Recursive Utilization of FINTG and FROOT.

Like the original SOLVE and INTEG did, both **FROOT** & **FINTG** support "crossed" nested calls from one another, i.e. you can call FROOT from an integrand function being used by FINTG, and you can call FINTG in the root-finding function definition for FROOT. However, it is not possible to recursively call either one of this functions sequentially from within a FOCAL routine. Any attempts to do so triggers the "*RECURSION*" error message and the execution aborts.

The SIROM provides a set of MCODE functions and two FOCAL routines to overcome this limitation. Each time FROOT/FINTG is executed it creates a dedicated memory buffer to store the application data and to perform all the math. The basis of the operation is the use of a secondary memory area for the nested call of the function, not conflicting with the initial memory buffer created in the first call. The main loop uses the initial buffer #14, and the operand function in turn creates a secondary buffer #14 to use for the nested loop – deleting it after it's complete.

In order to reuse the existing code, we'll trick the OS changing the id# of the initial buffer #14 right before the second call – *not deleting it but cloaking it in the I/O Memory area of the calculator*. The operand function re-labels the buffer with id#13 (using function **CLOAK**), then the nested call to FROOT/INTEG creates and uses a new buffer #14 to perform its task, and deletes it upon completion – returning the execution to the "operand" function FOCAL routine. Before the execution is returned to the driver program, the cloaked buffer is re-issued as id#14 (using function **EXPOSE**) so things can be picked up exactly where there were left off before calling the nested subroutine.

If you must know, all **CLOAK** and **EXPOSE** do is changing the buffer id#' of the initial buffer created in the first call to FROOT/INTEG - first from 14 to 13, and then back to 14. Prior to all this a third function (**RESET**) is used to check for pre-existing buffers with id#13 – deleting it if found.

RECURSION
USER RAD 1

2D Driver Routines and Rules of Engagement.

The main programs for double integrals and system of 2 equations are "**FITG2**" and "**FRT2**". Each one has an auxiliary routine associated with it, which acts as the first level operand functionand issues a second nested call for the integrand or the second equation appropriately, as follows:

For **FITG2**, the function name f(x,y) is expected in ALPHA, and the four integral limits in the stack in the pattern "y1, y2, x1, x2" – (y1,y2) for the outer integral, and (x1,x2) for the inner one.

- *The integrand function is to be programmed assuming x is in R01, and y in the stack.*

For **FRT2**, both function names are expected to be in Alpha separated by comma (like "F1,F2"), and the guesses entered in the stack, with the pattern "x1, x2, y1, y2" - with (x1, x2) for f1(x,y) and (y1, y2) for f(2(x,y).

- *The second operand function f2(x,y) is executed first. It assumes x in R01 and y in the stack.*
- *The first operand function f1(x,y) assumes x in R01 and y in R02.*
- *You decide which one is F1 and F2 by their order in the ALPHA string*

All buffer management is made automatically by the auxiliary routines "***2D**" and "***FG**".

Routine Listings.

Here are the routine listings for your perusal. Notably **FRT2** introduces more complexity to process the function names – entered as comma-separated strings in ALPHA – and due to the indirect call to f1(x,y) at the end of the auxiliary routine **"*FG"** - which is not required in the double integration case, as it's just one function involved. **CLAC** and **ASWAP** are borrowed from the ALPHA ROM – and need the Library#4 present in the calculator. They're only used for **FRT2**.

```
01  LBL "FRT2"
02  CLKEYS        no keys assigned
03  ASTO 00       save string
04  ASWAP         swap around ","
05  CLAC          remove second
06  ASTO 05       save in R05
07  CLA
08  ARCL 00       recall string
09  CLAC          remove second
10  ASTO 00       save in R00
11  STO 04        upper bound2
12  RDN
13  STO 03        lower bound2
14  RDN
15  RESET         reset buffers
16  "*FG"         first level operand
17  FROOT         call first round
18  RCL 02        y solution
19  X<>Y          arrange in stack
20  "|-,"         appends
21  ARCL 00       f1(x,y) name
22  ASWAP         swap around
23  RTN           done(!)
24  LBL "*FG"
25  STO 01        save x for later
26  CLOAK         mask buffer id#
27  RCL 03        lower bound2
28  RCL 04        upper bound2
29  CLA
30  ARCL 05       f2(x,y)
31  FROOT         nested call
32  EXPOSE        re-issue buf id#
33  STO 02        Save yo result
34  XEQ IND 00    calculates f1(x,Yo)
35  END
```

```
01  LBL " FITG2"
02  CLKEYS        no keys assigned
03  ASTO 00       save in R00
04  STO 03        upper limit2
05  RDN
06  STO 02        lower limit2
07  RDN
08  RESET         reset buffers
09  "2D"          first level operand
10  FINTG         call first round
11  RTN           done.
12  LBL "*2D"
13  STO 01        Save x for later
14  CLOAK         mask buffer id#
15  RCL 02        lower limit2
16  RCL 03        upper limit2
17  CLA
18  ARCL 00       f(x,y)
19  FINTG         nested call
20  EXPOSE        re-issue buf id#
21  END           ready
```

FINTG uses registers {R00-R03} and leaves the results in X and R01. The function name is left in ALPHA (6-chars max).

FRT2 uses registers {R00-R05} and leaves the results in the stack registers {X, Y} and {R01, R02} for the 2-equation roots. The comma-separated function names string is left in ALPHA (6-chars max for each name).

Comments.

The new functions to support the nested configuration are simplified versions of some general-purpose buffer utilities, available in other extension modules as follows:

- **RESET** is equivalent to the sequence { 13, **B?**, **CLB,** RDN }
- **CLOAK** is equivalent to the sequence { 14.013 , **REIDBF**, RDN}
- **EXPOSE** is equivalent to the sequence: { 13.014 , **REIDBF ,** RDN }

B? and **CLB** are available in the OS/X ROM, and **REIDBF** in the RAMPAGE ROM.

Using the simplified versions is more intuitive for math-oriented users, and besides freed up some space for additional examples in the SIROM.

While you can use **RESET** at any time (which will delete buff #13 if present, or do nothing if not present), using **CLOAK** and **EXPOSE** will generally result in the error message "*BUF ERR*". They're meant to be used only while buffer #14 exists, which is tightly controlled by the code in FINTG and FROOT – and furthermore, the SIROM uses the I/O_PAUSE interrupt as a "search & destroy" for buffer#14 at all times. Refer to the corresponding section in the **SandMath** manual to read more on this subject.

Caveat emptor:

There's a price to pay for this buffer trickery, and that's the loss of the USER key assignments. As you can see in the listings above, the main routines call **CLKEYS** to make the operation more reliable (this avoids spurious buffer errors due to memory overwrites). You can save them in an X-Mem file using **SAVEKA** and then recover them with **GETKA** after the fact (both functions are also available in the AMC_OS/X ROM).

These routines are not fast, their interest is in the methodology - not optimized for speed to say the least. If you need faster responses, then the SandMath provides dedicated MCODE functions for many of these and yet some more.

Bear in mind that the INTEG-based method to define special functions is not an efficient one from the mathematical standpoint, but it is a godsend for engineering problems. Also FROOT is not perfect or fool-proof either, so choosing a good initial guess is of high importance. If FRT2 fails to find a root (in either variable), it'll present the error message "NO ROOT" – Change the limits and try again.

The following examples should provide a good overview into the details of the programming.

Example 1. Calculate the integral of the Bessel Jn function, ITJ(1,3) = **INT** (0,3) { J(1,t).dt} using the integral definition as reference:

$$J_n(x) = \frac{1}{\pi}\int_0^\pi \cos(n\tau - x\sin\tau)\,d\tau.$$

Program Code is below. Note that you don't need to worry about the buffer management, that's done automatically by the driver routines all transparently to the user.

01	**LBL "ITJB"**		13	**LBL " *JN"**	*inner variable t in stack*
02	**X<>Y**	*order n to X*	14	**RAD**	*angular mode*
03	**STO 04**	*order saved in R04*	15	**RCL 04**	*get order*
04	**CLX**	*lower outer limit*	16	*****	*n.t*
05	**X<>Y**	*upper outer limit*	17	**X<>Y**	*inner variable t*
06	**0**	*lower inner limit*	18	**SIN**	*sin t*
07	**PI**	*upper inner limit*	19	**RCL 01**	*outer variable*
08	*"*JN"*	*function name*	20	*****	*x.sin t*
09	**XROM " ITG2"**	*double integration*	21	**-**	*n.t - x.sin t*
10	**PI**	*adjust factor*	22	**COS**	*cos (n.t - x.sin t)*
11	**/**	*final result*	23	**END**	*integrand complete.*
12	**RTN**	done.			

As mentioned before, *speed is not this method's forte*. Even on V41 in turbo mode it'll take a good 75 seconds to return 1.260052 (in FIX 6). This was not the goal of the example, but to clarify the general guidelines and showcase the conceptual approach. If you want a fast result you're encouraged to use **JBS** in the SandMath, or even better the **ITJ**(sub)function also in the SandMath, which uses the Generalized, Regularized Hypergeometric function for the calculation – a world of differences…

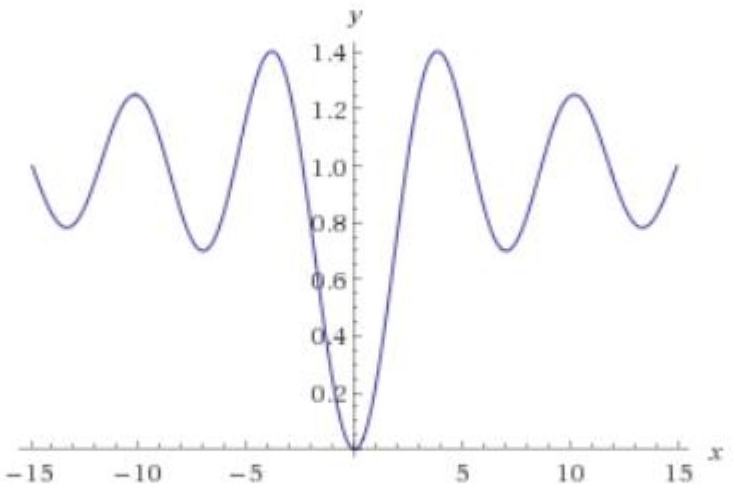
Comment. This particular example is of course much better dealt with using the well-known expression between the Bessel function J1 and J0 shown below (proving once again that it's always good to check your math before embarking in long and winding paths):

$$\int_0^x J_1(t)\,dt = 1 - J_0(x)$$

thus:

$$\int_0^3 J_1(t)\,dt = 1 - J_0(3) \approx 1.26005$$

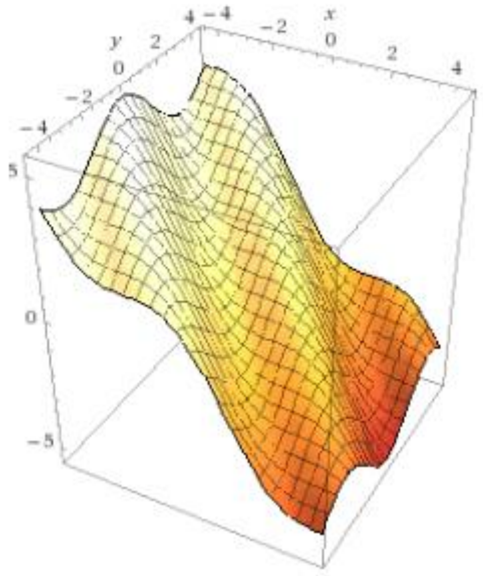
Here's an interesting plot showing the integral function of J1(x) between]-15 . 15[

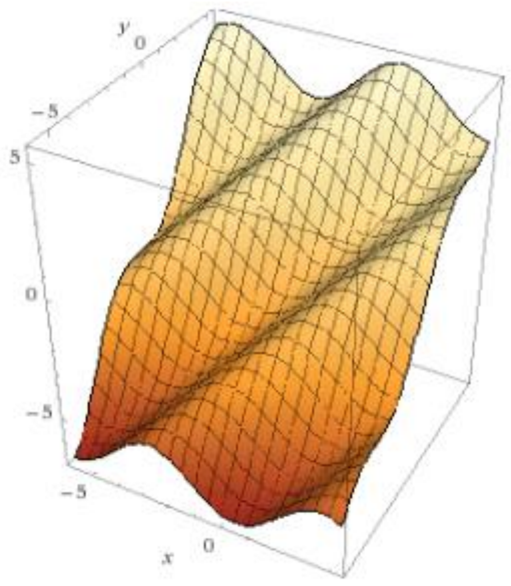


Example 2. Calculate the solution for the system of non-linear equations below:

$$\begin{cases} f1(x,y) = x - \sin(x + y) \\ f2(x,y) = y - \cos(x - y) \end{cases}$$

Solution: x = 0,935082064
y = 0,998020058





The equations are programmed as shown below. Note how the convention is observed, with the y value assumed in the stack for the second function and in R02 for the first one; whilst x is always assumed in R01 for both functions. The solutions are obtained in about 3 seconds (FIX 9) using V41 in Turbo mode.

ALPHA, "F1,F2" , ALPHA, 1, ENTER^, 2, ENTER^, 1, ENTER^, 2, XEQ "FRT2"

01	**LBL "F1"**		09	**LBL "F2"**	
02	**RCL 01**	*x*	10	**RAD**	
03	**RCL 02**	*y*	11	**CHS**	*-y*
04	**+**	*x+y*	12	**RCL 01**	*x*
05	**SIN**	*sin(x+y)*	13	**+**	*x-y*
06	**RCL 01**	*x*	14	**COS**	*cos(x-y)*
07	**-**	*sin(x+y)-x*	15	**X<>Y**	*y*
08	**RTN**		16	**-**	*cos(x-y)-y*
			17	**END**	

More Examples.

The table below lists the examples provided in the module, with their type of application and possible recursion/nested approaches.

Example	Description	Used Program	Application Type	Dimension
2DF1	Integrand example1	**2DITG**	2D Integration	Simple
2DF2	Integrand example2	**2DITG**	2D Integration	Simple
2DF3	Integrand example3	**2DITG**	2D Integration	Simple
F1XY	Integrand example1	**FITG2**	2D Integration	Recursive
F2XY	Integrand example2	**FITG2**	2D Integration	Recursive
F1, F2	Operands example1	**FRT2**	2-equation System,	Recursive
G1, G2	Operands example2	**FRT2**	2-equation System	Recursive
CI	Cosine Integral	**FINTG**	Single Integration	Simple
SI	Sine Integral	**FINTG**	Single Integration	Simple
ERF	Error Function	**FINTG**	Single Integration	Simple
FOURN	Fourier Coeffs.	**FINTG**	Single Integration	Simple
Q-1	Inverse Probability	**FROOT, FINTG**	Roots of integral	Nested
JNX	Bessel Function J	**FINTG**	Single Integration	Simple
JITX	Integral Bessel J	**FITG2**	2D Integration	Recursive
JZRN	Zeros Bessel J	**FROOT, FINTG**	Roots of integral	Nested

The integrand functions for these examples are direct transcriptions of the well-known definition formulas for the special functions, such as:

$$\mathrm{Si}(x) = \int_0^x \frac{\sin t}{t}\,dt \quad ; \quad \mathrm{Ci}(x) = \gamma + \ln x + \int_0^x \frac{\cos t - 1}{t}\,dt$$

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt. \quad ; \quad \Phi(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

For the Inverse Normal probability, the program solves for x using the definition of the cumulative probability based on the error function – which in turn is defined as an integral.

For the Fourier coefficients case, the function is expected to be periodic (but the generic period needn't be π), and centered around x=0. The formulas for T = π are given below:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)\,dx, \quad n \geq 0$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)\,dx, \quad n \geq 1$$

Example:- Calculate the first six coefficients for F(x) = x^ 2, assuming a period T=2π, centered in x0 = 0. As it's known, X^2 = 4/3 π^2 + SUM{ 4 cos(nx) /n^2 - 4π sin(nx) /n } |n=0,1,…

Using an accuracy of 6 decimal places:

```
01  LBL "FT"
02  X^2
03  RTN
```

a0 = 13,1595	b0 = 0
a1 = 4	b1 = -12,566
a2 = 1	b2 = -6,5797
a3 = 0,4444	b3 = -4,1888
a4 = 0,250	b4 = -3,1415
a5= 0,160	b5 = -2.513

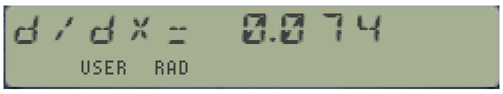
Other Programs included.

Besides the recursive routines the SIROM module includes a couple of FOCAL programs also geared towards the root finding and integration subjects.

The first program is "**SYS2**", original written by FJ Pamies Durá, (UPLE #35006) to solve non-linear systems of 2-equations. Here the operand functions assume x stored in R01 and y stored in R02 for both functions, therefore not entirely compatible with the requirements for the second function in **FRT2** –but it's easy to make a compatible version, just preamble the F2 routine with {RCL 02, ENTER^ } to make it compatible with **SYS2**, i.e:

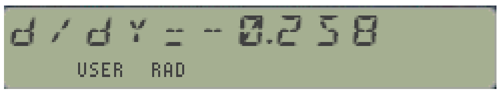
{ LBL "F22, RCL 02, ENTER^ , GTO "F2" } and:
{ LBL "G22, RCL 02, ENTER^, GTO "G2" }

This program is noticeable faster than **FRT2** because the iterative process follows a simultaneous approach on both variables (x,y) to approximate for the root, based on the two partial derivatives of the functions respectively. By contrast, on **FRT2** this is done on each variable independently, which in general will require more iterations and therefore longer execution times. It'll be akin to zigzagging versus doing a bee-line towards the root. It also shows the successive values of the approximations in the display:



```
d/dX=     0.074
    USER  RAD
```

,



```
d/dY= -0.258
    USER  RAD
```

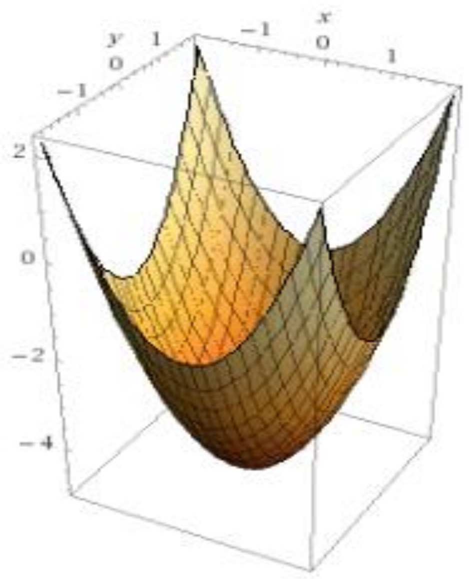
This program prompts for all the required data up front, in a "driver" arrangement. This includes the two function names, the accuracy (number of decimals), and the initial estimation (xo, yo) for the solution. **SYS2** uses data registers {R00 – R10}.

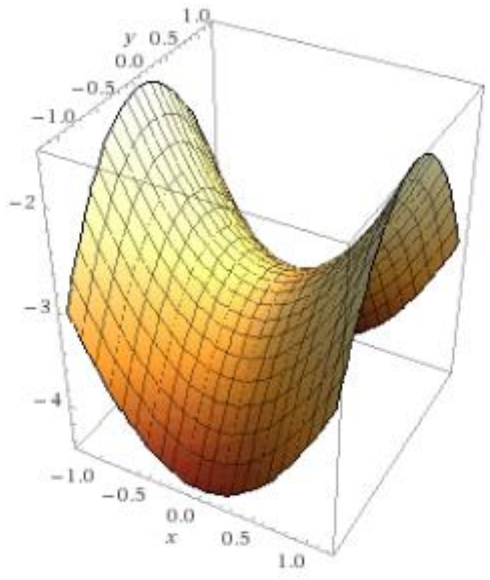
Example. Obtain the roots for the system of two equations below (available as "G1" and "G2")

$$\begin{cases} g1(x,y) = x^2 + y^2 - 5 \\ g2(x,y) = x^2 - y^2 - 3 \end{cases}$$

Solution: x = 2
y = 1

This is an interesting case because FRT2 not only is much slower (as we knew it was going to be), but also fails to find a root using initial guesses equal to the solutions, i.e. x0 = 2, y0=1.





The second program is "**2DITG**" – Valentín Albillo's neat example from DataFile for Double Integrals - that uses **FINTG** applied to the inner integral and with a 3-point, 5th. order Gaussian method for the outer integral. Three example routines (2DF1, 2DF2, 2DF3) are included in the ROM as follows:

$$I = \int_0^1 \int_1^2 (x^2 + y^2)\ .dy.dx \quad ; \quad I = \int_3^4 \int_1^2 1/(x+y)^2\ .dy.dx$$

$$I = \int_{-2.3}^{1.6} \int_{3.9}^{6.1} (e^{-x*x} + x^3 - y^3 * x^2 + 7) * \tan^{-1}(x-2) * \sin(y+3)\ .dy.dx$$

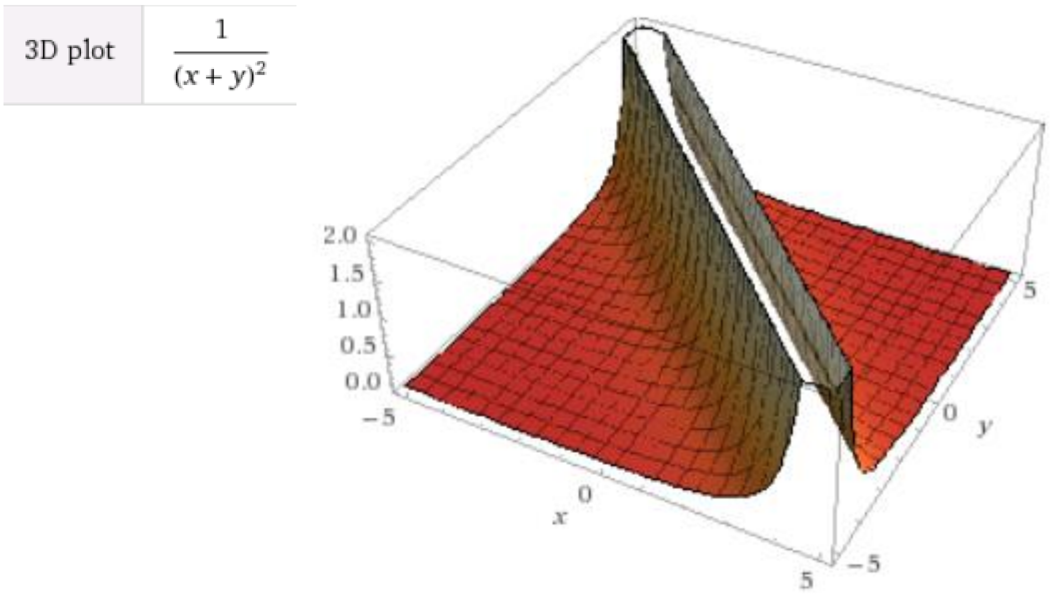
See the original article for details, available at:
http://web.archive.org/web/20110906135412/http://membres.multimania.fr/albillo/calc/pdf/DatafileVA024.pdf

In addition to the outer and inner limits in the stack (lower outer first, upper inner last) this program uses a parameter to specify the number of steps for the outer integration (a.k.a. the number of subintervals used). This is expected to be in R10 before executing the program – make sure you provide the value prior to executing 2DITG. The accuracy and execution times are directly impacted by the display settings (by virtue of FINTG) and the number of steps.

If you want to use **2DITG** with your own functions, the integrand routines expect x in R01 and y in the X register (as per **FINTG** conventions).

The results are:	I1 = 8/3 = 2.6666666
	I2 = Ln(25/24) = 0.040821
	I3 = 1321.275779

Note that because the data registers convention is the same for both main programs, you can use 2DF1, 2DF2 and 2DF3 as examples for **FITG2** – and conversely use F1XY and F2XY as examples for **2DITG**. In general, the execution times are very similar (and long) using M=10 for number of steps.



Finally, there is the PPC Solve routine, a classic must-have in everybody's toolbox. It is labeled "**SLV**" here, and the only difference is that it expects the function name in Alpha (no need to save it in R06 before). **SLV** uses data registers {R06-R09}. Set F10 if you want to see the iterations. Refer to the PPC ROM manual for details on SV, which also includes numerous examples of utilization.

Other References.

There was no more room available in this module to include more applications or examples, but many more fancy programs to calculate single and multiple integrals are available in the **INTEGRATOR** module, written in collaboration with Jean-marc Baillard and including a small arsenal of tools for the task. And yet more integration programs are available in the **SwapMath** module, written by Sherman Lowell and Ernest Gibbs and published in PPCCJ V11N7 p18 and PPCCJ V8N4 p31respectively – you're encouraged to refer to their QRG's for further details.

If what suits your fancy is systems of non-linear equations instead, you're encouraged to check the **NONLINEAR** module, which picks up where this module leaves off, including JM Baillard's programs to resolve systems with two, three and general case with "n" equations.