EQUATRAN

ALL-PURPOSE EQUATION SOLVER

Reference manual

PreFEED Corporation

About Notation of Statements

The format of each statement is denoted as shown below.

< Example >

$$\left\{ \begin{array}{c} \textbf{GLOBAL} \\ \textbf{LOCAL} \end{array} \right\} \quad a_1 = b_1 \quad «, a_2 = b_2, \cdots »$$

KEYWORD: An uppercase character string is a reserved word. This word is scripted in a source text as shown above. **GLOBAL** and **LOCAL** are examples of reserved words.

symbol : A lowercase character string is substituted with a symbol name, equation, constant, etc., in source text. a_1 and b_1 in the example correspond to this symbol.

$\left\{ \begin{array}{c} \cdots \\ \cdots \end{array} \right\}$ :: Braces (“{ }”) indicate that one of the enclosed character strings is selected.

« ⋯ » : Brackets (“« »”) indicate that the enclosed character strings can be omitted.

This example indicates that either GLOBAL or LOCAL is selected and one or multiple equations may follow.

Blank Page

EQUATRAN-G
ALL-PURPOSE EQUATION SOLVER
Reference Manual

2nd Edition

CONTENTS

2. EQUATRAN Statements

Blank Page

1. Syntax

1.1. Basics

The basic rules that apply to EQUATRAN source text are described below.

■ Character

The following characters are used in EQUATRAN source text:

• Alphabets	ABC···XYZ abc···xyz _
• Numerals	0123456789
• Special characters	+ - * / ^ = , . ; : < > ( ) [ ] % & | ! # $ ’ ¯

■ Statement

Source text and data text consist of a group of statements. A description of an equation is one of the statements.

• Symbol

A character string used to define a variable, label, parameter, table, etc., is referred to as a symbol. A symbol has a maximum of 8 alphanumeric characters (includes alphabets, numbers, and underline) and must start with an alphabet. It makes distinction between uppercase and lowercase characters.

The same symbol cannot have different meanings within one context (in main part or each macro). For example, the same symbol used as a built-in function name cannot be used as a variable name.

The following words are reserved words and cannot be used as symbols even if in uppercase or lowercase characters.

ALLI, BREAK, BY, CALL, CHANGE, END, EXTERNAL, FIND, FUNCTION, GLOBAL, INCLUDE, INITIAL, INPUT, INPUT0, INTEGRAL, LOCAL, MACRO, MAXIMIZE, MAXLOOP, MINIMIZE, OUTPUT, OUTPUT0, OUTPUT1, OUTPUT2, REPEAT, RESET, REV, STEP, TABLE, TREND, UNDER, UNTIL, VAR, VARIABLE, WHEN

• Comment

Block format comments enclosed between “/*” and “*/”, and the line format comment between “//” and the end of a line can be scripted as source text.

In block format comments other character string than “*/” can be described. If there are places where spaces can reside in text, a block format comment can be inserted in any space.

Character strings between “//” and the end of a line are assumed to be comments.

The first line of source text or data text is printed at the beginning when outputted to the printer, or outputted at the beginning when it is executed. It is preferable, therefore, to enter a title or a comment that indicates the content of the problem.

Comments cannot be nested (a comment cannot include another comment).

• Continuation of statement

If a statement extends beyond one line, the continuation symbol “..” is appended to the end of the line, indicating that the statement continues. However, the continuation symbol can be omitted in the following cases:

1.1. If the previous line ends with either of the following characters.

    + - * / ^ < > = # , & | :

1.2. If the next line starts with either of the following characters.

    = ,

A statement can contain a maximum of 2000 characters. However, extra spaces (redundant spaces: 2 or more successive spaces, spaces before and after a delimiter) are not counted.

<Example>

```
p=-2.0*X^2+0.5*X*y..
  -0.8*y^2+4*x-y
```

• Multi-statement

By inserting a semicolon “;” between statements two or more statements can be scripted on one line. There is no limitation as to the number of statements that can be scripted on one line.

<Example>

```
z+3=y; x=y-2.1*z; y-x/6=z
```

To make an equation easy to read, spaces can be inserted within the statement (e.g., at the beginning of the statement, and before and after an operator or parentheses). However, spaces cannot be inserted within character strings that have meaning as independent words.

<Example>

(Correct) `y=sin(x)+2*z`
A space can be inserted before and after the operator. Also, a space may be inserted between the function name and parenthesis.

(Incorrect) `RE SET y # 1`
A character string with meaning must not be separated by spaces.

• Label

A label is used to distinguish individual statements. Labels are used in the RESET statement, etc.

Specify a label by appending a colon “:” to the end of the label and placing it before a statement, as shown below.

«Label:» Statement

If label specification is omitted, a statement number is used instead. A statement number is a number applied to a statement when it is compiled. The number consists of 3 digits (line numbers) which are denoted by a $ at the beginning. A, B, C, ⋯ are appended to the end of the second and subsequent statements, respectively, in multi-statement. The correspondence of each statement can be made by outputting Source list 2.

If scripting the inequality constraint condition used for optimization calculation restriction and integral calculation interrupt, etc., just a label and an inequality sign can be scripted, as shown in the following example.

<Example>

```
c1:x<y+10
```

■ Equation script

An equation is scripted in conformance with the usual mathematical description method.

<Example>

$2x + y^3 = \frac{z}{4} \rightarrow$ `2*x+y^3=z/4`

• Operator

1.2.1. Arithmetic operator

The following operators can be used according to priority.

$$Exponent\ "\wedge"\ \begin{Bmatrix} Multiplication\ "*" \\ Division\ "/" \end{Bmatrix} \begin{Bmatrix} Addition\ "+" \\ Subtraction\ "-" \end{Bmatrix}$$

1.2.2. Relational operator

Equal	“==”
Not equal	“!=”
Less than	“<”
Greater than	“>”
Less than or equal	“<=”
Greater than or equal	“>=”

1.2.3. Logical operator

Logical negation (NOT)	“!”
Logical product (AND)	“&”
Logical sum (OR)	“|”

1.2.4. Priority

Operators have the following priority: arithmetic operator, relational operator, and logical operator.

The priority of calculations can be changed using parentheses.

- **Differential equation script**

The differential of variables are scripted using the differential sign (apostrophe " ' ") as follows:

$$\frac{dy}{dt} \rightarrow \texttt{Y'}$$

$$\frac{d^2y}{dt^2} \rightarrow \texttt{Y''}$$

y' and y'' are distinct variables. The variable name (*t* in this example) of independent variables is separately specified by the INTEGRAL statement.

An ordinary differential equation is scripted in source text using the above description method.

<Example>

$$\frac{d^2y}{d^2t} + a\frac{dy}{dt} + cy = t + d$$

$\rightarrow$ `y''+a*y'+c*y=t+d`

y can also be an array variable. In this case `y'` and `y''` automatically become an array variable of the same size. If a suffix is appended to a variable, it can be described as shown below.

```
y' (3)
```

- **Specifying an initial value**

To give an initial value to a variable to be integrated, the following # expression is used.

vname # expr

vname is the name of a variable given an initial value. A suffix can be appended to it.

expr is an initial value and can be provided with a constant, a variable name, or an equation. However, *expr* cannot be provided with such a value as an independent integral variable or as a function of the variable to be integrated.

• Logical operation

A logical value is a variable which is either true or false. EQUATRAN uses a real value to represent a logical value: 1.0 represents true and 0.0 represents false. Positive real numbers are true and 0 or negative real numbers are false.

For logical operation the following points should be noted.

(1) Relational operators cannot be used in succession.

<Example>

```
0<x<=2.7
```

This expression is not permitted.
It must be expressed as follows.

```
0<x&x<=2.7
```

(2) The priority of logical operators must be observed.

A logical product (&) has priority over a logical sum (|). If the logical sum is to be calculated prior to the logical product, use parentheses.

<Example>

```
(x>0|y>0)&(a<0|b<0)
```

(3) Using the IF function

If a logical operator, an arithmetic operator, and an equal sign are mixed, use the intrinsic function, IF, to make the equation easier to understand.

<Example>

```
x=IF(A>B+C&D<=0)
```

- **Conditional equation**

Variables for which the value differs depending on the condition are expressed as shown below.

$$
\begin{aligned}
a &= b_1 \ \text{«}\mathit{WHEN}\ c_1\text{»} \cdot\cdot \\
&= b_2\ \mathit{WHEN}\ c_2 \ \cdot\cdot \\
&\ \ \vdots \\
&= b_n\ \mathit{WHEN}\ c_n
\end{aligned}
$$

Where, a and b_1 to b_n are constants, variables, or operation expressions, and c_1 to c_n are logical operation expressions.

If the logical value of c_i is true, b_i is assigned to a. If the first WHEN term is omitted, c_1 is assumed to be true (and assumed as the default value).

If two or more c_i's that are true exist, the last one is valid.

If there is no c_i that is true, the calculation result is 0 and an error flag for an uncalculated variable value is set in a.

■ Constant

All constants are processed as real numbers. However, if the exponent of a power is an integer of 10 or less (not including decimal places), it is processed as an integer.

Exponents are denoted by an E.

Real numbers in the approximate range of $\pm 10^{-307}$ and 10^{308} can be used.

<Example>

```
123 4.567 -5.12
-1.7012E38 794.26e-5
```

■ Array

EQUATRAN can process linear or quadratic arrays.

Arrays cannot be used unless they have been declared beforehand by the VARIABLE statement. Also, care must be exercised as to which elements are to be calculated in equations including arrays.

• Array variable notation

An element of array variables is denoted by appending a suffix.

<Example>

If an array variable is declared `VAR a(3,2),b(5)`, then the element is `a(3,1),b(2)`,etc.

Partial arrays are represented by either of the following.

(1) Combination “.”

A partial array is represented by combining suffixes with a period “.”.

In this case suffixes must be arranged in incremental order.

(2) Continuous combination “:”

If the suffixes to be combined are continuous, they are represented by combining the lower and upper limits with a colon “:”.

(3) Omission of suffixes

If one suffix combines all the elements in a quadratic array variable, it can be omitted from the script.

If the first suffix is omitted, do not forget to insert a comma “,”.

<Example>

If an array variable is declared `VAR a(3,2),b(5)`, then

`b(1.3.5)` is the same as `(b(1),b(3),b(5))`.

`b(1.2.3.5)` is the same as `b(1:3.5)`.

`a(1:3,2)` is the same as `a(,2)`.

`a(1.3,1.2)` is the same as `a(1.3)`.

In quadratic array variables it is possible to represent the array that was transposed from the original array by appending an overline “¯” immediately after the variable name. Array transposition reverses the order of suffixes used to refer to the original array and does not create a new array variable.

<Example>

If an array variable is represented as `VAR a(3,2)`, `a¯` indicates a quadratic array of element numbers (2, 3).

`a¯(2,1)` and `a(1,2)`, or `a¯(2,3)` and `a(3,2)` indicate the same variable.

• Array constant notation

An array of constants is denoted as follows.

(1) For linear array

$(c_1, c_2, \cdots, c_n)$

(2) For quadratic array

$(c_{11}, c_{12}, \cdots, c_{1n})$ ..

$(c_{21}, c_{22}, \cdots, c_{2n})$ ..

$(c_{m1}, c_{m2}, \cdots, c_{mn})$ * c_i and c_{ij} are constants.

<Example>

```
(0.125,1.52,3.81)
(0.1,0.2)(0.3,0.5)(0.9,1.2)
```

• Calculation between array variables

Calculation between array variables is the calculation of elements that correspond to each variable.

<Example>

If an array variable is declared as

```
VAR a(3),b(3),c(2,3),d(2,3)
```

then, `a-b` means

```
(a(1)-b(1),a(2)-b(2),a(3)-b(3))
```

and, `c*d` means

```
┌ c(1,1)*d(1,1) c(1,2)*d(1,2) c(1,3)*d(1,3) ┐
└ c(2,1)*d(2,1) c(2,2)*d(2,2) c(2,3)*d(2,3) ┘
```

Calculation between variables of different dimensions is performed by expanding low-dimension variables to high-dimension variables.

<Example>

If an array variable is declared:

```
VAR a(3),c(2,3),d(2,3),x
```

then, `a*x` means

```
(a(1)*x,a(2)*x,a(3)*x)
```

and, `a-c` means

$$\begin{bmatrix} \texttt{a(1)-c(1,1)} & \texttt{a(2)-c(1,2)} & \texttt{a(3)-c(1,3)} \\ \texttt{a(1)-c(2,1)} & \texttt{a(2)-c(2,2)} & \texttt{a(3)-c(2,3)} \end{bmatrix}$$

and, `c/d(2,1)` means

$$\begin{bmatrix} \texttt{c(1,1)/d(2,1)} & \texttt{c(1,2)/d(2,1)} & \texttt{c(1,3)/d(2,1)} \\ \texttt{c(2,1)/d(2,1)} & \texttt{c(2,2)/d(2,1)} & \texttt{c(2,3)/d(2,1)} \end{bmatrix}$$

■ Variable

Variables up to quadratic array variables can be used and are declared by the VARIABLE statement.

This declaration can be omitted for scalar variables.

(1) Scalar variable

VARIABLE *vname*

(2) Linear array variable

VARIABLE *vname* (*n*)

(3) Quadratic array variable

VARIABLE *vname* (*n, m*)

n has element numbers of linear variables and *m* has those of quadratic variables. These variables can be used as quadratic array variables defined with *m* linear array variables which consist of *n* elements.

$$\text{A (m, n)} \rightarrow \quad \begin{array}{ccccc} A_{11} & A_{12} & A_{13} & \cdot\cdot & A_{1n} \\ A_{21} & A_{22} & A_{23} & & A_{2n} \\ A_{31} & A_{32} & A_{33} & & A_{3n} \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ A_{m1} & A_{m2} & A_{m3} & \cdot\cdot & A_{mn} \end{array}$$

<Example>

```
VAR b(5)=(0.1,0.3,0.5,0.8,1)
```

```
VAR b(5)
b=(0.1,0.3,0.5,0.8,1)
```

These two array variables have the same meaning.

The VARIABLE statement can be placed anywhere as long as it appears before the declared variables.

1.2.Built-in Function

EQUATRAN has an exponential function, logarithmic function, trigonometric function, hyperbolic trigonometric function, functions for determining the maximum/minimum element, sum, and product of array variables, dynamic functions dependent on an integral process, and other functions.

• Exponential and logarithmic functions

EXP(x) . . . . e^x

Exponential function

EXP10(x) . . . . . 10^x

Power of 10

SQRE(x). . . . . . x^2

Square of x

LOGE(x). . . . . . $log_e x$

Natural logarithm of x
$x > 0$

LOG10(x). . . . . $log_{10} x$

Common logarithm of x
$x > 0$

SORT(x) . . . . . . $\sqrt{x}$

Square root of x
$x \geq 0$

• Trigonometric function and inverse trigonometric function

Argument x is in radians.

SIN(x).	*sin x*
	Sine
COS(x)	*cos x*
	Cosine
TAN(x)	*tan x*
	Tangent

If argument x of an inverse trigonometric function is out of the defined area, an error results.

ASIN(x)	$sin^{-1}x$
	Arc sine
	$-1 \leq x \leq 1$
ACOS(x).	$cos^{-1}x$
	Arc cosine
	$-1 \leq x \leq 1$
ATAN(x).	$tan^{-1}x$
	Arc tangent

• Hyperbolic trigonometric function and inverse hyperbolic trigonometric function

Argument x is in radians.

SINH(x)	*sinh x*
	Hyperbolic sine
COSH(x).	*cosh x*
	Hyperbolic cosine
TANH(x)	*tanh x*
	Hyperbolic tangent

An inverse hyperbolic trigonometric function has a defined area of x.

ASINH(x).	$sinh^{-1}x$
	Hyperbolic arc sine
ACOSH(x)	$cosh^{-1}x$
	Hyperbolic arc cosine
	$x \geq 1$

ATANH(x) . . . . $tanh^{-1}x$

Hyperbolic arc tangent

$-1 \leq x \leq 1$

• Functions for processing array variables

Argument y must be an array variable in the sum, prod, maxof , and minof functions.

SUM(y). . . . . . . Total sum of array variables

$$\sum_{i=0}^{n} yi$$

PROD(y). . . . . . Total product of array variables

$$\prod_{i=0}^{n} yi$$

MAXOF(y). . . . Maximum element of array variables

$$\max_{i} yi$$

MINOF(y) . . . . Minimum element of array variables

$$\min_{i} yi$$

MAX and MIN must have 2 or more arguments.

MAX(a, b, c, ⋯) . . Maximum variable value

MIN(a, b, c, ⋯). . . Minimum variable value

• Other arithmetic functions

INT(x). . . . . Cut-off to integer

MOD(x, y) . . . . Remainder of x/y

SIGN(x) . . . . . . Sign output

The result is +1 or -1.

IF(x) . . . . Logical value

The result is 1.0 (true) or 0.0 (false).

ABS(x) . . . . |x|

Absolute value

- **Dynamic built-in function**

DERIV(v, vs)	Differential function
	Returns the derivative of variable v.
	vs is supplied with a value at the start of integral calculation.
DELAY(v, vs, d, N)	Obtains the value given to v as dead time.
	v: Variable name
	vs: Value at start of integral calculation
	d: Delay time
	N: Number of memory area creating delay
PREV(v, vs)	Previous value function
	Returns one step previous value in integral calculation of variable v.
	The value at the start of integral calculation is given to vs.
STEP(v)	Step function
	Returns 1 if v > 0, and 0 if v <= 0.
SYNC(v).	Synchronization
	This value is changed only at integral division points.
RANDU(n)	Uniform random number
	Obtains uniform random numbers between 0 and 1.
	An integer between 1 and 10 is given to n as a random serial number.
GAUSS(n)	Normal distribution random number
	Obtains a normal distribution random number with mean 0 and dispersion 1.
	An integer between 1 and 10 is given to n as a random serial number.

• List of functions

Notation	Meaning
`ABS(x)`	$\|x\|$
`ACOS(x)`	$cos^{-1}x$
`ACOSH(x)`	$cosh^{-1}x$
`ASIN(x)`	$sin^{-1}x$
`ASINH(x)`	$sinh^{-1}x$
`ATAN(x)`	$tan^{-1}x$
`ATANH(x)`	$tanh^{-1}x$
`COS(x)`	$cos\ x$
`COSH(x)`	$cosh\ x$
`DERIV(v, vs)`	Differential function
`DELAY(v, vs, d, N)`	Dead time
`EXP(x)`	e^x
`EXP10(x)`	10^x
`GAUSS(n)`	Normal distribution random number
`IF(x)`	Logical value
`INT(x)`	Cut-off to integer
`LOGE(x)`	$log_e x$
`LOG10(x)`	$log_{10} x$
`MAX(a, b, c, ···)`	Maximum variable value
`MIN(a, b, c, ···)`	Minimum variable value
`MAXOF(y)`	Maximum element of array
`MINOF(y)`	Minimum element of array
`MOD(x, y)`	Residual of *x*/*y*
`PREV(v, vs)`	Previous value function
`PROD(y)`	Total product of array variables
`RANDU(n)`	Uniform random number
`SIN(x)`	$sin\ x$
`SIGN(x)`	Sign output
`SINH(x)`	$sinh\ x$
`SQRE(x)`	x^2
`SRQT(x)`	$\sqrt{X}$
`STEP(v)`	Step function
`SUM(y)`	Total sum of array variables
`SYNC(v)`	Synchronization
`TAN(x)`	$tan\ x$
`TANH(x)`	$tanh\ x$

1.3.Built-in Constant

- Pi   _pi

  _pi = 3.141592653589793

- Base of natural logarithm   _e

  _e = 2.718281828459045

- Degree-to-radian conversion   _rad

  _rad = 57.29577951308232

<Example>

```
S=_pi*r^2

Vx=V*cos(theta/_rad)
Vy=V*sin(theta/_rad)
```

1.4.Input/Output

The INPUT statement is used to specify whether to input values interactively into variables at execution or to supply values to be loaded as data text. Also, the INPUT0 statement is used to specify values to be inputted by loading them as data text in array format.

The INITIAL statement is used to specify the initial values to integrand variables. Usage conforms to that of the INPUT statement.

To specify variables to output calculation results, the OUTPUT statement, OUTPUT0 statement, OUTPUT1 statement, or OUTPUT2 statement is used.

The OUTPUT statement specifies output to the window (or the temporary file), the OUTPUT0 statement specifies output to a result text file, and the OUTPUT1 and OUTPUT2 statements specify output to the result file 1 and result file 2, respectively.

<Example>

```
INPUT A,B,C
INITIAL t,X,Y
OUTPUT Xcal,Ycal
OUTPUT1 t,Xcal,Ycal
```

1.5.Numerical Table

If a numerical table is defined, it can be used in the same way as a function. Numerical values which are not in the numerical table are approximated by interpolating points within the range of the table or by extrapolating points out of the range of the table. The numerical table is convenient if calculation is performed based on measured data.

Numerical tables are defined with the TABLE statement.

<Example>

One-dimensional numerical table:

x	1.0	1.5	2.0	3.0	4.0	6.0
y	0.38	0.36	0.34	0.28	0.21	0.14

```
VARx(6),y(6)
TABLE y=tab1(x)REV
x=(1.0,1.5,2.0,3.0,4.0,6.0)
y=(0.38,0.36,0.34,0.28,0.21,0.14)
```

or

```
TABLE y=tab1(x)REV
x(6)=(1.0,1.5,2.0,3.0,4.0,6.0)
x(6)=(0.38,0.36,0.34,0.28,0.21,0.14)
```

Two-dimensional numerical table:

		x_1			
		1	2	3	4
x_2	-1	0.01	0.30	0.80	0.95
	0	0.02	0.20	0.40	0.90
	1	0.05	0.15	0.30	0.80

```
VAR x1(4),x2(3),y(3,4)
TABLE y=tab2(x2,x1)
x1=(1,2,3,4)
x2=(-1,0,1)
y=(0.01,0.3,0.8,0.95)..
  (0.02,0.2,0.4,0.9)..
  (0.05,0.15,0.3,0.8)
```

Values in a numerical table are scripted in the same way a function is referenced. One-dimensional numerical tables consist of one argument; two-dimensional numerical tables consist of two arguments.

Arguments are constants or arithmetic equations, which are provided with values that correspond to x_1 and x_2 in the numerical table definition statement.

References are made by linear interpolation or linear extrapolation in a numerical table where the STEP term has not been specified.

1.6.Convergence Calculation

EQUATRAN has a function to execute convergence calculation by automatically identifying equations required for the calculation (this is refered to as Auto-reset). The user can specify equations used to determine the convergence and the accuracy using the RESET statement (this is refered to as reset specification).

<Example>

```
eq:z^2+y=5.2
RESET z#0 BY eq
```

1.7.Optimization Calculation

EQUATRAN can resolve single-variable or multi-variable optimization problems.

Optimization calculation is executed by specifying independent variables and evaluation functions to be optimized with the FIND statement.

<Example>

```
FIND(x#1[0,20],y#1[0,20])..
MINIMIZE p UNDER cond UNTIL 0.5%
p=3*x^2+3*y^2-2*x*y-10*x-50*y+330
cond=IF(x<y & x+y<=20)
```

1.8.Integral Calculation

Integral calculation is specified by the INTEGRAL statement.

<Example>

```
INTEGRAL t[0,50] STEP h BY RKV BREAK cond
h=50/300
cond=IF(t>tnext)
```

- **Trend display of integral process**

The TREND statement is used to specify the trend display output of the integral process.

If the display period is not specified, the integral process is displayed with the integral division interval. If a period indivisible by the integral division interval is specified, the period is set to the multiple of the nearest integral division interval.

<Example>

```
INTEGRAL t[0,50]  STEP 0.01
TREND y[0,10],y’ [-5,5] STEP 0.1
```

- **Script of discontinuous phenomenon**

The CHANGE statement has a function to immediately change the specific value if a given condition is generated. Use this function to script a discontinuous phenomenon.

<Example>

```
CHANGE x#0 ON (x>10)
```

1.9.Repeated Calculation

The REPEAT statement can repeatedly calculate problems that do not include a differential equation. With this statement it is easy to example a case study such as changing variable values of parameters at even intervals.

<Example>

```
y=a*x^3+b*x^2+c*x+d
REPEAT x[0,10] STEP 0.5
```

1.10.Parameter

Parameters have a function to replace arbitrary symbols in source text with other character strings. There are two parameters: GLOBAL parameters which are valid throughout the entire source text, and LOCAL parameters which are only valid for statements in one macro or main.

Parameters are defined with the GLOBAL statement and the LOCAL statement.

<Example>

```
LOCAL N=5, parm=0.5*X
VAR a(N),b(N),c(N+2)
a(2:N)=b(1:N-1)+parm
```

These are replaced with the following parameters.

```
VAR a(5),b(5),c(5+2)
a(2:5)=b(1:5-1)+0.5*X
```

Every part in a statement can be replaced with parameters. However, reserved words cannot be replaced with parameters. The GLOBAL statement and LOCAL statement must be positioned before all statements that use parameter symbols.

If the same symbol is defined as a parameter in two or more places, the GLOBAL parameter has priority over the LOCAL parameter. If there are parameters of the same type, priority is assigned to those defined first.

If a parameter is only declared and no value is specified, it is used by inputting the value interactively or by loading it from a data text.

1.11.Macro

If a group of equations that are used repeatedly have been saved as batch under a name, the equations can be called at any time with the name. This function is known as a macro.

Macros must be defined before they can be called. A macro is defined by macro definition statements between the MACRO statement and the END statement.

From macro statement groups defined by macro definition statements, the statements in macros are embedded sequentially in the text where the CALL statement which is a macro call statement is scripted. In this case the following operations are performed.

- Replaced by a GLOBAL parameter
- Replaced by a LOCAL parameter defined by the CALL statement
- Replaced by a LOCAL parameter defined in macro definition statements
- Modifying a variable name and a label in macros by the CALL statement label

<Example>

```
MACRO SHEEP
    x=x*(1+r)
    y=y*R
END SHEEP

CALL SHEEP (x=x1,y=y1,r=r1,R=R1)
CALL SHEEP (x=x2,y=y2,r=r2,R=R2)
```

1.12. User Function

■ Defining user functions

User functions are defined between the FUNCTION statement and the END statement.

There are two types of user functions: the function type and the subroutine type. They have different argument specifications.

- **Definition of function-type user functions**

A function-type user function calculates one output variable (scalar) from input variables.

Input variable names are used in arguments by delimiting them with commas. Array variables can be used for input variables. However, array elements or partial arrays cannot be defined as arguments. A function name should be the name of a variable to be outputted along with a function name.

<Example>

```
FUNCTION psat (t,A)
    VAR A(3)
        t,
        psat
    loge(psat)=A(1)+A(2)+A(3)/T^2
    T=t+273.15
END psat
```

- **Definition of subroutine-type user functions**

Subroutine-type user functions can output array variables or multiple variables and can exchange input variables and output variables using arguments. Input variable names are specified for the first half of arguments by delimiting them with commas, and output variable names are specified for the second half of arguments by delimiting with a comma. An input variable and an output variable are delimited by a semicolon “;”.

<Example>

```
FUNCTION statist (D;mean,sigma)
    LOCAL N=10
    VAR D(N),mean,sigma
    s=sum(D)
    mean=s/N
    V=sum((D-mean)^2)/(N-1)
    sigma=sqrt(V)
END
```

- **Restrictions on function definition**

The symbol names of all variable names, excluding common GLOBAL parameters, used in a function definition script are valid only in this script.

The exchange of function values between a function and the function call side is performed only through arguments.

Statements can be scripted in the function script parts in the same way it is usually done in EQUATRAN, however the following statements cannot be used.

- EXTERNAL statement
- FUNCTION statement
- INPUT statement
- INITIAL statement

Also, another function cannot be defined in the definition of a function, and the main script cannot be placed before a function definition, except for comments and the definition statement of GLOBAL parameters.

■ Calling a user function

A user function is called from the main or function scripts using the function name. A function must have been defined by the FUNCTION statement (or the EXTERNAL statement) prior to being called. Also, calls cannot be performed recursively.

■ Outputting data from inside a function

To output data from inside a user definition function to a window or a file, append the output option flag (WITH term) when calling a function as follows:

fname (*arguments*) WITH (*olist*)

Specify olist from the following:

OUTPUT	Standard output
OUTPUT0	Result text file output
OUTPUT1	Result file output 1
OUTPUT2	Result file output 2
TREND	Trend output
DUMPR	Dump for RESET
DUMPF	Dump for FIND
TRACE.	Trace output

To specify two or more types of outputs, delimit them with commas.

<Example>

```
funct1(A,x,ms) WITH (TREND)
y=funct2(y,zz) WITH (OUTPUT,OUTPUT1)
```

1.13. Include Function

Other source text can be embedded in source text with the INCLUDE statement. The INCLUDE statement is provided with one file name without extension “eqs”. To include two or more statements, script the same number of INCLUDE statements. An INCLUDE statement can be nested only once. The file loaded by the INCLUDE statement in the file to be included must not contain the INCLUDE statement.

<Example>

```
  INCLUDE  LIBMATH
  INCLUDE  LIBTECH
```

Blank Page

2.EQUATRAN Statements

■ CALL

Embedding the defined macro in a context

«*label*:» **CALL** *mname* «(a_1=b_1 «,a_2=b_2,⋯»)»

label

Label used for modifying a variable name and a label.

mname

The name of a macro to be called.

In the parentheses the LOCAL parameters effective inside the macro that has been called are given in such a format that reserved word LOCAL is removed from the LOCAL statement.

a_1, a_2, ⋯ and b_1, b_2, ⋯ are the same as those in the parameter definition statement.

The statements in macros are embedded sequentially in the context where the CALL statement is scripted. In this case the following operations are performed.

- Replaced by a GLOBAL parameter
- Replaced by a LOCAL parameter defined by the CALL statement
- Replaced by a LOCAL parameter defined in macro definition statements
- Modifying a variable name and a label in macros by the CALL statement label

<Example>

```
CALL TOWER (HT=3.0,HS=100.0)
```

■ CHANGE

Changing a variable value during integral calculation

CHANGE (x_1#a_1, x_2#a_2, ⋯x_n#a_n) **ON** (*cond*)

x_i#a_i

# expression that changes a variable value and to be executed if the condition is triggered. If the condition is triggered, all # expressions are executed and the value of a_i at this time is substituted for x_i.

cond

A variable or logical equation that indicates the condition. It is assumed that the condition is generated if the logical value is changed from false (0 or negative value) to true (positive value).

The script of # equations is the same as that of the usual # equation that provides an integral initial value. A variable on the left side of a # equation must be the variable to be integrated. On the right side of a # equation a general equation can be scripted. Also, a_i can be an integral function with independent variables.

cond is usually a scalar variable or equation and also can script an array. If this is the case, all the # equations must be the array with the same number of elements as *cond*. Upon generation of the condition for each *cond* element, the corresponding # equation element is executed.

■ EXTERNAL

Declaration of external functions

EXTERNAL *fname* (*inlist* «*; outlist*»)

fname

Function name

inlist

List of input arguments

outlist

List of output arguments

The list of input and output arguments includes formal argument variable names. If variable names are of array, a suffix format that indicates array dimension is appended to the variable names by delimiting with a comma. Therefore, each item in arguments is formatted in either of the following.

symbol . . . . Scalar variable

symbol ( ). . . . . . One-dimensional array

symbol (,). . . . . . Two-dimensional array

Symbol is an arbitrary symbol name and does not need to conform to an argument variable name for function definition. For array variables the suffix format that indicates array dimension is provided with no suffix.

For function-type functions the *outlist* part can be omitted in the same way as the FUNCTION statement.

<Example>

```
EXTERNAL psat(t,A( ))
EXTERNAL statist(D( ); m, s)
```

■ FIND

Specification of optimization calculation

$$\textbf{FIND}\ \left\{\begin{matrix} x\#r[l,h] \\ (x_1\#r_1[l_1,h_1],\ x_2\#r_2[l_2,h_2],\ \cdots) \end{matrix}\right\}$$

$$\left\{\begin{matrix} \left\{\begin{matrix}\textbf{MAXIMIZE}\\ \textbf{MINIMIZE}\end{matrix}\right\} f \\ \\ \textbf{LEAST}\left\{\begin{matrix} f \\ (f_1,f_2,\cdots)\end{matrix}\right\} \end{matrix}\right\} \quad «\textbf{BY}\left\{\begin{matrix}\textbf{COMP}\\ \textbf{SQP}\end{matrix}\right\}»$$

$$«\textbf{UNDER}\left\{\begin{matrix} c \\ (c_1,c_2,\cdots)\end{matrix}\right\}»$$

$$«\textbf{UNTIL}\quad k\%» \quad «\textbf{MAXLOOP}\ \ n»$$

x

An independent variable to be optimized. *r*, *l*, and *h* are an initial value, lower limit, and upper limit, respectively, and given a constant, variable name, or variable name with a suffix. These initial value, lower limit, and upper limit are omissible. If there are multiple independent variables, they are delimited with commas and enclosed entirely with parentheses “( )” as shown above. However, the number of independent variables is limited up to 100.

f

A variable that represents an object function value. If LEAST is specified for this variable, the variable must be an array. Also in this case, multiple arrays can be specified, delimiting them with commas and enclosing entirely with parentheses “( )” as shown above.

MAXIMIZE

Instructs *f* to be maximized.

MINIMIZE

Instructs *f* to be minimized.

LEAST

Instructs that the sum of the squares of all array *f* elements is minimized.

BY ···

Specifies the optimizing calculation method. SQP specifies the SQP method (sequential quadratic programming method) and COMP specifies the complex method. If both are omitted, the complex method is used.

UNDER *c*

This term gives the restrictive conditions and is omitted if there is no restrictive condition. *c* is a variable name that represents a restrictive condition value. If there are multiple restrictive conditions, they are delimited with commas and enclosed entirely with parentheses "( )". If the SQP method is to be used, an equal sign format condition cannot be used in restrictive conditional equations.

UNTIL *k%*

This term is an option that gives an optimized accuracy and can be omitted. *k* must be a real constant. In Box's Complex method, the accuracy is the difference between the value of the independent variables at the actual optimization point and the values at the optimization point. In SQP method, the value is used to determine if the constraint is satisfied to reach optimum solution (as an absolute value).

MAXLOOP *n*

Specifies the upper limit of repeat times of search calculation for optimization. The maximum number of repeat times is given to *n* as an integer constant.

<Example>

```
FIND(x#1[0,20],y#1[0,20]) MINIMIZE p ..
   UNDER cond UNTIL 0.5%
p=3*x^2+3*y^2-2*x*y-10*x-50*y+330
cond=IF(x<y & x+y<=20)
```

■ FUNCTION ··· END

User function definition

FUNCTION *fname*

(*arguments*) (Function script)

END «*fname*»

fname

A symbol that is a function name. It makes no distinction between uppercase and lowercase characters, different from variable names, etc.

fname of the END statement can be omitted.

arguments

A list of variables that are arguments is given.

An input argument (a list of input arguments, if multiple, by delimiting them with commas) is provided for function-type user functions. An input argument and an output argument (if both are multiple, lists of input arguments and output arguments by delimiting them with commas) are provided by delimiting each with a semicolon for subroutine-type functions.

Namely, for a function-type function the FUNCTION statement is expressed as:

FUNCTION *fname* (vi_1 «, vi_2 ⋯»)

Also, for a subroutine-type function the FUNCTION statement is expressed as:

FUNCTION *fname* (vi_1 «, vi_2 ⋯» ; vo_1 «, vo_2 ⋯»)

■ GLOBAL/LOCAL

Parameter definition

$$\left\{\begin{matrix}\textbf{GLOBAL}\\ \textbf{LOCAL}\end{matrix}\right\}\ a_1\ \text{«}= b_1\text{»}\ \text{«}\textit{"explanatory term"}\text{»}\ \text{«}, a_2\ \text{«}= b_2\text{»}, \cdots\text{»}$$

a_1, a_2, ⋯

Parameter symbols to be replaced. b_1, b_2, ⋯ are character strings to be substituted for the corresponding parameter symbols. If any character string is not given, it is used by inputting the value interactively at compiling or by loading it from a data text. The contents of character strings b_1, b_2, ⋯ are arbitrary, but the following cannot be used.

- A character string including the parentheses “( )” that has no corresponding part.
- A character string including a comma that is not contained in the parentheses.

<Example>

```
LOCAL N=5,parm=0.5*X
```

Explanatory term

An arbitrary explanatory statement that explains a parameter within 40 characters.

The explanatory term is displayed along with the prompt of parameter entry at the time of compiling.

Every part in a statement can be replaced with parameters. However, reserved words cannot be replaced with parameters.

The GLOBAL statement and LOCAL statement must be positioned before all statements that use parameter symbols.

If the same symbol is defined as a parameter in two or more places, the GLOBAL parameter has priority over the LOCAL parameter. If there are parameters of the same type, priority is assigned to those defined first.

■ INCLUDE

Specifies embedding of another source file.

INCLUDE *filename*

filename

The name of a source text file to be embedded.

A source text file has an extension of .EQS. This extension is omitted in the INCLUDE statement. The INCLUDE statement can include only one file name. If two or more file names are to be included, the same number of INCLUDE statements must be scripted.

An INCLUDE statement can be nested only once. The file loaded by the INCLUDE statement in the file to be included must not contain the INCLUDE statement.

■ INITIAL

Specifies initial value input

$$\textbf{INITIAL} \left\{ \begin{array}{l} v_1 «, v_2, ...» \\ \textbf{ALLI} \end{array} \right\}$$

v_1, v_2, ...

Variable names that can be either scalar variable names or array variable names. If array variables are given without a suffix, all variables that have been defined are specified to be entered.

ALLI

Specifies the initial values of all to be integrated so that they are entered with no suffix appended.

■ INPUT/INPUT0

Specifies variable value input.

$$\left\{ \begin{array}{l} \textbf{INPUT} \\ \textbf{INPUT0} \end{array} \right\} \quad v_1 \quad «, v_2, ...»$$

v_1, v_2, ...

Variable names that can use either scalar variable names or array variable names. If array variables are given without a suffix, all variables that have been defined are specified to be entered.

If variables are specified with the INPUT0 statement, variable values are loaded from data text into the variables in the sequence they were specified.

■ INTEGRAL

Specifies integral calculation

INTEGRAL *vname* $[k_s,k_e]$ «**STEP** *h* «$[r_h]$» »

«**BY** $\left\{\begin{matrix}\textbf{RKF}\\ \textbf{RKV}\\ \textbf{EUL}\end{matrix}\right\}$ »

«**BREAK** $\left\{\begin{matrix}\boldsymbol{c}\\ (\boldsymbol{c_1},\boldsymbol{c_2},\cdots)\end{matrix}\right\}$ » «**UNTIL** $(e_a, e_r\%)$ »

vname

An independent variable name for integral. The independent variable values at the start point and end point are given to k_s and k_e, respectively. An array or array element can be specified as an independent variable.

STEP *h* [r_h]

In this term an integral division *h* is specified. $[r_h]$ is provided with the minimum division ratio. The minimum division ratio is valid only if the variable division integral method is used and can be omitted. The minimum division is determined by *h* x r_h. The default of r_h is 0.01.

BY ···

This term specifies an integral method by selecting one from the following. If the BY term is omitted, RKF is assumed to be specified.

RKF	4th-order Runge-Kutta method (fixed division)
RKV.	4th-order Runge-Kutta method (variable division)
EUL	Euler's method

BREAK *c*

This term gives an integration interrupt condition and is omitted if there is no interrupt condition. *c* is a variable name to give an interrupt condition. If there are multiple interrupt conditions, they are delimited with commas and enclosed entirely with parentheses "( )". If any one of logical values of *c*'s is true, integration is interrupted.

UNTIL (e_a, e_r%)

This term gives a tolerance of integration cutoff error and is valid only if the variable division integral method is specified. e_a is given an absolute permissible error, and e_r% is given a relative permissible error.

k_s, k_e, and h are provided with constants or variable names. If either is provided with a variable name, it must be a scalar variable. The STEP term can be omitted. If this is the case, h will be $\dfrac{|k_e - k_s|}{100}$.

<Example>

```
INTEGRAL t[0,50] STEP h BY RKV BREAK cond
h=50/300
cond=IF(t>tnext)
```

■ LOCAL

Refer to GLOBAL.

■ MACRO ··· END

Macro definition

MACRO ***mname***

(macro definition)

END ***mname***

mname

A symbol that gives a macro name. In macro definition statement groups the MACRO definition statement cannot be scripted.

The MACRO statement and END statement indicate the start and end of statement groups that are defined as macros.

■ OUTPUT/OUTPUT0/OUTPUT1/OUTPUT2

Specifies output variables.

The OUTPUT statement specifies output to a window (or the temporary file), OUTPUT0 statement specifies output to a result text file, OUTPUT1 and OUTPUT2 statements specify output to result file 1 and result file 2, respectively.

$$\left\{\begin{matrix}\textbf{OUTPUT}\\ \textbf{OUTPUT0}\\ \textbf{OUTPUT1}\\ \textbf{OUTPUT2}\end{matrix}\right\} \left\{\begin{matrix} v_1 \;«,v_2, ...» \\ \textbf{ALLI}\end{matrix}\right\} \;«\textbf{STEP}\;\; h»$$

v_1, v_2, ...

Specifies variable names to be outputted, which can be either scalar variable names or array variable names. If array variables are given without a suffix, all variables that have been defined are displayed. The sequence of outputting variable values is the same as that defined with those statements.

ALLI

Outputs the values of all variables to be integrated.

STEP *h*

A reserved word to give an output period with respect to integral calculation functions. If this term is added, output is executed periodically at a period of *h*. *h* must be a positive constant.

■ REPEAT

Specifies repeated calculations.

REPEAT *vname* [k_s, k_e] **STEP** *h*

vname

A variable to be calculated repeatedly. k_s and k_e are the start value and end value of repeat calculation, respectively.

STEP *h*

Specifies the division interval of repeated calculation.

k_s, k_e, and *h* are provided with constants or scalar variable names.

<Example>

```
y=a*x^3+b*x^2+c*x+d
REPEAT x[0,10] STEP 0.5
```

■ RESET

Specifies repeated convergence calculation.

RESET *x*#*r*[*l*, *h*] **BY** *label* «**UNTIL** *k%*»

x

Specifies an unknown variable to be assumed. *r* is an initial value, *l* is a lower limit, and *h* is an upper limit, which are provided with constant, variable name, or variable name with a suffix. These initial value, lower limit, and upper limit can be omitted and, if omitted, the default value is used for them.

BY *label*

A label that specifies an equation to check consistency between the left and right sides.

UNTIL *k%*

This term is an option that gives the accuracy of convergence calculation. If the difference between both sides of the equal sign of the *label* equation is less than k% of the right side value, the difference is assumed to be converged. If % is omitted, convergence is checked with absolute value. *k* is a real constant. The whole UNTIL term can be omitted, and if so, the default value is specified.

If *x* is an array, *label* must also be an array with the same number of elements.

If array variables are included in an equation with a label *label*, the equation itself becomes an array, and contains the same number of equations as that of elements of the highest dimension variable. Therefore, to specify one of those equations, it is necessary to append a suffix to the label.

<Example>

```
VAR z(3),y(3)
eq:z^2+y=5.2
```

The above equation contains three equations. One of them is expressed as:

```
z(2)^2+y(2)=5.2
```

To specify the above equation, the label is expressed as follows:

```
eq(2)
```

■ TABLE

Numerical table definition

$$\textbf{TABLE}\ \ y = tname(«x_2,\ » x_1)\ \ «\begin{Bmatrix}\textbf{REV}\\ \textbf{STEP}\end{Bmatrix}»$$

tname

A symbol that gives a table name.

(*x*$_2$, *x*$_1$)

One-dimensional array variables that give tables of arguments. If the table to be defined is an one-dimensional one, x_2 is omitted.

y

One-dimensional or two-dimensional numerical array variables that give numerical tables.

REV

An option that allows reverse reference to numerical tables (i.e., acquires argument values from values in the numerical tables). If reverse reference calculation is not necessary, REV can be omitted.

STEP

Usually in reference to numerical tables, numerical values are calculated by linear interpolation or linear extrapolation. If STEP has been specified, the numerical values corresponding to the values in the smaller argument tables are used.

y, x_1, and x_2 have been defined by the VARIABLE statement and at the same time have been given constant values with the equal sign (=). Arguments x_1 and x_2 must be arranged in incremental order.

■ TREND

Specifies trend display during integral calculation.

$$\textbf{TREND} \begin{Bmatrix} v_1,v_2,... \\ v_1[l_1,h_1],v_2[l_2,h_2],... \end{Bmatrix} \textbf{STEP}\ h$$

v_1, v_2, ⋯

Variable names to be displayed. They can be either scalar variable names or array variable names. A maximum of 20 variables can be displayed. If array variables are included, the total number of variable elements must be 20 or less.

[l_i, h_i] gives the graph display range (lower limit and upper limit) if trend display is performed with character graphs. l_i and h_i must be constants.

Character graphs can be obtained only if display ranges are specified for all variables. Even if only one display range is omitted, trend display will be performed with numerical values.

STEP *h*

Specifies a display period. h must be a constant.

If this STEP term is omitted, trend display is performed at each integral division interval that was specified by the INTEGRAL statement. If a period indivisible by the integral division interval is specified, the period is adjusted to the multiple of the nearest integral division interval.

■ VARIABLE

Variable declaration

VARIABLE can be abbreviated to VAR.

$$«\textbf{GLOBAL}» \left\{\begin{matrix}\textbf{VARABLE}\\ \textbf{VAR}\end{matrix}\right\} v_1 \;«, v_2, ...»$$

GLOBAL

Specifies the declaration of global variables.

v_1, v_2, $\cdots$

Each variable declares its name and property, being formatted as follows:

$$vname\ «(\ «m,»\ \ n)»\ «\left\{\begin{matrix}=\\ \#\end{matrix}\right\} expr»\ «"explanatory\ term"»$$

vname

A symbol that indicates a variable name.

(*m*, *n*) gives the dimension and the number of elements if those variables are array variables. If those are scalar variables, this term can be omitted. *n* and *m* must be 2 or more integer constants that give the number of elements.

expr

Constants or equations that give a value to those variables. If # is used in front of this term, it can give an initial value of variables to be integrated. The script format of these constants or equations is identical to those of general equations or # expression.

Explanatory term

An arbitrary explanatory statement that explains a variable within 40

characters. If the unit of the variable is included in the explanatory term, it

is usually enclosed
with [ ] at the end.

The explanatory term is displayed along with the prompt of calculation result output or variable value entry. Its contents are also used at graph creation.

The VARIABLE statement can be placed anywhere as long as it appears before the declared variables.

◆ Revision Record

● **Title** : **EQUATRAN Reference manual**

Nov. 1998/1st Edition

Newly published

May 2015/2nd Edition

Version 3.5 released

PreFEED Corporation

1-14-6 Nakane, Meguro-ku, Tokyo 152-0031 Japan
Tel: 81-3-5729-7180
http://www.prefeed.com