Simulation Reports CABO-TT

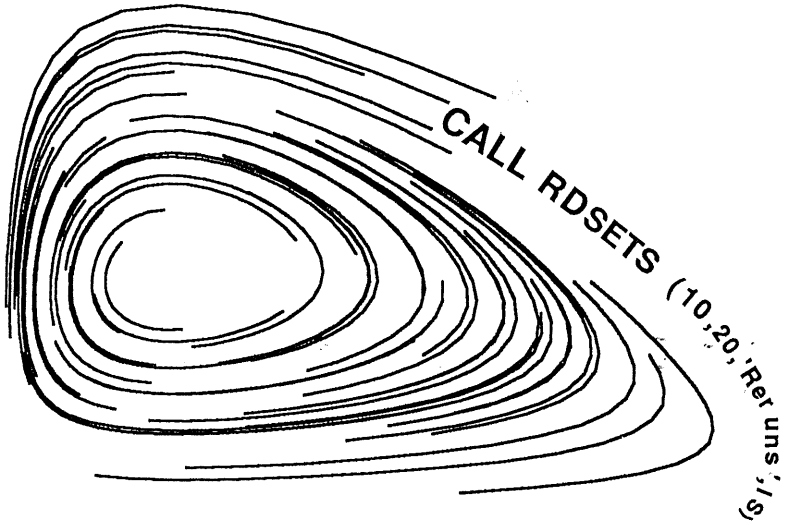
Reference manual of the

FORTRAN utility library TTUTIL

with applications

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

Simulation Report CABO-TT nr. 20



A joint publication of

Centre for Agrobiological Research (CABO-DLO)

and

Department of Theoretical Production Ecology (TPE), Agricultural University

Wageningen 1990

Simulation Reports CABO-TT

Simulation Reports CABO-TT is a series giving supplementary information on agricultural simulation models that have been published elsewhere. Knowledge of those publications will generally be necessary in order to be able to study this material.

Simulation Reports CABO-TT describe improvements of simulation models, new applications or translations of the programs into other computer languages. Manuscripts or suggestions should be submitted to:
H. van Keulen (CABO-DLO) or J. Goudriaan (TPE).

Simulation Reports CABO-TT are issued by CABO and TPE and they are available on request. Announcements of new reports will be issued regularly. Addresses of those who are interested in the announcements will be put on a mailing list on request.

Address

Simulation Reports CABO-TT
P.O. Box 14
6700 AA Wageningen
Netherlands

Authors affiliation

C. Rappoldt:[1]
Department of Theoretical Production Ecology
Agricultural University
P.O.Box 430
6700 AK Wageningen, The Netherlands

D.W.G. van Kraalingen:
Centre for Agrobiological Research (CABO-DLO)
P.O. Box 14
6700 AA Wageningen, The Netherlands

---

1 current address: Institute for Soil Fertility Research (IB-DLO)
P.O.Box 30003, 9750 RA Haren, The Netherlands.

Preface

There are some ever recurring problems associated with the use of FORTRAN programs. Reading data from files is often done using specific formats. That tends to make programs and data files difficult to adapt and to maintain. Furthermore, there are always problems programming output in a convenient form.

Over the last years we developed solutions for our own problems with respect to input, output, character strings and file handling. Each time we paid a little more attention than strictly necessary for the problem at hand. That has resulted in a slowly growing set of subroutines and functions that proved to be useful in almost any program. Most of these routines have gone through a number of revisions. We now use identical sources on VAX, IBM PC and Apple Macintosh. The set of 57 routines became our utility library TTUTIL. In addition to "pure utilities" the library also contains a number of routines that resulted from our work with simulation models.The source code of all routines is free. We hope that others will find the library a usefull tool in improving their programs.

This report contains all comment headers and parameter lists and may serve as a reference manual in using the library. The introductory chapters contain some background explanation and examples. In a separate chapter we made remarks on the use of FORTRAN in simulations models.

We thank Jan Goudriaan and Herman van Keulen for their comments on a draft of this report, and Willem Stol for his continuous help and his enthusiasm,

Wageningen, December 1990,

Kees Rappoldt
Daniel van Kraalingen

Contents

1. Introduction

This report describes a number of FORTRAN subroutines and functions that make up a utility library. Many routines are utilities in the sense that they do not make use of any mathematical or numerical method, do not contain measured data and do not depend on assumptions concerning some described system. Utilities simply perform their task with respect to input, output, string handling, file handling etc. They are tools for writing reliable and readable FORTRAN programs. That is also the reason why you will not find complete listings in this report. Only headers of all routines are given containing a description of their function.

Two groups of routines have a numerical character. These are the random number generators and the routines solving an initial value problem by means of the Runge-Kutta or Euler method. Further, a few utilities deal with often recurring calculations in numerical simulation models.

Chapter 2 describes the classification of the subroutines in more detail. This classification enables you to find your way through the library. It is followed by explanations on the use of three specific groups, the routines for interactive input, for reading data from file and for output to file. Chapter 4 is the manual part of the report. It consists of the comment headers of all library routines. Each header contains a list of arguments on which calls can be based. To prevent any ambiguity with respect to parameter types, also the declarations of the formal parameters have been included. In Chapter 5 you can find simple example programs that make use of various routines of the library.

Chapter 3 describes a method for writing simulation models in FORTRAN. The examples given are full-fledged programs for Euler integration and Runge-Kutta integration. The "user parts" of these programs contain little more than the difference or differential equations describing the simulated system. The "overhead" that is usually present in FORTRAN-based models is almost completely transferred to the TTUTIL library routines.

1.1. Why FORTRAN ?

There are good arguments for the use of FORTRAN in scientific computation. In the first place the language is standardized and compilers are available for most types of computers. There is no valid reason for writing unreadable FORTRAN programs. Wagener (1980), for instance, presents an excellent introduction into programming in FORTRAN.

Secondly, an overwhelming number of well-tested subroutines is available designed to support scientific computation. Well-known examples are the IMSL library (Anonymous,1987) containing routines for special functions, matrix calculations, differential equations and many more. More recently the elegant and well-documented routines from "Numerical Recipes" (Press et al., 1986) became available to anyone who buys the book. In addition, more than 50

libraries of FORTRAN sources are available from NETLIB, a file server reachable via bitnet as NETLIB@ORNL.GOV. Among these are the famous libraries EISPACK and LINPACK on which many IMSL routines are based. In Appendix G the index of the NETLIB system is given.

1.2. Working on different machines

Programming languages enable the writing of machine-independent and readable programs. Large FORTRAN programs therefore need to be written according to the FORTRAN-77 standard. We regard as bad practice the use of, for instance, VAX-FORTRAN language extensions in simulation models. Furthermore, experience learns that compilers running on micro computers are not always as "clever" as the VAX-FORTRAN compiler. "Difficult" constructions like function calls and string concatenation within subroutine calls sometimes lead to problems. Moreover, the different compilers all have their own deficiencies. A few examples:

- On IBM PC an existing file is not overwritten if it is opened with status "NEW",
- The use of the STOP statement on the Apple Macintosh leads to the disappearance of the output window. That implies that fatal error messages cannot be written in the form:
  STOP 'message'.

In the 57 subroutines and functions presently available in TTUTIL we use a conservative programming style. Nested constructions in single statements are omitted (they are not very clear anyway) and we "programmed around" the compiler deficiencies we met. All subroutines and functions run on an Atari ST computer with the Prospero compiler, on an IBM PC with the Microsoft compiler, on an Apple MacIntosh with the Absoft compiler and on the VAX using the VAX-FORTRAN compiler (see Appendix A, however, for a few changes to be made before using the library on an Atari ST).

2. The structure of the library TTUTIL

Table 1 gives a classification of the TTUTIL routines. The first eight groups list generally applicable subroutines and functions. The next four groups are related to simulation models, to a smaller or larger extent. The last group consists of two simple routines for manipulating a VT100 screen. Closely related routines have names beginning with the same acronym. For instance, the "DEC" routines **DECCHK**, **DECINT**, **DECREA** and **DECREC** are used for decoding character strings into real or integer values.

Table 1 can be used for an efficient search through the library if you are looking for a routine that solves a specific programming problem. The routine headers, given in Chapter 4, provide further information on the individual subroutines and functions. In general, there will be no need for any further documentation. A few groups, however, require a more detailed introduction. These are the ENT routines for interactive variable entry, the RD routines for reading data files with a convenient format and the OUT routines for easy output programming. The RK routines for Runge-Kutta integration are separately introduced in Chapter 3.

2.1. The ENT routines

The usual way to obtain interactive input from the user is to write a question to the screen and to read the answer from the screen. Exactly that is the function of the simple routines **ENTCHA**, **ENTINT** and **ENTREA**. They can be used to ask for a character string, an integer value and a real value, respectively. For instance, the statement

```
CALL ENTREA ('Size of square',SIZE)
```

writes the question "Size of square" to the screen and the number returned is assigned to the real variable SIZE. Several such calls together form a relatively short program section for interactive input. Successive questions are written neatly below each other and the cursor is always in column 53 of the screen, independent of question length.

Somewhat less trivial are the subroutines **ENTDCH**, **ENTDIN** and **ENTDRE**. Again, the three routines are meant for entering a character string, an integer value or a real value, respectively. As an additional input argument, however, they accept a default value. The default value is returned to the calling program when the user does not type in a new value and presses the <Enter> key only. The three ENTD routines write the default value between square brackets following the question. For instance, the statement

```
CALL ENTDRE ('Size of square',2.300,SIZE)
```

causes the following line being written to the screen:

```
Size of square [2.3]:
```

The user either supplies a new value or just presses <Return> to accept the default. Note that the second argument (the default value) may also be a variable. The variable SIZE could be used, for instance, as the second and third argument of ENTDRE. Than the (current) value

Table 1. Classification of the TTUTIL library routines. In the underlined routines COMMON blocks are used. The names of these blocks are /OUTCUT/ and /INFO/ (cf. Appendix B).

Interactive input

ENTREA, ENTINT, ENTCHA
ENTDRE, ENTDIN, ENTDCH

Input from file

RDAREA, RDDATA, RDFROM, RDINDX, RDINIT, RDSETS, RDSINT, RDSREA,
GETCH, GETREC, MOFILP, COPFIL

Output to file

OUTCOM, OUTDAT, OUTPLT, OUTARR,
PLTFUN, PLTHIS, COPFIL

Character string handling

DECCHK, EXTENS, IFINDC, ILEN, ISTART, REMOVE, STRIP,
UPPERC, WORDS

Decoding of character strings to values

DECCHK, DECINT, DECREA, DECREC

Files

EXTENS, FOPEN, FOPENG, COPFIL

Error handling

ERROR

AFGEN tables (lists of X,Y pairs)

AFINVS, LINT, PLTFUN, PLTHIS

Random numbers

BOXMUL, GAMMA, UNIFL

Euler method in (crop) simulation ; integration in model routine

CHKTSK, INTGRL, TIMER

Euler and Runge-Kutta method (general, using STATE array)

EUDRIV, RKDRIV, RKQCA, RK4A

Emulation of some CSMP functions

INSW, LIMIT, LINT, REAAND, REANOR

VT100 screen

CLS, POS

of SIZE is used as the default answer. In section 5.1 the use of that trick to simplify testing of newly written subroutines is illustrated.

In order to keep the cursor behind the question, the ENT routines make use of a "$" in the output format string. This is not allowed in standard FORTRAN and it forms the only exception to the use of standard FORTRAN in the entire library. There seems to be no standard way, however, to keep the cursor after writing to the screen. An additional advantage of using the ENT routines is that interactive input sections can be kept free of a non-standard dollar.

2.2. The RD routines

The ordinary method for reading data from a file consists of a number of READ statements, each reading data from a record of the file. That method clearly requires that the sequence of READ statements is consistent with the contents of the file. Moreover, array lengths have to be known in the program or have to be read as separate data items. Formatted reading, moreover, requires accurate positioning of the data items. In general, much time is invested in debugging such "simple" input sections.

The solution suggested sometimes in text books on FORTRAN is to read data as character strings and to perform the decoding in the program. That requires a considerable programming effort and a need was felt for generally applicable input routines based on that principle. The RD routines enable the construction of clear, short and robust input sections consisting of CALL's only. The idea is that the input file contains both the variable name and the associated value(s). Then, the RD routines are able to find the value(s) to be assigned to a certain input variable. The syntax rules for the data files are the following:

1) The file consists of variable names and numerical values, separated by an '=' sign. So: PLMX=20.4 is a valid specification.
2) Variable names have a maximum length of six characters.
3) More than one value may appear on the right-hand side of the equal sign, separated by commas. This indicates that the variable is an array. In case of continuation on a following line the comma should be on the preceding line.
4) Values may be repeated. For instance, 100 times 1.0 can be written as 100*1.0
5) Variables may appear in any order in the file.
6) Comment lines start with "*" in the first column, or "!" in any column (rest of line is ignored).
7) Different name–value combinations on the same line should be separated by a semi–colon ";".
8) Only the first 80 characters of each line of the data file are decoded.
9) The decimal point in a real value like 20. may be omitted.
10) No TAB's may occur.

The following example file contains all these:

<start of file INPUT.DAT>

```
* example data file
N   = 10                        ! single value
BB  = 0, 2, 4, 6                ! array of four elements
CCC = 10., 20.,                 ! array continued on next line
      30., 40.
DD  = 100*10.                   ! array of 100 elements
EE  = 10.; FF = 20.; G = 30.    ! more than one variable on a single line
```

<end of file>

The following input section reads the values of CCC, BB, EE, FF and N respectively from the above listed data file INPUT.DAT. In the declarations section the parameters ILBMAX and ILCMAX specify the declared lengths of the arrays BB and CCC:

```
*     declarations
      INTEGER N,ILBMAX,ILB,ILCMAX,ILC
      PARAMETER (ILBMAX=100,ILCMAX=100)
      REAL BB,CCC,EE,FF
      DIMENSION BB(ILBMAX),CCC(ILCMAX)

*     example of input section
      CALL RDINIT (30,40,'INPUT.DAT')
      CALL RDSREA ('EE' ,EE)
      CALL RDSREA ('FF' ,FF)
      CALL RDAREA ('CCC',CCC,ILCMAX,ILC)
      CALL RDAREA ('BB' ,BB ,ILBMAX,ILB)
      CALL RDSINT ('N'  ,N )
      CLOSE (30,STATUS='DELETE')
```

The actual input section begins with a call to **RDINIT**. During execution of this call the data file is opened as unit 31 (=30+1) and is completely read and checked on syntax errors. The values occurring in the file are stored in a temporary file called INPUT.TMP (in this case), which is opened as unit 30. Finally, the actual data file is closed. During this operation, error messages are written to the screen and to unit 40, which preferably is a logfile.

After initialization with **RDINIT**, the values of two real variables are read by means of two calls to **RDSREA** (the last part of this name stands for single real). The first argument of this subroutine is the name of the variable, written as a character constant. Using that name, the routine **RDSREA** identifies the value to be assigned to the variable. So the character string in the CALL should correspond to the variable name in the data file.

Routine **RDAREA** reads arrays of real values. In the above example it is called two times for reading the arrays CCC and BB from file. Note that the declared length (=maximum length) is an input argument of **RDAREA** and the actual array length is an output argument. Clearly, the number of values in the file should not exceed the declared length. That leads to a fatal error message. Finally, the value of the single integer variable N is read using **RDSINT**. Note that it is not necessary to read every variable from the data file. By means of the CLOSE statement

the temporary file INPUT.TMP is deleted.

The subroutines **RDINIT**, **RDSREA**, **RDAREA** and **RDSINT** are just user interfaces to a lower level subroutine **RDDATA** which is called by these four. Sometimes the name **RDDATA** occurs in error messages, for instance when a variable name is not present in the file or in case of an array length error. In such error messages also the variable name is given. The suspect RD call is then easily identified.

Using the RD routines, one should be aware of a few limitations. All values that occur in a data file are stored in an unformatted temporary file as single precision REAL numbers (4 byte REAL's). That implies that the number of significant digits is about 6. To obtain an integer number, routine **RDSINT** simply performs a nearest integer operation on the stored value (using the intrinsic NINT function). In case of integer values larger than about 1,000,000 the result may differ from the contents of the data file. Further, only numeric input is allowed. Thus no character strings and logical variables can be read from a file. In future versions a distinction will be made between the different variable types while reading the data file. As soon as a routine RDSCHA appears in the library (for reading a single character string), the limitations with respect to integer values and character variables will have been removed. Future versions may require the use of a decimal point in real numbers. Hence, you may save yourself some (future) error messages by using these in all real numbers in your files. Double precision numbers will probably not be supported. Instructions will be added, however, for adapting the whole set of RD routines for reading double precision numbers.

2.3. Making reruns with the RD routines

The use of the RD routines for input has the additional advantage of a built in "rerun facility". Calculations often need to be carried out for different values of input variables. Suppose that something is calculated using the input variables BB, EE and FF from the above file INPUT.DAT. The calculations have to be repeated for different combinations of BB and EE. Then a so called rerun file can be created by the user, containing the desired combinations of BB and EE. For example:

<start of rerun file RERUNS.DAT>

```
* example rerun file
BB = 1, 3, 5, 7   ;  EE = 10.          ! set 1
BB = 2, 4         ;  EE = 10.          ! set 2 (with a short array)
BB = 0, 2, 4, 6   ;  EE = 30.          ! set 3
BB = 1, 3, 5, 7,                       ! set 4 (with a long array)
     8, 9, 8, 9   ;  EE = 30.
BB = 2, 4, 5, 7   ;  EE = 30.          ! set 5
```

<end of file>

A rerun file consists of sets of variables. The order of the variables should be identical in all sets. The syntax rules are identical to those of an ordinary data file. Actual array lengths may differ between sets (see the example above). The following program structure takes care of all

input:

```
*     open logfile and read rerun file
      CALL FOPEN (40,'RERUNS.LOG','NEW','DEL')
      CALL RDSETS (20,40,'RERUNS.DAT',INSETS)
*     runs
      DO 10 ISET = 0,INSETS
*        select rerun set
         CALL RDFROM (ISET,.TRUE.)

*        an ordinary input section:
         CALL RDINIT (30,40,'INPUT.DAT')
         CALL RDAREA ('BB' ,BB ,ILBMAX,ILB)
         CALL RDSREA ('EE' ,EE)
         CALL RDSREA ('FF' ,FF)
         CLOSE (30,STATUS='DELETE')

*        calculations
         ........
10    CONTINUE
      CLOSE (20,STATUS='DELETE')
```

This requires some further explanation: With a call to **FOPEN** a log file is opened, which is used for writing a report on rerun file usage. In the call to **RDSETS** the rerun file is analysed and a short report is written to the log file (unit 40). When the rerun file is not present or empty the output variable INSETS is set to zero. Otherwise the number of variable sets is returned (in the above example there are 5 rerun sets). By means of the call to **RDFROM** in the DO–loop, a certain set is selected. Selection of set zero means that the contents of the original data file INPUT.DAT will be used. The input section for reading the values of BB, EE and FF is just an ordinary input section for reading variables from a data file. The RD routines, however, internally check whether reruns are being made and whether a non–zero set was selected. In that case, for variables occurring in the rerun file, the data file contents are replaced by the contents of the rerun file. Since this is a rather hidden activity, each replacement is reported to the log file. Again, the CLOSE statements remove the temporary files created by the RD routines.

The file "RERUNS.DAT" may be absent or present. If the file is absent, the above program section will carry out only one run using the contents of the data file.

Within a program, replacement of data can be switched off by selecting set 0 with **RDFROM** or by calling **RDSETS** with a space as filename (`CALL RDSETS (0,0,' ',I)`). The rerun facility has a global character, i.e. variables stored in different data files may occur in a single rerun file. Within the above DO–loop, for instance, the variables BB and EE could be read from different input files by writing two separate input sections (each containing a call to **RDINIT**). As a consequence, the use of identical variable names in different input files leads to problems when reruns are made for that variable. Then the value of both variables will be replaced by the contents of the rerun file. Both replacements will be reported to the produced log file.

2.4. The OUT routines

The OUT routines are used to generate neat output tables with a minimum of programming effort. During calculations the name and value of a variable can be sent to routine **OUTDAT** which behaves as a temporary output box. It stores the received output in a temporary file. After completion of the calculations a table can be constructed by means of a special call to **OUTDAT**. The table can be used as the final result or can be imported into a spreadsheet or a statistical program.

The use of **OUTDAT** is illustrated in the example program TEST below. A table and a printplot are created of the sine and cosine of x between 0 and π.

```
01          PROGRAM TEST
02          IMPLICIT REAL (A-Z)
03          INTEGER IX
04          PARAMETER (PI=3.141597)
05
06    *     initialize output ; X is independent
07          CALL OUTDAT (1, 20, 'X', 0.0)
08
09          DO 10 IX = 0,20
10             X = FLOAT(IX) * PI/20.0
11             SINX = SIN (X)
12             COSX = COS (X)
13    *        repeated output calls
14             CALL OUTDAT (2, 0, 'X'   , X   )
15             CALL OUTDAT (2, 0, 'SINX', SINX)
16             CALL OUTDAT (2, 0, 'COSX', COSX)
17    10    CONTINUE
18
19    *     table construction
20          CALL OUTDAT ( 4, 0, 'sine + cosine', 0.0)
21
22    *     printplot contruction
23          CALL OUTPLT ( 1, 'SINX')
24          CALL OUTPLT ( 1, 'COSX')
25          CALL OUTPLT ( 7, 'sine + cosine')
26
27    *     delete temporary
28          CALL OUTDAT (99, 0, ' ', 0.0)
29          STOP
30          END
```

Table 2 shows the output produced by this example program. The first parameter of the routines **OUTDAT** and **OUTPLT** is a task parameter. The first CALL to **OUTDAT** in line 7 of the above program (with ITASK=1) specifies that X will be the independent variable and that unit=20 can be used for the output file. Subsequent calls in lines 14,15 and 16 with ITASK=2 instruct **OUTDAT** to store the incoming names and values in a temporary file (with unit=21). The number of values that can be stored is only dependent on free disk space and not on RAM memory. The terminal call to **OUTDAT** in line 20 (with ITASK=4) instructs the routine to

Table 2. Output produced by example program TEST in the text.

```
*-------------------------------------------------------------------------
* Run no.:  1, (Table output)
* sine + cosine

      X          SINX          COSX

  .00000        .00000        1.0000
  .15708        .15643        .98769
  .31416        .30902        .95106
  .47124        .45399        .89101
  .62832        .58779        .80902
  .78540        .70711        .70711
  .94248        .80902        .58778
  1.0996        .89101        .45399
  1.2566        .95106        .30902
  1.4137        .98769        .15643
  1.5708        1.0000     -0.19809E-05
  1.7279        .98769       -.15644
  1.8850        .95106       -.30902
  2.0420        .89101       -.45399
  2.1991        .80902       -.58779
  2.3562        .70710       -.70711
  2.5133        .58778       -.80902
  2.6704        .45399       -.89101
  2.8274        .30901       -.95106
  2.9845        .15643       -.98769
  3.1416     -0.45280E-05    -1.0000


           sine + cosine

           Variable  Marker  Minimum value  Maximum value
           --------  ------  -------------  -------------
           SINX         1      -0.4528E-05       1.000
           COSX         2       -1.000           1.000

           Scaling: Common      -1.000           1.000

X

  .00000   I-------------------------------1-------------------------------2
  .15708   I               I               I    1          I                2
  .31416   I               I               I         1     I             2  I
  .47124   I               I               I              1I            2   I
  .62832   I               I               I               I  1      2      I
  .78540   I               I               I                  *           I
  .94248   I               I               I               I  2      1      I
  1.0996   I               I               I              2I            1   I
  1.2566   I               I               I         2     I             1  I
  1.4137   I               I               I    2          I                1
  1.5708   I-------------------------------2-------------------------------1
  1.7279   I               I          2    I               I                1
  1.8850   I               I     2         I               I             1  I
  2.0420   I               I2              I               I            1   I
  2.1991   I            2  I               I               I         1      I
  2.3562   I        2      I               I               I      1         I
  2.5133   I     2         I               I               I  1             I
  2.6704   I  2            I               I              1I                I
  2.8274   I 2             I               I         1     I                I
  2.9845   2               I               I    1          I                I
  3.1416   2-------------------------------1-------------------------------I
```

create an output table using the information stored in the temporary file. Dependent on the value of the task variable, different output formats are chosen. Tab-delimited format (for a spreadsheet) can be generated with ITASK=5, two column format with ITASK=6. The string between quotes is written above the output table.

The calls to **OUTPLT** in line 22 and 23 (with ITASK=1) instruct the routine to put "SINX" and "COSX" in a graph (up to 25 variables can be printed per plot). The subsequent call with ITASK=7 causes **OUTPLT** to create the plot. Two different widths of the printplot are possible, 80 and 132 columns, and two different types of scaling, a common scale and individual scales (see Table 3). The process can be repeated to create several print plots based on the same output data. The final call to **OUTDAT** (in line 28 with ITASK=99) deletes the temporary file.

Table 3. The task variable that should be supplied to **OUTPLT** to generate the different print plot types.

		Width	
		132	80
Scaling	Individual	4	6
	Common	5	7

2.5. Handling of fatal errors

Fatal errors are dealt with by calling the subroutine **ERROR** that writes a message to the screen, requires a <Return> from the keyboard and then terminates program execution. That procedure leads to identical results on all machines tested. A STOP statement with a message does not work on an Apple MacIntosh as explained in section 1.2. Users of TTUTIL can use subroutine ERROR also for their own fatal errors (see the header of **ERROR**).

3. Simulation in FORTRAN

Simulation is used here as a synonym for solving an initial value problem. A system is described by a number of state variables. For each state variable there is an equation describing its rate of change as function of time, of the state variables and of model parameters. Starting from the initial state, the future behavior of the system can be simulated by integrating the rates of change over time.

Simulation programs, written in a general programming language like FORTRAN, are especially useful when they are accessible by other programs. Then different models can be combined, a simulation model can be subjected to an algorithm for parameter optimization, it can be made part of an educational program, etc. In all these cases the simulation model has to have a clear structure that is accessible at different levels.

At the lowest level rates of change are calculated or new values of the state variables are calculated (integration). In principle, simulation consists of a sequence of such steps. This process should not be "free running", however. When time progress is controlled by a higher level routine (a model driver) different models can be coupled much more easily. The model driver contains a loop for time control, a so called dynamic loop, from which the actual model subroutines are called. Time progress is controlled by the driver. Then the different subprocesses are simulated simultaneously and interactions can be taken into account without needing to combine the two models into a newly written program. The model driver performs a complete simulation run and in turn, the driver may be called by a program specifying reruns of the model. This general idea has been worked out earlier for Euler integration in Van Kraalingen and Rappoldt (1989) and Van Kraalingen (1991).

Considering Euler integration, it is useful to make a distinction between differential and difference equations. The solution of differential equations requires either a small timestep or some type of accuracy control. The solution of difference equations implies the use of a fixed timestep that is usually related to some natural timescale of the simulated process. In both cases rates of change can be defined as changes of the state variables per unit of time. The implementation of Euler integration in a FORTRAN program is very similar for difference and differential equations. In case of difference equations the timestep is always kept fixed, also when an output time is passed. Differential equations enable the use of a variable timestep.

At first a driver is discussed for solving difference equations. In models for crop growth, for instance, the diurnal cycle of the physiological processes is often not considered explicitly. Instead, growth rates are derived from total daily photosynthesis. The system is described by a set of difference equations to be solved for a fixed time step of one day and the formalism of Euler integration is the only correct method. It is shown how the use of large common blocks can be avoided, how different models can be combined and how one can make use of the input, output and rerun facilities from the library TTUTIL.

Table 4. A simulation model with Euler integration

```
01        SUBROUTINE MODEL (ITASK,IUDAT,IULOG,OUTPUT,TIME,DELT,TERMNL)
02
03  *     Example model
04  *
05  *     ITASK  - task parameter, see IF-THEN-ELSE structure below    I
06  *     IUDAT  - unit number for data file reading                   I
07  *     IULOG  - unit number of open logfile                         I
08  *     OUTPUT - logical enabling dynamic output                     I
09  *     TIME   - time                                                I
10  *     DELT   - time step                                           I
11  *     TERMNL - driver will terminate run when set .TRUE.           O
12
13  *     declarations
14        IMPLICIT REAL (A-Z)
15        INTEGER ITASK, IUDAT, IULOG, ITOLD
16        LOGICAL OUTPUT, TERMNL
17        SAVE
18        DATA ITOLD/4/
19
20  *     check for illegal driver actions
21        CALL CHKTSK ('MODEL',IULOG,ITOLD,ITASK)
22
23        IF (ITASK.EQ.1) THEN
24  *        initialization
25           CALL RDINIT (IUDAT,IULOG,'MODEL.DAT')
26           CALL RDSREA ('PAR1',PAR1)
27           CALL RDSREA ('PAR2',PAR2)
28           CALL RDSREA ('A'   ,A   )
29           CALL RDSREA ('B'   ,B   )
30           CLOSE (IUDAT,STATUS='DELETE')
31
32        ELSE IF (ITASK.EQ.2) THEN
33  *        calculation of rates of change
34           RA = EQUATA (TIME,A,B,PAR1,PAR2)
35           RB = EQUATB (TIME,A,B,PAR1,PAR2)
36
37           IF (OUTPUT) THEN
38              CALL OUTDAT (2,0, 'A', A)
39              CALL OUTDAT (2,0, 'B', B)
40           END IF
41
42        ELSE IF (ITASK.EQ.3) THEN
43  *        integration
44           A = A + RA * DELT
45           B = B + RB * DELT
46
47        ELSE IF (ITASK.EQ.4) THEN
48  *        terminal actions (none)
49           CONTINUE
50        END IF
51
52        ITOLD = ITASK
53        RETURN
54        END
```

The solution of differential equations often requires more accurate integration methods. Extensive literature exists on that topic and only a few references can be given here. In Press et al. (1986) subroutines can be found enabling fourth order Runge-Kutta integration. In the IMSL library (Anonymous,1987) and in NETLIB libraries (cf. Appendix G) more powerful methods are available. In this chapter is shown that the program structure designed for Euler integration can be used in combination with a Runge-Kutta algortithm as well. A complete example will be given. The implementation of the Runge-Kutta method makes use of existing subroutines from Press et al. (1986) that have been adapted to our needs.

Finally, the driver for Runge-Kutta integration is simplified to a driver for Euler integration of differential equations.

3.1. Euler integration of difference equations

At first a model subroutine is discussed. Then it is shown how a model driver can be written. The full source text of the driver is given in Appendix D. Different sections of it will be discussed. With respect to some minor points the driver differs from the programs discussed in Van Kraalingen (1991). That has been done to reduce the differences with the driver for Runge-Kutta integration discussed in section 3.2.

3.1.1. *Example of a model subroutine*

A simulation model always consists of an initial section, a section for the calculation of rates of change, an integration procedure and an optional terminal section. In large models the transfer of variable values among these four sections is a problem. Often large common blocks are used that tend to appear everywhere in the program and finally lead to an unclear and inaccessible program structure. The solution proposed by Van Kraalingen & Rappoldt (1989) is the use of different sections within a single subroutine. The execution of a particular section is controlled by the value of the input parameter ITASK. In Table 4 a simple example subroutine is given.

In the initial section (lines 24-30, ITASK=1) two model parameters and the initial values of A and B are read from file MODEL.DAT. Rates of change are calculated in lines 34 and 35 as two functions (here unspecified). The calculated rates belong to the current state. Hence, any output (enabled by input parameter OUTPUT having the value .TRUE.) should occur at this point. The example model sends the state variables A and B to the output routine (lines 38 and 39). It could send the rates of change or derived variables as well. In lines 44 and 45 (ITASK=3), new values for the state variables are calculated.

Subroutine **CHKTSK** is called in line 21. It checks the driver for illegal sequences of ITASK values. Two successive integration calls (ITASK=3), for instance, lead to a fatal error. The variable ITOLD (the "previous task") is initialized at 4 in line 18. Hence, the first call to the model is equivalent to a call after termination and initialization is allowed.

3.1.2. *A driver for Euler integration*

The model driver controls a sequence of calls to the model. Initialization of the model is followed by a series of rate and integration calls and finally model execution is terminated. In addition, the driver takes care of the TIME and initializes and terminates the output routine **OUTDAT**. For crop growth simulation, Van Kraalingen (1991) wrote a "luxurious" driver that keeps track of the calendar date and supplies weather data to the model(s). Here, a simpler driver is discussed, mainly as an introduction to the Runge-Kutta routines. In Appendix D the complete listing of the simple Euler driver is given, which is not part of the library.

The driver subroutine starts by reading some "timer variables" from file TIMER.DAT. These are the start time, STTIME, and end time, FINTIM, of the simulation, the fixed time step, DELT, and the period between output points, PRDEL. The values are read from the file by means of the calls to **RDSREA** in the following program section:

```
*     read timer variables ; temporary use of unit IUDAT
      CALL RDINIT (IUDAT,IULOG,'TIMER.DAT')
      CALL RDSREA ('STTIME',STTIME)
      CALL RDSREA ('FINTIM',FINTIM)
      CALL RDSREA ('PRDEL' ,PRDEL )
      CALL RDSREA ('DELT'  ,DELT  )
      CLOSE (IUDAT,STATUS='DELETE')
```

The next step is to initialize time. Then the output routine OUTDAT is instructed to use TIME as the independent variable and the model is initialized:

```
*     initialize time
      TIME  = STTIME
*     initialize output
      CALL OUTDAT (1,IUOUT,'TIME',TIME)
*     initialize model(s)
      CALL MODEL (1,IUDAT,IULOG,.FALSE.,TIME,DELT,TERMNL)
```

The dynamic loop requires some more explanation. Its basic function is to perform a series of rate and integration calls to the model. That process continues until time reaches FINTIM. In designing the loop it should be kept in mind that only after rate calculation consistent output is possible. Since also at termination of the model output is desirable, at FINTIM rates of change have to be calculated, which are not integrated anymore. Hence, N integration steps require N+1 rate calculations. Basically, there are two solutions to this problem. Either at the start or at the end of the simulation one integration call is skipped. The former option is chosen by Van Kraalingen (1991). Here it is shown that the latter option also works. It leads to a construction that will also be used below in the driver for Runge-Kutta integration.

In addition to controlling the series of rate and integration calls, the driver also controls model output by means of the output flag OUTPUT. The flag should be set commensurate with the value of PRDEL. Furthermore, the driver should guarantee the generation of final output, also when the simulation is interrupted by the model (by setting its termination flag TERMNL).

The last parameter of the model routine is the logical TERMNL. It is an output parameter and may be set .TRUE. by the model if, for some reason, the simulation has to be terminated. The model driver then should take care of a final rate call with output enabled (OUTPUT=.TRUE.) followed by a terminal call to the model with ITASK=4. The terminal flag TERMNL may be set during the rate calculation or in the integration part.

The dynamic loop in Table 5 meets all these requirements. It makes use of a few auxiliary variables. The control variable of the loop is not the terminal flag TERMNL but a separate logical HALT. The reason will be clear after the explanation on terminal output below. Clearly, HALT should be initialized at .FALSE. before entering the loop (see the complete listing in Appendix D). The next output time is TNEXT (initialized as the start time) and IP keeps track of the number of intervals PRDEL (initialized at 0).

Table 5. Dynamic loop of driver for difference equations

```
*     dynamic loop (emulated DO WHILE)
10    IF (.NOT.HALT) THEN
*        output required ?
         OUTPUT = (TNEXT-TIME)/DELT.LT.0.01 .OR. TERMNL

*        write time to output ; rate call
         IF (OUTPUT) CALL OUTDAT (2,0,'TIME',TIME)
         CALL MODEL (2,IUDAT,IULOG,OUTPUT,TIME,DELT,TERMNL)

         IF (OUTPUT) THEN
*           get next output time ; leave dynamic loop ?
            IP    = IP + 1
            TNEXT = MIN (STTIME + IP * PRDEL, FINTIM)
            HALT  = (FINTIM-TIME)/DELT.LT.0.01 .OR. TERMNL
         END IF

         IF (.NOT.(HALT.OR.TERMNL)) THEN
*           integrate time and state variables in model(s)
            TIME = TIME + DELT
            CALL MODEL (3,IUDAT,IULOG,.FALSE.,TIME,DELT,TERMNL)
         END IF
      GOTO 10
      END IF
```

The first statement in the loop sets or resets the output flag OUTPUT. When the preset time TNEXT has been reached or passed, output is enabled and the current time is sent to routine **OUTDAT**. Following output, a new output time is calculated and the HALT flag is set if FINTIM has actually been reached or on request of the model (TERMNL=.TRUE.). Then, as long as the simulation continues, time and state variables are integrated.

Note that the value of TNEXT cannot exceed that of FINTIM. Hence, when TIME reaches FINTIM, output is generated and the HALT flag is set. With the HALT flag set so that integration of the final rates is skipped. The "termination request" flag, TERMNL, can be set by the model routine either during rate or integration calls. Setting it during an integration call

immediately leads to output during the next rate call and termination of model execution. When the model requests termination during a rate call, however, the driver's reaction depends on the output flag. If output has just been generated, the HALT flag is set, integration is skipped and model execution is terminated. If termination is requested without output having been generated, integration is skipped (TERMNL=.TRUE.), rate calculation is repeated, now with output enabled, after which the HALT flag is set. The two successive rate calls lead to a warning from routine **CHKTSK** called in the model routine. Such warnings may be omitted by setting the TERMNL flag in the integration sections of your models only.

When PRDEL is not a multiple of DELT, the desired output times are not precisely reached. The model will generate output at the first multiple of DELT exceeding the desired output time. That is the consequence of a fixed time step. Dealing with difference equations, a fixed time step is, however, the only correct method. When differential equations are solved the time step may be adjusted so as to reach exactly the next output time TNEXT. That should be done just before the integration call. The IF-ENDIF block, containing that call may be changed into:

```
      IF (.NOT.(HALT.OR.TERMNL)) THEN
*        integrate time and state variables in model(s)
         DELT1 = MIN (DELT, TNEXT-TIME)
         TIME  = TIME + DELT1
         CALL MODEL (3,IUDAT,IULOG,.FALSE.,TIME,DELT1,TERMNL)
      END IF
```

After leaving the dynamic loop the driver performs a terminal call to the model, generates table output with TIME in the first column and deletes a temporary file:

```
*     terminate model ; table output ; delete temporary file
      CALL MODEL (4,IUDAT,IULOG,OUTPUT,TIME,DELT,TERMNL)
      CALL OUTDAT (4,0,'MODEL',0.0)
      CALL OUTDAT (99,0,' ',0.0)
```

3.1.3. *Coupling different models*

When different systems have to be simulated simultaneously, the models describing them have to be coupled. In the driver discussed above, the calls to the routine MODEL may be replaced by blocks of calls to the various models. At first all models are initialized, subsequently all rates are calculated, all rates are integrated, etc. Output from all models may sent to output to routine **OUTDAT** as long as the variable names used are different. In the final output table the output of the different subroutines is combined.

Usually, the coupling will be carried out because interactions exist between the simulated subsystems. Model 2 may depend on the state variable A of model 1, for instance. That can be expressed in the program by adding A to the parameter list of both models. Then the rate calculation in model 2 can be made dependent on the added interaction variable A. In more complicated situations it may be more elegant to use a common block expressing the interaction between the two systems. Ideally, the interaction only takes place through to state

variables. Then all rates of change are calculated on the basis of the same overall system state and the order of the rate calls is not important. When also rates of change are part of the interaction, one should be more careful.

3.1.4. *Reruns*

When the RD routines are used for reading the timer variables in the driver and for all model input, reruns can be made varying the values of all variables. A very simple main program takes care of the reruns and calls the model driver:

```
      PROGRAM EULER

      IMPLICIT REAL (A-Z)
      INTEGER IS, INSETS, IULOG, IURER, IUDAT, IUOUT
      DATA IULOG/20/,IURER/30/,IUDAT/40/,IUOUT/50/

*     open log file and analyse rerun file
      CALL FOPEN (IULOG,'MODEL.LOG','NEW','DEL')
      CALL RDSETS (IURER,IULOG,'RERUNS.DAT',INSETS)
*     model runs
      DO 10 IS = 0,INSETS
         CALL RDFROM (IS,.TRUE.)
         CALL DRIVER (IUDAT,IULOG,IUOUT)
10    CONTINUE
      IF (INSETS.GT.0) CLOSE (IURER,STATUS='DELETE')
      STOP
      END
```

Note that the use of the various unit numbers is determined in the main program. The driver temporarily uses IUDAT and IUDAT+1 for reading timer variables and the example model uses the same units for reading its data file (real models might use more units). Also IURER and IUOUT should be the first of two free unit numbers.

3.2. Runge-Kutta integration of differential equations

Runge-Kutta integration of differential equations leads to more accurate results than the simple Euler method. A very readable introduction to the mathematical backgrounds of the method can be found in Press et al. (1986). The Runge-Kutta integration routines **RKQCA** and **RK4A** in the TTUTIL library are adapted versions of the routines given in that book. When using these subroutines you are strongly advised to study the relevant pages in Press et al. (1986) and you should refer to that book in any publication.

Applying a model-driving routine, Runge-Kutta integration is as easy to implement as Euler integration. The TTUTIL library contains a model driver **RKDRIV** that uses Runge-Kutta integration with stepsize control. The structure of **RKDRIV** is very similar to that of the Euler driver discussed above. The main difference is in the integration section. The Euler driver calls the model with ITASK=3, requesting the model routine to integrate its state variables. The Runge-Kutta driver on the other hand, calls a separate integration routine that simulates

the system over a selected time step. The integration section of **RKDRIV** reads:

```
*         one step with accuracy control using RKQCA
          IF (.NOT.HALT .AND. .NOT.TERMNL) THEN
*            limit time step
             DELNXT = MIN (DELNXT,DELMAX)

             IF (TIME+DELNXT .LT. TNEXT) THEN
*               accept advised step
                DELT = DELNXT
                CALL RKQCA (STATE,RATE,NDEC,NEQ,TIME,DELT,EPS,
     $           SCALE,DELDID,DELNXT,MODEL)
             ELSE
*               reduce time step; do not overwrite previous advise
                DELT = TNEXT-TIME
                CALL RKQCA (STATE,RATE,NDEC,NEQ,TIME,DELT,EPS,
     $           SCALE,DELDID,DUMMY,MODEL)
             END IF
          END IF
```

The dependence of the integration on HALT and TERMNL is the same as in the Euler driver previously discussed. Following the first IF statement the time step DELNXT is limited to DELMAX, a variable read by the driver from the file TIMER.DAT. Then, if TIME plus the time step does not surpass the next output time TNEXT, DELT is set equal to DELNXT and the integration routine **RKQCA** is called. That routine executes the model using the step DELT. That is accomplished by means of 11 rate calls to the model routine. When the result does not satisfy the accuracy criterion, a smaller step is taken. In addition to to the state variables of the model also TIME will have a new value when returning from **RKQCA**. The integration routine further returns a new value of DELNXT, the advised size for the next step.

When the value of TIME plus DELNXT exceeds that of the next output time TNEXT, DELT is reduced to TNEXT-TIME before calling **RKQCA**. Returning from **RKQCA**, the new advice for DELNXT may be a too small number (as a result of the step reduction). Therefore, that advice is ignored by means of the variable DUMMY and the current value of DELNXT will be used again.

The accuracy of the integration is controlled by the user supplied value of EPS in the file TIMER.DAT. That file also contains the value of the first time step TRY1 to begin with after initialization of the model. Hence, instead of the fixed time step DELT used in the Euler driver, TIMER.DAT should contain values for the maximum step DELMAX, the accuracy parameter EPS and the first step TRY1.

The integration routine **RKQCA** can only perform rate calls to the model when it "knows" the formal parameters of the model subroutine. Therefore, the parameter list of the model routine has been made independent of the problem at hand. Instead of meaningful names for state variables one has to use a dull array STATE with elements STATE(1), STATE(2), STATE(3),...etc. Table 6 shows a listing of a model subroutine that can be combined with the

Runge-Kutta driver **RKDRIV**. It is discussed below in more detail.

3.2.1. *A model subroutine for Runge-Kutta integration*

The predator-prey model in Table 6 consists of an initial section, a rate calculating section and a terminal section. It does not contain an integration section, since integration is performed by the integration routine **RKQCA**. As a consequence, the state and rate variables of the model must be formal parameters and the parameter list has to be standardized. Table 6 shows that arrays have been defined with state variables and calculated rates. These arrays have a declared length NDEC and an actual length NEQ (number of equations). Evidently, the use of names like STATE(.) and RATE(.) in the model routine would easily lead to unreadable programs and errors. Therefore, before any operation begins, the array elements are copied into local variables with more appropriate names (see lines 74 and 75). Similarly, calculated rates are copied into the RATE array before leaving the model routine (lines 89 and 90).

The function of the common block /INFO/ is to transfer some variable values between the driver and the model. The variables in /INFO/ cannot be included in the parameter list of the model routine since the model is also called from **RKQCA** and **RK4A**. The model needs a log file unit number IULOG. And the integer IUMOD is the first of a series of free unit numbers. In the example routine, IUMOD itself is used for output and IUMOD+2 for input (see lines 41,42,45,58 and 64). The logical TERMNL has the same function as in the "Euler model" discussed above. When it is set .TRUE., the driver **RKDRIV** will terminate the simulation. The function of KEEP will be explained in section 3.2.5.

The initial section of the model routine starts with setting local unitnumbers. Then the initial state (line 47-48), the model parameters (lines 50-55) and the SCALE (line 57) array are read from file. The array SCALE contains the order of magnitude of the values of the state variables. It is used in the evaluation of the accuracy criterion by routine **RKQCA**. For the i-th state variable the tolerated error is equal to EPS (from TIMER.DAT) multiplied by SCALE(i). (see Press et al. (1986) for a discussion of this method).

In the initial section also the output routine **OUTDAT** is initialized. Contrary to the situation with Euler integration, this initialization is not done by the driver and, consequently, the model routine has to do it itself. That leaves the user completely free in choosing the independent variable or writing alternativeoutput sections instead of using **OUTDAT**. The driver never has to be changed and has been made part of the library TTUTIL.

The rate–calculating section starts with assigning values to the local variables PREY and PRED. Subsequently, the rates of change are calculated according to the following differential equations (Rosenzweig, 1972 ; V=preys, P=predators):

$$\begin{cases} \dfrac{dV}{dt} = r\left(1 - \dfrac{V}{K}\right)V - c\left(1 - e^{-aV/c}\right)P \\ \dfrac{dP}{dt} = bc\left(1 - e^{-aV/c}\right)P - dP \end{cases}$$

Table 6. A model subroutine suitable for Runge-Kutta integration

```
001         SUBROUTINE MODEL (ITASK,OUTPUT,TIME,STATE,RATE,SCALE,NDEC,NEQ)
002
003   *     Predator prey simulation model that leading to a stable limit
004   *     cycle when the prey carrying capacity is relatively large.
005   *     The model originates from Rosenzweig (1972) and is discussed
006   *     also by May (1972). The parameter names used here are taken
007   *     from Roughgarden (1979). The STANDARD (!!) parameter list:
008   *
009   *     ITASK  - task of model routine                        I
010   *     OUTPUT = .TRUE. output request (ITASK=2 only)         I
011   *     TIME   - time                                         I
012   *     STATE  - state array of model                        I/O
013   *     RATE   - rates of change belonging to STATE          I/O
014   *     SCALE  - size scale of state variables               I/O
015   *     NDEC   - declared size of arrays                      I
016   *     NEQ    - Number of state variables, for ITASK=1       O
017   *                                         otherwise         I
018
019         IMPLICIT REAL (A-Z)
020
021   *     formal parameters, === do not change this section !! ====
022         INTEGER ITASK, NDEC, NEQ
023         DIMENSION STATE(NDEC),RATE(NDEC),SCALE(NDEC)
024         LOGICAL OUTPUT
025
026   *     common /INFO/,      === do not change this section !! ====
027         INTEGER IULOG, IUMOD, KEEP
028         LOGICAL TERMNL
029         COMMON /INFO/ IULOG,IUMOD,KEEP,TERMNL
030
031   *     local (non-common) variables
032         INTEGER IURES, IUDAT
033         REAL PREY, PRED, RPREY, RPRED
034         REAL R, K, A, C, B, D, FOUND
035         SAVE
036
037         IF (ITASK.EQ.1) THEN
038   *        initial
039   *        =======
040   *        get local unit numbers
041            IURES = IUMOD
042            IUDAT = IUMOD + 2
043
044   *        open input file
045            CALL RDINIT (IUDAT,IULOG,'MODEL.DAT')
046   *        initial state
047            CALL RDSREA ('PREY',PREY)
048            CALL RDSREA ('PRED',PRED)
049   *        model parameters (see data file for meaning of symbols)
050            CALL RDSREA ('R',R)
051            CALL RDSREA ('K',K)
052            CALL RDSREA ('A',A)
053            CALL RDSREA ('C',C)
054            CALL RDSREA ('B',B)
055            CALL RDSREA ('D',D)
056   *        scales
057            CALL RDAREA ('SCALE',SCALE,NDEC,NEQ)
058            CLOSE (IUDAT,STATUS='DELETE')
059
060   *        check the size of the SCALE array on file:
061            IF (NEQ.NE.2) CALL ERROR ('MODEL','SCALE length mismatch')
062
063   *        initialize output
```

```
064           CALL OUTDAT (1,IURES,'PREY',PREY)
065
066   *       initialize state variables
067           STATE(1) = PREY
068           STATE(2) = PRED
069
070         ELSE IF (ITASK.EQ.2) THEN
071   *       rates of change
072   *       ===============
073   *       assign state to local variable names
074           PREY = STATE(1)
075           PRED = STATE(2)
076
077   *       equations of the model using found preys per predator
078           FOUND = C * (1.0 - EXP (-1.0 * A * PREY / C))
079           RPREY = R * PREY * (1.0 - PREY/K) - FOUND * PRED
080           RPRED = (B * FOUND - D) * PRED
081
082   *       output section
083           IF (OUTPUT) THEN
084              CALL OUTDAT (2,0,'PREY',PREY)
085              CALL OUTDAT (2,0,'PRED',PRED)
086           END IF
087
088   *       assign result to rate array
089           RATE(1) = RPREY
090           RATE(2) = RPRED
091
092         ELSE IF (ITASK.EQ.4) THEN
093   *       terminal
094   *       ========
095   *       table output in two columns and delete temporary
096           CALL OUTDAT (6,0,'A stable limit cycle',0.0)
097           CALL OUTDAT (99,0,' ',0.0)
098         END IF
099
100         RETURN
101         END
```

The calculated rates of change are assigned to elements of the output array RATE. When output is enabled, the state variables are sent to **OUTDAT**. In the terminal section (lines 96 and 97) **OUTDAT** is called for a table with results. (Note that also rates can be sent to **OUTDAT.**)

The use of a standard parameter list has an important advantage. All calls to the model can be brought in a standard form. That implies that model execution can be made fully independent of the problem at hand. The user only has to supply a single subroutine describing the system whereas the driver does not need to be changed.

3.2.2. *The main program*

The model subroutine in Table 6 can be executed by means of the following (model-independent) main program:

```
      PROGRAM RK

      IMPLICIT REAL (A-Z)
      INTEGER IS, INSETS, IULOG, IURER, IUDRIV, IUMOD
      EXTERNAL MODEL
      DATA IULOG/20/,IURER/30/,IUDRIV/40/,IUMOD/50/

*     open log file and analyse rerun file
      CALL FOPEN (IULOG,'MODEL.LOG','NEW','DEL')
      CALL RDSETS (IURER,IULOG,'RERUNS.DAT',INSETS)
*     model runs
      DO 10 IS = 0,INSETS
         CALL RDFROM (IS,.TRUE.)
         CALL RKDRIV (IULOG,IUDRIV,IUMOD,MODEL)
10    CONTINUE
      IF (INSETS.GT.0) CLOSE (IURER,STATUS='DELETE')
      STOP
      END
```

In the DATA statement unit numbers are set for a log file, a rerun file, the file TIMER.DAT (read by the driver) and for use by the model routine. Note that all timer variables and model parameters may occur in the rerun file.

The name of the model subroutine is declared external by means of the EXTERNAL statement. Another name could be used in the above main program without having to change the driver **RKDRIV** and the integration routines in TTUTIL. On some computers the linker appears to have problems with handling external subprograms. Problems tend to arise especially when routines with external modules in their parameter list (like **RKDRIV**) are linked from an object library. Hence, inclusion of the driver **RKDRIV** by means of an include statement in the compiled source will almost always cure the problem. Also the lower level routines **RKQCA** and **RK4A** may have to be included. With Absoft FORTRAN on Apple Macintosh the problem can be solved by including in the main program (after the declarations) a fully dummy call to the model subroutine:

```
      CALL MODEL (0,.FALSE.,DUM,DUM,DUM,DUM,1,I)
```

It is easily verified that this call does absolutely nothing (cf. Table 6). It just takes over the function of the external statement that seems to be incorrectly dealt with.

3.2.3. *Results of the example model*

The above main program and the example model in Table 6 have been compiled and linked to the TTUTIL library. The data files used in combination with the program are given in Table 7.

The biological interpretation of the model parameters is given as comment in the file MODEL.DAT. The differential equations describe a predator–prey model with saturation. At low prey density, the number of preys eaten per predator is proportional to the prey density. At a high prey density, the predators have no difficulties in finding their preys and have enough to eat. As a consequence, the number of preys eaten per predator approaches a maximum. This saturation effect appears to destabilize the system. Without saturation a

Table 7. Data files belonging to the model in Table 6. Figure 1 shows the results.

```
<file MODEL.DAT>
* initial numbers + scales
PREY  =  700.0 ; PRED =  50.0  ! model run starting from inside the limit cycle
SCALE = 1000.0,         100.0
* model parameters ; values taken from Roughgarden (1979)
* ======================================================
R = 0.5   !  relative growth rate prey at low density
K = 3500  !  carrying capacity of environment for prey
A = 0.01  !  consumption of preys per prey per predator at low prey density
C = 10.0  !  consumption of preys per predator at high prey density
B = 0.02  !  predator's increase per eaten prey
D = 0.1   !  relative death rate predator without food
<End>

<file TIMER.DAT>
* simulation control variables
STTIME = 0.0    ! start time
FINTIM = 1000.  ! finish time
PRDEL  = 1.0    ! output time step
EPS    = 1E-4   ! tolerable relative integration error
TRY1   = 0.01   ! First try of time step
DELMAX = 5.0    ! Maximum allowed time step
<End>

<file RERUNS.DAT>
* rerun of the model starting from outside the limit cycle
PREY = 200.0 ; PRED = 15.0
<End>
```

predator–prey model has a stable equilibrium. For a relatively large carrying capacity K, this equilibrium is not very stable against disturbances, however. The saturation effect can be regarded as such a disturbance. Adding it to the model leads, for large values of K, to a so

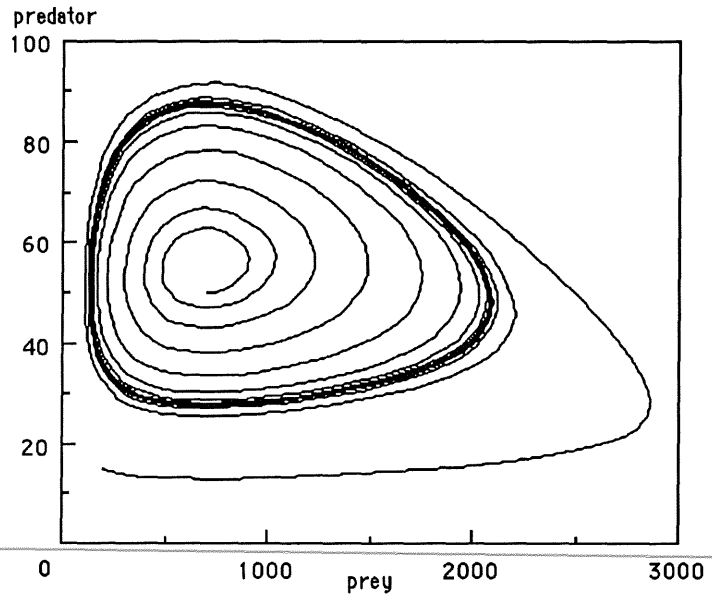


Figure 1. Stable limit cycles in predator-prey space generated by the simulation program in Table 6 and TTUTIL routines. The graph shows trajectories starting from two different initial points. Both lines approach the same limit cycle.

called stable limit cycle. The predator and prey populations oscillate. Amplitude and frequency of the oscillation are independent of the initial conditions.

In the model routine prey density was chosen as the independent output variable. That leads to an output table containing PREY and PRED as two columns (without TIME). Plotting that result leads to the curves in Figure 1. The data files specify two model runs with different initial conditions. The model parameters were taken from Roughgarden (1979) and the result in Figure 1 accurately corresponds to his Figure 22.7.

3.2.4. *Coupling different models*

When different models have to be coupled, each having the structure of the example model in Table 6, the procedure is somewhat more complex than for the EULER integration, where the driver can be easily adapted for performing calls to all models.

When a separate integration routine like **RKQCA** is used, however, only a single STATE array, a single RATE array and a single SCALE array exist. Hence, the arrays of the different models need to be combined in a single "super model" that is positioned between the driver and the actual model routines. The supermodel has the structure of an ordinary model subroutine and its only function is to subdivide up the full state arrays into contributions from the various models. The calls from the driver and the integration routine are then translated into calls to the different models.

Variables expressing the interaction between the different models can be added to the parameter lists of the model subroutines, since the separate models do not need to be in standard form anymore (only the "supermodel" has to be). A more elegant way is probably the use of small common blocks containing interaction variables. Then each submodel can still be used as an independent model subroutine. Together, with a "supermodel" above them, they interact.

3.2.5. *The variable KEEP*

The common block /INFO/ contains the variable KEEP which has a special function. Each integration step requires many rate calculations and the model is often called by the integration routine **RKQCA** (or in fact by **RK4A**, at an even lower level). Before starting an integration step, however, the driver **RKDRIV** calls the model once. That call is needed for several reasons. Sometimes the output flag OUTPUT is set by the driver, requesting the model not only to calculate rates but also to produce output. Moreover, the integration procedure requires valid rates of change at its starting point. Therefore the driver always has to call the model before integration. During that call the variable KEEP is equal to 1. After returning from the model, the driver sets KEEP back to 0 again. When KEEP=1, the model may calculate new values for discontinuous functions. Or, in complicated cases, it may change the values of its state variables to account for fast subprocesses that are not described by the differential equations incorporated in the model.

3.3. Euler integration of differential equations

The driver for Runge-Kutta integration can be easily adapted for Euler integration. The call to the integrating subroutine **RKQCA** is replaced by a DO-loop in which new states are calculated according to the simple Euler algorithm. The result is subroutine **EUDRIV**. This subroutine differs in two ways from the Euler driver for difference equations discussed above (the driver in Appendix D). Subroutine **EUDRIV** calls the same type of model subroutines as **RKDRIV**. Hence, it makes use of the STATE and RATE arrays and even of the array SCALE although this last one is not used in the integration algorithm. That implies that one can switch between Euler integration and Runge-Kutta integration by simply calling either **EUDRIV** or **RKDRIV** in the main program.

A second difference with the Euler driver in Appendix D is that **EUDRIV** adapts its timestep when an output time is approached. With DELT=2 and PRDEL=3, for instance, the successive timesteps are 2,1,2,1,2,1.... In case of the solution of difference equations the timestep has to be kept fixed. Hence, **EUDRIV** applies the Euler algorithm to differential equations while the driver in Appendix D (and the drivers discussed in Van Kraalingen (1991)) solve difference equations.

3.4. Concluding remark

The utilities for input and output of the TTUTIL library can be used in combination with a Runge-Kutta integration algorithm or with any other integration algorithm. In a simulation model the system description is preferably separated from the integration algorithm. Usually that will require the use of a separate integration routine by which the model is called. Also programs for solving partial differential equations, for instance models for water transport in the soil, can probably be substantially improved by writing a more abstract routine that takes care of the integration and separating it from the solved equations. Examples of an elegant program structure can be found in the IMSL library (1987).

4. Headers of the library routines

The headers consist of the following parts:

- The SUBROUTINE or FUNCTION statement.
- A short description of the function of the subprogram.
- A list of formal parameters with their meaning. The parameters are classified as input (I) or output (O) parameters.
- A list of called subroutines and/or functions. The lists give the names of all required subprograms. Only other TTUTIL routines are called.
- The author(s) and the date on which the last changes were made.
- In some cases additional explanation.
- The declarations of the formal parameters of the subprogram. They give the datatype of each parameter and the array lengths. In case of doubt the declarations are decisive.

```
      REAL FUNCTION BOXMUL()

*     Generates unit normal deviate by Box-Muller method

*     BOXMUL - pseudo-random standard normal deviate                    O
*
*     Subroutines and/or functions called:
*      - from library TTUTIL: UNIFL

*     Author: Kees Rappoldt
*     Date  : October 1989

*     Some remarks:
*     The Box-Muller (1958) method is based on inversion. No
*     inverse exists of the distribution of a single normal
*     variate However SQRT(X**2+Y**2) with X and Y both being
*     normal variates, has a distribution which can be inverted.
*
*     This enables elegant generation of normal variates
*     in combination with a generator for uniform variates
*     on (0,1). The commonly used linear congruential
*     generators, however, lead to pathological behaviour
*     (see section 6.7.3 in Bratley (1987)). This is not
*     the case with the generator UNIFL used in this program.
*
*     References:
*     Box, G.E.P., and M.E.Muller. (1958). A note on the
*         generation of random normal deviates.
*         Ann.Math.Stat. 29:610-611.
*     Bratley,P., B.L.Fox and L.E.Schrage. 1983. A guide to
*         simulation. Springer-Verlag New York Inc. 397 pp.


*     no formal parameters
```

```
      SUBROUTINE CHKTSK (MODULE, IULOG, ITOLD, ITASK)

*     The function of this routine is to check the new task
*     and previous task of simulating subroutines.

*     MODULE - Name of module from which CHKTSK is called  I
*     IULOG  - Log file where to write errors to           I
*     ITOLD  - Value of previous task                      I
*     ITASK  - Value of new task                           I
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR

*     Author: Daniel van Kraalingen
*     Date  : December 1989


*     formal parameters
      INTEGER IULOG, ITOLD, ITASK
      CHARACTER*(*) MODULE
```

```
      SUBROUTINE  CLS

*     This subroutine clears the screen (IBM only)
*
*     Author: Daniel van Kraalingen
*     Date:   Oct 1989
*
*     No subroutines called

*     no formal parameters
```

```
      SUBROUTINE  COPFIL  (IIN,  FILE,  IOUT)

*     Copies contents of file with unit number IIN and name FILE
*     to output file with unit number IOUT (the output file
*     should already be open and is left open. The input file is
*     closed after the contents have been copied.

*     IIN  - Unit number to be used to open input file with          I
*     FILE - File name of input file                                 I
*     IOUT - Unit number of file where input file should be copied   I
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: FOPENG, ERROR, ILEN, UPPERC

*     Author: Daniel van Kraalingen
*     Date  : November 1990


*     formal parameters
      INTEGER IIN, IOUT
      CHARACTER*(*) FILE
```

```
      LOGICAL FUNCTION DECCHK (STRING)

*     Checks if string STRING is a number.

*     DECCHK --> .TRUE. if the string is a number                    O
*     STRING - input string, NO trailing or leading blanks !         I
*
*     No subroutines and/or functions called

*     Author: Kees Rappoldt
*     Date  : September 1989

*     formal parameter
      CHARACTER*(*) STRING
```

```
      SUBROUTINE  DECINT  (IWAR,STRING,IVALUE)

*     Decodes an integer number from a character string

*     IWAR   - In case of error IWAR = 1, otherwise IWAR = 0        O
*     STRING - input string, NO trailing or leading blanks !       I
*     IVALUE - Integer value read from string                      O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK

*     Author: Kees Rappoldt
*     Date  : September 1989


*     formal parameters
      INTEGER IWAR,IVALUE
      CHARACTER*(*) STRING
```

```
      SUBROUTINE  DECREA  (IWAR,STRING,VALUE)

*     Decodes a real number from a character string

*     IWAR   - In case of error IWAR = 1, otherwise IWAR = 0        O
*     STRING - input string, NO trailing or leading blanks !       I
*     VALUE  - Real value read from string                         O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK

*     Author: Kees Rappoldt
*     Date  : September 1989


*     formal parameters
      INTEGER IWAR
      REAL VALUE
      CHARACTER*(*) STRING
```

```
      SUBROUTINE  DECREC  (RECORD,ILX,X)

*     Locates and decodes from the character string RECORD
*     (the first) ILX real numbers.
*     Numbers are separated by blanks(s) and/or comma(s).

*     RECORD - character string ; numbers max 20 characters      I
*     ILX    - number of REAL numbers to be decoded              I
*     X      - array containing the results                      O
*
*     Subroutines and functions called:
*     - from library TTUTIL: DECREA, ERROR, DECCHK

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      INTEGER ILX
      CHARACTER*(*) RECORD
      REAL X
      DIMENSION X(ILX)
```

```
      SUBROUTINE  ENTCHA  (QUEST,X)

*     Interactive entry of a character string.
*     Writes the text QUEST on screen as a "question" and
*     returns entered string to calling program.

*     QUEST - character string, for instance 'name'                I
*     X     - entered character string                             O
*
*     No subroutines and/or functions called

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      CHARACTER QUEST*(*),X*(*)
```

```
      SUBROUTINE  ENTDCH  (QUEST,SDEF,S)

*     Interactive entry of a CHARACTER string with a default.
*     Writes the text QUEST on screen as a "question" and
*     returns entered string to calling program.

*     QUEST - character string, for instance 'the value of P'      I
*     SDEF  - default string, assumed when <Return> is given       I
*     S     - entered CHARACTER string                             O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ILEN, ISTART

*     Author: Kees Rappoldt
*     Date: October 1989


*     formal parameters
      CHARACTER*(*) QUEST,SDEF,S
```

```
      SUBROUTINE  ENTDIN  (QUEST,IXDEF,IX)

*     Interactive entry of an INTEGER number with a default.
*     Writes the text QUEST on screen as a "question" and
*     returns entered number to calling program.

*     QUEST - character string, for instance 'the value of P'      I
*     IXDEF - default value assumed when <Return> is given         I
*     IX    - entered INTEGER number                               O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ILEN, ISTART

*     Author: Kees Rappoldt
*     Date  : October 1989

*     formal parameters
      INTEGER IXDEF,IX
      CHARACTER*(*) QUEST
```

```
      SUBROUTINE  ENTDRE  (QUEST,XDEF,X)

*     Interactive entry of a REAL number with a default. Writes the
*     text QUEST on screen as a "question" and returns entered number
*     to calling program.

*     QUEST - character string, for instance 'the value of P'        I
*     XDEF  - default value assumed when <Return> is given           I
*     X     - entered REAL number                                    O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ILEN

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      REAL XDEF,X
      CHARACTER*(*) QUEST
```

```
      SUBROUTINE  ENTINT  (QUEST,IX)

*     Interactive entry of an INTEGER number
*     Writes the text QUEST on screen as a "question" and
*     returns entered number to calling program.

*     QUEST - character string, for instance 'number N'        I
*     IX    - entered number                                   O
*
*     No subroutines and/or functions called

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      INTEGER IX
      CHARACTER QUEST*(*)
```

```
      SUBROUTINE  ENTREA  (QUEST,X)

*     Interactive entry of a REAL number.
*     Writes the text QUEST on screen as a "question" and
*     returns entered number to calling program.

*     QUEST - character string, for instance 'the value of P'        I
*     X     - entered REAL number                                    O
*
*     No subroutines and/or functions called

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      REAL X
      CHARACTER QUEST*(*)
```

```
      SUBROUTINE  ERROR  (MODULE,MESSAG)

*     Writes an error message to the screen and holds the
*     screen until the <RETURN> is pressed.
*     Then execution is terminated.

*     MODULE  - string containing the module name                    I
*     MESSAG  - string containing the message                        I
*
*     No subroutines and/or functions called

*     Author: Daniel van Kraalingen
*     Date  : October 1989


*     formal parameters
      CHARACTER*(*) MODULE, MESSAG
```

```
      SUBROUTINE  EXTENS  (FILEIN,NEWEXT,ICHECK,FILEOU)

*     Changes extension of filename. Output filename is filled
*     with characters of input filename and new extension until
*     end is reached. Output filename is in uppercase characters.
*     The old extension is the part of the filename that follows
*     a dot (.). A dot before a bracket (]) is neglected (VAX).
*     The input filename does not necessarily have an extension.

*     FILEIN - Input filename with old or without extension          I
*     NEWEXT - New extension ; is set to uppercase                   I
*     ICHECK = 1 --> check on equal output and input extension       I
*            = 0 --> no check
*     FILEOU - Output filename with new extension in uppercase       O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR, ILEN, UPPERC

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      INTEGER ICHECK
      CHARACTER*(*) FILEIN,FILEOU,NEWEXT
```

```
      SUBROUTINE  EUDRIV  (IUL,IUD,IUM,MODEL)

*     Solves an initial value problem with the simple Euler method.
*     This driver routine initializes the model, reads a control
*     file TIMER.DAT and drives the user supplied model until the
*     finish time FINTIM given in TIMER.DAT is reached.

*     IUL   - logfile unit number                                  I
*     IUD   - first of two free unit numbers used by this driver   I
*     IUM   - first of a series of free unit numbers for model     I
*     MODEL - external model routine                               I
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA,  ERROR, EXTENS,
*                            FOPENG, IFINDC,   ILEN, ISTART, RDDATA,
*                            RDINDX, RDINIT, RDSREA, UPPERC

*     Author: Kees Rappoldt
*     Date  : October 1990


*     The user supplied routine MODEL:
*     The differential equations are actually contained in the user
*     supplied subroutine MODEL which is called by this driver as:
*       CALL MODEL (ITASK,OUTPUT,TIME,STATE,RATE,SCALE,NDEC,NEQ)
*     Note that the user routine may have an arbitrary name which is
*     given as an EXTERNAL in the CALL to this driver EUDRIV. The
*     action of the user supplied model subroutine depends on the
*     value of ITASK in the following way.
*     ITASK = 1 The model is initialized. The number of state variables
*               (differential equations) NEQ is set. Model parameters are
*               set or are read from file. Time and states are set to their
*               initial values. Also the corresponding scales have to be
*               set. The scale array SCALE contains the order of magnitude
*               of each state variable in STATE. A scale needs to be a
*               positive number, for instance 0.001, 0.5 or 30000.0. Also
*               output should be initialized (files opened, headers etc.)
*     ITASK = 2 Values in the STATE array and the current TIME are used
*               to calculate rates of change for each status variable.
*               In order to prevent confusion it is advised to use local
*               and more meaningfull names for state variables than just
*               the input array elements STATE(1), STATE(2), etc. Then,
*               at first, the state array is copied into the local
*               variables, then the rates are calculated which are finally
*               copied into the output array RATE.
*     ITASK = 4 Terminal call to the model. Final output may have to be
*               generated, files closed, etc.
*
```

```
*     Some CALL's with ITASK=2 ("rate calls") take place with the logical
*     OUTPUT set to .TRUE. Then the user supplied model is allowed to
*     produce output to file and/or screen. The period between successive
*     output times is PRDEL, a variable read from the control file TIMER.DAT.
*
*     Information on the variables in common block /INFO/ is given in the
*     header of subroutine RKDRIV. The same common block is contained
*     in this routine, although all rate call's are done with KEEP=1.
*     Hence, both EUDRIV and RKDRIV can be linked to a main program in
*     which a choice can be made between the integration methods. The block
*     /INFO/ also contains the unit number of an opened logfile and the
*     first of a series of free unit numbers that can be used by the model.
*     Further /INFO/ contains a logical TERMNL. This logical can be set
*     to .TRUE. by the model routine at any moment. It causes the
*     termination of the current simulation run.


*     formal parameters
      INTEGER IUL,IUD,IUM
      EXTERNAL MODEL

*     common /INFO/
      INTEGER IULOG, IUMOD, KEEP
      LOGICAL TERMNL
      COMMON /INFO/ IULOG,IUMOD,KEEP,TERMNL
```

```
      SUBROUTINE  FOPEN  (IUNIT,FILE,STATUS,PRIV)

*     Opens a sequential, formatted file
*     after doing an inquiry about the existence.

*     IUNIT  - unit number used to open file                              I
*     FILE   - name of the file to be opened                              I
*     STATUS - status of the file                                         I
*     PRIV   - privilege ; in case status='new' and file exists:          I
*              = 'del' --> old file is overwritten
*              = 'nod' --> old file saved, program stopped
*              = 'unk' --> interactive choice
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR, ILEN, UPPERC

*     Author: Daniel van Kraalingen, Kees Rappoldt
*     Date  : July 1990

*     formal parameters
      INTEGER IUNIT
      CHARACTER*(*) FILE, STATUS, PRIV
```

```
      SUBROUTINE  FOPENG  (IUNIT,FILE,STATUS,TYPE,IRECL,PRIV)

*     Opens formatted, unformmated or binary
*     files with sequential or direct access.

*     IUNIT  - unit number used to open file                          I
*     FILE   - name of the file to be opened                          I
*     STATUS - status of the file                                     I
*     TYPE   - string containing code for FORM keyword (F,U or B)     I
*              and code for ACCESS keyword (S or D).
*     IRECL    record length of direct access files. Parameter        I
*              is dummy in case of sequential files.
*     PRIV   - privilege ; in case status='new' and file exists:      I
*              = 'del' --> old file is overwritten
*              = 'nod' --> old file saved, program stopped
*              = 'unk' --> interactive choice
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR, ILEN, UPPERC

*     Author: Kees Rappoldt
*     Date  : July 1990


*     formal parameters
      INTEGER IUNIT,IRECL
      CHARACTER*(*) FILE, STATUS, PRIV, TYPE
```

```
      REAL FUNCTION GAMMA (ALFA,BETA)

*     Generates a gamma distributed pseudo random variate. The gamma
*     distribution has two parameters, ALFA and BETA. This generator
*     works for 0.0 < ALFA <= 1.0 only (checked !).

*     ALFA  - Shape parameter of the distribution                  I
*     BETA  - Scaling parameter of the distribution                I
*     GAMMA - Function name, gamma variate                         O
*
*     Subroutines and functions called:
*     - from library TTUTIL: ERROR, UNIFL

*     Author: Kees Rappoldt
*     Date  : January 1990


*     This pseudo random generator is fully based on FUNCTION RGS
*     in the second edition (1987) of:
*     Bratley, P., B.L. Fox and L.E. Schrage. 1983. A guide
*        to simulation. Springer Verlag. New York. 397 pp.
*     The original function RGS (in Appendix L of the book) depends
*     on the parameter ALFA only. This function GAMMA is defined as:
*        GAMMA (ALFA, BETA)  =  BETA * RGS (ALFA)
*     So the result of the original program is multiplied by BETA
*     in this routine in order to simplify the scaling of the
*     generated gamma variates.
*     Further, the program has been adapted for the use of the UNIFL
*     function for the generation of uniformly distributed variates.


*     formal parameters
      REAL ALFA,BETA
```

```
      SUBROUTINE  GETCH  (IUNIT,INCHAR,CHARS,EOF)

*     Reads INCHAR characters from a sequential file.
*     Resets itself during the first CALL, on an End_Of_File
*     condition and when a new unit number is used.

*     IUNIT  - unit of opened file used for reading                I
*     INCHAR - number of characters to be read                     I
*     CHARS  - characters read from file                           O
*     EOF    - End_Of_File found ; CHARS may not be complete       O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR, ILEN

*     Author: Kees Rappoldt
*     Date  : January 1989


*     formal parameters
      INTEGER IUNIT,INCHAR
      CHARACTER*(*) CHARS
      LOGICAL EOF
```

```
      SUBROUTINE  GETREC  (IUNIT,RECORD,EOF)

*     Reads record from an open file skipping comment lines.
*     Comment lines have an asterisk (*) in their first
*     or second (!!) column (with a space in the first).

*     IUNIT  - unit of opened file used for reading      I
*     RECORD - returned record                           O
*     EOF    - End_Of_File found                         O
*
*     No subroutines and/or functions are used

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      INTEGER IUNIT
      CHARACTER RECORD*(*)
      LOGICAL EOF
```

```
      INTEGER  FUNCTION  IFINDC  (NAMLIS,ILDEC,IST,IEND,NAME)

*     Finds number of name in a list with names ; when
*     name is not in the list a zero value is returned
*     Character strings should be of the same length !!

*     IFINDC - element where a match was found                    O
*     NAMLIS - character string array, the "list"                 I
*     ILDEC  - declared length of array NAMLIS                    I
*     IST    - array element where search should start            I
*     IEND   - actual size of the list with names                 I
*     NAME   - name to be found in the list                       I
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR

*     Author: Kees Rappoldt
*     Date  : September 1989


*     formal parameters
      INTEGER ILDEC, IST, IEND
      CHARACTER*(*) NAMLIS, NAME
      DIMENSION NAMLIS(ILDEC)
```

```
      INTEGER FUNCTION ILEN (STRING)

*     Determines the significant length of a string.
*     If the string is empty a zero is returned.

*     ILEN   - returned length          O
*     STRING - input string             I
*
*     No subroutines and/or functions called

*     Author: Daniel van Kraalingen
*     Date  : October 1989


*     formal parameter
      CHARACTER*(*) STRING
```

```
      REAL FUNCTION INSW (X1,X2,X3)

*     Input switch depending on sign of X1 ;
*     function is equivalent to the CSMP-INSW.

*     INSW - returned value                                    O
*     X1   - identifier upon which the test is done            I
*     X2   - value of INSW in case X1 <  0,                    I
*     X3   - value of INSW in case X1 >= 0,                    I
*
*     No subroutines and/or functions used

*     Author: Daniel van Kraalingen
*     Date  : October 1989


*     formal parameters
      REAL X1,X2,X3
```

```
      REAL FUNCTION INTGRL (STATE, RATE, DELT)

*     Function value = STATE + RATE * DELT.

*     INTGRL - function name, new state          O
*     STATE  - old state                         I
*     RATE   - rate of change per unit time      I
*     DELT   - time step                         I
*
*     No subroutines of functions called

*     Author: Daniel van Kraalingen
*     Date  : December 1989


*     formal parameters
      REAL STATE, RATE, DELT
```

```
      INTEGER FUNCTION ISTART (STRING)

*     Determines the first significant character of a string.
*     If the string contains no characters, a zero is returned.

*     ISTART - returned value          O
*     STRING - input string            I
*
*     No subroutines and/or functions called

*     Author: Daniel van Kraalingen
*     Date  : October 1989


*     formal parameter
      CHARACTER*(*) STRING
```

```
      REAL FUNCTION LIMIT (MIN,MAX,X)

*     Returns value of X limited within the interval [MIN,MAX]

*     MIN - interval lower boundary    I
*     MAX - interval upper boundary    I
*     X   - Argument of function       I
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR

*     Author: Kees Rappoldt
*     Date  : January 1990


*     formal parameters
      REAL MIN,MAX,X
```

```
      REAL FUNCTION LINT (TABLE,ILTAB,X)

*     This function is a linear interpolation function. The
*     function also extrapolates outside the defined region in
*     case X is below or above the region defined by TABLE.
*     Extrapolation, however, results in a warning to the screen.

*     LINT  - function name, result of the interpolation     O
*     TABLE - a one-dimensional array with paired            I
*             data: x1,y1,x2,y2, etc.
*     ILTAB - number of elements in the array TABLE          I
*     X     - the value at which interpolation should        I
*             take place
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR

*     Author: Daniel van Kraalingen
*     Date  : January 1990

*     formal parameters
      INTEGER ILTAB
      REAL TABLE(ILTAB), X
```

```
      SUBROUTINE  MOFILP  (IUNIT)

*     Moves the file pointer across comment lines of data files
*     and puts the file pointer at the first non-comment record.
*     Comment lines have an asterisk (*) in their first or
*     second (!!) column (with a space in the first).

*     IUNIT - unit of opened file used for reading        I
*
*     No subroutines and/or functions called

*     Author: Daniel van Kraalingen
*     Date  : October 1989


*     formal parameter
      INTEGER IUNIT
```

```
      SUBROUTINE  OUTARR  (NAME,  ARRAY,  I1,  I2)

*     This function writes the contents of arrays to the OUTDAT routines.
*     See description of OUTDAT routines for further information.
*     Example:
*              CALL OUTDAT ('ABC',ABC,1,10)
*
*     This call writes the array elements ABC(1) to ABC(10) to
*     the OUTDAT routine which can then be used to generate an output
*     table.

*     NAME  - Name of array to be written                              I
*     ARRAY - Array itself                                             I
*     I1    - Array element where output should start                  I
*     I2    - Array element where output should finish                 I
*
*     Subroutines called:
*      - from library TTUTIL: OUTDAT, ILEN, FOPENG, IFINDC, ERROR,
*                             OUTCOM, UPPERC

*     Author: Daniel van Kraalingen
*     Date  : November 1990


*     formal parameters
      CHARACTER*(*) NAME
      INTEGER I1, I2
      REAL ARRAY(I1:I2)
```

```
      SUBROUTINE  OUTCOM  (STR)

*     Stores a text string which is written to
*     the output file generated by OUTDAT. A maximum
*     number of 25 strings of 80 characters can be stored.

*     STR - text string      I
*
*     Subroutines and/or functions called:
*     from library TTUTIL: IFINDC, ERROR, ILEN

*     Author: Daniel van Kraalingen
*     Date  : November 1990


*     Example:
*     CALL OUTCOM ('Potential production')
*     CALL OUTCOM ('and water limited production')
*     CALL OUTDAT (4, 0, 'Final output',0.)
*     both text strings will appear in the final output file.


*     formal parameters
      CHARACTER*(*) STR
```

```
      SUBROUTINE  OUTDAT  (ITASK,  IUNIT,  RN,  R)

*     Can be used to generate output files from within simulation models.
*     It should be initialized first, to define the name of the
*     independent variable and to set unit numbers (ITASK=1). The name
*     and value of the data are stored by calls with ITASK=2, supplying
*     the name and the value to OUTDAT. The output file is generated by a
*     call with ITASK=4, 5, or 6. These generate table output, spreadsheet
*     output and two column output, respectively.

*     ITASK - integer, can be 1, 2, 4, 5, or 6. 1 initializes the
*             subroutine, opens a temporary file for storage, and stores
*             the name of the independent variable and the unit number.
*             2 stores the name and value of the variable in a temporary
*             file. 4, 5, or 6 generate an output file.  After 4, 5, or 6
*             the subroutine can be used again by using ITASK=1.
*     IUNIT - unit number used for writing to output file. If the unit
*             defined during ITASK=1 is open this is used for output.
*             Otherwise a file 'res.dat' using that unit is created.
*             IUNIT+1 is used I/O to the temporary file.
*     RN    - string, name of variable, (up to 11 characters
*             will be used). If ITASK is 4, 5, or 6, this string
*             will be written to the output file as title
*             (not limited to 11 characters).
*     R     - value of variable (only effective at ITASK=2).
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ILEN, FOPENG, IFINDC, ERROR, OUTCOM, UPPERC

*     Author: Daniel van Kraalingen, Kees Rappoldt
*     Date  : November 1990


*     Example:
*
*     CALL OUTDAT (1,20,'TIME',0.)     initialization, TIME is
*                                      made independent variable
*     CALL OUTDAT (2,20,'TIME',TIME)   value of TIME is stored
*     CALL OUTDAT (2,20,'D(11)',D(11)) value of DTGA is stored
*
*     repeated calls to OUTDAT with ITASK=2
*
*     CALL OUTDAT (4,20,'Plottitle',0.)  generate output file with
*                                        table format
*     CALL OUTDAT (6,20,'plottitle',0.)  generate output file with
*                                        two column format
*     CALL OUTDAT (1,20,'TIME',0.)   initialize again
*     etc.
```

```
*       formal parameters
        INTEGER ITASK, IUNIT
        REAL R
        CHARACTER*(*) RN
```

```
      SUBROUTINE  OUTPLT  (ITASK,  RN)

*     Designed to be used in conjunction with OUTDAT, which is used
*     to write variable name and value to a temporary file. OUTPLT is
*     used to printplot a selection of the stored variables. By repeated
*     calls to the subroutine with ITASK=1, names of variable for which
*     the plot is wanted can be given to the subroutine.
*     By a call with ITASK=4, 5, 6, or 7, printplots are generated with
*     a width of 80 or 132 characters, either with individual scaling
*     or with common scaling (all variables scaled to the smallest and
*     largest value in the data set).

*     ITASK - integer, can be 1, 4, 5, 6, or 7. Repeated calls with a
*             1 instruct the routine to store variable names for
*             use in the printplot. Calls with 4, 5, 6, and 7 generate
*             the printplot with the variables as defined with ITASK=1.
*             4 means wide format, individual scale,
*             5 means wide format, common scale,
*             6 means small format, individual scale,
*             7 means small format, common scale,
*             If the unit defined during ITASK=1 of OUTDAT is open,
*             this is used for output. Otherwise
*             a file 'res.dat' with that unit is created.
*     RN    - string, name of variable, up to 11 characters will be
*             used. The value of the variable must have been stored
*             by previous calls to OUTDAT.
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ILEN, FOPENG, IFINDC, ERROR, UPPERC

*     Author: Daniel van Kraalingen
*     Date  : July 1990


*     Example:
*
*     CALL OUTPLT (1,'DTGA')          define DTGA to be plotted
*     CALL OUTPLT (1,'WSO')           define WSO to be plotted
*     CALL OUTPLT (5,'Plot title')    make printplot using
*                                     wide format, common scale


*     formal parameters
      INTEGER ITASK
      CHARACTER*(*) RN
```

```
      SUBROUTINE  PLTFUN  (IUNIT,FUN,ILF,ILINE,IMARK,FORM,LEGEND)

*     writes AFGEN/LINT table points to file in TTPLOT format
*       !!!!!!!!!!!! NO CHECKS ARE CARRIED OUT !!!!!!!!!!!

*     IUNIT  - unit of opened file used for writing                  I
*     FUN    - "AFGEN" table as defined by routine LINT              I
*     ILF    - length of table FUN (=twice the number of points)     I
*     ILINE  - line type (see TTPLOT documentation)                  I
*     IMARK  - marker type (see TTPLOT documentation)                I
*     FORM   - data pair FORMAT ; CHARACTER string,                  I
*              for example '(2F8.3)'
*     LEGEND - legend text ; CHARACTER string                        I
*
*     Subroutines and/or functions called:
*      - from library TTUTIL: ERROR

*     Author: Kees Rappoldt
*     Date  : October 1989

*     formal parameters
      INTEGER IUNIT,ILF,IMARK,ILINE
      REAL FUN
      DIMENSION FUN(2,ILF/2)
      CHARACTER*(*) FORM,LEGEND
```

```
      SUBROUTINE  PLTHIS  (IUNIT,HIST,ILH,ILINE,IMARK,FORM,
     $                     LEGEND,IDOWN)

*     Writes AFGEN/LINT table points to file in TTPLOT format
*     The table is drawn as a HISTOGRAM using the X values of
*     the table as mid-class points.

*     IUNIT  - unit of opened file used for writing                  I
*     HIST   - "AFGEN" table as defined by routine LINT              I
*     ILH    - array length of HIST                                  I
*     ILINE  - line type (see TTPLOT documentation)                  I
*     IMARK  - marker type (see TTPLOT documentation)                I
*     FORM   - data pair FORMAT ; CHARACTER string,                  I
*              for example '(2F8.3)'
*     LEGEND - legend text ; CHARACTER string                        I
*     IDOWN  =1 --> right side of interval is connected to x-axis    I
*            =0 --> this is not done
*
*     Subroutines and/or functions called:
*      - from library TTUTIL: ERROR

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      INTEGER IUNIT,ILH,ILINE,IMARK,IDOWN
      REAL HIST
      DIMENSION HIST(2,ILH/2)
      CHARACTER*(*) FORM,LEGEND
```

```
      SUBROUTINE  POS  (IX,IY,TEXT)

*     Positions text on VT100 screen

*     IX   - X-position of cursor (# of columns)                    I
*     IY   - Y-position of cursor (# of rows)                       I
*     TEXT - Text to be written on screen                           I
*
*     Functions called:
*      - from library TTUTIL: ILEN, ISTART

*     Author: Daniel van Kraalingen
*     Date  : October 1989


*     formal parameters
      INTEGER IX,IY
      CHARACTER*(*) TEXT
```

```
      SUBROUTINE  RDAREA  (XNAME,X,ILDEC,IFND)

*     Reads an array of REAL values from a data file,
*     that should be initialized with RDINIT.

*     XNAME  - Name of array, for which data are on file            I
*     X      - Array itself                                        O
*     ILDEC  - Declared length of X                                I
*     IFND   - Number of values found on file                      O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA,  ERROR, EXTENS,
*                            FOPENG, IFINDC,   ILEN, RDDATA, RDINDX,
*                            UPPERC

*     Author: Kees Rappoldt
*     Date  : November 1990

*     Example: The data file below contains values for
*     the variables A, B and ZTB:
*
*         * example
*         A = 3.4   ;   B = 5
*         * the following variable is an array
*         ZTB = 1.0, 3.4,
*               1.2, 4.8,
*               1.4, 5.9
*
*     With this routine one may read the array ZTB by:
*     CALL RDAREA ('ZTB',ZTB,100,ILZ).
*     The actual array length ILZ will get the value 6.
*     In the header of RDINDX a formal description of
*     the data file syntax can be found.
*     The lines of the data file are read until column 80.


*     formal parameters
      INTEGER ILDEC,IFND
      REAL X
      DIMENSION X(ILDEC)
      CHARACTER*(*) XNAME
```

```
      SUBROUTINE  RDDATA  (ITASK,IUNIT,IULOG,FILNAM,
     $                     IS,XNAME,X,NDEC,NREQ)

*     This is the central subroutine of the complete set of RD*
*     routines in the library TTUTIL. The simple routines RDSINT
*     RDSREA, RDAREA, RDSETS and RDFROM form user interfaces. The
*     actual work is done here. That implies that the headers of
*     all user interfaces also apply to this routine RDDATA. The
*     used ITASK values are:
*       ITASK = 1  <----  RDSETS
*       ITASK = 2  <----  RDFROM
*       ITASK = 3  <----  RDINIT
*       ITASK = 4  <----  RDSINT, RDSREA and RDAREA
*     Rerun and data files both are analysed by routine RDINDX
*     called for ITASK is 1 and 3.
*     Examples of data files can be found in the headers of
*     RDSETS, RDSINT, RDSREA and RDAREA. In the header of RDINDX
*     a formal description of the data file syntax can be found.

*     ITASK  - function control, see the header above                I
*     IUNIT  - Unit number used for the .TMP direct access file      I
*              in which values are written (ITASK=1 or 3).
*              IUNIT+1 is used for the data file itself, which is
*              closed after reading it.
*              Different unit numbers should be used in the calls to
*              RDSETS and RDINIT.
*     IULOG  - >0, Unit number of logfile used for data file syntax  I
*                  errors. When not opened RDINDX.LOG is created.
*              =0, Nothing is done with a logfile.
*              The unit number used with ITASK=1 (RDSETS call)
*              is stored and also used for a report about replaced
*              data file values during calls to RDSINT, RDSREA
*              and RDAREA.
*     FILNAM - Name of file (ITASK=1 or 3)                           I
*     IS     - For ITASK = 1: Number of sets, output                 I/O
*              For ITASK = 2: Requested set number, input
*     XNAME  - Name of variable, for which data are on file          I
*     X      - Variable itself, may be array                         O
*     NDEC   - Declared length of X, when not array use 1            I
*     NREQ   - For ITASK=2:                                          I/O
*                =0, non-used variables previous set --> logfile
*                >0, non-used variables --> fatal error
*              For ITASK=4:
*                =0, The number of values on file is returned
*                >0, Requested number of values, should match
*                    the number present on file (checked !!)
*
*     Subroutines and/or functions called:
```

```
*     - from library TTUTIL: DECCHK, DECINT, DECREA, RDINDX,  ERROR,
*                            EXTENS, FOPENG, IFINDC,   ILEN, UPPERC

*     Author: Kees Rappoldt
*     Date  : November 1990


*     ======= word size in bytes belonging to unformatted record =======
*     = on VAX this is 4 bytes/word; on ATARI, IBM and MAC 1 byte/word =
      INTEGER IWLEN
      PARAMETER (IWLEN=1)

*     formal parameters
      INTEGER ITASK,IUNIT,IULOG,IS,NDEC,NREQ
      REAL X
      DIMENSION X(NDEC)
      CHARACTER*(*) FILNAM,XNAME
```

```
      SUBROUTINE  RDFROM  (IS,FATAL)

*     Instructs RDDATA (and RDSINT, RDSREA and RDAREA) to use the
*     IS-th set from the rerun file. Note that set 0 (zero) means that
*     data file values are used. Selecting set 0 does not require
*     a previous call to RDSETS and set 0 may also be selected when
*     no rerun file exists or when it is empty.
*     Warnings are generated on non-used variables of the previous
*     set. If desired this may result in a fatal error (see FATAL).
*     Moving from set 0 to another set, no check is carried out.
*     The header of RDSETS contains further remarks.

*     IS    - =0, data file contents returned, disable replacement  I
*             >0, set number enabled
*     FATAL - .FALSE., non-used variables --> logfile               I
*             .TRUE., non-used variables --> fatal error
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA,  ERROR, EXTENS,
*                            FOPENG, IFINDC,   ILEN, RDDATA, RDINDX,
*                            UPPERC

*     Author: Kees Rappoldt
*     Date  : November 1990


*     formal parameters
      INTEGER IS
      LOGICAL FATAL
```

```
      SUBROUTINE  RDINDX  (IUNIT,SETS,TOSCR,TOLOG,IUL,DATFIL,
     $                     XBUF,ILBUF,IWLEN,NAMLIS,ILIND,INDPNT,
     $                     ILPNT,INFND,INSETS)

*     Produces an index of a data file. The index consists of a list of
*     variable names and an integer array pointing to decoded values
*     on a direct access file. The direct access file is opened
*     for reading at the end of the routine or deleted after errors.

*     IUNIT  - unit number used to open random access file for I/O   I
*              IUNIT+1 used to open data file (closed after reading).
*     SETS   - when .TRUE. more than a single set of values is       I
*              is allowed. The order of the names in all sets
*              should be identical.
*     TOSCR  - flag enabling error message output to screen          I
*     TOLOG  - flag enabling error message output to logfile         I
*     IUL    - unit number of logfile (when TOLOG is set)            I
*              if non-existent, RDINDX.LOG is created
*     DATFIL - name of data file                                     I
*     XBUF   - record buffer direct access file, overwritten        I/O
*     ILBUF  - array length XBUF, number of values on record         I
*     IWLEN  - word length in bytes for unformatted record           I
*     NAMLIS - list of variable names                                O
*     ILIND  - declared size of array NAMLIS                         I
*     INDPNT - points to value in temporary file                     O
*     ILPNT  - declared size of array INDPNT                         I
*     INFND  - number of variable names found                        O
*     INSETS - Number of sets in data file (0 when empty)            O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA,  ERROR, EXTENS,
*                            FOPENG, IFINDC,   ILEN,  UPPERC

*     Author: Kees Rappoldt
*     Date  : November 1990


*     The data should be present on file in the form "name = value(s)".
*     Such a string is called a field in the formal description below.
*     On each record (line) of a data file different fields may occur,
*     separated by a semi-colon.
*     A field may be continued on a following line. A name or an
*     individual number, however, should not be interrupted. Continuing
*     a series of numbers, the comma is expected before going to the
*     next line. On all lines comments may be present following an
*     exclamation mark (!). Lines beginning with an asterisk (*) are
*     ignored. Maximum name length is set by the variable type of NAMLIS.
*
```

```
*           record =   field {; field}
*            field =   name = number {, number}
*             name =   letter{letdig}
*           letdig =   < letter | digit >
*           number =   [repeat*]real
*           repeat =   INTEGER number
*             real =   REAL number (may be without decimal point)
*
*     When SETS is .TRUE. the data file may consist of more than a single
*     set of variable names and values. The order of the names in all
*     sets should be identical. In this case the pointer array INDPNT
*     contains INFND pointers for each set.
*     Examples of data files can be found in the headers of RDAREA, RDSREA,
*     RDSINT and RDSETS.


*     formal parameters
      INTEGER IUNIT,IUL,ILBUF,IWLEN,INDPNT,ILIND,ILPNT,INFND,INSETS
      REAL XBUF
      CHARACTER*(*) DATFIL,NAMLIS
      DIMENSION NAMLIS(ILIND),INDPNT(0:ILPNT),XBUF(ILBUF)
      LOGICAL SETS,TOSCR,TOLOG
```

```
      SUBROUTINE  RDINIT  (IUNIT,IULOG,DATFIL)

*     Initializes data file reading with the routines RDSINT,
*     RDSREA and RDAREA. An index of the data file is stored
*     as a local array containing variable names. The values
*     are written to a temporary file. After a call to RDINIT
*     the data file itself is closed again.
*     In the header of RDINDX a formal description of the
*     data file syntax can be found. For examples, see the
*     headers of RDSINT, RDSREA and RDAREA.

*     IUNIT  - unit number used to open random access file for I/O   I
*              IUNIT+1 used to open data file (closed after reading).
*     IULOG  - >0, Unit number of logfile used for data file syntax  I
*                  errors. When not opened RDINDX.LOG is created.
*              =0, Nothing is done with a logfile
*     DATFIL - Name of data file                                     I
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA,   ERROR, EXTENS,
*                            FOPENG, IFINDC,   ILEN,  RDDATA, RDINDX,
*                            UPPERC

*     Author: Kees Rappoldt
*     Date  : November 1990


*     formal parameters
      INTEGER IUNIT,IULOG
      CHARACTER*(*) DATFIL
```

```
      SUBROUTINE  RDSETS  (IUNIT,IULOG,SETFIL,INS)

*     Initializes subroutine RDDATA for reading data from a so called
*     "rerun file" containing sets of variable names with associated
*     values. The sets are used to replace corresponding data items in
*     a normal data file analysed with the routines RDINIT, RDSREA,
*     RDSINT and RDAREA.
*     This facility has a "global" character. A call to RDINIT does not
*     disable the replacement of values. So the sets in a
*     rerun file may contain variable names occurring in different
*     data files. Only a call to RDSETS with empty filename will
*     disactivate the replacement of numbers.
*     After a normal RDSETS call the 0-th set is activated meaning that
*     data file values are used. With RDFROM the sets of the rerun
*     file are actually activated.

*     IUNIT  - unit number used to open random access file for I/O   I
*              IUNIT+1 used to open rerun file (closed after reading).
*     IULOG  - >0, Unit number of logfile used for syntax error      I
*                  messages. When not opened RDINDX.LOG is created.
*              =0, Nothing is done with a logfile
*     SETFIL - Name of rerun file containing sets                    I
*              Empty --> number replacement is disactivated (INS=0)
*     INS    - Number of sets on file (when exists minimum 1)        O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA,   ERROR, EXTENS,
*                            FOPENG, IFINDC,   ILEN,  RDDATA, RDINDX,
*                            UPPERC

*     Author: Kees Rappoldt
*     Date  : November 1990


*     Example:
*     The following rerun file contains two sets of values for the
*     variables PAR1, PAR2 and ZTB:
*
*         * set 1
*         PAR1 = 3.4   ;   PAR2 = 5
*         ZTB = 1.0, 3.4,  1.2, 4.8,  1.4, 5.9
*         *
*         * set 2
*         PAR1 = 3.7   ;   PAR2 = 7
*         ZTB = 1.0, 3.1,  1.4, 9.8
*
*     After calling RDSETS, set 2 is activated by:
*     CALL RDFROM (2,FATAL). Reading then variable PAR1 from any
```

```
*     data file (just using RDINIT and RDSREA), the value from
*     the data file will be replaced by 3.7.
*     The lines of the data file are read until column 80.
*
*     The following statements form a typical application. The
*     rerun file RERUNS.DAT is assumed to contain sets of values
*     for parameter PAR1 of (sub)model A and parameter PAR2 of
*     (sub)model B. Both models make use of their own data file
*     containing also other variables. The structure below runs
*     the two models, at first for the values of PAR1 and PAR2 on
*     the data files, then for the sets in RERUNS.DAT. When the
*     file RERUNS.DAT is not there, a normal model run is made.
*
*     *     open logfile
*           CALL FOPEN (40,'LOGFILE.DAT','NEW','DEL')
*           CALL RDSETS (20,40,'RERUNS.DAT',INS)
*
*     *     at first the content of the data files is used (set 0)
*           DO 10 IS=0,INS
*              CALL RDFROM (IS,.TRUE.)
*              ........
*     *        statements occurring in model A:
*              CALL RDINIT (22,40,'MODELA.DAT')
*              CALL RDSREA ('PAR1',PAR1)
*              ........
*     *        statements occurring in model B:
*              CALL RDINIT (22,40,'MODELB.DAT')
*              CALL RDSREA ('PAR2',PAR2)
*              ........
*     10    CONTINUE
*
*     Note the following:
*     - The order of variables should be identical in all sets.
*     - When a non-zero logfile unit number is supplied, the usage of
*       values from the rerun file is reported. Also a change of
*       set number with RDFROM is reported and warnings are given
*       on non-used variables (of the previous set).
*     - In the above example the argument FATAL of RDFROM is set
*       to .TRUE. That leads to a fatal error on non-used variables
*       of the previous set (instead of a warning on logfile only)
*     - The (logfile) unit number used for the report is not
*       overwritten by the unit number in a RDINIT call.
*     - Values on the rerun file can only be used in combination
*       with a data file on which error free values occur for the
*       same variable. There is no access to the rerun file without
*       normal data file reading.
*     - After RDSETS "set 0" is selected meaning that the data file
*       values are not replaced. "Set 0" may also be selected just
*       using CALL RDFROM (0,FATAL).
```

```
*       formal parameters
        INTEGER IUNIT,IULOG,INS
        CHARACTER*(*) SETFIL
```

```
      SUBROUTINE  RDSINT  (XNAME,IX)

*     Reads a single INTEGER value from a data file.
*     The reading should be initialized with RDINIT.

*     XNAME  - Name of variable        I
*     IX     - Value of variable       O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA,   ERROR, EXTENS,
*                            FOPENG, IFINDC,   ILEN,  RDDATA, RDINDX,
*                            UPPERC

*     Author: Kees Rappoldt
*     Date  : November 1990


*     Example: The data file below contains values for
*     the variables A, IB and ZTB:
*
*         * example
*         A = 3.4   ;   IB = 5
*         * the following variable is an array
*         ZTB = 1.0, 3.4,
*               1.2, 4.8,
*               1.4, 5.9
*
*     With this routine one may read the value of IB by:
*     CALL RDSINT ('IB',IB).
*     Reading A results in the value 3 (the nearest integer
*     function is used). Reading the array ZTB with this
*     routines results in an error message. In the header of
*     RDINDX a formal description of the data file syntax
*     can be found.
*     The lines of the data file are read until column 80.


*     formal parameters
      INTEGER IX
      CHARACTER*(*) XNAME
```

```
      SUBROUTINE  RDSREA  (XNAME,X)

*     Reads a single REAL value from a data file.
*     The reading should be initialized with RDINIT.

*     XNAME  - Name of variable        I
*     X      - Value of variable       O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA,   ERROR, EXTENS,
*                            FOPENG, IFINDC,   ILEN,  RDDATA, RDINDX,
*                            UPPERC

*     Author: Kees Rappoldt
*     Date  : November 1990


*     Example: The data file below contains values for
*     the variables A, B and ZTB.
*
*         * example
*         A = 3.4   ;   B = 5
*         * the following variable is an array
*         ZTB = 1.0, 3.4,
*               1.2, 4.8,
*               1.4, 5.9
*
*     With this routine one may read the value of A by:
*     CALL RDSREA ('A',A).
*     Reading the array ZTB with this routines results in an
*     error message. In the header of RDINDX a formal description
*     of the data file syntax can be found.
*     The lines of the data file are read until column 80.


*     formal parameters
      REAL X
      CHARACTER*(*) XNAME
```

```
      REAL FUNCTION REAAND (X1, X2)

*     This function emulates the CSMP function AND.
*     REAl AND is similar to logical .AND. except that
*     arguments and results are REAL instead of LOGICAL
*     The definition of the function is:
*     REAAND = 1,  X1 > 0 and X2 > 0
*     REAAND = 0,  else

*     REAAND - Function result        O
*     X1     - first argument         I
*     X2     - second argument        I
*
*     No subroutines and/or functions used

*     Author: Daniel van Kraalingen
*     Date  : December 1989

*     formal parameters
      REAL X1, X2
```

```
      REAL FUNCTION REANOR (X1, X2)

*     This function emulates the CSMP function NOR.
*     REAl NOR is similar to logical expression
*     .NOT.(logical.OR.logical) except that
*     arguments and results are REAL instead of LOGICAL
*     The definition of the function is:
*     REANOR = 1  when  X1 <=0 and X2 <= 0
*     REANOR = 0  otherwise

*     REANOR - Function result          0
*     X1     - first argument           I
*     X2     - second argument          I
*
*     No subroutines and/or functions called

*     Author: Daniel van Kraalingen
*     Date  : November 1989

*     formal parameters
      REAL X1, X2
```

```
      SUBROUTINE  REMOVE  (STRING,CHR)

*     Replaces all unwanted characters (CHR)
*     in a string (STRING) with a <space>.
*     For example, if "e" is removed from
*     "bicentennial", the result is "bic nt nnial"

*     STRING - string that is used     I,O
*     CHR    - character to be removed  I
*
*     No subroutines and/or functions used

*     Author: Daniel van Kraalingen
*     Date  : October 1989


*     formal parameters
      CHARACTER STRING*(*),CHR*1
```

```
      SUBROUTINE  RK4A  (STATE,RATE,NDEC,NEQ,TIME,DELT,
     $                   STATE2,SCALE,MODEL)

*     Fourth order Runge Kutta integration over DELT
*     Adapted from routine RK4 from Press et al. (1986)

*     STATE  - state array of model                    I
*     RATE   - rates of change for (TIME,STATE)        I
*     NDEC   - declared size of input arrays           I
*     NEQ    - Number of state / rate variables        I
*     TIME   - time                                    I
*     DELT   - time step                               I
*     STATE2 - output state array                      O
*     SCALE  - size scale of state variables           I
*     MODEL  - external model called with ITASK=2      I
*              for rate calculation
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR

*     Author: Kees Rappoldt, adapted from Press et al. (1986)
*     Date  : February 1990


*     formal parameters
      INTEGER NDEC,NEQ
      REAL STATE,RATE,TIME,DELT,STATE2,SCALE
      DIMENSION STATE(NDEC),RATE(NDEC),STATE2(NDEC),SCALE(NDEC)
      EXTERNAL MODEL
```

```
      SUBROUTINE  RKDRIV  (IUL,IUD,IUM,MODEL)

*     Solves an initial value problem with the fourth order Runge Kutta
*     method described by Press et al. (1986). This driver routine
*     initializes the model, reads a control file TIMER.DAT and drives
*     the user supplied model until the finish time in TIMER.DAT is
*     reached.

*     IUL   - logfile unit number                                    I
*     IUD   - first of two free unit numbers used by this driver     I
*     IUM   - first of a series of free unit numbers for model       I
*     MODEL - external model routine                                 I
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: DECCHK, DECINT, DECREA, ENTDCH,  ERROR,
*                            EXTENS, FOPENG, IFINDC,   ILEN, ISTART,
*                            RDDATA, RDINDX, RDINIT, RDSREA,   RK4A,
*                             RKQCA, UPPERC

*     Author: Kees Rappoldt
*     Date  : October 1990


*     The user supplied routine MODEL:
*     The differential equations are actually contained in the user
*     supplied subroutine MODEL which is called by this driver as:
*       CALL MODEL (ITASK,OUTPUT,TIME,STATE,RATE,SCALE,NDEC,NEQ)
*     Note that the user routine may have an arbitrary name which is
*     given as an EXTERNAL in the CALL to this driver RKDRIV. The
*     action of the user supplied model subroutine depends on the
*     value of ITASK in the following way.
*     ITASK = 1 The model is initialized. The number of state variables
*               (differential equations) NEQ is set. Model parameters are
*               set or are read from file. Time and states are set to their
*               initial values. Also the corresponding scales have to be
*               set. The scale array SCALE contains the order of magnitude
*               of each state variable in STATE. A scale needs to be a
*               positive number, for instance 0.001, 0.5 or 30000.0. Also
*               output should be initialized (files opened, headers etc.)
*     ITASK = 2 Values in the STATE array and the current TIME are used
*               to calculate rates of change for each status variable.
*               In order to prevent confusion it is advised to use local
*               and more meaningfull names for state variables than just
*               the input array elements STATE(1), STATE(2), etc. Then,
*               at first, the state array is copied into the local
*               variables, then the rates are calculated which are finally
*               copied into the output array RATE.
*     ITASK = 4 Terminal call to the model. Final output may have to be
```

```
*                generated, files closed, etc.
*
*     Some CALL's with ITASK=2 ("rate calls") take place with the logical
*     OUTPUT set to .TRUE. Then the user supplied model is allowed to
*     produce output to file and/or screen. The period between successive
*     output times is PRDEL, a variable read from the control file TIMER.DAT.
*
*     At the start of a new time step (taken by the Runge Kutta routine
*     RKQCA), the state array STATE contains a valid (new) status of the
*     system. If anything has to be changed in the state array in order
*     to account for discontinuities, for instance, that should be done
*     at such moments. Therefore, the CALL's to MODEL at the beginning
*     a new time step are carried out with the common variable KEEP
*     equal to 1. Otherwise KEEP is 0.
*
*     This common variable is part of a small common block /INFO/. It
*     also contains the unit number of an opened logfile and the first
*     of a series of free unit numbers that can be used by the model.
*     Further /INFO/ contains a logical TERMNL. This logical can be set
*     to .TRUE. by the model routine at any moment. It causes the
*     termination of the current simulation run. Note that, after setting
*     TERMNL to .TRUE. a number of CALL's to MODEL will follow in order
*     to terminate the current time step and to produce final output.
*     Hence, when the flag TERMNL is set, it should never be reset by
*     the model. Then its status could be missed by the driver.


*     formal parameters
      INTEGER IUL,IUD,IUM
      EXTERNAL MODEL

*     common /INFO/
      INTEGER IULOG, IUMOD, KEEP
      LOGICAL TERMNL
      COMMON /INFO/ IULOG,IUMOD,KEEP,TERMNL
```

```
      SUBROUTINE  RKQCA  (STATE,RATE,NDEC,NEQ,TIME,DELTRY,
     $                    EPS,SCALE,DELDID,DELNXT,MODEL)

*     Runge Kutta integration with stepsize control.
*     Adapted from routine RKQC from Press et al. (1986)

*     STATE  - state array of model                    I/O
*     RATE   - rates of change for (TIME,STATE)        I/O
*     NDEC   - declared size of input arrays           I
*     NEQ    - Number of state / rate variables        I
*     TIME   - time                                    I/O
*     DELTRY - time step tried                         I
*     EPS    - relative accuracy of criterion          I
*     SCALE  - size scale of state variables           I
*     DELDID - time step taken                         O
*     DELNXT - advise for new step                     O
*     MODEL  - external model called with ITASK=2      I
*              for rate calculation
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ENTDCH, ERROR, ILEN, ISTART, RK4A

*     Author: Kees Rappoldt, adapted from Press et al. (1986)
*     Date  : February 1990


*     formal parameters
      INTEGER NDEC,NEQ
      REAL STATE,RATE,TIME,DELTRY,EPS,SCALE,DELDID,DELNXT
      DIMENSION STATE(NDEC),RATE(NDEC),SCALE(NDEC)
      EXTERNAL MODEL
```

```
      SUBROUTINE  STRIP  (STRING,CHR)

*     Strips unwanted characters (CHR) from a
*     string (STRING). The right hand side of the
*     string is adjusted, i.e. if "e" is stripped from
*     "bicentennial", the result is "bicntnnial"

*     STRING - string that is used     I,O
*     CHR    - character to be removed  I
*
*     Subroutines and/or functions called:
*      - from library TTUTIL: ILEN

*     Author: Daniel van Kraalingen
*     Date  : October 1989


*     formal parameters
      CHARACTER STRING*(*),CHR*1
```

```
      SUBROUTINE  TIMER  (ITASK,  DAYB,  DELT,  PRDEL,  FINTIM,
     $                    IYEAR,  TIME,  DAY,  IDAY,  TERMNL,  OUTPUT)

*     This subroutine updates TIME and related variables
*     each time it is called with ITASK=2. It will set TERMNL to
*     .TRUE. if FINTIM is reached.
*     The routine should be initialized first by a call
*     with ITASK=1. The first six arguments will then be made local.
*     Leap years are handled correctly.

*     ITASK  - task the routine should carry out (either 1 or 2)      I
*     DAYB   - start day of simulation
*              (1 <= DAY <= 365, 366 in leap years)                   I
*     DELT   - time step of simulation (multiple of 1 or
*              1/DELT = integer e.g. 0.25,1/3,0.5,1,2,3)              I
*     PRDEL  - Time between successive outputs (must be equal to
*              DELT or multiple)                                      I
*     FINTIM - Finish time of simulation (counted from start of
*              simulation !)                                          I
*     IYEAR  - start year with ITASK=1 and current year with
*              ITASK=2)                                              I/O
*     TIME   - time from start of simulation                          O
*     DAY    - Day number (REAL) of year=IYEAR                        O
*     IDAY   - Day number (INTEGER) of year=IYEAR                     O
*     TERMNL - Flag that indicates if FINTIM has been reached         O
*     OUTPUT - Flag that indicates if TIME is a multiple of PRDEL     O
*
*     Subroutines and/or functions called:
*     - from library TTUTIL: ERROR

*     Author: Daniel van Kraalingen
*     Date  : November 1990


*     formal parameters
      INTEGER ITASK, IYEAR, IDAY
      REAL DAYB, DELT, PRDEL, FINTIM, TIME, DAY
      LOGICAL TERMNL, OUTPUT
```

```
      REAL FUNCTION UNIFL()

*     High quality pseudo random number generator.

*     UNIFL - pseuso-random uniformly distributed variate              O
*
*     No subroutines and/or functions called

*     Author: Kees Rappoldt
*     Date  : October 1989


*     This pseudo random generator is fully based on FUNCTION UNIFL
*     in the second edition (1987) of Bratley et al. (see below).
*     A logical INIT has been added to the original program in order
*     to include seeds in the program (implicit initialization).
*
*     This generator is of the so called combined type. It does
*     not behave pathologically with the Box-Muller method for the
*     generation of normal variates, as do the commonly used linear
*     congruential generators (see also comments in FUNCTION BOXMUL).
*
*     The algorithm is:  X(i+1) = 40014 * X(i) mod (2147483563)
*                        Y(i+1) = 40692 * Y(i) mod (2147483399)
*                        Z(i+1) = (X(i+1)+Y(i+1)) mod (2147483563)
*     The random number returned is constructed dividing Z by its
*     range. The period of the generator is about 2.30584E+18.
*     The algorithm originates from L'Ecuyer (1986). In Bratley
*     et al.(page 332) more information can be found on seeds
*     and periods of X and Y.
*
*     References:
*     Bratley, P., B.L. Fox, L.E. Schrage. 1983. A guide to simulation
*         Springer-Verlag New York Inc. 397 pp.
*     L'Ecuyer,P. (1986). Efficient and portable combined pseudo-
*         random number generators. Commun. ACM (to appear).


*     no formal parameters
```

```
      SUBROUTINE  UPPERC  (STRING)

*     Converts string to uppercase characters

*     STRING - character string        I
*
*     No subroutines and/or functions called

*     Author: Daniel van Kraalingen, Kees Rappoldt
*     Date  : October 1989


*     formal parameter
      CHARACTER*(*) STRING
```

```
      SUBROUTINE  WORDS  (RECORD,ILW,SEPARS,IWBEG,IWEND,IFND)

*     Returns position of start and end of the (first) ILW words
*     in string RECORD. Valid separators are all the characters
*     present in string SEPARS, for instance ' ,' or '/ ,+=-'.

*     RECORD - character string                                  I
*     ILW    - number of words to be found                       I
*     SEPARS - string containing separator characters            I
*     IWBEG  - integer array containing start positions          O
*     IWEND  - integer array containing end positions            O
*     IFND   - integer containing the number of words            O
*
*     No subroutines and/or functions called

*     Author: Kees Rappoldt
*     Date  : October 1989


*     formal parameters
      INTEGER ILW,IWBEG,IWEND,IFND
      DIMENSION IWBEG(ILW),IWEND(ILW)
      CHARACTER*(*) RECORD,SEPARS
```

5 Examples

5.1. Testing a subroutine

The program below is a test program for subroutine ASTRO. The function of ASTRO is irrelevant. The input variables DAY and LAT are interactively entered with the current value used as the default. The tested subroutine can be executed with different values of its input parameters. Only modified values have to be typed in.

```
      PROGRAM EXAMP1

*     test subroutine ASTRO

      IMPLICIT REAL (A-Z)

*     initial defaults
      DAY = 1.0
      LAT = 52.5

10    CONTINUE

*     interactive input
      CALL ENTDRE ('Day number',DAY,DAY)
      CALL ENTDRE ('Latitude in degrees',LAT,LAT)

*     call to tested routine
      CALL ASTRO (DAY,LAT,DAYL,DAYLP,SINLD,COSLD)

*     write results
      WRITE (*,'(1X,A,F8.3,3(/,1X,A,F8.3))')
     $ 'DAYL  = ',DAYL,
     $ 'DAYLP = ',DAYLP,
     $ 'SINLD = ',SINLD,
     $ 'COSLD = ',COSLD

      GOTO 10
      END

*     The following include statement is used on the Apple MacIntosh.
*     On other machines the tested routine should be linked
*     to the test program or included with a slightly different
*     statement.
      INCLUDE HD40:FORTRAN:TTLIB:ASTRO.FOR
```

5.2. Data file transformation

Sometimes data have to be extracted from a file with a complicated structure. With an editor the work may be time-consuming. Importing such a file into a spreadsheet program may lead to complete chaos. Writing a FORTRAN program may be the solution. One is free in developing all kinds of criteria for the rejection or acceptation of "words" in the data file. In the example below, each line contains words (groups of characters separated by characters called "separators" ; see the header of **WORDS**). One expects that certain words, if present, are numbers. These numbers are written to an output file.

```
      PROGRAM EXAMP2

*     An example of data transformation
*     The user specifies two columns of an input file
*     The two columns need not exist on all lines
*     If they exist they should contain numbers
*     The numbers are written to an output file as
*     an X-value and an Y-value. A linear transformation
*     may be applied to the Y-values.

*     declarations
      INTEGER IWMAX,IFND,IWB,IWE,IX,IY,IWARX,IWARY
      PARAMETER (IWMAX=100)
      DIMENSION IWB(IWMAX),IWE(IWMAX)
      REAL X,Y,A,B
      CHARACTER MESSAG*40,FILIN*80,FILOUT*80,RECORD*80
      LOGICAL EOF

*     ask name input file
      CALL ENTCHA ('Filename',FILIN)
      CALL ENTDIN ('Column X',1,IX)
      CALL ENTDIN ('Column Y',2,IY)
      CALL ENTDRE ('Transform Y as A*Y+B, value of A',1.0,A)
      CALL ENTDRE ('And value of B',0.0,B)

*     change extension into 'OUT'
      CALL EXTENS (FILIN,'OUT',1,FILOUT)

*     open files
      CALL FOPEN (40,FILIN,'OLD','NVT')
      CALL FOPEN (41,FILOUT,'NEW','UNK')

*     read until EOF reached
20    CONTINUE
      CALL GETREC (40,RECORD,EOF)
      IF (.NOT.EOF) THEN
```

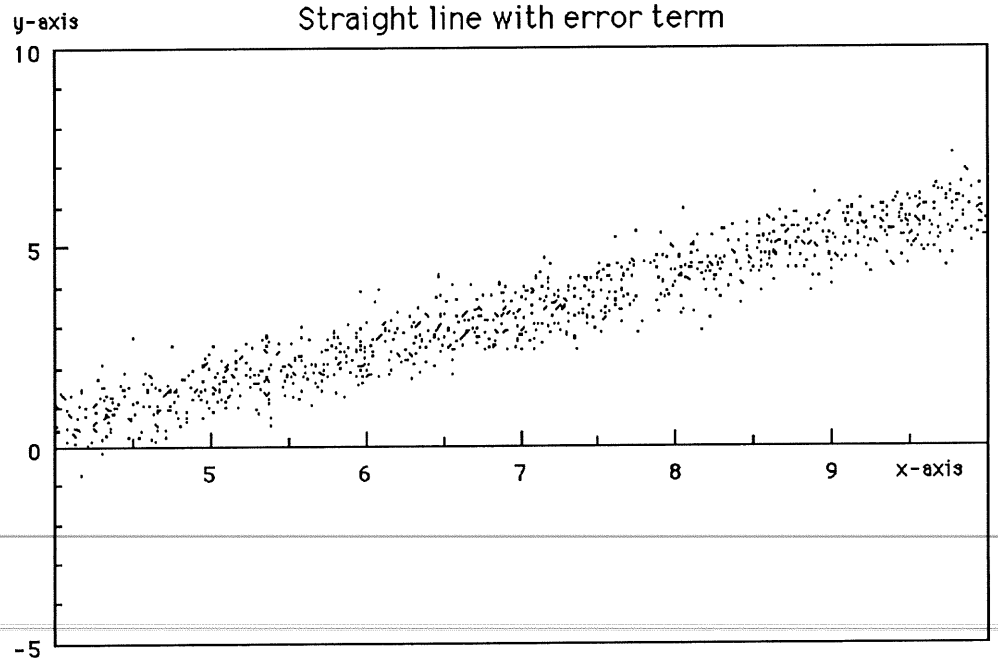
```
*        handle non-comment line
         CALL WORDS (RECORD, IWMAX, ' ,;=', IWB, IWE, IFND)

         IF (IFND.GE.IX .AND. IFND.GE.IY) THEN
*           decode IX-th and IY-th word
            CALL DECREA (IWARX, RECORD(IWB(IX):IWE(IX)), X)
            CALL DECREA (IWARY, RECORD(IWB(IY):IWE(IY)), Y)

            IF (IWARX.EQ.0 .AND. IWARY.EQ.0) THEN
*              correct values and write result to output
               WRITE (41,'(1X,2F12.4)') X, A*Y + B
            ELSE
*              error
               IF (IWARX.NE.0) MESSAG =
     $          RECORD(IWB(IX):IWE(IX))//' is not a number'
               IF (IWARY.NE.0) MESSAG =
     $          RECORD(IWB(IY):IWE(IY))//' is not a number'
               CALL ERROR ('EXAMP2',MESSAG)
            END IF
         END IF

      GOTO 20
      END IF

      STOP
      END
```

5.3. A scatter plot made with a random number generator

This example illustrates the call to the random number generator in the library. A number of (X,Y) points is generated and sorted. Sorting is carried out with help of the routine INDEXX in Press et al. (1986). The output is in two column format. A header is wrtten above the data containing some parameters used by the plotting program TTPLOT (Van Kraalingen and Rappoldt, 1988). That header is in no way essential, however.

```
      PROGRAM EXAMP3

*     Generates a scatter plot for Y = A * X + B + (error term)
*     The error term has a normal distribution with mean 0.0 and
*     standard deviation supplied by the user.
*     The x-values are randomly chosen on a user supplied interval.
*     The output file contains the calculated (X,Y) pairs with
*     the X-values in increasing order. A TTPLOT header is written
*     above the list of (X,Y) pairs.

*     declarations
      INTEGER I,INDX,N,NMAX
      REAL A,B,BOXMUL,SIGMA,UNIFL,X,Y
      PARAMETER (NMAX=10000)
      DIMENSION X(NMAX),Y(NMAX),INDX(NMAX)


*     ask input
      CALL ENTDRE ('Lower bound X interval', 4.0, X1)
      CALL ENTDRE ('Upper bound X interval', 10., X2)

      CALL ENTDRE ('Equation A*X+B, value of A',0.9,A)
      CALL ENTDRE ('and value of B',-3.0,B)

      CALL ENTDRE ('Standard deviation of error term',0.5,SIGMA)
      CALL ENTDIN ('Number of points',1000,N)

*     error check
      IF (N.GT.NMAX) CALL ERROR ('EXAMP3','N > NMAX')


*     generate points
      DO 10 I=1,N
         X(I) = X1 + (X2-X1) * UNIFL()
         Y(I) = A * X(I)  +  B  +  SIGMA * BOXMUL()
10    CONTINUE

*     sort points with respect to X value
      CALL INDEXX (N,X,INDX)
```

```
*     write results to file ; open file, write TTPLOT header
      CALL FOPEN (40,'RES.DAT','NEW','UNK')
      WRITE (40,'(A,9(/,A))')
     $ ' * Generated by EXAMP3.FOR',
     $ ' Straight line with error term',
     $ ' X 1 0 4 10 1 0.5 0 0',
     $ ' x-axis',
     $ ' Y 1 0 -5 10 5 1 0 0',
     $ ' y-axis',
     $ ' 0 0',
     $ ' *',
     $ ' 1 0 9',
     $ ' using BOXMUL'

      DO 20 I=1,N
         J = INDX(I)
         WRITE (40,'(1X,2F10.5)') X(J),Y(J)
20    CONTINUE

      STOP
      END

*     The following include statement is used on the Apple MacIntosh.
*     On other machines the required routine should be linked
*     to the test program or included with a slightly different
*     statement.
      INCLUDE HD40:FORTRAN:RECIPES:INDEXX.FOR
```



5.4. Checking a crop data file

In some crop growth simulation models so called partitioning factors are used. They describe the partitioning of the sugars produced by photosynthesis among the various plant organs. The partitioning factors are given on plant data files as "AFGEN tables" as a function of the development stage of the crop (see Appendix C). Evidently, the sum of all fractions has to be unity for all possible values of the development stage. The latter requirement is verified by this example program. It reads the relevant data from the data file used by a simulation model. The partitioning fractions may be somewhere in the data file.

```
      PROGRAM EXAMP4

*     Checks partitioning factors in a plant data file
*     Writes results to a TTPLOT file.

*     declarations
      INTEGER I,CROPT,ILMAX,ILFL,ILFS,ILFO,ILF
      REAL EFF,FLTB,FSTB,FOTB,LINT,FRL,FRS,FRO,TOT,SUM,DVS,FUNTB
      PARAMETER (ILMAX=100)
      DIMENSION FLTB(ILMAX),FSTB(ILMAX),FOTB(ILMAX),SUM(ILMAX)
      DIMENSION FUNTB(ILMAX)
      LOGICAL EXIST


*     initialize data reading
      CALL RDINIT (30,40,'Barley.dat')

*     read crop type, for demonstration only
      CALL RDSINT ('CROPT',CROPT)

*     read photosynthetic efficiency, for demonstration only
      CALL RDSREA ('EFF',EFF)

*     read partionioning tables
      CALL RDAREA ('FLTB',FLTB,ILMAX,ILFL)
      CALL RDAREA ('FSTB',FSTB,ILMAX,ILFS)
      CALL RDAREA ('FOTB',FOTB,ILMAX,ILFO)
      CLOSE (30,STATUS='DELETE')


*     check partitioning tables
      DO 10 I = 1, ILMAX/2
         DVS = 2.0 * FLOAT(I-1)/FLOAT(ILMAX/2-1)

         FRL = LINT (FLTB,ILFL,DVS)
         FRS = LINT (FSTB,ILFS,DVS)
         FRO = LINT (FOTB,ILFO,DVS)
```

```
          TOT = FRL + FRS + FRO

*         check
          IF (ABS(TOT-1.0).GT.0.001) WRITE (*,'(1X,A,F4.2)')
     $      'Partitioning error at DVS=',DVS

*         get sum table
          SUM(2*I-1) = DVS
          SUM(2*I)   = TOT
10     CONTINUE

*      write tables to TTPLOT plot file
       CALL FOPEN (50,'RES.DAT','NEW','UNK')
       WRITE (50,'(A,6(/,A))')
      $ ' * Generated by EXAMP4.FOR',
      $ ' Partitioning check',
      $ ' X 1 0 0 2 0.5 0.1 0 0',
      $ ' DVS',
      $ ' Y 1 0 0 1.4 0.2 0.1 0 0',
      $ ' Fraction',
      $ ' 0.1 1.3'

       CALL PLTFUN (50,FLTB,ILFL,2,7,'(1X,F4.2,F8.3)','leaves')
       CALL PLTFUN (50,FSTB,ILFS,3,8,'(1X,F4.2,F8.3)','stems')
       CALL PLTFUN (50,FOTB,ILFO,5,1,'(1X,F4.2,F8.3)','storage organs')
       CALL PLTFUN (50,SUM,ILMAX,1,0,'(1X,2F8.3)','total')

       STOP
       END
```

<begin of data file Barley.dat>

```
*
* Data file belonging to Example 4
* ================================
* Example of data file that can be read by the
* routines RDSINT, RDSREA and RDAREA.
*
* Variable names are followed by one or more values
* separated by a comma. Repetition of the same value
* may be coded by using 100*8.3, for instance (without
* spaces around the "*"). See the file below and
* a formal description in the header of RDINDX.
*
* Crop characteristics of Barley (Gerrie van de Ven, Egypt)
*
CROPT  = 2     ;     DSL = 2     ; AIRDUC = 0
SPAN   = 22.   ;   TBASE = 0.    ;  RDMCR = 125.
   DLO = 1.0   ;     DLC = 0.
   EFF = 0.40  ;    CFET = 1.00  ;  DEPNR = 4.5
```

```
* initial dry weight
TDWI = 100.0  ! in kg/ha

* initial rooting depth
RDI = 10.0    !  cm

* optimum development rates
DVRC1 = 0.034
DVRC2 = 0.058

* optimum rooting depth increase per day
RRI = 1.2

* extinction coefficient
KDIF = 0.6

* conversion factors
CVL = 0.72 ;  CV0 = 0.73 ;  CVR = 0.72 ;  CVS = 0.69

* partitioning
* ============
FRTB =  0.00,   0.70,    !  fraction to roots
        0.33,   0.50,
        1.00,   0.00,
        2.00,   0.00

FLTB =  0.00,   1.00,    !  fraction to leaves
        0.33,   1.00,
        0.80,   0.30,
        1.00,   0.00,
        2.00,   0.00

FSTB =  0.00,   0.00,    !  fraction to stems
        0.33,   0.00,
        0.80,   0.70,
        1.00,   0.50,
        1.01,   0.00,
        2.00,   0.00

FOTB =  0.00,   0.00,    ! fraction to storage organs
        0.80,   0.00,
        1.00,   0.50,
        1.01,   1.00,
        2.00,   1.00

* specific leaf area table
SLATB =  0.00,  0.0025,
         2.00,  0.0025

* development rate ; temperature dependence
DVRETB =  0.00,   0.00,
         35.00,   1.00,
         45.00,   1.00
```

<end of data file Barley.dat>

5.5. The influence of AMAX on crop assimilation

This example program shows that one may easily combine CALL's to routines from different libraries. For the calculations weather characteristics of a number of days in 1984 are needed. The program reads these data via the routines described in Van Kraalingen et al. (1990). It further performs calculations on photosysnthesis for two values of the variable AMAX (Spitters, Van Keulen and Van Kraalingen (1989)). Output tables and printplots are generated by the routines OUTDAT and OUTPLT from library TTUTIL.

```
      PROGRAM EXAMP5

*     This program compares cumulative assimilation over a gowing
*     season for two different AMAX values using a given
*     LAI as a forcing function. Radiation data are actual data.

      IMPLICIT REAL (A-Z)
      PARAMETER (IAR=20)
      REAL LAIT (IAR)
      INTEGER ILAIN, ISTAT, IDAY, IDAYS, IDAYE

      CALL RDINIT (30, 0, 'INPUT.DAT')
      CALL RDAREA ('LAIT', LAIT, IAR, ILAIN)
      CALL RDSREA ('AMAX1', AMAX1)
      CALL RDSREA ('AMAX2', AMAX2)
      CALL RDSREA ('EFF', EFF)
      CLOSE (30, STATUS='DELETE')

      CALL STINFO (11, 'HD40: WEER:DATA:', ' ', 'NL', 1, 1984,
     &             ISTAT, LONG, LAT, ALT, A, B)

      CALL OUTDAT (1, 40, 'DAY', 0.)

      IDAYS = NINT (LAIT(1))
      IDAYE = NINT (LAIT(ILAIN-1))

      DTGA1S = 0.
      DTGA2S = 0.
      DO 10 IDAY=IDAYS,IDAYE
         DAY = FLOAT (IDAY)
         CALL WEATHR (IDAY,ISTAT,
     &                RAD, TMIN, TMAX, VAPOUR, WIND, RAIN)
*        unit conversion
         RAD = RAD*1000.
*        LAI is a known function of day number
         LAI = LINT (LAIT, ILAIN, DAY)

         CALL DASS (DAY, LAT, RAD, 0.6, 0.2, LAI, AMAX1,
```

```
     $     EFF, DTGA1, DS0)
           DTGA1S = DTGA1S+DTGA
           CALL DASS (DAY, LAT, RAD, 0.6, 0.2, LAI, AMAX2,
     $     EFF, DTGA2, DS0)
           DTGA2S = DTGA2S+DTGA
     *     ratio of the two cumulative amounts
           D1D2 = DTGA1S/DTGA2S

     *     output
           CALL OUTDAT (2, 0, 'DAY', DAY)
           CALL OUTDAT (2, 0, 'RAD', RAD)
           CALL OUTDAT (2, 0, 'LAI', LAI)
           CALL OUTDAT (2, 0, 'DTGA1S', DTGA1S)
           CALL OUTDAT (2, 0, 'DTGA2S', DTGA2S)
           CALL OUTDAT (2, 0, 'D1D2', D1D2)
10    CONTINUE

     *    normal table output
          CALL OUTDAT (4, 0, 'Output van voorbeeldprogramma', 0.)

     *    printplot of two variables
          CALL OUTPLT (1, 'DTGA1S')
          CALL OUTPLT (1, 'DTGA2S')
          CALL OUTPLT (6, 'Output of example program')

     *    printplot of a single variable
          CALL OUTPLT (1, 'D1D2')
          CALL OUTPLT (6, 'Output of example program')

     *    table with a <TAB> between the columns (spreadsheet input !)
          CALL OUTDAT (5, 0, 'Output of example program', 0.)

     *    "TTPLOT" output
          CALL OUTDAT (6, 0, 'Output of example program', 0.)
          CALL OUTDAT (99, 0, ' ', 0.)

          STOP
          END

     *    The folllowing include statements are used on the Apple MacIntosh.
     *    On other machines the required routines should be linked
     *    or included with slightly different statements.
          INCLUDE HD40:FORTRAN:WEER.FOR
          INCLUDE HD40:FORTRAN:TTLIB:DASS.FOR
```

Appendix A: Non-standard statements and machine dependencies

The routines **CLS** and **POS** are intended specifically to handle output to a VT100 terminal or to the screen of an IBM PC and are not general output routines.

To hold the cursor after writing a question to the screen, a "$" is used in FORMAT strings in **ENTCHA**, **ENTDCH**, **ENTDIN**, **ENTDRE**, **ENTINT**, **ENTREA**, **FOPEN** and **POS**. If the syntax needs to be strictly standard or if the routines are used on an ATARI these "$"-characters need to be removed. They can be found be searching for the string "A$".

In **RDDATA** the integer **PARAMETER IWLEN** describes the length in bytes of the words used in records of unformatted direct access files. On the microcomputers ATARI, IBM and Apple this is 1 byte per word (the words are just bytes). On a VAX, however, a word consists of 4 bytes. The microcomputer version will run on the VAX, but the temporary files created by **RDDATA** and related routines will be 4 times larger than needed. A special VAX version (**IWLEN=4**) won't run on microcomputers, however. No error messages are given and results are unpredictable. Since the temporary files will under most circumstances be small, the microcomputer version is included in the library.

Appendix B: The common blocks in TTUTIL

Names of common blocks have to be unique across a program. In the library TTUTIL two common blocks occur. Their names are given below with the routines in which they are used. If you make use of these routines, or if the complete library is linked, other common blocks occurring in your program should be given different names.

COMMON **/OUTCUT/**	is declared in **OUTDAT**, **OUTPLT** and **OUTCOM**. The common block contains three variables that communicate file status and unit numbers between these three subroutines.
COMMON **/INFO/**	is declared in **EUDRIV** and **RKDRIV**. This block is meant to be declared in a user subroutine as well (cf. sections 3.2.1 and 3.2.5). It contains unit numbers and flags communicated between the user supplied subroutine (the "model routine") and the used "driver" routine **EUDRIV** or **RKDRIV**.

Appendix C: A note on AFGEN tables

Some of the programs in TTUTIL are intended to handle so-called AFGEN tables. An AFGEN table consists of a number of (X,Y) pairs. The values are stored in an array. The X-values in the array should be in increasing order. AFGEN tables are intended to be used by an interpolation routine. For linear interpolation, the function LINT is available in TTUTIL.

There are two ways of declaring an AFGEN table. The common method is to declare a one-dimensional array and to use the odd elements as X–values and the even elements as Y-values:

```
      INTEGER ILMAX
      PARAMETER (ILMAX=200)
      REAL TABLE (ILMAX)
```

The array TABLE has space for 100 data pairs. A DO-loop over the first 50 points looks like:

```
      DO 10 I=1,50
         X = TABLE (2*I-1)
         Y = TABLE (2*I)
         .....
10    CONTINUE
```

In some of the TTUTIL routines another method is used to declare an AFGEN table:

```
      INTEGER ILMAX
      PARAMETER (ILMAX=200)
      REAL TABLE (2,ILMAX/2)
```

The array is now two-dimensional. The element TABLE(1,I) is the I-th X value, the element TABLE(2,I) is the I-th Y value. The loop over the first 50 points now reads:

```
      DO 10 I=1,50
         X = TABLE (1,I)
         Y = TABLE (2,I)
         .....
10    CONTINUE
```

In some applications, the one-dimensional arrays lead may to awkward calculations with array indices. Using a two-dimensional array, the X- and Y-values are directly found from the point number. The two methods lead to the same values in the same places in computer memory. Therefore, both declaration methods can be used in combination with LINT and the other AFGEN table handling routines. Note that the array length should always be twice the number of (X,Y) pairs, the declaration as a two-dimensional array should always use the "2" as the first

index and the number of data pairs (the array length divided by 2) as the second.

The TTUTIL routines handling AFGEN tables all require the actual array length as an argument. So, it is good practice to always make a distinction between the declared length and the actual length, for instance by using ILFMAX and ILF as the declared and the actual length of the table TABLE. Reading data files, one should compare the value of ILF and ILFMAX, for instance by means of the statement:

```
IF (ILF.GT.ILFMAX) CALL ERROR ('TEST','ILF > ILFMAX')
```

Such error checks require a single statement only and are frequently used in all routines of TTUTIL. Subroutine **ERROR** handles fatal errors.

Using routine **RDAREA** for reading AFGEN tables from file, the above check can be omitted since it is carried out by **RDAREA**. The interpolation at X=2.1 in an AFGEN table TABLE present in file INPUT.DAT then reads:

```
INTEGER ILFMAX, ILF
PARAMETER (ILFMAX=200)
REAL Y,LINT, TABLE (200)

CALL RDINIT (40,50,'INPUT.DAT')
CALL RDAREA (' TABLE', TABLE,ILFMAX,ILF)
Y = LINT (TABLE,ILF,2.1)
```

See also Example program 4 and the routine headers of **RDINIT**, **RDAREA** and **LINT** for further documentation.

Appendix D: A simple driver for Euler integration

The use of Euler integration has been discussed in Chapter 3. An example model has been given and the structure of a driver has been discussed. Below, a full listing of the driver used is given. It is not part of the TTUTIL library. For crop growth simulation more luxurious drivers are available that take care of calendar and weather data (Van Kraalingen, 1991).

```
      SUBROUTINE DRIVER (IUDAT,IULOG,IUOUT)

*     Drives subroutine MODEL
*
*     IUDAT - first of two free unit numbers data file reading   I
*     IULOG - unit number of open logfile                        I
*     IUOUT - first of two free unit numbers for OUT routines    I

*     declarations
      IMPLICIT REAL (A-Z)
      INTEGER IUDAT,IULOG,IUOUT,IP
      LOGICAL OUTPUT,HALT,TERMNL
      SAVE

*     read timer variables ; temporary use of unit IUDAT
      CALL RDINIT (IUDAT,IULOG,'TIMER.DAT')
      CALL RDSREA ('STTIME',STTIME)
      CALL RDSREA ('FINTIM',FINTIM)
      CALL RDSREA ('PRDEL' ,PRDEL )
      CALL RDSREA ('DELT'  ,DELT  )
      CLOSE (IUDAT,STATUS='DELETE')
      IF (FINTIM.LT.STTIME) CALL ERROR ('DRIVER','FINTIM <= STTIME')
      IF (PRDEL.LE.0) CALL ERROR ('DRIVER','zero or negative PRDEL')
      IF  (DELT.LE.0) CALL ERROR ('DRIVER','zero or negative DELT')

*     initialize time
      TIME  = STTIME
*     initialize output
      CALL OUTDAT (1,IUOUT,'TIME',TIME)
*     initialize model(s)
      CALL MODEL (1,IUDAT,IULOG,.FALSE.,TIME,DELT,TERMNL)
*     initialize dynamic loop
      IP     = 0
      TNEXT  = STTIME
      TERMNL = .FALSE.
      HALT   = .FALSE.
```

```
*     dynamic loop (emulated DO WHILE)
10    IF (.NOT.HALT) THEN
*        output required ?
         OUTPUT = (TNEXT-TIME)/DELT.LT.0.01 .OR. TERMNL

*        write time to output ; rate call
         IF (OUTPUT) CALL OUTDAT (2,0,'TIME',TIME)
         CALL MODEL (2,IUDAT,IULOG,OUTPUT,TIME,DELT,TERMNL)

         IF (OUTPUT) THEN
*           get next output time ; leave dynamic loop ?
            IP     = IP + 1
            TNEXT = MIN (STTIME + IP * PRDEL, FINTIM)
            HALT   = (FINTIM-TIME)/DELT.LT.0.01 .OR. TERMNL
         END IF

         IF (.NOT.(HALT.OR.TERMNL)) THEN
*           integrate time and state variables in model(s)
            TIME = TIME + DELT
            CALL MODEL (3,IUDAT,IULOG,.FALSE.,TIME,DELT,TERMNL)
         END IF
      GOTO 10
      END IF

*     terminate model ; table output ; delete temporary file
      CALL MODEL (4,IUDAT,IULOG,OUTPUT,TIME,DELT,TERMNL)
      CALL OUTDAT (4,0,'MODEL',0.0)
      CALL OUTDAT (99,0,' ',0.0)

      RETURN
      END
```

Appendix E: Hierarchical order of TTUTIL routines

Some linkers scan object libraries only once. Then object libraries should have a hierarchical structure, i.e. lower level routines should follow higher level ones. Below a "level" is assigned to all TTUTIL routines. Higher level routines call lower level ones.

Level 6

EUDRIV, RKDRIV

Level 5

OUTARR, RDAREA, RDFROM, RDINIT, RDSETS
RDSINT, RDSREA

Level 4

OUTDAT, RDDATA

Level 3

COPFIL, DECREC, OUTCOM, OUTPLT, RDINDX,
RKQCA

Level 2

AFINVS, BOXMUL, CHKTSK, DECINT, DECREA,
ENTDCH, ENTDIN, ENTDRE, EXTENS, FOPEN ,
FOPENG, GAMMA , GETCH , IFINDC, LIMIT ,
LINT , PLTFUN, PLTHIS, POS , RK4A ,
STRIP , TIMER

Level 1

CLS , DECCHK, ENTCHA, ENTINT, ENTREA,
ERROR , GETREC, ILEN , INSW , INTGRL,
ISTART, MOFILP, REAAND, REANOR, REMOVE,
UNIFL , UPPERC, WORDS

Appendix F: The use of object libraries

Object libraries are made by compiling the FORTRAN sources of all routines. Subsequently, a special program for creating and maintaining libraries, a library manager, is used to create an object library. Below, at first, the creation of an object library is described for a VAX, an Apple Macintosh with the Absoft compiler and for an IBM PC with the Microsoft compiler. The actual linking procedure is described for the same three machines. Only the minimum number of required commands is explained. For details and a complete description of the library managers, the reader is referred to FORTRAN manuals.

Creating an object library:

VAX

It is assumed that the TTUTIL sources are present on a subdirectory UTIL. All routines are combined into a single source TEMP.FOR, which is compiled. The result is an object file TEMP.OBJ which is used to create the object library. After that, the object file can be deleted.

```
$ CREATE TEMP.F77
<CTRL>Z
$ APPEND *.FOR TEMP.F77
$ RENAME TEMP.F77 TEMP.FOR
$ FORTRAN/CHECK=(NOUNDERFLOW,OVERFLOW,BOUNDS) TEMP
```

Then create a library TTUTIL_LIB, using for instance:

```
$ LIBRARY/OBJECT/LOG/CREATE TTUTIL_LIB
```

and insert all objects into the library, using

```
$ LIBRARY/OBJECT/LOG/INSERT TTUTIL_LIB TEMP
$ DELETE TEMP.*;*
```

Apple

The names of all TTUTIL routines should be present in a separate list file. They are compiled then using the "compile list" command from the (Absoft) compiler menu. After selection of this list file, all listed subprograms are compiled. Do not forget to instruct the compiler for subprogram compilation (=option).

The library manager is called from the compiler menu ("command M"). It requires the name of the (new) library, TTUTIL.OLB, for instance. Then all object files have to be added to the library by means of successive library commands:

```
> A file1 (without extension)
> A file2
```

After entering all object names in this way, a "Q" is typed:

```
> Q
```

Before leaving the program, the library is created. Since the linker scans the library only once, the objects have to be added to the library in hierarchical order (see Appendix E).

IBM

At first, all source files have to be compiled, for instance using the following command, calling the Microsoft FORTRAN compiler.(option /G2 generates IBM-AT compatible code)

```
fl /Gt /4Yb /FPa /Od /c /AH /G2 *
```

Generally, a simpler procedure for calling the compiler will have been installed. A "*", instead of a filename causes all .FOR files to be compiled.

The library manager is run by typing the LIB command. After entering the library name TTUTIL, the creation of TTUTIL.LIB has to be confirmed. Then, all object files are inserted into the library by typing (from the DOS level):

```
LIB TTUTIL +file;
```

The order of insertion is not important.

Using an object library:

VAX

It is assumed that an object library, TTUTIL_LIB.OLB, is present on subdirectory UTIL. In linking a library to your programs you may profit from a logical TTUTIL, defined in your LOGIN.COM as follows:

```
$ ASSIGN DISK$....[username.UTIL]TTUTIL_LIB.OLB TTUTIL
```

A LINK command then reads

```
$ LINK PROG,SUB1,.....,TTUTIL/L
```

This command will work on all your directories, since the logical TTUTIL refers to the full filename of the object library.

Apple

It is assumed that an object library, TTUTIL.OLB, is present as:

```
HD40:FORTRAN:TTUTIL.OLB
```

It may be linked to a compiled program "`TEST apl`" by creating a linker control file with the following contents:

```
F TEST
L HD40:FORTRAN:TTUTIL.OLB
```

The linker is called from the compiler menu by "command K". Then the file menu of the linker is used to select the desired linker control file and you will see the program being linked to the library.

IBM

It is assumed that an object library, TTUTIL.LIB, has been created. The LINK command linking program TEST to the library reads:

```
LINK TEST,,,TTUTIL;
```

Library TTUTIL.LIB should be present on the current directory or on any other directory in the path.

Appendix G: List of NETLIB program libraries.

The NETLIB program libraries can be reached via the BITNET at NETLIB@ORNL.GOV. By means of simple commands one can order routines from many different libraries. Among these are the famous libraries EISPACK and LINPACK. The free availability of FORTRAN programs within the "computing community" is a very attractive side of the use of FORTRAN. Here is the index of the NETLIB system as it was sent to us on October 14, 1990. It contains an impressive **list of FORTRAN libraries !!**

The usual procedure is to order the index of a library that seems interesting. The index contains short descriptions of the subroutines and functions available in that library. Subsequently, you request the programs you need.

```
===== GENERAL NETLIB INDEX =====

WELCOME TO NETLIB, A SYSTEM FOR DISTRIBUTION OF MATHEMATICAL SOFTWARE
BY ELECTRONIC MAIL.  THIS INDEX IS THE REPLY YOU'LL GET TO:
        MAIL NETLIB@ORNL.GOV
        SEND INDEX.
TO EXAMINE THE FULL INDEX FOR ANY LIBRARY SEND A REQUEST OF THE FORM:
        SEND INDEX FROM EISPACK.
TO SEARCH FOR ALL SOFTWARE WITH CERTAIN KEYWORDS:
        FIND CUBIC SPLINE.
TO SEARCH FOR SOMEBODY IN GENE GOLUB'S ADDRESS LIST:
        WHO IS JOAN DOE?
DISPLAYS ENTRIES CONTAINING "JOAN" AND "DOE".  (NO SPELLING CORRECTION!)
YOU MAY INCLUDE SEVERAL REQUESTS IN A SINGLE PIECE OF MAIL, BUT PUT
EACH ON A SEPARATE LINE.

HERE ARE SOME ADDITIONAL FORMS A REQUEST MAY TAKE...
SEND DGECO FROM LINPACK
  (RETRIEVES ROUTINE DGECO AND ALL ROUTINES IT CALLS FROM THE LINPACK LIBRARY.)
SEND ONLY DGECO FROM LINPACK
  (RETRIEVES JUST DGECO AND NOT SUBSIDIARY ROUTINES.)
SEND DGECO BUT NOT DGEFA FROM LINPACK
  (RETRIEVES DGECO AND SUBSIDIARIES, BUT EXCLUDES DGEFA AND SUBSIDIARIES.)
SEND LIST OF DGECO FROM LINPACK
  (RETRIEVES JUST THE FILE NAMES RATHER THAN THE CONTENTS;
  THIS CAN BE HELPFUL WHEN ONE ALREADY HAS AN ENTIRE LIBRARY AND JUST
  WANTS TO KNOW WHAT PIECES ARE NEEDED IN A PARTICULAR APPLICATION.)

SEND THE REQUESTS TO "NETLIB@ORNL.GOV" EVEN THOUGH REPLIES APPEAR TO BE
COMING FROM "NETLIBD@ORNL.GOV".  YOU'LL BE TALKING TO A PROGRAM, SO DON'T
EXPECT IT TO UNDERSTAND MUCH ENGLISH.  IF YOUR UNIX SYSTEM DOESN'T TALK
DIRECTLY TO RESEARCH (AT&T BELL LABS IN MURRAY HILL, NEW JERSEY), YOU MIGHT
TRY FORWARDING THROUGH IHNP4 (BELL LABS IN CHICAGO) OR MCVAX (MATH CENTRUM
IN AMSTERDAM).  SOMEONE WILL BE PAYING FOR LONG DISTANCE PHONE CALLS, SO BE
REASONABLE!  THOSE WITH ACCESS TO THE ARPANET CAN USE NETLIB@ORNL.GOV (AT
OAK RIDGE NATIONAL LABS).  FOR AN INTRODUCTION TO THE MYSTERIES OF NETWORKS
AND ADDRESS SYNTAX, SEE J. QUARTERMAN AND J. HOSKINS, COMM. ACM (OCT 1986)
29,932-971.  FOR BACKGROUND ABOUT NETLIB, SEE JACK J. DONGARRA AND ERIC
GROSSE, DISTRIBUTION OF MATHEMATICAL SOFTWARE VIA ELECTRONIC MAIL, COMM.
ACM (1987) 30,403--407.

THE DEFAULT PRECISION IS DOUBLE; TO GET SINGLE, PREFIX THE LIBRARY NAME
WITH "S".  HOWEVER, IF THE LIBRARY ONLY COMES IN ONE PRECISION, THAT'S WHAT
YOU WILL BE SENT.  TO SAVE SPACE WE REMOVE SEQUENCE NUMBERS AND MAINTAIN A
CENTRAL SET OF MACHINE DEPENDENT CONSTANTS.  OTHERWISE THE CODES, WHICH ARE
ALMOST ALL IN FORTRAN, ARE AS RECEIVED FROM THE AUTHORS.  BUGS FOUND IN
CORE LIBRARIES LIKE EISPACK WILL RECEIVE PROMPT ATTENTION; IN GENERAL, WE
```

WILL FORWARD COMMENTS (AND ANNUAL LISTS OF RECIPIENTS) TO THE CODE AUTHORS.
MANY OF THESE CODES ARE DESIGNED FOR USE BY PROFESSIONAL NUMERICAL ANALYSTS
WHO ARE CAPABLE OF CHECKING FOR THEMSELVES WHETHER AN ALGORITHM IS SUITABLE
FOR THEIR NEEDS. ONE ROUTINE CAN BE SUPERB AND THE NEXT AWFUL. SO BE
CAREFUL!

-------QUICK SUMMARY OF CONTENTS---------
A - APPROXIMATION ALGORITHMS (ALMOST EMPTY, BUT SOON TO GROW)
ALLIANT - SET OF PROGRAMS COLLECTED FROM ALLIANT USERS
APOLLO - SET OF PROGRAMS COLLECTED FROM APOLLO USERS
BENCHMARK - VARIOUS BENCHMARK PROGRAMS AND A SUMMARY OF TIMINGS
BIHAR - BJORSTAD'S BIHARMONIC SOLVER
BMP - BRENT'S MULTIPLE PRECISION PACKAGE
CHENEY-KINCAID - PROGRAMS FROM THE TEXT NUMERICAL MATHEMATICS AND COMPUTING.
CONFORMAL - SCHWARZ-CHRISTOFFEL CODES BY TREFETHEN; BJORSTAD+GROSSE
CORE - MACHINE CONSTANTS, LEVEL 1, 2, AND 3 BLAS
DOMINO - COMMUNICATION AND SCHEDULING OF MULTIPLE TASKS; UNIV. MARYLAND
EISPACK - MATRIX EIGENVALUES AND VECTORS
ELEFUNT - CODY AND WAITE'S TESTS FOR ELEMENTARY FUNCTIONS
ERRATA - CORRECTIONS TO NUMERICAL BOOKS
FISHPACK - SEPARABLE ELLIPTIC PDES; SWARZTRAUBER AND SWEET
FITPACK - CLINE'S SPLINES UNDER TENSION
FFTPACK - SWARZTRAUBER'S FOURIER TRANSFORMS
FMM - SOFTWARE FROM THE BOOK BY FORSYTHE, MALCOLM, AND MOLER
FN - FULLERTON'S SPECIAL FUNCTIONS
GCV - GENERALIZED CROSS VALIDATION
GO - "GOLDEN OLDIES" GAUSSQ, ZEROIN, LOWESS, ...
GRAPHICS - RAY-TRACING
HARWELL - MA28 SPARSE LINEAR SYSTEM
HOMPACK - NONLINEAR EQUATIONS BY HOMOTOPY METHOD
ITPACK - ITERATIVE LINEAR SYSTEM SOLUTION BY YOUNG AND KINCAID
LANCZOS - CULLUM AND WILLOUGHBY'S LANCZOS PROGRAMS
LASO - SCOTT'S LANCZOS PROGRAM FOR EIGENVALUES OF SPARSE MATRICES
LINPACK - GAUSSIAN ELIMINATION, QR, SVD BY DONGARRA, BUNCH, MOLER, STEWART
LP - LINEAR PROGRAMMING
MACHINES - SHORT DESCRIPTIONS OF VARIOUS COMPUTERS
MATLAB - SOFTWARE FROM THE MATLAB USER'S GROUP
MICROSCOPE - ALFELD AND HARRIS' SYSTEM FOR DISCONTINUITY CHECKING
MINPACK - NONLINEAR EQUATIONS AND LEAST SQUARES BY MORE, GARBOW, HILLSTROM
MISC - EVERYTHING ELSE
NA-DIGEST - ARCHIVE OF MAILINGS TO NA DISTRIBUTION LIST
NAPACK - NUMERICAL ALGEBRA PROGRAMS
ODE - ORDINARY DIFFERENTIAL EQUATIONS
ODEPACK - ORDINARY DIFFERENTIAL EQUATIONS FROM HINDMARSH
PARANOIA - KAHAN'S FLOATING POINT TEST
PCHIP - HERMITE CUBICS FRITSCH+CARLSON
PICL - PORTABLE INSTRUMENTED COMMUNICATION LIBRARY FOR MULTIPROCESSORS
PLTMG - BANK'S MULTIGRID CODE; TOO LARGE FOR ORDINARY MAIL
POLYHEDRA - HUME'S DATABASE OF GEOMETRIC SOLIDS
PORT - THE PUBLIC SUBSET OF PORT LIBRARY
PPPACK - SUBROUTINES FROM DE BOOR'S PRACTICAL GUIDE TO SPLINES
QUADPACK - UNIVARIATE QUADRATURE BY PIESSENS, DE DONKER, KAHANER
SIAM - TYPESETTING MACROS FOR SIAM JOURNAL FORMAT
SLATEC - MACHINE CONSTANTS AND ERROR HANDLING PACKAGE FROM THE SLATEC LIBRARY
SPARSE - A SET OF C CODES FOR SPARSE SYSTEMS OF EQUATIONS
SPARSPAK - GEORGE + LIU, SPARSE LINEAR ALGEBRA CORE
SPECFUN - TRANSPORTABLE SPECIAL FUNCTIONS
TOEPLITZ - LINEAR SYSTEMS IN TOEPLITZ OR CIRCULANT FORM BY GARBOW
TOMS - COLLECTED ALGORITHMS OF THE ACM
VFFTPK - A VECTORIZED PACKAGE OF FORTRAN FOR FAST FOURIER TRANSFORM
Y12M - SPARSE LINEAR SYSTEM (AARHUS)

--------A BIT MORE DETAIL--------

THE FIRST FEW LIBRARIES HERE ARE WIDELY REGARDED AS BEING OF HIGH QUALITY.
THE LIKELIHOOD OF YOUR ENCOUNTERING A BUG IS RELATIVELY SMALL; IF YOU DO,
WE CERTAINLY WANT TO HEAR ABOUT IT!

CORE MACHINE CONSTANTS (I1MACH,R1MACH,D1MACH), BLAS (LEVEL 1, 2 AND 3)

EISPACK A COLLECTION OF FORTRAN SUBROUTINES THAT COMPUTE THE EIGENVALUES AND EIGENVECTORS OF NINE CLASSES OF MATRICES. THE PACKAGE CAN DETERMINE THE EIGENSYSTEMS OF COMPLEX GENERAL, COMPLEX HERMITIAN, REAL GENERAL, REAL SYMMETRIC, REAL SYMMETRIC BAND, REAL SYMMETRIC TRIDIAGONAL, SPECIAL REAL TRIDIAGONAL, GENERALIZED REAL, AND GENERALIZED REAL SYMMETRIC MATRICES. IN ADDITION, THERE ARE TWO ROUTINES WHICH USE THE SINGULAR VALUE DECOMPOSITION TO SOLVE CERTAIN LEAST SQUARES PROBLEMS.
DEVELOPED BY THE NATS PROJECT AT ARGONNE NATIONAL LABORATORY.
(D.P. REFER TO EISPACK, S.P. REFER TO SEISPACK)

FFTPACK A PACKAGE OF FORTRAN SUBPROGRAMS FOR THE FAST FOURIER TRANSFORM OF PERIODIC AND OTHER SYMMETRIC SEQUENCES
THIS PACKAGE CONSISTS OF PROGRAMS WHICH PERFORM FAST FOURIER TRANSFORMS FOR BOTH COMPLEX AND REAL PERIODIC SEQUENCES AND CERTIAN OTHER SYMMETRIC SEQUENCES.
DEVELOPED BY PAUL SWARZTRAUBER, AT NCAR.

FISHPACK A PACKAGE OF FORTRAN SUBPROGRAMS PROVIDING FINITE DIFFERENCE APPROXIMATIONS FOR ELLIPTIC BOUNDARY VALUE PROBLEMS.
DEVELOPED BY PAUL SWARZTRAUBER AND ROLAND SWEET.

FNLIB WAYNE FULLERTON'S SPECIAL FUNCTION LIBRARY. (SINGLE AND DOUBLE)

GO GOLDEN OLDIES: ROUTINES THAT HAVE BEEN WIDELY USED, BUT AREN'T AVAILABLE THROUGH THE STANDARD LIBRARIES.
NOMINATIONS WELCOME!

HARWELL SPARSE MATRIX ROUTINE MA28 FROM THE HARWELL LIBRARY. FROM IAIN DUFF

LINPACK A COLLECTION OF FORTRAN SUBROUTINES THAT ANALYZE AND SOLVE LINEAR EQUATIONS AND LINEAR LEAST SQUARES PROBLEMS. THE PACKAGE SOLVES LINEAR SYSTEMS WHOSE MATRICES ARE GENERAL, BANDED, SYMMETRIC INDEFINITE, SYMMETRIC POSITIVE DEFINITE, TRIANGULAR, AND TRIDIAGONAL SQUARE. IN ADDITION, THE PACKAGE COMPUTES THE QR AND SINGULAR VALUE DECOMPOSITIONS OF RECTANGULAR MATRICES AND APPLIES THEM TO LEAST SQUARES PROBLEMS.
DEVELOPED BY JACK DONGARRA, JIM BUNCH, CLEVE MOLER AND PETE STEWART.
(ALL PRECISIONS CONTAINED HERE)

PPPACK SUBROUTINES FROM: CARL DE BOOR, A PRACTICAL GUIDE TO SPLINES, SPRINGER VERLAG. THIS IS AN OLD VERSION, FROM AROUND THE TIME THE BOOK WAS PUBLISHED. WE WILL INSTALL A NEWER VERSION AS SOON AS WE CAN.

TOMS COLLECTED ALGORITHMS OF THE ACM. WHEN REQUESTING A SPECIFIC ITEM, PLEASE REFER TO THE ALGORITHM NUMBER.

IN CONTRAST TO THE ABOVE LIBRARIES, THE FOLLOWING ARE COLLECTIONS OF CODES FROM A VARIETY OF SOURCES. MOST ARE EXCELLENT, BUT YOU SHOULD EXERCISE CAUTION. WE INCLUDE RESEARCH CODES THAT WE HAVEN'T TESTED AND CODES THAT MAY NOT BE STATE-OF-THE-ART BUT USEFUL FOR COMPARISONS.
THE FOLLOWING LIST IS CHRONOLOGICAL, NOT BY MERIT:

MISC CONTAINS VARIOUS PIECES OF SOFTWARE COLLECTED OVER TIME AND:
THE SOURCE CODE FOR THE NETLIB PROCESSOR ITSELF;
THE PAPER DESCRIBING NETLIB AND ITS IMPLEMENTATION;
THE ABSTRACTS LIST MAINTAINED BY RICHARD BARTELS.

FMM ROUTINES FROM THE BOOK COMPUTER METHODS FOR MATHEMATICAL COMPUTATIONS, BY FORSYTHE, MALCOLM, AND MOLER.
DEVELOPED BY GEORGE FORSYTHE, MIKE MALCOLM, AND CLEVE MOLER.
(D.P. REFER TO FMM, S.P. REFER TO SFMM)

QUADPACK A PACKAGE FOR NUMERICAL COMPUTATION OF DEFINITE UNIVARIATE INTEGRALS.

DEVELOPED BY PIESSENS, ROBERT(APPL. MATH. AND PROGR. DIV.- K.U.LEUVEN)
DE DONKER, ELISE(APPL. MATH. AND PROGR. DIV.- K.U.LEUVEN
KAHANER, DAVID(NATIONAL BUREAU OF STANDARDS) (SLATEC VERSION)

TOEPLITZ A PACKAGE OF FORTRAN SUBPROGRAMS FOR THE SOLUTION OF SYSTEMS OF LINEAR EQUATIONS WITH COEFFICIENT MATRICES OF TOEPLITZ OR CIRCULANT FORM, AND FOR ORTHOGONAL FACTORIZATION OF COLUMN-CIRCULANT MATRICES.
DEVELOPED BY BURT GARBOW AT ARGONNE NATIONAL LABORATORY, AS A CULMINATION OF SOVIET-AMERICAN COLLABORATIVE EFFORT.
(D.P. REFER TO TOEPLITZ, S.P. REFER TO STOEPLITZ)

ITPACK ITERATIVE LINEAR SYSTEM SOLVERS FOR SYMMETRIC AND NONSYMMETRIC SPARSE PROBLEMS. INCLUDES ITPACK 2C (SINGLE AND DOUBLE PRECISION), ITPACKV 2C (VECTORIZED VERSION OF ITPACK 2C), AND NSPCG. DEVELOPED BY YOUNG AND KINCAID AND THE GROUP AT U OF TEXAS.

BIHAR BIHARMONIC SOLVER IN RECTANGULAR GEOMETRY AND POLAR COORDINATES. THESE ROUTINES WERE OBTAINED FROM PETTER BJORSTAD, VERITAS RESEARCH, OSLO NORWAY IN JULY 1984.

LANCZOS PROCEDURES COMPUTING A FEW EIGENVALUES/EIGENVECTORS OF A LARGE (SPARSE) SYMMETRIC MATRIX. JANE CULLUM AND RALPH WILLOUGHBY, IBM YORKTOWN.

LASO A COMPETING LANCZOS PACKAGE. DAVID SCOTT.

CONFORMAL CONTAINS ROUTINES TO SOLVE THE "PARAMETER PROBLEM" ASSOCIATED WITH THE SCHWARZ-CHRISTOFFEL MAPPING. INCLUDES:
SCPACK (POLYGONS WITH STRAIGHT SIDES) FROM NICK TREFETHEN.
CAP (CIRCULAR ARC POLYGONS) FROM PETTER BJORSTAD AND ERIC GROSSE.

FITPACK A PACKAGE FOR SPLINES UNDER TENSION. (AN EARLY VERSION)
FOR A CURRENT COPY AND FOR OTHER ROUTINES, CONTACT:
ALAN KAYLOR CLINE, 8603 ALTUS COVE, AUSTIN, TEXAS 78759, USA

BENCHMARK CONTAINS BENCHMARK PROGRAMS AND THE TABLE OF LINPACK TIMINGS.

MACHINES CONTAINS INFORMATION ON HIGH PERFORMANCE COMPUTERS THAT ARE OR SOON TO BE MADE AVAILABLE

MINPACK A PACKAGE OF FORTRAN PROGRAMS FOR THE SOLUTION OF SYSTEMS OF NONLINEAR EQUATIONS AND NONLINEAR LEAST SQUARES PROBLEMS.
FIVE ALGORITHMIC PATHS EACH INCLUDE A CORE SUBROUTINE AND AN EASY-TO-USE DRIVER. THE ALGORITHMS PROCEED EITHER FROM AN ANALYTIC SPECIFICATION OF THE JACOBIAN MATRIX OR DIRECTLY FROM THE PROBLEM FUNCTIONS. THE PATHS INCLUDE FACILITIES FOR SYSTEMS OF EQUATIONS WITH A BANDED JACOBIAN MATRIX, FOR LEAST SQUARES PROBLEMS WITH A LARGE AMOUNT OF DATA, AND FOR CHECKING THE CONSISTENCY OF THE JACOBIAN MATRIX WITH THE FUNCTIONS.
DEVELOPED BY JORGE MORE', BURT GARBOW, AND KEN HILLSTROM AT ARGONNE NATIONAL LABORATORY.
(D.P. REFER TO MINPACK, S.P. REFER TO SMINPACK)

PORT THE PUBLIC SUBSET OF THE PORT LIBRARY. INCLUDES THE LATEST VERSION OF GAY'S NL2SOL NONLINEAR LEAST SQUARES. THE REST OF THE PORT3 LIBRARY IS AVAILABLE BY LICENSE FROM AT&T.

Y12M CALCULATION OF THE SOLUTION OF SYSTEMS OF LINEAR SYSTEMS OF LINEAR ALGEBRA EQUATIONS WHOSE MATRICES ARE LARGE AND SPARSE.
AUTHORS: ZAHARI ZLATEV, JERZY WASNIEWSKI AND KJELD SCHAUMBURG

PCHIP IS A FORTRAN PACKAGE FOR PIECEWISE CUBIC HERMITE INTERPOLATION OF DATA. IT FEATURES SOFTWARE TO PRODUCE A MONOTONE AND "VISUALLY PLEASING" INTERPOLANT TO MONOTONE DATA.
FRED N. FRITSCH, LAWRENCE LIVERMORE NATIONAL LABORATORY

LP LINEAR PROGRAMMING - AT PRESENT, THIS CONSISTS OF ONE SUBDIRECTORY,

DATA: A SET OF TEST PROBLEMS IN MPS FORMAT, MAINTAINED BY DAVID GAY.
FOR MORE INFORMATION, TRY A REQUEST OF THE FORM
SEND INDEX FOR LP/DATA

ODE VARIOUS INITIAL AND BOUNDARY VALUE ORDINARY DIFFERENTIAL EQUATION SOLVERS: COLSYS, DVERK, RKF45, ODE
A SUBSET OF THESE IN SINGLE PRECISION IS IN THE LIBRARY SODE.

ODEPACK THE ODE PACKAGE FROM HINDMARCH AND OTHERS.
THIS IS THE DOUBLE PRECISION VERISON; TO GET SP REFER TO SODEPACK.
ALAN HINDMARCH, LAWRENCE LIVERMORE NATIONAL LABORATORY

ELEFUNT IS A COLLECTION OF TRANSPORTABLE FORTRAN PROGRAMS FOR TESTING THE ELEMENTARY FUNCTION PROGRAMS PROVIDED WITH FORTRAN COMPILERS. THE PROGRAMS ARE DESCRIBED IN DETAIL IN THE BOOK "SOFTWARE MANUAL FOR THE ELEMENTARY FUNCTIONS" BY W. J. CODY AND W. WAITE, PRENTICE HALL, 1980.

SPECFUN IS AN INCOMPLETE, BUT GROWING, COLLECTION OF TRANSPORTABLE FORTRAN PROGRAMS FOR SPECIAL FUNCTIONS, AND OF ACCOMPANYING TEST PROGRAMS SIMILAR IN CONCEPT TO THOSE IN ELEFUNT.
W.J. CODY, ARGONNE NATIONAL LABORATORY

PARANOIA IS A RATHER LARGE PROGRAM, DEVISED BY PROF. KAHAN OF BERKELEY, TO EXPLORE THE FLOATING POINT SYSTEM ON YOUR COMPUTER.

SLATEC LIBRARY DOE POLICY APPARENTLY PROHIBITS US FROM DISTRIBUTING THIS. CONTACT THE NATIONAL ENERGY SOFTWARE CENTER OR YOUR CONGRESSMAN.

HOMPACK IS A SUITE OF FORTRAN 77 SUBROUTINES FOR SOLVING NONLINEAR SYSTEMS OF EQUATIONS BY HOMOTOPY METHODS. THERE ARE SUBROUTINES FOR FIXED POINT, ZERO FINDING, AND GENERAL HOMOTOPY CURVE TRACKING PROBLEMS, UTILIZING BOTH DENSE AND SPARSE JACOBIAN MATRICES, AND IMPLEMENTING THREE DIFFERENT ALGORITHMS: ODE-BASED, NORMAL FLOW, AND AUGMENTED JACOBIAN.

DOMINO IS A SET OF C-LANGUAGE ROUTINES WITH A SHORT ASSEMBLY LANGUAGE INTERFACE THAT ALLOWS MULTIPLE TASKS TO COMMUNICATE AND SCHEDULES LOCAL TASKS FOR EXECUTION. THESE TASKS MAY BE ON A SINGLE PROCESSOR OR SPREAD AMONG MULTIPLE PROCESSORS CONNECTED BY A MESSAGE-PASSING NETWORK. (O'LEARY, STEWART, VAN DE GEIJN, UNIVERSITY OF MARYLAND)

GCV SOFTWARE FOR GENERALIZED CROSS VALIDATION, FROM: WOLTRING, (UNIVARIATE SPLINE SMOOTHING); BATES, LINDSTROM, WAHBA AND YANDELL (MULTIVARIATE THIN PLATE SPLINE SMOOTHING AND RIDGE REGRESSION).

CHENEY-KINCAID PROGRAMS FROM: WARD CHENEY & DAVID KINCAID, NUMERICAL MATHEMATICS AND COMPUTING.

POLYHEDRA A DATABASE OF ANGLES, VERTEX LOCATIONS, AND SO ON FOR OVER A HUNDRED GEOMETRIC SOLIDS, COMPILED BY ANDREW HUME.

GRAPHICS PRESENTLY JUST CONTAINS SOME C ROUTINES FOR TESTING RAY-TRACING

A APPROXIMATION ALGORITHMS (ALMOST EMPTY, BUT SOON TO GROW)
LOWESS: MULTIVARIATE SMOOTHING OF SCATTERED DATA; CLEVELAND+DEVLIN+GROSSE

APOLLO A SET OF PROGRAMS COLLECTED FROM APOLLO USERS.

ALLIANT A SET OF PROGRAMS COLLECTED FROM ALLIANT USERS.

PARMACS - PARALLEL PROGRAMMMING MACROS FOR MONITORS AND SEND/RECEIVE
RUSTY LUSK, ARGONNE NATIONAL LABORATORY, JUNE 5, 1987 (LUSK@MCS.ANL.GOV)

SCHED - THE SCHEDULE PACKAGE IS AN ENVIRONMENT FOR THE TRANSPORTABLE IMPLEMENTATION OF PARALLEL ALGORITHMS IN A FORTRAN SETTING.
JACK DONGARRA AND DAN SORENSEN, UNIV OF TENN. AND RICE UNIV.,
JUNE 5, 1987 (DONGARRA@CS.UTK.EDU SORENSEN@RICE.EDU)

NAPACK A COLLECTION OF FORTRAN SUBROUTINES TO SOLVE LINEAR SYSTEMS, TO ESTIMATE THE CONDITION NUMBER OR THE NORM OF A MATRIX, TO COMPUTE DETERMINANTS, TO MULTIPLY A MATRIX BY A VECTOR, TO INVERT A MATRIX, TO SOLVE LEAST SQUARES PROBLEMS, TO PERFORM UNCONSTRAINED MINIMIZATION, TO COMPUTE EIGENVALUES, EIGENVECTORS, THE SINGULAR VALUE DECOMPOSITION, OR THE QR DECOMPOSITION. THE PACKAGE HAS SPECIAL ROUTINES FOR GENERAL, BAND, SYMMETRIC, INDEFINITE, TRIDIAGONAL, UPPER HESSENBERG, AND CIRCULANT MATRICES. CODE AUTHOR: BILL HAGER, MATHEMATICS DEPARTMENT, PENN STATE UNIVERSITY, UNIVERSITY PARK, PA 16802, E-MAIL: HAGER@PSUVAX1.BITNET OR HAGER@PSUVAX1.PSU.EDU. RELATED BOOK: APPLIED NUMERICAL LINEAR ALGEBRA, PRENTICE-HALL, ENGLEWOOD CLIFFS, NEW JERSEY. BOOK SCHEDULED TO APPEAR IN DECEMBER, 1987.

SPARSPAK SUBROUTINES FROM THE BOOK "COMPUTER SOLUTION OF LARGE SPARSE POSITIVE DEFINITE SYSTEMS" BY GEORGE AND LIU, PRENTICE HALL 1981.

SPARSE A LIBRARY OF SUBROUTINES WRITTEN IN C THAT SOLVE LARGE SPARSE SYSTEMS OF LINEAR EQUATIONS USING LU FACTORIZATION. THE PACKAGE IS ABLE TO HANDLE ARBITRARY REAL AND COMPLEX SQUARE MATRIX EQUATIONS. BESIDES BEING ABLE TO SOLVE LINEAR SYSTEMS, IT IS SOLVES TRANSPOSED SYSTEMS, FIND DETERMINANTS, MULTIPLIES A VECTOR BY A MATRIX, AND ESTIMATE ERRORS DUE TO ILL-CONDITIONING IN THE SYSTEM OF EQUATIONS AND INSTABILITY IN THE COMPUTATIONS. SPARSE DOES NOT REQUIRE OR ASSUME SYMMETRY AND IS ABLE TO PERFORM NUMERICAL PIVOTING (EITHER DIAGONAL OR COMPLETE) TO AVOID UNNECESSARY ERROR IN THE SOLUTION. SPARSE ALSO HAS AN OPTIONAL INTERFACE THAT ALLOW IT TO BE CALLED FROM FORTRAN PROGRAMS.
KEN KUNDERT, ALBERTO SANGIOVANNI-VINCENTELLI. (SPARSE@IC.BERKELEY.EDU)

SLAP THIS IS THE OFFICIAL RELEASE VERSION 2.0 OF THE SPARSE LINEAR ALGEBRA PACKAGE: A SLAP FOR THE MASSES! IT CONTAINS "CORE" ROUTINES FOR THE ITERATIVE SOLUTION SYMMETRIC AND NON-SYMMETRIC POSITIVE DEFINITE AND POSITIVE SEMI-DEFINITE LINEAR SYSTEMS. INCLUDED IN THIS PACKAGE ARE CORE ROUTINES TO DO ITERATIVE REFINEMENT ITERATION, PRECONDITIONED CONJUGATE GRADIENT ITERATION, PRECONDITIONED CONJUGATE GRADIENT ITERATION ON THE NORMAL EQUATIONS, PRECONDITIONED BICONJUGATE GRADIENT ITERATION, PRECONDITIONED BICONJUGATE GRADIENT SQUARED ITERATION, ORTHOMIN ITERATION AND GENERALIZED MINIMUM RESIDUAL ITERATION. CORE ROUTINES REQUIRE THE USER TO SUPPLY "MATVEC" (MATRIX VECTOR MULTIPLY) AND "MSOLVE" (PRECONDITIONG) ROUTINES. THIS ALLOWS THE CORE ROUTINES TO BE WRITTEN IN A WAY THAT MAKES THEM INDEPENDENT OF THE MATRIX DATA STRUCTURE. FOR EACH CORE ROUTINE THERE ARE SEVERAL DRIVERS AND SUPPORT ROUTINES THAT ALLOW THE USER TO UTILIZE DIAGONAL SCALING AND INCOMPLETE CHOLESKY/INCOMPLETE LU FACTORIZATION AS PRECONDITIONERS WITH NO CODING. THE PRICE FOR THIS CONVIENCE IS THAT ONE MUST USE THE A SPECIFIC MATRIX DATA STRUCTURE: SLAP COLUMN OR SLAP TRIAD FORMAT.
WRITTEN BY MARK K. SEAGER & ANNE GREENBAUM

PROBLEM-SET: THIS SET OF DIRECTORIES IS A COLLECTION OF PROBLEMS FOR AUTOMATED THEOREM PROVERS. IT IS PARTIONED BY SUBJECT.
LARRY WOS, ARGONNE NATIONAL LABORATORY

SEQUENT SOFTWARE FROM THE SEQUENT USERS GROUP.
JACK DONGARRA 9/88

UNCON/DATA TEST PROBLEMS: UNCONSTRAINED OPTIMIZATION, NONLINEAR LEAST SQUARES.
PROBLEMS FROM MORE, GARBOW, AND HILLSTROM; FRALEY, MATRIX SQUARE ROOT; HANSON, SALANE; MCKEOWN; DE VILLIERS AND GLASSER; DENNIS, GAY, AND VU. COLLECTED BY CHRIS FRALEY.

JAKEF IS A PRECOMPILER THAT ANALYSES A GIVEN FORTRAN77 SOURCE CODE FOR THE EVALUATION OF A SCALAR OR VECTOR FUNCTION AND THEN GENERATES AN EXPANDED FORTRAN SUBROUTINE THAT SIMULTANEOUSLY EVALUATES THE GRADIENT OR JACOBIAN RESPECTIVELY.

A. GRIEWANK, ARGONNE NATIONAL LABORATORY, GRIEWANK@MCS.ANL.GOV, 12/1/88.

SPARSE-BLAS AN EXTENSION TO THE SET OF BASIC LINEAR ALGEBRA SUBPROGRAMS.
THE EXTENSION IS TARGETED AT SPARSE VECTOR OPERATIONS, WITH THE GOAL OF PROVIDING EFFICIENT, BUT PORTABLE, IMPLEMENTATIONS OF ALGORITHMS FOR HIGH PERFORMANCE COMPUTERS.
CONVEX!DODSON@ANL-MCS.ARPA MON AUG 31 19:53:21 1987 (DAVE DODSON)

VORONOI - COMPUTE VORONOI DIAGRAM OR DELAUNAY TRIANGULATION.
FROM RESEARCH!SJF THU MAY 5 14:09:33 EDT 1988

MATLAB - SOFTWARE FROM THE MATLAB USERS GROUP.
CHRISTIAN BISCHOF BISCHOF@MCS.ANL.GOV 12/89

PICL - IS A SUBROUTINE LIBRARY THAT IMPLEMENTS A GENERIC MESSAGE-PASSING INTERFACE FOR A VARIETY OF MULTIPROCESSORS. IT ALSO PROVIDES TIMESTAMPED TRACE DATA, IF REQUESTED.
AUTHORS: GEIST, HEATH, PEYTON, AND WORLEY, OAK RIDGE NATIONAL LAB.
WORLEY@MSR.EPM.ORNL.GOV 4/17/90.

PARALLEL - A DIRECTORY CONTAINING INFORMATION ON PARALLEL PROCESSING AND HIGH-PERFORMANCE COMPUTING.

MADPACK IS A A COMPACT PACKAGE FOR SOLVING SYSTEMS OF LINEAR EQUATIONS USING MULTIGRID OR AGGREGATION-DISAGGREGATION METHODS. IMBEDDED IN THE ALGORITHMS ARE IMPLEMENTATIONS FOR SPARSE GAUSSIAN ELIMINATION AND SYMMETRIC GAUSS-SEIDEL (UNACCELERATED OR ACCELERATED BY CONJUGATE GRADIENTS OR ORTHOMIN(1)). THIS PACKAGE IS PARTICULARLY USEFUL FOR SOLVING PROBLEMS WHICH ARISE FROM DISCRETIZING PARTIAL DIFFERENTIAL EQUATIONS, REGARDLESS OF WHETHER FINITE DIFFERENCES, FINITE ELEMENTS, OR FINITE VOLUMES ARE USED.
IT WAS WRITTEN BY CRAIG DOUGLAS.

PARAGRAPH - A GRAPHICAL DISPLAY SYSTEM FOR VISUALIZING THE BEHAVIOR OF PARALLEL ALGORITHMS ON MESSAGE-PASSING MULTIPROCESSOR ARCHITECTURES.
AUTHORS: JENNIFER ETHERIDGE AND MICHAEL HEATH, OAK RIDGE NATIONAL LAB.
MTH@ORNL.GOV, 4/19/90.

NAPACK A COLLECTION OF FORTRAN SUBROUTINES TO SOLVE LINEAR SYSTEMS, TO ESTIMATE THE CONDITION NUMBER OR THE NORM OF A MATRIX, TO COMPUTE DETERMINANTS, TO MULTIPLY A MATRIX BY A VECTOR, TO INVERT A MATRIX, TO SOLVE LEAST SQUARES PROBLEMS, TO PERFORM UNCONSTRAINED MINIMIZATION, TO COMPUTE EIGENVALUES, EIGENVECTORS, THE SINGULAR VALUE DECOMPOSITION, OR THE QR DECOMPOSITION.
THE PACKAGE HAS SPECIAL ROUTINES FOR BAND, SYMMETRIC, INDEFINITE, TRIDIAGONAL, UPPER HESSENBERG, AND CIRCULANT MATRICES.
CODE AUTHOR: BILL HAGER, MATHEMATICS DEPARTMENT, UNIVERSITY OF FLORIDA, GAINESVILLE, FL 32611, E-MAIL: HAGER@MATH.UFL.EDU.
RELATED BOOK: APPLIED NUMERICAL LINEAR ALGEBRA, PRENTICE-HALL, ENGLEWOOD CLIFFS, NEW JERSEY, 1988.

GMAT -
FROM MARK SEAGER (LLNL OCT 8, 1987)

STATEGRAPH.SHAR THIS SHAR FILE CONTAINS THE SOURCE CODE FOR THE GMAT STATEGRAPH ANALYSIS TOOL. THIS TOOL WILL ANALYZE MULTIPROCESSING TRACE FILES GENERATED BY THE CRAY COMPATIBILITY LIBRARY ON THE ALLIANT FX/8. THE INPUT FILE SPECIFICATION IS VERY SIMILAR TO THAT FOR MTDUMP FROM CRAY RESEARCH.

TIMELINE.SHAR THIS SHAR FILE CONTAINS THE SOURCE CODE FOR THE GMAT TIMELINE ANALYSIS TOOL. THIS TOOL WILL ASSIST THE USER IN A TIME BASED ANALYSIS OF MULTIPROCESSING TRACE FILES GENERATED BY THE CRAY COMPATIBILITY LIBRARY ON THE ALLIANT FX/8. THE INPUT FILE SPECIFICATION IS VERY SIMILAR TO THAT FOR MTDUMP FROM CRAY RESEARCH.

GMAT.SHAR THIS SHAR FILE CONTAINS DOCUMENTATION THE GMAT MULTI-PROCESSING TIME LINE AND STATE GRAPH TOOLS. IT ALSO INCLUDES SOME (COMPACED AND UUENCODED) SAMPLE TRACE FILES FOR USE WITH THE TOOLS.

VFFTPK - A VECTORIZED PACKAGE OF FORTRAN SUBPROGRAMS FOR THE FAST FOURIER TRANSFORM OF MULTIPLE REAL SEQUENCES

FORTRAN CONTAINS TOOLS SPECIFIC TO FORTRAN.
AT PRESENT, IT CONTAINS A SINGLE-DOUBLE PRECISION CONVERTER.

TYPESETTING TROFF AND LATEX MACROS, MOSTLY WRITTEN AT BELL LABS. ALSO, AMS-TEX MACROS BY ARNOLD, LUCIER, AND SIAM.

C++ MISCELLANEOUS CODES IN THE C++ LANGUAGE. AT PRESENT THIS INCLUDES HANSEN'S C++ ANSWER BOOK.

OPT MISCELLANEOUS OPTIMIZATION SOFTWARE. CONTAINS BRENT'S PRAXIS.

BIB BIBLIOGRAPHIES: GOLUB AND VAN LOAN, 2ND ED.