ΠΛΗΡΟΦΟΡΙΚΗ Ι (MATLAB)
Ενότητα 1

Σημειώσεις βασισμένες στο βιβλίο "Το MATLAB στην Υπολογιστική Επιστήμη και Τεχνολογία – Μια Εισαγωγή"

Περιεχόμενο μαθήματος:

- Αλγοριθμική επίλυση προβλημάτων
- Προγραμματισμός με MATLAB
- Εφαρμογές σε μαθηματικά και μη προβλήματα

Ανάπτυξη προγραμμάτων: MATLAB – Octave

Αλγόριθμος: Βήμα προς βήμα διαδικασία για την επίλυση κάποιου προβλήματος. Το πλήθος των βημάτων πρέπει να είναι πεπερασμένο.

- Είσοδος (≥ 0 δεδομένα)
- Έξοδος (≥ 1 αποτελέσματα)
- Ορισμένος (περιέχει σαφείς οδηγίες)
- Κάθε οδηγία, μεμονωμένα: εξαιρετικά απλή
- Καλύπτει όλες τις δυνατές καταστάσεις
- Εξασφαλίζει τον τερματισμό (πεπερασμένος αριθμός βημάτων ή χρόνος)

Πρόγραμμα: Ακριβής διατύπωση ενός αλγορίθμου σε μια γλώσσα προγραμματισμού.

Υπολογιστική επίλυση προβλήματος:

1) Ανάλυση δεδομένων του προβλήματος
2) Μαθηματική διατύπωση του προβλήματος
3) Ανάπτυξη του αλγόριθμου (σχεδιασμός ή επιλογή κατάλληλου αλγόριθμου

(συνήθως: ένα πρόβλημα – πολλοί αλγόριθμοι)

4) Διατύπωση αλγόριθμου σε γλώσσα προγραμματισμού: πρόγραμμα
5) Εκτέλεση προγράμματος για συγκεκριμένα δεδομένα
6) Ερμηνεία αποτελεσμάτων

Βασικές αλγοριθμικές ενέργειες

- Είσοδος / Έξοδος
- Πράξεις ή αναθέσεις τιμών σε μεταβλητές
- Έλεγχος ποσοτήτων – επιλογή δράσης
- Επαναληπτική εκτέλεση
- Τερματισμός

Θέμα εισαγωγικού μαθήματος: Μετατροπή μαθητικών τύπων σε πρόγραμμα υπολογιστή.

Πρόβλημα: Το εμβαδόν σφαίρας ακτίνας r δίνεται από τη σχέση: $A = 4\,\pi\,r^2$. Πώς αυξάνεται το A όταν αυξάνεται το r κατά δr;

i) $\delta A = 4\,\pi\,(r + \delta r)^2 - 4\,\pi\,r^2$
ii) $\delta A = 4\,\pi\,(2r + \delta r)\,\delta r$
iii) $\delta A = 4\,\pi\,r\,\delta r$

Ας δούμε πρώτα ένα πρόγραμμα που να υπολογίζει απλά το Α:

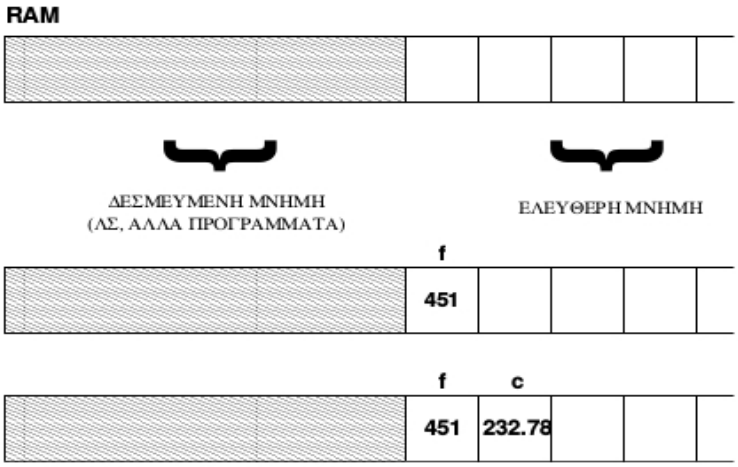
```
% Το script SurfArea
% r: η ακτίνα της σφαίρας
% A: το εμβαδόν της σφαίρας

r = input('Δώσε την ακτίνα r:');
A = 4*3.141592*r*r;
fprintf('Το εβδαδόν ισούται με %f.\n', A);
```

Η βασική δομή ενός προγράμματος:

Τα βασικά στοιχεία του:

- **Μεταβλητές** → θέσεις μνήμης για την αποθήκευση τιμών

- Έχουν όνομα και τιμή η οποία μπορεί να μεταβληθεί.



- Για να χρησιμοποιηθούν σε ένα πρόγραμμα, πρέπει πρώτα να έχουν πάρει κάποια τιμή (να έχουν αρχικοποιηθεί).

- Τα ονόματα ξεκινάνε με γράμμα και μπορούν να περιέχουν γράμματα, αριθμούς και _

- καλό είναι τα ονόματα να είναι περιγραφικά (π.χ., mesos_oros)
- υπάρχει διάκριση μετακύ κεφαλαίων και πεζών

- **Εκφράσεις** → υπολογισμοί που οδηγούν σε κάποια τιμή.

Αποτελούν συνδυασμούς μεταβλητών, σταθερών, τελεστών και συναρτήσεων.

- Σταθερές: αριθμητικές ποσότητες που δεν αλλάζουν

```
15    0     -2     1.38       4.0     +2.3
```

Ειδική σταθερά: `pi` → π

Παριστάνονται και σε εκθετική μορφή: ο αριθμός $x \cdot 10^y$ → `xey` ή `xEy`

Π.χ., 500 → `5e2`
`2.5e-3    9.109E-31      -4e10`

- Αριθμητικοί τελεστές:

- Μονομελείς: πρόσημα + και -
- Διμελείς: +, −, *, /, ^

- Υπολογισμός εκφράσεων:

Προτεραιότητα πράξεων:
1. παρενθέσεις, αρχίζοντας από τις εσωτερικές
2. *, /
3. +, -

Πράξεις στο ίδιο επίπεδο προτεραιότητας: από αριστερά προς τα δεξιά.

- **Εντολές εκχώρησης** → ο τρόπος εισόδου τιμών στις μεταβλητές.

Πραγματοποιούνται με τον τελεστή εκχώρησης =

Γενική μορφή: `μεταβλητή = έκφραση`

Βήματα: i) Υπολογίζεται η τιμή της έκφρασης στα δεξιά του τελεστή εκχώρησης
ii) Η τιμή αποθηκεύεται στη μεταβλητή στα αριστερά του τελεστή εκχώρησης

Π.χ.

```
x = 2 * 3.14;
y = x + 1;      [Το x έχει ήδη πάρει κάποια τιμή, οπότε είναι ok. Διαφορετικά θα ήταν λάθος!]
y = y + 1;      [Σωστό! Είναι εντολή εκχώρησης και όχι αλγεβρική εξίσωση]
A = 4*3.141592*r*r;      [Εντολή εκχώρησης από το πρόγραμμα]
```

Τώρα γνωρίζουμε ότι μπορεί να γραφεί καλύτερα: `A = 4*pi*r^2;`

- **Είσοδος** → ανάγνωση δεδομένων από το πληκτρολόγιο

- Με τη συνάρτηση `input`:

```
μεταβλητή = input('συμβολοσειρά')
     ↓                    ↓
όνομα μεταβλητής      κείμενο σε μονά εισαγωγικά
```

π.χ., `r = input('Δώσε την ακτίνα r: ')`

→ Εμφανίζεται η συμβολοσειρά στην οθόνη και περιμένει πληκτρολόγηση
→ Με το πάτημα του ENTER, ό,τι έχει πληκτρολογηθεί εκχωρείται (αποθηκεύεται) στη μεταβλητή στα αριστερά του =.

- **Έξοδος** → εμφάνιση αποτελεσμάτων στην οθόνη

➢ Με τη συνάρτηση `disp()`

```
disp(έκφραση)   ή    disp('μήνυμα')
```

π.χ., `disp(x)` → εμφανίζει την τιμή της μεταβλητής x
`disp(2+3)` → 5
`disp('Hello')` → Hello
`disp(x,y)` → ΛΑΘΟΣ! Η `disp` δέχεται μόνο ένα όρισμα.

➢ Με τη συνάρτηση `fprintf()`

```
fprintf('συμβολοσειρά με τελεστές προσαρμογής', λίστα μετ/τών)
```

- Η συμβολοσειρά: κείμενο που θα εμφανιστεί στην έξοδο
- Οι τελεστές προσαρμογής: ειδικοί τελεστές που καθορίζουν τον τρόπο και τη θέση εμφάνισης των τιμών των μεταβλητών που βρίσκονται στη λίστα μεταβλητών.
- Τελεστής αλλαγής γραμμής: `\n`

- Τελεστές προσαρμογής της `fprintf`:

  - Εισάγονται με το σύμβολο `%`
  - Συνηθέστεροι:
    - `%f` κανονική δεκαδική μορφή
    - `%d` για ακέραιους
    - `%e` εκθετική μορφή
    - `%g` γενική μορφή (συντομότερη δυνατή μορφή)

Γενικά: `%w.df` → εκτύπωση σε w συνολικά θέσεις με d δεκαδικά ψηφία.

Π.χ.,

```
r = 6367; A = 4*pi*r^2;

fprintf('Το εμβαδόν σφαίρας με ακτίνα %f είναι %f\n', r,A);
```

→ Το εμβαδόν σφαίρας με ακτίνα 6367.000000 είναι 509424190.194245

```
fprintf('Το εμβαδόν σφαίρας με ακτίνα %g είναι %e\n', r,A);
```

→ Το εμβαδόν σφαίρας με ακτίνα 6367 είναι 5.09424e+08

```
fprintf('Το εμβαδόν σφαίρας με ακτίνα %10.2f είναι %10.3e\n', r,A);
```

→ Το εμβαδόν σφαίρας με ακτίνα ___6367.00 είναι _5.094e+08

- **Σχόλια**

- Ξεκινάνε από το σύμβολο % μέχρι το τέλος της γραμμής
- Πολύ σημαντικά για την επεξήγηση ενός script ή τμημάτων κώδικα

- Το script `SurfArea` αποθηκεύεται ως: SurfArea.m
- Εκτελείται ως: `>> SurfArea`

→ Το πρόγραμμα για την επίλυση του αρχικού προβλήματος (πώς αυξάνεται το εμβαδόν σφαίρας (δA) όταν αυξάνεται η ακτίνα κατά δr):

```
% Script Eg1_1
% Αύξηση εμβαδού επιφάνειας

% Είσοδος δεδομένων
r = input('Δώσε την ακτίνα (χιλιόμετρα):');
delta_r = input('Δώσε την αύξηση (χιλιοστά):');

fprintf('Ακτίνα σφαίρας = %12.6f χιλιόμετρα\n', r)
fprintf('Αύξηση ακτίνας = %12.6f χιλιοστά\n\n', delta_r)
disp('Αύξηση εμβαδού επιφάνειας:')
dr = delta_r/10^6;  % μετατροπή mm σε km

% Μέθοδος 1
delta_A1 = (4*pi*(r + dr)^2 - 4*pi*r^2)*10^6;
fprintf('\n   Μέθοδος 1: %15.6f τετρ/κά μέτρα\n', delta_A1)

% Μέθοδος 2
delta_A2 = (4*pi*(2*r + dr)*dr)*10^6;
fprintf('   Μέθοδος 2: %15.6f τετρ/κά μέτρα\n', delta_A2)

% Μέθοδος 3
delta_A3 = (8*pi*r*dr)*10^6;
fprintf('   Μέθοδος 3: %15.6f τετρ/κά μέτρα\n', delta_A3)
```

→ Αποθήκευση ως Eg1_1.m
→ Εκτέλεση: `>> Eg1_1`

Παράδειγμα εξόδου για είσοδο: 6367 και 1.234:

```
Ακτίνα σφαίρας = 6367.000000 χιλιόμετρα
Αύξηση ακτίνας =    1.234000 χιλιοστά

Αύξηση εμβαδού επιφάνειας:

   Μέθοδος 1:   197464.823723 τετρ/κά μέτρα
   Μέθοδος 2:   197464.881659 τετρ/κά μέτρα
   Μέθοδος 3:   197464.881640 τετρ/κά μέτρα
```

- Σφάλματα παραδοχών του προγράμματος:

(αν υποθέσουμε ότι προσπαθούσαμε να υπολογίσουμε την αύξηση του εμβαδού της Γης)

  - Η Γη δεν είναι τέλεια σφαίρα, είναι ελλειψοειδές.
  - Η εσωτερική σταθερά pi ισούται με 3.141592265358979 και προφανώς όχι ακριβώς με το π.
- Σφάλματα υπολογισμών: δημιουργούνται κάθε φορά που γίνεται κάποια αριθμητική πράξη. Έτσι εξηγείται και το διαφορετικό αποτέλεσμα των Μεθόδων 1 και 2, οι οποίες είναι μαθηματικά ισοδύναμες.
- Σφάλματα προσεγγίσεων: όταν επιλέγουμε να προσεγγίσουμε τη λύση ενός προβλήματος, όπως π.χ. με τη Μέθοδο 3.

Κάποια επιπλέον στοιχεία:

- Μορφοποίηση εξόδου με την εντολή `format`:

  `format short`: εμφάνιση τιμών σε δεκαδική μορφή με 5 ψηφία.
  `format long`: εμφάνιση τιμών σε δεκαδική μορφή με 15 ψηφία.
  `format short e`: εμφάνιση τιμών σε επιστημονική μορφή με 5 ψηφία.
  `format long e`: εμφάνιση τιμών σε επιστημονική μορφή με 15 ψηφία.
  `format`: επαναφορά στην προεπιλεγμένη μορφή (short)

- Πολλαπλές εντολές ανά γραμμή

```
x = 5;
disp(x);
y = x^2;
```

  Ισοδύναμα:

```
x = 5; disp(x); y = x^2;
```

- Έλεγχος εμφάνισης με το `;`

```
x = 5;
disp(x);
```

  Ισοδύναμα:

```
x = 5
```

- Συνέχεια εντολής σε επόμενη γραμμή

  Με τον τελεστή `...`

  Π.χ.,

```
x = (4*pi*(r+dr)^2 - 4*pi*r^2)*10^6;
```

  Ισοδύναμα:

```
x = (4*pi ...
    *(r+dr)^2 - 4*pi*r^2)...
    *10^6;
```

ΠΛΗΡΟΦΟΡΙΚΗ Ι (MATLAB)
Ενότητα 2

Σημειώσεις βασισμένες στο βιβλίο "Το MATLAB στην Υπολογιστική Επιστήμη και Τεχνολογία – Μια Εισαγωγή"

Έλεγχος συνθηκών - `if`

Ας μελετήσουμε το πρόβλημα του υπολογισμού του ελάχιστου της συνάρτησης $f(x)=x^2+bx+c$ στο διάστημα $[L,R]$. Έτσι, θα πάρουμε μια ιδέα για τους ελέγχους συνθηκών.

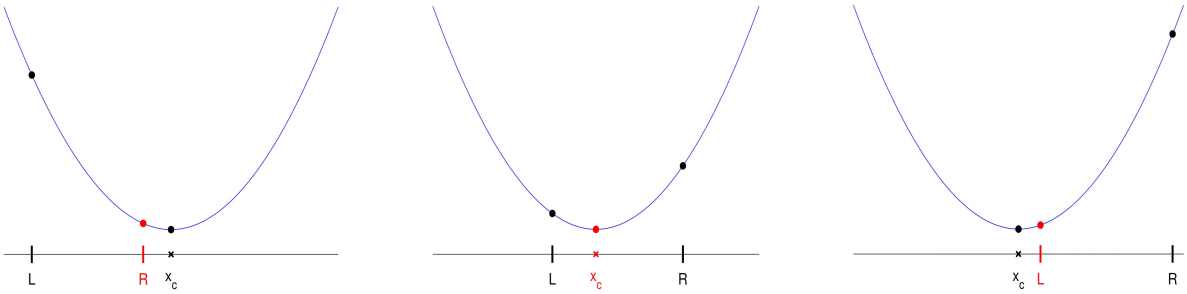
Γνωρίζουμε πως το ελάχιστο βρίσκεται στο κρίσιμο σημείο $x_c=-\frac{b}{2}$.

- Αν το $x_c \in [L,R]$, τότε το ελάχιστο είναι το $f(x_c)$.
- Αν το $x_c \notin [L,R]$, τότε το ελάχιστο είναι είτε το $f(L)$, είτε το $f(\mathrm{R})$.

Να γραφεί πρόγραμμα, το οποίο να ζητάει τους πραγματικούς αριθμούς *L*, *R*, *b* και *c* και να εμφανίζει το ελάχιστο της $f(x)$ στο $[L,R]$, καθώς και την τιμή του *x* στην οποία εμφανίζεται.

Υπάρχουν 3 περιπτώσεις:

(1) (2) (3)



(1)	(2)	(3)
$x_c > R$	$L \le x_c \le R$	$x_c < L$
min = f(R)	min = f(x_c)	min = f(L)

Ο ψευδοκώδικας του αλγορίθμου:

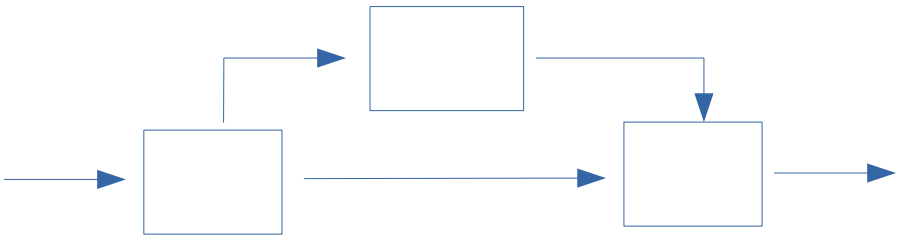
```
if xc < L
    εμφάνισε f(L) και L
else if L ≤ xc ≤ R
    εμφάνισε f(xc) και xc
else
    εμφάνισε f(R) και R
```

Για να μετατρέψουμε τον ψευδοκώδικα σε πρόγραμμα MATLAB χρειαζόμαστε:

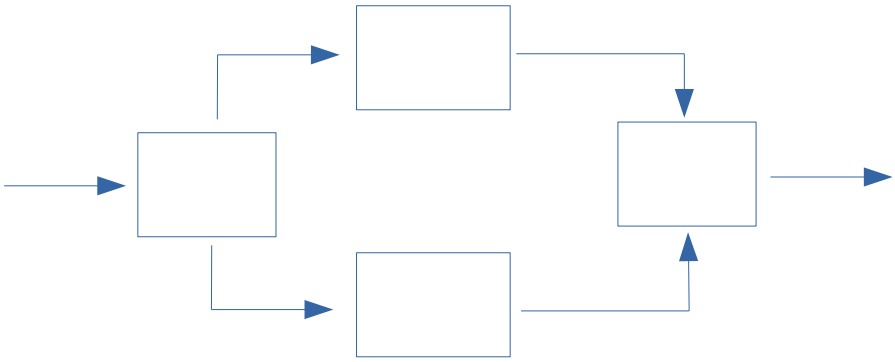
i. μια δομή γλώσσας που να επιτρέπει τη σύγκριση τιμών,
ii. μια δομή γλώσσας που να εκτελεί το κατάλληλο τμήμα κώδικα, ανάλογα με το αποτέλεσμα της σύγκρισης.

Άρα, υπάρχει η ανάγκη, ένα πρόγραμμα να μην εκτελείται πάντα σειριακά:

αλλά έτσι:



ή έτσι:



κτλ.

Δηλαδή, διάφορα τμήματα κώδικα εκτελούνται υπό συνθήκη.

Στο MATLAB, αυτό γίνεται με την εντολή if.

Η πιο βασική της μορφή είναι η δομή `if-else`:

Σύνταξη της `if-else`:

```
if λογική_έκφραση
   τμήμα κώδικα που εκτελείται αν η λογική έκφραση είναι αληθής
else
   τμήμα κώδικα που εκτελείται αν η λογική έκφραση είναι ψευδής
end
```

Λογικές (Boolean) εκφράσεις:

Όπως οι αριθμητικές εκφράσεις (πράξεις) όταν υπολογίζονται παράγουν αριθμητικές τιμές, οι λογικές εκφράσεις παράγουν την τιμή true (Αληθές) ή false (Ψευδές).

- Το false αντιστοιχεί στο 0 (και το 0 στο false).
- Το true αντιστοιχεί στο 1 (και οποιαδήποτε μη μηδενική τιμή αντιστοιχεί στο true).

Οι λογικές εκφράσεις υλοποιούνται με σχεσιακούς τελεστές (έναντι των αριθμητικών τελεστών των αριθμητικών εκφράσεων).

Μαθηματικά	MATLAB
<	<
≤	<=
>	>
≥	>=
=	==
≠	~=

Παράδειγμα της εντολής `if`:

Εκτελείται αν η συνθήκη είναι αληθής.

Εκτελείται αν η συνθήκη είναι ψευδής.

```
x=5;
y=2;
if (x>y)
    disp('Το x είναι μεγαλύτερο του y.');
else
    disp('Το x είναι μικρότερο ή ίσο του y.');
end
```

Λογικές Πράξεις:

Οι λογικές πράξεις υλοποιούνται με τους λογικούς τελεστές:

- Σύζευξη (ΚΑΙ – AND): `&&`
- Διάζευξη (Ή – OR): `||`
- Άρνηση (ΔΕΝ – NOT): `~`

} διμελείς τελεστές

Στο αρχικό παράδειγμα, ας υποθέσουμε ότι θέλουμε να εμφανίσουμε τα $f(x_c)$ και x_c αν είμαστε στην περίπτωση (2), (δηλαδή αν το $x_c \in [L,R]$) ή ένα πληροφοριακό μήνυμα σε αντίθετη περίπτωση.

Για να ελέγξουμε εάν το $x_c \in [L,R]$, απαιτούνται δύο συγκρίσεις: πρέπει το $x_c \geq L$ ΚΑΙ το $x_c \leq R$.

```
xc = -b/2;
if xc>=L && xc<=R
   fxc = c-(b/2)^2;
   fprintf('f(xc) = %6.3f xc = %6.3f\n', fxc,xc)
else
   disp('Είτε xc < L, είτε xc > R')
end
```

Πίνακας αληθείας:

p	q	p&&q	p||q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Αν το `p` είναι true, τότε το `~p` είναι false.

Αν το `p` είναι false, τότε το `~p` είναι true.

Προτεραιότητα τελεστών:

- πρώτα οι **αριθμητικοί** τελεστές
- μετά οι **σχεσιακοί** τελεστές και η άρνηση
- μετά οι διμελείς **λογικοί** τελεστές

Άλλες δομές του if:

- Σκέτο `if` :

```
if  λογική έκφραση
   κώδικας
end
```

π.χ.

```
if L>R
   temp = L;
   L = R;
   R = temp;
end
```

- `if – elseif` :

```
if λογική έκφραση 1
   κώδικας
elseif λογική έκφραση 2
   κώδικας
elseif λογική έκφραση 3
   κώδικας
else
   κώδικας
...
end
```

Εκτελείται, αν η λογική έκφραση 1 είναι true.

Εκτελείται, αν η λογική έκφραση 1 είναι false και η λογική έκφραση 2 είναι true.

Εκτελείται, αν η λογική έκφραση 1 είναι false και η λογική έκφραση 2 είναι false, αλλά η λογική έκφραση 3 είναι true.

Εκτελείται, αν όλες οι παραπάνω λογικές εκφράσεις είναι false.

→ Ας δούμε τώρα το πρόγραμμα για το αρχικό μας πρόβλημα, δηλαδή της εύρεσης του ελάχιστου της συνάρτησης $f(x)=x^2+bx+c$ στο διάστημα $[L,R]$.

```
% Script Eg1_2
%
% min της x^2+bx+c στο [L,R]

% Είσοδος δεδομένων
b=input('Δώσε το b: ');
c=input('Δώσε το c: ');
L=input('Δώσε το L: ');
R=input('Δώσε το R, με L<R: ');
fprintf('Δευτεροβάθμια: x^2+bx+c, b = %5.2f , c = %5.2f \n', b, c);
fprintf('Διάστημα: [L,R], L = %5.2f , R = %5.2f \n \n', L, R);

%Υπολογισμός κρίσιμου σημείου
xc= - b/2;
if xc < L
    % Το min είναι στο αριστερό όριο
    fL = L^2+b*L+c;
    fprintf('x ελαχιστοποίησης = %5.2f \n', L)
    fprintf('Ελάχιστη τιμή της f = %5.2f \n', fL)
elseif L <= xc && xc <=R
    % Το min στο κρίσιμο σημείο
    fxc = c - (b/2)^2;
    fprintf('x ελαχιστοποίησης = %5.2f \n', xc)
    fprintf('Ελάχιστη τιμή της f = %5.2f \n', fxc)
else
    % Το min στο δεξί όριο
    fR = R^2 + b*R +c
    fprintf('x ελαχιστοποίησης = %5.2f \n', R)
    fprintf('Ελάχιστη τιμή της f = %5.2f \n', fR)
end
```

→ Μια βελτίωση της δομής του προγράμματος:
Διαχωρισμός της εξόδου από τους υπολογισμούς

\+ : Λιγότερη πληκτρολόγηση (των εντολών εξόδου)
– : Ίσως ανάγκη δημιουργίας επιπλέον μεταβλητών

```
. . .
xc= - b/2;
if xc < L
    xmin = L;
elseif L <= xc && xc <=R
    xmin = xc;
else
    xmin = R;
end
fmin = xmin^2 + b*xmin +c;
fprintf('x ελαχιστοποίησης = %5.2f \n', xmin)
fprintf('Ελάχιστη τιμή της f = %5.2f \n', fmin)
```

} 1 φορά!

Παρατηρούμε εδώ, πως χρησιμοποιήσαμε ως επιπλέον μεταβλητή το xmin, αλλά επειδή κάναμε τον υπολογισμό του min της f έξω από το if, αντικαταστήσαμε τις τρεις μεταβλητές fL, fxc και fR με μία (fmin).

Κάποια επιπλέον στοιχεία:

► `min` και `max`:

Εύρεση ελάχιστου / μέγιστου μεταξύ δύο τιμών (μεταβλητών ή αριθμητικών εκφράσεων)

Σύνταξη: `min(αριθμητική έκφραση 1, αριθμητική έκφραση 2)`

► Αν *x* πραγματικός αριθμός:

`floor(x)` → στρογγυλοποίηση προς το $-\infty$.
`ceil(x)` → στρογγυλοποίηση προς το $+\infty$.
`round(x)` → στρογγυλοποίηση προς το πλησιέστερο
`fix(x)` → στρογγυλοποίηση προς το 0

► Αν *x*, *y* θετικοί ακέραιοι:

`rem(x,y)` → υπόλοιπο της διαίρεσης του *x* με το *y*.

[Μπορεί να εφαρμοστεί και σε πραγματικές τιμές]

► Άλλες χρήσιμες συναρτήσεις:

`sqrt(x)` → $\sqrt{x}$
`abs(x)` → $|x|$
`exp(x)` → e^x
`log(x)` → $\ln(x)$
`log10(x)` → $\log(x)$
`sin(x)` → ημ(*x*)
`cos(x)` → συν(*x*)
`tan(x)` → εφ(*x*)
`rand( )` → τυχαίος αριθμός στο (0,1)

►Shortcircuiting

Σε μια σύνθετη λογική έκφραση, το 2ο μέρος υπολογίζεται μόνο όταν δε μπορεί να βγει συμπέρασμα (να υπολογιστεί το αποτέλεσμα) μόνο από το 1ο.

Άρα στη λογική έκφραση (ΛΕ): `ΛΕ1 && ΛΕ2` η ΛΕ2 υπολογίζεται μόνο εάν η ΛΕ1 είναι true. Αυτό συμβαίνει διότι αν η ΛΕ1 ήταν false, το αποτέλεσμα θα ήταν ούτως ή άλλως false.

Ομοίως, στη λογική έκφραση: `ΛΕ1 || ΛΕ2` η ΛΕ2 υπολογίζεται μόνο εάν η ΛΕ1 είναι false.

Καταλήγουμε επομένως στο συμπέρασμα πως είναι προτιμότερο να συντάσσεται πρώτα η έκφραση που δε δημιουργεί προβλήματα, π.χ., `(x~=0) && (y/x==0)` και η απλούστερη από τις δύο εκφράσεις, για παράδειγμα: `(x>1) || (3*x^2 - 2*x < y/(x^2 + 2))`.

Π.χ., η έκφραση `(3==7) && (2==(3/0))` θα επέστρεφε ως αποτέλεσμα false (0) και όχι σφάλμα, που κάποιος θα περίμενε.

► Ένα παράδειγμα σύνθετης λογικής έκφρασης είναι ο έλεγχος για το εάν ένα έτος είναι δίσεκτο.

Ένα έτος θεωρείται δίσεκτο εάν διαιρείται με το 4 αλλά όχι με το 100 ή αν διαιρείται με το 400.

Ο τρόπος που θα εκφράζαμε τη λογική αυτή έκφραση είναι:

```
(rem(year,4) == 0 && rem(year,100) ~= 0) || rem(year,400) == 0
```

ή καλύτερα:

```
rem(year,400) == 0 || (rem(year,4) == 0 && rem(year,100) ~= 0)
```

► Σύγκριση λογικών μεταβλητών με τις τιμές true / false:

Παρακάτω, βλέπουμε δύο παραδείγματα, στα οποία χρησιμοποιούνται δύο λογικές μεταβλητές (`isLeapYear` και `isEven`) και μία ακέραια μεταβλητή (`n`).

Παράδειγμα 1:

Αντί της: `if isLeapYear == true` προτιμάται η: `if isLeapYear` .

Παράδειγμα 2:

Αντί της :

```
if  rem(n,2) == 0
    isEven = true;
else
    isEven = false;
end
```

προτιμάται η:

```
isEven = (rem(n,2) == 0);
```

.

ΠΛΗΡΟΦΟΡΙΚΗ Ι (MATLAB)
Ενότητα 3

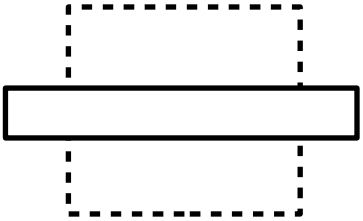
Σημειώσεις βασισμένες στο βιβλίο "Το MATLAB στην Υπολογιστική Επιστήμη και Τεχνολογία – Μια Εισαγωγή"

ΔΟΜΕΣ ΕΠΑΝΑΛΗΨΗΣ

Πρόβλημα: Για δεδομένο αριθμό Α, μα βρεθεί η $\sqrt{A}$.
Γεωμετρική αναδιατύπωση: Για θετικό αριθμό Α, να βρεθεί τετράγωνο με εμβαδόν Α.

- Μια αρχική ιδέα:

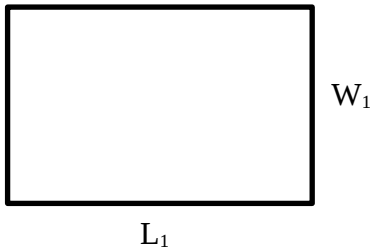
- Παρατήρηση:



Η απάντηση είναι μεταξύ του L και του W:
W < S < L

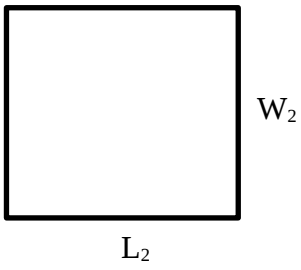
- Βασική ιδέα:

$$L_1 = \frac{L+W}{2}$$
$$W_1 = \frac{A}{L_1}$$



επανάληψη:

$$L_2 = \frac{L_1+W_1}{2}$$
$$W_2 = \frac{A}{L_2}$$



Ένα πρώτο script:

```
A = input('A:');
L0 = A; W0 = A/L0;
L1 = (L0+W0)/2; W1 = A/L1;
L2 = (L1+W1)/2; W2 = A/L2;
L3 = (L2+W2)/2; W3 = A/L3;
L4 = (L3+W3)/2; W4 = A/L4;
```

Δεν υπάρχει λόγος να δημιουργηθούν όλες αυτές οι μεταβλητές:

```
A = input('A:');
L = A; W = A/L;
L = (L+W)/2; W = A/L;
L = (L+W)/2; W = A/L;
L = (L+W)/2; W = A/L;
L = (L+W)/2; W = A/L;
```

Άρα: ανάγκη δομής που επαναλαμβάνει κώδικα για συγκεκριμένο αριθμό επαναλήψεων

→ Ο βρόχος **for**

```
for <τιμές μετρητή επανάληψης>
   εντολές που θα επαναληφθούν;
end
```

Συνήθης μορφή **for**:

```
for <μετ/τη> = <A.T.>:[Βήμα]:<T.T.>
   <εντολές>
end
```

Σημειολογία: < > : υποχρεωτικό μέρος
[ ] : προαιρετικό μέρος

Όταν το βήμα είναι μοναδιαίο:

```
for <μετ/τη> = <A.T.>:<T.T.>
     <εντολές>
end
```

Άρα το προηγούμενο παράδειγμα:

```
A = input('A:');
L = A; W = A/L;
for k = 1:4
   L = (L+W)/2;
   W = A/L;
end
```

και πιο γενικά

```
A = input('A:');
nSteps = input('nSteps:');
L = A; W = A/L;
for k = 1:nSteps;
   L = (L+W)/2;
   W = A/L;
end
```

Άλλα παραδείγματα:

```
for i=1:10
   disp(i);
end
```

```
for odd = 1:2:100
   disp(odd);
end
```

```
for even = 2:2:100
   disp(even);
end
```

```
for countdown = 10:-1:0
   disp(countdown);
end
```

```
for i=1:10
   x=rand;
   fprintf('%.3f\n',x);
end
```

Υπολογισμός αθροίσματος: 1+2+3+…+N:	Υπολογισμός γινομένου: 1*2*3*…*N:

`sum = 0;`
`for i=1:N`
`  sum = sum+i;`
`end`
`disp(sum);`
	`prod = 1;`
	`for i=1:N`
	`  prod = prod*i;`
	`end`
	`disp(prod);`

Πρόβλημα: Ένα κλαδί μοναδιαίου μήκους κόβεται σε τυχαίο σημεία. Τι μήκος θα έχει κατά μέσο όρο το μικρότερο κομμάτι;

Προσομοίωση: *η αναπαράσταση ενός πειράματος ή μιας φυσικής διεργασίας με τη χρήση ενός προγράμματος (μοντέλο).*

```
% μία δοκιμή του πειράματος
breakPt = rand;
if breakPt < 0.5
   shortPiece = breakPt;
else
   shortPiece = 1-breakPt;
end
```

```
breakPt = rand;
shortPiece = min(breakPt, 1-breakPt);
```

Αλγόριθμος λύσης του προβλήματος:

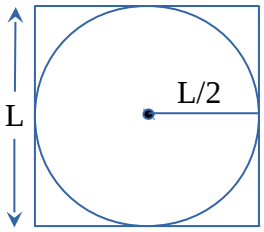
- Επανάληψη του πειράματος *n* φορές
- Υπολογισμός μέσου όρου
- Εμφάνιση αποτελέσματος

```
n = 10000; % αριθμός δοκιμών
total = 0; % τρέχον άθροισμα για το μήκος
for i=1:n
   breakPt = rand;
   shortPiece = min(breakPt, 1-breakPt);
   total = total+shortPiece;
end
avgLength = total/n;
fprintf('Το μέσο μήκος είναι %.2f\n', avgLength);
```

Παράδειγμα: μ.ο. 10 αριθμών από τον χρήστη:

```
n=10; total=0;
for k=1:n
   num = input('Δώσε έναν αριθμό:');
   total = total+num;
end
avg = total/n;
fprintf('Μέσος όρος: %.3%f\n', avg);
```

Πρόβλημα: Προσέγγιση του π με τη μέθοδο Monte Carlo



→ Ρίχνουμε *n* βελάκια στο τετράγωνο

→ hit: το βελάκι μέσα στον κύκλο

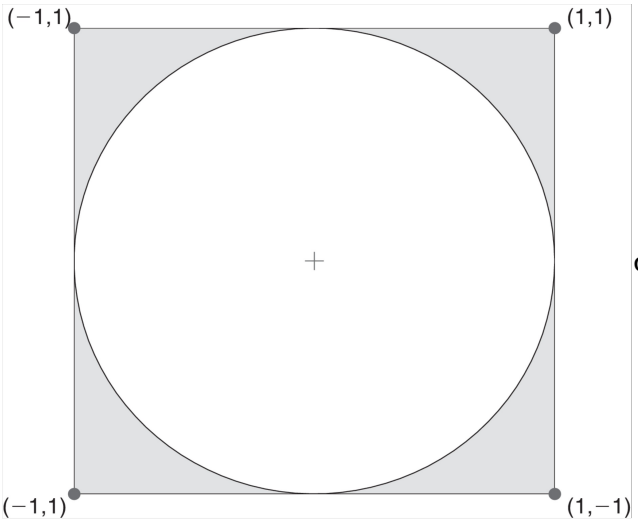
→ Το ποσοστό των hits προσεγγίζει τον λόγο των δύο εμβαδών:

$$\frac{hits}{n} \simeq \frac{E_{κύκλου}}{E_{τετραγώνου}} = \frac{\pi \frac{L^2}{4}}{L^2} \Rightarrow \pi \simeq 4\frac{hits}{n}$$

Άρα:

```
for i=1:n
   % ρίχνω το βελάκι i
   % αν το βελάκι είναι μέσα στον κύκλο:
   % hits = hits+1
end
myPi = 4*hits/n;
```

Ας υποθέσουμε ότι έχουμε τον μοναδιαίο κύκλο με κέντρο (0,0):



→ Ένα σημείο (x,y) είναι hit εάν $x^2+y^2 \le 1$

→ Πρέπει να βρεθεί τρόπος να παράγουμε τυχαία σημεία στο τετράγωνο.

Θέλουμε: -1<x<1 και -1<y<1, δηλαδή εύρος = 2 και διάστημα (-1, 1).

Η `rand` δίνει τυχαίους αριθμούς στο διάστημα (0, 1) (εύρος = 1).

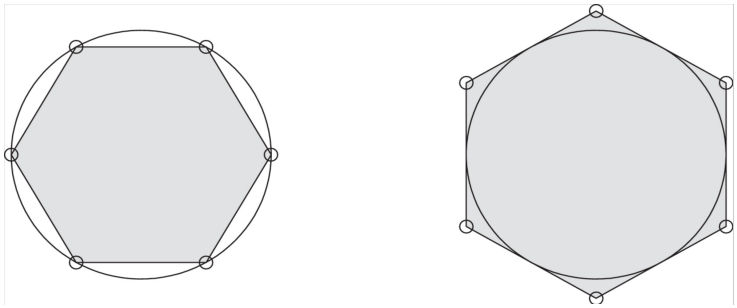
Με `rand` x 2 → εύρος 2, αλλά στο διάστημα (0, 2).

Άρα: `2*rand-1` → τυχαίοι αριθμοί στο (-1, 1).

To script:

```
n = 10000;
hits = 0;
for i=1:n
   %ρίχνω το βελάκι i
   x = 2*rand-1;
   y = 2*rand-1;
   % ελέγχω αν είναι hit
   if x^2 + y^2 <= 1
      hits = hits+1;
   end
end
myPi = 4*hits/n;
```

Πρόβλημα: Έστω *n* σημεία πάνω στον μοναδιαίο κύκλο.



$A_n = \frac{n}{2} \cdot \eta\mu\left(\frac{2\pi}{n}\right)$ $B_n = n \cdot \varepsilon\varphi\left(\frac{\pi}{n}\right)$

Όσο αυξάνεται το *n* (τα σημεία), το A_n αυξάνεται και το B_n μειώνεται και προσεγγίζουν το Ε του κύκλου, δηλαδή το π. Άρα, ο μέσος όρος τους αποτελεί μια καλή προσέγγιση του π:

$\pi \simeq \frac{A_n + B_n}{2}$, με το σφάλμα να εξαρτάται από την απόλυτη διαφορά των δύο εμβαδών.

Μας ενδιαφέρει ποια είναι η μικρότερη τιμή του *n* (n_*) για την οποία ικανοποιείται η σχέση:

$\left|A_{n_*} - B_{n_*}\right| \le \delta$, όπου δ η ανοχή σφάλματος π.χ. δ=0.00001.

<u>Αλγόριθμος:</u>

Δώσε την ανοχή σφάλματος δ και θέσε n=3.

Υπολόγισε τα A_3, B_3 και το όριο του σφάλματος B_3-A_3

Όσο το όριο του σφάλματος > δ, επανέλαβε:

Αύξησε το *n* κατά 1

Ενημέρωσε τα A_n και B_n και το όριο του σφάλματος.

Θέσε $n_* = n$ και εμφάνισε το $\frac{A_{n_*} + B_{n_*}}{2}$ (προσέγγιση του π).

Ο αλγόριθμος αυτός δεν μπορεί να υλοποιηθεί με `for` γιατί δεν είναι γνωστός εκ των προτέρων ο αριθμός των επαναλήψεων. Για τέτοιες επαναλήψεις υπάρχει ο βρόχος `while`.

→ Ο βρόχος **`while`**

```
while <λογική συνθήκη>
   <εντολές>
end
```

Σημαντικό: Πρέπει κάποια από τις εντολές να μεταβάλλει τη συνθήκη ώστε κάποτε να γίνεται false, διαφορετικά: ατέρμονας βρόχος.

To script:

```
delta = input('Δώσε το δ:');
n = 3;
A_n = (n/2)*sin(2*pi/n);
B_n = n*tan(pi/n);
errorBound = B_n - A_n;
while errorBound>delta
   n = n+1;
   A_n = (n/2)*sin(2*pi/n);
   B_n = n*tan(pi/n);
   errorBound = B_n - A_n;
end
nStar = n;
myPi = (A_n + B_n)/2;
fprintf(......);
```

Άλλα παραδείγματα:

- Άθροισμα άγνωστων πλήθους θετικών αριθμών:

```
sum = 0;
x = input('Δώσε θετικό αριθμό:');
while x>0
   sum = sum+x;
   x = input('Δώσε θετικό αριθμό ή μη-θετικό αριθμό για τερματισμό');
end
disp(sum);
```

→ Αν θέλαμε να εμφανίσουμε και πόσοι αριθμοί δόθηκαν;

- Άθροισμα ψηφίων ακεραίου:

Για τον ακέραιο *n*:

- Εύρεση τελευταίου ψηφίου: `rem(n,10)`
  π.χ. `rem(1975,10)` → `5`
- Αποκοπή τελευταίου ψηφίου: `fix(n/10)`
  π.χ. `fix(1975/10)` → `197`

```
n = input('Δώσε ακέραιο:');
dsum = 0;
while n>0   % αν βάζαμε n>=0: ατέρμονες επαναλήψεις
   dsum  =dsum+rem(n,10);
   n = fix(n/10);
end
fprintf('Το άθροισμα των ψηφίων του %g είναι %g\n', n,dsum);
```

Σχέση `for` και `while`:

`for μετ/τη = AT:B:TT`
`  εντολές;`
`end`
	`μετ/τη = AT;`
	`  while μετ/τη <= TT`
	`  εντολές;`
	`  μετ/τη = μετ/τη + B;`
	`end`

- Το `while` μπορεί να χρησιμοποιηθεί πάντα.
- Το `for` προτιμάται όταν το πλήθος των επαναλήψεων είναι γνωστό.
- Όταν το πλήθος των επαναλήψεων είναι άγνωστο: `while`

Π.χ., χρόνια για διπλασιασμό κεφαλαίου 1000 ευρώ με ετήσιο επιτόκιο 5%.

```
money = 1000;
years = 0;
while money < 2000
   money = money * 1.05;
   years = years + 1;
end
disp(years);
```

Πιο γενικό:

```
money = input('Κεφάλαιο?');
years = 0;
goal = 2*money;
while money < goal
  …
…
```

Η εντολή `break`

Π.χ. υπολογισμός γινομένου αγνώστου πλήθους αριθμών:

Χωρίς `break`

```
endOfData=0;
prod=1;
x=input('Δώσε αριθμό:');
while x~=endOfData
   prod=prod*x;
   x=input('Δώσε αριθμό ή...
         0 για τερματισμό');
end
disp(prod);
```

Με `break`

```
endOfData=0;
prod=1;
while true
   x=input('Δώσε αριθμό...
       ή 0 για τερματισμό');
   if x==endOfData
      break;
   end
   prod=prod*x;
end
disp(prod);
```

Η `break` τερματίζει τις επαναλήψεις `for` ή `while` που το εκτελούν.

Ένθετα `for` (nested)

```
N = input('N?');
for i=1:N
   for j=1:N
      if i<j
         fprintf('.');
      else
         fprintf('*');
      end
   end
fprintf('\n');
end
```

π.χ. για N=5:

```
 *   *   *   *   *

 .   *   *   *   *

 .   .   *   *   *

 .   .   .   *   *

 .   .   .   .   *
```

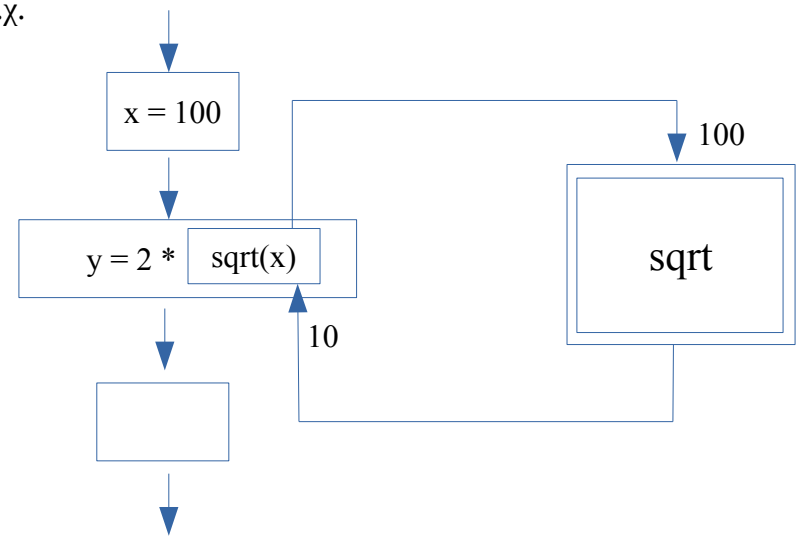
ΠΛΗΡΟΦΟΡΙΚΗ Ι (MATLAB)
Ενότητα 4

Σημειώσεις βασισμένες στο βιβλίο "Το MATLAB στην Υπολογιστική Επιστήμη και Τεχνολογία – Μια Εισαγωγή"

ΣΥΝΑΡΤΗΣΕΙΣ

Συνάρτηση ονομάζεται ένα τμήμα κώδικα (ή υποπρόγραμμα) το οποίο κάνει μια συγκεκριμένη διεργασία και μπορεί να καλείται από ένα πρόγραμμα (ή από μια άλλη συνάρτηση).

Π.χ.



Ουσιαστικά: x ⟶ sqrt ⟶ $\sqrt{x}$

(Η `sqrt` είναι μια εσωτερική συνάρτηση του MATLAB)

- Ένα script καλεί μια συνάρτηση
- Η ροή του προγράμματος μεταφέρεται προσωρινά στις εντολές της συνάρτησης
- Εκτελούνται οι εντολές της συνάρτησης
- Η ροή του προγράμματος επιστρέφει στο script

Όπως προαναφέρθηκε, μια συνάρτηση μπορεί να καλεί με τη σειρά της άλλες συναρτήσεις.

→ Ο τρόπος λειτουργίας μιας συνάρτησης δεν ενδιαφέρει τον χρήστη. Αυτό που τον ενδιαφέρει είναι ο τρόπος χρήσης της, δηλαδή:

- το όνομά της
- τι δέχεται
- τι επιστρέφει

→ Λόγοι ύπαρξης συναρτήσεων:

- Βοηθούν στη λογική σχεδίαση των προγραμμάτων
- Είναι «μαύρα κουτιά» που απλά χρησιμοποιούνται χωρίς να είναι απαραίτητη η γνώση της λειτουργίας τους
- Αποφυγή επανάληψης κώδικα στο ίδιο πρόγραμμα
- Επαναχρησιμοποίηση κώδικα σε άλλα προγράμματα
- Ευκολότερη τροποποίηση κώδικα

→ Είδη συναρτήσεων:

- Εσωτερικές συναρτήσεις του MATLAB (`sqrt`, `fprintf` κλπ)
- Ορισμένες από τον προγραμματιστή

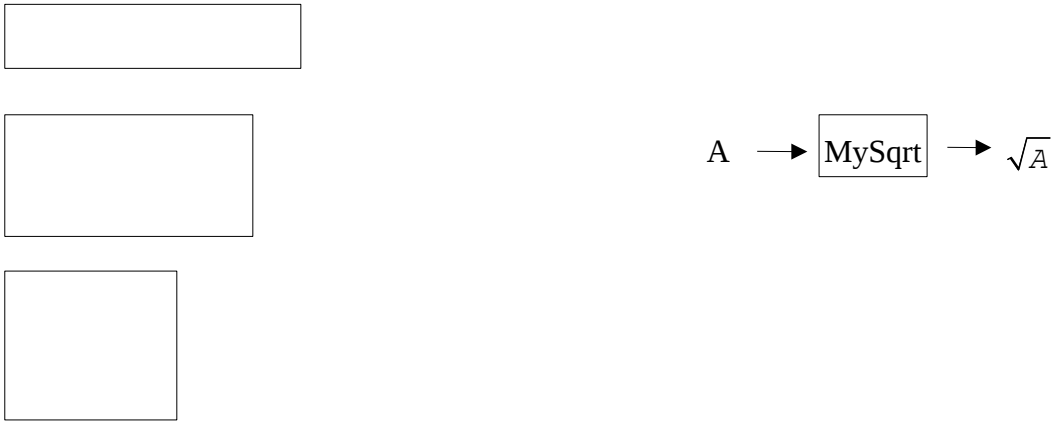
→ Σχεδιασμός συναρτήσεων:

- Όνομα
- Είσοδος / έξοδος ⟶ interface (περιβάλλον επικοινωνίας)
- Αλγόριθμος

Στο MATLAB:

```
function έξοδος = όνομα (είσοδος)   ⟶ επικεφαλίδα
% σχόλια περιγραφής                 ⟶ προδιαγραφές
<κώδικας>
   .
   .                                 } σώμα συνάρτησης
   .
```

►**Παράδειγμα:** Υλοποίηση της γεωμετρικής προσέγγισης της $\sqrt{A}$ με δημιουργία συνάρτησης.



Το script που είχε δημιουργηθεί (για δεδομένο Α):

```
nSteps = 10;
L = A;
W = A/L;
for i =1 : nSteps
      L = (L+W) / 2;
      W = A/L;
end
```

Όμως, αυτή η υλοποίηση δεν αντιμετωπίζει την περίπτωση Α=0.

Νέο script (για δεδομένο Α):

```
nSteps = 10;
if A==0
   s = 0;
else
   L = A;
   W = A/L;
   for i =1:nSteps
      L = (L+W) / 2;
      W = A/L;
   end
   s = L;
end
```

►Μετατροπή του script σε συνάρτηση:

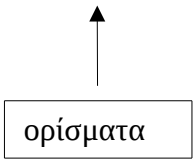
```
function s = MySqrt (A)
% Α ένας μη αρνητικός πραγματικός αριθμός και s μια προσέγγιση
% της τετραγωνικής του ρίζας
nSteps = 10;
if A==0
   s = 0;
else
   L = A;
   W = A/L;
   for i =1 : nSteps
      L = (L+W) / 2;
      W = A/L;
   end
   s = L;
end
```

Η συνάρτηση πρέπει να αποθηκευτεί στο αρχείο MySqrt.m

► Ας δούμε τώρα πως γίνεται η κλήση της συνάρτησης από ένα script:

```
x = 100;
y = MySqrt(x);
disp(y);
```

Γενικά, κλήση συνάρτησης: **`μεταβλητή = όνομα (είσοδος);`**



Άρα, γενικά:

- *Δημιουργία συνάρτησης:*

```
function[έξοδος1, έξοδος2, …] = όνομα (παράμετρος_εισόδου1,
                                        παράμετρος_εισόδου2, …)
```

- *Αποθήκευση:*

```
όνομα.m
```

- *Κλήση / εκτέλεση συνάρτησης:*

```
[έξοδος1, έξοδος2, …] = όνομα (όρισμα1, όρισμα2, …);
```

→ Όταν καλείται μια συνάρτηση, το `όρισμαΧ` παίρνει τη θέση της `παραμέτρου_εισόδουΧ`.

► Τοπικές μεταβλητές:

- Είναι οι μεταβλητές που ορίζονται (δημιουργούνται) μέσα σε κάποια συνάρτηση.
- Υπάρχουν μόνο μέσα στη συνάρτηση (Δημιουργούνται όταν καλείται η συνάρτηση / παύουν να υπάρχουν όταν η συνάρτηση επιστρέφει)

→ Οι παράμετροι εισόδου έχουν ουσιαστικά τα ίδια χαρακτηριστικά με τις τοπικές μεταβλητές:

- Δημιουργούνται όταν καλείται η συνάρτηση, μέσω της αρχικοποίησής τους με τις τιμές των ορισμάτων.
- Παύουν να υπάρχουν όταν η συνάρτηση επιστρέφει.

► "Υποσυναρτήσεις": συναρτήσεις που καλούνται από άλλες συναρτήσεις. Παράδειγμα:

Να γραφεί συνάρτηση binomial που να υπολογίζει τον διωνυμικό συντελεστή δύο μη αρνητικών ακεραίων n και k (όπου $n \geq k$):

$$\binom{n}{k} = \frac{n!}{k! \times (n-k)!}$$

Παρατηρούμε πως ο υπολογισμός παραγοντικού χρειάζεται τρεις φορές, άρα θα ήταν χρήσιμο να υλοποιηθεί σε μία «υποσυνάρτηση» την οποία θα χρησιμοποιεί η συνάρτηση `binomial`.

```
function f = fact(n)
f=1;
for i=2:n                          ⟶  fact.m
     f=f*i;
end
```

Οπότε, η συνάρτηση `binomial`:

```
function x = binomial (a,b)                    ⟶  binomial.m
x = fact(a) / (fact(b) * fact(a-b));
```

Κλήση:

```
>> binomial (10, 2)
     ans =
          45

>> a = binomial (10, 3)
      a =
         120
```

ΠΛΗΡΟΦΟΡΙΚΗ Ι (MATLAB)
Ενότητα 5

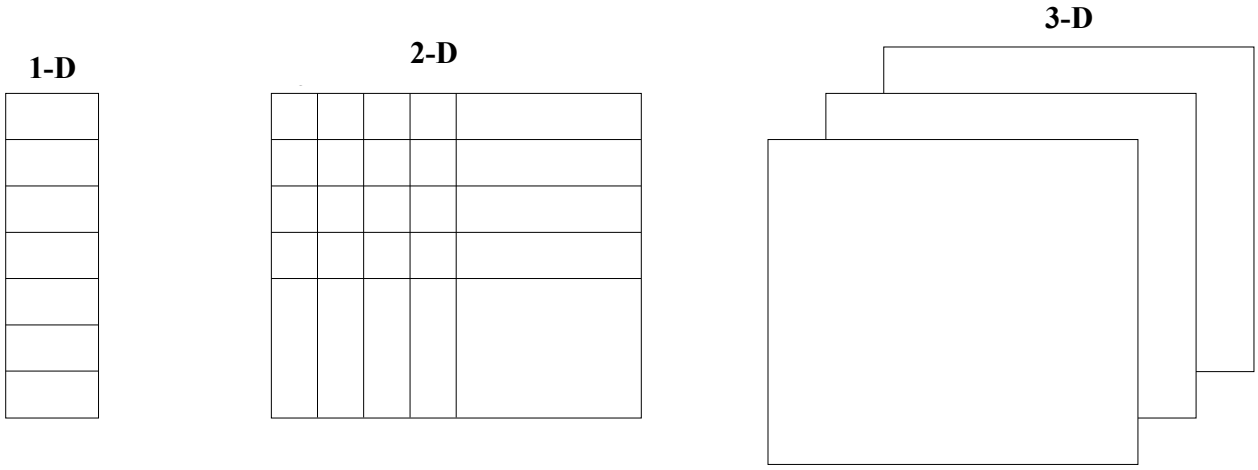
Σημειώσεις βασισμένες στο βιβλίο "Το MATLAB στην Υπολογιστική Επιστήμη και Τεχνολογία – Μια Εισαγωγή"

Πίνακες (Arrays) [1/2]

- Δομές δεδομένων για την αποθήκευση δεδομένων υπό ένα κοινό όνομα
- Η αποθήκευσή τους σε πίνακες βοηθάει κυρίως τη διαχείριση τους

Διάκριση πινάκων:

- Μονοδιάστατοι πίνακες (**1-D**)
- Δισδιάστατοι πίνακες (**2-D**)
- Τρισδιάστατοι πίνακες (**3-D**) κτλ.



Α) Μονοδιάστατοι πίνακες (Διανύσματα - Vectors):

Πρόκειται για διατεταγμένα στοιχεία ίδιου τύπου.
Έχουν **όνομα** και **μέγεθος** (πλήθος στοιχείων).

Π.χ.: Μέσες θερμοκρασίες ημέρας για μία εβδομάδα:

Δ	Τα	Τ	Π	Π	Σ	Κ
15	16.2	17.1	16	20.1	21.8	22

Αποθήκευση σε μονοδιάστατο πίνακα (διάνυσμα) Τ:

15	16.2	17.1	16	20.1	21.8	22
T(1)	**T(2)**	**T(3)**	**T(4)**	**T(5)**	**T(6)**	**T(7)**

- Οι θέσεις του διανύσματος είναι αριθμημένες (δείκτες)
- Οι δείκτες ξεκινούν από το 1.
- Το μέγεθος (μήκος ή πλήθος στοιχείων) δίνεται από την εντολή: `length(T)`
  (στην περίπτωσή μας `length(T)` → 7)

- **Δημιουργία διανυσμάτων:**

1) Άμεσα από τον προγραμματιστή:

α)
```
A = [10 20 30 40 50];
B = [3 2.5 4 0];
```

β) Με τον τελεστή : για πίνακες με ισαπέχουσες τιμές:

```
C = ΑρχικήΤιμή:Βήμα:ΤελικήΤιμή;
```

```
π.χ. C = 0:5:20; → C = [0 5 10 15 20];
D = 5:8; → D = [5 6 7 8];
```

2) Μέσω εσωτερικών συναρτήσεων του MATLAB:

```
x = rand(1,10) → διάνυσμα με 10 τυχαία στοιχεία
y = zeros(1,5) → διάνυσμα με 5 μηδενικά στοιχεία
z = ones(1,20) → διάνυσμα με 20 στοιχεία ίσα με 1
```

- **Πρόσβαση στα στοιχεία ενός διανύσματος:**

```
→ όνομα(δείκτης)
```

Ο δείκτης είναι πάντα **<u>ακέραια</u>** τιμή ή μεταβλητή ή αριθμητική έκφραση και δηλώνει τη θέση στο διάνυσμα (ξεκινώντας από το 1).

π.χ.:
```
>> A = [2 5 3 0 1];
>> A(2)  (Τυπώνει 5)
>> x = A(4);
>> disp(x); (Τυπώνει 0)
```

π.χ.: Εμφάνιση όλων των στοιχείων του Α:

```
for i = 1:length(A)
    fprintf('Το %go στοιχείο του Α είναι το %g.',i,A(i));
end
```

Τυπώνει:

```
Το 1ο στοιχείο του Α είναι το 2.
Το 2ο στοιχείο του Α είναι το 5.
Το 3ο στοιχείο του Α είναι το 3.
Το 4ο στοιχείο του Α είναι το 0.
Το 5ο στοιχείο του Α είναι το 1.
```

Είσοδος/Εξοδος διανυσμάτων:

- **Είσοδος (ανά στοιχείο):**

```
for i = 1:10
    message = sprintf('Δώσε το στοιχείο %g',i);
    A(i) = input(message);
end
```

Η συνάρτηση `sprintf` είναι ίδια με την `fprintf`, αλλά αποθηκεύει το αποτέλεσμά της σε μια συμβολοσειρά.

Μπορεί να χρησιμοποιηθεί για την εμφάνιση "μορφοποιημένου" κειμένου στην εντολή `input`, όπως παραπάνω.

- **Είσοδος ολόκληρου πίνακα:**

```
A = input('Δώσε τον πίνακα A');
```

Ο χρήστης δίνει: >>[5 15 20 0 4]

- **Έξοδος (ανά στοιχείο):**

```
for i = 1:10
    fprintf(A(%g) = %g\n',i,A(i));
    %ή πιο απλά: disp(A(i));
end
```

- **Έξοδος ολόκληρου πίνακα:** `disp(A);`

Παραδείγματα διανυσμάτων:

- **Μέση τιμή θερμοκρασιών και θερμοκρασιακές διαφορές:**

```
T = input('Δώσε τον πίνακα θερμοκρασιών: ');
sum = 0;
for i = 1:length(T)
    sum = sum + T(i);
end
mT = sum/length(T);
fprintf('Η μέση θερμοκρασία είναι %g.\n',mT);
for i = 1:length(T)
    DT(i) = T(i) - mT;
end
disp('Θερμοκρασιακές διαφορές: ');
disp(DT);
```

- **Μέγιστο κι ελάχιστο διανύσματος:**

```
list = input('Δώσε το διάνυσμα: ');
small = list(1);
large = list(1);
for i = 2:length(list)
    if list(i)<small
       small = list(i);
    else
       if list(i)>large
          large = list(i);
       end
    end
end
```

Γιατί δεν χρησιμοποιήθηκαν 2 ανεξάρτητα `if`;

- **Εσωτερικό γινόμενο διανυσμάτων:**

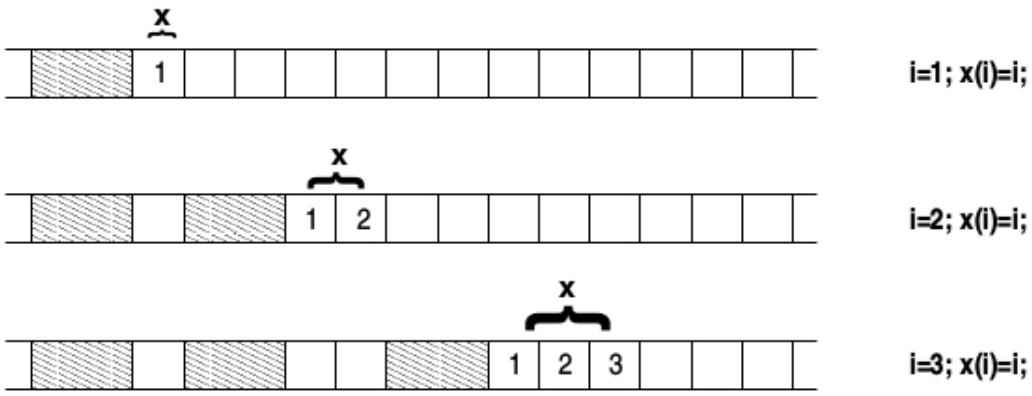
```
% έστω διανύσματα x, y ίδιου μήκους
p = 0;
for i = 1:length(x)
    p = p + x(i)*y(i);
end
```

- **Αρχικοποίηση διανυσμάτων:**

α)

```
for i = 1:100
   x(i) = i;
   %εντολές που δεσμεύουν μνήμη
end
```

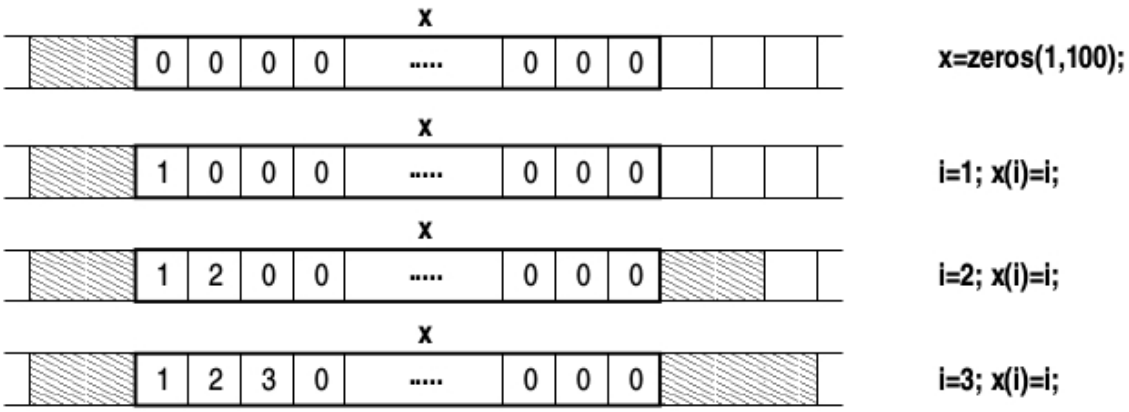
Εικόνα της μνήμης:



β) (με δημιουργία ολόκληρου του διανύσματος πριν την αρχικοποίηση)

```
x = zeros(1,100);
for i = 1:100
   x(i) = i;
   %εντολές που δεσμεύουν μνήμη
end
```

Εικόνα της μνήμης:



- **Ταξινόμηση διανύσματος με τον αλγόριθμο της επιλογής (selection sort):**

1. Ξεκίνα από την πρώτη θέση.
2. Βρες το ελάχιστο στοιχείο από τη θέση αυτή και μετά.
3. Ενάλλαξε (swap) τα στοιχεία της θέσης που ξεκίνησες και της θέσης του ελάχιστου.
4. Πήγαινε στην επόμενη θέση και επανάλαβε το “Βήμα 2”, μέχρι να φτάσεις στην προτελευταία θέση.

Υλοποίηση με συνάρτηση:

```
function v = selection(v)
for i = 1:length(v)-1
    minIndex = i; % έστω ότι το στοιχείο αυτό είναι το ελάχιστο
    %αναζήτηση του ελάχιστου:
    for j = i+1:length(v)
        if v(j)<v(minIndex)
           minIndex = j; % η θέση του ελάχιστου στοιχείου
        end
    end
    if minIndex ~= i; % αν βρέθηκε κάποιο μικρότερο στοιχείο
       temp = v(i);
       v(i) = v(minIndex);
       v(minIndex) = temp;
    end
end
```

- **Ταξινόμηση διανύσματος με τον αλγόριθμο εισαγωγής (insertion sort):**

1. Ξεκίνα από το δεύτερο στοιχείο.
2. Σύγκρινε το στοιχείο με ένα-ένα τα στοιχεία που βρίσκονται αριστερά του **μέχρι να βρεις** (άρα while-loop) κάποιο μικρότερό το κι όσο βρίσκεις κάποιο μεγαλύτερό του, ενάλλασσέ τα.
3. Πήγαινε στο επόμενο στοιχείο και επανάλαβε το “Βήμα 2”.

Υλοποίηση με συνάρτηση:

```
function v = insertion(v)
for i = 2:length(v)
    j = i;
    while j>1 && v(j-1)>v(j)
       temp = v(i);
       v(j) = v(j-1);
       v(j-1) = temp;
       j = j - 1;
    end
end
```

- **Γραμμική αναζήτηση (linear search):**

→ Ξεκινώντας από το πρώτο στοιχείο, σύγκρινε ένα-ένα τα στοιχεία του διανύσματος με το προς αναζήτηση στοιχείο (**key**), μέχρι να το βρεις.

Υλοποίηση με συνάρτηση:

```
function pos = linearSearch(v,key)
loc = 1;     % η τρέχουσα θέση αναζήτησης
pos = -1;    % η θέση του στοιχείου αναζήτησης
             % Θα επιστρέψει -1 αν δε βρεθεί
hit = false; % flag για το αν έχει βρεθεί το προς αναζήτηση στοιχείο
             % (δεν είναι απαραίτητο)
while loc<=length(v) && ~hit
      if v(loc)==key
         pos = loc;
         hit = true;
      else
         loc = loc + 1;
      end
end
```

- **Δυαδική αναζήτηση (binary search):**

→ Προϋπόθεση: Η λίστα πρέπει να είναι ταξινομημένη!

1. Σύγκρινε το αναζητούμενο στοιχείο με το μεσαίο στοιχείο της λίστας
2. Όσο αυτά διαφέρουν και υπάρχουν ακόμα στοιχεία στη λίστα:
   - αν είναι μικρότερο του στοιχείου, επανάλαβε το “Βήμα 1” για το αριστερό τμήμα της λίστας.
   - αν είναι μεγαλύτερο του στοιχείου, επανάλαβε το “Βήμα 1” για το δεξί τμήμα της λίστας.

Υλοποίηση με συνάρτηση:

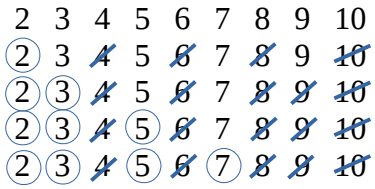
```
function pos = binarySearch(v,key)
left = 1;          % το αριστερό άκρο της τρέχουσας λίστας
right = length(v); % το δεξί άκρο της τρέχουσας λίστας
pos = -1;
hit = false;
while left<=right && ~hit
      mid = fix((left+right)/2);
      if v(mid)==key
         pos = mid;
         hit = true;
      elseif key<v(mid)
         right = mid - 1;
      else
         left = mid + 1;
      end
end
```

- **Το κόσκινο του Ερατοσθένη (3ος αιώνας π.Χ.):**

Αλγόριθμος για τη εύρεση των πρώτων αριθμών από το 2 έως το N.

1. Γράψε όλους τους αριθμούς από το 2 έως το N.
2. Κύκλωσε τον πρώτο διαθέσιμο αριθμό (θα είναι πρώτος).
3. Διέγραψε τα πολλαπλάσιά του.
4. Πήγαινε στο “Βήμα 2”.

π.χ.: για N=10 έχουμε:



Υλοποίηση με συνάρτηση:

```
function primes = eratosthenis(N)
P = 1:N   % η λίστα των αριθμών
P(1) = 0; % Όσοι διαγράφονται θα γίνονται 0. Το 1 ΔΕΝ είναι πρώτος.
for i = 2:sqrt(N)
    if P(i)            % αν το i δεν έχει διαγραφεί
       for j = 2*i:i:N % πήγαινε σε κάθε πολλαπλάσιο του i
           P(i) = 0;   % και διέγραψέ το
       end
    end
end
% Μέτρημα των πρώτων αριθμών που βρέθηκαν
NP = 0;
for i = 2:N
    if P(i)
       NP = NP + 1;
    end
end

primes = zeros(1,NP);
k = 0;
for i = 2:N
    if P(i)
       k = k + 1;
       primes(k) = i;
    end
end
```

- **Δημιουργία διανύσματος με τη συνάρτηση `linspace`**

```
linspace(Αρχική_Τιμή, Τελική_Τιμή, πλήθος_στοιχείων)
```

Π.χ., `x = linspace(0, 2*pi, 9);`

→ `[0  0.7854  1.5708  2.3562  3.1416  3.9270  4.7124  5.4978  6.2832]`

- **Πράξεις με διανύσματα:**

Έστω τα διανύσματα:

```
a = [10 8 -5]
b = [2 4 1]
```

```
c = 5*a   →    c: [50 40 -25]
c = a/2   →    c: [5 4 -2.5]
```

```
c = -a    →    c: [-10 -8 5]
c = 1./a  →    c: [.100 .125 -.200]
c = a+5   →    c: [15 13 0]
c = a.^2  →    c: [100 64 25]
c = a+b   →    c: [12 12 -4]
c = a-b   →    c: [8 4 -6]
c = a.*b  →    c: [20 32 -5]
c = a./b  →    c: [5 2 -5]
```

Έστω το διάνυσμα `v = [2 5 7]`

```
v(4) = 8;          →    v: [2 5 7 8]
v(6) = 10;         →    v: [2 5 7 8 0 10]
v(2:4)             →    [5 7 8]
v([4 6])           →    [8 10]
[v(1:2) v([4 6])]  →    [2 5 8 10]
```

- **Διανύσματα και γραφικές παραστάσεις:**

Να γραφεί πρόγραμμα που να εμφανίζει τη γραφική παράσταση της συνάρτησης

$f(x)=\frac{\eta\mu(5x)\cdot e^{-\frac{x}{2}}}{1+x^2}$ στο διάστημα [-2, 3].

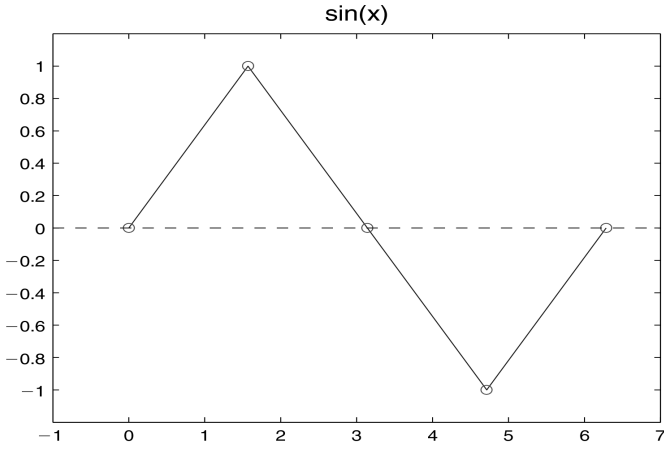
→ Ένα πιο απλό πρόβλημα: σχεδίαση του ημ(x) στο [0, 2π].

“Με το χέρι”, θα παράγαμε έναν πίνακα τιμών, π.χ.:

x	0.000	1.571	3.142	4.712	6.283
$\sin(x)$	0.000	1.000	0.000	−1.000	0.000

Στη συνέχεια θα συνδέαμε τα αντίστοιχα 5 σημεία:

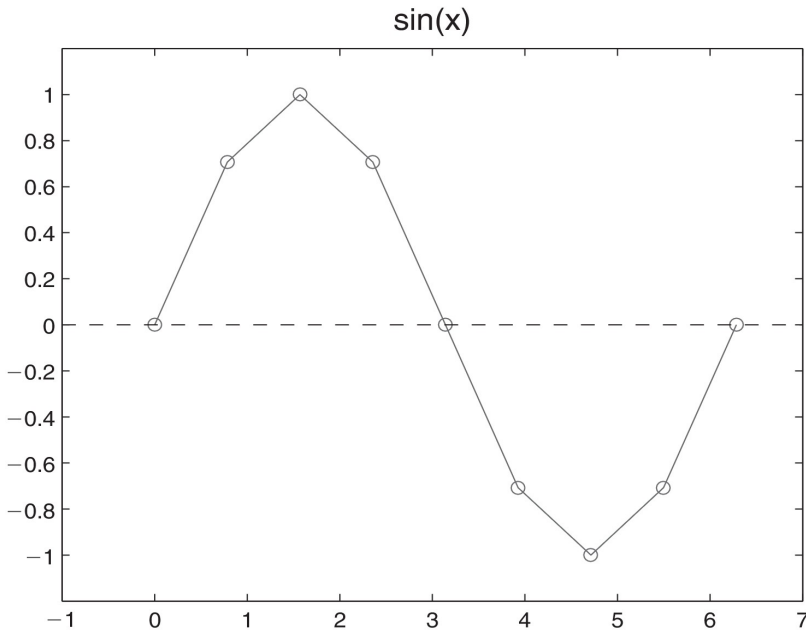
(0, 0), (1.571, 1), (3.142, 0), (4.712, -1), (6.283, 0)



Αν δημιουργήσουμε 9 δειγματοληπτικά σημεία αντί για 5:

x	0.000	0.785	1.571	2.356	3.142	3.927	4.712	5.498	6.283
$\sin(x)$	0.000	0.707	1.000	0.707	0.000	−0.707	−1.000	−0.707	−0.000

θα προκύψει το εξής:



Για τη γραφική αποτύπωση μιας συνάρτησης *y = f(x)* σε ένα διάστημα [*L*, *R*]:

1. Δημιουργία πίνακα με τις τιμές του *x* από το δεδομένο διάστημα.
2. Δημιουργία πίνακα με τις τιμές του *y* που αντιστοιχούν (μέσω της *f*) στις τιμές του *x*.
3. Συνένωση των σημείων που ορίζονται από τα ζεύγη *xy* και εμφάνιση της γραμμής που προκύπτει.

```
x = linspace(0,2*pi,9);
y = sin(x);
plot(x,y);
```

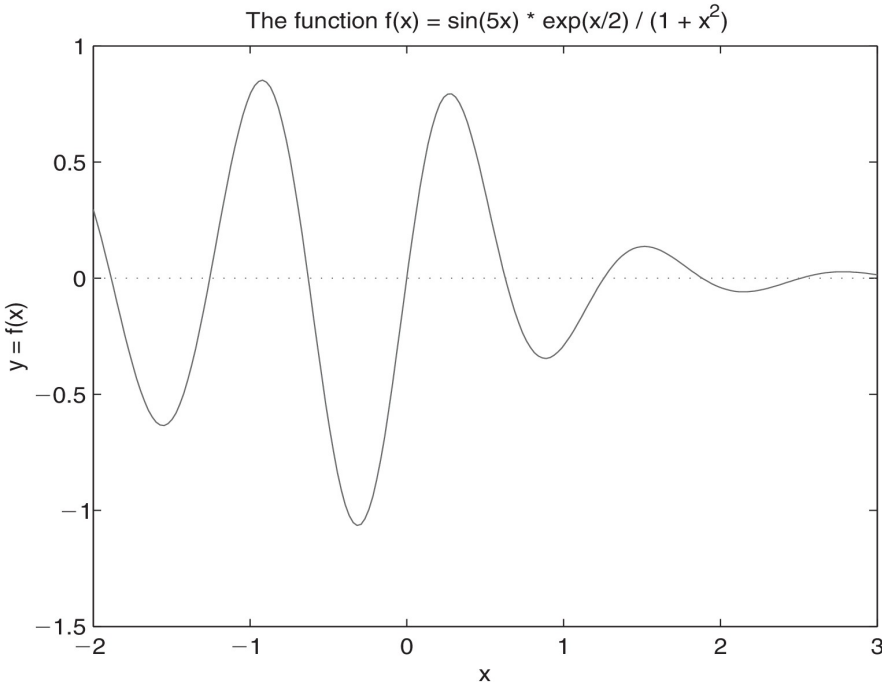
Π.χ., το ακόλουθο script σχεδιάζει το ημ(*x*) για διάφορα πλήθη δειγματοληπτικών σημείων:

```
for n = 25:25:500
   x = linspace(0,2*pi,n);
   y = sin(x);
   plot(x,y);
   title(sprintf('n = %3d', n));
   pause;
end
```

→ Script για τη σχεδίαση της αρχικής συνάρτησης:

```
x = linspace(2,3,100);
y1 = 5*x;
y2 = sin(y1);
y3 = -x;
y4 = y3/2;
y5 = exp(y4);
y6 = y2.*y5;
y7 = x.^2;
y8 = 1+y7;
y = y6./y8;
plot(x,y);
```

} `y = sin(5*x).*exp(-x/2)./(1+x.^2);`



Πολλαπλά γραφήματα: `plot(x,y1,x,y2,x,y3,...)`

Η συνάρτηση `plot` με παραμέτρους μορφοποίησης:

`plot(x,y,'XYZ')`
- `X`: τύπος γραμμής
- `Y`: τύπος σημείων
- `Z`: χρώμα σημείων και γραμμής

Επιλογές:

`X: - : -. --`
`Y: . o + x *`
`Z: b` (μπλε) `w` (άσπρο) `c` (γαλάζιο) `m` (μωβ) `r` (κόκκινο) `y` (κίτρινο) `g` (πράσινο) `k` (μαύρο)

π.χ., `plot(x,y,'-or');` → κόκκινη συνεχής γραμμή με σημεία-κύκλους

Βασικές εντολές γραφήματος:

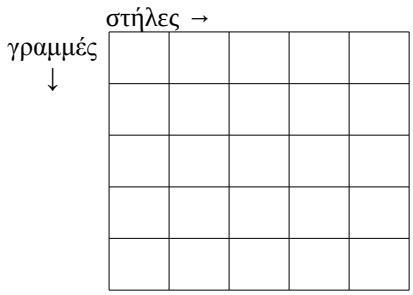
Τίτλος γραφήματος: `title('......');`
Τίτλος άξονα x: `xlabel('......');`
Τίτλος άξονα y: `ylabel('......');`

ΠΛΗΡΟΦΟΡΙΚΗ Ι (MATLAB)
Ενότητα 6

Σημειώσεις βασισμένες στο βιβλίο "Το MATLAB στην Υπολογιστική Επιστήμη και Τεχνολογία – Μια Εισαγωγή"

Πίνακες (Arrays) [2/2]

Β) Δισδιάστατοι πίνακες



Δημιουργία πινάκων:

- `A = [1 2 3; 4 5 6];` → $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

- `B = [1 2 3`
  `4 5 6];` → $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

- `C(3,2) = 5;` → $\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 5 \end{bmatrix}$

- `D1 = ones(4,3);` → $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

- `D2 = rand(10,5);`
  `...`

Πρόσβαση σε συγκεκριμένο στοιχείο ενός πίνακα:

```
όνομα(γραμμή , στήλη)
```

π.χ.: `B(2,1)` → 4

Αναφορά σε ολόκληρες γραμμές ή στήλες:

`B(:,2)` → $\begin{bmatrix}2\\5\end{bmatrix}$ (δεύτερη στήλη)

`B(1,:)` → `[1 2 3]` (πρώτη γραμμή)

Αναφορά σε συγκεκριμένα τμήματα ενός πίνακα:

π.χ.:

```
E = [ 1  2  3  4  5
      6  7  8  9 10
     11 12 13 14 15
     16 17 18 19 20];
```

`E(2:3,3:4)` → $\begin{bmatrix}8 & 9\\13 & 14\end{bmatrix}$

Ακόμα και σε μη-συνεχόμενα στοιχεία, π.χ. `E(:,[2,4])` → $\begin{bmatrix}2 & 4\\7 & 9\\12 & 14\\17 & 19\end{bmatrix}$

Είσοδος στοιχείο ανά στοιχείο:

```
for i = 1:10
    for j = 1:5
    A(i,j) = input('Δώσε στοιχείο:');
    end
end
```

Είσοδος ολόκληρου πίνακα:

```
A = input('Δώσε τον πίνακα A'); → […; …; …];
```

Η συνάρτηση `size`:

Αν `A` δισδιάστατος πίνακας p × q, τότε:

```
[a, b] = size(A);  → a = p, b = q
size(A,1)  → p (γραμμές)
size(A,2)  → q (στήλες)
```

`A = [ones(1,3); 1:3]` → $\begin{bmatrix}1 & 1 & 1\\ 1 & 2 & 3\end{bmatrix}$

`B = [ones(1,3); 1:4]` → ΛΑΘΟΣ!

`A(1,4) = 5;` → $\begin{bmatrix}1 & 1 & 1 & 5\\ 1 & 2 & 3 & 0\end{bmatrix}$

`A(2,7) = 8;` → $\begin{bmatrix}1 & 1 & 1 & 5 & 0 & 0 & 0\\ 1 & 2 & 3 & 0 & 0 & 0 & 8\end{bmatrix}$

Παραδείγματα:

- Υπολογισμός μέσου όρου γραμμής:

```
function v = avgRows(A)
% Συνάρτηση που επιστρέφει διάνυσμα με τους μέσους όρους
% των στοιχείων της κάθε γραμμής δισδιάστατου πίνακα.

[r, c] = size(A);
v = zeros(r,1); % διάνυσμα-στήλη για τους μέσους όρους
for i = 1:r
   sum = 0;
   for j = 1:c
      sum = sum + A(i,j);
   end
   v(i) = sum/c;
end
```

- Πολλαπλασιασμός πίνακα με διάνυσμα

```
function [y, err] = matvec(A, x)
% Υπολογισμός y = Ax με A ∈ R_m×n , x ∈ R_n , y ∈ R_m

[m, n] = size(A);
err = 0;
if size(x,1)~= n, err = 1; return; end;
y = zeros(m,1);
for i = 1:m
   sum = 0;
   for j = 1:n
      sum = sum + A(i,j)*x(j);
   end
   y(i) = sum;
end
```

- Γινόμενο πινάκων

```
function [C, err] = matmat(A, B)
% Υπολογισμός: C = AB με A ∈ R_m×n , B ∈ R_n×k , C ∈ R_m×k

[m, n] = size(A);
k = size(B,2);
err = 0;
if size(B,1)~=n, err=1; return; end;
C = zeros(m,k);
for i = 1:m
   for j = 1:k
      sum = 0;
      for l = 1:n
         sum = sum + A(i,l)*B(l,j);
      end
      C(i,j) = sum;
   end
end
```

Πρόβλημα: Το έτος Υ, έστω τέσσερα νησιά S_1 , S_2 , S_3 και S_4 με ένα εκατομμύριο κατοίκους το καθένα, καθώς και ο πίνακας πιθανοτήτων μετάβασης:

Πρός: \ Από:	S_1	S_2	S_3	S_4
S_1	0.32	0.17	0.11	0.46
S_2	0.18	0.43	0.32	0.33
S_3	0.27	0.22	0.39	0.14
S_4	0.23	0.18	0.18	0.07

Να υπολογιστεί η κατανομή του πληθυσμού το έτος Υ+5.

<u>Λύση:</u>

Μετά από ένα χρονικό βήμα, ο πληθυσμός π.χ. του S_1 είναι:

$$\Nέος\Πλ_{S_1} = P_{S_1 \text{-->} S_1} * \Πλ_{S_1} + P_{S_2 \text{-->} S_1} * \Πλ_{S_2} + P_{S_3 \text{-->} S_1} * \Πλ_{S_3} + P_{S_4 \text{-->} S_1} * \Πλ_{S_4}$$

Διάνυσμα με τιμές πληθυσμών στην αρχή κάποιου χρονικού βήματος:

`x`: 4×1, όπου `x(i)` ο πληθυσμός του S_i στην αρχή και
`y`: 4×1, όπου `y(i)` ο πληθυσμός του S_i στο τέλος ενός χρονικού βήματος

Ο πίνακας μετάβασης μπορεί να αποθηκευτεί σε έναν 2-D πίνακα:

```
P = [0.32 0.17 0.11 0.46; 0.18 0.43 0.32 0.33;
     0.27 0.22 0.39 0.14; 0.23 0.18 0.18 0.07];
```

→ Για τον νέο πληθυσμό κάθε νησιού:

```
for i=1:4
    y(i) = P(i,1)*x(1)+P(i,2)*x(2)+P(i,3)*x(3)+P(i,4)*x(4);
end
```

Ο παραπάνω αλγόριθμος, αν εκτελεστεί 5 φορές θα μας δώσει τους ζητούμενους πληθυσμούς μετά από 5 έτη.

Αν είχαμε γενικά *n* νησιά:

```
for i=1:n
    %υπολογισμός του y(i)
end
```

Αντιγραφή της γραμμής *i* του `P` σε ένα διάνυσμα `r`:

```
for j=1:n
    r(j) = P(i,j);
end
```

Υπολογισμός αθροίσματος:

```
s = 0;
for j = 1:n
    s = s + r(j)*x(j);
end
y(i) = s;
```

```
y(i) = 0;
for j = 1:n
    y(i) = y(i) + P(i,j)*x(j);
end
```

και για τα *n* νησιά:

```
for i = 1:n
    y(i) = 0;
    for j = 1:n
        y(i) = y(i) + P(i,j)*x(j);
    end
end
```

Επειδή η διαδικασία αυτή θα πρέπει να επαναλαμβάνεται για κάθε έτος, μπορεί να αποτελέσει μια συνάρτηση:

```
function y = Transition(P,x)
[n,n] = size(P);
y = zeros(n,1);
for i = 1:n
    for j = 1:n
        y(i) = y(i) + P(i,j)*x(j);
    end
end
```

Οπότε, ο υπολογισμός των πληθυσμών για N βήματα:

```
for k = 1:N
    x = Transition(P,x);
end
```

ΠΛΗΡΟΦΟΡΙΚΗ Ι (MATLAB)
Ενότητα 7

Σημειώσεις βασισμένες στο βιβλίο "Το MATLAB στην Υπολογιστική Επιστήμη και Τεχνολογία – Μια Εισαγωγή"

Αναδρομή (Recursion)

Αναδρομή: όταν μία συνάρτηση καλεί τον εαυτό της.

Πρόγραμμα για τον υπολογισμό του *n*!

i) Χωρίς αναδρομή (fact1.m):

```
function nfact = fact1(n)
nfact = 1;
for i=2:n
   nfact = nfact*i;
end
```

ii) Με αναδρομή (fact2.m):

```
function nfact = fact2(n)
if n==0
   nfact = 1;
else
   nfact = n*fact2(n-1);
end
```

- Βασική περίπτωση: επιστρέφει τιμή χωρίς αναδρομική κλήση
- Αναδρομικό βήμα: `n*fact2(n-1)`

➢ Η ακολουθία των αναδρομικών κλήσεων (άρα των ορισμάτων των κλήσεων αυτών) θα πρέπει να συγκλίνει στη βασική περίπτωση (εδώ, το `n` να γίνει τελικά 0).

Όταν π.χ. καλέσουμε την `fact2` για τον αριθμό 5:

```
>> x = fact2(5);
```

γίνεται το εξής:

```
fact2(5)
   fact2(4)
      fact2(3)
         fact2(2)
            fact2(1)
               fact2(0)
                  nfact = 1  →  επιστρέφεται στην κλήση fact2(1)
            nfact = 1*1 = 1  →  επιστρέφεται στην κλήση fact2(2)
         nfact = 2*1 = 2  →  κτλ..
      nfact = 3*2 = 6
   nfact = 4*6 = 24
nfact = 5*24 = 120
```

Κάποια προβλήματα της αναδρομής:

- Υπολογισμός του $H_n = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$ (n-αρμονικός)

```
function h = harm1(n)
if n==1
   h = 1;
else
   h = harm1(n-1) + 1/n;
end
```

→ ΠΡΟΒΛΗΜΑ: Μεγάλες απαιτήσεις μνήμης. Ο αντίστοιχος επαναληπτικός αλγόριθμος χρειάζεται ελάχιστη μνήμη:

```
function h = harm2(n)
h = 1;
for i = 1:n
   h = h + 1/i;
end
```

- Υπολογισμός αριθμών Fibonacci

  $F_n = F_{n-1} + F_{n-2}$, για $n>2$, με $F_1 = 1$ και $F_2 = 2$.

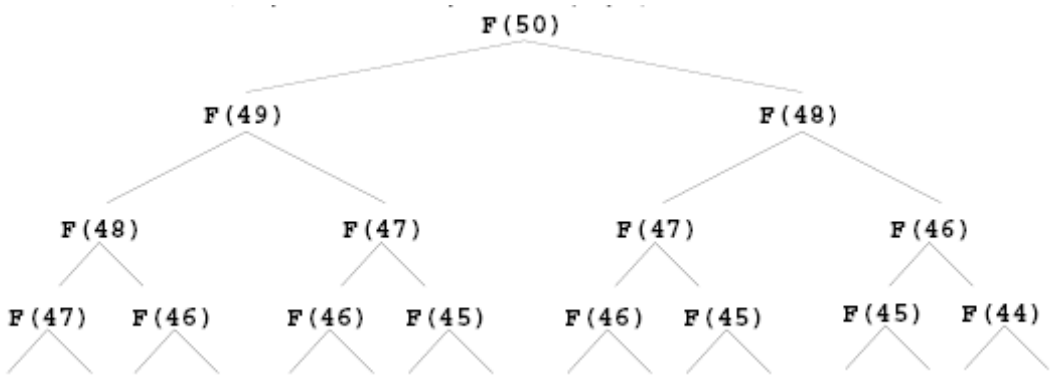
```
function f = fib1(n)
if n==1
   f = 1;
elseif n==2
   f = 2;
else
   f = fib1(n-1)+fib1(n-2);
end
```

ή:

```
if n<=2
   f = n;
```

→ ΠΡΟΒΛΗΜΑ: Περιττή επανάληψη υπολογισμών.

Π.χ., για τον υπολογισμό του F_{50}:



Η `fib1(1)` καλείται πάνω από 20 δις φορές!...

Άρα, ο επαναληπτικός αλγόριθμος είναι πολύ πιο αποδοτικός:

```
function f = fib2(n)
x = 1;
y = 2;
for i = 1:n
   f = x;
   x = y;
   y = f + x;
end
```

ΠΛΗΡΟΦΟΡΙΚΗ Ι (MATLAB)
Ενότητα 8

Σημειώσεις βασισμένες στο βιβλίο "Το MATLAB στην Υπολογιστική Επιστήμη και Τεχνολογία – Μια Εισαγωγή"

Συστήματα αρίθμησης

Bits & Bytes

Bit: η μικρότερη μονάδα πληροφορίας
- μία από δύο πιθανές καταστάσεις
  (ναι / όχι, αληθές / ψευδές, on / off)

➢ κωδικοποίηση σε 0 ή 1 → δυαδικό σύστημα

- Το δυαδικό σύστημα επιτρέπει την κατασκευή εξαρτημάτων που έχουν 2 μόνο καταστάσεις
  - υψηλή / χαμηλή τάση (chips)
  - φορά μαγνητικού πεδίου (δίσκοι)
  - ανάκλαση ή όχι φωτεινής δέσμης (CD-ROM, DVD κτλ)
  - παρουσία ή όχι ηλεκτρικού φορτίου (flash)

➢ Η πληροφορία αποθηκεύεται και επεξεργάζεται σαν bits.

Byte: 8 bits που λειτουργούν ως μία ενιαία μονάδα

Δεκαδικό σύστημα

- Βάση το 10
- 10 ψηφία: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - ➢ ένα δεκαδικό ψηφίο εκφράζει μία από δέκα πιθανές επιλογές (μετράει 10 πράγματα, έχει 10 διακριτές τιμές)
  - ➢ δύο δεκαδικά ψηφία εκφράζουν μία από 100 πιθανές επιλογές
  - ➢ κτλ.

π.χ., $1975 = 1\times1000 + 9\times100 + 7\times10 + 5\times1$

$= 1\times10^3 + 9\times10^2 + 7\times10^1 + 5\times10^0$

Δυαδικό σύστημα

- Βάση το 2
- 2 ψηφία: 0, 1 (bits)
  - ένα δυαδικό ψηφίο εκφράζει μία από δύο πιθανές επιλογές (μετράει 2 πράγματα, έχει 2 διακριτές τιμές)
  - δύο δυαδικά ψηφία εκφράζουν μία από 4 πιθανές επιλογές
  - κτλ.

π.χ., $11001_2 = 1\times16 + 1\times8 + 0\times4 + 0\times2 + 1\times1$

$= 1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$ = 25 (στο δεκαδικό)

Δεκαδικοί σε δυαδικούς

- Ανάλυση σε δυνάμεις του 2:

π.χ., $42 = 32 + 10 = 32 + 8 + 2 = 2^5 + 2^3 + 2^1 = 101010_2$

- Διαδοχικές διαιρέσεις με 2 (μόνο για ακέραιους)

Πηλίκο	**42**	21	10	5	2	1	0
Υπόλοιπο		0	1	0	1	0	1

→ **101010_2**

Οι δυαδικοί αριθμοί είναι «ογκώδεις»

- **Οκταδικοί**: 0, 1, 2, 3, 4, 5, 6, 7
- **Δεκαεξαδικοί**: 0-9, A, B, C, D, E, F

2-δικό – 8-δικό: κάθε δυαδική *τριάδα* ↔ ένα οκταδικό στοιχείο

001 100 101 001 000 . 101 101 100
1 4 5 1 0 . 5 5 4

2-δικό – 16-δικό: κάθε δυαδική *τετράδα* ↔ ένα δεκαεξαδικό στοιχείο

0111 1011 1010 0011 . 1011 1100 0100
7 B A 3 . B C 4

Αριθμητική στο δυαδικό σύστημα (γενικά)

Η πρόσθεση στηρίζεται στους κανόνες: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, και 1 + 1 = 10

```
π.χ.,     1010          01101
        + 1100        + 10111
        ------        -------
         10110         100100
```

Ο πολλαπλασιασμός γίνεται όπως και στο δεκαδικό σύστημα:

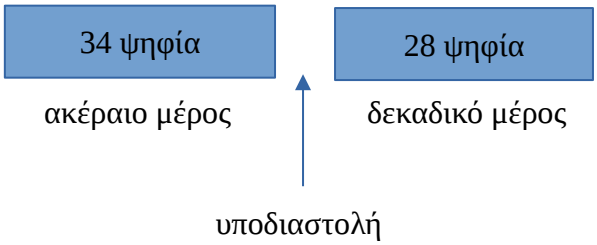
```
          1001
        x 1010
        ------
          0000
         1001
        0000
      + 1001
      ---------
        1011010
```

Αναπαράσταση πραγματικών αριθμών

- Υπολογισμοί με πολύ μεγάλους και πολύ μικρούς αριθμούς, π.χ.:
  - μάζα ηλεκτρονίου: 9×10^{-28} gr
  - μάζα ήλιου: 2×10^{33} gr

Αναπαράσταση σταθερής υποδιαστολής (fixed-point):

Για τους παραπάνω αριθμούς, απαιτούνται:



- Ανάγκη για σύστημα αναπαράστασης όπου το εύρος των αριθμών είναι ανεξάρτητο από το πλήθος των σημαντικών ψηφίων: **αναπαράσταση κινητής υποδιαστολής (floating-point)**

Αναπαράσταση κινητής υποδιαστολής:

- Από εκθετική μορφή:

$$n = f \times 10^{e}$$

με $0.1 \le f < 1$ ή $f = 0$

όπου f κλασματικό μέρος (mantissa), e εκθέτης (exponent)

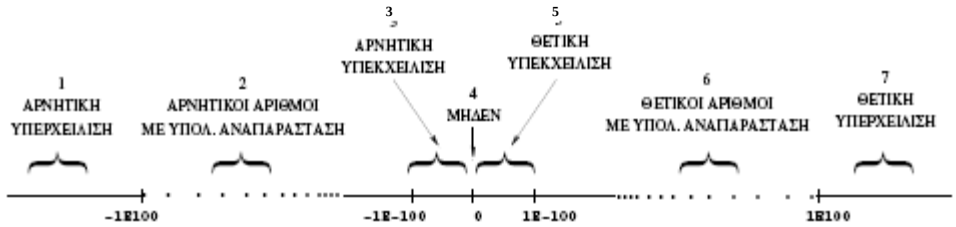
Για το f: *Χρειάζεται αποθήκευση μόνο των δεκαδικών του ψηφίων (αποθηκεύεται ως ακέραιος).*

Άρα, για την αναπαράσταση ενός πραγματικού, χρησιμοποιούνται 2 ακέραιοι: f και e.

- π.χ., ο 3.141592 αποθηκεύεται ως $f = +3141592$ και $e = +1$.

- Έστω αναπαράσταση με 3-ψήφιο f και 2-ψήφιο e:

Αναπαράσταση μεγεθών από $+0.100\times10^{-99}$ έως $+0.999\times10^{+99}$ (με 5 ψηφία (3+2) και 2 πρόσημα) :



- "Μεγάλοι" αρνητικοί αριθμοί $< -0.999\times10^{99}$
- Αρνητικοί αριθμοί από -0.999×10^{99} έως -0.100×10^{-99}
- "Μικροί" αρνητικοί αριθμοί με μέτρο $< 0.100\times10^{-99}$
- Το μηδέν
- Μικροί θετικοί αριθμοί με μέτρο $< 0.100\times10^{-99}$
- Θετικοί αριθμοί ανάμεσα σε 0.100×10^{-99} και 0.999×10^{99}
- Μεγάλοι θετικοί αριθμοί $> 0.999\times10^{99}$

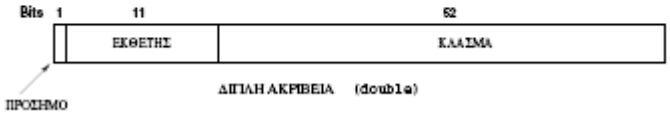
Οι περιοχές 1, 3, 5, 7 δεν έχουν υπολογιστική αναπαράσταση.

<u>IEEE 754</u> – Standard για αναπαράσταση / υπολογισμούς κινητής υποδιαστολής

- Δύο ακρίβειες:

32-bit:

64-bit:



- Ειδικά σύμβολα:
  - `-Inf`, `+Inf`: περιοχές 1 και 7 αντίστοιχα
  - `NaN`: για αδύνατες πράξεις, π.χ. 0 / 0, 0 × ∞, ∞ / ∞
- Δυνατότητα αναπαράστασης και κάποιων αριθμών στις περιοχές 3 και 4.

<u>Αναπαράσταση χαρακτήρων</u>

Κωδικοποιούνται σαν ακέραιοι:

χαρακτήρας ↔ ακέραιος

Υπάρχουν διάφορα σύνολα χαρακτήρων: ASCII, Unicode, κτλ.

- **<u>ASCII</u> (American Standard Code for Information Interchange):**

  Έχει 256 χαρακτήρες → απεικόνιση στους ακέραιους 0-255.

  π.χ., ‘a’ ↔ 97, ‘B’ ↔ 66, ‘0’ ↔ 48.

  Οι πρώτοι 128 χαρακτήρες είναι οι πιο σημαντικοί.
  128-255: μη λατινικά αλφάβητα και γραφικά σύμβολα.

- **<u>Unicode</u>:**

  65536 χαρακτήρες. Οι 128 πρώτες θέσεις είναι ίδιες με του ASCII.