FF505/FY505

Computational Science

Lecture 2
More on Array and Math Functions

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Software Comparison
2. Overview (continued)
3. More on Arrays
4. Math Functions

Resume

Communication: Next week Laboratory sessions with Paolo start!

- MATLAB, numerical computing vs scientific computing, alternatives
- MATLAB Desktop
- Variables and Files
- Arrays, matrices, operations
- Control flow
- Vectorization
- Script files
- Solving systems of linear equations

Outline

Why MATLAB?

1. You should learn first to do symbolic computations on paper.
   This will always be necessary to understand.
2. Matlab is much more efficient than Maple when computations become heavy
3. Matlab is more intuitive to program
   However if you prefer to use Maple for the project that is fine.
4. Matlab is spread in the industry

Performance Comparison

Some advanced applications in mathematics and physics need to handle matrices of huge size, eg:

- discretization of (partial) differential equations
- finite element methods
- discrete laplacians.

Matrix storage

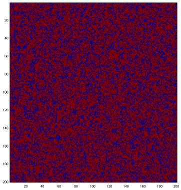
```
S = sparse((rand(5,5) < 2/3))
issparse(S)
M = full(S)
[i,j,k] = find(M);

save sparse i j k;
S=sparse(i,j,k);
```

```
M =
    1 0 1 1 1
    0 1 1 1 0
    1 1 1 1 1
    1 1 1 1 1
    0 1 0 1 1
```

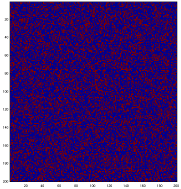
```
S =
   (1,1) 1
   (3,1) 1
   (4,1) 1
   (2,2) 1
   (3,2) 1
   (4,2) 1
   (5,2) 1
   (1,3) 1
   (2,3) 1
   (3,3) 1
   (4,3) 1
   (1,4) 1
   (2,4) 1
   ...
```

If X is an $m \times n$ matrix with nz nonzero elements then `full(X)` requires space to store $m \times n$ elements. On the other hand, `sparse(X)` requires space to store nz elements and $(nz + n + 1)$ integers.

```
S = sparse(+(rand(200,200) < 2/3));
A = full(S);
whos
Name Size Bytes Class
  A 200X200 320000 double array
  S 200X200 318432 double array (sparse)
imagesc(A) %pcolor(A)
```



```
S = sparse(+(rand(200,200) < 1/3));
A = full(S);
whos
  Name Size Bytes Class Attributes
  A 200x200 320000 double
  S 200x200 163272 double sparse
imagesc(A) %pcolor(A)
```



MATLAB and Octave

```
tic, load TestA;
load Testb; toc
tic, c=A\b; toc
```

```
>> whos
  Name Size Bytes Class Attributes
  A 1000000x1000000 51999956 double sparse
  b 1000000x1 8000000 double
  c 1000000x1 8000000 double
```

MATLAB

```
Elapsed time is 0.191414 seconds.
Elapsed time is 0.639878 seconds.
```

Octave

```
octave:1> comparison
Elapsed time is 0.276378 seconds.
Elapsed time is 0.618884 seconds.
```

Performance Comparison – MAPLE

```
> A:=ImportMatrix("TestA.mat",source=MATLAB);
memory used=72.8MB, alloc=72.9MB, time=0.51
memory used=122.8MB, alloc=122.8MB, time=0.59
memory used=192.4MB, alloc=192.2MB, time=1.57
memory used=262.6MB, alloc=224.3MB, time=2.44
memory used=300.7MB, alloc=224.3MB, time=3.24
...
memory used=1264.3MB, alloc=520.3MB, time=38.07
memory used=1325.2MB, alloc=568.3MB, time=43.73
memory used=1392.2MB, alloc=621.1MB, time=50.59
memory used=1465.8MB, alloc=679.2MB, time=58.95
memory used=1546.9MB, alloc=743.1MB, time=69.12
                                          [ 1000000 x 1000000 Matrix ]
                                A := ["A", [ Data Type: float[8] ]]
                                          [ Storage: sparse ]
                                          [ Order: Fortran_order ]
> b:=ImportMatrix("Testb.mat",source=MATLAB);
memory used=1621.2MB, alloc=743.1MB, time=76.11
                                            [ 1000000 x 1 Matrix ]
                                  b := ["b", [ Data Type: float[8] ]]
                                            [ Storage: rectangular ]
                                            [ Order: Fortran_order ]
> with(LinearAlgebra):
> c:=LinearSolve(A,b);
Error, (in simplify/table) dimensions too large
```

Performance Comparison – R

```
library(R.matlab)
library(Matrix)
S<-readMat("sparse.mat")
b<-readMat("Testb.mat")

A <- sparseMatrix(S$i,S$j,S$k)

system.time(try(solve(A,b)))

Am <- as.(A,"matrix")
# fails
# Error in asMethod(object) :
# Cholmod error 'problem too large' at file ../Core/cholmod_dense.c, line 105
```

Outline

Working with Octave

Plain text files

- uses a simple character set such as ASCII
- only non-printing characters usable to format the text, are newline and tab.

Examples: Notepad (win), TextEdit (mac), Emacs, gedit, kate (linux)

Word processor files

- contain formatted text, adding content that enables text to appear in boldface and italics, to use multiple fonts, and to be structured into columns and tables.

Examples: Word, OpenOffice, LibreOffice

Order of Operations

1. parenthesis, from innermost
2. exponentiation, from left to right
3. multiplication and division with equal precedence, from left to right
4. addition and subtraction with equal precedence, from left to right

```
>>4^2-12-8/4*2
ans =
    0
>>4^2-12-8/(4*2)
ans =
    3
>> 3*4^2 + 5
ans =
    53
>>(3*4)^2 + 5
ans =
    149
```

```
>>27^(1/3) + 32^(0.2)
ans =
    5
>>27^(1/3) + 32^0.2
ans =
    5
>>27^1/3 + 32^0.2
ans =
    11
```

Programming Style

- Document your scripts:
  - author and date of creation
  - what the script is doing
  - which input data is required
  - the function that the user has to call
  - definitions of variables used in the calculations and units of measurement for all input and all output variables!
- Organize your script as follows:
  1. input section (input data and/or input functions)
     Eg: `x=input("give me a number")`, `input("enter a key",'s')`
  2. calculation section
  3. output section (functions for displaying the output on the screen or files)
     Eg: `display(A)`, `display("text")`

Example

```
% Program M3eP32.m
% Program Falling_Speed.m: plots speed of a falling object.
% Created on March 1, 2009 by W. Palm III
%
% Input Variable:
% tfinal = final time (in seconds)
%
% Output Variables:
% t = array of times at which speed is computed (seconds)
% v = array of speeds (meters/second)
%
% Parameter Value:
g = 9.81; % Acceleration in SI units
%
% Input section:
tfinal = input('Enter the final time in seconds:');
%
% Calculation section:
dt = tfinal/500;
t = 0:dt:tfinal; % Creates an array of 501 time values.
v = g*t;
%
% Output section:
plot(t,v),xlabel('Time (seconds)'),ylabel('Speed (meters/second)')
```

Getting Help

- `help funcname`: Displays in the Command window a description of the specified function `funcname`.
- `lookfor topic`: Looks for the string topic in the first comment line (the H1 line) of the HELP text of all M-files found on MATLABPATH (including private directories), and displays the H1 line for all files in which a match occurs.
  Try: `lookfor imaginary`
- `doc funcname`: Opens the Help Browser to the reference page for the specified function `funcname`, providing a description, additional remarks, and examples.

Outline

1-D Arrays

Vectors: To create a row vector, separate the elements by semicolons. Use square brackets. For example,

```
>>p = [3,7,9]
p =
   3 7 9
```

You can create a column vector by using the transpose notation (').

```
>>p = [3,7,9]'
p =
   3
   7
   9
```

You can also create a column vector by separating the elements by semicolons. For example,

```
>>g = [3;7;9]
g =
   3
   7
   9
```

Appending vectors:

```
r = [2,4,20];
w = [9,-6,3];
u = [r,w]
u =
    2 4 20 9 -6 3
```

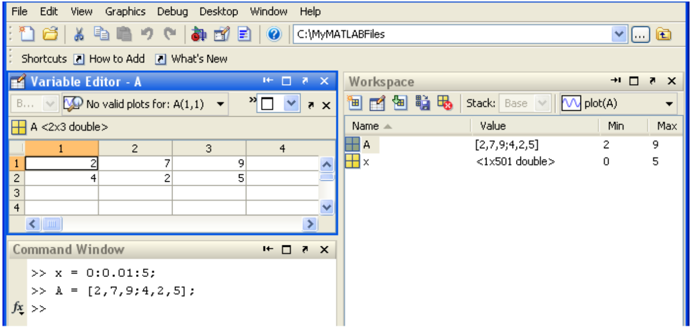
```
r = [2,4,20];
w = [9,-6,3];
u = [r;w]
u =
    2 4 20
    9 -6 3
```

2-D Arrays

Matrices: spaces or commas separate elements in different columns, whereas semicolons separate elements in different rows.

```
>> A = [2,4,10;16,3,7]
A =
     2 4 10
    16 3 7

>>c = [a b]
c =
   1 3 5 7 9 11

>>D = [a ; b]
D =
   1 3 5
   7 9 11
```

Variable Editor



Multidimensional Arrays

Consist of two-dimensional matrices layered to produce a third dimension. Each layer is called a page.

```
cat(2,A,B) % is the same as [A,B].
cat(1,A,B) % is the same as [A;B].
```

```
>> A = magic(3); B = pascal(3);
>> C = cat(4,A,B) %concatenate matrices along DIM

C(:,:,1,1) =

    8 1 6
    3 5 7
    4 9 2

C(:,:,1,2) =
    1 1 1
    1 2 3
    1 3 6
```

Array Operations

- **Addition/Subtraction**: trivial
- **Multiplication**:
  - of an array by a scalar is easily defined and easily carried out.
  - of two arrays is not so straightforward:
    MATLAB uses two definitions of multiplication:
    - array multiplication (also called element-by-element multiplication)
    - matrix multiplication
- **Division and exponentiation** MATLAB has two forms on arrays.
  - element-by-element operations
  - matrix operations

Element-by-Element Operations

Symbol	Operation	Form	Examples
+	Scalar-array addition	A + b	[6,3]+2=[8,5]
-	Scalar-array subtraction	A - b	[8,3]-5=[3,-2]
+	Array addition	A + B	[6,5]+[4,8]=[10,13]
-	Array subtraction	A - B	[6,5]-[4,8]=[2,-3]
.*	Array multiplication	A.*B	[3,5].*[4,8]=[12,40]
./	Array right division	A./B	[2,5]./[4,8]=[2/4,5/8]
.\	Array left division	A.\B	[2,5].\[4,8]=[2\4,5\8]
.^	Array exponentiation	A.^B	[3,5].^2=[3^2,5^2]
			2.^[3,5]=[2^3,2^5]
			[3,5].^[2,4]=[3^2,5^4]

Matrix-Matrix Multiplication

In the product of two matrices A * B,
the number of columns in A must equal the number of rows in B.

The product AB has the same number of rows as A and the same number of columns as B. For example

```
>>A = [6,-2;10,3;4,7];
>>B = [9,8;-5,12];
>>A*B
ans =
    64 24
    75 116
    1 116
```

Remark:
Matrix multiplication does not have the commutative property; that is, in general, $AB \neq BA$. Make a simple example to demonstrate this fact.

`cross(A,B)` cross product: eg: moment $\mathbf{M} = \mathbf{r} \times \mathbf{F}$

`dot(A,B)` dot product: eg. computes component of force $\mathbf{F}$ along direction $\mathbf{r}$

Useful Functions

```
%% misc useful functions

% max (or min)
a = [1 15 2 0.5]
val = max(a)
[val,ind] = max(a)

% find
find(a < 3)
A = magic(3) %N-by-N matrix
      constructed from the integers 1
      through N^2 with equal row, column,
       and diagonal sums.
[r,c] = find(A>=7)

% sum, prod
sum(a)
prod(a)
floor(a) % or ceil(a)
max(rand(3),rand(3))
max(A,[],1)
min(A,[],2)
A = magic(9)
sum(A,1)
sum(A,2)
sum(sum( A .* eye(9) ))
sum(sum( A .* flipud(eye(9)) ))
```

```
sum(sum( A .* flipud(eye(9)) ))


% pseudo-inverse
pinv(A) % inv(A'*A)*A'

% check empty e=[]
isempty(e)
numel(A)
size(A)
prod(size(A))
```

```
sort(4:-1:1)
sort(A) % sorts the columns
```

Useful Functions

Inner product

```
v=1:10
u=11:20
u*v' % inner or scalar product
ui=u+i
ui'
v*ui' % inner product of C^n
norm(v,2)
sqrt(v*v')
```

Eigenvalues and eigenvectors

```
A = ones(6)
trace(A)
A = A - tril(A)-triu(A,2)
eig(A)

diag(ones(3,1),-1)
[V,D]=eig(diag(1:4))

orth(A) % orthonormal basis
```

Useful Functions

Working with polynomials:

$$f(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \ldots + a_{n-1} x^2 + a_n x + a_{n+1}$$

is represented in MATLAB by the vector

$$[a_1, a_2, a_3, \ldots, a_{n-1}, a_n, a_{n+1}]$$

```
help polyfun
r=roots([1,-7,40,-34]) % x^3-7x^2+40x-34
poly(r) % returns the polynomial whose roots are r
roots(poly(1:20))
poly(A) % coefficients of the characteristic polynomial, det(lambda*EYE(SIZE(A)) - A)
```

Useful Functions

Product and division of polynomials:

$$f(x)g(x) = (9x^3 - 5x^2 + 3x + 7)(6x^2 - x + 2) =$$
$$= 54x^5 - 39x^4 + 41x^3 + 29x^2 - x + 14$$

$$\frac{f(x)}{g(x)} = \frac{9x^3 - 5x^2 + 3x + 7}{6x^2 - x + 2} = 1.5x - 0.5833$$

and a remainder of $-0.5833x + 8.1667$.

```
f = [9 -5 3 7];
g = [6 -1 2];
product = conv(f,g)
product =
   54 -39 41 29 -1 14

[quotient,remainder] = deconv(f,g)
quotient =
   1.5000 -0.5833
remainder =
       0 0 -0.5833 8.1667
```

Reshaping

```
%% reshape and replication
A = magic(3) % magic square
A = [A [0;1;2]]
reshape(A,[4 3]) % columnwise
reshape(A,[2 6])
v = [100;0;0]
A+v
A + repmat(v,[1 4])
```

Outline

Common Mathematical Functions

Exponential	
`exp(x)`	Exponential; e^x.
`sqrt(x)`	Square root; $\sqrt{x}$.
Logarithmic	
`log(x)`	Natural logarithm; $\ln x$.
`log10(x)`	Common (base-10) logarithm; $\log x = \log_{10} x$.
Complex	
`abs(x)`	Absolute value; x.
`angle(x)`	Angle of a complex number x.
`conj(x)`	Complex conjugate.
`imag(x)`	Imaginary part of a complex number x.
`real(x)`	Real part of a complex number x.
Numeric	
`ceil(x)`	Round to the nearest integer toward ∞.
`fix(x)`	Round to the nearest integer toward zero.
`floor(x)`	Round to the nearest integer toward $-\infty$.
`round(x)`	Round toward the nearest integer.
`sign(x)`	Signum function: $+1$ if $x > 0$; 0 if $x = 0$; -1 if $x < 0$.

Common Mathematical Functions

Trigonometric*	
`cos(x)`	Cosine; $\cos x$.
`cot(x)`	Cotangent; $\cot x$.
`csc(x)`	Cosecant; $\csc x$.
`sec(x)`	Secant; $\sec x$.
`sin(x)`	Sine; $\sin x$.
`tan(x)`	Tangent; $\tan x$.
Inverse trigonometric†	
`acos(x)`	Inverse cosine; $\arccos x = \cos^{-1} x$.
`acot(x)`	Inverse cotangent; $\operatorname{arccot} x = \cot^{-1} x$.
`acsc(x)`	Inverse cosecant; $\operatorname{arccsc} x = \csc^{-1} x$.
`asec(x)`	Inverse secant; $\operatorname{arcsec} x = \sec^{-1} x$.
`asin(x)`	Inverse sine; $\arcsin x = \sin^{-1} x$.
`atan(x)`	Inverse tangent; $\arctan x = \tan^{-1} x$.
`atan2(y,x)`	Four-quadrant inverse tangent.

*These functions accept x in radians.
†These functions return a value in radians.

Common Mathematical Functions

Hyperbolic	
`cosh(x)`	Hyperbolic cosine; $\cosh x = (e^x + e^{-x})/2$.
`coth(x)`	Hyperbolic cotangent; $\cosh x / \sinh x$.
`csch(x)`	Hyperbolic cosecant; $1/\sinh x$.
`sech(x)`	Hyperbolic secant; $1/\cosh x$.
`sinh(x)`	Hyperbolic sine; $\sinh x = (e^x - e^{-x})/2$.
`tanh(x)`	Hyperbolic tangent; $\sinh x/\cosh x$.
Inverse hyperbolic	
`acosh(x)`	Inverse hyperbolic cosine
`acoth(x)`	Inverse hyperbolic cotangent
`acsch(x)`	Inverse hyperbolic cosecant
`asech(x)`	Inverse hyperbolic secant
`asinh(x)`	Inverse hyperbolic sine
`atanh(x)`	Inverse hyperbolic tangent

Resume

- Review of previous lecture
- Large sparse matrices and performance comparison
- Arrays
- Mathematical Functions

Next time:

- writing your own functions (and small programs)
- advanced plotting
- random numbers generation