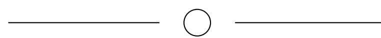
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, Τμήμα Μαθηματικών

Εργαστήρια Πληροφορικής Ι

Χειμερινό Εξάμηνο 2023-24



```
01. function f=fibonacci(n)
02.     if n>1
03.         f=fibonacci(n-1)+fibonacci(n-2);
04.     elseif n==1
05.         f=1;
06.     else
07.         f=0;
08.     endif
09. return
```

Τελευταία ενημέρωση: 25/09/2023 Ώρα: 21.55

Τα πνευματικά δικαιώματα της γραμματοσειράς ανήκουν στο Τμ. Μαθηματικών του Παν. Αιγαίου. Περισσότερες πληροφορίες μπορείτε να βρείτε εδώ:

http://iris.math.aegean.gr/kerkis/

ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ

αβγδεζηθικλμνξοπρστυφχψω

ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ

αβγδεζηθικλμνξοπρστυφχψω

ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ

αβγδεζηθικλμνξοπρστυφχψω

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdedghijklmnopqrstuvwxyz

BCDEFGHIJKLMNOPQRSTUVWXYZ

abcdedghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdedghijklmnopqrstuvwxyz

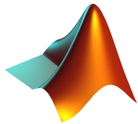
0123456789,./:“”()[]{}<>-!#$%&*

*0123456789,./:“”()[]<>{}-!#$%&**

0123456789,./:“”()[]<>{}-!#$%&*

Κέρκης © Τμήμα Μαθηματικών, Πανεπιστήμιο Αιγαίου.

Βασικές εντολές «έν τάχει»



Για τη Matlab

Εντολές υπολογισμών: Στη Matlab υπάρχει η δυνατότητα εκτέλεσης πράξεων, με τον εξής τρόπο (η ans αποτελεί κάθε φορά «απάντηση» της Matlab):

Πρόσθεση: Χρησιμοποιείται ο τελεστής «+»:

```
1054+145
ans = 1199
```

Πολλαπλασιασμός: Χρησιμοποιείται ο τελεστής «*»:

```
54*234
ans = 12636
```

Δύναμη: Χρησιμοποιείται ο τελεστής «^»:

```
38^9
ans = 165216101262848
```

Αφαίρεση: Χρησιμοποιείται ο τελεστής «−»:

```
8461-6352
ans = 2109
```

Διαίρεση: Χρησιμοποιείται ο τελεστής «/»:

```
26/5
ans = 5.2
```

Χωράει σε: Χρησιμοποιείται ο τελεστής «\»:

```
5\36
ans = 7.2
```

Συναρτήσεις: Στη Matlab υπάρχουν διάφορες συναρτήσεις, μερικές στοιχειώδεις ακολουθούν:

Στρογγυλοποίηση «προς τα κάτω» / ακέραιο μέρος: Χρησιμοποιείται η «floor»:

```
floor(34.2)
ans = 34
floor(34.99)
ans = 34
```

Στρογγυλοποίηση «προς τα πάνω»: Χρησιμοποιείται η «ceil»:

```
ceil(34.2)
ans = 35
ceil(34.99)
ans = 35
```

Δίκαιη στρογγυλοποίηση: Χρησιμοποιείται η «round»:

```
round(34.2)
ans = 34
round(34.99)
ans = 35
```

Συνάρτηση προσήμου: Χρησιμοποιείται η «sign»:

```
sign(-10)
ans = -1
sign(10)
ans = 1
sign(0)
ans = 0
```

Απόλυτη τιμή: Χρησιμοποιείται η «abs»:

```
abs(-10)
ans = 10
abs(10)
ans = 10
```

Υπόλοιπο διαίρεσης: Χρησιμοποιείται η «rem» ή η «mod»:

```
rem(10,3)
ans = 1
mod(10,3)
ans = 1
```

Τριγωνομετρικές συναρτήσεις: Για το ημίτονο, το συνημίτονο και την εφαπτομένη, χρησιμοποιούνται αντίστοιχα τα «sin», «cos» και «tan» (τα ορίσματα είναι σε rad):

```
sin(42)
ans = -0.91652
cos(42)
ans = -0.39999
tan(42)
ans = 2.2914
```

Εν τω μεταξύ, βασική εντολή είναι η «pi», που δίνει μία προσέγγιση του π.

Εκθετική συνάρτηση: Χρησιμοποιείται η «exp»:

```
exp(4)=54.598
```

Εν τω μεταξύ, βασική εντολή είναι η «e», που δίνει μία προσέγγιση του e.

Τετραγωνική ρίζα: Χρησιμοποιείται η «sqrt»:

```
sqrt(256)=16
```

Φυσικά, η σειρά των πράξεων γίνεται από τα αριστερά προς τα δεξιά, με σεβασμό στη σειρά των πράξεων. Επομένως:

```
8/8*2
ans = 2
```

και όχι:

```
8/8*2
ans = 0.5
```

Μεταβλητές: Στη Matlab οι μεταβλητές μπορούν να έχουν οποιοδήποτε (αλφαριθμητικό) όνομα, με τους εξής περιορισμούς:

Δεν είναι δυνατόν να ξεκινούν με αριθμό:

```
2x=1
parse error: syntax error
```

Δεν είναι δυνατόν να οριστούν μέσω τύπου ή να περιέχουν σύμβολο που έχει κι άλλη χρήση:

```
x+2=5
parse error: invalid left hand side of assignment
```

ή:

```
x)=5
parse error: syntax error
```

Παρόλα αυτά, τα παρακάτω είναι αποδεκτά:

```
x2=5
x_=5
```

Καλύτερα να μην έχουν όνομα συνάρτησης:

```
sin=1
```

Αργότερα, αν χρειαστούμε την «πραγματική» συνάρτηση `sin`, θα έχουμε πρόβλημα:

```
sin(22)
error: sin(22): out of bound 1 (note: variable 'sin' shadows
function)
```

Παρατηρήστε ότι η εκχώριση σε μεταβλητή γίνεται απλά με το «=».

Εκχώριση σε μεταβλητή: Χρησιμοποιείται το «=»:

```
x=45212
```

Προσοχή: Το «=» στην εκχώριση δεν λειτουργεί μεταθετικά. Δηλαδή, δεν μπορούμε να γράψουμε:

```
5=x
parse error: invalid constant left hand side of assignment
```

Λογικοί τελεστές: Οι λογικοί τελεστές είναι ιδιαίτερα χρήσιμοι στα `if` και `while`, τα οποία θα δούμε παρακάτω.

Είναι ίσο: Χρησιμοποιείται το «==»:

```
1==1
ans = 1
2==1
ans = 0
```

Στην ουσία μπορούμε βλέπουμε τους λογικούς τελεστές ως ερώτηση για τον υπολογιστή. Για παράδειγμα, για το ==, ρωτάμε αν δύο αντικείμενα είναι ίδια, κι ο υπολογιστής απαντά 1 αν είναι ίδια, ενώ 0 αν δεν είναι. Γενικά το 1 μπορεί να αντικατασταθεί από το true, ενώ το 0 από το false.

Δεν είναι ίσο: Χρησιμοποιείται το «~=»:

```
1~=1
ans = 0
2~=1
ans = 1
```

Η σημασία της περισπωμένης «~» βρίσκεται σε αυτό που λέμε «άρνηση μίας μαθηματικής πρότασης», και πιθανότατα θα έχετε συνατήσει στο μάθημα των θεμελίων. Η άρνηση υπάρχει και στην Matlab, και στην ουσία εναλλάσει τις λογικές τιμές true και false.

Άρνηση: Χρησιμοποιείται το «~». Για παράδειγμα, εάν θεωρήσουμε τις μαθηματικές πρότασεις 1==1 και 2==1:

```
~(1==1)
ans = 0
~(2==1)
ans = 1
```

Παρατηρήστε ότι 1~=1 είναι ίδιο με το ~(1==1), κι επίσης το 2~=1 είναι ίδιο με το ~(2==1).

Άλλες σχέσεις μεταξύ μαθηματικών προτάσεων θα δούμε ευθύς αμέσως. Θα ξεκινήσουμε (ξανά) με την «==», γιατί είναι χρήσιμο να παρατηρήσουμε μία ιδιαιτερότητα: η εν λόγω σχέση μπορεί να χρησιμοποιηθεί για να ελεγχθεί αν δύο μαθηματικές προτάσεις είναι ίδιες, δηλαδή αν έχουν τις ίδιες τιμές αληθείας.

Εάν θεωρήσουμε τις προτάσεις 1==3 και 34==4 (που και οι δύο είναι ψευδείς, με τιμή 0), τότε:

```
(1==3)==(34==4)
ans = 1
```

Εάν θεωρήσουμε τις προτάσεις 1==3 και 4==4 (που και η πρώτη είναι ψευδής, με τιμή 0, και η δεύτερη αληθής, με τιμή 1), τότε:

```
(1==3)==(4==4)
ans = 0
```

Άλλες σχέσεις μεταξύ προτάσεων είναι οι ακόλουθες:

Ή το ένα ή το άλλο ή και τα δύο: Χρησιμοποιείται το «||»:

```
(1==2)||(12==4)
ans = 0
(1==1)||(12==4)
ans = 1
(1==2)||(12==12)
ans = 1
(1==1)||(12==12)
ans = 1
```

Και: Χρησιμοποιείται το «&&»:

```
(1==2)&&(12==4)
ans = 0
(1==1)&&(12==4)
ans = 0
(1==2)&&(12==12)
ans = 0
(1==1)&&(12==12)
ans = 1
```

Επίσης, αναφέρουμε τις σχέσεις διάταξης.

Είναι μικρότερο: Χρησιμοποιείται το «<»:

```
1<1
ans = 0
2<1
ans = 0
1<2
ans = 1
```

Είναι μεγαλύτερο: Χρησιμοποιείται το «>»:

```
1>1
ans = 0
2>1
ans = 1
1>2
ans = 0
```

Είναι μικρότερο ή ίσο: Χρησιμοποιείται το «<=»:

```
1<=1
ans = 1
2<=1
ans = 0
1<=2
ans = 1
```

Είναι μεγαλύτερο ή ίσο: Χρησιμοποιείται το «>=»:

```
1>=1
ans = 1
2>=1
ans = 1
1>=2
ans = 0
```

Διανύσματα και πίνακες: Όπως προδίδει και η ίδια η ονομασία της Matlab (Matrix και Lab), μεγάλη έμφαση δίνεται στα διανύσματα και στους πίνακες. Για την ακρίβεια, τα περισσότερα αντικείμενα στη Matlab αντιμετωπίζονται ως πίνακες. Ξεκινώντας από τα διανύσματα, έχουμε τα ακόλουθα:

Διανύσματα γραμμή με αναγραφή: Ο απλούστερος τρόπος να οριστεί ένα διάνυσμα είναι με αναγραφή όλων των στοιχείων του. Για παράδειγμα:

```
01. v=[1,3,2]
02. v=
03.     1   3   2
```

(την γραμμή 01 γράφουμε εμείς και τις 02, 03 τις εμφανίζει η Matlab).

Διανύσματα στήλη με αναγραφή: Η διαφορά με τα διανύσματα γραμμή βρίσκεται στο κόμμα. Εδώ, υπάρχει ερωτηματικό.

```
01. v=[1;3;2]
02. v=
03.     1
04.     3
05.     2
```

(την γραμμή 01 γράφουμε εμείς και τις 02, 03, 04, 05 τις εμφανίζει η Matlab).

Διανύσματα γραμμή που εμφανίζουν «κανονικότητα»: Πολλές φορές (κι αυτό θα το δούμε στο for) μας ενδιαφέρει μία αρίθμηση που είναι συνεχόμενη ή εμφανίζει ένα αριθμητικό μοτίβο. Σ' αυτήν την περίπτωση, μπορούμε να γράψουμε:

```
v=start:step:end
```

όπου start είναι ο αριθμός από τον οποίον ξεκινά η αρίθμηση, step είναι ο αριθμός που προστίθεται ώστε να ληφθεί κάθε επόμενος όρος και end είναι το άνω φράγμα της αρίθμησης (ώστε κάποτε να τερματιστεί). Αν το βήμα είναι 1, μπορεί να παραλειφθεί από τον συμβολισμό και να γραφεί v=start:end.

```
v=1:1:5
v=
   1   2   3   4   5

v=1:2:5
v=
   1   3   5

v=2:2:5
v=
   2   4

v=10:-1:1
v=
   10   9   8   7   6   5   4   3   2   1

v=5:-2:1
v=
   5   3   1
```

Αφού οριστεί ένα διάνυσμα, μπορεί να ανακτηθεί οποιοδήποτε στοιχείο του.

Ανάκτηση στοιχείου: Εάν έχουμε ένα διάνυσμα v, με v(index) μπορούμε να εξάγουμε το στοιχείο του στη θέση index.

```
v=[1,3,2]
v=
   1   3   2
v(2)
ans = 3
```

Επίσης, μπορούν να οριστούν πίνακες («διανύσματα» δύο διαστάσεων). Υπενθυμίζουμε ότι στα μαθηματικά συμβολίζουμε τους πίνακες αυτούς με:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

δηλαδή, τα στοιχεία $a_{i,j}$ έχουν δείκτες που δείχνουν τις συντεταγμένες τους, ο πρώτος τη γραμμή κι ο δεύτερος τη στήλη. Επειδή ο προηγούμενος πίνακας έχει m γραμμές και n στήλες, λέμε ότι είναι ένας $m \times n$ πίνακας. Ειδικοί πίνακες είναι οι μηδενικοί:

$$\mathrm{O}_{m,n} = \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}}_{n} \left.\vphantom{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}\right\} m$$

και οι (τετραγωνικοί) ταυτοτικοί:

$$\mathrm{Id}_n = \underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}}_{n} \left.\vphantom{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}\right\} n$$

Πίνακες με αναγραφή: Και πάλι, ο απλούστερος τρόπος κανείς να ορίσει έναν πίνακα (όπως και στα διανύσματα) είναι με αναγραφή των στοιχείων του. Σημειώστε ότι με κόμμα ξεχωρίζουμε τα στοιχεία σε γραμμές, ενώ με ερωτηματικό τις στήλες.

```
01. A=[1,2,3; 4,5,6; 7,8,9]
02. A=
03.     1   2   3
04.     4   5   6
05.     7   8   9
```

(την γραμμή 01 γράφουμε εμείς και τις 02, 03, 04, 05 τις εμφανίζει η Matlab).

Μηδενικοί πίνακες: Αρκετά χρήσιμοι (ιδίως σε αρχικοποιήσεις) είναι οι πίνακες που έχουν στοιχεία μόνο μηδενικά. Αυτούς μπορούμε γενικά να τους ορίσουμε μέσω της εντολής `zeros(m,n)` ή, εάν θέλουμε τετραγωνικό πίνακα, `zeros(n)`.

```
A=zeros(2,3)
A=
    0   0   0
    0   0   0
B=zeros(3)
B=
    0   0   0
    0   0   0
    0   0   0
```

Ταυτοτικοί πίνακες: Είναι επίσης δυνατόν κανείς να ορίσει τον $n \times n$ ταυτοτικό πίνακα, μέσω της εντολής eye(n).

```
A=eye(4)
A=
    1   0   0   0
    0   1   0   0
    0   0   1   0
    0   0   0   1
```

Υπάρχει επίσης μία γενίκευση, η eye(m,n), που ουσιαστικά τοποθετεί στον zeros(m,n) τον μεγαλύτερο δυνατό ταυτοτικό πίνακα (πάνω αριστερά).

```
A=eye(4,3)
A=
    1   0   0
    0   1   0
    0   0   1
    0   0   0
A=eye(3,4)
B=
    1   0   0   0
    0   1   0   0
    0   0   1   0
```

Κενός πίνακας / διάνυσμα: Ορίζεται ως εξής:

```
A=[]
```

Αφού οριστούν πίνακες, μπορεί να ανακτηθεί οποιοδήποτε στοιχείο τους.

Ανάκτηση στοιχείου: Εάν έχουμε έναν πίνακα A, μπορούμε να εξάγουμε το στοιχείο του στη θέση (k,l).

```
A=[1,3; 2,0]
A=
    1   3
    2   0
A(2,1)
ans = 2
```

Πράξεις σε διανύσματα και πίνακες: Μεταξύ πινάκων (άρα και διανυσμάτων) μπορούν να οριστούν πράξεις πρόσθεσης, βαθμωτού πολλαπλασιασμού και πολλαπλασιασμού. Συγκεκριμένα, ας θεωρήσουμε ότι έχουμε πίνακες A (διάστασης $m \times n$), B (διάστασης $m \times n$) και C (διάστασης $n \times p$), τους οποίους θα τους συμβολίσουμε με το γενικό τους στοιχείο, $A = (a_{i,j})_{i,j}$, $B = (b_{i,j})_{i,j}$ και $C = (c_{i,j})_{i,j}$, για ευκολία. Για την πρόσθεση έχουμε:

$$A + B = (a_{i,j})_{i,j} + (b_{i,j})_{i,j} = (a_{i,j} + b_{i,j})_{i,j}$$

για τον βαθμωτό πολλαπλασιασμό με $\lambda \in \mathbb{R}$:

$$\lambda \cdot A = \lambda \cdot (a_{i,j})_{i,j} = (\lambda \cdot a_{i,j})_{i,j}$$

και για τον πολλαπλασιασμό:

$$A \cdot C = \left(\sum_{k=1}^{n} a_{i,k} c_{k,j} \right)_{i,j}$$

Από τους αρισμούς αυτούς, οι πίνακες $A+B$, $\lambda \cdot A$ είναι $m \times n$, ενώ ο $A \cdot C$ είναι $m \times p$.

Στη Matlab αυτές οι πράξεις υπάρχουν, όπως κι όσες προκύπτουν από αυτές (ύψωση σε δύναμη, αφαίρεση).

Πρόσθεση πινάκων: Χρησιμοποιείται το σύμβολο ≪+≫:

```
A=[1,2; 3,4]
A=
   1   2
   3   4
B=[3,2; 1,1]
B=
   3   2
   1   1
A+B
ans =
   4   4
   4   5
```

Πολλαπλασιασμός πινάκων: Χρησιμοποιείται το σύμβολο ≪*≫:

```
A=[1,2; 3,4]
A=
   1   2
   3   4
B=[3,2; 1,1]
B=
   3   2
   1   1
A*B
ans =
    5    4
   13   10
```

Δύναμη πίνακα:: Για διαδοχικούς πολλαπλασιασμούς, χρησιμοποιείται το σύμβολο ≪^≫:

```
A=[1,2; 3,4]
A=
   1   2
   3   4
A^2
ans =
    7   10
   15   22
```

Αφαίρεση πινάκων: Χρησιμοποιείται το σύμβολο ≪−≫:

```
A=[1,2; 3,4]
A=
   1   2
   3   4
B=[3,2; 1,1]
B=
```

```
    3   2
    1   1
A-B
ans =
   -2   0
    2   3
```

Η ιδιαιτερότητα του πολλαπλασιασμού (και της δύναμης) σε πίνακες είναι ότι δεν είναι σημειακή πράξη. Δεν ισχύει δηλαδή $(a_{i,j})_{i,j} \cdot (b_{i,j})_{i,j} = (a_{i,j} \cdot b_{i,j})_{i,j}$. Πάντως, εάν κανείς διαθέτει δύο $m \times n$ πίνακες, $A = (a_{i,j})_{i,j}$ και $B = (b_{i,j})_{i,j}$, είναι δυνατόν στη Matlab να οριστεί ο κατά σημείο πολλαπλασιασμός (κι αντίστοιχα η κατά σημείο δύναμη):

Κατά σημείο πολλαπλασιασμός: Χρησιμοποιείται το σύμβολο «.*»:

```
A=[1,2; 3,4]
A=
    1   2
    3   4
B=[3,2; 1,1]
B=
    3   2
    1   1
A.*B
ans =
    3   4
    3   4
```

Κατά σημείο δύναμη: Χρησιμοποιείται το σύμβολο «.^»:

```
A=[1,2; 3,4]
A=
    1   2
    3   4
B=[3,2; 1,1]
B=
    3   2
    1   1
A.^B
ans =
    1   4
    3   4
```

Συναρτήσεις σε διανύσματα και πίνακες: Διάφορες χρήσιμες συναρτήσεις σε διανύσματα και πίνακες είναι οι ακόλουθες:

Μήκος διανύσματος: Χρησιμοποιείται η «length»:

```
v=[1,6,9,3,4]
v=
    1   6   9   3   4
length(v)
ans = 5
```

Ελάχιστο και μέγιστο στοιχείο διανύσματος: Χρησιμοποιούνται οι «min» και «max» αντίστοιχα:

```
v=[1,6,9,3,4]
v=
   1   6   9   3   4
min(v)
ans = 1
max(v)
ans = 9
```

Μέγεθος (διαστάσεις) πίνακα: Χρησιμοποιείται η «size»:

```
A=[1,6; 9,3; 1,2]
A=
   1   6
   9   3
   1   2
size(A)
ans =
   3   2
```

Είναι δυνατόν η συνάρτηση size (όπως κι οποιαδήποτε άλλη πλειότιμη συνάρτηση) να δώσει τιμές σε διακεκριμένες μεταβλητές κι όχι σε πίνακα. Για παράδειγμα:

```
A=[1,6; 9,3; 1,2]
A=
   1   6
   9   3
   1   2
[m,n]=size(A)
m = 3
n = 2
```

Ελάχιστα και μέγιστα στοιχεία πίνακα: Με τις συναρτήσεις «min» και «max» είναι δυνατόν να εξαχθεί το ελάχιστο και μέγιστο στοιχείο αντίστοιχα, κάθε στήλης του πίνακα. Η εξαγωγή γίνεται σε διάνυσμα.

```
A=[1,6; 9,3; 1,2]
A=
   1   6
   9   3
   1   2
min(A)
ans =
   1   2
max(A)
ans =
   9   6
```

Αν λοιπόν επιθυμούμε το ελάχιστο ή μέγιστο στοιχείο όλου του πίνακα, θα πρέπει να γράψουμε:

```
min(min(A))
ans = 1
max(max(A))
ans = 9
```

Υπενθυμίζουμε ότι, εάν A είναι ο πίνακας:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

τότε ο ανάστροφός του είναι ο:

$$A^T = \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{bmatrix}$$

Επομένως, ο A^T προκύπτει από τον A κάνοντας τις γραμμές στήλες (το στοιχείο στη θέση i, j είναι το $a_{j,i}$ κι όχι το $a_{i,j}$).

Ανάστροφος πίνακας: Για να βρούμε τον ανάστροφο ενός πίνακα, χρησιμοποιούμε τη συνάρτηση transpose:

```
A=[1,2,3; 4,5,6]
A=
   1   2   3
   4   5   6
transpose(A)
ans =
   1   4
   2   5
   3   6
```

Επίσης, ορισμένες φορές ένας τετραγωνικός πίνακας A έχει αντίστροφο. Δηλαδή, υπάρχει πίνακας A^{-1} ώστε:

$$A^{-1} \cdot A = A \cdot A^{-1} = \mathrm{Id}$$

Αντίστροφος πίνακας: Για να βρούμε τον αντίστροφο ενός πίνακα (που έχει αντίστροφο), χρησιμοποιούμε τη συνάρτηση inverse:

```
A=[0,1; 1,1]
A=
   0   1
   1   1
inverse(A)
ans =
   -1   1
    1   0
```

Ορίζουσα πίνακα: Για να ελέγξουμε εάν ένας πίνακας είναι αντιστρέψιμος, ελέγχουμε ότι η ορίζουσα det είναι μη μηδενική.

```
A=[0,1; 1,1]
A=
   0   1
   1   1
```

```
det(A)
ans = -1
det(zeros(2))
ans = 0
```

Βρόγχοι `if`, `while` **και** `for`**:**

`if:` Με την εντολή `if`[κάποια λογική συνθήκη] μπορούμε να εκτελέσουμε εντολές μόνο σε συγκεκριμένες περιπτώσεις. Για παράδειγμα, το παρακάτω θα εκτελεστεί και θα βρούμε το αποτέλεσμα της πράξης `x+234`:

```
x=1
if x>=0
   x+234
endif
```

ενώ το παρακάτω δεν θα εκτελεστεί και δεν θα βρούμε το αποτέλεσμα της πράξης `x+234`:

```
x=-1
if x>=0
   x+234
endif
```

Δίπλα από το `if` εισάγονται λογικές προτάσεις που φτιάχνονται (συνήθως) χρησιμοποιώντας τους λογικούς τελεστές που γράψαμε παραπάνω. Για παράδειγμα:

```
x>=1
x==1
x<=y
(x<=y)||(y>=0)
```

και ούτω καθεξής.

`elseif:` Ενδέχεται σε δύο διαφορετικά ενδεχόμενα να θέλουμε να κάνουμε δύο διαφορετικές διαδικασίες. Σ' αυτήν την περίπτωση, χρησιμοποιούμε το `elseif`. Στο παρακάτω εκχωρούμε `y=x-1` εάν `x>=1`, ενώ `y=-1` εάν `x==-3`.

```
if x>=1
   y=x-1
elseif x==-3
   y=-1
endif
```

Για τις υπόλοιπες περιπτώσεις, το πρόγραμμα δεν κάνει τίποτα.

Προσοχή: Για να δουλέψει το πρόγραμμα, η τιμή `x` θα πρέπει -με κάποιον τρόπο- να έχει προσδιοριστεί (για παράδειγμα, νωρίτερα στο πρόγραμμα).

`else:` Με την εντολή `else` μπορούμε να συμπεριλάβουμε όλες τις εναπομείνουσες περιπτώσεις σε ένα `if`. Για παράδειγμα, το παρακάτω `else` εκτελείται μόνο όταν `x==0`, δηλαδή όταν `~((x<0)||(x>0))`.

```
if x<0
   H=0
elseif x>0
   H=1
```

```
else
   H=0.5
endif
```

Επίσης, το παρακάτω `else` εκτελείται μόνο όταν `x~=0`.

```
if x==0
   d=1
else
   d=0
endif
```

Προσοχή: Για να δουλέψει το πρόγραμμα, η τιμή x θα πρέπει -με κάποιον τρόπο- να έχει προσδιοριστεί (για παράδειγμα, νωρίτερα στο πρόγραμμα).

`while:` Με την εντολή `while`[μία λογική συνθήκη] μπορούμε να εκτελούμε εντολές με επανάληψη, όσο η [λογική συνθήκη] είναι αληθής. Για παράδειγμα, το παρακάτω εκτελείται έως ότου `x==89`:

```
01. x=1
02. while x<89
03.     x=x+1
04. endwhile
```

Παρατηρήστε εδώ ότι το = στην γραμμή 03 εκφράζει εκχώριση. Δηλαδή, το x παίρνει την τιμή του συν 1. (Έτσι λοιπόν, δεν υπάρχει κάποιο παράδοξο $x = x + 1 \Rightarrow 0 = 1$, αφού το ίσον δεν είναι ακριβώς ίσον, είναι εκχώριση). Διαφορετικά θα μπορούσαμε να είχαμε γράψει (χρησιμοποιώντας τον τελεστή +=):

```
03. x+=1
```

Στους βρόγχους `while` μπορεί να προστεθούν εντολές `break` και `continue`, οι οποίες είναι αρκετά χρήσιμες, ακόμη κι αν φέρουν κάποια υποκειμενικότητα στη χρήση τους. Υπάρχουν σχολές που χρησιμοποιούν τα `break` και `continue` για τη χρησιμότητά τους, κι άλλες που τα αποφεύγουν λόγω του ότι κάνουν τα προγράμματα δυσανάγνωστα (αν και ίσως αυτό ίσχυε περισσότερο σε παλαιότερες γλώσσες προγραμματισμού).

`break:` Με την εντολή `break` μπορούμε να σταματήσουμε πρόωρα την εκτέλεση ενός βρόγχου επανάληψης `while`. Για παράδειγμα, ας φτιάξουμε το ακόλουθο πρόγραμμα, που βρίσκει τον μικρότερο διαιρέτη `least_div` (εκτός του 1) ενός φυσικού αριθμού x (μεγαλύτερου του 1):

```
least_div=1
d=2
while least_div~=1
   if mod(x,d)==0
      least_div=d
   endif
   d=d+1
endwhile
```

Με την εντολή `break` το παραπάνω πρόγραμμα μπορεί να πάρει την ακόλουθη μορφή:

```
01. d=2
02. while true
03.     if mod(x,d)==0
```

```
04.         least_div=d
05.         break
06.     endif
07.     d=d+1
08. endwhile
```

Σχόλια: Στη γραμμή 02 η εντολή «while true» σημαίνει while για πάντα, αφού το true (δηλαδή η λογική τιμή 1) είναι πάντοτε μία αληθής μαθηματική πρόταση. Στη γραμμή 05 το break σταματά την επανάληψη όταν το d διαιρέσει το x.

Προσοχή: Για να δουλέψει το πρόγραμμα, η τιμή x θα πρέπει -με κάποιον τρόπο- να έχει προσδιοριστεί (για παράδειγμα, νωρίτερα στο πρόγραμμα).

continue: Με την εντολή continue μπορούμε να παραλείψουμε τις εντολές που εμφανίζονται μετά απ' αυτό, για εκείνη την επανάληψη στην οποία αυτό εμφανίστηκε. Για να καταλάβουμε τη λειτουργία του, ας φτιάξουμε ένα πρόγραμμα το οποίο θα κάνει το εξής: Θα δίνεται ένα διάνυσμα αριθμών v, κι εμείς θα υπολογίζουμε το άθροισμα στο οποίο κάθε αρνητικός όρος του v θα συνισφέρει −2, ενώ κάθε μη αρνητικός με 1.

```
k=1
sum=0
while k<=length(v)
   if v(k)<0
      sum=sum-2
   else
      sum=sum+1
   endif
   k=k+1
endwhile
```

Με την εντολή continue το παραπάνω πρόγραμμα μπορεί να πάρει την ακόλουθη μορφή:

```
01. k=0
02. sum=0
03. while k<length(v)
04.     k=k+1
05.     if v(k)<0
06.         sum=sum-2
07.         continue
08.     endif
09.     sum=sum+1
10. endwhile
```

Σχόλια: Στη γραμμή 07 το continue ξεχνά τις εντολές στις επόμενες γραμμές (αφού τελειώσει το if), δηλαδή ξεχνά την γραμμή 09. Επίσης, η γραμμή 04 προστίθεται πριν το continue, διότι διαφορετικά θα υπήρχε περίπτωση να μην εκτελεστεί, κι άρα ο δείκτης k να μην αυξηθεί. Το πρόγραμμα τότε δεν θα τερμάτιζε ποτέ.

Προσοχή: Για να δουλέψει το πρόγραμμα, το διάνυσμα v θα πρέπει -με κάποιον τρόπο- να έχει προσδιοριστεί (για παράδειγμα, νωρίτερα στο πρόγραμμα).

for: Ένας άλλος τρόπος να γίνει επαναληπτικά μία διαδικασία είναι μέσω του βρόγχου for. Η ιδιαιτερότητα του for είναι ότι μπορεί να κάνει επανάληψη για κάθε στοιχείο ενός διανύσματος. Για παράδειγμα, αν v=[1,4,6,3], το παρακάτω πρόγραμμα θα υπολογίσει το άθροισμα 1+4+6+3.

```
01. sum=0
02. for elem=v
03.     sum=sum+elem
04. endfor
```

Σχόλια: Στη γραμμή 02 το elem=v δείχνει άλλη μία χρήση του «=». Σε αυτήν την περίπτωση το χρησιμοποιούμε όπως το «ανήκει (∈)».

Είναι πολύ συχνό στη θέση του v να υπάρχει ένα διάνυσμα 1:n. Για παράδειγμα, εάν θέλουμε να υπολογίσουμε το άθοισμα:

$$\sum_{k=1}^{1043} \frac{6}{k^2}$$

μπορούμε να γράψουμε:

```
sum=0
for k=1:1043
    sum=sum+6/k^2
endfor
```

Είσοδος και έξοδος: Μέχρι τώρα έχουμε αναφερθεί σε προγράμματα που μπορούν να κάνουν υπολογισμούς, δεν μπορούν όμως να έχουν κάποια είσοδο ή έξοδο. Για την ακρίβεια, τα προγράμματα έχουν έξοδο (το πρόγραμμα δίνει μία «απάντηση»), απλώς όχι όπως θα θέλαμε. Τι εννοούμε:

Εάν το πρόγραμμα:

```
01. sum=0
02. for k=1:3
03.     sum=sum+6/k^2
04. endfor
```

εκτελεστεί, φυσιολογικά εμείς θέλουμε ως έξοδο, στην οθόνη μας, το «τελευταίο sum», δηλαδή το αποτέλεσμα της πράξης:

$$\frac{6}{1} + \frac{6}{4} + \frac{6}{9}$$

Στην πραγματικότητα αυτό δεν συμβαίνει, κι ως αποτέλεσμα παίρνουμε το ακόλουθο:

```
sum=0
sum=6
sum=7.5
sum=8.1667
```

στο οποίο το επιθυμητό εποτέλεσμα υπάρχει, όμως με επιπλέον «άχρηστη» πληροφορία. Αυτό συμβαίνει διότι η Matlab κάθε φορά που συναντά μία μεταβλητή, εμφανίζει την τιμή της στην οθόνη. Επομένως, εμφανίζει την sum στη γραμμή 01, καθώς και κάθε άλλη sum στη γραμμή 03, κατά τις διάφορες επαναλήψεις.

Σίγαση: Για να αποφύγουμε το παραπάνω πρόβλημα, μπορούμε να εισάγουμε δίπλα από κάθε μεταβλητή (και στο τέλος της γραμμής), της οποίας η τιμή δεν θέλουμε να εμφανιστεί, ένα ερωτηματικό «;». Έτσι λοιπόν, το προηγούμενο πρόγραμμα μπορεί να τροποποιηθεί ώστε να εμφανίζει στην οθόνη μόνο το τελευταίο sum:

```
01. sum=0;
02. for k=1:3
03.     sum=sum+6/k^2;
04. endfor
05. sum
```

Το μόνο sum που εμφανίζεται είναι αυτό της γραμμής 05, που προσθέσαμε. Συγκεκριμένα εμφανίζεται:

```
sum=8.1667
```

Υπάρχουν κι άλλοι τρόποι εμφάνισης, τους οποίους θα δούμε αμέσως τώρα.

disp: Ένας άλλος τρόπος να εμφανιστεί η τιμή (και μόνο αυτή) μίας μεταβλητής είναι μέσω της εντολής disp. Ο κώδικας:

```
x=1;
disp(x)
```

θα εμφανίσει:

```
1
```

Με την disp μπορούν να εμφανιστούν και συμβολοσειρές. Για να δηλώσουμε μία συμβολοσειρά, την εσωκλείουμε σε «'». Ο κώδικας:

```
disp('some text')
```

δίνει:

```
some text
```

fprintf: Μέσω της fprintf, μπορεί να εμφανιστεί ταυτόχρονα η τιμή μίας μεταβλητής και κείμενο (κάτι που μέσω της disp είναι αδύνατο). Αυτό γίνεται με τον εξής τρόπο: Εάν x είναι ένας πραγματικός αριθμός:

```
fprintf('Some text %f', x)
```

Μεταξύ των «'» υπάρχει το κείμενο προς εμφάνιση, μαζί με τον χαρακτήρα «%f». Εκεί ακριβώς, στον χαρακτήρα «%f», θα εμφανιστεί η τιμή του x. Την μεταβλητή x την γράφουμε έξω από τα «'». Το f έχει την έννοια του «float», δηλαδή του πραγματικού αριθμού. Αν θέλαμε να εμφανίσουμε ακέραιο (χωρίς υποδιαστολή και μηδενικά), θα γράφαμε «%d» (decimal). Αντίστοιχα, για χαρακτήρα χρησιμοποιείται το «%c» (character) και για συμβολοσειρά το «%s» (string). Ένα παράδειγμα ακολουθεί:

```
sum=0;
for k=1:3
   sum=sum+6/k^2;
endfor
fprintf('The result of the summation is %f', sum)
```

το οποίο εμφανίζει:

```
The result of the sumation is 8.1667
```

Είναι δυνατόν να εμφανιστούν δύο (ή περισσότερες) διαφορετικές μεταβλητές στο κείμενο, εισάγωντας δύο (ή περισσότερα) «%» και δύο (ή περισσότερες) μεταβλητές.

```
sum=0;
for k=1:3
   sum=sum+6/k^2;
endfor
fprintf('The result of the summation is %f and not %d',
sum, 2)
```

Αυτό εμφανίζει:

```
The result of the summation is 8.1667 and not 2
```

Εξίσου σημαντική είναι και η είσοδος σε ένα πρόγραμμα. Μερικά προγράμματα, για να λειτουργήσουν, χρειάζονται κάποιου είδους είσοδο, όπως (για παράδειγμα) ένα πρόγραμμα που ελέγχει εάν ένας αριθμός είναι πρώτος ή όχι. Σε αυτό το πρόγραμμα θα πρέπει να «πούμε» ποιος αριθμός πρέπει να ελεγθεί, καθώς το ίδιο το πρόγραμμα δεν τον «γνωρίζει» a priori.

input: Με την εντολή input μπορούμε να αναθέσουμε μία τιμή σε μία μεταβλητή.

```
x=input('')
```

Μάλιστα, αν μεταξύ των εισαγωγικών βάλουμε ένα κείμενο, η input θα εμφανίσει το κείμενο. Για παράδειγμα, η:

```
x=input('Give me some number: ')
```

θα εμφανίσει:

```
Give me some number:
```

Έχοντας δει την εντολή input, μπορούμε να κατασκευάσουμε ένα πρόγραμμα που θα ελέγχει εάν ένας αριθμός είναι πρώτος ή όχι.

```
01. n=input('Give me a natural number greater than 1: ');
02. k=2;
03. while true
04.    if mod(n,k)==0
05.       fprintf('Number %d is not prime', n)
06.       break
07.    elseif k==n-1
08.       fprintf('Number %d is prime', n)
09.       break
10.    endif
11.    k=k+1;
12. endwhile
```

Χωρίς break, θα μπορούσαμε να γράψουμε:

```
01. n=input('Give me a natural number greater than 1: ');
02. is_prime=true;
03. k=2;
04. while (k<n)&&(is_prime==true)
```

```
05.     if mod(n,k)==0
06.         is_prime=false
07.     endif
08.     k=k+1;
09. endwhile
10. if is_prime=true
11.     fprintf('Number %d is prime', n)
12. else
13.     fprintf('Number %d is not prime', n)
14. endif
```

Σχόλια: Για το πρώτο πρόγραμμα: Στη γραμμή 03 ξεκινά ένα while για πάντα. Στη γραμμή 06 αυτό το while σταματά όταν βρεθεί ένας διαιρέτης. Αν δεν βρεθεί διαιρέτης στο σύνολο $\{1, 2, 3, \cdots, n-1\}$, το πρόγραμμα σταματά όταν k==n-1, με το break της γραμμής 09.

Σχόλια: Για το δεύτερο πρόγραμμα: Ελέγχουμε όλους τους αριθμούς k<n, εάν διαιρούν το n. Αν εμφανιστεί κάποιος διαιρέτης, με την γραμμή 06 η is_prime γίνεται false και η αναζήτηση διαιρετών (το while) σταματά. Διαφορετικά ελέγχονται όλοι οι αριθμοί k<n και η is_prime παραμένει true (όπως ορίστηκε στη γραμμή 02).

Κατασκευή συναρτήσεων: Στη Matlab κανείς μπορεί να κατασκευάσει συναρτήσεις. Οι συναρτήσεις, σε αντίθεση με τα απλά προγράμματα, έχουν όνομα και μπορούν να κληθούν σε προγράμματα ή στον κώδικα άλλων συναρτήσεων. Είναι χρήσιμες για διάφορους λόγους, ένας απ' αυτούς είναι ο οργάνωση που προσδίδεται σε ένα πρόγραμμα. Ένας άλλος είναι η δυνατότητα να εκτελεστεί ένας αλγόριθμος αναδρομικά.

Προσοχή: Όταν κατασκευάζουμε μία συνάρτηση, το όνομα της συνάρτησης και το όνομα του αρχείου θα πρέπει να συμφωνούν.

Συνάρτηση χωρίς έξοδο: Ξεκινούμε με τις συναρτήσεις χωρίς έξοδο. Παρά την ονομασία τους, οι συναρτήσεις αυτές μπορεί να έχουν έξοδο (υπό τη μορφή κειμένου για παράδειγμα) η έξοδος όμως αυτή δεν μπορεί να χρησιμοποιηθεί. Η sin είναι συνάρτηση με έξοδο, και γι' αυτό μπορούμε να κάνουμε πράξεις:

```
(sin(pi/4)-3)/8
ans = -0.28661
```

Αντίθετα, μία συνάρτηση χωρίς έξοδο δεν δίνει ούτε αριθμό, ούτε συμβολοσειρά και γενικά καμμία έξοδο που ανατίθεται στην ans. Η «έξοδος» είναι ένας αριθμός ή μία συμβολοσειρά, που όμως δεν εκχωρείται στην ans. Έτσι λοιπόν, δεν μπορεί να χρησιμοποιηθεί σε άλλα προγράμματα.

Θα δώσουμε ένα παράδειγμα μίας τέτοιας συνάρτησης: Χρησιμοποιώντας έναν από τους κώδικες για τον έλεγχο των πρώτων αριθμών, κατασκευάζουμε μία συνάρτηση (χωρίς έξοδο) isitprime που θα ελέγχει εάν ένας αριθμός (φυσικός, μεγαλύτερος του 1) είναι πρώτος ή όχι.

```
01. function isitprime(n)
02.     k=2;
03.     while true
04.         if mod(n,k)==0
05.             fprintf('Number %d is not prime', n)
06.             break
```

```
07.         elseif k==n-1
08.            fprintf('Number %d is prime', n)
09.            break
10.         endif
11.         k=k+1;
12.      endwhile
13. endfunction
```

Σχόλια: Παρατηρήστε ότι γίνεται απλώς εκτύπωση ενός μηνύματος χωρίς αυτό να αποθηκεύεται κάπου.

Συνάρτηση με έξοδο: Μία συνάρτηση με έξοδο περιέχει περισσότερη πληροφορία σε σχέση με κάποια αντίστοιχη χωρίς έξοδο. Αυτό συμβαίνει διότι το αποτέλεσμα της συνάρτησης εκχωρείται σε μία μεταβλητή, η οποία έπειτα μπορεί να χρησιμοποιηθεί. Παρακάτω θα δούμε το παράδειγμα της συνάρτησης του ελέγχου των πρώτων, με έξοδο.

```
01. function p=isitprime(n)
02.      k=2;
03.      while true
04.         if mod(n,k)==0
05.            p=false
06.            break
07.         elseif k==n-1
08.            p=true
09.            break
10.         endif
11.         k=k+1;
12.      endwhile
13. return
```

Με εφαρμογή της συνάρτησης παίρνουμε:

```
isitprime(80)
ans = 0
isitprime(13)
ans = 1
```

Σχόλια: Σε αντίθεση με την χωρίς έξοδο συνάρτηση, στην γραμμή 01 χρησιμοποιούμε το `p` για να αναφερθούμε στην τιμή της συνάρτησης. Έπειτα, στην γραμμή 13, με το `return` επιστρέφουμε την τιμή της συνάρτησης, μέσω της `ans`.

Μία συνάρτηση με έξοδο μπορεί να κληθεί σε άλλες συναρτήσεις ή προγράμματα, ακόμη και στον εαυτό της! Η κλήση μίας συνάρτησης στον κώδικα της ίδιας συνάρτησης είναι ακριβώς η ιδέα της αναδρομής.

Είναι φυσιολογικό κανείς να αναρρωτηθεί εάν θα υπάρξει κάποιο πρόβλημα κατά την κλήση της ίδιας της συνάρτησης στη συνάρτηση. Η απάντηση είναι ότι, εάν γίνει σωστά, η αναδρομή δεν θα δημιουργήσει προβλήματα. Η ιδέα είναι η εξής: Ας υποθέσουμε ότι μία συνάρτηση, έστω `myfunc`, για να υπολογίσει το `myfunc(n)` χρειάζεται να υπολογίσει το `myfunc(n-1)`. Για να υπολογίσει κανείς το `myfunc(200)`, βάσει της κατασκευής της συνάρτησης, επαγωγικά θα πρέπει να υπολογίσει το `myfunc(1)`. Εάν βάλουμε «με τα βίας» (με ένα `if`) στον κώδικα της συνάρτησης την τιμή της `myfunc(1)`, θα μπορούν να υπολογιστούν οι `myfunc(2)`, `myfunc(3)`, `myfunc(4)`, $\cdots$, `myfunc(199)`, άρα και η ζητούμενη `myfunc(200)`.

Ένα παράδειγμα τέτοιας συνάρτησης είναι η ακολουθία $(a_n)_{n=0}^{\infty}$ που ορίζεται αναδρομικά με τον εξής τρόπο:

$$
\begin{aligned}
&a_0 = 0 \\
&a_n = (a_{n-1} + 1)^2, \text{ για κάθε } n \in \{1, 2, 3, 4, \cdots\}
\end{aligned}
$$

Ένας κώδικας που εμπλέκει αναδρομή για τον υπολογισμό της παραπάνω ακολουθίας είναι ο επόμενος:

```
function f=myfunc(n)
   if n~=0
      f=(myfunc(n-1)+1)^2;
   else
      f=0;
   endif
return
```

Ένα άλλο παράδειγμα είναι η αναδρομική ακολουθία του Fibonacci, που ορίζεται από τον τύπο:

$$
\begin{aligned}
&F_0 = 0 \\
&F_1 = 1 \\
&F_n = F_{n-1} + F_{n-2}, \text{ για κάθε } n \in \{2, 3, 4, 5, \cdots\}
\end{aligned}
$$

Για τον υπολογισμό των όρων της, μπορούμε να χρησιμοποιήσουμε κώδικα με αναδρομή.

```
function f=fibonacci(n)
   if n>1
      f=fibonacci(n-1)+fibonacci(n-2);
   elseif n==1
      f=1;
   else
      f=0;
   endif
return
```

Εργαστήρια για το μάθημα «*141. Πληροφορική Ι*»