Lecture 1: Introduction to Scilab

Ahmed Kebaier
kebaier@math.univ-paris13.fr

HEC, Paris

Outline

Outline

- After launching Scilab, you can test the following commands

```
-->help                    // To open Scilab's help
-->help mot-clé            // Get a description of the Keyword
-->apropos mot-clé         // Get pages related to the Keyword
-->quit                    // To quit Scilab
```

- After launching Scilab, you can test the following commands

```
-->help                    // To open Scilab's help
-->help mot-clé            // Get a description of the Keyword
-->apropos mot-clé         // Get pages related to the Keyword
-->quit                    // To quit Scilab
```

- The command-line interface `clear`, `clc` et `clf` erase respectively data in memory, the commands on the screen and the plots. These commands should be executed regularly to avoid errors and make the memory free.

- After launching Scilab, you can test the following commands

```
-->help                    // To open Scilab's help
-->help mot-clé            // Get a description of the Keyword
-->apropos mot-clé         // Get pages related to the Keyword
-->quit                    // To quit Scilab
```

- The command-line interface `clear`, `clc` et `clf` erase respectively data in memory, the commands on the screen and the plots. These commands should be executed regularly to avoid errors and make the memory free.

- Test the following command-line on the interface:

```
-->x=1
-->A=ones(3,4)
-->x+A
```

- Often a file entitled for example **test.sce**, containing the following instructions :

```
clc;clf;clear;
A=ones(3,4)
1+A
```

- Often a file entitled for example **test.sce**, containing the following instructions :

```
clc;clf;clear;
A=ones(3,4)
1+A
```

- Under Scilab, tape : $-->$ exec("test.sce")
- We can also create a function "***.sci**". For this,
  1. Open a file entitled for example **carre.sci**, containing the following instructions :

```
function d = carre(x)
d= x.*x
endfunction
```

  2. Under Scilab load and execute the file carre.sci :
     $-->$ getf("carre.sci") // if the file is in the current folder.

Function `carre` is now defined under scilab :

$-->$ x=[0,1,2,3,4]

$-->$ carre(x)

Exercise

Plot the cos function on the interval $[-\pi, \pi]$. Create a mesh of 11 points.

Exercise

Plot the cos function on the interval $[-\pi, \pi]$. Create a mesh of 11 points.

Solution: Enter the following lines in the command window

```
x=linspace(-%pi,%pi,11);
y=cos(x)./(1+x.^2);
clf()
plot(x,y,b)
```

Outline

- The easiest way to define a matrix $n \times m$ under Scilab is to write all its components using the keyboard:

$$A = [a_{1,1}, \ldots, a_{1,m}; \ldots; a_{n,1}, \ldots, a_{n,m}]$$

- The easiest way to define a matrix $n \times m$ under Scilab is to write all its components using the keyboard:

$$A = [a_{1,1}, \ldots, a_{1,m}; \ldots; a_{n,1}, \ldots, a_{n,m}]$$

- Elementary operations. Test on examples !

```
-->  A+B      //sum
-->  A*B      //product
-->  A.*B     //operations are performed component wise
-->  A^ 2     //equivalent to A*A
-->  A.^ 2    //equivalent to A.*A
-->  det(A)   //determinent of A
-->  A'       //transpose of A
-->  inv(A)   //inverse of A
```

- if A is a matrix but not a vector `diag(A,k)` extracts the diagonal number k as a column vector.
- A*B is the matrix product of the matrices A and B (or product between a scalar and a vector or matrix).
- A' performs the transposition of matrix A.
- If A is a square invertible matrix you can solve the linear system $Ax = b$ using $x = A \backslash b$
(a `P A = LU` factorization of the matrix, followed by an estimation of its condition number, and finally by solving the 2 triangular systems, are done in a transparent manner)

Exercise

- give a value to the variable n then define the $n \times n$ matrix :

$$A = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$

- Define a vector $b \in \mathbb{R}^n$ using `rand(n,1)`, compute the solution of `Ax = b`. Compute the relative residual $\|Ax - b\|/\|b\|$ using the function `norm`.
- Compute $E = \frac{1}{2}x^T Ax - b^T x$. Define another vector $y \in \mathbb{R}^n$ using `rand`, compute $F = \frac{1}{2}y^T Ay - b^T y$ and verify that $E < F$.
- Build the following matrix: $B = A = \begin{pmatrix} A & I_n \\ I_n & A \end{pmatrix}$

Solution

```
n = 7;
v = -ones(1,n-1);
A = diag(v,-1) + 2*eye(n,n) + diag(v,1)
```

Solution

```
n = 7;
v = -ones(1,n-1);
A = diag(v,-1) + 2*eye(n,n) + diag(v,1)
// another solution
// A = diag(v,-1) + diag(2*ones(1,n)) + diag(v,1)
```

Solution

```
n = 7;
v = -ones(1,n-1);
A = diag(v,-1) + 2*eye(n,n) + diag(v,1)
// another solution
// A = diag(v,-1) + diag(2*ones(1,n)) + diag(v,1)
b = rand(n,1);
x = A\ b
res = norm(A*x-b)/norm(b)
E=0.5*x'*A*x - b'*x
```

Solution

```
n = 7;
v = -ones(1,n-1);
A = diag(v,-1) + 2*eye(n,n) + diag(v,1)
// another solution
// A = diag(v,-1) + diag(2*ones(1,n)) + diag(v,1)
b = rand(n,1);
x = A\ b
res = norm(A*x-b)/norm(b)
E=0.5*x'*A*x - b'*x
y=rand(n,1);
F=0.5*y'*A*y - b'*y
```

Solution

```
n = 7;
v = -ones(1,n-1);
A = diag(v,-1) + 2*eye(n,n) + diag(v,1)
// another solution
// A = diag(v,-1) + diag(2*ones(1,n)) + diag(v,1)
b = rand(n,1);
x = A\ b
res = norm(A*x-b)/norm(b)
E=0.5*x'*A*x - b'*x
y=rand(n,1);
F=0.5*y'*A*y - b'*y
E<F
```

Solution

```
n = 7;
v = -ones(1,n-1);
A = diag(v,-1) + 2*eye(n,n) + diag(v,1)
// another solution
// A = diag(v,-1) + diag(2*ones(1,n)) + diag(v,1)
b = rand(n,1);
x = A\ b
res = norm(A*x-b)/norm(b)
E=0.5*x'*A*x - b'*x
y=rand(n,1);
F=0.5*y'*A*y - b'*y
E<F
B = [ A , eye(n,n) ;...  eye(n,n), A ]
```

Assignments and extractions

Try: `A = rand(3,4) // create a matrix`
`// change coef (2,2) of A`

Assignments and extractions

Try:

```
A = rand(3,4) // create a matrix
// change coef (2,2) of A
A(2,2) = -1
// extract coef (2,3) of A and assign it to variable
c
```

Assignments and extractions

Try:

```
A = rand(3,4) // create a matrix
// change coef (2,2) of A
A(2,2) = -1
// extract coef (2,3) of A and assign it to variable
c
c = A(2,3)
// extract row 2 of A (it is assigned to ans)
```

Assignments and extractions

Try:

```
A = rand(3,4) // create a matrix
// change coef (2,2) of A
A(2,2) = -1
// extract coef (2,3) of A and assign it to variable
c
c = A(2,3)
// extract row 2 of A (it is assigned to ans)
A(2,:)  // change row 2 of A
```

Assignments and extractions

Try: A = rand(3,4) // create a matrix

```
// change coef (2,2) of A
A(2,2) = -1
// extract coef (2,3) of A and assign it to variable
c
c = A(2,3)
// extract row 2 of A (it is assigned to ans)
A(2,:)  // change row 2 of A
A(2,:)  = ones(1,4)
// change column 3 of A
```

Assignments and extractions

Try: A = rand(3,4) // create a matrix

```
// change coef (2,2) of A
A(2,2) = -1
// extract coef (2,3) of A and assign it to variable
c
c = A(2,3)
// extract row 2 of A (it is assigned to ans)
A(2,:)  // change row 2 of A
A(2,:)  = ones(1,4)
// change column 3 of A
A(:,3) = 0 // extract submat (1,3)x(1,2) and assign
it to B
```

Assignments and extractions

```
Try: A = rand(3,4) // create a matrix
// change coef (2,2) of A
A(2,2) = -1
// extract coef (2,3) of A and assign it to variable
c
c = A(2,3)
// extract row 2 of A (it is assigned to ans)
A(2,:)  // change row 2 of A
A(2,:)  = ones(1,4)
// change column 3 of A
A(:,3) = 0 // extract submat (1,3)x(1,2) and assign
it to B
B = A([1,3],[1 2])
// change the same sub-matrix
```

Assignments and extractions

Try: A = rand(3,4) // create a matrix

```
// change coef (2,2) of A
A(2,2) = -1
// extract coef (2,3) of A and assign it to variable
c
c = A(2,3)
// extract row 2 of A (it is assigned to ans)
A(2,:)  // change row 2 of A
A(2,:)  = ones(1,4)
// change column 3 of A
A(:,3) = 0 // extract submat (1,3)x(1,2) and assign
it to B
B = A([1,3],[1 2])
// change the same sub-matrix
A([1,3],[1 2]) = [-10,-20;-30,-40]
```

Another very useful vector constructor In the command window

Try the following expressions:

```
I = 1:5
J = 1:2:6
// try also J = 1:2:7 which give the same vector
K = 10:-1:5
```

The syntax is `init val:inc:lim` and this builds a row vector with init val as the first coefficient, the others components being obtained from the previous one by adding it inc until lim is not overtaken.

Exercise 3

1. Copy-paste your previous script exercise2.sce in a new file exercise3.sce and use a small value for n (e.g. n = 5) (Rmk: we need only the part of the code which defines A and B: you can remove unuseful lines of code).
2. Continue the script by creating the following new matrices:
   1. C such that $C_{i,j} = A_{i,n+1-j}$ i.e. by reversing the column order of A;
   2. D such that $C_{i,j} = A_{2i-1,j}$ i.e. taking one row over two of matrix A;
   3. E the matrix formed by the B submatrix of rows and columns $n-2$, $n-1$, n, $n+1$, $n+2$, $n+3$.

Solution

```
n=5;
v=-ones(1,n-1);
A=diag(v,-1) + 2*eye(n,n) + diag(v,1)
B=[ A , eye(n,n) ;...  eye(n,n), A ]
```

Solution

```
n=5;
v=-ones(1,n-1);
A=diag(v,-1) + 2*eye(n,n) + diag(v,1)
B=[ A , eye(n,n) ;...  eye(n,n), A ]
C = A(:,n:-1:1)
```

Solution

```
n=5;
v=-ones(1,n-1);
A=diag(v,-1) + 2*eye(n,n) + diag(v,1)
B=[ A , eye(n,n) ;...  eye(n,n), A ]
C = A(:,n:-1:1)
D = A(1:2:n,:)
```

Solution

```
n=5;
v=-ones(1,n-1);
A=diag(v,-1) + 2*eye(n,n) + diag(v,1)
B=[ A , eye(n,n) ;...  eye(n,n), A ]
C = A(:,n:-1:1)
D = A(1:2:n,:)
E = B(n-2:n+3,n-2:n+3)
```

The component-wise algebra

Three useful operators .*, ./ and .^ :

1. x and y matrices with the same dimensions:
   - `z=x.*y` is the component-wise product, i.e. $z_{i,j} = x_{i,j} y_{i,j}$
   - `z=x./y` is the component-wise product, i.e. $z_{i,j} = x_{i,j} / y_{i,j}$
   - Useful shortcut: if s is a scalar, `z=s./y` gives $z_{i,j} = s_{i,j} / y_{i,j}$ **but** $z = 1./y$ doesn't work as expected ! (use $z = 1 \ ./y$).
2. x matrix and p scalar:
   - `z=x.^ p` is the component-wise power: $z_{i,j} = x_{i,j}^p$.
   - `z=p.^ x` is the component-wise power: $z_{i,j} = p^{x_{i,j}}$.

The plot function I

```
x = linspace(0,2*%pi,31);
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
clf(); // clear the graphic window
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
clf(); // clear the graphic window
plot(x,y1,"b-",x,y2,"r--"); // only lines
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
clf(); // clear the graphic window
plot(x,y1,"b-",x,y2,"r--"); // only lines
scf(1); // select graphic window 1 to be the default
graphic window
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
clf(); // clear the graphic window
plot(x,y1,"b-",x,y2,"r--"); // only lines
scf(1); // select graphic window 1 to be the default
graphic window
clf(); // clear the graphic window
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
clf(); // clear the graphic window
plot(x,y1,"b-",x,y2,"r--"); // only lines
scf(1); // select graphic window 1 to be the default
graphic window
clf(); // clear the graphic window
subplot(2,1,1); // split the graphic window and use
subpart 1
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
clf(); // clear the graphic window
plot(x,y1,"b-",x,y2,"r--"); // only lines
scf(1); // select graphic window 1 to be the default
graphic window
clf(); // clear the graphic window
subplot(2,1,1); // split the graphic window and use
subpart 1
plot(x,y1,"ro",x,y2,"bx"); // only symbols
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
clf(); // clear the graphic window
plot(x,y1,"b-",x,y2,"r--"); // only lines
scf(1); // select graphic window 1 to be the default
graphic window
clf(); // clear the graphic window
subplot(2,1,1); // split the graphic window and use
subpart 1
plot(x,y1,"ro",x,y2,"bx"); // only symbols
subplot(2,1,2); // split the graphic window and use
subpart 2
```

The plot function I

```
x = linspace(0,2*%pi,31);
y1 = sin(x); y2 = cos(x);
scf(0); // select graphic window 0 to be the default
graphic window
clf(); // clear the graphic window
plot(x,y1,"b-",x,y2,"r--"); // only lines
scf(1); // select graphic window 1 to be the default
graphic window
clf(); // clear the graphic window
subplot(2,1,1); // split the graphic window and use
subpart 1
plot(x,y1,"ro",x,y2,"bx"); // only symbols
subplot(2,1,2); // split the graphic window and use
subpart 2
plot(x,y1,"r--o",x,y2,"g-x"); // both lines and
symbols
```

The plot function II

1. A title with title(string title).
2. x and y labels with `xlabel(string xlabel)` and `ylabel(string ylabel)`
3. A legend for the curves with `legend(curve1 leg, curve2 leg)`

Programming tools I

Functions. In scilab a function definition takes the form:

```
function [y1,y2,...,yn] = function_name(x1,x2,..xm)
// the body of the function define the output
arguments y1,...,yn
// in function of the input arguments x1,...,xm
endfunction
```

Such a definition can be written in a script (before the part of the script which uses it) or better in another file (with a name traditionaly ending with .sci). You can write any number of functions in a file. In this case you have to **load** the file in scilab before we can use them.

Programming tools II

if tests

They permit to execute different blocks of code depending on boolean expressions:

```
if bool_expression then
// block executed when bool_expression is TRUE
.....
else
// block executed when bool_expression is FALSE
.....
end
```

Example

```
x = rand()
if x < 0.5 then
y = -1;
else
y = 1;
end
```

Programming tools III

For loop

```
for i = row_vector
// body of the loop
.....
end
```

- the number of iterations equal the number of components of the row vector
- at iteration k the loop variable i is equal to `row_vector(k)`.
- Very often the row vector is of the form first `val:inc:lim`.

Programming tools III

For loop

```
for i = row_vector
// body of the loop
.....
end
```

- the number of iterations equal the number of components of the row vector
- at iteration k the loop variable i is equal to `row_vector(k)`.
- Very often the row vector is of the form first `val:inc:lim`.

It is possible to exit prematurely a for loop using the break statement:

```
for i = 1:n
.....
if special_condition_test then, break, end
....
end
```

Programming tools IV

while loop

A while loop allows to repeat a block of code while a boolean expression is true:

```
while bool_expression
// block
....
end
```

Programming tools IV

while loop

A while loop allows to repeat a block of code while a boolean expression is true:

```
while bool_expression
// block
....
end
```

Try:

```
x = 1;
while x < 1000, x = 2*x, end
```

Programming tools IV

while loop

A while loop allows to repeat a block of code while a boolean expression is true:

```
while bool_expression
// block
....
end
```

Try:

```
x = 1;
while x < 1000, x = 2*x, end
```

It is also possible to exit prematurely a while loop with the break statement.