Informatics 1 CG
Introduction to Matlab

Carina Silberer

Lab 1 – Week 3

January 22, 2014

Matlab

- a high-level language
- interactive environment (you can program on the fly)
- for numerical computation, visualization, programming
- command-line interface and GUI
- commercially developed by MathWorks
- open source variant: GNU octave
  highly compatible with Matlab

Getting started

Starting Matlab

From Dice terminal - opens GUI

```
[<name>]<studentID>: matlab
```

or for command-line interface

```
[<name>]<studentID>: matlab -nodisplay
```

Exiting Matlab

```
>> exit
```

Arithmetic Operations and Functions

Operations

$8 + 10$
$8 - 10$
$5 * 14$
$7/8$
7^3
$3.8 * 10^3$
$3.8 * 10^-3$

Built-in in matlab

```
>> 8+10
>> 8-10
>> 5*14
>> 7/8
>> 7^3
>> 3.8e3
>> 3.8e-3
```

Some built-in functions in matlab

```
>> sin(x)
>> exp(x)
```

To list all built-in elementary functions

```
>> help elfun
```

Data Types (1)

Numeric Types – Integer and floating-point numbers

```
>> 344
>> 3.44
```

Booleans (logicals): 0 or 1 (false or true)

Characters and Strings

```
>> 'this is a string'
```

Matrices and Vectors (▸ More later)

Use variables to store values.

```
>> a = 300/15-15
>> b = a*2.5
>> s = 'Hello, I am another string.'
```

Logical Operators

not
or
and

```
>> ~1
>> 0 | 1
>> 0 & 1
```

equal to
not equal to
less than
greater than or equal to

```
>> 3 == 4
>> a ~= b
>> a < b
>> a >= b
```

IF Statements

```
if CONDITION
    EXP
end
```

```
>> a = 3;
>> if a>2
disp('Decreasing a by 1.')
a = a-1;
end
```

```
if CONDITION
    EXP
else
    EXP
end
```

```
>> if a>2
disp('Decreasing a by 1.')
a = a-1;
else
disp('a is smaller than or equal to 2.')
end
```

Loops

FOR-loop

for INDEX=START:END
    Execute this block
    for a fixed number
    of times.
end

```
>> a = 0;
>> for i=1:10
disp('Increasing a by 1.')
a = a+1;
end
```

WHILE-loop

while CONDITION
    Execute this block
    as long as condition
    is met.
end

```
>> a = 0;
>> while a<10
disp('Increasing a by 1.')
a = a+1;
end
```

Use `break` statement to leave loop mid-way;

Useful Commands

Use ; to suppress answer on display.

```
>> a = 300/15;
```

Use `disp()` to display content.

```
>> disp(a)
    20
```

Ask for value of variable.

```
>> a
```

Use `clear` to remove variable from memory.

```
>> clear a
```

Clear complete memory.

```
>> clear all
```

Use % for comments.

```
>> c = (10-13)*5 % should be -15
```

Data Types (2) – Matrices

Vectors

$$V1 = \begin{pmatrix} 1 & 2.1 & -3 \end{pmatrix}$$

$$V2 = \begin{pmatrix} 1 \\ 2.1 \\ -3 \end{pmatrix}$$

```
>> V1 = [1,2.1,-3]
>> V2 = [1;2.1;-3]
```

Vectors are one-dimensional matrices.

Matrices

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 1 \\ 3 & 1 & 2 & 2 \end{pmatrix}$$

```
>> M = [1,2,3,4; 2,3,1,1; 3,1,2,2]
>> M = [1 2 3 4; 2 3 1 1; 3 1 2 2]
```

Commas separating column elements can be omitted.

Accessing Matrix Elements (1)

```
>> M = [1 8 6; 2 4 7; 5 9 3];
```

Use indexing to access matrices, where index starts from 1.

$\mathbf{M}_{i,j}$, i.e. element in row i, column j

```
>> M(2,3) % row 2, column 3
7
>> M(3) % row 3, column 1
5
```

Use `end` keyword to indicate last index.

```
>> M(3,end) % row 3, last column
3
>> M(end-1,2) % row 2, column 2
4
```

Accessing Matrix Elements (2)

```
>> M = [1 8 6; 2 4 7; 5 9 3];
```

Use colon : to select columns or rows.

$\mathbf{M}_{i,:}$ complete row i
$\mathbf{M}_{:,j}$ complete column j

```
>> M(2,:) % row 2
ans =
     2     4     7

>> M(:,3) % column 3
ans =
     6
     7
     3
```

$\mathbf{M}_{s:e,j}$ rows s to e of column j
$\mathbf{M}_{i,c:d}$ columns c to d of row i

```
>> M(2:3,:)
ans =
     2     4     7
     5     9     3

>> S = M(2:end,1:2)
S =
     2     4
     5     9
```

Side note on Colon

With colon a new vector can be created ranging from a:c

```
>> V = 10:15 % a = 10, c = 15
V =
    10    11    12    13    14    15
```

With a:b:c an increment is specified.

```
>> W = 10:1.5:15 % b = 1.5
W =
   10.0000   11.5000   13.0000   14.5000
```

Modifying Matrices (1)

```
>> M = [1 8 6; 2 4 7; 5 9 3];
```

Replace items directly.

$\mathbf{M}_{row,column} = new\_value$
$\mathbf{M}_{s:e,c:d} = new\_values$

```
>> M(2,3) = 22;
>> M(1:2,2:3) = [8 9; 7 6];
```

Add or remove a column or row.

$\mathbf{M}_{:,end+1} = new\_column$
$\mathbf{M}_{end+1,:} = new\_row$
$\mathbf{M}_{:,j} = []$
$\mathbf{M}_{i,:} = []$

```
>> M(:,end+1) = [4; 5; 4];
>> M(2,:) = [];
>> disp(M)
     1     8     9     4
     5     9     3     4
```

Modifying Matrices (2)

```
>> M = [1 8; 2 7]; V1 = [5; 9]; V2 = [13 0 0]; V3 = [0 1 2];
```

Concatenate matrices (or vectors).

[,] Concatenate horizontally

```
>> M2 = [M, V1]; disp(M2)
     1     8     5
     2     7     9
```

[;] Concatenate vertically

```
>> M3 = [M2; V2; V3]; disp(M3)
     1     8     5
     2     7     9
    13     0     0
     0     1     2
```

Mathematical Operations with Matrices (1)

```
>> A = [4 5; 6 7]; B = [3 8; 9 1]; C = [2,3,4;5,5,1];
```

Sum and difference works just as with scalars.

$\mathbf{A} + \mathbf{B}$
$\mathbf{A} - \mathbf{B}$

```
>> disp(A - B)
     1    -3
    -3     6
```

Beware that matrices need to be of same dimensions.

```
>> disp(A - C)
??? Error using ==> plus
Matrix dimensions must agree.

>> disp(size(A))
     2     2

>> disp(size(C))
     2     3
```

Mathematical Operations with Matrices (2)

```
>> A = [4 5; 6 7]; B = [3 8; 9 1]; C = [2,3,4;5,5,1];
```

Product

$\mathbf{A} \cdot \mathbf{B}$	Matrix product
$\mathbf{A} \cdot b$	Product with scalar
$\mathbf{A}^b$	Matrix power

```
>> disp(A*B)
    57    37
    81    55
>> disp(A*C)
    33    37    21
    47    53    31
>> P = B^3;
>> S = B*1.3;
```

Element-wise operations

$(\mathbf{A_{ij}} \times \mathbf{B_{ij}})_{i=1,\ldots,n;j=1,\ldots,m}$
$(\mathbf{A_{ij}}^b)_{i=1,\ldots,n;j=1,\ldots,m}$
$(\mathbf{A_{ij}} \div \mathbf{B_{ij}})_{i=1,\ldots,n;j=1,\ldots,m}$

```
>> disp(A.*B)
    12    40
    54     7
>> disp(A.*C)
??? Error using ==> times
Matrix dimensions must agree.
>> P2 = A.^3;
>> D = A./B;
```

Useful Matrix Functions (1)

```
>> M = [2,3,4;5,5,1];
```

Size of M.

```
>> size(M);
```

Transpose of M.

```
>> disp(M') % == transpose(M);
     2     5
     3     5
     4     1
```

$\sum_{i=1}^{n} M_{i,:}$ Sum of rows.
$\sum_{j=1}^{m} M_{:,j}$ Sum of columns.

```
>> sum(M); % == sum(M,1);
     7     8     5

>> sum(M'); % == sum(M,2);
     9
    11
```

Useful Matrix Functions (2)

```
>> M = [2,3,4;5,5,1;8,0,1]; V1 = [1;44;3]; V2 = [9;8;7];
```

Diagonal of M.

```
>> disp(diag(M))
     2
     5
     1
```

Upper triangular part of M.
Lower triangular part of M.

```
>> disp(triu(M))
     2     3     4
     0     5     1
     0     0     1

>> tril(M);
```

$\sum_{i=1}^{n} V1_i V2_i$ Vector dot product.

```
>> disp(dot(V1,V2))
   382
```

Useful Matrix Functions (3)

```
>> M = [2,3,4;5,5,1;8,0,1];
```

Replicate and tile matrices: `repmat(M, [a,b])`

```
>> disp(repmat(M, [1,3])) % 1-by-3 tiling
     2     3     4     2     3     4     2     3     4
     5     5     1     5     5     1     5     5     1
     8     0     1     8     0     1     8     0     1

>> disp(repmat(M, [2,1])) % 2-by-1 tiling
     2     3     4
     5     5     1
     8     0     1
     2     3     4
     5     5     1
     8     0     1
```

Useful Matrix Functions (4)

Create specialized n-by-m matrices with $func(n, m)$

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

```
>> ones(3,4);
```

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
>> eye(3); % n=m=3
```

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$$

```
>> zeros(1,4);
```

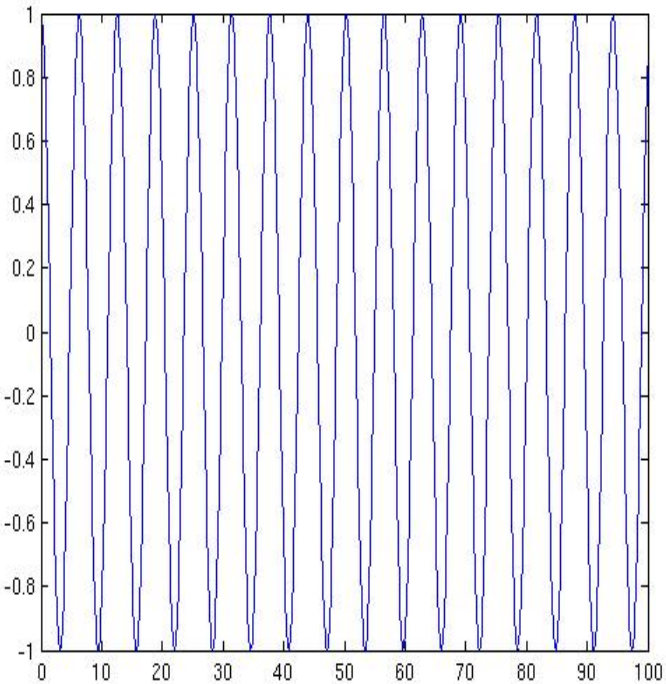
Create pseudorandom matrix.

```
>> disp(rand(3,4))
    0.3922    0.7060    0.0462
    0.6555    0.0318    0.0971
    0.1712    0.2769    0.8235
```

Plotting Functions

Plot 2-D lines with `plot(X,Y)`

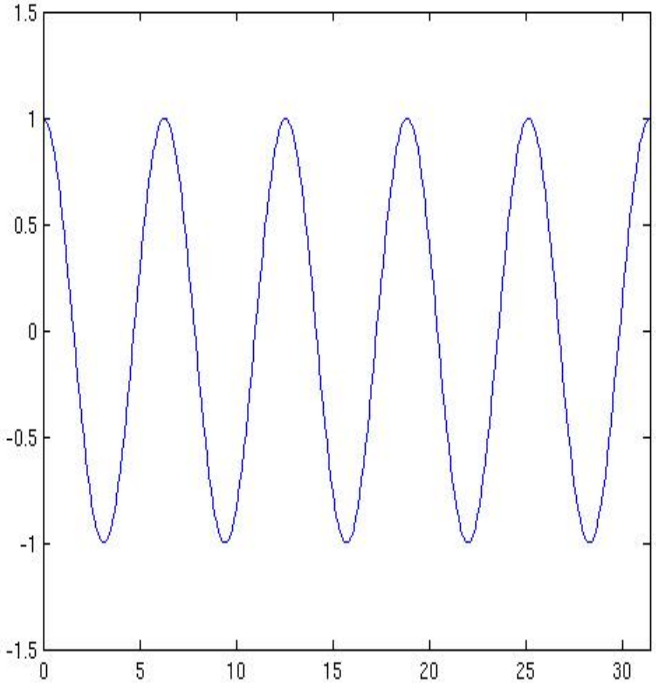
X specifies locations along x-axis, and Y gives data values to plot.




```
>> X = 1:.1:100;
>> Y = cos(X);
>> figure
>> plot(X,Y)
```

Plotting Functions

Set axis limits with `axis([x1 x2 y1 y2])`

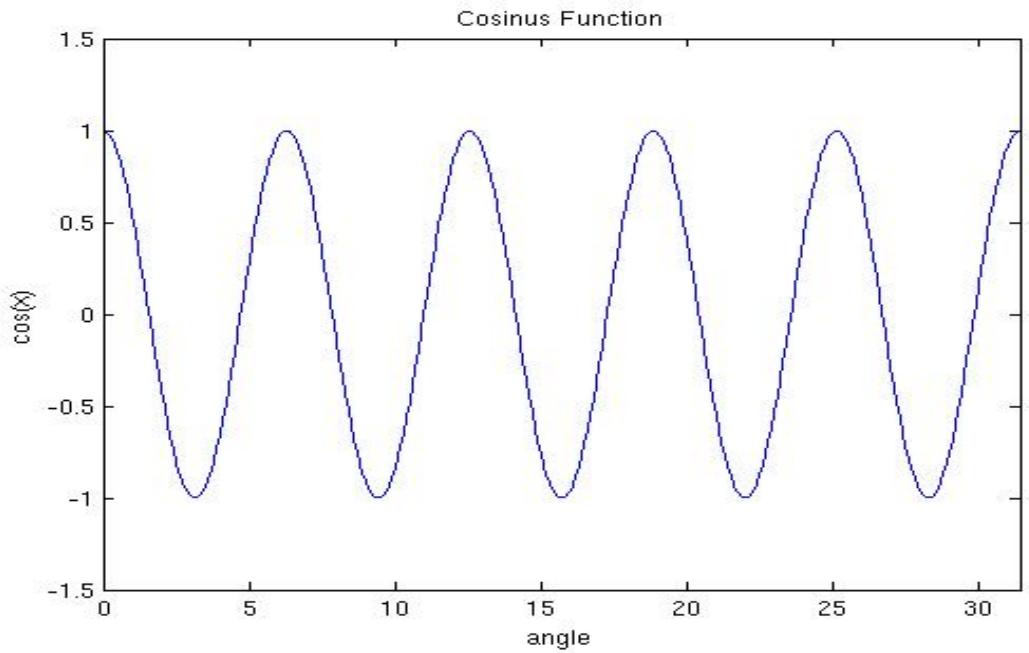



```
>> X = 1:.1:100;
>> Y = cos(X);
>> figure;
>> plot(X,Y)
>> axis([0 10*pi -1.5 1.5]);
```

Plotting Functions

Add title and labels with `title(str)`, `xlabel(str)`, `ylabel(str)`

```
>> X = 1:.1:100; Y = cos(X); figure; plot(X,Y)
>> axis([0 10*pi -1.5 1.5]);
>> title('Cosinus Function', 'fontsize', 10);
>> xlabel('angle');
>> ylabel('cos(x)');
```



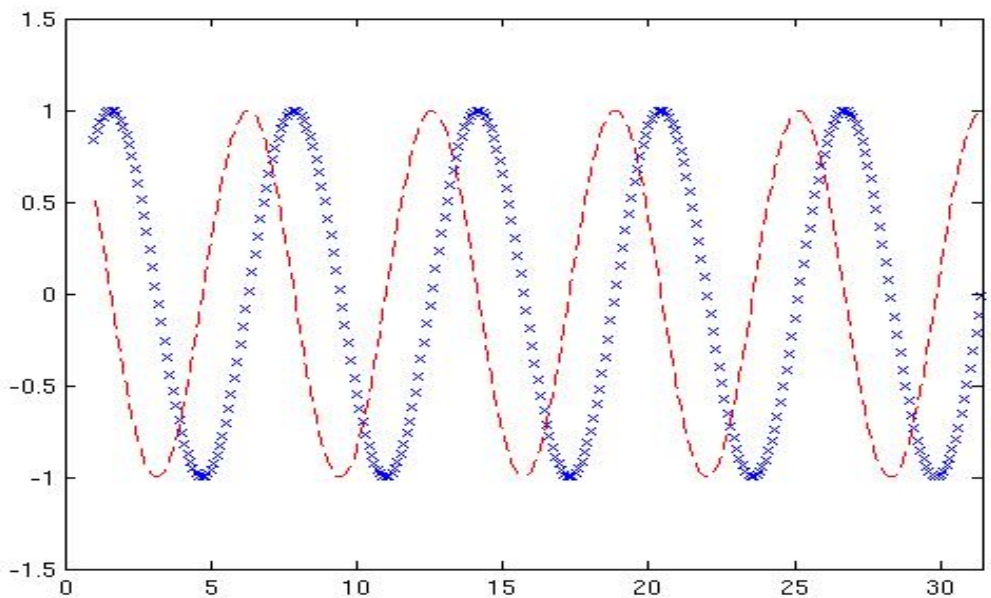
Plotting Functions

Plot several 2-D lines with `plot(X1,Y1, ..., Xn,Yn)`

```
>> X = 1:.1:100; Y1 = cos(X); Y2 = sin(X); figure;
>> plot(X,Y1,X,Y2)
```

Specify line style, marker, and colour with `plot(X1,Y1,'LSpec', ..., Xn,Yn,'LSpec')`

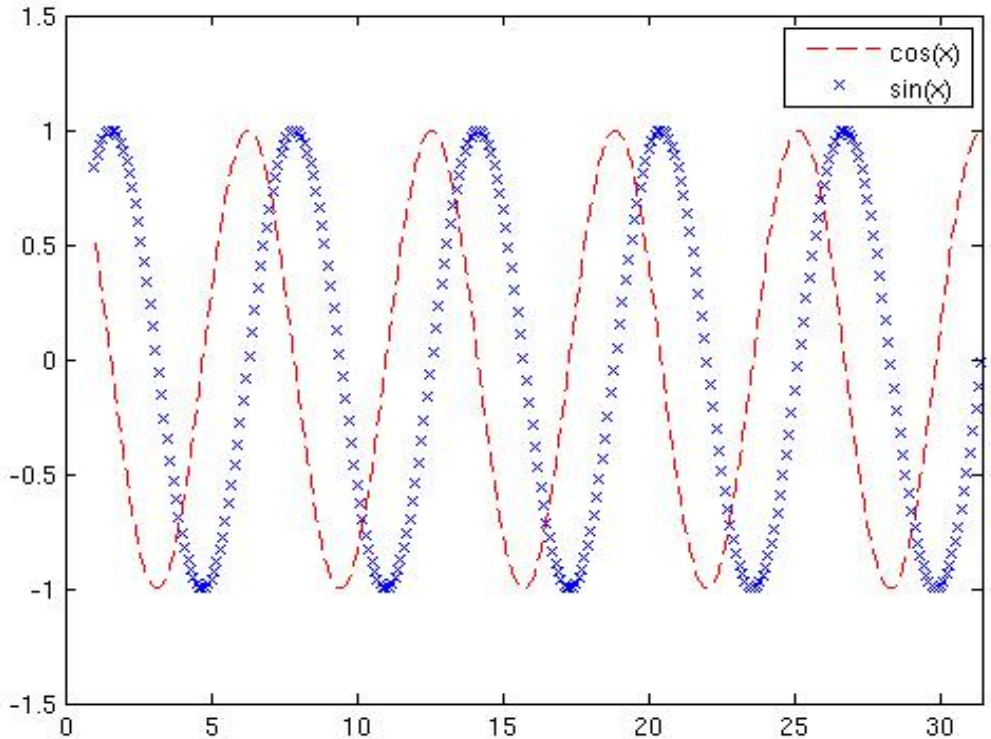
```
>> plot(X,Y1,'--r',X,Y2,'x') % red, dashed line; crosses
```



Plotting Functions

Add legend with `legend('Legend1', ..., 'Legendn')`

```
>> legend('cos(x)','sin(x)')
```



Plotting Discrete Data

Plot discrete data as a bar chart with `bar(Y)` or `bar(X,Y)`

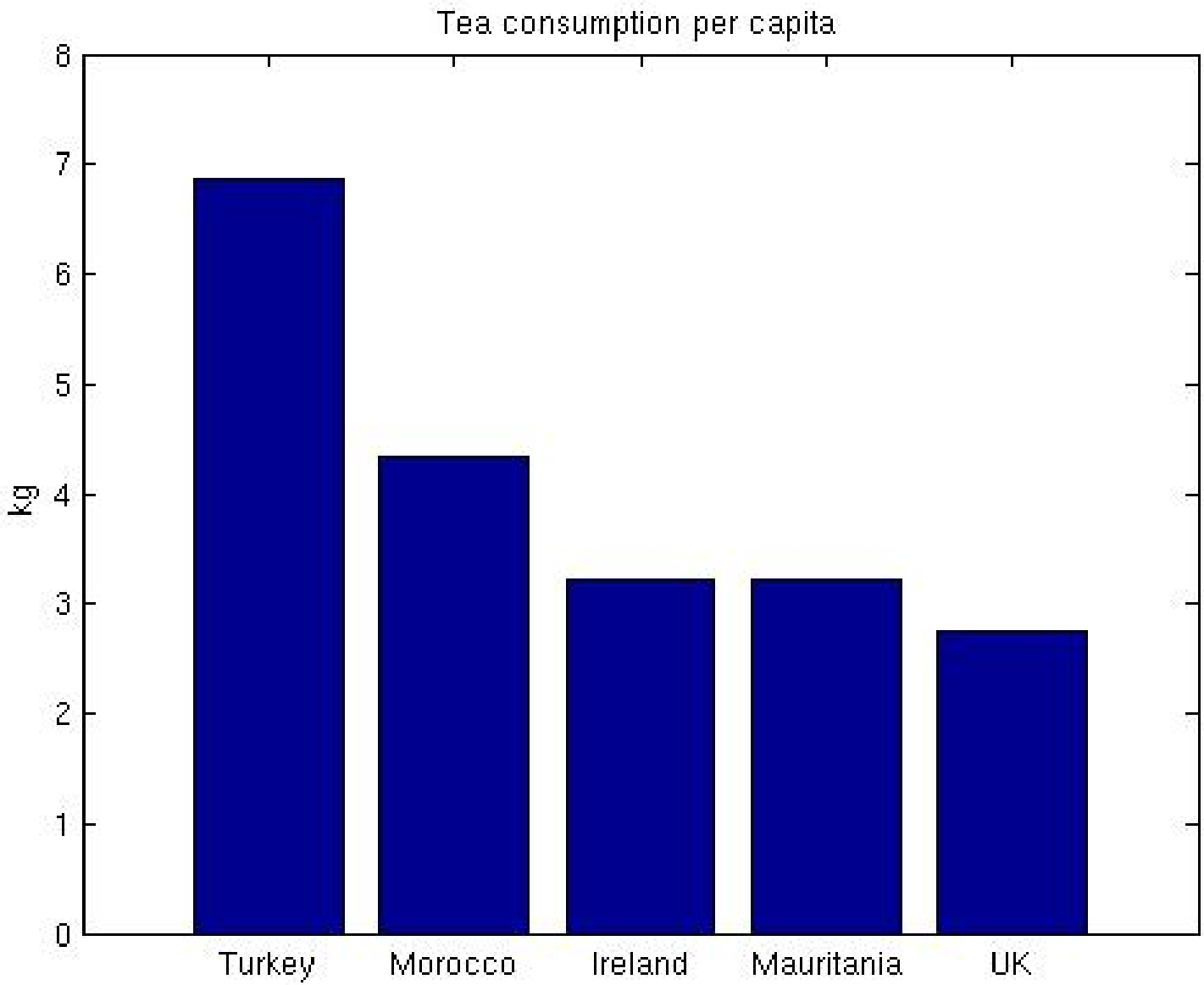
The table lists the countries with the highest annual per capita consumption of tea in kg (2009)[a]:

country	Turkey	Morocco	Ireland	Mauritania	UK
kg	6.87	4.34	3.22	3.22	2.74

[a] wikipedia.org/wiki/List_of_countries_by_tea_consumption_per_capita

```
>> Y = [6.87, 4.34, 3.22, 3.22, 2.74];
>> X = {'Turkey','Morocco','Ireland','Mauritania','UK'};
>> bar(Y)
>> ylabel('kg')
>> title('Tea consumption per capita')
>> axis([0 length(X)+1 0 8])
>> set(gca, 'XTickLabel', X) % set country labels
```

Plotting Discrete Data



Defining Functions

Declare function name, inputs and outputs with

`function [y1,...,yN] = func_name(x1,...,xM)`

- Function codes are saved in files with extension `.m`.
- Multiple functions can be declared in one file.
- The file name should match the name of the first function in the file.

Define function in file my_sum.m

```
function [sum] = my_sum(x)
    sum = 0;
    for i=1:length(x)
      sum = sum+x(i);
    end
```

Call function from command line

```
>> s = my_sum([1,2,3,4.4]);
>> s
s =
    10.4000
```

Exercise

The previously defined function my_sum only works for row vectors (1-by-m matrices):

```
>> s = my_sum([1,2;4,5])
s =
     5
```

Modify the code so that it can compute the sum of the rows of n-by-m matrices:

```
>> s = my_sum([1,2;4,5])
s =
     5     7
```