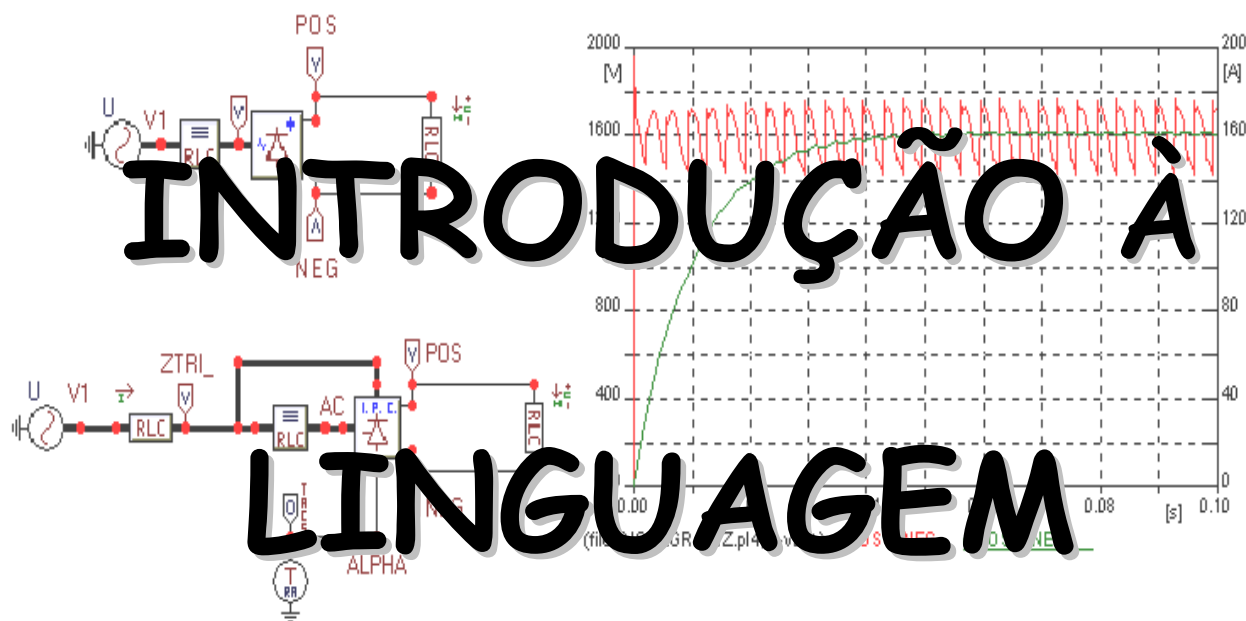
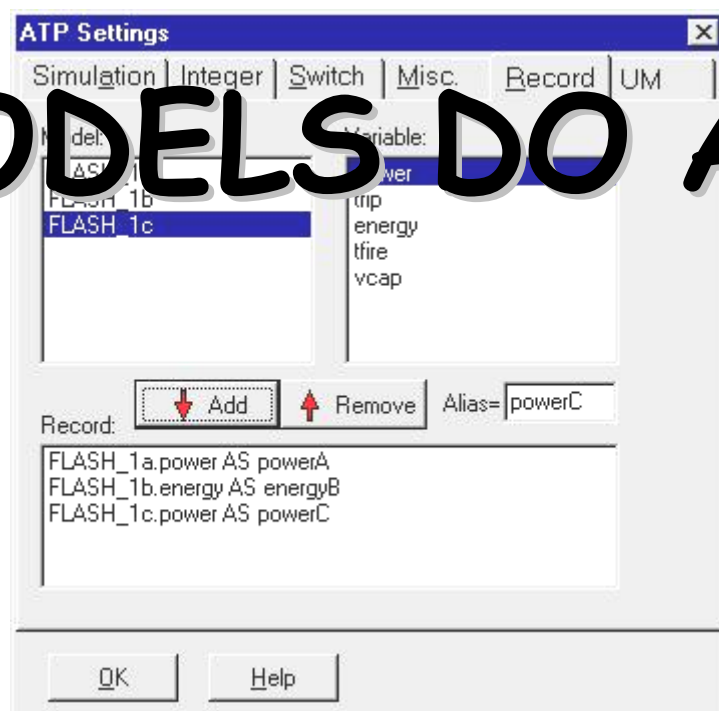




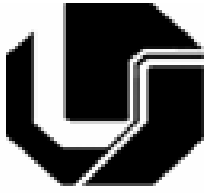
UNIVERSIDADE FEDERAL DE UBERLÂNDIA  
FACULDADE DE ENGENHARIA ELÉTRICA  
LABORATÓRIO DE QUALIDADE DA ENERGIA ELÉTRICA



# INTRODUÇÃO À LINGUAGEM MODELS DO ATP



Uberlândia – MG  
15 de Setembro de 2007



UNIVERSIDADE FEDERAL DE UBERLÂNDIA  
FACULDADE DE ENGENHARIA ELÉTRICA  
LABORATÓRIO DE QUALIDADE DA ENERGIA ELÉTRICA

# **INTRODUÇÃO A LINGUAGEM MODELS DO ATP**

**Esta apostila visa fornecer uma introdução à linguagem MODELS do software ATP. A linguagem MODELS permite, a partir de um algoritmo, modelar dispositivos de controle e de sistema que podem ser usados no ATP. A sua elaboração foi, principalmente, com base nas informações (tradução) dos documentos [1], [2] e [3] indicados na referência bibliográfica.**

**Ivandro Antonio Bacca**

**Uberlândia – MG  
15 de Setembro de 2007**

# SUMÁRIO

<b>Capítulo 1 .....</b>	<b>1</b>
1   INTRODUÇÃO.....	1
<b>Capítulo 2 .....</b>	<b>2</b>
2   OS GRUPOS PRINCIPAIS DA MODELS .....	2
2.1   DATA .....	3
2.2   CONST.....	3
2.3   VAR.....	4
2.4   HISTORY .....	5
2.5   DELAY CELLS .....	6
2.6   INIT.....	6
2.7   EXEC .....	7
2.8   USE e RECORD.....	8
2.9   UM ARQUIVO EQUIVALENTE TACS .....	8
<b>Capítulo 3 .....</b>	<b>11</b>
3   FORMATO.....	11
3.1   COMENTÁRIO .....	11
3.2   NÚMEROS, VARIÁVEIS, VETORES.....	12
3.2.1   NÚMEROS.....	12
3.2.2   VARIÁVEIS.....	13
3.2.3   VETORES .....	13
3.3   DELIMITADORES .....	13
3.4   LINHAS EM BRANCO .....	14
3.5   SINAL DE IGUAL E ATRIBUIÇÃO DE VALORES .....	14
3.6   PARÊNTESES, COLCHETES E CHAVES .....	15
3.6.1   PARÊNTESES.....	15
3.6.2   COLCHETES .....	16
3.6.3   CHAVES .....	16
<b>Capítulo 4 .....</b>	<b>17</b>
4   MODELS NO ATP .....	17

4.1	INPUT .....	17
4.2	OUTPUT .....	18
4.3	Outras características do exemplo 2 .....	19

## **Capítulo 5 .....**

5	MAIS A RESPEITO DAS DECLARAÇÕES .....	23
5.1	Declaração DATA e CONST.....	23
5.1.1	DATA.....	23
5.1.2	CONST .....	24
5.2	INPUT e OUTPUT.....	25
5.2.1	INPUT.....	25
5.2.2	OUTPUT .....	26
5.3	VAR.....	26
5.4	HISTORY e DELAY CELLS .....	27
5.4.1	HISTORY.....	28
5.4.2	Diretiva HISTORY em USE.....	29
5.4.3	HISTORY EM INIT.....	30
5.5	DELAY CELLS .....	30
5.6	Procedimento de inicialização – INIT .....	31
5.7	EXPRESSÕES E FUNÇÕES.....	32
5.7.1	EXPRESSÕES NUMÉRICAS E LÓGICAS .....	32
5.7.2	FUNÇÕES NUMÉRICAS .....	34
5.8	FUNÇÕES DE DECLARAÇÃO .....	34
5.9	FUNÇÕES POINTLIST.....	35

## **Capítulo 6 .....**

6	EXPRESSÕES DE SIMULAÇÃO E FUNÇÕES .....	37
6.1	FUNÇÕES .....	37
6.1.1	EXPRESSÃO DERIVADA COM POLINÔMIO LINEAR.....	37
6.1.2	Função DERIVADA .....	37
6.1.3	FUNÇÃO INTEGRAL.....	38
6.1.4	EXPRESSÃO SUM (SOMA).....	40
6.2	EXPRESSÕES DE SIMULAÇÃO .....	41
6.2.1	EXPRESSÕES DE SIMULAÇÃO RESIDENTES .....	41

6.2.2	FUNÇÃO PREVVAL.....	41
6.2.3	FUNÇÃO HISTDEF .....	41
6.2.4	CLAPLACE e LAPLACE função de transferência .....	42
<b>Capítulo 7 .....</b>		<b>45</b>
7	ALGORITMO DE CONTROLE .....	45
7.1.1	IF – ENDIF (SE) .....	45
7.1.2	DO – REDO (FAÇA – REFAÇA) .....	46
7.1.3	WHILE – DO – ENDWHILE (ENQUANTO – FAÇA) .....	47
7.1.4	FOR – DO – ENDFOR (PARA – FAÇA) .....	48
7.2	COMBINE (COMBINAR).....	51
7.3	TIMESTEP .....	53
7.4	RECORD – WRITE (REGISTRO – ESCREVER).....	53
7.4.1	RECORD .....	53
7.4.2	WRITE .....	54
7.5	MULTIPLE MODELS .....	54
<b>Capítulo 8 .....</b>		<b>58</b>
8	VERSÃO MODELS DE FONTES E DISPOSITIVOS DA TACS .....	58
8.1	INTRODUÇÃO.....	58
8.2	MODELOS EQUIVALENTS DE TACS E FONTES DE EMTP .....	58
8.2.1	SOURCE 11 - ‘LAVEL SIGNAL AND SINGLE PULSE’ .....	59
8.2.2	SOURCE 14 - ‘COSINE’ .....	59
8.2.3	SOURCE 23 - ‘PULSE TRAIN’ .....	60
8.2.4	SOURCE 24 - ‘RAMP TRAIN’ .....	60
8.2.5	SOURCE 13 - ‘TWO SLOPE RAMP’ .....	61
8.2.6	SOURCE 15 - ‘DOUBLE EXPONENCIAL SURGE’ .....	61
8.3	MODELOS EQUIVALENTS DE DISPOSITIVOS DE TACS.....	62
8.3.1	DEVICE 50 - ‘FREQUENCY METER’ .....	62
8.3.2	DEVICE 51, 52 - ‘RELAY SWITHCH’ .....	63
8.3.3	DEVICE 54 - ‘PULSE DELAY’ .....	63
8.3.4	DEVICE 55, 56, 57 - ‘DIGITIZER, POINT BY POINT NONLINEARITY, TIME SEQUENCED SWITCH’ .....	64
8.3.5	DEVICE 61 - ‘SIGNAL SELECTOR’ .....	66

8.3.6	DEVICE 63 - 'INSTANTENOUS MAXIMUM/MINIMUM .....	67
8.3.7	DEVICE 64 - 'MAXIMUM/MINIMUM TACKING' .....	68
8.3.8	DEVICE 65 - 'ACCUMULATOR – COUNTER' .....	69
8.3.9	DEVICE 66 - 'RMS VALUE' .....	70
REFERÊNCIA BIBLIOGRÁFICA .....		71
APÊNDICE A - SINTAXE DO IDIOMA DE MODELOS .....		72
APÊNDICE B – PALAVRAS-CHAVES DA LINGUAGEM MODELS .....		82
APÊNDICE C – CONSTANTES PRÉ-DEFINIDAS E VARIÁVEIS .....		88
APÊNDICE D – FUNÇÕES PRÉ-DEFINIDAS .....		92

# Capítulo 1

## 1 INTRODUÇÃO

MODELS é um idioma de descrição de uso geral apoiado por um jogo de ferramentas de simulação para a representação e estudo de sistemas que apresenta variação de tempo. O idioma MODELS provê um formato que focaliza na descrição da estrutura de um modelo e na função de seus elementos.

É pretendido que a descrição de um modelo está ego - documentando e podem ser usados ambos então como o documento de descrição usado por representar o sistema e como os dados usados na simulação atual.

Um sistema pode ser descrito em MODELS como um arranjo de submodels inter-relacionado, independente de um ao outro na descrição interna (por exemplo, selecionando os nomes das variáveis deles/delas) deles/delas e na simulação (por exemplo, selecionando o tamanho do passo de tempo de simulação individual deles/delas) deles/delas.

Também há uma distinção clara em MODELS entre a descrição de um modelo e o uso de um modelo. Podem ser desenvolvidos modelos individuais separadamente, agrupados em uma ou mais bibliotecas de modelos e usado em outros modelos como blocos independentes na construção de um sistema.

Finalmente, a descrição de cada modelo tem um livre-formato, sintaxe palavra chave-acumulada de contexto local, e não requer formato fixo em sua representação.

# Capítulo 2

## 2 OS GRUPOS PRINCIPAIS DA MODELS

A linguagem MODELS é usada como parte de um arquivo de dados do ATP-EMTP.

A seção MODELS de um arquivo de dados do ATP deve ter pelo menos três grupos MODELS principais de códigos, cada um introduzido por uma palavra-chave, bem como as palavras chaves ou palavra pedida no EMTP. Os três grupos são encabeçados por MODELS, MODEL e USE. A palavra-chave MODELS começa um arquivo de dados da MODELS no ATP. Os grupos MODEL e USE são terminados por ENDMODEL e ENDUSE, respectivamente. Em seguida um quarto grupo, opcional, encabeçado pela palavra-chave RECORD pode ser inserido para mostrar a saída e plotar as variáveis. A parte MODELS de um arquivo de dados do ATP é terminada pela palavra-chave ENDMODELS. MODELS é inserida no arquivo ATP exatamente da mesma maneira que a TACS, seguindo a mesma estrutura dos miscellaneous. A estrutura principal da MODELS é:

```

MODEL model_name
--      comments
    DATA --      declaration
    .... entries
    CONST --      declaration
    .....entries
    HISTORY -- directive
    ..... entries
    DELAY CELLS --directive
    ..... entries
    --      comments
    VAR   -- declaration
    ..... entries
    INIT  -- procedure
    ..... entries
    ENDINIT
--      comments
    EXEX -- procedure
    .... model simulation algorithm
    ENDEXEX
ENDMODEL

```



## 2.1 DATA

DATA é a declaração para a primeira seção da MODEL. O arquivo entrado aqui permite ao usuário atribuir valores aos elementos. Estes valores podem ser atribuídos no grupo USE, sem ter a mudança da MODEL. No exemplo 1/M o arquivo entrado é:

```
DATA
    tperiod { dflt: 0.0 }
    ton      { dflt: 0.0 }
```

Onde ‘tperiod’ e ‘ton’ são parâmetros do impulse train, ‘vin’, entrada para o primeiro bloco no modelo. Note que estes parâmetros são escritos abaixo da diretiva DATA no grupo USE com o valor atual a ser usado, tperiod = 50ms e ton = 30ms.

Note que o grupo USE não contém uma seção DATA, o valor padrão entrado na MODEL mal seja usado. Se o usuário quer mudar o valor padrão em DATA deve proceder da seguinte forma:

```
DATA
    tperiod := 0.05
```

Abaixo temos outros arquivos onde foi utilizada a declaração DATA.

<pre>DATA n          -- an array dimension used in the model DATA freq {dflt: 50} -- providing a default value</pre>
----------------------------------------------------------------------------------------------------------------------

## 2.2 CONST

CONST é a declaração para a segunda seção da MODEL. O arquivo entrado é uma constante fixada no modelo, e não pode ser modificada dentro do grupo USE. No exemplo 1/M a entrada é:

```
CONST
    kt{val: 0.05}
    kgain { val:2.0}
```

Onde ‘kgain’ é o ganho da função de transferência do primeiro bloco, e ‘kt’ é o coeficiente do termo S no denominador.

A seguir temos outro arquivo que utilizou a declaração CONST.

```
CONST twopi {val: 2*pi}
CONST power_of_2[0..8] {val: [1,2,4,8,16,32,64,128,256]}
```

## 2.3 VAR

VAR é a declaração para a próxima seção da MODEL. Esta seção serve para declarar todas as variáveis usadas no grupo MODEL, exceto para aqueles declarados em INPUT, DATA e CONST. Diferente da TACS, MODELS não identifica automaticamente as variáveis utilizadas, e omissão de incorporá-la abaixo de VAR ou outra diretiva listada. No exemplo 1/M as seguintes variáveis são declaradas:

```
VAR vin, vout, vdel, tcount, dela
```

Onde:

- ⇒ ‘vin’ é o sinal de entrada;
- ⇒ ‘vout’ é a saída do primeiro bloco;
- ⇒ ‘vdel’ é o atraso da saída;
- ⇒ ‘dela’ é o delay de tempo no bloco delay;
- ⇒ ‘tcount’ é contador para geração do sinal de entrada pulse train.

A seguir temos outro arquivo no qual utilizou a declaração VAR.

```
DATA n          -- number of terminals
INPUT vterm[1..n] -- value of measured terminal voltages
    vref {dflt: 0} -- value of a reference voltage, =0 if not used
VAR vtmax[1..n]  -- tracked maximum absolute value of each terminal voltage
    vmax         -- tracked maximum absolute value of all terminal voltages
```

```

INIT
...
vtmax[1..n]:=0    -- initialize tracked maximum values
vmax:=0          -- initialize overall tracked maximum value
...
ENDINIT
EXEC
...
FOR i:=1 TO n DO      -- for each terminal
    vtmax[i]:= max(vtmax[i], abs(vterm[i])) -- update tracked maximum
ENDFOR
vmax:= max(vmax, vtmax[1..n]) -- update overall tracked maximum
...
ENDEXEC

```

## 2.4 HISTORY

HISTORY é a diretiva para a próxima seção. Um número de funções simuladas exige o conhecimento da história passada das variáveis envolvidas. Um exemplo óbvio é o delay de um sinal. No exemplo1, as variáveis de entrada e saída da Transformada de Laplace e a função do sinal de delay requerem o mantimento da trilha dos valores de tempo passado. Consequentemente, HISTORY é requerida para as seguintes variáveis:

```

HISTORY
    vin {dflt: 0}
    vout {dflt: 0}
    vdel {dflt: 0}

```

A seguir temos outra forma na qual foi utilizada da declaração HISTORY.

```
y := delay(x, delaysize)
```

```

HISTORY y1
    y2 {dflt: 0}
    y3 {dflt: A*sin(omega*t)}

```

```
HISTORY integral(y) {dflt: expression}
```

```

HISTORY y[n1..n2] {dflt: expression}      -- same value for each element
HISTORY y[n1..n2] {dflt: array expression} -- individual value for each element

```

```
HISTORY y[n1..n2] {dflt: [expr1, expr2, expr3, ... ]}
```

```
HISTORY y[n1..n2]
```

```
HISTORY y[n1..n2] := [expr1, expr2, expr3, ... ]
```

## 2.5 DELAY CELLS

A diretiva DELAY CELLS instrui MODELS para por de lado um número de celas de memória da MODELS para recordar os valores passados das variáveis. Isto pode ser feito em declaração de valor padrão para todas as variáveis declaradas sob HISTORY, ou para variáveis individuais. No exemplo 1/M:

```

DELAY CELLS  DFLT: 10
              CELLS (vout) : 100
              CELLS (vdel) : 100

```

Um outro exemplo da utilização da declaração DELAY CELLS esta ilustrada abaixo:

<pre> DELAY CELLS DFLT:  max(delay1, delay2)/timestep +1 CELLS(y1, y2): 50 CELLS(y3): 10 </pre>
-------------------------------------------------------------------------------------------------

<pre> DELAY CELLS DFLT:  max(delay1, delay2)/timestep +1 CELLS(y1, y2): 50 CELLS(y3): 10 </pre>
-------------------------------------------------------------------------------------------------

## 2.6 INIT

INIT é o procedimento para inicializar a seção, terminando pela palavra-chave ENDINIT. Todas as variáveis usadas na seção EXEC da MODEL devem ser inicializadas, exceto para as variáveis declaradas em DATA, HISTORY, CONST e INPUT. Nenhuma inicialização de uma variável pode produzir resultados incorretos.

Em adição para inicialização, a seção INIT também reserva ao usuário para definir constantes em termos de constantes de entrada ou definidas nas seções DATA e CONST. O primeiro tempo usado na MODELS pode inicializar todas as variáveis usadas em EXEC, exceto as entradas nas seções DATA e CONST, para evitar qualquer erro potencial. No exemplo 1/M:

```

INIT
    tcount := 0.0
    dela   := 0.010
ENDINIT

```

## 2.7 EXEC

EXEC é a palavra-chave que define a seção onde a simulação é executada pela simulação do algoritmo. Esta seção deve ser terminada pela palavra-chave ENDEXEC.

No exemplo 1/M é ilustrado um sistema simples de controle consistindo de dois blocos em série. O primeiro bloco é 'kgain/ (1+kts)' com uma entrada 'vin' e uma saída 'vout'. A saída do primeiro bloco é alimentada através de um delay de 10ms, dando uma saída final 'vdel'.

A entrada 'vin' é um "trem de pulso" de amplitude 100, começando em  $t = 20\text{ms}$ , um período de tempo de 50ms e um pulso ON de 30ms. O primeiro bloco de controle é descrito por MODELS 'Laplace function':

```

laplace (vout/vin) := kgain|s0/(1|s0 + kt|s1)

```

O segundo bloco, que é um simples delay, é entrado como:

```

vdel := delay(vout, dela)
      where the variable 'dela' is set to 10 ms
      in the INIT section

```

O uso de um 'IF' no bloco para a geração do trem de pulso 'vin' é explicado no exemplo. O exemplo também mostra a geração mais simples de um trem de pulso por uma expressão lógica.

As indicações disponíveis na MODELS para desenvolver um algoritmo de simulação, seus usos e sintaxes são discutidas nos próximos capítulos.

## 2.8 USE e RECORD

O grupo USE no exemplo 1/M no modelo ‘model\_1’ é usado no contexto deste arquivo de dados sob o nome ‘test’. A seção DATA no grupo USE, especifica o período do ‘pulse train’ como 50ms, o comprimento do sinal em nível alto, ‘ton’ como 30ms e o tempo de início como 20ms. Esta especificação cancela o valor padrão especificado no grupo MODEL.

O grupo RECORD no exemplo 1/M especifica ‘vin’, ‘vout’ e ‘vdel’ como três variáveis em MODEL ‘test’ para serem visualizadas como saída.

Um exemplo da utilização da declaração USE está ilustrado abaixo:

```
USE modelname AS instance_identifier
  use directives
ENDUSE
```

```
FOR i:=1 TO n DO
  ...
  USE somemodel AS someid[i]
  ...
ENDFOR
```

## 2.9 UM ARQUIVO EQUIVALENTE TACS

Para aqueles que são familiares com a TACS o exemplo 1/T mostra a parcela da TACS em um arquivo ATP, para a simulação de um bloco de controle como no exemplo 1/M. É possível fazer alguns pontos de comparação entre os dois arquivos.

## Example 1/M

```

-----BEGIN NEW DATA CASE
$CLOSE, UNIT=4 STATUS=DELETE
$OPEN, UNIT=4 FILE=mod_1.pl4 FORM=UNFORMATTED STATUS=UNKNOWN RECL=8000 !
C deltat   tmax   xopt   copt   epsilon   tolmat   tstart
  .0001    .150
C print   points   connec   s-s   minmax   vary   again   plot
  1         1         0         1         1         1         1
MODELS
C      no enty in this group
MODEL   model_1      -- the name of this model
DATA
  tperiod { dflt: 0.0}, ton { dflt : 0.0}, tstart { dflt: 0.0
CONST
  kt {val: 0.05}, kgain { val: 2.0}
VAR vin, vout, vdel, tcount, dela
HISTORY
  vin {dflt: 0}, vout dflt: 0 } vdel {dflt: 0}
DELAY CELLS DFLT: 10
  CELLS (vout): 100
  CELLS (vdel): 100
INIT
  tcount := 0
  dela := 0.010
ENDINIT
EXEC
  IF t> tstart THEN
    -- generate a train of pulses starting at t= tstart
    --
    tcount := tcount + timestep
    -- reset counter at end of on/off period
    IF tcount > tperiod THEN tcount := 0  ENDIF
    -- vin 100 or 0 depending time within period
    IF tcount < ton THEN
      vin := 100.0
    ELSE vin := 0.0
    ENDIF
    --
    -- a more compact generation of a train of pulses is given by :
    -- vin:= 100 * ( (t - tstart) MOD tperiod < ton)
    --
    -- calculate output from k/(a+bs) block
    laplace (vout/vin) := kgain|s0/(1|s0 + kt|s1)
    -- delay output
    vdel := delay(vout, dela)
  ENDIF
ENDEXEC
ENDMODEL

C -----
USE model_1 AS test
DATA tperiod := 0.05
  ton := 0.03
  tstart := 0.02
ENDUSE
C
RECORD
  test.vin AS vin
  test.vout AS vout
  test.vdel AS vdel
ENDMODELS
C BRANCH CARDS  EMTF  data section
SRCE 10.
BLANK CARD ENDING BRANCHES
C no switches
BLANK CARD ENDING SWITCHES
14SRCE 1.0 50.
BLANK CARD ENDING SOURCES
SRCE
BLANK CARD ENDING NODE VOLTAGE REQUEST
BLANK CARD ENDING PLOT
BEGIN NEW DATA CASE
BLANK

```

### Example 1/T

```

-----
BEGIN NEW DATA CASE
$CLOSE, UNIT=4 STATUS=DELETE
$OPEN, UNIT=4 FILE=tacs_1.pl4 FORM=UNFORMATTED STATUS=UNKNOWN RECL=8000 !
C deltat      tmax      xopt      copt      epsiln      tolmat      tstart
  .0001      .150
C print      points      connec      s-s      minmax      vary      again      plot
  1          1          0          1          1          1          1          1
TACS HYBRID
C
99DELA      = 0.010
23VIN      100.0      0.050      0.030      0.020
C
1VOUT      +VIN      2.0
  1.0
  1.0      .050
C
98VDEL 53+VOUT      0.011 DELA
C
C      initializations
77VIN      0.0
77VOUT1      0.0
77DELA      .010
C
C      ***** TACS OUTPUTS *****
C
33VIN      VOUT      VDEL
BLANK      end of TACS
C BRANCH CARDS EMTP data section
  SRCE      10.
BLANK CARD ENDING BRANCHES
C no switches
BLANK CARD ENDING SWITCHES
14SRCE      1.0      50.
BLANK CARD ENDING SOURCES
  SRCE
BLANK CARD ENDING NODE VOLTAGE REQUEST
BLANK CARD ENDING PLOT
BEGIN NEW DATA CASE
BLANK

```



# Capítulo 3

## 3 FORMATO

A sintaxe da MODELS reserva um estilo livre de formatação. Nenhum das palavras-chaves, declarações ou indicações devem ser especificados em um local certo de coluna, como na TACS. As entradas não são diferenciadas em maiúsculas e minúsculas, mas é recomendável que as palavras-chaves sejam especificadas com letras maiúsculas, como MODEL, INPUT, VAR, USE, etc. fazendo com que a estrutura do arquivo apresente certa facilidade de editar e ler. Somente existe a restrição que não é possível deixar linhas em branco, uma vez que isso será rejeitado pelo ATP.

### 3.1 COMENTÁRIO

O usuário pode, entretanto colocar uma linha em branco utilizando a letra ‘C’ na coluna 1 (um).

Existem três caminhos para entrar com linhas de comentário:

- ⇒ Tipo ‘C’ na primeira coluna seguido por uma linha em branco;
- ⇒ Tipo ‘--’ em qualquer lugar da linha seguido por um comentário;
- ⇒ Utilizando a palavra-chave COMMENT seguida de um comentário, que pode estar compreendidos em diversas linhas. A palavra-chave ENDCOMMENTS é utilizada para terminar a seção COMMENT.

O exemplo seguinte ilustra a utilização dos comandos para a inserção de um comentário:

```

C      this is a comment
or
--   this is a comment
--   this is a continuation of the comment
or

COMMENT
    this is the first comment line
    this is the second comment line
    etc. etc.
ENDCOMMENT

```

Uma outra forma de se utilizar os comentários está ilustrado abaixo:

```

MODEL m1

comment -----
| This is a block comment describing this model      |
| (the vertical bars and the hyphens are only decorative) |
| ----- endcomment
VAR a  -- this is an in-line comment describing this variable

```

## 3.2 NÚMEROS, VARIÁVEIS, VETORES.

### 3.2.1 NÚMEROS

Um número (valor numérico) tem que ser uma série de dígitos decimais, com sinal positivo ou negativo. O primeiro caractere de um número deve ser um dígito.

Para a notação científica, o valor numérico é seguido, sem um espaço, pela letra 'E' seguida do valor do coeficiente inteiro. O seguinte exemplo esclarece isto.

```

12, -3, 0.95, -12.44, -0.4
0.3E-1, -3.6E+9, 4E9, 3.E-04

```

Note que um número com a condição ponto .4 não deve ser especificado e sim utilizando um 0 (zero) antes, isto é, 0.4.

### 3.2.2 VARIÁVEIS

Variáveis são consultadas por seus nomes. Nomes são para ser digitados como um vetor de caracteres contínuos. O primeiro caractere de um nome não deve ser um dígito. Qualquer outro caractere que tem uma letra, dígito ou outra forma, é interpretado como final de um nome. Um nome pode ser de qualquer comprimento, não é limitado em 6 (seis) caracteres como no EMTP. Ao passar nomes para o EMTP ou para plotar via o RECORD, a limitação de seis caracteres se aplica a:

```
John
John_and_Mary
test_example
```

### 3.2.3 VETORES

As variáveis dos vetores são especificadas pelo nome do vetor, seguida pela faixa de variação do vetor especificada dentro dos colchetes, como mostra o exemplo a seguir:

<code>y[3]</code>	the third element of an array
<code>y[1..3]</code>	an array with three elements
<code>y[1..n]</code>	an array with n elements

## 3.3 DELIMITADORES

A linguagem MODELS utiliza espaços em branco, vírgulas, ponto e vírgula, ‘carriage return’ e ‘line feeds’ como delimitadores ou separadores. A única exceção é o ‘carriage return’ e ‘line feeds’ que deve ser seguido de um comentário, introduzido pelo símbolo ‘--’.

Exemplos:

A declaração de variáveis usadas na MODEL na seção VAR pode ser de vários tipos como, por exemplo:

```

      VAR      a
              x1
              b[1..3]
or
      VAR      a, x1, b[1..3]
or
      VAR      a  x1  b[1..3]

```

A declaração HISTORY das variáveis pode ser das seguintes formas:

```

      HISTORY  y{dflt:0}
              x{dflt:sin(omega*t)}
or
      HISTORY  y{dflt:0}  x{dflt:sin(omega*t)}

```

Enquanto espaços em branco são reservados como separadores (delimitadores) como mostram acima, seu uso não é recomendado porque pode conduzir a erros de datilografia que não são facilmente detectados.

### 3.4 LINHAS EM BRANCO

As linhas em branco não devem ser especificadas na linguagem MODELS, assim como acontece com os cartões do ATP. A especificação de tais linhas leva a erro da execução do algoritmo. A forma de implementar essas linhas em branco é utilizando os comandos de comentário, como já explicado acima.

### 3.5 SINAL DE IGUAL E ATRIBUIÇÃO DE VALORES

Na linguagem MODELS existe uma diferença entre sinal de igual e símbolo de atribuição, que pode causar certa confusão entre usuários do FORTRAN ou BASIC.

O símbolo de atribuição usado na MODELS é ‘:’, usado por exemplo em declarações DATA ou HISTORY, a atribuição de um valor e fazer a variável

igual a este valor é ‘ := ‘, e o para igualar uma expressão a um determinado valor é ‘ = ‘. Sendo assim, quando se quer fazer uma atribuição a uma variável deve-se utilizar o comando ‘ := ‘, já quando se quer igualar uma variável a algum valor utiliza-se o comando ‘ = ‘. Por exemplo:

```
DATA kx {dflt: 1.1}
      k2 := 20
IF a =30 THEN etc
```

A seguir temos outras formas de se utilizar esses sinais:

```
IF a = b THEN ... ENDIF
```

```
w[2] := expression
```

```
laplace(y/x) := (s_numerator)/(s_denominator)
```

## 3.6 PARÊNTESES, COLCHETES E CHAVES

### 3.6.1 PARÊNTESES

O parêntese ‘ ( ) ‘, o colchete ‘ [ ] ‘ e as chaves ‘ { } ‘ podem ser utilizadas na MODELS.

Parêntese ‘ ( ) ‘ são usados na MODELS da mesma maneira como este tipo de parêntese como são comumente utilizados em matemática. MODELS utiliza parênteses nas expressões lógicas e numéricas, funções de argumentos etc., mas não para denotar índices ou faixa de variação de vetores. Por exemplo:

```
x := y * ( a + b )
ya := sqrt (x/b)
laplace ( y/x ) := k:s0 / (1|s0 + b |s1)
xt := integral ( cos ( omega * t))
```

Parênteses circulares são também usados igualmente em algumas das indicações relativas declaração, como as abaixo da diretiva INIT.

```
histdef ( x ) : cos ( omega * t)
```

Uma outra forma de se utilizar os parênteses é:

```
y := delay(x1, delay_value)
integral(w) := expression
laplace(x1/x2) := (1|s0) / (1|s0 + tau|s1)
histdef(integral(y2)) := 2*histdef(y1)
```

```
y := a -(2*b -c)
```

### 3.6.2 COLCHETES

Colchetes ‘ [ ] ’ são usados apenas em vetores, para denotar um elemento ou faixa de variação do vetor. Por exemplo:

```
a [1..3] := 3.0 meaning that all elements of 'a'
are equal to 3.0
```

```
a [1..3] := [ 1, 2, 3] meaning
a[1], a[2], a[3] equal to 1,2,3 respectively
```

Outro exemplo da utilização dos colchetes é:

```
wabs[1..n] := abs(w[1..n])
```

```
vpp[1..3] := [va-vb, vb-vc, vc-va]
```

### 3.6.3 CHAVES

Chaves ‘ { } ’ são usados nas declarações, definindo limites ou atribuições. Por exemplo:

```
DATA xf { dflt :1.0 }
HISTORY xy { dflt : sin ( omega * t) }
y := sum ( c4| -t|a + t|b ) {min -0.5 max +1.0}
```

Uma outra forma de se utilizar as chaves é:

```
INPUT va { v(NODEA) }
CONST twopi { val: 2*pi }
DATA freq { dflt: 50 }
FUNCTION windspeed FOREIGN { ixarg: 2 }
```

# Capítulo 4

## 4 MODELS NO ATP

O modelo do exemplo 1 não teve conexão para o EMTP. A única conexão do Model\_1 para fora foi à definição dos parâmetros da função de transferência na declaração DATA no grupo USE.

No exemplo 2/M é demonstrado como MODELS é conectado para o EMTP. A rede mostra que no EMTP parte do exemplo 2 é uma fonte AC – GENA de frequência 50Hz conectada para BUSA, a chave de medição de BUSA para BUSB, um tiristor entre BUSB e BUSC e uma carga de 100MW, 10 MVar representada por um ramo  $r+jx$  entre BUSC e o terra. MODEL tem o nome Model\_2, é exigido para fazer duas coisas. Para conectar a carga através da descarga do tiristor em  $t = 50\text{ms}$ , e então calcular a potência ativa absorvida pelo resistor. Recordar que a saída tipo 4 no EMTP fornece somente o valor instantâneo da potência aparente  $S = V \cdot I$

Model\_2 determina a potência de fluxo ativo através da chave de medição localizada entre BUSA e BUSB. As variáveis para serem passadas para o EMTP são as tensões de BUSA e as correntes de BUSA para BUSB. MODELS necessita de três etapas para realizar o algoritmo.

### 4.1 INPUT

No exemplo analisado a entrada que recebe o sinal de tensão do ponto BUSA é usado como 'volt', e a corrente que é medida através da chave de medição BUSA-BUSB em BUSB pode ser usada como 'cur', como mostrado abaixo.

```
INPUT volt {v(BUSA)}
        cur {i(BUSB)}
```

MODEL tem que saber se as variáveis usadas em Model\_2 são entradas do sinal externas de Model\_2. Isto é feito pela diretiva (ou declaração) INPUT na MODEL.

```
INPUT  vv:= volt
       ii:= cur
```

No grupo USE, é declarado que as variáveis ‘volt’ e ‘cur’ importadas de uma fonte externa são usadas como variáveis ‘vv’ e ‘ii’ respectivamente.

```
INPUT  vv:= volt
       ii:= cur
```

Uma outra forma de se utilizar a diretiva INPUT é:

INPUT vterm[1..3]	-- value of measured terminal voltages
vref {dflt: 0}	-- value of a reference voltage, =0 if not used
DATA n	-- number of terminals
INPUT vterm[1..n]	-- value of measured terminal voltages
vref {dflt: 0}	-- value of a reference voltage, =0 if not used

## 4.2 OUTPUT

No exemplo 2, uma saída do Model\_2 é requerida para ser o sinal de gatilho do tiristor no EMTP. A variável para fazer isto é ‘aa’. Como para as entradas na MODEL as saídas são declaradas como segue:

- Abaixo da palavra MODELS é declarado que FIRE será usado no EMTP;
- Model\_2 é informado que a variável ‘aa’ será usada como saída da MODEL

```
OUTPUT aa
```

A diretiva OUTPUT também pode ser usada da seguinte forma:

DATA n	-- number of terminals
INPUT vterm[1..n]	-- value of measured terminal voltages
vref {dflt: 0}	-- value of a reference voltage, =0 if not used
VAR vtmax[1..n]	-- tracked maximum absolute value of each terminal voltage
vmax	-- tracked maximum absolute value of all terminal voltages
OUTPUT vtmax[1..n], vmax	-- can be used as outputs of the model



### 4.3 Outras características do exemplo 2

- ⇒ A declaração DATA tem somente uma variável 'freq', sendo pedido 'freq' em USE, abaixo de DATA;
- ⇒ Abaixo da declaração VAR, todas as variáveis usadas na MODEL são listadas com exceção das variáveis declaradas em INPUT e DATA;
- ⇒ HISTORY informa o passado das variáveis 'vv' e 'ii' para que elas possam ser requeridas na simulação;
- ⇒ Na seção INIT o valor de 'tper' e 'tmult' são definidos, que serão usados como constantes na simulação. Além disso, 'pavg' e 'aa' são inicializado com 0 (zero);
- ⇒ A declaração DELAY CELLS especifica a quantidade de células de memória para a história dos valores passados;
- ⇒ A simulação, na seção EXEC, é explicada pela codificação própria. A energia é atualizada e somada para cada passo de tempo (1/period) (u.i.) para o valor corrente de energia e subtraída da soma do produto de 't-tperiod'. O sinal para o gatilho dos tiristores no EMTP é gerado pelo bloco IF.

O exemplo 2/T mostra a versem TACS do exemplo 2/M.

### Example 2/M

```

-----
BEGIN NEW DATA CASE                                {active power calculation}
$CLOSE, UNIT=4 STATUS=DELETE
$OPEN, UNIT=4 FILE=mod_2.pl4 FORM=UNFORMATTED STATUS=UNKNOWN RECL=8000
!
PRINTED NUMBER WIDTH, 13, 2,
C deltat      tmax      xopt      copt      epsiln      tolmat      tstart
  .0001      .200      50.      0.0
C print      points      connec      s-s      minmax      vary      again      plot
  1          1          0          0          0          0          0          1
C
MODELS
  INPUT volt {v(BUSA)}
          cur {i(BUSB)}
  OUTPUT FIRE
MODEL model_2
  DATA freq {dflt: 0}
C
  VAR
    pavg, tmult
    tper,tmult, vdel, idel, aa
C
  INPUT vv {dflt: 0} ii {dflt: 0}
C
  OUTPUT aa
C
  HISTORY vv {dflt: 0} ii {dflt: 0}
C
  INIT
    tper := 1/freq
    tmult := tper/timestep
    pavg := 0
    aa := 0
  ENDINIT
C
  DELAY CELLS (vv): 200
          CELLS (ii): 200
C
EXEC
C
  calculate active power
  vdel := delay (vv, tper )
  idel := delay (ii, tper )
  pavg := pavg +( vv * ii - vdel * idel)/tmult
C
  IF t > 0.05 then aa :=1.0 endif
-- the above can be replaced with a logical expression
--   a:= t>= 0.05
C
ENDEXEC
ENDMODEL

```

## Example 2/M Continued

```

-----
USE model_2 AS test
  DATA freq := 50
INPUT
  vv:= volt
  ii:= cur
OUTPUT FIRE := aa
ENDUSE
C
RECORD test.vv      AS vv
      test.ii      AS ii
      test.pavg     AS pavg
ENDMODELS
C      EMTF      data
C      =====
C BRANCH CARDS
  GENA  BUSA          .010
  BUSC          99.01  9.9
  BUSA1 BUSC          .010
  BUSC1 BUSC          .010
BLANK CARD ENDING BRANCHES
  BUSA  BUSB
  11BUSB BUSA1        0.    0.0
  11BUSC1 BUSB        0.    0.0
BLANK CARD ENDING SWITCHES
  14GENA      141000.0    50.    0.
BLANK CARD ENDING SOURCES
  BUSB
BLANK CARD ENDING NODE VOLTAGE REQUEST
BLANK CARD ENDING PLOT
BEGIN NEW DATA CASE
BLANK

```

Note:

In the above model the execution is defined in two steps for clarity. In an actual application the following would execute faster:

```

      p      := vv * ii
      pavg := pavg * ( p - delay (p,tper) ) / tmult

```

## Example 2/T

```

-----
BEGIN NEW DATA CASE
$CLOSE, UNIT=4 STATUS=DELETE
$OPEN, UNIT=4 FILE=tacs_2.pl4 FORM=UNFORMATTED STATUS=UNKNOWN RECL=8000 !
C deltat tmax xopt copt epsiln tolmat tstart
  .0001 .150
C print points connec s-s minmax vary again plot
  1 1 0 1 1 1 1
TACS HYBRID
C
99DELA = 0.020
99FREQ = 50.0
99TPER = 1 / FREQ
99TMULT = TPER / DELTAT
90BUSA
91BUSB
0CURT +BUSB
0VOLT +BUSA
88VDEL 53+VOLT .021 DELA
88IDEL 53+CURT .021 DELA
88PAVG = PAVG1 + ( VOLT * CURT - VDEL * IDEL ) / TMULT
0PAVG1 +PAVG
11FIRE 1.0 0.050
C
C initialization
77PAVG 0.0
77PAVG1 0.0
77FIRE 0.0
C
C ***** TACS OUTPUTS *****
C
33VIN VOLT CURT PAVG VDEL IDEL
BLANK end of TACS
C EMTP data same as for Example 2/M

```

# Capítulo 5

## 5 MAIS A RESPEITO DAS DECLARAÇÕES

Alguns comentários sobre o uso de diretivas e declarações serão dadas a seguir.

### 5.1 Declaração DATA e CONST

Para o dado de entrada que não são de uma fonte externa da MODEL, que é proveniente de outra MODEL ou EMTP, pode ser entrado também abaixo de DATA ou CONST. Estas entradas podem ser números ou expressões que contém algumas das constantes já conhecidas da MODELS. A importante diferença entre DATA e CONST é que os valores entrados abaixo de DATA podem ser cancelados na diretiva DATA no grupo USE, enquanto que os valores entrados em CONST não podem ser modificados externamente.

#### 5.1.1 DATA

A seguinte entrada são aceitáveis como arquivos de DATA:

```
DATA frequency
or
DATA frequency {dflt:0}
or
DATA omega {dflt:2*pi*50}

or for an array
DATA volt[1..3] {dflt: [va,vb,vc]}
```

No exemplo acima, 'pi' é uma constante residente (já definida na linguagem MODELS), cujo valor é 3.14...

Se o usuário desejar usar no modelo diferentes frequências, ele pode fazer da seguinte forma:

```

MODELS
MODEL test
  DATA frequency {dflt:0}
  DATA omega {dflt: 2*pi*frequency}
  -- etc
  -- description of model
ENDMODEL
USE test AS test
  DATA frequency := 50
  -- etc.
ENDUSE
ENDMODELS

```

Note que ‘frequency DATA’ ou ‘DATA frequency’ {dflt: 0} são entradas equivalentes. É recomendado que a especificação do valor padrão {dflt:0} seja sempre usada para relembrar o valor usado em USE –DATA.

Uma outra forma de se utilizar a diretiva DATA é:

<pre> DATA n                -- an array dimension used in the model DATA freq {dflt: 50} -- providing a default value </pre>
------------------------------------------------------------------------------------------------------------------------------

### 5.1.2 CONST

A sintaxe para entrada de uma constante em CONST é:

```

CONST k1 {val : 50}
or
CONST id {val : pi*20.3}
or for an array
CONST
  a[1..3] {val: [ 20, 30, 40 ] }

```

Os valores das constantes declaradas em CONST não podem ser modificados externamente. A declaração CONST não inclui uma variável declarada em outra declaração ou sinal externo da MODEL. As constantes residentes são:

```

pi    = 3.14...
false = 0,  true  = 1
no     = 0,  yes   = 1
open   = 0,  closed = 1
off    = 0,  on    = 1

```

Uma outra forma de se aplicar a diretiva CONST é:

```
CONST twopi {val: 2*pi}
CONST power_of_2[0..8] {val: [1,2,4,8,16,32,64,128,256]}
```

## 5.2 INPUT e OUTPUT

### 5.2.1 INPUT

Sinais INPUT para a MODEL são variáveis declaradas que são informadas para a MODEL como um sinal externo da MODEL, proveniente de outro modelo ou do EMTP. Sua sintaxe é:

```
INPUT  volt {dflt: 0}
                                current {dflt: 0}
or
INPUT  volt
      current
```

Vamos supor que o modelo preciso de uma entrada ‘volts’ e ‘current’, o nome do modelo é ‘Mary’. As variáveis do EMTP que podem ser usadas por ‘Mary’ são chamadas ‘voltout’ e ‘curout’ para serem usadas, respectivamente, como ‘volt’ e ‘current’. As declarações para ‘Mary’ tem o seguinte formato:

```
MODELS
  INPUT voltout    {v(BUSA)}  -- BUSA from
EMTP
  curout          {i(BUSB)}  -- BUSB from EMTP
MODEL Mary
  INPUT volt      {dflt:0}
      current {dflt:0}
  .... other declarations
EXEC
  ..... simulation
ENDEXEC
ENDMODEL
USE Mary as Mary
  INPUT volt      := voltout
      current := curout
  .... other directives
ENDUSE
ENDMODELS
```

### 5.2.2 OUTPUT

A declaração OUTPUT informa a MODEL que a variável pode ser passada para outra MODEL ou EMTP. Assim como INPUT, a declaração OUTPUT dentro da MODEL tem uma correspondência com a diretiva USE, onde o usuário pode definir o nome que a saída será passada para outra MODEL.

No exemplo abaixo é assumido que a variável ‘fia’ e ‘fib’, geradas no modelo ‘John’ são passadas para a MODEL ‘Mary’ com os nomes ‘ftxa’ e ‘ftxb’. Os dois modelos são usados em algum arquivo MODELS. O procedimento é:

```
MODELS
MODEL John
....
OUTPUT fia, fib
-- rest of model John
ENDMODEL
USE John AS John
OUTPUT ftxa := fia ftxb := fib
ENDUSE
MODEL Mary
INPUT ff1 ff2
-- rest of model Mary
ENDMODEL
USE Mary AS Mary
INPUT ff1 := ftxa ff2:= ftxb
-- etc.
ENDUSE
ENDMODELS
```

Deve ser notado que as variáveis que aparecem nas diretivas INPUT e OUTPUT e qualquer grupo MODEL de um arquivo MODELS.

## 5.3 VAR

MODELS requer que todas as variáveis sejam declaradas uma vez, mas não mais que uma vez. A não declaração ou declaração incorreta de uma variável resulta em erros. Variáveis podem ser declaradas em DATA, CONST e



INPUT. Variáveis usadas na MODEL, mas não declaradas nestas seções devem ser entradas nas declarações como VAR. O nome das variáveis de saída como usados em outros modelos ou externamente, não será declarado em VAR. A sintaxe da declaração VAR é mostrada no seguinte exemplo:

```
VAR y, x, a[1..3], ba[1..n]
```

MODELS tem um número de variáveis residentes. Frequentemente as seguintes variáveis são usadas:

- ⇒ t - valor presente do tempo de simulação em segundos;
- ⇒ prevtime - valor prévio do tempo de simulação em segundos
- ⇒ timestep - intervalo de ponta da simulação em segundos;
- ⇒ starttime - tempo inicial da simulação em segundos;
- ⇒ stoptime - tempo final da simulação em segundos

Outra forma de se utilizar a diretiva VAR é:

```
DATA n                -- number of terminals
INPUT vterm[1..n]    -- value of measured terminal voltages
      vref {dflt: 0}  -- value of a reference voltage, =0 if not used
VAR vtmax[1..n]      -- tracked maximum absolute value of each terminal voltage
      vmax           -- tracked maximum absolute value of all terminal voltages
```

```
INIT
...
vtmax[1..n]:=0        -- initialize tracked maximum values
vmax:=0              -- initialize overall tracked maximum value
...
ENDINIT
EXEC
...
FOR i:=1 TO n DO      -- for each terminal
  vtmax[i]:= max(vtmax[i], abs(vterm[i])) -- update tracked maximum
ENDFOR
vmax:= max(vmax, vtmax[1..n]) -- update overall tracked maximum
...
ENDEXEC
```

## 5.4 HISTORY e DELAY CELLS

Existem algumas funções na MODELS, assim como na TACS, que precisam primeiramente conhecer a história das variáveis para então executarem o passo de tempo. Estas variáveis podem precisar para de certo passo de tempo para sua evolução. HISTORY não é gerada pela MODELS sem a intervenção do

usuário como acontece com a TACS. O usuário tem que definir a história requerida pela variável.

### 5.4.1 HISTORY

MODELS tem três caminhos para informar uma história. Na diretiva HISTORY na MODEL, na diretiva HISTORY em USE ou a atribuição da história com 'histdef' em INIT na MODEL.

MODELS nomeiam uma prioridade a estas tarefas de história. Primeiro o diretiva de HISTORY em MODEL é esquadrinhado, então o diretiva de HISTORY em uso que anula as definições na diretiva de HISTORY em MODEL. Terceiro, são examinadas as tarefas de histdef em INIT, que anulam as duas tarefas prévias. A primeira vez de MODELS de tempo não deveria usar a tarefa de histdef que pode causar confusão e raramente seria requerida.

História deve ser definida para variáveis em funções de demora, integrais, Funções de Laplace, equações diferenciais, etc. A exigência deveria ser óbvia do uso de uma variável. Onde o usuário pode não estar claro na exigência, história pode ser definida como uma precaução. Se for omitido quando exigido, MODELS enviarão uma mensagem de MATANÇA para ensinar que o usuário sobre a definição de história.

A sintaxe da definição da história é:

```
HISTORY y2 { dft:0 }
(HISTORY y2 can be used only if the history
of y2 is in the USE section. The first
time MODELS user should avoid this)
or
HISTORY y {dflt : ampl*omega*t)}
( with omega entered under DATA or CONST )
or
HISTORY a[1..4] {dflt: 0}
or
HISTORY volt[1..3] {dflt : [va,vb,vc]}
(with va,vb,vc imported by IMPUT)
```

É importante notar aquele cuidado extremo deveria ser exercitado em história definível em termos de outra função, ou até mesmo em termos da mesma função. Neste caso a história passada da função para a qual história é declarada pode ser substituída com a história da outra função. Por exemplo:

```
HISTORY y {dflt: y*cos(omega*t)}
      or
HISTORY y {dflt: x }
      x {dflt: sin(t)}
```

Causará a história de y para em cima de - escrito por y.cos (ômega.t) ou x.sin (t) respectivamente. Isto é, na maioria dos casos, não é o que o usuário quer fazer. É por isto que na maioria de casos HISTORY de y {dflt:0} é a declaração segura para usar. O usuário avançado deveria estudar o manual para aplicações mais sofisticadas.

Uma outra forma de se utilizar a diretiva HISTORY é:

```
histdef(y) := expression written as a function of the variable "t"
```

```
histdef(y) := 2 * histdef(x)
```

```
histdef(y) := 2 * x
```

#### 5.4.2 Diretiva HISTORY em USE

Como notado sobre HISTORY declarada em MODEL pode ser reentrada em USE e será a história administrativa. O benefício para o usuário é que o próprio MODEL precisa que não seja mudado pelo usuário, ao re-usar um modelo particular. A declaração de história segue as mesmas regras em uso como a declaração de história debaixo de MODEL, exceto a sintaxe é diferente e não há nenhuma opção de default. A sintaxe é:

```
HISTORY y := 0
        y := 2*sin(t)
```

### 5.4.3 HISTORY EM INIT

Ambas as declarações de história em MODEL e em USE é substituído pela declaração de história em INIT. Como notado acima, esta não é uma opção freqüentemente usada e deveria ser evitado pela primeira vez de MODELS de tempo. A sintaxe é

```
histdef (y) := sin(omega*t)
histdef (x) := 0
```

## 5.5 DELAY CELLS

De mãos dadas com a declaração HISTORY vai à declaração de DELAY CELLS. Em TACS a tarefa de celas para história passada é interna a ATP contanto que o tamanho de mesas de TACS não seja excedido em qual embala uma MATANÇA diz para o usuário o que fazer. Em MODELS, o usuário tem que contar ATP quantas celas de memória de MODELS são requeridas. Esta exigência é governada pelo delay máximo e, é especificada em qualquer função de delay usado na MODEL, tipicamente  $y = \text{delay}(x, td)$  onde  $td$  é o delay em segundos. Por exemplo:  $y = \text{delay}(x, 0.03)$  que é pedido o valor de  $x$  de cada vez 30 ms atrás do tempo de simulação atual, ou  $y = x(t - td)$ . O número de celas de memória que a MODELS tem que apartar para a simulação é então 30 ms divididos pelo passo usado na simulação. Se o passo de tempo for 0.1 ms, então no anterior exemplo o número de celas de demora requerido são 300.

DELAY CELLS podem ser pedidas individualmente para uma variável, ou globalmente por uma declaração de falta para todas as variáveis para as quais são requeridas delay cells. Formas alternativas de declaração são:

```

DELAY CELLS DEFAULT: 10
      or
DELAY CELLS y: 200
      CELLS x: 100
      or
DELAY CELLS DEFAULT :2
      CELLS y: tper/timestep
      CELLS x: 300

```

Note que o valor padrão especificado aplica a todas as variáveis que requerem delay cells, mas para qual nenhum número específico de células é entrado.

Uma outra forma de se utilizar a diretiva DELAY CELLS é:

<pre> DELAY CELLS DFLT:    max(delay1, delay2)/timestep +1 CELLS(y1, y2): 50 CELLS(y3):    10 </pre>
------------------------------------------------------------------------------------------------------

## 5.6 Procedimento de inicialização – INIT

Inicialização de variáveis é realizada no procedimento INIT. Esta seção é encabeçada pela palavra chave INIT e terminada por ENDINIT. MODELS requerem que todas as variáveis usadas na simulação sejam inicializadas antes da execução do algoritmo. MODELS não inicializará para zero.

Isto é feito na seção de INIT, com exceção das variáveis definidas em DATA, CONST e HISTORY. Inicialização pode ser feita em termos de uma constante entrada em DATA ou CONST como mostrada em Exemplo 2/M.

Inicialização de uma variável pode estar nomeando um valor (inclusive zero) simplesmente à variável ou nomeando uma expressão ou história que usa o histdef como notada acima. A primeira vez o usuário de MODELS deveria usar só tarefa de valores numéricos.

Por exemplo:

```

INIT
  y1 := 0
  x1:= 5
  Z := omega*t
  histdef(uv) := 3*t
ENDINIT

```

No anterior exemplo o valor de 'ômega' ou foi declarado debaixo de DATA ou CONST.

## 5.7 EXPRESSÕES E FUNÇÕES

O tipo seguinte de funções é usado na MODEL

- ⇒ Expressões Numéricas e lógicas e funções;
- ⇒ Funções de Simulação;
- ⇒ Funções Estrangeiras;
- ⇒ Funções Externas.

### 5.7.1 EXPRESSÕES NUMÉRICAS E LÓGICAS

Expressões numéricas e lógicas são bem parecidas a essas usadas por FORTRAN ou BASIC. Por exemplo:

```

Numerical expressions:
  x := a * b
  x := a ** -2.21
  x := ( a + b ) * ( c + d ) / x ** 2
  etc.

```

Note o uso novamente do ' := ' (símbolo de tarefa) indicando que a expressão dá à direita que lado é 'nomeou a x.' (No sentido exato, em termos do idioma de MODELS, os anteriores não são expressões mas expressões de tarefas).

Uma expressão numérica é seguido por um limite posto dentro ' { } ' seguindo a expressão como segue:

```

x := ( a + b ) * ( c + d ) { min:-0.76 }
or
v := 2 * omega*t { min: 0.95  max: 1.8 }
or
y := ( a + x ) { max:0.6 }

```

Logical expressions:

```

y := a > b
y := a > b or c < a

```

or in the Boolean form

```

y := bool (x)
y := AND (a, b, c)
y := NOR (a, b, c )
etc.

```

```

for true y = 1, otherwise y = 0

```

Expressões lógicas e numéricas podem ser combinadas como no exemplo seguinte que mostra um trem de pulsos:

```

y:= 100*(fract(t-tstrt)/period) <= ton/period)

```

A seguir temos a principais funções lógicas:

<b>logical binary op</b>	OR AND > >= < <= = <>	<i>true</i> if either member is <i>true</i> <i>true</i> only if both members are <i>true</i> <i>true</i> if left greater than right <i>true</i> if left greater than or equal to right <i>true</i> if left smaller than right <i>true</i> if left smaller than or equal to right <i>true</i> if left equal to right <i>true</i> if left not equal to right
<b>numerical binary op</b>	+, - *, / ** MOD	addition and subtraction multiplication and division exponentiation modulo (division remainder)
<b>logical unary op</b>	NOT	<i>true</i> if member is false or <=0
<b>numerical unary op</b>	-	negative of member

### 5.7.2 FUNÇÕES NUMÉRICAS

Funções numéricas em MODELOS, como expressões, são muito iguais às funções numéricas em Fortran ou Basic.

Existem algumas funções que são próprias da MODELS. As funções residentes seguintes estão disponíveis:

```
abs(x),  sqrt(x),  exp(x),  ln(x),  log10(x),
log2(x)
recip(x),  factorial(x),  trunc(x),  fract(x),
round(x)
sign(x),  rad(x),  deg(x),  random(),  sin(x),
cos(x)
tan(x),  asin(x),  acos(x),  atan(x),  sinh(x),  cosh(x)
asinh(x),  acosh(x),  atanh(x),  atan2(x1,x2),
binom(n,r)
permut(n,r),  x1 mod x2

min(x1, x2, x3, .....),  max(x1, x2, x3, ....)
norm( x1, x2, x3.....),  ( sqrt of squares of x1,
x2, x3... )
```

## 5.8 FUNÇÕES DE DECLARAÇÃO

O usuário pode definir uma função e usar isto como no exemplo simples seguinte:

```
FUNCTION Cathy (a1, a2, b) := (1.3*a1 + a2**2) / b
used as
      y:=  Cathy (x1, x2, x3)

      note that the order of x1, x2, x3 has to be the
      same as a1, a2, b.
```

Os argumentos da FUNCTION são locais (falsos argumentos), eles não têm que ser declarados debaixo de VAR. Os falsos argumentos não são visíveis fora da função. Os nomes deles/delas são permitidos ser idêntico a esses fora da função. Duplicação de nomes de constantes residentes e variáveis de MODELS é proibida.

Outra forma de se utilizar a diretiva FUNCTION é:



```
FUNCTION somename(argname1, argname2, ... ) := expression
```

```
somename(value1, value2, ... )
```

```
somename(expr, array_expr, ... )
```

```
FUNCTION c_ampl(real, imag) := sqrt(real*real + imag*imag)
...
ym := c_ampl(xr, xi)
```

```
VAR cx[1..2] -- a complex variable as real and imaginary
FUNCTION c_ampl(real, imag) := sqrt(real*real + imag*imag)
...
ym := c_ampl(cx[1..2])
```

## 5.9 FUNÇÕES POINTLIST

O usuário pode definir uma função de  $y=f(x)$  entrando em uma tabulação de  $y$  e  $x$  em ordem crescente de  $x$ . A sintaxe é:

```
FUNCTION name POINTLIST
      (x1,  y1)
      (x2,  y2)
      . . . . .
      (xn, yn)
```

O pointlist funciona da seguinte maneira:

- ⇒  $z := \text{nome}(xi, \text{pol})$ ;
- ⇒ 'pol' provê o modo de interpolação entre os valores de  $x$ .
- ⇒  $\text{pol}=0, 1, 2$  proverão o valor de  $y$  como interpolação descontínua, linear ou quadrática. Omitindo  $\text{pol}$  dará interpolação linear como falta.

O exemplo seguinte é a relação normalizada entre o disparado de ângulo e atual por um thyristor, dando só cinco pontos na curva:

```
FUNCTION alpha POINTLIST
-- firing anglgle current
      (0.0000, 0.000)
      (0.0576, 0.333)
      (0.2414, 0.555)
      (0.5718, 0.777)
      (1.0000, 1.000)
```

Então adquirir o valor  $\alpha$  de um delay de ângulo de demora

```
ix := alpha (dela)
```

Outra maneira de se utilizar a diretiva POINT LIST é:

```
FUNCTION temp_dep POINTLIST (-273, 1.e-6), (-60, 1.e-4), (0, 1.e2), etc...
```

```
VAR a, k
FUNCTION curve2(x) ... -- a statement function or a point list for k=2
FUNCTION curve4(x) ... -- a statement function or a point list for k=4
FUNCTION curve6(x) ... -- a statement function or a point list for k=6
FUNCTION f POINTLIST (2, curve1(a))
                    (4, curve2(a))
                    (6, curve3(a))
...
a:= some value      -- used inside the function f(k)
k:= 2.5             -- any value between 2 and 6
y:=f(k)
```

# Capítulo 6

## 6 EXPRESSÕES DE SIMULAÇÃO E FUNÇÕES

### 6.1 FUNÇÕES

#### 6.1.1 EXPRESSÃO DERIVADA COM POLINÔMIO LINEAR

Será usado por avançado MODELA os usuários

**DIFFEQ Differential Equation**

Outra utilização da diretiva DIFFEQ é:

```
DIFFEQ(polynomial)|y := x
```

```
expr|D0 ± expr|D1 ± expr|D2 ± ...
```

#### 6.1.2 Função DERIVADA

A sintaxe da derivada de primeira ordem é:

```
y := DERIV ( x ) { dmax:a dmin:b }
```

Onde 'x' é a variável e 'dmax' e 'dmin' são limites dinâmicos

A sintaxe da derivada de segunda ordem é:

```
y := DERIV2 ( x ) { max:a min:b }
```

Example:

```
MODEL diff
  CONST omega { val: 314 }
  VAR x, dx, y
  HISTORY x {dflt: 0}
  --
  EXEC
    x :=sin ( omega * t )
    dx := deriv (x) {max: 0.6 *cos(omega*t)}
  ENDEXEC
ENDMODEL
```

### 6.1.3 FUNÇÃO INTEGRAL

A sintaxe da expressão integral é:

```
y := INTEGRAL ( x ) { dmax:a dmin:b }
```

Onde ‘x’ é a variável e ‘dmax’ e ‘dmin’ são limites dinâmicos

Podem ser usados só limites dinâmicos com integrais da MODELS. Isto significa que o integral não é cortado, mas limitado pelo processo de integração.

O usuário tem que declarar x e integral (x) debaixo de HISTORY e tais declarações podem não ser triviais.

Exemplos:

Deixe o integral de x = a seja calculado

```
MODEL test
  VAR a, x, y
  HISTORY x {dflt :0}
  integral (x) { dflt
:0 }
  --
  EXEC
    x := a*t
    y := integral (x) { dmax: 50 dmin: 10 }
  ENDEXEC
ENDMODEL
```

Pode ser reajustado o valor de integral durante a simulação pelo integral reajuste da expressão da integral:

```
integral (x) := expression
```

Exemplo:

```

MODEL test
  CONST omega { val: 2*pi*50 }
  VAR y, y1, x
  HISTORY x { dflt: 0 }

  integral(x) { dflt : -1/omega }
  --
  EXEC
    x := sin ( omega*t )
    y1 := integral ( x )
    y := omega * y1
  IF t > 0.05 THEN integral ( x ) := 0 ENDIF
  ENDEXEC
ENDMOEL

```

Outra forma de se utilizar a diretiva INTEGRAL é:

```
y := integral(x)
```

```
integral(x) := new_value
```

$$\Delta integral = \frac{x1 + x0}{2} \cdot \Delta t$$

```
y:=integral(x) {dmin:expression, dmax:expression}  -- dynamic limits
```

```
y := integral(x)
```

```
y := y {min:expression, max:expression}  -- static limits
```

```
MODEL ilim  -- illustrates dynamic limit on integral in MODELS
```

```
VAR x, y  -- step signal =1 from t=0 to t<=1, =-1 for t>1
```

```
  dlim  -- integral(x), dynamic limit=0.7
```

```
  nolim -- integral(y), no limit
```

```
  slim  -- static limit=0.7 applied to 'nolim'
```

```
HISTORY x {dflt: 0}
```

```
  y {dflt: 0}
```

```
  integral(x) {dflt: 0}
```

```
  integral(y) {dflt: 0}
```

```
EXEC
```

```
  x:= 1
```

```
  IF t>1 THEN x:= -1 ENDIF
```

```
  y:= x
```

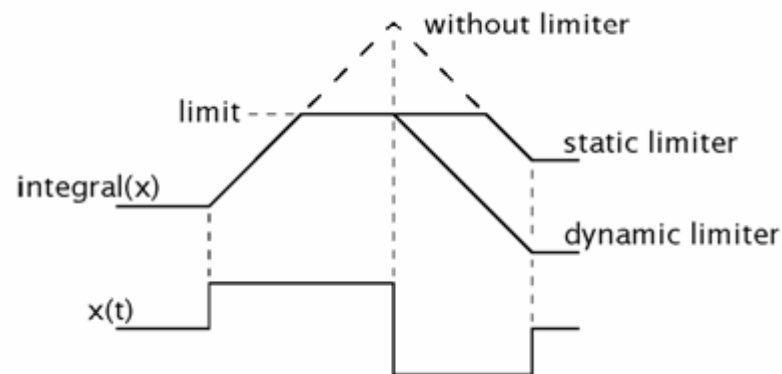
```
  dlim :=integral(x) {dmax: 0.7}
```

```
  nolim :=integral(y)
```

```
  slim :=nolim {max:0.7}
```

```
ENDEXEC
```

```
ENDMODEL
```



#### 6.1.4 EXPRESSÃO SUM (SOMA)

A expressão de SUM avalia o valor numérico de um polinômio.

$$k = a + bx + cx_2 + \dots + zx_n$$

A sintaxe da expressão é:

```
k := SUM ( a + b|x1 + c|x2 + ... ) {max:5 min: -1}
```

Exemplo:

Avalie o k polinomial =  $um + bx + cx^2$

```
MODEL test
  DATA a {dflt : 3.5}
  CONST b {val: 2.0}
        c {val :4.4}
  VAR k, x, x1
  --
  EXEC
    x := t
    x1 := t**2
    k := sum ( a + b|x + c|x1 )
  ENDEXEC
ENDMODEL
```

Note que o que é terminado pela expressão de SUM pode ser feito sem isto escrever uma declaração de expressão numérica simplesmente. A razão para a expressão de SUM é que pode ser usado dentro de um grupo COMBINE para a solução simultânea de declarações.

## 6.2 EXPRESSÕES DE SIMULAÇÃO

### 6.2.1 EXPRESSÕES DE SIMULAÇÃO RESIDENTES

MODELS provêem várias funções de simulação. Frequentemente as mais usadas são:

`delay ( x, d, pol )`

O valor da variável um de cada vez t-d. 'pol' é o indicador de interpolação opcional. Se 'pol' é omitida a interpolação é linear. pol = 0,1,2 posto para interpolação discreta, linear e quadrática respectivamente.

**Advertindo:** A função de delay provê a história da variável 'x', a um intervalo discreto atrás do tempo atual, e não necessariamente ao 'td' de intervalo preciso. Lá está um erro que depende do tamanho do passo de tempo usado no modelo. Alguns modelos podem ser extremamente sensíveis a isto, enquanto resultando em um erro grande na produção do modelo.

### 6.2.2 FUNÇÃO PREVVAL

`prevval ( x )`

O valor de x no momento de simulação prévio

### 6.2.3 FUNÇÃO HISTDEF

`histdef ( x )`

Declaração de história usada debaixo do processo de INIT

#### 6.2.4 CLAPLACE e LAPLACE função de transferência

MODELS provêem duas versões da função de Laplace (transferência). O geralmente usado é uma função de transferência com coeficientes constantes. O segundo é a função de Laplace onde os coeficientes são variáveis. Com coeficientes variáveis a função que LAPLACE deve ser usado.

Com coeficientes constantes deveria ser usado CLAPLACE porque é mais rápido que LAPLACE. Com CLAPLACE são avaliados só uma vez os coeficientes da função de transferência ao começo de simulação, considerando que com LAPLACE eles são avaliados no começo de cada passo de tempo.

A sintaxe do LAPLACE ou função de CLAPLACE é determinada para CLAPLACE só como isto seria usada principalmente pela primeira vez de MODELS de tempo:

```
CLAPLACE (y/x) := ( k0|s0 + k1|s1 + k2|s2 ....
etc) /
                ( l0|s0 + l1|s1 + l2|s2 etc)
```

Onde:

- ⇒ x = variável de entrada;
- ⇒ y = saída;
- ⇒ k0 de produção...l0.. É s0 de coeficientes constantes, s1, s2, s3 simbolizam o operador de s para a energia.

A função de Laplace pode ser usada com limites estáticos e dinâmicos. Com limites estáticos é cortada à produção, com limites dinâmicos a produção está limitada pelo processo de avaliar a função.

```
For static limits: CLAPLACE (y/x) {max: u min: l } :=
For dynamic limits: CLAPLACE (y/x) {dmax: u dmin: l } :=
```



Devem ser declarados y e x em uma declaração de história. Na maioria dos casos será suficiente para entrar na história com dflt:0

Exemplo 1/M espetáculos uma função de Laplace simples. A versão de LAPLACE estava lá usada, mas baseado nas anteriores discussões, o CLAPLACE seria preferido com os coeficientes da função constante ao longo da simulação. Resultados serão os mesmos, mas solução é mais rápida com CLAPLACE.

O exemplo seguinte ilustra parte de um sistema de controle. O sinal de contribuição para o primeiro bloco é a soma de um sinal de referência 'sref', um sinal 'sin', e um sinal de avaliação derivado 'sfeed', levado do sinal de produção 'sed.' O sinal de produção 'sout' é cortado + /- 0.05.

A função de transferência para o bloco dianteiro é:

$$kgain/(1+ tes)$$

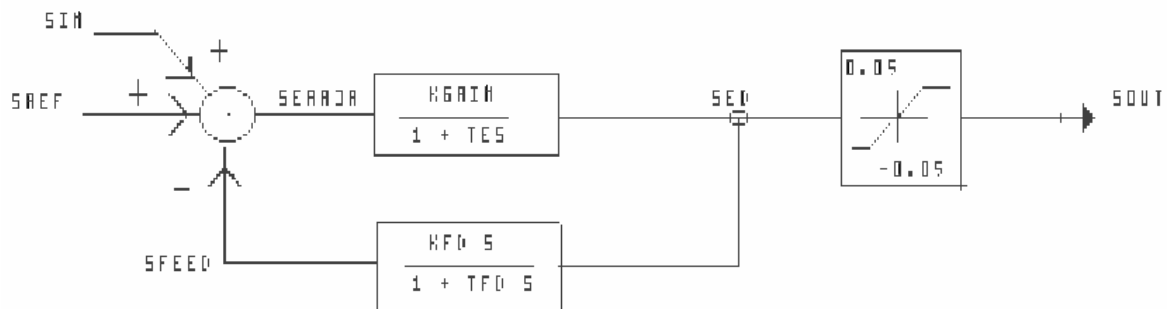
A função de transferência para a avaliação é:

$$kfds/(1 + tfds)$$

```

MODEL
--
  DATA
    kagin {dflt: 50}
    kfd   {dflt:0.02}
  CONST
    te    {val: 0.5}
    tfd   {val: 1.2}
  VAR error, sin, sref, sfeed, sed, sout
  HISTORY
    error {dflt:0}
    sfeed {dflt:0}
    sed   {dflt:0}
  INIT
    sin := 1.0
    sout := 0
  ENDINIT
--
EXEC
error := sum( 1|sref + 1|sin - 1|sfeed)
CLaplace ( sed / error) :=
  ( kgain|s0 / ( 1|s0 + te|s1 )
CLaplace ( sfeed / sed ) :=
  ( kfd|s1 ) / ( 1|s0 + tfd|s1 )
sout := sed {max: 0.05 min: -0.05 }
ENDEXEC
ENDMODEL

```



Note que para a 'correta' avaliação de uma avaliação, a opção COMBINE tem que ser usada como explicada no próximo Capítulo.

Outra forma de se usar as funções de transferência é:

$$\frac{Y(S)}{X(S)} = \frac{b_0 + b_1s + b_2s^2}{a_0 + a_1s + a_2s^2}$$

```
LAPLACE(y/x) := (numerator)/(denominator)
```

```
(expr|s0 ± expr|s1 ± expr|s2 ± ... )
```

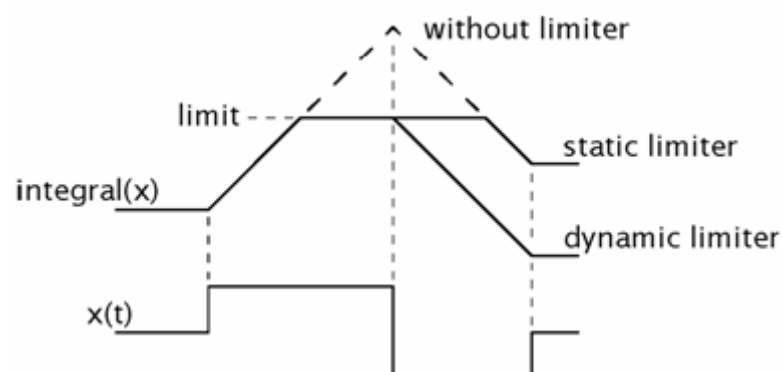
$$\frac{Y(z)}{X(z)} = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + \dots}{a_0 + a_1z^{-1} + a_2z^{-2} + \dots}$$

```
ZFUN(y/x) := (numerator)/(denominator)
```

```
(expr|z0 ± expr|z-1 ± expr|z-2 ± ... )
```

```
LAPLACE(y/x) {dmin: expr} := (...)/(...)
```

```
LAPLACE(y/x) := (...)/(...)  
z := y {min: expr}
```



# Capítulo 7

## 7 ALGORITMO DE CONTROLE

MODELS uso vários algoritmos para controle de declarações semelhante aos usados em outras linguagens de programação, como o Fortran e Basic.

Os controles de algoritmo seguintes são possíveis:

```
- IF .... ENDF
- FOR .... ENDFOR
- DO ---- ENDDO
- WHILE -- ENDWHILE
- REDO
- SEQUENCE
- COMBINE
```

### 7.1.1 IF – ENDF (SE)

A estrutura desse controle é:

```
IF logical expr. THEN
    statement list
ELSIF logical expr. THEN
    statement list
ELSE
    statement list
ENDIF
```

Por exemplo:

```
IF case = 1 THEN
    a:= 33
    cb:=sqrt (d1)
ELSIF case >=2 and case < 5 THEN
    a := 24
    cb := sqrt (d2)
ELSE
    a := 20
    cb := d3
ENDIF
```

Outra forma de se utilizar o laço IF é:

```
IF condition_is_true THEN
  statement list
ENDIF
```

```
IF condition_is_true THEN
  statement list
ELSIF another_condition_is_true THEN
  another statement list
ELSIF ...
...
ENDIF
```

```
IF condition_is_true THEN
  statement list
ELSIF another_condition_is_true THEN
  another statement list
...
ELSE
  another statement list
ENDIF
```

```
IF condition_is_true THEN
  statement 1
  statement 2
  IF condition THEN
    another statement list
  ...
ENDIF
...
ENDIF
```

```
MODEL if_example  -- illustrates the IF structure in MODELS
VAR k
INIT
  k:=0
ENDINIT
EXEC
  IF k=0 THEN
    k:=1
  ELSIF k=1 THEN
    write("In first 'elsif' with k=1")
    k:=2
  ELSIF k=2 THEN
    write("In second 'elsif' with k=2")
    k:=3
  ELSE
    write("In 'else' with k=3")
  ENDIF
ENDEXEC
ENDMODEL
```

### 7.1.2 DO – REDO (FAÇA – REFAÇA)

A declaração DO ensina MODELS a executar um conjunto de instruções entre o DO e a terminação ENDDO. Se qualquer laço é requerido dentro de DO, pode ser feito por uma declaração ou diretiva REDO. REDO é uma expressão lógica satisfeita, reiniciará a execução de ordens depois DO e adiante REDO.

```

EXEC
DO
    write ( 'start/restart DO ' )
    write ( ' k = ',k )
    k := k - 1
    IF k > 0 REDO ENDIF
    x := 10
    write ( ' x = ',x )
ENDDO
ENDEXEC

```

Uma outra forma de se utilizar o laço DO é:

```

DO
    statement list
ENDDO

```

```

redo_flag := TRUE
WHILE redo_flag = TRUE DO
    statement list
ENDWHILE

```

### 7.1.3 WHILE – DO – ENDWHILE (ENQUANTO – FAÇA)

A declaração WHILE ensina MODELS para executar a instrução depois WHILE e antes de ENDWHILE sujeitassem a uma expressão lógica que está satisfeito.

Exemplo:

```

EXEC
VAR    k
WHILE k > 0 DO
    k := k-1
    write ( ' k = ',k )
ENDWHILE
ENDEXEC

```

Outro exemplo pode ser para o uso de WHILE:

```

WHILE condition_is_true DO
    statement list
ENDWHILE

```

```

MODEL while_example -- illustrates the use of WHILE
VAR k
EXEC
  IF t=timestep THEN -- execute only at the first time step
    write("*** Begin results of model 'while_example' ***")
    k:=5
    WHILE k>0 DO
      write('k=', k, ' in loop ')
      k:=k-1
    ENDWHILE
    write('k=', k, ' after loop ')
    write("*** End results of model 'while_example' ***")
  ENDIF
ENDEXEC
ENDMODEL

```

#### 7.1.4 FOR – DO – ENDFOR (PARA – FAÇA)

FOR permite a repetição de um conjunto de ordens controlada um aumentando monotonamente ou índice decrescente, em passos definidos, e onde a gama do índice está definida por um TO expressão de k. Isto é semelhante para o FOR - NEXT volta em BASIC.

A sintaxe é:

```

FOR i := k1 TO k2 BY d
  DO
    -- list of instructions
  ENDFOR

```

BY é opcional com uma valor padrão de 1 (um).

Como em BASIC, FOR podem ser aninhadas voltas, a declaração de ENDFOR tem que só ser dada uma vez.

```

FOR i := 1 TO 5
  FOR k := 3 to 1 BY -1
    DO
      instructions
    ENDFOR
  ENDFOR

```

Outra forma de se utilizar o laço FOR é:

```

FOR i:= list of values
FOR j:= list of values
FOR ...
DO
  statement list
ENDFOR

```

```

FOR n:= 1 to 4
FOR m:= n to 4
DO
  write( m ) -- 1,2,3,4, 2,3,4, 3,4, 4
ENDFOR

```

```

FOR z:= 0 TO 5,      -- 0,1,2,3,4,5
      6,             -- 6
      7 TO 9 BY 1 -- 7,8,9

```

```

MODEL for_example  -- illustrates the use of FOR in MODELS
VAR k
EXEC
  IF t=timestep THEN -- execute only at the first time step
    write("*** Begin results of model 'for_example' ***")

    FOR z:= 0 TO 5,      -- 0,1,2,3,4,5
          6,             -- 6
          7 TO 9 BY 1 -- 7,8,9
    FOR y:= 0 TO 8 BY 2 -- 0,2,4,6,8
    DO
      k:=z*10 +y          -- 0,2,4,6,8,10,12,...,94,96,98
      write(k)
    ENDFOR

    FOR z:= 9 to 0 by -1 -- 9,8,7,6,5,4,3,2,1,0
    FOR y:= 5, 0          -- 5,0
    DO
      write(z*10 +y)      -- 95,90,85,80,75,...,15,10,5,0
    ENDFOR

    write("*** End results of model 'for_example' ***")
  ENDIF
ENDEXEC
ENDMODEL

```

```

FOR angle:= 0 TO 2*pi BY pi/4
DO
  write('tan(', deg(angle), ') = ', tan(angle))
ENDFOR

```

Exemplo 3 ilustra a geração de três voltagens senoidais com 120 graus realize em fases troca e a geração de incendiar 500 microsec. pulsos sincronizaram com a zero transição das voltagens.

## Example 3

```

-----
BEGIN NEW DATA CASE
$CLOSE, UNIT=4 STATUS=DELETE
$OPEN, UNIT=4 FILE=model 3.pl4 FORM=UNFORMATTED STATUS=UNKNOWN
RECL=8000! PRINTED NUMBER WIDTH, 13, 2,
C deltat tmax xopt copt epsiln tolmat tstart
.0001 .080 50. 0.0
C print points connec s-s minmax vary again plot
1 1 0 0 0 0 0 1
C
=====
=MODELS
MODEL test
CONST omega { val: 2*pi*50 }
C
VAR
volt [ 1..3], fire[1..3]
C
C
HISTORY
fire[1..3] {dflt:0}
volt[1..3] {dflt:0}
INIT
-- no initialization req'd, initialized in HISTORY
ENDINIT
C
DELAY CELLS DFLT: 10
C
EXEC
-- generate three phase voltages
FOR i:= 1 TO 3 DO
volt[i] := sin ( omega*t + (i-1)*2*pi/3 )
ENDFOR
-- generate 500 us firing pulses at 0 transition of voltages
-- note use of logical expression
-- fire = 1 if both volt and - volt(delayed) are > zero
FOR i := 1 TO 3 DO
fire[i] := volt[i] and -delay (volt[i],0.0005)
ENDFOR
ENDEXEC
ENDMODEL
USE test AS test
timestep min:0
ENDUSE
C -----
RECORD test.volt[1] AS va
test.volt[2] AS vb
test.volt[3] AS vc
test.fire[1] AS fira
test.fire[2] AS firb
test.fire[3] AS firb
ENDMODELS

C BRANCH CARDS
SRCE 10.
BLANK CARD ENDING BRANCHES
BLANK CARD ENDING SWITCHES
14SRCE 1.0 50.
BLANK CARD ENDING SOURCES
SRCE
BLANK CARD ENDING NODE VOLTAGE REQUEST
BLANK CARD ENDING PLOT
BEGIN NEW DATA CASE
BLANK

```



## 7.2 COMBINE (COMBINAR)

Como em TACS, MODELS executarão a instrução na seção de EXEC na sequência de entrada. Em muitas aplicações de sistema de controle é desejável para maior precisão, executar algumas das declarações, ou controle de blocos, simultaneamente. Um exemplo típico seria uma volta de controle de avaliação. COMBINE - ENDCOMBINE provê os meios dentro de MODELS fazer isto. A sintaxe é:

```
COMBINE AS groupx
statements
ENDCOMBINE
```

A significação de 'groupx' é permitir mais que um COMBINE seção em um MODELO. O exemplo simples é:

```
EXEC
COMBINE AS group1
  error := sum( 1|sref + 1|sin  - 1|sfeed)
  --
  CLaplace ( sed / error) :=
    ( kgain|s0 / ( 1|s0 + te|s1 )
  --
  CLaplace ( sfeed / sed ) :=
    ( kfd|s1 ) / ( 1|s0 + tfd|s1 )
  --
  ENDCOMBINE
sout := sed {max: 0.05 min: -0.05 }
ENDEXEC
```

Outra forma de se usar a diretiva COMBINE é:

```
COMBINE AS identifier
statement list
ENDCOMBINE
```

```
COMBINE AS feedback_controller
equations of the feedback controller
ENDCOMBINE
```

```
FOR i:=1 TO n DO
  COMBINE AS probe[i]
    equations of the probe circuit, indexed with "i"
  ENDCOMBINE
ENDFOR
```

```

MODEL comb1
VAR a, b
EXEC
  COMBINE AS first_group
    b := sum( 6| - 1|a )
    a := sum( t|b )
  ENDCOMBINE
  write('t=',t,', a=',a,', b=',b,', a+b=',a+b)
ENDEXEC
ENDMODEL

```

```

MODEL comb1
VAR a, b
EXEC
  COMBINE AS first_group
    b := sum( 6| - 1|a )
    a := sum( t|b ) { max: 4 }
  ENDCOMBINE
  write('t=',t,', a=',a,', b=',b,', a+b=',a+b)
ENDEXEC
ENDMODEL

```

```

MODEL comb2 -- illustrates linear COMBINE groups in MODELS
VAR y1, y2, y3, x, h
HISTORY y1 {dflt: sin(t)}
        y2 {dflt: cos(t)}
        y3 {dflt: -sin(t)}
        integral(y2) {dflt: sin(t)}
        integral(y3) {dflt: cos(t)}
        x {dflt: cos(t)}
INIT
  h:=timestep/2
ENDINIT
EXEC
  write(' ')
  write('Results at t=', t, ' :')

  COMBINE AS first_group
    y1:= integral(y2)

```

```

    y2:= integral(y3)
    y3:= sum(-1|y1)
  ENDCOMBINE
  write(sin(t),':',y1)

  COMBINE AS second_group
    y2:= deriv(y1)
    y3:= deriv(y2)
    y1:= sum(-1|y3)
  ENDCOMBINE
  write(sin(t),':',y1)

  COMBINE AS third_group
    laplace(y1/y2) := 1|s0 / 1|s1
    laplace(y2/y3) := 1|s0 / 1|s1
    y3:= sum(-1|y1)
  ENDCOMBINE
  write(sin(t),':',y1)

  COMBINE AS fourth_group
    zfun(y2/y1) := (1|z0 -1|z-1)/(h|z0 +h|z-1) -- (s/1)
    laplace(y3/y2) := 1|s1 / 1|s0
    y1:= sum(-1|y3)
  ENDCOMBINE
  write(sin(t),':',y1)

ENDEXEC
ENDMODEL

```

## 7.3 TIMESTEP

MODELS permite ao usuário ter um passo de tempo de execução em MODELS diferente do usado no EMTP. A diretiva TIMESTEP pode ser entrada ou no grupo MODEL, ou no grupo de USE. A sintaxe é:

```
TIMESTEP max:xx min:yy
e.g. TIMESTEP max: 0.001
      TIMESTEP min: 0.002
```

Isto pode acelerar a execução de um arquivo de ATP onde um passo de tempo muito pequeno é usado para acomodar exigências da rede elétrica substancialmente, mas a solução em MODELS não requer este passo de tempo pequeno.

Uma forma de se usar a diretiva TIMESTEP é:

```
TIMESTEP MAX: 0.1 * min(tau1, tau2, tau3)
TIMESTEP MIN: 0.01 * min(tau1, tau2, tau3)
```

## 7.4 RECORD – WRITE (REGISTRO – ESCREVER)

### 7.4.1 RECORD

A declaração de RECORD é colocada ao término dos MODELS sendo o último grupo em MODELS, enquanto seguindo os grupos de USE. A palavra de pedido de RECORD é seguida pelo pedido de variáveis para ser produzida na forma de:

```
modelname.varnamex AS outnam
```

Onde 'modelname' é o nome do MODEL que contém a variável nomeada 'varnamex' e 'outnam' é o nome determinado para esta variável para a gravação da produção em disco. A razão para esta duplicação aparente é que MODELS permitem nomes de qualquer comprimento, considerando que ATP permite nomes muito tempo só 6 (seis) caráter. Além disto, a produção pedida de

variáveis é tratada exatamente por ATP o mesmo modo como eles estão em EMTP ou no 33 código TACS produziu pedido. O uso de RECORD permite que seja plotado o sinal escolhido.

### 7.4.2 WRITE

Para a prova e depurando modelos, o usuário pode pôr a declaração 'write' no EXEC separado do modelo, monitorar a operação do passo a passo e execução do modelo. A informação exigida será escrita no arquivo .lis durante a execução do modelo em cada passo de tempo a menos que caso contrário dirigisse por um se declaração. A sintaxe desta declaração é:

```
write ( ' any text    ', x1, ' ', x2, ' ' etc )
```

Onde 'qualquer texto' é uma descrição das variáveis ser registrado e 'x1', 'x2' etc. são as variáveis a ser monitoradas. Note aquele 'x1', 'x2' etc. não só possa ser variáveis, mas também expressões como delay (x, t).

## 7.5 MULTIPLE MODELS

Pode haver mais de um MODEL em um arquivo de MODELS. Por exemplo:

```
MODELS
  MODEL John
  .....
  ENDMODEL
  USE John AS John
  .....
  ENDUSE
  MODEL Mary
  .....
  ENDMODEL
  USE Mary AS Mary
  .....
  ENDUSE
ENDMODELS
```

Assuma que o usuário desenvolveu dois modelos, um para o cálculo de única fase de energia ativa em um sistema de CA, e para energia reativa. Estes modelos são nomeados 'power' e 'reactive' respectivamente.

O modelo 'power' é o seguinte:

```
--          POWER
MODEL power
--
  VAR pavg, vdel, idel
  INPUT vv {dflt: 0}, ii {dflt: 0}
        freq {dflt:0}, tper{dflt:0}, tmult{dflt:0}
  HISTORY vv {dflt: 0}
          ii {dflt: 0}
  INIT pav := 0
  ENDINIT
    DELAY CELLS (vv): tper/timestep
    CELLS (ii): tper/timestep
  EXEC
  -- calculate active power
    vdel := delay (vv, tper )
    idel := delay (ii, tper )
    pavg := pavg +( vv * ii - vdel * idel)/tmult
  ENDEXEC
ENDMODEL
```

O modelo 'reactive' é o seguinte:

```
--          REACTIVE
MODEL reactive
  VAR qavg, tperq, tmultq, vdel, idel
  INPUT vv {dflt: 0}, ii {dflt :0}
        freq, tper, tmult
  HISTORY vv {dflt: 0}, ii {dflt:0}
  INIT
    tperq := tper/2, tmultq := tmult/2, qavg := 0
  ENDINIT
  DELAY CELLS (vv): tper/timestep
  CELLS (ii): tper/timestep
  -- calculate reactive
  EXEC
  vdel:= delay (vv, tper )
  idel := delay (ii, tperq)
  qavg:=qavg+(ii * delay (vv,tperq)-vdel * idel)/tmultq
  ENDEXEC
ENDMODEL
```

Em Exemplo 4, serão calculadas energia ativa e energia reativa por um ramo na rede de EMTF, passando a voltagem e corrente ao MODEL ' model\_3 ', e integrando em model\_3 os modelos 'power' e 'reactive' por declarações \$ INCLUDE.

Note que a inclusão dos dois grupos MODEL nos arquivos MODELS por \$ INCLUDE é há pouco uma característica de edição, economiza copia dos arquivos completos em MODELS. USE declarações têm que ser acrescentadas a cada MODEL. Eles têm que especificar as entradas ao grupo MODELO.

#### Example 4

```
-----
BEGIN NEW DATA CASE
$CLOSE, UNIT=4 STATUS=DELETE
$OPEN, UNIT=4 FILE=mod 4.pl4 FORM=UNFORMATTED STATUS=UNKNOWN RECL=8000 !
PRINTED NUMBER WIDTH, 13, 2,
C deltat   tmax   xopt   copt   epsiln   tolmat   tstart
  .0001     .100    50.    0.0
C print   points   connec   s-s   minmax   vary   again   plot
      1         1         0       0       0       0       0       1
MODELS
  INPUT volt {v(BUSA)}
        cur {i(BUSB)}
  OUTPUT freq, tper, tmult, vv, ii
MODEL model_3
DATA freq {dflt: 0}
--
VAR
  tmult, tper
  INPUT vv {dflt: 0}
        ii {dflt: 0}
  OUTPUT freq, tper, tmult
        vv, ii
  HISTORY vv {dflt: 0} ii {dflt: 0}
  INIT
    tper := 1/freq
    tmult := tper/timestep
  ENDINIT
--
EXEC
ENDEXEC
ENDMODEL
--
USE model_3 AS test
DATA freq := 50
INPUT
  vv:= volt ii := cur
OUTPUT freq:= freq, tpe := tper, tmult : tmult
  vv := vv, ii := ii
ENDUSE
-- REACTIVE MODEL
$INCLUDE reactive.dat
```

```

USE reactive AS reactive
  INPUT
    vv := vv, ii := ii, freq := freq, tper := tper, tmult := tmult
  ENDUSE
  --          POWER
  $INCLUDE power.dat
  USE power AS power
  INPUT
    vv:= vv, ii:= ii
    tper := tper, tmult := tmult, freq := freq
  ENDUSE

C -----
--
RECORD test.vv      AS vv
      test.ii      AS ii
      reactive.qavg AS qavg
      power.pavg   AS pavg
ENDMODELS
--
C BRANCH CARDS
  GENA BUSA          .010
  BUSB              99.01  9.9
BLANK CARD ENDING BRANCHES
  BUSA BUSB
BLANK CARD ENDING SWITCHES
14GENA 141000.0 50. 0.
BLANK CARD ENDING SOURCES
  BUSB
BLANK CARD ENDING NODE VOLTAGE REQUEST
BLANK CARD ENDING PLOT
BEGIN NEW DATA CASE
BLANK

```

MEASURING

-1

# Capítulo 8

## 8 VERSÃO MODELS DE FONTES E DISPOSITIVOS DA TACS

### 8.1 INTRODUÇÃO

Este capítulo visa ajudar os usuários da MODELS que TACS previamente usado e é acostumado para usar as fontes de TACS e dispositivos em arquivos de dados em desenvolvimento, a versão de MODELS equivalente destas fontes e dispositivos esteve preparada. Isto também proverá ajuda a usuários que não têm TACS usado previamente para desenvolver MODELS arquivos.

São apresentadas as fontes e dispositivos na forma de Modelo individual que o usuário pode integrar em um arquivo de MODELS, ou por um MODEL separado, ou copiando a codificação do MODEL em um arquivo de MODELS que contém só um único MODEL. Dependendo do método de uso, a declaração de DATA no MODEL, ou diretiva de DATA na seção de USE tem que ser completado pelo usuário.

O usuário é aconselhado para estudar o Livro de leitura antes de tentar usar estes modelos.

### 8.2 MODELOS EQUIVALENTS DE TACS E FONTES DE EMTP

List of Sources:

TACS Code 11	-- Level Signal and Single Pulse
TACS Code 14	-- Cosine
TACS Code 23	-- Pulse Train
TACS Code 24	-- Ramp Train (Saw-tooth)
EMTP Type 13	-- Two-slope Ramp
EMTP Type 15	-- Double Exponential



### 8.2.1 SOURCE 11 - 'LAVEL SIGNAL AND SINGLE PULSE'

```

*****
TACS Source Code 11 - Level Signal and Single Pulse
=====
MODEL single_pulse      --
--
    DATA  tstart {dflt:0}      -- pulse start time sec.
           tstop  {dflt:1000}   -- cut-off time sec.
           ampl  {dflt :1}      -- amplitude
           width {dflt:0}       -- pulse width
-- for unit pulse enter a large pulse width, say 100 sec.
    VAR spulse
EXEC
    IF t > tstart and tstop > t THEN
        spulse := ampl * (t > tstart and t < tstart + width)
    ENDIF
ENDEXEC
ENDMODEL
-

```

### 8.2.2 SOURCE 14 - 'COSINE'

```

*****
TACS Source Code No. 14      Cosine Wave
=====
MODEL cosine -- cosine source
--
    DATA  tstart {dflt:0}      -- start time
           tstop  {dflt:1000}   -- cut-off time
           ampl  {dflt :1.0}    -- amplitude
           freq  {dflt:60}      -- frequency
           angle {dflt:0}       -- angle deg. at t=tstart
    VAR cosine, omega
        INIT  omega := 2*pi*freq ENDINIT
EXEC
    cosine := ampl * ( cos (omega*(t-tstart) + angle*pi/180) )
               * AND ( (t - tstart), (tstop-t) )
ENDEXEC
ENDMODEL
=====

```

### 8.2.3 SOURCE 23 - 'PULSE TRAIN'

```

*****
TACS Source Code No. 23      Pulse Train
=====
MODEL pulse_train      --
--
    DATA  tstart {dflt:0}      -- starting time
           tstop  {dflt :1000}  -- cut-off time
           ampl   {dflt:1}      -- amplitude
           width  {dflt:0}      -- pulse width
           period {dflt:0}      -- pulse period
    VAR pulse_tr
EXEC
    pulse_tr := ampl * ((t - tstart) mod period < width)
                * AND ( (t - tstart), (tstop-t) )
ENDEXEC
ENDMODEL
=====

```

### 8.2.4 SOURCE 24 - 'RAMP TRAIN'

```

*****
TACS Source Code No. 24      Ramp Train
=====
MODEL saw_tooth
--
    DATA  tstart {dflt:0}      -- starting time
           tstop  {dflt :1000}  -- cut-off time
           ampl   {dflt:1}      -- amplitude
           width  {dflt:0}      -- pulse width
           period {dflt:0}      -- pulse period
    VAR sawtooth, x
    INIT x := ampl/period ENDINIT
EXEC
    sawtooth := x * ((t - tstart) mod period )
                * AND ( (t - tstart), (tstop-t) )
ENDEXEC
ENDMODEL
=====

```

### 8.2.5 SOURCE 13 - 'TWO SLOPE RAMP'

```

*****
EMTP Type No. 13      Two Slope Ramp
=====
MODEL double_ramp  -- linear rise and fall
--
  DATA tstart {dflt:0}      -- start of signal
        ampl {dflt:1.0}     -- amplitude at time tstart + t0
        a1 {dflt:0}         -- amplitude at time tstart + t1
        t0 {dflt:0}         -- rise time
        t1 {dflt:0}         -- time to amplitude a1
--
  VAR rampout, rup, rdown, tsl1, tsl2
  INIT rampout:=0, tsl1:=ampl/t0,  tsl2:= (ampl-a1)/(t1-t0)
  ENDINIT
EXEC
  rup := (timestep * tsl1)
        * AND ( (t-tstart), (tstart + t0 -t) )
  rdown := -(timestep * tsl2)
        * AND ( t-tstart - t0 - timestep, rampout )
  rampout := rampout + rup + rdown
ENDEXEC
ENDMODEL
=====

```

### 8.2.6 SOURCE 15 - 'DOUBLE EXPONENCIAL SURGE'

```

*****
BMTP Type No. 15      Double Exponential Surge
=====
MODEL d_surge  -- double exponential surge
--
  DATA  tstart {dflt:0}      -- start of signal
        tstop {dflt:1000}    -- cut-off signal
        ampl {dflt:1}        -- amplitude
        a {dflt:-10}         -- 1/a time const. first exponential
        b {dflt:-30}         -- 1/b time const. second
exponential
  VAR surgeout
  OUTPUT surgeout  -- delete if not used as separate MODEL
EXEC
  surgeout:= ampl * (exp(a*(t-tstart)) - exp(b*(t-tstart)))
        * AND((t-tstart), (tstop-t))
ENDEXEC
ENDMODEL
=====

```

## 8.3 MODELOS EQUIVALENTS DE DISPOSITIVOS DE TACS

List of Devices:

TACS Device 50	--	Frequency Meter
TACS Device 51/52	--	Relay Switch
TACS Device 53/54	--	Transport and Pulse Delay
TACS Device 55	--	Digitizer
TACS Device 56	--	Point by Point Nonlinearity
TACS Device 57	--	Time Sequenced Switch
TACS Device 58	--	ControlledIntegrator
TACS Device 59	--	Single Derivative
TACS Device 60	--	Input IF Component
TACS Device 61	--	Signal Selector
TACS Device 62	--	Sample and Track
TACS Device 63	--	Inst. Maximum/minimum
TACS Device 64	--	Maximum/minimum Tracking
TACS Device 65	--	Accumulator - Counter
TACS Device 66	--	R.M.S. Value

### 8.3.1 DEVICE 50 - 'FREQUENCY METER'

```

*****
TACS Device 50          Frequency Meter
=====
MODEL freq_meter
--
  DATA  tstart{dflt:0}    -- start metering
         tstop {dflt:1000} -- cut off
  VAR volt, freq, a, t
      volt -- sample signal 60 Hz
--
  note that to get a reasonably accurate result the period
-- should be a near exact multiple of at least 50 times of
-- the timestep used.  E.g. for 60 Hz timestep = 0.000333
--
-- sin(omega.t) is shown as an example
  HISTORY volt {dflt:0}
--
  INIT freq:=0, tf:= 10000 ENDINIT
--
EXEC
  volt := sin ( 2*pi*60)
  a:= AND ( (volt), (- prevval(volt)), (t-tstart), (tstop-t) )
  if a then
    freq:= 1 / (tf+timestep)
    tf := 0
  else tf := tf + timestep
  endif
ENDEXEC
ENDMODEL
=====

```

### 8.3.2 DEVICE 51, 52 - 'RELAY SWITCHCH'

```

*****
TACS Device 51-52          Relay switchch
=====
MODEL relay_switch  -- relay operated switch - TACS Code 51
                   -- or level triggered switch - TACS Code 52
  DATA  tstart{dflt:0}  -- start metering
         tstop {dflt:1000} -- cut-off
         gain  {dflt:0}
         th_hold {dflt:0}  -- threshold value for operation
         oper {dflt:0}  -- 0 for n/o, 1 for n/c
         type { dflt:0}  -- 0 abs values ( type 51 )
                   1 actual values ( type 52 )
--
-- note: INPUT and OUTPUT statements not required if model is
--       is integrated into another model which generates these
variables
--       see the Primer on the correct use of INPUT / OUTPUT
  INPUT drive_s  -- driving signal
         in1, in2, in3  -- inputs
  OUTPUT rel_out  -- device 51/52 output
--
  VAR rel_out  -- output of 51 or 52
     sumin
  INIT rel_out:= 0 ENDINIT  -- initialize because if t> loop
--
  EXEC
    IF t> tstart and tstop > t then
      sumin := in1 + in2 + in3
      if oper =0 and type = 0 then
        rel_out := (gain *sumin) *  (abs(drive_s) >= abs(th_hold) )
-- n/o
      elseif oper = 0 and type = 1 then
        rel_out := (gain *sumin) *  ( drive_s >= th_hold ) -- n/o
      elseif oper = 1 and type = 0 then
        rel_out := (gain * sumin) * (abs(drive_s) <= abs(th_hold) )
-- n/c
      elseif oper = 1 and type = 1 then
        rel_out := (gain * sumin) * (drive_s <= th_hold ) -- n/c
      endif
    ENDIF
  ENDEXEC
ENDMODEL
=====

```

### 8.3.3 DEVICE 54 - 'PULSE DELAY'

```

*****
TACS Devices 53 and 54          Transport Delay and Pulse Delay
=====
-- MODEL  transport delay and pulse delay TACS Codes 53, 54
--       these can be simulated in MODELS by the delay function and
--       appropriate history declaration
=====

```

### 8.3.4 DEVICE 55, 56, 57 - 'DIGITIZER, POINT BY POINT NONLINEARITY, TIME SEQUENCED SWITCH'

```

*****
TACS Devices 55  Digitizer
                56  Point by point Nonlinearity
                57  Time Sequenced Switch

=====
MODEL  point_list
--      *****
--      simulated in MODELS by the POINT LIST functions
C      function of angle
      VAR x1, e, i, swcont
      INIT e := 0  ENDINIT
--      =====
--      this corresponds to TACS Device 55  Digitizer
      FUNCTION stairs POINTLIST -- staircase x1(t) function
--
--          t          x1
--          ( 0.0,    1.0)
--          ( 0.04,   1.0)
--          ( 0.07,   2.0)
--          ( 0.10,   3.0)
--          ( 0.15,   4.0)
--          ( 0.20,   5.0)
--          ( 0.25,   6.0)
--          ( 0.30,   7.0)
--
--      =====
--      this corresponds to TACS Device 56  Point - by Point Nonlinearity
      FUNCTION saturation POINTLIST -- e(i) saturation function
--
--          i          e
--          ( 0.0,    0.0 )
--          ( 1.0,    0.5 )
--          ( 5.0,    1.0 )
--          ( 5.5,    1.0 ,)
--          ( 6.0,    1.1 )
--          ( 7.0,    1.15 )
--          ( 8.0,    1.2 )
--          ( 10.0,   1.25 )
--          ( 20.0,   1.4 )

```

```

-- =====
--   this corresponds to TACS Device 57 Time Sequenced Switch
--   FUNCTION switch_control POINTLIST
--       t      switch_control
--       ( 0.0      0 )
--       ( 0.1      1 )
--       ( 0.2      0 )
--       ( 0.25     1 )
--       ( 0.40     0 )
EXEC
    if t> 0 then
    x1 := stairs (t,0)    -- example for device 55
    endif
    FOR i := 0 TO 20 BY 1.0 DO
    e := saturation(i)   -- example for device 56
    ENDFOR
    swcont := switch_control(t,0)  -- example for device 57
--
    ENDEXEC
ENDMODEL
    USE point_list AS point_list  -- example only
    timestep min:0
    ENDUSE
C
RECORD    freq_meter.freq AS freq
          relay_switch.rel_out AS relout
          point_list.swcont AS swcont
ENDMODELS
C
=====

```

### 8.3.5 DEVICE 61 - 'SIGNAL SELECTOR'

```

*****
TACS Device 61  Signal selector
--
--   Covered by the algorithm control statements
--   in the MODELS language - see Manual Chapter 2.3.2
=====

*****
TACS Device 62  Sample and Track
=====
MODEL sample_track
--
    DATA tstart {dflt:0}      -- start
        tstop  {dflt:1000}    -- cut off
    VAR track, x, insig, trackout, sampleout
    HISTORY sampleout {dflt:0}
        trackout {dflt:0}
    INIT trackout:=0  sampleout := 0 ENDINIT
--
EXEC
-- *****
--   note: insig and track functions are given here as an example only
--   they are to be replaced by the user's respective functions
--
--   assume that the signal to be sampled is :
--       insig := 2 + 3 * exp (-t) * sin ( 62.8 * t)
--   and the tracking or sampling signal is:
--       track := (t-0.02) mod 0.1 < 0.03
--
-- *****
--   IF t > tstart and tstop > t then
--       output for the tracking option
--       if track > 0 then
--           trackout := insig
--       else trackout := prevval (trackout)
--       endif
--
--       output for the sampling option
--       if track > 0 and prevval (track) = 0 then
--           sampleout := insig
--       else sampleout := prevval ( sampleout)
--       endif
--   ENDIF
ENDEXEC
ENDMODEL
=====

```



### 8.3.6 DEVICE 63 - 'INSTANTENOUS MAXIMUM/MINIMUM

```

*****
TACS Device 63      Instantaneous Maximum/minimum
=====
MODEL inst_maxmin
  DATA_tstart {dflt:0}
        tstop  {dflt:1000}
  VAR in1, in2, in3, max_out, min_out
  INIT max_out := 0  min_out := 0 ENDINIT
--
EXEC
-- *****
-- note: in1, in2, in3 functions are given here as an example only
-- they are to be replaced by the user's respective functions
-- assume that the three input signals are a set of 3-ph voltages
-- at 10 Hz
  in1 := sin ( 62.8 * t)
  in2 := sin ( 62.8 * t - 2*pi/3)
  in3 := sin ( 62.8 * t + 2*pi/3)
-- *****
  IF t > tstart and t < tstop then
    max_out := max ( in1, in2, in3)
    min_out := min ( in1, in2, in3)
  ENDIF
ENDEXEC
ENDMODEL
=====

```

### 8.3.7 DEVICE 64 - 'MAXIMUM/MINIMUM TACKING'

```

*****
TACS Device 64   Maximum/minimum tacking
=====
MODEL maxmin_trak
  DATA tstart {dflt:0}      -- start
        tstop  {dflt:1000}   -- cut off
        reset_value {dflt:0}
  --
  VAR in11, in22      -- input variables
      smax            -- inst. max output
      smin            -- inst. min output
      hold            -- hold signal
      s1              -- sum of in11 and in22
  INIT s1 := 0 smax :=0  smin :=0  hold:=0 ENDINIT
  --
  --
EXEC
  -- *****
  -- note: in11, in22 functions are given here as an example only
  -- they are to be replaced by the user's respective functions
  in11:= sin ( 62.8 * t)  -- 10 Hz sin
  in22:= 4.0 *t
  hold:= 1 * ( t mod 0.2 < 2*timestep )
  -- *****
  s1 := SUM ( 1|in11 + 1|in22 )
  --
  IF t > tstart and tstop > t THEN
    if hold = 0 then
      if s1 >= smax then smax := s1 endif
      if s1 <= smin then smin := s1 endif
    endif
  --
    if hold = 1 then
      smax := reset_value
      smin := reset_value
    endif
  ENDIF
ENDEXEC
ENDMODEL
=====

```

### 8.3.8 DEVICE 65 - 'ACCUMULATOR – COUNTER'

```

*****
TACS Device 65   Accumulator - counter
=====
MODEL accum count  -- Accumulator-counter      TACS   Code 65
  DATA tstart {dflt:0}
        tstop  {dflt:1000}
        reset_value {dflt:0}
  VAR inc1, inc2      -- assumed input
        hold          -- hold signal
        count         -- output
  HISTORY count { dflt :0}
  INIT count:=0      ENDINIT
--
EXEC
-- *****
-- note: inc1, inc2  functions are given here as an example only
-- they are to be replaced by the user's respective functions
  inc1 := sin (628 * t)
  inc2 := exp (2*t)
  hold:= 1 * ( t mod 0.2 < 2*timestep )
-- *****
  IF t > tstart and t < tstop then
    if hold = 0 then
      count := (inc1 + inc2 ) + delay(count,timestep)  endif
    if hold=1 then count := reset_value endif
  ENDIF
ENDEXEC
ENDMODEL

```

### 8.3.9 DEVICE 66 - 'RMS VALUE'

```

*****
TACS Device 66   R.M.S. Value Calculator
=====
MODEL rms
--      calculates the rms value of a function for a specified
frequency
--      Warning!  the period time must be a near exact multiple of the
--      timestep used, otherwise a cumulative round-off error occurs.
--      e.g 0.000166 for a 60 Hz signal
--      DATA  tstart{dflt:0}      -- start metering
--              tstop {dflt:1000}  -- cut-off
--              freq  {dflt:50}    -- frequency
--
--      VAR    volt    -- example only 60 Hz sinusoidal
--              voltrms -- rms value of volt
--              tper    -- period time
--              tt      -- multiplier
--              omega   -- 2 pi f
--
--      HISTORY volt {dflt:0}
--      DELAY CELLS (volt): 1000
--
--      INIT tper:= 1/freq, tt := timestep/tper
--              tper := 1/freq
--              tt := timestep/tper
--              omega := 2*pi*freq
--              voltrms :=0
--      ENDINIT
--      note: volt asssd track functions are given here as an example only
--      they are to be replaced by the user's respective functions
--
--      EXEC
--          if t>tstart and tstop > t then
--              volt := 141 * sin ( omega * t )
--              voltrms:= sqrt(voltrms**2 + tt*(volt**2 - delay(volt, tper)**2))
--          endif
--      ENDEXEC
--      ENDMODEL
=====

```

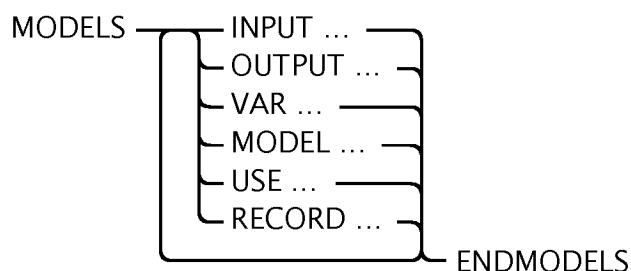
## REFERÊNCIA BIBLIOGRÁFICA

- [1] L. Dubé, B. Ceresoli, K. Fehrle, S. Ghani, **“MODELS PRIMER - For First Time MODELS Users – version 1”**, By Gabor Furst G.Furst Inc, Vancouver, Canada.
- [2] L. Dube, **“USERS GUIDE TO MODELS IN ATP”**. Abril, 1996.
- [3] **“MODELS IN ATP LANGUAGE MANUAL”**, Agosto, 1995.
- [4] L. Dubé, **”Modelling a RL Parallel Circuit Using TYPE-94 Iterated Component”**, DEI Simulation Software USA, fevereiro, 1996.
- [5] H. Wehrend, **“Working with MODELS foreign models in the WATCOM ATP for NT / OS2”**, SEG Germany, fevereiro, 1997.
- [6] L. Dubé, **“How to Use MODELS-Based User-Defined Network Components in ATP”**, DEI Simulation USA, fevereiro, 1997.
- [7] M. Kizilcay, L. Dubé, **“Post Processing Measured Transients Data using MODELS in the ATP-EMTP”**, Fachhochschule Osnabruck, Germany, maio, 1997.
- [8] O. P. Hevia, **“BODE Plot Calculation with ATP MODELS”**, GISEP-UTN Facultad Regional Santa Fé, Argentina, novembro, 2001.
- [9] J. A. Martinez, **“Educational use of EMTP MODELS for the Study OF Rotating Machine Transients”**, Departament d’Enginyeria Eléctrica – Universitat Politècnica de Catalunya, Barcelona Espanha, novembro, 1993.
- [10] J. W. Kalat, **“ATP-EMTP as a Practical Tool for Thermal Network Modeling and Heat Transfer Simulation”**, Warsaw University of Technology, Polônia, novembro, 2001.

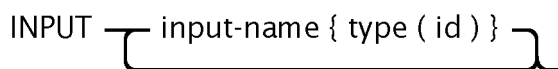
## APÊNDICE A - SINTAXE DO IDIOMA DE MODELOS

Esta é uma referência sumária à sintaxe do idioma de MODELOS. Consiste em toda a estrutura e os diagramas de sintaxe contidos no Manual de Idioma, sem qualquer texto explicativo.

### ➤ Seção MODELS no ATP

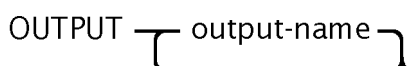


### ➤ Seção Declaração de entrada na MODELS

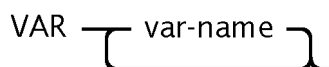


*type*: v, i, switch, mach, imssv, imssi, tacs, pl4, atp

### ➤ Seção Declaração de saída na MODELS



### ➤ Seção Declaração de variável na MODELS

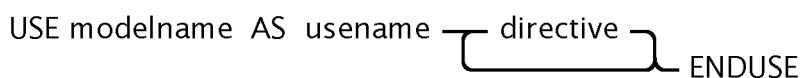


### ➤ Seção Declaração de um modelo na MODELS

declaração de um modelo local

declaração modelo externo

### ➤ Declaração USE na MODELS



➤ Diretiva RECORD na MODELS

RECORD { record-element AS label }

➤ Declaração em um modelo

Elementos constantes

Elementos de dados

Elementos de entrada

Elementos de saída

Elementos variáveis

Funções

Submodelos

➤ Declaração de um elemento constante

CONST { constant-element { VAL: expr } }

➤ Declaração de um elemento de dados

DATA { data-element  
data-element { DFLT: expr } }

➤ Declaração de um elemento de entrada

INPUT { input-element  
input-element { DFLT: expr } }

➤ Declaração de um elemento de saída

OUTPUT { output-element }

➤ Declaração de um elemento de saída

VAR { var-element }

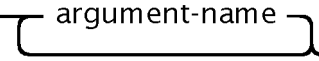
➤ Declaração de uma função

Funções statement

Função point list

Função externa

➤ Declaração de uma função statement

FUNCTION name (  argument-name ) := expr

➤ Declaração de uma função point list

FUNCTION name POINTLIST  ( expr , expr )

➤ Declaração de uma função estrangeira

FUNCTION name FOREIGN idname { IXARG : expr }

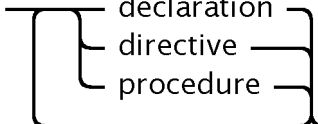

➤ Declaração de um sub-modelo

Modelo local

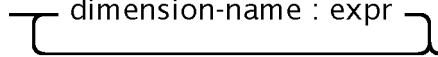
Modelo estrangeiro

Modelo externo

➤ Declaração de um modelo local

MODEL name  declaration  
directive  
procedure  ENDMODEL

➤ Declaração de um modelo estrangeiro

MODEL name FOREIGN idname {  dimension-name : expr } }



➤ Declaração de um modelo externo

MODEL name EXTERNAL

➤ Diretivas de simulação de um modelo

Diretiva history

Diretiva time step

Diretiva de interpolação de entrada

Diretiva delay cells

➤ Diretiva history

HISTORY { history-element  
                  history-element { DFLT: expr } }

➤ Diretiva time step

TIMESTEP { MIN : expr  
                  MAX : expr }

➤ Diretiva interpolation de entrada

INTERPOLATION { DEGREE ( list ) : expr  
                                  DFLT }

➤ Diretiva delay cells

DELAY { CELLS ( list ) : expr  
                                  DFLT }

➤ Procedimento de inicialização

INIT statement-list ENDINIT

➤ Procedimento de execução

EXEC statement-list ENDEXEC

➤ Procedimento de chamada

PROC name stmt-list ENDPROC  
 PROC name ( arg-list ) stmt-list ENDPROC  
 PROC name [ range ] stmt-list ENDPROC  
 PROC name [ range ] ( arg-list ) stmt-list ENDPROC

➤ Statement (declaração)

Designação de declaração

Algoritmo controle de declaração

➤ Designação de declaração

Valor de atribuição

Equações diferenciais

Laplace funções de transferência

Z funções de transferência

Integral valor de atribuição

Declaração da historia

➤ Valor de atribuição

$$\left\{ \begin{array}{l} \text{name} \\ \text{name [ expr ]} \\ \text{name [ expr .. expr ]} \end{array} \right\} := \text{expr}$$

➤ Equações diferenciais

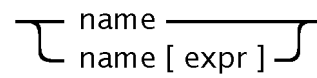
DIFFEQ ( D-polynomial ) | y := x

CDIFFEQ ( D-polynomial ) | y := x

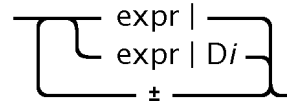
DIFFEQ ( D-polynomial ) | y { limits } := x

CDIFFEQ ( D-polynomial ) | y { limits } := x

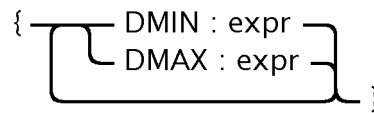
## ➤ Y,x



## ➤ D - polinomial



## ➤ limites



## ➤ Laplace funções de transferência

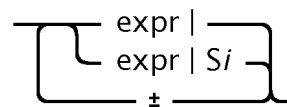
LAPLACE(y / x) := (S-polynomial) / (S-polynomial)

CLAPLACE(y / x) := (S-polynomial) / (S-polynomial)

LAPLACE (y / x) { limites } := (S-polynomial) / (S-polynomial)

CLAPLACE (y / x) { limites } := (S-polynomial) / (S-polynomial)

## ➤ S - Polinomial



## ➤ Z funções de transferência

ZFUN(y / x) := (z<sup>-1</sup>-polynomial) / (z<sup>-1</sup>-polynomial)

CZFUN(y / x) := (z<sup>-1</sup>-polynomial) / (z<sup>-1</sup>-polynomial)

ZFUN(y / x) { limites } := (z<sup>-1</sup>-polynomial) / (z<sup>-1</sup>-polynomial)

CZFUN(y / x) { limites } := (z<sup>-1</sup>-polynomial) / (z<sup>-1</sup>-polynomial)

➤ Integral valor de atribuição

$$\text{integral ( } \left. \begin{array}{l} \text{name} \\ \text{name [ expr ]} \\ \text{name [ expr .. expr ]} \end{array} \right) := \text{expr}$$

$$\text{integral ( name [ expr .. expr ] ) := array expr}$$

➤ Declaração da história

$$\text{histdef ( history-element ) := expr}$$

➤ Declaração de algoritmo de controle

IF declaração

WHILE declaração

FOR declaração

DO declaração

REDO declaração

COMBINE declaração

ERROR declaração

USE declaração

Procedimento de chamada

➤ IF declaração

$$\begin{array}{l} \text{— IF expr THEN statement-list} \\ \left\{ \begin{array}{l} \text{ELSIF expr THEN statement-list} \\ \text{ELSE statement-list} \end{array} \right. \text{ENDIF} \end{array}$$

➤ WHILE Declaração

$$\text{WHILE expr DO statement-list ENDWHILE}$$

➤ FOR Declaração

```

FOR argument-name := value-list
DO statement-list ENDFOR

```

➤ DO Declaração

```

DO statement-list ENDDO

```

➤ REDO Declaração

```

DO ... REDO ... ENDDO

```

➤ COMBINE Declaração

```

COMBINE AS identifier statement-list ENDCOMBINE
COMBINE ITERATE AS identifier statement-list ENDCOMBINE
COMBINE ITERATE { max-iter } AS identifier statement-list
ENDCOMBINE

```

➤ ERROR Declaração

```

ERROR statement-list STOP

```

➤ USE Declaração

```

USE modelname AS username
{
  ITERATE
  directive
}
ENDUSE

```

➤ Procedimento de chamada

instance.procname(arg-list)

WRITE(write-list)

WRITE1(write-list)

WRITE2(write-list)

➤ Declaração de diretivas em USE

USE diretiva de dados

USE diretiva de entrada

USE diretiva de saída

USE diretiva de historia

USE diretiva time step

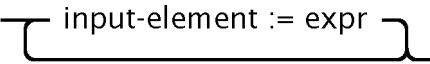
USE diretiva de entrada de interpolação

USE diretiva delay cells

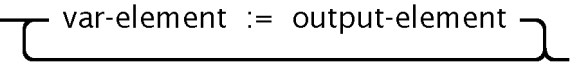
➤ USE diretiva de dados

DATA 

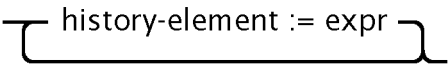
➤ USE Diretiva de entrada

INPUT 

➤ USE Diretiva de saída

OUTPUT 

➤ USE Diretiva da história

HISTORY 

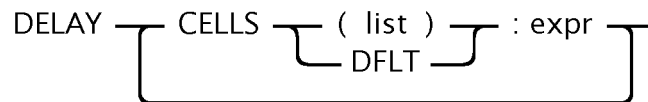
➤ USE diretiva time step

TIMESTEP 

➤ USE diretiva da interpolação de entrada



➤ USE diretiva delay cells



➤ Expressões

Expressão regular

Expressão soma

Expressão derivada

Expressão integral

➤ Expressão regular

regular expression

regular expression { limits }

➤ Expressão soma

$y := \text{sum} ( \text{polynomial} )$

$y := \text{sum} ( \text{polynomial} ) \{ \text{limits} \}$

➤ Expressão derivada

$y := \text{derivpol} ( \text{D-polynomial} ) | x$

$y := \text{derivpol} ( \text{D-polynomial} ) | x \{ \text{limits} \}$

➤ Expressão integral

$y := \text{integral} ( x )$

$y := \text{integral} ( x ) \{ \text{limits} \}$

## APÊNDICE B – PALAVRAS-CHAVES DA LINGUAGEM MODELS

Esta seção provê uma avaliação estruturada condensada da sintaxe do idioma de MODELOS. Mostra só as palavras chaves do idioma, sem os detalhes das regras de sintaxe cheias.

### ➤ start/end

MODELS ... ENDMODELS

### ➤ Elementos de declaração

INPUT ... { *type*( ... ) }

*type*: v, i, switch, mach, imssv, imssi, tacs, pl4, atp

OUTPUT...

VAR ...

### ➤ Modelos de declaração

MODEL ... ENDMODEL

MODEL ... FOREIGN ... { IXDATA: ... IXIN: ... IXOUT: ... IXVAR:  
... }

### ➤ Diretiva USE

USE ... ENDUSE

### ➤ Diretiva RECORD

RECORD ... AS ...



➤ Elementos de declaração

CONST ... { VAL: ... }

DATA ... { DFLT: ... }

INPUT ... { DFLT: ... }

OUTPUT ...

VAR ...

➤ Declaração FUNCTION

FUNCTION ... := ...

FUNCTION ... POINTLIST ( ... , ... ) ( ... , ... ) ...

FUNCTION ... FOREIGN ... { IXARG: ... }

➤ Declaração de submodelos

MODEL ... ENDMODEL

MODEL ... FOREIGN ... { IXDATA: ... IXIN: ... IXOUT: ... IXVAR:  
... }

MODEL ... EXTERNAL

➤ Diretivas de simulação

HISTORY ... { DFLT: ... }

HISTORY INTEGRAL( ... ) { DFLT: ... }

TIMESTEP MIN: ... MAX: ...

INTERPOLATION DEGREE ( ... ) : ...

INTERPOLATION DEGREE DFLT: ...

DELAY CELLS ( ... ) : ...

DELAY CELLS DFLT : ...

➤ Procedimento de simulação

INIT ... ENDINIT

EXEC ... ENDEXEC

PROC ... ENDPROC

➤ declarações de tarefa

... := ...

DIFFEQ( ... ) | ... { DMIN: ... DMAX: ... } := ...

CDIFFEQ( ... ) | ... { DMIN: ... DMAX: ... } := ...

LAPLACE( ... ) | ... { DMIN: ... DMAX: ... } := ...

CLAPLACE( ... ) | ... { DMIN: ... DMAX: ... } := ...

ZFUN( ... ) | ... { DMIN: ... DMAX: ... } := ...

CZFUN( ... ) | ... { DMIN: ... DMAX: ... } := ...

INTEGRAL( ... ) := ...

HISTDEF( ... ) := ...

➤ Declaração de algoritmo de controle

IF ... THEN ... ELSIF ... THEN ... ELSE ... ENDIF

WHILE ... DO ... ENDWHILE

FOR ... := ... DO ... ENDFOR

DO ... REDO ... ENDDO

COMBINE AS ... ENDCOMBINE

COMBINE ITERATE AS ... ENDCOMBINE

COMBINE ITERATE { ... } AS ... ENDCOMBINE

ERROR ... STOP

USE ... ENDUSE

instance.procname( ... )

WRITE( ... )

WRITE1( ... )

WRITE2( ... )

➤ Palavras-chaves da diretiva USE

DATA ... := ...

INPUT ... := ...

OUTPUT ... := ...

HISTORY ... := ...

TIMESTEP MIN: ... MAX: ...

INTERPOLATION DEGREE( ... ) : ...

INTERPOLATION DEGREE DFLT: ...

DELAY CELLS( ... ) : ...

DELAY CELLS DFLT : ...

➤ Palavras-chaves das expressões

regular-expression { MIN: ... MAX: ... }

SUM( ... ) { MIN: ... MAX: ... }

DERIVPOL( ... ) { MIN: ... MAX: ... }

INTEGRAL( ... ) { DMIN: ... DMAX: ... }

➤ Palavras-chaves de expressões regulares

... OR ...

... AND ...

... > ...

... >= ...

... < ...

... <= ...

... = ...

... <> ...

... + ...

... - ...

... \* ...

... / ...

... \*\* ...

... MOD ...

*num value*

[ ... , ... ... ]

...

- ...

NOT ...

*name*

*name* [ ... ]

*name* [ ... .. ... ]

*function name* ( ... , ... ... )

( *regular expression* )

➤ Palavras-chave de comentários

-- ... end-of-line

COMMENT ... ENDCOMMENT

ILLUSTRATION ... ENDILLUSTRATION

## APÊNDICE C – CONSTANTES PRÉ-DEFINIDAS E VARIÁVEIS

O idioma de MODELOS provê um jogo de constantes pré-definidas e variáveis globais e locais pré-definidas. Os nomes deles/delas estão pré-definidos e são visíveis em qualquer modelo. Os valores das constantes pré-definidas serem - definido globalmente. Os valores das variáveis pré-definidas levam informação de simulação que pode ser acessada dentro de qualquer modelo.

### ➤ Constantes pré-definidas

As constantes pré-definidas pré-nomearam valores e podem ser usadas dentro de qualquer expressão, em qualquer modelo de uma simulação. Eles são:

<i>pi</i>	3.14159...
<i>inf</i>	a large number, typically $10^{20}$
<i>undefined</i>	88888.88888
<i>false, true</i>	the equivalent numerical values 0 and 1
<i>no, yes</i>	the equivalent numerical values 0 and 1
<i>open, closed</i>	the equivalent numerical values 0 and 1
<i>off, on</i>	the equivalent numerical values 0 and 1

O inf de valor de infinidade é dependente na plataforma de computador que é usado. O valor indefinido é o valor levado por qualquer variável que ainda não foi nomeado um valor durante uma simulação. É

representado o valor das constantes lógicas internamente como os valores 0 e 1 numéricos.

➤ Variáveis pré-definidas

Três variáveis estão globalmente definidas em uma simulação. Eles levam o valor do arranque e terminando tempo da simulação e o valor de passo de tempo impôs pelo ambiente, concha ou aplicação que controlam a simulação.

O valor destas variáveis pode ser acessado através de nome dentro de qualquer expressão, em qualquer modelo de uma simulação. Eles são:

<i><b>starttime</b></i>	the value of $t$ at which the simulation started
<i><b>stoptime</b></i>	the value of $t$ at which the simulation will end
<i><b>startstep</b></i>	the value of the initial time step at the outermost level

Os valores de *starttime* e *startstep* são fixos ao começo da simulação. O valor de *stoptime* está definido pela aplicação que controla a simulação e pode mudar durante uma simulação.

➤ Variáveis pré-definidas

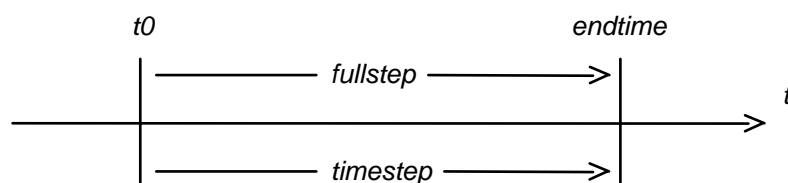
Os nomes das variáveis locais pré-definidas são o mesmo em todos os modelos de uma simulação, mas os valores deles/delas são locais a cada exemplo de cada modelo usado em uma simulação.

O valor destas variáveis pode ser acessado através de nome dentro de qualquer expressão, em qualquer modelo de uma simulação. Eles são:

<i>t</i>	the present simulation time
<i>prevtime</i>	the simulation time at the last update
<i>timestep</i>	the size of the interval between <i>prevtime</i> and <i>t</i>
<i>endtime</i>	the target simulation time of a USE call
<i>fullstep</i>	the size of the interval over which to update at USE call
<i>minstep</i>	the variable lower limit imposed on <i>timestep</i>
<i>maxstep</i>	the variable upper limit imposed on <i>timestep</i>

Estas variáveis levam informação sobre como o estado de um modelo é atualizado de um tempo de simulação prévio a um tempo de simulação designado, em cada chamada de atualização para um exemplo modelo de uma declaração de USO.

Em um caso onde *timestep*, o passo de tempo usado no modelo chamado, é o mesmo tamanho como *fullstep*, o intervalo de chamada que também é o passo de tempo do modelo de chamada, a relação entre estas variáveis é simples:



$$\Rightarrow \text{minstep} \leq \text{fullstep} \leq \text{maxstep}$$

$$\Rightarrow \text{timestep} = \text{fullstep}$$

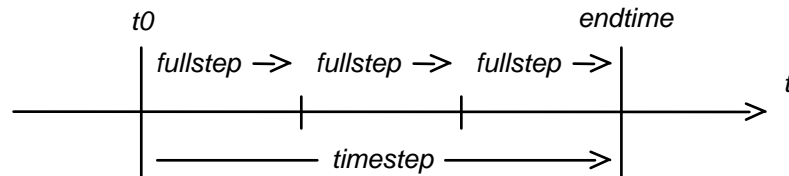
$$\Rightarrow t = \text{endtime} = t0 + \text{fullstep} = t0 + \text{timestep}$$

$$\Rightarrow \text{prevtime} = t0$$

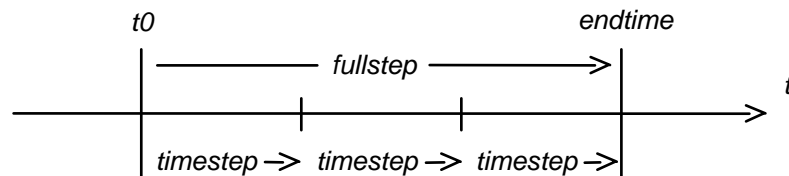
Em um caso onde *fullstep* é menor que *minstep*, o exemplo modelo não se atualiza ao tempo externo, porque é considerado que o aumento de



tempo é muito pequeno para mudar o estado presente do modelo significativamente. As esperas modelo até  $\text{endtime} - t_0 \geq \text{minstep}$  para atualizar seu estado, em um passo de tempo grande.



Em um caso onde *fullstep* é maior que o passo de tempo maior permitiu *maxstep*, o passo inteiro é subdividido em substituto-passos de um tamanho que satisfaz o *maxstep* limite superior e o exemplo modelo é executado sucessivamente para cada substituto-passo, até que o *endtime* de tempo designado é alcançado. Só então é controle devolvido ao modelo de chamada, junto com os valores de produção do modelo a tempo  $t = \text{endtime}$ . Para este caso, é a relação entre as variáveis como segue:



$$\Rightarrow \text{fullstep} > \text{maxstep}$$

$$\Rightarrow \text{timestep} \leq \text{maxstep}$$

$$\Rightarrow \text{timestep} < \text{fullstep}$$

$$\Rightarrow t = \{ t_0 + \text{timestep}, t_0 + 2 * \text{timestep}, \dots, \text{endtime} \}$$

$$\Rightarrow \text{prevtime} = \{ t_0, t_0 + \text{timestep}, \dots, \text{endtime} - \text{timestep} \}$$

## APÊNDICE D – FUNÇÕES PRÉ-DEFINIDAS

MODELOS provêem um jogo de funções pré-definidas. Há funções numéricas embutidas, funções lógicas e funções de valor fortuitas. Também há um jogo de funções de simulação que fazem referência explícita ou implícita à história armazenada das variáveis de um modelo.

MODELOS permitem usar funções de único-argumento com listas de argumentos para estender o uso de funções de único-argumento para formar aritmética. Quando determinado uma lista de argumentos como contribuição, uma função que normalmente é uma função de único-argumento devolverá uma lista correspondente de únicos valores como produção, a pessoa produziu para cada argumento (um-para-um).

### ➤ **Funções numéricas**

Esta é uma lista de todas as funções pré-definidas que devolvem valores numéricos. Eles se agrupam como um-para-um funciona, dois-argumento funciona, enquanto devolvendo um único valor e n-argumento funciona, enquanto devolvendo um único valor.

### ➤ **1-para-1 funções que devolvem um valor por cada argumento**

Quando usado com um único argumento, estas funções devolvem um único valor de produção. Quando usado com uma lista de valores de argumento, eles devolvem como muito valor de produção, cada correspondendo à função que é aplicada a cada valor proveu na lista de argumento.

Isto é ilustrado debaixo de usar a função de valor absoluto:

```

abs(a)          -- returns one value: the absolute value of a
abs(expression) -- returns one value: the absolute value of the expression
abs(a, b)       -- returns [abs(a), abs(b)]
abs(expression list) -- returns array of absolute values of each list element
abs(x[1..2])    -- returns [abs(x[1]), abs(x[2])]
abs(a, x[1..2]) -- returns [abs(a), abs(x[1]), abs(x[2])]
abs(array expression) -- returns array of absolute values of each array element

```

Seguir é o 1-para-1 funções numéricas:

<b><i>abs( )</i></b>	absolute value	
<b><i>sqrt( )</i></b>	square root	arg<0 produces error
<b><i>exp( )</i></b>	exponential	
<b><i>ln( )</i></b>	natural log	arg<0 produces error
<b><i>log10( )</i></b>	base 10 log	arg<0 produces error
<b><i>log2( )</i></b>	base 2 log	arg<0 produces error
<b><i>recip( )</i></b>	protected division	division by zero returns value=inf
<b><i>factorial( )</i></b>	factorial	arg<0 produces error
<b><i>trunc( )</i></b>	integer part	
<b><i>fract( )</i></b>	fractional part	
<b><i>round( )</i></b>	rounding	
<b><i>sign( )</i></b>	sign function	-1 if <0; 0 if 0; +1 if >0
<b><i>rad( )</i></b>	degrees to radians	

<i>deg( )</i>	radians to degrees	
<i>sin( )</i> , <i>cos( )</i> , <i>tan( )</i>	trigonometric	arg in radians
<i>asin( )</i> , <i>acos( )</i> , <i>atan( )</i>	inverse trigonometric	return value in radians
<i>sinh( )</i> , <i>cosh( )</i> , <i>tanh( )</i>	hyperbolic	arg in radians
<i>asinh( )</i> , <i>acosh( )</i> , <i>atanh( )</i>	inverse hyperbolic	return value in radians

➤ 2-argumento funciona, enquanto devolvendo um único valor.

Estas funções sempre devolvem um único valor e não pode ser estendido para formar aritmética como o prévio 1-para-1 funções. Cada um dos dois argumentos pode ser dado na forma de uma expressão de único-valor ou os dois argumentos podem ser especificados junto como uma expressão de ordem de dois-elemento. Isto que usa a função de *atan2( )* é ilustrado aqui:

```
atan2(a, b)      -- returns one value: arctan(a/b)
atan2(expr1, expr2) -- returns one value: arctan(expr1/expr2)
atan2(x[1..2])   -- returns one value: arctan(x[1]/x[2])
```

Seguir é o 2-para-1 funções numéricas:

<i>atan2( )</i>	4-quadrant arctan(a/b)	accepts zero-value arguments
<i>binom( )</i>	binomial function C(n,r)	returns $n! / (n-r)! r!$
<i>permut( )</i>	permutation function P(n,r)	returns $n! / (n-r)!$

- n-argumento funciona, enquanto devolvendo um único valor

Estas funções levam como contribuição uma lista de valores e devolvem um único valor. A lista de expressões de contribuição pode incluir únicos valores e valores de ordem. O exemplo seguinte ilustra como os valores de contribuição podem ser especificados:

```
min(a, b)          -- returns the minimum value of the two arguments a and b
min(expr1, expr2, ...) -- returns the minimum value of the argument list
min(x[1..2])       -- returns the minimum value of the two args x[1] and x[2]
min(a, x[1..2])    -- returns the minimum value of a, x[1], and x[2]
min(array expression) -- returns the minimum value of all array elements
```

Seguir é o n-para-1 funções numéricas:

<b><i>min</i></b> ( )	minimum value	
<b><i>max</i></b> ( )	maximum value	
<b><i>norm</i></b> ( )	Euclidian norm, or magnitude	returns sqrt(sum(arg <sup>2</sup> )))

- Funções lógicas

Esta é uma lista de todas as funções pré-definidas que devolvem valores lógicos. Eles se agrupam como um-para-um funciona e n-argumento funciona, enquanto devolvendo um único valor.

Os valores lógicos verdadeiro e falso é representado em MODELOS pelos valores 1 e 0 numéricos. Conversão entre valores numéricos e lógicos é automática e é determinado pelo contexto no qual eles são usados. Isto permite usar valores numéricos em um contexto lógico (> 0

são verdades,  $< = 0$  são falsos) e valores lógicos ser usado em um contexto numérico (verdadeiro é 1, falso é 0).

➤ 1-para-1 funções que devolvem um valor por cada argumento

Há só uma função nesta categoria. Provê conversão explícita de numérico para lógico ("Booleano").

Quando usado com um único argumento, devolve um único valor de produção. Quando usado com uma lista de valores de argumento, isto retorna como muitos valores de produção, correspondendo a cada valor na lista de argumento.

<b><i>bool( )</i></b>	numerical to logical	returns <i>true</i> if $\text{arg} > 0$ , <i>false</i> if $\text{arg} \leq 0$
-----------------------	----------------------	-------------------------------------------------------------------------------

Aqui são exemplos mostrando como a função pode ser usada:

<b><i>bool(a)</i></b>	<b>-- returns one value: conversion value of a</b>
<b><i>bool(expression)</i></b>	<b>-- returns one value: conversion value of the expression</b>
<b><i>bool(a, b)</i></b>	<b>-- returns [bool(a), bool(b)]</b>
<b><i>bool(expression list)</i></b>	<b>-- returns array of conversion value of each list element</b>
<b><i>bool(x[1..2])</i></b>	<b>-- returns [bool(x[1]), bool(x[2])]</b>
<b><i>bool(a, x[1..2])</i></b>	<b>-- returns [bool(a), bool(x[1]), bool(x[2])]</b>
<b><i>bool(array expression)</i></b>	<b>-- returns array of conversion value of each element</b>

➤ n-argumento funciona, enquanto devolvendo um único valor

Estas funções levam como contribuição uma lista de valores e devolvem um único valor lógico. A lista de expressões de contribuição pode

incluir únicos valores e valores de ordem. São esperados valores lógicos por cada argumento. Se são providos valores numéricos, eles são convertidos automaticamente para o lógico deles/delas equivalente, usando a mesma regra como mostrado acima. O exemplo seguinte ilustra como os valores de contribuição podem ser especificados:

```
and(a, b)          -- returns true only if a and b are true
and(expr1, expr2, ...) -- returns true only if all argument values are true
and(x[1..2])       -- returns true only if x[1] and x[2] are true
and(a, x[1..2])    -- returns true only if a and x[1] and x[2] are true
and(array expression) -- returns true only if all array elements are true
```

Seguir é o n-para-1 funções lógicas:

<b><i>and( )</i></b>	logical AND	returns <i>true</i> only if all args are <i>true</i>
<b><i>or( )</i></b>	logical OR	returns <i>true</i> if one or more args are <i>true</i>
<b><i>nand( )</i></b>	logical NOT AND	returns <i>true</i> if one or more args are <i>false</i>
<b><i>nor( )</i></b>	logical NOT OR	returns <i>true</i> only if all args are <i>false</i>
<b><i>xor( )</i></b>	logical exclusive OR	returns <i>true</i> if only one arg is <i>true</i>

➤ Funções de valores aleatórios

Há uma função de valor fortuita neste momento pré-definido em MODELOS. A função não usa nenhum argumento e devolve um valor entre zero e um:

<b><i>random( )</i></b>	random value	returns a random value between 0 and 1
-------------------------	--------------	----------------------------------------

A sucessão de números fortuitos gerada de chamadas repetidas ao acaso de função () é reprodutível de uma corrida de simulação para o próximo. Gerar sucessões não-reprodutíveis requer uma chamada pelo programa para a rotina de data / tempo do sistema operacional do computador. Na versão de Fortran presente de MODELOS, isto não está disponível, porque o Fortran não provê uma função standard para fazer isto. Quando contanto, é compilador-dependente. Na versão de C futura de MODELOS, estará disponível, porque C provê uma função compilador-independente standard para isto.

Enquanto isso foram pré-unidas duas funções de C estrangeiras a ATP: C\_seed () para gerar uma semente nova e C\_random () para gerar uma sucessão de números fortuitos baseado naquela semente. C\_seed (x) usa um falso argumento de qualquer valor. Não devolve nenhum valor. C\_random (x) usa um falso argumento de qualquer valor. Devolve um valor entre 0 e 1. O uso destas funções não afeta a sucessão de números gerada pelos MODELOS funcione fortuito ().

O código de fonte para as duas funções de C é o seguinte:

```
#include <stdlib.h>

double C_SEED(void) /* randomly seeds the C library function rand() */
{ date_time_seed();
  return 0;
}

double C_RANDOM(void) /* returns a random number between 0 and 1 */
{ return (double)rand()/(double)RAND_MAX;
}
```

Estas funções podem ser declaradas em um modelo como segue:



```

FUNCTION seed FOREIGN C_SEED {ixarg: 1}
FUNCTION rand FOREIGN C_RANDOM {ixarg: 1}

```

Onde os nomes C\_SEED e C\_RANDOM já são registrados em fgnfun de sequência de dados no arquivo nomeado fgnmod.para de ATP.

Depois que fosse declarado em um modelo, as funções podem ser usadas então como no exemplo seguinte:

```

INIT
  dum := seed(0)      -- functions in MODELS can only be used inside an
expression
  ...
ENDINIT
EXEC
  ...
  y := x * rand(0)    -- returns a value between 0 and 1, then multiplied by x
  ...
ENDEXEC

```

### ➤ Funções de simulação

MODELOS provêem simulação especial não funciona disponível em linguagens de programação regular, para o fato que eles fazem uma referência implícita ao valor avançando de tempo e para a história acumulada das variáveis para as quais eles são aplicados. Estas são funções como derivado, previamente avalie, etc., como mostrado abaixo.

Ao usar as funções de simulação, o primeiro argumento na lista de argumento sempre é um único elemento de valor-propriedade, por exemplo,  $x$ , ou  $y[n]$ , e nunca é uma expressão. Isto é porque uma expressão tem nenhuma história à qual pode recorrer ao calcular o valor da função.

Algumas funções de simulação requerem um segundo argumento, por exemplo, que o valor da demora aplicou a uma variável.

Todas as funções de 2-argumentos, menos `histval()`, aceitam um terceiro argumento opcional para indicar o método de interpolação usado pela função. Só os valores 0, 1 e 2 são reconhecidos para isto, enquanto indicando respectivamente: nenhuma interpolação (o sinal varia em passos), interpolação linear e interpolação quadrática. Interpolação linear é a falta e o "1" sempre pode ser omitida.

As funções de simulação são:

<i>deriv(x)</i>	$dx/dt$ , calculated at end of step
<i>deriv2(x)</i>	$d^2x/dt^2$ , calculated at mid-step
<i>prevval(x)</i>	value of $x$ at previous time step
<i>delay(x, dval)</i> <i>delay(x, dval, pol)</i>	value of $x$ at time $(t - dval)$
<i>predval(x, tval)</i> <i>predval(x, tval, pol)</i>	predicted value of $x$ at time $tval$
<i>predtime(x, xval)</i> <i>predtime(x, xval, pol)</i>	predicted time when $x$ will equal $xval$
<i>backval(x, tval)</i> <i>backval(x, tval, pol)</i>	value of $x(tval)$ within last step interval

<b><i>backtime(x, xval)</i></b> <b><i>backtime(x, xval, pol)</i></b>	time when $x$ was equal to $xval$ within last step
<b><i>histval(x, tval)</i></b>	value of $x(tval)$ , using the history expression associated with $x$
<b><i>histdef(x)</i></b>	value of the history expression of $x$ evaluated at generic time $t$ , equivalent to writing $histval(x,t)$

➤ **deriv(x)**

O tempo derivado de  $x$  é calculado com a suposição que é linear em cima de cada intervalo de passo de tempo, quer dizer, que o segundo derivado é constante em cima do intervalo. O programa calcula as duas derivadas de meio-passo primeiro

$$\text{deriv}_{0,5} = \frac{x(t_0) - x(t_1)}{t_0 - t_1}$$

$$\text{deriv}_{1,5} = \frac{x(t_1) - x(t_2)}{t_1 - t_2}$$

Para calcular o segundo derivado a  $t_1$ :

$$\text{deriv}_{2,t_1} = \frac{\text{deriv}_{0,5} - \text{deriv}_{1,5}}{t_{0,5} - t_{1,5}}$$

Que é usado extrapolar o primeiro meio-passo linearmente derivado,  $\text{deriv}_0$ , então. 5, para o fim do intervalo, que usam o segundo derivado como seu declive:

$$\text{deriv}_{t_0} = \text{deriv}_{0,5} + \text{deriv}_{2,t_1} \frac{t_0 - t_1}{2}$$

Quando o passo de tempo for bastante pequeno para cobrir as frequências mais altas do sinal, este método é mais preciso que usando o meio-passo simples derivado.

Porém, o derivado de meio-passo é menos sensível a descontinuidades e deveria ser usado em vez de `deriv()` quando ao passo de tempo não é correspondido bem aos conteúdos de frequência do sinal, fazendo:

```
y:=(x-prevval(x))/timestep
```

Em vez de usar a função embutida:

```
y:=deriv(x)
```

É possível aplicar limites ao valor calculado de um derivado. Porque a função não leva nenhuma variável interna de um passo para o próximo, um limite dinâmico não tem nenhum significado para o derivado. Pode ser aplicado só limite estático a um derivado, como parte da expressão onde a função é usada:

```
y:=deriv(x) {min:expression, max:expression}
```

#### ➤ `deriv2(x)`

A segunda ordem derivado de  $x$  com respeito a tempo,  $d^2x/dt^2$ , é calculado como o meio-passo derivado do primeiro derivado, isto calculado a meio-passo para os últimos dois intervalos. O método leva em conta a possibilidade para o passo de tempo ser variável.

O segundo derivado é usando calculado:

$$\frac{\text{deriv}_{0,5} - \text{deriv}_{1,5}}{t_{0,5} - t_{1,5}}$$

Ou, substituindo com os detalhes de cálculo cheios:

$$\frac{\frac{x(t_0) - x(t_1)}{t_0 - t_1} - \frac{x(t_1) - x(t_2)}{t_1 - t_2}}{\frac{t_0 + t_1}{2} - \frac{t_1 + t_2}{2}}$$

Quando o passo de tempo for constante, isto é equivalente a calcular o segundo derivado como:

$$\frac{x(t) - 2x(t - \Delta t) + x(t - 2\Delta t)}{\Delta t^2}$$

Deve ser observado que o valor calculado é na realidade o valor do segundo derivado ao  $t_1$  de tempo prévio, por causa do método de meio-passo. Isto significa que o valor do segundo derivado a tempo  $t$  só busca disponível uma demora de uma vez de tempo.

O segundo derivado de uma variável pode ser usado em qualquer expressão, por exemplo:

`y:=deriv2(x)`

É possível aplicar limites ao valor calculado de um segundo derivado. Porque a função não leva nenhuma variável interna de um passo para o próximo, um limite dinâmico não tem nenhum significado para esta operação. Pode ser aplicado só limite estático a um segundo derivado, como parte da expressão onde a função é usada:

`y:=deriv2(x) {min:expression, max:expression}`

➤ prevval(x)

Esta função devolve  $x(t - \Delta t)$ , o valor de uma variável ao passo de tempo prévio. Pode ser usado em qualquer expressão, por exemplo:

```
y:=prevval(x)
```

Podem ser aplicados limites estáticos como parte da expressão onde a função é usada:

```
y:=prevval(x) {min:expression, max:expression}
```

➤ delay(x, dval, pol)

Esta função devolve  $x(t - dval)$ , o valor de uma variável a um último tempo da simulação.

Quando isso pedir além de cachoeiras de tempo antes do começo da simulação, isso é antes do valor da variável pôde ser começado localizado, a função de demora () usa a expressão de história associada com aquela variável para calcular o valor passado e devolve o histval de valor (x, t-dval).

Quando os pediram além de cachoeiras de tempo depois do começo da simulação onde localizou valores da variável estiverem disponíveis, interpolação é usada para casos onde o t-dval de valor não corresponde exatamente a um dos momentos discretos da simulação. Por padrão, quando o terceiro argumento não é especificado, interpolação linear é usada. Caso contrário, o argumento de pol pode ser dado um valor de 0, 1 ou 2, para passo, interpolação linear ou quadrática.

Usando a função de demora pode requerer uma quantia considerável de armazenamento por manter rasto dos valores passados de uma variável (veja a simulação CELAS de DEMORA diretivas). Porém, o cálculo é rápido, como usa uma procura binária para localizar os valores armazenados.

➤ `predval(x, tval, pol)`

Esta função devolve  $x(tval)$ , o valor predito de uma variável há um tempo futuro da simulação.

O valor é extrapolação usando calculada dos mais recentes valores da variável. Por padrão, quando o terceiro argumento não é especificado, extrapolação linear é usada. Caso contrário, o argumento de `pol` pode ser dado um valor de 0, 1 ou 2, para passo, interpolação linear ou quadrática. Usando extrapolação de passo não é útil neste caso, porque devolve o valor presente da variável simplesmente.

➤ `predtime(x, xval, pol)`

Esta função devolve  $t(xval)$ , o tempo futuro predito quando a variável levará o `xval` de valor.

O valor é extrapolação usando calculada dos mais recentes valores da variável. Por padrão, quando o terceiro argumento não é especificado, extrapolação linear é usada. Caso contrário, o argumento de `pol` pode ser dado um valor de 0, 1 ou 2, para passo, interpolação linear ou quadrática. Usando extrapolação de passo não é útil neste caso, porque se `xval` for diferente de  $x(t)$ , então aquele valor que usa extrapolação de passo não pode ser alcançado.

Quando o valor extrapolado da variável não puder alcançar o `xval` de valor, a função devolve o valor indefinido, iguale a 88888.88888.

➤ `backval(x, tval, pol)`

Esta função devolve o valor  $x$  ( $tval$ ), o valor de uma variável a um último tempo da simulação, com a restrição que tem que cair dentro do último intervalo de passo. Para acesso para valores passados além do passo prévio, nós precisamos usar a função de demora ().

Ao primeiro passo de uma simulação, o último tempo pedido cairá antes do começo da simulação, quer dizer, antes do valor da variável pôde ser começado localizado. Neste caso, funciona o `backval ()` usa a expressão de história associada com aquela variável para calcular o valor passado e devolve o `histval` de valor ( $x$ ,  $tval$ ).

Quando os pediram além de cachoeiras de tempo depois do começo da simulação onde localizou valores da variável estiverem disponíveis, interpolação é usada para calcular o valor da variável entre os pontos de simulação. Por padrão, quando o terceiro argumento não é especificado, interpolação linear é usada. Caso contrário, o argumento de `pol` pode ser dado um valor de 0, 1 ou 2, para passo, interpolação linear ou quadrática.

➤ `backtime(x, xval, pol)`

Esta função devolve o tempo  $t$  ( $xval$ ) ao qual a variável teve o  $xval$  de valor, com a restrição que o tempo tem que cair dentro do último intervalo de passo.

O valor é interpolação usando calculada dos mais recentes valores da variável. Por padrão, quando o terceiro argumento não é especificado, interpolação linear é usada. Caso contrário, o argumento de `pol` pode ser dado um valor de 0, 1 ou 2, para passo, interpolação linear ou quadrática. Usando extrapolação de passo não é útil neste caso,



porque se  $x_{val}$  for diferente de  $x(t-timestep)$ , então aquele valor que usa interpolação de passo não pode ser alcançado.

Quando o valor interpolado da variável não puder alcançar o  $x_{val}$  de valor, a função devolve o valor indefinido, igual a 88888.88888.

➤ `histval(x, tval)`

Esta função usa a expressão de história especificada para o  $x$  variável para calcular o valor de  $x(tval)$ . De nenhuma interpolação é precisada, porque a expressão de história proverá um valor calculado a  $t$  igual para  $tval$ . Embora seja pretendido que uma expressão de história é usada para valores calculados de uma variável a valores de  $t$  que precede o começo da simulação, a função de `histval()` pode ser usada sem qualquer restrição no valor de  $tval$ .

Veja a seção em diretiva de simulação para uma discussão em nomear uma expressão de história a uma variável.

➤ `histdef(x)`

Usando esta função é equivalente a usar `histval(x, t)`. Para qualquer valor de  $t$ , esta função devolverá o valor correspondente da expressão de história associado com o  $x$  variável. No princípio olhe, isto pode parecer inútil. Fica mais interessante quando nós considerarmos isso durante a simulação, o resolutor manipula o valor do  $t$  variável atrás das cenas sempre que precisa avaliar uma expressão de história em certo momento  $t$ . Usando o `histdef` de função ( $x$ ) dentro da expressão que define a expressão de história de outro  $y$  variável nos dá um modo para fazer uma expressão de história dependente na expressão de história definido para outra variável. Por exemplo, nós usaríamos isto em uma declaração de história como segue:

$$\text{HISTORY } y \{ \text{dflt: histdef}(x) - 2^* \text{histdef}(w) \}$$

Ou em uma tarefa de história em uma declaração de USO:

$$\text{HISTORY } y := \text{histdef}(x) - 2^* \text{histdef}(w)$$

Ou em uma tarefa de história no procedimento de INIT:

$$\text{histdef}(y) := \text{histdef}(x) - 2^* \text{histdef}(w)$$

Em cada caso, teria sido equivalente para escrever a expressão da expressão de história como:

$$\text{histval}(x,t) - 2^* \text{histval}(w,t)$$

Acessar o valor da expressão de história de x e w a qualquer valor de t selecionou na ocasião pelo resolutor avalia a expressão.