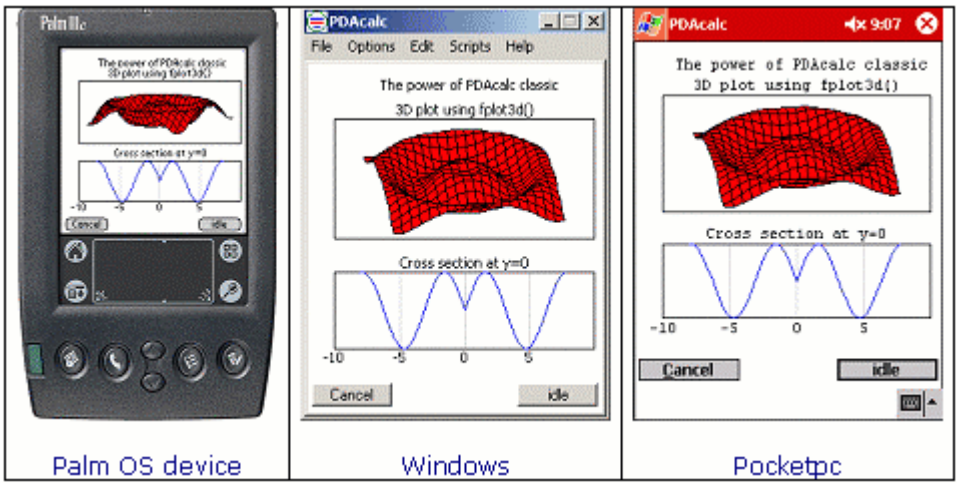
PDAcalc classic

The calculator for numerical analyses on the
Palm®, PocketPC and Windows platforms



User manual

Version 1.30

ADACS LLC

Advanced Digital & Analog Consulting Service

Email: info@adacs.com
Phone: 803 322 – 8312
Fax: 803 547 - 4667
Web site: http://www.adacs.com

Table of contents

CHAPTER 1 7

INTRODUCTION 7
A GALLERY OF SCRIPTS AND GRAPHICS 8
DIFFERENCES BETWEEN PLATFORMS 9
THE MAIN SCREEN 9
BASIC CALCULATIONS 11
MODIFYING KEY ASSIGNMENTS 12
THE MENU ITEMS 13
Options 13
Preferences 13
Display formats 14
Variables 15
Default Keyboard 15
Select module 15
Register 16
Register online 16
Edit 17
Scripts 18
Load scripts 18
Edit script 19
Debug scripts 19
Worksheets 19
Plot function 20
Solve Equation 21
Help 22
Functions 22
Site licenses 22
Legal agreement 22
About 22
Synchronize scripts with a Palm device 22
Register online 22
Visit our web site 22
Functions online 23
Browse scripts 23
User information 23
Upload/Update script 23
PUTTING IT ALL TOGETHER 23

CHAPTER 2 24

BUILT-IN FUNCTIONS ON PDACALC CLASSIC 24
COMPLEX: 24
BASIC: 24
CALCULUS: 25
FINANCIAL: 26
LOGICAL: 26
BASE CONVERSION: 26
PROBABILITY & STATISTICS 27
PDAcalc classic statistical and probability functions 28
USER-DEFINED FUNCTIONS 30

CHAPTER 3 31

GRAPHICS 31
DEFAULT COLORS 32
GRAPHICS EXAMPLES 32

CHAPTER 4 ..... **35**

PROGRAMMING PDACALC CLASSIC 35
A PROGRAMMING PRIMER 35

CHAPTER 5 ..... **42**

3D FUNCTIONS 42
3D GRAPHING 43

CHAPTER 6 ..... **44**

USING EXCEL 44

APPENDIX A **45**

FUNCTIONS, OPERATORS, AND COMMANDS 45
Basic functions *45*
+ 45
abs(x) 45
cbrt(x) 45
ceil(x) 46
exp(x) 46
floor(x) 46
fmt(t,w,p,tr) 47
frac(n) 48
ln(x) 48
log(x) 48
max(a,b) 48
min(a,b) 48
mod(x,y) 48
round(x) 49
sqr(x) 49
Color functions *49*
gselcol(idx) 50
gsetcol(idx,r,g,b) 50
Complex functions *50*
arg(cplx) 51
conj(cplx) 51
Im(cplx) 51
pol(r,a) 51
Re(cplx) 51
Conversion functions *51*
cel(T) 52
deg(a) 52
dms(hr,min,sec) 52
fah(T) 52
met(yr,ft,in,p) 52
rad(a) 52
Date functions *52*
date(mm,dd,yyyy,text) 53
days(sec) 53
time() 53
weeks(sec) 53
Financial functions *53*
fv(rate,nper,pmt,pv,type) 54
inter(nper,pmt,pv,fv,t) 54
nper(rate,pmt,pv,fv,type) 55
pmt(rate,nper,pv,fv,type) 55
pv(rate,nper,pmt,fv,type) 56
Flow_control functions *56*

else ... 57
error(condition,text) ... 57
exit(exitVal) ... 57
if(condition) ... 58
while(condition) ... 59
Graphical functions ... *59*
fplot3d(f,x1,y1,x2,y2) ... 60
garc(x,y,r,a1,a2) ... 60
gaxis(x1,y1,x2,y2,gx,gy) ... 61
gaxis3d(col,x,y,z,t) ... 61
gcir(x,y,r) ... 62
gclrs() ... 62
gcont() ... 62
gcprt(x,y,strV) ... 63
gfcir(x,y,r) ... 63
ghlin(y) ... 63
ginit3d(x1,y1,x2,y2) ... 64
glin(x,y) ... 64
glin2(x,y) ... 65
glin3(x,y) ... 65
glin4(x,y) ... 65
glin5(x,y) ... 65
gline(x1,y1,x2,y2) ... 65
gline2(x1,y1,x2,y2) ... 65
gline4(x1,y1,x2,y2) ... 65
gline5(x1,y1,x2,y2) ... 65
gmove(x,y) ... 66
gmove2(x,y) ... 66
gmove3(x,y) ... 66
gmove3d(idx,x,y,z) ... 66
gmove4(x,y) ... 66
gmove5(x,y) ... 66
gpnt(x,y,t) ... 66
gprbar(x,Min,Max,Y) ... 66
gprt(x,y,strV) ... 66
grect(x1,y1,x2,y2) ... 66
greset() ... 67
grotate3d(rotLR,rotUD) ... 67
grprt(x,y,v1,v2) ... 67
gstdc(x1,y1,x2,y2) ... 67
gtadd(strV) ... 67
gvlin(x) ... 67
Interactive functions ... *67*
inpv(mess,v) ... 68
iskey(str) ... 68
key(pos,str) ... 68
mode1(strV) ... 68
mode2(strV) ... 69
result(strV) ... 69
Logical functions ... *69*
and(h,h) ... 69
not(x) ... 69
or(h,h) ... 69
shl(h,b) ... 69
shr(h,b) ... 69
xor(h,h) ... *70*
Relational functions ... *70*

!= 70
&& 70
< 70
<= 70
== 70
> 70
>= 70
|| 71
Special functions 71
beta(n,m) 71
betai(a,b,x) 71
erf(x) 71
fft(r1,r2) 72
gamma(x) 72
gcd(x,y) 72
ifft(start,end) 73
linint(x1,y1,x2,y2,x) 73
lngam(x) 73
perc(a,b) 73
pval(x,p1,p2,p3,p4,p5) 74
root(f,x) 74
sat(x,min,max) 74
sinc(a) 74
Statistics functions 74
ftest(c1,n1,n2,c2) 74
nCr(n,m) 75
nPr(n,m) 75
rnd() 75
rndn() 75
scget(r,c) 75
schi(c1,c2) 75
scnorm(x,mu,sig) 75
scorr(c) 75
scput(r,c,v) 75
serre(c) 76
serrr(c) 76
sget(r,c) 76
smax(c) 76
smean(c) 76
smin(c) 76
snorm(x,mu,sig) 76
splot(c1,c2) 76
sput(r,c,v) 76
sqrc(c) 76
sqrv(x) 77
sregc(c) 77
sregl(c) 77
srplot(Col,r1,r2,Type) 77
ssum(c) 77
stclr() 77
stdev(c) 77
Stdev(c) 77
svar(c) 77
sVar(c) 77
sxy(c) 78
syint(c) 78
Trigonometric functions 78

acos(a) ... 78
acosh(a) ... 78
asin(a) ... 78
asinh(a) ... 78
atan(a) ... 78
atanh(a) ... 78
cos(a) ... 78
cosh(a) ... 79
sin(a) ... 79
sinh(a) ... 79
stdeg() ... 79
strad() ... 79
tan(a) ... 79
tanh(a) ... 79

APPENDIX B ... 80

SPECIFICATIONS ... 80

APPENDIX C ... 81

DATA FORMATS ... 81

APPENDIX D ... 82

DISPLAY FORMAT ... 82

APPENDIX E ... 83

CONSTANTS ... 83

APPENDIX F ... 84

SAMPLE SCRIPTS ... 84
Biorhytms ... 84
Graph demo ... 85
FFT example program ... 86
FFT built-in functions ... 87
Quadratic regression example ... 88
Chi-square test ... 89
Opamp ... 90
Root function ... 91

APPENDIX G ... 92

CURVE SKETCHING ... 92

APPENDIX H ... 93

USEFUL WEB LINKS ... 93

APPENDIX I ... 94

AFTERWORD ... 94

APPENDIX J ... 94

LEGAL AND DISCLAIMERS ... 94

APPENDIX K ... 94

CONTACT INFORMATION ... 94

Chapter 1

Introduction

If this is your first time reading this manual, chances are you've just installed an unregistered copy of PDAcalc classic onto your PDA to try it out, perhaps to compare it to other Palm 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 PDAcalc classic is up to the challenge. PDAcalc classic 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.* It comes with 190 built-in functions, most of which can take imaginary numbers as arguments, as well as yield imaginary results when appropriate.

PDAcalc classic was designed to be the most powerful calculator for PDA's today, allowing the user to do more complicated calculations than ever before. It was also designed to be as flexible as possible. It has a user-configurable keyboard, and is a fully programmable color-graphics calculator. Versions of PDAcalc classic exist for the Palm, PocketPC, and Windows platforms. A script written on one platform will run on all of them.

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 classic comes with many ready-to-use scripts, and we maintain a library of scripts for the PDAcalc classic for you to download at http://www.pdacalc.com This means a solution to meet your needs may already exist! PDAcalc classic 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 classic to familiarize yourself with this calculator. This manual was designed to serve as an easy-to-follow guide to PDAcalc classic, as well as a handy reference for those tackling real-world problems with our calculator.

And most of all, enjoy!

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

A Gallery of Scripts and Graphics

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

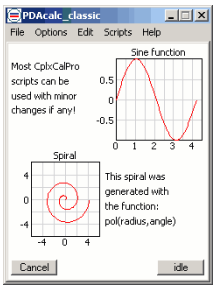


Figure 1

Draw multiple graphs and text on the graphical screen.

Read more

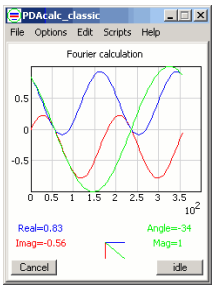


Figure 2

Use colors in graphs and text. Related text and plotted curves are displayed in the same color.

Read more

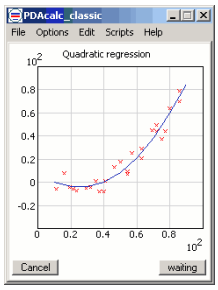


Figure 3

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

Read more

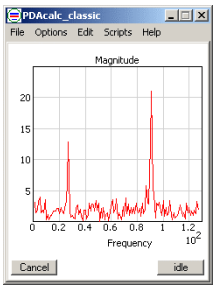


Figure 4

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

Read more

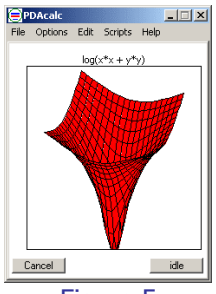


Figure 5

Draw 3D functions with just a few simple functions.

Read more

If you want to investigate the particulars of any of these graphics now, just follow their hyperlinks.

Do Your Numerical Problems Require the Power of Arrays and Matrices to Solve?

ADACS is pleased to announce MtrxCal to its suite of numerical analyses tools for the Palm platform. MtrxCal comes with the same technical specifications for numerical precision and accuracy as our users have come to expect from PDAcalc classic, and is available for download at http://www.adacs.com/mtrxcal/index.htm. Check it out!

Differences between platforms

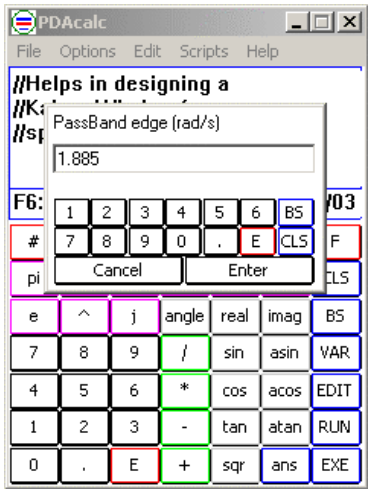


Figure 6

Windows

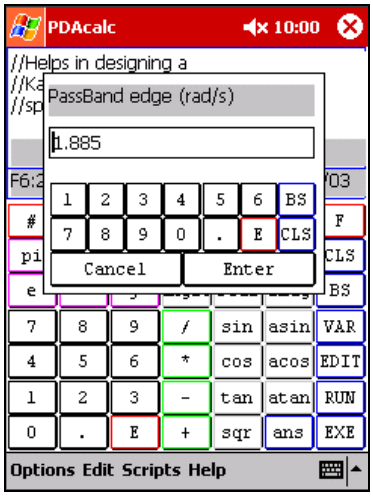


Figure 7

PocketPC

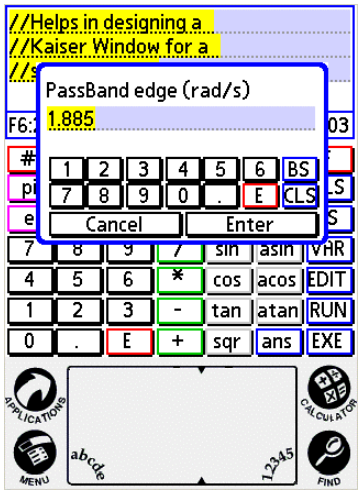


Figure 8

Palm OS

As shown above, PDAcalc classic was designed to have the same look and feel on all platforms. The only differences are the locations of the menus, available memory, 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 Main Screen

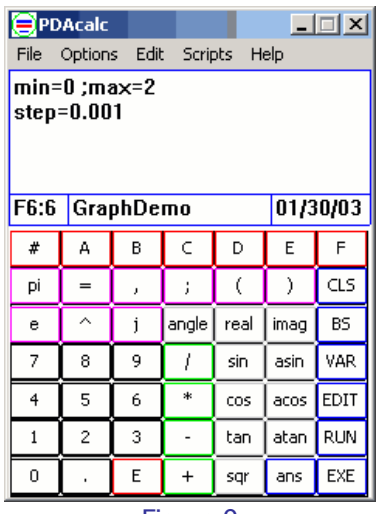


Figure 9

PDAcalc classic has two screens: a main screen and a graphics screen. This is the main screen. The top 3 lines are called the scratchpad. These show your input and intermediate results. 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 this display format of PDAcalc classic you're using, the name of the script you're running, and the date. Below that is the keyboard.

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

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	The real of x.
Im	The imaginary of x.
j	The imaginary portion of a complex number.
sqr	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 classic’s built-in functions. Choosing a function from the list places it on the scratchpad.
EDIT	Brings up the program currently loaded into PDAcalc classic for editing.
RUN	Evaluates the three lines in the scratchpad, then runs the loaded program.
EXE	Evaluates the three lines in the scratchpad only.

Basic calculations

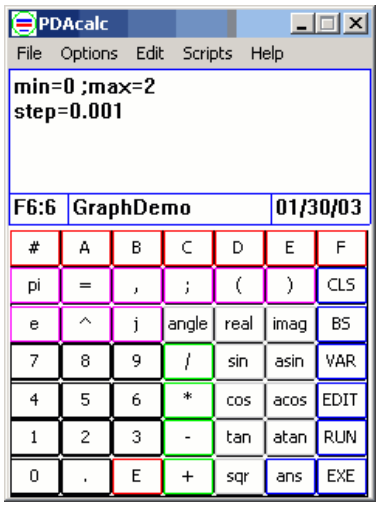
Now that we're familiar with the main screen, let's do some basic calculations on PDAcalc classic using the default preferences:

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/ 4) [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 j)	ans*(7+11j) -40+j10	Multiplying imaginary numbers
(-62+24j) / (-1+3j)	[CLS] / (5 + 3j)	Ans/(5+3j) -5+5j	Dividing imaginary numbers
sin(sqr(-1))	[CLS][sin] [sqr] – 1)) [EXE]	sin(sqr(-1)) 0 + 1.175201j	Most functions on PDAcalc classic can take complex numbers as arguments.
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, electronics engineers (like myself), commonly use the 'j' instead the 'i' since the '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 complex numbers. 4+5j*4-2j will yield a different result than (4+5j)*(4-2j)

Modifying Key Assignments

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:



All the keys were assigned by reading the text file on the right. Notice the tilde is replaced by a comma since the comma is used as a delimiter between key assignments.

These key assignment files are stored in the keyboard category of the database. Use the program editor to create or edit these files. Select [Scripts] from the menu and then select [load script]. This shows the available scripts.

```
#,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
```

Figure 10

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 classic determines if a "special" key was pressed. When a "special" key is pressed, the PDAcalc classic 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 classic executes the lines in the scratchpad (the top three lines of the display) when that key is pressed. These keywords are reserved for PDAcalc classic key assignment:

EXE	Evaluate the three lines in the scratchpad.
RUN	Evaluate the three lines in the scratchpad, then run the program.
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 Menu Items

Options

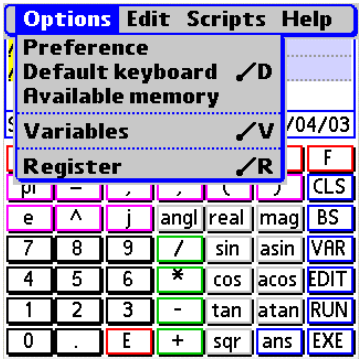


Figure 11

Preferences

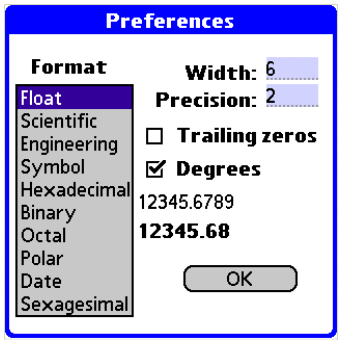


Figure 12

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

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 10 which means a number like 1234.56789 will be rounded and displayed as 1234.5679

PDAcalc classic default preferences:

format ……...……..	float
width ……	6
precision ……...……..……	4
angular measurement ……..……..	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)
tr: trailing zeros. (0 or 1)

Preferences for angular measurement can be changed with the functions:

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

Display formats

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 classic.
123456.789	float	7	3	123456.789	ah-hah! PDAcalc classic 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 classic rounds off the decimal part of input, then hexes the result.
1234.567	binary	4	9	10011010010	PDAcalc classic 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

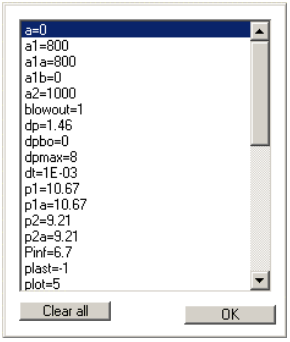


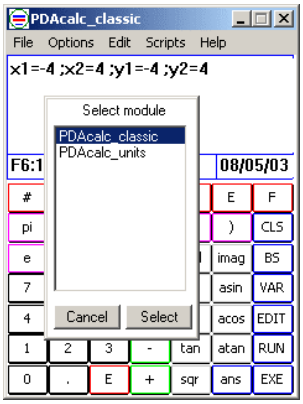
Figure 13

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. When you see this on the main screen, just select [Options] => [Variables] and [Clear all] to continue.

Default Keyboard

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

Select module



Use the select module screen (under the options menu) to select a different module. At the time of writing this was not available on the palm platform.

The windows and pocketpc version checks for available libraries, DLL's, in the PDAcalc directory and shows them in the list.

It will only be a matter of time before more modules will be available.

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

Register

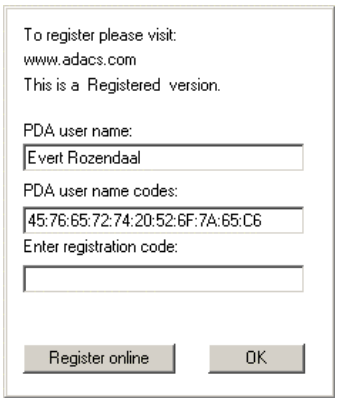


Figure 14

You can use PDAcalc classic 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 classic; most importantly, this allows the user to write new scripts. 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 the information needed to generate the registration code for the different platforms 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. Only the information needed to generate your registration code(s), PDA user names, are transferred to our web site. After clicking the [Register online] button, you can watch the URL of the web site to see the information transferred.
Please select the right module before registering online.

Register online

Figure 15

After tapping the [Register online] button on the register screen of the Windows version, your PDA user names will be transferred to our web site and the screen shown at the left appears. Select the platform(s) you'd like to register, and a discount will be calculated. The more platforms you register, the higher the discount. During our introductory offer period, there is an additional 15% discount. This might vary, depending on when you register.

After entering your information, press [Continue order]; and a printout of your order will be shown. We recommend printing this page for your own records. Press [Continue order] again, and you will be transferred to a **secure server** to enter 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.

Edit

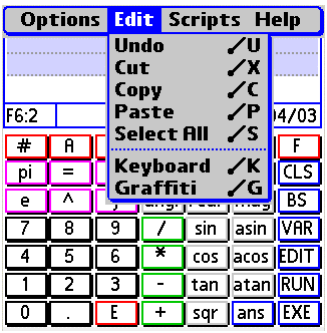


Figure 16

This is the standard [Edit] menu. This menu is a little different on the different platforms depending on the standards of the platform. PDAcalc classic 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

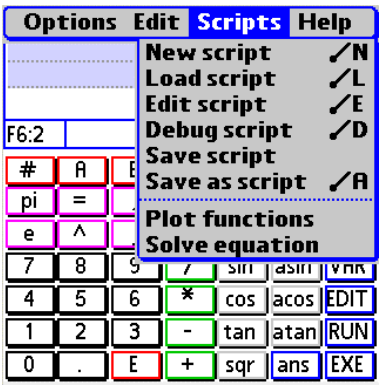


Figure 17

Load scripts

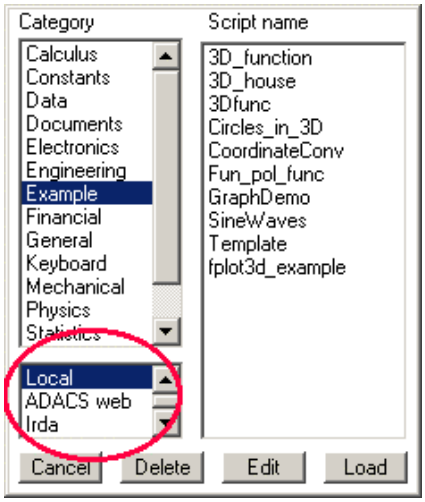


Figure 18

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 are supported by PDAcalc.

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 either delete, edit or load the script.

Edit script

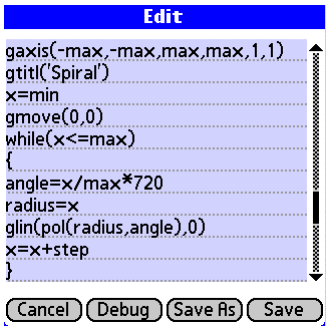


Figure 19

Edit on palm system

On the Windows version, the edit screen is bigger and shows line numbers. The smaller screen size on the Palm is why line numbers are not shown on the Palm version.

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.

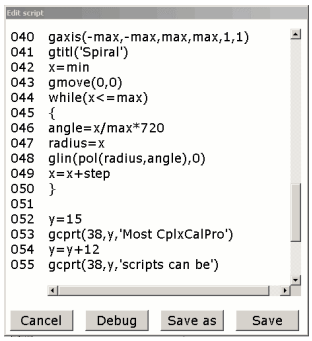


Figure 20

Edit on windows

Debug scripts

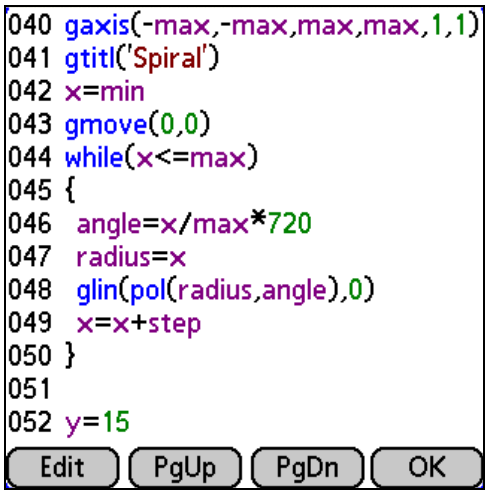


Figure 21

Debug script on palm

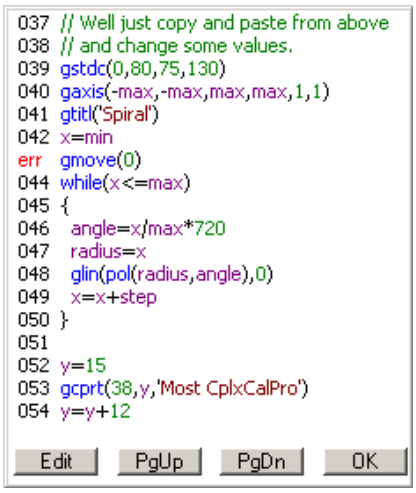


Figure 22

Debug script on windows

The debug screen will show the scripts color coded. It will also show the first error if the script contain an error as shown on the right. 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.

Worksheets

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

Plot function

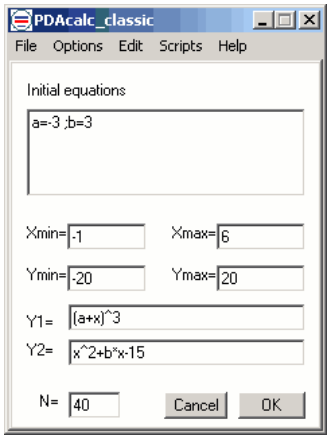


Figure 23

Select [Plot function] from [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.

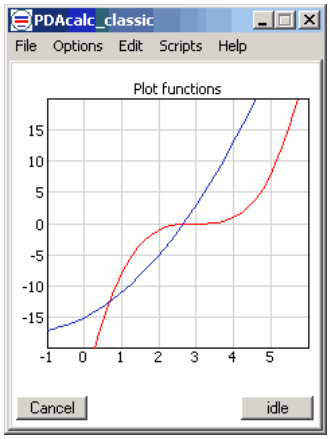


Figure 24

The next six values set 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.

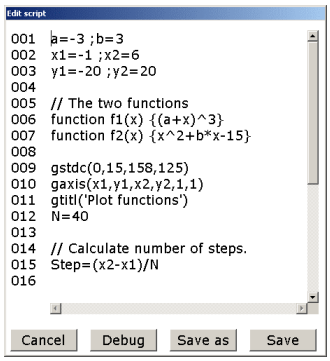


Figure 25

Solve Equation

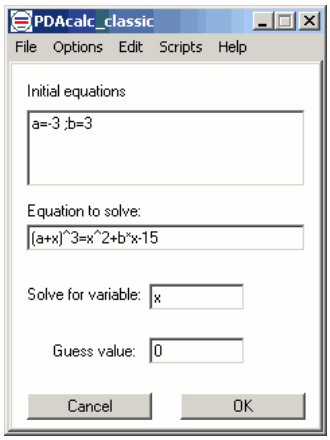


Figure 26

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 27

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 the 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.

Help

Functions

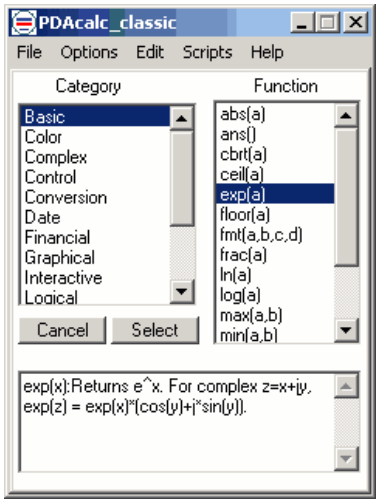


Figure 28

This will show a list of all the available built-in functions. Select a category and tap on a function for a little description of the function.

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 of schools might have there own special copy which this screen indicates.

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 available only 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 platforms you like to register. Notice that when you select a different module from the [Select module] item in the [Options] menu you will be transferred to a different web page.

Visit our web site

Clicking this menu item will open your browser and show our main web site when connected to the internet.

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 them.

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. This is where you can update the descriptions of the scripts. When your script contains an error, the script can not be uploaded. You will have to fix the error before uploading.

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.

Chapter 2

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%	10.075
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 Δx, and the definition of a derivative, $f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$, comes from the difference quotient of function *f*. *h* then is the difference between two values of *x*, x_0 and x_1. 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

First, enter them into a PDAcalc classic text file. Press [EDIT] from the text screen. If you have no program loaded, PDAcalc classic takes you to its database of text files. If you do have a program loaded, PDAcalc classic places that program in the editor and brings it up; in that case, press [Done]. You should now be in the database of text files.
Press [New] to open a new text file. Give it a name on the first line. We'll call ours "Stat Data". You can begin writing your data on the next line, separating each variable with a comma. Once you finish, your file should look like this:

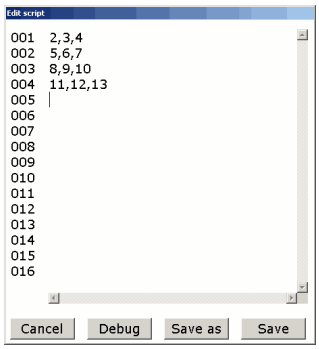


Figure 29

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.

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.

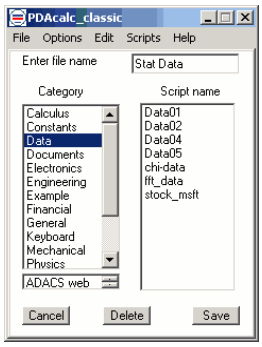


Figure 30

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.

```
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.
```

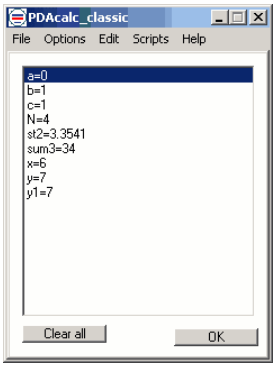


Figure 31

PDAcalc classic 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.
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

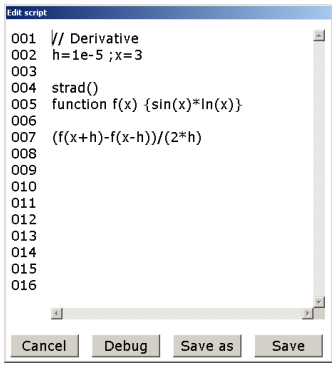


Figure 32

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.

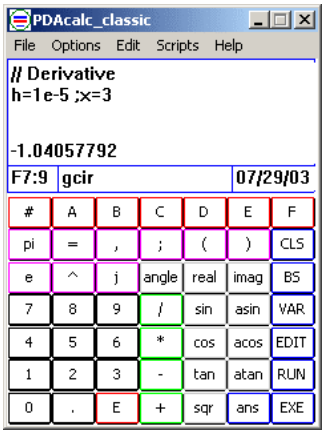


Figure 33

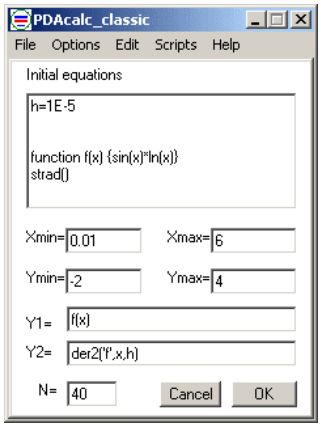


Figure 34

User-defined functions can also be used when plotting functions using the plot function worksheet. 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)**.

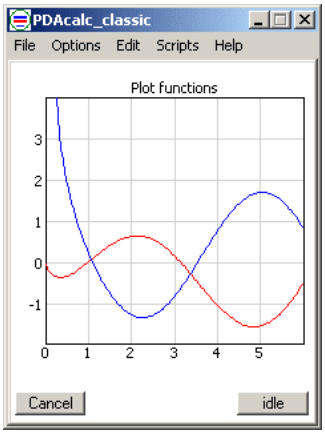


Figure 35

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].

Chapter 3

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:

pixel (0, 0) **pixel (160, 0)**

pixel (0,160) **pixel (160,160)**

Figure 36

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 **Prog.** menu item, select **Edit program**; and in the program database display, choose **New**. 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 this:

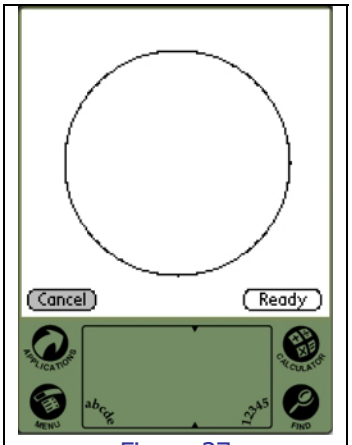


Figure 37

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

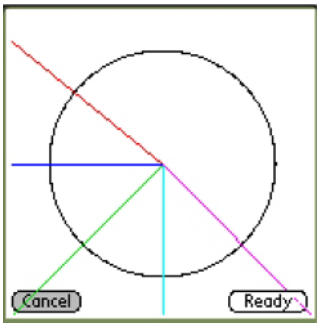


Figure 38

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)

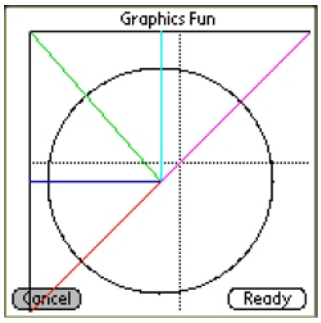


Figure 39

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)

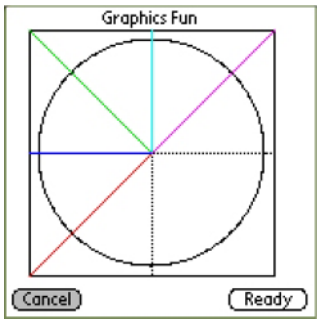


Figure 40

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.

Chapter 4

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/PDAcalc classic/downloads.htm. 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 on your PDA, change the program name (the first line); 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. If you have not registered PDAcalc classic yet, this might make you eligible to receive the registration code for free. See our web page for more information.*

A Programming Primer

What is a script?* For the purposes of this manual, let's define scripting as a set of instructions run on PDAcalc classic, 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.
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.

---

* We've scoured our references in search of the concise, comprehensive, yet witty definition to enlighten those PDAcalc classic users who, though like all PDAcalc classic users demonstrate good taste and sound judgment in using our product, are nonetheless new to the joys of programming. We have returned empty-handed. This surely saddens us as much as it does you; for where we'd hoped to cut-and-paste we must now think and write, and where you've anticipated reading sweetness and light you must now read what we write.

▽ Again, this assumes that you've ensured the soundness of your program. Later in the primer we'll touch upon points to consider when testing for soundness.

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.

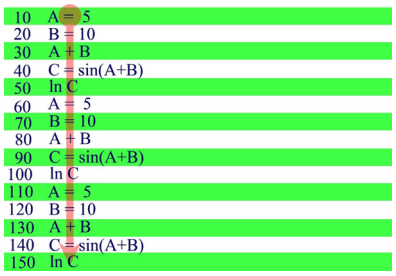


Figure 41

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

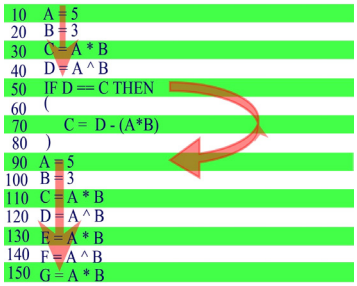


Figure 42

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

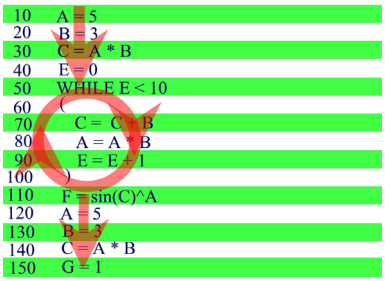


Figure 43

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 Programming 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:

* OK; the typography of these operators is decidedly UNIX, whence comes C, C++, Java, and Perl, all of which use the same typography for their relational operators. Can you tell which language PDAcalc classic is written in? (Hint: those of you who answer correctly earn an A++.)

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)
```

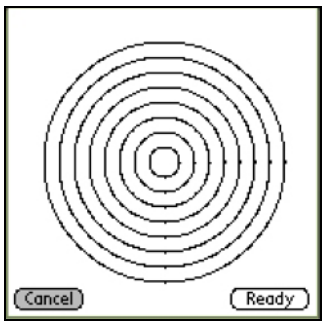


Figure 44

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 while(r<=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:

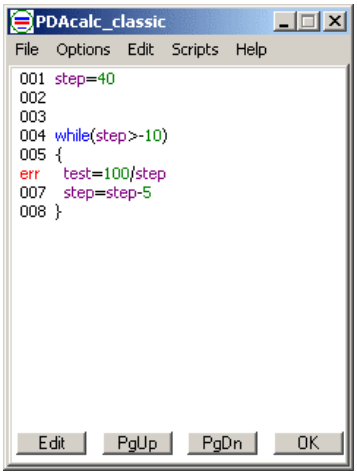


Figure 45

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.

Chapter 5

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 }
```

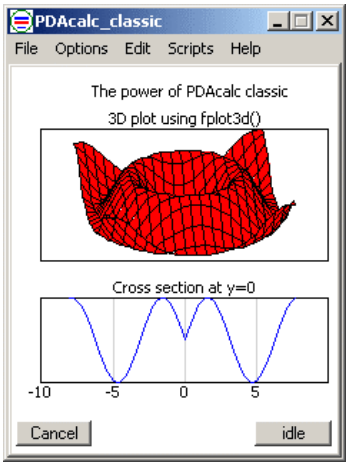


Figure 46
rotUD=-120

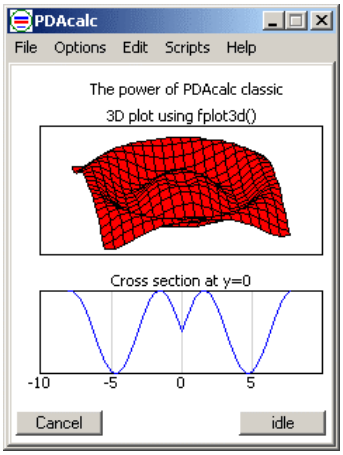


Figure 47
rotUD=-75

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.

```
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 fplot3d('f',-360,-360,360,360)
010 gtitl('3D function')
```

Next fplot3d() will allocate memory to store 20 X 20 values. 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.

3D graphing

```
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 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,c,s,0)
037  gline3d(3,0,c,s)
038 }
```

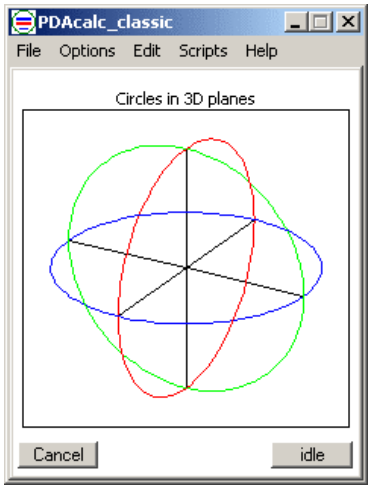


Figure 48

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.

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.

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. Line with a remark sign in front will be ignored.

Chapter 6

Using excel

To make sure the calculation are performed correctly in excel ☺ you can easily transfer the data files between excel and PDAcalc classic. The data files are normally stored in the directory:
C:\Program Files\ADACS\PDAcalc\PDAcalc_classic\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.

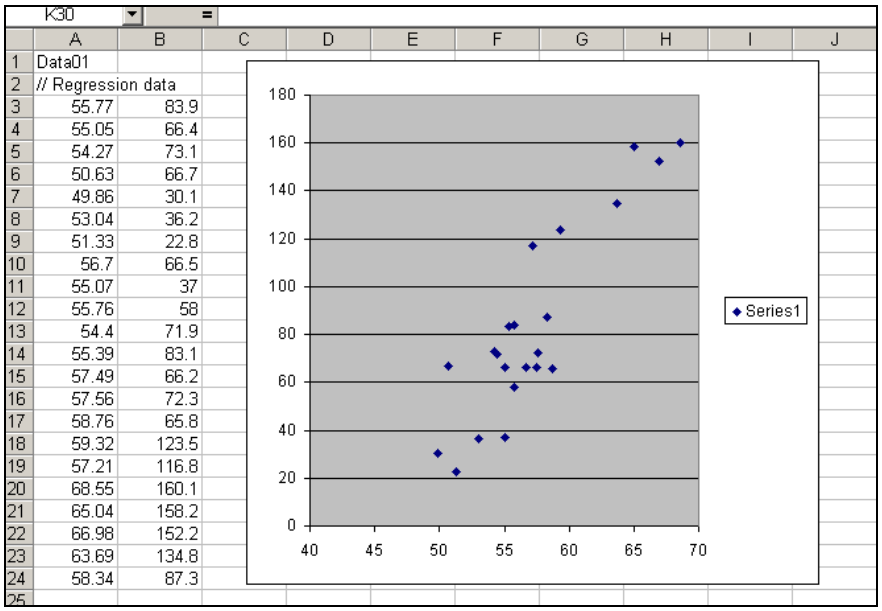


Figure 49

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.

Appendix A

Functions, Operators, and Commands

Basic functions

Function ***Description***

\+

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 parenthesis 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

---

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
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.

frac(n)

Calculates the factorial.

$$frac(n)=\begin{cases} n*(n-1)...2*1 & n=1,2,... \\ 1 & n=0 \end{cases}$$

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

Color functions

Function	*Description*

---

gselcol(idx)

Select a color.

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)			
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

Function	*Description*

---

arg(cplx)	*returns the angle of a complex number*
	
	a=arg(3+3j) // 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

Function	*Description*

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

Function	*Description*

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:** 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

Function	*Description*

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.

nper is the total number of payment periods in an annuity.

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.

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

Function	*Description*

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

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 middel 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

Graphical functions

Function	*Description*

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(x,y,r)

Draw circle at x,y with radius r.

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.

Example

```
001 per=0.06
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.*

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).*
gpnt(x,y,t)	*Draw symbol t at x,y.*
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
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.*
	
	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.
	
	See also: gaxis

---

gtadd(strV)	*Add the value or text of strV to the previously drawn text.* **See also:** gprt , gcprt , glprt

---

gvlin(x)	*Draw vertical line at x position.*

Interactive functions

Function	*Description*

---

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 // Puts the 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 statements below 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 // Puts the 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 statements below 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

Function	***Description***
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

Function	*Description*
!=	*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

Function	*Description*
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
erf(x)	*Error function of x.* The error function is defined by $$\mathrm{erf}(x) = \frac{2}{\mathrm{sqrt}(pi)} \int_0^x e^{-t^2}\, dt$$

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

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

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.

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

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

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

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).*

---

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

Function	***Description***

---

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.*

---

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.*

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.*
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.*

Trigonometric functions

Function	*Description*
acos(a)	*Arccosine*
acosh(a)	*Inverse hyperbolic cosine.*
asin(a)	*Arcsine*
asinh(a)	*Inverse hyperbolic sine.*
atan(a)	*Arctangent*
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.*

Appendix B

Specifications

This NON-RPN calculator has the following technical specifications:

- 3D graphics
- Linear regression
- Financial functions
- Statistical functions
- Root finding functions
- FFT (fourier transforms)
- Color coded debug screen
- Syntax checking on scripts
- Fully configurable keyboard
- Auto-scaling of the graphs
- Equation solver worksheet
- Fully reconfigure the keyboard
- Complex numbers and functions
- 3D functions, draw lines in 3D space, rotate objects in 3D space etc.
- Calculus functions, integral, derivatives, etc.
- Plot functions without any programming
- Browse our web site for free user scripts
- Load and run user scripts directly from our web site
- IEEE-754 64-bit Double Precision, a floating point format ranging from -2.23E-308 to 1.79E+308.
- Display formats: Float, Scientific, Engineering, Symbol, Hexadecimal, Binary, Octal, Polar, Date and Sexagesimal.
- Angular units: Radians and Degrees

Appendix C

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.

Appendix D

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.

Appendix E

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	m3 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	$m^{-3}\ mol^{-1}$
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

Appendix F

Sample 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)
```

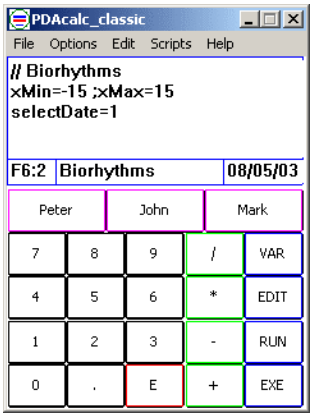


Figure 50

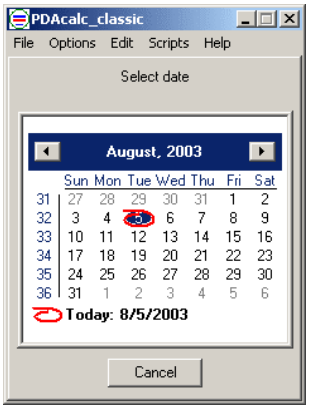


Figure 51

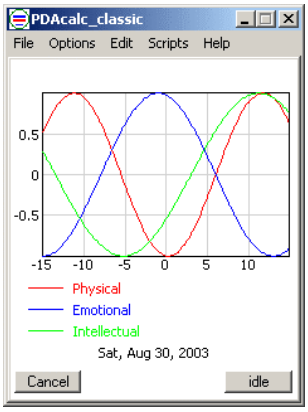
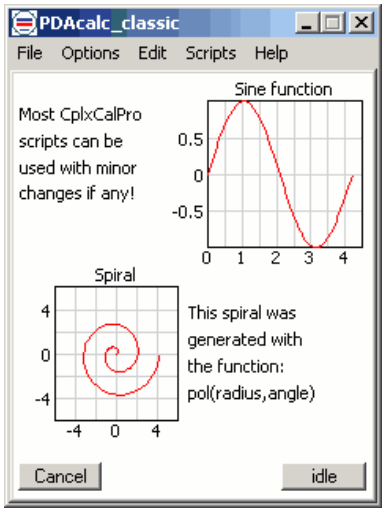


Figure 52

Graph demo

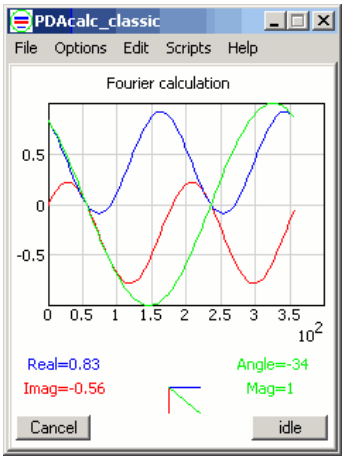
```
001 // Example graph script
002 min=0;max=4.2
003
004
005 step=(max-min)/100
006
007 // Indicate where to put the graph
008 // on the screen.
009 gstdc(70,10,159,65)
010
011 // Minimum and maximum values
012 // on the graph. The last two
013 // arguments indicate if the values
014 // should be shown.
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)')
```



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.

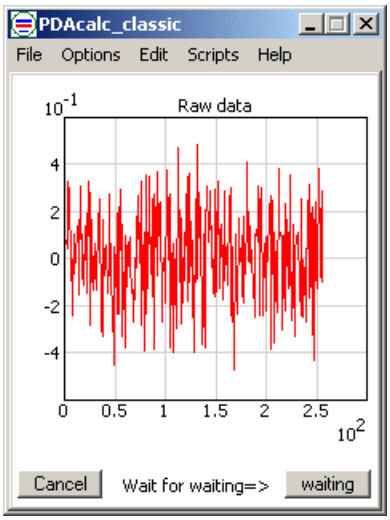
```
Phase=124
Xmin=0;Xmax=360
Ymin=-1;Ymax=1

stdeg()
fmt(0,6,6,0)

gtitl('Fourier calculation')
gstdc(0,15,159,100)
gaxis(Xmin,Ymin,Xmax,Ymax,1,1)
N=100
Step=(Xmax-Xmin)/N
x=Xmin

Real=0
Imag=0
while(x<=Xmax)
{
 y3=sin(x+Phase)
 y1=sin(x)
 Imag=Imag+y1*y3
 y2=cos(x)
 Real=Real+y2*y3
 if(x==Xmin)
 {
  gmove(x,y1*y3)
  gmove2(x,y2*y3)
  gmove3(x,y3)
 }
 else
 {
  glin(x,y1*y3)
  glin2(x,y2*y3)
  glin3(x,y3)
 }
 x=x+Step
}

greset()
x=80
y=135
mul=20
Real=Real/(N/2)
Imag=Imag/(N/2)
//gfcir(x,y,mul)
yMul=y-Imag*mul
xMul=x+Real*mul
gline(x,y,x,yMul)
gline2(x,y,xMul,y)
gline3(x,y,xMul,yMul)
fmt(0,3,2,0)

// Select color of line 2
gselcol(2)
grprt(30,125,'Real=',Real)

// select color of line 1
gselcol(1)
grprt(30,135,'Imag=',Imag)

v=Real+0+1j*Imag

// Select color of line 3
gselcol(3)
grprt(140,125,'Angle=',arg(v))
grprt(140,135,'Mag=',abs(v))
```

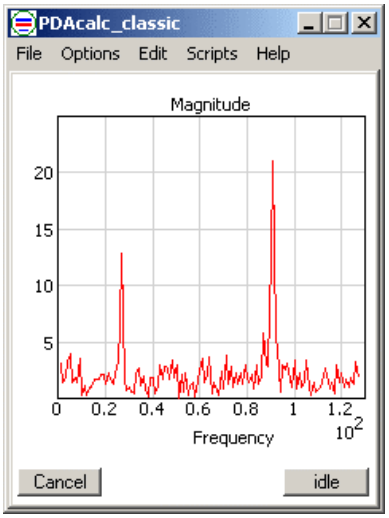


For more information about fourier transforms please visit our website.

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')
```

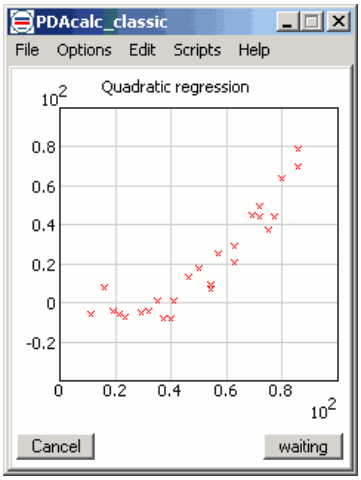


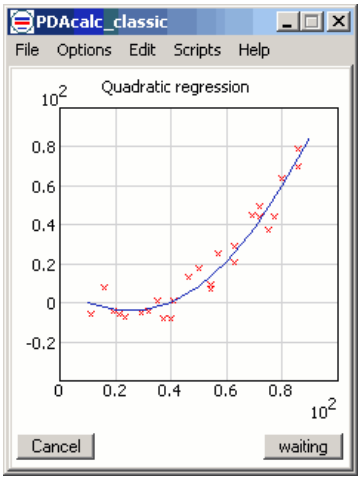


The sdata(datafile) returns the number of rows read from the data file. The function srplot(1,1,N,0) plots the raw data. Notice that there is not need to setup anything for the labeling of the plot since that is all done for you by the srplot() function.

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






```
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 while(i<=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 $\mathbf{2}n^{2/5}$ 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:

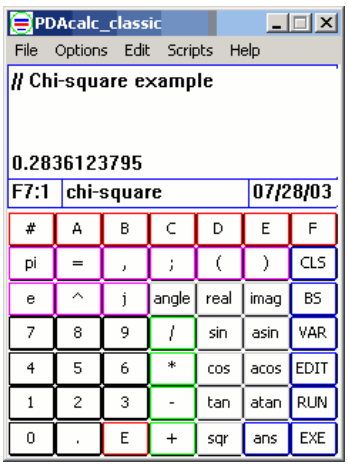
The results of an experiment are shown below in the row observed.

	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	

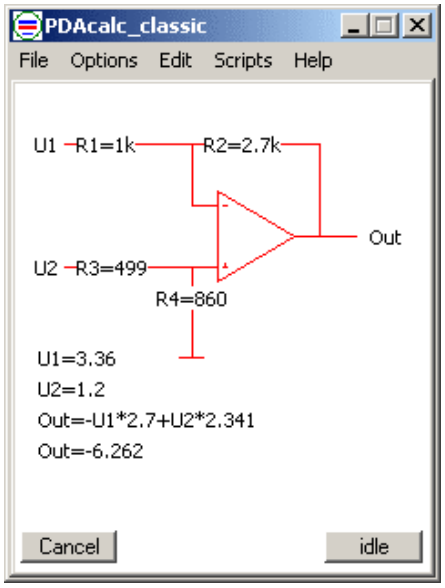
$$\chi^2 = \sum \frac{(observed - \exp ected)^2}{\exp ected} = 0.283612379$$

```
// 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)
```



Opamp



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

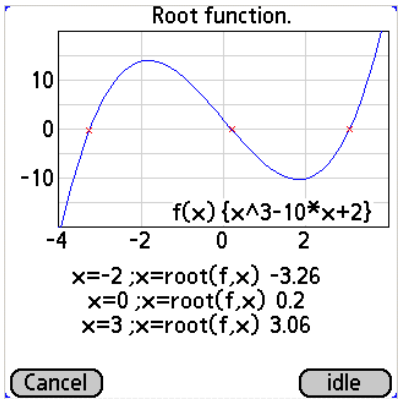


Figure 53

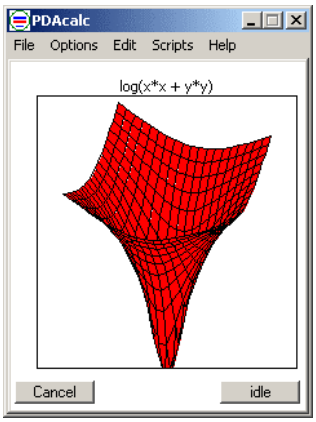


Figure 54

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)
```

```
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)')
```

Appendix G

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.

Appendix H

Useful Web Links

URL	what you'll find there
http://www.adacs.com	us!
http://physics.nist.gov/cuu/index.html	The NIST Reference on Constants, Units, and Uncertainty, including in-depth information on the metric system
http://mathworld.wolfram.com/	Eric Weisstein's World of Mathematics Possibly the best mathematics reference work on the World Wide Web
http://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.
http://www.acm.org/	ACM Association for Computing Machinery, the world's first educational and scientific computing society.

Appendix I

Afterword

This manual is intended to serve as a tutorial and reference to PDAcalc classic. 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 classic 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.

Appendix J

Legal 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.

Appendix K

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*

Any comments or suggestions are very welcome!