Глава 2. Типы переменных

Содержание главы:

Специальные константы

Скалярные величины

Определенные в Scilab стандартные скалярные переменные начинаются со знака **%**. Часть специальных переменных предопределена. Они защищены и не могут быть удалены пользователем (но могут быть переопределены).

%i Мнимая единица: sqrt(-1)=%i =i

%pi Число Pi =3.1415927

%e Число e = 2.7182818

%eps Это условный нуль, то есть такое максимальное число, что 1+%eps=1 %eps=2.220E-16

%inf Бесконечность=Inf

%nan NotANumber: неопределено= -Nan

%s Переменная, значение которой равно "s",т.е.%s=s или s=poly(0,"s")

%t Булевское true=T

%f Булевское false=F

~%t Отрицание true=F

~%f Отрицание false=T

Замечание: Пользователь может определять свои скалярные величины и без знака **%**.

Числовые матрицы

Скалярные величины

Могут быть действительные и мнимые.
Пример мнимой величины.

```
a=5-2*%i
```

Результат:

```
a=
5.-2.i
```

Разрешены действия над мнимыми числами.

Вектора

Знаки **!** заменяют изображение круглых скобок для матриц.
Вектор-строка

```
->v=[1,-3%i,7]
v =
! 1. - 3. i 7. !
```

Замечание: v=[1+3] есть число 4, а то же самое, но с пробелом перед "+" v=[1 +3] - уже вектор-строка. Если же пробел был после знака "+", то это опять число.
Задана операция транспонирования: <вектор-строка>'
Пример.
```
b=v'
```
Результат:
```
b =
! 1. !
! 4. !
! 8. !
```
Вектор-столбец: элементы должны быть разделены с помощью точки с запятой.
```
-->v=[4;3+%i;3.33]
v =
! 4.     !
! 3. + i !
! 3.33   !
```
Аналогичная вектор-строка должна быть закончена знаком '.
```
-->v=[4;3+%i;3.33]'
v =
! 4. 3. - i 3.33 !
```
Для вектора (строки) x[n], где x[0]=xmin, x[n]=xmax, а x[i]=x[i-1]+delta, допустимо следующее задание данных: x=[xmin:delta:xmax], где xmax, xmin и delta могут быть арифметическимими выражениями.
Пример.
```
x=[0:0.1:2*%pi]
```
В Scilab определены арифметические операции над векторами.

Как создать двумерную матрицу?

Способ 1.
Прямое задание матрицы.
Пример.
```
b=[11 22 33;21 22 23;31 32 33]
```
Результат:
```
b =
! 11. 22. 33. !
! 21. 22. 23. !
! 31. 32. 33. !
```

Способ 2.
Составление матрицы из нескольких подматриц.
Пример.
```
a=[1 2;3 4]
a =
! 1. 2. !
! 3. 4. !

b=[5 5;7 8]
b =
! 5. 5. !
! 7. 8. !

c=[9 10;11 12]
c =
! 9.  10. !
! 11. 12. !
```

```
d=[a,b,c]
d =
! 1. 2. 5. 5. 9.  10. !
! 3. 4. 7. 8. 11. 12. !
```

Способ 3.
В Scilab существует тип переменных "разреженная матрица" (sparse). Разреженная матрица - это матрица, большинство элементов которой равны 0. Использование такого типа переменных для больших матриц экономит машинную память и повышает быстродействие.
Для задания матриц такого типа служит команда **sparse**. Чтобы конвертировать разреженную матрицу в привычный вид служит команда **full**.

Пример.
```
sp=sparse([1,2;4,5;3,10],[1,2,3])
sp =
( 4, 10) sparse matrix

( 1,  2) 1.
( 3, 10) 3.
( 4,  5) 2.

r=full(sp)
r =

! 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. !
! 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. !
! 0. 0. 0. 0. 0. 0. 0. 0. 0. 3. !
! 0. 0. 0. 0. 2. 0. 0. 0. 0. 0. !
```

Как создать многомерную матрицу?

Используйте команду **hypermat**.

Какие существуют функции формирования массивов "специального вида"?

В Scilab имеются следующие стандартные функции:

ones - формирование массива единиц

eye - формирование единичной матрицы

zeros - формирование массива нулей

rand - формирование массива элементов, распределенных по равномерному закону (с помощью генератора случайных чисел)

: - формирование векторов и подматриц

Подробнее смотри ниже.

Как сформировать единичную матрицу (все элементы нулевые кроме диагональных)?

С помощью команды **eye**.
Синтаксис
X=eye(m,n)
X=eye(A)
X=eye()

Параметры
A,X : матрицы
m,n :целые числа
m=<число строк>, n=<число столбцов>
Матрицы A и X могут быть неквадратными.
eye(<число строк>,<число столбцов>) служит для определения матрицы, у которой все элементы равны нулю, кроме элементов главной диагонали, значение которых равно "1". Если <число строк>=<число столбцов>, то это - единичная матрица.
Пример 1.

```
eye(2,3)
ans =
! 1. 0. 0. !
! 0. 1. 0. !
```

Пример 2.
Обращание заданной матрицы `v` произвольного размера в матрицу с главной диагональю =1 и остальными элементами, равными 0.

```
v=[2 -3.5 7]
a=eye(v)
```

Результат:

```
a =
! 1. 0. 0. !
```

Т.е. размер матрицы сохранился, а значение элементов изменилось.

Пример 3.
eye() задает матрицу с неопределенными размерами. Они будут определены, когда к этой матрице будет добавлена матрица с конкретными размерами.

```
r=eye();
q=[1 2 3;10 20 30];
w=q+r
```

Результат:

```
w =
!  2.  2.  3. !
! 10. 21. 30. !
```

Как быстро создать матрицу, все элементы которой =1?

Используйте команду **ones**. Простейший случай: **ones(<число строк>,<число столбцов>)**.

Пример.

```
-->ones(2,3)
ans =
! 1. 1. 1. !
! 1. 1. 1. !
```

Для создания вектора-строки служит **ones(1,<число столбцов>)**
Для создания вектора-столбца служит **ones(<число строк>,1)**
Для создания многомерной матрицы, все элементы которой =1, служит **ones(m1,m2,..,mn)**, где **m1, m2,..mn**- целые положительные числа.

Как быстро создать матрицу, все элементы которой =0?

Используйте команду **zeros**, аналогичную команде **ones**.
Для двумерной матрицы следует использовать конструкцию **zeros(<число строк>,<число столбцов>)**

Как создать строку, элементы которой будут последовательными целыми числам от N до M?

Нужно выполнить команду **N:M**.

Пример.
```
a=2:5
```
Результат:
```
a =
! 2. 3. 4. 5. !
```
Конструкция **a=(2:5)** даст эквивалентный результат.

Генератор случайных чисел

Команда **rand** генерирует последовательность случайных чисел.
Синтаксис
rand(m1,m2,.. [,key])
rand(x [, key])
rand()
rand(key)
rand("seed" [,n])
rand("info")
Параметры
mi : integers
key : символьная переменная (character string), принимающая значение "uniform"и "normal". Значение "uniform" соответствует равномерному распределению, а "normal" - Гауссовскому.
x : матрица. Во внимание принимается только ее размер.

Примеры использования.
1) `rand(m1,m2)` Образование случайной матрицы из m1 строк и m2 столбцов (m1 на m2).
2) `rand(m1,m2,..,mn)` Образование случайной матрицы размером m1 на m2,.. на mn.
3) `rand(A)` Образование случайной матрицы такого же размера, какого была матрица A. Матрица rand(A) комплексная, если матрица A была комплексная.
```
s=1:4; a=rand(s) a = ! .9184708 .0437334 .4818509 .2639556 !
```
4) `rand()` без аргументов дает случайное скалярное число, случайным образом изменяющееся при следующем вызове.
5)`rand("normal")` или `rand("uniform")` позволяют задать распределение случайных

чисел.
6)`rand("info")` возвращает значение переменной **key**.
7)`rand("seed" [,n])` позволяет ре-инициализировать генератор за счет изменения ядра.

Какие операции можно выполнить над числовыми матрицами?

В Scilab определены основные операции над матрицами:

A+B	Сложение
A-B	Вычитание
c*A	Умножение на число
A*B	Умножение матрицы на матрицу
A'	Транспонирование (поменять местами строки и столбцы)
matrix	Преобразование вектора или матрицы в матрицу (вектор) другого размера

Существуют и другие операции. Синтаксис таких операций смотрите в описании, но действие таких операций на первый взгляд мало понятно, и, вероятно, не стоит на них останавливаться. Возможно, есть еще какие-то нестандартные операции. Изучайте их самостоятельно.

Как найти сумму элементов матрицы?

С помощью команды **sum**.
Синтаксис
y=sum(x)
y=sum(x,'r') или **y=sum(x,1)**
y=sum(x,'c') или **y=sum(x,2)**
Параметры
x : вектор или матрица (действительная, комплексная, разреженная (sparse) или полиномиальная)
y : скаляр или вектор

`sum(x)` даст сумму всех элементов вектора или матрицы `x`, .

`y=sum(x,'r')` или, эквивалентно, `y=sum(x,1)` есть строка, равная поэлементной сумме столбцов.
(`y(j)= sum(x(i,j),i=1,m)`).

`y=sum(x,'c')` или, эквивалентно, `y=sum(x,2)` есть столбец, равный поэлементной сумме строк.
`(y(i)= sum(x(i,j),j=1,n))).`

Пример.

```
-->a=[10,20;300,400]
a =
! 10.  20.  !
! 300. 400. !
-->b=sum(a)
b =
730.
```

```
-->c=sum(a,1) //это эквивалентно c=sum(a,"r")
c =
! 310. 420. !

-->c=sum(a,2) //это эквивалентно c=sum(a,"c")
c =
! 30.  !
! 700. !
```

Как найти произведение элементов матрицы?

С помощью команды **prod**.
Синтаксис
y=(x)
y=prod(x,'r') или **y=prod(x,1)**
y=prod(x,'c') или **y=prod(x,2)**
Параметры
x : вектор или матрица (действительная, комплексная, разреженная (sparse) или полиномиальная)
y : скаляр или вектор

Пример.

```
-->a=[10,20;300,400]
a =

! 10.  20.  !
! 300. 400. !

-->help prod

-->prod(a)
ans =
24000000.
-->prod(a,1) //это эквивалентно c=prod(a,"r")
ans =
! 3000. 8000. !
-->prod(a,2) //это эквивалентно c=prod(a,"c")
ans =
! 200.    !
! 120000. !
```

Нечисловые (символьные) матрицы

Символьные матрицы - это матрицы, элементы которых имеют строковый тип (strings). Строковые элементы должны быть заключены в одинарные или в двойные кавычки (это безразлично).
Пример.

```
-->a=["2+3" 'bb";'cc' 'dd"]
a =
!2+3 bb !
!       !
!cc  dd !
```

Символьные матрицы можно складывать друг с другом. При этом происходит слияние соответствующих элементов.

Пример:
```
->a=["a","b";"c","d"];
-->b=["x1","y1";"z1","w1"];
-->a+b
ans =
!ax1 by1 !
!        !
!cz1 dw1 !
```

Какие действия можно совершать над символьными матрицами?

Над символьными матрицами в Scilab разрешены следующие операции:
Символьное сложение: команда **addf**
Символьное умножение: команда **mulf**
Решение символьных линейных систем: команда **solve**
Символьная триангуляция: команда **trianfml**
Есть и другие команды: **trisolve**, **evstr**. Синтаксис смотрите с помощью системы "Help".

Пример.
```
-->w1="a";
-->w2="b";
-->c=w1+w2 // Это слияние
c =
ab
-->s=addf(w1,w2) // Это символьное сложение
s =
a+b
```

Полиномы и полиномные матрицы (алгебраическое представление)

Как определить алгебраический полином?

Способ 1.
Используйте команду **poly**.
Синтаксис
[p]=poly(a,"x", ["flag"])
Параметры
a : матрица или действительное число
x : символьная переменная
"flag" : строковая переменная, принимающая значение "roots" или "coeff". По умолчанию принимает значение "roots".
Можно писать сокращенно: "flags"= "r" или "c".
Флаг "c" означает, что параметр **a** будет иметь смысл коэффициентов полинома.
Следует добавить, что образ мысли разработчиков крайне странен...
Я бы рекомендовала все-таки флаг "c". Что происходит с флагом "r" совсем непонятно.

Обратите внимание на следующий пример:
`x=poly(5,"y")`
Это эквивалентно `x=poly(5,"y","r")` или `x=poly(5,"y","roots")`

Результат:
```
x=
-5+y
```
`x=poly(5,"y","c")` даст просто 5. Это эквивалентно `x=poly(5,"y","coeff")`.
Рекомендуемый способ:
```
-->poly([1 2 3],"x","c")
ans =
1 + 2x + 3x^2
```
Примечание: знак степени на экране не содержит "^", и выглядит так, как будто это число в другой строке, что приводит к сильному недоумению... (Не пугайтесь)

Способ 2.
Этот способ является более наглядным.
```
x=poly(0,"x");
// Эту строку обычно забывают. Она определяет символьную переменную x="x".
1+2*x+3*x^2
```

Замечание:
Scilab может работать только с одной переменной, заданной этим методом.
Выполнение программы
```
a=poly(0,"a");
b=poly(0,"b");
c=a+b
```
дает в результате ошибку:
```
!--error 4
undefined variable : %p_a_p
```

Выражение же `d=a*7+12` дозволено...

Как получить полиномную дробь?

Создайте числитель и знаменатель дроби с помощью команды **poly** и воспользуйтесь знаком деления /
Пример.
```
-->poly([1 -1],"x","c")/poly([1 2 3],"x","c")
ans =
1 - x
------------
1 + 2x + 3x^2
```

Как производить операции в символьном виде?

Так же, как и в численном виде. Очень удобно для выполнения алгебраических задач в средней школе.
Пример.
Пусть мы хотим умножить один полином на другой, например, $(1-x)*(1+2x^2)$.
```
p1=poly([1 -1],"x","c");
p2=poly([1 0 2],"x","c");
p1*p2
```
Результат:
```
ans =
1 - x + 2x^2 - 2x^3
```

Как создать символьную матрицу, элементы которой являются полиномами?

Пример.
```
s=poly(0,"s");
p=1+2*s+s^2;
M=[p,p-1;p+1,2]
```
Результат:
```
M =
!                          !
! 1 + 2s + s^2    2s + s^2  !
!                          !
! 2 + 2s + s^2    2         !
```
Можно найти детерминант этой матрицы в символьном виде с помощью команд **det(M)**,**detr(M)** или **determ(M)**.

Булевые матрицы

Булевые (логические) матрицы - это матрицы, элементами которых являются Булевые константы.
Определение Булевых констант:
%t=T TRUE
%f=F FALSE
Синтаксис для определения Булевых матриц такой же, как и для обычных матриц.

Над Булевыми переменными определены следующие операции:

== equal	Равенство
~	Логическое отрицание
|	Логическое "или" (or)
&	Логическое "и" (and)

Пример 1.
Применение команды оператора "==" к вектор-строке.
```
-->[1,2]==[1,3]
ans =
! T F !
```

Пример 2.
Определим, какие элементы вектор-строки больше 2.
```
a=[1 2 4 5 1.5];
a>2
ans =
! F F T T F !
```
Пример 3. Сравнение Булевых матриц.
```
A=[%t,%f,%t,%f,%f,%f];
B=[%t,%f,%t,%f,%t,%f];
A | B
ans =
! T F T F T F !
A&B
```

```
ans =
! T F T F F F !
```
Если q=%t , тогда значение q будет равно T, а ~q будет иметь значение F.

Целочисленные матрицы

Какие типы целочисленных данных (integer) определены в Scilab?

В Scilab определены 6 типов данных типа integer:
32 bit signed integer (sub-type 4)
32 bit unsigned integer (sub-type 14)
16 bit signed integer (sub-type 2)
16 bit unsigned integer (sub-type 23)
8 bit signed integer (sub-type 2)
8 bit unsigned integer (sub-type 12)

Как преобразовать данные из одного типа переменных в другой?

Операторы конверсии в данные целого типа:
y=int8(X) : возвращает значения из области [-128,127]
y=uint8(X) : возвращает значения из области [0,255]
y=int16(X) : возвращает значения из области [-32768,32767]
y=uint16(X) : возвращает значения из области [0, 65535]
y=int32(X) : возвращает значения из области [-2147483648,2147483647]
y=uint32(X) : возвращает значения из области [0, 4294967295]
Параметры
X : матрица из действительных или целых чисел
y : матрица из целых чисел, закодированных из 1, 2 или 4 байт.

Пример.
```
x=[0.11 3.2 27.5 187];
int16(x)
ans =
!0 3 27 187 !
int8(x)
ans =
!0 3 27 -69!
```
Вместо 187 мы получили -69, так как максимальное значение допустимого числа в этом типе =127 (то есть <187)

Операторы конверсии в данные действительного типа:
double - конверсия из любого типа данных integer в действительное число.
Синтаксис
y=double(X)
Параметры
X : действительная или целая матрица
y : матрица из действительных чисел. Число считается частным случаем матрицы.

Тип данных "список"

Тип данных "список" состоит из набора нескольких объектов данных необязательно одинакового типа (матрицы, списки и т. д.). Удобно применять их для структурирования данных. Подобный тип переменных присутствует, например, в языке C++. Существуют следующие типы списков: "обыкновенные" (ordinary), типизированные (typed-lists) и матрично-ориентированные списки. Им соответствуют операторы **list**, **tlist** и **mlist**.

1) **"Обыкновенный" список** определяется с помощью функции **list**.
Синтаксис
list(a1,....an)
Пример списка, состоящего из 3-х участников.
```
L=list(1,"w",ones(2,2))
```
Если хотим вызывать отдельные компоненты, то это делается так:
```
->L(1) даст 1
->L(2) даст w
->L(3) даст
! 1. 1. !
! 1. 1. !
```
Списки могут быть вложенными друг в друга.
Пример вложенного списка.
```
a=list(1,"w",list(2.2,"cat","dog"))
```
Элемент a(3)(2) соответствует значению "cat".

Для переопределения этого элемента достаточно выполнить `a(3)(2)=89.3`. Заметим, что мы взяли новое значение совсем другого типа: не символьное значение, а число.

Разрешено создание пустых списков **list()**. Это список из нулевого количества элементов.

2) **"Типизированный" список** определяется с помощью функции **tlist**. Первый элемент такого списка должен быть или символьной переменной (character string) или вектором из таких переменных.
Синтаксис
tlist(typ,a1,....an)
Параметры
typ : Символьная переменная или вектор, состоящий из символьных переменных
ai : любые объекты Scilab

Пример типизированного списка.
```
L=tlist(["name";"weight";"gross"],"Sveta",[65,173])
```
Результат:
```
L=

        L(1)
!name   !
!       !
!weight !
!       !
!gross  !

        L(2)
Sveta

        L(3)
! 65. 173. !
```

При этом вместо `L(2)` мы можем вызывать `L.weight` . Результат тоже `Sveta`. `L(1)(1)` даст значение `name`.
Это удобный тип переменных для программирования. Разрешены операции умножения одного типизированного списка на другой.

3) **Матрично-ориентированные списки** в Scilab обозначаются **mlist**. Объекты **mlist** очень схожи с объектами **tlist**. Единственное отличие их состоит в том, что если M является объектом **mlist**, то для любого индекса i, который не является именем поля, M(i) не есть i-тое поле списка, а интерпретируется как i вход M, имеющий вид вектора.
Пример.

```
T=tlist(['V','name','value'],['a','b','c'],[1 2 3]);
M=mlist(['V','name','value'],['a','b','c'],[1 2 3]);
```

Величины `T(1)`, `T(2)` и `T(3)` одинаковы с `M(1)`, `M(2)` и `M(3)`. Вызов `T(3)` допустим, а `M(3)` - нет. Вместо этого следует использовать вызов `M.value`. Вызов `T.value` тоже допустим.

Как узнать тип переменной?

С помощью оператора **type**
Синтаксис
[i]=type(x)
Параметры
x : объект Scilab
i : целое

type(x) возвращает целое число, характеризующее тип переменной "**x**" соответственно нижеприведенному списку принимаемых значений:
1 : действительная или комплексная постоянная матрица
2 : полиномиальная матрица
4 : Булева матрица
5 : разреженная (sparse) матрица (смотрите ее определение с помощью **help sparse**
8 : целая матрица из 1, 2 или 4 байт
10 : матрица из символьных переменных
11 : нескомпилированнвя функция
13 : скомпилированная функция
14 : библиотека функций
15 : список (list)
16 : типизованный список (tlist)
128 : указатель (pointer)

Пример.

```
-->type(5.63)
ans =
1.
```

Как узнать тип объекта Scilab?

С помощью оператора **typeof**. Подробно смотри в описании.

N-мерные массивы

Как определить N-мерный массив?

Возможно несколько вариантов решения.
Способ 1.
Массив из n1 строк и n2 столбцов может быть определен следующим образом:

```
n1=3;n2=4;z=9;
d(n1,n2)=9
```

Результат:

```
d =
! 0. 0. 0. 0. !
! 0. 0. 0. 0. !
! 0. 0. 0. 9. !
```

Аналогично можно определить многомерный массив. Он будет весь нулевым кроме элемента с максимальными индексами по всем переменным. Зададим его равным числу, например, =77.

```
x(n1,n2,n2,n3,n4)=77
```

Для определения нулевой матрицы:

```
d(n1,n2)=0
```

Способ 2.
Определим матрицу с целыми элементами, равными последовательным целым числам, с помощью команды **hypermat**.

```
hypermat([2,3],7:12)
ans =
! 7. 9.  11. !
! 8. 10. 12. !
```

Способ 3.
Этот способ наиболее естественен.
Будем задавать матрицу с клавиатуры построчно. Число строк предварительно не задается.
Пример.

```
-->V=[11 12 13;
-->21 22 23;
-->31 32 33;
-->41 42 43]
```

Результат:

```
V =
! 11. 12. 13. !
! 21. 22. 23. !
! 31. 32. 33. !
! 41. 42. 43. !
```

Как определить размер массива?

Способ 1.
С помощью команды **size(d)**, где **d** - массив, который мы хотим определить.
Для 2-мерного массива в результате получим **[nr,nc]=size(x)**, где **nr**- число строк, а **nc**- число столбцов.

Пример.

```
d(3,4)=9
d =
! 0. 0. 0. 0. !
! 0. 0. 0. 0. !
! 0. 0. 0. 9. !

size(d)
ans =
! 3. 4. !
```

Можно получить для двумерного массива число строк и как `size(s,1)` или `size(d,"r")`, а число столбцов как `size(s,2)` или `size(d,"c")`.
Общее число элементов массива можно определить с помощью оператора **lenght(<имя объекта>)**. Оно будет равно произведению всех размерностей массива. Для приведенного выше примера `lehgth(d)` будет равно `12`.

Способ 2.
С помощью оператора **hypermat**.
Синтаксис
M=hypermat(dims [,v])
Параметры
dims : вектор размеров гиперматрицы
v : вектор ввода данных гиперматрицы Значение его по умолчанию равно `zeros(prod(dims),1)`, то есть нулевой вектор.
Примеры.

```
->a=hypermat([2,3])
a =
! 0. 0. 0. !
! 0. 0. 0. !

-->q=hypermat([2,3],7:12)
q =
! 7. 9.  11. !
! 8. 10. 12. !
```

N-мерная матрица закодирована в виде списка mlist с двумя полями. Для определения размерности этой матрицы выполним:

```
-->q.dims
ans =
! 2. 3. !
```

Для определения содержания этой матрицы:

```
-->q.entries
ans =
! 7.  !
! 8.  !
! 9.  !
! 10. !
! 11. !
! 12. !
```

Чтобы определить значение конкретного элемента этой матрицы

```
-->q.entries(5)
ans =
11
```

Замечание: По-видимому, элементы нумеруются с одним индексом последовательно, "столбец за столбцом".
Пример.
Построим матрицу M из 2-х строк и 3-х столбцов и зададим ее элементы:

```
-->M=hypermat([2,3]);
-->M.entries(1)=11;
```

```
-->M.entries(2)=21;

-->M.entries(3)=12;

-->M.entries(4)=22;

-->M.entries(5)=13;

-->M.entries(6)=31;
```

В результате построена матрица:

```
M =
! 11. 12. 13. !
! 21. 22. 31. !
```

Определение (изображение) линейной системы

Команда **syslin** определяет линейную систему.
Синтаксис
[sl]=syslin(dom,A,B,C [,D [,x0]])
либо
[sl]=syslin(dom,N,D)
либо
[sl]=syslin(dom,H)
Параметры
dom : символьная переменная, принимающая значение 'c'и 'd', или [] или скаляр.
dom определяет временной домен системы и может принимать следующие значения:
dom='c' для систем непрерывных во времени,
dom='d' для дискретно-временных систем,
n для образцов систем с периодом сбора n (в секундах).
dom=[] если время домена не определено.

A,B,C,D : матрицы определения положения (D - необязательная матрица по умолчанию равная значению нулевой матрицы). Для неправильных систем D -полиномиальная матрица.
x0 : вектор; по умолчанию его значение равно 0.
N, **D** : полиномиальные матрицы.
H : рациональная матрица или линейное определение интервала
sl : tlist ("syslin" list)изображающее линейную систему.

Функции (макросы)

Функции (макросы)в Scilab похожи на те, что мы уже встречали в других языках программирования.
Функции могут иметь аргумент, сами являться аргументом другой функции, быть членом списка, участвовать в операциях сравнения, вызываться рекурсивно.
Функция начинается со слова **function** и заканчивается словом **endfunction**. Обычно функции определены в текстовом файле, набранном во внешнем редакторе (например, в

Windows в редактоope Wordpad или в "блокноте") и загружаются в Scilab с помощью команды **exec("filename")**. Можно создавать функции и внутри Scilab. Вместо двойных кавычек можно писать одинарные. То же самое можно выполнить с помощью меню **File operation** (**Load**, **getf**, **Exec**). В дальнейшем будет показано как загружать функции в файл "filename" и компилировать их.
Первая строка функции может быть следующей:
function[y1,...,yn]=my_name(x1,...,xk),
где **yi** - выходные переменные и **xi** - входные переменные. Подробно об использовании макросов смотри в главе "Программирование". Там же есть примеры их применения.

Как создать функцию средствами пакета Scilab?

С помощью команды **deff**.
Подробно смотри в главе 3 раздел "Загрузка функций".

Пример 1.
Создадим функцию с именем **fun** двух аргументов **t** и **y**, результатом которой будет трехмерный вектор, первый элемент которого равен **t+y**, второй элемент равен **t-y**, а третий элемент равен **t*y**.

```
deff('[w]=fun(t,y)',[
'w(1)=t+y;';
'w(2)= t-y;';
'w(3)= t*y;'])
//Вызовем эту функцию
q=fun(5,7);
```

Результат:
```
q =

! 12. !
! - 2. !
! 35. !
```

Пример 2.
Создадим функцию, вычисляющую координаты сферы радиуса **r** и посторим ее.

```
x=(0:0.5:10);
y=(0:0.5:10);
r=10;
//уравнение для сферы x^2+y^2+z^2=r^2 Вычислим z=sqrt(r^2-x^2-y^2);
deff('[z]=surf(x,y)','z=sqrt(r^2-x^2-y^2)');
fplot3d1(x,y,surf);
```

Замечание: Сложные функции лучше создавать на языках Fortran или C, а затем линковать вместе с пакетом Scilab. (Смотри главу 6.)

Библиотеки

Библиотеки - это коллекции стандартных функций, которые могут либо автоматически загружаться в Scilab, либо вызываться и загружаться пользователем. Библиотеки создаются командой **lib**. Примеры библиотек даны в директории /macros/. Файлы с

расширением .sci написаны в ASCII кодах и содержат текст функций. Файлы с расширением **.bin** являются уже откомпилированными функциями. Откомпилированные функции библиотеки автоматически вызываются в Scilab при первом вызове. Чтобы построить собственную библиотеку следует использовать команду **genlib**.

В стандартный пакет Scilab входят библиотеки часто использующихся прикладных функций:

- Библиотека Control (Classical, LQG, H-infinity,...).
- Линейная алгебра
- Пакет оптимизации LMI (решение линейных матричных неравенств) оптимизации.
- Библиотека обработки сигналов (Signal processing).
- Библиотека моделирования (решение дифференциальных уравнений и др.).
- Библиотека оптимизации (дифференцируемость и недифференцируемость, решение LQ).
- Интерактивная библиотека Scicos моделирования динамических систем.
- Библиотека Metanet (анализ и оптимизация сетей).

Объекты

В Scilab определены следующие объекты:
"constant" если объект действительная или комплексная матрица
"polynomial" если объект действительная или полиномиальная матрица
"function" если объект функция
"string" если объект матрица из символьных переменных
"boolean" если объект булевая матрица
"list" если объект список
"rational" если объект рациональная матрица (transfer matrix)
"state-space" если объект "state-space" модель (смотри syslin)
"sparse" если объект разреженная(sparse) матрица
"boolean sparse" если объект разреженная (sparse) булевская матрица
Эти типы взяты из help для операции typeof моей версии scilab, в файле intro.pdf приводятся объекты с небольшим разночтением.
Обо всех объектах, определенных в Scilab, можно узнать, посмотрев описание функции **typeof**, которая служит для определения типа интересующего нас объекта.

Операции над матрицами

В Scilab разрешены следующие базовые операции над матрицами:

[ ] определение матрицы, "сцепление" (concatenation)

; разделитель строк

( ) расширение m=a(k)

( ) вставка a(k)=m

' транспонирование

+ сложение

- вычитание
* умножение
\ левое деление
/ правое деление
^ возведение в степень
.* поэлементный способ умножения
.\ поэлементный способ левого деления
./ поэлементный способ правого деления
.^ поэлементный способ возведения в степень
.*. тензорное умножение (кронекерное умножение)
./. кронекерное правое деление
.\. кронекерное левое деление

Что это за операции, будет показано ниже. Операции, включающие в себя слово "кронекерное", названы так в честь польского математика Леопольда Кронекера. Аналоги таких операций есть в известном коммерческом продукте MATLAB.

Индексирование

Для полного обзора возможностей индексирования посмотрите Help по операциям **extraction** и **insertion**.

Как осуществляется индексирование элементов в матрицах?

Индексирование матрицы может осуществляться выбором строк и столбцов, или, используя булевые индексы, или с помощью символа **$**.
Пример 1.

```
->A=[1 2 3;11 22 33]
Результат
A =
! 1.  2.  3.  !
! 11. 22. 33. !
```

Пример 2.
Пусть мы хотим присвоить величине m значение одного из элементов матрицы A.

```
-1->m=A(2)
```

Здесь индексирование проводилось в одну линию: по первому столбцу сверху-вниз, а затем по- второму столбцу сверху-вниз и так далее. Такая нумерация не совсем удобна.

```
m =
11.
```

Более понятен способ m=A(,)

```
m=A(1,2) // даст для нашей матрицы 2
->A([2 1],2)
```

даст

```
ans =
! 22. !
! 2.  !
```

Что это значит, не знаю...
```
-->A(:,3) //дает содержание всего третьего столбца
ans =
! 3.  !
! 33. !
->A(:) //дает содержание всей матрицы в виде вектор-столбца.
```
Возможны и более экзотические непонятные комбинации:
```
A(:,3:-1:1)
A([%t %f %f %t])
A(1:2,$-1)
A($:-1:1,2)
A($)
```
Изучайте это сами, если хотите (и если думаете, что это может Вам в жизни пригодиться)...

Пример.
Заполним матрицу одинаковыми элементами.
Шаг 1
```
-->x="fox"
```
Сейчас **x** - строковая переменная, равная "fox". А теперь создадим матрицу, все элементы которой будут "fox".
Шаг 2
```
-->f=x([1 1;1 1;1 1])
```
Результат:
```
f =

!fox fox !
!        !
!fox fox !
!        !
!fox fox !
```
Замечание: Существуют много богатых возможностей для создания и индексирования матриц. На первый взгляд некоторые из них малопонятны и неясно, где применимы, это будет ясно в дальнейшем.

Точность вычислений и определение формата вывода числового результата

На первый взгляд кажется, что пакет Scilab имеет недостаточную высокую точность вычисления, а именно 8 значащих цифр.

Пример.
```
a=12345.6789012345
```
Результат:
```
a =
12345.679
```

Но это не так: 8 значащих цифр - это формат для вывода числа на экран по умолчанию. На самом деле Scilab "знает" число гораздо точнее. Для того, чтобы контролировать количество выводимых разрядов числа на печать можно применить, например, команду **printf** с заданным форматом. Формат вывода задается по тем же правилам, что и для языка С. Например, формату **f** соответствует синтаксическая формула
f double; [-]m.dddddd, d's = precision (default 6)

Пример.
```
r=0.1234567890123456789
```
Результат:
```
r =
.1234568

printf("%1.12f",r);
```

Результат:
```
0.123456789012
```

Замечание: Применение формата **%f** без указания формата выводит 6 знаков после запятой.
Пример.
```
x=12345678.12345
```
Результат:
```
x =
12345678.

printf("%f",x);
```
Результат:
```
12345678.123450
```

Чему равна величина, неотличимая по малости от нуля в пакете Scilab мне совсем неясно, но для действительных чисел она, по-видимому, никак не больше, чем **e-16** (а возможно и меньше)!

Последнее обновление 17.12.2004 WebMaster