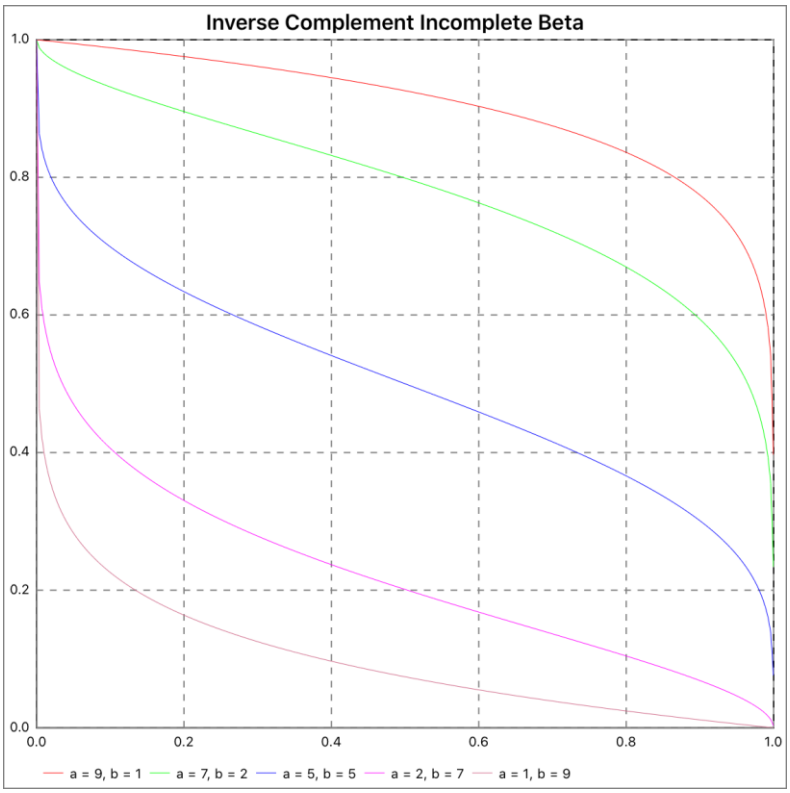

Figure 126 - Inverse Complement Incomplete Beta Plot

6.24.24 Special.ibeta_c_inv_a

`special.ibeta_c_inv_a(x, a, b)` – Returns inverse complement of incomplete *Beta* pivot `a` and `b` of `x`

Plot example:

```
xMin    = 0.0
xMax    = 1.0
inc     = 0.1

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.ibeta_c_inv_a(x, 9.0, 1.0)
y2      = special.ibeta_c_inv_a(x, 7.0, 2.0)
y3      = special.ibeta_c_inv_a(x, 5.0, 5.0)
```

```
y4      = special.ibeta_c_inv_a(x, 2.0, 7.0)
y5      = special.ibeta_c_inv_a(x, 1.0, 9.0)

Utils.Graph5It("Inverse Complement Incomplete Beta A", \
               "Plt084_i_beta_c_inv_a",     \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "a = 9, b = 1",              \
               "a = 7, b = 2",              \
               "a = 5, b = 5",              \
               "a = 2, b = 7",              \
               "a = 1, b = 9",              \
               xMin, xMax, 0.0, 10.0, 0, 0)
```

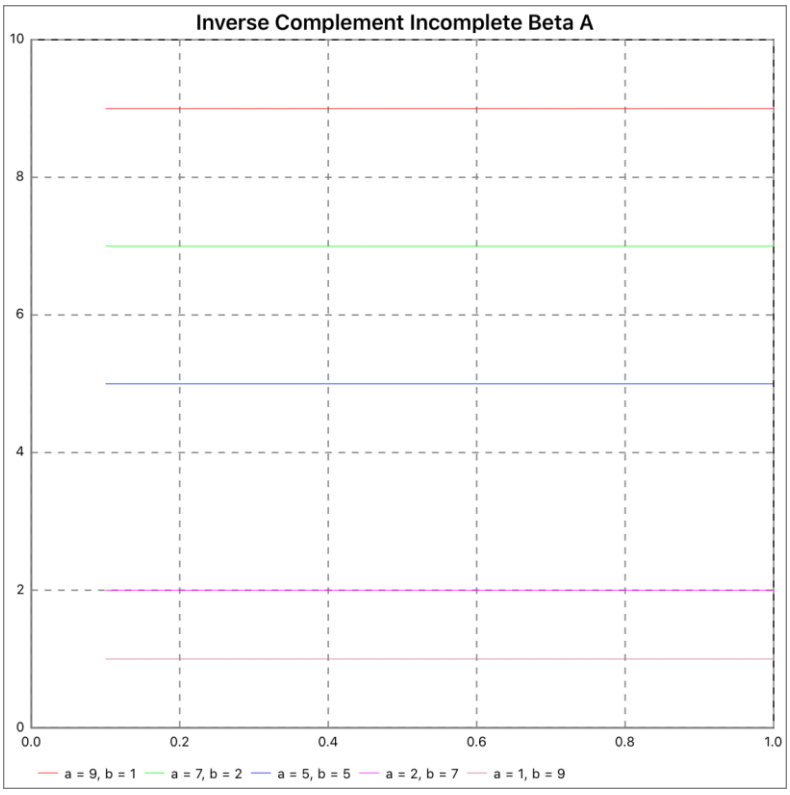

Figure 127 - Inverse Complement Incomplete Beta A Plot

6.24.25 Special.ibeta_c_inv_b

`special.ibeta_c_inv_b(x, a, b)` – Returns inverse complement of incomplete *Beta* pivot `a` and `b` of `x`

Plot example:

```
xMin    = 0.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.ibeta_c_inv_b(x, 9.0, 1.0)
y2      = special.ibeta_c_inv_b(x, 7.0, 2.0)
```

```
y3      = special.ibeta_c_inv_b(x, 5.0, 5.0)
y4      = special.ibeta_c_inv_b(x, 2.0, 7.0)
y5      = special.ibeta_c_inv_b(x, 1.0, 9.0)

Utils.Graph5It("Inverse Complement Incomplete Beta B", \
               "Plt085_i_beta_c_inv_b",     \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "a = 9, b = 1",              \
               "a = 7, b = 2",              \
               "a = 5, b = 5",              \
               "a = 2, b = 7",              \
               "a = 1, b = 9",               \
               xMin, xMax, 0.0, 50.0, 0, 0)
```

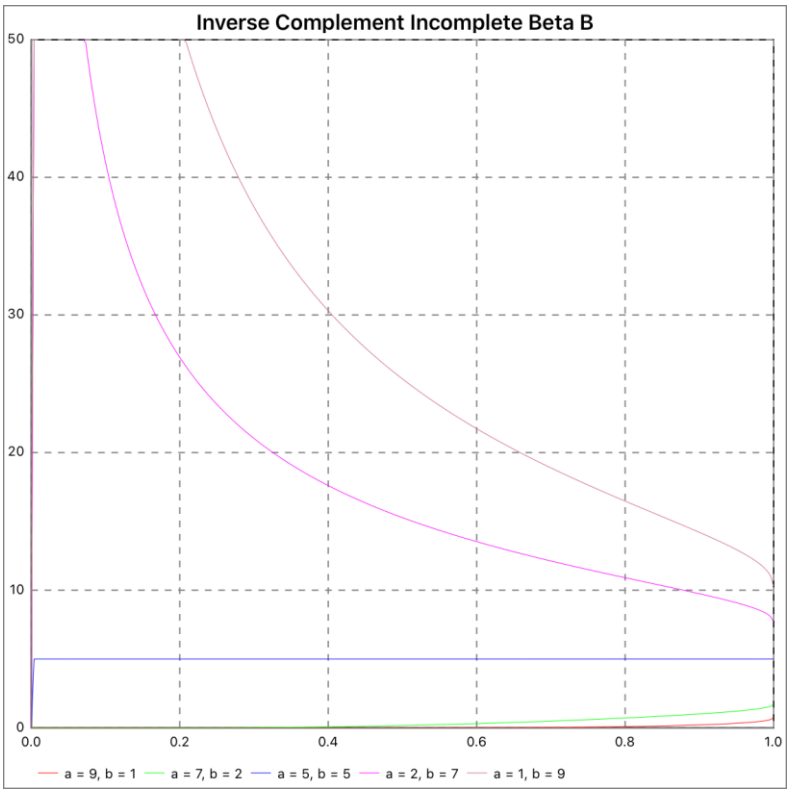

Figure 128 - Inverse Complement Incomplete Beta B Plot

6.24.26 Special.ibeta_der

`special.ibeta_der(x, a, b)` – Returns partial derivative incomplete *Beta* pivot `a` and `b` of `x`

The `x`, `a`, and `b` arguments are expressed $((1-x)^{b-1}x^{a-1}) / beta(x, a)$.

Plot example:

```
xMin    = 0.0
xMax    = 1.0
inc     = 0.01
```

```
x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.ibeta_der(x, 9.0, 1.0)
y2      = special.ibeta_der(x, 7.0, 2.0)
y3      = special.ibeta_der(x, 5.0, 5.0)
y4      = special.ibeta_der(x, 2.0, 7.0)
y5      = special.ibeta_der(x, 1.0, 9.0)

Utils.Graph5It("Incomplete Beta Derivative",\
               "Plt086_i_beta_der",          \
               Plot.CHART.LINE,              \
               x, y1, y2, y3, y4, y5,        \
               "a = 9, b = 1",               \
               "a = 7, b = 2",               \
               "a = 5, b = 5",               \
               "a = 2, b = 7",               \
               "a = 1, b = 9",               \
               xMin, xMax, 0.0, 4.0, 0, 0)
```

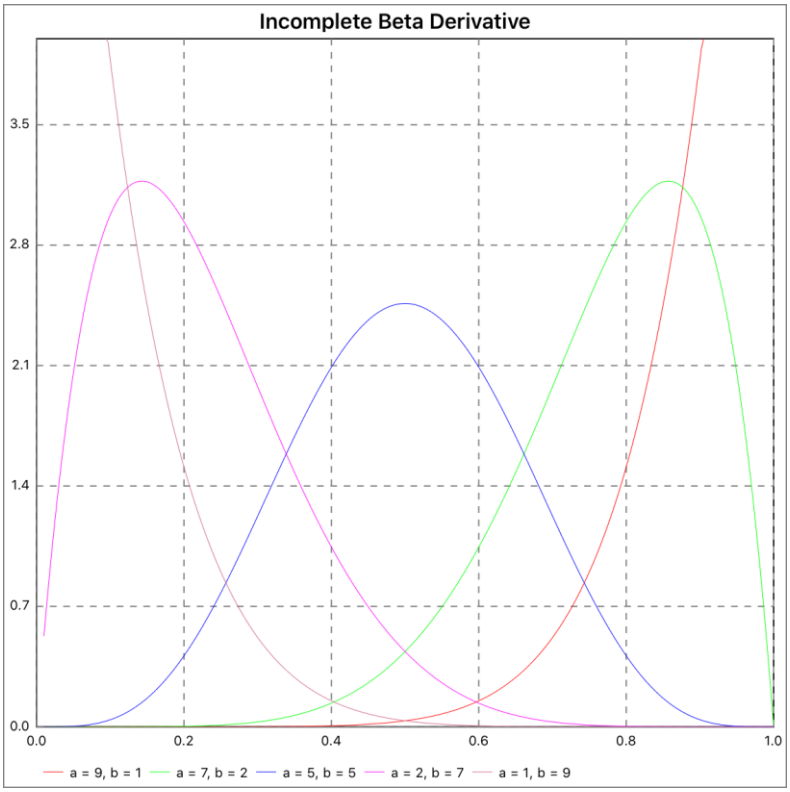

Figure 129 - Incomplete Beta Derivative Plot

6.24.27 Special.erf

`special.erf(x)` – Returns *Error Function* of `x`

The `x` argument is expressed as $2 / \sqrt{\pi} \cdot \int e^{-t^{\wedge}2} dt$ over an integral from `0` to `x`.

⇒ LINK: https://en.wikipedia.org/wiki/Error_function

Plot example:

```
xMin    = -3.0
xMax    = 3.0

x       = Utils.PlotX(xMin, xMax)
y       = Special.erf(x)

Utils.PlotIt("Error Function",          \
             "Plt087_erf",              \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, -1.0, 1.0, 0, 0)
```

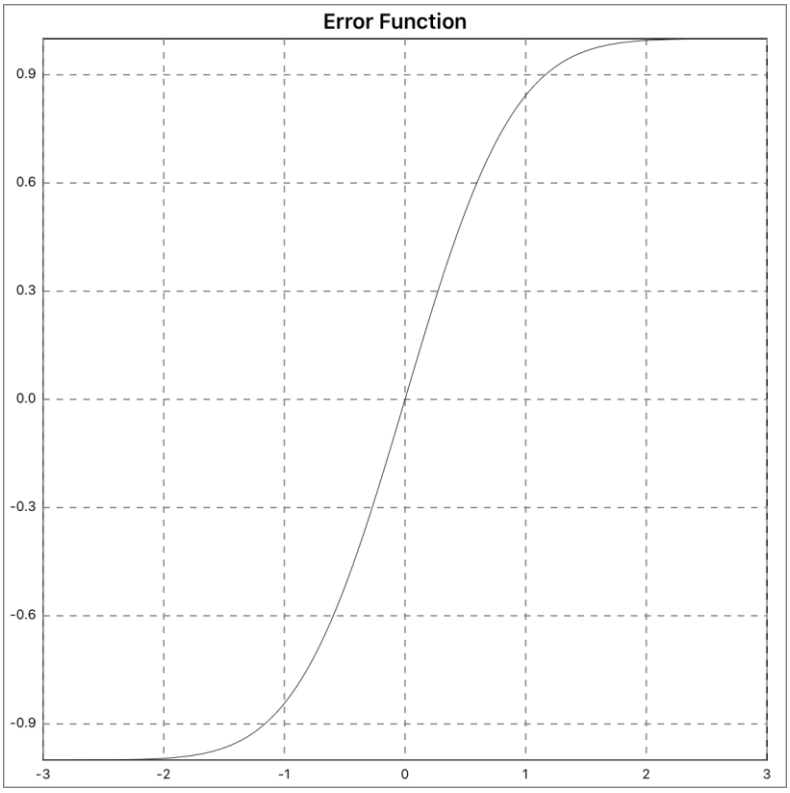

Figure 130 - Error Function Plot

6.24.28 Special.erfc

`special.erfc(x)` – Returns complement *Error Function* of `x`

The `x` argument is expressed as $1-2 / \sqrt{\pi} \cdot \int e^{-t^2} dt$ over an integral from `0` to `x`.

Plot example:

```
xMin    = -3.0
xMax    = 3.0

x       = Utils.PlotX(xMin, xMax)
y       = Special.erfc(x)
```

```
Utils.PlotIt("Error Complement Function", \
             "Plt088_erfc",               \
             Plot.CHART.LINE,             \
             x, y,                        \
             xMin, xMax, 0.0, 2.0, 0, 0)
```

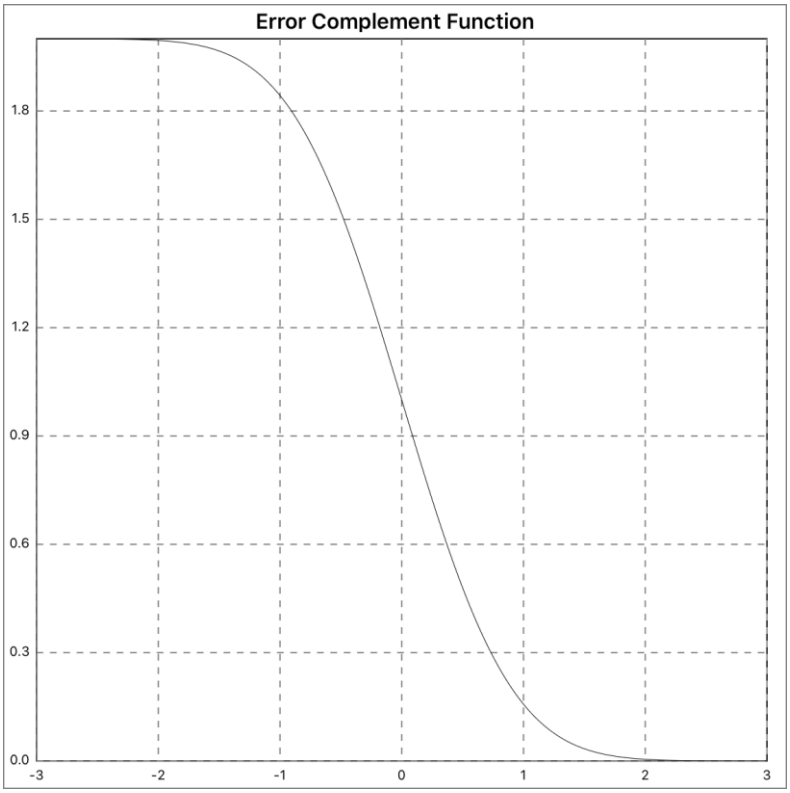

Figure 131 - Error Complement Function Plot

6.24.29 Special.erf_inv

`special.erf_inv(x)` – Returns inverse *Error Function* of `x`

Plot example:

```
xMin    = -1.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y       = Special.erf_inv(x)

Utils.PlotIt("Error Inverse Function",  \
             "Plt090_erf_inv",          \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, -3.0, 2.0, 0, 0)
```

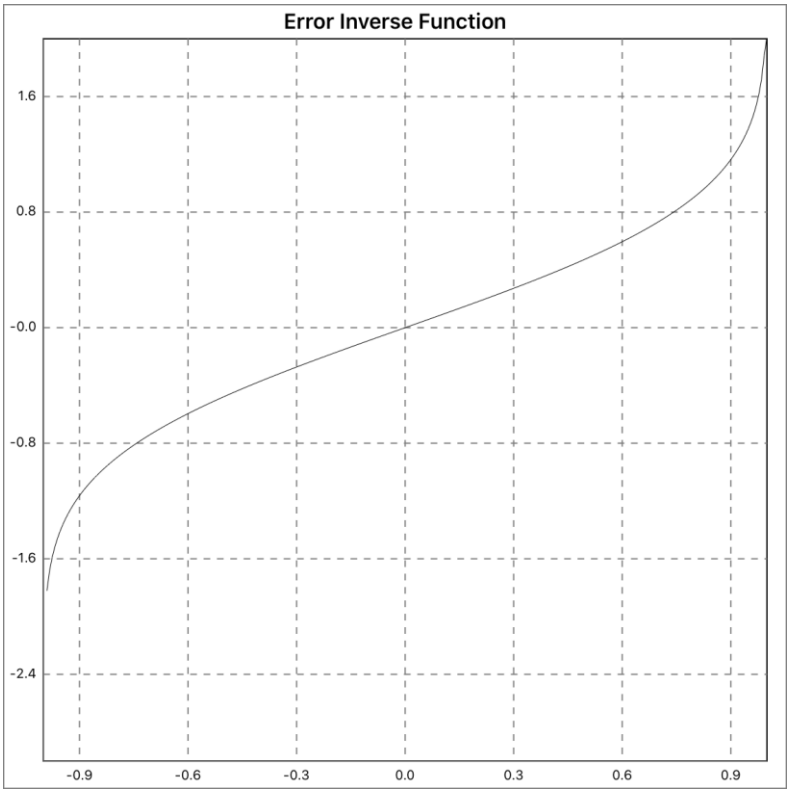

Figure 132 - Error Inverse Function Plot

6.24.30 Special.erfc_inv

`special.erfc_inv(x)` – Returns inverse complement *Error Function* of `x`

Plot example:

```
xMin    = 0.0
xMax    = 2.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y       = Special.erfc_inv(x)

Utils.PlotIt("Error Complement Inverse Func", \
             "Plt091_erfc_inv",               \
             Plot.CHART.LINE,                 \
             x, y,                            \
             xMin, xMax, -3.0, 3.0, 0, 0)
```

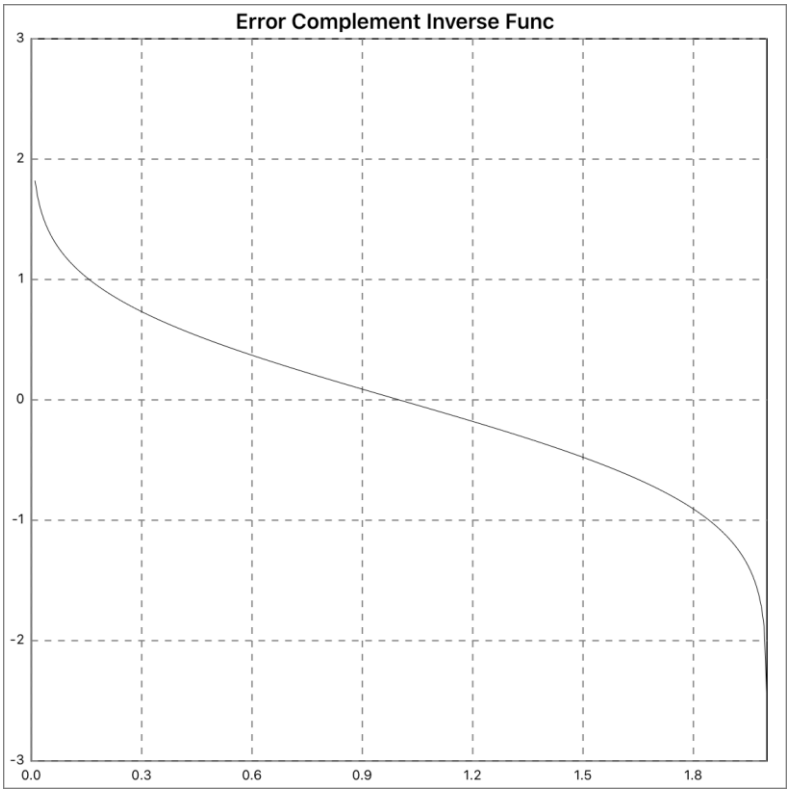

Figure 133 - Error Complement Inverse Function Plot

6.24.31 Special.legendre_p

`special.legendre_p1(x, n)` – Returns *Legendre Polynomial* of the first kind pivot `n` of `x`
`special.legendre_p2(x, n, m)`

The `n` and `x` arguments are expressed as $ldx^l / 2^l l! dx^l \cdot (x^2\text{-}1)^l$. The optional `m` argument is expressed as $(-1)(1 - x^2)^{m/2} \cdot d^m P_l(x)/dx^m$. The range of `n` and `m` is `> 0` and `x` is $-1 \leq x \leq -1$.

⇒ LINK: https://en.wikipedia.org/wiki/Legendre_polynomials

Plot example:

```
xMin    = -1.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.legendre_p(x, 1)
y2      = special.legendre_p(x, 2)
y3      = special.legendre_p(x, 3)
y4      = special.legendre_p(x, 4)
y5      = special.legendre_p(x, 5)

Utils.Graph5It("Legendre Polynomials",      \
               "Plt092_legendre_poly",      \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "n = 1",                     \
```

```
                "n = 2",                        \
                "n = 3",                        \
                "n = 4",                        \
                "n = 5",                        \
                xMin, xMax, -1.0, 1.0, 0, 0)
```

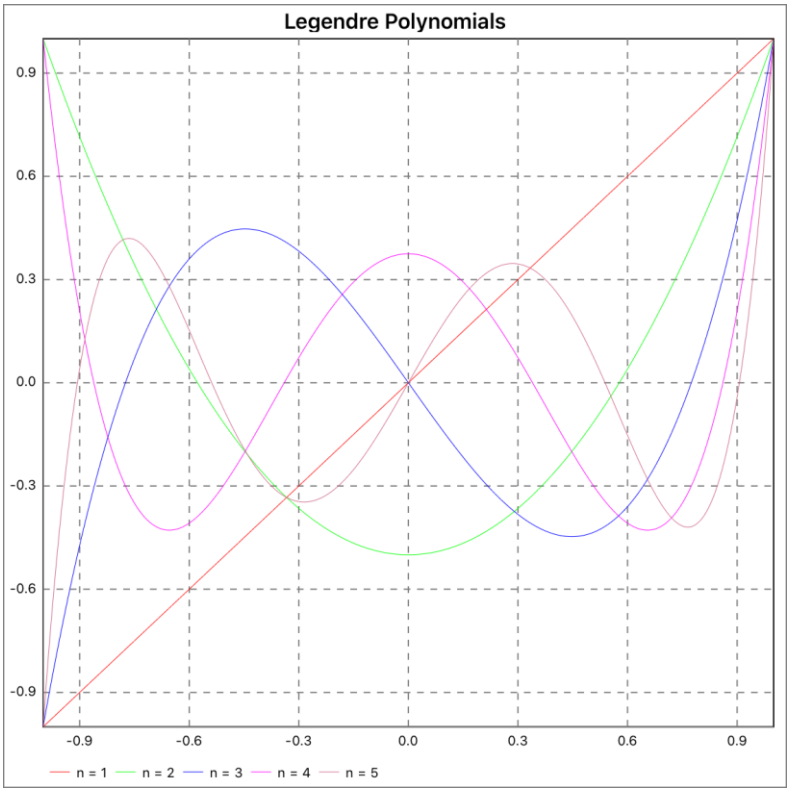

Figure 134 - Legendre Polynomials 1 Plot

Another plot lot example:

```
xMin    = -1.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.legendre_p(x, 1, 2)
y2      = special.legendre_p(x, 2, 2)
y3      = special.legendre_p(x, 3, 2)
y4      = special.legendre_p(x, 4, 2)
y5      = special.legendre_p(x, 5, 2)

Utils.Graph5It("Legendre Polynomials",      \
               "Plt093_legendre_poly",      \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "n = 1, m 2",                \
               "n = 2, m 2",                \
               "n = 3, m 2",                \
               "n = 4, m 2",                \
               "n = 5, m 2",                \
```

```
                xMin, xMax, -15.0, 15.0, 0, 0)
```

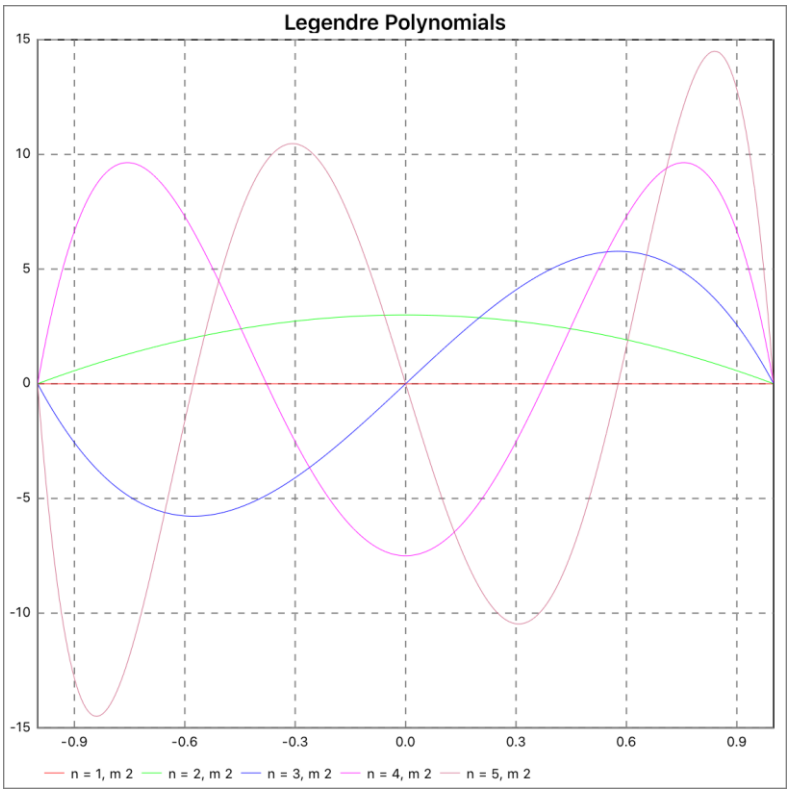

Figure 135 - Legendre Polynomials 2 Plot

6.24.32 Special.legendre_p_pri

`special.legendre_p_pri(x, n)` – Returns derivative *Legendre Polynomial* order `n` of `x`

Plot example:

```
x       = Utils.PlotX(xMin, xMax)
y1      = special.legendre_p_pri(x, 1)
y2      = special.legendre_p_pri(x, 2)
y3      = special.legendre_p_pri(x, 3)
y4      = special.legendre_p_pri(x, 4)
y5      = special.legendre_p_pri(x, 5)

Utils,Graph5It("Legendre Polynomials Prime",\
               "Plt094_legendre_poly_pri",  \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "n = 1",                     \
               "n = 2",                     \
               "n = 3",                     \
               "n = 4",                     \
               "n = 5",                     \
               xMin, xMax, -10.0, 15.0, 0, 0)
```

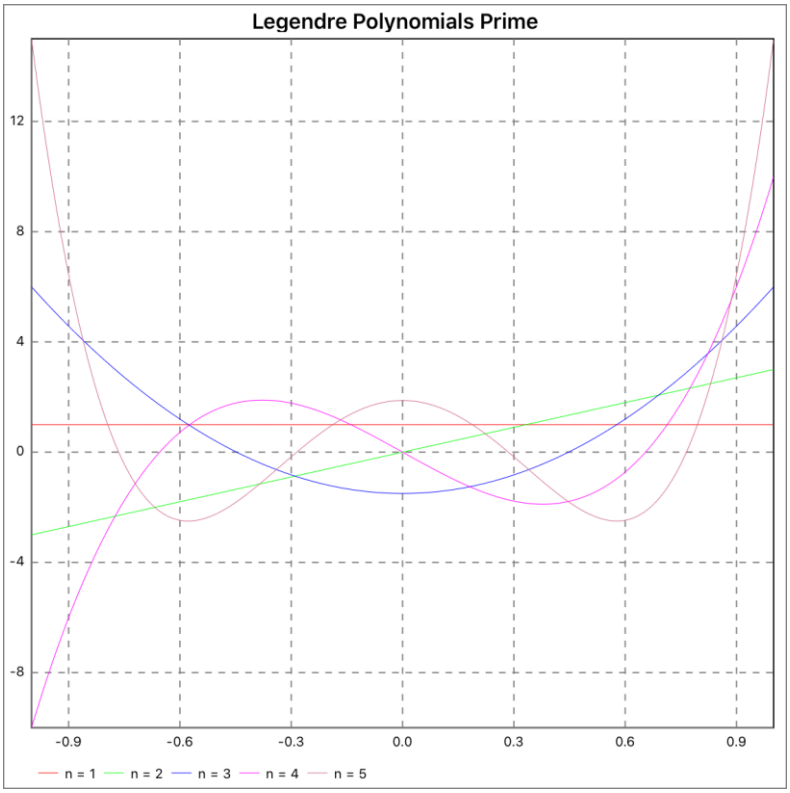

Figure 136 - Legendre Polynomials Prime Plot

6.24.33 Special.legendre_q

`special.legendre_q(x, l)` – Returns *Legendre Polynomial* of the second kind pivot `l` of `x`

The x and `l` arguments are expressed as $(2l + 1)xP_l(x) - lP_{l-1}(x) / (l + 1)$.

Plot example:

```
xMin    = -1.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = Special.legendre_q(x, 1)
y2      = Special.legendre_q(x, 2)
y3      = Special.legendre_q(x, 3)
y4      = Special.legendre_q(x, 4)
y5      = Special.legendre_q(x, 4)

Utils.Graph5It("Legendre Polynomials 2nd Kind", \
               "Plt095_legendre_q",             \
               Plot.CHART.LINE,                 \
               x, y1, y2, y3, y4, y5,           \
               "I = 1",                         \
               "I = 2",                         \
               "I = 3",                         \
               "I = 4",                         \
```

```
                "I = 5",                                \
                xMin, xMax, -1.0, 1.0, 0, 0)
```

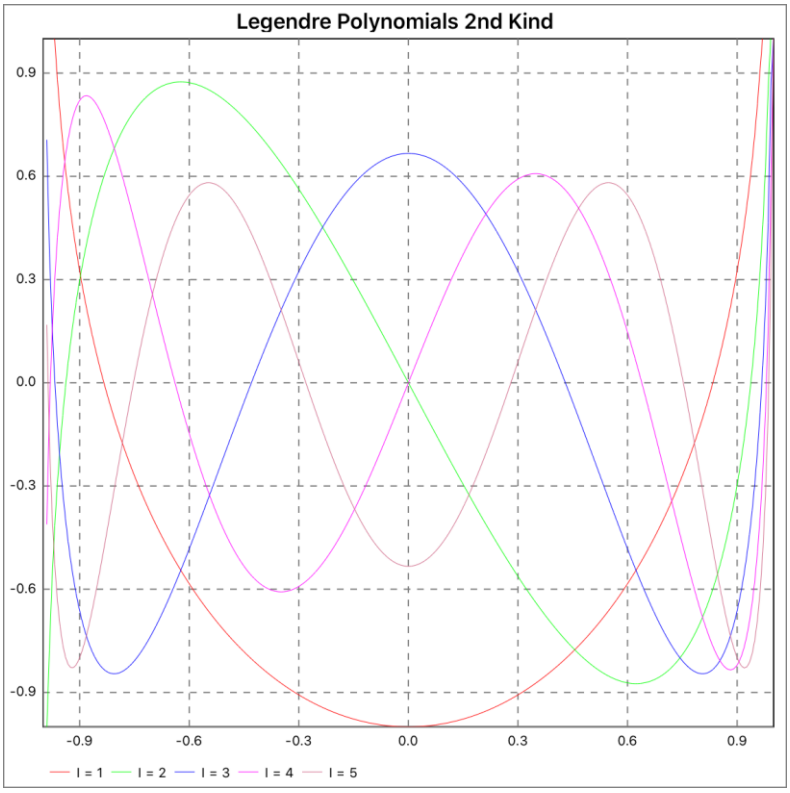

Figure 137 - Legendre Polynomials 2nd Kind Plot

6.24.34 Special.laguerre

`special.laguerre(x, n)` – Returns *Laguerre Polynomial* of order `n` of `x`
`special.laguerre_2(x, n, m)`

The `x` and `n` arguments are expressed as $e^x d^n / n! dx^n \cdot (x^n e^{-x})$. The optional `m` argument is expressed as $(-1)^m \cdot d^m/dx^m \cdot laguerre\_1(x, n + m)$

⇒ LINK: https://en.wikipedia.org/wiki/Laguerre_polynomials

Plot example:

```
xMin    = -5.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.laguerre(x, 0)
y2      = Special.laguerre(x, 1)
y3      = Special.laguerre(x, 2)
y4      = Special.laguerre(x, 3)
y5      = Special.laguerre(x, 4)

Utils.Graph5It("Laguerre Polynomials",    \
               "Plt096_laguerre_poly",    \
```

```
                Plot.CHART.LINE,            \
                x, y1, y2, y3, y4, y5,      \
                "n = 0",                    \
                "n = 1",                    \
                "n = 2",                    \
                "n = 3",                    \
                "n = 4",                    \
                xMin, xMax, -10.0, 20.0, 0, 0)
```

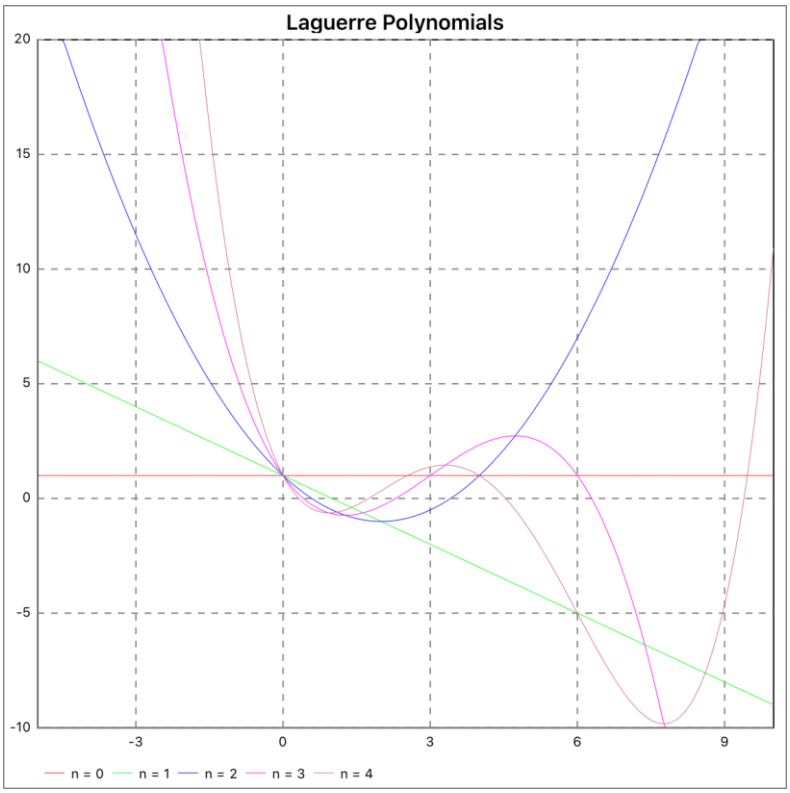

Figure 138 - Laguerre Polynomials 1 Plot

Another plot example:

```
xMin    = -5.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.laguerre(x, 0, 2)
y2      = special.laguerre(x, 1, 2)
y3      = special.laguerre(x, 2, 2)
y4      = special.laguerre(x, 3, 2)
y5      = special.laguerre(4, 2, x)

Utils.Graph5It("Laguerre Polynomials",      \
               "Plt097_laguerre_poly",      \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "n = 0, m = 2",              \
               "n = 1, m = 2",              \
```

```
                "n = 2, m = 2",                 \
                "n = 3, m = 2",                 \
                "n = 4, m = 2",                 \
                xMin, xMax, -15.0, 20.0, 0, 0)
```

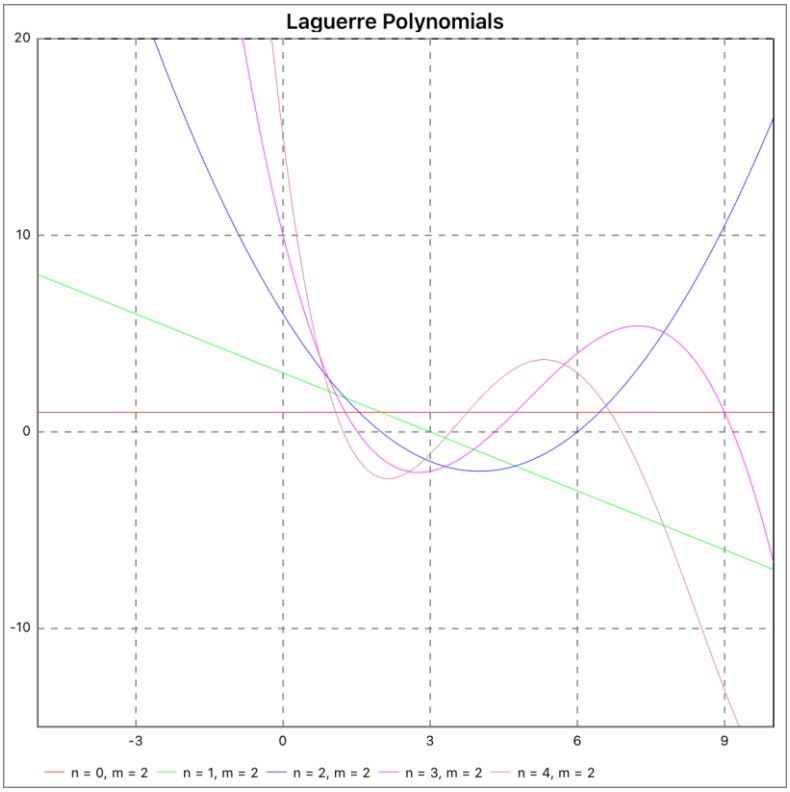

Figure 139 - Laguerre Polynomials 2 Plot

6.24.35 Special.hermite

`special.hermite(x, n)` – Returns *Hermite Polynomial* of order `n` of `x`

The `n` and `x` arguments are expressed as $(-1)^n e^{x\wedge 2} \cdot d^2 / dx^3 \cdot e^{-x\wedge 2}$.

⇒ LINK: https://en.wikipedia.org/wiki/Hermite_polynomials

Plot example:

```
xMin    = -2.0
xMax    = 2.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.hermite(x, 0)
y2      = Special.hermite(x, 1)
y3      = Special.hermite(x, 2)
y4      = Special.hermite(x, 3)
y5      = Special.hermite(x, 4)

Utils.Graph5It("Hermite Polynomials",    \
               "Plt098_hermite_poly",    \
```

```
Plot.CHART.LINE,          \
x, y1, y2, y3, y4, y5,    \
"n = 0",                  \
"n = 1",                  \
"n = 2",                  \
"n = 3",                  \
"n = 4",                  \
xMin, xMax, -20.0, 20.0, 0, 0)
```

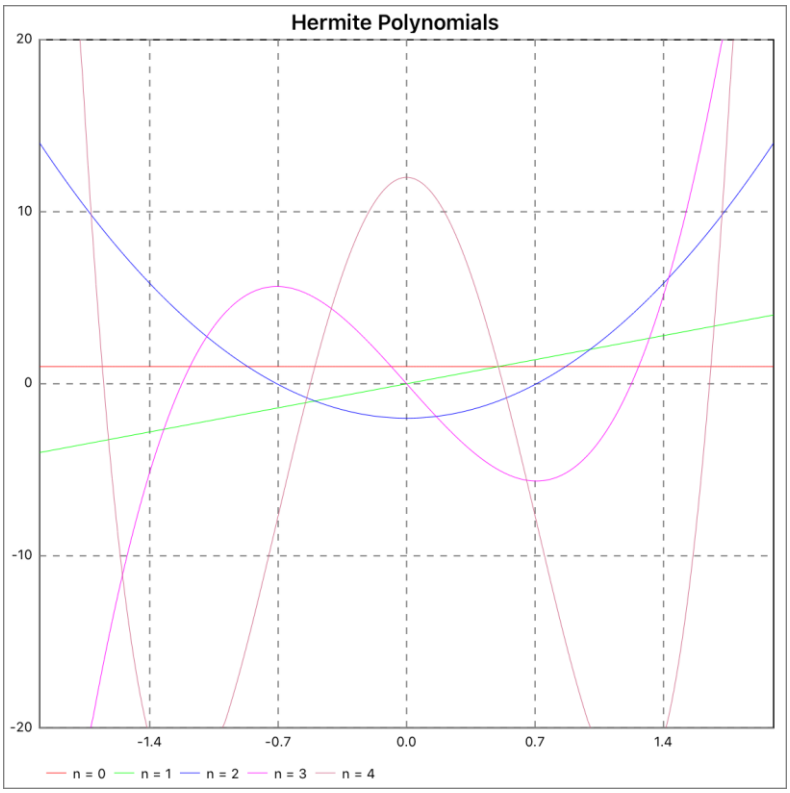

Figure 140 - Hermite Polynomials Plot

6.24.36 Special.bessel_i

`special.bessel_i(x, n)` – Returns modified *Bessel* of the first kind order `n` of `x`

The `n` and `x` arguments are expressed as $(½ \cdot x)^n \sum(-¼ \cdot x^2)^k / k! \Gamma(n+k+1)$ summation k from 0 to ∞.

⇒ LINK: https://en.wikipedia.org/wiki/Bessel_function

Plot example:

```
xMin    = -20.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.bessel_i(x, 0)
y2      = Special.bessel_i(x, 2)
y3      = Special.bessel_i(x, 5)
y4      = Special.bessel_i(x, 7)
```

```
y5      = Special.bessel_i(x, 10)

Utils.Graph5It("Bessel I",                \
               "Plt099_bessel_i",         \
               Plot.CHART.LINE,           \
               x, y1, y2, y3, y4, y5,     \
               "v = 0",                   \
               "v = 2",                   \
               "v = 5",                   \
               "v = 7",                   \
               "v = 10",                  \
               xMin, xMax, -20.0, 20.0, 0, 0)
```

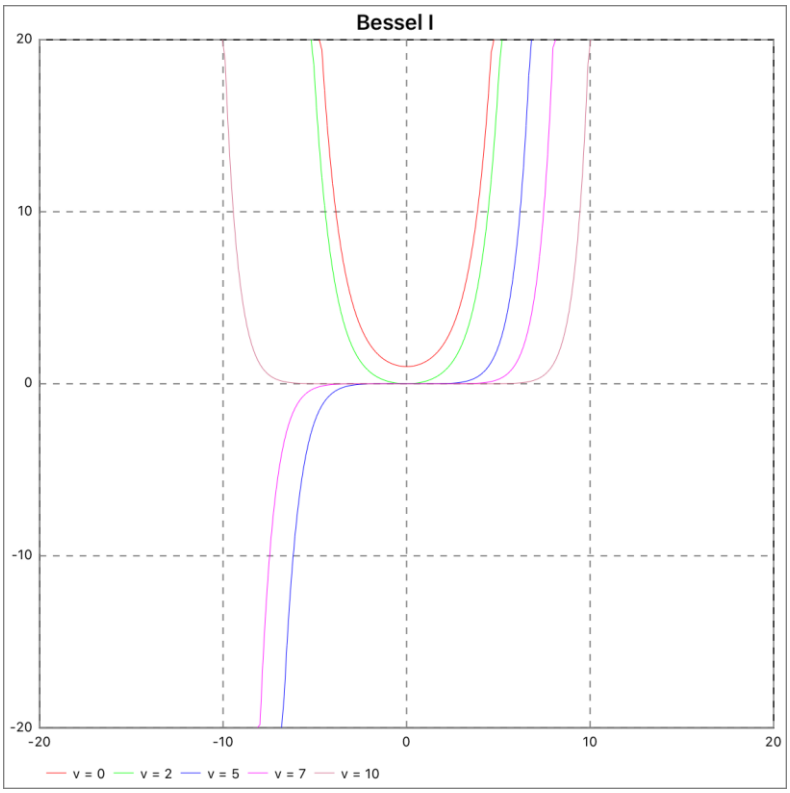

Figure 141 - Bessel I Plot

6.24.37 Special.bessel_i_pri

`special.bessel_i_pri(x, n)` – Returns modified *Bessel* derivative of the first kind order `n` of `x`

The `n` and `x` arguments are expressed as *(bessel_i(x, n–1) + bessel_i(x, n+1)) / 2*.

Plot example:

```
xMin    = -5.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.bessel_i_pri(x, 0)
y2      = special.bessel_i_pri(x, 2)
```

```
y3      = special.bessel_i_pri(x, 5)
y4      = special.bessel_i_pri(x, 7)
y5      = special.bessel_i_pri(x, 10)

Utils.Graph5It("Bessel I Prime",              \
               "Plt100_bessel_i_pri",         \
               Plot.CHART.LINE,               \
               x, y1, y2, y3, y4, y5,         \
               "v = 0",                       \
               "v = 2",                       \
               "v = 5",                       \
               "v = 7",                       \
               "v = 10",                      \
               xMin, xMax, -20.0, 20.0, 0, 0)
```

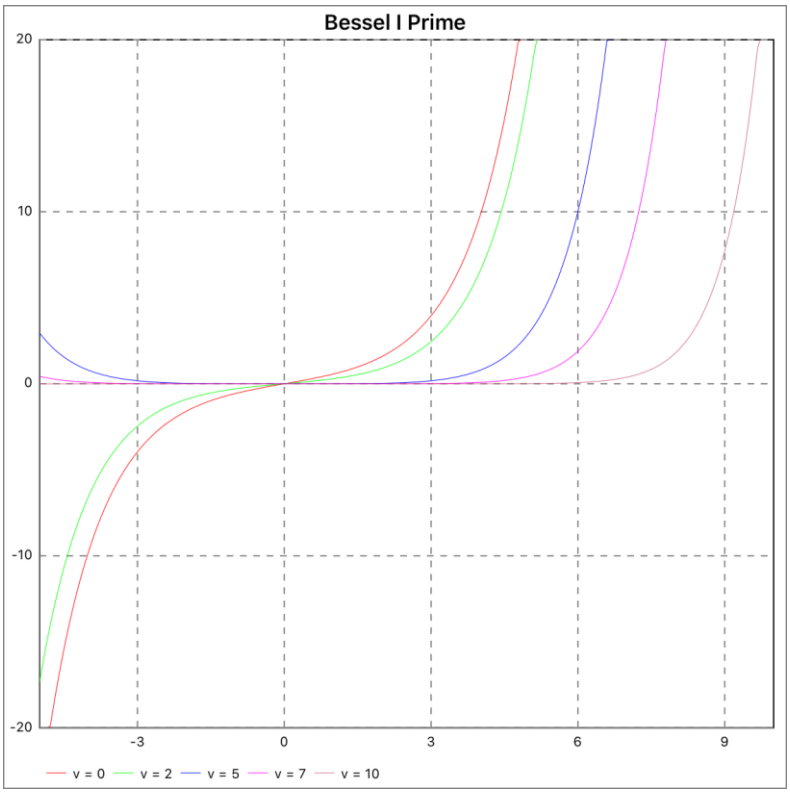

Figure 142 - Bessel I Prime Plot

6.24.38 Special.bessel_j

`special.bessel_j(x, n)` – Returns *Bessel* of the first kind order `n` of `x`

The `n` and `x` arguments are expressed as $(½ \cdot x)^n \sum(-¼ \cdot x^2)^k / k! \Gamma(n+k+1)$ summation k from 0 to ∞.

Plot example:

```
xMin    = -20.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
```

```
y1      = Special.bessel_j(x, 0)
y2      = Special.bessel_j(x, 1)
y3      = Special.bessel_j(x, 2)
y4      = Special.bessel_j(x, 3)
y5      = Special.bessel_j(x, 4)

Utils.Graph5It("Bessel J",                          \
               "Plt101_bessel_j",                   \
               Plot.CHART.LINE,                     \
               x, y1, y2, y3, y4, y5,               \
               "v = 0", "v = 1",                    \
               "v = 2", "v = 3", "v = 4",           \
               xMin, xMax, -1.0, 1.0, 0, 0)
```

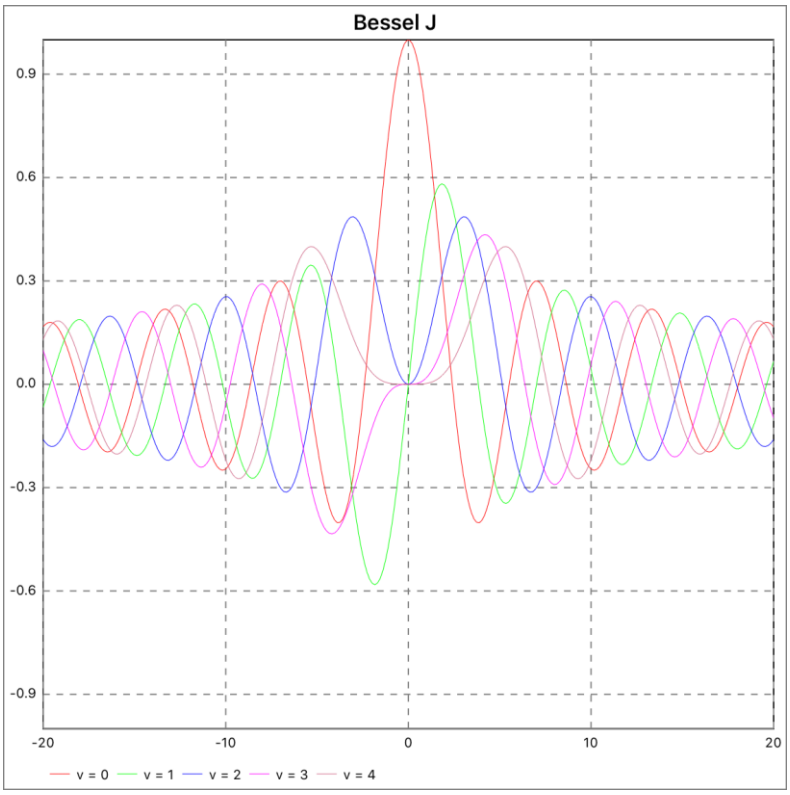

Figure 143 - Bessel J Plot

6.24.39 Special.bessel_j_pri

`special.bessel_j_pri(x, n)` – Returns *Bessel* derivative of the first kind order `n` of `x`

The `n` and `x` arguments are expressed as *(bessel_j(x, n-1) - bessel_j(x, n+1)) / 2*.

Plot example:

```
xMin    = -20.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.bessel_j_pri(x, 0)
```

```
y2      = special.bessel_j_pri(x, 1)
y3      = special.bessel_j_pri(x, 2)
y4      = special.bessel_j_pri(x, 3)
y5      = special.bessel_j_pri(x, 4)

Utils.Graph5It("Bessel J Prime",              \
               "Plt102_bessel_j_pri",         \
               Plot.CHART.LINE,               \
               x, y1, y2, y3, y4, y5,         \
               "v = 0",                       \
               "v = 1",                       \
               "v = 2",                       \
               "v = 3",                       \
               "v = 4",                       \
               xMin, xMax, -1.0, 1.0, 0, 0)
```

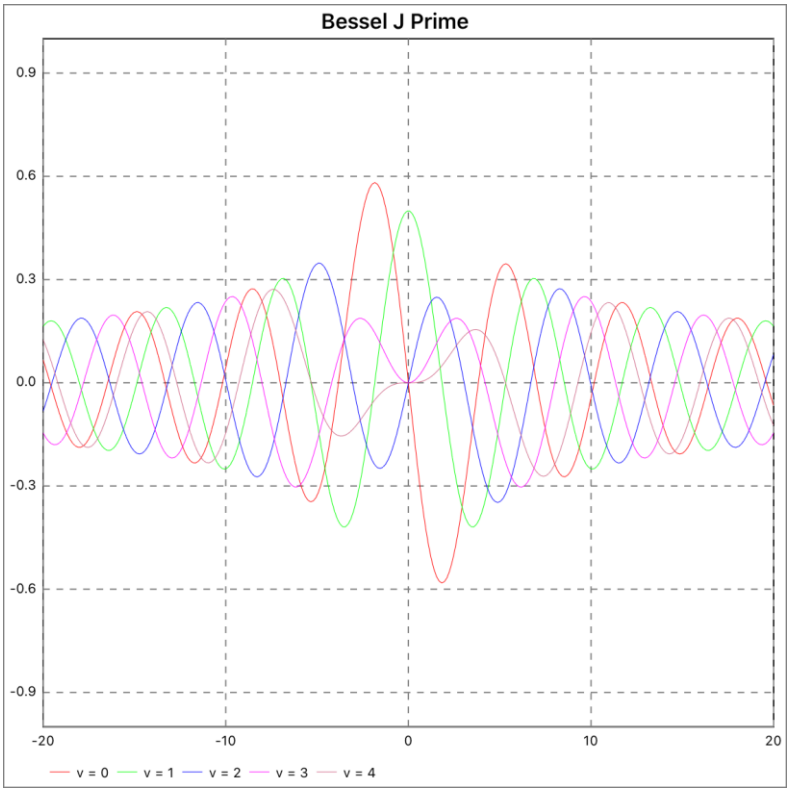

Figure 144 - Bessel J Prime Plot

6.24.40 Special.bessel_k

`special.bessel_k(x, n)` – Returns modified *Bessel* of the second kind order `n` of `x`

The `n` and `x` arguments are expressed as $\pi/2 \cdot (bessel\_i(x, n) - bessel\_i(x, n)) / sin(n\pi)$.

Plot example:

```
xMin    = 0.0
xMax    = 10.0
```

```
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = Special.bessel_k(x, 0)
y2      = Special.bessel_k(x, 2)
y3      = Special.bessel_k(x, 5)
y4      = Special.bessel_k(x, 7)
y5      = Special.bessel_k(x, 10)

Utils.Graph5It("Bessel K",                \
               "Plt103_bessel_k",         \
               Plot.CHART.LINE,           \
               x, y1, y2, y3, y4, y5,     \
               "v = 0",                   \
               "v = 2",                   \
               "v = 5",                   \
               "v = 7",                   \
               "v = 10",                  \
               xMin, xMax, 0.0, 10.0, 0, 0)
```

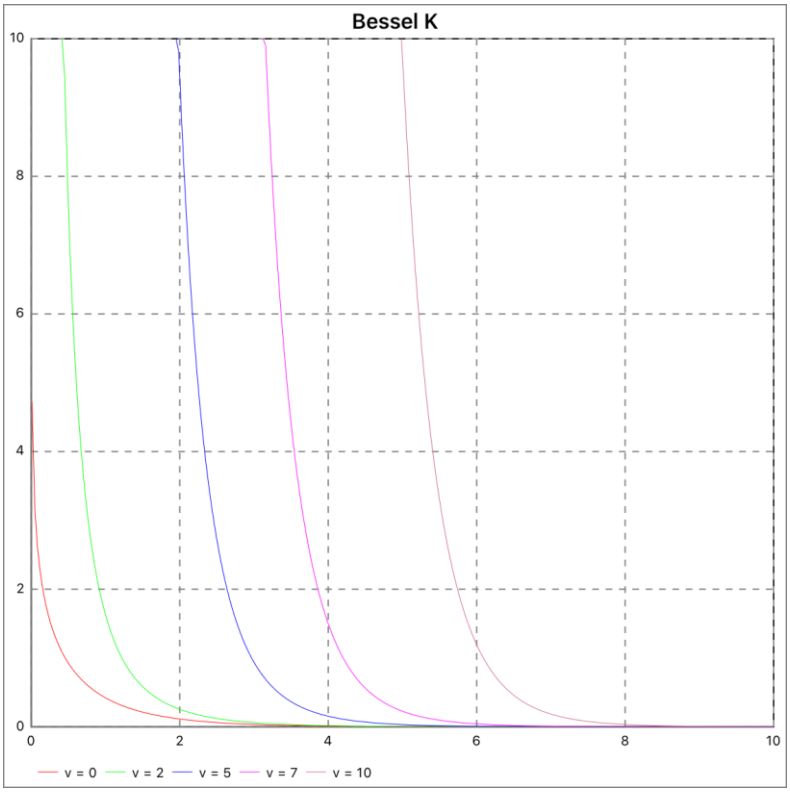

Figure 145 - Bessel K Plot

6.24.41 Special.bessel_k_pri

`special.bessel_k_pri(x, n)` – Returns modified *Bessel* derivative of the second kind order `n` of `x`

The `n` and `x` arguments are expressed as *(bessel_k(x, n–1) + bessel_k(x, n+1)) / -2*.

Plot example:

```
xMin     = 0.0
xMax     = 10.0
inc      = 0.01

x        = Utils.PlotX(xMin + inc, xMax)
y1       = special.bessel_k_pri(x, 0)
y2       = special.bessel_k_pri(x, 2)
y3       = special.bessel_k_pri(x, 5)
y4       = special.bessel_k_pri(x, 7)
y5       = special.bessel_k_pri(x, 10)

Utils.Graph5It("Bessel K Prime",              \
               "Plt104_bessel_k_pri",         \
               Plot.CHART.LINE,               \
               x, y1, y2, y3, y4, y5,         \
               "v = 0",                       \
               "v = 2",                       \
               "v = 5",                       \
               "v = 7",                       \
               "v = 10",                      \
               xMin, xMax, -20.0, 0.0, 0, 0)
```

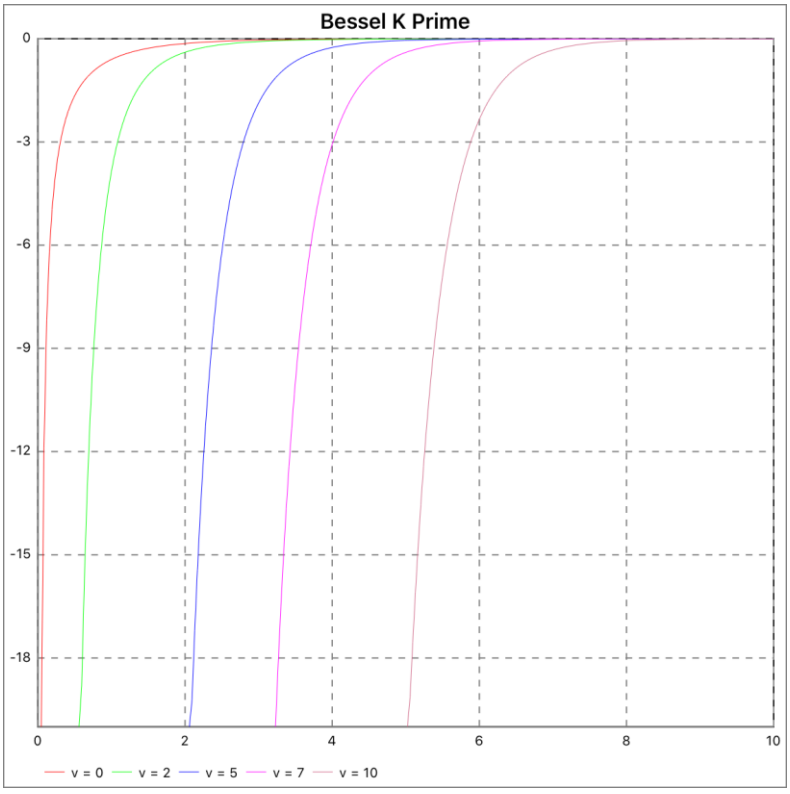

Figure 146 - Bessel K Prime Plot

6.24.42 Special.bessel_sph

`special.bessel_sph(x, n)` – Returns *Spherical Bessel* of the first kind order `n` of `x`

The n and x arguments are expressed as $\sqrt{\pi/2x} \cdot bessel\_j(x, n + ½)$.

Plot example:

```
xMin     = 0.0
xMax     = 20.0

x        = Utils.PlotX(xMin, xMax)
y1       = Special.bessel_sph(x, 0)
y2       = Special.bessel_sph(x, 2)
y3       = Special.bessel_sph(x, 5)
y4       = Special.bessel_sph(x, 7)
y5       = Special.bessel_sph(x, 10)

Utils.Graph5It("Spherical Bessel J",     \
               "Plt105_sph_bessel_j",    \
               Plot.CHART.LINE,          \
               x, y1, y2, y3, y4, y5,    \
               "v = 0",                  \
               "v = 2",                  \
               "v = 5",                  \
               "v = 7",                  \
               "v = 10",                 \
               xMin, xMax, -1.0, 1.0, 0, 0)
```

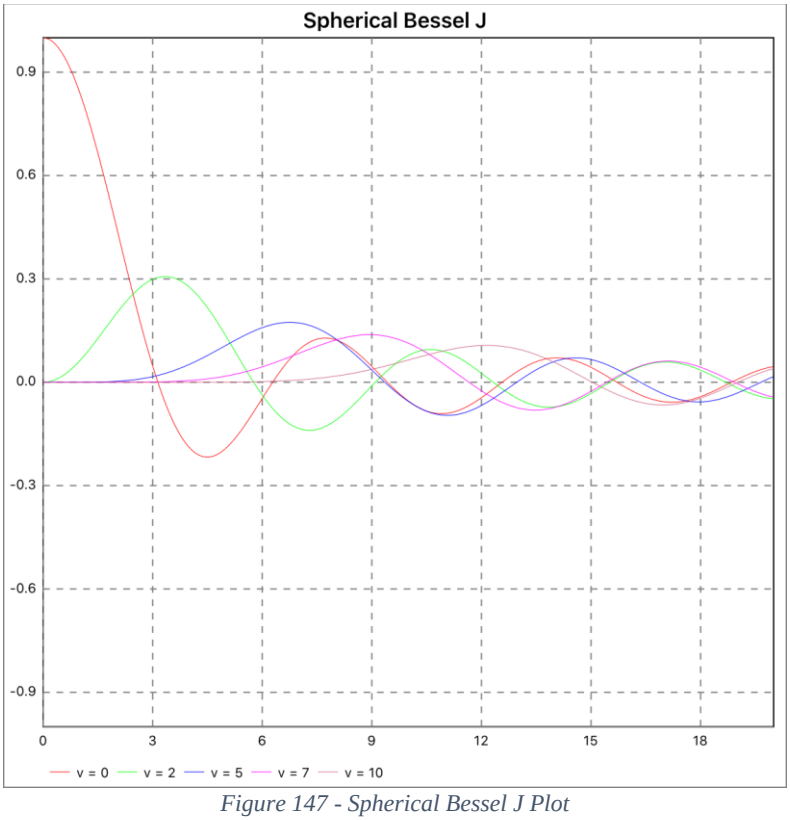

Figure 147 - Spherical Bessel J Plot

6.24.43 Special.neumann

`special.neumann(x, n)` – Returns *Bessel* of the second kind order `n` of `x`

The `n` and `x` arguments are expressed as *bessel_j(x, n) · cos(nπ) - bessel_j(x, n) / sin(nπ)*.

Plot example:

```
xMin     = 0.0
xMax     = 20.0
inc      = 0.01

x        = Utils.PlotX(xMin + inc, xMax)
y1       = Special.neumann(x, 0)
y2       = Special.neumann(x, 1)
y3       = Special.neumann(x, 2)
y4       = Special.neumann(x, 3)
y5       = Special.neumann(x, 4)

Utils.Graph5It("Bessel Y",                \
               "Plt106_neumann",          \
               Plot.CHART.LINE,           \
               x, y1, y2, y3, y4, y5,     \
               "v = 0",                   \
               "v = 1",                   \
               "v = 2",                   \
               "v = 3",                   \
               "v = 4",                   \
               xMin, xMax, -5.0, 1.0, 0, 0)
```

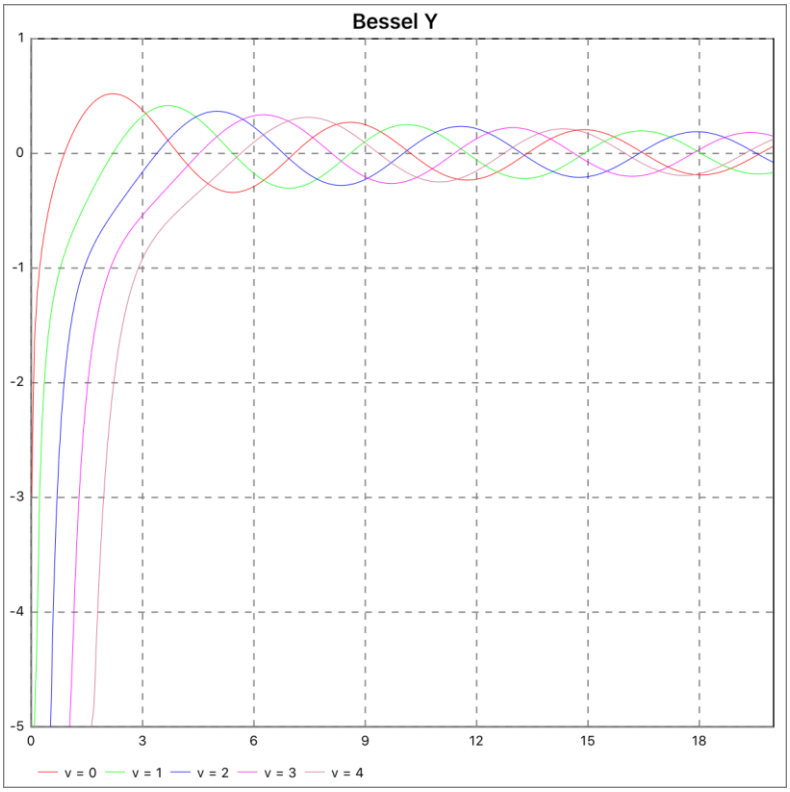

Figure 148 - Bessel Y

6.24.44 Special.neumann_pri

`special.neumann_pri(x, n)` – Returns *Bessel* derivative of the second kind order `n` of `x`

The `n` and `x` arguments are expressed as *(neumann(x, n–1) + neumann(x, n+1)) / 2*.

Plot example:

```
xMin    = 0.0
xMax    = 20.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.neumann_pri(x, 0)
y2      = special.neumann_pri(x, 1)
y3      = special.neumann_pri(x, 2)
y4      = special.neumann_pri(x, 3)
y5      = special.neumann_pri(x, 4)

Utils.Graph5It("Bessel Y Prime",            \
               "Plt107_neumann_pri",        \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "v = 0",                     \
               "v = 1",                     \
               "v = 2",                     \
               "v = 3",                     \
               "v = 4",                     \
               xMin, xMax, -1.0, 15.0, 0, 0)
```

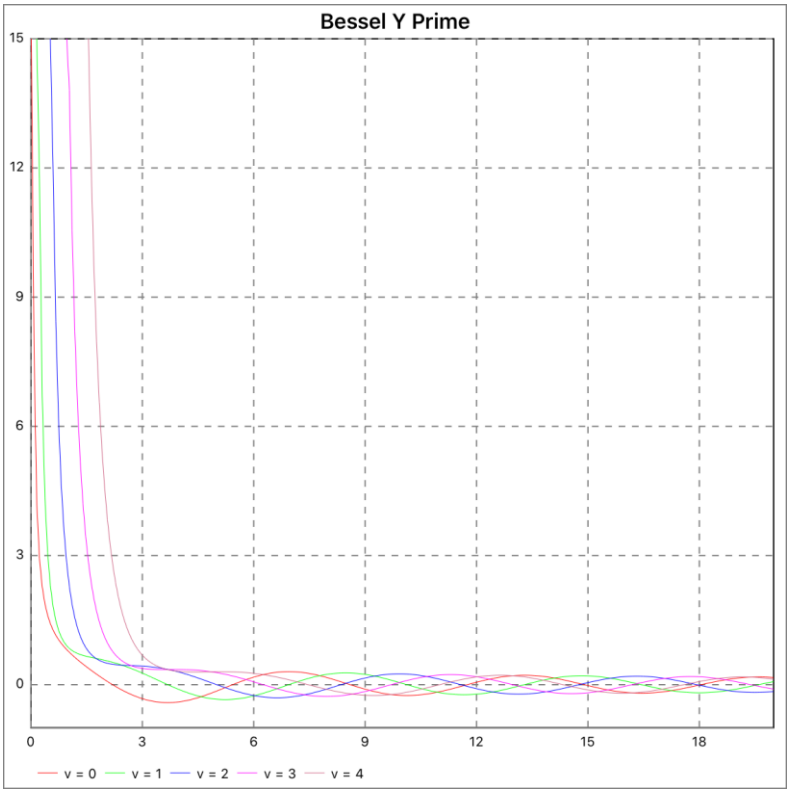

Figure 149 - Bessel Y Prime Plot

6.24.45 Special.neumann_sph

`special.neumann_sph(x, n)` – Returns *Spherical Bessel* of the second kind order `n` of `x`

The `n` and `x` arguments are expressed as $\sqrt{\pi/2x} \cdot neumann(x, n + ½)$.

Plot example:

```
xMin    = 0.0
xMax    = 20.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = Special.neumann_sph(x, 1)
y2      = Special.neumann_sph(x, 2)
y3      = Special.neumann_sph(x, 5)
y4      = Special.neumann_sph(x, 7)
y5      = Special.neumann_sph(x, 10)

Utils.Graph5It("Spherical Bessel Y",     \
               "Plt108_neumann_sph",     \
               Plot.CHART.LINE,          \
               x, y1, y2, y3, y4, y5,    \
               "v = 0",                  \
               "v = 2",                  \
               "v = 5",                  \
               "v = 7",                  \
```

```
                "v = 10",                   \
                xMin, xMax, -5.0, 1.0, 0, 0)
```

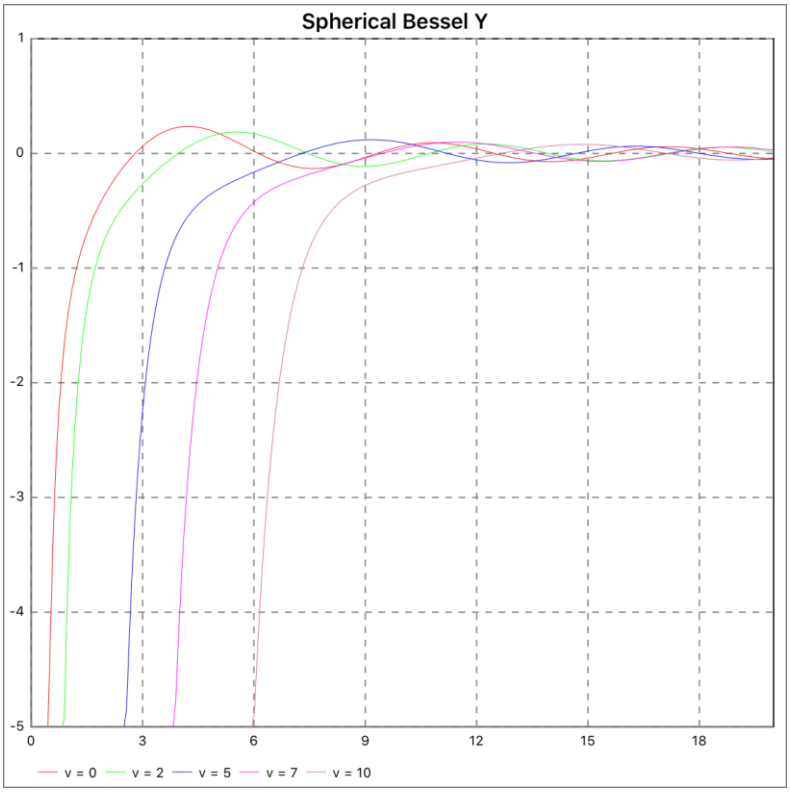

Figure 150 - Spherical Bessel Y Plot

6.24.46 Special.hankel_1

`special.hankel_1(x, n)` – Returns *Hankel* of the first kind order `n` of `x`

The `x` and `n` arguments are expressed as *bessel_j(x, n) + neumman(x, n).*

Plot example:

```
xMin    = -20.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = toReal(special.hankel_1(x, 0))
y2      = toReal(special.hankel_1(x, 1))
y3      = toReal(special.hankel_1(x, 2))
y4      = toReal(special.hankel_1(x, 3))
y5      = toReal(special.hankel_1(x, 4))

Utils.Graph5It("Hankel 1",                  \
               "Plt109_hankel_1",           \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
               "v = 0",                     \
               "v = 1",                     \
```

```
                "v = 2",                        \
                "v = 3",                        \
                "v = 4",                        \
                xMin, xMax, 0.0, 20.0, 0, 0)
```

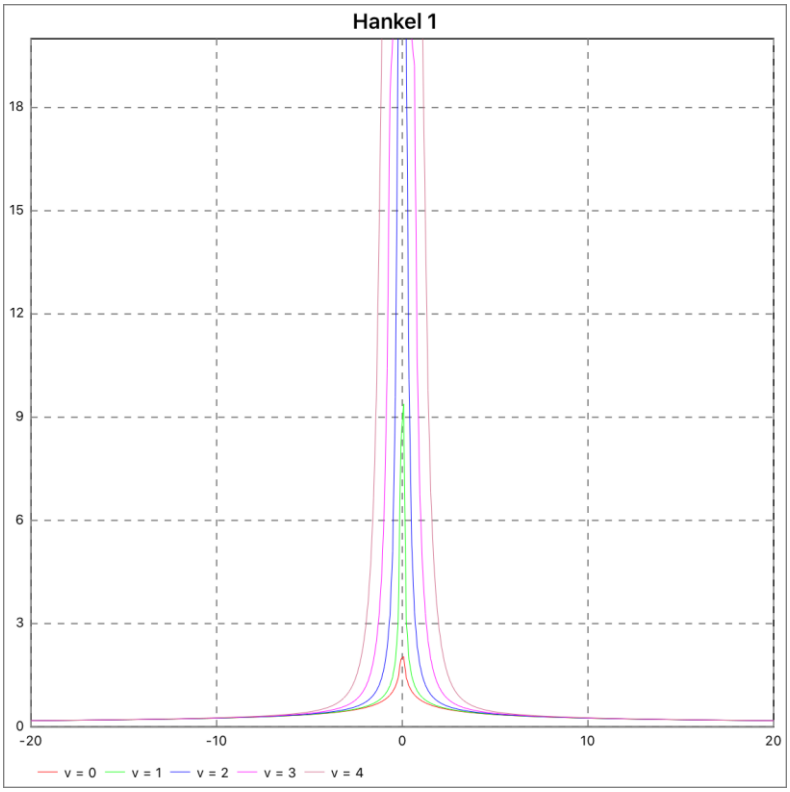

Figure 151 - Hankel 1 Plot

6.24.47 Special.hankel_1_sph

`special.hankel_1_sph(x, n)` – Returns *Spherical Hankel* of the first kind order `n` of `x`

The `x` and `n` arguments are expressed as $\sqrt{\pi/2} \cdot 1/\sqrt{x} \cdot hankel\_1(x, n+½)$.

Plot example:

```
xMin    = -20.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = toReal(special.hankel_1_sph(x, 0))
y2      = toReal(special.hankel_1_sph(x, 1))
y3      = toReal(special.hankel_1_sph(x, 2))
y4      = toReal(special.hankel_1_sph(x, 3))
y5      = toReal(special.hankel_1_sph(x, 4))

Utils.Graph5It("Spherical Hankel 1",        \
               "Plt110_hankel_1_sph",       \
               Plot.CHART.LINE,             \
               x, y1, y2, y3, y4, y5,       \
```

```
                "v = 0",                        \
                "v = 1",                        \
                "v = 2",                        \
                "v = 3",                        \
                "v = 4",                        \
                xMin, xMax, 0.0, 20.0, 0, 0)
```

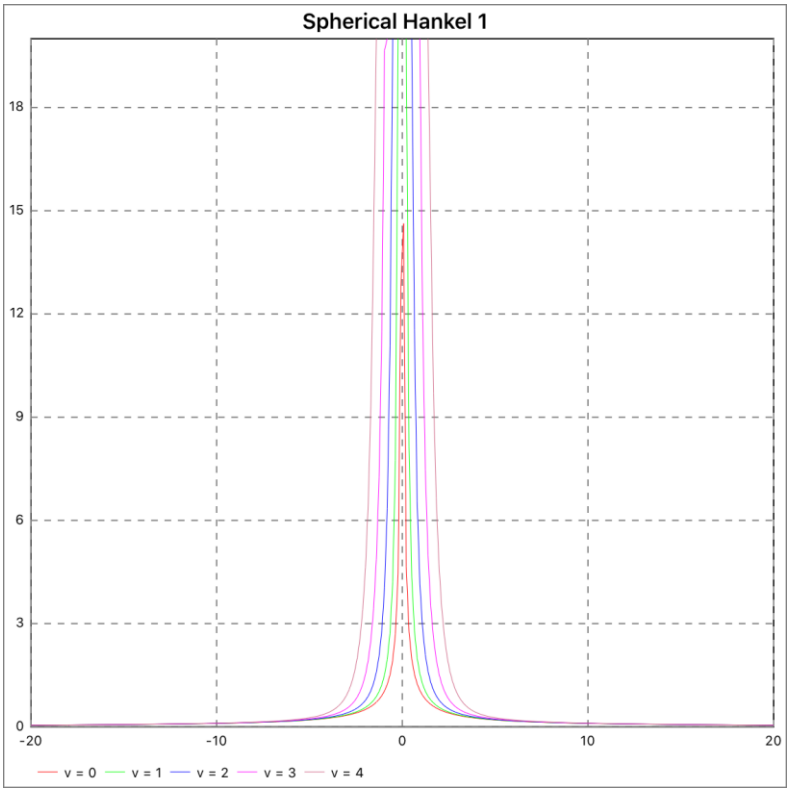

Figure 152 - Spherical Hankel 1 Plot

6.24.48 Special.hankel_2

`special.hankel_2(x, n)` – Returns *Hankel* of the second kind order `n` of `x`

The `x` and `n` arguments are expressed as *bessel_j(x, n) - neumman(x, n)*.

Plot example:

```
xMin    = -20.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = toReal(special.hankel_2(x, 0))
y2      = toReal(special.hankel_2(x, 1))
y3      = toReal(special.hankel_2(x, 2))
y4      = toReal(special.hankel_2(x, 3))
y5      = toReal(special.hankel_2(x, 4))

Utils.Graph5It("Hankel 2",                      \
               "Plt111_hankel_2",               \
```

```
Plot.CHART.LINE,                \
x, y1, y2, y3, y4, y5,          \
"v = 0",                        \
"v = 1",                        \
"v = 2",                        \
"v = 3",                        \
"v = 4",                        \
xMin, xMax, 0.0, 20.0, 0, 0)
```

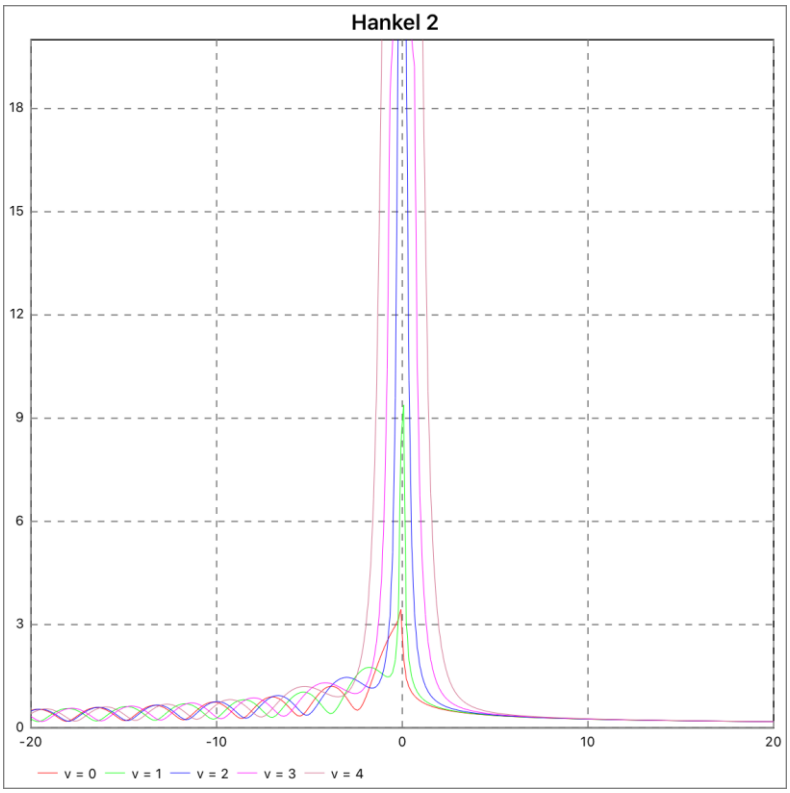

Figure 153 - Hankel 2 Plot

6.24.49 Special.hankel_2_sph

`special.hankel_2_sph(x, n)` – Returns *Spherical Hankel* of the second kind order `n` of `x`

The `x` and `n` arguments are expressed as $\sqrt{\pi/2} \cdot 1/\sqrt{x} \cdot hankel\_2(x, n+½)$.

Plot example:

```
xMin    = -20.0
xMax    = 20.0

x       = Utils.PlotX(xMin, xMax)
y1      = toReal(special.hankel_2_sph(x, 0))
y2      = toReal(special.hankel_2_sph(x, 1))
y3      = toReal(special.hankel_2_sph(x, 2))
y4      = toReal(special.hankel_2_sph(x, 3))
y5      = toReal(special.hankel_2_sph(x, 4))
```

```
Utils.Graph5It("Spherical Hankel 2",          \
               "Plt112_hankel_2_sph",         \
               Plot.CHART.LINE,               \
               x, y1, y2, y3, y4, y5,         \
               "v = 0",                       \
               "v = 1",                       \
               "v = 2",                       \
               "v = 3",                       \
               "v = 4",                       \
               xMin, xMax, 0.0, 20.0, 0, 0)
```

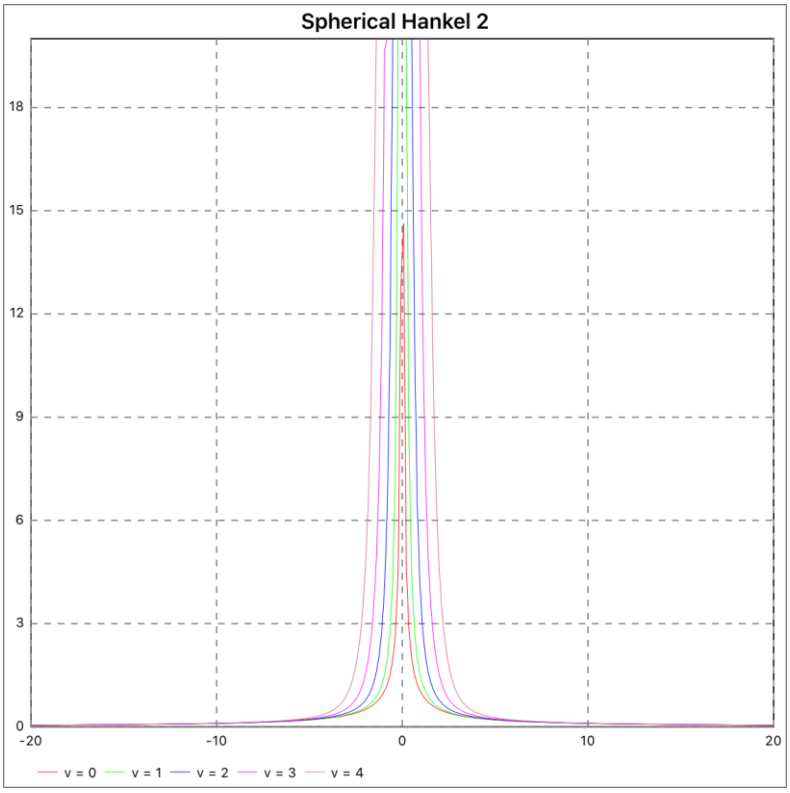

Figure 154 - Spherical Hankel 2 Plot

6.24.50 Special.airy_ai

`special.airy_ai(x)` – Returns *Airy's* Ai of `x`

The `x` argument is expressed in two following expressions:

- `x > 0`, $\pi^{-1} \cdot \sqrt{bessel\_k(\pm x/3, (\frac{2}{3} \cdot x^{\frac{2}{3}})}$
- `x < 0`, $(\sqrt{x/3}) \cdot (bessel\_j(\frac{1}{3}, \frac{2}{3} \cdot x^{\frac{2}{3}}) + bessel\_j(-\frac{1}{3}, \frac{2}{3} \cdot x^{\frac{2}{3}}))$

⇒ LINK: https://en.wikipedia.org/wiki/Airy_function

Plot example:

```
xMin     = -20.0
xMax     = 10.0
```

```
x       = Utils.PlotX(xMin, xMax)
y       = Special.airy_ai(x)

Utils.PlotIt("Airy Ai",                  \
             "Plt113_airy_ai",           \
             Plot.CHART.LINE,            \
             x, y,                       \
             xMin, xMax, -0.5, 0.6, 0, 0)
```

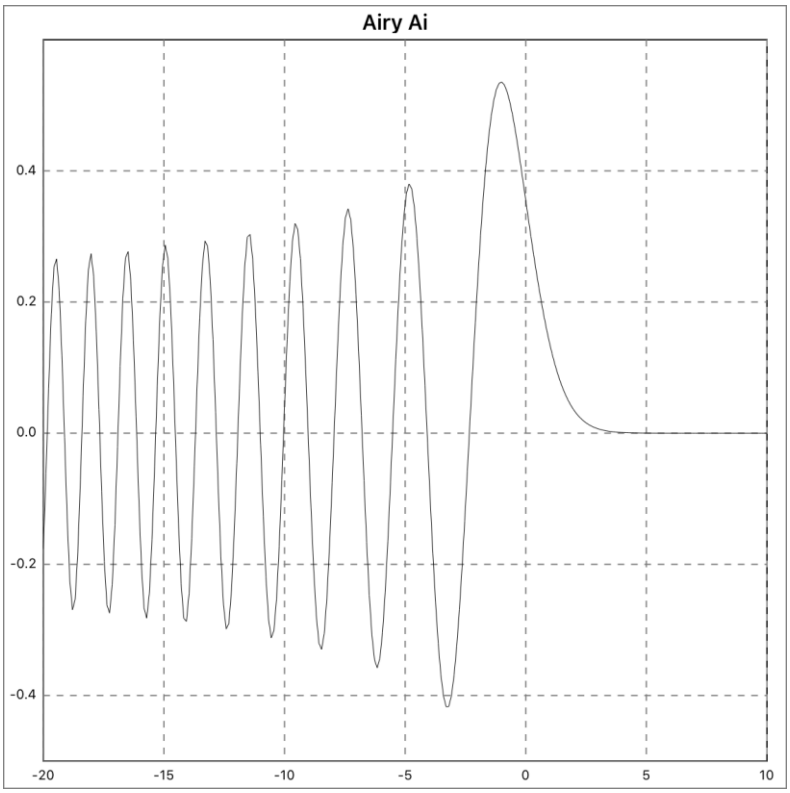

Figure 155 - Airy Ai Plot

6.24.51 Special.airy_ai_pri

`special.airy_ai_pri(x)` – Returns *Airy*'s derivative Ai of `x`

The `x` argument is expressed in two following expressions:

- `x > 0`, $x/\pi\sqrt{3} \cdot \sqrt{bessel\_k(\pm 2/3, (2/3 \cdot x^{2/3})}$
- `x < 0`, $x/3 \cdot (bessel\_j(2/3, 2/3 \cdot x^{2/3}) - bessel\_j(-2/3, 2/3 \cdot x^{2/3}))$

Plot example:

```
xMin    = -20.0
xMax    = 5.0

x       = Utils.PlotX(xMin, xMax)
y       = Special.airy_ai_pri(x)

Utils.PlotIt("Airy Ai'",                 \
```

```
                "Plt114_airy_ai_pri",       \
                Plot.CHART.LINE,            \
                x, y,                       \
                xMin, xMax, -1.5, 1.5, 0, 0)
```

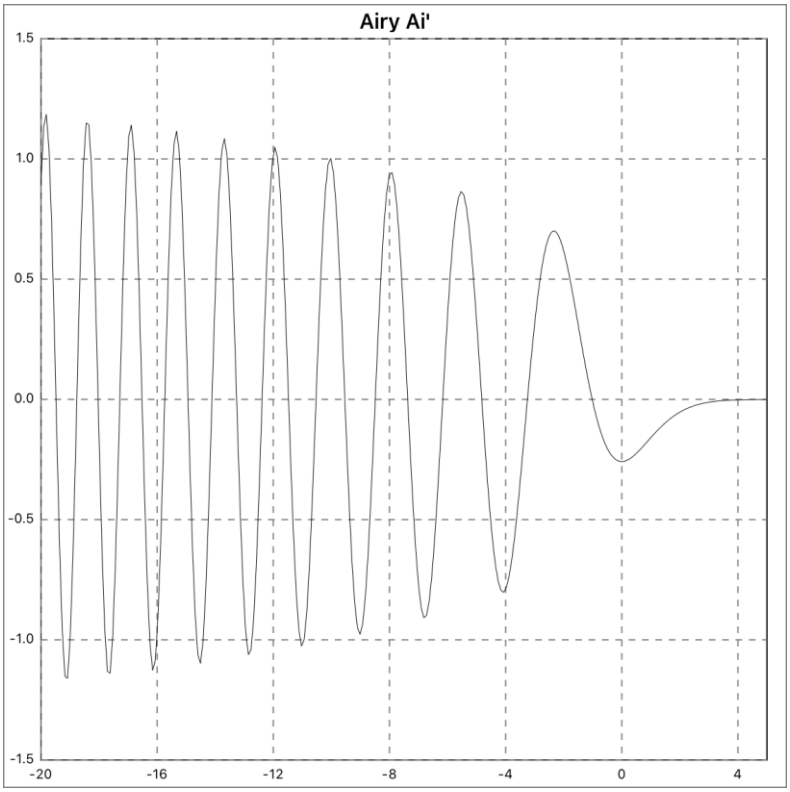

Figure 156 - Airy Ai Prime Plot

6.24.52 Special.airy_bi

`special.airy_bi(x)` – Returns *Airy's* Bi of `x`

The `x` argument is expressed in two following expressions:

- `x > 0`, $(\sqrt{x/3}) \cdot (bessel\_i(⅓, ⅔ \cdot x^{⅔}) + bessel\_i(-⅓, ⅔ \cdot x^{⅔}))$
- `x < 0`, $(\sqrt{x/3}) \cdot (bessel\_j(-⅓, ⅔ \cdot x^{⅔}) - bessel\_j(⅓, ⅔ \cdot x^{⅔}))$

Plot example:

```
xMin    = -20.0
xMax    = 5.0

x       = Utils.PlotX(xMin, xMax)
y       = Special.airy_bi(x)

Utils.PlotIt("Airy Bi",                 \
             "Plt115_airy_bi",          \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, -0.5, 15.0, 0, 0)
```

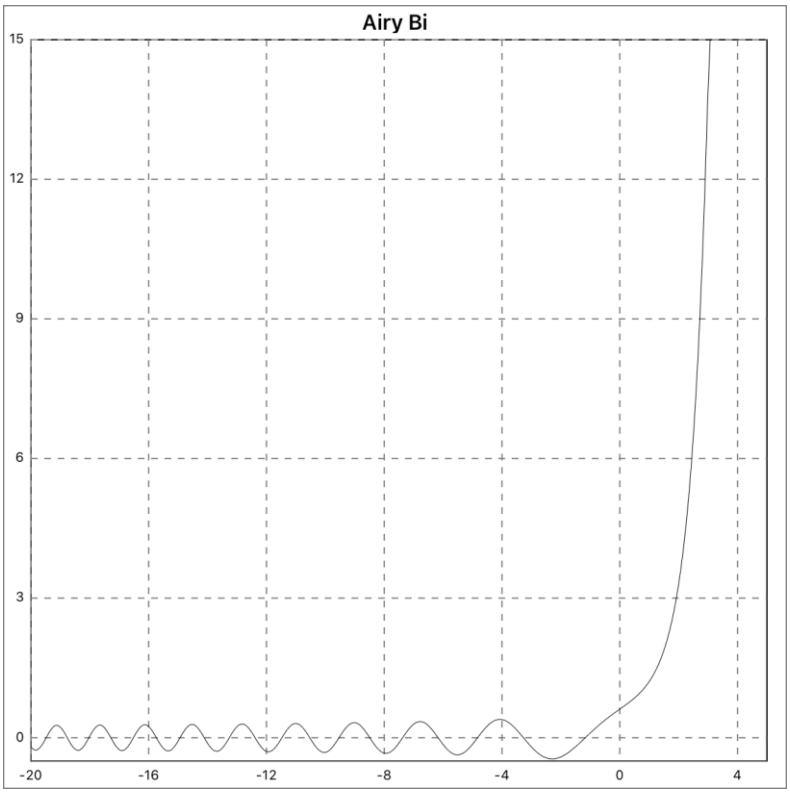

Figure 157 - Airy Bi Plot

6.24.53 Special.airy_bi_pri

`special.airy_bi_pri(x)` – Returns *Airy's* derivative Bi of `x`

The `x` argument is expressed in two following expressions:

- x > 0, $(x/\sqrt{3}) \cdot (bessel\_i(2/3, 2/3 \cdot x^{2/3}) + bessel\_i(-2/3, 2/3 \cdot x^{2/3}))$
- x < 0, $(x/\sqrt{3}) \cdot (bessel\_j(-2/3, 2/3 \cdot x^{2/3}) + bessel\_j(2/3, 2/3 \cdot x^{2/3}))$

Plot example:

```
xMin    = -20.0
xMax    = 2.5

x       = Utils.PlotX(xMin, xMax)
y       = Special.airy_bi_pri(x)

Utils.PlotIt("Airy Bi'",                \
             "Plt116_airy_bi_pri",      \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, -2.0, 15.0, 0, 0)
```

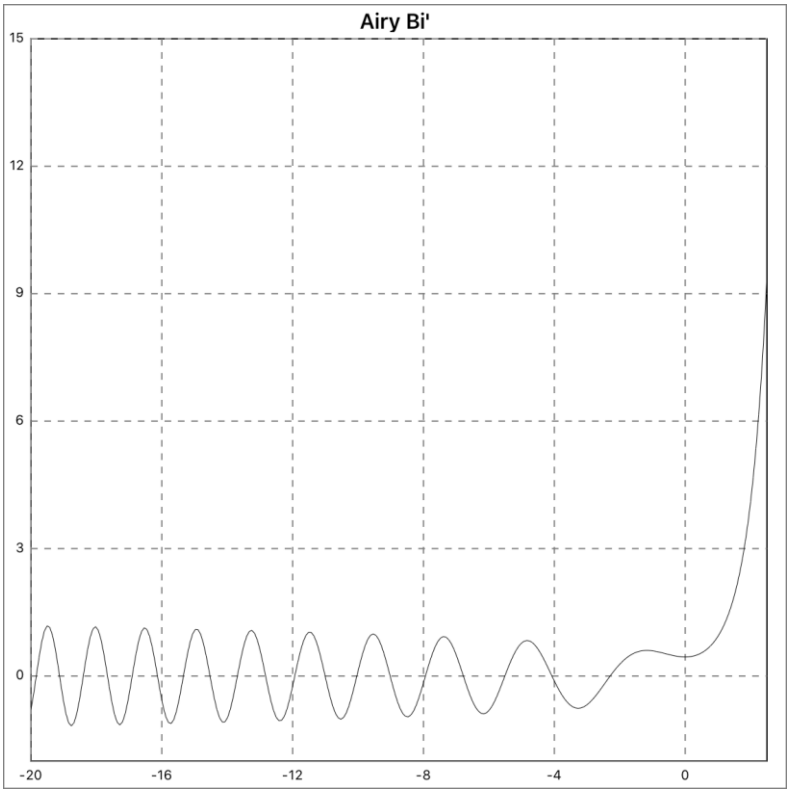

Figure 158 - Airy Bi Prime Plot

6.24.54 Special.elliptic_int_1

`special.elliptic_int_1(x)` – Returns complete *Elliptic Integral* of the first kind of `x`
`special.elliptic_int_1(x, p)`

The `x` argument is expressed as $\int d\theta / \sqrt{(1-x^2 \cdot \sin^2\theta)}$ over an integral `0` to $\pi/2$. The optional `p` argument's integral will be from `0` to φ. The range of `x` is `-1 ≤ x ≤ 1`.

⇒ LINK: https://en.wikipedia.org/wiki/Elliptic_integral

Plot example:

```
xMin    = -1.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y       = special.elliptic_int_1(x)

Utils.PlotIt("Elliptic of the 1st Kind A",  \
             "Plt117_elliptic_int1",        \
             Plot.CHART.LINE,               \
             x, y,                          \
             xMin, xMax, 0.0, 2.5, 0, 0)
```

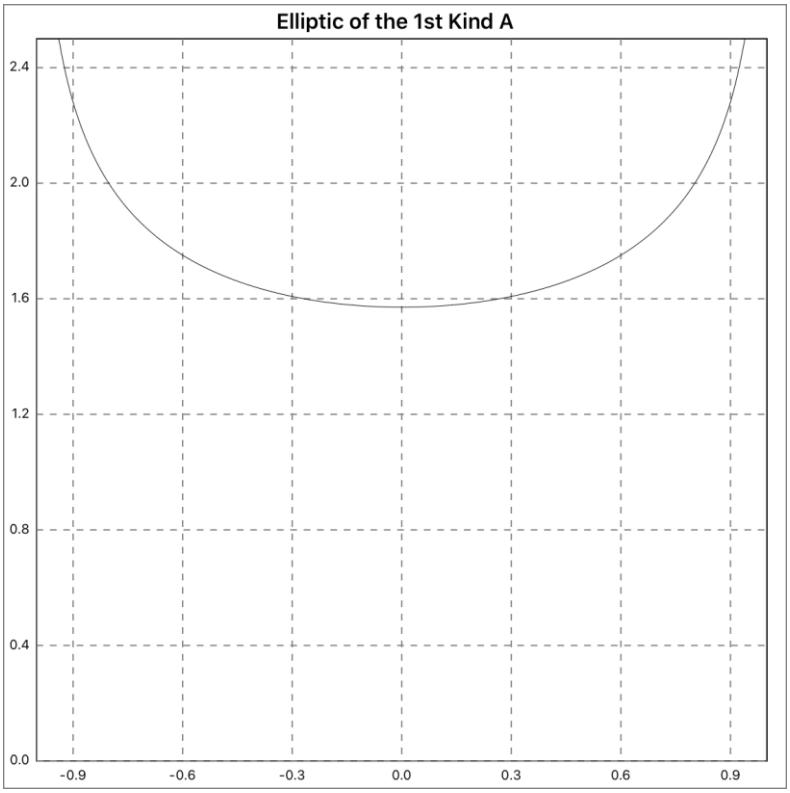

Figure 159 - Elliptic of the 1st Kind A Plot

Another plot example:

```
xMin    = -1.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.elliptic_int_1(x, 0.5)
y2      = Special.elliptic_int_1(x, 0.75)
y3      = Special.elliptic_int_1(x, 1.25)

Utils.Graph3It("Elliptic of the 1st Kind B", \
               "Plt118_elliptic_int1",       \
               Plot.CHART.LINE,              \
               x, y1, y2, y3,                \
               "phi = 0.5",                  \
               "phi = 0.75",                 \
               "phi = 1.25",                 \
               xMin, xMax, 0.0, 2.5, 0, 0)
```

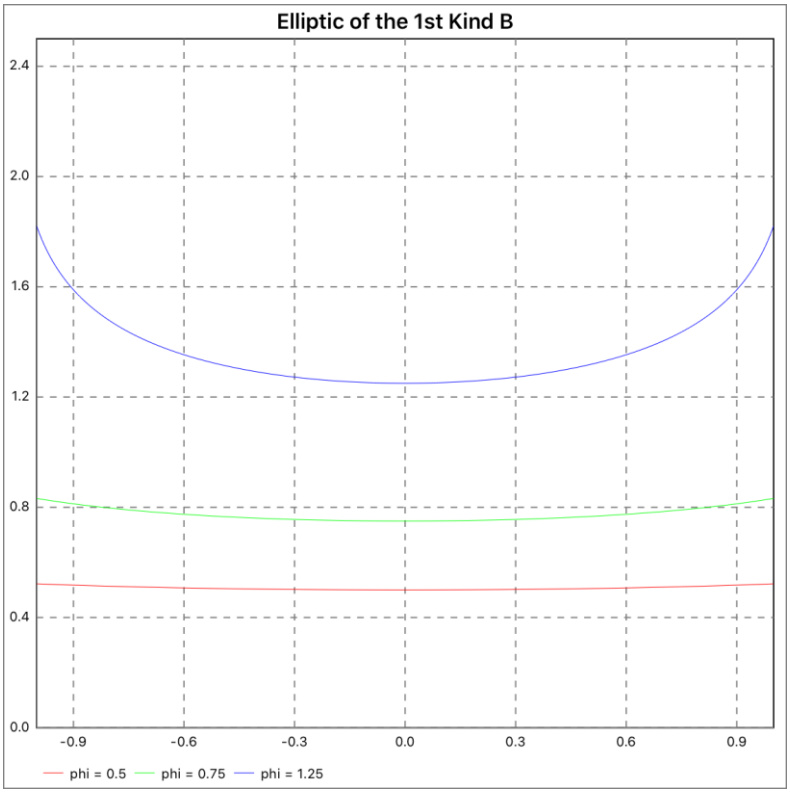

Figure 160 - Elliptic of the 1st Kind B Plot

6.24.55 Special.elliptic_int_2

`special.elliptic_int_2(x)` – Returns complete *Elliptic Integral* of the second kind of `x`
`special.elliptic_int_2(x, p)`

The `x` argument is expressed as $\int d\boldsymbol{\theta} \cdot \sqrt{(1-x^2 \cdot sin^2\boldsymbol{\theta})}$ over an integral `0` to $\pi/2$. The optional `p` argument's integral will be from `0` to **φ**. The range of `x` is `-1 ≤ x ≤ 1`.

Plot example:

```
xMin    = -1.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y       = special.elliptic_int_2(x)

Utils.PlotIt("Elliptic of the 2nd Kind A",  \
             "Plt119_elliptic_int2",        \
             Plot.CHART.LINE,               \
             x, y,                          \
             xMin, xMax, 0.0, 2.0, 0, 0)
```

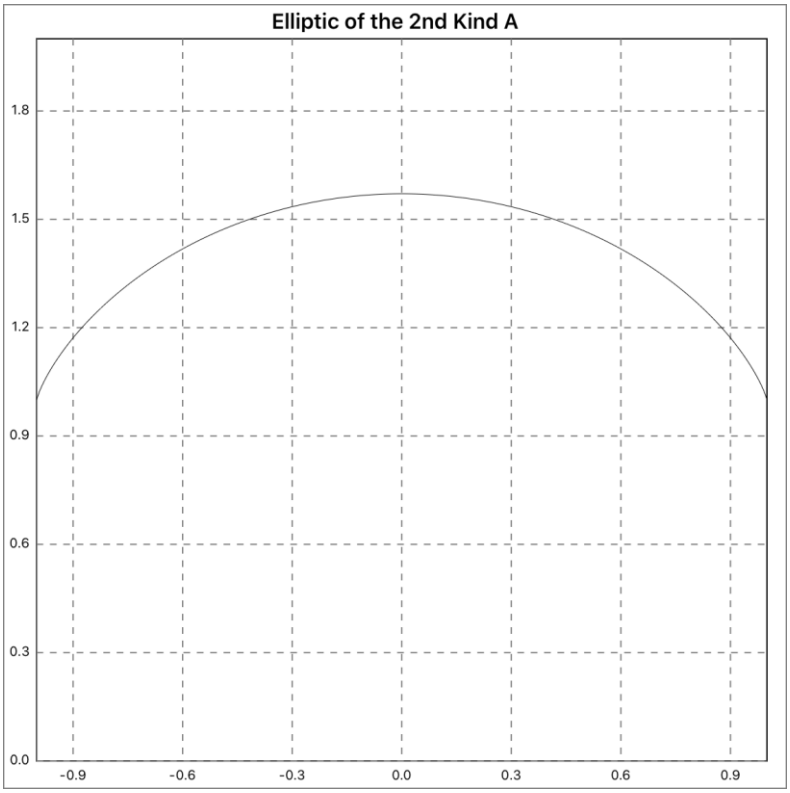

Figure 161 - Ellipitic of the 2nd Kind A Plot

Another plot example:

```
xMin    = -1.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.elliptic_int_2(x, 0.5)
y2      = Special.elliptic_int_2(x, 0.75)
y3      = Special.elliptic_int_2(x, 1.25)

Utils.Graph3It("Elliptic of the 2nd Kind B", \
               "Plt120_elliptic_int2",       \
               Plot.CHART.LINE,              \
               x, y1, y2, y3,                \
               "phi = 0.5",                  \
               "phi = 0.75",                 \
               "phi = 1.25",                 \
               xMin, xMax, 0.0, 2.0, 0, 0)
```

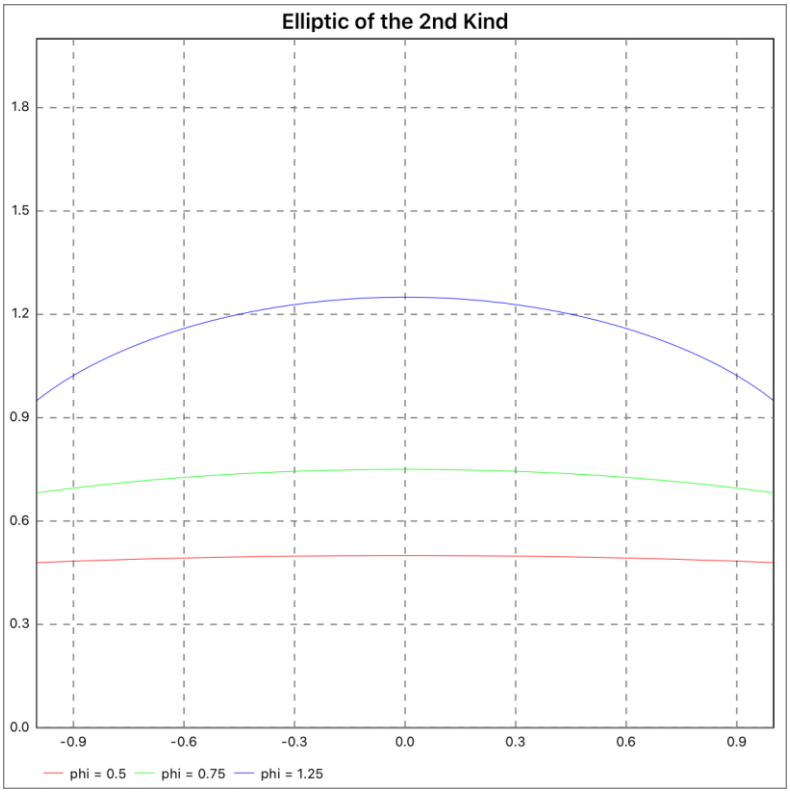

Figure 162 - Ellipitic of the 2nd Kind B Plot

6.24.56 Special.elliptic_int_3

`special.elliptic_int_3(x, n)` – Returns complete *Elliptic Integral* of the third kind `n` of `x`
`special.elliptic_int_3(x, n, p)`

The `x` argument is expressed as $\int d\theta / [(1-n \cdot sin^2\theta) \cdot \sqrt{1-x^2 \cdot sin^2\theta}]$ over an integral `0` to $\pi/2$. The optional `p` argument's integral will be from `0` to φ. The range of `x` is `-1 ≤ x ≤ 1`, and, the range of `n` is `n < 1`.

Plot example:

```
xMin    = -1.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.elliptic_int_3(x, 0.00)
y2      = special.elliptic_int_3(x, 0.50)
y3      = special.elliptic_int_3(x, 0.25)
y4      = special.elliptic_int_3(x, 0.50)

Utils.Graph4It("Elliptic of the 3rd Kind A", \
               "Plt121_elliptic_int3",       \
               Plot.CHART.LINE,              \
               x, y1, y2, y3, y4,            \
               "n = 0",                      \
               "n = 0.5",                    \
               "n = 0.25",                   \
```

```
                "n = 0.75",                     \
                xMin, xMax, 1.0, 4.0, 0, 0)
```

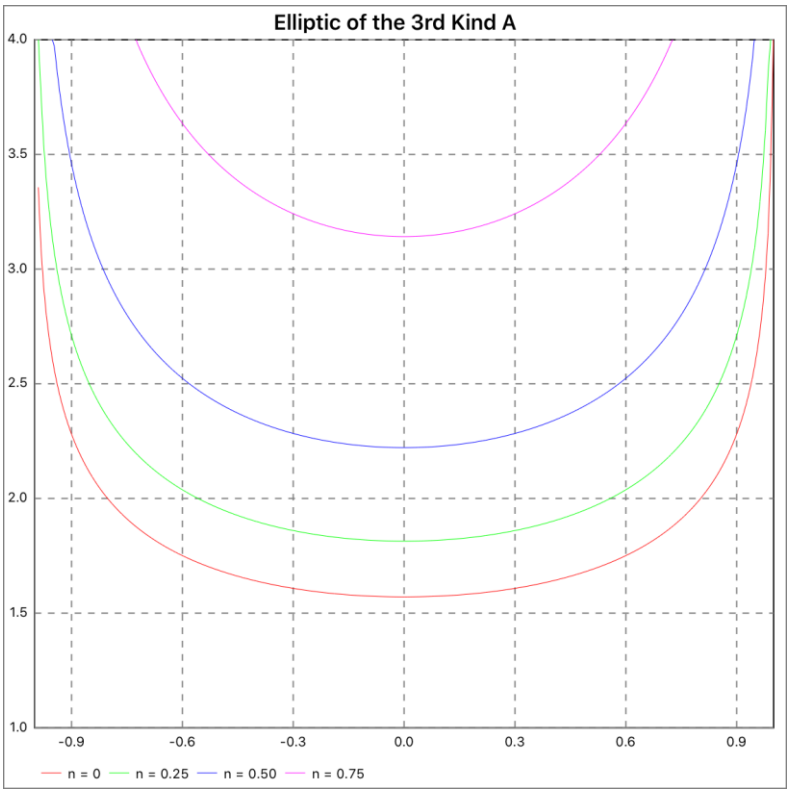

Figure 163 - Elliptic of the 3rd Kind A Plot

Another plot example:

```
xMin    = -1.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.elliptic_int_3(x, 0.00, 1.25)
y2      = Special.elliptic_int_3(x, 0.50, 1.25)
y3      = Special.elliptic_int_3(x, 0.25, 0.25 ÷ 2.0)
y4      = Special.elliptic_int_3(x, 0.50, 0.75 ÷ 2.0)

Utils.Graph4It("Elliptic of the 3rd Kind B", \
               "Plt122_elliptic_int3",       \
               Plot.CHART.LINE,              \
               x, y1, y2, y3, y4,            \
               "n = 0, phi = 1.25",          \
               "n = 0.5, phi = 1.25",        \
               "n = 0.25, phi = n / 2",      \
               "n = 0.75, phi = n2 / 2",     \
               xMin, xMax, 1.0, 4.0, 0, 0)
```

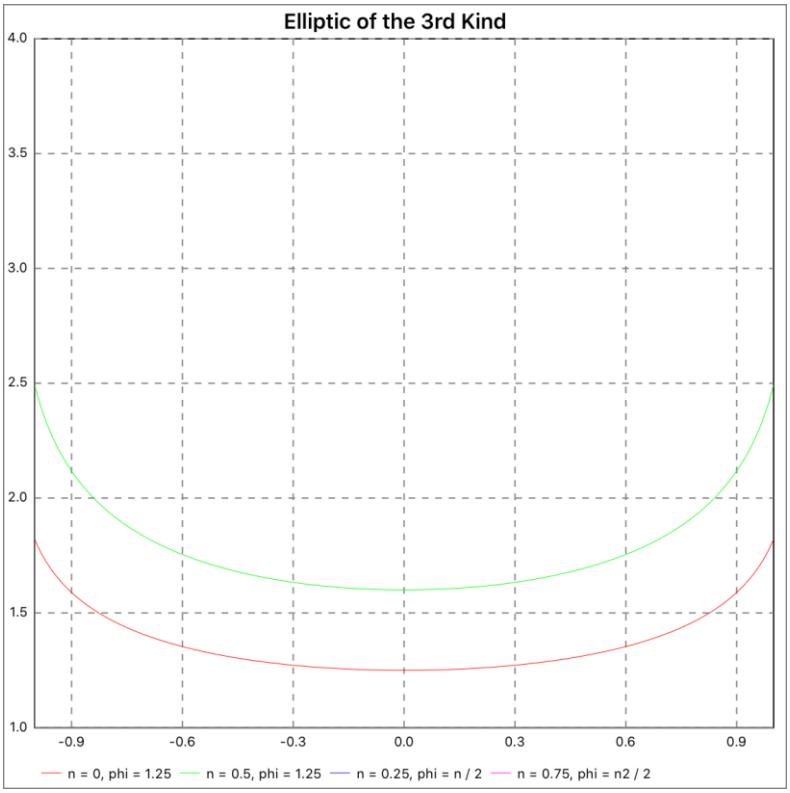

Figure 164 - Elliptic of the 3rd Kind B Plot

6.24.57 Special.elliptic_int_d

`special.elliptic_int_d(x)` – Returns complete *Elliptic Integral* D of x
`special.elliptic_int_d(x, p)`

The `x` argument is expressed as $d\theta \int sin^2\theta / [\sqrt{1-x^2 \cdot sin^2\theta}]$ over an integral `0` to $\pi/2$. The optional `p` argument's integral will be from 0 to φ. The range of x is `-1 ≤ x ≤ 1`.

⇒ LINK: https://en.wikipedia.org/wiki/Legendre_form

Plot example:

```
xMin    = -1.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y       = special.elliptic_int_d(x)

Utils.PlotIt("Elliptic Integral D A",       \
             "Plt123_elliptic_d",           \
             Plot.CHART.LINE,               \
             x, y,                          \
             xMin, xMax, 0.0, 4.0, 0, 0)
```

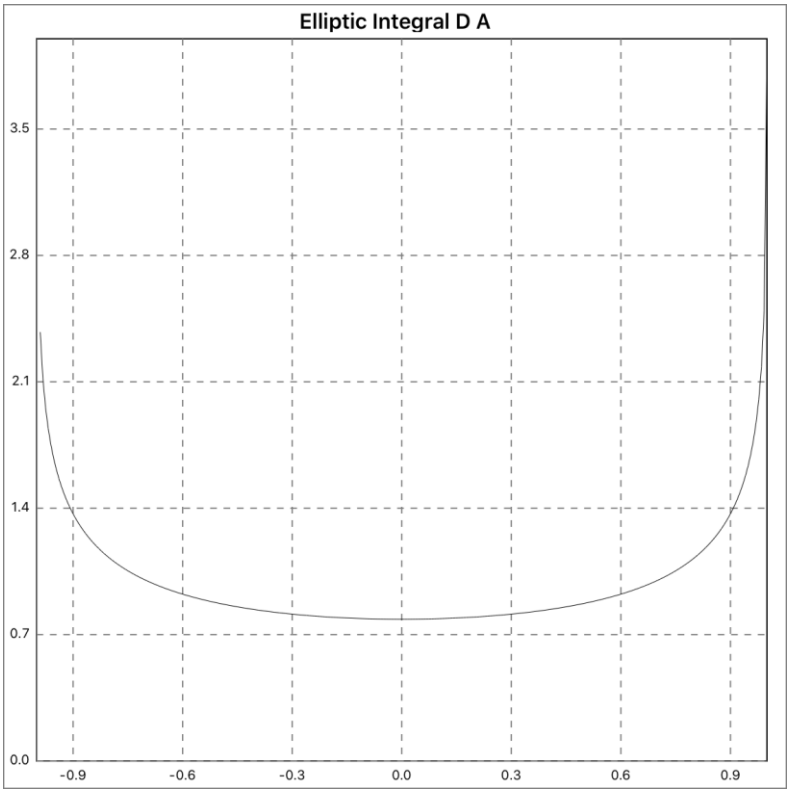

Figure 165 - Elliptic Integral D A Plot

Another plot example:

```
xMin    = -1.0
xMax    = 1.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.elliptic_int_d(x, 0.50)
y2      = special.elliptic_int_d(x, 0.75)
y3      = special.elliptic_int_d(x, 1.00)

Utils.Graph3It("Elliptic Integral D B",     \
               "Plt124_elliptic_d",         \
               Plot.CHART.LINE,             \
               x, y1, y2, y3,               \
               "p = 0.50",                  \
               "p = 0.75",                  \
               "p = 1.0",                   \
               xMin, xMax, 0.0, 0.4, 0, 0)
```

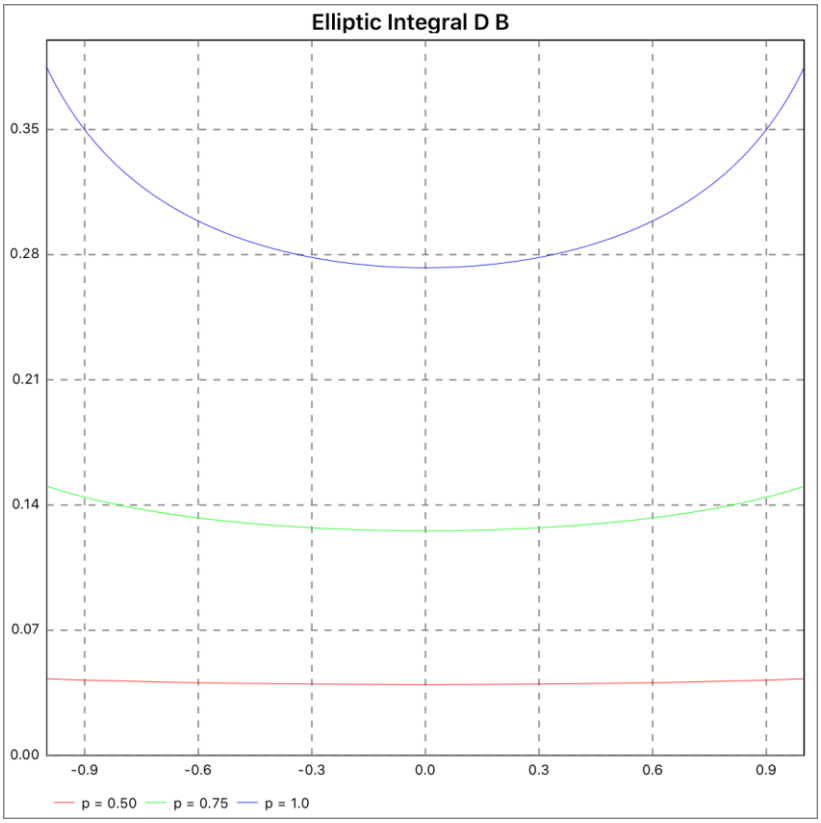

Figure 166 - Elliptic Integral D B Plot

6.24.58 Special.elliptic_int_rc

`special.elliptic_int_rc(x, y)` – Returns Carlson's *Elliptic Integral* of R_C of `x` and `y`

The `x` and `y` arguments are expressed as $\frac{1}{2}\int (t+x)^{-\frac{1}{2}}(t+y)^{-1}dt$ over an integral `0` to ∞. The range of x is `0 ≤ x < ∞` and range of `y` is `y != 0`.

⇒ LINK: https://en.wikipedia.org/wiki/Carlson_symmetric_form

Plot example:

```
xMin    = -10.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.elliptic_int_rc(x, 0.50)
y2      = special.elliptic_int_rc(x, 0.75)
y3      = special.elliptic_int_rc(x, 1.00)

Utils.Graph3It("Carlson's Elliptic Integral of R(c)", \
               "Plt125_elliptic_int_rc",              \
               Plot.CHART.LINE,                       \
               x, y1, y2, y3,                         \
               "y = 0.50",                            \
               "y = 0.75",                            \
               "y = 1.0",                             \
```

```
                xMin, xMax, 0.0, 4.0, 0, 0)
```

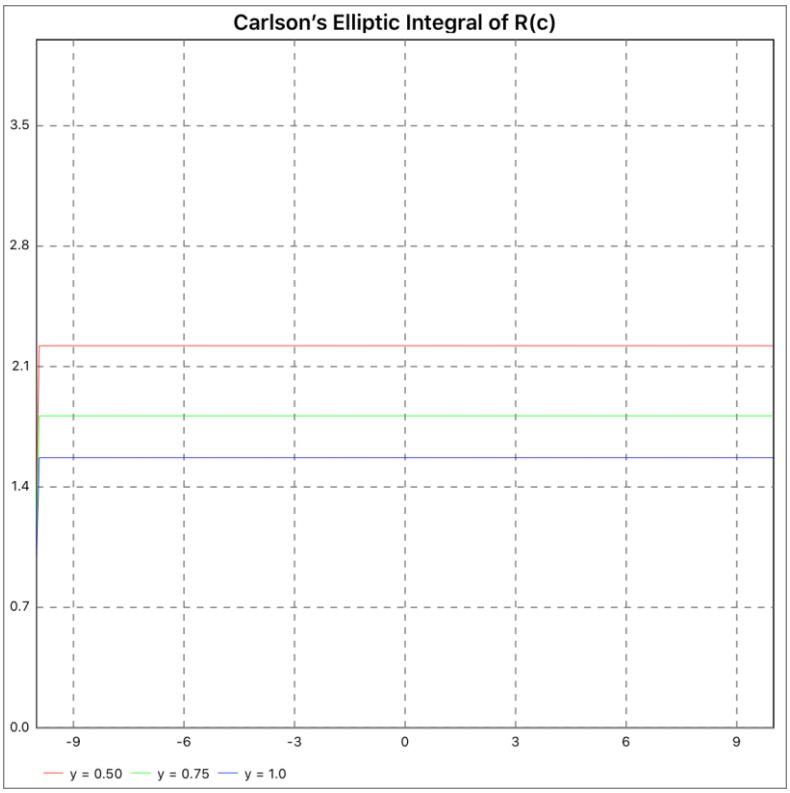

Figure 167 - Carlson's Elliptic Integral of R(c) Plot

6.24.59 Special.elliptic_int_rd

`special.elliptic_int_rd(x, y, z)` – Returns Carlson's *Elliptic Integral* of R_D of `x`, `y`, and `z`

The `x`, `y`, and `z` arguments are expressed as $\frac{3}{2}\int [(t+x)\,(t+y)]^{-\frac{1}{2}}(t+z)^{-\frac{3}{2}}dt$ over an integral `0` to ∞. The range of `x` is $0 \leq x < \infty$.

Plot example:

```
xMin    = -10.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.elliptic_int_rd(x, 0.50, 1.0)
y2      = special.elliptic_int_rd(x, 0.75, 1.0)
y3      = special.elliptic_int_rd(x, 1.00, 1.0)

Utils.Graph3It("Carlson's Elliptic Integral of R(d)", \
               "Plt126_elliptic_int_rd",              \
               Plot.CHART.LINE,                       \
               x, y1, y2, y3,                         \
               "y = 0.50, z = 1.0",                   \
               "y = 0.75, z = 1.0",                   \
               "y = 1.00, z = 1.0",                   \
               xMin, xMax, 0.0, 4.0, 0, 0)
```

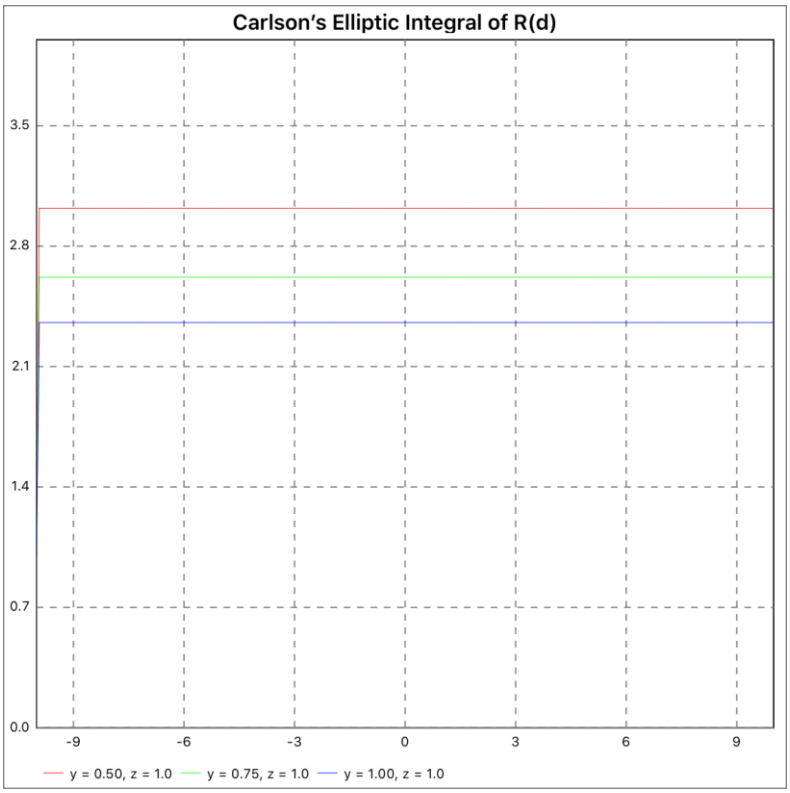

Figure 168 - Carlson's Elliptic Integral of R(d) Plot

6.24.60 Special.elliptic_int_rf

`special.elliptic_int_rf(x, y, z)` – Returns Carlson's *Elliptic Integral* of R_F of `x`, `y`, and `z`

The `x`, `y`, and `z` arguments are expressed as $\frac{1}{2}\int[(t+x)(t+y)(t+z)]^{-\frac{1}{2}}dt$ over an integral `0` to ∞.

Plot example:

```
xMin    = -10.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.elliptic_int_rf(x, 0.50, 1.0)
y2      = special.elliptic_int_rf(x, 0.75, 1.0)
y3      = special.elliptic_int_rf(x, 1.00, 1.0)

Utils.Graph3It("Carlson's Elliptic Integral of R(f)", \
               "Plt127_elliptic_int_rf",              \
               Plot.CHART.LINE,                       \
               x, y1, y2, y3,                         \
               "y = 0.50, z = 1.0",                   \
               "y = 0.75, z = 1.0",                   \
               "y = 1.00, z = 1.0",                   \
               xMin, xMax, 0.0, 4.0, 0, 0)
```

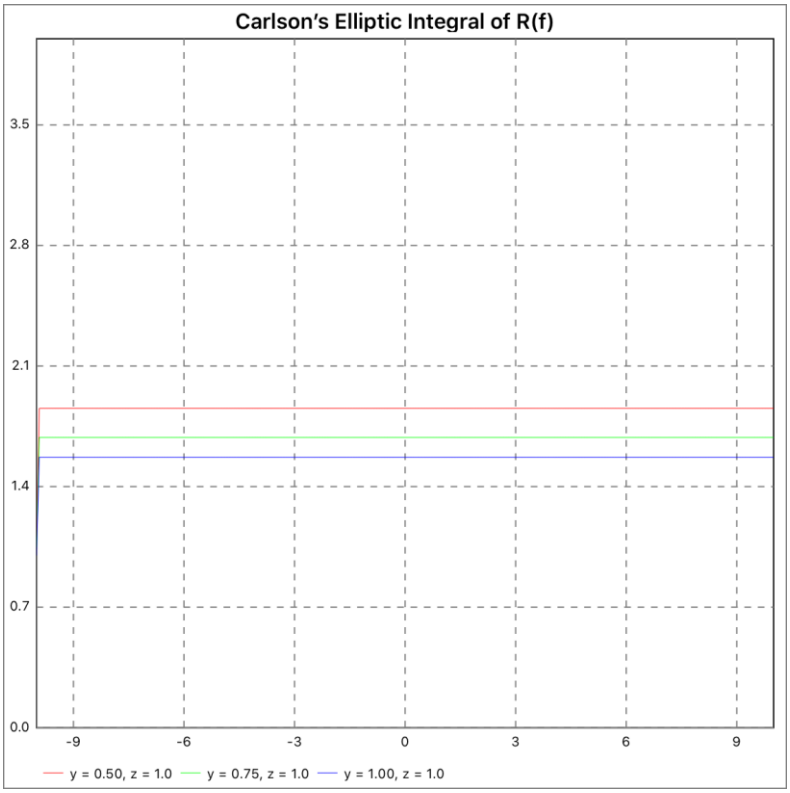

Figure 169 - Carlson's Elliptic Integral of R(f) Plot

`special.elliptic_int_rg(x, y, z)` – Returns Carlson's *Elliptic Integral* of R_G of `x`, `y`, and `z`

The `x`, `y` and `z` arguments are expressed as $¼\pi\int\int\sqrt{(x\cdot sin^2\theta cos^2\varphi + y\cdot sin^2\theta sin^2\varphi + z\cdot cos^2\theta)}\cdot sin\theta d\cdot\theta d\varphi$ over first integral `0` to 2π on second integral `0` to π.

Plot example:

```
xMin    = -10.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y1      = special.elliptic_int_rg(x, 0.50, 1.0)
y2      = special.elliptic_int_rg(x, 0.75, 1.0)
y3      = special.elliptic_int_rg(x, 1.00, 1.0)

Utils.Graph3It("Carlson's Elliptic Integral of R(g)", \
               "Plt128_elliptic_int_rg",              \
               Plot.CHART.LINE,                       \
               x, y1, y2, y3,                         \
               "y = 0.50, z = 1.0",                   \
               "y = 0.75, z = 1.0",                   \
               "y = 1.00, z = 1.0",                   \
               xMin, xMax, 0.0, 4.0, 0, 0)
```

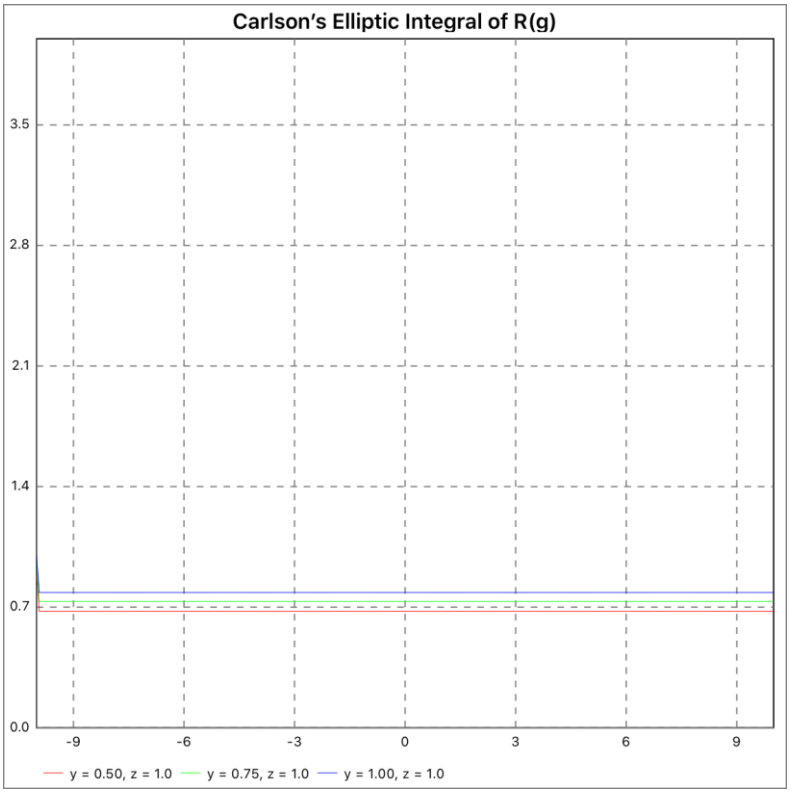

Figure 170 - Carlson's Elliptic Integral of R(g) Plot

6.24.61 Special.exp_int_en

`special.enp_int_en(x, n)` – Returns *Exponential Integral* E's `n` of `x`

The `n` and `x` argument are expressed as $\int e^{-xt}dt / t^n$ over an integral from `0` to ∞.

⇒ LINK: https://en.wikipedia.org/wiki/Exponential_integral

Plot example:

```
xMin    = 0.0
xMax    = 2.0

x       = Utils.PlotX(xMin, xMax)
y1      = Special.exp_int_en(x, 2)
y2      = Special.exp_int_en(x, 3)
y3      = Special.exp_int_en(x, 4)

Utils.Graph3It("Exponential Integral En", \
               "Plt129_exp_int_en",       \
               Plot.CHART.LINE,           \
               x, y1, y2, y3,             \
               "n = 2", "n = 3", "n = 4", \
               xMin, xMax, 0.0, 1.0, 0, 0)
```

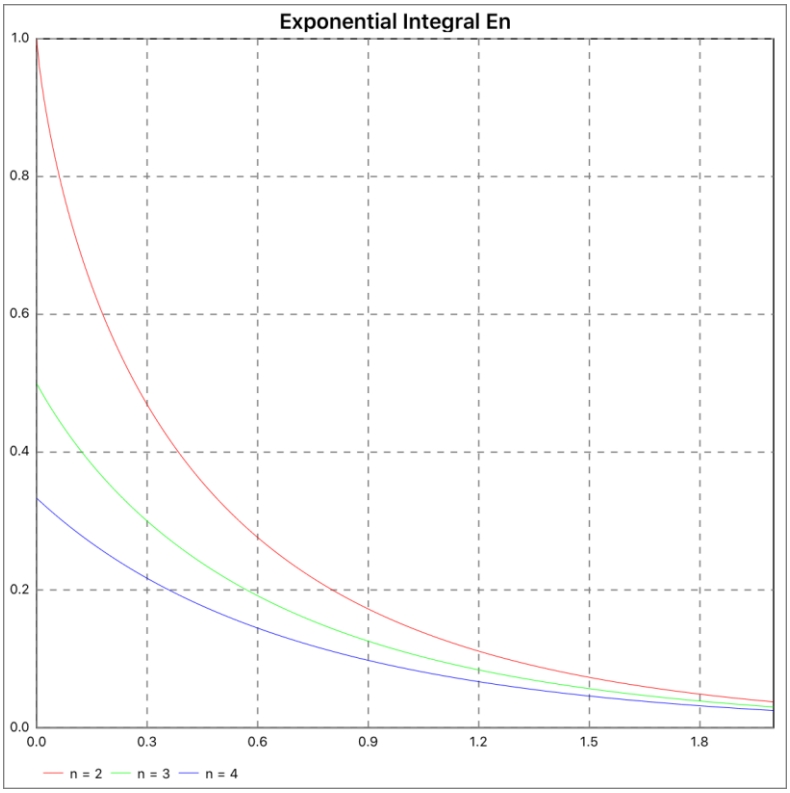

Figure 171 - Exponential Integral En Plot

6.24.62 Special.exp_int_ei

`special.exp_int_ei(x)` – Returns *Exponential Integral* of `x`

The `n` and `x` argument are expressed as $\int e^{-t}dt / t$ over an integral from `-x` to ∞.

Plot example:

```
xMin    = -3.0
xMax    = 4.0

x       = Utils.PlotX(xMin, xMax)
y       = Special.exp_int_ei(x)

Utils.PlotIt("Exponential Integral Ei", \
             "Plt130_exp_int_ei",       \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, -20.0, 20.0, 0, 0)
```

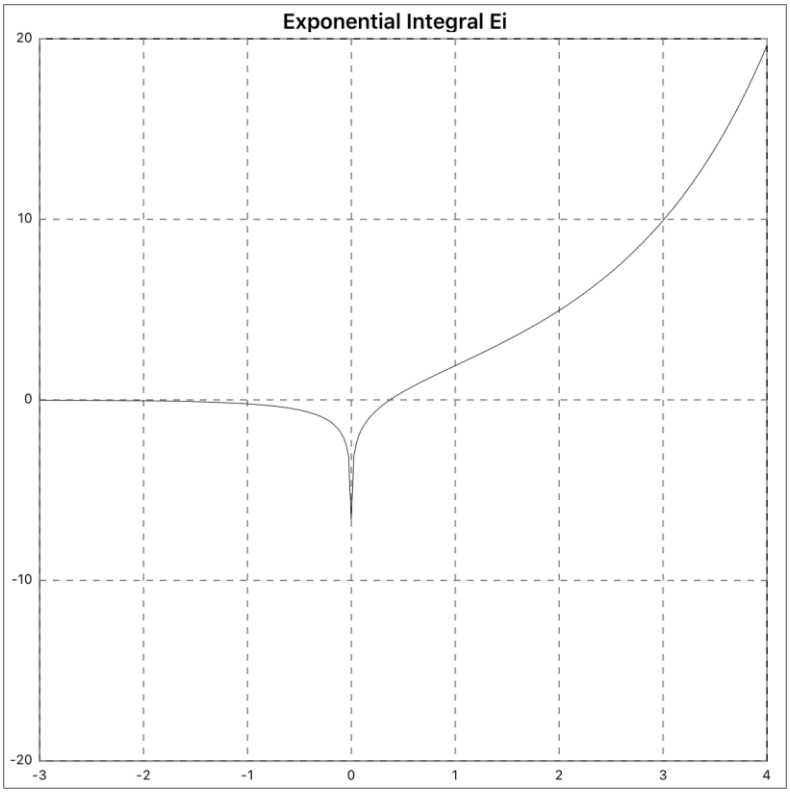

Figure 172 - Exponential Integral Ei Plot

6.24.63 Special.heuman_lambda

`special.heuman_lambda(x, p)` – Returns *Heuman Lambda* p of x

The `x` and `p` arguments are expressed as $2/\pi \cdot (((1-x^2)\sin\varphi\cos\varphi)/\Delta) \cdot (elliptic\_int\_rf(0,1-x^2,1) + elliptic\_int\_rj(0,1-x^2,1,x^2/\Delta^2))$. The range of x is $-1 \leq x \leq 1$, whereas $\Delta^2 = 1 - (1-x^2) \cdot \sin^2\varphi$.

Plot example:

```
xMin    = -1.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.heuman_lambda(x, 0.50)
y2      = special.heuman_lambda(x, 0.75)
y3      = special.heuman_lambda(x, 1.00)

Utils.Graph3It("Heuman Lambda",             \
               "Plt131_heuman_lambda",      \
               Plot.CHART.LINE,             \
               x, y1, y2, y3,               \
               "y = 0.50",                  \
               "y = 0.75",                  \
               "y = 1.00",                  \
               xMin, xMax, 0.0, 1.0, 0, 0)
```

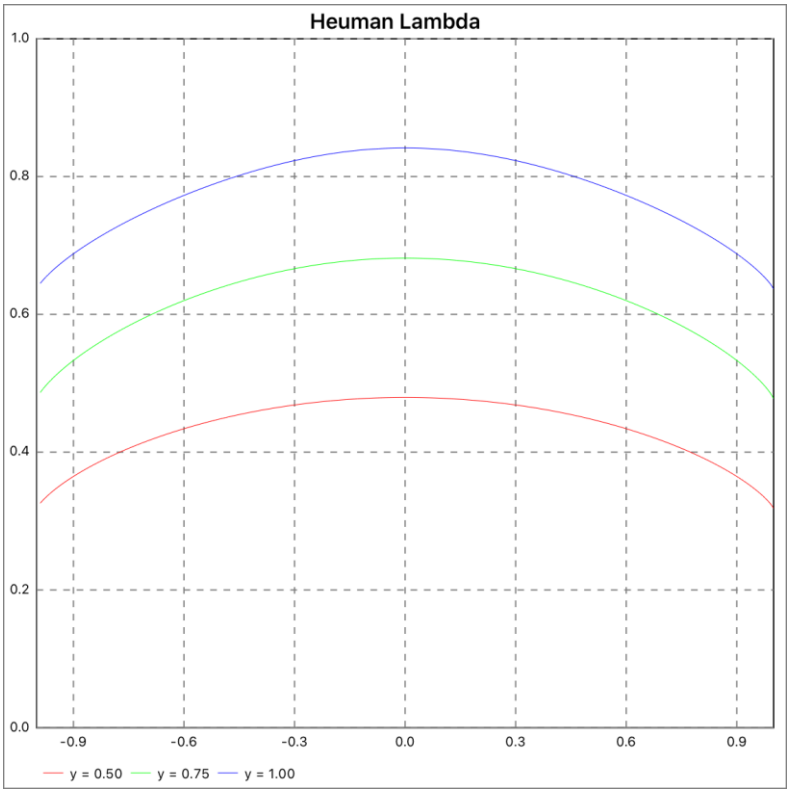

Figure 173 - Heuman Lambda Plot

6.24.64 Special.owens_t

`special.owens_t(x, h)` – Returns *Owen's T* of `h` and `x`

The `h` and `x` arguments are expressed as $\frac{1}{2\pi} \int exp\{-\frac{1}{2}h^2(1+x^2)\} / (1+x^2)dx$.

⇒ LINK: https://en.wikipedia.org/wiki/Owen%27s_T_function

Plot example:

```
xMin    = -1.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.owens_t(x, 0.50)
y2      = special.owens_t(x, 0.75)
y3      = special.owens_t(x, 1.00)

Utils.Graph3It("Owens's T",                 \
               "Plt132_owens_t",            \
               Plot.CHART.LINE,             \
               x, y1, y2, y3,               \
               "h = 0.50",                  \
               "h = 0.75",                  \
               "h = 1.00",                  \
               xMin, xMax, -0.2, 0.2, 0, 0)
```

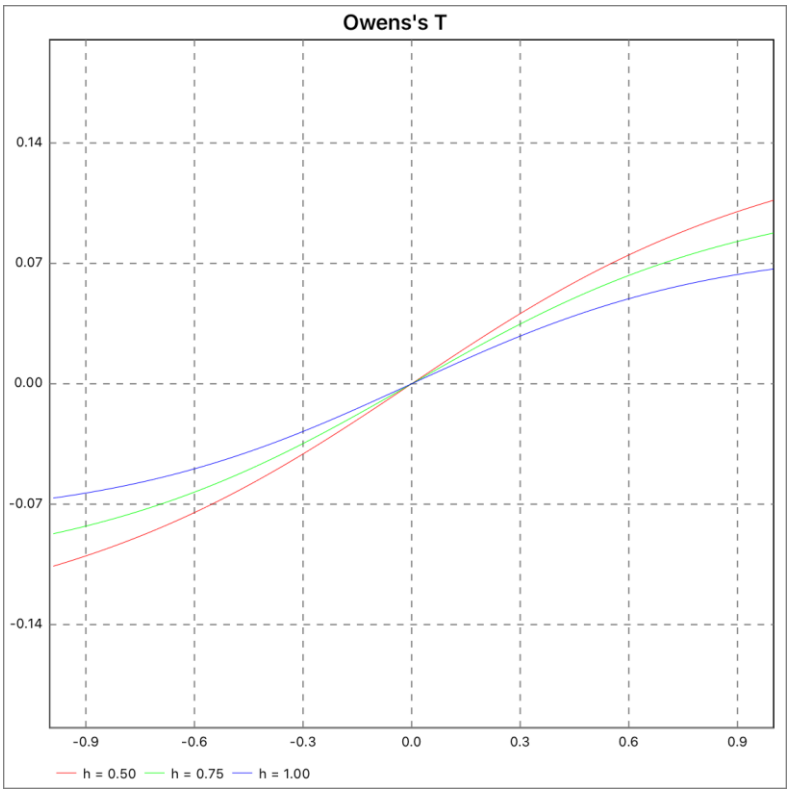

Figure 174 - Owen's T Plot

6.24.65 Special.zeta

`special.zeta(x)` – Returns *Riemann Zeta* value of `x`

The `x` argument is expressed as $\sum 1/k^x$ over summations from `1` to ∞.

⇒ LINK: https://en.wikipedia.org/wiki/Riemann_zeta_function

Plot example:

```
xMin    = -20.0
xMax    = 10.0

x       = Utils.PlotX(xMin, xMax)
y       = Special.zeta(x)

Utils.PlotIt("Zeta",                    \
             "Plt133_zeta",             \
             Plot.CHART.LINE,           \
             x, y,                      \
             xMin, xMax, -4.0, 40.0, 0, 0)
```

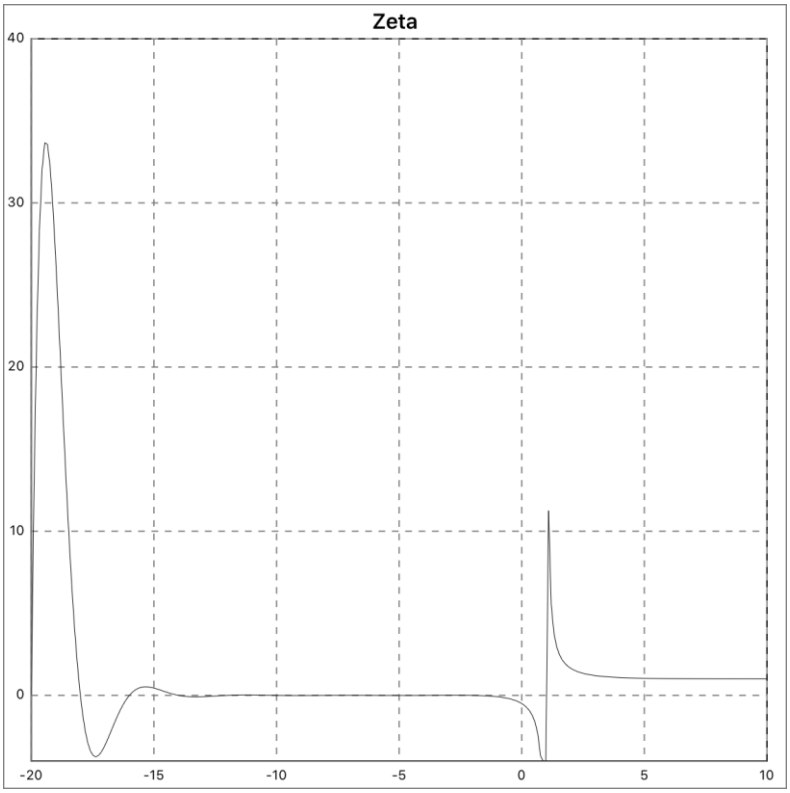

Figure 175 - Zeta Plot

6.24.66 Special.jacobi_zeta

`special.jacobi_zeta(x, p)` – Returns *Jacobi Zeta* `p` of x

The `x` and p arguments are expressed as $x^2/(3 \cdot bessel\_k(x)) \cdot sin\boldsymbol{\varphi}\, cos\boldsymbol{\varphi} \cdot \sqrt{(1-k^2sin^2\boldsymbol{\varphi})} \cdot elliptic\_int\_rj(0, 1-x^2,1,1-x^2sin^2\boldsymbol{\varphi})$. The range of x is `-1 ≤ x ≤ 1`.

⇒ LINK: https://en.wikipedia.org/wiki/Jacobi_zeta_function

Plot example:

```
xMin    = -1.0
xMax    = 1.0
inc     = 0.01

x       = Utils.PlotX(xMin + inc, xMax)
y1      = special.jacobi_zeta(x, 0.50)
y2      = special.jacobi_zeta(x, 0.75)
y3      = special.jacobi_zeta(x, 1.00)

Utils.Graph3It("Jacobi Zeta",                \
               "Plt134_jacobi_zeta",         \
               Plot.CHART.LINE,              \
               x, y1, y2, y3,                \
               "y = 0.50",                   \
               "y = 0.75",                   \
```

```
"y = 1.00",                    \
xMin, xMax, 0.0, 0.6, 0, 0)
```

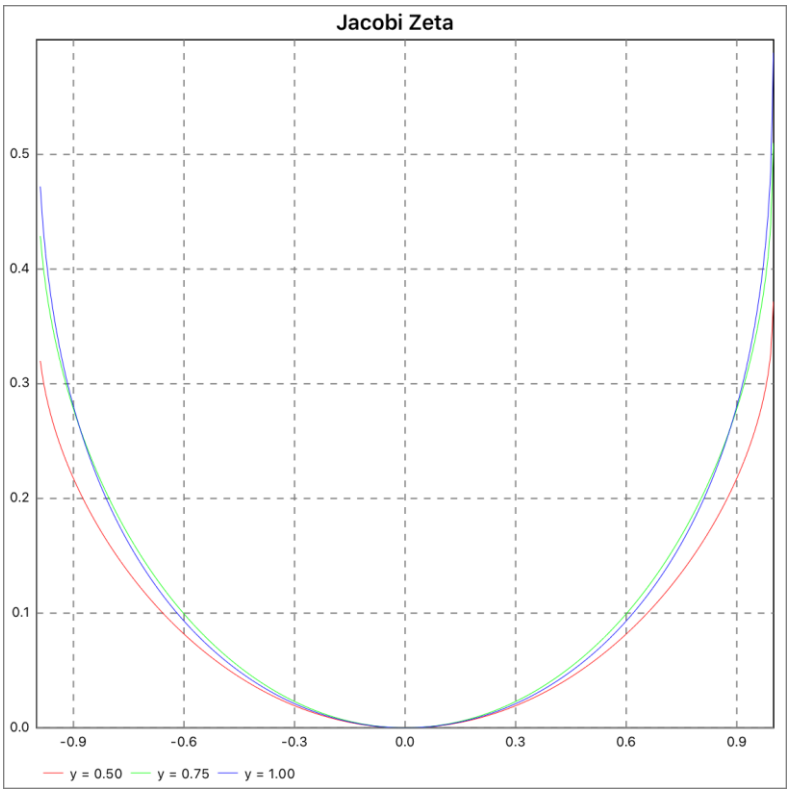

Figure 176 - Jacobi Zeta Plot

6.25 Dialog

The *Dialog* is created by `Create.dialog` function to display a script's dialog requesting input from the user to be used by the running script.

The following sub-section examples will use the following dialog variable:

```
d = Create.Dialog()

d.add(Dialog.CONTROL.LABEL,   "title",  "My Dog", Dialog.LABEL.CENTER)
d.add(Dialog.CONTROL.TEXT,    "name",   "Name",   "Penny")
d.add(Dialog.CONTROL.SLIDER,  "age",    "Age",    "1:15:1")
d.add(Dialog.CONTROL.BOOLEAN, "male",   "Male",   false)
d.add(Dialog.CONTROL.BUTTON,  "ok",     "OK",     "OK")
d.add(Dialog.CONTROL.BUTTON,  "cancel", "Cancel", "CANCEL")
```

6.25.1 Intrinsic

6.25.1.1 Functions

6.25.1.1.1 .add

`d.add(t, v, l, i)` – Adds dialog control

The `t` argument is the dialog control type displaying text label, text field, button, switch button, or slider using associated `Dialog.CONTROL.LABEL`, `Dialog.CONTROL.TEXT`, `Dialog.CONTROL.BUTTON`, `Dialog.CONTROL.BOOLEAN`, or `Dialog.CONTROL.SLIDER` constants, respectively.

The `v` argument is the dialog variable name associated with the control.

The `l` argument is the dialog's labeling of the control.

The `i` argument is initialization value of the control to be displayed in the dialog.

- The `Dialog.CONTROL.LABEL`'s argument accepts one of the following `Dialog.LABEL.LEFT`, `Dialog.LABEL.CENTER`, or `Dialog.LABEL.RIGHT` constants.
- If the `Dialog.CONTROL.TEXT`'s argument contains a `*` character, the text input is displayed as secured character inputs like typing in a password.
- The `Dialog.CONTROL.BUTTON`'s argument accepts `true` or `false` values.
- The `Dialog.CONTROL.SLIDER`'s argument format is `min:max:increment`, whereas `min` is the minimum value of the slider, `max` is the maximum value, and `inc` is optional increment value of incrementing values of the slider's control. If the `inc` value is not specified, the default value is `1`.

6.25.1.1.2 .prompt

`d.prompt()` – Displays dialog in the app

For example:

```
r = d.prompt()
if (r == "OK")
   print "name: ", d.value("name")
   print "age:  ", d.value("age")
   print "male: ", d.value("male")
end
```

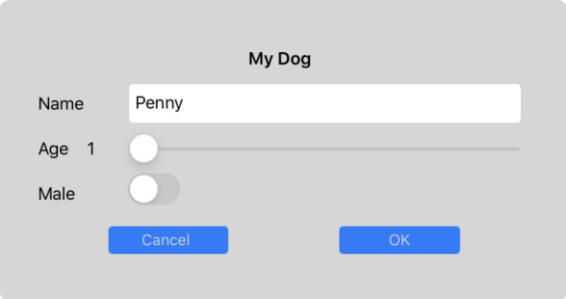

Figure 177 - User's Defined Dialog

6.25.1.1.3 .value

`d.value(v)` – Returns the dialog's variable resultant value

The `v` argument is the dialog's variable name used to return its resultant value supplied by the user.

For example:

- `d.value("name")` Returns user's input of the dog's `name`
- `d.value("age")` Returns user's input of the dog's `age`
- `d.value("male")` Returns user's input of the dog's `sex`

6.25.1.1.4 .clear

`d.clear()` – Clears all the added items to the dialog from previous `add` intrinsic function calls

6.25.1.2 Fields

6.25.1.2.1 .size

`d.size` – Returns number of controls in the dialog

For example:

- `d.size` Returns `6`

6.25.1.2.2 .info

`d.info` - Returns textual information about the dialog

6.25.2 Constants

6.25.2.1 Dialog.CONTROL.LABEL

`Dialog.CONTROL.LABEL` – Displays text label control constant

6.25.2.2 Dialog.CONTROL.TEXT

`Dialog.CONTROL.TEXT` – Displays text field control constant

6.25.2.3 Dialog.CONTROL.BUTTON

`Dialog.CONTROL.BUTTON` – Displays button control constant

6.25.2.4 Dialog.CONTROL.BOOLEAN

`Dialog.CONTROL.BOOLEAN` – Displays boolean's switch on/off control constant

6.25.2.5 Dialog.CONTROL.SLIDER

`Dialog.CONTROL.SLIDER` – Displays slider control constant

6.25.2.6 Dialog.LABEL.LEFT

`Dialog.LABEL.LEFT` – Left-justifies the label display constant

6.25.2.7 Dialog.LABEL.CENTER

`Dialog.LABEL.CENTER` – Center-justifies the label display constant

6.25.2.8 Dialog.LABEL.RIGHT

`Dialog.LABEL.RIGHT` – Right-justifies the label display constant

6.26 UI

The *User Interface* (UI) functions allow execution of a script to interact with the app during its execution.

6.26.1 UI.tab

`ui.tab(t)` – Selects app's tab panel

The `t` argument is the app's tab selection of *Calculator*, *Scripts*, *Graphs*, or *Outputs* tabs using `UI.CALULCATOR`, `UI.SCRIPTS`, `UI.GRAPHS`, or `UI.OUTPUTS` constants, respectively.

For example:

- `ui.tab(UI.CALULCATOR)` Switches app's display to *Calculator* tab panel
- `ui.tab(UI.GRAPHS)` Switches app's display to *Graphs* tab panel

6.26.2 UI.command

`ui.command(c)` – Execute UI command

The `c` argument is UI's command used to execute in the app.

⇒ NOTE: There is currently only one UI command available: `UI.CLEAR`. The `UI.CLEAR` command clears the logging outputs in the *Outputs* tab.

For example:

- `ui.command(UI.CLEAR)` Clears *Outputs* tab logging outputs

6.26.3 UI.config

`ui.config(n, v)` – Modifies UI's configuration parameters

The `n` argument is name of the configuration parameter and `v` argument is the value used to modify its setting. Refer to ***Quick Tour***'s description of the *Preferences* dialog for some of the equivalent mappings to the `ui.config`'s function.

For example:

- `ui.config(UI.CONFIG.AUTO_RECALC, true)` Configures setting automatic recalculation of expressions containing changed variables
- `ui.config(UI.CONFIG.COLORIZE_SCRIPT, false)` Configures settings of the script's text to be colorized to highlight different parts of the text's syntax
- `ui.config(UI.CONFIG.AUTO_INDENTS, false)` Configures settings of text's *`()`*, *`[]`*, *`{}`*, `""`, or `''` insertion of the matching character when entering expression to calculate in the ***Calculator*** tab or the script's text in the ***Scripts*** tab
- `ui.config(UI.CONFIG.MAX_EXPRS, 20)` Configures settings of the maximum number of expressions displayed in the ***Calculator*** tab before rolling off the *Expressions* column display
- `ui.config(UI.CONFIG.MAX_MEMORY, 1024)` Configures settings of the maximum amount of memory that can be used by the projects and running the scripts
- `ui.config(UI.CONFIG.MAX_OUTPUTS, 32)` Configures settings of the maximum amount of memory that can be used to display in the ***Outputs*** tab before rolling off oldest outputted messages
- `ui.config(UI.CONFIG.KEYBOARD, true)` Configures settings to use app's built-in keyboard or standard system keyboard
- `ui.config(UI.CONFIG.KEYBOARD_CLICKS, true)` Configures settings to generate a clicking sound on keyboard entries

• `ui.config(UI.CONFIG.CLASSIC_KEYS, false)`	Configures settings to use `*` and `/` characters rather than × and ÷ characters in the built-in keyboard
• `ui.config(UI.CONFIG.DOT_INTEGER, false)`	Configures outputting integer values using period character for every three digits for readability purposes

6.26.4 UI.askPassword

`Ui.askPassword(m)` – Asks the user for a password

The function will prompt a password dialog to return the password's string. If the string is empty, assume the user has canceled the prompt's dialog without supplying a password.

6.26.5 UI.isDarkMode

`ui.isDarkMode()` – Returns app's dark mode state

The function will return `true` if the app is in dark mode; otherwise, it will return `false`.

6.26.6 Constants

6.26.6.1 UI.CALCULATOR

`UI.CALCULATOR` – App's *Calculator* tab constant

6.26.6.2 UI.SCRIPTS

`UI.SCRIPTS` – App's *Scripts*Calculator tab constant

6.26.6.3 UI.GRAPHS

`UI.GRAPHS` – App's *Graphs* tab constant

6.26.6.4 UI.OUTPUTS

`UI.OUTPUTS` – App's *Outputs* tab constant

6.26.6.5 UI.CLEAR

`UI.CLEAR` – App's command to clear outputted logging in *Outputs* tab

6.26.6.6 UI.CONFIG.AUTO_INDENTS

`UI.CONFIG.AUTO_INDENTS` – Configures automatic script indentations constant

6.26.6.7 UI.CONFIG.AUTO_INSERTS

`UI.CONFIG.AUTO_INSERTS` – Configures automatic character insertion constant

6.26.6.8 UI.CONFIG.AUTO_RECALC

`UI.CONFIG.AUTO_RECALC` – Configures automatic re-evaluate expressions constant

6.26.6.9 UI.CONFIG.CLASSIC_KEYS

`UI.CONFIG.CLASSIC_KEYS` – Configures using `*` and `/` classic characters constant

6.26.6.10 UI.CONFIG.COLORIZE_SCRIPT

`UI.CONFIG.COLORIZE_SCRIPT` – Configures colorizing script's text constant

6.26.6.11 UI.CONFIG.DOT_INTEGER

`UI.CONFIG.DOT_INTEGER` – Configures outputting integer with period character for every 3 digits constant

6.26.6.12 UI.CONFIG.KEYBOARD

`UI.CONFIG.KEYBOARD` – Configures using built-in app's keyboard constant

6.26.6.13 UI.CONFIG.KEYBOARD_CLICKS

`UI.CONFIG.KEYBOARD_CLICKS` – Configures generated clicking sound on keyboard entry constant

6.26.6.14 UI.CONFIG.MAX_EXPRS

`UI.CONFIG.MAX_EXPRS` – Configures maximum number of expressions displayed constant

6.26.6.15 UI.CONFIG.MAX_MEMORY

`UI.CONFIG.MAX_MEMORY` – Configures script's execution maximum memory constant

6.26.6.16 UI.CONFIG.MAX_OUTPUTS

`UI.CONFIG.MAX_OUTPUTS` – Configures maximum bytes displayed in the *Outputs* tab constant

6.27 Unit

The following table consists of all the unit conversion functions used to convert x argument's floating-point value to the equivalent unit's converted value. The input and output floating-point values are unitless. Meaning, if passed `1{ft}` to the `Unit.in2ft` function it will pass the floating-point value of `1` that will be converted to `1/12` of a foot.

For example:

- `Unit.in2ft(12)` Returns `1`

Category	Function	Description
Linear	`Unit.in2ft(x)`	inch to feet
	`Unit.in2kft(x)`	inch to kilo-feet
	`Unit.in2yd(x)`	inch to yard
	`Unit.in2mi(x)`	inch to mile
	`Unit.in2rod(x)`	inch to rod
	`Unit.in2mm(x)`	inch to millimeter
	`Unit.in2cm(x)`	inch to centimeter
	`Unit.in2dm(x)`	inch to decimeter
	`Unit.in2m(x)`	inch to meter
	`Unit.in2km(x)`	inch to kilometer
	`Unit.in2nmi(x)`	inch to nautical mile
	`Unit.ft2in(x)`	feet to inch
	`Unit.ft2kft(x)`	feet to kilo-feet
	`Unit.ft2yd(x)`	feet to yard
	`Unit.ft2mi(x)`	feet to mile
	`Unit.ft2rod(x)`	feet to rod
	`Unit.ft2mm(x)`	feet to millimeter
	`Unit.ft2cm(x)`	feet to centimeter
	`Unit.ft2dm(x)`	feet to decimeter
	`Unit.ft2m(x)`	feet to meter
	`Unit.ft2km(x)`	feet to kilometer
	`Unit.ft2nmi(x)`	feet to nautical mile
	`Unit.kft2in(x)`	kilo-feet to inch
	`Unit.kft2ft(x)`	kilo-feet to feet
	`Unit.kft2yd(x)`	kilo-feet to yard
	`Unit.kft2mi(x)`	kilo-feet to mile
	`Unit.kft2rod(x)`	kilo-feet to rod
	`Unit.kft2mm(x)`	kilo-feet to millimeter
	`Unit.kft2cm(x)`	kilo-feet to centimeter
	`Unit.kft2dm(x)`	kilo-feet to decimeter
	`Unit.kft2m(x)`	kilo-feet to meter
	`Unit.kft2km(x)`	kilo-feet to kilometer
	`Unit.kft2nmi(x)`	kilo-feet to nautical mile
	`Unit.yd2in(x)`	yard to inch
	`Unit.yd2ft(x)`	yard to feet
	`Unit.yd2kft(x)`	yard to kilo-feet

Category	Function	Description
	`Unit.yd2mi(x)`	yard to mile
	`Unit.yd2rod(x)`	yard to rod
	`Unit.yd2mm(x)`	yard to millimeter
	`Unit.yd2cm(x)`	yard to centimeter
	`Unit.yd2dm(x)`	yard to decimeter
	`Unit.yd2m(x)`	yard to meter
	`Unit.yd2km(x)`	yard to kilometer
	`Unit.yd2nmi(x)`	yard to nautical mile
	`Unit.mi2in(x)`	mile to inch
	`Unit.mi2ft(x)`	mile to feet
	`Unit.mi2kft(x)`	mile to kilo-feet
	`Unit.mi2yd(x)`	mile to yard
	`Unit.mi2rod(x)`	mile to rod
	`Unit.mi2mm(x)`	mile to millimeter
	`Unit.mi2cm(x)`	mile to centimeter
	`Unit.mi2dm(x)`	mile to decimeter
	`Unit.mi2m(x)`	mile to meter
	`Unit.mi2km(x)`	mile to kilometer
	`Unit.mi2nmi(x)`	mile to nautical mile
	`Unit.rod2in(x)`	rod to inch
	`Unit.rod2ft(x)`	rod to feet
	`Unit.rod2kft(x)`	rod to kilo-feet
	`Unit.rod2yd(x)`	rod to yard
	`Unit.rod2mi(x)`	rod to mile
	`Unit.rod2mm(x)`	rod to milli-meter
	`Unit.rod2cm(x)`	rod to centi-meter
	`Unit.rod2dm(x)`	rod to decimeter
	`Unit.rod2m(x)`	rod to meter
	`Unit.rod2km(x)`	rod to kilometer
	`Unit.rod2nmi(x)`	rod to nautical mile
	`Unit.mm2in(x)`	milli-meter to inch
	`Unit.mm2ft(x)`	milli-meter to feet
	`Unit.mm2kft(x)`	milli-meter to kilo-feet
	`Unit.mm2yd(x)`	milli-meter to yard
	`Unit.mm2mi(x)`	milli-meter to mile
	`Unit.mm2rod(x)`	milli-meter to rod
	`Unit.mm2cm(x)`	milli-meter to centimeter
	`Unit.mm2dm(x)`	milli-meter to decimeter
	`Unit.mm2m(x)`	milli-meter to meter
	`Unit.mm2km(x)`	milli-meter to kilometer

Category	Function	Description
	`Unit.mm2nmi(x)`	milli-meter to nautical mile
	`Unit.cm2in(x)`	centi-meter to inch
	`Unit.cm2ft(x)`	centi-meter to feet
	`Unit.cm2kft(x)`	centi-meter to kilo-feet
	`Unit.cm2yd(x)`	centimeter to yard
	`Unit.cm2mi(x)`	centimeter to mile
	`Unit.cm2rod(x)`	centimeter to rod
	`Unit.cm2mm(x)`	centimeter to millimeter
	`Unit.cm2dm(x)`	centimeter to decimeter
	`Unit.cm2m(x)`	centimeter to meter
	`Unit.cm2km(x)`	centimeter to kilometer
	`Unit.cm2nmi(x)`	centimeter to nautical mile
	`Unit.dm2in(x)`	decimeter to inch
	`Unit.dm2ft(x)`	decimeter to feet
	`Unit.dm2kft(x)`	decimeter to kilo-feet
	`Unit.dm2yd(x)`	decimeter to yard
	`Unit.dm2mi(x)`	decimeter to mile
	`Unit.dm2rod(x)`	decimeter to rod
	`Unit.dm2mm(x)`	decimeter to millimeter
	`Unit.dm2cm(x)`	decimeter to centimeter
	`Unit.dm2m(x)`	decimeter to meter
	`Unit.dm2km(x)`	decimeter to kilometer
	`Unit.dm2nmi(x)`	decimeter to nautical mile
	`Unit.m2in(x)`	meter to inch
	`Unit.m2ft(x)`	meter to feet
	`Unit.m2kft(x)`	meter to kilo-feet
	`Unit.m2yd(x)`	meter to yard
	`Unit.m2mi(x)`	meter to mile
	`Unit.m2rod(x)`	meter to rod
	`Unit.m2mm(x)`	meter to millimeter
	`Unit.m2cm(x)`	meter to centimeter
	`Unit.m2dm(x)`	meter to decimeter
	`Unit.m2km(x)`	meter to kilometer
	`Unit.m2nmi(x)`	meter to nautical mile
	`Unit.km2in(x)`	kilometer to inch
	`Unit.km2ft(x)`	kilometer to feet
	`Unit.km2kft(x)`	kilometer to kilo-feet
	`Unit.km2yd(x)`	kilometer to yard
	`Unit.km2mi(x)`	kilometer to mile
	`Unit.km2rod(x)`	kilometer to rod

Category	Function	Description
	`Unit.km2mm(x)`	kilometer to millimeter
	`Unit.km2cm(x)`	kilometer to centimeter
	`Unit.km2dm(x)`	kilometer to decimeter
	`Unit.km2m(x)`	kilometer to meter
	`Unit.km2nmi(x)`	kilometer to nautical mile
	`Unit.nmi2in(x)`	nautical mile to inch
	`Unit.nmi2ft(x)`	nautical mile to feet
	`Unit.nmi2kft(x)`	nautical mile to kilo-feet
	`Unit.nmi2yd(x)`	nautical mile to yard
	`Unit.nmi2mi(x)`	nautical mile to mile
	`Unit.nmi2rod(x)`	nautical mile to rod
	`Unit.nmi2mm(x)`	nautical mile to millimeter
	`Unit.nmi2cm(x)`	nautical mile to centimeter
	`Unit.nmi2dm(x)`	nautical mile to decimeter
	`Unit.nmi2m(x)`	nautical mile to meter
	`Unit.nmi2km(x)`	nautical mile to kilometer
Time	`Unit.ps2ns(x)`	pico-second to nanosecond
	`Unit.ps2ms(x)`	pico-second to millisecond
	`Unit.ps2s(x)`	pico-second to second
	`Unit.ps2min(x)`	pico-second to minute
	`Unit.ps2h(x)`	pico-second to hour
	`Unit.ps2da(x)`	pico-second to day
	`Unit.ps2wk(x)`	pico-second to week
	`Unit.ps2mn(x)`	pico-second to month
	`Unit.ps2yr(x)`	pico-second to year
	`Unit.ns2ps(x)`	nano-second to picosecond
	`Unit.ns2ms(x)`	nano-second to millisecond
	`Unit.ns2s(x)`	nano-second to second
	`Unit.ns2min(x)`	nano-second to minute
	`Unit.ns2h(x)`	nano-second to hour
	`Unit.ns2da(x)`	nano-second to day
	`Unit.ns2wk(x)`	nano-second to week
	`Unit.ns2mn(x)`	nano-second to month
	`Unit.ns2yr(x)`	nano-second to year
	`Unit.ms2ps(x)`	millisecond to picosecond
	`Unit.ms2ns(x)`	millisecond to nanosecond
	`Unit.ms2s(x)`	millisecond to second
	`Unit.ms2min(x)`	millisecond to minute
	`Unit.ms2h(x)`	millisecond to hour
	`Unit.ms2da(x)`	millisecond to day

Category	Function	Description
	`Unit.ms2wk(x)`	millisecond to week
	`Unit.ms2mn(x)`	millisecond to month
	`Unit.ms2yr(x)`	millisecond to year
	`Unit.s2ps(x)`	second to picosecond
	`Unit.s2ns(x)`	second to nanosecond
	`Unit.s2ms(x)`	second to millisecond
	`Unit.s2min(x)`	second to minute
	`Unit.s2h(x)`	second to hour
	`Unit.s2da(x)`	second to day
	`Unit.s2wk(x)`	second to week
	`Unit.s2mn(x)`	second to month
	`Unit.s2yr(x)`	second to year
	`Unit.min2ps(x)`	minute to picosecond
	`Unit.min2ns(x)`	minute to nanosecond
	`Unit.min2ms(x)`	minute to millisecond
	`Unit.min2s(x)`	minute to second
	`Unit.min2h(x)`	minute to hour
	`Unit.min2da(x)`	minute to day
	`Unit.min2wk(x)`	minute to week
	`Unit.min2mn(x)`	minute to month
	`Unit.min2yr(x)`	minute to year
	`Unit.h2ps(x)`	hour to picosecond
	`Unit.h2ns(x)`	hour to nanosecond
	`Unit.h2ms(x)`	hour to millisecond
	`Unit.h2s(x)`	hour to second
	`Unit.h2min(x)`	hour to minute
	`Unit.h2da(x)`	hour to day
	`Unit.h2wk(x)`	hour to week
	`Unit.h2mn(x)`	hour to month
	`Unit.h2yr(x)`	hour to year
	`Unit.da2ps(x)`	day to pico-second
	`Unit.da2ns(x)`	day to nanosecond
	`Unit.da2ms(x)`	day to millisecond
	`Unit.da2s(x)`	day to second
	`Unit.da2min(x)`	day to minute
	`Unit.da2h(x)`	day to hour
	`Unit.da2wk(x)`	day to week
	`Unit.da2mn(x)`	day to month
	`Unit.da2yr(x)`	day to year
	`Unit.wk2ps(x)`	week to pico-second

Category	Function	Description
	`Unit.wk2ns(x)`	week to nanosecond
	`Unit.wk2ms(x)`	week to millisecond
	`Unit.wk2s(x)`	week to second
	`Unit.wk2min(x)`	week to minute
	`Unit.wk2h(x)`	week to hour
	`Unit.wk2da(x)`	week to day
	`Unit.wk2mn(x)`	week to month
	`Unit.wk2yr(x)`	week to year
	`Unit.mn2ps(x)`	month to pico-second
	`Unit.mn2ns(x)`	month to nanosecond
	`Unit.mn2ms(x)`	month to millisecond
	`Unit.mn2s(x)`	month to second
	`Unit.mn2min(x)`	month to minute
	`Unit.mn2h(x)`	month to hour
	`Unit.mn2da(x)`	month to day
	`Unit.mn2wk(x)`	month to week
	`Unit.mn2yr(x)`	month to year
	`Unit.yr2ps(x)`	year to pico-second
	`Unit.yr2ns(x)`	year to nanosecond
	`Unit.yr2ms(x)`	year to millisecond
	`Unit.yr2s(x)`	year to second
	`Unit.yr2min(x)`	year to minute
	`Unit.yr2h(x)`	year to hour
	`Unit.yr2da(x)`	year to day
	`Unit.yr2wk(x)`	year to week
	`Unit.yr2mn(x)`	year to month
Frequency	`Unit.ghz2mhz(x)`	gigahertz to megahertz
	`Unit.ghz2khz(x)`	gigahertz to kilohertz
	`Unit.ghz2hz(x)`	gigahertz to hertz
	`Unit.mhz2ghz(x)`	megahertz to gigahertz
	`Unit.mhz2khz(x)`	megahertz to kilohertz
	`Unit.mhz2hz(x)`	megahertz to hertz
	`Unit.khz2ghz(x)`	kilohertz to gigahertz
	`Unit.khz2mhz(x)`	kilohertz to megahertz
	`Unit.khz2hz(x)`	kilohertz to hertz
	`Unit.hz2ghz(x)`	hertz to gigahertz
	`Unit.hz2mhz(x)`	hertz to megahertz
	`Unit.hz2khz(x)`	hertz to kilohertz
Power	`Unit.w2mw(x)`	watt to milliwatt
	`Unit.w2kw(x)`	watt to kilowatt

Category	Function	Description
	`Unit.w2uw(x)`	watt to microwatt
	`Unit.mw2w(x)`	milliwatt to watt
	`Unit.mw2kw(x)`	milliwatt to kilowatt
	`Unit.mw2uw(x)`	milliwatt to microwatt
	`Unit.kw2w(x)`	kilowatt to watt
	`Unit.kw2mw(x)`	kilowatt to milliwatt
	`Unit.kw2uw(x)`	kilowatt to microwatt
	`Unit.uw2w(x)`	microwatt to watt
	`Unit.uw2mw(x)`	microwatt to milliwatt
	`Unit.uw2kw(x)`	microwatt to kilowatt
Voltage	`Unit.v2mv(x)`	volt to millivolt
	`Unit.v2kv(x)`	volt to kilovolt
	`Unit.v2uv(x)`	volt to microvolt
	`Unit.mv2v(x)`	millivolt to volt
	`Unit.mv2kv(x)`	millivolt to kilovolt
	`Unit.mv2uv(x)`	millivolt to microvolt
	`Unit.kv2v(x)`	kilovolt to volt
	`Unit.kv2mv(x)`	kilovolt to millivolt
	`Unit.kv2uv(x)`	kilovolt to microvolt
	`Unit.uv2v(x)`	microvolt to volt
	`Unit.uv2mv(x)`	microvolt to millivolt
	`Unit.uv2kv(x)`	microvolt to kilovolt
Amperage	`Unit.i2mia(x)`	amperage to milli-amperage
	`Unit.i2ki(x)`	amperage to kilo-amperage
	`Unit.i2ui(x)`	amperage to micro-amperage
	`Unit.mia2i`	milli-amperage to amperage
	`Unit.mia2ki(x)`	milli-amperage to kilo-amperage
	`Unit.mia2ui(x)`	milli-amperage to micro-amperage
	`Unit.ki2i(x)`	kilo-amperage to amperage
	`Unit.ki2mia(x)`	kilo-amperage to milli-amperage
	`Unit.ki2ui(x)`	kilo-amperage to micro-amperage
	`Unit.ui2i(x)`	micro-amperage to amperage
	`Unit.ui2mia(x)`	micro-amperage to milli-amperage
	`Unit.ui2ki(x)`	micro-amperage to kilo-amperage
Farad	`Unit.fa2mfa(x)`	farad to milli-farad
	`Unit.fa2kfa(x)`	farad to kilo-farad
	`Unit.fa2ufa(x)`	farad to micro-farad
	`Unit.fa2pfa(x)`	farad to pico-farad
	`Unit.mfa2fa(x)`	milli-farad to farad
	`Unit.mfa2kfa(x)`	milli-farad to kilo-farad

Category	Function	Description
	`Unit.mfa2ufa(x)`	milli-farad to micro-farad
	`Unit.mfa2pfa(x)`	milli-farad to pico-farad
	`Unit.kfa2fa(x)`	kilo-farad to farad
	`Unit.kfa2mfa(x)`	kilo-farad to milli-farad
	`Unit.kfa2ufa(x)`	kilo-farad to micro-farad
	`Unit.kfa2pfa(x)`	kilo-farad to pico-farad
	`Unit.ufa2fa(x)`	micro-farad to farad
	`Unit.ufa2mfa(x)`	micro-farad to milli-farad
	`Unit.ufa2kfa(x)`	micro-farad to kilo-farad
	`Unit.ufa2pfa(x)`	micro-farad to pico-farad
	`Unit.pfa2fa(x)`	pico-farad to farad
	`Unit.pfa2mfa(x)`	pico-farad to milli-farad
	`Unit.pfa2kfa(x)`	pico-farad to kilo-farad
	`Unit.pfa2ufa(x)`	pico-farad to micro-farad
Energy	`Unit.erg2j(x)`	energy to joule
	`Unit.erg2ev(x)`	energy to electron volt
	`Unit.erg2kwh(x)`	energy to kilowatt hour
	`Unit.j2erg(x)`	joule to energy
	`Unit.j2ev(x)`	joule to electron volt
	`Unit.j2kwh(x)`	joule to kilowatt hour
	`Unit.ev2erg(x)`	electron volt to energy
	`Unit.ev2j(x)`	electron volt to joule
	`Unit.ev2kwh(x)`	electron volt to kilowatt hour
	`Unit.kwh2erg(x)`	kilowatt hour to energy
	`Unit.kwh2j(x)`	kilowatt hour to joule
	`Unit.kwh2ev(x)`	kilowatt hour to electron volt
Temperature	`Unit.k2c(x)`	kelvin to celsius
	`Unit.k2f(x)`	kelvin to fahrenheit
	`Unit.c2k(x)`	celsius to kelvin
	`Unit.c2f(x)`	celsius to fahrenheit
	`Unit.f2k(x)`	fahrenheit to kelvin
	`Unit.f2c(x)`	fahrenheit to celsius
Angle	`Unit.d2r(x)`	degree to radian
	`Unit.d2mr(x)`	degree to milliradian
	`Unit.d2gr(x)`	degree to gradient
	`Unit.r2d(x)`	radian to degree
	`Unit.r2mr(x)`	radian to milliradian
	`Unit.r2gr(x)`	radian to gradient
	`Unit.mr2d(x)`	milliradian to degree
	`Unit.mr2r(x)`	milliradian to radian

Category	Function	Description
	`Unit.mr2gr(x)`	milliradian to gradient
	`Unit.gr2d(x)`	gradient to degree
	`Unit.gr2r(x)`	gradient to radian
	`Unit.gr2mr(x)`	gradient to milliradian
Weight	`Unit.oz2lb(x)`	ounce to pound
	`Unit.oz2mg(x)`	ounce to milligram
	`Unit.oz2g(x)`	ounce to gram
	`Unit.oz2kg(x)`	ounce to kilogram
	`Unit.lb2oz(x)`	pound to ounce
	`Unit.lb2mg(x)`	pound to milligram
	`Unit.lb2g(x)`	pound to gram
	`Unit.lb2kg(x)`	pound to kilogram
	`Unit.mg2oz(x)`	milligram to ounce
	`Unit.mg2lb(x)`	milligram to pound
	`Unit.mg2g(x)`	milligram to gram
	`Unit.mg2kg(x)`	milligram to kilogram
	`Unit.g2oz(x)`	gram to ounce
	`Unit.g2lb(x)`	gram to pound
	`Unit.g2mg(x)`	gram to milligram
	`Unit.g2kg(x)`	gram to kilogram
	`Unit.kg2oz(x)`	kilogram to ounce
	`Unit.kg2lb(x)`	kilogram to pound
	`Unit.kg2mg(x)`	kilogram to milligram
	`Unit.kg2g(x)`	kilogram to gram
Liquid	`Unit.l2ga(x)`	liter to gallon
	`Unit.ga2l(x)`	gallon to liter

Table 27 - Unit Conversions

6.28 Miscellaneous

6.28.1 Memory.budget

`memory.budget()` – Returns total amount of memory in bytes available in the device

6.28.2 Memory.current

`memory.current()` – Returns approximate amount of memory in bytes currently used by the app

6.28.3 Memory.Maximum

`memory.maximum()` – Returns maximum amount of memory in bytes allowed by the app

Refer to additional details in setting this value in the *Preferences* dialog from the *Configure* button as described in *Menu Bar* sub-section.

7 Technical Notes

7.1 Download User Manual

Download this user's manual at https://zynergyapps.com/docs/CalcPro-0.0.0.pdf.

7.2 Boost Library

This app uses *Boost's Math library 2.6.0*'s classes and methods called from the *Distributions* and *Special* library functions. The following table shows a mapping of the app's function to the Boost's class or method. For your reading pleasure, please refer to the Boost's document for detailed explanation of these classes or methods: https://zynergyapps.com/docs/Boost-Math-2.6.0.pdf.

Function	Boost	Type
Create.Distribution(Dist.bernoulli)	boost::math::bernoulli	Class
Create.Distribution(Dist.betad)	boost::math::beta_distribution	Class
Create.Distribution(Dist.binomial)	boost::math::binomial	Class
Create.Distribution(Dist.cauchy)	boost::math::cauchy	Class
Create.Distribution(Dist.chi_squared)	boost::math::chi_squared	Class
Create.Distribution(Dist.exponential)	boost::math::exponential	Class
Create.Distribution(Dist.fisher_f)	boost::math::fisher_f	Class
Create.Distribution(Dist.gamma)	boost::math::gamma_distribution	Class
Create.Distribution(Dist.geometric)	boost::math::geometric	Class
Create.Distribution(Dist.inv_chi_squared)	boost::math::inverse_chi_squared	Class
Create.Distribution(Dist.inv_gamma)	boost::math::inverse_gamma	Class
Create.Distribution(Dist.inv_gaussian)	boost::math::inverse_gaussian	Class
Create.Distribution(Dist.laplace)	boost::math::laplace	Class
Create.Distribution(Dist.logistic)	boost::math::logistic	Class
Create.Distribution(Dist.log_normal)	boost::math::lognormal	Class
Create.Distribution(Dist.neg_binomial)	boost::math::negative_binomial	Class
Create.Distribution(Dist.non_cen_chi_squared)	boost::math::non_central_chi_squared	Class
Create.Distribution(Dist.normal)	boost::math::normal	Class
Create.Distribution(Dist.pareto)	boost::math::pareto	Class
Create.Distribution(Dist.poisson)	boost::math::poisson	Class
Create.Distribution(Dist.rayleigh)	boost::math::rayleigh	Class
Create.Distribution(Dist.skew_normal)	boost::math::skew_normal	Class
Create.Distribution(Dist.students_t)	boost::math::students_t	Class
Create.Distribution(Dist.triangular)	boost::math::triangular	Class
Create.Distribution(Dist.uniform)	boost::math::uniform	Class
Create.Distribution(Dist.weibull)	boost::math::weibull	Class
Stat.find_degrees_of_freedom	boost::math:: chi_squared::find_degrees_of_freedom students_t::find_degrees_of_freedom	Function
Stat.find_lower_bound_on_p	boost::math:: binomial::find_lower_bound_on_p geometric::find_lower_bound_on_p negative_binomial::find_lower_bound_on_p	Function
Stat.find_upper_bound_on_p	boost::math:: binomial::find_upper_bound_on_p geometric::find_upper_bound_on_p negative_binomial::find_upper_bound_on_p	Function
Special.airy_ai	boost::math::airy_ai	Function

Function	Boost	Type
Special.airy_ai_pri	boost::math::airy_ai_prime	Function
Special.airy_bi	boost::math::airy_bi	Function
Special.airy_bi_pri	boost::math::airy_bi_prime	Function
Special.bessel_i	boost::math::cyl_bessel_i	Function
Special.bessel_i_pri	boost::math::cyl_bessel_i_prime	Function
Special.bessel_j	boost::math::cyl_bessel_j	Function
Special.bessel_j_pri	boost::math::cyl_bessel_j_prime	Function
Special.bessel_k	boost::math::cyl_bessel_k	Function
Special.bessel_k_pri	boost::math::cyl_bessel_k_prime	Function
Special.bessel_sph	boost::math::sph_bessel	Function
Special.beta	boost::math::beta	Function
Special.beta_c	boost::math::betac	Function
Special.digamma	boost::math::digamma	Function
Special.elliptic_int_1	boost::math::ellint_1	Function
Special.elliptic_int_2	boost::math::ellint_2	Function
Special.elliptic_int_3	boost::math::ellint_3	Function
Special.elliptic_int_d	boost::math::ellint_d	Function
Special.elliptic_int_rc	boost::math::ellint_rc	Function
Special.elliptic_int_rd	boost::math::ellint_rd	Function
Special.elliptic_int_rf	boost::math::ellint_rf	Function
Special.elliptic_int_rg	boost::math::ellint_rg	Function
Special.erf	boost::math::erf	Function
Special.erf_inv	boost::math::erf_inv	Function
Special.erfc	boost::math::erfc	Function
Special.erfc_inv	boost::math::erfc_inv	Function
Special.exp_int_ei	boost::math::expint	Function
Special.exp_int_en	boost::math::expint	Function
Special.gamma_p	boost::math::gamma_p	Function
Special.gamma_p_der	boost::math::gamma_p_derivative	Function
Special.gamma_p_inv	boost::math::gamma_p_inv	Function
Special.gamma_p_inv_a	boost::math::gamma_p_inva	Function
Special.gamma_q	boost::math::gamma_q	Function
Special.gamma_q_inv	boost::math::gamma_q_inv	Function
Special.gamma_q_inv_a	boost::math::gamma_q_inva	Function
Special.hankel_1	boost::math::cyl_hankel_1	Function
Special.hankel_1_sph	boost::math::sph_hankel_1	Function
Special.hankel_2	boost::math::cyl_hankel_2	Function
Special.hankel_2_sph	boost::math::sph_hankel_2	Function
Special.hermite	boost::math::hermite	Function
Special.heuman_lambda	boost::math::heuman_lambda	Function
Special.ibeta	boost::math::ibeta	Function
Special.ibeta_c	boost::math::ibetac	Function
Special.ibeta_c_inv	boost::math::ibetac_inv	Function
Special.ibeta_c_inv_a	boost::math::ibetac_inva	Function
Special.ibeta_c_inv_b	boost::math::ibetac_invb	Function
Special.ibeta_der	boost::math::ibeta_derivative	Function
Special.ibeta_inv	boost::math::ibeta_inv	Function
Special.ibeta_inv_a	boost::math::ibeta_inva	Function
Special.ibeta_inv_b	boost::math::ibeta_invb	Function
Special.jacobi_zeta	boost::math::jacobi_zeta	Function
Special.laguerre	boost::math::laguerre	Function
Special.legendre_p	boost::math::legendre_p	Function
Special.legendre_p_pri	boost::math::legendre_p_prime	Function
Special.legendre_q	boost::math::legendre_q	Function
Special.lgamma	boost::math::lgamma	Function
Special.neumann	boost::math::cyl_neumann	Function
Special.neumann_pri	boost::math::cyl_neumann_prime	Function
Special.neumann_sph	boost::math::sph_neumann	Function
Special.owens_t	boost::math::owens_t	Function
Special.polygamma	boost::math::polygamma	Function
Special.tgamma	boost::math::tgamma	Function
Special.tgamma_delta	boost::math::tgamma_delta_ratio	Function
Special.tgamma_lower	boost::math::tgamma_lower	Function

Function	Boost	Type
Special.tgamma_ratio	boost::math::tgamma_ratio	Function
Special.trigamma	boost::math::trigamma	Function
Special.zeta	boost::math::zeta	Function

Table 28 - Boost's Library Function Mappings

8 Contact

- If you have a general questions about the app, please send e-mail to mailto:support@zynergyapps.com.
- If you encountered a bug in the app, please relay details to mailto:bug@zynergyapps.com:
    - If the bug is in evaluating an expression in the *Calculator* tab or scripting in the *Scripts* tab, please provide a simple example encountering the artifact.
    - If the bug is in the User Interface (UI), please provide sequence of steps using the app to replicate the artifact where it is misbehaving.
- If you have any suggestions, requests, or recommendations for the app, send email to mailto:suggestions@zynergyapps.com.

9 Licenses

9.1 Agreement

LICENSED APPLICATION END USER LICENSE AGREEMENT

Apps made available through the App Store are licensed, not sold, to you. Your license to each App is subject to your prior acceptance of either this Licensed Application End User License Agreement ("Standard EULA"), or a custom end user license agreement between you and the Application Provider ("Custom EULA"), if one is provided. Your license to any Apple App under this Standard EULA or Custom EULA is granted by Apple, and your license to any Third Party App under this Standard EULA or Custom EULA is granted by the Application Provider of that Third Party App. Any App that is subject to this Standard EULA is referred to herein as the "Licensed Application." The Application Provider or Apple as applicable ("Licensor") reserves all rights in and to the Licensed Application not expressly granted to you under this Standard EULA.

a. Scope of License: Licensor grants to you a nontransferable license to use the Licensed Application on any Apple-branded products that you own or control and as permitted by the Usage Rules. The terms of this Standard EULA will govern any content, materials, or services accessible from or purchased within the Licensed Application as well as upgrades provided by Licensor that replace or supplement the original Licensed Application, unless such upgrade is accompanied by a Custom EULA. Except as provided in the Usage Rules, you may not distribute or make the Licensed Application available over a network where it could be used by multiple devices at the same time. You may not transfer, redistribute or sublicense the Licensed Application and, if you sell your Apple Device to a third party, you must remove the Licensed Application from the Apple Device before doing so. You may not copy (except as permitted by this license and the Usage Rules), reverse-engineer, disassemble, attempt to derive the source code of, modify, or create derivative works of the Licensed Application, any updates, or any part thereof (except as and only to the extent that any foregoing restriction is

prohibited by applicable law or to the extent as may be permitted by the licensing terms governing use of any open-sourced components included with the Licensed Application).

b. Consent to Use of Data: You agree that Licensor may collect and use technical data and related information—including but not limited to technical information about your device, system and application software, and peripherals—that is gathered periodically to facilitate the provision of software updates, product support, and other services to you (if any) related to the Licensed Application. Licensor may use this information, as long as it is in a form that does not personally identify you, to improve its products or to provide services or technologies to you.

c. Termination. This Standard EULA is effective until terminated by you or Licensor. Your rights under this Standard EULA will terminate automatically if you fail to comply with any of its terms.

d. External Services. The Licensed Application may enable access to Licensor's and/or third-party services and websites (collectively and individually, "External Services"). You agree to use the External Services at your sole risk. Licensor is not responsible for examining or evaluating the content or accuracy of any third-party External Services, and shall not be liable for any such third-party External Services. Data displayed by any Licensed Application or External Service, including but not limited to financial, medical and location information, is for general informational purposes only and is not guaranteed by Licensor or its agents. You will not use the External Services in any manner that is inconsistent with the terms of this Standard EULA or that infringes the intellectual property rights of Licensor or any third party. You agree not to use the External Services to harass, abuse, stalk, threaten or defame any person or entity, and that Licensor is not responsible for any such use. External Services may not be available in all languages or in your Home Country, and may not be appropriate or available for use in any particular location. To the extent you choose to use such External Services, you are solely responsible for compliance with any applicable laws. Licensor reserves the right to change, suspend, remove, disable or impose access restrictions or limits on any External Services at any time without notice or liability to you.

e. NO WARRANTY: YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT USE OF THE LICENSED APPLICATION IS AT YOUR SOLE RISK. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE LICENSED APPLICATION AND ANY SERVICES PERFORMED OR PROVIDED BY THE LICENSED APPLICATION ARE PROVIDED "AS IS" AND "AS AVAILABLE," WITH ALL FAULTS AND WITHOUT WARRANTY OF ANY KIND, AND LICENSOR HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH RESPECT TO THE LICENSED APPLICATION AND ANY SERVICES, EITHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES AND/OR CONDITIONS OF MERCHANTABILITY, OF SATISFACTORY QUALITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF ACCURACY, OF QUIET ENJOYMENT, AND OF NONINFRINGEMENT OF THIRD-PARTY RIGHTS. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY LICENSOR OR ITS AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY. SHOULD THE LICENSED APPLICATION OR SERVICES PROVE DEFECTIVE, YOU ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES OR LIMITATIONS ON APPLICABLE STATUTORY RIGHTS OF A CONSUMER, SO THE ABOVE EXCLUSION AND LIMITATIONS MAY NOT APPLY TO YOU.

f. Limitation of Liability. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT SHALL LICENSOR BE LIABLE FOR PERSONAL INJURY OR ANY INCIDENTAL, SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, LOSS OF DATA, BUSINESS INTERRUPTION, OR ANY OTHER COMMERCIAL DAMAGES OR LOSSES, ARISING OUT OF OR RELATED TO YOUR USE OF OR

INABILITY TO USE THE LICENSED APPLICATION, HOWEVER CAUSED, REGARDLESS OF THE THEORY OF LIABILITY (CONTRACT, TORT, OR OTHERWISE) AND EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OF LIABILITY FOR PERSONAL INJURY, OR OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS LIMITATION MAY NOT APPLY TO YOU. In no event shall Licensor's total liability to you for all damages (other than as may be required by applicable law in cases involving personal injury) exceed the amount of five dollars ($5.00). The foregoing limitations will apply even if the above stated remedy fails of its essential purpose.

g. You may not use or otherwise export or re-export the Licensed Application except as authorized by United States law and the laws of the jurisdiction in which the Licensed Application was obtained. In particular, but without limitation, the Licensed Application may not be exported or re-exported (a) into any U.S.-embargoed countries or (b) to anyone on the U.S. Treasury Department's Specially Designated Nationals List or the U.S. Department of Commerce Denied Persons List or Entity List. By using the Licensed Application, you represent and warrant that you are not located in any such country or on any such list. You also agree that you will not use these products for any purposes prohibited by United States law, including, without limitation, the development, design, manufacture, or production of nuclear, missile, or chemical or biological weapons.

h. The Licensed Application and related documentation are "Commercial Items", as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation", as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States.

i. Except to the extent expressly provided in the following paragraph, this Agreement and the relationship between you and Apple shall be governed by the laws of the State of California, excluding its conflicts of law provisions. You and Apple agree to submit to the personal and exclusive jurisdiction of the courts located within the county of Santa Clara, California, to resolve any dispute or claim arising from this Agreement. If (a) you are not a U.S. citizen; (b) you do not reside in the U.S.; (c) you are not accessing the Service from the U.S.; and (d) you are a citizen of one of the countries identified below, you hereby agree that any dispute or claim arising from this Agreement shall be governed by the applicable law set forth below, without regard to any conflict of law provisions, and you hereby irrevocably submit to the non-exclusive jurisdiction of the courts located in the state, province or country identified below whose law governs:

> If you are a citizen of any European Union country or Switzerland, Norway or Iceland, the governing law and forum shall be the laws and courts of your usual place of residence.
>
> Specifically excluded from application to this Agreement is that law known as the United Nations Convention on the International Sale of Goods.

9.2 Third-Parties

- **Boost 1.71.0**
  Boost License: https://www.boost.org/LICENSE_1_0.txt

- **Charts 3.6.0**
  Copyright © 2016 Daniel Cohen Gindi & Philipp Jahoda
  Apache License 2.0: http://www.apache.org/licenses/LICENSE-2.0

- **JSON 3.10**
  Copyright © 2013-2018 Niels Lohmann
  MIT License: https://opensource.org/licenses/mit-license.html

- **Least Squares**
  Copyright © 2018 Manas Sharma
  MIT License: https://opensource.org/licenses/mit-license.html

- **Text Kit Line Numbers**
  Copyright © 2013 Mark Alldritt
  MIT License: https://opensource.org/licenses/mit-license.html

- **Search WKWebView**
  Copyright © 2014 2019 Scott Stahurski
  MIT License: https://opensource.org/licenses/mit-license.html

- **Icon8**
  License: https://icons8.com/license