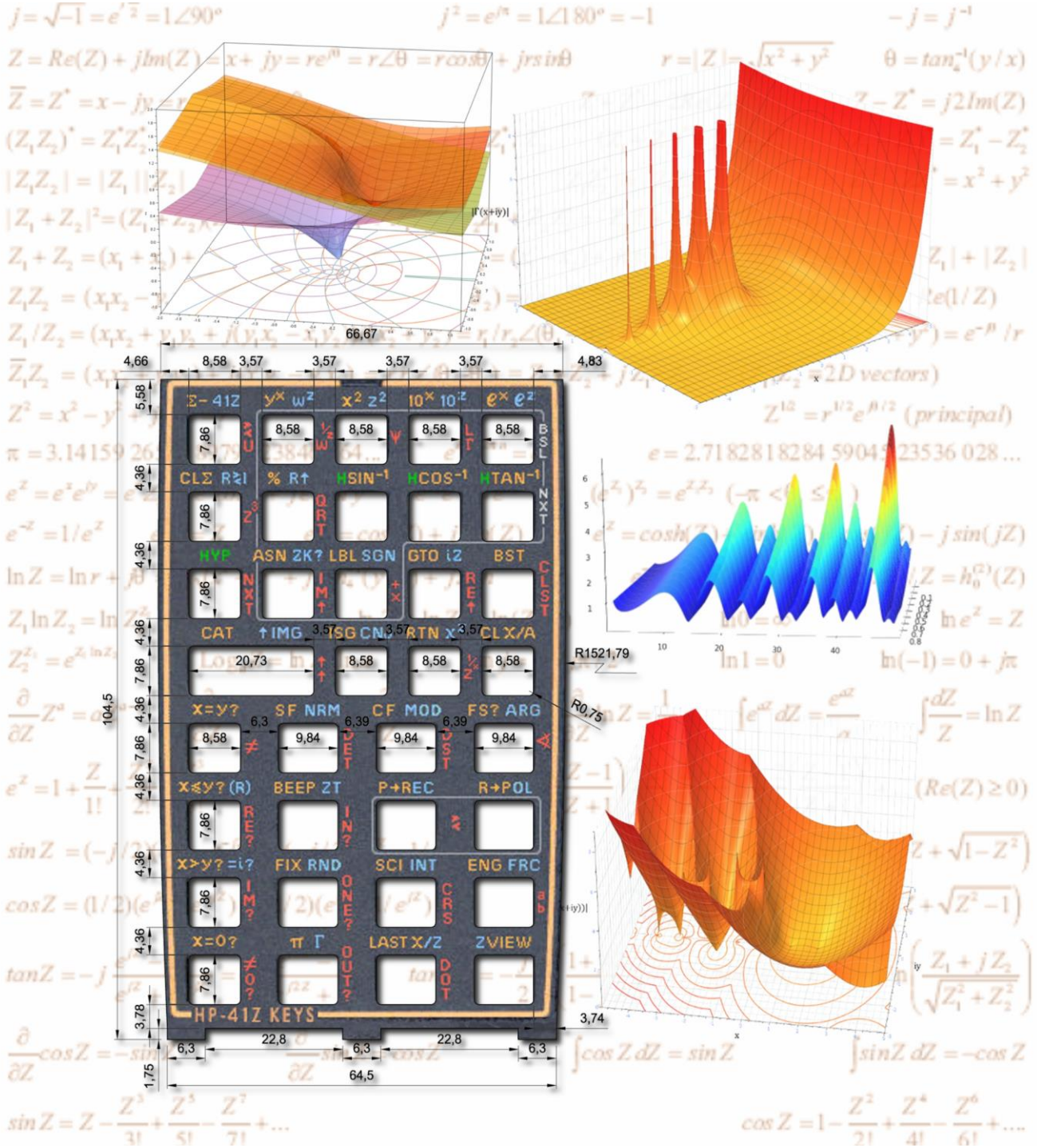
HP-41Z DELUXE++
Complex Number Module



Written and developed by Ángel Martin

May 2021

This compilation, revision A.9.2.0

The author wishes to thank the contributors to this project in various ways, as follows:

Jean-Marc Baillard, a constant reference for all math routines. He also contributed programs for the Riemann's Zeta, Poly-Gamma and Complete Elliptic Integrals.
Greg McClure, who contributed the Complex Derivative engine and the Continued Fractions.
Håkan Thörngren for his assistance and advices on the Memory Buffer implementation,
W. Doug Wilder, who wrote the initial code for the non-merged functions in program mode,
Valentín Albillo, who wrote the original "PROOT" FOCAL program,
M. Luján García, who prepared the 41Z Keys overlay bitmap file.

Some graphics taken from http://www.clarku.edu/~djoyce/complex, copyright 1999 by David E. Joyce.
Some graphics taken from http//www.wikipedia.org

Screen captures taken from V41, Windows-based emulator developed by Warren Furlow. See http://www.hp41.org/

Table of Contents.

0. Preamble: a Complex Relapse.- Reloaded. 7

1. Introduction.

1.1. Function Launchers. 8
1.2. Last Function functionality 9

2. Complex Stack, number entering and displaying.

2.1 Rectangular vs. Polar modes 11
2.2 Data entry conventions 12

3. User interface enhancements.

3.1 Display and Conversion functions 13
3.2 Complex Natural Data entry 14
3.3 The Complex User Assignments 17
3.4 The Complex Keyboard 18

4. Stack and Memory functions.

4.1. Stack functions group 22
4.2. ZSTO/ZRCL Math function groups 26

5. Complex Math.

5.1. Simple Arithmetic 27
5.2. Exponentials and Powers that be 29
5.3. Complex Logarithm 34

6. Complex Geometry

6.1 Basic functions 36
6.2 Complex Comparisons 39

7. Complex Trigonometry

7.1 Basic Functions 42
7.2 Complex Fibonacci numbers 45

8. 2D-vectors or complex numbers?

8.1 Two parallel worlds 46
8.2 Alternate Displaying: Quads and Tones 47

9. Polynomial Roots and Evaluations

9.1. Polynomial Evaluation 48
9.2. Polynomial Primitive and Derivatives 49
9.3. Solution to quadratic and cubic equations 50
9.4. General Polynomial Root Finder 53

10. It's a Gamma-Zeta world out there

10.1. Lanczos approximation 56
10.2. Digamma and LnGamma 58
10.3 Poly-Gamma function 60
10.4 Inverse Gamma and Catalan Numbers 61
10.5. Riemann's Zeta function 62
10.6. Lambert W function 64

11. Complex Means, Elliptic Integrals and DFT.

11.1. Arithmetic, Harmonic and Geometric Means 65
11.2. Dual-means: AGM and GHM 65
11.3. Complete Elliptic Integrals K(m), E(m) 66
11.4. Incomplete Elliptic Integrals, F(a; m) and E(a; m) 67
11.5. Discrete Fourier Transform (DFT) 70

12. General Methods and Special Functions

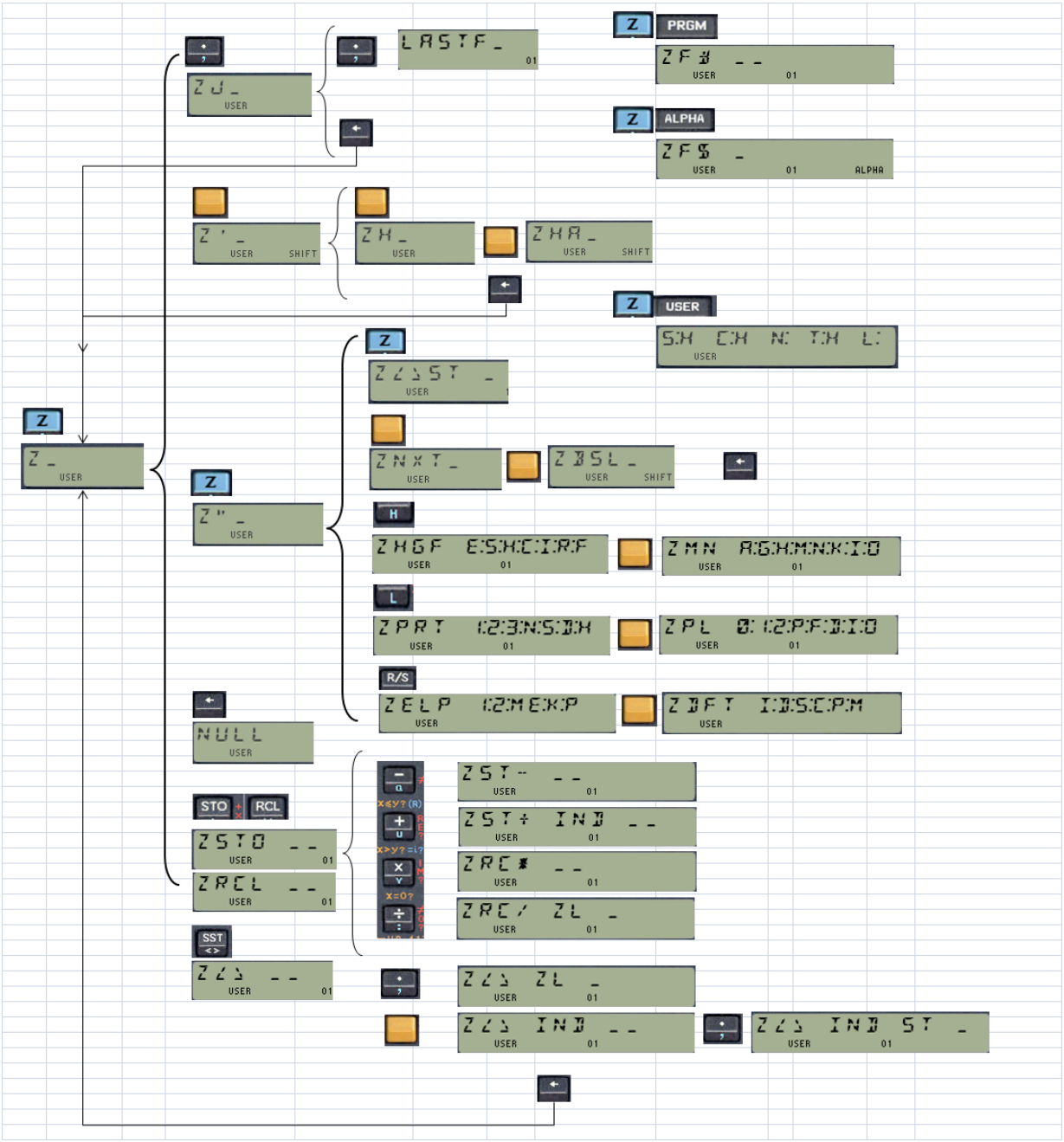
12.0. Real variable Functions 72
12.1. Multi-valued functions Driver 73
12.2. Solutions to f(z)=0 75
12.3. Successive Approximation Method 81
12.4. Complex Function Derivatives 83
12.5. Complex Continued Fractions 87
12.6. Bessel Functions 91
12.7. Spherical Hankel functions 96
12.8. Weber & Anger functions 97
12.9. Dilogarithm and Polylogarithm 100
12.10 Lerch Transcendent 101
12.11 Exponential Integrals 102

Appendices.

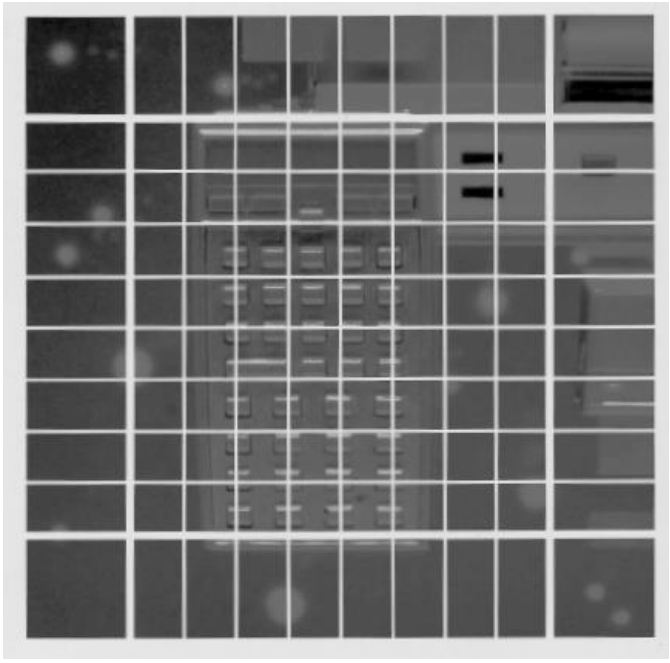
a0.- Appendix: Delta-Wye Transformation 104
a0.- Saving & Restoring the Z-Stack in X-Memory 105
a1.- Complex Buffer functions 107
a2.- Complex Keyboard key-maps 113
a3.- Formula compendium 114
a4.- Quick Reference Guide 115
a5.- Complex functions logic 119

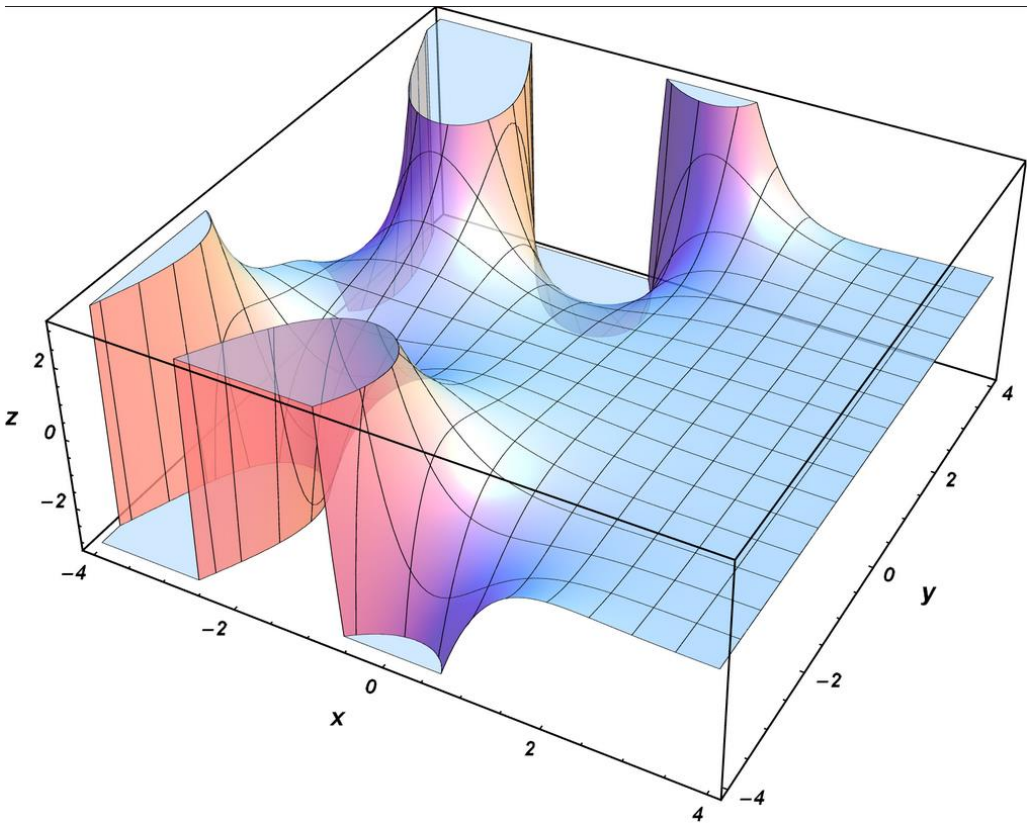
Appendix 0. – Sketch of the 41Z Launchers Map

The figure below shows the hierarchy and dependencies between all launchers. Note that only those choices prompting to other levels are shown, not all prompting functions (like ZSTO, ZRCL, Z<>, ZVIEW, ^IM/AG, etc.



Note. Within reason, this module adopts the general convention to always use MCODE headers for all functions, even for those which really are FOCAL programs. This improves readability, reduces the code size, and facilitates coding them as extensions to the launchers. The drawback is that the 41 OS interprets the programs to be in PRIVATE mode and therefore you won't be able to see the steps. Use the program listings within this manual instead. Their names are in BLACK font color to differentiate them from the native MCODE ones, which are in BLUE.




z
2
0
−2
−4
−2
0
2
4
x
4
2
0
−2
−4
y

41Z Deluxe – Complex Number Module for the HP-41

0. Preamble - A Complex Relapse - Reloaded

The 41Z module was the author's first project to use a combination of both MCODE and math techniques put together in service of a dedicated purpose. The design of the complex stack in particular was the subject of careful implementation and extensive testing – glad to say the effort has paid off and that the design has worked well to date.

This new revision benefits from bank-switching and the usage of Library#4 – a dedicated ROM packed with MCODE routines used frequently and repeatedly by several other modules (SandMath, PowerCL amongst others). Library#4 is located in page 4, and must be present on the system for this version of the 41Z module to work properly. All interaction occurs behind the scenes and transparently to the user.

There is a Library presence check made upon the Calculator ON event, showing an error message if it's not found - but otherwise the library is completely invisible to the user. Refer to the appropriate instructions manual for installation details. For compatibility reasons, *make sure you have revision "Q" or higher of the Library#4 ROM*.

Changing the original code to take advantage of the library took some effort, but the benefits of doing so have been twofold: The revised code is more robust and better structured, plus a lot of room was recovered and used for new functionality.

The following summarizes the most important changes in the Deluxe version:

1. Extended the memory access functions functionallity to fully support the stack registers. Both directly and with indirect arguments, with dedicated prompts and interrelationships. Furthermore, the function arguments are now entered as non-merged program lines directly by the function itself. This implementation applies to **ZRCL**, **ZSTO**, **Z<>**, and **ZVIEW** and is a direct port from the Total_Rekall module applied to complex registers. Also added RCL Math functions to the set.

2. Implemented an auxiliary FAT to allocate many other additions – mostly in the High-Level Math but not exclusively. Also re-instated the less relevant functions (such as **ZIMAG**, **ZREAL**, **ZHALF**, **ZDBL**, etc.) as sub-functions in the auxiliary FAT. The auxiliary FAT is also the home for all the second-tier sub-launchers underneath the main ΣZL function, such as **ZMTV**, **ZHYP**, **ZNXT**, **ZBSL**, etc.

3. Added sub-function launchers – **ZF$** by name and **ZF#** by index#. This implementation is analogous to other modules and adheres to the U/I guidelines developed for sub-function design and usage. ALPHA prompts will be made directly in the by-name launchers. The sub-function index is added as a non-merged program line in PRGM mode.

4. Convenient implementation of the "Last Function" functionality – for direct re-execution of the last used function without retyping its name or navigating the launchers and menu structures. All functions called from any of the dedicated launchers will be captured, included main-FAT entries or functions from other modules as well.

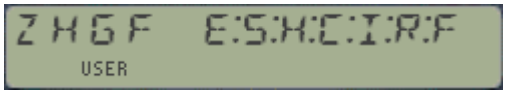
5. Addition of MCODE implementations of the Continued Fractions evaluation (**ZCF2V**) and 10-point Complex Derivative Engine (**ZDERV**) – both written by Greg McClure. Use it to calculate the first and second derivatives of a user-defined function programmed in memory as a FOCAL routine using the 41Z functions.

6. Added set of MCODE functions for Complex Means (Arithmetic, Geometric, Harmonic and their dual forms), as well as Polynomial Evaluation, including the first and second derivatives and its primitive. The complex **ZAGM** will also be used for the Elliptic Integrals routines.

7. Added set of functions to calculate the Complete and incomplete Elliptic integrals of first and second kinds - with complex amplitudes or modulus. Some routines require the SandMath module to work.

8. Added multiple functions in the High-Level math section. Seven of them are to calculate the Error function and the Exponential, Sine, Cosine (and their Hyperbolic counterparts) integrals – **ZHGF**, the Complex Hypergeometric Function (written by Jean-Marc Baillard). **ZERF**, **ZEI**, **ZCI**, **ZHCI**, **ZSI**, and **ZHSI** - all using the Hypergeometric Function method. The remaining three are **ZLERCH** for the Lerch transcendental, plus **ZLI2** and **ZLIN**, to calculate the Polylogarithm. All of them work with complex arguments.

9. Usage of section headers, so they can be called in FOCAL programs to perform actual calculations. This is the case for **–ZVECTOR** (which performs **ZGPRD**), **-ZSTACK** (which does **HARMN**) and **–HL ZMATH** (which performs **2^X-1**). These "hidden" functions are only used in dedicated sections of the module and/or FOCAL programs. This includes double-duty usage of the new function **ZHGF** –In RUN mode it is a new function launcher, grouping the functions that implement this calculation method. However in a running program it performs the actual execution work.



10. Added two MCODE functions for the Discrete Fourier Transform calculation on a set of complex data points, direct and inverse. **ZDFT** and **ZIDFT** will work on a set of complex data registers defined by its control word bbb.eee in X – returning the transformed set to a contiguous set of registers following that sample.

11. And last but not least, numerous changes in the code all throughout the module, rearranged sections and overall improvement in the consistency and usability of the functions - notably **NXTNRT** prompts when called from the **ZNEXT** launcher; now allows using the top two key rows (A – J) for index shortcuts 1-10.

Warning: due to all those function removals and additions, this version of the 41Z module has slightly different function arrangement in the FATs. If you have written your own programs using 41Z functions they may not match the new XROM id#'s and therefore will need to be re-written. At this point in the game this is highly unlikely, but just in case this is to be observed.

Note for Advanced Users:

The 41Z Deluxe is a bank-switched module. The bank switching will happen on both pages simultaneously therefore the module should not be plugged on "straddled" port configurations. Note also that you cannot configure only one page of the 41Z Deluxe module – therefore the footprint will always take a complete external port in the ROM bus.

Page	Bank-1	Bank-2
Upper Page XROM #04	Main FAT w/ High-Level Math, Zvectors,	Function Tables and Launcher M-Code
Lower Page XROM #01	Main and Aux-FATs, Z-stack	Lower-level Math Routines. MCODE only.

1. Introduction.

Complex Number handling is perhaps the most notable area where the HP-41 didn't have a comprehensive set of native functions, written in machine code and so taking advantage of the speed and programming enhancements derived from it. While both the Math Pack and the Advantage Rom provide FOCAL programs for complex number treatment, neither of them could be properly consider as a full function set to the effect of, for instance, the powerful Matrix handling functions contained in the Advantage Rom (in turn an evolution of those implemented in the CCD Module).

The 41Z module provides a significant number of functions that should address the vast majority of complex number problems, in a user-friendly context, and with full consistency. To that goal this manual should also contribute to get you familiar with their usage and applications, hopefully learning a couple of new things and having some fun during the process.

The implementation provided in this 16k-module is a fourth-generation code, building on the initial 41Z ROM released by the author in April 2005 – and on the previous version released in 2009. Numerous improvements have been added to the initial function set, notably the addition of a *4-level complex stack, a POLAR mode,* and a fully featured *complex mode keyboard*. Memory management is facilitated by prompting functions that deal with complex arguments, like **ZSTO**, **ZRCL**, (both with full math support), **Z<>,** and **ZVIEW** – all of them fully programmable as well.

1.1. Launchers and Last Function functionality.

The 41Z Deluxe includes full support for the "LASTF" functionality. This is a handy choice for repeat executions of the same function (i.e. to execute again the last-executed function), without having to type its name or navigate the different launchers to access it.

The implementation is not universal – it only covers functions invoked using the dedicated launchers, but not those called using the mainframe XEQ function. The following table summarizes the launchers that include this feature:

Module	Launchers	LASTF Method
41Z "Deluxe"	**ΣZL _**	Captures (sub)fnc id#
	ZHGF, ZPRT, ZNEXT, ZBSL, ZHYP	Captures (sub)fnc id#
	ZF$ _	Captures fnc NAME
	ZF# _ _ _	Captures (sub)fnc id#
	CAT+ (XEQ)	Captures (sub)fnc id#

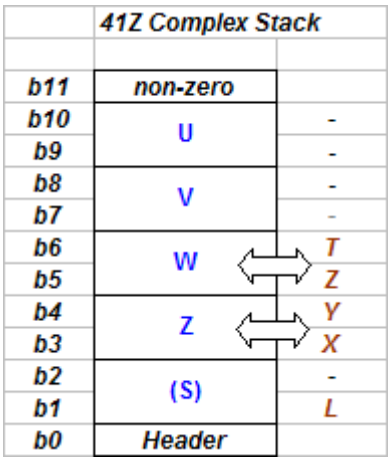
LASTF Operating Instructions

The Last Function feature is triggered by pressing the radix key (decimal point - the same key used by LastX) ***twice*** at the "**Z:** " prompt. When this feature is invoked, it first shows "LASFT" briefly in the display, quickly followed by the last-function name. Keeping the key depressed for a while shows "NULL" and cancels the action. In RUN mode the function is executed, and in PRGM mode it's added as a program step if programmable, or directly executed if not programmable.

If no last-function record yet exists, the error message "NO LASTF" is shown. If the buffer #9 (used to store the last function id# code) is not present, the error message is "NO BUF" instead.

2. Complex Stack, number entering and displaying.

A four-level complex stack is available to the user to perform all complex calculations. The complex stack levels are called ***U***, ***V***, ***W***, and ***Z*** – from top to bottom. Each level holds two real numbers, the imaginary and real parts of the corresponding complex number. Besides them, a "LastZ" complex register ***S*** temporarily stores the argument of the last executed function.



The complex stack uses a dedicated buffer in main memory. It is created and maintained by the 41Z module and its operation should be transparent to the user. This buffer is independent from the real stack (X, Y, Z, and T registers) but it's important however to understand how they interact with each other. A complex number uses two real stack levels (like X and Y), but a single complex stack level (like ***Z*** or ***W***). The figure on the left shows the relationship between the complex and real stacks, which is automatically maintained upon function execution, as we'll see later on.

The real stack is used to enter the complex number values, real and imaginary parts. The input sequence varies depending on the method used *but all functions will expect the imaginary part in the Y register and the real part in the X register.* More about this later.

The contents of complex and real stack levels are *automatically synchronized* before and after each complex operation is performed. This may just involve real levels X,Y and complex level ***Z*** if it's a monadic (or unary) operation requiring a single complex argument, or may also involve real levels Z,T and complex level ***W*** if it's a dual operation requiring two complex arguments.

Monadic functions will assume that the real numbers in X,Y are the most up-to-date values for the real and imaginary parts of the complex argument. They will overwrite the contents of complex level ***Z***. This allows quick editing and modification of the complex argument prior to executing the function.

Dual functions will assume that the second argument is stored in ***W***, that is level 2 of the complex stack, and *will thus ignore the values contained in real stack registers Z,T*. Note that because the real stack overflows when trying to hold more than four different values, it is not a reliable way to input two complex numbers at once.

The design objective has been to employ as much as possible the same rules and conventions as for the real number stack, only for complex numbers instead. This has been accomplished in all aspects of data entering, *with the exception of automated complex stack lift*: with a few exceptions, entering two complex numbers into the complex stack requires pressing **ZENTER^** to separate them.

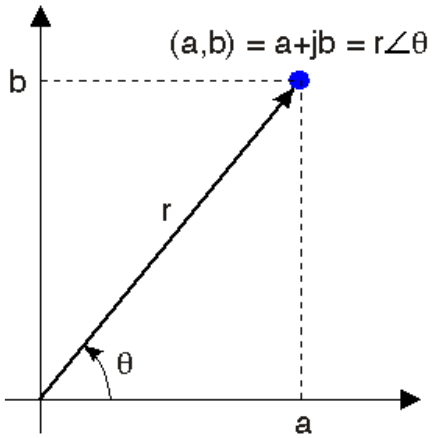
Once again: entering two complex numbers into the complex stack is accomplished by executing **ZENTER^** to separate the first and second complex number. Exceptions to this rule are the other complex-stack lifting functions, such as **ZGEU**, **LASTZ**, **ZPI**, **ZRCL**, **ZRPL^**, **ZIMAG^**, **ZREAL^**, **^IM/AG**, and the "***Complex Keypad***". Here the left-side symbol "^" (SHIFT-N) represents an input action.

2.1 Rectangular vs. Polar forms.

The HP-41 sorely lacks a polar vs. Rectangular mode. This limitation is also overcome on the 41Z module, with the functions **POLAR** and **RECT** to switch back and forth between these modes. It uses an internal flag in the complex buffer, not part of the 41 system flags. The operation is simplified in that *complex numbers are always stored in their rectangular (or Cartesian) form*, *z*=x+yi.

So while all functions expect the argument(s) in rectangular form, yet *the results are shown in the appropriate format as defined by the POLAR or RECT mode*. (The notable exception is **ZPOL**, which is stored as values in Polar form). However, direct manual entry of complex in polar form is also possible using the **^IM/AG** function described in the following pages.

Note also that the POLAR mode is directly affected by the angular mode as well, as it occurs with real argument values.



Note: The POLAR display of the complex number requires an additional R-P conversion after the result is calculated in Cartesian form. The Polar form is temporarily stored in the Real stack registers T,Z – *which typically have no active role in the Complex Stack* and therefore can always be used as scratch. Once again, no changes are made to either X,Y registers or Complex stack level **Z**.

2.2 Data Entry Conventions

And how about complex number entering? Here the world divides in two camps, depending on whether the sequence is: "Re(*z*), ENTER^, Im(*z*)" – like on the HP-42S - , or its reverse: "Im(*z*), ENTER^, Re(*z*)" – like on the HP-32/33S and other FOCAL programs -. With the 41Z module you can do it either way, but it's important to remember that *regardless of how you introduce the numbers, all functions expect the imaginary part in the Y real-stack register and the real part in the X real-stack register.*

Fast data entry will typically use the sequence Im(z) , ENTER^, Re(z), followed by the complex function. This is called the "Direct" data entry, as opposed to the "Natural" data entry, which would first input the real part. The 41Z module includes the function "**^IM/AG**" that can be used to input the number using the "Natural" convention (reversed from the Direct one).

Its usage is the same as the "i"-function on the HP-35s, to separate the real and the imaginary parts. The sequence is completed by pressing ENTER^ or R/S, after which the imaginary part will be left in the Y register and the real part in the X register as explained before.

(Incidentally, the 42S implementation of the complex stack isn't suitable for a true 4-level, since the COMPLEX function requires two levels prior to making the conversion!)

Other functions and special functionality in the 41Z module can be used as shortcuts to input purely real or imaginary numbers more efficiently. For instance, to enter the imaginary unit one need only press: 1, **ZIMAG^** (which is also equivalent to executing the **IMAGINE** function) – or simply **"ΣZL, Radix, 1**" using the "complex keypad". And to enter 4 as a complex number, just press: 4, **ZREAL^ -** or simply "**ΣZL, 4**" using the "complex keypad".

Incidentally, the 42S implementation fails short from delivering a true 4-level stack, due to the COMPLEX function and the fact that it requires two stack levels to be available to combine the complex number. In this regard the 41Z solution is a better one.

Two (opposite) alternatives to data entry: COMPLEX key on the 42S, and " ï" key on the 35S

3. User interface enhancements.

Table-3.1: Functions to enhance the user interface.

Index	Function	Group	Description
1	**ZK?YN _**	Usability	Activates and deactivates the Complex Assignments
2	**ΣZL _**	Usability	Accesses most of the 41Z functions plus special features
3	**ZAVIEW _ _**	Display	Views complex number in X,Y (prompts for # decimal places)
4	**POLAR**	Display	Displays complex numbers in Polar form
5	**RECT**	Display	Displays complex numbers in Rectangular form
6	**^IM/AG _**	Usability	Inputs Imaginary Part (or Argument) of complex number

These functions facilitate the showing of the complex number on the display, and the conversion between the polar and rectangular forms. They enhance the usability by supplying a system to handle the lack of native complex number treatment capabilities of the calculator.

3.1 Display mode and conversion functions.

ZAVIEW _ __ _	**Complex number AVIEW**	Uses ALPHA registers	Prompts for # decimal places

Shows the contents of the complex stack level **Z** in the display, using the current complex display mode (POLAR or RECT).:

RECT: Re(*z*) + J Im(*z*) ; where Re(*z*) is stored in register X and Im(*z*) in register Y.
POLAR: Mod(z) <| Arg(z); *shown but not stored in the X,Y stack registers* (!)

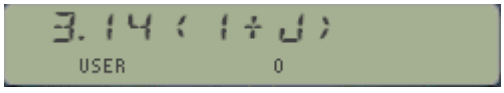
Note that **ZAVIEW** uses the ALPHA register, thus the previous contents of the M, N and O registers will be lost.

The displaying will respect the current DEG, RAD, or GRAD angular mode (in POLAR form), the current FIX, SCI or ENG settings. In RUN mode you have the choice to input the number of decimal places in the function's prompt – whilst in Program it'll use the selected settings on the calculator. Note that "J" precedes the imaginary part, as this improves legibility with real-life complex numbers, with decimal imaginary parts.

For a enhanced visualization, **ZAVIEW** *won't show decimal zeros if the number is an integer.* This is done automatically regardless of the number of decimal places selected in the calculator; so one can immediately tell whether the real or imaginary parts are true integers as opposed to having some decimal content hidden in the least significant places not shown.

versus:


ZAVIEW *will also extract common factor if both the real and imaginary parts are equal:*

or also:


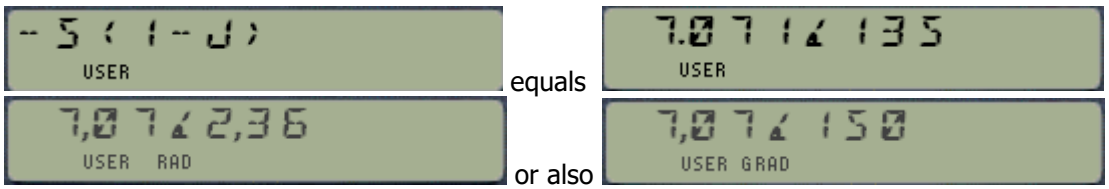
Executing the functions **POLAR** and **RECT** will also display the complex number currently stored in X,Y

POLAR	**Sets POLAR mode on**	Displays number	Shows in SET mode
RECT	**Sets RECT mode on**	Displays number	Shows in SET mode
ZPOL	**Convert to Polar**	Converts X,Y to POLAR	*Always shows in POLAR*
ZREC	**Convert to Rectangular**	Converts X,Y to RECT	Shows in SET mode

ZPOL Converts the complex number in the **Z** stack level from rectangular to polar mode. If executed in run mode, the display shows the value of its magnitude (its module) and its argument, as follows:

Mod < Arg ; where:
Mod = |z| and Arg=α **[*z* = |z|* e^iα]**

The argument value will be expressed in the angular settings currently selected: DEG, RAD, or GRAD.



ZREC is the reciprocal function, and will convert the complex number in **Z** (assumed to be in polar form) to rectangular form, showing it on the display (in run mode) in identical manner as **ZAVIEW**.

In fact, if it weren't because of the displaying capabilities, these two functions will be identical to the pair R-P and P-R, standard on the calculator. Recognizing this, they're assigned to the very same position as their real counterparts on the Complex User keyboard.

Notice that contrary to the **POLAR** and **RECT** functions (which only display the values), **ZPOL** and **ZREC** *perform the actual conversion of the values and store them in the stack registers* (complex and real). This is also very useful to enter complex numbers directly in polar form, simply using the sequence: (direct data entry: Angle first, then modulus):

- Arg(z), ENTER^, |z|, **ZREC** -> Re(z) + J Im(z)

3.2 Complex Natural Data Entry.

This function belongs to its own category, as an automated way to input a complex number using the "Natural" data entry convention: Real part first, Imaginary part next. Its major advantage (besides allowing the natural data entry sequence) is that *it performs a complex stack lift upon completion of the data entry*, thus there's no need to use **ZENTER^** to input the complex number into the complex stack. That alone would justify its inclusion on the 41Z module.

^IM/AG _	**Inputs Im(z)/Arg(z) Part**	Does Stack Lift	Prompting function

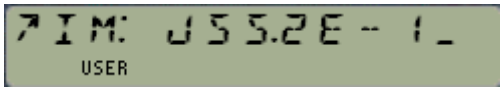
The function will prompt for the imaginary part (or the argument if in POLAR mode) of the complex number being entered. The design mimics that on the HP-35S calculator, and it's used as a way to separate the two complex parts during the complex number data entering.

A few important considerations are:

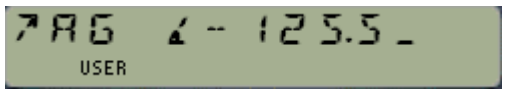
- The real part (or module) must be introduced right *before* calling it, so it's in X during the data entry.
- The keyboard is redefined to allow for numeric digits, RADIX, CHS and EEX as only valid keys.
- The radix symbol used (comma or dot) is controlled by the user flag 28.
- Only one RADIX character will be allowed in the mantissa – and none in the exponent.
- Only nine digits will be used for the mantissa, and two in the exponent. **^IM/AG** will not check for that during the input process, but exceeding entries will simply be ignored.
- Only one EEX can exist in the imaginary part - **^IM/AG** will check for that.
- Only one CHS can be used for the mantissa sign, **^IM/AG** will check for that.
- Multiple CHS can be used for the exponent sign, but **^IM/AG** will apply the arithmetic rules to determine the final sign as follows: odd number is negative, even number is positive.
- Pressing Back Arrow will remove the last entry, be that a number, Radix, EEX or CHS. If the entry is the first one it will cancel the process and will discard the real part as well.
- The sequence must be ended by pressing ENTER^ or R/S.
- The display cue is different depending on the actual complex mode (RECT or POLAR), and it's controlled automatically.
- Upon completion, the complex number is pushed into the **Z** complex stack level, and placed on the X,Y real stack registers as well following the same 41Z convention: real part in X and imaginary part in Y. The complex stack is lifted and the real stack is synchronized accordingly.

The screens below show usage examples in RECT and POLAR modes:

until finally:

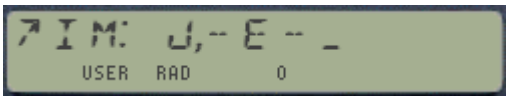


ending as:

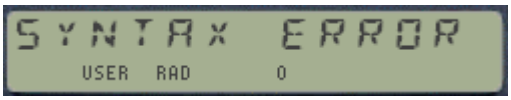


Note: To extract the numeric value from the input string, **^IM/AG** executes the same code as the X-function **ANUM**. All conversion conventions will follow the same **ANUM** logic. Suffice it to say that the implementation of **^IM/AG** is not absolute perfect and you can trip it up if that's what you really want – but it should prevent likely errors that could yield incorrect results. It's a very convenient way to meet this need solving the diverse issues associated with its generic character.

If the input string doesn't yield any sensible numeric result, the message "SYNTAX ERROR" is briefly shown in the display, and the stack is restored to its status prior to executing **^IM/AG**.



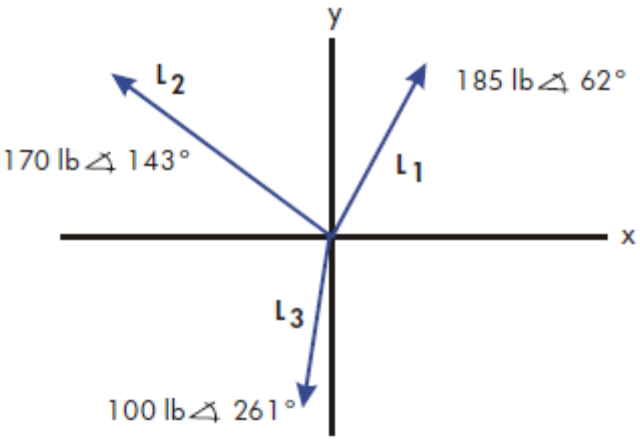
will trigger:



Some apparently incorrect syntax constructions will however be properly interpreted by **^IM/AG**, returning a valid imaginary part. This is for instance the case with multiple negative signs in the exponent, or decimal values after negative sign in the mantissa. Such is the flexibility of the **ANUM** function!

Example: *Vector Load addition* (taken from the 35s User Guide):-

We start by setting POLAR and DEG modes, then using the **^IM/AG** function three times will set the three complex numbers on the complex stack, and finally simply execute the complex addition function Z+ twice:



POLAR, DEG
185, **^IM/AG**, 62, ENTER^
170, **^IM/AG**, 143, R/S
100, **^IM/AG**, 261, R/S
Z+, Z+

Result: -> 178,9372 <) 111,1489

Or in Rectangular mode (as it's saved in XY):
RECT -> -64,559 + J166,885

Note the following points:

- We used indistinctly ENTER^ and R/S to terminate the complex number entry.
- No need to store intermediate results as the complex buffer can hold up to four levels.
- We didn't need to use **ZENTER^** to push the complex numbers into the complex stack because the stack-lift was performed by **^IM/AG**.

With regard to the data entry sequence, one could have used **ZREC** instead of **^IM/AG** – albeit in that case it would have been in "direct mode", as opposed to the more intuitive natural convention. It also requires pressing **ZENTER^** to push each number into the complex stack.

This is the keystroke sequence and partial results (assuming we're in **POLAR** mode)

62, ENTER^, 185, **ZREC**, **ZENTER^**	-> 185 <)62
143, ENTER^, 170, **ZREC**, **ZENTER^**	-> 170 <)143
261, ENTER^, 100, **ZREC**	-> 100 <)-99
Z+, Z+	-> 178,9372 <) 111,1489

One last remark about data displaying vs. data entry.- As it was explained before, **ZPOL** will convert the complex number into Polar coordinates, and it will be displayed in **POLAR** form even if **RECT** mode is selected. This is the single one exception all throughout the 41z module, and it will only work immediately after pressing **ZPOL** but not for subsequent executions of **ZAVIEW** – which always expects the number is stored in rectangular form, and therefore will show an incorrect expression.

3.3 The Complex User Assignments.

The 41Z module provides a convenient way to do user key assignments *in masse*. Given the parallelisms between the real and complex number functions, the natural choice for many of the functions is "predetermined" to be that of their real counterparts.

A single function is used for the mass-assignment (or de-assignment) action:

ZK?YN _	**Complex User Assignments**		Prompting function

ZK?YN automates the assignment and de-assignment of 37 functions. It prompts for a Yes/No answer, as follows:

- Answering "Y" will assign the complex functions to their target keys
- Answering "N" will de-assign them, and
- Pressing "Back Arrow" will cancel the function – and display the Z-level content.
- Any other key input (including ON) will be ignored.

The assignment action will be indicated by the message "Z-KEYS: ON" or "Z-KEYS OFF" in the display during the time it takes to perform, followed by "PACKING" – and possibly "TRY AGAIN" should the enough number of memory registers not exist.

Note that **ZK?YN** is *selective*: any other key assignment not part of the complex functions set will not be modified.

Keycode	Unshifted Keys		Shifted Keys	
11	s+	ZHYP	s-	ZNXT
12	1/X	ZINV	y^x	W^Z
13	SQRT	ZSQRT	x^2	Z^2
14	LOG	ZLOG	10^x	ZALOG
15	LN	ZLN	e^x	ZEXP
21	x<>y	Z<>W	CLs	ZTRP
22	RDN	ZRDN	%	ZRUP
23	SIN	ZSIN	ASIN	ZASIN
24	COS	ZCOS	ACOS	ZACOS
25	TAN	ZTAN	ATAN	ZATAN
33	STO	ZSTO	LBL	n/a
34	RCL	ZRCL	GTO	n/a
41	ENTER^	^IMG	CAT	ZENTER^
42	CHS	n/a	ISG	ZNEG
44	EEX	n/a	CLx	CLZ
51	-	Z-	x=y?	Z=W?
61	+	Z+	x<=y?	Z=WR?
71	*	Z*	x>y?	Z=I?
81	/	Z/	x=0?	Z=0?
83	,	n/a	LASTx	LASTZ
84	R/S	ZAVIEW	VIEW	ZVIEW

Table 3.3. Complex key assignments done by ZK?YN

3.4 The Complex Keyboard.

As good as the user assignments are to effectively map out many of the 41Z functions, this method is not free from inconveniences. Perhaps the biggest disadvantage of the Complex Assignments is that it's frequently required to toggle the user mode back and forth, depending on whether it's a complex or a real (native) function to be executed.

Besides that, the Complex Assignments consume a relative large number of memory registers that can be needed for other purposes. Lastly, there are numerous 41Z functions not included on the user assignments map, and no more "logical" keys are available without compromising the usability of the calculator.

To solve these quibbles, the 41Z module provides an alternative method to access the majority of the complex functions, plus some unique additional functionality. It's called the ***Complex Keyboard***, accessed by the function **ΣZL**: a single key assignment unleashes the complete potential of the module, used as a ***complex prefix***, or in different combinations with the SHIFT key and with itself.

Figure 3.4. Complex Keyboard overlay (with ΣZL assigned to Sigma+).
On the left: the version for V41. On the right, for i41CX

© 2009 M. Luján García.

The 41Z overlay can be downloaded from the HP-41 archive website, at:
http://www.hp41.org/LibView.cfm?Command=View&ItemID=893

To use it with V41 emulator, replace the original file "*large.bmp*" in the V41 directory with the 41Z bitmap file, after renaming it to the same file name.

Here's how to access all the functions using **ΣZL**:

- **Direct functions**. Simply press "**Z**" as a prefix to denote that the next function will operate on a complex argument, and not on a real one. These functions don't have any special marks, as they correspond to the standard functions on the HP-41 keyboard. There are *twenty 41Z functions* directly accessible like these.

Examples: Pressing **Z**, LN will execute **ZLN**; pressing **Z**, COS will execute **ZCOS**, etc…
Pressing **Z**, + will execute **Z+**; pressing **Z**, R/S will execute **ZAVIEW**,

- **Shifted functions**. Press "**Z**" followed by the SHIFT key. These functions are either marked in blue when different from the standard SHIFTED ones, or just marked in yellow as part of the standard HP-41 keyboard (like x=y?, which will execute **Z=W?** if the pressed key sequence is this: **Z**, SHIFT, x=y?

Examples: pressing **Z**, SHIFT, LN will execute **ZEXP**; pressing **Z**, SHIFT, SIN will execute **ZASIN**,
Pressing **Z**, SHIFT, R/S will execute **ZVIEW** (a prompting function itself).

There are *thirty-one 41Z functions* accessible using this SHIFTED method.

- **Dual (alternate) functions**. Press "**Z**" *twice* as a double prefix to access the dual complex functions and many others. These functions are marked in red, on the right side of each available key.

Examples: Pressing **Z, Z**, 7 will execute **ZWDET**; pressing **Z, Z**, 5 will execute **ZWCROSS**, , and so on with all the "red-labeled" keys.

Pressing **Z, Z**, ENTER^ will execute **ZREPL**; pressing **Z, Z, Z** will execute **Z<>U**
There are *twenty-five 41Z functions* accessible using this Dual method.

- **Multi-value functions**. As a particular case of the dual functions case above, the ZNEXT function group is enabled by pressing "**Z**" *twice* and then SHIFT. This group is encircled on the keyboard overlay, and sets the five multi-value functions as follows: **NXTASN, NXTACS, NXTATN, NXTLN**, and **NXTNRT** (this one will also prompt for the root order, as an integer number 0-9).

Notice that pressing SHIFT while in the NEXT section toggles the display to "ZBSL". Use it as a shortcut to access the different Bessel functions of first and second kind provided in the 41, as follows: **ZJBS**, **ZIBS**, **ZKBS**, and **ZYBS**. – as well as **EIZ/IZ**, a particular case of Spherical Hankel h1(0,z).

- **Hyperbolic functions**. Press "**Z**" followed by SHIFT *twice* to access the three direct hyperbolics. Pressing SHIFT a third time will add the letter "A" to the function name and will enable the inverse functions. This action toggles with each subsequent pressing of SHIFT. (Watch the 41Z building up the function name in the display as you press the keys…)

Example: Pressing **Z**, SHIFT, SHIFT, SHIFT, **SIN** will execute **ZASINH**

- **Complex Keypads**. Press "**Z**" followed by a numeric key (0 to 9) to enter the corresponding digit as a complex number in the complex stack. Pressing "**Z**" followed by the Radix key, and then the numeric key will input the digit as an imaginary number as opposed to as a real number into the complex stack. This is a very useful shortcut to quickly input integer real or imaginary values for complex arithmetic or other operations (like multiplying by 2, etc.)

Pressing **Z**, XEQ calls the function **^IM/AG** for the Natural Data entry. This is obviously not shown on the keyboard – which has no changes to the key legends for un-shifted functions. Note that there are three different ways to invoke **^IM/AG**, as follows:

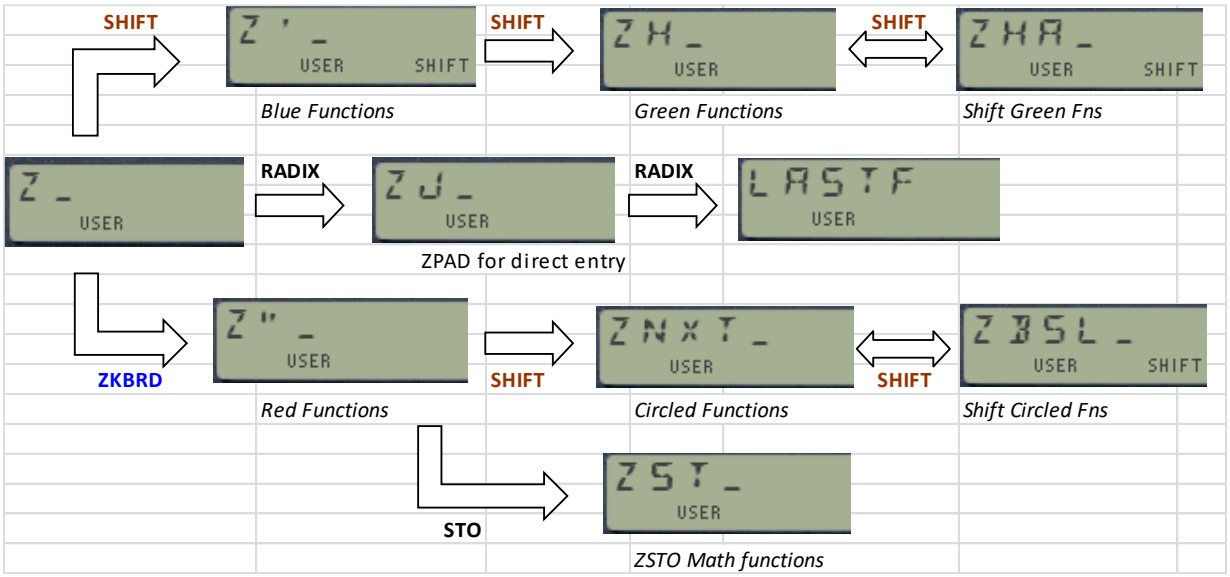
XEQ, ALPHA, SHIFT, N, I, M, /, A, G, ALPHA	-> the standard HP-41 method, or:
Z, SHIFT, ENTER^	-> shown in blue in the overlay, or:
Z, XEQ	-> not shown.

- ***Other keystrokes***. The 41Z module takes control of the calculator keyboard when **ΣZL** is executed. Available keys are determined by the partial key sequence entered, as defined on the 41Z Keys overlay and as explained above. Pressing **USER** or **ALPHA** will have no effect, and pressing **ON** at any time will shut the calculator off. The *back arrow* key plays its usual important role during data entering, and also undoes the last key pressed during a multi-shifted key sequence. Try it by yourself and you'll see it's actually easier than giving examples on how it works here.

In summary: a complete new keyboard that is accessed by the "**Z**" blue prefix key. This being the only requisite, it's a near-perfect compromise once you get used to it – but if you don't like it you can use the User Assignments , the choice is yours.

Quick Recap:

The figure below shows the main different modes of the **ΣZL** function, the real cornerstone of the 41Z module:



Press the Back-arrow key to bring the command chain back to the starting point (ΣZL). Pressing it twice shows "NULL" and cancels out the sequence.

Pressing non-relevant keys (i.e. those not supposed to be included in the corresponding mode) causes the display to blink, and maintain the same prompt (no action taken).

4. Stack and Memory functions.

Let **Z** and **W** be the lower two levels of the complex stack, and "z" and "w" two complex numbers stored in **Z** and **W** respectively. **Z** = Re(z)+ j Im(z); **W** = Re(w) + j Im(w)

Note the use of "j" to express the imaginary unit, instead of "i" . This isn't done to favor those EE's in the audience (you know who *we* are), but rather due to the displaying limitations of the 41 display: no lower-case letters for either i or j, and better-looking for the latter one in caps.

Note also that despite their being used interchangeably, the complex stack register "**Z**" – in bold font – and the real stack register "Z" – in regular font – are not the same at all.

Table-4.1: Stack and memory function group.

Index	Function	Name	Description
1	**ZTRP**	Re(z)<>Im(z)	Exchanges (transposes) Re and Im for number in level **Z**.
2	**ZENTER^**	Complex ENTER^	Enters X,Y into complex level **Z, lifts complex stack.**
3	**ZREPL**	Complex Stack Fill	Fills complex stack with value(s) in X,Y
4	**ZRDN**	Complex Roll Down	Rolls complex stack down
5	**ZRUP**	Complex Roll Up	Rolls complex stack up
6	**ZREAL^**	Inputs real Z	Enters value in X as real-part only complex number
7	**ZIMAG^**	Inputs imaginary Z	Enters value in X as imaginary complex number
8	**Z<>W**	Complex Z<>W	Swaps complex levels **Z** and **W**
9 (*)	**Z<>ST _ _**	Complex Z<> level	Swaps complex levels **Z** and any stack level (0-4)
10 (*)	**ZRCL _ _**	Complex Recall	Recalls complex number from memory to level **Z**
11 (*)	**ZSTO _ _**	Complex Storage	Stores complex number in **Z** into memory
12 (*)	**Z<> _ _**	Complex Exchange	Exchanges number in level **Z** and memory
13 (*)	**ZVIEW _ _**	Complex Display	Shows Complex number stored in memory register
14	**CLZ**	Clears Level Z	Deletes complex level **Z**
15	**CLZST**	Clears Complex Stack	Clears all complex levels **U, V, W**, and **Z**
16	**ZREAL**	Extracts real part	*Removed*. Replace with: X<>Y, CLX, X<>Y
17	**ZIMAG**	Extracts Imag part	*Removed*. Replace with: CLX
18	**LASTZ**	Last number used	Recovers the last complex number used

*(*) Note: These functions are **fully programmable**. When used in a program their argument is taken from the next program line, see below for details.*

4.1 Stack and memory functions group.

Let's start with the individual description of these functions in more detail, beginning with the simplest.

ZTRP	**Z Transpose**	Does Re <>Im	

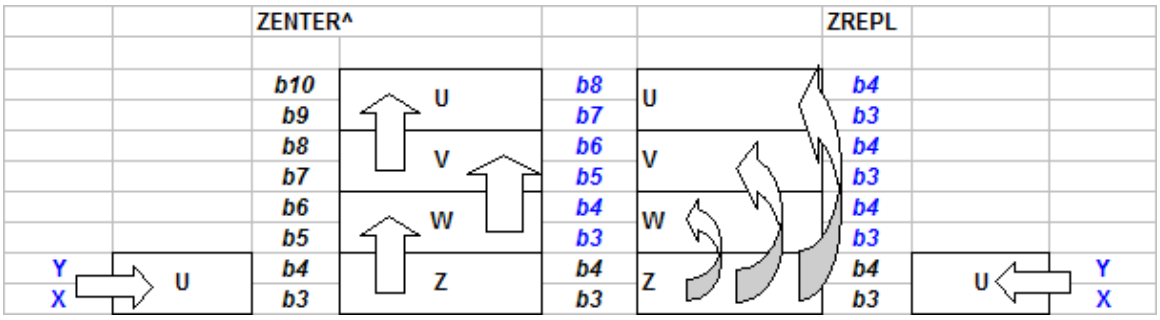
This function's very modest goal is to exchange the real and imaginary parts of the complex number stored in the **Z** level of the complex stack.

Hardly a worthwhile scope, you'd say, considering that the standard function X<>Y does the same thing? Indeed it is quite similar (and as such it's logically assigned to the shifted X<>Y key). But it's not quite the same, as in run mode **ZTRP** also shows on the display the complex number after transposing their real and imaginary parts. Besides, as it was mentioned in the introduction, this function may play an important role during data entry: it is the one to use when entering the real part first, as per the following sequence: Re(*z*), ENTER^, Im(*z*), **ZTRP**

Thus its use is analogous to the "COMPLEX" function on the HP-42S, also required to enter the complex number in the stack, from its two real components. Note that the other, alternative data entering sequence doesn't require using ZTRP, although the order of the real and imaginary parts is reversed (and arguably less intuitive): Im(*z*), ENTER^, Re(*z*). Either one of these two is entirely adequate once you become familiar with it and get used to using it - it's your choice.

ZENTER^	**Enters X,Y into levels Z, W**	Does Stack lift	
ZRPL^	**Fills complex stack**		

ZENTER^ enters the values in X,Y as a complex number in the **Z** stack level, and performs stack lift (thus duplicates **Z** into **W** as well – and **U** is lost due to the complex stack spill-over). As said in the introduction, *always* use **ZENTER^** to perform stack lift when entering two (or more) complex numbers into the complex stack. This is required for the correct operation of dual complex functions, like **Z+**, or when doing chain calculations using the complex stack (which, unlike the real XYZT real stack, it does NOT have an automated stack lift triggered by the introduction of a new real number).



ZRPL^ simply fills the complex stack with the values in the real registers X,Y. This is convenient in chained calculations (like the Horner method for polynomial evaluation). If executed in run mode it also displays the number in **Z**. This is in fact a common characteristic of all the functions in the 41Z module, built so to provide visual feedback on the action performed.

ZREAL^	**Enters X in Z as (x+j0)**	Does Stack Lift	
ZIMAG^	**Enters X in Z as (0+jX)**	Does Stack Lift	

These functions enter the value in X either as a purely real or purely imaginary number in complex form in the **Z** stack level, and perform stack lift. If executed in run mode it also displays the number in **Z** upon completion.

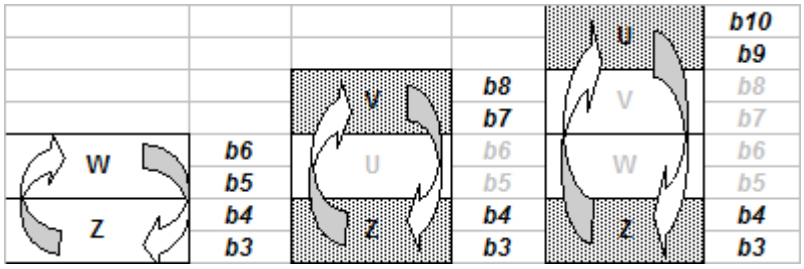
CLZ	**Clears complex stack level Z**		
CLZST	**Clears complete complex stack**		
ZREAL	Extracts Real part from Z	Sub-function	X<>Y, CLX, X<>Y
ZIMAG	Extracts Imaginary part from Z	Sub-function	CLX

Use these four functions to partially or completely clear (delete) the contents of the complex stack **Z** level, or the complete complex stack. No frills, no caveats. The real stack will also be cleared appropriately. Note that contrary to the real CLX function, the execution of **CLZ** will save the complex number in the complex LastX level "S".

Z<>ST _	**Exchanges Z and Stack**	Level# = 0,1,2,3,4	Prompting function (*)
Z<>W	**Exchanges Z and W**		

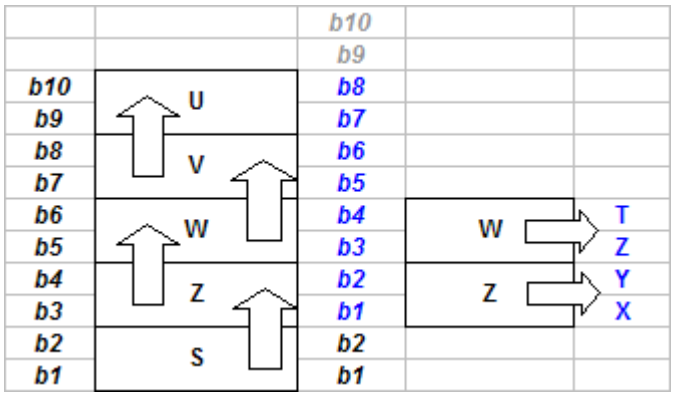
(*) Fully programmable, see note in following pages.

Use these functions to swap the contents of the **Z** and **U**/**V**/**W** levels of the complex stack respectively. As always, the execution ends with **ZAVIEW** in run mode, displaying the new contents of the **Z** register.(which is also copied into the XY registers).



LASTZ	**Recalls last number used to Z**	Does Stack Lift	

Similar to the LASTX function, **LASTZ** recalls the number used in the immediate preceding operation back to the **Z** level of the complex stack. A complex stack lift is performed, pushing the contents of **Z** up to the level **W**, and losing the previous content of **U**.



The majority of functions on the 41Z module perform an automated storage of their argument into the LastZ register, enabling the subsequent using of **LASTZ**. This will be notated in this manual when appropriate under each function description.

Example: to calculate [(z^2 + z)/2] simply press: **Z^2, LASTZ, Z+, ZHALF**

Example: Calculate the following expression without using any data registers:

F(z) = Ln [z + SQR(z^2 + 1)], for z= 20+20i

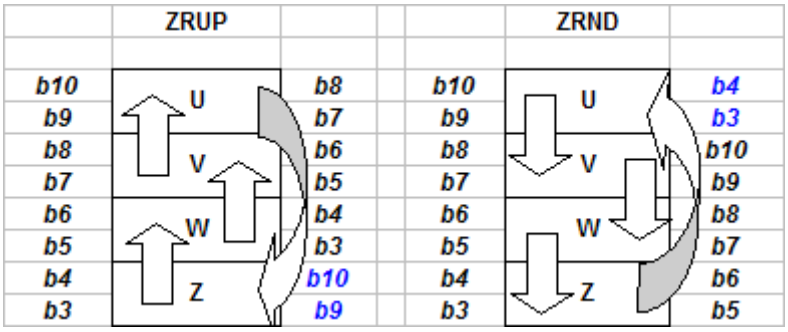
Solution:

2 , 0 , ENTER^ , **ZRPL**	-> puts 20+20i in all 4 levels of the complex stack
Z^2 , 1 , **ZREAL^** , **Z+**	-> could have used "1, +" as a more direct method
ZSQRT , **Z+** , **ZLN**	**-> 4,035+J0,785**

Congratulations! You just calculated the hyperbolic arcsine of (20+20i).

ZRDN	**Rolls complex stack down**		
ZRUP	**Rolls complex stack up**		

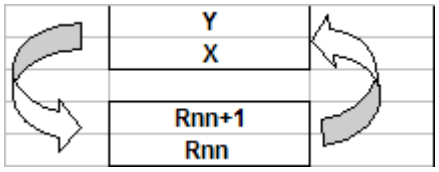
Like their real stack counterparts, these functions will roll the complex stack down or up respectively. If executed in run mode it also displays the number in **Z**. Real stack registers will be synchronized accordingly.



Be aware that although **ZRDN** and **ZRUP** do not perform stack lift, they update the Z complex register with the values present in X,Y upon the function execution. This behavior is common across all 41Z functions.

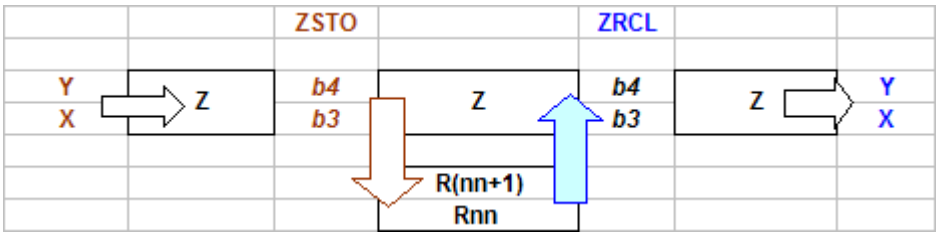
ZVIEW _ _	**Displays Complex Register value**		Prompting function
Z<> _ _	**Exchanges Z and complex register**		Prompting function

Like its real counterparts, these functions view or exchange the content of the complex stack level **Z** with that of the complex storage register given as its argument. Two standard storage registers are used, as per the above description.



ZRCL _ _	**Recall from Complex Register**	**Does Stack lift**	Prompting function
ZSTO _ _	**Store in Complex Register**		Prompting function

Like their real counterparts, these functions are used to Recall or store the complex number in Z from or into the complex register which number is specified as the function's argument. In fact two (real) storage registers are used, one for the imaginary part and another for the real part. This means that ZRnn corresponds to the real storage registers R2nn and R(2nn+1).



ZRCL will perform complex stack lift upon recalling the contents of the memory registers to the Z stack level. Also note that, following the 41Z convention, **ZSTO** will overwrite the Z level with the contents of X,Y if these were not the same. This allows walk-up complex data entering.

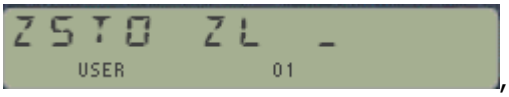
These functions are ***fully programmable***. When in program mode (either running or SST execution), the index input is taken from the following program line after the function. For this reason they are sometimes called *non-merged* functions. In fact, the number denoting the argument can have any combination of leading zeroes (like 001, 01, 1 all resulting in the same). Moreover, when the argument is zero then such index line can be omitted if any non-numeric line follows the function. This saves bytes and makes programs more legible.

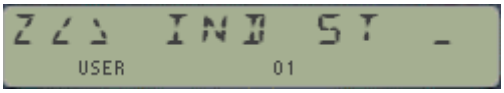
The original implementation was written by W. Doug Wilder, and it was almost as powerful and convenient as the one used by the HEPAX module for its own multi-function groups. I enhanced it further with an automated parameter input feature: when entered into a program, the index input will be added automatically in a second program line by the function.

Similar to the real counterparts, keys on the first two rows can be used as *shortcut for indexes 1-10*.

Note that ***indirect addressing is also supported*** (say **ZRCL IND _ _**) pressing the SHIFT key. Also note that in the Deluxe edition of the 41Z, their logic fully supports the use of the complex stack registers (i.e. **ZRCL ZL _ followed by a Z-stack level: {U, V, W, Z, and S}**) pressing the RADIX key; as well as the combination of both indirect and stack addressing (i.e. **ZRCL IND ST _ _ followed by a REAL stack / data register number**) sequentially pressing the SHIFT and RADIX keys. This extends the model of the native calculator functions to the complex data registers, where obviously an indirect pointer is always a real number by definition.

For example:



,


Where the left prompt will only allow for one of the five compelx Z-Stack levels letters, and the right prompt will allow for any of the 16 choices available as real stack (including the synthetic registers as well - be careful with those!).

Note that as of revision "O" of the Library#4 module, in program mode the argument entered by the function will be automatically entered in the second program step for the IND, ZL, and IND ST cases. In fact the *indirect addressing is nothing more that adding 128 to the address*, (or 0x80 Hex) thus it is handled by simply adding such factor to the index in the prompt line. Similarly, by adding 112 (or 0x70 Hex) for complex Z-stack levels, or the addition of both 0x80- and 0x70 (i.e. 240) for the IND ST combination.

Lastly, a NONEXISTENT message will be shown if the storage register pointed at is not available in main memory. Registers can be made available using the **SIZE** function of the calculator.

Note for advanced users: Pressing the EEX key will also activate the prompt-lengthener adding a third field to the prompt. This is of limited usability since for Complex registers it would require setting a real SIZE above 200 in the calculator.

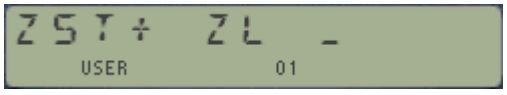
4.2. ZSTO/ZRCL Math function groups.

Function	Name		Comments
ZST+ _ _	**Store Adding to ZReg**		Prompting function
ZST- _ _	**Store subtracting from ZReg**		Prompting function
ZST* _ _	**Store multiplying to ZReg**		Prompting function
ZST/ _ _	**Store Dividing by ZReg**		Prompting function
ZRC+ _ _	**Add ZReg to Z**		Prompting function
ZRC- _ _	**Subtract ZReg from Z**		Prompting function
ZRC* _ _	**Multiply Z by ZReg**		Prompting function
ZRC/ _ _	**Divide Z by ZReg**		Prompting function

One of the newest additions to the 41Z function set.- allow storage and recall math in a concise format, saving bytes and programming steps in FOCAL programs. Their equivalence with standard functions would have to be done using four steps, and disturbing the Complex Stack as follows:

1.- ZENTER^,
2.- Z<>(nn)
3.- MATH (Z+, Z-, Z*, Z/)
4.- Z<>(nn)

With the support of Z-stack registers and INDirection it is possible to use the same shortcuts and conviniencies as there are available for the real case in the standard calculator. For example to multiply a number by two you use **ZST+**, RADIX, "Z" :



Which expects a letter representing the complex stack. i.e. {**Z**, **W**, **U**, **V**}, and "**S**" for LastZ.

Complex Stack manipulation is now simply a matter of using **Z<>** _ _ with the corresponding Z-stack level letter in the RADIX prompts. Not to be confused with the stack level# input required by the function **Z<>ST** _ _, which only allows decimal values between 1-5 as valid entries. Similar but not the same – in particular when if comes to INDirection.

These functions are fully programmable using the same non-merged technique described in the previous page for the standard cases. Like them, the argument is automatically entered by the function as a second program line. The same considerations apply for Z-Stack and indirect registers, which are automagically entered in the non-merged program step.

The RCL/STO Math functions can be accessed directly from the **ZRCL** and **ZSTO** prompts by pressing the corresponding arithmetic key. In fact, you can "navigate" you way about all the choices between the three memory access functions **ZRCL**, **ZSTO** and **Z<>** as well as their arithmetic extensions (excluding Z<>) simply by pressing any of the appropriate keys during the prompts of any of them.

5. Complex Math.

Complex numbers are much more than a simple extension of the real numbers into two dimensions. The Complex Plane is a mathematical domain with well-defined, own properties and singularities, and it isn't in the scope of this manual to treat all its fundamental properties. On occasions there will be a short discussion for a few functions (notably the logarithms!), and some analogies will be made to their geometric equivalences, but it is assumed throughout this manual that the user has a good understanding of complex numbers and their properties.

5.1. Arithmetic and Simple Math.

Table-5.1:- Arithmetic functions.

Index	Function	Formula	Description
1	**Z+**	Z=w+z	Complex addition
2	**Z-**	Z=w-z	Complex subtraction
3	**Z***	Z=w*z	Complex multiplication
4	**Z/**	Z=w/z	Complex division
5a	**ZINV**	Z=1/z	Complex inversion, direct formula
5b	**1/Z**	Z=1/r e^(-iArg)	Complex inversion, uses TOPOL
6	**ZDBL**	z=2*z	Sub-function. Can be replaced with: 2, ST* Z, *
7	**ZHALF**	z= z/2	Sub-function. Can be replaced with: 2. ST/ Z, /
8	**ZRND**	Z=rounded(z)	Rounds Z to display settings precision
9	**ZINT**	Z=Int(z)	Takes integer part for Re(z) and Im(z)
10	**ZFRC**	Z=Frc(z)	Takes fractional part for Re(z) and Im(z)
11	**ZPI***	Z=z□	Simple multiplication by pi

Here's a description of the individual functions within this group.

Z+	**Complex addition**	Z=w+z	Does LastZ
Z-	**Complex subtraction**	Z=w-z	Does LastZ
Z*	**Complex multiplication**	Z=w*z	Does LastZ
Z/	**Complex division**	Z=w/z	Does LastZ

Complex arithmetic using the RPN scheme, with the first number stored in the **W** stack level and the second in the **Z** stack level. The result is stored in the **Z** level, the complex stack drops (duplicating **U** into **V**), and the previous contents of **Z** is saved in the LastZ register.

ZINV	**Direct Complex inversion**	Z=1/z	Does LastZ
1/Z	**Uses POLAR conversion**	Z=1/r e^(-iArg)	Does LastZ

Calculates the reciprocal of the complex number stored in **Z**. The result is saved in **Z** and the original argument saved in the LastZ register. Of these two the direct method is faster and of comparable accuracy – thus it's the preferred one, as well as the one used as subroutine for other functions.

This function would be equivalent to a particular case of **Z/**, where w=1+0j, and not using the stack level **W**. Note however that **Z/** implementation is not based on the **ZINV** algorithm [that is, making use of the fact that : w/z = w * (1/z)], but based directly on the real and imaginary parts of both arguments.

Example. Calculate z/z using **ZINV** for z=i

We'll use the direct data entry, starting w/ the imaginary part:

1, ENTER^, 0, **ZINV**	-> 0-j1
LASTZ	-> 0+j1
Z*	-> 1+j0

Note that *integer numbers are displayed without decimal zeroes,* simplifying the visual display of the complex numbers.

ZDBL	**Doubles Z**	Z=2*z	Does LastZ
ZHALF	**Halves Z**	Z=z/2	Does LastZ

These two sub-functions are provided to save stack level usage and programming efficiency. The same result can also be accomplished using their generic forms (like **Z*** and **Z/**, with *w*=2+0j), but the shortcuts are faster and simpler to use.

Example. Taken from the HP-41 Advantage manual, page 97.

Calculate: $z_1/(z_2+z_3)$; for: $z_1=(23+13i)$; $z_2=(-2+i)$, and $z_3=(4-3i)$

If the complex stack were limited to 2 levels deep, we would need to calculate the inverse of the denominator and multiply it by the numerator, but using the 4-level deep complex stack there's no need to resort to that workaround. We can do as follows:

13, ENTER, 23, **ZENTER^**	-> 23+j13
1, ENTER^, 2, CHS, **ZENTER^**	-> -2+j1
3, CHS, ENTER^, 4, **Z+**	-> 2(1-j)
Z/	-> 2,500+j9

Note that 41Z *automatically takes common factor when appropriate*, and that integer numbers are displayed without decimal zeroes to simplify the visuals display of the complex numbers. Non-integers are displayed using the current decimal settings, but of course full precision (that is 9 decimal places) is always used for the calculations (except in the rounding functions).

ZRND	**Rounds Complex number**	Z=Rounded(z)	Does LastZ
ZINT	**Takes integer parts**	Z=Int[Re(z)+jInt[Im(z)	Does LastZ
ZFRC	**Takes Fractional parts**	Z=Frc[Re(z)+jFrc[Im(z)	Does LastZ

These functions will round, take integer part or fractional part both the real and imaginary parts of the complex number in **Z.** The rounding is done according to the current decimal places specified by the display settings.

ZPI*	**Multiplies by pi**	Z=⍰*z	Does LastZ

Simple multiplication by pi, used as a shortcut in the Bessel FOCAL programs. Has better accuracy than the FOCAL method, as it used internal 13-digit math.

5.2. Exponential and powers that be.

Table-5.2: Exponential group.

Index	Function	Formula	Description
1a	**ZEXP**	Z=REC(e^x, y)	Complex exponential (method one)
1b	**E^Z**	See below	Complex Exponential (method two)
2	**Z^2**	Z=REC(r^2, 2□)	Complex square
3a	**ZSQRT**	Algebraic Formula	Principal value of complex square root
3b	**SQRTZ**	Z=REC(r^1/2, □/2)	Principal value of complex square root
4	**W^Z**	Z=e^z*Ln(w)	Complex to complex Power
5	**W^1/Z**	Z=e^1/z*Ln(w)	Complex to reciprocal complex Power
6	**X^Z**	Z=e^z*Ln(x)	Real to complex power
7	**X^1/Z**	Z=e^z*Ln(x)	Real to reciprocal complex power
8	**Z^X**	Z=e^x*Ln(z)	Complex to real Power
9	**Z^1/X**	Z=e^1/x*Ln(z)	Complex to reciprocal real Power
10	**ZALOG**	Z=e^z*Ln(10)	Complex decimal power
11	**NXTRTN**	Z=z*e^j 2□/N	Next value of complex nth. Root

Looking at the above formula table it's easy to realize the importance of the exponential and logarithmic functions, as they are used to derive many of the other functions in the 41Z module. It is therefore important to define them properly and implement them in an efficient way.

The 41Z module includes two different ways to calculate the complex exponential function. The first one is based on the trigonometric expressions, and the second one uses the built-in polar to rectangular routines, which have enough precision in the majority of practical cases. The first method is slightly more precise but takes longer computation time.

ZEXP	**Complex Exponential**	Z=REC(e^x, y)	Does LastZ
E^Z	**Complex Exponential**	Trigonometric	Does LastZ

One could have used the rectangular expressions to calculate the result, as follows:

e^z = e^x * (cos y + i sin y), thus: Re(z) = e^(x) * cos y ; and: Im(z) = e^(x) * sin y

and this is how the sub-function **E^Z** has been programmed. It is however more efficient (albeit slightly less precise) to work in polar form, as follows:

since z= x+iy, then e^z = e^(x+iy) = e^x * e^iy,

and to calculate the final result we only need to convert the above number to rectangular form.

Example.- Calculate exp(z^{-2}), for z=(1+i)

1, ENTER^, **ZENTER^**	-> 1(1+j)
2, CHS, **Z^X**	-> 0 – j 0,500
ZEXP	-> 0,878 – j 0,479

Another method using **W^Z** and the complex keypad function (**ZREAL^**):

1, ENTER^, **ZENTER^**	-> 1(1+j)
2, CHS, **ZREAL^**	-> -2 + j 0
W^Z, ZEXP	-> 0,878 – j 0,479

or alternatively, this shorter and more efficient way: (leaves **W** undisturbed)

1, ENTER^, **Z^2, ZINV, ZEXP** -> 0,878 – j 0,479

Note how this last method doesn't require using **ZENTER^** to terminate the data input sequence, as the execution of monadic functions will automatically synchronize the complex stack level Z with the contents of the real X,Y registers.

Z^2	**Complex square**	Z=REC(r^2, 2⍰)	Does LastZ
ZSQRT	**Complex square root**	Algebraic Formula	Does LastZ
SQRTZ	**Complex square root**	Z=REC(r^1/2, ⍰/2)	Does LastZ

Two particular cases also where working in polar form yields more effective handling. Consider that:

Z^2 = |z|^2 * e^2iα, and:
Sqrt(z) = z^1/2 = Sqrt(|z|) * e^iα, where α=Arg(z),

It is then simpler first converting the complex number to its polar form, and then apply the individual operations upon its constituents, followed by a final conversion back to the rectangular form.

Note that this implementation of **ZSQRT** only offers one of the two existing values for the square root of a given complex number. The other value is easily obtained as its opposite, thus the sum of both square roots is always zero.

Such isn't exclusive to complex arguments, for the same occurs in the real domain – where there are always 2 values, x1 and –x1, that satisfy the equation SQRT[(x1)^2].

As with other multi-valued functions, the returned value is called the *principal value* of the function. See section 6 ahead for a more extensive treatment of this problem.

W^Z	**Complex to complex Power**	Z=e^[z*Ln(w)]	Does LastZ
W^1/Z	**Complex to reciprocal Power**	Z=e^[Ln(w)/z]	Does LastZ

The most generic form of all power functions, calculated using the expressions:

w^z = exp[z*Ln(w)], and
w^1/z = exp[Ln(w) / z]

The second function is a more convenient way to handle the reciprocal power, but it's obviously identical to the combination **ZINV, W^Z**.

Example: calculate the inverse of the complex number 1+2i using **W^Z**:- Then obtain its reciprocal using **ZINV** to verify the calculations.

2, ENTER^, 1, **ZENTER^**	number stored in level **W** (also as: 1, ENTER^, 2, **ZTRP**)
0, ENTER^, -1	exponent –1 stored in level **Z** (also as: -1, ENTER^, 0, **ZTRP**)
W^Z	result: 0,200 – j 0,400
ZINV	result: 1,000 + j 2

Note that the final result isn't exact – as the decimal zeroes in the real part indicate there's a loss of precision in the calculations.

Z^X	**Complex to real power**	Z=e^[x*Ln(z)]	Does LastZ
Z^1/X	**Complex to reciprocal real**	Z=e^[Ln(z)/x]	Does LastZ
X^Z	**Real to complex power**	Z=e^[z*Ln(x)]	Does LastZ
X^1/Z	**Real to reciprocal complex**	Z=e^[1/z*Ln(x)]	Does LastZ
ZALOG	**10 to complex power**	Z=e^[z*Ln(10)]	Does LastZ

These five functions are calculated as particular examples of the generic case W^Z. Their advantage is a faster data entry (not requiring inputting the zero value) and a better accuracy in the results

Z^1/X is identical to: 1/X, **Z^X**
X^1/Z is identical to: RDN, **ZINV**, R^, **X^Z**

Data entry is different for hybrid functions, with mixed complex and real arguments. As a rule, the second argument is stored into its corresponding stack register, as follows:

- x into the real stack register X for **Z^X** and **Z^1/X**
- *z* into the complex stack register **Z** for **X^Z** and **X^1/Z**

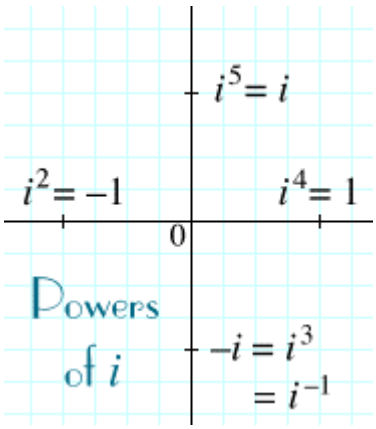
The first argument needs to be input first, since this is an RPN implementation.

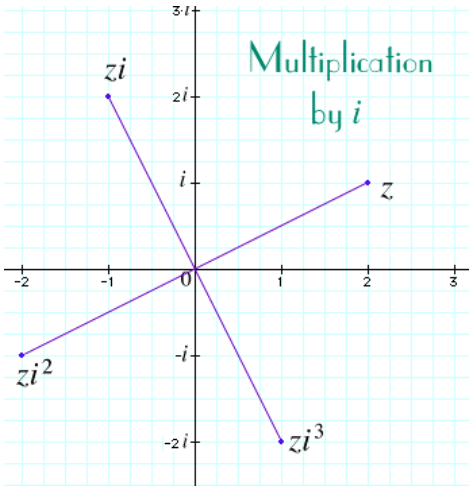
Because **ZALOG** is a monadic function, it expects z in the stack level **Z**, and thus it doesn't disturb the complex stack.

Example: Calculate (1+2i)^3 and 3^(1+2i)

2, ENTER^, 1, **ZENTER^**, 3, **Z^X** results: (1+2i)^3 = -11 – j 2
2, ENTER^, 1, **ZENTER^**, 3, **X^Z** results: 3^(1+2i) = -1,759 + j 2,430

Example: Verify the powers of the imaginary unit, as per the picture below.- You can use either **Z^X**, with z=(0+i) and x=1,2,3,4,5; or alternatively **W^Z**, with w=(0+i) and z=(1+0i), (2+0i), (3+0i), etc.





This keystroke sequence will quickly address the even powers:

0, ENTER^, 1, **ZTRP** -> 0 + j1 i
Z^2 -> -1 + j0 $i^2 = -1$
Z^2 -> 1 + J0 $i^4 = 1$

Whilst this will take care of the rest (and also in general):

0, ENTER^, 1, **ZTRP**	-> 0 + j1	i
3, **Z^X**	-> 0 - j1	$i^3 = -i$
LASTZ	-> 0 + j1	
5, **Z^X**	-> 0 + j1	$i^5 = i$

Note in this example that for enhanced usability **Z^X** stores the original argument in the LastZ register, even though it wasn't strictly located in the **Z** level of the complex stack. The same behavior is implemented in **X^Z**.

Alternatively, using **W^Z** and **ZREPL**:

1, ENTER^, 0, **ZREPL**	-> 0 + j1	i
0, ENTER^, 2, **W^Z**	-> -1 + j0	$i^2 = -1$
ZRDN	-> 0 + j1	i
0, ENTER^, 3, **W^Z**	-> 0 - j1	$i^3 = -i$
ZRDN	-> 0 + j1	i
0, ENTER^, 4, **W^Z**	-> 1 + j0	$i^4 = 1$
ZRDN	-> 0 + j1	i
0, ENTER^, 5, **W^Z**	-> 0 + j1	$i^5 = i$

Examples.- Calculate the value of: z = 2^1/(1+i); and z=(1+i)^1/2

These two have a very similar key sequence, but they have different meaning:

Solution: 1, ENTER^, ENTER^, 2, **X^1/Z** -> 1,330 – J0,480
Solution: 1, ENTER^, ENTER^, 2, **Z^1/X** -> 1,099 + j0,455

NXTNRT	**Next value of Nth. Root**	Z=z0*e^j 2⍰/N	z0 is the principal value

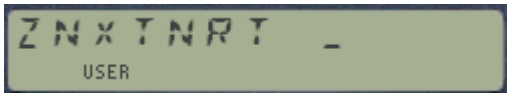
In its general form, the solution to the Nth. Root in the complex plane admits multiple solutions. This is because of its logarithmic nature, since the logarithm is a multi-valued function (see discussion in next section).

$$Z^{1/N} = e^{[Ln(z)/N]} = e^{[Ln(|z|)+i(\alpha+2\pi)]/N} = e^{[Ln(|z|)+i\alpha]/N} * e^{j\,2\pi/N}$$

From this we derive the general expression**:** **Next(z^1/N) = z^1/N * e^(j 2 π /N)**

thus there are N different Nth. Roots, all separated by (2π over N). See the geometric interpretation on section 7 ahead for further discussion on this.

When executed in a program or RUN mode, data entry for this function expects N in the X register, and *z* in the **Z** complex stack level. However when the Complex Keyboard shortcut is used, *the index N is prompted as part of the entry sequence* – a much more convenient way.



Shortcut: Z, Z, SHIFT, SQRT

Example:- Calculate the *two* square roots of 1.

0, ENTER^, 1, **ZENTER^,** 2, **Z^1/X**	-> 1 + j 0
2, **NXTNRT** (plus **ZRND**)	-> -1 + j 0

Note that the previous root is temporarily stored in the LastZ register:

LASTZ	-> 1 + j 0 (previous root)

See section 9 for a general application program to calculate the n different Nth. Roots of a complex number

Example.- Calculate the *three* cubic roots of 8.

Using "direct" data entering: [Im(z), ENTER^, Re(z)]

0, ENTER^, 8, **ZENTER^**, 3, **Z^1/X**	-> 2 + j 0
NXTNRT _ 3	-> -1,000 + j 1,732
NXTNRT_ 3	-> -1,000 – j 1,732

Note: for this example use the *Complex Keyboard* **ΣZL** to execute **NXTNRT**, as follows:

Z, **Z**, SHIFT, SQRT, and then input 3 at the last prompt.

Example: Calculate both quadratic roots of 1 + 2i.

2, ENTER^, 1, **ZSQRT**	gives the first root: z= 1,272 + j 0,786
NXTNRT_ 2	gives the second root: z=-1,272 – j 0,786
NXTNRT_ 2	reverts to the first, principal value, of the root.

This verifies that both roots are in fact on the same straight line, separated 180 degrees from each other and with the same module.

Example: Calculate the three cubic roots of 1 + 2i.

2, ENTER^, 1, **ZENTER^**	inputs z in the complex stack level **Z**
3, 1/X, **Z^X**	gives the main root: z= 1,220 + j 0,472
NXTNRT_ 3	gives the second root: z=-1,018 + j 0,82
NXTNRT_ 3	give the third and last: z=-0,201 – j 1,292

In the next section we'll discuss the logarithm in the complex plane, a very insightful and indeed interesting case study of the multi-valued functions.

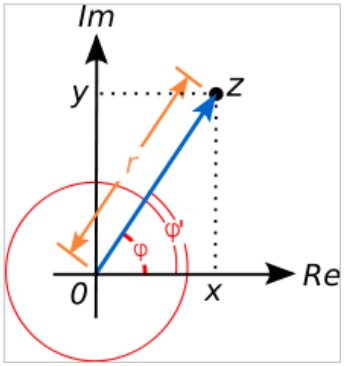
5.3. Complex Logarithm.

Table-x: Logarithm group.

Index	Function	Formula	Description
1	**ZLN**	Z=Ln|z|+i	Principal value of natural logarithm
2	**ZLOG**	Z=Ln(z)/Ln10	Principal value of decimal logarithm
3	**ZWLOG**	Z=Ln(z)/Ln(w)	Base-w logarithm of z
4	**NXTLN**	Z=z+2 j	Next value of natural logarithm

The first thing to say is that a rigorous definition of the logarithm in the complex plane requires that its domain be restricted, for if we defined it valid in all the plane, such function wouldn't be continuous, and thus neither *holomorfic* (or expressible as series of power functions).

This can be seen intuitively if we consider that:



Since: $z = |z|*e^{i\rho}$, then:
$\ln z = \ln |z| + \ln (e^{i\rho}) = \ln(|z|) + i\rho$

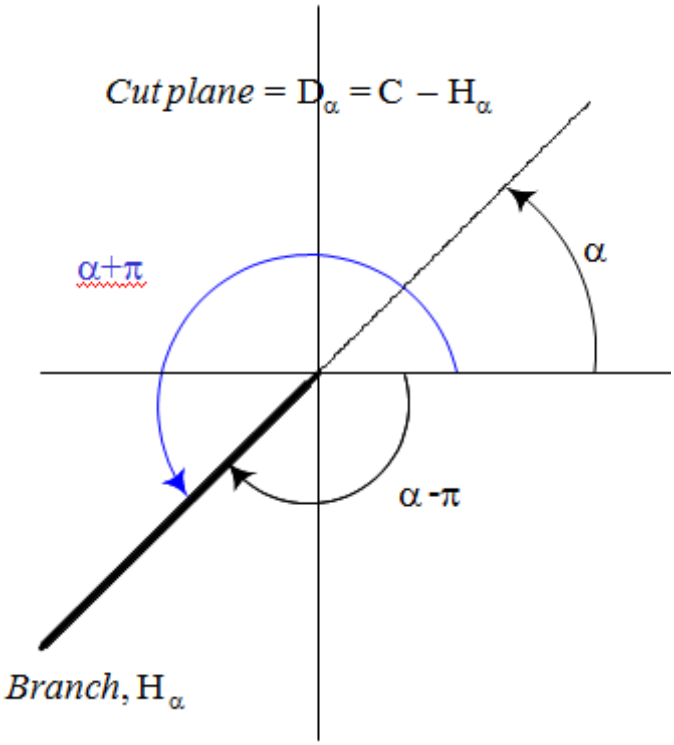
But also
$z = |z|*e^{i} (\rho+2\pi) = |z|*e^{i} (\rho +4\pi)=\ldots. = |z|*e^{i} (\rho +2\pi n)$

Then we'd equally have multiple values of its logarithm, as follows:
$\ln(z) = \ln(|z|) + i\rho = \ln(|z|)+i (\rho+2\pi) = \ldots.$ Or generally:

$\ln z = \ln|z|+i (\rho+2\pi n)$; where n is a natural number.

To deal with this multi-valued nature of the function, mathematicians define the different ***branches of the complex*** *logarithm,* - $\log_\alpha$ – as the single one and only logarithm which argument is comprised between $(\alpha -\pi)$ and $(\alpha +\pi)$, thus within the open interval $]\ \alpha -\pi, \alpha+\pi\ [$

Its domain isn't the whole complex plane, but it excludes a semi-straight line, centered at the origin, that forms an angle α with the real axis, as shown in the picture. Such set is called the "***torn" or cut complex plane*** *at angle* α*".* Thus the principal value of the logarithm really should be called Log_0, as it tears (or cuts) the complex plane by the real negative semi-axis, or otherwise α =0. This means it is *NOT defined* for any negative numbers, and when those need to be subject of its application, a different cut should be chosen.



Therefore all arguments should be comprised between 180 and –180 degrees, as it would correspond to this definition of "Log_0".

In practicality, the values calculated by **ZLN** always lie within this interval, since they use the internal routines of the calculator, [TOPOL] and [TOREC].

The angle α should not be confused with the base of the logarithm, which is always the number e – that is, there are natural logarithms. (See http://en.wikipedia.org/wiki/Branch_point for a more rigorous description of this subject).

After this theoretical discussion, let's see the functions from the 41Z module:-

ZLN	Natural logarithm	Z=Ln|z|+i⍰	Does LastZ

Calculates the principal value of the natural logarithm, using the expression:

Ln z = Ln|z| + i.γ, where γ = Arg(z) belongs to]-π, π]

Example: check that: z=Ln(e^z), for z=(1+i) and z=(2+4i)

1, ENTER^, **ZEXP, ZLN** -> 1,000 + j 1,000
4, ENTER^, 2, **ZEXP, ZLN** -> 2 – j 2,283

How do you explain the last result? Is it correct? Try executing **NXTLN** (see below) on it…

NXTLN -> 2 + j 4,000 - that's more like it!

ZLOG	Decimal logarithm	Z=Ln(z)/Ln10	Does LastZ

Calculates the principal value of the decimal logarithm using the expression:

Log z = Ln z / Ln(10)

Example: check that: z=Log(10^z), for z=(1+i) and z=(2+4i)

1, ENTER^, **ZALOG, ZLOG** -> 1(1+j)
4, ENTER^, 2, **ZALOG, ZLOG** -> 2 + j 1,271

How do you explain the last result? Is it correct? Have you found a bug on the 41Z?

ZWLOG	Base-W Logarithm	Z=Ln(z)/Ln(w)	Does LastZ

General case of ZLOG, which has w=10. This is a dual function,

Log z = Ln z / Ln w

NXTLN	Next Natural logarithm	Z=z0+2⍰ j	z0 is the principal value

Calculates the next value of the natural logarithm, using the expression:

Next(Ln z) = Ln(z) + 2π j

So the different logarithms are "separated" a distance of value 2π in their imaginary parts. This works both "going up" as well as "going down", thus each time **NXTLN** is executed two values are calculated and placed in complex levels Z and W. You can use **Z<>W** to see them both.

6. Complex Geometry.

The next set of functions admits a geometrical interpretation for their results. Perhaps one of the earliest ways to approach the complex numbers was with the analogy where the real and imaginary parts are equivalent to the two coordinates in a geometric plane.

Table-6.1: Complex geometric group.

Index	Function	Formula	Description
1	**ZMOD**	|z|=SQR(x^2+y^2)	Module or magnitude of a complex number
2	**ZARG**	□ =ATAN(y/x)	Phase or angle of a complex number
3a	**ZNEG**	Z=-z	Opposite of a complex number
3b	**ZCHSX**	Z=(-1)^x * z	Opposite (by X) of a complex number
4	**ZCONJ**	Z=x-y j	Conjugated of a complex number
5	**ZSIGN**	Z=z/|z|	Sign of a complex number
6	**ZNORM**	Z=|z|^2	Norm of a complex number
7	**Z*I**	Z=z*i	Rotates z 90 degrees counter clockwise
8	**Z/I**	Z=z/i	Rotates z 90 degrees clockwise

In fact, various complex operations admit a geometrical interpretation. An excellent reference source for this can be found at the following URL: http://www.clarku.edu/~djoyce/complex.

Let's see the functions in detail.

ZMOD	**Module of z**	|z|=SQR(x^2+y^2)	Does LastZ
ZARG	**Argument of z**	□=ATAN(y/x)	Does LastZ

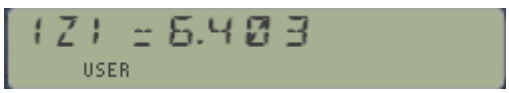
This pair of functions calculates the module (or magnitude) and the argument (or angle) of a complex number, given by the well-known expressions:

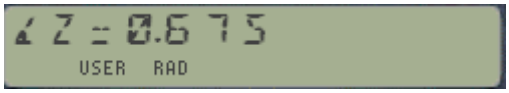
$$|z| = SQR(x^2 + y^2)$$
$$\alpha = ATAN(y/x)$$

Since they use the internal [TOPOL] routine (like R-P does), the argument will always be given between 180 and –180 degrees (or equivalent in the selected angular mode).

The result is saved in the complex **Z** register, and the real X,Y stack levels – as a complex number with zero imaginary part. The original complex number is stored in the Last**Z** register. The other complex stack levels **W, V, U** aren't disturbed.

These functions display a meaningful description when used in run mode, as can be seen in the pictures below, for z= 5+4 j and RAD mode.



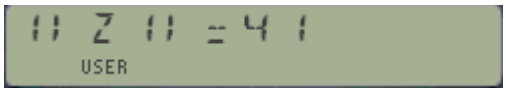


ZNORM	Norm of z	||Z||=|z|^2	Does LastZ

This function calculates the norm of a complex number, also known as the square of its module"

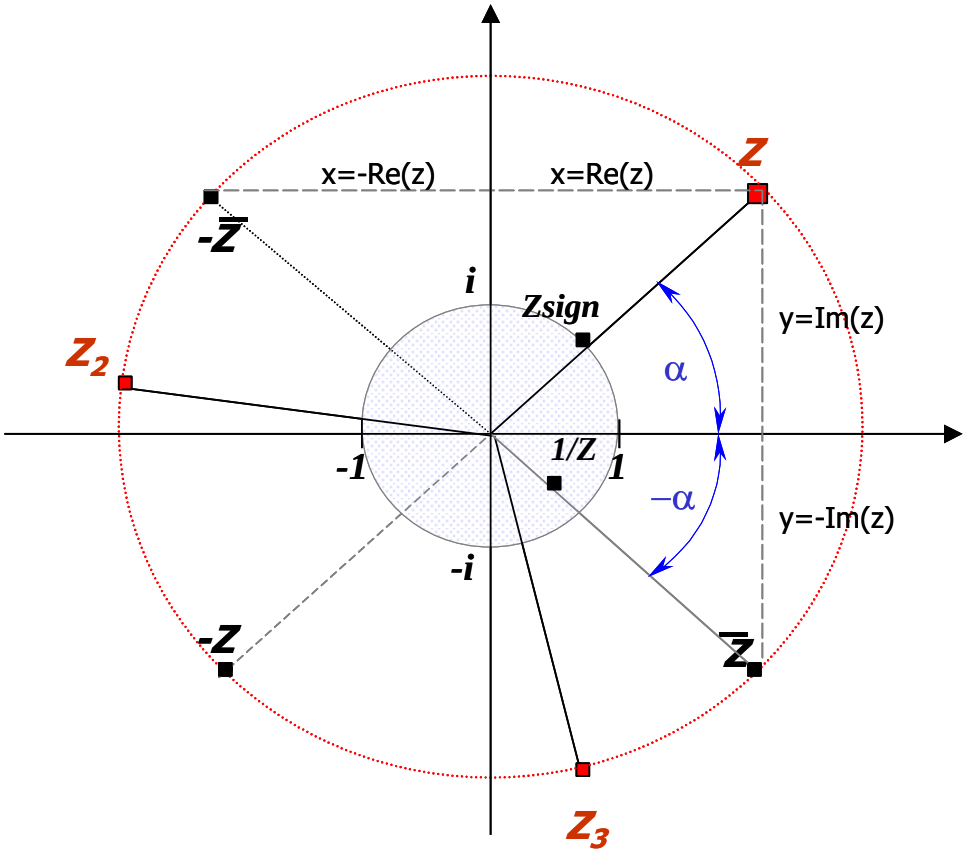
$$||z|| = |z|^2 \; ; \;\; \text{thus:} \;\; Znorm = x^2 + y^2$$

When executed in run mode, the display shows a meaningful representation for it, like in the example below, also for z = 4 + 5 j :



ZSIGN	Module of z	Z=z/|z|	Does LastZ

This function calculates the sign of a complex number. As an extension to the SIGN function for the real domain, it is a complex number with magnitude of one (i.e. located on the unit circle), that indicates the direction of the given original number. Thus obviously: Zsign = z / |z|



The figure above shows the unit circle and the relative position in the complex plane for the opposite (-z), conjugate (zc), and opposite conjugate (-zc) of a given number z.
Note that the inverse of z (1/z) will be located inside of the unit circle, and over the direction defined by the negative of its argument [-Arg(z)]

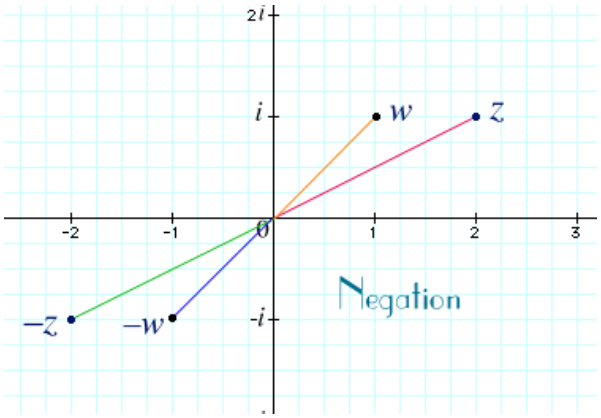
Note that if z happens to be a cubic root of another number (i.e. z^3), then the other two roots (z_2 and z_3) will have the same module and be located at 120 degrees from each other, on the red circle line.

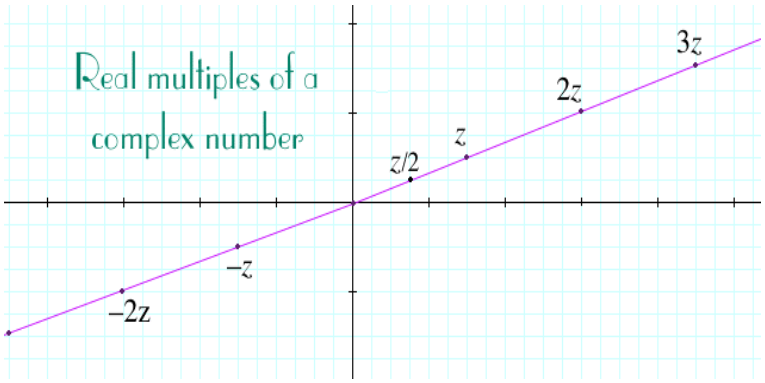
ZNEG	**Opposite of z**	Z=-z	Does LastZ
ZCHSX	**Opposite of z by X**	Z=(-1)^x * z	Does LastZ
ZCONJ	**Conjugate of z**	Z=x-y j	Does LastZ

This pair of functions calculate the opposite- or the multiple-opposite by (-1)^x – and the conjugate of a complex number z=x+y i, as follows:

$$-z = -x - y\,I, \quad \text{and} \quad z^* = x - y\,I$$

See the figure below for the geometric interpretation of **ZNEG** and multiplication by real numbers:





Z*I	**Multiply by i**	Z=z*i	Rotates z 90 deg ccw
Z/I	**Divide by i**	Z=z/i	Rotates z 90 deg cw

The main role of these two functions is as subroutines for the trigonometric set, and they are also provided for completion sake. Their geometric interpretation is a 90 degrees rotation of the complex number either clockwise or counter-clockwise respectively.

These functions are used as subroutines for several others, like the direct and inverse trigonometric. The dependencies between hyperbolic and trigonometric ultimately involves multiplication by i, which is really a matter of swapping the real and imaginary parts, with the appropriate sign change in each case.

6.2 Complex Comparisons.

The 41Z module includes a comprehensive set of comparison checks, based on the complex numbers themselves and their modules (for relative position in the complex plane). Checks for purely real or imaginary cases are also provided. The main utilization for these functions is in program mode, as conditional decisions under program control based on the different values.

Table 6.2. Complex comparisons function group.

Index	Function	Formula	Description
1	**Z=0?**	Is z=0?	Checks if z is zero
2	**Z#0?**	Is z#0?	Checks if z is not zero
3	**Z=I?**	Is z=i?	Checks if z is the imaginary unit
4	**Z=W?**	Is z=w?	Checks if z and w are the same
5	**Z=WR?**	Is z=w rounded?	Checks if rounded z and rounded w are the same
6	**Z#W?**	Is z#w?	Checks if z and w are different
7	**ZUNIT?**	Is |z|=1?	Checks if z is on the unit circle
8	**ZIN?**	Is |z|<1?	Checks whether z is inside the unit circle
9	**ZOUT?**	Is |z|>1?	Checks whether z is outside the unit circle
10	**ZREAL?**	Is z a real number?	Checks whether Im(z)=0
11	**ZIMAG?**	Is z true imaginary?	Checks whether Re(z)=0
12	**ZINT?**	Is z true integer?	Checks whether Im(z)=0 and FRC[Re(z)]=0
13	**ZGSS?**	Is z Gaussian?	Checks whether Re(z) and Im(z) are both integers
14	**ZQUAD**	Shows Quad# msg.	Sets corresponding User Flag, clears others.

It's well know that, contrary to real numbers, the complex plane isn't an ordered domain. Thus we can't establish ordered relationships between two complex numbers like they are done with real ones (like x>y, x<y?, etc.).

There are however a few important cases that can also be used with complex numbers, as defined by the following functions.- As it is standard, they respond to the "*do if true*" logic, skipping the next program line when false.

Z=W?	**Compares z with w**	Are they equal?	
Z#W?	**Compares z with w**	Are they different?	
Z=WR?	**Compares z with w rounded**	Are they equal?	
Z=0?	**Compares z with zero**	Are they equal?	
Z#0?	**Compares z with zero**	Are they different?	
Z=I?	**Compares z with i**	Are they equal?	

The first two functions compare the contents of the **Z** and **W** stack levels, checking for equal values of both the real and imaginary parts.

z=w iff Re(z)=Re(w) and Im(z)=Im(w)

The third function, **Z=WR?** Will establish the comparison *on the rounded values of the four real numbers*, according to the current display settings on the calculator (i.e. number of decimal places shown). This is useful when programming iterative calculations involving conditional decisions.

Rnd(z) = Rnd(w) iff abs[Re(z)]=abs[Re(w)] and: abs(Im(z)] = abs[Im(w)]

The remaining three functions on the table are particular applications of the general cases, checking whether the **Z** complex stack level contains zero or the imaginary unit:

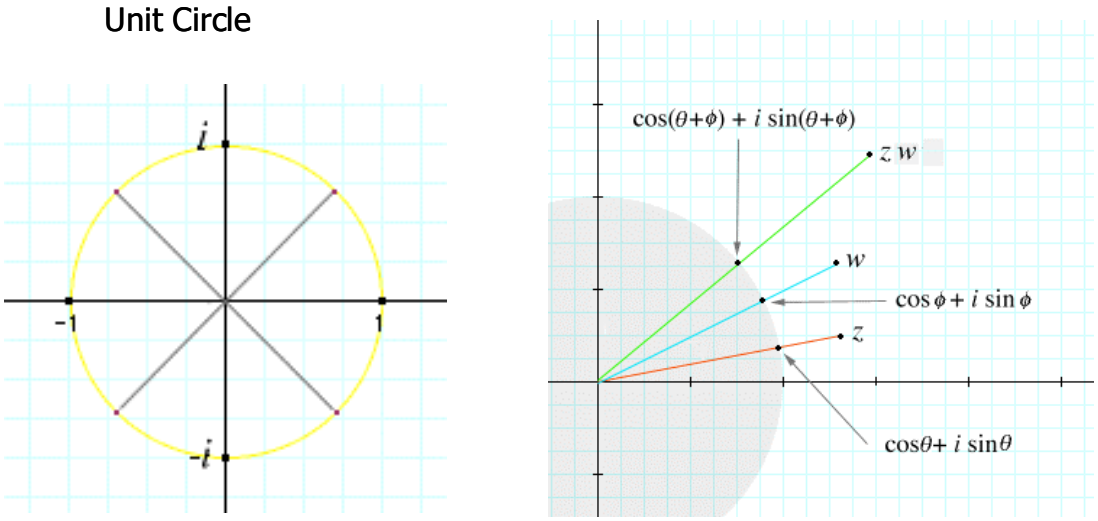
z=0 iff Re(z)=0 and Im(z)=0
z=i iff Re(z)=0 and Im(z)=1

Some of the inverse comparisons can be made by using standard functions, as follows:

- use **X#0?** To check for Z#0? Condition
- Use **X#0?** To check for Z#I? Condition

ZUNIT?	**Checks if z is on the unit circle**	|z|=1?	
ZIN?	**Checks if |z|<1**	|z|<1?	Sub-function
ZOUT?	**Checks if |z|>1**	|z|>1?	

These three functions base the comparison on the actual location of the complex number referred to the unit circle: inside of it, on it, or outside of it. The comparison is done using the number's modulus,



as a measure of the distance between the number and the origin.

Example: For z=4+5j , calculate its sign and verify that it's located on the unit circle:

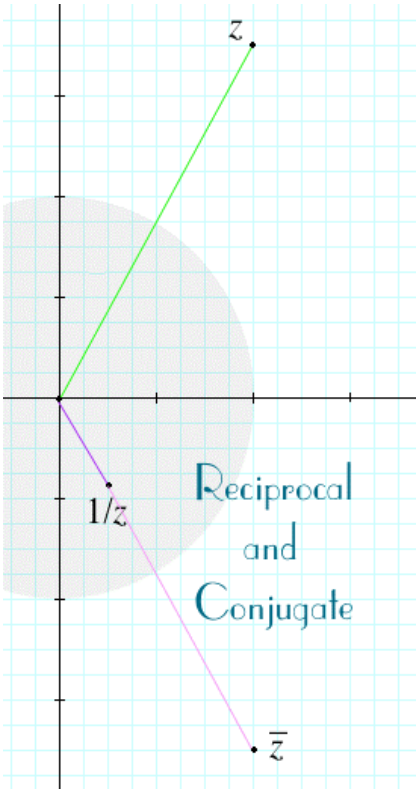
5, ENTER^, 4, **ZSIGN**, → result: Zsign = 0,625 + j 0,781
ZUNIT? → result: "YES"
DEG**, POLAR** → result: 1,00 < 51,34 (in degrees)

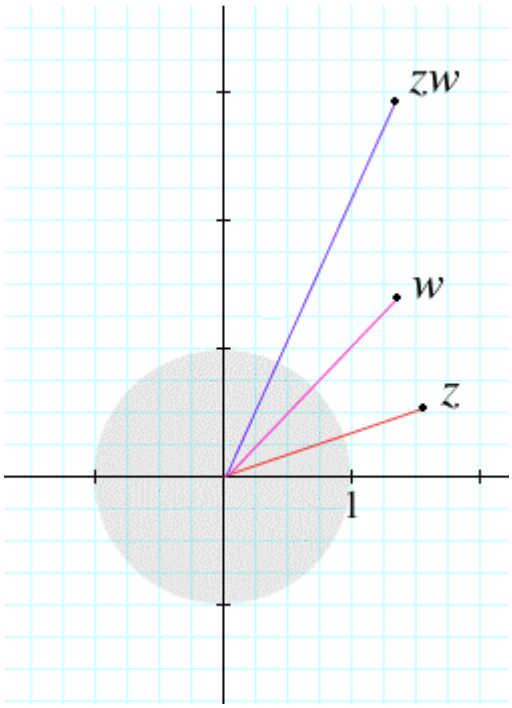
In program mode the behavior is ruled by the "do if true" rule, skipping the next line if false.

ZQUAD	**Returns Quadrant# for z**	Sets flag 1-2-3-4	Shows message

A new function to display the quadrant number (1 to 4) and set the user flag matching its value.

the message goes away in a few instants.





ZREAL?	**Checks if z is purely real**	Im(z)=0?	
ZIMAG?	**Checks if z is purely imaginary**	Re(z)=0?	
ZINT?	**Checks if z is an integer**	Im(z)=0, Re(z) integer	
ZGSS?	**Checks if z is Gaussian**	Both Re(z) and Im(z) integers	

The first two functions check whether the complex number is purely a real or imaginary number.

Do not mistake these comparison functions with the other pair, {**ZREAL** and **ZIMAG**}, which cause the number to change to become either real or imaginary – nor with {**ZREAL^** and **ZIMAG^**}, which are used to input complex numbers of the selected type based on the value stored in the real stack level X.

The third one extends the scope of ZREAL?, adding the condition of being a true integer number:

- **ZINT?** True means **ZREAL?** True, and FRC(Re(z))=0

Do not mistake it with **ZINT**, which causes the complex number to have no decimal figures in BOTH its real and imaginary parts – *therefore it's result not a Real number*!

ZINT? Is used in the FOCAL programs to calculate Bessel Function, as a quick an effective way to determine if the order is integer – which triggers different expressions for the formulas.

Like it occurs with any built-in comparison function, there's no action taken on the original number, which will remain unchanged.

7. Complex Trigonometry.

Table 7.1. Complex trigonometry function group.

Index	Function	Formula	Description
1	**ZSIN**	sin z = -i *sinh (iz)	Complex Sine
2	**ZCOS**	cos z = cosh (iz)	Complex Cosine
3	**ZTAN**	tan z = - i * tanh (iz)	Complex Tangent
4	**ZHSIN**	sinh z = 1/2 * [e^z – e^-z]	Complex Hyperbolic Sine
5	**ZHCOS**	cosh z = 1/2 * [e^z + e^-z]	Complex Hyperbolic Cosine
6	**ZHTAN**	tanh z = (e^z-e^-z)/(e^z+e^-z)	Complex Hyperbolic Tangent

And their inverses:

7	**ZASIN**	asin z = -i * asinh (iz)	Complex Inverse Sine
8	**ZACOS**	acos z = □/2 – asin z	Complex inverse Cosine
9	**ZATAN**	atan z = -i * atanh (iz)	Complex Inverse Tangent
10	**ZHASIN**	asinh z = Ln[z + SQ(z^2 + 1)]	Complex Inverse Hyperbolic Sine
11	**ZHACOS**	acosh z = Ln[z + SQ(z^2 – 1)]	Complex Inverse Hyperbolic Cosine
12	**ZHATAN**	atanh z = 1/2 * Ln[(1+z)/(1-z)]	Complex Inverse Hyperbolic Tangent

This section covers all the trigonometric and hyperbolic functions, providing the 41Z with a complete function set. In fact, their formulas would suggest that despite their distinct grouping, they are nothing more than particular examples of logarithm and exponential functions (kind of *"logarithms in disguise"*).

Their usage is simple: the argument is taken from the complex-**Z** level and *always* saved on the LastZ register. The result is placed on the complex-**Z** level. Levels **W, V, U** are preserved in all cases, including the more involved calculations with **ZTAN** and **ZATAN** (those with the devilish names), for which extensive use of scratch and temporary internal registers is made.

The formulas used in the 41Z are:

$$\sin z = -i \cdot \sinh(iz) \qquad \sinh z = \tfrac{1}{2}\cdot[e^{z} - e^{-z}]$$
$$\cos z = \cosh(iz) \qquad \cosh z = \tfrac{1}{2}\cdot[e^{z} + e^{-z}]$$
$$\tan z = -i \cdot \tanh(iz) \qquad \tanh z = (e^{z} - e^{-z})/(e^{z} + e^{-z})$$

$$\text{asin } z = -i \cdot \text{asinh}(iz) \qquad \text{asinh } z = \text{Ln}[z + \text{SQ}(z^2 + 1)]$$
$$\text{acos } z = \pi/2 - \text{asin } z \qquad \text{acosh } z = \text{Ln}[z + \text{SQ}(z^2 - 1)]$$
$$\text{atan } z = -i \cdot \text{atanh}(iz) \qquad \text{atanh } z = \tfrac{1}{2}\cdot\text{Ln}[(1+z)/(1-z)]$$

So we see that interestingly enough, the hyperbolic functions are used as the primary ones, also when the standard trigonometric functions are required. This could have also been done the other way around, with no particular reason why the actual implementation was chosen.

Example. Because of their logarithmic nature, also the inverse trigonometric and hyperbolic functions will be multi-valued. Write a routine to calculate all the multiple values of ASIN z.

```
01 LBL "ZASIN"      08 ZRCL 00      15 ZAVIEW
02 ZASIN            09 ZNEG         16 PSE
03 ZSTO 00          10 ZSTO 00      17 E
04 ZAVIEW           11 RCL 02       18 ST+ 02
05 E                12 PI           19 GTO 00
06 STO 02           13 *            20 END
07 LBL 00           14 +
```

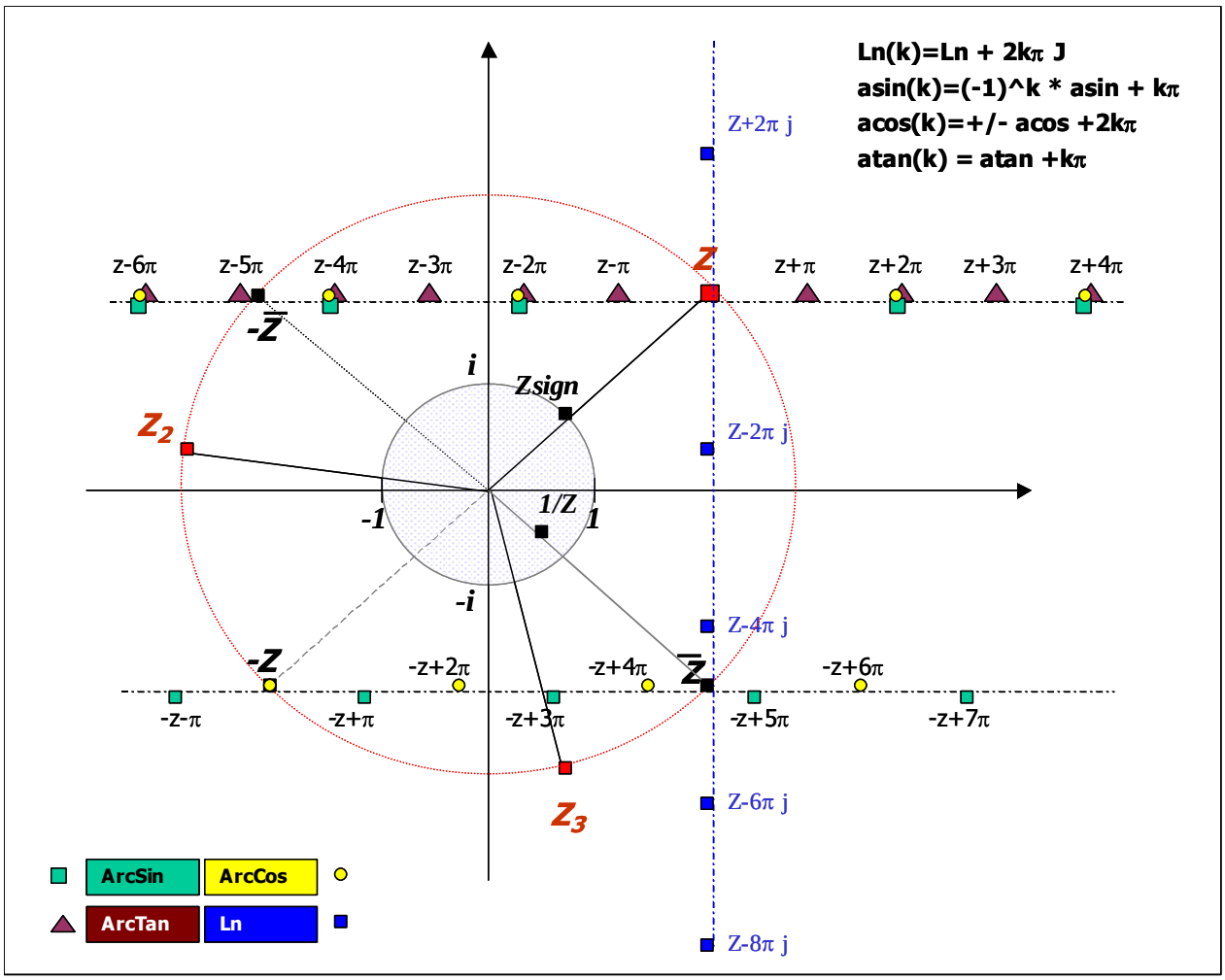
The 41Z module includes functions to calculate next values for complex ASIN, ACOS and ATAN, as follows: **NXTASN, NXTACS**, and **NXTATN**. Using the first one the program above changes to this very simplified way:

```
01  LBL "ZASIN2"      04  ZAVIEW      07  END
02  ZASIN              05  NXTASN
03  LBL 00             06  GTO 00
```

The key map is shown in the figure on the right, and can be accessed using:

- [Z], [SHIFT] for the direct ones, and
- [Z], [SHIFT], [SHIFT] for the inverses.

Using the general expressions we can obtain the multiple values of a given function from its principal value "**Z**" of a given function, as follows:



- the multiple values for ASIN(z) -in green squares- are placed on the two straight lines parallel to the x axis, y=Im[ASIN(z)] and y=–Im[ASIN(z)], and are separated at intervals of 2π length on each line.

- the multiple values for ACOS(z) –in yellow circles– are placed on the same two straight lines, and are separated at intervals of 2π length on each line.
- the multiple values for ATAN(z) –in brown triangles- are placed on the upper of those straight lines, separated at intervals of π length on it.
- the multiple values for Ln(z) –in blue squares- are placed on the vertical straight line x=Re[LN(z)], and separated at intervals of 2π length on it.
- the three different values for z^1/3 are placed in the circle r=|z|^1/3, and are separated at 120 degrees from each other (angular interval).

NXTASN	**Next Complex ASIN**		Does LastZ
NXTACS	**Next Complex ACOS**		Does LastZ
NXTATN	**Next Complex ATAN**		Does LastZ

Let z0 be the principal value of the corresponding inverse trigonometric function. Each of these three functions returns *two* values, z1 and z1' placed in complex stack levels **Z** and **W**. z1 will be shown if the function is executed in RUN mode. You can use **Z<>W** to see the value stored in **W** (that is, z1')

The NEXT values z and z1' are and given by the following recursion formulas:

Next ZASIN:

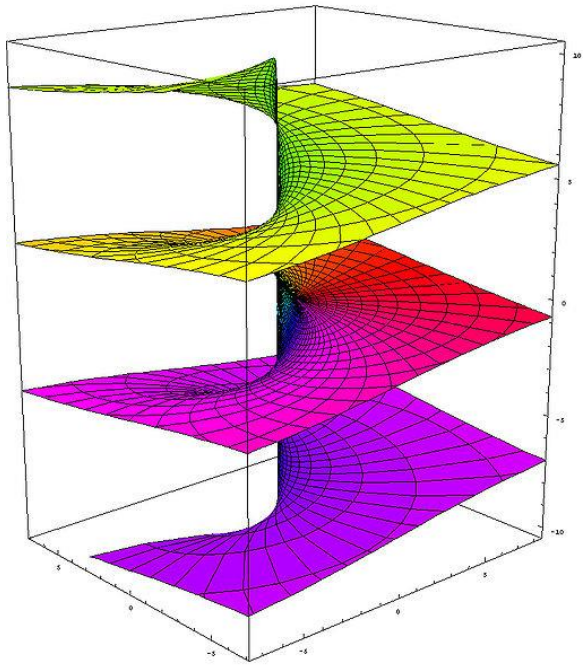
Z1 = Z0 + 2 pi
Z1'= -Z0 + pi

Next ZACOS:

Z1 = Z0+ 2 pi
Z1' = -Z0 + 2 pi

Next ZATAN:

Z1=Z0 + pi
Z1'= Z0 – pi



The figure on the right plots the multi-valued imaginary part of the complex logarithm function, which shows the branches. As a complex number *z* goes around the origin, the imaginary part of the logarithm goes up or down:

For further information on multi-valued complex functions see the following excellent reference:
http://en.wikipedia.org/wiki/Branch_point

Note: See section 9 ahead for further details on multi-valued functions, with the FOCAL driver program **ZMTV** (ZMulTiValue) that calculates all the consecutive results of the eight multi-value functions.

7.2 Complex Fibonacci Numbers. { ZFIB }

This short routine uses Binet's formula applied to the complex domain to calculate the Fibonacci number of a given complex "index". The result is another complex number that for integer cases coincides with the well-known Fibonacci series of course.

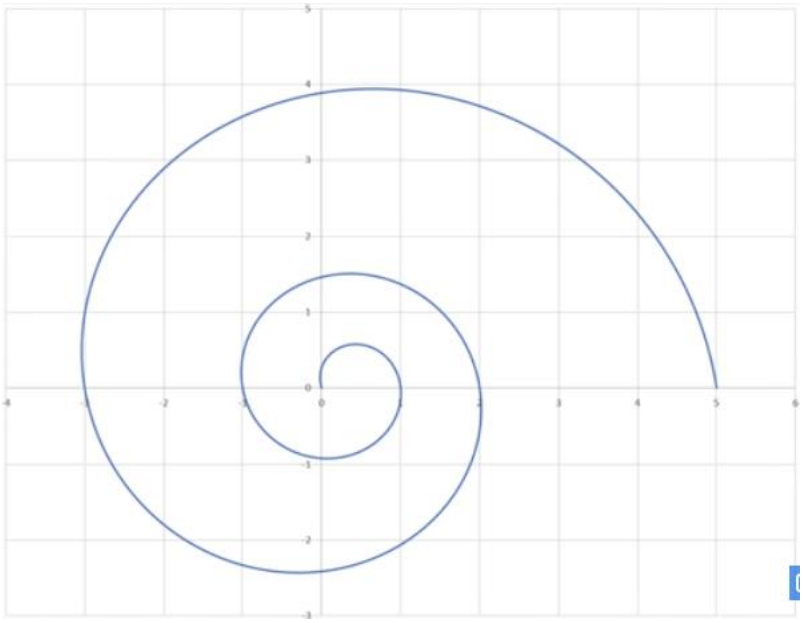
Binet's formula interpolation to noninteger real indexes (below left) provides an easy expression for the determination that guarantees real values also for the interpolated Fibonacci numbers

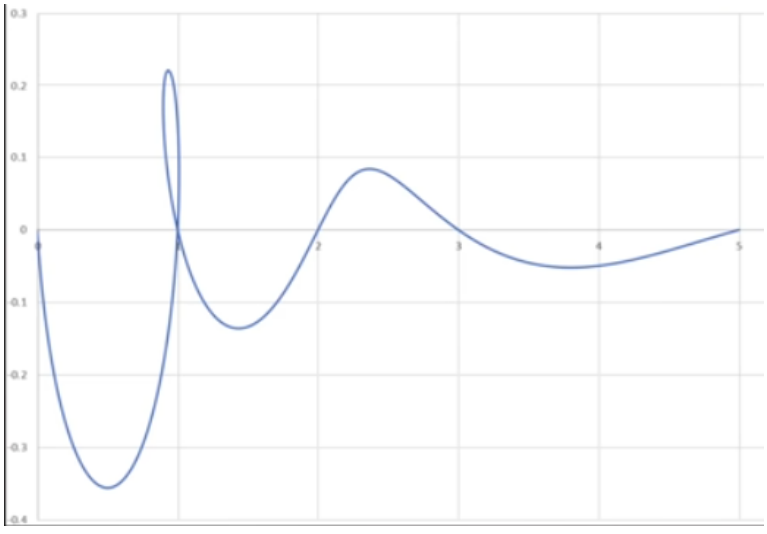
$$f_x^* = \frac{\varphi^x - \cos(\pi x)\varphi^{-x}}{\sqrt{5}} \qquad F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}$$

But it is in the complex domain (above right) when things become quite interesting, as can be seen in the graphical representations below, showing the locus of output results when the input values are negative real numbers (figure 1) and positive real value (figure 2). Note that both figures are not at the same scale/ (see also the animation at: https://www.geogebra.org/m/ypqcuqcs)

Program listing:

```
01  LBL "ZFIB"
02  ZRPL^
03  1.618033989
04  X^Z
05  ZENTER^
06  0
07  LASTX
08  1/X
09  CHS
10  ZRUP
11  W^Z
12  5
13  SQRT
14  ST/ Z
15  /
16  ZAVIEW
17  END
```





Example:

ZFIB (1.5)

0, ENTER^, 1,5, ΣZ$ "ZFIB"

Result:

0.9201+J0.21

8. 2D-vectors or complex numbers?

One of the common applications for complex numbers is their treatment as 2D vectors. This section covers the functions in 41Z that deal with vector operations between 2 complex numbers.

Table 8.1. 2D vectors function group.

Index	Function	Formula	Description
1	**ZWANG**	Arg(ZW) = Arg(Z) – Arg(W)	Angle between 2 vectors
2	**ZWDIST**	|W-Z| = SQR[(Wx-Zx)^2 – (Wy-Zy)^2]	Distance between 2 points
3	**ZWDOT**	Z*W = Zx*Wx + Zy*Wy	2D vector Dot product
4	**ZWCROSS**	Z x W = |z| *|w| *Sin(Angle)	2D vector Cross product
5	**ZWDET**	|ZW| = Wx*Zy – Wy*Zx	2D determinant
6	**ZWLINE**	a=(Y1-Y2) / (X1-X2) b=Y2 – a*X2	Equation of line through two points

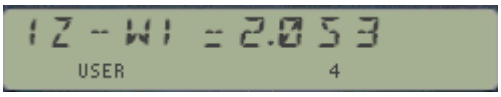
These functions use **W** and **Z** levels of the complex stack, leaving the result in level **Z** after performing complex stack drop. The original contents of **Z** is saved in the LastZ register.

The following screen captures from V41 show the different displays for these functions:

Let z = 4 <45 degrees, and w= 3 <75 degrees .

45, ENTER^, 4, **ZREC** -> 2,828(1+j)
ZREPL [don't forget or Z will be overwritten]
75, ENTER^, 3, **ZREC** -> 0,776 + 2,898J

1. **ZWANG**,- angle defined between both vectors (in degrees in this case)
2. ZRDN , LASTZ, **ZWDIST** – distance between both complex numbers

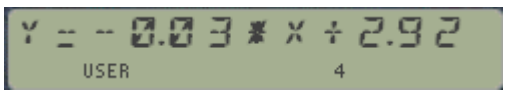
and


The angle will be expressed in the selected angular unit.

3. ZRDN , LASTZ, **ZWDOT** - dot product of both vectors
4. ZRDN, LASTZ, **ZWCROSS** - magnitude of the cross product of both vectors

and


5. ZRDN, LASTZ, **ZWDET** - magnitude of the determinant of both vectors
6. ZRDN, LASTZ, **ZWLINE** - equation of the straight line linking both points

and


(*) Note that despite having a simpler formula, ZWDET shows less precision than ZWCROSS.

Alternate Displaying: Quads and Tones.

ZDISP	**Compacted LCD view**	Positive RE, IM	D. Wilder
ZQUAD	**Shows Quad message**	Sets User flag	
ZTONE	**Sounds Tone using Y,X**	Duration and frequency	

These three functions provide additional user feedback on the complex value in the stack level Z (i.e. stack registers Y,X). Use them as a complement to the main **ZAVIEW**, each has interesting aspects but cannot be a full replacement to **ZAVIEW** given their shortcomings.

ZDISP main value is that it only uses the LCD to display a compacted version of the complex number. This leaves the ALPHA register undisturbed, in cases it needs to maintain its contents through a visualization of the Z result.

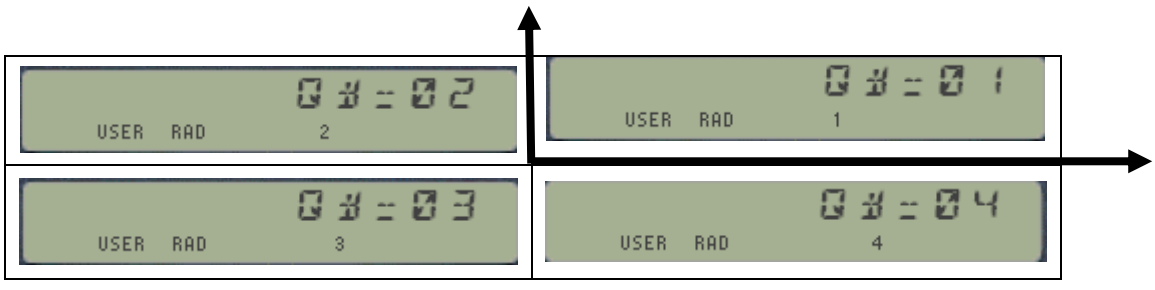
or:


- Each Real and Imaginary parts are allowed six LCD characters,
- Each is split as follows: three for the mantissa, one for the exponent sign and two more for the exponent itself.
- The two strings of six characters are separated by a comma to thell them apart
- No scrolling is supported, as all values are represented using the equivalent to a SCI 02 format.

Needless to say its main shortcoming is that it does not support negative values in the real or imaginary parts. This can be partially palliated using **ZQUAD** prior to **ZDISP**, as this will set the corresponding user flag depending on the complex location, leaving the other three flags from F1-F4 cleared:

- F1 set if bth Re(z)>0 and Im(z)>0
- F2 set if Re(z)<0 and Im(z)>0
- F3 set if both Re(z) and Im(z)<0
- F4 set if Re(z)?0 and Im(z)<0

ZQUAD will briefly show an informative message with the quadrant number, then it'll revert to the standard **ZAVIEW** output to end. It'll also reset the user flags 1-2-3-4 corresponding to the quadrant.



ZTONE will sound an acoustic tone using the information in the Y,X registers for frequency and duration; thus at least in theory each complex value is associated to its unique sound. In practice however the typical values make all sounds rather alike so it is more of a curiosity than of practical value. - for instance all real values (Im(z)=0) will have the same "blank" tone.

9. Polynomial Roots and Evaluation

A classic in calculator history just got improved. The 41Z Deluxe adds to the set new fast MCODE functions for the evalution of polynomials with complex coefficients, as well as their primitive and their first and second derivatives.

Table 9.1. Polynomial Evaluations group.

ZPL	**Polynomial Evaluation**	Control word in X	Does LastZ
ZPLI	**Primitive of Polynomial**	Control word in X	Does LastZ
ZPD1	**Pol. First derivative**	Control word in X	Does LastZ
ZPD2	**Pol. Second Derivative**	Control word in X	Does LastZ
ZINPT	**Data Input Routine**	Control word in X	FOCAL Routine
ZOUPT	**Data Output Routine**	Control word in X	FOCAL Routine

Besides the evaluation point z0, the evaluation functions require a control word as input parameter. This control word defines the complex register range used to store the polynomial coefficients, in the usual form "bbb.eee", *with the highest term coeff. stored in ZRbbb*. If the dregree of the polynomial is "n" there should be n+1 complex registers in the range, i.e. (eee-bbb) = n

Like the other hybrid functions in the module, you need to enter the complex value first (z0) and then the real value (control word) in the X-register – which will push z0 one level up in the REAL stack. The result will be retuned in the complex-Z register, with z0 saved in LastZ - but the control word is lost (i.e. not saved in LastX).

The utility routines **ZINPT** and **ZOUPT** come very handy to enter the polynomial coefficients in the complex registers. They too use the same control word bbb.eee to define the complex register range used for the input/output action. Let's see one example next.

Example: Evaluate the polynomial, main primitive and derivatives in the point z0=1+i for the 4th.-degree polynomial: **P(z) = (1+i) z^4 – (2-3i) z^3 + (-1+2i) z^2 + z – (1+i)**

First we introduced the coefficients in the complex data registers ZR00 to ZR04 as follows. Note that the index in the prompts refers to the complex register to use, *and not to the polynomial term*:

```
0.004, ZF$ "ZINPT",          "Z0=?"
1, ENTER^, R/S               "Z1=?"
3, ENTER^, 2, CHS, R/S       "Z2=?"
2, ENTER^, 1, CHS, R/S       "Z3=?"
0, ENTER^, 1, R/S            "Z4=?"
1, ENTER^, ZNEG, R/S         shows Z-level again
```

With the data safely stored in {ZR00 – ZR04} we proceed to calculate the results. First we enter the evaluation point in the complex Z register, followed by the control word in X:

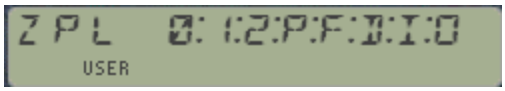
```
1, ENTER^, ENTER^, 0.004,   XEQ "ZPL"    =>   -10-J16
LASTZ, 0.004,               ZF$ "ZPD1"    =>   -39-J10
LASTZ, 0.004,               ZF$ "ZPD2"    =>   -56+J34
LASTZ, 0.004                ZF$ "ZPLI"    =>    1.333-J7.600
```

Note how the result for the polynomial and derivatives have integer real and imaginary parts (i.e. are Gaussian numbers) – but the primitive is not. We'll revisit these results when we cover the Complex Derivative Engine in the next chapters.

Preview: Polynomial Roots and Values Launchers - both together now.

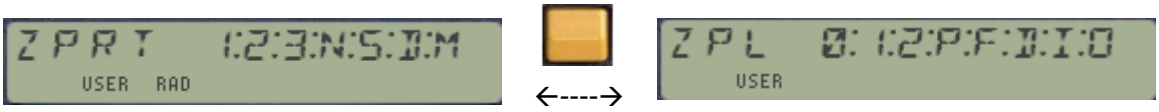
A convenient grouping of the polynomial functions provides access to the individual choices from a common prompt**.** To access it you can use its dedicated launcher from the complex keyboard shortcut - just press:

[**Z**], [**A**], [**L**], [SHIFT]



Options "**I**" / "**O**" will trigger **ZINPUT** and **ZOUPT** respectively. Note that there are other functions also included here ("F" and "D"), which are related to the first derivative and continued fractions. They will be covered in another chapter later in the manual.

Note also that pressing [SHIFT] again this launcher toggles with the corresponding for the Complex Roots, as shown below:



A convenient grouping of the root-finding applications provides access to the root finders for the first, second, third and n-th. degree polynomials, as well as the general-purpose **ZSOLVE.** To access it you can call the sub-function **ZPRT**, or using the complex keyboard shortcut just press:

[**Z**], [**A**], [L]

The first-degree option is for function **ZWLINE** - not strictly a root finder but being such a simple case it's convenient to have it also in the group.

For **ZQRT** and **ZCRT** the coefficients are expected to be in the complex stack prior to the execution – whilst **ZPROOT** and **ZSOLVE** will prompt for the required entries.

Solution of Quadratic and Cubic equations.

ZQRT	**Roots of 2nd. Degree Eq.**	Coeffs. in Z-Stack	All MCODE
ZCRT	**Roots of 3rd. Degree Eq.**	Coeffs. In Z-Stack	FOCAL program
ZQUDR	**Driver for ZCRT**	Prompts for values	FOCAL Routine

ZQRT Solves the roots of a quadratic equation with complex coefficients, as follows:

$C_1 * z^2 + C_2 * z + C_3 = 0$; where C_1, C_2, C_3, and z are complex numbers

By applying the general formula: $z_{1,2} = [-C_2 +/- SQR(C_2^2 - 4C_1*C_3)] /2*C_1$

Example 1.- find out the roots of **(1+i)*z² + (-1-i)*z + (1-i) = 0**

1, ENTER^, ZENTER^
1, CHS, ENTER^, ZENTER^
1, CHS, ENTER^, 1, XEQ "ZQRT"
"RUNNING..." followed by: " 1,300+j0,625"
Z<>W " -0,300-j0,625"

We see that contrary to the real coefficients case, here the roots are NOT conjugated of one another.

ZQRT is entirely written in MCODE. It expects the three complex coefficients stored in levels **V**, **W**, and **Z** of the complex stack. The driver program below is an example using FOCAL instructions instead. Note also that *no memory registers are used,* and all calculations are performed using exclusively the complex stack. The core of the program is from lines 16 to 37, or just 21 programming steps to resolve both roots.

1	LBL "ZQDR"	16	ZENTER^	31	ZENTER^
2	*"aZ^2+bZ+c=0"*	17	ZR^	32	ZNEG
3	AVIEW	18	Z/	33	ZR^
4	PSE	19	LASTZ	34	Z+
5	*"IM^RE a=?"*	20	ZR^	35	ZRDN
6	PROMPT	21	Z<>W	36	Z+
7	ZENTER^	22	Z/	37	ZRUP
8	*"IM^RE b=?"*	23	ZHALF	38	SF 21
9	PROMPT	24	ZNEG	39	ZAVIEW
10	ZENTER^	25	ZENTER^	40	Z<>W
11	*"IM^RE c=?"*	26	ZENTER^	41	CF 21
12	PROMPT	27	Z^2	42	ZAVIEW
13	*"RUNNING..."*	28	ZR^	43	END
14	AVIEW	29	Z-		
15	LBL "ZQRT"	30	ZSQRT		

Solving the Cubic Equation.

Example 2. Obtain the three roots of **(1+2i) z^3 – (2-i) z – 3i = 0**

We type: 2, ENTER^, 1, **[Z]**, 0, **ZENTER^**, 1, ENTER, 2, CHS, **[Z]** , **[,]**, 3, **ZNEG**
to obtain the three solutions in the complex stack, as follows:

XEQ "ZCRT" → z1 = -0,117-J0,910
ZRDN → z2 = -0,922+J1,047
ZRDN → z3 = 1,039-J0,136

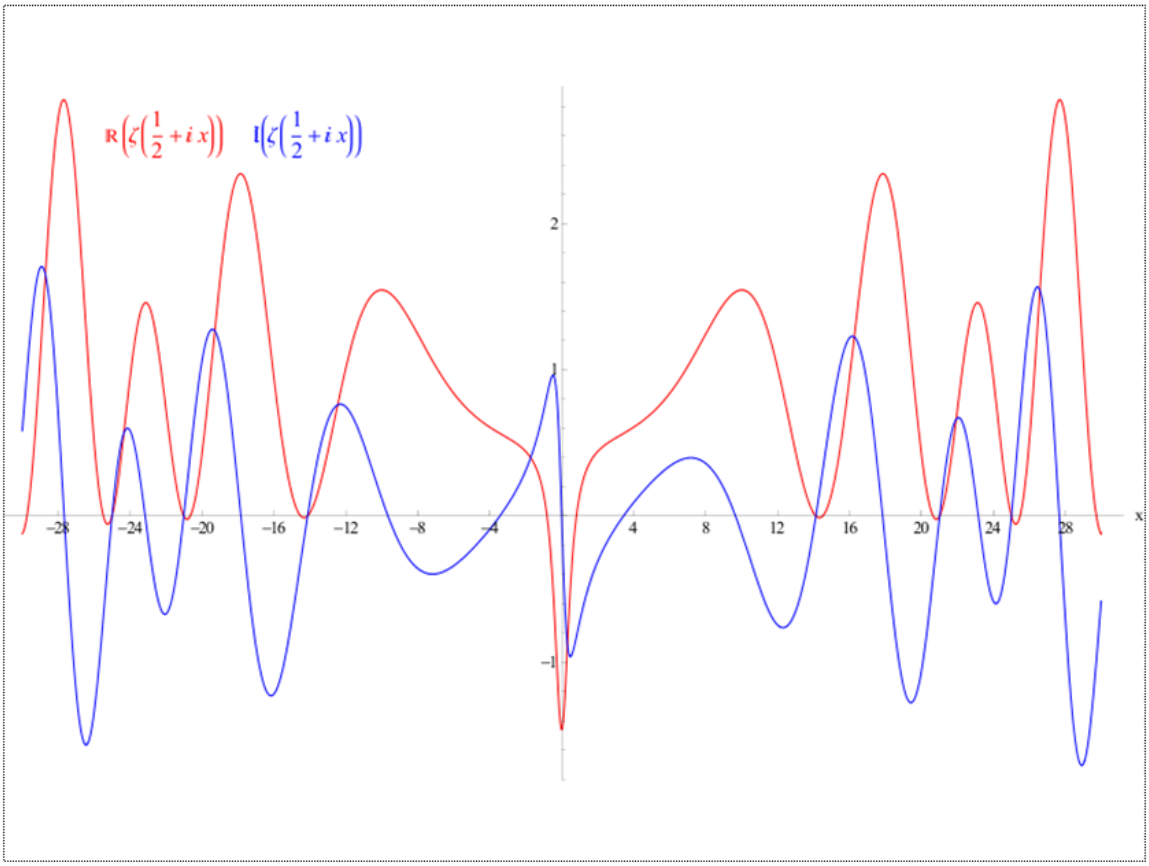
Two ways to skin the third-degree Equation Cat.

The programs below show two alternative solutions for the third degree equation roots. Note the existing symmetry between them, in fact identical until step 31. The version on the left is the implemented in the 41Z module. Both use a variation of the Cardano-Vieta formulas involving some trigonometry tricks that notably reduce the number of steps.

1	**LBL "ZCRT"**	*Main version*	**LBL "ZCRT2"**	*Alterntaive Version*
2	**ZRUP**	*a3*	**ZRUP**	*a3*
3	**Z/**	*a0/a3*	**Z/**	*a0/a3*
4	**ZSTO (00)**	*a0'*	**ZSTO (00)**	*a0'*
5	**Z<>W**	*a1*	**Z<>W**	*a1*
6	**LASTZ**	*a3*	**LASTZ**	*a3*
7	**Z/**	*a1/a3*	**Z/**	*a1/a3*
8	**ZSTO 01**	*a1'*	**ZSTO 01**	*a1'*
9	**ZRUP**	*a'2*	**ZRUP**	*a'2*
10	**LASTZ**	*a3*	**LASTZ**	*a3*
11	**Z/**	*a2/a3*	**Z/**	*a2/a3*
16	**3**		**3**	
17	**ST/ Z**		**ST/ Z**	
18	**/**		**/**	
19	**ZSTO 02**	*a2' / 3*	**ZSTO 02**	*a2' / 3*
12	**Z^2**	*a2^2 / 9*	**Z^2**	*a2^2 / 9*
13	3		**3**	
14	**ST* Z**		**ST* Z**	
15	*****	*a2^2 / 3*	*****	*a2^2 / 3*
20	**Z-**	*a1-a2^2 / 3*	**Z-**	*a1-a2^2 / 3*
21	**ZRCL 02**	*a2 /3*	**ZRCL 02**	*a2 /3*
22	**Z^3**	*a2^3 / 27*	**Z^3**	*a2^3 / 27*
23	**ZDBL**	*2 a2^3 /27*	**ZDBL**	*2 a2^3 /27*
24	**ZRCL 01**	*a1*	**ZRCL 01**	*a1*
25	**ZRCL 02**	*a2/3*	**ZRCL 02**	*a2/3*
26	**Z***	*a1*a2 /3*	**Z***	*a1*a2 /3*
27	**Z-**	*(a2^3 / 27)- (a1*a2/3)*	**Z-**	*(a2^3 / 27)- (a1*a2/3)*
28	**ZRCL (00)**	*a0*	**ZRCL (00)**	*a0*
29	**Z+**	*q = a0 + (a2^3 / 27)- (a1*a2/3)*	**Z+**	*q = a0 + (a2^3 / 27)- (a1*a2/3)*
30	**ZHALF**	q/2	**ZHALF**	q/2
31	**Z<>W**	p	**Z<>W**	p
32	**3**		**-3**	
33	**ST/ Z**		**ST/ Z**	
34	**/**	*p/3*	**/**	*-p/3*
35	**Z/**	*3q/2p*	**Z/**	*-3q/2p*
36	**LASTZ**	*p/3*	**LASTZ**	*-p/3*
37	**ZSQRT**	sqr(p/3)	**ZSQRT**	sqr(-p/3)
38	**ZSTO (00)**		**ZSTO (00)**	
39	**Z/**	*3q/2p / sqr(p/3)*	**Z/**	*-3q/2p / sqr(-p/3)*
40	**ZHASIN**		**ZASIN**	
41	**3**		**3**	
42	**ST/ Z**		**ST/ Z**	
43	**/**	*1/3 asin[ ]*	**/**	
44	**ZRPL^**	*Fill complex stack*	**ZRPL^**	*Fill complex stack*
45	**,002**		**,002**	
46	**STO 02**		**STO 02**	
47	**RDN**		**RDN**	
48	***LBL 02***	*Data output loop*	**LBL 02**	*Data output loop*

49	**RCL 02**		**RCL 02**	
50	**INT**		**INT**	
51	**120**	*2kπ/3*	**120**	*2kπ/3*
52	**D-R**		**D-R**	
53	*****		*****	
54	**ST+ Z**	*add to imaginary part*	**+**	*add to real part*
55	**RDN**	*tidy up stack*	**ZSIN**	
56	**ZHSIN**		**ZRCL (00)**	
57	**ZRCL (00)**		**Z***	
58	**Z***		**ZDBL**	
59	**ZDBL**		**ZRCL 02**	*a2/3*
60	**ZNEG**		**Z-**	
61	**ZRCL 02**	*a2/3*	**ZAVIEW**	
62	**Z-**		**ZRUP**	*save in Z-stack*
63	**ZAVIEW**	*Show progress…*	**ISG 02**	*Increase counter*
64	**ZRUP**	*save in Z-stack*	***GTO 02***	*Go for next*
65	**ISG 02**	*Increase counter*	**END**	*done*
66	***GTO 02***	*Go for next*		
67	**END**	*done*		

As you can see the density of 41Z functions is remarkable. The 41Z complex function set and complex stack enables the programmer to treat complex calculations as though they used real numbers, not worrying about the real or imaginary parts but working on the complex number as single entity. In fact, exercising some care (notably to ensure complex stack lift), you could almost translate many FOCAL programs by replacing the standard functions one-to-one with the equivalent complex ones. That's why it's important that the function set be as complete as possible, and that the complex stack implementation follows the same RPN conventions.



Roots of Complex Polynomials. { **ZPROOT** , **ZPLRT** }

ZPROOT	**Roots of Polynomials**	Data entry/output	*Valentín Albillo*
ZPLRT	**Polynomial Roots**	Uses Newton method	*Martin - Baillard*

These programs calculate all the roots of a polynomial of degree n, and with complex coefficients. It is therefore *the most general case of polynomial root finders* that can possibly be used, as it also will work when the coefficients are real.

- The first one is a wonderful example of FOCAL capabilities, and very well showcases the versatility of the HP-41C (even without the 41Z module). It was first published on PPC Technical Notes, PPCTN – the journal of the Australian chapter of the PPC. The program includes data entry and output, simply answer the prompts as they're presented. See the program listing in the appendix below.

- The second is a direct implementation of the Newton method combined with a deflation technique for each root found using the iterative process. It is based on JM Baillard's example for real roots (see paragraph #1.f at: *http://hp41programs.yolasite.com/polynomials.php*), simply replacing the standard HP-41 functions with 41Z equivalents – to make it valid in the complex domain. This method takes advantage of the polynomial evaluation and first derivative MCODE functions (**ZPL** and **ZPLD1**) , which should reduce considerably the execution time provided that a good initial guess is provided.

The routine assumes the polynomial coefficients are stored in Complex Data registers ZR(bbb) to ZR(eee) - the initial guess is the {Z,Y} stack registers, and the polynomial control word "bbb.eee" in the X-register (using Complex Data register indexes). You can automate the data entry process using sub-function ZINPT, make sure that the first complex register used is no lower than ZR03 (i.e. real registers {R06 and R07})

```
01 LBL "ZPRT  bbb,eee
02 STO 02
03 STO 03     reg range
04 STO 04
05 RDN
06 ZSTO (00)
07 ISG 04
08 LBL 01
09 ZRCL (00)
10 ZAVIEW
11 RCL 03     reg range
12 ZPL
13 LASTZ
14 RCL 03     reg range
15 ZPLD1      (ZF# 47)
16 Z/
17 ZST- (00)
18 ZRCL (00)
19 Z=0?
20 SIGN
21 Z/
22 ZMOD
23 E-8        tolerance
24 X<Y?
25 GTO 01
26 E-3        0,001
27 ST- 03     deflate pol
28 RCL 03     reduced deg
29 STO 05     used as index
30 CLZ
31 LBL 02
32 ZRC* (00)
33 ZST+ IND 05
34 ZRCL IND 05
35 ISG 05
36 GTO 02
37 ZRCL (00)
38 ZSTO IND 05
39 ISG 04
40 GTO 01
41 RCL 02
42 E
43 +          bbb+1,eee
44 END
```

Registers used by ZPLRT.

The program uses ZR00 (i.e. {R00-R01}) to hold the current complex guess, and registers R02-R05 for scratch. Therefore the polynomial coefficients cannot be stored in complex registers below ZR03 (i.e. {R06-R07}).

This method convergence is quite fast, which also contributes to the general good performance. This however is conditioned to a good initial guess as entered by the user.

Example 1.- Calculate the roots of P(x) = **2.x^5 + 3.x^4 – 35.x^3 – 10.x^2 + 128.x -74**

Using **ZINPT** we introduce the six coefficients in registers {ZR03 – ZR08}, then enter the guess z0=(1+0i) and the control word for the polynomial, then execute the routine. The successive values are shown, and upon completion the control word of the roots is left in X – so you can use **ZOUPT** to review. The roots are all real, with values:

ZRCL 03 -> -4.373739462
ZRCL 04 -> -2.455070118
ZRCL 05 -> 2.984066207
ZRCL 06 -> 1.641131729
ZRCL 07 -> 0.703611645

Example 2.- Calculate the three roots of: $x^3 + x^2 + x + 1$

XEQ "ZPROOT"	-> "DEGREE=?"
3, R/S	-> "IM^RE (3)=?"
0, ENTER^, 1, R/S	-> "IM^RE (2)=?"
0, ENTER^, 1, R/S	-> "IM^RE (1)=?"
0, ENTER^, 1, R/S	-> "IM^RE (0)=?"
0, ENTER^, 1, R/S	-> "SOLVING…"
	-> "FOUND ROOT#3", and "SOLVING…"
	-> "FOUND ROOT#2", and "SOLVING…"
	-> "FOUND ROOT#1"
	➔ -5,850E-14-j1 (that is, -i)
	➔ 5,850E-14+j1 (that is, i)
	➔ -1+j1,170E-13 (that is, -1)

Example 3.- Calculate the four roots of: $(1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z - 1$

XEQ "ZPROOT"	-> "DEGREE=?"
4, R/S	-> "IM^RE (4)=?"
2, ENTER^, 1, R/S	-> "IM^RE (3)=?"
2, CHS, ENTER^, 1, CHS, R/S	-> "IM^RE (2)=?"
3, CHS, ENTER^, CHS, R/S	-> "IM^RE (1)=?"
0, ENTER^, 1, R/S	-> "IM^RE (0)=?"
0, ENTER^, 1, CHS, R/S	-> "SOLVING…"
	-> "FOUND ROOT#4", and "SOLVING…"
	-> "FOUND ROOT#3", and "SOLVING…"
	-> "FOUND ROOT#2", and "SOLVING…"
	-> "FOUND ROOT#1"
	1,698+J0,802 R/S
	➔ -0,400-J0,859 R/S
	➔ 0,358+J0,130 R/S
	➔ -0,656-J0,073

The four solutions are:

z_1 = 1,698 + 0,802 j or: 1,878 <) 25,27
z_2 = -0,400 - 0,859 j or: 0,948 <)-114,976
z_3= 0,358 + 0,130 j or: 0,381 <) 9,941
z_4 = -0,,656 - 0,073 j or: 0,660 <)-173,676

(*) You can also use the Z-pad to input real coefficients, i.e. [**Z**], 1 instead of 0, ENTER^, 1.

Appendix.- Program Listing for ZPROOT

1	LBL "ZPROOT"		44	CF 00		87	E-3		130	*GTO 02*	
2	SIZE?		45	CHS		88	ST+ 01		131	RCL 08	
3	*"DEGREE=?"*		46	STO 04		89	RCL 03		132	ST* Z	
4	PROMPT		47	FIX 2		90	STO IND 05		133	*	
5	STO Z		48	RND		91	RCL 04		134	DSE 08	
6	ST+X	*2N*	49	FIX 6		92	STO IND 06		135	*GTO 02*	
7	11		50	X#0?		93	DSE 00		136	RTN	
8	+	*2N+11*	51	*GTO 01*		94	*GTO 06*		137	*LBL 00*	
9	X>Y?		52	SIGN		95	TONE 5		138	ZENTER^	
10	PSIZE		53	STO 04		96	RCL 01		139	RCL 04	
11	RCL Z		54	*LBL 01*		97	INT		140	RCL 03	
12	STO 00	*N*	55	RCL 00		98	E1		141	Z*	
13	STO 03	*N*	56	STO 08		99	-		142	RCL IND 05	
14	9,008		57	SF 01		100	E3		143	FS? 01	
15	+		58	*XEQ 11*		101	/		144	RCL 08	
16	STO 01	*N+9,008*	59	R-P		102	ST- 05		145	FS? 01	
17	STO 05	*N+9,008*	60	1/X		103	FIX 3		146	*	
18	X<>Y	*2N+11*	61	STO 07		104	SF 21		147	+	
19	E		62	X<>Y		105	*LBL 10*		148	FS? 00	
20	-	*2N+10*	63	CHS		106	ISG 00		149	STO IND 05	
21	STO 02	*2N+10*	64	STO 08		107	NOP		150	X<>Y	
22	STO 06		65	CF 01		108	RCL IND 06		151	RCL IND 06	
23	FIX 0		66	*XEQ 11*		109	RCL IND 05		152	FS? 01	
24	CF 29		67	ZENTER^		110	ZAVIEW		153	RCL 08	
25	*LBL 05*		68	RCL 08		111	DSE 06		154	FS? 01	
26	*"IM^RE("*	*N*	69	RCL 07		112	DSE 05		155	*	
27	ARCL 03		70	P-R		113	*GTO 10*		156	+	
28	*")-)=?"*		71	Z*		114	CF 21		157	FS? 00	
29	PROMPT		72	ST- 03		115	SF 29		158	STO IND 06	
30	STO IND 05		73	X<>Y		116	RTN		159	X<>Y	
31	X<>Y		74	ST- 04		117	*LBL 11*		160	FS? 01	
32	STO IND 06	*N-1*	75	ZRND		118	RCL 01		161	DSE 08	
33	DSE 03		76	Z#0?		119	STO 05		162	*LBL 02*	
34	X<>Y		77	*GTO 01*		120	RCL 02		163	DSE 06	
35	DSE 06		78	FIX 0		121	STO 06		164	DSE 05	
36	DSE 05		79	*"FOUND ROOT#"*		122	FC? 01		165	*GTO 00*	
37	*GTO 05*		80	ARCL 00		123	*GTO 13*		166	END	
38	RCL 03		81	AVIEW		124	E-3				
39	*LBL 06*		82	SF 00		125	ST+ 05				
40	*"SOLVING..."*		83	*XEQ 11*		126	*LBL 13*				
41	AVIEW		84	E		127	RCL IND 06				
42	SF 25		85	ST+ 05		128	RCL IND 05				
43	SF 99		86	ST+ 06		129	FC? 01				

10. It's a Gamma-Zeta world out there.

This section describes the different functions and programs included on the 41Z that deal with the calculation of the Gamma and Zeta functions in the complex plane. A group of six functions in total, three completely written in machine code and three as FOCAL programs, with a couple of example applications to complement it.

Table 10.1. Gamma function group.

ZGAMMA	**Complex Gamma function**	for z#-k, k=integer	Does LastZ
ZLNG	**Gamma Logarithm**	see below	Does LastZ
ZPSI	**Complex Digamma (Psi)**	see below	Does LastZ
ZIGAM	**Inverse of Gamma**	Iterative method	FOCAL program
ZPSIN	**Complex Poly-Gamma**	See below	FOCAL program
ZZETA	**Complex Riemann Zeta**	For z#1	FOCAL program

ZGAMMA uses the Lanczos approximation to compute the value of Gamma. An excellent reference source is found under http://www.rskey.org/gamma.htm, written by Viktor T. Toth. To remark that **ZGAMMA** is implemented completely in machine code, even for Re(z)<0 using the reflection formula for analytical continuation.

For complex numbers on the positive semi-plane [Re(z)>0], the formula used is as follows

$$\Gamma(z) = \frac{\sum_{n=0..N} q_n z^n}{\prod_{n=0..N} (z+n)} (z+5.5)^{z+0.5} e^{-(z+5.5)}$$

q_0 =	75122.6331530
q_1 =	80916.6278952
q_2 =	36308.2951477
q_3 =	8687.24529705
q_4 =	1168.92649479
q_5 =	83.8676043424
q_6 =	2.5066282

$$\Gamma(1-z)\,\Gamma(z) = \frac{\pi}{\sin(\pi z)}$$

And the following identity (reflection formula) is used for numbers in the negative semi-plane: [Re(z)<0]: which can be re-written as: **Γ(z) * Γ(-z) = -π / [z*Sin(π z)]**

For cases when the real part of the argument is negative [Re(z)<0], **ZGAMMA** uses the analytical continuation to compute the reflection formula – all internal in the MCODE and transparent to the user.

Example 1.- Calculate Γ(1+i)

1, ENTER^, **ZGAMMA** -> "RUNNING…", followed by -> 0,498-j0,155

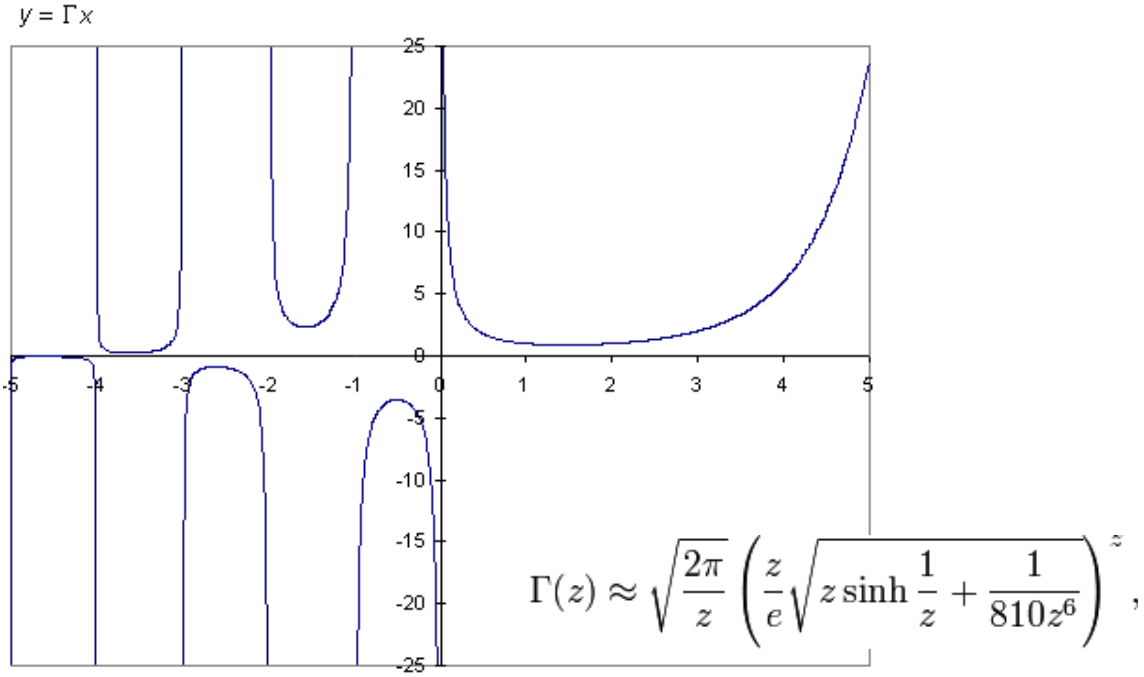
Example 2.- Verify that Γ(1/2) = SQR(π)

0, ENTER^, 0.5, **ZGAMMA** -> 1,772 + j0
PI, SQRT, **ZREAL^**, **Z-** -> -2,00E-9 + j0

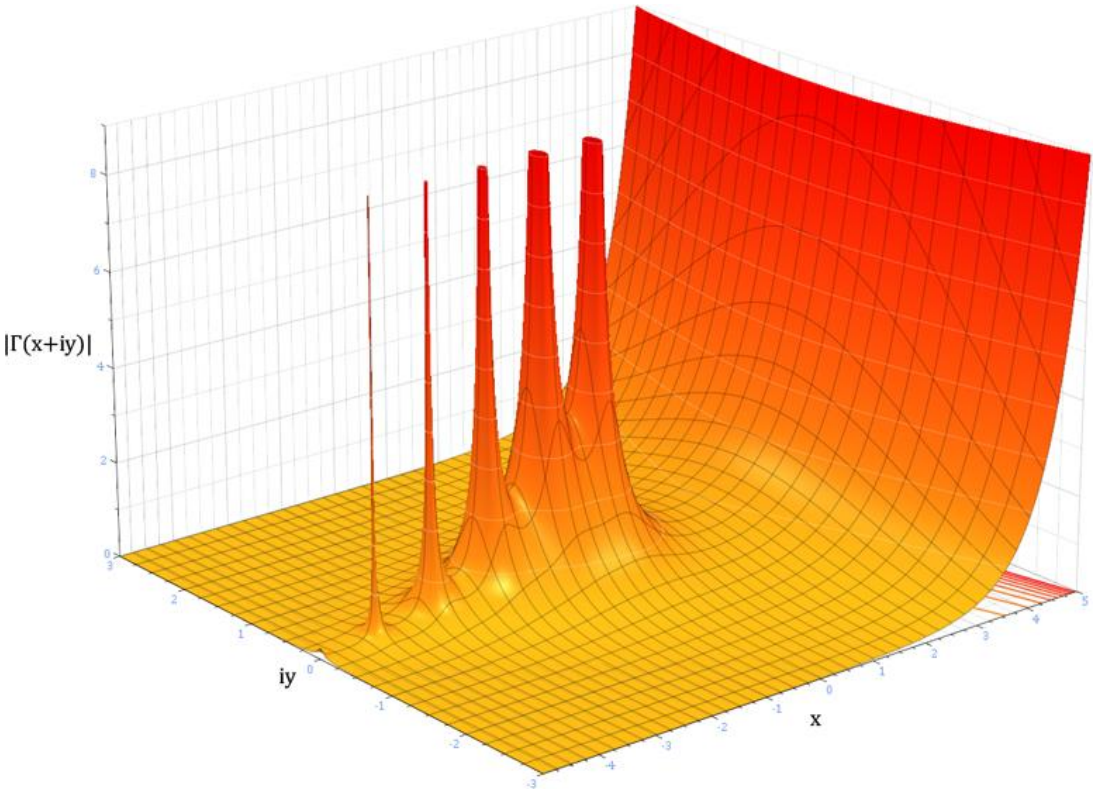
Example 3.- Calculate Γ(-1.5+i)

1, ENTER^, 1.5, CHS, **ZGAMMA** -> 0,191 + j0,174

The graphic below (also from the same web site) shows Gamma for real arguments. Notice the poles at x=0 and negative integers. Also below the Stirling's approximation for Gamma:



The following graphic showing the ***module of the Complex Gamma*** function is taken from http://en.wikipedia.org/wiki/Gamma_function.- Note the poles at the negative integers and zero.



Example: Use **ZLNG** to calculate Γ(1+i) and compare it with the value obtained by **ZGAMMA**

1, ENTER^, **ZGAMMA, LASTZ, ZLNG, ZEXP, Z-** -> 2,400E-9+j3,000E-10

Digamma and LogGamma Functions { **ZPSI** , **ZLNG** }

Both the Digamma and LogGamma are implemented entirely in MCODE – fast execution and full LastZ support of the original argument. No data registers are used, no additional complex stack levels.

The formula used is the approximation for Digamma when x>8:

$$\Psi(x) = \log(x) - \frac{1}{2x} - \frac{1}{12x^2} + \frac{1}{120x^4} - \frac{1}{252x^6} + O\left(\frac{1}{x^8}\right)$$

programmed as: **u^2{[(u^2/20-1/21)u^2 + 1/10]u^2 –1}/12 – [Ln u + u/2]**,

where **u=1/x**; and using the following precision correction factor when x<8

$$\Psi(x+1) = \Psi(x) + \frac{1}{x}.$$

Equivalent Program listings.- The two FOCAL programs listed below calculate the Digamma and the Gamma functions for complex arguments. The first one is an example using the asymptotic approximation as described below, whilst the second one is an extension of the MCODE function **ZGAMMA**, using the reflection formula for arguments with Re(z)<1 (programmed in turn as another MCODE function, **ZGNZG**).

```
01  LBL "ZPSI"
02  ZREPL^
03  7 E-3
04  STO O
05  CLZ
06  LBL 00
07  Z<>W
08  RCL O
09  INT
10  +
11  ZINV
12  Z+
13  ISG O
14  GTO 00
15  ZSTO
16  E
17  Z<>W
18  8
19  +
20  ZINV
21  ZSTO (00)
22  Z^2
23  ZREPL
24  20
25  ZREAL^
26  Z/
27  21
28  1/X
29  ZREAL^
30  Z-
31  Z*
32  0.1
33  +
34  Z*
35  E
36  -
37  Z*
38  12
39  ZREAL^
40  Z/
41  ZRCL (00)
42  ZLN
43  LASTZ
44  ZHALF
45  Z+
46  Z-
47  ZRCL
48  E
49  Z-
50  ZAVIEW
51  END
```

for x>8

Psi(x) = ln x - 1/(2x) -1/(12x^2) + 1/(120x^4) - 1/(252x^6) + 1/(240x^8)

together with the relationship: Psi(x+1) = Psi(x) + 1/x

```
01  LBL "ZG"
02  ZENTER^
03  X<>Y
04  X#0?
05  GTO 00
06  X<>Y
07  X>0?
08  GTO 00
09  INT
10  LASTX
11  X#Y?
12  GTO 00
13  0
14  1/X
15  LBL 00
16  ZRDN
17  CF 00
18  X<0?
19  SF 00
20  FS? 00
21  ZNEG
22  FS? 00
23  INCX
24  ZGAMMA
25  FC? 00
26  GTO 01
27  LASTZ
28  ZGNZG
29  Z<>W
30  Z/
31  LBL 01
32  ZAVIEW
33  END
```

The following two programs calculate the Logarithm of the Gamma function for complex arguments. The first one uses the Stirling approximation, with a *correction factor* to increase the precision of the calculation. This takes advantage of the **ZGPRD** function, also used in the Lanczos approximation.

$$2\ln\Gamma(z) \approx \ln(2\pi) - \ln z + z\left(2\ln z + \ln\left(z\sinh\frac{1}{z} + \frac{1}{810z^6}\right) - 2\right),$$

correction factor: **LnΓ(z) = LnΓ(z+7) – Ln[PROD(z+k)|k=1,2..6]**

The second one applies the direct definition by calculating the summation until there's no additional contribution to the partial result when adding more terms. In addition to being much slower than the Stirling method, this is also dependent of the display precision settings and thus not the recommended approach. It is not included on the 41Z but nevertheless is an interesting example of the utilization of some of its functions, like **Z=WR?** and the memory storage registers, **ZSTO** and **ZRCL**.

01	LBL "ZLNG"	
02	7	
03	+	(z+7)
04	ZSTO (00)	
05	*Text-0*	*NOP*
06	6	
07	Z^X	
08	810	
09	ST* Z	
10	*	
11	ZINV	
12	ZRCL (00)	
13	ZINV	
14	ZSINH	
15	ZRCL (00)	
16	Z*	
17	Z+	
18	ZLN	
19	ZRCL (00)	
20	ZLN	
21	ZDBL	
22	Z+	
23	2	
24	-	
25	ZRCL (00)	
26	Z*	
27	ZRCL (00)	
28	ZLN	
29	Z-	
30	PI	
31	ST+ X	
32	LN	
33	+	
34	ZHALF	
35	ZRCL (00)	
36	*Text-0*	*NOP*
37	7	
38	-	z
39	ZGPRD	
40	ZLN	
41	Z-	
42	ZAVIEW	
43	END	

01	LBL "ZLNG2"
02	1
03	STO 02
04	RDN
05	ZSTO (00)
06	XEQ 05
07	LBL 00
08	ZENTER^
09	XEQ 05
10	Z+
11	Z=WR?
12	GTO 02
13	GTO 00
14	LBL 02
15	ZRCL (00)
16	ZLN
17	Z-
18	ZRCL (00)
19	0,5772156649
20	ST* Z
21	*
22	Z-
23	ZAVIEW
24	RTN
25	LBL 05
26	ZRCL (00)
27	RCL 02
28	ST/ Z
29	/
30	ZENTER^
31	1
32	+
33	ZLN
34	Z-
35	1
36	ST+ 02
37	RDN
38	END

Max ZREG#	Size
n/a	1
0	2
	3
1	4
	5
2	6
	7
3	8
	9
4	10
	11
5	12
	13
6	14
	15
7	16
	17
8	18
	19
9	20
	21
10	22
	23
11	24
	25
...	...

The table above shows the correspondence between the complex register number (ZRnn) and the required SIZE in the calculator. Note that a minimum of SIZE 002 is required for ZR00 to exist.

$$\ln\Gamma(z) = -\gamma z - \ln z + \sum_{k=1}^{\infty}\left[\frac{z}{k} - \ln\left(1+\frac{z}{k}\right)\right]$$

Poly-Gamma Function { **ZPSIN** }

To complete the set of Gamma-related functions, here's a 41Z version of JM Baillard's program to calculate PSIN for a generic integer degree n. The program listing is given below, notice the usage of the STO math functions as well as other 41Z fixtures (like the complex stack and data register management) showcasing the applicability of the function set.

```
1 LBL "ZPSIN
  02  STO 09
  03  RDN
  04  ZSTO 01
  05  CLX
  06  STO 04
  07  STO 05
  08  LBL 01
  09  ZRCL 01
  10  ZSTO 00
  11  RCL 09
  12  8
  13  +
  14  X<Y?
  15  GTO 00
  16  CLX
  17  E
  18  RCL 09
  19  +
  20  CHS
  21  Z^X
  22  ZST+ 02
  23  E
  24  ST+ 02
  25  GTO 01
  26  LBL 00
  27  ZRCL 00
  28  ZSTO 03
  29  ZRCL 01
  30  Z^2
  31  ZINV
  32  ZSTO 00
  33  ZSTO 01
  34  RCL 09
  35  9
  36  +
  37  FACT
  38  39.6
  39  /
  40  0
  41  X<>Y
  42  ZST* 00
  43  XEQ 05
  44  RCL 09
  45  7
  46  +
  47  FACT
  48  ST- 00
  49  XEQ 05
  50  0
  51  40
  52  ZST/ 00
  53  RCL 09
  54  5
  55  +
  56  FACT
  57  ST+ 00
  58  XEQ 05
  59  0
  60  42
  61  ZST/ 00
  62  RCL 09
  63  3
  64  +
  65  FACT
  66  ST- 00
  67  XEQ 05
  68  0
  69  6
  70  ZST/ 00
  71  RCL 09
  72  FACT
  73  STO 08
  74  ZST* 02
  75  ZRCL 00
  76  ZSTO 01
  77  ZRCL 03
  78  ZINV
  79  RCL 08
  80  ST* Z
  81  *
  82  ZSTO 00
  83  Z<>W
  84  ZST+ 00
  85  NOP
  86  2
  87  ST/ 00
  88  ST/ 01
  89  RCL 09
  90  X=0?
  91  GTO 00
  92  E
  93  -
  94  FACT
  95  ST+ 00
  96  ZRCL 00
  97  ZSTO 01
  98  ZRCL 03
  99  RCL 09
 100  CHS
 101  Z^X
 102  Z*
 103  GTO 02
 104  LBL 00
 105  ZRCL 00
 106  ZSTO 01
 107  ZRCL 03
 108  ZLN
 119  Z-
 110  LBL 02
 111  ZRCL 02
 112  Z+
 113  RCL 09
 114  ZCHSX
 115  ZNEG
 116  ZAVIEW
 117  RTN
 118  LBL 05
 119  ZRCL 01
 120  ZST* 00
 121  END
```

Examples: Calculate (n=2) Tetra- and (n-3) Penta-gamma of z=1+i, and w=-1-i :

```
1, ENTER^, ENTER^, 2, ZF$ "ZPSIN"      => "RUNNING..." =>  0.369+J0.767
1, ENTER^, ENTER^, 3, LASTF            => "RUNNING..." => -1.523-J0.317
1, CHS, ENTER^, ENTER^, 2, ZF$ "ZPSIN" => "RUNNING..." => -0.131+J0.733
1, CHS, ENTER^, ENTER^, 3, LASTF       => "RUNNING..." =>  2.977+J0.317
```

Inverse Gamma Function and Catalan Numbers { ZIGAM , ZCTLN }

Here's the extension to the complex realm of the Inverse Gamma function first introduced in the SandMath module. Like its real variable counterpart, this is not a very useful beyond the academic interest: arter all, who needs to know what arguments yield a given gamma function result?

Well if you'd ever need to know, here's where you can get *some* answers - and I deliberately say some because in the complex plane this is a multi-valued function, which it's yet to be seen whether it has any formation rule for the different branche… but that, I guess, is another story altogether.

You can refer to the SandMath manual for a description of the algorithm used, which is applied directly here simply replacing the real functions with their complex counterparts.

The function is located in the -DELUXE section of the auxiliary FAT, and you can access it either by means of the sub-function launchers or via the extended "General Methods" launcher, ΣZL, [A], [R/S]

Example1: Obtain a complex value z which yields Γ(z) = 1+i

1, ENTER^, [Z], ALPHA, "ZIGAM" -> "RUNNING…"
0 0.412574972 – J 0.404915377

Note that the function follows an iterative process (Newton's method actually_. Each time an iteration is completed the program shows the module of the difference between the current and previous arguments, which when convergence exists it will be decreasing until it's less that the 1 E-8 tolerance used.

Example2: Use the ZLASTF feature to obtain which real value x yields Γ(x) = 2

[Z], 2, [Z], [,],[,] -> "RUNNING…"
0.442877396 – J 9.0000000 E-24

Complex Catalan Numbers

Based on the classic combinatorian definition, one can extend the concept using the Gamma function instead of the factorials as follows:

$$C_n = \frac{4^n \Gamma(n+1/2)}{\sqrt{\pi}\,\Gamma(n+2)},$$

Where n is a complex number (not neccessarily an integer). So we see it basically consistes of two calculations of the Gamma function, which in the 41Z module is conveniently implemented as an MCODE function – so a trivial FOCAL routine does the trick.

Examples: Obtain the C(n) values for n=1, 2, 3, and i+1

C(1) = 1.000000001 + J0
C(2) = 1.999999990 + J0
C(3) = 5.000000011 + J0
C(1+i) = 0.661301105 + J 0.443764974

10.1. Riemann's Zeta function. { ZZETA }

Included in the 41Z is an implementation of the Borwein algorithm to calculate the Zeta function. Considering the task at hand this does an excellent job, providing accurate results in acceptable execution times. Obviously won't win the speed contest, nor will it help you find non-trivial zeroes outside of the critical strip ☺

Example: calculate ζ(2)

```
2, ZREAL^, ZZETA        ->  1,645+J0
FIX 9                   ->  1,644934066
```

The program is a modified version of JM Baillard's ZETAZ, written for complex arguments – only adapted to use the 41Z complex stack and related functions. See the program listing in next page if interested. The algorithm is summarized as follows:

- For the case Re(z)<0.5 , 2 formulas may be used

  ζ(z) = ζ(1-z) 2^z π^(z-1) sin(π(z/2)) Γ(1-z)

  ζ(z) = ζ(1-z) π^(z-1/2) Γ((1-z)/2) / Γ(z/2)

- If Re(z) >=0.5

  ζ(z) = χ(z) / (1-2^(1-z))

where:

χ (z) = Σ{(-1)^k/k^z}, k=0,1,2,…

is calculated by:

χ (z) = (-1/dn) Σ{(-1)^k (dk-dn)/(k+1)^z}, k=0 to n-1

where:

dk = n Σ{(n+j-1)! 4^j)/((n-j)!(2j)!} , j=0 to k

with an error:

| e | <= (3/(3+sqrt(8))^n) [1+2 Im(z)] exp [p Im(z) / 2]

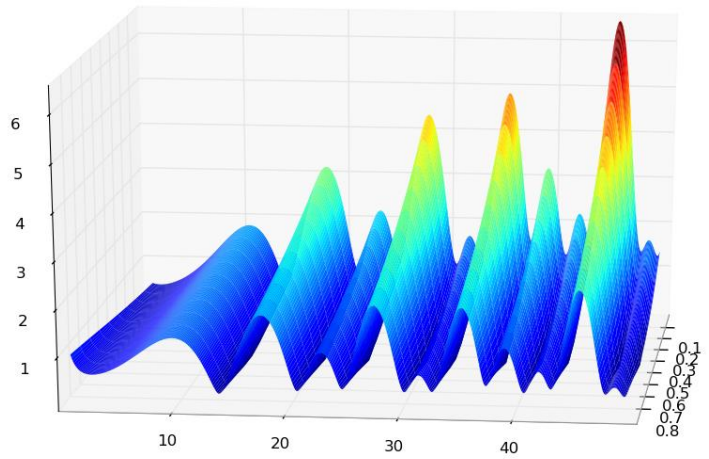
Note that dk is calculated using the following approach:

dk = e(0)+e(1)+...+e(k)

where :

e(0)=1 and

$$e(j+1) = \frac{2(n^2 - j^2)\, e(j)}{[(1+j)(2j+1)]}$$



FOCAL program for ZZETA:- Uses R00 to R11. No Flags used.

01	LBL "ZZETA"		51	ST+ X		101	X^2
02	X=0?		52	LN1+X		102	RCL 10
03	*GTO 00*		53	+		103	DSE X
04	*.5*		54	*3 E10*		104	NOP
05	CHS		55	LN		105	X^2
06	**ZAVIEW**		56	+		106	-
07	RTN		57	*8*		107	ST+ X
08	LBL OO		58	SQRT		108	/
09	CF 00		59	*3*		109	STO 03
10	**ZSTO 03**	R06 - Re(z)	60	+		110	ST+ 05
11	**ZSTO 00**	R07 - Im(z)	61	LN		111	DSE 10
12	*,5*		62	/		112	*GTO 01*
13	X<=Y?		63	INT		113	RCL 05
14	*GTO 00*		64	*E*		114	ST/ 08
15	SF 00		65	+		115	ST/ 09
16	SIGN		66	STO 10		116	RCL 07
17	-		67	STO 02		117	CHS
18	**ZNEG**	R06: - Im(z)	68	LASTX		118	STO 11
19	**ZSTO 03**	R07: 0,5-Re(z)	69	STO 11		119	RCL 06
20	**XEQ 00**		70	STO 03		120	CHS
21	**ZRCL 00**		71	STO 05		121	*2*
22	**ZNEG**		72	CHS		122	LN
23	*E*		73	X<>Y		123	*
24	+		74	Y^X		124	*E*
25	*2*		75	CHS		125	RAD
26	ST/ Z		76	STO 04		126	P-R
27	/		77	CLX		127	ENTER^
28	**ZGAMMA**		78	STO 08		128	DEG
29	**Z***		79	STO 09		129	*E*
30	**ZRCL 00**		80	LBL 01		130	ST+ 11
31	*,5*		81	**ZRCL 03**		131	-
32	-		82	**ZNEG**		132	RCL 11
33	PI		83	RCL 10		133	**2^X-1**
34	**X^Z**		84	**X^Z**		134	ST* Z
35	**Z***		85	RCL 05		135	X<> T
36	**ZRCL 00**		86	RCL 04		136	ST* T
37	*2*		87	CHS		137	ST+ T
38	**ST/ Z**		88	STO 04		138	RDN
39	/		89	*		139	+
40	**ZGAMMA**		90	ST* Z		140	**ZST/ 04**
41	**Z/**		91	*		141	**ZRCL 04**
42	**ZAVIEW**		92	**ZST+ 04**		142	FC? 00
43	RTN		93	RCL 10		143	**ZAVIEW**
44	LBL 00		94	ENTER^		144	END
45	PI		95	ST+ Y			
46	*2*		96	ST* Y			
47	/		97	-			
48	RCL 06	Re(z)	98	RCL 03			
49	ABS		99	*			
50	ST* Y		100	RCL 02			

10.2 Lambert W function. { ZWL , ZAWL }

ZWL	Lambert W(z)		FOCAL program
ZAWL	Inverse of Lambert-W	**z* e^z**	Does LastZ

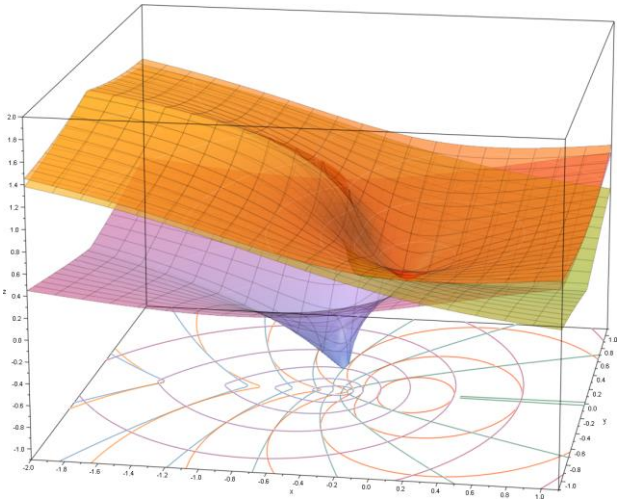
These two functions provide a dedicated way to compute the Lambert-W function and its inverse.
The FOCAL program uses an iterative method to compute W(z), using z0=1+Ln(z) as initial guess for Re(z)>0, and simply z0=(1+ i) elsewhere.

This program is based on a real-mode version written by JM Baillard, just applying the seamless transposition method provided by the 41Z module. In the vast majority of cases convergence is provided for all complex arguments, with 8-decimal digits accuracy. It uses the **Z=WR?** Function on FIX 8 mode to determine that two consecutive iterations are equal.

The inverse function is a simple product: $W^{-1}(z) = z * e^z$.

Not worth the FAT entry, you say? For one thing, doing it in MCODE allows for 13-digit accuracy in the calculations. Besides, how often will you forget the exact formula? Better safe than sorry…

```
01  LBL "ZWL"
02  Z=0?
03  GTO 00
04  ZSTO (00)
05  E
06  +
07  Z=0?
08  ISG Y(2)
09  ZLN
10  FIX 8
11  LBL 01
12  ZREPL
13  ZNEG
14  ZEXP
15  ZRCL (00)
16  Z*
17  Z-
18  Z<>W
19  E
20  +
21  Z/
22  Z-
23  Z=WR?
24  GTO 00
25  GTO 01
26  LBL 00
27  FIX 3
28  ZAVIEW
29  END
```



Note that **ZWL** is a FOCAL program, and thus you won't be able to use LASTZ to recover the initial argument. This is common to all the function simplemented as FOCAL routines instead of full MCODE functions.

Examples. Calculate W(1+i) and trace back the original argument using the inverse function.

1, ENTER^, XEQ "**ZWL**" -> 0.657+J0.325
ZAWL -> 1.000+J1.000

Another version using SOLVE is listed in section 12.2, with slightly more accurate results , but significantly slower execution and a few trouble spots (near 1/e and -1/e).

11. Complex Means, Elliptic integrals and DFT.

This section covers the MCODE functions to calculate Arithmetic, Harmonic and Geometric single Means on a set of data, plus the dual means AGM and GHM of two complex arguments. These are related to the Elliptic Integrals, also obtained via the Hypergeometric function and other dedicated formulas.

Table 10.1. Complex Means Functions.

ZAMN	Complex Arithmetic Mean	Control word in X	bbb.eee
ZHMN	Complex Harmonic Mean	Control word in X	bbb.eee
ZGMN	Complex Geometric Mean	Control word in X	bbb.eee
ZAGM	Complex Arithmetic-Geometric Mean	Arguments in Z, W	Does LastZ
ZGHM	Complex Geometric-Harmonic Mean	Arguments in Z, W	Does LastZ

For the single means the data is expected to be stored in a contiguous set of Compex Data registers, ZRbbb to ZReee. You can use the utility **ZINPT** to populate those registers. The functions require the control word in the X-register to define the register range for the calculation.

Example1. Calculate the three single means for the set of complex values stored in the following data registers: ZR00 = -1 – i ; ZR01 = 1 + i ; ZR02 = 2 + 2i ; ZR03 = 3 + 3i

0.003, **ZF$** "ZAMN" => 1.250(1+J)
0.003, **ZF$** "ZHMN" => 4.800(1+J)
0.003, **ZF$** "ZGMN" => 2.213+J0

For the dual means, the same definitions for real numbers hold in the complex plane. There's no special considerations to the Arithmetic and harmonic means of complex arguments, but since the n-th root is used in the Geometric mean, it'd have a multi-value result. This becomes of singular importance in the calculation of the arithmetic-geometric mean of two values, as the convergence has many different paths – all leading to different final results.

The implementation uses the following criteria for chosen value of the geometric mean, c= sqr(ab) (see: https://www.math.leidenuniv.nl/scripties/carls.pdf): *if it is the "correct" square root for the geometric mean step, then Im[c/(a+b)] is strictly positive (i.e. > 0), otherwise replace c with -c.* Furthermore this imposes the condition that Im(c) and Im(a+b) have the same sign.

Example 2. Calculate the dual means AGM and GHM for the complex pair: z= 2-4i and w= -3+i

4, CHS, ENTER^, 2, **ZENTER^** 2-J4
1, ENTER, 3, CHS, XEQ "**ZAGM**" => -1.343-J2.146

Exact result:

$$\text{agm}(-3+i, 2-4i)$$

Decimal approximation:

$$-1.343237282754998341806890738047584266195139297354352475 5\ldots - 2.145601168378194819897575380210627218579039669296930165 5\ldots i$$

4, CHS, ENTER^, 2, **ZENTER^** 2-J4
1, ENTER, 3, CHS, **ZF$** "ZGHM" => -4.268-J3.604

Which verifies the known relationship: $$M(x,y) = \frac{1}{AG(\frac{1}{x}, \frac{1}{y})}$$

11.1 Complex Elliptic Integrals.

Table 10.2. Complex Elliptic Integrals.

ZELIP1	Incomplete Elliptic Int. 1st kind	Complex Argument in {Z,Y}, real modulus in X	Needs SandMath
ZELIP2	Incomplete Eliiptic Int. 2nd kind		
ZELIPE	Complete Ellipt. Int. 2nd kind	Complex Modulus in "Z"	Uses ZHGF
ZELIPK	Complete Ellipt. Int. 1st kind		
ZELK	Uses ZAGM for Complete Ellip.Int.		Uses ZAGM
ZELPKE	Both Complete Integrals		JM Baillard

The Elliptic integrals are covered in several FOCAL programs as shown in the table above. Note that:

- For the Incomplete types the amplitude can be a complex number but the modulus is expected to be a real value. This method uses dedicated formulas that apply the real expressions on a repeated basis according to changes of variable , *and it requires the SandMath module to be plugged in as well*. Here the function name **ZELIP1** corresponds to F(z; m) , and **ZELIP2** corresponds to E(z; m).

- for the Complete types (where the amplitude is therefore 90 degrees) the modulus can be a complex number. Here two methods are available, one based on the hypergeometric function (slower and requires |modulus|<1), and another based on the complex AGM – faster and without that restriction.

- No provision is made for the case where both amplitude and modulus are complex numbers. To check the results you can use the syntax "EllipticF" and "EllipticE" on WolframAlpha using two arguments for incomplete cases or just one argument for complete cases.

Let's see a few examples next. Be aware that the execution time can range from long to very long depending on the case. You can abort the execution pressing the R/S key at any time.

Example1: calculate the complete Elliptic integrals for a = 2+3i

The first thing we notice is that |z|>1, thus the hypergeometric method is not going to converge – so discard using **ZELIPE** and **ZELIPK**. Being based on the AGM method, function **ZELK** is the faster way to obtain the 1st. kind resut - but using **ZELPKE** we can get both results on a single execution as follows:

3, ENTER^, 2, **ZF$** "ZELPKE" => 1.043+J0.630
Z<>W => 1.473-J1.232

Example 2. Calculate the incomplete Elliptic integrals for a= 1-i, m=0.5

1, CHS, ENTER^, CHS, ENTER^, .5, **ZF$** "ZELIP1" => 0.804+J1.163
EllipticF(1-i, .5): http://www.wolframalpha.com/input/?i=EllipticF%281-i,+.5%29

1, CHS, ENTER^, CHS, ENTER^, .5, **ZF$** "ZELIP2" => 1.128+J0.789
EllipticE(1-i, .5): http://www.wolframalpha.com/input/?i=EllipticE%281-i,+.5%29

Formulas used (from Abramowitz-Stegun, Section 14.4)

Writing z= (phi+ i psi) then we have for the first kind:

$$F(\phi + i\psi|m) = F(\lambda|m) + iF(\mu|1-m)$$

Where cot^2 (λ) is the positive root of the quadratic equation:

$$x^2-[\cot^2\varphi+m\sinh^2\psi\csc^2\varphi-m_1]x-m_1\cot^2\varphi=0$$
$$\text{and } m\tan^2\mu=\tan^2\varphi\cot^2\lambda-1.$$

And similarly for the second kind integral:

$$E(\varphi+i\psi\backslash\alpha)=E(\lambda\backslash\alpha)-iE(\mu\backslash 90°-\alpha)+iF(\mu\backslash 90°-\alpha)+\frac{b_1+ib_2}{b_3}$$

where now:

$$b_1=\sin^2\alpha\sin\lambda\cos\lambda\sin^2\mu(1-\sin^2\alpha\sin^2\lambda)^{\frac{1}{2}}$$
$$b_2=(1-\sin^2\alpha\sin^2\lambda)(1-\cos^2\alpha\sin^2\mu)^{\frac{1}{2}}\sin\mu\cos\mu$$
$$b_3=\cos^2\mu+\sin^2\alpha\sin^2\lambda\sin^2\mu$$

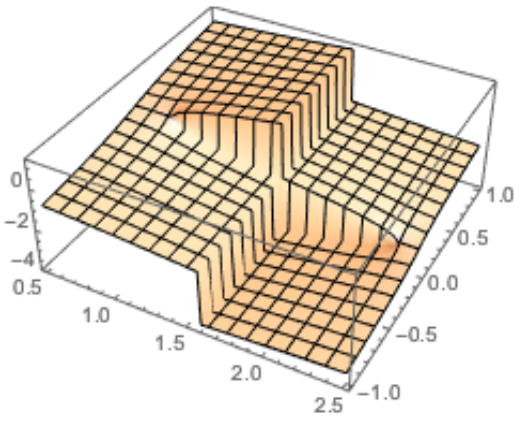
as you can see an elaborate set of equations that requires a relatively long FOCAL program even if some functions from the SandMath really expedite things significantly. Refer to next page for the FOCAL program listing as a reference.

The solution is therefore expressed as a linear combination of the real-variable case for the Elliptic integrals, which are also included in the SandMath as functions **ELIPF** and **LEI1** and **LEI2**.

The elliptic integrals have a complicated structure in the complex plane.

```
In[83]:= Plot3D[Im[EllipticF[px + I py, 2]], {px, 0.5, 2.5}, {py, -1, 1}, PlotPoints -> 60]
```

Out[1]=



Program Listing: Incomplete Elliptic Integrals. *(SandMath required.)*

Data Registers: R00-R08 ; User flag: F1

```
01  LBL "ZELIP1"
02  SF 01
03  GTO 00
04  LBL "ZELIP2"
05  CF 01
06  LBL 00
07  RAD
08  STO  00
09  RDN
10  STO  01
11  RDN
12  HSIN
13  X^2
14  /
15  RCL  00
16  E
17  -
18  *
19  E
20  STO T(0)
21  RDN
22  QROOT
23  X<Y?
24  X<>Y
25  STO 02
26  RCL 01
27  TAN
28  X^2
29  *
30  E
31  -
32  RCL 00
33  /
34  SQRT
35  ATAN
36  STO 01
37  E
38  RCL 00
39  -
40  RCL 01
41  ELIPF
42  FS? 01
43  GTO 00
44  STO 01
45  RCL 00
46  RCL 02
47  SQRT
48  1/X
49  ATAN
50  ELIPF
51  STO 00
52  ZRCL 00
53  ZAVIEW
54  RTN
55  LBL 00
56  STO 08
57  RCL 00
58  RCL 02
59  SQRT
60  1/X
61  ATAN
62  STO 02
63  SIN
64  X^2
65  *
66  E
67  -
68  CHS
69  STO 03
70  SQRT
71  RCL 00
72  *
73  RCL 02
74  E
75  P-R
76  *
77  *
78  RCL 01
79  SIN
80  X^2
81  *
82  RCL 01
83  SIN
84  X^2
85  STO 06
86  RCL 00
87  SQRT
88  ASIN
89  COS
90  RCL 01
91  SIN
92  *
93  X^2
94  CHS
95  E
96  +
97  SQRT
98  RCL 03
99  *
100 RCL 01
101 E
102 P-R
103 *
104 *
105 RCL 02
106 STO 05
107 SIN
108 *
109 X^2
110 RCL 00
111 *
112 RCL 01
113 COS
114 X^2
115 +
116 ST/ 06
117 ST/ 07
118 RCL 08
119 ST+ 07
120 E
121 RCL 00
122 STO 08
123 -
124 RCL 01
125 LEI2
126 ST- 07
127 RCL 08
128 RCL 05
129 LEI2
130 ST+ 06
131 ZRCL 03
132 ZAVIEW
133 END
```

Granted, this listing doesn't have much of a 41Z flavor to it since it really operates on real variable functions. Pulling all stops with the aid of the SandMath we deflect the complex variable with linear combinations as per the formulas shown before.

Program Listing: Complete Elliptic integrals

Data registers: R00-R08 ; no user flags.

```
01  LBL "ZELPKE
02  ZNEG
03  ZSTO 02
04  E
05  STO 08          counter
06  STO 00          real part
07  +               1-z
08  ZSQRT           sqr(1-z)
09  ZSTO 01         initial value
10  CLX
11  STO 01          initial z0 = (1+0i)
12  LBL 01
13  ZRCL 00
14  ZRCL 01
15  Z-
16  2
17  ST/ Z
18   /              arithmetic mean
19  Z^2             AM^2
20  2
21  RCL 08          k
22  Y^X             2^k
23  ST* Z
24  *
25  ZST- 02
26  ZRCL 00
27  ZRCL 02          -z - (2^k *AM^2)
28  Z+
29  2
30  ST/ Z
31   /              halves it
32  ZSTO 03
33  ZRCL 01
34  ZRCL 00
35  Z*
36  ZSQRT           Geometric Mean, GM
37  ZSTO 01
38  ZRCL 03
39  ZSTO 00
40  CLX
41  SIGN
42  ST+ 08
43  RCL 08
44  8
45  X>Y?
46  GTO 01          -63 bytes
47  ZRCL 00
48  0
49  2
50  ST* Z
51  *               doubles it
52  ZINV
53  ZPI*            more accurate
54  ZSTO 00
55  ZRCL 02
56  2
57  ST/ Z
58   /              halves it
59  E
60  +
61  Z*
62  ZSTO 01
63  ZRCL 00
64  ZAVIEW
65  END
```

Upon completion both complete integrals of the 1st and 2nd kinds are left in the complex stack levels Z and W. They're also saved in ZR00 and ZR01 respectively.

Note.- Many of these functions appear on CAT'2 as M-Code entries, instead of as FOCAL programs. This is achieved by using a clever technique shown by W. Doug Wilder (author of the BLDROM), which allows cleaner and convenient program listings (no ugly "XROM" description before the program title). These programs however cannot be copied into main memory using COPY. Another drawback is that frequently they are interpreted as PRIVATE by the 41 OS, nor could they be "looked-up" using GTO + global LBL, since there's no global LBL for them.

11.2 Complex Discrete Fourier Transform {[**ZDFT**], [**ZIDFT**[}.

An interesting subject on its own right, the Discrete Fourier Transform has had little coverage on the 41 platform – perhaps the single exception being JM Baillard's Spectral Analysis pages. A reason for this scarcity may be the slow CPU speed, rendering the applicability to just academic cases for small sets of data. The advent of the 41CL and or course SW emulators make this less of an issue, as the examples below will show.

On the 41Z Deluxe the direct and inverse DFT are implemented entirely as MCODE functions. The "n" data points are expected to be in contiguous Complex Data registers, starting with ZRbbb to ZReee. Then you enter the control word "bbb.eee" - complex indexes - in the X register and call the function. You can use **ZINPT** to enter those values in memory.

When the execution completes the transformed data values are placed in the following block of Complex data registers {ZR(eee+1) to ZR(eee+n)}, and the new control word is left in X – so you can use **ZOUPT** to review the results.

This implementation just scratches the surface of the topic. It uses the straight-forward definition for the transform (not fast algorithms like in the FFT case). The code however has several shortcuts to accelerate the calculations when any of the indexes are zero – which results in an exponential value equal to one. See the formulas below for the direct (left) and inverse (right) cases.

$$X_k \overset{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i k n / N}, \quad k \in \mathbb{Z} \qquad x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{2\pi i k n / N}, \quad n \in \mathbb{Z}$$

Unavoidably rounding errors are the reason that some result values won't show as integers. This is an inherent limitation of the 10-digit accuracy, which unfortunately can't be extended to 13-digit in many areas despite being written in MCODE.

Example. Calculate the DFT for the set of values in the left colum below. (see application at: http://calculator.vhex.net/calculator/fast-fourier-transform-calculator-fft/1d-discrete-fourier-transform).

i	a_i
1	1 + 2j
2	3 + 4j
3	5 + 6j
4	7 + 8j
5	9 + 10j
6	11 + 12j
7	13 + 14j
8	15 + 16j

Result :

i	
1	64 + 72j
2	-27.313708 + 11.313708j
3	-16 + 0j
4	-11.313708 - 4.686292j
5	-8 - 8j
6	-4.686292 - 11.313708j
7	0 - 16j
8	11.313708 - 27.313708j

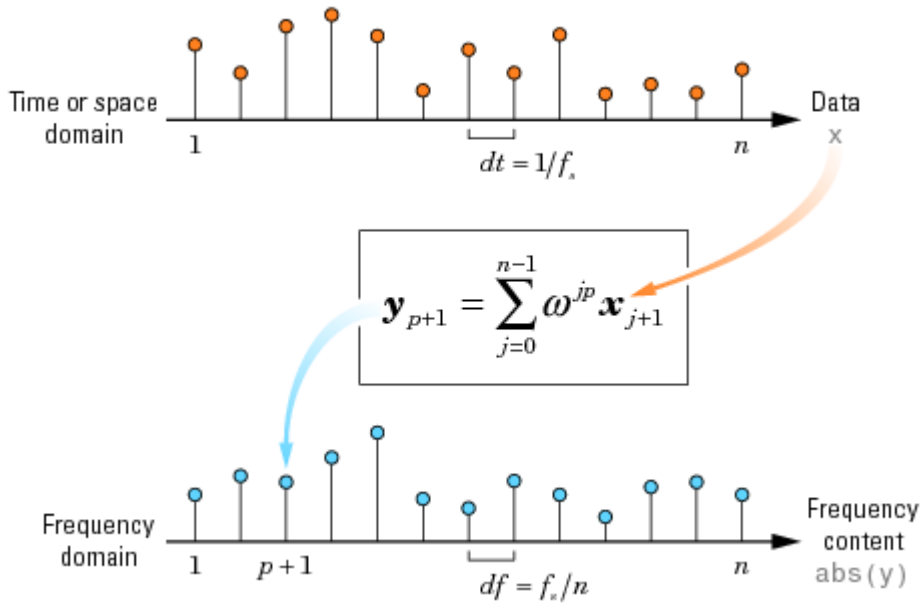
To obtain the original sample (assuming complex regs C01 – C08 were used) you can do:
18,002016, REGMOVE, 1.008, XEQ "ZIDFT"

A sample size of 8 complex values takes about 25 seconds to complete on a normal-speed HP-41, and just shy of 1 second on the 41CL at Turbo-50; not bad for such a venerable machine.

The FOCAL program below is a rough equivalent of the MCODE function. Execution times for this program are about four to five times longer than the MCODE counterpart.

```
01  LBL "ZDFT"
02  CF 01
03  GTO 00
04  LBL "ZIDFT"
05  SF 01
06  *LBL 00
07  STO 00          N
08  E3/E+
09  STO M(5)        j,00N
10  *LBL 01         outer loop
11  VIEW M(5)
12  RCL 00          N
13  STO N(6)
14  E3/3+
15  STO O(7)        k,00N
16  RCL 5(M)        j,00N
17  INT             j
18  ST+  N(6)       dest: ZR(N+j)
19  E
20  -               j-1
21  PI
22  *
23  ST+ X(3)        2p.(j-1)
24  RCL 00          N
25   /              2p.(j-1)/N
26  STO 01
27  CLZ
28  ZSTO IND N(6)   reset destination
29  *LBL 02         inner loop
30  RCL 0(7)        k,00N
31  INT             k
32  E
33  -               k-1
34  RCL 01          2p.(j-1)/N
35  *               2p.(j-1)(k-1)/N
36  FC? 01
37  CHS
38  E
39  P-R
40  ZRC* IND O(7)
41  ZST+ IND N(6)
42  ISG O(7)        next k
43  GTO 02          loop back
44  FC? 01
45  GTO 00
46  ZRCL IND 01
47  RCL 00
48  ST/ Z
49   /
50  ZSTO IND 01
51  *LBL 00
52  ISG M(5)        next j
53  GTO 01          loop back
54  END
```

The functions will check that enough data registers are available. If not, the "NONEXISTENT" message will be presented; adjust the size and try again. Make sure complex data register ZR00 is not used to store the sample – which must start at ZR01. This is because (real) data registers R00 and R01 are used for scratch calculations by these functions.



12. Complex General Methods.

Most of the following functions are complex versions of general methods, included either to illustrate actual programming of the complex number functions of the module or to provided a parallel environment to the real-variable case.

	Function	Description	Author
	ZMTV	Multi-valued functions	ÁM
	ZSOLVE	Solves f(z)=0 by secant method	ÁM
	ZNWT	Complex Step (Real) Differentiation	ÁM
	ZHALL	Solves f(z)=0 by Halley's method	ÁM
	ZDERV	Complex 1st & 2nd Deriviatives	Greg McClure
	ZCF2V	Complex Continued Fractions	Greg McClure
	ZCSX	Fresnel Integrals.	JM Baillard
	ZKLV1	Weber & Anger functions	JM Baillard

12.0 Real Functions as Complex Extensions { ZCSX , ZKLV1 }

Here's an interesting approach to the calculation of some real-variable functions, treated as the real and imaginary parts of a complex extension that uses complex-variable arguments. Two examples are included:

1. The Kelvin functions of 1st kind, ber(n, x) & bei(n, x); and
2. The Fresnel Integrals, C(x) and S(x)

The expressions are based on the hypergeometric function, which also in the complex variable becomes a real power horse of high applicability for the programming of the routines.

$$ber_n(x) + i\, bei_n(x) = (x\,(i-1)/\mathrm{sqrt}(8))^n \;\; {}_0\mathbf{F}_1(\, n+1 \,;\, i\, x^2/4\,) \;\; / \;\; \Gamma(n+1)$$

$$c(x) + i\, s(x) = x \;\; {}_1\mathbf{F}_1(\, 1/2 \,;\, 3/2 \,;\, i\, \pi\, x^2/2\,)$$

Note that the input parameters are real values, and thus are expected to be in the real stack X- and Y- registers. The output will show a complex number, where it's to be understood it reflects the two solutions arranged as real and imaginary parts.

Example1: Calculate the Kelvin functions for x= π and n=sqrt(2)

2, SQRT, PI, **ZF$** "ZKLV1" -> " RUNNING…" => -0.674-J1.597
FIX 9 for $ber_{sqrt(2)}(\pi)$ => -0.674095956
X<>Y for $bei_{sqrt(2)}(\pi)$ => -1.597357212

Example2: Calculate the Fresnel Integrals for x= 1.4

1.4, **ZF$** "ZCSX" -> "RUNNING…" => 0.543+J0.714
FIX 9 for C(1,4) => 0.543095784
X<>Y for S(1.4) => 0.713525077

Note that x must remain "small", say x < 2. For x = 3 , the errors are of the order of 10 -6 and the results are meaningless with x = 4

12.1 Multi-valued Functions. { ZMTV }

ZMTV	Multi-valued functions		

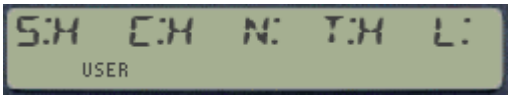
This program calculates all possible values for the multi-valued functions, including the n different N$^{th.}$ roots of a complex number, all the inverse trigonometric and hyperbolic, plus the logarithm itself (source of all the multi-valued scenarios).

Due to the 64-function limit of the 41 ROM FAT structure. these routines are all part of a common entry into the module catalog. To access it you use the **ZNEXT** prompt, followed by the XEQ key – i.e:
[**Z**], [**"A"**], [SHIFT], [**"K"**]

When invoked, the program prompts a menu of choices as follows:

A – ASIN	B – ACOS	C: Nth. Root	D: ATAN	E: Ln
a – HSIN	b.- HACOS		d.- HATAN	

Or more succinctly:



For each case the program will calculate the principal value followed by all the other values with each subsequent pressing of [R/S]. Remember that the top keys need to be free from user assignments for this scheme to work, as per the 41 OS conventions.

All trigonometric functions expect *z* into the **Z** level of the complex stack. Data entry is the same for all of them except for the n-th root, which expects N in the real-stack register X, and *z* in **Z**. Only the first N values will be different, running into cyclical repetition if continued.

This is a simple program, mostly written to document an example for the 41Z functions. Use it to get familiar with these concepts, and to understand fully the NXT function set as well.

Note that in version 9L the FAT entry for **ZMTV** was removed – the same functionality exists accessed via the launcher menus. Refer to the following sections for details.

Example: Obtain all values of ASIN [Sin(1+j)]

1, ENTER^, **ZSIN**	-> 1,298+j0,635
ZMTV	-> "S:H C:H N: T:H L:"
A	-> 1,000+j1
R/S	-> 2,142-j1
R/S	-> 7,283+j1
R/S	-> 8,425-j1
etc…	

Alternatively, using the **NXTASN** function:

Note that here we start with the first value of the function, i.e. 1+j

1, ENTER^, **NXTASN**	-> 7,238+j1
Z<>W	-> 2,142-j1
NXTASN	-> 8,425-j1
NXTASN	-> 14,708-j1

Program listing.- Alternative older version, superseded in revision 4L.

Note the use of flag 22 for numeric entry: the catalog of functions will display continuously until one choice is made, (expected between 1 and 8), and all initial prompting will be skipped.

```
 1  LBL "ZMTV"
 2  CF 22
 3  LBL 20
 4  "FCN#.=? 1-8
 5  AVIEW
 6  PSE
 7  PSE
 8  FC? 22
 9  GTO 90
10  INT
11  ABS
12  90
13  +
14  RDN
15  SF 25
16  GTO IND T
17  GTO 20
18  LBL 90
19  CF 21
20  "1:- ZACOS"
21  AVIEW
22  PSE
23  "2:- ZACOSH"
24  AVIEW
25  PSE
26  "3:- ZASIN"
27  AVIEW
28  PSE
29  "4:- ZASINH"
30  AVIEW
31  PSE
32  "5:- ZATAN"
33  AVIEW
34  PSE
35  "6:- ZATANH"
36  AVIEW
37  PSE
38  "7:- ZLN"
39  AVIEW
40  PSE
41  "8:- Z^1/N"
42  AVIEW
43  PSE
44  GTO 20
45  LBL 95
46  ZATAN
47  GTO 06
48  LBL 93
49  ZASIN
50  ZSTO 00
51  ZAVIEW
52  E
53  STO 02
54  LBL 03
55  ZRCL 00
56  ZNEG
57  ZSTO 00
58  RCL 02
59  PI
60  *
61  +
62  ZAVIEW
63  PSE
64  E
65  ST+ 02
66  GTO 03
67  LBL 91
68  ZACOS
69  ZSTO 00
70  ZAVIEW
71  E
72  STO 02
73  LBL 01
74  ZRCL 00
75  RCL 02
76  ST+X
77  PI
78  *
79  STO 03
80  +
81  ZAVIEW
82  PSE
83  ZRCL 00
84  ZNEG
85  RCL 03
86  +
87  ZAVIEW
88  PSE
89  E
90  ST+ 02
91  GTO 01
92  LBL 94
93  ZHASIN
94  GTO 07
95  LBL 92
96  ZHACOS
97  GTO 07
98  LBL 96
99  ZHATAN
100 LBL 06
101 ZAVIEW
102 PSE
103 PI
104 +
105 GTO 06
106 LBL 97
107 ZLN
108 LBL 07
109 ZAVIEW
110 PSE
111 NXTLN
112 GTO 07
113 LBL 98
114 CF 00
115 "N=?"
116 PROMPT
117 ABS
118 INT
119 X=0?          zeroth. Root?
120 RTN
121 STO 00
122 E
123 -             N-1
124 STO 01
125 X=0?
126 SF 00         unit root?
127 E
128 +             N
129 1/X           1/N
130 Z^X           main value
131 SF 21
132 ZAVIEW
133 FS?C 00
134 GTO 08
135 LBL 05
136 RCL 00
137 NXTNRT
138 ZAVIEW
139 DSE 01
140 GTO 05
141 LBL 08
142 CF 21
143 END
```

12.2 Solution to f(z)=0. { ZSOLVE , ZHALL }

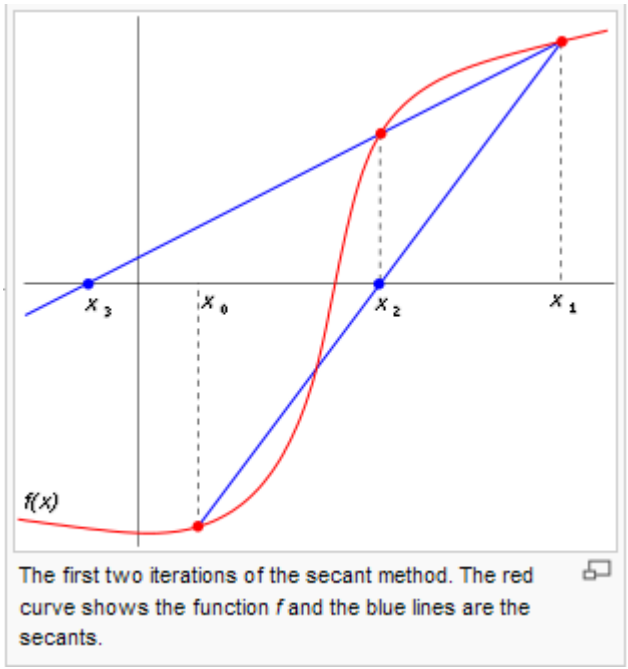
The next application uses the Secant Method to obtain roots of a complex equation, given two estimations of the solution. A general discussion on root-finding algorithms is beyond the scope of this manual – this example is intended to show the capabilities of the 41Z module, in particular how programming with complex numbers becomes as simple as doing it for real numbers using the native function set.

See the following link for further reference on this subject (albeit just for real variable):
http://en.wikipedia.org/wiki/Secant_method

The secant method is defined by the recurrence relation:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n).$$

which will be calculated until there's no significant contribution to the new value – as determined by the function **Z=WR?**.



The first two iterations of the secant method. The red curve shows the function *f* and the blue lines are the secants.

Program listing:-

As it's the case with this type of programs, the accuracy of the solution depends of the display settings, and the convergence (i.e. likelihood to find a root) will depend on the initial estimations.

The program works internally with 8-digit precision, therefore will largely benefit from the turbo-mode settings on V41 to dramatically reduce the execution time.

1	LBL "ZSOLVE"	20	Z<>W	39	Z-
2	FS? 06	21	ZSTO (00)	40	Z*
3	GTO 06	22	XEQ IND 06	41	ZNEG
4	AON	23	ZSTO	42	ZRCL
5	"F. NAME=?"	24	2	43	1
6	PROMPT	25	LBL 01	44	Z+
7	AOFF	26	ZRCL	45	ZENTER^
8	ASTO 06	27	1	46	Z<>
9	PREC=?	28	XEQ IND 06	47	1
10	PROMPT	29	ZREPL	48	Z=WR?
11	FIX IND X	30	Z<>	49	GTO 02
12	"Z1=? (Y^X)"	31	2	50	GTO 01
13	PROMPT	32	Z-	51	LBL 02
14	ZENTER^	33	Z#0?	52	FC? 06
15	"Z2=? (Y^X)"	34	Z/	53	FIX 3
16	PROMPT	35	ZRCL	54	FC? 06
17	LBL 06	36	1	55	ZAVIEW
18	ZSTO	37	ZENTER^	56	END
19	1	38	Z<> (00)		

User flag 06 is for subroutine usage: when set, the data input will be skipped. In that case the relevant data is expected to be in the appropriate registers, as follows:

ZR03= Initial estimation z1,
ZR04 = initial estimation z2
R12 = Function's name,
FIX set manually to required precision.

Example 1.- Calculate one root of the equation: **Sinh(z) + z^2 + pi = 0**

Which we easily program using 41Z functions as follows:

LBL "ZT", **ZHSIN, LASTZ, Z^2, Z+,** PI, +, END.

Using the initial estimations as z0=0, and z1=1+i, we obtain:

Root = -0,27818986 + j 1,81288037

Example 2.- Calculate two roots of the equation: **e^(z) = z**

programmed as follows: LBL "ZE", **ZEXP**, **LASTZ**, **Z-**, END

using the estimations: {z0=-1-j & z1=1+j} - note that both roots are conjugated!

Root1 = 0,3181315 + j 1,3372357
Root2 = 0,3181315 - j 1,3372357

Example 3.- Calculate the roots of the polynomials from section 10.1 and 10.3:

$P2 = (1+i)*z^2 + (-1-i)*z + (1-i)$
$P3 = z^3 + z^2 + z + 1$
$P4 = (1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z - 1$

Re-written using the Honer's method as follows:

P2 = z [(-1-i) - z(1+i)] + (1-i)
P3 = z [1 + z(1+z)] +1
P4 = z {1 + z [(3-3i) – z [(1+2i) - z(1+2i)]] } - 1

Use the following estimations for the P4 example:-

{z0=-1-j ; z1=1+j} for root #1 ; {z0=1+j ; z1=2+2j} for root #2,
{z0=-2j ; z1= 2j} for root #3 ; {z0= 4j ; z1= 5j} for root #4

ZSOLVE Register Usage.

Notice that to avoid register incompatibilities **ZSOLVE** uses complex registers ZR03 – ZR06 (i.e. registers R06 – R12). This allows its direct application to calculate zeroes of functions using the lower register range (which is the typical case), like the Exponential integral and associates, which in turn all use complex registers ZR00 – ZR02 (i.e. R00 - R05) . This removes the need to use cumbersome REGMOVE program steps with its memory-hungry control words.

The programs below can be used to obtain the roots as per the examples given before:

$(1+i)*z^2 + (-1-i)*z + (1-i) = 0$		
1		LBL " Z2"
2		ZREPL
3		E
4		ENTER^
5		Z*
6		ZENTER^
7		-1
8		ENTER^
9		Z+
10		Z*
11		ZENTER^
12		-1
13		ENTER^
14		CHS
15		Z+
16		END

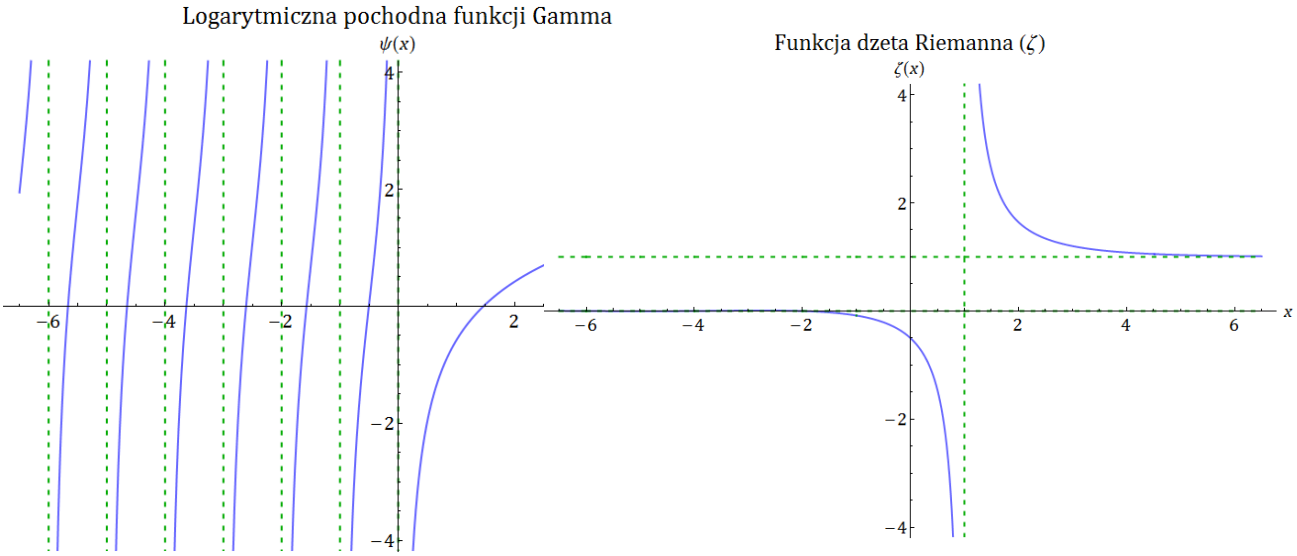
$Z^3 + Z^2 + Z + 1$		
1		LBL " Z3"
2		ZREPL
3		1
4		+
5		Z*
6		1
7		+
8		Z*
9		1
10		+
11		END

$(1+2i)*z^4 + (-1-2i)*z^3 + (3-3i)*z^2 + z - 1$		
1		LBL " Z4"
2		ZREPL
3		2
4		ENTER^
5		1
6		Z*
7		LASTZ
8		Z-
9		Z*
10		ZENTER^
11		-3
12		ENTER^
13		CHS
14		Z+
15		Z*
16		1
17		+
18		Z*
19		1
20		-
21		END

*Note the usage of stack-lifting functions to separate entries (**LASTZ** and **ZENTER^**)*

1		LBL " Z4"
2		ZREPL
3		4
4		Z^X
5		ZENTER^
6		2
7		ENTER^
8		1
9		Z*
10		Z<>W
11		3
12		Z^X
13		ZENTER^
14		-2
15		ENTER^
16		-1
17		Z*
18		Z+
19		Z<>W
20		Z^2
21		ZENTER^
22		-3
23		ENTER^
24		CHS
25		Z*
26		Z+
27		Z+
28		1
29		-
30		END

Lastly, a few other excellent programs written by Jean-Marc Baillard address the general solution to the equation f(z)=0. They don't use functions from the 41Z module, but are mentioned here for their obviously close related content. The programs can be found at the following link: http://www.hpmuseum.org/software/41/41cmpxf.htm



Application example.- Using ZSOLVE to calculate the Lambert W function.

In this example we see a few techniques applied together, combining the capabilities of the 41Z in a convenient way. The solution is a direct application of the definition, requiring very simple extra programming – albeit with the logical slow performance.

The Lambert *W* function is given by the following functional equation:

$z = W(z)\, e^{W(z)}$, for every complex number *z*.

Which cannot be expressed in terms of elementary functions, but can be properly written with the following short program:

1		LBL "ZWL"
2		ZSTO
3		4
4		ZLN
5		ZENTER^
6		E
7		+
8		SF 06
9		"^WL"
10		ASTO 06
11		ZSOLVE
12		ZAVIEW
13		RTN
14		LBL "^W"
15		ZEXP
16		LASTZ
17		Z*
18		ZRCL
19		4
20		Z-
21		END

The complex value is expected to be in the **Z** complex stack level, and X,Y registers upon initialization. Set the FIX manually for the required precision.

Because **ZSOLVE** uses all the complex stack levels and registers 0 to 6 (Note: *this was changed in revision 4L – see pg. 59*) , the argument is saved in the complex register 4 – corresponding to real registers 8 and 9, thus a SIZE 10 or higher is required (see register correspondence map below).

We solve for W(z)=z, using as the function initial estimations the logarithm of the same argument and the same point plus one, perhaps not a refined choice but sufficient to ensure convergence in the majority of cases. Some calculated values are:

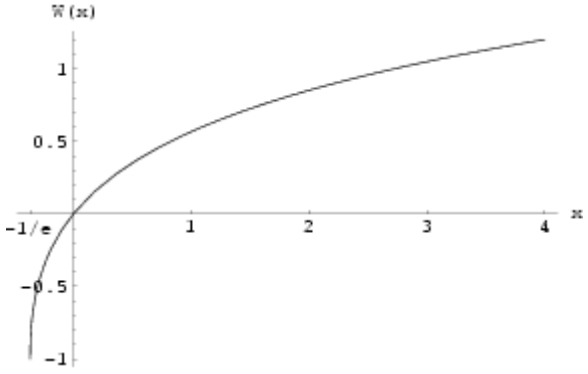
$$W(0) = 0$$
$$W(1) = \Omega \approx 0.56714329\ldots$$
$$W(e) = 1$$
$$W(-1) \approx -0.31813 - 1.33723i$$

This example is not meant to compete with a dedicated program using an iterative algorithm, yet it showcases the versatility of the approach. The obvious speed shortcomings are diminished when ran on the 41CL or modern emulators like V41.

The Taylor series of *W*₀ around 0 is given by:

$$W_0(x) = \sum_{n=1}^{\infty} \frac{(-n)^{n-1}}{n!}\, x^n$$

Another technique (somehow a brute-force approach) would employ this definition to calculate successive terms of the summation until their contribution to the sum is negligible. This method would only be applicable within the convergence region.



See the following links for further references on the Lambert W function:
http://en.wikipedia.org/wiki/Lambert_W_function
http://mathworld.wolfram.com/LambertW-Function.html

12.3. Newton's Method with Complex Step Differentiation.

This method is used to calculate real function derivatives, just as a quasi-magical application of complex variables. Complex step differentiation is a technique that employs complex arithmetic to obtain the numerical value of the first derivative of a real valued analytic function of a real variable, avoiding the loss of precision inherent in traditional finite differences. This is then used n Newton's method in the usual way.

We're concerned with an *analytic* function. Mathematically, that means the function is infinitely differentiable and can be smoothly extended into the complex plane. Computationally, it probably means that it is defined by a single "one line" formula, not a more extensive piece of code with if statements and for loops.

Let $F(z)$ be such a function, let x_0 be a point on the real axis, and let h be a real parameter. Expand $F(z)$ in a Taylor series off the real axis.

$$F(x_0+ih)=F(x_0)+i.hF'(x_0)-h^2F''(x_0)/2! - ih^3F^{(3)}/3!+...$$

Take the imaginary part of both sides and divide by h

$$F'(x_0)=Im(F(x_0+ih))/h+O(h^2)$$

Armed with the 41Z arsenal of functions it's very likely that your real function can be programmed as an equation in the complex variable too. Then all it takes is to calculate the value of said complex function in a complex point close to the real argument x_0, offset by a very small amount in the imaginary axis ih. The program expects the program name in ALPHA and the values of h and x0 in the Y,X stack registers, and it returns the real derivative value in X. it uses data registers R00 to R02.

```
01  LBL "ZNWT"
02  ASTO 02
03  ZSTO (00)
04  LBL 00
05  FS? 10
06  VIEW 00
07  ZRCL (00)
08  XEQ IND 02
09  X<>Y
10  /
11  RCL 01
12  *
13  ST- 00
14  RND
15  X#0?
16  GTO 00
17  RCL 00
18  END
```

What's remarkable is that with just one execution of the complex function we calculate both the real function's value (the real part) and its derivative (the imaginary part with correction) at the same time. Note also the clever use of complex data register C00 to store z0 = x0 +ih, and then how it keeps calculating the complex function value until two successive iterations are equal for the current FIX selected in the calculator.

Something's remarkable when the root-finding routine is almost shorter than the equation use to program the function!

Time for some examples. The first one just a simple polynomial to try our hand with the new method, taken from the MoHPC forum: https://www.hpmuseum.org/forum/thread-6667.html

Calculate the three roots of the third degree polynomial: $x^3 - x^2 - x + 0{,}5 = 0$

We program the equation as shown below:

```
01LBL "Z3"          06  Z-
02  Z^3             07  .5
03  LASTZ           08  +
04  Z^2             09  END
05  Z+
```

And type:

```
ALPHA, "Z1", ALPHA
,01, ENTER^, 0, XEQ "ZNWT"      =>   0.40301587
.01, ENTER^, 2, XEQ "ZNWT"      =>  1.45174468
.01, ENTER^, -2, XEQ "ZNWT"     =>  -0.85476055
```

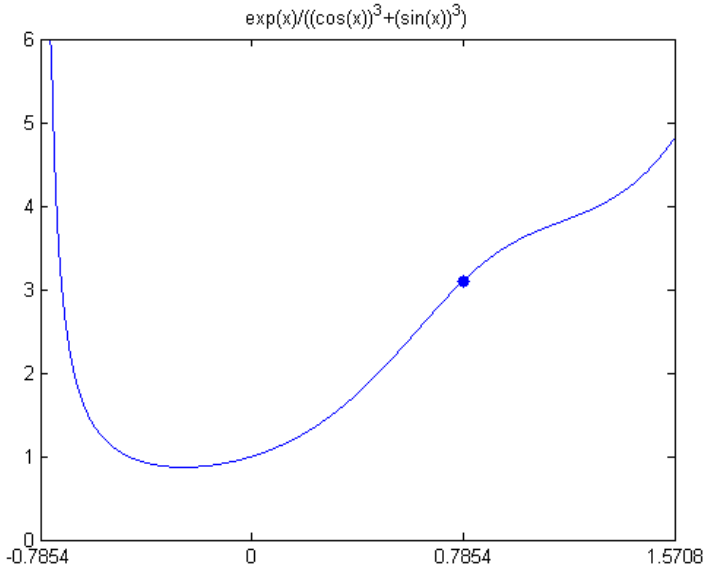
And then a more elaborate example adapted from the seminal reference:
https://blogs.mathworks.com/cleve/2013/10/14/complex-step-differentiation/

The blog uses the function F(x) given below, which does not have any real roots:

$$F(x) = \frac{e^x}{(\cos x)^3 + (\sin x)^3}$$

For our purposes let's calculate the roots of, say g(x) = F(x) – π

1. LBL "Z2"
2. ZEXP
3. LASTZ
4. ZSIN
5. LASTZ
6. ZCOS
7. 3
8. Z^X
9. Z<>W
10. 3
11. Z^X
12. Z+
13. Z/
14. PI
15. -
16. END



And type:

```
ALPHA, "Z2", ALPHA
,01, ENTER^, 1, XEQ "ZNWT"      =>  0.79830245
```

12.4 Successive Approximations Method. { ZSAM }

The next application uses the successive approximation method to obtain the roots of a system of n non-linear equations, provided that the equations can be written as an explicit form of each variable. This is usually doable, but not always possible, and even when it is the method is slow – but should be a reliable approach provided that sensible initial guesses are provided.

The program includes data entry and results output routines, i.e. a classic "driver" structure for additional convenience. The core routine is adapted from JM Baillard's FNZ posted at : *http://hp41programs.yolasite.com/approx.php*

Some modifications to the original core routine FNZ were required to adjust the register mapping to the 41Z convention. Using native 41Z functions also resulted in a code reduction, which is always a good thing.

Example. Let's solve the system of the two equations below:

z1 = (z1^2 - z2)^1/3 ,
z2 = (z2^2 - z1)^1/4

Programmed as follows:

```
01  LBL "Z1="      11  LBL "Z2="
02  ZRCL 01        12  ZRCL 02
03  Z^2            13  Z^2
04  ZRCL 02        14  ZRCL 01
05  Z-             15  Z-
06  3              16  4
07  Z^1/X          17  Z^1/X
08  RTN            18  END
```

Using (1+i) as initial guesses for both z1 and z2, the results are obtained in a few seconds on the 41-CL, or with an emulator in Turbo mode.

z1 = R02 + i R03 = 1.038322757 + 0.715596476 i
z2 = R04 + i R05 = 1.041713085 - 0.462002405 i

The program listing is provided below.

```
LBL "ZSAM"
SIZE?
"N=?"
PROMPT
STO 00
3
*
E
+
X>Y?
PSIZE
XEQ 00
RCL 00
E3/E+
ZINPT
*LBL 01
VIEW 02
CLA
RCL 00
ST+ X
STO M
*LBL 02
RCL 00
ST+ X
E
+
RCL M
+
RCL IND X
RDN
XEQ IND T
ZENTER^
ZRCL IND M
Z-
ZMOD
ST+ N(6)
DSE M(5)
GTO 02
X<>N
E-8
RCL 00
*
X<Y?
GTO 01
RCL 00
E3/E+
ZOUPT
RTN
LBL 00
RCL 00
ST+ X
2.1
+
STO 02
RCL 00
E3/E+
STO 01
AON
CF 23
*LBL 05
"F#"
ARCLI 01
" /-? "
ARCL IND 02
STOP
FS?C 23
ASTO IND 02
ISG 02
ISG 01
GTO 05
AOFF
END
```

Comments.

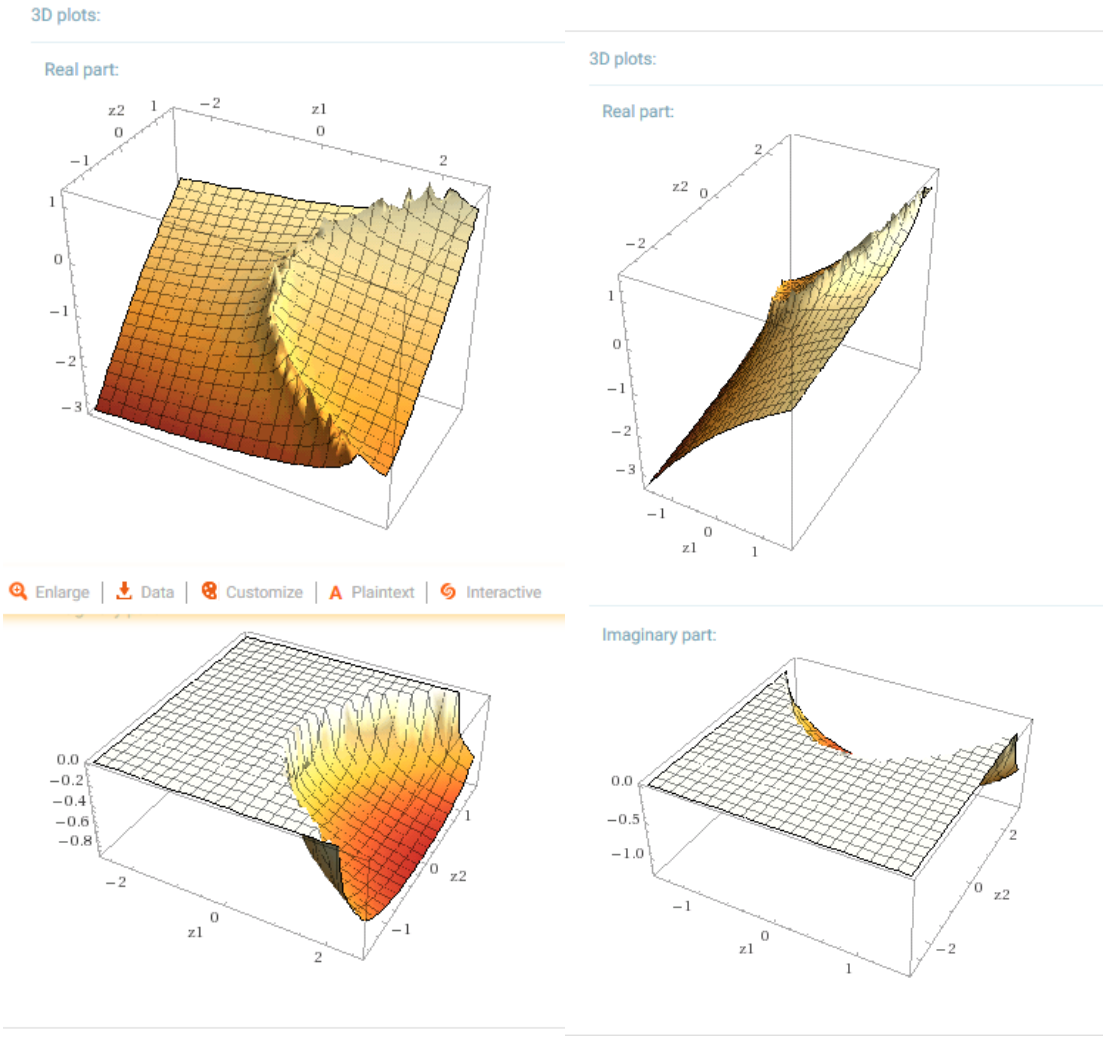
E3/E+ is a shortcut for the sequence { 1E3, /, 1, +} and ARCLI 01 is the short form for { CF 29, FIX 0, ARCL 01, FIX 3 SF 29 } – or other combination using functions like AIP, ARCLI, or AINT.

Note how ZRCL is happy using indirect stack arguments – written as non-merged program steps, which are automatically added by the function itself when entered in the program.

Because registers M and N are used, the execution should not be done using the single-step – as that will overwrite these registers with the intermediate results of the complex functions (which is not done in running mode).

Note that to call the respective equations, we first get the global label name in the X-register using the RCL IND X step; then do RDN and XEQ IND T. Could we have used XEQ IND X directly? It turns out not really, because surely the equation routines will use the complex Z-Stack, and that will complain if the current content of the {X,Y} registers cis Alpha data. There's no real reason for this behavior, sunt so far that's how the omplex buffer reacts – thus the work-around using the T register for the call.

See below the graphics of both functions (real and imaginary parts). The solution of the system would represent where both real parts and both imaginary parts intersect.



12.5 Function Derivatives. { **ZDERV** } (by Greg McClure)

The **ZDERV** function calculates the 1st and 2nd derivatives of a global function defined by the user (and thus visible via Catalog 1). The function needs to be continuous thru the range around the value at which the derivatives of the function are desired. The program uses data registers {R00-R08} as follows:

- ZR00 (R00/R01) is the summing register for the first derivative partials calculated each pass. It should be left alone (read only).
- ZR01 (R02/R03) is the summing register for the second derivative partials calculated each pass. It should be left alone (read only).
- ZR02 (R04/R05) is the current Z for calculation by routine pointed to by alpha for XEQA. It was initialized to the value entered by the user in X, Y (and also in complex Z). It is modified by **ZDERV** each step, so it is the next value for the user routine when called. It is up to the user program to decide when to use the value (it is not required to be saved by the user if not needed at the beginning of the user program this way). It should be considered read only.
- ZR03 (R06/R07) is the complex step size ZS entered by the user in complex stack level W. It should be left alone (read only).
- R08 is initialized to 0 and contains the current step number (0 to 10). It is used by the logic to know when to go from Z+5*StepSize to Z-5*StepSize (right after handling step #5) and when to stop (right after step#9, when it increments to 10). So, for example, if z = 1+0i and ZS = 0.3+0i, the sampling will be: 1, 1.03, 1.06, 1.09, 1.12, 1.15, 0.85, 0.88, 0.91, 0.94, 0.97 (each wit h +0i) for the 10 points. Again it should be left alone (read only).
- So any of registers R00 thru R08 shouldn't be disturbed by the user program. As long as the user program name is 6 or less characters, it can be ASTO'd / ARCL'd by the program if required. The user program can use ANY of the stack registers and any of the complex stack registers, as long as the final result ends up in X,Y. Never mind that it is duplicated in complex Z, as it should be there if followed 41Z protocol for the program.
- Note that if the FOCAL user program contains high-level math complex functions (such as **ZGAMMA**) then its LBL name should also be stored in a separate data register, say R09. This is needed because the more complex functions make internal usage of the ALPHA registers, which therefore would be compromised. Should that occur you're likely to get a "NONEXISTENT" error message when attempting the execute the user program from within **ZDERV**.

Besides the user function name in ALPHA, the program takes two input values, both of them complex: the point where the derivatives are to be evaluated, and the complex step size to use for the derivative evaluation formula (this is a measure of the distance between points sampled). When developing this program, many formulas were available to use… this program uses the 10-point formulas developed by Jean-Marc Baillard.

The formulas used are exact for any complex polynomial of degree < 11 : - f(x+k.h) is denoted fk to simplify these expressions -

$$df/dx = (1/2520.h).[\ 2100.(\ f_1 - f_{-1}\ ) - 600.(\ f_2 - f_{-2}\ ) + 150.(\ f_3 - f_{-3}\ ) - 25.(\ f_4 - f_{-4}\ ) + + 2.(\ f_5 - f_{-5}\ )\ ] + O(h^{10})$$

$$d^2f/dx^2 = (1/25200.h^2).[\ -73766\ f_0 + 42000.(\ f_1 + f_{-1}\ ) - 6000.(\ f_2 + f_{-2}\ ) + 1000.(\ f_3 + f_{-3}\ )$$

$$- 125.(f_4 + f_{-4}) + 8.(f_5 + f_{-5})] + O(h^{10})$$

The implementation of **ZDERV** also makes use of a hidden function, **ZDRTN**. It is NOT designed to be used in the user function created, which only need RTN or END to terminate the FOCAL code that defines them. Why then is XQRTN needed? The operating system normally does not allow returning to MCODE from FOCAL programs. So to overcome this restriction **ZDERV** jumps to a mini-FOCAL program that contains **ZDRTN** to execute the user function and return back to the **ZDERV** MCODE after doing a real RTN.

All this is transparent to the user, who needs only to provide the function name in ALPHA and the input values in the W- and Z- complex stack levels as described above. The execution ends with the fisrt derivative value in both complex stack level Z and ZR00, and the second derivative value in both complex stack W and ZR01.

Example 1. Derivatives of SIN

Let's say we want to find the derivative of f(z) = sin(z) at z=1. First we need to create a Global label program to define the function (as it cannot use mainframe function names). Note that there's no need to preserve the routine name in R09 as **ZSIN** does not use the ALPHA registers internally.

```
01  LBL "SINZ"
02  ZSIN
03  END
```

Let's try a step value of .03 (so the points sampled will be (.85, .88, .91, …, 1.12, 1.15).

```
Type: 0, ENTER^, .03, ZENTER^              =>  0.030+j0
0, ENTER^, 1,  XEQ "ZDERV_ "SINZ"  ALPHA   =>  "RUNNING..."
```

On return, both ZR00 and Z contain 0.540302302 (the actual 1st. derivative is 0.54032306) and ZR01 and W contains -0.841470900 (the actual 2nd derivative is -0.841470985).

Testing the sine function for other values and step sizes is easy if you use the explicit derivatives, f'(sin(z)) = cos(z), and f''(cos(z)) = -sin(z), that is to say, you can test the values obtained by this program for this example by taking the cos(z) and –sin(z) for the actual 1st and 2nd derivative values.

Example 2.- Calculate f '(1+i) & f "(1+i) for: f(z) = exp(-z^2)

We program the function using any global LBL , 6 characters or less

```
01  LBL "EX2"
02  Z^2
03  ZNEG
04  ZEXP
05  END
```

If we choose h = 0.03(1+i) as step-size we type:

```
0.03, ENTER^,  ZENTER^                =>  0.003(1+j)
1, ENTER^, 1,  XEQ "ZDERV_ "EX2"  ALPHA   =>  "RUNNING..."
```

f '(1+i) = -0.986301184 + j2.650888353; and **Z<>W**
f "(1+i) = 8.106657849 - j1.510648148 ;

Choosing the best h-value is not easy but h ~ 0.03 (in both axis) "often" produces good results. Be aware that unfortunately the better step-size for the first derivative may not be a good one for the second, and vice-versa.

Example 3. Cubic Polynomial Derivatives.

With the following coefficients stored in ZR06-ZR09 (i.e. R12-R19), calculate the derivatives in z=1+j of the cubic polynomial. Use Zstep=0.1+0.1j. The results should be -3+j17 and 4+j16 for the 1st and 2nd derivatives respectivey, as calculated by **ZDP1** and **ZPD2**.

ZR06 = 1+j - third degree coeff.
ZR07 = 2+2j - second degree coeff.
ZR08 = 3+3j - first degree coeff.
ZR09 = 4+4j - independent term.

We start by programming the function under the user label "ZP69", taking advantage of the ZPL function to do the polynomial evaluation. Note that this uses the ALPHA register M internally for scratch, thus we need to preserve the global program name in another data register and restore it after the evaluation is done. We'll use R09 for this purpose. Note as well that the usage of storate registers must be compatible with **ZDERV** requirements, which uses ZR00 to ZR03

```
01  LBL "ZP69"
02  ASTO 09      LBL name preserved
03  ZRCL 02      initial argument
04  NOP          to separate numeric steps
05  6.009        control word
06  ZPL          evaluates polyn
07  CLA          clear scratch
08  ARCL 09      routine LBL restored
09  END
```

Then we enter the function parameters as usual:

```
0.1, ENTER, ZENTER^,                      =>  0.100(1+J)
1, ENTER^, 1,  XEQ "ZDERV_ "ZP69"  ALPHA  =>  "RUNNING..."
```

Which shortly returns with the exact solutions in the complex stack: -3+J17; **Z<>W** 4+J16

Example 4. Derivatives of Gamma function.

Let's now do a high-level math example using **ZGAMMA**, which also messes with the ALPHA registers thus we need to save the global label in R09 in this case as well. Let's calculate the derivatives in the point z0 = 1+i, also using a step size zh = 0.1 (1+j)

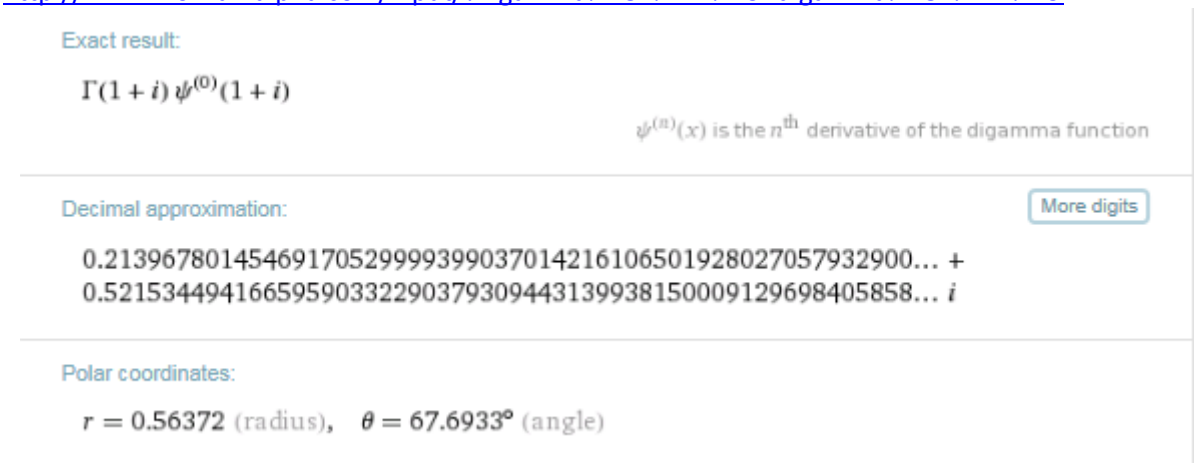
```
01  LBL "GAM"
02  ASTO 09
03  ZGAMMA
04  CLA
05  ARCL 09
06  END
```

```
0.1, ENTER^, ZENTER^,                     =>   0.100(1+J)
1, ENTER^,  XEQ "ZDERV_ "GAM"  ALPHA      =>   0.2140+J0.5215
Z<>W                                      =>  -0.4338-J0.1875
```

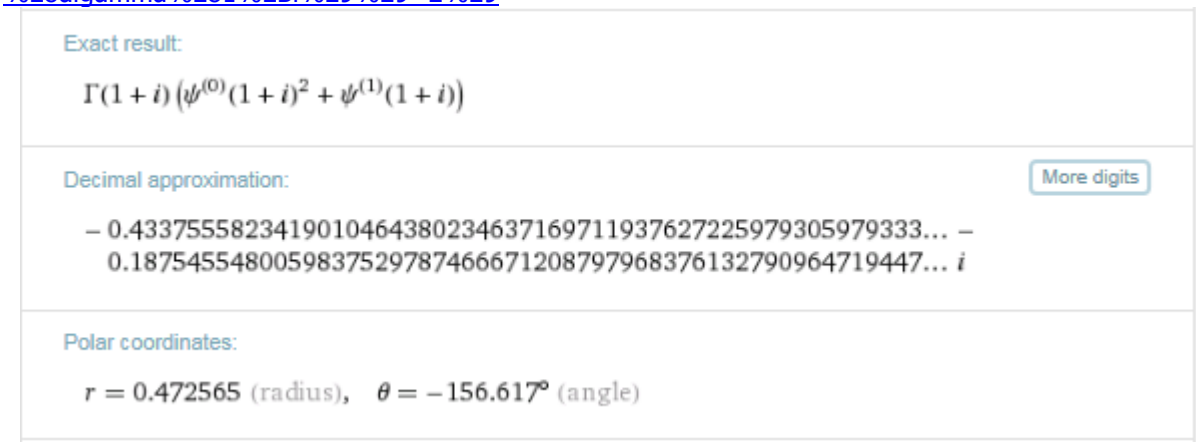
The first derivative should equal ZPSI * ZGAMMA, and it does!,

1, ENTER^, **ZGAMMA**, **ZSTO** 02, **LASTZ**, **ZPSI**, **ZRC*** 02 => 0.2140+J0.5215

Which can also be verified using WolframAlpha, see:
http://www.wolframalpha.com/input/?i=gamma%281%2Bi%29*digamma%281%2Bi%29



And similarly for the second derivative using the tri-gamma function:
http://www.wolframalpha.com/input/?i=gamma%281%2Bi%29*%28trigamma%281%2Bi%29+%2B%28digamma%281%2Bi%29%29^2%29



Example 5: Halley's Method.

This example clearly illustrates the usefulness of **ZDERV** – applied to the Halley's method to obtain the roots of a function. Contrary to the secant algorithm, the Halley's method only needs one initial estimation, and the convergence is meant to be faster - reducing so the execution time.

The following FOCAL program lists the code (set FIX as needed for precision):

```
01 LBL "ZHALL
02 "FNAME?"
03 AON
04 PROMPT
05 ASTO 09
06 AOFF
07 "Z0="?"
08 PROMPT
09 ZSTO 02
10 .1
11 ENTER^
12 ZSTO 03
13 LBL 00
14 CLA
15 ARCL 09
16 ZRCL 03
17 ZRCL 02
18 ZDERV
19 ZRCL 02
20 XEQ IND 09
21 ZRPL^
22 ZRC* (00)
23 ZDBL
24 ZRCL (00)
25 Z^2
26 ZDBL
27 ZRCL 01
28 ZRUP
29 Z*
30 Z-
31 Z/
32 ZNEG
33 ZRC+ 02
34 ZENTER^
35 Z<> 02
36 Z=WR?
37 GTO 01
38 GTO 00
39 LBL 01
40 ZAVIEW
41 END
```

12.6 Continued Fractions. { **ZCF2V** } (by Greg McClure)

Continued Fractions are expressions of the form:

$$B(0)+\frac{A(1)}{B(1)+}\frac{A(2)}{B(2)+}\cdots\frac{A(N)}{B(N)+}\cdots$$

The use of + in the denominator indicates that the remainder of the terms actually are part of that denominator. So the above expression means B(0) + A(1) / [B(1) + A(2) / [B(2) + A(3) / […]]].

This can be mathematically abbreviated as B(0) + [A(1), A(2), A(3), … ; B(1), B(2), B(3), …] which will be used here. The number of expressions may or may not be infinite.

Many values are easily expressed as continued fractions. Some examples are:

Tanh(x) = [X, X^2, X^2, X^2, … ; 1, 3, 5, 7, …]
Pi = [4, 1^2, 3^2, 5^2, 7^2, … ; 1, 2, 2, 2, 2, …] (one of MANY representations of Pi)
1 / (e-1) = [1, 2, 3, 4, … ; 1, 2, 3, 4, …] (again one of MANY representations of e)

The simpler form of continued fractions often used are expressions with A(n)=1, therefore of the form: B(0)+1/(B(1)+) 1/(B(2)+)… 1/(B(n)+)… mathematically abbreviated as: [B(0); B(1), B(2), B(3), …]. For example: e = [2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, …]

The Z**CF2V** function is designed to calculate a complex continued fraction value. It requires a user created subroutine that calculates A(n) and B(n) for n >= 1. The function assumes z is available in ZR01 and n available in R12 for this program, and should leave A(n) in complex stack level "Z" and B(n) in complex stack level "W" on completion of the user subroutine. The subroutine must be callable by a global label (of up to 7 characters). The program uses R00 thru R12.

To execute **ZCF2V**, put the value of B(0) in complex stack level "W", and the value of evaluation point z in complex stack level "Z". Execute **ZCF2V** to evaluate the continued fraction - which will prompt for the name of the routine that calculates both A(N) and B(N) and will write it into the alpha register to evaluate the continued fraction. In a program execution (no prompting) you need to enter the user program name in ALPHA prior to the ZCF2V step.

Here is an example of use of **ZCF2V**. Let's say we want to evaluate the Tanh function mentioned above. We would create the following program in memory (assume we use the label TT):

```
01 LBL "ZTH"
02 RCL 12      ; get n from R12
03 1           ; Is it 1?
04 X#Y?
05 GTO 01      ; No, skip to LBL 01
06 CLX
07 X<>Y        ; make it complex
09 ZRCL 01     ; B(1) = 1+j0 in "W",
10 RTN         ; A(1) = z in "Z"
11 LBL 01
09 -           ; (n–1) in X
10 RCL 12      ; get n again
11 +           ; (2n–1) in X
12 0
13 X<>Y        ; make it complex
14 ZRCL 01
15 Z^2         ; B(n) = (2n – 1)+j0 in "W",
16 END         ; A(n) = z^2 in "Z"
```

To evaluate Tanh(1) with B(0)=0 enter the following:

0 , ENTER^, **ZENTER^**, ENTER^, 1, **ZCF2V_** "ZTH ALPHA. -> "RUNNING…"

The answer of 0.761594156 (assuming FIX 9) is displayed in a few seconds. The value returned should be accurate to at least 9 significant digits.

Try now to evaluate Tanh(1+i), which answer is 1.083923328 + j 0.271752586:

[**Z**] 0, **ZENTER^**, 1, ENTER^, XEQ "**ZCF2V**_"ZTH ALPHA =>
RUNNING...", => **1.0839+J0.2718**

Example 2. If f(z) is defined by: $b_0 = 0.2 + 0.3\,i$; $a_n = 2.z + n$, $b_n = z^2 + n^2$ (n > 0) ; evaluate f(1+2.i). We program the function components as follows:

01	LBL "FZ"	08	2
02	**ZRCL** 01	09	ST* Z
03	**Z^2**	10	*
04	RCL 12	11	RCL 12
05	X^2	12	+
06	+	13	END
07	**ZRCL** 01		

Calculated as:

0.3, ENTER^, 0.2, **ZENTER^**, 2, ENTER^, 1, XEQ "**ZCF2V**_"FZ" => **1.0846-J0.7498**

Register usage for ZCF2V.

First, name of function must be in Alpha (up to 7 chars allowed). The prompting makes that easy for keyboard usage!

Like it was the case for ZDERV, the *user program name must be in the ALPHA register each time the function is to be evaluated.* If the user program contains functions that alter the contents of ALPHA then you'll need to restore said user program name as part of the user program itself. Typically you save it on entry (say in an available data register like R15) and restore it upon completion of the continued fraction.

- Register ZR00 is the continuing estimate of F(N) and hopefully gets closer and closer to the real solution (or we wind up with an infinite loop). It should be considered read only.
- Register ZR01 is the saved value of z. The user program that calculates the next A(N) and B(N) terms can use this value. It should be considered read only.
- Register ZR02 is C(N) from the modified Lentz formula. It should be considered read only.
- Register ZR03 is D(N) from the modified Lentz formula. It should be considered read only.
- Register ZR04 is A(N) saved from the user program. It can be reused by the user program but will be replaced on reentry to the ZCF2V calculation loop.
- Register ZR05 is B(N) saved from the user program. It can be reused by the user program but will be replaced on reentry to the ZCF2V calculation loop.
- Register R12 is the current loop count (N). The user program that calculates the next A(N) and B(N) terms can use this value. It should be considered read only.

If an infinite loop is occurring, pressing R/S should stop the program on the next entry to the user program. If no infinite loop is occurring, the answer should eventually show up in X,Y (Complex Z).

Example 2. Bessel Functions Jn(x) and Yn(x). { **JYNX** }

This example showcases the use of continued fractions to calculate the Bessel functions of first and second kinds, Jn(x) and Yn(x), for *real values of order and argument*. It is a very interesting application that has the benefit to avoid the limitations of the direct methods when the order and/or argument are large. Therefore, unlike unlike the counterpart functions in the SandMath, the following program produces accurate results for large arguments.

You should note that this approach involves solving two continued fractions, one in the complex domain and another in the real domain – therefore both the 41Z and SandMath modules need to be plugged in the calculator.

Formulae:

Let ZCF be the complex continued fraction defined by:

ZCF = [(0.5^2 – n^2)/(2x + 2i + (1.5^2 – n^2)/(2x + 4i + ...))]

And CF be the real continued fraction defined by:

CF= -1/(((2n + 2)/x) - 1/(((2n + 4)/x) - .....))
D = denominator of CF

Let: p + i.q = -1/(2x) + i .[1 + (1/x) [ZCF] }
And: s = (p - CF - n/x)

then we have the following expressions:

Jn(x) = sign(D). sqrt[(2q/(π.x) / ((q^2 + s^2)]
Yn(x) = [s / q] . Jn(x)

Numeric application:

10 ENTER^ XEQ "JYNX" => J10(10) = 0.207486107
X<>Y Y10(10) = -0.359814151 (in 2mn27s)

3.14, ENTER^, 100, XEQ "JYNX" => J3.14(100) = 0.079535723
X<>Y Y3.14(100) = 0.006582327 (in 4mn14s)

The method doesn't work if n is a negative integer. However in that case, if n < 0 we can use the relations

Jn = J-n cos n.Pi + Y-n sin n.Pi , and
Yn = -J-n sin n.Pi + Y-n cos n.Pi

If x < 0 the results are generally complex and won't be properly calculated by this program.

Data Register Usage.

"**JYNX**" needs data registers R00 to R13. {R00 - R12} are used by ZCF2V, plus one additional register (R13) is needed to save the value of the order "N".

Note that both **ZCF2V** and **CF2**V have slightly different conventions as to where the arguments are stored: in CR01 for ZCF2V, which transtaled to R02 and R03 for the real and imaginary parts. All this is transparent to the user for this example.

The Program Listing is shown below. Note the calculation for the p and q factors takes advantage of the complex result returned by ZCFV, transposing the real and imaginary parts as per the multiplication by "i" in the definition formulae:

$$p = -1/x \cdot [\ 1/2 + Im(ZCF)]$$
$$q = [\ 1 + Re(ZCF/x)\ ]$$

Credits: The original program was written by Jean-Marc Bailalrd, and has been adapted to use the MCODE implementations of the continued fractions routines. Thanks also to Greg McClure for his assistance provided for the adaptation.

```
01 LBL "JYNX"
02 STO 01        x
03 X<>Y
04 STO 13        N
05 "ZCF"
06 CLST
07 ZENTER^
08 0
09 RCL 01
10 ZCF2V
11 RCL 02        x
12 STO 01
13 ST/ Z
14 /
15 E
16 +
17 STO 10        q
18 X<>Y
19 CHS
20 RCL 01        x
21 ST+ X
22 1/X
23 –
24 STO 09        p
25 "CF"
26 0
27 RCL 01        x
28 CF2V
29 CHS
30 RCL 09        p
31 +
32 RCL 13        N
33 RCL 01        x
34 /
35 –
36 STO 11        s
37 RCL 10        q
38 R-P
39 LASTX         q
40 ST+ X         2q
41 PI
42 RCL 01        x
43 *             π.x
44 /
45 SQRT
46 X<>Y
47 /
48 RCL 05
49 SIGN
50 *
51 STO 12
52 RCL 11        s
53 *
54 RCL 10
55 /
56 RCL 12
57 CLD
58 RTN
59 LBL "ZCF"
60 RCL 12        N
61 ST+ X         2N
62 RCL 02        x
63 ST+ X         2x
64 ZENTER^
65 0
66 RCL 12        N
67 0,5
68 –
69 X^2
70 RCL 13        N
71 X^2           N^2
72 –
73 RTN
74 LBL "CF"
75 X<>Y
76 STO 05        Bn
77 X<>Y
78 RCL 02        n
79 RCL 13        N
80 +
81 ST+ X
82 RCL 01        x
83 /
84 -1
85 END
```

Note that this program is not available in the 41Z Module, but it has been included in the "Advantage_Math" ROM, a collection of applications using the advanced modules like the 41Z, SandMath and SandMatrix, sometimes *used together* .

Bessel functions for complex variable are covered in the next sections of the manual.

12.7 Bessel and Hankel functions.

This section represents an interesting "*tour de force*" within the 41Z module – taking the humble 41 system to the realm of true high-level math. Use it or leave it, it's all a matter of choice – but programming techniques and valid algorithms are always interesting, despite its obvious speed shortcomings.

Index	Function	Description	
1	**ZJBS**	Complex Bessel J function	First kind
2	**ZIBS**	Complex Bessel I function	First kind
3	**ZKBS**	Complex Bessel K function	Second kind
4	**ZYBS**	Complex Bessel Y function	Second kind
5	**EIZ/IZ**	Spherical Hankel first kind order zero	SHK1 (0, z)
6	**ZSHK1**	Spherical Hankel first kind	SHK1 (n, z)
7	**ZSHK2**	Spherical Hankel second kind	SHK2 (n, z)
8	**ZANGJ**	Anger Function	J (n, z); n real
9	**ZWEBE**	Weber Function	W (n, z); n real

See the paper "*Bessel functions on the 41 with the SandMath Module*" by the author, for an extensive description of the (real-number) Bessel Functions on the 41 system. In fact, following the "*do it as it's done with real numbers*" standard philosophy of the 41Z module, the complex versions of these programs are very similar to those real-number counterparts described in said paper.

The formulae used are as follows:

$$\mathbf{J(n,z) = \Sigma \{U_k \mid k=1,2,\ldots.\} * (z/2)^n / \Gamma(n+1)}$$

$$U(k) = -U(k-1) * (z/2)^2 / k(k+n)$$

$$U(0) = 1$$

$$\mathbf{Y_n(x) = [\, J_n(x) \cos(n\pi)) - J_{-n}(x) \,] / \sin(n\pi))}$$

$$\mathbf{K_n(x) = (\pi/2) [\, I_{-n}(x) - I_n(x) \,) / \sin(n\pi)) \,]}$$

n # .... -3 ; -2 ; -1 ; 0 ; 1 ; 2 ; 3 ..

Like for the real arguments case, there is one auxiliary functions **ZBS#**, used to perform intermediate calculations needed by the main programs: **ZJBS**, **ZIBS** (first kind), and **ZYBS**, **ZKBS** (second kind). Other auxiliary functions are:

- **ZGEU** Euler's gamma constant as a complex number, and
- **HARMN** to obtain the harmonic number of a given integer: (uses "**-ZSTACK**")

$$H(n) = \Sigma\, [1/k] \mid k=1,2\ldots n \quad (*)$$

The expressions used to calculate the results are different for integer orders (remember the singularities of Gamma), requiring special branches of the main routines. For that reason two other functions have been added to the 41Z as follows:

- **ZINT?** to determine integer condition, and
- **ZCHSX** to simplify calculation of z*(-1)^k

Both the function order and the argument are complex numbers, which are expected to be on complex stack levels **W** (order) and **Z** (argument) prior to the execution of the function. The result is placed on the Z-level complex stack.

Below are the program listings for each particular case.-

a) *Bessel Functions of the first kind.* Uses R00 – R08. Uses Flags 0-1

1	LBL ZJBS			48	Z*	*n*
2	CF 00			49	ZRCL 00	*n*
3	GTO 00			50	RCL M	k
4	LBL ZIBS			51	+	*n+k*
5	SF 00			52	LASTX	*k*
8	LBL 00			53	ST* Z	*k(n+k)*
8	CF 01			54	*	
8	Z<>W			55	Z/	
9	ZINT?	*is n integer?*		56	ZSTO 02	*U(k)*
10	XEQ 05			57	ZRCL 03	*SUM(k-1)*
11	Z<>W			58	Z+	*SUM(k)*
12	ZHALF	*z/2*		59	ZENTER^	
13	*XROM "ZBS"*			60	Z<> 03	*SUM(k-1)*
14	FS? 01	*n integer*		61	Z=W?	
15	RCL 01			62	GTO 01	
16	FS? 01			63	*E*	
17	ZCHSX	*J(-n, z) = (-1)^n J(n, z)*		64	ST+ M	*k=k+1*
18	LBL 04			65	GTO 02	
19	ZAVIEW			66	LBL 01	
20	RTN			67	ZRCL 00	*n*
21	LBL 05			68	INCX	*(n+1)*
22	X<0?	*n<0?*		69	CF 02	
23	SF 01			70	X<0?	
24	ABS			71	SF 02	
25	*RTN*			72	X<0?	
26	LBL "ZBS"			73	ZNEG	-z
27	Z#0?			74	ZGAMMA	
28	GTO 00			75	FC? 02	
29	Z=W?			76	*GTO 00*	
30	E			77	LASTZ	-z
31	GTO 04			78	ZGNGZ	
32	LBL 00			79	Z<>W	
33	-ZSTACK	*running...*		80	Z/	
34	ZSTO 01	(z/2)		81	LBL 00	
35	Z<>W	*n*		82	Z/	
36	ZSTO 00	*n*		83	ZRCL 01	*(z/2)*
37	*E*	*1*		84	ZRCL 00	*n*
38	ZREAL	*1+J0*		85	W^Z	*(z/2)^n*
39	ZSTO 02	*1+J0*		86	Z*	
40	ZSTO 03	*1+J0*		87	END	
41	STO M	*k=1*				
42	LBL 02			CR00 - n		
43	ZRCL 01			CR01 - Z/2		
44	Z^2	*(z/2)^2*		CR02 - Uk		
45	ZRCL 02	*Uk-1*		CR03 - SUM		
46	FC? 00			CR04 - result		
47	ZNEG					

Examples:- Calculate JBS(1+i, -1-i) and IBS(-0.5+i; 1-0,5i)

1, ENTER^, **ZENTER^**, **ZNEG**, **ZJBS** --> -8,889 + j 2,295
1, ENTER^, 0,5, CHS, **ZENTER^**, ENTER^, 1, **ZIBS** --> 3,421 + j 1,178

b) *Bessel functions of the second kind.* Uses R00 – R08. Uses flags 0-2

1		LBL "ZB1"	***SUM{f(n,x)}***	1		LBL "ZB2"	***SUM{g(n,x)}***
2		CLZ		2		CLZ	
3		ZSTO 02	*Jn / In*	3		ZSTO 03	*reset partial SUM*
4		ZSTO 04	*SUM*	4		RCL 00	*ABS(n)*
5		STO 01	*k=0*	5		X=0?	*n=0?*
6		LBL 02		6		RTN	*skip it*
7		XEQ 10	*summing term*	7		DECX	
8		Z=0?	*x=0?*	8		E3	
9		GTO 01	*ignore term*	9		/	*0,00(n-1)*
10		ZRCL 04	*S(k-1)*	10		STO 08	
11		Z+	*S(k)*	11		LBL 05	
12		ZENTER^		12		ZRCL 01	*x/2*
13		Z<> 04		13		RCL 08	*k,00(n-1)*
14		Z=W?	*are they equal?*	14		INT	
15		RTN	*Final result(s)*	15		STO 01	*k*
16		LBL 01		16		ST+ X	*2k*
17		E	*increase index*	17		RCL 00	*n*
18		ST+ 01	*k=k+1*	18		-	*2k-n*
19		GTO 02		19		Z^X	*(x/2)^(2k-n)*
20		LBL 10	***Function to Sum***	20		RCL 00	*n*
21		ZRCL 01	*x/2*	21		RCL 01	*k*
22		RCL 01	*k*	22		-	*n-k*
23		ST+ X	*2k*	23		DECX	*n-k-1*
24		RCL 00	*n*	24		FACT	*(n-k-1)!*
25		+	*2k+n*	25		RCL 01	*k*
26		Z^X	*(x/2)^(2k+n)*	26		FACT	*k!*
27		ZENTER^		27		/	*(n-k-1)! / K!*
28		RCL 01	*k*	28		ST* Z	
29		FACT	*k!*	29		*	*[**]*
30		LASTX	*k*	30		FC? 00	***is it Yn?***
31		RCL 00	*n*	31		GTO 00	
32		+	*k+n*	32		RCL 01	*k*
33		FACT	*(k+n)!*	33		ZCHSX	*(-1)^k * term*
34		*	*k! * (k+n)!*	34		LBL 00	
35		ZREAL		35		ZRCL 03	
36		Z/	*k-th. Term*	36		Z+	
37		FS? 00	***is it Kn?***	37		ZSTO 03	
38		*GTO 00*		38		ISG 08	
39		RCL 01	*k*	39		GTO 05	*(k+1),00(n-1)*
40		ZCHSX	*[term] * (-1)^k*	40		ZRCL 03	
41		LBL 00		41		FC? 00	is it Yn?
42		Z<> 02	***ZST+ 02***	42		RTN	
43		ZRCL 02		43		RCL 00	n
44		Z+	***f(k) + SUM(k-1)***	44		ZCHSX	*SUM*(-1)^n*
45		Z<> 02	Jn / In	45		END	
46		ZENTER^					
47		RCL 01	*k*				
48		HARMN	*H(k)*				
49		LASTX	*k*	*Note: functions DECX and INCX*			
50		RCL 00	*n*	*can be replaced by standard*			
51		+	*k+n*	*FOCAL sequences:*			
52		HARMN	*H(k+n)*				
53		+	*H(k)+H(k+n)*			DECX = 1, -	
54		ZREAL				INCX = 1, +	
55		Z*					
56		END					

1	*LBL "ZYBS"*	*Integer Index*			48	LBL 05	*integer orders*
2	CF 00				49	CF 01	
3	*GTO 00*				50	X<0?	*negative*
4	*LBL "ZKBS"*				51	SF 01	
5	SF 00				52	ABS	
6	*LBL 00*				53	STO 00	n
7	ZHALF				54	*XROM "ZB2"*	
8	ZSTO 01	(z/2)			55	ZNEG	*-[SUM*(-1)^n]*
9	Z<>W	n			56	ZSTO 03	
10	ZINT?				57	*XROM "ZB1"*	*to obtain both!*
11	GTO 05				58	ZRCL 03	
12	Z<>W		ZNEG		59	Z<>W	
13	*XROM "ZBS"*		Z<>W		60	Z-	
14	ZSTO 02	*Jn / In*	*XROM "ZBS"*	*J-n (z)*	61	ZRCL 01	*x/2*
15	FS? 00		FS? 00		62	ZLN	*Ln(x/2)*
16	*GTO 00*		ZNEG	*-J-n(z)*	63	GEU	*g*
17	ZRCL 00		ZSTO 04		64	+	*g+Ln(x/2)*
18	PI		ZRCL 00	*-n*	65	ZRCL 02	*J(n,x) or I(n,x)*
19	ST* Z		ZNEG	*n*	66	Z*	*[ ]*J/I(n,x)*
20	*		ZRCL 01	(z/2)	67	ZDBL	
21	ZCOS		*XROM " ZBS"*		68	Z+	K(n,x)/Y(n,x)
22	Z*		ZSTO 02	*Jn / In*	69	FC? 00	is it Yn?
23	LBL 00		FS? 00		70	GTO 04	FINAL STEPS
24	ZSTO 04		*GTO 00*		71	RCL 00	*n*
25	ZRCL 00	*n*	ZRCL 00		72	INCX	*(n+1)*
26	ZNEG	*-n*	PI		73	ZCHSX	*K(n,x)* (-1)^(n+1)*
27	ZRCL 01	(z/2)	ST* Z		74	ZHALF	
28	*XROM " ZBS"*		*		75	*GTO 03*	*Exit*
29	ZRCL 04		ZCOS		76	LBL 04	*Yn*
30	Z<>W		Z*		77	PI	
31	Z-		LBL 00		78	ST/ Z	
32	ZRCL 00	*-n*	ZRCL 04		79	/	
33	ZNEG	*n*	Z+		80	FC? 01	*negative index?*
34	PI		ZRCL 00	*n*	81	GTO 03	*Exit*
35	ST* Z		PI		82	RCL 00	*n*
36	*		ST* Z		83	ZCHSX	
37	ZSIN		*		84	LBL 03	
38	Z/		ZSIN		85	ZSTO 03	
39	FC? 00		Z/		86	ZAVIEW	
40	GTO 03	*Exit*	FC? 00		87	END	
41	PI		GTO 03	*Exit*			
42	2		PI				
43	/		2				
44	CHS		/				
45	ST* Z		ST* Z				
46	*		*				
47	LBL 03	*Exit*	LBL 03	*Exit*			

The formulae used for integer orders are as follows:

$$\pi\, Y_n(x) = 2[\gamma + \text{Ln } x/2]\, J_n(x) - \Sigma\, (-1)^k\, f_k(n,x) - \Sigma\, g_k(n,x)$$

$$(-1)^{n+1}\ \ 2\, K_n(x) = 2\,[\gamma + \text{Ln } x/2]\, I_n(x)\ - \Sigma\, f_k(n,x) - \ (-1)^n\, \Sigma\, (-1)^k\, g_k(n,x)$$

$$g_k\,(n,x) = (x/2)^{2k-n}\ [(n-k-1)!\ /\ k!]\ \ ;\ k=0,2,\ldots(n-1)$$

$$f_k\,(n,x) = (x/2)^{2k+n}\ [H(k) + H(n+k)]\ /\ [k!\ (n+k)!]\ \ ;\ k=0,1,2,\ldots..$$

Example: - Calculate **KBS** (-0.5+i; 1-0,5i)

1, ENTER^, 0,5, CHS, **ZENTER^**, ENTER^, 1, XEQ "ZKBS" → 0,348 + j 0,104

Example:- Calculate YBS (-1,-1)

0, ENTER^, 1, CHS, **ZENTER^**, XEQ "ZYBS" → - 0,781 + j 0,880

This last example shows how even real arguments can yield complex results.

Example. - Calculate JBS and IBS for (1+2i, -1-3i)

2, ENTER^, 1, **ZENTER^**
3, CHS, ENTER^, 1, CHS, XEQ "ZIBS" → 35,813 - j 191,737

2, ENTER^, 1, **ZENTER^**
3, ENTER^, 1, **ZNEG,** XEQ "ZJBS" → - 257,355 - j 12,633

12.7. Hankel and Spherical Hankel functions. { ZSHK1, ZSHK2, EIZ/IZ }

With the Bessel functions in the pocket it takes a litte more than a trivial exercise to write a few short routines to calculate the Hankel and Spherical Hankel functions – both of the first and second kind. Their defining expressions are as follows:

$$H_\alpha^{(1)}(x) = J_\alpha(x) + iY_\alpha(x) \qquad h_n^{(1)}(x) = j_n(x) + iy_n(x)$$
$$H_\alpha^{(2)}(x) = J_\alpha(x) - iY_\alpha(x) \quad ; \qquad h_n^{(2)}(x) = j_n(x) - iy_n(x).$$

These linear combinations are also known as Bessel functions of the third kind, and it's just an association of the previous two kinds together. Here the spherical analogues of the Hankel functions are based on the Spherical Bessel functions as follows:

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+1/2}(x),$$
$$y_n(x) = \sqrt{\frac{\pi}{2x}} Y_{n+1/2}(x) = (-1)^{n+1} \sqrt{\frac{\pi}{2x}} J_{-n-1/2}(x).$$

Example: Calculate HK1 and HK2 of zero order for z= (1+i)

[Z], [0] , **ZENTER^**, [1], ENTER^, XEQ "ZSHK1" => 0,055-J0,254
[Z], [0] , **ZENTER^**, [1], ENTER^, XEQ "ZSHK2" => 1,878-J0,409

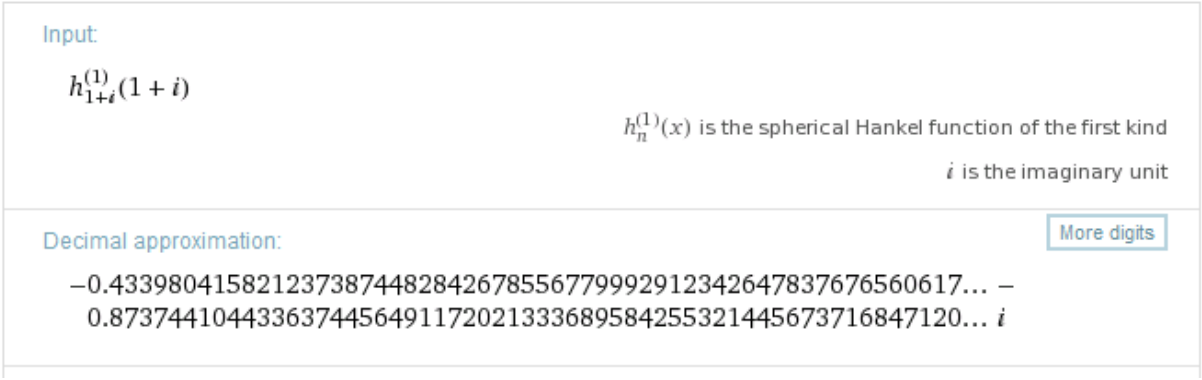
Note that fort the zero-th order **SHK1** we can also use the **EIZ/IZ** function, which uses the direct exponentials formula and therefore comes to the same result in a much shorter time (shown below with 9 decimal digits):

Re = 0,055396883; Im = -0,254162993

These functions are also valid for the non-integer order cases, for example: n= (1+ i) and z= (1+i):

1, ENTER^, **ZENTER^**, XEQ "ZSHK1" => -0,434-J0,874

Which has a 9-digit accuracy when compared to the Wolfram Alpha result – astonishing if you consider the long and winding process needed to get to their result – all done behind the scenes.

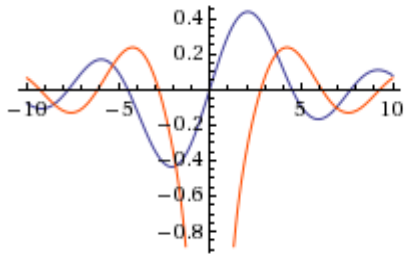


The FOCAL programs below list the simple code snippets to program the regular (**ZHK1** and **ZHK2**) and spherical pairs **ZSHK1** and **ZSHK2**. Note that J is obtained during the Y calculation, thus there's no need to repeat the execution for it – we retrieve its value from complex register ZR02. Note how the complex stack performs a vital role in these programs – storing the intermediate results unaffected by the complex calculations that take place.

01	**LBL "ZSHK1"**	
02	CF 03	
03	GTO 03	
04	**LBL "ZSHK2"**	
05	SF 03	
06	LBL 03	
07	**Z<>W**	
08	*,5*	
09	+	
10	**Z<>W**	
11	**ZYBS**	
12	FS? 03	
13	**ZNEG**	
14	**Z*I**	
15	**ZRCL 02**	*JBS*
16	**Z+**	
17	**ZRCL 01**	z/2
18	*4*	
19	ST* Z	2z
20	*	
21	**ZINV**	
22	**ZPI***	
23	**ZSQRT**	
24	**Z***	
25	**ZAVIEW**	
26	END	

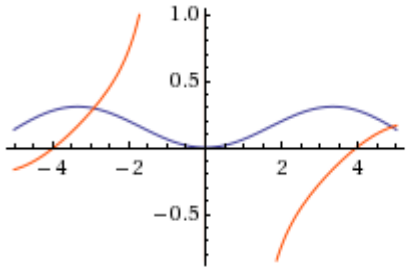
01	**LBL "ZHK1"**	
02	SF 03	
03	GTO 03	
04	**LBL "ZHK2"**	
05	CF 03	
06	LBL 03	
07	**ZYBS**	
08	FS? 03	
09	**ZNEG**	
10	**Z*I**	
11	**ZRCL 02**	*JBS*
12	**Z+**	
13	**ZAVIEW**	
14	END	

The plots below show the Spherical Hankel-1 function for orders 1 and 2, for a short range of the real argument x. Obviously the results are complex as well, thus the real and imaginary parts are plotted separately.



(x from −10 to 10)

real part
imaginary part



(x from −4 to 4)

real part
imaginary part

$$h_0^{(1)}(z) = -i\,e^{jz}\,\frac{1}{z}$$

$$h_1^{(1)}(z) = -e^{jz}\,\frac{z+i}{z^2}$$

$$h_2^{(1)}(z) = i\,e^{jz}\,\frac{z^2+3\,i\,z-3}{z^3}$$

$$h_3^{(1)}(z) = e^{jz}\,\frac{z^3+6\,i\,z^2-15\,z-15\,i}{z^4}.$$

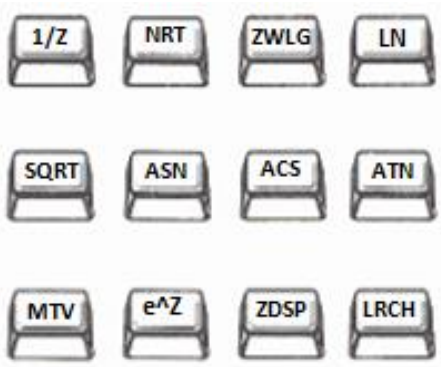
Complex Keyboard shortcuts.- the Bessel and Hankel functions can be accessed pressing SHIFT when the NEXT indicator is shown, as per the following sequence:

[Z], **[Z]**, [SHIFT], [SHIFT] -> then [**I**], [**J**], for **ZJBS** and **ZJBS** or [**K**], [**L**] for **ZKBS** and **ZYB**S.

The same group can be used to access **ZWL** & **ZAWL** (Complex Lambert and its inverse) and **EIZ/IZ**, the Spherical Hankel function of first kind and order zero $h^{(1)}$ (0,z)

, then SHIFT:

The key maps below summarizes all the special assignments in the [**BSSL**] (left) and [**NEXT**] (right) groups. Notice that the mnemonics h(1)n and h(2)n correspond to the **ZSH1** and **ZHS2** functions. Note as well the inclusion of the "alternative" versions **SQRTZ**, **e^Z** and **1/Z** in the [NEXT] group – so you can quickly compare them with the main functions for accuracy and speed.



12.8. Weber and Anger Functions. { ZANGJ, ZWEBE }

In mathematics, the Anger function, introduced by C. T. Anger (1855), is a function defined as

$$\mathbf{J}_\nu(z) = \frac{1}{\pi}\int_0^\pi \cos(\nu\theta - z\sin\theta)\,d\theta$$

The Weber function introduced by H. F. Weber (1879), is a closely related function defined by:

$$\mathbf{E}_\nu(z) = \frac{1}{\pi}\int_0^\pi \sin(\nu\theta - z\sin\theta)\,d\theta$$

The Anger and Weber functions are related by:

$$\sin(\pi\nu)\mathbf{J}_\nu(z) = \cos(\pi\nu)\mathbf{E}_\nu(z) - \mathbf{E}_{-\nu}(z)$$
$$-\sin(\pi\nu)\mathbf{E}_\nu(z) = \cos(\pi\nu)\mathbf{J}_\nu(z) - \mathbf{J}_{-\nu}(z)$$

so in particular if v is not an integer they can be expressed as linear combinations of each other. If v is an integer then Anger functions **J**v are the same as Bessel functions **J**v, and Weber functions can be expressed as finite linear combinations of Struve functions (Hn and Ln).

The expressions used in the 41Z module are based on the Hypergeometric function, therefore use the ascending series method - as follows:

$\mathbf{J_n}(z)$ = + (z/2) sin(90°n) $_1\mathbf{F}_2$(1 ; (3-n)/2 , (3+n)/2 ; $-z^2/4$) / Γ((3-n)/2) / Γ((3+n)/2)
+ cos(90°n) $_1\mathbf{F}_2$(1 ; (2-n)/2 , (2+n)/2 ; $-z^2/4$) / Γ((2-n)/2) / Γ((2+n)/2)

and:

$\mathbf{E_n}(z)$ = - (z/2) cos(90°n) $_1\mathbf{F}_2$(1 ; (3-n)/2 , (3+n)/2 ; $-z^2/4$) / Γ((3-n)/2) / Γ((3+n)/2)
+ sin(90°n) $_1\mathbf{F}_2$(1 ; (2-n)/2 , (2+n)/2 ; $-z^2/4$) / Γ((2-n)/2) / Γ((2+n)/2)

Note that even if the argument z can be a complex number, this implementation requires the order v to be a real value so the dual-complex case is not supported. The input parameters are expected in the real registers {Z, Y, X}, with the order in the X- register as per the standard 41Z conventions.

Examples. Calculate the weber and Anger functions for v =π, and z=1+i

1, ENTER, 1, PI, **ZF$** "ZANGJ"	-> "RUNNING..."	=> -0.064+J0.041
1, ENTER^, 1, PI, **ZF$** "ZWEBE"	-> "RUNNING..."	=> 0.211+J0.077

Which can be checked in WolframAlpha using the syntax shown in the link:

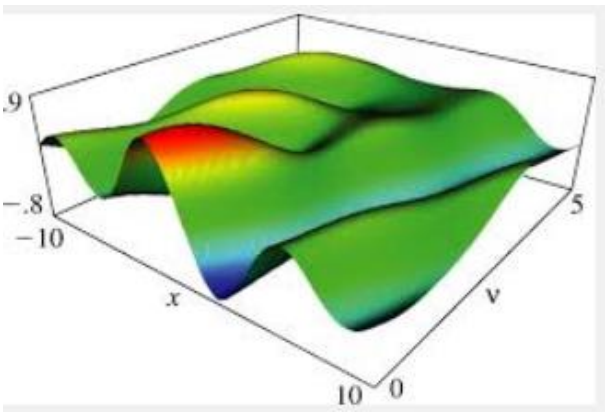
http://www.wolframalpha.com/input/?i=AngerJ%28pi,+1%2Bi%29
http://www.wolframalpha.com/input/?i=WeberE%28pi,+1%2Bi%29

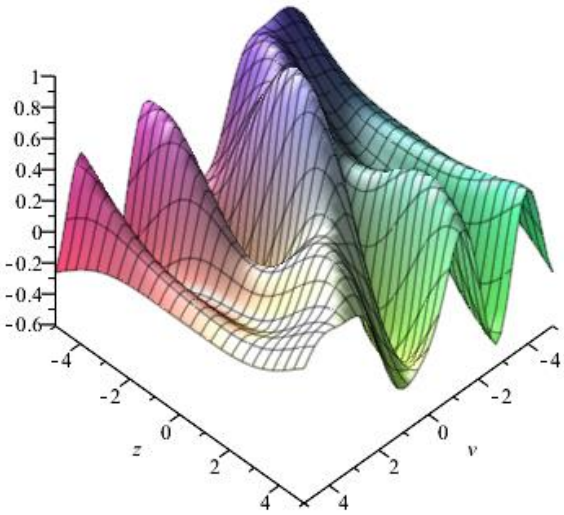
Program Listing for Weber and Anger routines.

```
01  LBL "ZWEBE"
02  SF 00
03  GTO 00
04  LBL "ZANGJ"
05  CF 00
06  LBL 00
07  RAD
08  STO 00
09  RDN
10  2
11  ST/ Z
12  /
13  ZSTO 02
14  E
15  STO 01
16  RCL 00
17  2
18  /
19  -
20  STO 02
21  LASTX
22  E
23  +
24  STO 03
25  XEQ 00
26  RCL 00
27  PI
28  *
29  2
30  /
31  FS? 00
32  SIN
33  FC? 00
34  COS
35  ST* Z
36  *
37  ZENTER^
38  RCL 00
39  3
40  +
41  2
42  /
43  STO 03
44  3
45  RCL 00
46  -
47  2
48  /
49  STO 02
50  XEQ 00
51  ZRCL 02
52  Z*
53  RCL 00
54  PI
55  *
56  2
57  /
58  FS? 00
59  COS
60  FC? 00
61  SIN
62  ST* Z
63  *
64  FS? 00
65  ZNEG
66  Z<>W
67  ZRDN
68  Z+
69  ZAVIEW
70  RTN

71  LBL 00
72  ZRCL 02
73  Z^2
74  ZNEG
75  RCL 01
76  2
77  RDN
78  RDN
79  ZHGF
80  ZENTER^
81  0
82  RCL 02
83  ZGAMMA
84  Z/
85  ZENTER^
86  0
87  RCL 03
88  ZGAMMA
89  Z/
90  END
```

Registers used: R00-R05
Flags used: F0





12.9 Dilogarithm and Polylogarithm. { ZLIN, ZLI2 }

The Polylogarithm (also known as Jonquière's function) is a special function Lis(z) that is defined by the infinite sum, or power series

$$\mathrm{Li}_s(z) = \sum_{k=1}^{\infty} \frac{z^k}{k^s} = z + \frac{z^2}{2^s} + \frac{z^3}{3^s} + \cdots .$$

Only for special values of the order s does the Polylogarithm reduce to an elementary function such as the logarithm function. The above definition is valid for all complex orders s and for all complex arguments z with |z| < 1; it can be extended to |z| ≥ 1 by the process of analytic continuation. See the reference: http://people.reed.edu/~crandall/papers/Polylog.pdf

The implementation of the Polylogarythm is a very rudimentary one, more as an example of direct porting of the real variable routine than anything else. It's based on Jean-Marc's version, that can be found at: http://hp41programs.yolasite.com/dilogarithm.php

Both parameters can be complex numbers, although the series representation used forces the condition that z must be inside the unit circle, that is |z|<1. The program will stop with an error message if |z|>1. Note also that this method is not valid either for points on the unit circle, |z|=1. You can use function **ZLI2** for the dilogarithm, which also works in this case.

In terms of its usage, s is expected to be in level-2 of the complex stack (W), and z in level-1 (Z). Let's see a couple of examples.

Example 1. Calculate Li(2; 0.3+0.4i)

0, ENTER^, 2, **ZENTER^** → 2+J0
(the Z-keypad version: [Z], **2** does the same easier)

.4, ENTER^, .3, XEQ "ZLIN" → 0,266+J0,461

or with FIX 9 settings:
Re = 0.266596867
Im = 0.461362892

Example 2. Calculate Li(1+i, 0.3+0.4i)

1, ENTER^, **ZENTER^** → 1(1+J)
.4, ENTER^, .3, XEQ "ZLIN" → 0,326+J0,565

or with FIX 9 settings:
Re = 0,326456748
Im = 0,565254656

As you can see the program listing doesn't get any easier – so despite its limitations (long execution time, no analytic continuation) it's worthwhile including in the module.

Note that **ZLIN** and ZLI2 are FOCAL programs, and therefore the argument z won't be saved in the LastZ complex register.

01	LBL "ZLIN"
02	"|Z|>1"
03	ZOUT?
04	PROMPT
05	ZSTO 01
06	Z<>W
07	ZSTO 02
08	CLX
09	STO 06
10	E
11	ZSTO 00
12	CLZ
13	LBL 01
14	ZENTER^
15	ZRCL 00
16	ZRCL 01
17	Z*
18	ZSTO 00
19	ISG 06
20	NOP
21	ZRCL 02
22	RCL 06
23	X^Z
24	Z/
25	Z+
26	Z#W?
27	*GTO 01*
28	ZAVIEW
29	END

12.10. Lerch Transcendent Function. { ZLRCH }

The Lerch Transcendent function can be seen as an extension of the Polylogarithm, and therefore it's easy to modify the previous program to the more general case – adding a third argument "α" as follows:

$$\Phi(z,s,\alpha)=\sum_{n=0}^{\infty}\frac{z^n}{(n+\alpha)^s}.$$

note that contrary to the Polylogarithm case, the summation starts at n=0; not at n=1. This would represent an issue if the power function returned a DATA ERROR condition for zero exponent (the zero-th. term being z^0 / 0^s. However the 41Z implementation returns zero for this case, and therefore we can use the same program to calculate both the Polylogarithm and Lerch function – taking α=0 for the additional argument in Lis:

$$\mathrm{Li}(s,z) \sim= \mathrm{Lerch}\,(z,s,0)$$

To be sure the above expression is just a programming trick, but it's not mathematically correct. The proper relationship between both functions is given by:

$$\mathrm{Li}_s(z)=z\Phi(z,s,1).$$

Example 1. Calculate

Φ(0.3+0.4 i ; 3+4 i ; 1+2 i)

4, ENTER^, 3, **ZENTER^** → 3+J4
2, ENTER^, 1, **ZENTER^** → 1+J2
.4, ENTER^, .3, XEQ "ZLRCH" → 7,658-J1,515,

or with FIX 9 settings:

Re = 7,658159105
Im = -1,515114367

Notice the input order convention for the arguments, with *z always entered last,* in the Z-level of the complex stack.

Other useful relationships also involving the Lerch Transcendent functions are shown below:

Riemann Zeta: (*)

$$\zeta(s)=\Phi(1,s,1).,$$

Legendre Chi:

$$\chi_n(z)=2^{-n}z\Phi(z^2,n,1/2).$$

(*) The convergence is very slow, thus using the dedicated **ZZETA** program is a much more convenient approach.

01	LBL "ZLRCH"	
02	*"|Z|>1"*	
03	ZOUT?	
04	PROMPT	
05	ZSTO 01	x
06	CLZ	
07	SIGN	
08	ZSTO 00	*x^0 = 1*
09	ZRDN	
10	ZSTO 02	*a*
11	Z<>W	
12	ZSTO 03	*s*
13	ZNEG	*-s*
14	W^Z	*1/a^s*
15	LBL 01	*Σ(k-1)*
16	ZRCL 01	*x*
17	ZRCL 00	*x^(k-1)*
18	Z*	*x^k*
19	ZSTO 00	
20	ZRCL 02	*a+k-1*
21	*E*	
22	ST+ 05	
23	+	*a+k*
24	ZRCL 03	s
25	W^Z	*(a+k)^s*
26	Z/	*x^k / (a+k)^s*
27	Z+	*Σk*
28	Z#WR?	
29	*GTO 01*	
30	ZAVIEW	
31	END	

12.11. Exponential Integrals.

New since revision 4L, this section groups the Exponential Integral and related functions – all calculated using the Hypergeometric function representation.

Index	Function	Description	
1	**ZHGF**	Complex Hypergeometric function	*Author: Jean-Marc Baillard*
2	**ZEI**	Complex Exponential Integral	
3	**ZCI**	Complex Cosine Integral	
4	**ZHCI**	Complex Hyperbolic Cosine Integral	
5	**ZSI**	Complex Sine integral	
6	**ZHSI**	Complex Hyperbolic Sine Integral	
7	**ZERF**	Complex Error function	

The key enabler for this group is of course the MCODE implementation of the Complex Hypergeometric function **ZHGF** – written by Jean-Marc Baillard. See the excellent web-site at:
http://hp41programs.yolasite.com/complexhypergeo.php

The rest of the functions are easily obtained as simple and short FOCAL programs, using the well-know equivalence expressions. Their argument is a complex number, taken from the Z-level of the complex stack (XY registers). In terms of usability they are grouped in their own launcher, invoked by pressing [**H**] at the **Z"** prompt; that is:

[**Z**], [A], [H] →

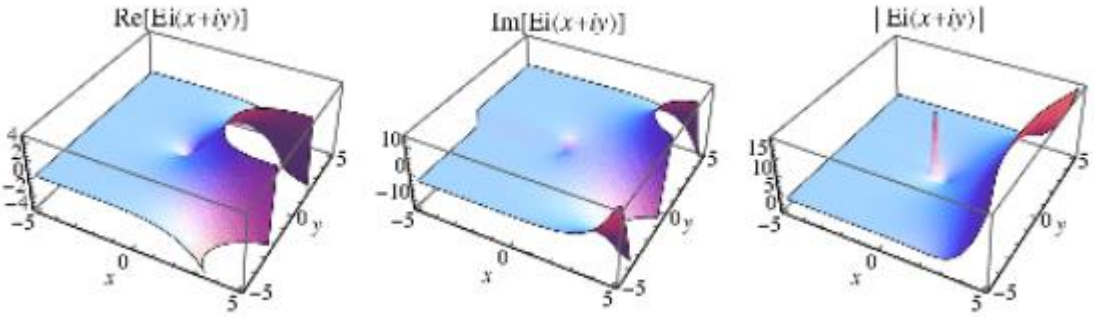
Examples.-

Calculate erf(1+i) and Ei(1+i)

1, ENTER^, [**Z**], [A], [H], [**"R"**] → 1,316+J0,190
1, ENTER^, [**Z**], [A], [H], [**"E"**] → 1,765+J2,388

Calculate Ei, Ci, Si and their hyperbolic counterparts for the same argument z=(1+i)

1, ENTER^, [**Z**], [A], [H], [**"S"**] → 1,104+J0,882
1, ENTER^, [**Z**], [A], [H], [**"H"**] → 0,882+J1,104
1, ENTER^, [**Z**], [A], [H], [**"C"**] → 0,882+J0,287
1, ENTER^, [**Z**], [A], [H], [**"I"**] → 0,882+J1,284



See the program listing in next page, showing the economy of programming when using a power horse like **ZHGF** to do all the heavy lifting for you.

FOCAL Listing: Exponential integrals. Uses R00 – R05

```
01  LBL "ZERF"
02  ZENTER^
03  Z^2
04  E
05  STO 01
06  1.5
07  STO 02
08  CLX
09  E
10  R^
11  R^
12  ZHGF
13  LASTZ
14  ZNEG
15  ZEXP
16  Z*
17  Z*
18  PI
19  SQRT
20  1/X
21  ST+ X
22  ST* Z
23  *
24  ZAVIEW
25  END
```

```
01  LBL "ZSI"
02  SF 00
03  GTO 00
04  LBL "ZHSI"
05  CF 00
06  LBL 00
07  ZENTER^
08  ZHALF
09  Z^2
10  FS? 00
11  ZNEG
12  .5
13  STO 01
14  3
15  *
16  STO 02
17  STO 03
18  CLX
19  E
20  ENTER^
21  2
22  R^
23  R^
24  ZHGF
25  Z*
26  ZAVIEW
27  END
```

```
01  LBL "ZEI"
02  E
03  STO 01
04  STO 02
05  E
06  +
07  STO 03
08  STO 04
09  ENTER^
10  R^
11  R^
12  ZHGF
13  LASTZ
14  Z*
15  LASTZ
16  GTO 01
17  LBL "ZCI"
18  SF 00
19  GTO 00
20  LBL "ZHCI"
21  CF 00
22  LBL 00
23  ZENTER^
24  ZHALF
25  Z^2
26  FS? 00
27  ZNEG
28  ZENTER^
29  E
30  STO 01
31  STO 02
32  CLX
33  2
34  STO 03
35  STO 04
36  1.5
37  STO 05
38  ST+ X
39  R^
40  R^
41  ZHGF
42  Z*
43  Z<>W
44  LBL 01
45  ZLN
46  Z+
47  ZGEU
48  Z+
49  ZAVIEW
50  END
```

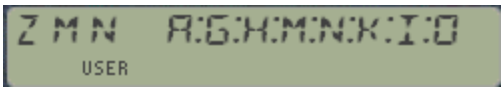
12.12. Exponential Integrals, Complex Means and General Methods Launchers.

All this many functions sure enough will benefit from having "theme" launchers grouping them, for easier access and logical segregation. The usability is enhanced and doesn't require overlays for the most frequente options within the groups.

The first one combines the Exponential Integrals and the Complex Means. Use the key sequence below to access it, and then the [SHIFT] key to toggle between uts two parts:

[**Z**], [**A**], [H]

[**Z**], [**A**], [H], [SHIFT]



See below the function correspondence for each launcher:

Exponential Integrals	
[E]	ZEI
[S]	ZSI
[H]	ZHSI
[C]	ZCI
[I]	ZHCI
[R]	ZERF
[F]	ZHG

Complex Means	
[A]	ZAMN
[G]	ZGMN
[H]	ZHMN
[M]	ZAGM
[N]	ZGHM
[K]	ZELK
[I]	ZINPT
[O]	ZOUPT

Finally the remaining Lauchers deal with Eliptical Functions and Complex Methods. You access these groups using the keyword combinations shown below:

[**Z**], [**A**], [R/S]

[**Z**], [**A**], [R/S], [SHIFT]

Elliptic Functions	
[1]	ZELIP1
[2]	ZELIP2
[L]	ZELK
[E]	ZELIPE
[K]	ZELIPK
[P]	ZELPKE

DFT/Other Functions	
[I]	ZIDFT
[D]	ZDFT
[S]	"ZSAM"
[C]	ZCTLN
[P]	ZPSIN
[M]	ZIGAM

Appendix. Saving & Restoring the Z-Stack in X-Memory. { **SAVEZS** , **GETZS** }

Two functions are included in the Deluxe 41Z Module to save and restore the complete complex stack buffer in extended memory. This includes all complex levels, the real stack and the current POLAR/REC settings. The functions are **SAVEZS** and **GETZS**.

In program execution, the file name is expected by these functions to be in ALPHA. For convenience, in RUN mode the functions will prompt for the file name automatically - remember that ALPHA is constantly being updated with the complex number values, so without this automated prompting feature you would need to re-write the file name in-between operations.

You can use them to preserve their contents in a permanent X-Mem file. Only one active complex buffer is allowed in the calculator, but you can choose from several X-Mem files holding different complex stacks, to upload their contents on demand. Therefore prior to executing **GETZS** you need to ensure that there's no buffer#8 in memory – you can use function CLB in the AMC_OS/X module for that. Failure to do so will generate the error message 'DUP BUF"

The X-mem file has a custom type "Z", with code=8. The file size is always 12 registers. If you're using the AMC_OS/X Module the CAT"4 enumeration includes support for this file type, which will be properly shown as a 'Z" type:

Where here the complex stack file name is "ZSTACK".

	Buffer Layout	
b11	non-zero	-
b10	L4 (U)	-
b9		-
b8	L3 (V)	-
b7		-
b6	L2 (W)	T
b5		Z
b4	L1 (Z)	Y
b3		X
b2	L0 (S)	L
b1		-
b0	Header	-

Note 3.- The Hypergeometric Function is also the preferred method used for the calculation of the Exponential Integrals and the Error function – which have been programmed as simple FOCAL examples of the former. See the descriptions in the SandMath module users' Manual for additional reference.

Note 4.- The programs supplied for the Polylogarithm and Lerch functions are simplified and necessarily non-rigorous, not using contour integrals or residues. See the references below for a formal treatment of the problem, clearly exceeding the scope of this manual.-

http://rspa.royalsocietypublishing.org/content/459/2039/2807.full.pdf
http://rspa.royalsocietypublishing.org/content/463/2080/897.full.pdf

Appendix.- Delta-Wye Transformation.

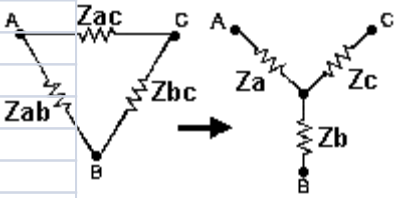
Here's a token of appreciation for the EE audiences – using the 41Z to tackle a classic: Delta-Wye impedance transformation for 3-phase systems. The simple program below is all there is to it – behold the power of the 41Z complex stack in action :-)

Delta <-> Why conversions			
LBL "D-Y	LBL "DYD"		
SF 00	ZRCL 00	*Za / Zab*	
GTO 00	ZRCL 01	*Zb / Zbc*	
LBL "Y-D	Z+	*Za+Zb / Zab+Zbc*	
CF 00	FC? 00		
LBL 00	*GTO 01*		
"Za"	ZRCL 02	*Zab*	
FS? 00	Z+	*Zab+Zbc+Zca*	
"1/-b"	ZINV	*1/(Zab+Zbc+Zca)*	
"1/-=?"	ZRPL^		
PROMPT	ZRCL 00	*Zab*	
ZSTO 00	ZRCL 02	*Zca*	
"Zb"	Z*	*ZabZca*	
FS? 00	Z*	*Za = ZabZca*	
"1/-c"	Z<>W	*1/(Zab+Zbc+Zca)*	
"1/-=?"	ZRCL 01	*Zbc*	
PROMPT	ZRCL 00	*Zab*	
ZSTO 01	Z*	*ZabZbc*	
"Zc"	Z*	*Zb = ZabZbc/(Zab+Zbc+Zca)*	
FS? 00	ZRUP	*1/(Zab+Zbc+Zca)*	
"1/-a"	ZRCL 02	*Zca*	
"1/-=?"	ZRCL 01	*Zbc*	
PROMPT	Z*	*ZbcZca*	
ZSTO 02	Z*	*Zc = ZbcZca/(Zab+Zbc+Zca)*	
XEQ "DYD"	*RTN*		
ZSTO 02	LBL 01		
ZRDN	LASTZ	*Zb*	
ZSTO 01	ZRCL 00	*Za*	
ZRDN	Z*	*ZaZb*	
ZSTO 00	ZRCL 02	*Zc*	
ZRDN	Z/	*ZaZb/Zc*	
ZRDN	Z+	*Zab = Za+Zb+ZaZb/Zc*	
ZVIEW 00	ZRCL 01	*Zb*	
ZVIEW 01	ZRCL 00	*Za*	
ZVIEW 02	Z/	*Zb/Za*	
RTN	ZRCL 02	*Zc*	
	Z*	*ZbZc/Za*	
	LASTZ	*Zc*	
	Z+	*Zc+ZbZc/Za*	
	ZRCL 01	*Zb*	
	Z+	*Zb+Zc+ZbZc/Za*	
	ZRCL 00	*Za*	
	ZRCL 01	*Zb*	
	Z/	*Za/Zb*	
	ZRCL 02	*Zc*	
	Z*	*ZaZc/Zb*	
	LASTZ	*Zc*	
	Z+	*Zc+ZaZc/Zb*	
	ZRCL 00	*Za*	
	Z+	*Za+Zc+ZaZc/Zb*	
	RTN		

$$Z_1 = \frac{Z_b Z_c}{Z_a + Z_b + Z_c}$$

$$Z_2 = \frac{Z_c Z_a}{Z_a + Z_b + Z_c}$$

$$Z_3 = \frac{Z_a Z_b}{Z_a + Z_b + Z_c}$$



Appendix 1.- Complex Buffer functions.

This appendix lists the buffer handling functions included in the 41Z DIAGNOSTICS module, and thus are not related to the Complex Number treatment per se. This set is only useful to diagnose problems or to bypass the normal execution of the module's "standard" functions, therefore its usage is not recommended to the casual user (i.e. do it at your own risk!).

Function	Description	Input	Output
-HP 41Z	Initializes Z Buffer	None	Buffer created
CLZB	Clears Z buffer	None	Buffler cleared
L1=XY?	Is L1 equal to XY?	None	Y/N, skip if false
L1<>L _	**Swap L1 & Level**	**Level# as suffix**	levels exchanged
L1<>LX	Swap L1 & Level	level in X	levels exchanged
L2=ZT?	Is L2 equal to ZT?	None	Y/N, skip if false
L2>ZT	Copies L2 into ZT	None	L2 copied to ZT
LVIEW _	**View Level**	**Level# as suffix**	*Transposed value!*
LVIEWX	View level by X	level in X	*Transposed value!*
PREMON	Copies XY into L0 and finds Zbuffer	Re(z) in X; Im(z) in Y	none
PSTMON	Copies XY into L1 and synch's up	Complex stack Z	Re(z) in X; Im(z) in Y
RG>ZB _ _	**Copies registers to Z buffer**	**Reg# as suffix**	data copied from registers
ST>ZB	Copies real stack to L1 & L2	None	stack copied to buffer
XY>L _	**Copies XY into Level**	**Level# as suffix**	XY copied to LEVEL
XY>L0	Copies XY into L0	Re(z) in X; Im(z) in Y	XY copied to L0
XY>L1	Copies XY into L1	Re(z) in X; Im(z) in Y	XY copied to L1
ZB>RG _ _	**copies buffer to registers**	**Reg# as suffix**	data copied to registers
ZB>ST	Copies L1 & L2 into real stack	None	buffer copied to Stack
ZBDROP	Drops Z buffer one level	None	levels dropped
ZBHEAD	Z buffer Header info	None	header register in ALPHA
ZBLIFT	Lifts Z buffer one level	None	buffer lifted
ZBSHOW	Shows Z Buffer	None	shows header & all levels

(*) Items highlighted in yellow indicate prompting functions.

Buffer layout. The complex buffer has 5 levels, labelled L0 to L4; that's 10 memory registers plus the header and footer registers – for a total of 12 registers. The function names in this group use the Level number (L0 to L4) to identify each level, as opposed to the **U, V, W**, and **Z** notation employed in previous sections of the manual.

	Buffer Layout		Buffer Details
b11	*non-zero*	-	*The buffer has 12 memory registers*
b10	**L4 (U)**	-	*Buffer registers are labeled b0 to b11*
b9		-	*Header is located at the bottom*
b8	**L3 (V)**	-	*A non-zero register is at the top*
b7		-	*Each Level uses two buffer registers*
b6	**L2 (W)**	T	*Levels are labeled L0 to L4*
b5		Z	
b4	**L1 (Z)**	Y	
b3		X	
b2	L0 (S)	L	
b1		-	
b0	Header	-	

The buffer header (b0 register) is placed at the lowest memory address. It contains the buffer id#, its size, and its initial address (when it was first created – no updates if it's re-allocated later on).

Buffer creation is done automatically by the 41Z module upon power on (when the 41 awakes from deep sleep), using the corresponding poll point in the module. The contents of the real stack registers XYZT is copied into the buffer levels L1 & L2 upon initialization.

The buffer is maintained by the 41 OS, which handles it when modifying the layout of main memory – either changing the SIZE settings, or modifying the user key assignments. The buffer id# is 8, and thus should be compatible with any other memory buffer that uses a different id# (an example of which are the TIMER alarms, with id#=10).

Should for any reason the buffer get damaged or erased (like when using the function **CLZB**), the message "NO Z-STACK" would appear when trying to execute any of the 41Z module functions. *To manually re-create the complex buffer* simply execute the first function in the module, "**–HP 41Z**" - either by using XEQ or the Complex Keyboard sequence "**Z**, SHIFT, **Z**". This requires at least 12 memory registers to be available or the error message "NO ROOM" will be shown.

Because the buffer can be dynamically re-allocated by the 41 OS upon certain circumstances, it's not possible to store its address to be reused by the functions. *Every 41Z function would first seek out the buffer address prior to proceeding with its calculation*. Fortunately this takes very little overhead time.

Buffer synchronization with the appropriate real-stack levels is also performed automatically by the 41Z functions, as follows:

- In the input phase (pre-execution), monadic functions will copy the XY contents into level L1 prior to executing their code. Dual functions will do the same for the second argument **Z**, and will use the current contents of the L2 level as first argument **W**.

- In the output phase (post-execution) the results will be placed in the complex buffer levels and then copied to the real stack registers as appropriate: XY for monadic functions, and XZYT for dual functions.

That's the reason why the real stack should just be considered as a *scratch pad* to prepare the data (like doing math on the real values), as only levels X,Y will be used. You must use **ZENTER^** to push the **W** argument into the complex level L2. In other words: real stack registers T,Z will be ignored!

The same consideration applies when performing chain calculations: because there's no automated complex stack lift, *the result of a monadic function would be overwritten by the subsequent input unless it is first pushed into the complex stack*, using **ZENTER^** *or another 41Z function that does stack lift.*

Example: Calculate Ln(1+i) + (2-i)

The following sequence use the direct data entry, entering Im(z) first.
1, ENTER^, **ZLN**, **ZENTER^**, 1, CHS, ENTER^, 2, **Z+** -> 2,347-j0,215

Some functions perform stack lift by default, and thus **ZENTER^** is not required before them.
They are as follows:

- **LASTZ**
- **ZRCL _ _**
- **ZREAL^** (also when using the complex real keypad, Z plus digit key)
- **ZIMAG^** (also when using the complex imaginary keypad, Z, radix, plus digit key)
- **^IM/AG** Probably the most intricate function in the module

The following sequence uses natural data entry - entering Re(z) first - as an alternative method for the previous example. Note that because **^IMG** does stack lift, it's not necessary to use **ZENTER^**

1, **^IMG,** 1, R/S**, ZLN,** 2, **^IMG,** 1, CHS, R/S**, Z+** -> 2,347-j0,215

Buffer synchronization with the real stack registers can be tested and forced using the following functions in this group:

L1=XY? - Tests for the first buffer level and XY registers
XY>L1 - Copies X,Y into level L1
L2=ZT? - Tests for second buffer level and Z,T registers
L2>ZT - Copies L2 into registers Z,T
ST>ZB - Copies real stack XYZT to buffer levels L1 & L2
ZB>ST - Copies L1 & L2 to the real stack XYZT

To dump the complete contents of the complex buffer into memory registers and back you can use these two complementary functions:

ZB>RG _ _ - Copies complex buffer to memory registers
RG>ZB _ _ - Copies memory registers to complex buffer

Note that **RG>ZB** won't check for valid header data, thus it expects the contents to be correct – like with a previously execution of **ZB>RG**. Remember that the header register is a non-normalized number (NNN), thus do not recall it using RCL or X<>.

Other functions to manipulate the contents of the buffer levels are:

L1<>L _ - swaps buffer level L1 and level given by prompt
L1<>LX - swaps buffer level L1 and level input in X
XY>L0 - copies registers X,Y into buffer level L0 (used to save arguments into LastZ)
XY>L _ - copies registers X,Y into buffer level given by prompt
ZBDROP - drops contents of complex buffer one level (used during **ZRDN**)
ZBLIFT - lifts contents of complex buffer one level (used by **ZRUP**, **ZENTER^** and others)

All these functions act on the complex buffer, but will not display the "resulting" complex number (i.e. will not trigger **ZAVIEW** upon completion). To see (view) the contents of the buffer levels without altering their position you can use the following functions:

LVIEW _ - prompts for level number (0 – 4)
LVIEWX - expects level number in X
ZBSHOW - lists the contents of all buffer levels
ZBHEAD - shows in Alpha the decoded buffer header

Note that with these functions all complex level contents will be shown transposed, that is: Im(z) + j Re(z).

finally, the other two functions are auxiliary and mainly used to perform action between the two lower and upper 4k-pages within the 41Z module: (*)

PREMON - Finds Z Buffer address, Copies XY into L0 and checks X,Y for ALPHA DATA
PSTMON - Copies the Z complex level into X.Y

(*) *Note: FAT entries for these two functions were removed in newer versions of the module.*

Because of its relevance and importance within the 41Z module, the following section lists the buffer creation and interrogation routines – pretty much the heart of the implementation. Consider that they are called at least twice every time a function is executed and you'll appreciate their crucial role in the whole scheme!

Remember that the buffer is refreshed (or created) each time the calculator is turned on, and that it gets reallocated when key assignments or other buffers (like timer alarms) are made – yet it's theoretically possible that it gets "unsynchronized" or even lost altogether, and therefore the assignment to the **–HP 41Z** function as well.

1	BUFFER	SYNCH2	A996	39C	PT= 0	*Synch2: write XYZT into [b3-b6]*
2	BUFFER	*(from upper)*	A997	130	LDI S&X	
3	BUFFER		A998	003	CON: 3	*X reg address*
4	BUFFER		A999	146	A=A+C S&X	*pointer to b3*
5	BUFFER		A99A	270	RAM SELECT	*select stack reg*
6	BUFFER		A99B	266	C=C-1 S&X	*decrease stack pointer*
			A99C	0EE	C<>B ALL	*store stack pointer in B*
b6	L2	T	A99D	038	READ DATA	*stack value*
b5	L2	Z	A99E	**0AE**	**A<>C ALL**	*swap stack value and bk address*
b4	L1	Y	A99F	**270**	**RAM SELECT**	*select bk*
b3	L1	X	A9A0	**0AE**	**A<>C ALL**	*swap bk address and stack value*
			A9A1	2F0	WRIT DATA	*write stack value into bk*
13	BUFFER		A9A2	166	A=A+1 S&X	*b(k+1) address*
14	BUFFER		A9A3	0EE	C<>B ALL	*restore stack address*
15	BUFFER		A9A4	3DC	PT=PT+1	
16	BUFFER		A9A5	*054*	*?PT=4*	*loop 4 times*
17	BUFFER		A9A6	*3A3*	*JNC -12d*	
18	BUFFER		A9A7	3E0	RTN	
1	INITIALIZE	*Header*	A9A8	*0AB*	*"+"*	
2	INITIALIZE	*Header*	A9A9	*01A*	*"Z"*	
3	INITIALIZE	*Header*	A9AA	*031*	*"1"*	
4	INITIALIZE	*Header*	A9AB	*034*	*"4"*	
5	INITIALIZE	*Header*	A9AC	*020*	*" "*	
6	INITIALIZE	*Header*	A9AD	*010*	*"P"*	*Check for Library#4 first, then*
7	INITIALIZE	*Header*	A9AE	*008*	*"H"*	*Create IO buffer 880C000…*
8	INITIALIZE	*Header*	A9AF	*02D*	*"-"*	*- unless it's already there*
9	INITIALIZE	-HP 41Z	A9B0	04E	C=0 ALL	*Programmable!*
10	INITIALIZE		A9B1	15C	PT= 6	
11	INITIALIZE		A9B2	110	LD@PT- 4	*put "4000" in ADR field*
12			A9B3	330	FETCH S&X	
13	*first we must check for Lib#4*		A9B4	106	A=C S&X	*put byte in A[S&X]*
14	*because [CHKBUF] resides*		A9B5	130	LDI S&X	
15	*in there (!)*		A9B6	*023*	CON:	*signature value*
16			A9B7	366	?A#C S&X	*are they different?*
17	INITIALIZE		A9B8	*0BB*	*JNC +23d*	*no, next thing*
18	INITIALIZE		A9B9	320	DSPTOG	*display ON*
19	INITIALIZE		A9BA	*3C1*	*?NC XQ*	*Enable & Clear Disp*
20	INITIALIZE		A9BB	*0B0*	*->2CF0*	*[CLLCDE]*
21	INITIALIZE		A9BC	*3BD*	*?NC XQ*	*Message Line*
22	INITIALIZE		A9BD	*01C*	*->07EF*	*[MESSL]*
23	INITIALIZE		A9BE	*00E*	*"N"*	
24	INITIALIZE		A9BF	*00F*	*"O"*	
25	INITIALIZE		A9C0	*020*	*" "*	
26	INITIALIZE		A9C1	*00C*	*"L"*	
27	INITIALIZE		A9C2	*009*	*"I"*	*"NO LIBRARY"*
28	INITIALIZE		A9C3	*002*	*"B"*	
29	INITIALIZE		A9C4	*012*	*"R"*	
30	INITIALIZE		A9C5	*001*	*"A"*	
31	INITIALIZE		A9C6	*012*	*"R"*	
32	INITIALIZE		A9C7	*219*	*"Y"*	
33	INITIALIZE		A9C8	*3DD*	*?NC XQ*	*Left Justified format*
34			A9C9	*0AC*	*->2BF7*	*[LEFTJ]*
35	*halting here means the rest*		A9CA	108	SETF 8	
36	*of polling points won't be done!*		A9CB	*201*	*?NC XQ*	
37			A9CC	*070*	*->1C80*	*[MSG105]*
38	INITIALIZE		A9CD	*3ED*	*?NC GO*	*HALT execution*
39	INITIALIZE		A9CE	*08A*	*-> 22FB*	*[ERR110]*

40	INITIALIZE	NEXT	A9CF	0D9	?NC XQ	Check for CX OS
41	INITIALIZE		A9D0	110	->4436	[NOCX4]
42	INITIALIZE	ZSTACK	A9D1	130	LDI S&X	
43	INITIALIZE		A9D2	008	CON: 8	Buffer id# in C(0)
			A9D3	2A9	?NC XQ	Check for Buffer
b11	non-zero	-	A9D4	10C	->43AA	[CHKBF4]
b10	L4	-	A9D5	0A3	JNC +20d	Not Found - Create it !!
b9		-	A9D6	038	READ DATA	reload id# in MS field
b8	L3	-	A9D7	2DC	PT= 13	
b7		T	A9D8	210	LD@PT- 8	Buffer id#
b6	L2	Z	A9D9	2F0	WRIT DATA	
b5		Y	A9DA	375	PORT DEP:	0.- write XYZT into [b3-b6]
b4	L1	X	A9DB	03C	XQ	to initialize L1 & L2
b3		-	A9DC	196	->A996	[SYNCH2]
b2	L0	L	A9DD	130	LDI S&X	A[S&X] holds b7 address
b1			A9DE	004	CON: 4	adds 4 to it
b0	Header		A9DF	206	C=C+A S&X	b11 addr
			A9E0	270	RAM SLCT	non-zero the last buffer reg
57	INITIALIZE		A9E1	2F0	WRIT DATA	this should do it
58	INITIALIZE		A9E2	04E	C=0 ALL	
59	INITIALIZE		A9E3	270	RAM SLCT	Select Chip0
60	INITIALIZE		A9E4	2CC	?FSET 13	Exit if PRG Running
61	INITIALIZE		A9E5	360	?C RTN	
62	INITIALIZE		A9E6	3AD	PORT DEP:	Show X,Y
63	INITIALIZE		A9E7	08C	GO	(Respects ZMODE)
64	INITIALIZE		A9E8	02A	->AC2A	[ZAVIEW]
65	INITIALIZE	CREATE	A9E9	066	A<>B S&X	First free reg. address (from .END.)
66	INITIALIZE		A9EA	04E	C=0 ALL	
67	INITIALIZE		A9EB	270	RAM SLCT	Select Chip0
68	INITIALIZE		A9EC	285	?NC XQ	
69	INITIALIZE		A9ED	014	->05A1	[MEMLFT]
70	INITIALIZE		A9EE	106	A=C S&X	number of "free regs"
71	INITIALIZE		A9EF	130	LDI S&X	Must be at least 12 free regs.
72	INITIALIZE		A9F0	00C	CON: 12	(header + 5 complex stack levels)
73	INITIALIZE		A9F1	306	?A<C S&X	Enough Memory?
74	INITIALIZE		A9F2	33D	?C GO	Show "No Room" msg
75	INITIALIZE		A9F3	0C3	->30CF	[NORMER]
76	INITIALIZE		A9F4	0E6	B<>C S&X	First free reg. address (from .END.)
77	INITIALIZE		A9F5	270	RAM SLCT	select buffer header
78	INITIALIZE		A9F6	106	A=C S&X	buffer address in A S&X
79	INITIALIZE		A9F7	2DC	PT= 13	
80	INITIALIZE		A9F8	210	LD@PT- 8	Buffer id#
81	INITIALIZE		A9F9	210	LD@PT- 8	Buffer id#
82	INITIALIZE		A9FA	010	LD@PT- 0	Buffer size
83	INITIALIZE		A9FB	310	LD@PT- C	Buffer size
84	INITIALIZE		A9FC	2F0	WRIT DATA	Store Header Reg
85	INITIALIZE		A9FD	2EB	JNC -35d	A9CA
1	BUFFER	NOBUFER	A9FE	215	?NC XQ	Build Msg - all cases
2	BUFFER		A9FF	0FC	->3F85	[APRMSG2]
3	BUFFER		AA00	00E	"N"	
4	BUFFER		AA01	00F	"O"	
5	BUFFER		AA02	020	" "	
6	BUFFER		AA03	01A	"Z"	
7	BUFFER		AA04	02D	"-"	MSG: "NO Z-STACK"
8	BUFFER		AA05	013	"S"	
9	BUFFER		AA06	014	"T"	
10	BUFFER		AA07	001	"A"	
11	BUFFER		AA08	003	"C"	
12	BUFFER		AA09	20B	"K"	
13	BUFFER		AA0A	1F1	?NC GO	Left], Show and Halt
14	BUFFER		AA0B	0FE	->3F7C	[APEREX]

Notice how we finish with **ZAVIEW** to show the current complex number in the stack upon buffer creation. [CHKBUF] does not create the buffer, but reads its address into register A and the content of the header into register C.

Appendix 2. Complex Keyboard key maps.

The following table shows the detailed key map supported by the **ΣZL** complex keyboard function launcher.

Level					Function
I	II	III	IV	V	Name
Z	1/X				ZINV
Z	SQRT				ZSQRT
Z	LOG				ZLOG
Z	LN				ZLN
Z	X<>Y				Z<>W
Z	RDN				ZRDN
Z	SIN				ZSIN
Z	COS				ZCOS
Z	TAN				ZTAN
Z	XEQ				^IMG _
Z	STO				ZSTO _ _
Z	RCL				ZRCL _ _
Z	SST				Z<> _ _
Z	ENT^				ZENTER^
Z	CHS				ZNEG
Z	EEX				Z^X
Z	-				Z-
Z	+				Z+
Z	*				Z*
Z	/				Z/
Z	0-9				Z0-Z9
Z	R/S				ZAVIEW
Z	,	0-9			ZJ0-ZJ9
Z	Z	1/X			W^1/Z
Z	Z	SQRT			ZPSI
Z	Z	LOG			ZLNG
Z	Z	LN			e^Z
Z	Z	X<>Y			Z<>V
Z	Z	RDN			ZQRT
Z	Z	XEQ			ZIMAG^
Z	Z	STO			ZREAL^
Z	Z	RCL			Z/I
Z	Z	SST			CLSTZ
Z	Z	ENT^			ZRPL
Z	Z	EEX			Z^1/X
Z	Z	-			Z#W?
Z	Z	7			ZWDET
Z	Z	8			ZWDIST
Z	Z	9			ZWANG
Z	Z	+			ZREAL?
Z	Z	4			ZIN?
Z	Z	5			ZWCROSS
Z	Z	*			ZIMAG?
Z	Z	1			ZUNIT?
Z	Z	2			ZWLINE
Z	Z	/			Z#0?
Z	Z	0			ZOUT?
Z	Z	,			ZWDOT
Z	Z	Z			Z<>U

Level					Function
I	II	III	IV	V	Name
Z		-HP 41Z			-HP 41Z
Z		Y^X			W^Z
Z		X^2			Z^2
Z		10^X			ZALOG
Z		e^X			ZEXP
Z		X<>Y			ZTRP
Z		RDN			ZRUP
Z		ASIN			ZASIN
Z		ACOS			ZACOS
Z		ATAN			ZATAN
Z		ASN			ZK?YN
Z		LBL			ZSIGN
Z		GTO			Z*I
Z		CAT			^IMG _
Z		ISG			ZCONJ
Z		RTN			X^Z
Z		CLX			CLZ
Z		X=Y?			Z=W?
Z		SF			ZNORM
Z		CF			ZMOD
Z		FS?			ZARG
Z		X<=Y?			Z=WR?
Z		BEEP			ZTONE
Z		P-R			ZREC
Z		R-P			ZPOL
Z		X>Y?			Z=I?
Z		FIX			ZRND
Z		SCI			ZINT
Z		ENG			ZFRC
Z		X=0?			Z=0?
Z		PI			ZGAMMA
Z		LASTX			LASTZ
Z		VIEW			ZVIEW _ _
Z			SIN		ZSINH
Z			COS		ZCOSH
Z			TAN		ZTANH
Z				SIN	ZASINH
Z				COS	ZACOSH
Z				TAN	ZATANH
Z	Z		SQRT		ZNXTNRT _
Z	Z		LN		ZNXTLN
Z	Z		SIN		ZNXTASN
Z	Z		COS		ZNXTACS
Z	Z		TAN		ZNXTATN
Z	Z			LOG	ZKBS
Z	Z			LN	ZYBS
Z	Z			COS	ZIBS
Z	Z			TAN	ZJBS
Z	Z			SIN	ZWL
Z	Z			SQRT	EIZ/IZ

Appendix 3.- Formula Compendium.

Elementary complex numbers and functions – By W. Doug Wilder.

$$j=\sqrt{-1}=e^{j\frac{\pi}{2}}=1\angle 90^o \qquad j^2=e^{j\pi}=1\angle 180^o=-1 \qquad -j=j^{-1}$$

$$Z=Re(Z)+jIm(Z)=x+jy=re^{j\theta}=r\angle\theta=r\cos\theta+jr\sin\theta \qquad r=|Z|=\sqrt{x^2+y^2} \qquad \theta=\tan_4^{-1}(y/x)$$

$$\overline{Z}=Z^*=x-jy=re^{-j\theta}=r\angle-\theta \qquad Z+Z^*=2Re(Z) \qquad Z-Z^*=j2Im(Z)$$

$$(Z_1Z_2)^*=Z_1^*Z_2^* \qquad (Z_1Z_2^*)^*=Z_2Z_1^* \qquad (Z_1/Z_2)^*=Z_1^*/Z_2^* \qquad (Z_1+Z_2)^*=Z_1^*+Z_2^* \qquad (Z_1-Z_2)^*=Z_1^*-Z_2^*$$

$$|Z_1Z_2|=|Z_1||Z_2| \qquad |Z_1/Z_2|=|Z_1|/|Z_2| \qquad |Z_1Z_2^*|=|Z_1Z_2| \qquad r^2=|Z|^2=ZZ^*=x^2+y^2$$

$$|Z_1+Z_2|^2=(Z_1+Z_2)(Z_1+Z_2)^*=Z_1Z_1^*+Z_1Z_2^*+Z_2Z_1^*+Z_2Z_2^*=|Z_1|^2+2Re(Z_1Z_2^*)+|Z_2|^2$$

$$Z_1+Z_2=(x_1+x_2)+j(y_1+y_2) \qquad Z_1-Z_2=(x_1-x_2)+j(y_1-y_2) \qquad |Z_1+Z_2|\le|Z_1|+|Z_2|$$

$$Z_1Z_2=(x_1x_2-y_1y_2)+j(x_1y_2+y_1x_2)=r_1r_2\angle(\theta_1+\theta_2)=r_1r_2e^{j(\theta_1+\theta_2)} \qquad Re(1/Z^*)=Re(1/Z)$$

$$Z_1/Z_2=(x_1x_2+y_1y_2+j(y_1x_2-x_1y_2))/(x_2^2+y_2^2)=r_1/r_2\angle(\theta_1-\theta_2) \qquad Z^{-1}=(x-jy)/(x^2+y^2)=e^{-j\theta}/r$$

$$\overline{Z}_1Z_2=(x_1x_2+y_1y_2)+j(x_1y_2-y_1x_2)=r_1r_2\angle(\theta_2-\theta_1)=Z_1\bullet Z_2+jZ_1\times Z_2 \qquad (Z_1,Z_2=2D\ vectors)$$

$$Z^2=x^2-y^2+j2xy=r^2e^{j2\theta} \qquad Z^{1/2}=r^{1/2}e^{j\theta/2}\ (principal)$$

$$\pi=3.14159\ 26535\ 89793\ 23846\ 264\ldots \qquad e^{Z+j2\pi n}=e^Z \qquad e=2.71828\ 18284\ 59045\ 23536\ 028\ldots$$

$$e^Z=e^xe^{jy}=e^x\angle y=e^x\cos y+je^x\sin y \qquad e^{Z_1}e^{Z_2}=e^{Z_1+Z_2} \qquad (e^{Z_1})^{Z_2}=e^{Z_1Z_2} \quad (-\pi<\theta_1\le\pi)$$

$$e^{-Z}=1/e^Z \qquad e^{\ln Z}=Z \qquad e^{jZ}=\cos(Z)+j\sin(Z) \qquad e^Z=\cosh(Z)+\sinh(Z)=\cos(jZ)-j\sin(jZ)$$

$$\ln Z=\ln r+j\theta=\ln\sqrt{x^2+y^2}+j\tan_4^{-1}(y/x)+j2\pi n \qquad e^{-jZ}/(-jZ)=(\sin Z+j\cos Z)/Z=h_0^{(2)}(Z)$$

$$Z_1\ln Z_2=\ln Z_2^{Z_1} \qquad \ln Z_1+\ln Z_2=\ln(Z_1Z_2) \qquad \ln 0=\infty \qquad \ln e^Z=Z$$

$$Z_2^{Z_1}=e^{Z_1\ln Z_2} \qquad \mathrm{Log}_aZ=\ln Z/\ln a \qquad \ln j=0+j\pi/2 \qquad \ln 1=0 \qquad \ln(-1)=0+j\pi$$

$$\frac{\partial}{\partial Z}Z^a=aZ^{a-1} \qquad \frac{\partial}{\partial Z}e^{aZ}=ae^{aZ} \qquad \frac{\partial}{\partial Z}a^Z=a^Z\ln a \qquad \frac{\partial}{\partial Z}\ln Z=\frac{1}{Z} \qquad \int e^{aZ}dZ=\frac{e^{aZ}}{a} \qquad \int\frac{dZ}{Z}=\ln Z$$

$$e^Z=1+\frac{Z}{1!}+\frac{Z^2}{2!}+\frac{Z^3}{3!}+\ldots \qquad \ln Z=\frac{2}{1}\left(\frac{Z-1}{Z+1}\right)+\frac{2}{3}\left(\frac{Z-1}{Z+1}\right)^3+\frac{2}{5}\left(\frac{Z-1}{Z+1}\right)^5+\ldots\ (Re(Z)\ge 0)$$

$$\sin Z=(-j/2)(e^{jZ}-e^{-jZ})=(-j/2)(e^{jZ}-1/e^{jZ}) \qquad \sin^{-1}Z=-j\ln\left(jZ+\sqrt{1-Z^2}\right)$$

$$\cos Z=(1/2)(e^{jZ}+e^{-jZ})=(1/2)(e^{jZ}+1/e^{jZ}) \qquad \cos^{-1}Z=-j\ln\left(Z+\sqrt{Z^2-1}\right)$$

$$\tan Z=-j\frac{e^{jZ}-e^{-jZ}}{e^{jZ}+e^{-jZ}}=-j\frac{e^{j2Z}-1}{e^{j2Z}+1} \qquad \tan^{-1}Z=-\frac{j}{2}\ln\left(\frac{1+jZ}{1-jZ}\right) \qquad \tan_8^{-1}(Z_2/Z_1)=-j\ln\left(\frac{Z_1+jZ_2}{\sqrt{Z_1^2+Z_2^2}}\right)$$

$$\frac{\partial}{\partial Z}\cos Z=-\sin Z \qquad \frac{\partial}{\partial Z}\sin Z=\cos Z \qquad \int\cos Z\,dZ=\sin Z \qquad \int\sin Z\,dZ=-\cos Z$$

$$\sin Z=Z-\frac{Z^3}{3!}+\frac{Z^5}{5!}-\frac{Z^7}{7!}+\ldots \qquad \cos Z=1-\frac{Z^2}{2!}+\frac{Z^4}{4!}-\frac{Z^6}{6!}+\ldots$$

$$\sinh Z=(1/2)(e^Z-e^{-Z})=-j\sin(jZ) \qquad \sinh^{-1}Z=\ln\left(Z+\sqrt{Z^2+1}\right)$$

$$\cosh Z=(1/2)(e^Z+e^{-Z})=\cos(jZ) \qquad \cosh^{-1}Z=\ln\left(Z+\sqrt{Z^2-1}\right)$$

$$\tanh Z=\frac{e^Z-e^{-Z}}{e^Z+e^{-Z}}=\frac{e^{2Z}-1}{e^{2Z}+1}=-j\tan(jZ) \qquad \tanh^{-1}Z=\frac{1}{2}\ln\left(\frac{1+Z}{1-Z}\right)$$

$$\cos Z=\cos x\cosh y-j\sin x\sinh y \qquad \sin Z=\sin x\cosh y+j\cos x\sinh y$$

Appendix 4.- Quick Reference Guide.

The tables in the following six pages list all 41Z functions in alphabetical order.
New functions in the Deluxe edition have pink background. Sub-functions are in brown font color.

#	Function	Description	Formula	Input	Output	Comments
1	**-HP 41Z**	***Initializes Complex Stack***	***Z=XY; W=ZT***	***none***	*Initializes Z buffer & ZAVIEW*	*runs on CALC ON*
2	**W^1/Z**	Complex Y^1/X	w^1/z = exp(Ln w / Z)	w in ***W***; z in ***Z*** (XY)	w^1/z in ***Z*** (XY)	*Drops Buffer*
3	**W^Z**	Complex Y^X	w^z = exp(z*Ln w)	w in ***W***; z in ***Z*** (XY)	w^z in ***Z*** (XY)	*Drops Buffer*
4	**ZF# _ _ _**	Launcher by index	n/a	Sub-function index	Executes Sub-function	
5	**ZF$ _**	Launcher by Name	n/a	Sub-function Name	Executes Sub-function	
6	**Z+**	Complex addition	(x1+x2) + i (y1+y2)	w in ***W***; z in ***Z*** (XY)	w+z in ***Z*** (XY)	*Drops Buffer, LastZ*
7	**Z-**	Complex substraction	w-z = w + (-z)	w in ***W***; z in ***Z*** (XY)	w-z in ***Z*** (XY)	*Drops Buffer, LastZ*
8	**Z***	Complex multiplication	(x1*x2 - y1*y2) + i (x1*y2 + y1*x2)	w in ***W***; z in ***Z*** (XY)	w*z in ***Z*** (XY)	*Drops Buffer, LastZ*
9	**Z/**	Complex division	w/z = w * (1/z)	w in ***W***; z in ***Z*** (XY)	w/z in ***Z*** (XY)	*Drops Buffer, LastZ*
10	**Z^1/X**	Hybrid Y^X	z^1/n = r^1/n * exp(i*Arg/n)	x in X reg; z in Y,Z regs	z^1/x in ***Z*** (XY)	*does LastZ*
11	**Z^2**	Complex X^2	z^2 = r^2 * exp(2i*Arg)	z in ***Z*** (XY)	z^2 in ***Z***, (XY)	*does LastZ*
12	**Z^3**	Cubic power	z=z^3	z in ***Z*** (Im in Y, Re in X)	result in ***Z*** (XY)	*more accurate than Z^X*
13	**Z^X**	Hybrid Y^X	z^n = r^n * exp(i*n*Arg)	x in X reg; z in Y,Z regs	z^x in ***Z***, (XY)	*does LastZ*
14	**Z=0?**	Is z=0?	is z=0?	z in ***Z*** (XY)	YES/NO (skips if false)	
15	**Z=I?**	Is z=I?	is z=i?	z in ***Z*** (XY)	YES/NO (skips if false)	
16	**Z=W?**	Is z=w?	is z=w?	w in ***W***; z in ***Z*** (XY)	YES/NO (skips if false)	
17	**Z=WR?**	are z & w equal if rounded?	is Rnd(z)=Rnd(w)?	w in ***W***; z in ***Z*** (XY)	YES/NO (skips if false)	
18	**Z#0?**	is z equal to zero?	is z#0?	z in ***Z*** (XY)	YES/NO (skips if false)	
19	**Z#W?**	Is z equal to w?	is z=w?	w in ***W***; z in ***Z*** (XY)	YES/NO (skips if false)	
20	**ZACOS**	Complex ACOS	acos z = pi/2 - asin z	z in ***Z*** (XY)	acos(z) in ***Z*** (XY)	*does LastZ*
21	**ZALOG**	Complex 10^X	e^[z*ln(10)]	z in ***Z*** (XY)	10^z in ***Z*** (X,Y) and ALPHA	*does LastZ*
22	**ZASIN**	Complex ASIN	asin z = -i * asinh (iz)	z in ***Z*** (XY)	asin(z) in ***Z***, (XY)	*does LastZ*
23	**ZATAN**	Complex ATAN	atan z = -i * atanh (iz)	z in ***Z*** (XY)	atan(z) in ***Z*** (XY)	*does LastZ*
24	**ZCOS**	Complex COS	cos z = cosh (iz)	z in ***Z*** (XY)	cos(z) in ***Z*** (XY)	*does LastZ*
25	**ZEXP**	Complex e^X	e^x * e^(iy)	z in ***Z*** (XY)	e^z in ***Z*** (XY) and ALPHA	*does LastZ*
26	**ZHACOS**	Complex Hyp. ACOS	acosh z = Ln[z + SQ(z^2 - 1)]	z in ***Z*** (XY)	acosh(z) in ***Z*** (XY)	*does LastZ*
27	**ZHASIN**	Complex Hyp. ASIN	asinh z = Ln[z + SQ(z^2 + 1)]	z in ***Z*** (XY)	asinh(z) in ***Z*** (XY)	*does LastZ*
28	**ZHATAN**	Complex Hyp. ATAN	atanh z = 1/2 * Ln[(1+z)/(1-z)]	z in ***Z*** (XY)	atanh(z) in ***Z*** (XY)	*does LastZ*
29	**ZHCOS**	Complex Hyp. COS	cosh z = 1/2 * [e^z + e^-z]	z in ***Z*** (XY)	cosh(z) in ***Z*** (XY)	*does LastZ*
30	**ZHSIN**	Complex Hyp. SIN	sinh z = 1/2 * [e^z - e^-z]	z in ***Z*** (XY)	sinh(z) in ***Z*** (XY)	*does LastZ*

#	Function	Description	Formula	Input	Output	Comments
31	**ZHTAN**	Complex Hyp. TAN	tanh z = (e^z-e^-z)/(e^z+e^-z)	z in ***Z*** (XY)	tanh(z) in ***Z*** (XY)	*does LastZ*
32	**ZCF2V _**	Complex Continued Fractions	f(z) = B(0) + A1/[B1 +A2/[B2 + A3/[B3+..]]]	User program with Bn and An	F(Z) if convergence	*Prompts for Prgm Name*
33	**ZDERV _**	Complex Function Derivatives	df/dz and d2f/dz2	User program w/ f(z)	f"(z) in "W", f'(z) in "Z"	*Prompts for Prgm Name*
34	**ZINT?**	Checks if Z is an integer number	are Im(z)=0 *and* FRC[Re(z)]=0?	z in ***Z*** (Im in Y, Re in X)	YES/NO (skips if false)	*used in Bessel fncs*
35	**ZINV**	Complex Inversion	x/(x^2 + y^2) - i y/(x^2 + y^2)	z in ***Z*** (XY)	1/z in ***Z*** (XY) and ALPHA	*does LastZ*
36	**ZLN**	Complex LN	ln(z) = ln(r) + i*Arg	z in ***Z*** (XY)	Ln(z) in ***Z*** (XY)	*does LastZ*
37	**ZLOG**	Complex LOG	log(z) = ln(z)/ln(10)	z in ***Z*** (XY)	Log(z) in ***Z*** (X,Y)	*does LastZ*
38	**ZNEG**	Complex CHS	-z = -x - iy	z in ***Z*** (XY)	-z in ***Z*** (XY)	*does LastZ*
39	**ZOUT?**	Is z outside the unit circle?	is |z|>1?	z in ***Z*** (XY)	YES/NO (skips if false)	
40	**ZPI***	Product by pi	z*p	z in ***Z*** (XY)	result in ***Z*** (XY)	*more accurate than FOCAL*
41	**ZGSS?**	Is z Gaussian?	Re(z) and Im(z) integers?	z in ***Z*** (XY)	YES/NO (skips if false)	
42	**ZRND**	Rounds Z to display settings	rounded values to display	z in ***Z*** (XY)	Rounded Re & Im in ***Z*** (XY)	*does LastZ*
43	**ZSIN**	Complex SIN	sin z = -i *sinh (iz)	z in ***Z*** (XY)	sin(z) in ***Z*** (XY)	*does LastZ*
44	**ZSQRT**	Complex SQRT (Direct)	sqr(z)=sqr(r) * e^(i*Arg/2)	z in ***Z*** (XY)	main value of z^1/2 in ***Z*** (XY)	*does LastZ*
45	**ZTAN**	Complex TAN	tan z = - i * tanh (iz)	z in ***Z*** (XY)	tan(z) in ***Z*** (XY)	*does LastZ*
46	**ZUNIT?**	Is z on the unit circle?	is |z|=1?	z in ***Z*** (XY)	YES/NO (skips if false)	
47	**-ZSTACK**	***Section Header***	***n/a***	***none***	*Shows "Running…" msg*	
48	**CLZ**	Clears Z	Re(z)=0=Im(z)	none	***Z*** level (XY) cleared	
49	**CLZST**	Clears Z-Stack	n/a	none	Z-Stack Cleared	
50	**LASTZ**	Complex LASTX	n/a	none	Last z in X,Y regs;	*Lifts Buffer*
51	**ZAVIEW _ _**	Shows Complex Z	n/a	z in ***Z*** (XY)	Shows z in ALPHA	
52	**ZENTER^**	Copies Z into the W register	n/a	z in ***Z*** (XY)	Pushes z one level Up	*Lifts Buffer*
53	**Z<> _ _**	Complex Exchange	n/a	Reg# as suffix	Exchanges Z with regs contents	*Prompting*
54	**Z<>ST _ _**	Exchanges Z and Level#	n/a	z in XY, level# in prompt	z in L#; L# in L1 & X,Y	*Prompting*
55	**Z<>W**	Exchange Z and W (L2)	n/a	w in ***W***, z in ***Z*** (XY)	z in L2 & Z,T w in L1 & X,Y	
56	**ZIMAG^**	Enter imaginary number	n/a	Im(z) in X	zero in X; Im(z) in Y	*Lifts Buffer*
57	**ZRCL _ _**	Complex RCL	n/a	Reg# as suffix	z in X,Y - lifts stack	*Lifts Buffer, Prompting*
58	**ZRDN**	Z-Stack Roll Down	n/a	Stack Levels	Rolls Down stack	*Drops Buffer*
59	**ZREAL^**	Enter Real number in Z	n/a	Re(z) in X	Re(z) in X;, Zero in Y	*Lifts Buffer*
60	**ZRPL^**	Replicates z in all levels	L4=L3=L2=L1	z in ***Z*** (XY)	z in all 4 levels	*Lifts Buffer*
61	**ZRUP**	Z-Stack Roll Up	n/a	Stack Levels	Rolls Up stack	*Lifts Buffer*
62	**ZSTO _ _**	Complex STO	n/a	Reg# as suffix	Stores z in consequtive regs	*Prompting*
63	**ZVIEW _ _**	Complex View	n/a	Reg# as suffix	Shows z in ALPHA	*Prompting*
64	**ZK?YN _**	Block Key Assignments	n/a	prompt-driven	Makes / Removes assignments	*may do PACKING*
1	**^IM/AG _**	***Natural Data Entry***	***Re ^ IM or r ^ arg***	***Re(z) in X, Im(Z) as suffix***	*z in **Z** (XY), stack lifted*	*Prompting, Lifts Buffer*
2	**GETSZ _**	Get z=Stack file from X-Mem	n/a	File Name in Alpha	Copies file to Buffer #8	*Includes REC/POLAR*

#	Function	Description	Formula	Input	Output	Comments
3	**NXTACS**	Next ACOS Value	z1,2 = +/- z0 + 2p	z0 in ***Z*** (XY)	z1 in ***W,*** z2 in ***Z*** (XY)	*does LastZ*
4	**NXTASN**	Next ASIN Value	z1,2 = +/- z0 + 2p/2	z0 in ***Z*** (XY)	z1 in ***W,*** z2 in ***Z*** (XY)	*does LastZ*
5	**NXTATN**	Next ATAN value	z1,2 = z0 +/- p	z0 in ***Z*** (XY)	z1 in ***W,*** z2 in ***Z*** (XY)	*does LastZ*
6	**NXTLN**	Next Ln(z)	next(k) = Ln(z) + 2kp J	LN(z) in ***Z*** (XY) regs	z1 in ***W,*** z2 in ***Z*** (XY)	*does LastZ*
7	**NXTRTN _**	Next Complex Root	next(k) = z^1/n * e^(2kp/n J)	n in X reg.; z^1/n in Z,Y regs	$z^{1/n}$ * e^(2p/n J) in ***Z*** (XY)	*does LastZ*
8	**SAVEZS _**	Saves z-Buffer to X-Mem	n/a	File Name in ALPHA	Copies buffer #8 to File	*Includes REC/POLAR*
9	**ZCHSX**	Sign Change by X	$(-1)^n$ * z	x in X reg; z in Y,Z regs	{(-1)^x * z} in ***Z*** (XY)	*does LastZ*
10	**ZGEU**	Euler's gamma constant	γ=0,577215665	none	g constant as complex	*Lifts Buffer*
11	**ΣZL _**	Complex keyboard launcher	n/a	Prompt-driven	Launches function	*prompting, launcher*
12	**ZPL**	Complex Polynomial Evaluation	P(z) = Σ ak z^k	Control word bbb.eee	Polynomial result	*Coeffs. Expected in ZRegs*
13	**ZRC+ _ _**	RCL addition	Z= Z + cR	z in ***Z*** (XY), data in cR	Adds cR to z	*does LastZ*
14	**ZRC- _ _**	RCL subtraction	Z = z – cR	z in ***Z*** (XY), data in cR	Subtracts cR from z	*does LastZ*
15	**ZRC* _ _**	RCL product	Z = z * cR	z in ***Z*** (XY), data in cR	Multiplies z by cR	*does LastZ*
16	**ZRC/ _ _**	RCL division	Z = z / cR	z in ***Z*** (XY), data in cR	Divides z by cR	*does LastZ*
17	**ZST+ _ _**	STO Addition	cR = cR + z	z in ***Z*** (XY), data in cR	Adds z to complex register#	*prompting*
18	**ZST- _ _**	STO Subtraction	cR = cR - z	z in ***Z*** (XY), data in cR	Subtract z from complex register#	*prompting*
19	**ZST* _ _**	STO Multiply	cR = cR * z	z in ***Z*** (XY), data in cR	Multiplies z to complex register#	*prompting*
20	**ZST/ _ _**	STO Divide	cR = cR / z	z in ***Z*** (XY), data in cR	Divides complex register by z	*prompting*
21	**-ZVECTOR**	***Section Header***	***n/a***	***none***	***Displays Revision Number***	
22	**POLAR**	Sets POLAR mode on	sets the Polar flag in Buffer	none	shows Re(z)+J Im(z)	
23	**RECT**	Sets RECT mode on	clears the Polar flag in Buffer	none	shows r <) arg	
24	**ZAGM**	Arithmetic-Geometric Mean	AGM	w in ***W***, z in ***Z*** (XY)	Result in Z(XY)	*does LastZ*
25	**ZARG**	Argument of Z	atan(y/x)	z in ***Z*** (XY)	Arg(z) in X, (Y reg void)	*zeroes Y, LastZ*
26	**ZMOD**	Module of Z	|z|=sqr(x^2+y^2)	z in ***Z*** (XY)	Mod(z) in X, (Y reg void)	*zeroes Y, LastZ*
27	**ZNORM**	Norm of Z (I.e. square of Module)	||z||=|z| ^2	z in ***Z*** (XY)	(mod(z)^2) in X,Y	*zeroes Y, LastZ*
28	**ZPOL**	Converts to Polar notation	R-P	z in ***Z*** (XY)	Mod(z) in X; Arg(z) in Y	*does LastZ*
29	**ZREC**	Convers to Rectangular notation	P-R	Mod(z) in X; Arg(z) in Y	Re(z) in X; Im(z) in Y	*does LastZ*
30	**ZWANG**	Angle between Z and W	arg(zw) = Arg(z) - Arg(w)	z in ***Z*** (XY)	ang(z,w) in X (Y void)	*Drops Buffer LastZ*
31	**ZWCROSS**	Cross product of Z and W	z x w = |z| *|w| *Sin(Angle)	w in ***W***, z in ***Z*** (XY)	z x w in X (Y void)	*Drops Buffer LastZ*
32	**ZWDET**	Determinant of Z and W	|zw| = x2*y1 - y2*x1	w in ***W***, z in ***Z*** (XY)	det(z,w) in X (Y void)	*Drops Buffer LastZ*
33	**ZWDIST**	Distance between Z and W	|w-z| = SQR[(x2-x1)^2 - (y2-y1)^2]	w in ***W***, z in ***Z*** (XY)	dist(z,x) in X (Y void)	*Drops Buffer LastZ*
34	**ZWDOT**	Dot product of Z and W	z*w = x1*x2 + y1*y2	w in ***W***, z in ***Z*** (XY)	dot(z,w) in X, (Y void)	*Drops Buffer LastZ*
35	**ZWLINE**	Line equation defined by Z and W	a=(y1-y2) / (x1-x2)	w in ***W***, z in ***Z*** (XY)	y=ax+b in ALPHA; b in Y, a in X	*Drops Buffer LastZ*
36	**ZWLOG**	Base-w Logarithm	base w in W, arg. In Z	w in ***W***, z in ***Z*** (XY)		*Drops Buffer, LastZ*
37	**-HL ZMATH**	***Section Header***	***Calculates 2^x-1***	***x in X***	***Result in X***	*used in ZZETA*

#	Function	Description	Formula	Input	Output	Comments
38	**ZAWL**	Inverse of Lambert W	z* e^z	z in ***Z*** (XY)	result in ***Z*** (XY)	*does LastZ*
39	**ZBS#**	Bessel subroutine 1st./2nd. Kind	see manual, Flag 6 controls case	w in W, z/2 in Z	w in ZR00, z/2 in ZR01	FOCAL
40	**ZCI**	Cosine Integral	Ci(z) = -(z^2/4) F_{23}(1, 1; 2, 2; 3/2, -z^2/4)	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
41	**ZCRT**	Complex Cubic Eq. Roots	Cubic ecuation roots	A,B,C,D in Z-Stack	roots in ***V***, ***W***, and ***Z*** (XY) levels	FOCAL
42	**ZEI**	Exponential Integral	Ei = γ + ln|z|+ z* F_{22}(1,1; 2,2; z)	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
43	**ZERF**	Error Function	erf(z) = 2z/sqr(π) e^(-z^2) F_{11}(1, 3/2; z^2)	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
44	**ZGAMMA**	Complex G(z) for z#0, -1, -2…	Lanczos approximation	z in ***Z*** (XY)	G(z) in ***Z*** (XY)	*uses reflection for Re(z)<0*
45	**ZHCI**	Hyperbolic Cosine Integral	Chi(z) = (z^2/4) F_{23}(1, 1; 2, 2; 3/2, z^2/4)	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
46	**ZHGF**	Hypergeometric Function	See manual	see manual	result in ***Z*** (XY)	*by Jean-Marc Baillard*
47	**ZHSI**	Hyperbolic Sine Integral	Shi(z) = z * F_{12} (1/2 , 3/2, 3/2, z^2/4)	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
48	**ZIBS**	Bessel I function	see manual	w in ***W***, z in ***Z*** (XY)	I(w,z) in ***Z*** (XY)	FOCAL
49	**ZJBS**	Bessel J function	see manual	w in ***W***, z in ***Z*** (XY)	J(w,z) in ***Z*** (XY)	FOCAL
50	**ZKBS**	Bessel K function	see manual	w in ***W***, z in ***Z*** (XY)	K(w,z) in ***Z*** (XY)	FOCAL
51	**ZLI2**	Dilogarithm	Li(2,z) = Σ(z^k /k^2); k=1,2…	z in ***Z*** (XY)	result in ***Z*** (XY)	*by Jean-Marc Baillard*
52	**ZLIN**	Polylogarithm	Li(s,z) = Σ(z^k /k^s); k=1,2…	order w in W; arg. z in Z	result in ***Z*** (XY)	FOCAL
53	**ZLNG**	Gamma Logarithm function	Stirling method w/ correction	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
54	**ZLRCH**	Lerch Transcendent	Fi(z,s,a) = Σ[z^k /(k+a)^s]; k=,0,1…	s,a, z in ***U***, ***W***, and ***Z***(XY)	result in ***Z*** (XY)	FOCAL
55	**ZPROOT**	Roots of complex polynomials	Iterative	Prompt-driven	roots in ***W*** and ***Z*** (XY) levels	*by Valentin Albillo*
56	**ZPSI**	Complex Digamma	Approximation	z in ***Z*** (XY)	Psi(z) in X,Y regs. And ALPHA	FOCAL
57	**ZQRT**	Complex Quadratic Eq. Roots	Quadratic ecuation roots	A,B,C in Zstack	Calculates roots of equation	FOCAL
58	**ZSHK1**	Spherical Hankel h1	$h^{(1)}$(w,z)	order w in W; arg. z in Z	result in ***Z*** (XY)	FOCAL
59	**ZSHK2**	Spherical Hankel h2	$h^{(2)}$(w,z)	order w in W; arg. z in Z	result in ***Z*** (XY)	FOCAL
60	**ZSI**	Sine Integral	Si(z) = z * F_{12} (1/2 , 3/2, 3/2, -z^2/4)	z in ***Z*** (XY)	result in ***Z*** (XY)	FOCAL
61	**ZSOLVE**	Solves for F(z)=0	Newton's method	Fnc. name in R06	Calculates one root for f(z)	FOCAL
62	**ZWL**	Lambert W function	see manual	z in ***Z*** (XY)	W(z) in ***Z*** (XY)	FOCAL
63	**ZYBS**	Bessel Y function	see manual	w in ***W***, z in ***Z*** (XY)	Y(w,z) in ***Z*** (XY)	FOCAL
64	**ZZETA**	Riemann Zeta function	Borwein Algorithm	z in ***Z*** (XY)	result in ***Z*** (XY)	*by Jean-Marc Baillard*
0	**-IMAGINE**	***Section Header***	n/a	n/a	n/a	
1	**1/Z**	alternative ZINV (Uses TOPOL)	1/r * exp(-i arg)	z in ***Z*** (XY)	1/z in X,Y registers and ALPHA	*does LastZ*
2	**e^Z**	alternative ZEXP	e^z = e^x * (cos y + i sin y)	z in ***Z*** (XY)	exp(z) in **Z** (XY)	*does LastZ*
3	**EIZ/IZ**	spherical hankel h1(0,z)	$h^{(1)}$(0,z) = exp(i*z) / i*z	z in ***Z*** (XY)	r esult in ***Z*** (XY)	*does LastZ*
4	**SQRTZ**	Alternative SQRT (Uses TOPOL)	sqr(z)=sqr(r) * e^(i*Arg/2)	z in ***Z*** (XY)	main value of z^1/2 in ***Z*** (XY)	*does LastZ*
5	**X^1/Z**	Hybrid Y^X	a^1/z = exp(1/ z*Ln a)	x in X reg; z in Y,Z regs	x^z in ***Z*** (XY)	*does LastZ*
6	**X^Z**	Hybrid Y^X	a^Z = exp(z*Ln a)	x in X reg; z in Y,Z regs	x^z in ***Z*** (XY)	*does LastZ*
7	**Z*I**	Multiplies by I (90 deg. Rotation)	iz = -Im(z) + I Re(z)	z in ***Z*** (XY)	z*i in L1 & XY	*does LastZ*
8	**Z/I**	Divides by I (-90 deg. Rotation)	iz = -Im(z) + I Re(z)	z in ***Z*** (XY)	z*i in L1 & XY	*does LastZ*

#	Function	Description	Formula	Input	Output	Comments
9	**ZBSL _**	Bessel Functs. Sub-Launcher	n/a	Prompts for Function	Executes Function	
10	**ZCONJ**	Complex Conjugate	conj = x – iy	z in ***Z*** (XY)	Inverts sign of Im(z)	*does LastZ*
11	**ZDISP**	Displays Z in LCD	Z = Re:Im	Values in Y, X	String in LCD	*No negative values!*
12	**ZDBL**	Doubles z	2*z	z in ***Z*** (XY)	2z in Z (XY)	*does LastZ*
13	**ZFRC**	Makes Re(z), Im(z) fractional	Int(Re(z) = Int(Im(z)) = 0	z in ***Z*** (XY)	Result in ***Z*** (XY)	*does LastZ*
14	**ZHALF**	Halves z	z/2	z in ***Z*** (XY)	z/2 in Z (XY)	*Does LastZ*
15	**ZHGF _**	Hypergeometric Launcher	n/a	Prompts for choice	Executes function	
16	**ZHYP _**	Hyperbolics Launcher	n/a	Prompts for choice	Executes function	
17	**ZIMAG?**	is Im(z)=0?	is Im(z)=0?	z in ***Z*** (XY)	YES/NO (skips if false)	
18	**ZIN?**	Is z inside the unit circle?	is |z|<1?	z in ***Z*** (XY)	YES/NO (skips if false)	
19	**ZINT**	Makes Re(z) and Im(z) integers	Re(z) = Int[Re(z)]; Im(z)=Int[Im(z)]	z in ***Z*** (XY)	Result in ***Z*** (XY)	Does LastZ
20	**ZMTV _**	Multi-functions Launcher	n/a	Prompts for choice	Executes function	FOCAL
21	**ZNXT _**	NEXT function Launcher	n/a	Prompts for Choice	Executes functions	
22	**ZPI**	Pi as a complex number	Zpi = pi + j0	none	Pi in Z(XY)	*Lifts Buffer*
23	**ZPRT _**	Poly-roots functions Launcher	n/a	Prompts for Choice	Executes function	
24	**ZREAL?**	Is Re(z)=0?	Is Re(z)=0?	z in ***Z*** (XY)	YES/NO (skips if false)	
25	**ZQUAD**	Shows quadrant for z	Quad# as function of location	z in ***Z*** (XY)	Sets corresponding user flag 1-4	Clears other flags 1-4
26	**ZSIGN**	Complex SIGN	sign = z/|z|	z in ***Z*** (XY)	z/Mod(z) in X,Y	*does LastZ*
27	**ZTONE**	Makes a sound	Frequency and duration	z in ***Z*** (XY)	Makes sound	*Shows Z at end*
28	**ZTRP**	Exchanges Re(Z) and Im(Z)	zTrp = y + iX	z in ***Z*** (XY)	Im(z) in X, Re(z) in Y	*does LastZ*
29	**-DELUXE**	***Section Header***	n/a	n/a	n/a	
30	**ZAMN**	Complex Arithmetic Mean	AM = Σ zk / n	Control word bbb.eee in X	Result in ***Z*** (XY)	*Data expected in ZRegs*
31	**ZANGJ**	Anger J(n,z) Function	See manual	z in(ZY), n in X	Result in ***Z*** (XY)	
32	**ZCRF**	Carlson Integral 1st. kind	See manual	n,n,p,z in stack	Result in X	Complex conjugate
33	**ZCRJ**	Carlson Integral 3rd. kind	See manual	n,n,p,z in stack	Result in X	Complex conjugate
34	**ZCSX**	Fresnel Integrals C(x) & S(x)	See manual	X in X	S(x) in Y, C(x) in X	*FOCAL*
35	**ZELIP1**	Incomplete Elliptic integral 1st kind	Complex amplitude, real modulus	a in (Y,Z), m in X	Result in ***Z*** (XY)	*FOCAL*
36	**ZELIP2**	Incomplete Elliptic Integral 2nd kind	Complex amplitude, real modulus	a in (Y,Z), m in X	Result in ***Z*** (XY)	*FOCAL*
37	**ZELIPE**	Complete Elliptic Integral 2nd kind	Uses Hypergeometric functions	Complex m in ***Z*** (XY)	Result in ***Z*** (XY)	*Requires |z|<1*
38	**ZELIPK**	Complete Elliptic Integral 1st kind	Uses Hypergeometric functions	Complex m in ***Z*** (XY)	Result in ***Z*** (XY)	*Requires |z|<1*
39	**ZELK**	Complete Elliptic Integral 1st kind	Uses AGM	Complex m in ***Z*** (XY))	Result in ***Z*** (XY)	*FOCAL*
40	**ZELPKE**	Comlete Elliptic Intg. 1st & 2nd kinds	Uses AGM and AGM2	Complex m in ***Z*** (XY)	Results in ***W*** and ***Z*** (XY)	*FOCAL*
41	**ZGHM**	Geometric-Harmonic Mean	GHM	w in ***W***, z in ***Z*** (XY)	Result in ***Z*** (XY)	*does LastZ*
42	**ZGMN**	Complex Geometric mean	GM = [Π Zk]^1/k	Control word bbb.eee in X	Geometric mean in ***Z*** (XY)	*Data expected in ZRegs*

#	Function	Description	Formula	Input	Output	Comments
43	**ZHMN**	Complex Harmonic Mean	HM = Σ 1 / [1/zk]	Control word bbb.eee in X	Harmonic mean in ***Z*** (XY)	*Data expected in ZRegs*
44	**ZINPT**	Enters complex data in ZRegs	n/a	Control word bbb.eee in X	Data is stored sequentially	*FOCAL*
45	**ZKLV1**	Kelvin Functions 1st kind	Uses Hypergeometric Function	x in X	bei(x) in Y, ber(x) in X	*FOCAL*
46	**ZOUPT**	Shows complex data	n/a	Control word bbb.eee in X	Data is shown sequentially	*SF 21 to stop each value*
47	**ZPD1**	Complex Polynomial 1st derivative	P'(z) = Σ k ak z^*k-1 |k=1,2 .. n	z0 in (Y,Y) ; bbb.eee in X	dP(z)/dz in "Z" (XY)	*does Lastz*
48	**ZPD2**	Complex Polynomial 2nd derivative	P"(z) = Σ k (k-1) ak z^k-2 | k=2,3..n	z0 in (Y,Y) ; bbb.eee in X	d2P(z)/dz2 in "Z" (XY)	*does Lastz*
49	**ZPLI**	Complex Polynomial Primitive	IT[P(z)] = Σ ak z^ k+1 / (k+1) | k=0,1..n	z0 in (Y,Y) ; bbb.eee in X	Result in ***Z*** (XY)	*does LastZ*
50	**ZPSIN**	Complex Poly-Gamma function	See manual	Z in (Y,X) ; n in X	Result in ***Z*** (XY)	
51	**ZSJB**	Complex Spherical Bessel J(w,z)	j(w,z) = sqr(π/2z) J(w+1/2, z)	w in ***W***, z in ***Z*** (XY)	Result in ***Z*** (XY)	
52	**ZSYB**	Complex Spherical Bessel Y(w,z)	y(w,z) = sqr(π/2z) Y(w+1/2, z)	w in ***W***, z in ***Z*** (XY)	Result in ***Z*** (XY)	
53	**ZWEBE**	Weber function E(n,z)	See manual	z in (Y,X) ; n in X	Result in ***Z*** (XY)	
54	**CAT+ _**	Sub-function Catalog	n/a	none	Sequential Enumeration	*XEQ executes function*
55	**(c)**	Copyright Message	n/a	none	Shows copyright in ALPHA	*"(c) AMC 2016"*
56	**LASTF**	Executes last function	n/a	Previous call by launcher	Re-executes function	*Includes sub-functions*

1	**-ZBUFFER**	***Section Header***	n/a	None	None	
2	**CLZB**	Clears Z buffer	n/a	None	buffler cleared	
3	**L1=XY?**	is L1 equal to XY?	n/a	None	Y/N, skip if false	
4	**L1<>L _ _**	Swap L1 & Level	*n/a*	Level# as suffix	levels exchanged	*Prompting*
5	**L1<>L2**	Swap L1 & L2	n/a	None	levels exchanged	
6	**L1<>L3**	Swap L1 & L3	n/a	None	levels exchanged	
7	**L1<>L4**	Swap L1 & L4	n/a	None	levels exchanged	
8	**L1<>LX**	Swap L1 & Level	n/a	level in X	levels exchanged	
9	**L2=ZT?**	is L2 equal to ZT?	n/a	None	Y/N, skip if false	
10	**L2>ZT**	Copies L2 into ZT	n/a	None	L2 copied to ZT	
11	**LVIEW _**	View Level	n/a	Level# as suffix	*Transposed value!*	*Prompting*
12	**LVIEWX**	View level by X	n/a	level in X	*Transposed value!*	
13	**PREMON**	Copies XY into L0 and finds Zbuffer	n/a	Re(z) in X; Im(z) in Y	none	
14	**PSTMON**	Copies XY into L1 and synch's up	n/a	Re(z) in X; Im(z) in Y	None	
15	**RG>ZB _ _**	Copies registers to Z buffer	n/a	Reg# as suffix	data copied from registers	*Prompting*
16	**ST>ZB**	Copies real stack to L1 & L2	n/a	None	stack copied to buffer	
17	**XY>L _**	Copies XY into Level	n/a	Level# as suffix	XY copied to LEVEL	*Prompting*
18	**XY>L0**	Copies XY into L0	n/a	Re(z) in X; Im(z) in Y	XY copied to L0	
19	**XY>L1**	Copies XY into L1	n/a	Re(z) in X; Im(z) in Y	XY copied to L1	
20	**ZB>RG _ _**	copies buffer to registers	n/a	Reg# as suffix	data copied to registers	*Prompting*
21	**ZB>ST**	Copies L1 & L2 into real stack	n/a	None	buffer copied to Stack	
22	**ZBDROP**	Drops Z buffer one level	n/a	None	levels dropped	Drops Buffer
23	**ZBHEAD**	Zbuffer Header info	n/a	None	header register in ALPHA	
24	**ZBLIFT**	Lifts Z buffer one level	n/a	None	buffer lifted	*Lifts Buffer*
25	***ZBVIEW***	Shows Z Buffer	n/a	None	shows header & all levels	FOCAL
26	**-B UTILS**	***Section Header***	n/a	None	None	
27	**B?**	Does buffer exist?	n/a	buffer id# in X	YES/NO (skips if false)	CCD Module
28	**BLIST**	lists all buffers existing	n/a	none	list in Alpha	D. Yerka
29	**BLNG?**	Buffer length	n/a	buffer id# in X	buffer size in X	CCD Module
30	**BX>RG**	copies buffer to registers	n/a	buffer id# in X	data copied into R00 to end	David Assm
31	**CLB**	Clear buffer	n/a	buffer id# in X	Clears buffer from memory	CCD Module
32	**FINDBX**	finds buffer address	n/a	buffer id# in X	buffer address in X	D. Yerka
33	**MAKEBX**	makes buffer in RAM	n/a	(id#,size) in X	buffer created	D. Yerka
34	**RG>BX**	copies registers to buffer	n/a	Data in R00 to Rnn	Copied to Buffer	David Assm

(*) Buffer functions have been moved to the BUFFERLAND Module, under a dedicated section for the 41Z case.

Appendix 5.- Buffer logic function table.

			Pre-Exec						Post-Exec				
			Alpha in XY	XY to L0	XY to L1	Buffer LIFT	L2 -> ZT		Buffer DROP	XY into L1	L1,2 -> XYZT	ZAVIEW	
1	- HP-41 Z	Initialize Buffer	yes	no	yes	no	no		no	no	no	yes	
2	W^Z	Power	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	yes	POSTDUAL
3	Z+	Addition	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	yes	POSTDUAL
4	Z-	Substraction	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	yes	POSTDUAL
5	Z*	Multiply	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	yes	POSTDUAL
6	Z/	Divide	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	yes	POSTDUAL
7	ZWANG	Angle between	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	no	PSTDUAL-2
8	ZWCROSS	Cross Product	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	no	PSTDUAL-2
9	ZWDET	Determinat	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	no	PSTDUAL-2
10	ZWDIST	Distance	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	no	PSTDUAL-2
11	ZWDOT	Dot Product	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	no	PSTDUAL-2
12	ZWLINE	Line Equation	yes	yes	no	no	yes	PREDUAL	yes	yes	yes	no	PSTDUAL-2
13	Z=W?	is Z=W?	yes	no	yes	no	yes	PREDUL-2	no	no	no	no	
14	Z=WR?	is Z=W round?	yes	no	yes	no	yes	PREDUL-2	no	no	no	no	
15	Z#W?	is Z not W?	yes	no	yes	no	yes	PREDUL-2	no	no	no	no	
16	Z=0?	is Z Zero?	yes	no	yes	no	no	PREMON-2	no	no	no	no	
17	Z#0?	is Z not zero?	yes	no	yes	no	no	PREMON-2	no	no	no	no	
18	Z=I?	is Z = i?	yes	no	yes	no	no	PREMON-2	no	no	no	no	
19	ZREAL?	Is Z real?	yes	no	yes	no	no	PREMON-2	no	no	no	no	
20	ZIMAG?	Is Z imag?	yes	no	yes	no	no	PREMON-2	no	no	no	no	
21	ZIN?	|Z|<1?	yes	no	yes	no	no	PREMON-2	no	no	no	no	
22	ZOUT?	|Z|>1?	yes	no	yes	no	no	PREMON-2	no	no	no	no	
23	ZUNIT?	|Z|=1?	yes	no	yes	no	no	PREMON-2	no	no	no	no	
24	X^Z	Hybrid Power	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
25	Z^2	Z^2	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
26	Z^X	Z^X	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
27	ZACOS	ACOS	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
28	ZACOSH	ACOSH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
29	ZALOG	10^Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
30	ZASIN	ASIN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON

31	ZASINH	ASINH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
32	ZATAN	ATAN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
33	ZATANH	ATANH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
34	ZCONJ	X-Yj	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
35	ZCOS	COS	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
36	ZCOSH	COSH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
37	ZDBL	2*Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
38	ZEXP	E^Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
39	ZHALF	Z/2	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
40	ZINV	1/Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
41	ZLN	Ln(Z)	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
42	ZINT		yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
43	ZFRC		yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
44	ZLOG	Log(Z)	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
45	ZNEG	-Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
46	ZRND	rounded Z	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
47	ZSIGN	Sign(Z)	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
48	ZSIN	SIN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
49	ZSINH	SINH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
50	ZSQRT	Square Root	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
51	ZTAN	TAN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
52	ZTANH	TANH	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
53	ZTRP	Re<>Im	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
54	ZARG	Zarg	yes	yes	no	no	no	PREMON	no	yes	yes	***no***	***PSTMON-2***
55	ZMOD	**|Z|**	yes	yes	no	no	no	PREMON	no	yes	yes	***no***	***PSTMON-2***
56	ZNORM	|Z|^2	yes	yes	no	no	no	PREMON	no	yes	yes	***no***	***PSTMON-2***
57	ZREC	Rectangular	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
58	ZPOL	Polar Notation	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
59	e^Z	alternate ZEXP	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
60	EIZ/IZ	function	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
61	Z^1/X	hybrid power	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
62	Z*I	rotation	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
63	Z/I	rotation	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
64	NXTASN	Next ASIN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
65	NXTACS	Next ACOS	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
66	NXTATN	Next ATAN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON

67	NXTLOG	Next LN	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
68	NXTNRT	Next Nth. Root	yes	yes	no	no	no	PREMON	no	yes	yes	yes	POSTMON
69	ZAVIEW	Output Z	yes	no	no	no	no		no	no	no	yes	
70	CLZ	Clear Z	no	no	no	no	no		no	yes	yes	yes	POSTMON
71	ZIMAG	Clear Re(z)	no	yes	no	no	no		no	yes	yes	yes	POSTMON
72	ZREAL	Clear Im(z)	no	yes	no	no	no		no	yes	yes	yes	POSTMON
73	CLZST	Clear Zstack	no	no	no	no	no		no	no	yes	yes	PSTMON-3
74	Z<>	Exchange	yes	no	no	no	no	PREMON	no	yes	yes	yes	POSTMON
75	Z<>W	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3
76	Z<>R	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3
77	Z<>S	Exchange Stack	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3
78	LASTZ	last argument	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	PSTMON-3
79	ZR^	Roll Up Zstack	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	PSTMON-3
80	ZRCL	Recall to Z	yes	no	yes	yes	no	PREMON-2	no	yes	yes	yes	POSTMON
81	IMAGINE	inputs Im(z)	yes	no	yes	yes	no	PREMON-2	no	yes	yes	yes	POSTMON
82	ZENTER^	Enter level	yes	no	yes	yes	no	PREMON-2	no	no	yes	yes	PSTMON-3
83	ZREAL^	Input number	yes	no	no	yes	no	PREMON	no	yes	yes	yes	POSTMON
84	ZIMAG^	Input number	yes	no	no	yes	no	PREMON	no	yes	yes	yes	POSTMON
85	ZRDN	Roll Down ZSTK	yes	no	yes	no	no	PREMON-2	yes	no	yes	yes	PSTMON-3
86	ZREPL	Replicates Z	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3
87	ZSTO	Stores Z	yes	no	yes	no	no	PREMON-2	no	no	yes	yes	PSTMON-3