SCILAB

Глава 4. Базовые примитивы

Базовые примитивы будут рассмотрены обзорно. Подробно смотрите в описании "Manual".

Содержание главы:

Среда Scilab и вход/выход (Input/Output)

Эта часть описывает наиболее важные аспекты среды Scilab:
1) настройка конфигурации системы;
2) обмен данными между пакетом Scilab и внешней средой: запись данных во внешнюю среду (файл) и считывание данных из файла в Scilab.

Стартовая загрузка команд пользователем

При стартовой запуске Scilab пользователь может автоматически загрузить функции, библиотеки, переменные, и выполнить команды, используя файл **.scilab** в головном каталоге. Это необходимо, если Вы хотите запускать сессию Scilab в "background" моде. Если некоторые функции или библиотеки часто используются. В этом случае команды **getf exec** или **load** могут быть использованы в файле **.scilab** для автоматической загрузки. Стандартные "On-line" справки содержатся в ASCII файлах с расширением **.cat** в каталоге **\scilab\man** и далее по соответствующим разделам (подкаталогам).

Пример получения справки.

```
--> help classmarkov
```

Для собственных библиотек и функций можно создать собственные описания. Для этого

просто надо создать ASCII-файл с расширением **.cat** в своем каталоге, добавить путь к этому новому каталогу в переменную **%helps**. Как это делается, прочтите в файле **\scilab\man\readme**).
Можно посмотреть стандартное содержание **%helps**:

```
-->%helps
%helps =
column 1
!D:/SCILAB/man/programming !
!                          !
!D:/SCILAB/man/graphics    !
!                          !
!D:/SCILAB/man/elementary  !
!                          !
!D:/SCILAB/man/fileio      !
!                          !
!D:/SCILAB/man/functions   !
!                          !
```

и так далее.
Scilab загружается с определенным количеством переменных и примитивов. После загрузки команда **who** дает список имеющихся переменных. Команда **whos()** дает этот список более детально. Переменные могут быть записаны во внешние бинарные файлы с помощью команды **save** и впоследствие загружены извне в среду Scilab с помощью команды **load**. Для уничтожения переменных применяется команда **clear**. Библиотечные функции загружаются с помощью команды **lib**. Список функций, доступных в библиотеке, может быть получен, используя команду **disp**.

Команда **disp** показывает содержание переменных различного типа.
Синтаксис
disp(x1,[x2,...xn])
Значения **xi** изображаются в текущем формате. **xi** -различные объекты (матрицы констант, строки, функции, списки и т. д.) **disp(name_library)** позволит нам увидеть список функций, доступных в библиотеке с соответствующим названием.

Примеры.

```
x=5;y=8;a=[2 3 4;5 6 7];
disp(x,y,a)
```

Результат:

```
! 2. 3. 4. !
! 5. 6. 7. !

disp([1 2],3)
deff('[]=%t_p(l)','disp(l(3),l(2))')
disp(tlist('t',1,2))
disp(alglib) // дает список доступных функций из алгебраической библиотеки
```

Результат выполнения **disp(alglib)**:

```
Functions files location :SCI/macros/algebre/

aff2ab      classmarkov      chsolve      chfact
companion      colcomp      coff
eigenmarkov      fullrfk      fullrf
genmarkov      glever      givens
householder      im_inv      kernel      kroneck
linsolve      nlev      orth      psmall      pbig
pencan      polar      penlaur      projspec      pen2ea
proj      quaskro      range      rowshuff      rowcomp
randpencil      spantwo      sqroot      spanplus
spaninter      trace
```

Если мы хотим узнать синтаксис употребления функции **polar** из этой библиотеки, то следует запросить `disp(polar)` (указывать имя библиотеки не нужно). Назначение этой функции можно узнать, получив "help" по этой команде в каталоге **\scilab\man\linear** в файле **polar.man**. Почему-то имя этого каталога не совпадает с именем библиотеки. Для полной справки используйте **help polar**. Сами библиотеки находятся в каталоге **\scilab\macros**.
Пример.

```
disp(polar)
[ro,teta]=polar(a)
```

Как узнать о загруженных переменных в систему scilab?

Применяйте команду **who** для короткой формы отчета и **whos** - для длинной.

Синтаксис команды **who**
who
who()
names=who('local')
[names,mem]=who('local')
names=who('global')
[names,mem]=who('global')
Описание
who дает текущие имена переменных.
who('local') или **who('get')** возвращает текущие имена переменных и память.
who('global') возвращает глобальные имена переменных и память.

who и **who()** дают один и тот же результат.

Применение **who** непосредственно после загрузки Scilab в частном случае дала нам следующий результат:

```
your variables are...
startup ierr demolist %scicos_display_mode
scicos_pal %scicos_menu %scicos_short %helps
MSDOS home PWD TMPDIR percentlib soundlib
xdesslib utillib tdcslib siglib s2flib roblib optlib
metalib elemlib commlib polylib autolib armalib alglib
intlib mtlblib WSCI SCI %F %T %z
%s %nan %inf $ %t %f %eps
%io %i %e
using 5860 elements out of 1000000.
and 46 variables out of 1791
```

Синтаксис команды **whos**
whos()
whos -type typ
whos -name nam
Параметры
typ : имя выбранного типа переменной. Для определения типа переменной (поле - type)смотри команду **typeof**
nam : первый символ выбранных имен

Пример 1.

```
whos -name %
```

Результат:

```
Name Type Size Bytes

%_opt string 1 by 1 48
%_sel string 1 by 1 32
%scicos_display_mode constant 1 by 1 24
%scicos_menu list 2304
%scicos_short string 11 by 2 376
%helps string 24 by 2 4584
%F boolean 1 by 1 24
%T boolean 1 by 1 24
%z polynomial 1 by 1 56
%s polynomial 1 by 1 56
%nan constant 1 by 1 24
%inf constant 1 by 1 24
%t boolean 1 by 1 24
%f boolean 1 by 1 24
%eps constant 1 by 1 24
%io constant 1 by 2 32
%i constant 1 by 1 32
```

Пример 2.

```
--> whos -type boolean
```

Результат:

```
Name Type Size Bytes

MSDOS boolean 1 by 1 24
%F boolean 1 by 1 24
%T boolean 1 by 1 24
%t boolean 1 by 1 24
%f boolean 1 by 1 24
```

Пример 3.

code>whos() Получаем информацию о всех имеющихся переменных и библиотеках

Пример 4.

```
whos -name a
```

Получаем информацию обо всех переменных, названия которых начинаются с буквы "a". Если мы введем в процессе сессии новую переменную, то она появится в списке, получаемом с помощью команды **who**.

Как уничтожить переменные?

Для этого служит команда **clear**.

Синтаксис

clear a уничтожение переменной a.

clear уничтожение всех переменных, кроме переменных защищенных командой **predef**. Для полного уничтожения всех переменных следует применить пару команд: **predef(0)** и **clear**.

Замечание: Нельзя уничтожить встроенные в Scilab служебные переменные, начинающиеся со знака **%**.

```
clear %i даст сообщение
!--error 13
redefining permanent variable
```
Для уничтожения глобальных переменных существует команда **clearglobal**

Как сохранить данные на диске?

С помощью команды **save** или с помощью меню **File-Save**. Данные в файл записываются в бинарном формате. По-видимому, следует давать этому файлу и соответствующее расширение.

Синтаксис
save(filename [,x1,x2,...,xn])
save(fd [,x1,x2,...,xn])
Параметры
filename : имя файла, включающее пути к нему(тип character string)
fd : файловый дескриптор для последующего вызова командой **mopen**
xi : имена записываемых Scilab переменных

save(filename) без указания переменных запишет в файл все текущие переменные.
save(fd) запишет все текущие переменные в файл, определенный дескриптором fd.
save(filename,x,y) или **save(fd,x,y)** запишет только поименованные переменные x и y.

Загрузка переменных из образованного с помощью команды **save** файла на магнитном диске в пакет Scilab осуществляется командой **load**.

Пример.
```
save('D:\zzz\mymat.bin') // Загрузка всех текущих данных в файл

b=[11 22 33;21 22 23;31 32 33];
a=789;
save('val.dat',a,b);
clear a // уничтожение переменной
clear b
load('val.dat','a','b');

// последовательная запись в файл
fd=mopen('TMPDIR/foo','wb')
for k=1:4, x=k^2;save(fd,x,k),end
mclose(fd)

// последовательная загрузка из файла
fd=mopen('TMPDIR/foo','rb')
for i=1:4, load(fd,'x','k');x,k,end
mclose(fd)

// добавление данных в старый файл
fd=mopen('TMPDIR/foo','r+')
mseek(0,fd,'end')
lst=list(1,2,3)
save(fd,lst)
mclose(fd)
```

Как загрузить в Scilab данные, записанные ранее в файл с помощью команды save?

С помощью команды **load**
Синтаксис
load(filename [,x1,...,xn])
load(fd [,x1,...,xn])

Параметры
filename строка, содержащая путь к файлу
fd : файловый дескриптор, связанный с открытием файла
xi : имена Scilab-переменных, заданные в виде строк

load(filename,'x','y') или **load(fd,'x','y')** загружает только переменные x,y.
Пример.

```
a=eye(2,2);b=ones(a);
save('d:\mydir\vals.dat',a,b);
clear a
clear b
load('d:\mydir\vals.dat','b');
```

В результате в среду Scilab будет загружена величина b.

Замечание: Команда **save** записывает данные в файл вместе с именами переменных. Если мы не знаем имена нужных переменных, по команде `load('vals.dat')` мы загрузим в среду Scilab все переменные из заданного файла "vals.dat" и можем с помощью команды **who** узнать его содержимое. Для лучшего контроля над данными все же лучше использовать команды **write** и **read**.

Запись и вызов данных

Хотя команды **save** и **load** удобны, гораздо больше контроля за передачей данных между файлами и средой Scilab предоставляют функции **read** и **write**, аналогичные функциям языка Fortran, либо функции **mfscanf** и **mfprint**, аналогичные функциям языка C. Без указания директория файлы данных образуются в каталоге **/scilab/bin/**.

Пример для любителей языка Fortran.

```
-->x=[1 2 %pi;%e 3 4]
x =

! 1.        2. 3.1415927 !
! 2.7182818 3. 4.        !

-->write("x.dat",x)
-->clear x
-->xnew=read("x.dat",2,3)
xnew =

! 1.        2. 3.1415927 !
! 2.7182818 3. 4.        !
```

Пример для любителей языка C.

```
-->x=[1 2 %pi;%e 3 4]
x =
! 1.         2. 3.1415927 !
! 2.7182818 3. 4.         !

-->fd=mopen("x_c.dat","w")
-->mfprintf(fd,"%f %f %f\n",x)

-->mclose(fd)
-->clear x
-->fd=mopen("x_c.dat","r")
-->xnew(1,1:3)=mfscanf(fd,"%f %f %f\n");
-->xnew(2,1:3)=mfscanf(fd,"%f %f %f\n");

-->xnew
xnew =
! 1.         2. 3.1415927 !
! 2.7182818 3. 4.         !
-->mclose(fd)
```

Замечание: В отличие от способа записи данных с помощью команды **save**, образованные вышеизложенными способами файлы "x.dat" и "x_c.dat" написаны в ASCII-кодах и являются читаемыми в текстовых редакторах и других прикладных программах. Имена переменных и их тип в этих файлах не содержатся. Это просто последовательность чисел, допускающая их считывание в различном виде.

Синтаксис команды **read**
[x]=read(file-desc,m,n,[format])
[x]=read(file-desc,m,n,k,format)
Параметры
file-desc : имя файла (строковая переменная) или дескриптор файла (целое число)
m, n : целые (размерности матрицы x). Если неизвестно число строк, следует установить m=-1, чтобы был считан весь файл.
format : строка, определяющая формат в стиле "Fortran".
k : целое число или вектор

Примеры формата.

```
(1x,e10.3,5x,3(f3.0))
(10x,a20)
```

Пример.
Пусть мы создали, неважно, каким образом, на диске файл "x_c.dat" со следующим содержанием:

```
1.000000 2.000000 3.141593
2.718282 3.000000 4.000000
```

Рассмотрим несколько вариантов считывания его содержания (полного или частичного) в среду Scilab с помощью оператора **read**:

```
-->xnew=read("x_c.dat",2,3)
xnew =
! 1.         2. 3.1415927 !
! 2.7182818 3. 4.         !

-->xnew=read("x_c.dat",1,3)
```

```
xnew =
! 1.         2. 3.1415927 !

-->xnew=read("x_c.dat",1,1)
xnew =
1.

-->xnew=read("x_c.dat",1,2)
xnew =
! 1. 2. !

-->xnew=read("x_c.dat",2,2)
xnew =
! 1.       2. !
! 2.718282 3. !

-->xnew=read("x_c.dat",2,1)
xnew =
! 1.       !
! 2.718282 !
```

Для большей наглядности лучше пользоваться командой **mfscanf**.

Пример для того же вышеиспользованного файла "x_c.dat".

```
-->fd=mopen("x_c.dat","r"); // Открытие файла
-->a1=mfscanf(fd,"%f");
-->a2=mfscanf(fd,"%f");
-->a3=mfscanf(fd,"%f");
-->a4=mfscanf(fd,"%f");
-->a5=mfscanf(fd,"%f");
-->a6=mfscanf(fd,"%f");
-->mclose(fd) // Желательно после считывания закрыть файл
```

В результате будем иметь:

```
a1 =1.
a2 =2.
a3 = 3.141593
a4 = 2.718282
a5 = 3.
a6 = 4.
```

Вариант, не имеющий прототипа в Fortran и C: использование функции **read** напрямую. Пример.

Пусть мы имеем файл данных `d:/my/my_file.dat` в формате ASCII в виде нескольких столбцов `x[i]` и `y[i]`. Число столбцов больше или равно трем. Число строк `N`, но нам его знать не обязательно. Хотим считать только первых три столбца из файла.

```
-->my_dat= read("d:/my/my_file.dat", -1, 3);
```

Результат:

```
my_dat =
! 1. 1.  6.  !
! 2. 4.  9.  !
! 3. 9.  14. !
! 4. 16. 21. !
! 5. 25. 30. !
! 6. 36. 41. !
```

Замечание: Цифра "`3`" в команде **read** означает, что мы считываем из каждой строки по три числа. `my_dat` - это матрица из трех столбцов и `N` строк. Выражение `my_dat(:, 2)` означает второй столбец файла данных.

Как открыть файл?

С помощью команды **mopen**.
Эта команда может быть использована для открытия файла способом, совместимым с процедурой открытия файла в языке C. Без аргумента swap-файл кодируется в "little endian IEEE format".
Синтаксис
[fd,err]=mopen(file [, mode, swap])
Параметры
file : символьная переменная. Путь к файлу, который мы хотим открыть.
mode : символьная переменная, контролирует моду, в которой мы открываем файл. Основные принимаемые значения mode такие же как и в языке C: **r** - для чтения, **w** - для записи, **a** - для добавления и **+** для открытия с обновлением. Кроме того **mode** может принимать значение **b** - для индикации бинарности файла.
swap : скаляр.
err : скаляр. Индикатор ошибок.
fd : целое положительное число. Параметр **fd** возвращает дескриптор.
r или **rb** : файл открыт для чтения.
w или **wb** : создает новый файл для записи, или открывает и обрезает его до нулевой длины.
a или **ab** : добавление (открывает файл для записи в конец существующей информации или создает файл для записи).
r+ или **r+b** : открывает файл для перезаписи (чтение и запись).
w+ или **w+b** : обрезает файл до нулевой длины или создает файл для перезаписи (обновление).
a+ или **a+b** : добавление (открывает файл для обновления, записи в конец существующего файла, или создания нового файла для записи).

Пример.
```
-->fd=mopen("my.dat","w")
```

Как закрыть файл?

Команда **mclose** закрывает файл, предварительно открытый командой **mopen**.
Синтаксис
err=mclose([fd])
mclose('all')
Параметры
fd : скаляр. Параметр **fd** -положительное целое число, используемое в качестве дескриптора функции открытия файла mopen.
err : скаляр. Является индикатором ошибки.
Описание
Если указание значения **fd** отсутствует, то за него принимается **fd** для последнего открытого файла.
mclose('all') закрывает все файлы, открытые с помощью **file('open',..)** или **mopen**.

Список часто употребляемых функций

Приведем перечень наиболее часто употребляемых функций. Многие из них подробно описаны в данном описании. Полное описание их дано в Help. Они могут быть рекомендованы для первоначального изучения. Обратите внимание на ссылки на другие команды в "help" для этих функций.

Элементарные функции: **sum**, **prod**, **sqrt**, **diag**, **cos**, **max**, **round**, **sign**,**fft**
Сортировка: **sort**, **gsort**, **find**
Специальные матрицы: **zeros**, **eye**, **ones**, **matrix**, **empty**
Линейная алгебра: **det**, **inv**, **qr**, **svd**, **bdiag**, **spec**, **schur**
Многочлены (полиномы): **poly**, **roots**, **coeff**, **horner**, **clean**, **freq**
Кнопки, диалог: **x_choose**, **x_dialog**, **x_mdialog**, **getvalue**, **addmenu**
Линейные системы: **syslin**
Генератор случайных чисел: **rand**
Программирование: **function**, **deff**, **argn**, **for**, **if**, **end**, **while**, **select**, **warn-ing**, **error**, **break**, **return**
Символы сравнения: ==, >=, >, =, **&** (**and**),| (**or**)
Выполнение файла: **exec**
Элементы отладки: **pause**, **return**, **abort**
Функции сглаживания (Spline), интерполяции: **splin**, **interp**, **interpln**
Символьные строки (Character strings): **string**, **part**, **evstr**, **execstr**
Графика: **plot**, **xset**, **driver**, **plot2d**, **xgrid**, **locate**, **plot3d**, **Graph-ics**
Ode solvers: **ode**, **dassl**, **dassrt**, **odedc**
Оптимизация: **optim**, **quapro**, **linpro**, **lmitool**
Взаимное связывание динамических систем: **scicos**
Сопряжение с процедурами, напмсанными на языках C и Fortran : **link**, **fort**, **addinter**, **intersci**

Нелинейные вычисления

Scilab обладает несколькими мощными нелинейными примитивами для моделирования и оптимизации. Для решения дифференциальных уравнений полезны следующие функции:
ode
odepack
dassl
dassrt
%ODEOPTIONS -полезная глобальная переменная
optim - для минимизации нелинейных функций

Доступны некоторые алгоритмы нелинейной оптимизации из библиотеки **modulopt**. (Смотри описание **help optim**)

Специальные Scilab-функции или процедуры языков C- и Fortran могут быть использованы в качестве аргументов в процедурах высокого уровня (таких, как **ode**, **optim**, **dassl**, **impl**, **intg**, **odedc**, **fsolve** и т. д.). Такие процедуры должны связываться со средой Scilab с помощью команды **link**.

Диалог XWindow

Иногда удобно открывать специальное диалоговое Window-окно для интерактивного ввода параметров функции или демонстрационной программы. Эта возможность реализуется с помощью команд **x_dialog**, **x_mdialog**, **x_choose**, **x_matrix**, **x_message** и **x_message_modeless**. Рассмотрим некоторые способы создания диалогового окна.

Способ 1.
С помощью команды **x_dialog**.
Синтаксис
result=x_dialog(labels,valueini)
Параметры
labels : вектор-стобец из строковых переменных, являющихся комментариями к диалогу.
valueini : вектор из символьных переменных размера **n**, инициализирующий начальные значения.
result : вектор отклика. Состоит из символьных переменных и имеет размерность **n**. Значения ему присваиваются при нажатии в диалоговом окне кнопки "Ok". При нажатии кнопки "Cancel" равен `[]`.

Пример.
```
res=x_dialog('enter a 2x3 matrix ',["0 0";"0 0";"0 0"])
```
Возможный результат после ввода данных:
```
res =

!1 2         !
!            !
!11 2        !
!            !
!sveta vera  !
```
Замечание: Этот способ плох тем, что на самом деле в данном примере Вы можете ввести не матрицу 2 на 3, а вектор (матрицу) совершенно произвольной длины (размера), если проигнорируете предложенный в виде начальных данных шаблон. Для поставленной задачи лучше было использовать команду **x_matrix**.

Способ 2.
Команда **x_mdialog** позволяет вводить в диалоговом режиме какие-либо переменные (вектор или матрицу).
Синтаксис
result=x_mdialog(title,labels,default_inputs_vector)
result=x_mdialog(title,labelsv,labelsh,default_input_matrix)
Параметры
Все параметры являются символьными переменными.
title : заголовок диалогового окна
labels, **labels**, **labels** : название вводимых переменных
default_inputs_vector и **default_input_matrix** : значения вводимых переменных по умолчанию. Их наличие обязательно.
result : символьная матрица или вектор

Пример.

```
result=x_mdialog("Эксперимент 1",["Рост[см]" "Вес[кг]" "Фамилия"], ["150"
"45" "Иванов"])
```

В результате мы получим диалоговое окно для ввода параметров. Результат выбора при нажатии кнопки диалогового окна **[Готово]** будет записан в вектор `result`.

```
result =

!173      !
!         !
!72       !
!         !
!Шваброва !
```

При нажатии кнопки **[Отмена]** получим `result =[]`.

Способ 3.
Команда **x_choose** позволяет осуществить интерактивный выбор в диалоговом окне одного из нескольких полей. Возвращает номер выбранного поля.
Синтаксис
[num]=x_choose(items,title [,button])
Параметры
items : вектор-столбец из символьных переменных, одну из которых мы хотим выбрать.
title : символьный вектор-столбец, являющийся комментарием для диалога
button : строка, Текст, появляющийся на кнопке. По умолчанию принимает значение "Cancel".
num : целое число, номер выбранного опциона или равен 0, если диалог возобновляется с помощью кнопки "Cancel"

Пример.

```
n=x_choose(['Кекс';'Пирог';'Котлета'],['Меню столовой'])
```

При выборе "Котлета" результат:

```
n =
3.
```

Замечание: Использовать команду с полем **button** мне не удалось.

Способ 4.
Редактирование матрицы в диалоговом режиме с помощью команды **x_matrix**.
Синтаксис
[result]=x_matrix(label,matrix-init)
Параметры
label : строковая переменная (комментарий)
matrix-init : матрица из действительных чисел

Замечание: В отличии от команды **x_dialog**, наличие начальных инициализируемых значений обязательно.

Пример.

```
m=x_matrix('Введите матрицу размером 3x3',rand(3,3))
```

Способ 5.
С помощью команды **x_message** можно получить диалог для ответа на вопрос в диалоговом окне с помощью выбора предлагаемых кнопок (buttons). Команда возвращает результат в виде номера выбранной кнопки.
Синтаксис
[num]=x_message(strings [,buttons])
Параметры
strings : строковый вектор, являющийся текстом сообщения
buttons : строковая переменная или вектор из двух строковых переменных. Определяет текст на кнопках. По умолчанию принимается значение "Ok".
num : номер выбранной кнопки (если определены две кнопки).

Пример.
```
r=x_message("Ты меня любишь?")
//В этом случае величина r не определена!!!
r=x_message("Тебя зовут Иннокентий?",["Нет", "Возможно"])
```
При выборе "Нет" `r=1`, в противном случае `r=2`.

Способ 6.
С помощью команды **x_message_modeless**. Очень похожа на команду **x_message** без второго параметра. Обычно применяется для текущей информации.

Tk-Tcl (Tcl / Tk) диалог

Существует интерфейс между Scilab и Tk-Tcl.
Tcl/Tk -это интерпретируемый язык программирования, снабженный библиотекой. Tcl расшифровывается как Tool Commnd Language. Этот язык полезен для написания приложений. Для тех, кто хочет с ним ознакомиться: http://www.florin.ru/iso/tcl-tk/I_gu10.htm
Объекты графического интерфейса пользователя могут быть созданы с помощью функции **uicontrol**. Базовые примитивы называются **TK_EvalFile**, **TK_EvalStr**, **TK_GetVar**, **TK_SetVar**.