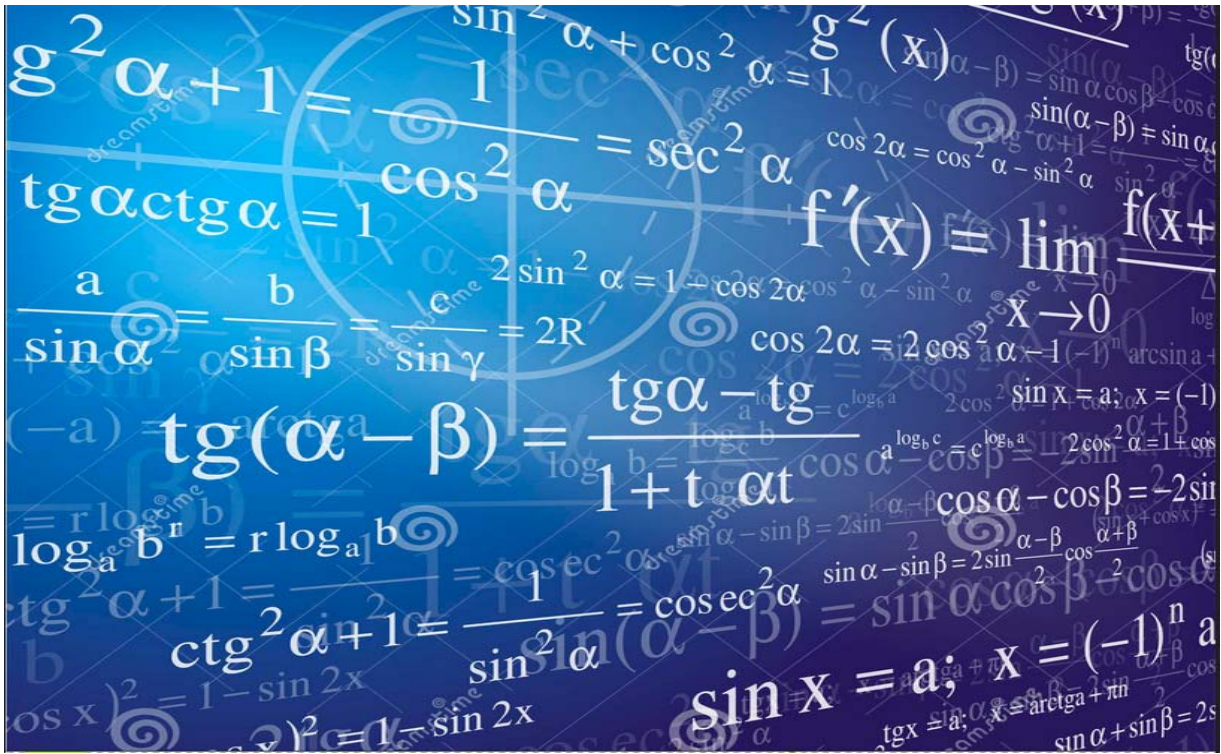


EQUATION SOLVER ROM

HP-41 Module



```

:A: :B: :C: :D: :E:
  USER      1

```

```

SELF-PR Y/N
  USER      1   PRGM

```

```

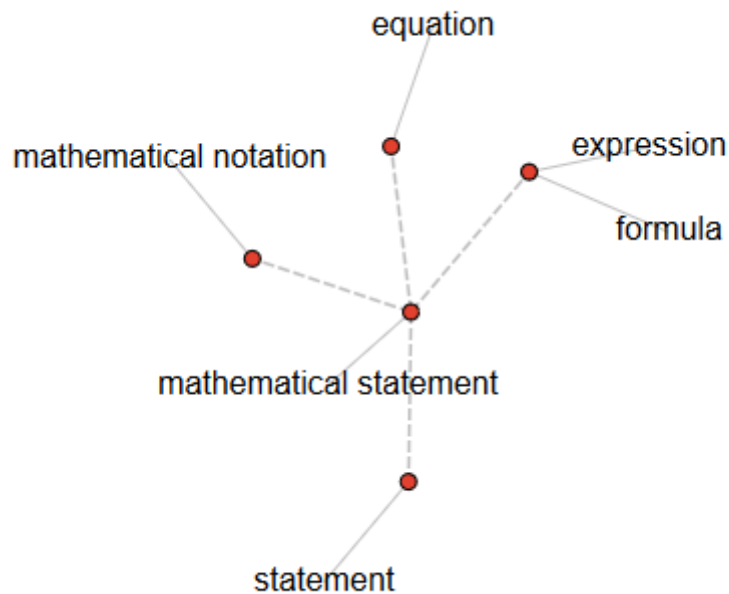
NO MVAR5
  USER      1

```

Written & Programmed by Ángel Martín
Revision 1-AB, February 2019

This compilation revision 1.1.1

Copyright © 2018-19 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 - 1AB

HP-41 Module

Table of Contents

1. Introduction	
a. From SOLVE to Solver	4
b. Scope, Intent and Dependencies	4
c. Module function Summary	5
2. Theory of Operation	
a. Variable Declaration	8
b. Program Editing vs. Running Modes	8
c. Building the Solver Program	9
d. Solving and Resolving	10
e. Tricks & Treats	12
f. A Look under the Hood	14
g. Mini Equation Library & Examples	15
3. Equation Library	
a. New Record Pointer Functions	18
b. Mark Fleming's Equation Library	19
c. A new twist to the old Solver	21
d. Show me the Money	22
Appendix. AOS Simulator	23

Handwritten mathematical equations on a piece of paper:

$$g_3(\xi) = \begin{cases} b_8 \exp(\mu_4 \xi) \\ b_5 \exp(\mu_1 \xi) + \frac{c_1}{2\mu_1 - 1} \left(\frac{\partial R_1}{\partial t_1} - 2iko \frac{\partial R_1}{\partial y_1} \right) \xi \exp(\mu_1 \xi) \\ b_6 \exp(\mu_2 \xi) + \frac{c_2}{2\mu_2 - 1} \left(\frac{\partial R_1}{\partial t_1} - 2iko \frac{\partial R_1}{\partial y_1} \right) \xi \exp(\mu_2 \xi) \\ \frac{c_1}{1} \left(\frac{\partial S_1}{\partial t_1} + 2iko \frac{\partial S_1}{\partial y_1} \right) \xi \exp(\mu_1 \xi) \end{cases}$$

$$\mu_{3,4} = (1 \pm d_0)/2, \quad d_0 = \sqrt{1 + 16k_0^2}$$

Equation SolverROM

Revision 1-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 in the Formula Evaluation manual, and for that you'll need to follow the syntax and other operation rules explained there with 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 **SLVS** 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 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.

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.

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-PM	ALPHA to Program	Text in ALPHA	Ángel Martin
02	A-PM7	ALPHA to Program (7 Chars)	Test in ALPHA	Ángel Martin
03	CLVAR\$	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	MVAR\$ _ _ _ _	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	VMENU	View Menu	Vars in Buffer	Ángel Martin
19	VMNU+	View Menu+	Vars in Buffer	Ángel Martin
20	?T=L	Test for equal values	Values in T, L	Ángel Martin
21	"#"	Auxiliary function	Data in program	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	AOS	AOS Simulator	FileName in ALPHA	Greg McClure
25	ARCLCHR	ARCL Character	FileName in ALPHA	Håkan Thörngren
26	READREC	Read Record to ALPHA	Data in Record	Ángel Martin
27	REC-	Move record one down	Pointer position	Ángel Martin
28	REC+	Move record one up	Pointer position	Ángel Martin
29	SEEK*	Seek record by X	FName in ALPHA, n in X	Ángel Martin
30	"APP\$"	Append Equation	To file "EQNS"	Mark Fleming
31	"APPEQN"	Append Equation	To file in ALPHA	Mark Fleming
32	"DELEQN"	Delete Equation	Removes four records	Mark Fleming
33	"EQNLIB"	Equation Library	Main Driver Program	Fleming - Martin
34	"INITEQN"	Initialize Library	Creates EQNS File	Mark Fleming
35	"SAR"	Search & Replace	Prompts for values	Mark Fleming
36	"SLVEQ\$"	Equation Solver Driver	Prompts for data	Greg McClure
37	"SV\$"	Solves for X	Equation in ALPHA	Ángel Martin
38	"TVM\$"	TVM equation	Prompts for inputs	Greg McClure
39	"CH2X"	Change to X	Text in ALPHA, char\$ in X	Mark Fleming
40	/+ /	Sums of Inverses	Values in X, Y	Ángel Martin
41	SIGMD	Sigmoid Function	Argument in X	Ángel Martin
42	3PMT _ _ _	3-Prompts Test	none	Ángel Martin

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: Standard set function names don't end with plus sign "+", whereas extended set functions do.

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.
- 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 letter 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'll 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.

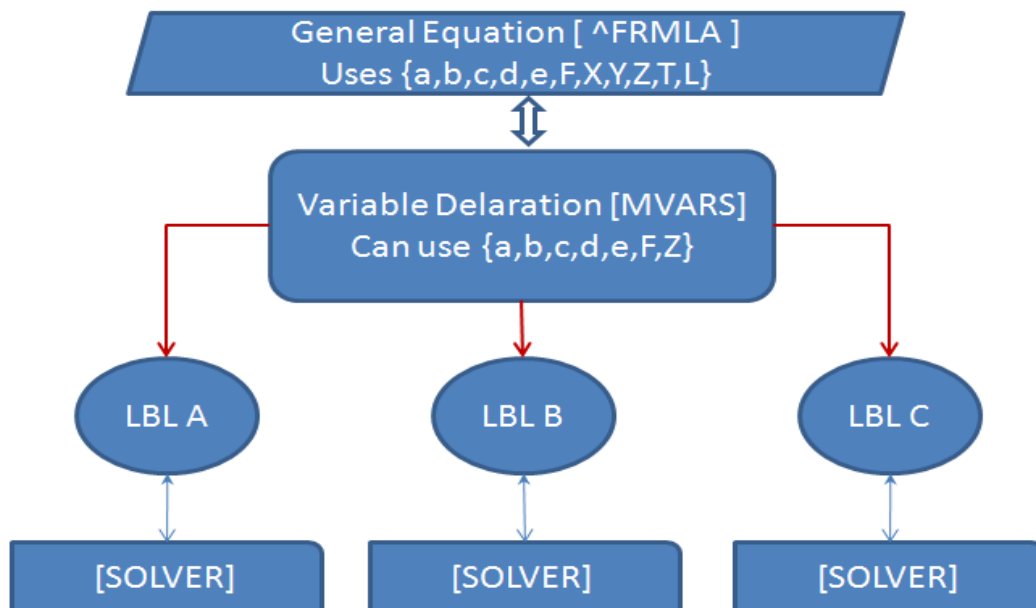
For example, **MVARS** plus "YZFC" will create the two program steps at the current location:

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

A word on writing the General Equation.

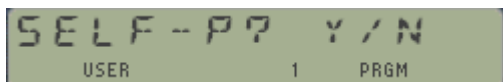
As you should know by now the variables available to the **^FRMLA** writing are the five stack registers and the six buffer registers, i.e. {X, Y, Z, T, L} plus {a, b, c, d, e, F}. Not all of these can be freely used in your general equation because the Solvers need the stack registers X, Y, T and L 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 all local labels, 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 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:

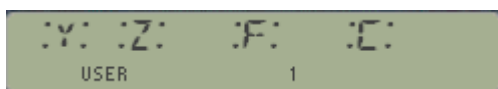
nn+2	<u>LBL A</u>	nn+6	SOLVER	nn+10	STOP
nn+3	SOLVER	nn+7	STOP	nn+11	<u>LBL D</u>
nn+4	STOP	nn+8	<u>LBL C</u>	nn+12	SOLVER
nn+5	<u>LBL B</u>	nn+9	SOLVER	nn+13	STOP

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 **SLV\$** way (using the secant method) and the **FROOT** way (using a combination of Newton and secant methods depending on the cases. 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 recurrent 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.



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 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 LBL "STD"	↔	20 LBL "XTD"	
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 LBL A		24 LBL A	
06 SOLVER		25 SOLVR+	
07 STOP		26 STOP	
08 LBL B		27 LBL B	
09 SOLVER		28 SOLVR+	
10 STOP		29 STOP	
11 LBL C		30 LBL C	
12 SOLVER		31 SOLVR+	
13 STOP		32 STOP	
14 LBL D		33 LBL D	
15 SOLVER		34 SOLVR+	
16 STOP		35 STOP	
17 LBL E		36 LBL E	
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 **SLV\$** or the Newton/Secant combination in **FROOT**. This is controlled by the status of User flag 01 when you press the “Solve for the Unknown” soft key:

- If UF 01 is Clear => Secant Method by **SLV\$**
- If UF 01 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 won't set the Data Entry Flag (!)
3. Note that after the solution for the unknown has been calculated, **it is *not* automatically stored by the program in the variable mapped to the menu letter**. You need to do that manually using the function **LET=** (also in the Formula Evaluation ROM). 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. If your formula wants to go beyond that boundary you cannot use the automated Solvers. Refer to the “Equation Library” for a work-around that allows these cases.

Finally, the Solvers use data registers {R00 – R04} and {R07 – R10} to store the **general** equation and the **muted** equation respectively. You should refrain from using them in the FOCAL programs 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 peak 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+**

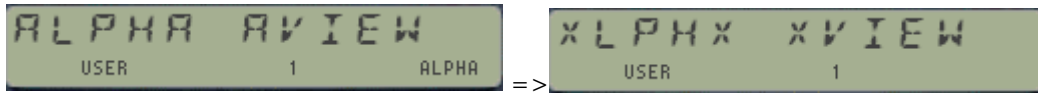
Here's a short description of their capabilities.

- **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 BLDRROM 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** is a global ROM access point to the muting process performed by SOLVER. This consists of replacing the letter used for the unknown with an "X" – so it is prepared for the **EVAL\$** instruction. In this generic form it is a prompting function, expecting the ASCII value of the character to mute in the prompt (in decimal). For example, using 65 as input will turn the string on the left to the one on the right:

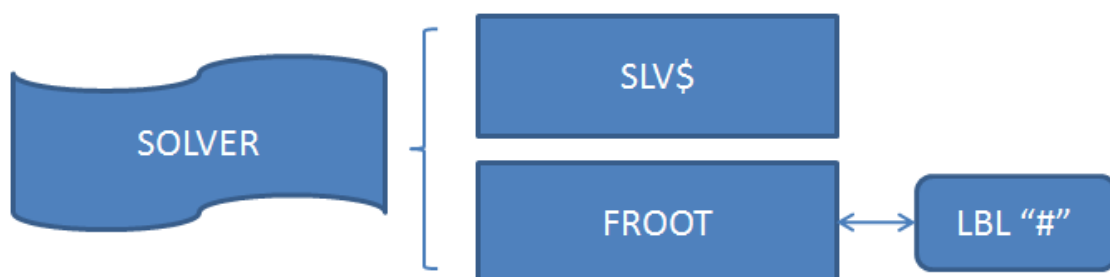


- **A-PM7** is the secret weapon used to insert any text string from ALPHA into program memory directly. **A-PM7** breaks the text in ALPHA in "chunks" up to 7-chars long, thus potentially will insert four text lines for 24 characters long text. This function is used internally by **MVAR\$** and **MVRS+**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

Do not confuse it 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 be required. You can use **A-PM** to enter the general equation as a program text line once it has been created in ALPHA by **^FRMLA**.

- **?T=L** is an auxiliary function that checks whether the values in the T and L 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:

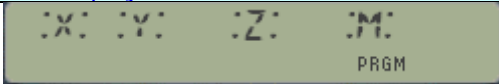
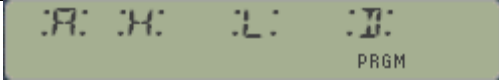
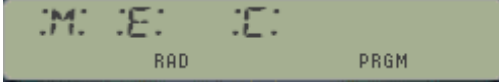
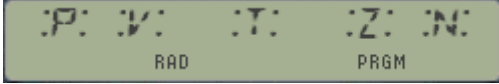

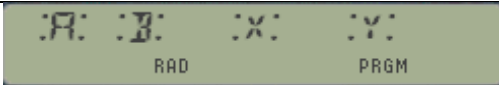

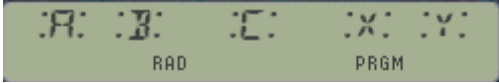
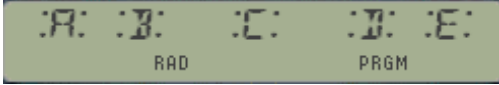
```
01 LBL "#"
02 RCL$ 07 ; brings the muted equation to ALPHA
03 EVAL$ ; evaluates the equation into X
04 END
```



Note that the Solvers don't use the stand-alone routine **SLV\$** 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	
KPL	Kepler Equation $E - ec \cdot \sin E = m$ 3 Variables, MVARS	
RGAS	Real Gas Equation $P \cdot V = Z \cdot N \cdot R \cdot T$ 5 Variables + 1 constant, MVARS	
VdW	Van-der-Waals Gas Equation $P + (a/Vm^2) = R \cdot T / (Vm - b)$ 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, MVRS+	

Looking at the code you can see that all the 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.

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

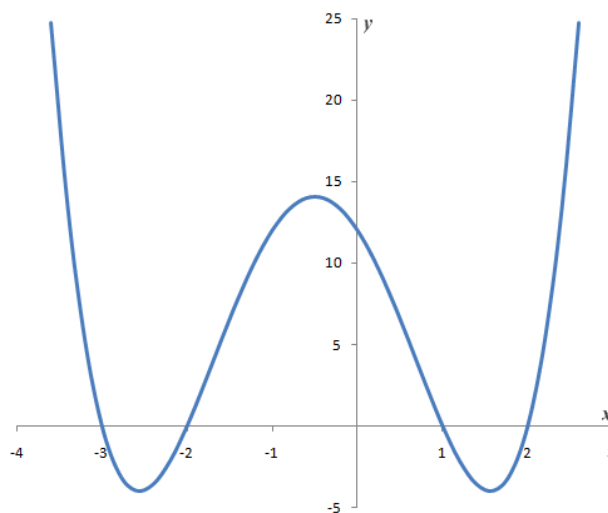
$$y = x^4 + A \cdot x^3 + B \cdot x^2 + C \cdot x + D \quad \text{becomes:} \quad F = E^4 + A \cdot E^3 + B \cdot E^2 + C \cdot E + D$$

Of these only the Van-der-Waals and the Polynomial Equations would 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** case as it's nothing more than a Cubic Equation "in disguise".

Generally 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 user flag 01 to select your choice of solvers: Clear for the internal case, Set for **FROOT**.

Numerical Examples.

- 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, and $x = 0.239123279$ for cubic.
- 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: $y = 2.236068$
- Catenary Equation - $d = H [1 - (1/\cosh(L/2a))]$
 Given $H = 42 \text{ m}, L = 100 \text{ m}, a = 43.5 \text{ m}$ and default $[a,b] = (0, 1)$
 Solves: $d = 17.814791 \text{ m}$
- Kepler Equation. - $E - ec. \sin E = m$
 Given $ec = 0.2$, and $m = 0.8$ and default $[a,b] = (0, 1)$
 Solves: $E = 0.964333888$
- 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)
- Van-der-Waals Equation - $P + (a/Vm^2) = R.T/(Vm-b)$
 Given $a = 14.66 ; b = 0.1226 ; P = 5 \text{ kPa}, T = 25^\circ\text{C}$, and $[a,b] = (1,5)$
 Solves: $Vm = 0.614322294 \text{ m}^3/\text{mol}$ (Warning: requires FROOT in the S&I ROM)
- Quartic Equation. - $F = E^4 + A.E^3 + B.E^2 + C.E + D$
 Given $a = 2; b = -7; c = -8; d = 12; F = 0$ and default $[a,b] = (0, 1)$
 Solves: $E = 1.0000$
 Can you find the other roots? Try changing the a^b initial interval...



Program Listing.

```

4:09PM 02/09
01 *LBL "KPL"
02 RAD
03 "b-c*S(b)-a"
04 MVARS
05 "MEC"
06 *LBL "Y=P1"
07 "a*c+b-d"
08 MVARS
09 "ABXY"
10 *LBL "Y=P2"
11 "a*d^2+b*d+c-e"
12 GTO 01
13 *LBL "Y=P3"
14 "d^3+a*d^2+b*d+c"
15 "-e"
16 *LBL 01
17 MVARS
18 "ABCXY"
19 *LBL "3DM"
20 "Q(a^2+b^2+c^2)-"
21 "d"
22 MVARS
23 "XYZM"
24 *LBL "CTRY"
25 "b*(1-(1/HC(c/2/"
26 "a)))-d"
27 MVARS
28 "AHLD"
29 *LBL "RGAŞ"
30 XEQ 00
31 "a*b-c*d*e*F"
32 MVARS
33 "PVTZN"
34 *LBL "VdW"
35 XEQ 00
36 "a+d/b^2-F*c/(b-"
37 "e)"
38 MVARS
39 "PVTAB"
40 *LBL A
41 SOLVER
42 STOP
43 *LBL B
44 SOLVER
45 STOP
46 *LBL C
47 SOLVER
48 STOP
49 *LBL D
50 SOLVER
51 STOP
52 *LBL E
53 SOLVER
54 STOP
55 *LBL 00
56 8.314459848
57 LET=
58 6
59 END

4:13PM 02/09
01 *LBL "Y=P4"
02 "e*(c+e*(b+e*(a+"
03 "e)))+d-F"
04 MVRS+
05 "ABCDEF"
06 *LBL A
07 SOLVR+
08 STOP
09 *LBL B
10 SOLVR+
11 STOP
12 *LBL C
13 SOLVR+
14 STOP
15 *LBL D
16 SOLVR+
17 STOP
18 *LBL E
19 SOLVR+
20 STOP
21 *LBL F
22 SOLVR+
23 END

```

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's not 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 straight forward method.

See below the program listing of Mark's EQNLIB using the new pointer functions:-

Mark Fleming's Equation Library - Program Listing

01 *LBL "EQNLIB"	51 AVIEW	101 *LBL 00
02 LKAOFF	52 E	102 99
03 "EQNS"	53 TOGF	103 XEQ 09
04 CLX	54 RTN	104 LET=
05 SEEKPTA	<u>55 *LBL J</u>	105 3
06 GETREC	56 RCLPT	106 RTN
07 AVIEW	57 INT	<u>107 *LBL D</u>
08 SF 27	58 RCLX	108 FC?C 22
09 RTN	59 4	109 GTO 00
<u>10 *LBL F</u>	60 MOD	110 LET=
11 CLX	61 -	111 4
12 4	62 3	112 GTO J
13 GTO 00	63 X<>Y	113 *LBL 00
<u>14 *LBL G</u>	64 +	114 E2
15 CLX	65 SEEKPT	115 XEQ 09
16 4	66 GETREC	116 LET=
17 CHS	67 AVIEW	117 4
18 *LBL 00	68 X<> L	118 RTN
19 ADVREC	69 SEEKPT	<u>119 *LBL E</u>
20 SF 25	70 RTN	120 FC?C 22
21 GETREC	<u>71 *LBL A</u>	121 GTO 00
22 CF 25	72 FC?C 22	122 LET=
23 AVIEW	73 GTO 00	123 5
24 PSE	74 LET=	124 GTO J
25 X<0?	75 1	125 *LBL 00
26 GTO G	76 GTO J	126 101
27 GTO F	77 *LBL 00	127 XEQ 09
<u>28 *LBL H</u>	78 97	128 LET=
29 RCLPT	79 XEQ 09	129 5
30 INT	80 LET=	130 RTN
31 STO Y	81 1	131 *LBL 09
32 4	82 RTN	132 "a^b?"
33 MOD	<u>83 *LBL B</u>	133 PROMPT
34 X=0?	84 FC?C 22	134 FC? 22
35 ISG Y	85 GTO 00	135 E-99
36 ""	86 LET=	136 FC?C 22
37 X#0?	87 2	137 E
38 DSE Y	88 GTO J	138 2
39 ""	89 *LBL 00	139 ADVREC
40 X<>Y	90 98	140 GETREC
41 SEEKPT	91 XEQ 09	141 CHS
42 GETREC	92 LET=	142 ADVREC
43 AVIEW	93 2	143 R^
44 RTN	94 RTN	144 XROM "CH2X"
<u>45 *LBL I</u>	<u>95 *LBL C</u>	145 FC? 01
46 FS? 01	96 FC?C 22	146 XROM "SV\$"
47 "IN"	97 GTO 00	147 FS? 01
48 FC? 01	98 LET=	148 XEQ IND 00
49 "EX"	99 3	149 RTN
50 "TERNAL SLV"	100 GTO J	<u>150 *LBL "CH2X"</u>

151 *LBL 02	163 AROT	06 X<>Y
152 ENTER^	164 RDN	07 EVALT
153 POSA	165 GTO 02	08 "Y-Z*(Y-X)/(Z-T)"
154 X<0?	166 *LBL 00	09 EVAL\$
155 GTO 00	167 RDN	10 FS? 10
156 AROT	168 END	11 VIEW X
157 ATOX		12 RCL\$
158 RDN	<u>01 *LBL "SV\$"</u>	13 7
159 "`X"	02 STO\$	14 X#Y?
160 E	03 7	15 GTO 00
161 -	04 *LBL 00	16 CLD
162 CHS	05 EVALZ	17 END

The following equations and auxiliary text lines are written to the ASCII file "EQNS" upon execution of the routine "INIEQN\$":

00	LINEAR	32	RLC FREQ.
01	Y=AX+B	33	F0=1/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 LAW3
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 B C D	47	F M1 M2 R G
16	POSROOT	48	INTEREST
17	X1=(-B+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-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

[A new twist to the Old Solver.](#) { **SLVEQ\$** }

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 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 sero
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	LKAON	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. few other functions are provided.

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

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: \$7,509.004951
 END: \$7,509.755401

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