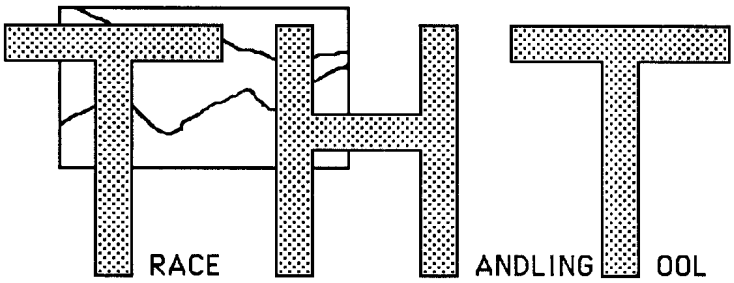
LRP 337/87



User Manual and Report (V 1.0 - october 87)

J.-M. Moret

1. Introduction

The data volume of a Tokamak is divided in shot files, each of them mainly containing information as a function of time. By analogy with the use of an oscilloscope, these functions are called traces. In the TCA Tokamak, there are between 200 and 300 such traces acquired by ADC's with local memory during the 3 second discharge sequence and then transferred to a PDP 11-60. The acquisition frequency of these traces is very variable, ranging from a typical 500 Hz to a few MHz; the number of points varies from 300 to a few thousands; this results in a total data volume per shot of the order of 200 kBytes.

This data volume has to be analysed. Software exists to display graphically the raw traces and standard combinations of these traces (multiplication by gains, standard derived functions). In the past, when more complex or less standard analysis was required, a particular program had to be rewritten, built and checked, leading to excessive compilation, linking and debugging times on a small PDP 11, together with a required operator expertise and time waste.

The aim of the software described in this paper is to solve this situation, giving the user the opportunity to perform even complex analysis without the need to rewrite a full Fortran program. Most of the programs previously written shared a large amount of code including the data retrieval and the graphic routines. The basic idea was to use a very versatile and easy to learn command set, which is interpreted by the program. Since most of the actual computing is done by the compiled subroutines, the interpreted nature of this package was not expected to lead to an excessive inefficiency.

Requirements to produce a good and general analysis tool lie in (a) a wide variety of powerful mathematical treatments covering the simple arithmetic operations, standard mathematical functions, digital signal treatment (filtering, integral and derivative, FFT), all acting on traces, i.e. arrays of values in time, (b) a very versatile graphic display including dividing the page into frames, choice of axis, title and comments, and (c) a userfriendly command syntax, easily readable, with the use of symbols and keywords, like a high level language.

A first attempt was made at a very low level to provide such a flexible tool: Experience showed that none of the above conditions were met satisfactory, and the problem was attacked from the start.

The resulting software was called **THT, Trace Handling Tool.**

2. Program description

The syntax of the command was chosen to be very close to that of Fortran, a widely known language among scientists and in which the corresponding subroutines were all written. The details of the syntax are described in Appendix 1. To give a general idea, an example of a **THT** application, together with the corresponding graphical output, are shown below for a real case.

2.1 Problem posed

The aim in the example shown is to plot a complex signal **COS** + i **SIN** in the complex plane and its amplitude and argument as a function of the variable **DEN**. The first part of the session initialises the different parameters needed by the subroutine **QDL**. In this subroutine, the graphic layout is described by calls to **PAGE**, **BOX** and **AXIS**. Then the requested shot file is open by **SHOT** and traces are fetched by **FUN**. Resampling between **T1** and **T2** is performed by **CLOCK**. Calculations mainly consist in extracting the amplitude and the argument from **COS** + i **SIN**. Finally the graphic is done by calls to **PLOT**; some values are displayed by **VALUE**.

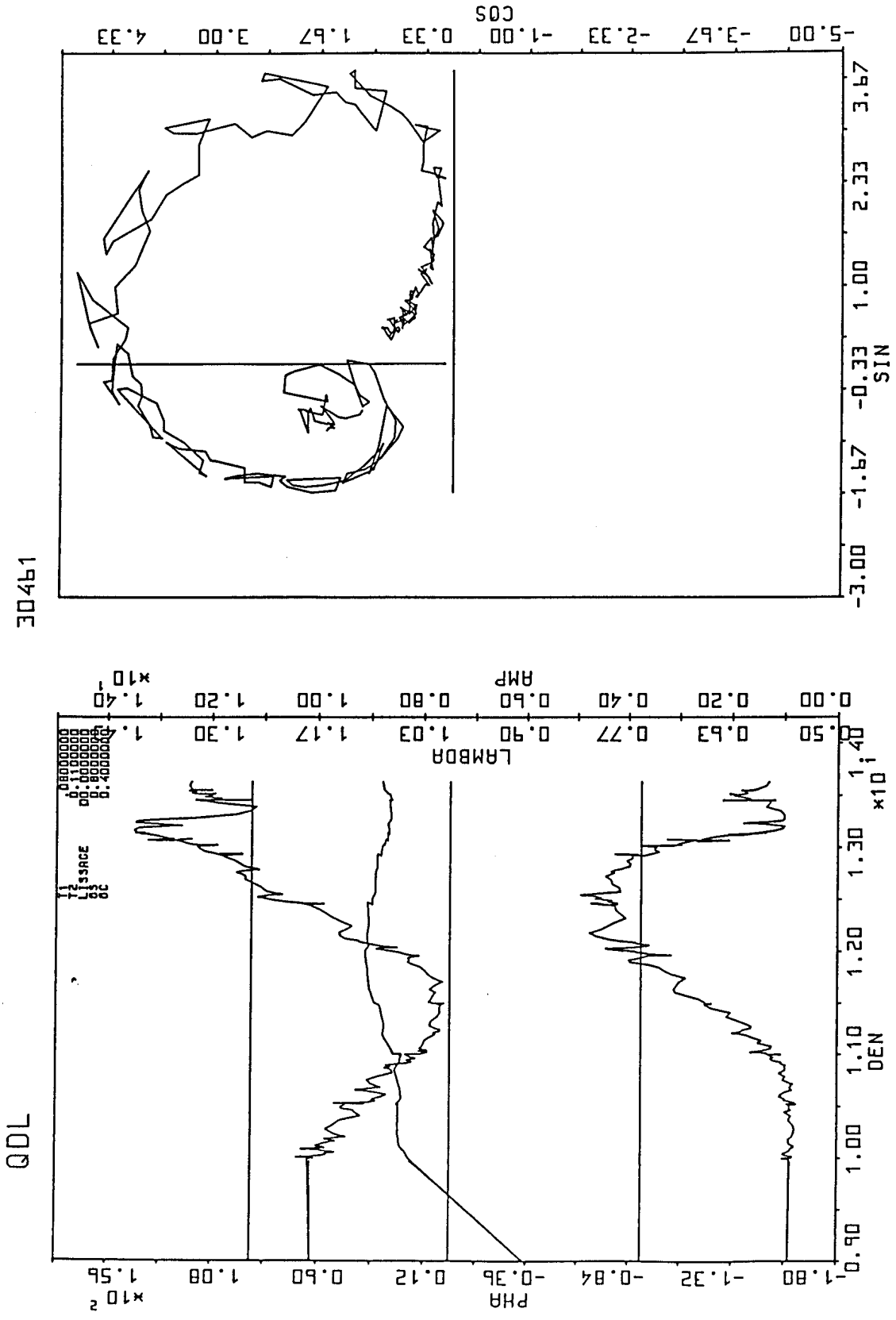
2.2 THT terminal session[1]

```
>THT
THT> NSHOT=30461
THT> T1=.08
THT> T2=.11
THT> OS=.8
THT> OC=.4
THT> @QDL
     ;GRAPHIC LAYOUT DESCRIPTION
     CALL PAGE(3,'QDL')
     CALL BOX(1,1211,204,205)
     CALL BOX(2,1214,204,203)
     CALL BOX(3,2214,201,202)
     CALL BOX(4,1213,204,19)
     CALL AXIS(201,-3,4,'SIN')
     CALL AXIS(202,-5,5,'COS)
     CALL AXIS(203,0,15,'AMP')
     CALL AXIS(204,9,14.25,'DEN')
     CALL AXIS(205,-180,-180,'PHA')
```

[1] in what follows, echo from the computer is typed in **bold characters**, and input from the user in outline characters, and out session comments in *italic*.

```
        ;FETCH AND RESAMP
        CALL SHOT(NSHOT)
        SIN=FUN(-214)
        COS=FUN(-215)
        CALL CLOCK((T2#T1)/SAMP(2),SAMP(2),T1)
        DEN=FUN(6)
        LAMBDA=FUN(14)
        ;CALCULATIONS
        C1=90
        C2=0
        C3=-90
        SIN=SIN#SIN[1]
        COS=COS#COS[1]
        PHASE=57.3*ATAN2(SIN,COS)
        AMP=(SIN*SIN+COS*COS)^.5
        ;PLOT
        CALL PLOT(COS,SIN,3)
        CALL PLOT(C2,SIN,3)
        CALL PLOT(COS,C2,3)
        CALL PLOT(DEN,C1,1)
        CALL PLOT(DEN,C2,1)
        CALL PLOT(DEN,C3,1)
        CALL PLOT(DEN,LAMBDA,4)
        CALL VALUE(T1,'T1',1)
        CALL VALUE(T2,'T2',1)
        CALL VALUE(OS,'OS',1)
        CALL VALUE(OC,'OC',1)
THT>    ^Z
>
```

2.3 THT graphical output



3. Note on usage

The **arithmetic expressions** are essentially identical to the form encountered in all high level language, with the same operation hierarchy, the use of the parenthesis and calls to generic functions, but the operations are performed on trace arrays. For example the algebraic expression

$$\sqrt{\frac{1 - \sin^2(x_i)}{1 + \cos^2(x_i)}}, \quad i = 1,...,n$$

is coded as

((1#SIN(X)^2)/(1+COS(X)^2))^0.5

This expression will be estimated for each element of the array **X**. If one of the variables was undefined, it is set to zero. An array element is specified by placing the index between square brackets, eg:

$f_i - f_1, \; i = 1,...,n$

is coded as

F#F[1]

The **assignment statement** has the form

THT>A = B + C

If the variable **A** does not exist, it is created. In this form all the elements of the array **A** are assigned. An index range of the array is specified by placing the index range in square brackets, separated by a colon, giving the following possibilities :

A=	updates **A** between indices 1 and n where n is the dimension of **A**
A[I1:I1]=	updates **A** between indices **I1** and **I2**
A[:I]=	updates **A** between indices 1 and **I**
A[I:]=	updates **A** between indices **I** and n

Different examples follow :

$$f_i = 2,\ i = i_1,\ldots,i_2$$

is coded as

```
THT>F[I1:I2] = 2
```

If the lower bound is not specified, the beginning of the array is assumed; if the upper bound is omitted, the end of the array is the default; eg:

```
THT>STEP = 0
THT>STEP [50:] = 1
```

will create a step function starting at index 50;

```
THT>DELTA = 0
THT>DELTA [50:50] = 1
```

will create a delta function at index 50.

The use of **generic procedure** is done by a procedure call statement, eg:

```
THT>CALL PLOT (T,TRACE,1)
```

will draw the array **TRACE** as a function of **T**. Each argument appearing in the list may be an expression, eg:

```
THT>CALL PLOT(T,TRACE+10,1)
```

Arguments are limited to being input parameters for the procedure or the function which do not export any result to the calling program section. A detailed list of the generic functions and procedures is given in Appendix 2.

The user can define his own **subroutines**, keeping in mind that the variables are global to the whole program as if the whole subroutine were inserted in line. As implemented on the PDP 11, the subroutine statements have to be stored in a separate file, and its execution is performed by the following command :

```
THT>@SUBROUTINE
```

where **SUBROUTINE.THT** is the name of the particular file containing the **THT**

subroutine statements. This file is simply created by a text editor with one statement per line. The execution returns to the calling program section when the end of the file is reached. A maximum nesting of 4 calling levels is implemented.

THT allows two **control structures**. The first one is the **conditional control**; eg:

```
THT>IF (CHOICE[1] :EQ: 1)
THT>        A = COS(OMEGA*T)
THT>EIF
THT>IF (CHOICE[1] :EQ: 2)
THT>        A = SIN(OMEGA*T)
THT>EIF
```

will create a sine or a cosine wave, according to the value of **CHOICE**. It must be specified that the test if performed on the first element of each expression appearing in the test, even if one of the expression is not a constant so that in the example the index specification in **CHOICE[1]** can be omitted. Also, since this control structure uses a single flag, nested conditional statements are not possible.

The second control structure is the **loop structure**, for example

```
THT>COEFF[1:1] = 1
THT>COEFF[2:2] = 2
THT>COEFF[3:3] = 1
THT>COEFF[4:4] = -1
THT>I=0
THT>F=0
THT>LOOP
THT>        I=I+1
THT>        F=F+COEFF[I]*X^(I#1)
THT>        IF (I :EQ: 4)
THT>                EXIT
THT>        EIF
THT>REPEAT
```

will estimate a given polynomial form for **F**. Note the statement sequence included between **LOOP** and **REPEAT** is stored in memory. Thus the loop control structure may not be nested, and there is a limit on the loop length. An exit test must **always** be present **inside** the loop.

4. The way THT works

This section explains the way **THT** interprets the commands and how it performs the operations. Each command line is converted in a sequentially executable list of operations. Most of these operations involve the use of a stack where data is temporarily stored. In order to be able to use a stack structure, the conversion to these sequential operations consists mainly of permutations of symbols or reordering of the commands, eg: all arithmetic expressions are converted into Reverse Polish Notation (RPN), stack based arithmetic language, eg: Helwett-Packard calculators. Details concerning this conversion are listed in Appendix 3, and some examples are given here.

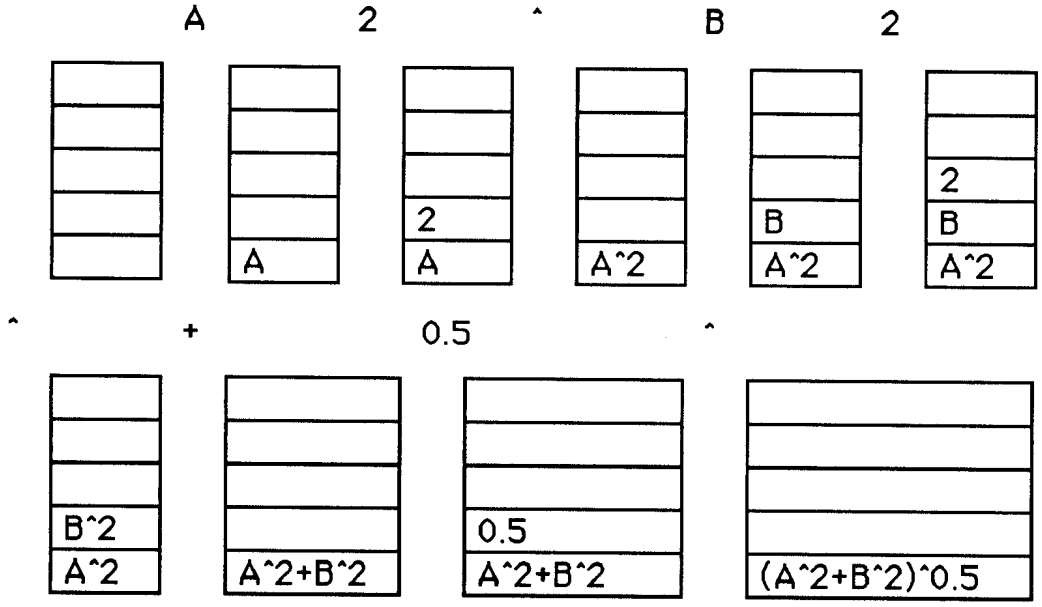
The **first example** involves the evaluation of an expression and its assignment to a variable. The studied command is

THT>MOD[1:5]=(A^2+B^2)^0.5

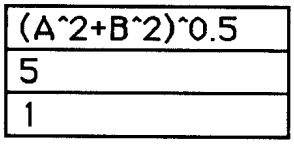
According to rules given in the Appendix 3, this command is converted into

| 1 | 5 | A | 2 | ^ | B | 2 | ^ | + | 0.5 | ^ | = | MOD |

The evaluation of the expression itself is | A | 2 | ^ | B | 2 | ^ | + | 0.5 | ^ | and is performed as on an RPN calculator : each time a variable name or a constant appears, the top of the stack is filled with the values of the corresponding array; each time an arithmetic operation appears, it is performed on the top of the stack, the stack pointer is decremented and the result is left on the top of the stack. Some mathematical functions with only one argument, eg: SIN, modify only the top of the stack without moving the pointer. At the end of the evaluation, the final result is found on the top of the stack.



The two first values | 1 | 5 | appearing at the beginning of the sequentially converted command are the bounds of assignments; they are pushed on the the stack before the evaluation of the expression, so that at this time, the stack status is :



The sequential operations | = | MOD | will if necessary create the variable **MOD** and then update it between indices 1 and 5 with the value located at the top of the stack.

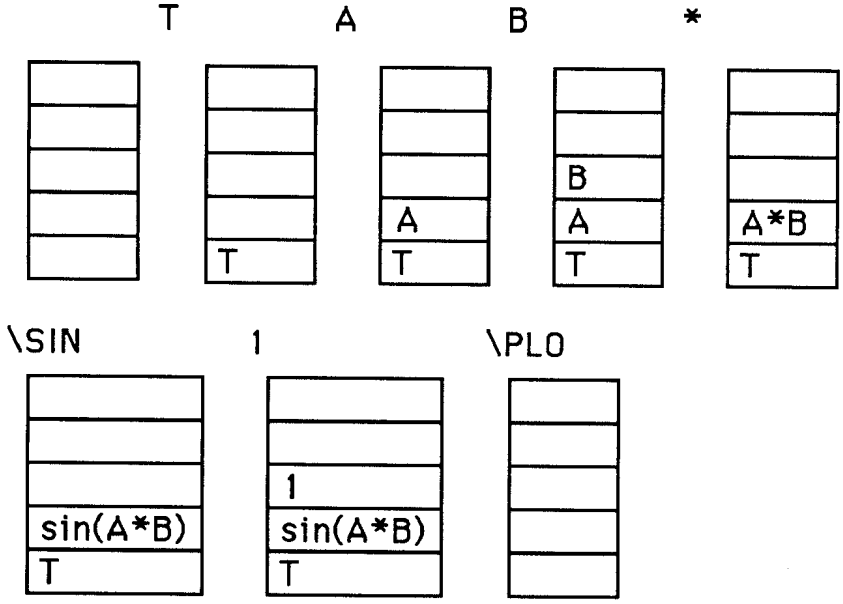
Another example is the call to a generic procedure :

THT>CALL PLOT(T,SIN(A*B),1)

According to the sequential language command rules, the operation set is

| T | A | B | * | \SIN | 1 | \PLO |

Each argument is successively pushed in the stack; during these successive pushes the expressions are evaluated, and finally the generic procedure is called, with its arguments on the top of the stack :



5. Implementation details

This section develops some implementation details which are in part specific to the host system, eg: memory limitation, programming language and acquisition system.

First the **acquisition interface** is done by calls to generic procedure and function : **CALL SHOT** opens a requested shot file and all subsequent accesses to the data will refer to this shot; these accesses are the function **FUN** which will return, according to the local normalisation, a raw voltage trace if the argument is negative and a predefined derived function if the argument is positive, and the function **CON** which will return the specified acquisition constant, eg : an amplifier gain, the filling gas mass, etc.. The success of the operation **SHOT** and **FUN** can be tested through the function **SAMP**[2].

As the calculations in **THT** are performed between traces with different acquisition timing, a **time base** has to be specified and controlled. This time base consists of three numbers; the number of points of a trace n, the acquisition period dt and the time at the first point t_1. All three can be accessed by the function **SAMP**[2]. By default, n is set to its maximum value (1024) while dt and t_1 are both set to 0. When the first trace is fetched through **FUN**, these three numbers are updated to the values of the fetched trace. All subsequent call to **FUN** will cause a resampling of the requested trace according to the first trace. Nevertheless the time base can be imposed by a call to the procedure **CLOCK**, during which all previously created variables are resampled with the requested time base. Note that the special case **CLOCK(0,0,0)** will reset the time base and consequently reset **all THT** variables.

The language used, **Fortran 77**, is somewhat inadequate for this kind of application lacking in reentrance, local, dynamical or structured variables, which lead to inelegant code and major limitations. This explains why the use of nested expressions, which appear as nested parenthesis, are limited, as well as nesting of conditional statements.

The small size of the PDP 11 memory requires the stack and the variable set to be placed in scratch files. Attempts are made to avoid access to these files : (a) real 32 bits traces are compressed in 16 bit integers if the number of points exceeds 512; (b) traces containing only one element (constants) are kept in memory; (c) variables are not actually in the stack but are represented by a pointer to their allocation.

The basic idea to perform all calculations and data transfers through the described stack and the imposed modularity of the software allow to add or modify generic procedures and functions very easily.

Details of the graphic philosophy and layout are listed in the Appendix 4. Appendix 5 give

[2] see Appendix 3.

a commented example of a full **THT** session.

Acknowledgments

This high level interpreter is imposed on top of the full TCA data retrieval and graphics packages, developed by many people during the operation of the TCA tokamak. This work was partially supported by the Fonds National Suisse de la Recherche Scientifique.

Appendix 1 : THT syntax definition

program = {statement}.
statement = [comment | assignment | procedure_call | subroutine_call | if_statement | loop_statement | EXIT] ↵.
comment = ; {character | space}.
assignment = variable [[[index] : [index]]] = expression.
procedure_call = CALL procedure parameter_list.
subroutine_call = @ subroutine.
if_statement = IF (expression : test_operator : expression) ↵ {statement} EIF.
loop_statement = LOOP ↵ {statement} REPEAT.
expression = ' string ' | term {addition_operator term}.
term = factor {multiplication_operator factor}.
factor = sub_factor {^ sub_factor}.
sub_factor = number | variable [[index]] | function parameter_list | (expression).
parameter_list = (expression {,expression}).
addition_operator = + | #.
multiplication_operator = * | /.
test_operator = LT | LE | EQ | GE | GT.
variable = designator.
index = expression.
function = designator.
procedure = designator.
subroutine = file_name.
designator = letter {letter | digit}.
number = [-] {digit} [.] {digit} [E [-] {digit}].
string = {character}.
character = letter | digit | ! | @ | # | $ | % | ^ | * | (|) | - | _ | = | + | [|] | / | \ | . | , | : | ; | |.
letter = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z.
digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.
file_name = [3]

[3] running system specification.

Appendix 2 : THT generic functions and procedures[4]

Functions

A = ABS (X)	$A_i = \|X_i\|$		$i=1,\ldots,n$	
A = ATAN2(X,Y)	$A_i = \arg(Y_i + j \cdot X_i)$		$i=1,\ldots,n$	
A = AVG(X)	$A_i = (\Sigma_{j=1,\ldots n} X_j)/n$		$i=1,\ldots,n$	
A = CON(N)	A_i = acquisition constant number N		$i=1,\ldots,n$	
A = COS(X)	$A_i = \cos(X_i)$		$i=1,\ldots,n$	
A = DER(X)	$A_i = (X_i - X_{i-1})/dt$		$i=2,\ldots,n$	
	$A_1 = 0$			
A = EXP(X)	$A_i = \exp(X_i)$		$i=1,\ldots,n$	
A = EXT(X)	$A_1 = i \mid X_i = \min_{j=1,\ldots n}(X_j)$			
	$A_2 = i \mid X_i = \max_{j=1,\ldots n}(X_j)$			
A = FUN(N)	A = acquisition function number N		if $N > 0$	
	A = acquisition trace number -N		if $N < 0$	5
A = IND(T)	$A_i = \text{int}(T-t_1)/dt + 1$	if $T \in [t_1,t_2]$	$i=1,\ldots,n$	
	$A_i = 1$	if $T < t_1$	$i=1,\ldots,$	
	$A_i = n$	if $T > t_2$	$i=1,\ldots,n$	
A = INT(X)	$A_i = A_{i-1} + X_i \cdot dt$		$i=2,\ldots,n$	
	$A_1 = 0$			
A = LN(X)	$A_i = \ln(X_i)$		$i=1,\ldots,n$	
A = MAX(X,Y)	$A_i = \max(X_i, Y_i)$		$i=1,\ldots,n$	
A = MIN(X,Y)	$A_i = \min(X_i, Y_i)$		$i=1,\ldots,n$	
A = ROLL(X,N)	$A_i = X_{((i-1-N) \bmod n)+1}$		$i=1,\ldots,n$	6
A = SAMP(N)	A_i = error ID[7]	if $N = 0$	$i=1,\ldots,n$	
	$A_i = n$	if $N = 1$	$i=1,\ldots,n$	
	$A_i = dt$	if $N = 2$	$i=1,\ldots,n$	
	$A_i = t_1$	if $N = 3$	$i=1,\ldots,n$	
A = SIN(X)	$A_i = \sin(X_i)$		$i=1,\ldots,n$	
A = SMOOTH(X,N)	$A_i = \Sigma_{j=i-N,i+N} X_j \cdot g(i-j) / \Sigma_{j=i-N,i+N} g(i-j)$		$i=N,\ldots,n-N$	

[4] in this Appendix, n refers to the number of points of the arrays, dt the time interval between two points, t_1 the time at the first point and t_2 the time at the last point.

[5] if the time base was not specified by a previous call to CLOCK or FUN, the time base of the specified function will define the array time base. If the time base already exists, the array will be resampled according to the existing time base; the beginning and the end are filled with the first and last value respectively if necessary.

[6] roll the array to the right if $N > 0$, and to the left if if $N < 0$.

[7] error ID 0 ok
1 CALL SHOT failed
2 FUN failed

	$g(i-j) = \exp(-((i-j)/N)^2)$			
	$A_i = A_N$		$i=1,...,N-1$	
	$A_i = A_{n-N}$		$i=n-N+1,...,n$	
A = TIME(N)	$A_i = t_1 + (i-1) \cdot dt$	if $N = 0$	$i=1,...,n$	
	$A_i = (i-1)/n \cdot dt^{-1}$	if $N = 1$	$i=1,...,n$	8
A = TFI(X,Y)	returns the imaginary part of the FFT of $Z = X + j \cdot Y$			9
A = TFR(X,Y)	returns the real part of the FFT of $Z = X + j \cdot Y$			9
A = UNSAVE(N,fn)	unsaves the trace stored in file fn at record number N			10
A = ZERO(X)	A_1 = number of zero crossing of X			
	$A_i = j \mid X_j \cdot X_{j-1} < 0$, crossing indices		$i=2,...,A_1$	
A = ZFILTR(X,C)	$A_i = \Sigma_{j=0,C[31]} C_{20+j} \cdot X_{i-j} - \Sigma_{j=1,C[30]} C_{10+j} \cdot A_{i-j} + C_{32}$		$i=k+1,...,n$	
	$A_i = C_i$		$i=1,...,k$	
			$k=\max(C_{30},C_{31})$	

Generic procedures

CALL AXIS(AXIS,MIN,MAX,title) defines a user axis number **AXIS**, where **AXIS** = 201,...,206, going from **MIN** to **MAX**, labelled **title.**

CALL BOX(BOX,F,XAXIS,YAXIS) defines a box number **BOX**, where **BOX** = 1,...16, **XAXIS** and **YAXIS** are the X and Y axis number. If the axis number is negative, an autoscaling is chosen. **F** defines the format[11] of the box by : **F** = 1000·position + 10·format + axis_position.

CALL CLOCK(N,DT,T1) changes the time base according to n = **N**, dt = **DT** and t_1 = **T1**. All existing variables are resampled during this operation.

CALL DEBUG(LEVEL) changes debugging level according to :

LEVEL = 0	no debugging
LEVEL = 1	displays command lines
LEVEL = 2	displays sequential operations
LEVEL = 3	displays variable set and stack status

CALL PAGE(F,title) creates a page labelled **title. F** describes the format and the output unit according to : **F** = 10·format[9] + unit. If the format is not 0, all boxes

---

[8] create the corresponding frequency array for FFT.

[9] when this function is called, n is reduced to the nearest but smaller or equal power of 2.

[10] see also CALL SAVE.

[11] see Appendix 4 for graphic display layout.

will be defined with autoscaling axis.

CALL PLOT(X,Y,BOX) plots the array **Y** as a function of **X** according to the definition of box number **BOX.**

CALL SAVE(X,N,fn) saves the array **X** in record number **N** of file **fn.** The file is a direct access block read file; each record contains five blocks; the first block contains n, dt and t_1; the array is stored in the last four blocks. If n>512 **SAVE** cannot be used.

CALL SHOT(N) opens the shot number **N.** If the operation failed, the function **SAMP(0)** is set to 1. All subsequent calls to the function **FUN** and **CON** will refer to the opened shot.

CALL TYPE(X,N,fn) dumps the first **N** values of the array **X** in file **fn.** **fn** is a sequential file, written with one value per line accompanied with the sample number and its corresponding time. If **fn** begins with character 'T', the dump will be performed on the terminal.

CALL VALUE(X,title,BOX) will display the first value of **X** accompanied by label **title** in box number **BOX.** If more than one call to **VALUE** is performed on the same box, the display will be vertically aligned.

Appendix 3 : THT sequential command conversion

assignment	[index index] expression variable =
procedure_call	parameter_list \designator
subroutine_call[12]	
if_statement	expression expression test_operator {statement} \EIF
loop_statement	\LOO {statement | \EXI} \REP
index	expression
expression	string | term {term addition_operator}
term	factor {factor multiplication_operator}
factor	sub_factor {sub_factor ^}
sub_factor	number | variable [index \OF] | parameter_list \function | expression
addition_operator	+ | #
multiplication_operator	* | /
test_operator	\LT | \LE | \EQ | \GE | \GT
variable	designator
parameter_list	expression {expression}

[12] running system facility.

Appendix 4 : Graphic specifications

Format

An oblong A4 page is divided in boxes. Their can be between 1 and 4 lines of boxes, each of them containing from 1 to 4 boxes. The format is a two digits number N_xN_y specifying respectively the number of rows and lines.

Position

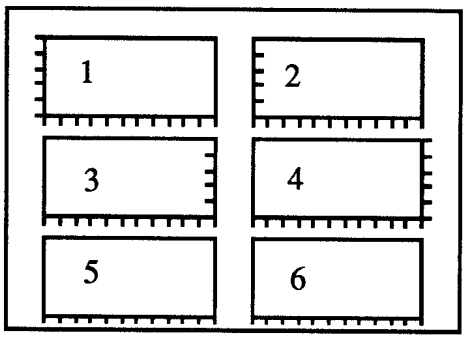
The position of one specific box is labelled by a number N_p scanning the box lines of the page ($N_p \leq N_xN_y$).

Axis position

The position of the vertical axis is specified by a number N_a. 1 refers to the outer left, 2 to the inner left, 3 to the inner right and 4 to the outer right position. 0 is used when no axis is needed. There is always one horizontal axis placed below each box.

Boxes size

N_x		N_y	
1	21 cm	1	15 cm
2	10.5 cm	2	7 cm
3	7.5 cm	3	5 cm
4	5.5 cm	4	4 cm



Here is an example of the graphical layout for a $N_xN_y = 23$. N_p is specified inside each box. Vertical axis position N_a for box 1, 2, 3 and 4 is respectively 1, 2, 3 and 4.

Appendix 5 : Example

```
1       THT> NSHOT=28714
2       THT> @EXAMPLE
3            CALL SHOT(NSHOT)
4            REAL=FUN(-171)
5            IMAG=FUN(-172)
6            PHASE=ATAN2(IMAG,REAL)*57.3
7            I=IND(0)
8            PHASE=PHASE#PHASE[I]
9            LOOP
10                   I=I+1
11                   JUMP=PHASE[I]#PHASE[I#1]
12                   IF (JUMP :GE: 180)
13                           PHASE[I:]=PHASE+360
14                   EIF
15                   IF (JUMP :LE: -180)
16                           PHASE[I:]=PHASE#360
17                   EIF
18                   IF (I :EQ: SAMP(1))
19                           EXIT
20                   EIF
21           REPEAT
22           I=EXT(PHASE)
23           I=I[2]
24           MAX=PHASE[I]
25           CALL PAGE(3,'EXAMPLE')
26           CALL AXIS(201,-5,5,'REAL')
27           CALL AXIS(202,-5,5,'IMAG')
28           CALL AXIS(203,0,800,'PHASE')
29           CALL AXIS(206)-.01,.2,'TIME')
30           CALL BOX(1,1131,206,201)
31           CALL BOX(2,2131,206,202)
32           CALL BOX(3,3131,206,203)
33           T=TIME(0)
34           CALL PLOT(T,REAL,1)
35           CALL PLOT(T,IMAG,2)
36           CALL PLOT(T,PHASE,3)
37           CALL VALUE(MAX,'MAXIMUM',3)
38      THT> ^Z
```

1 the variable **NSHOT** is created and initialised to 28714. As all variables are global, this variable can be used in the following subroutine.

2 subroutine **EXAMPLE** is called. The lines of this subroutine (3 to 37) are contained in the file EXAMPLE.THT.

3 opens the shot number **NSHOT**. All subsequent access to the data will refer to this shot.

4 the variable **REAL** is created and filled with the trace number 171. The time base is to the acquisition setting of this trace.

5 the variable **IMAG** is created and filled with the trace number 172 resampled according to the current time base, i.e. the one of the trace 171.

6 the variable **PHASE** is created and filled with the argument of the complex **(REAL,IMAG)** by mean of **ATAN2**. Conversion to degree is made by multiplying the result by 57.3.

7 the variable **I** is created and initialised with the index of time 0.

8 **PHASE** is set to zero at time 0 by subtracting the value at this time.

9 beginning of a **LOOP**.

10 this loop runs over the array index **I**, incremented at each loop.

11 **JUMP** contains the difference between successive point of **PHASE**.

12...14 if this difference is greater than 180°, a whole turn is added to the end of the array **PHASE** (positive fringe jump).

15...17 if this difference is less than -180°, a whole turn is subtracted to the end of the array **PHASE** (negative fringe jump).

18...20 if the last point **(SAMP(1))** is reached, **EXIT** of the loop.

21 end of the loop.

22 indices of the minimum and maximum of **PHASE** are stored in the first two elements of **I**.

23 **I** contains the index of the maximum.

24 **MAX** is created and contains the maximum of **PHASE**.

25 the procedure **PAGE** prepares a graphic page on unit **3**, with title **EXAMPLE**. This call is necessary to obtain graphic output. If **SHOT** was previously called, the page will also contain the shot number.

26...29 these four calls to **AXIS** create user axis number **201, 202, 203** and **206**, with their respective bounds and titles.

30...32 these three calls to **BOX** create three boxes labelled **1, 2** and **3**. The page format is **13** and axis will be drawn on the outer left of the box. Calls to **BOX** and **AXIS** can be interchanged; a call to **PAGE** does not reset these definitions.

33 creates the time array in variable **T** for use in plotting.

34...36 plot the variable **REAL, IMAG** and **PHASE** as a function of the time **T** in box **1, 2** and **3** respectively.

37 **VALUE** writes the first element of **MAX** in box **3** with the label **MAXIM.**

38 exit **THT.**