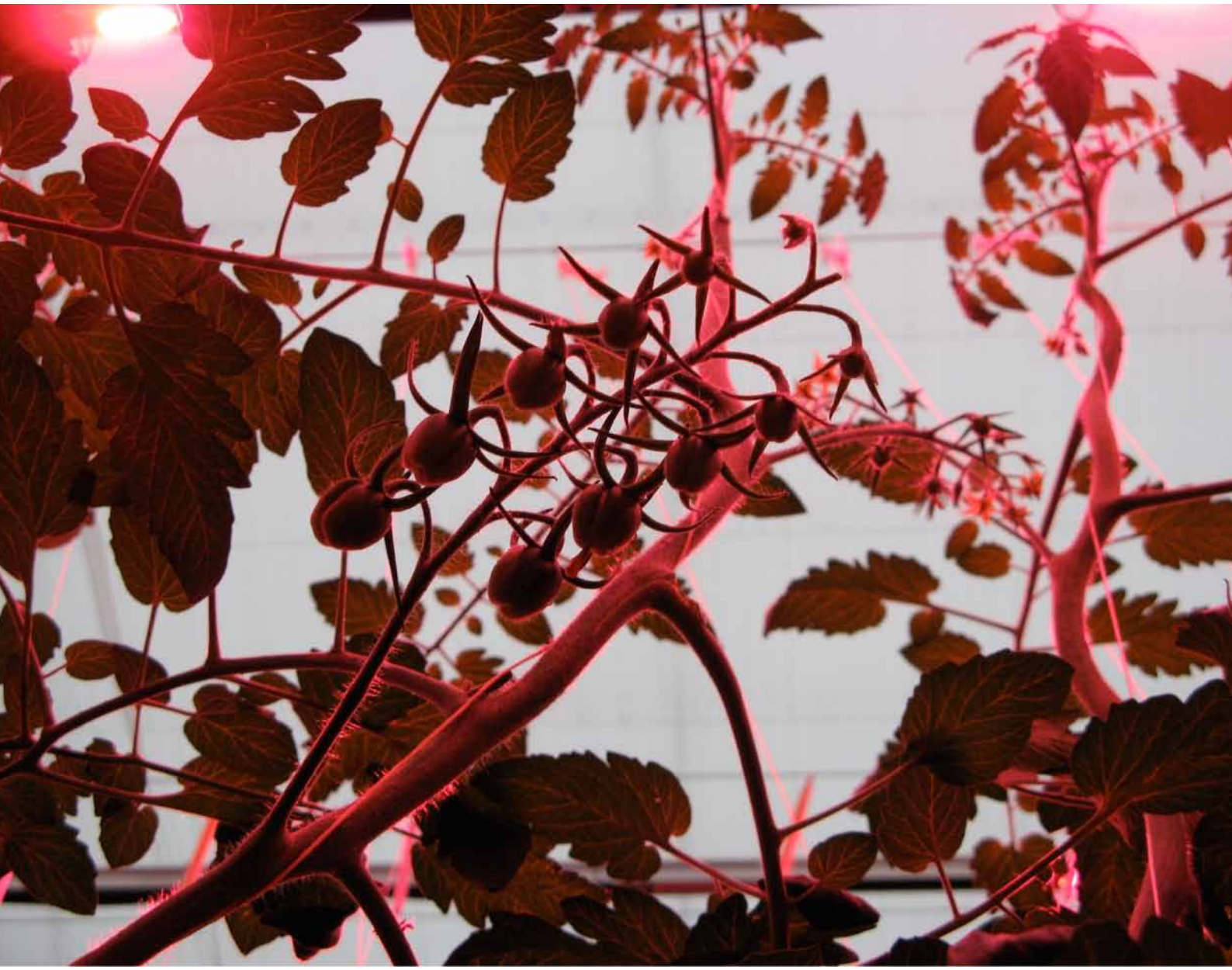
The Fortran Simulation Translator 4.16

plotting, sensitivity analysis, events and measured variables

C. Rappoldt, D.W.G. van Kraalingen

EcoCurves



December 2021

EcoCurves rapport 32, ISSN 1872-5449

This document has been created in December 2021

The most recent version of this manual is available from
www.ecocurves.nl/Support/FST/FSTadditions.pdf

Part of the FSTwin installation is the FST translator which is provided by EcoCurves BV "as is" and without warranties. EcoCurves BV cannot accept any responsibility for errors leading to incorrect simulation results.

FSTwin is compatible with the GFortran compiler and with Ghostscript for viewing plots. The use of these third party programs requires that you comply with the terms and conditions for obtaining a valid license. This is solely your responsibility and the use of FST does not alter this in any way.

The FSTwin installation is accompanied by two separate installers, which can optionally be used to perform a standard installation of GFortran and/or Ghostscript. These are programs under the GNU General Public License, however, and you must comply with the terms and conditions of this license if you use GFortran and/or Ghostscript in combination with FST. This also applies if you use an existing installation of these programs or if you download newer versions.

The Fortran Simulation Translator 4.16

The Fortran Simulation Translator 4.16

plotting, sensitivity analysis, events and measured variables

C. Rappoldt[1], D.W.G. van Kraalingen[2]

[1]EcoCurves BV, Kamperfoelieweg 17, 9753 ER Haren, Nederland
E-mail: kees.rappoldt@ecocurves.nl

[2]Wageningen Environmental Research, P.O. Box 47, 6700 AA Wageningen, Nederland
E-mail: daniel.vankraalingen@wur.nl

EcoCurves rapport 32

EcoCurves BV, Haren, 2021

REFERAAT

C. Rappoldt, D.W.G. van Kraalingen, 2021. *The Fortran Simulation Translator 4.16 ; plotting, sensitivity analysis, events and measured variables.* EcoCurves rapport 32, EcoCurves BV, Haren. 104 blz.

This updated manual describes the changes and additions made in FST 3 and FST 4, the plotting features, measured data input, the sensitivity statement and events, all introduced by means of example programs. In this latest edition for the FST translator 4.16 there is are new chapters for the sensitivity statement and for the plotting style. Furthermore the manual begins with a chapter "For those who dislike manuals", where many of the added features and briefly explained by means of a single example program.

Keywords: model, simulation, language, plotting, event, crop growth

This document has been created in December 2021

The most recent version of this manual is available from

www.ecocurves.nl/Support/FST/FSTadditions.pdf

Kamperfoelieweg 17, 9753 ER Haren (gn), Nederland
Tel.: (050) 5370392; e–mail: kees.rappoldt@ecocurves.nl

Voorplaat: "Greenhouse"

This document has been created using the LaTeX typesetting system.

[EcoCurves BV December 2021]

Contents

List of Figures

Example programs

List of Tables

Preface

This manual describes the changes and additions made in FST 3 and FST 4, the plotting features, measured data input, the sensitivity statement and events, all introduced by means of example programs. In this latest edition there are new chapters for the sensitivity statement and for the plotting style. Furthermore the manual begins with a short chapter "For those who dislike manuals", in which many of the added features and briefly explained by means of a single example program.

The FST translator has been updated to version 4.16, a 64 bits application with minor changes, bug fixes and a few increased capacity settings. The updated and now fully 64 bits graphical user interface FSTwin (by Daniel van Kraalingen) supports GFortran for running the translated model and Ghostscript for viewing results. This update has been supported financially by the Plant Science group of Wageningen University.

The Fortran code generated by the translator is also compatible with the latest Intel Fortran compiler (part of the free oneAPI HPC package). For this compiler and for working under OSX or Linux there is no graphical user interface available, however. On request, the translator application, the object libraries required and a batch file for running FST models can be obtained from the authors.

Additions to FST have emerged from practical needs during the development and application of large models by EcoCurves, Plant-Dynamics and Photosyntax. Examples are time and state events, plotting statements and access to measured data. A very efficient addition is the sensitivity statement which allows an automated sensitivity analysis on a model parameter.

We thank Ad Schapendonk, Peter Leffelaar and XinYou Yin for their continuous enthusiasm.

Haren / Wageningen, February 2013, December 2021
Kees Rappoldt, Daniel van Kraalingen

Chapter 1

Introduction

FST is an easily learned simulation language which is useful in especially the development of crop growth models. Important models like SUCROS (Goudriaan & van Laar, 1994), Lintul, Gecros (Yin & van Laar, 2005) and ORYZA2000, the IRRI[1] rice model (Bouman *et al.*, 2001) have been written in FST. These models are used over the world by many agronomists, not all of them having affinity with the underlying numerics.

The FST translator translates a completely specified simulation model into a Fortran-90 program with datafiles. These datafiles contain the values for the model parameters and the Fortran program contains an input section for reading the model parameters from file.

1.1 Fortran compilers, Ghostscript

The FST translator[2] translates a model into Fortran-90 source code with data files. Actually running a model requires also a Fortran compiler. The translator produces standard Fortran which can be used in combination with any Fortran compilers on any platform. For the combination of Windows and GFortran compiler there is a graphical user interface FSTwin available. For OSX and Unix in combination with the (now free) Intel Fortran compiler, the required files are available on request (kees.rappoldt@ecocurves.nl).

The GFortran compiler is part of the "Minimalist GNU for Windows" or MinGW compiler set (see `http://www.mingw.org/wiki/Getting_Started`). The most recent version of this compiler can be found at `https://sourceforge.net/projects/mingw-w64/files/`.

Ghostscript (e.g. `https://github.com/ArtifexSoftware/ghostpdl-downloads/releases/download/gs952/gs952w64.exe`) converts postscript and EPS files into bitmapped formats or PDF. It is used by the FSTwin to convert the EPS files produced by the model into bitmaps in order to visualise the graphs on screen.

[1] International Rice Research Institute.
[2] File FST.exe in the installed FSTwin directory

1.2 Legal issues

Part of the FSTwin installation is the FST translator which is provided by EcoCurves BV "as is" and without warranties. EcoCurves BV cannot accept any responsibility for errors leading to incorrect simulation results.

FSTwin is compatible with GFortran and with Ghostscript for viewing plots. The use of these third party programs requires that you comply with the terms and conditions for obtaining a valid license. This is solely your responsibility and the use of FST does not alter this in any way.

The FSTwin installation is accompanied by two separate installers, which can optionally be used to perform a standard installation of GFortran and/or Ghostscript. These are programs under the GNU General Public License, however, and you must comply with the terms and conditions of this license if you use GFortran and/or Ghostscript in combination with FST. This also applies if you use an existing installation of these programs or if you download newer versions.

1.3 This manual

Chapter 2 for those who dislike manuals is an efficient introduction to the newer elements of FST. It contains a single example model with an event, plotting statements and a sensitivity analysis.

Chapter 3 describes sensitivity runs. With just a single statement a series of simulation runs is made with a varying parameter value.

Chapter 4 contains a description of the time and state events. A time event is a change in the system which takes place at a specified moment in time. Something is added to the system, for instance, a process is activated or the value of a parameter suddenly changes. A state event allows the same kind of sudden changes but it does not happen at a preset moment in time, but whenever a certain condition is met. Harvest takes place, for instance, when a crop reaches a certain stage.

Chapter 5 describes the FST plotting statements. These statements produce publication quality plots, simply by running the model (see for details section 5.6.1 at page 66). Chapter 8 describes access to external (measured) data. These are the features added to version 4 of FST.

Chapter 7 contains a description of the calendar connection in the GENERAL translation mode of FST. Some other changes are documented in Chapter 9. In appendices some reference material has been collected, including a brief version history with bug fixes.

This manual[3] explains the additions made in FST 3 and FST 4. A more complete description of the structure of an FST model and of all basic FST statements can be found in the original manual Rappoldt & van Kraalingen (1996), which is available as www.ecocurves.nl/Support/FST/FSTmanual(1996).pdf.

[3] A recent version of this manual is available from www.ecocurves.nl/Support/FST/FSTadditions.pdf.

Chapter 2

For those who dislike manuals

The example model at the next two pages contains a sensitivity statement, an event section and plotting statements. In just a single example you find out about these additions to FST, each of them covered more fully in a subsequent chapter.

The model describes a mass going up and down at the end of a spring. The oscillator is called harmonic since the force of the spring is proportional to the deviation from equilibrium position. There is friction, which makes it a damped oscillator, and a periodic force is applied, which makes it a forced oscillator and not a free one.

2.1 An example model in 39 statements

Statement 1 is the TITLE of the model. There are no subroutines to be declared in DEFINE_CALL statements, so in statement 2 the actual Model begins.

2.1.1 Parameters and simulated time

Statements 3 to 7 define the parameters mass, spring constant, friction constant and the period and amplitude of the force applied. Note that the comment text behind each statement describes the parameter and its unit.

Statement 8 means that the movement is simulated over 100 s. Statement 9 specifies the integration method, fourth-th order Runge-Kutta with error control with an initial time step of 0.01 s, defined by Delt in statement 8.

2.1.2 Sensitivity analysis

The sensitivity statement 10 means that 100 model runs have to be made for equally spaced values of the parameter PERIOD between 1.0 and 20.0.

2.1.3 Constants and initial calculations

Statements 11 and 12 define constants used further down. Statement 14 in the INITIAL section calculates the angular frequency of the applied force.

2.2 Text of example model

Here is the example model containing several additions to the original FST. For more background on the model itself see https://en.wikipedia.org/wiki/Harmonic_oscillator#Simple_harmonic_oscillator or https://en.wikipedia.org/wiki/Resonance.

```
0001   TITLE Forced Damped Harmonic Oscillator
       ! A harmonic oscillator is brought into movement by an external sinusoidal force with period PERIOD. The
       ! oscillator is damped by means of a friction force propertional to velocity. If the external period
       ! is close to the oscillator's natural period, the amplitude reaches large values. This is called resonance.
       !
       ! The amplitude of the oscillation is calculated by simulating the movement and "measuring" the amplitude
       ! during the last period before FINTIM. Measuring is "switched on" by means of a time event.

0002   MODEL
       ! parameters
0003   PARAMETER   MASS =  1.0 ! [  kg  ] mass of harmonic oscillator ; ratio of force and acceleration
0004   PARAMETER      K =  1.0 ! [N m-1 ] spring constant ; ratio of elastic force and displacement X
0005   PARAMETER      B =  0.4 ! [Ns m-1] friction constant: ratio of force and velocity
0006   PARAMETER PERIOD = 80.0 ! [  s   ] period of external sinusoidal force on oscillator
0007   PARAMETER     F0 =  1.0 ! [  N   ] amplitude of driving force
       ! Continuous interation over a time period sufficiently long to reach stationary oscillations
0008   TIMER STTIME=0.0 ; FINTIM=100.0 ; DELT = 0.01
0009   TRANSLATION\_GENERAL DRIVER='RKDRIV'

       ! this generates the reruns
0010   SENSITIVITY Varying=PERIOD ; BeginRange=1.0 ; EndRange=20.0 ; NumberOfRuns=100

0011   CONSTANT PI = 3.14159265
0012   INCON  ZERO = 0.0       ! Initial value used for all state variables

0013   INITIAL
0014   OMEGA = 2.0*PI / PERIOD ! angular frequency related to period

0015   SETTING Switch = 0.0    ! initial value of "Last Period" switch
0016   EVENT Last Period
```

```
      !  at precisely one PERIOD before FINTIM this switch is turned on by means of this time event
0017     FirstTime FINTIM - PERIOD
0018     NewValue Switch = 1.0
0019  ENDEVENT

0020  DYNAMIC               ! deviation from equilibrium X position and velocity V are the state variables
0021  RX = V                ! velocity is the rate of change of X
0022  X  = INTGRL (ZERO,RX) ! position is the integral of the velocity
0023  RV = Acceleration     ! acceleration is the rate of change of V
0024  V  = INTGRL (ZERO,RV) ! velocity is the integral of acceleration

      ! Acceleration is calculated from the mass of the object and the forces acting on it.
0025  TotalForce   = AppliedForce - ElasticForce - Friction !  Sum of the various forces
0026  AppliedForce = F0 * COS(OMEGA * TIME)                 !  External force applied has prescribed PERIOD and amplitude
0027  ElasticForce = K * X                                  !  Elastic directed to the equilibrium position at X=0
0028  Friction     = B * V                                  !  Friction is proportional to velocity
0029  Acceleration = TotalForce / MASS                      !  Newton's law: Acceleration = (Total Force) / Mass

      ! Distance moved is the integral of the absolute velocity
0030  MovedDistance = INTGRL(ZERO,AbsoluteV) ! MovedDistance increases from zero as soon as AbsoluteV is positive
0031  AbsoluteV     = Switch * ABS(V)        ! which is after the "Last Period" time event at which Switch is set to 1.0

0032      TERMINAL
0033  SimulatedAmpl  = MovedDistance / 4.0   ! a quarter of MovedDistance should be close to the amplitude
0034  AnalyticalAmpl = F0 / SQRT(MASS**2 * (K/MASS-OMEGA**2)**2 + OMEGA**2 * B**2)   ! analytical solution

      ! sensitivity plot for all runs, combining the initial scalar PERIOD as X value with terminal Y values
0035  DEFINE\_AXIS Ampax = '0.0 3.1 0.5 0.1 0.0 amplitude'
0036  ASSIGN\_AXIS Ampax > AnalyticalAmpl, SimulatedAmpl
      ! FrameType=2 means sensitvity plot
0037  CURVE XVAR=PERIOD ; YVAR=AnalyticalAmpl ; Legend = 'analytical' ; Curwid=4.00 ; Frame='Resonance' ; FrameType=2
0038  CURVE XVAR=PERIOD ; YVAR=SimulatedAmpl  ; Legend = 'simulated'  ; Marsiz=0.05 ; Martyp=8  ; Marcol='red'
0039  END
```

2.2.1 Dynamic calculations

In the DYNAMIC section, statements 21 to 24 define the two state variables, the deviation X from equilibrium position and the velocity V. The acceleration is the rate of change RV of the velocity. The velocity is the rate of change RX of X.

In statements 25 to 28 the total force acting on the mass is calculated from which statement 29 finds the acceleration by Newton's second law.

This completes the actual model. The program sofar does 100 simulation runs with different values of the parameter PERIOD, but it does not yet produce any output.

2.2.2 "Measuring" the amplitude

The total distance moved during one period of a stable oscillation is 4 times the amplitude. Hence, one way to "measure" the amplitude is calculating the distance moved during a simulated period. In order to calculate this distance, in either positive or negative direction, we need to integrate the absolute velocity (statements 30 and 31). The integration is "switched on" by setting the variable Switch from 0.0 to 1.0 at precisely one PERIOD before FINTIM.

Switch is a so called SETTING variable (statement 15), which may change value at an event. Statements 16 to 19 specify a time event which takes place one PERIOD before FINTIM. At precisely this time the simulation is holded, the Switch is set to 1.0, and the simulation resumes.

In the TERMINAL section (following statement 32) we divide MovedDistance by 4.0 in order to get an estimate of the stationary amplitude (statement 33). There is also an analytical expression for it (statement 34).

2.2.3 Plotting the result

In statements 35 to 38 a plot is constructed with both the simulated and theoretical amplitude as function of PERIOD. The two CURVE statements define an actual curve for AnalyticalAmpl and a series of markers for SimulatedAmpl.

The name of the plot is "Resonance", which is defined by the frame keyword in statement 37. The second "curve" in statement 38 is without frame keyword and is therefore automatically added to the frame of statement 37. Figure 2.1 shows the plot.

2.3 Some thoughts about this model

2.3.1 About SENSITIVITY runs

The use of a Sensitivity statement prevents the use of ordinary rerun sections following the END statement. Further, run 0 defined by (in this case) the PARAMETER PERIOD statement is omitted and only the runs specified by Sensitivity are executed.

Sensitivity runs are not limited to model parameters. See Chapter 3 for details.

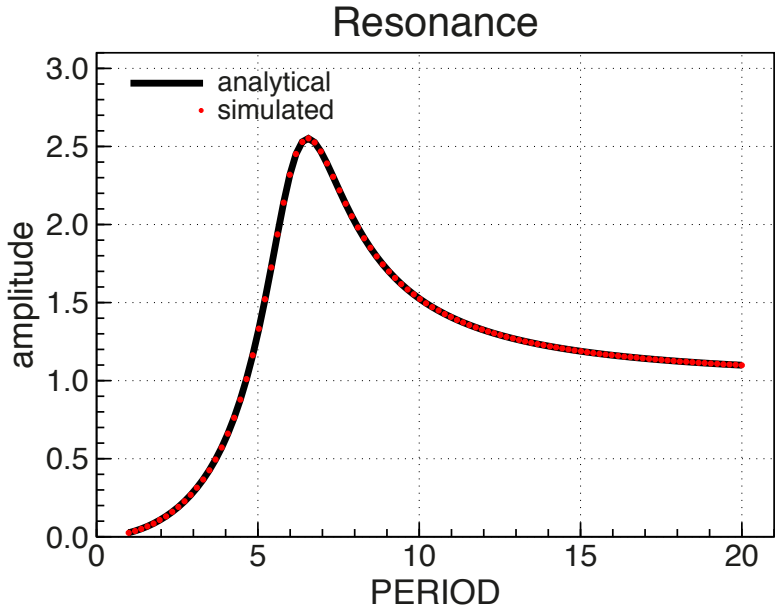


Figure 2.1. Sensitivity plot created by the example model in section 2.2 at page 16. For values of PERIOD close to the natural period of the oscillator, the amplitude may become very large, especially for low friction values.

2.3.2 About EVENT sections

In an event-endevent section state variables and settings may be change value by means of one or more NewValue statements. The event in the above oscillator model was used to switch on amplitude "measurement", i.e. to do an observation on model behaviour.

Events, however, may also be part of the simulated process. A harvest event, for instance, means that the crop status is set back to some initial value for regrowth, or the crop is removed altogether and the system waits for a seeding event, at a sufficiently large soil water content for instance.

Periodic events are useful for simulating discrete processes. For instance, if photosynthesis is simulated in continuous time, a daily growth event at midnight can be used to distribute the assimilates over the various plant parts. Such a periodic event looks like

```
SETTING Counter=1.0
EVENT
   FirstTime = Counter * OneDay
   NewValue Counter = Counter + 1.0
!  all periodic calculations
   . . .
   NextTime = Counter * OneDay
END EVENT
```

Not just time events can be inserted in the simulation, but also state events. A state event occurs if a calculated quantity crosses zero. When that happens the simulation is halted, the event takes place and the simulation continues. For instance, if the coordinate Y of an object crosses zero, we may want to reverse the velocity VY

(a state variable). In an event section we must then insert `ZeroCondition`Y and `NewValue`VY=-VY. The simulated object then bounces back from the zero plane.

Chapter 4 contains a more complete description of events in FST.

2.3.3 About "measuring" the simulated amplitude

Our measurement of the simulated amplitude is based on the assumption that the oscillation is stationary approaching FINTIM, i.e. that the oscillation repeats itself and that any significant effect of the initial upswing has disappeared.

This clearly could be verified by repeating the calculations for various FINTIM values. We could also simply make a plot of the position X. More interesting is to plot the extreme positions reached during each simulated period by making use of the fact that at the extremes the velocity crosses zero. The following state event will do the job.

```
EVENT
   ZeroCondition V
!  note that the variable Xextreme is defined at event times only
   Xextreme = X
!  plot just the position Xextreme at event times, not all positions X.
   CURVE YVAR=Xextreme ; Marsiz=0.05 ; Martyp=8  ; Marcol='red' ; ...
         Frame='Approaching stability'
ENDEVENT
```

The result is 100 plots like the one in Figure 2.2. Each plot contains both the lower and upper extremes reached during each oscillation period.

The plot demonstrates that the oscillation will approximately be stationary at times approaching 100 s. By adding "`PRINT` Xextreme" this may also be verified from the actual values in the output file RES.DAT.

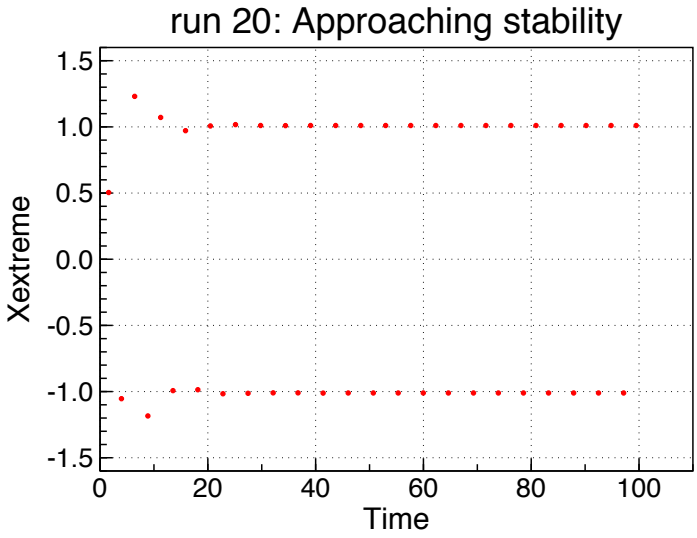


Figure 2.2. The extreme positions during the oscillation simulated by the example model in section 2.2 at page 16 with a state event added. The effect of the initial upswing quickly disappears and the oscillation becomes stationary.

2.3.4 About the CURVE statements

The idea of the plot in Figure 2.1 is that the variables AnalyticalAmpl and SimulatedAmpl are both plotted relative to the same vertical axis. This has been realized by defining an axis and assigning it to both variables. Without an ASSIGN_AXIS statement, a separate axis would be drawn for each of the variables.

Finally a remark is made on the FrameType keyword in statement 37 in section 2.2. Normally a pair of values (X,Y) is plotted only if both values were sent to output at the same simulated TIME. For instance, an X value calculated in the INITIAL and a Y value calculated in the TERMINAL section will never appear together as an (X,Y) pair in a plot, with one exception.

By setting Frametype=2, the frame becomes a sensitivity plot and a single plot is made with results obtained in multiple runs. Then, if the XVAR variable and the YVAR variable of a CURVE statement occur just once in the output for each run, the pairs (X,Y) are plotted in a single frame. This also holds for array variables which may lead to a series a curves, one for each run. These curves could be stationary concentration profiles for instance.

Occurring just once usually means calculated in the INITIAL or TERMINAL section of the model. But also variables calculated during a unique event occur once and may be plotted in a sensitivity plot as function of a parameter value or so.

2.3.5 About the model itself: Resonance

Without friction and without external force, a harmonic oscillator with spring constant k and mass m has a "natural period" T_0 given by

$$T_0 = 2\pi\sqrt{\frac{m}{k}} \quad .$$

which means for the angular frequency ω_0

$$\omega_0 = \frac{2\pi}{T_0} = \sqrt{\frac{k}{m}} \quad .$$

With friction and an external periodic force with period T, the amplitude of the oscillator reaches a maximum for T slightly above the natural period T_0 of the oscillator.

At https://en.wikipedia.org/wiki/Harmonic_oscillator#Simple_harmonic_oscillator and https://en.wikipedia.org/wiki/Resonance more can be found on harmonic oscillation and resonance. The maximum amplitude gets higher for less friction. Also large elastic structures may resonate with an external force. This can be dangerous.

Chapter 3

Sensitivity runs

Sensitivity runs are made to explore the effect of a model parameter on a results of the model. This requires a number of model runs from which then results are combined in a single plot.

3.1 The sensitivity statement

The Sensitivity statement is meant to automate the creation of reruns in case you just want to vary the value of an input value, usually a model parameter. In a sensitivity statement you provide the parameter name, the value range and the number of runs. FST then creates a series of runs, just as if a (long) list of rerun sections would have been specified. A Sensitivity statement cannot be combined with ordinary rerun sections.

Examples of sensitivity statements can be found in Listing 4.1 at page 29 and in section 2.2 at page 16. A valid Sensitivity statement requires the following keywords:

Varying or **IntVarying** The name of a real or integer variable for which sensitivity runs should be made. The variables allowed are listed below.

BeginRange The value used in the first model run. If the sensitivity variable is integer (keyword IntVarying is used), the start value is nint(BeginRange). Hence, the nearest integer is used.

EndRange The value used in the last model run. If the sensitivity variable is integer (keyword IntVarying is used), the end value is nint(EndRange).

NumberOfRuns The number of runs made. The parameter values are equally spaced over [BeginRange,EndRange]. For an integer variable this keyword is optional. If it is absent all integers between nint(BeginRange) and nint(EndRange) are used.

LogRange Optional variable. For LogRange >= 1, the values used are equally spaced on a logarithmic axis. A logarithmic spacing requires a positive range. In case of an integer variable, selected with keyword IntVarying, the logarithmic spacing wil be approximate.

Note that the execution of "run 0" is suppressed by a sensitivity statement[1].

[1] In case of ordinary reruns specified "by hand" in reruns sections at the end of the FST model, the first run executed is "run 0" which is defined by all "standard values" in the actual FST model. After that the runs defined by the rerun sections are executed.

The variables which may occur as the **Varying** or **IntVarying** variable are

- Any scalar model parameter defined with a PARAMETER statement.
- The Timer variables STTIME, FINTIM, DELT, RGSEED.
- The Translation_General variables DELMAX, EPS, SEVTOL, StartYear, StartDOY.
- The Weather control variable IYEAR.
- The Measurements variables CycleStYear, CycleStMonth, CycleStDay, CyclePeriodInDays for cyclic use of measured data.

For integer variables (RGSEED, StartYear, IYEAR, CycleStYear, CycleStMonth, CycleStDay and CyclePeriodInDays) the keyword IntVarying has to be used.

If you want to vary the initial value of a state variable, say Xini, you have to define Xini=P, define P as a model parameter and then make sensitivity runs on P.

3.2 Sensitivity plots

A sensitivity plot combines results from all sensitivity runs in a single plot. A plot becomes a sensitivity plot by adding "FrameType=2" to a CURVE statement[2].

Often the varying sensitivity variable, say P, has to be plotted as the X variable (define CURVE Xvar=P ; . . .) and there is just a single value for each run. The Y variable may be any result, as long as it is defined only once, in an initial, terminal or event section.

The resulting plot then really shows the way in which Y depends on the varying variable. Examples of such sensitivity plots can be found as Figure 2.1 at page 19, and as Figurefig:Stone at page 30.

A more detailed description of sensitivity plots is given in section 5.2.7 starting at page 46. Eamples are given there of combining entire curves in a single plot, illustrating for instance the stability of an equilibrium (X,Y) position, starting at various initial positions (Figure 5.9 at page 47).

[2]The default "FrameType=1" defines an ordinary plot, with a separate frame for each run.

Chapter 4

Time and state events

Events interrupt the normal simulation cycle of rate calculations and status updates. The simulation is interrupted in order to change something in parameter values or even the system status. After the event, the simulation continues, until possibly another event takes place.

This chapter describes how events can be initiated and what sort of things can be done during an event. There are two types of event. A time event simply takes place at a prescribed moment in simulated time. When the event time is reached the instructions belonging to the event are executed, a new event time may be set, and the simulation continues.

State events are more complicated. A state event takes place when a certain condition is reached, for instance if the state variable A reaches the value 5.0, a state event must happen. In FST, this takes the form of a zero condition equal to $A-5.0$. If this expression becomes (almost) zero, the event takes place. Then, after the "event function" has moved away from zero and becomes zero once more, the event takes place again.

Below a detailed description is given for both event types. At first, however, a new type of variable needs to be introduced, the setting variable.

4.1 What for are Setting variables?

During events the value of a state variable may be changed. This represents a sudden, instantaneous change which cannot be described in an ordinary rate of change approach.

In practice we also want to apply sudden changes to parameter like variables or control variables. A constant background temperature for instance, which acts as a driving variable and suddenly changes. Or a control variable which is used to switch between measured and calculated data. The problem with ordinary initial variables is that they are sent to output only once and can never change value anymore.

We therefore need a variable type which can be calculated initially, which can be changed during events, and which is sent to dynamic "PRDEL output". This is what a setting variable is meant for.

4.2 Defining Setting variables

A setting variable is defined by means of a SET statement in the following way

```
! example of the use of a setting variable
  DECLARATIONS
  ...
  INITIAL
  SET CumulativeAmount = 0.0
  SET NitrogenContent = 20.0 * MAX(Ncon, 2.0)
  ...
```

The setting variables `CumulativeAmount` and `NitrogenContent` are defined by means of a SET statement. The second example shows that the SET statement is a calculation. It is *not* a value assignment like PARAMETER, INCON or CONSTANT statements, but expressions can be used as if the setting variable was an ordinary calculated variable. Also array expressions can be used if the setting variable is declared as an array before. In fact, any initially calculated variable can be made into a setting variable by just putting the keyword SETTING (or just SET) in front of the calculation[1].

The following rules apply to setting variables (the keyword SETTING may be used instead of SET).

- The SET or SETTING statement may occur only in the INITIAL section. The defining expression may refer to PARAMETERs, CONSTANTs, driver supplied variables, or other initially calculated variables.
- A setting variable can be defined and used as any other initially calculated variable. This implies that the calculations in SET statements are sorted (put into computable order) together with the other initial calculations.

There are, however, three differences between a setting variable and an ordinary, initially calculated variable.

1. If a setting variable is listed in a PRINT or CURVE statement, it is sent to dynamic "PRDEL output", together with the dynamic output variables.
2. A setting variable may change value during an event, like state variables.
3. A setting variable may act as a rate of change in an INTGRL statement.

In a model without events, setting variables are just initially calculated variables, behaving like dynamic variables with respect to output, but with no other special function.

4.3 Events step by step

4.3.1 Time event

An event section contains everything which has to be specified about an event, when it takes place and what should happen. The simplest event section is

[1] With the exception of variables calculated in a subroutine call. If such a variable, say A, needs to become a setting variable, a help variable (say Help) is first calculated in the subroutine and then assigned to the setting variable by SET A = HELP.

```
! example of and event section
  DYNAMIC
  ...
  EVENT
     FIRSTTIME StTime + 2.0
  ENDEVENT
```

This initiates a time event at 2.0 time units after start time. During the event, however, nothing happens, it just interrupts the simulation and no new event time is specified. A periodic time event can be initiated by

```
! example of and event section
  DYNAMIC
  PARAMETER Period = 4.0
  ...
  EVENT Periodic
     FIRSTTIME StTime + 2.0
     NEXTTIME Time + Period
  ENDEVENT
```

This initiates a series of time events, beginning at 2.0 time units after start time and returning every Period time units thereafter. Still, however, the event does not change anything in the system status. Note that the event is named "Periodic", a name which appears in the logfile and in messages.

In the next example this is different. Each time event changes the setting variable named `SetPoint` and resets a state variable StateA to zero.

```
! example of event section
  INCON Aini = 1.2345
  INITIAL
  SET SetPoint = 10.0
  ...
  DYNAMIC
  RateA = ...
  StateA = INTGRL(Aini, RateA)
  ...
  EVENT
     FIRSTTIME StTime + 2.0
     PARAMETER Period = 4.0
     NEXTTIME Time + Period
     NEWVALUE StateA = 0.0
     NEWVALUE SetPoint = -SetPoint
  ENDEVENT
```

The first NEWVALUE statement redefines the state variable StateA at zero. SetPoint is redefined as the opposite of the old SetPoint. Hence, the variable SetPoint is initially +10.0 and then, beginning at 2.0 time units after starttime, it switches periodically between +10.0 to −10.0.

4.3.2 State event

State events do not contain a FIRSTTIME and NEXTTIME statement but instead contain a ZEROCONDITION. The ZEROCONDITION statement contains an expression which triggers the event when it crosses zero (from positive to negative or vice versa). This conditions is also called the "event function".

Suppose we want to change the SetPoint variable from the previous example each time the integral StateA reaches the value TOP. This is done with

```
! example of and event section
  INCON Aini = 1.2345
  INITIAL
  SET SetPoint = 10.0
  ...
  DYNAMIC
  RateA = ...
  StateA = INTGRL(Aini, RateA)
  ...
  EVENT
     ZEROCONDITION StateA - Top
     PARAMETER Top = 200.0
     NEWVALUE StateA = 0.0
     NEWVALUE SetPoint = -SetPoint
  ENDEVENT
```

Like in the previous example, the state StateA is the integral of RateA over time, beginning at Aini. But now, if StateA ever reaches 200.0, the event takes place. The setpoint changes into its opposite value and StateA is reset.

In Translation_General mode, a zero-crossing of the event function (the expression in the ZeroCondition statement) is followed by an iterative search to the precise time at which the event function reaches zero (see section 4.6.1). In FSE mode, step size is fixed and the state event takes place immediately after detecting the zero-crossing of the event function (see section 4.6.2).

Note that events not necessarily take place. In the last example, if StateA never reaches the value Top, the event will never happen.

4.4 Example model with Sensitivity and two Events

Events are used in the first place to simulate sudden changes in the state variables or in the environment of the simulated process. In a crop simulation, for instance, there are many soil state or crop state dependent actions, like seeding, irrigation, pesticide application or harvest.

A somewhat different use of events is "monitoring model behavior". The event does not actually interfere with the process but is used to observe when or under which circumstances a specific "event" (literally) takes place.

As an example Listing 4.1 contains a complete model of the parabolic path of a thrown away stone. The model does not just simulate the parabola, but also observes the maximum height reached and the time at which the stone reaches the ground again. Upon reaching the ground the simulation is halted.

The maximum height and the time at which this height is reached are "measured" as function of the initial vertical velocity with help of the Sensitivity statement. This statement instructs FST to do 50 runs with different values of parameter InitialVY. The observed dependence is shown in a sensitivity plot (FrameType=2) containing results from the 50 runs. We summarize:

- The model simulates the parabolic path of a stone, thrown away with a certain initial velocity. Air resistance is not accounted for but could be added.
- The maximum height reached is "measured" by the model by means of a state event taking place when the vertical velocity goes from positive to negative.
- The relation between the initial vertical velocity and the height reached is investigated numerically by means of an automated sensitivity analysis.

Listing 4.1 The parabolic path of a stone without air resistance.

```
Title Stone Parabola
PARAMETER Gravity   =  9.8 ! [m/s^2] acceleration of gravity at earth surface
PARAMETER InitialVX =  2.0 ! [ m/s ] initial horizontal velocity
PARAMETER InitialVY = 10.0 ! [ m/s ] initial upward velocity
! plot of the simulated path
Define_Axis Haxis = '0.0 100.0  20.0 10.0 0.0 X (m)'
Define_Axis Vaxis = '0.0 500.0 100.0 50.0 0.0 Y (m)'
Assign_Axis Haxis > X  ! this assigns the axis Haxis to the variable X
Assign_Axis Vaxis > Y
CURVE XVAR=X; YVAR=Y; CURCOL='Red'; Legend='(x(t),y(t))'; Frame='Path'
TIMER PRDEL = 0.2 ! [ s ] periodic output at intervals PRDEL
INITIAL
  IX  = 0.0       ! [ m ] initial position
  IY  = 0.0       ! [ m ]
  IVX = InitialVX ! [m/s] initial velocity
  IVY = InitialVY ! [m/s]
! simulation control ; note the large FINTIM
  TIMER  STTIME=0.0 ; FINTIM=10000.0 ; DELT=0.1
  TRANSLATION_GENERAL TRACE=1 ; DRIVER='RKDRIV'

! The simulation is halted as soon as the stone is back on the ground. This is
! realized with a Finish condition triggered by the "Impact of Stone" state event.
  Setting HaltFlag = 0.0     ! flag to be set at impact
  EVENT Impact of Stone      ! event name appears in logfile if TRACE > 0
!   event does not take place at initial since there is no zero-crossing then
    ZeroCondition Y          ! time of zero height is found iteratively
    NewValue HaltFlag = 1.0  ! raise the flag !!
    Finish HaltFlag > 0.5    ! Finish could be placed outside the event section
  ENDEVENT
DYNAMIC
  VX = VelocityX  ! the rate of change of the position (m) is the velocity (m/s)
  VY = VelocityY
  X  = INTGRL (IX, VX)
  Y  = INTGRL (IY, VY)

! the rate of change of the velocity (m/s) is the acceleration (m/s^2)
  AX = 0.0       ! constant horizontal velocity
  AY = -Gravity  ! gravity
  VelocityX = INTGRL(IVX, AX)
  VelocityY = INTGRL(IVY, AY)

! maximum height is reached when the vertical velocity crosses zero
  EVENT Top Reached           ! event name appears in logfile if TRACE > 0
    ZeroCondition VelocityY   ! time of zero vertical velocity found iteratively
    TopX = X                  ! store position at event time
    TopY = Y                  !
    TimeAtTop = Time          ! store time of event
  ENDEVENT
TERMINAL
  TimeOfImpact = Time  ! time of impact (should be two times TimeAtTop)
  PRINT TimeAtTop, TimeOfImpact
! this sensitivity statement explores the role of the initial vertical velocity:
  SENSITIVITY Varying=InitialVY; BeginRange=10.0; EndRange=100.0; NumberOfRuns=50

! a sensitivity plot (FrameType=2) combines the results for all runs
  Define_Axis VYaxis  = 'initial vertical speed (m/s)' ! automatic axis range
  Assign_Axis VYaxis  > InitialVY
  Define_Axis TTaxis  = 'time after start (s)'
  Assign_Axis TTaxis  > TimeAtTop
  Define_Axis TOPaxis = 'reached height (m)'
  Assign_Axis TOPaxis > TopY
! this plots the top time and top height as function of initial vertical velocity
  CURVE Frame='Top'; FrameType=2; ...
        XVAR=InitialVY;YVAR=TimeAtTop;MARTYP=9;MARCOL='red' ;Legend='time to top'
  CURVE XVAR=InitialVY;YVAR=TopY     ;MARTYP=9;MARCOL='blue';Legend='height'
END
```

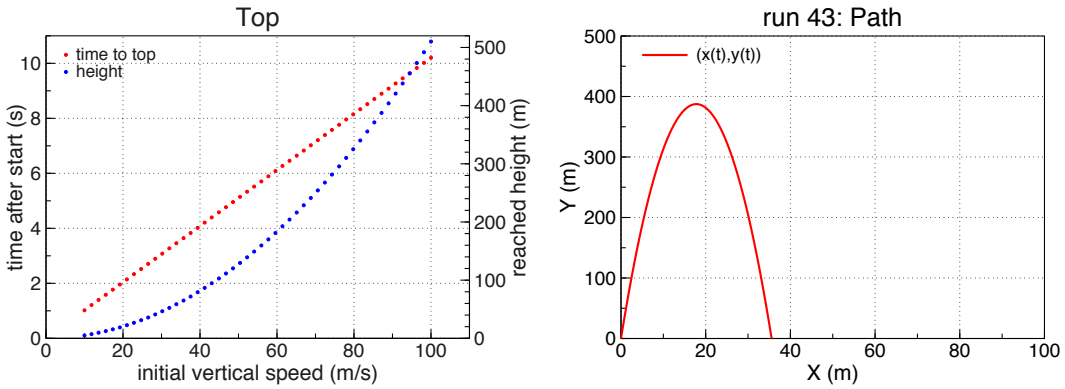


Figure 4.1. The sensitivity plot "Top" created with the stone model in Listing 4.1. For each of the 50 runs separate points have been drawn (see MARTYP in the Curve statements). The right hand plot is the path frame for run 43.

- The simulation is halted when the stone is back on the ground. An "Impact of Stone" state event detects this and is triggers a finish condition.
- This "Impact of Stone" state event could be used also for reversing the vertical velocity. In that case the object would bounce back and begin a new parabola. If you try that, remove the Finish condition and the HaltFlag and add to the event section VelocityY = −VelocityY.
- All simulated paths and the result of the sensitivity analysis are visualized in high quality output plots.

Figure 4.1 contains the sensitivity plot with datapoints from 50 runs and also one of the simulated paths.

Each curve in the sensitivity plot consists of the (X, Y) points for 50 runs. This works only if both X and Y occur once and only once in model output, which is the case here since X is an initial variable (parameter InitialVY) and the Y variables are the event variables TopY and TimeAtTop defined in an event that takes place only once during a model run (see also section 5.3.4).

4.5 Event sections: the rules

Many rules are about the references that can be made in event sections to other variables of the model. A FIRSTTIME statement, for instance, should not refer to dynamic variables, since the first time event time must be calculated during the initial phase of the simulation.

The rules of the game:

1. An event section begins with an EVENT statement and it ends with an ENDEVENT statement.

2. The word EVENT may be followed by the name of the event. The name is an arbitrary string with at most 31 characters. It may include spaces and there are no quotes around it. This name appears in the logfile and in some messages. The name is not a variable and cannot be referred to in any way.

3. An FST program may contain several event sections (actually about 50).

4. An event section is contained in the INITIAL or in the DYNAMIC section of the model. Its position in the FST model is not significant. It does not have any consequences for the way in which the dynamic calculations are sorted and written to the generated Fortran code. Hence, an event section may be put close to the initial or dynamic statements to which it is naturally related. Event sections may also be grouped at the beginning or end of the DYNAMIC section. This is a matter of style and taste.

5. However, when during simulation two or more events occur simultaneously, the order of execution depends on the order of the event sections in the FST model. Details on this can be found in section 4.7.

6. A time event section must contain one and only one FIRSTTIME statement.

7. A FIRSTTIME statement contains a constant or *expression* specifying the first event time as function of initially known variables (PARAMETERs, CONSTANTs), driver supplied variables and initially calculated variables including setting variables.

8. A time event section may further contain a single NEXTTIME statement.

9. A NEXTTIME statement specifies the next time event time as a constant or *expression*. The expression may refer to initially known variables, initially calculated variables, dynamically calculated variables, driver supplied variables and to variables calculated in the same event section where the NEXTTIME statement is in.

10. A state event section must contain one and only one ZEROCONDITION statement.

11. A ZEROCONDITION statement contains a scalar expression which may refer to initially known variables, driver supplied variables and all initially and dynamically calculated variables, including state and setting variables.

12. An event section may contain one or more calculation statements, defining variables which are not defined elsewhere in the model. The expressions, subroutine calls and function calls used may refer to initially known variables, driver-supplied variables and initially or dynamically calculated variables, *and to other calculated variables defined in other calculation statements in the same event section.* Just like initial, dynamic and terminal calculations, the calculations in each event section are sorted by the FST translator.

13. An event section may contain one or more NEWVALUE statements.

14. Each NEWVALUE statement redefines a state variable or a setting variable, which may be either a scalar or array variable.

15. A NEWVALUE definition of a *scalar* (non-array) state or setting may refer to itself (the old value of the state or setting).

16. The NEWVALUE definition of a *array* state or setting cannot refer to itself. Such a calculation requires a help variable, an array with the same length, which is calculated in the event section as a copy of the (old value of) state ot setting array to be changed.

17. A NEWVALUE definition *must not refer to any other state or setting variable which is redefined in the same event section.*

18. NEWVALUE statement may refer to all *other* initially known or dynamically calculated variables, to initially calculated variables, driver supplied variables, or to calculated variables defined in calculation statements in the same event section.

19. Just like the INITIAL, DYNAMIC or TERMINAL section of an FST program an EVENT section may contain statements like PARAMETER, CONSTANT, TIMER. The function of such statements does not depend on their position anywhere between INITIAL and END. These statements do not interfere with the functionality of the event section.

Rule number 17 probably requires some clarification. The reason for this rule is that, by allowing such references, the order in which the NEWVALUE instructions are executed would make a difference. The use of "old values" can always be realized by calculating a help variable as a copy of a state variable or setting, and then using the calculated help variable in a NEWVALUE expression.

The order in which the operations specified are carried out is as follows:

1. The sorted calculation statements are executed.
2. Output variables (mentioned in Print or Curve) just calculated in this event section are sent to output.
3. The NEWVALUE assignments are executed. Their order is arbitrary (see the rules above).
4. In case of a time event, the NEXTTIME is calculated. This implies that state and setting variables possibly occurring in the NEXTTIME expression refer to *new* values for those states and settings which were just redefined.

This order does not depend on the order of the statements in the event section.

4.6 Reaching a state event

Sofar, the way in which the ZEROCONDITION expression is treated has remained a bit vague. The reason is that this depends on the type of simulation carried out.

4.6.1 General mode

The event function is monitored during the simulation and as soon as it crosses zero (from either side) the time at which the zero crossing occurs is found iteratively by means of a number of bisection steps.

There clearly is some tolerance involved here. This is the value of SEVTOL (State Event Tolerance), which may be specified in a TRANSLATION_GENERAL statement, but which has a default value of 1.0×10^{-5}. As soon as the event function is within SEVTOL from zero, the event is triggered[2].

If the TRANSLATION_GENERAL control variable TRACE is set to 4, the iterative search is reported to the logfile. Listing 4.2 contains such a report from the stone model in Listing 4.1.

The first lines show a few regular integration and output steps. Then the "Top Reached" event function (see Listing 4.1) seems to cross zero. In 17 iteration steps

[2]SEVTOL may be referenced in expressions.

Listing 4.2 Part of the logfile written by the model in Listing 4.1 if the TRACE variable is set at 4.

```
   +  0.20000     -->   7.2000 try next   0.26216
Output flag set =====   7.2000 rate call
   +  0.20000     -->   7.4000 try next   0.26216
Output flag set =====   7.4000 rate call

Step     Time     Event  Event function
----     ----     -----  --------------
  0     7.6000      2    -0.19430      ([Top Reached] state event)
  1     7.5000      2     0.78570
  2     7.5500      2     0.29570
  3     7.5750      2     5.06978E-02
  4     7.5875      2    -7.18022E-02
  5     7.5812      2    -1.05522E-02
  6     7.5781      2     2.00728E-02
  7     7.5797      2     4.76031E-03
  8     7.5805      2    -2.89594E-03
  9     7.5801      2     9.32189E-04
 10     7.5803      2    -9.81873E-04
 11     7.5802      2    -2.48419E-05
 12     7.5801      2     4.53674E-04
 13     7.5802      2     2.14416E-04
 14     7.5802      2     9.47870E-05
 15     7.5802      2     3.49725E-05
 16     7.5802      2     5.06531E-06

   +  0.18017     -->   7.5802 try next   0.26216
Output flag set =====   7.5802 rate call preparing for event
Output flag set =====   7.5802 rate call with State Event(s)
                        7.5802 [Top Reached] state event (error 5.06531E-06)
   +  1.98273E-02 -->   7.6000 try next   0.10000
Output flag set =====   7.6000 rate call
   +  0.10000     -->   7.7000 try next   0.35257
   +  0.10000     -->   7.8000 try next   0.35257
```

(straightforward bisection) the zero event function is reached with a sufficient accuracy. Then an event-preparing rate call *with output* takes place, sending dynamic variables to output at their pre-event values.

The actual event call to the model is a rate call with output enabled and with the proper event flags set. The model takes care of the event handling (including output of event-calculated output variables) and then does a regular rate calculation with dynamic output. After completion, the event is reported once more and the normal simulation cycle is resumed.

Note that model calls in the above description are calls to the Fortran-95 model which was generated by the FST translator from the FST source code in Listing 4.1. The iteration table also refers to the "Top Reached" event with number 2, simply because this event is the second one in the FST source code.

A state event cannot occur without zero *crossing* of the event function. This implies that the function must first be at least SEVTOL away from zero and then it may cross zero (again).

4.6.2 FSE mode

The FSE mode of the translator leads to simulations with a fixed time step, often set to one day for crop growth models. In fact there is no continuous time in FSE

mode but there are just discrete time steps. In this situation an adapted time step in order to reach precisely an event time would be inconsistent with the approach.

Therefore, in FSE mode, events take place as soon as the event time is reached or passed, or as soon as a zero condition is reached or passed. There is no iterative search for a precise state event time nor a calculated time step in order to precisely reach the preset time of a time event.

This approach is a bit crude. It is the only approach, however, which seems consistent with the fixed steps of the process simulation. Crop harvest, for instance, takes place at a certain day and not at 16:20:10 in the afternoon. The same holds for fertilization, weeding or other events that may occur in a simulation of crop growth with a one day time step.

4.6.3 Scaling the event function

The value of SEVTOL is an absolute tolerance. For an event function with values in the order of, say, one million it does not make sense to require an accuracy of 10^{-5}. Then the tolerance SEVTOL may be set to a larger value. Increasing the SEVTOL value, however, will also affect other state events. So, a larger value of SEVTOL is an option only in models containing a single, or a few similar state events.

A better method is to scale the event function, i.e. divide it by a constant in such a way that its values lie at a reasonable distance from zero, for instance in $[-1, +1]$. An example is an event that takes place when some coordinate X reaches the value of (parameter) A. The event function would then be $(X - A)$. If this function becomes very large it is better to write $(X - A)/A$ or to use any other system size parameter as a scaling constant, like in $(X - A)$/SystemSize.

4.6.4 Missed state events

In principle state events can be missed if, during a single time step, an event function crosses zero multiple times. Therefore the time step should be prevented to become too large, especially if the variable time step Runge-Kutta method is used. This can be done using the TRANSLATION_GENERAL variable DELMAX.

4.7 Simultaneous events

In case of a model with several event sections, two or more events may occur simultaneously, on purpose, or by chance. Even in GENERAL mode the time at which a state event takes place (found iteratively) may sometimes coincide with a time event or another state event. Here we explain how the generated Fortran program treats such a situation.

There are two ways of dealing with two simultaneous events. The first method ("Update Once") is to execute the code for both events (event calculations, change state or setting) and then recalculate once all dynamic variables in order to have new and updated rates of change. The second method is "handle event 1", "recalculate dynamic variables", "handle event 2" and "recalculate dynamic variables". This method is referred to as "Insert Updates".

If an update after event 1 does not have any consequences for the calculations and

the NEWVALUE statement(s) in event 2, the two methods lead to the same result. The FST translator, however, does not verify such an event independence and it is possible to write an FST model for which the results depend on the way in which simultaneous events are handled. This is different for the two translation modes.

In FSE mode the "Update Once" method is used for all events (state or time). Hence, there is no dynamic update in between the execution of simultaneous events. The order of events is the order of their respective event sections in the FST model.

In GENERAL mode the "Update Once" method is used as well, but for the *state events only*. If there are simultaneous state and/or time events pending, the simulation driver first handles all *state events* in a single call to the generated Fortran model. The state events are handled in the order of their respective event sections in the FST model. After execution of all pending state events, the dynamic variables are updated once[3], just like in FSE mode.

Pending *time events* in GENERAL mode, however, are handled *after all pending state events*. For *time events* the GENERAL simulation driver uses the "Insert Updates" method. Each time event is followed by an update of the dynamic variables[4] and simultaneous time events are handled in the order of the (time) event sections in the FST model.

[3]Given the SEVTOL tolerance, the event functions of the pending state events all cross zero for the current system status. A dynamic update after handling just one of the events (an inserted update) could have implications for the other event functions, for which it was just decided they cross zero. Hence, when several event functions cross zero simultaneously, the driver must assume these events are indeed simultaneous and the events take place without update of the dynamic status in between.

[4]This can be seen as inserting a zero length time step between simultaneous time events.

Chapter 5

Plotting in FST

Section 5.2 contains a brief plotting course. Examples are given in the form of complete model listings together with the resulting plots. We start with a one-statement-plot and proceed to more complex examples with axes under user control. Finally, the results of a series of model runs are combined in a single plot.

Systematic treatment of the plotting statements then begins in section 5.3. A complete list of CURVE statement variables is given and the defaults are described. Chapter 5.3.4 describes what happens (or not happens) if plotted variables are defined in initial, dynamic, terminal or event sections of the model. In section 5.4 the construction of a calendar based time axis is explained.

The plotting style can be influenced by editing the file "PlotPreferences.dat". If such a file is not there, a default is generated which may be edited in order to change things like the size of the plots, the size of the axis and legend texts, the legend position, etc. Some suggestions are given in section 6.

5.1 Why plotting in FST?

The Windows interface of FST contains a facility to plot output variables which is very useful for model development and explorative calculations. Often, however, the same plots need to be made again and again, after each model run. The plots are not saved on file and only a single plot can be seen on screen. Routine inspection of large amounts of output and "production runs" therefore require a faster plotting method.

This has been achieved by adding plotting statements to the FST language. These statements fully specify the required plots, including things like axis text and curve color. With these plotting statements the design of clear plots has become part of the model development process. After investing some time in a neat plotting section, the modeler is rewarded with a fast and clear presentation of results, created automatically after each model run.

The plots are produced in publication quality EPS files (Encapsulated Postscript files). People using the LaTeX typesetting system can import EPS plots into their documents. EPS files can also be easily converted into either postscript or PDF files by the free utilities Ghostview and Ghostscript. A vector graphics editor, e.g. Adobe Illustrator, can be used to edit and combine EPS files and add arrows, explanatory text or pictures. Under OSX on a Macintosh, EPS files are opened by the standard Preview application.

Listing 5.1 This is the FST program for exponential growth from the original FST manual Rappoldt & van Kraalingen (1996, Listing 2.1). A single, short CURVE statement has been appended to the model.

```
INITIAL                                          calculations from here are initial
DYNAMIC                                          calculations from here are dynamic
  X  = INTGRL (IX,RX)                            state, initial value and rate of change
  RX = A * X                                     calculation of rate of change
INCON IX=1.0                                     set initial value
PARAMETER A=0.1                                  a single model parameter
TRANSLATION_GENERAL DRIVER='RKDRIV'              the translation mode is selected
TIMER STTIME=1.0; FINTIM=10.0; DELT=0.1          timing the simulation
PRINT X                                          tabular output
TIMER PRDEL=0.5                                  time interval between output times
CURVE YVAR=X                                     plotting the result
```

5.2 Plotting course by example

5.2.1 A quick plot

Listing 5.1 shows the first example model from the original FST manual Rappoldt & van Kraalingen (1996). Just a single CURVE statement has been added, which produces the graph in Figure 5.1.

This CURVE statement obviously means that the variable X has to be plotted on a vertical axis ("y-axis"), which is what the program does. The time interval between the output points is PRDEL, the interval used for tabular output as well.

Since only a variable name was specified in the CURVE statement, quite some defaults were used:

- No "x variable" was specified. By default this becomes TIME.
- No axis was specified for the y variable. A default axis was drawn with the variable name X along it and with an appropriate numerical range.
- No axis was specified for the x variable TIME. Also for TIME, a default axis is used.

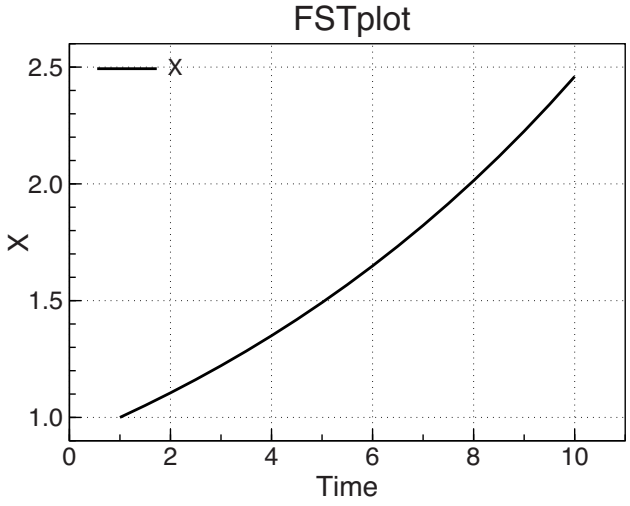


Figure 5.1. Example plot created with the CURVE statement in Listing 5.1.

- No plot name was specified. The default name FSTplot is used.
- No curve type, curve width, curve color or markers were specified. By default, the simulated points are not marked and are connected by black lines with a width of 2 points[1].

If the resulting plot is satisfactory, the use of defaults is alright. Larger models, however, will usually require different line types, different colors or even multiple plots. The next sections shows how this can be achieved.

5.2.2 More detailed CURVE statements

The exponential growth model in Listing 5.1 is really too small to serve as a suitable plotting example. Therefore, in Listing 5.2 a slightly more complex model is used. The model describes two competing populations, with size X1 and X2, each one with logistic growth in the absence of the other.

Obviously, the first thing to do is to plot the two state variables as function of time. This requires two curve statements like in Listing 5.2. Figure 5.2 shows the resulting plot. The curve statements specify that also markers are shown (marker type MARTYP is set at 8) with size[2] MARSIZ of 0.07 cm. The points are connected using solid lines (CURTYP=1) of width CURWID set at 3.0 points[3].

[1] A point is equal to 1/72 inch or 0.35 mm. It is commonly used in typography for font size and is often used for line width as well.

[2] MARSIZ is actually *half* of the marker size (for circular markers the radius).

[3] A point is equal to 1/72 inch or 0.35 mm. It is commonly used in typography for font size and

Listing 5.2 Plotting statements defining a red and a blue curve, each with its points shown as small black dots. Figure 5.2 on page 40 shows the resulting plot.

```
TITLE Lotka-Volterra competition
MODEL
  INITIAL
  DYNAMIC
! the state variables
  X1 = INTGRL (IX1,RX1)
  X2 = INTGRL (IX2,RX2)
  INCON IX1 = 100.0
  INCON IX2 = 100.0
! the growth rates
  RX1 = RGR1 * X1
  RX2 = RGR2 * X2
! relative growth rates are reduced by competition
  RGR1 = A1 * (1.0 - X1/K11 - X2/K12)
  RGR2 = A2 * (1.0 - X2/K22 - X1/K21)
! parameters
  PARAMETER A1=0.1 ; A2=0.2          ! maximum relative growth rates
  PARAMETER K11=1000.0 ; K22=2000.0  ! carrying capacities
  PARAMETER K12=4000.0 ; K21=2500.0  ! the competition parameters
! simulation control
  TIMER STTIME=0.0 ; FINTIM=200.0 ; DELT=0.1 ; PRDEL=5.0
  TRANSLATION_GENERAL DRIVER='RKDRIV'
! plotting
  CURVE YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.; MARTYP=8; MARSIZ=0.07
  CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=3.; MARTYP=8; MARSIZ=0.07
END
```

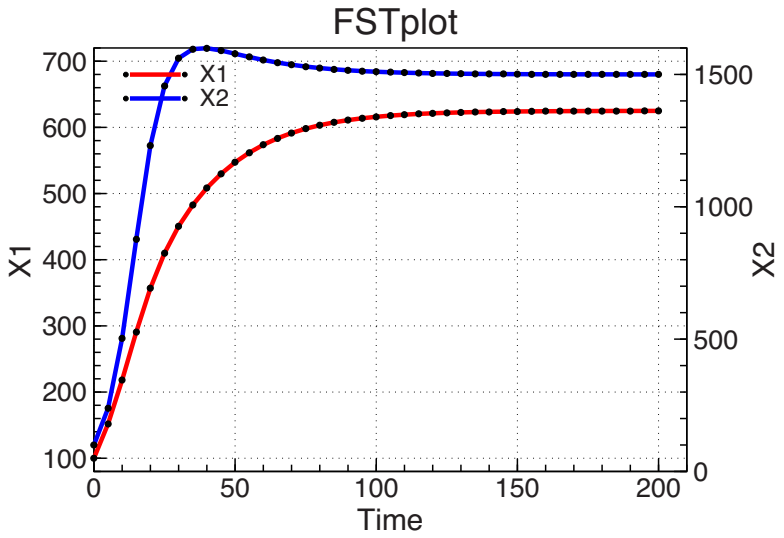


Figure 5.2. Example plot with two curves created with the CURVE statements in model Listing 5.2. Note that each curve refers to its own y-axis.

The various line types and marker types are listed in the Appendices A and B. It usually requires some experimentation to get lines and markers right. Without any specification a solid black curve is drawn. If markers are specified, however, no line is drawn unless also a curve type different from zero is chosen, like in Listing 5.2.

Figure 5.2 also illustrates the default behavior of FST with respect to axes. For each variable a plotting axis is generated with the variable name along it. The numerical range is derived from the simulated values.

In some cases we may want a single axis for two or more variables, for instance in case they both are concentrations, or the components of a vector. Further, instead of just a variable name, we want a more meaningful axis text containing also the unit used. This is the subject of the next section.

5.2.3 A shared axis

If the variables X1 and X2 must share a single axis, this axis needs to be given a name, say "PopAxis". Using this name the axis can be assigned to the variables X1 and X2 by means of `ASSIGN_AXIS PopAxis > X1,X2`. FST will then plot the two variables to a single axis and will write the axis text "X1" along it[4].

Instead of letting FST decide, an axis may also be *explicitly defined* in a statement `DEFINE_AXIS PopAxis='population size (#)'`. This definition replaces the default axis text 'X1'. FST still figures out, however, what the numerical range of the axis should be.

The modeler has full control by using a complete axis definition statement like `DEFINE_AXIS PopAxis='0.0 2000.0 500.0 100.0 0.0 population size (#)'`. This sets the range to $[0, 2000]$, writes labels (numbers) at $0, 500, 1000, \ldots, 2000$ and

is often used for line width as well.

[4]The text written along the axis is the name of the first assigned variable. This prevents problems in case of many variables sharing a single axis.

Listing 5.3 Plotting statements defining two curves plotted using a single y-axis. Figure 5.3 shows the resulting plot.

```
TITLE Lotka-Volterra competition
MODEL
  INITIAL
  DYNAMIC
! the state variables
  X1 = INTGRL (IX1,RX1)
  X2 = INTGRL (IX2,RX2)
  INCON IX1 = 100.0
  INCON IX2 = 100.0
! the growth rates
  RX1 = RGR1 * X1
  RX2 = RGR2 * X2
! relative growth rates are reduced by competition
  RGR1 = A1 * (1.0 - X1/K11 - X2/K12)
  RGR2 = A2 * (1.0 - X2/K22 - X1/K21)
! parameters
  PARAMETER A1=0.1 ; A2=0.2          ! maximum relative growth rates
  PARAMETER K11=1000.0 ; K22=2000.0  ! carrying capacities
  PARAMETER K12=4000.0 ; K21=2500.0  ! the competition parameters
! simulation control
  TIMER STTIME=0.0 ; FINTIM=200.0 ; DELT=0.1 ; PRDEL=5.0
  TRANSLATION_GENERAL DRIVER='RKDRIV'
! plotting
  ASSIGN_AXIS PopAxis > X1, X2
  DEFINE_AXIS PopAxis = '0.0 2000.0 500.0 100.0 0.0 population size (#)'
  CURVE YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
  CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=3.0; Legend='number X2'
END
```

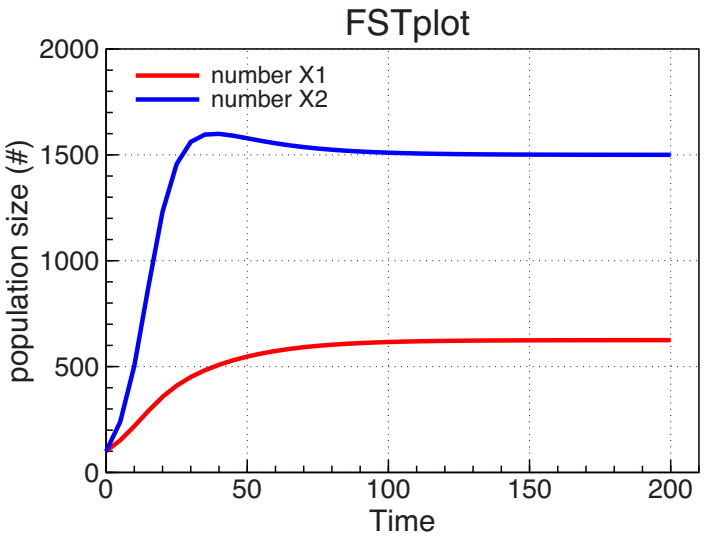


Figure 5.3. Example plot created with the model in Listing 5.3. Note that the two curves share a single y-axis.

applies a tickmark distance of 100.0. With the fifth value in the axis definition (here 0.0) one may control the position of the first label[5].

This last axis definition is part of the model in Listing 5.3, which produces the plot in Figure 5.3. Note that we have omitted the markers again, made use of the default curve type and also added an explicit legend text for each curve.

5.2.4 Two separate axes

Even if two model variables have the same unit, their values may be very different. In such cases a shared axis, though desirable from a conceptual point of view, is not very practical.

Two user-defined axes are defined and assigned in a straightforward way, like in the following plotting section

```
! plotting
  ASSIGN_AXIS X1axis > X1
  ASSIGN_AXIS X2axis > X2
  DEFINE_AXIS X1axis = '0.0 1000.0 250.0  50.0 0.0 population 1 (#)'
  DEFINE_AXIS X2axis = '0.0 2000.0 500.0 100.0 0.0 population 2 (#)'
  CURVE YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
  CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=3.0; Legend='number X2'
```

The resulting plot is shown in Figure 5.4.

[5] A zero value as in this example does not do anything. If you want the axis to begin at -200 and the labels should nevertheless be put at $0, 500, 1000, \ldots, 2000$, the axis definition would be DEFINE_AXIS PopAxis='-200.0 2000.0 500.0 100.0 -500.0 population size (#)'. This sets the starting point for the label generation at -500 and produces the labels at the desired positions (instead of the awkward series $-200, +300, +800, \ldots$).

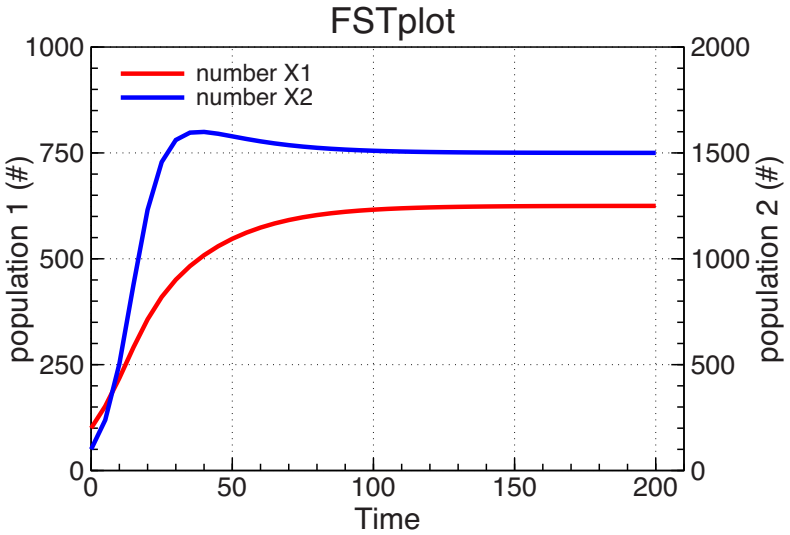


Figure 5.4. Competion model plot with two user-defined axes, made with the plotting statements in section 5.2.4.

5.2.5 A second plot

A proper inspection of model results will usually require several plots, each one illustrating a different aspect of the simulation. As an example, we will add a plot of the trajectory in state space to our competition model[6]

Plots in FST are called "frames", each frame being a rectangular figure with a title, axes, curves and a legend. The various frames are distinguished from each other by their names. The time plot in Figure 5.4, for instance, could be named "TimePlot" and this exactly what is done in the plotting section of Listing 5.4. We will now explain this section in detail.

The first four statements define and assign the axes and are exactly the same as in section 5.2.4. The CURVE statements in Listing 5.4 are different though:

CURVE 1. The first frame is named "TimePlot". For clarity, the x-variable is set with XVAR=TIME and the first plotted variable X1 is specified in the same way as before (note the continuation line of this statement).

CURVE 2. This statement defines the curve for X2 in exactly the same way as before. No new frame name is mentioned, which implies that the curve for X2 is added to the previous frame (the "TimePlot" frame). The same holds for the x-variable XVAR. It is omitted from this statement implying that the previous x-variable (TIME) is used again.

CURVE 3. The second frame is named "StateSpace" and a curve is defined by setting the x-variable at X1 and the Y-variable at X2. The curve width is 3 points, the curve color is default (black) and there will be no markers.

The first plot in Figure 5.5 is almost identical to the one in Figure 5.4. The only difference is that the plot title is now "TimePlot", as stated in the first CURVE statement.

[6]State space in this case is the plane spanned by an axis X1 and an axis X2. Each point (X1,X2) of the plane represents a possible status of the system and a simulation in time corresponds to a trajectory in the plane. Visualization of this trajectory is useful in the study of model behavior.

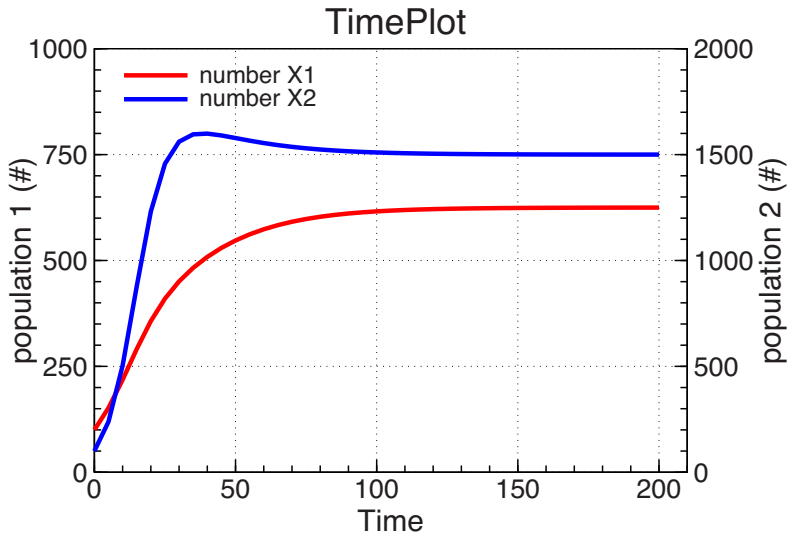


Figure 5.5. Timeplot made with the first two CURVE statements in Listing 5.4.

Listing 5.4 Plotting statements defining the time plot in Figure 5.5 and the state space plot in Figure 5.6.

```
TITLE Lotka-Volterra competition
MODEL
  INITIAL
  DYNAMIC
! the state variables
  X1 = INTGRL (IX1,RX1)
  X2 = INTGRL (IX2,RX2)
  INCON IX1 = 100.0
  INCON IX2 = 100.0
! the growth rates
  RX1 = RGR1 * X1
  RX2 = RGR2 * X2
! relative growth rates are reduced by competition
  RGR1 = A1 * (1.0 - X1/K11 - X2/K12)
  RGR2 = A2 * (1.0 - X2/K22 - X1/K21)
! parameters
  PARAMETER A1=0.1 ; A2=0.2          ! maximum relative growth rates
  PARAMETER K11=1000.0 ; K22=2000.0  ! carrying capacities
  PARAMETER K12=4000.0 ; K21=2500.0  ! the competition parameters
! simulation control
  TIMER STTIME=0.0 ; FINTIM=200.0 ; DELT=0.1 ; PRDEL=5.0
  TRANSLATION_GENERAL DRIVER='RKDRIV'
! plotting
  ASSIGN_AXIS X1axis > X1
  ASSIGN_AXIS X2axis > X2
  DEFINE_AXIS X1axis = '0.0 1000.0 250.0  50.0 0.0 population 1 (#)'
  DEFINE_AXIS X2axis = '0.0 2000.0 500.0 100.0 0.0 population 2 (#)'
  CURVE XVAR=TIME ; FRAME='TimePlot' ; ...
        YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
  CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=3.0; Legend='number X2'
  CURVE XVAR=X1 ; YVAR=X2; FRAME='StateSpace'; ...
        CURWID=3.0; Legend='trajectory of (X1,X2)'
END
```

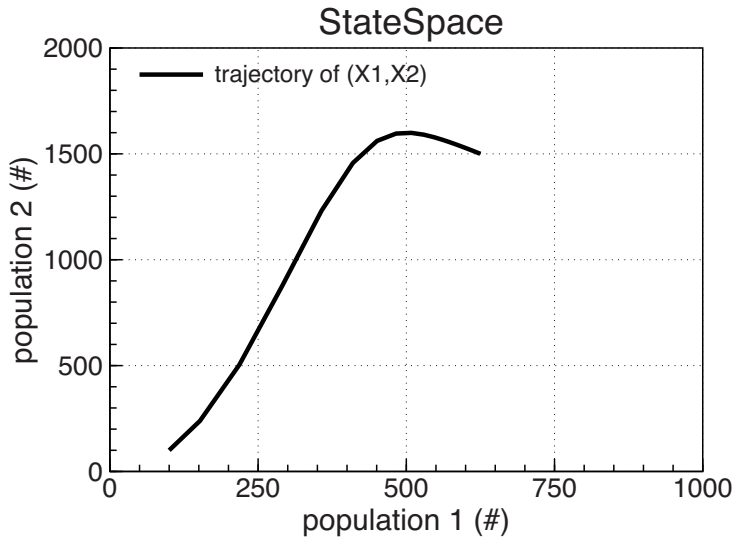


Figure 5.6. State space plot made with the plotting section in Listing 5.4. Note that the two axes are the same as in the timeplot of Figure 5.5.

The second plot in Figure 5.6 is the desired state space plot. Note that the axes used for the variables X1 and X2 are the same as in the timeplot of Figure 5.5. This illustrates an important feature of FST plotting: *an axis in FST belongs to one or more variables and not to a plot or frame.*

Hence, the axis assigned to X1 in the first ASSIGN_AXIS statement is used in all frames in which variable X1 is plotted. There is no need to worry about the drawing of x- and y-axes. The rule is very simple: all axes required for the plotted variables are drawn along or aside from the frameThere may be multiple vertical axes and multiple horizontal axes. The x- and y-axis implied by the first CURVE statement of the frame are the primary axes, drawn at their usual positions.

A variable can be plotted only relative to axis assigned to it. We cannot choose two different axes for the same variable if we plot it twice. In such a (rare) case we need to plot a copy of the variable in the second plot.

Listing 5.5 The competition model with three reruns produces the four timeplots in Figure 5.7 and the four state space plots in Figure 5.8.

```
TITLE Lotka-Volterra competition
MODEL
  INITIAL
  DYNAMIC
! the state variables
  X1 = INTGRL (IX1,RX1)
  X2 = INTGRL (IX2,RX2)
  INCON IX1 = 100.0
  INCON IX2 = 100.0
! the growth rates
  RX1 = RGR1 * X1
  RX2 = RGR2 * X2
! relative growth rates are reduced by competition
  RGR1 = A1 * (1.0 - X1/K11 - X2/K12)
  RGR2 = A2 * (1.0 - X2/K22 - X1/K21)
! parameters
  PARAMETER A1=0.1 ; A2=0.2          ! maximum relative growth rates
  PARAMETER K11=1000.0 ; K22=2000.0  ! carrying capacities
  PARAMETER K12=4000.0 ; K21=2500.0  ! the competition parameters
! simulation control
  TIMER STTIME=0.0 ; FINTIM=200.0 ; DELT=0.1 ; PRDEL=5.0
  TRANSLATION_GENERAL DRIVER='RKDRIV'
! plotting
  ASSIGN_AXIS X1axis > X1
  ASSIGN_AXIS X2axis > X2
  DEFINE_AXIS X1axis = '0.0 1000.0 250.0  50.0 0.0 population 1 (#)'
  DEFINE_AXIS X2axis = '0.0 2000.0 500.0 100.0 0.0 population 2 (#)'
  CURVE XVAR=TIME ; FRAME='TimePlot' ; ...
        YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
  CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=3.0; Legend='number X2'
  CURVE XVAR=X1 ; YVAR=X2; FRAME='StateSpace'; ...
        CURWID=3.0; Legend='trajectory of (X1,X2)'
END
  INCON IX1 = 200.0
END
  INCON IX1 = 300.0
END
  INCON IX1 = 400.0
END
```

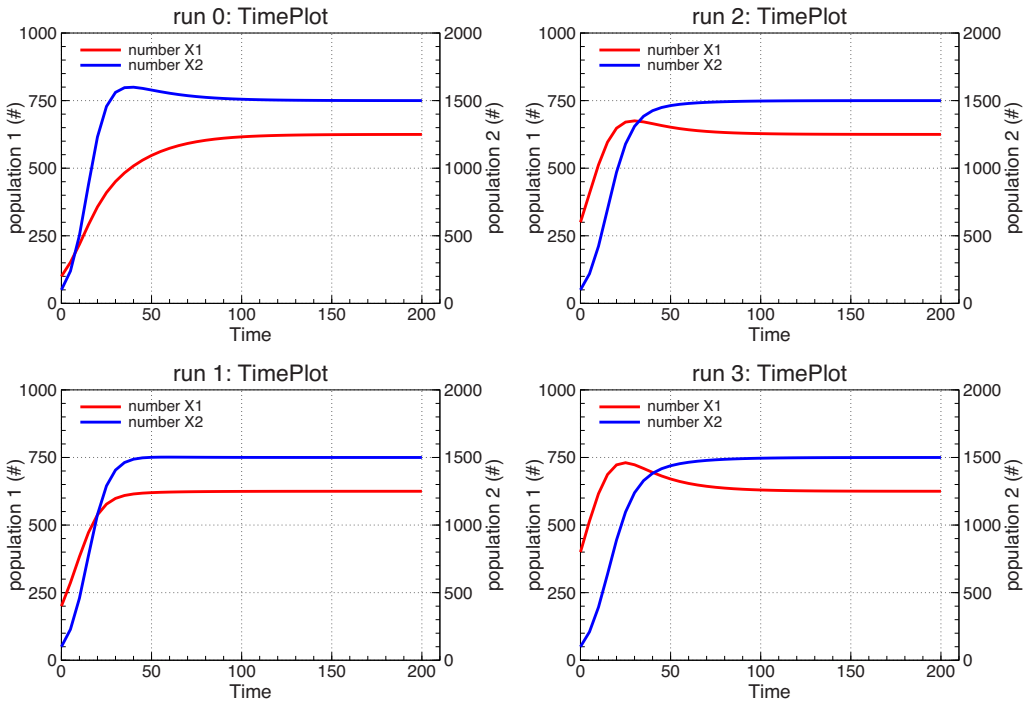


Figure 5.7. Time plots made by means of the three reruns in Listing 5.5.

5.2.6 Reruns and plotting

At the end of a model, reruns may be defined by means of new values for model parameters or initial constants (see for details Rappoldt & van Kraalingen, 1996, section 3.6). This has been applied in Listing 5.5, in which the initial value of X1 is redefined in three rerun sections, leading to a total of four model runs.

The reruns produce a separate plot for each run, as you might expect. Figure 5.7 shows the four timeplots made with the model in Listing 5.5 at page 45. The run number is automatically added to the plot title. The four plots, however, are written in the same file, an EPS file with four pages. This allows a comparison of runs by simply stepping through the EPS pages on screen. Collecting the plots in a single figure, as in Figure 5.7, requires a graphics editor like Adobe Illustrator. Similarly, Figure 5.8 shows the four state space plots.

A warning

If an axis range is not explicitly defined in a DEFINE_AXIS statement, FST will find out a suitable range (e.g. section 5.2.2). *It will do so for each run seperately, however.* Hence, if you step through the pages of an EPS file the plots change not just by the changes in the model, but also by changes in the axes. We therefore advise the use of a defined axis range in case reruns are made.

5.2.7 Combining runs in a sensitivity plot

Sofar we have considered plots containing results from a single model run. This "normal type" is the default type and could have been set explicitly by adding

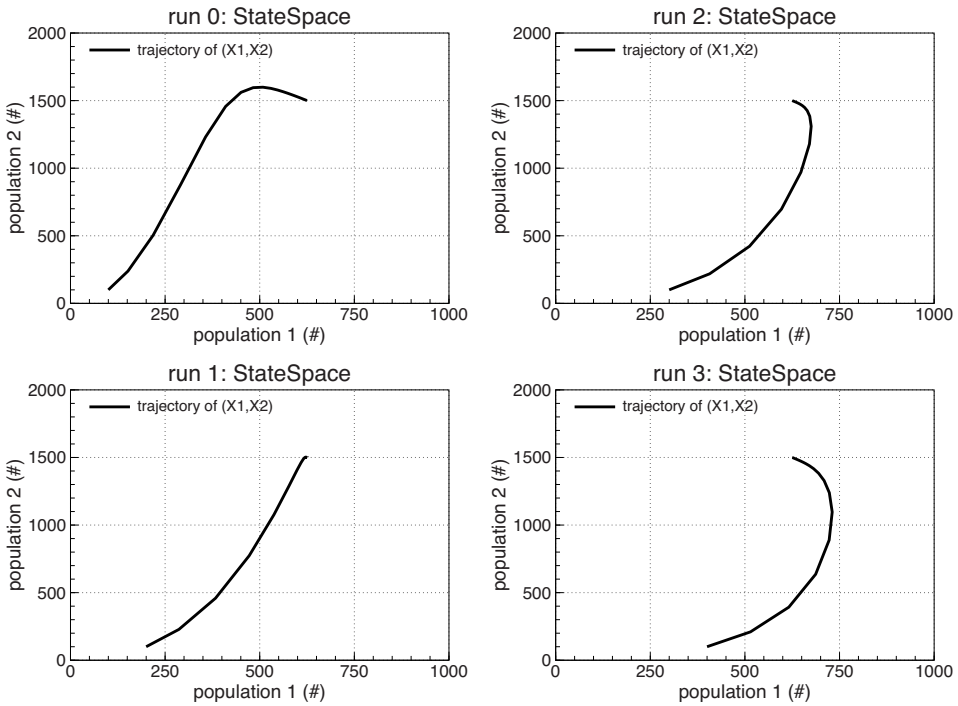


Figure 5.8. State space plots made by means of the three reruns in Listing 5.5.

FrameType=1 to the CURVE statements. There is a second type of plot, which is set by **FrameType=2**. The CURVE statements in Listing 5.5 then become

```
CURVE XVAR=TIME ; FRAME='TimePlot'; FrameType=2; ...
      YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=3.0; Legend='number X2'
CURVE XVAR=X1 ; YVAR=X2; FRAME='StateSpace'; FrameType=2; ...
      CURWID=3.0; Legend='trajectory of (X1,X2)'
```

This combines, for each frame, all runs into a single plot. The resulting time and state space plots in Figure 5.9 indeed combine the curves in Figures 5.7 and 5.8. This type of plot is called "sensitivity plot" for reasons that will become more clear in the next section.

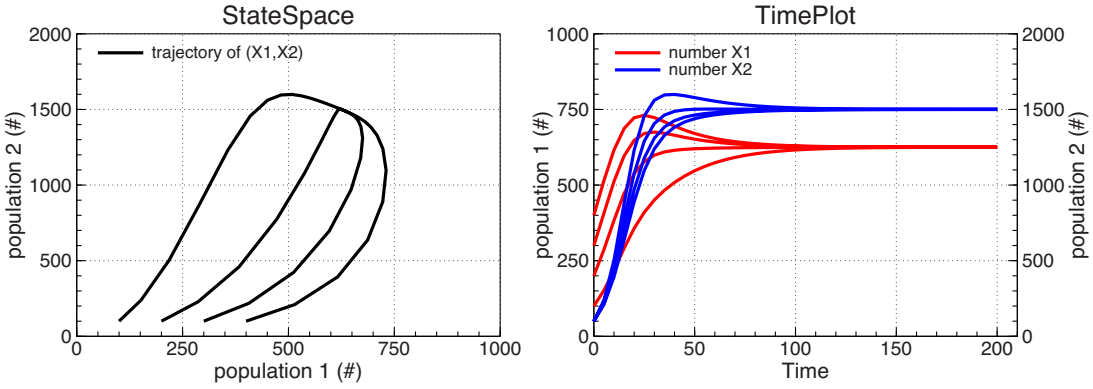


Figure 5.9. The result of adding **FrameType=2** to the CURVE statements inListing 5.5 (which results in the CURVE statements in section 5.2.7).

5.2.8 More on sensitivity plots

Figure 5.9 illustrates a well known property of this kind of competition models. Under certain conditions, the system evolves to a stable equilibrium (X1,X2), irrespective of the initial populations sizes. We will not go into the mathematics or the biology here, but we study this equilibrium numerically by means of 30 reruns for different values of the model parameter K12.

The 30 runs are made by the FST model in Listing 5.6. The model itself did not change at all, but there are a few technical changes relative to Listing 5.5 on which we make the following comments

- The value of FINTIM has been increased to 500.0 in order to reach equilibrium in all reruns.
- PRDEL has been reduced to 2.0 in order to get more smooth curves.
- A terminal section has been added to the model in which the final (equilibrium) values of X1 and X2 are assigned to FX1 and FX2 respectively. These terminal variables FX1 and FX2 are set only once during a model run, at the end of the simulation.
- The axes X1axis and X2axis are assigned to the terminal variables FX1 and FX2 as well.
- The time plot is the same as the one in section 5.2.7. Only the curve width has been reduced since there will be 60 curves in the plot now.
- In a third type 2 frame named "Sensitivity" the equilibrium values FX1 and FX2 are plotted as function of varying parameter K12. Note that no axis is defined for K12, which means that the FST will construct one.
- A fourth type 2 frame named "Equilibrium (X1,X2)" is a plot of the equilibrium status (FX1,FX2) in state space. Note that the legend text has been set to a space which means that no legend is drawn.
- The SENSITIVITY statement automatically generates 30 reruns in which parameter K12 is given values in the range [2200,7000].

The four plots produced can be inspected in Figure 5.10. The plots StateSpace and TimePlot at the top just combine results of all runs, very much like Figure 5.9. The curves in these plots would also be there as ordinary plots made with FrameType=1 (but they would be in 30 different plots).

The two bottom plots in Figure 5.10 are different. Each curve consists of 30 interconnected points and each of those points is the result of a model run. Along the curves the *model parameter* K12 changes instead of time or some dynamic variable. As a type 1 frame these curves would not exist at all[7].

5.2.9 Changing the time axis

In most cases, the default axis for TIME will be satisfactory. An advantage of the default is that it automatically changes into a calendar time axis (section 5.2.10) in case of a calendar connection (Chapter 7). Here we briefly discuss how to change the time axis *in the absence of a calendar connection.*

[7] You might expect that the frame "Sensitivity" with FrameType=1 produces 30 plots, each with two points (K12,FX1) and (K12,FX2). This is not what really happens, however. SInce K12 is part of the initial output and FX1 and FX2 are part of the terminal output, a type 1 frame with XVAR=K12 and YVAR=FX1 is empty (try it!).

Listing 5.6 The example model with automated reruns on parameter K12 with the combined results of 30 runs in four sensitivity plots shown in Figure 5.9.

```
TITLE Lotka-Volterra competition
MODEL
  INITIAL
  DYNAMIC
! the state variables
  X1 = INTGRL (IX1,RX1)
  X2 = INTGRL (IX2,RX2)
  INCON IX1 = 100.0
  INCON IX2 = 100.0
! the growth rates
  RX1 = RGR1 * X1
  RX2 = RGR2 * X2
! relative growth rates are reduced by competition
  RGR1 = A1 * (1.0 - X1/K11 - X2/K12)
  RGR2 = A2 * (1.0 - X2/K22 - X1/K21)
! parameters
  PARAMETER A1=0.1 ; A2=0.2          ! maximum relative growth rates
  PARAMETER K11=1000.0 ; K22=2000.0  ! carrying capacities
  PARAMETER K12=4000.0 ; K21=2500.0  ! the competition parameters
! simulation control
  TIMER STTIME=0.0 ; FINTIM=500.0 ; DELT=0.1 ; PRDEL=2.0
  TRANSLATION_GENERAL DRIVER='RKDRIV'
! final values of X1 and X2
  TERMINAL
  FX1 = X1
  FX2 = X2
! plotting
  ASSIGN_AXIS X1axis > X1, FX1
  ASSIGN_AXIS X2axis > X2, FX2
  DEFINE_AXIS X1axis = '0.0 1000.0 250.0  50.0 0.0 population 1 (#)'
  DEFINE_AXIS X2axis = '0.0 2000.0 500.0 100.0 0.0 population 2 (#)'
! time plot
  CURVE XVAR=TIME ; FRAME='TimePlot' ; FrameType=2; ...
        YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=1.0; Legend='number X1'
  CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=1.0; Legend='number X2'
! state space
  CURVE XVAR=X1 ; YVAR=X2; FRAME='StateSpace'; FrameType=2; ...
        CURWID=1.0; Legend='trajectory of (X1,X2)'
! sensitivity plot
  CURVE XVAR=K12; FRAME='Sensitivity'; FrameType=2; ...
        YVAR=FX1; CURCOL='Red' ; CURWID=3.0; Legend='equilibrium X1'
  CURVE YVAR=FX2; CURCOL='Blue'; CURWID=3.0; Legend='equilibrium X2'
! equilibrium positions in state space
  CURVE XVAR=FX1 ; YVAR=FX2; FRAME='Equilibrium (X1,X2)'; FrameType=2; ...
        CURWID=3.0; Legend=' '
! automated reruns on K12
  SENSITIVITY Varying=K12; BeginRange=2200.;EndRange=7000.;NumberOfRuns=30
END
```

Suppose we want to change the time axis in Figure 5.5 at page 43. Then we may add the following statements to the model in Listing 5.4 at page 44.

```
DEFINE_AXIS Time_axis = '0.0 200.0 40.0 10.0 0.0 time (days)'
ASSIGN_AXIS Time_axis > TIME
```

This works exactly as for any other variable. There are a few limitations, however.

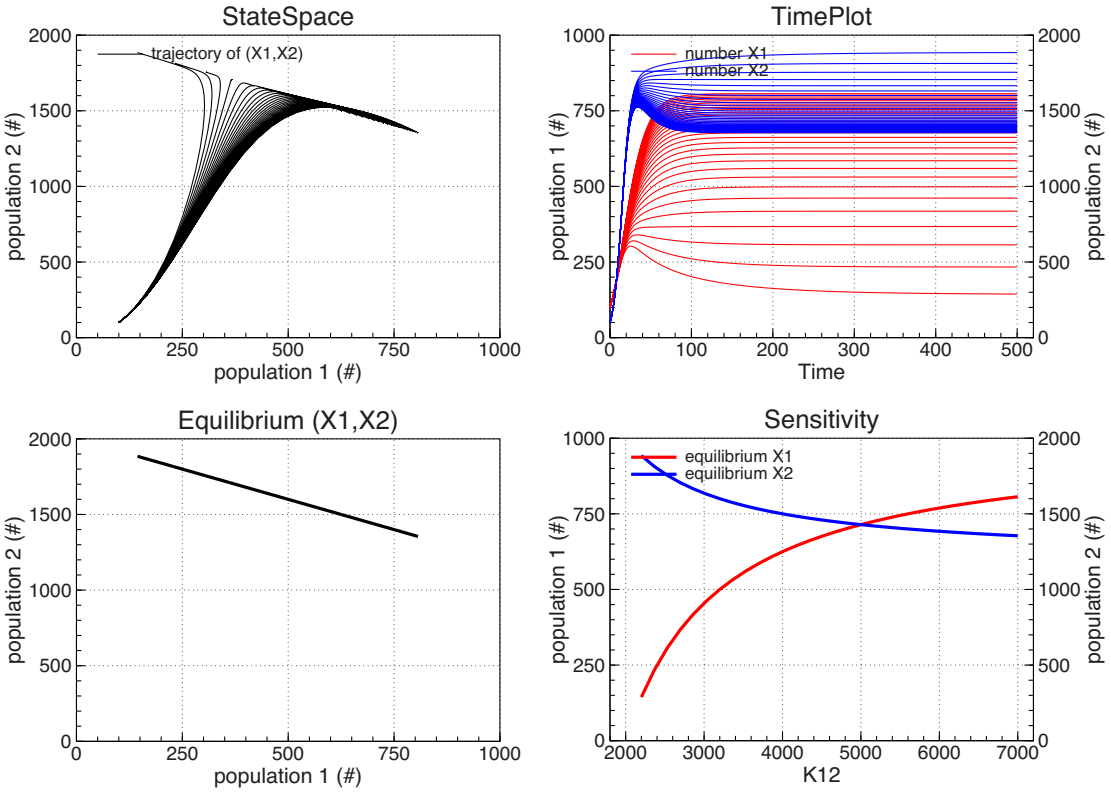


Figure 5.10. The four sensitivity plots made with the model in Listing 5.6 at page 49.

- The name is the axis **must be "Time_axis"**. No other axis can be assigned to variable TIME.
- There must not be a calendar connection (e.g. Chapter 7). If there is a calendar connection "Time_axis" can still be used but the label and tick distances in the axis definition will have another meaning (e.g. section 5.2.10).

The above Time_axis results in a horizontal axis with range $[0, 200]$ (instead of $[0, 210]$ of the automatic axis in Figure 5.5), a label distance of 40 days (instead of 50) and an axis text "time (days)" instead of the automatic text "Time".

5.2.10 Plotting calendar time

Calendar connection

A calendar connection means that each value of simulated time TIME corresponds to a unique date and time of the ordinary calendar. A calendar connection consists of the following two elements (see also Chapter 7).

- The value of STTIME, which is the initial value of TIME, is associated with a precisely defined time on the calendar.
- The unit of simulated TIME is set. This unit in combination with the value of TIME−STTIME then determines the calendar time of any value of TIME.

In the competition model used here as an example we can make a calendar connection by adding for instance

```
TRANSLATION_GENERAL StartYear=2010 ; StartDOY=1.0 ; OneDay=1.0
```

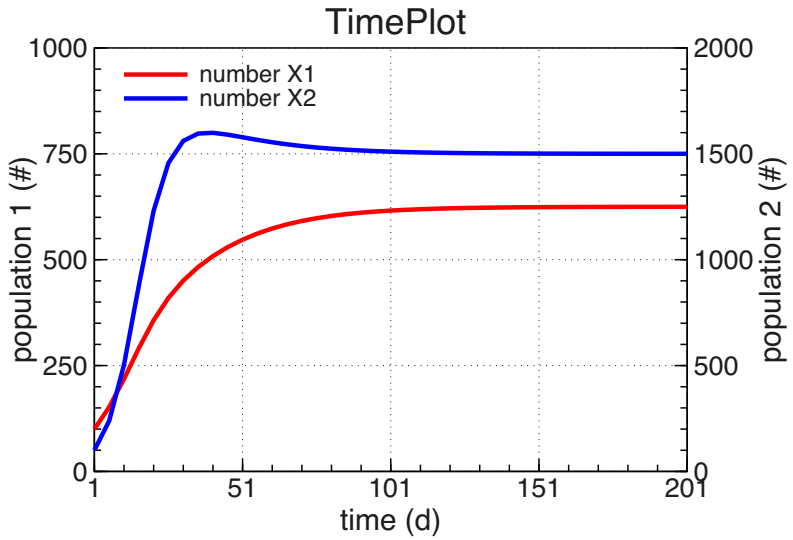


Figure 5.11. The frame TimePlot made by adding the calendar connection in section 5.2.10 to the model in Listing 5.4. The time axis is a default calendar time axis.

This sets STTIME equal to the begin of day 1 of 2010 (STTIME is 00:00:00 at January 1, 2010) and further declares that one calendar day (24 hours) consists of 1.0 units of simulated time, which implies that simulated time is in days[8].

Default calendar time axis

If we add the above statement to the model in Listing 5.4 the TimePlot changes into the one in Figure 5.11. Depending on the length of the simulated period, the time axis will be given in hours, days, months or years. Note that day numbers along the time axis are day-of-year numbers which start at 1.00 at the beginning of a new year[9].

Changing the calendar time axis

The default division of a calendar time axis in hours, days, months or years can be changed for each frame by defining a time label TLabel in a CURVE statement. This time label divides calendar time in hours, days, weeks, months or years and also defines what has to be written as time labels along the axis.

In Listing 5.7 at page 53 you find an example. The time label has been set for the TimePlot frame by TLABEL='$IW=#WEEKNR$'. The resulting plot is shown in Figure 5.12a. There is a division of the time axis in weeks (the 'W' left of the '=' does that) and the labels are placed in the middle of week intervals (the 'I' left of the '=' does that). The labels themselves consist of the weeknumber '#WEEKNR'.

Note that the distinct weeks in Figure 5.12a are not simply periods of 7 days. They

[8] If simulated time is in seconds, we would write OneDay=86400.0. If we simulate time in hours, OneDay=24.0, etc.

[9] The day-of-year number is not an integer number. If StartDOY is set at 1.5 the simulation starts at 12:00 of January,1. The reason for using these day-of-year numbers (starting the year at 1.0 instead of 0.0) is compatibility with crop growth models in which weather data, start time and harvest time are expressed as day-of-year numbers.

are weeks from monday to sunday with numbers according to the ISO 8601 standard (http://en.wikipedia.org/wiki/Seven-day_week#Week_numbering and http://tuxgraphics.org/toolbox/calendar.html). In Figure 5.12a you may notice that the first week label is 53. The reason is that January,1 2010 was a friday and therefore belonged to week 53 of the previous year.

The structure of the time label string is as follows.

- The entire string is between dollar signs '$'.
- The two character code before the equality sign '=' determines the way in which time is divided.
- The first character is either 'L' or 'I' which stands for 'label' or 'interval'. In case of a labeled axis the time labels (day numbers, week numbers, whatever) are placed at a certain time, like numbers along an ordinary numerical axis. In case of an interval axis the time labels are placed in the middle of a time interval, like the week numbers in Figure 5.12a.
- The second character is 'H' (for hours), 'D' (for days), 'W' (for weeks), 'M' (for months) or 'Y' (for years).
- At the right side of the '=' sign there is one or more codes for week number, month name in English, month name in Dutch, etc. Between the codes there may be characters which are copied to each label.

In Figure 5.12 a few examples can be studied. The axis in Figure 5.12c is a labeled month axis ("$LM") in which the label consists of a combination of #day (which is day of month) and #monthst (which is the short english month name). Note the

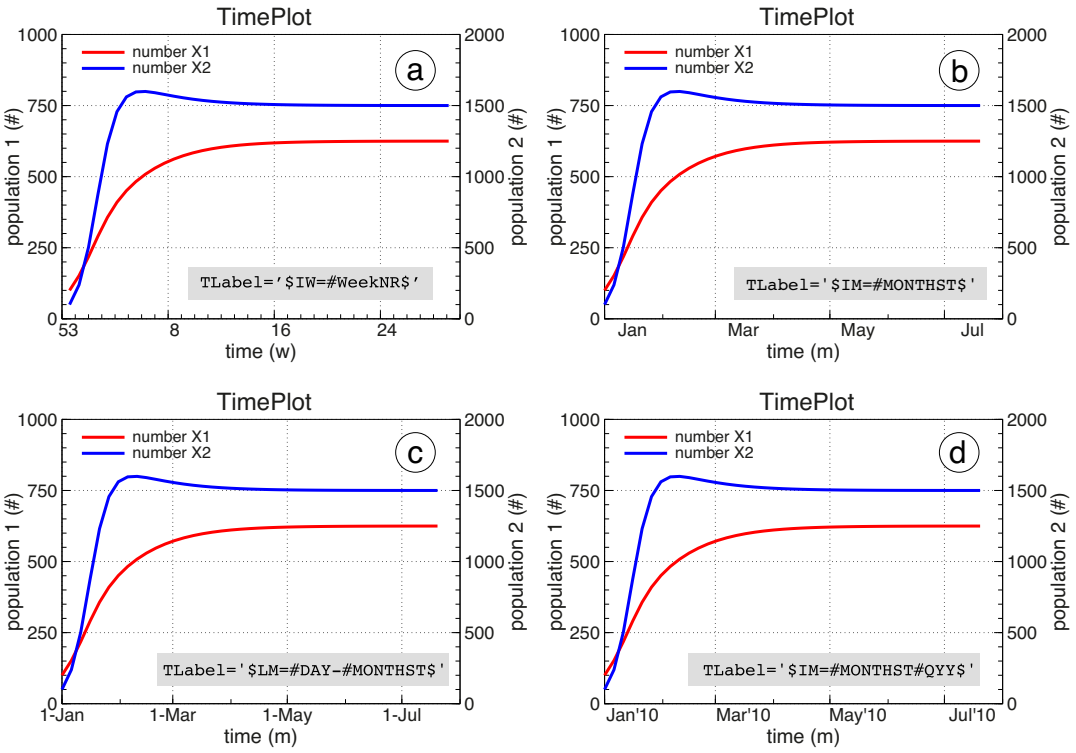


Figure 5.12. Plots made with the model in Listing 5.7 at page 53 using different TLabel settings in the first CURVE statement of frame TimePlot. The four plots contain the TLabel string used for their time axis construction.

Listing 5.7 The example model with the use of TLabel for specifying a calendar time axis. Examples are shown in Figure 5.12.

```
TITLE Lotka-Volterra competition
MODEL
  INITIAL
  DYNAMIC
! the state variables
  X1 = INTGRL (IX1,RX1)
  X2 = INTGRL (IX2,RX2)
  INCON IX1 = 100.0
  INCON IX2 = 100.0
! the growth rates
  RX1 = RGR1 * X1
  RX2 = RGR2 * X2
! relative growth rates are reduced by competition
  RGR1 = A1 * (1.0 - X1/K11 - X2/K12)
  RGR2 = A2 * (1.0 - X2/K22 - X1/K21)
! parameters
  PARAMETER A1=0.1 ; A2=0.2          ! maximum relative growth rates
  PARAMETER K11=1000.0 ; K22=2000.0  ! carrying capacities
  PARAMETER K12=4000.0 ; K21=2500.0  ! the competition parameters
! simulation control
  TIMER STTIME=0.0 ; FINTIM=200.0 ; DELT=0.1 ; PRDEL=5.0
  TRANSLATION_GENERAL DRIVER='RKDRIV'
! calendar connection
  TRANSLATION_GENERAL StartYear=2010 ; StartDOY=1.0 ; OneDay=1.0
! plotting
  ASSIGN_AXIS X1axis > X1
  ASSIGN_AXIS X2axis > X2
  DEFINE_AXIS X1axis = '0.0 1000.0 250.0  50.0 0.0 population 1 (#)'
  DEFINE_AXIS X2axis = '0.0 2000.0 500.0 100.0 0.0 population 2 (#)'
  CURVE XVAR=TIME; Frame='TimePlot'; TLABEL='$IM=#MONTHST#QYY$'; ...
        YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
  CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=3.0; Legend='number X2'
END
```

difference with Figure 5.12b in which the month name is used to label an interval month axis ("$IM").

More details on time labels can be found in Appendix D . A list of all label codes can be found there as well, in Table D.2 at page 98.

Tuning the hour, day, week, month or year axis

The primary tool of changing a calendar time axis is the time label string discussed in the previous section. The axis "Time_axis" can be used to do some fine-tuning.

Useful is changing the axis text. If the time label produces an axis in weeks, for instance, an axis text "week" looks better than the default text "time (w)" (see Figure 5.12a). This can be achieved by

```
DEFINE_AXIS Time_axis = 'week'
ASSIGN_AXIS Time_axis > TIME
```

Numbers added to the time axis definition have their usual meaning. They define the axis range, the label and tick distance, and the value of the first label (cf. section 5.2.3 and section 5.2.9). In case of a calendar connection, however, the

label and tick distance in "Time_axis" have a slightly different meaning. Writing the time axis definition as `Time_axis='X1 X2 X3 X4 X5 text'`, the following rules apply.

- **The numbers X1 and X2 in ordinary order (X1 < X2)** supply the axis range [X1,X2] in units simulation time. The two values may lie below STTIME and above FINTIM respectively. In that case the axis' range will be extended beyond the simulated time interval. Two equal values (X1=X2, e.g. `0.0 0.0`) mean that FST finds out a proper range.
- **The numbers X1 and X2 in reverse order (X1 > X2)** also supply the axis range (now [X2,X1]), but FST extends the axis in both directions to the nearest hour, day, week, month or year, depending on the TLabel used.
- **Number X3** is the label distance *in axis time units, as selected by the TLabel.* The interval axis in Figure 5.12a, for instance, uses a label distance of 8 weeks. With `DEFINE_AXIS Time_axis='0.0 0.0 4.0 1.0 0.0 week'` this is changed into 4 weeks. Similarly, the label distance for a month axis as in Figure 5.12b can be set to 3 months (quarters). A zero X3 means that no labels are plotted. A negative value means that FST finds out.
- **Number X4** is the tick distance, *also in axis time units.* The ticks are drawn starting from the first label in both directions. For an interval axis with labels, the tick distance is always 1.0 axis time unit and cannot be changed. A zero X4 means that no ticks are plotted. A negative value means that FST finds out.
- Note that **label distance X3 and tick distance X4** are interpreted as integer values (the integer nearest to the supplied value is taken). Only in case of an axis in hours, the fractional parts of X3 and X4 are used to plot labels and/or ticks at fractional positions between the full hours.
- **Number X5** is the value of the first label in units simulation time (the units of STTIME and FINTIM). FST uses this value as an indication, since the actual label position must correspond to either the middle (in case of an interval axis) or the beginning (otherwise) of an hour, day, week, month, or year. FST labels the interval in which the supplied X5 lies[10]. A zero X5 means that FST finds out[11].

These rules are applied in a few examples below. The combination of a TLabel and a Time_axis allows the construction of practical calendar time axes, which is important if measured data are evaluated and compared with simulation results.

Some more examples

As a first example we take the model in Listing 5.7 at page 53 with the month interval axis in Figure 5.12b. We extend the plotted time range and write full english month names as quarterly labels. The Time_axis and Curve statements doing this are

```
DEFINE_AXIS Time_axis = '-20.0 230.0 3.0 1.0 10.0'
ASSIGN_AXIS Time_axis > TIME
CURVE XVAR=TIME; Frame='TimePlot'; TLABEL='$IM=#MonthLT$'; ...
      YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
```

[10] The label is actually be written only if it is within range. If the value X5 provided lies left of axis begin, the label distance X3 is used to figure out where the first label should appear.

[11] If you want 0.0 as a first label position, you have to supply a slightly different value which lies in the same hour, day, week, month or year.

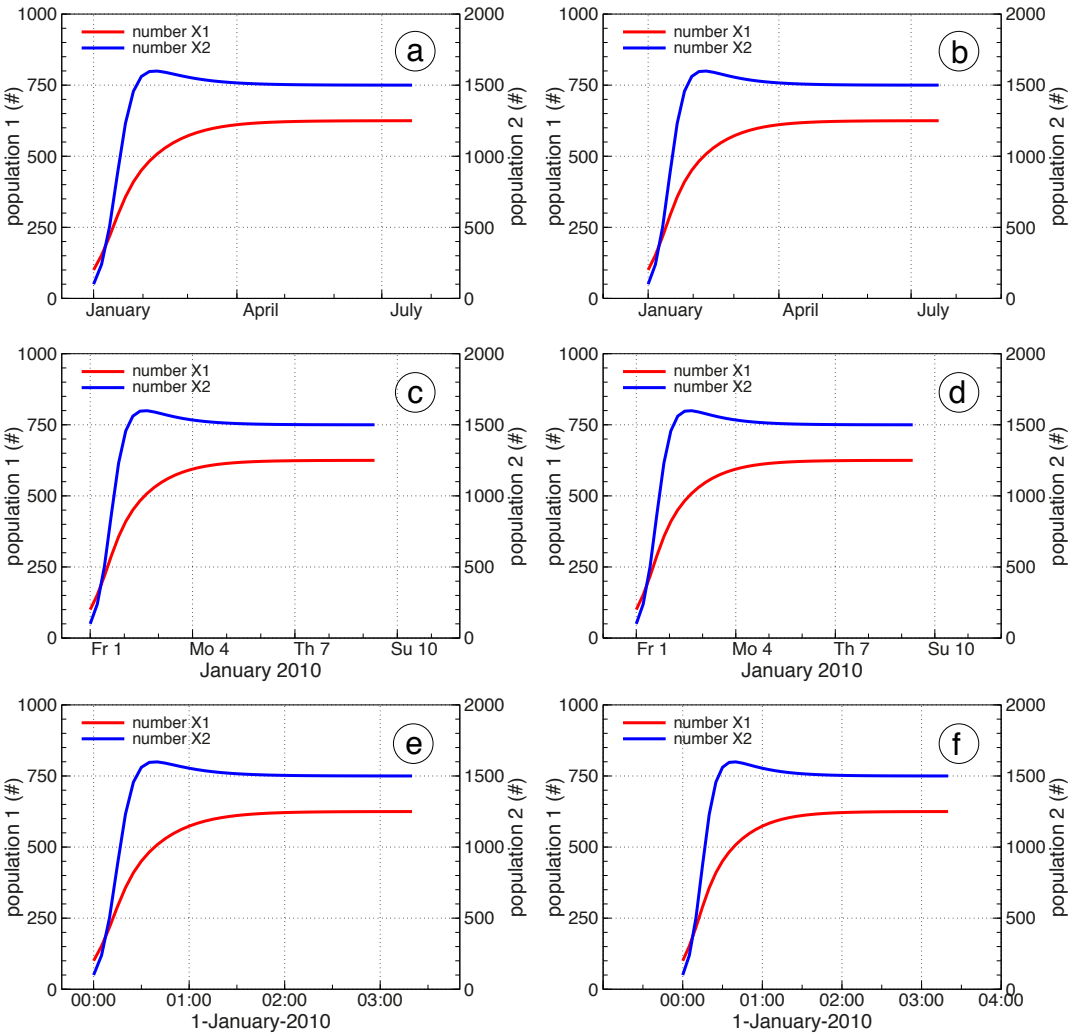


Figure 5.13. Plots made using Listing 5.7 with various Time_axis definitions added. For a detailed description see the text on the examples at page 54. (a) Month interval axis beginning in december (time=−20) and ending in august with first label at Time=10, which is january. (b) Same but with axis bounds automatically extended to the nearest month begin. (c,d) Simulation time unit changed into hours. (e,f) Simulation time units changed to minutes (by means of OneDay=1440.0), hourly time labels with half hour ticks.

The result is shown in Figure 5.13a. A few remarks on the Time_axis statement:

- The calendar connection in Listing 5.7 states that simulation time is in days (OneDay=1.0), starting at January,1 of 2010. This means that the axis start value of −20 lies somewhere in december of the previous year.
- The first label position is X5=10.0, ten days after the start at STTIME=0.0, which is clearly january.
- There is no axis text in the Time_axis statement and no text appears along the time axis. Full month names indeed may be sufficient.
- If we write `Time_axis='230.0 -20.0 3.0 1.0 10.0'`, with start and end values in reverse order, we get Figure 5.13b.

For a second example we first change the unit of simulation time into hours by

writing `OneDay=24.0` in the Listing 5.7 competition model[12]. In the plotting section we then write

```
DEFINE_AXIS Time_axis = '-20.0 260.0 3.0 1.0 10.0 January 2010'
ASSIGN_AXIS Time_axis > TIME
CURVE XVAR=TIME; Frame='TimePlot'; TLABEL='$ID=#WDAY #DAY$'; ...
      YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
```

and show the result in Figure 5.13c. The start of the axis at −20 hours is now at December, 31 of the previous year. The first label position is X5=10.0, ten hours after the start, which is clearly the first of january. We get a nicer axis, beginning and ending at midnight, by reversing again the start and end values (`Time_axis='260.0 -20.0 3.0 1.0 10.0'`). Figure 5.13d shows the result.

As a final example we change the simulated time to minutes by `OneDay=1440.0` and write

```
DEFINE_AXIS Time_axis = '-20.0 230.0 1.0 0.50 10.0 1-January-2010'
ASSIGN_AXIS Time_axis > TIME
CURVE XVAR=TIME; Frame='TimePlot'; TLABEL='$LH=#HOUR:#MIN$'; ...
      YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
```

The result is shown in Figure 5.13e, with Figure 5.13f again the result for reversed interval bounds. We made use here of the fact that an hour axis can be subdivided by tickmarks and/or labels. Note also that the plotted axis is *not an interval axis* but contains time labels at interval boundaries (the axis type is "$LH"). By specifying the minutes in each label[13] ambiguity is avoided.

Time labels could be defined as a full time specification with year, month, day and time, but this usually requires more room than there is. Therefore, the period as a whole can be described in the axis text, as in the previous example. This axis text, however, will clearly not change automatically if another StartYear is chosen. Therefore, the consistency between calendar connection, axis labels and axis text remains the modelers responsibility.

5.2.11 Plotting simulated time

If you want to use simulated time in a plot (the time variable between STTIME and FINTIM), the use of just TIME works only as long as there is no calendar connection. With a calendar connection a plotted TIME is converted to automatically calandar time as we have seen in the previous sections.

If you want to use the simulated time in a plot, also *with* a calendar connection, you have to plot a copy of TIME. This has been done in Listing 5.8 which produces the plot in Figure 5.14.

The copy of time is named SimTIME and will proceed from STTIME to FINTIM during the simulation, also with a calendar connection. The variable TIME does as well, but only in calculations. On output, values of TIME are converted to calendar times.

[12] This implies another *unit of time* for all model equations. In case of a real model we would have to change all parameter values accordingly. Here we do not do that, which means that the model remains the same numerically, with identical results, now in hours instead of in days.

[13] The colon between the hours and the minutes is part in the label format in the TLabel keyword.

Listing 5.8 Simulated time (between STTIME and FINTIM) can always be plotted, also with calendar connection, by using a copy of TIME and treat that copy as any other variable. The resulting TimePlot is shown in Figure 5.14.

```
TITLE Lotka-Volterra competition
MODEL
  INITIAL
  DYNAMIC
! the state variables
  X1 = INTGRL (IX1,RX1)
  X2 = INTGRL (IX2,RX2)
  INCON IX1 = 100.0
  INCON IX2 = 100.0
! the growth rates
  RX1 = RGR1 * X1
  RX2 = RGR2 * X2
! relative growth rates are reduced by competition
  RGR1 = A1 * (1.0 - X1/K11 - X2/K12)
  RGR2 = A2 * (1.0 - X2/K22 - X1/K21)
! parameters
  PARAMETER A1=0.1 ; A2=0.2          ! maximum relative growth rates
  PARAMETER K11=1000.0 ; K22=2000.0  ! carrying capacities
  PARAMETER K12=4000.0 ; K21=2500.0  ! the competition parameters
! simulation control
  TIMER STTIME=0.0 ; FINTIM=200.0 ; DELT=0.1 ; PRDEL=5.0
  TRANSLATION_GENERAL DRIVER='RKDRIV'
! calendar connection
  TRANSLATION_GENERAL StartYear=2010 ; StartDOY=1.0 ; OneDay=1.0
! plotting
  SimTIME = TIME  ! a copy of time is used in the plots
  DEFINE_AXIS STIMaxis = '0.0 200.0 40.0 10.0 0.0 time (days)'
  ASSIGN_AXIS STIMaxis > SimTIME
  ASSIGN_AXIS X1axis > X1
  ASSIGN_AXIS X2axis > X2
  DEFINE_AXIS X1axis = '0.0 1000.0 250.0  50.0 0.0 population 1 (#)'
  DEFINE_AXIS X2axis = '0.0 2000.0 500.0 100.0 0.0 population 2 (#)'
  CURVE XVAR=SimTIME ; FRAME='TimePlot' ; ...
        YVAR=X1; CURCOL='Red' ; CURTYP=1; CURWID=3.0; Legend='number X1'
  CURVE YVAR=X2; CURCOL='Blue'; CURTYP=1; CURWID=3.0; Legend='number X2'
END
```

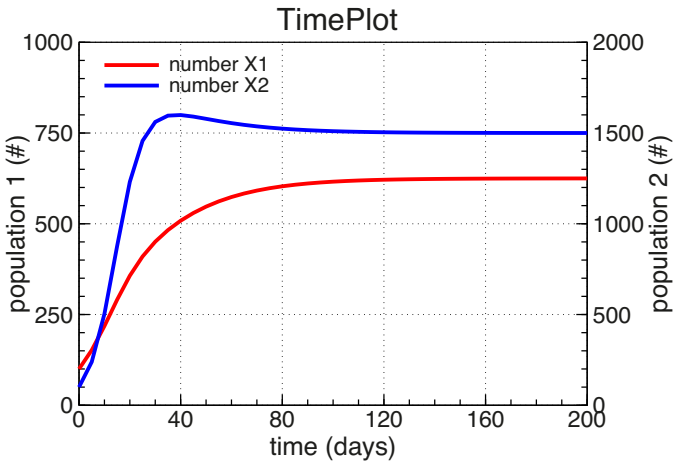


Figure 5.14. Result of the plotting statements in Listing 5.8.

5.3 How it works

There will be some overlap between the material in this older section and the newly written plotting course in the previous section 5.2. This section covers also things like plotting array variables and changing the plotting style.

5.3.1 Overview

If one or more CURVE statements are found, a number of actions takes place. At first the FST translator generates two additional files. A file named 'FSTplot.dat' contains a description of the requested plots.

The second file generated is 'PlotPreferences.dat' which contains plotting style options. This file can be edited by the user in order to change style options of the plots (see section 6 at page 69). Once a file 'PlotPreferences.dat' is present in the model directory, it is not overwritten by subsequent runs of the FST translator. This means that user changes are preserved until the preferences file is renamed, moved away or deleted.

Besides generating these two files, the FST translator inserts output statements in the generated Fortran code through which the requested data values are written to three binary files during the simulation. *After completion of all runs* the requested plots are produced by combining the binary data, the plot description in 'FSTplot.dat' and the preferences in 'PlotPreferences.dat'.

5.3.2 The CURVE statement

A plot (or frame) is defined by means of one or several CURVE statements. Each CURVE statements plots an additional variable in the frame. A CURVE may actually be a curve, formed by connecting (x,y) value points, or the (x,y) points are drawn as markers.

The structure of a CURVE statement is similar to the structure of many other FST statements. CURVE attributes are defined by means of expressions in the form *keyword=value*. The possible keywords and their meaning are.

XVAR The name of the "x-variable", just as a variable name and without quotes, e.g. XVAR=DevelopmentStage. The default XVAR variable is the one from the previous CURVE statement. If there is no previous CURVE statement, TIME is used.

YVAR The name of the "y-variable", just as a string and without quotes, e.g. YVAR=TotalDryWeight

LEGEND The legend text for the curve as text in quotes, e.g. LEGEND='simulated dry weight (g/m2)'.

FRAME The name of the plot in quotes, e.g. FRAME='crop weights'. The default FRAME name is the one from the previous CURVE statement. Hence, if this attribute is omitted, the curve is included in the previous plot. In practice, a plot is defined as a group of CURVE statements, the first one defining the plot name by means of the FRAME attribute.

CURTYP The type of curve as an integer value in [0,7]. See Appendix A. The default is 1 which is a drawn curve. If a marker type is specified, however, the default is 0, which means that no curve is drawn.

CURWID The width of the curve in points. A point is 1/72 inch, which is about 0.035 cm.

CURCOL The color of the curve defined by means of the name of the color in quotes. See Appendix C. The default color is 'black'.

MARTYP The type of marker a an integer value in [0,15]. See Appendix B

MARSIZ Half the size of the markers in centimeter.

MARCOL The color of the markers defined by means of the name of the color in quotes. See Appendix C. The default color is 'black'.

TLABEL This controls the format of a calendar time axis, as explained in section 5.2.10 and in section 5.4

FRAMETYPE This has default value 1 for an ordinary plot and has value 2 for a so called sensitivity plot in which the results of all runs are combined. In the special case that each run produces a single, scalar (x,y) datapair, and FrameType=2, the instructions for plotting markers and/or curves (CurTyp, MarTyp, CurCol, etc.) are applied to the collection of these (x,y) values. All runs together produce then a single curve.

Unlike many other FST statements, CURVE statements cannot normally be placed in arbitrary order[14]. We suggest that the CURVE statements defining a plot (or frame) are kept together in order to make changes easily.

Note that final curve widths and marker sizes are all affected by the scaling of the plot, which is defined in 'PlotPreferences.dat' (see section 5.3). By default, this preference file defines the size of the horizontal axis as 14 cm and the size of the vertical axis as 10 cm, *to which a default scale factor of 0.80 is applied*. A scale factor of 0.50 yields publication quality plots with vertical axis size of 5 cm[15].

5.3.3 User defined axes

Usually, a frame (or plot) contains various curves representing related variables, like different weights, different concentrations, etc. These variables should refer to a common axis with an appropriate text along it.

`DEFINE_AXIS WeightAxis='8 52 10 2 10 dry weight (gram/m2)'` defines axis WeightAxis starting at value 8.0, ending at 52.0, with a label distance of 10.0, a tick distance of 2.0, and the first label at value 10.0. The text along the axis is 'dry weight (gram/m2)'.

An axis is assigned to one or more plotted variables by means of a statement like
`ASSIGN_AXIS WeightAxis > LeafWeight, StemWeight, TotalWeight`

[14]Unless every CURVE statement contains a FRAME attribute, the result will depend on the order of the CURVE statements.

[15]Since the plots are written as vector graphics, there is in principle no need for size reduction. In principle, plot size could be set at its final value of, say, 5 cm with a scale factor of 1.00. Nevertheless, the result of a larger plot size in combination with a modest size reduction is easier to judge on screen and usually simplifies the choice of curve widths and marker sizes.

Listing 5.9 Plotting statements defining a frame with two curves, each with its own vertical axes. Figure 5.15 on page 60 shows the resulting plot.

```
CURVE XVAR=TIME ; YVAR=PRED ; CURCOL='Red'  ; Frame = 'Predator-Prey'
CURVE XVAR=TIME ; YVAR=PREY ; CURCOL='Blue'
DEFINE_AXIS PREDaxis='0.0  120.0   20.0  10.0 0.0 predator(#)'
DEFINE_AXIS PREYaxis='0.0 3500.0 1000.0 500.0 0.0 prey(#)'
ASSIGN_AXIS PREYaxis > PREY
ASSIGN_AXIS PREDaxis > PRED
! initial values and parameters
INCON IPREY = 200.0 ; IPRED = 15.0
PARAMETER R = 0.5; A = 0.01; C = 10.0; B = 0.02; D = 0.1; K = 2500.
! dynamic calculations
RPREY = R * PREY * (1.0 - PREY/K) - FOUND * PRED
RPRED = (B * FOUND - D) * PRED
FOUND = C * (1.0 - EXP (-1.0 * A * PREY / C))
PRED = INTGRL (IPRED,RPRED)
PREY = INTGRL (IPREY,RPREY)
! control
TIMER  STTIME=0.0; FINTIM=200.0; DELT=1.0 ; PRDEL=0.5
TRANSLATION_GENERAL DRIVER='RKDRIV' ; EPS=0.001
END
```

This statement assigns the axis WeightAxis to the listed variables. Note that an axis is always assigned to variables, *and not to a frame*. Each frame automatically contains all the axes needed for the curves plotted in it.

Axis names can be chosen by the modeler, as long as the usual rules for variable names are followed. There is one exception, however. If an axis is assigned to variable TIME, its name *must be* Time_Axis.

Unlike other variable types, an axis assigned to one or several variables does not need to be defined. If left undefined, a default axis will be generated which spans all the variable values involved. An axis definition may also contain only an axis text and no values. This is useful if the value range is (yet) unknown.

Listing 5.9 shows an example with two user-defined axes. Figure 5.15 shows the resulting plot.

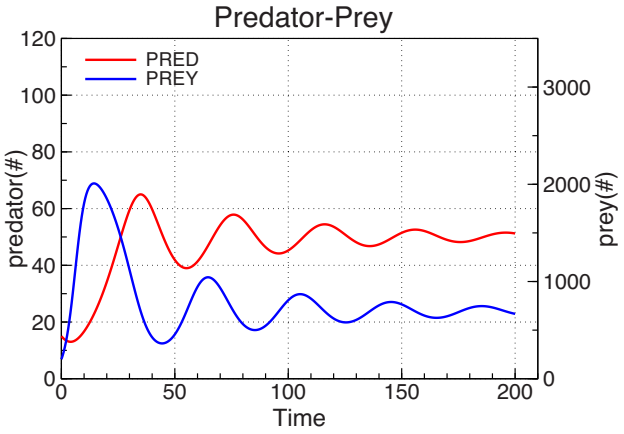


Figure 5.15. Frame with two vertical axes. The variable names are used as legend text, since the CURVE statements in Listing 5.9 do not specify legend texts. Note that no time axis was defined. It has been created automatically.

5.3.4 Which data is actually plotted?

This seems an easy question, but it is not. In an FST model we have initial calculations, dynamic calculations, terminal calculations and event sections. And all these variables can serve as either XVAR or YVAR in curve statements[16]. So what is actually plotted?

It is helpful to understand that initial variables are sent to output initially, terminal variables after the simulation, variables defined in event sections are sent to output *only at event time* and the ordinary dynamic variables are sent to output periodically with time interval PRDEL.

If you inspect a RES.DAT output file with tabulated output[17] you may notice that the dynamic variables also get values at start time (just after initialization), at all event times and at terminal time. So dynamic variables are "always" there[18], but the other ones not.

The rule

The general rule is that the x and y values of a plotted pair must refer to the same moment in time. Hence, if (XVAR,YVAR) is a pair of dynamic variables, valid (x,y) pairs will be there at periodic output times, event times and terminal.

An initially calculated variable (or a parameter) will be sent to output only once, and there will be just a single, initial (x,y) point available if the initial variable is plotted as function of a dynamic variable. In combination with an event variable, there may be no valid (x,y) pair at all.

Events

Also event variables[19] can be plotted. Since an event may take place at arbitrary moments in time, event output will not normally coincide with periodic "PRDEL output". In order to allow event variables to be plotted as function of dynamic variables, the statements for periodic output are executed at event time as well.

So in case of an event, additional dynamic output takes place just before *and just after* the event. This means that each event leads to two additional values for all dynamic variables, at the same moment in time, but not necessarily with the same values. This may lead to "jumps" in the graph, as a result of the discontinuities taking place at event time.

An event variable is calculated in an event section on the basis of parameters and other variable values *just before the event takes place*. Therefore, the compatible dynamic values are also those just before the event takes place. A plot will therefore contain only (x,y) combinations just before the event.

Example

In the model of Listing 5.10 (Leffelaar, pers. comm.) at page 63, the event variable MegaDose is plotted as function of time. Results are shown in Figure 5.16. Indeed, MegaDose values appear at the event times only. Therefore, MegaDose has been

[16] For all plotted variables holds that the position of the curve statement is not significant. Only the section of definition determines when, and how often, the plotted variable is sent to output.
[17] Such a file is made with PRINT statements.
[18] This includes state variables and SETTINGs that may be changed in events.
[19] Event variables are variables defined in an event section.

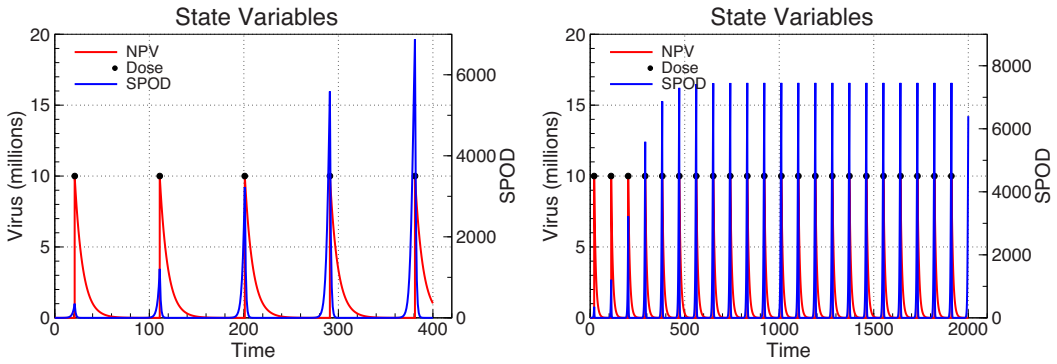


Figure 5.16. Results of the model in Listing 5.10. The left hand plot shows a simulation over 400 time units (FINTIM=400.0) with 5 events.The right hand plot has been created by setting FINTIM=2000.0

plotted as a series of markers. The state variables are defined at all times and have been drawn as curves[20].

Note that the definition of variable MegaDose in Listing 5.10 appears after the NewValue statements. The FST translator, however, guarantees that all event calculations are carried out *before* any Newvalue statement is executed. The position of event calculations within the event section is therefore arbitrary and not significant.

Exceptions to the rule

Also to the above rule there are exceptions. The first one will be discussed in section 5.5. An initially calculated array of values can be used in combination with a dynamic array in order to plot, for instance, concentrations at a series of fixed positions. The condition is that the re-used values are defined once and only once.

The second exception refers to a scalar (X,Y) pair defined at different moments of the simulation. Such a value pair can be plotted by means of a sensitivity plot (a FrameType=2 plot). The X and Y variable must be scalar and *each of them is sent to output once and only once*. This holds for instance for a parameter X and a terminal Y, or a variable defined in a one-time event. Listing 5.6 with Figure 5.10 at page 50 in section 5.2.8 provides a detailed example.

5.4 Calendar time

The topic of a calendar time axis has been extensively treated in section 5.2.10 of the plotting course. Various examples were presented there and Figures 5.12 and 5.13 at pages 52 and 55 give a impression of the possibilities.

The plotting of a calendar axis is important since a clear and practical time axis is a great help in comparing model results with measured values (cf. section 8).

[20]The idea of the model is that a more frequent application of the virus keeps the disease level at a lower value. Alternatively, a state event may be used to trigger virus application if the disease level exceeds a threshold.

Listing 5.10 This model (Leffelaar, pers. comm.) describes the control of a crop disease by means of application of a virus in a time event. Figure 5.16 at page 62 shows the resulting plot. See the text for comments.

```
TITLE Spodoptera exigua and NPV

! plotting
CURVE Frame = 'State Variables' ; XVAR=TIME ; ...
      YVAR=MegaNPV ; CURTYP=1; CURWID=2.0; CURCOL='Red'  ; Legend='NPV'
CURVE YVAR=MegaDose; MARTYP=8; MARSIZ=0.1; MARCOL='Black'; Legend='Dose'
CURVE YVAR=SPOD    ; CURTYP=1; CURWID=2.0; CURCOL='Blue' ; Legend='SPOD'
DEFINE_AXIS VirusAxis = '0.0 20.0 5.0 1.0 0.0 Virus (millions)'
ASSIGN_AXIS VirusAxis > MegaNPV, MegaDose

INITIAL
CONSTANT Miljoen=1.0E+6
! Initial values state variables
INCON INPV=0.   ; ISPOD=0.01
! Parameters related to the virus
PARAM LIMNPV=3.E5
PARAM TCNPVU=10. ; TCNPVL=50.
PARAM DOSE  =1.E7
! Parameters related to the pest
PARAM MXSPOD=1.E4; RGR=0.5; RDR=1.0
! Parameters related to management
PARAM First =21. ; Period = 90.
! The Cumulative Dose of NPV is also calcutated.
SET CumDose = 0.

DYNAMIC
! State variables
NPV     = INTGRL(INPV , RDEGR)
SPOD    = INTGRL(ISPOD, RSPOD)
MegaNPV = NPV / Miljoen

EVENT
!  Spraying the NPV virus for the "First" time and thereafter with
!  intervals "Period". The amount of NPV virus is contained in Dose.
   FIRSTTIME StTime + First
   NEXTTIME  Time + Period
   NEWVALUE  NPV     = NPV + Dose
   NEWVALUE  CumDose = CumDose + Dose

!  defined in event section means plotted at events only
   MegaDose = Dose / Miljoen
ENDEVENT

! Rates of degradation of NPV density due to Ultra-Violet Light (TCNPVU)
! and decrease of NPV density due to Growth and Shedding of leaves (TCNPVL)
RDEGR = -(NPV/TCNPVL + NPV/TCNPVU)
! Rate of change of Spodoptera Exigua
GRSPO =  RGR*SPOD*(1.0-SPOD/MXSPOD)
DRSPO = -RDR*SPOD
RSPOD =  INSW(NPV-LIMNPV, GRSPO, DRSPO)
! Simulation control
TIMER STTIME=0.0 ; FINTIM=2000.0 ; PRDEL=1.0 ; DELT=0.1
TRANSLATION_GENERAL DRIVER='RKDRIV'
END
```

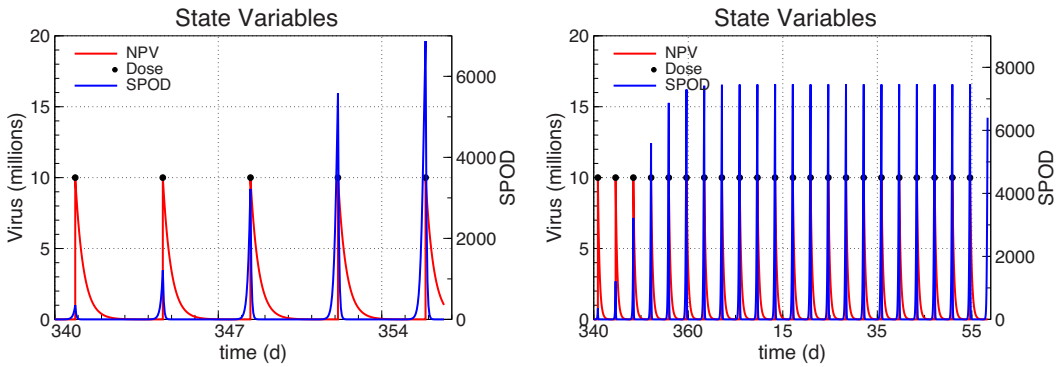


Figure 5.17. Results of the model in Listing 5.10 after connecting simulated time to calendar time, starting at day 340 of 2010 and assuming the hour as unit of time. See also text for comments.

Therefore we use here another example to illustrate a few points, mainly with respect to the axes produced automatically under various circumstances.

The model in Listing 5.10 is connected to calendar time by adding to the model `TRANSLATION_GENERAL StartYear=2010; StartDOY=340.0; OneDay=24.0`. The value of OneDay implies an hour as the unit of time[21]. Running the model with merely this addition (and another time with FINTIM=2000.0) produces the plot in Figure 5.17.

A few remarks need to be made on the horizontal axis, the time axis of the plots.

- Time values along the time axis have been replaced by Day_Of_Year values between 1 and 365 (366 for a leap year).
- The value of TIME in the model not affected by a calendar connection, and is still simulated between start time STTIME and finish time FINTIM. Only the time axis in the plot is affected.
- For the relatively small number of days in the left hand plot of Figure 5.17, the day numbers are placed in the middle of the day they refer to. The numbers do not label a moment in time, but a time interval.
- For a large number of days (right hand plot), FST switches to the traditional method and the day numbers are written at the beginning of the day they refer to.

Depending on the time range of the simulation, FST plots a time axis in hours, days, months or years. The choice of FST may not always be satisfactory, in which case the user may want to overrule the default calendar time axis.

Changing the calendar time axis, however, is not entirely trivial. In section 5.2.10 various examples were given already. In the example below in section 5.4.1 we just show how to change the time axis in Figure 5.17.

It is advised to use the default first. Then a TLABEL keyword can be added to a CURVE statement belonging to the plot, and finally a Time_Axis might be defined for fine-tuning (see section 5.2.10).

[21] This correct unit of time is probably a day, but this is not of any concern in this manual.

Listing 5.11 Alternative plotting section for the model in Listing 5.10. Time has been connected to the calendar and a TLABEL has been added.

```
! plotting
CURVE Frame = 'State Variables' ; XVAR=TIME ; ...
      TLABEL='$IW=week #WeekNr #Year$'; ...
      YVAR=MegaNPV ; CURTYP=1; CURWID=2.0; CURCOL='Red'   ; Legend='NPV'
CURVE YVAR=MegaDose; MARTYP=8; MARSIZ=0.1; MARCOL='Black'; Legend='Dose'
CURVE YVAR=SPOD    ; CURTYP=1; CURWID=2.0; CURCOL='Blue' ; Legend='SPOD'
DEFINE_AXIS VirusAxis = '0.0 20.0 5.0 1.0 0.0 Virus (millions)'
ASSIGN_AXIS VirusAxis > MegaNPV, MegaDose
! calendar connection
TRANSLATION_GENERAL StartYear=2010 ; StartDOY=340.0 ; OneDay=24.0
```

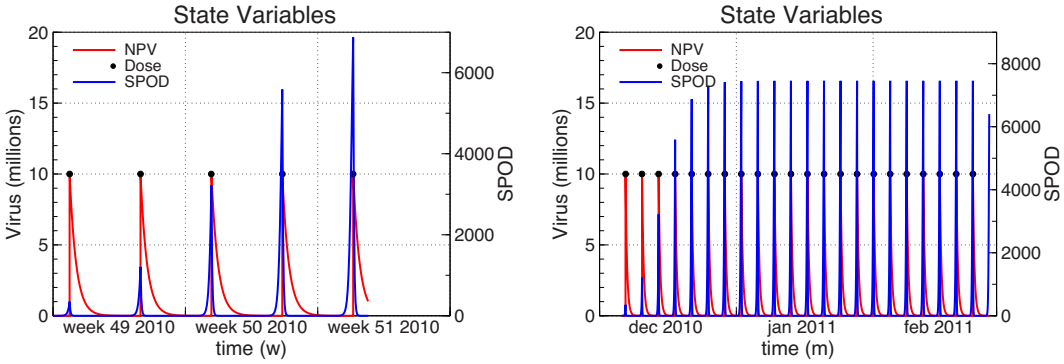


Figure 5.18. The same curves as in Figures 5.16 and 5.17, now with a calendar time axis defined in the model. See Listing 5.11 and the text for details.

5.4.1 Another calendar time axis example

The left hand plot in Figure 5.18 shows again the simulation results with the model in Listing 5.10 over 400 hours. Here, the calendar axis is subdivided in weeks, however, by means of the TLABEL string `'$IW=week #WeekNr #Year$'` in the first CURVE statement of Listing 5.11.

The week numbers in the labels are the ones defined by the ISO 8601 standard which is commonly used in diaries (http://en.wikipedia.org/wiki/Seven-day_week#Week_numbering and http://tuxgraphics.org/toolbox/calendar.html). Weeks always begin at monday. Most years have 52 weeks, some have 53.

The right hand plot in Figure 5.18 is again the result for 2000 simulated hours, using the TLABEL string `'$IM=#maandS #Year$'`. More examples can be found in section 5.2.10. The codes used in TLABEL strings are listed in Appendix D.

There are many possibilities for constructing a calendar time axis. For practical (crop growth) models which run over a season, a year or several years, it will be worthwhile to take some time and design a proper time axis, especially if measured values are included in the plots and discussed with field experts.

5.5 Plotting array variables

The XVAR and YVAR attribute of a CURVE may be the name of an array variable. There are obviously three cases:

scalar X, array Y. Each element of the Y-array is treated as a scalar variable and plotted as function of X. Hence, if the array size of Y is 12, there will be 12 curves.

array X, scalar Y. Each element of the X-array is treated as a scalar variable and Y is plotted as function of it. Hence, if the array size of X is 12, there will be 12 curves.

array X, array Y. The two arrays *must have equal length, say* N. For each output time (at which X and Y are both defined), a curve is plotted from the data pairs $(X(1), Y(1)), (X(2), Y(2)), \ldots, (X(N), Y(N))$.

The situation with two arrays often appears in case of a one dimensional grid of N values at positions $X(1), X(2), \ldots, X(N)$. Each curve then describes a spatial profile of Y, e.g. a concentration profile.

Typically, a grid is static and values defined on it (e.g. concentrations) change. Hence, the X values defining the grid do not change and the same values are used again and again. Therefore, an exception has been made to the requirement of equal output times for X and Y.

If only a single value of the array X is found, this value is accepted and re-used for all output times of Y. This allows the user to define the grid variable X in the initial section of the model, where it belongs *if the grid is static.*

5.6 Technical details

A running model actually writes the plots as EPS files. By the graphical user interface FSTwin these EPS files are converted into PDF files, with help of a call to Ghostscript. If you do not use FSTwin and instead use a batch file for translating, compiling and running the model, you may find the information below useful.

5.6.1 Processing EPS files

The generated files are readable ASCII files containing vector drawings in EPS (Encapsulated Postscript) format. The EPS code can be inspected using any text editor and the graphics can be converted using Ghostscript[22]. A suitable ghostscript command for converting "plot.eps" into "plot.pdf" is

```
C:\"Program Files"\gs\gs9.52\bin\gswin64c -dSAFER -dNOPAUSE -dBATCH
-dQUIET -dEPSCrop -sDEVICE=pdfwrite -dPDFSETTINGS=/printer -dNOPLATFONTS
-dCompatibilityLevel=1.4 -dMaxSubsetPct=100 -dSubsetFonts=true
-dEmbedAllFonts=true -sOutputFile="plot.pdf" -f "plot.eps"
```

Ghostscript can also be used to convert the EPS drawing into various bitmapped formats. Under Mac OSX, EPS files are opened by default in the program "Preview", which converts them as well.

[22] Ghostscript for Windows can be downloaded from https://github.com/ArtifexSoftware/ghostpdl-downloads/releases/download/gs9550/gs9550w64.exe. Ghostscript for Linux can be downloaded from https://github.com/ArtifexSoftware/ghostpdl-downloads/releases/download/gs9550/ghostscript-9.55.0-linux-x86_64.tgz. Ghostscript for Mac OSX has been prepared by Richard Koch primarily for use with LaTeX. It is dowloaded from https://pages.uoregon.edu/koch/Ghostscript-9.55-Full.pkg.

Actually including a graph in a publication may require some fine-tuning. The FST version number needs to be removed and you may wish to add additional text, equations or arrows to the figure. This requires the use of a vector graphics editor. A well known commercial one is Adobe Illustrator. A list of editors can be found at http://en.wikipedia.org/wiki/List_of_vector_graphics_editors.

Edited EPS files can be included in document preparation programs like LaTeX or Word, or may be sent to the publisher as separate files.

Sometimes printing problems can be prevented by making sure that the vector graphics does not contain fonts (Helvetica, Times, etc). This can be accomplished by converting all text to drawings using the "create outlines" command in the "Type" menu of Adobe Illustrator.

Adobe Illustrator also allows you to convert all colors (FST uses the RGB color model) into CMYK colors which is *sometimes* required for high quality printing, especially in combination with photographs. Modern digital printing machines, however, usually deliver good results from RGB files.

Many more suggestions on the preparation of professional artwork can be found at Cadmus Art Support (http://art.cadmus.com/da/index.jsp). Cadmus (used by for instance the PNAS journal) requires the use of Adobe Illustrator.

EPS files may also be converted into bitmapped formats like JPG or TIFF. Unless a very high resolution is used, however, this leads to loss of quality. Bitmaps can always be distinguished from vector drawings by zooming in. At some point, a bitmap will reveal its pixels.

5.6.2 Removing date, model name and FST version "by hand"

If you want to include the generated graphics quickly into a LaTeX or Word document, it may be a good idea to remove "by hand" the footer under the graph. There are two ways.

If you run FST models from a batch file there is an easy method. After translating the model you may edit the generated file "fstplot.dat". In that file you find the plotting instructions generated form the CURVE statements. The various plots are listed in a table like structure with column headers "Plot", "Mode", "TimeAxisCommand" and "WriteFooter". Prior to running the model you change the WriteFooter entries into ".false.".

If you use FSTwin and you do not have access to an EPS or PDF graphical editor you may just edit the text of the EPS file.

- Open the generated EPS file in a simple editor like Notepad (on Windows) or TextEdit (on Mac OSX).
- Look for the line "%%EndPageSetup" around line 65 of the file.
- A few lines further down you see a line containing the date, the model name, the filename and the FST version number ending with "la".
- This line can be deleted or (better) de-activated by putting a single "%" sign in front of it.
- Save the file.

Now the plot has no longer the version message below it.

5.6.3 Limitations

The maximum number of frames created by an FST model is 40 and the maximum number of CURVE statements is 500. There are no other practical limits.

Many thousands of (x,y) values are plotted almost instantly. Nevertheless, one should be careful not to use a too small value of PRDEL. This does not improve curve smoothness and just leads to large EPS files, large documents, slow editing and slow conversion to other graphical file types.

Chapter 6

The plotting style file

Listing 6.1 contains the default style file PlotPreferences.dat as it is generated by the FST translator. The file may be edited and is *not overwritten* by the translator as long as it remains there. Hence, after deleting the style file, FST creates a new, default one during the next translator run.

The meaning of all style variables is explained in the file itself. Therefore just a few remarks are made.

6.1 Plot size

It is important to realize that PaperSize, XLLcm and YLLcm are in absolute centimeters and that all other sizes (plot size itself, pen widths, font size, axis separation distance) are scaled using the scale factor "Scale". Hence, this scale factor is applied to all curve widths and marker sizes specified in the CURVE statements of the model.

Sometimes a plot becomes unusually large as a result of a larger Scale, a large DefaultXsize or DefaultYsize, by adding many additional axes, or by values plotted outside the frame. In this case it may be necessary to increase "paper size" by adapting PaperSizeXcm and/or PaperSizeYcm in the PlotPreferences file.

6.2 Thin lines at label positions

By default thin dashed lines are drawn at the position of axis labels. These thin lines ("long labels") are helpful in technical plots and during model development. You can get rid of them by setting the variable "LongLabels" at ".false.".

6.3 Plot title and legend

Similarly, the inclusion of the plot title on top of the plot can be suppressed by setting "WriteTitle" at ".false." and the position of the legend text can be changed by means of the variables "XposLegend" and "YposLegend". These, however, are *relative* positions in the frame of the plot. An X-position above 1.00 leads to a legend at the right hand side of the plot.

Listing 6.1 File PlotPreferences.dat with style options. This file can be edited. A default file is generated by FST after removing the existing file from the working directory.

```
! By editing this file you change the style of plots created in subsequent
! model runs. By deleting this file you force the FST translator to create
! a new copy containing the default values.

! General output options
! ======================
ScreenMessages  = .false.   ! to suppress most screen output
FileForEachPage = .false.   ! to get each page on a separate file
WriteTitle      = .true.    ! plot name on top of each frame

! Document and plot size
! ======================
PaperSizeXcm = 18.0     ! document size, does not depend on Scale
PaperSizeYcm = 11.5     !
XLLcm        =  2.5     ! Position of Lower-Left corner of plot
YLLcm        =  2.0     ! (fixed, not scaled)
Scale        = 0.80     ! plot scale factor (size, fonts, pen widths, distances)
DefaultXsize = 14.0     ! X axis in cm (will be scaled)
DefaultYsize = 10.0     ! Y axis in cm (will be scaled)

! Fonts
! =====
TitleFont    = 24       ! Helvetica font sizes in pixels (will be scaled)
AxisTextFont = 20       !
LegendFont   = 16       !
LabelFont    = 18       !

! Line widths in pixel
! ====================
PenWidthAxis    = 1.0   ! pixel (will be scaled)
PenWidthLabels  = 1.0   !
PenWidthTicks   = 1.0   !
PenWidthMarkers = 0.8   !

! Label and tick mark options
! ===========================
LongTicks          = .false. ! grid of long dashed tickmarks
PenWidthLongTicks  = 0.4     ! pixel
LongLabels         = .true.  ! grid of long dashed labels
PenWidthLongLabels = 0.6     ! pixel
TickLineLength     = 0.16    ! cm. A negative value means the ticks/labels
LabelLineLength    = 0.30    ! are plotted to the outside of the plot.

! Additional axes
! ===============
AddXaxesOnTop    = .true. ! additional X-axes added above or below graph
AddXaxisOffset   = 0.0    ! distance between graph edge and 2-nd X-axis
AddXaxisDistance = 2.0    ! distance between additional X-axes
AddYaxisOffset   = 0.0    ! distance between right graph edge and 2-nd Y-axis
AddYaxisDistance = 2.6    ! distance between additional Y-axes

! Plot name and legend
! ====================
XposLegend  = 0.05        ! (-) legend X-position relative to primary X-axis
YposLegend  = 0.94        ! (-) legend Y-position relative to primary Y-axis
```

6.4 Footer

If you want to get rid of the footer (the date, model name and FST version number in small type below the plot), you have either to edit the plot in a graphical editor like Adobe Illustrator or use one of the methods described in section 5.6.2 at page 67. It cannot be done by means of the preferences file.

Chapter 7

Calendar connection

7.1 Introduction

The FSE mode of the translator is most often used for crop simulation. The FSE mode requires the day as unit of time and all rates of change to be expressed as amounts per day. The FSE mode requires the specification of WEATHER data. Therefore, in FSE mode, there is always a connection between simulated time and calendar time.

The TRANSLATION_GENERAL mode of FST does not imply a certain unit of time. Equations may be dimensionless or in convenient time units, depending on the problem. The addition of WEATHER is possible, also in GENERAL mode. *By doing just that*, however, the start time STTIME suddenly becomes a Day-Of-Year value (between 1.000 and 366.000 for a non-leap year), and the unit of simulated time must be a day.

Hence, in FST 2, the use of WEATHER in general mode eliminated the possibility of a convenient unit of simulated time and convenient values of STTIME and FINTIM. And *without* a WEATHER statement[1], simulated time is in unspecified units, there is no connection between simulated time and calendar time and access to measured data would be impossible.

The solution is the addition of a few optional TRANSLATION_GENERAL variables, which explicitly define a connection between simulated time and calendar time.

7.2 Connecting the calendar

Since FST 3, a method exists for connecting calendar time to the simulation in GENERAL mode. Three additional TRANSLATION_GENERAL variables are introduced:

StartYear. The year at which the simulation starts.

StartDOY. The Day-Of-Year (between 1.0000 and 366.0000 for a non-leap year) at which the simulation starts.

OneDay. The length of a day in model time units. For example, "OneDay=86400.0" means the model is in seconds, "OneDay=1440.0" means the

[1] In FSE mode a WEATHER statement is obligatory.

model in in minutes, “OneDay=24.0” means hours and “OneDay=0.1” means decades as the unit of time of the model equations.

The combination of StartYear and StartDOY defines the start time as a calendar time. The FST translator identifies the value of STTIME with this calendar time and from that moment on, the simulated Time (starting at STTIME and ending at FINTIM), is connected to the calendar using the value of OneDay. Hence, by defining these three TRANSLATION_GENERAL variables, the *simulated time becomes connected to the calendar time.*

The advantage of this method over the old method of FSE 2 is that the user is free to choose convenient values of STTIME and FINTIM, needed for other aspects of the model. The choice is independent of the simulated calendar interval and there is no implied unit of time.

Another advantage is that a calendar connection does not require a WEATHER statement anymore. Using subroutines, other types of input data become possible. A WEATHER statement is still valid, however.

7.3 Calendar connection with WEATHER

Like in FST 2, weather data are made available through a WEATHER statement. There is just one problem: The start year StartYear is also defined by the WEATHER variable IYEAR. Therefore, a program explicitly defining a calendar connection with StartYear, StartDOY and OneDay, must not contain IYEAR anymore.

Without the three new variables, however, the old method still works. A WEATHER statement implies “StartYear=IYEAR”, “StartDOY=STTIME” and “OneDay=1.0” and the generated datafile TIMER.DAT contains just that.

7.4 The available calendar variables

A calendar connection in GENERAL mode, either explicitly (by means of the new TRANSLATION_GENERAL variables), or implicitly (by WEATHER use) makes available the following variables as “driver-supplied variables”:

iDOY The current Day-Of-Year as an integer variable, in the range $[1, \ldots, 365]$ for a non-leap year and $[1, \ldots, 366]$ for a leap year.

DOY The current Day-Of-Year as a real number (with a fractional part).

Year The current year number as a real variable.

ClockTime The clocktime as a real value between 0.0 and 23.99999.

SimDays Simulated time sofar, a real value in days (with a fractional part).

iHourOfDay The hour number of the day, as an integer value in the range $[1, \ldots, 24]$.

FractSec Fractional seconds reading of a digital clock as a real number in $[0.00, \ldots, 1.00)$.

ClockSec The integer seconds reading of a digital clock, in $[0, \ldots, 59]$.

ClockMin The integer minutes reading of a digital clock, in $[0, \ldots, 59]$.

ClockHour The integer hours reading of a digital clock, in $[0, \ldots, 23]$.

ClockDay The calendar day as an integer, in $[0, \ldots, 31]$.

ClockMonth The calendar month as an integer value, in $[1, \ldots, 12]$.

ClockYear The integer year reading of a digital clock, in value equal to the variable Year.

Although the variables DOY, Year, ClockTime, SimDays and FractSec are single precision real values, the internal timing calculations of the simulation driver take place in double precision arithmetic. The driver-supplied clock variables are therefore accurate, also in case of a simulated time interval of thousands of days (many years).

From these variables only iDOY, DOY and Year are available in FSE mode[2]. In FSE mode the calendar is connected by means of the required WEATHER statement.

Note that the integer calendar variables can directly be used to select an array element by means of the ELEMNT function, like in

```
! example of the use of an integer calendar variable
  DECLARATIONS
  ARRAY Values
  ...
  MODEL
  PARAMETER MonthValues(1:6) = 2.0 ; MonthValues(6:N) = 3.0
  INITIAL
  ARRAY_SIZE N=12
  X = ELEMNT (Values, ClockMonth)
  ...
  END
```

The ELEMNT function is an intrinsic FST array function described in Rappoldt & van Kraalingen (1996, section 4.3.4.2).

7.5 Referring to StartYear, StartDOY and OneDay

The new control variables StartYear, StartDOY and OneDay can be referenced in calculations. The first two are integer variables and OneDay is a real variable. The use of these variables requires an explicit definition in a TRANSLATION_GENERAL statement, however. An implicit calendar connection with just a WEATHER statement, does not allow references to StartYear, StartDOY and OneDay.

This limitation should not be a problem in practice since a calendar connection with WEATHER is most likely to occur in older FST models, in which the new variables are not used anyway. For newly written programs, an explicit connection is the preferred method.

[2]The WEATHER variable iYear is the start year of the simulation and cannot be referenced in an FST model. The current year of the simulation is available as the REAL variable Year, which may be converted to an integer by NINT(Year). In general mode the integer variable ClockYear can be used.

Chapter 8

Measured variables

8.1 Introduction

Measured variables in a simulation model serve two purposes:

- Comparison of simulated values with measured values. This can be simply done by plotting both the simulated and measured variable in a single frame, or by calculating some sort of average deviation between the two.
- Sometimes the modeler wants to drive the simulation with measured values. For instance, a biological process depends on temperature and a measured series of temperatures is used to drive the simulation.

The use of weather data in a crop model is a good example. In FST weather data can be accessed through the WEATHER statement. The WEATHER statement, however, works for daily weather data only and not for other types of data.

The new FST statements MEASUREMENTS and MEASURED represent a more generic mechanism for accessing data from within an FST model.

Usually, files containing measured variables have the structure of spreadsheets with columns for date and time, and with one or more columns with measured variables. This is precisely the type of file which can now be used in FST. In principle, the modeler simply has to specify the name of the file, the name of the data and time columns, and the name of the columns containing the variables to be used.

8.2 Example model with measured data

Table 8.1 shows a portion of the spreadsheet type of file which is used in the examp[le model of Listing 8.1. Clearly, the FST translator must "know" the name of the datafile, the name of the date and time columns and the names of the columns which are to be used as dynamic variables in the calculations.

If there are large time gaps or missing data (denoted by a hyphen), FST uses linear interpolation between the provided values. Further, FST will not perform simulation steps outside the data range for which the data is provided. Hence, the simulated time must be part of the time spanned by the measured data.

This requires of course a calendar connection which is made in Listing 8.1 by means of the first Translation_General statement which sets the start time at the beginning of day 340 of 2008 and sets the hour as the unit of simulated time.

Table 8.1. Portion of the spreadsheet like datafile 'sample.txt' which is used in the FST model of Listing 8.1. The data consists of columns with a header containing the names. The file is a comma- or tab-delimited text file. Note the format of the date and time columns. They are separate columns and the time may contain seconds as well. Date and time formats can be changed in spreadsheet programs like NeoOffice or Microsoft Excel.

date	time	Toutside	abuist003	aCO202	rCO201	wCO201
2008/12/01	00:05	4.4	44	773	524	443
2008/12/01	00:10	4.5	43	765	524	448
2008/12/01	00:15	4.5	45	745	528	454
2008/12/01	00:20	4.4	48	748	522	456
2008/12/01	00:25	4.4	43	728	521	454
2008/12/01	00:30	4.5	44	679	521	445
2008/12/01	00:35	4.6	43	634	513	447
2008/12/01	00:40	4.5	43	-	513	452
2008/12/01	00:45	4.5	41	-	514	450
2008/12/01	00:50	4.6	40	-	514	447
2008/12/01	00:55	4.7	42	586	513	436
2008/12/01	01:00	4.8	42	585	516	443
2008/12/01	01:05	4.7	42	528	517	452
2008/12/01	01:10	4.7	42	520	522	465
...	...	...	...	...	...	...

The measured variables can be used in all model sections since they are defined for all time values occurring in the simulation. They can also be plotted which is a convenient way of inspecting the data (e.g. the example output plot in Figure 8.1 at page 80).

8.3 The input file

The MEASUREMENTS statement can be used to specify the following variables

Datafile The name of the datafile as a string, e.g. Datafile='MyData.txt'. The file needs to be a plain text file with the structure of a spreadsheet: columns with a name on top. There may be comment lines starting with an exclamation mark "!" or an asterisk "*".

DateColumn The name of the column containing the date, e.g. DateColumn='Datum'. This is an optional variable. If not specified, FST assumes there is a column named 'Date'.

TimeColumn The name of the column containing the time, e.g. DateColumn='Tijd'. This is an optional variable. If not specified, FST assumes there is a column named 'Time'.

Delimited This is an optional integer variable describing the way in which columns are delimited. Possible values are 1 (TAB-delimited), 2 (Comma-delimited) or 3 (Space-delimited). If the Delimitted variable is omitted, or if a zero value is given, FST tries to find out using word separation in the first non-comment line.

In many cases only a file name will be required and the statement will look like `MEASUREMENTS Datafile='MyData.txt'`.

Listing 8.1 The use of measured data in a model requires spreadsheet like input file with columns for date, time and the desired input variables. Figure 8.1 at page 80 shows the result of the program below. A part of the datafile 'sample.txt' is shown in Table 8.1.

```
Title Use of Measured data

! - Example file 'sample.txt' contains many columns, just two of which are
!   needed in this program, Toutside and rCO201 (see Measured statement).
! - There must be a date and time column, otherwise FST cannot couple
!   the data to simulated time.
! - The delimited code is set at 0 (unknown). FST tries to find out how
!   the columns are separated. Sometimes you may have to choose
!   1 (TAB-delimited), 2 (Comma-delimited) or 3 (Space(s) only).
! - There are missing data in the example file, the separating TAB's are
!   present, however, so FST can distinguish between the columns.

MEASUREMENTS Datafile = 'sample.txt' ; DateColumn = 'Date' ; ...
             TimeColumn = 'Time' ; Delimited = 0

! - There must be a calendar connection. Otherwise there is no
!   connection between datafile time and simulated time.
!   The model TIME in this example is in hours since Oneday=24.0.
TRANSLATION_GENERAL StartYear=2008 ; StartDOY=340.0 ; OneDay=24.0

! just two variables are needed here
MEASURED Toutside, rCO201

! measured data can be used in calculations
ToutsidePlus5 = Toutside + 5.0

! measured data can be used in plotting
DEFINE_AXIS Tax = 'Temperature (oC)'
DEFINE_AXIS Cax = 'CO2 (ppm)'
ASSIGN_AXIS Tax > Toutside, ToutsidePlus5 ! the temperatures share an axis
ASSIGN_AXIS Cax > rCO201
! three curves in a plot
CURVE XVAR = TIME; YVAR = Toutside; Frame = 'Measured' ; ...
      Legend = 'Outside Temp'; TLABEL='$ID=#Wday #DAY-#MONTHST-#Year$'
CURVE XVAR = TIME; YVAR = ToutsidePlus5 ; CURTYP = 5 ; CURWID=1.0 ; ...
      Legend = 'Outside Temp + 5'
CURVE YVAR = rCO201; CURCOL='Red'

TRANSLATION_GENERAL TRACE=1 ; DRIVER='EUDRIV'
TIMER STTIME=0.0 ; FINTIM=48.0 ; DELT=0.01 ; PRDEL=0.01  !  48 hours !
END
```

A datafile is usually constructed by exporting a sheet as a TXT or CSV file from a spreadsheet program. As field delimiter a tab, space or comma can be used. Text fields should not be delimited, however, so there should not be quotes around the column names.

Note that the Date and Time columns must be two separate columns. The format of the Date column is one of the TTUTIL formats. Usually either yyyy/mm/dd or dd-mmm-yyyy, for example 1994/08/31 or 3-sep-1994. Unfortunately, the common format dd/mm/yyyy *cannot* be used and files containing it have to be changed[1].

[1]This is most easily done by importing the file in a spreadsheet program, change the format of the date column and re-export the file as a text file.

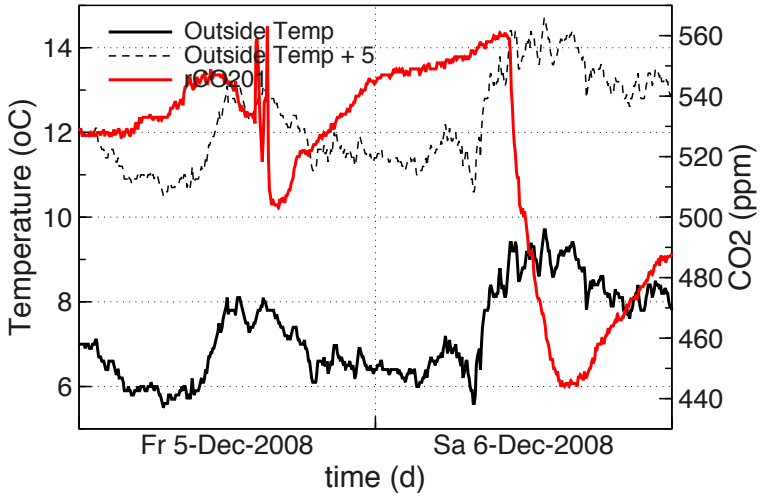


Figure 8.1. The Measured statement in Listing 8.1 makes available the columns 'Toutside', 'rCO201' in datafile Sample.txt. The data can be used as any other defined variable, in calculations and in Curve statements for plotting. The model in Listing 8.1 simulates just two days. Plots like these can be used to check and inspect the data.

The time column must have the usual format hh:mm:ss. Seconds may be omitted. Fractional seconds may be added.

Missing values must be marked as a hyphen "-". Missing values may occur for measured variables, but not in the date and time columns. During model translation, FST checks the datafile.

Simulated time must be contained in the time range spanned by the date and time columns of the input file.

8.4 Getting the measured variables

The names of the variables to be used in the model are simply listed in a MEASURED statement, e.g. "`MEASURED CO2,Oxygen`'. The FST translator treats these variables as defined by the MEASURED statement. On model execution, time

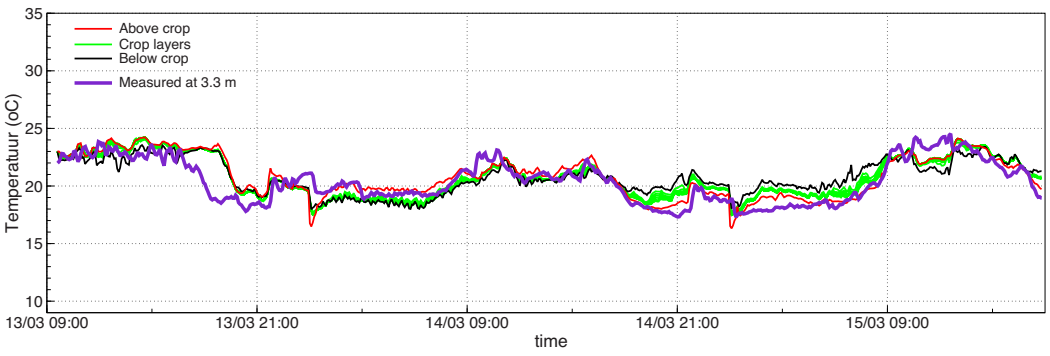


Figure 8.2. Plot of simulated temperatures (above the crop, a number of crop layers and below the crop), and temperatures measured at 3.3 m. The plot has been made with the EcoCurves model for crop growth and greenhouse climate.

Listing 8.2 Statements producing the graph in Figure 8.2, in which measured and simulated temperatures are compared. Produced with the EcoCurves model for greenhouse climate. Note that variable CropT is an array variable plotted as a number of green curves.

```
! this reads the data
MEASUREMENTS Datafile='temp.txt'
MEASURED TempM

! plot all temperatures in a single frame
CURVE  Frame='Temperature' ; TLABEL='$LH=#Day/#Month #Hour:#Minutes$'; ...
       YVAR=Above ; CurCol='Red'    ; Legend = 'Above crop'
CURVE  YVAR=CropT ; CurCol='Green'  ; Legend = 'Crop layers'
CURVE  YVAR=Below ; CurCol='Black'  ; Legend = 'Below crop'
CURVE  YVAR=TempM ; CurCol='Purple' ; Legend = 'Measured at 3.3 m'

! temperature axis
DEFINE_AXIS TempAxis = '9.0 35.0 5.0 1.0 10.0 Temperature (oC)'
ASSIGN_AXIS TempAxis > Above, CropT, Below, Temperature
```

interpolated values are supplied to the model (linear interpolation between successive data). This obviously requires a calendar connection, either explicitly (see Chapter 7), or by means of a WEATHER statement.

Measured variables may be used as ordinary scalar variables in all calculations, including those in event sections, and may be plotted by means of CURVE statements.

8.5 Example from practice

Figure 8.2 shows several simulated greenhouse temperatures and a single measured temperature. The plot originates from calculations with the "Explorer Kasklimaat" model of EcoCurves.

This is a large applied model, around 20,000 lines of code, which simulates crop growth, crop development and greenhouse climate. It is an FST model[2], written in close cooperation with Plant-Dynamics. The FST facilities for measured data, plotting and event handling are heavily used.

Listing 8.2 shows the statements required for constructing a plot like the one in Figure 8.2. The plot shown is wide, which can be realized by setting a different plot size in PlotPreferences.dat (cf. section 5.3).

[2]Large portions of the model have been written as linked Fortran-2003 modules.

Chapter 9

Other changes

9.1 Syntax

In FST 3 and FST 4, the following changes have been made to the syntax checking procedures of the translator:

The length of variable names has been increased to 31 characters, in accordance with the Fortran standard (Metcalf *et al.*, 2004; Chapman, 2008).

Warnings on the use of lowercase characters have been removed from the translator. Character case is not significant, however. This means that the same variable may occur in the program in various combinations of lowercase and uppercase characters.

A program line may be up to 132 characters long, including the continuation code "`...`", which has been left unchanged.

Statements with a asterisk "`*`" at the first position or statements beginning with an exclamation mark "`!`" in any position are treated as comment statements.

An FST statement may be terminated by an exclamation mark followed by comment text. This is especially useful in parameter statements for adding a unit and description to the defined parameter (e.g. Listing 4.1 at page 29).

The use of lowercase characters is a matter of taste and style. Lowercase characters allow names as "VelocityX" or "MolarVolume" which are usually considered to be more readable than names like MOLARVOLUME. It may be a good idea to keep the FST keywords themselves in uppercase, which leads to statements like

```
! example of the use of lowercase variable names
  CONSTANT MolarVolume = 22.4
  PARAMETER Height = 2.0 ; Width = 3.0
  INCON InitialVelocityX = 3.4 ; InitialVelocityY = 5.6
```

Assignments in the generated Fortran will contain the variable names as they appear in the FST calculation statements. The first occurrence of a variable in FST is used in the generation of declarations, datafiles and variable listings. An exception is the use of FUNCTION names in AFGEN function calls. For technical reasons, these names appear in uppercase in the generated Fortran.

By combining a Sensitivity statement with a sensitivity plot in the same model, you can automatically do a sensitivity analysis on a model parameter. Examples can be found as Listing 5.6 at page 49 and Listing 4.1 at page 29.

9.2 New intrinsic functions

9.2.1 The intrinsic function SimulationTime

The intrinsic function SimulationTime converts a Date/Time specification into a value of the simulation time between STTIME and FINTIM. This clearly requires a calendar connection, either explicitly by setting StartYear, StartDOY and OneDay in TRANSLATION_GENERAL mode, or implicitly by means of the WEATHER statement in FSE mode[1].

The arguments of SimulationTime are six *integer* values, variables or expressions, Year, Month, Day, Hour, Minute, Second, in this order. For example, if the time interval between Jan-15 and June-1 of the start year is needed in a simulation as variable DeltaT, this variable can be calculated in the INITIAL section as

```
! example of SimulationTime function call
  deltaT = SimulationTime(iYear,6,1,0,0,0) - SimulationTime(iYear,1,15,0,0,0)
```

The value of DeltaT will be in days in TRANSLATION_FSE mode, but DeltaT will be in other time units in TRANSLATION_GENERAL mode, depending on the value of OneDay in the calendar connection. And of course, in leap years the period will be one day longer since February, 29 lies within the specified period.

A more interesting application of the function SimulationTime is the calculation of the time of a time event (see Chapter 4). The statement below calculates the first event time as 15-May-2008 13:14:17 plus an additional X time units.

```
! a calculated time in a FirstTime statement
  EVENT
    FirstTime SimulationTime(2008,5,15,13,14,17) + X
    ...
  ENDEVENT
```

The construction below uses an event date as model parameter. For just a date (and a fixed time 00:00:00, say) we need three numbers. A parameter array Date is declared with array size 3. The three elements are converted into integer numbers in the SimulationTime call. Here is the code:

```
! a date as a model parameter
ARRAY Date(1:ND)
...
ARRAY_SIZE ND=3
PARAMETER Date(1:2)=2008.0, 5.0 ; Date(3:ND)=15.0
...
EVENT
  FirstTime SimulationTime(NINT(Date(1)),NINT(Date(2)),NINT(Date(3)), 0,0,0)
  ...
```

This construction allows reruns on calendar dates at which an event takes place!

9.2.2 The intrinsic functions SUM and DOT_PRODUCT

The Fortran-95 intrinsic functions SUM and DOT_PRODUCT (Metcalf *et al.*, 2004; Chapman, 2008) have been added to the list of supported Fortran intrinsics. SUM

[1] In TRANSLATION_FSE mode the start year is given as IYEAR in the required WEATHER statement, the value of STTIME is the start value of the calendar DOY (Day Of Year) and the unit of time is always one day.

accepts a single array argument and DOT_PRODUCT requires two arguments declared as arrays with identical upper and lower bounds.

Explicit array bounds in calls to SUM and DOT_PRODUCT are not supported by FST, however. If you want that, you have to use the FST intrinsic functions ARSUMM and ARIMPR respectively. SUM and DOT_PRODUCT have been added for increased speed when the entire arrays need to be summed or multiplied.

9.2.3 Other new intrinsic functions

Other additions to the list of supported Fortran intrinsic functions are CEILING, FLOOR, AMAX0, AMAX1, AMIN0, AMIN1 and FLOAT. The definition of these functions can be found in descriptions of the Fortran language.

9.3 String arguments of subroutines and functions

Fortran subroutines could always be called from FST for doing standardized calculations or for calculations which are impossible, inefficient or clumsy in the FST language itself. The arguments of a Fortran subroutine or function (see next section) should be declared using a declaration statement in the DECLARATIONS section of the program.

The declaration allows the translator to determine which of the actual arguments in the call(s) are defined in the subroutine and which ones are merely used. This is important for determining the order of the calculations[2].

In FST 3, the string constant STRING is introduced as an input argument type, in addition to integer, real scalar, and real array input. Only string constants are allowed, e.g. `'For calculating A'`. String expressions or string variables are not allowed in FST. String arguments allow subprograms to generate meaningful messages if something goes wrong. An example is given in the next section 9.4.

9.4 User defined functions

User-defined functions are treated in the same way as subroutines. Function calls can be written directly in calculation statements, which may lead to more elegant programs than the use of subroutines in combination with intermediate variables. Limitations, however, are that functions must return a single precision, scalar, real variable and that all function arguments must be input arguments.

An example is the declaration of a function MichaelisMenten in the following way:

```
! example of the use of lowercase variable names
  DECLARATIONS
  DEFINE_FUNCTION MichaelisMenten (STRING, INPUT, INPUT, INPUT)
  ...
  MODEL
  Xloss = MichaelisMenten ('Reaction 1', Vmax1, K1, X) + ...
          MichaelisMenten ('Reaction 2', Vmax2, K2, X)
  ...
  END
```

[2] In the generated Fortran (not in FST), a variable must be defined before it can be used.

The DEFINE_FUNCTION statement declares four input arguments, a string constant and three real values, variables or function calls. In the calls to Michaelis-Menten you see that the real arguments are the Vmax, the "half rate concentration" K and the substrate concentration X. Note that the same Michaelis-Menten expression (in the actual Fortran function) is used for two different reactions.

The string constant in the calls can be used by the function itself for meaningful messages in case there is something wrong. In case of negative concentrations X, for instance, the function might force the program to stop, but without further information the user does not even know which function call *and which concentration* caused the problem. This is solved by including the string in an error message written from the function.

9.5 Appended Fortran subprograms

Fortran subroutines and functions can be appended to the FST model. In principle, the translator copies them to the generated Fortran file and leaves it to the compiler to check the Fortran code. However, the appended Fortran must be either in free source form or in fixed source form (Metcalf *et al.*, 2004; Chapman, 2008). A mixture of the two is not allowed and would not make sense since Fortran compilers do not like a mixed form file either.

This implies that the translator has to decide in which of the two forms the Fortran code was written. The translator may fail to do this properly, unless the FST user knows the rules of the game. So what are the rules?

- The Fortran source form is determined using the first non-empty line after the STOP statement.
- If this line begins with an exclamation mark '`!`' (at any position), the source form is set to free. The line is considered as a Fortran-95 style comment statement.
- If the line begins with a asterisk '`*`', a '`c`' or a '`C`' at the first position, the source form is set to fixed. The line is considered as a Fortran-77 style comment statement.
- Otherwise, if the first character is at position 7 or later, the source form is assumed to be fixed. If the position of the first character is in [1,6], the source form is assumed to be free.

For the FST part of the model, the use of an asterisk or exclamation mark in comment lines does not matter. After the STOP statement, however, you should be aware of the difference.

Sometimes these simple rules may lead to a surprise, for instance, if your first Fortran-95 subroutine happens to start at position 7, the source form is classified as fixed. Problems are easily solved by adding a comment in the proper source form, on top of the Fortran code.

Once the source form is determined, all statements, continuation lines and comment statements should conform to *either the fixed or the free* source form. If the subprograms are in fixed source form, for instance, exclamation marks as comment characters are not allowed. If your subprograms are in free source form, the Fortran-77 method of statement continuation or the asterisk "`*`" at the beginning of a comment line is not allowed.

Note that in the Fortran 95/2003 standard, the fixed source is an obsolescent feature of the language, a candidate for deletion in future versions of the standard.

9.5.1 Number of subroutine and function arguments

The translator checks the number of arguments in SUBROUTINE and FUNCTION statements against the FST declaration of the subprogram. It does so by finding the first words of non-comment statements. If the first word happens to be SUBROUTINE or FUNCTION, the arguments are counted.

Note that this will not always work for functions, since functions may begin in Fortran like "REAL FUNCTION MichaelisMenten". In this case, the function statement will not be found and the number of arguments will not be verified. Verification is also impossible if the called Fortran programs are separately compiled and then linked with the model, or if they are part of a precompiled object library. Hence, the user must always be aware of possible problems with the argument list of linked subprograms. Such problems usually lead to a runtime crash. What is always verified, however, is the consistency between the DEFINE_CALL or DEFINE_FUNCTION declaration in FST and the actual calls.

We further remark that FST does not support the use of keyword arguments or optional arguments in subprogram calls[3]. This implies that Fortran-95 subprograms requiring such a call cannot be used directly from FST. In such a situation the user will have to write a trivial interface routine that translates the simple Fortran-77 style call from FST into the desired Fortran-95 call. The interface routine then contains a USE statement for a Fortran-95 module or an explicit interface description for the Fortran-95 subprogram.

9.5.2 What does the translator do with Fortran?

Fortran in free source form, is copied to the generated source file completely unchanged. This implies that modern Fortran-95 modules can be appended to FST code without any problem[4].

Fortran in fixed source form, is converted to free source form by changing the continuation and the comment lines. All other characteristics of the old Fortran-77 routines are left untouched like the use of numbers as statement labels. So what you basically get, is free source form Fortran beginning at position 7.

The conversion is necessary since the model itself is generated by the translator in free source form and compilers tend to complain about source form mixtures in the same file.

The user is advised to replace the original Fortran by the converted Fortran copied from Model.f90[5]. Alternatively, Michael Metcalf's program[6] can be used for a more

[3] In Fortran 95 a subroutine SUB with a dummy argument A can be called using 'call SUB(A=myvar)', where 'myvar' is the actual argument in the calling program. Keyword arguments in calls are especially useful in case of optional arguments, which do not *have to* be there. All this, however, requires the subroutine interface to be known at compile time in the calling program, either through a USE statement or through an explicit interface declaration. Subprogram calls from FST are Fortran-77 style calls without keyword arguments.

[4] But they cannot be called directly from FST, see section 9.5.1.

[5] A Fortran section beginning with a subroutine statement at position 7 may then be incorrectly classified as fixed form. This can be repaired by adding a comment line beginning with an exclamation mark on top of the Fortran code.

[6] Available from many websites, e.g. ftp.numerical.rl.ac.uk/pub/MandR/convert.f90 or

complete conversion.

9.5.3 The Fixed/Free form

There is a source form which can be combined with both free and fixed source form Fortran source files. This is the Fixed/Free source form (Metcalf *et al.*, 2004; Chapman, 2008) with continued lines coded with an ampersand (`&`) at position 73 *and* any character at position 6 of the continuation line. If your Fortran is in this form, the addition of an exclamation mark comment line on top of the Fortran section will cause the translator to switch to free form. It will otherwise complain about a line length above 72.

The Fixed/Free source form may lead to error messages on continued SUBROUTINE or FUNCTION statements for which the translator attempts to determine the number of arguments. In such a case, the fixed/free form continuation should be removed from these statements. For other statements the Fixed/Free source form should not be a problem.

9.6 Minor changes

A warning used to be given on the use of a scalar variable as an array argument in a function or subroutine call. The warning told the user that the array length appearing in the subroutine is 1. This situation has been changed into an error condition since (1) In Fortran-95 there is a distinction between degenerate arrays[7] and scalars, (2) Usually it *was* an error (a forgotten array declaration) and (3) Arrays with length 1 can be declared if necessary.

The OUTPUT statement for generating matrix printer plots is no longer maintained. We do not test it anymore, it may not function properly and we intend to completely remove it from the FST language. If there are users who cannot do without, please let us know.

A few subprograms linked with the generated Fortran, have been moved from the utility library TTUTIL to the drivers library. The moved subprograms are TIMER2, INTGRL, INSW, FCNSW, LIMIT, LINT2 and CHKTSK. The files have been converted to Fortran-90 free format and CHKTSK has been adapted to the new event handling ITASK=5 section of FSE models.

Finally, numerous small improvements in the text of error messages and warnings have been made.

http://www.nag.co.uk/nagware/Examples/convert.f90.

[7] A degenerate array is an array with length 1.

References

Bouman, B. A. M., Kropff, M. J., Tuong, T. P., Woperies, M. C. S., ten Berge, H. F. M., van Laar, H. H., 2001. ORYZA2000: modeling lowland rice. Technical report, International Rice Research Institute and Wageningen University and Research Centre, Los Baños (Philippines) and Wageningen. 235 pp.

Chapman, S. J., 2008. Fortran 95/2003 for scientists and engineers. MaxGraw-Hill, New York.

Goudriaan, J., van Laar, H. H., 1994. Modelling potential crop growth processes. Textbook with exercises. Current Issues in Production Ecology, Volume 2. Kluwer Academic Publishers, Dordrecht.

Metcalf, M., Reid, J., Cohen, M., 2004. Fortran 95/2003 explained. Oxford University Press, Oxford.

Rappoldt, C., van Kraalingen, D. W. G., 1996. The Fortran Simulation Translator FST version 2.0. Technical report, DLO Research Inistitute for Agrobiology and Soil fertility; The C.T.de Wit graduate school for Production Ecology, Wageningen, the Netherlands. Quantitative Approaches in Systems Analysis No. 5.

Yin, X., van Laar, H. H., 2005. Crop System Dynamics. Wageningen Academic Publishers, Wageningen. 155 pp.

Appendices

Appendix A

Curve types

Figure A.1 shows the dashing pattern of the various curve types for two different curve widths. Note that a scale factor of 0.60 was applied while including this figure into this document.

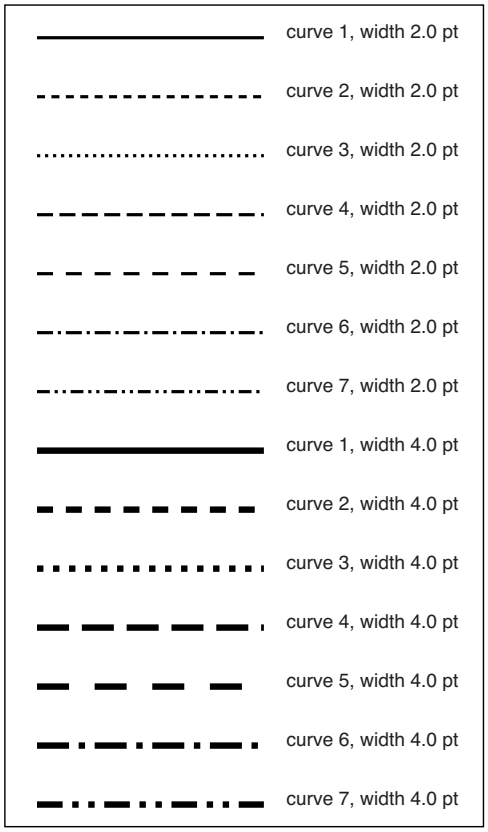


Figure A.1. Curve types for two different values of the curve width. Note that this figure has been scaled with a factor 0.60 while including it into this document.

Appendix B

Marker types

Figure B.1 shows the various marker types for two different marker sizes. The used colors are "Blue" and "DeepPink" and "ForestGreen" (cf. Appendix C). Note that a scale factor of 0.60 was applied while including the figure into this document.

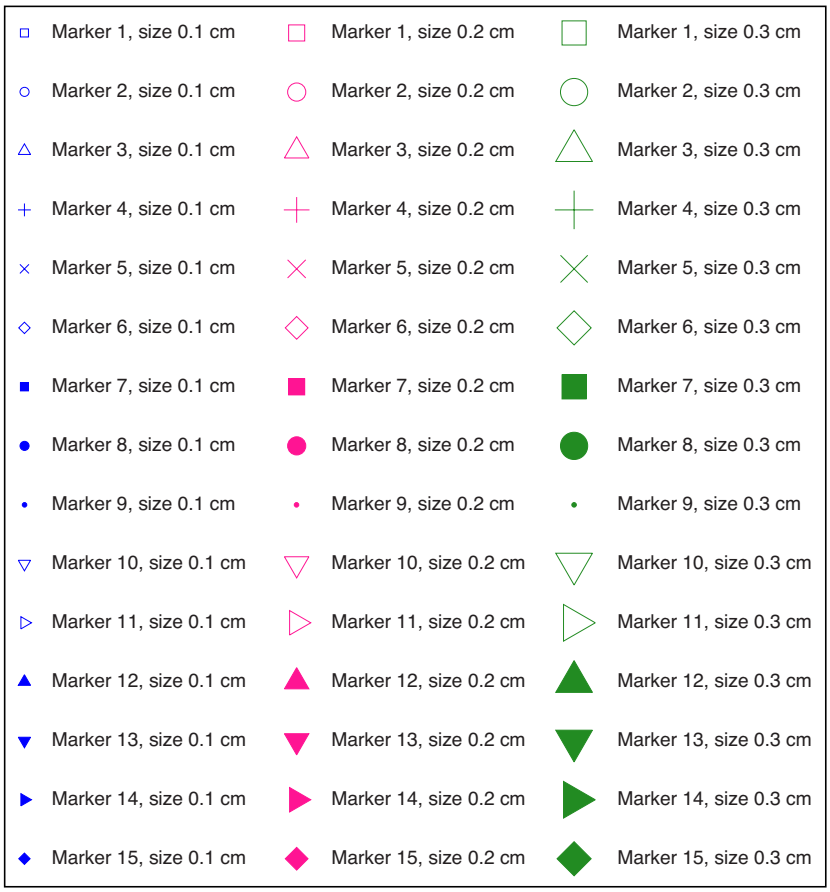


Figure B.1. Marker types at two sizes. Note that this figure has been scaled with a factor 0.60 while including it into this document. See the text for some remarks.

Some remarks:

- The unscaled size of the markers is about two times the value of the markersize CURVE attribute MARSIZ.
- Marker 9 always has an unscaled size of 1 mm. It does not react on MARSIZ values.
- There are filled markers with a solid color (e.g. markers 7 and 8) and open, non-filled markers (e.g. markers 1,2 and 3). The open markers are not transparent, but white.
- The line width used to draw the open markers can be changed in PlotPreferences.dat, a file which is generated on the model directory by the FST translator and which is *not* overwritten as long it is kept in place (cf. section 5.3).
- The line width used to draw the open markers does not depend on the size of the markers.

Appendix C

Curve and Marker colors

Figure C.1 shows the possible curve and marker colors with their names. The use of these names is not case sensitive.

Figure C.1. Color names to be used in CURVE statements.

Appendix D

Time label strings

Time label strings are assigned to the TLABEL attribute of a CURVE. A time label string controls how a calendar time axis looks like. The string is constructed as follows

- A TLABEL string consists of a description of the axis type and a time label in the form "$ *AxisType* = *TimeLabel* $".
- The *AxisType* consists of 2 characters, immediately following the opening $-sign. The first one is an "I" for Interval or a "L" for Label. This describes the position of the time labels at either the centre of a time interval or at a point in time.
- The second character selects the time interval to be used for subdividing the axis: "H" for hours, "D" for minutes, "W" for weeks, "M" for months and "Y" for years.
- The text between the =-sign and the trailing $-sign defines the *TimeLabel*. This definition makes use of certain standard strings which are replaced during plotting by calendar time attributes, numbers or names describing the calendar time. Other characters are just copied to each label.
- Table D.2 lists the possible calendar time attributes that can be used in the time label definition.

In Table D.1 some example label strings are given with a typical result. TLABEL strings are among the few things which are *not thoroughly verified* by the FST translator. Incorrect strings may therefore lead to runtime errors *after completion of all model runs*.

Table D.1. Examples of calendar time labels. Note that text which is not recognized as a standard calendar time attribute (see the list in Table D.2) is simply copied to the label. For instance a space, a hyphen, a comma or a word like "week".

time label	typical result
#Hours:#Minutes:#Seconds	12:34:20
week #WeekNR, #Year	week 21, 2010
#Wdag #Day-#MaandST-#Qyy	wo 19-feb-'11
#MaandLT #Year	september 2008

Table D.2. Calendar time attributes which may be used in time label strings. The codes all start with # and are replaced by the actual values for the calendar time label or interval. In case of an interval its midpoint is used for deriving the values. The use of these attributes is *not case sensitive*.

Attribute	Meaning
#Year	The year number, e.g. 2011
#Month	The month number from 1 to 12
#Day	The day in month between 1 and 31
#Hours	The current hour between 0 and 23
#Minutes	The minutes on the clock between 0 and 59
#Seconds	The seconds on the clock between 0 and 59
#FSeconds	The fractional seconds, should be preceded by #Seconds
#MonthST	Short month name like Jan, Feb, Mar
#MonthLT	Full month name like January, February, March
#MaandST	Short name in dutch like jan, feb, mar
#MaandLT	Full name in dutch, like januari, februari, maart
#WeekDay	Monday, Tuesday, Wednesday, ...
#WeekDag	maandag, dinsdag, woensdag, ...
#WeekNR	ISO 8601 week number
#Wday	Short week day, like Mo, Tu, We, Th
#Wdag	Short week day in dutch, ma, di, wo
#Qyy	Year number with century replaced by a quote, like '11
#DOY	Day of year between 1 and 365 (366 for a leap year)

Note that an interval axis requires a different type of label than a traditional point label axis. It is useless, for instance to describe a week interval as the clocktime 12:00:00 at its centre. A week number, possibly in combination with a year number, is a more meaningful choice. Some trial and error will usually be required for getting an axis right.

Appendix E

FST version History

This version history contains additions and bug fixes, not in a particular order.

Version 4.16

- Bug resolved in driver subroutine MeasuredVariables which caused an error for simulated times within 1 second from the time range of the data.
- Bug resolved in the treatment of the Sensitivity statement. Sensitivity runs can now correctly be made on the variables controlling the cyclic use of measured data (CycleStYear, CycleStMonth, CycleStDay, CyclePeriodInDays).
- InTranslation_General mode: Default values for timer variables STTIME and FINTIM, and for Translation_General variables StartYear and StartDOY as derived from measured data, are written to file with higher accuracy, thus preventing small timing errors.
- Command line options for the FST translator must be provided fully. A string with just the first few characters of the option is no longer recognised.
- The options start with a hyphen "-" on all platforms OSX, Windows and Linux.
- Command FST ? or FST -help returns the full list of options and describes how the name of the model file can be given.
- Spaces in the FST filename are allowed. The filename must then be between "double quotes".
- The FST model file must have extension .fst.
- Bug resolved in the treatment of free format Fortran statements beginning in column 1.
- For INTGRL statements with array arguments the warning on expansion in a do-loop is suppressed.
- Warning on absent weather file no longer cut off at 80 characters.
- Warning on the generation of PlotPreferences.dat has been removed.
- Maximum number of plots (i.e. Frames) by means of CURVE statements is now 40.

- Maximum number of sorted sections is 50 and event-endevent sections is 47.
- List of forbidden subroutine names adapted. Subprograms inside a linked Fortran module are visible in the generated model only if the linked module is declared in a USE statement. Hence, names of subprograms inside modules of linked libraries are no longer forbidden if these modules are not USE'd in the actual model subroutine. For linked library modules being used, however, all contained public subprograms and public module variable names are forbidden as names of called subprograms in FST.
- Call ActivateLicense for the EcoCurves plotting facilities has been added to the generated Fortran source.
- New Sensitivity keyword IntVarying for sensitivity runs on INTEGER variables like StartYear. The range is still provided as real constants which are used on a nearest integer basis. Number of runs may be omitted. In that case all integer values in range are used.
- Fixed a bug preventing reruns on the name of a Measurements Datafile.
- StartDOY can now be outside [1.0,367.0]. In this case a sensitivity analysis can be done on StartDOY covering a time interval of multiple years. Note that StartDOY=0.0 means december,31 of the previous year (the year before StartYear).
- Line length in res.dat has been increased.
- Fixed a bug resulting in Internal error message on cyclic use of measured data.
- No output error message for models without CURVE or PRINT has been omitted since models can produce output in subroutines.
- Fixed a bug in calendar use without a CURVE statement in Translation_General mode.
- Fixed a bug by which a time event at FINTIM was executed twice.
- In Translation_General mode a PRDEL value of 0.0 now properly suppresses periodic output.
- In Translation_General mode default values for STTIME, FINTIM, StartDOY and StartYear are derived from a Measurements Datafile. See the chapter on Measured data for details.

Version 4.12

- Axis range and Firstlabel position in supplied Time_axis is no longer neglected. Bugs in calendar axis plotting resolved.
- Improved PlotPreferences.dat file.
- Bug fixed. A left Parenthesis as the start of an expression in ZeroCondition, FirstTime, NextTime and Finish statements is now properly recognized.
- Sensitivity statement added.
- Curve keyword FrameType added. FrameType=2 specifies a sensitivity plot in which the results from all runs are combined. In case of one scalar (x,y)

pair per run, the curve specification is applied to the curve formed by the (x,y) results of all runs.

- New supplied calendar variables iWeek, iDOW and TimeEOW.
- Adaptations to TTUTIL 4.25. Routine OUTDAT for the generation of tabular output ("RES.DAT") no longer uses a temporary file RES.BIN but stores all values in memory.
- FST capacity settings increased to 500 array declarations, 2000 symbols, 4000 cross references, 200 state variables, 500 substatements, 500 NewValue statements, 2000 actual subroutine/function arguments, 300 called subroutines/functions, 1000 declared subroutine/function arguments.
- Inline comments added.
- EVENT names introduced. Runtime event messages to the logfile now include the event name, which simplifies error search.
- EVENT section may be put in the INITIAL section in order to improve readability of the program if, for instance, an initial SETTING variable changes value in one or a few event sections.
- In the absence of terminal calculations, the calculations in event section(s) were incorrectly sorted. Bug resolved.
- Global check on NewValue statements contained bug for programs without state variables.
- An ARRAY_SIZE variable which is not used for declaring any arrays is no longer an error, just a warning (the variable may be used as an integer constant, which is legal now).
- CURVE, ASSIGN_AXIS, DEFINE_AXIS, MEASUREMENTS and MEASURED statements added to the language.
- Improved synchronization between periodic output and the time event times. Now, every time step the output times are checked for being an extremely small (double precision) time step away from the nearest time event time. In that case the output time is set equal to the time event time in order to prevent unnecessary tiny steps. This applies to the GENERAL mode only since FSE mode uses fixed interval time steps.
- A few FST WARNING texts have been corrected and shortened.
- Driver supplied variable RunNumber (in Translation_General mode only).
- Bug in FST/utility subroutine SearchBackOutsideString resolved. The bug caused in rare cases program line breaks inside quoted strings.
- Added WARNING if a ZeroCondition does not depend on dynamically calculated values. Useful especially if SETTINGs are used in event functions. Explenation: A SETTING may change value in (other) events. The zero crossing of an event function caused by changes of a SETTING, however, is not detected as a state event, since the sign change happens momentarily at event time, and not during an integration step.
- Incorrect driver name (RKDRIV) in event-related messages from EUDRIV corrected.

- FatalERR. The TTUTIL routine FatalERR now uses the TTutilPrefs module to look up the previously set error mode. FatalERR then behaves (1) as the old default, (2) as a special error routine (displaying a message stored in TTutilPrefs) or (3) writes the error also to an error file, e.g. "model_errors.txt". This means all three types of FatalERR are now combined into a single ttutil subroutine. FST no longer needs to be linked with a TTUTIL library without FatalERR.
- Afbreken van lange regels zonder spaties, operators en haakjes (zoals een heel lang call statement) ging niet goed omdat de zaak wordt afgehakt precies op kolom 132. Dat is gecorrigeerd door de comma toe te voegen als toegelaten character voor het afbreken.
- TTUTIL improved speed of UPPERC and IFINDC.
- Linear interpolation with AFGEN or CSPLIN has now been implemented in Fortran with help of Module Interpolation in which function name and slopes are buffered. As a result, interpolation calls are much faster now. LINT2 is no longer used.
- In GeneralDrivers the maximum number of state variables has been set at 50000 (state arrays A(N) count N times).

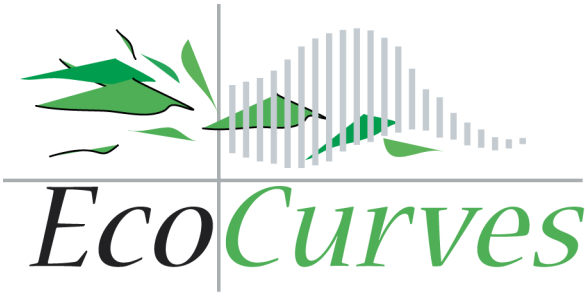
This document has been created in December 2021

The most recent version of this manual is available from
www.ecocurves.nl/Support/FST/FSTadditions.pdf

Part of the FSTwin installation is the FST translator which is provided by EcoCurves BV "as is" and without warranties. EcoCurves BV cannot accept any responsibility for errors leading to incorrect simulation results.

FSTwin is compatible with the GFortran compiler and with Ghostscript for viewing plots. The use of these third party programs requires that you comply with the terms and conditions for obtaining a valid license. This is solely your responsibility and the use of FST does not alter this in any way.

The FSTwin installation is accompanied by two separate installers, which can optionally be used to perform a standard installation of GFortran and/or Ghostscript. These are programs under the GNU General Public License, however, and you must comply with the terms and conditions of this license if you use GFortran and/or Ghostscript in combination with FST. This also applies if you use an existing installation of these programs or if you download newer versions.


EcoCurves