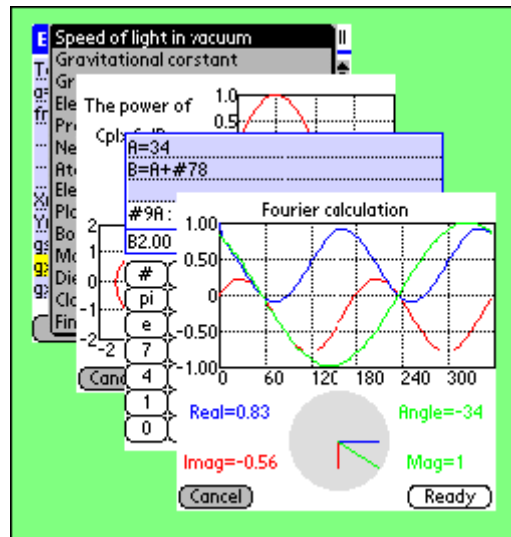


CplxCalPro

The calculator for numerical analyses on the
palm® Operating System.



User manual

Version 4.15

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

TABLE OF CONTENTS..... 2

CHAPTER 1..... 6

INTRODUCTION..... 6

A GALLERY OF PROGRAMS AND GRAPHICS..... 7

CHAPTER 2..... 8

BASICS..... 8

THE MAIN SCREEN..... 8

BASIC CALCULATIONS..... 10

MODIFYING KEY ASSIGNMENTS..... 12

THE MENU ITEMS..... 13

OPTIONS..... 13

Preferences..... 13

Display formats..... 14

Variables..... 16

Clear all..... 16

Clear memory..... 16

Default Keyboard..... 16

Memory..... 17

Register..... 17

EDIT..... 17

PROG..... 18

Load program..... 18

Edit program..... 19

Plot function..... 19

Solve Equation..... 21

Copy text program..... 22

Delete text program..... 22

Create text program..... 22

HELP..... 22

Functions:..... 22

Constants:..... 23

Site licenses:..... 23

Legal agreement..... 23

CHAPTER 3..... 24

PUTTING IT ALL TOGETHER..... 24

CHAPTER 4.....	27
BUILT-IN FUNCTIONS ON CPLXCALPRO	27
COMPLEX:	27
BASIC:	27
TRIGONOMETRIC:	28
CALCULUS:	28
FINANCIAL:	29
LOGICAL:	29
BASE CONVERSION:	29
PROBABILITY & STATISTICS:	30
CHAPTER 5.....	33
USER-DEFINED FUNCTIONS	33
CHAPTER 6.....	34
GRAPHICS ON CPLXCALPRO	34
GRAPHICS EXAMPLES	35
CHAPTER 7.....	39
PROGRAMMING CPLXCALPRO	39
A PROGRAMMING PRIMER	39
CPLXCALPRO'S PROGRAMMING COMMANDS:	41
PROGRAMMING EXAMPLES:	42
APPENDIX A	48
TECHNICAL SPECIFICATIONS	48
APPENDIX B	49
DATA FORMATS	49
FLOAT:	49
ENGINEERING:	49
SYMBOL:	49
HEXADECIMAL:	50
BINARY:	50
OCTAL:	50
POLAR:	50
DATE:	50
SEXAGESIMAL:	50

APPENDIX C	51
DISPLAY FORMAT	51
APPENDIX D	52
FUNCTIONS, OPERATORS, AND COMMANDS	52
BASE CONVERSION:	52
BASIC:	52
CALCULUS:	53
COLOR:	53
COMPLEX:	53
CONVERSION:	53
DATE:	53
FINANCIAL:	55
FLOW CONTROL:	55
FORMAT:	55
GRAPHICS:	56
INTERACTIVE:	58
LOGICAL:	58
RELATIONAL:	58
SPECIAL:	59
PROBABILITY & STATISTICS:	59
STRING:	60
TRIGONOMETRIC:	62
APPENDIX E	63
CONSTANTS	63
APPENDIX F	64
SAMPLE PROGRAMS	64
GRAPH DEMO:	64
FFT EXAMPLE PROGRAM:	64
FFT BUILT-IN FUNCTIONS:	66
QUADRATIC REGRESSION EXAMPLE:	66
CHI-SQUARE TEST:	67
OPAMP:	68
ROOT FUNCTION:	68
APPENDIX G	69
THE PALM® OS (POS) EMULATOR	69

APPENDIX H..... 70

CURVE SKETCHING 70

APPENDIX I..... 71

USEFUL WEB LINKS 71

AFTERWORD..... 72

Chapter 1

Introduction

If this is your first time reading this manual, chances are you've just hotsync'd an unregistered copy of CplxCalPro 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 as 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 CplxCalPro is up to the challenge.

CplxCalPro is a power user's non-RPN calculator. It uses a 64-bit double-precision floating-point format, providing an approximate numerical range of $-2.23\text{E}-308 \leq n \leq 1.80\text{E}308$.^{*} It comes with 190 built-in functions, most of which can take imaginary numbers as arguments, as well as yield imaginary results when appropriate.

CplxCalPro was designed to be the most powerful calculator for palm PDAs today, allowing the user to do more complicated calculations than ever before. It was also designed to be as flexible as possible.

Tailor CplxCalPro to meet your needs. It has a user-configurable keyboard, and is a fully programmable color-graphics calculator. 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, CplxCalPro hotsync'd onto your PDA comes with many ready-to-use programs in its database, and we maintain a library of programs for the CplxCalPro for you to download at <http://www.adacs.com/CplxCalPro/downloads.htm>. This means a solution to meet your needs may already exist!

CplxCalPro has a memopad-like built-in database to hold, retrieve, and manage programs. Programs can be grouped in the database by category. And the number of programs the database can hold is limited only by the amount of free memory in your PDA.

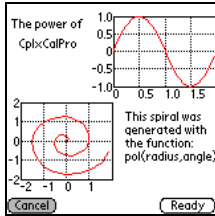
Please take a few moments to sit down with this manual and CplxCalPro to familiarize yourself with this calculator. This manual was designed to serve as an easy-to-follow guide to CplxCalPro, as well as a handy reference for those tackling real-world problems with our calculator.

And most of all, enjoy!

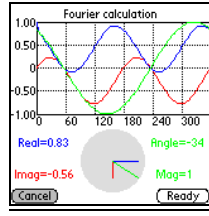
^{*} Refer to [Appendix A, Technical Specifications](#), for the exact range.

A Gallery of Programs and Graphics

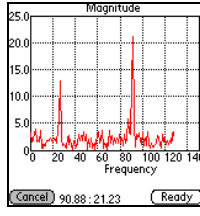
For those of you who want to know if looking into CplxCalPro is worth your time, we offer here a small gallery of graphics to illustrate its power:



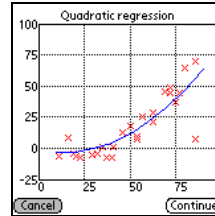
[Draw multiple graphs and text on the graphical screen.](#)



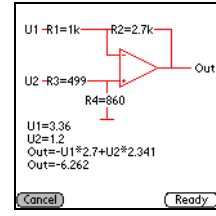
[Use colors in graphs and text. Related text and plotted curves are displayed in the same color.](#)



[CplxCalPro has a suite of built-in functions that makes doing sophisticated analyses a breeze.](#)



[Do statistical analyses on large sets of data. Here CplxCalPro plots the data points, then draws their trendline.](#)



[Draw a diagram. In this program, change values on the main screen and press \[RUN\]. CplxCalPro redraws the diagram and recalculates the output voltage.](#)

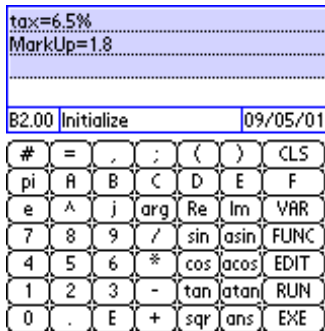
If you want to investigate the particulars of any of these graphics now, just follow their hyperlinks.

Chapter 2

Basics

The “Cplx” in “CplxCalPro” stands for “complex”; and though it’s supposed to stand for this calculator’s ability to handle [complex numbers](#), it can just as easily stand for the complexity inherent in the power of our product. So let’s look at its features by taking CplxCalPro one step at a time.

Whenever CplxCalPro starts, it goes through a startup sequence:

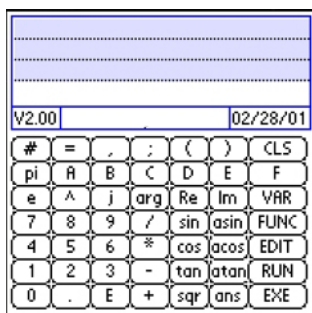


On startup, CplxCalPro tries to find a user-created file in its database called "Initialize". If it exists, CplxCalPro skips the title line and evaluates the second, third, and fourth lines of the file. The variables on those lines are displayed on the first, second, and third lines of the screen and also stored in permanent variable space. This is where commonly used variables should be assigned that will be used in several programs over the course of a session. More than one variable assignment to a line is possible by separating assignments with a semi-colon (;).

Variable assignments made on lines below line 4 of the file are stored in program space that CplxCalPro clears before loading a program. After checking for the initialization program, CplxCalPro will, depending on the settings in the preference form screen, load the previously used program. For more on these settings, please refer to the section on the [preference screen](#).

After CplxCalPro initializes, you see:

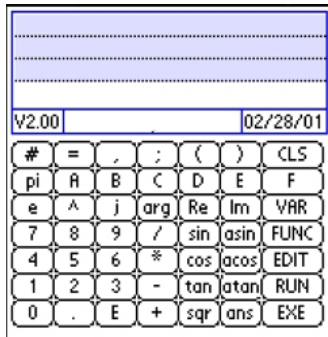
The Main Screen



CplxCalPro 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 which version of CplxCalPro you’re using, the name of the program 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.

The Keyboard



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 CplxCalPro’s built-in functions. Choosing a function from the list places it on the scratchpad.
EDIT	Brings up the program currently loaded into CplxCalPro 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 CplxCalPro using the [default preferences](#):

<i>The Problem Statement:</i>	<i>You press:</i>	<i>CplxCalPro Displays:</i>	<i>Remarks:</i>
3 + 4	3 + 4 [EXE]	3+4 7	Pretty straightforward.
(3+4) * 2	* 2 [EXE]	ans*2 14	Use the previous result for the current calculation.
2^((3+4)*2)	2 ^ [ans] [EXE]	2^ans 16,384	ans key supplies the previous result in the equation.
(2^((3+4)*2))^(1/14)	[ans] ^ (1 / 14) [EXE]	ans^(1/14) 2	Fractional exponents.
same as above	[CLS] (2 ^ ((3 + 4) * 2)) ^ (1 / 14) [EXE]	(2^((3+4)*2))^(1/14) 2	CplxCalPro follows algebraic order of precedence when evaluating expressions.
same as above, but with a deliberate error	with your stylus, delete a) from the expression in the scratchpad, then press [EXE]) missing	CplxCalPro's interpreter does syntax error-catching.
(-1)^(1/2)	[sqr] - 1) [EXE]	sqr(-1) 0 + j1	Imaginary numbers! Note that a prepended 'j' designates the imaginary part of a complex number CplxCalPro.*
<i>The Problem Statement:</i>	<i>You press:</i>	<i>CplxCalPro Displays:</i>	<i>Remarks:</i>

* Most math books use the postpended 'i' to designate the imaginary part of a complex number; e.g. $\sqrt{-1} = 1i$. CplxCalPro, however, was developed for practical use. 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. $4+j5 \cdot 4-j2$ will yield a different result than $(4+j5) \cdot (4-j2)$

$(0 + j1) + (2 + j3)$	$+(2 + j3)$	ans+(2+j3) 2+j4	Adding imaginary numbers
$(2+j4) - (3+j)$	$-(3 + j)$	ans-(3+j) -1+j3	Subtracting imaginary numbers
$-(1+j5) * (7+j11)$	$*(7 + j 11)$	ans*(7+j11) -40+j10	Multiplying imaginary numbers
$(-62+j24) / (-1+j3)$	$/ (-1 + j 3)$	Ans/(5+j3) -5+j5	Dividing imaginary numbers
$\sin(\text{sqr}(-1))$	[CLS][sin] [sqr] - 1)) [EXE]	$\sin(\text{sqr}(-1))$ 0 + j1.175201	Most functions on CplxCalPro can take complex numbers as arguments.
let A=4*5	[CLS] A = 4 * 5 [EXE]	A=4*5 20	Variable assignments.
A/3	/ 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
$\text{sqr}(A^2+B^2)$	[CLS] [sqr] (A * A + B * B) [EXE]	$\text{sqr}(A^2+B^2)$ 5	$\text{sqr}(3^2+4^2)$ Entering the problem using the ^ key yields the same result. CplxCalPro follows the algebraic order of precedence

Modifying Key Assignments

The user can assign all keys on the main screen. A simple text file, 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:

V2.00				11/19/01		
#	=	,	;	{	}	CLS
pi	A	B	C	D	E	F
e	A	j	arg	Re	Im	VAR
7	8	9	/	sin	asin	FUNC
4	5	6	*	cos	acos	EDIT
1	2	3	-	tan	atan	RUN
0	.	E	+	sqr	ans	EXE

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.

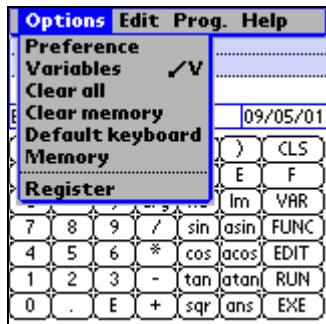
```
#,=,~,;,(),CLS
pi,A,B,C,D,E,F
e,^,j,@arg,@Re,@Im,VAR
7,8,9/,@sin,@asin,FUNC
4,5,6,*,@cos,@acos,EDIT
1,2,3,-,@tan,@atan,RUN
0,.,E,+,sqr,ans,EXE
```

These key assignment files are stored in the keyboard category of the database. Use the program editor to create or edit these files. Select [\[prog.\]](#) from the menu and then select [load program]. This shows the contents of the database. Next select the keyboard category and you should see at least two files, “Default keys” and “Logic keys”. Make sure the edit box at the bottom is not checked and select “Logic keys”. This should change the main screen. Now tap on the edit button in the main screen to see the key assignment file. After a key is pressed, CplxCalPro determines if a "special" key was pressed. When a "special" key is pressed, the CplxCalPro executes a "special" function accordingly. Reserved keywords determine if a key is “special”. If the keyword EXE is assigned to a key, for instance, CplxCalPro executes the lines in the scratchpad (the top three lines of the display) when that key is pressed. These keywords are reserved for CplxCalPro key assignment:

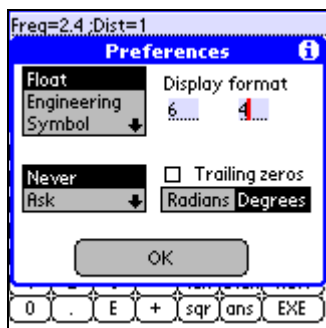
- | | |
|------|--|
| EXE | Evaluate the three lines in the scratchpad. |
| RUN | Evaluate the three lines in the scratchpad, then run the program. |
| EDIT | Edit the program. |
| FUNC | Show a list of all the functions. tapping one on the list puts it in the scratchpad. |
| VAR | Show a list of all the assigned variables. |
| @ | 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 <code>iskey()</code> 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



Preferences



The preferences form allows you to set the display format of numbers, to set angular measurements in radians or degrees; and to specify whether the program used at the close of a CplxCalPro session should be reloaded at the start of the next session.

The following formats are supported: Float, 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

CPLXCALPRO DEFAULT PREFERENCES:

format	float
width	6
precision	4
angular measurement	degrees
trailing zeros	no
ask if previously-loaded program should be reloaded...	never

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-eng, 2-sym, 3-hex, 4-bin, 5-oct, 6-pol, 7-date, 8-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:

<u>Number</u>	<u>Format</u>	<u>Width</u>	<u>Precision</u>	<u>Display</u>	<u>Comments</u>
123456.789	float	7	2	123,456.79	Default settings of CplxCalPro.
123456.789	float	7	3	123,456.789	ah-hah! CplxCalPro 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.
123456.789	engineering	1	9	123.456789E03	Engineering formats by $10^{(3n)}$, where $n \geq 0$.
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	
<u>Number</u>	<u>Format</u>	<u>Width</u>	<u>Precision</u>	<u>Display</u>	<u>Comments</u>
1234.567	hexadecimal	4	9	#4D2:1,235	CplxCalPro rounds off the decimal part

					of input, then hexes the result.
1234.567	binary	4	9	10011010010	CplxCalPro 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



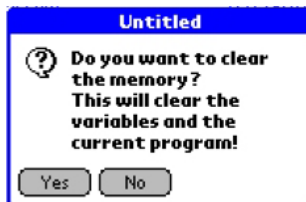
The variables screen (under the options menu) not only shows you variables and their values, but also allows you to change those values. Just select the variable whose value you want to change, and the value will appear on the edit line. Now change the value behind the equal sign and press [Update].

Clear all



Think of this as CplxCalPro's soft-reset. This clears variable values and the currently loaded program, and reloads the default keyboard.

Clear memory



This clears variable values and the currently loaded program, but leaves the user-defined keyboard *as is*.

Default Keyboard

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

Memory

Allocated memory

Strings: 1%

Misc. memory: 1%

Functions steps: 0%

Free memory: 21438

OK

The total amount of memory is divided into three separate memory spaces. One is dedicated for the storage of strings. A miscellaneous storage space is used for what the name suggests, and the last one is used for storage of the function steps.

This screen will show you how the total available memory is used for the current program.

Register

Register

To Register: www.handango.com

*** Registered version ***

This version has all the features.

User Name:
james

Enter registration code:
.....

OK

You can use CplxCalPro 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 CplxCalPro; most importantly, this allows the user to write new keyboard layouts, graphics files, and programs.

Edit

Options	Edit	Prog.	Help
	Undo	/U	
	Cut	/X	
	Copy	/C	
	Paste	/P	
	Select All	/S	
	Keyboard	/K	
	Graffiti	/G	
82.00			5/01
# =			LS
pi A			F
e A			VAR
7 8 9	/ sin asin		FUNC
4 5 6	* cos acos		EDIT
1 2 3	- tan atan		RUN
0 . E +	sqr ans		EXE

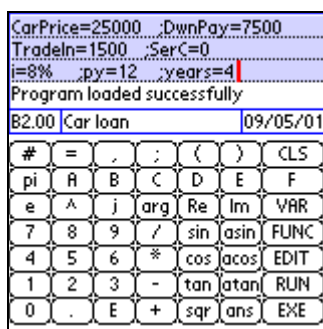
This is the standard Palm [Edit] menu. CplxCalPro allows you to copy-and-paste values from the result line into the scratchpad, or to share information across applications (e.g., to copy values from the result line into a memo).

Prog.**Load program**

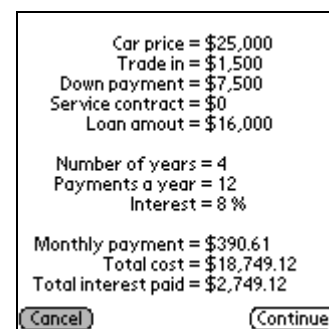
Selecting [Load program] from the [Prog.] menu will display programs available in the database similar to the screen on the left.



Tapping a program's name loads the program into the program buffer of CplxCalPro's runtime interpreter. During the load, the interpreter checks the program for errors.

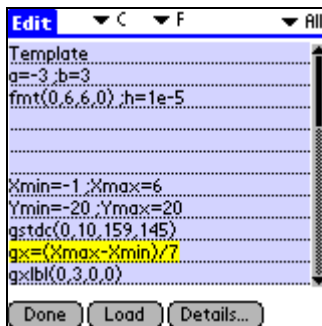


If it finds none, CplxCalPro returns to the text screen, where the program's variables and initial values are placed in the scratchpad, "Program loaded successfully!" is placed in the result line, and the program name is placed in the status line.



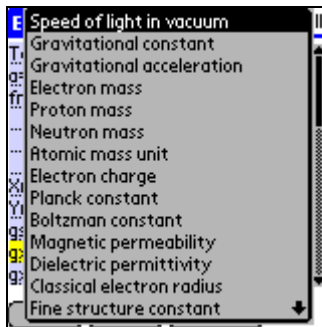
The screenshot above left shows the result of successfully loading the program "Car loan". Now you can change the variable assignments in the scratchpad. Pressing [RUN] then executes the program. With the values for "Car loan" in the screenshot on the left loaded into the program, pressing [RUN] brings up the screen above on the right.

Edit program



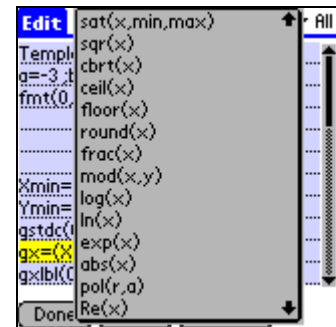
Selecting [Edit program] from the [Prog.] menu brings up the same screen as [Load program], but the edit checkbox at the bottom is selected; as in the example on the right. Tapping the name of a program allows you to edit that program with the same functionality you have when writing and editing memos.

Notice the "C" and "F" at the top of the screen. The first is a drop-down list of built-in constants; the second a drop-down list of built-in functions.



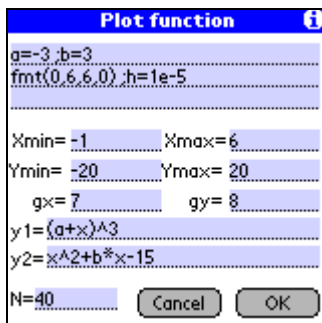
Selecting a constant puts its assignment wherever the cursor is in the program. Selecting the speed of light, for instance, will put the assignment $c=2.99792458E8$ at the cursor position. This way the constant is saved with the program. Constants not used do not allocate any memory space of the user program.

Selecting a function enters it at the cursor location, speeding up entry of that function.

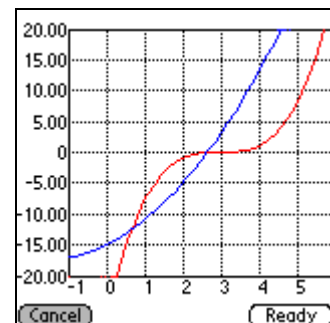


The next two items, [Plot function] and [Solve equation], use worksheets, and share information across worksheets. Worksheets are little forms in which you enter some parameters that will be used to create a program.

Plot function



Select [Plot function] from [Prog.] 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. We also set format values for the numbers generated (here, display format = float, width = 6, precision = 6; and no trailing zeros). The 'h' is used by [Solve equation] and will be discussed later.



The next six values set plot parameters. $Xmin$, $Ymin$, $Xmax$, and $Ymax$, set the plot boundaries, while gx and gy define how many grid lines are plotted along the x-axis and y-axis, respectively.

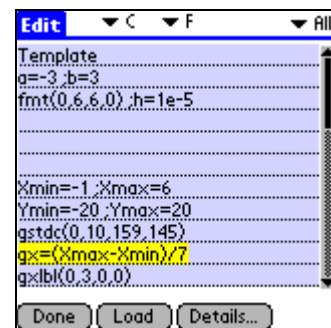
Using [Plot function], CplxCalPro will plot 2 functions simultaneously (however, using a program, CplxCalPro plots as many functions as you want). You enter each function on its own line at $y1=$ and $y2=$.

Finally, N is the number of points used to plot a graph.

Press [OK] and a template program is created. This program is then executed, resulting in the plot on the right. Notice that the two lines intersect twice within the plot's boundaries. In this example, we are interested specifically in $Xmin$ and $Xmax$. Tapping on the screen shows the x,y coordinates of the stylus tap within the graph at the bottom of the screen. Tapping where the curves intersect gives the x,y coordinates of their intersection.

Return to the main screen. Press the [EDIT] key. The program that was created using the parameters from [Plot function] comes up. You can change the name from "Template" to any name you choose by replacing "Template" in the first line to whatever name you like, then saving this modified program.

We encourage you to modify programs, parameters, etc. on CplxCalPro to get a better understanding of how to use these features.



Solve Equation

Solve equation	
Init equation:	
a=-3 ,b=3	
fFmt(0.6,6,0) ,h=1e-5	
Equation to solve:	
(a+x)^3 = x^2+b*x-15	
Solve for variable: x	
Guess: 0	TOL: 1e-8
Cancel	OK

Select [Solve equation] from [Prog.] menu. CplxCalPro 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] solves 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 *TOL*: textfield value is the tolerance of precision between the answer returned by CplxCalPro and the real value of the root. The smaller you make this value, the more accurate the answer but the longer CplxCalPro takes to generate a result. 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 $\text{abs}(x) < \text{TOL}$.

// (a+x)^3 = x^2+b*x-15	
a=-3 ,b=3	
fFmt(0.6,6,0) ,h=1e-5	
x=0.679116	
B2.00	Template 09/05/01
#	= , ; () CLS
pi	A B C D E F
e	A j arg Re Im VAR
7	8 9 / sin asin FUNC
4	5 6 * cos acos EDIT
1	2 3 - tan atan RUN
0	. E + sqr ans EXE

When you press [OK], CplxCalPro 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 CplxCalPro 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 CplxCalPro (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 $X_{min}=-5$, $X_{max}=20$, $Y_{min}=-40$, and $Y_{max}=200$, then press [OK]. CplxCalPro 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.

Copy text program

This menu item allows you to bring a text file of a CplxCalPro program or key assignment file that you've hotsync'd onto your PDA into CplxCalPro's program database. Once in the database, it is available to run. The text file will actually reside in your PDA's memory as a PRC file, and using "Copy text program" will copy it into CplxCalDB.PDB. Once in the database the text file can be deleted.

Delete text program

This menu item allows you to delete a text (PRC) file from your PDA's memory that CplxCalPro recognizes as a CplxCalPro program or key assignment file. You do this after copying the file into CplxCalProDB.PDB in order to keep your PDA's memory uncluttered.

Create text program

This menu item allows you to copy a CplxCalPro program, key assignment file, or graphic from CplxCalPro's program database into your PDA's available memory. The copy is a PRC file. From there it can be hotsync'd onto your personal computer and manipulated as a simple text file.

IF YOU WISH TO CREATE CplxCalPro PROGRAMS OR KEY ASSIGNMENT FILES ON YOUR PC, you will need a program that converts simple text files on your PC into PRC files. Several such programs, like MakeDocW, are available at www.handango.com. Make sure *compression mode* is *NOT* selected on your conversion program when making PRC files for CplxCalPro.

Help

Functions:

This will show a list of all the available built-in functions. Tapping a function will insert the function at the current cursor position.

Constants:

This will show a list of all the available built-in constants. Tapping a function will insert the constant at the current cursor position.

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.

Chapter 3

Putting It All Together

Wow! We've covered quite a bit of ground in just one chapter. As you can see, just off the main screen and with the default keyboard, CplxCalPro puts tremendous capabilities and computing power at your fingertips, 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 were introduced later. In this chapter, we're going to put it all together: we're going to walk through examples of how to change CplxCalPro's initialization file and keyboard; and we're going to walk through downloading a program from our online library, installing it, and running it.

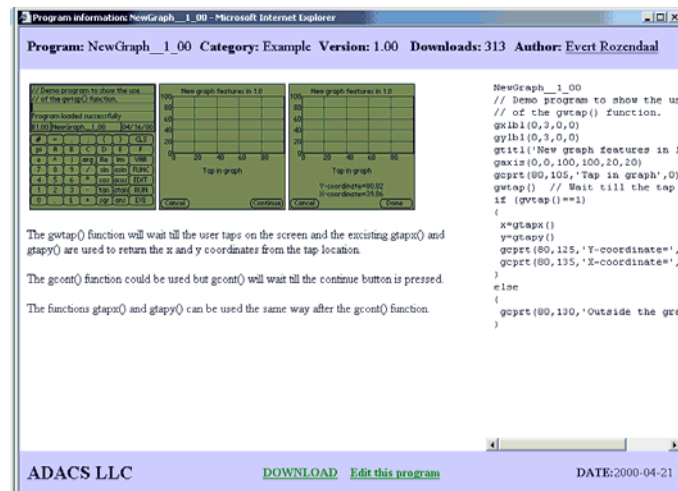
There is an example in the built-in database that you can copy to the clipboard and paste into a new text file. Then you can edit the text file to put the keys where you would like them. Where you put the keys or what you put in them, is up to you. These text files must be in the "keyboard" category. The program checks the category before processing the file. This is how the program determines if the file is a program or a keyboard assignment file.

Problem: You know that CplxCalPro is the calculator to meets your needs. You've gone to our online library of programs and found a program there that might meet your needs. How do you get the program from our library into your CplxCalPro's database of programs?

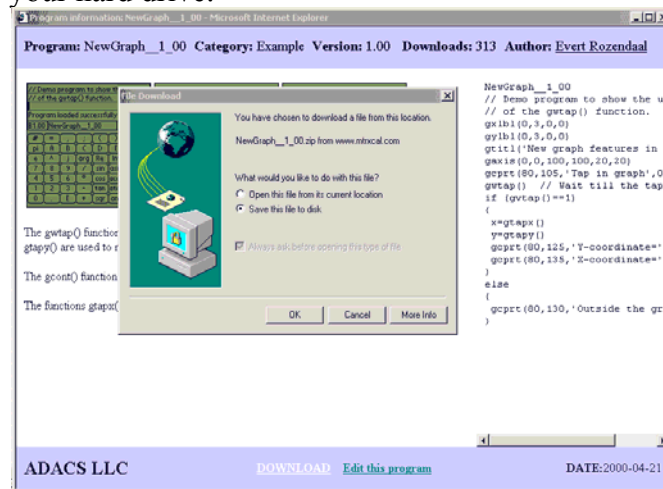
Answer: Let's say, for the sake of this example, that NewGraph__1_00 in the "Examples" category of the online library is what you. You can use the search feature to find this program as shown below. Select 'Program name' type 'new' and press 'Search' and all programs with 'new' in it will be shown.

#	Program name	Category	Version	Date	Author
1	loan_new(751)	Financial	1.00	2000-04-28	Haig Terzian
Loan calculator which also calculates the interest.					
2	NewGraph__1_00(315)	Example	1.00	2000-04-21	Evert Rozendaal
Shows the use of the new gwtap() function.					
3	NewText__1_00(172)	Example	1.00	2000-04-21	Evert Rozendaal
Example program with the new inpcat(str,v) function.					

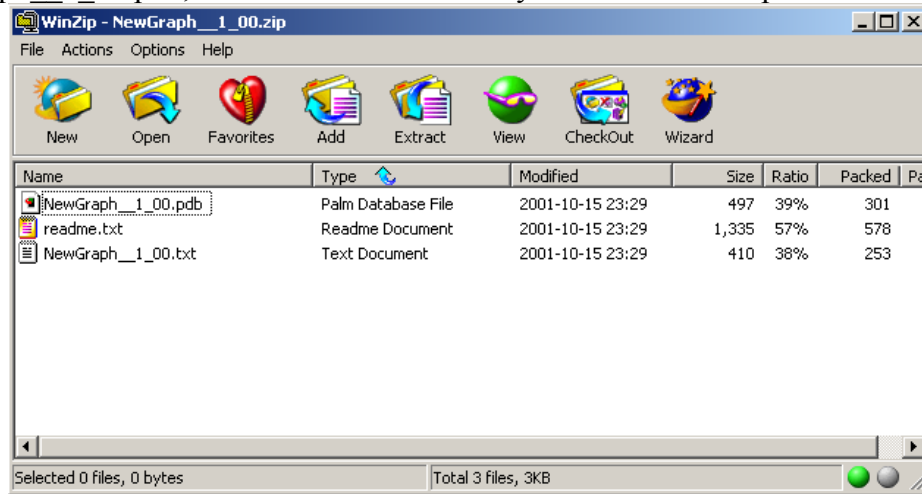
Click the 'NewGraph__1_00' hyperlink and the screen below will appear.



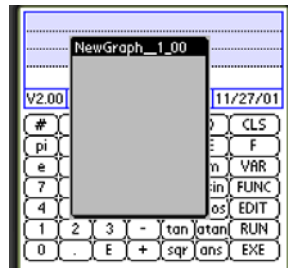
Each program information page has a “DOWNLOAD” hyperlink. You click it, and opt to save the zip file to your hard drive.



That done, you open the zip file using an unzip program. Download an unzip program from <http://www.winzip.com> if you don't have one already. You could extract the entire contents; but impatient to run NewGraph_1_00, you opt simply to double-click NewGraph_1_00.pdb, which loads the file into your PDA's desktop “Install Tool”.



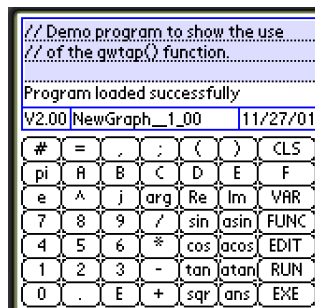
You hotsync your palm PDA, which loads NewGraph__1_00.pdb onto your device. Then you bring up CplxCalPro, bring up its menu items, select [Prog.], and [Copy text program]. CplxCalPro shows you the text file “NewGraph__1_00” in your PDA’s memory.



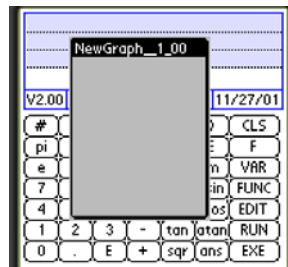
You tap it to bring it into CplxCalPro’s program database, then go back to the [Prog.] menu and choose [Load program].



There it is, at the top of the list, NewGraph__1_00. You tap on its name, read as CplxCalPro tells you, “Loading program, Please wait!!!”, then reports in the result line, “Program loaded successfully”.



It’s now ready for you to use or edit to meet your specific needs. As a final step, you choose [Delete text program] from the [Prog.] menu, which shows you NewGraph__1_00 in your PDA’s memory.



Tapping on its name brings up a confirmation alert asking if you’re sure you want to delete NewGraph__1_00. Tapping [Yes] deletes the PDB, but leaves the program in CplxCalPro’s program DB.

Chapter 4

Built-In Functions on CplxCalPro

CplxCalPro 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-j)	-45
conj(x)	Returns conjugate of x.	cplx	cplx	conj(3-j4)	conj(3+j4)
pol(r,a)	Returns rectangular value of radius r and angle a.	real	cplx	pol(3, 45)	2.12+j2.12
Im(x)	Returns imaginary of x.	cplx	real	Im(3-j4)	-4
Re(x)	Returns real of x.	cplx	real	Re(3-j4)	3

Basic:

OPERATOR or FUNCTION	REMARK	ARG	YIELD	EXAMPLE	
				Input	Output
%	If only percent is evaluated, returns decimal equivalent of percent. If percent is second part of arithmetic expression, takes percent of first part as second part, then evaluates expression.	real	real	50%	.5
				10 + 7.5%	10.75
abs(x)	Returns absolute value if x is real and returns magnitude if x is complex.	real	abs(x)	abs(-5)	5
		cplx	mag(x)	abs(sqrt(-1))	1
cbrt(x)	Returns cube root.	real	real	cbrt(-5)	-1.71
ceil(x)	Returns x if x is int, else returns next int > x	real	real	ceil(-2.5)	-2
exp(x)	Returns e^x . For complex $z=x+jy$, $\exp(z) = \exp(x) * (\cos(y) + j\sin(y))$.				
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+j3
		cplx	cplx	sqr(-5+j12)	2+j3
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.				

Trigonometric:

FUNCTION	REMARK
sin(a)	sine
cos(a)	cosine
tan(a)	tangent
asin(a)	arcsine
acos(a)	arccosine
atan(a)	arctangent
sinh(a)	Hyperbolic sine.
cosh(a)	Hyperbolic cosine.
tanh(a)	Hyperbolic tangent.
acosh(a)	Inverse hyperbolic cosine.
asinh(a)	Inverse hyperbolic sine.
atanh(a)	Inverse hyperbolic tangent.

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	$f''(x)$
<p>h is also known as Δx, and the definition of a derivative, $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$, 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 CplxCalPro, 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 CplxCalPro while calculating derivatives.</p>			

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:

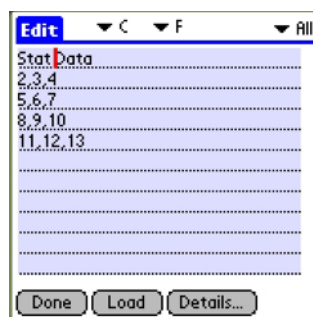
CplxCalPro can do powerful statistical analyses on multivariate data elements. The data elements can be entered into a text file in CplxCalPro's database for processing. The values in this text file can be converted using the `sdata('file')` function and stored in an array for processing. The array can also be filled using the `sadd(r,c,x)` or `scadd(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 CplxCalPro text file. Press [EDIT] from the text screen. If you have no program loaded, CplxCalPro takes you to its database of text files. If you do have a program loaded, CplxCalPro 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:



Press [Done]. You should see your file's name at the top of the database list.

Press [Return] to get back to the text screen. In the scratchpad, enter **stclr()** and press [EXE]. Now enter **sdata('Stat Data')**. Press [EXE]. This loads CplxCalPro's memory array with the values in the file and returns the number of data elements in the result line; in this case, 4. Enter **stdev(2)** and press [EXE]. The result line displays 3.35, the standard deviation of 3, 6, 9 and 12. Enter **ssum(3)** and press [EXE]. The result line displays 34, the sum of 4, 7, 10 and 13.

Finally, to calculate the linear regression, where column one holds the x-values and column two the y-values, enter **a=sqrc(3); b=sqrc(2); c=sqrc(1)** on the first line of the scratchpad, and **sqr(6)** on the second line, and press [EXE]. The result is $a*6^2+b*6+c$, or the calculated y-value when x is six.

To verify the answer you can also enter **a*6^2+b*6+c** on the second line instead of **sqr(6)**.

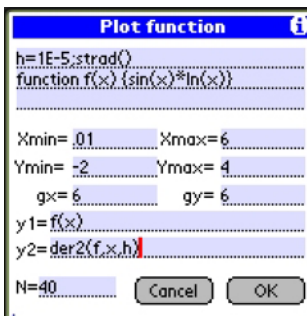
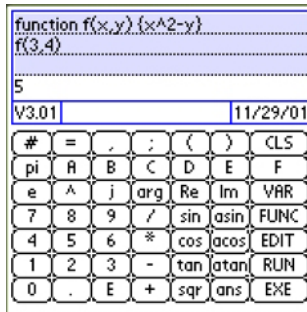
CplxCalPro statistical and probability functions.	
FUNCTION	REMARK
!	Factorial
fac(n)	Factorial
ftest(c1,n1,c2,c2)	Returns the result of an F-test. An F-test returns the one-tailed probability that the variances in column c1 and column c2 are not significantly different. Use this function to determine 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.
sadd(r,c,v)	Store value v at row r and column c. When v is a complex use scadd(r,c,v) instead!!
scadd(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 values in 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 quadratic regression. A=sqrc(3) B=sqrc(2) C=sqrc(1) See QuadReg.prc user program for and example.
sqr(x)	Returns the value for the quadratic regression $Y=A*X^2 + B*X + C$ See 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 $\text{sqr}(\text{var}())$ of column c.
Stdev(c)	Returns the sample standard deviation $\text{sqr}(\text{Var}())$ of column c.
svar(c)	Returns the population variance $(1/N * (A(n)-\text{mean})^2)$ of column c.
sVar(c)	Returns the sample variance $(1/(N-1) * (A(n)-\text{mean})^2)$ 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.

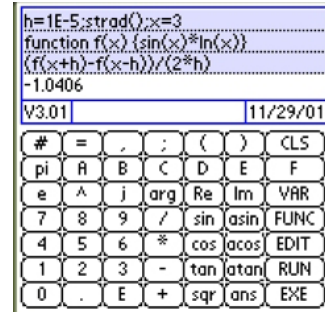
Chapter 5

User-Defined Functions

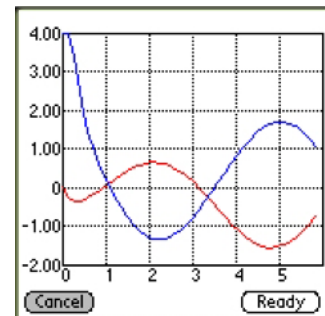
Starting in version 3.0 you can create your own functions for use on CplxCalPro.



The screenshot on the left shows how you can define your own function. User-defined functions can take up to four arguments. User-defined functions are first defined using the CplxCalPro function **function f(x){ }**, then invoked. The screenshot on the right shows how the derivative of a function can be calculated with the definition of a derivative of f at x , returning the same value for a given x as **der1(f,x,h)** does.



User-defined functions can also be used when plotting functions using the plot function worksheet. The function $f(x)$ is defined at the top, then invoked as $y1$. Notice that $y2$ uses **der2(f,x,h)**.



Please don't forget that the first three lines in a program, the scratchpad, are processed differently than the rest of the program. To review, when you press [EXE] only the scratchpad is evaluated. When you press [RUN] first the scratchpad is evaluated, then CplxCalPro runs the loaded program. So how does this relate to user-defined functions? It means that when you define a function in the scratchpad and use it in the program, you CANNOT change the function in the scratchpad, press [RUN], and get the correct result. 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].

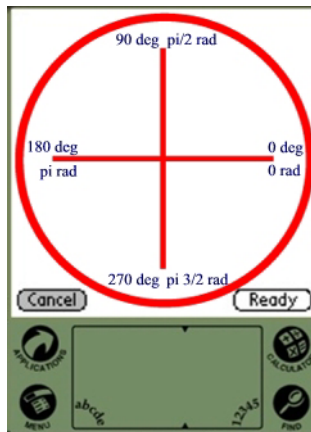
Chapter 6

Graphics on CplxCalPro

Graphics on CplxCalPro allows you to see how input has been transformed to output. The visual display of quantitative information¹ makes large amounts of data or complex data understandable. On CplxCalPro 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 CplxCalPro's graphics screen:

pixel (0, 0)

pixel (160, 0)



pixel (0,160)

pixel (160,160)

CplxCalPro Graphics Screen and The Palm Touchscreen Coordinate System

Superimposed in red is CplxCalPro'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. CplxCalPro 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.

CplxCalPro has many graphics functions to render objects on the graphics screen. These objects include lines, arcs, circles, rectangles, axes, and text. You can create interactive graphics on CplxCalPro with graphics functions like **gvtap()** and **gtapx()**, which, as their names suggest, process stylus taps on the touch screen. CplxCalPro uses a table of 16 colors. Colors are set using **gsetcol(idx,r,g,b)**. **idx** indicates the index in the table.

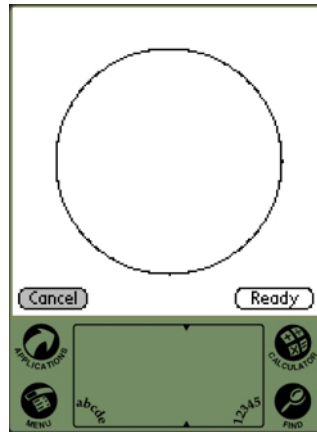
¹ Yes, this phrase is lifted from the title of the book, [The Visual Display of Quantitative Information](#), by Edward Tufte, whose 3-volume treatment on rendering information into visuals is highly recommended.

Default Colors on CplxCalPro				
On startup, CplxCalPro sets these colors for its color table. you can change them using gsetcol(idx,r,g,b)				
idx	color	r	b	g
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	black	0	0	0
11	unassigned (black)			
12	unassigned (black)			
13	unassigned (black)			
14	unassigned (black)			
15	unassigned (black)			

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 CplxCalPro'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, CplxCalPro will print "Program loaded successfully". Press [RUN] and watch as CplxCalPro draws this:



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.

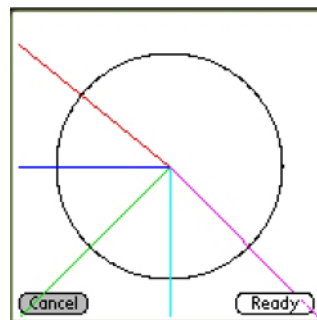
`greset()` Resets the graphics mapping to the default of 160 by 160 pixels.
`gcir(x,y,r)` Draw circle at x,y with radius r.

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 CplxCalPro returns you to the text screen, press [RUN]. On a color palm PDA, you should see:



Going counterclockwise from palm touchscreen coordinates (0,0), CplxCalPro draws lines from the center of the circle out. Notice that lines are drawn through the buttons

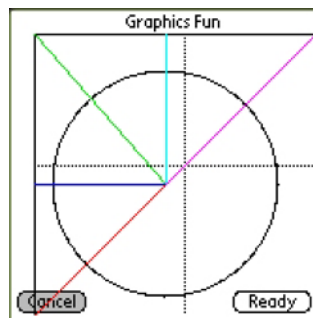
[Cancel] and [Ready]. This is because we have not set the device coordinates for the graph; making all pixels on the touchscreen available.

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(80,80,60)**, if you enter:

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

you get:



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)
```

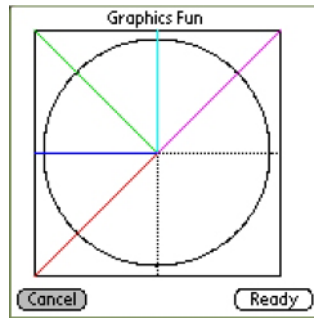
Then, to center the axes, we use $Xmax-Xmin$ and $Ymax-Ymin$:

```
gaxis(10,10,140,65,65)
```

Finally, we change the coordinates of our graphics objects:

```
gcir(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)
```

And this is what we get:



Other functions used in graphics on CplxCalPro include:

<code>garc(x,y,r,a1,a2)</code>	Draws an arc angle, beginning at a start angle $a1$, and ending at angle $a2$. The angle can be in radians or degrees. Use <code>strad()</code> or <code>stdeg()</code> to set for radians or degrees.
<code>gaxis(x1,y1,x2,y2,gx,gy)</code>	Set axis with minimum values $x1$ and $y1$ and maximum value $x2$ and $y2$ with grid lines at gx and gy .
<code>gclrs()</code>	Clear graphics screen. The following functions only appear in the dropdown function list on the main screen.
<code>gcont()</code>	Wait until continue button is pressed.
<code>gcprt(x,y,'StrV',v)</code>	Draw text 'StrV' and value v at centered at x,y position.
<code>gfcir(x,y,r)</code>	Fills a circle at x,y with radius r .
<code>ghlin(y)</code>	Draw horizontal line at y position.
<code>grect(x1,y1,x2,y2)</code>	Draw rectangle.
<code>grprt(x,y,'StrV',v)</code>	Draw text 'StrV' and value v at right x,y position.
<code>gvlin(x)</code>	Draw vertical line at x position.
<code>gvprt(x,y, 'StrV',v)</code>	Print text on graphical screen vertical.
<code>gxlbl(t,w,p,l)</code>	Set the x-axis labeling. t,w,p are the type, width and precision of the numbers displayed on the axis.
<code>gylbl(t,w,p,l)</code>	Same as <code>gxlbl</code> for the y-axis.
<code>gselcol(idx)</code>	Select a color from the color table. V.2
<code>gsetcol(idx,r,g,b)</code>	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. V.2

For a complete list of graphics functions, refer to [Appendix D](#).

Because the power and usefulness of graphics on CplxCalPro 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 CplxCalPro. 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 7

Programming CplxCalPro

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 programs, you should look at: <http://www.adacs.com/CplxCalPro/downloads.htm>. We encourage this for two reasons: first, our online library of programs is teeming with examples of how other programmers solved problems specific to the strengths and limits of CplxCalPro. 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' programs, 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 CplxCalPro. 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 CplxCalPro 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 program? ^{*} For the purposes of this manual, let's define programming as a set of instructions run on CplxCalPro, 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 CplxCalPro to perform (CplxCalPro

^{*} We've scoured our references in search of the concise, comprehensive, yet witty definition to enlighten those CplxCalPro users who, though like all CplxCalPro 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.

programs can hold as many as 1200 steps). 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.[▽]

CplxCalPro's programming language is simple yet powerful. It lacks the GOTO statement and so requires programs 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.

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 CplxCalPro programs that use the concepts illustrated here are in the programming examples section below.

```

10 A = 5
20 B = 10
30 A = B
40 C = sin(A+B)
50 In C
60 A = 5
70 B = 10
80 A = B
90 C = sin(A+B)
100 In C
110 A = 5
120 B = 10
130 A = B
140 C = sin(A+B)
150 In C
  
```

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

```

10 A = 5
20 B = 3
30 C = A * B
40 D = A ^ B
50 IF D = C THEN
60 (
70 C = D - (A * B)
80 )
90 A = 5
100 B = 3
110 C = A * B
120 D = A ^ B
130 E = A * B
140 F = A ^ B
150 G = A * B
  
```

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

```

10 A = 5
20 B = 3
30 C = A * B
40 E = 0
50 WHILE E < 10
60 (
70 C = C * B
80 A = A * B
90 E = E + 1
100 )
110 F = sin(C)^A
120 A = 5
130 B = 3
140 C = A * B
150 G = 1
  
```

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.

[▽] 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.

CplxCalPro's Programming Commands:

Let's look briefly at CplxCalPro's programming commands and operators before we tackle some sample programs:

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. 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.
else\n{\n}\n	
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.
sleep(n)	wait until n seconds have passed, then continue program execution.
wait(n)	wait until user presses a key before continuing program execution.
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 CplxCalPro, they can be relational (meaning that CplxCalPro tests one data value against another to decide action) or interactive (meaning that CplxCalPro waits for input from the user).

Here are CplxCalPro'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

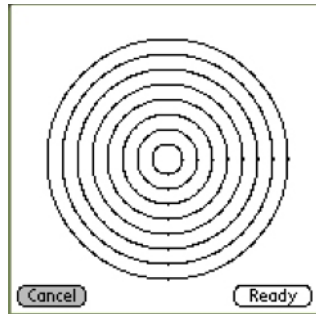
* OK; the typography of these operators is decidedly UNIX, whence comes C, C++, Java, and Perl, all of which use the same typography for their relational operators. Can you tell which language CplxCalPro is written in? (Hint: those of you who answer correctly earn an A++.)

gvtap()	button is pressed. Valid tap. Returns 1 if user tapped in graph. Returns false if outside graph.
gwtap()	Wait until user taps on the screen. The functions gtapx() and gtapy() can be used to return the x and y coordinates of the last drawn graph.

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

Programming Examples:

More Graphics Fun! This time we want to draw concentric circles, evenly spaced, on the graphics screen, like in this screenshot:



I can think of several ways to do this. One is to go back to the Graphics Fun file we wrote in the chapter on graphics, and write the **gcir(x,y,r)** function 8 times, keeping **x** and **y** constant while increasing the value of **r** by 8 in each consecutive **gcir(x,y,r)**, like this:

```
greset()
gcir(80,80,8)
gcir(80,80,16)
gcir(80,80,24)
gcir(80,80,32)
gcir(80,80,40)
gcir(80,80,48)
gcir(80,80,56)
gcir(80,80,64)
```

Remember, a CplxCalPro program starts with its name on line 1 and scratchpad variables (that is, variables whose values can be changed after the program has successfully loaded and before a program run) on lines 2,3, and 4. The code, therefore, starts on line 5. If you

wish to enter this program and run it, you must at least give the program a name on line 1, and start writing the code from line 5.

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(80,80,8)**, varying only in their radius values.

Such repetition of code begs casting the **gcir(x,y,r)** function in repetition flow. Few (actually, none) would argue that it isn't more efficient to write a small program in which we write the **gcir(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 CplxCalPro, we use:

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

Between the curly brackets {} we'll put the kernel of this program, **gcir(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(x,y,r)**, so that it becomes **gcir(80,80,r)**. The **condition** to keep program flow in the loop, executing **gcir(x,y,r)** over and over, will be **r <= 64**.

Now we have a while loop that looks like this:

```
while(r<=64)
{
  gcir(80,80,r)
}
```

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=0**. And what's the other thing we're missing? Reading the program aloud might help us find out. Reading aloud, we might say something like, "We 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..." 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, program flow will never leave the while loop as CplxCalPro endlessly draws circles of radius zero. **A common programming 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 program that will meet our specs:

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

```
}
```

Let's go ahead and clear the graphics screen each time we run it. Put **greset()** above **r=0**. Reading the program 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.

greset()	greset()
r=0	gcir(80,80,8)
while(r<=64)	gcir(80,80,16)
{	gcir(80,80,24)
gcir(80,80,r)	gcir(80,80,32)
	gcir(80,80,40)
r=r+8	gcir(80,80,48)
}	gcir(80,80,56)
	gcir(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 $4n*r$, $0 \leq r \leq 64$, instead of what we have now, $8n*r$, $0 \leq r \leq 64$. Had we drawn our concentric circles by the program 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 program on the left, this change to the program 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 program 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()	greset()
r=0	x=0
while(r<=64)	while(x<=8)
{	{
gcir(80,80,r)	r=x*8
r=r+8	gcir(80,80,r)
}	x=x+1
	}

Now we have three programs, all yielding the same result. Programs or parts of programs 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 program, I admit; but we're using this program 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.

DrawBullseye

```
//initialize
greset()
radius=0

//draw concentric circles
//making each circle's radius
//8 pixels > than last
while(radius<=64)
{
    gcir(80,80,radius)

    radius=radius+8
}
```

As written, our program requires editing if we want to change any of its parameters; and once running, accepts no input from the user. CplxCalPro gives the user greater flexibility for changing program variables, and allows user interaction during program run. The next two changes to our program show how.

Let's start with program interaction: once the user runs the program, 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).

DrawBullseye
deltar = 8

```
//initialize
greset()
radius=0

//wait for user to press continue button
gcont()

//draw concentric circles
//making each circle's radius
//8 pixels > than last
while(radius<=64)
{
    gcir(80,80,radius)

    radius=radius+deltar
}
```

After this program 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 programs have parameters. Within their parameters, their writers try to ensure that they work properly and do not give bad output. After you write a program on CplxCalPro, you should test it for soundness. This means finding values for your program's variables that will cause the program to fail, or give bad output; and once you find them, either having your program catch the errors, or document what causes errors as your program's parameters.

A valid example for a CplxCalPro program would be to determine if any variable causes a divisor to become zero while your program runs. Because division by zero is undefined, if it happens in your program, your program will terminate abnormally and CplxCalPro will display:



How to prevent this? If the value of the divisor variable is set by the user, you can document your program with a comment like this:

// if varFoobar = 0, program will terminate abnormally!

If the value of the divisor variable changes during the program run, you might be able to catch the error in such a way that the program 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 program terminate abnormally — if **divisor** = 0. In this example, setting **test = 0** meets program specs.

Appendix A

Technical specifications

This NON-RPN calculator has the following technical specifications:

- IEEE-754 64-bit Double Precision, a floating point format ranging from -2.2250738585072014E-308 to 1.7976931348623157E+308.
- 190 built in functions.
- 30 Graphical functions.
- 30 built in constants.
- Drop down lists for constants and functions.
- Display formats: Float, Engineering, Symbol, Hexadecimal, Binary, Octal, Polar, Date and Sexagesimal.
- Angular units: Radians and Degrees (Grad was not added since we did not receive any requests for it).

Due to the complex memory structures used, the next three values are depending on the program used but are within 1% tolerance.

- Maximum number of strings: 150
- Maximum number of function steps: 1200
- Maximum number of characters in a program: 8096
- Built in, memopad like, database.
- Built in help.
- Financial functions.
- Statistical functions.
- Fully supported clipboard.
- Variable list and ability to change variables in list.
- Graphical output screen.
- Syntax checking on programs.
- Total memory allocation screen.
- User-configurable main screen keyboard
- Most functions, including trigonometry functions, accept complex arguments.

Appendix B

Data Formats

Float: The native data format of CplxCalPro.

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]. CplxCalPro will display 511E6. The exponent, in this case 6, will always be a multiple of three. The symbol format will show an SI postfix instead of E6.

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
Symbol:	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 C

Display format

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

CplxCalPro 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 CplxCalPro 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, CplxCalPro 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 CplxCalPro 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 D

Functions, Operators, and Commands

'StrV' -- This can be either a string or a variable.

// -- Anything behind the // is ignored by the program and can be used to add remarks in the program.

Base conversion:

OPERATOR or FUNCTION	REMARK	ARG	YIELDS	EXAMPLE	
				Input	Output
#	Enter in hexadecimal format	real	real	#A9C3	43,459
&	Enter value in binary format	real	real	&1010	10

Basic:

OPERATOR or FUNCTION	REMARK	ARG	YIELDS	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%	0.5
				10 + 7.5%	10.75
abs(x)	Returns absolute value if x is real and returns magnitude if x is complex.	real	abs(x)	abs(-5)	5
		cplx	mag(x)	abs(sqrt(-1))	1
cbrt(x)	Returns cube root.	real	real	cbrt(-5)	-1.71
ceil(x)	Returns x if x is int, else returns next int > x	real	real	ceil(-2.5)	-2
exp(x)	Returns e^x. For complex z=x+jy, exp(z) = exp(x)*(cos(y)+j*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+j3
		cplx	cplx	sqr(-5+j12)	2+j3
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 y.

Calculus:

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

Color:

FUNCTION	REMARK
gselcol(idx)	Select a color from the color table. V.2
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. V.2

Complex:

FUNCTION	REMARK	ARG	YIELDS	EXAMPLE	
				Input	Output
arg(x)	Returns the angle of x.	cplx	real	arg(1-j)	-45
conj(x)	Returns conjugate of x.	cplx	cplx	conj(3-j4)	conj(3+j4)
pol(r,a)	Returns rectangular value of radius r and angle a.	real	cplx	pol(3, 45)	2.12+j2.12
Im(x)	Returns imaginary of x.	cplx	real	Im(3-j4)	-4
Re(x)	Returns real of x.	cplx	real	Re(3-j4)	3

Conversion:

OPERATOR or FUNCTION	REMARK	EXAMPLE	
		Input	Output
cel(t)	Convert t from Fahrenheit to Celsius.	cel(75)	23.889
deg(a)	Convert a from radians to degrees.	deg(pi)	180
dms(deg,mm,ss)	Converts degrees, minutes and seconds to degrees.	dms(10,30,0)	10.5
fah(t)	Convert t from Celsius to Fahrenheit.	fah(23.889)	75
met(Yr,Ft,Inch, p)	Convert Yards, Feet, Inches and part of inches to meters.	met(1,2,3,1/16)	1.6018
rad(a)	Convert a from degrees to radians.	Rad(180)	3.1416

Date:

FUNCTION	REMARK
----------	--------

date(mm,dd,yy,'StrV')	Returns seconds between 1904 and selected date. When the string 'StrV' is used it will popup a date selection window. When 'StrV' equals zero the month mm, day dd and year yy are used.
days(sec)	Returns the number of days.
time()	Returns the current time in seconds since 1904.
weeks(sec)	Returns the number of weeks.

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. 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 -\$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.

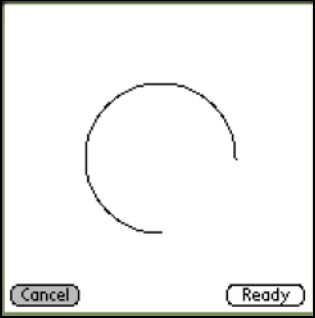

Flow Control:


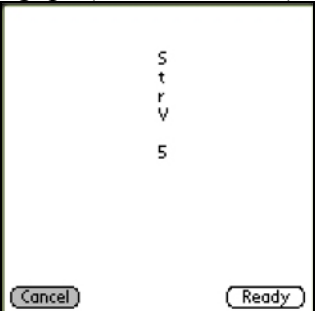
COMMAND	REMARK
else\n{\n}\n	Executes the commands between brackets when the condition for the ifstatement is false.
exit(n)	Terminate program.n=0 normal termination.n=1 termination due to error.
if(cond)\n{\n}\n	If (condition) is true, execute the commands between brackets.
init()	Returns one only the first time after executing a program.This is mainly used for initializing variables.
sleep(n)	Wait until n seconds are passed.
wait(n)	Wait until user presses a key.
while(cond)\n{\n}\n	While (condition) is true, execute the commands between brackets.

Format:

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

Graphics:

OPERATOR or FUNCTION	REMARK	EXAMPLE
		NOTE: in these examples angles entered as radians.
garc(x,y,r,a1,a2)	<p>Draws an arc angle. Use strad() or stdeg() to set radians or degrees.</p> <p>x, y: x, y-coordinates of vertex. r: radius of arc a1: start angle a2: end angle</p>	<p>garc(80,80,40,0,0.5*pi)</p> 
gaxis(x1,y1,x2,y2,gx,gy)	<p>x1,y1: min values of axis x2,y2: max values of axis gx,gy: values to draw gridlines</p>	
gcir(x,y,r)	Draw circle at x,y with radius r.	
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)	<p>x, y: center values of string, value StrV, v: string, value to display</p>	<p>gcprt(80,80,'value = ',2.7183)</p> 
gfcir(x,y,r)	Fills a circle at x,y with radius r.	
ghlin(y)	Draw horizontal line at y position.	
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.	
gline4(x1,y1,x2,y2)	Draw line from x1,y1 to x2,y2.	
gline5(x1,y1,x2,y2)	Draw line from x1,y1 to x2,y2.	
glprt('StrV',v)	Draw text 'StrV' and v add next line.	
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).	
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.	
gppt(x,y,'StrV',v)	Draw text 'StrV' and value v at x,y position.	
grect(x1,y1,x2,y2)	Draw rectangle.	
greset()	Resets the graphics mapping to the default of 160 by 160 pixels.	
grprt(x,y,'StrV',v)	Draw text 'StrV' and value v at right x,y position.	
gstdc(x1,y1,x2,y2)	Set device coordinates for graph all values must be between 1 and 160.	
gtadd('StrV',v)	Add 'StrV' and v to last drawn text.	<p>gcprt(30,20,'value = ',2.7183) gtadd(' pi = ',3.14159)</p> 
gtapx()	Returns the x value of the last time you tapped on a graph. Use this function after the gcont() or the gwtap() function.	
gtapy()	Same as gtapx() but returns the y value.	
gtitl('StrV',v)	Draw title 'StrV' and v to graph.	
gvlin(x)	Draw vertical line at x position.	
gvprt(x,y,'StrV',v)	<p>Print text on graphical screen vertical.</p> <p>x,y: uppermost pixel of text</p>	<p>gvprt(80,20,'StrV',5)</p> 
gvtap()	Valid tap. Returns 1 if user tapped in graph. Returns false if outside graph.	
gwtap()	Wait until user taps on the screen. The functions gtapx() and gtapy() can be used to return the x and y coordinates of the last drawn graph.	

gxlbl(t,w,p,l)	Set the x-axis labeling. t,w,p are the type, width and precision of the numbers displayed on the axis. When the last argument equals one a logarithm scale will be drawn.
gylbl(t,w,p,l)	Same as gxlbl for the y-axis.

Interactive:

COMMAND	REMARK
clra()	Clear all buffers and program.
clrinp()	Clears the scratchpad. (Top three lines on main screen)
inp('StrV')	Put up dialog box with 'StrV' message. A value can be entered in the dialog box which value is returned.
inpcat('StrV','StrV')	Put the two 'StrV'-string/var. at the end of the text in the scratchpad.
inpv('StrV',v)	Put up dialog box with the first 'StrV' message. The second argument v is the default value. A value can be entered in the dialog box which value is returned.
iskey('StrV')	Returns 1 when button 'StrV' was pressed.
key(n,'StrV')	Set text in button at location n.
keybrd('rec')	Read keyboard settings out of record 'rec'
mess('title','mess')	Pop up a message box with title and message.
mode1('StrV','StrV')	Overwrites the version string on the status line with the strings str and the variable v.
mode2('StrV','StrV')	Overwrites the date string on the status line with the strings str and the variable v.
result('StrV','StrV')	Writes the string str and the variable to the result line on the main screen, line four.

Logical:

FUNCTION	REMARK	EXAMPLE	
		Input	Output
and(h,h)	Bitwise and.	and(4,6)	4
not(x)	Returns 0 if x!=0 else 1 .		
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)	2

Relational:

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

Special:

FUNCTION	REMARK
root(f,x)	Return the value of x at which the expression of function f(x) is equal to zero. This is a special function that CAN NOT be used within an other function. See example program Root function
beta(n,m)	Beta function
betai(a,b,x)	Returns the incomplete beta function.
erf(x)	Error function of x.
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. See FFT example.
gamma(x)	Gamma function
gcd(x,y)	Greatest common divider.
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.
linint(x1,y1,x2,y2,x)	Linear interpolation. Calculates $(y2-y1)/(x2-x1)*(x-x1)+y1$
lgam(x)	Ln of gamma function
perc(a,b)	Percentage change. Calculates $(b-a)/a*100$
pval(x,p1,p2,p3,p4,p5)	Returns poly values $(p1*x+p2*x^2+p3*x^3+p4*x^4+p5*x^5)$.
proot(x,p1,p2,p3,p4,p5)	Returns the poly root for function $f(x)= p1*x+p2*x^2+p3*x^3+p4*x^4+p5*x^5$ X is the guess value. This is where proot() start searching for a root.
res(r,tol)	Finds the closed resistor value r with tolerance tol.
sat(x,min,max)	Returns x if $min < x < max$ else max or min.
sinc(a)	$\sin(\pi*x)/(\pi*x)$ returns 1 if $x=0$

Probability & Statistics:

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

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.
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.
sqr(c)	Returns the coefficients for the quadratic regression. A=sqr(3) B=sqr(2) B=sqr(1) See QuadReg.prc user program for an example.
sqr(x)	Returns the value for the quadratic regression $Y=A*X^2 + B*X + C$ See 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 $\sqrt{\text{var}()}$ of column c.
Stdev(c)	Returns the sample standard deviation $\sqrt{\text{Var}()}$ of column c.
svar(c)	Returns the population variance $(1/N * (A(n)-\text{mean})^2)$ of column c.
sVar(c)	Returns the sample variance $(1/(N-1) * (A(n)-\text{mean})^2)$ of column c.
sxy(c)	Returns $A(1,n) * A(c,n)$.
syint(c)	Returns y-intersect.
ttest(c1,n1,c2,n2,tail,ty pe)	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.

String:

FUNCTION	REMARK
str(s1,s2,s3,s4,s5,s6)	Returns the concatenated strings. See String.prc example.
strcat('str','str')	Concatenates string s2 to string s1.

<code>strcpy('str','str')</code>	Copies string s2 into string s1.
----------------------------------	----------------------------------

Trigonometric:

FUNCTION	REMARK
acos(a)	Arccosine
acosh(a)	Inverse hyperbolic cosine.
asin(a)	Arcsine
asinh(a)	Inverse hyperbolic sine.
atan(a)	Arctangent
atan2(y,x)	Returns the four quadrant arctangent of the real parts of the elements of X and Y.
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 E

Constants

Constant	Name	Value on CplxCalPro	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	$\text{m}^3 \text{kg}^{-1} \text{s}^{-2}$
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	$\text{W m}^{-2} \text{K}^{-4}$
Na	Avogadro's constant	6.0221367E23	mol^{-1}
Vm	Ideal gas volume at STP	2.24141E-2	$\text{m}^3 \text{mol}^{-1}$
R	Universal gas constant	8.31451	$\text{J mol}^{-1} \text{K}^{-1}$
F	Faraday constant	9.6485309E4	C mol^{-1}
RH	Quantum Hall resistance	2.58128056E4	Ohm

Appendix F

Sample Programs

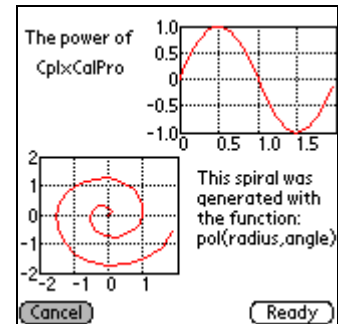
Graph demo:

```

Graph demo
min=0
max=2
step=0.05
gstdc(65,10,159,70)
gxlbl(0,1,1,0)
gylbl(0,1,1,0)
gaxis(min,-
1,max,1,0.5,0.5)
x=min
gmove(0,0)
while(x<=max)
{
  y=sin(x/max*360)
  glin(x,y)
  x=x+step
}

gstdc(2,75,80,145)
gxlbl(0,1,0,0)
gylbl(0,1,0,0)
gaxis(-max,-max,max,max,1,1)
x=min
gmove(0,0)
while(x<=max)
{
  angle=x/max*720
  radius=x
  glin(pol(radius,angle),0)
  x=x+step
}
gppt(10,10,'The power of ',0)
gppt(15,25,'CplxCalPro',0)
gppt(90,80,'This spiral
was',0)
glprt('generated with',0)
glprt('the function:',0)
glprt('pol(radius,angle)',0)

```



FFT example program:

Let's take a look at the "FFT calculate" program. This program is included in the database of programs and is located in the electronics category. The functions shown in green either set a color or draw using a color.

```

FFT calculate
Phase=60

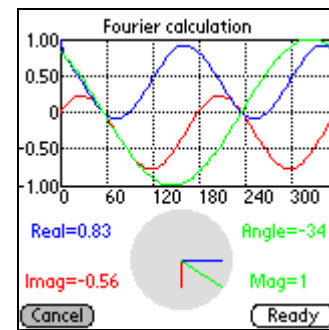
stdeg()
fmt(0,6,6,0)
gtitl('Fourier calculation',0)
Xmin=0 ;Xmax=360
Ymin=-1 ;Ymax=1
gstdc(0,15,159,100)
gx=(Xmax-Xmin)/6
gxlbl(0,3,0,0)
gy=(Ymax-Ymin)/4
gylbl(0,3,2,0)
gaxis(Xmin,Ymin,Xmax,Ymax,gx,gy)
N=100
Step=(Xmax-Xmin)/N
x=Xmin

greset()
x=80
y=125
mul=25
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(30,105,'Real=',Real)

// select color of line 1

```




```
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==0)
  {
    gmove(x,y1)
    gmove2(x,y2)
    gmove3(x,y3)
  }
  else
  {
    glin(x,y1*y3)
    glin2(x,y2*y3)
    glin3(x,y3)
  }
  x=x+Step
}
```

```
gselcol(1)
grprt(30,130,'Imag=',Imag)

v=Real+j*Imag

// Select color of line 3
gselcol(3)
grprt(140,105,'Angle=',arg(v))
grprt(140,130,'Mag=',abs(v))
```

For more information about fourier transforms please visit our website.

FFT built-in functions:

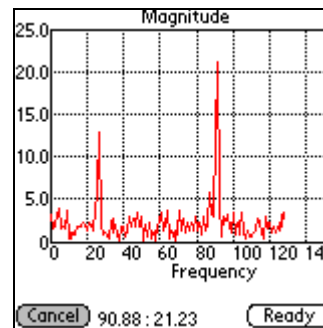
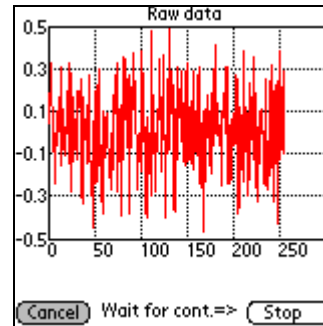
```

FFT function
datafile='fft_data'

N=sdata(datafile)
srplot(1,1,N,0)
gtitl('Raw data',0)
gprrt(44,146,'Wait for cont.=>',0)
fft(1,N)

i=1
while(i<=N)
{
    sadd(i,3,abs(scget(i,1)))
    i=i+1
}
gcont()
srplot(3,1,N/2,0)
gtitl('Magnitude',0)
gcprtr(100,125,'Frequency',0)

```



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:

```

Quad_regression
// Quadratic_regression
x1=0 ;x2=100 ;sx=25 ;xStep=10
y1=-25 ;y2=100 ;sy=25

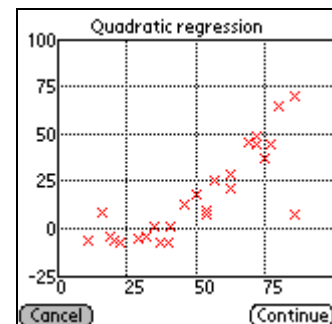
n=sdata('Data02')

gxlbl(0,2,0,0)
gylbl(0,2,0,0)
gtitl('Quadratic regression',0)
gstddc(5,15,158,145)
gaxis(x1,y1,x2,y2,sx,sy)
x=sget(1,1)
y=sget(1,2)
gmove(x,y)

// next three lines are not needed when
// sqrv(x) function is used.
a=sqrc(3)
b=sqrc(2)
c=sqrc(1)

// Show the fitting
first=1
x=x1+xStep
while(x<x2)
{
    y=sqrv(x) // same as y=a*x^2+b*x+c
    if (first==1)
    {
        gmove2(x,y)
        first=0
    }
}

```



```

// Plot data points
i=2
while(i<=n)
{
  x=sget(i,1)
  y=sget(i,2)
  gpnt(x,y,2)
  i=i+1
}

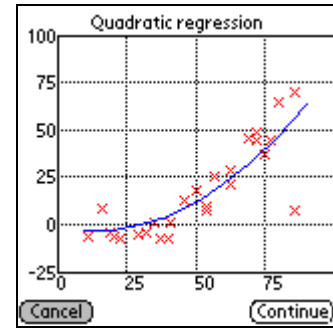
// Wait for user to press
// continue.
gcont()

}
else
{
  glin2(x,y)
  x=x+xStep
}

gcont()
gclrs()

gcprt(80,30,'Quadratic regression',0)
gpnt(5,50,"a)
gtadd("**X^2 + ",b)
gtadd("**X + ",c)

```



Chi-square test:

The chi-square test compares a sample of data to a statistical hypothesis (probability distribution). The chi-square is a “goodness of fit” test that is applicable to nominal scale data (discrete functions). The data are tallies of observations in categories.

To perform the chi-square test, observed values from an experiment are compared to the expected values based on the probability model. The following statistic is calculated and compared to a table of chi-square critical values:

$$\chi^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}}$$

chi-data	Observed
9.125	
3.40	
3.42	
1.12	

Fig. 1

stclr()	
n=sdata('chi-data')	
schi(1,2)	
0.28361	
B2.00 Chi-square test	09/09/01

Fig. 2

The expected values in Fig. 1 are in the first column and the observed values in the second column.

Fig. 2 shows how the Chi-square test can be calculated using only three functions.

Opamp:

```

Opamp
R1=1K ;R2=2.7K
R3=499 ;R4=860
U1=3.36 ;U2=1.2
gline(20,20,120,20)
fmt(2,1,1,0)
gprrt(8,15,'U1',0)
gprrt(25,15,'R1=',R1)
gprrt(75,15,'R2=',R2)
gline(70,20,70,40)
gline(70,40,80,40)

// minus sign
gline(82,40,84,40)
gline(80,35,80,65)
gline(20,60,80,60)

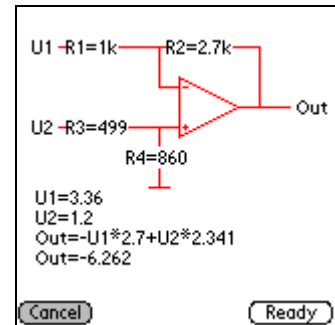
// plus sign
gline(82,60,84,60)
gline(83,59,83,61)

gprrt(25,55,'R3=',R3)
gprrt(8,55,'U2',0)
gline(80,35,110,50)
gline(80,65,110,50)
gline(110,50,135,50)
gprrt(140,45,'Out',0)
gline(120,20,120,50)
gline(70,60,70,90)
gcprtr(70,70,'R4=',R4)
gline(65,90,75,90)

fmt(1,4,3,0)
gprrt(10,90,'U1=',U1)
gprrt(10,100,'U2=',U2)

// Start calculations
b=(R4/(R4+R3))*((R2+R1)/R1)
gprrt(10,110,'Out=-U1*',R2/R1)
gtadd('+U2*',b)
gprrt(10,120,'Out=',-
U1*R2/R1+U2*b)

```

**Root function:**

```

Root function
function f(x) {x^3-10*x+2}
x1=-4 ;x2=4 ;y1=-20 ;y2=20 ;N=50

gtitl('Root function.',0)
s=(x2-x1)/N
gstdc(1,20,159,110)
gxlbl(0,2,0,0)
gylbl(0,2,0,0)
gaxis(x1,y1,x2,y2,2,5)
x=x1
gmove2(x,f(x))
while(x<=x2)
{
    glin2(x,f(x))
    x=x+s
}

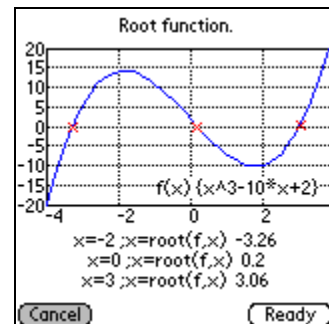
gprrt(70,84,'f(x) {x^3-10*x+2}',0)

x=-2 ;x=root(f,x) ;gpnt(x,f(x),2)
gcprtr(80,110,'x=-2 ;x=root(f,x) ',x)

x=0 ;x=root(f,x) ;gpnt(x,f(x),2)
gcprtr(80,120,'x=0 ;x=root(f,x) ',x)

x=3 ;x=root(f,x) ;gpnt(x,f(x),2)
gcprtr(80,130,'x=3 ;x=root(f,x) ',x)

```



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

Appendix G

The Palm® OS (POS) Emulator

This appendix has been added because we understand the need for fast development of CplxCalPro programs. Let's face it: entering a maximum of 1200 program steps with Graffiti can be daunting; and touch-typists, which most who deal a lot with computational devices by necessity become, are more efficient typing in data than stroking it in by stylus. The Palm OS Emulator makes program development easier if only because it allows you to type in your program. Yes, other keyboard options exist, given the rise in popularity of keyboards for PDAs*; but we feel that their use is rather straightforward and so doesn't need to be explained here.

This is the Palm OS Emulator:



To use it, you need to download both the emulator and at least one ROM image for it from the Palm website <http://www.palmos.com/dev/tools/emulator>. As is explained at the site, you can download the emulator immediately; You can use the rom transfer program to transfer the ROM image from your palm to your PC or MAC. To download other ROM images for the emulator, you must [Join the Palm OS Developer Program](#). Membership that gives you access to the ROM images is free; however, until you enter the password the Palm Developer Program emails to you when entering the Resource Pavilion (your email address will be your user name), the resource pavilion web page will not have hyperlinks to the ROM images on it. It does take about a day for your request for membership to be processed.

* In fact, we use Palm keyboards.

Appendix H

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 I

Useful Web Links

URL

<http://www.adacs.com>

<http://physics.nist.gov/cuu/index.html>

<http://mathworld.wolfram.com/>

<http://ieee.org/>

<http://www.acm.org/>

what you'll find there us!

The NIST Reference on Constants, Units, and Uncertainty, including in-depth information on the metric system

Eric Weisstein's World of Mathematics

Possibly the best mathematics reference work on the World Wide Web

The IEEE, for electrical engineers and those of you curious about the standards that hardware, software (including CplxCalPro!), and firmware adheres to.

ACM Association for Computing Machinery, the world's first educational and scientific computing society.

Afterword

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