PDAcalc

User manual

Version 1.60

PDAcalc

Warning and Disclaimers

ADACS LLC makes no warranty, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall ADACS LLC, be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of ADACS LLC regardless of the form of action, shall not exceed the purchase price of this application. Moreover, ADACS LLC, shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

Contact information

ADACS LLC

Advanced Digital & Analog Consulting Services
12076 Marsh Hen Lane
Tega Cay, SC 29708
Phone: 803.833.8312
Fax: 803.547.4667
Email: support@adacs.com
Web site: *www.adacs.com*

Table of contents

About ADACS LLC

ADACS LLC is a leading provider of graphical programmable calculators for the personal digital assistants PDA's. Our founder and president, Evert H. Rozendaal, designed the first calculator, CplxCal, for the palm platform using complex numbers in 1998. CplxCalPro followed in 2000 and was the first graphical programmable calculator for the palm platform. MtrxCal, a matrix calculator especially designed for the palm, was released in 2001. In 2003 PDAcalc modules where released to bridge the gap between different platforms. PDAcalc modules are currently available for palm devices, windows and the PocketPC.

Tell us what you think

As the reader of this book and the user of PDAcalc, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, and any other words of wisdom you're willing to pass our way.

Acknowledgment

At various stages of the development of our products, a number of people have given us invaluable comments. In this regard we owe a debt of gratitude to James Derry for his invaluable help writing this manual, Gilfred B. Swartz for his help with PDAcalc matrix. We also like to thank John Molinder, Doug McCready, Haig Terzian and Meric Ozcan for all their help and support.

Introduction

If this is your first time reading this manual, chances are you've just installed an unregistered copy of PDAcalc onto your PDA to try it out, perhaps to compare it to other PDA calculators available on the Web, and see if it meets your needs. Whether those needs are of a professional who requires a calculator to process field results or a student who's trying to get a grasp on science, math, or engineering concepts, we feel that the PDAcalc is up to the challenge. PDAcalc is a power user's non-RPN calculator. It uses a 64-bit double-precision floating-point format, providing an approximate numerical range of –2.23E-308 <= n <= 1.80E308.*

PDAcalc was designed to be the most flexible calculator for PDA's today, allowing the user to do more complicated calculations than ever before. It has a user-configurable keyboard, and has several different modules. Versions of PDAcalc exist for the Palm, PocketPC, and Windows platforms. A script written on one platform will run on all of them. Write once, run everywhere.

PDAcalc provides a framework of sorts upon which any one of our PDAcalc modules will run. Those modules, in the order in which they were released, are: PDAcalc units, a module specially designed for automatic unit conversions; PDAcalc classic, a module for general purpose calculations; and PDAcalc matrix, a module for matrix calculations.

* Refer to Appendix A, Technical Specifications, for the exact range.

Below an image of each:

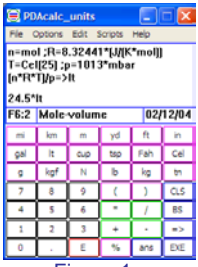


Figure 1

PDAcalc_units

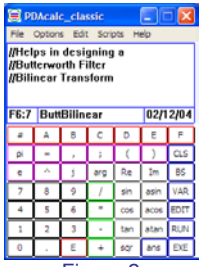


Figure 2

PDAcalc_classic

Figure 3

PDAcalc_matrix

If you're shaky on programming or graphics manipulation, don't panic. This manual goes over both, and includes a primer to walk you through the fundamentals of programming. Furthermore, PDAcalc comes with many ready-to-use scripts, and we maintain a library of scripts for the PDAcalc modules for you to download at http://www.pdacalc.com This means a solution to meet your needs may already exist! PDAcalc allows you to store, retrieve, and manage scripts. Scripts can be grouped by category. And the number of scripts is limited only by the amount of free memory in your PDA. Please take a few moments to sit down with this manual and PDAcalc to familiarize yourself with this calculator. This manual was designed to serve as an easy-to-follow guide to PDAcalc, as well as a handy reference for those tackling real-world problems with our calculator.
And most of all, enjoy!

PDACalc and its Modules

A Gallery of Scripts and Graphics

For those of you who want to know if looking into PDAcalc is worth your time, we offer here a small gallery of graphics to illustrate its power:

PDAcalc units

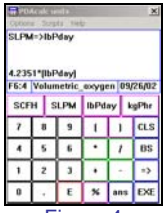


Figure 4

Figure 5

Notice how the result in Figure 5 is automatically converted to liters.

PDAcalc classic

General purpose, fully programmable, graphical calculator.

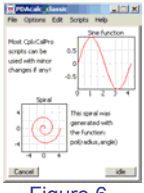


Figure 6

Draw multiple graphs and text on the graphical screen.

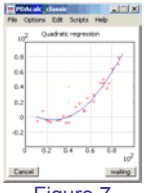


Figure 7

Do statistical analyses on large sets of data. Here PDAcalc classic plots the data points, then draws their trend line.

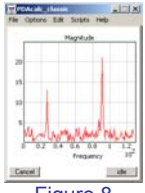


Figure 8

PDAcalc classic has a suite of built-in functions that makes doing sophisticated analyses a breeze.

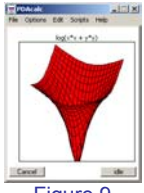


Figure 9

Draw 3D functions with just a few simple functions.

PDAcalc matrix

Most powerful, fully programmable, matrix calculator for your PDA.

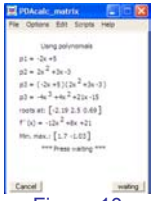


Figure 10

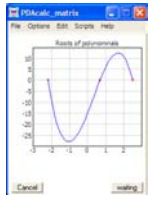


Figure 11

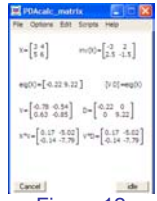


Figure 12

Differences between platforms

Figure 13

Windows

Figure 14

PocketPC

Figure 15

Palm OS

As shown above, PDAcalc mostly has the same look and feel on all platforms. The only differences are the locations of the menus, memory available, the speed at which the scripts are executed, and some additional functions on the windows platform. The Windows version contains additional functions like uploading scripts directly to our web site, browsing scripts on our website, etc. This is not supported on the palm and PocketPC because a bigger screen is required for these features. We use screen shots from different platforms in this manual.

The Calculator

As stated in the introduction, PDAcalc serves as a GUI framework of sorts for its modules. The advantages of writing PDAcalc this way include:

- Less memory required for multiple modules.
- Windows and PocketPC only need to install one DLL for a new module.
- No learning curve for the GUI when adding a module.
- Quicker turnaround for adding features.
- Synchronization of scripts between different platforms.

A uniform GUI for modules means that the default main screen for the modules, and their menus, differ only in support of their unique features. Note the uniformity of the main screens of the modules:

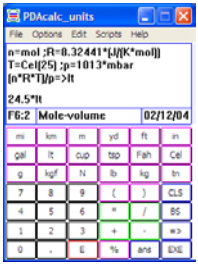


Figure 16

Figure 17

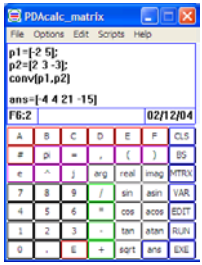


Figure 18

A module can have several screens. Let's focus on the main screen as it is uniform across modules. The top 3 lines are called the scratchpad. The fourth line displays the result of your calculation. It also becomes the value of ans, a buffer to hold intermediate results of your calculations. The fifth line is the status line. It shows the display format of PDAcalc you're using, the name of the script you're running, and the date. Below that is the keyboard. The default layout of the keyboard depends on the module.

From this screen you can enter equations and perform immediate computations, just as you can if you were using a dedicated algebraic calculator.* The difference is you can carry the answer from one calculation to another on the 4th line.

* Here we use "dedicated" to mean special-purpose hardware. Examples include Casio, TI, and HP calculators.

Modifying Key Assignments

Some keys on the main screen have a special meaning or perform a special function. These keys are listed below.

Keys	Function
#	Hexadecimal format key.
= / * - +	Arithmetic operators.
0-9	Numeric input keys.
^	Exponentiation.
,	Comma. Separates arguments.
;	Semi-colon. Separates statements.
ans	The variable which holds the results of intermediate calculations. Pressing this key enters "ans" at the cursor location.
( )	Parentheses modify precedence of arithmetic operators.
pi	Math constant equal to 3.14159265358979284808.
e	Math constant equal to 2.7182818284590458404.
arg	Enters the function "arg(" at the cursor location. Arg(x) returns the angle of x.
Re / real	The real of x.
Im / imag	The imaginary of x.
j	The imaginary portion of a complex number.
sqr sqrt	The square root of x.
sin, asin, cos, acos, tan, atan, sqr	Enters basic trig functions or their inverses.
A B C D E F	Pressing any of these keys enters the variables A-F at the cursor location. These keys are also used for hexadecimal entry.
CLS	Clears the scratchpad area.
VAR	Displays a list of all the assigned variables.
FUNC	Key that, when pressed, brings up list of PDAcalc's built-in functions. Choosing a function from the list places it on the scratchpad.
EDIT	Brings up the program currently loaded into PDAcalc for editing.
RUN	Evaluates the three lines in the scratchpad, then runs the loaded program.
EXE	Evaluates the three lines in the scratchpad only.

The user can assign all keys on the main screen. A simple text file, the keyboard assignment file, determines key assignments. Each line of text maps to a row of keys, and a comma separates the text for each key. Frequently-used functions can have keys assigned to them for quick and easy entry of your equations. Let's look again at the default keyboard:

Figure 19

All the keys were assigned by reading the text file on the right. Notice the substitution of the tilde for the comma. The comma is used as a delimiter between key assignments.

These key assignment files are stored in the keyboard category of the PDAcalc database. Use the program editor to create or edit these files.

```
#,A,B,C,D,E,F
pi,=,~,;,(,),CLS
e,^,j,@angle,@real,@imag,BS
7,8,9,/,@sin,@asin,VAR
4,5,6,*,@cos,@acos,EDIT
1,2,3,-,@tan,@atan,RUN
0,.,E,+,@sqr,ans,EXE
```

Select [Scripts] from the menu and then select [load script]. This shows the available scripts in the selected category. Next, select the keyboard category and you should see at least two files, “MainKeys” and “Programmer”. Select “Programmer” and then tap [Edit] to edit the script or tap [load] to load the script. When loading a keyboard script, the keyboard will change accordingly. After a key is pressed in the main screen, PDAcalc determines if a “special” key was pressed. When a “special” key is pressed, the PDAcalc executes a “special” function accordingly. Reserved keywords determine if a key is “special”. If the keyword EXE is assigned to a key, for instance, PDAcalc executes the lines in the scratchpad (the top three lines of the display) when that key is pressed.

Reserved keywords for PDAcalc key assignment

EXE	Evaluate the three lines in the scratchpad.
RUN	Evaluate the three lines in the scratchpad, then run the script.
EDIT	Edit the program.
FUNC	Show a list of all the functions. Tapping one on the list puts it in the scratchpad.
VAR	Show a list of all the assigned variables.
FLT	Set display format to float.
HEX	Set display format to hexadecimal.
BIN	Set display format to binary.
OCT	Set display format to octal.
@	Put an open-parenthesis in the scratchpad. This saves time when using functions.
&	Evaluate the three lines in the scratchpad, then run the program. The iskey() function can be used to test for key.
=	Put the equal sign at the cursor position.
$	Variable assignment of a key. Pressing a key assigned a variable puts the variable, an equal sign, and the value of the variable in the result line.

The graphical screen

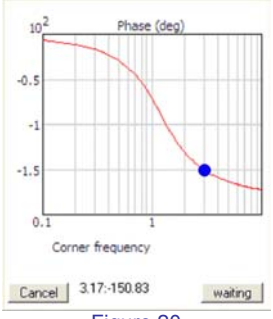


Figure 20

The graphical screen of PDAcalc is shown in Figure 21. When you tap inside an active graph the cursor position will be shown at the bottom of the screen. Due to the limited screen space a power of ten multiplication is used for the y-axis in the screen shot. In this example multiply the y-values by 10^2. The blue dot indicates the cursor position shown at the bottom of the screen. When you plot multiple graphs the last plotted graph is the active graph.

The button at the bottom right shows the current state of the script. When 'waiting' is shown the program waits till this button is pressed before continuing the script. Pressing the cancel button will cancel the program and return to the main screen.

The Menu Items

The menu items are depending on the module and the platform. However most of the menu items are the same for all modules and platforms.

Options

Figure 22

Preferences

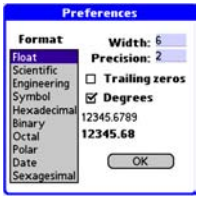


Figure 23

The preferences form allows you to set the display format of numbers, angular measurements in radians or degrees* and whether to show trailing zeros in the result.

The following formats are supported: Float, Scientific, Engineering, Symbol, Hexadecimal, Binary, Octal, Polar, Date and Sexagesimal.

The display format determines the width and precision of the displayed numbers. The sum of these two numbers can't exceed 24. In the example on the left, the sum is 8, which means a number like 1234.56789 will be rounded and displayed as 12345.68

Default preferences

format ……	float
width ……..……	6
precision ……..……	4
angular measurement, if supported by module……	degrees
trailing zeros……..	no

The format function fmt(t,w,p,tr) can be used to set format options to be used during the execution of a calculation or program.
t: 0-float, 1-Scientific, 2-eng, 3-sym, 4-hex, 5-bin, 6-oct, 7-pol, 8-date, 9-sexagesimal
w: width of number (0-15)
p: precision of number (0-15)

* Please note that because the matrix module was written to be compatible with Matlab, trig functions in this module take or return radians only. The matrix module does not support angular measurements in degrees.

tr: trailing zeros. (0 or 1)

Preferences for angular measurement, if supported by the module, can be changed with the functions:

stdeg() Sets angular format to degrees.
Strad() Sets angular format to radians.

Display format example

Some examples of how numbers are displayed in different formats, widths and precisions by changing the preferences:

Number	Format	Width	Precision	Display	Comments
123456.789	float	7	2	123456.79	Default settings of PDAcalc.
123456.789	float	7	3	123456.789	ah-hah! PDAcalc retained the last digit.
123456.789	float	1	5	1.23457E05	Only one digit in front of the period.
123456.789	float	1	9	1.23456789E05	Nine digits maximum behind the period.
0.000123456	float	1	7	0.00012346	Shows leading zeros.
0.000123456	scientific	1	7	1.23456E-04	No leading zeros
123456.789	engineering	1	9	123.456789E03	Engineering formats by 10^(3n), where n >= 0.
123456.789	engineering	1	9	123.456789E03	No difference.
123456.789	symbol	1	9	123.456789k	“symbol” means SI symbols, and “k” means “kilo” or “multiply by 1000”.
1234567	hexadecimal	4	9	#12D678:1,234,567	
1234.567	hexadecimal	4	9	#4D2:1,235	PDAcalc rounds off the decimal part of input, then hexes the result.
1234.567	binary	4	9	10011010010	PDAcalc also bins rounded input.
1234.567	octal	4	9	2322	Also octal.
Sqr(-1)	polar	4	9	pol(1, 90)	Polar format displays complex numbers in polar format. 90 is the angle in degrees.
1+j1	polar	9	4	pol(1.4142, 45)	As expected the magnitude is sqr(2).
3090000000	date	0	4	Fri, Nov 30, 2001	When width is set to 0, date is displayed in full format.
3090000000	date	1	4	11/30/01	When width is set to 1, date is displayed in compressed format.
1.5	sexagesimal	9	4	1°30’0”	Sexagesimal format converts a decimal input into DMS.
1.75	sexagesimal	9	4	1°45’0”	3/4’s of 60 minutes is 45

Variables

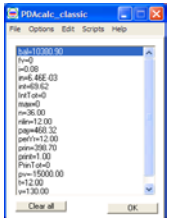


Figure 24

The variables screen (under the options menu) shows you variables and their values. Press [Clear all] to clear all variables. The variables are not cleared before loading a script. Loading and running scripts keeps on adding variables to this list till the list is full. When the list is full, you will see a message on the main screen to clear variables.

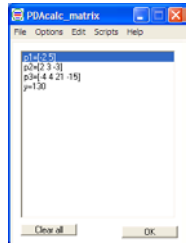


Figure 25

When you see this on the main screen, just select [Options] => [Variables] and [Clear all] to continue. When a matrix is to big to be displayed in the variables screen three periods will be shown after the last shown element of the matrix. The closing bracket will not be shown either.

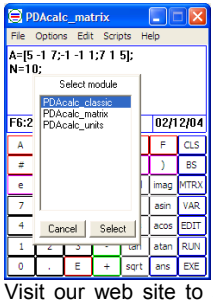
Default Keyboard

Just as this menu item name says, selecting it clears the user-defined keyboard and loads the default keyboard.

Available memory

This option shows memory information, not applicable on PDAcalc for Windows and PocketPC.

Select module



Use the select module screen (under the options menu) to select a different module. The windows and PocketPC version checks for available libraries, DLL's, in the PDAcalc directory and shows them in the list.

Over time ADACS intends to release more modules.

Visit our web site to sign up for our newsletter and receive emails when new modules are available.

Beam program

This option will beam the program from a palm device to an other palm device. At the time of writing the manual this feature was not available on PDAcalc for Windows and the PocketPC.

Beam all scripts

This option will beam the scripts from a palm device to an other palm device. At the time of writing the manual this feature was not available on PDAcalc for Windows and the PocketPC.

Register

Figure 26

You can use PDAcalc without registering for about three weeks. This allows you to evaluate the calculator to see if it meets your needs.

Registration gives the user access to all features of PDAcalc. It also disables the window asking you to register.

We highly recommend that you use the windows version to register the different versions even if you do not want to register the windows version. During the installation, all PDA user name for the different platforms, needed to generate the registration code, is stored on your system.

After tapping on [Register online], this information is transferred to our web site, avoiding errors; and you can register for multiple platforms. After clicking the [Register online] button, you can watch the URL of the web site to see the information transferred.

Select the product(s) you like to register and entering your personal information, press [Next step]; and a printout of your order will be shown. We recommend printing this page for your own records. Press [Next step] again, and you will be transferred to a secure server to enter your credit card information. Notice the https:// at the beginning of the URL before entering your credit card information. After completing the credit card information form, press [process] to complete the order. After the order is processed, our local server receives an email and generates your registration code(s) automatically. Due to the importance of this process, we keep our local server up and running 24/7. Your registration code(s) are normally emailed to you within 30 minutes after the order is processed successfully.

Edit

Figure 27

This is the standard [Edit] menu. This menu is a little different on the different platforms depending on the standards of the platform. PDAcalc allows you to copy-and-paste values from the result line into the scratchpad, or to share information across applications (e.g., to copy values from the result line into a memo).

Scripts

Figure 28

New script

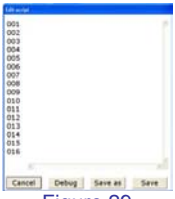


Figure 29

Selecting [New script] from the [Scripts] menu brings up the script editor. The script editor is much like a simple text editor, with each line numbered. Numbering lines simplifies debugging scripts.

Load script

Selecting [Load script] from the [Scripts] menu will display the available scripts. Before showing this window, PDAcalc will check the supported devices on your system and show them in the circled area. This means that if your device is connected to web you can select [ADACS web] to load scripts directly from our web site. At the time of writing not all communications links where supported yet by PDAcalc.

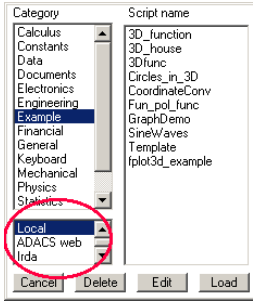


Figure 30

Just run PDAcalc and check the list to see which ones are supported. Possible communication links are:

local	Local storage on your device.
ADACS web	ADACS web site.
Serial	RS232 serial communication.
Irda	Infra red communication.
Wi-Fi	Local wireless network.

After selecting a different communication link, a new list of available scripts will be shown depending on the available scripts via the selected link. Select one of the scripts and tap on one of the buttons at the bottom of the screen to delete, edit or load the script. After pressing the load button PDAcalc will check for a file with the same name in the keyboard category. When such a file exists in the keyboard category that file will also be loaded and the keyboard will change accordingly.

Edit script

On the Windows version, the script editor screen is bigger than it is on PDAs, and shows line numbers. The smaller screen size on the Palm is why line numbers are not shown on the Palm version.

Figure 31

Edit on palm system

After viewing or editing your script, you can debug your script, save the script under a different name or save the script using its current name.

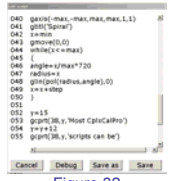


Figure 32

Edit on windows

Debug scripts

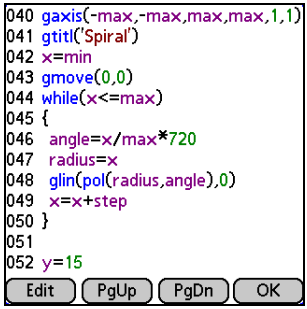


Figure 33

Debug script on palm

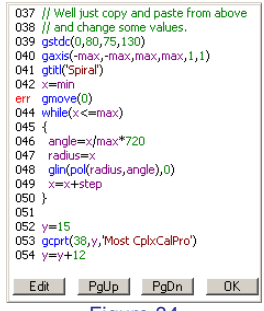
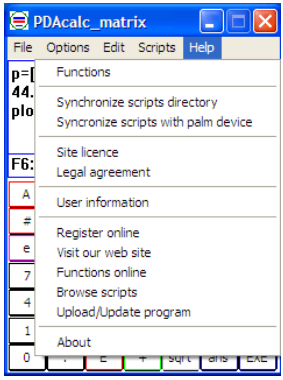


Figure 34

Debug script on windows

The debug screen will show the scripts color coded. It will also show the first error if the script contains an error as shown on the right. The screen shots above show a little PDAcalc classic scripts as an example. Notice that there is an argument missing in the gmove() function, and the red 'err' message replaces the line number where the error occurs. As commonly used in programming languages, parts of the script are indented to improve readability. This is known as "pretty print" formatting and is done automatically by PDAcalc classic and PDAcalc matrix.

Help



Functions

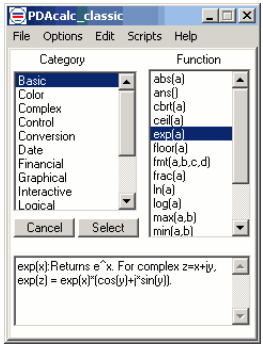


Figure 35

This item shows a list of all the available built-in functions. Select a category and tap on a function for a brief description of the function. The categories and functions are different for different modules.

The function descriptions, as shown at the bottom of Figure 36, are stored in the document category. You can edit the descriptions by selecting Scripts – Load script and select the Document category.

Visit our web site for a more detailed description of the functions.
http://www.pdacalc.com

Site licenses

We have special volume discounts. This screen will indicate if you have a discount version or a standard version. Companies or schools might have there own special copy indicated by this screen.

Legal agreement

This screen will show you what you probably expected already!

About

Also shows our web site address for the latest information.

The help items below are only available on the Windows version.

Synchronize scripts with a Palm device

Select this menu item to toggle synchronization of scripts between a palm device and your windows version. When the check mark is shown all scripts are synchronized when the hotsync button is pressed on the palm cradle.

Register online

Clicking this menu item will open your browser and transfer your PDA user names to our web sites when connected to the Internet. Enter your information and select the product(s) you like to register. If you like to order more than one module or for more than one platforms take a look at the bundles for discounts.

Visit our web site

Clicking this menu item will open your browser and show our main web site. It should be no surprise that an Internet connection is required.

Functions online

Clicking this menu item will open your browser and show a list of built-in functions. These pages contain additional information about the functions and user-contributed notes.

Browse scripts

Clicking this item will open the browser and show the latest scripts that were uploaded to our web site. To load a script, see Load scripts

User information

Select this item to enter your user information. This information is used when you upload a script to our web site. A password is used to prevent other people from updating your scripts and taking the credit for it.

Upload/Update script

Clicking this item will convert your script to color coded HTML and upload your script to our web site. Enter a short description of the script and enter a long description of your script and press upload script to complete the upload. When you make changes to your script and select Upload/Update script the descriptions stored on our web server will be shown in a edit field. Use this field to update the description of the script. When your script contains an error, the script can not be uploaded. You will have to fix the error before uploading.

Creating a document

If you like to document your script you can use the Upload/Update menu item to create the color coded script in HTML format. Highlight the script and press [ctrl] + C to copy the script into the clipboard. Open a word processor or HTML editor and paste the script into it.

Scratchpad and program space

When you load a script the first three lines will be shown in the scratchpad of the main screen. When you press the EXE button only the three lines in the scratchpad will be evaluated. When you press the RUN button for the first time the three lines in the scratchpad will be evaluated. Next the rest of the script will be converted and optimized to run as fast as possible and stored in program space. Then the code in program space will be executed.

The next time you press the RUN button the three lines in the scratchpad will be evaluated again and then the code in program space will be executed without the step of converting and optimizing. Converting and optimizing can take a long time when using a palm device with a slow processor. Of course it also depends on the size of the script. The main thing to remember is to put variables you change frequently in the scratchpad, first three lines. Otherwise if you like to change a variable within the script you have to edit the script and wait for PDAcalc to convert and optimize the script again.
When you change the variables in the scratchpad on the main screen the changes will not be saved in the script. Hence the name scratchpad.

Startup sequence

When you start PDAcalc for the first time the program will search for available modules. The available modules will be shown in the module selection window. After selecting a module PDAcalc will search for the initialization script. When found it will execute this script. Use this script to initialize frequently used variables. For the classic and matrix module the initialization script should be saved as 'Initialize' in the General category. For the units module the script should be save as 'Initialize' in the conversion category. Then PDAcalc will search for an initialization script in the keyboard category. When a script called 'Initialize' is found the keyboard category the script will be loaded and the keyboard will change accordingly.

The next time you start PDAcalc the last used module will be loaded automatically. After loading the module PDAcalc will search for the initialization files as described above.

Where to Go From Here

The full functionality of PDAcalc depends on its modules. Now that you've gone over the main screens and menus, you should refer to whichever section pertains to the module you're interested in.

Units module

Now that we're familiar with the main screen, let's do some basic calculations on PDAcalc units using the default preferences:

Basic calculations

Objective:	You press:	Unit system	Displays:	Remarks:
3 + 4	3 + 4 [EXE]	N/A	3+4 7	Pretty straightforward.
(3+4) * 2	[CLS][ans]* 2 [EXE]	N/A	ans*2 14	ans key supplies the previous result in the equation.
Convert 4 miles to kilometers.	[CLS] 4[mi] [=>] km [EXE]	English -> SI	6.44*km	PDAcalc units has a => key for making conversions.
How many gallons will a container of 1.2 yards wide, 2 feet deep and 11 inches high hold?	[CLS] 1.2[yd]*2[ft]*11[in] [=>] gal [EXE]	English	49.37*gal	PDAcalc units takes care of conversions to a consistent unit (say, all units of length to inches). Note that inches cubed and gallons are volume measurements.
How many liters will a container of 1.2 yards wide, 2 meters deep and 11 feet high hold?	[CLS] 1.2[yd]*2[m]*11[ft][=>][lt] [EXE]	English, SI ->SI	7357.92*lt	A strange problem statement, as units of length from different unit systems are specified. However, PDAcalc units is not as confused as I am.
Given c=299792458 m/s, what is the distance in miles that light travels in 1 year?	[CLS] (299792458m/s)*365 .25day[=>]mi	SI->English	5.88E12*mi	A calculation that any self-respecting Trekkie must perform at least once per calculating device.
What is the speed of light in furlongs/fortnight?	[CLS] furlong=201.168m 299792458m/s[=>]fu rlong/(2wk)	SI->English	1.8E12*furl ong/(2wk)	The furlong and fortnight are the units of length of time preferred by the Luddite faction in Starfleet Academy.
110V * 16A	[CLS]110V*16A [EXE]	SI	1760*W	PDAcalc units has many more units than can be displayed on the main screen. To use them, just type them in on the PC version, or write them in with your stylus on the Palm and PocketPC versions. Note that we didn't have to enter [=>] W. When a unit to convert into is not specified, PDAcalc units returns the value in its base unit. Base units are SI. The watt is the unit of power in SI.
110V + 16A	[CLS] 110V+16A [EXE]	SI	1: Error V+A	Of course. Volts and amps are units of different physical dimensions (the volt is a unit of electrical potential, while the ampere is one of electric current), and so cannot be added. Please don't expect the impossible from our calculators!

Preferences

The preferences screen allows you to change the format in which numbers are displayed and the default currency. The PDAcalc units preferences screen includes the option to select either British or American measurements.

As shown on in figure 37 six different formats are supported. This screen also shows how the value 12345.6789 will be displayed based on the settings. The screenshot shows the engineering format. When the number is greater than 10^{Width} an exponent is used to display the value. In the screenshot the value 12345.6789 is greater than 10^{4} and the precision, decimal places, is two.

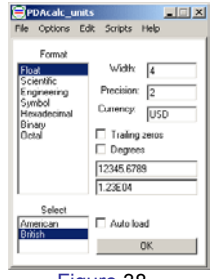


Figure 38

Select between American or British units. Check the 'auto load' check box if you like the last used script to be loaded automatically when starting PDAcalc units.

Assigning units

You can create and assign values to units not defined on PDAcalc units. Let's assume you'd like to compare the price of .7 kg of apples in the US with their price in Euros. The US dollar (USD) is defined on PDAcalc, but no other currency. We create and assign values to other currencies in terms of the USD. Consider the following:

EUR=1.03USD
apples =1.29USD/lb
0.70kg*apples=>EUR

Here we've created and assigned a value to the Euro (EUR) to terms of the USD, then used it in our calculations. Enter those three lines on PDAcalc units, then press [EXE].

Help information

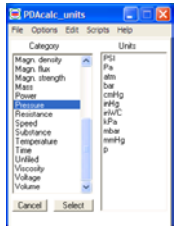


Figure 39

To select a unit that is not shown on the keyboard, from the Help menu, select Units. Units are organized by category; most categories describe the physical dimension of the units that fall under them. Select a category, and a list of units will be shown. You can add units not already defined on PDAcalc units. Added units must be defined in the base unit of the category (as EUR is defined in terms of the USD in the example above). User-added units will be show in the appropriate category. After adding a unit like EUR=1.03USD this unit will only be shown in the currency category.

Base units

Description	Abbreviation	Unit
Distance	m	meter
Mass	kg	kilogram
Time	sec	seconds
Current	A	ampere
Temperature	Cel	Celsius
Luminous intensity	cd	candela
Substance	mol	mol
Currency	CUR	Default currency.

User scripts

Loading a script from this category will display the contents of the script in the scratchpad. This is where you store little scripts that you need to change frequently. A good example is the little script below:

EUR=1.03USD
apples=1.29USD/lb
0.70kg*apples=>EUR

After loading this script it is easy to change in the scratchpad and press [exe] to calculate the new result. Changes are not saved.

Conversion

Loading a script from this category will not display the contents of the script in the scratchpad. This is where you store the bigger scripts that don't change. After a script in the category is loaded PDAcalc™ units will check for a script in the keyboard category with the exact same name. When such a file exists it will be loaded and the keyboard layout will be changed accordingly.

Keyboard

Loading a script from this category will change the layout of the keyboard. Below is the little script used to create the keyboard layout on the right.

Figure 40

```
mi,km,m,yd,ft,in,cm
pt,gal,lt,cup,tsp,Fah,Cel
mg,g,ct,oz,N,lb,kg,tn
7,8,9,(,),CLS
4,5,6,*,+,BS
1,2,3,/,-,=>
0,.,E,%,ans,EXE
```

Notice the commas between the text for each button. Deleting the top line for instance will increase the height of each button to use the same screen space for the keyboard. The width of the keys is also automatically adjusted and depending on the number of buttons per row.

Functions

Function	Description	Unit
Sqr(X)	Square root	none
Cel(T)	Converts celsius to default temperature unit	Temperature
Fah(T)	Converts Fahrenheit to default temperature unit	Temperature

Examples

PDAcalc units includes the following example script:

Volumetric example

From the conversion category select the Volumetric_oxygen script.

```
001 // Volumetric flow and mass flow
002
003 // Oxygen
004 // 1 atmosphere
005 // 70 degrees F
006 // 90% purity
007 density=1.33kg/m^3
008
009 SLPM=lt/min
010 SCFH=ft^3/hr
011 lbPday=lb/(density*day)
012 tnPday=tn/(density*day)
013 kgPhr=kg/(density*hr)
```

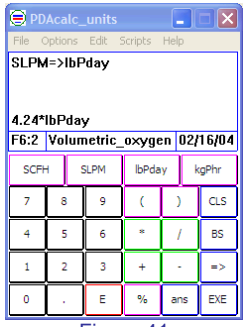


Figure 41

Notice that // marks the start of a comment; all that follows will not be processed.

After pressing load the Volumetric_oxygen script from the keyboard category will be loaded and the screen shown on the right will appear. The density in this script can be easily changed to any other gas or fluid.

Mole-volume example

$Given:$

$n = 1 \cdot mole$

$R = 8.31441 \cdot \frac{\mathrm{J}}{\mathrm{K} \cdot \mathrm{mol}}$

$T = 25^{\circ} Celsius$

$P = 1013 \cdot mbar$

Calculate the mol-volume V_m in liters, i.e. for the volume that 1 mol of a gas occupies. Let's take a moment to consider what we've got here. The problem statement gives us values to physical dimensions of the ideal gas law, PV = nRT. The dimensions of the universal gas constant *R* are energy per degree per mole. The units that the physical dimensions take, and therefore that their values take, depend upon which unit system we use. In one unit system, *R* is 8.31441 joules per Kelvin per mole. In another, it is 8.314×10^7 ergs per Kelvin per mole, in another 1.986 calories per Kelvin per mole; and in yet another it is 0.08207 liter-atmosphere per Kelvin per mole. (The inside cover of many chemistry books list at least three of these values for R; the units of R tell us which unit system a given value of R works in.) In chemistry classes, we're taught to choose our value of R based on the unit of the output we want, then if necessary convert values of the other physical dimensions to units of that unit system; finally, set up the ideal gas law equation algebraically to solve for the unknown, then crank away on your Curta, slide away on your slipstick, or bang away on your calculator to get the answer.
Fine, but look what we're given as a value of R, and what we're asked for as output. Following the advice of our chemistry teachers, we should choose R=0.08207 liter-atmosphere per Kelvin per mole, since we're being asked to give an answer in liters, then work the problem.

But, hey: we're solving this problem on PDAcalc_units. Look how we set it up and solve for V:

$$V_m = \frac{n \cdot R \cdot T}{p}$$

$$V_m = \frac{1 \cdot mole \cdot 8.31441 \cdot \frac{J}{K \cdot mol} \cdot 298.15 \cdot K}{1013 \cdot mbar}$$

Now set up and render the equation as a script in which we specify in the last line that the output is to be given in liters (using =>lt), thus:

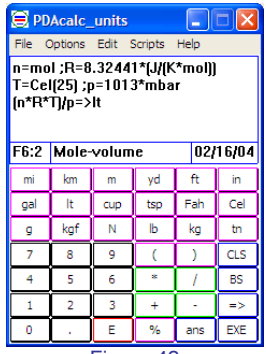


Figure 42

```
001 n=mol;R=8.32*(J/(K*mol))
002 T=Cel(25);p=1013*mbar
003 (n*R*T)/p=>lt
```

And PDAcalc units prints the result in liters.

Without specifying a unit for the output, the result renders in m^3, the default measure of volume in the unit system of R and the other physical dimensions in the problem statement.
Of course you should save the constant R in a script if you use it often and change the layout of the keys for easier entry of the values and units.

Helical Coil Spring example

A tightly wound helical coil spring is made from a material whose shear modulus is G. The bar from which the spring is made has a diameter D. The spring has a coil radius r with Nc active coils. What is the change in length of the spring from its unstretched length when the spring hangs vertically with one end fixed and a block of mass m_b attached to its other end?
PDAcalc units includes the following example script:
From the conversion category select the HelicalCoil script.

```
001 G=8E10*N/m^2
002 D=0.02m
003 Nc=80
004 r=0.08m
005 mb=200kg
006
007 // The stiffness
008 k=(G*D^4)/(64*Nc*r^3)
009
010 // The change in length
011 // of the spring due to
012 // the attached mass
013 x=(mb*g)/k
```

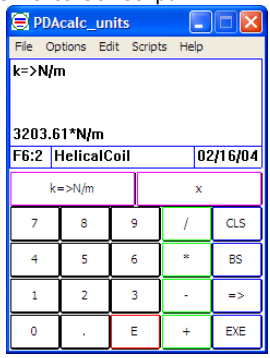


Figure 43

Predefined units

Abbreviation	Multiplication	Unit	Unit system
Constants			
pi	3.141592654		
g	9.80665	m/s^2	
Distance			
mi	1609.344	m	
km	1000	m	
yd	0.9144	m	
ft	0.3048	m	
in	0.0254	m	
cm	0.01	m	
Area			
acre	4046.86	m^2	
ha	10000	m^2	
Volume			
tsp	5.00E-06	m^3	
cup	0.000236587	m^3	
lt	0.001	m^3	
gal	***0.0037854***	***m^3***	***American***
gal	***0.00454609***	***m^3***	***British***
pt	0.0004732	m^3	

Mass

ct	0.0002	kg
dyn	1.02E-06	kg
gr	0.001	kg
mg	1.00E-06	kg
N	0.101971621	kg
oz	0.0283495	kg
lb	0.453592	kg
tn	1000	kg

Voltage

mV	0.001	V
kV	1000	V

Current

mA	0.001	A
kA	1000	A

Resistance

kohm	1000	ohm
Mohm	1.00E+06	ohm

Power

mW	0.001	W
kW	1000	W

Induction

mH	1.00E-03	H
uH	1.00E-06	H

Capacity

uF	1.00E-06	F
nF	1.00E-09	F
pF	1.00E-12	F

Frequency

kHz	1.00E+03	Hz
MHz	1.00E+06	Hz

Speed

mph	5793638.4	m/s
kph	3.60E+06	m/s

Force

mN	0.001	N
kN	1000	N

Pressure

atm	101325	Pa
bar	1.00E+05	Pa
mbar	100	Pa
cmHg	1333	Pa
inHg	3386.28	Pa
inWC	249.061	Pa
mmHg	133.3223684	Pa
PSI	6894.757293	Pa
kPa	1000	Pa

Magnetic density

Gs	1.00E-04	T

Energy

Btu	1055.055853	J	
Btu39F	***1059.67***	***J***	***British***
Btu59F	***1054.8***	***J***	***British***
Btu60F	***1054.68***	***J***	***British***
kCal	4.19E+03	J	
cal	4.1868	J	
eV	1.60E-19	J	
kWh	3600000	J	

Temperature

Fah	C = (F - 32) * 5 / 9	Cel

Classic module
Basic calculations

Now that we're familiar with the main screen, let's do some basic calculations on PDAcalc classic using the default preferences:

The Problem Statement:	**You press:**	**PDAcalc classic Displays:**	**Remarks:**
3 + 4	3 + 4 [EXE]	3+4 7	Pretty straightforward.
(3+4) * 2	[CLS]* 2 [EXE]	ans*2 14	Use the previous result for the current calculation.
2^((3+4)*2)	[CLS] 2 ^ [ans] [EXE]	2^ans 16,384	ans key supplies the previous result in the equation.
(2^((3+4)*2))^(1/14)	[CLS] [ans] ^ (1 / 14) [EXE]	ans^(1/14) 2	Fractional exponents.
Same as above	[CLS] (2 ^ ((3 + 4)*2)) ^ (1/ 14) [EXE]	(2^((3+4)*2))^(1/14) 2	PDAcalc classic follows algebraic order of precedence when evaluating expressions.
Same as above, but with a deliberate error	delete a) from the expression in the scratchpad, then press [EXE]	1:')' expected	PDAcalc classic's interpreter does syntax error-catching.
(-1)^(1/2)	[CLS] [sqr] – 1) [EXE]	sqr(-1) 0 +1j	Imaginary numbers! Note that a prepended 'j' designates the imaginary part of a complex number PDAcalc classic.*
(0 + 1j) + (2+3j)	[CLS] +(2 + 3 j)	ans+(2+3j) 2+j4	Adding imaginary numbers
(2+4j) – (3+1j)	[CLS] –(3 + 1j)	ans-(3+1j) -1+j3	Subtracting imaginary numbers
-(1+5j) * (7+11j)	[CLS] *(7 + 11j)	ans*(7+11j) -40+j10	Multiplying imaginary numbers
(-62+24j) / (-1+3j)	[CLS] / (5 + 3j)	Ans/(5+3j) -5+5j	Dividing imaginary numbers
let A=4*5	[CLS] A = 4 * 5 [EXE]	A=4*5 20	Variable assignments.
A/3	[CLS] / 3 [EXE]	ans/3 6.67	20 / 3
A*3	[CLS] A * 3 [EXE]	A*3 60	20*3
let A=3; B=4	[CLS] A = 3 ; B = 4 [EXE]	A=3; B=4 4	Separate multiple variable assignments on the same line with a semicolon
sqr(A^2+B^2)	[CLS] [sqr] (A * A + B * B) [EXE]	sqr(A*A+B*B) 5	sqr(3^2+4^2) Entering the problem using the ^ key yields the same result. PDAcalc classic follows the algebraic order of precedence

---

Most math books use the postpended 'i' to designate the imaginary part of a complex number; e.g., sqr(-1) =1i. Engineering books use the prepended 'i' or 'j'. Math books use the 'j' or 'i' when a complex number is raised to the power 'e'; e.g., e^jb. In practice, electronic engineers (like myself), commonly use the 'j' instead of the 'i' since 'i' is used to designate current. It is a good practice to put parentheses around a complex number. This is not needed when adding or subtracting complex numbers but is needed when multiplying or dividing them. 4+5j*4-2j yields a different result than (4+5j)*(4-2j).

Worksheets

The items [Plot function] and [Solve equation] use worksheets, and share information across worksheets. Worksheets are little forms in which you enter parameters that will be used to create a script.

Plot function

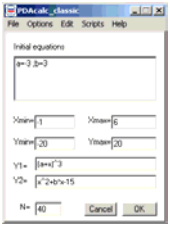


Figure 44

Select [Plot function] from the [Scripts] menu. You should see the screen on the left. The textbox at the top allows you to set initial conditions for the plot. In this example, 'x' is the unknown variable; 'a' and 'b' are fixed. We set their values here.

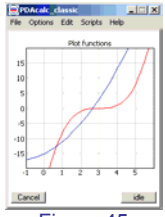


Figure 45

The next six values set the plot parameters. *Xmin, Ymin, Xmax,* and *Ymax,* set the plot boundaries. Using [Plot function], PDAcalc classic will plot 2 functions simultaneously (however, using a program, PDAcalc classic plots as many functions as you want). You enter each function on its own line at *y1*= and *y2*=.
Finally, *N* is the number of points used to plot a graph.

Press [OK] and a template script is created. Press [RUN] to execute the script resulting in the plot on the right. Notice that the two lines intersect twice within the plot's boundaries.

Return to the main screen. Press the [EDIT] key. The script that was created using the parameters from [Plot function] comes up. Using the [Save as] key, you can save this script under a different name in a different category.

We encourage you to modify scripts, parameters, etc. on PDAcalc classic to get a better understanding of how to use these features.

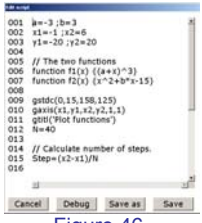


Figure 46

Solve Equation

Figure 47

Select [Solve equation] from [Scripts] menu. PDAcalc classic uses a numerical root-finding algorithm called the Newton-Raphson method. This method uses the numeric derivative of the function whose roots you're looking for, which is what the variable 'h' in *Init equation* is for. This method is fast and accurate, but it requires you to enter an initial guess in the *Guess:* textfield.

Notice that the equations on the left-hand side (or LHS) and right-hand side (RHS) of the equal sign on the *Equation to solve:* are lines *y1* and *y2* in the [Plot function] worksheet above. Changing equations *y1* and *y2* in that worksheet will change the LHS and RHS of this equation. Pressing [OK] will create a template script to solve for variable 'x'. If you'd rather solve for 'a', just change 'a=-3' to 'x=0' in the *Init equation:* textfield and 'x' to 'a' in the *Solve for variable:* textbox.

The program keeps trying to converge upon the root's real value until the error is within tolerance or the number of tries exceeds the limit. The error in the equation of our example is abs(x) < TOL. TOL is an internal value with a default of 1E-8.

Figure 48

When you press [OK], PDAcalc classic creates a template, just as it did for [Plot function], then solves the equation. Because the RHS is one function, and the LHS is another, [Solve equation] searches for where the two are equal (or intersect when plotted). Because the Newton-Raphson method converges on one real value for a root near the guess value, it can find just one root at a time.

And just as with [Plot function], pressing the [EDIT] key on the main screen exposes the template to you. Go back to [Solve equation]. Now change the guess value from 0 to 2, press [OK], and notice PDAcalc classic comes up with a different value for x. This is because the value of this guess was close to the second point of intersection. This brings us to an important point on the use of PDAcalc classic (and other graphing calculators) when analyzing functions by plotting them and solving for their roots: are there other values of x where the RHS and LHS functions intersect? How can we know?

Return to [Plot function], and change the plot boundaries to *Xmin=-5, Xmax=20, Ymin=-40, and Ymax=200, then press [OK].*

PDAcalc classic now reveals a third intersection that it didn't earlier because *our plot boundaries were not sufficiently great to capture the detail we needed* (careful: there are also instances in which one could say *our plot boundaries were not sufficiently large to capture the detail we needed*). Using [Plot function] and [Solve equation] together creates a powerful method for getting roots: a plot of the function(s) gives you good values for those initial guesses you have to put in. And the point? There is no substitute for having solid knowledge of the general behavior of the functions you're working with. That includes knowing how to sketch curves of these functions, given their parameters. Curve sketching is beyond the scope of this manual, but we've included a reference of points to consider when sketching curves (see Appendix H); and we wish to point out that good, instructive websites exist to allow you to learn or review the skill. Just type "curve sketching" into the textbox of your favorite Internet search engine, and browse the results. There's bound to be at least one website that meets your tastes and needs.

Putting It All Together

Wow! We've covered quite a bit of ground in just one chapter. As you can see, just with the main screen and default keyboard, PDAcalc classic puts tremendous capabilities and computing power at your fingertips, much of it just a few stylus taps away. We've intentionally glossed over many of them when introducing them to you because using them requires knowledge of other capabilities that are introduced later. In this chapter, we're going to put it all together: we're going to walk through examples of how to change PDAcalc classic's initialization file and keyboard; and we're going to walk through downloading a program from our online library, installing it, and running it.

Built-In Functions on PDAcalc classic

PDAcalc classic comes with more than 190 functions with applications in math, sciences, engineering, statistics and finance. From version 3.0 on, this includes functions that return the first and second derivatives of a function *f* at *x*. In this chapter, we look at some of those functions, as well as some of their applications. We will start with **Complex** number functions.

Complex

FUNCTION	REMARK	ARG	YIELDS	EXAMPLE	
				Input	Output
arg(x)	Returns the angle of x.	cplx	real	arg(1-1j)	-45
conj(x)	Returns conjugate of x.	cplx	cplx	conj(3-4j)	conj(3+4j)
pol(r,a)	Returns rectangular value of radius r and angle a.	real	cplx	pol(3, 45)	2.12+2.12j
Im(x)	Returns imaginary of x.	cplx	real	Im(3-4j)	-4
Re(x)	Returns real of x.	cplx	real	Re(3-4j)	3

Basic

OPERATOR or FUNCTION	REMARK	ARG	YIELD	EXAMPLE	
				Input	Output
%	If only percent is evaluated, returns decimal equivalent of percent. If percent is second part of arithmetic expression, takes percent of first part as second part, then evaluates expression.	Real	real	50%	.5
				10 + 7.5%	10. 75
abs(x)	Returns absolute value if x is real and returns magnitude if x is complex.	Real	abs(x)	abs(-5)	5
		cplx	mag(x)	abs(sqrt(-1))	1
cbrt(x)	Returns cube root.	Real	real	cbrt(-5)	-1.71
ceil(x)	Returns x if x is int, else returns next int > x	real	real	ceil(-2.5)	-2
exp(x)	Returns e^x. For complex z=x+yj, exp(z) = exp(x)*(cos(y)+1j*sin(y)).				
Floor(x)	Returns x if x is int, else returns next int < x	real	real	floor(-3.2)	-4
frac(x)	Returns fractional part of x	real	real	frac(-3.2)	-0.2
round(x)	Returns next int < x if fractional part of x between .0 and .49-bar, else next int > x	real	real	round(-3.7)	-4
sqr(x)	Returns square root.	Real	real	sqr(-9)	0+3j
		cplx	cplx	sqr(-5+12j)	2+3j
ln(x)	Returns natural logarithm of x.				
log(x)	Returns common logarithm (base 10) of x.				
max(a,b)	if a > b returns a else b.				
min(a,b)	if a < b returns a else b.				
mod(x,y)	Returns x modulo of y.				

Calculus

FUNCTION	REMARK	ARG	YIELDS
int(f,x1,x2)	Solves a definite integral	function f, x1, x2	
der1(f,x,h)	Returns the first derivative of function f at x.	function f, x, h	f'(x)
der2(f,x,h)	Returns the second derivative of function f at x.	function f, x, h	f''(x)

h is also known as dx, and the definition of a derivative comes from the difference quotient of function f. h then is the difference between two values of x, x0 and x1. The definition above yields the derivative of a function as h tends to zero. For PDAcalc classic, a small difference for h should be chosen, one that is close enough to zero to yield results accurate within the format precision on your PDAcalc classic while calculating derivatives.

Financial

FUNCTION	REMARK
fv(rate,nper,pmt,pv,type)	Returns the future value of an investment based on periodic, constant payments and a constant interest rate.
Inter(nper,pmt,pv,fv,t)	Returns the interest for an investment based on number of periods, periodic constant payments.
Nper(rate,pmt,pv,fv,type)	Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.
Pmt(rate,nper,pv,fv,type)	Calculates the payment for a loan based on constant payments and a constant interest rate.
	For the monthly payment on a $10,000 loan at an annual rate of 7 percent that you must pay off in 10 months: pmt(7%/12, 10, 10000, 0, 0) returns -$1,032.36
	For the same loan, if payments are due at the beginning of the period, the payment is: pmt(7%/12, 10, 10000, 0, 1) returns -$1,026.38
pv(rate,nper,pmt,fv,type)	Returns the present value of an investment. The present value is the total amount that a series of future payments is worth now. For example, when you borrow money, the loan amount is the present value to the lender.

Logical

FUNCTION	REMARK	EXAMPLE	
		Input	Output
and(h,h)	Bitwise AND.	and(4,6)	4
not(x)	Returns 0 if x!=0 else 1 .	not(1)	0
or(h,h)	Bitwise OR	or(4,6)	6
shl(h,b)	Bitwise shift left	shl(4,1)	8
shr(h,b)	Bitwise shift right	shr(4,1)	2
xor(h,h)	Bitwise EXCLUSIVE OR	xor(4,6)	10

Base conversion

OPERATOR	REMARK	ARG	EXAMPLE
#	Signify input is hexadecimal	hex integer	#A9C3
&	Signify input is binary	binary integer	&1010

Probability & Statistics

PDAcalc classic can do powerful statistical analyses on multivariate data elements. The data elements can be entered into a text file in PDAcalc classic's database for processing. The values in this text file can be converted using the sdata('file') function and stored in an array for processing. The array can also be filled using the sput(r,c,x) or scput(r,c,x) functions. It can handle a maximum of 512 rows (data elements) and a maximum of 5 columns (variables).
Let's look at some basic statistical functions by way of example. For illustrative purposes, let's say you have four data elements of 3 variables to analyze:

	Var 1	Var 2	Var 3
Element 1	2	3	4
Element 2	5	6	7
Element 3	8	9	10
Element 4	11	12	13

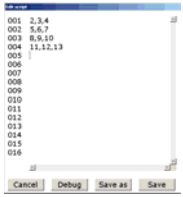


Figure 49

Select [New script] from the [Scripts] menu and enter the numbers as shown on the left. Then select [Save as] and save the file in the data category as Stat Data.

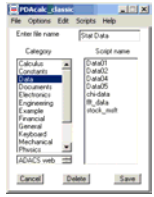


Figure 50

Select [New script] again and create the little script below to calculate the standard deviation of column two, the sum of column three and the quadratic regression.

```
001 // Stat test
002
003
004 // Load data in array
005 N=sdata('Stat Data')
006
007 // Calculate standard deviation
008 // of column two.
009 st2=stdev(2)
010
011 // Calculate sum of column three
012 sum3=ssum(3)
013
014 // The Quadratic regression,
015 // Column one holds x-values
016 // Column two holds y-values
017 a=sqrc(3)
018 b=sqrc(2)
019 c=sqrc(1)
020
021 // Set x-value
022 x=6
023
024 y=sqrv(x)
025 y1=a*x^2+b*x+c
026 // Notice that y and y1 contain the
027 // same values.
```

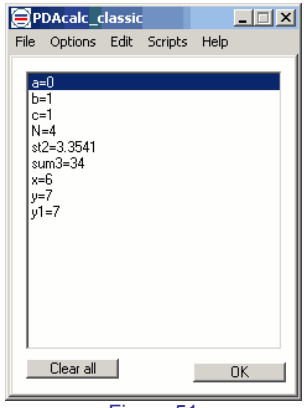


Figure 51

Remarks on a line are started with "//" and are not needed to run a script. Save the script and press [RUN] in the main screen. Next, select [VAR] from the main screen and verify the variables.

Statistical and probability functions

FUNCTION	REMARK
!	Factorial
fac(n)	Factorial
ftest(c1,n1,c2,c2)	Returns the result of an F-test. An F-test returns the one-tailed probability that the variances in column c1 and column c2 are not significantly different. Use this function to determine whether two samples have different variances. The arguments n1 and n2 indicate the number of data points in column c1 and c2. See F-test.pdb user program.
nCr(n,m)	Combination
nPr(n,m)	Permutation
rnd()	Generates a random number in the range [0, 1] with a uniform distribution and good statistical properties.
Rndn()	Uses the Polar Method to return a random number with a normal distribution and a mean of zero.
sdata(file)	Clears statistical variables and fill the statistical array with the data points of file. Make sure the file is a string by using single quotes. Example: rows=sdata('Data02') The file Data02 will be read from the data category. The return value contains the number of rows read.
ssave(file)	This function will save all data points of the statistical array in a file. The file will be stored in the data category. Example: sdata('Data02') ssave('Data10') If the file Data02 contains remarks they will not be stored in file Data10.
sput(r,c,v)	Store value v at row r and column c. When v is a complex use scadd(r,c,v) instead!!
scput(r,c,v)	Store value v at row r and column c. When v is a complex value the real part will be stored in column c , and the imaginary par
scget(r,c)	Returns a complex value from the array. After a scadd(3,2,v) the function v=scget(3,2) will return the complex value. Column two is used for the real values and column three is used for the imaginary values.
schi(c1,c2)	Chi-squared function. c1 - column of expected values. c2 - column of observed values.
scnorm(x,mu,sig)	Returns the cumulative standard normal distribution. (m=mu, sig=stdev)
scorr(c)	Returns the correlation between the values in column 1 and the valuesin column r.

sdata('rec')	Clear statistical variables and fill array with values of record 'rec'.
serre(c)	Returns the standard error of estimate.
serrr(c)	Returns the standard error of regression.
sget(r,c)	Get value at row r and column c.
smax(c)	Maximum value in column c.
smean(c)	Returns the mean. (Sum / N) of column c.
smin(c)	Minimum value in column c.
snorm(x,mu,sig)	Returns the standard normal distribution. (m=mu, sig=stdev)
splot(c1,c2,T)	Plot values in column c1 versus values in column c2 using T. T=0 line, T=1 diamonds, T=2 plus-signs.
sqrc(c)	Returns the coefficients for the quadatic regression. A=sqrc(3) B=sqrc(2) B=sqrc(1) See QuadReg.prc user program for and example.
sqrv(x)	Returns the value for the quadratic regression Y=A*X^2 + B*X + CSee QuadReg.prc user program for and example.
sregc(c)	Returns the regression coefficient of column 1 and column c.
sregl(c)	Plots the regression line for column 1 and column c.
srplot(Col,r1,r2,Type)	Plot the range starting at row r1 to row r2 of column Col. Type specifies the type of plot 0-line 1-diamond points 2-cross points 3-plus points. This function will clear the screen use the maximum size to draw the plot and use autoaxis labeling.
ssum(c)	Sum of values in column c.
stclr()	Clear statistical variables.
stdev(c)	Returns the population standard deviation sqr(var()) of column c.
Stdev(c)	Returns the sample standard deviation sqr(Var()) of column c.
svar(c)	Returns the population variance (1/N * (A(n)-mean)^2) of column c.
sVar(c)	Returns the sample variance (1/(N-1) * (A(n)-mean)^2) of column c.
sxy(c)	Returns A(1,n) * A(c,n) .
syint(c)	Returns y-intersect.
ttest(c1,n1,c2,n2,tail,type)	Returns the probability associated with a Student's t-Test. Use ttest to determine whether two samples are likely to have come from the same two underlying populations that have the same mean. The arguments n1 and n2 indicate the number of data points in column c1 and c2. Tail specifies the number of distribution tails. If tails = 1, ttest uses the one-tailed distribution. If tails = 2, ttest uses the two-tailed distribution. Type is the kind of t-Test to perform and should be set to two. See Student_ttest.pdb user program.

User-Defined Functions

The screenshot on the left shows how you can define your own function. User functions cannot be defined in the scratchpad. Line seven will calculate the derivative of the function f at point x. Since the derivative is a commonly used function. PDAcalc classic has a build-in function for calculating the derivative. Enter der1('f',x,h) at line seven and the same result will be shown.

Figure 52

Due to speed considerations, local variables in user-defined functions are not stored on a stack. This means that recursion, calling the same function within a function, is not permitted.
User-defined functions can also be used when plotting functions using the plot function worksheet.

Figure 53

Figure 54

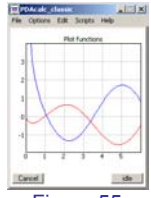


Figure 55

Remember that user functions cannot be defined in the scratchpad. The first three lines of the initial equations are copied into the scratchpad which is why there are a couple of blank lines between the first equation and the function *f(x)*.

Notice that *y2* uses der2('f',x,h). Please don't forget that the first three lines in a program, the scratchpad, are processed differently than the rest of the program. To review, when you press [EXE] only the scratchpad is evaluated. When you press [RUN], first the scratchpad is evaluated, then PDAcalc classic runs the loaded program. So how does this relate to user-defined functions? You have to change the function in the program, then reload the program. Just press [EDIT], change the function, then load the program and press [RUN].

Graphics

Graphics on PDAcalc classic allows you to see how input has been transformed to output. The visual display of quantitative information[1] makes large amounts of data or complex data understandable. On PDAcalc classic it allows you, among other things, to investigate the behavior of functions, to see patterns in data, or to draw diagrams that illustrate concepts. Below is PDAcalc classic's graphics screen:

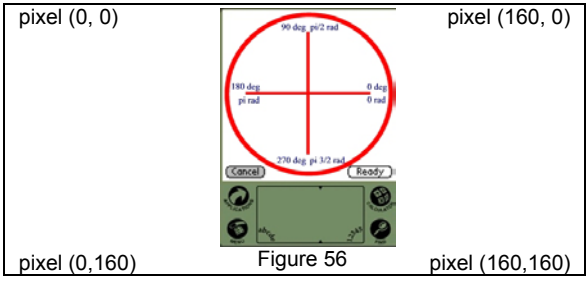


Figure 56

PDAcalc classic graphics coordinate system superimposed in red is PDAcalc classic's angular coordinate system (see the garc(x,y,r,a1,a2) example in Appendix D).

Graphics are created by turning pixels on the touch screen off or on; those that are turned on are set to a gray tone or color. The touch screen has an area of 160x160 pixels. PDAcalc classic uses the entire touch screen as its graphics screen, and it follows the palm platform convention for defining pixel coordinates. Pixel (0, 0) is the uppermost left pixel, pixel (0,160) is the lowermost left pixel, pixel (160,160) is the lowermost right pixel; and pixel (160, 0) is the uppermost right pixel. PDAcalc classic has many graphics functions to render objects on the graphics screen. These objects include lines, arcs, circles, rectangles, axes, and text. PDAcalc classic uses a table of 16 colors. Colors are set using gsetcol(idx,r,g,b). idx indicates the index in the table.

[1] Yes, this phrase is lifted from the title of the book, The Visual Display of Quantitative Information, by Edward Tufte, whose 3-volume treatment on rendering information into visuals is highly recommended.

Default Colors

idx	color	red	green	blue
0	white	255	255	255
1	red	255	0	0
2	green	0	210	0
3	blue	0	0	255
4	cyan	0	255	255
5	magenta	255	0	255
6	yellow	255	255	0
7	gray	180	180	180
8	light blue	210	210	255
9	light gray	210	210	210
10	unassigned (black)	0	0	0
11	unassigned (black)	0	0	0
12	unassigned (black)	0	0	0
13	unassigned (black)	0	0	0
14	unassigned (black)	0	0	0
15	unassigned (black)	0	0	0

On startup, PDAcalc classic sets these colors for its color table. you can change them using gsetcol(idx,r,g,b)

By default, gline(x1,y1,x2,y2) takes the color whose index = 1 and renders a line in the graphics screen of that color, gline2(x1,y1,x2,y2) takes the color whose index = 2, and so on. Five lines that can be made and manipulated independently of each other using gmove(), glin() and gline(). You select colors in the color table using selcol(idx). See the fft example program.

Graphics Examples

Let's draw some graphics, if only to get the feel of PDAcalc classic's graphics capabilities. Bring up the Scripts. menu item, select Edit script; and in the program database display, choose New.

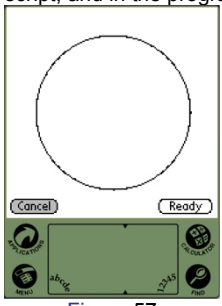


Figure 57

A blank text file comes up. In the first line write the name that you want to call this file (I've called mine "GraphicsFun", one word, no spaces). Leave the next 3 lines blank. On the fifth line, write greset (); and on the next line, write gcir(80,80,60). Press [Load] at the bottom of the screen. If everything went ok, PDAcalc classic will print "Program loaded successfully". Press [RUN] and watch as PDAcalc classic draws as shown in figure 58

Okay, so maybe I misnamed the file; but, hey: great things are built from small parts. We'll see these functions later in the next chapter.

greset()	Resets the graphics mapping to the default of 160 by 160 pixels.
gcir(col,x,y,r)	Draw circle at x,y with radius r using color index col.

The graphics functions used in "GraphicsFun"

Let's spruce it up a bit by adding lines and color (even if your palm PDA doesn't support color, you might want to walk through these enhancements). Edit the "Graphics Fun" file. Press [EDIT] on the text screen and enter these lines at the end of the file:

```
gline(80,80,0,0)
gline2(80,80,0,80)
gline3(80,80,0,160)
gline4(80,80,80,160)
gline5(80,80,160,160)
```

Press [Save]. Once PDAcalc classic returns you to the text screen, press [RUN]. On a color Palm PDA, you should see:

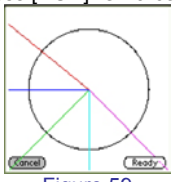


Figure 59

Going counterclockwise from Palm touchscreen coordinates (0,0), PDAcalc classic draws lines from the center of the circle out. Notice that lines are drawn through the buttons [Cancel] and [Ready]. This is because we have not set the device coordinates for the graph; making all pixels on the touchscreen available.

How do we fix this? With the gstdc(x1,y1,x2,y2) function. For illustrative purposes, let's give our graph a title with the gtitl('StrV',v) function, as well as draw an axis with gaxis(x1,y1,x2,y2,gx,gy).

Between greset () and gcir(1,80,80,60), if you enter:

```
gstdc(10,10,160,160)
gaxis(10,10,160,160,80,80)
gtitl('Graphics Fun',0)
```

Figure 60

In this example, we shifted the graph down and right by defining device coordinates *Xmin* and *Ymin* as 10 pixels down and 10 right. Ten down was necessary to write the title. But we didn't get the output we'd hoped for; the buttons [Cancel] and [Ready] are still

exposed to our graphics objects, and it's obvious with the axes drawn that we have an offset that we didn't expect.
Fixing this requires that we further change the device coordinates for the graph. Let's try:
gstdc(10,10,140,140)
Then, to center the axes, we use *Xmax-Xmin* and *Ymax-Ymin*:
gaxis(10,10,140,140,65,65)
Finally, we change the coordinates of our graphics objects:

```
gcir(1,75,75,60)
gline(75,75,0,0)
gline2(75,75,0,75)
gline3(75,75,0,150)
gline4(75,75,75,150)
gline5(75,75,150,150)
```

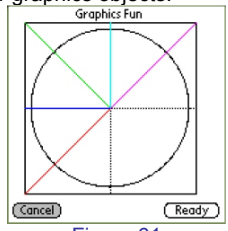


Figure 61

Other functions used in graphics on PDAcalc classic include:

garc(x,y,r,a1,a2)	Draws an arc angle, beginning at a start angle a1, and ending at angle a2. The angle can be in radians or degrees. Use strad() or stdeg() to set for radians or degrees.
gaxis(x1,y1,x2,y2,gx,gy)	Set axis with minimum values x1 and y1 and maximum value x2 and y2. When gx equals zero no grid lines will be shown for the x-value. When gx equals one the grid lines will be shown. We will leave it up to the user to experiment with gy if wondering.
gclrs()	Clear graphics screen. The following functions only appear in the dropdown function list on the main screen.
gcont()	Wait until continue button is pressed.
gcprt(x,y,'StrV',v)	Draw text 'StrV' and value v at centered at x,y position.
gfcir(x,y,r)	Fills a circle at x,y with radius r.
ghlin(y)	Draw horizontal line at y position.
grect(x1,y1,x2,y2)	Draw rectangle.
grprt(x,y,'StrV',v)	Draw text 'StrV' and value v at right x,y position.
gvlin(x)	Draw vertical line at x position.
gvprt(x,y, 'StrV',v)	Print text on graphical screen vertical.
gselcol(idx)	Select a color from the color table.
gsetcol(idx,r,g,b)	Sets a color in the color table. The line colors used in the graphs use idx 1-5. Index 0 is white and index 15 is black

For a complete list of graphics functions, refer to **Appendix D**.

Because the power and usefulness of graphics on PDAcalc classic become apparent either when graphing functions or programming, we will put off doing other graphing examples until the next chapter, which deals with programming on PDAcalc classic. However, as we saw when plotting functions earlier, we think it is important to stress once again that on a graphing calculator, seeing is not always believing.

3D functions

```
001 ampl=8// Amplitude
002 // Change values to rotate graph
003 rotLR=80;rotUD=-75
004
005
006
007 // Change user funtion below
008 function f(x,y)
009 {
010   ampl*sin(sqrt(x*x+y*y))
011 }
012
013 // Set to radians
014 strad()
015
016 gcprt(90,10,'The power of PDAcalc classic')
017
018 // Indicate where to put graph on screen
019 gstdc(15,25,159,80)
020
021 // Maximum x and y values for graph.
022 // Make bigger that used in the function for
023 // rotation.
024 ginit3d(-12,-12,12,12)
025
026 gtitl('3D plot using fplot3d()')
027
028 // rotate around x,y,z axis
029 grotate3d(rotLR,rotUD)
030
031 // Plot function using the max.
032 // and min. values.
033 fplot3d('f',-8,-8,8,8)
034
035 gstdc(15,95,159,130)
036
037 // The last argument is zero so
038 // the values at the y-axis are not
039 // shown. 040 gaxis(-8,-ampl,8,ampl,1,0)
041 gtitl('Cross section at y=0')
042 x=-8
043 gmove2(x,f(x,0))
044 while(x<=8)
045 {
046   glin2(x,f(x,0))
047   x=x+0.2
048 }
```

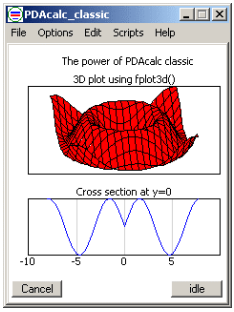


Figure 62

rotUD=-120

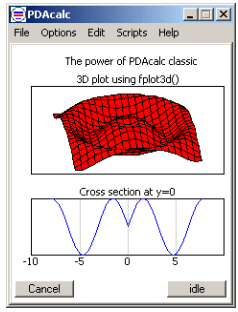


Figure 63

rotUD=-75

The function ginit3d() initializes PDAcalc classic for 3D graphing and sets the minimum and maximum values of an axis used to draw functions. The function fplot3d() is used to plot the function.

Notice the maximum and minimum values of this function are smaller then in ginit3d() since rotating the graph requires more space. The function fplot3d() first checks if 3D graphing was initialized already. When it was not initialized fplot3d() will initialize 3D graphing using default values as shown it the little script on the right. Next fplot3d() will allocate memory to store 20 X 20 values.

```
001 ampl=150// Amplitude
002
003
004 function f(x,y)
005 {
006   // Change user funtion belo
w
007   ampl*cos(sqrt(x*x+y*y))
008 }
009 fplot3d('f',-360,-360,360,360)
010 gtitl('3D function')
```

The range of x-values and the range of y-values will be divided into to 20 equally spaced points. A total of 400 points. The user function will be called with the x and y-value for each point and the return value of the function is stored in memory. After all 400 points are calculate the draw the surface plot using highly optimized routines.

```
001 // Set rotation parameters
002 rotLR=30;rotUD=-25
003 N=100// Number of steps
004
005
006 gstdc(5,15,159,140)
007 ginit3d(-1.2,-1.2,1.2,1.2)
008 gtitl('Circles in 3D planes')
009
010 // Rotate around the axis.
011 grotate3d(rotLR,rotUD)
012
013 // Notice the last argument is zero
014 gaxis3d(15,1,1,1,0)
015 gaxis3d(15,-1,-1,-1,0)
016
017 max=360
018 step=max/N
019 t=0
020 s=sin(t)
021 c=cos(t)
022
023 // Set initial points.
024 gmove3d(1,c,0,s)// xz-plane
025 gmove3d(2,c,s,0)// xy plane
026 gmove3d(3,0,c,s)// yz plane
027
028 whilet<=max)
029 {
030   t=t+step
031   s=sin(t)
032   c=cos(t)
033
034   // Draw lines
035   gline3d(1,c,0,s)
036   gline3d(2,c,s,0)
037   gline3d(3,0,c,s)
038 }
```

To draw lines in 3D space use the gmove3d(idx,x,y,z) and the gline3d(idx,x,y,z) functions. Use gmove3d(idx,x,y,z) to set the starting point of each line. Different lines have a different index, first argument, which also determents the color of the line. These colors can be changed using the **gsetcol()** function if needed.

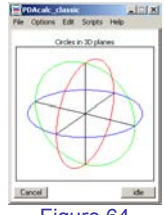


Figure 64

We encourage the user to remark parts of the script when it is not clear how this script works. For example you can put the remark sign '//' at the beginnings of lines 25, 26,36 and 37 to only plot a circle in the xz-plane. Lines starting with remark signs are ignored by the interpreter. The sine and cosine functions are used to draw the circles in the three different planes. We use three planes in 3D space instead of only one plane, the xy plane, in 2D space.

Classic built-in functions

Basic functions

+

This operator can also be used to add strings and numbers,
result='Volume='+5.456+' liters'

After values are converted to strings adding them will concatenate the string and not add the values as shown below.

Result = A + B		
Result	**A**	**B**
string 'Vol=123'	string 'Vol='	string '123'
string '45 kHz'	value 45	string ' kHz'
string 'W=12.56'	string 'W='	value 12.56
value 47.12	value 34.67	value 12.45

Note:
If you want to use the minus sign to add a negative number put it between parentheses as shown below.
val='Val='+(-x)

---

abs(x)

Returns absolute value if x is real and returns magnitude if x is complex.

a=abs(4.2) // a=4
b=abs(3+4j) // b=5

See also: floor , ceil , round

ans

This variable always contains the result of the last calculation.

Press [CLR] in the main screen to clear the scratchpad and press +5 [EXE] to add 5 to the previous answer. Press [EXE] again and see what happens.

---

cbrt(x)

Returns cube root.

a=cbrt(27) // a=3 3*3*3=27

See also: sqr

---

ceil(x)

Returns x if x is integer. If x=4.1 ceil(x) returns 5.

a=ceil(4.1) // a=5
b=ceil(-3.2) // b=3

See also: floor , round

---

exp(x)

Returns e^x. For complex z=x+jy,
exp(z)=exp(x)(cos(y)+j*sin(y)).*

Calculates the exponent of e (the Neperian or Natural logarithm base)

a=exp(3) // a=20.085537
b=ln(20.085537) // b=3

See also: log , ln

---

floor(x)

Returns x if x is integer. If x=4.9 floor(x) returns 4.

a=floor(4.9) // a=4
b=floor(-2.3) // b=-3

See also: ceil , round

---

fmt(t,w,p,tr)

Set display format t: 0-float, 1-sci, 2-eng, 3-sym, 4-hex, 5-bin, 6-oct, 7-pol, 8-date, 9-sexagesimal w: width of number (0-15) p: precision of number (0-15) tr: trailing zeros. (0 or 1)

Float The native data format.

Scientific When a number cannot be displayed using width and precision settings, it is displayed in scientific format. An exponent will be used to show small numbers instead of leading zeros. With a precision setting of 7 the number 0.0001232456 will be shown as 1.23456E-04

Engineering When a number cannot be displayed using width and precision settings, it is displayed in engineering format. Enter 5.11e8 for example in the scratchpad and press [exe]. 511E6 will displayed. The exponent, in this case 6, will always be a multiple of three. The symbol format will show an SI postfix instead of E6.

Symbol When a number cannot be displayed using the width and precision settings, it is displayed in symbol format. This is especially important when numbers are rendered on the graph screen in order to make sure all numbers are printed using the same space.

Name	SI Postfix	Power of 10
femto	f	-15
pico	p	-12
nano	n	-9
micro	u	-6

milli	m	-3
kilo	K,k	3
mega	M	6
	G	9
tera	T	12

Hexadecimal Positive integers rendered in base 16 format

Binary Positive integers rendered in base 2 format.

Octal Positive integers rendered in base 8 format.

Polar Complex values converted to magnitude and angle.

Date Positive values are converted to dates.

Sexagesimal Mixed decimal fractions rendered in H.M.S format.

---

frac(n)

Calculates the factorial.

```
          / n*(n-1)...2*1   n=1,2,...
frac(n)=|
          \ 1               n=0
```

So, for example, frac(4)=4*3*2*1=24

---

ln(x)

Returns natural logarithm of x.

a=ln(20.085537) // a=3 2.718282^3=20.085537

See also: exp , log

---

log(x)

Returns common logarithm (base 10) of x.

a=log(1000) // a=3 10^3=1000

See also: ln , exp

---

max(a,b)

if a > b returns a else b.

y=max(5,8) // y=8

See also: min , sat

---

min(a,b)

if a < b returns a else b.

y=min(3,8) // y=3

See also: max , sat

---

mod(x,y)

*Returns the remainder of dividing the dividend (x) by the divisor (y). The reminder (r) is defined as: x = i * y + r, for some integer i. If y is non-zero, r has the same sign as x and a magnitude less than the magnitude of y.*

Return the remainder of x/y
a=mod(10,4) // a=2

See also: gcd

---

round(x)

Returns closed integer. If x=4.4 round(x) return 4. If x=4.6 round(x) returns 5

Returns the rounded value of x.

a=round(4.4) // a=4
b=round(4.6) // b=5

See also: floor , ceil

---

sqr(x)

Returns square root.

a=sqr(16) // a=4

See also: cbrt

---

sqrt(x)

Returns square root of x

Color functions

---

gselcol(idx)

Select a color.

Default colors				
5	magenta	255	0	255
6	yellow	255	255	0

6	yellow	255	255	0
7	gray	180	180	180
8	light blue	210	210	255
9	light gray	210	210	210
10	unassigned (black)	0	0	0
11	unassigned (black)			
12	unassigned (black)			
13	unassigned (black)			
14	unassigned (black)			
15	unassigned (black)			

See also: gsetcol

---

gsetcol(idx,r,g,b)

Change the color of index idx.

Use this function to change the amount of red,green and blue of the color idx. See gselcol for a list of default colors.

See also: gselcol

Complex functions

---

arg(cplx)

returns the angle of a complex number

a=arg(3+3i) // a=0.785398

a=a*(360/(2*pi)) // convert to degrees. a=45

See also: pol

---

conj(cplx)

return the complex conjugate.

a=conj(3+4j) // a=3-4j

Im(cplx)

Return the imaginary part of a complex number.

a=3+4j
b=Im(a) // b=4

See also: Re

---

pol(r,a)

returns the complex number of a vector with a length

c=3+4j
strad() // Set radians
r=abs(c) // r=sqr(3*3+4*4)
a=arg(c) // a=atan(3/4)
p=pol(r,a) // p=3+4j
See also: arg , abs

Re(cplx)

return the real part of a complex number.

a=3+4j
b=Re(a) // b=3

See also: Im

Conversion functions

---

cel(T)

Converts temperature from Fahrenheit to Celcius.

a=cel(77) // a=25

See also: fah

---

deg(a)

Converts radians to degrees

a=deg(pi) // a=180

See also: rad

---

dms(hr,min,sec)

Converts sexagesimal value to decimal value.

---

fah(T)

Converts temperature from Celcius to Fahrenheit.

a=fah(25) // a=77

See also: cel

---

met(yr,ft,in,p)

Converts yards, feet, inches

Calculates yr * 0.9144 + ft * 0.3048 + (in / p) * 0.0254

If p equals zero it will be set to 1.0 to avoid devision by zero.

To convert 1/16 inch to meters use met(0,0,1,16)

---

rad(a)

Converts degrees to radians.

a=rad(180) // a=pi

See also: deg

Date functions

---

date(mm,dd,yyyy,text)

Enter date and this function will return the time in seconds since Jan. 01 1904. When text is not equal to zero a dialog box will appear to select a date.

Use this function to ask for a date or convert a date to seconds since Jan. 01 1904. When the arguments mm,dd and year are zero the current date is used.

See also: days , time , weeks

---

days(sec)

Returns the number of days since Jan. 01 1904.

See also: time , weeks

---

time()

Returns the current time in seconds since Jan. 01 1904.

See also: days , weeks

---

weeks(sec)

Returns the number of weeks since Jan. 01 1904.

See also: days , time

Financial functions

---

fv(rate,nper,pmt,pv,type)

Returns the future value of an investment based on periodic, constant payments and a constant interest rate.

rate	is the interest rate per period.
nper	is the total number of payment periods in an annuity.
pmt	is the payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes.
pv	is the present value, or the lump-sum amount that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero), and you must include the pmt argument.
type	is the number 0 or 1 and indicates when payments are due.

Suppose you want to save money for a special project occurring a year from now. You deposit $1,000 into a savings account that earns 6 percent annual interest compounded monthly. You plan to deposit $100 at the beginning of every month for the next 12 months. How much money will be in the account at the end of 12 months?
a=fv(6%/12, 12, -100, -1000, 1) // a=2301.40183

See also: inter , nper , pmt , pv

---

inter(nper,pmt,pv,fv,t)

Returns the interest for an investment based on number of periods, periodic constant payments.

nper	is the total number of payment periods in an annuity.
pmt	is the payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes.
pv	is the present value, or the lump-sum amount that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero), and you must include the pmt argument.
fv	is the future value, or a cash balance you want to attain after the last payment is made.
type	is the number 0 or 1 and indicates when payments are due.

See also: fv , nper , pmt , pv

---

nper(rate,pmt,pv,fv,type)

Returns the number of periods for an investment based on

periodic, constant payments and a constant interest rate.

rate	is the interest rate per period.
pmt	is the payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes.
pv	is the present value, or the lump-sum amount that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero), and you must include the pmt argument.
fv	is the future value, or a cash balance you want to attain after the last payment is made.
type	is the number 0 or 1 and indicates when payments are due.

See also: fv , inter , pmt , pv

---

pmt(rate,nper,pv,fv,type)

Calculates the payment for a loan based on constant payments and a constant interest rate.

rate	is the interest rate per period.
fv	is the future value, or a cash balance you want to attain after the last payment is made.

fv	is the future value, or a cash balance you want to attain after the last payment is made.
type	is the number 0 or 1 and indicates when payments are due.

The following formula returns the monthly payment on a $10,000 loan at an annual rate of 7 percent that you must pay off in 10 months pmt(7%/12, 10, 10000, 0, 0) equals -$1,032.36. For the same loan, if payments are due at the beginning of the period, the payment is: pmt(7%/12, 10, 10000, 0, 1) equals -1026.38

See also: fv , inter , nper , pv

---

pv(rate,nper,pmt,fv,type)

Returns the present value of an investment. The present value is the total amount that a series of future payments is worth now. For example, when you borrow money, the loan amount is the present value to the lender.

rate	is the interest rate per period.
nper	is the total number of payment periods in an annuity.
pmt	is the payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes.
fv	is the future value, or a cash balance you want to attain after the last payment is made.
type	is the number 0 or 1 and indicates when payments are due.

See also: fv , inter , nper , pmt

Flow control functions

---

else

Optional to the if command. executes the commands between brackets when the condition for the if statement is false. Note that the selection flow of an if-then-else statement lets the condition decide which of two sets of program lines (those within the curly brackets following the if(cond) or those within the curly brackets following the else will be executed.

Eample:
This little script will change the key at Change key at row 2 column 3.

```
001 // if example
002 Dia=10
003
004 // Change key at row 2 column 3
005 key(23,'&Dia')
006
007 // Check if key was pressed
008 if(iskey('Dia'))
009 {
010 result('Diameter='+Dia*pi)
011 }
012 else
013 {
014 result('Dia not pressed')
015 }
```

After loading this script press [RUN] and the key will change. Next press the Dia key and the result will be shown. When an other key is pressed the script line between the else brackets will be executed.

See also: if , while

---

error(condition,text)

The script will terminate when the condition is true. The text will be shown on the result line of the main screen.

This function is the same as:

```
001 // error example
002
003
004 if(condition)
005 {
006   exit(text)
007 }
```

See also: exit

---

exit(exitVal)

Use this function to exit a script. The variable exitVal will be shown on the result line of the main screen.

See also: error

---

if(condition)

If (condition) is true, execute program lines within the curly brackets. Note that when the if command is used by itself, the condition decides only if additional program lines in your program (those within the curly brackets) will be executed.

Eample:
This little script will change the key at
Change key at row 2 column 3.

```
001 // if example
002 Dia=10
003
004 // Change key at row 2 column 3
005 key(23,'&Dia')
006
007 // Check if key was pressed
008 if(iskey('Dia'))
009 {
010   result('Diameter='+Dia*pi)
011 }
012 else
```

```
013 {
014   result('Dia not pressed')
015 }
```

After loading this script press [RUN] and the key will change. Next press the Dia key and the result will be shown. When an other key is pressed the script line between the else brackets will be executed.

See also: else , while

---

init

Returns one only the first time the script is run.

Returns one only the first time the script is run. Use this function to initialize variables.

---

while(condition)

while (condition) is true, execute the program lines between the curly brackets. program flow enters the while-loop, and continues flowing through it in a loop until the exit condition is met (that is, the while(condition) becomes false).

Example:
Little scripts to draw circles in the middle of the screen.

```
001 // while example
002
003
004
005 deltaR=8
006 r=deltaR
007 col=0
008 while(r<=64)
009 {
010   gcir(col,80,80,r)
011   r=r+deltaR
012   col=col+1
013 }
```

See also: if , else

---

fplot3d(f,x1,y1,x2,y2)

Plot a function in 3D space. The function name needs to be between single quotes!
Example

```
001 ampl=150// Amplitude
002
003
004 function f(x,y)
005 {
006   // Change user funtion below
007   ampl*cos(sqrt(x*x+y*y))
008 }
009
010 // The fplot3d() checks if a graphical
011 // area was initialized already.
012 // If no graphical was initialized it
013 // will initialize the default area plus
014 // default rotation for you.
015 fplot3d('f',-360,-360,360,360)
016
017 // Put the gtitl() after fplot3d() because
018 // gtitl() should only be called after
019 // gaxis() which is called in fplot3d()
020 gtitl('3D function')
```

Note:
The function has to be declared in the program area and not in the scratchpad.

---

garc(x,y,r,a1,a2)

Draws an arc angle. Use strad() or stdeg() to set radians or degrees.

Use this function to draw an arc at position x,y a radius of r and starting at angle a1 to angle a2

See also: gcir , gfcir , strad , stdeg

---

gaxis(x1,y1,x2,y2,gx,gy)

Set min. and max. values of an axis used to draw functions.

The arguments gx and gy can be set to zero or one.
When set to zero the values at the axis is not shown.

Example
Little script to plot a function.

```
001 a=-3;b=3
002 x1=-4;x2=4
003 y1=-0.5;y2=1
004
005 // The function
006 function f1(x){sinc(x)}
007
008 gstdc(0,15,158,125)
009 gaxis(x1,y1,x2,y2,1,1)
010 gtitl('sinc functions')
011 N=40
012
013 // Calculate number of steps.
014 Step=(x2-x1)/N
015
016 // Set initial points
017 x=x1
018 gmove(x,f1(x))
019
020 // Connect points
021 while(x<=x2)
022 {
023 glin(x,f1(x))
024 x=x+Step
025 }
```

Note
Use the plot function from the menu to generate a script to plot a function. After the script is generated you can edit the script.

See also: gstdc

---

gaxis3d(col,x,y,z,t)

Draws the 3D axis. If t equals zero the end values are not shown.

See also: ginit3d , gmove3d , gline3d

---

gcir(col,x,y,r)

Draw circle at x,y with radius r using the color index col.

gcir(2,80,80,20) will draw a circle at the centre of the graphics screen with a radius of 20 using color 2. The default color for index 2 is blue. This can be changed using the gsetcol function.

See also: garc , gfcir

---

gclrs()

Clears the graphics screen.

See also: greset

---

gcont()

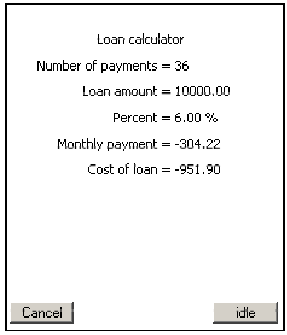
Wait until continue button is pressed.

Calling this function will stop the execution of the script and wait till the user taps on the continue button on the graphics screen.

---

gcprt(x,y,strV)

Draws the values or text of strV at y-position centered around the x-position.



A little example script using different text functions:

```
001 per=6%
002 N=36// Number of payments
003 loan=10000// loan amount
004
005 y=20
006 // Calculate monthly payment
007 mp=pmt(per/12,N,loan,0,0)
008
009 // Print text a y-position centered at x-position.
010 gcprt(80,y,'Loan calculator')
011 y=y+15
012 fmt(0,6,2,0)
013
014 // Print text ending first argument at position 100
015 grprt(100,y,'Number of payments=',N)
016
017 y=y+15
018 fmt(0,6,2,1)
019 grprt(100,y,'Loan amount=',loan)
020
021 y=y+15
022 grprt(100,y,'Percent=',per)
023
024 y=y+15
025 grprt(100,y,'Monthly payment=',mp)
026
027 // Calculate cost of loan
028 cost=loan+mp*N
029 y=y+15
030 grprt(100,y,'Cost of loan=',cost)
```

See also: gprt , gtadd , glprt

---

gfcir(x,y,r)

Fills a circle at x,y with radius r.

gfcir(2,80,80,20) will fill a circle at the centre of the graphics screen with a radius of 20 using color 2. The default color for index 2 is blue. This can be changed using the gsetcol function.

See also: gsetcol , gcir

---

ghlin(y)

Draw horizontal line at y position.

---

ginit3d(x1,y1,x2,y2)

Sets the minimum and maximum values for the graphing area.

Example
This example script will draw three circles in 3D-space.

```
001 // Set rotation parameters
002 rotLR=30;rotUD=-25
003 N=100// Number of steps
004
005
006 gstdc(5,15,159,140)
007 ginit3d(-1.2,-1.2,1.2,1.2)
008 gtitl('Circles in 3D planes')
009
010 // Rotate around the axis.
011 grotate3d(rotLR,rotUD)
012
013 // Notice the last argument is zero
014 gaxis3d(15,1,1,1,0)
015 gaxis3d(15,-1,-1,-1,0)
016
017 max=360
018 step=max/N
019 t=0
```

```
020 s=sin(t)
021 c=cos(t)
022
023 // Set initial points.
024 gmove3d(1,c,0,s)// xz-plane
025 gmove3d(2,s,c,0)// xy plane
026 gmove3d(3,0,s,c)// yz plane
027
028 while(t<=max)
029 {
030   t=t+step
031   s=sin(t)
032   c=cos(t)
033
034   // Draw lines
035   gline3d(1,c,0,s)
036   gline3d(2,s,c,0)
037   gline3d(3,0,s,c)
038 }
```

See also: gmove3d , gline3d , grotate3d

---

glin(x,y)

Draw line from previous position to x,y (line 1).

---

glin2(x,y)

Draw line from previous position to x,y (line 2).

---

glin3(x,y)

Draw line from previous position to x,y (line 3).

---

glin4(x,y)

Draw line from previous position to x,y (line 4).

---

glin5(x,y)

Draw line from previous position to x,y (line 5).

---

gline(x1,y1,x2,y2)

Draw line from x1,y1 to x2,y2.

---

gline2(x1,y1,x2,y2)

Draw line from x1,y1 to x2,y2.

---

gline3(x1,y1,x2,y2)

Draw line from x1,y1 to x2,y2.

---

gline3d(idx,x,y,z)

Draws a line from the current position to the x,y,z coordinates. Use gmove3d to set the start point of the line.

The x,y,z coordinates are converted to 2D space and become the current position for the line idx.

See also: gmove3d

---

gline4(x1,y1,x2,y2)

Draw line from x1,y1 to x2,y2.

---

gline5(x1,y1,x2,y2)

Draw line from x1,y1 to x2,y2.

---

glprt(strV)

Put string or value, strV, at next line.
See also: gprt , gcprt

---

gmove(x,y)

Move to start position x,y (line 1).

---

gmove2(x,y)

Move to start position x,y (line 2).

---

gmove3(x,y)

Move to start position x,y (line 3).

---

gmove3d(idx,x,y,z)

Sets the current point of line idx. Use gline3d draw a line from the current position.

See also: gline3d , ginit3d , gaxis3d

---

gmove4(x,y)

Move to start position x,y (line 4).

---

gmove5(x,y)

Move to start position x,y (line 5).

---

gpersp3d(dist)

Apply a perspective transformation.

Perspective transformations make objects in distance look smaller than object closer to the screen.

---

gpnt(x,y,t)

Draw symbol t at x,y.

Symbol: 1 - diamond, 2 - star, 3 - plus

See also: gpnt3d

---

gpnt3d(col_idx,x,y,z,t)

Draws symbol t at position x,y,z using the color index col_idx.

Draws symbol t at position x,y,z using the color index col_idx.

Symbol: 1 - diamond, 2 - star, 3 - plus

See also: gprt

---

gprbar(x,Min,Max,Y)

Use this function to plot a process bar. First initialize the bar with the minimum and maximum values and Y which determines the position on the screen. For updating the process bar use zero as the minimum, maximum and Y value.

---

gprt(x,y,strV)

Draws the values or text of strV beginning at the x,y position.

See gcprt for an example script.

See also: glprt , gtadd , gcprt

---

gprt3d(col,x,y,z,strV)

Draws the values or text of strV beginning at the x,y,z position.

See also: gprt

---

grect(x1,y1,x2,y2)

Draw rectangle.

---

greset()

Resets the graphics mapping to the default of 160 by 160 pixels.

---

grotate3d(rotLR,rotUD)

Sets rotation in 3D-space. rotLR set the left right rotation and rotUD sets the up down rotation.

See ginit3d for an example script.

See also: ginit3d

---

grprt(x,y,v1,v2)

Draws the values or text of v1 and v2 at y-position with the last character of v1 at x-position.

Use this function to line up equal signs for example.
See gcprt for an example script.

See also: gprt , gcprt

---

gstdc(x1,y1,x2,y2)

Set device coordinates for graph all values must be between 1 and 160.

Only supported for compatility reasons. Starting at version 1.60 use subplot() instead.

Example
gstdc(10,15,100,140) // Define the area to be used for a graph.
gaxis(-5,-2,4,3,1,0) // Set the minimum and maximum values for the axis.
Since gx=1 the labels on the x-axis will be shown. The labels on the y-axis will not be shown since gy=0.

Where possible use subplot() function instead of gstdc()

See also: gaxis , subplot

---

gtadd(strV)

Add the value or text of strV to the previously drawn text.

See also: gprt , gcprt , glprt

---

gtapx

Returns the x-value of the cursor position.

Returns the x-value of the cursor position. Use the gwtap() to wait till the user taps within the graph.

See also: gtapy , gwtap

---

gtapy

Returns the y-value of the cursor position.

Returns the y-value of the cursor position. Use the gwtap() to wait till the user taps within the graph.

See also: gwtap , gtapx

---

gtitl(string)

Puts string above graph.

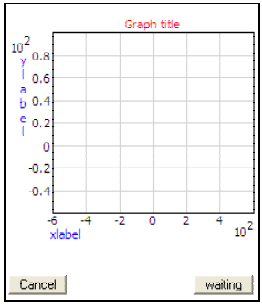


Figure 1

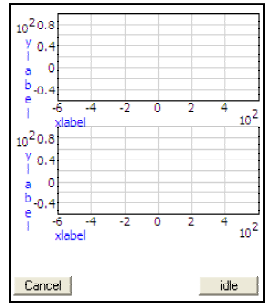


Figure 2

In version 1.60 the functions subplot(), xlable() and ylabel() where added. We try to use the limited graphical space on a PDA as good as possible however the limited space does create some problems also due to the different screen resolutions of the different PDA's.

This script is just a little example of how the labels and title are positioned automatically based on the size of the graph.

```
001 // gtitl(), xlabel() and ylabel()
002 x1=-500;x2=500;y1=-50;y2=100
003
```

```
004
005 // Little example script to show how
006 // the labels and title are automatically
007 // positioned.
008
009 function graph()
010 {
011   gaxis(x1,y1,x2,y2,1,1)
012   gselcol(1)
013   gtitl('Graph title')
014
015   gselcol(2)
016   xlabel('xlabel')
017   ylabel('ylabel')
018 }
019
```

```
020 subplot(1,1,1)
021 graph()
022
023 // Wait for user to press the continue
024 // button at the bottom of the screen.
025 gcont()
026
027 // Clear the screen
028 gclrs()
029
030 subplot(2,1,1)
031 graph()
032
033 subplot(2,1,2)
034 graph()
```

See also: xlabel , ylabel

---

gvlin(x)

Draw vertical line at x position.

---

gwtap

Wait till the user taps on the graphical screen to obtain the x and y value for the cursor position.

Wait till the user taps on the graphical screen to obtain the x and y value for the cursor position. Use the gtapx() and gtapy() functions to obtain the x and y values.

See also: gtapx , gtapx

---

gxlog

Use logarithmic scale for the x-axis.

Use logarithmic scale for the x-axis. Call this function before gaxis()

See also: gaxis , gylog

---

gylog

Use logarithmic scale for the y-axis.

Use logarithmic scale for the y-axis. Call this function before gaxis()

See also: gaxis , gxlog

---

subplot(r,c,pos)

Divides the graphical screen into r rows and c columns. The next graph will be drawn in area pos.

For example subplot(2,2,p) will divide the graphical screen into two rows and two columns. The next graph will be plotted at position p.

subplot(2,2,1) plot at top left corner.
subplot(2,2,2) plot at top right corner.
subplot(2,2,3) plot at bottom left corner.
subplot(2,2,4) plot at bottom right corner.

---

xlabel(string)

Puts string below the graph.

See gtitl() for more information.

See also: gtitl , ylabel

---

ylabel(string)

Puts string at the left of the graph.

See gtitl() for more information.

See also: gtitl , xlabel

Interactive functions

---

inpv(mess,v)

Show a dialog box with the message indicated by mess. The variable v will be shown that can be changed using the keypad in the dialog box.

---

iskey(str)

Checks if a key was pressed.

The function returns one if the key indicated by str was pressed.

Example:

```
001 // key, iskey example
002
003
004 // Text pmt in the button at the second row third column.
005 key(23,'pmt')
006
007 // Check if the key pmt was pressed.
008 if(iskey('pmt'))
009 {
010   // Execute when the pmt button was pressed.
011 }
```

See also: key

---

key(pos,str)

key(23,'pmt') will put the text pmt in the button at the second row third column.

The function returns one if the key indicated by str was pressed.

Example:

```
001 // key, iskey example
```

```
002
003
004 // Text pmt in the button at the second row third column.
005 key(23,'pmt')
006
007 // Check if the key pmt was pressed.
008 if(iskey('pmt'))
009 {
010   // Execute when the pmt button was pressed.
011 }
```

See also: iskey

---

mode1(strV)

Puts the text or values of strV on the main screen where by default the display format is shown.

See also: mode2 , result

---

mode2(strV)

Puts the text or values of strV on the main screen where by default the date is shown.

See also: mode1 , result

---

result(strV)

Puts the text or values of strV on the main screen at the result line.

After executing a script this function can be used to show information about the result.

```
len=123
result('Length='+len) // Will show the the text
Length=123 at the result line.
```

See also: mode1 , mode2

Logical functions

---

and(h,h)

Bitwise and.

---

not(x)

Returns 0 if x!=0 else 1 .

---

or(h,h)

Bitwise or

---

shl(h,b)

Bitwise shift left

---

shr(h,b)

Bitwise shift right

---

xor(h,h)

Bitwise exclusive or

Relational functions

!=

Not equal to

&&

Logical and operation.

<

Less than

<=

Less than or equal to

==

Equal to

>

Greater than

>=

Greater than or equal to

||

Logical or operation.

Special functions

beta(n,m)

Beta function

The beta function is defined by

$$beta(z, w) = \int_0^1 t^{(z-1)} (1-t)^{(w-1)}\, dt$$

See also: betai

betai(a,b,x)

Returns the incomplete beta function.

The betai function is defined by

$$betai(a,b,x) = \frac{1}{beta(a, b)} \int_0^x t^{(a-1)} (1-t)^{(b-1)}\, dt$$

See also: beta

cnd1(x)

Cumulative normal distribution function.

See also: cnd2

---

cnd3(a,b,rho)

Cumulative distribution function for a bivariate normal distribution.

See also: cnd1

---

der1(f,x,h)

Returns the first derivative of function f at x

Example:

```
001 // Derivaties
002 tp=3
003
004 // Change function below
005 function f(x){x^2}
006
007 // Notice the single quotes around
008 // the function name below
009 d1=der1('f',tp,10E-06)
010
011 // Calculate second derivative
012 d2=der2('f',tp,10E-06)
```

See also: der2

---

der2(f,x,h)

Returns the second derivative of function f at x.

See also: der1

---

erf(x)

Error function of x.

The error function is defined by

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

```
sqrt(pi) /
         0
```

---

fft(r1,r2)

Calculates the fourier transform of the rows starting at row r1 till row r2. Column one should contain the real values and the second column should contain the imaginary values. Make sure the second column contains zeros when using only real numbers.

Returns a finite Fourier transform. If *N* is the number of rows element of the return value is equal to

```
  N
-----
>  scget(i,1)*exp[-2*pi*j*(i-N/2-1)*(k-N/2-1)/N]
-----
 i = 1
```

See also: ifft

---

gamma(x)

Gamma function

The gamma function is defined by

$$gamma(x) = \int_0^{+inf} e^{-t}\, t^{(x-1)}\, dt$$

See also: lngam

---

gcd(x,y)

Greatest common divider.

gcd(45,63) returns 9

See also: mod

---

ifft(start,end)

Calculates the inverse fourier transform of the rows start till end. Column one should contain the real values and the second column should contain the imaginary values. Make sure the second column contains zeros when using only real numbers.

Returns the inverse Fourier transform. If *N* is the number of rows elements.

```
 N
-----
>     scget(i,1)*exp[+2*pi*j*(i-N/2-1)*(k-N/2-1)/N]
-----
i = 1
```

See also: fft

---

linint(x1,y1,x2,y2,x)

Linear interpolation. Calculates (y2-y1)/(x2-x1)(x-x1)+y1*

Linear Interpolation is a method that can be used for predicting. Very often something changes over a period of time: an object might change its position; a computer graphic image might change its shape; a population might increase. **Linear interpolation** allows you to predict an unknown value if you know any two particular values and assume that the rate of change is constant.

Linear interpolation assumes

1. that you know two particular values x1,y1 and x2,y2.
2. that the process is changing at a constant rate
3. that you desire to find an unknown data point. The y-value at x.

---

lngam(x)

Returns the natural logorithm of the gamma function.

The gamma function is defined by

$$\Gamma(z) = \int_0^{+\infty} e^{-t}\, t^{(z-1)}\, dt$$

See also: gamma

---

perc(a,b)

*Percentage change. Calculates (b-a)/a*100*

---

pval(x,p1,p2,p3,p4,p5)

*Returns poly values (p1*x+p2*x^2+p3*x^3+p4*x^4+p5*x^5).*

---

res(val,type)

Returns the closed resistor value of select type. Types 6,12,24,96,192

Returns the closed resistor value.
The EIA "E" series specify the preferred values for various tolerances.

Valid types are:
6 - E6 20% tolerance (seldom used)
12 - E12 10% tolerance
24 - E24 5% tolerance (and usually 2% tolerance)
96 - E96 1% tolerance
192 - E192 .5%, .25%, .1% and higher tolerances

res(43,24) will return the closed resistor value of with a 5% tolerance.

Click here for more resistor information

---

root(f,x)

Return the value of x at which the expression of function f(x) is equal to zero.

A little example script for the function f(x^3-10*x+2) that has multiple roots.

```
function f(x){x^3-10*x+2}

// find the root using guess value
x1=-2;x1=root('f',x1)
x2=0 ;x2=root('f',x2)
x3=3 ;x3=root('f',x3)
```

The root that will be found when there are multiple roots is depending on the guess value x.

---

sat(x,min,max)

Saturation function. Returns x when between limits otherwise returns the limit.

sat(10,5,15) returns 10
sat(20,5,15) returns 15
sat(2,5,15) return 5
See also: min , max

---

sinc(a)

*sin(pi*x)/(pi*x) returns 1 if x=0*

Statistics functions

---

fac(n)

Returns the factorial of n

fac(5) returns 1*2*3*4*5

---

ftest(c1,n1,n2,c2)

Returns the result of an F-test. An F-test returns the one-tailed probability that the variances in column c1 and column c2 are not significantly different.

Use this function to determine whether two samples have different variances. The arguments n1 and n2 indicate the number of data points in column c1 and c2.

See also: ttest

---

nCr(n,m)

Combination

For a lottery you have to pick 6 digits between 1 and 50. Your changes of winning is 1 in nCr(50,6).

See also: nPr

---

nPr(n,m)

Permutation

For a lottery you have to pick 5 digits between 1 and 35 but you must pick them in the correct order. Your changes of winning are nPr(35,5).

See also: nCr

---

rnd()

Generates a random number in the range [0, 1] with a uniform distribution and good statistical properties.

---

rndn()

Uses the Polar Method to return a random number with a normal distribution and a mean of zero.

---

scget(r,c)

Returns a complex value from the array. After a scadd(3,2,v) the function v=scget(3,2) will return the complex value. Column two is used for the real values and column three is used for the imaginary values.

---

schi(c1,c2)

Chi-squared function. c1 - column of expected values. c2 - column of observed values.

---

scnorm(x,mu,sig)

Returns the cumulative standard normal distribution. (m=mu, sig=stdev)

---

scorr(c)

Returns the correlation between the values in column 1 and the values in column r.

---

scput(r,c,v)

Store value v at row r and column c. When v is a complex value the real part will be stored in column c , and the imaginary part.

---

sdata(file)

Clear statistical variables and fill statistical array with data points of file. Make sure file is string by using single quotes.

Example:
rows=sdata('Data02');

The file Data02 will be read from the data category. The return value contains the number of rows read.

See example scripts in the statistical category.

See also: ssave

---

serre(c)

Returns the standard error of estimate.

The standard error of estimate is measure that indicates how far predicted data points are, on average, from the actual data points.

See also: serr

---

serrr(c)

Returns the standard error of regression.

---

sget(r,c)

Get value at row r and column c.

---

smax(c)

Maximum value in column c.

See also: smin

---

smean(c)

Returns the mean. (Sum / N) of column c.

---

smin(c)

Minimum value in column c.

See also: smax

---

snorm(x,mu,sig)

Returns the standard normal distribution. (m=mu, sig=stdev)

---

splot(c1,c2)

Plot values in column c1 versus values in column c2. See FFT example.

---

sput(r,c,v)

Store value v at row r and column c. When v is a complex use scput(r,c,v) instead!!

See also: scput

---

sqrc(c)

Returns the coefficients for the quadratic regression. A=sqrc(3) B=sqrc(2) B=sqrc(1) See QuadReg.prc user program for an example.

---

sqrv(x)

*Returns the value for the quadratic regression Y=A*X^2 + B*X + CSee QuadReg.prc user program for an example.*

---

sregc(c)

Returns the regression coefficient of column 1 and column c.

---

sregl(c)

Plots the regression line for column 1 and column c.

srplot(Col,r1,r2,Type)

Plot the range starting at row r1 to row r2 of column Col. Type specifies the type of plot 0-line 1-diamond points 2-cross points 3-plus points. This function will clear the screen use the maximum size to draw the plot and use autoaxis labeling.

See also: splot

---

ssave(file)

This function will save all data points of the statistical array in a file. The file will be stored in the data category.

Example:
sdata('Data02')
ssave('Data10')

If the file Data02 contains remarks they will not be saved in file Data10.

See also: sdata

---

ssum(c)

Sum of values in column c.

---

stclr()

Clear statistical variables.

---

stdev(c)

Returns the population standard deviation sqr(var()) of column c.

Stdev(c)

Returns the sample standard deviation sqr(Var()) of column c.

---

svar(c)

Returns the population variance (1/N(A(n)-mean)^2) of column c.*

---

sVar(c)

*Returns the sample variance (1/(N-1) * (A(n)-mean)^2) of column c.*

---

sxy(c)

*Returns A(1,n) * A(c,n) .*

---

syint(c)

Returns y-intersect.

---

ttest(c1,n1,c2,n2,tail,type)

Returns the probability associated with a Student's t-Test. Use ttest to determine whether two samples are likely to have come from the same two underlying populations that have the same mean.

The arguments n1 and n2 indicate the number of data points in column c1 and c2. Tail specifies the number of distribution tails. If tails = 1, ttest uses the one-tailed distribution. If tails = 2, ttest uses the two-tailed distribution. Type is the kind of t-Test to perform and should be set to two. See Student_ttest.pdb user program.

See also: ftest

Trigonometric functions

---

acos(a)

Arccosine

---

acosh(a)

Inverse hyperbolic cosine.

---

asin(a)

Arcsine

---

asinh(a)

Inverse hyperbolic sine.

---

atan(a)

Arctangent

---

atan2(x,y)

Retuns the four quadrant arctangent of the real parts of the elements of X and Y.

See also: atan

---

atanh(a)

Inverse hyperbolic tangent.

---

cos(a)

Cosine

---

cosh(a)

Hyperbolic cosine.

---

sin(a)

Sine

---

sinh(a)

Hyperbolic sine.

---

stdeg()

Set angular format to degrees.

---

strad()

Set angular format to radians.

---

tan(a)

Tangent

---

tanh(a)

Hyperbolic tangent.

Example scripts

Biorhytms

Biorhythms script in keyboard category

```
&Peter,&John,&Mark
7,8,9,/,VAR
4,5,6,*,EDIT
1,2,3,-,RUN
0,.,E,+,EXE
```

After loading this script the keyboard layout will change and display names in the keys at the first row. Press one of these keys and the biorhythms for that person will be shown. The main purpose of this script is to show the use of the date function and how to use scripts in the keyboard category.

Notice that when you load a script PDAcalc will check in the keyboard category for a file with the same name as the script.

```
// Biorhythms
xMin=-15;xMax=15
selectDate=1

bSel=0
if(iskey('Peter'))
{
  // Peter's date of birth
  birthDate=date(1,3,1964,0)
  bSel=1
}
if(iskey('John'))
{
  // John's date of birth
  birthDate=date(1,28,1974,0)
  bSel=1
}
if(iskey('Mark'))
{
  // Mark's date of birth
  birthDate=date(9,21,1970,0)
  bSel=1
}

error(bSel==0,'Please select name')

if(selectDate==1)
{
  curDate=date(0,0,0,'Select date')
}
else
{
  curDate=time()
}

// date() returns seconds
// Calculate number of days.
nDays=(curDate-birthDate)/(24*60*60)

gstdc(0,15,155,90)
gaxis(xMin,-1,xMax,1,1,1)

x=xMin
y1=sin(360*((x+nDays)/23))
y2=sin(360*((x+nDays)/28))
y3=sin(360*((x+nDays)/33))
gmove(x,y1)
gmove2(x,y2)
gmove3(x,y3)

while (x<=xMax)
{
  y1=sin(360*((x+nDays)/23))
  y2=sin(360*((x+nDays)/28))
  y3=sin(360*((x+nDays)/33))
  glin(x,y1)
  glin2(x,y2)
  glin3(x,y3)
  x=x+0.2
}

greset ()

y=105
gline(10,y,30,y)
gselcol(1)
gprt(35,y,'Physical')

y=115
gline2(10,y,30,y)
gselcol(2)
gprt(35,y,'Emotional')

y=125
gline3(10,y,30,y)
gselcol(3)
gprt(35,y,'Intellectual')

// Set for date format
fmt(8,6,1,0)
gselcol(15)
gcprt(80,135,curDate)

// Set to default display format
fmt(0,6,4,0)
```

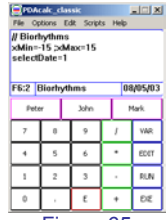


Figure 65

Figure 66

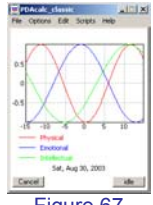


Figure 67

Graph demo

```
001 // Example graph script
002 min=0;max=4.2
003
004
005 step=(max-min)/100
006
007 // Indicate where to put the graph
008 // on the screen.
009 gstdc(70,10,159,65)
010
011 // Minimum and maximum values
012 // on the graph. The last two
013 // arguments indicate if the values
014 // should be shown.
015 gaxis(min,-1,max,1,1,1)
016
017 // Title on top of graph
018 // Put this function after gaxis()
019 gtitl('Sine function')
020
021 x=min
022
023 // Set initial point to start
024 // drawing from.
025 gmove(0,0)
026
027 // Continue executing the statements
028 // between brackets while condition
029 // is true.
030 while(x<=max)
031 {
032   y=sin(x/max*360)
033   glin(x,y)
034   x=x+step
035 }
036
037 // Well just copy and paste from above
038 // and change some values.
039 gstdc(0,80,75,130)
040 gaxis(-max,-max,max,max,1,1)
041 gtitl('Spiral')
042 x=min
043 gmove(0,0)
044 while (x<=max)
045 {
046   angle=x/max*720
047   radius=x
048   glin(pol(radius,angle),0)
049   x=x+step
050 }
051
```

```
052 gprt(3,15,'Most CplxCalPro')
053 glprt('scripts can be')
054 glprt('used with minor')
055 glprt('changes if any!')
056
057 gprt(80,90,'This spiral was')
058 glprt('generated with')
059 glprt('the function:')
060 glprt('pol(radius,angle)')
```

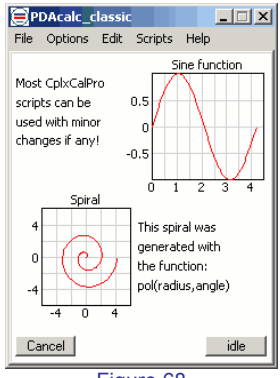


Figure 68

FFT example program

Let's take a look at the "FFT calculate" program. This program is included in the database of scripts and is located in the electronics category. Notice the function gselcol() to select the different text colors.

```
001 Phase=124
002 Xmin=0;Xmax=360
003 Ymin=-1;Ymax=1
004
005 stdeg()
006 fmt(0,6,6,0)
007
008 gtitl('Fourier calculation')
009 gstdc(0,15,159,100)
010 gaxis(Xmin,Ymin,Xmax,Ymax,1,1)
011 N=100
012 Step=(Xmax-Xmin)/N
013 x=Xmin
014
015 Real=0
016 Imag=0
017 while (x<=Xmax)
018 {
019   y3=sin(x+Phase)
020   y1=sin(x)
021   Imag=Imag+y1*y3
022   y2=cos(x)
023   Real=Real+y2*y3
024   if(x==Xmin)
025   {
026     gmove(x,y1*y3)
027     gmove2(x,y2*y3)
028     gmove3(x,y3)
029   }
030   else
031   {
032     glin(x,y1*y3)
033     glin2(x,y2*y3)
034     glin3(x,y3)
035   }
036   x=x+Step
037 }
038 greset ()
039 x=80
040 y=135
041 mul=20
042 Real=Real/(N/2)
043 Imag=Imag/(N/2)
044 //gfcir(x,y,mul)
045 yMul=y-Imag*mul
046 xMul=x+Real*mul
047 gline(x,y,x,yMul)
048 gline2(x,y,xMul,y)
049 gline3(x,y,xMul,yMul)
050 fmt(0,3,2,0)
051
052 // Select color of line 2
053 gselcol(2)
054 grprt(30,125,'Real=',Real)
055
056 // select color of line 1
057 gselcol(1)
058 grprt(30,135,'Imag=',Imag)
059
060 v=Real+0+1j*Imag
061
062 // Select color of line 3
063 gselcol(3)
064 grprt(140,125,'Angle=',arg(v))
065 grprt(140,135,'Mag=',abs(v))
```

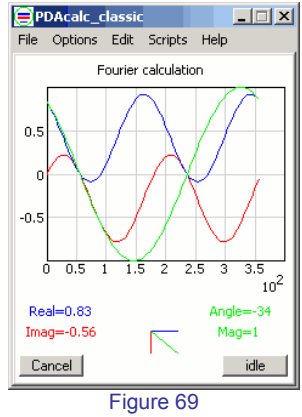


Figure 69

For more information about fourier transforms please visit our website.

FFT built-in functions

```
001 // Fourier transform
002
003
004 // Read data points
005 N=sdata('fft_data')
006
007 // plot data points
008 srplot(1,1,N,0)
009 gtitl('Raw data')
010
011 // Put message on screen
012 // since it takes a while on
013 // a palm device
014 gprt(50,152,'Wait for waiting=>')
015
016 // Calculate the fourier transform.
017 fft(1,N)
018
019 i=1
020 while(i<=N)
021 {
022   // Calculate magnitude and
023   // put in column three.
024   sadd(i,3,abs(scget(i,1)))
025   i=i+1
026 }
027
028 // Wait to continue.
029 gcont()
030 // Clear graphics screen.
031 gclrs()
032
033 // Plot magnitude
034 srplot(3,1,N/2-1,0)
035
036 gtitl('Magnitude')
037 gcprt(100,135,'Frequency')
```

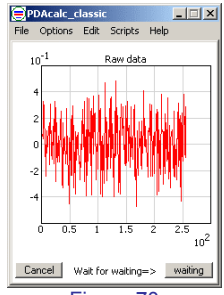


Figure 70

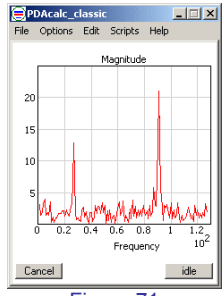


Figure 71

The sdata(datafile) returns the number of rows read from the data file. The function srplot(1,1,N,0) plots the raw data. Notice that there is not need to setup anything for the labeling of the plot since that is all done for you by the srplot() function.
After starting the program make sure you wait till the text in the stop button changes to continue to indicate the fft calculation is ready.

Quadratic regression example

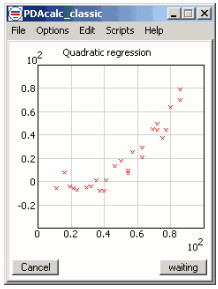


Figure 72

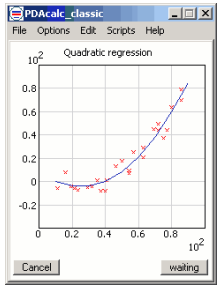


Figure 73

```
001 // Quadratic regression
002 x1=0;x2=100;sx=25;xStep=10
003 y1=-25;y2=100;sy=25
004
005 n=sdata('Data02')
006
007
008 gtitl('Quadratic regression')
009 gstdc(5,15,158,125)
010 gaxis(x1,y1,x2,y2,1,1)
011 x=sget(1,1)
012 y=sget(1,2)
013 gmove(x,y)
014
015 // Plot data points
016 i=2
017 whilei<=n)
018 {
019   x=sget(i,1)
020   y=sget(i,2)
021   gpnt(1,x,y,2)
022   i=i+1
023 }
024
025 // Wait for user to press
026 // continue.
027 gcont()
028
```

```
029 // Show the fitting
030 first=1
031 x=x1+xStep
032 while (x<x2)
033 {
034   y=sqrv(x) // same as  y=a*x^2+b*x+c
035   if(first==1)
036   {
037     gmove2(x,y)
038     first=0
039   }
040   else
041   {
042     glin2(x,y)
043   }
044   x=x+xStep
045 }
046
047 gcont()
048 gclrs()
049
050 gcprt(80,30,'Quadratic regression')
051
052 // Get the three values for the
053 // quadratic equation.
054 a=sqrc(3)
055 b=sqrc(2)
056 c=sqrc(1)
057
058 gprt(5,50,a+'*X^2 + '+b+'*X + '+c)
```

Chi-square test

The test requires that the data first be grouped. The actual number of observations in each group is compared to the expected number of observations and the test statistic is calculated as a function of this difference. The number of groups and how group membership is defined will affect the power of the test (i.e., how sensitive it is to detecting departures from the null hypothesis). Power will not only be affected by the number of groups and how they are defined, but by the sample size and shape of the null and underlying (true) distributions. Despite the lack of a clear "best method", some useful rules of thumb can be given.

When data are discrete, group membership is unambiguous. Tabulation or cross tabulation can be used to categorize the data. Continuous data present a more difficult challenge. One defines groups by segmenting the range of possible values into non-overlapping intervals. Group membership can then be defined by the endpoints of the intervals. In general, power is maximized by choosing endpoints such that group membership is equiprobable (i.e., the probabilities associated with an observation falling into a given group are divided as evenly as possible across the intervals).

One rule-of-thumb suggests using the value $2n^{2/5}$ as a good starting point for choosing the number of groups. Another well known rule-of-thumb requires every group to have at least 5 data points.

Example:

The results of an experiment are shown below in the row observed.

	Group 1	Group 2	Group 3	Group4	Sum
Probability model:	9	3	3	1	sum=16
Expected Ratio:	9/16	3/16	3/16	2/16	
Observed:	125	40	42	12	sum=219
Expected:	(9/16)* 219	(3/16)* 219	(3/16)* 219	(1/16)* 219	

$$\chi^2 = \sum \frac{(observed - \exp ected)^2}{\exp ected} = 0.283612379$$

// chi-data file
9,125
3,40
3,42
1,12

001 // Chi-square example
002
003
004 sdata('chi-data')
005 schi(1,2)

Figure 74

Opamp

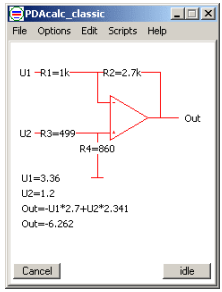


Figure 75

Little program to calculate the output of an electronic circuit with an operational amplifier, opamp.

```
001 R1=1000;R2=2700
002 R3=499;R4=860
003 U1=3.36;U2=1.2
004 gline(20,20,120,20)
005 fmt(3,3,1,0)
006 gprt(8,20,'U1')
007 gprt(25,20,'R1='+R1)
008 gprt(75,20,'R2='+R2)
009 gline(70,20,70,40)
010 gline(70,40,80,40)
011
012 // minus sign
013 gline(82,40,84,40)
014 gline(80,35,80,65)
015 gline(20,60,80,60)
016
017 // plus sign
018 gline(82,60,84,60)
019 gline(83,59,83,61)
020 gprt(25,60,'R3='+R3)
021 gprt(8,60,'U2')
022 gline(80,35,110,50)
023 gline(80,65,110,50)
024 gline(110,50,135,50)
025 gprt(140,50,'Out')
026 gline(120,20,120,50)
027 gline(70,60,70,90)
028 gcprt(70,70,'R4='+R4)
029 gline(65,90,75,90)
030
031 fmt(3,4,3,0)
032 gprt(10,90,'U1='+U1)
033 gprt(10,100,'U2='+U2)
034
035 // Start calculations
036 b=(R4/(R4+R3))*((R2+R1)/R1)
037 gprt(10,110,'Out=-U1*'+R2/R1)
038 gtadd('+U2*'+b)
039 gprt(10,120,'Out='+(-U1*R2/R1+U2*b))
```

Root function

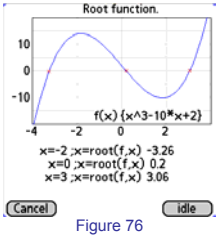


Figure 76

Notice the guess values, second argument in root function. The root function will start searching at this point for a solution.

```
001 x=4;h=10E-13
002 x1=-4;x2=4;y1=-20;y2=20
003
004
005 function f(x){x^3-10*x+2}
006
007 // Number of steps to plot the graph.
008 N=50
009
010 // Calculate step size
011 s=(x2-x1)/N
012
013 // Set area to draw graph.
014 gstdc(1,10,159,90)
015 gaxis(x1,y1,x2,y2,2,5)
016 gtitl('Root function.')
017 x=x1
018
019 // Set start position
020 gmove2(x,f(x))
021 while (x<=x2)
022 {
023   glin2(x,f(x))
024   x=x+s
025 }
026
027 gprt(70,84,'f(x) {x^3-10*x+2}')
028
029 // find the root using guess value
030 x=-2;x=root('f',x)
031
032 // Draw X at the position
033 // Change the two for diffent symbol
034 gpnt(1,x,f(x),2)
035 gcprt(80,110,'x=-2 ;x=root(f,x)  '+x)
036
037 x=0;x=root('f',x);gpnt(1,x,f(x),2)
038 gcprt(80,120,'x=0 ;x=root(f,x)  '+x)
039
040 x=3;x=root('f',x);gpnt(1,x,f(x),2)
041 gcprt(80,130,'x=3 ;x=root(f,x)  '+x)
```

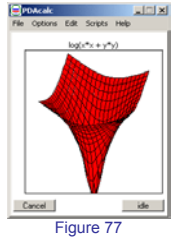


Figure 77

```
001 // log(x*x+y*y)
002
003
004 function f(x,y)
005 {
006   // Change user funtion below
007   log(x*x+y*y)
008 }
009
010 // The fplot3d() function checks if a
011 // graphical area was initialized already.
012 // If no graphical was initialized it
013 // will initialize the default area plus
014 // default rotation for you.
015 fplot3d('f',-1,-1,1,1)
016
017 // Put the gtitl() after fplot3d() because
018 // gtitl() should only be called after
019 // gaxis() which is called in fplot3d()
020 gtitl('log(x*x+y*y)')
```

To plot a different function change line 007 and the minimum and maximum values at line 15.

The green remarks are not needed to run the script. The script below will perform the same calculations and only shows the functions needed

```
001 // log(x*x+y*y)
002
003
004 function f(x,y)
005 {
006   log(x*x+y*y)
007 }
008 fplot3d('f',-1,-1,1,1)
009 gtitl('log(x*x+y*y)')
```

Programming PDAcalc classic

This is programming…You are able to push what the computer can do. You control every single small detail…You're twelve, thirteen, fourteen, whatever. Other kids are playing soccer. Your grandfather's computer is more interesting. His machine is its own world, where logic rules.

— Linus Torvalds, *Just For Fun*

Before you start writing your own scripts, you should look at: http://www.adacs.com. We encourage this for two reasons: first, our online library of scripts is teeming with examples of how other programmers solved problems specific to the strengths and limits of PDAcalc classic. Learning from others saves you time and effort. Second, a program might already exist to meet your needs, or could with some tweaking. Changing others' scripts, then running them to see what happens, is another way to learn programming (proper attribution, however, must always be respected).
Let's say a program exists in our library whose description suggests it seems close to what you're looking for. Download it. Inspect its comments and algorithm. Once you understand what the program does, and you believe you know how to modify it to meet your needs; edit it, change the program name; save the result, and run it. After everything works please do not forget about other users of PDAcalc classic. If your edits result in a program that either solves another set of problems than the original or significantly improves upon the original, you might want to consider uploading it to our website.

A Programming Primer

What is a script? For the purposes of this manual, let's define scripting as a set of instructions run on a PDAcalc module, some or all of which are executed, one at a time, during a run. The instructions that are executed transform input to output in a manner that could be done with pencil and paper. The set of instructions must stop running in a finite length of time.
So if we can carry out these instructions by hand, with pencil and paper, what use is a program? Simply put: speed and accuracy in repetition. Let's say your program solves a certain kind of problem that takes 20 steps on PDAcalc classic to perform. Once you've ensured that your program gives correct output, you can use your program as often as you need. Running it takes as few as one step to do (let's say putting in new values for the variables is another step); and your 20-step problem is solved much faster than you could do it by hand; and with every run you're confident that the results are not affected by missteps in the calculation.
PDAcalc classic's programming language is simple yet powerful. It lacks the GOTO statement and so requires scripts to be structured. Structured programming simplifies the order in which instructions are executed, which is called program flow. In structured programming there are only three kinds of flow control: sequence, selection, and repetition.

112 Programming PDAcalc classic

A program that has only sequential flow starts at the top, executes all of its instructions in the order they're written; and after it executes the last instruction, it stops.

A program that has one selection point in its flow runs sequentially until it reaches that point. The branch can have only one selection (two paths); but it can also have many. At the selection point the program evaluates a condition; which path of instructions the program follows depends on the outcome of that evaluation.

A program that has one repetitive loop in its flow runs sequentially until it reaches the loop's entry point. At the loop's exit point the program evaluates a condition; it continues running in the loop until the exit condition is satisfied.

These pictures are intended to show program flow only. the program lines themselves are nonsense. Real PDAcalc classic scripts that use the concepts illustrated here are in the programming examples section below.

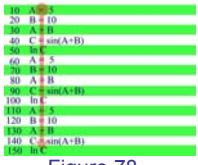


Figure 78

Figure 79

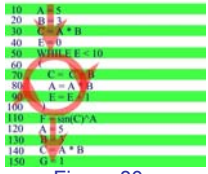


Figure 80

sequential flow: all steps are executed, one at a time, and in the order written.

selection flow: at the selection point, a condition is evaluated; the evaluation decides which steps will be evaluated, and which not.

repetition flow: program flow enters a loop, where it stays until an exit condition is satisfied.

note: sequential flow is the most basic. even in a selection path or repetitive loop program lines are executed one at a time, in the order they're written, until program flow reaches the end of the path or loop.

PDAcalc classic's Commands

Let's look briefly at PDAcalc classic's programming commands and operators before we tackle some sample scripts:

COMMAND	REMARK
if(cond)\n{\n}\n	If (condition) is true, execute program lines within the curly brackets. note that when the if command is used by itself, the condition decides only if additional program lines in your program (those within the curly brackets) will be executed.
else\n{\n}\n	optional to the if command. executes the commands between brackets when the condition for the if statement is false. note that the selection flow of an if-then-else statement lets the condition decide which of two sets of program lines (those within the curly brackets following the if(cond) or those within the curly brackets following the else will be executed.
while(cond)\n{\n}\n	while(condition) is true, execute the program lines between the curly brackets. program flow enters the while-loop, and continues flowing through it in a loop until the exit condition is met (that is, the while(condition) becomes false).
init()	returns one only the first time after executing a program.used mainly for initializing variables.
exit(n)	terminates program. n = 0 normal termination. n = 1 termination due to error.
Yea, yea, we know: use of the backslash-n (\n) to denote newline is UNIX convention.	

What are these conditions whose values decide program flow? On PDAcalc classic, they can be relational (meaning that PDAcalc classic tests one data value against another to decide action) or interactive (meaning that PDAcalc classic waits for input from the user).

Here are PDAcalc classic's relational operators:

OPERATOR	REMARK
!=	Not equal to
&&	Logical and operation.
||	Logical or operation.
<	Less than
<=	Less than or equal to
==	Equal to
>	Greater than
>=	Greater than or equal to

And here are some of the interactive functions whose values can decide program flow:

COMMAND	REMARK
iskey('StrV')	Returns 1 when button 'StrV' is pressed.
gcont()	Wait until continue button is pressed.

OK, you've slogged through enough talk. Let's look at some simple scripts to see how we can put this knowledge to use.

Programming Examples

More Graphics Fun! This time we want to draw concentric circles, evenly spaced, on the graphics screen, like in this screenshot:
I can think of several ways to do this. One is to go back to the Graphics Fun file we wrote in the chapter on graphics, and write the gcir(col,x,y,r) function 8 times, keeping x and y constant while increasing the value of r by 8 in each consecutive gcir(col,x,y,r), like this:

```
001 // concentric circles
002
003
004
005 greset()
006 gcir(1,80,80,8)
007 gcir(1,80,80,16)
008 gcir(1,80,80,24)
009 gcir(1,80,80,32)
010 gcir(1,80,80,40)
011 gcir(1,80,80,48)
012 gcir(1,80,80,56)
013 gcir(1,80,80,64)
```

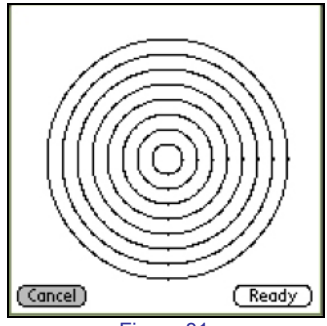


Figure 81

Remember, a PDAcalc classic scratchpad variables (that is, variables whose values can be changed after the script has successfully loaded and before a script run) on lines 1,2, and 3. The code, therefore, starts on line 4. If you wish to enter this script and run it start writing the code from line 4.
But I don't know why you'd bother. This code is an example of sequential flow, but it's not a very good one. Why? Because seven lines are essentially repeats of one line, gcir(1,80,80,8), varying only in their radius values.
Such repetition of code begs casting the gcir(1,x,y,r) function in repetition flow. Few (actually, none) would argue that it isn't more efficient to write a small script in which we write the gcir(1,x,y,r)

function once, but in such a way that it gets executed as many times as we want concentric circles.
For repetition flow on PDAcalc classic, we use:

```
while (cond)
{
}
```

Between the curly brackets {} we'll put the kernel of this script, gcir(col,x,y,r), which we already know is going to draw those eight concentric circles. Because x and y are constant, let's put their value in gcir(1,x,y,r), so that it becomes gcir(1,80,80,r). The condition to keep script flow in the loop, executing gcir(1,x,y,r) over and over, will be r <= 64. Now we have a while loop that looks like this:

```
006 while (r<=64)
007 {
008   gcir(1,80,80,r)
009   r=r+8
010 }
```

And we're missing two things. What is the value of r before we enter the loop? Right now it's not set. On the line above the while loop, we write r=8. And what's the other thing we're missing? Reading the script aloud might help us find out. Reading aloud, we might say something like, "We set r equal to eight. Now while r is less than or equal to 64, draw a circle with center at 80,80, and radius of…" Here is the second thing we're missing: a statement that changes the value of r each time we go through the loop. As the code is written now, script flow will never leave the while loop as PDAcalc classic endlessly draws circles of radius zero. A common scripting mistake is writing repetitive structures that either terminate abnormally or not at all. How to fix this? We write the statement that changes the value of r at the bottom of our while loop.
Here's a script that will meet our specs:

```
001 // concentric circles
002
003
004 greset ()
005 r=8
006 while (r<=64)
007 {
008   gcir(1,80,80,r)
009   r=r+8
010 }
```

Let's go ahead and clear the graphics screen each time we run it. Put greset () above r=0.
Reading the script aloud, we might now say something like, "We clear the graphics screen and set r equal to zero. Now while r is less than or equal to 64, draw a circle with center at 80,80, and radius of r. Increase r by 8 each time after a circle is drawn."
Let's look at each way we've decided to draw our concentric circles.

```
001 // concentric circles
002
003
004 greset()
005 r=8
006 whiler<=64)
007 {
008   gcir(1,80,80,r)
009   r=r+8
010 }
```

```
001 // concentric circles
002
003
004
005 greset()
006 gcir(1,80,80,8)
007 gcir(1,80,80,16)
008 gcir(1,80,80,24)
009 gcir(1,80,80,32)
010 gcir(1,80,80,40)
011 gcir(1,80,80,48)
012 gcir(1,80,80,56)
013 gcir(1,80,80,64)
```

This brings us to yet another advantage of programming: flexibility. Let's say we want next to draw concentric circles with radii 4n*r, 0 <= r <= 64, instead of what we have now, 8n*r, 0 <= r <= 64. Had we drawn our concentric circles by the script on the right, not only would we have to change our value of r on each line, we'd have to add another 8 lines. But because we are using the script on the left, this change to the script specs requires only that we change 8 to 4 in the statement r=r+8.

And because you love Graphics Fun so much, I know you'll let me flog away at it a bit more. Could the script on the left be written yet another way, and yield the same result? Sure! We could have an index variable, x, such that its value is tested as the conditional and its value is multiplied by a constant to yield r. It would look like the code on the right.

```
greset ()
r=0
while (r<=64)
{
  gcir(80,80,r)
  r=r+8
}
```

```
greset ()
x=0
while(x<=8)
{
  r=x*8
  gcir(80,80,r)
  x=x+1
}
```

Now we have three scripts, all yielding the same result. Scripts or parts of scripts that yield the same output given the same input are called functionally equivalent. So which of these is preferred? Well, of course the one that's all sequential flow is least preferred. Index values are used like x is in the code on the right, but when it is called or used more than once in the repetitive loop; here it only adds one more line of code, and that slows down code execution (by only a bit, to be sure). In general, the simpler the code, the better. The code on the left is preferred.

We know Graphics Fun runs and outputs what we want, but have we finished with it? A minimalist would say yes; but we're missing documentation. We use documentation in code to explain, even to ourselves, what the code is doing. It seems overmuch in such a small script, I admit; but we're using this script for learning purposes.

Documentation comes in two forms: self-documenting code and comments.
Self-documenting code is code whose parts the programmer gives names to, names that explain what those parts are or what they do. In our example, the name “Graphics Fun” is less descriptive than “DrawBullseye” so that’s what we’ll write on line one. Also, r as a variable name here is fine, since r stands for “radius” in math; but to be explicitly self-documenting, we’ll change the name. Comments follow double-slash (//) and are used to explain what the code is doing.

```
001 //initialize
002 radius=0
003
004 //draw concentric circles
005 //making each circle's radius
006 //8 pixels > than last
007 while(radius<=64)
008 {
009   gcir(1,80,80,radius)
010   radius=radius+8
011 }
```

As written, our script requires editing if we want to change any of its parameters; and once running, accepts no input from the user. PDAcalc classic gives the user greater flexibility for changing script variables, and allows user interaction during script run. The next two changes to our script show how.
Let’s start with script interaction: once the user runs the script, we want it to wait until the user taps the [Continue] button on the graphics screen before drawing any circles. To do that, we simply put the command gcont() after the initialization lines.
And finally, one last requirement: we want the user to be able to change by how much the radius grows (or, delta-r) with each circle as often as the user likes without having to edit the code. To meet the requirement, we must create another variable; let’s call it deltar (for delta-r, of course). Lets also change the color of circle so we add an other variable col for this purpose.

```
001 deltar=8
002
003
004 //initialize
005 greset ()
006 radius=0
007
008 //draw concentric circles
009 //making each circle's radius
010 //8 pixels > than last
011 col=0
012 while (radius<=64)
```

```
013 {
014   gcir(col,80,80,radius)
015   col=col+1
016   radius=radius+deltar
017 }
```

After this script successfully loads, the scratchpad on the textscreen shows deltar = 8. Now the user can change the value by how much the radius grows by changing the value of deltar in the scratchpad.
We wish to end this primer with an observation: all scripts have parameters. Within their parameters, their writers try to ensure that they work properly and do not give bad output. After you write a script on PDAcalc classic, you should test it for soundness. This means finding values for your script's variables that will cause the script to fail, or give bad output; and once you find them, either having your script catch the errors, or document what causes errors as your script's parameters.
A valid example for a PDAcalc classic script would be to determine if any variable causes a divisor to become zero while your script runs. Because division by zero is undefined, if it happens in your script, your script will terminate abnormally and PDAcalc classic will display:

```
001 step=40
002
003
004 while(step>-10)
005 {
006   test=100/step
007   step=step-5
008 }
```

After loading this script and pressing [RUN] the screen on the right appears. When it is not clear what the error was you can press [OK] and look at the error message at the result line of the main screen.
This line should show:
6:Division by zero:

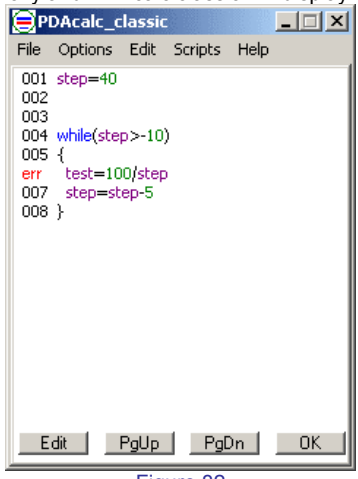


Figure 82

How to prevent this? If the value of the divisor variable is set by the user, you can document your script with a comment like this:
// if varFoobar = 0, script will terminate abnormally!
If the value of the divisor variable changes during the script run, you might be able to catch the error in such a way that the script still gives good output. Look at the following example:
if (divisor != 0)

```
{
   test = 12/divisor
}
else
{
   test = 0
}
```

In this example we catch the error by testing for the value of the variable (cleverly named divisor) before performing the division, test = 12/divisor. The else clause says what to do instead of letting the script terminate abnormally if divisor = 0. In this example, setting test = 0 meets script specs.

Matrix module

PDAcalc matrix is the most powerful programmable matrix calculator especially designed for the PDA. Enter numerous values in one matrix and perform calculations on all the values at the same time. Plot all the values with a simple command using a linear or logarithmic scale. Yes, even auto-scaling is supported. Let PDAcalc matrix do the work for you! Now that we're familiar with the main screen, let's do some basic calculations on PDAcalc matrix using the default preferences:

Advantage of a matrix calculator

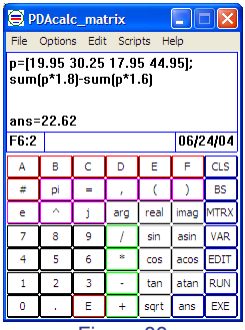


Figure 83

Lets just start with a simple example. Image you are selling products with prices of $19.95 $30.25 $17.95 and $44.95. You want to increase the mark-up from 1.6 to 1.8. What is the difference if you sell one of each?

Enter the script, well the two lines, as shown in Fig. 1 and press [exe] at the bottom. Normal calculators work with one value at a time. PDAcalc matrix works with as many values as there are in the matrix.
If the markup is not the same for each product you can create an other matrix like m=[1.6 1.7 1.6 1.8] and enter tot=p.*m. Notice the period in front of the *. This is needed when you like to multiply each element, value, individually. Without the period PDAcalc matrix will use the matrix multiply instead.

Basic calculations

The Problem Statement:	You press:	PDAcalc matrix Displays:	Remarks:
3 + 4	3 + 4 [EXE]	3+4 7	Pretty straightforward.
(3+4) * 2	[CLS]* 2 [EXE]	ans*2 14	Use the previous result for the current calculation.
2^((3+4)*2)	[CLS] 2 ^ [ans] [EXE]	2^ans 16,384	The ans key supplies the previous result in the equation.
(2^((3+4)*2))^(1/14)	[CLS] [ans] ^ (1 / 14) [EXE]	ans^(1/14) 2	Fractional exponents.
Same as above	[CLS] (2 ^ ((3 + 4)*2)) ^ (1/ 14) [EXE]	(2^((3+4)*2)) ^(1/14) 2	PDAcalc matrix follows algebraic order of precedence when evaluating expressions.
Same as above, but with a deliberate error	delete a) from the expression in the scratchpad, then press [EXE]	1:')' expected	PDAcalc matrix's interpreter does syntax error-catching.
(-1)^(1/2)	[CLS] [sqr] – 1) [EXE]	sqr(-1) 0 +1j	Imaginary numbers! Note that a prepended 'j' designates the imaginary part of a complex number PDAcalc matrix.*
Solve the linear system: 2x+ y -2z=10 3x+2y+2z=1 5x+4y+3z=4	[CLS][MTRX] [2 1 -2; 3 2 2; 5 4 3] \ [10; 1; 4] [EXE]	ans= [1;2;-3]	PDAcalc_matrix uses MATLAB typography for reading and writing matrices. A matrix begins and ends with brackets, and semicolons separate rows. The backslash (\) is used to set up system-of-linear-equations problems in the form Ax=b. The answer to this problem reads: x=1, y=2, z=-3.
Solve the linear system: 4x- 2y =5 -6x+3y =1	[CLS] [4 -2; -6 3] \ [5;1] [EXE]	1:Singular matrix:solve()	The system has no solution (that is, these lines do not intersect). A system with an infinite number of solutions (that is, the system of equations describe one line only) gives the same message.
Multiply the scalar (-2+1j) into the vector [2 4 6]	[CLS] (-2+1j)*[2 4 6] [EXE]	ans= [-4+2j -8+4j -12+6j]	Do arithmetic operations with vectors and matrices. Here we multiply a scalar having an imaginary part with a vector.
Multiply the column vector [-2+1j; 4; 5] and row vector [2j 4 6j]	[CLS] [-2+1j; 4; 5]*[2j 4 6j] [EXE]	ans= [-2-4j -8+4j -6-12j 0+8j 16 0+24j 0+10j 20 0+30j]	PDAcalc matrix lets us work with imaginary numbers in vectors and matrices.

let A=[2 3 5; 7 11 13; 17 19 23]	[CLS] A =[2 3 5; 7 11 13; 17 19 23] [EXE]	A= [2 3 5 7 11 13 17 19 23]	Variable assignments.
Get the (2,3) element of A	A(2,3) [EXE] (do not clear the matrix assignment for A. put the instructions A(2,3) on the second line.)	A= [2 3 5 7 11 13 17 19 23] ans= 13	Note: do not clear the matrix assignment for A. put the instructions A(2,3) on the second line.
Put 29 in the (2,3) position of A	A(2,3)=29 A [EXE] (do not clear the matrix assignment for A. modify the instructions on the second line to A(2,3)=29, and A on the 3d line)	A= [2 3 5 7 11 13 17 19 23] ans= [2 3 5 7 11 29 17 19 23]	Note: do not clear the matrix assignment for A. modify the instructions on the second line to A(2,3)=29, and A on the 3d line.
Find the determinant of A.	det(A) [EXE] (do not clear the matrix assignment for A. put the instructions det(A) on the second line.)	A= [2 3 5 7 11 13 17 19 23] ans= -78	Note: do not clear the matrix assignment for A. put the instructions det(A) on the second line.
Find the eigenvalues of A	eig(A) [EXE] (do not clear the matrix assignment for A. put the instructions eig(A) on the second line.)	A= [2 3 5 7 11 13 17 19 23] ans= [36.811728 - 1.917028 1.1053]	Note: do not clear the matrix assignment for A. put the instructions eig(A) on the second line. eig(A) returns a vector of the eigenvalues of A. [V, D] = eig(A) returns matrices of eigenvalues (D) and eigenvectors (V) of matrix A.

Screen shots

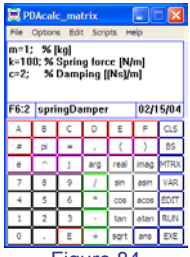


Figure 84

The main screen of the matrix module is shown on the left. Tap on the [MTRX] to go to the matrix screen as shown on the right. Notice how the results of the equations in scratchpad are shown below the keypad.

Figure 85

When an equation is terminated with a semi-colon the result will not be shown. This is why the contents of variable B is not shown.
Press the [RET] button to return to the main screen.

Lets start with a simple script. After reading The calculator section of this manual you know already how to create a new script. If you are more adventures, did not read the script, you can just select Scripts for the main menu and select new script. Enter the script below and select save as. Next select load from the Scripts menu to load the script you just saved. This will convert, optimize and store the script

into memory so it can be executed as fast as possible. Next press [RUN] and a screen as shown below will appear.

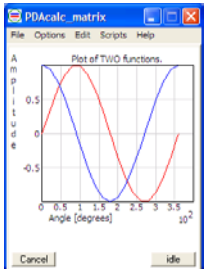


Figure 86

```
001 x=0:20:360;
002 r=x./180*pi;
003
```

```
004
005 plot(x,sin(r),'r',x,cos(r),'b')
006 title('Plot of TWO functions.')
007 xlabel('Angle [degrees]')
008 ylabel('Amplitude')
```

Note: The last three lines are only to print the labels and title. These are not needed for drawing the plot. To save space on the limited screen a multiplication factor is used for the x-axis. In this case 100.

Notice that the variables shown in the scratchpad, top three line on the main screen, are easy to change. Pressing [RUN] will execute the same script using the changed variables without loading the script again. This is a big advantage since loading a big script using a palm device can take a while.

Matrix multiplication

Let A and B denote two matrices that have this characteristic: The number of columns in A equals the number of rows in B. These matrices are conformable with respect to each other, and they can be combined by an operation known as the multiplication of matrices. Let C denote the matrix formed by multiplying A and B. Matrix C is the product of A and B, and the operation is expressed symbolically as C=AB.

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + \ldots + a_{in}b_{nj} \quad (1)$$

Well are you impressed already? When I see this kind of formula for the first time my first thought is yeh yeh just give me a practical example of how useful it is and then I might be impressed.

Let take a look at an example with two cities A and B.
Every year 40% of the people in city A move to city B.
Every year 30% of the people in city B move to city A.
After one year we have in city A: 0.6A + 0.3B (2)
After one year we have in city B: 0.4A + 0.7B (3)
After two years we have in city A:
0.6 (0.6 A + 0.3 B) + 0.3 (0.4 A + 0.7 B) = 0.48A + 0.39B (4)
After two years we have in city B:
0.4 (0.6 A + 0.3 B) + 0.7 (0.4 A + 0.7 B) = 0.52A + 0.61B (5)

Ok let me stop here and leave it up to the reader to keep on going for the next couple of years.

First year $$y1 = \begin{bmatrix} 0.6 & 0.3 \\ 0.4 & 0.7 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 0.6A & 0.3B \\ 0.4A & 0.7B \end{bmatrix} \quad (6)$$

Second year $$y2 = \begin{bmatrix} 0.6 & 0.3 \\ 0.4 & 0.7 \end{bmatrix} \left(\begin{bmatrix} 0.6 & 0.3 \\ 0.4 & 0.7 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} \right) \quad (7)$$

Ok time to enter an equation into PDAcalc matrix
First select the matrix screen and then type:
[0.6 0.3; 0.4 0.7] * [0.6 0.3; 0.4 0.7] and press [exe].

The screen on the right should appear.
Look back at the equations (4) and (5) Instead of [0.6 0.3; 0.4 0.7] * [0.6 0.3; 0.4 0.7] you could write the follow script:
n=2;
y=[0.6 0.3; 0.4 0.7];
p=y^n

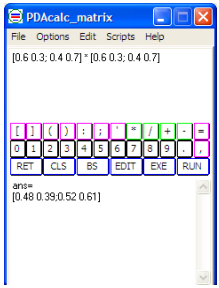


Figure 87

If you calculated the populations after 5 years then you can verify the result by changing
n =2 to n=5 and press [exe].
Apply equation (1) to the two matrices in equation (7) and that is what MtrxCal did to multiply the two matrices.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix} \quad (9)$$

User functions

When you like to execute the same set of commands, for which there is not a build-in function, several times within the same script you can create a user function.

```
001 % user function example
002 A=magic(4);
003
```

```
004  % Declare function after scratchpad!!!
005 function [y1,y2]=example(x)
006   y1=min(x);
007   y2=max(x);
008 return
009
010 text(20,10,'User functions in PDAcalc matrix')
011
012 y=50;
013 ret=example(A);
014 mtrx2scr(10,y,'example(',A,')=',ret)
015
016 [a b]=example([4 7 3]);
017 y=y+40;
018 text(10,y,'[a b]=example([4 7 3])')
019 y=y+10;
020 text(10,y,'a='+a)
021
022 y=y+10;
023 text(10,y,'b='+b)
```

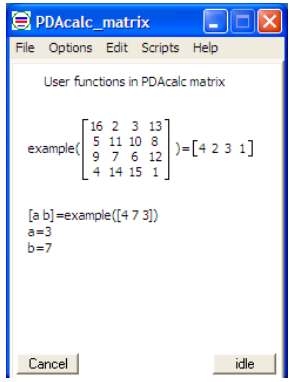


Figure 88

In the example script a user function is created at line 5. This function returns two values, y1 and y2. The function is called at line 13 and line 16.

The variables within a function are not stored on a stack that means that recursion, calling the same function within the function, is not allowed. This is mainly because of speed and memory considerations on a palm device. Also notice that the function is declared within the same script and needs to be declared before it is called.

User functions have to be declared after the scratchpad, first three lines.

Differential equations

Lets use a practical example to show to use the ode45() function to solve a differential equations.

Consider, for example a mass spring damper system like a car suspension and damping.

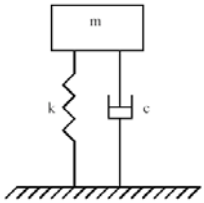


Figure 89

The formula to calculate the displacement of the mass m is:

$$m\frac{d^2x}{dt^2}+c\frac{dx}{dt}+kx=0 \qquad (1)$$

In which c is the damping and k is the spring force. This needs to be rewritten in two first order differential equation for the ode45('f',tspan,init) function.

$$\frac{dx}{dt} = v \qquad (2)$$

$$\frac{dv}{dt} = -\left[\frac{c}{m}v + \frac{k}{m}x\right] \qquad (3)$$

Next write the function in PDAcalc matrix:

```
005 function yp=system(t,y)
006   yp=[y(2);(-(a*y(2))-(b*y(1)))];
007 end
```

The first row in the return variable yp is the second derivate, formula (2), and the second row returns the first derivative, formula (3). The complete script is shown on the left. Notice the time span at line 12 and the initial values of y0 at line 13.

```
001 m=1;    % mass [kg]
002 k=100; % Spring force [N/m]
003 c=2;    % Damping [(Ns)/m]
```

```
004
005 function yp=system(t,y)
006   yp=[y(2);(-(a*y(2))-(b*y(1)))];
007 end
008
009 a=c/m;
010 b=k/m;
011
012 tspan=[0 4];
013 y0=[0.02;0];
014 [t,y]=ode45('system',tspan,y0);
015 plot(t,y(:,1));
016 xlabel('time')
017 ylabel('Displacement')
018 title('Displacement Vs Time')
```

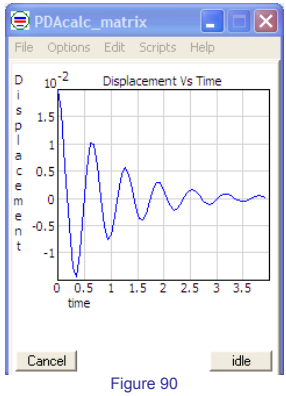


Figure 90

The function called by ode45() should have two arguments, time span and initial values, and one return value. The ode45() will return a row vector for higher order differential equations.
The method used in ode45() is a fixed Runge-Kutta method. At the time of writing this manual a fixed method of 60 equally spaced intervals is used. The number of intervals, iterations, can be changed using the iter() function. Using a high number of iterations can take a very long time to calculate. Please be careful when using the iter() function.

Lorentz contraction

This script shows how to use the iter() function, line 15, to increase the accuracy of the ode45() function. Please be careful using this script on a slow running palm OS device since executing this script will take a very long time. Changing line 15 to iter(60) will have the same result as not using the iter() function at all. It is the default.

```
001 % Do not use on a palm device!
002 to=0;
003 tfinal=30;

004
005
006 function F=lorentz(t,n)
007   x=n(1);
008   z=n(3);
009   y=n(2);
010   F=[5*(y-x);25*x-y-x*z;-3*z+x*y];
011 return
012
013 tspan=[to tfinal];
014 yo=[10;10;11];
015 iter(1000)
016 [T,Y]=ode45('lorentz',tspan,yo);
017
018 plot(Y(:,1),Y(:,2));
019 title(' x and y ')
```

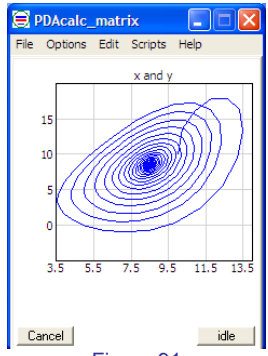


Figure 91

Integration

```
001 % Integration range
002 a=-2;
003 b=5;
```

```
004 % Declare function after scratchpad!
005 function y=test(x)
006   y=sin(x).*x;
007 return
008
009 % Integrate from a till b
010 area=quad('test',a,b);
011
012 % Create range of 60 points for plot().
013 x=a:(b-a)/60:b;
014
015 plot(x,test(x))
016 title('Integration example')
017 xlabel('Area='+area)
```

Calculates: $area = \int_a^b x * \sin(x) dx$

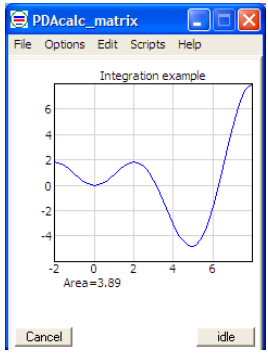


Figure 92

First create a user function containing the function you like to integrate. Then call the quad function as shown on line 10. The returned value is the integrated area under the curve.

Matrix built-in Functions

Arithmetic functions

.*

Array multiply

[2 3;4 5].*[6 7;8 9]=[12 21;32 45]
[2 3;4 5]*[6 7;8 9]=[36 41;64 73]

./

Array divide

[2 3;4 5]./[6 7;8 9]=[0.33 0.43;0.5 0.56]
[2 3;4 5]/[6 7;8 9]=[3 -2;2 -1]

See also: /

.^

Array power.

[4 5].^[3 2]=[64 25]

/

Matrix divide

Example: [2 3;4 5]./[6 7;8 9]=[0.33 0.43;0.5 0.56] [2 3;4 5]/[6 7;8 9]=[3 -2;2 -1]

---

kron(A,B)

Returns the Kronecker tensor product of A and B

For example, if X is 2 by 2, then KRON(A,B) is
```
[ A(1,1)*B  A(1,2)*B
  A(2,1)*B  A(2,2)*B]
```

\

*Backslash or matrix left division. If A is a square matrix, A\B is roughly the same as inv(A)*B*

Example:
[2 3;1 -1]\[14;-3]=[1;4]
Solves for 2*x+3*y=14 and x-y=-3

---

^

Matrix power.

A^M
All elements of M have to be an integer and can not be complex.

When raising A to the power of one value the value can complex and real.

See also: __

Bitwise functions

---

bitand(A,B)

Returns the bit-wise AND operation of A and B.

bitand([3 4 5],[6 3 9]) returns [2 0 1]

See also: ____ , _____

---

bitcmp(A,N)

negative integer.

bitcmp(3,4) returns 12.
Complement of 0011 as 2^4

---

bitget(A,N)

Returns the value of bit N in A.

See also: bitset

---

bitor(A,B)

Returns the bit-wise AND operation of A and B.

See also: bitand , bitxor

---

bitset(A,N)

Set the bit a position N in A.

bitset(5,4) = 13

See also: bitget

---

bitshift(A,N)

Shift the bits in A by N bits.

bitshift(8,-1) = 4

---

bitxor(A,B)

Returns the bit-wise EXOR operation of A and B.

See also: bit , and , bitor

Complex functions

---

abs(M)

Returns the absolute value of each element in the matrix. When an element is complex the magnitude will be returned.

---

angle(M)

Returns the phase angles of each element, in radians, of a matrix with complex elements.

See also: unwrap

---

conj(M)

Return the complex conjugate of each element in the matrix.

---

imag(M)

Return the complex values of each element as a real value.

See also: real

---

isreal

Returns true when all elements are real and false when at least one element is complex.

See also: imag , real

---

real(M)

Return only the real value of each element.

See also: ____ , isreal , _____ , ____

---

unwrap(V)

Unwraps radian angles.

Tries to change angle jumps greater then pi.

See also: angle

Date functions

---

now()

Returns the seconds past between the current time and Jan. 1 1904.

Exponential functions

---

exp(M)

Return the exponential of the elements of M, e to the power M

See also: log , log10 , pow2

---

log(M)

See also: exp , log2 , pow2

---

log10(m)

Return common, base 10, logarithm of M

See also: exp , log , log2

---

log2(M)

Return base 2 logarithm of M.

log2(64)=6 because 2^6=64

See also: pow2

---

pow2(M)

Base 2 power. Same as 2.^M

See also: log2

---

sqrt(M)

Returns the square root of each element in M

Flow control functions

---

else

execute statements when if condition is false.

```
001 % if test
002 a=4
003 b=8
004
005 if a>b
006   text(40,50,a+' > '+b)
007 else
008   text(40,50,a+' <= '+b)
009 end
```

See also: if

Used by if,while and for statement to indicate the end of a group of statements.

Used by if,while and for statement to indicate the end of a group of statements to be executed when the condition is true.

See also: if , while , for

---

error('msg')

Terminate execution of current script and show message.

Use this function to show an error message when conditions are not met to continue execution of current script.

---

exit(ret)

Exit script and return value

See also: error

---

for

Repeat statements

for i=StartValue:SetValue:LastValue
% Statements
end

Example:

```
001 % loop example
002
003
004 for y=10:20:110
005   text(15,y,'Y-position '+y)
006 end
007
008 y=10;
009 while y<=110
010   text(80,y,'Y-position '+y)
011   y=y+20;
012 end
```

The example will print the y values on the graphical screen by increments of 20 using a while loop and a for loop.

To decrement use a negative step value: for y=110:-20:10

See also: if , while

---

if condition

execute folowing statements when if condition is true

```
001 % if test
002 a=4
003 b=8
004
005 if a>b
006   text(40,50,a+' > '+b)
007 else
008   text(40,50,a+' <= '+b)
009 end
```

See also: else

---

while condition

Execute folowing statements while condition is true

Example:

```
001 % loop example
002
003
004 for y=10:20:110
005   text(15,y,'Y-position '+y)
006 end
007
008 y=10;
009 while y<=110
010   text(80,y,'Y-position '+y)
011   y=y+20;
012 end
```

The example will print the y values on the graphical screen by increments of 20 using a while loop and a for loop.

See also: for , if

Graphical functions

---

arc(col,x,y,r,a1,a2)

arc(color, x-position, y-postion, radius, angle1, angle 2)

For an example script using several draw functions take a look at:
smithcart

See also: line , circle

axis(...)

Set the current axis.

axis off % Does not show the axis nor the labels.
axis([-10 10 -20 20]) % Sets x,y axis limits for 2D plot.
axis([-10 10 -20 20 -30 30]) % Sets x,y,z axis limits for 3D plot.

See also: plot , bar , stem

bar(x,n)

Draws a bar graph using the vectors x and n.

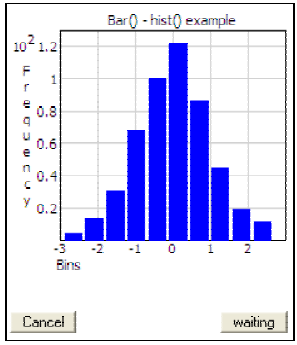


Figure 1

```
001 % bar() hist() example
002 N=500;% Random numbers
003 bins=10;% Number of bins
```

```
005 % Generate random numbers
006 y=randn(1,N);
007 save('random',y);
008
009 % Sort elements of y into bins
010 [n x]=hist(y,bins);
011 bar(x,n)
```

```
012 title('Bar() - hist() example')
013 xlabel('Bins')
014 ylabel('Frequency')
```

See also: hist , plot , stem

---

circle(col,x,y,r)

circle(color, x-postion, y-position, radius)

For an example script using several draw functions take a look at:
smithcart

See also: arc , line

---

clf

Clears the graphics screen.

---

greset

Resets the coordinate mapping. Normally not needed.

---

line(x,y)

Draws lines between the elements of the vectors in x and y.

Example: line([1 2],[3 4]) % Draws a line between the points (1,3) and (2,4)

For an example script using several draw functions take a look at:
smithcart

See also: cirle , arc

---

loglog(...)

loglog(...) is the same as plot(...), except logarithmic scales are used for both the X- and Y- axes.

See plot function

See also: plot , semilogx , semilogy

---

mesh(M)

Creates a 3-D mesh plot.

See also: plot

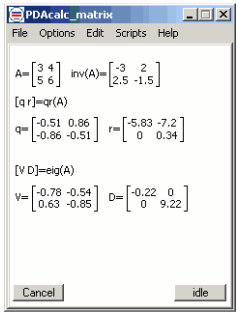
---

mtrx2scr(X,Y,S,A,...)

Show a matrix in graphical form. When there are to many elements in the array [...] will be shown instead.

mtrx2scr(10,40,'A=',A) will show the string 'A=' at position (10,40) and show the contents of matrix A behind it.

```
001 A=[3 4;5 6];
002
003
004
005 mtrx2scr(5,20,'A=',A,'
inv(A)=',inv(A))
006
007 [q r]=qr(A);
008 text(5,40,'[q r]=qr(A)')
009 mtrx2scr(5,60,'q=',q,' r=',r)
010
011 [V D]=eig(A);
012 text(5,90,'[V D]=eig(A)')
013 mtrx2scr(5,110,'V=',V,' D=',D)
```



See also: poly2scr

---

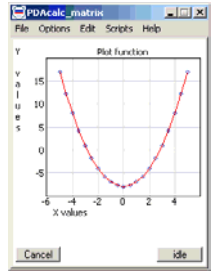
plot(...)

plot(X,Y) plots vector Y versus vector X.

plot(X) will plot the elements of X versus the index. When X is complex the imaginary part will be plot versus the real part of the complex elements.
plot(X,Y,S) plots Y versus X using S.
The first character in S is the color. The second is a point type and the third is the line type.

```
001 % Plot graph
002 X=-5:0.5:5
003 Y=X.^2-8
004 % The graph functions
need
005 % to be in the program
space.
006 plot(X,Y,'bd',X,Y,'r')
007 title('Plot function')
008 xlabel('X values')
009 ylabel('Y values')
```

First char.	Second char.	Third char.
r red	+ plus	- line
g green	x x-mark	
b blue	d diamond	
c cyan		
m magenta		



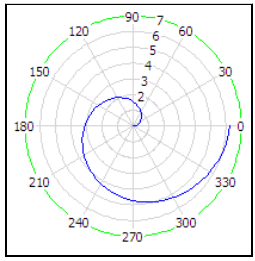
See also: semilogx , semilogy , loglog , title , xlabel , ylabel , bar , stem

---

polar(theta,rho)

Creates a polar plot using the angles theta, in radians, versus the radius rho.

Creates a polar plot using the angles theta, in radians, versus the radius rho.




```
001 % polar
002 theta=0:0.1:2*pi+0.1;
003 rho=theta;
004
005 polar(theta,rho)
```

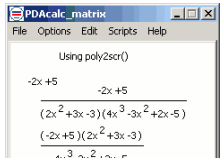
See also: plot , loglog , semilogx , semilogy

---

poly2scr(x,y,n,str,v,...)

poly2scr(10,50,2,'F(s)=','s',p1,p2,p3)
This function will draw the first n polynomials above the division line and p3 below the division line. The arguments str and v are optional.

You can use multiple arguments with this function. The first and second are always the x,y position. The y-position will be centered and the x-position is the position on the left. If there are only three



```
008 y=y+15;
009 poly2scr(25,y,1,p1,p2,p3)
010 y=y+35;
011 poly2scr(25,y,2,p1,p2,p3)
012 y=y+35;
013
poly2scr(15,y,2,'H(s)=','s',p1,p2,p3)
```

See also: mtrx2scr

---

semilogx(...)

Same as plot(...), except a logarithmic (base 10) scale is used for the X-axis.

See the plot function.

See also: plot , , loglog , semilogy

---

semilogy(...)

Same as plot(...), except a logarithmic (base 10) scale is used for the Y-axis.

See plot function

See also: plot , , semilogx , loglog

---

stem(...)

Plots a discrete sequence.

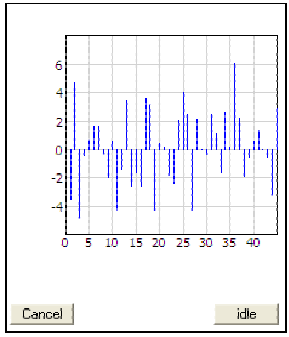


Figure 1

```
001 % Stem example
002 N=45;Ampl=3;
003
```

```
005 stem(y)
```

See also: plot , bar

---

subplot(r,c,pos)

Divides the graphical screen into r rows and c colums. The next graph will be drawn in area pos.

For example subplot(2,2,p) will divide the graphical screen into two rows and two columns. The next graph will be plotted at position p.

subplot(2,2,1) plot at top left corner.
subplot(2,2,2) plot at top right corner.
subplot(2,2,3) plot at bottom left corner.
subplot(2,2,4) plot at bottom right corner.

See also: plot , bar , stem , axis

---

text(X,Y,S)

Puts the string S at position X and Y on the graphical screen.

When S is scalar value the value is converted to a string.
When S is a matrix the matrix will be shown on one line if posible:

```
A=[3 4;5 6];


text(10,50,'A='+A)
```

will put A=[=[3 4;5 6] at position 10,50

See also: mtrx2scr , poly2scr

---

title(S)

Puts a title above the current graph.

See plot() for an example script

See also: plot , xlabel , ylabel

---

xlabel(S)

Puts the string S below the current graph.

See plot() for an example script

See also: plot , title , ylabel

---

ylabel(S)

Puts the string S at the left of the current graph.

See plot() for an example script

See also: plot , title , xlabel

Interactive functions

Shows a popup windows with the text S in which the user can enter a value. The default value is V.

It is normally quicker to assign the values in the scratchpad and pressing run. However you can also use the statement below to popup a window and wait till the user enters a value which will be returned by the inpv() functions.
len=inpv('Enter length',12)

Will return 1 if key S was pressed on the screen. Returns zero otherwize.

See also: key

---

key(pos,str)

key(23,'pmt') will put the text pmt in the button at the second row third column.

See also: iskey

---

pause

Stop executing script and wait till user taps the screen.

See also: clf

Logical functions

&

AND operator

Example: [0 0 4] & [0 5 6] = [0 0 1]

|

OR operator

Example: [0 0 4] | [0 5 6] = [0 1 1]

~

NOT operator

Example: ~[0 5 6] = [1 0 0]

Matrix functions

---

cart2pol(X,Y,Z)

Converts cartesian coordinates to polar coordinates.

[angle,radius,height] = cart2pol(x,y,z)
converts the cartesian coordinates x,y,z to to cylindrical coordinates

See also: sph2cart , cart2sph , pol2cart

---

cart2sph(x,y,z)

Converts cartesian coordinates to spherical coordinates.

[azimuth,elevation,radius] = cart2sph(x,y,z)
Converts x,y,z in cartesian coordinates to spherical coordinates.

See also: sph2cart , cart2pol , pol2cart

---

ceil(M)

Rounds the elements of M to the nearest higher integer.

See also: floor , round

---

chol(A)

*Cholesky factorization. L=chol(A) returns a lower triangular matrix L such that A = L*L'.*

This function computes the Cholesky factorization, i.e. it computes a lower triangular matrix L such that A = L*L'.

See also: qr , lu

---

cond(x)

Returns the ratio of the largest singular value of x to the smallest.

Returns the condition of a matrix, the ratio of the largest singular value of x to the smallest.

See also: norm

---

conv(A,B)

Convolves, polynomial multiplication of, vector A and B.

conv([2 3 4],[5 6 7]) returns: [10 27 52 45 28]
(2x^2 + 3x + 4)(5x^2 + 6x + 7) = 10x^4 + 27x^3 +

52x^2 + 45x + 28

See also: deconv

---

cross(A,B)

Returns the cross product of the vectors A and B.

See also: dot

---

deconv(A,B)

Deconvolution, polynomial division of, vector A and B

See also: conv

---

det(M)

Calculates the determinant of M

See also: inv

---

diag(V)

Creates a diagonal matrix.

diag([1 2 3]) returns [1 0 0;0 2 0;0 0 3]

See also: ones , zeros

---

dot(A,B)

must be vectors of the same length.

See also: cross

---

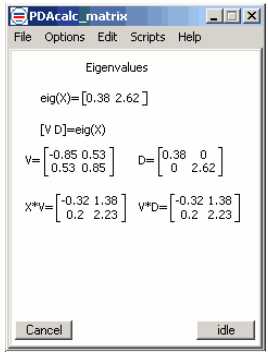
eig(M)

[V,D] = EIG(M) returns a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors.

[V D] = eig(M) will return a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors. M must have real elements but the returned matrices can be complex.

X*V = V*D.

eig(M) will only return the eigenvalues in a row matrix.




```
002 X=pascal(2)
003
004
005 % Calculate eigenvalues
006 text(50,10,'Eigenvalues')
007
008 mtrx2scr(20,30,'eig(X)=',eig(X))
009 [V D]=eig(X);
010 text(20,50,'[V D]=eig(X)')
011
012 mtrx2scr(10,70,'V=',V)
013 mtrx2scr(80,70,' D=',D)
014
015 mtrx2scr(10,100,'X*V=',X*V)
016 mtrx2scr(80,100,' V*D=',V*D)
```

See also: svd

---

eye(m,n)

eye(n) creates an identity matrix (n-by-n). eye(m,n) is an m-by-n matrix with 1's on the diagonal and zeros elsewhere.

eye(3) returns [1 0 0;0 1 0;0 0 1]

See also: diag , zeros , ones , eye

---

filter(B,A,X)

filters the data in vector X with the filter described by vectors A and B using "Direct Form II Transposed"

See also: conv , deconv

---

floor(M)

Rounds the elements of M to the nearest lower integer.

See also: ceil , round

---

freqspace(N)

Returns the implied frequency range for equally spaced. [0:1/N:1]

See also: logspace

Returns the inverse of the matrix M

See also: det

---

kron(A,B)

Returns the Kronecker tensor product of A and B.

For example, if X is 2 by 2, then KRON(X,Y) is

```
[ A(1,1)*B  A(1,2)*B
  A(2,1)*B  A(2,2)*B]
```

---

length(A)

Returns the length of vector A

See also: size

---

linspace(first,last)

Returns a vector of 100 elements equally spaces. linspace(first,last,n) returns n elements equally spaces.

See also: logspace , freqspace

---

logspace(first,last,n)

Returns a row vector of n logarithmically equally spaced points between decades 10^first and 10^last

logspace(first,last) returns 50 equally spaces points.

See also: linspace

---

lu(M)

*[L,U] = LU(X) stores an upper triangular matrix in U and a lower triangular matrix in L, so that X = L*U. X must be square.*

See also: qr

---

magic(M)

Return a matrix with equal row, column, and diagonal sums

See also: eye , diag , ones , zeros , pascal

---

mod(X,Y)

Return the modulus. The input X and Y must be real arrays of the same size

---

norm(A)

Return the norm of a vector or matrix

If A is a vector returns sqrt(sum(abs(A).^2)).
If A is a matrix returns the largest singular value of A, max(svd(A)).

See also: svd

---

ode45('func',[x1 x2],init_vals)

Integrates a system of differential equations.

The function called by ode45() should have two arguments, time span and initial values, and one return value. The ode45() will return a row vector for higher order differential equations.

The method used in ode45() is a fixed Runge-Kutta method. Currently a fixed method of 60 equally spaced intervals is used. The number of intervals, iterations, can be changed using the iter() function. Using a high number of iterations can take a very long time to calculate. Please be careful when using the iter() function.

See also: quad

---

ones(m,n)

Creates a ones matrix.

ones(2,2) returns [1 1;1 1]

See also: zeros , diag , pascal , magic

Returns a pascal matrix

See also: ones , zeros , diag , magic

---

pol2cart(angle,radius,height)

Converts from cylindrical coordinates to cartesian coordinates.

[x,y,z]=pol2cart(angle,radius,height)
Converts the cylindrical coordinates (angle, radius, height) to cartesian coordinates x,y,z.

See also: sph2cart , cart2sph , cart2pol

---

poly(A)

Returns a polynomial with roots of A.

poly([2 3]) returns [1 -5 6]
(x-2)*(x-3)=x^2-5x+6

See also: roots , polyder , polyval

---

polyder(A)

Returns the derivative of the polynomial whose coefficients are the elements of vector A.

See also: poly , roots , polyval

---

polyval(P,A)

Evaluate the polynomial, P, at all values of A

Example:
`polyval([p3 p2 p1],[x1 x2]) returns value of [p3*x1^2+p2*x1+p1 p3*x2^2+p2*x2+p1]`

See also: poly , roots , polyder

---

qr(M)

*QR decomposition. [Q R]=qr(M) returns an upper triangular matrix R and a unitary matrix Q so that M = Q*R*

Factors M into Q and R. Q is the upper triangular and R the lower triangular.

See also: lu , chol

---

quad('func',a,b)

Returns the result of Simpson's rule to approximate the integral of 'func' between a and b

Returns the result of Simpson's rule to approximate the integral of 'func' between a and b. A default of 60 slices is used which can be changed using the iter() function.

```
001 % Integration
002 a=-
003 b=5;
```

```
005 function y=test(x)
006 y=sin(x).*x;
007 return
008
009 % Integrate from a till b
010 area=quad('test',a,b);
011
012 % create range of 60 points
013 % to plot function
014 x=a:(b-a)/60:b;
015 plot(x,test(x))
016 title('Integration example')
017 xlabel('Area='+area)
```

See also: ode45 , iter

---

rank(A)

Returns polynomial roots.

Example:
p=conv([1 2],[1 3]); r=roots(p)

First the convolution function is used to multiply two polynomials.

The returned matrix p=[1 5 6]; (x+2)*(x+3)=x^2+5*x+6
roots(p) returns [-3 -2]

See also: conv , deconv , eig

---

round(M)

Rounds the elements of M to the nearest integer.

See also: floor , ceil

---

rref(M)

Returns a reduced row echelon form of A.

See also: eig , svd

---

size(M)

[r c]=size(M) returns the number of rows in r and the number of colums in c.

See also: length

---

sph2cart(azimuth,elevation,radius)

[x,y,z]= sph2cart(azimuth,elevation,radius)

Converts azimuth,elevation,radius in spherical coordinates to cartesian coordinates.

See also: cart2sph , cart2pol , pol2cart

---

svd(M)

*[U S V] = svd(M) returns a diagonal matrix S, unitary matrices U and V so that X = U*S*V'*

See also: eig

---

trace(A)

Returns the sum of the diagonal elements of matrix A.

---

zeros(m,n)

creates a zero matrix.

zeros(3) returns [0 0 0;0 0 0;0 0 0]

See also: ones , diag , zeros , magic , pascal

PDAcalc functions

---

fmt(t,w,p,tr)

Set display format
t: 0-float, 1-sci, 2-eng, 3-sym, 4-hex, 5-bin, 6-oct, 7-pol, 8-date, 9-sexagesimal
w: width of number (0-15)
p: precision of number (0-15)
tr: trailing zeros. (0 or 1)

Engineering

The native data format.

When a number cannot be displayed using width and precision settings, it is displayed in scientific format. An exponent will be used to show small numbers instead of leading zeros. With a precision setting of 7 the number 0.0001232456 will be shown as 1.23456E-04

When a number cannot be displayed using width and precision settings, it is displayed in engineering format. Enter 5.11e8 for example in the scratchpad and press [exe]. 511E6 will displayed. The exponent, in this case 6, will always be a multiple of three. The symbol format will show an SI postfix instead of E6.

Symbol

When a number cannot be displayed using the width and precision settings, it is displayed in symbol format. This is especially important when numbers are rendered on the graph screen in order to make sure all numbers are printed using the same space.

Name	SI Postfix	Power of 10
femto	f	-15
pico	p	-12

nano	n	
micro	u	
milli	m	-3
kilo		3
	M	
giga	G	9
tera	T	12

Hexadecimal Positive integers rendered in base 16 format

Binary Positive integers rendered in base 2 format.

Octal Positive integers rendered in base 8 format.

Polar Complex values converted to magnitude and angle.

Date Positive values are converted to dates.

Sexagesimal H.M.S format.

---

iter(i)

Change the number of iterations for calculating a numerical approximation.

For example the function ode45() uses a default of 60 equally spaced values to calculate a system of differential equations. The number of iterations can be increased using this function when called before the ode45() function is called. Be careful when using this function since a high number of iterations can take a very long time to calculate.

See also: ode45

---

mode1(strV)

Puts the text or values of strV on the main screen where by default the display format is shown

See also: mode2

---

mode2(strV)

Puts the text or values of strV on the main screen where by default the date is shown.

See also: mode1

Relational functions

<

Less than

<=

Less then or equal to

==

Equal to

>

Greater then

>=

Greater then or equal to

~=

Not equal to

Special functions

---

bartlett(N)

Returns the N-point Bartlett window.

See also: boxcar , blackman , hamming , hanning , kaiser , triang

---

besseli(order,n)

Returns modified bessel function of the 1st kind

See also: besselj

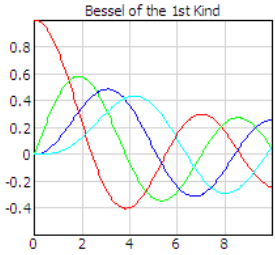
---

besselj(order,A)

Bessel function of the 1st kind

Bessel functions are solutions to the differential equation:
x2y" + xy' + (x2 - v2) Y = 0

```
001 % Bessel of the 1st Kind
002 x=0:0.05:10;
003
004 y0=besselj(0,x);
005 y1=besselj(1,x);
006 plot(x,y0,'r',x,y1,'g')
007 y2=besselj(2,x);
008 y3=besselj(3,x);
009 plot(x,y2,'b',x,y3,'c')
010 title('Bessel of the 1st Kind')
```



See also: besselj

---

blackman(N)

Returns the N-point blackman window.

See also: bartlett , boxcar , hamming , hanning , kaiser , triang

---

boxcar(N)

Returns the N-point boxcar window.

See also: bartlett , blackman , hamming , hanning , kaiser , triang

---

erf(A)

Error function of each element in A

The error function is defined by

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$$

---

fft(V,N)

Returns the discrete Fourier transform.

When using two argments the vector V will be padded with zeros if V has less than N points and truncated if it has more.

If N is the number of element the return value is equal to:

$$\sum_{i=1}^{N} V(i)*\exp[-2*pi*j*(i-N/2-1)*(k-N/2-1)/N]$$

See also: ifft , bartlett , blackman , boxcar , hamming , hanning , kaiser , triang

---

freqs(...)

[h,w,]=freqs(b,a)
[h,w,]=freqs(b,a,n)

h - complex frequency response
w - frequency points
a - numerator coefficients
b - denominator coefficients
n - number of frequancies used to calculate the response

Without any return arguments this function will plot the magnitude and phase response.

See also: impz , freqz

---

freqz(...)

Calculates the frequency response of a digital filter.

[h,w]=freqz(b,a)
[h,w]=freqz(b,a,n)

[h,f]=freqz(b,a,n,fs)

h - complex frequency response
w - frequency points
f - frequencies. a - numerator coefficients
b - denominator coefficients
n - number of samples
fs - sample frequency

Without any return arguments this function will plot magnitude and phase response.

See also: impz , freqs

---

gamma(A)

The gamma function is defined by

$$gamma(x) = \int_0^{+inf} e^{-t}\, t^{(x-1)}\, dt$$

See also: gammaln

---

gammaln(A)

Logarithm of gamma function.

See also: gamma

Returns the N-point hamming window.

See also: bartlett , blackman , boxcar , hanning , kaiser , triang

---

hanning(N)

Returns the N-point hanning window.

See also: bartlett , blackman , boxcar , hamming , kaiser , triang

---

ifft(V)

Returns the inverse discrete Fourier transform.

Returns the inverse Fourier transform. If N is the number of elements.

$$\frac{1}{N}\sum_{i=1}^{N} V(i)*\exp[+2*pi*j*i-N/2-1)*(k-N/2-1)/N]$$

See also: fft , bartlett , blackman , boxcar , hamming , hanning , kaiser , triang

---

impz(...)

Calculates the impulse response of a digital filter.

[h,t]=impz(b,a)
[h,t]=impz(b,a,ns)
[h,t]=impz(b,a,ns,fs)

Returns the impulse response in vector h and the sample times in vector t.

a - numerator coefficients
b - denominator coefficients
n - number of samples
fs - sample frequency

Without any return arguments the function will plot the impulse response.

See also: freqz , freqs

---

kaiser(N)

Returns the N-point kaiser window.

See also: bartlett , blackman , boxcar , hamming , hanning , kaiser , triang

---

triang(N)

Returns the N-point triangular window.

See also: bartlett , boxcar , blackman , hamming , hanning , kaiser

---

Statistics functions

---

cumsum(A)

Cumulative sum of elements.

cumsum([2 3 4])=[2 5 9]

See also: diff , prod

Returns the differences between elements

diff([2 5 9])=[3 4]

See also: prod , cumsum

---

hist(y,bins)

[n,x]=hist(y) Bins the elements of y in 10 equally spaced bins

[n x]=hist(y,bins) Bins the elements of y in equally spaced bins. The second argument determents the number of bins.

See also: bar

---

load(file)

M=load('file') Loads the datapoints of a file into matrix M.

M=load('file') Loads the datapoints of a file into matrix M.

See also: save

Returns the maximum of each column

max([3 5;6 8])=[6 8]

See also: min

---

mean(A)

Returns the mean of each column

mean([3 5;6 8])=[4.5 6.5]

See also: min , max , mean

Returns the minimum of each column

max([3 5;6 8])=[3 5]

See also: max , mean

---

polyfit(x,y,N)

Returns a vector of coeff. in decreasing order of degree N to fit data points x,y

Uses the least-square fit to find the coefficients of degree N to fit data points x,y

```
001 % polyfit example
002 x=[3 5 8 12];
003 y=[4 7 2 6];
004
005 p=polyfit(x,y,3);
006 xi=min(x):0.1:max(x);
007 yi=polyval(p,xi);
008 plot(x,y,'b+',xi,yi,'r')
```

See also: polyval , roots , poly

---

prod(A)

Returns the product of all elements

prod([3 4 5]) = 3*4*5 = 60

prod(1:5) returns factorial 5

See also: diff , cumsum

---

rand(m,n)

Returns an n-by-m matrix of random numbers between 0 and 1

See also: randn

---

randn(n,m)

Returns an n-by-m matrix of normally distributed random numbers.

See also: rand

---

save(file,M)

Saves the elements of matrix M into a file.

Saves the elements of matrix M into a file. The file will be stored in the data category.

---

sum(A)

Returns the sum of each column of matrix A

See also: min , max , cumsum

Trigonometric functions

---

acos(A)

Inverse cosine function of each element in A

---

acosh(A)

---

asin(A)

Inverse sine function of each element in A

---

asinh(A)

Inverse hyperbolic sine function of each element in A

atan(A)

Inverse tangent function of each element in A

atan2(y,x)

Retuns the four quadrant arctangent of the real parts of the elements of X and Y.

See also: atan

atanh(A)

Inverse hyperbolic tangent function of each element in A

cos(A)

Cosine function of each element in A

cosh(A)

sin(A)

Sine function of each element in A

Hyperbolic sine function of each element in A

Tangent function of each element in A

tanh(A)

Inverse tangent function of each element in A

Example scripts

Net Present Value

The net present value of a cash flow Cf is returned by the formula:

$$NPV(Cf,i)=\sum_t \frac{Cf_t}{(1+i)^t}$$

Suppose we are given the opportunity to make an investment of $4000 which will provide a return of $1000 next year and $2000 for each following two years. What is the internal rate of return on the investment if the discount rate (the cost of borrowing $4000) is 7%. How do we calculate this using PDAcalc matrix?

```
001 Cf=[-4000 1000 2000 2000];
```
Create an array of cash flow.

```
002 i=0.07;
```
Assign 7% to a variable

```
003 t=0:length(Cf)-1;
```
Create a range from 0 to 3, length(Cf) will return four so we subtract one.

```
004 Denom=(1+i).^t;
```
Create an array with values for the denominator.

```
005 sum(Cf./Denom)
```
Here we just divide each element in array Cf by each element in the array Denom and sum the result.

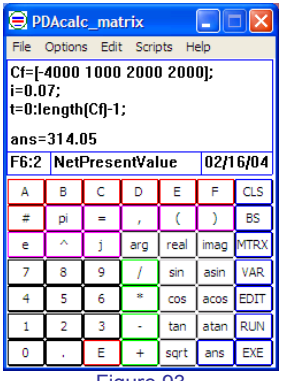


Figure 93

Notice the period before the ^ and the /. Without the period before the / MtrxCal will perform a matrix divide and that is not what we want in this case.

Now you can change the values in the Cf array and the percent to calculate the net present value for different investments. The text of this script can be copied, without making any changes, to matlab and will give the same answer.

Wave Period

Some of you might be familiar with fourier transforms but if you are not familiar with it don't worry. This example shows how a square wave can be represented by sinusoidal waveforms.

$$f(t) = \frac{4\sum_{h=1}^{H} b_{h-1} * \cos(h * w * t)}{\pi} \qquad H = 5 \qquad B = 1, 0, -\frac{1}{3}, 0, \frac{1}{4}$$

This can be implemented in several different ways but the main purpose of the script is to show how easy and powerful MtrxCal is. Notice how the arrays are multiplied and added in the example. The same can be done with an even smaller script in MtrxCal but would be more difficult to understand. Also notice how long it takes to calculate and plot the graph.

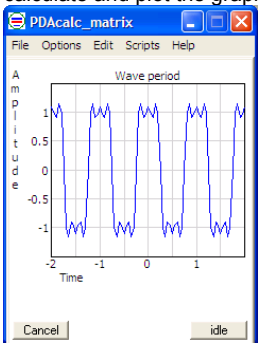


Figure 94

```
001 t=-2:0.05:2;
002 w=2*pi;
003
```

```
004
005 y1=cos(w.*t);
006 y3=-cos(3.*w.*t)/3;
007 y5=cos(5.*w.*t)/5;
008 y=4*(y1+y3+y5)/pi;
009 plot(t,y)
010 title('Wave period')
011 ylabel('Amplitude')
012 xlabel('Time')
```

Notice how the arrays are multiplied and added in the example above. When an array only has one element you don't have to use the .* operator to multiply.

Solve quadratic

Enter the x,y coordinates of three points and this script will calculate the a,b and c for the quadratic equation:

$$f(x) = ax^2 + bx + c$$

The script will solve three linear equations for three unknowns. You can also try to fit the data point with different equations by changing the matrix A.

```
001 x1=-5;y1=45;
002 x2=1;y2=10;
003 x3=4;
```

```
004 y3=60;
005 A=[x1^2 x1 1;x2^2 x2 1;x3^2 x3 1;];
006 B=[y1 y2 y3]';
007 C=A\B;
008 a=C(1)
009 b=C(2)
010 c=C(3)
011 x=x1:(x3-x1)/30:x3;
012 y=a*x.*x+b*x+c;
013 plot(x,y)
014 plot(x1,y1,'r+')
015 plot(x2,y2,'r+')
016 plot(x3,y3,'r+')
017 title('Solve quadratic')
018 ylabel('Y values')
019 xlabel('X-values')
```

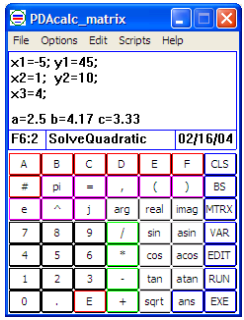


Figure 95

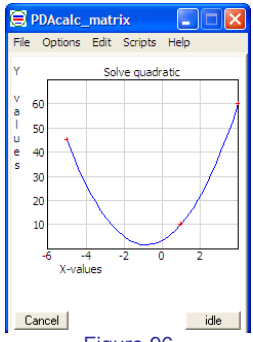


Figure 96

RC network

The frequency response is calculated using the Laplace transform

$$H(s) = \frac{\frac{1}{sC}}{R + \frac{1}{sC}} = \frac{1}{1 + sRC}$$

Replacing s by jw and taking the absolute value of H(s) give the frequency response.

```
001 R=1000;
002 C=10E-07;
003 Fc=1/(2*pi*R*C)
```

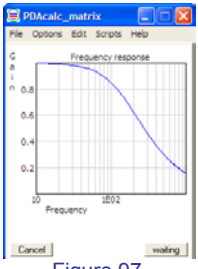


Figure 97

```
004
005 f=[10:10:1000];
006 w=0+1j*2*pi*f;
007 H=1./(1+w.*R*C);
008 semilogx(f,abs(H))
009 title('Frequency response')
010 xlabel('Frequency')
011 ylabel('Gain')
012 pause
013 clf
014 semilogx(f,angle(H))
015 title('Phase response')
016 xlabel('Frequency')
017 ylabel('Angle')
```

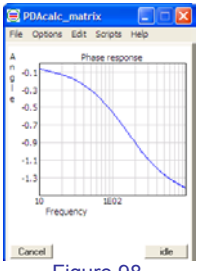


Figure 98

Replacing abs(H) with angle(H) gives the phase angle in radians. Replace abs(H) with 180*angle(H)/pi to get the angle in degrees. The cutoff frequency is also printed in line two and is 159.15 Hertz. The angle at the cutoff is –45 as shown in the Phase response.

Closed contour

Let's take the function $H(z)=\frac{z-1}{(z+1)(z^2+z+1)}$ and create a closed contour with a radius of 1.1

Now lets see if all the poles are within the contour.

```
001 R=1.1;
002 Theta=0:0.03:2*pi;
003
```

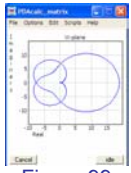


Figure 99

```
004
005 z=R*exp(0+1j*Theta);
006 H=(z-1)./((z+1).*(z.^2+z+1));
007
008 plot(H)
009 title('W-plane')
010 xlabel('Real')
011 ylabel('Imaginary')
```

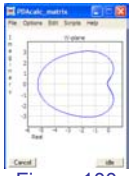


Figure 100

The radius Figure 99 is 1.1 and the radius in Figure 100 is 0.6
Even if the script is not totally clear to you it does show how easy it is to impress people with just a little script. We like to encourage people to write these scripts on any other calculator for the palm platform.

Hanging pendulum problem

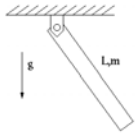


Figure 101

The equation for a hanging pendulum, using F=ma, is:

$mL\ddot{\theta} = -mg\sin(\theta)$ where theta is the angle of the pendulum from the vertical, m is its mass, L its length, and g the acceleration due to gravity.

By introducing the angular velocity $\omega = \frac{d\theta}{dt}$ we can write this as a system of two first order ordinary differential equations,

$\dot{\theta} = \omega$ and $L\dot{\omega} = -g\sin(\theta)$

These are used in the user function which is called by the ode45() functions as shown in the script.

```
001 m=1;%  mass of the bar
002 g=9.81;%  gravity
003 l=1;%  length of the bar
```

```
004
005 function der=pendulum(t,x)
006   der1=x(2);
007   der2=-(3*g)/(2*l)*sin(x(1));
008   der=[der1; der2];
009 end
010
011 % initial & final times
012 tspan=[0 5];
013
014 % initial conditions
015 theta=10E-04;% angle
016 vel=0;% angular_velocity
017 [t,x]=ode45('pendulum',tspan,[theta;vel]);
018
019 Theta=x(:,1);
020 vel=x(:,2);
021 plot(t,Theta,'b');
022 title('Angular Displacement vs. Time');
023 ylabel('rad');
024 xlabel('time (s)');
025 pause
026 clf
027 plot(t,vel,'r');
028 title('Angular Velocity vs. Time');
029 ylabel('rad/s');
030 xlabel('time (s)');
```

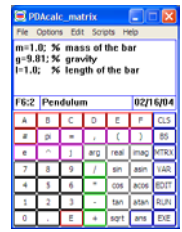


Figure 102

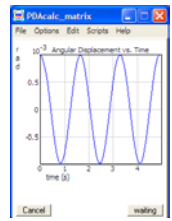


Figure 103

After loading the script the screen as shown in Figure 102will appear. Pressing run will show Figure 103. Press the [waiting] button and the angular velocity vs. time will be shown.

Appendix
Technical specifications

These NON-RPN calculator modules have the following technical specifications:

	Units	Classic	Matrix
Fully configurable keyboard	X	X	X
Color coded debug screen	X	X	X
Syntax checking on scripts	X	X	X
Browse our web site for free user scripts	X	X	X
Load and run user scripts directly from our web site	X	X	X
IEEE-754 64-bit Double Precision, a floating point format ranging from -2.23E-308 to 1.79E+308.	X	X	X
Display formats: Float, Scientific, Engineering, Symbol, Hexadecimal, Binary, Octal.	X	X	X
Display formats: Polar, Date and Sexagesimal.		X	X
Angular units: Radians and Degrees		X	
Complex numbers and functions		X	X
Linear regression		X	
Financial functions			X
Statistical functions		X	X
Root finding functions		X	X
FFT (fourier transforms)		X	
3D graphics		X	X
Auto-scaling of the graphs		X	X
3D functions, draw lines in 3D space, rotate objects in 3D space etc		X	X
Calculus functions, integral, derivatives, etc.		X	X

Data Formats

Float The native data format.

Scientific When a number cannot be displayed using width and precision settings, it is displayed in scientific format. An exponent will be used to show small numbers instead of leading zeros. With a precision setting of 7 the number 0.0001232456 will be shown as 1.23456E-04

Engineering When a number cannot be displayed using width and precision settings, it is displayed in engineering format. Enter 5.11e8 for example in the scratchpad and press [exe]. 511E6 will displayed. The exponent, in this case 6, will always be a multiple of three. The symbol format will show an SI postfix instead of E6.

Symbol When a number cannot be displayed using the width and precision settings, it is displayed in symbol format. This is especially important when numbers are rendered on the graph screen in order to make sure all numbers are printed using the same space.

Name	SI Postfix	Power of 10
femto	f	-15
pico	p	-12
nano	n	-9
micro	u	-6
milli	m	-3
kilo	K,k	3
mega	M	6
giga	G	9
tera	T	12

Hexadecimal Positive integers rendered in base 16 format

Binary Positive integers rendered in base 2 format.

Octal Positive integers rendered in base 8 format.

Polar Complex values converted to magnitude and angle.

Date Positive values are converted to dates.

Sexagesimal Mixed decimal fractions rendered in H.M.S format.

Display format

Considerations on Width, Precision, Accuracy, and Round-Off.

PDAcalc classic uses the IEEE-754 specification for 64-bit Double Precision floating-point numbers. As you know or should appreciate, floating-point numbers are not ideal numbers; rather, they are representations of ideal numbers. Somewhere to the right of the decimal point, floating-point numbers become inexact. The 64-bit Double Precision specification was chosen for PDAcalc classic to ensure that that inexactitude would not show up in the results of most calculations that require the precision of engineering applications (say, no more than 12 significant digits). However, PDAcalc classic was designed to be a power user's calculator. It puts the power of explicitly specifying the width and precision of numbers in the user's hands; this power can expose the inexactitude of these numbers to those users, as well as affect the accuracy of results. Furthermore, not understanding width and precision when changing their values can lead to answers that are simply wrong and don't make sense.

Most users should have no practical need to push PDAcalc classic to the limits of its accuracy simply because the accuracy of numerical results are determined by how many significant digits there are in the input. In general, the number of significant digits in the output that is meaningful is equal to the least number of significant digits in the input.

Using Excel

To make sure that calculations are performed correctly in Excel ☺, you can easily transfer data files between Excel and PDAcalc. The default directory in which the data files are stored is depending on the module.
PDAcalc classic data files are stored in the directory:
C:\Program Files\ADACS\PDAcalc\PDAcalc_classic\Data

For PDAcalc matrix in:
C:\Program Files\ADACS\PDAcalc\PDAcalc_matrix\Data

You can download a data file from this directory using the comma delimiter format into excel. To plot the data file use the XY (scatter plot) type.

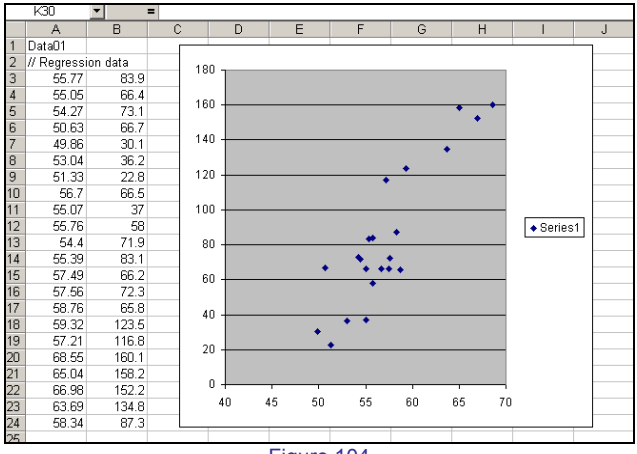


Figure 104

For compatibility reasons with the palm platform, the first line in a data file must be the name of the file. You can also write your data files from excel to the data directory when you use the same data format. To transfer your data file to the palm, just press the hotsync button.

Constants

Constant	Name	Value	Dimensions
pi		3.1415926535897932	none
e		2.7182818284590452	none
c	speed of light in vacuum	2.99792458E8	$m\ s^{-1}$
G	Newtonian constant of gravitation	6.67259E-11	$m^3\ kg^{-1}\ s^{-2}$
g	standard gravitational acceleration	9.80665	$m\ s^{-2}$
me	electron mass	9.1093897E-31	kg
mp	proton mass	1.6726231E-27	kg
mn	neutron mass	1.6749286E-27	kg
u	atomic mass unit (unified)	1.6605402E-27	kg
q	electron charge	1.60217733E-19	10^{-19} C
h	Planck constant	6.6260755E-34	J s
k	boltzmann constant	1.380658E-23	$J\ K^{-1}$
u0	magnetic permeability	1.2566370614E-6	$H\ m^{-1}$
e0	dielectric permittivity	8.854187817E-12	$F\ m^{-1}$
re	classical electron radius	2.81794092E-15	m
al	fine structure constant	7.29735308E-3	none
a0	Bohr radius	5.29177249E-11	m
R	Rydberg constant	1.097373153E7	m^{-1}
Fq	Fluxoid quantum	2.06783461E-15	Wb
ub	Bohr magneton	9.2740154E-24	$J\ T^{-1}$
ue	Electron magnetic moment	9.2847701E-24	$J\ T^{-1}$
uN	Nuclear magneton	5.0507866E-27	$J\ T^{-1}$
uP	Proton magnetic moment	1.41060761E-26	$J\ T^{-1}$
un	Neutron magnetic moment	9.6623707E-27	$J\ T^{-1}$
Lc	Compton wavelength (electron)	2.42631058E-12	m
Lcp	Compton wavelength (proton)	1.32141002E-15	m
sig	Stefan-Boltzmann constant	5.67051E-8	$W\ m^{-2}\ K^{-4}$
Na	Avogadro's constant	6.0221367E23	mol^{-1}
Vm	Ideal gas volume at STP	2.24141E-2	$m\text{-}3\ mol^{-1}$
R	Universal gas constant	8.31451	$J\ mol^{-1}\ K^{-1}$
F	Faraday constant	9.6485309E4	$C\ mol^{-1}$
RH	Quantum Hall resistance	2.58128056E4	Ohm

Curve Sketching

This appendix is intended only as a reference in curve sketching. The points to consider are not fully illustrated, as doing so falls outside the scope of a calculator user manual. However, just as we did in the main body of this manual, we wish to point out that good, instructive websites exist to allow you to learn or review the skill. Just type "curve sketching" into the textbox of your favorite Internet search engine, and browse the results. There's bound to be at least one website that meets your tastes and needs.

Know an equation by its curve. The curves of functions have general characteristics. Knowing those characteristics for the functions you're working with saves time when sketching them.

Define the domain: values of x that let the denominator in your equation = 0 are excluded.

Define the range: this means solving for x

Find all x-intercepts: set y = 0 in your equation and solve

Find all y-intercept(s): set x = 0 in your equation and solve

Find vertical asymptote: these are the values of x that let the denominator in your equation = 0

Find horizontal asymptote: let /x/ tend to infinity. if y approaches zero, horizontal asymptote @ y=0. if y approaches a nonzero number b, horizontal asymptote @ y=b.

Find concavity: take f'(x) of f(x). for any given x, the larger the value of /f'(x)/, the steeper the slope of f(x). positive f'(x) means upward slope. negative f'(x) means downward slope.

Find minmax: take f'(x) of f(x). set y=0 for f'(x) and solve.

Useful Web Links

URL	what you'll find there
www.adacs.com	us!
physics.nist.gov/cuu	The NIST Reference on Constants, Units, and Uncertainty, including in-depth information on the metric system
mathworld.wolfram.com	Eric Weisstein's World of Mathematics. Possibly the best mathematics reference work on the World Wide Web
www.ieee.org/	The IEEE, for electrical engineers and those of you curious about the standards that hardware, software (including PDAcalc classic!), and firmware adheres to.
www.acm.org/	ACM Association for Computing Machinery, the world's first educational and scientific computing society.

Afterword

This manual is intended to serve as a tutorial and reference to PDAcalc modules. Effort has been made to make the manual readable while ensuring conciseness, accuracy, and a thoroughness over the basics of calculator use to allow a user to start quickly applying PDAcalc modules to problem-solving. In other words, this manual is intended to serve you. If you find errors in the manual, or a passage difficult to understand, or what you consider to be a glaring omission, please let us know. We will consider all constructive criticism.