

## Appendix 5 The Fortran Simulation Translator (FST), a simulation language\*

D.W.G. van Kraalingen, C. Rappoldt & H.H. van Laar

### A5.1 Introduction

The FST language and the corresponding FST software feature a powerful and easy-to-use simulation language providing clear error messages. The syntax of the FST language was based on the syntax of CSMP (IBM, 1975). FST was developed because there was a need for a simple simulation language that, at the same time, allows the user to shift to the more powerful and flexible simulation environment that Fortran provides. This shift is made easy because the FST translator translates the FST source file into a clean and versatile Fortran program and also generates corresponding data files. In 2001, the C.T. de Wit Graduate School for Production Ecology and Resource Conservation of Wageningen University sponsored a project to adapt the FST software for Windows, FSTWin, an easy to use shell is available to run the model, and to display the model results graphically, and clear error messages are given. FSTWin needs a Compaq Visual Fortran compiler. Primarily, FST should be seen as a language for education and simple modelling purposes. The quality of the generated Fortran, however, provides an excellent starting point for users who need even more flexibility than FST provides. How these generated Fortran routines work is discussed in Chapter 10 in Leffelaar (1998).

In this Appendix, the simple model for the simulation of logistic growth of yeast will be used to explain FST. Further, a more detailed description of the features of FST is given.

### A5.2 The structure of the model

A simulation program in FST should start with a `TITLE` keyword containing a short identification of the program. In FST, three sections can be distinguished: `INITIAL`, `DYNAMIC` and `TERMINAL`. These keywords indicate that the computations must be performed before, during and after a simulation run, respectively. Each section keyword closes the former section. If there is no section specified, FST considers the program to be in the `DYNAMIC` section. The entire program is terminated by the keyword `END` (see Listing A5.1).

The `INITIAL` section can well be used to specify the input data (initial conditions and parameters) and the time variables, and to define data output and the integration method used in the model. Furthermore, the computation of results that are used as input for the dynamic section of the program may be executed here. The use of the `INITIAL` section is optional.

The `DYNAMIC` section contains the complete description of the model dynamics,

\* Revised version of Appendix 5 in Goudriaan, J. & H.H. van Laar, 1994. Modelling potential crop growth processes. Kluwer Academic Publishers, Dordrecht. 219

together with any other computation required during the simulation. It is therefore normally the most extensive section in a model.

The `TERMINAL` section can be used for computations and specific output that is only available at the end of the simulation run. This could be a computation based on the final values of one or more variables. As in the `INITIAL` section, the computations are executed only once. The `TERMINAL` section is also optional.

FST is provided with a sorting algorithm to free the user from the task of correctly sequencing the statements. Sorting of the statements and checking of the model integrity are done for each section separately.

### A5.3 FST example program simulating logistic growth

The development of an FST simulation model will be demonstrated by solving the differential equation for logistic growth of yeast:

$$dY/dt = RGR \cdot Y \cdot (1 - Y/Y_{\max}) \quad (\text{A5.1})$$

In this example,  $Y$  is the amount of yeast (state variable) and  $dY/dt$  the rate of yeast growth (rate variable). In FST, this equation is programmed by the following two statements:

```
Y = INTGRL (IY,RV)
RV = RGR*Y*(1.-Y/YMAX)
```

The first line defines the name of the state variable ( $Y$ ) and the name of the rate of change of  $Y$  ( $RV$ ) (for terminology see Chapter 1 in Leffelaar, 1993 or 1998). The second line is the actual calculation of the rate of change of  $Y$ . Several additional values have to be defined to be able to calculate the dynamics of logistic growth with these two program lines, such as the value of the relative growth rate ( $RGR$ ), the initial value of the quantity  $Y$  ( $IY$ ), and the maximum value that the quantity of  $Y$  can attain ( $YMAX$ ). The quantities  $RGR$  and  $YMAX$  are called *parameters*, because they should be constant during the model execution. The quantity  $IY$  is called an *initial constant* because it specifies the start value of a state variable. The value of model parameters is defined in a `PARAMETER` statement; initial constants are defined in an `INCON` statement. Suppose we want to begin the simulation with a value of 1 for the state variable  $Y$ , a relative growth rate of 0.2 and a maximum value of  $Y$  ( $YMAX$ ) of 20. The model will then be:

```
TITLE Logistic growth of yeast
INITIAL
  INCON IY = 1.
  PARAMETER RGR = 0.2; YMAX = 20.
DYNAMIC
  Y = INTGRL (IY,RV)
  RV = RGR*Y*(1.-Y/YMAX)
END
```

The next step is to choose a solution scheme such as Euler (with a fixed time step of integration), or Runge-Kutta (a variable time step of integration) and to specify a start time, a finish time and a time step of integration (Chapter 2, Leffelaar, 1998).

Suppose we want to simulate from time equals 0 (START TIME) to time equals 40 (FINISH TIME) with a time step of integration of 0.5 (DELTA) using the Runge-Kutta method (DRIVER='RKDRIV'). Since the Runge-Kutta method determines its own time step, DELT=0.5 is the first guess.

To complete the model, we only have to specify the output. If we would like to study the simulated values of Y and RY, we should place them on a PRINT statement and we should add the desired time interval between different outputs (PRDEL) to the TIMER statement (e.g. 5). All this is shown in Listing A5.1.

Listing A5.1. FST program demonstrating the simulation of logistic growth.

---

```

TITLE Logistic growth of yeast
INITIAL
  INCON      IY = 1.
  PARAMETER RGR = 0.2;  YMAX = 20.
  TIMER  STTIME = 0.;  FINTIM = 40.;  DELT = 0.5;  PRDEL = 5.
  PRINT Y, RY
  TRANSLATION_GENERAL DRIVER='RKDRIV'

DYNAMIC
  Y = INTGRL (IY,RY)
  RY = RGR*Y*(1.-Y/YMAX)
END

```

---

An important feature of FST is that the statements do not have to be in a computational order as would have been necessary with an ordinary programming language such as Fortran. An advantage of this is that it allows the user to define the model statements in a conceptual order. FST takes care of sorting the statements in a computational order while checking the integrity of the model.

Execution of the model presented in Listing A5.1 gives the following output:

```

*-----
* Output table number : 0 (=first output table)
* Output table format : Table output
* Simulation results
* Logistic growth of yeast

```

TIME	Y	RY
0.000000	1.0000	0.19000
5.00000	2.5032	0.43798
10.0000	5.6001	0.80641
15.0000	10.278	0.99923
20.0000	14.837	0.76605
25.0000	17.730	0.40244
30.0000	19.100	0.17182
35.0000	19.659	6.69605E-02
40.0000	19.873	2.51711E-02

```

*-----

```

Summarizing the above process, in FST models the following types of statements can at least be recognized:

- 1 Statements that specify the mathematical model:
  - the state variables,
  - the initial conditions (=start values of the state variables),
  - the differential equations specifying the rates of change of state variables.
- 2 Statements specifying how to solve the model:
  - the integration algorithm,
  - the start and finish time of the simulation.
- 3 Statements specifying the output of the model:
  - the variables to be output,
  - the time interval between different outputs.

#### A5.4 Comment lines and FST statements

Basically two types of program lines can be distinguished in FST, FST statements and comment lines. *Comment lines* are useful for the programmer to explain in words e.g. what is calculated in a particular place in the program. Comment lines should always start with an asterisk (\*) in the first column and they are usually written above the statements to which they belong. FST statements can start in any column between 1 and 72. FST statements can be divided across several lines by putting three dots at the end of each line that is continued on the next line (the last dot may appear in column 72). An example is given in Listing A5.2.

Listing A5.2. Example showing the syntax of comment lines and statements.

---

```
* The differential equation for logistic growth      <- comment line
  Y = INTGRL (IY,RY)                               <- normal statement
  RY = RGR*Y*(1.-Y/YMAX)                           <- normal statement
PARAMETER RGR = 0.2;...                             <- statement to be continued
          YMAX = 20.                                <- rest of continued statement
```

---

#### A5.5 Rules for FST keywords, variable names and values

In FST statements, several types of words can be distinguished such as FST keywords, variables, and values. Keywords are words that have a special meaning to FST such as `PRINT`, `TIMER`, `PARAMETER`. Such keywords are reserved names. Variables are the words that the user defines to represent the problem, such as the state variables, the rates of change, model parameters, etc. There are some restrictions in FST on the spelling of keywords and variable names. All keywords and variable names should be in uppercase. The variable names are made up of letters and digits up to a total length of six characters and the first character must be a letter. They can be chosen freely as long as no reserved keywords are used. Also some reserved FST variables exist (e.g. `TIME`). In case of using reserved variable names, FST will give an error message. The format of the values that are specified

within an FST program can be either a floating-point format such as: 0.198 or 27.45 or a scientific format such as 2.342E-12.

### A5.6 Definition of input values of the model

(PARAMETER, INCON, CONSTANT, FUNCTION)

As is shown in the example of Listing A5.1, values of model parameters are specified with a `PARAMETER` keyword, and initial values of the state variables are specified with the `INCON` keyword. (Initial values of state variables can also be defined in a calculation as is shown later.) Two more keywords exist for specifying values relevant to the model. The `CONSTANT` keyword can be used for physical or mathematical constants (e.g. the ratio of the circumference to the diameter of a circle, `PI`). Often the need exists to have a parameter which varies in time or which depends on some other model variable. This can be achieved by the `FUNCTION` keyword. After the `FUNCTION` keyword a user-specified variable name follows to which values of  $(x,y)$  coordinates can be assigned. This variable name can be used for linear interpolation by the `AFGEN` function (Arbitrary Function GENERator). The variable name that is used as independent variable ( $x$  in the pair  $x,y$ ) at which the function is interpolated is specified in the `AFGEN` function (see Listing A5.3). After a `FUNCTION` keyword only one variable name can be defined whereas after the `PARAMETER`, `INCON` and `CONSTANT` keywords several variable names can be defined.

Listing A5.3. Illustration of the syntax of the `PARAMETER`, `INCON`, `CONSTANT` and `FUNCTION` keyword.

---

```
PARAMETER A=5.; B=1.E10; C=0.51E-3
INCON A=0.; H=10.
CONSTANT PI=3.141592
FUNCTION TMTB=0.,2000., 1.,3000.,...   <- table of four (x,y) pairs
      2.,2000., 3.,1000.
Y = AFGEN (TMTB,TIME)                 <- Linear interpolation of table TMTB,
      e.g.: if TIME=1.5, Y will get the value 2500.
```

---

### A5.7 Hierarchy of operations in expressions, and the use of FST functions and Fortran functions

In FST expressions, the same hierarchical rules apply as in Fortran expressions. Expressions are evaluated in order of hierarchy of the operators, and then from left to right if the operators have the same hierarchy, except for exponentiation, which is evaluated from right to left (the expression `A**B**C` is evaluated as `A**(B**C)`). Parentheses can be used to overrule the hierarchy of the operators (Table A5.1).

In expressions, next to the above-mentioned arithmetic operators, functions can be called to perform specific tasks. As FST is translated to Fortran, many of the Fortran intrinsic functions can be used in FST. Also some special FST functions

exist that can be used in expressions. The use of several of such functions is presented in Listing A5.4.

Each function has its own number of arguments which must all be present. The only exception to this rule being the functions for taking the minimum or maximum of a set of variables: `MIN` and `MAX`. An argument may be a value or a variable name. A list of FST and Fortran functions is given in Table A5.2 at the end of this Appendix.

Table A5.1. The order of hierarchy of arithmetic operators.

Operator	Function	Order of hierarchy
( )	Grouping of operations	First
**	Exponentiation	Second
*	Multiplication	Third
/	Division	Third
+	Addition	Fourth
-	Subtraction	Fourth
=	Assignment	Fifth

Listing A5.4. Use of FST and Fortran intrinsic functions.

```

* FST intrinsic functions
A = LIMIT(B, C, A)      <- A is limited between the values of B and C
C = INSW(D, C1, C2)    <- C is C1 if D < 0 else C2
* Fortran intrinsic function
A = MAX(10., B)        <- A is assigned the largest of 10 or B
F = G*SIN(H)*COS(I)   <- F is assigned result of g*sin(h)*cos(i)

```

### A5.8 FST keywords for output (`TITLE`, `PRINT`)

In FST, several possibilities are present to get output of the model. The first one is the `TITLE` keyword. Any text that is written behind the `TITLE` keyword is written to the output file just above the output table (if any). Variables behind the `PRINT` keyword are written in a tabular format to the output file. The first eight variables are printed together in the first block, the second eight in the second block etc. Different `PRINT` keywords generate different tables. This can be useful if e.g. states should be printed in separate tables.

### A5.9 FST run control keywords (`TRANSLATION_GENERAL`, `TRANSLATION_FSE`, `TIMER`, `FINISH`)

A model should be provided with information about the integration method, start time, finish time, time step of integration and time interval for model output. The choice of the integration method is firstly determined by the type of translation chosen. The Fortran Simulation Translator can generate two different Fortran

routines. A model routine that can run under a general simulation driver when the keyword `TRANSLATION_GENERAL` is used, or an FSE compatible model routine, when the keyword `TRANSLATION_FSE` is used (FSE is a standard method of crop simulation in Fortran, developed at AB-DLO and TPE-WAU in Wageningen, see van Kraalingen (1995)). When `TRANSLATION_FSE` is used the solution scheme uses the fixed time step integration method of Euler. With `TRANSLATION_GENERAL` the solution scheme has to be specified explicitly behind the keyword with `DRIVER='EUDRIV'` for the fixed time step integration method of Euler or `DRIVER='RKDRIV'` for a variable time step integration method of Runge-Kutta. The information about time has to be specified behind the `TIMER` keyword with assignments to variable `STTIME` for start time, to `FINTIM` for finish time, to `DELT` for time step of integration and to `PRDEL` for the time interval between outputs (Listing A5.1). When a Runge-Kutta solution scheme is chosen the value of the time step (`DELT`) is used as a first guess at the start of the simulation. The time step finally taken is controlled by an error criterion (see Leffelaar, 1993 or 1998).

When `TRANSLATION_GENERAL` is chosen, the variable `TRACE` can be specified behind the `TIMER` keyword. Its value determines the amount of information that is reported in a log file about the integration process. The default value of `TRACE` is zero, a value of four gives a detailed report.

In some models, the termination of the run should not take place when `TIME` reaches `FINTIM`, but should occur when for instance a state or rate variable exceeds a certain criterion. This can be achieved by using the `FINISH` keyword. Behind this keyword two expressions are compared by an '<' or an '>' sign. When this expression is true, the simulation run is finished (e.g. `FINISH A>10.` or `FINISH A*B<C/E`). Several finish conditions can be used. The `FINISH` keyword is usually not very effective to avoid division by zero's etc. because the finish expressions are evaluated after the other expressions of the model are evaluated.

#### **A5.10 Weather data in FST programs (WEATHER)**

Weather data will often be used in FST programs. Although weather data could be defined with large `FUNCTION` tables, this would be cumbersome. Instead the `WEATHER` keyword was developed through which weather data from the Wageningen-AB/TPE weather data library (see Listing A5.5) are available to the FST model. The library is invoked by the line:

```
WEATHER  WTRDIR='directory';  CNTR='country code';  ISTN=number; ...
          IYEAR=year
```

Here, the `WEATHER` keyword assignments refer to the directory of the weather data files (`WTRDIR='directory'`), the country code (`CNTR='country code'`), the station number within the country (`ISTN=number`) and the year to which the weather data belong (`IYEAR=year`), respectively. The country code, the station number and the year are combined into the file name of the weather data. The call for the weather data file `NLD1.985` present on the directory `C:\SYS\WEATHER\`, with data from Wageningen (station no 1) 1985, The Netherlands, is specified in FST as:

```
WEATHER WTRDIR='C:\SYS\WEATHER\'; CNTR='NLD'; ISTN=1; IYEAR=1985
```

Provided the data file(s) are present and complete, the following reserved variable names can be used within the FST model (note that the radiation is already multiplied by 1000 to get  $\text{J m}^{-2} \text{d}^{-1}$ ):

Variable name	Meaning	Unit
RDD	Total daily global radiation	$\text{J m}^{-2} \text{d}^{-1}$
TMMN	Minimum air temperature	°Celsius
TMMX	Maximum air temperature	°Celsius
VP	Vapour pressure at 9 a.m.	kPa
WN	Average wind speed	$\text{m s}^{-1}$
RAIN	Total daily rainfall	$\text{mm d}^{-1}$
DOY	Daynumber of year (=time)	d
LAT	Latitude of the site	degrees
YEAR	Yearenumber	y

If the WEATHER system reports that one or more of the weather variables as missing, the run is terminated only when these variables are actually used in the model. For a full description of the weather data file format etc. see van Kraalingen *et al.* (1991).

#### A.5.11 Rerun facility, the END keyword

Often the study of the effect of changing a parameter or initial condition is wanted. This requires the modification of one or more lines of the program and run the program again from the beginning. In FST, any variable behind a PARAMETER, INCON, FUNCTION, TIMER, DRIVER or WEATHER keyword can be given a new value on lines below an END keyword. A new run is then carried out automatically with only those specifications changed. A rerun section should be closed by an END keyword. New equations may not be introduced in rerun sections. So changes in the model structure should be implemented in the model itself.

#### A.5.12 Fortran subroutines with FST, the STOP keyword

Another powerful aspect of FST is that Fortran subroutines can be called from within the INITIAL, DYNAMIC and TERMINAL sections using a CALL keyword. These routines can be part of the FST program itself but they can also be linked from an object library. Fortran subroutines can be written below a STOP keyword with the usual Fortran syntax (column 7-72). The STOP keyword itself appears below the lines where reruns can occur. The STOP statement marks an important point in the FST program because from there only a syntax check of the SUBROUTINE statement is carried out by the Fortran Simulation Translator. This is because lines below a STOP statement have a much more complex Fortran syntax. During the translation, Fortran statements are simply appended to the generated model routine.

Listing A5.5. Example of the contents of a weather data file (NLD1.985).

```

*-----*
* Station name: Wageningen (Haarweg), Netherlands
* Year: 1985
* Author: Peter Uithol -99.000: NIL VALUE
* Source: Natuur- en Weerkunde via Nel van Keulen
* Longitude: 5 40 E, latitude: 51 58 N, altitude: 7 m.
*
* Column Daily value
* 1 station number
* 2 year
* 3 day
* 4 radiation (kJ m-2 d-1)
* 5 minimum temperature (degrees Celsius)
* 6 maximum temperature (degrees Celsius)
* 7 early morning vapour pressure (kPa)
* 8 mean wind speed (height: 2 m) (m s-1)
* 9 precipitation (mm d-1)
*-----*
5.67 51.97 7. 0.00 0.00
1 1985 1 660. 0.2 5.7 0.670 5.4 6.8
1 1985 2 2200. -2.9 0.7 0.490 2.2 0.1
1 1985 3 2280. -5.5 0.2 0.520 1.6 0.0
1 1985 4 3310. -18.4 -5.2 0.230 0.7 0.4
1 1985 5 4220. -19.5 -7.8 0.210 0.6 0.0
1 1985 6 1940. -13.8 -6.2 0.310 3.9 1.7
1 1985 7 4040. -21.4 -9.7 0.200 3.3 0.0
1 1985 8 2710. -21.3 -6.2 0.240 2.2 0.0
1 1985 9 930. -7.8 -3.8 0.380 4.1 0.1
1 1985 10 1540. -11.6 -3.6 0.390 2.1 0.6
1 1985 11 2380. -5.3 -3.5 0.430 2.0 0.0
1 1985 12 2310. -7.9 1.7 0.470 3.3 1.6
.
.
1 1985 362 940. -6.2 -3.0 0.430 1.1 0.0
1 1985 363 2000. -3.7 -1.2 0.460 1.7 0.0
1 1985 364 900. -5.3 -2.4 0.450 2.9 0.0
1 1985 365 3410. -6.2 -3.0 0.660 5.4 0.0

```

The rules for a `CALL` from an FST program to a subroutine are the same as for a call from a Fortran program, i.e. the number, type and sequence of the arguments must be identical both in the `CALL` and in the `SUBROUTINE` statement. In FST, as we have seen, statements from `INITIAL`, `DYNAMIC` and `TERMINAL` sections are automatically sorted. A statement with a `CALL`, however, does not provide sufficient information for the sorting process because it is not clear which arguments are input and which are output. To resolve this lack of information for the sorting process, on top of the FST programs the input/output relations of the `CALL`'s that are used in the program must be defined. This is done with the `DEFINE_CALL` keyword. Suppose we

want to use a Subroutine `SUB1` with four arguments, the first three being input, the last being output:

```
SUBROUTINE SUB1 (X1,X2,X3,X4)
X4 = X1*X2*(1.-X2/X3)
RETURN
END
```

The `CALL` from the FST program can be:

```
CALL SUB1 (RGR,Y,YMAX,RY)
```

The input/output relations of the subroutine are defined on top of the FST program by:

```
DEFINE_CALL SUB1 (INPUT,INPUT,INPUT,OUTPUT)
```

When a routine comes from a link library, only the Fortran subroutine itself should be omitted, the `CALL` and the `DEFINE_CALL` remain necessary. For more details see Rappoldt & van Kraalingen (1996).

An example program of the discussed features of FST is given in Listing A5.6.

### A5.13 Array variables in FST

Array variables are helpful when calculations have a similar structure, and need to be done for a number of states, e.g. a number of soil layers or a number of (competing) crop species.

State and rate variables, parameters and initial constants can all be used as array variables. Arrays, contrary to scalars, have to be declared before they can be used. Each array declaration sets the lower and upper bound of the array's subscript range, and has to be defined on top of the FST program. The upper bound can be variable and can be defined by the array size variable (integer number), which is defined by the `ARRAY_SIZE` keyword.

Using the example of logistic growth of yeasts (Listing A5.1), but now for two yeast species that interfere through the production of the same waste product alcohol (ALC) (`ALCP` is alcohol production rate; `ALPF` is alcohol production factor; `RED` is reduction factor):

```
TITLE Mixed culture of yeast
ARRAY Y(1:N) , IY(1:N) , RY(1:N) , RGR(1:N)
ARRAY ALCP(1:N) , ALPF(1:N) , RED(1:N)
ARRAY_SIZE N=2
```

The parameter arrays and `incon` arrays can be defined in a series of sub-statements:

```
PARAM RGR(1) = 0.21; RGR(2:N) = 0.06
PARAM ALPF(1) = 0.12; ALPF(2:N) = 0.26
INCON IY(1:N) = 0.45
```

Listing A5.6. Example program demonstrating the use of subroutines, reruns, and `TERMINAL`.

---

<pre> DEFINE_CALL SUB1 (INPUT, INPUT, INPUT, OUTPUT) TITLE Logistic growth of yeast INITIAL   IY = SQRT (4.)-1.    PARAMETER RGR=0.2; YMAX=20.   TIMER STTIME=0.; FINTIM=40.; DELT=0.5; PRDEL=5.   PRINT Y, Y2    PRINT RY    TRANSLATION_GENERAL DRIVER='RKDRIV'  DYNAMIC   Y = INTGRL (IY, RY)   CALL SUB1 (RGR, Y, YMAX, RY)  TERMINAL    Y2 = YMAX / (1. + ((YMAX-IY) / IY) * EXP (-RGR*TIME))  END   PARAMETER YMAX=30.; RGR=0.25 END   PARAMETER YMAX=40.; RGR=0.30 END STOP    SUBROUTINE SUB1 (X1, X2, X3, X4)   IMPLICIT REAL (A-Z)   X4 = X1*X2*(1.-X2/X3)   RETURN   END </pre>	<pre> &lt;- Define input/output of subroutine &lt;- Title for output file &lt;- Start initial section &lt;- Calculation of value for initial constant  &lt;- Time information &lt;- Create an output table with Y and Y2  &lt;- Create an output table with RY  &lt;- Generate general Fortran routine, use Runge Kutta &lt;- Start dynamic section  &lt;- Call to subroutine  &lt;- Start section executed at end of each run &lt;- Calculation final value analytical solution &lt;- Start of rerun section &lt;- Definition rerun 1  &lt;- Definition rerun 2  &lt;- Start section FOR- TRAN subroutine(s) &lt;- Start of a Fortran subroutine </pre>
--	--

---

Since the initial amount of the two yeasts is the same, the `INCON` array `IY` could also be defined as:

```
INCON IY = 0.45
```

because the `INCON` statement contains the ‘full-range’ definition of array `IY`. For clarity, however, it is recommended to use explicitly the array indications.

The initial condition of `ALC` is set at 0., and the maximum at 1.5; these variables will be read as scalar variables (they are not defined as an array, because of the assumption that the same alcohol is produced by the two yeast species). When 99% of the maximum alcohol concentration is reached (`LALC`) the simulation will stop (see `FINISH` condition):

```

INCON IALC = 0.
PARAM MALC = 1.5
LALC = 0.99 * MALC
TIMER STTIME=0.; FINTIM=150.; DELT=0.5; PRDEL=2.
TRANSLATION_GENERAL DRIVER = 'EUDRIV'
PRINT Y(1:N), ALC

```

In the PRINT statement the full-range of Y will be printed.

The first four assignments in the DYNAMIC section (below) can be programmed in the two ways showed, left with explicit array indications and right without:

<pre> DYNAMIC Y(1:N) = INTGRL(IY(1:N),RY(1:N)) RY(1:N) = RGR(1:N)*Y(1:N)*(1.-RED(1:N)) ALC = INTGRL(IALC, RALCP) ALCP(1:N) = ALPF(1:N)*RY(1:N) </pre>	<pre> DYNAMIC Y = INTGRL(IY,RY) RY = RGR*Y*(1.-RED) ALC = INTGRL(IALC,RALCP) ALCP= ALPF*RY </pre>
---	---

The right side equations look like ordinary statements from an FST program without arrays. The *N* initial amounts of IY and the *N* rates of change RY belong to the *N* amounts of Y, and the INTGRL statement looks the same as for ordinary non-array variables. Since the translator knows that the variables are arrays, there is no need for describing the expression for each yeast, separately. The concise form e.g. ALCP=ALPF\*RY will be expanded by the translator. This way of writing can be done because the full range of the arrays is used. In other cases, array indications should explicitly be given. It is nevertheless recommended to always use the left side notation.

For the summation of the array elements of RALCP (the two rates of alcohol production) the FST intrinsic function ARSUMM can be used (see Table A5.2 for more information on FST intrinsic functions).

```
RALCP = ARSUMM(ALCP,1,N)
```

The assignment for the reduction factors (RED) is more complicated, because data have to be read from different tables. However, all elements belonging to one array must be calculated in one statement separated with semi-colons (;). Here, the *two* reduction factors are calculated in *two* sub-statements separated by a semi-colon. The first sub-statement defines the reduction for yeast species 1, and the second sub-statement defines the reduction for yeast species 2 to *N*. To maintain the overview, each new element can best start at a new line, so continuation dots are needed (...).

```

RED(1) = AFGEN(RED1T,ALC/MALC); ...
RED(2:N) = AFGEN(RED2T,ALC/MALC)

```

The simulation stops when ALC gets greater than LALC:

```

FINISH ALC>LALC
END

```

A full account of the biological aspects of the yeast-alcohol problem is given by de Wit and Goudriaan (see Leffelaar, 1993 or 1998).

#### A5.14 Fortran subroutines and FST array variables

Array variables in Fortran subroutines follow the same rules as scalar variables with respect to the number, type and sequence of the arguments, which should be identical. In the case of array variables, however, the arguments must be specified as array arguments.

In the `DEFINE_CALL` statement, the array arguments can be `INPUT_ARRAY` or `OUTPUT_ARRAY`. Also the array size can be an argument in the `DEFINE_CALL`, i.e. `INTEGER_INPUT`. In this way, no changes need to be made in the subroutine when e.g. the number of species would be taken as 3.

Adapting the program of Listing A5.6 for two yeasts species gives:

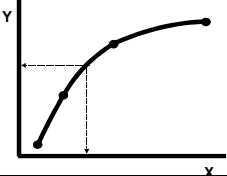
```
TITLE Mixed culture of yeasts
DEFINE_CALL SUB1(INTEGER_INPUT, INPUT_ARRAY, INPUT_ARRAY, ...
                INPUT_ARRAY, OUTPUT_ARRAY)
ARRAY Y(1:N)    , IY(1:N)  , RY(1:N)  , RGR(1:N)
ARRAY ALC(1:N), ALPF(1:N), RED(1:N)
ARRAY_SIZE N=2
INITIAL
  PARAM RGR(1) = 0.21; RGR(2:N) = 0.06
  PARAM ALPF(1) = 0.12; ALPF(2:N) = 0.26
  INCON IY(1:N) = 0.45
  INCON IALC = 0.
  PARAM MALC = 1.5
  FUNCTION RED1T = 0.,0., 1.,1.
  FUNCTION RED2T = 0.,0., 1.,1.
      LALC = 0.99 * MALC
  TIMER STTIME=0.; FINTIM=150.; DELT=0.5; PRDEL=2.
  TRANSLATION_GENERAL DRIVER='EUDRIV'
  PRINT Y(1:N), ALC
DYNAMIC
  Y(1:N) = INTGRL(IY(1:N), RY(1:N))
  CALL SUB1(N,RGR,Y,RED,RY)
  ALC = INTGRL(IALC, RALCP)
  RALCP = ARSUMM(ALCP,1,N)
  ALCP(1:N) = ALPF(1:N)*RY(1:N)
  RED(1) = AFGEN(RED1T, ALC/MALC); ...
  RED(2:N) = AFGEN(RED2T, ALC/MALC)
  FINISH ALC>LALC
END
STOP
SUBROUTINE SUB1(N,X1,X2,X3,R)
  IMPLICIT REAL(A-Z)
  INTEGER N,I
  REAL X1(N), X2(N), X3(N), R(N)
  SAVE
  DO 10 I=1,N
    R(I) = X1(I)*X2(I)*(1.-X3(I))
10 CONTINUE
  RETURN
END
```

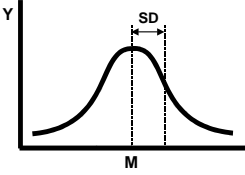
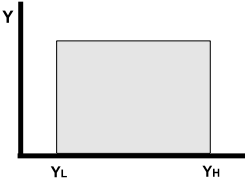
### A5.15 References

- IBM Corporation, 1975. Continuous Simulation Modeling Program III (CSMP III). Program reference manual SH 19-7001-3. Data Processing Division, 1133 Westchester Avenue, White Plains, New York, 206 pp.
- Kraalingen, D.W.G. van, 1995. The FSE system for crop simulation. Version 2.1. Quantitative Approaches in System Analysis No 1, DLO-Research Institute for Agrobiological and Soil Fertility and C.T. de Wit Graduate School for Production Ecology, Wageningen, 27 pp.
- Kraalingen, D.W.G. van, W. Stol, P.W.J. Uithol & M.G.M. Verbeek, 1991. User Manual of CABO/TPE Weather System. CABO/TPE internal communication, Centre for Agrobiological Research and Department of Theoretical Production Ecology, Wageningen, 27 pp.
- Leffelaar, P.A. (Ed.), 1993. On Systems Analysis and Simulation of Ecological Processes. Current Issues in Production Ecology, Volume 1, Kluwer Academic Publishers, Dordrecht, 294 pp.
- Leffelaar, P.A. (Ed.), 1998. On Systems Analysis and Simulation of Ecological Processes. Second Edition. Current Issues in Production Ecology, Volume 4, Kluwer Academic Publishers, Dordrecht, 318 pp.
- Rappoldt, C. & D.W.G. van Kraalingen, 1996. The Fortran Simulation Translator FST version 2.0. Introduction and Reference Manual. Quantitative Approaches in Systems Analysis No. 5, June 1996. DLO-Research Institute for Agrobiological and Soil Fertility and C.T. de Wit Graduate School for Production Ecology, Wageningen, The Netherlands, 178 pp.

Table A5.2. List of FST intrinsic functions. The input arguments K and L denote an integer constant, variable or expression. Array arguments are written as A or B. The symbol F means an FST interpolation function and all other input arguments are real constants, variables or expressions. From Rappoldt & van Kraalingen (1996).

FST function	Mathematical notation or Graph
<p>Y = AFGEN (F, X)</p> <p>Linear interpolation between (x,y) function points.</p> <p>Y - Result of interpolation, estimated F(X)</p> <p>F - Table of (x,y) values specified with FUNCTION statement</p> <p>X - Value of independent variable</p>	
<p>Y = ARIMPR (A, B, K, L)</p> <p>Returns the improduct of a vector A and a vector B calculated over the subscript range K,...,L.</p> <p>Y - Returned improduct</p> <p>A - Array variable seen as vector</p> <p>B - Array variable seen as vector</p> <p>K - Start of subscript range</p> <p>L - End of subscript range with L≥K</p>	$y = \sum_{i=K}^L A_i B_i$
<p>Y = ARLENG (A, K, L)</p> <p>Returns the length of the vector with elements A(K),..., A(L) in (L-K+1)-dimensional space.</p> <p>Y - Returned length</p> <p>A - Array variable seen as vector</p> <p>K - Start of subscript range</p> <p>L - End of subscript range with L≥K</p>	$y = \sqrt{\sum_{i=K}^L A_i^2}$
<p>Y = ARMAXI (A, K, L)</p> <p>Returns the maximum value among the array elements A(K),..., A(L).</p> <p>Y - Resulting number</p> <p>A - Array variable</p> <p>K - Start of subscript range</p> <p>L - End of subscript range with L≥K</p>	$y = \max_{i=K}^L A_i$
<p>Y = ARMEAN (A, K, L)</p> <p>Returns the arithmetic mean of the array elements A(K),..., A(L).</p> <p>Y - Maximum value</p> <p>A - Array variable</p> <p>K - Start of subscript range</p> <p>L - End of subscript range with L≥K</p>	$y = \frac{\sum_{i=K}^L A_i}{L - K + 1}$
<p>Y = ARMINI (A, K, L)</p> <p>Returns the minimum value among the array elements A(K),..., A(L).</p> <p>Y - Resulting number</p> <p>A - Array variable</p> <p>K - Start of subscript range</p> <p>L - End of subscript range with L≥K</p>	$y = \min_{i=K}^L A_i$

<p>Y = ARSMPS (A, K, L, D)  Simpson's integral of a function over L-K closed intervals <math>(x_K, x_{K+1}), (x_{K+1}, x_{K+2}), \dots, (x_{L-1}, x_L)</math>. At the L-K+1 points the function takes the values A(K),..., A(L). All intervals have equal width D.</p> <p>Y - Approximate integral  A - Array containing function values  K - Start of subscript range  L - End of subscript range with L&gt;K  D - Interval width</p>	$y = D \sum_{i=K}^L w_i A_i$ <p>in with the coefficients <math>w_i</math> follow</p> <ul style="list-style-type: none"> <li>- trapezoidal rule for</li> <li>- extended <math>1/n^3</math> rule for</li> <li>- alternative extended Simpson's rule for <math>n \geq 8</math></li> </ul>
<p>Y = ARSTDV (A, K, L)  Returns the standard deviation of the array elements A(K),..., A(L) seen as a sample.</p> <p>Y - Returned standard deviation  A - Array variable  K - Start of subscript range  L - End of subscript range with L&gt;K</p>	$y = \sqrt{\frac{\sum_{i=K}^L (A_i - \bar{A})^2}{L - K}}$
<p>Y = ARSUMM (A, K, L)  Returns the sum of the array elements A(K),..., A(L).</p> <p>Y - Resulting sum  A - Array variable  K - Start of subscript range  L - End of subscript range with L≥K</p>	$y = \sum_{i=K}^L A_i$
<p>Y = CSPLIN (F, X)  Natural cubic splines interpolation between (x,y) function points according to Press <i>et al.</i> (1989).</p> <p>Y - Result of interpolation, estimated F(X)  F - Table of (x,y) values specified with FUNCTION statement  X - Value of independent variable</p>	
<p>Y = ELEMNT (A, K)  Returns value of the K-th element of array A after verifying its existence by comparing K with the declared array bounds.</p> <p>Y - Returned element value  A - Array variable  K - Subscript</p>	$y = A_K$
<p>Y = FCNSW (X, Y1, Y2, Y3)  Input switch. Y is set equal to Y1, Y2 or Y3 depending on the value of X.</p> <p>Y - Returned as either Y1, Y2 or Y3  X - Control variable  Y1 - Returned value of Y if X&lt;0  Y2 - Returned value of Y if X=0  Y3 - Returned value of Y if X&gt;0</p>	$\begin{cases} y = y_1, & x < 0 \\ y = y_2, & x = 0 \\ y = y_3, & x > 0 \end{cases}$

<p>Y = INTGRL (YI, YR)  Integration command in the form of a function call. The algorithm of the numerical integration depends on the selected translation mode and driver.</p> <p>Y - State variable  YI - Initial value of Y, must be a variable  YR - Rate of change, must be a variable</p>	$y(t) = y(0) + \int_0^t \frac{dy(t)}{dt} dt$
<p>Y = INSW (X, Y1, Y2) Input switch. Y is set equal to Y1 or Y2 depending on the value of X.</p> <p>Y - Returned as either Y1 or Y2  X - Control variable  Y1 - Returned value of Y if X&lt;0  Y2 - Returned value of Y if X≥0</p>	$\begin{cases} y = y_1, & x < 0 \\ y = y_2, & x \geq 0 \end{cases}$
<p>Y = LIMIT (XL, XH, X)  Y is equal to X but limited between XL and XH</p> <p>Y - Returned as X bounded on [XL,XH]  XL - Lower bound of X  XH - Upper bound of X</p>	$\begin{cases} y = x, & x_1 \leq x \leq x_h \\ y = x_1, & x < x_1 \\ y = x_h, & x > x_h \end{cases}$
<p>Y = NOTNUL (X)  Y is equal to X but 1.0 in case of X=0.0. Note that X is evaluated without any tolerance interval.</p> <p>Y - Returned result  X - Checked for being zero</p>	$\begin{cases} y = x, & x \neq 0 \\ y = 1, & x = 0 \end{cases}$
<p>Y = REAAND (X1, X2)  Returns 1.0 if both input values are positive, otherwise Y=0.0.</p>	$\begin{cases} y = 1, & x_1, x_2 > 0 \\ y = 0, & x_1 \leq 0 \text{ or } x_2 \leq 0 \end{cases}$
<p>Y = REANOR (X1, X2)  Returns 1.0 if both input values are less than or equal to zero, otherwise Y=0.0.</p>	$\begin{cases} y = 1, & x_1, x_2 \leq 0 \\ y = 0, & x_1 > 0 \text{ or } x_2 > 0 \end{cases}$
<p>Y = RGNORM (M, SD)  Random number Generator which returns numbers with an univariate normal distribution.</p> <p>Y - Returned random number  M - Mean of the normal distribution  SD- Standard deviation of the distribution</p>	
<p>Y = RGUNIF (YL, YH)  Random number Generator which returns numbers with a uniform distribution on (YL, YH).</p> <p>Y - Returned random number  YL - Lower bound of interval  YH - Upper bound of interval</p>	

List of Fortran intrinsic functions with explanation. Taken from Rappoldt & van Kraalingen (1996).

Fortran function	Explanation	Mathematical notation	Restrictions
ABS (X)	absolute value of $x$	$ x $	
INT (X)	the integer part of $x$ , result is integer	$\text{int}(x)$	
AINT (X)	the integer part of $x$ , converted to real	$\text{int}(x)$	
NINT (X)	the nearest integer, result is integer	$\text{int}(x)$	
ANINT (X)	the nearest integer converted to real	$\text{int}(x)$	
MAX (X1, X2, . . . , Xn)	maximum value among the real arguments	$\max(x_1, x_2, \dots, x_n)$	$n \geq 2$
MIN (X1, X2, . . . , Xn)	minimum value among the real arguments	$\min(x_1, x_2, \dots, x_n)$	$n \geq 2$
MOD (I, J)	remainder of $i/j$ with sign of $i$ , result is integer	$i \bmod j$	$j \neq 0$
AMOD (X, Y)	remainder of $x/y$ with sign of $x$ , result is real	$x \bmod y$	$y \neq 0$
COS (X)	cosine of $x$ , $x$ in radians	$\cos(x)$	
COSH (X)	hyperbolic cosine of $x$	$\cosh(x)$	
ACOS (X)	arccosine of $x$ in range $[0, \pi]$	$\arccos(x)$	$-1 \leq x \leq 1$
EXP (X)	exponential function	$e^x$	
LOG (X)	natural logarithm of $x$	${}^e \log x$	$x > 0$
ALOG (X)	natural logarithm of $x$	${}^e \log x$	$x > 0$
LOG10 (X)	base 10 logarithm of $x$	${}^{10} \log x$	$x > 0$
ALOG10 (X)	base 10 logarithm of $x$	${}^{10} \log x$	$x > 0$
REAL (I)	the real number nearest to integer $I$		
SQRT (X)	square root of $x$	$\sqrt{x}$	$x \geq 0$
SIN (X)	sine of $x$ , $x$ in radians	$\sin(x)$	
SINH (X)	hyperbolic sine of $x$	$\sinh(x)$	
ASIN (X)	arc sin of $x$ in range $[-\pi/2, \pi/2]$	$\arcsin(x)$	$-1 \leq x \leq 1$
TAN (X)	tangent of $x$ , $x$ in radians	$\tan(x)$	$x \bmod \pi/2 \neq \pi/4$
TANH (X)	hyperbolic tangent of $x$	$\tanh(x)$	
ATAN (X)	arc tangent of $x$ in range $[-\pi/2, \pi/2]$	$\arctan(x)$	
ATAN2 (X, Y)	arc tangent of $x/y$ in range $[-\pi/2, \pi/2]$	$\arctan(x/y)$	