Notes on the back story of this letter:

I sent this *13-page* letter to ***Richard Nelson*** in order to submit as many of my materials as possible before I left home for many months in a few weeks. After reminding him of my latest (and yet unpublished) contributions, I proceeded to describe the new ones attached to this letter, namely:

> (1) A very short and fast ***NxN Matrix Inversion*** program for the ***HP-41C***, able to invert square matrices from 1x1 up to 16x16 (256 elements) in ~10"-30 min., i.e. *1.5x-2x* faster than the similar program in *HP*'s (dreadful) *Math Pac*, which furthermore was limited to 14x14 matrices (196 elements.)
>
> (2) A ***Fourier Series - Harmonic Analysis*** program for the ***HP-41C***, which is the much more capable version (up to 141 harmonics at a time and *2x-3x* faster) of the same program for the ***HP-67***, also included with the letter.
>
> (3) A ***Fourier Series - Harmonic Analysis*** program for the ***HP-67***, able to quickly compute and store up to 10 harmonics at a time, while also computing the sum of squared errors and allowing the computation of projections for any given *x*, plus an automatic sweep capability.
>
> (3) A ***Function Integration (FI)*** subroutine & driver, submitted for its possible inclusion in the *PPC ROM* (needless to say, it wasn't). It's the shortest and fastest such routine, based on the 3-point *Gauss-Seidel* quadrature formula and thus providing *5th*-order accuracy using just 3 function evaluations per subinterval, greatly improving on *Simpson's Rule*'s mere 3rd-order accuracy. Several examples are featured, including a short program which programmatically calls this routine to quickly and accurately compute the volume of a given solid of revolution.

An extended discussion on the defeating of the ***HP-41C***'s **`PRIVATE`** feature follows and I include in the letter proper my own method to defeat it based on using the *synthethic* pair **`STO/RCL b`**, with all pertinent details and comments, finally asking *Mr. Nelson* if he knew about this possibility, and also asking him to send me a note should he feel interested. Of course, no note was ever sent, as expected.

Valentin Albillo, 13-06-2022

Richard Nelson
Editor, PPC Journal
2541 W. Camden Place
Santa Ana , CA 92704
U.S.A.

Valentin Albillo (4747)
Padre Rubio, 61 - 2º C
Madrid 29
SPAIN

Dear Richard: First of all, congratulations for your excellent work every month. I hope you enjoyed the programs I submitted last month (41c programs about linear equations, eigenvalues, systems of differential equations) , and I hope too to see them published one of this years. Here are some more for you to enjoy them , as I don't want to see you bored, you see ! . They are two 41c programs about Matrix inversion (notice how short and fast) and Fourier Series, and one 67/97 program about Harmonic Analysis & Synthesis (useful for electrical engineers) . Sorry, I can't send you the 67/97 program recorded in a card, as I have no 67/97 to do it.

Notice too the subroutine "FI" (Function Integration), included as an input for the Custom ROM . It is the shortest, fastest, more accurate routine I could think of: only 102 bytes, including interactive prompting, etc. I hope you like it (and the rest of the membership, as well) . It is stored on a single side of the mag card.

Reading the V7 N3 issue, I found the "FEEDBACK" column most interesting: it is about the possibility of defeating the PRIVATE feature. I agree with DAVID R. KAPLAN (3678) about the fact that the methods of defeating the PRIVATE should be kept in secret, but only to the "outside" world. Among us, the members of PPC, all this - "mystery" is quite ridiculous: PRIVATE is almost useless, and it is high time to talk about it, so that members who really want to protect their programs stop thinking about the PRIVATE feature to do the work. You also mentioned in V7 N3 P-2,3 a couple of programs that clear PRIVATE. I don't think any fancy program is needed at all: there is a most simple, straightforward method that does the same, and I don't believe a simpler method to be possible:

-all that is required is to have STO b, RCL b assigned to keys. Use a status card or whatever.
-before loading the PRIVATE program, do the following: GTO .. , switch to PRGM , press SIN , GTO .. , switch to RUN mode.
-load the PRIVATE program. Do not switch to PRGM mode. It is essential that you don't ever see the PRIVATE message in the display. Should you ever see it, go to another place of program memory where you can freely "see" program memory, see it actually (switch to PRGM mode, then back to RUN mode) and return to the top of the PRIVATE program using CATALOG 1 for instance.
-once you are at the top of the PRIVATE program (that is just the case if you have just loaded the program), in RUN mode, press RCL b .
-then, go to the part of program memory just before the PRIVATE program (where you loaded the SIN) , and delete the SIN (but don't delete the END). PACK , and switch to RUN mode. The contents of Rb remain in X. Press STO b , switch to PRGM , and there you are, you are viewing the top of the PRIVATE program . You may see it all using SST . Using BST, GTO .nnn, inserting or deleting anything, causes the program to become PRIVATE once more. Using SST , you may reach its END, then wrap-around to the beginning without trouble. Very important : if you pass a card thru the card reader, the program will be recorded onto the card, but this time, it will not be private, so you can record it, clear the PRIVATE version, load the UNPRIVATED version, then make any desired changes, . If you press BST while seeing the PRIVATE program, it will turn PRIVATE. Simply go to another place where PRIVATE is not active (use CAT 1), then re-store the number in X in STO b, and switch to PRGM. You should be able to see the program again.

This procedure basically consist in position yourself a single byte ahead the PRIVATE program. This "splits" its label, so the PRIVATE is not seen. Be careful not to insert any instruction while seeing a PRIVATE program: the program becomes PRIVATE, but the instruction is inserted (or deleted, if you pressed back-arrow).

Did you knew about it before ? It is really simple: only RCL b, STO b ! In fact, RCL b , STO b, are most useful. They allow the creation of any text using any characters standard or non-standard without any trouble. No "CODE","DECODE" programs, only STO b, RCL b . It couldn't be simpler! I am writing now an article about applications of STO b, RCL b, but I don't know if the private defeat method should be published or not. It depends on you. Please, send me a note if you think it may be published.

Nothing to add, thank you for everything:

Best regards,

41C - MATRIX INVERSION

This program finds the inverse of a general NxN matrix, where N ranges from 1 to 16, both limits included, using a non-gaussian method. Program has been optimized to be short (40 registers) without loss of convenience, and fast (a 16x16 matrix takes about 36').

The program is written so that a zero pivot will - cause no trouble: it is skipped, and the following pivot is tested. All zero pivots are remembered, (its location) and its corresponding interchange is performed later. This avoids most problems when dealing with unconvenient matrices. There is one insoluble case, however: if all - the pivots in the main diagonal are zero, the program - stops, showing a message of error (program generated). This is a very rare case, but may occur. See examples.

The method used is the interchange method : consider the system $Ax=b$, which has the same matrix A we are trying to invert. The vector b has n components, the same number that the solution vector x . The purpose of the method is to interchange a component of b by a component of x at a time. After n independents interchanges have - been performed, the papers of b and x are reversed, and the system is now $A^{-1}b = x$, where the matrix is the inverse of A.

The algorithm is as follows:

FOR k=1 to N
LET $a_{kk} = 1/a_{kk}$
LET $a_{ik} = a_{ik} \cdot a_{kk}$, $i = 1,2,\ldots,n$, $i \neq k$
LET $a_{kj} = a_{kj} \cdot (-a_{kk})$, $j = 1,2,\ldots,n$, $j \neq k$
LET $a_{ij} = a_{ij} - a_{ik} \cdot a_{kj} \cdot a_{kk}$, $i= 1,2,\ldots,n$, $i \neq k$
$j= 1,2,\ldots,n$, $j \neq k$
NEXT k

a special procedure takes place if $a_{kk}=0$. k is incremented by one, and flagged, so that it will be remembered as an interchange that needs already to be done. After a successful interchange has been performed, a search is done for the minimum k which remains undone. If no k is - found, the work is done. If no interchange is succesful, an error condition is generated. That only happens if all remaining pivots in the main diagonal are zero, a very infrequent case.

PROGRAM CHARACTERISTICS

The program has 170 lines, fits into 40 registers (let the final END of program memory be the END of the program), and requires SIZE N^2+7 , where N is - the order of the matrix. It is much faster than the MATH 1A module program "MATRIX". Comparative times are given below:

MI : : :	4''	20''	1'16''	9'12''	30'	36'
MATRIX :	10''	56''	2'49''	15'32''	(45')	(54')
order N:	1	3	5	10	15	16

- the times on brackets are extrapolations, as the MATRIX - program cannot solve more than 14x14 matrices .
- if you have: no RAM : up to 4x4 , 3 RAMs : up to 14x14
  1 RAM : id. 8x8 , 4 RAMS : id. 16x16
  2 RAMs : id. 12x12
- The program uses the flags 0 thru N-1, but this is not- apparent to the user, as the status of all flags is stored and recalled again before the program stops, using synthetic functions STO & RCL d , so the user has all - flags except flag 19 for his use.
- Synthetic functions using registers M,N,O,d are used to save 3 registers (so that no RAMs allows up to 4x4) and to restore the status of all flags after running.

-it includes fully alpha input & output prompts, including the error message. A detector of insufficient SIZE is also built-in. The flags annunciators of the display indicate which interchanges have been already performed.

INSTRUCTIONS : load the program

(1) XEQ "MI" → N=? , key in the order of the matrix
N R/S → A1,1=? , (if SIZE nnn appears, the present size is insufficient to invert - your matrix. XEQ SIZE nnn, then R/S)
input a_{11}

a_{11} R/S → A1,2=? , keep on introducing all matrix - ... ... elements up to a_{nn}
a_{nn} R/S → the program will begin to invert the matrix. The flag annunciators will turn on as the associated interchange is performed. Once all interchanges have been done,

	→ A11=its value	the inverted matrix repla-
R/S	→ A12=its value	ces the original one in
...	... ...	storage.
R/S	→ ANN=its value	
R/S	→ program stops	

(2) to reinvert the inverse (to test accuracy, for instance) , simply press R/S ,and you'll get the original matrix.

(3) for another case, goto step (1)

WARNINGS : program uses the following synthetic functions:
RCL d, STO d, RCL M, STO M, STO N, ST+ N, ST- N
ST+ O, STO O, RCL IND N, RCL IND O
they are used to save 3 registers, and to restore all flags

-if all pivots are zero along the main diagonal, the program stops with "ERROR" in the display, after restoring all flags. (The matrix may be singular, but it is not - neccesary). This is quite unfrequent.

-the program is not adapted to run with a printer. It uses PROMPT instead of AVIEW, so R/S is neccesary in order to output the elements of the inverse. This may be easily - changed if required, but remember that the printer slows down the execution speed significantly.

-flag 19 is used (not restored) to control INPUT/OUTPUT , so don't turn off the machine while I/O is taking place.

EXAMPLE : Invert this matrix :

$$A = \begin{pmatrix} 2 & 2 & 3 & 2 \\ 2 & 2 & 3 & 1 \\ 11 & 5 & 4 & 6 \\ 2 & 1 & 1 & -9 \end{pmatrix}$$

XEQ "MI" → N=? , this is a 4x4 matrix
4 R/S → A1,1=? , 2 R/S → A1,2=?
2 R/S → A1,3=? , 3 R/S → A1,4=?
2 R/S → A2,1=? , 2 R/S → A2,2=?
2 R/S → A2,3=? , 3 R/S → A2,4=?
1 R/S → A3,1=? , 11 R/S → A3,2=? , 5 R/S → A3,3=?
4 R/S → A3,4=? , 6 R/S → A4,1=? , 2 R/S → A4,2=?
1 R/S → A4,3=? , 1 R/S → A4,4=? ,-9 R/S →

the program starts to execute. Watch the flags annunciators. The 0 is on, as the first pivot is $a_{11}=2\neq 0$, so the interchange is done. However, the next annunciator that - turns on is the 2. This is because the new a_{22} is 0, so - it is skipped, and the next a_{kk}, which is a_{33} is not 0 , and the interchange is performed. AFTER THAT, the a_{22} is tested again, and it remains to be 0, so the a_{44} is tried, and as it is $\neq$ 0, the interchange takes place, so the 3 turns on. Then, the a_{22} is tested once more, it is found $\neq$ 0, and the last interchange is done, the 1 is on, and as no interchange remains, the work is done finally. The flags are restored, and the inverse is output:

→ A1,1=70.0000 , R/S → A1,2=-71.0000 , R/S →
→ A1,3=-1.0000 , R/S → A1,4=7.0000 , R/S →

→ A2,1=-252.0000 , R/S → A2,2=255.0000 , R/S →
→ A2,3=4.0000 , R/S → A2,4=-25.0000 , R/S →
→ A3,1=121.0000 , R/S → A3,2=-122.0000, R/S →
→ A3,3=-2.0000 , R/S → A3,4=12.0000 , R/S →
→ A4,1=1.0000 , R/S → A4,2=-1.0000 , R/S →
→ A4,3=0.0000 , R/S → A4,4=2.0000E-11

so, the inverse is:

(t=41 seconds)

$$A^{-1} = \begin{pmatrix} 70 & -71 & -1 & 7 \\ -252 & 255 & 4 & -25 \\ 121 & -122 & -2 & 12 \\ 1 & -1 & 0 & 0 \end{pmatrix}$$

MATRIX INVERSION

```
01 LBL "MI"
02 FIX Ø
03 CF 29
04 CF 19
05 "N=?"
06 PROMPT
07 STO Ø5
08 X^2
09 6
10 +
11 SF 25
12 RCL IND X
13 FS?C 25
14 GTO 99
15 1
16 +
17 "SIZE "
18 ARCL X
19 PROMPT
20 LBL 99
21 7
22 STO Ø4
23 RCL Ø5
24 1 E3
25 /
26 1
27 +
28 STO ØØ
29 STO Ø2
30 STO Ø3
31 LBL 1Ø
32 FIX Ø
33 "A"
34 ARCL Ø2
35 "⊦,"
36 ARCL Ø3
37 "⊦="
38 FIX 4
39 FS? 19
40 ARCL IND Ø4
41 FC? 19
42 "⊦?"
43 PROMPT
44 FC? 19
45 STO IND Ø4
46 ISG Ø4
47 X()Y
48 ISG Ø3
49 GTO 1Ø
50 RCL ØØ
51 STO Ø3
52 ISG Ø2
53 GTO 1Ø
54 FS?C 19
55 RTN
56 1.ØØ1
57 -
58 STO ØØ
59 STO Ø1
60 RCL d
61 STO M
62 Ø
63 STO d
64 LBL 91
65 FS? IND Ø1
66 GTO 9Ø
67 RCL Ø1
68 RCL Ø1
69 RCL Ø5
70 *
71 +
72 7
73 STO Ø6
74 STO O
75 +
76 RCL IND X
77 X=Ø?
78 GTO 9Ø
79 1/X
80 STO IND Y
81 STO Ø4
82 X()Y
83 RCL Ø1
84 ST+ O
85 -
86 RCL ØØ
87 STO Ø3
88 STO Ø2
89 +
90 STO N
91 LBL Ø4
92 RCL Ø2
93 RCL Ø1
94 X=Y?
95 GTO Ø7
96 RCL Ø3
97 X=Y?
98 GTO Ø6
99 RCL IND N
100 RCL IND O
101 *
102 RCL Ø4
103 *
104 ST- IND Ø6
105 LBL Ø6
106 1
107 ST+ Ø6
108 ST+ N
109 ISG Ø3
110 GTO Ø4
111 RCL Ø5
112 ST- N
113 LBL Ø2
114 ST+ O
115 RCL ØØ
116 STO Ø3
117 ISG Ø2
118 GTO Ø4
119 GTO 14
120 LBL Ø7
121 RCL Ø5
122 ST+ Ø6
123 GTO Ø2
124 LBL 14
125 STO Ø6
126 RCL Ø5
127 ST* Ø6
128 RCL Ø1
129 ST+ Ø6
130 *
131 +
132 7
133 ST+ Ø6
134 +
135 STO Ø2
136 LBL ØØ
137 RCL Ø1
138 RCL Ø3
139 X=Y?
140 GTO ØØ
141 RCL Ø4
142 ST* IND Ø6
143 CHS
144 ST* IND Ø2
145 LBL ØØ
146 RCL Ø5
147 ST+ Ø6
148 ISG Ø2
149 X()Y
150 ISG Ø3
151 GTO ØØ
152 SF IND Ø1
153 RCL ØØ
154 STO Ø1
155 LBL 13
156 FC? IND Ø1
157 GTO 91
158 ISG Ø1
159 GTO 13
160 RCL M
161 STO d
162 SF 19
163 GTO 99
164 LBL 90
165 ISG Ø1
166 GTO 91
167 RCL M
168 STO d
169 "ERROR"
170 PROMPT
171 .END.
```

Status : 170 lines / 40 registers / SIZE N^2+7 / FIX 4

Registers: 00= 0.00(N-1) , 01= k , 02= i , 03= j
04= pivot , 05= N , 06= aux. , 07= a_{11}
... ... , $N^2+6=a_{nn}$, M = flags, N = aux.
O = auxiliar

Speed : $t = 3.37 - 0.08\,N + 0.33\,N^2 + 0.52\,N^3$ in seconds
N=5 , 1 m 16 s ; N=10 , 9 m 12 s ; N=16 , 36 m

VALENTIN ALBILLO (4747)

The purpose of this program is to determine - 2N+1 constants a_p (p=0,1,...,N) and b_p (p=1,2,...,N) in - such a way that the equations

$$f_n = \tfrac{1}{2}a_o + \text{SIGMA}(p=1,N)\ \left(a_p \cos\frac{2PInp}{2N+1} + b_p \sin\frac{2PInp}{2N+1}\right)$$

where n = 0,1,...,2N

are satisfied, where f_n are given numbers. The f_n may be thought of as the values of a function $f(x)$ at the points

$$x_n = \frac{2PIn}{2N+1} \quad (n = 0,1,\ldots,2N)$$

In other words, this program performs the Fourier analysis of a given set of N data points: finds all required harmonics (a_k,b_k) and, optionally, prints them without the user´s presence being ever required.

The coefficients a_k,b_k are given by:

$$a_p = \frac{2}{2N+1}\ \text{SIGMA}(n=0,2N)\ \left(f_n \cos\frac{2PInp}{2N+1} \right)$$

$$b_p = \frac{2}{2N+1}\ \text{SIGMA}(n=1,2N)\ \left(f_n \sin\frac{2PInp}{2N+1} \right)$$

The method used by this program, evaluates a_p,b_p using a - single trigonometric function calculation, namely for the values

$$\sin\frac{2PI}{2N+1} \quad \text{and} \quad \cos\frac{2PI}{2N+1}$$

no other values are computed, which results in faster execution times. All other trigonometric values are computed using a recursive procedure. The algorithm is as follows:

INPUT N , f_n FOR N = 0,1,...,2N
$C_1 = \cos 2PI/(2N+1)$, $S_1 = \sin 2PI/(2N+1)$
(1) p = 0 , $C_p = 1$, $S_o = 0$
(2) then, calculate for each p

$U_{2N+2} = U_{2N+1} = 0$

$U_n = f_n + 2C_pU_{n+1} - U_{n+2}$ (n=1,2,...,2N)

$a_p = \frac{2}{2N+1}(f_o + C_pU_1 - U_2)$, $b_p = \frac{2}{2N+1} S_p U_1$

OUTPUT a_p, b_p

IF p = N , stop. Else, $C_{p+1} = C_1C_p - S_1S_p$
$S_{p+1} = C_1S_p + S_1C_p$

replace p by p+1 and repeat step (2)

PROGRAM CHARACTERISTICS

This program is 133 lines long, it is - about 28 registers long, and requires SIZE N+8 , where N is the number of function values which will be used by the - Fourier analysis. Using all 4 m.modules, a maximum of 283 data points may be analyzed at once, to yield automatica - lly as many as 141 harmonics.

The program has a built-in SIZE detector that prompts you if the present SIZE is insufficient to run successfully your problem.

The program is both short and fast : due to the method used (only one evaluation of sin, cos), it is significantly faster than other programs. For instance, - to find a pair of harmonics of 5 given data, it takes 3 sec. and to find the same pair for 50 data, only takes 21 seconds

One important fact about the program is that , if you have a printer, you may go away while program computes the harmonics. All required harmonics (user´s selectable number of harmonics) will be printed, labeled and - spaced. This is because all data are introduced at once before any computation is carried away. This allows the user to input all N data, press R/S and go to have a meal, while the machine computes and prints all required harmonics. On the other hand, the MATH 1A pack program "FOUR" required the user to be present throughout the whole process, to - feed the data continuously, as the data were not stored - (but the computed harmonics were), while this program stores all data (but the computed harmonics are not). Besides, all data remain unaltered by the program calculations, so they can be used for any other purpose (such as other - kind of curve fitting or integration).

INSTRUCTIONS : Load the program (a single mag card)

(1) XEQ "FR" → N=? , key in the number of data points
n R/S → K=? , key in the number of harmonics that you want to compute (this is, the order of the higher harmonic that you want: if you chose K=1, a_0, b_0, a_1 & b_1 will be computed. This number K should be between 0 and INT(N/2), as a maximum of INT(N/2) are neccesary to give exact fit for all N data points. The - machine selects always the minimum between K and INT(N/2)

(If SIZE nnn appears, your present SIZE is insufficient to run the problem. XEQ SIZE nnn , then R/S)

K R/S → Y1=? , key in the value of the 1st data point
y_1 R/S → Y2=? , keep on introducing $y_2, \dots, y_n$
... ...
y_n R/S → a0=its value
R/S → b0=its value
R/S → a1=its value
R/S → a2=its value
... ...
R/S → ak=its value
R/S → bk=its value
R/S → program stops

- if you have a printer, all values will be automatically - printed and spaced. No R/S - are needed. Otherwise, press R/S to go on after writing - down each value. If you chose K harmonics, and after seeing the k-th harmonic you feel - that you need more, simply press R/S to proceed to the computation of the k+1-th harmonic pair. This may be repeated up to a maximum of INT(N/2) harmonics. The x values of the function are - not needed, as the harmonics do not depend on them, as - long as they are equally spaced.

EXAMPLES : Given the following data, find harmonics up to the second.

X ::	0	5	10	15	20	25	30
Y ::	1	3	4	2	0	6	5

XEQ "FR" → N=? , 7 data
7 R/S → K=? , up to 2nd
2 R/S → Y1=? , the 1st val.

1 R/S → Y2=? , 3 R/S → Y3=? , 4 R/S → Y4=? , 2 R/S → Y5=?
0 R/S → Y6=? , 6 R/S → Y7=? , 5 R/S → a0=6.0000
R/S → b0=0.0000

R/S → a1=0.5602 , R/S → b1=-0.7559
R/S → a2=-2.4408, R/S → b2=-0.7559 , R/S → -0.7559

so, we have : $a_0 = 6.0000$, $b_0 = 0.0000$
$a_1 = 0.5602$, $b_1 = -0.7559$
$a_2 = -2.4408$, $b_2 = -0.7559$

if we want now another harmonic, the 3rd order one:

R/S → a3=-0.1194 , R/S → b_3=0.7559 , R/S → 0.7559

so that: $a_3 = -0.1194$, $b_3 = 0.7559$

WARNING : remember that the x should be equally spaced, and that Y1 is always the y-value corresponding to the first x-value, nominally x = 0 . If the first x is not equal to 0 , a simple reordering of the y-values is - needed, based on the periodic properties of y.

FOURIER SERIES - HARMONIC ANALYSIS

```
01 LBL "FR"
02 RAD
03 CF 29
04 SF 21
05 FIX 0
06 "N=?"
07 PROMPT
08 STO 00
09 7
10 +
11 SF 25
12 RCL IND X
13 FS?C 25
14 GTO 04
15 "SIZE "
16 1
17 +
18 ARCL X
19 PROMPT
20 LBL 04
21 PI
22 ST+ X
23 RCL 00
24 /
25 1
26 STO 06
27 P-R
28 STO 01
29 X()Y
30 STO 02
31 "K=?"
32 PROMPT
33 RCL 00
34 2
35 /
36 INT
37 X)Y?
38 X()Y
39 1 E3
40 /
41 STO 05
42 RCL 00
43 LAST X
44 /
45 ST+ 06
46 0
47 STO 04
48 LBL 00
49 "Y"
50 ARCL 06
51 "⊦=?"
52 PROMPT
53 RCL 06
54 7
55 +
56 X()Y
57 STO IND Y
58 ISG 06
59 GTO 00
60 1
61 LBL 02
62 STO 03
63 0
64 STO 07
65 STO 06
66 RCL 00
67 7.008
68 +
69 LBL 03
70 RCL IND X
71 RCL 07
72 -
73 RCL 06
74 STO 07
75 RCL 03
76 *
77 ST+ X
78 +
79 STO 06
80 RDN
81 DSE X
82 GTO 03
83 ADV
84 SF 05
85 "a"
86 LBL 07
87 FC? 05
88 "b"
89 FIX 0
90 RCL 05
91 INT
92 ARCL X
93 "⊦="
94 FIX 4
95 RCL 06
96 FC? 05
97 GTO 05
98 RCL 03
99 *
100 RCL 08
101 +
102 RCL 07
103 -
104 GTO 06
105 LBL 05
106 RCL 04
107 *
108 LBL 06
109 ST+ X
110 RCL 00
111 /
112 ARCL X
113 AVIEW
114 FS?C 05
115 GTO 07
116 ISG 05
117 GTO 01
118 RTN
119 LBL 01
120 RCL 03
121 RCL 04
122 RCL 01
123 ST* Z
124 *
125 RCL 03
126 RCL 02
127 *
128 +
129 X() 04
130 RCL 02
131 *
132 -
133 GTO 02
134 .END.
```

Status : 133 lines / 28 registers / SIZE N+8

Registers:
00 = N , 03 = C , 06 = u_1 , 09 = y_2
01 = C_1 , 04 = S , 07 = u_2 , ...
02 = S_1 , 05 = P , 08 = y_1 ,N+7 = y_N

Máx. data :
0 RAM = up to 27 , 3 RAMs = up to 219
1 RAM = id. 91 , 4 RAMs = id. 283
2 RAMs = id 155 ,

Running time: t = 0.98 + 0.40 N per (a_k,b_k) (seconds)

N=5 , 3 sec. ; N= 10 , 5 sec. ; N = 30 , 14 sec ;

VALENTIN ALBILLO (4747)

Given N equally spaced data points - (x,y) that are samples of a periodic function, this program calculates and stores up to 10 harmonics (a_k,b_k) at one time. Any error during data input may be corrected, projections of y based on k harmonics can manually or automatically be performed over a given interval (linear sweep) The sum of squared errors for each value of k (no. of harmonics used to make projections of y) is also available. This is very useful to decide the number of harmonics which are needed to achieve a given accuracy. The RMS (root mean squares) may be computed from these data. All computed harmonics may be viewed at any time.

The program is relatively fast: it takes about 2 seconds per harmonic per point in calculation, and 3 seconds per harmonic in evaluation.

Theory : We have N data points (x_i,y_i), where the x_i are equally spaced: $x_i = 0, h, 2h, \ldots$

A linear substitution is applied to the x to yield x=0,1,2,... . Then:

if N is odd=2L+1 : $a_k=\frac{2}{2L+1}$ SIGMA(x=0,2L) $(y(x)\cos\frac{2PI}{N}kx)$

(k=0,1,...L) $\quad b_k=\frac{2}{2L+1}$ SIGMA(x=0,2L) $(y(x)\sin\frac{2PI}{N}kx)$

$\hat{y}(x)=\frac{1}{2}a_o$ + SIGMA(k=1,L) $(a_k\cos\frac{2PI}{N}kx + b_k\sin\frac{2PI}{N}kx)$

if N is even=2L : $a_k=\frac{1}{L}$ SIGMA(x=0,2L-1) $(y(x)\cos\frac{PI}{L}kx)$

(k=0,1,...,L) $\quad b_k=\frac{1}{L}$ SIGMA(x=0,2L-1) $(y(x)\sin\frac{PI}{L}kx)$

$\hat{y}(x)=\frac{1}{2}a_o$+SIGMA(k=1,L-1) $(a_k\cos\frac{PI}{L}kx + b_k\sin\frac{PI}{L}kx)+\frac{1}{2}C$

where $C=a_L\cos PI\,x$

The sum of squared errors, if N is odd is:

$S_{MIN\ k}$ = SIGMA(x=0,Nh) $(y(x)-\hat{y}_k(x))^2$ =
= $\frac{1}{2}(2L+1)$ SIGMA(k=M+1,L) $(a_k^2 + b_k^2)$

and very similarly is N is even.

INSTRUCTIONS : load program (1 card)

(1) Enter h (first nonzero value of x) & N (no. of data):

h ENTER N, A → x_{N-1} (prompting for the y_{N-1})

(2) Enter $y_i=y(x_i)$ values : y_i R/S → x_{i-1}

(3) repeat (2) until all y values have been entered. After entering y(0) :

y(0) R/S → a_o
→ (1) → a_1 → b_1
→ (2) → a_2 → b_2
... ...
→ (L) → a_L → b_L
→ 0.0000

the i is paused, then the a_i, b_i are printed or paused for you to copy them down. A maximum of INT(N/2) harmonics will be computed.

(4) optional: to redisplay harmonics: press B → a_o , etc

(5) to make projections of y : -with the maximum number of harmonics: enter x: x E → $\hat{y}(x)$

-with k harmonics:

enter k: k fE → k ; enter x: x E → $\hat{y}_k(x)$
(for another x, k need not be entered again)

- if automatic sweep is desired: enter no. of harmonics desired (if not the maximum) : k fE → k
enter initial x (=x_o) & increment: x_o ENTER inc D →
→ (x_i) → $\hat{y}_k(x_i)$ → etc. This is, the x are paused, and the y are printed. Then, another a & y ,

(6) to delete a mistake while introducing the y values,

assume the recently introduced $y(x_i)$ was an error:

then, press fA → last x value

the correct $y(x_i)$ R/S → x_{i-1} , etc

(8) To compute the sum of squared errors:
- -if fE has been previously used: press first fC to restore the maximum number of harmonics.
- -then , C → (k) → $S_{MIN\ k}$ → ... → (0) → $S_{MIN\ 0}$ → 0.0000

(9) to reset the number of harmonics to its maximum, for projections or otherwise, press fC → max. no.

(10) for another case, goto (1)

WARNINGS : - N may be any integer greater than 1
- h must not be 0

-if N is less than or equal to 19, program calculates INT(N/2) harmonics, which ensures exact fit. If N is greater than 19 , a maximum of 9 harmonics (up to a_9,b_9 included) will be calculated, due to storage limitations

-a_o is 2 times the mean value of y(x) over the period

-The subroutine to delete a mistake corrects any value any number of times , except y(0)

-the harmonics are displayed rounded to 4 decimals, but retain its full value in their respective registers.

-All computed harmonics will be displayed.

-The $S_{MIN\ k}$ is valid only if N is less than 20. Otherwise, some unknown constant should be added to each S_{MIN}. Of course, if N is less than 20, L=INT(N/2), then $S_{MIN\ L}$ is zero, because it is an exact fit, no error at all, so $S_{MIN\ L}$ =0 is not displayed. To achieve correct values, the stored number of harmonics must be the maximum L. This is always the case, except if you used fE. If in doubt, press fC to restore this number to the maximum. The values of S_{MIN} are useful to decide which minimum no. of harmonics give a certain accuracy. The RMS error is

RMS error = SQRT($S_{MIN\ k}$/N)

-If N is less than 20, $\hat{y}(x)$ is an exact fit. Otherwise, it is a 9-order least-squares trigonometric approximation.

EXAMPLE : Given the following data, find its mean value, all required harmonics to fit the data, and chose the number of harmonics needed to compute y(x) to an RMS error not greater than 0.6 . Predict y (12)

X	0	5	10	15	20	25	30
Y	1	3	4	2	0	6	5

-load the program: there are 7 data spaced by 5's :

5 ENTER 7 A → 30 ; 5 R/S → 25 ; 6 R/S → 20 ; 0 R/S → 15
2 R/S → 10 ; 4 R/S → 5 ; 3 R/S → 0 ; 1 R/S →

→ 6.0000 (a_o) → (1) → 0.5602 (a_1) → -0.7559 (b_1)
→ (2) → -2.4408 (a_2) → -0.7559 (b_2)
→ (3) → -0.1194 (a_3) → 0.7559 (b_3)
→ 0.0000

so, the mean value is $\frac{1}{2}a_o$ = 3.0000 . To chose the no. of harmonics required to fit the data, we compute the S_{MIN}:

press C: C → (2) → 2.0499 → (1) → 24.9015 → (0) → 28.0000
→ 0.0000

the RMS errors are: (2)=0.5411 ; (1)=1.8861 ; (0)=2.0000
so using harmonics up to 2nd order, we have RMS less than 0.6 . To predict y(12) :

-using all 3 harm.: 12 E → 3.7324 (=$\hat{y}$(12)
-using 2 harm: 2 fE → 2.0000 , 12 E → 3.7149 ($\hat{y}_2$(12))

VALENTIN ALBILLO (4747)

 FOURIER SERIES - HARMONIC ANALYSIS - DISCRETE DOMAIN

```
001 LBLA      31 25 11
002 CF 0      35 61 00
003 CF 2      35 61 02
004 RAD          35 42
005 CLREG        31 43
006 P()S         31 42
007 CLREG        31 43
008 STO E        33 15
009 STO 0        33 00
010 DSZ (i)      32 33
011 2               02
012 X()Y         35 52
013 /               81
014 PI           35 73
015 x               71
016 STO D        33 14
017 R down       35 53
018 STO C        33 13
019 GSB c     32 22 13
020 LBL 0     31 25 00
021 RCL 0        34 00
022 X=0?         31 51
023 SF 2      35 51 02
024 F? 0      35 71 00
025 RCL A        34 11
026 F? 0      35 71 00
027 CHS             42
028 F? 0      35 71 00
029 GTO 7        22 07
030 RCL C        34 13
031 x               71
032 R/S             84
033 RCL E        34 15
034 /               81
035 2               02
036 x               71
037 LBL 7     31 25 07
038 STO A        33 11
039 RCL B        34 12
040 ST I         35 33
041 LBL 1     31 25 01
042 RCL 0        34 00
043 RCL D        34 14
044 x               71
045 RC I         35 34
046 x               71
047 RCL A        34 11
048 P→R          31 72
049 STO+(i)   33 61 24
050 X()Y         35 52
051 P()S         31 42
052 STO+(i)   33 61 24
053 P()S         31 42
054 DSZ          31 33
055 GTO 1        22 01
056 RCL A        34 11
057 P()S         31 42
058 STO+ 0    33 61 00
059 P()S         31 42
060 F? 2      35 71 02
061 GTO B        22 12
062 1               01
063 F? 0      35 71 00
064 CLX             44
065 F? 0      35 71 00
066 CF 0      35 61 00
067 STO- 0    33 51 00
068 GTO 0        22 00
069 LBL B     31 25 12
070 P()S         31 42
071 RCL 0        34 00
072 P()S         31 42
073 DSP 4        23 04
074 RND          31 24
075 -X-          31 84
076 1               01
077 ST I         35 33
078 LBL 3     31 25 03
079 DSP 0        23 00
080 PSE          35 72
081 DSP 4        23 04
082 RCL (i)      34 24
083 RND          31 24
084 -X-          31 84
085 P()S         31 42
086 RCL (i)      34 24
087 P()S         31 42
088 RND          31 24
089 -X-          31 84
090 ISZ          31 34
091 RCL B        34 12
092 RC I         35 34
093 X(=Y?        32 71
094 GTO 3        22 03
095 CLX             44
096 RTN          35 22
097 LBL D     31 25 14
098 STO A        33 11
099 X()Y         35 52
100 LBL 4     31 25 04
101 PSE          35 72
102 GSB E     31 22 15
103 -X-          31 84
104 RCL 0        34 00
105 RCL C        34 13
106 x               71
107 RCL A        34 11
108 +               61
109 GTO 4        22 04
110 LBL E     31 25 15
111 RCL C        34 13
112 /               81
113 STO 0        33 00
114 RCL B        34 12
115 ST I         35 33
116 0               00
117 LBL 5     31 25 05
118 RC I         35 34
119 RCL 0        34 00
120 x               71
121 RCL D        34 14
122 x               71
123 ENTER           41
124 COS          31 63
125 RCL (i)      34 24
126 x               71
127 X()Y         35 52
128 SIN          31 62
129 P()S         31 42
130 RCL (i)      34 24
131 P()S         31 42
132 x               71
133 +               61
134 +               61
135 DSZ          31 33
136 GTO 5        22 05
137 P()S         31 42
138 RCL 0        34 00
139 P()S         31 42
140 2               02
141 /               81
142 +               61
143 F? 1      35 71 01
144 RTN          35 22
145 RCL E        34 15
146 2               02
147 RCL B        34 12
148 x               71
149 X=Y?         32 51
150 GTO 8        22 08
151 R up         35 54
152 RTN          35 22
153 LBL 8     31 25 08
154 LASTX        35 82
155 ST I         35 33
156 R up         35 54
157 RCL 0        34 00
158 PI           35 73
159 x               71
160 COS          31 63
161 RCL (i)      34 24
162 x               71
163 2               02
164 /               81
165 -               51
166 RTN          35 22
167 LBL C     31 25 13
168 SF 3      35 51 03
169 F? 1      35 71 01
170 CF 3      35 61 03
171 RCL B        34 12
172 ST I         35 33
173 0               00
174 LBL 6     31 25 06
175 RC I         35 34
176 1               01
177 -               51
178 DSP 0        23 00
179 PSE          35 72
180 DSP 4        23 04
```

181	R down	35 53	196	-X-	31 84	211	CF 1	35 61 01
182	RCL(i)	34 24	197	DSZ	31 33	212	RCL E	34 15
183	x^2	32 54	198	GTO 6	22 06	213	2	02
184	P()S	31 42	199	CLX	44	214	/	81
185	RCL(i)	34 24	200	RTN	35 22	215	ENTER	41
186	P()S	31 42	201	LBL a	32 25 11	216	INT	31 83
187	x^2	32 54	202	CF 2	35 61 02	217	X≠Y?	32 61
188	+	61	203	SF 0	35 51 00	218	SF 1	35 51 01
189	RCL E	34 15	204	1	01	219	9	09
190	x	71	205	STO+ 0	33 61 00	220	X()Y	35 52
191	2	02	206	GTO 0	22 00	221	X)Y?	32 81
192	F? 3	35 71 03	207	LBL e	32 25 15	222	X()Y	35 52
193	x^2	32 54	208	STO B	33 12	223	STO B	33 12
194	/	81	209	RTN	35 22	224	RTN	35 22
195	+	61	210	LBL c	32 25 13			

status : 224 steps / CF 0,1,2,3 / RAD / FIX 4

registers : 0=used , 1=a_1 , 2=a_2 , 3=a_3 , ... , 9=a_9
S0=a_0 , S1=b_1 , S2=b_2 , S3=b_3 , ... , S9=b_9
A=inc , B=≠ hr, C=h , D=$\frac{2PI}{N}$, E=N , I=index

LABELS :

f	DELETE		RESET		≠ HARM
	h N-INPUT	REVIEW	SMINK	AUTOEV.	X − Ŷ
	A	B	C	D	E

VALENTIN ALBILLO (4747)

- the following lines are to be considered as an input to your column "ROM PROGRESS" .

Reading past issues of PPCJ I noticed your request for some integration routine to be included in the PPC Custom Rom. As I am very fond of numerical analysis, I have tried many integration methods : Newton-Côtes, Simpson's rule, and - gaussian methods of all types. The result of this research may be resumed as follows:

a) An integration formula should have numerical coefficients as simple as possible: this saves storage registers, running time, or both.
b) It is far more convenient a simple formula which is applied to many subintervals than a , say, 20-degree formula applied once to the whole interval.
c) Very low order formulas are not convenient, as they yield a poor accuracy for reasonable amounts of running time.

After searching for a suitable method to be implemented in the PPC Custom Rom, I selected the 3-point gaussian method applied to a certain number of subintervals. The method is as follows:

we seek $\int_a^b f(x)dx$; the change of variable $x = \frac{1}{2}(b+a)+\frac{1}{2}(b-a)t$, $dx = \frac{1}{2}(b-a)dt$ transforms the interval (a,b) to $(-1,1)$. The 3-point formula gives now:

$$\int_{-1}^{+1} y(x)dx = \frac{8}{9}y(0) + \frac{5}{9}(y(SQRT(0.6))+y(-SQRT(0.6)))$$

don't be fooled by its very simple aspect: this method is an exact one if $y(x)$ is a polynomial of degree 5 or less. If - $y(x)$ may be approximated by such a polynomial, this method will give a fairly good approximation to the integral.

Notice the very important fact that the numeric constants are very simple (5,SQR(0.6),8,9) and that only 3 evaluations of $y(x)$ are needed to yield 5-order accuracy. On the other hand, the Simpson's rule has also simple coefficients, and perform 3 evaluations of $y(x)$ too, but it gives only 3-order accuracy, so the present method is nearly double exact.

The included routine, labeled "FI" (function integration) should be considered for its inclusion in the PPC Custom Rom.

```
01 LBL "FI"
02 "NAME?"
03 AON
04 PROMPT
05 AOFF
06 ASTO 00
07 "N?"
08 PROMPT
09 STO 01
10 "a↑b?"
11 PROMPT
12 LBL "FIP"
13 STO N
14 -
15 RCL 01
16 /
17 STO O
18 2
19 /
20 ST+ N
21 .6
22 SQRT
23 *
24 STO M
25 CLX
26 STO 02
27 LBL 00
28 RCL N
29 RCL M
30 +
31 XEQ IND 00
32 5
33 *
34 ST- 02
35 RCL N
36 RCL M
37 -
38 XEQ IND 00
39 5
40 *
41 ST- 02
42 RCL N
43 XEQ IND 00
44 8
45 *
46 ST- 02
47 RCL O
48 ST+ N
49 DSE 01
50 GTO 00
51 RCL 02
52 *
53 18
54 /
55 CLA
56 END
```

It is 56 lines, 102 bytes long . It requires SIZE 003 only. It uses synthetic functions: registers N,M,O are used as auxiliar ones, to help save 3 registers. They are cleared before the program termination, to avoid "garbage" in - the ALPHA register. The reason to use M,N,O is two-fold: to save 3 normal registers, and to increase speed. If synthetic functions can't be tolerated, simply change address M by 03, N by 04 and O by 05. Minimum size will be 006 ins - tead of 003. The line 55, CLA could be deleted then, and 9 additional bytes would be saved.

The routine uses no flags, and does not change angular mode, status or whatever, except the ALPHA register, which is used, then cleared. The routine may be called in two - different ways:

-an interactive way: call "FI" , and the program will prompt for the name of the function to be integrated, the required number of subintervals, and the integration limits, a,b. Then, proceed to compute the integral and return its value to the display.

-a non-interactive, programmable way: call FIP (FI programmable) , which assumes the following:

NAME is stored in 00 (name of function)
n, number subintervals is stored in 01 (no. of subinterv.)
lower limit , a , is in Y
upper limit , b , is in X (n should be integer ≥ 1)

then, proceeds to compute the integral, and returns control to your own program, with the integral value in the X register, (NAME remains unchanged, n is lost). No subroutine levels (apart from your LBL name) are used.

The routine is both, <u>short</u> and <u>very fast</u>. Some comparative examples are given: (the accuracy depends on n)

$\int_0^1 \sin(x^2)\,dx$, using LBL "FF" , x^2 , SIN , END
set RAD mode, if n=1 , I=0.310276885 (3'')
error=8 E-6
(the greater n, the better the accuracy) if n=5 , I=0.310268299 (13'')
error=3 E-9

$\int_0^{PI/2} \sin x\,dx$, n = 4 gives I=1.000000002 in about 10 seconds

$\int_0^1 SQRT(1+x^3)\,dx$, n= 2 gives I=1.111447947 in 7 seconds
error= 2 E-8

$\int_0^4 e^{-x^2}\,dx$, n=5 gives I=0.886226904 in 11 seconds, e=2 E-8

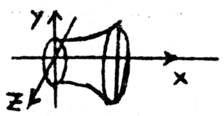
in contrast, Simpson's rule requires 32 (vs. 15) evaluations of f(x) to yield only 6 places.

As a final remark, this routine may be applicated for many - purposes: - Fourier series , elliptic (or whatever) integrals & functions, Normal distribution, Areas, lengths, volumes of solids, least-squares curve fitting for continuous data, etc.

As an example, a program to compute the volume of the solid of revolution generated by a user's specified curve is included. It is not for consideration to be included in the ROM, but only to show a possible aplication: (2 subinterv.)

01 <u>LBL"VOLUME"</u>	07 "AUX"	13 XEQ"FIP"	19 AVIEW
02 "NAME?"	08 ASTO 00	14 PI	20 RTN
03 AON	09 2	15 *	21 <u>LBL"AUX"</u>
04 PROMPT	10 STO 01	16 FIX 4	22 <u>XEQ IND 03</u>
05 AOFF	11 "X1↑X2?"	17 "VOLUME="	23 x^2
06 ASTO 03	12 PROMPT	18 ARCL X	24 END

for example, compute the volume of the solid of revolution obtained by the turn of the catenary y=½(3EXP(x/3)+3EXP(-x/3)) around the x axis, between x=0, x=1.2 :



-load the curve:

```
01 LBL"CATEN"
02 3
03 /
04 E↑X
05 ENTER
06 1/X
07 +
08 1.5
09 *
10 END
```

XEQ"VOLUME" → NAME?
CATEN, R/S → X1↑X2?
0 ENTER 1.2 R/S →
→ VOLUME=35.7976 (8 seconds)

(the volume is also in X, and to FIX 9, it is 35.79755410 , exact to almost 9 places)

VALENTIN ALBILLO (4747)