Laboratory 1

Introduction to Scilab

1.1 Introduction

SCILAB is a high-performance interactive software package for scientific and engineering computation. SCILAB integrates numerical analysis, matrix computation, signal processing and graphics in an easy-to-use environment where problems and solutions are expressed just as they are written mathematically, without traditional programming. SCILAB works essentially with one kind of object, a rectangular numerical matrix with possibly complex elements.

1.1.1 Entering simple matrices

Matrices can be introduced in SCILAB in several different ways:

- Entered by an explicit list of elements.
- Generated by built-in statements and functions.
- Created in script files.
- Loaded from external data files.

The SCILAB language contains no dimension statements or type declarations. Storage is allocated automatically up to the amount available on any particular computer. The easiest method of entering matrices is to use an explicit list. The explicit list of elements is separated by blanks or commas, and is surrounded by brackets, [ ]. The semicolon indicates the end of a row. For example, entering the statement

```scilab
-->A=[1 2 3;4 5 6;7 8 9]
```

results in the output
The matrix $A$ is saved for later used.

### 1.1.2 Matrix elements

Matrix elements can be any SCILAB expressions; for example,

```scilab
--> x = [-1.3, sqrt(3), (1+2+3)*4/5]
```

results in

$$x = 
\begin{bmatrix}
-1.3000 & 1.7321 & 4.8000
\end{bmatrix}$$

Individual matrix elements can be referenced with indices inside parentheses, $(i,j)$. Continuing the example

```scilab
--> x(5) = abs(x(1))
```

produces

$$x = 
\begin{bmatrix}
-1.3000 & 1.7321 & 4.8000 & 0 & 1.3000
\end{bmatrix}$$

Notice that the size of $x$ is automatically increased to accommodate the new element and that the undefined intervening elements are set to zero. Big matrices can be constructed using little matrices as elements. For example, we could attach another row to our matrix $A$ with

```scilab
--> r = [10, 11, 12];
--> A = [A; r]
```

which results in

```scilab
--> A = 
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12
\end{bmatrix}
```
Little matrices can be extracted from big matrices using :, For example,

```plaintext
-->A=A(1:3,:)
```

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{bmatrix}
\]

### 1.1.3 Getting workspace information

Detailed information showing the size of each of the current variables is obtained with `whos`,

```plaintext
-->whos -type constant
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>constant</td>
<td>3 by 3</td>
<td>88</td>
</tr>
<tr>
<td>startup</td>
<td>constant</td>
<td>1 by 1</td>
<td>24</td>
</tr>
<tr>
<td>ierr</td>
<td>constant</td>
<td>1 by 1</td>
<td>24</td>
</tr>
<tr>
<td>%scicos_display_mode</td>
<td>constant</td>
<td>1 by 1</td>
<td>24</td>
</tr>
<tr>
<td>%nan</td>
<td>constant</td>
<td>1 by 1</td>
<td>24</td>
</tr>
<tr>
<td>%inf</td>
<td>constant</td>
<td>1 by 1</td>
<td>24</td>
</tr>
<tr>
<td>%eps</td>
<td>constant</td>
<td>1 by 1</td>
<td>24</td>
</tr>
<tr>
<td>%io</td>
<td>constant</td>
<td>1 by 2</td>
<td>32</td>
</tr>
<tr>
<td>%i</td>
<td>constant</td>
<td>1 by 1</td>
<td>32</td>
</tr>
</tbody>
</table>

### 1.1.4 Complex numbers and matrices

The imaginary unit j is produced by the following statement

```plaintext
-->j=sqrt(-1)
```

\[
\begin{bmatrix}
0 + 1.0000i \\
\end{bmatrix}
\]

The imaginary unit is already predefined in SCILAB in %i. One convenient way of defining a complex matrix is illustrated by the statement

```plaintext
A=[1 2;3 4]+%i*[5 6;7 8]
```

\[
\begin{bmatrix}
1.0000 + 5.0000i & 2.0000 + 6.0000i \\
3.0000 + 7.0000i & 4.0000 + 8.0000i \\
\end{bmatrix}
\]
1.1.5 Matrix operations

Matrix operations are fundamental to SCILAB; wherever possible they are indicated the way they would be in a textbook or on paper, subject only to the character set limitations of the computer.

1.1.5.1 Transpose

The special character prime ’ (apostrophe) denotes the transpose of a matrix

```
-->A=A’
```

\[
A =
\begin{bmatrix}
1.0000 - 5.0000i & 3.0000 - 7.0000i \\
2.0000 - 6.0000i & 4.0000 - 8.0000i
\end{bmatrix}
\]

Notice that not only the transposition of A but also the complex conjugation occurred. Let's recover the original matrix A

```
-->A=A’
```

\[
A =
\begin{bmatrix}
1.0000 + 5.0000i & 2.0000 + 6.0000i \\
3.0000 + 7.0000i & 4.0000 + 8.0000i
\end{bmatrix}
\]

For an unconjugated transpose, use A.’

```
-->A=A.’
```

\[
A =
\begin{bmatrix}
1.0000 + 5.0000i & 3.0000 + 7.0000i \\
2.0000 + 6.0000i & 4.0000 + 8.0000i
\end{bmatrix}
\]

1.1.5.2 Addition and subtraction

Addition and subtraction of matrices are denoted by + and -. The operations are defined whenever the matrices have the same dimensions. For example, with the above matrices, A+x is not correct because A is 2-by-2 and x is 1-by-5. However, if

```
-->A=[1 2 3;4 5 6;7 8 0]
```

\[
A =
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{bmatrix}
\]
and

\[ \rightarrow B = A' \]

\[
B = \\
\begin{array}{ccc}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 0 \\
\end{array}
\]

then

\[ \rightarrow C = A + B \]

\[
C = \\
\begin{array}{ccc}
2 & 6 & 10 \\
6 & 10 & 14 \\
10 & 14 & 0 \\
\end{array}
\]

is acceptable.

1.1.5.3 Matrix multiplication

Multiplication of matrices is denoted by \(*\). The operation is defined whenever the "inner" dimensions of the two operands are the same; that is, \(x \times y\) is permitted if the second dimension of \(x\) is the same as the first dimension of \(y\). For example,

\[ \rightarrow x = [-1 \ 0 \ 2]' \]

\[
x = \\
\begin{array}{c}
-1 \\
0 \\
2 \\
\end{array}
\]

\[ \rightarrow y = x - 1 \]

\[
y = \\
\begin{array}{c}
-2 \\
-1 \\
\end{array}
\]

6
\( \Rightarrow y \times x' \)

\[
\begin{array}{ccc}
2 & 0 & -4 \\
1 & 0 & -2 \\
-1 & 0 & 2 \\
\end{array}
\]

\( \Rightarrow x' \times y \)

\[
\begin{array}{c}
4 \\
\end{array}
\]

\( \Rightarrow A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}; \)

\( \Rightarrow b = A \times x \)

\[
b = \begin{bmatrix} 5 \\ 8 \\ -7 \end{bmatrix}
\]

Similarly, matrix division can be expressed as

\( \Rightarrow x = A \backslash b \)

\[
x = \begin{bmatrix} -1 \\ 0 \\ 2 \end{bmatrix}
\]

In the example of matrix division, we just solved the following system of linear equations

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
=
\begin{bmatrix}
5 \\
8 \\
-7
\end{bmatrix}
\]  

(1.1)

where

\[
x = \begin{bmatrix} -1 \\ 0 \\ 2 \end{bmatrix}
\]  

(1.2)

An important operator in SCILAB is : (colon). To explore some of its uses, type
Description

Colon symbol : can be used to form implicit vectors. (see also linspace, logspace)
j:k is the vector \([j, j+1, ..., k]\) (empty if \(j>k\)).
j:d:k is the vector \([j, j+d, ..., j+m*d]\)

The colon notation can also be used to pick out selected rows, columns and elements of vectors and matrices (see also extraction, insertion)
A(,:) is the vector of all the elements of A regarded as a single column.
A(:,j) is the \(j\)-th column of A
A(j:k) is \([A(j), A(j+1), ..., A(k)]\)
A(:,j:k) is \([A(:,j), A(:,j+1), ..., A(:,k)]\)
A(:)=w fills the matrix A with entries of \(w\) (taken column by column if \(w\) is a matrix).

See Also

matrix, for, linspace, logspace,

1.2 Strings

Strings are delimited by either single or double quotes ‘ ’ or “ ”. If one of these two characters is to be inserted in a string, it has to be preceded by a delimiter, which is again a single or double quote. Basic operations on strings are the concatenation (operator “ + ”) and the function \text{length}, which gives the string length. A matrix whose entries are strings can be built in SCILAB, and the two previous operations extend to string matrix arguments as do the usual row and column concatenation operators.

-->S="A string with a quote character <<"’>>"
S =

A string with a quote character <<’>>

1.3 Polynomial Matrices

Polynomial object can be defined in SCILAB. Most operations defined for constant matrices are also available for polynomial matrices. Polynomials are defined with the command \text{poly}. The polynomial can be defined by its roots
--> P = poly([1 3], 's')
P =
    2
    3 - 4s + s

or by its coefficients

--> P = poly([3 -4 1], 's', 'c')
P =
    2
    3 - 4s + s

Sometimes it is convenient to define the polynomial explicitly as

--> s = poly(0, 's'); P = s^2 - 4*s + 3
P =
    2
    3 - 4s + s

Polynomial matrices can be defined

--> A = [poly([-1 -2], 's') poly([-3 -4], 's'); ...
     poly(0, 's') s^2 - 4*s + 3]
A =
    2
    2 + 3s + s  12 + 7s + s
     s
    3 - 4s + s

The symbol \ldots \ is used to continue on the next line.

1.4 Lists

SCILAB lists are built with the following functions: \textit{list, tlist} and \textit{mlist}. These three functions are structure builders in the sense that they are used to aggregate under a unique variable name a set of objects of different types. They are implemented as an array of variable-size objects. A type corresponds to each builder function, and they are recursive types (a list element can be a list.)

- If the \textit{list} function is used, the stored objects are accessed by an index giving their position in the list.
l = list(1, ["a", "b"])

1 =

    1(1)
    1.
    1(2)

!a b !

--> l($+1) = "hello"

1 =

    1(1)
    1.
    1(2)

!a b !

    1(3)

hello

--> typeof(l)
ans =

list

--> l(2) = "toto"

1 =

    1(1)
    1.
    1(2)
Notice how the dollar sign $ is used to denote the last entry in an array.

- Objects built with the builder *tlist* have a new dynamic type and the objects stored in them can be accessed by names

```scilab
t->F.name="polar"

t=0:.1:1;

f=(t+ones(t)).*exp(%i*t);

F.magnitude=abs(f);

F.phase=atan(imag(f)./real(f))*180/%pi;

F
```

```
 name: "polar"
 magnitude: [1,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2]
 phase: [1x11 constant]

F.phase(3)
```

```
an =

11.459156
```

A linear state-space system is characterized in terms of four matrices, $A$, $B$, $C$, $D$. The SCILAB function *ssrand* defines a random system with given input, output and state sizes.

```scilab
t->S=ssrand(1,1,2)

S =
```
S(2) = A matrix =
- 0.7616491  1.4739763
  0.6755537  1.1443051

S(3) = B matrix =
  0.8529775
  0.4529708

S(4) = C matrix =
  0.7223316  1.9273332

S(5) = D matrix =
  0.

where

--> A=S.A
A =
- 0.7616491  1.4739763
  0.6755537  1.1443051

--> B=S.B
B =
  0.8529775
  0.4529708

--> C=S.C
C =
    0.7223316  1.9273332

-->D=S.D
D =
    0.

The SCILAB function *syslin* allows the definition of a given dynamical linear system

    -->P=syslin('c',A,B,C,D)
P =

    P(1)  (state-space system:)
    !lss A B C D X0 dt !
       P(2) = A matrix =
          - 0.7616491  1.4739763
             0.6755537  1.1443051
       P(3) = B matrix =
             0.8529775
             0.4529708
       P(4) = C matrix =
             0.7223316  1.9273332
       P(5) = D matrix =
             0.

The object type of *P* is

    -->typeof(P)
an =
    state-space
1.5 Functions

A function is known through its calling syntax. A SCILAB-coded function can be defined interactively using the keyword *function* and *endfunction*. For example,

```
-->function y=ceibo(x,h);y=h(x); endfunction
```
```
-->typeof(ceibo)
ans =

function
```

and the *ceibo* function can be invoked as

```
-->ceibo(%pi/2,sin)
ans =

1.
```
```
-->ceibo(%pi/2,cos)
ans =

6.123D-17
```

Notice that functions can be used as function arguments. In the previous example we can use *ceibo* for matrix extraction

```
-->v=rand(1,10)
v =

    column 1 to 5
    0.7263507  0.1985144  0.5442573  0.2320748  0.2312237
    column 6 to 10
    0.2164633  0.8833888  0.6525135  0.3076091  0.9329616
```
```
-->ceibo(3,v)
ans =

0.5442573
```
As with other variables, functions can be removed or masked by assignment. For example, if a session starts with the command \( \text{sin}=4 \), then the \( \text{sin} \) is 4 and no longer the \( \text{sin} \) primitive. However, primitives and SCILAB functions available through libraries are just hidden by this mechanism. If \( \text{sin} \) is cleared by the command \( \text{clear sin} \), then the \( \text{sin} \) primitive will return to the current environment.

\[
\text{--sin=[1 3];}\n\]

Warning : redefining function: \( \text{sin} \)

\[
\text{--sin(2)} \n\text{ans = } \n3. \n\text{--clear sin} \n\text{--sin(2)} \n\text{ans = } \n0.9092974
\]

### 1.6 Scilab Programming

A SCILAB program is a set of SCILAB commands to be executed in a specific order that can be placed in a text (flat ASCII) file and executed with SCILAB command \( \text{exec} \). Such a file is called a \textit{script} and may contain function definitions. By convention, SCILAB script file names are terminated with the suffix \textit{.sce}, but if the script contains only function definitions, then the suffix \textit{.sci} is used. In that case the command \( \text{getf} \) can be used to load the function.

#### 1.6.1 Branching

See SCILAB help manual

#### 1.6.2 Iteration

See SCILAB help manual

### 1.7 Problems

Given the RLC circuit below, obtain
A. the state equations using the current through the inductor as the first state variable, and the voltage across the capacitor as the second state variable.

B. The step and impulse response using the Scilab commands `syslin` and `csim`.

Your report should be clear, short and to the point. It is due on 2-4-2015.