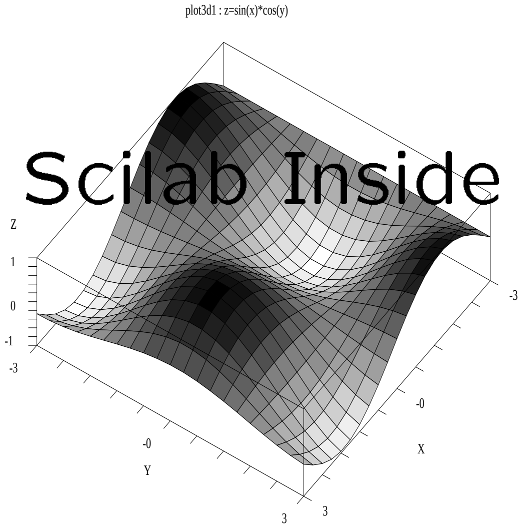
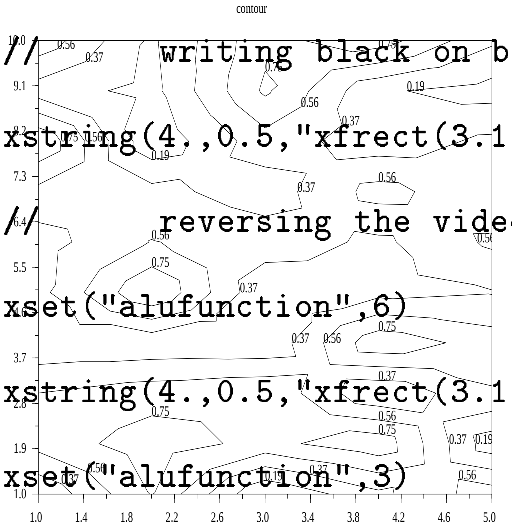
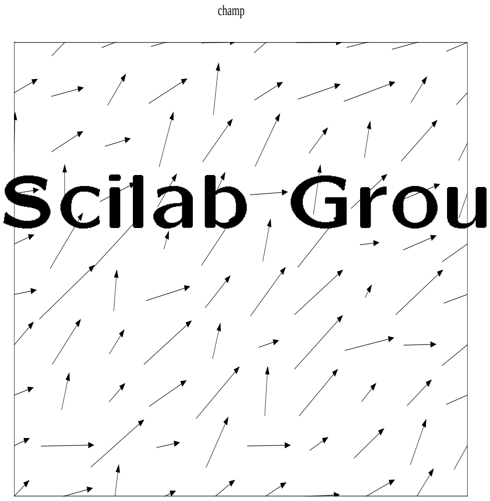
Guide For Developpers

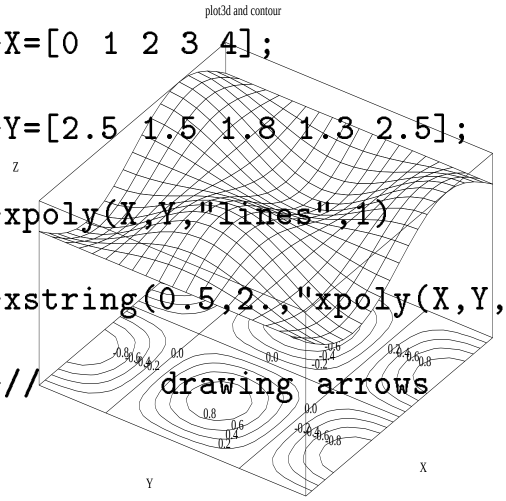
Scilab Inside

```
-->plot(1:10)

-->xbasc()

-->//      simple rectangle

-->xrect(0,1,3,1)

-->//      filling a rectangle

-->xfrect(3.1,1,3,1)

-->//      writing in the rectangle

-->xstring(0.5,0.5,"xrect(0,1,3,1)")

-->//      writing black on black !

-->xstring(4.,0.5,"xfrect(3.1,1,3,1)")

-->//      reversing the video

-->xset("alufunction",6)

-->xstring(4.,0.5,"xfrect(3.1,1,3,1)")

-->xset("alufunction",3)

-->//      drawing a polyline

-->X=[0 1 2 3 4];

-->Y=[2.5 1.5 1.8 1.3 2.5];

-->xpoly(X,Y,"lines",1)

-->xstring(0.5,2.,"xpoly(X,Y,""lines""

-->//      drawing arrows
```









Scilab Group

Scilab
Internals

Scilab Group
INRIA Meta2 Project/ENPC Cergrene

INRIA - Unité de recherche de Rocquencourt - Projet Meta2
Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
E-mail : scilab@inria.fr

Contents

We describe here the internal structure of Scilab, in particular the way Fortran subroutine are hard interfaced with Scilab: description of the stack where all Scilab data are put and description of the internal coding of Scilab data structures.

1 From the Scilab call to the prompt

We describe here the sequence of the "master" programs of Scilab which are executed before the prompt and how Scilab is organized.

The program `main` is `scilab` (in the sub-directory `routines/default`) which begins with the call of `inffic` (in the sub-directory `routines/sun`) which initializes the names of the main files needed by Scilab (for the help, save, graphics ...). Then the routine `matlab` (in the sub-directory `routines/default`) is called twice. The first call corresponds to a silent initialization of the database and the second one to the execution of the start-up.

1.1 The main routine matlab

The initializations of the database are done by the include file `<scilab dir>/routines/stack.h`; the other initialization done by `matlab` is `nunit` which is the maximum number of logical units and then `matlab` calls the routine `inibas` (in the sub-directory `routines/system`). Numerous initializations are done by this routine such as : the unit numbers for the input and the output,the predefined variables, the character set,...

After that `matlab` calls the routine `matla0` (in the sub-directory `routines/sytem`) which is a simple call of the routine `parse` (in the sub-directory `routines/system`) followed by a call to one of the interfaces. This is done by the mean of the include file `callinter.h` (in the sub-directory `routines`). `parse` is the Scilab parsing function : after examination of a command, `parse` returns `fun` which is the number of the interface to be called by `matla0`.

1.2 The Scilab parsing function and the interfaces

After the analysis of a Scilab command by `parse` a Scilab function (written in Scilab language) or a fortran (or C) routine can be called. In the last case the call is done by the corresponding interface : all the interfaces are in the sub-directory `routines/interf` and each of them contains the sequence of its routines.

The organization of this internal structure is represented by figure 1.

2 The databasis

2.1 The fortran structure

The leading program of Scilab is written in fortran and so the database is organized in fortran arrays. This database is composed of the 4 following arrays (in fact 3 arrays but one of them is interpreted in two different manners):

- Names of the variables :
  `IDSTK(NSIZ,LSIZ)` integer array . `IDSTK(1:NSIZ,K)` is the code of the name of the variable number `K`.

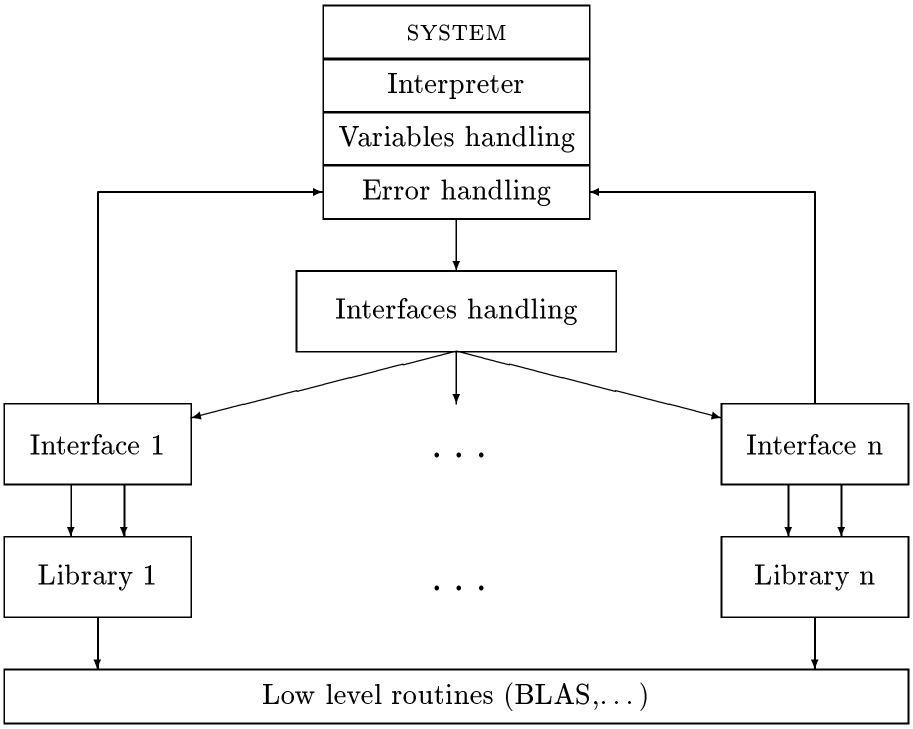


Figure 1: Internal structure of Scilab

- Addresses of the starting location :
  `LSTK(LSIZ)` vector of integers. `LSTK(K)` is the adress of the starting location of the variable `K` in the stack `STK`, and `LSTK(K + 1) - 1` is the adress of the last word of this variable in `STK`.

- Definitions of all the variables :
  `STK(VSIZ)` is the double float vector of the definitions of all the variables known in Scilab and the working area.

- `ISTK` vector of integers "equivalent" to `STK` (occupying the same place in the memory).

The description of the database is completed by 4 integers :

- Maximum number of variables:
  `ISIZ` is the dimension of the arrays `IDSTK` and `LSTK`. `ISIZ` is the maximum number of variables (permanent and temporary) which can be managed simultaneously by the system (for example 500).

- Dimension of the stack `STK`:
  `VSIZ` . `VSIZ` is the size of the stack defined in double precision words containing the variables (permanent and temporary) and the working area (for example 2000000). `VSIZ` and `ISIZ` are constants which can only be changed by modifying the assigned in the include file `routines/stack.h`

- Location limit of temporary variables:
  `TOP` pointer on the arrays `LSTK` and `IDSTK`: the variables with a number from 1 to `TOP` are temporary variables (parameter of a fonction , evaluation of expressions, ...,). `LSTK(TOP+1)` is the current first free address of the stack `STK`.

- Location limit of permanent variables:
  `BOT`. The variables numbered from `BOT` to `ISIZ - 1` are permanent variables (variables created by an assignment : `a=expr...`). `LSTK(BOT)-1` is the last free address of the stack `STK`. The relation `TOP + 1 < BOT` must be always satisfied (to avoid overwritting). The figure 2 presents the 3 arrays of the database. Then the figure 3 describes the stack, the figure 4 gives the correspondance between the indices of the array `STK-ISTK`. This correspondance is again presented in figure 5 and the detail of the decomposition between the description of a variable and its values is in figure 6.

Remark:
A double-float is equivalent to two integers. Converting the address from `STK` to `ISTK` is done through the fonctions `iadr` and `sadr`.

We have the following relations:

```
il1 = iadr(l)
l = sadr(il1)
sadr(il2) = l + 1
```

The database is transmitted to the different routines by the labelled commons :

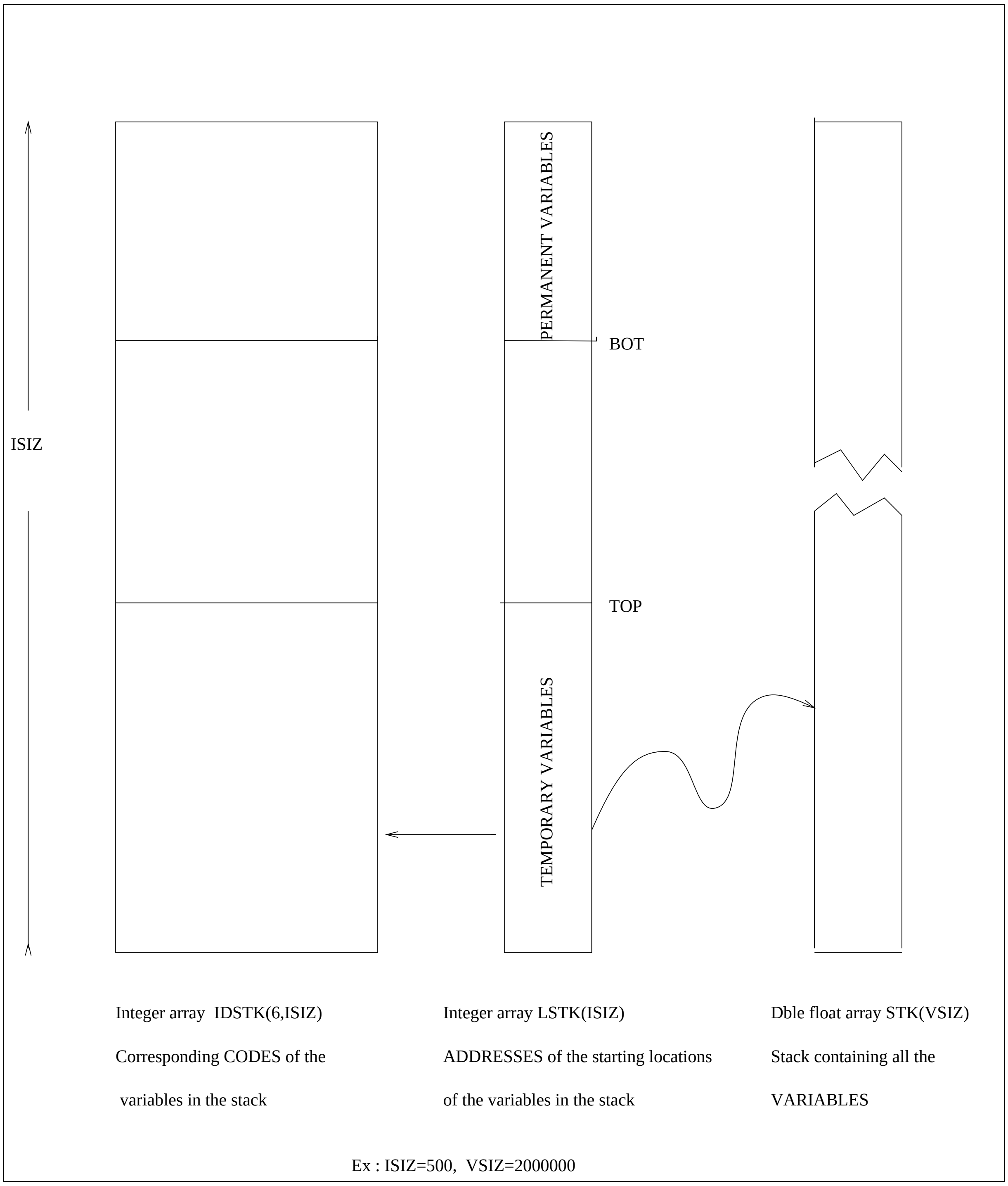


Figure 2: The 3 arrays of the database

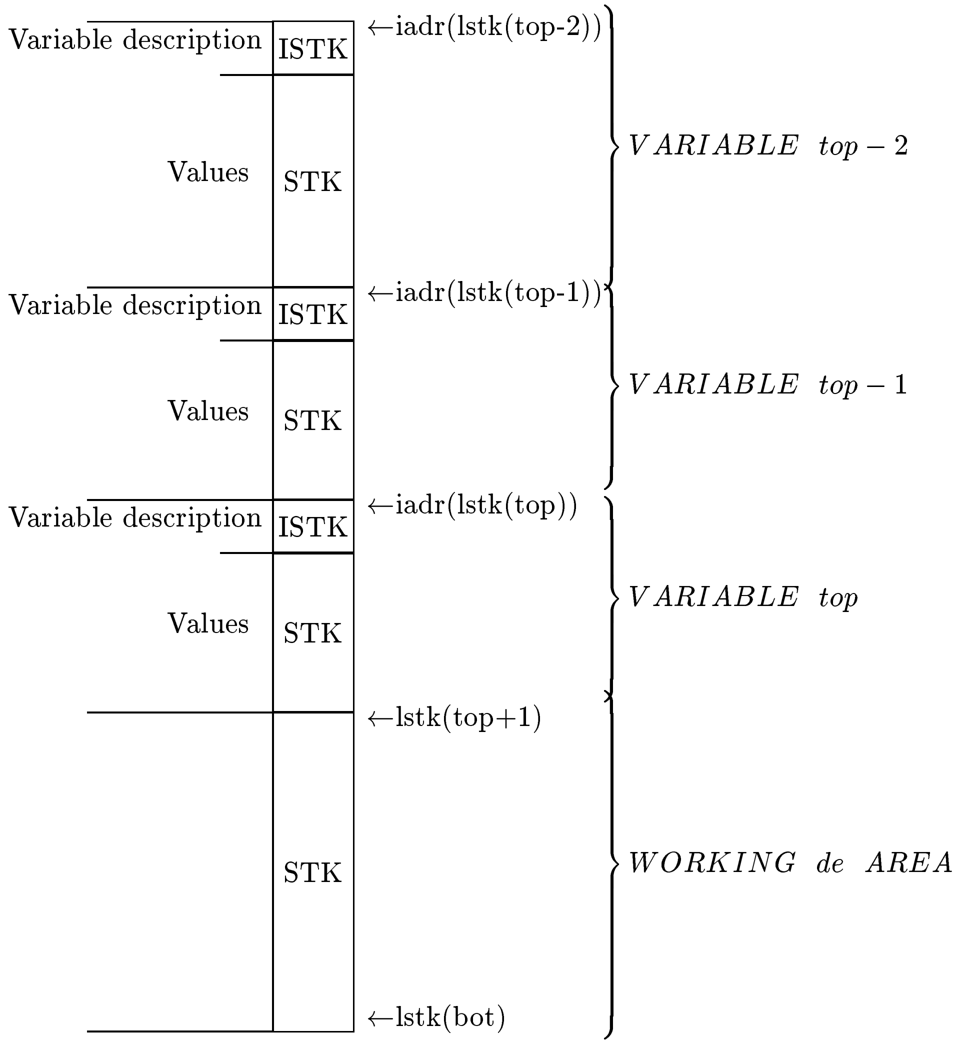


Figure 3: Description of the stack

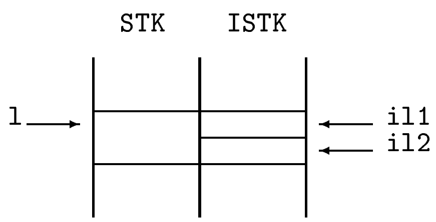


Figure 4: STK to ISTK conversion

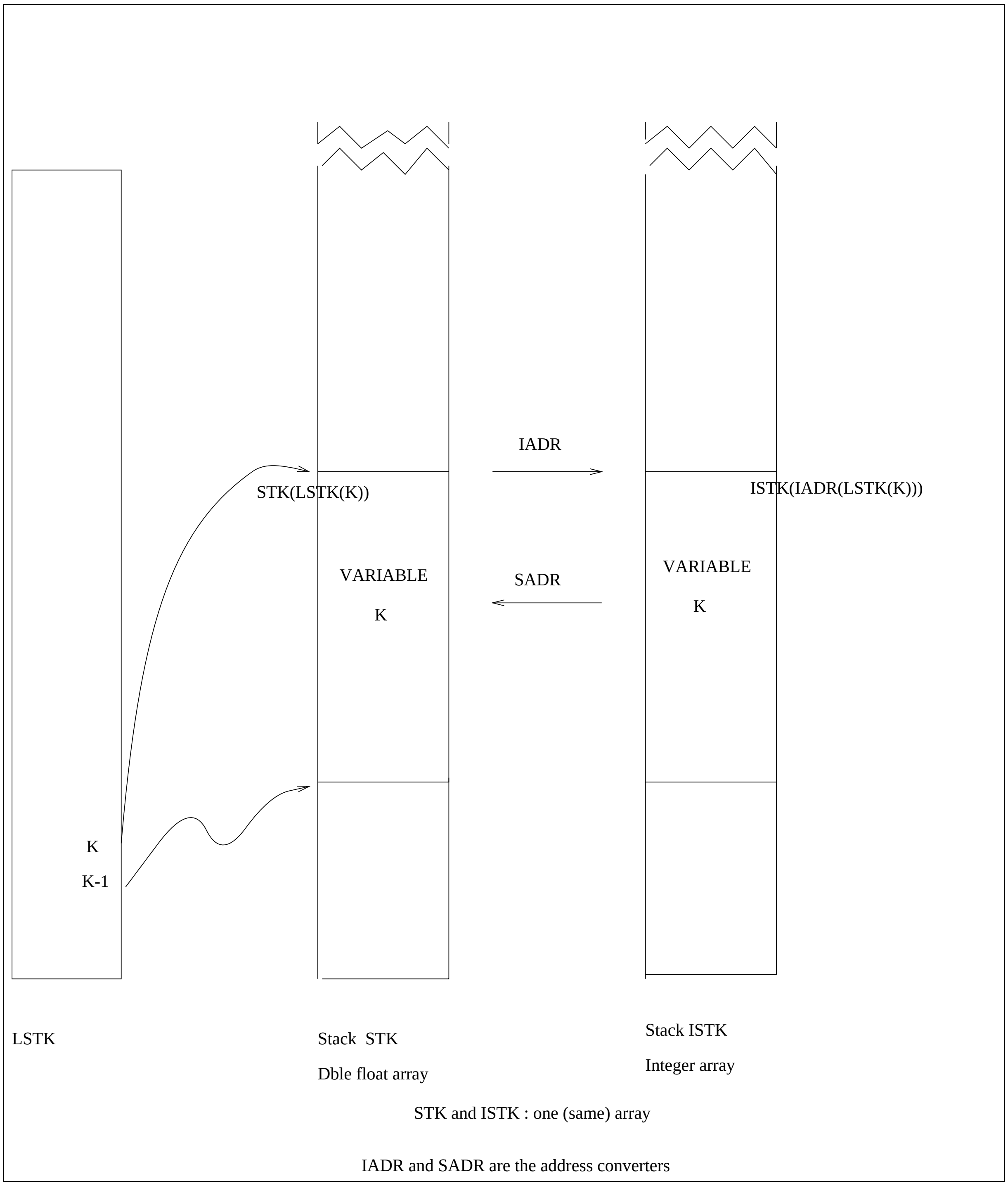


Figure 5: Correspondance of the arrays

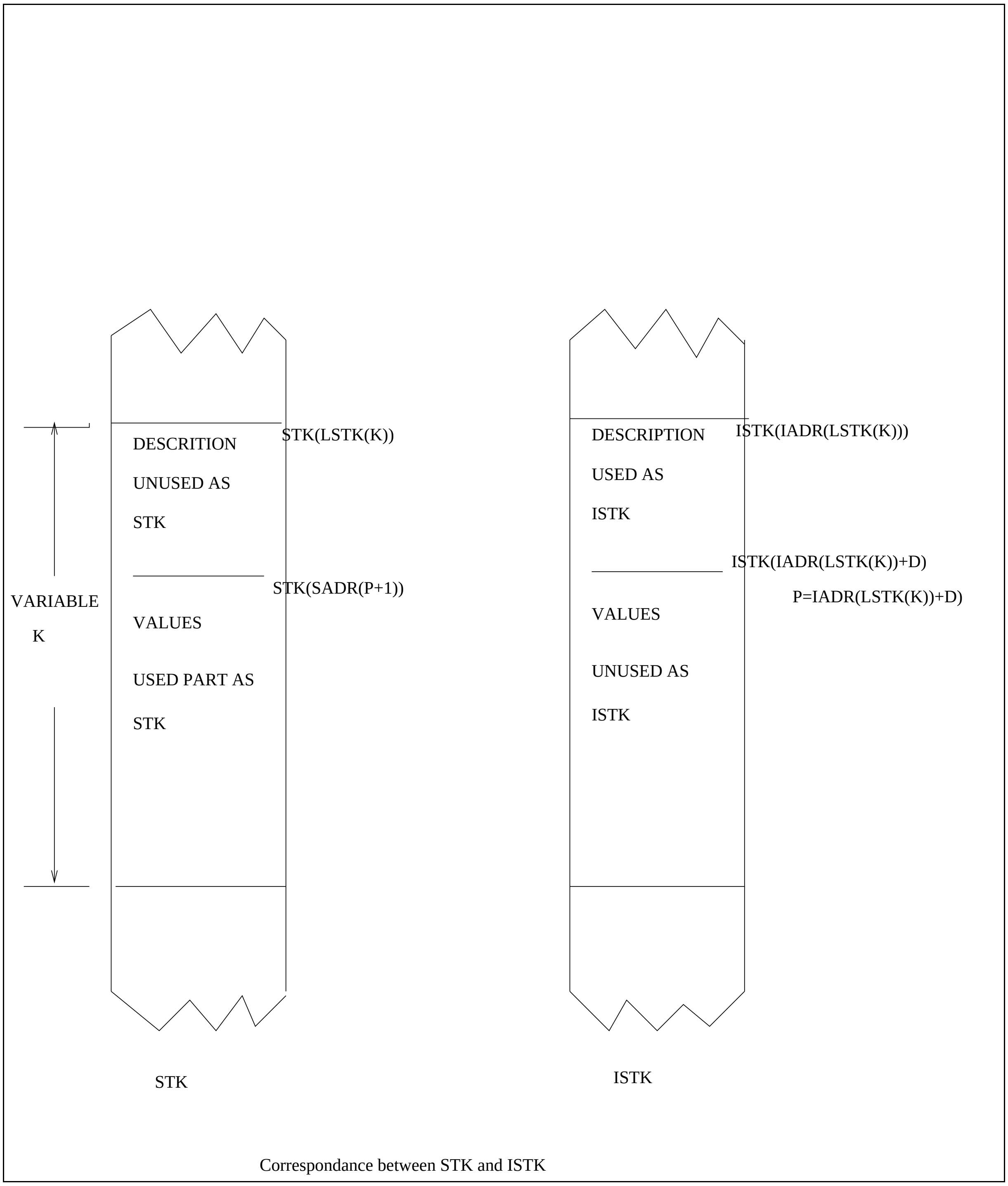


Figure 6: Description of a variable location

```
COMMON /STACK/ STK
COMMON /VSTK/ BOT,TOP,IDSTK,LSTK,LEPS,BBOT,BOTO
```

defined in the include file `stack.h`. `LEPS` is the address of `STK` where is stored the value of the machine precision $b^{(1-t)}$ (b=base, t= length of the mantissa).

2.2 Coding the different types of variables

Let `k` be the number of the variable considered and `il= iadr(LSTK(k))`: as seen before `il` is a pointer towards the first word of the stack `ISTK` corresponding to this variable. `ISTK(il)` defines the type of the variable. We will now consider the differents datatypes and present their corresponding description and the organization of the part of the stack containing the values.

2.2.1 Scalar matrix type

- `ISTK(il)` is equal to 1.
- `ISTK(il+1)` contains the line number m of the matrix.
- `ISTK(il+2)` contains the column number n.
- `ISTK(il+3)` is $= 0$ if the matrix coefficients are real and $= 1$ if they are complex numbers.

Let `l1= sadr(il+4)`, then

- `STK(l1:l1+m*n-1)` are the real parts of the matrix elements, the element `(i,j)` is stored in `STK(l1+(i-1)+(j-1)*m)`.

If `ISTK(il+3)` is equal to 1, then

- `STK(l1+m*n:l1+2*m*n-1)` are the imaginary parts of the elements, the part (i, j) is stored in `STK(l1+m*n+(i-1)+(j-1)*m)`.

The figure 7 presents the description of this type.

2.2.2 Character string matrix

If `ISTK(il)` is equal to 10 :

- `ISTK(il+1)` contains the number of the lines m of the matrix,
- `ISTK(il+2)` contains the number of the columns n of the matrix,
- `ISTK(il+3)` not used.

The character matrix datatype is represented by fig. 8.

In Scilab the characters are coded by integers (cf 2.3), the function `cvstr` (see `routines/system/cvstr.f`) translates the ASCII code to the Scilab code and conversly.

2.2.3 Polynomial matrix

This datatype is represented by fig. 9.

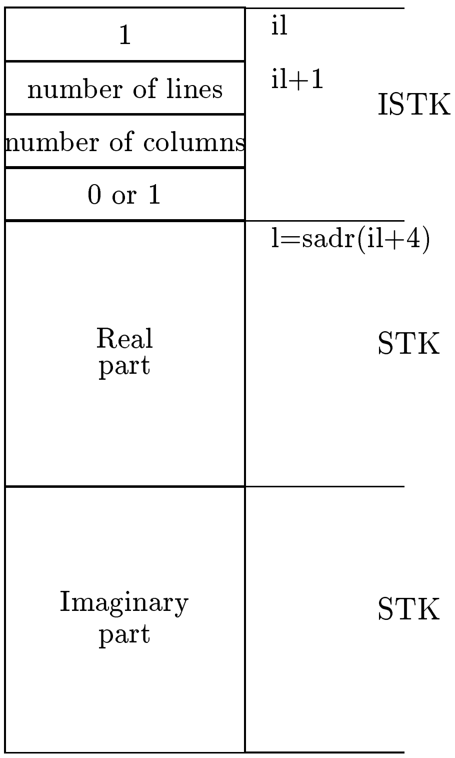


Figure 7: Real or complex matrix

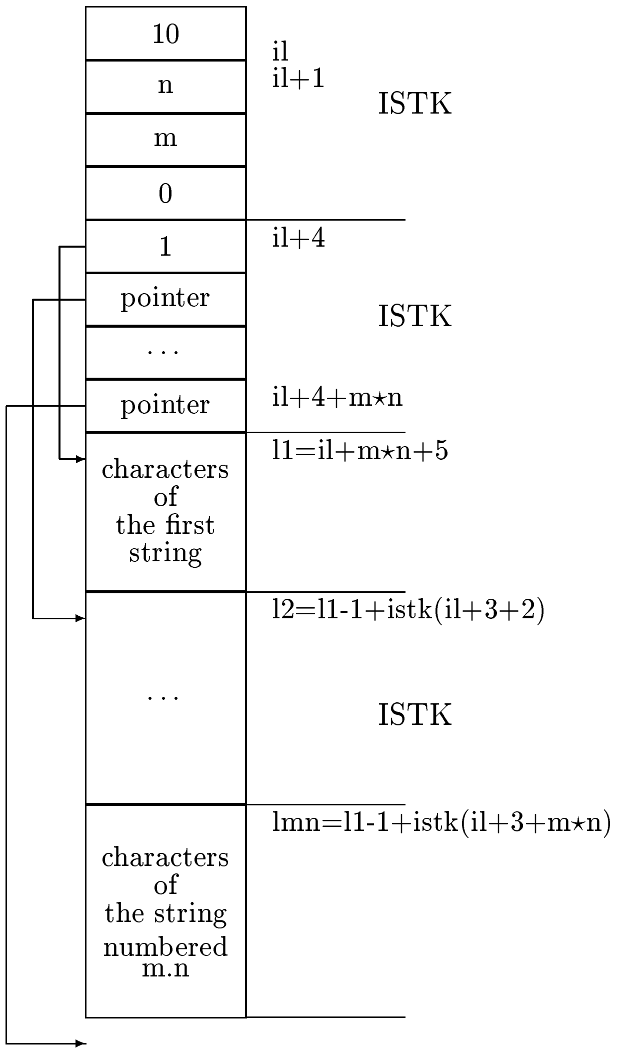


Figure 8: Character string matrix. m : number of lines ,n : number of columns

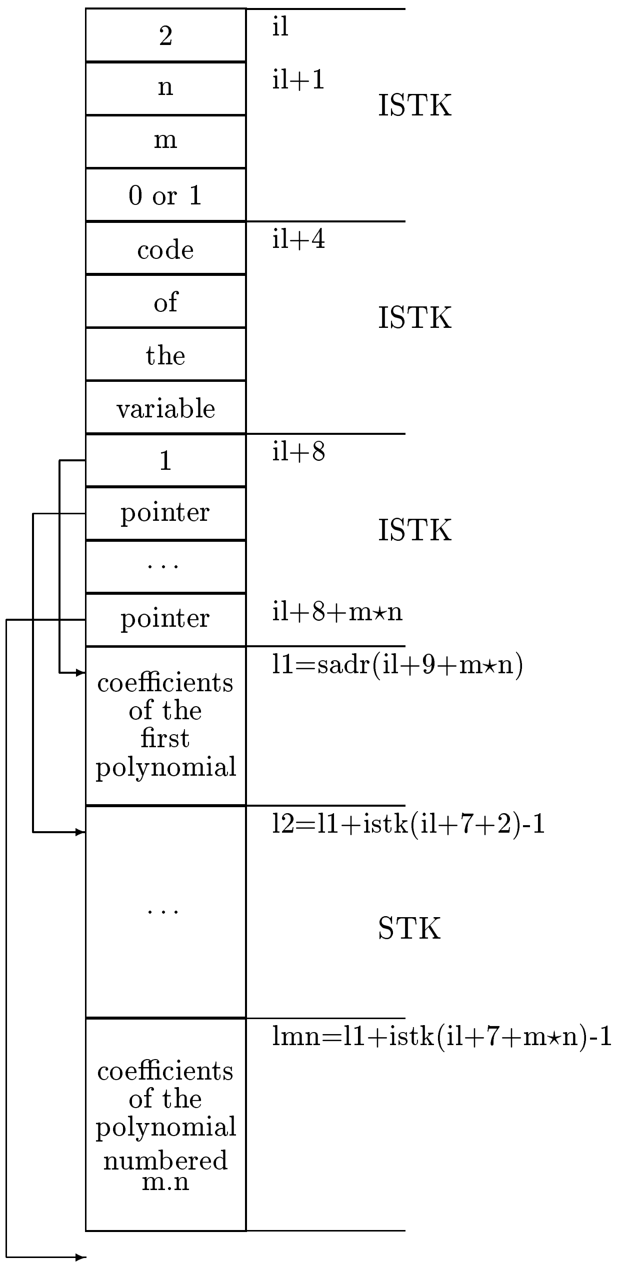


Figure 9: Polynomial matrix. m : number of lines , n : number of columns

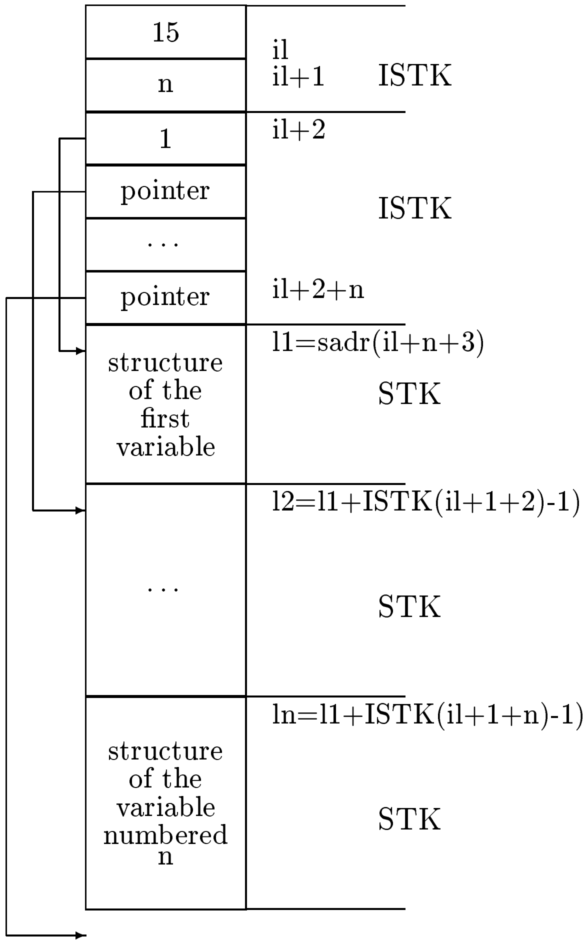


Figure 10: List. n : number of the elements of the list

2.2.4 Lists

`ISTK(il)=15`

- `ISTK(il+1)` contains the number of elements n of the list.

  If `ilp=il+2` and `l=sadr(ilp+n+1)-1`, then:

- `ISTK(ilp+(i-1))` contains the pointer `li` such that `STK(l+li)` is the first word of the element `i`, the number of the words (de mots (in `STK`) of the element numbered `i` is given by
  `ni=ISTK(ilp+i) - ISTK(ilp+i-1)`.

- `STK(l+li:l+li+ni-1)` contains the whole structure of the variable corresponding to this element.

This datatype is described by fig. 10.

2.2.5 Functions

`ISTK(il)` is equal to 11. In this case its description is divided in 3 fields : the first one describes the output parameters of the function, the second one is devoted to the input parameter and the last one contains the set of instructions.

Si $\texttt{ils} = \texttt{il} + 1$:

- `ISTK(ils)` contains the number m of the output parameters.
- `ISTK(ils+1:ils+NSIZ*n)` contains the names of the output variables in Scilab code form compressed on `NSIZ` integers.

Let $\texttt{ile} = \texttt{ils} + NSIZ * n + 1$, then

- `ISTK(ile)` contains the number m of the input parameters.
- `ISTK(ile+1:ile+NSIZ*m)` contains the names of the output variables in Scilab code form compressed on `NSIZ` integers.

Let $\texttt{ilt} = \texttt{ile} + NSIZ * n + 1$, then:

- `ISTK(ilt)` contains the length (number of characters) l of the code of the function.
- `ISTK(ilt+1:ilt+l)` the code of the function in Scilab code.

For the compiled functions `ISTK(ilt+1:ilt+l)` contains a sequence of integers fefining the compiled code. The integers equal to 99 are lines separator.

2.2.6 Library

$\texttt{ISTK(il)} = 14$

- `ISTK(il+1)` contains the number `nf` of characters of the name of the file containing the functions.
- `ISTK(il+2:il+1+nf)` contains the sequence of characters in Scilab code form.

If $\texttt{ilh} = \texttt{il} + 2 + \texttt{nf}$:

- `ISTK(ilh)` contains the number `nh` of characters of the name of the file containing the “help”.
- `ISTK(ilh+1:ilh+nh)` contains the Scilab code of the characters.

Let $\texttt{iln} = \texttt{ilh} + \texttt{nh} + 1$:

- `ISTK(iln)` contains the number `nm` of functions, and `ISTK(iln+2i-1:iln+2i)` contains the compact code of the name of the `i`-th function, for `i` from 1 to `nm`.

CHARACTERS	SCILAB CODES
0-9	0-9
a-z or A-Z	10-35
_	36
#	37
~	38
blank	40
(	41
)	42
;	43
:	44
+	45
-	46
*	47
/	48
\ or $	49
=	50
.	51
,	52
' or "	53
[or {	54
] or }	55
%	56
|	57
&	58
< or `	59
>	60
~ or @	61

Table 1: Scilab codes for known characters

2.3 The code of the Scilab characters

The following table 1 gives the internal code of the the Scilab characters.

The upper-case characters and some equivalents are coded by the lower-case code with a sign change.

The function `cvstr` (see `routines/system/cvstr.f`) translates the code ASCII to the Scilab code and conversely.

```
      subroutine cvstr(n,line,str,job)
c!purpose
c     translates a character string written in Scilab code
c     to a standard string
c     the eol (99) are replaced by !
c
c!calling sequence
c     call cvstr(n,line,str,job)
c
c     with
c
c     n: integer, length of the string to be translated
c
c     line: vector of integers which are the codes of the characters
c
c     string: character, contains ASCII characters
c
c     job: integer, if equal to 1: code-->ascii
c                   if equal to 0: ascii-->code
```

3 Scilab Fortran Interfaces

This section describes the rules to follow for writing an interface allowing to add a new primitive to the system; of course we forget here the parts depending on the host computer (compiler, linker,...). The Scilab structure is resumed in figure 1.

3.1 Interfaces handling

The link between the Scilab primitives and the corresponding interfaces is done by the routine `funtab`. This routine is **automatically produced** by the program `bin/newfun` with the file `routines/default/fundef`. This routine handle two tables initialized by `DATA`.

The first table (`funn(NSIZ,funl)`) contains the coded names of the functions known by Scilab, `funl` being the number of these functions.

The other table (`funp(funl)`) define 2 integers `fun` and `fin` for each known function; these 2 integers are represented by the integer `100*fun+fin`, where :

- `fun` indicates the interface program implementing the function.
- `fin` indicates the function inside the interface program.

For adding a new primitive it is necessary to add its name and the value 100⋆fun+fin in the file `routines/default/fundef` following the format specification.

Example :

```
abs                       601    0
atan                      625    0
cos                       624    0
```

define the pointers towards the Scilab primitives `abs`, `atan`, `cos`.

Running the program `bin/newfun` or more easily the Makefile allows then the generation of the file `funtab.f`.

3.2 Interface routine

When a Scilab function is called, the system calls the corresponding interface program after having defined the variables `fin`, `lhs` and `rhs` in the common `/com/` and configured the database: `common/STACK/` and `/VSTK/`.

3.2.1 The variables lhs and rhs

These variables indicate the numbers of left-parameters (`lhs`) and right-parameters (`rhs`) used for the call of the function.

EXAMPLE: `[x,y]=foo(a,b,c)` gives `lhs` = 2 and `rhs` = 3.

The interface must check if these parameters numbers agree with the allowed values for the functions. Different variants can be defined for a unique function by using the possibility of a variable length for the parameters list.

In case of incompatibility for the parameter list, the interface program calls the error handling program (`error`) with the code 41 (`lhs`) or 42 (`rhs`) and return the prompt.

3.3 A working example

We define here a fortran routine and then we interface it with Scilab. We are in the case of the Scilab data types are not simple fortran types. The code of this routine is the following :

```
      subroutine dmptr(pm,d,n,tr,dt)
c!purpose
c     Computes the trace of a square polynomial matrix pm
c!Calling sequence
c
c    subroutine dmptr(pm,d,n,tr,dt)
c    double precision pm(*),tr(*)
c    integer d(*),dt
c
c     pm : array of polynomial matrix coefficients:
c          pm=[coeff(pm(1,1)),coeff(pm(2,1)),...coeff(pm(n,n))]
c     d  : array of pointer on the first coefficient of pm(i,j)
c     d=[1,1+degree(pm(1,1)),1+degree(pm(1,1))+degree(pm(2,1)),...1+...+degree(pm(n,n))]
c     n  : size of pm matrix
c     tr : array of trace polynomial coefficients
```

```
c     dt : degre of trace polynomial
c!
      double precision pm(*),tr(*)
      integer d(*),dt,n
c
      integer deg
c     computes trace degree
      dt=0
      do 01 i=1,n
         ii=i+(i-1)*n
         dt=max(dt,d(ii+1)-d(ii))
 01   continue
c     initialize tr coefficients to 0.0d0
      call dset(dt,0.0d0,tr,1)
c sum of the diagonal elements of pm
      do 10 i=1,n
         ii=i+(i-1)*n
         deg=d(ii+1)-d(ii)
         write(*,*) i,ii,deg

         if(deg.gt.0) then
            do 05 k=1,deg
               tr(k)=tr(k)+pm(d(ii)-1+k)
 05         continue
         endif
 10   continue
      return
      end
```

We now write the code `newint` corresponding to the new interface i.e. defining the Scilab command `tr=trace(mp)`

```
      subroutine newint
      include 'routines/stack.h'
      double precision sr,si
      integer iadr, sadr, id(nsiz)
      iadr(l)=l+l-1
      sadr(l)=(l/2)+1
      rhs = max(0,rhs)
      lw = lstk(top+1)
      l0 = lstk(top+1-rhs)
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      if (fin .eq. 1) then
         if (rhs .ne. 1) then
            call error(39)
            return
         endif
         if (lhs .ne. 1) then
```

```
            call error(41)
            return
         endif
         il1 = iadr(lstk(top-rhs+1))
         if (istk(il1) .eq. 1) then
            if (istk(il1+1) .ne. istk(il1+2)) then
               err=1
               call error(20)
               return
            endif
            n1 = istk(il1+1)
            l1 = sadr(il1+4)
            it1 = istk(il1+3)
            sr = 0.0d0
            si = 0.0d0
            do 10 i = 1,n1
               sr = sr+stk(l1+(i-1)+(i-1)*n1)
 10         continue
            if(it1 .eq. 1) then
               do 11 i = 1,n1
                  si = si+stk(l1+n1*n1+(i-1)+(i-1)*n1)
 11            continue
            endif
            stk(l1) = sr
            if(it1 .eq. 1) stk(l1+1) = si
            istk(il1+1) = 1
            istk(il1+2) = 1
            if (si .eq. 0.0d0) istk(il1+3) = 0
            lstk(top+1) = l1+1+it1
         elseif (istk(il1) .eq. 2) then
            if (istk(il1+1) .ne. istk(il1+2)) then
               err = 1
               call error(20)
               return
            endif
            n1 = istk(il1+1)
            id1 = il1+8
            l1 = sadr(id1+n1*n1+1)
            it1 = istk(il1+3)
            idt=0
            do 20 i=1,n1
               ii=i+(i-1)*n1
               idt=max(idt,istk(id1+ii)-istk(id1-1+ii))
 20         continue
            lr = lw
            err = lr+idt*(it1+1) - lstk(bot)
            if (err .gt. 0) then
               call error(17)
```

```
            return
         endif
         call dmptr(stk(l1),istk(id1),n1,stk(lr),idt)
         if (it1 .ne. 0) then
            l1i = l1+istk(id1+n1*n1)-1
            lri = lr+idt
            call dmptr(stk(l1i),istk(id1),n1,stk(lr+idt),idt0)
         endif
         istk(il1+1) = 1
         istk(il1+2) = 1
         istk(il1+3) = it1
         istk(il1+8)=1
         istk(il1+9)=idt+1
         l1 = sadr(il1+10)
         call dcopy((idt+1)*(it1+1),stk(lr),1,stk(l1),1)
         lstk(top+1)=l1+(idt+1)*(it1+1)
      else
         buf='First argument is nor a matrix nor a polynomial matrix'
         call error(999)
         return
      endif
   endif
   end
```

We now reconsider the previous code with the comments for the different steps of the procedure

```
      subroutine newint
C INCLUDING THE DATABASE PARAMETERS
C REPLACE SCIDIR BY THE SCILAB PATH
      include 'SCIDIR/routines/stack.h'
      double precision sr,si
      integer iadr, sadr, id(nsiz)
C DEFINITION OF THE ADDRESS CONVERTERS
      iadr(l)=l+l-1
      sadr(l)=(l/2)+1
      rhs = max(0,rhs)
C ADDRESSES OF THE BOUNDS OF THE LOCATIONS OF THE RIGHT HAND SIDE PARAMETERS
      l0 = lstk(top+1-rhs)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      if (fin .eq. 1) then
C BEGINNING OF THE CODE TO BE ADDED
C     SCILAB tr=trace(mp)
C     =================
C CHECK NUMBER OF CALLING RIGHT HAND SIDE (rhs) ARGUMENTS
         if (rhs .ne. 1) then
            call error(39)
```

```
            return
         endif
C CHECK NUMBER OF CALLING LEFT HAND SIDE (lhs) ARGUMENTS
         if (lhs .ne. 1) then
            call error(41)
            return
         endif
C CHECK NOW VARIABLE mp (NUMBER 1)
         il1 = iadr(lstk(top-rhs+1))
         if (istk(il1) .eq. 1) then
C il1 IS THE TYPE OF THE VARIABLE (SEE FIG. 3.6)
C++++++++++++++ STANDARD MATRIX CASE
            if (istk(il1+1) .ne. istk(il1+2)) then
C     .       Non square matrix
               err=1
               call error(20)
               return
            endif
            n1 = istk(il1+1)
C l1 ADDRESS OF MATRIX ELEMENTS (REAL PART)
            l1 = sadr(il1+4)
C it1 REAL/COMPLEX FLAG (0 or 1)
            it1 = istk(il1+3)
C INLINE PROCEDURE TO COMPUTE THE MATRIX TRACE
            sr = 0.0d0
            si = 0.0d0
            do 10 i = 1,n1
               sr = sr+stk(l1+(i-1)+(i-1)*n1)
 10         continue
            if(it1 .eq. 1) then
C     .    if complex computes imaginary part
               do 11 i = 1,n1
                  si = si+stk(l1+n1*n1+(i-1)+(i-1)*n1)
 11            continue
            endif
C STORE RESULT IN PLACE OF mp
            stk(l1) = sr
            if(it1 .eq. 1) stk(l1+1) = si
C SET RESULT SIZES FOR STACK HANDLING
            istk(il1+1) = 1
            istk(il1+2) = 1
            if (si .eq. 0.0d0) istk(il1+3) = 0
C RETURN ADDRESS OF THE FIRST FREE POSITION IN THE STACK
            lstk(top+1) = l1+1+it1
C END OF STANDARD MATRIX CASE
         elseif (istk(il1) .eq. 2) then
C++++++++++++++POLYNOMIAL MATRIX CASE (SEE FIG. 3.8)
            if (istk(il1+1) .ne. istk(il1+2)) then
```

```
C     .       non square matrix
               err = 1
               call error(20)
               return
            endif
            n1 = istk(il1+1)
C id1 STARTING ADDRES OF POINTERS
            id1 = il1+8
C l1 STARTING ADDRESS OF THE COEFFICIENTS
            l1 = sadr(id1+n1*n1+1)
C it1 REAL/COMPLEX FLAG (0/1)
            it1 = istk(il1+3)
C COMPUTING THE SIZE OF THE RESULT
            idt=0
            do 20 i=1,n1
               ii=i+(i-1)*n1
               idt=max(idt,istk(id1+ii)-istk(id1-1+ii))
 20         continue
C CKECKING AVAILABLE MEMORY
C SET RESULT POINTER TO THE FIRST FREE STACK ADDRESS
            lr = lw
C SET ERR TO THE NEGATIVE OF THE FREE SPACE
            err = lr+idt*(it1+1) - lstk(bot)
            if (err .gt. 0) then
C     .       Not enough memory
               call error(17)
               return
            endif
C CALLING THE PROCEDURE TO COMPUTE THE MATRIX TRACE
C     .    Real part
            call dmptr(stk(l1),istk(id1),n1,stk(lr),idt)
            if (it1 .ne. 0) then
C     .       Imaginary part
               l1i = l1+istk(id1+n1*n1)-1
               lri = lr+idt
               call dmptr(stk(l1i),istk(id1),n1,stk(lr+idt),idt0)
            endif
C DEFINITION OF THE RETURN VARIABLE
C SET THE RESULT HEADER (ISTK PART OF THE STACK)
C     .    row size
            istk(il1+1) = 1
C     .    column size
            istk(il1+2) = 1
C     . real/complex flag
            istk(il1+3) = it1
C     . pointers
            istk(il1+8)=1
            istk(il1+9)=idt+1
```

```
C MOVE COMPUTED VALUE IN ITS FINAL PLACE
            l1 = sadr(il1+10)
            call dcopy((idt+1)*(it1+1),stk(lr),1,stk(l1),1)
C RETURN ADDRESS OF THE FIRST FREE POSITION IN THE STACK
            lstk(top+1)=l1+(idt+1)*(it1+1)
C     . End of polynomial matrix case
         else
C+++++++++++++INVALID ARGUMENT TYPE CASE
            buf='First argument is nor a matrix nor a polynomial matrix'
            call error(999)
            return
         endif
C END OF TRACE FUNCTION
      endif
      end
```

After that we have to compile the routines `dmptr.f` and `newint.f`. Then we use the Scilab function `addinter` to link the new called fortran routine `dmptr.o` and the new interface calling fortran routine `newint.o` with Scilab. The last argument of `addinter` is the calling name for Scilab (for a complete description see the on-line help of `addinter`). During all the Scilab session, `mytrace` remains defined and can be used as predefined function.

```
getf('SCI/macros/util/addinter.sci')
addinter('dmptr.o newint.o','newint','mytrace')
a=diag([%s+1,2,3,4])
mytrace(a)
```

The result is :

```
-->getf('SCI/macros/util/addinter.sci')

-->addinter('dmptr.o newint.o','newint','mytrace')

linking newint_ defined in dmptr.o newint.o  with Scilab


-->a=diag([%s+1,2,3,4])
 a  =

!   1 + s    0     0     0  !
!                           !
!   0        2     0     0  !
!                           !
!   0        0     3     0  !
!                           !
!   0        0     0     4  !
```

```
-->mytrace(a)
 ans  =

    10 + s
```

It may be preferable to add definitively a new interface to Scilab. In this case the easiest way for a user is to give the name `matusr` to the interface entry point and to replace the standard `<scilab dir>/routines/default/matusr.f` file by his own file. Then the user has to add the new function names in `<scilab dir>/routines/default/fundef` and execute make.

List of Figures

List of Tables