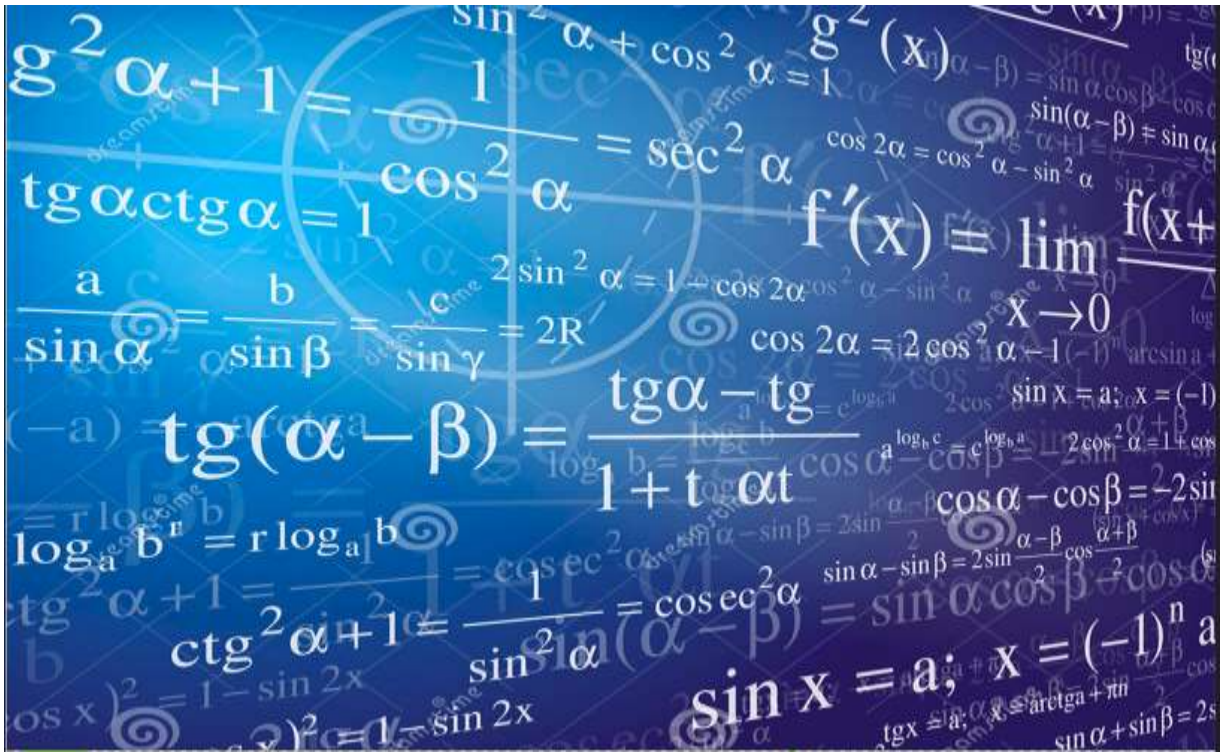


EQUATION SOLVER ROM

HP-41 Module



```

:R: :B: :C: :D: :E:
  USER      1

```

```

SELF-PR Y/N
  USER      1      PRGM

```

```

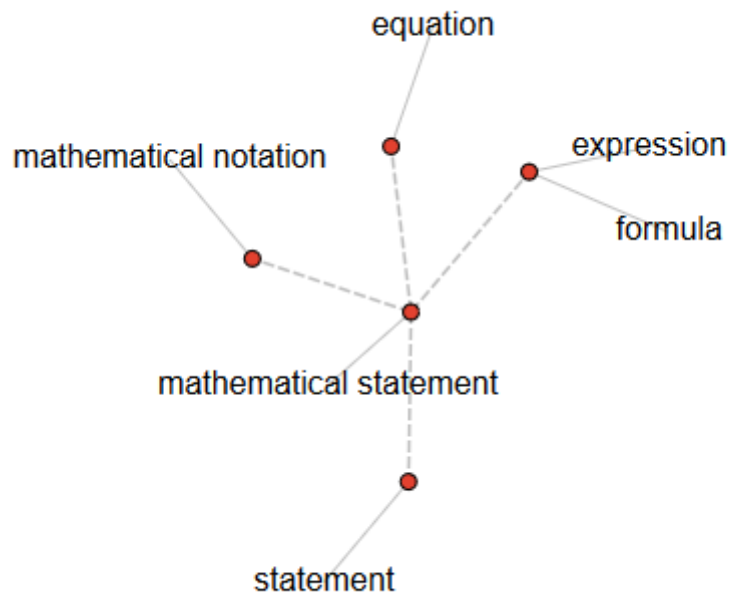
NO MVAR5
  USER      1

```

Written & Programmed by Ángel Martín, Mark Fleming & Greg McClure
Revision 3-AB, March 2020

This compilation revision 1.3.1

Copyright © 2018-2020 Ángel Martín



Published under the GNU software license agreement.

Original authors retain all copyrights and should be mentioned in writing by any part utilizing this material. No commercial usage of any kind is allowed.

Front cover image taken from: <https://www.dreamstime.com/royalty-free-stock-photography-mathematics-background-image20849947>

Thanks to Greg McClure and Mark Fleming for their contributions, suggestions for improvement and revisions to the manuals.

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

EQUATION_SOLVER -3A8

HP-41 Module

Table of Contents

1. Introduction	
a. Module function Summary	4
b. From SOLVE to Solver	6
c. Scope, Intent and Dependencies	6
2. Theory of Operation	
a. Variable Declaration	8
b. Program Editing vs. Running Modes	9
c. Building the Solver Program	10
d. Solving and Resolving	11
e. Tricks & Treats	13
f. A Look under the Hood	16
g. Mini Equation Library & Examples	18
3. Equation Libraries	
a. New Record Pointer Functions	23
b. A new twist to an old Solver	26
c. Show me the Money	27
d. Mark Fleming's Equation Library	28
 Appendix. AOS Simulator	 40

Equation Solver ROM – Function Summary

The table below lists all functions available in the module. All of them are programmable and directly accessible by the user, as it'll be explained in the sections that follow. The EVAL_EQNS section is an update to the work previously done by Mark Fleming and Greg McClure, with a few new functions added for convenience sake.

#	Name	Description	Input	Author
00	-SOLVER 1AB	Section header	n/a	n/a
01	A-PM7	ALPHA to Program (7 Chars)	Test in ALPHA	Ángel Martin
02	CLB7	Clear Buffer #7	none	Ángel Martin
03	CLVARS	Clear Variables	Data in buffer	Ángel Martin
04	DEDUP	De-duplicate String	String in ALPHA	Ángel Martin
05	DOSELF _	Self-Programming	Number of blocks	Ángel Martin
06	DOSLF+ _	Self-Programming+	Number of blocks	Ángel Martin
07	LCDV	LCD Variables	Data in Buffer	Ángel Martin
08	LCDV+ _	LCD Variables+	Data in Buffer	Ángel Martin
09	LKAOFF	Suspend Local Keys	Key Assignments	Ángel Martin
10	LKAON	Resume Local Keys	Key Assignments	Ángel Martin
11	MPREP	Menu Preparation	none	Ángel Martin
12	MUTE _ _	Mute Variable	ASCII char in prompt	Ángel Martin
13	MVARS _ _ _ _ _	Declare Variables	Prompts for letters	Ángel Martin
14	MVRS+ _ _ _ _ _	Declare Variables+	Prompts for letters	Ángel Martin
15	SHOW	Show text in LCD	Text in program line	Doug Wilder
16	SOLVER	Solve for Unknown	Data in program	Ángel Martin
17	SOLVR+	Solve for Unknown+	Data in program	Ángel Martin
18	UNMUTE	Undoes muted string	ASCII char in prompts0	Ángel Martin
19	VMENU	View Menu	Vars in Buffer	Ángel Martin
20	VMNU+	View Menu+	Vars in Buffer	Ángel Martin
21	Z=T?	Test for equal values	Values in Z,T	Ángel Martin
22	-EVAL\$ EQNS	Section header	n/a	n/a
23	ADVREC	Advance Record N Pos.	FileName in ALPHA, N in X	Ángel Martin
24	ARCLCHR	ARCL Character	FileName in ALPHA	Håkan Thörngren
25	READREC	Read Record to ALPHA	Data in Record	Ángel Martin
26	REC-	Move record one down	Pointer position	Ángel Martin
27	REC+	Move record one up	Pointer position	Ángel Martin
28	SEEK*	Seek record by X	FName in ALPHA, n in X	Ángel Martin
29	"APP\$"	Append Equation	To file "EQNS"	Mark Fleming
30	"APPEQN"	Append Equation	To file in ALPHA	Mark Fleming
31	"DELEQN"	Delete Equation	Removes four records	Mark Fleming
32	"EQNLIB"	Equation Library	Main Driver Program	Fleming - Martin
33	"INITEQN"	Initialize Library	Creates EQNS File	Mark Fleming
34	"SAR"	Search & Replace	Prompts for values	Mark Fleming
35	"#"	Auxiliary function	Data in program	Ángel Martin
36	"SV\$+"	Solves for X	Equation in ALPHA	Martin-McClure
37	"a^b?"	Prompts for guesses	n/a	Ángel Martin
38	LASTb	Recall Last buffer reg#	As saved by GET/LET	Ángel Martin
39	"APP\$"	Custom Eq. Library	Filename in ALPHA	Mark Fleming

The following section in next page includes individual examples of equations:

40	-EQ\$LIB	Section Header	n/a	n/a
41	3PMT _ _ _	Triple prompt	Hex Values in prompt	Ángel Martin
42	/+ /	Sums of Inverses	Values in prompt	Ángel Martin
43	SIGMD	Sigmoid Function	Argument in X	Ángel Martin
44	"3DM"	3D Vector Module	" :X: :Y: :Z: :M: "	Ángel Martin
45	"CTRY"	Catenary Curve	" :A: :H: :L: :D: "	Ángel Martin
46	"HTX"	Heat Exchanger	" :1: :2: :I: :O: :Q: "	Ángel Martin
47	"KPL"	Kepler Equation	" :M: :E: :C: "	Ángel Martin
48	"LMOV"	Linear Movement	" :X: :V: :A: :T: "	Ángel Martin
49	"RdK"	Redlich-Kwong EOS	" :P: :V: :T: :A: :B: "	Ángel Martin
50	"RGA\$"	Real Gas EOS	" :P: :V: :Z: :N: :T: "	Ángel Martin
51	"TVM\$"	TVM equation	Prompts for inputs	Martin-McClure
52	"VdW"	Van-der-Waals EOS	" :P: :V: :T: :A: :B: "	Ángel Martin
53	"Y=P1"	Straight Line Eq.	" :A: :B: :X: :Y: "	Ángel Martin
54	"Y=P2"	Quadratic Equation	" :A: :B: :C: :X: :Y: "	Ángel Martin
55	"Y=P3"	Cubic Equation	" :A: :B: :C: :X: :Y: "	Ángel Martin
56	"Y=P4"	Quartic Equation	" :A: :B: :C: :D: :E: "	Ángel Martin

This module also contains Mark Fleming's Equation Library, with the following equations included:

00	LINEAR	24	OHMS LAW
01	$Y=AX+B$	25	$E=IR$
02	$c*a+d-b$	26	$b*c-a$
03	X Y A B	27	E I R
04	QUADRATIC	28	PARALLEL R
05	$Y=AX^2+BX+C$	29	$1/R1=1/R2+1/R3$
06	$c*a^2+d*a+e-b$	30	$1/b+1/c-1/a$
07	X Y A B C	31	R1 R2 R3
08	CUBIC	32	RLC FREQ.
09	$Y=X^3+AX^2+BX+C$	33	$F0=1/\text{SQRT}(LC)$
10	$a^3+c*a^2+d*a+e-b$	34	$1/Q(b*c)-a$
11	X Y A B C	35	F0 L C
12	4TH ORDER	36	GAS EQUATION
13	$D+X(C+X(B+X(A+X)))$	37	$PV=NRT$
14	$e+a*(d+a*(c+a*(b+a)))$	38	$c*(16629/2000)*d-a*b$
15	X? A B C D	39	P V N T
16	POSROOT	40	LIN. MOTION
17	$X1=(-B+\text{SQRT}(B^2-4AC))/2A$	41	$X=VT+1/2*AT^2$
18	$(\#b+Q(b^2-4*a*c))/2/a-d$	42	$c*b+1/2*d*b^2-a$
19	A B C X1	43	X T V A
20	NEGROOT	44	NEWTONS LAW3
21	$X2=(-B-\text{SQRT}(B^2-4AC))/2A$	45	$F=G*M1*M2/R^2$
22	$(\#b-Q(b^2-4*a*c))/2/a-d$	46	$e*b*c/d^2-a$
23	A B C X2	47	F M1 M2 R G

(*) Note that due to space constraints the "Interest" and "TVM" equations are not included in this version. You can add those manually to the EQNS ASCII file using the information in section 3.d, page #31.

Equation Solver ROM

Revision 3-AB - HP-41 Module

Introduction. From SOLVE to SOLVER.

Welcome to the Equation Solver ROM, the logical next step that extends the Formula Evaluation Module and expands on its capabilities by providing a full-fledged Equation Solver.

Perhaps the last remaining open subject to address on the HP-41 platform, Equation Solvers have become a standard fixture since the HP-42S days, which had the first soft-keys, SOLVE-based implementation on HP calculators. Much has happened since, and successive generations have refined the initial concept in different aspects as new functionality was being added to their operating systems. (see: <https://support.hp.com/us-en/document/c01822098>)

As you can guess, the implementation on this ROM follows the same approach present on the HP-42, relying on the local labels and the data entry flag. Chances are you're already familiar with it so it should be relatively simple to grasp -but this module adds an interesting twist by utilizing formula expressions directly, using the functionality from the Formula Evaluation Module.

Even if it's not strictly required to be proficient on the Formula Evaluation functionality, knowing your way around that module will facilitate using the Equation Solvers. You're therefore encouraged to read the Formula Evaluation ROM manual for a deeper understanding on the underpinnings of this module. You'll need to write the main equation to solve following the conventions from the Formula Evaluation manual, and for that you'll need to follow the syntax and other operation rules explained there in detail.

Scope, Intent and Dependencies

There are two sets of SOLVER functions in this module, *the standard set* that handles up to five variables; and *the extended set* – allowing up to six variables in the equations. Regarding the SOLVE capabilities, each of them may use a direct **SV\$+** algorithm based on the secant-method, or a more sophisticated one based on **FROOT**, featuring a combination of Newton and Secant methods. The former is sufficient in most cases for Science & Engineering equations, but both methods are at your disposal to use them as you see fit. The latter requires that the **"Solve & Integrate"** ROM ("SIHP" for the CL, with -SOLINTG 2D CAT'2 header) be plugged in the calculator as well. This ROM offers the same solving functionality also found in the SandMath's **FROOT**, which in turn is the same one originally from the HP41 Advantage's **SOLVE**.

Note that in both cases *the equation is not programmed using the standard FOCAL language*, but as an ALPHA string that is later interpreted by the **EVAL\$** functions from the Formula Evaluation ROM. This ALPHA string is the basis of the SOLVER operation, as it facilitates the selection of the appropriate variable to solve for in a dynamic and automated way.

This module requires the Formula_Evaluation module, revision 2H.

As for other dependencies, this module is a **Library#4-aware ROM** that requires the library#4 (revision **R47** or higher) to be plugged in. Also, the ROM is only compatible with the CX OS, as internal routines from it are used.

Theory of Operation.

As hinted at in the introduction section, the Equation Solver operation is based on a dynamic and automated selection of the variable to solve for, as defined in a user program (FOCAL) that includes the general equation inter-relating multiple variables. Regardless of how many variables make out the general equation, five or six of them (depending on SOLVER set used) can be included in the SOLVER operation.

The elements of the FOCAL program are as follows:

- The user first writes said general main equation as an alphabetical expression, using the conventions defined by the Formula Evaluation functions. This expression may have a combination of variables, parameters and constants linked by operations and syntax rules. You can use the **^FRMLA** function in the Formula Evaluation ROM to enter the expression, or you may also do it directly typing the equation in ALPHA if you're comfortable using special characters (not part of the standard ALPHA keyboard but accessible using the AMC_OS/X module)
- Next, the Solver Variables need to be declared – i.e. a subset of the variables and parameters included in the alphabetical expression are defined as potential knowns/unknowns. This definition becomes pivotal in the structure of the user program used to enter the known values and to trigger the calculation of the unknown ones. It is made with the **MVARS** function, which must be located right after the general equation step – with no other program lines in between.
- This is to be picked-up by the second part of the Solver, which is always executed in every action – either to assign a value to a known variable, or to trigger the solving of the unknown. As this requirement implies, each menu option needs to call the **SOLVER** function and act accordingly depending on the local label it is located under, and whether the data entry user flag (UF 22) is set.
- The FOCAL program must have a local label associated to each variable declared. This local label will be accessed by pressing the Top Keys in the calculator ({A-E} and also [F] in the extended solver case). The action performed will depend on whether a value is entered before pressing the soft-key (meaning the value is assigned to that variable) or if it's directly pressed (meaning the value will be calculated (solved for) using the main equation).

The functions provided in the module are used for the definition of variables, creation of the FOCAL program and user operation of the solvers. They offer automation and convenient data input features that make most of the underlying details, all transparent to the user.

Note.- To differentiate the two Solver sets, the names of the functions use the following convention: Extended set function names end with the plus sign "+", whereas Standard set functions don't.

Declaration of Variables. { **MVARS** , **MVRS+** }

The first step to define the SOLVER consists of telling the calculator which of the variables written in the general equation will be used. This is accomplished by entering the variable names at the prompt offered by the MVARS/MVRS+ functions, *using only one letter per variable*.



The available choices depend on the solver set, as follows:

- Any letter { A to Z } can be used in the declaration for the standard set – including numbers 0-9 using the SHIFTed keys.
- Only letters { A to F, and X, Y, Z, T, L } can be used in the declaration for the extended set – but even if allowed, you should not use X, Y, T, L because these are used as scratch by the solving routines. Refer to the block diagram in next page for an overview of the hierarchical relationships amongst the sections involved in the complete process.

So right now, you see that the extended set restricts the variable names, even if it offers the possibility to use one extra variable in the Solver. This is a compromise needed to maintain the code size and buffer resources within reasonable specs, the overarching design criteria that always applies in MCODE programming.

Here's how the functions work:

- The user can enter fewer variables than the length of the prompt field – pressing R/S or the radix key at any time will terminate the variable declaration step – and only the letters already filled in will be used in the menu choices. Terminating them without any letter entered will show the "NO MVARS" error message.
- The functions will automatically de-duplicate possible repeat entries, making only one menu item per given letter.
- For the standard set the variables will be presented in the menu in the same order as they are entered in the prompts. The user needs to bear this important fact in mind, as the variable names in the general equation need to be mapped to the menu letters *by position*, i.e. using the input order: variable "a" for the first entered letter, variable "b" for the second, etc.
- For the extended set they will be sorted alphabetically. This facilitates the mapping of their letters to the variables *by name irrespective of the local label they're input from*. Only when all six of them are to be used there's a direct name-to-label correspondence: Letter [A] maps to variable "a", letter [B] maps to variable "b", etc. In principle all 10 letters are accepted but note the additional restriction on which variables are available to the solver later on.

Program Editing vs. Running modes

Both **MVARS** and **MVRS+** have very different behavior depending on when they're used, either during program editing or while running the program. During program editing they'll display the prompt fields as described above, for the user to declare the solver variables.

When the declaration completes (either filling all prompts or capping the entry using R/S or Radix), the function will store the menu letters in the header of buffer #7, from where they will be picked up by the other functions, and it will insert two lines in the current program: one for itself (to be executed when the program runs), followed by a text line with the selected variable letters.

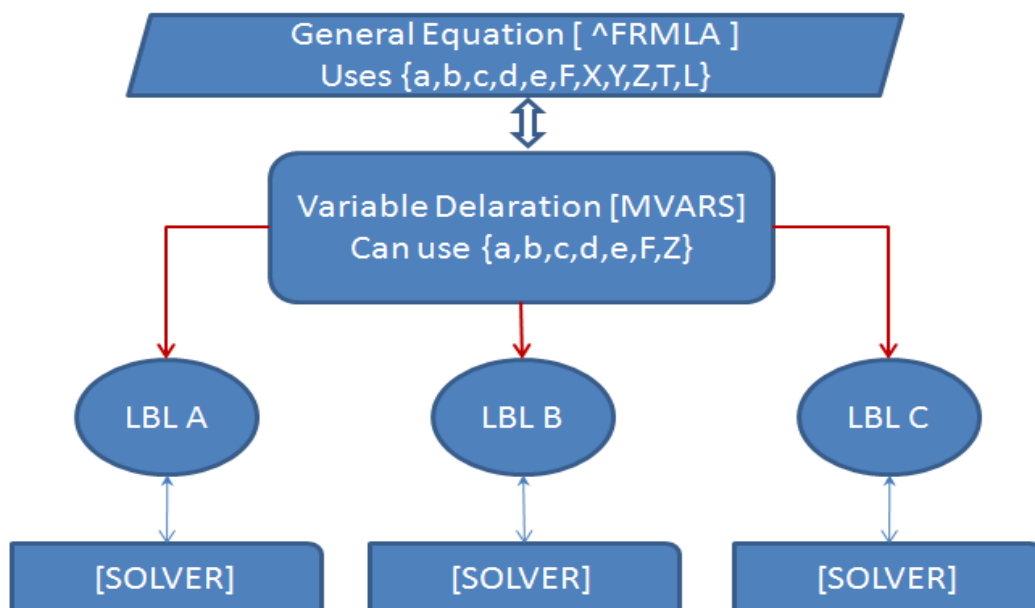
For example, **MVARS** plus "YZFC" will create the two program steps at the current location:

```
nn      MVARS
nn+1    "YZFC"
```

A word on writing the General Equation.

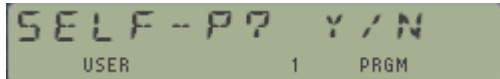
As you should know by now the variables available to the **^FRMLA** writing are the five stack registers and the six buffer registers, i.e. {X, Y, Z, T, L} plus {a, b, c, d, e, F}. Not all of these can be freely used in your general equation because the Solvers need the stack registers X, Y, T and L for scratch during the evaluation of the functions. This leaves us with the six buffer registers plus register Z available for the equation. This is further restricted to just the buffer registers in the 5-Vars case, mapped by the position in the MVARS string.

You can use just as many as known/unknown variables in your equation, but you can also use the others to hold parameters or other constants - this saves characters in the formula. Use the function **LET=** to assign the parameter values as needed.



Building the Solver Program.

Both components of the Solver need to play their roles, therefore **MVARS** will now offer the user the possibility to auto-create the rest of the FOCAL program needed for the Solver to work – by adding automatically the needed local labels (as many as filled out fields in the prompt), the matching **SOLVER** statements and auxiliary steps required to accommodate the menu letters declared.



Answering "N" will terminate this stage without adding the lines (the user will need to do it later manually!), whilst answering "Y" will proceed inserting the additional lines required for the correct use of the Solver.

The rule here is that each menu letter will need one local label, followed by the **SOLVER** function, plus a STOP instruction to halt the execution and continue entering values. For instance, using the same example with four menu letters declared it'll insert the following 12 program steps:

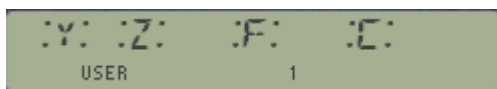
nn+2	<u>LBL A</u>	nn+6	SOLVER	nn+10	STOP
nn+3	SOLVER	nn+7	STOP	nn+11	<u>LBL D</u>
nn+4	STOP	nn+8	<u>LBL C</u>	nn+12	SOLVER
nn+5	<u>LBL B</u>	nn+9	SOLVER	nn+13	STOP

Obviously **MVRS+** will insert **SOLVR+** instructions instead, as these two always need to be paired up. The baton is passed to the appropriate counterpart!

Note that the local label letters are completely unrelated to the menu letter – except in the sequence order entered at the prompts. Which also determines the mapping to the EVAL\$ variables as follows:

Menu Letter "Y" -> EVAL\$ var "a" ; LBL A	Menu Letter "F" -> EVAL\$ var "c" ; LBL C
Menu Letter "Z" -> EVAL\$ var "b" ; LBL B	Menu Letter "C" -> EVAL\$ var "d" ; LBL D

This will be presented in the display as follows when the MVARS function is executed during the running program:



This FOCAL "skeleton" may well be all you need to proceed, in which case all you need to do is add an END statement (or GTO ..) to complete the FOCAL program – just make sure it has a global label, and *don't forget to define the general equation before the MVARS step*

You're of course free to edit the FOCAL program further, adding any other instruction needed that you see fit (say angular modes for trigonometry, etc.) – but you mustn't alter the "skeleton" written by **MVARS**. The function **SOLVER** in particular *must always be right after the local label*, as this condition is expected and used to determine its actual location.

Solving and Resolving. {**SOLVER** , **SOLVR+** }

Once we've come to this point it's time to hand it out to the actual SOLVE engine. The first thing to say is that the expression of the equation follows the $f(x) = 0$ form, where just $f(x)$ is programmed as the general equation.

The Solver allows for two approaches, the **SV\$+** way (using the secant method) and the **FROOT** way (using a combination of Newton and secant methods depending on the cases). The former uses the built-in routine **SV\$+** dedicated to this purpose. For the latter you need to plug in the "Solve & Integrate" ROM that provides the **FROOT** function.

A few considerations on the secant method: - It is defined by the recurrence relation for the successive iterations of the root:

$$x_n = x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

As can be seen from the recurrence relation, the secant method requires two initial values, x_0 and x_1 , which should ideally be chosen to lie close to the root. The iterates x_n , of the secant method converge to a root of **$f(x)$** , if the initial values x_0 and x_1 are sufficiently close to the root. Obviously, this requires that x_0 and x_1 cannot be equal, and furthermore even if they are different it also imposes an additional condition to avoid dividing by zero: $f(x_0)$ must be different from $f(x_1)$.

These limitations can tip the scale and render the method inadequate for some more finicky equations – making the **FROOT** option better suited to the task. It employs a combination of the Newton and secant methods, depending on the function's behavior in the vicinity of the guesses supplied by the user.

The method starts with a function **$f(x)$** defined over the real numbers x , the function's derivative **f'** , and an initial guess x_0 for a root of the function **f** . If the function satisfies the assumptions made in the derivation of the formula and the initial guess is close, then a better approximation x_1 is:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad ; \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

The process is repeated until a sufficiently accurate value is reached.

The Solver program always prompts for two guesses (a and b). If no values are entered the program will use the defaults as 0 and 1 – which surprisingly works just fine for many equations – even if the execution time may be longer than if more targeted initial values are used.



Just press R/S to accept the default values [0, 1] – or enter them as you see needed.

Examples. Prepare a Solver FOCAL program to handle the general equation: " $a + b + c + d = e$ "

Since there are only five variables involved, we're free to use either one of the two Solver set available. Let's do it for both for the sake of complete documentation.

First using the standard set. We'll label the menu items "J, K, L, M, and N"

In PRGM mode we insert a global label and the equation, followed by **MVARs** "JKLMN"– and we take advantage of the Self-programming option answering "Y" to the choice. We'll complete the task by removing the last STOP step (we won't use it this time) and typing GTO .. to add the END and pack the program memory area.

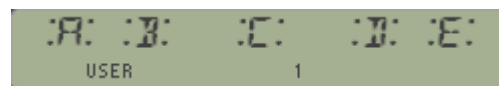
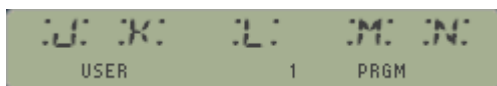
Next using the extended set. Naturally labeling the menu items "A, B, C, D, and E".

In PRGM mode we insert another global label, followed by **MVRS+** "ABCDE" – and again we take advantage of the Self-programming option. As before, we finish by typing GTO

See below the two programs created so far:

01	<u>LBL "STD"</u>		20	<u>LBL "XTD"</u>
02	"a+b+c+d-e"	↔	21	"a+b+c+d-e" <i>Same equation!</i>
03	MVARs		22	MVRS+
04	"JKLMN"		23	"ABCDE"
05	<u>LBL A</u>		24	<u>LBL A</u>
06	SOLVER		25	SOLVR+
07	STOP		26	STOP
08	<u>LBL B</u>		27	<u>LBL B</u>
09	SOLVER		28	SOLVR+
10	STOP		29	STOP
11	<u>LBL C</u>		30	<u>LBL C</u>
12	SOLVER		31	SOLVR+
13	STOP		32	STOP
14	<u>LBL D</u>		33	<u>LBL D</u>
15	SOLVER		34	SOLVR+
16	STOP		35	STOP
17	<u>LBL E</u>		36	<u>LBL E</u>
18	SOLVER		37	SOLVR+
19	END		38	END

It's all ready to go now: calling each of the programs will generate the following menu screens, standard solver on the left and extended solver on the right respectively:



Using J=1, K=2, L=3, M=4 => N= 10 ; Using A=1, B=1, C=1, D=1 => E= 5

The sequences being: 1, XEQ [A], 2, XEQ [B], 3, XEQ [C], 4, XEQ [D], XEQ [E]

And: 1, XEQ [A], 1, XEQ [B], 1, XEQ [C], 1, XEQ [D], XEQ [E]

Tricks and Treats.

As mentioned previously, you can choose the solving method employed by the programs, either the secant method in **SV\$+** or the Newton/Secant combination in **FROOT**. This is controlled by the status of User flag 00 when you press the "Solve for the Unknown" soft key:

- If UF 00 is Clear => Secant Method by **SV\$+**
- If UF 00 is Set => Newton/Secant combo by **FROOT**

Don't forget to plug the "Solve & Integrate" ROM for the second case.

Apart from that important consideration, the following observations should be borne in mind:

1. Using the Data Entry flag is a convenient way to distinguish between the value assignment and the call for solving the unknown, but it's not perfect. The most important limitation is that you need to enter actual numeric values for F22 to be set, not being enough with recalling them from a data register using **RCL nn**. Another scenario that frequently trips folks up is using **PI**, which doesn't activate the flag either. Therefore make sure you set it manually (SF 22) or force the condition with dummy operations like { 0, + }; or: { 1, * }
2. You can use the function **GET=** (in the Formula Evaluation) to recover the values currently stored in the variables. Be aware that – consistent with the RCL situation – here too such action won't set the Data Entry Flag (!)
3. Note that as of revision 2-AB of the module, after the solution for the unknown has been calculated it is automatically stored by the program in the variable mapped to the menu letter. This is handy to verify the obtained results, plugging it as a known and back-calculating some of the previously known variables.
4. The **SOLVER** functions will ignore pressing of local Labels if the corresponding letter hasn't been previously declared – even if you manually manage to add the local label yourself – or if it's a left-over placed there from previous executions or **MVARS** that used more variables.
5. The extended Solver (+) can use up to six variables, but their letters are limited to those of the buffer registers. Furthermore, the variable mapping is done by their name within the declaration string, irrespective of the location of the local labels. For instance, the string "BCF" is using the buffer registers b, c and F behind the scenes. As a corollary, when all six variables are used the sixth one will always be "F", and even if not shown in the display it'll be mapped to the local label [F] (i.e. the X<>Y key).
6. Perhaps the strongest limitation of this design – the general equation must fit in the ALPHA registers, i.e. it cannot exceed 24 characters. However, if your formula wants to go beyond that boundary *you can use a "chained" strings approach as an established workaround*. This consists of expressing the general equation as a combination of two, using functions f() and g(), so that the evaluation can be done in two stages. The syntax rule is that the secondary function g() must start with a dollar sign "\$" – this tells **SV\$+** that a primary function f() is also to be used, and therefore how to conform f(g()).

Summary recap.

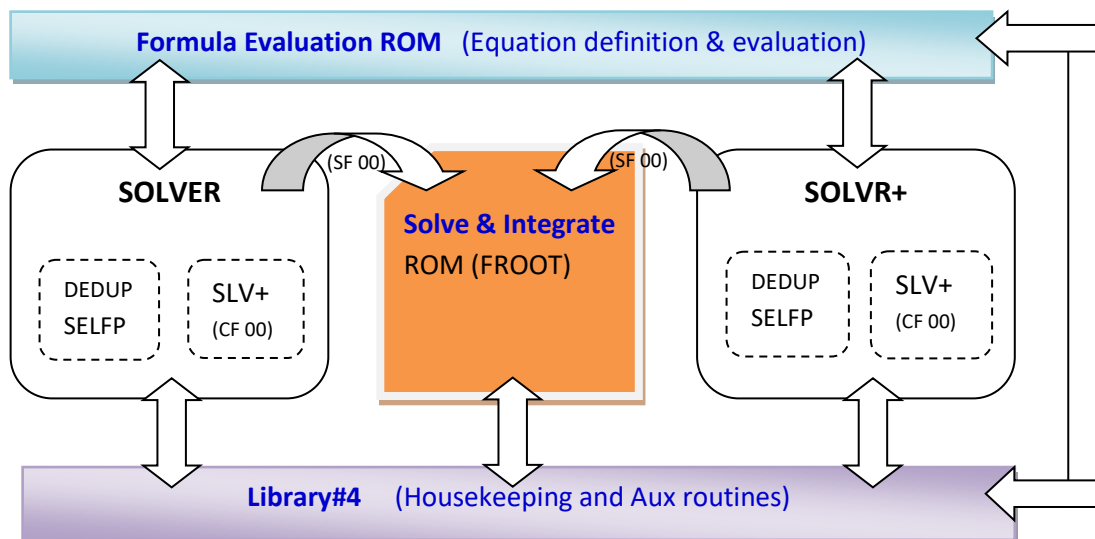
The table below shows the main attributes for both solvers in a comparative way:

Solver	# Vars	Named	Eq. Mapping	DEDUP?	Self-Prog?
SOLVER	5	Any letter	{a - f} By Position	Yes	Yes
SOLVR+	6	{A – F}	{a - f} By Name	Yes	Yes

The Solve technique choices and dependencies are shown below:

Solve Type	Trigger	Limits	Chained Eqs	Dependencies
SLV+	CF 00	a^b ; [0,1] default	Yes	LIB#4, Form_Eval
FROOT	SF 00	a^b ; [0,1] default	No	Lib#4, Form_Eval, SIROM

The conceptual diagram below shows the dependencies for each case:



Example. Write a FOCAL SOLVER routine for the Time Value of Money.

Equation: $PV + (1 + ip) PMT/i [1 - (1+i)^{-n}] + FV (1+i)^{-n} = 0$

Modified: $PV + FV \cdot (1+i)^{-n} + PMT \cdot (p + 1/i) \cdot [1 - (1+i)^{-n}] = 0$

Options: CF 02: End mode SF 02: Begin mode

We know that a full TVM equation is going to require more than 24 characters, therefore a chained strings technique is necessary. We have split in two as follows:

Secondary Equation, g(\$\$) – it is executed first, leaving the result in the T register

Primary equation, f() – it's executed last, leaving the result in the Z register. It uses "T" as one of the variables, picking up the calculated result by g().

In fact, the formula needed a bit of changes to make it fit within the 48-char restriction – but the end result is fully compliant and compatible with the solver design!

Here's the program listing; note the chain sign ("\$\$") in the secondary equation in step 09. Also note that the TVM variables "PIMNF" are listed in step11, and therefore mapped in that sequence to the buffer registers {a,b,c,d,e}. Finally, user flag 02 controls the BEGIN/END modes and needs to be set up manually by the user *before* executing TVM\$.

01 <u>LBL "TVM\$"</u>	14 STOP
02 E	15 <u>LBL B</u>
03 FS? 02 ; begin mode?	16 SOLVER
04 CLX	17 STOP
05 LET= 06 ; buffer "F"	18 <u>LBL C</u>
<i>06 "a+e*T+c*(1-T) "</i> ; primary	19 SOLVER
<i>07 "-*(F+100/b)"</i> ;con't	20 STOP
08 STO\$ 12 ; string stored	21 <u>LBL D</u>
09 <i>"\$(1+b/100)^#d"</i> ; secondary	22 SOLVER
10 <u>MVAR\$</u>	23 STOP
<i>11 "PIMNF"</i>	24 <u>LBL E</u>
12 <u>LBL A</u>	25 SOLVER
13 SOLVER	26 END

Data Registers Usage.

In the **standard (one-liner) form**, the Solvers use data registers {R00 – R04} and {R04 – R07} to store the **general** equation and the **muted** equation respectively. In this configuration the general equation is expected to be in the ALPHA registers for **SV\$+**.

In the **Chained (two-liner) form**, the Solvers also need data registers {R12 – R15} and {R08 – R11} to store the secondary equation and its muted form respectively. Note that in a two-liner configuration it is up to the FOCAL routine to store the secondary equation in registers {R12 – R15}, where it is expected by **SV\$+**. You can refer to the "**RdK**" and "**HTX**" examples for details.

It comes without saying that you should refrain from using these registers (depending on the case) in the FOCAL routines prepared for the Solvers.

A look under the hood.

A few other functions are provided that may become handy to you, either to play around during the learning phase or to take a peek on specific sub-sections of the Solver operation. When needed, these functions are also named according to the naming convention for standard and extended sets - like **VMENU**, vs. **VMNU+**, or **LCDV** vs. **LCDV+**

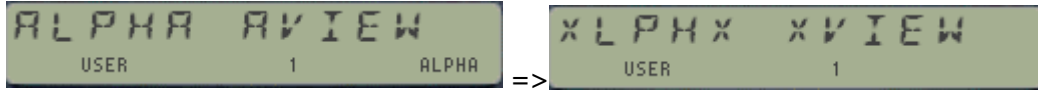
Here's a short description of their capabilities.

- **MPREP** is a convenient shortcut to prepare for the use of the Solver – taking care of the following housekeeping tasks: (1:) Clears UF 22, (2:) Clears UF 01, (3:) sets USER mode on, and (4:) Disables the local key assignments (in the 2 top-rows) so they don't interfere with the local labels. You can insert it as a program step in your FOCAL Solver programs if you want.
- **LKAOFF** and **LKAON** are used to disable or enable the key assignments on the local keys (2 top rows). Use them individually if you prefer this to the **MPREP** "bundled" way.
- **VMENU** and **VMNU+** read the variable declarations from the buffer header and build the menu choices in the display and ALPHA registers. This is automatically done during the execution of functions **MVARS** and **SOLVER** - and their extended counterparts.
- **LCDV** and **LCDV+** also read the variable declarations, then build a text string in the LCD (but not ALPHA). This string is used internally by **MVARS** and **MVRS+** to do the de-duplication and alphabetical sorting of their names. Note that the standard solver LCD string is shown with a dot behind each letter, to distinguished from an equal string from the extended set:



- **CLVARS** is a short routine that lets you clear the variable declarations, resetting the buffer header to the default zero values. Using any of the menu information functions above when they have been cleared will show the "NO VARS" message.
- **SHOW** is a handy function written by Doug Wilder, initially available in the BLDRAM and repurposed here (and previously in the ALPHA ROM as well). It allows "reading" a text string into the LCD without disturbing the ALPHA registers – which is very convenient if ALPHA has information that cannot be overwritten. This is how the menu names string is read by **MVARS**, whilst the general equation is still in the ALPHA registers.
- **DEDUP** is a global entry to the de-duplication routine. It'll handle strings in ALPHA of up to five characters in length, but not more. Larger strings will be truncated on entry.
- **DOSELF** and **DOSLF+** are also global ROM entries, this time to the self-programming code that is used by **MVARS/MVRS+**. In this form it is a prompting function, asking for the number of "blocks" to insert in program memory – each block comprised by the local LABEL, **SOLVER** (or **SOLVR+**) and STOP. Be careful not to enter a value larger than 10 or you'll run out of local labels to use!

- **MUTE** and **UNMUTE** are global ROM access points to the muting and unmuting processes performed by SOLVER. This consists of swapping the letter used for the unknown with an "X" – so it is prepared for the **EVAL\$** instruction. In this generic form they are prompting functions, expecting the ASCII decimal value of the character to mute (or restore) in the prompt. For example, using 65 as input will turn the string on the left to the one on the right:

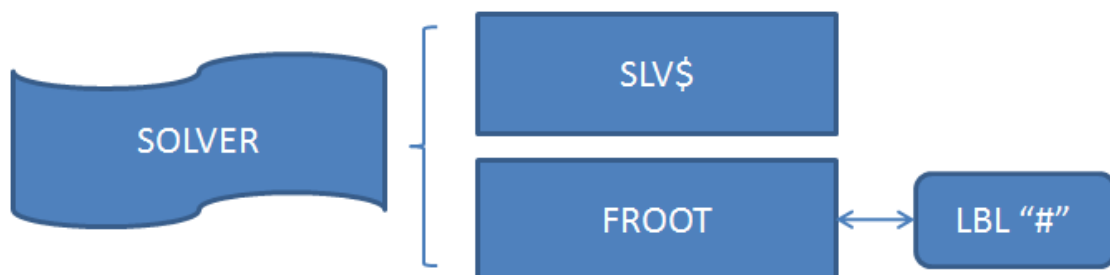


- **A-PM7** is the secret weapon used to insert any text string from ALPHA into program memory directly. **A-PM7** breaks the text in ALPHA in "chunks" up to 7-chars long, thus potentially will insert four text lines for 24 characters long text. This function is used internally by **MVAR\$** and **MVRS+** to enter the prompt values into the text line that follows itself in the program.

Not to be confused with the **A-PM** function in the Formula_Evaluation module, which uses the maximum length permitted in the text line, i.e. 15 characters – and therefore only two lines at most will ever be required. You can use **A-PM** to enter any general equation as a program text line once it has been created in ALPHA by **^FRMLA**.

- **T=Z?** is an auxiliary function that checks whether the values in the Z and T stack registers are equal. The result determines if the next line is skipped or not, pretty much like all standard test functions such as **X=Y?**
- Finally, **"#"** is a scratch FOCAL routine used by FROOT in case that the Newton/Secant option is selected (setting UF 01) during the Solver operation. You can ignore this one altogether, it's only there for housekeeping reasons – but if you're curious below is the program listing for your information:

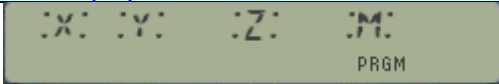
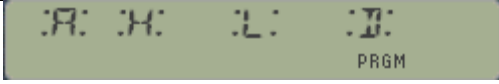
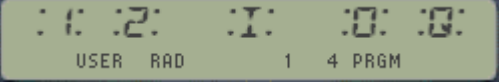

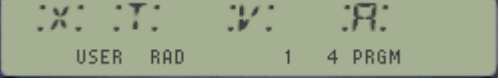


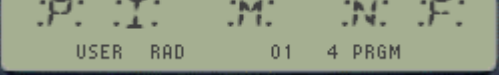
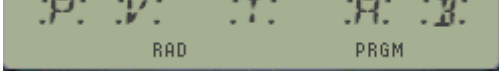

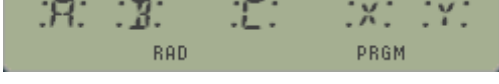

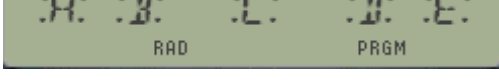
```
01 LBL "#"
02 RCL$ 04 ; brings the muted equation to ALPHA
03 EVAL$ ; evaluates the equation into X
04 END
```



Note that neither **SOLVER** or **SOLVR+** use the stand-alone routine **SV\$+** included in the module, but a dedicated version (embedded into the MCODE) reserved solely for this purpose.

Mini-Equation Library Examples.

The module comes equipped with a few examples of utilization of the Variable Solvers; use them to become familiar with the approach before attempting to write your own equations.

Routine	Equation	LCD Display
3DM	3D Vector Module $M = \text{SQRT}(x^2 + y^2 + z^2)$ 4 variables, MVARS	
CTRY	Catenary Curve $d = H [1 - (1/\cosh(L/2a))]$ 4 Variables, MVARS	
HTX	Heat Exchangers (Counterflow) See equations in next pages 5 1/2 Variables, MVARS , Chained .	
KPL	Kepler Equation $E - e \cdot \sin E = m$ 3 Variables, MVARS	
LMOV	Linear Movement $x = v \cdot t + a \cdot t^2 / 2$ 4 variables, MVARS	
RGAS	Real Gas Equation $P \cdot V = Z \cdot N \cdot R \cdot T$ 5 Variables + 1 constant, MVARS	
RdK	Redlich-Kwong EOS $P + a / [\sqrt{T} \cdot V_m \cdot (V_m + b)] = RT / (V_m - b)$; 5 Vars + 1 const, MVARS	
TVM\$	Time Value of Money (uses UF 02) See chained equations in page 13 5 1/2 Variables, MVARS , Chained	
VdW	Van-der-Waals EOS $P + (a/V_m^2) = R \cdot T / (V_m - b)$ 5 Variables + 1 constant, MVARS	
Y=P1	Straight Line Equation $y = A \cdot x + B$ 4 Variables, MVARS	
Y=P2	Quadratic Equation $y = A \cdot x^2 + B \cdot x + C$ 5 Variables, MVARS	
Y=P3	Cubic Equation $Y = x^3 + B \cdot x + C$ 5 Variables, MVARS	
Y=P4	Quartic Equation $Y = x^4 + A \cdot x^3 + B \cdot x^2 + C \cdot x + D$ 6 Variables, MVARS+	

The Quartic Equation sits by itself, as it uses the Extended Solver (**MVRS+** and **SLVR+**) to handle the six variables involved. This means that, contrary to the others, the variables must be named using the same letter as the buffer registers they're mapped to. In this case the classic equation

$$y = x^4 + A \cdot x^3 + B \cdot x^2 + C \cdot x + D \text{ becomes: } F = E^4 + A \cdot E^3 + B \cdot E^2 + C \cdot E + D$$

Of all these only the Van-der-Waals EOS, the Redlich-Kwong EOS and the Polynomial Equations require using initial intervals different from the default one $[0, 1]$. This is obviously due to the different roots that may exist, which also applies to the **VdW** and **RdK** cases- as it's nothing more than Cubic Equations "in disguise".

For the most part the internal Solver is capable of finding the solutions – but you may want to plug the "**Solve & Integrate ROM**" to use **FROOT**, a much more capable implementation. **Remember to use flag 00 to select your choice of solvers: Clear for the internal case, Set for FROOT.**

Numerical Examples.

1. Quadratic & Cubic Equations:- $y = A.x^2 + B.x + C$
 Given $a = 1, b = -4, c = -1, y = 0$, and default $[a,b] = (0, 1)$
 Solves: $x = -0.236067978$ for quadratic, $x = -0.239123279$ for cubic.
2. 3D Vector Module. - $M = \text{SQRT}(x^2 + y^2 + z^2)$
 Given $|v| = 5, x = 2, y = 4$ and default $[a,b] = (0, 1)$
 Solves: $z = 2.236068$
3. Catenary Equation - $d = H [1 - (1/\text{Cosh}(L/2a))]$
 Given $H = 42 \text{ m}, L = 100 \text{ m}, a = 43.5 \text{ m}$ and default $[a,b] = (0, 1)$
 Solves: $d = 17.814791 \text{ m}$
4. Kepler Equation. - $E - e.c. \sin E = m$
 Given $ec = 0.2$, and $m = 0.8$ and default $[a,b] = (0, 1)$
 Solves: $E = 0.964333888$
5. Linear Movement - $x = v.t + a.t^2/2$
 Given $x = 1 \text{ m}, V = 3 \text{ m/s}, a = 2 \text{ m/s}^2$ and default $[a,b] = (0, 1)$
 Solves: $t = 0.302775638 \text{ s}$
6. Real Gas Equation. - $P.V = Z.N.R.T$
 Given $P = 5 \text{ kPa}, V = 10 \text{ l}, T = 25^\circ\text{C}, Z = 0.161074$ and default $[a,b] = (0, 1)$
 Solves: $n = 0.125283 \text{ mol}$ (*Warning: always use SI units*)
7. Redlich-Kwong EOS:- $P + a / [\text{Sqrt}(T).V_m.(V_m+b)] = RT/(V_m-b)$
 Given $a = 14.66 ; b = 0.1226 ; P = 5 \text{ kPa}, T = 25^\circ\text{C}$, and $[a,b] = (0.1, 1.0)$
 Solves: $V_m = 0.617958693 \text{ m}^3/\text{mol}$
8. Van-der-Waals Equation- $P + (a/V_m^2) = R.T / (V_m-b)$
 Given $a = 14.66 ; b = 0.1226 ; P = 5 \text{ kPa}, T = 25^\circ\text{C}$, and $[a,b] = (0.1, 1.0)$
 Solves: $V_m = 0.614322294 \text{ m}^3/\text{mol}$
Note: it is easier with FROOT in the S&I ROM – can use wider intervals for guesses.

9. Time-Value of Money. - $[PV + FV \cdot (1+i)^{-n} + PMT \cdot [1 - (1+i)^{-n}] \cdot (p + 1/i) = 0$

Given "End mode" (CF 02), $FV = 0$, $PMT = \$650$, $n = 360$ months, and $I = 14.25\%$ (yearly) Solves: $PV = \$5,395.59$

10. Heat Exchangers (Counter flow). (Note: Use $k_2 = -k_2$ for Parallel flow)

This one includes a prompt for the product $A.U$, taken here as a constant of the exchanger. If this also needs to be a design variable then the 6-Variables **SLVR+** should be used instead,

Equation used: $Q = k_1 \cdot \{ [T1(i) - T2(o)] / (1 - k_{12}) \} \cdot \{ \exp [-U \cdot A \cdot (1 - k_{12}) / k_1] - 1 \}$

with $k_{12} = k_1/k_2$, $k_1 = m_1' \cdot Cp_1$, and $k_2 = m_2' \cdot Cp_2$

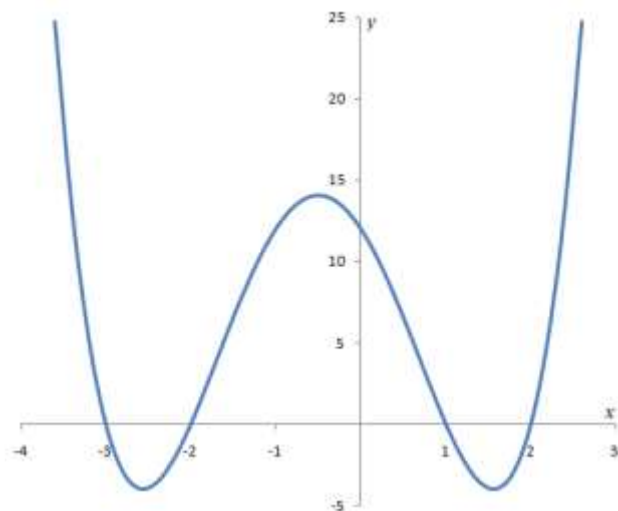
Given $UA = 115.8185$ kcal/°C.h ; $k_1 = 5$ kcal/ °C.min ; $k_2 = 7.7368$ kcal/°C.min ;

$T1(i) = 20$ °C; and $T2(i) = 90$ °C. Solves: $Q = 5,999,998.79$ kcal/min

11. Quartic Equation. - $F = E^4 + A \cdot E^3 + B \cdot E^2 + C \cdot E + D$

Given $a=2$; $b=-7$; $c=-8$; $d=12$; $F=0$ and default $[a,b] = (0, 1)$

Solves: $E = 1.777777$ Can you find the other roots? Try changing the a^b initial guesses...



Program Remarks.

Looking at the code you can see that all the examples above using **MVARS** are placed together in the same FOCAL program, with the individual global labels and equations sharing the same local labels' section. This is a very convenient arrangement that saves a lot of room, and it's possible because of the design of the **MVARS** and **SOLVER** functions.

For the main program below, note the chain sign in the "**HTX**", "**TVM\$**" and "**RdK**" routines, signaling to the **SV\$+** solver routine that it is a two-liner implementation. Also note the common use of the soft-key labels for all routines, possible because **MVARS** halts the program execution at the program pointer. Note as well that the **FROOT** method only supports the one-liner case, and therefore cannot be used with chained equations (**TVM\$** and **RdK**). This is also handled by the **SOLVER** functions, which automatically clears UF 00 when the chain sign is found in the secondary equation.

Program Listing

```

01 *LBL "LMOV"
02 "c*b+(d*b^2)/2-a"
03 MVARs
04 "XTVA"
05 *LBL "HTX"
06 "AU=?"
07 PROMPT
08 LET=
09 6
10 "e-T*(E(F*(a-b))/"
11 "|- a/b)-1)"
12 STO$
13 12
14 "$a*b*(c-d)/(b-a"
15 "|-)"
16 MVARs
17 "12IOQ"
18 *LBL "TVM$"
19 E
20 FS? 02 ; begin mode?
21 CLX
22 LET= ; load constant
23 6 ; in buffer "F"
24 "a+e*T+c*(1-T)"
25 "|-*(F+100/b)"
26 STO$ ; load primary
27 12 ; in {R12-R15}
28 "$ (1+b/100)^#d" ;secondary eq
29 MVARs
30 "PIMNF"
31 *LBL "KPL"
32 RAD ; angular mod
33 "b-c*S(b)-a"
34 MVARs ; show menu
35 "MEC" ; menu choices
36 *LBL "Y=P1"
37 "a*c+b-d"
38 MVARs ; show menu
39 "ABXY" ; menuchoices
40 *LBL "Y=P2"
41 "a*d^2+b*d+c-e"
42 GTO 01
43 *LBL "Y=P3"
44 "d^3+a*d^2+b*d+c"
45 "|- -e"
46 *LBL 01
47 MVARs ; show menu
48 "ABCXY" ; menu choices

49 *LBL "3DM"
50 "Q(a^2+b^2+c^2)-"
51 "|- d"
52 MVARs ; show menu
53 "XYZM" ; menu choices
54 *LBL "CTRY"
55 "b*(1-(1/HC(c/2/"
56 "|- a))) -d"
57 MVARs ; show menu
58 "AHLD" ; menu choices
59 *LBL "RGA$"
60 XEQ 00 ; load constant
61 "a*b-c*d*e*F"
62 MVARs
63 "PVTZN"
64 *LBL "RdK"
65 "a+T-F*c/(b-e)"
66 STO$ ; load primary
67 12 ; in {R12-R15}
68 "$d/Q(c)/b/(b-e)" ;secondary eq
69 GTO 01
70 *LBL "VdW"
71 "a+d/b^2-F*c/(b-"
72 "|- e)"
73 *LBL 01
74 XEQ 00 ; load constant
75 MVARs
76 "PVTAB"
77 *LBL A
78 SOLVER ; solvevar A
79 STOP
80 *LBL B
81 SOLVER ; solve var B
82 STOP
83 *LBL C
84 SOLVER ; solve var C
85 STOP
86 *LBL D
87 SOLVER ; solvevar D
88 STOP
89 *LBL E
90 SOLVER ; solvevar E
91 STOP
92 *LBL 00
93 8.314459848
94 LET=
95 6
96 END

```

Ending with the Quartic equation, there's nothing special to remark in this case except that we're using MVRs+ and SLVR+ to handle 6 variables in the solver – and therefore there are six local labels instead of five.

01 <u>*LBL "Y=P4"</u>		13 SOLVR+	; solve var C
02 "e*(c+e*(b+e*(a+")		14 STOP	
03 "`e))) + d - F"		15 <u>*LBL D</u>	
04 MVRs+	; show menu	16 SOLVR+	; solve var D
05 "ABCDEF"	; menu choices	17 STOP	
06 <u>*LBL A</u>		18 <u>*LBL E</u>	
07 SOLVR+	; solve var A	19 SOLVR+	; solve var E
08 STOP		20 STOP	
09 <u>*LBL B</u>		21 <u>*LBL F</u>	
10 SOLVR+	; solve var B	22 SOLVR+	; solve var F
11 STOP		23 END	
12 <u>*LBL C</u>			

Other examples not included in the ROM.

Parallel Resistors: $1/\Sigma R = 1/R_1 + 1/R_2 + 1/R_3$

01 <u>*LBL "RPAR"</u>	10 SOLVER
02 "1/(1/b+1/c+1/d)"	11 STOP
03 >"-a"	12 <u>*LBL C</u>
04 MVARs	13 SOLVER
05 "S123"	14 STOP
06 <u>*LBL A</u>	15 <u>*LBL D</u>
07 SOLVER	16 SOLVER
08 STOP	17 STOP
09 <u>*LBL B</u>	18 END

Note that all resistances must be non-zero, and that the equation variables declaration must be explicit!

Example: $R_1 = R_2 = R_3 = 1 \Rightarrow \Sigma R = 0.545454545$

Equation Libraries Revisited.

In this chapter you'll find an update to the works done by Greg McClure and Mark Fleming on related subjects, like the AOS Simulator and the Equation Library respectively. See the excellent manual available at: <http://www.hpmuseum.org/forum/thread-8795.html>)

New Record and Pointer Functions.

A few new record pointer functions are included to complement the original set from the Extended Functions module. The intent was to facilitate the operation of the Equation Library FOCAL programs, saving some steps here and there and providing more flexibility in their use.

The functions are shown on the table below:-

Function	Description	Input	Output
ADVREC	Advance Pointer in Record	Number of positions in X	Pointer is moved
ARCLCHR	ARCL Characters	Number of Chars in X	Chars added to ALPHA
READREC	Read Nth. Record	N in X	String in ALPHA
REC-	Move pointer one position down	none	Pointer moved
REC+	Move pointer one position up	None	Pointer moved
SEEK*	Seek pointer (Customized)	Pointer position in X	Pointer Set to new pos.

The pointer functions mostly deal with updating the file header location where the pointer position is saved. They verify that the chosen position is within the boundaries of the ASCII file and adjust it accordingly. See the File Header diagram below for details:

T	A	D	R	-	C	H	R	R	E	C	S	Z	E
13	12	11	10	9	8	7	6	5	4	3	2	1	0

An interesting challenge arises because the records are of variable length, so there isn't a constant number of characters per record. This is handled by reading the record-length nybble, located at the beginning of each record.

For comparison purposes the standard approach used by the original X-Functions always requires recalling the pointer first using **RCLPT(A)**, adding or subtracting the number of positions using the stack, and resetting the pointer using **SEEKPT(A)**. This alters the stack registers and requires multiple steps per action – as opposed to using new pointer functions, with a more straightforward method. for example, **REC+** is functionally equivalent to (but has none of the shortcomings of):

RCLPT(A), INT, 1, +, SEEKPT(A)

See page #35 for the program listing of Mark's **EQNLIB** program using the new pointer functions and taking advantage of the modified **LET=** behavior that saves a substantial number of FOCAL program steps. -

MCODE Listings.

See below the MCODE for those of you curious and with an inclination to look under the hood. As you can see the REC+/REC- routines are fairly short, mostly leveraging the OS routine [CURFLT] to do the heavy lifting.

Header	AF37	0AB	"+"		<u>Next Record</u>
Header	AF38	003	"C"		FName in ALPHA
Header	AF39	005	"E"		
Header	AF3A	012	"R"		Ángel Martín
REC+	AF3B	108	SETF 8		
	AF3C	033	JNC +06		[MERGE]
Header	AF3D	0AD	"_ "		<u>Previous Record</u>
Header	AF3E	003	"C"		FName in ALPHA
Header	AF3F	005	"E"		
Header	AF40	012	"R"		Ángel Martín
REC-	AF41	104	CLRF 8		
MERGE	AF42	03E	B=0 MS		uses current file!
	AF43	249	?NC XQ		
	AF44	0F0	->3C92		[CURFLT]
	AF45	0B0	C=N ALL		A(10:8) - 1 rg addr (name)
	AF46	0FC	RCR 10		N(12:10) - add file header
	AF47	270	RAMSLCT		select file header addr
	AF48	038	READATA		read FLDH value
	AF49	106	A=C S&X		put file size in A.X
	AF4A	1A6	A=A-1 S&X		REC# is zero-based
	AF4B	03C	RCR 3		move rec# to C.X
	AF4C	10C	?FSET 8		next?
	AF4D	03B	JNC +07		no, skip
NEXT	AF4E	366	?A#C S&X		is SZE=REC# ?
	AF4F	063	JNC +12d		yes, do nothing
	AF50	306	?A<C S&X		is SZE<REC# ?
	AF51	057	JC +10d		yes, do nothing
	AF52	226	C=C+1 S&X		increase record
	AF53	023	JNC +04		[RESTORE]
PREV	AF54	2E6	?C#0 S&X		
	AF55	033	JNC +06		
	AF56	266	C=C-1 S&X		decrease record
RESTORE	AF57	03C	RCR 3		
	AF58	046	C=0 S&X		clear byte#
	AF59	13C	RCR 8		rotate back
	AF5A	2F0	WRTDATA		update header
IGNORE	AF5B	046	C=0 S&X		
	AF5C	270	RAMSLCT		select chip0
	AF5D	3E0	RTN		

And here's the listing for the more general **ADVREC**:

Header	AF0F	083	"C"	
Header	AF10	005	"E"	
Header	AF11	012	"R"	
Header	AF12	016	"V"	
Header	AF13	004	"D"	
Header	AF14	001	"A"	
ADVREC	AF15	03E	B=0 MS	<u>Advance Record</u>
	AF16	249	?NC XQ	FName in ALPHA
	AF17	0F0	->3C92	delta in X
	AF18	046	C=0 S&X	
	AF19	270	RAMSLCT	
	AF1A	0F8	READ 3(X)	
	AF1B	248	SETF 9	Ángel Martin
	AF1C	2FE	?C#0 MS	uses current file!
	AF1D	013	JNC +02	
	AF1E	244	CLRF 9	
	AF1F	38D	?NC XQ	
	AF20	008	->02E3	[CURFLT]
	AF21	0E6	C<>B S&X	
	AF22	0B0	C=N ALL	
	AF23	0FC	RCR 10	
	AF24	270	RAMSLCT	
	AF25	038	READATA	
	AF26	106	A=C S&X	
	AF27	1A6	A=A-1 S&X	
	AF28	066	A<>B S&X	
	AF29	03C	RCR 3	
	AF2A	24C	?FSET 9	
	AF2B	037	JC +06	
DECR	AF2C	0A6	A<>C S&X	
	AF2D	246	C=A-C S&X	
	AF2E	043	JNC +08	
	AF2F	046	C=0 S&X	
	AF30	033	JNC +06	
INCR	AF31	206	C=C+A S&X	
	AF32	066	A<>B S&X	
	AF33	306	?A<C S&X	
	AF34	013	JNC +02	
	AF35	0A6	A<>C S&X	
	AF36	03C	RCR 3	
	AF37	046	C=0 S&X	
	AF38	13C	RCR 8	
	AF39	2F0	WRTDATA	
	AF3A	046	C=0 S&X	
	AF3B	270	RAMSLCT	
	AF3C	3E0	RTN	

A new twist to an Old Solver. {SVEQ\$ }

Once upon a time there was a FOCAL program used as a driver to select equations, their known variables and to solve for the unknowns. Said driver program was based on the **SOLVE** function within the HP-41 Advantage and used the standard FOCAL approach to program each of the equation subroutines.

The new twist consists of replacing the FOCAL programming with formula strings evaluated by **EVAL\$** instead – straight forward once you get comfortable with the Formula Evaluation functionality!

The program listing is shown below, note the use of user flag F6 (as a proxy for the data entry flag status in the Driver program) to signal whether calculation or menu displaying should be performed by the equation subroutines. Note as well that the selection of the unknown variable is made by storing the register index in R00 – so the equation variable will be retrieved with a RCL IND 00 statement, where the valid range is 1 to 5 (for R00 to R05).

Finally, the program assumes that at the menu has least three variables (no point in using a solver for trivial cases, or is it?) and checks that the menu string length is long enough when the fourth and fifth variable are called upon (pressing LBL D or LBL E respectively).

1	LBL "SLVEQ\$"		33	LBL D	4th param entered / to be solved if valid
2	SF 27		34	FS? 05	valid length?
3	LKAOFF		35	SF 04	yes, set middle length
4	LBL F		36	FC? 04	valid length?
5	"EQ NAME: "		37	GTO 01	nope, loop back
6	PMTA		38	4	yes, mark r04
7	ASTO 06		39	GTO 02	
8	LBL 01		40	LBL E	5th param entered / to be solved if valid
9	CF 04	default	41	FC? 05	valid length?
10	CF 05	default	42	GTO 01	nope, loop back
11	SF 06	to display the menu	43	5	yes, mark r05
12	XEQ IND 06	display menu in LCD	44	LBL 02	common param handling
13	ALENG	get its length	45	FC? 22	was it entered?
14	E1	as a proxy for # of Vars	46	GTO 00	no, must need to be solved
15	X<Y?		47	X<>Y	
16	SF 05	flags LEN>10	48	STO IND Y(2)	save in proper register
17	CLX		49	GTO 01	get next params
18	9		50	LBL 00	
19	X<=Y?		51	STO 00	save location to solve
20	SF 04	flags LEN >= 9	52	E-99	to avoid zero
21	CF 22	reset data entry flag	53	E	assume virtual 0 and 1 for guesses
22	PROMPT		54	"a^b=?"	
23	GTO 01	nothing entered, ask for params again	55	PROMPT	get better guesses if supplied
24	LBL A	1st param entered / to be solved	56	CLA	
25	E	mark r01	57	ARCL 06	get equation name
26	GTO 02		58	CF 06	signal solve action
27	LBL B	2nd param entered / to be solved	59	FROOT	in the Solve & Integrate ROM
28	2	mark r02	60	STOP	
29	GTO 02		61	GTO 01	get next params
30	LBL C	3rd param entered / to be solved	62	LBL J	exit routine,
31	3	mark r03	63	LKAOFF	restore user key assignments for top two rows
32	GTO 02		64	END	

Show me the Money. { **TVM\$** }

The Time Value of Money equation poses some challenges to the Equation Library Solver in a couple of accounts: the number of variables involved exceeds the standard capability, and the length of the formulas goes beyond the 24-characters boundary of the ALPHA registers.

Greg wrote the **TVM\$** subroutine below to overcome these limitations, a mini-Solver dedicated to this particular subject that relies on a chained **EVAL\$** calculation. This routine is accessed by the main driver program **SLVEQ\$**– which prompts for the equation name and handles the value entering for the known variables as well as the trigger to solve for the unknown.

By the way **SLVEQ\$** also uses the **FROOT** function from the "Solve & Integrate" Module to obtain the root – so make sure it is plugged in the calculator when you work on this subject.



The program listing for the **TVM\$** routine is shown below.

1	LBL "TVM\$"	global label	15	RTN	done.
2	FS? 06	Show menu?	16	LBL 10	
3	GTO 00	yes, divert	17	RCL 01	R01 -> L
4	STO IND 00	no, save value in mapped register	18	STO L	
5	XEQ 10	store data in buffer	19	RCL 02	R02 -> T
6	"1+b/100"		20	RCL 03	R03 -> Z
7	EVALY		21	RCL 04	R04 -> Y
8	"c*((1-Y^#d"	write first part	22	RCL 05	R05 -> X
9	>")/b*100)"		23	"XYZTL"	
10	FS? 00	BEGIN mode?	24	SHFL	puts stack in buffer
11	"*Y"	yes, add pre-fix	25	RTN	
12	EVAL\$	evaluate it	26	LBL 00	
13	"a+X+e*Y^#d"	write the second part	27	"PV I PM N FV"	menu variables
14	EVAL\$	chained calculation	28	END	done.

Numerical Example:

Calculate the future payment of an initial capital of \$5,000 with a 3% annual interest with yearly deposits of \$500 during 5 years. Use Begin and End modes to compare results.

Solutions: BEGIN: \$-8,450.938
 END: \$-8,530.575

An Equation Library Using the Formula Evaluation ROM (M. Fleming)



The Formula Evaluation ROM gives the HP-41CX/CL owner the ability to evaluate algebraic expressions stored as text strings in the Alpha register or in extended memory text files. The ROM functionality is quite extensive, as shown by the examples in the manual. The examples included the use of a solver program to find the root(s) of a given expression.

The first application of the ROM that came to my mind was the Equation Library from the HP48 series calculators. Wouldn't it be nice to have a set of equations to choose from, equations that you could then establish values and solve for unknowns. Even better would be the ability to add your own equations to the library. The EQNLIB program and its associated Extended Memory¹ text file EQNS provides that capability.

Limitations

Any formula you use in the Formula Evaluation ROM must fit within the 24 character size limit of the Alpha register. Additionally, the Equation Library uses the five formula variables 'a' through 'e' for the top row of user keys. This limits any equation you use to only five independent variables. As we will see in a later section, more variables and longer equations can be accommodated through equation chaining.

New in revision B, the Equation Library program allows you to use two formulas for more complicated expressions. The result of a first formula can be used in a second formula to complete the calculation. This does require a larger expression to be broken into two partial expressions, the details for which are explained in a later section.

Notation

Throughout this manual I will use *equation* to mean an entry in the Equation Library, *formula* to mean the alpha string accepted and processed by the Formula Evaluation ROM, and *expression* to mean the common or formal mathematical representation, often with implied multiplication, as in $E = IR$.

¹ Although written for the HP-41CX/CL extended memory, EQNLIB can be adapted for use with HEPAX memory by changing the file operation commands.

Program Setup

The ROM &MOD files contain mostly FOCAL programs. The Equation Library is dependent on the Formula Evaluation ROM version 1F, which itself is dependent on Library#4 and the HP-41CX ROM. Two emulators have been tested with the above configuration as well as a calculator with the 41CL CPU board.

On the V41 emulator add the Equation Library MOD file to your 41cx setup file along with the dependent modules. For go41cx/cxt, copy the MOD file to the files/modules directory and perform an import. You can obtain a MOD file created by Greg McClure for go41 that contains the Formula Evaluation ROM, the Formula Apps ROM and OS/X3 at the end of [this post](#)². Plug Greg's OSX_BS4X7H_EVAL_S4module into port 4 (turn off OSX& Library4 in the settings) and the Equation Library module into port 3. If you have a 41CL calculator, then transfer the ROM file image to a suitable RAM page (830 for instance) and plug the RAM page into an empty port page. Do the same for the Formula Evaluation ROM (FORM_1F.ROM)file.

Once configured, the program INIEQN will create an initial Equation Library text file in extended memory called EQNS. Other library files can be created containing your own equations. The format of a library file will be described in detail in a later section. For now, run INIEQN to get started with Equation Library.

Program Usage

Program operation is straightforward. Execute EQNLIB and it will display the name of the first equation in the default Equation Library (EQNS). If you have created your own library file then enter the name of the file in Alpha and execute EQN\$.

Use the F key (NEXT) to scroll forward through the list of equations and the G key (PREV) to scroll backwards. The H key (VIEW) lets you switch between the name of an equation and its formal algebraic expression as shown below.



²<http://hpmuseum.org/forum/thread-8581-post-76569.html>

You can pick an equation and start solving by pressing the J key (MENU) to display the solver menu. To set a value for a variable, key in the value then press the user key beneath the variable. The menu is redisplayed each time you set the value of a variable, and you can press the J key to redisplay the menu after solving for a variable. For the Quadratic equation, we would first need to establish the values of A, B and C.



Using the example from the Formula Evaluation ROM manual, let's use the values $A=1$, $B=4$ and $C=1$. If we set X to 5 then press Y, we get the value 46. Specify 41 as the value of Y and press X to compute a value of 4.6332 for X. Naturally, if we'd like to compute the root of this quadratic, we would set Y to 0. We end up with a result of -0.2679, but this is only one of two real roots. How do we find the other?

Establishing Initial Guesses

By default the initial guesses for the solver are 0 and 1, which will work fine for many equations. For equations with multiple roots we need a way to supply our own initial guesses. EQNLIB provides this ability. Just enter your lower and upper guess at the prompt " $a^b=?$ ". Since the initial guesses 0 and 1 returned a negative root for our example quadratic equation, the second root must be less than -0.27, so type

`-1 ENTER -5 SHIFT R/S`

The result of the calculation is -3.7321 for the second root.

Equation Chaining

The Equation Library uses the Formula Evaluation ROM variables ‘a’ through ‘e’, and these variables retain their values between expression evaluations. This allows one equation to establish values and another equation to use them, much the same way as using the same variable name in multiple equations with the HP Solver. Let’s continue with the quadratic equation to illustrate the idea of equation chaining.

Consider the expression for calculating the two real roots of a quadratic equation:

$$x_1, x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Let us now have two equations adjacent to one another in the Equation Library and call them POSROOT and NEGROOT. Each would have a menu with the constants A, B and C as the first three user keys and with X1 or X2 following. Each equation would show a different expression and use a different formula string. A hint of the structure of an equation in the Equation Library can be gleaned from the table below.

Equation Name	POSROOT	NEGROOT
Equation Expression	X1=-B+SQRT (B^2-4AC) / 2A	X2=-B-SQRT (B^2-4AC) / 2A
Equation Formula	(#b+Q (b^2-4*a*c)) / 2/a-d	(#b-Q (b^2-4*a*c)) / 2/a-d
Equation Menu	A B C X1	A B C X2

Select the POSROOT equation and hit the MENU key (“J”) to display its user key menu. Enter the values for A, B and C and then press the X1 user menu key (“D”) to solve for the first root. Press the NEXT button (“F”) then press the “D” key again to solve for the second root. No need to press the MENU key before solving for the second root!

Obviously this is a simple example of what can be a powerful technique to extend the capabilities of the Equation Library and is described in the Formula Evaluation ROM manual itself.

Summing It Up

A summary of the keys and flags used by the EQNLIB program is shown below.



Key	Label	Function	Description
F	NEXT	Next equation	
G	PREV	Previous equation	
H	VIEW	Toggle display	Switch between Name & Expression
J	MENU	Initiate Solver	
A-E	Variables a - e	Set or Solve Variable	Enter data to set variable or none to solve
SHIFT A-E	Initial Guesses	Solve Variable	Enter Y=lower guess, X=higher guess
Flag	Clear	Set	Usage
Flag 10	No trace	Trace	Show intermediate solver results

Execute EQNLIB and it will display the name of the first equation in the library. From there you can scroll through the set of equations using the NEXT and PREV key. If you use VIEW to switch to the equation expression, then scrolling will show all of the available expressions in the library.

The internal solver uses flag 10 to indicate whether to display intermediate results. If the solver doesn't seem to be converging on an answer, try pressing R/S and then set flag 10 with SF 10. When you resume the program with R/S you should see the intermediate values flash by.

Finally, many equations need a good set of guesses to even return a valid result. The equation for the equivalent resistance of two resistors in parallel is a good example. Unless your guesses bracket the correct answer, the result displayed by the solver will be incorrect or will not converge.

Equations with multiple closely-spaced roots can also be a challenge. Try the cubic equation with constants A=6, B=11 and C=6. The roots are -1, -2 and -3. The default initial guesses will find the first, but the others require a bit of hunting around with your guesses.

Now that you know how to use the Equation Library, let's see how to add your own equations.

Roll Your Own Equation

Since we used a general form of the quadratic equation, let's try to create our own root finder for a fourth-order equation. We can write out a normalized fourth order power expression as

$$y = x^4 + ax^3 + bx^2 + cx + d$$

This expression has six unknowns which exceeds our limit of five. But by applying Horner's Method, with y set to zero, we get

$$d + x(c + x(b + x(a + x)))$$

Now we can construct an entry in the EQNS file to find roots of this expression. First, we need a name for the Equation Library entry. Names should be twelve characters or less to avoid display scroll. Let's choose "4TH ORDER" for a name. Next, we need an expression in the HP-41 character set for the above. The only change needed to use the above expression would be to change all characters to uppercase.

The next step is to design the equation menu and write the formula. If we place the power equation constants 'a' to 'd' first in the menu, then we can use the expression exactly as it is but with the 'e' variable substituting for x in the expression above. But rather than take the easy approach (and to illustrate the more general way to map an expression to a formula) I'll place the x variable in the menu first. Let's create a menu like **X? A B CD** where the question mark following the variable name X reminds us that this is what we want to solve for³. Here's an illustration that will help with the mapping task.

Menu	X?	A	B	C	D
Key/Variable	A/ a	B/ b	C/ c	D/ d	E/ e
Expression	d+x (c+x (b+x (a+c)))				
Formula	e+a* (d+a* (c+a* (b+d)))				

The formula is slightly confusing, but only because we're using an expression with our 'a' through 'e' variables names in it. Try doing the above mapping process with the expression for Ohm's Law. Easier to visualize now?

Adding a new equation means adding four more records to the end of the EQNS XM text file. A program called APPEQN (Append Equation) is listed in the Appendix that will do exactly that for you. Just modify the four text lines and run the program. Another approach is to use ED to enter the four lines. ED cannot produce the special characters that OS/X or CCD provides, so you can use temporary substitutes and then run the SAR (Search and Replace) program also listed in the appendix. I would

³ Mind you, we can always set the X value and three of the constants and then solve for the fourth, can't we? Why not!

suggest using '<', and '>' for parentheses and '#' (SHIFT-H) for not-equal when entering the equation, then run the search and replace program where you can enter the proper character at the replace prompt.

So here then are the four records we need to add to the Equation Library file:

```
4TH ORDER
D+X(C+X(B+X(A+X)))
e+a*(d+a*(c+a*(b+a)))
X?A B CD
```

Once you enter these lines into EQNS, run EQNLIB and give the equation a try. Taking the example from the Formula Evaluation ROM manual, try using 2 for A, 3 for B and 4 for C, 5 for D and then solve for X. With the default initial guesses you get -0.8569 for the answer. Can you find the other three roots?

To support more complex expressions, you can break the expression into two parts, with the result of the first expression serving as input to the second. To do this you create one formula that computes the partial expression. The result is placed in stack register T by the EVALT function. The second formula uses register T in the formula to incorporate the partial result from the first expression. Let's use the following formula for converting between Nominal and Effective interest rate as an illustration.

$$Effective\ Rate = \left[\left(1 + \frac{Nominal\ Rate}{100 \times P} \right)^P - 1 \right] \times 100$$

Rates are expressed as percentages such as 4.5% and P is the number of payments per year. Let the formulas be based on the following menu

EFF NOM P

We can use the power expression in parentheses as the first partial expression. The formula would then be

$$(1+b/100/c)^c$$

The formula that incorporates this result would then be

$$(T-1) * 100 - a$$

The result through substitution would be the complete formula

$$((1+b/100/c)^c - 1) * 100 - a$$

Equation Library Format

An Equation Library is an extended memory text file. Each equation in the library occupies four records, none of which should exceed the 24 character length of the Alpha register. For each equation, the first record is the name of the equation, the second record is the expression or hint text displayed to the programmer when switching between views, the third record is the formula itself and the fourth record is the menu displayed when solving the equation.

00	LINEAR	32	RLC FREQ.
01	$Y=AX+B$	33	$F0=1/\text{SQRT}(LC)$
02	$c*a+d-b$	34	$1/Q(b*c)-a$
03	X Y A B	35	$F0 L C$
04	QUADRATIC	36	GAS EQUATION
05	$Y=AX^2+BX+C$	37	$PV=NRT$
06	$c*a^2+d*a+e-b$	38	$c*(16629/2000)*d-a*b$
07	X Y A B C	39	P V N T
08	CUBIC	40	LIN. MOTION
09	$Y=X^3+AX^2+BX+C$	41	$X=VT+1/2*AT^2$
10	$a^3+c*a^2+d*a+e-b$	42	$c*b+1/2*d*b^2-a$
11	X Y A B C	43	X T V A
12	4TH ORDER	44	NEWTONS LAW⁴
13	$D+X(C+X(B+X(A+X)))$	45	$F=G*M1*M2/R^2$
14	$e+a*(d+a*(c+a*(b+a)))$	46	$e*b*c/d^2-a$
15	X? A BC D	47	F M1 M2 R G
16	POSROOT	48	INTEREST
17	$X1=(-B+\text{SQRT}(B^2-4AC))/2A$	49	P=PERIODS
18	$(\#b+Q(b^2-4*a*c))/2/a-d$	50	$((1+b/100/c)^c-1)*100-a$
19	A B C X1	51	EFF NOM P
20	NEGROOT	52	+INTEREST
21	$X2=(-B-\text{SQRT}(B^2-4AC))/2A$	53	$(T-1)*100-a$
22	$(\#b-Q(b^2-4*a*c))/2/a-d$	54	$(1+b/100/c)^c$
23	A B C X2	55	EFF NOM P
24	OHMS LAW	56	+TVM END MODE
25	$E=IR$	57	$a+T+e*(1+b)^{\#d}$
26	$b*c-a$	58	$(1+b)*c*((1-(1+b)^{\#d})/b)$
27	E I R	59	PV I PM N FV
28	PARALLEL R	60	+TVM BEG MODE
29	$1/R1=1/R2+1/R3$	61	$a+T+e*(1+b)^{\#d}$
30	$1/b+1/c-1/a$	62	$c*((1-(1+b)^{\#d})/b)$
31	R1 R2 R3	63	PV I PM N FV

The table above shows the set of equations for the EQNS extended memory text file. When a complex expression needs a formula that exceeds the 24 character limit for formula size, the formula can be broken into two partial formulas as outlined in the preceding section. Place the first partial formula in the third record and the second formula in the second record. To flag the equation as a two-formula problem put a '+' sign at the beginning of the equation name.

⁴ G is 6.67408×10^{-11} in SI units.

Notice the two versions of the interest rate calculation example. The formula was short enough to fit the 24-character limit as shown in the **INTEREST** equation entry. The **+INTEREST** equation entry is our example of how to handle more complex formulas and the way they are represented in the library.

Greg provided the last pair of TVM equations as a fair illustration of what can be accomplished with the Formula Evaluation ROM. Both examples require a formula that is too long to fit within the 24-character Alpha limit and hence must be split into two formulas. The Interest rate (I) should be expressed as a percentage divided by the number of periods per year, e.g. 6% would be 0.06/12 for something like a car loan. P is the number of payments, as in 360 for a 30 year mortgage. If Present Value (PV) and Future Value (FV) are positive then Payment (PM) will be a negative value.

Creating Your Own Library

Now that you know the format of an equation entry, how would you add new equations to the default EQNS library or create your own custom library? A new custom library is just a text file created by putting the filename in Alpha and a starting file size of say 4 registers in X and then executing the CRFLAS command.

Use the COPY command to copy the APPEQN program to user memory. The program has two entry points; the default APPEQN label that will append a new equation entry to the default EQNS library and the APP\$ label that will append a new equation entry to the text file named in the Alpha register.

Edit the four text line records for name, expression, formula and menu. Don't forget for a long text record you may need to add a second line starting with the append symbol (SHIFT-K) to complete the full text of an expression or formula record. The program, through either entry point, will go to the last record in the file, extend the file length by ten registers, and then append the text records you've entered. Repeat until you've added all of your equations. You can delete APPEQN when you are finished.

Appendix

Listings for the EQNLIB program and various support programs in the EQNLIB module are provided.

SAR: Search and Replace program

1	<i>LBL "SAR"</i>	15	POSFL
2	<i>"FNAME: "</i>	16	X<0?
3	PMTA	17	GTO 01
4	<i>LBL "SAR\$"</i>	18	ALENG
5	0	19	DELCHR
6	SEEKPTA	20	CLA
7	<i>"SERCH: "</i>	21	ARCL 01
8	PMTA	22	INSCHR
9	ASTO 00	23	GTO 00
10	<i>"RPLCE: "</i>	24	LBL 01
11	PMTA	25	CLA
12	ASTO 01	26	WORKFL
13	LBL 00	27	END
14	ARCL 00		

This program will search for and replace strings in an extended memory text file. The search and replace strings are limited to a maximum of six characters. The program will save the name of the XM text file and restore it to the Alpha register so that you can invoke the ED editor to examine results. The last three text pointers are also left on the stack so that you can roll the stack to one of these pointers and do a SEEKPT before invoking the editor.

The internal entry SAR\$ can be used to bypass the prompt for the XM text file name if it is already in the Alpha register. Note that because the filename is saved to a single register, the name is also limited to six characters.

If you need to insert the special CCD/OSX characters like '(' into a text file with ED then just use the unshifted (USER mode on) character like '<' instead. You can use the search and replace to correct the characters in your file. (*)

(*)This won't be necessary if you use the **ED+** function form the Warp_Core Module, which includes support for all lower-case and special characters.

APPEQN: Append Equation programs

1	LBL "APPEQN"	17	"EQN. NAME"
2	"EQNS"	18	PMTA
3	LBL"APP\$"	19	APPREC
4	0	20	"EXPRESSN. "
5	SEEKPTA	21	PMTA
6	LBL 00	22	APPREC
7	1	23	"FORMULA"
8	+	24	AVIEW
9	SF 25	25	PSE
10	SEEKPT	26	^FRMLA
11	FC?C 25	27	APPREC
12	GTO 00	28	"EQN. MENU"
13	FLSIZE	29	PMTA
14	4	30	APPREC
15	+	31	CLA
16	RESZFL	32	WORKFL
		33	END

The APPEQN program will seek to the end of the EQNS Equation Library file and append a new equation of your own design. Simply edit the four text lines (in bold) for your equation then run the program. If you want to append to an equation file other than the default EQNS, then put the name of the file in the Alpha register and execute APP\$.

EQNLIB& INIEQN: Main Equation Library programs

```

11:11AM 03/04
01*LBL "EQNLIB"
02 LKAOFF
03 "EQNS"
04 CLX
05 SEEKPTA
06 GETREC
07 AVIEW
08 SF 27
09 RTN
10*LBL F
11 CLX
12 4
13 GTO 00
14*LBL G
15 CLX
16 4
17 CHS
18 *LBL 00
19 ADVREC
20 SF 25
21 GETREC
22 CF 25
23 AVIEW
24 PSE
25 X<0?
26 GTO G
27 GTO F
28*LBL H
29 RCLPT
30 INT
31 STO Y
32 4
33 MOD
34 X=0?
35 ISG Y
36 ""
37 X#0?
38 DSE Y
39 ""
40 X<>Y
41 SEEKPT
42 GETREC
43 AVIEW
44 RTN
45*LBL I
46 "IN"
47 FC? 00
48 "EX"
49 "TERNAL SLV"

50 AVIEW
51 0
52 TF
53 RTN
54*LBL J
55 RCLPT
56 INT
57 RCL X
58 4
59 MOD
60 -
61 3
62 X<>Y
63 +
64 SEEKPT
65 GETREC
66 AVIEW
67 X<> L
68 SEEKPT
69 RTN
70*LBL A
71 LET=
72 1
73 FS?C 22
74 GTO J
75 GTO 09
76*LBL B
77 LET=
78 2
79 FS?C 22
80 GTO J
81 GTO 09
82*LBL C
83 LET=
84 3
85 FS?C 22
86 GTO J
87 GTO 09
88*LBL D
89 LET=
90 4
91 FS?C 22
92 GTO J
93 GTO 09
94*LBL E
95 LET=
96 5
97 FS?C 22
98 GTO J
99 *LBL 09

100 CF 01
101 GETREC
102 43
103 ATOX
104 X#Y?
105 GTO 09
106 SF 01
107 REC+
108 GETREC
109 LASTb
110 MUTE
111 STO$
112 4
113 REC-
114 *LBL 09
115 LASTb
116 XROM "a^b"
117 2
118 ADVREC
119 GETREC
120 CHS
121 ADVREC
122 R^
123 MUTE
124 RDN
125 RDN
126 FC? 00
127 XROM "SV$+"
128 FS? 00
129 XEQ IND 08
130 LET=
131 0
132 RTN
133*LBL "a^b"
134 "a^b?"
135 PROMPT
136 FC? 22
137 E-99
138 FC?C 22
139 E
140 END

11:10AM 03/04
01*LBL "SV$+"
02 STO$
03 4
04 *LBL 01
05 FC? 01
06 GTO 00
07 EVALT

```

08 RCL\$	17 GTO 01	26 EVAL\$
09 8	18 *LBL 00	27 FS? 10
10 EVALZ	19 EVALZ	28 VIEW X
11 X<>Y	20 X<>Y	29 RCL\$
12 RCL\$	21 *LBL 01	30 4
13 4	22 EVALT	31 X#Y?
14 EVALT	23 T=Z?	32 GTO 01
15 RCL\$	24 RTN	33 END
16 8	25 "Y-Z*(Y-X)/(Z-T)"	

01*LBL "INIEQN"

02 "EQNS"	39 "POSROOT"	77 "F0=1/SQRT(LC)"
03 90	40 APPREC	78 APPREC
04 CRFLAS	41 "X1=(-B+SQRT(B^2"	79 "1/Q(b*c)-a"
05 "LINEAR"	42 -"4AC))/2A"	80 APPREC
06 APPREC	43 APPREC	81 "F0 L C"
07 "Y=AX+B"	44 "(#b+Q(b^2-4*a*c"	82 APPREC
08 APPREC	45 -"))/2/a-d"	83 "GAS EQUATION"
09 "c*a+d-b"	46 APPREC	84 APPREC
10 APPREC	47 "A B C X1"	85 "PV=NRT"
11 "X Y A B"	48 APPREC	86 APPREC
12 APPREC	49 "NEGROOT"	87 "c*(16629/2000)*"
13 "QUADRATIC"	50 APPREC	88 "b-a*b"
14 APPREC	51 "X2=(-B-SQRT(B^2"	89 APPREC
15 "Y=AX^2+BX+C"	52 -"4AC))/2A"	90 "P V N T"
16 APPREC	53 APPREC	91 APPREC
17 "c*a^2+d*a+e-b"	54 "(#b-Q(b^2-4*a*c"	92 "LIN. MOTION"
18 APPREC	55 -"))/2/a-d"	93 APPREC
19 "X Y A B C"	56 APPREC	94 "X=VT+1/2*AT^2"
20 APPREC	57 "A B C X2"	95 APPREC
21 "CUBIC"	58 APPREC	96 "c*b+(1/2)*d*b^2"
22 APPREC	59 "OHMS LAW"	97 -" -a"
23 "Y=X^3+AX^2+BX+C"	60 APPREC	98 APPREC
24 APPREC	61 "E=IR"	99 "X T V A"
25 "a^3+c*a^2+d*a+e"	62 APPREC	100 APPREC
26 APPREC	63 "b*c-a"	101 "NEWTONS LAW"
27 "X Y A B C"	64 APPREC	102 APPREC
28 APPREC	65 "E I R"	103 "F=G*M1*M2/R^2"
29 "4TH ORDER"	66 APPREC	104 APPREC
30 APPREC	67 "PARALLEL R"	105 "e*b*c/d^2-a"
31 "X]4+AX]3+BX]2+C"	68 APPREC	106 APPREC
32 -"X+D"	69 "1/R1=1/R2+1/R3"	107 "F M1 M2 R G"
33 APPREC	70 APPREC	108 APPREC
34 "e+a*(d+a*(c+a*("	71 "1/b+1/c-1/a"	109 CLA
35 -"b+a)))"	72 APPREC	110 SF#
36 APPREC	73 "R1 R2 R3"	111 9
37 "X? A B C D"	74 APPREC	112 END
38 APPREC	75 "RLC FREQ."	
	76 APPREC	

Appendix. AOS Simulator

Written by Greg McClure, this FOCAL program was first released in the GJM ROM and is added here for completion.

The AOS (Algebraic Operating System) program is designed to allow entry of data and operations using operations and parenthesis as written. The partial answers are saved in Extended Memory in a small file created by the user when AOS initializes. It follows operation hierarchy. So "(" and "*" are performed before "+", etc).

B.1 AOS Overview

The Algebraic Operating System emulator is designed to act like non-RPN calculators that use parenthesis and pending operations to solve numeric math operations. This program requires an Extended memory file (name AOS) to store data for pending operations for parenthesis operation. The program does not require any other memory except for the stack (which is fully used).

B.2 AOS Flag Usage

Flag	Use when set
0	+ pending (flag 1 MUST be clear)
1	- pending (flag 0 MUST be clear)
2	* pending (flag 3 MUST be clear)
3	/ pending (flag 2 MUST be clear)
4	^ pending
5	Open ('s pending

B.3 AOS User Keyboard

[A]: AOS +	[B]: AOS -	[C]: AOS *	[D]: AOS /	[E]: AOS ^
[F]: AOS ([G]: AOS)			[J]: AOS = (R/S)

B.4 AOS User Instructions

After XEQ "AOS" the AOS flags and AOS buffer will initialize. It will ask for the size of the Extended Memory file to use. If the AOS Data file already exists, it will ask for the new size. If no new size is given the data file is not resized. User mode will be enabled.

B.5 AOS Example

Usage of the AOS program is best served by a simple example.

Calculate $(1+2)*(3/4)+(5^{1/2})$

Enter	Keypress	Comments (and Annun.s)	Annunciators (red = on)	Output
	XEQ "AOS"	Reset AOS	01234	"SIZE?" (if no file) "NEW SIZE?" (if file)
20	R/S	Small array		0.0000
	F	(0.0000
1	A	1 +	01234	1.0000
2	G	2), + performed	01234	3.0000
	C	*	01234	3.0000
	F	(, * with value saved	01234	3.0000
3	D	3 /	01234	3.0000
4	G	4), / performed, * with value recalled	01234	0.7500
	A	+, * performed	01234	2.2500
	F	(01234	2.2500
5	E	5 ^	01234	5.0000
	F	(, ^ with value saved	01234	5.0000
1	D	1 /	01234	1.0000
2	G	2), / performed, ^ with value recalled	01234	0.5000
	G), ^ performed, + with value recalled	01234	2.2361
	J or R/S	= final + performed	01234	4.4861

In this example, after entering the final 2, instead of using G the final answer could have been calculated by entering J or R/S (J or R/S will perform all pending parenthesis and functions).

For those interested, the data file saves required values from the stack and the status of the flags every time the AOS "(" function is performed. It restores the flags and data values required back to the stack when AOS ")" is performed. The annunciators show which operations and how many stack registers will be stored (only one register is required for the operations saved).

B.6 Program Listing

Starts in next page...

01 LBL "AOS"

02 RAD	53 FS?C 01	105 XEQ 11
03 SF 27	54 -	106 FS? 00
04 "AOS"	55 FS?C 00	107 XEQ 11
05 SF 25	56 +	108 CLX
06 FLSIZE	57 RTN	109 X<>F
07 FS?C 25	58 GTO 09	110 XEQ 11
08 GTO 00	59*LBL A	111 R^
09 "SIZE?"	60 XEQ 12	112 RTN
10 PROMPT	61 SF 00	113 GTO 09
11 "AOS"	62 RTN	114*LBL 10
12 CRFLD	63 GTO 09	115 STO [
13 GTO 01	64*LBL B	116 CLX
14*LBL 00	65 XEQ 12	117 RCLPT
15 RCLFLAG	66 SF 01	118 DSE X
16 FIX 0	67 RTN	119 ""
17 X<>Y	68 GTO 09	120 SEEKPT
18 "NEW SZ <"	69*LBL C	121 X<> [
19 ARCL X	70 XEQ 13	122 GETX
20 ">?"	71 SF 02	123 X<> [
21 X<>Y	72 RTN	124 SEEKPT
22 STOFLAG	73 GTO 09	125 CLX
23 RDN	74*LBL D	126 X<> [
24 CF 22	75 XEQ 13	127 RTN
25 PROMPT	76 SF 03	128*LBL G
26 FC? 22	77 RTN	129 XEQ 12
27 GTO 01	78 GTO 09	130 RCLPT
28 CHS	79*LBL E	131 X=0?
29 RESZFL	80 XEQ 14	132 GTO 00
30*LBL 01	81 SF 04	133 RDN
31 CLST	82 RTN	134 XEQ 10
32 CLA	83*LBL J	135*LBL 00
33 SEEKPT	84*LBL 09	136 X<>F
34 X<>F	85 XEQ G	137 RDN
35 X<> L	86 FC? 05	138 ENTER^
36 +	87 RTN	139 ENTER^
37 XEQ F	88 GTO 09	140 ENTER^
38 XEQ G	89*LBL 11	141 FS? 00
39 GTO 12	90 SAVEX	142 XEQ 10
40*LBL 14	91 CLX	143 FS? 01
41 FS?C 04	92 +	144 XEQ 10
42 Y^X	93 RTN	145 FS? 02
43 RTN	94 *LBL F	146 XEQ 10
44*LBL 13	95 SF 05	147 FS? 03
45 XEQ 14	96 ENTER^	148 XEQ 10
46 FS?C 03	97 RDN	149 FS? 04
47 /	98 FS? 04	150 XEQ 10
48 FS?C 02	99 XEQ 11	151 R^
49 *	100 FS? 03	152 RTN
50 RTN	101 XEQ 11	153 GTO J
51*LBL 12	102 FS? 02	154 END
52 XEQ 13	103 XEQ 11	
	104 FS? 01	