Principles of Technical Computing

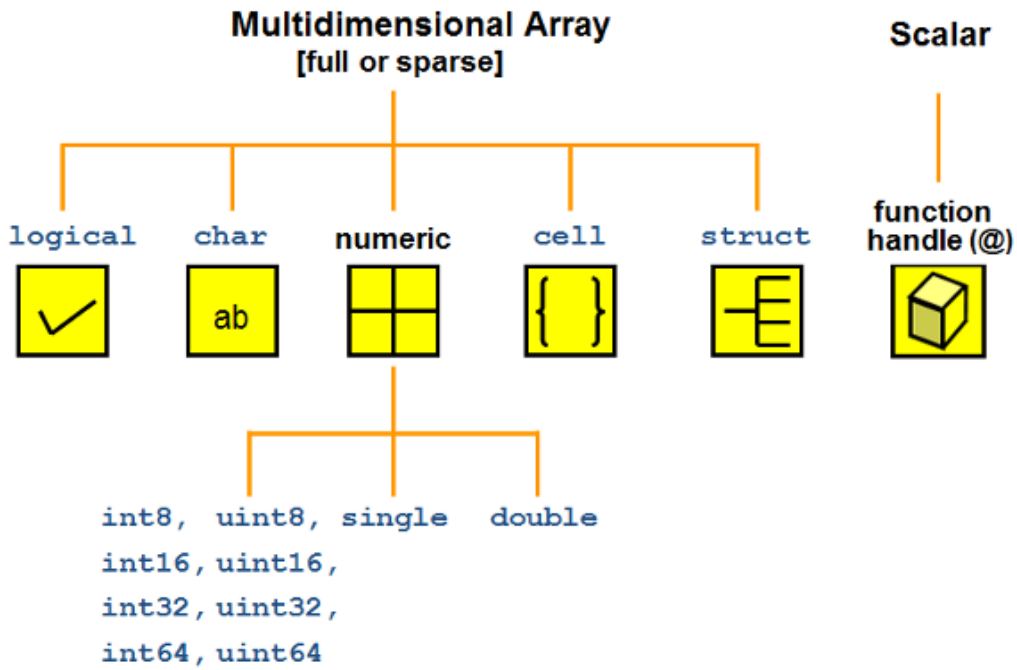
Lecture 2 – Symbolic functionality. Arrays and other data structures in Matlab

Matylda Jabłońska-Sabuka

Lappeenranta University of Technology

Overview of Week 2

- One- and multidimensional arrays
- Addressing vector/matrix elements
- Useful commands to operate on matrices
- Structures
- Cell arrays



Vectors in Matlab

Ways of defining arrays

```
yr = [1984 1986 1988 1990];
yr = [1984, 1986, 1988, 1990];
yr = 1984:2:1990;

p = [127; 130; 136; 145; 158];
p = [127 130 136 145 158]';

s = [1, 0.9, 0.8, 0.7];
s = 1:-0.1:0.7;

x = linspace(a,b,n);
```

Addressing vector elements

The address of an element in a vector is its position in the row (or column). For a vector named `ve`, `ve(k)` refers to the element in position `k`.

For example, if the vector `ve` has eight elements
`ve = [35 46 78 23 5 14 81 3]`
then
`ve(4)`=23, `ve(7)`=81 and `ve(1)`=35.

```
>> VCT=[35 46 78 23 5 14 81 3]
VCT =
     35     46     78     23     5     14     81     3
>> VCT(4)
ans =
     23
>> VCT(6)=273
VCT =
     35     46     78     23     5     273     81     3
>> VCT(2)+VCT(end)
ans =
     49
```

What is wrong in this code?

```
>> VCT=[35 46 78 23 5 14 81 3]
>> VCT(5)^VCT(9)+sqrt(VCT(6))

???  Index exceeds matrix dimensions.
```

Creating matrices in Matlab

Two-dimensional arrays (matrices) – $M_{m \times n}$

```
variable = [row1; row2; row3];
variable = [row1
            row2
            row3];

>> a=[5 35 43; 4 76 81; 21 32 40]
a =
     5 35 43
     4 76 81
    21 32 40
>>
```

Variables inserted in matrix elements can be predefined or constructed/computed while creating the matrix

```
>> cd=6; e=3; h=4;
>> Mat=[e, cd*h, cos(pi/3); h^2, sqrt(h*h/cd), 14]
Mat =
      3.0000    24.0000     0.5000
     16.0000     1.6330    14.0000
>>
```

An array can be constructed from a number of smaller arrays

```
>> A=[1:2:11; 0:5:25; linspace(10,60,6); ...
67 2 43 68 4 13]
A =
     1     3     5     7     9    11
     0     5    10    15    20    25
    10    20    30    40    50    60
    67     2    43    68     4    13
>>
```

What is wrong in this code?

```
>> A=[1 3 4 8; linspace(1,4,3); 2:2:8]

???  Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

A vector/matrix can also be initially defined as empty, and then extended by new elements.

```
V = [];
for i=1:10
      V = [V i];
end
```

Results in `V = [1 2 3 4 5 6 7 8 9 10];`

Elementary matrices

`eye(m,n)`	Returns an m-by-n matrix with 1 on the main diagonal
`eye(n)`	Returns an n-by-n square identity matrix
`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones
`diag(A)`	Extracts the diagonal of matrix A
`rand(m,n)`	Returns an m-by-n matrix of random numbers

```
>> zr=zeros(3,4)
zr =
     0     0     0     0
     0     0     0     0
     0     0     0     0
>> ne=ones(4,3)
ne =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
>>
```

```
>> idn=eye(4)
idn =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> e=eye(4,3)
e =
     1     0     0
     0     1     0
     0     0     1
     0     0     0
>>
```

Addressing matrix elements in Matlab

Addressing matrix elements

The address of an element in a matrix is its position, defined by the row number and the column number where it is located.
For example, if the matrix is

$$\texttt{ma} = \begin{bmatrix} 3 & 11 & 6 & 5 \\ 4 & 7 & 10 & 2 \\ 13 & 9 & 0 & 8 \end{bmatrix}$$

then, `ma(1,1)` equals 3 and `ma(2,3)` equals 10.

```
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
MAT =
      3     11      6      5
      4      7     10      2
     13      9      0      8
>> MAT(3,1)=20
MAT =
      3     11      6      5
      4      7     10      2
     20      9      0      8
>> MAT(3,1)-MAT(1,2)
ans =
     9
```

What is wrong in these codes?

```
>> a=4; b=6; c=a^2-b/2;
>> B=[1 3 a 8; b 4 7 d; 2:2:8]
Undefined function or variable 'd'.
```

```
>> a=4; b=6; c=a^2-b/2;
>> B=[1 3 a 8; b 4 7 1; 2:b:8]
Error using horzcat
Dimensions of arrays being concatenated are not
consistent.
```

Using a colon operator

`v(:)`	Refers to all the elements of the vector `v` (either a row or a column vector)
`v(m:n)`	Refers to elements `m` through `n` of the vector `v`
`A(:,n)`	Refers to the elements in all the rows of column `n` of the matrix `A`
`A(n,:)`	Refers to the elements in all the columns of row `n` of the matrix `A`
`A(:,m:n)`	Refers to the elements in all the rows between columns `m` and `n` of the matrix `A`
`A(m:n,:)`	Refers to the elements in all the columns between rows `m` and `n` of the matrix `A`
`A(m:n,p:q)`	Refers to the elements in all the rows `m` through `n` and columns `p` and `q` of the matrix `A`

Operations on vectors and matrices in Matlab

Operations on vectors and matrices

NOTE: operations on arrays in MATLAB follow elementary algebraic rules.

- operations `A+B`, `A-B` are allowed if and only if `A` and `B` are of the same size (have equal number of rows and columns)
- operation `A*B` is allowed if and only if number of columns in `A` equals number of rows in `B`.

Element-by-element operations

To operate on every vector/matrix element separately, use the . operator. E.g. to multiply all elements of vector `x` by 2, the following two statements are equivalent: `2*x` or `x.*2`

When willing to square each vector/matrix element, type `x.^2`.

What is wrong in this code?

```
>> A = [1 2 3; 7 2 8];
>> B = [4 5; 7 3; 6 9];
>> C = A+B
Matrix dimensions must agree.

>> A = [2 3; 6 3; 7 4];
>> B = [9 4; 8 0; 1 3];
>> C = A*B
Error using  *
Incorrect dimensions for matrix multiplication.  Check that the
number of columns in the first matrix matches the number of
rows in the second matrix.  To perform elementwise
multiplication, use '.*'.
```

Remark

Most MATLAB functions are prepared to work on multidimensional inputs. Depending on the function (mathematical operation), it works on them element by element or column by column.

For example, function `aX = abs(X)` returns an absolute value of input variable `X`.

- If `X` is a number, then `aX` is a number.
- If `X` is a vector, then `aX` is a vector.
- If `X` is a matrix, then `aX` is a matrix.

But, function `sX = sum(X)` works on arrays, so

- If `X` is a number, then `sX` is a number.
- If `X` is a vector (either row or column), then `sX` is a number.
- If `X` is a matrix, then `sX` is a row vector.

By default, `sum` (and many similar functions) work along the first dimension (sum up all the rows into one).
However, using `sum(X,n)` will sum up the array along the `n`th dimension.

Multidimentional (D>2) arrays in Matlab

Multidimensional arrays $A_{n_1 \times n_2 \times \ldots \times n_k}$

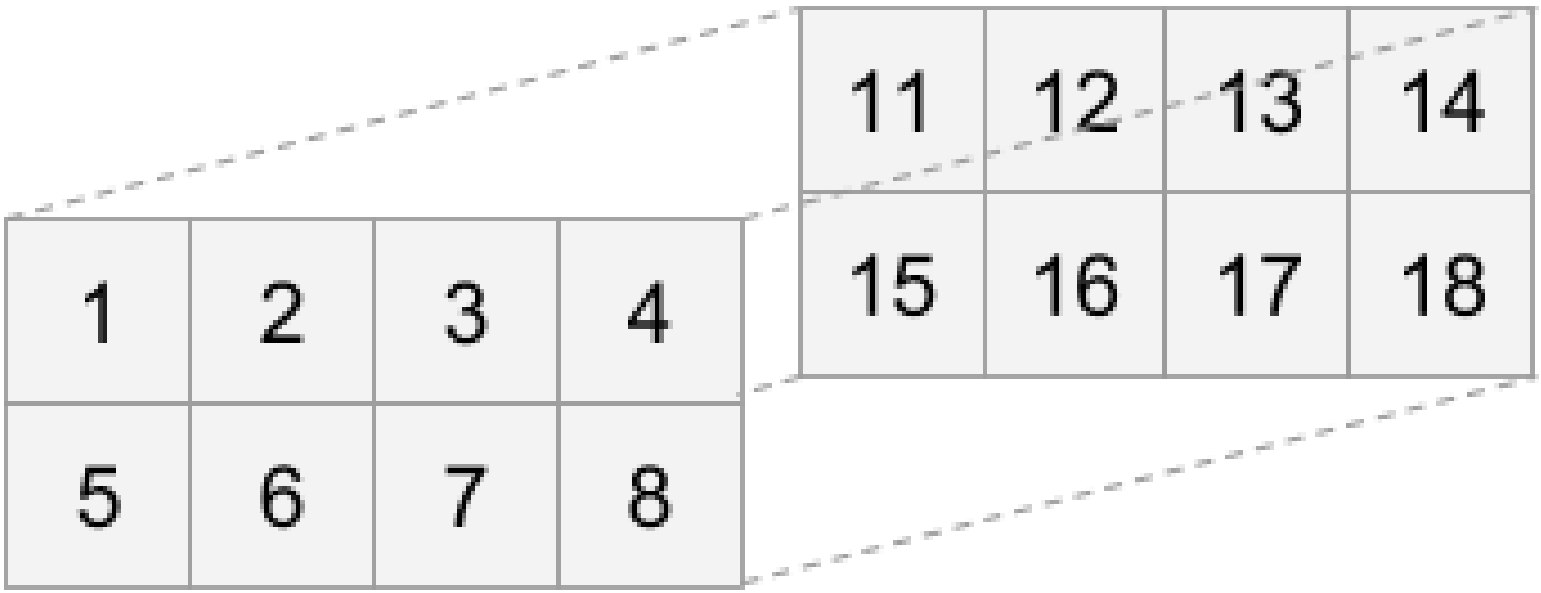
Multidimensional arrays in the MATLAB environment are arrays with more than two subscripts. One way of creating a multidimensional array is by calling `zeros`, `ones`, `rand`, or `randn` with more than two arguments. For example,

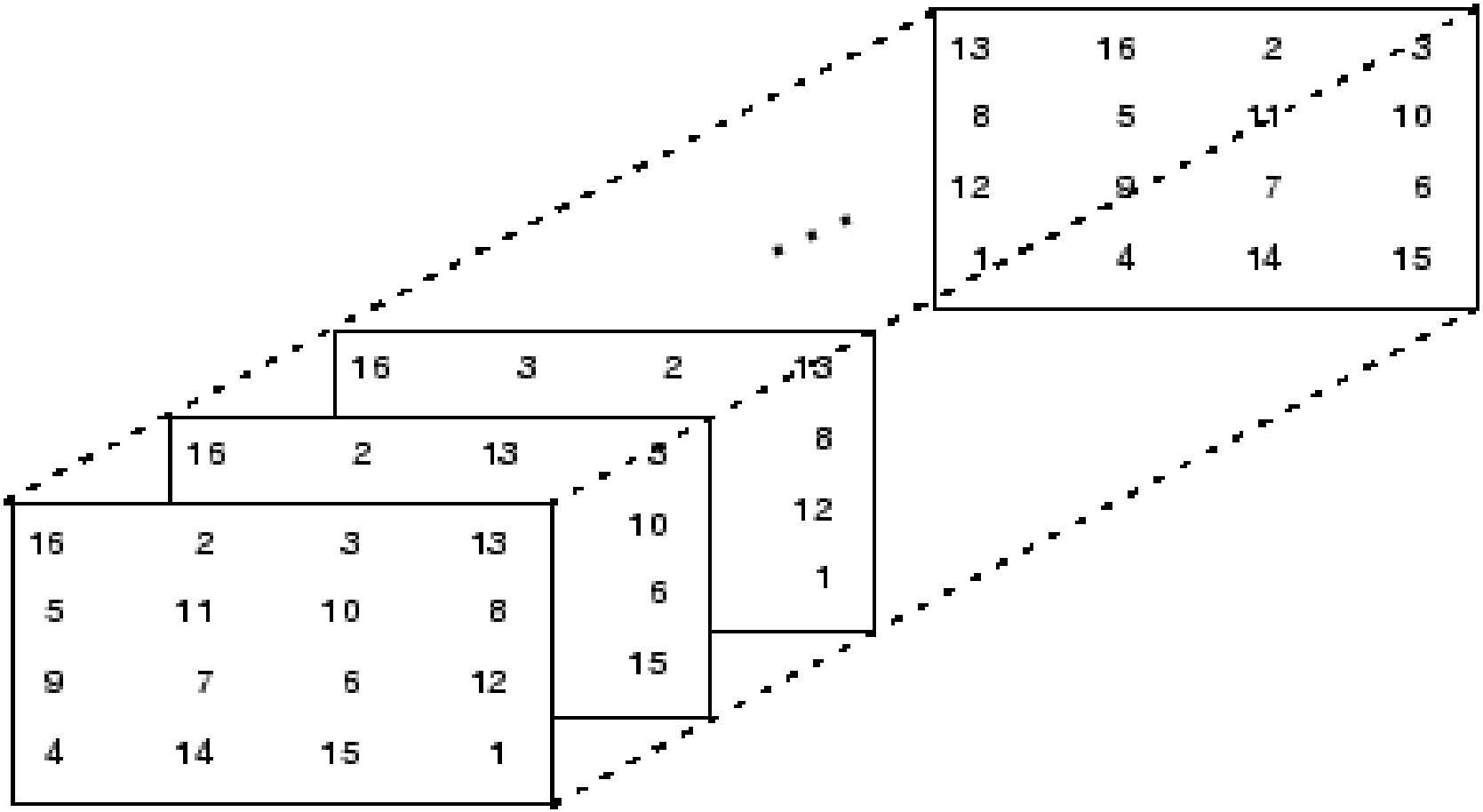
```
R = randn(3,4,5)
```

creates a 3-by-4-by-5 array with a total of 3*4*5 = 60 normally distributed random elements.

Example

```
A = 1:8; B = 11:18;
A = reshape(A,4,2)';
B = reshape(B,4,2)';
C(:,:,1) = A;
C(:,:,2) = B;
```




13 16 2 3
8 5 11 10
12 9 7 6
1 4 14 15
16 3 2 13
16 2 13 3
8
10 12
6 1
15
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1

Useful commands for matrices (arrays) in Matlab

Useful commands – `size(X)`

`>> d = size(X)`
returns the sizes of each dimension of array `X` in a vector `d`

`>> [m,n] = size(X)`
returns the size of matrix `X` in separate variables `m` and `n`.

`>> m = size(X,dim)`
returns the size of the dimension of `X` specified by scalar `dim`

The size of the second dimension of `rand(2,3,4)` is 3.

```
>> m = size(rand(2,3,4),2)
```

Here the size is output as a single vector.

```
>> d = size(rand(2,3,4))
```

Here the size of each dimension is assigned to a separate variable.

```
>> [m,n,p] = size(rand(2,3,4))
```

Useful commands – `length(X)`

`length(X)` returns the length of vector `X`.
It is equivalent to `max(size(X))` for non-empty arrays and 0 for empty ones.

The output is always a single number. For instance, for

```
>> l = size(rand(2,3,4))
```

the output is `l=4`.

Useful commands – `find(X)`. Logical indexing.

`ind = find(X)` locates all nonzero elements of array `X`,
and returns the linear indices of those elements in vector `ind`.

If `X` is a row vector, then `ind` is a row vector;
otherwise, `ind` is a column vector.

If `X` contains no nonzero elements or is an empty array,
then `ind` is an empty array.

Function `find(X)` is commonly used with logical input
(which takes values 0 or 1).

```
>> X = [1 0 4 -3 0 0 0 8 6];
>> indices = find(X)
```

returns linear indices for the nonzero entries of X.

```
indices =
     1     3     4     8     9
```

```
>> find(X > 2)
```

returns linear indices corresponding to the entries of X that are greater than 2.

```
ans =
     3     8     9
```

```
>> A= magic(4)
A =
     16      2      3      13
     5      11     10      8
     9      7      6      12
     4      14     15      1
>> [r,c]= find(A>10);
>> r', c'
ans =
     1      2      4      4      1      3
ans =
     1      2      2      3      4      4
```

Cell arrays in Matlab

Cell array

Cell arrays in MATLAB are multidimensional arrays whose elements are copies of other arrays. A cell array of empty matrices can be created with the `cell` function. But, more often, cell arrays are created by enclosing a miscellaneous collection of things in curly braces, {}. The curly braces are also used with subscripts to access the contents of various cells. For example,

```
C = {A sum(A) sum(sum(A))}
```

produces a 1-by-3 cell array. The three cells contain the magic square, the row vector of column sums, and the product of all its elements. When C is displayed, you see

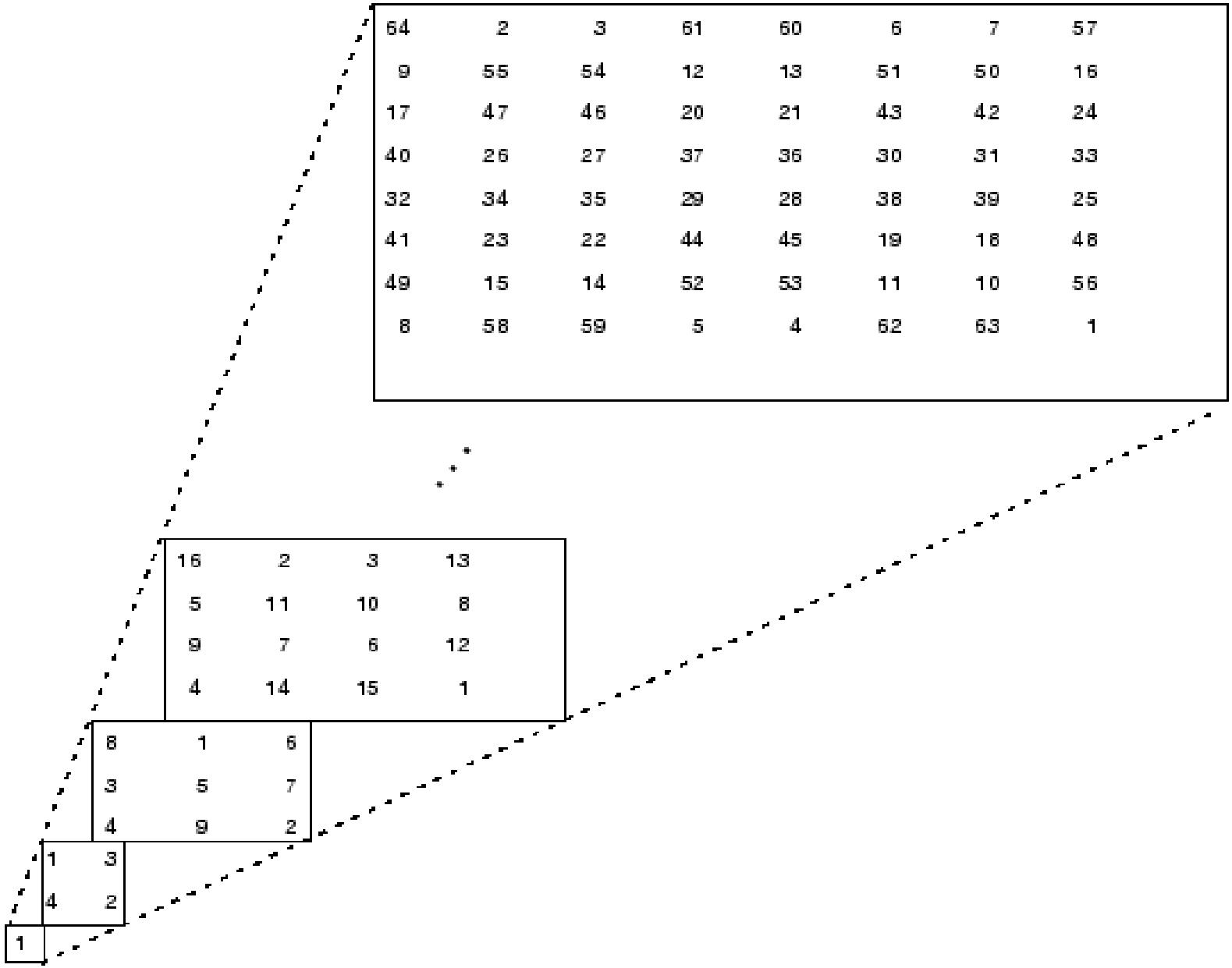
```
C =
    [4x4 double]    [1x4 double]    [136]
```

You can use three-dimensional arrays to store a sequence of matrices of the same size. Cell arrays can be used to store a sequence of matrices of different sizes. For example,

```
M = cell(8,1);
for n = 1:8
   M{n} = magic(n);
end
M
```

produces a sequence of magic squares of different order:

```
M =
   [          1]
   [ 2x2 double]
   [ 3x3 double]
   ⋮
   [ 8x8 double]
```


64 2 3 61 60 6 7 57
9 55 54 12 13 51 50 16
17 47 46 20 21 43 42 24
40 26 27 37 36 30 31 33
32 34 35 29 28 38 39 25
41 23 22 44 45 19 18 48
49 15 14 52 53 11 10 56
8 58 59 5 4 62 63 1
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
8 1 6
3 5 7
4 9 2
1 3
4 2
1

Characters and text

Enter text into MATLAB using single quotes. For example,

```
s = 'Hello'
```

Converting character array to a numeric matrix

```
a = double(s); s = char(a);
```

Concatenation with square brackets,

```
h = [s, ' world']
```

```
v = [s; ' world']
```

```
S=char('A','rolling','stone','gathers','momentum.')
```

```
C={'A';'rolling';'stone';'gathers';'momentum.'}
```

Structures in Matlab

Structures

Structures are multidimensional MATLAB arrays with elements accessed by textual field designators. For example,

```
S.name = 'Ed Plum';
S.score = 83;
S.grade = 'B+';
```

creates a scalar structure with three fields:

```
S =
    name:   'Ed Plum'
   score:   83
   grade:   'B+'
```

Like everything else in the MATLAB environment, structures are arrays, so you can insert additional elements. In this case, each element of the array is a structure with several fields. The fields can be added one at a time,

```
S(2).name = 'Toni Miller';
S(2).score = 91;
S(2).grade = 'A-';
```

or an entire element can be added with a single statement:

```
S(3) = struct('name','Jerry Garcia', ...
   'score',70,'grade','C')
```

Now the structure is large enough that only a summary is printed:

```
S =
1x3 struct array with fields:
   name
   score
   grade
```

There are several ways to reassemble the various fields into other MATLAB arrays. They are mostly based on the notation of a comma-separated list. If you type

```
S.score
```

it is the same as typing

```
S(1).score, S(2).score, S(3).score
```

which is a comma-separated list.
MATLAB can create a numeric row vector containing the score field of each element of structure array S by

```
scores = [S.score]
scores =
   83   91   70
```

To create a character array from one of the text fields (name, for example), call the char function on the comma-separated list produced by S.name:

```
names = char(S.name)
names =
   Ed Plum
   Toni Miller
   Jerry Garcia
```

Similarly, you can create a cell array from the name fields by enclosing the list-generating expression within curly braces:

```
names = {S.name}
names =
   'Ed Plum'   'Toni Miller'   'Jerry Garcia'
```

Example 1

Create a vector `x` with the elements:

$$x_n = (-1)^{n+1}/(2n-1)$$

where n are positive integers from 1 onwards. Add up the elements of the version of this vector that has 100 elements. Do it with and without a loop.

Example 2

Given the array `A = [2 7 9 7; 3 1 5 6; 8 1 2 5]` provide the command that will

(a) assign the even-indexed columns of `A` to an array called `B`,

(b) assign the odd-indexed rows to an array called `C`,

(c) convert `A` into a 4-by-3 array,

(d) compute the reciprocal (inverse) of each element of `A`,

(e) compute the square-root of each element of `A`.

Example 3

Given an array containing BMI indices of a group of ten people `[28.0 29.5 17.0 29.6 25.1 16.5 19.4 23.7 30.3 30.4]`, split it into three arrays with different ranges:

- underweight with $BMI < 18.5$,
- normal weight with $18.5 \leq BMI \leq 24.9$
- overweight with $BMI > 24.9$.

Then, using an appropriate MATLAB function, sort each of these three arrays in descending order.
Hint: You can do the splitting using a loop or logical indexing.

Example 4

The Fibonacci numbers are computed according to the following relation:

$$F_n = F_{n-1} + F_{n-2}$$

with $F_0 = F_1 = 1$. Compute the first 10 Fibonacci numbers.
Hint: Define the first two numbers as first two elements of a vector. Create a loop which calculates all other numbers from 3rd to 10th using the previous two numbers.

Example 5

Calculate the following sums without using loops:

- $1 + 2 + 3 + ... + 20$
- $15 + 25 + 35 + ... + 4005$
- $2 \cdot 4 + 4 \cdot 6 + 6 \cdot 8 + ... + 98 \cdot 100$
- $1^2 + 2^3 + 3^4 + ... + 10^{11}$
- $\frac{1}{1+2} + \frac{2}{2+3} + \frac{3}{3+4} + ... + \frac{30}{30+31}$

Example 6

We say that an n by n matrix A is symmetric when for all $i, j = 1, 2, \ldots, n$

$$A(i, j) = A(j, i).$$

Write a script using loops which will check any matrix whether it is symmetric or not and will display the result in the Command Window.

Example 7

Mary has savings of \$78.40. She is also getting regularly pocket money of \$5 per week. However, one day she notices a new weekly magazine related to her hobby, which costs \$7.29, and decides to start buying it every week.

(a) How long will it take for Mary until she will not afford another issue of the magazine?

(b) How much money will she have in her piggy bank after buying the last issue?

For each case, store the values of all the weeks in a vector.

Symbolic functionality in Matlab (additional material)

Symbolic functionality of Matlab

Symbolic Math Toolbox software lets you to perform symbolic computations within the MATLAB numeric environment. It provides tools for solving and manipulating symbolic math expressions and performing variable-precision arithmetic for a broad range of mathematical tasks such as:

- Differentiation
- Integration
- Linear algebraic operations
- Simplification
- Transforms
- Variable-precision arithmetic
- Equation solving

To declare variables `x` and `y` as symbolic objects use the `syms` command:

```
syms x y
```

You can manipulate the symbolic objects according to the usual rules of mathematics. For example:

```
>> x + x + y
ans =
    2*x + y
```

You also can create formal symbolic mathematical expressions and symbolic matrices.

Symbolic Math Toolbox software also enables you to convert numbers to symbolic objects. To create a symbolic number, use the sym command:

```
>> a = sym('2')
```

If you create a symbolic number with 15 or fewer decimal digits, you can skip the quotes:

```
>> a = sym(2)
```

You can use `sym` or `syms` to create symbolic variables. The `syms` command:

- Does not use parentheses and quotation marks: `syms x`
- Can create multiple objects with one call
- Serves best for creating individual single and multiple symbolic variables

The `sym` command:

- Requires parentheses and quotation marks: `x = sym('x')`. When creating a symbolic number with 15 or fewer decimal digits, you can skip the quotation marks: `f = sym(5)`.
- Creates one symbolic object with each call.
- Serves best for creating symbolic numbers and symbolic expressions.
- Serves best for creating symbolic objects in functions and scripts.

Suppose you want to use a symbolic variable to represent the golden ratio
$\phi = \frac{1+\sqrt{5}}{2}$

The command

```
>> phi = sym('(1 + sqrt(5))/2');
```

achieves this goal.

Now you can perform various mathematical operations on `phi`. For example,

```
>> f = phi^2 - phi - 1
```

returns

```
f =
     (5^(1/2)/2 + 1/2)^2 - 5^(1/2)/2 - 3/2
```

Now suppose you want to study the quadratic function $f = ax^2 + bx + c$.

One approach is to enter the command

```
>> f = sym('a*x^2 + b*x + c');
```

which assigns the symbolic expression $ax^2 + bx + c$ to the variable `f`.

However, in this case, Symbolic Math Toolbox software does not create variables corresponding to the terms of the expression: `a`, `b`, `c`, and `x`.

To perform symbolic math operations on f, you need to create the variables explicitly. A better alternative is to enter the commands

```
>> a = sym('a');
>> b = sym('b');
>> c = sym('c');
>> x = sym('x');
```

or simply

```
>> syms a b c x
```

Then, enter

```
>> f = a*x^2 + b*x + c;
```

Symbolic computations

To show the order of a polynomial or symbolically differentiate or integrate a polynomial, use the standard polynomial form with all the parentheses multiplied out and all the similar terms summed up. To rewrite a polynomial in the standard form, use the expand function:

```
>> syms x
f =
     (x^2- 1)*(x^4 + x^3 + x^2 + x + 1)*(x^4 - x^3 +
     x^2 - x + 1);

>> expand(f)
ans =
     x^10 - 1
```

The `factor` simplification function shows the polynomial roots. If a polynomial cannot be factored over the rational numbers, the output of the `factor` function is the standard polynomial form. For example, to factor the third-order polynomial, enter:

```
>> syms x
>> g = x^3 + 6*x^2 + 11*x + 6;
>> factor(g)

ans =
     (x + 3)*(x + 2)*(x + 1)
```

Substitutions in symbolic expressions

You can substitute a symbolic variable with a numeric value by using the `subs` function. For example, evaluate the symbolic expression `f` at the point `x = 1/3`:

```
>> syms x
>> f = 2*x^2 - 3*x + 1;
>> subs(f, 1/3)
ans =
     2/9
```

The `subs` function does not change the original expression `f`:

```
>> f
f =
     2*x^2 - 3*x + 1
```

When your expression contains more than one variable, you can specify the variable for which you want to make the substitution. For example, to substitute the value `x = 3` in the symbolic expression

```
>> syms x y
>> f = x^2*y + 5*x*sqrt(y);
```

enter the command

```
>> subs(f, x, 3)
ans =
     9*y + 15*y^(1/2)
```

Differentiation in symbolic expressions

To differentiate a symbolic expression, use the `diff` command. The following example illustrates how to take a first derivative of a symbolic expression:

```
>> syms x
>> f = sin(x)^2;
>> diff(f)

ans =
    2*cos(x)*sin(x)
```

For multivariable expressions, you can specify the differentiation variable. If you do not specify any variable, MATLAB chooses a default variable by its proximity to the letter `x`:

```
>> syms x y
>> f = sin(x)^2 + cos(y)^2;
>> diff(f)

ans =
    2*cos(x)*sin(x)
```

To differentiate the symbolic expression `f` with respect to a variable `y`, enter:

```
>> diff(f, y)

ans =
    -2*cos(y)*sin(y)
```

Some other available operations

- Integration: `int`
- Solving equations: `solve`
- Plotting of symbolic functions: `ezplot`, `ezplot3`, etc.
- Function limits: `lim`
- Many more... Check Symbolic Toolbox.

Matrix of symbolic variables

```
>> syms a b c
>> A = [a b c; c a b; b c a]

A =
     [ a, b, c]
     [ c, a, b]
     [ b, c, a]
```

Find the sum of all the elements of the first row:

```
>> sum(A(1,:))
ans =
     a + b + c
```

Create the 2-by-4 matrix `A` with the elements `A1_1, ..., A2_4`:

```
>> A = sym('A', [2 4])
A =
     [ A1_1, A1_2, A1_3, A1_4]
     [ A2_1, A2_2, A2_3, A2_4]
```

To control the format of the generated names of matrix elements, use %d in the first argument:

```
>> A = sym('A%d%d', [2 4])
A =
     [ A11, A12, A13, A14]
     [ A21, A22, A23, A24]
```