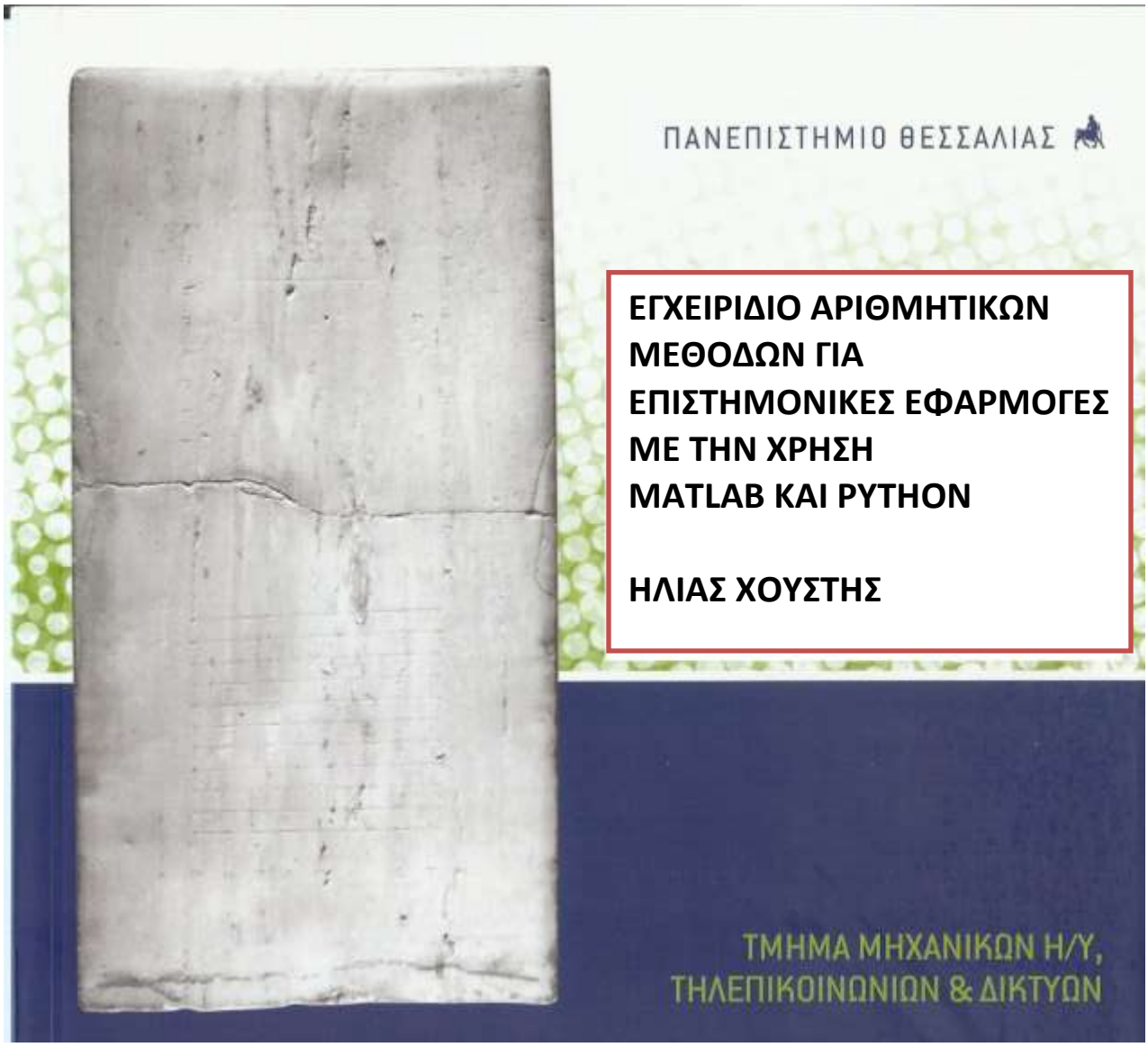

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΕΓΧΕΙΡΙΔΙΟ ΑΡΙΘΜΗΤΙΚΩΝ ΜΕΘΟΔΩΝ ΓΙΑ ΕΠΙΣΤΗΜΟΝΙΚΕΣ ΕΦΑΡΜΟΓΕΣ ΜΕ ΤΗΝ ΧΡΗΣΗ MATLAB ΚΑΙ PYTHON
ΗΛΙΑΣ ΧΟΥΣΤΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ & ΔΙΚΤΥΩΝ

Μάιος 2012

# Αντί Προλόγου

Η μοντέρνα επιστήμη εξαρτάται από τις δυνατότητες μας να εκτελούμε μεγάλης κλίμακας υπολογισμούς για να προσομοιώνουμε τους νόμους της φύσης. Το αντικείμενο των εφαρμοσμένων μαθηματικών είναι η ανάπτυξη και η ανάλυση μοντέλων για την προσομοίωση των φυσικών φαινομένων και το αντικείμενο του επιστημονικού υπολογισμού είναι η ανάπτυξη υπολογιστικών μεθόδων και αλγορίθμων για την προσομοίωση αυτών των μοντέλων σε όσο το δυνατόν πραγματικές συνθήκες. Τα μαθηματικά μοντέλα είναι μια συλλογή από μαθηματικές οντότητες (όπως μεταβλητές, παράμετροι, σταθερές) και σχέσεις μεταξύ των (όπως εξισώσεις, τύποι, ανισότητες).

Τα μαθηματικά μοντέλα ταξινομούνται σύμφωνα με τη φύση των μαθηματικών μεταβλητών που περιλαμβάνονται σ' αυτά. Υπάρχουν δύο βασικές κατηγορίες μεταβλητών. Η μία κατηγορία παριστά ποσότητες, οι οποίες, τουλάχιστον θεωρητικά, μπορούν να μετρηθούν ακριβώς και λέγονται **προσδιορίσιμες** ή **ντετερμινιστικές** μεταβλητές. Η άλλη κατηγορία παριστά ποσότητες που ποτέ δεν μπορούν να μετρηθούν ακριβώς και λέγονται **στοχαστικές** μεταβλητές. Τα μοντέλα με προσδιορίσιμες μεταβλητές διακρίνονται σε **στατικά** και **δυναμικά**. Τα δυναμικά διαφέρουν από τα στατικά στο ότι επιπλέον περιγράφουν μια κατάσταση ή φαινόμενο που μεταβάλλεται χρονικά. Τα μοντέλα με στοχαστικές μεταβλητές λέγονται στοχαστικά, και για την ανάλυσή τους χρησιμοποιούνται μαθηματικές τεχνικές από τις πιθανότητες και την στατιστική.

Στο εγχειρίδιο αυτό θεωρούμε τη λύση διαφόρων μη στοχαστικών μοντέλων που χρησιμοποιούνται συχνά στις επιστημονικές εφαρμογές. Γενικά, η λύση ενός μαθηματικού μοντέλου είναι η διαδικασία εκείνη που μετασχηματίζει το αρχικό μοντέλο στη μορφή

ΜΕΤΑΒΛΗΤΗ = ΤΥΠΟΣ (ΔΕΔΟΜΕΝΑ, ΠΑΡΑΜΕΤΡΟΙ, ΜΕΤΑΒΛΗΤΕΣ)

Για παράδειγμα, θεωρούμε το απλό στατικό μοντέλο $Ax^2 + Bx + C = O$, όπου η μεταβλητή x προσδιορίζεται από τον τύπο $x = \sqrt[2]{(B^2 - 4AC)} / 2A$ σαν συνάρτηση των δεδομένων A, B και C. Δυστυχώς, ο άμεσος προσδιορισμός των μεταβλητών ενός μαθηματικού μοντέλου από κάποιο τύπο είναι συχνά αδύνατος ή ασύμφορος. Για παράδειγμα, θεωρούμε τον προσδιορισμό του χρόνου t στις αποστάσεις *x= .1, .2, ..., .5*, για τις οποίες η θερμοκρασία *T(x,t)* ενός ημι-απείρου μέσου γίνεται 1000 F (δηλαδή T(x, t) = 1000) όταν

$$T(x,t) = T_0 + g_0 / k(2\sqrt{\frac{at}{\pi}}e^{-x^2/4at} - x(1 - erf(\frac{x}{2\sqrt{at}})))$$

με

$$erf(y) = 2\sqrt{\pi}\int_0^y e^{-z^2} dz .$$

Στις περιπτώσεις τέτοιων μοντέλων, εφαρμόζοντας προσεγγίσεις και μετασχηματισμούς στο αρχικό μοντέλο, κατασκευάζουμε είτε ένα ενναλακτικό προσεγγιστικό τύπο ή ένα αλγόριθμο, δηλαδή ένα πεπερασμένο σύνολο από βήματα που προσδιορίζουν μια μερική λύση του μοντέλου, εφαρμόζοντας πεπερασμένο αριθμό από αριθμητικές πράξεις στα δεδομένα και τις παραμέτρους του. Η παραπάνω διαδικασία αναφέρεται σαν η **αριθμητική μέθοδος επίλυσης** του μοντέλου.

Ο κύριος σκοπός του εγχειρίδιου είναι η ανάπτυξη και η μελέτη αριθμητικών μεθόδων που εκτελούνται από υπολογιστές. Η μεθοδολογία που ακολουθούμε για το σκοπό αυτό περιλαμβάνει:

- **Θεωρητική κατανόηση του μοντέλου**: προσδιορισμός συνθηκών που εγγυώνται την ύπαρξη λύσης και μαθηματική συμπεριφορά της λύσης.
- **Ανάπτυξη αλγορίθμων για την αριθμητική λύση του μοντέλου**: προσδιορισμός ενός πεπερασμένου συνόλου από βήματα, που το καθένα εφαρμόζει ένα πεπερασμένο αριθμό αριθμητικών πράξεων στα δεδομένα.
- **Ανάπτυξη λογισμικού (software):** παράσταση ενός ολοκληρωμένου αλγορίθμου στη μορφή προγράμματος για υπολογιστή που μπορεί να χρησιμοποιηθεί από άλλους.
- **Μελέτη της μαθηματικής συμπεριφοράς των αλγορίθμων**: προσδιορισμό συνθηκών που εγγυώνται τη σύγκλιση της αριθμητικής λύσης στην πραγματική λύση και την επίδραση των διάφορων σφαλμάτων, που εισάγονται στη διαδικασία αριθμητικής λύσης, στην ακρίβεια της αριθμητικής λύσης.
- **Μελέτη της υπολογιστικής συμπεριφοράς των αλγορίθμων**: ασυμπτωτική εκτίμηση του έργου που απαιτεί ο αλγόριθμος ως προς διάφορους παραμέτρους (όπως: αριθμός βημάτων του αλγόριθμου, παράμετροι μοντέλου, αρχιτεκτονική υπολογιστικού συστήματος) για τον προσδιορισμό αριθμητικής λύσης, που ικανοποιεί ορισμένα κριτήρια σύγκλισης ή κόστους.

Η ύλη του εγχειριδίου είναι οι σημειώσεις και παρουσιάσεις του μαθήματος «Επιστημονικός Υπολογισμός» για το οποίο ήμουν υπεύθυνος κατά την διάρκεια της θητείας μου στο Τμήμα Μηχανικών ΗΥ, Τηλεπικοινωνιών και Δικτύων του Πανεπιστήμιο Θεσσαλίας. Η ύλη παρουσιάζεται σε μορφή **εγχειριδίου** εμπλουτισμένη με παραδείγματα στην μορφή προγραμμάτων MatLab και Python και σχετικές εργαστηριακές ασκήσεις. Μέρος της ύλης προέρχεται και είναι προσαρμοσμένη στα Ελληνικά από αντίστοιχες σημειώσεις μαθημάτων άλλων συναδέλφων του εξωτερικού των οποίων η ύλη είναι διαθέσιμη στο διαδίκτυο και προσφέρει μια μοναδικότητα στην παρουσίαση και περιεχόμενο κατάλληλη για μηχανικούς. Σχετικές αναφορές στις πηγές που χρησιμοποιήθηκαν γίνονται στα κεφάλαια που ακολουθούν. Υπάρχει σημαντική βιβλιογραφία στο συγκεκριμένο επιστημονικό πεδίο που υποδεικνύει την σημαντικότητα του.

Είμαι υπόχρεος στην συνάδελφο Γιώτα Τσομπανοπούλου και στη συνεργάτισσα Ιωάννα Καφφέ (που για πολλά χρόνια υποστήριξαν το μάθημα του Επιστημονικού Υπολογισμού), την Όλγα Χατζηδήμου, και τους φοιτητές του Τμήματος μου για την συμβολή τους στην ανάπτυξη αυτών των σημειώσεων. Επίσης, ευχαριστώ τον συνάδελφο και μαθητή Μανώλη Βάβαλη για τις παρατηρήσεις του και επεξεργασία του κειμένου.

Τέλος, οι σημειώσεις αυτές διατίθενται δωρεάν σε όσους ενδιαφέρονται να μάθουν αυτή την ενδιαφέρουσα και χρήσιμη ύλη και να εξοικειωθούν με την χρήση των δύο σημαντικών περιβαλλόντων επίλυσης επιστημονικών προβλημάτων MatLab και Python.

ΦΩΤΟΓΡΑΦΙΑ ΕΞΩΦΥΛΛΟΥ: **Ο Άβακας της Σαλαμίνας.**

Πρόκειται για τον παλαιότερο γνωστό άβακα (300 π.χ.). Ήρθε στο φως το 1846 στο νησί της Σαλαμίνας. Είναι μια πλάκα του άσπρου πεντελικού μαρμάρου μήκους 149cm, πλάτους 75cm και πάχους 4.5cm, στην οποία είναι χαραγμένες 5 ομάδες συμβόλων. Στο ένα μέρος του άβακα είναι ένα σύνολο 5 παράλληλων γραμμών που διαιρούνται από μια κάθετη γραμμή. Από την άλλη πλευρά της ρωγμής υπάρχει μια άλλη ομάδα 11 παράλληλων γραμμών, που διαιρείται πάλι σε δύο τμήματα με μια κάθετο. Η τρίτη, έκτη και ένατη γραμμή είναι σημειωμένες με x στα σημεία τομής τους με την κάθετο. Τρία σύνολα Ελληνικών συμβόλων αριθμών υπάρχουν στις πλευρές της πλάκας.

# ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

# ΚΕΦΑΛΑΙΟ 1: ΠΟΛΥΩΝΥΜΑ ΚΑΙ ΠΟΛΥΩΝΥΜΙΚΗ ΠΑΡΕΜΒΟΛΗ

## Περιεχόμενα

**Σφάλμα! Δεν έχει οριστεί σελιδοδείκτης.**

### 1. Γενικές ιδιότητες των πολυωνύμων:

- Ένα γενικό πολυώνυμο ***n*** βαθμού:

$$y = P_n(x) = c_1 x^n + c_2 x^{n-1} + \ldots + c_{n-1} x^2 + c_n x + c_{n+1}$$

- Παραγοντοποίηση πολυωνύμων ***n*** βαθμού με ***n*** ρίζες $x_1, x_2, \ldots, x_n$ (πιθανόν μιγαδικές ή με πολλαπλότητα):

$$y = P_n(x) = c_1 (x - x_1)*(x - x_2)*\ldots*(x - x_n)$$

- Η μέθοδος Horner για φωλιασμένους πολλαπλασιασμούς πολυωνύμων:

$$y = P_n(x) = ( \ldots ( ( ( c_1*x + c_2 )*x + c_3 )*x + c_4) \ldots )*x + c_{n+1}$$

**Παράδειγμα:**

$y = x^4 + 2x^3 - 7x^2 - 8x + 12$
$y = (x-1)*(x-2)*(x+2)*(x+3)$
$y = ( ( (x + 2)*x - 7)*x - 8)*x + 12$

### 2. Πολυώνυμα με MATLAB:

Αναπαράσταση πολυωνύμων από το διάνυσμα των συντελεστών του

```
p1 = [ 1,2,-7,-8,12], p2 = [ 2,3,5,9,5]
```

```
p1 =
     1     2    -7    -8    12
p2 =
     2     3     5     9     5
```

- **roots:** υπολογίζει όλες τις ρίζες ενός πολυωνύμου από τους συντελεστές του

```
r1 = roots(p1)', r2 = roots(p2)'
```

```
r1 =
   -3.0000   -2.0000    2.0000    1.0000
r2 =
   0.2500 - 1.5612i   0.2500 + 1.5612i  -1.0000            -1.0000
```

- **poly:** υπολογίζει τους συντελεστές ενός πολυωνύμου από τις ρίζες του

```
q1 = poly(r1), q2 = poly(r2)
% NB: coefficients q2 are different from coefficients of p2 by a factor
of 2
% The leading-order coefficient c1 is always normalized to 1.

q1 =    1.0000    2.0000   -7.0000   -8.0000   12.0000
q2 =    1.0000    1.5000    2.5000    4.5000    2.5000

      % rounding errors may occur in computations of roots
      % Wilkinson's example:
   roots(poly(1:20))'

ans =
Columns 1 through 11
20.0003   18.9970   18.0118   16.9695   16.0509   14.9319   14.0684
12.9472   12.0345   10.9836   10.0063
  Columns 12 through 20
8.9983    8.0003    7.0000    6.0000    5.0000    4.0000    3.0000
2.0000    1.0000


% large rounding error occurs when highly multiple roots are computed
% y = (x-1)^6
r = [ 1 1 1 1 1 1 ]; p = poly(r), rr = roots(p)'

p =     1    -6    15   -20    15    -6     1
rr =   1.0042 - 0.0025i   1.0042 + 0.0025i   1.0000 - 0.0049i   1.0000
+ 0.0049i   0.9958 - 0.0024i   0.9958 + 0.0024i
```

- **polyval:** υπολογίζει την τιμή του πολυωνύμου σε ένα δεδομένο σημείο

```
y1 = polyval(p1,2.5)
x = [ 0 : 0.2 : 1]; y = polyval(p2,x)

y1 =   18.5625
y =    5.0000    7.0272    9.6432   13.1072   17.7552   24.0000

% implementation of the Horner algorithm for nested multiplication:
function [y] = HornerMultiplication(p,x)
      [n,m] = size(x);
      y = p(1)*ones(n,m);
      for k = 2:length(p)
            y = y.*x + p(k);
      end
```

```
y1 = HornerMultiplication(p1,2.5)
y = HornerMultiplication(p2,x)

y1 =
   18.5625
y =
    5.0000    7.0272    9.6432   13.1072   17.7552   24.0000
```

• **polyder:** υπολογίζει τους συντελεστές της παραγώγου ενός δεδομένου πολυωνύμου

```
% P(x) = c(1) x^n + c(2) x^(n-1) + … + c(n) x + c(n+1)
% P'(x) = n c(1) x^(n-1) + (n-1) c(2) x^(n-2) + … + c(n)
Pder1 = polyder(p1), Pder2 = polyder(p2)

Pder1 =     4     6   -14    -8
Pder2 =     8     9    10     9
```

- **polyint:** υπολογίζει τους συντελεστές του ολοκληρώματος ενός δεδομένου πολυωνύμου

```
% P(x) = c(1) x^n + c(2) x^(n-1) + … + c(n) x + c(n+1)
% P'(x) = c(1) x^(n+1)/(n+1) + c(2) x^n/n + … + c(n) x^2/2 + c(n+1) x + c(n+2)
% c(n+2) are constant of integration (to be defined)
Pint1 = polyint(p1,10) % the constant of integration is 10
Pint2 = polyint(p2)  % the constant of integration is 0 (default)

Pint1 =    0.2000    0.5000   -2.3333   -4.0000   12.0000   10.0000
Pint2 =    0.4000    0.7500    1.6667    4.5000    5.0000         0
```

- **conv:** Υπολογίζει το γινόμενο δύο πολυωνύμων

```
% Let p1 be polynomial of order n, p2 be polynomial of order m,
% conv(p1,p2) is polynomial of order (n+m)
p = conv(p1,p2)

p =     2     7    -3   -18   -12   -57   -47    68    60
```

- **deconv:** Υπολογίζει τη διαίρεση δύο πολυωνύμων

```
% Let p1 be polynomial of order n, p2 be polynomial of order m,
% [pq,pr] = deconv(p1,p2) computes the quotient polynomial pq of order
(n-m)
% and the remainder polynomial pr of order (m-1) such that p1 = p2*pq +
pr
[pq,pr] = deconv(p1,p2)

pq =    0.5000
pr =         0    0.5000   -9.5000  -12.5000    9.5000
```

- **υπόλοιπο (residue):** υπολογίζει το ανάπτυγμα μερικών κλασμάτων (υπόλοιπα-residues)

```
  % Let p1 be polynomial of order n and p2 be polynomial of order m
  % [C,X,R] = residue(p1,p2) finds coefficients C of the residue terms,
  % locations of poles X and the remainder term of a partial fraction expansion
  % of the ratio of two polynomials p1(x)/p2(x).
     % Example: no multiple roots,
     %    p1(x)       C(1)       C(2)             C(n)
     %    ----  =  -------- + -------- + ... + -------- + R(x)
     %     p2(x)     x - X(1)   x - X(2)         x - X(n)

[C,X,R] = residue(p2,p1)

C =   -5.2000
       1.2500
       4.9500
      -2.0000

X =   -3.0000
      -2.0000
       2.0000
       1.0000
R =     2

% the inverse operation:
% from partial fraction expansion to the ratio of two polynomials
[q1,q2] = residue(C,X,R)

q1 =    2.0000    3.0000    5.0000    9.0000    5.0000
q2 =    1.0000    2.0000   -7.0000   -8.0000   12.0000
```

- **polyfit:** υπολογίζει τους συντελεστές των πολυωνύμων παρεμβολής ***n***-βαθμού που διέρχονται από ένα σύνολο (***n***+1) σημείων δεδομένων

```
x = [ 1,2,-2,-3,0]; y = [0,0,0,0,12];
p = polyfit(x,y,4) % 4 is the order of the polynomial through 5 data points
pp = poly([1,2,-2,-3]) % the same operation if roots of polynomial are
known

p =    1.0000    2.0000   -7.0000   -8.0000   12.0000
pp =     1     2    -7    -8    12
```

## 3. Πολυωνυμική παρεμβολή

**Πρόβλημα:** Για ένα σύνολο **(*n*+1)** σημείων δεδομένων:

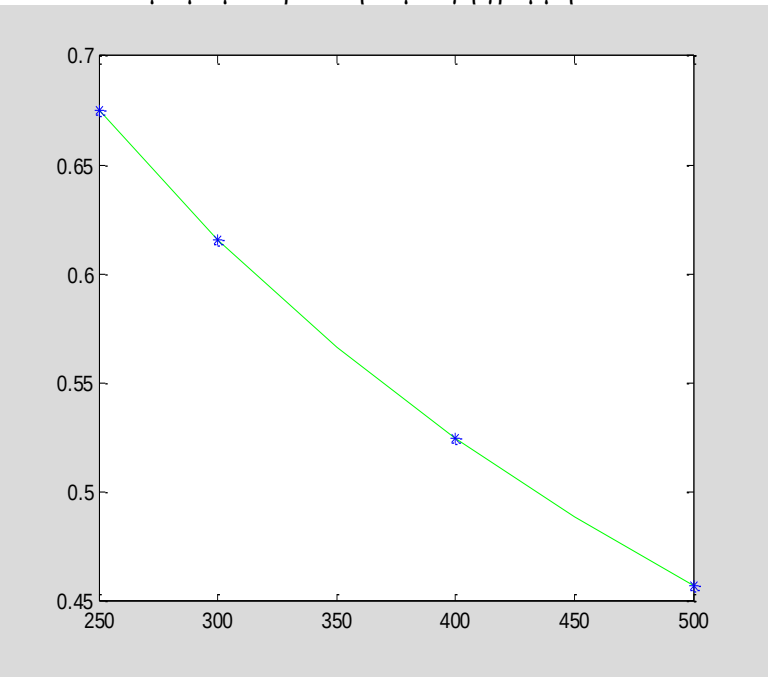
$$(x_1,y_1);\ (x_2,y_2);\ \ldots;\ (x_n,y_n);\ (x_{n+1},y_{n+1})$$

Βρείτε ένα πολυώνυμο βαθμού ***n***, $y = P_n(x)$, το οποίο διέρχεται από όλα τα δεδομένα σημεία (***n***+1). Υποθέστε ότι τα $x_1, x_2, \ldots, x_n, x_{n+1}$ ορίζονται με αύξουσα σειρά, π.χ. $x_1<x_2<\ldots<x_n<x_{n+1}$.
Το πολυώνυμο $y=P_n(x)$ ονομάζεται **πολυώνυμο παρεμβολής** (interpolation) για τα $x_1<x<x_{n+1}$ και **πολυώνυμο παρεκβολής** (extrapolation) για τα $x<x_1$ και $x>x_{n+1}$.

**Παράδειγμα:** Δίνεται ένα σύνολο με πέντε σημεία δεδομένων για ένα χαρακτηριστικό τάσης-ρεύματος μίας διόδου zener

| | | | | | |
|---|---|---|---|---|---|
| ***Τάση*** | ***-1.00*** | ***-0.75*** | ***-0.50*** | ***-0.25*** | ***-0.00*** |
| ***Ρεύμα*** | ***-14.58*** | ***-6.15*** | ***-1.82*** | ***-0.23*** | ***0.00*** |

Στο παρακάτω σχήμα, τα πέντε σημεία του παραδείγματος «συνδέονται» ή «παρεμβάλονται» με ένα πολυώνυμο $y = P_4(x)$ τετάρτου βαθμού. Τα σημεία δεδομένων σημειώνονται με μπλέ αστερίσκους και το πολυώνυμο με μια πράσινη συμπαγή γραμμή.




**Μέθοδοι προσδιορισμού του πολυωνύμου παρεμβολής:**

- Δυναμοσειρές ( παρεμβολή Vandermond)
- Παρεμβολή τύπου Lagrange
- Παρεμβολή τύπου Newton (Θα παρουσιαστεί στο κεφάλαιο 11)

## 4. Παρεμβολή Van der Monde:

Για να προσδιοριστούν οι $(n+1)$ συντελεστές $c_k$ του πολυωνύμου παρεμβολής $y = P_n(x)$ πρέπει να λυθούν οι εξής $(n+1)$ συνθήκες παρεμβολής:

$$P_n(x_k) = y_k, \kappa=1,\ldots,n+1.$$

Οι συνθήκες αυτές παράγουν ένα σύστημα $(n+1)$ γραμμικών εξισώσεων με $(n+1)$ αγνώστους, τους συντελεστές του πολυωνύμου παρεμβολής $\{c_i\}_{i=1}^{n+1}$:

$$c_1 (x_k)^n + c_2 (x_k)^{n-1} + \ldots + c_{n-1} (x_k)^2 + c_n x_k + c_{n+1} = y_k \quad , k=1,\ldots,n+1.$$

**Παράδειγμα**

$$p(x) = c_1 x^3 + c_2 x^2 + c_3 x + c_4$$

$$x_1 = 250: \quad \mathrm{p}(x_1) = 0.675 = c_1(250)^3 + c_2(250)^2 + c_3(250) + c_4$$

$$x_1 = 300: \quad \mathrm{p}(x_1) = 0.616 = c_1(300)^3 + c_2(300)^2 + c_3(300) + c_4$$

$$x_1 = 400: \quad \mathrm{p}(x_1) = 0.525 = c_1(400)^3 + c_2(400)^2 + c_3(400) + c_4$$

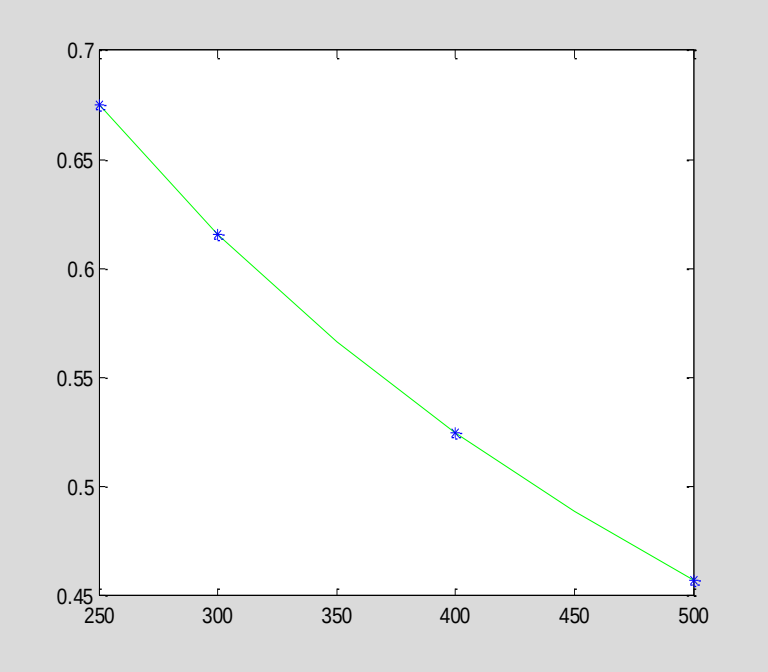
$$x_1 = 500: \quad \mathrm{p}(x_1) = 0.457 = c_1(500)^3 + c_2(500)^2 + c_3(500) + c_4$$

$$\begin{bmatrix} (250)^3 & (250)^2 & 250 & 1 \\ (300)^3 & (300)^2 & 300 & 1 \\ (400)^3 & (400)^2 & 400 & 1 \\ (500)^3 & (500)^2 & 500 & 1 \end{bmatrix} \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} 0.675 \\ 0.616 \\ 0.525 \\ 0.457 \end{Bmatrix}$$

Το παραπάνω γραμμικό σύστημα μπορεί να επιλυθεί με τη χρήση του επιλυτή της MATLAB που συμβολίζεται με «\». Ο παρακάτω κώδικας σχηματίζει και λύνει το σύστημα Van der Monde για το παραπάνω παράδειγμα:

```
x = [ 250,300,400,500]; A = vander(x)
    y = [ 0.675, 0.616, 0.525, 0.457]; c = A\y'; c'
    xInt = 250 : 50 : 500; yInt = polyval(c,xInt);
    plot(xInt,yInt,'g',x,y,'b*');cond(A)
```

```
A =
    15625000       62500         250           1
    27000000       90000         300           1
    64000000      160000         400           1
   125000000      250000         500           1
ans =
   -0.0000    0.0000   -0.0029    1.1830
ans =
  9.3065e+009
```




Στην περίπτωση του παραδείγματος, το πολυώνυμο παρεμβολής είναι

$$p(x) = -0.0000000026x^3 + 0.0000042700x^2 - 0.0029370000x + 1.1830000000$$

**Σημείωση**: Για μεγάλες τιμές του ***n,*** οι πίνακες Van der Monde είναι "κακής κατάστασης ("ill-conditioned ", δείτε το κεφάλαιο 7) και οι επιλυτές της γραμμικής άλγεβρας παράγουν ανακριβές αριθμητικό αποτέλεσμα. Όπως θα μάθουμε στο κεφάλαιο 7, η συμπεριφορά του συστήματος μπορεί να εκτιμηθεί από το μέγεθος του «αριθμού κατάστασης» του πίνακα A. Στο παραπάνω παράδειγμα ο αριθμός αυτός είναι `9.3065e+009` που είναι `πολύ μεγάλος` άρα το σύστημα είναι κακής κατάστασης. Ο προσδιορισμός των πολυωνύμων παρεμβολής τύπου Lagrange και Newton δεν απαιτούν να επιλύσουμε γραμμικά συστήματα εξισώσεων. Την μέθοδο Newton θα την αναπτύξουμε στο Κεφάλαιο 11.

## 5. <u>Παρεμβολή Lagrange:</u>

Μία βάση του γραμμικού χώρου των πολυωνύμων βαθμού ***n*** αποτελούν τα πολυώνυμα του Lagrange που ορίζονται ως εξής:

$$L_{n,j}(x) = \prod_{\substack{i=1 \\ i \neq j}}^{n+1} \frac{(x - x_i)}{(x_j - x_i)}$$

$L_{n,j}(x)$ είναι πολυώνυμο ***n*** βαθμού
$L_{n,j}(x_i) = 0$ για κάθε $i \neq j$
$L_{n,j}(x_j) = 1$

Το πολυώνυμο παρεμβολής $y = P_n(x)$ δίδεται από τον παρακάτω τύπο σαν συνάρτηση των πολυωνύμων Lagrange $L_{n,j}(x)$:

$$P_n(x) = \sum_{j=1}^{n+1} y_j L_{n,j}(x) = y_1 L_{n,1}(x) + \ldots + y_{n+1} L_{n,n+1}(x)$$

**Παρατηρήσεις**: Το πολυώνυμο παρεμβολής $y = P_n(x)$ είναι $n$ βαθμού και διέρχεται από όλα τα $(n+1)$ σημεία δεδομένων. Μπορεί να αποδειχθεί ότι το πολυώνυμο παρεμβολής είναι μοναδικό, δηλαδή τα πολυώνυμα $y = P_n(x)$ που βρέθηκαν με τη μέθοδο Van der Monde και με τη μέθοδο Lagrange είναι τα ίδια. Παρακάτω ορίζουμε το πολυώνυμο Lagrange σε μορφή MATLAB συνάρτησης και την οποία εφαρμόζουμε στο προηγούμενο παράδειγμα.

```
function [yi] = LagrangeInter(x,y,xi)
   % Lagrange interpolation algorithm
   % x,y - row-vectors of (n+1) data values (x,y)
   % xi - a row-vector of values of x, where the polynomial y = Pn(x) is
evaluated
   % yi - a row-vector of values of y, evaluated with y = Pn(x)

n = length(x) - 1; % order of interpolation polynomial y = Pn(x)
ni = length(xi); % number of points where the interpolation is to be
evaluated
L = ones(ni,n+1); % the matrix for Lagrange interpolating polynomials
L_(n,j)(x)
                  % L has (n+1) columns for each point j = 1,2,...,n+1
                  % L has ni rows for each point of xi
for j = 1 : (n+1)
    for i = 1 : (n+1)
       if (i ~= j)
L(:,j) = L(:,j).*(xi' - x(i))/(x(j)-x(i));
end
end
end

yi = y*L';

x = [ 250,300,400,500];
    y = [ 0.675, 0.616, 0.525, 0.457];
    xInt = 250 : 50 : 500; yInt = LagrangeInter(x,y,xInt);
    plot(xInt,yInt,'g',x,y,'b*');
```

**Παραδείγματα παρεμβολής Lagrange**

1ου βαθμού πολυώνυμο Lagrange για τα σημεία παρεμβολής $\{(x_i, f(x_i)\}_{i=1}^{2}$

$$p_1(x) = L_1 f(x_1) + L_2(x) f(x_2) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2)$$

2ου βαθμού πολυώνυμο Lagrange για $\{(0,1),(1,1.71828),(4,54.5982)\}$

$$p_2(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3)$$

$$p_2(x) = \frac{(x-1)(x-4)}{(0-1)(0-4)} f(0) + \frac{(x-0)(x-4)}{(1-0)(1-4)} f(1) + \frac{(x-0)(x-1)}{(4-0)(4-1)} f(4)$$

$$p_2(2) = \frac{(2-1)(2-4)}{(0-1)(0-4)}(1.0) + \frac{(2-0)(2-4)}{(1-0)(1-4)}(1.71828) + \frac{(2-0)(2-1)}{(4-0)(4-1)}(54.5982) = 12.224067$$

3ου βαθμού πολυώνυμο Lagrange για $\{(0,1),(1,1.71828),(4,54.5982),(3,20.08554)\}$

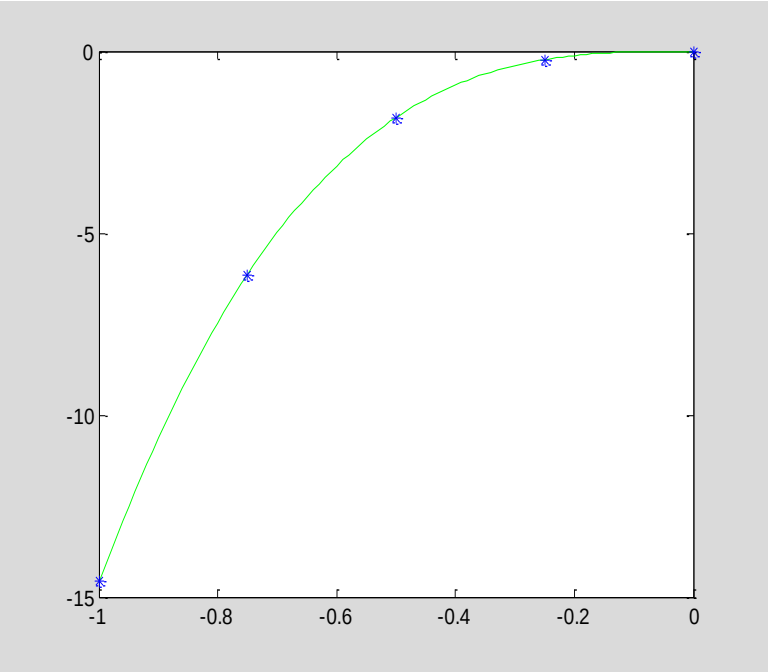
$$p_3(x) = \frac{(x - x_2)(x - x_3)(x - x_4)}{(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)} f(x_1) + \frac{(x - x_1)(x - x_3)(x - x_4)}{(x_2 - x_1)(x_2 - x_3)(x_2 - x_4)} f(x_2)$$
$$+ \frac{(x - x_1)(x - x_2)(x - x_4)}{(x_3 - x_1)(x_3 - x_2)(x_3 - x_4)} f(x_3) + \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_4 - x_1)(x_4 - x_2)(x_4 - x_3)} f(x_4)$$

$$p_3(x) = \frac{(x-1)(x-4)(x-3)}{(0-1)(0-4)(0-3)} f(0) + \frac{(x-0)(x-4)(x-3)}{(1-0)(1-4)(1-3)} f(1)$$
$$+ \frac{(x-0)(x-1)(x-3)}{(4-0)(4-1)(4-3)} f(4) + \frac{(x-0)(x-1)(x-4)}{(3-0)(3-1)(3-4)} f(3)$$

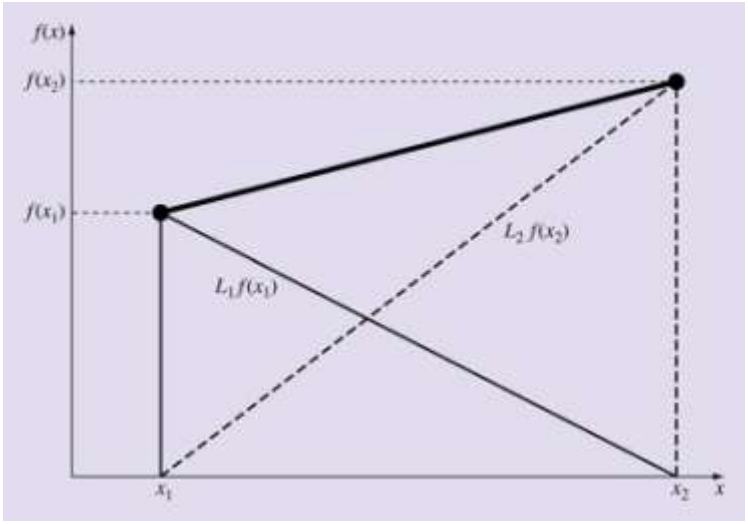
$$p_3(2) = \frac{2}{-12}(1.0) + \frac{4}{6}(2.71828) + \frac{-2}{12}(54.5982) + \frac{-4}{-6}(20.08554) = 5.936187$$

Παράδειγμα 4ου βαθμού πολυώνυμο Lagrange σε μορφή MATLAB και η γραφική του παράσταση

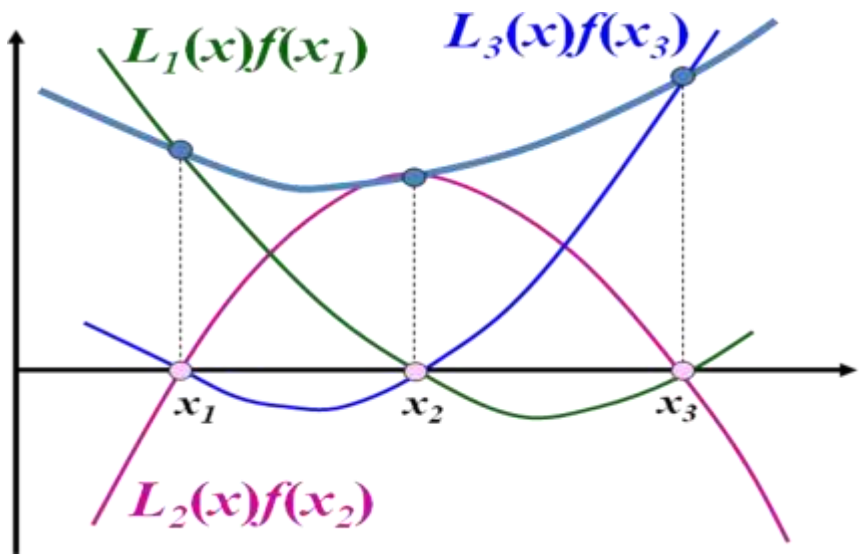
```
x = [ -1,-0.75,-0.5,-0.25,-0];
y = [ -14.58,-6.15,-1.82,-0.23,-0.00];
xInt = -1 : 0.01 : 0;
yInt = LagrangeInter(x,y,xInt);
plot(xInt,yInt,'g',x,y,'b*');
```




Παρακάτω παρουσιάζουμε γραφικά την «βάση» του **γραμμικού πολυωνύμου** παρεμβολής Lagrange.

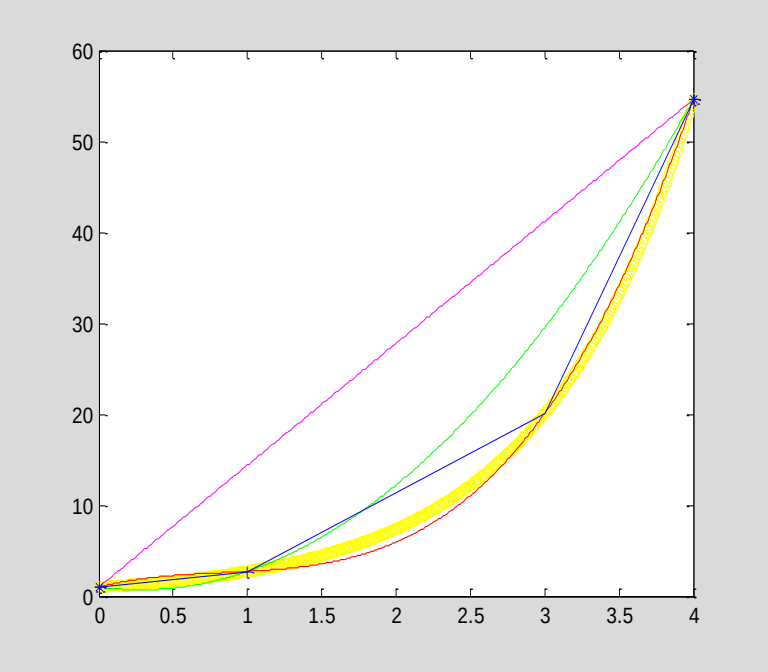



Παρακάτω παρουσιάζουμε γραφικά την «βάση» του **τετραγωνικού πολυωνύμου** παρεμβολής Lagrange.




**Παράδειγμα**: Γραφική παράσταση της συνάρτησης $f(x) = e^x$ και των πολυωνύμων παρεμβολής 1ου, 2ου και 3ου βαθμού στο διάστημα [0,4]

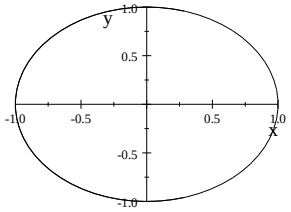
```
%grafikh parastash ths synarthshs exp(x)
xInt = 0 : 0.01 : 4; yExact = exp(xInt);
plot(xInt,yExact,'yo');
hold on;
% polyonimo Lagrange bathmoy 1
x = [0, 4];
y = exp(x);
yInt = LagrangeInter(x,y,xInt);
plot(xInt,yInt,'m',x,y,'b*');
hold on;
%polyonimo Lagrange bathmoy 2
x = [0,1, 4]; y = exp(x);
yInt = LagrangeInter(x,y,xInt);
plot(xInt,yInt,'g',x,y,'b+');
hold on;
%polyonimo Lagrange bathmoy 3
x = [0,1,3, 4]; y = exp(x);
yInt = LagrangeInter(x,y,xInt);
plot(xInt,yInt,'r',x,y,'b-');
```



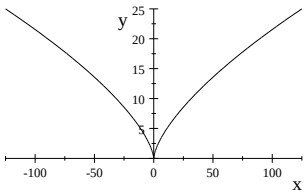

## 6. 2Δ παραμετρικές καμπύλες

Παραμετρική καμπύλη είναι μια διανυσματική συνάρτηση $P(t)$, δηλαδή μια συνάρτηση $P: R \to R^n$. Μπορούμε να γράψουμε την $P$ από τον τύπο της ή διαμέσου των συντεταγμένων της: $P(t) = (x(t), y(t))$. Παραδείγματα τέτοιων καμπυλών ορισμένων διαμέσου συντεταγμένων είναι:

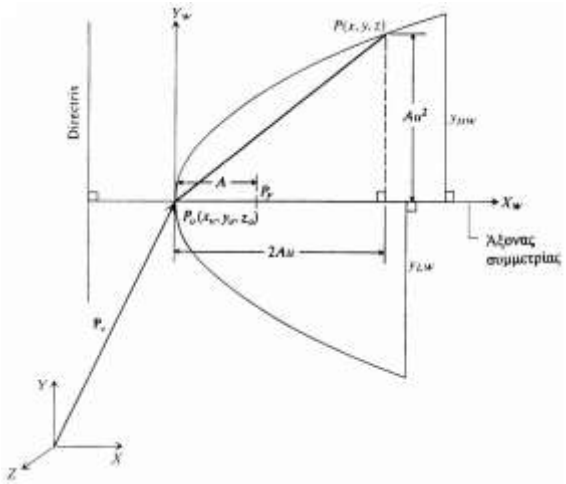
$P(t) = (\cos(t), \sin(t))$



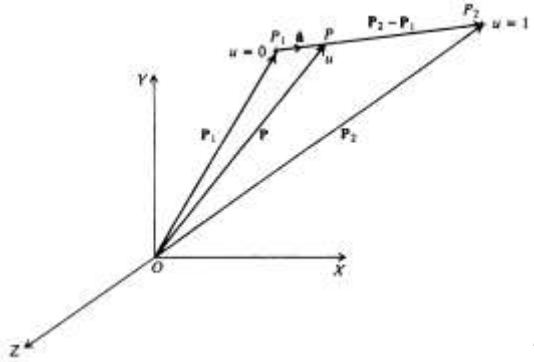

$P(t) = (t^3, t^2)$




$x = x_\upsilon + 2Au$ , $y = y_\upsilon + 2Au^2$ , $z = z_u$




Παραδείγματα καμπυλών ορισμένων σαν ένα συνδυασμός σημείων της καμπύλης: Για παράδειγμα αν $\vec{P}_1 = (x_1, y_1)$ και $\vec{P}_2 = (x_2, y_2)$ είναι δύο σημεία μιας ευθείας γραμμής τότε η παραμετρική μορφή της ευθείας μπορεί να οριστεί από την διανυσματική συνάρτηση $\vec{P} = \vec{P}_1 + u(\vec{P}_2 - \vec{P}_1) = (1-u)\vec{P}_1 + u\vec{P}_2$ για $0 \le u \le 1$



ή από τις συντεταγμένες της $x = x_1 + u(x_2 - x_1)$ , $y = y_1 + u(y_2 - y_1)$.

Οι συντελεστές $(1-u)$ και $u$ των σημείων $\overleftrightarrow{P}_1$ και $\overleftrightarrow{P}_2$ στα γραφικά λέγονται *συναρτήσεις μείξης (blending)*. Γενικά, για μια καμπύλη $\overleftrightarrow{P}(t) = f_0(t)\overleftrightarrow{P}_0 + f_1(t)\overleftrightarrow{P}_1 + ... + f_n(t)\overleftrightarrow{P}_n$ και ένα σύνολο σημείων $\left\{\vec{P}_i = (x_i, y_i)\right\}_{i=0}^{n}$ έχουμε

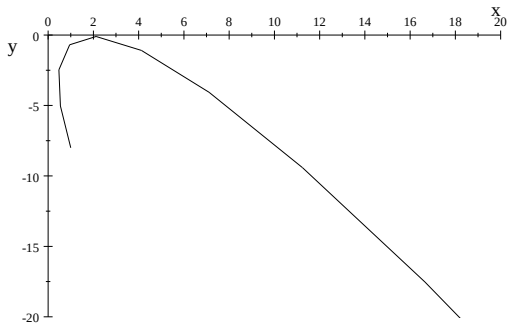
$x(t) = f_0(t)x_0 + f_1(t)x_1 + ... + f_n(t)x_n$ και $y(t) = f_0(t)y_0 + f_1(t)y_1 + ... + f_n(t)y_n$ .

## 7. Πολυωνυμικές καμπύλες

Πολυωνυμικές καμπύλες είναι αυτές που οι συντεταγμένες τους είναι πολυώνυμα. Παράδειγμα τέτοιας καμπύλης μαζί με την γραφική παράσταση της είναι

$$P(t) = (1-t)^3\begin{pmatrix}4\\1\end{pmatrix} + 3(1-t)^2\begin{pmatrix}0\\-3\end{pmatrix} + 2(1-t)t^2\begin{pmatrix}0\\3\end{pmatrix} + t^3\begin{pmatrix}4\\-1\end{pmatrix}$$ ή

$$P(t) = ((1-t)^3 + 4t^3, (1-t)^3 - 9(1-t)^2 + 6(1-t)t^2 - t^3)$$



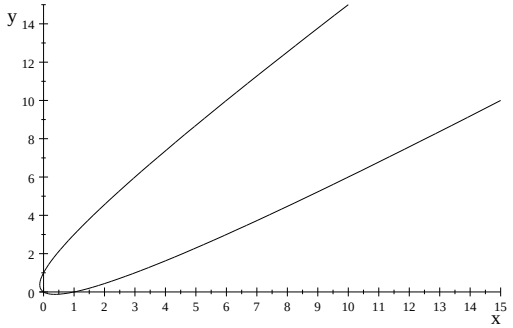

Ο βαθμός μιας πολυωνυμικής καμπύλης είναι ο μέγιστος βαθμός των πολυωνυμικών συντεταγμένων της.

## 8. Παρεμβολή με πολυωνυμικές παραμετρικές καμπύλες

Εδώ εξετάζουμε την εύρεση μιας παραμετρικής καμπύλης $\vec{P}(t)$ που διέρχεται από ένα σύνολο δεδομένων σημείων. Έστω, τα δεδομένα σημεία $\vec{P_0}, \vec{P_1}, ..., \vec{P_n}$ που αντιστοιχούν στις τιμές των παραμέτρων $t_0, t_1, ..., t_n$. Να βρεθεί η παραμετρική καμπύλη $\vec{P}(t)$ έτσι ώστε $\vec{P}(t_i) = \vec{P}_i$. Παρατηρούμε ότι αν επιλέξουμε την παραμετρική παράσταση της καμπύλης $\vec{P}(t) = f_0(t)\vec{P}_0 + f_1(t)\vec{P}_1 + ... + f_n(t)\vec{P}_n$ όπου $f_i(t_j) = \{^{1 \text{ αν } i=j}_{0 \text{ αν } i\neq j} \quad j = 0, \ldots n$ τότε η καμπύλη $\vec{P}(t)$ διέρχεται από το σύνολο των δεδομένων σημείων. Δεν είναι δύσκολο να αποδείξει κανείς ότι οι «blending» συναρτήσεις $\{f_i\}_{i=0}^n$ που ικανοποιούν τις παραπάνω συνθήκες είναι οι $f_i(t) = \prod_{j\neq i} \frac{t-t_j}{t_i-t_j}$ .

**Άσκηση**

Να βρεθεί η τετραγωνική καμπύλη που παρεμβάλει τα σημεία $\vec{P}(-1) = (1,0)$ , $\vec{P}(0) = (0,0)$ , $\vec{P}(1) = (0,1)$ . Να αποδείξετε ότι η παραμετρική παράσταση της καμπύλης και το γράφημα της είναι $P(t) = \frac{1}{2}(t^2 - t)(1,0) - (t^2 - 1)(0,0) + \frac{1}{2}(t^2 + t)(0,1) = \left(\frac{1}{2}t^2 - \frac{1}{2}t, \quad \frac{1}{2}t^2 + \frac{1}{2}t\right)$




## 9. Καμπύλες Bezier

Οι καμπύλες Bezier πήραν το όνομα τους από τον δημιουργό τους, Dr. Pierre Bezier, ένα μηχανικό της εταιρείας αυτοκινήτων Renault. Οι καμπύλες αυτές επιτρέπουν στους σχεδιαστές να τις χρησιμοποιούν χωρίς να διαθέτουν σημαντική γνώση μαθηματικών. Θα παρουσιάσουμε τις κυβικές Bezier καμπύλες που βασίζονται σε τέσσερα δεδομένα σημεία: $P_0, P_1, P_2, P_4$.

Θεωρούμε την παραμετρική παράσταση της κυβικής καμπύλης Bezier
$P(t) = m_0(t)P_0 + m_1(t)P_1 + m_2(t)P_0 + m_3(t)P_1$ όπου $0 \le t \le 1$ και επιλέγουμε τις συναρτήσεις μείξης $\{m_0(t), m_1(t), m_2(t), m_3(t)\}$ έτσι ώστε

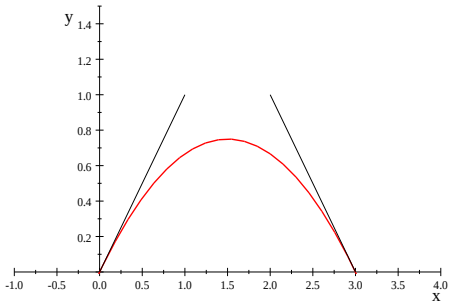
α) $m_0(0) = 1, m_1(0) = m_2(0) = m_3(0) = 0$ , δηλαδή $P(0) = P_0$ ,

β) $m_0(1) = m_1(1) = m_2(1) = 0$ , $m_3(1) = 1,$ δηλαδή $P(1) = P_3$ ,

γ) $P(t)$ εφάπτεται της ευθείας $\overline{P_0P_1}$ στο $P_0$ και της ευθείας $\overline{P_2P_3}$ στο $P_3$ .

**Άσκηση**: Να αποδεχθεί ότι οι συναρτήσεις $\{m_i\}$ (οι οποίες είναι γνωστές ως πολυώνυμα Bernstein) είναι $m_0(t) = (1-t)^3$ $m_1(t) = 3t(1-t)^2$ , $m_2(t) = 3t^2(1-t)$ , $m_3(t) = t^3$ .

**Παράδειγμα**: $P(t) = m_0(t)(0,0) + m_1(t)(1,1) + m_2(t)(2,1) + m_3(t)(3,0)$
$:\left(3t(t-1)^2 - 6t^2(t-1) + 3t^3, \quad 3t(t-1)^2 - 3t^2(t-1)\right)$



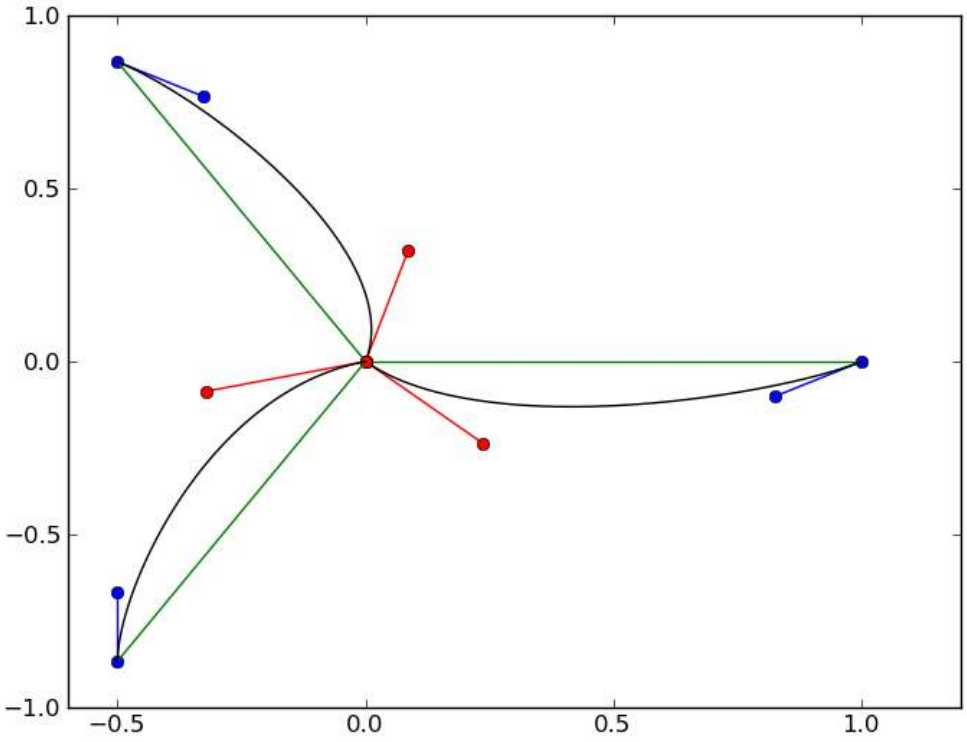

```
function r = bezier(t,node,ctrl)
% BEZIER Cubic Bezier curve.
% Input:
%   t      parameter evaluation values (between 0 and 1)
%   node   coordinates of endpoints (two colums)
%   ctrl   coordinates of control points (two colums)
% Output:
%   r      points on curve (one column per t value)

t = t(:).';                          % make it a row vector

% Bernstein polyomials.
B0 = (1-t).^3;
B1 = 3*t.*(1-t).^2;
B2 = 3*t.^2.*(1-t);
B3 = t.^3;

r = node(:,1)*B0 + ctrl(:,1)*B1 + ctrl(:,2)*B2 + node(:,2)*B3;
```

**Παράδειγμα**
Περιγραφή καμπυλών Bezier στο περιβάλλον Python και το γράφημα τους κάνοντας χρήση της Python γραφικής βιβλιοθήκης matplotlib

```
import scipy as sp
import matplotlib.pyplot as plt

#### Inputs

#A list of P0's and P3's. Must be the same length
origins = [[1,0],[-0.5,sp.sqrt(3)/2], [-0.5,-sp.sqrt(3)/2]]
destinations = [[0,0],[0,0],[0,0]]

#The angle the control point will make with the green line
blue_angle = sp.pi/6
red_angle = sp.pi/4

#And the lengths of the lines (as a fraction of the length of the green
one)
blue_len = 1./5
red_len = 1./3

### Workings

#Generate the figure
fig = plt.figure()
ax = fig.add_subplot(111)
ax.hold(True)

#Setup the parameterisation
t = sp.linspace(0,1,100)

for i in xrange(len(origins)):
    #Read in the origin & destination points
    POx,POy = origins[i][0], origins[i][1]
    P3x,P3y = destinations[i][0], destinations[i][1]

    #Add those to the axes
    ax.plot(POx,POy, 'ob')
    ax.plot(P3x,P3y, 'or')
    ax.plot((POx,P3x),(POy,P3y), 'g')

    #Work out r and theta (as if based at P3)
    r = ((POx-P3x)**2 + (POy-P3y)**2)**0.5
    theta = sp.arctan2((POy-P3y),(POx-P3x))

    #Find the relevant angles for the control points
    aO =  theta + blue_angle+ sp.pi
    aD = theta - red_angle

    #Work out the control points
    P1x, P1y = POx+ blue_len*r*sp.cos(aO), POy + blue_len*r*sp.sin(aO)
    P2x, P2y = P3x+ red_len*r*sp.cos(aD), P3y + red_len*r*sp.sin(aD)

    #Plot the control points and their vectors
    ax.plot((P3x,P2x),(P3y,P2y), 'r')
    ax.plot((POx,P1x),(POy,P1y), 'b')
    ax.plot(P1x, P1y, 'ob')
    ax.plot(P2x, P2y, 'or')

    #Use the Bezier formula
```

```
    Bx = (1-t)**3*POx + 3*(1-t)**2*t*P1x + 3*(1-t)*t**2*P2x + t**3*P3x
    By = (1-t)**3*POy + 3*(1-t)**2*t*P1y + 3*(1-t)*t**2*P2y + t**3*P3y

    #Plot the Bezier curve
    ax.plot(Bx, By, 'k')

#Save it
plt.savefig('totally-awesome-bezier.png')
plt.show()
```

## ΑΠΟΤΕΛΕΣΜΑ

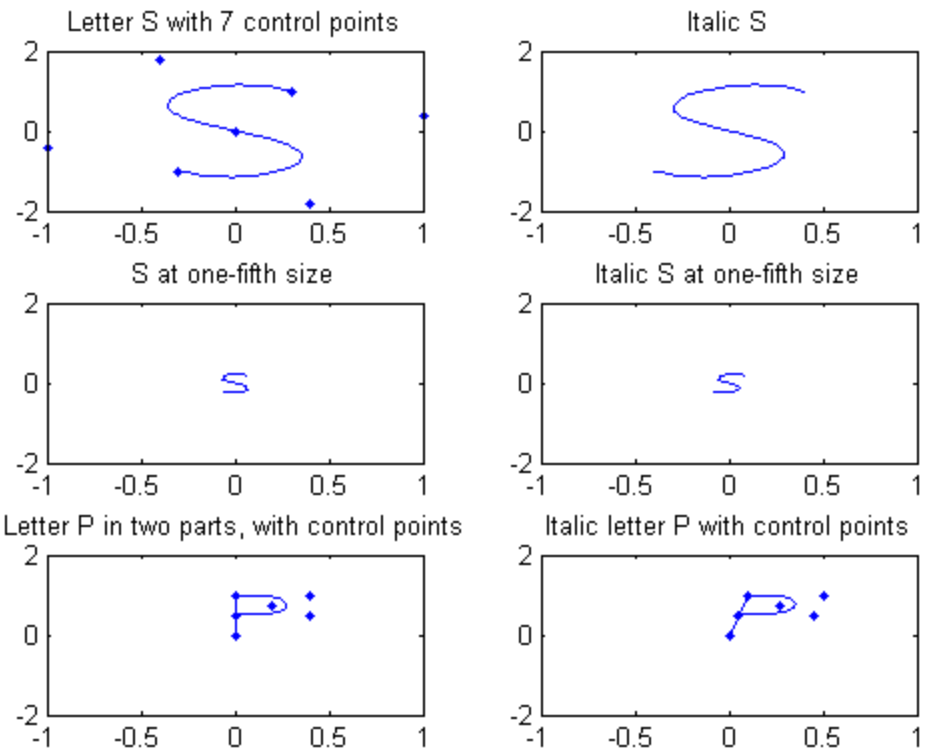



### Εφαρμογές

Οι καμπύλες Bezier χρησιμοποιούνται στο σχεδιασμό γραμματοσειρών έτσι ώστε να είναι ανεξάρτητες από εκτυπωτές και άλλες συσκευές απεικόνισης. Ο παρακάτω κώδικας παρουσιάζει παραδείγματα Bezier καμπυλών για γραμματοσειρές. Μια άλλη εφαρμογή είναι ο χώρος των γραφικών και γεωμετρικός σχεδιασμός που θα τον δούμε σε επόμενο κεφάλαιο.

```
% Bezier_examples.m displays some examples of Bezier curves


clf;                                                % clear the figure


subplot(3,2,1);


S = [[0.3 1]; [-0.4 1.8]; [-1 -0.4]; [0 0]]';  % Control points for S
```

```
S = [S -S(:,3) -S(:,2) -S(:,1)];                % More points by
symmetry
Bezier(S,100,1);                                % Plot Bezier curve for
S
title('Letter S with 7 control points');
axis([-1 1 -2 2]);


subplot(3,2,2);


A = [[1 0.1]; [0 1]];                           % Matrix for italic
shear
Bezier(A*S,100,0);                              % Bezier curve for
sheared S
title('Italic S');
axis([-1 1 -2 2]);


subplot(3,2,3);


D = [[0.2 0]; [0 0.2]];                         % 1/5 scale matrix
Bezier(D*S,100,0);                              % Bezier curve for
sheared S
title('S at one-fifth size');
axis([-1 1 -2 2]);


subplot(3,2,4);


D = [[0.2 0]; [0 0.2]];                         % 1/5 scale matrix
Bezier(D*A*S,100,0);                            % Bezier curve for
sheared S
title('Italic S at one-fifth size');
axis([-1 1 -2 2]);


subplot(3,2,5);


Post = [[0 0]; [0 1]]';                         % post for letter P
Loop = [[0 1]; [0.4 1]; [0.2 0.75]; [0.4 0.5]; [0 0.5]]'; % loop for
letter P
Bezier(Post,10,1);
Bezier(Loop,100,1);
title('Letter P in two parts, with control points');
axis([-1 1 -2 2]);


subplot(3,2,6);
Bezier(A*Post,10,1);
Bezier(A*Loop,100,1);
title('Italic letter P with control points');
axis([-1 1 -2 2]);
```




**10. Αναφορές**

1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
3. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
4. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
5. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
6. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.
7. An introduction to splines for use in computer graphics and geometric modeling, R.H. Bartels, J.C. Beatty, B.A. Barsky, Morgan Kaufmann Publishers, Inc., 1987.

# ΚΕΦΑΛΑΙΟ 2: ΤΜΗΜΑΤΙΚΗ ΠΟΛΥΩΝΥΜΙΚΗ ΠΑΡΕΜΒΟΛΗ

## Περιεχόμενα



## 1. Τμηματικά πολυώνυμα

**Παρατήρηση**: Οπως τα παρακάτω παραδείγματα αποδεικνύουν, τα πολυώνυμα παρεμβολής μεγάλου βαθμού δεν είναι ικανοποιητικά για την προσέγγιση συναρτήσεων ή δεδομένων. Για το σκοπό αυτό ορίζουμε μια νέα κατηγορία προσεγγιστικών συναρτήσεων που αναφέρονται σαν τμηματικά πολυώνυμα ή splines.

**Ορισμός:** Γενικά, μια συνάρτηση λέγεται τμηματικό πολυώνυμο ή *spline* βαθμού ***k*** ως προς την διαμέριση $x_1 < x_2 < \cdots < x_n$ του πεδίου ορισμού της $[a = x_1, b = x_n]$ όταν έχει τις παρακάτω ιδιότητες:

(1) $s \in [x_1, x_n]$;

(2) όλες οι $j$ βαθμού παράγωγοι $s^{(j)}$, $j = 0, 1, 2, \cdots, k-1$ είναι συνεχείς στο $[x_1, x_n]$ ;

(3) $s$ είναι πολυώνυμο βαθμού $\leq k$ σε κάθε υποδιάστημα $[x_i, x_{i+1}]$.

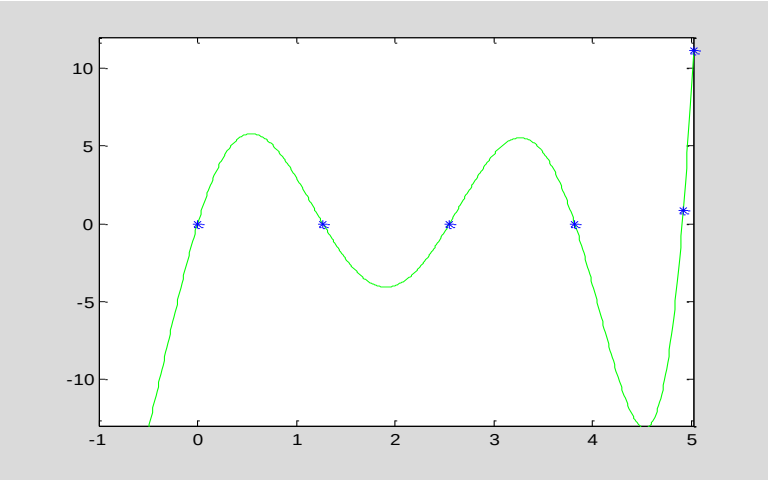
Τα μειονεκτήματα της πολυωνυμικής παρεμβολής:

**Παράδειγμα 1.1**: Δίνεται ένα σύνολο με επτά σημεία δεδομένων για το χαρακτηριστικό τάσης-ρεύματος μιας διόδου zener.

| ***Τάση*** | ***-1.00*** | ***0.00*** | ***1.27*** | ***2.55*** | ***3.82*** | ***4.92*** | ***5.02*** |
|---|---|---|---|---|---|---|---|
| ***Ρεύμα*** | ***-14.58*** | ***0.00*** | ***0.00*** | ***0.00*** | ***0.00*** | ***0.88*** | ***11.17*** |

Αν συνδέσουμε τα επτά σημεία με ένα πολυώνυμο $y = P_6(x)$ 6ου βαθμού, η συνεχής πολυωνυμική παρεμβολή που λαμβάνουμε παρουσιάζει απότομες αυξομειώσεις μεταξύ των σημείων δεδομένων όπως παρατηρούμε στο σχήμα 1.1.

```
x=[-1, 0, 1.27, 2.55, 3.82, 4.92, 5.02];
y=[-14.58, 0., 0., 0., 0., .88, 11.17];
xInt = -1 : 0.01 : 5.02; c = polyfit(x,y,6);
yInt = polyval(c,xInt);
plot(xInt,yInt,'g',x,y,'b*');axis([-1,5.02,-13,12]);
```

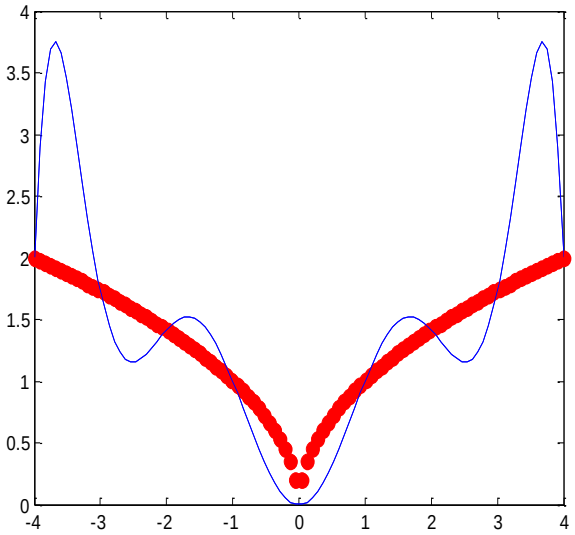



**Σχήμα 1.1**: Γραφική παράσταση πολυωνύμου παρεμβολής βαθμού 6ου για τα δεδομένα που χαρακτηρίζουν την τάση-ρεύματος μιας διόδου zener.

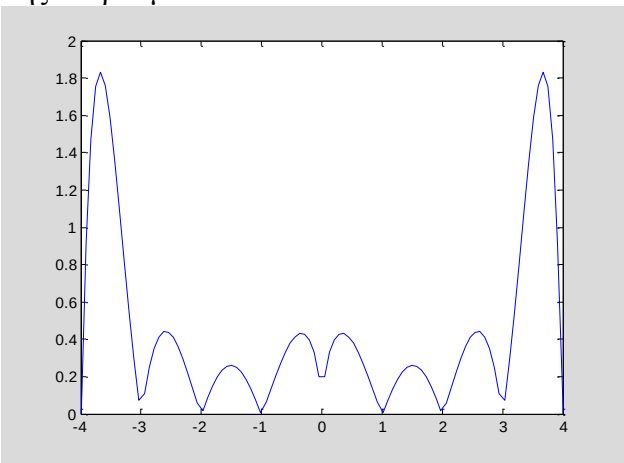
Το πολυώνυμο $y = P_6(x)$, που είναι 6ου βαθμού, μπορεί να έχει πέντε «ακρότατα» σημεία (μέγιστα και ελάχιστα). Πραγματικά και τα πέντε ακρότατα σημεία φαίνονται στη παραπάνω εικόνα. Αυτό ονομάζεται **αυξομείωση των τιμών του πολυωνύμου**. Αν το $n$ είναι μεγάλο, τα πολυώνυμα παρεμβολής εμφανίζουν μεγάλα σφάλματα και σπάνια χρησιμοποιούνται για αναπαράσταση (προσέγγιση) καμπυλών.

**Παράδειγμα 1.2**: Θεωρείστε την συνάρτηση $f(x) = sqrt(abs(x))$ και βρείτε το πολυώνυμο που παρεμβάλει την συνάρτηση στα σημεία $-4, -3, -2, -1, 0, 1, 2, 3, 4$ και το σφάλμα του (δηλαδή την διαφορά μεταξύ της δοθείσης συνάρτησης και του πολυωνύμου παρεμβολής). Ο παρακάτω κώδικας χρησιμοποιεί την MatLab συνάρτηση **polyfit** για να βρει το πολυώνυμο παρεμβολής 8ου βαθμού που διέρχεται από τα δοθέντα σημεία, υπολογίζει το σφάλμα, και παριστά γραφικά τα αποτελέσματα.

```
x = linspace(-4,4,9); y = sqrt(abs(x));
% given function with 9 equispaced data points
c = polyfit(x,y,8); xInt = linspace(-4,4,100); yInt = polyval(c,xInt);
            % polynomial interpolation on a tense grid
yExact = sqrt(abs(xInt)); eLocal = abs(yExact-yInt);
            % exact values and local error e(x) of the polynomial
interpolation
figure(1)
hold on
plot(xInt,yExact,'ro','MarkerSize',8','MarkerFaceColor',[1,0,0])
plot(xInt,yInt,'b')
figure(3)
plot(xInt,eLocal)
```




**Σχήμα 1.2**: Η κόκκινη γραμμή παριστά την συνάρτηση **sqrt(abs(x))** και η μπλε το πολυώνυμο παρεμβολής 8ου βαθμού.



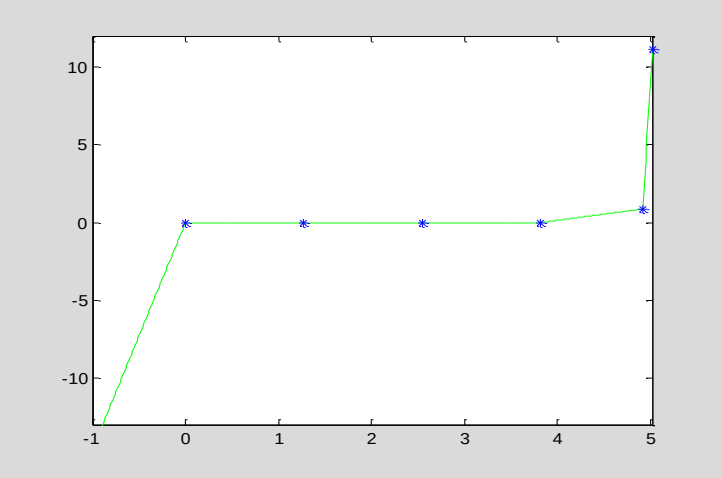

**Σχήμα 1.3**: Η μπλε καμπύλη παριστά το σφάλμα, δηλαδή την διαφορά μεταξύ της συνάρτησης sqrt(abs(x)) και το πολυώνυμο παρεμβολής 8ου βαθμού σε 100 σημεία.

Η γραφική λύση του παραπάνω προβλήματος δίδεται στο σχήμα 1.2 και το σφάλμα της προσέγγισης στο σχήμα 1.3 από τα οποία συμπεραίνουμε ότι η προσέγγιση της δοθείσης συνάρτησης δεν είναι ικανοποιητική όπως δείχνει και το μέγεθος του σφάλματος.

Παρακάτω θα δούμε ότι η τμηματική πολυωνυμική παρεμβολή που αποτελείται τοπικά από πολυώνυμα μικρότερου βαθμού αναπαριστούν μια επαρκώς ομαλή γραμμή ***y = S(x).***

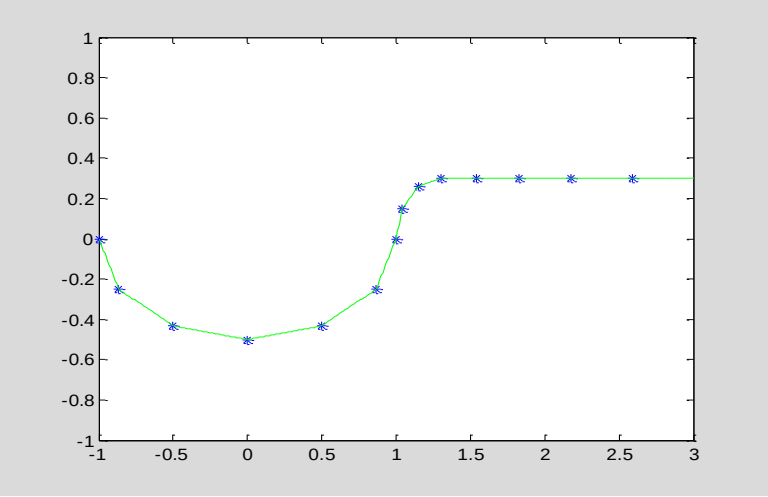
Η MatLab συνάρτηση **interp1** προσδιορίζει μια τμηματική παρεμβολή ως προς ένα σύνολο δεδομένων σημείων και την υπολογίζει ανάμεσα στα σημεία παρεμβολής. Παρακάτω δίνουμε παραδείγματα εφαρμογής της παρεμβολής με splines για τα δεδομένα που χαρακτηρίζουν την τάση-ρεύματος μιας διόδου zener.

```
x=[-1, 0, 1.27, 2.55, 3.82, 4.92, 5.02];
y=[-14.58, 0., 0., 0., 0., .88, 11.17];
xInt = -1 : 0.01 : 5.02; yInt =
interp1(x,y,xInt);plot(xInt,yInt,'g',x,y,'b*');axis([-1,5.02,-13,12]);
```



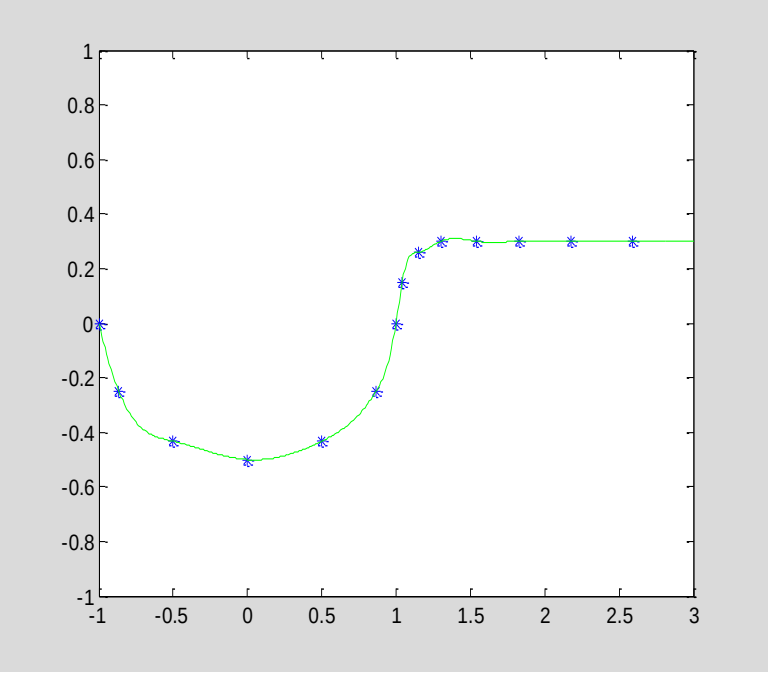

**Σχήμα 1.4**: Τμηματικό πολυώνυμο βαθμού 1 για τα δεδομένα που χαρακτηρίζουν την τάση-ρεύματος μιας διόδου zener.

```
x = [ -1,-.866,-
.5,0,0.5,0.866,1,1.0402,1.15,1.3,1.54,1.828,2.1736,2.5883,3.086];
y = [ 0,-0.25,-0.433,-0.5,-0.433,-
0.25,0,0.15,0.2598,0.3,0.3,0.3,0.3,0.3,0.3];
xInt = -1 : 0.01 : 3; yInt = interp1(x,y,xInt); % default linear
interpolation
plot(x,y,'b*',xInt,yInt,'g'); axis([-1,3,-1,1]);
```




**Σχήμα 1.5**: Τμηματικό πολυώνυμο βαθμού 1 για ένα μεγαλύτερο σύνολο δεδομένων που χαρακτηρίζουν την τάση-ρεύματος μιας διόδου zener.

```
yInt = interp1(x,y,xInt,'spline'); % piecewise cubic spline
interpolation
plot(x,y,'b*',xInt,yInt,'g'); axis([-1,3,-1,1]);
```




**Σχήμα 1.6**: Τμηματικό πολυώνυμο βαθμού 3 για τα δεδομένα που χαρακτηρίζουν την τάση-ρεύματος μιας διόδου zener.

## 2. Παρεμβολή με τμηματικά πολυώνυμα

**Πρόβλημα:** Θεωρείστε ένα σύνολο ***(n)*** σημείων δεδομένων:

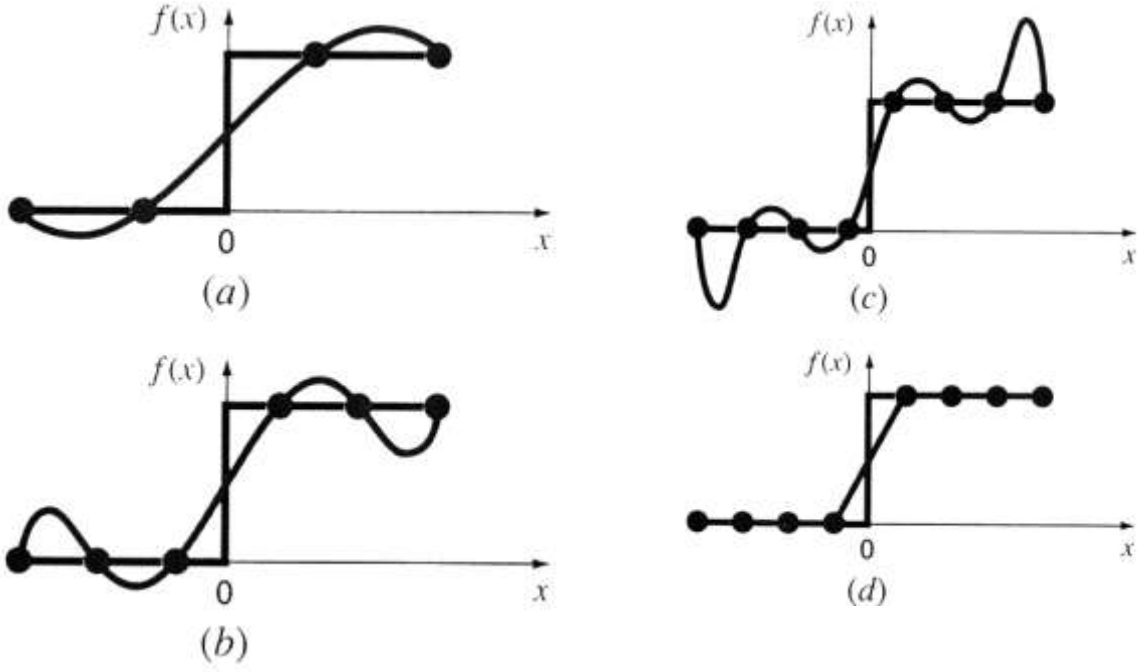
$$(x_1,y_1);\ (x_2,y_2);\ \ldots;\ (x_n,y_n)$$

Βρείτε μια "ομαλή" συνάρτηση (*Ομαλές λέγονται οι συναρτήσεις που έχουν συνεχείς παραγώγους)* τύπου spline ***y = S(x)*** η οποία συνδέει το παραπάνω σύνολο ***(n)*** δεδομένων σημείων. Η συνάρτηση ***y = S(x)*** αποτελείται από πολυώνυμα ***$y = S_j(x)$*** μικρού βαθμού τα οποία συνδέουν δυο συνεχόμενα σημεία δεδομένων: ***$(x_j,y_j)$*** and ***$(x_{j+1},y_{j+1})$.***

Ας θυμηθούμε ότι τα τμηματικά πολυώνυμα ή splines βαθμού $k$ είναι οι συναρτήσεις που ορίζονται ως προς μια διαμέριση $a = x_1 < x_2 < \cdots < x_n = b$ του πεδίου ορισμού των $[a,b]$ έτσι ώστε

(1) $s \in [x_1, x_n]$;
(2) $s^{(j)}$, $j = 0, 1, 2, \cdots, k-1$ είναι όλες συνεχείς στο $[x_1, x_n]$ όπου $s^{(j)}$ είναι η j βαθμού παράγωγος;
(3) $s$ είναι πολυώνυμο βαθμού $\le k$ σε κάθε διάστημα $[x_i, x_{i+1}]$.

Για να κατανοήσουμε την χρησιμότητα των splines για την προσέγγιση "δύσκολων" συναρτήσεων, στο παρακάτω γράφημα συγκρίνουμε γραφικά για άλλη μια φορά την πολυωνυμική παρεμβολή με την γραμμική τμηματική παρεμβολή για την προσέγγιση μιας βηματικής (step) συνάρτησης.



**Σχήμα 2.1**: Προσέγγιση της βηματικής συνάρτησης με (a) πολυώνυμο παρεμβολής 3ου βαθμού, (b) 5ου βαθμού, (c) 7ου βαθμού, και (d) γραμμική spline.

Παρακάτω ορίζουμε μια σειρά από splines που χρησιμοποιούνται στην πράξη.

## 3. Παρεμβολή με γραμμικά τμηματικά πολυώνυμα (1ου βαθμού)

Δύο συνεχόμενα σημεία δεδομένων $(x_j,y_j)$ και $(x_{j+1},y_{j+1})$ μπορούν να ενωθούν με μια ευθεία γραμμή:

$$y = S_j(x) = y_j + \frac{(y_{j+1} - y_j)}{(x_{j+1} - x_j)} (x - x_j) = a_j + b_j (x - x_j)$$

Οι γραμμικές συναρτήσεις παρεμβολής $y = S_j(x)$ for $1 \le j \le n-1$ συνδέουν όλα τα δεδομένα σημεία $(n)$ αλλά δεν έχουν συνεχείς πρώτες παραγώγους στα δεδομένα σημεία. Έτσι, για ένα σύνολο δεδομένων σημείων που τους ονομάζουμε «κόμβους»: $(x_i, y_i,\ i = 1, 2, \cdots, n)$, η γραμμική spline είναι $s_i(x) = a_i x + b_i,\ for\ x \in [x_i, x_{i+1}],\quad i = 1, 2, \cdots, n-1$

Η σύνδεση των γραμμικών πολυωνύμων στους εσωτερικούς κόμβους συνεπάγεται ότι το γραμμικό τμηματικό πολυώνυμο είναι συνεχές ($C^0$) , δηλαδή ισχύουν οι $2(n-2)$ σχέσεις $s_i(x_i) = y_i$ και $s_i(x_{i+1}) = y_{i+1}$ στους εσωτερικούς κόμβους. Οι συνθήκες αυτές συμπεριλαμβανομένων και των συνθηκών παρεμβολής στα άκρα δίνουν ένα σύνολο $2n-2$ εξισώσεων για τον προσδιορισμό των συντελεστών $a_i, b_i, i = 1, 2, \cdots, n-1$. Η λύση των εξισώσεων αυτών προσδιορίζει κάθε τμηματικό γραμμικό πολυώνυμο που στην μορφή Lagrange ορίζεται ως $s_i(x) = y_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i} = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i),\quad x \in [x_i, x_{i+1}]$

**Παράδειγμα 3.1**: Να βρεθούν οι γραμμικές spline για τα ακόλουθα δύο σύνολα τιμών (I, II)

| i | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| x | 0 | 5 | 7 | 8 | 10 |
| y | 0 | 2 | -1 | -2 | 20 |

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| x | 0 | 1 | 2 | 3 | 4 | 5 |
| y | 1 | 1 | 1 | -1 | -1 | -1 |

Γραμμική spline για το σύνολο I:

$$s_1(x) = 0\frac{x-5}{0-5} + 2\frac{x-0}{5-0} = 2.5x, \quad x \in [0, 5]$$

$$s_2(x) = 2\frac{x-7}{5-7} - 1\frac{x-5}{7-5} = -1.5x + 9.5, \quad x \in [5, 7]$$

$$s_3(x) = -1\frac{x-8}{7-8} - 2\frac{x-7}{8-7} = -x + 6, \quad x \in [7, 8]$$

$$s_4(x) = -2\frac{x-10}{8-10} + 20\frac{x-8}{10-8} = 11x - 90, \quad x \in [8, 10]$$

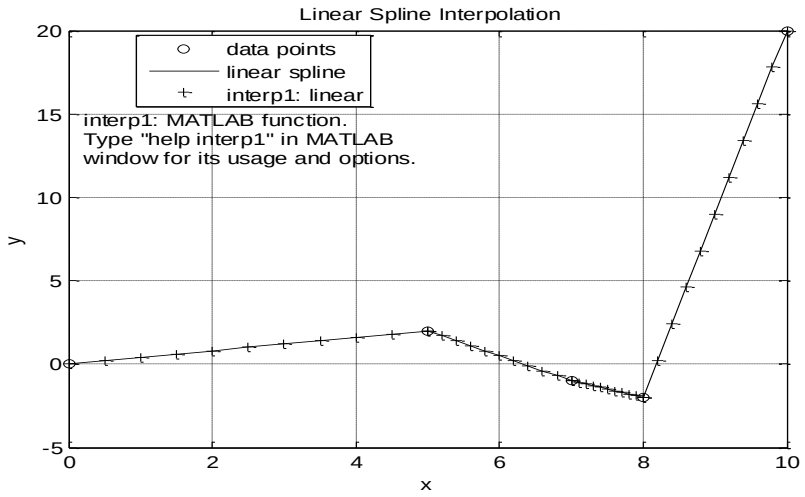
Ο κώδικας MATLAB "*linear_spline_examples.m*" μπορεί να χρησιμοποιηθεί για να παράγουμε το γράφημα 3.1. που υλοποιεί τα παρακάτω βήματα:

```
(1)  Input τα δεδομένα σημεία παρεμβολής x(i) ,y(i), i = 1,n; % 
τα παριστούμε γραφικά χρησιμοποιώντας την συνάρτηση plot;
(2)  For i = 1, n-1, do
        A = linspace(x(i),x(i+1),nint)   % Διαιρούμε κάθε 
διάστημα σε  nint σημεία
        Call c1=linear_spline  % καλούμε την συνάρτηση 
linear_spline.m
        Call c2=interp1  % Καλούμε την MATLAB συνάρτηση (default 
option: linear)
        Plot c1
        Plot c2
     Enddo
```

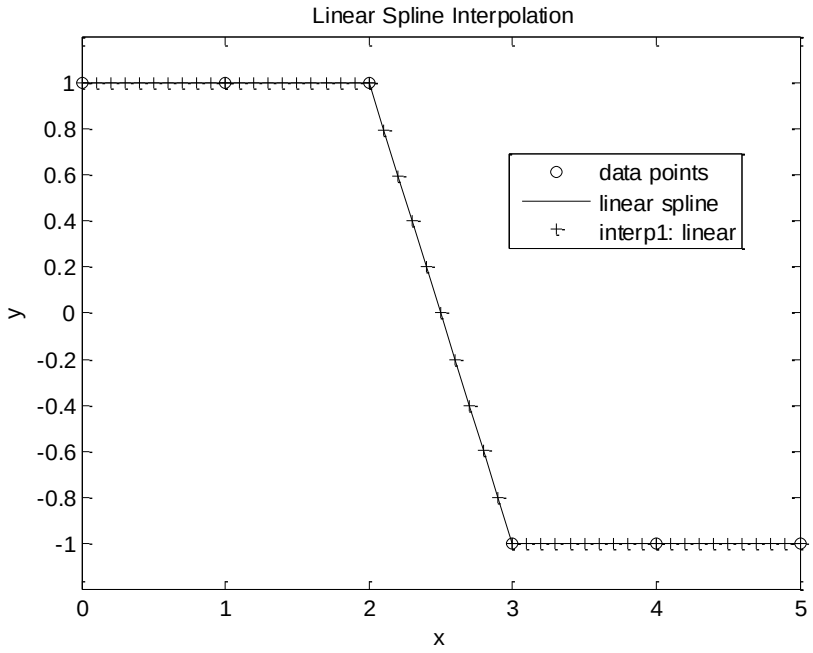
Η συνάρτηση "*linear_spline.m*" εφαρμόζει το ακόλουθο αλγόριθμο:

```
(1)  Εισάγουμε το x για να βρούμε την τιμή της παρεμβολής y στο
   εν λόγω σημείο;
(2)  Βρίσκει το υποδιάστημα που περιέχει το x; Ελέγχει αν το x
   είναι εκτός του ολικού διαστήματος που περιλαμβάνει τα σημεία
   παρεμβολής x; Αν ισχύει τότε  σταματάει το πρόγραμμα
   διαφορετικά συνεχίζει.
(3)   Υπολογίζει το y από την εξίσωση  του γραμμικού πολυωνύμου
   y=si(x).
```

Παρακάτω δίδονται οι γραφικές παραστάσεις των splines βαθμού 1 για τα δυο σύνολα δεδομένων I και II.




**Σχήμα 3.1:** Γραμμική spline για το σύνολο δεδομένων Ι.

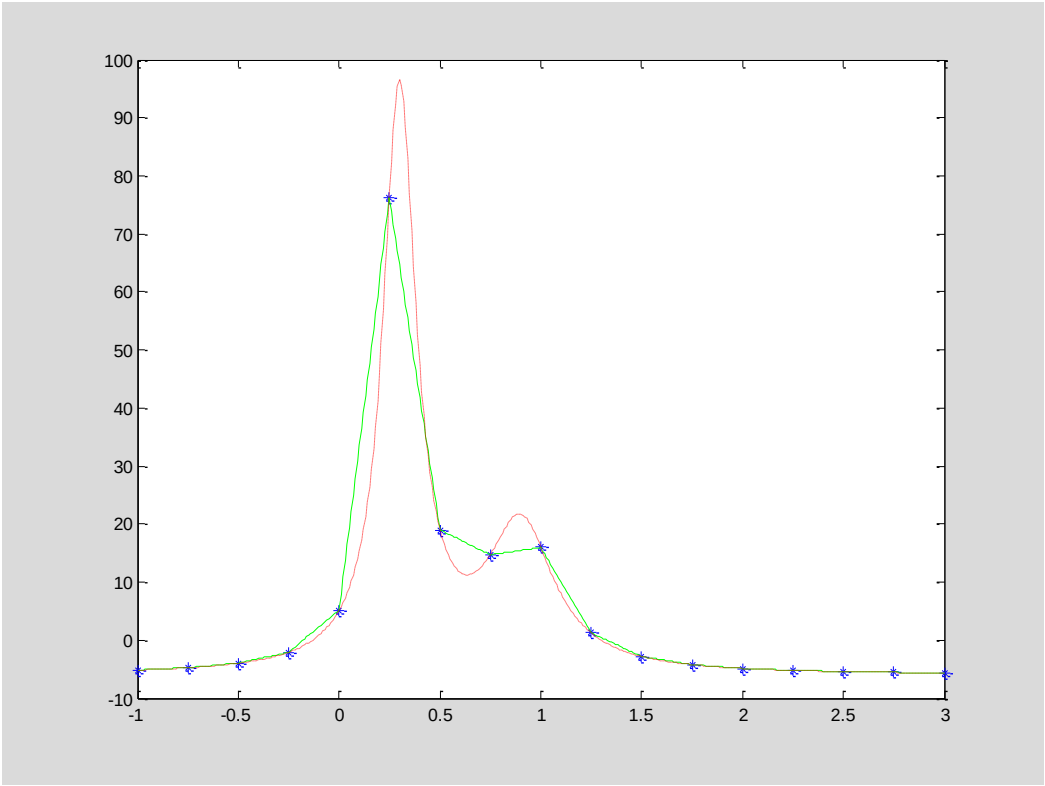



**Σχήμα 3.2:** Γραμμική spline για το σύνολο δεδομένων ΙΙ.

**Παρατηρήσεις**: 1) Η τμηματική γραμμική συνάρτηση παρεμβολής χρησιμοποιείται στο σχεδιασμό της γραφικής παράστασης συναρτήσεων ***y = f(x)*** για δεδομένο σύνολο σημείων. 2) Η τμηματική γραμμική συνάρτηση παρεμβολής δεν δίνει πάντα ικανοποιητικές προσεγγίσεις όπως το παρακάτω παράδειγμα δείχνει.

```
% computation of coefficients of linear interpolants from a given data
set [x,y]
x = -1 : 0.25 : 3; y = humps(x);
n = length(y)-1; a = y(1:n); b = (y(2:n+1)-y(1:n))./(x(2:n+1)-x(1:n));

% evaluation of linear interpolants at data points xInt
```

```
xInt = -1 : 0.01 : 3;
for j = 1 : length(xInt)
      if xInt(j) ~= x(n+1)
   iInt(j) = sum(x <= xInt(j));
      else
            iInt(j) = n;
      end
end
yInt = a(iInt) + b(iInt).*(xInt-x(iInt));
yEx = humps(xInt);
plot(x,y,'b*',xInt,yInt,'g',xInt,yEx,'r:'); axis([-1,3,-10,100]);
```




**Σχήμα 3.3:** Γραμμική spline παρεμβολής για την MATLAB συνάρτηση humps.

## 4. Τμηματική τετραγωνική (πολυώνυμο βαθμού 2) παρεμβολή

### 4.1 Ορισμός της τετραγωνική spline παρεμβολής μέσω τριγωνικού συστήματος εξισώσεων παρεμβολής

Για το σύνολο δεδομένων $\{(x_1,y_1),(x_2,y_2),(x_3,y_3),\cdots,(x_n,y_n)\}$ η spline παρεμβολής βαθμού 2 στο διάστημα $[x_1,x_n]$ ορίζεται από τα τμηματικά πολυώνυμα δευτέρου βαθμού:

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2, \text{ στο διάστημα } I_\iota = [x_\iota, x_{\iota-1}] \text{ για i=1,...,n-1}$$

Ο προσδιορισμός των 3(n-1) συντελεστών $a_i$, $b_i$, $c_i$ επιτυγχάνεται από την απαίτηση να ικανοποιούνται οι παρακάτω συνθήκες:

1. ***Το τμηματικό πολυώνυμο πρέπει να διέρχεται από τα σημεία ($x_i$ , $y_i$ ) (συνθήκες συνέχειας). Αυτό σημαίνει ότι οι τιμές των παρακειμένων πολυωνύμων πρέπει να είναι ίσες σε όλους τους κόμβους. Αυτές είναι 2(n-2) + 2 = 2(n-1) συνθήκες (εξισώσεις)***

$$\begin{cases} s_i(x_i) = f_i = a_i \\ s_i(x_{i+1}) = f_{i+1} = a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 = f_i + b_i h_i + c_i h_i^2 \end{cases}$$

όπου $h_i = x_{i+1} - x_i$

2. ***Η 1η παράγωγος είναι συνεχής στους εσωτερικούς κόμβους $x_i$,***

Αυτό απαιτεί (n-2) συνθήκες

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1})$$
$$b_i + 2c_i(x_{i+1} - x_i) = b_{i+1} + 2c_{i+1}(x_{i+1} - x_{i+1})$$
$$b_i + 2c_i h_i = b_{i+1}$$

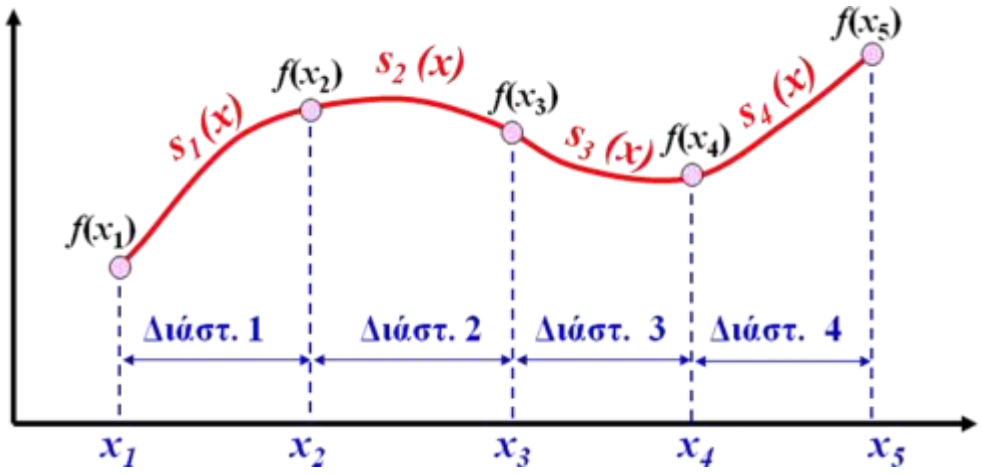
3. ***Υποθέστε ότι η δεύτερη παράγωγος είναι μηδέν στο πρώτο κόμβο***

$$s_1''(x_1) = 2c_1 = 0 \quad \Rightarrow \quad c_1 = 0$$

Παρατηρούμε ότι το σύνολο των παραπάνω συνθηκών είναι ***2(n–1) + (n–2) + (1) = 3(n–1)*** που είναι ίσος με τους συντελεστές του τμηματικού πολυωνύμου που πρέπει να προσδιοριστεί.

Για n=5 η τμηματική τετραγωνική spline είναι

$$s(x) = \begin{cases} [x_1, x_2]: & s_1(x) = a_1 + b_1(x - x_1) + c_1(x - x_1)^2 \\ [x_2, x_3]: & s_2(x) = a_2 + b_2(x - x_2) + c_2(x - x_2)^2 \\ [x_3, x_4]: & s_3(x) = a_3 + b_3(x - x_3) + c_3(x - x_3)^2 \\ [x_4, x_5]: & s_4(x) = a_4 + b_4(x - x_4) + c_4(x - x_4)^2 \end{cases}$$




Συνθήκες συνεχείας της spline είναι:

$$\begin{cases} a_1 = f_1 \\ a_2 = f_2 \\ a_3 = f_3 \\ a_4 = f_4 \end{cases} \quad \& \quad \begin{cases} f_2 = f_1 + b_1 h_1 + c_1 h_1^2 \\ f_3 = f_2 + b_2 h_2 + c_2 h_2^2 \\ f_4 = f_3 + b_3 h_3 + c_3 h_3^2 \\ f_5 = f_4 + b_4 h_4 + c_4 h_4^2 \end{cases}$$

Συνθήκες συνέχειας παραγώγου της spline είναι:

$$\begin{cases} b_2 + 2c_2 h_2 = b_3 \\ b_3 + 2c_3 h_3 = b_4 \\ b_4 + 2c_4 h_4 = b_5 \end{cases}$$

Συνθήκη δευτέρας παραγώγου της spline στα άκρα του διαστήματος ορισμού είναι:

$$s_1''(x_1) = 2c_1 = 0 \quad \Rightarrow \quad c_1 = 0$$

Η παράσταση των παραπάνω συνθηκών σε μορφή πίνακα είναι:

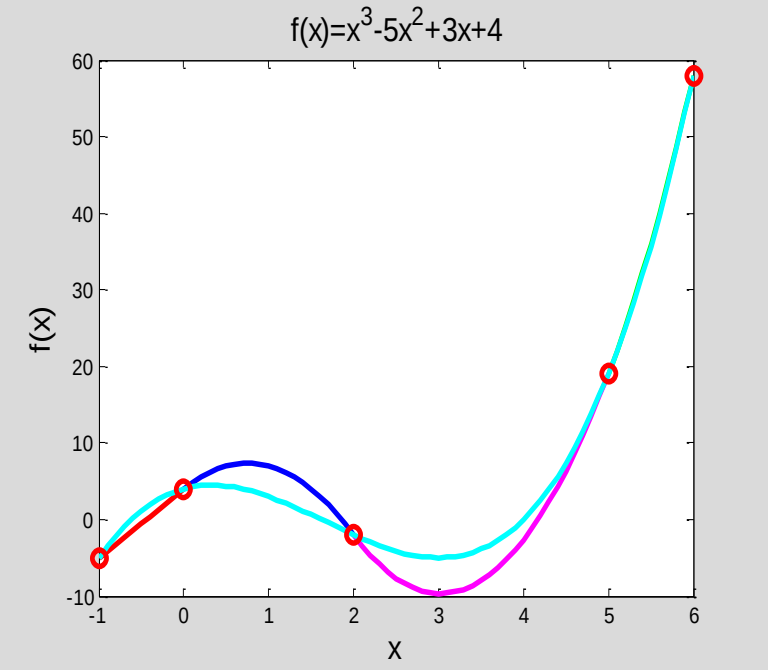
$$\begin{bmatrix}
1 & 0 & 0 & & & & & & & & & \\
0 & h_1 & h_1^2 & & & & & & & & & \\
 & & & 1 & 0 & 0 & & & & & & \\
 & & & 0 & h_2 & h_2^2 & & & & & & \\
 & & & & & & 1 & 0 & 0 & & & \\
 & & & & & & 0 & h_3 & h_3^2 & & & \\
 & & & & & & & & & 1 & 0 & 0 \\
 & & & & & & & & & 0 & h_4 & h_4^2 \\
0 & 1 & 2h_1 & 0 & -1 & 0 & & & & & & \\
 & & & 0 & 1 & 2h_2 & 0 & -1 & 0 & & & \\
 & & & & & & 0 & 1 & 2h_3 & 0 & -1 & 0 \\
0 & 0 & 1 & & & & & & & & &
\end{bmatrix}
\begin{Bmatrix} a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ a_3 \\ b_3 \\ c_3 \\ a_4 \\ b_4 \\ c_4 \end{Bmatrix}
=
\begin{Bmatrix} f_1 \\ f_2 - f_1 \\ f_2 \\ f_3 - f_2 \\ f_3 \\ f_4 - f_3 \\ f_4 \\ f_5 - f_4 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

Η παράσταση των παραπάνω συνθηκών σε MatLab και η λύση τους είναι:

```
%function [A, b] = quadratic(x, f)
%exact solution f(x)=x^3-5x^2+3x+4
x=[-1 0 2 5 6]; f=[-5 4 -2 19 58];
h1=x(2)-x(1); h2=x(3)-x(2); h3=x(4)-x(3); h4=x(5)-x(4);
f1=f(1); f2=f(2); f3=f(3); f4=f(4); f5=f(5);
A=[1    0    0    0    0    0    0    0    0    0    0    0
   0   h1  h1^2   0    0    0    0    0    0    0    0    0
   0    0    0    1    0    0    0    0    0    0    0    0
   0    0    0    0   h2  h2^2   0    0    0    0    0    0
   0    0    0    0    0    0    1    0    0    0    0    0
   0    0    0    0    0    0    0   h3  h3^2   0    0    0
   0    0    0    0    0    0    0    0    0    1    0    0
   0    0    0    0    0    0    0    0    0    0   h4  h4^2
   0    1  2*h1   0   -1    0    0    0    0    0    0    0
   0    0    0    0    1  2*h2   0   -1    0    0    0    0
   0    0    0    0    0    0    0    1  2*h3   0   -1    0
   0    0    1    0    0    0    0    0    0    0    0    0];
b=[f1; f2-f1; f2; f3-f2; f3; f4-f3; f4; f5-f4; 0; 0; 0; 0];
p = A\b
dx = 0.1;
z1=x(1):dx:x(2); s1=p(1)+p(2)*(z1-x(1))+p(3)*(z1-x(1)).^2;
z2=x(2):dx:x(3); s2=p(4)+p(5)*(z2-x(2))+p(6)*(z2-x(2)).^2;
z3=x(3):dx:x(4); s3=p(7)+p(8)*(z3-x(3))+p(9)*(z3-x(3)).^2;
z4=x(4):dx:x(5); s4=p(10)+p(11)*(z4-x(4))+p(12)*(z4-x(4)).^2;
H=plot(z1,s1,'r',z2,s2,'b',z3,s3,'m',z4,s4,'g');
xx=-1:0.1:6; fexact=xx.^3-5*xx.^2+3*xx+4;
hold on; G=plot(xx,fexact,'c',x,f,'ro');
set(H,'LineWidth',3); set(G,'LineWidth',3,'MarkerSize',8);
H1=xlabel('x'); set(H1,'FontSize',15);
H2=ylabel('f(x)'); set(H2,'FontSize',15);
H3=title('f(x)=x^3-5x^2+3x+4'); set(H3,'FontSize',15);
```

```
p =
   -5.0000
    9.0000
         0
```

```
    4.0000
    9.0000
   -6.0000
   -2.0000
  -15.0000
    7.3333
   19.0000
   29.0000
   10.0000
```




**Σχήμα 4.1**: Τετραγωνική spline παρεμβολής για την συνάρτηση f(x)=x^3-5x^2+3x+4. Το γράφημα των τμηματικών πολυωνύμων της spline παρίστανται με διαφορετικό χρώμα. Το γράφημα της δοθείσης συνάρτησης απεικονίζεται με μπλε χρώμα.

## 4.2 Ορισμός τετραγωνικής spline παρεμβολής με αναδρομικό τύπο

Θεωρούμε την παρακάτω διαμέριση του πεδίου ορισμού της τετραγωνικής spline

$$\text{Σημεία παρεμβολής}: \ (x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots, (x_n, y_n)$$
$$\text{Υποδιαστήματα}: \quad I_1 = [x_1, x_2] \ , \ I_2 = [x_2, x_3] \ , \cdots, I_{n-1} = [x_{n-1}, x_n]$$

Τότε, η τετραγωνική spline από τον ορισμό της ορίζεται σε κάθε υποδιάστημα ως

$$s_i(x) = a_i x^2 + b_i x + c_i, \quad \text{για } x \in [x_i, x_{i+1}], \quad i = 1, 2, \cdots, n-1$$

και οφείλει να παρεμβάλει τα δοθέντα σημεία και η παράγωγος της να είναι συνεχής (δηλαδή, η τετραγωνική spline ανήκει στους χώρους $C^0$ και $C^1$). Από τις συνθήκες παρεμβολής (συνέχειας), έχουμε τις εξισώσεις

$$s_i(x_i) = y_i, \quad s_i(x_{i+1}) = y_{i+1}, \quad i = 1,2,...n-1.$$

Οι συνθήκες συνέχειας της πρώτης παραγώγου ( $C^1$) μας δίνουν τις εξισώσεις

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}), \quad i = 1, 2, \cdots, n-2$$

Το παραπάνω σύνολο των τριών (3) συνθηκών δίνουν *3n-4* εξισώσεις που δεν είναι αρκετές να προσδιορίσουν μονοσήμαντα τους *3n-3* συντελεστές της τετραγωνικής spline. Επομένως, χρειαζόμαστε μια επιπλέον συνθήκη για να την ορίσουμε πλήρως. Συνήθως, απαιτούμε $s'_1(x_1) = 0$. Στην περίπτωση αυτή η spline αναφέρεται σαν η φυσική (natural) τετραγωνική spline. Άλλες συνθήκες όπως $s'_1(x_1) = s'_{n-1}(x_n)$ μπορούν να εφαρμοστούν.

Για να προσδιορίσουμε τις τετραγωνικές splines, πρώτα συμβολίζουμε με $d_i = s'_i(x_i)$. Το γεγονός, ότι η $s'(x)$ είναι γραμμική spline για το σύνολο δεδομένων $(x_i, d_i, i = 1,2, \ldots, n)$, μας οδηγεί στην παρατήρηση ότι

$$s_i'(x) = d_{i+1}\frac{x - x_i}{x_{i+1} - x_i} + d_i\frac{x_{i+1} - x}{x_{i+1} - x_i} = \frac{(d_{i+1} - d_i)x - d_{i+1}x_i + d_i x_{i+1}}{x_{i+1} - x_i}$$

$$= \frac{d_{i+1} - d_i}{x_{i+1} - x_i}(x - x_i) + d_i$$

Ολοκληρώνοντας την συνάρτηση αυτή ως προς x και λαμβάνοντας υπόψη τις συνθήκες συνέχειας (παρεμβολής) $(s_i(x_i)=y_i)$ βρίσκουμε ότι

$$s_i(x) = \frac{d_{i+1} - d_i}{2(x_{i+1} - x_i)}(x - x_i)^2 + d_i(x - x_i) + y_i$$

Θέτοντας $x = x_{i+1}$, λαμβάνουμε

$$y_{i+1} = \frac{d_{i+1} - d_i}{2(x_{i+1} - x_i)}(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i) + y_i$$

που μας δίδει την αναδρομική εξίσωση για τον προσδιορισμό των $d_i$'ς

$$d_{i+1} = 2\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - d_i, \quad i = 1, 2, \cdots, n-1$$

Αν γνωρίζουμε το $d_1$, τότε το $d_i$ μπορεί να προσδιορισθεί από την παραπάνω εξίσωση. Διάφορες τιμές μπορούν να δοθούν στο $d_1$.
Για την φυσική ( natural) spline, $d_1 = 0$.
Αν $d_1 = d_2$, τότε $d_1$ μπορεί να υπολογιστεί από τον τύπο

$$d_1 = d_2 = 2\frac{y_2 - y_1}{x_2 - x_1} - d_1 \Rightarrow d_1 = \frac{y_2 - y_1}{x_2 - x_1}$$

Αν $d_1 = d_n$, τότε $d_1$ μπορεί να υπολογιστεί από τον τύπο

$$d_1 = d_n = 2\sum_{i=2}^{n-1}\left((-1)^{n-i+1}\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) - (-1)^n d_1 \Rightarrow \begin{cases} d_1 = \sum_{i=2}^{n-1}\left((-1)^{n-i+1}\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right), & n\ \text{άρτιος} \\ \text{όχι λύση} & n\ \text{περιττός} \end{cases}$$

Αυτή η συνθήκη δεν μπορεί να εφαρμοστεί αν ο n είναι περιττός αριθμός (Γιατί;).

Η παραπάνω διαδικασία εύρεσης της τετραγωνικής spline περιγράφεται στο MATLAB κώδικα "*quadratic_spline_examples.m*" που στηρίζεται στα εξής βήματα:

```
(1)  Input σημεία παρεμβολής x(i) ,y(i), i = 1,n; τα
παριστούμε γραφικά με την συνάρτηση plot;
(2)  For i = 1, n-1, do
   A = linspace(x(i),x(i+1),nint)  % Divide each data
interval into nint points
   Call c1=quadratic_spline with d1 = 0;  % Natural
quadratic spline
   Call c2=quadratic_spline with d1 = d2;
   Plot c1
   Plot c2
Enddo
```

Η συνάρτηση "*quadratic_spline.m*" στηρίζεται στον ακόλουθο αλγόριθμο:

```
(1)  Input x για να βρούμε την τιμή της παρεμβολής y;
Ελέγχουμε αν το x εκτός του ολικού διαστήματος που
περιλαμβάνει τα σημεία  παρεμβολής x; If yes σταματάμε το
πρόγραμμα διαφορετικά συνεχίζουμε;
(2)  βρίσκουμε το υποδιάστημα που περιέχει το x;
(3)       Υπολογίζουμε d1; Υπολογίζουμε di  από την
αναδρομική εξίσωση
(4)  Υπολογίζουμε το y από την εξίσωση
```

$$y = s_i(x) = \frac{d_{i+1} - d_i}{2(x_{i+1} - x_i)}(x - x_i)^2 + d_i(x - x_i) + y_i$$

**Παράδειγμα 4.1.** Να προσδιοριστεί η τετραγωνική spline για τα δεδομένα του παραδείγματος 3.1 σύνολο Ι.

Επιλέγουμε $d_1 = 0$, τότε έχουμε

$$d_2 = 2\frac{2-0}{5-0} - 0 = 0.8, \quad d_3 = 2\frac{-1-2}{7-5} - 0.8 = -3.8$$

$$d_4 = 2\frac{-2+1}{8-7} + 3.8 = 1.8, \quad d_5 = 2\frac{20+2}{10-8} - 1.8 = 20.2$$

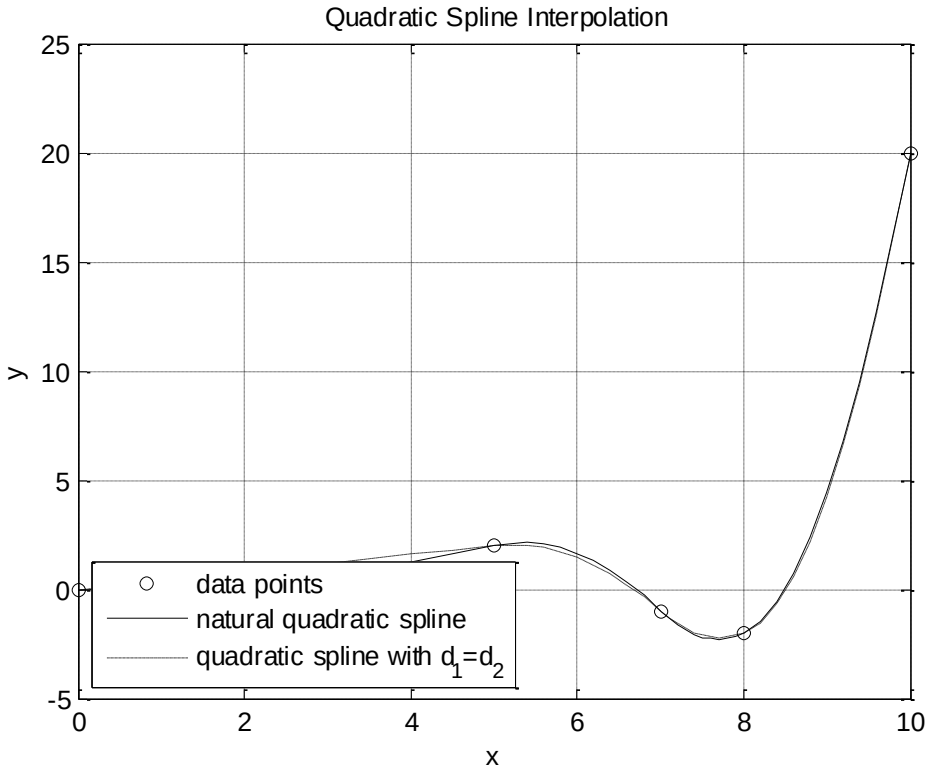
$$s_1(x) = \frac{0.8-0}{2(5-0)}(x-0)^2 + 0(x-0) + 0 = 0.08x^2, \quad x \in [0, 5]$$

$$s_2(x) = \frac{-3.8-0.8}{2(7-5)}(x-5)^2 + 0.8(x-5) + 2 = -1.15x^2 + 12.3x - 289.5, \quad x \in [5, 7]$$

$$s_3(x) = \frac{1.8+3.8}{2(8-7)}(x-7)^2 - 3.8(x-7) - 1 = 2.8x^2 - 9.4x + 162.8, \quad x \in [7, 8]$$

$$s_4(x) = \frac{20.2-1.8}{2(10-8)}(x-8)^2 + 1.8(x-8) - 2 = 4.6x^2 - 71.8x + 273.6, \quad x \in [8, 10]$$

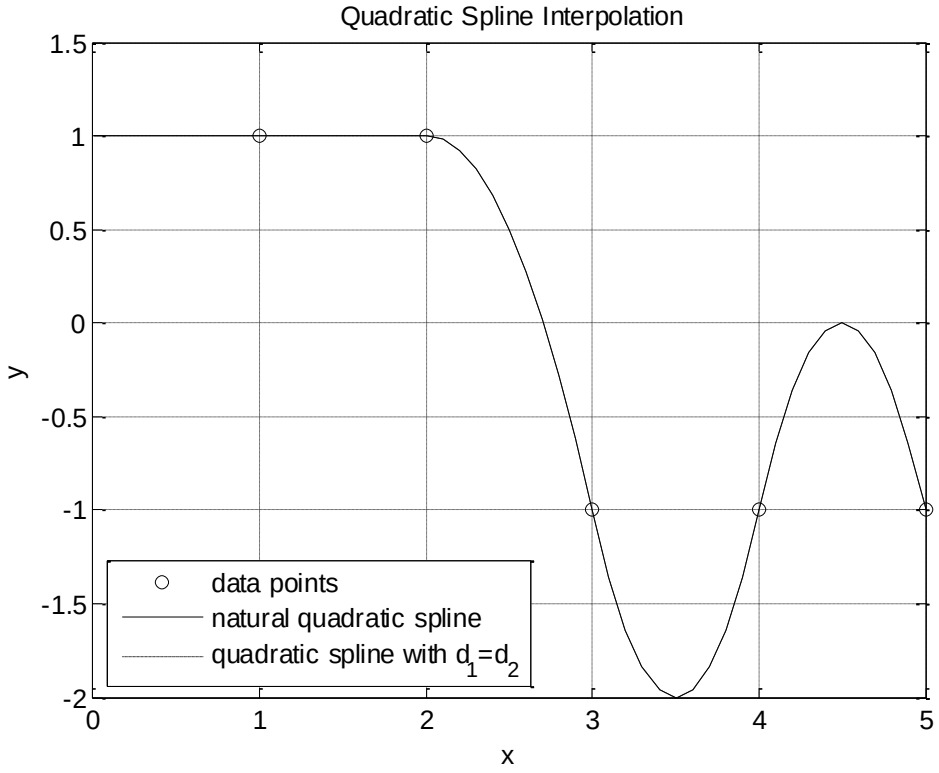
**Παρατήρηση**: Τα αποτελέσματα για τις splines με επιλογή $d_1 = d_2$, δίδονται στο γράφημα 4.2. Στο σημείο $x = 0$, η φυσική spline είναι επίπεδη λόγο της μηδενικής κλίσης. Η άλλη spline έχει γραμμική κλίση στο πρώτο διάστημα. Σημειώστε ότι οι δύο splines διαφέρουν ελάχιστα.




**Σχήμα 4.2.** Τετραγωνική Splines για το σύνολο δεδομένων I.

**Παράδειγμα 4.2**. Η τετραγωνική spline για τα δεδομένα του παραδείγματος 3.1 σύνολο II δίδεται στο σχήμα 4.3. Παρατηρείστε ότι οι τετραγωνικές splines με $d_1 = 0$ και $d_1 = d_2$ είναι ίδιες διότι $d_2=0$ . Επίσης, οι splines στο 4ο και 5ο διαστήματα έχουν μεγάλες

ταλαντώσεις.




**Σχήμα 4.3.** Τετραγωνική Splines για το σύνολο δεδομένων ΙΙ.

## 5. <u>Παρεμβολή με κυβικές splines:</u>

Θεωρούμε τα ***(n+1)*** σημεία δεδομένων $(x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots, (x_{n+1}, y_{n+1})$ και εξετάζουμε την περίπτωση δύο συνεχόμενα σημεία δεδομένων $(x_j, y_j)$ και $(x_{j+1}, y_{j+1})$ να ενωθούν μέσω ενός κυβικού πολυωνύμου:

$$y = S_j(x) = a_j + b_j (x - x_j) + c_j (x - x_j)^2 + d_j (x - x_j)^3$$

Οι κυβικές spline συναρτήσεις παρεμβολής $y = S_j(x)$ for $1 \le j \le n$ συνδέουν όλα τα ***(n+1)*** σημεία δεδομένων $(x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots, (x_{n+1}, y_{n+1})$ και έχουν συνεχείς πρώτες και δεύτερες παραγώγους. Οι κυβικές splines παρεμβολής παρέχουν την πιο ομαλή λύση στο πρόβλημα της παρεμβολής και είναι κατάλληλες για το σχεδιασμό «ομαλών» συναρτήσεων $y = f(x)$ από ένα σύνολο δεδομένων σημείων.

**Πρόβλημα:** Βρείτε τους συντελεστές $[a_j,b_j,c_j,d_j]$ των κυβικών spline συναρτήσεων παρεμβολής έτσι ώστε να ικανοποιούν τις συνθήκες:

*Συνέχεια των συναρτήσεων* $\boxed{S_j(x_j) = y_j \ , \ S_j(x_{j+1}) = y_{j+1}}$

*Συνέχεια των κλίσεων-παραγώγων* $\boxed{S'_j(x_j) = S'_{j-1}(x_j)}$

*Συνέχεια των δευτέρων παραγώγων:* $\boxed{S''_j(x_j) = S''_{j-1}(x_j)}$

Σημειώστε ότι τα πολυωνυμικά τμήματα $S_i(x)$ των κυβικών splines έχουν τις εξής ιδιότητες:

$S_i(x)$ – κυβικά πολυώνυμα
$S'(x)$ – τετραγωνικά πολυώνυμα (κλίση)
$S_i''(x)$ – γραμμικά πολυώνυμα (καμπυλότητα)

Υπάρχουν $(2n)$ συνθήκες για τη συνέχεια των συναρτήσεων, $(n-1)$ συνθήκες για τη συνέχεια των κλίσεων και $(n-1)$ συνθήκες για τη συνέχεια των δευτέρων παραγώγων, δηλαδή το σύνολο των συνθηκών είναι $(4n-2)$. Ωστόσο, υπάρχουν $(4n)$ άγνωστοι συντελεστές $[a_j,b_j,c_j,d_j]$. Επομένως για την εύρεση μιας κυβικής spline θα πρέπει να προστεθούν δύο ακόμη συνθήκες .

Αυτές οι συνθήκες ορίζονται στα άκρα του διαστήματος $[x_1, x_{n+1}]$. Η πρώτη επιλογή απαιτεί την δεύτερη παράγωγο της spline στα άκρα να είναι μηδέν και στην περίπτωση αυτή η spline λέγεται **φυσική (natural)** spline. Οι συνθήκες αυτές είναι

$$\boxed{S''_1(x_1) = 0; \quad S''_n(x_{n+1}) = 0}$$

**Λύση:** Αν συμβολίζουμε με $m_j = S''_j(x_j)$ και $h_j = x_{j+1} - x_j$, τότε η συνέχεια των δεύτερων παραγώγων μας επιτρέπει να προσδιορίσουμε δύο από τους συντελεστές των τμηματικών πολυωνύμων της κυβικής spline:

$$\boxed{c_j = \frac{m_j}{2}} \quad \text{και} \quad \boxed{d_j = \frac{m_{j+1} - m_j}{6h_j}}$$

Από τις συνθήκες παρεμβολής ή από την συνέχεια των splines, βρίσκουμε τους δύο άλλους συντελεστές:

$$\boxed{a_j = y_j} \quad \text{and} \quad \boxed{b_j = \frac{y_{j+1} - y_j}{h_j} - \frac{m_{j+1} + 2m_j}{6} h_j}$$

Έτσι, όλοι οι συντελεστές ορίζονται συναρτήσει των τιμών των $m_j$. Οι τιμές των $m_j$ βρίσκονται από τη συνέχεια των κλίσεων στα εσωτερικά σημεία $(n-1)$ και προσδιορίζονται από την λύση των παρακάτω $(n-1)$ εξισώσεων:

$$\boxed{h_{j-1}\, m_{j-1} + 2\,(h_{j-1} + h_j)\, m_j + h_j\, m_{j+1} = 6\left[\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}}\right]; \qquad j = 2,\ldots,n}$$

Οι συνοριακές τιμές για τις φυσικές spline συναρτήσεις είναι:

$$\boxed{\boldsymbol{m_1 = 0} \text{ and } \boldsymbol{m_{n+1} = 0}.}$$

Μια άλλη κατηγορία κυβικών splines, που αναφέρονται **μη-κομβικές (non-knots)** splines, χαρακτηρίζονται από την επιλογή των συνοριακών τιμών έτσι ώστε:

$$\boxed{m_1 = (1 + \frac{h_1}{h_2})m_2 - \frac{h_1}{h_2} m_3 \text{ and } m_{n+1} = (1 + \frac{h_{n-1}}{h_n})m_n - \frac{h_{n-1}}{h_n} m_{n-1}}$$

Το σύστημα που προσδιορίζει τις τιμές $\boldsymbol{m_j}$ για τις φυσικές splines μπορεί να παρασταθεί στην μορφή του τριγωνικού πίνακα

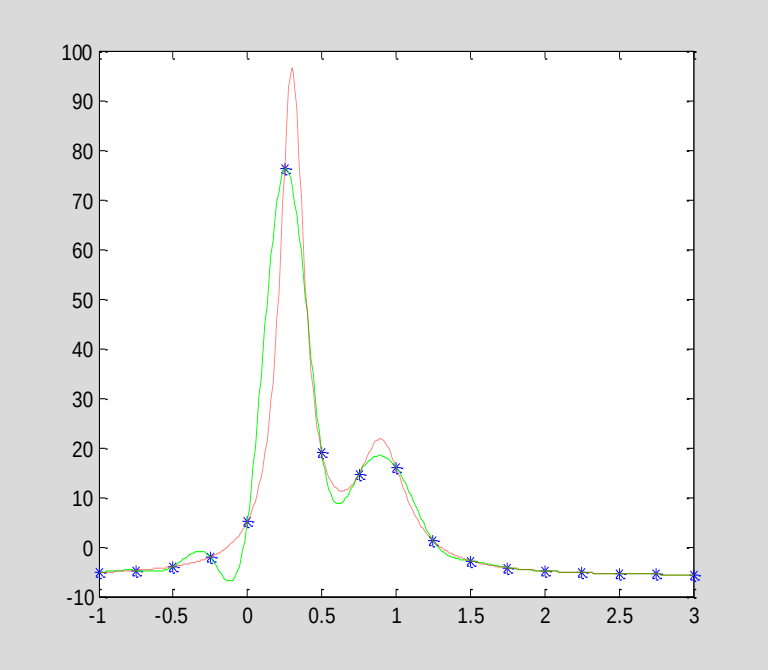
$$\begin{bmatrix} 1 & & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & h_2 & 2(h_2+h_3) & h_3 & & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & & & 1 \end{bmatrix} \begin{Bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_{n-1} \\ m_n \end{Bmatrix}$$

$$= 6 \begin{Bmatrix} 0 \\ (f[x_3,x_2] - f[x_2,x_1]) \\ (f[x_4,x_3] - f[x_3,x_2]) \\ \vdots \\ (f[x_n,x_{n-1}] - f[x_{n-1},x_{n-2}]) \\ 0 \end{Bmatrix} \text{ όπου } f[x_i,x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

Ο παρακάτω MATLAB κώδικας βρίσκει την φυσική κυβική spline για την humps συνάρτηση.

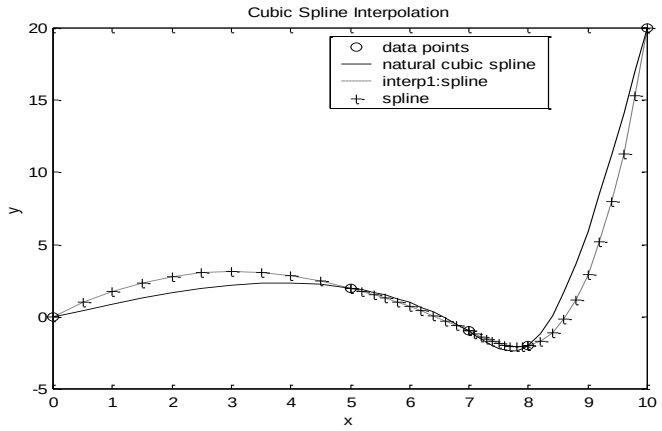
```
% computation of coefficients of cubic spline interpolants from a set
[x,y]
x = -1 : 0.25 : 3;      % data partition
y = humps(x);                                % function values
n = length(x)-1;
h = x(2:n+1)-x(1:n); % vector of step sizes of the partition of x
A = 2*diag(h(1:n-1)) + 2*diag(h(2:n)) + diag(h(2:n-1),1) + diag(h(2:n-
1),-1);
b = 6*((y(3:n+1)-y(2:n))./h(2:n)-(y(2:n)-y(1:n-1))./h(1:n-1));
m = A\b';
m = [0;m;0]'; % solving the linear system for natural splines
a = y(1:n);
b = (y(2:n+1)-y(1:n))./h(1:n)-h(1:n).*(m(2:n+1)+2*m(1:n))/6;
c = 0.5*m(1:n);
d = (m(2:n+1)-m(1:n))./(6*h(1:n));

% evaluation of cubic spline interpolants at data points xInt
```

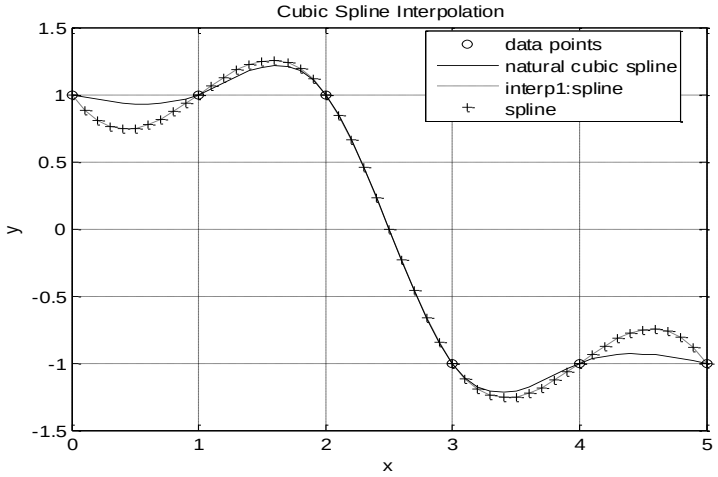
```
xInt = -1 : 0.01 : 3;       % another partition with better resolution
for j = 1 : length(xInt)
      if xInt(j) ~= x(n+1)
iInt(j) = sum(x <= xInt(j));
      else
            iInt(j) = n;
      end
end
yInt = a(iInt) + b(iInt).*(xInt-x(iInt)) + c(iInt).*(xInt - x(iInt)).^2
+ d(iInt).*(xInt - x(iInt)).^3;
yEx = humps(xInt);
plot(x,y,'b*',xInt,yInt,'g',xInt,yEx,'r:'); axis([-1,3,-10,100]);
```




Ο κώδικας "*cubic_spline_examples.m*" είναι παρόμοιος με τους κώδικες "*linear_spline_examples.m*" και χρησιμοποιεί τον κώδικα για τον προσδιορισμό της φυσικής splines "*cubic_spline.m*" και τις δύο MATLAB συναρτήσεις **interp1** και **spline**.

Οι παραπάνω κυβικές splines για τα δεδομένα του παραδείγματος 3.1 ( σύνολα I και II) απεικονίζονται στα παρακάτω γραφήματα.




## 6. Κυβική παρεμβολή Hermite:

Δύο συνεχόμενα σημεία δεδομένων $(x_j,y_j)$ και $(x_{j+1},y_{j+1})$ μπορούν να συνδεθούν με τη χρήση ενός κυβικού πολυωνύμου Hermite:

$$y = S_j(x) = a_j + b_j (x - x_j) + c_j (x - x_j)^2 + d_j (x - x_j)^2 (x - x_{j+1})$$

Οι κυβικές συναρτήσεις παρεμβολής Hermite $y = S_j(x)$ for $1 \le j \le n$ ενώνουν τα σημεία δεδομένων $(x_j,y_j)$ με τις προκαθορισμένες κλίσεις $s_j = y'_j$ σε κάθε σημείο δεδομένων. Επομένως, οι κυβικές συναρτήσεις παρεμβολής Hermite έχουν συνεχής πρώτες παραγώγους στα σημεία δεδομένων. Ωστόσο, δεν έχουν συνεχείς δεύτερες παραγώγους στα σημεία δεδομένων. Οι κυβικές συναρτήσεις παρεμβολής Hermite δεν χρησιμοποιούνται σε γραφικές εφαρμογές, γιατί το ανθρώπινο μάτι μπορεί να εντοπίσει ασυνέχειες στις καμπύλες της δεύτερης παραγώγου (καμπυλότητα - curvature).

**Πρόβλημα:** Βρείτε τους συντελεστές $[a_j,b_j,c_j,d_j]$ των κυβικών συναρτήσεων παρεμβολής Hermite από τις παρακάτω συνθήκες:

Συνέχεια συναρτήσεων/Συνθήκες παρεμβολής: $\boxed{S_j(x_j) = y_j, S_j(x_{j+1}) = y_{j+1}}$

Συνέχεια κλίσεων: $\boxed{S'_j(x_j) = s_j;\ S'_{j-1}(x_j) = s_{j+1}}$

Υπάρχουν ***(2n)*** συνθήκες για τη συνέχεια των συναρτήσεων και ***(2n)*** συνθήκες για τη συνέχεια των κλίσεων, δηλαδή ο συνολικός αριθμός συνθηκών είναι ***(4n)***. Υπάρχουν***(4n)*** άγνωστοι συντελεστές ***[$a_j$,$b_j$,$c_j$,$d_j$].*** Επομένως το πρόβλημα έχει μία μοναδική λύση.

**Λύση:** Στο αριστερό ακραίο σημείο $x = x_j$ κάθε υποδιαστήματος, βρίσκουμε δύο συντελεστές:

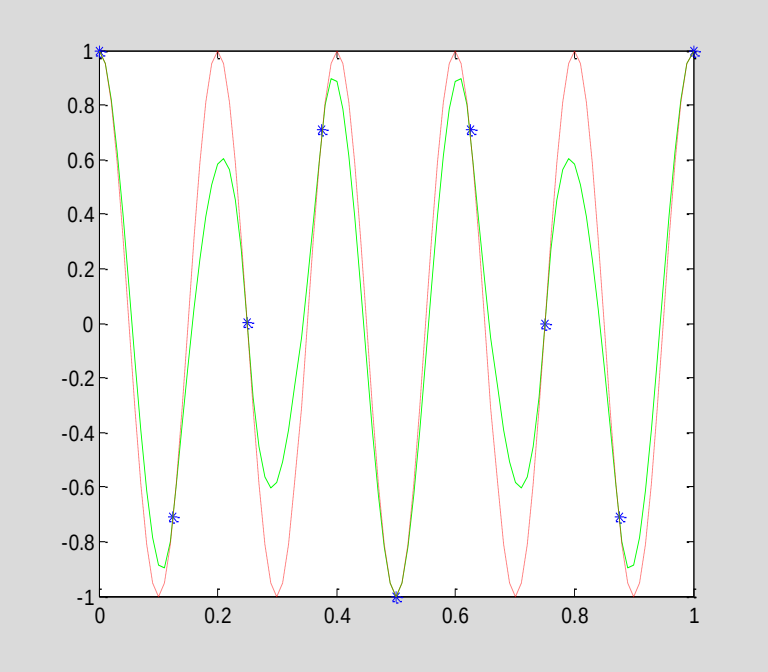
$$\boxed{a_j = y_j} \quad \text{και} \quad \boxed{b_j = s_j.}$$

Στο δεξί ακραίο σημείο $x = x_{j+1}$ κάθε υποδιαστήματος βρίσκουμε ένα σύστημα συναρτήσεων για το $c_j$, και $d_j$ που έχει την λύση:

$$\boxed{c_j = \frac{1}{h_j}\left[\ \frac{y_{j+1} - y_j}{h_j} - s_j\ \right]} \quad \text{and} \quad \boxed{d_j = \frac{1}{h_j^{\,2}}\left[\ s_{j+1} + s_j - 2\frac{y_{j+1} - y_j}{h_j}\ \right]}$$

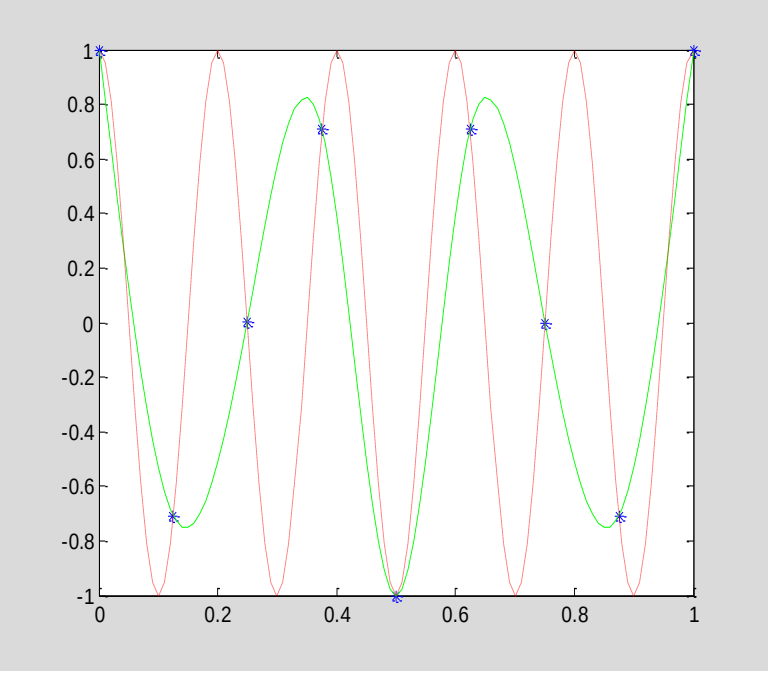
Ο παρακάτω MatLab κώδικας προσδιορίζει την κυβική spline τύπου Hermite για μια περιοδική συνάρτηση

```
% computation of coefficients of cubic Hermite interpolants from a set
[x,y]
x = 0:0.125:1;                              % data partition
y = cos(10*pi*x);                     % function values
s = -10*pi*sin(10*pi*x);      % slope values
n = length(x)-1;
h = x(2:n+1)-x(1:n); % vector of step sizes of the partition of x
a = y(1:n);
b = s(1:n);
c = ((y(2:n+1)-y(1:n))./h(1:n)-s(1:n))./h(1:n);
d = (s(2:n+1)+s(1:n)-2*(y(2:n+1)-y(1:n))./h(1:n))./(h(1:n).^2);
% evaluation of cubic Hermite interpolants at data points xInt
xInt = 0 : 0.01 : 1;       % another partition with better resolution
for j = 1 : length(xInt)
      if xInt(j) ~= x(n+1)
iInt(j) = sum(x <= xInt(j));
      else
            iInt(j) = n;
      end
end
yInt = a(iInt) + b(iInt).*(xInt-x(iInt)) + c(iInt).*(xInt - x(iInt)).^2
+ d(iInt).*(xInt - x(iInt)).^2.*(xInt-x(iInt+1));
yEx = cos(10*pi*xInt);
plot(x,y,'b*',xInt,yInt,'g',xInt,yEx,'r:');
```




Το παραπάνω παράδειγμα συνάρτησης είναι δύσκολο να προσεγγιστεί με την μέθοδο της πολυωνυμικής παρεμβολής, επειδή η συνάρτηση έχει γρήγορη ταλάντωση και δίνονται πολύ λίγα σημεία δεδομένων. Ωστόσο, η παρεμβολή με την κυβική spline Hermite προσομοιώνει ικανοποιητικά τη συμπεριφορά της συνάρτησης. Για σύγκριση, δίνουμε παρακάτω το αποτέλεσμα της κυβικής παρεμβολής spline για το ίδιο σύνολο δεδομένων. Χωρίς πληροφορίες για τις κλίσεις της συνάρτησης στα δεδομένα σημεία, η κυβική παρεμβολή spline προσεγγίζει χειρότερα την συνάρτηση.

```
x = 0:0.125:1; y = cos(10*pi*x); n = length(x)-1; h = x(2:n+1)-x(1:n);
A = 2*diag(h(1:n-1)) + 2*diag(h(2:n)) + diag(h(2:n-1),1) + diag(h(2:n-
1),-1);
b = 6*((y(3:n+1)-y(2:n))./h(2:n)-(y(2:n)-y(1:n-1))./h(1:n-1));
m = A\b'; m = [0;m;0]'; a = y(1:n);
b = (y(2:n+1)-y(1:n))./h(1:n)-h(1:n).*(m(2:n+1)+2*m(1:n))/6;
c = 0.5*m(1:n); d = (m(2:n+1)-m(1:n))./(6*h(1:n));
xInt = 0 : 0.01 : 1;       % another partition with better resolution
yInt = a(iInt) + b(iInt).*(xInt-x(iInt)) + c(iInt).*(xInt - x(iInt)).^2
+ d(iInt).*(xInt - x(iInt)).^3;
yEx = cos(10*pi*xInt);
plot(x,y,'b*',xInt,yInt,'g',xInt,yEx,'r:');
```



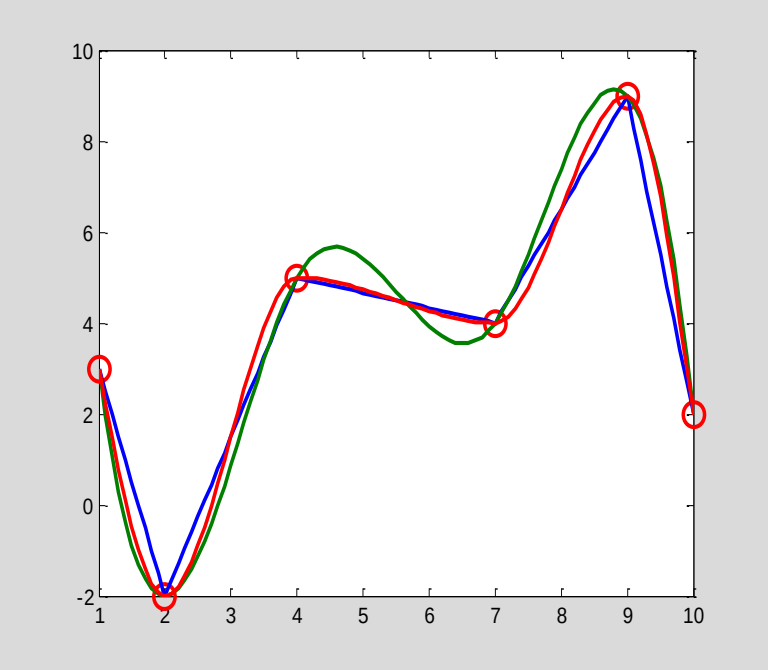

## 7. MatLab συναρτήσεις για παρεμβολή με τμηματικά πολυώνυμα

Παρακάτω δίδονται οι MATLAB συναρτήσεις για παρεμβολή με **1-δίαστασης splines**

```
yy = spline (x, y, xx)
yi = interp1 (x, y, xi, 'method')
yi = interp1 (x, y, xi, 'linear')
yi = interp1 (x, y, xi, 'spline') - not-a-knot spline
yi = interp1 (x, y,, xi, 'pchip') - cubic Hermite
yi = interp1 (x, y,, xi, 'cubic') - cubic Hermite
yi = interp1 (x, y,, xi, 'nearest') - nearest neighbor
PP = SPLINE(X,Y) - provides the piecewise polynomial form
of the
    cubic spline interpolant to the data values Y at the
data sites X
```

Παραδείγματα χρήσης των παραπάνω συναρτήσεων είναι

```
x=[1 2 4 7 9 10];
y=[3 -2 5 4 9 2];
xi=1:0.1:10;
y1=interp1(x,y,xi,'linear');
y2=interp1(x,y,xi,'spline');
y3=interp1(x,y,xi,'cubic');
plot(x,y,'ro',xi,y1,xi,y2,xi,y3,'LineWidth',2,'MarkerSize',12)
print -djpeg spline00.jpg
```




```
x=[-1, 0, 1.27, 2.55, 3.82, 4.92, 5.02];
y=[-14.58, 0., 0., 0., 0., .88, 11.17];
xInt = -1 : 0.01 : 5.02; yInt =
interp1(x,y,xInt);plot(xInt,yInt,'g',x,y,'b*');axis([-1,5.02,-13,12]);
```

```
x = [ -1,-.866,-
.5,0,0.5,0.866,1,1.0402,1.15,1.3,1.54,1.828,2.1736,2.5883,3.086];
y = [ 0,-0.25,-0.433,-0.5,-0.433,-
0.25,0,0.15,0.2598,0.3,0.3,0.3,0.3,0.3,0.3];
xInt = -1 : 0.01 : 3; yInt = interp1(x,y,xInt); % default linear
interpolation
plot(x,y,'b*',xInt,yInt,'g'); axis([-1,3,-1,1]);
```

```
yInt = interp1(x,y,xInt,'spline'); % piecewise cubic spline
interpolation
plot(x,y,'b*',xInt,yInt,'g'); axis([-1,3,-1,1]);
```

```
yInt = interp1(x,y,xInt,'cubic'); % piecewise cubic Hermite
interpolation
plot(x,y,'b*',xInt,yInt,'g'); axis([-1,3,-1,1]);
```

Επιπλέον, οι MATLAB συναρτήσεις σχετικές με τις splines είναι

- **ppval:** υπολογιζει τις τιμές των κυβικών splines (συναρτήσεων) και των συναρτήσεων παρεμβολής Hermite στις pp-αναπαραστάσεις τους

```
y1 = ppval(spline(x,y),-0.25)
  % computes the value of y at x = -0.25 of the cubic spline
interpolant
y2 = ppval(pchip(x,y),-0.25)
      % computes the value of y at x = -0.25 of the cubic Hermite
interpolant

y1 =
   -1.9560
y2 =
   -1.2936
```

- **unmkpp:** εξάγει κόμβους, συντελεστές, πλήθος τμημάτων και βαθμό από την κυβική συνάρτηση παρεμβολής και τις συναρτήσεις παρεμβολής Hermite στις pp-αναπαραστάσεις τους.

```
x = [0,1,2,3]; y = humps(x);
[P,R,n,m] = unmkpp(spline(x,y))
      % P is the vector of x containing break points (length: n+1)
      % R is the matrix of coefficients of cubic spline interpolants
(size: n-by-m)
      % n - number of pieces of the interpolants (length(x)-1)
  % m - number of coefficients in each piece (order of the polynomial +
1)
   % Example: n = 3 and m = 4
      % Sj(x) = aj + bj*(x-xj) + cj*(x-xj)^2 + dj*(x-xj)^3 - piece
between [xj,xj+1]
      % aj = R(:,4); bj = R(:,3); cj = R(:,2); dj = R(:,1);
P =    0     1     2     3
R =         8.6251  -41.7147   43.9131    5.1765
            8.6251  -15.8394  -13.6409   16.0000
            8.6251   10.0360  -19.4443   -4.8552
n =     3
m =     4

x = [0,1,2,3]; y = humps(x);
[P,R,n,m] = unmkpp(pchip(x,y))
P =    0     1     2     3
R =         5.0158  -20.8552   26.6629    5.1765
            40.2008  -61.0560         0   16.0000
            0.0567    0.6698   -1.5096   -4.8552
n =     3
m =     4
```

- **mkpp:** αποδίδει μια pp-αναπαράσταση ενός τμηματικού πολυώνυμου από τους κόμβους του και τους συντελεστές του.

```
      x = [0,1,2,3]; y = humps(x)
[P,R,n,m] = unmkpp(spline(x,y));
SS = mkpp(P,R) % the same as SS = spline(x,y)
  yy = ppval(SS,x) % evaluation of pp-representation of a polynomial
y =    5.1765   16.0000   -4.8552   -5.6383
```

```
SS =         form: 'pp'
                   breaks: [0 1 2 3]
             coefs: [3x4 double]
                   pieces: 3
             order: 4
                   dim: 1
yy =    5.1765   16.0000   -4.8552   -5.6383
```

## 8. Δύο διαστάσεων splines παρεμβολής στην MatLab

Η MATLAB συνάρτηση **zi = interp2 (x, y, z, xi, yi)** ορίζεται ως προς το παρακάτω πλέγμα σημείων

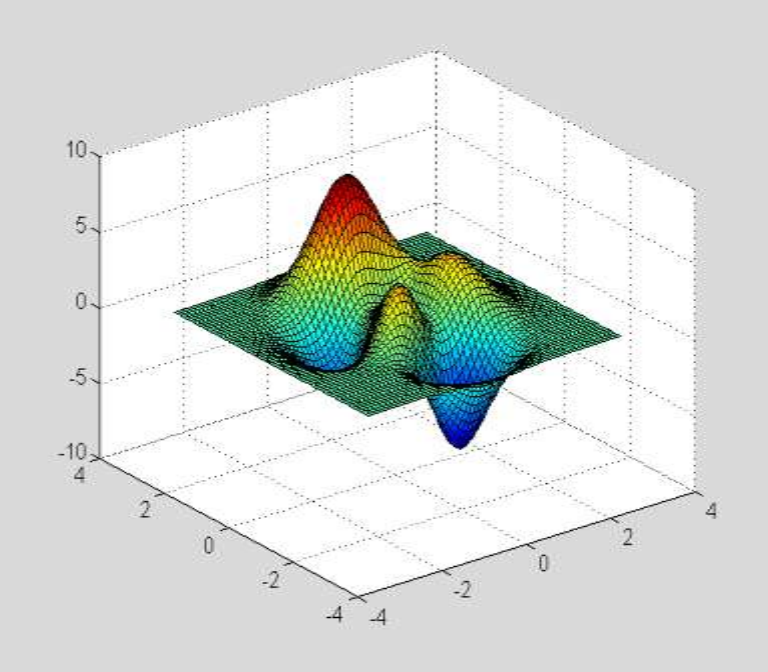
$$\begin{cases} (x_1, y_1), (x_1, y_2), \cdots\cdots, (x_1, y_m) \\ (x_2, y_1), (x_2, y_2), \cdots\cdots, (x_2, y_m) \\ \quad \vdots \qquad\quad \vdots \qquad\qquad \vdots \\ (x_n, y_1), (x_n, y_2), \cdots\cdots, (x_n, y_m) \end{cases}$$

$$z(i, j) = f(x(i), y(j))$$

Ο παρακάτω κώδικας είναι ένα παράδειγμα χρήσης της ***MATLAB συνάρτησης*** **interp2**

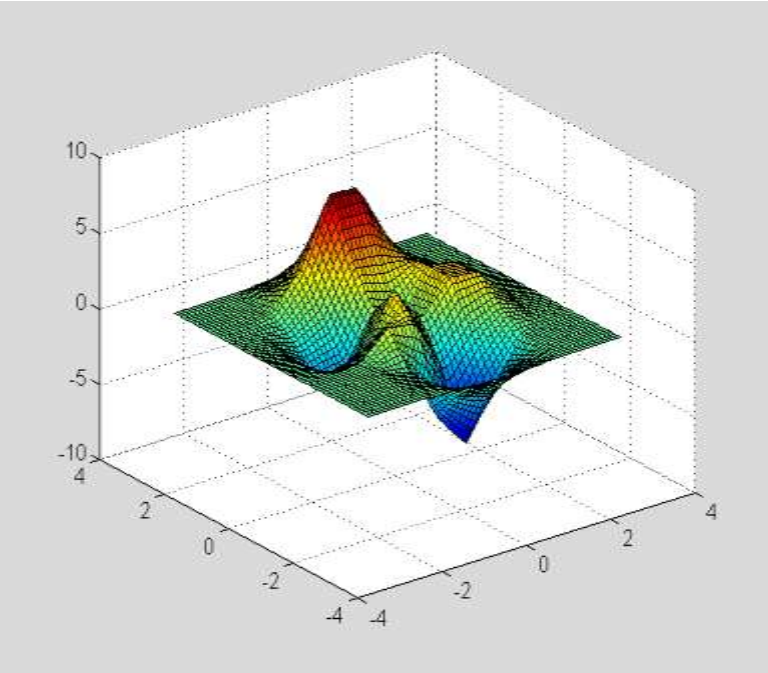
```
[x,y,z]=peaks(100); [xi,yi]=meshgrid(-3:0.1:3,-3:0.1:3);
zi = interp2(x,y,z,xi,yi); surf(xi,yi,zi)
print -djpeg075 peaks1.jpg
```

Γραφική Παράσταση λύσης για 100 ×100 σημεία




```
[x,y,z]=peaks(10); [xi,yi]=meshgrid(-3:0.1:3,-3:0.1:3);
zi = interp2(x,y,z,xi,yi); surf(xi,yi,zi)
print -djpeg075 peaks2.jpg
```

Γραφική Παράσταση λύσης για 10 ×10 σημεία



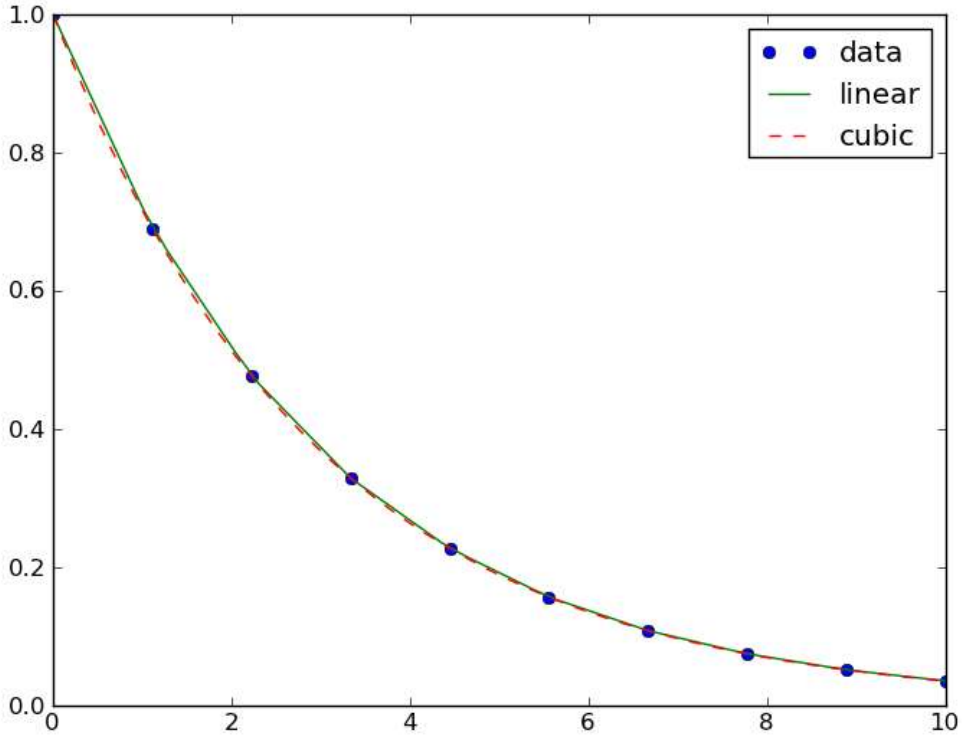
## 9. Λογισμικό για splines παρεμβολής στο περιβάλλον Python

### α) Παρεμβολή με γραμμικές και κυβικές spline χρησιμοποιώντας συναρτήσεις της βιβλιοθήκης SciPy

Η βιβλιοθήκη SciPy υποστηρίζει την συνάρτηση `interp1d` που υπολογίζει την γραμμική και κυβική spline για ένα σύνολο δεδομένων σημείων. Η βιβλιοθήκη περιγράφεται στην ιστοσελίδα http://docs.scipy.org/doc/scipy/reference/tutorial/index.html.
Παρακάτω δίδεται κώδικας χρήσης της συνάρτησης `interp1d` για την παρεμβολή της συνάρτησης $e^{-x/3}$ με γραμμικές και κυβικές splines

```
from scipy.interpolate import interp1d
import numpy as np
x = np.linspace(0, 10, 10)
y = np.exp(-x/3.0)
f = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')
xnew = np.linspace(0, 10, 40)
import matplotlib.pyplot as plt
plt.plot(x,y,'o',xnew,f(xnew),'-', xnew, f2(xnew),'--')
plt.legend(['data', 'linear', 'cubic'], loc='best')
plt.show()
```

OUTPUT




## β) Αλγόριθμος παρεμβολής με κυβικές spline στο περιβάλλον PYTHON

Θυμηθείτε ότι η πρώτη φάση του προσδιορισμού της φυσικής κυβικής spline παρεμβολής είναι να βρούμε τις εξισώσεις που ικανοποιεί η καμπυλότητα της spline $m_j = S''_j(x_j)$ στα σημεία παρεμβολής (κόμβους) όπου $m_0 = m_{n+1} = 0$ και να βρούμε την λύση τους. Αυτή η φάση θα υλοποιηθεί από την συνάρτηση **`curvatures`** . Η δεύτερη φάση είναι να υπολογίσουμε την κυβική spline παρεμβολής σε ένα τυχαίο σημείο x. Η φάση αυτή υλοποιείται από την συνάρτηση **`evalSpline`**. Η συνάρτηση **`findSegment`** εμπεριέχεται στην evalSpline και βρίσκει το υποδιάστημα της spline που περιέχει το *x* χρησιμοποιώντας την μέθοδο της διχοτόμησης (bisection). Περισσότερες λεπτομέρειες για την λύση PYTHON θα βρείτε στο βιβλίο "*Numerical Methods in Engineering with PYTHON*, Cambringe University Press, 2005" του Jann Kiusallas. Τα υποπρογράμματα **`LUdecomp3(c,d,e)`**, **`LUsolve3(c,d,e,k)`** λύνουν το γραμμικό σύστημα για την εύρεση της καμπυλότητας της splines στα σημεία παρεμβολής.

```
## module cubicSpline
''' k = curvatures(xData,yData).
    Returns the curvatures of cubic spline at its knots.

    y = evalSpline(xData,yData,k,x).
    Evaluates cubic spline at x. The curvatures k can be
    computed with the function 'curvatures'.
'''
from numarray import zeros,ones,Float64,array
```

```
from LUdecomp3 import *

def curvatures(xData,yData):
    n = len(xData) - 1
    c = zeros((n),type=Float64)
    d = ones((n+1),type=Float64)
    e = zeros((n),type=Float64)
    k = zeros((n+1),type=Float64)
    c[0:n-1] = xData[0:n-1] - xData[1:n]
    d[1:n] = 2.0*(xData[0:n-1] - xData[2:n+1])
    e[1:n] = xData[1:n] - xData[2:n+1]
    k[1:n] =6.0*(yData[0:n-1] - yData[1:n]) \
                /(xData[0:n-1] - xData[1:n]) \
             -6.0*(yData[1:n] - yData[2:n+1])   \
                /(xData[1:n] - xData[2:n+1])
    LUdecomp3(c,d,e)
    LUsolve3(c,d,e,k)
    return k

def evalSpline(xData,yData,k,x):

    def findSegment(xData,x):
        iLeft = 0
        iRight = len(xData)- 1
        while 1:
            if (iRight-iLeft) <= 1: return iLeft
            i =(iLeft + iRight)/2
            if x < xData[i]: iRight = i
            else: iLeft = i

    i = findSegment(xData,x)
    h = xData[i] - xData[i+1]
    y = ((x - xData[i+1])**3/h - (x - xData[i+1])*h)*k[i]/6.0 \
      - ((x - xData[i])**3/h - (x - xData[i])*h)*k[i+1]/6.0   \
      + (yData[i]*(x - xData[i+1])                            \
       - yData[i+1]*(x - xData[i]))/h
    return y



## module LUdecomp3
''' c,d,e = LUdecomp3(c,d,e).
    LU decomposition of tridiagonal matrix [c\d\e]. On output
    {c},{d} and {e} are the diagonals of the decomposed matrix.

    x = LUsolve3(c,d,e,b).
    Solves [c\d\e]{x} = {b}, where {c}, {d} and {e} are the
    vectors returned from LUdecomp3.
'''

def LUdecomp3(c,d,e):
    n = len(d)
    for k in range(1,n):
        lam = c[k-1]/d[k-1]
        d[k] = d[k] - lam*e[k-1]
        c[k-1] = lam
    return c,d,e
```

```
def LUsolve3(c,d,e,b):
    n = len(d)
    for k in range(1,n):
        b[k] = b[k] - c[k-1]*b[k-1]
    b[n-1] = b[n-1]/d[n-1]
    for k in range(n-2,-1,-1):
        b[k] = (b[k] - e[k]*b[k+1])/d[k]
    return b
```

Ο παρακάτω κώδικας βρίσκει την κυβική spline που παρεμβάλει τα σημεία
`xData = array([0.15,2.3,3.15,4.85,6.25,7.95])` και
`yData = array([4.79867,4.49013,4.2243,3.47313,2.66674,1.51909])`που
ανήκουν στην συνάρτηση `4.8*cos(pi*x/20.0)`

```
#!/usr/bin/python
## example3_4
from scipy import array,arange
from math import pi,cos
from newtonPoly import *

xData = array([0.15,2.3,3.15,4.85,6.25,7.95])
yData = array([4.79867,4.49013,4.2243,3.47313,2.66674,1.51909])
a = coeffts(xData,yData)
print " x    yInterp   yExact"
print "-----------------------"
for x in arange(0.0,8.1,0.5):
    y = evalPoly(a,xData,x)
    yExact = 4.8*cos(pi*x/20.0)
    print "%3.1f %9.5f %9.5f"% (x,y,yExact)
raw_input("\nPress return to exit")
```

OUTPUT
Παράγει τον παρακάτω πίνακα τιμών για την spline παρεμβολής και την ακριβή τιμή της συνάρτησης.

```
x   yInterp   yExact
-----------------------
0.0  4.80003  4.80000
0.5  4.78518  4.78520
1.0  4.74088  4.74090
1.5  4.66736  4.66738
2.0  4.56507  4.56507
2.5  4.43462  4.43462
3.0  4.27683  4.27683
3.5  4.09267  4.09267
4.0  3.88327  3.88328
4.5  3.64994  3.64995
5.0  3.39411  3.39411
5.5  3.11735  3.11735
6.0  2.82137  2.82137
6.5  2.50799  2.50799
7.0  2.17915  2.17915
7.5  1.83687  1.83688
8.0  1.48329  1.48328
```

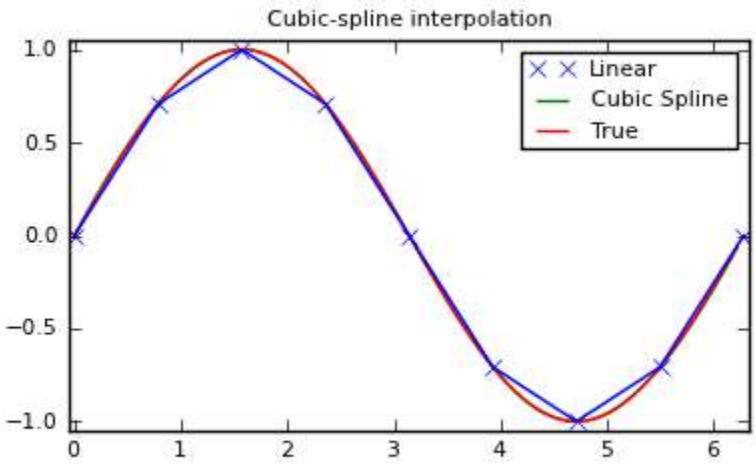
### γ) Υπολογισμοί με ρουτίνες της βιβλιοθήκης scipy.interpolate

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from scipy import interpolate
```

Cubic-spline

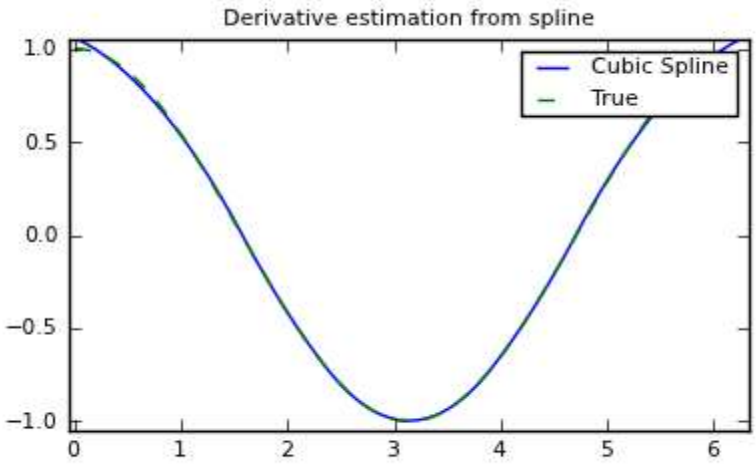
```
>>> x = np.arange(0,2*np.pi+np.pi/4,2*np.pi/8)
>>> y = np.sin(x)
>>> tck = interpolate.splrep(x,y,s=0)
>>> xnew = np.arange(0,2*np.pi,np.pi/50)
>>> ynew = interpolate.splev(xnew,tck,der=0)
>>> plt.figure()
>>> plt.plot(x,y,'x',xnew,ynew,xnew,np.sin(xnew),x,y,'b')
>>> plt.legend(['Linear','Cubic Spline', 'True'])
>>> plt.axis([-0.05,6.33,-1.05,1.05])
>>> plt.title('Cubic-spline interpolation')
>>> plt.show()
```

(Source code)



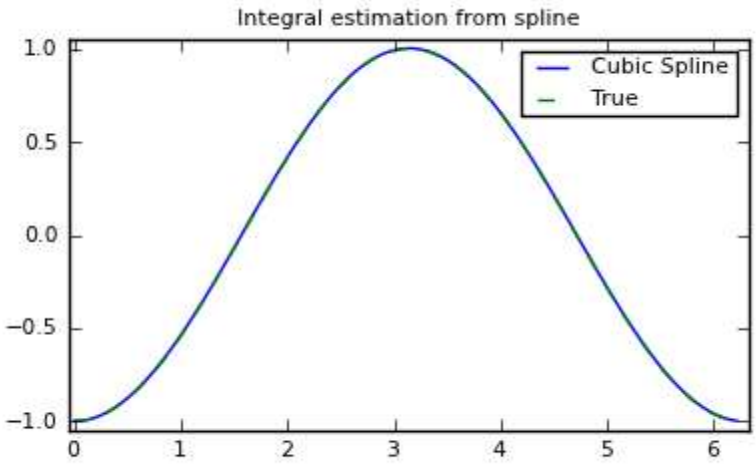

Derivative of spline

```
>>> yder = interpolate.splev(xnew,tck,der=1)
>>> plt.figure()
>>> plt.plot(xnew,yder,xnew,np.cos(xnew),'--')
>>> plt.legend(['Cubic Spline', 'True'])
>>> plt.axis([-0.05,6.33,-1.05,1.05])
>>> plt.title('Derivative estimation from spline')
>>> plt.show()
```




## Integral of spline

```
>>> def integ(x,tck,constant=-1):
>>>     x = np.atleast_1d(x)
>>>     out = np.zeros(x.shape, dtype=x.dtype)
>>>     for n in xrange(len(out)):
>>>         out[n] = interpolate.splint(0,x[n],tck)
>>>     out += constant
>>>     return out
>>>
>>> yint = integ(xnew,tck)
>>> plt.figure()
>>> plt.plot(xnew,yint,xnew,-np.cos(xnew),'--')
>>> plt.legend(['Cubic Spline', 'True'])
>>> plt.axis([-0.05,6.33,-1.05,1.05])
>>> plt.title('Integral estimation from spline')
>>> plt.show()
```

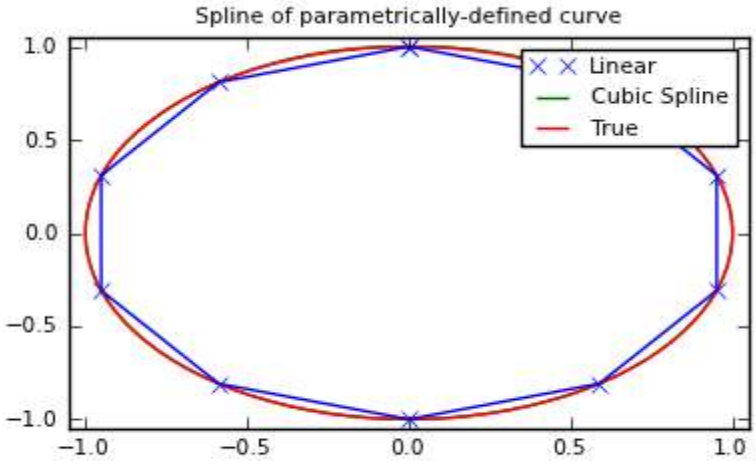



## Roots of spline

```
>>> print interpolate.sproot(tck)
[ 0.      3.1416]
```

## Parametric spline

```
>>> t = np.arange(0,1.1,.1)
>>> x = np.sin(2*np.pi*t)
>>> y = np.cos(2*np.pi*t)
>>> tck,u = interpolate.splprep([x,y],s=0)
>>> unew = np.arange(0,1.01,0.01)
>>> out = interpolate.splev(unew,tck)
>>> plt.figure()
>>>
plt.plot(x,y,'x',out[0],out[1],np.sin(2*np.pi*unew),np.cos(2*np.pi*unew
),x,y,'b')
>>> plt.legend(['Linear','Cubic Spline', 'True'])
>>> plt.axis([-1.05,1.05,-1.05,1.05])
>>> plt.title('Spline of parametrically-defined curve')
>>> plt.show()
```




## 10. Βιβλιοθήκη παρεμβολής στο περιβάλλον Python: scipy.interpolate

Το **scipy.interpolate** πακέτο υλοποιεί διαφόρους μεθόδους παρεμβολής με splines σε μια και δύο διαστάσεις. Το περιεχόμενο της βιβλιοθήκης **scipy.interpolate** δίδεται παρακάτω ή με την εντολή help(scipy.optimize).

- Interpolation (**scipy.interpolate**)
  - 1-D interpolation (**interp1d**)
  - Multivariate data interpolation (**griddata**)
  - Spline interpolation
  - Spline interpolation in 1-d: Procedural (interpolate.splXXX)
  - Spline interpolation in 1-d: Object-oriented (**UnivariateSpline**)
  - Two-dimensional spline representation: Procedural (**bisplrep**)
  - Two-dimensional spline representation: Object-oriented (**BivariateSpline**)
- Using radial basis functions for smoothing/interpolation
  - 1-d Example
  - 2-d Example

## 11. Αναφορές

1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. http://www.esm.psu.edu/courses/emch407/
3. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
4. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
5. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
6. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.

7. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.
8. An introduction to splines for use in computer graphics and geometric modeling, R.H. Bartels, J.C. Beatty, B.A. Barsky, Morgan Kaufmann Publishers, Inc., 1987.

# ΚΕΦΑΛΑΙΟ 3: ΠΑΡΕΜΒΟΛΗ ΣΕ 2Δ ΔΙΑΣΤΑΣΕΙΣ & ΣΦΑΛΜΑΤΑ ΠΟΛΥΩΝΥΜΙΚΗΣ ΠΑΡΕΜΒΟΛΗΣ

## Περιεχόμενα



### 1. Διδιάστατη παρεμβολή

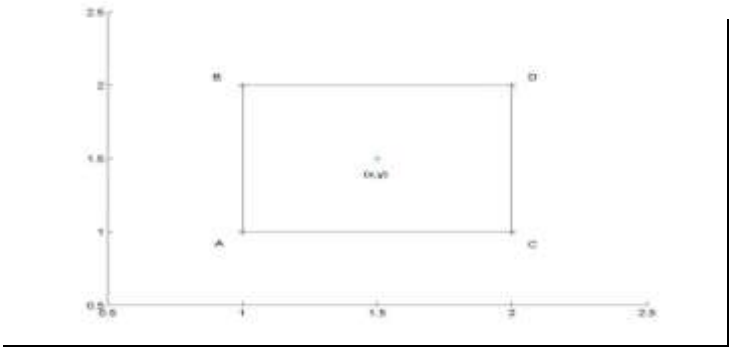
**Πρόβλημα:** Δίδονται τέσσερα σημεία σε ένα ορθογώνιο πλέγμα στο χώρο των τριών διαστάσεων:

$$(x_1,y_1,z_1);\ (x_2,y_2,z_2);\ (x_3,y_3,z_3);\ (x_4,y_4,z_4)$$

Βρείτε τις τιμές του ***z*** στα εσωτερικά σημεία του ορθογωνίου

**Παράδειγμα:** Θεωρείστε τα τέσσερα σημεία δεδομένων

| | *A* | *B* | *C* | *D* |
|---|---|---|---|---|
| *x* | *1* | *1* | *2* | *2* |
| *y* | *1* | *2* | *1* | *2* |
| *z* | *5* | *8* | *7* | *10* |



τα οποία ανήκουν στο επίπεδο: ***z = 2x + 3y.*** Βρείτε την παρεμβολή της συνάρτησης στο εσωτερικό (κεντρικό) σημείο ***(x,y).***

**Interpl2:** υπολογίζει την παρεμβολή των τιμών του ***z*** στα σημεία ***(x,y)*** από ένα σύνολο δεδομένων τιμών $(x_i,y_i,z_i)$.

```
x = [ 1,2]; y = [1,2]; [X,Y] = meshgrid(x,y)
z = 2*X+3*Y % data values at the vertex points of (x,y)
xInt = 1.5; yInt = 1.5; zInt = interp2(x,y,z,xInt,yInt)
zExact = 2*xInt + 3*yInt % exact value at the center point
```

```
X =          1     2
             1     2
Y =    1     1
       2     2
z =    5     7
       8    10
```

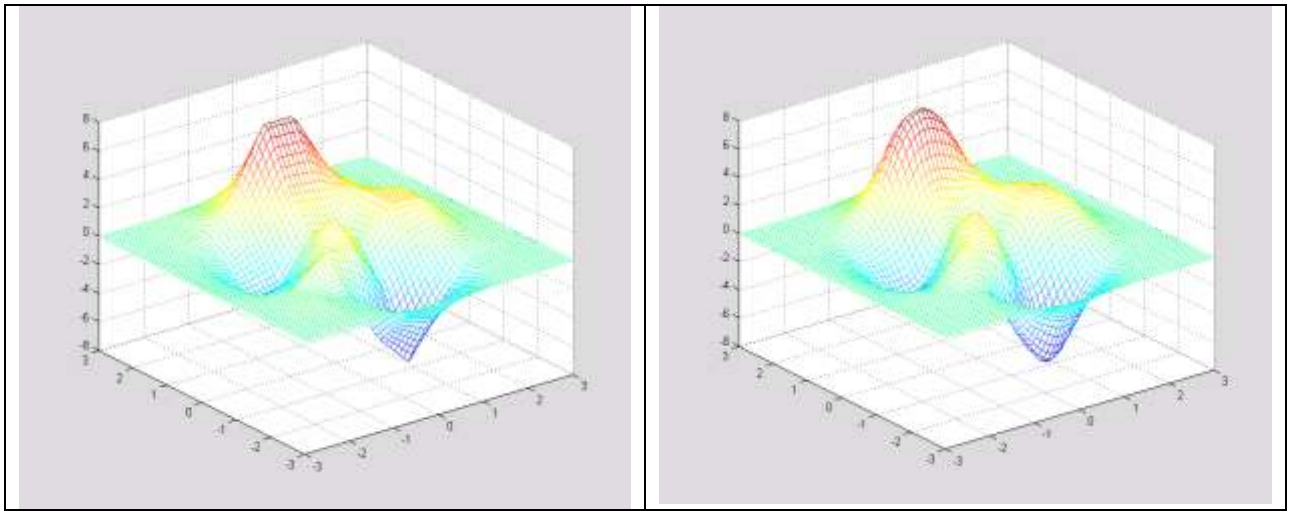
```
zInt =    7.5000
zExact =    7.5000
```

```
  x = [1,2,3]; y = [1,2,3]; [X,Y] = meshgrid(x,y); z = 2*X+3*Y;
zInt = interp2(x,y,z,xInt,yInt,'linear') % bilinear interpolation
zInt = interp2(x,y,z,xInt,yInt,'spline') % bicubic spline interpolation
zInt = interp2(x,y,z,xInt,yInt,'cubic') % bicubic Hermite interpolation
```

```
zInt =    7.5000
zInt =    7.5000
zInt =    7.5000
```

**datagrid:** προσαρμόζει μια διδιάστατη spline σε ανομοιόμορφα δεδομένα των οποίων οι συντεταγμένες τοποθετούνται στα διανύσματα (X,Y,Z). Οι μέθοδοι παρεμβολών στο **"datagrid"** είναι διαφορετικοί από αυτές στο **"interpl2"** και βασίζονται στην τριγωνοποίηση δεδομένων Delaunay.

```
[x,y,z] = peaks(10); [xi,yi] = meshgrid(-3:.1:3,-3:.1:3);
 zi = interp2(x,y,z,xi,yi,'cubic'); mesh(xi,yi,zi)
```



**Λύση του προβλήματος με splines βαθμού 1:**

Μονοδιάστατη παρεμβολή κατά μήκος του [A,B]:

$$z_{[A,B]} = z_A \frac{y - y_B}{y_A - y_B} + z_B \frac{y - y_A}{y_B - y_A}$$

Μονοδιάστατη παρεμβολή κατά μήκος του [C,D]:

$$z_{[C,D]} = z_C \frac{y - y_D}{y_C - y_D} + z_D \frac{y - y_C}{y_D - y_C}$$

Μονοδιάστατη παρεμβολή κατά μήκος του [(A,B), (C,D)]:

$$z = z_{[A,B]} \frac{x - x_C}{x_A - x_C} + z_{[C,D]} \frac{x - x_A}{x_C - x_A}$$

MatLab κώδικας για την παραπάνω λύση

```
x = [1,2]; y = [1,2]; z = [ 5,7;8,10]; xInt = 1.5; yInt = 1.5;
zAB = z(1,1)*(yInt-y(2))/(y(1)-y(2))+z(2,1)*(yInt-y(1))/(y(2)-y(1));
zCD = z(1,2)*(yInt-y(2))/(y(1)-y(2))+z(2,2)*(yInt-y(1))/(y(2)-y(1));
zInt = zAB*(xInt-x(2))/(x(1)-x(2))+zCD*(xInt-x(1))/(x(2)-x(1))

zInt =    7.5000
```

## 2. Σφάλματα στην πολυωνυμική παρεμβολή

Το σφάλμα προσέγγισης μεταξύ της συνάρτησης $y = f(x)$ και του πολυωνύμου παρεμβολής $y = P_n(x)$ ανάμεσα σε $(n+1)$ σημεία δεδομένων είναι ανάλογη με το υπόλοιπο του πολυωνύμου $(n+1)$ βαθμού $\frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_1)(x-x_2)...(x-x_{n+1})$ :

$$| f(x) - P_n(x) | \le \frac{M_{n+1}}{(n+1)!} |(x-x_1)(x-x_2)\ldots(x-x_n)(x-x_{n+1})|, \qquad M_{n+1} = \max_{x_1 < x < x_{n+1}} | f^{(n+1)}(x)|$$

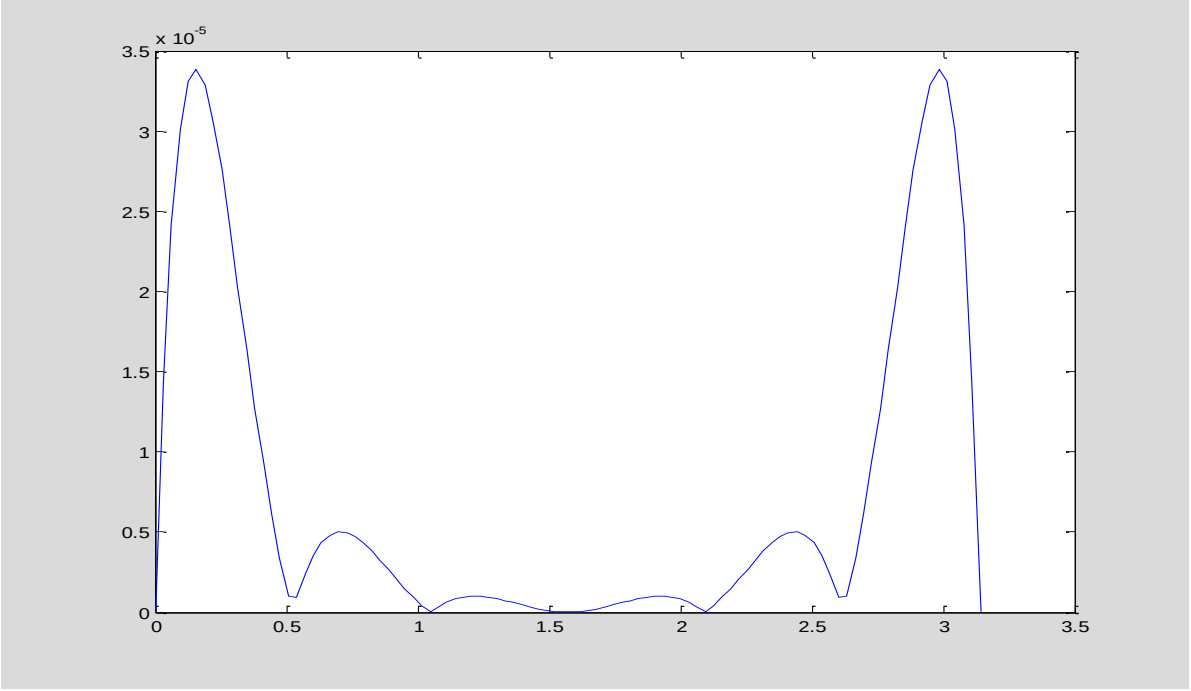
Εάν τα σημεία δεδομένων είναι ισαπέχοντα με σταθερό πλάτος βήματος $h$, τότε το τοπικό σφάλμα της πολυωνυμικής παρεμβολής $e_n(x) = | f(x) - P_n(x) |$ φράσεται από:

$$| f(x) - P_n(x) | \le \frac{M_{n+1}}{4(n+1)} h^{n+1}$$

Το σφάλμα μειώνεται όταν το πλάτος βήματος $h$ γίνεται μικρότερο με ένα σταθερό αριθμό δεδομένων σημείων $(n+1)$. Εάν το διάστημα παρεμβολής είναι σταθερό, ο αριθμός των σημείων δεδομένων $(n+1)$ αυξάνεται με μικρότερο πλάτος βήματος. Σε αυτή την περίπτωση το σφάλμα αποκοπής μειώνεται όσο αυξάνεται η τιμή του $n$. Ωστόσο, το σφάλμα στρογγυλοποίησης αυξάνεται τιμές όσο αυξάνεται η τιμή του $n$. Ως αποτέλεσμα, η πολυωνυμική παρεμβολή γίνεται χειρότερη με μεγαλύτερο αριθμό σημείων μετά από κάποιο βέλτιστο αριθμό $n = n_{opt}$!

Στο παρακάτω κώδικα υπολογίζουμε το σφάλμα πολυωνυμικής παρεμβολής για την συνάρτηση sin(x) στο διάστημα [0,π]:

```
x = linspace(0,pi,7); y = sin(x);
       % given function with 7 equispaced data points
c = polyfit(x,y,6); xInt = linspace(0,pi,100); yInt = polyval(c,xInt);
                  % polynomial interpolation on a tense grid
yExact = sin(xInt); eLocal = abs(yExact-yInt);
                  % exact values and local error e(x) of the polynomial
interpolation
plot(xInt,eLocal)
```




Το τοπικό σφάλμα $e_n(x)$ εξαφανίζεται στα δεδομένα σημεία, δηλαδή στα σημεία παρεμβολής $x = x_1,x_2,\ldots,x_m,x_{n+1}$ όπως θα περιμέναμε. Το τοπικό σφάλμα έχει τοπικά μέγιστα στα μέσα των διαστημάτων μεταξύ των δύο παρακείμενων σημείων. Το τοπικό σφάλμα είναι μεγαλύτερο στα άκρα του διαστήματος παρεμβολής. Το τοπικό σφάλμα είναι μικρότερο στη μέση του διαστήματος παρεμβολής. Αυτές οι ιδιότητες είναι συνηθισμένες για πολυωνυμική παρεμβολή με ισαπέχοντα σημεία δεδομένων. Μπορούν να δικαιολογηθούν από τη συμπεριφορά του υπολοίπου του πολυωνύμου $(n+1)$ βαθμού.

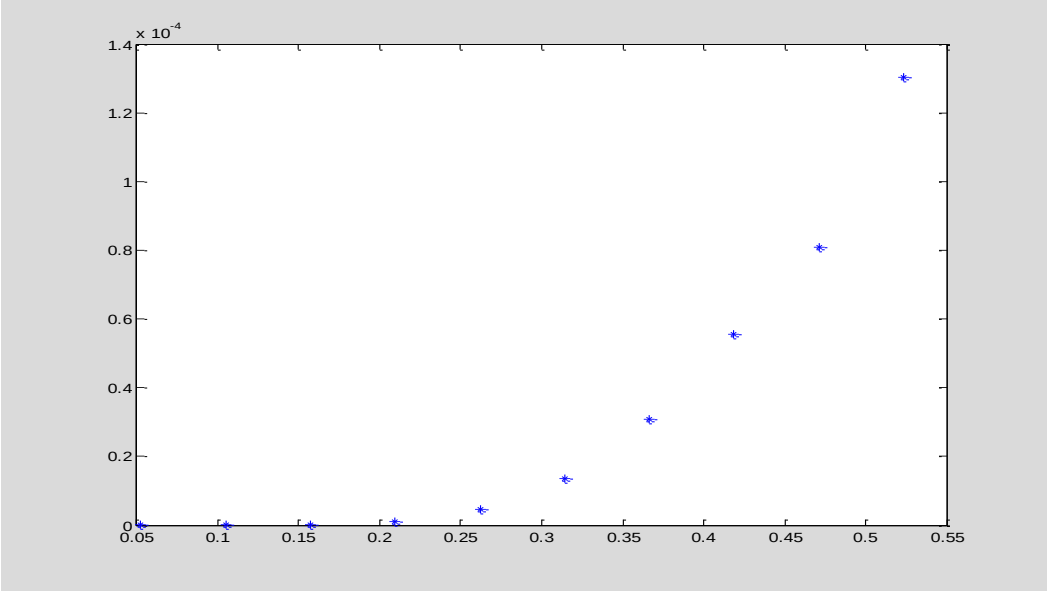
Για την μελέτη της υπολογιστικής διαφοράς του σφάλματος θα χρησιμοποιήσουμε MatLab συνάρτηση polyfit. Η συνάρτηση αυτή βρίσκει το πολυώνυμο παρεμβολής, αν ο αριθμός των δεδομένων σημείων είναι ίσος με το βαθμό του πολυωνύμου, διαφορετικά βρίσκει το πολυώνυμο που προσεγγίζει την συνάρτηση με την μέθοδο των ελαχίστων τετραγώνων που θα την μελετήσουμε στο επόμενο κεφάλαιο.

**Ο παρακάτω κώδικας**
α) υπολογίζει το σφάλμα προσέγγισης της συνάρτησης sin(x) με πολυώνυμα σε διάφορες νόρμες όταν ι) το h μικραίνει και το n διατηρείτε σταθερό και ιι) το h μικραίνει ή το n μεγαλώνει και το διάστημα ορισμού παραμένει σταθερό και
β) το σφάλμα τριγωνομετρικής παρεμβολής της ίδιας συνάρτησης.
Τα συμπεράσματα διατυπώνονται σαν σχόλια του κώδικα.

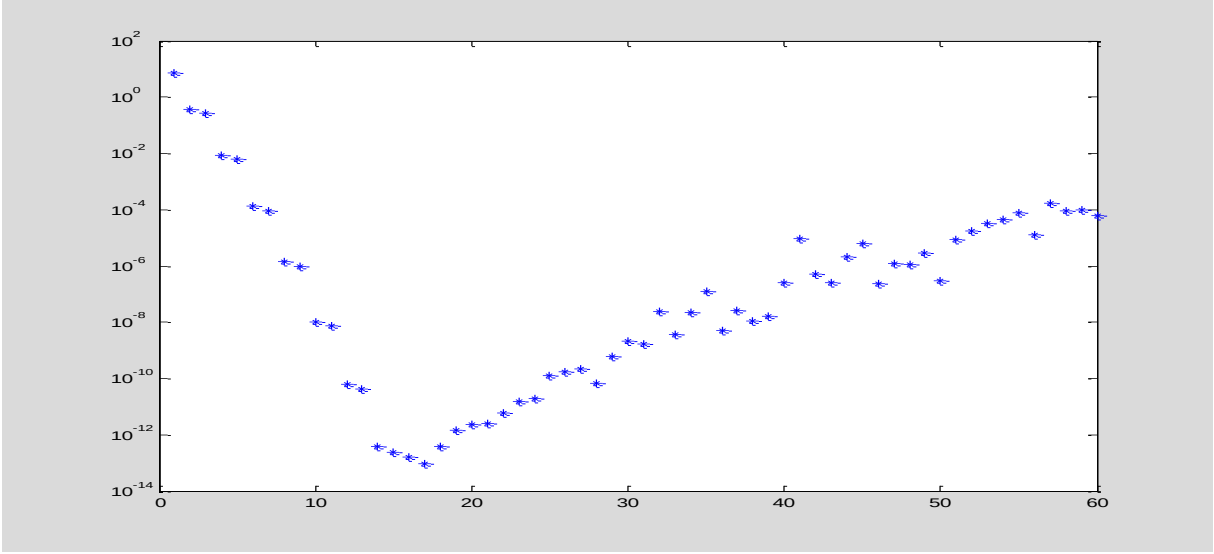
```
 % error of polynomial interpolation with smaller h and fixed n
      m = 10;
for k = 1 : m
  x = linspace(0,pi*(m-k+1)/m,7); y = sin(x);
  h(k) = x(2)-x(1); % step size of the data points
  c = polyfit(x,y,6);
  xInt = linspace(0,pi*(m-k+1)/m,100); yInt = polyval(c,xInt);
      yExact = sin(xInt); eLocal = abs(yExact-yInt);
```

```
      eGlobal(k) = norm(eLocal,2); % global error as the total mean
square error
end
plot(h,eGlobal,'*')
a = polyfit(log(h),log(eGlobal),1);
      % Checking the theory: eGlobal = c h^(n+1)
      % In log-log scale: log(eGlobal) = (n+1) log(h) + log(c)
      % The first coefficient of the linear regression must be
approximately 7
power = a(1)
```

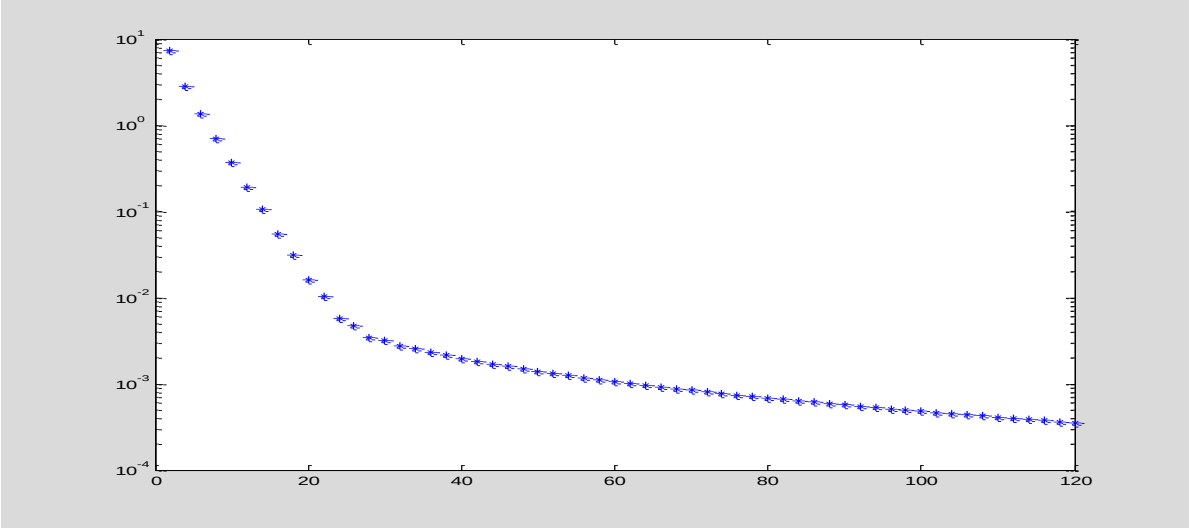
```
power =    6.2531
```




```
% error of polynomial interpolation with smaller h and fixed interval
[0,pi]
m = 60;
for n = 1 : m
   x = linspace(0,pi,n+1); y = sin(x);
   h(n) = x(2)-x(1); % step size of the data points
   c = polyfit(x,y,n);
   xInt = linspace(0,pi,100); yInt = polyval(c,xInt);
      yExact = sin(xInt); eLocal = abs(yExact-yInt);
      eGlobal(n) = norm(eLocal,2); % global error as the total mean
square error
end
semilogy(eGlobal,'*');
% the optimal number n = nOpt occurs where the total of
% truncation error and rounding error reaches the minimal value
% The optimal number is approximately nOpt = 18
% The polynomial interpolation becomes worse with larger values of n.
```




```
% error of trigonometric interpolation with smaller h and fixed
interval [-1,1]
Nmax = 60;
for m = 1 : Nmax
   x = linspace(-1,1,2*m+1); y = 1./(1 + 25*x.^2);
   n = 2*m; L = 2; xx = [x(m+1:n),x(1:m)]'; yy = [y(m+1:n),y(1:m)]';
   for j = 0 : m
      a(j+1) = 2*yy'*cos(2*pi*j*xx/L)/n;
      b(j+1) = 2*yy'*sin(2*pi*j*xx/L)/n;
   end
   xInt = linspace(-1,1,201); yInt = 0.5*a(1)*ones(1,length(xInt));
   for j = 1 : (m-1)
       yInt = yInt + a(1+j)*cos(2*pi*j*xInt/L) +
b(1+j)*sin(2*pi*j*xInt/L);
   end
   yInt = yInt + a(m+1)*cos(2*pi*m*xInt/L);
       yExact = 1./(1 + 25*xInt.^2); eLocal = abs(yExact-yInt);
       eGlobal(n) = norm(eLocal,2); % global error as the total mean
square error
end
semilogy(eGlobal,'*');
% The error of trigonometric interpolation decreases with larger n
```




**Παρατήρηση:** Στην περίπτωση της πολυωνυμικής προσέγγισης και για σταθερό πεδίο ορισμού της συνάρτησης το σφάλμα μικραίνει μέχρι ορισμένη τιμή του n=18 και μετά μεγαλώνει. Στην περίπτωση της τριγωνομετρικής προσέγγισης το σφάλμα μικραίνει για αυξανόμενες τιμές του n μέχρι να ξεπεράσουμε την ακρίβεια της μηχανής.

## 3. Σφάλματα στην παρεμβολή με τμηματικά πολυώνυμα

## 4. Αναφορές

1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
3. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
4. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
5. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
6. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.

# ΚΕΦΑΛΑΙΟ 4: ΤΡΙΓΩΝΟΜΕΤΡΙΚΗ ΠΑΡΕΜΒΟΛΗ

## Περιεχόμενα



Η μονάδα ολοκληρωμένων κυκλωμάτων επεξεργασίας ψηφιακού σήματος (DSP) αποτελεί τη βάση πολλών ηλεκτρονικών συσκευών όπως κινητά τηλέφωνα, μονάδες ελέγχου CD και DVD, ηλεκτρονικά αεροπλάνων και αυτοκινήτων, τηλεοράσεων, ψηφιακών καμερών, και κάθε λογής ηλεκτρονικού εξοπλισμού. Η βασική λειτουργία του DSP είναι η ικανότητα να κάνει γρήγορους υπολογισμούς όπως FFTς (Fast Fourier Transforms) και να εφαρμόζει φίλτρα θορύβου και τεχνικές αναγνώρισης φωνής και εικόνων.

- 

### 1. Γενικές ιδιότητες τριγωνομετρικών σειρών:

Γνωρίζουμε από την θεωρία ότι οποιαδήποτε συνάρτηση $y = f(x)$ που είναι συνεχής και περιοδική συνάρτηση του $x$ με περίοδο $L$ $(f(x)=f(x+L))$ μπορεί να αντικατασταθεί με μία τριγωνομετρική (Fourier) σειρά (άπειρο άθροισμα συνημίτονων και ημιτόνων):

$$f(x) = \frac{1}{2} a_0 + \sum_{j=1}^{\infty} a_j \cos(2\pi jx/L) + b_j \sin(2\pi jx/L), \quad 0 \le x \le L$$

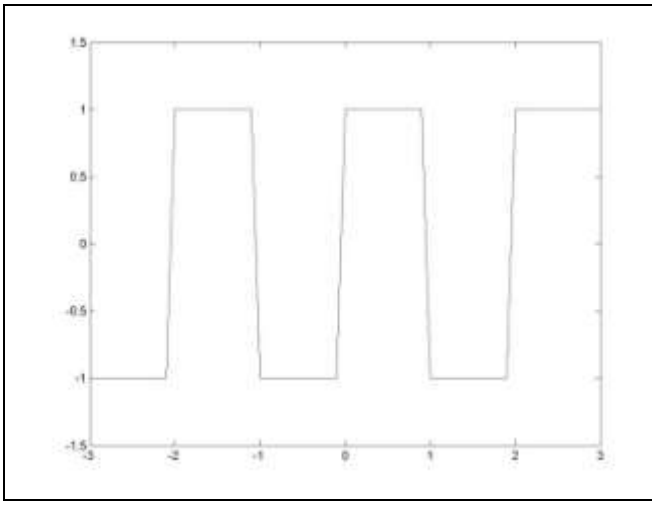
όπου $(a_j,b_j)$ είναι συντελεστές Fourier που υπολογίζονται από τη συνάρτηση $f(x)$ μέσω των ολοκληρωμάτων:

$$a_j = \frac{2}{L} \int_0^L f(x) \cos(2\pi jx/L)\, dx, \quad j = 0,1,2,\ldots,\infty$$

και

$$b_j = \frac{2}{L}\int_0^L f(x)\sin(2\pi jx/L)\,dx, \quad j = 1,2,\ldots,\infty$$

Παράδειγμα: f(x) = sign(x), -1 < x < 1 ; f(x + 2) = f(x), L = 2.



***Τριγωνομετρικές σειρές για ημητονικές συναρτήσεις:***

$$f(x) = \sum_{k=1}^{\infty} \frac{4}{\pi(2k-1)} \sin\pi(2k-1)x$$

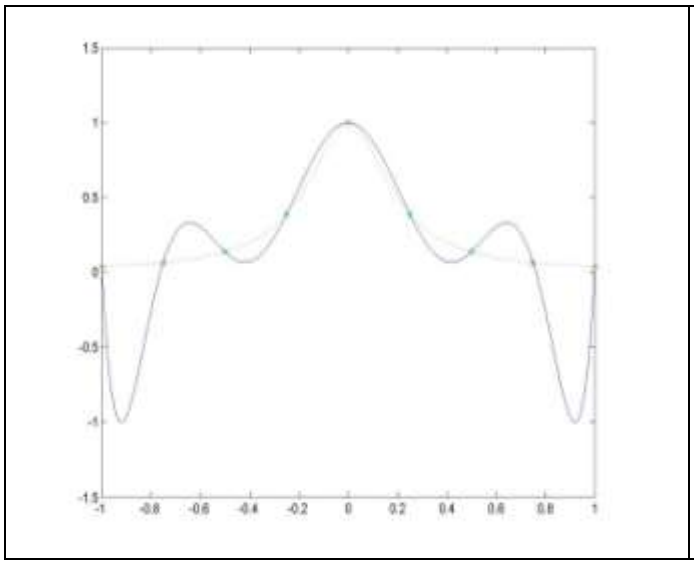
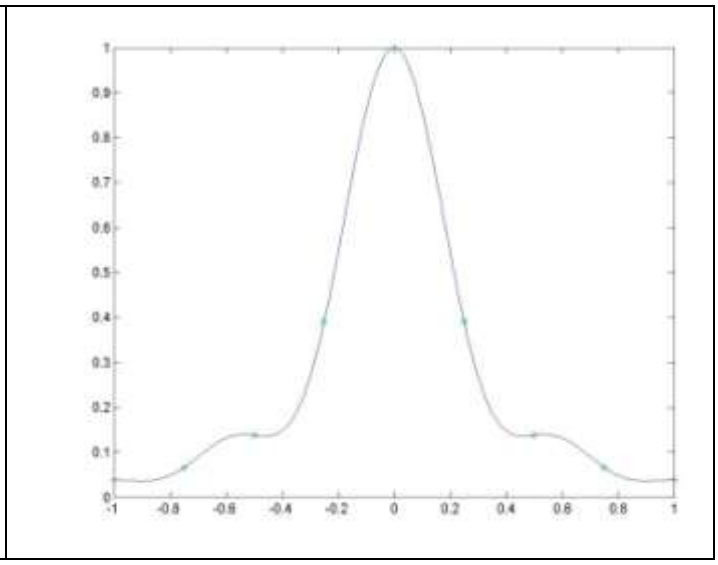
## 2. Τριγωνομετρική παρεμβολή:

**Πρόβλημα:** Δίνεται ένα σύνολο ***(n+1)*** σημείων:

$$(x_0, y_0),\ (x_1, y_1),\ \ldots,\ (x_{n-1}, y_{n-1}),\ (x_n, y_n),$$

έτσι ώστε οι τιμές για το ***x*** να αντιστοιχούν σε ισαπέχοντα σημεία στο διάστημα $x \in [0,L]$ και οι τιμές για το ***y*** να είναι περιοδικές με περίοδο ***L,*** δηλαδή $y_n = y_0$. Βρείτε ένα πεπερασμένο άθροισμα συνημίτονων και ημιτόνων $y = f(x)$ (αναφέρεται σαν τριγωνομετρικό πολυώνυμο) που να ενώνει όλα τα ***(n)*** πρώτα δεδομένα σημεία.

**Παράδειγμα:** Θεωρείστε μια παρεμβολή της συνάρτησης Runge $y = 1/(1 + 25\,x^2)$**.** Το πολυώνυμο παρεμβολής (αριστερά) δεν παράγει ακριβή προσέγγιση της Runge συνάρτησης εξαιτίας των αυξομειώσεων των τιμών του πολυωνύμου μεταξύ των δεδομένων σημείων. Αντιθέτως, η τριγωνομετρική παρεμβολή (δεξιά) αναπαράγει τη συνάρτηση Runge στο διάστημα $-1 < x < 1$ με σημαντικά «μικρότερο» σφάλμα.

**Οι τεχνικές για τον προσδιορισμό των τριγωνομετρικών πολυωνύμων παρεμβολής είναι:**

- Λύση των γραμμικών εξισώσεων παρεμβολής για την εύρεση των συντελεστών του τριγωνομετρικού πολυωνύμου (συντελεστές Fourier)
- Αθροιστικοί τύποι για εύρεση των συντελεστών Fourier
- Διακριτός μετασχηματισμός Fourier

## 3. Γραμμικά συστήματα για την εύρεση των συντελεστών Fourier:

Στο διάστημα ***[0,L]*** επιλέγουμε ένα σύνολο ισαπέχοντων σημείων***:***

$$x_i = iL/n, \quad i = 0,1,\ldots,n-1$$

Η τριγωνομετρική παρεμβολή ***y = f(x)*** , θα πρέπει να έχει ***n*** συντελεστές ***[$a_j$,$b_j$],*** οι οποίοι θα πρέπει να βρεθούν από τις ***n*** εξισώσεις: ***$y_i$ = f($x_i$).*** Εάν το ***n*** είναι άρτιο, δηλαδή ***n=2m***, η τριγωνομετρική παρεμβολή ***y = f(x)*** παίρνει τη μορφή:

$$f(x) = \frac{1}{2} a_0 + \sum_{j=1}^{m-1} \left( a_j \cos(2\pi jx/L) + b_j \sin(2\pi jx/L) \right) + a_m \cos(2\pi mx/L)$$

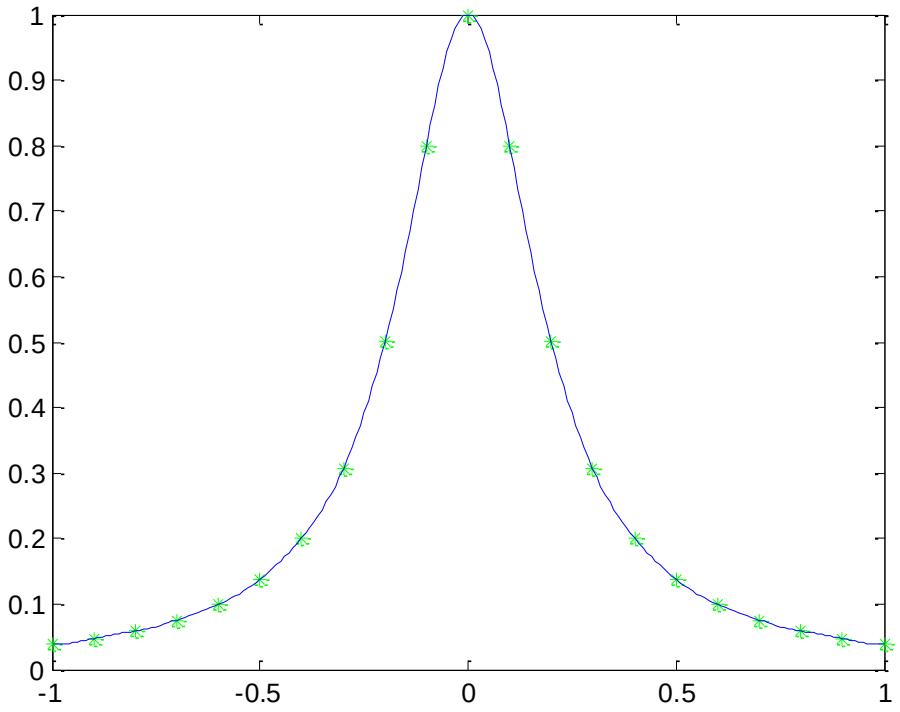
Η παραπάνω συνάρτηση αναφέρεται σαν το τριγωνομετρικό πολύωνυμο παρεμβολής ***n*** βαθμού.

Οι συντελεστές Fourier μπορούν να βρεθούν από την λύση του παρακάτω γραμμικού συστήματος:

$$y_i = \frac{1}{2} a_0 + \sum_{j=1}^{m-1} \left( a_j \cos(2\pi\, j\, x_i / L) + b_j \sin(2\pi\, j\, x_i / L) \right) + a_m \cos(2\pi\, m x_i / L), \quad j=0,n-1$$

Το παρακάτω MatLab πρόγραμμα ορίζει και λύνει το παραπάνω σύστημα για την συνάρτηση Runge στο διάστημα [-1,1] και παράγει την γραφική παράσταση των δεδομένων και του τριγωνομικού πολυωνύμου σε 10 σημεία:

```
x = -1 : 0.1 : 1; % equally-spaced partition of the interval: -1 < x <
1
   y = 1./(1 + 25*x.^2); % values of the Runge function
   n = length(x)-1; m = n/2; L = 2; n
   xx = [x(m+1:n),x(1:m)]'; yy = [y(m+1:n),y(1:m)]';
        % continuation of data to the positive interval for x in [0,L]
   A = 0.5*ones(n,1);  % building the coefficient matrix for linear
system
   for j = 1 : (m-1)
            A = [ A, cos(2*pi*j*xx/L), sin(2*pi*j*xx/L) ];
   end
A = [ A, cos(2*pi*m*xx/L) ];A
   c = A\yy; % computations of Fourier coefficients in a special order
   a = [ c(1); c(2:2:n) ]; b = [0; c(3:2:n)]; % coefficients are column
vectors
   a',b', xInt = -1: 0.01 : 1; % interpolation partition on the same
interval
   yInt = 0.5*a(1)*ones(1,length(xInt));
   for j = 1 : (m-1)
      yInt = yInt + a(1+j)*cos(2*pi*j*xInt/L) +
b(1+j)*sin(2*pi*j*xInt/L);
   end
   yInt = yInt + a(m+1)*cos(2*pi*m*xInt/L);

   plot(x,y,'*g',xInt,yInt)
```




## 4. Αθροιστικοί τύποι για τον προσδιορισμό των συντελεστών Fourier:

Αν θεωρήσουμε τα ισαπέχοντα σημεία $\left\{ x_i = \frac{i}{n} L \right\}_{i=0}^{n}$ του διαστήματος

$x \in [0,L]$ και υποθέσουμε ότι $n=2m$ τότε, οι συντελεστές Fourier $[a_j,b_j]$ μπορούν να υπολογιστούν από τους παρακάτω άμεσους τύπους άθροισης:

$$a_j = \frac{2}{n} \sum_{i=1}^{n} y_i \cos(2\pi j x_i / L), \quad j = 0,1,2,\ldots,m$$

και

$$b_j = \frac{2}{n} \sum_{i=1}^{n} y_i \sin(2\pi j x_i / L), \quad j = 1,2,\ldots,m-1$$

Αυτοί οι τύποι αντιπροσωπεύουν τον διακριτό μετασχηματισμό Fourier στο σύνολο σημείων δεδομένων $(x_i,y_i)$.

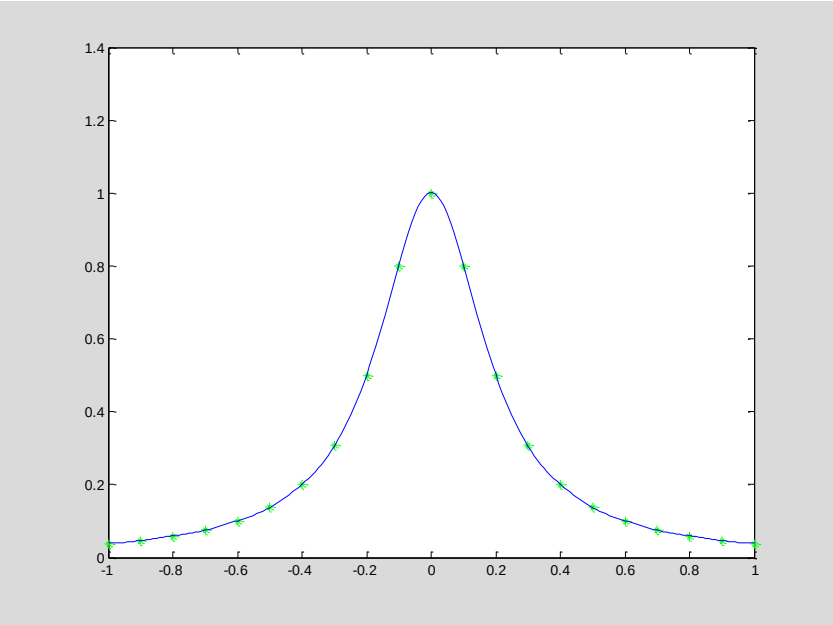
Το παρακάτω πρόγραμμα υπολογίζει τους συντελεστές της σειράς Fourier για ένα σύνολο σημείων της συνάρτησης Runge και παράγει την γραφική της παράσταση. Παρατηρούμε ότι η προσέγγιση αναπαράγει πιστά την συνάρτηση.

```
   P = A'*A; % summation formulas follows from the fact that A'*A is
diagonal
P(1:6,1:6)
% the discrete cosines and sines are orthogonal with respect to sums!
```

```
ans =
    5.0000    0.0000   -0.0000    0.0000   -0.0000    0.0000
    0.0000   10.0000   -0.0000    0.0000         0   -0.0000
   -0.0000   -0.0000   10.0000   -0.0000   -0.0000    0.0000
    0.0000    0.0000   -0.0000   10.0000    0.0000    0.0000
   -0.0000         0   -0.0000    0.0000   10.0000   -0.0000
    0.0000   -0.0000    0.0000    0.0000   -0.0000   10.0000
```

```
x = -1 : 0.1 : 1; y = 1./(1 + 25*x.^2);
n = length(x)-1; m = n/2; L = 2;
xx = [x(m+1:n),x(1:m)]'; yy = [y(m+1:n),y(1:m)]';
for j = 0 : m
a(j+1) = 2*yy'*cos(2*pi*j*xx/L)/n;
b(j+1) = 2*yy'*sin(2*pi*j*xx/L)/n;
% inner (dot) product is used for computations
% b(1) = b(m+1) = 0
end
a',b', xInt = -1: 0.01 : 1;
yInt = 0.5*a(1)*ones(1,length(xInt));
for j = 1 : (m-1)
      yInt = yInt + a(1+j)*cos(2*pi*j*xInt/L) +
b(1+j)*sin(2*pi*j*xInt/L);
end
yInt = yInt + a(m+1)*cos(2*pi*m*xInt/L);
plot(x,y,'*g',xInt,yInt,'b');
```

```
ans =    0.5492    0.3442    0.1756    0.0970    0.0499    0.0279
0.0140    0.0084    0.0041    0.0032    0.0020
ans =  1.0e-016 *
         0   -0.0833   -0.2220         0    0.1110         0    0.1110
-0.1110    0.0555   -0.0278    0.0471
```




## 5. Διακριτός μετασχηματισμός Fourier

Από την ταυτότητα του Euler $e^{i\theta} = \cos(\theta) + i\sin(\theta)$ μπορούμε να συμπεράνουμε εύκολα ότι το μερικό τριγωνομετρικό άθροισμα της σειράς Fourier μπορεί να παρασταθεί από το μιγαδικό άθροισμα:

$$f(x) = \frac{1}{n}\sum_{j=-m}^{m} c_j \, exp\left(2\pi i * \frac{jx}{L}\right), \quad i = \sqrt{-1}\ , \quad 0 \le x \le L \qquad (1)$$

όπου $c_j = n\,(a_j - ib_j)\,/\,2$ και $c_{-j} = n\,(a_j + ib_j)\,/\,2$. Οι μιγαδικοί συντελεστές Fourier $c_j$ υπολογίζονται με τον εξής τύπο :

$$c_j = \sum_{k=1}^{n} y_k \, exp\left(-2\pi i * \frac{jx_k}{L}\right), \qquad j = 0, \pm 1, \pm 2, \ldots, \pm m \qquad (2)$$

Για ένα σύνολο δεδομένων σημείων $(x_k, y_k)$ όπου $k = 0,1,\ldots,n-1$ και $n = 2m$, οι τύποι (2) συνήθως αναφέρονται ως διακριτός μετασχηματισμός Fourier (**fft**), ενώ οι τύποι (1) στα σημεία $(x_k, y_k)$ αναφέρονται ως αντίστροφος διακριτός μετασχηματισμός Fourier (**ifft**). Χρησιμοποιώντας την απλή παρατήρηση ότι

$$c_{-j} = exp\left(2\pi i * \frac{jx_k}{L}\right) = exp\left(2\pi i * \frac{jk}{n}\right) = exp\left(2\pi i * \frac{(j-n)k}{n}\right) = c_{n-j}$$

οι διακριτοί και αντίστροφα διακριτοί μετασχηματισμοί Fourier μπορούν να επαναπροσδιοριστούν μόνον ως προς τους θετικούς δείκτες:

$$c_j = \sum_{k=0}^{n-1} y_k \; exp(-2\pi i * \frac{jx_k}{L}), \quad j = 0,1,2,\ldots,n-1 \quad (1^*)$$

και

$$y_k = \frac{1}{n} \sum_{j=0}^{n-1} c_j \; exp(2\pi i * \frac{jx_k}{L}), \; k=0,1,\ldots,n-1. \quad (2^*)$$

Οι τύποι αυτοί ισχύουν μόνο για τα διακριτά σημεία δεδομένων $(x_k,y_k)$. Για να μπορέσουμε να παρεμβάλουμε την συνάρτηση $y = f(x)$ πέρα από τα σημεία δεδομένων $(x_k,y_k)$ θα πρέπει να χρησιμοποιήσουμε τον τύπο του τριγωνομετρικού πολυωνύμου παρεμβολής με συντελεστές:

$$a_j = \frac{2}{n} Re(c_j); \quad b_j = -\frac{2}{n} Im(c_j); \quad j = 0,1,\ldots,m,$$

όπου $m = n/2$.

## 6. Συναρτήσεις MatLab για fft και ifft

- **fft:** υπολογίζει το διακριτό μετασχηματισμό Fourier (1*), δηλαδή τους συντελεστές $c_j$ από τις τιμές $\{y_k\}$
- **ifft:** υπολογίζει τον αντίστροφο διακριτό μετασχηματισμό Fourier (2*), δηλαδή τις τιμές $y_k$ από τους συντελεστές $\{c_j\}$
-

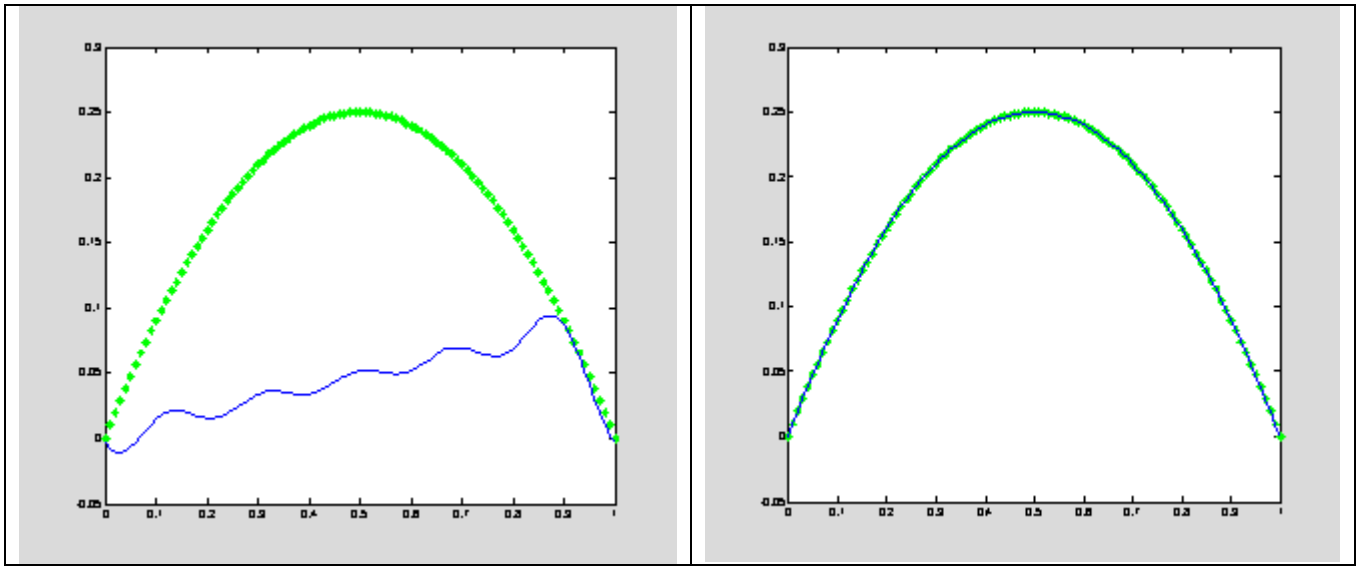
## 7. Προσέγγιση με Τριγωνομετρικά Πολυώνυμα

-
- Αν $m < n/2$, τότε το τριγωνομετρικό πολυώνυμο προσεγγίζει την $y = f(x)$ με την μέθοδο των ελαχίστων τετραγώνων (Κεφάλαιο 6), αφού τα δοθέντα σημεία $(x_k,y_k)$ είναι περισσότερα από τους συντελεστές Fourier $(a_j,b_j)$ που πρέπει να προσδιοριστούν. Οι συναρτήσεις
-
- • **fft(y,N), ifft(y,N):** υπολογίζουν το τριγωνομετρικό πολυώνυμο παρεμβολής αν $N = length(y)$, τριγωνομετρική προσέγγιση αν $N < length(y)$ και το τριγωνομετρικό πολυώνυμο παρεμβολής συμπληρώνοντας με μηδενικά στοιχεία το $y$ αν $N > length(y)$

**Παράδειγμα:** Εφαρμόζει την συνάρτηση fft για το προσδιορισμό των συντελεστών του τριγωνομετρικού πολυωνύμου παρεμβολής της συνάρτησης y=x(x-1) στο διάστημα [0,1] και παράγει την γραφική παράσταση των δεδομένων και τις τιμές του πολυωνύμου σε 100 σημεία.

```
x = 0 : 0.01 : 1; y = x.*(1-x);    % data points
n = length(x)-1; m = n/2; L = 1;
xx = x(1:n)'; yy = y(1:n)';
N = 10; mN = N/2; c = fft(yy,N);
a = 2*real(c(1:mN+1))/N; b = -2*imag(c(1:mN+1))/N;
aa = a', bb = b'
xInt = 0: 0.01 : 1; yInt = 0.5*a(1)*ones(1,length(xInt));
for j = 1 : (mN-1)
```

```
  yInt = yInt + a(1+j)*cos(2*pi*j*xInt/L) +
b(1+j)*sin(2*pi*j*xInt/L);
end
yInt = yInt + a(mN+1)*cos(2*pi*mN*xInt/L);
plot(x,y,'*g',xInt,yInt,'b');

aa =    0.0843   -0.0100   -0.0093   -0.0092   -0.0091   -0.0091
bb =         0   -0.0277   -0.0124   -0.0065   -0.0029         0

```



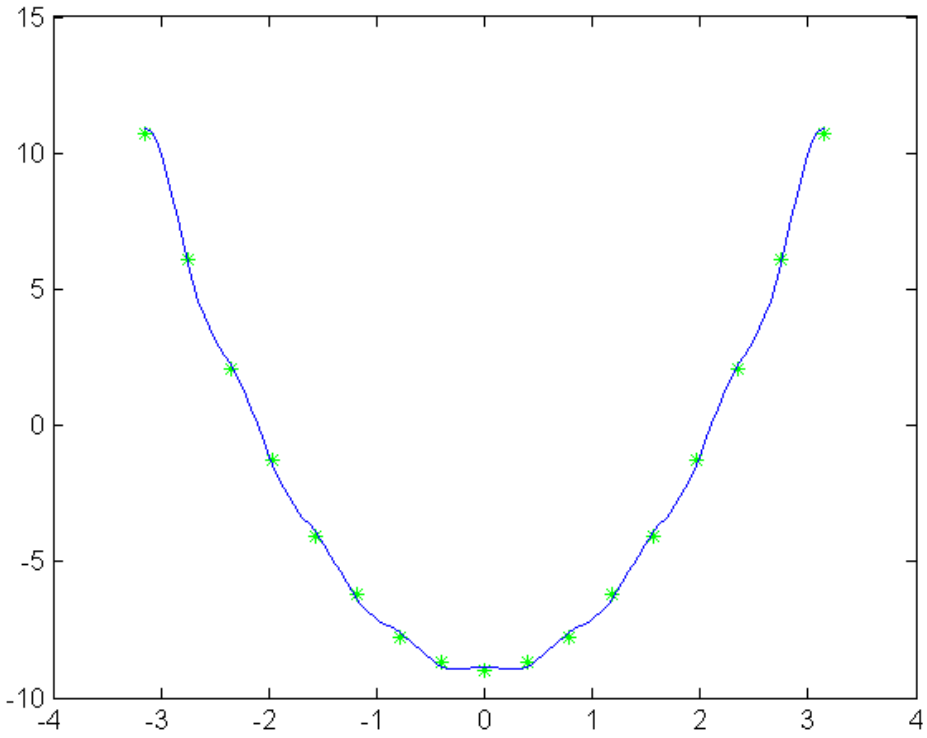
- Το αριστερό γράφημα δείχνει την τριγωνομετρική προσέγγιση της $y = x(x-1)$ με $N$ = ***10*** .
- Το δεξιό γράφημα δείχνει την τριγωνομετρική παρεμβολή της $y = x(x-1)$ με $N = n$ = ***100***.

**Παράδειγμα:** Να προσδιοριστεί το τριγωνομετρικό πολυώνυμο $2^{ου}$ βαθμού που παρεμβάλει την συνάρτηση $f(x) = 2x^2 - 9$ στο διάστημα $[-\pi, \pi]$ με τους αθροιστικούς τύπους και συνάρτηση fft

```
x=[-pi:pi/8:pi];y= 2*x.^2-9;
n = length(x)-1; m = n/2; L = 2*pi;
xx = [x(m+1:n),x(1:m)]'; yy = [y(m+1:n),y(1:m)], yy = yy';
for j = 0 : m
a(j+1) = 2*yy'*cos(2*pi*j*xx/L)/n;
b(j+1) = 2*yy'*sin(2*pi*j*xx/L)/n;
end
a =a, b =b      % Fourier coefficients of trigonometric
series
c = fft(yy); c = c'
aF = 2*real(c(1:m+1))/n; bF = -2*imag(c(1:m+1))/n;
a = aF, b = bF  % Fourier coefficients derived from FFT
```

```
yF = ifft(c)    % the same function y is recovered, i.e. yF
= ifft(fft(y)) = y
xInt = -pi: pi/64 : pi;
yInt = 0.5*a(1)*ones(1,length(xInt));
for j = 1 : (m-1)
    yInt = yInt + a(1+j)*cos(2*pi*j*xInt/L) +
b(1+j)*sin(2*pi*j*xInt/L);
end
yInt = yInt + a(m+1)*cos(2*pi*m*xInt/L);
plot(x,y,'*g',xInt,yInt,'b');

yy =
  Columns 1 through 5
   -9.0000   -8.6916   -7.7663   -6.2242   -4.0652
  Columns 6 through 10
   -1.2894    2.1033    6.1128   10.7392    6.1128
  Columns 11 through 15
    2.1033   -1.2894   -4.0652   -6.2242   -7.7663
  Column 16
   -8.6916
a =
  Columns 1 through 5
   -4.7377   -8.1036    2.1061   -0.9992    0.6169
  Columns 6 through 10
   -0.4461    0.3613   -0.3206    0.3084    0.0032
  Column 11
    0.0020
b =
  1.0e-014 *
  Columns 1 through 5
         0   -0.0111    0.0333   -0.0444    0.0658
  Columns 6 through 10
   -0.3220    0.0999   -0.1221    0.1315   -0.0011
  Column 11
         0
c =
  Columns 1 through 5
  -37.9018  -64.8288   16.8485   -7.9940    4.9348
  Columns 6 through 10
   -3.5690    2.8907   -2.5650    2.4674   -2.5650
  Columns 11 through 15
    2.8907   -3.5690    4.9348   -7.9940   16.8485
  Column 16
  -64.8288
a =
  Columns 1 through 5
   -4.7377   -8.1036    2.1061   -0.9992    0.6169
```

```
  Columns 6 through 9
   -0.4461    0.3613   -0.3206    0.3084
b =
     0     0     0     0     0     0     0     0     0
yF =
  Columns 1 through 5
   -9.0000   -8.6916   -7.7663   -6.2242   -4.0652
  Columns 6 through 10
   -1.2894    2.1033    6.1128   10.7392    6.1128
  Columns 11 through 15
    2.1033   -1.2894   -4.0652   -6.2242   -7.7663
  Column 16
   -8.6916
```




## 8. Προσδιορισμός του fft και ifft μέσω πολυωνυμικής παρεμβολής στις ρίζες της μονάδος

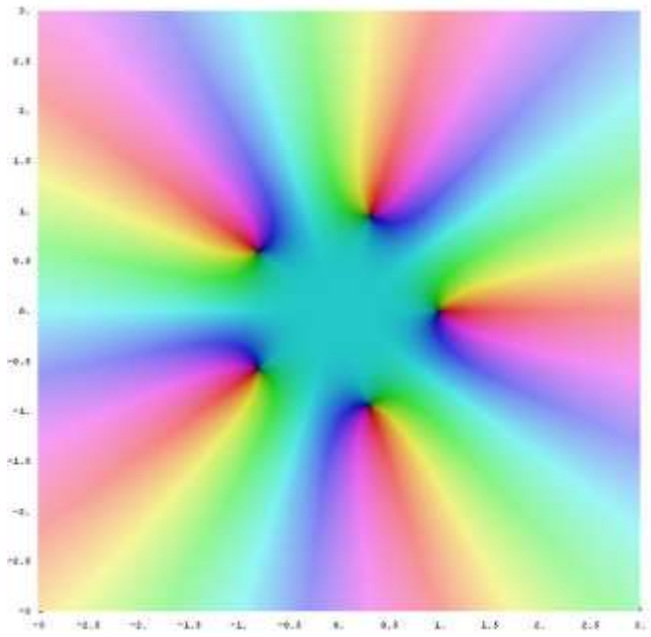
**Ισχυρισμός:** Ο διακριτός Fourier μετασχηματισμός (DFT) του διανύσματος $x = [x_0, \ldots, x_{n-1}]$ είναι το διάνυσμα $y = [y_0, \ldots, y_{n-1}]$ μπορεί να οριστεί από τον τύπο

$y_k = \frac{1}{n}\sum_{j=0}^{n-1} x_j \omega^{jk}$ όπου $\omega = \exp(-2\pi i / n)$ και ο αντίστροφος μετασχηματισμός Fourier από τον τύπο $x_k = \frac{1}{n}\sum_{j=0}^{n-1} y_j \omega^{-jk}$.

Για την επαλήθευση του παραπάνω ισχυρισμού παρατηρούμε ότι αν επιλέξουμε το διάνυσμα ***x*** να είναι τα ***n*** ισαπέχοντα σημεία στο διάστημα [*0,L*], δηλαδή $x_k = \frac{k}{n}L,\ k = 0,...,n-1$, τότε

$$\exp(-2\pi i \frac{jx_k}{L}) = \exp(-2\pi i \frac{kL/n}{L}) = \exp(-2\pi ik/n)$$

**Σημειώστε** ότι αν $\omega = \exp(-2\pi i / n)$ είναι η βασική ρίζα της μονάδος ($z^n = 1$)**,** τότε οι υπόλοιπες ρίζες της μονάδος είναι οι δυνάμεις της βασικής ρίζας, δηλαδή $\omega^k = \exp(-2\pi ik / n)$. Το παρακάτω γράφημα απεικονίζει με μαύρο χρώμα τις ρίζες της εξίσωσης $z^5=1$.



Από την μορφή των εξισώσεων (1*) , μπορούμε να συμπεράνουμε ότι ο μετασχηματισμός Fourier αντιστοιχεί στην εύρεση του πολυωνύμου παρεμβολής στις n ρίζες της μονάδος, δηλαδή στην λύση του παρακάτω συστήματος Vandermonde ως προς $\{c_i\}$,

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix}$$

$$y = V_n c, \quad (V_n)_{k,j} = \omega_n^{kj}$$

Παρόμοια, μπορούμε να συμπεράνουμε ότι οι εξισώσεις (2*), που ορίζουν το ifft, αντιστοιχούν στην λύση του αντίστροφου σύστηματος Vandermonde

$$c = V_n^{-1} y, \ (V_n^{-1})_{k,j} = (V_n^{-1})_{j,k} = \omega_n^{-kj} / n \text{ for } j,k = 0,1,.....,n-1.$$

**Παράδειγμα:**

Ο κώδικας που ακολουθεί βρίσκει τους Fourier συντελεστές για ένα πεπερασμένο σύνολο σημείων της συνάρτησης $\frac{1}{1+25x^2}$ στο διάστημα [-1,1].

α) υπολογίζοντας τα διακριτά αθροίσματα (1) και
β) εφαρμόζοντας τον μετασχηματισμό fft
Επιπλέον, πως ο αντίστροφος μετασχηματισμός Fourier ifft ανακτά τις τιμές της συνάρτησης από τους συντελεστές Fourier.

```
x = -1 : 0.5 : 1; y = 1./(1 + 25*x.^2); % data points
n = length(x)-1; m = n/2; L = 2;
xx = [x(m+1:n),x(1:m)]'; yy = [y(m+1:n),y(1:m)], yy = yy';
for j = 0 : m
  a(j+1) = 2*yy'*cos(2*pi*j*xx/L)/n;
  b(j+1) = 2*yy'*sin(2*pi*j*xx/L)/n;
end
a =a, b =b     % Fourier coefficients of trigonometric series
c = fft(yy); c = c'
aF = 2*real(c(1:m+1))/n; bF = -2*imag(c(1:m+1))/n;
aF = aF, bF = bF  % Fourier coefficients derived from FFT
yF = ifft(c)    % the same function y is recovered, i.e. yF =
ifft(fft(y)) = y
```

```
yy =   1.0000    0.1379    0.0385    0.1379
a =    0.6572    0.4808    0.3813
b =  1.0e-017 *
         0         0    0.4710
c =    1.3143    0.9615    0.7626    0.9615
aF =    0.6572    0.4808    0.3813
```

```
bF =      0      0      0
yF =    1.0000    0.1379    0.0385    0.1379
```

Η συνάρτηση ***y = f(x)*** μπορεί να υπολογιστεί στα σημεία που είναι διαφορετικά από τα $(x_k,y_k)$ από το τριγωνομετρικό άθροισμα:

$$f(x) = \frac{1}{2} a_0 + \sum_{j=1}^{m-1} \left( a_j \cos(2\pi jx/L) + b_j \sin(2\pi jx/L) \right) + a_m \cos(2\pi mx/L)$$

Η συνάρτηση αυτή αναφέρεται και σαν τριγωνομετρικό πολυώνυμο βαθμού *n*.

**Άσκηση**

Στη παράγραφο αυτή είδαμε ότι ο μετασχηματισμός Fourier μπορεί να παρασταθεί σαν το γινόμενο του Van der Monde πίνακα ως προς τις ρίζες της μονάδος. Για $N=4$, μετασχηματισμός Fourier είναι $y=W_4 x$ , όπου $\omega = e^{-i2\pi/4}$ είναι οι ρίζες της εξίσωσης $z^3=1$ και $W_4$ ο παρακάτω πίνακας

$$W_4 = \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^4 & \omega^8 & \omega^{12} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\imath & -1 & \imath \\ 1 & -1 & 1 & -1 \\ 1 & \imath & -1 & -\imath \end{pmatrix}$$

- Αποδείξτε ότι $W_4$ είναι συμμετρικός αλλά όχι Ερμιτιανός και Ορθομοναδιαίος.
- Ο αντίστροφος μετασχηματισμός Fourier είναι $x = \frac{1}{4} W_4^H y (= \frac{1}{4} W_4^H W_4 x = \frac{4I}{4} x)$

$$W_4^H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \imath & -1 & -\imath \\ 1 & -1 & 1 & -1 \\ 1 & -\imath & -1 & \imath \end{pmatrix}$$

## 9. Ένας αποδοτικός τρόπος για τον υπολογισμό των τριγωνομετρικών πολυωνύμων

Ο υπολογισμός του τριγωνομετρικού πολυωνύμου ***n=2m*** βαθμού

$$f(x) = \frac{1}{2} a_0 + \sum_{j=1}^{m-1} \left( a_j \cos(2\pi jx/L) + b_j \sin(2\pi jx/L) \right) + a_m \cos(2\pi mx/L)$$

μπορεί να γίνει χωρίς τον υπολογισμό των **sin** και **cos** στο τύπο αλλά χρησιμοποιώντας «έξυπνα» τις συναρτήσεις fft και ifft.

Για το σκοπό αυτό θεωρούμε τις τιμές της $y_k = f(x_k)$ στα σημεία παρεμβολής $\left\{ x_k = \frac{k}{n} L \right\}_{k=0}^{n-1}$ και ***p>n*** (***p*** άρτιος) τα σημεία υπολογισμού $\left\{ s_k = \frac{k}{p} L \right\}_{k=0}^{p-1}$ της συνάρτησης

$$P(x) = \frac{1}{2} a_0 + \sum_{j=1}^{p/2-1} \left( a_j \cos(2\pi jx/L) + b_j \sin(2\pi jx/L) \right) + a_{p/2} \cos(2\pi mx/L)$$

όπου $a_j = b_j = 0$ για $j = \frac{n}{2}+1, \ldots, \frac{p}{2}$. Οι συντελεστές του $P(x)$ $a_j$ και $b_j$ για $j = 0, \ldots, n/2$ υπολογίζονται εφαρμόζοντας τον fft στο διάνυσμα δοθέντων σημείων *y*. Για τον υπολογισμό του $P(x)$ στα σημεία ***s*** εφαρμόζουμε τον αντίστροφο ifft.

Οι παρακάτω κώδικες υλοποιούν την παραπάνω παρατήρη:

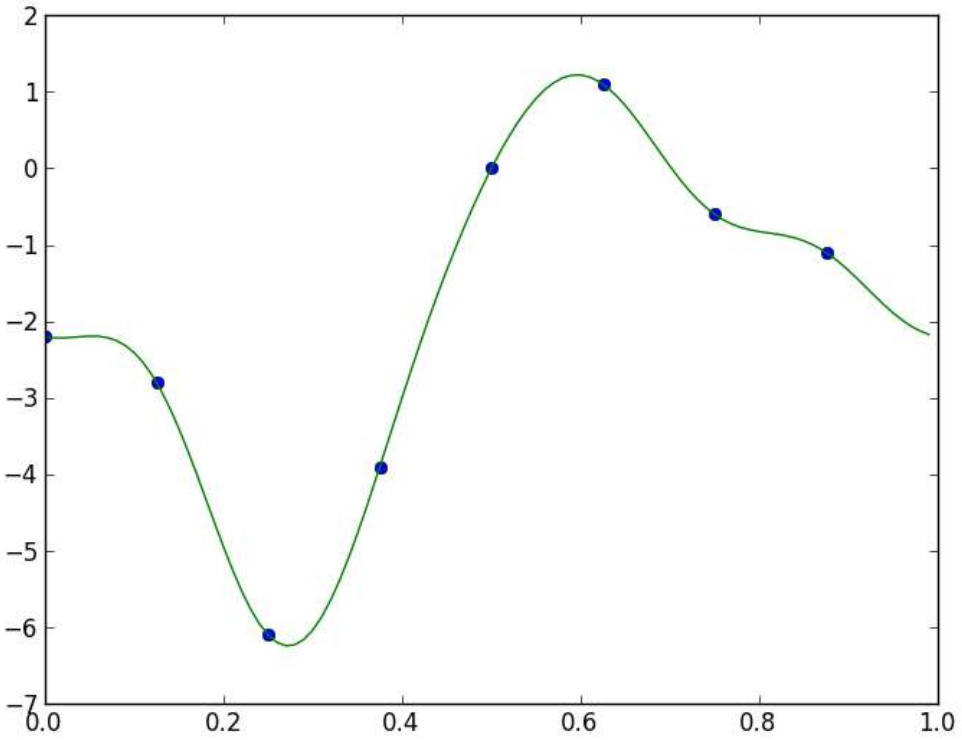
**MatLab κώδικας**

```
%Interpolate n data points on [a,b] with trig function P(x)
% and plot interpolant at p (>=n) evenly spaced points.
%Input: interval [a,b], data points y, even number of data
% points x n in [a,b], even number xp p>=n in [a,b]
%Output: data points of interpolant yp
function yp=dftinterp(inter,y,n,p)
a=inter(1);b=inter(2);x=a+(b-a)*(0:n-1)/n; xp=a+(b-a)*(0:p-
1)/p;
c=fft(y); % apply DFT
cp=zeros(p,1); % yp will hold coefficients for ifft
cp(1:n/2+1)=c(1:n/2+1); % move n frequencies from n to p
cp(p-n/2+2:p)=c(n/2+2:n); % same for upper tier
yp=real(ifft(cp))*(p/n); % invert fft to recover data
plot(t,x,'o',xp,yp) % plot data points and interpolant
```

**Python κώδικας**

```
from numpy import *
from scipy import fft, ifft
from pylab import plot, show
def dftinterp( inter, y, n, p ):
x = linspace( inter[0], inter[1], n, endpoint=False ) # n
evenly-spaced interpolation points
xp = linspace( inter[0], inter[1], p, endpoint=False ) # p
evenly-spaced time points
c = fft(y) # apply DFT
cp = zeros(p,dtype=complex) # ψp will hold coefficients for
ifft
cp[:n/2+1] = c[:n/2+1] # move n frequencies from n to p
cp[p-n/2+1:] = c[n/2+1:] # same for upper tier
yp = real( ifft(cp) )*float(p)/n # invert fft to recover
data
```

```
plot( x, y, 'o', xp, yp ) # plot data points and
interpolant
show()
dftinterp( [0,1], [-2.2, -2.8, -6.1, -3.9, 0.0, 1.1, -0.6,
-1.1], 8, 100 )
```

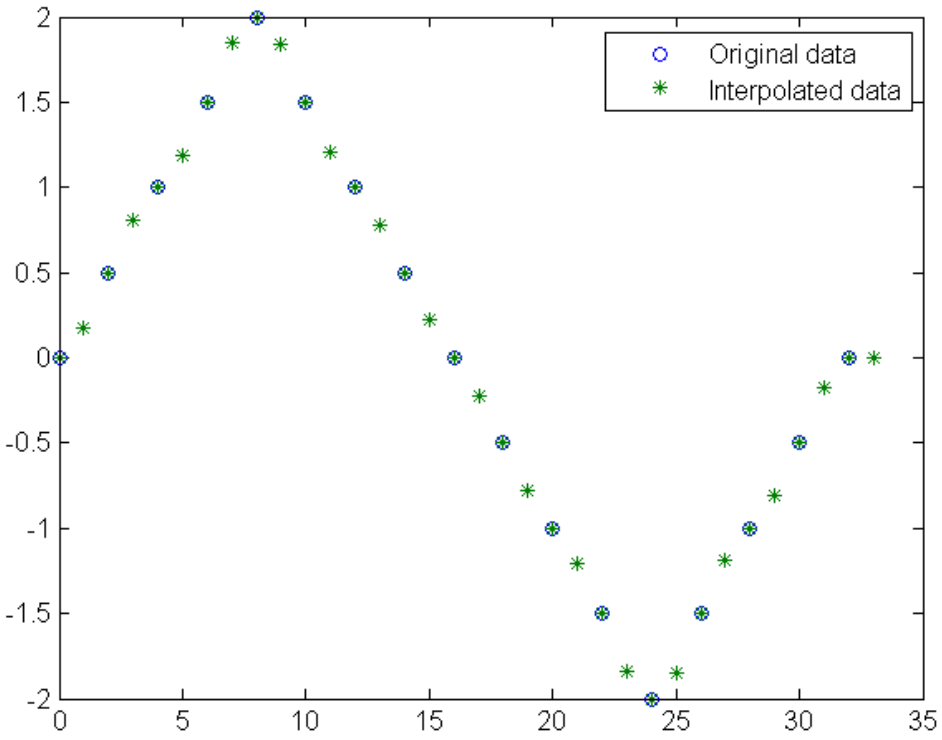



**Σημειώστε** ότι ο παραπάνω αποδοτικός υπολογισμός του τριγωνομετρικού πολυωνύμου υλοποιείται από την MatLab συνάρτηση Y=`interpft(X,N)` όπου Y διάνυσμα μήκους N τιμών περιοδικής συνάρτησης που λαμβάνεται εφαρμόζοντας παρεμβολή στο μετασχηματισμό Fourier της περιοδικής συνάρτησης X . Υποθέστε την περιοδική συνάρτηση x(t) με περίοδο L, και X(i)=x(T(i)) όπου {T(i)=(i-1)*L/M, i=1:M} είναι σύνολο ισαπέχοντων σημείων. Τότε y(t) είναι μια άλλη περιδιοδική συνάρτηση με την ίδια περίοδο L και Y(j)=y(T(j)) όπου T(j) = (j-1)*L/N, j = 1:N, N = length(Y). Αν N είναι ακέραιο πολλαπλάσιο του M, τότε Y(1:N/M:N) = X

**Παράδειγμα:** Τριγωνομετρική Παρεμβολή με την MatLab συνάρτηση interpft

Υποθέστε ότι τιμές μια συνάρτησης σε ισαπέχοντα σημεία οποιαδήποτε διαστήματος είναι `y = [0 .5 1 1.5 2 1.5 1 .5 0 -.5 -1 -1.5 -2 -1.5 -1 -.5 0];` Να υπολογιστεί το τριγωνομετρικό πολυώνυμο των παραπάνω τιμών με την συνάρτηση `interpft` (χρήση fft) και να υπολογιστεί σε ένα σύνολο τιμών μεγαλύτερο των τιμών της συνάρτησης. Σημειώστε ότι στην τριγωνομετρική παρεμβολή οι x-συντεταγμένες των σημείων μπορεί να οριστούν σε οποιαδήποτε διάστημα. (Γιατί;)

```
y = [0 .5 1 1.5 2 1.5 1 .5 0 -.5 -1 -1.5 -2 -1.5 -1 -.5 0];
N = length(y);
L = 2;% period of function
M = N*L;% number of evaluation points of trigonometric
polynomial
x = 0:L:L*N-1;% selectet x-coordinates of given equidistant
data
xi = 0:M-1;% selected x-coordinates of equidistant
evaluation pts
yi = interpft(y,M);%finds the interpolating trigo-poly and
evaluates at M pts
plot(x,y,'o',xi,yi,'*')
legend('Original data','Interpolated data')
```

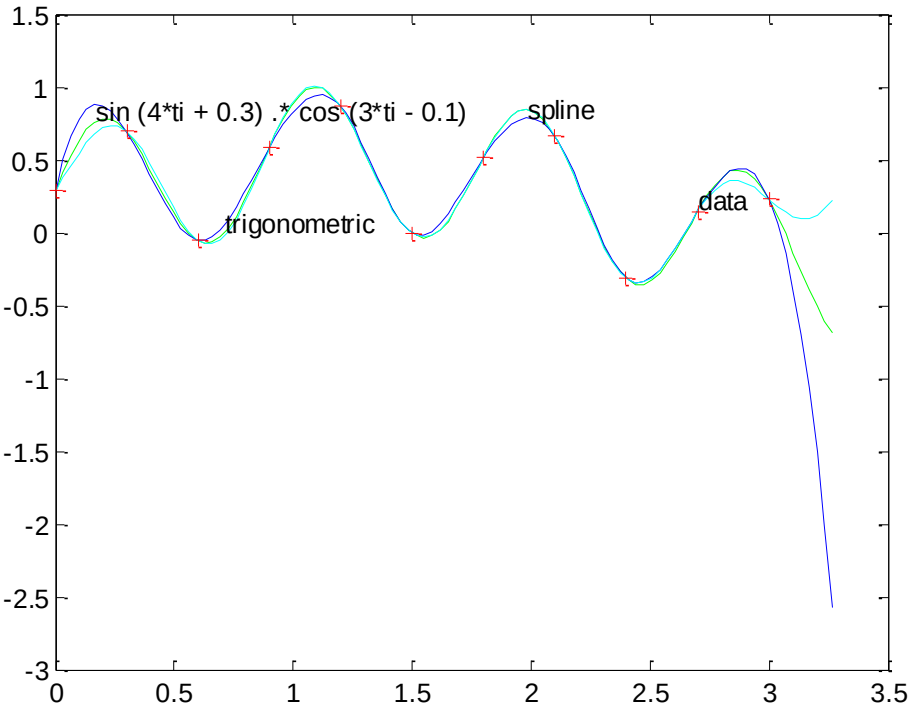



**Παράδειγμα 2**

Ο παρακάτω κώδικας συγκρίνει την μέθοδο παρεμβολής με κυβικές splines και τριγωνομετρικά πολυώνυμα για την συνάρτηση `y = sin (4*t + 0.3) .* cos (3*t - 0.1)`

```
t = 0 : 0.3 : pi; dt = t(2)-t(1);
n = length (t); k = 100;
ti = t(1) + [0 : k-1]*dt*n/k;
y = sin (4*t + 0.3) .* cos (3*t - 0.1);
yp = sin (4*ti + 0.3) .* cos (3*ti - 0.1);
```

```
plot (ti, yp, 'g',ti, interp1 (t, y, ti, 'spline'),'b', ...
    ti, interpft (y, k), 'c', t, y, '+r');
gtext('sin (4*ti + 0.3) .* cos (3*ti - 0.1)')
gtext('spline')
gtext('trigonometric')
gtext('data')
```




## Ασκήσεις

a. Αποδείξτε ότι οι ρίζες της $x^4 = 1$ είναι $e^{ik\pi/4}$ για κ=0,1,2,3 όπου $i = \sqrt{-1}$ η μιγαδική μονάδα.

b. Αποδείξτε τις παρακάτω σχέσεις για τις n-οστες ρίζες της μονάδος

$\omega^n = 1$, $e^{2\pi ik/n}$ για $k = 0, 1, ..., n-1$, όπου $e^{iu} = \cos u + i \sin u$ και $\iota = \sqrt{-1}$,

ι) Αν $\omega_n = e^{2\pi i/n}$, η κύρια n-th ρίζα της μονάδος τότε οι διαφορετικές ρίζες της μονάδος δίδονται από τις δυνάμεις

$\omega_n^0, \omega_n^1, \omega_n^2, ..., \omega_n^{n-1}$

(σημειώστε ότι ο πάνω δείκτης στο συμβολισμό συμβολίζει τον εκθέτη της n-οστής ρίζας)

Να αποδειχθούν οι παρακάτω σχέσεις

$\omega_n^n = ?$, $\omega_n^0 = 1$, $\omega_n^j \omega_n^k = \omega_n^{(j+k) \bmod n}$, $\omega_n^{-1} = \omega_n^{n-1}$

ιι) Να αποδειχθεί για κάθε μη αρνητικούς ακεραίους ότι

$\omega_{dn}^{dk} = \omega_n^k$

ιν) Να αποδειχθεί για κάθε άρτιο θετικό ακέραιο ισχύει

$$\omega_n^{n/2} = \omega_2 = -1$$

v) Αποδείξτε τους παρακάτω ισχυρισμούς

c) Αν n: άρτιος θετικός τότε

Τα τετράγωνα των n-th μιγαδικών ριζών της μονάδος είναι n/2 μιγαδικές (n/2)-th ρίζες της μονάδος.

$(\omega_n^k)^2 = \omega_n^{2k} = \omega_{n/2,}^k$ όπου $k \in Z^+ \cup \{0\}$ και $Z^+$ συμβολίζει το σύνολο των θετικών ακεραίων

$(\omega_n^{k+n/2})^2 = \omega_n^{2k+n} = \omega_n^{2k} = \omega_{n/2}^k$

$\Rightarrow \omega_n^k$ και $\omega_n^{k+n/2}$ έχουν το ίδιο τετράγωνο

d) Αποδείξτε ότι το σύστημα Vandermode για την εύρεση του πολυωνύμου παρεμβολής στις ρίζες της μονάδος είναι

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

$$y = V_n a, \quad (V_n)_{k,j} = \omega_n^{kj}$$

$$a = V_n^{-1} y$$

e) Να αποδειχθεί ότι τα στοιχεία του αντίστροφου $V_n^{-1}$ είναι

$$j,k = 0,1,.....,n-1, \quad (V_n^{-1})_{j,k} = \omega_n^{-kj} / n$$

f) Υπολογίστε το πολυώνυμο παρεμβολής $A(x) = \sum_{j=0}^{n-1} a_j x^j$ στις ρίζες $\omega_n^0, \omega_n^1, ....., \omega_n^{n-1}$,

Υποθέστε ότι n είναι δύναμης του 2

'Εστω $a =< a_0, a_1, ...., a_{n-1} >$, και $y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}$

$y =< y_0, y_1, ...., y_{n-1} >$ είναι ο διακριτός Fourier μετασχηματισμός (DFT) του διανύσματος των συντελεστών

$a=<a_0, a_1, ...., a_{n-1} >$,

$y = \mathrm{DFT}_n(a)$

g) Να εφαρμοσθεί ο FFT αλγόριθμος για n=4. Να δειχθούν όλα τα βήματα του αλγορίθμου.

## 10. Αλγόριθμος Fast Fourier Transform (FFT)

Ας θυμηθούμε ότι DFT ορίζεται από το τύπο $y_l = \sum_{k=0}^{n-1} x_k \omega^{lk}$ για $l = 0,1,...,n-1$ όπου

$\omega = e^{-j2\pi/n}$ και $j = \sqrt{-1}$ $c_j = \sum_{k=0}^{N-1} x_k \omega^{jk}$. Αποδείξαμε ότι οι παραπάνω εξισώσεις μπορούν να γραφτούν σε μορφή πίνακα $W_n x = y.$ όπου $W_n$ είναι ο Vandemode πίνακας στις ρίζες της εξίσωσης
$z^n = 1$ και ο fft στον πολλαπλασισμό πίνακα επί διάνυσμα. Γενικά, πολλαπλασιασμός πίνακα $n \times n$ επί διάνυσμα $n \times 1$ έχει πολυπλοκότητα $O(n^2)$. Στην συνέχεια, θα αναπτύξουμε μια υλοποίηση του αλγορίθμου DFT, δηλαδή τον υπολογισμό του γινομένου $W_n x$ σε $O(n \ln n)$ πράξεις. Ο αλγόριθμος αυτός καλείται Fast Fourier Transform (FFT).
Ο γρήγορος υπολογισμός του DFT βασίζεται στην παρακάτω διαίρει και βασίλευε στρατηγική, δηλαδή κάνοντας ξεχωριστά υπολογισμούς με άρτιους και περιττούς όρους του πολυωνύμου παρεμβολής:

$$A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + ..... + a_{N-2} x^{n/2-1}$$
$$A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + ..... + a_{N-1} x^{n/2-1}$$
$$A(x) = A^{[0]}(x^2) + x A^{[1]}(x^2)$$
$$\text{Έστω} \begin{cases} y_k^{[0]} = A^{[0]}(\omega_{n/2}^k) \\ y_k^{[1]} = A^{[1]}(\omega_{n/2}^k) \end{cases}$$

$$\begin{aligned} y_k = A(\omega_n^k) &= A^{[0]}(\omega_n^{2k}) + \omega_n^{2k} A^{[1]}(\omega_n^{2k+n}) \\ &= A^{[0]}(\omega_{n/2}^k) - \omega_n^k A^{[1]}(\omega_{n/2}^k) \\ &= y_k^{[0]} + \omega_n^k y_k^{[1]} \end{aligned}$$

$$\begin{aligned} y_{k+n/2} = A(\omega_n^{k+n/2}) &= A^{[0]}(\omega_n^{2k+n}) + \omega_n^{k+n/2} A^{[1]}(\omega_n^{2k}) \\ &= A^{[0]}(\omega_{n/2}^k) + \omega_n^{k+n/2} A^{[1]}(\omega_{n/2}^k) \\ &= y_k^{[0]} + \omega_n^{k+n/2} y_k^{[1]} = y_k^{[0]} - \omega_n^k y_k^{[1]} \end{aligned}$$

**Οι παραπάνω υπολογισμοί οδηγούν στον παρακάτω αναδρομικό (recursive) αλγόριθμο για FFT**

```
Recursive-FFT(a)
{  n=length[a];    /* n: power of 2  */
   if n=1 then return a;
   ω_n = e^{2πi/n};
   ω=1
   a^[0] = (a_0, a_2, ....., a_{n-2});
   a^[1] = (a_1, a_3, ....., a_{n-1});
   y^[0] = Recursive-FFT(a^[0]);
   y^[1] = Recursive-FFT(a^[1]);
   for  k=0  to  (n/2 - 1)  do
```

Μια υλοποίηση του αλγορίθμου στην MatLab δίδεται από τον παρακάτω κώδικα. Σημειώστε ότι αν ***n*** είναι δύναμη του 2 χρησιμοποιεί τον αναδρομικό αλγόριθμο διαφορετικά κάνει το πολλαπλασιασμό του πίνακα παρεμβολής στις ρίζες της μονάδος με το input διάνυσμα.

```
function y = fftx(x)
%FFTX Fast Finite Fourier Transform.
x = x(:);
n = length(x);
omega = exp(-2*pi*i/n);

if rem(n,2) == 0

% Recursive divide and conquer.
k = (0:n/2-1)';
w = omega.^k;
u = fftx(x(1:2:n-1));
v = w.*fftx(x(2:2:n));
y = [u+v; u-v];

else

% The Fourier matrix.
j = 0:n-1;
k = j';
F = omega.^(k*j);
y = F*x;end
```

MatLab χρησιμοποιεί τον FFTW αλγόριθμο που θεωρείται η ποιο ταχύτερη υλοποίηση του DFT (www.fftw.org)

**Πολυπλοκότητα του αναδρομικού αλγορίθμου:** $O(n\log n)$

Αν υποθέσουμε ότι ο χρόνος εκτέλεσης του αλγορίθμου είναι $T(n)$

--- Το βήμα 1 απαιτεί $2T(n/2)$ ,

--- Το βήμα 2 απαιτεί $O(n)$ .

--- $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = nlogn$

**Απόδειξη**

Έστω $n = 2^k$

$T(n) = n + 2T(n/2)$

$= n + 2(n/2 + 2T(n/4))$
$= n + n + 4T(n/4)$
$= n + n + 4(n/4 + 2T(n/8))$
$= n + n + n + 8T(n/8)$
…
$= in + 2iT(n/2i)$
$= kn + 2kT(1)$
$= n\log_2 n + nT(1) = O(n\log_2 n)$

## 11. Αναφορές


1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
3. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
4. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
5. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
6. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.

# ΚΕΦΑΛΑΙΟ 5: ΠΡΟΣΕΓΓΙΣΗ ΣΥΝΑΡΤΗΣΕΩΝ ΚΑΙ ΔΕΔΟΜΕΝΩΝ ΜΕ ΤΗΝ ΜΕΘΟΔΟ ΤΩΝ ΕΛΑΧΙΣΤΩΝ ΤΕΤΡΑΓΩΝΩΝ

## Περιεχόμενα



## 1. Πολυωνυμική προσέγγιση

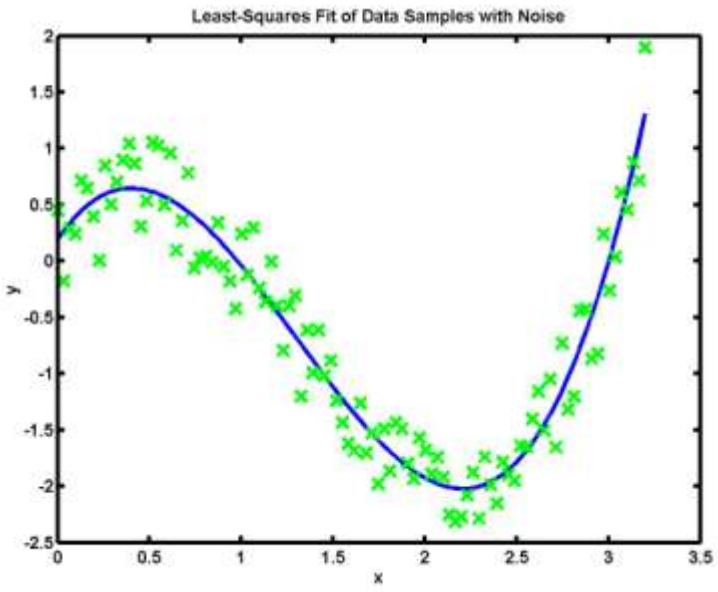
**Πρόβλημα:** Δεδομένου ενός συνόλου ***(n+1)*** σημείων:

$$(x_1,y_1),(x_2,y_2), \ldots, (x_m,y_n), (x_{n+1},y_{n+1})$$

Βρείτε ένα πολυώνυμο βαθμού $m < n$, $y = \Phi_m(x)$, που «ταιριάζει – προσεγγίζει – προσαρμόζει» (fit) καλύτερα το σύνολο δεδομένων σημείων ***(n+1).***

**Παρατήρηση:** Εάν το σύνολο δεδομένων έχει επηρεαστεί από μη ακριβείς μετρήσεις, εξαιτίας θορύβου ή αν οι τιμές των δεδομένων στρογγυλοποιηθούν, η πολυωνυμική παρεμβολή παράγει μια πολύ δύσκαμπτη καμπύλη με απότομες γωνίες και μεγάλο λάθος παρεμβολής. Στην περίπτωση αυτή, αντί της πολυωνυμικής παρεμβολής χρησιμοποιείται μια άλλη τεχνική που βασίζεται στην **αριθμητική προσέγγιση** των τιμών των δεδομένων σημείων.

Η γραφική παράσταση που ακολουθεί μας δείχνει την αριθμητική προσέγγιση περισσοτέρων από εκατό δεδομένα από ένα πολυώνυμο 3ου βαθμού:




Συνήθως, η πολυωνυμική προσέγγιση είναι ικανοποιητική όταν οι τιμές των δεδομένων προσαρμόζονται από ένα πολυώνυμο μικρού βαθμού. Σε πολλές περιπτώσεις, οι απλές (πολυωνυμικές) καμπύλες προβλέπονται από λύσεις θεωρητικών μοντέλων (π.χ. διαφορικές εξισώσεις). Οι μέθοδοι αριθμητικών προσεγγίσεων επιτρέπουν στους ερευνητές να συγκρίνουν θεωρητικά μοντέλα με δείγματα πραγματικών δεδομένων.

**Μέθοδοι:**

- Ελαχιστοποίηση του τετραγώνου του ολικού σφάλματος
- Υπερ-προσδιορισμένο γραμμικό σύστημα
- MATLAB **"polyfit"** συναρτήσεις

## 2. Ελαχιστοποίηση του τετραγώνου του ολικού σφάλματος

Ας υποθέσουμε ότι $y = \Phi_m(x)$ είναι ένα πολυώνυμο $m$ βαθμού, το οποίο προσαρμόζεται στα δεδομένα σημεία ελαχιστοποιώντας το τετράγωνο του ολικού σφάλματος μεταξύ των δεδομένων σημείων και των τιμών του:

$$\boxed{(min)\ E = \sum_{j=1}^{n+1} \left( y_j - \Phi_m(x_j) \right)^2}$$

Στην περίπτωση της γραμμικής παλινδρόμησης, δηλαδή όταν η συνάρτηση Φ είναι πολυώνυμο βαθμού 1, $y = \Phi_1(x) = c_1 x + c_2$

το ολικό σφάλμα είναι: $E = \sum_{j=1}^{n+1} (y_j - c_1 x_j - c_2)^2$

Οι συνθήκες ελαχιστοποίησης του ολικού σφάλματος δίδονται από το σύστημα:

$$\frac{\partial E}{\partial c_1} = -2 \sum_{j=1}^{n+1} x_j (y_j - c_1 x_j - c_2) = 0$$

$$\frac{\partial E}{\partial c_2} = -2 \sum_{j=1}^{n+1} (y_j - c_1 x_j - c_2) = 0$$

του οποίου η λύση προσδιορίζει τους συντελεστές του γραμμικού πολυωνύμου.

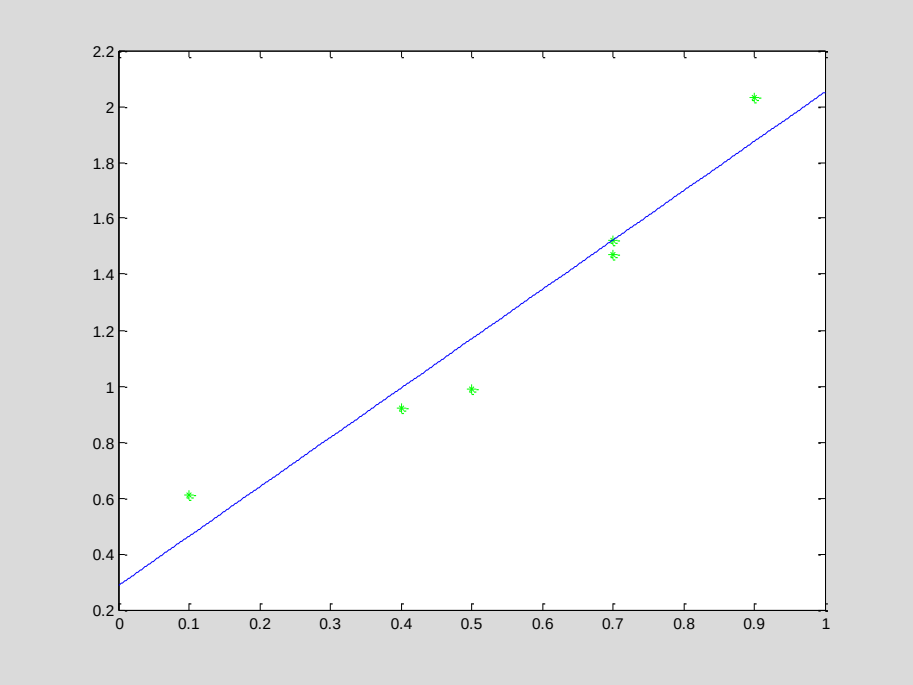
Η μέθοδος ελαχίστων τετραγώνων (ΕΤ) χρησιμοποιείται για να προσδιορίζει τις παραμέτρους μιας δεδομένης οικογένειας καμπυλών έτσι ώστε να ταιριάζουν καλύτερα σε ένα σύνολο δεδομένων. Η χρήση της δικαιολογείται για δυο τουλάχιστον λόγους:

a) Υπάρχουν φορές που έχουμε ένα σύνολο δεδομένων το οποίο είναι κατάλληλο για παρεμβολή, αλλά έχουμε αποφασίσει ότι δε χρειαζόμαστε τόση ακρίβεια όση μας προσφέρει η παρεμβολή. Δεδομένων των επιτρεπτών ορίων αβεβαιότητας στους υπολογισμούς μας, μπορεί να είμαστε ικανοποιημένοι με μια απλή συνάρτηση (πιθανώς πολυωνυμική) που περνάει κοντά από τα σημεία των δεδομένων μας αλλά όχι ακριβώς πάνω από όλα αυτά.
b) Σε πολλές πειραματικές διαδικασίες, τα σημεία δεδομένων μας περιέχουν ένα σφάλμα μέτρησης και δεν μπορούν να χρησιμοποιηθούν άμεσα για να γίνουν προβλέψεις. Στην περίπτωση αυτή, βάσει κάποιων θεωρήσεων, προσδιορίζουμε τη γενική μορφή μιας συνάρτησης που θα έπρεπε να ταιριάζει στη συμπεριφορά των δεδομένων. Μετά, εφαρμόζουμε τη μέθοδο των ελαχίστων τετραγώνων για να επιλέξουμε τις τιμές των απροσδιόριστων παραμέτρων της επιλεγμένης συνάρτησης.

Το παρακάτω MatLab πρόγραμμα υλοποιεί την παραπάνω μέθοδο για ένα σύνολο σημείων.

```
x = [ 0.1,0.4,0.5,0.7,0.7,0.9 ]
y = [ 0.61,0.92,0.99,1.52,1.47,2.03]
a11 = sum(x.^2); a12 = sum(x); a21 = sum(x); a22 =
sum(ones(1,length(x)));
A = [ a11,a12; a21,a22]; % the coefficient matrix of the minimization
problem
b1 = sum(x.*y); b2 = sum(y);
b = [ b1; b2 ]; % right-hand-side of the minimization problem
c = A \ b  % solution of the minimization problem
xApr = 0 : 0.001 : 1; yApr = c(1)*xApr + c(2);
plot(x,y,'*g',xApr,yApr,'b');

x =    0.1000    0.4000    0.5000    0.7000    0.7000    0.9000
y =    0.6100    0.9200    0.9900    1.5200    1.4700    2.0300
```

```
c =    1.7646    0.2862
```




## 3. Υπέρ-προσδιοριμένα γραμμικά συστήματα

Ας υποθέσουμε ότι το πολυώνυμο προσέγγισης $y = \Phi_m(x)$ περνάει απ' όλα τα δεδομένα σημεία: $\Phi_m(x_j) = y_j$. Στην περίπτωση αυτή ($m<n$), οι συνθήκες παρεμβολής ορίζουν ένα υπερ-προσδιορισμένο γραμμικό σύστημα $A\,c = b$ ( ο αριθμός των εξισώσεων $(n+1)$ υπερβαίνει τον αριθμό των αγνώστων $(m+1)$). Εφόσον, ο πίνακας των συντελεστών $A$ είναι μη – αντιστρέψιμος (singular), δεν υπάρχει λύση ενός τέτοιου υπερ-προσδιορισμένου συστήματος. Ωστόσο, μπορούμε να λύσουμε το σύστημα αν το μετασχηματίσουμε πολλαπλασιάζοντας με τον ανάστροφο του $A$ τα δύο μέρη του συστήματος: $(A'*A)\,c = (A'*b)$. Ο πίνακας συντελεστών $(A'*A)$ είναι πλέον ένας 2 επι 2 αντιστρέψιμος πίνακας έτσι ώστε να υπάρχει μια μοναδική λύση για το $c$. Αυτή η λύση είναι η ίδια με αυτή που δίνει το ελάχιστο του τετραγώνου του ολικού σφάλματος $E$.

Γραμμική παλινδρόμηση (Linear regression)

$$c_1 x_j + c_2 = y_j \qquad j = 1,2,\ldots,(n+1)$$

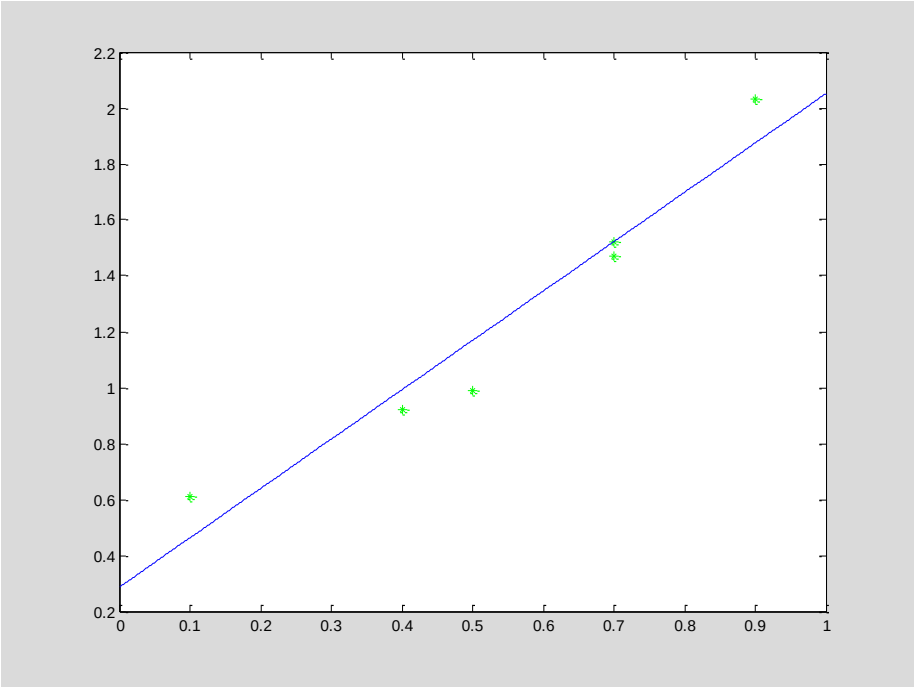
ως λύση του συστήματος $(A'*A)\,c = (A'*b)$.

Η υλοποίηση της παραπάνω λύσης δίδεται από τον παρακάτω MatLab κώδικα:

```
x = [ 0.1,0.4,0.5,0.7,0.7,0.9 ];
y = [ 0.61,0.92,0.99,1.52,1.47,2.03];
A = [ x', ones(length(x),1) ];
   % the coefficient matrix of the over-determined problem
b = y'; % right-hand-side of the over-determined problem
c = (A'*A) \ (A'*b)  % solution of the over-determined problem
```

```
xApr = 0 : 0.001 : 1; yApr = c(1)*xApr + c(2);
plot(x,y,'*g',xApr,yApr,'b');
```

```
c =    1.7646    0.2862
```



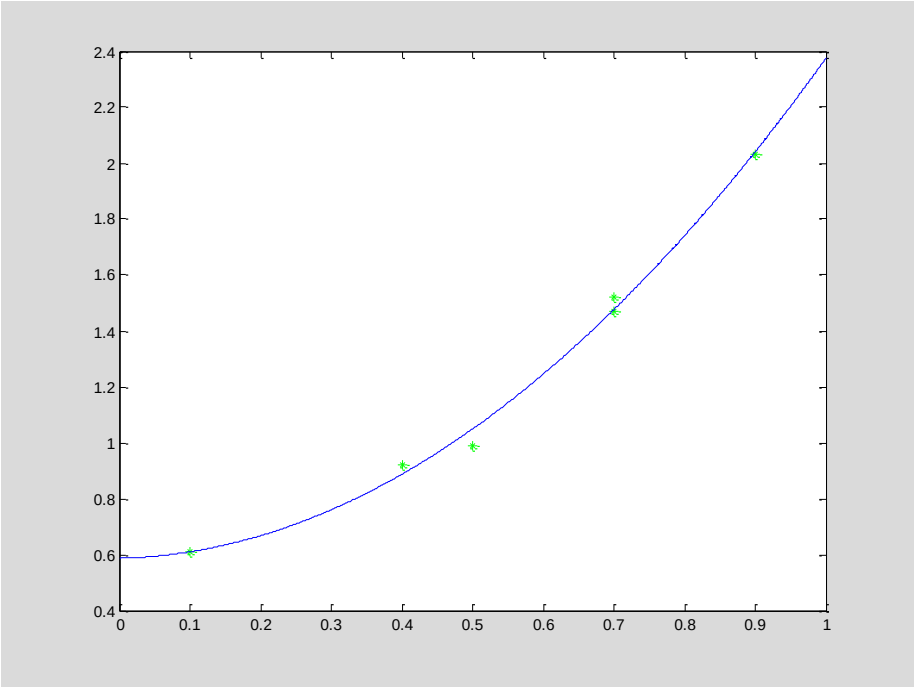

```
A, b, c = A\b
   % MATLAB solves the over-determined system in the least square sense
E = sum((y-c(1)*x-c(2)).^2)
```

```
A = 0.1000    1.0000     b = 0.6100              c = 1.7646
    0.4000    1.0000         0.9200                  0.2862
    0.5000    1.0000         0.9900
    0.7000    1.0000         1.5200              E = 0.0856
    0.7000    1.0000         1.4700
    0.9000    1.0000         2.0300
```

Το τετράγωνο του ολικού σφάλματος ***E*** γίνεται μικρότερο εάν ο βαθμός ***m*** < ***n*** του πολυώνυμο προσέγγισης $y = \Phi_m(x)$ αυξάνει**.** Το τετράγωνο του ολικού σφάλματος ***E*** είναι μηδέν, εάν ***m=n,*** δηλαδή το πολυώνυμο προσέγγισης $y = \Phi_m(x)$ είναι το πολυώνυμο παρεμβολής $y = P_n(x)$ που διέρχεται από όλα τα ***(n+1)*** σημεία δεδομένων. Ο παρακάτω κώδικας επιβεβαιώνει αυτή την παρατήρηση.

```
x = [ 0.1,0.4,0.5,0.7,0.7,0.9 ];
y = [ 0.61,0.92,0.99,1.52,1.47,2.03];
n = length(x)-1;
for m = 1 : n
A = vander(x); A = A(:,n-m+1:n+1);
   b = y'; c = A\b; yy = polyval(c,x);
E = sum((y-yy).^2)  ;
   fprintf('m = %d, E = %6.5f\n',m,E);
end
m = 2; A = vander(x); A = A(:,n-m+1:n+1); b = y';
c = A\b; xApr = 0 : 0.001 : 1; yApr = polyval(c,xApr);
plot(x,y,'*g',xApr,yApr,'b');
```

```
m = 1, E = 0.08564
m = 2, E = 0.00665
m = 3, E = 0.00646
m = 4, E = 0.00125
Warning: Matrix is singular to working precision.
m = 5, E =    Inf
```




Το σύστημα των γραμμικών εξισώσεων για την προσέγγιση της ελαχιστοποίησης του τετραγώνου ***(A'*A) c = (A'*b)*** είναι κακής κατάστασης όταν το ***n*** είναι μεγάλο καθώς και όταν τα σημεία δεδομένων περιλαμβάνουν συνδυασμό πολύ μικρών και πολύ μεγάλων τιμών για το ***x.*** Σ' αυτές τις περιπτώσεις πολύ συχνά χρησιμοποιούμε προσεγγίσεις με ορθογώνια πολυώνυμα.

- Για ***n+1*** δεδομένα**,** η συνάρτηση MatLab **polyfit** υπολογίζει τους συντελεστές πολυωνύμου προσέγγισης βαθμού ***m*** **με** την μέθοδο των ελαχίστων τετραγώνων όταν ***m<n*** και συντελεστές πολυωνύμων παρεμβολής όταν το ***m=n***

```
c = polyfit(x,y,1)
c =    1.7646    0.2862
```

## 4. Μη-γραμμική προσέγγιση

Πολλές φορές οι θεωρητικές εξαρτήσεις δεδομένων μπορούν να αναπαρασταθούν με μη-γραμμικές συναρτήσεις παρά με πολυώνυμα. Για παράδειγμα, οι θεωρητικές καμπύλες μπορούν να προσεγγιστούν από δυναμοσυναρτήσεις και εκθετικές συναρτήσεις. Στις περιπτώσεις αυτές, οι άξονες ***(x,y)*** μπορεί να επαναπροσδιοριστούν έτσι ώστε να μπορεί να χρησιμοποιηθεί γραμμικό πολυώνυμο για την προσέγγιση των μη-γραμμικών συναρτήσεων με την μέθοδο των ελαχίστων τετραγώνων.

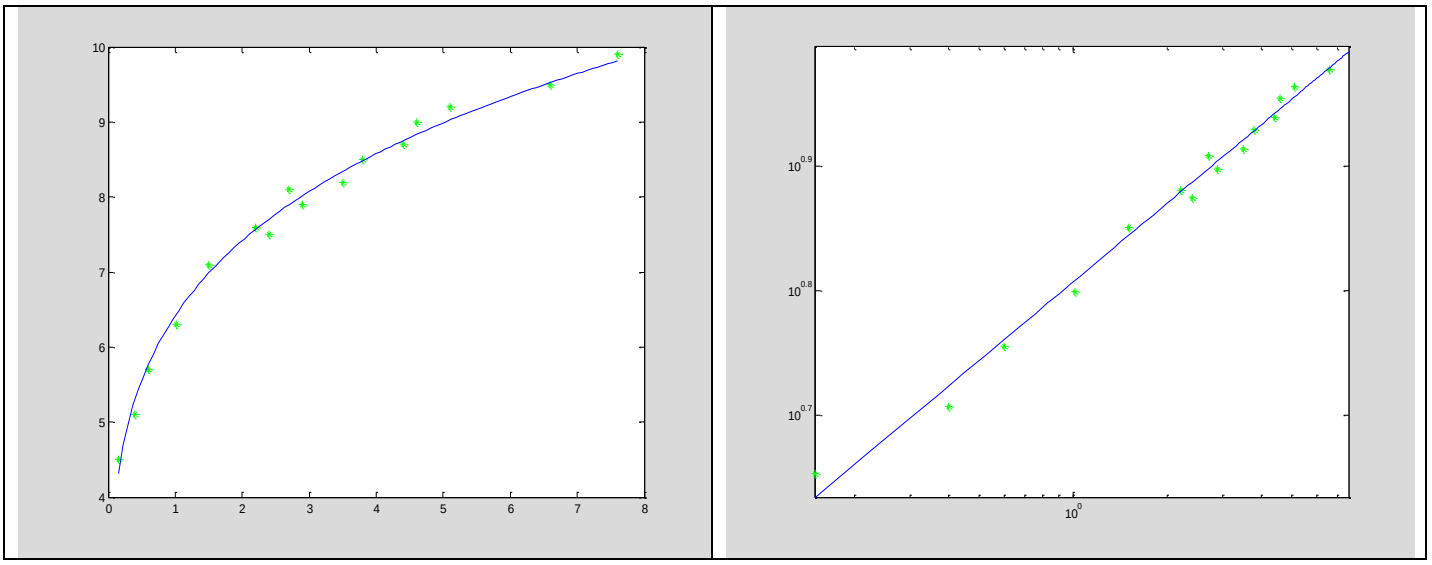
### α) Προσέγγιση δεδομένων με δυναμοσυναρτήσεις: $y = \beta x^{\alpha}$

Στην περίπτωση αυτή, παίρνοντας το λογάριθμο της δυναμοσυνάρτησης μετασχηματίζουμε την συνάρτηση σε γραμμική στους άξονες ***(logx,logy)***:

$$\log y = \alpha \log x + \log \beta$$

και το δείγμα δεδομένων μπορεί να προσεγγιστεί με μια γραμμική παλινδρόμηση. Ο παρακάτω κώδικας δίνει ένα παράδειγμα της υλοποίησης της μεθόδου

```
x =
[0.15,0.4,0.6,1.01,1.5,2.2,2.4,2.7,2.9,3.5,3.8,4.4,4.6,5.1,6.6,7.6];
y = [4.5,5.1,5.7,6.3,7.1,7.6,7.5,8.1,7.9,8.2,8.5,8.7,9.0,9.2,9.5,9.9];
c = polyfit(log(x),log(y),1) % linear regression in logarithmic axis
xInt = linspace(x(1),x(length(x)),100);
yInt = exp(c(2))*xInt.^(c(1));
plot(x,y,'g*',xInt,yInt,'b');  % nonlinear regression in (x,y)
loglog(x,y,'g*',xInt,yInt,'b'); % linear regression in (logx,logy)

c =    0.2094    1.8588
```




### β) Προσέγγιση δεδομένων με εκθετικές συναρτήσεις: $y = \beta e^{\alpha x}$

Στην περίπτωση αυτή παίρνουμε το λογάριθμο της εκθετικής συνάρτησης:

$$\log y = \alpha x + \log \beta$$

Σε ημιλογαριθμικούς άξονες ***(x,logy),*** το δείγμα δεδομένων μπορεί να προσεγγιστεί με γραμμικό πολυώνυμο.

### γ) Προσέγγιση με γραμμικό συνδυασμό μη-γραμμικών συναρτήσεων

$$y = c_1 f_1(x) + c_2 f_2(x) + \ldots + c_n f_n(x)$$

**Η γενική μη-γραμμική προσέγγιση** $y = f(x;c_1,c_2,\ldots,c_n)$ αντιστοιχεί στην λύση ενός πολύπλοκου μη γραμμικού προβλήματος βελτιστοποίησης.

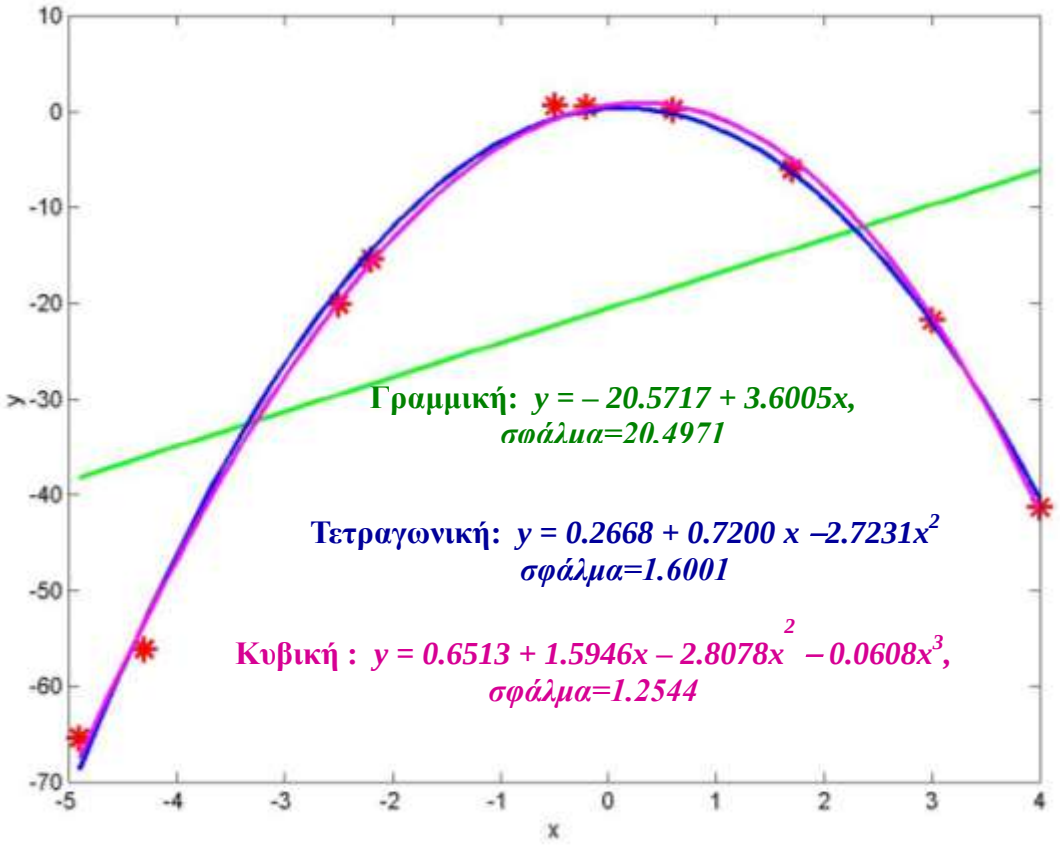
## 5. Παραδείγματα χρήσης της Matlab συνάρτησης polyfit

**Παράδειγμα**: Ο παρακάτω κώδικας προσαρμώζει τα πολυώνυμα 1ου, 2ου, και 3ου βαθμού στα παρακάτω δεδομένα, υπολογίζει το σφάλμα τους

$$E_2(f) = (\frac{1}{n}\sum_{k=1}^{m}|f(x_k) - y_k|^2)^{1/2}$$

και παράγει την γραφική παράσταση δεδομένων και πολυωνύμων.

```
x = [ -2.5   3.0  1.7  -4.9 0.6 -0.5   4.0  -2.2  -4.3 -0.2];
y = [-20.1 -21.8 -6.0 -65.4 0.2  0.6 -41.3 -15.4 -56.1  0.5];
n=length(x);
sx=sum(x);sx2=sum(x.^2);
x1=min(x); x2=max(x); xx=x1:(x2-x1)/100:x2;
c1=polyfit(x,y,1);
ny1=polyval(c1,xx);
c2=polyfit(x,y,2);
ny2=polyval(c2,xx);
c3=polyfit(x,y,3);
ny3=polyval(c3,xx);
e1=sqrt( sum(abs(polyval(c1,x)-y).^2 )/n )
e2=sqrt( sum(abs(polyval(c2,x)-y).^2 )/n )
e3=sqrt( sum(abs(polyval(c3,x)-y).^2 )/n )
H=plot(x,y,'r*',xx,ny1,'g',xx,ny2,'b',xx,ny3,'m');
xlabel('x'); ylabel('y');
set(H,'LineWidth',3,'MarkerSize',12);
```




## 6. Παραδείγματα χρήσης της συνάρτησης polyfit Python

**Παράδειγμα**: Ο παρακάτω κώδικας προσαρμώζει τα πολυώνυμα 1ου, 3ου, και 30ου βαθμού στα παρακάτω δεδομένα, υπολογίζει το σφάλμα τους

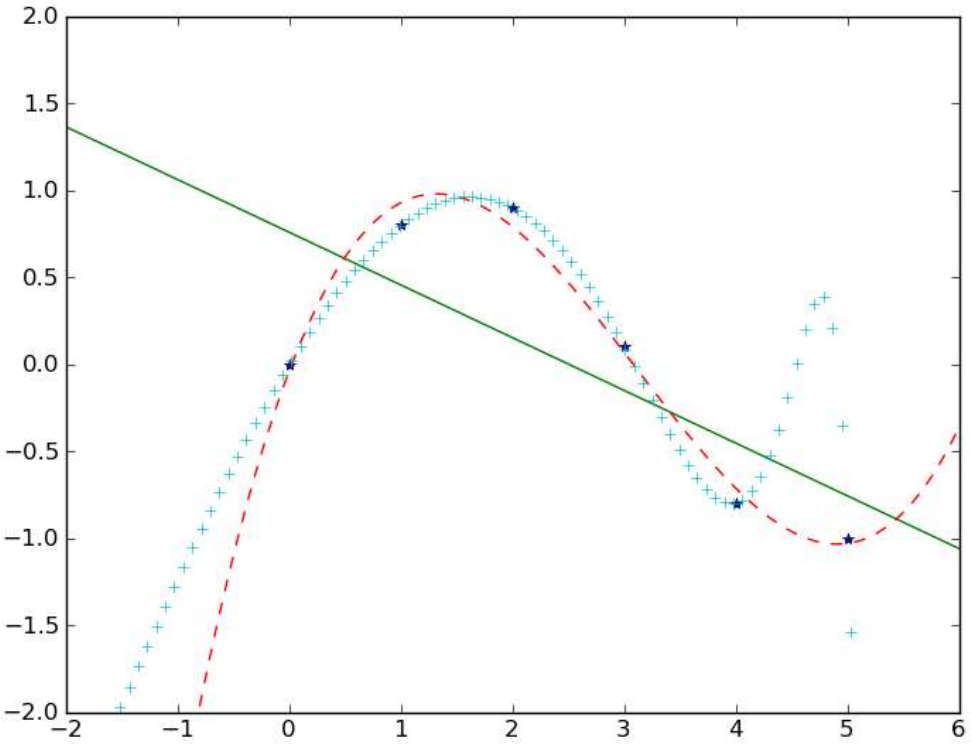
$$E_2(f) = (\frac{1}{n}\sum_{k=1}^{m}|f(x_k) - y_k|^2)^{1/2}$$

και παράγει την γραφική παράσταση δεδομένων και πολυωνύμων.

```
import matplotlib.pyplot as plt
import numpy as np
```

```
from math import sqrt
x = np.array([0.0, 1.0, 2.0, 3.0,  4.0,  5.0])
y = np.array([0.0, 0.8, 0.9, 0.1, -0.8, -1.0])
n=len(x)
xx= np.linspace(-2,6,100)
c1=np.polyfit(x,y,1)
ny1=np.polyval(c1,xx)
c2=np.polyfit(x,y,3)
ny2=np.polyval(c2,xx)
c3=np.polyfit(x,y,30)
ny3=np.polyval(c3,xx)
e1=sqrt( sum(abs(np.polyval(c1,x)-y)**2 )/n )
e2=sqrt( sum(abs(np.polyval(c2,x)-y)**2 )/n )
e3=sqrt( sum(abs(np.polyval(c3,x)-y)**2 )/n )
print e1, e2, e3
plt.plot(x, y, '*', xx, ny1, '-', xx, ny2, '--',xx, ny3, '+')
plt.ylim (-2, 2)
plt.show()
```

το σφάλμα για κάθε πολυώνυμο είναι:
0.499142121187 0.297369419121 2.44818757089e-15
Εξηγείστε το αποτέλεσμα για το πολώνυμο 30ου βαθμού.



## 7. Ασκήσεις

### Άσκηση 1: Ένα απλό πρόβλημα ελαχίστων τετραγώνων (ΕΤ)

Ένας πλανήτης ακολουθεί ελλειπτική τροχιά, που μπορεί να παρασταθεί στις καρτεσιανές συντεταγμένες (x,y) από την εξίσωση: $ay^2 + bxy + cx + dy + e = x^2$

Χρησιμοποιείστε την MATLAB για να προσδιορίσετε τις παραμέτρους a, b, c, d, e λύνοντας το σύστημα ΕΤ, λαμβάνοντας υπόψη τις παρακάτω παρατηρήσεις της θέσης του πλανήτη:

```
x = [ 1.02 0.95 0.87 0.77 0.67 0.56 0.44 0.30 0.16 0.01 ]';
y = [ 0.39 0.32 0.27 0.22 0.18 0.15 0.13 0.12 0.13 0.15 ]';
```

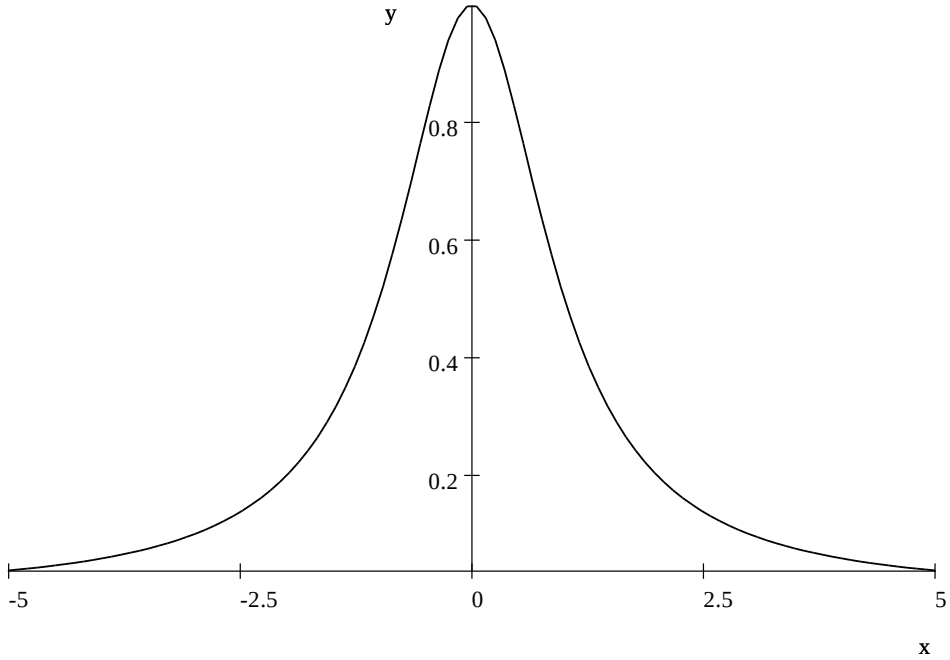
Κάντε την γραφική παράσταση της τροχιάς στα δεδομένα σημεία στο (x, y) επίπεδο.

Το ΕΤ πρόβλημα είναι ασταθές. Για να δείτε την επίδραση που έχει η αστάθεια του στην λύση, μεταβάλετε τα δεδομένα ελαφρώς προσθέτοντας σε κάθε συντεταγμένη του κάθε σημείου ένα τυχαίο αριθμό με ομαλή κατανομή στο διάστημα [-0.005, 0.005] και υπολογίστε την λύση του ΕΤ με τα διαταραχθέντα σημεία.

Ποια είναι η ακρίβεια που παίρνετε με τα διαταραχθέντα σημεία; Ποιος είναι ο αριθμός κατάστασης του συστήματος;

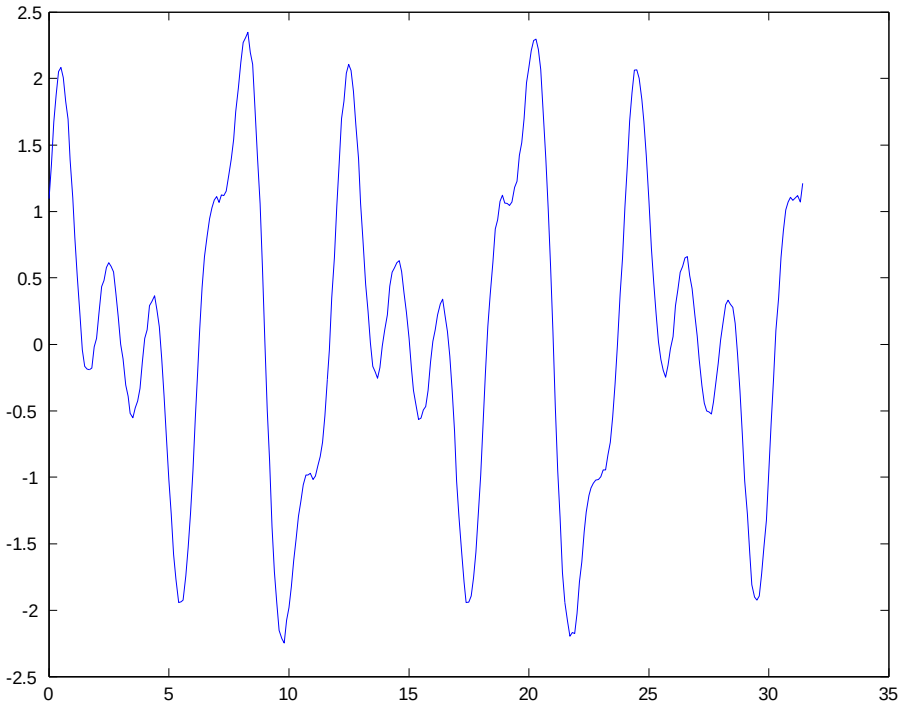
### Άσκηση 2: Προσέγγιση ελαχίστων τετραγώνων και λύση υποπροσδιορισμένων συστημάτων

Θεωρήστε τη συνάρτηση $f(x) = \frac{1}{1+x^2}$




στο διάστημα [-5,5] . Στο παραπάνω γράφημα απεικονίζεται η συνάρτηση $f(x)$ στην περιοχή του πεδίου ορισμού της.

Επιπλέον, θεωρήστε τα δεδομένα στο αρχείο data.mat των οποίων το γράφημα είναι




**Πρόβλημα 1:** Γράψτε ένα MATLAB πρόγραμμα για να βρείτε τα πολυώνυμα βαθμών $n = 3, 7, 15, 21$ που προσεγγίζουν την $f(x)$ λαμβάνοντας υπόψη 60 σημεία της που αντιστοιχούν σε 60 ισαπέχοντα x σημεία στο διάστημα $[-5, 5]$.

- Χρησιμοποιείστε τις κανονικές (normal) εξισώσεις και εφαρμόστε τις QR και SVD μεθόδους για την λύση τους.
- Σχεδιάστε τα πολυώνυμα από κάθε μέθοδο και τα δεδομένα στην ίδια γραφική παράσταση.
- Υπολογίστε και σχεδιάστε τα σφάλματα για κάθε περίπτωση.
- Εξηγήστε τα αποτελέσματα. Βλέπετε μια συγκεκριμένη συμπεριφορά?

**Πρόβλημα 2:**

- Επαναλάβετε τα βήματα του προβλήματος 1 αλλά για τα δεδομένα στη $2^{η}$ εικόνα.
- Θεωρήστε την προσέγγιση $a\cos(\pi t/3) + b\sin(\pi t/3) + c\sin(\pi t)$ και προσδιορίστε την για το ίδιο σύνολο δεδομένων.
- Εξηγήστε τα αποτελέσματά σας.

**Πρόβλημα 3:**

Σε όλες τις προσεγγίσεις που υπολογίσατε, υπολογίστε και τη νόρμα του σφάλματος και την Ευκλείδεια νόρμα των υπολοίπων (των σημειακών δηλαδή σφαλμάτων) για κάθε γραμμικό σύστημα που χρησιμοποιήσατε.

**Παρατηρήσεις: Μπορεί να βρείτε χρήσιμο το παρακάτω πρόγραμμα σε Matlab**

```
%Polynomial least square approximation of the function
%y=exp(sin(4*t))defined in the domain [0,1]
% the user defined parameters of this program are:
% a) m - the number of data points chosen in the interval
[0,1]
% b) n - the degree of the polynomial approximation
% c) mval - the number of points at which the polynomial
approximation is
% computed
% programmer: Elias Houstis
m=30;n=11;mval=60; % parameter definition
t=(0:m-1)'/(m-1); % set a partition of [0,1] with m
elements in array t
y=exp(sin(4*t)); % the values of approximate function at
the partitioning points.
hold off
plot(t,y,'g*') % plots the known data points
hold on
A=[]; %defines Van der Monde matrix
for i=1:n
 A=[A t.^(i-1)]; %generates the Van der Monde matrix
end
a=A'*A; %forms the normal equations
b=A'*y;
% computes the conditional numbers and residuals of the
normal and
% overdetermined systems
cond(a)
cond(A)
%Solve the overdetermined system using Lu and QR
factorization methods.
%Appropriate routines are selected by MATLAB and computes
the norms of the
%residuals
x=a\b;
r=a*x-b;
norm(r)
xqr=A\y;
rqr=A*xqr-y;
norm(rqr)
%permutes the unknown vector according to the requirements
of the polval
```

```
%routine or function
for i=1:n
 p(i)=x(n-i+1);
end
% computes the polynimial at mval points
tt=(0:mval-1)'/(mval-1);
yy=polyval(p,tt);
% plots the polynomial in the same graph with the data
points
plot(tt,yy,'r-')
```

## 5. Αναφορές


1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. https://sites.google.com/a/uni-konstanz.de/na09/Home/
3. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
4. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
5. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
6. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
7. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.

# ΚΕΦΑΛΑΙΟ 6: ΠΡΟΣΕΓΓΙΣΗ ΣΥΝΑΡΤΗΣΕΩΝ ΚΑΙ ΔΕΔΟΜΕΝΩΝ ΜΕ ΤΡΙΓΩΝΟΜΕΤΡΙΚΑ ΠΟΛΥΩΝΥΜΑ

## Περιεχόμενα



Signal processing

## 1. Προσέγγιση συναρτήσεων κατά Taylor

Μια συνηθισμένη μέθοδος για να κατανοήσουμε "πολύπλοκα" μαθηματικά μοντέλα είναι να επιχειρήσουμε να τα προσεγγίσουμε ως όρια σχετικά "απλών" μοντέλων. Το σύνηθες παράδειγμα είναι να προσεγγίσουμε μια τυχαία συνάρτηση $f : R \to R$ ως γραμμικό συνδυασμό μονωνύμων 1, $x$, $x^2$,... όπως είδαμε στα κεφάλαια 1 και 2. Σε αυτή τη περίπτωση η προσέγγιση τάξης ***n*** της συνάρτησης $f(x)$ είναι της μορφής

$$c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n = \sum_{j=0}^{n} c_j x^j$$

όπου $c_j$ είναι πραγματικοί αριθμοί που εξαρτώνται από τη συνάρτηση $f$. Αν η $f$ δεν είναι πολυώνυμο, δεν υπάρχει ***n*** ώστε η προσέγγιση ***n*** τάξεως της συνάρτησης $f(x)$ να είναι

ταυτόσημη της $f(x)$ και έτσι αντί γι' αυτό ελπίζουμε ότι το όριο των προσεγγίσεων της τάξης ***n*** συγκλίνει στην $f(x)$, δηλαδή ισχύει

$$f(x) = \lim_{n\to\infty} \sum_{j=0}^{n} c_j x^j$$

Υπάρχουν πολλά που θα πρέπει να προσδιοριστούν σε μια δήλωση σαν κι αυτή. Για παράδειγμα τι σημαίνει αυτό το όριο και για ποιες τιμές του ***x*** μπορεί να ισχύει αυτή η ισότητα. Προς το παρόν θα πρέπει να αρκεστούμε στο ερώτημα του πως μπορούμε να βρούμε τους συντελεστές $c_j$ με συστηματικό τρόπο. Με την προϋπόθεση ότι η $f(x)$ έχει τις απαραίτητες παραγώγους, μια απάντηση σε αυτό το ερώτημα δίνεται από τη θεωρία της σειράς Taylor. Για σημεία $x$ που είναι επαρκώς κοντά στο 0, έχουμε:

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2}x^2 + \cdots + \frac{f^n(0)}{n!} + \cdots$$

Παρατηρείστε ότι για να βρούμε τους συντελεστές $c_j$ πρέπει να έχουμε πληροφορίες για τις παραγώγους της $f(x)$ στο $x = 0$.

**Παράδειγμα:**
Ας υποθέσουμε ότι $f(x) = e^x$ τότε γνωρίζουμε ότι $f^{(n)}(x) = e^x$ για όλα τα $n$. Επομένως $f^{(n)}(0) = 1$ για κάθε ***n*** και έτσι η σειρά Taylor για το $e^x$ είναι

$$e^x = 1 + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \cdots$$

Επίσης, η MATLAB έχει μια ενσωματωμένη εντολή για να υπολογίζει τη σειρά Taylor μιας συνάρτησης: taylor. Για παράδειγμα αν θέλαμε να βρούμε τους δέκα πρώτους όρους της σειράς Taylor για το $e^x$, αντί για τους πρώτους έξι (που μόλις τώρα αναφέραμε), χρησιμοποιούμε τις ακόλουθες εντολές της MATLAB:

```
>> syms x
>> taylor('exp(x)',x,10)
ans =
x^9/362880 + x^8/40320 + x^7/5040 + x^6/720 + x^5/120 + x^4/24 + x^3/6 + x^2/2 + x
+ 1
```

Η συνάρτηση taylor έχει τρείς παραμέτρους: η πρώτη είναι η συνάρτηση (παρατηρείστε ότι βρίσκεται εντός απλών εισαγωγικών), η δεύτερη είναι η μεταβλητή, και η τρίτη είναι ο αριθμός των όρων που θα θέλαμε στη σειρά. Παρατηρείστε επίσης ότι πρέπει να ξεκινήσουμε δηλώνοντας το $x$ ως *συμβολική* μεταβλητή χρησιμοποιώντας την εντολή syms.

**Άσκηση 1:** Υπολογίστε τους πρώτους επτά όρους της σειράς Taylor για την $f(x) = -\log(1 - x)$. Υπολογίστε τους επτά πρώτους όρους της σειράς Taylor για την $g(x) = e^x/\cos(x) + \tan^3(x)$.

## 2. Προσέγγιση συναρτήσεων με σειρές Fourier

Ενώ οι σειρές μονωνύμων $\{x^j\}$ είναι χρήσιμες για την προσέγγιση τυχαίων συναρτήσεων, σε ορισμένα προβλήματα πρέπει να επιλέξουμε σειρές συναρτήσεων που έχουν τις ίδιες ιδιότητες με αυτές της συνάρτησης που πρόκειται να προσεγγίσουν. Για παράδειγμα, εάν θέλουμε να προσεγγίσουμε τις συναρτήσεις $f : \mathrm{R} \to \mathrm{R}$ οι οποίες ικανοποιούν την $f(x) = f(x + 2\pi)$ (δηλαδή συναρτήσεις που είναι περιοδικές με περίοδο $2\pi$), είναι συνηθισμένο να χρησιμοποιούμε σειρές των παρακάτω συναρτήσεων

$$\cdots, e^{-3ix}, e^{-2ix}, e^{-ix}, 1, e^{ix}, e^{2ix}, e^{3ix}, \cdots.$$

Εδώ το $i$ είναι ο μιγαδικός αριθμός $\sqrt{-1}$. Παρόλο που μπορεί να φαίνεται παράξενο να χρησιμοποιούμε μιγαδικές συναρτήσεις για να προσεγγίσουμε πραγματικές συναρτήσεις $f : \mathrm{R} \to \mathrm{R}$ είναι ένα απλό αλλά σημαντικό μαθηματικό αποτέλεσμα το ότι αυτές οι συναρτήσεις «αρκούν» για να εκφράσουν όλες αυτού του είδους τις περιοδικές συναρτήσεις. Δηλαδή, ισχύει ότι για κάθε συνάρτηση $f(x)$ η οποία ικανοποιεί την $f(x) = f(x + 2\pi)$ υπάρχουν πραγματικοί αριθμοί

$\cdots, a_{-3}, a_{-2}, a_{-1}, a_0, a_1, a_2, a_3, \cdots$

έτσι ώστε

$$f(x) = \lim_{n\to\infty}[\sum_{j=-n}^{n} a_j e^{ijx}]$$

Αυτό ονομάζεται ανάπτυγμα σειρών Fourier για τη συνάρτηση $f(x)$. Εάν θέλετε μπορείτε να δείτε το $\sum_{j=-n}^{n} a_j e^{ijx}$ ως την προσέγγιση $n$ τάξεως της συνάρτησης $f(x)$. Οι συντελεστές $\{a_i\}$ ονομάζονται μιγαδικοί συντελεστές Fourier της συνάρτησης $f(x)$.

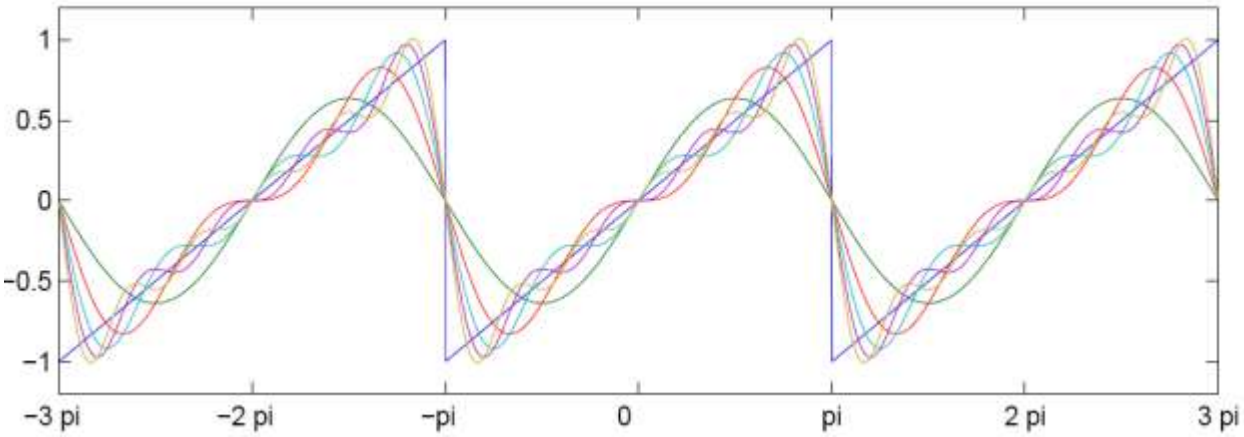
## Παράδειγμα

Η συνάρτηση

$$f(x) = 2(\frac{x+\pi}{2\pi} - \left\lfloor (\frac{x+\pi}{2\pi}) \right\rfloor) - 1$$

φαίνεται στη γραφική παράσταση 1 με μπλε χρώμα. Στους ίδιους άξονες δείχνουμε τις προσεγγίσεις πρώτης, δεύτερης, τρίτης, τέταρτης και πέμπτης τάξης της $f(x)$ (με πράσινο, κόκκινο, γαλάζιο, μωβ και κίτρινο χρώμα αντίστοιχα) χρησιμοποιώντας τους μιγαδικούς συντελεστές Fourier

$a_0 = 0$, $a_{\pm 1} = -\frac{i}{\pi}$, $a_{\pm 2} = \frac{i}{2\pi}$, $a_{\pm 3} = -\frac{i}{3\pi}$, $a_{\pm 4} = \frac{i}{4\pi}$, and $a_{\pm 5} = -\frac{i}{5\pi}$




**Σχήμα** 1. Η συνάρτηση $f(x)$ και οι προσεγγίσεις Fourier για το παράδειγμα

Όπως και με τις σειρές Taylor υπάρχουν λογικές ερωτήσεις που προκύπτουν για τις σειρές Fourier της συνάρτησης $f(x)$. Εδώ, θα περιορίσουμε τη συζήτησή μας στο πως μπορούμε να υπολογίσουμε τους αντίστοιχους συντελεστές Fourier. Ευτυχώς, όπως με τις σειρές Taylor, η θεωρία των σειρών Fourier είναι εξοπλισμένη με μια εύκολη μέθοδο υπολογισμού αυτών των συντελεστών. Αλλά ενώ οι σειρές Taylor χρησιμοποιούν παραγώγους για τον υπολογισμό των συντελεστών, για τις σειρές Fourier χρησιμοποιούμε ολοκληρώματα για να υπολογίσουμε τους αντίστοιχους συντελεστές. Συγκεκριμένα,

$$a_j = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-jix}dx.$$

## Παράδειγμα:

Παρατηρείστε ότι στο παραπάνω παράδειγμα, η συνάρτηση $f(x)$ ισοδυναμεί με την $g(x) = x/\pi$ στο διάστημα $[-\pi, \pi]$. Για να βρούμε το μιγαδικό συντελεστή Fourier $a_1$ για την $f(x)$, θα πρέπει να τρέξουμε τη συνάρτηση `int` στη MatLab.

```
>> clear x; syms x;
>> int((x/pi)*exp(-i*x),x,-pi,pi)/(2*pi)
ans =
-i*pi
```

Η εντολή `int` της MatLab έχει τέσσερις παραμέτρους: η πρώτη είναι η συνάρτηση προς ολοκλήρωση, η δεύτερη είναι η μεταβλητή της ολοκλήρωσης, η τρίτη είναι το αριστερό άκρο (σημείο) του διαστήματος ολοκλήρωσης και η τελική παράμετρος είναι το δεξί άκρο (σημείο) του διαστήματος ολοκλήρωσης. Παρατηρείστε, ότι το αποτέλεσμα του παραπάνω κώδικα είναι ο συντελεστής που χρησιμοποιήσαμε στο προηγούμενο παράδειγμα. Για να βρούμε τους υπόλοιπους συντελεστές θα πρέπει να υπολογίσουμε

παρόμοια ολοκληρώματα με το παραπάνω. Για παράδειγμα, για να βρούμε το συντελεστή $a_{-2}$ υπολογίζουμε

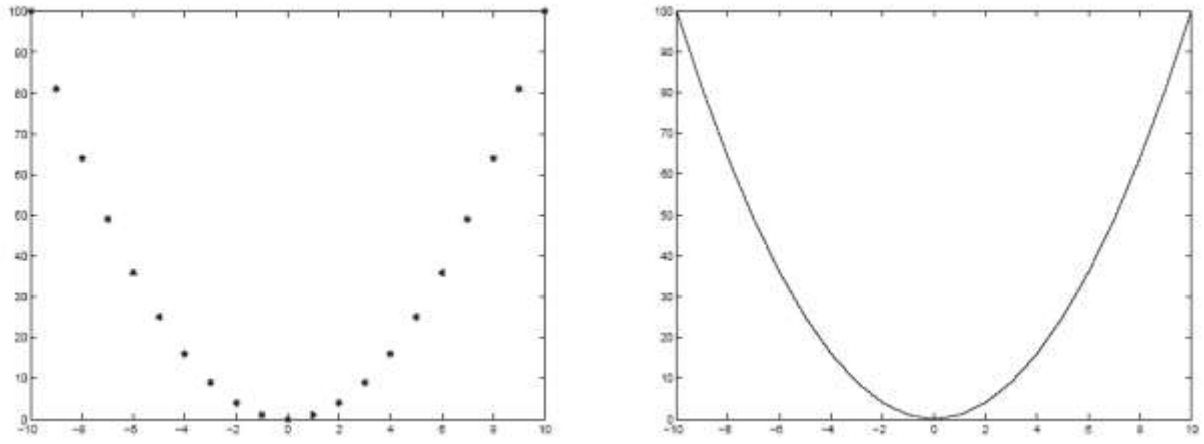
```
>> clear x; syms x;
>> int((x/pi)*exp(-i*-2*x),x,-pi,pi)/(2*pi)
```

Το προηγούμενο παράδειγμα μας δίνει μια σημαντική ιδέα στην εφαρμογή των σειρών Fourier: πρέπει να γνωρίζουμε μόνο τη συμπεριφορά της συνάρτησης στο διάστημα $[-\pi, \pi]$ για να μπορέσουμε να υπολογίσουμε τους συντελεστές Fourier. Πραγματικά αυτή η παρατήρηση μπορεί να χρησιμοποιηθεί για να δώσει την ανάπτυξη Fourier μιας συνάρτησης στο διάστημα $[-\pi, \pi]$ ακόμη και αν η συνάρτηση δεν είναι περιοδική. Με αυτό τον τρόπο, στην ουσία φτιάχνουμε μια περιοδική συνάρτηση με τα δεδομένα της συνάρτησης $f(x)$ στο διάστημα $[-\pi, \pi]$.

## 3. Από τη συνεχή στη διακριτή: Διακριτές σειρές Fourier

Οι προσεγγιστικές μέθοδοι που έχουν αναπτυχθεί μέχρι τώρα μας επιτρέπουν να γράψουμε τις συναρτήσεις $f : \mathrm{R} \to \mathrm{R}$ ως αθροίσματα βασικών συναρτήσεων. Αυτές οι μέθοδοι επεκτείνονται και σε άλλες κατηγορίες συναρτήσεων. Ειδικότερα, είναι συχνά πιο βολικό να δουλεύουμε με συναρτήσεις των οποίων τα πεδία ορισμού είναι διακριτά αντί να είναι συνεχή. Για παράδειγμα, μια συνάρτηση με διακριτό πεδίο ορισμού θα είναι απλά μια συνάρτηση της οποίας το πεδίο ορισμού είναι υποσύνολο των ακεραίων. Για να είναι πιο απλό, ορίζουμε το Z να αντιπροσωπεύει το σύνολο των ακεραίων:
$\mathrm{Z} := \{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\}$.

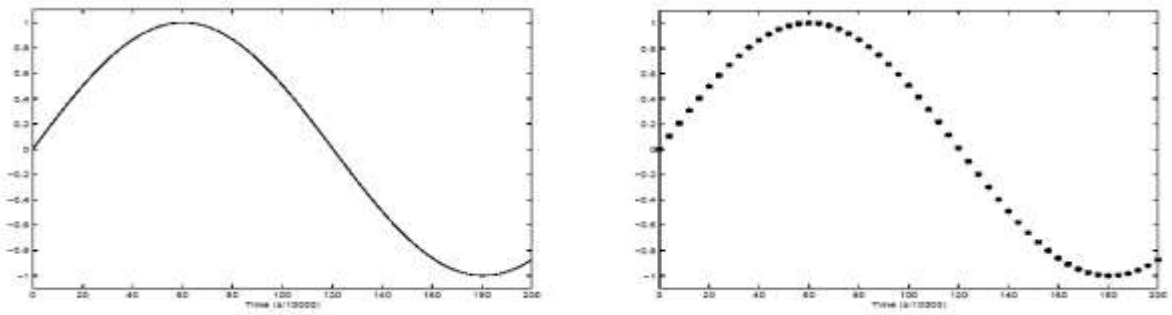
**Παράδειγμα:**
Η συνάρτηση $f : \mathrm{Z} \to \mathrm{R}$ που δίνεται από την $f(x) = x^2$ είναι μια συνάρτηση με διακριτό πεδίο ορισμού, ενώ η συνάρτηση $g : \mathrm{R} \to \mathrm{R}$ που δίνεται από τη $g(x) = x^2$ δεν είναι (βλέπε Σχήμα 2).



**Σχήμα** 2. Η $f(x)$ έχει διακριτό πεδίο ορισμού ενώ η $g(x)$ δεν έχει

Οι συναρτήσεις με διακριτά πεδία ορισμού εμφανίζονται παντού στη καθημερινή μας ζωή. Για παράδειγμα, ο ήχος που παράγεται από μια χορδή πιάνου για τι Μι ματζόρε

αντιπροσωπεύεται από μια συνάρτηση της οποίας η γραφική παράσταση φαίνεται στα αριστερά του Σχήματος 3 όπου ψηφιοποιείται ο ήχος έτσι ώστε να μπορεί να γίνει η εγγραφή του σε CD, ωστόσο, παράγει κωδικοποιημένες πληροφορίες οι οποίες θα μοιάζουν με αυτό που βλέπουμε στη δεξιά πλευρά του Σχήματος 3.



**Σχήμα** 3. Συνεχής (πραγματικός) σε αντίθεση με διακριτό (ψηφιοποιημένο) ήχο

Αυτό το παράδειγμα είναι μια συνηθισμένη χρήση συναρτήσεων με διακριτό πεδίο ορισμού: στην επιστήμη και την τεχνολογία τα δεδομένα του «πραγματικού κόσμου» φαίνονται σε διακριτά χρονικά διαστήματα, και τα δείγματα δεδομένων χρησιμοποιούνται ως μοντέλο του παρατηρημένου φαινομένου. Γενικά, ελπίζουμε να βρούμε αρκετές πληροφορίες έτσι ώστε ένας άνθρωπος να μην μπορεί να ξεχωρίσει μεταξύ του πραγματικού (συνεχούς) φαινομένου και της κωδικοποιημένης (διακριτής) αναπαράστασης. Η κωδικοποίηση του ήχου απαιτεί να πάρουμε δείγματα τουλάχιστον 48.000 φορές το δευτερόλεπτο.

Τώρα ας υποθέσουμε ότι $f : \mathrm{Z} \to \mathrm{R}$ είναι μια συνάρτηση με διακριτό πεδίο ορισμού. Ας υποθέσουμε επίσης ότι είναι $m$-περιοδική, δηλαδή ότι το $m$ είναι ένας ακέραιος που έχει την ιδιότητα ότι για κάθε άλλον ακέραιο $x$, ισχύει $f(x + m) = f(x)$. Όπως και στην περίπτωση του συνεχούς πεδίου ορισμού, η $f(x)$ μπορεί να γραφτεί ως άθροισμα βασικών συναρτήσεων. Η διαφορά σε αυτή την περίπτωση είναι ότι οι βασικές μας συναρτήσεις είναι της μορφής $e^{i2\pi\, j/mx}$ για τις τιμές του $j$ που κυμαίνονται μεταξύ του1 και του $m$. Επομένως προσπαθούμε να βρούμε συντελεστές $b_j$ με

(1) $$f(x) = \sum_{j=1}^{m} b_j e^{i2\pi \frac{j}{m} x}$$

Εφόσον η συνάρτησή μας είναι διακριτή, η αντίστοιχη σειρά ονομάζεται διακριτή σειρά Fourier της $f(x)$. Παρατηρείστε ότι εδώ η συνάρτηση μας $f(x)$ εκφράζεται ως μια πεπερασμένη σειρά. Στη πραγματικότητα, η διακριτή σειρά Fourier έχει $m$ παραμέτρους, όπου $m$ είναι η περίοδος της συνάρτησης $f(x)$. Ο υπολογισμός των συντελεστών $b_j$, ξανά απαιτεί τον υπολογισμό ενός ολοκληρώματος κάποιου είδους. Ειδικότερα έχουμε

$$b_j = \frac{1}{m} \sum_{x=1}^{m} f(x) e^{-i2\pi \frac{j}{m} x}$$

Εφόσον η συνάρτηση μας έχει πεδίο ορισμού το οποίο είναι διακριτό αντί να είναι συνεχές, η ολοκλήρωση που πραγματοποιήσαμε στη συνεχή περίπτωση μπορεί να αντικατασταθεί χρησιμοποιώντας άθροιση.

**Παράδειγμα:**

Θεωρούμε τη διακριτή συνάρτηση $f(x)$ η οποία ικανοποιεί την $f(x) = f(x+6)$ έτσι ώστε $f(0) = f(2) = f(4) = 0$ και $f(1) = f(3) = f(5) = 1$. Τότε υπολογίζουμε το συντελεστή $b_2$ ακόλουθα:

```
>>  (1/6)*(1*exp(-i*2*pi*2/6*1)+1*exp(-i*2*pi*2/6*3)+1*exp(-
i*2*pi*2/6*5))
ans =
-1.8504e-016 -1.1102e-016i
```

Αυτό σημαίνει ότι $b_2 = -1.8504 \times 10^{-16} + -1.1102 \times 10^{-16}i$. Αυτοί οι πολύ μικροί αριθμοί είναι πιθανόν αποτέλεσμα σφάλματος στρογγυλοποίησης των υπολογισμών της MatLab και έτσι το $b_2 = 0$.

**Άσκηση 2:** Υπολογίστε τους συντελεστές $b_0$, $b_1$, $b_3$, $b_5$ για τη συνάρτηση $f(x)$ του προηγούμενου παραδείγματος. Στη συνέχεια χρησιμοποιήστε την εξίσωση 1 για να υπολογίσετε τις $f(0)$, $f(1)$, $f(2)$, $f(3)$, $f(4)$, $f(5)$ και $f(6)$.

## 4. Προσέγγιση συναρτήσεων με τριγωνομετρικά πολυώνυμα

Για την προσέγγιση περιοδικών συναρτήσεων, τα τριγωνομετρικά πολυώνυμα βαθμού m $S_m(t) = \frac{a_0}{2} + a_1\cos(t) + \ldots + a_m\cos(mt) + b_1\sin(t) + \ldots + b_m\sin(mt)$ παίζουν ένα σημαντικό ρόλο. Υποθέτουμε ότι ο συντελεστές $a_0, a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_m$ είναι πραγματικοί αριθμοί. Σημειώστε, ότι κάθε τριγωνομετρικό πολυώνυμο είναι περιοδική συνάρτηση με περίοδο 2π. Δηλαδή, $S_m(t) = S_m(t + 2\pi)$. Αν η συνάρτηση $f$ που μας ενδιαφέρει να προσεγγίσουμε με τριγωνομετρικά πολυώνυμα έχει περίοδο $T \neq 2\pi$ τότε θα χρειασθεί να την μετασχηματίσουμε σε 2π-περιοδική συνάρτηση $f(t) = f(\frac{T}{2\pi}t)$ $\hat{f}(t) = f(\frac{T}{2\pi}t)$ και μετά να την προσεγγίσουμε με $S_m(t)$. Ένα T-περιοδικό πολυώνυμο για την προσέγγιση της $f$ είναι $S_m(\frac{2\pi}{T}t)$. Ένα τριγωνικό πολυώνυμο μπορεί να παρασταθεί με διαφορετικούς τρόπους, χρησιμοποιώντας τον τύπο του Euler ($e^{i\theta} = \cos(\theta) + i\sin(\theta)$ ) έχουμε

$\sin(t) = \frac{e^{it} - e^{-it}}{2i}$, $\cos(t) = \frac{e^{it} + e^{-it}}{2}$, $S_m(t) = \sum_{-m}^{m} c_k e^{ikt}$ $\sin(t) = \frac{e^{it} - e^{-it}}{2}$, $S_m(t) = \sum_{-m}^{m} c_k e^{ikt}$

με συντελεστές

$a_k = c_k + c_{-k},\ k = 0,1,\ldots,m$ $και$ $b_k = i(c_k - c_{-k}),\ k = 1,\ldots,m$

$a_k = c_k + c_{-k}, k = 0,1, \ldots, d$ $και$ $b_k = i(c_k - c_{-k}),\ k = 1, \ldots, d$. Το γεγονός ότι υποθέσαμε ότι $S_m(t)$ παίρνει πραγματικές τιμές μπορούμε να αποδείξουμε εύκολα ότι

$a_k = 2\,\mathrm{Re}\,l(c_k),\ k = 0,1,...,m$ *και* $b_k = -2\,\mathrm{Im}(c_k),\ k = 1,...,m$
$a_k = 2Rel(c_k),\ k = 0,1,...,d$ *και* $b_k = -2Im(c_k),\ k = 1,...,d$ όπου *Rel(z)* και *Im(z)* παριστούν το πραγματικό και μιγαδικό μέρος ενός μιγαδικού αριθμού z.

**Σημειώστε** ότι

α) η τριγωνομετρική προσέγγιση έχει άμεση σχέση με την ανάπτυξη μιας περιοδικής συνάρτησης σε σειρά Fourier $f(t) = \sum_{-\infty}^{+\infty} c_k e^{ikt}$ $f(t) = \sum_{-\infty}^{\infty} c_k e^{ikt}$

β) από τον ορισμό του εσωτερικό γινομένου δύο μιγαδικών συναρτήσεων με περίοδο 2π, $\langle f, g\rangle = \frac{1}{2\pi}\int_0^{2\pi} f(t)\overline{g}(t)dt$ όπου $\overline{g}(t)$ είναι η συζυγής μιγαδική συνάρτηση, έχουμε

$\langle e^{ikt}, e^{ilt}\rangle = \frac{1}{2\pi}\int_0^{2\pi} e^{ikt}e^{-ilt}dt = 1$ αν $k = l$ και $=0$ αν $k \neq l$

γ) **Ισχύει το θεώρημα του Weierstrass:** Για κάθε συνεχή 2π-περιοδική συνάρτηση $f : R \to R$ $f: R \to R$ και κάθε ε >0, υπάρχει *m=m(ε)* και $S_m(t)$ έτσι ώστε $|S_m(t) - f(t) < \varepsilon|$ $\left|S_m(t) - f(t)\right| < \varepsilon$ για όλα τα $t \in R$.

## 5. Τριγωνομετρική παρεμβολή

**Θεώρημα**: Έστω ένα σύνολο συναρτήσεων $f_0(t), f_1(t),..., f_{n-1}(t)$ και σύνολο σημείων $(t_0, x_0), (t_1, x_1), \ldots, (t_{n-1}, x_{n-1})$, και ισχύει ότι ο πίνακας

$$A = \begin{bmatrix} f_0(t_0) & f_0(t_1) & . & . & . & f_0(t_{n-1}) \\ f_1(t_0) & f_1(t_1) & . & . & . & f_1(t_{n-1}) \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ f_{n-1}(t_0) & f_{n-1}(t_1) & . & . & . & f_{n-1}(t_{n-1}) \end{bmatrix}$$

είναι ορθοκανονικός ( $A^{-1} = A^T$ ). Αν $y = Ax$ τότε η συνάρτηση $F(t) = \sum_{i=0}^{n-1} y_i f_i(t)$ παρεμβάλει τα δοθέντα σημεία, δηλαδή έχουμε $F(t_k) = x_k,\ k = 0,...,n-1$ (Απόδειξη;)

Αν επιλέξουμε τα σημεία $t_k = k(2\pi / N)$ , $k = 0,1,...,N-1$ όπου ***N=2m*** στο διάστημα $[0, 2\pi)$ και επιλέξουμε τις συναρτήσεις $\{1/2, \cos(t),..., \cos((m-1)t), \cos t(mt), \sin(t),..., \sin((m-1)t)\}$, τότε μπορούμε να αποδείξουμε ότι ο αντίστοιχος πίνακας του θεωρήματος *A* είναι ορθογώνιος λόγω των ιδιοτήτων των τριγωνομετρικών συναρτήσεων (δες παράρτημα σε αυτό το κεφάλαιο), τότε μπορούμε να υπολογίσουμε τους συντελεστές του τριγωνομετρικού πολυωνύμου

$S_m(t) = \frac{a_0}{2} + a_1\cos(t) + ... + a_m\cos(mt) + b_1\sin(t) + ... + b_m\sin(mt)$ $a_0, a_1, a_2, ..., a_m, b_1, b_2, ..., b_m$ έτσι ώστε να παρεμβάλει τα σημεία $\{(t_0, x_0), (t_1, x_1), ..., (t_{N-1}, x_{N-1})\}$, δηλαδή ισχύουν οι σχέσεις $x_k = f(t_k),\ k = 0,...,N-1$ $x_k = f\left(\kappa\frac{2\pi}{N}\right), k = 0,..,N-1.$

Επειδή οι τριγωνομετρικές συναρτήσεις είναι **ορθογώνιες** στα δεδομένα σημεία, το

γινόμενο $Ax$ προσδιορίζει τους συντελεστές του τριγωνομετρικού πολωνύμου $S_m(t)$, δηλαδή έχουμε

$a_i = \frac{2}{N}\sum_{k=0}^{N-1} x_k \cos(ik\,\frac{2\pi}{N})$ , για $i = 0,...,m-1$ , $b_i = \frac{2}{N}\sum_{k=0}^{N-1} x_k \sin(ik\,\frac{2\pi}{N})$ για $i = 1,...,m-1$

## 6. Fourier παρεμβολή σε περισσότερα από $N$ σημεία (Μέθοδος Ελαχίστων Τετραγώνων)

**Θεώρημα**: Έστω m ≤ n ακέραιοι και $(t_0, x_0)$, $(t_1, x_1)$, …, $(t_{n-1}, x_{n-1})$, δεδομένα σημεία μιας συνάρτησης. Θερούμε το διάνυσμα y = Ax, όπου A είναι ορθογώνιος πίνακας που αντιστοιχεί στην βάση του προσεγγιστικού χώρου συναρτήσεων $f_0(t), f_1(t),..., f_{n-1}(t)$. Τότε, η συνάρτηση παρεμβολής με την παραπάνω βάση είναι

$$F_n(t) = \sum_{i=0}^{n-1} y_i f_i(t)$$

και η καλλίτερη προσέγγιση με την μέθοδο των ελαχίστων τετραγώνων είναι

$$F_m(t) = \sum_{i=0}^{m-1} y_i f_i(t).$$

**Παρατήρηση**: Αυτό είναι ένα όμορφο αποτέλεσμα. Με άλλα λόγια μας λέει ότι, δοθέντων n σημείων δεδομένων, για την εύρεση της καλλίτερης τριγωνομετρικής προσέγγισης με την μέθοδο των ΕΤ βαθμού m < n αρκεί να υπολογίσουμε την τριγωνομετρική παρεμβολή με n όρους και να κρατήσουμε μόνον τους πρώτους m όρους. Σημειώστε ότι, μπορούμε να κρατήσουμε όποιους m όρους «βολεύουν» την εφαρμογή.

**Πρόβλημα**: Θεωρείστε τα σημεία $t_k = k(2\pi / N)$ , $k = 0,1,...,N-1$ στο διάστημα $[0, 2\pi)$. Να βρεθούν οι συντελεστές του τριγωνομετρικού πολυωνύμου $a_0, a_1, a_2, ..., a_m, b_1, ..., b_{m-1}$ έτσι ώστε το πολυώνυμο

$S_m(t) = \frac{a_0}{2} + a_1\cos(t) + ... + a_m\cos(mt) + b_1\sin(t) + ... + b_m\sin((m-1)t)$

προσεγγίζει τα σημεία $\{(t_0, x_0), (t_1, x_1), ...(t_{N-1}, x_{N-1})\}$ , όπου $N > 2m$.

**Λύση:** Εφόσον, η συνάρτηση από την οποία προέρχονται τα σημεία $\{x_k\}$ $\{x_k\}$ είναι περιοδική, δηλαδή $x_N = x_1$ $x_N = x_1$ και αφού οι τριγωνομετρικές συναρτήσεις είναι ορθογώνιες στα δεδομένα σημεία βρίσκουμε το τριγωνομετρικό πολυώνυμο παρεμβολής για τα *N* σημεία και κρατάμε τους πρώτους *2m* όρους που έχουν συντελεστές

$a_i = \frac{2}{N}\sum_{\kappa=0}^{N-1} x_k \cos(ik\,\frac{2\pi}{N})$ , για $i = 0,...,m$ , $b_i = \frac{2}{N}\sum_{\kappa=0}^{N-1} x_k \sin(ik\,\frac{2\pi}{N})$ για $i = 1,...,m-1$

**Παρατήρηση:**

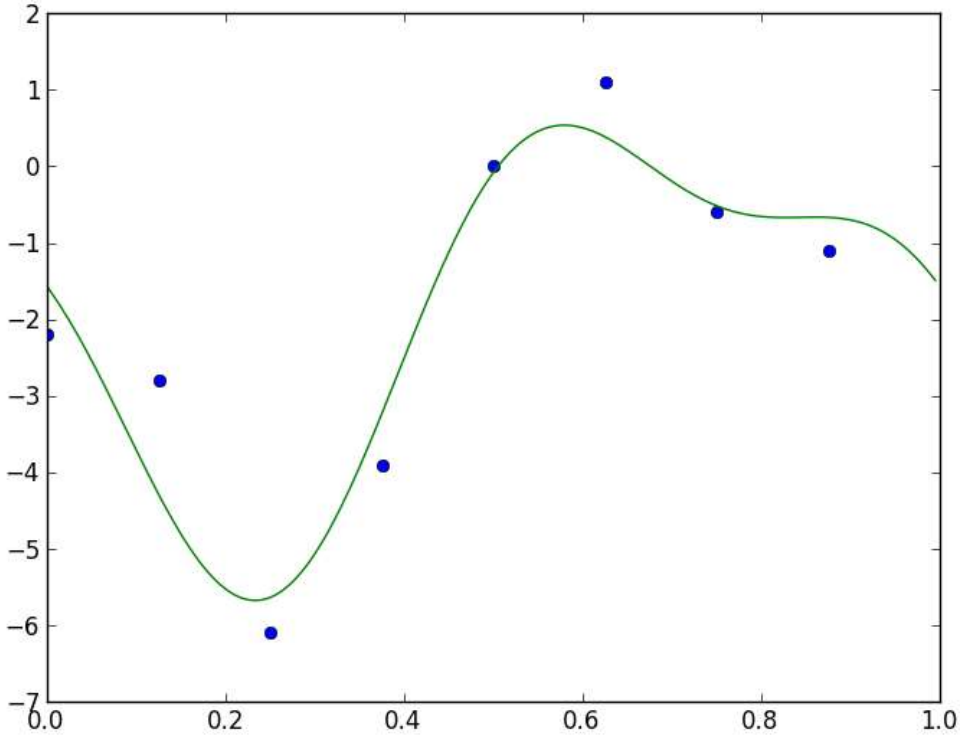
Η παραπάνω ιδιότητα των τριγωνομετρικών πολυωνύμων παρεμβολής ή ελαχίστων τετραγώνων προσεγγίσεις, να περιέχουν τις πληροφορίες για μικρότερου βαθμού τριγωνομετρικές προσεγγίσεις δεν ισχύει για τα αλγεβρικά πολυώνυμα. Δηλαδή, η παραβολική προσέγγιση ελαχίστων τετραγώνων στα σημεία (0, 3), (1, 3), (2, 5) είναι y

$= x^2 - x + 3$. Αν τώρα, προσαρμόσουμε στα δεδομένα το γραμμικό πολυώνυμο, τότε λαμβάνουμε $y= 8/3x+ 1$. Σημειώστε, ότι το γραμμικό πολυώνυμο δεν έχει καμμία σχέση με την προηγούμενη παραβολική προσέγγιση.

**Παράδειγμα**: Υλοποίηση της μεθόδου ελαχίστων τετραγώνων με τριγωνομετρικά πολυώνυμα

**Python κώδικας**:

```
from numpy import *
from scipy import fft, ifft
from pylab import plot, show
def dftfilter(inter,x,m,n,p):
c = inter[0]; d = inter[1]
t = linspace(c,d,n,endpoint=False) # time points for data
(n)
tp = linspace(c,d,p,endpoint=False) # time points for
interpolant (p)
y = fft(x) # compute interpolation coefficients
yp = zeros( p, dtype=complex ) # yp will hold coefficients
for ifft
yp[:m/2] = y[:m/2] # keep only first m frequencies
yp[ m/2] = real(y[m/2]) # since m is even, keep cos term
only
if m < n: # unless at the maximum frequency,
yp[p-m/2] = yp[m/2] # add complex conjugate to
corresponding place in upper tier
yp[p-m/2+1:p] = y[n-m/2+1:n] # more conjugates for upper
tier
xp = real(ifft(yp))*(float(p)/n) # invert fft to recover
data
plot(t,x,'o',tp,xp) # plot data and least square approx
show()
return xp
print   dftfilter([0,1],[-2.2,-2.8,-6.1,-3.9,0.0,1.1,-0.6,-
1.1],4,8,200)
```




## 7. Παρατηρήσεις για DFT

Ο αλγόριθμος DFT έχει περιγραφεί στο κεφάλαιο 4 μαζί με την γρήγορη υλοποίηση του FFT. Εδώ, τον ξαναθυμόμαστε και τον συγκρίνουμε με την συνεχή παραλαγή του. Η παρακάτω εξίσωση ορίζει τον διακριτό μετασχηματισμό Fourier :

$$X_n = \sum_{k=0}^{N-1} x_k e^{\frac{-i2\pi nk}{N}} \qquad (2)$$

Σημειώστε ότι η (2) είναι παρόμοια με τον συνεχή Fourier μετασχηματισμό που ορίζεται από την εξίσωση $X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt$ $X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$

εκτός του ότι:

- η συνεχής συνάρτηση $x(t)$ (το σήμα μας) έχει αντικατασταθεί από την ακολουθία $x_k$
- ο συνεχής χρόνος $t$ έχει αντικατασταθεί από το δείκτη χρόνου $k$ που παίρνει τιμές 0, 1, 2, …$N$-1
- Το μήκος του σήματος (ή το κομμάτι που μετασχηματίζεται) είναι $N$ δείγματα
- Το ολοκλήρωμα αντικαθιστάται με άθροισμα
- Η συνεχής συνάρτηση που παριστά την συχνότητα $X(\omega)$ έχει αντικατασταθεί με ακολουθία συχνοτήτων $X_n$
- η συνεχής συχνότητα $\omega$ έχει αντικατασταθεί με τον δείκτη $n$ που παίρνει τιμές 0, 1, 2, …,$N$-1

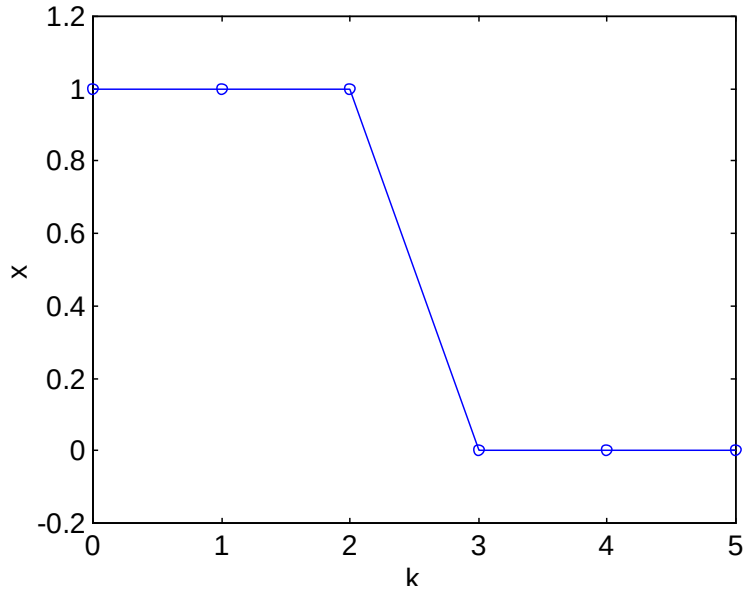
Υπάρχει ο αντίστοιχος αντίστροφος μετασχηματισμός Fourier (***Inverse Discrete Fourier Transform)*** που παίρνει το φάσμα συχνοτήτων και το μετασχηματίζει σε χρονικό σήμα:

$$x_k = \frac{1}{N}\sum_{n=0}^{N-1} X_n e^{\frac{i2\pi nk}{N}}$$

## 8. DFT Παράδειγμα: Ορθογώνιος σφυγμός (pulse) ( $j=\sqrt{-1}$ $j=\sqrt{-1}$)

Θεωρούμε ένα πεπερασμένο σφυγμό :

$x_0 = 1$
$x_1 = 1$
$x_2 = 1$
$x_3 = 0$
$x_4 = 0$
$x_5 = 0$




Για τον υπολογισμό του DFT του σήματος, πρέπει να βρούμε κάθε τιμή του $X$:

$$X_0 = \sum_{k=0}^{5} x_k e^{\frac{-j2\pi 0k}{6}} = \sum_{k=0}^{5} x_k = x_0 + x_1 + x_2 + x_3 + x_4 + x_5 = 3$$

$$X_1 = \sum_{k=0}^{5} x_k e^{\frac{-j2\pi 1k}{6}} = x_0 e^{\frac{-j2\pi 0}{6}} + x_1 e^{\frac{-j2\pi 1}{6}} + x_2 e^{\frac{-j2\pi 2}{6}} = 1 + e^{\frac{-j\pi}{3}} + e^{\frac{-j2\pi}{3}}$$

Μπορούμε να ελαττώσουμε το πλήθος των αλγεβρικών πρέξεων χρησιμοποιώντας μια νέα μεταβλητή . Έστω,

$$z = e^{\frac{-j\pi}{3}} = 0.5 - j0.866$$

$z$ είναι περιοδική με περίοδο 6 (διότι $e^{-j2\pi}$ είναι περιοδική με περίοδο 1). Αυτό μας βοηθά να υπολογίσουμε:

$$z^2 = e^{\frac{-j2\pi}{3}} = -0.5 - j0.866$$
$$z^3 = e^{-j\pi} = 1$$
$$z^4 = z^{-2} = -0.5 + j0.866$$
$$z^5 = z^{-1} = 0.5 + j0.866$$
$$z^6 = z^0 = 1$$
$$z^8 = z^2$$
$$z^{10} = z^4$$

Αντικαθιστώντας τις παραπάνω τιμές στον DFT , λαμβάνουμε:

$$X_1 = 1 + z + z^2 = 1 + 0.5 - j0.866 - 0.5 - j0.866 = 1 - j1.732$$

Ομοίως,

$$X_2 = \sum_{k=0}^{5} x_k e^{\frac{-j2\pi 2k}{6}} = \sum_{k=0}^{5} x_k z^{2k} = x_0 z^0 + x_1 z^2 + x_2 z^4 = z^0 + z^2 + z^4$$

$$= 1 - 0.5 - j0.866 - 0.5 + j0.866 = 0$$

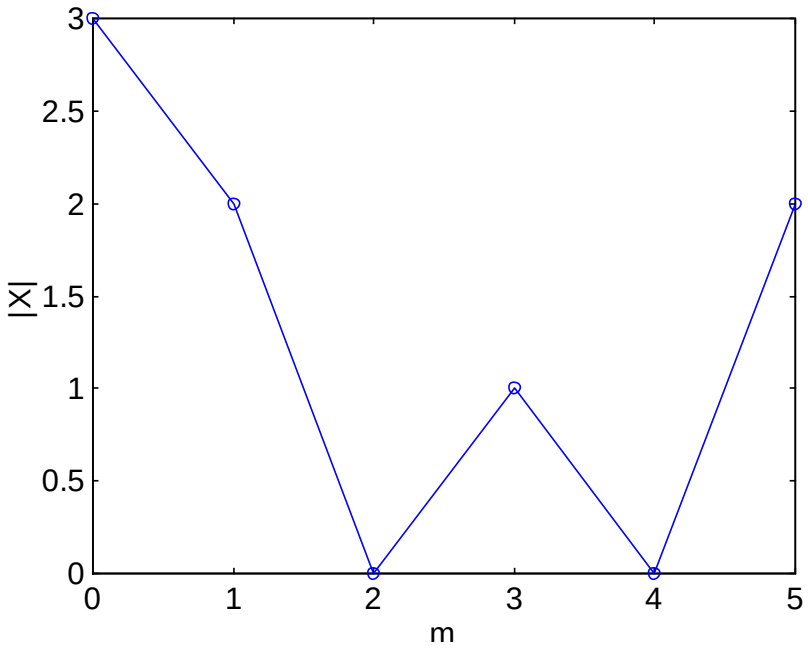
$$X_3 = \sum_{k=0}^{5} x_k z^{3k} = x_0 z^0 + x_1 z^3 + x_2 z^6 = z^0 + z^3 + z^6$$

$$= 1 - 0.5 - j0.866 - 0.5 + j0.866 = 0$$

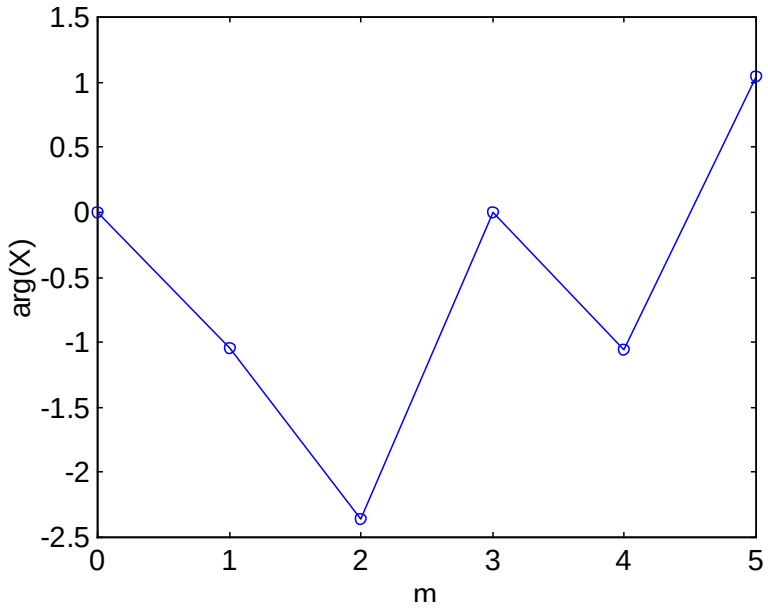
$$X_4 = \sum_{k=0}^{5} x_k z^{4k} = z^0 + z^4 + z^8 = 1 - 0.5 + j0.866 - 0.5 - j0.866 = 0$$

$$X_5 = \sum_{k=0}^{5} x_k z^{5k} = z^0 + z^5 + z^{10} = 1 + 0.5 + j0.866 - 0.5 + j0.866 = 1 + j1.732$$

Τώρα μπορούμε να κάνουμε την γραφική παράσταση του φάσματος:




και την φάση φάσματος:

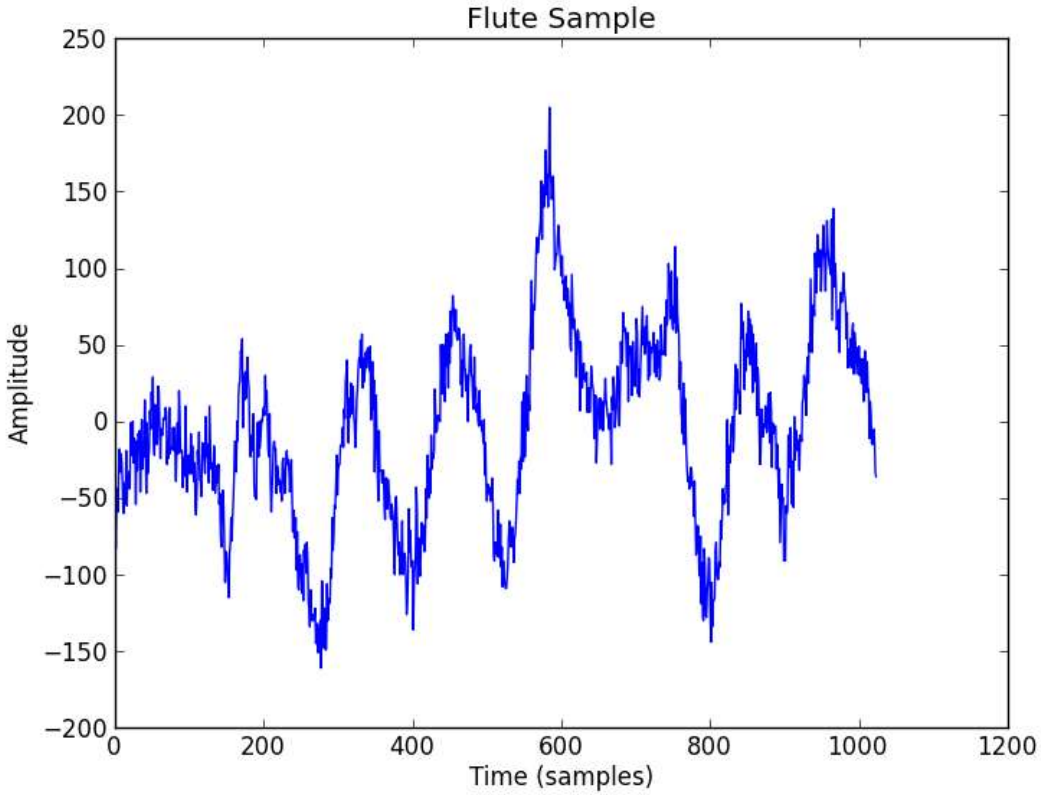



## 9. Ήχος, Θόρυβος, και Φίλτρα

Ο MatLab κώδικας dftfilter.m που παρουσιάστηκε στην παράγραφο 7 είναι ένα παράδειγμα ψηφιακής επεξεργασίας σημάτων. Οι μετασχηματισμοί Fourier χρησιμοποιούνται για να μεταφέρουν τις πληροφορίες ενός σήματος {x0, . . . , xn−1} από το «πεδίο χρόνου» στο «πεδίο συχνοτήτων», όπου μπορούμε να επεξεργαστούμε τη πληροφορία ποιο εύκολα. Μετά την επεξεργασία του σήματος στο πεδίο συχνοτήτων στέλνουμε πίσω το σήμα στο πεδίο χρόνου με την εφαρμογή του αντιστρόφου μετασχηματισμού FFT.

Ο παρακάτω κώδικας Python δείχνει την χρήση της βιβλιοθήκης SciPy για την φόρτωση ενός κομματιού μουσικής (π.χ. φλάουτο) που έχει εγγραφεί στο μορφότυπο wav και την γραφική της παράσταση με την χρήση της βιβλιοθήκης Matplotlib.

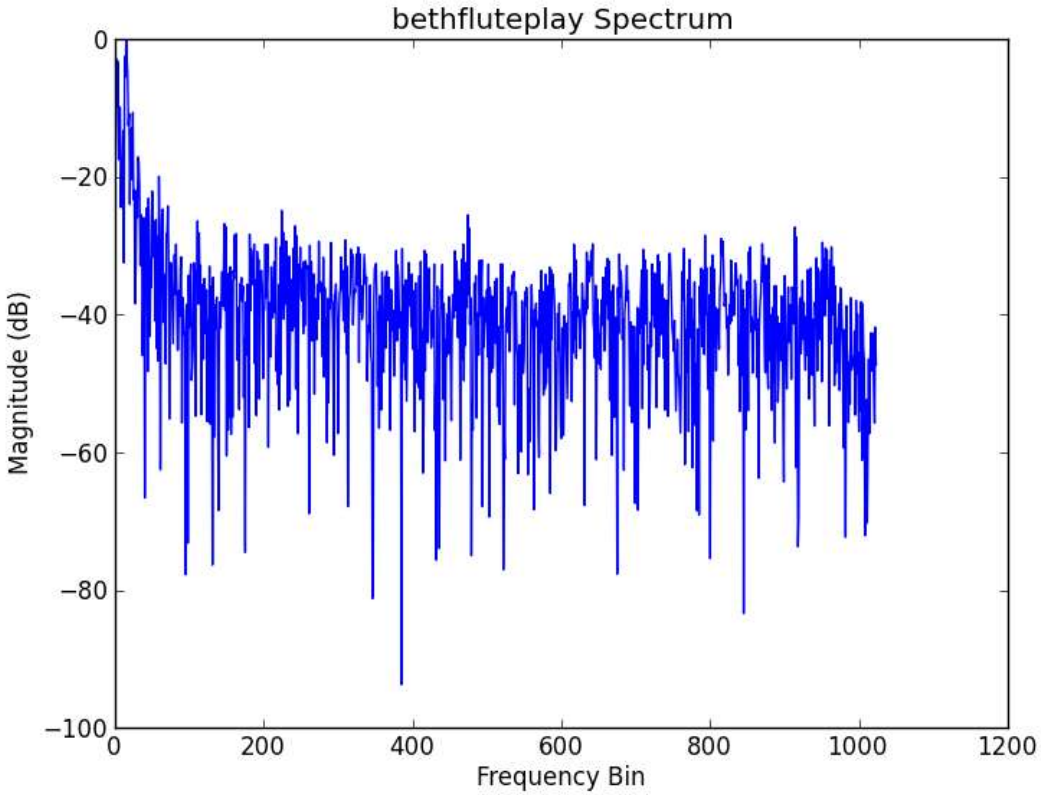
```
from scipy.io.wavfile import read
import matplotlib.pyplot as plt
import numpy, scipy, pylab, random
# read audio samples
input_data = read("bethfluteplay.wav")
audio = input_data[1]
# plot the first 1024 samples
plt.plot(audio[0:1024])
# label the axes
plt.ylabel("Amplitude")
plt.xlabel("Time (samples)")
# set the title
plt.title("Flute Sample")
# display the plot
#plt.show()
```




Ο παρακάτω Python κώδικας χρησιμοποιεί την βιβλιοθήκη SciPy για να εφαρμόσει μετασχηματισμό Fast Fourier Transform (FFT) σε ένα καρέ του ηχητικού σήματος που ορίζεται

από την συνάρτηση Hanning και μετά να παράγει την γραφική παράσταση του φάσματος συχνοτήτων του σήματος. Ο FFT υπολογίζεται στην γραμμή 14, με μιγαδικούς συντελεστές που μετατρέπονται σε πραγματικές συντεταγμένες με τη χρήση της συνάρτησης abs και αποθηκεύονται στην μεταβλητή mags. Το μέγεθος των τιμών μετατρέπεται από τη γραμμική κλίμακα σε ντεσιμπέλ κλίμακα στη γραμμή 16 και κανονικοποιούνται έτσι ώστε να έχει μέγιστη τιμή 0 db στην γραμμή 18. Στην συνέχεια γίνεται γραφική παράσταση των τιμών.

```
import scipy
from scipy.io.wavfile import read
from scipy.signal import hann
from scipy.fftpack import rfft
import matplotlib.pyplot as plt

# read audio samples
input_data = read("flute.wav")
audio = input_data[1]
# apply a Hanning window
window = hann(1024)
audio = audio[0:1024] * window
# fft
mags = abs(rfft(audio))
# convert to dB
mags = 20 * scipy.log10(mags)
# normalise to 0 dB max
mags -= max(mags)
# plot
plt.plot(mags)
# label the axes
plt.ylabel("Magnitude (dB)")
plt.xlabel("Frequency Bin")
# set the title
plt.title("Flute Spectrum")
plt.show()
```



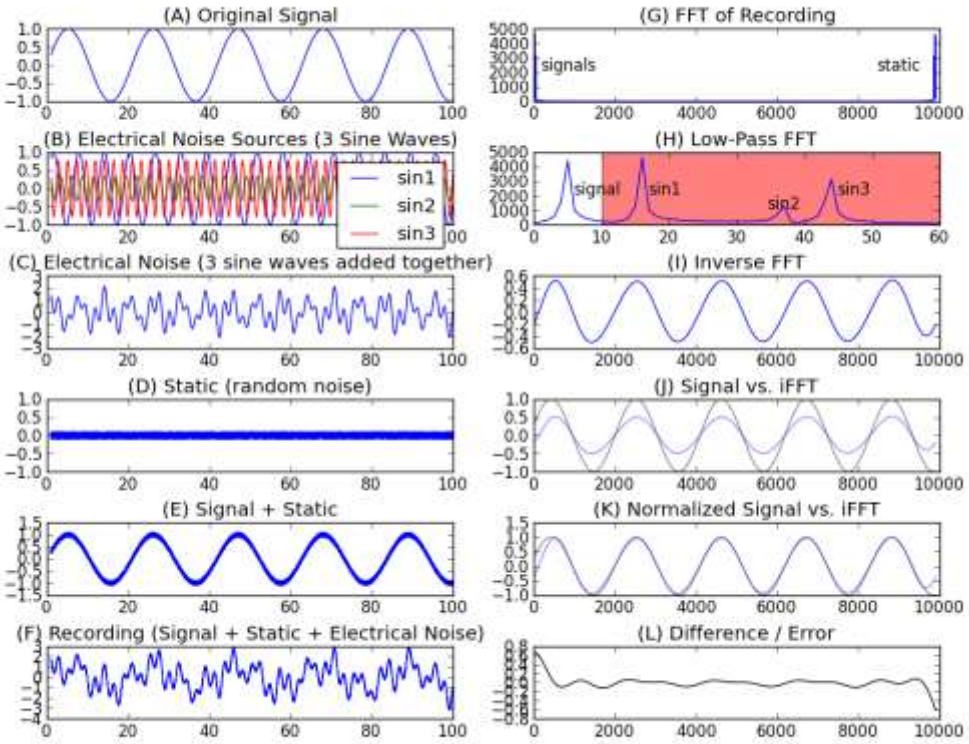

**Παράδειγμα:** Στο παρακάτω σχήμα τυπώνονται οι παρακάτω τυπικές επεξεργασίες ενός σήματος

A. Το Αρχικό σήμα που θέλουμε να επεξεργαστούμε (π.χ. το σήμα των παλμών της καρδιάς μας)
B. Τον ηλεκτρικό θόρυβο (π.χ. 3 sine κύματα διαφορετικών μεγεθών και περιόδων)
C. Περισσότερο Θόρυβο (π.χ. πρόσθεση των 3 sine κυμάτων)
D. Στατικό θόρυβο (π.χ. τυχαίο θόρυβο που παράγεται από κάποια γεννήτρια τυχαίων αριθμών)
E. Σήμα συν στατικό θόρυβος
F. Πρόσθεση σήματος (A) στατικού θορύβου (D) και ηλεκτρικού θορύβου (C)
G. Εφαρμογή FFT στο (F). Σημειώστε ότι χαμηλή συχνότητα μεγεθύνεται κοντά στο 0 λόγο του σήματος και ηλεκτρικού θορύβου και υψηλή συχνότητα κορυφώνεται κοντά στο 10,000 εξαιτίας του στατικού θορύβου.
H. Στο γράφημα αυτό περιοριζόμαστε (ζουμ στην περιοχή (F) ) θέτοντας 0 τις συχνότητες που είναι μεγαλύτερες των 10Hz (red). Αυτό ονομάζεται φίλτρο χαμηλών συχνοτήτων. Το γράφημα δείχνει 4 σημεία αιχμής (peaks), ένα για το αρχικό σήμα και 3 για το υψηλής συχνότητας θόρυβο.

I. Στο γράφημα αυτό βλέπουμε το αποτέλεσμα εφαρμογής του αντιστρόφου FFT (iFFT) στο σήμα που παράγεται με την εφαρμογή του φίλτρου χαμηλών συχνοτήτων που παράγει σχεδόν το αρχικό σήμα.

J. Εδώ συγκρίνεται το σήμα iFFT με το αρχικό και βλέπουμε ότι χρειάζεται κοινωνικοποίηση στο [0,1]. Το αποτέλεσμα της κοινωνικοποίησης είναι το γράφημα K.

K. Εδώ βλέπουμε το σφάλμα της διαφοράς του iFFT και του αρχικού σήματος και διαπιστώνουμε ότι είναι μεγαλύτερο στο άκρα του φάσματος.




## 10. Ιδιότητες Τριγωνομετρικών Συναρτήσεων

$$\sin(a + 2m\pi) = \sin(a)$$
$$\cos(a + 2m\pi) = \cos(a)$$
$$\sin(a + (2m+1)\pi) = -\sin(a)$$
$$\cos(a + (2m+1)\pi) = -\cos(a)$$
$$\sin(a+b) = \sin(a)\cos(b) + \sin(b)\cos(a)$$
$$\sin(a-b) = \sin(a)\cos(b) - \sin(b)\cos(a)$$
$$\sin(a)\cos(b) = \frac{1}{2}(\sin(a+b) + \sin(a-b))$$
$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b)$$
$$\cos(a-b) = \cos(a)\cos(b) + \sin(a)\sin(b)$$

$\cos(a)\cos(b) = \frac{1}{2}(\cos(a+b)+\cos(a-b))$

$\sin(a)\sin(b) = \frac{1}{2}(\cos(a-b)-\cos(a+b))$

**11. <u>Euler's formula</u>**

$e^{it} = \cos(t) + i\sin(t)$ , $i = \sqrt[2]{-1}$

Σύμφωνα με το θεώρημα του Taylor (στο 0)

$\sin(t) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} t^{2k+1}$ , περιττός $f(-t) = -f(t)$

$\cos(t) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} t^{2k}$ , άρτιος $f(-t) = f(t)$

$$e^{it} = 1 + it - \frac{1}{2}t^2 - \frac{1}{6}it^3 + \frac{1}{24}t^4 + \frac{1}{120}it^5 - \frac{1}{720}t^6 - \frac{1}{5040}it^7 + \frac{1}{40320}t^8 + O\left(t^9\right) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k)!} t^{2k} + i\sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} t^{2k+1}$$

- Ο αντίστροφος του $e^{i} = \cos(\theta) + i\sin(\theta)$ είναι $e^{-i} = \cos(\theta) - i\sin(\theta)$ .
- $\bar{z} = a - ib$ ονομάζεται ο συζυγής του $z = a + ib$
- $|z| = |\bar{z}| = |a+ib| = \sqrt[2]{(a+ib)(a-ib)} = \sqrt[2]{a^2+b^2}$
- Ο συζυγής ανάστροφος (conjugate transpose) του μιγαδικού διανύσματος

$$x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \text{ είναι } x^H = \begin{bmatrix} \bar{x}_1 & \bar{x}_2 & . & . & . & \bar{x}_n \end{bmatrix}$$

- Το εσωτερικό γινόμενο δύο μιγαδικών αριθμών ορίζεται ως $\langle x, y \rangle = x^H y$
- Ο συζυγής ανάστροφος (conjugate transpose) ενός μιγαδικού πίνακα $W = \{w_{i,j}\}$ είναι $W^H = \{\bar{w}_{j,i}\}$
- Ένας $nxn$ μιγαδικός πίνακας $U$ είναι μοναδιαίος (unitary) αν $U^H U = UU^H = I$
- Ένας $nxn$ μιγαδικός πίνακας $U$ είναι Hermitian αν $U^H = U$ .

**12. <u>Ορθογώνιες Συναρτήσεις (Orthogonal functions)</u>**

Βάση τριγωνομετρικών πολυωνύμων $\{1, \cos(t), ..., \cos((n-1)t), \sin(t), ..., \sin((n-1)t)\}$

Ισχυρισμός: οι συναρτήσεις της βάσης είναι ορθογώνιες συναρτήσεις στο διάστημα $[-\pi, \pi]$

Υποθέστε $a \neq b$

$\langle 1, \cos at \rangle = \int_{-\pi}^{\pi} \cos at dt = \frac{1}{a}\sin at \,|_{-\pi}^{\pi} = 0$

$\langle 1, \sin at \rangle = \int_{-\pi}^{\pi} \sin at dt = -\frac{1}{a}\cos at \,|_{-\pi}^{\pi} = 0$

$\langle \sin at, \cos bt \rangle = \int_{-\pi}^{\pi} \sin at \cos bt dt = \int_{-\pi}^{\pi} \left(\frac{1}{2}\sin\left(at+bt\right) + \frac{1}{2}\sin\left(at-bt\right)\right) dt = 0$

$$\langle \cos at, \cos bt \rangle = \int_{-\pi}^{\pi} \cos at \cos bt dt = \tfrac{1}{2}\int_{-\pi}^{\pi} (\cos(a+b)t + \cos(a-b)t) dt = 0$$

$$\langle \sin at, \sin bt \rangle = \int_{-\pi}^{\pi} \sin at \sin bt dt = \int_{-\pi}^{\pi} \left( \tfrac{1}{2}\cos\left(at-bt\right) - \tfrac{1}{2}\cos\left(at+bt\right)\right) dt = 0$$

$$\langle 1,1 \rangle = \int_{-\pi}^{\pi} 1 dt = 2\pi$$

Υποθέστε $a = b$

$$\langle \sin at, \cos bt \rangle = \int_{-\pi}^{\pi} \sin at \cos bt dt = \int_{-\pi}^{\pi} \left( \tfrac{1}{2}\sin\left(at+bt\right)\right) dt = 0$$

$$\langle \cos at, \cos bt \rangle = \int_{-\pi}^{\pi} \cos at \cos bt dt = \tfrac{1}{2}\int_{-\pi}^{\pi} (\cos(a+b)t + 1) dt = \pi$$

$$\langle \sin at, \sin bt \rangle = \int_{-\pi}^{\pi} \sin at \sin bt dt = \tfrac{1}{2}\int_{-\pi}^{\pi} \left(1 - \cos\left(at+bt\right)\right) dt = \pi$$

Βάση τριγωνομετρικών πολυωνύμων $\{1, \cos(t), ..., \cos((N-1)t), \sin(t), ..., \sin((N-1)t)\}$ .

Θεωρείστε τα σημεία $t_k = k(2\pi / N)$ , $k = 0,1,..., N-1$ στο διάστημα $[0, 2\pi)$ .

Κάθε διακεκριμένη συνάρτηση μπορεί να θεωρηθεί σαν ένα $Nx1$ διάνυσμα. Για παράδειγμα

$$f_1(\overleftrightarrow{t}) = 1 = \begin{bmatrix} 1 \\ 1 \\ . \\ . \\ . \\ 1 \end{bmatrix}, \quad f_2(\overleftrightarrow{t}) = \cos(\overleftrightarrow{t}) = \begin{bmatrix} \cos(t_0) \\ \cos(t_1) \\ . \\ . \\ . \\ \cos(t_{N-1}) \end{bmatrix} = \begin{bmatrix} \cos(0(2\pi / N)) \\ \cos(1(2\pi / N)) \\ . \\ . \\ . \\ \cos((n-1)2\pi / N) \end{bmatrix}, \ldots$$

Οι συναρτήσεις αυτές είναι ορθογώνιες ως προς τα σημεία $\overleftrightarrow{t} = \{t_k\}$

$$\langle 1,1 \rangle = \sum_{\kappa=0}^{N-1} 1 = N$$

$$\langle 1, \cos(m\overleftrightarrow{t}) \rangle = \sum_{k=0}^{N-1} \cos(mk\tfrac{2\pi}{N-1}) = 0$$

$$\langle 1, \sin(m\overleftrightarrow{t}) \rangle = \sum_{k=0}^{N-1} \sin(mk\tfrac{2\pi}{N-1}) = 0$$

$$\langle \cos n\overleftrightarrow{t}, \cos(m\overleftrightarrow{t}) \rangle = \tfrac{1}{2}\sum_{k=0}^{N-1} [\cos((n+m)k\tfrac{2\pi}{n-1}) + \cos((n-m)k\tfrac{2\pi}{n-1})] = \begin{cases} 0 & n-m \text{ και } n+m \text{ δεν είναι πολλαπλάσια του N} \\ N/2 & n-m \text{ και } n+m \text{ είναι πολλαπλάσια του N} \end{cases}$$

$$\langle \cos n\overleftrightarrow{t}, \sin(m\overleftrightarrow{t}) \rangle = \tfrac{1}{2}\sum_{k=0}^{N-1} [\sin((n+m)k\tfrac{2\pi}{N-1}) - \sin((n-m)k\tfrac{2\pi}{N-1})] = 0$$

$$\langle \sin n\overleftrightarrow{t}, \sin(m\overleftrightarrow{t}) \rangle = \tfrac{1}{2}\sum_{k=0}^{N-1} [\cos((n-m)k\tfrac{2\pi}{n-1}) - \cos((n+m)k\tfrac{2\pi}{N-1}) = \begin{cases} 0 & n-m \text{ και } n+m \text{ δεν είναι πολλαπλάσια του N} \\ N/2 & n-m \text{ και } n+m \text{ είναι πολλαπλάσια του N} \end{cases}$$

**Άσκηση 1:** Υπολογίστε τους πρώτους εφτά όρους της σειράς Taylor για την $f(x) = -\log(1 - x)$. Υπολογίστε τους εφτά πρώτους όρους της σειράς Taylor για την $g(x) = e^x/cos(x) + tan^3(x)$.

**Άσκηση 3:** Υπολογίστε τους συντελεστές $b_0$, $b_1$, $b_3$, $b_5$ για τη συνάρτηση $f(x)$ του προηγούμενου παραδείγματος. Στη συνέχεια χρησιμοποιήστε το Equation 1 για να υπολογίσετε τις $f(0)$, $f(1)$, $f(2)$, $f(3)$, $f(4)$, $f(5)$ και $f(6)$.

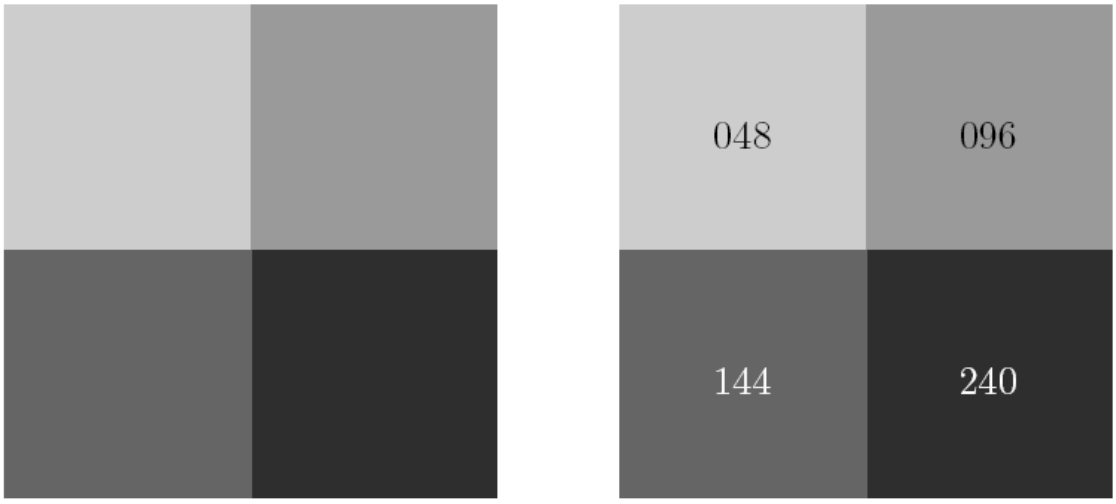
## 13. ΣΗΜΑΝΤΙΚΕΣ ΕΦΑΡΜΟΓΕΣ: ΕΡΓΑΣΙΕΣ

### 13.1 Ψηφιακές εικόνες και DFT σε δύο διαστάσεις

Όπως και τα ηχητικά κύματα ψηφιοποιούνται για να τα χειριστούμε ή να τα αποθηκεύσουμε σε υπολογιστή ή CD και οι εικόνες υπόκεινται σε διαδικασία ψηφιοποίησης για να εμφανιστούν στην οθόνη τουυπολογιστή σας. Για να απλοποιήσουμε τη περιγραφή αυτής της διαδικασίας, θα υποθέσουμε ότι η εικόνα που θέλουμε να ψηφιοποιήσουμε είναι μια ασπρόμαυρη φωτογραφία. Η ιδέα είναι σχετικά απλή: η εικόνα διαιρείται σε μικρά τετράγωνα που ονομάζονται pixels. Σε κάθε pixel μετριέται ο μέσος όρος σκίασης της φωτογραφίας. Κανονικά ένα pixel που είναι απόλυτα λευκό, θα έχει ποσοστό σκίασης 0 και ένα pixel που είναι τελείως μαύρο θα έχει ποσοστό σκίασης 255. Οι αποχρώσεις του γκρι μεταξύ άσπρου και μαύρου λαμβάνουν ακέραιες μετρήσεις μεταξύ 1 και 254, με τα πιο σκούρα γκρι να λαμβάνουν μεγαλύτερες μετρήσεις. Αφού μετρηθεί η σκίαση του κάθε pixel, αυτές οι μετρήσεις καταχωρούνται σε ένα πίνακα. Είναι συνηθισμένο ότι μια εικόνα που είναι $m$ pixels σε πλάτος και $n$ pixels σε ύψος να αποθηκεύεται σε ένα πίνακα $A$, $n \times m$, του οποίου η καταχώρηση στη $i$th σειρά και $j$th στήλη δίνει τη σκίαση του pixel στη $i$th σειρά και $j$th στήλη της διαιρεμένης εικόνας. Θα ονομάσουμε $a_{i,j}$ η καταχώρηση στην $i$th σειρά και $j$th στήλη του $A$.

**Παράδειγμα**:

Στην παρακάτω εικόνα έχουμε μεγεθύνει μια εικόνα η οποία αποτελείται από τέσσερα μόνο pixels ( που έχουν τακτοποιηθεί σε ένα τετράγωνο). Δίνουμε το βαθμό σκίασης για το καθένα από αυτά τα pixel.



Σχήμα 4. Εικόνα σε μορφή Pixel

Ο πίνακας που αντιστοιχεί σε αυτή την εικόνα είναι

$$A=\begin{bmatrix} 48 & 96 \\ 144 & 240 \end{bmatrix}.$$

Η χρήση των ακεραίων από το 0 ως το 255 για να αντιπροσωπεύσουμε τη σκίαση, δεν είναι ένας αυστηρός κανόνας, καθώς οι διαφορετικοί τύποι αρχείων και τα διαφορετικά προγράμματα παρουσίασης εικόνων μπορεί να έχουν διαφορετικές λειτουργίες. Για παράδειγμα, είναι συνηθισμένο να παρουσιάζουμε τη σκίαση ενός pixel με ένα δεκαδικό αριθμό μεταξύ του 0 και 1. Η MatLab μπορεί να δείχνει εικόνες με οποιαδήποτε ευκολία και θα χρησιμοποιήσει αυτή την ικανότητα για κάποιες από τις ασκήσεις μας.

Μια εικόνα $n \times m$ μπορεί να θεωρηθεί ως μια συνάρτηση $f(x, y)$ διακριτού πεδίου ορισμού σε 2 διαστάσεις που ορίζεται από την $f(x, y) = a_{x,y}$ καθώς τιμές του $x$ τρέχουν από το 1 μέχρι το $m$ και τιμές του $y$ τρέχουν από το 1 μέχρι το $n$. Έχοντας αυτό κατά νου, μπορούμε να επιχειρήσουμε να αποσυνθέσουμε μια εικόνα στις συστατικές της «συχνότητες» βρίσκοντας τους συντελεστές $c_{k,l}$ με

$$f(x,y)=\sum_{k=1}^{n}\sum_{l=1}^{m} c_{k,l} e^{i\left(\frac{2\pi kx}{m}+\frac{2\pi iy}{n}\right)} \quad (2)$$

Αυτή είναι η λογική του μετασχηματισμού Fourier που είδαμε παραπάνω μόνο που αντί να έχουμε να κάνουμε με συναρτήσεις διακριτού πεδίου ορισμού μιας μεταβλητής, τώρα έχουμε μια συνάρτηση με διακριτό πεδίο ορισμού αλλά με δύο μεταβλητές. Όπως και στη περίπτωση της μιας μεταβλητής, αυτοί οι συντελεστές υπολογίζονται χρησιμοποιώντας «διακριτά ολοκληρώματα (δηλ. αθροίσματα):

Ο στόχος μας είναι να χρησιμοποιήσουμε αυτή τη διακριτή σειρά Fourier με δυο μεταβλητές για να χειριστούμε εικόνες.

**Παράδειγμα**.

Ας κάνουμε ένα λίγο πιο πολύπλοκο παράδειγμα. Η MatLab θα μας επιτρέψει να εισάγουμε εικόνες από το διαδίκτυο όπως η δημοφιλής εικόνα lena.jpg. Κατεβάστε την στον υπολογιστή σας κάνοντας δεξί κλικ στην εικόνα και επιλέγοντας «αποθηκεύστε την εικόνα ως…». Η MatLab εισάγει την εικόνα χρησιμοποιώντας την εντολή imread. Παρατηρείστε ότι όταν εισάγαμε την παρακάτω εικόνα έχουμε ένα ερωτηματικό στο τέλος της εντολής. Αν δεν συμπεριλάβετε το ερωτηματικό η MatLab θα σας δείξει όλο τον πίνακα που αντιστοιχεί στην εικόνα και αυτό θα είναι πραγματικό πρόβλημα. Γι' αυτό λοιπόν χρησιμοποιείστε το ερωτηματικό!

```
>> Image=imread('C:\Documents and Settings\Public\Desktop\lena.png');
```

Για να δείτε ότι όντως έχετε κατεβάσει την εικόνα χρησιμοποιείστε την εντολή imshow.
```
>> imshow(Image)
```

Στο περιβάλλον Python η βιβλιοθήκη scipy.misc διαθέτει την εικόνα αυτή μαζί με άλλες δημοφιλής (εκτελέστε help(scipy.misc) για να δείτε το περιεχόμενο της.

```
>>>import scipy.misc
>>>lena = scipy.misc.lena()
>>>import matplotlib.pyplot as plt
```

Για να δείτε ότι όντως έχετε κατεβάσει την εικόνα χρησιμοποιείστε την εντολή imshow του Python.
```
>>>plt.imshow(lena)
>>>plt.show()
```
και θα δείξει το Σχήμα 5.


Για να δείτε την εικόνα σε gray μορφή εκτελέστε τις εντολές Python
```
>>>plt.gray()
>>>plt.imshow(lena)
>>>plt.show()
```

Τώρα θα θέλαμε να χρησιμοποιήσουμε τον μετασχηματισμό Fourier για να αλλάξουμε την εικόνα. Για να το κάνουμε αυτό, πρώτα θα πρέπει να υπολογίσουμε τους συντελεστές Fourier όπως στην εξίσωση 2. Φυσικά, είναι πιο εύκολο να υπολογίσουμε τον κάθε συντελεστή ξεχωριστά. Η MatLab και Python υπολογίζουν όλους τους συντελεστές $c_{k,l}$ και τους αποθηκεύουν αμέσως σε έναν πίνακα $n \times m$. Αυτό γίνεται εάν τρέξουμε την εντολή fft2. (Ξανά, παρατηρείστε ότι η εντολή τερματίζεται με ερωτηματικό για να αποφύγουμε να τυπώσουμε ένα τεράστιο πίνακα!)

**MatLab κώδικας**

```
>> J=fft2(Image);
```

(Παρατηρείστε ότι η εντολή MatLab τερματίζεται με ερωτηματικό για να αποφύγουμε να τυπώσουμε ένα τεράστιο πίνακα! Αυτό δεν ισχύει στην γλώσσα Python!)

**Python κώδικας**

```
>>> from scipy import fftpack
>>>J=fftpack.fft2(lena)
```

Ο πίνακας που παράγεται περιέχει όλες τις πληροφορίες για τις συχνότητες της εικόνας σας, έτσι ώστε οι συντελεστές $c_{k,l}$ να μετρούν τη σχετική δύναμη της «συχνότητας» $e^{i(\frac{2\pi kx}{m}+\frac{2\pi iy}{n})}$ της εικόνας σας.

Για παράδειγμα, ας υποθέσουμε ότι παραλείπουμε όλα τα στοιχεία του πίνακα $J$ του οποίου οι $i$ ή $j$ συντεταγμένες είναι τουλάχιστον 50 και αποθηκεύουμε αυτές τις τιμές σε ένα πίνακα που ονομάζουμε Jblur.

**MatLab κώδικας**

```
>> Jblur=J;
>> for i=50:843
for j=50:843
Jblur(i,j)=0;
end
end;
```

Με τι μοιάζει η εικόνα που αντιστοιχεί στο Jblur; Η μαθηματική διαδικασία που μετατρέπει πληροφορίες συχνοτήτων σε εικόνα ονομάζεται αντίστροφος μετασχηματισμός Fourier και είναι απλά η εξίσωση 2. Ευτυχώς, η MatLab έχει ενσωματωμένη συνάρτηση για τον αντίστροφο μετασχηματισμό Fourier η οποία ονομάζεται ifft2.

```
>> Iblur=ifft2(Jblur);
```

Το ίδιο ισχύει και για την Python.

Αυτό δεν είναι το μόνο που πρέπει να κάνουμε για να μπορέσουμε να δούμε το αποτέλεσμα της εικόνας . Θα πρέπει επίσης να σταθμίσουμε τα δεδομένα του Iblur έτσι ώστε η MatLab να γνωρίζει πως πρέπει να δείξει την εικόνα. Αυτό το κάνουμε διαιρώντας το κάθε στοιχείο του Iblur με το μέγεθος του μεγαλυτέρου στοιχείου του Iblur (αυτό σημαίνει ότι τα στοιχεία του Iblur θα είναι τώρα μεταξύ του -1 και του 1).

```
>> Iblur = Iblur/max(max(abs(Iblur)));
```

Τώρα είμαστε έτοιμοι να δούμε την εικόνα. Πληκτρολογώντας

```
>> imshow(Iblur)
```

μας δείχνει την εικόνα

**Άσκηση 4**: Η μέθοδος που χρησιμοποιήσαμε για να θολώσουμε την εικόνα πριν δεν είναι και πολύ εξελιγμένη, μια και απλά παρέλειψε όλες τις συχνότητες πάνω από ένα ορισμένο πεδίο ακόμη και αν αυτές οι συχνότητες συνέβαλλαν σημαντικά. Μια καλύτερη μέθοδος παραλείπει τις «ασήμαντες» συχνότητες. Παραλείπει οποιαδήποτε συχνότητα η οποία είναι σχετικά μικρή σε σύγκριση με τη κυρίαρχη συχνότητα που δίνεται από το $J(1, 1)$. (Παρατηρείστε επίσης ότι με το «σχετικά μικρής» στην προηγούμενη πρόταση, μετράμε το μέγεθος λογαριθμικά. Αυτό συμβαίνει γιατί το μέγεθος των συχνοτήτων γενικά ποικίλει εκθετικά).

```
>> Jblur2=J;
>> for i=1:843
for j=1:843
```

```
if (log(1+J(i,j))<.5*(log(1+J(1,1)))) Jblur2(i,j)=0;
end
end
end;
>> Iblur2=ifft2(Jblur2)/max(max(abs(ifft2(Jblur2))));
>> imshow(Iblur2)
```

Τρέξτε αυτό το κώδικα και δείτε πως επηρεάζει τη δεδομένη εικόνα. Τώρα πειραματιστείτε με το συντελεστή 0.5 που χρησιμοποιείται στην εντολή if. Σε ποια τιμή του συντελεστή έχετε χάσει το περισσότερο μέρος από τις λεπτομέρειες της εικόνας; Τώρα τρέξτε τον παραπάνω κώδικα αλλά αλλάξτε την ανισότητα στην εντολή if από «less than» σε «greater than».

Πειραματιστείτε με το συντελεστή 0.5 και ειδικότερα φέρτε το συντελεστή αρκετά κοντά στο 1. Τι βλέπετε;

Υλοποιείστε την παραπάνω επεξεργασία της εικόνας στην γλώσσα Python.

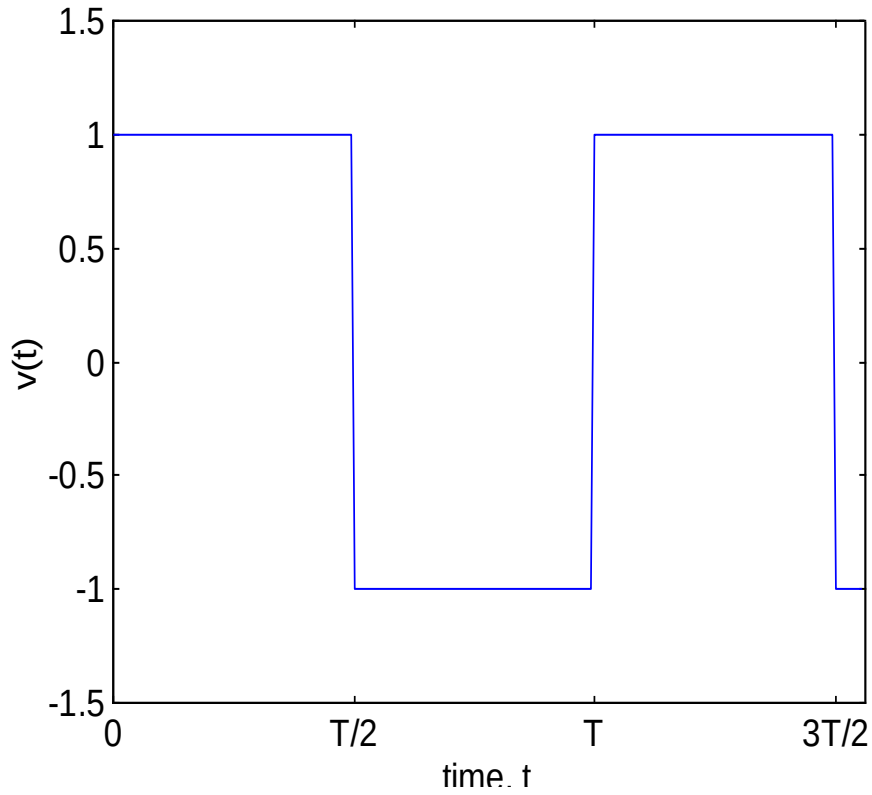
15

15.1

## 13.2 Fourier σύνθεσης ενός τετραγωνικού κύματος

Θεωρείστε το σήμα

$$v(t)=\begin{cases} 1, & 0<t<\dfrac{T}{2} \\ -1, & \dfrac{-T}{2}<t<0 \end{cases}$$



**Να προσδιοριστούν οι συντελεστές της σειράς Fourier**
**Να παρασταθούν γραφικά στο ίδιο γράφημα οι πέντε πρώτοι όροι της σειράς**
**Να παρασταθεί το άθροισμα των πέντε πρώτων όρων μαζί με το σήμα.**
**Να παρασταθεί το άθροισμα των 30 πρώτων όρων μαζί με το σήμα (Fourier σύνθεσης)**
**Να παρασταθεί γραφικά το φάσμα συχνότητας του σήματος, δηλαδή $|b_n|$ ως προς την βασική συχνότητα σε Hz. Υποθέστε ότι η βασική συχνότητα είναι 250 Hz που αυτό σημαίνει η περίοδος T=1/250.**

### 13.3 Ανθρώπινη ακοή και ελάχιστη τετραγωνική τριγωνομετρική προσέγγιση

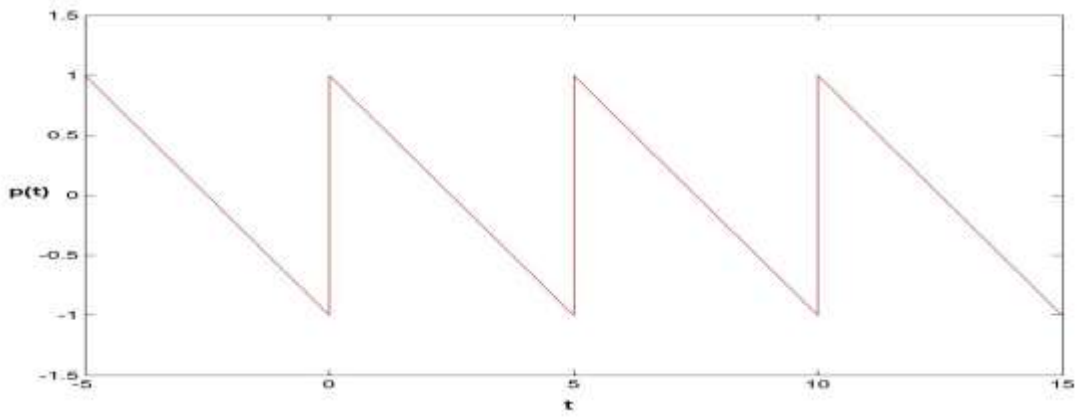
***Μαθηματικά:***

Το ανθρώπινο αυτί αναγνωρίζει ηχητικά κύματα, τα οποία είναι διαφορετικά στο χρόνο και στην ατμοσφαιρική πίεση. Τα ηχητικά κύματα φτάνουν στο αυτί με διαφορετικές συχνότητες και το αυτί ενεργοποιεί σήματα κατά μήκος νευρικών κυττάρων μέχρι τον εγκέφαλο με τις ίσιες συχνότητες. Τα σήματα ερμηνεύονται ως ήχος. Όμως, το ανθρώπινο αυτί αναγνωρίζει τα ηχητικά κύματα που έχουν συχνότητες μεταξύ ενός συγκεκριμένου εύρους, το οποίο είναι περίπου ***από 20 έως 20,000*** κύκλους ανά δευτερόλεπτο. Επομένως, το αυτί μόνο προσεγγίζει τα εισερχόμενα ηχητικά κύματα περικόπτοντας τα σήματα τα οποία έχουν συχνότητες εκτός του εύρους. Η εργασία εκμεταλλεύεται την τριγωνομετρική προσέγγιση για το μοντέλο του ανθρώπινου αυτιού.

Έστω ότι ένα ηχητικό κύμα ***p(t)*** έχει το σχέδιο των δοντιών πριονιού με βασική συχνότητα των ***2000*** κύκλων ανά δευτερόλεπτο. Ένα τέτοιο κύμα είναι περιοδικό στον χρόνο με περίοδο ***T = 1/2000 = 0.0005*** δευτερόλεπτα. Η εξίσωση ***p(t)*** μπορεί να εκφραστεί αναλυτικά:

$$\boldsymbol{p(t) = \frac{2}{T}\left(\frac{T}{2} - t\right)}$$

και γραφικά:




Το αυτί αναγνωρίζει μόνο ημιτονοειδής παραλλαγές της ατμοσφαιρικής πίεσης οι οποίες είναι τριγωνομετρικές συναρτήσεις του χρόνου:

$$\boldsymbol{q(t) = \frac{1}{2}a_0 + \sum_{j=1}^{m-1} a_j \cos\left(\frac{2jt\pi}{T}\right) + b_j \sin\left(\frac{2jt\pi}{T}\right) + a_m \cos\left(\frac{2mt\pi}{T}\right)}$$

όπου $\boldsymbol{(a_j,b_j)}$ είναι τα πλάτη των ημιτονοειδών στοιχείων ενός πολύπλοκου ηχητικού κύματος, καθώς οι συχνότητες των ημιτονοειδών στοιχείων $\boldsymbol{j/T}$ περικόπτονται στον ακέραιο $\boldsymbol{m}$ έτσι ώστε $\boldsymbol{m/T \le 20{,}000}$. Η συνάρτηση $\boldsymbol{q(t)}$ είναι επίσης περιοδική με περίοδο $\boldsymbol{T}$.

Το ηχητικό κύμα $\boldsymbol{p(t)}$ με περίοδο $\boldsymbol{T}$ δεν είναι ένα πεπερασμένο άθροισμα των ημιτονοειδών στοιχείων, ενώ το $\boldsymbol{q(t)}$ είναι. Επομένως, το ηχητικό κύμα $\boldsymbol{p(t)}$ παράγει την απάντηση του αυτιού $\boldsymbol{q(t)}$ το οποίο είναι μόνο η προσέγγιση του ηχητικού κύματος, και δίνεται περικόπτοντας τις συχνότητες που είναι μεγαλύτερες από $\boldsymbol{m/T}$. Η ίδια απάντηση παράγεται εάν το ηχητικό κύμα είναι $\boldsymbol{q(t)}$.

Τα πλάτη $\boldsymbol{(a_j,b_j)}$ της συνάρτησης $\boldsymbol{q(t)}$ μπορούν να βρεθούν από την ελάχιστη-τετραγωνική τριγωνική προσέγγιση:

$$\boldsymbol{a_j = \frac{2}{T}\int_0^T p(t)\cos\left(\frac{2jt\pi}{T}\right)dt, \quad j = 0,1,2,\ldots,m; \qquad b_j = \frac{2}{T}\int_0^T p(t)\sin\left(\frac{2jt\pi}{T}\right)dt, \quad j = 1,2,\ldots,m-1}$$

Η ελάχιστη-τετραγωνική τριγωνική προσέγγιση ελαχιστοποιεί το μέσο τετραγωνικό σφάλμα:

$$\boldsymbol{E = \int_0^T [p(t) - q(t)]^2\, dt}$$

Από το παράδειγμα του $\boldsymbol{p(t)}$, μπορούμε αναλυτικά να βρούμε ότι $\boldsymbol{a_j = 0; \quad b_j = \frac{2}{j\pi}}$.

***Ζητούμενα:***

Να αναπτύξετε έναν υπολογιστικό αλγόριθμο για την ελάχιστη-τετραγωνική τριγωνομετρική προσέγγιση

Να χρησιμοποιήσετε την τριγωνομετρική προσέγγιση για διαφορετικές τιμές του ***m***
Να κατανοήσετε το φαινόμενο του Gibbs το οποίο εμφανίζεται στις ασυνέχειες του τριγωνομετρικού αθροίσματος

***<u>Το MATLAB για την ελάχιστη-τετραγωνική τριγωνομετρική προσέγγιση</u>***

Έστω το διάστημα $t \in [0,T]$ το οποίο χωρίζεται σε $(n+1)$ ίσα σημεία πλέγματος**:**

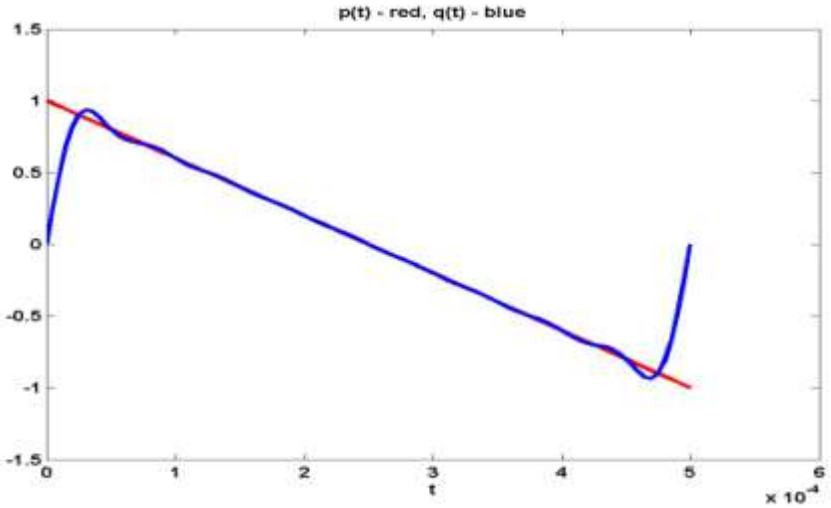
$$t_i = \frac{(i-1)T}{n}, \quad i=1,2,\ldots,n,n+1$$

Έστω $n = 2m$. η τριγωνομετρική προσέγγιση $q(t)$ έχει $2m$ συντελεστές $[a_j,b_j]$, οι οποίοι βρίσκονται από τις εξισώσεις: $p(t_i) = q(t_i)$. Όταν η συνάρτηση $p(t)$ είναι συνεχής στα εξωτερικά σημεία του διαστήματος $[0,T]$, οι συντελεστές $[a_j,b_j]$ μπορούν να υπολογιστούν από τους άμεσους τύπους αθροίσματος:

$$a_j = \frac{2}{n}\sum_{i=1}^{n} p(t_i)\cos\left(\frac{2jt_i\pi}{T}\right), \quad j = 0,1,2,\ldots,m; \qquad b_j = \frac{2}{n}\sum_{i=1}^{n} p(t_i)\sin\left(\frac{2jt_i\pi}{T}\right), \quad j = 1,2,\ldots,m-1$$

Όταν η συνάρτηση $p(t)$ έχει άλμα στα εξωτερικά σημεία του διαστήματος $[0,T]$, οι συντελεστές $[a_j,b_j]$, μπορούν να υπολογιστούν από:

$$a_j = \frac{2}{n}\sum_{i=2}^{n} p(t_i)\cos\left(\frac{2jt_i\pi}{T}\right) + \frac{1}{n}[p(0)+p(L)], \quad j = 0,1,\ldots,m; \quad b_j = \frac{2}{n}\sum_{i=2}^{n} p(t_i)\sin\left(\frac{2jt_i\pi}{T}\right), \quad j = 1,\ldots,m-$$




***<u>Βήματα για τον κώδικα του MATLAB:</u>***
Θέστε $T = 0.0005$, $m = 20000$ $T$, και $n = 2m$.
Διαμερίστε το $[0,T]$ σε ένα διάνυσμα γραμμής $t$ με $(n+1)$ σημεία πλέγματος.
Υπολογίστε ένα διάνυσμα γραμμής $p$ από την συνάρτηση $p(t)$ στα $(n+1)$ σημεία πλέγματος.
Υπολογίστε τους συντελεστές $(a_j,b_j)$ από την τριγωνομετρική προσέγγιση $q(t)$.

Σχεδιάστε την τριγωνομετρική προσέγγιση ***q(t)*** για τον διαμερισμό $t_{int}$ του διαστήματος μεταξύ $0 < t < T$.
Σχεδιάστε την συνάρτηση του ηχητικού κύματος ***p(t)*** στο ίδιο γράφημα.

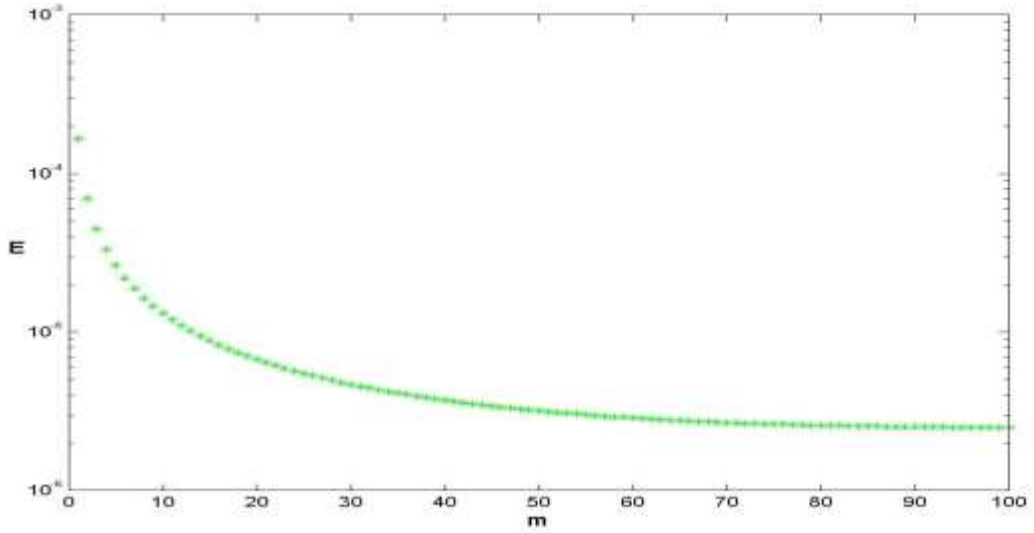
***Επιπλέον χρήση του MATLAB:***
Ελέγξτε το τοπικό σφάλμα της τριγωνομετρικής προσέγγισης $e(t) = |p(t) - q(t)|$ για τις τιμές του $t$ κοντά στα τελικά σημεία $t = 0$ και $t = T$.
Ελέγξτε το τοπικό σφάλμα $e(t) = |p(t) - q(t)|$ για τις τιμές του $t$ κοντά στο μεσαίο σημείο $t = T/2$?

***<u>Το MATLAB για τα σφάλματα των ελαχίστων τετραγωνικών τριγωνομετρικών συναρτήσεων</u>***

Η ελάχιστη τετραγωνική τριγωνομετρική προσέγγιση γίνεται καλύτερη όσο ο αριθμός των όρων στο τριγωνομετρικό προσεγγιστικό πολυώνυμο ***q(t)*** γίνεται μεγαλύτερος. Το μέσο τετραγωνικό σφάλμα ***E*** μπορεί να υπολογιστεί από τον κανόνα του τραπεζίου για την αριθμητική ολοκλήρωση:

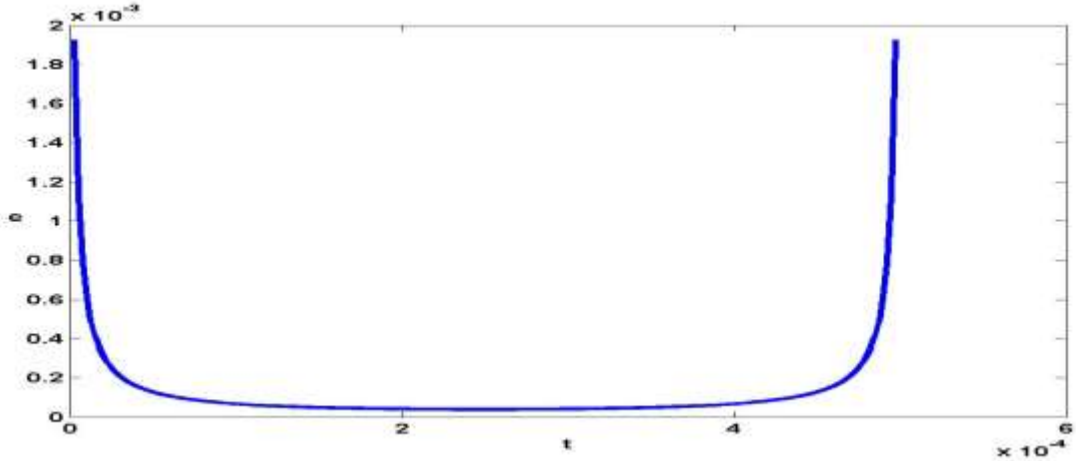
$$E = \frac{h}{2}\sum_{i=1}^{n} \left[(p_i - q_i)^2 + (p_{i+1} - q_{i+1})^2\right],$$

όπου το ***h*** είναι το μέγεθος του βήματος σε μια διαμέριση του διαστήματος ***[0,T].*** Θεωρητικά, το μέσο τετραγωνικό σφάλμα ***E*** τείνει στο μηδέν όσο ο αριθμός ***m*** πλησιάζει στο άπειρο. Όμως, επειδή το ***E*** βρίσκεται προσεγγιστικά από τον κανόνα του τραπεζίου, το μέσο τετραγωνικό σφάλμα ***E*** πλησιάζει στο σφάλμα της αριθμητικής ολοκλήρωσης:




Όμως, η τριγωνομετρική προσέγγιση ***q(t)*** με άπειρους όρους (η σειρά Fourier) μπορεί να μην συγκλίνει στην αρχική συνάρτηση ***p(t)*** σε κάθε σημείο του ***t*** στο ***[0,T].*** Εάν η συνάρτηση ***p(t)*** έχει ένα άλμα ασυνέχειας στο σημείο $t = t_0$, τότε η τριγωνομετρική προσέγγιση ***q(t)*** συγκλίνει σε μία μέση τιμή του ***p(t)*** στο σημείο του άλματος. Για παράδειγμα, το ηχητικό κύμα ***p(t)*** στο δικό μας παράδειγμα έχει ένα άλμα ασυνέχειας στο $t = 0$ και η τριγωνομετρική προσέγγιση ***q(t)*** συγκλίνει στο $0.5( p(0+) + p(0-) ) = 0.5$ *αντι* $(1 - 1) = 0$ στο σημείο $t = 0$. Στα υπόλοιπα σημεία του ***t*** στο ***[0,T],*** η τριγωνομετρική

προσέγγιση ***q(t)*** συγκλίνει στις τιμές του ***p(t),*** αλλά το τοπικό σφάλμα της τριγωνομετρικής προσέγγισης ***e(t)*** δεν είναι ομοιόμορφα μικρό για όλες τις τιμές του ***t,*** εάν η συνάρτηση ***p(t)*** έχει ένα άλμα ασυνέχειας. Η συμπεριφορά του τοπικού σφάλματος ***e(t)*** για αρκετά μεγάλη τιμή ***m = 100*** φαίνεται παρακάτω:




***Βήματα για την συγγραφή του κώδικα του MATLAB:***

1. Ορίστε το ***T*** όπως προηγουμένως. Ορίστε ένα βρόγχο για τις τιμές του ***m*** από το ***1*** έως το ***100.***
2. Έστω ***n = 2 m*** και ορίστε μια ομοιόμορφη διαμέριση του ***[0,T]*** σε ένα διάνυσμα γραμμής ***t*** με ***(n+1)*** σημεία πλέγματος.
3. Υπολογίστε το διάνυσμα γραμμής ***p*** από την συνάρτηση ***p(t)*** στα ***(n+1)*** σημεία πλέγματος.
4. Υπολογίστε τους συντελεστές $(a_j,b_j)$ της τριγωνομετρικής προσέγγισης ***q(t).***
5. Υπολογίστε το μέσο τετραγωνικό σφάλμα ***E*** για κάθε τιμή του ***m***. Σώστε το ***E*** σαν διάνυσμα.
6. Μετά τον τερματισμό του βρόγχου για το ***m***, σχεδιάστε το διάνυσμα ***E*** ως προς το ***m*** σε ημιλογαριθμική κλίμακα.
7. Για ***m = 100***, σχεδιάστε το τοπικό σφάλμα της τριγωνομετρικής προσέγγισης ***e(t)*** = $|p(t) - q(t)|$ για τον κατακερματισμό $t_{int}$ του διαστήματος μεταξύ $0 < t < T$.

***Επιπλέον χρήση του MATLAB:***
Σχεδιάστε το τοπικό σφάλμα ***e(t)*** = $|p(t) - q(t)|$ για ***m = 10, 50, 150, 200***.

***<u>ΕΡΩΤΗΣΕΙΣ:</u>***
Υπολογίστε την τριγωνομετρική προσέγγιση του ***q(t)*** της συνάρτησης ***p(t)*** στο $t \in [0,2\pi]$ για ***m = 10***:

$$p(t) = \frac{3+4\sin t}{5-4\cos t}$$

Υπολογίστε την τριγωνομετρική προσέγγιση ***q(t)*** της συνάρτησης ***p(t)*** στο $t \in [0,2\pi]$ για ***m = 10***:

$$p(t) = e^{\cos t}\,[\cos(\sin t) + \sin(\sin t)]$$

## 14. Βιβλιοθήκη Python για Fourier Transforms (`scipy.fftpack`)

**Περιεχόμενο**



## 15. Αναφορές

1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. http://palmer.wellesley.edu/~aschultz/teaching/math52/
3. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
4. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
5. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
6. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
7. Python For Audio Signal Processing, John GLOVER, Victor LAZZARINI and Joseph TIMONEY, The Sound and Digital Music Research Group National University of Ireland, Maynooth Ireland, http://lac.linuxaudio.org/2011/papers/40.pdf
8. Numerical Analysis, Tim Sauer, Pierson, Second Edition, 2006.

# ΚΕΦΑΛΑΙΟ 7: ΑΡΙΘΜΗΤΙΚΗ ΓΡΑΜΜΙΚΗ ΑΛΓΕΒΡΑ

## Περιεχόμενα



70% of scientific problems involved the solution of linear system

## 1. Προβλήματα Γραμμικής Άλγεβρας

Έστω $A=(a_{i,j}, i, j=1,...,n)$ ένας τετραγωνικός πίνακας διάστασης $n\times n$, και $x=(x_1,...,x_n)^T$, $b=(b_1,...,b_n)^T$ διανύσματα στηλών ή πίνακες $n\times 1$Να υπολογισθούν

- *η λύση του γραμμικού συστήματος* $Ax=b$
- *ο πίνακας X (nxm) που ικανοποιεί την εξίσωση συστημάτων* $AX=B$ *όπου η λύση της εξίσωσης ισοδυναμεί με την λύση του συνόλου των συστημάτων* $Ax_i=b_i, i=1,...,m$
- *ο αντίστροφος του πίνακα* $A$
- *η ορίζουσα του πίνακα* $A$
- *οι νόρμες του* $A$
- *ο δείκτης κατάστασης (conditional number) του A*

## 2. Θεωρία Γραμμικών Συστημάτων

Η λύση του συστήματος γραμμικών εξισώσεων

$$Ax=b \quad (1.1)$$

όπου Α είναι ένας nxn πίνακας και τα διανύσματα $x$, $b$ τάξης $n$ εμφανίζεται σε κάποια φάση της λύσης των 75% από τα επιστημονικά προβλήματα. Σημειώστε, ότι πριν πάμε να λύσουμε ένα πρόβλημα, καλό θα είναι να ελέγχουμε την ύπαρξη λύσης του καθώς και την μοναδικότητά της. Ευτυχώς, η θεωρία των γραμμικών συστημάτων παρέχει μια σειρά από συνθήκες που εγγυώνται την λύση του (1.1).

**Έστω *A* είναι τετραγωνικός πίνακας τάξης** n**. Τότε οι παρακάτω συνθήκες είναι ισοδύναμες:**

1. **Για κάθε** b**, το σύστημα** $Ax=b$ **έχει λύση**
2. **Αν η λύση του συστήματος** $Ax=b$ **υπάρχει, είναι μοναδική**
3. **Για κάθε** $x$, $Ax=0 \Rightarrow x=0$
4. **Οι στήλες (γραμμές) του** $A$ **είναι γραμμικά ανεξάρτητες**
5. **Υπάρχει πίνακας** $A^{-1}$ **έτσι ώστε** $A^{-1}A=AA^{-1}=I$
6. $\det(A)\neq 0$

**Παρατήρηση**: Ο πίνακας *A που ικανοποιεί τις παραπάνω συνθήκες λέγεται αντιστρέψιμος (nonsingular). Το γινόμενο AB είναι αντιστρέψιμο αν και μόνον αν οι πίνακες A και B είναι αντιστρέψιμοι. Αν ο πίνακας* $A^T$ *είναι αντιστρέψιμος τότε και ο (* $A^{-1})^T$ *είναι αντιστρέψιμος.*

## 3. Python παράσταση πινάκων και συστημάτων Ax=b

```
>>> ...
2x1-7x2+4x3=9
x1+9x2-6x3=1
-3x1+8x2+5x3=6
...
>>> from scipy import linalg
>>> mat([[2., -7, 4, 9],[1, 9, -6, 1],[-3, 8, 5, 6]])
>>> A
matrix([[ 2., -7.,  4.,  9.],
```

```
        [ 1.,  9., -6.,  1.],
        [-3.,  8.,  5.,  6.]])
```

**Ένα γραμμικό σύστημα που προκύπτει από την εφαρμογή της μεθόδου παρεμβολής με πολυώνυμα**

**python program**

```
from numpy import zeros, array
from math import *
from scipy import linalg
#defines vandermode matrix for the l
def vandermode(v):
    n = len(v)
    a = zeros((n,n))
    for j in range(n):
        a[:,j] = v**(n-j-1)
    return a
# x- coordinates of interpolation points
v = array([1.0, 1.2, 1.4, 1.6, 1.8, 2.0])
print v
#coefficent matrix
A=vandermode(v)
print A
#right hand side of the system – y coordinates of the
interpolation points
b = array([0.0, 1.0, 0.0, 1.0, 0.0, 1.0])
```

**outpout**

```
v=[ 1.   1.2  1.4  1.6  1.8  2. ]
a=[[  1.         1.         1.         1.         1.         1.
]
 [  2.48832    2.0736     1.728      1.44       1.2        1.
]
 [  5.37824    3.8416     2.744      1.96       1.4        1.
]
 [ 10.48576    6.5536     4.096      2.56       1.6        1.
]
 [ 18.89568   10.4976     5.832      3.24       1.8        1.
]
 [ 32.        16.         8.         4.         2.         1.
]]
>>> b = array([0.0, 1.0, 0.0, 1.0, 0.0, 1.0])
>>> b
array([ 0.,  1.,  0.,  1.,  0.,  1.])
```

**4. Τέστ μοναδικής λύσης του συστήματος $Ax = b$**

Ένα σύστημα $Ax=b$ $n$ γραμμικών εξισώσεων και $n$ αγνώστων έχει μοναδική λύση αν η ορίζουσα $\det(A) \neq 0$. Στην περίπτωση αυτή, οι γραμμές και στήλες του είναι γραμμικά ανεξάρτητες που σημαίνει ότι καμία γραμμή (ή στήλη) δεν είναι γραμμικός συνδυασμός των άλλων γραμμών (ή στηλών). Αν ο πίνακας συντελεστών Α είναι μη-αντιστρέψιμος,

τότε οι εξισώσεις έχουν άπειρο αριθμό λύσεων ή καμία λύση με βάση τίς τιμές των στοιχείων του σταθερού διανύσματος b. Για παράδειγμα, στο παρακάτω σύστημα

$$2x + y = 3$$
$$4x + 2y = 6$$

η δεύτερη εξίσωση είναι πολλαπλάσιο της πρώτης και κάθε συνδυασμός τιμών των x και y που ικανοποιούν την πρώτη εξίσωση είναι επίσης λύση της δεύτερης. Ο αριθμός τέτοιων συνδυασμών είναι άπειροι. Στην περίπτωση των εξισώσεων

$$2x + y = 3$$
$$4x + 2y = 0$$

το σύστημα δεν έχει λύση, διότι η δεύτερη εξίσωση του είναι ισοδύναμη με την εξίσωση 2x + y = 0, και έρχεται σε αντίθεση με την πρώτη εξίσωση. Επομένως, κάθε λύση της πρώτης εξίσωσης δεν ικανοποιεί την άλλη.

## 5. <u>Μέθοδοι λύσης του Ax=b: Απαλοιφή Gauss</u>

Οι άμεσες (\direct) μέθοδοι λύσης του συστήματος $Ax = b$ βασίζονται στο μετασχηματισμό των στοιχείων του πίνακα των συντελεστών A και του σταθερού διανύσματος b. Παρακάτω, περιγράφουμε την μέθοδο απαλοιφής Gauss που την εφαρμόζουμε σε ένα 3x3 σύστημα

$$\begin{aligned} b_{11}x_1 + b_{12}x_2 + b_{13}x_3 &= u_1 \ , \\ b_{21}x_1 + b_{22}x_2 + b_{23}x_3 &= u_2, \\ b_{31}x_1 + b_{32}x_2 + b_{33}x_3 &= u_3 \end{aligned} \qquad (1.1)$$

Σαν πρώτο βήμα, αντικαθιστούμε την δεύτερη εξίσωση με μια εξίσωση που προκύπτει πολλαπλασιάζοντας την *πρώτη εξίσωση με τον παράγοντα* $-b_{21}/b_{11}$ και προσθέτοντας το γινόμενο στην δεύτερη εξίσωση. Παρομοίως, αντικαθιστούμε την τρίτη εξίσωση με την εξίσωση που προκύπτει πολλαπλασιάζοντας την πρώτη εξίσωση με τον παράγοντα $-b_{31}/b_{11}$ και προσθέτοντας το γινόμενο στην τρίτη εξίσωση. Το αποτέλεσμα είναι το σύστημα

$$\begin{aligned} b_{11}x_1 + b_{12}x_2 + b_{13}x_3 &= u_1 \\ b'_{22}x_2 + b'_{23}x_3 &= u'_2, \\ b'_{32}x_2 + b'_{33}x_3 &= u'_3 \end{aligned} \qquad (1.2)$$

στο οποίο $b'$ και $u'$ είναι οι νέοι συντελεστές μετά από τους παραπάνω μετασχηματισμούς. Στην συνέχεια πολλαπλασιάζοντας την δεύτερη εξίσωση (1.2) με $-b'_{32}/b'_{22}$ , και προσθέτοντας αυτή στην τρίτη εξίσωση του (1.2). Το αποτέλεσμα των δύο βημάτων είναι ο τριγωνικός πίνακας

$$\begin{aligned} b_{11}x_1 + b_{12}x_2 + b_{13}x_3 &= u_1 \ , \\ b'_{22}x_2 + b'_{23}x_3 &= u'_2, \\ b''_{33}x_3 &= u''_3 \end{aligned} \qquad (1.3)$$

όπου $b_{33}^{''}$ και $u_3^{''}$ είναι αποτέλεσμα των παραπάνω αριθμητικών πράξεων. Το σύστημα (1.3) μπορεί να λυθεί εύκολα με την μέθοδο της **πίσω-αντικατάστασης (back substitution)**, όπου ο άγνωστος $x_3$ υπολογίζεται από την τρίτη εξίσωση; ο $x_2$ υπολογίζεται από την δεύτερη εξίσωση και το $x_1$ μπορεί να βρεθεί από την πρώτη εξίσωση. Η παραπάνω διαδικασία είναι μια "αποδοτική" μέθοδος λύσης του συστήματος (1.1), του αντιστρόφου ενός πίνακα, της ορίζουσας ενός πίνακα, και άλλων προβλημάτων της γραμμικής άλγεβρας.

## 6. Στοιχειώδεις Πίνακες

Ο παραπάνω μετασχηματισμός ενός γραμμικού συστήματος μπορεί να πραγματοποιηθεί πολλαπλασιάζοντας τον πίνακα A και το διάνυσμα b με τους αποκαλούμενους στοιχειώδεις πίνακες:

*a.* Ένας στοιχειώδης πίνακας πρώτου είδους είναι ένας *nxn* διαγώνιος πίνακας **Q**, που σχηματίζεται αντικαθιστώντας το $i^{th}$ διαγώνιο στοιχείο του μοναδιαίου πίνακα **I** με την μη-μηδενική τιμή **q**. ***Ένα παράδειγμα τέτοιου πίνακα περιγράφεται στο παρακάτω python πρόγραμμα***

```
>>> from scipy import linalg
>>> a=array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])
>>> a
array([[1, 0, 0, 0],
 [0, 1, 0, 0],
 [0, 0, 1, 0],
 [0, 0, 0, 1]])
>>> a[2,2]=100
>>> a
array([[ 1, 0, 0, 0],
 [ 0, 1, 0, 0],
 [ 0, 0, 100, 0],
 [ 0, 0, 0, 1]])
>>> linalg.det(a)
100.0
>>> linalg.inv(a)
array([[ 1.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  1.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.01,  0.  ],
       [ 0.  ,  0.  ,  0.  ,  1.  ]])
```

*b.* Ένας στοιχειώδης πίνακας δευτέρου είδους είναι ένας *nxn* πίνακας **R**, που σχηματίζεται ανταλλάσοντας τις γραμμές **j** και **i** του μοναδιαίου πίνακα **I**. ***Ένα παράδειγμα τέτοιου πίνακα περιγράφεται στο παρακάτω python πρόγραμμα με n=4, i=1, και j=3.***

```
>>> R=mat([[ 0,  0,  1,  0], [0,  1,  0,  0], [1,  0,  0,  0], [0,
0,  0,  1]])
>>> R
matrix([[0, 0, 1, 0],
 [0, 1, 0, 0],
 [1, 0, 0, 0],
 [0, 0, 0, 1]])
>>> dot(R,R)
matrix([[1, 0, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]])
```

*Παρατήρηση:* det(R) =-1, και R είναι ο αντίστροφος του εαυτού του, δηλαδή, RR=I.

*c.* Ένας στοιχειώδης πίνακας τρίτου είδους είναι ένας *nxn* πίνακας **S**, που σχηματίζεται αντικαθιστώντας το στοιχείο **( j, i )** (j ≠i) του μοναδιαίου πίνακα **I**. ***Ένα παράδειγμα τέτοιου πίνακα περιγράφεται στο παρακάτω python πρόγραμμα με n=4, i=3, και j=1.***

```
>>> S=mat([[1,0,0,0],[0,1,0,0],[-100,0,1,0],[0,0,0,1]])
>>> S
matrix([[ 1,  0,  0,  0],
 [ 0,  1,  0,  0],
 [-100,  0,  1,  0],
 [ 0,  0,  0,  1]])
>>> linalg.det(S)
1.0
```

## 7. <u>Μετασχηματισμός του πίνακα Α με στοιχειώδεις πίνακες</u>

Πολλαπλασιασμό από τα αριστερά του pxn πίνακα Α με στοιχειώδη πίνακες. Για παράδειγμα, στο παρακάτω πρόγραμμα σχηματίζουμε τα γινόμενα QA, RA, και SA, με n = 3, i = 2,. j = 3, και p = 4.

a. QA

```
>>> A= mat([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>> A
matrix([[ 1,  2,  3,  4],
 [ 5,  6,  7,  8],
 [ 9, 10, 11, 12]])
>>> Q=mat([[1,0,0],[0,30,0],[0,0,1]])
>>> Q
matrix([[ 1,  0,  0],
 [ 0, 30,  0],
```

```
 [ 0, 0, 1]])
>>> QA=dot(Q,A)
>>> QA
matrix([[ 1, 2, 3, 4],
 [150, 180, 210, 240],
 [ 9, 10, 11, 12]])
```

b. RA

```
>>> R=mat([[1,0,0],[0,0,1],[0,1,0]])
>>> R
matrix([[1, 0, 0],
        [0, 0, 1],
        [0, 1, 0]])
>>> RA=dot(R,A)
>>> RA
matrix([[ 1,  2,  3,  4],
        [ 9, 10, 11, 12],
        [ 5,  6,  7,  8]])
```

c. SA

```
S=mat([[1,0,0],[0,1,2],[0,0,1]])
>>> S
matrix([[1, 0, 0],
        [0, 1, 2],
        [0, 0, 1]])
>>> SA=dot(S,A)
>>> SA
matrix([[ 1,  2,  3,  4],
        [23, 26, 29, 32],
        [ 9, 10, 11, 12]])
```

## 8. Παράδειγμα χρήσης στοιχειωδών πινάκων για την υλοποίηση της μεθόδου απαλοιφής Gauss

```
"'
 2x1-7x2+4x3=9
 x1+9x2-6x3=1
 -3x1+8x2+5x3=6
"'
C=mat([[2., -7, 4, 9, 1, 0, 0],[1, 9, -6, 0, 0, 1,0],[-3,
8, 5, 6, 0, 0, 1]])
print C
# Gauss elimination
```

```
S1=mat([[1, 0, 0],[-C[1,0]/C[0,0], 1, 0],[0, 0, 1]])
print S1
S2=mat([[1, 0, 0],[0, 1, 0],[-C[2,0]/C[0,0], 0, 1]])
print S2
C1=dot(S1,C)
print C1
C2=dot(S2,C1)
print C2
S3=mat([[1, 0, 0],[0, 1, 0],[0, -C2[2,1]/C2[1,1], 1]])
C3=dot(S3,C2)
print C3
S4=mat([[1/C3[0,0],0,0],[0,1/C3[1,1],0],[0,0,1/C3[2,2]]])
print S4
C4=dot(S4,C3)
print C4
OUTPUT
C=[[ 2. -7.  4.  9.  1.  0.  0.]
   [ 1.  9. -6.  0.  0.  1.  0.]
   [-3.  8.  5.  6.  0.  0.  1.]]
 S1=[[ 1.   0.   0. ]
    [-0.5  1.   0. ]
    [ 0.   0.   1. ]]
S2= [[ 1.   0.   0. ]
     [ 0.   1.   0. ]
     [ 1.5  0.   1. ]]
C1=[[  2.   -7.    4.    9.    1.    0.    0. ]
    [  0.   12.5  -8.   -4.5  -0.5   1.    0. ]
    [ -3.    8.    5.    6.    0.    0.    1. ]]
C2=[[  2.   -7.    4.    9.    1.    0.    0. ]
    [  0.   12.5  -8.   -4.5  -0.5   1.    0. ]
    [  0.   -2.5  11.   19.5   1.5   0.    1. ]]
C3=[[  2.   -7.    4.    9.    1.    0.    0. ]
    [  0.   12.5  -8.   -4.5  -0.5   1.    0. ]
    [  0.    0.    9.4  18.6   1.4   0.2   1. ]]
```

## 9. Μέθοδοι επίλυσης συστημάτων Ax=b παραγοντοποιώντας τον πίνακα A

Οι μέθοδοι ανήκουν στην κατηγορία των απευθείας μεθόδων (direct methods) και χρησιμοποιούν πράξεις με στοιχειώδεις πίνακες για τον μετασχηματισμό του $Ax=b$ σε ποιο απλά ισοδύναμα συστήματα. Ο παρακάτω πίνακας δείχνει τρις δημοφιλής μεθόδους.

| Μέθοδς | Αρχική Μορφή | Τελική Μορφή |
|---|---|---|
| Απαλοιφή Gauss | $Ax=b$ | $Ux=c$ |
| LU παραγοντοποίηση | $Ax=b$ | $LUx=b$ |
| Απαλοιφής Gauss-Jordan | $Ax=b$ | $Ix=c$ |

Στο παραπάνω πίνακα U παριστά ένα άνω τριγωνικό πίνακα, L ένα κάτω τριγωνικό και I

παριστά τον μοναδιαίο πίνακα. Ένας 3x3 άνω τριγωνικός έχει την μορφή

$$U = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

και ένας 3x3 κάτω τριγωνικός πίνακας εμφανίζεται στην μορφή

$$L = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & U_{33} \end{bmatrix}$$

## 10. Τριγωνικοί Πίνακες

**Οι τριγωνικοί πίνακες** παίζουν ένα σημαντικό ρόλο στην γραμμική άλγεβρα διότι απλοποιούν πολλούς υπολογισμούς. Για παράδειγμα, θεωρείστε το σύστημα Lx = c,

$$\begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & U_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} L_{11}x_1 \\ L_{21}x_1 + L_{22}x_2 \\ L_{31}x_1 + L_{32}x_2 + U_{33}x_3 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Αν λύσουμε τις εξισώσεις αρχίζοντας από την πρώτη οι υπολογισμοί είναι εύκολοι, αφού κάθε εξίσωση περιλαμβάνει έναν άγνωστο κάθε φορά. Έτσι η λύση είναι:

$x_1 = \left\{ \frac{c_1}{L_{11}} \right\}$ *if* $L_{11} \neq 0$

$L_{21}x_1 + L_{22}x_2 = c_2 \Rightarrow x_2 = (c_2 - L_{21}x_1) / L_{22}$,

$L_{31}x_1 + L_{32}x_2 + U_{33}x_3 = c_3, \Rightarrow x_3 = (c_3 - L_{31}x_1 - L_{32}x_2) / L_{33}$

Η διαδικασία αυτή λέγεται εμπρός αντικατάσταση (**forward substitution)**. Παρομοίως, μπορούμε να λύσουμε το *Ux = c*, που παράγει η απαλοιφή Gauss, αρχίζοντας από την τελευταία εξίσωση. Η διαδικασία αυτή λέγεται πίσω αντικατάσταση (**back substitution**).

## 11. Λύση *Ax=b* με μεθόδους βασισμένους στην παραγοντοποίηση του *A*

Οι εξισώσεις LUx = b, που αντιστοιχούν στην LU παραγοντοποίηση του A, μπορούν να λυθούν γρήγορα αν τις αντικαταστήσουμε με δύο σύνολα εξισώσεων: Ly = b και Ux = y. Τώρα το Ly = b μπορεί να λυθεί με την μέθοδο της εμπρός αντικατάστασης και η λύση του Ux = y μπορεί να υπολογισθεί με πίσω αντικατάσταση. Η εξίσωση Ix = c, που παράγεται από την απαλοιφή Gauss-Jordan, είναι ισοδύναμη με x = c.

## 12. Αλγόριθμος Απαλοιφής του Gauss

*Φάση Απαλοιφής*

Ας δούμε τις εξισώσεις κατά την διάρκεια της φάσης απαλοιφής των αγνώστων. Ας υποθέσουμε ότι οι κ πρώτες γραμμές του A έχουν μετασχηματιστεί σε άνω τριγωνική μορφή. Επομένως, ο επόμενος οδηγός εξίσωσης είναι η *kth* εξίσωση, και όλες οι εξισώσεις κάτω από αυτήν πρέπει να μετασχηματιστούν. Αξίζει να σημειώσουμε ότι τα στοιχεία του πίνακα A μετά από κ βήματα δεν είναι οι αρχικές συντελεστές των

αγνώστων εκτός από την πρώτη εξίσωση αφού αυτά έχουν τροποποιηθεί από την εφαρμογή της μεθόδου απαλοιφής. Για να μετασχηματίσουμε ολόκληρο τον πίνακα A σε άνω τριγωνικό πίνακα, εισάγουμε τους δείκτες *k* και *i*. Ο *k* δείχνει την γραμμή οδήγησης άρα παίρνει τιμές *k = 0,1, ... , n- 1* και ο *i* δείχνει την γραμμή που πρέπει να μετασχηματιστεί άρα παίρνει τιμές *i = k + 1, k + 2, ... , n*. Η φάση απαλοιφής του αλγορίθμου περιγράφεται ως εξής:

```
for k in range(0,n-1):
for i in range(k+1,n):
  if a[i,k] != 0.0:
  lam = a[i,k]/a[k,k]
  a[i,k+l:n] = a[i,k+l:n] - lam*a[k,k+l:n]
  b[i] = b[i] - lam*b[k]
```

*Πίσω Αντικατάσταση Φάση (Back Substitution Phase)*
Ας θεωρήσουμε την φάση πίσω αντικατάστασης όπου $x_n, x_{n-1}, ..., x_{k+1}$ έχουν ήδη βρεθεί, και είμαστε έτοιμοι να υπολογίσουμε τον άγνωστο $x_k$ από την kth εξίσωση. Ο αντίστοιχος αλγόριθμος που υλοποιεί αυτό το βήμα είναι:

```
for k in range(n-1,-1,-1):
  x[k]=(b[k] - dot(a[k,k+l:n] ,x[k+l:n]))/a[k,k]
```

## 13. Python πρόγραμμα για την απαλοιφή Gauss: gaussElimin

Η συνάρτηση gaussElimin συνδυάζει την απαλοιφή και την πίσω αντικατάσταση. Κατά την διάρκεια της πίσω αντικατάστασης το b αντικαθίσταται από το διάνυσμα x. Έτσι η παράμετρος b περιέχει το δεύτερο μέρος του συστήματος σαν input και την λύση σαν output.

```
## module gaussElimin
x = gaussElimin(a,b)
'''
Solves [a]{b}
{x} by Gauss elimination
'''
from numpy import dot
def gaussElimin(a,b):
n = len(b)
# Elimination phase
for k in range(0,n-l):
  for i in range(k+l,n):
    if a[i,k] != 0.0:
    lam = a[i,k]/a[k,k]
    a[i,k+l:n] = a[i,k+l:n] - lam*a[k,k+l:n]
    b[i] = b[i] - lam*b[k]
# Back substitution
```

```
for k in range(n-1,-1,-1):
  b[k] = (b[k] - dot(a[k,k+1:n] ,b[k+1:n]))/a[k,k]
  return b
```

**14. <u>Λύση πολλαπλών συστημάτων</u>**

Πολλές φορές υποχρεωνόμαστε να λύσουμε εξισώσεις $Ax = b$ με διαφορετικά δεύτερα μέρη b. Ας συμβολίζουμε τα σταθερά διανύσματα $b_1, b_2 \ldots b_m$ και τις αντίστοιχες λύσεις με $x_1, x_2, ..., x_m$ . Τότε τα παραπάνω συστήματα μπορούν να γραφούν σε μορφή πίνακα $AX = B$ , όπου $X = [x_1, x_2, ..., x_m]$ $B = [b_1, b_2, ..., b_m]$ είναι $n x m$ πίνακες που οι στήλες των είναι οι λύσεις και τα δεύτερα μέρη αντίστοιχα. Ένας αποδοτικός τρόπος για την λύση αυτών των εξισώσεων είναι να εφαρμόσουμε την μέθοδο απαλοιφής στο διευρυμένο πίνακα [A, B].

**Παράδειγμα**

Ένας $n$ x Vandermode πίνακας A ορίζεται από τα στοιχεία $A_{ij} = v_i^{n-j}, i = 1, 2, ..., n, j = 1, 2, ..., n$ όπου $v_i$ $v$ είναι γνωστά διανύσματα. Να χρησιμοποιηθεί η python συνάρτηση **gaussElimin** για να υπολογισθεί το σύστημα $Ax = b$ , όπου $A$ είναι ένας $6x6$ Van der Monde πίνακας που αντιστοιχεί στο διάνυσμα $v = [1.0, 1.2, 1.4, 1.6, 1.8, 2.0]^T$ και $b = [0, 1, 0, 1, 0, 1]^T$ Υπολογίστε την ακρίβεια της λύσης. (Οι Van der Monde πίνακες είναι κακής κατάστασης (ill-conditioned), θα αναφερθούμε παρακάτω στο φαινόμενο αυτό) .

**Λύση**

**ΚΩΔΙΚΑΣ**

```
from numpy import zeros,array,product, diagonal,dot
from gaussElimin import *
def vandermode(v):
    n=len(v)
    a=zeros((n,n))
    for j in range(n):
        a[:,j] =v**(n-j-1)
    return a
v=array([1.0, 1.2, 1.4, 1.6, 1.8, 2.0])
b=array([0.0, 1.0, 0.0, 1.0, 0.0, 1.0])
a=vandermode(v)
# Save original matrix
# and the constant vector
aOrig = a.copy() # save original matrix
bOrig = b.copy() # save original right side
x = gaussElimin(a,b)
det = product(diagonal(a))
print 'x =\n' ,x
print '\ndet =' ,det
print  '\nCheck  result:  [a]{x}  -  b  =\n',  dot(aOrig,x)  -
bOrig
```

```
raw_input('\nPress return to exit')
```

**OUPUT**

```
x =
[ 416.66666667 -3125.00000004 9250.00000012 -13500.00000017
9709.33333345 -2751.00000003]
det = -1.13246207999e-006
Check result: [a]{x} - b =
[-4.54747351e-13        4.54747351e-13        -1.36424205e-12
4.54747351e-13
-3.41060513e-11 9.54969437e-12]
```

Επειδή η ορίζουσα του A είναι πολύ μικρή σχετικά με τις τιμές των στοιχείων του A, περιμένουμε το σφάλμα μηχανής να επηρεάζει την ακρίβεια των υπολογισμών. Δεδομένο ότι η ακριβής λύση είναι $x = [1250/3, -3125, 9250, 13500, 29128/3, 2755]^T$ και η υπολογιστική λύση ικανοποιεί το σύστημα με ακρίβεια περίπου $10^{-11}$ συμπεραίνουμε ότι η υπολογιστική λύση θα είναι ακριβής στα πρώτα 10 δεκαδικά ψηφία.

## 15. LU Μέθοδοι

Είναι δυνατόν να αποδειχθεί ότι κάθε τετραγωνικός πίνακας A μπορεί να γραφεί σαν γινόμενο ενός κάτω L και άνω U τριγωνικών πινάκων: A=LU. Η διαδικασία υπολογισμού των L και U για κάθε πίνακα A είναι γνωστή σαν LU παραγονοποίηση και δεν είναι μοναδική. Τρεις δημοφιλείς παραγοντοποιήσεις αναφέρονται στο παρακάτω πίνακα.

| Όνομα | Περιορισμοί |
|---|---|
| Doolittle's παραγοντοποίηση | $L_{ii} = 1, i = 1,2,...,n$ |
| Crout's παραγοντοποίηση | $U_{ii} = 1, i = 1,2,...,n$ |
| Choleski's παραγοντοποίηση | $L = U^T$ |

**Doolittle's Αλγόριθμος παραγοντοποίησης**

```
## module LUdecomp
' a = LUdecomp(a).
LU decomposition: [L][U] = [a]. The returned matrix
45 2.3 LU Decomposition Methods
[a] = [L\U] contains [U] in the upper triangle and
the nondiagonal terms of [L] in the lower triangle.
x = LUsolve(a,b).
Solves [L][U]{x} = b, where [a] = [L\U] is the matrix
returned from LUdecomp.
'
from numpy import dot
def LUdecomp(a):
n = len(a)
for k in range(0,n-1):
for i in range(k+1,n):
if a[i,k] != 0.0:
```

```
lam = a [i,k]/a[k,k]
a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
a[i,k] = lam
return a
def LUsolve(a,b):
n = len(a)
for k in range(1,n):
b[k] = b[k] - dot(a[k,0:k],b[0:k])
for k in range(n-1,-1,-1):
b[k] = (b[k] - dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
return b
```

## 16. Η μέθοδος απαλοιφής του Gauss με οδήγηση

Η κλασσική μέθοδος απαλοιφής Gauss και αυτή με οδήγηση διαφέρουν στο ότι στην αρχή κάθε βήματος της απαλοιφής η εναλλαγή γραμμών γίνεται με το παρακάτω κριτήριο.

*Αν υπάρχουν* $n$ *εξισώσεις, τότε υπάρχουν και* $(n-1)$ *βήματα εμπρός απαλοιφής. Στην αρχή του* $k^{th}$ *βήματος της εμπρός απαλοιφής, βρίσκομαι το μέγιστο των*

$$|a_{kk}|, |a_{k+1,k}|, \ldots\ldots\ldots\ldots, |a_{nk}|$$

***Τότε αν το μέγιστο είναι η τιμή*** $|a_{pk}|$ ***στην*** $p^{th}$ ***γραμμή,*** $k \le p \le n$***, τότε εναλλάσσουμε γραμμές*** $p$ ***και*** $k$**.**

Τα άλλα βήματα της απαλοιφής παραμένουν τα ίδια με την κλασσική μέθοδο Gauss**{** XE "Gauss" **}**. Η πίσω αντικατάσταση παραμένει η ίδια.

## 17. Ο αλγόριθμος απαλοιφής του Gauss με οδήγηση

```
for k in range(0,n-1):
# Find row containing element with largest relative size
p = int(argmax(abs(a[k:n,k])/s[k:n])) + k
# If this element is very small, matrix is singular
if abs(a[p,k]) < tol: error.err('Matrix is singular')

# Check whether rows k and p must be interchanged
if p != k:
# Interchange rows if needed
swap.swapRows(b,k,p)
swap.swapRows(s,k,p)
swap.swapRows(a,k,p)
# Proceed with elimination
```

Η Python εντολή **int(argmax(v))** επιστρέφει τον δείκτη του μεγαλυτέρου στοιχείου στο

διάνυσμα v. Οι αλγόριθμοι για την εναλλαγή γραμμών και στηλών περιλαμβάνονται στο module swap που περιγράφεται παρακάτω. Η συνάρτηση **swapRows** εναλλάσσει γραμμές i και j του πίνακα ή διανύσματος v, ενώ **swapCols** εναλλάσσει στήλες i και j του πίνακα.

```
## module swap
' swapRows(v,i,j).
Swaps rows i and j of vector or matrix [v].
swapCols(v,i,j).
Swaps columns i and j of matrix [v].
'
def swapRows(v,i,j):
if len(v.getshape()) == 1:
v[i],v[j] = v[j],v[i]
else:
temp = v[i].copy()
v[i] = v[j]
v[j] = temp
def swapCols(v,i,j):
temp = v[:,j].copy()
v[:,j] = v[:,i]
v[:,i] = temp
```

## 18. gaussPivot αλγόριθμος

Η συνάρτηση gaussPivot εκτελεί την απαλοιφή Gauss με οδήγηση γραμμών. Η μόνη διαφορά με την gaussElimin είναι η εναλλαγή γραμμών (row swapping).

```
## module gaussPivot
' x = gaussPivot(a,b,tol=1.0e-9).
Solves [a]{x} = {b} by Gauss elimination with
scaled row pivoting
'
from numpy import *
import swap
import error
def gaussPivot(a,b,tol=1.0e-9):
n = len(b)
# Set up scale factors
s = zeros((n))
for i in range(n):
     s[i] = max(abs(a[i,:]))
     for k in range(0,n-1):
# Row interchange, if needed
          p = int(argmax(abs(a[k:n,k])/s[k:n])) + k
          if abs(a[p,k]) < tol:
          error.err('Matrix is singular')
          if p != k:
          swap.swapRows(b,k,p)
```

```
            swap.swapRows(s,k,p)
            swap.swapRows(a,k,p)
# Elimination
for i in range(k+1,n):
     if a[i,k] != 0.0:
     lam = a[i,k]/a[k,k]
     a[i,k+1:n] = a [i,k+1:n] - lam*a[k,k+1:n]
     b[i] = b[i] - lam*b[k]
     if abs(a[n-1,n-1]) < tol:
     error.err('Matrix is singular')
# Back substitution
for k in range(n-1,-1,-1):
     b[k] = (b[k] - dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
     return b
```

## 19. LUpivot

Ο αλγόριθμος της απαλοιφής Gauss μπορεί εύκολα να τροποποιηθεί για να υλοποίηση την μέθοδο παραγοντοποίησης του Doolittle's. Η ποιο σημαντική διαφορά είναι ότι πρέπει να κρατάμε τις εναλλαγές των γραμμών κατά την διάρκεια της παραγοντοποίησης.
Στην LUdecomp αυτά τα δεδομένα αποθηκεύονται στην ακολουθία **seq**. Αρχικά η **seq** περιέχει [0, 1, 2, . . .]. Η πληροφορία αυτή χρησιμοποιείται στη φάση λύσης (LUsolve), που εφαρμόζει τις εναλλαγές στο δεύτερο μέρος του συστήματος πριν εφαρμόσουμε εμπρός και πίσω αντικατάσταση.

```
## module LUpivot
''' a,seq = LUdecomp(a,tol=1.0e-9).
LU decomposition of matrix [a] using scaled row pivoting.
The returned matrix [a] = [L\U] contains [U] in the upper
triangle and the nondiagonal terms of [L] in the lower
triangle.
Note that [L][U] is a row-wise permutation of the original
[a];
the permutations are recorded in the vector {seq}.
x = LUsolve(a,b,seq).
Solves [L][U]{x} = {b}, where the matrix [a] = [L\U] and
the
permutation vector {seq} are returned from LUdecomp.
'''
from numarray import argmax,abs,dot,zeros,Float64,array
import swap
import error
def LUdecomp(a,tol=1.0e-9):
n = len(a)
seq = array(range(n))
73 2.5 Pivoting
```

```
# Set up scale factors
s = zeros((n),type=Float64)
for i in range(n):
s[i] = max(abs(a[i,:]))
for k in range(0,n-1):
# Row interchange, if needed
p = int(argmax(abs(a[k:n,k])/s[k:n])) + k
if abs(a[p,k]) < tol:
error.err('Matrix is singular')
if p != k:
swap.swapRows(s,k,p)
swap.swapRows(a,k,p)
swap.swapRows(seq,k,p)
# Elimination
for i in range(k+1,n):
if a[i,k] != 0.0:
lam = a[i,k]/a[k,k]
a[i,k+1:n] = a[i,k+1:n] - lam*a[k,k+1:n]
a[i,k] = lam
return a,seq
def LUsolve(a,b,seq):
n = len(a)
# Rearrange constant vector; store it in [x]
x = b.copy()
for i in range(n):
x[i] = b[seq[i]]
# Solution
for k in range(1,n):
x[k] = x[k] - dot(a[k,0:k],x[0:k])
for k in range(n-1,-1,-1):
x[k] = (x[k] - dot(a[k,k+1:n],x[k+1:n]))/a[k,k]
return x
```

## 20. Συστήματα $Ax = b$ κακής κατάστασης (Ill-Conditioning)

Στην περίπτωση αριθμητικής λύσης συστημάτων, έχουν έννοια τα ερωτήματα: τι συμβαίνει όταν ο πίνακας συντελεστών είναι σχεδόν μη αντιστρέψιμος (singular): δηλαδή όταν η $\det A$ είναι πολύ μικρή; Για να εκτιμήσουμε ότι η ορίζουσα του πίνακα Α είναι «μικρή», θα πρέπει να έχουμε μια τιμή αναφοράς- ως προς ποιο μέγεθος είναι μικρή; Αυτό το σημείο αναφοράς είναι η νόρμα του πίνακα που συμβολίζεται ως $\|A\|$ . Τώρα μπορούμε να πούμε ότι η ορίζουσα είναι μικρή αν

$$\det(A) \le \|A\|$$

Υπάρχουν διάφορες νόρμες για πίνακες όπως

$$\|A\|_1 = \max_{1\le j\le n}\left(\sum_{i=1}^{n}|a_{ij}|\right)$$

$$\|A\|_{\infty} = \max_{1\le i\le n}\left(\sum_{j=1}^{n}|a_{ij}|\right)$$

$$\|A\|_{2} = \left(\sum_{1\le i,j\le n}|a_{ij}|^{2}\right)^{\frac{1}{2}}$$

Ένας τρόπος μέτρησης της κατάστασης του πίνακα είναι ο αριθμός κατάστασης του πίνακα Α που ορίζεται ως

$$cond(A) = \|A\|\|A^{-1}\|$$

Αν αυτός ο αριθμός είναι κοντά στην μονάδα τότε λέμε ότι ο πίνακας Α είναι καλής κατάστασης (well-conditioned). Ο αριθμός κατάστασης αυξάνει με το βαθμό κακής κατάστασης και γίνεται άπειρο για μη αντιστρέψιμους πίνακες. Δυστυχώς, κοστίζει να υπολογίσει κανείς τον αριθμό κατάστασης για μεγάλους πίνακες. Στην πράξη μπορεί να είναι αρκετό να συγκρίνει κανείς την ορίζουσα του πίνακα με το μέγεθος των στοιχείων του πίνακα. Αν οι εξισώσεις είναι κακής κατάστασης, μικρές αλλαγές στους συντελεστές του πίνακα επιφέρει μεγάλες αλλαγές στην λύση. Για παράδειγμα, θεωρούμε το σύστημα

$$2x + y = 3$$
$$2x + 1.001y = 0$$

Που έχει ως λύση $x = 1501.5, Y = -3000$ . Αφού η $\det(A)$ = 2(1.001) - 2(1) = 0.002 είναι πολύ μικρότερη από τους συντελεστές των αγνώστων, οι εξισώσεις είναι κακής κατάστασης Αν αλλάξουμε την δεύτερη εξίσωση σε 2x + 1.002y = 0 και ξανά λύσουμε το σύστημα παίρνουμε σαν λύση $x = 751.5, Y = -1500.$ Σημειώστε ότι 0.1% αλλαγή στο συντελεστή του $y$ παράγει 100% αλλαγή στην λύση! **Αριθμητική λύση συστημάτων κακής κατάσταση (ill-conditioned) δεν μπορούμε να την εμπιστευτούμε. Άρα θα πρέπει να υπολογίζουμε την ορίζουσα του πίνακα μετά την λύση του συστήματος και να την συγκρίνουμε με το μέγιστο στοιχείο του πίνακα σε απόλυτες τιμές.**

## 21. Λογισμικό Python για Γραμμική Άλγεβρα (numpy.linalg)

### 21.1 Γινόμενο Πινάκων και Διανυσμάτων

| | |
|---|---|
| **dot**(a, b[, out]) | Dot product of two arrays. |
| **vdot**(a, b) | Return the dot product of two vectors. |
| **inner**(a, b) | Inner product of two arrays. |
| **outer**(a, b) | Compute the outer product of two vectors. |
| **tensordot**(a, b[, axes]) | Compute tensor dot product along specified axes for arrays >= 1-D. |
| **einsum**(subscripts, *operands[, out, dtype, ...]) | Evaluates the Einstein summation convention on the operands. |
| **linalg.matrix_power**(M, n) | Raise a square matrix to the (integer) power ***n***. |
| **kron**(a, b) | Kronecker product of two arrays. |

### 21.2 Παραγοντοποίηση Πινάκων

| | |
|---|---|
| **linalg.cholesky**(a) | Cholesky decomposition. |
| **linalg.qr**(a[, mode]) | Compute the qr factorization of a matrix. |
| **linalg.svd**(a[, full_matrices, compute_uv]) | Singular Value Decomposition. |

### 21.3 Ιδιοτιμές και ιδιοδιανύσματα Πινάκων

| | |
|---|---|
| **linalg.eig**(a) | Compute the eigenvalues and right eigenvectors of a square array. |
| **linalg.eigh**(a[, UPLO]) | Return the eigenvalues and eigenvectors of a Hermitian or symmetric matrix. |
| **linalg.eigvals**(a) | Compute the eigenvalues of a general matrix. |
| **linalg.eigvalsh**(a[, UPLO]) | Compute the eigenvalues of a Hermitian or real symmetric matrix. |

### 21.4 Νόρμες και αριθμός κατάστασης πινάκων

| | |
|---|---|
| **linalg.norm**(x[, ord]) | Matrix or vector norm. |
| **linalg.cond**(x[, p]) | Compute the condition number of a matrix. |
| **linalg.det**(a) | Compute the determinant of an array. |
| **linalg.slogdet**(a) | Compute the sign and (natural) logarithm of the determinant of an array. |
| **trace**(a[, offset, axis1, axis2, dtype, out]) | Return the sum along diagonals of the array. |

### 21.5 Λύση γραμμικών εξισώσεων και αντίστροφοι πίνακες

| | |
|---|---|
| **linalg.solve**(a, b) | Solve a linear matrix equation, or system of linear scalar equations. |
| **linalg.tensorsolve**(a, b[, axes]) | Solve the tensor equation `a x = b` for x. |
| **linalg.lstsq**(a, b[, rcond]) | Return the least-squares solution to a linear matrix equation. |
| **linalg.inv**(a) | Compute the (multiplicative) inverse of a matrix. |
| **linalg.pinv**(a[, rcond]) | Compute the (Moore-Penrose) pseudo-inverse of a matrix. |
| **linalg.tensorinv**(a[, ind]) | Compute the ‘inverse’ of an N-dimensional array. |

## 22. Αναφορές

1. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.

2. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
3. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.

# ΚΕΦΑΛΑΙΟ 8: ΛΥΣΗ ΥΠΕΡ (ΥΠΟ) ΠΡΟΣΔΙΟΡΙΣΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ ΜΕ QR

## Περιεχόμενα



### 1. Πρόβλημα υπερ-προσδιορισμένων συστημάτων

Δοθέντος ενός πίνακα A διαστάσεων $mxn$ όπου $m \geq n$ , και ενός $mx1$ διανύσματος $b$, να βρεθεί διάνυσμα $x$ έτσι ώστε το $Ax$ να είναι η καλλίτερη προσέγγιση το $b$ ·

Σημειώστε ότι το σύστημα $Ax = b$ είναι υπερ-προσδιορισμένο, έτσι δεν μπορούμε να προσδιορίσουμε την ακριβή λύση παρά μόνον με προσεγγιστικές μεθόδους και γιαυτό υπάρχουν πολλοί τρόποι. Εμείς επιλέγουμε να μελετήσουμε και να εφαρμόσουμε την μέθοδο των **ελαχίστων τετραγώνων** για την λύση τέτοιων συστημάτων.

### 2. Μέθοδος Ελαχίστων Τετραγώνων

Η λύση των ελαχίστων τετραγώνων του υπερπροσδιορισμένου συστήματος $Ax = b$ είναι το διάνυσμα $x$ που ελαχιστοποιεί το Ευκλείδια μήκος του υπολοίπου διανύσματος $r = Ax - b$ , δηλαδή $\|r\|_2 = (r^T r)^{1/2}$

### 3. Θεωρία επίλυσης υπερ-προσδιορισμένων συστημάτων

Δοθέντος ενός πίνακα A διαστάσεων $mxn$ όπου $m \geq n$, και ενός $mx1$ διανύσματος $b$ και $x$ ικανοποιεί την σχέση $A^T(b - Ax) = 0,$ τότε για κάθε διάνυσμα $y$ ισχύει $\|b - Ax\|_2 \leq \|b - Ay\|_2$ . Δηλαδή, η $x$ είναι η καλλίτερη προσέγγιση της λύσης ως προς την Εκλείδια νόρμα.

**Άσκηση 1:** Να αποδειχθεί το παραπάνω θεώρημα.

4. **Κανονικές Εξισώσεις**

Από την σχέση $A^T(b-Ax)=0$ συμπεραίνουμε ότι

α) το $nxn$ σύστημα $A^TAx=A^Tb$, που αναφέρονται ως οι κανονικές εξισώσεις, προσδιορίζει την λύση των ελαχίστων τετραγώνων του υπερπροσδιορισμένου συστήματος,

β) ο πίνακας $A^TA$ είναι συμμετρικός (Γιατί;) ,

γ) ο πίνακας $A^TA$ είναι αντιστρέψιμος εάν και μόνον εάν οι στήλες του $A$ είναι γραμμικά ανεξάρτητες (Γιατί;),

δ) $(Az)^T(b-Ax)=0$ δηλαδή $r=b-Ax$ είναι κάθετο σε όλα τα διανύσματα του χώρου των στηλών του $A$ που συμβολίζεται ως $R(A)$ . Σημειώστε, ότι η λύση των ελαχίστων τετραγώνων χωρίζει το διάνυσμα $b$ σε δύο διανύσματα $b=Ax+r$ όπου $Ax$ είναι η κάθετη προβολή του $b$ στο $R(A)$ και $r$ είναι ορθογώνια στο $R(A)$ ,

ε) Αν οι στήλες του $A$ είναι γραμμικά ανεξάρτητες τότε η λύση των ελαχίστων τετραγώνων υπάρχει και είναι μοναδική. Επιπλέον έχουμε $x=A^+b=(A^TA)^{-1}Ab$ και ο πίνακας $A^+=(A^TA)^{-1}A$ λέγεται ψευδοαντίστροφος του $A$ .

5. **Κατάσταση των κανονικών εξισώσεων**

Κάθε *x* που ικανοποιεί τις $A^TA\,x = A^Tb$ είναι η ΕΤ λύση του $Ax = b$. Όταν *A* είναι πλήρους τάξης (full column rank), $A^TA$ είναι συμμετρικός θετικά ορισμένος (positive definite) πίνακας και μπορεί να παραγοντοποιηθεί σύμφωνα με την μέθοδο Cholesky. Επιπλέον, ο σχηματισμός του γινομένου $A^TA$ μπορεί να γίνει αιτία σοβαρών επιπτώσεων του σφάλματος στρογγύλευσης. Επίσης, $k(A^TA)=k(A^2)$ $\kappa(A^TA) = \kappa(A^2)$, έτσι για παράδειγμα ένας πίνακας Α με

αριθμό κατάστασης $10^8$ συνεπάγεται ότι οι κανονικές εξισώσεις έχουν αριθμό κατάστασης $10^{16}$. Ο αριθμός πράξεων που απαιτούνται για να παραχθούν οι κανονικές εξισώσεις και η παραγοντοποίηση Cholesky είναι περίπου $n^2(m + n/3)/2$ flops.

**Άσκηση: Δίδονται** $A\in C^{mxn}(m\ge n)$ και $b\in C^m$**. Έστω το διάνυσμα** $x\in C^m$ **ελαχιστοποιεί την νόρμα του σφάλματος των ελαχίστων τετραγώνων** $\|r\|_2=\|b-Ax\|_2$ **όπου** $\|r\|_2=\sqrt{r^2}$ **. Ποιες από παρακάτω σχέσεις ισχύουν**

α) $A^Tr=0$

β) r κάθετος στο range(A)

γ) $grad(r^Tr)=0$

δ) $2x^TA^TA-2bA^T=0$ ή $A^TAx=A^Tb$

ε) Το πρόβλημα ελαχίστων τετραγώνων (ΕΤ) έχει λύση και είναι μοναδική αν Α είναι πλήρης τάξης (rank) (οι στήλες του Α είναι γραμμικά ανεξάρτητες)

ζ) $Pb = Ax$ όπου $\mathrm{P} \in \mathrm{C}^{mxm}$ είναι μία ορθογώνια προβολή στο range(A). Σημειώστε ότι ένας πίνακας είναι ορθογώνια προβολή αν ισχύει $P = P^T$ και $P^2 = I$ .

**6. <u>Πώς θα λύσουμε το $Pb = Ax$ ; Πως βρίσκουμε το x;</u>**

α) Είναι η διαφορά των διανυσμάτων ***b*** και ***Pb***, $Pb - b$ κάθετος στο **range(A);** Γιατί;
β) Ισχύει ότι για κάθε ***x***, τα διανύσματα ***Ax*** και ***(Pb − b)*** είναι ορθογώνια;
Για να βρούμε το εσωτερικό γινόμενο των παραπάνω διανυσμάτων βρίσκομαι το γινόμενο $\boldsymbol{(Ax)^T(Pb - b) = 0}$. Για δύο πίνακες $A^T$ και $B^T$, ισχύει $(AB)^T = B^T A^T$. Οπότε η προηγούμενη σχέση μπορεί να γραφτεί $(Ax)^T(Pb - b) = x^T A^T(Pb - b) = 0$. Η σχέση αυτή μπορεί να ισχύσει για κάθε ***x*** αν $\boldsymbol{A^T(Pb-b) = 0}$ και $\boldsymbol{A^T Ax = A^T b}$. Σημειώστε όταν οι στήλες του $A$ είναι γραμμικά ανεξάρτητες τότε $\boldsymbol{A^T A}$ έχει αντίστροφο και $x = (A^T A)^{-1} A^T \mathrm{b}$ και $\mathrm{A}^+ \equiv (A^T A)^{-1} A^T$ .

**Παράδειγμα 1**: Να βρεθεί το πολυώνυμο δευτέρου βαθμού ( $p(x) = a + bx + cx^2$ ) που προσεγγίζει τα σημεία $\begin{bmatrix} x & -1 & -0.5 & 0 & 0.5 & 1 \\ y & 1 & 0.5 & 0 & 0.5 & 2 \end{bmatrix}$

Η εύρεση του πολυωνύμου (δηλαδή να βρεθούν οι συντελεστές - $a, b, c$ ) έτσι ώστε $p(x)$ να είναι ποιο κοντά στα δεδομένα σημεία μπορεί να θεωρηθεί σαν η λύση των εξισώσεων $\{p(x_i) = a + bx_i + cx_i^2 = y_i\}_{=1}^5$ που οδηγεί σε ένα υπερπροσδιορισμένο σύστημα. Το σύστημα αυτό για τα συγκεκριμένα σημεία μπορεί να γραφεί στη μορφή.

$$\begin{bmatrix} 1 & -1.0 & 1.0 \\ 1 & -0.5 & 0.25 \\ 1 & 0.0 & 0.0 \\ 1 & 0.5 & 0.25 \\ 1 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1.0 \\ 0.5 \\ 0.0 \\ 0.5 \\ 2.0 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & -1 & .0 \\ 1 & -0 & .5 \\ 1 & 0 & .0 \\ 1 & 0 & .5 \\ 1 & 1 & .0 \end{bmatrix}, \quad A^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1.0 & -0.5 & 0 & 0.5 & 1.0 \\ 1.0 & 0.25 & 0 & 0.25 & 1.0 \end{bmatrix},$$

$$A^T A = \begin{bmatrix} 5 & 0 & 2.5 \\ 0 & 2.5 & 0 \\ 2.5 & 0 & 2.125 \end{bmatrix}, \quad b = \begin{bmatrix} 1.0 \\ 0.5 \\ 0.0 \\ 0.5 \\ 2.0 \end{bmatrix}, \quad A^T b = \begin{bmatrix} 4.0 \\ 1.0 \\ 3.25 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 0 & 2.5 \\ 0 & 2.5 & 0 \\ 2.5 & 0 & 2.125 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 4.0 \\ 1.0 \\ 3.25 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 8.5693 \times 10^{-2} \\ 0.4 \\ 1.4287 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 0 & 2.5 \\ 0 & 2.5 & 0 \\ 2.5 & 0 & 2.125 \end{bmatrix} \begin{bmatrix} 8.5693 \times 10^{-2} \\ 0.4 \\ 1.4287 \end{bmatrix} = \begin{bmatrix} 4.0002 \\ 1.0 \\ 3.2502 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1.0 & 1.0 \\ 1 & -0.5 & 0.25 \\ 1 & 0.0 & 0.0 \\ 1 & 0.5 & 0.25 \\ 1 & 1.0 & 1.0 \end{bmatrix}$$

$\begin{bmatrix} 5 & 0 & 2.5 \\ 0 & 2.5 & 0 \\ 2.5 & 0 & 2.125 \end{bmatrix}$, αριθμός κατάστασης: 9.4983

## 7. Επίλυση γραμμικών συστημάτων με παραγοντοποίηση QR

«Η ιδέα της QR παραγοντοποίησης είναι η ποιο σημαντική στην αριθμητική άλγεβρα» (N. Trefethen).

Μέχρις εδώ, μάθαμε δύο μεθόδους για την επίλυση συστημάτων γραμμικών εξισώσεων, **Ax=b**:

1. Πολλαπλασιασμός με τον ανάστροφο: $\mathbf{x}=\mathbf{A}^T\mathbf{b}$, εάν ο **A** έχει **ορθογώνιες στήλες**.
2. Γκαουσιανή απαλοιφή για γενικούς αντιστρέψιμους πίνακες, όπου ο **A** μετατρέπεται σε άνω τριγωνικό πίνακα **U** ή παραγοντοποιείται σε **LU** όπου **L** και **U** είναι κάτω και άνω τριγωνικός.

Η πρώτη μέθοδος είναι πολύ απλή, αλλά είναι εφαρμόσιμη μόνο για ειδικούς (ορθογώνιους) πίνακες. Η Γκαουσιανή απαλοιφή είναι γενική αλλά έχει δύο μεγάλα μειονεκτήματα:

- Μπορεί να είναι «ασταθής αριθμητικά» για ορισμένους πίνακες
- Η ανάλυση σφάλματος για τον LU-αλγόριθμο είναι αρκετά δύσκολη.

Παρόλα τα μειονεκτήματα, η Γκαουσιανή απαλοιφή χρησιμοποιείται εδώ και 200 χρόνια. Μόνο με την ανάπτυξη των ψηφιακών υπολογιστών, εμφανίστηκε ο τρίτος αλγόριθμος ο οποίος συνδυάζει τα πλεονεκτήματα από τις δύο προγενέστερους μεθόδους: παραγοντοποιεί έναν γενικό πίνακα **A** σε ένα γινόμενο ενός ορθογώνιου πίνακα και ενός άνω-τριγωνικού πίνακα: **A=QR.**

**Άσκηση 2:**

Στο MATLAB, η παραγοντοποίηση QR ενός πίνακα **A** υπολογίζεται με την συνάρτηση [Q,R]=qr(A).Κατεβάστε την συνάρτηση solveLinearSystem.m από το link. Η συνάρτηση επιλύει το σύστημα **Ax=b** χρησιμοποιώντας μια από τις 2 μεθόδους που βασίζονται είτε στην παραγοντοποίηση LU- είτε στην παραγοντοποίηση QR. Αυτές οι μέθοδοι ορίζονται σε ένα μπλοκ «switch» που δεν έχει υλοποιηθεί ακόμη. Η Άσκηση σας είναι να:

- Υλοποιήσετε τις δύο μεθόδους.
- Κατασκευάσετε μερικά τυχαία γραμμικά συστήματα και συγκρίνετε τις λύσεις που παράγονται από τις μεθόδους.
- Ελέγξετε τις μεθόδους σε ειδικούς πίνακες *60-επί-60*.

## 8. QR-θεωρία

Δοθέντος ενός πίνακα A διαστάσεων *mxn* όπου $m \geq n$ και έχει γραμμικά ανεξάρτητες στήλες. Τότε υπάρχει ένας μοναδικός *mxn* πίνακας **Q** με $\mathbf{Q^TQ=D}$, **D**=diag($d_1,\ldots,d_n$), $d_k>0$, k=1,…,n και ένας μοναδικός άνω τριγωνικός πίνακας **R** με $\mathbf{r_{kk}}$=1, k=1,2,…,n έτσι ώστε **A=QR**

## 9. Λύση Ελαχίστων Τετραγώνων (ΕΤ) με την μέθοδο QR

Η σχέση $A^T(b-Ax)=0$ μπορεί να γραφεί $R^TQ^T(b-Ax)=0$. Επιπλέον, έχουμε $\mathbf{Q^TA=Q^TQR=DR}$ και **R** είναι αντιστρέψιμο. Σημειώστε ότι $R^TQ^Tb=R^TQ^TAx=R^TDRx$ και $Rx=D^{-1}Q^Tb=y$. Άρα η λύση x βρίσκετε από την λύση του άνω τριγωνικού συστήματος $Rx=y$.

## 10. Gram-Schmidt ορθογωνοποίσης βάσης διανυσματικού χώρου

Δοθέντος ενός συνόλου γραμμικών ανεξαρτήρτων διανυσμάτων $\{\mathbf{v_1,v_2,\ldots,v_n}\}$ να βρεθεί ένα ορθογώνιο σύνολο $\{\mathbf{w_1,w_2,\ldots,w_n}\}$ με την μέθοδο του Gram-Schmidt**.** Υπάρχουν δύο (μαθηματικά ισοδύναμες, αλλά αριθμητικά διαφορετικές) υλοποιήσεις της διαδικασίας ορθογωνοποίησης του Gram-Schmidt: κλασική και τροποποιημένη

Η κλασική διαδικασία είναι η λεγόμενη Gram-Schmidt που ξεκινά με το να θεωρήσουμε $\mathbf{w_1=v_1}$, στην συνέχεια ορίζουμε $\mathbf{w_2=v_2}+s_{21}\mathbf{w_1}$ και απαιτούμε να είναι ορθογώνιο ως προς $\mathbf{w_1}$, δηλαδή $\mathbf{(w_1,w_2)=0=(w_1,v_2)}+s_{21}\mathbf{(w_1,w_1)}$, το οποίο ισχύει αν $s_{21}=\frac{\langle w1,w2\rangle}{\langle w1,w1\rangle}$ $s_{21}=\frac{(\mathbf{w1,v2})}{(\mathbf{w1,w1})}$. Αυτό ισοδυναμεί με το να προβάλουμε $\mathbf{v_2}$ στην

κατεύθυνση του $\mathbf{w_1}$. Στην συνέχεια ορίζουμε $\mathbf{w_3=v_3+s_{31}w_1+s_{32}w_2}$ και επιλέγουμε $s_{31}$, $s_{32}$ έτσι ώστε $\mathbf{(w_3,w_1)=(w_3,w_2)=0}$. Η διαδικασία επαναλαμβάνεται για τα υπόλοιπα διανύσματα.

**11. <u>Υπολογισμός του Q, R, και y με τριγωνική ορθογωνοποίηση Gram-Schmidt</u>**

Υπολογίζουμε την ακολουθία των πινάκων $A=A^{(1)},A^{(2)},...,A^{(n+1)}=Q$ όπου $A^{(k)}$ έχει την μορφή $A^{(k)}=(q_1,...,q_{k-1},a_k^{(k)},...,a_n^{(k)})$ . Οι πρώτες (k-1) στήλες είναι οι στήλες της $Q$ , και έχουν ήδη ορθογωνοποιηθεί στα $q_1,...,q_{k-1}$ . Στο $k$ βήμα ορθογωνοποιούμε τα διανύσματα $a_{k+1}^{(k)},...,a_n^{(k)}$ ως προς $q_k$ :

$$q_k = a_k^{(k)}, \quad d_k = q_k^T q_k, \quad r_{kk} = 1,$$
$$a_j^{(k+1)} = a_j^{(k)} - r_{kj} q_k \ , \ r_{kj} = \frac{q_k^T a_j^{(k)}}{d_k},$$
$$j = k+1, \ldots, n.$$

Το διάνυσμα $b$ μετασχηματίζεται με τον ίδιο τρόπο $b=b^{(1)},b^{(2)},...,b^{(n+1)}$ όπου $b^{(k+1)}=b^{(k)}-y_k q_k$ , $y_k=\frac{q_k^T b^{(k)}}{d_k}$ . Το $b^{(n+1)}=r$ . Μετά $n$ βήματα λαμβάνουμε $Q=(q_1,q_2,...,q_n),\quad R=(r_{kj})$ , $y=(y_1,\ldots,y_n)^T$ έτσι ώστε $Q^TQ=diag(d_k)$ , $A=QR$ , $b=Qy+r.$ Για τον υπολογισμό του $R$ και $y$ απαιτούνται 2 φορές περισσότερες πράξεις από το να λύσουμε τις κανονικές εξισώσεις.

Μέχρι στιγμής έχουμε υποθέσει ότι οι στήλες του $A$ είναι γραμμικά ανεξάρτητες. Από την σχέση $Q=AR^{-1},$ όπου $S=R^{-1}$ είναι άνω τριγωνικός με μοναδιαία διαγώνια σημεία, συμπεραίνουμε ότι $q_k = s_{1k}\ a_1 + s_{2}\ a_2 + \ldots +$ Ας υποθέσουμε ότι $a_1,a_2,\ldots,a_{k-1}$ είναι γραμμικά ανεξάρτητα και $a_k$ είναι γραμμικώς συνδυασμός των $a_1,a_2,\ldots,a_{k-1}$ επομένως και των $q_1,q_2,...,q_{k-1}$ . Τότε θα πρέπει να ισχύει $a_k^{(k)}=0$ και η διαδικασία ορθογωνοποίησης σταματά. Όμως, αν $rank(A)>k-1$ τότε υπάρχει διάνυσμα $a_j^{(k)}\neq 0$ για $k\le j\le n$ και εναλλάσσοντας στήλες $k$ και $j$ μπορούμε να συνεχίσουμε την διαδικασία και στις στήλες $q$ που είναι γραμμικά εξαρτημένες. Η παρατήρηση αυτή οδηγεί σε μια παραλλαγή της μεθόδου Gram-Schmidt με οδήγηση στηλών όπου στο $k$ βήμα επιλέγουμε $s$ σαν ο μικρότερος ακέραιος που ικανοποιεί την σχέση $\left\|a_s^{(k)}\right\|_2=\max_{k\le j\le n}\left\|a_j^{(k)}\right\|_2$ και τότε εναλλάσσουμε τις στήλες $k$ και $s.$

Η παραπάνω παραλλαγή της Gram-Schmidt μπορεί να εφαρμοσθεί και όταν $A$ έχει γραμμικά εξαρτημένες στήλες, δηλαδή όταν $rank(A)=p<n$ . Τότε λαμβάνουμε την παραγοντοποίηση $Q=(q_1,...,q_n),\quad A=Q(R,S),\quad b=Qy+r$ όπου $R$ είναι $rxr$ άνω τριγωνικός πίνακας $S$ είναι $rx(n-r)$ (Για απλοποίηση του συμβολισμού έχουμε υπόθεση ότι οι εναλλαγές στηλών έχουν γίνει εκ των προτέρων). Στην περίπτωση αυτή η λύση των ελαχίστων τετραγώνων ικανοποιεί την σχέση $(R,S)x=y$ και μπορεί να γραφεί $x=(x_1,x_2)^T$ , με $x_1=R^{-1}y-R^{-1}Sx_2$ όπου $x_2$ είναι ένα τυχαίο διάνυσμα με

$(n-r)$ στοιχεία. Επομένως, μπορούμε να θεωρήσουμε ως μερική λύση την $x_1 = R^{-1}y$, $x_2=0$ . Οι παραπάνω υπολογισμοί υλοποιούνται από τις MatLab συναρτήσεις cgs.m και mgs.m:

```
function [Q, R] = mgs(A, varargin)
% QR factorization by modified Gram-Schmidt iteration
%
% Usage: [Q,R] = mgs(A)
%        [Q,R] = mgs(A, 1)
%        [Q,R] = mgs(A, 2)
%
% INPUT:
% A         - m-by-n matrix
% (optional)
% verbose   - {0,1,2}. 0 - non verbose (default);
%                      1 - displays A, Q and R at each iteration
steps;
%                      2 - displays AR = Q the implicit matrix
equations,
%                          demonstrating the "triangular
orthogonalization".
% OUTPUT:
% Q         - m-by-n unitary matrix;
% R         - square n-by-n upper-triangular matrix.
%
% ALGORITHM:
% Algorithms 8.1 from "Numerical linear algebra" by L. Trefethen.
%
% Examples:
%           A = randn(5,3); [Q,R] = mgs(A);
%           A = randn(5); [Q,R] = mgs(A, 1);
%           A = randn(5); [Q,R] = mgs(A, 2);
%
% See also: cgs, hr, slu, splu, backSub, forSub, solveLinearSystem.

% Check input:
error(nargchk(1,2,nargin));
% Defaults:
verbose = 0;
% Parse input:
if nargin>1
    verbose = varargin{1};
end


% MAIN:
[m, n] = size(A);
Q = zeros(m,n);
R = zeros(n);
% _____________________________________________ Algorithm begin:
for k=1:n
    % Normalization
```

```
    R(k,k) = norm(A(:,k));
    A(:,k) = A(:,k)/R(k,k);             % Q is written in place of A
    % Projection
    R(k,k+1:n) = A(:,k)'*A(:,k+1:n);
    projAonQ = A(:,k)*R(k,k+1:n);
    A(:,k+1:n) = A(:,k+1:n) - projAonQ;
% _____________________________________ Algorithm end.

    % ___________________________________ Visualization:
    switch verbose
        case 1
            Q(:,k) = A(:,k);
            figure(k);clf;
            dispMEq(['A^{(' num2str(k) ')};Q;R'], A, Q, R)
            title(['Step ' num2str(k)], 'FontSize', 16)
        case 2
            figure(k);clf;
            Aold = A; Aold(:,k+1:n) = Aold(:,k+1:n) + projAonQ;
            Aold(:,k) = Aold(:,k)*R(k,k);
            Rk = eye(n);
            Rk(k,k:n) = [1 -R(k,k+1:n)]/R(k,k);
            dispMEq(['A^{(' num2str(k-1) ')}*R_{' num2str(k)
'}=A^{(' num2str(k) ')}'],...
                    Aold, Rk, A);
            title(['Step ' num2str(k)], 'FontSize', 16);
    end
end
Q = A;
function [Q, R] = cgs(A, varargin)
% QR factorization by classical Gram-Schmidt iteration
%
% Usage: [Q,R] = cgs(A)
%        [Q,R] = cgs(A, 1)
%
% INPUT:
% A         - m-by-n matrix
% (optional)
% verbose   - {0,1}. 0 - non verbose (default);
%                    1 - displays the orthogonalization steps:
%                        for m<=3 - each step is visualized
geometrically,
%                        for m>3  - the A and current Q and R are
displayed in the text mode.
%
% OUTPUT:
% Q         - square m-by-n unitary matrix;
% R         - n-by-n upper-triangular matrix.
%
% ALGORITHM:
% Algorithm 7.1 from "Numerical linear algebra" by L. Trefethen.
%
% Examples:
%           A = randn(5); [Q,R] = cgs(A);
%           A = randn(3); [Q,R] = cgs(A,1);
```

```
%           A = randn(5); [Q,R] = cgs(A,1);
%
% See also: mgs, slu, splu, hr, backSub, forSub, solveLinearSystem.

% Check input:
error(nargchk(1,2,nargin));
% Defaults:
verbose = 0;
% Parse input:
if nargin>1, verbose = varargin{1}; end
[m,n] = size(A);
if verbose && m<=3, verbose = 2; elseif verbose && m>3, verbose = 1;
end

% MAIN:
Q = zeros(m,n);
R = zeros(n);
% ____________________________________________ Algorithm begin:
for jj=1:n
    % Projection:
    R(1:jj-1,jj) = Q(:,1:jj-1)'*A(:,jj);
    projQ = Q(:,1:jj-1)*R(1:jj-1,jj);
    temp = A(:,jj) - projQ;
    % Normalization:
    R(jj,jj) = norm(temp);
    Q(:,jj) = temp/R(jj,jj);
% ____________________________________________ Algorithm end.


    % ___________________________________________ Visualization:
    switch verbose
        case 1
            figure(jj);clf;
            dispMEq('A;Q;R',A,Q,R);
        case 2
            labelsA = cellstr(strcat('a_', num2str((1:n)') ));
            labelsQ = cellstr(strcat('q_', num2str((1:jj)') ));

            figure(jj);clf;hold on;
            drawVector(A, 'ro-', labelsA);
            drawVector(Q(:,1:jj), 'gs-', labelsQ);
            drawSpan(Q(:,1:jj-1))
            drawVector([projQ temp], 'b.-');
            drawLine([A(:,jj) projQ]);
            drawLine([A(:,jj) temp]);

            axis equal
            hold off
    end
end
```

**Άσκηση 3:**

Οι παραπάνω συναρτήσεις cgs() και mgs() χρησιμοποιούν το **drawLA Toolbox** (δέστε παράρτημα 2 των σημειώσεων για να το κατεβάσετε από το www.mathworsks.com) παίρνουν μία προαιρετική παράμετρο "0", "1" ή "2" που καθορίζει αν θα εμφανίζονται τα ενδιάμεσα βήματα του μετασχηματισμού της συνάρτησης κατά την εκτέλεση της ή όχι: 0 – για να μην εμφανίζονται, 1 ή 2 – για να εμφανίζονται. Εκτελέστε τα παρακάτω πειράματα:

1. ***Γεωμετρική αναπαράσταση της κλασικής διαδικασίας Gram-Schmidt***: τρέξτε A=rand(2); [Q,R]=cgs(A,1); και A=rand(3); [Q,R]=cgs(A,1);. Σιγουρευτείτε πως καταλαβαίνετε τα σχήματα που παράγονται.
2. ***Διαφορά μεταξύ της κλασικής και της τροποποιημένης διαδικασίας Gram-Schmidt***: για να δείτε πως εμφανίζονται στους υπολογισμούς οι Q και R τρέξτε cgs() και mgs() σε μεγαλύτερους πίνακες: A = rand(5); [Q1,R1] = cgs(A,1); [Q2,R2] = mgs(A,1).
3. ***Τριγωνική ορθογωνοποίηση***: ο πίνακα Q εμφανίζεται σαν το αποτέλεσμα της σειράς του πολλαπλασιασμού του A με συγκεκριμένους άνω τριγωνικούς πίνακες: **A R1 R2 R3 ... Rn = Q**. θέστε την προαιρετική παράμετρο του mgs() "2" για να δείτε την επαναληπτική διαδικασία: A=rand(5); [Q,R]=mgs(A,2).

## 12. QR παραγοντοποίηση με την Householder μέθοδο (MatLab συνάρτηση **qr**):

Ας θεωρήσουμε τον πίνακα $H(v) = I - \frac{2vv^T}{v^T v}$ όπου $v$ είναι ένα τυχαίο $nx1$ μη μηδενικό διάνυσμα, $I$ είναι $nxn$ μοναδιαίος πίνακας, $vv^T$ το εξωτερικό γινόμενο του διανύσματος $v$ με τον εαυτό του, και $v^T v$ το εσωτερικό γινόμενο του $v$ με τον εαυτό του. Ο πίνακας $H(v)$ έχει μια σειρά από χρήσιμες ιδιότητες όπως $H^2(v) = I$ που σημαίνει ότι $H^{-1}(v) = H(v)$ . Επιπλέον, $H^T(v) = H(v)$ που συνεπάγει ότι ο $H(v)$ είναι συμμετρικός ορθογώνιος πίνακας.

Σημειώστε ότι *Hx* είναι η αντανάκλαση του *x* ως προς το "hyperplane" κάθετο στο διάνυσμα *v* ($z^T v = 0$) (δες το παρακάτω σχήμα)

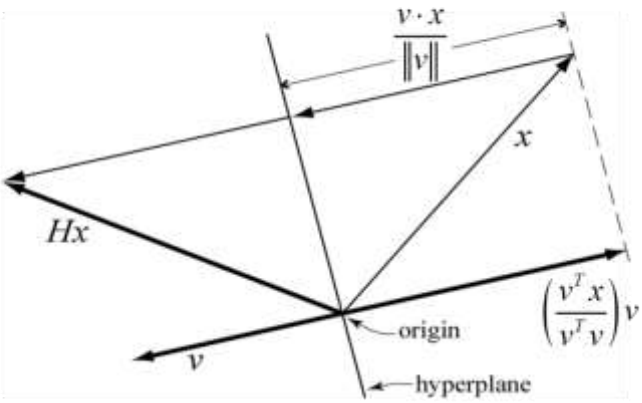
$$Hx = x - (\frac{2v^T x}{v^T v})v$$

και ισχύει $p_A(\lambda) = \det(A - \lambda I) = 0$ $Hx = x - (\frac{2v^T x}{v^T v})v$ όπου $\frac{2v^T x}{v^T v}$ είναι βαθμοτή τιμή.

$$-\lambda^3 + 4\lambda^2 - \lambda = 0$$

$$y = x(x-1)$$

Η γεωμετρική ερμηνεία του μετασχηματισμού Householder δίδεται στο παρακάτω σχήμα.




Στην συνέχεια δείχνουμε πως μια ακολουθία μετασχηματισμών Householder παράγει $QR$ παραγοντοποίηση ενός τετραγωνικού πίνακα $A$. Η γενίκευση του σε μη τετραγωνικούς πίνακες είναι προφανής. Έστω

$$A = QR = Q\begin{bmatrix} r_{11} & r_{12} & . & . & . & r_{1n} \\ & r_{22} & . & . & . & r_{2n} \\ & & . & & & . \\ & & & . & & . \\ & & & & . & . \\ & & & & & r_{nn} \end{bmatrix}, \qquad QQ^T = I$$

Ας αρχίσουμε με την πρώτη στήλη η οποία έχει ένα μη μηδενικό το πρώτο της στοιχείο. Για κάθε $x \in R^n$ , μπορούμε να βρούμε ένα διάνυσμα $v$ που ο αντίστοιχος μετασχηματισμός ανάκλασης $P$ μηδενίζει όλα τα στοιχεία του $x$ :

$$Px = [I - 2\frac{vv^T}{v^T v}]x = x - 2\frac{(v*x)}{|v|^2}v = \begin{bmatrix} b \\ 0 \\ . \\ . \\ . \\ 0 \end{bmatrix} = be^1$$

Από την σχέση αυτή συμπεραίνουμε ότι πρέπει $v = x + \sigma e^1$ , καθώς ισχύουν τα παρακάτω

$$\|v\|^2 = (x + \sigma e^1) \cdot (x + \sigma e^1) = \|x\|^2 + 2\sigma x_1 + \sigma$$

$$v \cdot x = (x + \sigma e^1) \cdot x = \|x\|^2 + \sigma x_1$$

και Householder μετασχηματισμός στο $x$ είναι

$$Hx = x - \frac{2(\|x\|^2 + x_1)}{\|x\|^2 + 2x_1 +}(x + e^1) = [1 - \frac{2(\|x\|^2 + x_1)}{\|x\|^2 + 2x_1 +}]x - [\frac{2(\|x\|^2 + x_1)}{\|x\|^2 + 2x_1 +}]e^1$$

Η ικανοποίηση της εξίσωσης $Hx = be^1$ απαιτεί $1 - \frac{2(\|x\|^2 + \sigma x_1)}{\|x\|^2 + 2\sigma x_1 + \sigma} = 0 \Rightarrow = \sigma = \varepsilon\|x\|$ και

$$\varepsilon = sign(x_1) = \begin{cases} -1, x_1 < 0 \\ 1, x_1 \geqslant 0 \end{cases}$$

και $Hx = -\varepsilon\|x\|e^1$

Έτσι βρήκαμε ένα ορθογώνιο μετασχηματισμό $H = U_1$ έτσι ώστε

$$U_1 A = \begin{bmatrix} r_{11} & r_{12} & . & . & . & r_{1n} \\ & r_{22}^1 & . & . & . & r_{2n}^1 \\ & . & . & . & . & . \\ & . & . & . & . & . \\ & . & . & . & . & . \\ & r_{n2}^1 & . & . & . & r_{nn}^1 \end{bmatrix}$$

Στην συνέχεια προσδιορίζουμε ένα Household μετασχηματισμό $W_2$ που εφαρμόζεται στο $(n-1)x(n-1)$ πίνακα $\begin{bmatrix} r_{22}^1 & . & . & . & r_{2n}^1 \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ r_{n2}^1 & . & . & . & r_{nn}^1 \end{bmatrix}$ για να μηδενίσουμε όλα τα σημεία της πρώτης στήλης εκτός το πρώτο και κατασκευάζουμε τον $nxn$ πίνακα $U_2 = \begin{bmatrix} 1 & 0 \\ 0 & W_2 \end{bmatrix}$

έτσι ώστε $U_2 U_1 A = \begin{bmatrix} r_{11} & r_{12} & . & . & . & r_{1n} \\ & r_{22}^1 & . & . & . & r_{2n}^1 \\ & 0 & r_{33}^2 & . & . & r_{3n}^2 \\ & . & . & . & . & . \\ & . & . & . & . & . \\ & . & . & . & . & . \\ & 0 & r_{n3}^2 & . & . & r_{nn}^2 \end{bmatrix}$

Η διαδικασία επαναλαμβάνεται έως ότου

$$U_{n-1}U_{n-2}\cdots U_2U_1A=\begin{bmatrix} r_{11} & r_{12} & . & . & . & r_{1n} \\ & r_{22}^1 & . & . & . & r_{2n}^1 \\ & 0 & r_{33}^2 & . & . & r_{3n}^2 \\ & . & 0 & . & . & . \\ & . & . & . & . & . \\ & . & . & . & . & . \\ & 0 & 0 & . & . & r_{nn}^{n-1} \end{bmatrix}=R$$

οπότε $Q=U_1^TU_2^T\cdots U_{n-2}^TU_{n-1}^T$ .

Σημειώστε ότι ο αλγόριθμος απαιτεί $\sim n^3$ πράξεις και έχει υλοποιηθεί στην MatLab από την συνάρτηση **qr.**

Στην γενική περίπτωση όπου $A$ mxn όπου m>n έχουμε

$$Q^TA=\begin{bmatrix} R \\ 0 \end{bmatrix} \quad \text{or} \quad QQ^TA=A=Q\begin{bmatrix} R \\ 0 \end{bmatrix}$$

**Παράδειγμα 2**: Να επαναληφτεί το παράδειγμα 1 χρησιμοποιώντας την QR μέθοδο.

$$A=\begin{bmatrix} 1 & -1.0 & 1.0 \\ 1 & -0.5 & 0.25 \\ 1 & 0.0 & 0.0 \\ 1 & 0.5 & 0.25 \\ 1 & 1.0 & 1.0 \end{bmatrix}, \quad b=\begin{bmatrix} 1.0 \\ 0.5 \\ 0.0 \\ 0.5 \\ 2.0 \end{bmatrix}$$

**Βήμα 1**: Βρίσκομαι το διάνυσμα $v_1=a_1+\varepsilon\|a_1\|e^1$ με $a_1=(1\ 1\ 1\ 1\ 1)^T$ και $\|a_1\|=\sqrt{5}=2.236$ και ε=-1 για τη αποφυγή του σφάλματος διαγραφής εφόσον το πρώτο στοιχείου του $a_1$ είναι θετικό

$$v_1=a_1+\varepsilon\|a_1\|e^1$$

$$v_1=\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}-\begin{bmatrix} -2.236 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}=\begin{bmatrix} 3.236 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$U_1 = I - 2v_1 v_1^T / v_1^T v_1$$

$$U_1 A = \begin{bmatrix} -2.236 & 0 & -1.118 \\ 0 & -0.191 & -0.405 \\ 0 & 0.309 & -0.655 \\ 0 & 0.809 & -0.405 \\ 0 & 1.309 & 0.345 \end{bmatrix}, \qquad U_1 b = \begin{bmatrix} -1.789 \\ -0.362 \\ -0.862 \\ -0.362 \\ 1.138 \end{bmatrix}$$

**Βήμα 2**: Βρίσκομαι το νέο διάνυσμα από το $a_2$(σημειώστε ότι παίρνουμε τα στοιχεία κάτω από τον διαγώνιο) βρίσκομαι $U_2$ και τον εφαρμόζουμε στο προηγούμενο αποτέλεσμα

$$v_2 = \begin{bmatrix} 0 \\ -0.191 \\ 0.309 \\ 0.809 \\ 1.309 \end{bmatrix} - \begin{bmatrix} 0 \\ 1.581 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1.772 \\ 0.309 \\ 0.809 \\ 1.309 \end{bmatrix}$$

$$U_2 U_1 A = \begin{bmatrix} -2.236 & 0 & -1.118 \\ 0 & 1.581 & 0 \\ 0 & 0 & -0.725 \\ 0 & 0 & -0.589 \\ 0 & 0 & 0.047 \end{bmatrix}, \qquad U_2 U_1 b = \begin{bmatrix} -1.789 \\ 0.632 \\ -1.035 \\ -0.816 \\ 0.404 \end{bmatrix}$$

**Βήμα 3**: Βρίσκομαι το νέο διάνυσμα από το $a_3$(σημειώστε ότι παίρνουμε τα στοιχεία κάτω από τον διαγώνιο) βρίσκομαι $U_3$ και τον εφαρμόζουμε στο προηγούμενο αποτέλεσμα

$$v_3 = \begin{bmatrix} 0 \\ 0 \\ -0.725 \\ -0.589 \\ 0.047 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0.9354 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1.6605 \\ -0.5892 \\ 0.0467 \end{bmatrix}$$

$$U_3 U_2 U_1 A = \begin{bmatrix} -2.236 & 0 & -1.118 \\ 0 & 1.581 & 0 \\ 0 & 0 & 0.9354 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad U_3 U_2 U_1 b = \begin{bmatrix} -1.7889 \\ 0.6325 \\ 1.3363 \\ 0.0258 \\ 0.3371 \end{bmatrix}$$

$$x = \begin{bmatrix} 0.086 & 0.400 & 1.429 \end{bmatrix}^T$$

Λύνοντας το άνω τριγωνικό σύστημα $Rx=c_1$ έχουμε:

Η παραγοντοποίηση QR του Housholder υλοποιείται στην συνάρτηση hr.m

```
function [A, b] = hr(A, varargin)
% QR factorization by Householder reflections
%
% Usage:   R = hr(A)
%        [ R, b_out ] = hr(A, b)
%        [ R, ...   ] = hr(..., 1)
% INPUT:
% A         - m-by-n matrix.
% (optional)
% b         - right-hand-side m-by-1 vector in Ax = b;
% verbose   - {0,1}. 0 - non verbose (default);
%                    1 - displays the orthogonalization steps:
%                        for m<=3 - each step is visualized geometrically,
%                        for m>3  - the transformations from A to R are displayed
%                                   in the text mode.
% OUTPUT:
% R         - square n-by-n upper-triangular matrix
%             resulting from unitary transformation R = Q'*A.
% b_out     - new right-hand-side n-by-1 vector for Rx = b_out, b_out=Q'*b.
%
% ALGORITHM:
% Algorithms 10.1 and 10.2 from "Numerical linear algebra" by L. Trefethen.
%
% Examples:
%           A = randn(5, 3); R = hr(A);
%           A = randn(5); R = hr(A, 1);
%           A = randn(2); R = hr(A, 1);
%           A = randn(3); R = hr(A, 1);
%
% See also: cgs, mgs, slu, splu, backSub, forSub, solveLinearSystem.


% Check input:
error(nargchk(1,3,nargin));
% Defaults:
b = zeros(size(A,1),0);
verbose = 0;
% Parse input:
if nargin>1
    for ii=1:nargin-1
        if isscalar(varargin{ii})
            verbose = varargin{ii};
            labels = {'a_1','a_2','a_3'};
            axLabels = {'x', 'y', 'z'};
        else
            b = varargin{ii};
            if size(b,1)<2, error('The r.h.s. vector b must be a column-vector');end;
        end
    end
end


% MAIN:
[m,n]=size(A);
for k=1:min(m-1,n)

    % _____________________________________ Visualization:
    if verbose && m < 4
        figure(k); clf; hold on;
        drawVector(A(k:m,k:n), 'bo-', labels(k:n), 'AxesLabels', axLabels(k:n));
        title(['QR: step ' num2str(k)]);
    end % ____________________________________________________ .

    % _____________________________________ Algorithm begin:
    x = A(k:m,k);
    sigma = norm(x);
    e1 = zeros(size(x)); e1(1) = 1;
    vk = sign( x(1)+(x(1)==0) )*sigma*e1 + x;
    vk = vk/norm(vk);

    % _____________________________________ Visualization:
    if verbose && m<4
        drawPlane(vk); alpha(.1);
```

```
        drawLine([A(k:m,k), [-sign(x(1))*sigma; zeros(m-k,1)]], '2r');
    end % _____________________________________________________.

    % Apply transformation to the matrix
    A(k:m,k:n) = A(k:m,k:n) - 2*vk*(vk'*A(k:m,k:n));

    if ~isempty(b)  % apply the transformation to the r.h.s. vector
        b(k:m) = b(k:m) - 2*vk*(vk'*b(k:m));
    end
    % ____________________________________________ Algorithm end.

    % ____________________________________________ Visualization:
    if verbose
        if m<4
            drawVector(A(k:m,k:n), 'gs-', labels(k:n) );
            hold off
        else
            figure(k);clf;
            dispMEq(['A_{' num2str(k) '}'] ,A);
        end
    end
end
A = triu(A(1:n,1:n));
if ~isempty(b), b = b(1:n); end
```

## Άσκηση 2: Σύγκριση μεθόδων για την λύση του ΕΤ

Στόχος της άσκησης είναι να συγκριθούν οι μέθοδοι της άσκησης 2 ως προς την ταχύτητα και ακρίβεια. Για το σκοπό αυτό συγκρίνουμε τους αλγορίθμους σε υπολογιστικά δύσκολα προβλήματα όπως προσέγγιση δεδομένων με υψηλού επιπέδου πολυωνύμων.

Στη άσκηση αυτή πρέπει να συγκρίνεται τις μεθόδους σε *m* δεδομένα σημεία `(t_i,b_i)` προσαρμόζαντας ένα πολυώνυμο *n-1* βαθμού:

$$p_{n-1}(t) = x_n t^{n-1} + x_{n-1} t^{n-2} + \ldots + x_1$$

όπου οι *n* συντελεστές **x** είναι οι άγνωστοι.

Η διαδικασία σύγκρισης έχει ως εξής.

Για δεδομένες τιμές *n* και *m* εφαρμόστε τα παρακάτω βήματα:

- Ορίστε ένα *m*-διάνυσμα *t* που οι συντεταγμένες αντιστοιχούν σε ισαπέχοντα σημεία μεταξύ 0 και 1 (εφαρμόστε την MATLAB συνάρτηση, `t=linspace(0,1,m)`).
- Σχηματίστε τον πίνακα *mxn* **A**.
- Δημιουργείστε ένα τυχαίο *n*-διάνυσμα που αντιπροσοπεύει την λύση: `x = randn(n,1)`.
- Από την λύση αυτή, υπολογείστε το δεξιό m-διάνυσμα, **b** = *p*(**t**).

- Λύστε το ΕΤ σύστημα, **Ax=b**, με τις μεθόδους της άσκησης 2 και υπολογείστε το απόλυτο σφάλμα της λύσης **x** και το υπόλοιπο: `res=abs(A*x-b)`.

Εφαρμόστε τα παραπάνω βήματα για *n=3,4,...,12* και *m=2*n*. Για να συγκεντρώσουμε ικανοποιητικά στατιστικά στοιχεία για να εκτιμήσουμε το χρόνο εκτέλεσης των αλγορίθμων και το σφάλμα των επαναλάβατε τους υπολογισμούς, *N*=1000 φορές.

Για ολοκληρώσετε την αξιολόγηση των μεθόδων θα πρέπει να παράγετε τα παρακάτω τα γραφήματα:

Απόλυτο σφάλμα vs. αριθμό κατάστασης, μέσο όρο των υπολοίπων vs. n, μέσο όρο χρόνου εκτέλεσης vs. n για κάθε μέθοδο.

Περιγράψτε τις παρατηρήσεις σας.

**Ο σκελετός του κώδικα για την σύγκριση των μεθόδων:**

```
methods = {'neq', 'qr_mgs', 'qr_hr', 'svd'};
nn = 3:12;                                      % degrees of polynomials
% Arrays for the results
T   = zeros(length(nn), length(methods));     % Execution time
MAE = zeros(length(nn), length(methods));     % Mean absolute error
Res = zeros(length(nn), length(methods));     % Mean residual
cnd = zeros(length(nn),1);                    % Matrix condition number

N = 1000;                              % number of iterations
for k = 1:length(nn)
    n = nn(k); m = 2*n;
    % YOUR CODE HERE: form the matrix A and store its condition number.
    for ii=1:length(methods)
        err = 0; res = 0;
        t0 = cputime;
        for jj=1:N   % Iterate N times to collect time and error
statistics

            % YOUR CODE HERE: generate the "ground truth" and the
            % corresponding r.h.s. vector b.

            x  = solveLSS(A,b,methods{ii});
            err = err + abs(x - x_true);
            res = res + abs(b - A*x);
        end
        T(k, ii) = (cputime - t0)/N;     % Mean time
        MAE(k, ii) = sum(err)/N;         % Mean absolute error
        Res(k, ii) = sum(res)/N;         % Mean residuals
    end
end
% _____________________________________________________________ Plots
% YOUR 4 PLOTS HERE:
```

## 13. Υπολογισμοί στο περιβάλλον Python και περιεχόμενο βιβλιοθήκης NumPy

**Παράδειγμα** υπολογισμού της λύσης ενός υπερ-προσδιορισμένου πίνακα με την μέθοδο ET (κανονικές εξισώσεις) και QR με την χρήση προγραμμάτων της NumPy
ΚΩΔΙΚΑΣ

```
from numpy import *
# generating a random overdetermined system
A = random.rand(5,3)
b = random.rand(5,1)
x_lstsq = linalg.lstsq(A,b)[0] # computing the numpy
solution
Q,R = linalg.qr(A) # qr decomposition of A
Qb = dot(Q.T,b) # computing Q^T*b (project b onto the range
of A)
x_qr = linalg.solve(R,Qb) # solving R*x = Q^T*b
# comparing the solutions
print 'overdetermined system'
print 'A:',A
print 'b:',b
print 'qr solution'
print x_qr
print 'lstqs solution'
print x_lstsq
```

**OUTPUT**

```
overdetermined system
A: [[ 0.5912765   0.1903997   0.08812992]
 [ 0.91597286  0.41655643  0.30507245]
 [ 0.88845183  0.83489894  0.0262819 ]
 [ 0.14501993  0.31840504  0.57668724]
 [ 0.78988119  0.80282517  0.45605456]]
b: [[ 0.72436741]
 [ 0.13049554]
 [ 0.64724984]
 [ 0.41084585]
 [ 0.29129265]]
qr solution
[[ 0.35572351]
 [ 0.21722409]
 [ 0.07921489]]
lstqs solution
[[ 0.35572351]
 [ 0.21722409]
 [ 0.07921489]]
```

## 14. Αναφορές

1. https://sites.google.com/a/uni-konstanz.de/na09/Home/methods/fpns

2. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
3. Numerical Linear Algebra, L.N. Trefethen and David Bau, SIAM, 1997
4. Numerical Methods for Chemical Engineering, Kenneth J. Beers, Cambridge, 2007

# ΚΕΦΑΛΑΙΟ 9: ΙΔΙΟΤΙΜΕΣ ΚΑΙ ΙΔΙΑΖΟΥΣΕΣ ΤΙΜΕΣ

## Περιεχόμενα



### 1. Εύρεση ιδιοτιμών και ιδιοανυσμάτων πινάκων

1) **Πρόβλημα ιδιοτιμών και ιδιοανυσμάτων πινάκων**: Οι ιδιοτιμές $\lambda_i$ και ιδιοανύσματα $\mathbf{x_i}$ ενός nxn πίνακα **A** ικανοποιούν την σχέση $(\mathbf{A}-\lambda_i\mathbf{I})\mathbf{x_i}=0$, $\mathbf{x_i}\neq \mathbf{0}$. Από την σχέση αυτή συμπεραίνουμε ότι οι ιδιοτιμές $\lambda_i$ είναι οι n ρίζες του χαρακτηριστικού πολυωνύμου $p_A(\lambda) = \det(A - \lambda I) = 0$, όπου $p_A(\lambda)$ είναι n βαθμού πολυώνυμο ως προς λ.
2) **Υπολογισμός ιδιοτιμών και ιδιοανυσμάτων πινάκων:** Αν μια ιδιοτιμή είναι γνωστή τότε το ιδιοάνυσμα προκύπτει από την λύση του ομογενούς συστήματος $(\mathbf{A}-\lambda_i\mathbf{I})\mathbf{x_i}=0$. Αν $\mathbf{x_i}$ είναι γνωστό τότε το $\lambda_i$ υπολογίζεται από την σχέση $\lambda_i = \frac{x_i^T A x_i}{x_i^T x_i}$.
3) **Παράδειγμα προσδιορισμού ιδιοτιμών και ιδιοανυσμάτων συμμετρικών πινάκων**

Θεωρούμε τον πίνακα $A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$ Να βρεθούν οι ιδιοτιμές και ιδιοανύσματά του. Οι ιδιοτιμές βρίσκονται από το χαρακτηριστικό πολυώνυμο $\det(A - \lambda I) = 0$ ή $-\lambda^3 + 4\lambda^2 - 3\lambda = 0$ που έχει ρίζες 1, 0, 3. Τα ιδιοανύσματα βρίσκονται από την λύση των ομογενοί συστημάτων $(A - \lambda I)x = 0$. Τα συστήματα αυτά έχουν άπειρες λύσεις, οπότε πρέπει να υποθέσετε κάποια τιμή για το $x_1$ και να βρείτε τα υπόλοιπα από τις υπόλοιπες εξισώσεις. Για την εύρεση του ιδιοανύσματος που αντιστοιχεί στην ιδιοτιμή $\lambda_3$=3, λύνουμε το σύστημα

$$\begin{bmatrix} 1-\lambda_3 & -1 & 0 \\ -1 & 2-\lambda_3 & -1 \\ 0 & -1 & 1-\lambda_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$ θέτοντας $x_1=1$ και λύνοντας τις υπόλοιπες εξισώσεις βρίσκομαι $x_2=-2$ και $x_3=1$. Έτσι ο πίνακας των ιδιοανυσμάτων είναι $X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -2 \\ 1 & -1 & 1 \end{bmatrix}$. Συνηθίζεται να χρησιμοποιούμε τα μοναδιαία ιδιοανύσματα $X = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{2} & 1/\sqrt{4} \\ 1/\sqrt{3} & 0 & -2/\sqrt{4} \\ 1/\sqrt{3} & -1/\sqrt{2} & 1/\sqrt{4} \end{bmatrix}$

**4) Ιδιότητες των ιδιοτιμών και ιδιοανυσμάτων συμμετρικών πινάκων**

- Όλες οι ιδιοτιμές συμμετρικών πινάκων είναι πραγματικές
- Όλες οι ιδιοτιμές συμετρικών και θετικά ορισμένων πινάκων είναι πραγματικές και θετικές
- Ο πίνακας των ιδιοανύσματων συμμετρικών πινάκων είναι ορθοκανονικός, δηλαδή ισχύει $\mathbf{X}^T\mathbf{X} = \mathbf{I}$.
- Αν οι ιδιοτιμές του Α είναι $\lambda_i$ , τότε οι ιδιοτιμές του αντιστρόφου $\mathbf{A}^{-1}$ είναι $1/\lambda_i$

**5) Υπολογισμός των ιδιοτιμών συμμετρικών πινάκων με την QR μέθοδο**

Σημειώστε ότι ο άνω τριγωνικός πίνακας R που βρίκαμε εφαρμόζοντας την QR παραγοντοποίηση δεν έχει τις ίδιες ιδιοτιμές με τον Α. Όμως ο πίνακας $A^{[1]}=Q^TAQ$ είναι όμοιος με το Α, δηλαδή έχει τις ίδιες ιδιοτιμές λόγο της ιδιότητας του Q. Εφαρμόζοντας τον παραπάνω μετασχηματισμό ομοιότητας επαναλλειπτικά παίρνουμε τον αλγόριθμο $A^{[k]}=Q^{[k]}R^{[k]}$,$A^{[k+1]}=(Q^{[k]})^TA^{[k]}Q^{[k]}$). Αποδεικνύεται ότι

$$A^{[k\to\infty]} = \begin{bmatrix} r_{11} & r_{12} & \cdot & \cdot & \cdot & r_{1P} \\ & r_{22} & \cdot & \cdot & \cdot & r_{2P} \\ & 0 & r_{33} & \cdot & \cdot & r_{3p} \\ & \cdot & 0 & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot & \cdot & \cdot \\ & 0 & 0 & \cdot & \cdot & r_{pp} \end{bmatrix}$$

είναι ένας «block» άνω τριγωνικός πίνακας όπου οι πίνακες $r_{11}$, $r_{22, \ldots,}$ $r_{PP}$ είναι 1x1 ή 2x2 πίνακες. Για την πρώτη περίπτωση, τα διαγώνια στοιχεία είναι οι ιδιοτιμές του Α. Για την δεύτερη περίπτωση, οι 2x2 υποπίνακες $R_{jj}$ έχουν δύο ιδιοτιμές που είναι συζυγής μιγαδικές (για Α πραγματικούς) και είναι ιδιοτιμές του Α. Σημειώστε ότι η μέθοδος είναι εφαρμόσιμη για κάθε πραγματικό πίνακα Α. Τα ιδιοανύσματα μπορούν

να προσδιορισθούν από τα ομογενοί συστήματα $(A-\lambda_i I)x_i=0$ Η σύγκλιση της μεθόδου αυτής μπορεί να βελτιωθεί αν τροποποιήσουμε το παραπάνω αλγόριθμο θεωρόντας τον κανόνα $A^{[k]}-\mu I=Q^{[k]}R^{[k]}, A^{[k+1]}-\mu I=(Q^{[k]})^T A^{[k]}Q^{[k]}$ με $\mu=a^{[k]}_{n,n}$.

6) **Παράδειγμα εφαρμογής της QR μεθόδου για την εύρεση των ιδιοτιμών ενός πίνακα**

Τρέξτε το παρακάτω MatLab κώδικα για διαφορετικές τιμές της παραμέτρου Niter και παρατηρήσετε την σύγκλιση της μεθόδου.

```
A=[4 0 -1 1;0 4 2 -1;1 -2 4 1;-1 1 -1 4];
Niter=50;
A_w=A;
N=4;
for i=1:Niter
mu=A_w(N,N)
[Q,R]=qr(A_w-mu*eye(N));
A_w=R*Q+mu*eye(N);
end
A_w
```

**Άσκηση 6 :** α) Υπολογίστε τις ιδιοτιμές του πίνακα Α. β) Συγκρίνετε τις απαντήσεις σας με τις τιμές που μας δίνει η συνάρτηση της MatLab `[V,D] = eig(A, γ)` υπολογείστε τις ιδιοτιμές προσεγγίζοντας τις ρίζες του χαρακτηριστικού πολυωνύμου.

## 2. Ανάλυση πίνακα σε ιδιάζουσες τιμές - Singular Value Decomposition (SVD)

### Θεωρία 1: Singular Value Decomposition (SVD)

Κάθε πραγματικός $mxn$ πίνακας $A$ μπορεί να παραγοντοποιηθεί μαναδικά ως

$A = UDV^T$ όπου

$U$ είναι $mxn$ και ορθογώνιος πίνακας (οι στήλες του είναι ιδιονύσματα του $mxm$ πίνακα $AA^T$ )

( $AA^T = UDV^TVDU^T = UD^2U^T$ )

$V$ είναι $nxn$ και ορθογώνιος πίνακας (οι στήλες του είναι τα ιδιονύσματα του πίνακα $AA^T$ )

( $A^TA = VDU^TUDV^T = VD^2V^T$ )

$D$ είναι $nxn$ διαγώνιος πίνακας με $D = diag(\sigma_1, ..., \sigma_n)$ όπου τα στοιχεία

του είναι οι ιδιοτιμές του του Α διατεταγμένες έτσι ώστε $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$

(Αν σ είναι η ιδιοτιμή του $A$ , το τετραγωνό της είναι η ιδιοτιμή του $A^TA$ ) ,

**Λήμμα**: Αν $U = (u_1, u_2, ..., u_n)$ και $V = (v_1, v_2, ..., v_n)$ **,** τότε

$$A = \sum_{i=1}^{r} \alpha_i u_i v_i$$

(όπου $r$ είναι ο βαθμός (rank) του $A$ )

**Παράδειγμα**

Η SVD παραγοντοποίηση του

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix},$$

:

$$\begin{bmatrix} 0.4544 & -0.54177 & 0.70711 \\ 0.76618 & 0.64262 & -1.431 \times 10^{-38} \\ 0.4544 & -0.54177 & -0.70711 \end{bmatrix} \begin{bmatrix} 5.3723 & 0 & 0 \\ 0 & 0.37228 & 0 \\ 0 & 0 & 6.0138 \times 10^{-39} \end{bmatrix} \begin{bmatrix} 0.4544 & 0.76618 & 0.4544 \\ 0.54177 & -0.64262 & 0.54177 \\ 0.70711 & -5.2353 \times 10^{-39} & -0.70711 \end{bmatrix}$$

$$= \sum_{i=1}^{2} \sigma_i u_i v_i^T$$

Σημειώστε ότι α) Α είναι μη αντιστρέψιμος (Γιατί;), β) η δεύτερη ιδιοτιμή είναι πολύ ποιο μικρή από την πρώτη – αν την αγνοήσουμε - τότε ο Α μπορεί να παρασταθεί χωρίς μεγάλο σφάλμα ως εξής:

$$A = \begin{bmatrix} 5.3723 \end{bmatrix} \begin{bmatrix} 0.4544 \\ 0.76618 \\ 0.4544 \end{bmatrix} \begin{bmatrix} 0.4544 & 0.76618 & 0.4544 \end{bmatrix}$$

$$: \begin{bmatrix} 1.1093 & 1.8704 & 1.1093 \\ 1.8704 & 3.1537 & 1.8704 \\ 1.1093 & 1.8704 & 1.1093 \end{bmatrix}$$

γ) Ποιος είναι ο αντίστροφος του; Ποια είναι η προσέγγιση του;

### 3. Θεωρία: Υπολογισμός της τάξης (rank) ενός πίνακα χρησιμοποιώντας SVD

- Η τάξη ενός πίνακα είναι ίση με τον αριθμό των μη μηδενικών ιδιοτιμών.

### 4. Θεωρία: Υπολογισμός του αντιστρόφου ενός πίνακα χρησιμοποιώντας SVD

**Παράδειγμα**: $A^{-1}=V\ D_0^{-1}\ U^T$

$$= \begin{bmatrix} 0.4544 & 0.76618 & 0.4544 \\ 0.54177 & -0.64262 & 0.54177 \\ 0.70711 & -5.2353\times 10^{-39} & -0.70711 \end{bmatrix} \begin{bmatrix} 5.3723 & 0 & 0 \\ 0 & 1/0.37228 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.4544 & -0.54177 & 0.70711 \\ 0.76618 & 0.64262 & -1.431\times 10^{-38} \\ 0.4544 & -0.54177 & -0.70711 \end{bmatrix} =$$

$$\begin{bmatrix} 1.6153 & 1.2767 & 5.9809\times 10^{-2} \\ -1.2767 & -1.1639 & 7.1309\times 10^{-2} \\ 5.9809\times 10^{-2} & -7.1309\times 10^{-2} & 9.3071\times 10^{-2} \end{bmatrix}$$

## 5. Υπολογισμός του αριθμού κατάστασης με SVD

Ας θεωρήσουμε το σύστημα $Ax = b$

**Ορισμό**: Αν μια μικρή αλλαγή στο $b$ μπορεί να οδηγεί σε σχετικά μεγάλες αλλαγές στην λύση $x$ , τότε λέμε ότι $A$ είναι κακής κατάστασης ( ill-conditioned).

Το πηλίκο $\frac{\sigma_1}{\sigma_n}$ (η μεγαλύτερη ως προς την μικρότερη ιδιάζουσα (ιδιοτιμή) τιμή του Α) σχετίζεται με τον αριθμό κατάστασης του Α (όσο μεγαλύτερο είναι το πηλίκο τόσο ο Α είναι ποιο κοντά στον να μην είναι αντιστρέψιμος (ιδιάζων πίνακας- singular))

## 6. Λύση του συστήματος ελαχίστων χρησιμοποιώντας SVD

Ας θεωρήσουμε το υπερπροσδιορισμένο σύστημα $Ax = b$, ( $A$ είναι $m \times n$ με $m > n$ ). Έστω το υπόλοιπο $r = Ax - b$ . Το διάνυσμα x* που οδηγεί στο μικρότερο υπόλοιπο ονομάζεται η λύση των ελαχίστων τετραγώνων του συστήματος (σημειώστε ότι είναι μια προσεγγιστική λύση).

$\| r \| = \| Ax^* - b \| \le \| Ax - b \|$ για κάθε $x \in R^n$

Μολονότι η ΕΤ λύση υπάρχει πάντοτε, δεν είναι μοναδική!

Η ΕΤ λύση x με την μικρότερη νόρμα $\| x \|$ είναι μοναδική και δίδεται από την εξίσωση:

$$A^T Ax = A^T b \quad \text{ή} \quad x = (A^T A)^{-1} A^T b = A^+ b$$

Παράδειγμα:

$$\begin{bmatrix} -11 & 2 \\ 2 & 3 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 7 \\ 5 \end{bmatrix}$$

$$x = \left[ \begin{bmatrix} -11 & 2 & 2 \\ 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} -11 & 2 \\ 2 & 3 \\ 2 & -1 \end{bmatrix} \right]^{-1} \begin{bmatrix} -11 & 2 & 2 \\ 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 7 \\ 5 \end{bmatrix},$$

$$\begin{bmatrix} -11 & 2 & 2 \\ 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} -11 & 2 \\ 2 & 3 \\ 2 & -1 \end{bmatrix} = \begin{bmatrix} 129 & -18 \\ -18 & 14 \end{bmatrix},$$

$$\begin{bmatrix} \frac{7}{741} & \frac{3}{247} \\ \frac{3}{247} & \frac{43}{494} \end{bmatrix} \begin{bmatrix} -11 & 2 & 2 \\ 2 & 3 & -1 \end{bmatrix} =$$

$$\begin{bmatrix} -\frac{59}{741} & \frac{41}{741} & \frac{5}{741} \\ \frac{10}{247} & \frac{141}{494} & -\frac{31}{494} \end{bmatrix}$$

$$= \begin{bmatrix} -7.9622 \times 10^{-2} & 5.5331 \times 10^{-2} & 6.7476 \times 10^{-3} \\ 4.0486 \times 10^{-2} & 0.28543 & -6.2753 \times 10^{-2} \end{bmatrix} \begin{bmatrix} 0 \\ 7 \\ 5 \end{bmatrix}$$

$$: \begin{bmatrix} 0.42106 \\ 1.6842 \end{bmatrix}$$

### 7. Υπολογισμός $A^+$ χρησιμοποιώντας SVD

Αν $A^TA$ είναι κακής κατάστασης (ill-conditioned) ή ιδιάζων (singular), μπορούμε να χρησιμοποιήσουμε SVD για να βρούμε την ΕΤ λύση από τον τύπο:
$x = A^+b = VD_0^{-1}U^Tb$
Αν ο $A$ είναι κακής κατάστασης (ill-conditioned) ή ιδιάζων (singular), μπορούμε να χρησιμοποιήσουμε SVD για να βρούμε την ΕΤ λύση τετραγωνικών συστημάτων
$x = A^{-1}b = VD_0^{-1}U^Tb$. Αυτή είναι η ποιο αριθμητικά σταθερή μέθοδος και δουλεύει για κάθε πίνακα.

### 8. Υπολογισμός λύσης ομογενών συστημάτων

Ας υποθέσουμε ότι $b = 0$, τότε το γραμμικό σύστημα λέγεται ομογενές: $Ax = 0$
(υποθέτουμε $A$ είναι $mxn$ και $A = UDV^T$)
Η λύση με την ελάχιστη νόρμα είναι $x = 0$ (τετριμένη λύση).

Για την λύση ομογενών γραμμικών συστημάτων, μετασχηματίζουμε την λύση ΕΤ βάζοντας το περιορισμό: $\| x \| = 1$

Αυτό είναι ένα πρόβλημα βελτιστοποίησης με περιορισμούς: $\min_{\|x\|=1} \| Ax \|$

Η λύση με ελάχιστη νόρμα για ομογενή συστήματα δεν είναι μοναδική.

Ειδική περίπτωση: $rank(A) = n - 1 \quad (m \geq n - 1, \sigma_n = 0)$ η λύση είναι

$x = av_n$ ( $a$ είναι σταθερά )

Γενική περίπτωση: $rank(A) = n - k$

$(m \geq n - k, \sigma_{n-k+1} = ... = \sigma_n = 0)$ η λύση είναι

$x = a_1 v_{n-k+1} + a_2 v_{n-k-1} + ... + a_k v_n$ ( $a_i$ είναι σταθερές)

με $a_1^2 + a_2^2 + ... + a_k^2 = 1$

**Άσκηση 7:**

Υποθέστε ότι μετράμε το ύψος και το βάρος των έξι αντικειμένων και γράψτε τα δεδομένα με την μορφή γραμμών στον πίνακα **A**. Εδώ φαίνεται ο πίνακας δεδομένων Ύψους-Βάρους:

```
A = [ 47 15; 93 35; 53 15; 45 10; 67 27; 42 10 ];
```

Αλλά ίσως, είναι πιο ενδιαφέρον να δουλέψουμε με τα δικά μας δεδομένα. Ας συγκεντρώσουμε τον πίνακα κελιών Data:

```
Data = { 'Name'       'Height [cm]'  'Weight [kg]'  'Age [y]'
   'Vladimir'   172 62 30;
   'Ioan'       180 89 28;
   'Johannes'   184 77 24;
   'Mario'      178 65 25;
   'Phil'       192 93 25;
   'Markus'     181 65 23;
   'Anton'      186 80 23   }
```

Οι πίνακες κελιών είναι βολικοί, αλλά είναι δύσκολο να εξάγετε τις αριθμητικές τιμές από αυτούς. Χρησιμοποιείστε τις παρακάτω εντολές για να πάρετε τον πίνακα Height-Weight:

```
tmp = Data(2:end,2:end-1);
A = reshape([tmp{:}], size(tmp) );
```

Τα δεδομένα Height-Weight δίνονται στην κανονική βάση, δηλαδή, στο καρτεσιανό σύστημα συντεταγμένων. Η SVD βρίσκει την εναλλακτική ("καλύτερη") βάση για αυτά: οι στήλες του πίνακα **V**.

- Αναπαραστήστε τα δεδομένα Weight σαν συνάρτηση του Height.
- Υπολογίστε τον SVD του πίνακα Height-Weight **A** και αναπαραστήστε τις δύο γραμμές με τον γραμμικό συνδιασμό των δύο στηλών του πίνακα **V**. Χρησιμοποιήστε την συνάρτηση drawSpan().
- Ποιες είναι οι συντεταγμένες των δεδομένων σημείων στη βάση του **V**;
- Ποιος από την ομάδα μας έχει το μεγαλύτερο/ μικρότερο "ύψος";
- Ποιος είναι πιθανόν να είναι ένας «outlier»;

Ο ΚΩΔΙΚΑΣ ΣΑΣ ΕΔΩ:

```
Data = { 'Name'      'Height [cm]'  'Weight [kg]'  'Age [y]'
     'Vladimir'   172 62 30;
     'Ioan'       180 89 28;
     'Johannes'   184 77 24;
     'Mario'      178 65 25;
     'Phil'       192 93 25;
     'Markus'     181 65 23;
     'Anton'      186 80 23   };
% Get the Height-Weight matrix A
tmp = Data(2:end,2:end-1);
A = reshape([tmp{:}], size(tmp) );     % The height-weight matrix

A(:,1) = A(:,1) - mean(A(:,1));       % From each column substract its mean
A(:,2) = A(:,2) - mean(A(:,2));

[U,S,V] = svd(A);                     % compute SVD
PC = U*S;                             % the principle components

% Plot the data and the principle axes
figure(1); clf;
plot(A(:,1),A(:,2),'o', 'MarkerSize', 10, 'LineWidth', 2);
xlabel('Height'); ylabel('Weight');
axis equal; box on;

drawSpan(V(:,1), '2g-');          % 1st principle axes
drawSpan(V(:,2), '2m-');          % 2nd principle axes

% Display the coordinate lines
for ii=1:size(A,1)
   drawLine([A(ii,:)' PC(ii,1)*V(:,1)])
   drawLine([A(ii,:)' PC(ii,2)*V(:,2)])
end
drawAxes(2,'k');
legend('Heigth-Weight data', '1st principle axis', '2nd principle axis', 2)

% Min, max and outlier detection
disp('The principle components (coordinates):')
disp(PC)

% Find the largest/smallest person
```

```
[tmp, ix1] = max(PC(:,1));
[tmp, ix2] = min(PC(:,1));
disp('The largest/smallest persons are:')
disp(Data([ix1 ix2]+1,1))

% Find an outlier
[tmp, ix] = max(abs(PC(:,2)));
disp('The outlier is:')
disp(Data(ix+1,1) )
```

```
The principle components (coordinates):
   16.6299    3.5541
  -11.3289    6.9166
   -1.8989   -1.5140
   11.4966   -0.7644
  -19.7595   -2.5135
   10.3069   -3.5184
   -5.4460   -2.1604

The largest/smallest persons are:
    'Vladimir'
    'Phil'

The outlier is:
    'Ioan'
```

## 9. Αναφορές


1. https://sites.google.com/a/uni-konstanz.de/na09/Home/methods/fpns
2. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
3. Numerical Linear Algebra, L.N. Trefethen and David Bau, SIAM, 1997
4. Numerical Methods for Chemical Engineering, Kenneth J. Beers, Cambridge, 2007

# ΚΕΦΑΛΑΙΟ 10: ΑΡΙΘΜΗΤΙΚΗ ΕΠΙΛΥΣΗ ΜΗ- ΓΡΑΜΜΙΚΩΝ ΕΞΙΣΩΣΕΩΝ

## Περιεχόμενα



## 1. Περιγραφή του προβλήματος

Η γενική μορφή μιας γραμμικής εξίσωσης είναι $f(x)=0$ όπου $f$ είναι συνάρτηση μεταβλητής x. Παραδείγματα μη γραμμικών εξισώσεων είναι:

$$x^4+x^3+1=0$$
$$xe^{-x}=7 \text{ ή } xe^{-x}-7=0$$
$$\log x=x \text{ ή } \log x-x=0$$

Οι λύσεις της $f(x)=0,$ -δηλαδή κάθε $\alpha$ έτσι ώστε $f(\alpha)=0$ -ονομάζονται ρίζες της εξίσωσης ή μηδενικά σημεία της συνάρτησης. Μερικές μη γραμμικές εξισώσεις μπορούν να λυθούν αναλυτικά, όπως $ax^2+bx+c=0$ για τις οποίες γνωρίζουμε αναλυτικούς τύπους ( $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ ) που περιγράφουν την λύση των.

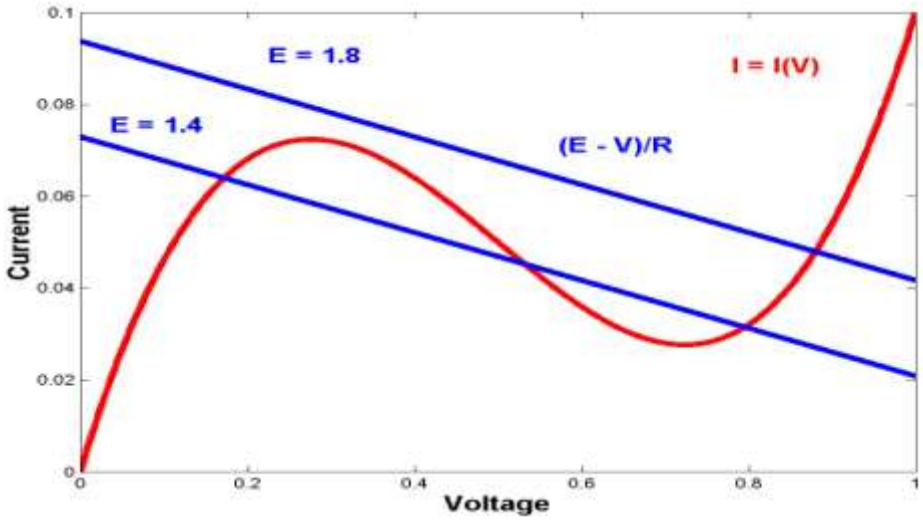
Το ίδιο ισχύει για πολυωνυμικές εξισώσεις τρίτου και τετάρτου βαθμού. Γενικά, δεν μπορούμε να βρούμε αναλυτικές λύσεις μη γραμμικών εξισώσεων και πρέπει αυτές να υπολογισθούν προσεγγιστικά με αριθμητικές μεθόδους.

Η ρίζα $\alpha$ της εξίσωσης $f(x)=0$ είναι πολλαπλότητας $m$ αν $f(x)=(x-a)^m g(x)$, όπου $g(x)$ είναι συνεχής συνάρτησης και $g(\alpha)\neq 0$. Για παράδειγμα, η εξίσωση $f(x)=x^3-2x^2+x=0$ έχει ρίζες $x=1$ πολλαπλότητας 2 και $x=0$ πολλαπλότητας 1, διότι μπορεί να γραφτεί στην μορφή $f(x)=x^3-2x^2+x=x(x-1)^2=0.$ Η $x=0$ ρίζα λέγεται και **απλή**.

**Παράδειγμα:** Θεωρείστε μια δίοδο σήραγγα που έχει την ακόλουθα χαρακτηριστικά τάσης ***(V)*** -ρεύματος ***(I):***

$$I = I(V) = V^3 - 1.5\,V^2 + 0.6\,V$$

Η δίοδος σήραγγα συνδέεται με μια αντίσταση ***(R)*** και μια πηγή τάσεως ***(E)***. Εφαρμόζοντας το νόμο τάσης του Kirchhoff μπορούμε να δούμε ότι το σταθερό ρεύμα κατά μήκος της διόδου θα πρέπει να ικανοποιεί την εξίσωση:
I(V) = ( E - V ) / R
Για δεδομένες τιμές των ***E*** και ***R***, αυτή η εξίσωση είναι της μορφής ***f(x) = 0,*** όπου πρέπει να βρούμε την τάση ***x = V.*** Αν φτιάξουμε τις δύο γραφικές παραστάσεις ***I = I(V)*** και ***I = (E-V)/R*** μαζί, μπορούμε να δούμε ότι θα έχουμε είτε τρείς είτε μία πιθανή λύση. Οι πραγματικοί υπολογισμοί των ριζών βασίζονται στους αλγόριθμους εύρεσης ριζών που θα εισάγομαι παρακάτω.




**Παράδειγμα εύρεσης ριζών πολυωνύμου με την χρήση της MatLab**

```
% simple example of root finding algorithms for polynomial
functions
R = 19.2; E = 1.4; c = [ 1,-1.5,0.6+1/R,-E/R];
% coefficients of the polynomial function f(x) = 0
p = roots(c)' % three steady currents through the tunnel
diode
E = 1.8; c = [ 1,-1.5,0.6+1/R,-E/R]; p = roots(c)' % one
(real) steady current
```

```
p =    0.7955    0.5323    0.1722
p =    0.8801    0.3099 - 0.1023i   0.3099 + 0.1023i
```

## 2. Αριθμητικές Μέθοδοι

Γενικά οι αριθμητικές μέθοδοι για αυτό το πρόβλημα κατατάσσονται σε δύο κατηγορίες:

a) αργές μέθοδοι: **διχοτόμησης, διχοτόμησης/εσφαλμένης θέσης (regular falsi),** που δουλεύουν πάντα αλλά συγκλίνουν αργά
b) ταχείς μέθοδοι: μέθοδος **του newton, μέθοδος τέμνουσας**

**Σημείωση**: Όπου είναι δυνατόν, αξίζει να κάνετε την γραφική παράσταση της συνάρτησης $f(x)$ διότι το γράφημα ή πίνακας τιμών της σας δίνει μια ιδέα για την θέση των ριζών. Για παράδειγμα, θεωρείστε την $f(x)=(\frac{x}{2})^2-\sin x=0$ . Ο παρακάτω πίνακας των τιμών της *f(x)*, δείχνει ότι υπάρχει ρίζα μεταξύ 1.8 και 2.0 αφού η συνάρτηση είναι συνεχής.

$$\begin{bmatrix} x & (\frac{x}{2})^2 & \sin x & f(x) \\ 1.6 & 0.64 & 0.996 & <0 \\ 1.8 & 0.81 & 0.976 & <0 \\ 2.0 & 1.00 & 0.909 & >0 \end{bmatrix}$$

Στο κεφάλαιο αυτό αναπτύσσουμε τις παρακάτω μεθόδους

1. Μέθοδοι της διχοτόμου και εσφαλμένης θέσης
2. Επαναληπτικές (Newton-Raphson, εφαπτομένης, σταθερού σημείου) μέθοδοι
3. Αλγόριθμοι εύρεσης ριζών με MATLAB

### α) Μέθοδος Διχοτόμησης

Η μέθοδος είναι εφαρμόσιμη για ρίζες περιττής πολλαπλότητας, δηλαδή το γράφημα της *f(x)* τέμνει τον άξονα x. Η μέθοδος παράγει μια ακολουθία από υποδιαστήματα $I_k=(a_k,b_k)$ που περιέχουν την ρίζα $\alpha$ ακολουθώντας το παρακάτω αλγόριθμο:

θεωρείστε την περίπτωση που $f(a_0)$ και $f(b_0)$ έχουν αντίθετα πρόσημα, δηλαδή $f(a_0)f(b_0)<0$ οπότε $\alpha\in I_0=(a_0,b_0)$

Για κάθε $k=1,2,3,\ldots$ $I_k$ παράγεται από το διάστημα $I_{k-1}$ ως εξής:

$m_k=\frac{a_{k-1}+b_{k-1}}{2}$ (το μέσον του $I_{k-1}$ )

αν $f(m_k)=0$ τότε $\alpha=m_k$

διαφορετικά αν $f(m_k)f(a_{k-1})>0$ , τότε $I_k=(m_k,b_{k-1})$ και $a_k=m_k,\ b_k=b_{k-1}$

διαφορετικά $I_k=(a_{k-1},m_k)$ και $a_k=a_{k-1},\ b_k=m_{k-1}$

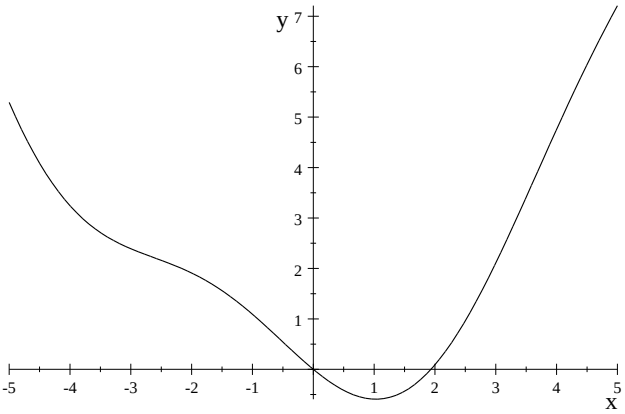
Σημειώστε ότι ισχύει $f(a_k)f(b_k)<0$ , επομένως $\alpha\in I_k$ .

Παρακάτω παραθέτουμε την υλοποίηση του αλγορίθμου σε python.

## Αλγόριθμος Διχοτόμησης σε python

```
## module bisect
' root = bisect(f,x1,x2,switch=0,tol=1.0e-9).
    Finds a root of f(x) = 0 by bisection.
    The root must be bracketed in (x1,x2).
    Setting switch = 1 returns root = None if
    f(x) increases as a result of a bisection.
'
from math import log,ceil
import error

def bisect(f,x1,x2,switch=0,epsilon=1.0e-9):
    f1 = f(x1)
    if f1 == 0.0: return x1
    f2 = f(x2)
    if f2 == 0.0: return x2
    if f1*f2 > 0.0: error.err('Root is not bracketed')
    n = ceil(log(abs(x2 - x1)/epsilon)/log(2.0))
    for i in range(n):
        x3 = 0.5*(x1 + x2); f3 = f(x3)
        if (switch == 1) and (abs(f3) >abs(f1)) \
                         and (abs(f3) > abs(f2)):
            return None
        if f3 == 0.0: return x3
        if f2*f3 < 0.0:
            x1 = x3; f1 = f3
        else:
            x2 = x3; f2 = f3
    return (x1 + x2)/2.0
```

**Παράδειγμα**: Θεωρούμε την εξίσωση $f(x)=(x/2)^2-\sin x=0$ που η γραφική της παράσταση είναι




Αν επιλέξουμε $a_0=1.5,\ b_0=2$ , τότε έχουμε $f(1.5)<0$ και $f(2)>0$ άρα η ρίζα είναι μεταξύ 1.5 και 2. Ο πίνακας δίνει τα αποτελέσματα εφαρμογής του αλγορίθμου.

| $k$ | $a_{k-1}$ | $b_{k-1}$ | $a_{k-1}-b_{k-1}$ | $m_k$ | $f(m_k)$ |
|---|---|---|---|---|---|
| 1 | 1.5 | 2 | 0.5 | 1.75 | $<0$ |
| 2 | 1.75 | 2 | 0.25 | 1.875 | $<0$ |
| 3 | 1.875 | 2 | 0.125 | 1.9375 | $>0$ |
| 4 | 1.875 | 1.9375 | 0.0625 | 1.90625 | $<0$ |
| 5 | 1.90625 | 1.9375 | 0.03125 | | |

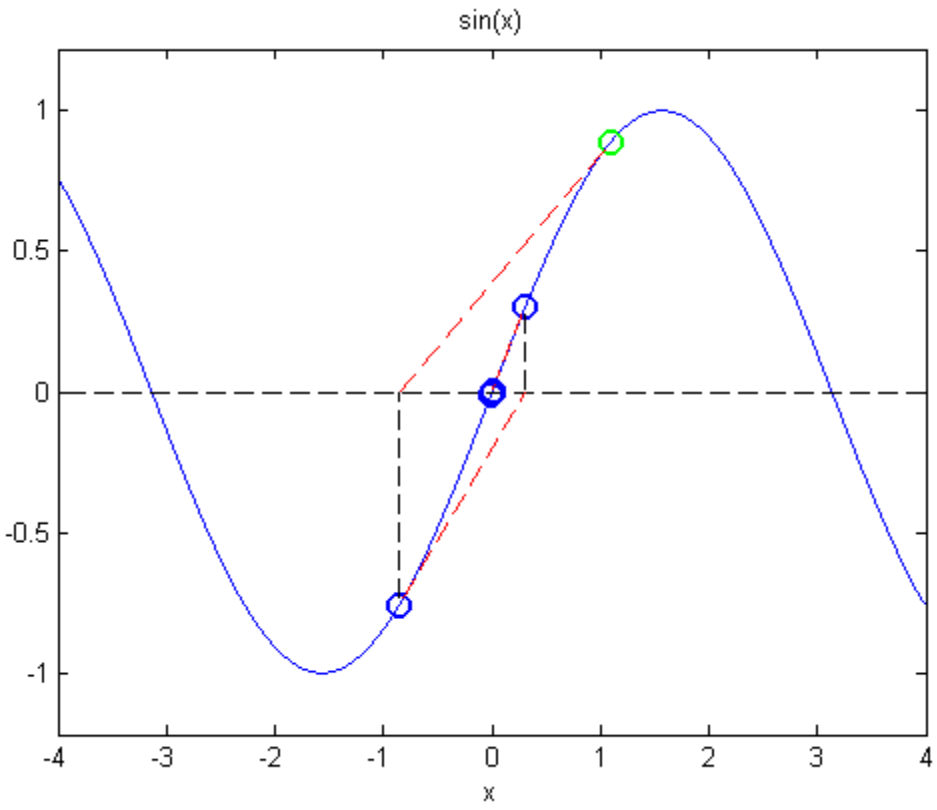
Έτσι η ρίζα είναι $\alpha = 1.90625 \pm 0.03125$ . Σημειώστε ότι η σύγκλισή της είναι βέβαιη αλλά αργή. Μετά από *n* βήματα, η ρίζα περιλαμβάνεται στο διάστημα $(a_n, b_n)$ μήκους $b_n - a_n = \frac{1}{2}(b_{n-1} - a_{n-1}) = \ldots = \frac{1}{2^n}(b_0 - a_0)$ . Το $m_{n+1}$ είναι μια προσέγγιση του $\alpha = m_{n+1} \pm \frac{1}{2^{n+1}}(b_0 - a_0)$. Έτσι το σφάλμα στο βήμα $n+1$ είναι το μισό του βήματος $n$ .

| **Πλεονεκτήματα** | **Μειονεκτήματα** |
|---|---|
| Συγκλίνει πάντα για ρίζες πολλαπλότητας περιττής | Συγκλίνει αργά |
| Εύκολος υπολογισμός του σφάλματος | Δεν βρίσκει τις ρίζες πολλαπλότητος άρτιας |
| Δεν επηρεάζεται από την πολλαπλότητα της ρίζας | |

### β) Μέθοδος Newton

**Ορισμός**: Αν ένα σημείο $x_0$ είναι κοντά στην ρίζα $\alpha$ , τότε η εφαπτομένη γραμμή της συνάρτησης $f(x)$ στο $x_0$ είναι μια καλή προσέγγιση της $f(x)$ κοντά στο $\alpha$ . Έτσι, η ρίζα της εφαπτομένης - δηλαδή το σημείο στο οποίο η εφαπτομένη συναντά το άξονα $x - x_1$ είναι καλλίτερη προσέγγιση του $\alpha$ από την τιμή $x_0$ . Σημειώστε, ότι η κλίση της εφαπτομένης δίνεται από τον τύπο $\frac{f(x_0)}{x_0 - x_1} = f'(x_0)$ . Έτσι $x_0 - x_1 = \frac{f(x_0)}{f'(x_0)} \Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$ . Επαναλαμβάνοντας αυτή την διαδικασία, υπολογίζουμε μια καλλίτερη προσέγγιση $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$ και γενικότερα για $n = 0, 1, 2, .$ $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. Αν το $x_0$ βρίσκετε κοντά στο $\alpha$ τότε $x_n \to \alpha$ καθώς $n \to \infty$ .

Το παρακάτω γράφημα απεικονίζει τις πρώτες 5 επαναλήψεις της μεθόδου Newton για την εύρεση μιας ρίζας της συνάρτησης *sin(x)*




**Παραδείγματα εφαρμογής της μεθόδου στην MatLab**

```
% Newton-Raphson method for finding the root of the
nonlinear equation:
% f(x) = x*sin(1/x)-0.2*exp(-x)
x = 0.55; % starting approximation for the root
eps = 1; tol = 10^(-14); total = 100; k = 0; format long;
while ((eps > tol) & (k < total)) % two termination
criteria
     f = x*sin(1/x)-0.2*exp(-x); % the function at x = x_k
     f1 = sin(1/x)-cos(1/x)/x+0.2*exp(-x); % the derivative
at x = x_k
  xx = x-f/f1; % a new approximation for the root
  eps = abs(xx-x); x = xx; k = k+1;
  fprintf('k = %2.0f, x = %12.10f\n',k,x);
end
```

```
k =  1, x = 0.2769143433
k =  2, x = 0.3717802027
k =  3, x = 0.3636237820
k =  4, x = 0.3637156975
k =  5, x = 0.3637157087
k =  6, x = 0.3637157087
```

```
% Example of several roots of the nonlinear equations:
```

```
% Newton-Raphson method converges to one of the roots,
% depending on the initial approximation
% f(x) = cos(x)*cosh(x) + 1 = 0 (eigenvalues of a uniform
beam)

eps = 1; tol = 10^(-14); total = 100; k = 0; x = 18;
while ((eps > tol) & (k < total))
     f    =    cos(x)*cosh(x)+1;    f1    =    sinh(x)*cos(x)-
cosh(x)*sin(x);
  xx = x-f/f1; eps = abs(xx-x); x = xx; k = k+1;
end
fprintf('k = %2.0f, x = %12.10f\n',k,x);

k =  7, x = 17.2787595321
eps = 1; k = 0; x = 9;
while ((eps > tol) & (k < total))
  f = cos(x)*cosh(x)+1; f1 = sinh(x)*cos(x)-cosh(x)*sin(x);
  xx = x-f/f1; eps = abs(xx-x); x = xx; k = k+1;
end
fprintf('k = %2.0f, x = %12.10f\n',k,x);

k =  7, x = 7.8547574382

eps = 1; k = 0; x = 5;
while ((eps > tol) & (k < total))
     f    =    cos(x)*cosh(x)+1;    f1    =    sinh(x)*cos(x)-
cosh(x)*sin(x);
  xx = x-f/f1; eps = abs(xx-x); x = xx; k = k+1;
end
fprintf('k = %2.0f, x = %12.10f\n',k,x);

k =  6, x = 4.6940911330

eps = 1; k = 0; x = 2;
while ((eps > tol) & (k < total))
     f    =    cos(x)*cosh(x)+1;    f1    =    sinh(x)*cos(x)-
cosh(x)*sin(x);
  xx = x-f/f1; eps = abs(xx-x); x = xx; k = k+1;
end
fprintf('k = %2.0f, x = %12.10f\n',k,x);

k =  5, x = 1.8751040687
```

**Αλγόριθμος Newton στην python**

```
## module newtonRaphson
```

```
''' root = newtonRaphson(f,df,a,b,tol=1.0e-9).
    Finds a root of f(x) = 0 by combining the Newton-Raphson
    method with bisection. The root must be bracketed in (a,b).
    Calls user-supplied functions f(x) and its derivative df(x).
'''
def newtonRaphson(f,df,a,b,tol=1.0e-9):
    import error
    fa = f(a)
    if fa == 0.0: return a
    fb = f(b)
    if fb == 0.0: return b
    if fa*fb > 0.0: error.err('Root is not bracketed')
    x = 0.5*(a + b)
    for i in range(30):
        fx = f(x)
        if abs(fx) < tol: return x
      # Tighten the brackets on the root
        if fa*fx < 0.0:
            b = x
        else:
            a = x; fa = fx
      # Try a Newton-Raphson step
        dfx = df(x)
      # If division by zero, push x out of bounds
        try: dx = -fx/dfx
        except ZeroDivisionError: dx = b - a
        x = x + dx
      # If the result is outside the brackets, use bisection
        if (b - x)*(x - a) < 0.0:
            dx = 0.5*(b-a)
            x = a + dx
      # Check for convergence
        if abs(dx) < tol*max(abs(b),1.0): return x
    print 'Too many iterations in Newton-Raphson'
```

### γ) Βαθμός Σύγκλισης

Έστω $x_0, x_1, x_2, ..., x_n, ...$ μια ακολουθία προσεγγίσεων της ρίζας $\alpha$ που παράγεται από μια αριθμητική μέθοδο και ισχύει $\lim_{n\to\infty} x_n = \alpha$ . Έστω $\varepsilon_n = \alpha - x_n$ . Αν $\lim_{n\to\infty} \frac{|\varepsilon_{n+1}|}{|\varepsilon_n|^p} = C$ για κάποιο $p$ και κάποια μη μηδενική σταθερά $C$ , τότε λέμε ότι η μέθοδος έχει βαθμό σύγκλισης *p* και ασυμπτωτική σταθερά σφάλματος *C*. Για την περίπτωση της μεθόδου Newton, αν υποθέσουμε ότι η $f$ είναι δύο φορές παραγωγίσημη και $\alpha$ είναι απλή ρίζα τότε έχουμε από το Θεώρημα του Taylor

$$f(\alpha) = f(x_n) + (\alpha - x_n) f'(x_n) + \frac{(-x_n)^2}{2} f''(\xi) = 0$$

όπου $\xi$ είναι ένα άγνωστο σημείο μεταξύ $x_n$ και $\alpha$ ; δηλαδή

$$\alpha - x_n + \frac{f(x_n)}{f'(x_n)} = \frac{-(\alpha - x_n)^2 f''(\xi)}{2 f'(x_n)} \Rightarrow \alpha - x_{n+1} = \frac{-(\alpha - x_n)^2 f''(\xi)}{2 f'(x_n)}$$

$$\Rightarrow \frac{|_{n+1}|}{|_n|^2} = \frac{|f''(\xi)|}{2|f'(x_n)|} \Rightarrow \lim_{n\to\infty} \frac{|_{n+1}|}{|_n|^2} = \frac{|f''(\alpha)|}{2|f'(\alpha)|}$$

Αν ισχύει $\lim_{n\to\infty} x_n = \alpha$ συμπεραίνουμε ότι $p = 2$ και $C = \frac{1}{2}\left|\frac{f''(\alpha)}{f(\alpha)}\right|$ αντίστοιχα, δηλαδή η μέθοδος Newton έχει βαθμό σύγκλισης 2.

**Σημείωση:** Αν ο βαθμός πολλαπλότητας της ρίζας $\alpha$ είναι $m$ τότε έχουμε

$$f(x) = (x-\alpha)^m g(x),\ g(\alpha) \neq 0$$

$$\Rightarrow \quad f'(x) = (x-\alpha)^m g'(x) + m(x-\alpha)^{m-1} g(x)$$
$$= (x-\alpha)^{m-1}[(x-\alpha)g'(x) + mg(x)]$$

Από όπου καταλήγουμε ότι $f'(\alpha) = 0$ εκτός αν $m = 1$ οπότε $f'(\alpha) = g(\alpha) \neq 0$ από ορισμό. Για να έχουμε τετραγωνική συμπεριφορά σύγκλισης στην Newton θα πρέπει η ρίζα να είναι απλή. Διαφορετικά

$$x_{n+1} - \alpha = x_n - \alpha - \frac{f(x_n)}{f'(x_n)}$$
$$= x_n - \alpha - \frac{(x_n-\alpha)g(x_n)}{(x_n-\alpha)g'(x_n)+mg(x_n)}$$
$$\frac{|x_{n+1}-\alpha|}{|x_n-\alpha|} = \left|1 - \frac{g(x_n)}{(x_n-\alpha)g'(x_n)+mg(x_n)}\right|$$

και

$$\lim_{n\to\infty} \frac{|_{n+1}|}{|_n|} = \left|1 - \frac{g(\alpha)}{mg(\alpha)}\right| = 1 - \frac{1}{m} = \frac{m-1}{m}\ ,\ m \geqslant 2$$

επομένως υποθέτοντας ότι $\lim_{n\to\infty} x_n = \alpha$. Έτσι ο βαθμός σύγκλιση της μεθόδου Newton είναι 1 (γραμμική) για $m > 1$ και καθώς $m$ μεγαλώνει $\frac{m-1}{m} \to 1$.

### δ) Παραλλαγή της μεθόδου Newton για ρίζες με πολλαπλότητα $m > 1$

Παρατηρούμε ότι η συνάρτηση $q(x) = \frac{f(x)}{f'(x)}$ έχει πάντα απλές ρίζες διότι

$$q(x) = \left(\frac{(x-\alpha)g(x)}{(x-\alpha)g'(x)+mg(x)}\right)$$

και

$$q'(x) = \frac{[mg(x)+g'(x)(x-\alpha)]\left[g(x)+g'(x)(x-\alpha)\right]-(x-\alpha)g(x)[(x-\alpha)g''(x)+(m+1)g'(x)]}{\left(mg(x)+g'(x)(x-\alpha)\right)^2}$$

Σημειώστε ότι $q(\alpha) = 0$ και $q'(\alpha) \neq 0$ (Γιατί;).

Αν εφαρμόσουμε την μέθοδο Newton στην συνάρτηση $q(x)$ τότε θα συγκλίνει τετραγωνικά ( $p = 2$ ) και η μέθοδος περιγράφεται από τις εξισώσεις

$$x_{n+1} = x_n - \frac{q(x_n)}{q'(x_n)}$$

$$q'(x) = \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2}$$

$$\Rightarrow \frac{q(x)}{q'(x)} = \frac{f(x)f'(x)}{f'(x)^2 - f(x)f''(x)}$$

και

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{f'(x_n)^2 - f(x_n)f''(x_n)}$$

Έτσι έχουμε μια μέθοδο που συγκλίνει τετραγωνικά για κάθε ρίζα ανεξαρτήτως της πολλαπλότητας της αλλά απαιτεί να γνωρίζουμε την δεύτερη παράγωγο της $f$.

**Άσκηση**: Αν $f(x) = k(x-\alpha)^m$ για κάποια σταθερά $k$ τότε η Newton επαναληπτική εξίσωση είναι

$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}$

και $m$ βρίσκεται κατά προσέγγιση από τον ασυμπτωτικό τύπο

$$\frac{|\varepsilon_{n+1}|}{|\varepsilon_n|} \approx \frac{m-1}{m}$$

**Παράδειγμα**

$f(x) = x - \cos x \quad (a = 0.739085)$

$f'(x) = \sin x + 1$

$x_{n+1} = x_n - \frac{x_n - \cos x_n}{\sin x_n + 1} = \frac{1}{\sin x_n + 1}(\cos x_n + x_n \sin x_n)$

| $n$ | $x_n$ | $\varepsilon_n = \alpha - x_n$ | $x_{n+1} - x_n$ |
|---|---|---|---|
| 0 | 0 | 0.739085 | 1 |
| 1 | 1 | –0.260915 | –0.249636 |
| 2 | 0.750364 | –0.011279 | –0.011251 |
| 3 | 0.739113 | –0.000028 | –0.000028 |
| 4 | 0.739085 | 0 | 0 |
| 5 | 0.739085 | 0 | |

Παρατηρήστε ότι

$\frac{|\varepsilon_1|}{|\varepsilon_0|^2} = 0.47765$

$\frac{|\varepsilon_2|}{|\varepsilon_1|^2} = 0.165681$

$\frac{|\varepsilon_3|}{|\varepsilon_2|^2} = 0.220098$

και ότι το $x_{n+1} - x_n$ είναι μια καλή προσέγγιση του σφάλματος $x_n - \alpha$ .

**Παράδειγμα**

$f(x) = x^2 - 4x + 4 = 0 \quad a = 2 \quad m = 2$

$f'(x) = 2x - 4$

Μέθοδος Newton: $x_{n+1} = x_n - \frac{x_n^2 - 4x_n + 4}{2x_n - 4} = \frac{1}{2}x_n + 1$

| $n$ | $x_n$ | $\varepsilon_n = \alpha - x_n$ | $x_{n+1} - x_n$ |
|---|---|---|---|
| 0 | 1 | 1 | 0.5 |
| 1 | 1.5 | 0.5 | 0.25 |
| 2 | 1.75 | 0.25 | 0.125 |
| 3 | 1.875 | 0.125 | 0.0625 |
| 4 | 1.9375 | 0.0625 | |

Παρατηρούμε γραμμική σύγκλιση αφού

$$\left|\frac{n+1}{n}\right| = \left|\frac{\alpha - x_{n+1}}{\alpha - x_n}\right| = \frac{1}{2} = \left|\frac{x_{n+1} - x_n}{x_n - x_{n-1}}\right|$$

**Άσκηση:** Να εφαρμοσθεί η Newton για πολλαπλές ρίζες.
**Παράδειγμα**

$f(x) = x^4 - 2x^3 + 2x - 1 = 0 \quad \alpha = 1$
$f'(x) = 4x^3 - 6x^2 + 2$

και εφαρμόζοντας Newton παίρνουμε

$$x_{n+1} = x_n - \frac{x^4 - 2x^3 + 2x - 1}{4x^3 - 6x^2 + 2} = \frac{1}{4x+2}(-x^2 + 4x_n x + 2x_n + 1)$$

| $n$ | $x_n$ | $\varepsilon_n = \alpha - x_n$ | $x_{n+1} - x_n$ | $\left\|\frac{\alpha - x_{n+1}}{\alpha - x_n}\right\|$ | $\left\|\frac{x_{n+1} - x_n}{x_n - x_{n-1}}\right\|$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.5 | 0.5 | 0.375 |
| 1 | 0.5 | 0.5 | 0.1875 | 0.625 | 0.59227 |
| 2 | 0.6875 | 0.3125 | 0.11102 | 0.64474 | 0.62836 |
| 3 | 0.79852 | 0.20148 | 0.06976 | 0.65376 | 0.64464 |
| 4 | 0.86828 | 0.13172 | 0.04497 | 0.65859 | 0.65288 |
| 5 | 0.91325 | 0.08675 | 0.02936 | 0.66156 | |
| 6 | 0.94261 | 0.05739 | | | |

Εδώ δεν γνωρίζουμε την πολλαπλότητα της ρίζας! Μπορούμε όμως να την υπολογίσουμε από το τύπο $\frac{m-1}{m} \sim \frac{2}{3} \to m = 3$ . Τώρα μπορούμε να εφαρμόσουμε την παραλλαγή Newton και να πάρουμε

$$x_{n+1} = x_n - 3\frac{f(x_n)}{f'(x_n)}$$
$$= x_n - 3\frac{x^4 - 2x^3 + 2x - 1}{4x^3 - 6x^2 + 2} = \frac{1}{4x+2}(-3x^2 + 4x_n x + 2x_n + 3)$$

η οποία μας δίνει μια γρήγορη σύγκλιση όπως ο παρακάτω πίνακας υποδεικνύει.

| $n$ | $x_n$ |
|---|---|
| 1 | 1.5 |
| 2 | 1.03125 |
| 3 | 1.00016 |
| 4 | 0.99993 |
| 5 | 0.99961 |
| 6 | 1.00006 |
| 7 | 1.00000 |

### ε) Η μέθοδος τέμνουσας

Η μέθοδος μπορεί να θεωρηθεί ως μια παραλλαγή της Newton αν αντικαταστήσουμε την παράγωγο $f'(x)$ με την προσέγγιση της $\frac{f(x_{n+1})-f(x_n)}{x_{n+1}-x_n}$. Δηλαδή αντί να προσεγγίσουμε την $f(x)$ με την εφαπτομένη στο $x_n$ προσεγγίζουμε την συνάρτηση με την τέμνουσα που διέρχεται από τα σημεία $(x_n, f(x_n))$ και $(x_{n-1}, f(x_{n-1}))$. Αν εξισώσουμε τις δύο διαφορετικές εκφράσεις για την κλίση της τέμνουσας λαμβάνουμε την επαναληπτική εξίσωση

$$\frac{f(x_n)}{x_n-x_{n-1}} = \frac{f(x_n)-f(x_{n-1})}{x_n-x_{n-1}} \Rightarrow x_{n+1} = x_n - \frac{(x_n-x_{n-1})f(x_n)}{f(x_n)-f(x_{n-1})}$$

Μπορούμε να αποδείξουμε ότι ο βαθμός σύγκλισης είναι 1.618 - λίγο μικρότερος της newton - για απλές ρίζες.

Το πλεονέκτημα της είναι το γεγονός ότι δεν απαιτεί γνώση της παραγώγου της $f$ αλλά απαιτεί δύο αρχικές προσεγγίσεις της ρίζας για να δουλέψει. Όπως και στη περίπτωση της Newton η σύγκλιση εξαρτάται από τις αρχικές τιμές και η σύγκλιση της επιβραδύνει για ρίζες με πολλαπλότητα > 1. Η μέθοδος θα μπορούσε να συνδυαστεί με την μέθοδο της διχοτόμησης και την παραλλαγή της για να βρεθούν καλές αρχικές προσεγγίσεις της ρίζας. Αν μπορούμε να προσεγγίσουμε την πολλαπλότητα της ρίζας τότε ο παρακάτω τύπος δίνει μια βέλτιστη μέθοδο για πολλαπλές ρίζες.

$$x_{n+1} = x_n - m\frac{(x_n-x_{n-1})f(x_n)}{f(x_n)-f(x_{n-1})}$$

**MatLab παράδειγμα**

```
% Secant method for finding the root of the nonlinear equation:
% f(x) = x*sin(1/x)-0.2*exp(-x)
eps = 1; tol = 10^(-14); total = 100; k = 1; format long;
x = 0.55; f = x*sin(1/x)-0.2*exp(-x); % starting approximation for the root
x0 = x; f0 = f; x = x + 0.01;
```

```
while ((eps > tol) & (k < total)) % two termination
criteria
     f = x*sin(1/x)-0.2*exp(-x); % the function at x = x_k
     xx = x-f*(x-x0)/(f-f0); % a new approximation for the
root
  eps = abs(xx-x); x0 = x; f0 = f; x = xx; k = k+1;
  fprintf('k = %2.0f, x = %12.10f\n',k,x);
end
k =  2, x = 0.2715890602
k =  3, x = 0.3878094689
k =  4, x = 0.3650168333
k =  5, x = 0.3636699429
k =  6, x = 0.3637157877
k =  7, x = 0.3637157087
k =  8, x = 0.3637157087
k =  9, x = 0.3637157087
```

### ζ) Μέθοδος διχοτόμησης/εσφαλμένης θέσης (regular falsi)

Αυτή η μέθοδος είναι παραλλαγή της μεθόδου διχοτόμησης όπου αντί να βρίσκουμε το μέσον $m_{k+1}$ του διαστήματος $I_k=(a_k,b_k)$ βρίσκουμε την τομή του άξονα του $x$ με την τέμνουσα που διέρχεται από τα σημεία $(a_k, f(a_k))$ και $(b_k, f(b_k))$. Στην περίπτωση αυτή

$$m_{k+1}=\frac{a_k f(b_k)-b_k f(a_k)}{f(b_k)-f(a_k)}$$

Σημειώστε ότι όπως και στην περίπτωση της μεθόδου διχοτόμησης η μέθοδος δουλεύει για ρίζες πολλαπλότητας περιττής και συγκλίνει γραμμικά.

### η) Μέθοδος σταθερού σημείου

Μια μη γραμμική εξίσωση $f(x)=0$ μπορεί να γραφτεί στην μορφή $x=\varphi(x)$ με πολλούς τρόπους. Στην περίπτωση αυτή η ρίζα $\alpha$ της $f(x)=0$ είναι το σταθερό σημείο της $x=\varphi(x)$ δηλαδή $\alpha=\varphi(\alpha)$ . Έτσι, η επαναληπτική μέθοδος που ορίζεται από την εξίσωση $x_{n+1}=\varphi(x_n)$ συγκλίνει κάτω από ορισμένες συνθήκες. Για παράδειγμα $x^3-2=0$ μπορεί να γραφεί

a) $x=x^3+x-2$

b) $x=\frac{2+5x-x^3}{5}$

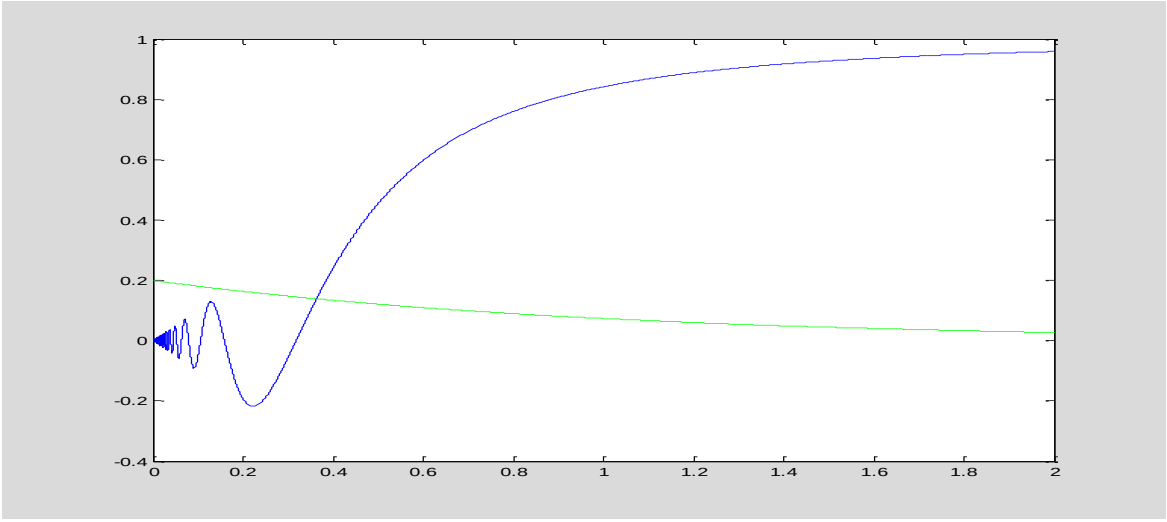
Αν επιλέξουμε $x_0=1.2$ παίρνουμε δύο διαφορετικές προσεγγίσεις

$$\begin{bmatrix} n & (a) & (b) \\ 1 & 0.928 & 1.2544 \\ 2 & -0.273 & 1.2596 \\ 3 & -2.293 & 1.2599 \\ 4 & -16.349 & 1.25992 \end{bmatrix}$$

Η (a) δεν συγκλίνει ενώ η (b) συγκλίνει. Από τις παρακάτω εξισώσεις μπορούμε να δούμε κάτω από ποιες συνθήκες η μέθοδος συγκλίνει.

$$\begin{aligned} & x_1 = \varphi(x_0) \\ \Rightarrow\ & \alpha - x_1 = \alpha - \varphi(x_0) = \varphi(\alpha) - \varphi(x_0) \\ & \qquad = \varphi'(\xi_0)(\alpha - x_0),\ x_0 \le \xi_0 \le \alpha \\ & \alpha - x_2 = \varphi'(\xi_1)(\alpha - x_1) \\ & \qquad = \varphi'(\xi_0)\varphi'(\xi_1)(\alpha - x_0),\ x_1 \le \xi_1 \le \alpha \\ \Rightarrow\ & \alpha - x_n = \varphi'(\xi_0)\varphi'(\xi_1)...\varphi'(\xi_{n-1})(\alpha - x_0) \end{aligned}$$

```
% Graphical solution of the nonlinear equation: x*sin(1/x)
= 0.2*exp(-x)
x = 0.0001 : 0.0001 : 2; f1 = x.*sin(1./x); f2 = 0.2*exp(-
x); plot(x,f1,'b',x,f2,'g');
```




```
% Contraction mapping method for finding the root of the
nonlinear equation:
% f(x) = x*sin(1/x)-0.2*exp(-x)
x = 0.364; % starting approximation for the root
eps = 1; tol = 0.0001; total = 8; k = 0;
```

```
while ((eps > tol) & (k < total)) % two termination
criteria
  xx = x + x*sin(1/x)-0.2*exp(-x); % a new approximation
for the root
  eps = abs(xx-x); x = xx; k = k+1;
  fprintf('k = %2.0f, x = %6.4f\n',k,x);
end
stability = abs(1 + sin(1/x)-cos(1/x)/x+0.2*exp(-x));
if (stability >= 1)
     fprintf('The contraction mapping method is UNSTABLE');
else
     fprintf('The contraction mapping method is STABLE\n');
     fprintf('The numerical approximation for the root: x =
%6.4f',x);
end
k =  1, x = 0.3649
k =  2, x = 0.3684
k =  3, x = 0.3827
k =  4, x = 0.4391
k =  5, x = 0.6442
k =  6, x = 1.1832
k =  7, x = 2.0070
k =  8, x = 2.9393
The contraction mapping method is UNSTABLE
```

Επομένως, αν $|\varphi'(\xi_k)| \le M$ για όλα τα $k$, τότε $|\varepsilon_n| \le M^n |\varepsilon_0|$ και η σύγκλιση στην περιοχή του $\alpha$ και $x_0$ προϋποθέτει ότι $M<1$ δηλαδή $|\varphi'(x)|<1$. Η σύγκλιση της μεθόδου είναι γραμμική

$$x_{n+1}-\alpha = \varphi(x_n)-\varphi(\alpha) = \varphi'(\xi)(x_n-\alpha)$$
$$\Rightarrow \lim_{n\to\infty}\left|\frac{n+1}{n}\right| = |\varphi'(\alpha)|$$

Προτιμούμε την Newton αν είναι εφαρμόσιμη. Αν επιλέξουμε φ(x) έτσι ώστε $\varphi^p(\alpha)\neq 0$ και

$$\varphi'(\alpha)=\varphi''(\alpha)=\ldots=\varphi^{(p-1)}(\alpha)=0$$

τότε

$$x_{n+1}-\alpha = \varphi(x_n)-\varphi(\alpha)$$
$$= \varphi(\alpha)+(x_n-\alpha)\varphi'(\alpha)+\frac{(x_n-\alpha)^2}{2}\varphi''(\alpha)+\ldots+\frac{(x_n-\alpha)^{p-1}}{(p-1)!}\varphi^{(p-1)}(a)+$$

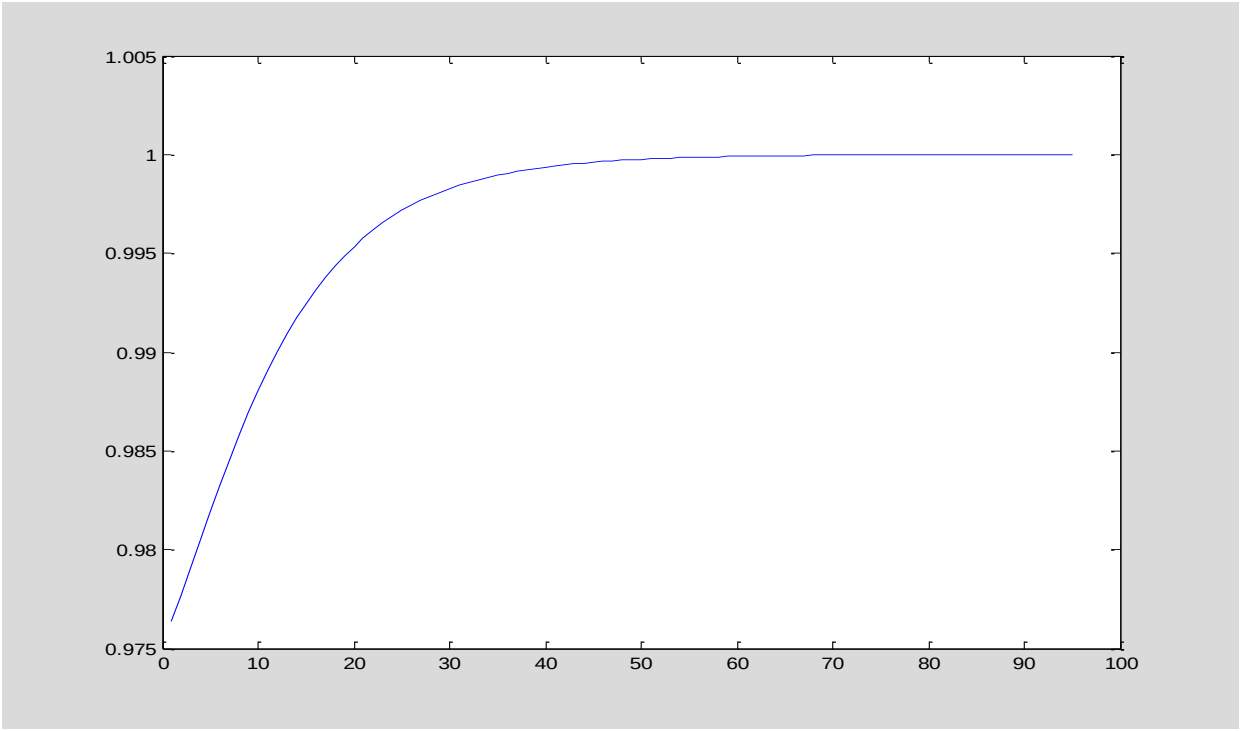
$$+\frac{(x_n-\alpha)^p}{p!}\varphi^{(p)}(\xi)-\varphi(\alpha)$$
$$=\frac{(x_n-\alpha)^p}{p!}\varphi^{(p)}(\xi)$$
$$\Rightarrow \lim_{n\to\infty}\left|\frac{n+1}{n}\right| = \frac{\left|\varphi^{(p)}(\alpha)\right|}{p!}$$

δηλαδή ο βαθμός σύγκλησης της μεθόδου είναι p.

**Παράδειγμα**

```
R = 19.2; E = 1.4;
x = 0.4; % starting approximation for the root
eps = 1; tol = 0.000001; total = 10000; k = 0;
while ((eps > tol) & (k < total)) % two termination
criteria
  xx = x + x^3-1.5*x^2+0.6*x-E/R+x/R; % a new approximation
for the root
  eps = abs(xx-x); x = xx; k = k+1; e(k) = x-0.880133;
end
stability = abs(1 + 3*x^2-3*x+0.6+1/R);
if (stability >= 1)
     fprintf('The contraction mapping method is UNSTABLE');
else
     fprintf('The contraction mapping method is STABLE\n');
     fprintf('The numerical approximation for the root: x =
%8.6f',x);
end
ee = e(2:k)./e(1:k-1); plot(ee);
```

```
The contraction mapping method is STABLE
The numerical approximation for the root: x = 0.532249
```




## 3. Συστήματα μη γραμμικών εξισώσεων

Έστω οι μη γραμμικές εξισώσεις $\vec{f}(\vec{x}) = 0$ όπου

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix} \qquad \vec{f}(\vec{x}) = \begin{pmatrix} f_1(x_1, x_2, ..., x_n) \\ f_2(x_1, x_2, ..., x_n) \\ . \\ . \\ . \\ f_n(x_1, x_2, ..., x_n) \end{pmatrix}$$

Στην περίπτωση αυτή η μέθοδος του Newton γενικεύεται ως εξής:

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - J^{-1}(\vec{x}^{(k)}) f(\vec{x}^{(k)})$$

όπου $J^{-1}$ είναι ο αντίστροφος του Jacobi πίνακα

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & . & . & . & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & . & . & . & \frac{\partial f_2}{\partial x_n} \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & . & . & . & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

Στην περίπτωση αυτή πρέπει να λύσουμε μια ακολουθία γραμμικών συστημάτων

$$J(\vec{x}^{(k)})\delta\vec{x}^{(k)} = -f(\vec{x}^{(k)})$$

όπου

$$\delta\vec{x}^{(k)} = \vec{x}^{(k+1)} - \vec{x}^{(k)}$$

υποθέτοντας μια αρχική προσέγγιση $x^{(0)}$ .

**Παράδειγμα υπολογισμών με Python και SciPy**

Για να βρούμε τις ρίζες ενός πολυωνύμου, η εντολή **roots** μπορεί να χρησιμοποιηθεί. Για να βρεθεί η ρίζα ενός μη-γραμμικού συστήματος, υπάρχει η εντολή **fsolve**. Στο παράδειγμα, βρίσκομαι τις ρίζες της εξίσωσης

$x + 2\cos(x) = 0$,

και το σύνολο των μη-γραμμικών εξισώσεων

$x_0 \cos(x_1) = 4,$

$x_0 x_1 - x_1 = 5.$

Τα αποτελέσματα είναι

$x = -1.0299$ και $x_0 = 6.5041,\ x_1 = 0.9084$

ΚΩΔΙΚΑΣ

```
>>> from numpy import *
>>> def func(x):
...     return x + 2*cos(x)
>>> def func2(x):
...     out = [x[0]*cos(x[1]) - 4]
...     out.append(x[1]*x[0] - x[1] - 5)
...     return out
>>> from scipy.optimize import fsolve
>>> x0 = fsolve(func, 0.3)
>>> print x0
-1.02986652932
>>> x02 = fsolve(func2, [1, 1])
>>> print x02
[ 6.50409711  0.90841421]
```

## 4. MATLAB συναρτήσεις για την εύρεση ριζών για μη-γραμμικές συναρτήσεις

- **fzero**: εντοπίζει μια ρίζα μη-γραμμικής συνάρτησης ξεκινώντας με μια αρχική προσέγγιση
- **fminbnd:** ψάχνει για το ελάχιστο μιας μη γραμμικής συνάρτησης σε ένα πεπερασμένο διάστημα
- **fminsearch:** ψάχνει για το ελάχιστο μιας μη-γραμμικής συνάρτησης με μια αρχική προσέγγιση (χρησιμοποιώντας τη μέθοδο simplex των Nelder-Mead της άμεσης έρευνας)

```
% MATLAB algorithn for finding the root of the nonlinear
equation:
% f(x) = x*sin(1/x)-0.2*exp(-x)
x0 = 0.55; f = 'x*sin(1/x)-0.2*exp(-x)';
x = fzero(f,x0);
fprintf('The root is found at x = %12.10f\n',x);

The root is found at x = 0.3637157087

 % Search of zero of f(x) is the same as the search of minimum
for [f(x)]^2
x = fminbnd('(x*sin(1/x)-0.2*exp(-x))^2',0.2,0.5)
x = fminsearch('(x*sin(1/x)-0.2*exp(-x))^2',0.3)
opt = optimset('TolX',10^(-10)); % termination tolerance on the
value of x
x = fminbnd('(x*sin(1/x)-0.2*exp(-x))^2',0.2,0.5,opt)

x =   0.36371013129529
x =   0.36369140625000
x =   0.36371570742172

% The search fails when no roots are found
% The function y = 1 + x^2 has no zeros
x = fzero('1 + x^2',1)

Exiting fzero: aborting search for an interval containing a sign
change
    because NaN or Inf function value encountered during search
(Function value at -1.716199e+154 is Inf)
Check function or try again with a different starting value.
x =   NaN
```

## 5. Ασκήσεις

### Άσκηση 1: `nlnewton()` και τετραγωνική σύγκλιση

Η μέθοδος υλοποιείται στην συνάρτηση `nlnewton()` που παρατίθεται στο τέλος της εργασίας. Η συνάρτηση απαιτεί 3 υποχρεωτικά ορίσματα: τα δύο πρώτα ορίζουν `f(x)` και `f'(x)` και το τρίτο την εκτίμηση σας για την αρχική προσέγγιση της ρίζας `x0`.

α) Τα σχόλια της συνάρτησης `nlnewton()` περιλαμβάνουν ένα παράδειγμα για τον υπολογισμό της `sqrt(2)`. Τρέξτε αυτό το παράδειγμα. Ποιος είναι ο αριθμός των ακριβή ψηφίων που παίρνετε;

β) Το 4ο προαιρετικό όρισμα της `nlnewton()` είναι μια συνάρτηση που ορίζει ο χρήστης και καλείται σε κάθε επανάληψη. Η συνάρτηση παίρνει 4 παραμέτρους εισόδου:

αριθμό επαναλήψεων, την τρέχουσα τιμή του `x0`, υπολογισμένη `dx=x1-x0` και το όνομα μη-γραμμικής συνάρτησης. Ένα παράδειγμα της συνάρτησης, `IterFcn.m` είναι:

```
function IterFcn(nit,x0,dx,nlf)
% User defined function that is called at each iteration of
nlnewton().
%
% Usage: IterFcn(nit,x0,dx,nlf)
% INPUT:
% nit       - iteration number;
% x0        - current x0 value;
% dx        - x-increment computed at the iteration;
% nlf       - handle to the nonlinear function.

if nit==0, clc; end
disp([nit,x0+dx,nlf(x0+dx)])
```

Τρέξτε τον παρακάτω κώδικα και παρατηρείστε την ***τετραγωνική σύγκλιση***:

```
format long
format compact
f=@(x)         x.^2-2;         J=@(x)         2*x;         x0=1;
x=nlnewton(f,J,x0,@IterFcn);
```

Ο ΚΩΔΙΚΑΣ ΣΑΣ ΕΔΩ:

```
   0   1  -1
  1.000000000000000  1.500000000000000  0.250000000000000
  2.000000000000000  1.416666666666667  0.006944444444445
  3.000000000000000  1.414215686274510  0.000006007304883
  4.000000000000000  1.414213562374690  0.000000000004511
  5.000000000000000  1.414213562373095  0.000000000000000
# of iterations: 5
```

**Άσκηση 2: Οπτικοποίηση σύγκλισης**

Τροποποιείστε την συνάρτηση `IterFcn.m`, έτσι ώστε

- Στην 0 επανάληψη, να δημιουργεί ένα γράφημα της δοσμένης συνάρτησης και του αρχικού σημείου (`x0, f(x0)`) εκκίνησης. Υπόδειξη: χρησιμοποιείστε την συνάρτηση `ezplot(f,[x1 x2])` για να κάνετε το γράφημα της `f(x)` στο διάστημα `[x1 x2]`;
- Στις επόμενες επαναλήψεις, να αναπαριστά τις εφαπτόμενες γραμμές του διαγράμματος της `f(x)` στο `x0` που διασταυρώνονται με τον άξονα x και την νέα τιμή (`x1, f(x1)`).

Με την νέα `IterFcn()`, παρατηρείστε την διαδικασία επίλυσης των παρακάτω συναρτήσεων:

a) `f(x)=x.^2-2` στο διάστημα `[0 10]` για αρχικές τιμές `x0=1,.5,.1,0`;
b) `f(x)=sin(x)` στο διάστημα `[-4 4]`. Δοκιμάστε τις πέντε αρχικές τιμές: `x0=1.1:.1:1.5`. Ο αλγόριθμος συγκλίνει πάντα στο κοντινότερο μηδέν;
c) `f(x)=x.^3-3*x.^2+5` στο διάστημα`[-4 4]`για `x0=-3:.1:3`. Η σύγκλιση γίνεται πάντα με ομαλό τρόπο; Ποιο είναι το πρόβλημα με το σημείο $x_0$=1?
d) `f(x)=sign(x-2).*sqrt(abs(x-2))` στο διάστημα `[0 4]` με `x0=1`. Η μέθοδος Newton συγκλίνει; αποκλίνει; Γιατί;

Ο ΚΩΔΙΚΑΣ ΣΑΣ ΕΔΩ:

**Άσκηση 3: Μέθοδος τέμνουσας (secant)**

Υλοποιείστε την μέθοδο στην συνάρτηση `nlsecant()` τροποποιώντας τον κώδικα `nlnewton()`. Η συνάρτηση σας θα απαιτεί μόνο 2 υποχρεωτικά ορίσματα: την συνάρτηση και τα σημεία εκκίνησης.

Εφαρμόστε την `nlsecant()` στην συνάρτηση της προηγούμενης άσκησης. Συγκρίνετε η σύγκλιση με αυτή της `nlnewton()` ?

Ο ΚΩΔΙΚΑΣ ΣΑΣ ΕΔΩ:

**Άσκηση 4: MATLAB `fzero()`**

Η συνάρτηση του `fzero()` βρίσκει την ρίζα μιας πραγματικής συνάρτησης με έναν συνδυασμό των μεθόδων secant, διχοτόμησης (*bisection)* και αντίστροφη τετραγωνική παρεμβολή (*inverse quadratic interpolation).* Χρησιμοποιείστε `fzero(f,x0)` για να βρείτε το `sqrt(1)` με σημεία εκκίνησης `x0=10` και `x0=11`. Τι παρατηρείτε;

YOUR CODE HERE:

**Άσκηση 5: πολυδιάστατη μέθοδος Newton**

Τροποποιείστε την συνάρτηση `nlnewton.m` για να βρίσκει το μηδέν μιας συνάρτησης με διανύσματα. Υπόδειξη: χρειάζεται να τροποποιήσετε δύο γραμμές μόνο.

Η βοήθεια του `nlnewton()` περιλαμβάνει ένα παράδειγμα εφαρμογής της μεθόδου για σύστημα μη-γραμμικών εξισώσεων 2-επί-2. Ελέγξτε τις συναρτήσεις σας σε αυτό το παράδειγμα. Δοκιμάστε διαφορετικά αρχικά διανύσματα, π.χ. ,`x0=[1 2]'` .

Πως συμπεριφέρεται η μέθοδος για το παρακάτω σύστημα;

$x_1^3 - x_2 = 0,$

$x_2^3 - x_1 = 0$

Ο ΚΩΔΙΚΑΣ ΣΑΣ ΕΔΩ:

**Άσκηση 6: μέθοδος Newton**

Καθένα από τα παρακάτω συστήματα μη-γραμμικών εξισώσεων μπορεί να παρουσιάζουν κάποια δυσκολία στον υπολογισμό της λύσης. Χρησιμοποιήστε την συνάρτηση `nlnewton()` για να επιλύσετε τα συστήματα για το δοσμένο σημείο εκκίνησης. Ο μέγιστος αριθμός των επαναλήψεων να είναι 100.

Σε μερικές περιπτώσεις, η δική σας –συνάρτηση μπορεί να αποτυγχάνει και να μην συγκλίνει ή μπορεί να συγκλίνει σε ένα άλλο σημείο και όχι στην λύση. Όταν αυτό συμβαίνει, προσπαθήστε να εξηγήσετε τον λόγο αυτής της συμπεριφοράς.

a)

$$x_1 + x_2(x_2(5 - x_2) - 2) = 13,$$
$$x_1 + x_2(x_2(1 + x_2) - 14) = 29,\ x_0 = [15, -2]^T$$

b)

$$x_1^2 + x_2^2 + x_3^2 = 5,$$
$$x_1 + x_2 = 1,$$
$$x_1 + x_3 = 3,\ x_0 = [\frac{1+\sqrt{3}}{2}, \frac{1-\sqrt{3}}{2}, \sqrt{3}]^T$$

c)

$$x_1 + 10x_2 = 0,$$
$$\sqrt{5}(x_3 - x_4) = 0,$$
$$(x_2 - x_3)^2 = 0,$$
$$\sqrt{10}(x_1 - x_4)^2 = 0,\ x_0 = [1, 2, 1, 1]^T$$

d)

$$x_1 = 0,$$
$$\frac{10x_1}{x_1 + 0.1} + 2x_2^2 = 0, x_0 = [1.8, 0]^T$$

e)

$$x_1 = 0,$$
$$\frac{10x_1}{x_1 + 0.1} + 2x_2^2 = 0, x_0 = [1.8, 0]^T$$

```
function [x, nit] = nlnewton(nlfc, J, x0, itfc)
% Find zero of a nonlinear function by the Newton's method.
%
% Usage:   x        = nlnewton(nlfc, J, x0)
%        [ x, nit ] = nlnewton(nlfc, J, x0)
% INPUT:
% nlfc     - handle to the nonlinear function, f(x);
% J          - handle to the function computing functions
Jacobian matrix, A = J(x);
% x0       - initial guess vector;
% (optional:)
%  itfc        - an auxiliary function executed at every
iteration step,
% OUTPUT:
% x        - n-by-1 vector solving f(x)=0;
% (optional:)
```

```
%  nit           -  number  of  iterations  performed  until
convergence;
%
% Examples:
% 1D:
%  f=@(x) x.^2-2; J=@(x) 2*x; x0=1; x=nlnewton(f,J,x0);
% 2D:
%  f = @(x) [x(1)+sin(x(2)); x(1)*cos(x(2))+x(2)];
%  J = @(x) [1 cos(x(2)); cos(x(2)) -x(1)*sin(x(2))+1 ];
%  x0 = [1 1]';
%  x = nlnewton(f,J, x0);
%
% See also: nlsecant, nlbreuden.

% Input check:
error(nargchk(3,4,nargin));

% Defaults:
maxIt = 100;      % maximum number of iterations
th  = 1e-14;      % threshold
nit = 0;          % iterations counter
if nargin<4, itfc = []; end;

% Start iterations:
x = x0;
dx = 0;
if ~isempty(itfc), itfc(nit,x,dx,nlfc); end
while abs(nlfc(x)) > th
    nit = nit + 1;
    if nit > maxIt
        warning(['Convergence  failed:  maximum  number  of
iterations (' num2str(maxIt) ') was exceeded.']);
        return;
    end
    dx = -nlfc(x)/J(x);
    if ~isempty(itfc), itfc(nit,x,dx,nlfc); end
    x = x + dx;
end

if nargout < 2
    display( [ '# of iterations: ' num2str(nit)] )
end

function IterFcn(nit,x0,dx,nlf)
% User defined function that is called at each iteration of
nlnewton().
%
```

```
% Usage: IterFcn(nit,x0,dx,nlf)
% INPUT:
% nit       - iteration number;
% x0        - current x0 value;
% dx        - x-increment computed at the iteration;
% nlf       - handle to the nonlinear function.

if nit==0, clc; end
disp([nit,x0+dx,nlf(x0+dx)])
```

## 6. Αναφορές


1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. https://sites.google.com/a/uni-konstanz.de/na09/Home/
3. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
4. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
5. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
6. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
7. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1

# ΚΕΦΑΛΑΙΟ 11:ΑΡΙΘΜΗΤΙΚΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

## Περιεχόμενα

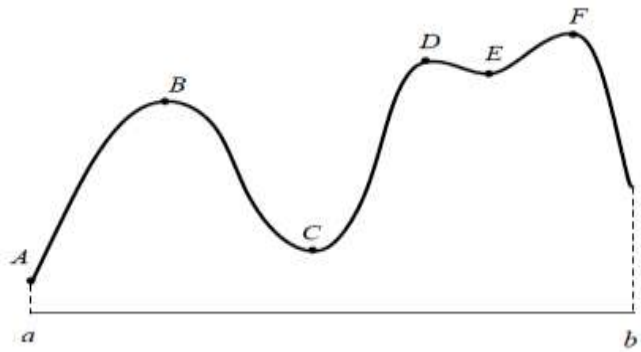


## 1. Περιγραφή προβλήματος

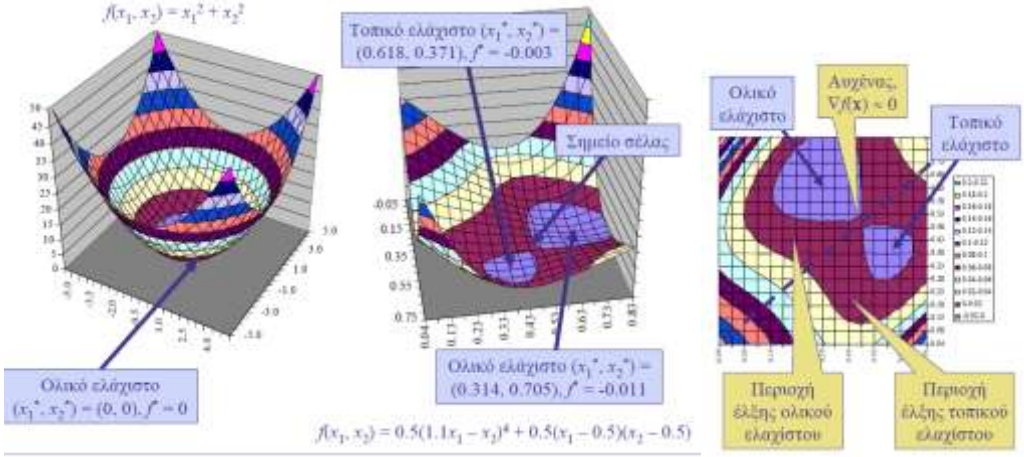
**Πρόβλημα**: Δοθέντος μιας συνάρτησης $f: S \subseteq R^n \to R$ να βρεθεί το ελάχιστό της $f(x_1, x_2, ..., x_n)$ δηλαδή $\min_{x \in S} f(x)$. Η λύση $x^*$ λέγεται το **ελάχιστο στάσιμο** σημείο της $f$ και $f(x^*)$ η ελάχιστη τιμή της $f$. Η συνάρτηση αναφέρεται ως **στοχική** (objective). Συνήθως το S είναι ένα κουτί! Δηλαδή, κάθε μεταβλητή μεταβάλλετε σε ένα πεπερασμένο διάστημα. Το $x^*$ λέγεται **τοπικό ελάχιστο** αν $f(x^*) \le f(x)$ για κάθε $x$ σε μια μικρή περιοχή του $x^*$. Αν η σχέση αυτή ισχύει σε όλο το S τότε λέγεται **ολικό ελάχιστο**.

**Σημειώστε** ότι θα ήταν σημαντικό αν κάθε τοπική λύση του προβλήματος ελαχιστοποίησης ήταν ολική. Αυτό ισχύει μόνον αν η $f$ είναι **κυρτή**. Η λύση του προβλήματος μεγιστοποίησης $\max_{x \in S} f(x)$ είναι η λύση του προβλήματος ελαχιστοποίησης $\min_{x \in S}(-f(x))$. Τα σημεία στα οποία η στοχική συνάρτηση $f$ ελαχιστοποιείται ή μεγιστοποιείται αναφέρονται σαν **ακρότατα σημεία**.

**Παραδείγματα ακρότατων σημείων**: Το παρακάτω διάγραμμα δείχνει τα "ακρότατα" σημεία μια συνάρτησης με μια μεταβλητή σε ένα διάστημα [a, b]. Η παράγωγος μηδενίζεται στα σημεία B, C, D, E, F. Τα σημεία B και D είναι τοπικά μέγιστα. Τα σημεία C και E είναι τοπικά ελάχιστα. Το ολικό μέγιστο εμφανίζεται στο F όπου η παράγωγος μηδενίζεται. Το ολικό ελάχιστο εμφανίζεται στο αριστερό τελικό σημείο A του διαστήματος, έτσι ώστε η παράγωγος δεν χρειάζεται να μηδενίζεται.

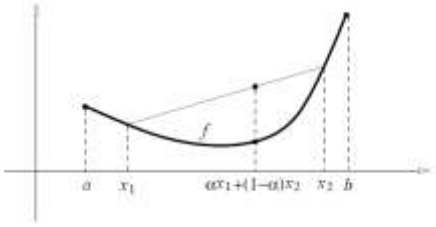



Τα παρακάτω γραφήματα δείχνουν ακρότατα σημεία συναρτήσεων με δύο μεταβλητές.




## 2. Ορισμοί μαθηματικών εννοιών

**Ορισμοί**: Μία συνάρτηση $f$ είναι κυρτή αν ισχύει $f(\alpha x_1 + (1-\alpha)x_2) \le \alpha f(x_1) + (1-\alpha) f(x_2)$ για κάθε σημείο του ευθύγραμμου τμήματος που περνάει από τα σημεία $x_1$ και $x_2$ και για κάθε α με τιμές $0 \le \alpha \le 1$. Δέστε το παρακάτω σχήμα




**Συμβολισμοί**:

- Διάνυσμα κλίσης (Gradient): Αν $f(x_1, x_2, \ldots, x_n)$ είναι μια πραγματική συνάρτηση $n$ μεταβλητών, τότε διάνυσμα
  $\left( \frac{\partial f}{\partial x_1}(c_1, c_2, \ldots, c_n), \frac{\partial f}{\partial x_2}(c_1, c_2, \ldots, c_n), \ldots, \frac{\partial f}{\partial x_{n1}}(c_1, c_2, \ldots, c_n) \right)$ είναι το διάνυσμα κλίσης ( gradient ) του $f$ στο σημείο $(c_1, c_2, \ldots, c_n)$ και συμβολίζεται με $\nabla f$ . Για $n = 3$, το διάνυσμα $\nabla f$ στο $(a, b, c)$ είναι κάθετο στη τομή της επιφάνειας $f(x, y, z) = f(a, b, c)$ στο σημείο $(a, b, c)$.

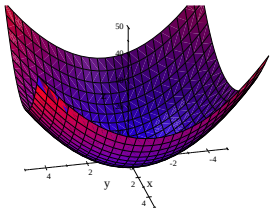
- Έσσιαν (Hessian) είναι ο $n \times n$ πίνακας

$$H = \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

όπου τα στοιχεία του είναι οι δεύτεροι μερικοί παράγωγοι της $f(x_1, x_2, \ldots, x_n)$.

**Παράδειγμα 1**: Να βρεθεί τα διανύσματα κλίσης, η Έσσιαν, και η γραφική παράσταση της συνάρτησης $f(x, y) = x^2 + y^2$. Το διάνυσμα κλίσης (Gradient) είναι $(2x \; 2y \; 0)^T$ και η Έσσιαν (Hessian) είναι

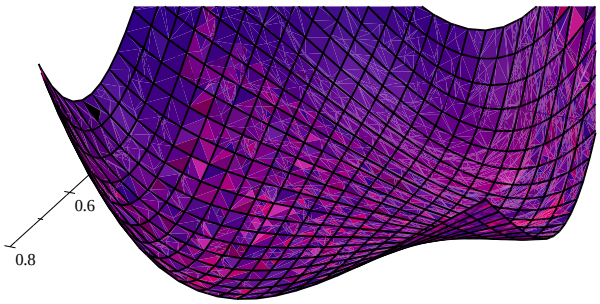
$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$



**Παράδειγμα 2**: Να βρεθεί τα διανύσματα κλίσης, η Έσσιαν, και η γραφική παράσταση της συνάρτησης $f(x, y) = 0.5(x-y)^4 + 0.5(x-0.5)(y-0.5)$ ,

Gradient είναι $\begin{pmatrix} 0.5y + 2.0(x-y)^3 - 0.25 \\ 0.5x - 2.0(x-y)^3 - 0.25 \end{pmatrix}$ και Hessian είναι

$\begin{pmatrix} 6.0(x-y)^2 & 0.5 - 6.0(x-y)^2 \\ 0.5 - 6.0(x-y)^2 & 6.0(x-y)^2 \end{pmatrix}$ και το γράφημα είναι



## 3. Θεωρητική θεμελίωση των αριθμητικών μεθόδων

**Θεωρία**: Περίπτωση ελαχιστοποίησης συναρτήσεων με μία μεταβλητή $f(x)$

- Συνθήκη βελτιστοποίησης 1: $f'(x^*) = \frac{df(x^*)}{dx} = 0$
- Συνθήκη βελτιστοποίησης 2: $f''(x^*) = \frac{d^2 f(x^*)}{dx^2} \geq 0$ (θετική καμπυλότητα όπως δείχνει το παρακάτω σχήμα)

**Θεωρία**: Περίπτωση ελαχιστοποίησης συναρτήσεων με πολλές μεταβλητές $f(x_1, x_2, ..., x_n)$. Από το Θεώρημα Taylor έχουμε $f(x^* + hp) = f(x^*) + hp^T g(x^*) + \frac{1}{2}h^2 hp^T H(x^*)p + O(h^3)$. Αν υποθέσουμε ότι $g(x^*) = \nabla f(x^*) \neq 0$ τότε μπορούμε να βρούμε ένα $p$ έτσι ώστε $hp^T g(x^*) < 0$ ( $p = -g(x^*) / \left\| g(x^*) \right\|$ ) οπότε για πολύ μικρό $h$ έχουμε $f(x^* + hp) < f(x^*)$.

Άρα μια ικανή συνθήκη για να έχει η συνάρτη ελάχιστο είναι $g(x^*) = \nabla f(x^*) = 0$. Για να διακρίνουμε μεταξύ μεγίστου ή ελαχίστου, πρέπει να εξετάσουμε τις τιμές της δευτέρας παραγώγου της *f*. Στην περίπτωση συναρτήσεων με πολλές μεταβλητές για να έχουμε ελάχιστο πρέπει να εξετάσουμε την θετικότητα της σχέσης $p^T H(x) p$ στο σημείο $x^*$.

**Παράδειγμα**: $f(x,y) = \frac{x-y}{x^2+y^2+1}$ , Gradient είναι $\begin{pmatrix} \frac{1}{x^2+y^2+1} - 2x\frac{x-y}{(x^2+y^2+1)^2} \\ -\frac{1}{x^2+y^2+1} - 2y\frac{x-y}{(x^2+y^2+1)^2} \end{pmatrix}$ , Hessian είναι

$$\begin{bmatrix} 8x^2\frac{x-y}{(x^2+y^2+1)^3} - 4\frac{x}{(x^2+y^2+1)^2} - 2\frac{x-y}{(x^2+y^2+1)^2} & 2\frac{x}{(x^2+y^2+1)^2} - 2\frac{y}{(x^2+y^2+1)^2} + 8xy\frac{x-y}{(x^2+y^2+1)^3} \\ 2\frac{x}{(x^2+y^2+1)^2} - 2\frac{y}{(x^2+y^2+1)^2} + 8xy\frac{x-y}{(x^2+y^2+1)^3} & 4\frac{y}{(x^2+y^2+1)^2} - 2\frac{x-y}{(x^2+y^2+1)^2} + 8y^2\frac{x-y}{(x^2+y^2+1)^3} \end{bmatrix}$$

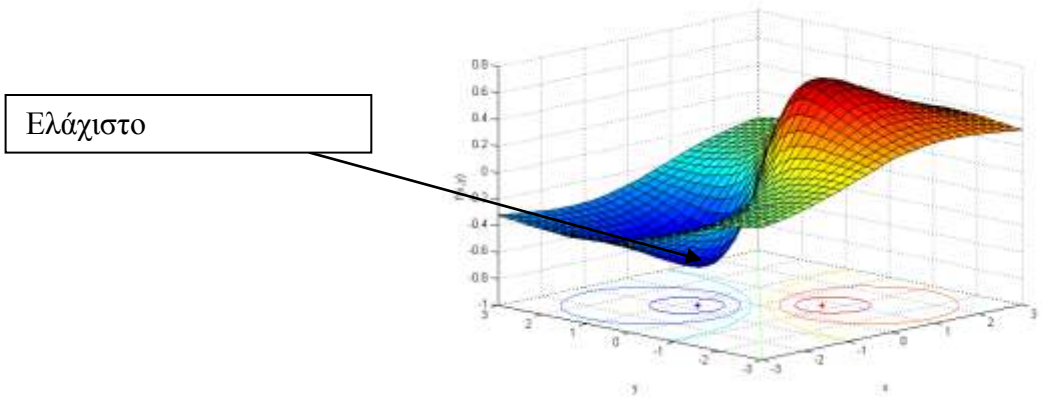
$\begin{bmatrix} \frac{1}{x^2+y^2+1} - 2x\frac{x-y}{(x^2+y^2+1)^2} = 0 \\ -\frac{1}{x^2+y^2+1} - 2y\frac{x-y}{(x^2+y^2+1)^2} = 0 \end{bmatrix}$ , η λύση είναι:

$\left[x = -\frac{1}{2}\sqrt{2}, y = \frac{1}{2}\sqrt{2}\right], \left[x = \frac{1}{2}\sqrt{2}, y = -\frac{1}{2}\sqrt{2}\right]$

$f(-\frac{1}{2}\sqrt{2}, \frac{1}{2}\sqrt{2}) = -\frac{1}{2}\sqrt{2}$

$f(\frac{1}{2}\sqrt{2}, -\frac{1}{2}\sqrt{2}) = \frac{1}{2}\sqrt{2}$

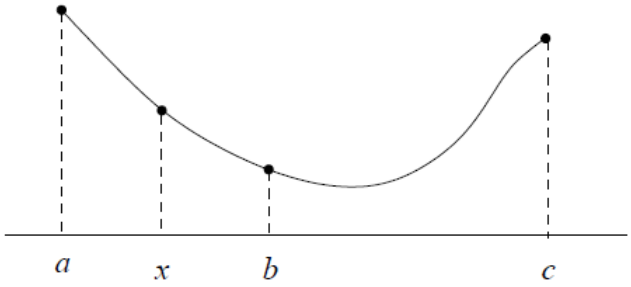
Μπορούμε να επαληθέψουμε τα παραπάνω αποτελέσματα κάνοντας γραφική παράσταση την συνάρτηση, και τα διανύσματα κλήσης.



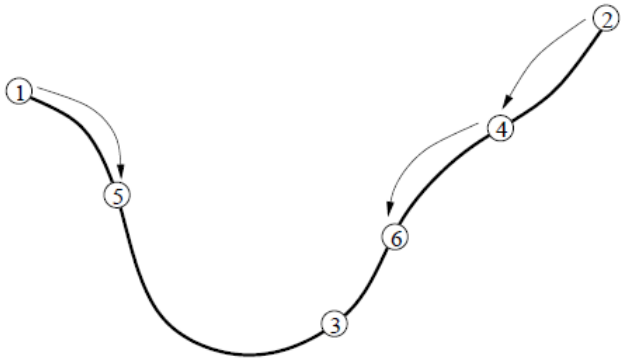

## 4. Μέθοδοι άμεσης αναζήτησης (direct search methods): Χρυσής Τομής (Golden Section Search)

Η μέθοδος είναι ανάλογος της μεθόδου διχοτόμησης για προσδιορισμό ριζών συναρτήσεων. Η βασική ιδέα είναι να βρεθεί κάποιο διάστημα "εγκλεισμού" των ακρότατων σημείων (ελάχιστο, μέγιστο) της συνάρτησης $f$ και στην συνέχεια να σμικρύνουμε αυτό το διάστημα. Η δυσκολία είναι ότι δεν γνωρίζουμε το ακρότατο σημείο οπότε δεν μπορούμε να είμαστε σίγουρη ότι το σημείο βρίσκετε στο διάστημα. Αν μας ενδιαφέρει μόνον ένα τοπικό ελάχιστο, τότε αρκεί να βρούμε ένα διάστημα που περιλαμβάνει μια ρίζα της παραγώγου της $f$ ($f'$) μέσω μιας προσέγγισης της παραγώγου. Ο εσωτερικός επαναληπτικός βρόχος (loop) δουλεύει ως εξής:

Αρχίζουμε με τρία σημεία a, b, και c έτσι ώστε $a < b < c$ και $f(b) < \min\{f(a), f(c)\}$,




στην συνέχεια επιλέγουμε ένα σημείο x, για παράδειγμα, το μέσο του διαστήματος [a, b]. Αν *f(x) > f(b)*, όπως δείχνει το παραπάνω σχήμα, τότε τα τρία νέα σημεία είναι [x, b, c]. Για να βεβαιωθούμε ότι το διάστημα [a, c] μικραίνει θα πρέπει να εναλλάξουμε τα τρία σημεία. Για παράδειγμα, αν έχουμε ήδη διχοτομήσει τα σημεία [a, b] τότε διχοτομούμε [b, c] στο επόμενο βήμα. Το σχήμα δείχνει πώς δουλεύει η μέθοδος.




Αρχικά, το ελάχιστο φράσσεται από τα σημεία 1, 3, 2. Η συνάρτηση υπολογίζεται στο 4 και αντικαθιστά το 2; μετά υπολογίζουμε *f* στο 5, και αντικαθιστούμε το 1; μετά υπολογίζουμε *f* στο 6, και αντικαθιστούμε το 4. Σημειώστε ότι η συνάρτηση στο κεντρικό σημείο είναι πάντοτε μικρότερη από τις τιμές της στα εξωτερικά σημεία. Το ελάχιστο σημείο φράσσεται από τα σημεία 5, 3, 6 μετά από τρία βήματα. Παρακάτω περιγράφεται ο αλγόριθμος της χρησής τομής σε ψευδοκώδικα και στην γλώσσα Python.

**Αλγόριθμος χρυσής τομής**
**input: a, b**

```
Initialize:
x1 = a + (b-a)*0.382
x2 = a + (b-a)*0.618
f1 = f(x1)
f2 = f(x2)
Loop:
if f1 > f2 then
  a = x1; x1 = x2; f1 = f2
  x2 = a + (b-a)*0.618
  f2 = f(x2)
else
  b = x2; x2 = x1; f2 = f1
  x1 = a + (b-a)*0.382
  f1 = f(x1)
endif
```

```
## module goldSearch in python
''' a, b = bracket(f,xStart,h)
Finds the brackets (a, b) of a minimum point of the user-
supplied scalar function f(x).
The search starts downhill from xStart with a step length
h.
x,fMin = search(f,a,b,tol=1.0e-6) Golden section method for
determining x that minimizes
the user-supplied scalar function f(x). The minimum must be
```

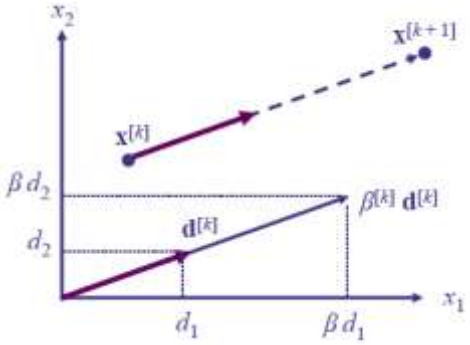
```
bracketed in (a,b).
'''
from math import log
def bracket(f,x1,h):
c = 1.618033989
f1 = f(x1)
x2 = x1 + h; f2 = f(x2)
# Determine downhill direction and change sign of h if
needed
if f2 > f1:
h = -h
x2 = x1 + h; f2 = f(x2)
# Check if minimum between x1 - h and x1 + h
if f2 > f1: return x2,x1 - h
# Search loop
for i in range (100):
h = c*h
x3 = x2 + h; f3 = f(x3)
if f3 > f2: return x1,x3
x1 = x2; x2 = x3
f1 = f2; f2 = f3
print Bracket did not find a mimimum
def search(f,a,b,tol=1.0e-9):
nIter = -2.078087*log(tol/abs(b-a))
R = 0.618033989
C = 1.0 - R
# First telescoping
x1 = R*a + C*b; x2 = C*a + R*b
f1 = f(x1); f2 = f(x2)
# Main loop
for i in range(nIter):
if f1 > f2:
a = x1
x1 = x2; f1 = f2
x2 = C*a + R*b; f2 = f(x2)
else:
b = x2
x2 = x1; f2 = f1
x1 = R*a + C*b; f1 = f(x1)
if f1 < f2: return x1,f1
else: return x2,f2
```

## 5. Μέθοδοι κλίσης (gradient methods) αναζήτησης τοπικών ακροτάτων

Πρόκειται για επαναληπτικές αριθμητικές μεθόδους που, ξεκινώντας από μια αρχική τιμή $x_{[0]}$, βελτιώνουν σταδιακά την τιμή της στοχικής (objective) συνάρτησης *f*, μεταβαίνοντας στο

επόμενο σημείο με εφαρμογή του γενικού κανόνα:
$\mathbf{x}_{[k+1]} = \mathbf{x}_{[k]} + \boldsymbol{\beta}_{[k]}\, \mathbf{d}_{[k]}$ όπου β είναι μια βαθμοτή παράμετρος κλίμακας και **d** μια διεύθυνση στο $R^n$, τέτοιες ώστε:

$$\mathbf{f(x_{[k+1]}) < f(x_{[k]}),\ για\ κάθε\ επανάληψη\ k}$$




Η παραπάνω διαδικασία εγγυάται σύγκλιση στο τοπικό ελάχιστο που βρίσκεται εγγύτερα στο σημείο εκκίνησης $x_{[0]}$. Για **d** μπορεί να επιλέξουμε την διεύθυνση που ορίζει η κατεύθυνση κλίσης $\nabla f(x)$.

**Σημειώστε** ότι το διάνυσμα κατεύθυνσης κλίσης έχει τις εξής δύο ιδιότητες:

α) Είναι η διεύθυνση που η συνάρτηση μεγαλώνει ποιο γρήγορα. Η αντίθετη κατεύθυνση κλίσης δείχνει προς τα πού η συνάρτηση μικραίνει ποιο γρήγορα.
β) Το μέγεθος του διανύσματος κλίσης (η Ευκλείδεια νόρμα $\|g\| = \sqrt{\left(\frac{df}{dx_1}\right)^2 + \left(\frac{df}{dx_2}\right)^2 + \ldots + \left(\frac{df}{dx_n}\right)^2}$ ) υποδεικνύει την ταχύτητα με την οποία η συνάρτηση αλλάζει σε κάθε σημείο.

Επομένως, αν όλες οι μερικοί παράγωγοι είναι μηδέν τότε το μέγεθος του διανύσματος κλίσης είναι μηδέν, και η συνάρτηση δεν αλλάζει σε αυτό το σημείο (το σημείο αναφέρεται σαν **στάσιμο (stationary) σημείο**). Ο αλγόριθμος της απότομης κατάβασης μπορεί να περιγραφεί από το ψευδοκώδικα

*for* k=0,1,2,...until satisfied do
  $u = \nabla f(x^k)$
  if $u = 0$ then stop
    else minimize the function $g(t) = f(x^{(k)} - tu)$ w.r.s.t t
      let $t^* > 0$ be the closest such minimum to zero
      $x^{(k+1)} \leftarrow x^{(k)} - t^* u$

Οι επιμέρους τεχνικές διαφοροποιούνται ανάλογα με τον τρόπο ορισμού των β και d.

## 6. Κλασικές τεχνικές κλίσεων

Η απλούστερη τεχνική κλίσης είναι η μέθοδος της πλέον απότομης κατάβασης (steepest descent) για προσδιορισμό τοπικού ελαχίστου (απότομης ανάβασης (steepest ascent) για προσδιορισμό τοπικού μεγίστου), όπου η διεύθυνση $d_{[k]}$ είναι αντίθετη στην κλίση $\nabla f(x_{[k]})$ της συνάρτησης. Η διαδικασία αναζήτησης γράφεται:

$$\mathbf{x_{[k+1]} = x_{[k]} - \beta_{[k]} \nabla f(x_{[k]})}.$$

Για την εύρεση του τοπικού μεγίστου εφαρμόζετε η μέθοδος της απότομης ανάβασης (steepest ascent) όπου η διεύθυνση είναι η κλίση $\nabla f(x_{[k]})$ της συνάρτησης. Η μέθοδος αυτή είναι γνωστή και σαν **ανάβαση λόφου** ("Hill Climbing")

Κάθε νέο σημείο $x_{[k+1]}$ είναι η θέση ελαχίστου της f κατά μήκος της διεύθυνσης που ορίζει η κλίση της. Συνεπώς, το $\beta_{[k]}$ προσδιορίζεται με τρόπο ώστε να ελαχιστοποιείται η έκφραση

$$\mathbf{g(\beta_{[k]}) = f(x_{[k]} - \beta_{[k]} \nabla f(x_{[k]}))}$$

Με τον τρόπο αυτό, προκύπτει ένα πρόβλημα βελτιστοποίησης μιας μεταβλητής, που επιλύεται με τυπικές αριθμητικές μεθόδους (π.χ. χρυσή τομή).

Η πορεία σύγκλισης της μεθόδου είναι αργή (μικρά βήματα), ενώ η μετακίνηση είναι πάντα κάθετη στη διεύθυνση του προηγούμενου βήματος. Στη μέθοδο συζυγών κλίσεων (conjugate gradient) των Fletcher-Reeves (1964), η πορεία επιταχύνεται, αφού η νέα διεύθυνση προκύπτει ως γραμμικός συνδυασμός των κλίσεων στο τρέχον και το προηγούμενο σημείο, με βάση τη σχέση:

$$\mathbf{x_{[k+1]} = x_{[k]} - \beta_{[k]} [\nabla f(x_{[k]}) + \gamma_{[k]} \nabla f(x_{[k-1]})]}$$

όπου: $\gamma_{[k]} = \|\nabla f(x_{[k]})\|^2 / \|\nabla f(x_{[k-1]})\|^2$ και $\beta_{[k]}$ παράμετροι προσδιορίζονται με τρόπο ώστε να ελαχιστοποιείται η έκφραση $\mathbf{g(\beta_{[k]}) = f(x_{[k+1]})}$.

Παρακάτω περιγράφεται ο αλγόριθμος του fletcherReeves σε Python

```
## module fletcherReeves
' xMin,nIter = optimize(F,gradF,x,h=0.01,tol=1.0e-6
Fletcher-Reeves method of minimizing a function.
F(x) = user-supplied function to be minimized.
gradF(x) = user-supplied function for grad(F).
x = starting point.
h = initial search increment used in 'bracket'.
xMin = mimimum point.
nIter = number of iterations.
'
from numpy import array,zeros,dot
from goldSearch import *
from math import sqrt
def optimize(F,gradF,x,h=0.1,tol=1.0e-6):
def f(s): return F(x + s*v) # Line function along v
n = len(x)
g0 = -gradF(x)
v = g0.copy()
F0 = F(x)
for i in range(200):
a,b = bracket(f,0.0,h) # Minimization along
s,fMin = search(f,a,b) # a line
x = x + s*v
F1 = F(x)
g1 = -gradF(x)
if (sqrt(dot(g1,g1)) <= tol) or (abs(F0 - F1) < tol):
return x,i+1
gamma = dot((g1 - g0),g1)/dot(g0,g0)
v = g1 + gamma*v
g0 = g1.copy()
F0 = F1
print fletcherReeves did not converge
```

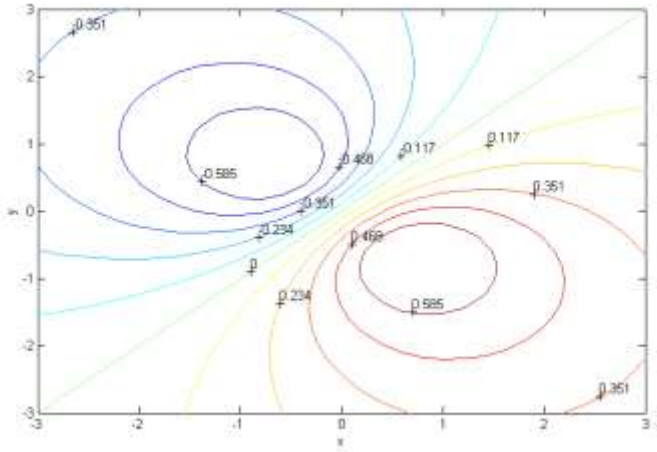
## 7. <u>Παράδειγμα υπολογισμού του μέγιστου μιας συνάρτησης με την μέθοδο απότομης ανάβασης</u>

*Ιδέα: Εφαρμογή απότομης ανάβασης (steepest ascent ή "hill climbing") στρατηγική.*

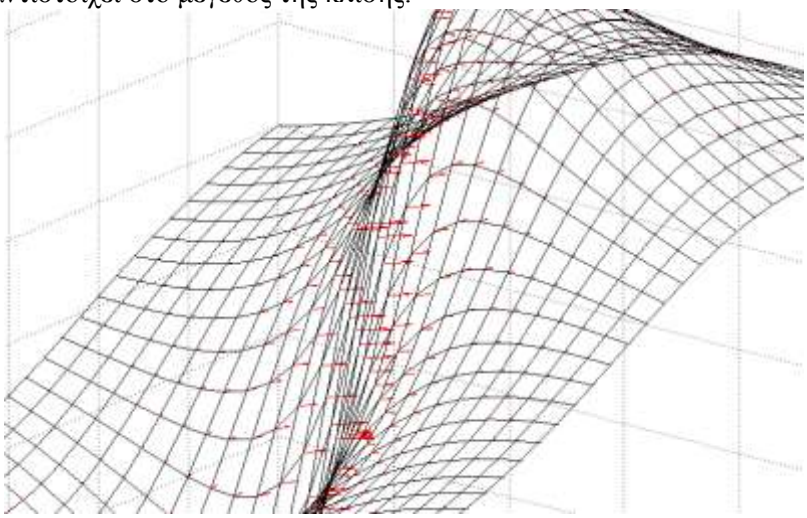
Αρχίζοντας με μια αρχική προσέγγιση του σημείου μεγιστοποίησης της συνάρτησης και βρίσκομαι την κατεύθυνση όπου η συνάρτηση μεγαλώνει τα μέγιστα. Στην συνέχεια κινούμαστε προς αυτή την κατεύθυνση με κάποιο (μικρό) βήμα. Η διαδικασία αυτή επαναλαμβάνεται έως ότου η συνάρτηση δεν αλλάζει σημαντικά.

**Υψομετρική καμπύλη (Contour plot) συνάρτησης** $f(x,y)=\frac{x-y}{x^2+y^2+1}$

(κάθε υψομετρική καμπύλη αντιστοιχεί σε σταθερή τιμή της συνάρτησης)

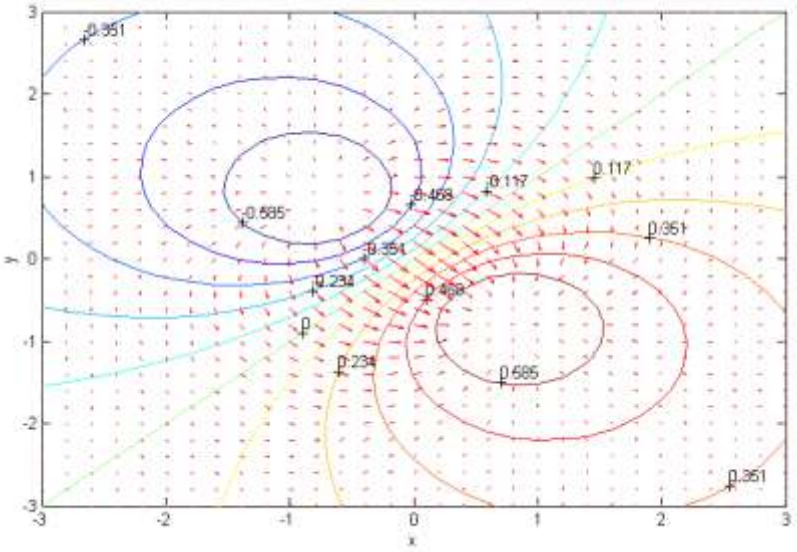



**Παρατήρηση**: Οι υψομετρικές καμπύλες είναι ποιο πυκνές εκεί που η συνάρτηση μεγαλώνει γρηγορότερα. Σε κάθε σημείο, μπορούμε να παρατηρήσουμε την κατεύθυνση που η συνάρτηση μεγαλώνει ταχύτερα. Στο παρακάτω γράφημα, παραστούμε τα διανύσματα κλίσης όπου το μέγεθος τους αντιστοιχεί στο μέγεθος της κλίσης.

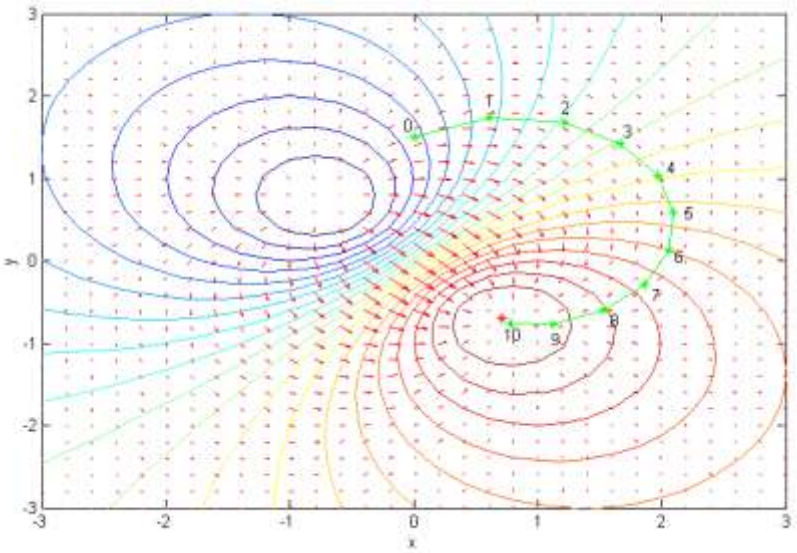


Τα ίδια στοιχεία μπορούν να απεικονισθούν στο παρακάτω υψομετρικό γράφημα. Παρατηρούμε ότι τα διανύσματα κλίσης είναι κάθετα στις υψομετρικές γραμμές και ότι η συνάρτηση

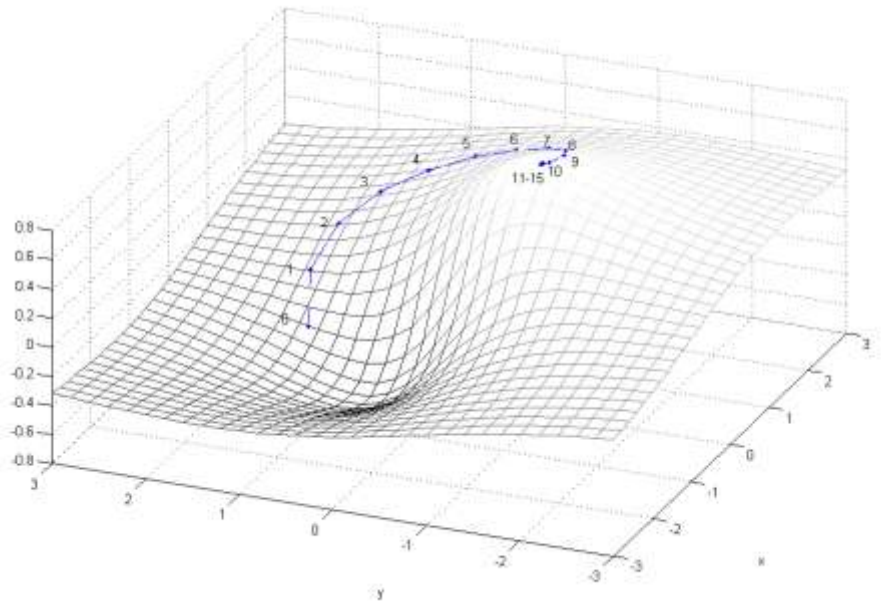
μεγαλώνει στο σημείο (x=0,y=0).




Αν ακολουθήσουμε το κανόνα ότι σε κάθε σημείο κινούμεθα στην κατεύθυνση της μέγιστης ανάβασης, θα ακολουθήσουμε την πράσινη "τροχιά" που απεικονίζεται στο παρακάτω υψομετρικό γράφημα.

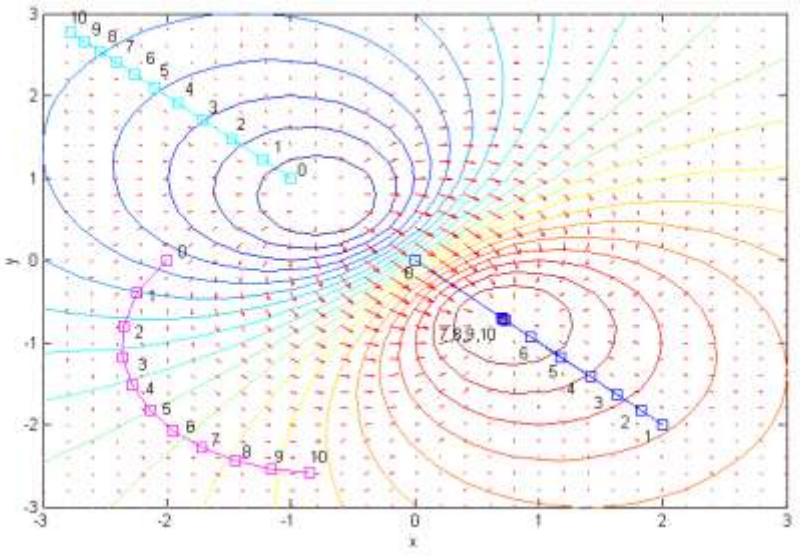



ή την τροχιά που δείχνει το 3D γράφημα :

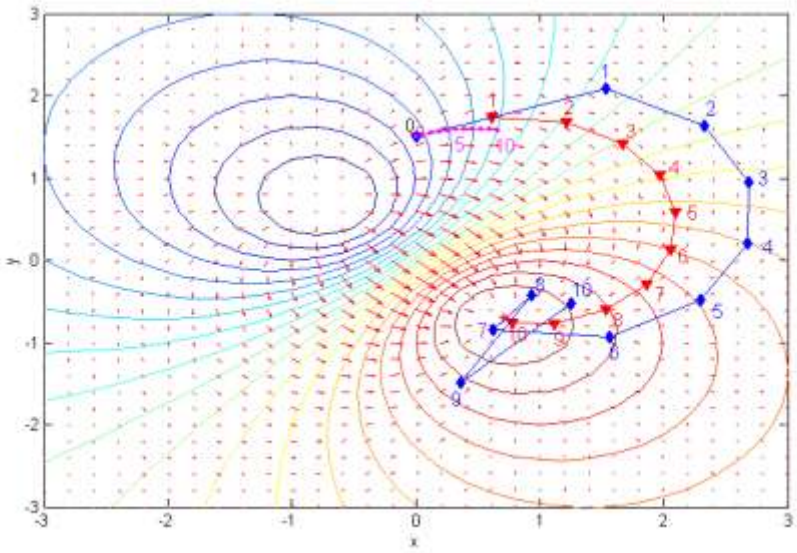


Τα βήματα του αλγορίθμου συμβολίζονται με ακεραίους αριθμούς πάνω στην "τροχιά" όπου με "0" συμβολίζεται το σημείο εκκίνησης που είναι η αρχική προσέγγιση του ακρότατου σημείου. Το διάγραμμα δείχνει ότι η μέθοδος συγκλίνει στο μέγιστο.

i) Παρακάτω δείχνουμε γραφικά ότι η σύγκλιση εξαρτάται από το σημείο εκκίνησης που το συμβολίζουμε **$(x_0,y_0)$.** Σημειώστε, ότι όταν επιλέξουμε το **$(x_0=0, y_0=0)$** ως σημείο εκκίνησης, τότε η μέθοδος συγκλίνει πολύ γρήγορα. Στην περίπτωση του σημείου **$(x_0=-2, y_0=0)$** τότε η σύγκλιση είναι αργή, και όταν **$(x_0=-1, y_0=1)$** η μέθοδος αποκλίνει δηλαδή δεν βρίσκει το μέγιστο.



ιι) Σημειώστε ότι η σύγκλιση της μεθόδου εξαρτάται από το μήκος του βήματος. Στο γράφημα παρατηρούμε ότι μικρό μήκος βήματος απαιτεί πολλά βήματα και μεγάλο μήκος βήματος μπορεί να οδήγηση σε μη σύγκλιση.



Οι παρατηρήσεις ι) και ιι) είναι τα μειονεκτήματα της μεθόδου.

**Παράδειγμα:** max( $f(x,y)=\frac{x-y}{x^2+y^2+1}$

Παράμετροι του αλγορίθμου: $\beta$ =2 $x_{(0)}$=0 $y_{(0)}$=1.5

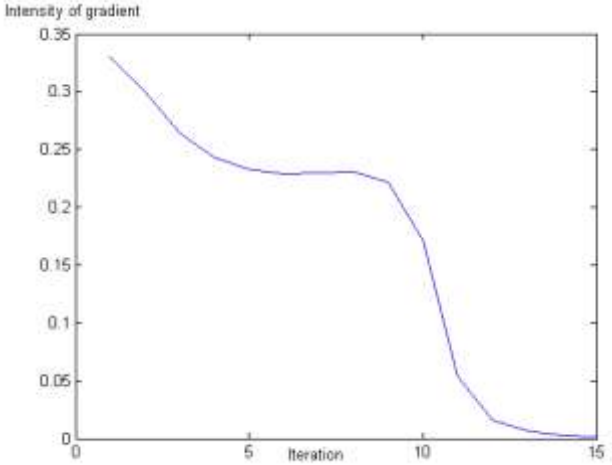
Κριτήρια σύγκλισης : $|f_{(i)}(x,y)-f_{(i-1)}(x,y)|<\varepsilon$ with $\varepsilon = 5*10^{-7}$

| Επανάληψη i | x | y | $\left(\frac{df}{dx}\right)$ | $\left(\frac{df}{dy}\right)$ | Μέγεθος του gradient $\|g\|$ | Τιμή της συνάρτησης στο x,y στη i επανάληψη $f_{(i)}(x,y)$ | Αλλαγή της συνάρτησης σε διαδοχικές επαν. $f_{(i)}(x,y)-f_{(i-1)}(x,y)$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1.5000 | 0.3077 | 0.1183 | 0.3297 | -0.461538461538462 | |
| 1 | 0.6154 | 1.7367 | 0.2990 | -0.0259 | 0.3001 | -0.255144131278668 | 0.20639 |
| 2 | 1.2134 | 1.6849 | 0.2288 | -0.1320 | 0.2642 | -0.0887796215061965 | 0.16636 |
| 3 | 1.6711 | 1.4210 | 0.1473 | -0.1931 | 0.2429 | 0.0430297030843086 | 0.13181 |
| 4 | 1.9657 | 1.0348 | 0.0646 | -0.2232 | 0.2324 | 0.156860600023865 | 0.11383 |
| 5 | 2.0949 | 0.5884 | -0.0175 | -0.2283 | 0.2290 | 0.262697531337831 | 0.10584 |
| 6 | 2.0598 | 0.1318 | -0.0969 | -0.2085 | 0.2299 | 0.366526165355723 | 0.10383 |
| 7 | 1.8659 | -0.2851 | -0.1664 | -0.1602 | 0.2310 | 0.471420408518152 | 0.10489 |
| 8 | 1.5331 | -0.6056 | -0.2056 | -0.0815 | 0.2212 | 0.57536025581446 | 0.10394 |
| 9 | 1.1220 | -0.7687 | -0.1715 | 0.0070 | 0.1717 | 0.663462359678314 | 0.088102 |

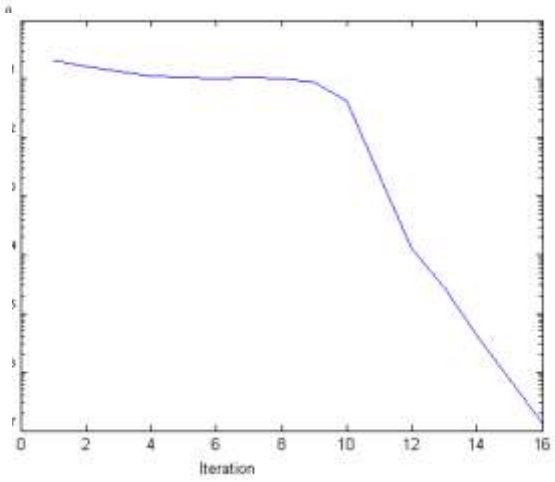
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 10 | 0.7789 | -0.7547 | -0.0449 | 0.0292 | 0.0536 | 0.704695822057818 | 0.041233 |
| 11 | 0.6890 | -0.6962 | 0.0132 | -0.0080 | 0.0154 | 0.706945983742316 | 0.0022502 |
| 12 | 0.7153 | -0.7122 | -0.0057 | 0.0035 | 0.0067 | 0.707074008658418 | 0.00012802 |
| 13 | 0.7038 | -0.7051 | 0.0023 | -0.0014 | 0.0027 | 0.707101616702102 | 2.7608e-005 |
| 14 | 0.7085 | -0.7079 | -0.0010 | 0.0006 | 0.0011 | 0.707105865892936 | 4.2492e-006 |
| 15 | 0.7065 | -0.7068 | 0.00039 | -0.0002 | 0.0004 | 0.707106626310556 | 7.6042e-007 |
| 16 | 0.7073 | -0.7072 | | | | 0.707106754462975 | 1.2815e-007 |

Το προσεγγιστικό ακρότατο σημείο είναι: (0.7073,-0.7072) και η τιμής της στοχικής συνάρτησης f(x,y)= 0.707106754462975

**Σημείωση**: Το μέγεθος του διανύσματος κλίσης μπορεί να χρησιμοποιηθεί σαν ένα άλλο κριτήριο σύγκλισης.




Αλλαγή του μεγέθους του διανύσματος κλίσης




Αλλαγή της στοχικής συνάρτησης

## 8. Μέθοδος Newton

**Ιδέα:** Για την εύρεση των ακρότατων σημείων της $f$ την προσεγγίζουμε με μια απλούστερη συνάρτηση (π.χ. ένα πολυώνυμο) και βρίσκουμε τα ακρότατα σημεία της προσέγγισης.

**Υλοποίηση της ιδέας**

Από το θεώρημα του Taylor και για συναρτήσεις δύο μεταβλητών έχουμε

$$f\left(x+\Delta x, y+\Delta y\right)= f\left(x,y\right)+\frac{\Delta x}{1}\frac{d}{dx}f\left(x,y\right)+\frac{\Delta y}{1}\frac{d}{dy}f\left(x,y\right)+$$

$$\frac{\Delta x^2}{2}\frac{d^2}{d^2x}f\left(x,y\right)+\frac{\Delta x\Delta y}{2}\frac{d^2}{dxdy}f\left(x,y\right)+\frac{\Delta y\Delta x}{2}\frac{d^2}{dydx}f\left(x,y\right)+\frac{\Delta y^2}{2}\frac{d^2}{d^2y}f\left(x,y\right)+...$$

Αν έχουμε $n$ μεταβλητές $x_1,\ldots,x_n$ μπορούμε να θεωρήσουμε την παρακάτω προσέγγιση:

$$f\left(x_1+\Delta x_1,\ldots,x_n+\Delta x_n\right)= f\left(x_1,\ldots,x_n\right)+\sum_{i=1}^{n}\Delta x_i\frac{d}{dx_i}f\left(x_1,\ldots,x_n\right)+\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta x_i\Delta x_j\frac{d^2}{dx_i dx_j}f\left(x_1,\ldots,x_n\right)$$

Σημειώστε ότι και οι δύο πλευρές της εξίσωσης εξαρτώνται από τα $\Delta x_i$ - δηλαδή είναι συναρτήσεις του $\Delta x_i$ .

Επομένως, η νέα συνάρτηση *h*:

$$h(\Delta x_1,\ldots,\Delta x_n) = f(x_1,\ldots,x_n) + \sum_{i=1}^{n} \Delta x_i \frac{d}{dx_i} f(x_1,\ldots,x_n) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \Delta x_i \Delta x_j \frac{d^2}{dx_i dx_j} f(x_1,\ldots,x_n)$$

παριστά την προσέγγιση της τιμή $f$ στην περιοχή του σημείου $x_1,\ldots,x_n$ και είναι παραγωγίσημη ως προς $\Delta x_i$ .

Η εύρεση της μεγίστης τιμής της $f$ ισοδυναμεί με τον προσδιορισμό των $\Delta x_1,\ldots,\Delta x_n$ έτσι ώστε η τιμή της είναι η μέγιστη δυνατή. Με βάση την θεωρία αυτό συμβαίνει στα σημεία που οι μερικές παράγωγοι μηδενίζονται. Επομένως, αρκεί να προσδιορίσουμε τις τιμές των μεταβλητών $\Delta x_1,\ldots,\Delta x_n$ έτσι ώστε:

$$\frac{d}{d\Delta x_k} h(\Delta x_1,\ldots,\Delta x_n) = 0, k = 1,\ldots,n$$

Ο υπολογισμός των μερικών παραγώγων μετά από κατάλληλες απλοποιήσεις οδηγεί στις σχέσεις:

$$\frac{d}{d\Delta x_k} h(\Delta x_1,\ldots,\Delta x_n) = \frac{d}{d\Delta x_k}\left(\Delta x_k \frac{d}{dx_k}\cdot f(x_1,\ldots,x_n)\right) + \frac{d}{d\Delta x_k}(\Delta x_k)\cdot\frac{1}{2}\sum_{\substack{j=1,n\\ j\neq k}} \Delta x_j \frac{d^2}{dx_k dx_j} f(x_1,\ldots,x_n)$$

$$+\frac{d}{d\Delta x_k}(\Delta x_k)\cdot\frac{1}{2}\sum_{\substack{i=1,n\\ i\neq k}}^{n} \Delta x_i \frac{d^2}{dx_i dx_k} f(x_1,\ldots,x_n) + \frac{1}{2}\cdot\frac{d}{d\Delta x_k}\left(\Delta x_k{}^2\right)\frac{d^2}{dx_k^2} f(x_1,\ldots,x_n)$$

και τελικά στις σχέσεις

$$\frac{d}{d\Delta x_k} h(\Delta x_1,\ldots,\Delta x_n) = \frac{d}{dx_k}\cdot f(x_1,\ldots,x_n) + \frac{1}{2}\sum_{\substack{j=1,n\\ j\neq k}}^{n} \Delta x_j \frac{d^2}{dx_k dx_j} f(x_1,\ldots,x_n)$$

$$+\frac{1}{2}\sum_{\substack{i=1,n\\ i\neq k}}^{n} \Delta x_i \frac{d^2}{dx_i dx_k} f(x_1,\ldots,x_n) + \frac{1}{2} 2\Delta x_k \frac{d^2}{dx_k^2} f(x_1,\ldots,x_n) = 0 \quad .$$

Σε διανυσματική μορφή η σχέση αυτή γράφεται ως:

$g(\mathbf{x}) + H\Delta\mathbf{x} = 0$

όπου g(**x**) είναι το διάνυσμα κλίσης και *H* ο Έσιαν πίνακας από την οποία λαμβάνουμε:

$\Delta\mathbf{x} = -inv(H)\cdot g(\mathbf{x})$

Επομένως, η μέθοδος Newton μπορεί να υλοποιηθεί από τον παρακάτω αλγόριθμο.

1.Επιλέγουμε σαν αρχική προσέγγιση του ακρότατου σημείου $\mathbf{x} = [x_1^{(0)}, x_2^{(0)},\ldots x_n^{(0)}]$

2. Σε αυτό το σημείο υπολογίζουμε την τιμή του διανύσματος κλίσης $g(\mathbf{x})$ και την τιμή του πίνακα *H*

3.Ενημερώνουμε την τιμή του **x** ως: $\mathbf{x} = \mathbf{x} + \Delta\mathbf{x}$ όπου $\Delta\mathbf{x} = -inv(H)\cdot g(\mathbf{x})$

4. Επαναλαμβάνουμε τα βήματα 2,3 εφόσον υπάρχει σημαντική αλλαγή στις τιμές της f(**x**) σε δύο διαδοχικές επαναλήψεις .

Παράδειγμα: **Εφαρμόζουμε τον αλγόριθμο στο πρόβλημα μεγιστοποίησης** $\max f(x,y)=\frac{x-y}{x^2+y^2+1}$

Για αυτή την περίπτωση έχουμε $g(x,y)=\left[\frac{-x^2+y^2+2xy+1}{\left(x^2+y^2+1\right)^2},\frac{-x^2+y^2-2xy-1}{\left(x^2+y^2+1\right)^2}\right]^T$

και H:

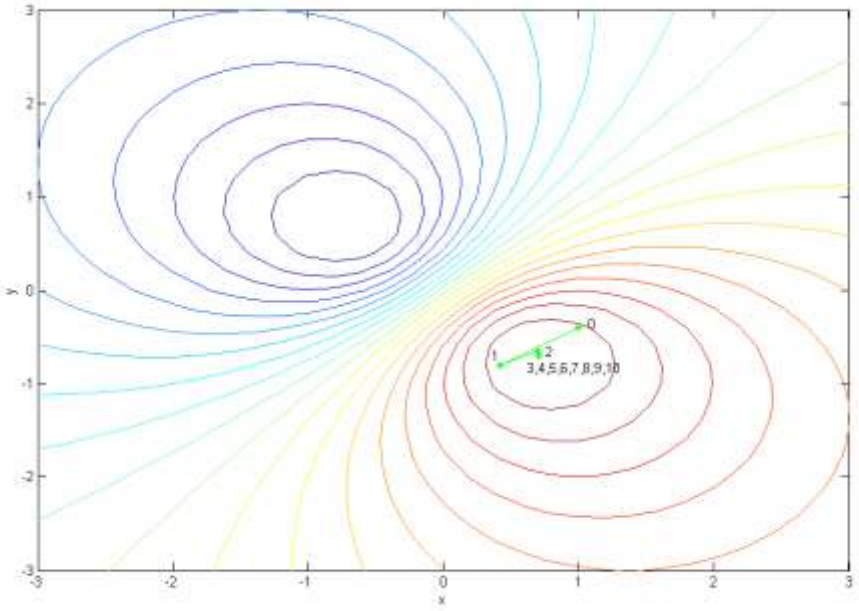
$$H=\begin{bmatrix}\frac{d}{dx}\left(\frac{-x^2+y^2+2xy+1}{\left(x^2+y^2+1\right)^2}\right) & \frac{d}{dy}\left(\frac{-x^2+y^2+2xy+1}{\left(x^2+y^2+1\right)^2}\right)\\ \frac{d}{dx}\left(\frac{-x^2+y^2-2xy-1}{\left(x^2+y^2+1\right)^2}\right) & \frac{d}{dy}\left(\frac{-x^2+y^2-2xy-1}{\left(x^2+y^2+1\right)^2}\right)\end{bmatrix}$$

$$H=\begin{bmatrix}\frac{-6x+2y}{\left(x^2+y^2+1\right)^2}+\frac{8(x-y)x^2}{\left(x^2+y^2+1\right)^3} & \frac{-2y+2x}{\left(x^2+y^2+1\right)^2}+\frac{8(x-y)xy}{\left(x^2+y^2+1\right)^3}\\ \frac{-2y+2x}{\left(x^2+y^2+1\right)^2}+\frac{8(x-y)xy}{\left(x^2+y^2+1\right)^3} & \frac{-2x+6y}{\left(x^2+y^2+1\right)^2}+\frac{8(x-y)y^2}{\left(x^2+y^2+1\right)^3}\end{bmatrix}$$

Επιλέγοντας σαν αρχική προσέγγιση: $x^{(0)}=1$, $y^{(0)}=-0.4$;
και σαν κριτήριο σύγκλισης την σχέση $|f^{(i)}(x,y)-f^{(i-1)}(x,y)|<\varepsilon$ with $\varepsilon = 5*10^{-7}$ (όπως κάναμε και στην περίπτωση του αλγορίθμου Gradient) λαμβάνουμε:

| Επανάληψη *i* | x | y | $\left(\frac{df}{dx}\right)$ | $\left(\frac{df}{dy}\right)$ | Μέγεθος διανύσματος κλίσης $\|g\|$ | Τιμή της συνάρτησης x,y στην i-th επανάληψη $f^{(i)}(x,y)$ | Μεταβολή της συνάρτησης σε δύο διαδοχικές επαναλήψεις $f^{(i)}(x,y)-f^{(i-1)}(x,y)$ |
|---|---|---|---|---|---|---|---|
| 0 | 1 | -0.4 | -0.13717 | -0.22291 | 0.26173 | 0.64815 | |
| 1 | 0.41968 | -0.80928 | 0.23846 | 0.047147 | 0.24308 | 0.67117 | 0.023025 |
| 2 | 0.70094 | -0.65483 | 0.005285 | -0.039203 | 0.039557 | 0.70609 | 0.034913 |
| 3 | 0.70591 | -0.70431 | 0.00085219 | -0.0019835 | 0.0021588 | 0.7071 | 0.0010173 |
| 4 | 0.7071 | -0.7071 | 4.6837e-06 | -7.8425e-06 | 9.1347e-006 | 0.70711 | 3.2769e-006 |
| 5 | 0.70711 | -0.70711 | | | | 0.70711 | 5.9001e-011 |

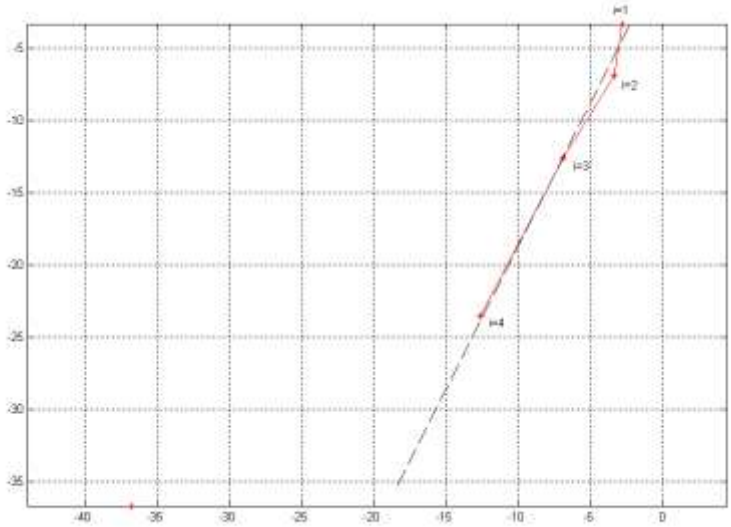
και




Μπορούμε να δούμε ότι ο αλγόριθμος συγκλίνει γρηγορότερα από τον κλασσικό Gradient αλγόριθμο

Η γραφική παράσταση των $\log\left|f^{(i-1)}-f_{opt}\right|, \log\left|f^{(i)}-f_{opt}\right|$ μας δίνει την σχέση $\log\left|f^{(i)}-f_{opt}\right| \approx c+2\log\left|f^{(i-1)}-f_{opt}\right|$. Επομένως, η μέθοδος Newton έχει βαθμό σύγκλισης 2.

$\log\left|f^{(i)}-f_{opt}\right|$

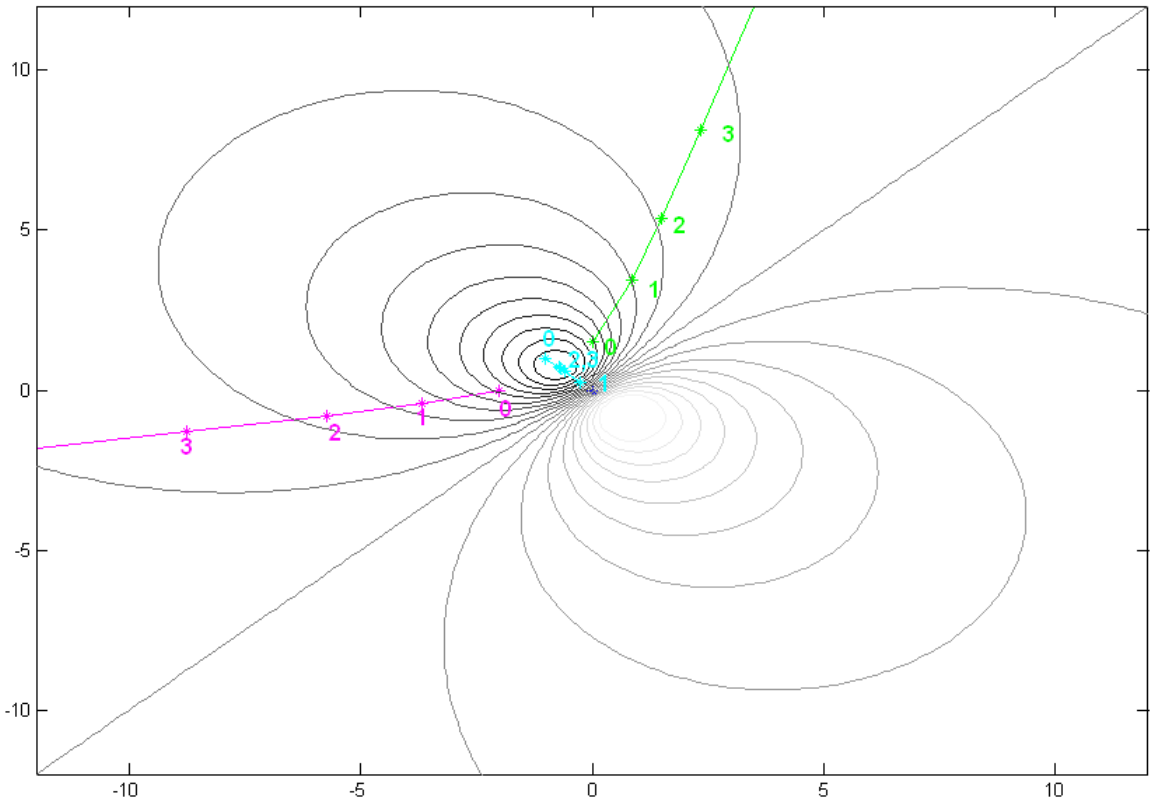



$\log\left|f^{(i-1)}-f_{opt}\right|$

***Προβλήματα με τον αλγόριθμο του Newton***

1. Συχνά δεν συγκλίνει

Για πολλές περιπτώσεις που η μέθοδος gradient συγκλίνει, η μέθοδος Newton δεν συγκλίνει. Για $x^{(0)}=0, y^{(0)}=0,$ ο πίνακας H είναι ιδιάζων (singular) (det(H)=0) έτσι δεν μπορεί να εφαρμοσθεί η μέθοδος Newton. Για $x^{(0)}=-1, y^{(0)}=-1$ Newton's μέθοδο βρίσκει το *ελάχιστο* όχι το μέγιστο (επομένως η μέθοδος δεν μας εγγυάται ότι θα βρούμε το σωστό ακρότατο σημείο!). Τελικά, για τα αρχικά σημεία , $x^{(0)}=0, y^{(0)}=1.5$ and $x^{(0)}=-2, y^{(0)}=0$ η μέθοδος Newton αποκλίνει (αντίθετα από την Gradient method που συγκλίνει) .




2. Σε σύγκριση με την μέθοδο Gradient , η Newton απαιτεί περισσότερους υπολογισμούς ιδιαίτερα για συναρτήσεις με μεγάλο αριθμό μεταβλητών. Συγκεκριμένα απαιτεί τον υπολογισμό του πίνακα H και τον αντίστροφό του.

Για να αποφύγουμε τα προβλήματα της Newton και να βελτιώσουμε την Gradient, οι παρακάτω τεχνικές έχουν προταθεί:

**Conjugate Gradient Algorithm (CGA)**

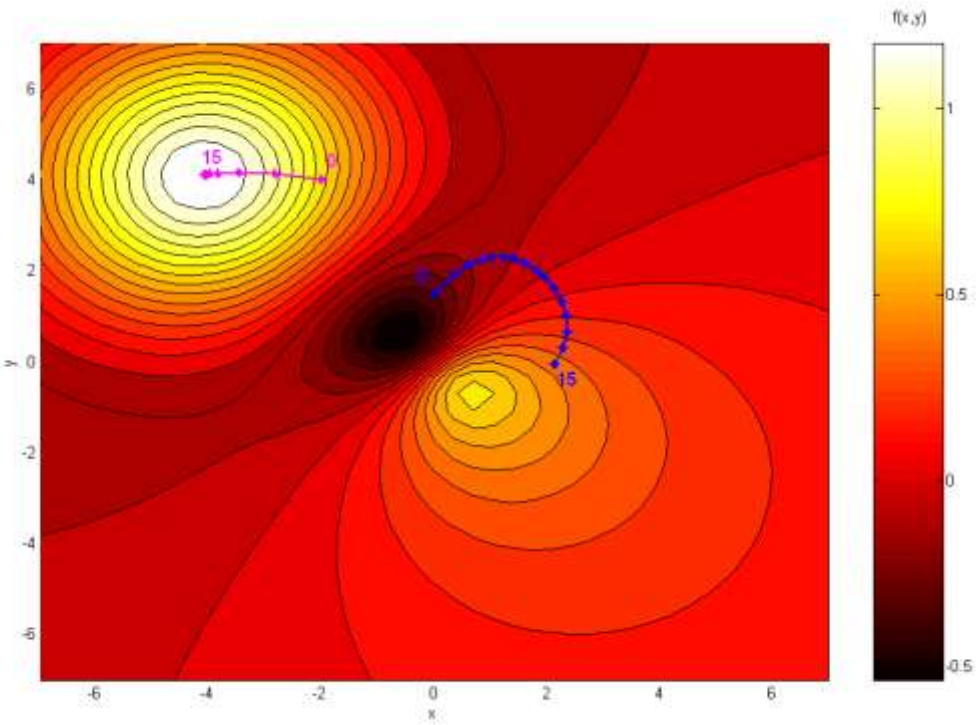
- Στις διαδοχικές επαναλήψεις , οι κατευθύνσεις είναι ορθογώνιες μεταξύ τους
  Quasi-Newton Method
- Αντί του υπολογισμού του πίνακα *H* τον προσεγγίζουμε

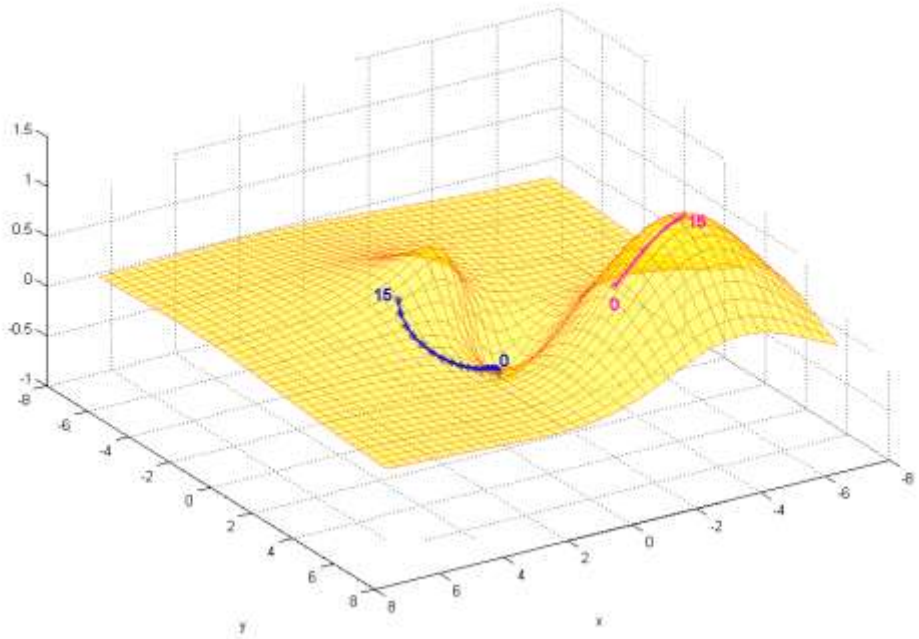
Όλες οι μέθοδοι που περιγράψαμε βρίσκουν ένα **τοπικό ακρότατο σημείο**.

Για παράδειγμα, παρακάτω είναι τα αποτελέσματα εφαρμογής της Gradient μεθόδου στο παρακάτω πρόβλημα

$$\max f(x,y)=\frac{x-y}{x^2+y^2+1}+1.5e^{-\frac{(x+4)^2+(y-4)^2}{10}}$$

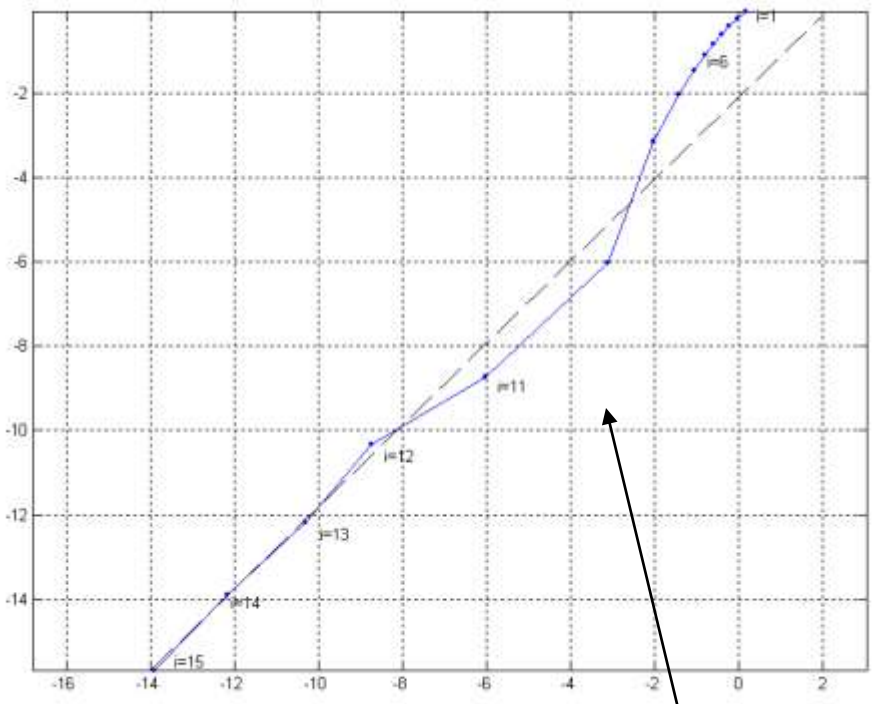
όπου η σύγκλιση στο τοπικό ελάχιστο x=+sqrt(0.5), y=-sqrt(0.5) και όχι στο ολικό μέγιστο x=-4. y=4 εξαρτάται από το αρχικό σημείο που επιλέγει.




Επομένως, το σημαντικό πρόβλημα είναι πως εκτελούμαι **ολική** βελτιστοποίηση ( *global* optimization)!

$\log\left|f^{(i)} - f_{opt}\right|$




$\log\left|f^{(i-1)} - f_{opt}\right|$

Κλίση 1, βαθμός σύγκλισης 1

## 9. Βιβλιοθήκη βελτιστοποίησης Python: scipy.optimize

Το **scipy.optimize** πακέτο υλοποιεί τους ποιο γνωστούς αλγορίθμους βελτιστοποίησης. Μια λεπτομερής λίστα αλγορίθμων βρίσκονται: στο **scipy.optimize** (μπορούν να βρεθούν με την εντολή help(scipy.optimize)).
Η βιβλιοθήκη περιλαμβάνει:

1. Unconstrained and constrained minimization of multivariate scalar functions (**minimize**) using a variety of algorithms (e.g. BFGS, Nelder-Mead simplex, Newton Conjugate Gradient, COBYLA or SLSQP)
2. Global (brute-force) optimization routines (e.g., **anneal**)
3. Least-squares minimization (**leastsq**) and curve fitting (**curve_fit**) algorithms
4. Scalar univariate functions minimizers (**minimize_scalar**) and root finders (**newton**)
5. Multivariate equation system solvers (**fsolve**)
6. Large-scale multivariate equation system solvers (e.g. **newton_krylov**)

## 10. Αναφορές

1. Optimization Theory and Applications, D. A. Pierre, Dover Publications, 1986
2. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
3. http://amrc.desu.edu/web/dpokrajac/478/

# ΚΕΦΑΛΑΙΟ 12: ΠΡΟΣΕΓΓΙΣΕΙΣ ΠΕΠΕΡΑΣΜΕΝΩΝ ΔΙΑΦΟΡΩΝ ΓΙΑ ΑΡΙΘΜΗΤΙΚΕΣ ΠΑΡΑΓΩΓΟΥΣ ΚΑΙ NEWTON ΠΟΛΥΩΝΥΜΙΚΗ ΠΑΡΕΜΒΟΛΗ

## Περιεχόμενα



## 1. Προς τα εμπρός, προς τα πίσω και κεντρικές διαφορές για προσέγγιση παραγώγων:

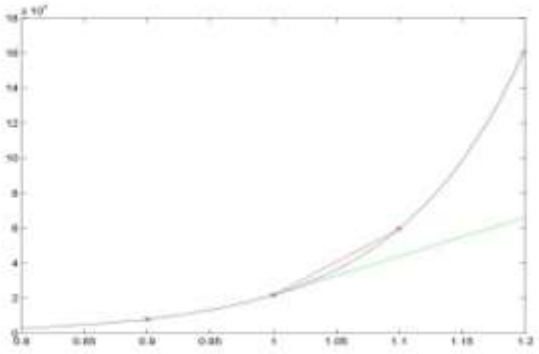
**Πρόβλημα:** Δίνεται ένα σύνολο σημείων δεδομένων κοντά στο σημείο $(x_0,y_0)$:

$$\ldots, (x_{-2},y_{-2}), (x_{-1},y_{-1}), (x_0,y_0), (x_1,y_1), (x_2,y_2), \ldots$$

Ας υποθέσουμε ότι οι τιμές δεδομένων αντιπροσωπεύουν τις τιμές μιας συνάρτησης $y = f(x)$. Βρείτε μια αριθμητική προσέγγιση των παραγώγων $f'(x_0), f''(x_0), \ldots$ της συνάρτησης $y = f(x)$ στο σημείο $(x_0,y_0)$.

**Παράδειγμα:** Τα γραμμικά ηλεκτρικά κυκλώματα αποτελούνται από αντιστάσεις, πυκνωτές, επαγωγείς και πηγές τάσεων και ρευμάτων. Σε ένα απλό δίκτυο αντίστασης-πυκνωτή μίας διόδου (RL) που λειτουργεί με μια πηγή ρεύματος, η τάση $V = V(t)$ αναπτύσσεται κατά μήκος των τερματικών διόδων όταν ένα ρεύμα $I = I(t)$ εφαρμόζεται στη δίοδο εισόδου. Η τάση στην έξοδο $V(t)$ μπορεί να προσδιοριστεί ως άθροισμα της πτώσης της τάσης κατά μήκος της αντίστασης $R\,I(t)$ και της πτώσης της τάσης κατά μήκος στου επαγωγού $L\,I'(t)$. Η παράγωγος $I'(t)$ μπορεί να βρεθεί από το ρεύμα εισόδου $I(t)$ που μετριέται σε διαφορετικές χρονικές στιγμές:

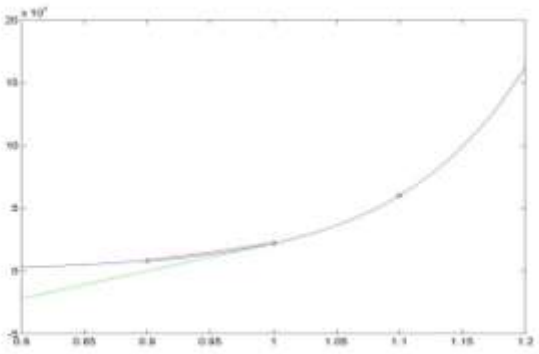
**Λύση:** Η πρώτη παράγωγος $f'(x_0)$ της συνάρτησης $y = f(x)$ στο σημείο $(x_0,y_0)$ μπορεί να προσεγγιστεί με την καμπύλη της τέμνουσας γραμμής η οποία περνάει από δύο σημεία (γραμμική τμηματική παρεμβολή). Η καμπύλη της τέμνουσας γραμμής ονομάζεται προς τα εμπρός, προς τα πίσω ή κεντρικές προσεγγίσεις με διαφορές εάν τα σημεία βρίσκονται στα δεξιά του σημείου $(x_0,y_0)$ (μελλοντικά δεδομένα), στα αριστερά του σημείου $(x_0,y_0)$ (παρελθοντικά δεδομένα) ή και στις δύο πλευρές, αντίστοιχα.



**Προσέγγιση με διαφορές προς τα εμπρός:**
**Forward difference approximation:**
Η τέμνουσα γραμμή περνάει από τα σημεία
$(x_0,y_0)$ και $(x_1,y_1)$.

$$f'(x_0) \approx D_{forward}(f;x_0) = \frac{y_1 - y_0}{x_1 - x_0}$$

Οι προς τα εμπρός διαφορές είναι χρήσιμες στη λύση προβλημάτων αρχικών τιμών για διαφορικές εξισώσεις με μεθόδους ενός βήματος προβλέψεως-διορθώσεως (όπως είναι οι μέθοδοι Euler).

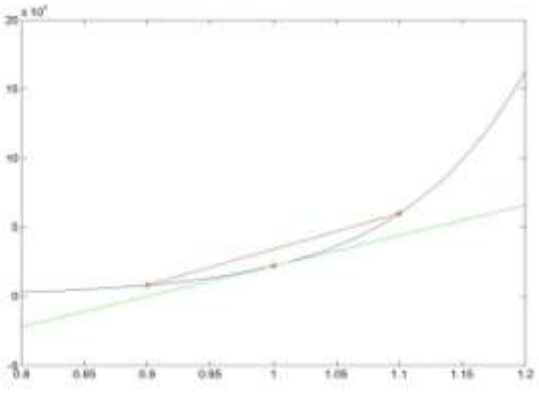


**Προσέγγιση με διαφορές προς τα πίσω:**
**Backward difference approximation:**

Η τέμνουσα γραμμή περνάει από τα σημεία
$(x_{-1},y_{-1})$ και $(x_0,y_0)$.

$$f'(x_0) \approx D_{backward}(f;x_0) = \frac{y_0 - y_{-1}}{x_0 - x_{-1}}$$

Οι προς τα πίσω διαφορές είναι χρήσιμες για την προσέγγιση των παραγώγων εάν οι τιμές δεδομένων είναι διαθέσιμες στο παρελθόν αλλά όχι στο μέλλον (όπως για παράδειγμα οι μέθοδοι τέμνουσας για την εύρεση ριζών και προβλημάτων ελέγχου).



**Κεντρική προσέγγιση με διαφορές:**
**Central difference approximation:**

Η τέμνουσα περνάει από τα σημεία
$(x_{-1},y_{-1})$ and $(x_1,y_1)$.

$$f'(x_0) \approx D_{central}(f;x_0) = \frac{y_1 - y_{-1}}{x_1 - x_{-1}}$$

Οι κεντρικές διαφορές είναι χρήσιμες στη λύση προβλημάτων συνοριακών τιμών για διαφορικές εξισώσεις με μεθόδους πεπερασμένων διαφορών.

Ο παρακάτω κώδικας υπολογίζει τις 3 προσεγγίσεις της συνάρτησης $e^{10x}$ στο σημείο x=1:

```
h  = 0.1; x0 = 1; x_1 = x0-h; x1 = x0 + h;
% the three data points are located at equal distance h (the step size)
y0  =  exp(10*x0);  y_1  =  exp(10*x_1);  y1  =  exp(10*x1);  yDexact  =
10*exp(10*x0);
yDforward = (y1-y0)/h; % simple form of forward difference for equally
spaced grid
yDbackward = (y0-y_1)/h;  % simple form of backward difference
yDcentral = (y1-y_1)/(2*h);  % simple form of central difference
fprintf('Exact  =  %6.2f\nForward  =  %6.2f\nBackward  =  %6.2f\nCentral  =
%6.2f',yDexact,yDforward,yDbackward,yDcentral);
```

```
Exact = 220264.66
Forward = 378476.76
Backward = 139233.82
Central = 258855.29
```

## 2. Σφάλματα αριθμητικής διαφοροποίησης:

Η αριθμητική διαφοροποίηση είναι εγγενώς μια διαδικασία κακής κατάστασης. Δύο παράγοντες προσδιορίζουν σφάλματα που προκαλούνται όταν η παράγωγος $f'(x_0)$ αντικαθίσταται από μια προσέγγιση με διαφορές: σφάλματα αποκοπής και στρογγυλοποίησης (Κεφάλαιο 14)

Θεωρώντας τα ισαπέχοντα σημεία δεδομένων με σταθερό πλάτος βήματος: : $h = x_1 - x_0 = x_0 - x_{-1}$ και εφαρμόζοντας το θεώρημα του Taylor μπορούμε να προσδιορίσουμε τα σφάλματα αποκοπής (truncation) για τις τρις μεθόδους αριθμητικής διαφοροποίησης. Συγκεκριμένα έχουμε:

- **Το σφάλμα αποκοπής προς τα εμπρός προσέγγισης με διαφορές είναι:**

$$f'(x_0) - D_{forward}(f,x_0) = -\frac{h}{2} f''(x), \quad x \in [x_0,x_1]$$

Παρατηρούμε ότι το σφάλμα προς τα εμπρός προσέγγισης με διαφορές είναι ανάλογο του $h$, δηλαδή είναι της τάξης (βαθμού) του $O(h)$. Επίσης, το σφάλμα είναι ανάλογο της δεύτερης παραγώγου της συνάρτησης $f(x)$ σε ένα εσωτερικό σημείο $x$ του προς τα εμπρός διαστήματος με διαφορές.

- **Το σφάλμα αποκοπής προς τα πίσω προσέγγιση με διαφορές είναι:**

$$f'(x_0) - D_{backward}(f,x_0) = \frac{h}{2} f''(x), \quad x \in [x_{-1},x_0]$$

Το σφάλμα της προς τα πίσω προσέγγισης με διαφορές είναι το ίδιο άσχημο με αυτό της προς τα εμπρός προσέγγισης με διαφορές. Το σφάλμα είναι βαθμού $O(h)$ και είναι ανάλογο της δεύτερης παραγώγου της συνάρτησης $f(x)$ σε ένα εσωτερικό σημείο $x$ του προς τα πίσω διαστήματος διαφοράς.

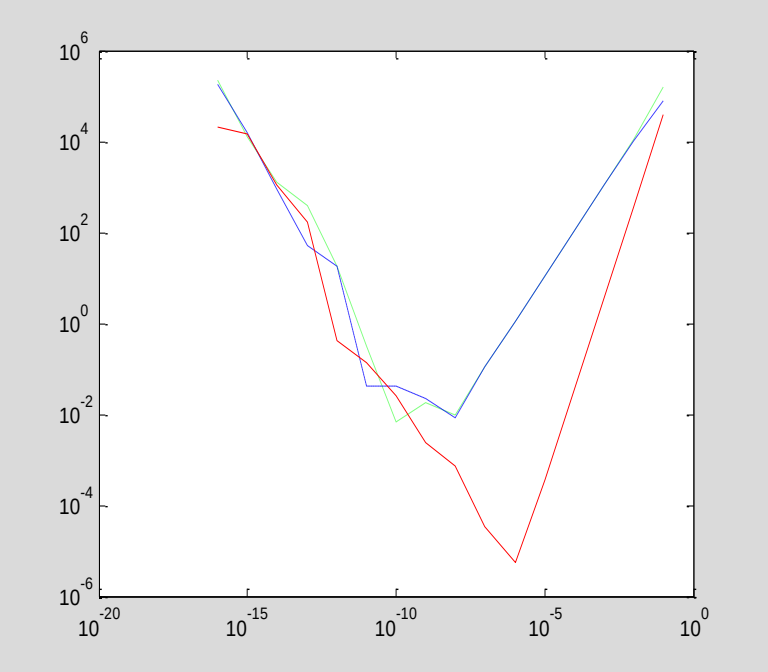
- **Το σφάλμα αποκοπής προς κεντρική προσέγγιση με διαφορές είναι:**

$$\mathbf{f'(x_0) - D_{central}(f,x_0) = -\frac{h^2}{6} f'''(x), \quad x \in [x_{-1},x_1]}$$

Το σφάλμα αποκοπής της κεντρικής προσέγγισης με διαφορές είναι ανάλογο του $\boldsymbol{h^2}$ και όχι του ***h,*** δηλαδή συμπεριφέρεται ως $\boldsymbol{O(h^2)}$**.** Το σφάλμα αυτό είναι επίσης ανάλογο της τρίτης παραγώγου της συνάρτησης ***f(x)*** σε ένα εσωτερικό σημείο ***x*** του κεντρικού διαστήματος διαφορών. Η κεντρική προσέγγιση διαφοράς είναι ένας μέσος όρος των προς τα εμπρός και προς τα πίσω διαφορών και παράγει μια πολύ πιο ακριβέστερη προσέγγιση της παραγώγου για μια δεδομένη μικρή τιμή του ***h,*** σε σύγκριση με τις προς τα εμπρός και προς τα πίσω διαφορές. Αν οι τιμές των δεδομένων είναι διαθέσιμες και από τα δεξιά και από τα αριστερά του σημείου $\boldsymbol{(x_0,y_0)}$, η χρήση της κεντρικής προσέγγισης με διαφορές είναι η προτιμότερη.

Ο παρακάτω κώδικας υπολογίζει τις 3 προσεγγίσεις της παραγώγου της συνάρτησης $\boldsymbol{e^{10x}}$ στο σημείο x=1 για διαφορετικές τιμές του ***h*** και απεικονίζει το απόλυτο σφάλμα της εμπρός (πράσινο), πίσω (μπλε), και κεντρικής (κόκκινο) πεπερασμένης διαφοράς για την προσέγγιση της παραγώγου $\boldsymbol{10e^{10x}}$ :

```
h = logspace(-1,-16,16);
x0 = 1; x_1 = x0-h; x1 = x0+h;
y0 = exp(10*x0); y_1 = exp(10*x_1); y1 = exp(10*x1); yDe =
10*exp(10*x0);
yDf = (y1-y0)./h; yDb = (y0-y_1)./h; yDc = (y1-y_1)./(2*h);
eDf = abs(yDf-yDe); eDb = abs(yDb-yDe); eDc = abs(yDc-yDe);
loglog(h,eDf,'g:',h,eDb,'b--',h,eDc,'r');
```




**Σχήμα 11.1**: Το σφάλμα των 3ων προσεγγίσεων της παραγώγου της συνάρτησης $e^{10x}$

Από το γράφημα συμπεραίνουμε ότι το σφάλμα μικραίνει καθώς η τιμή του ***h*** μικραίνει και μεγαλώνει όταν τιμές του ***h*** μικρύνουν αρκετά. (Γιατί;)

Παρατηρούμε ότι αν το πλάτος βήματος ***h*** μεταξύ δύο σημείων γίνει μικρότερο, το σφάλμα της προσέγγισης με διαφορές μειώνεται γρηγορότερα για κεντρικές προσεγγίσεις με διαφορές και πιο αργά για προς τα εμπρός και προς τα πίσω προσεγγίσεις με διαφορές. Για παράδειγμα, αν το ***h*** μειωθεί κατά ***10***, το σφάλμα της κεντρικής προσέγγισης με διαφορές μειώνεται κατά ***100***, ενώ τα σφάλματα των προς τα εμπρός και προς τα πίσω διαφορών μειώνονται μόνο κατά ***10***.

Όταν το ***h*** γίνεται πολύ μικρό, οι προσεγγίσεις με διαφορές είναι σχεδόν ίσων τιμών με αυτές του ***f(x)*** στα δύο σημεία. Οποιοδήποτε σφάλμα στρογγυλοποίησης λόγω υπολογισμών του ***f(x)*** μεγεθύνεται από ένα συντελεστή ***1/h.*** Ως αποτέλεσμα, το σφάλμα στρογγυλοποίησης μεγαλώνει με το ***h*** για πολύ μικρές τιμές του ***h.*** Αν ***eps*** είναι η ακρίβεια της μηχανής (δες Κεφάλαιο 14) τότε το βέλτιστο πλάτος βήματος $h = h_{opt}$ μπορεί να υπολογιστεί από την ελαχιστοποίηση του αθροίσματος των σφαλμάτων προσέγγισης και στρογγυλοποίησης:

- **Προσέγγιση με διαφορές προς τα εμπρός:**

$$e_{forward} = |f'(x_0) - D_{forward}(f,x_0)| < M_2 \frac{h}{2} + 2\frac{eps}{h},$$

όπου $M_2 = max\,|f''(x)|$. Το ελάχιστο σφάλμα συμβαίνει στο $h = h_{opt} = 2\sqrt{eps / M_2}$, όπου $e_{forward} = 2\sqrt{epsM_2}$.

- **Κεντρική προσέγγιση με διαφορές:**

$$e_{central} = |f'(x_0) - D_{central}(f,x_0)| < M_3 \frac{h^2}{6} + 2\frac{eps}{h},$$

όπου $M_3 = max\,|f'''(x)|$. Το ελάχιστο σφάλμα συμβαίνει για $h = h_{opt} = \sqrt[3]{6eps / M_3}$, όταν $e_{forward} = 3\sqrt[3]{eps^2 M_3 / 6}$.

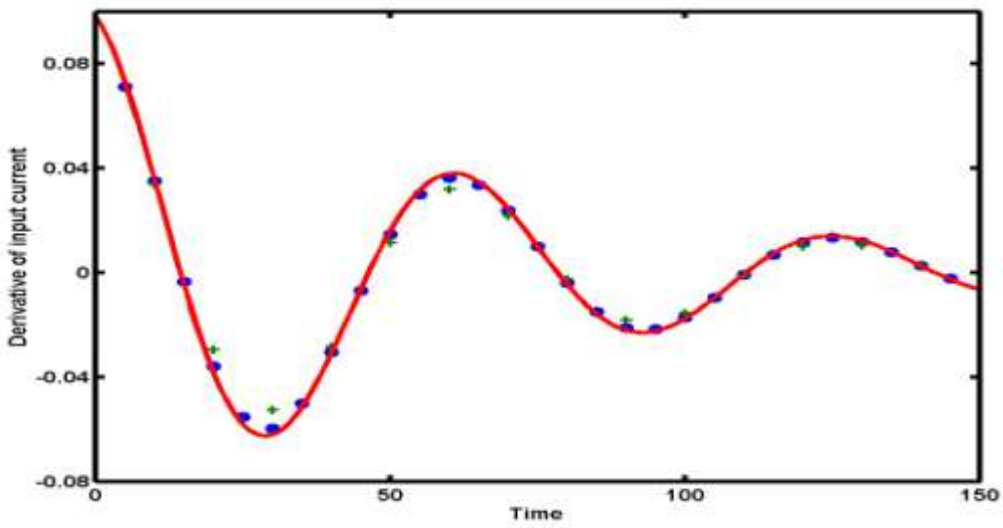
Παρακάτω υπολογίζουμε τις βέλτιστες τιμές του ***h*** για την συνάρτηση $e^{10x}$ και το αντίστοιχο σφάλμα για εμπρός και κεντρική προσέγγιση της παραγώγου της $e^{10x}$.

```
M2 = 100*exp(10*(x0+0.1));
M3 = 1000*exp(10*(x0+0.1));
hoptForward = 2*(eps/M2)^(1/2)
hoptCentral = (6*eps/M3)^(1/3)
eoptForward = 2*(eps*M2)^(1/2)
eoptCentral = 3*(eps^2*M3/6)^(1/3)

hoptForward =
  1.2180e-011
hoptCentral =
  2.8127e-008
eoptForward =
  7.2924e-005
eoptCentral =
  2.3683e-008
```

Παρατηρούμε ότι η κεντρική προσέγγιση για μεγαλύτερο ***h*** δίνει καλλίτερο σφάλμα! (Γιατί αυτό είναι επιθυμητό;)

**Παράδειγμα:** Δύο κεντρικές προσεγγίσεις με διαφορές εφαρμόζονται για να υπολογιστεί η παράγωγος του ρεύματος ***I'(t)***: τα πράσινα συν αντιστοιχούν στο ***h = 10*** και οι μπλε τελείες

αντιστοιχούν στο ***h = 5.*** Η ακριβής παράγωγος ***I'(t)*** σημειώνεται με τη κόκκινη συμπαγή καμπύλη.




## 3. Η παρεμβολή Newton με προς τα εμπρός διαφορές:

**Πρόβλημα:** Δίνεται ένα σύνολο σημείων δεδομένων ***(n+1):***

$$\boxed{(x_1,y_1),\ (x_2,y_2),\ \ldots,\ (x_n,y_n),\ (x_{n+1},y_{n+1})}$$

Βρείτε ένα πολυώνυμο ***n*** βαθμού, $y = P_n(x)$, που να περνάει από όλα τα ***(n+1)*** σημεία δεδομένων. Το πολυώνυμο παρεμβολής τύπου Newton είναι ένα διακριτό πολυώνυμο τύπου Taylor της μορφής:

$$\boxed{y = P_n(x) = c_0 + c_1 (x-x_1) + c_2(x-x_1)(x-x_2) + c_n (x - x_1)(x - x_2)\ldots(x-x_n)}$$

όπου οι συντελεστές $c_j$ υπολογίζονται από τις διαγώνιες διαιρεμένες διαφορές.

**Σύγκριση με την μέθοδο Van der Monde και Lagrange:** Η πολυωνυμική παρεμβολή Lagrange είναι βολική όταν τα ίδια σημεία πλέγματος $[x_1,x_2,\ldots,x_{n+1}]$ χρησιμοποιούνται επανειλημμένως σε αρκετές εφαρμογές. Η παρεμβολή Lagrange, ωστόσο, δεν είναι χρήσιμη όταν επιπλέον σημεία δεδομένων προστίθενται ή αφαιρούνται για να βελτιωθεί η εμφάνιση της καμπύλης παρεμβολής. Το σύνολο δεδομένων πρέπει να το υπολογίσουμε ξανά από την αρχή κάθε φορά που σημεία δεδομένων αλλάζουν. Η πολυωνυμική παρεμβολή του Newton είναι πιο αποδοτική όταν προσθέτουμε και αφαιρούμε επιπλέον σημεία δεδομένων. Επίσης, για τον υπολογισμό του μπορούμε να εφαρμόσουμε εύκολα τον αλγόριθμο του Horner. Αξίζει να σημειώσετε, ότι οι μέθοδοι Newton, Van der Monde, και Lagrange το ίδιο πολυώνυμο παρεμβολής, $y = P_n(x)$, ***n*** βαθμού που συνδέει ***(n+1)*** σημεία δεδομένων (Γιατί;). Η διαφορά μεταξύ των πολυωνύμων παρεμβολής Van der Monde, Lagrange και Newton είναι μόνο ο τρόπος προσδιορισμού των.

**Λύση:** Οι συντελεστές $c_j$ του πολυωνύμου Newton μπορούν να βρεθούν από τις συνθήκες: $P_n(x_k) = y_k$, δηλαδή από την λύση του γραμμικού σύστηματος: $Ac = y$. Ο αλγόριθμος της απαλοιφής

Gauss ή οι διάφοροι μέθοδοι απαλοιφής που έχουν εισαχθεί στο κεφάλαιο 7, μπορούν να χρησιμοποιηθούν για να αποδειχθεί ότι το $c_j = d_{j,j}$, όπου $d_{j,j}$ είναι τα διαγώνια στοιχεία του πίνακα των προς τα εμπρός διαιρεμένων διαφορών:

| ***Σημεία πλέγματος grid points*** | ***Τιμές δεδομένων data values*** | ***Διαφορές πρώτης τάξεως first differences*** | ***Διαφορές δεύτερης τάξεως second differences*** | ***Διαφορές τρίτης τάξεως third differences*** | ***Διαφορές τέταρτης τάξεως fourth differences*** |
|---|---|---|---|---|---|
| $x_1$ | $y_1$ | | | | |
| $x_2$ | $y_2$ | $f[x_1,x_2]$ | | | |
| $x_3$ | $y_3$ | $f[x_2,x_3]$ | $f[x_1,x_2,x_3]$ | | |
| $x_4$ | $y_4$ | $f[x_3,x_4]$ | $f[x_2,x_3,x_4]$ | $f[x_1,x_2,x_3,x_4]$ | |
| $x_5$ | $y_5$ | $f[x_4,x_5]$ | $f[x_3,x_4,x_5]$ | $f[x_2,x_3,x_4,x_5]$ | $f[x_1,x_2,x_3,x_4,x_5]$ |

Οι μηδενικές διαιρεμένες διαφορές είναι απλά οι τιμές της συνάρτησης $y = f(x)$ στο σημείο $(x_k,y_k)$. Οι πρώτες διαιρεμένες διαφορές $f[x_k,x_{k+1}]$ είναι προσέγγιση με διαφορές προς τα εμπρός για τη συνάρτηση $y = f(x)$ στο $(x_k,y_k)$:

$$f[x_k,x_{k+1}] = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

Η δεύτερης, τρίτης και μεγαλύτερης τάξης προς τα εμπρός διαιρεμένη διαφορά μπορεί να κατασκευαστεί με τη χρήση του αναδρομικού κανόνα:

$$f[x_k,x_{k+1},\ldots,x_{k+m}] = \frac{f[x_{k+1},\ldots,x_{k+m}] - f[x_k,\ldots,x_{k+m-1}]}{x_{k+m} - x_k}$$

Με τη χρήση διαιρεμένων διαφορών $c_{j-1} = f[x_1,x_2,\ldots,x_j]$, το πολυώνυμο παρεμβολής του Newton $y = P_n(x)$ έχει την άμεση αναπαράσταση:

$$P_n(x) = \sum_{j=1}^{n+1} f[x_1,x_2,\ldots,x_j] \left( \prod_{i=1}^{j-1} (x - x_i) \right)$$

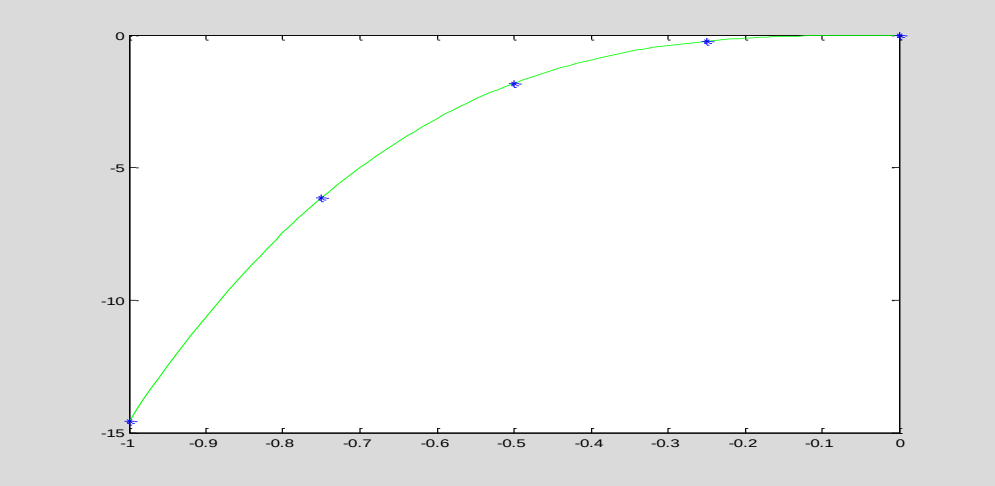
Ο παρακάτω κώδικας υπολογίζει το πολυώνυμο παρεμβολής Newton λύνοντας το σύστημα των συνθηκών παρεμβολής.

```
% example of explicit computation of coefficients of Newton polynomials
x = [ -1,-0.75,-0.5,-0.25,-0]; n = length(x)-1;
y = [ -14.58,-6.15,-1.82,-0.23,-0.00];
A = ones(n+1,1); % the coefficient matrix for linear system Ac = y
for j = 1 : n
A = [ A, A(:,j).*(x'-x(j))];
end
A, c = A\(y'); c = c'
A = 1.0000         0         0         0         0
    1.0000    0.2500         0         0         0
    1.0000    0.5000    0.1250         0         0
    1.0000    0.7500    0.3750    0.0938         0
    1.0000    1.0000    0.7500    0.3750    0.0938
```

```
c =  -14.5800   33.7200  -32.8000   14.5067    0.2133
```

Η συνάρτηση MatLab που υπολογίζει τον πίνακα πεπερασμένων διαφορών και τους συντελεστές του πολυωνύμου Newton και παράγει την γραφική του παράσταση δίδεται παρακάτω.

```
function [yi,c] = NewtonInter(x,y,xi)
% Newton interpolation algorithm
% x,y - row-vectors of (n+1) data values (x,y)
% xi - a row-vector of x-values, where interpolation is to be found
% yi - a row-vector of interpolated y-values
% c - coefficients of Newton interpolating polynomial
n = length(x) - 1; % the degree of interpolation polynomial
ni = length(xi); % the number of x-values, where interpolation is to be
found
D = zeros(n+1,n+1); % the matrix for Newton divided differences
D(:,1) = y'; % zero-order divided differences
for k = 1 : n
    D(k+1:n+1,k+1) = (D(k+1:n+1,k)-D(k:n,k))./(x(k+1:n+1)-x(1:n-k+1))';
end
c = diag(D);
% computation of values of the Newton interpolating polynomial at
values of xi
% the algorithm uses the Horner's rule for polynomial evaluation
yi = c(n+1)*ones(1,ni); % initialization of the vector yi as the
coefficient of the highest degree
for k = 1 : n
    yi = c(n+1-k)+yi.*(xi-x(n+1-k)); % nested multiplication
end
x = [ -1,-0.75,-0.5,-0.25,-0]; y = [ -14.58,-6.15,-1.82,-0.23,-0.00];
xInt = -1 : 0.01 : 0;
[yInt,c] = NewtonInter(x,y,xInt);
c = c', plot(xInt,yInt,'g',x,y,'b*');
c =  -14.5800   33.7200  -32.8000   14.5067    0.2133
```




## 4. Πεπερασμένες διαφορές με MATLAB

Έχουμε τα σημεία δεδομένων $x_1,x_2,\ldots,x_n,x_{n+1}$ που είναι ισαπέχοντα με σταθερό πλάτος βήματος $h = x_2 - x_1$. Τότε οι διαιρεμένες διαφορές μπορούν να ξαναγραφτούν ως:

$$f[x_k,x_{k+1},\ldots,x_{k+m}] = \frac{\Delta_m f[x_k]}{m!\,h^m},$$

Όπου $\Delta_m f[x_k]$ είναι η ***m-th*** προς τα εμπρός διαφορά της συνάρτησης $y = f(x)$ στο σημείο $(x_k,y_k)$.

$$\Delta_1 f[x_0] = y_1 - y_0$$
$$\Delta_2 f[x_0] = y_2 - 2y_1 + y_0$$
$$\Delta_3 f[x_0] = y_3 - 3y_2 + 3y_1 - y_0$$
$$\Delta_4 f[x_0] = y_4 - 4y_3 + 6y_2 - 4y_1 - y_0$$
$$\Delta_5 f[x_0] = y_5 - 5y_4 + 10y_3 - 10y_2 + 5y_1 - y_0$$

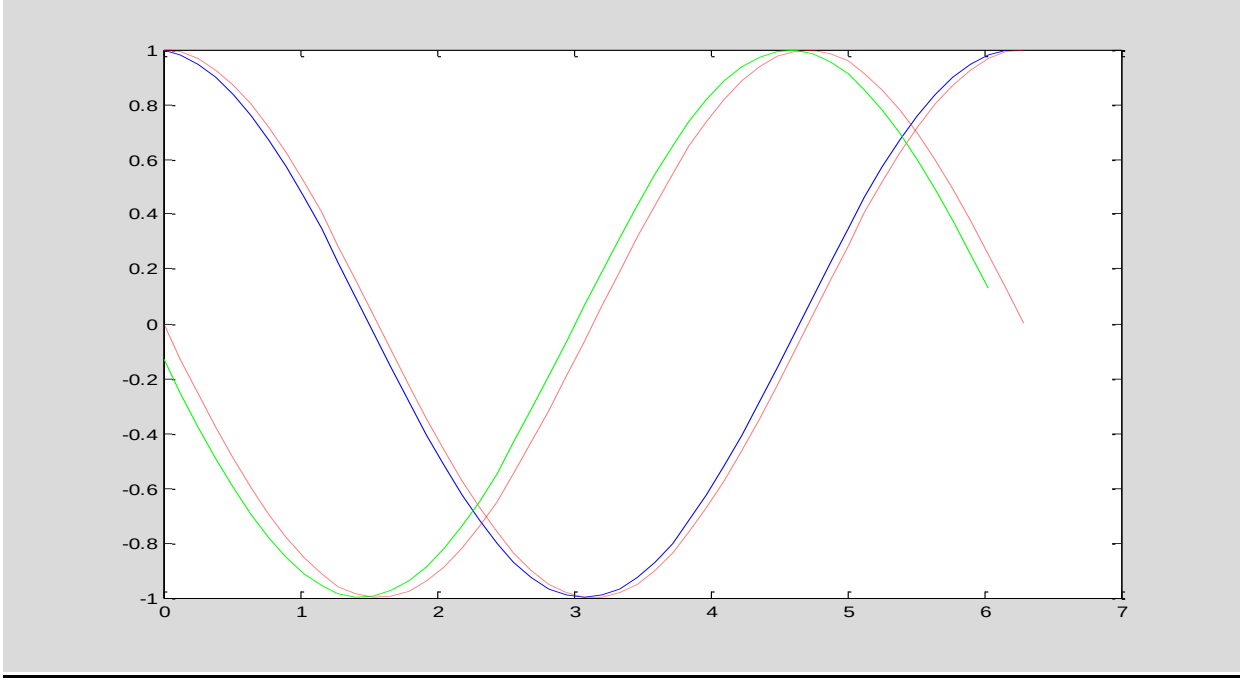
Οι παράγωγοι των πολυωνύμων παρεμβολής $y = P_n(x)$ προσεγγίζουν παραγώγους της συνάρτησης $y = f(x)$. Ταιριάζοντας την ***n-th*** παράγωγο του πολυωνύμου $y = P_n(x)$ με $f^{(n)}(x_0)$, βρίσκουμε την προς τα εμπρός προσέγγιση με διαφορές παραγώγων μεγαλύτερου βαθμού:

$$f^{(n)}(x_0) \approx \frac{\Delta_n f[x_0]}{h^n}$$

- **diff(y):** υπολογίζει την προς τα εμπρός διαφορά πρώτου βαθμού για ένα δεδομένο διάνυσμα ***y***.
- **diff(y,n):** υπολογίζει την προς τα εμπρός διαφορά ***n-th*** βαθμού για ένα δεδομένο διάνυσμα ***y***.
- **gradient(u):** υπολογίζει οριζόντιες και κάθετες προς τα εμπρός διαφορές για ένα δεδομένο πίνακα για να προσεγγίσει τις ***x***- και ***y***-παραγώγους του ***u(x,y): grad(u) = [$u_x$,$u_y$]***
- **del2(u):** υπολογίζει τη διακριτή Λαπλασιανή για ένα δεδομένο πίνακα ***u:*** $\Delta u = u_{xx} + u_{yy}$

**Παράδειγμα:**

```
n = 50; x = linspace(0,2*pi,n); h = x(2)-x(1); y = sin(x); % function
y1 = diff(y)/h; y2 = diff(y,2)/h^2; y1ex = cos(x); y2ex = -sin(x);%
derivatives
plot(x(1:n-1),y1,'b',x,y1ex,':r',x(1:n-2),y2,'g',x,y2ex,':r');
```




## 5. Ιεραρχίες υψηλού βαθμού προσεγγίσεων με διαφορές:

Τα πολυώνυμα παρεμβολής Newton και οι πίνακες διαιρεμένων διαφορών μπορούν να κατασκευαστούν για προς τα πίσω διαφορές εφόσον η σειρά των σημείων δεδομένων $x_1,x_2,\ldots,x_n,x_{n+1}$ είναι τυχαία. Αν βάλουμε τα σημεία δεδομένων σε φθίνουσα σειρά, το πολυώνυμο Newton αντιπροσωπεύει τις προς τα πίσω διαφορές. Είναι πιο δύσκολο να κατασκευάσουμε πίνακες για κεντρικές διαφορές. Εφόσον οι κεντρικές διαφορές είναι και οι πιο ακριβείς προσεγγίσεις, πρέπει να σχεδιαστούν ειδικοί αλγόριθμοι για να αυτοματοποιηθεί η παραγωγή των συντελεστών κεντρικών προσεγγίσεων με διαφορές για παραγώγους υψηλών τάξεων.

- **Ιεραρχία των προς τα εμπρός διαφορών**

Ας υποθέσουμε ότι τα σημεία δεδομένων είναι ισαπέχοντα με σταθερό πλάτος βήματος ***h.*** Επιλέξτε ένα σημείο $(x_0,y_0)$ και βρείτε μια προσέγγιση με διαφορές προς τα εμπρός για την ***n-th*** παράγωγο ως αποτέλεσμα εσωτερικού γινομένου:

$$f^{(n)}(x_0) \approx \frac{1}{h^n} D_n * y$$

όπου $y = [\, y_0, y_1, y_2, \ldots]$

```
     n = 100; A = diag(ones(n-1,1),1)-diag(ones(n,1));
     m = 8; B = A;
     for k = 1 : m-1
D(k,:) = B(1,1:m);
B = B*A;
end
D = D
D   -1     1     0     0     0     0     0     0
```

```
     1    -2     1     0     0     0     0     0
    -1     3    -3     1     0     0     0     0
     1    -4     6    -4     1     0     0     0
    -1     5   -10    10    -5     1     0     0
     1    -6    15   -20    15    -6     1     0
    -1     7   -21    35   -35    21    -7     1
```

**ιεραρχία κεντρικών διαφορών**

Ας υποθέσουμε ότι τα σημεία δεδομένων είναι ισαπέχοντα με σταθερό πλάτος βήματος ***h.*** Επιλέξτε ένα σημείο $(x_0,y_0)$ και βρείτε μια κεντρική προσέγγιση με διαφορές για την ***n-th*** παράγωγο ως αποτέλεσμα εσωτερικού γινομένου:

$$f^{(2m-1)}(x_0) \approx \frac{1}{2h^{2m-1}} D_{2m-1} * y; \qquad f^{(2m)}(x_0) \approx \frac{1}{h^{2m}} D_{2m} * y$$

όπου $y = [\ \ldots, y_{-2}, y_{-1}, y_0, y_1, y_2, \ \ldots]$

```
n = 100; A = diag(ones(n-1,1),1)-diag(ones(n-1,1),-1);
A2 = diag(ones(n-1,1),1)+diag(ones(n-1,1),-1)-2*diag(ones(n,1));
m = 4; B = A; C = A2; k = 1;
while (k < (2*m-2) )
D(k,:) = B(m,1:2*m-1);
D(k+1,:) = C(m,1:2*m-1);
B = B*A2; C = C*A2;
k = k+2;
end
D = D
D =  0     0    -1     0     1     0     0
     0     0     1    -2     1     0     0
     0    -1     2     0    -2     1     0
     0     1    -4     6    -4     1     0
    -1     4    -5     0     5    -4     1
     1    -6    15   -20    15    -6     1
```

## 6. Μέθοδος παρεμβολής Richardson για διαφορές υψηλής τάξεως

Οι αναδρομικοί τύποι διαφορών για παραγώγους μπορούν να προσδιοριστούν ακυρώνοντας το σφάλμα αποκοπής σε κάθε τάξη αριθμητικής προσέγγισης. Η μέθοδος αυτή ονομάζεται "μέθοδος παρεκβολής Richardson". Η μέθοδος μπορεί να χρησιμοποιηθεί μόνο άν οι τιμές των δεδομένων είναι ισαπέχουσες με σταθερό πλάτος βήματος ***h***.

- **Αναδρομικές διαφορές προς τα εμπρός:**

Η ιεραρχία των προς τα εμπρός διαφορών για πρώτης και μεγαλύτερης τάξης παραγώγους έχει σφάλμα αποκοπής της τάξεως ***O(h)***. Ορίζουμε την προς τα εμπρός προσέγγιση με διαφορές για μία παράγωγο $f^{(n)}(x_0)$ ως $D_1(h)$. Υπολογίζουμε την προσέγγιση με δύο πλάτη βήματος ***h*** και ***2h:***

$$f^{(n)}(x_0) = D_1(h) + \alpha h; \qquad f^{(n)}(x_0) = D_1(2h) + 2\alpha h$$

όπου $\alpha$ είναι ένας άγνωστος συντελεστής για το σφάλμα αποκοπής. Ακυρώνοντας το σφάλμα αποκοπής της τάξεως ***O(h),*** ορίζουμε μια καινούργια προς τα εμπρός προσέγγιση με διαφορές για την ίδια παράγωγο:

$$f^{(n)}(x_0) = 2D_1(h) - D_1(2h) = D_2(h)$$

Η νέα προς τα εμπρός προσέγγιση με διαφορές $D_2(h)$ για την ίδια παράγωγο είναι πιο ακριβής εφόσον το σφάλμα αποκοπής είναι της τάξεως $O(h^2)$.

- **Παράγωγος πρώτης τάξεως (βαθμού) $f'(x_0)$:**

Η πρώτης τάξης προσέγγιση $D_1(h)$ είναι διαιρεμένη διαφορά δύο σημείων, ενώ η δεύτερης τάξης προσέγγιση $D_2(h)$ είναι διαιρεμένη διαφορά τριών σημείων:

$$D_1(h) = \frac{y_1 - y_0}{h}, \qquad D_2(h) = \frac{-y_2 + 4y_1 - 3y_0}{2h}$$

- **Παράγωγος δεύτερης τάξης $f''(x_0)$:**

Η πρώτης τάξεως προσέγγιση $D_1(h)$ είναι διαιρεμένη διαφορά τριών σημείων, ενώ η δεύτερης τάξης προσέγγιση $D_2(h)$ είναι διαιρεμένη διαφορά τεσσάρων σημείων:

$$D_1(h) = \frac{y_2 - 2y_1 + y_0}{h^2}, \qquad D_2(h) = \frac{-y_3 + 4y_2 - 5y_1 + 2y_0}{h^2}$$

Η διαδικασία μπορεί να συνεχιστεί για να βρούμε την προς τα εμπρός προσέγγιση με διαφορές υψηλής τάξεως $D_m(h)$ με σφάλμα αποκοπής $O(h^m)$. Ο αναδρομικός τύπος για την προς τα εμπρός παρεκβολή με διαφορά του Richardson:

$$D_{m+1}(h) = D_m(h) + \frac{D_m(h) - D_m(2h)}{2^k - 1}$$

- **Αναδρομικές κεντρικές διαφορές**

Η ιεραρχία των κεντρικών διαφορών για πρώτης και υψηλότερης τάξης παραγώγους έχει σφάλμα αποκοπής της τάξεως $O(h^2)$. Ορίζουμε την κεντρική προσέγγιση με διαφορές για μια παράγωγο $f^{(n)}(x_0)$ ως $D_1(h)$. Υπολογίζουμε τη προσέγγιση με δύο πλάτη βήματος $h$ και $2h$:

$$f^{(n)}(x_0) = D_1(h) + \alpha h^2; \quad f^{(n)}(x_0) = D_1(2h) + 4\alpha h^2$$

όπου $\alpha$ είναι ένας άγνωστος συντελεστής για το σφάλμα αποκοπής. Ακυρώνοντας το σφάλμα αποκοπής της τάξεως $O(h^2)$, ορίζουμε μια καινούργια κεντρική προσέγγιση με διαφορές για την ίδια παράγωγο:

$$f^{(n)}(x_0) = \frac{4D_1(h) - D_2(2h)}{3}$$

Η νέα κεντρική προσέγγιση με διαφορές $D_2(h)$ για την ίδια παράγωγο είναι πιο ακριβής εφόσον το σφάλμα αποκοπής είναι της τάξεως $O(h^4)$.

- **Παράγωγος πρώτης τάξεως $f'(x_0)$:**

Η πρώτης τάξεως προσέγγιση $D_1(h)$ είναι διαιρεμένη διαφορά τριών σημείων, ενώ η δεύτερης τάξης προσέγγιση $D_2(h)$ είναι διαιρεμένη διαφορά πέντε σημείων:

$$D_1(h) = \frac{y_1 - y_{-1}}{2h}, \qquad D_2(h) = \frac{-y_2 + 8y_1 - 8y_{-1} + y_{-2}}{12h}$$

- **Παράγωγος δεύτερης τάξης:**

Η πρώτης τάξεως προσέγγιση $D_1(h)$ είναι διαιρεμένη διαφορά τριών σημείων, ενώ η δεύτερης τάξης προσέγγιση $D_2(h)$ είναι διαιρεμένη διαφορά πέντε σημείων:
:

$$D_1(h) = \frac{y_1 - 2y_0 + y_{-1}}{h^2}, \qquad D_2(h) = \frac{-y_2 + 16y_1 - 30y_0 + 16y_{-1} - y_{-2}}{12h^2}$$

Η διαδικασία μπορεί να συνεχιστεί για να βρούμε την κεντρική προσέγγιση με διαφορές $D_m(h)$ με σφάλμα αποκοπής $O(h^{2m})$ . Ο αναδρομικός τύπος για την προς τα εμπρός παρεκβολή με διαφορά του Richardson:

$$D_{m+1}(h) = D_m(h) + \frac{D_m(h) - D_m(2h)}{4^k - 1}$$

- **Αριθμητικοί Αλγόριθμοι**

Για να υπολογίσουμε τη κεντρική προσέγγιση με διαφορές μιας παραγώγου $f^{(n)}(x_0)$ ***μέχρι την τάξη*** $m$, μικρότερης τάξης κεντρικές προσεγγίσεις με διαφορές πρέπει να υπολογιστούν με μεγαλύτερα πλάτη βήματος: $h, 2h, 4h, 8h, \ldots, (m-1)h$.

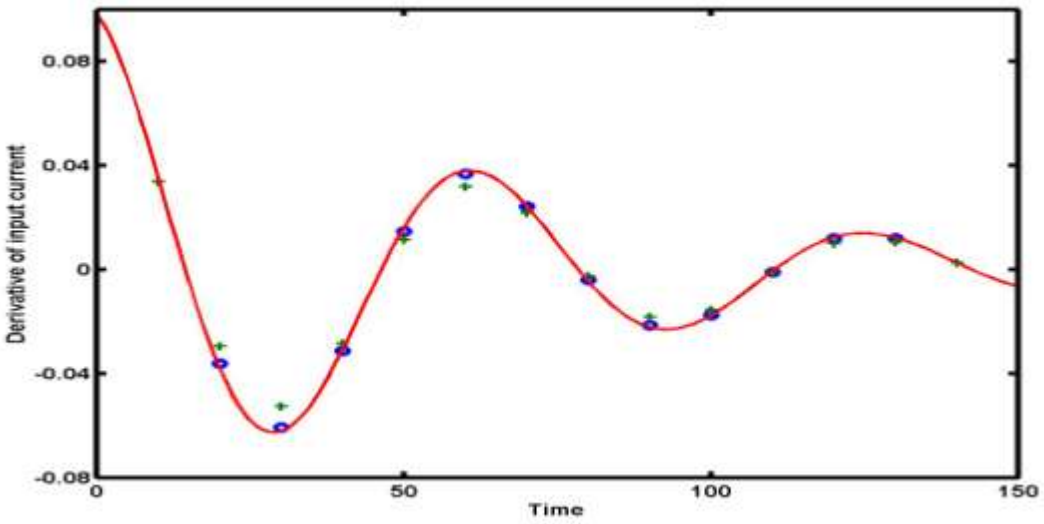
Οι προσεγγίσεις μπορούν να τοποθετηθούν σε ένα πίνακα αναδρομικών παραγώγων:

| *step size* | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ |
|---|---|---|---|---|---|
| $h$ | $D_1(h)$ | | | | |
| $2h$ | $D_1(2h)$ | $D_2(h)$ | | | |
| $4h$ | $D_1(4h)$ | $D_2(2h)$ | $D_3(h)$ | | |
| $8h$ | $D_1(8h)$ | $D_2(4h)$ | $D_3(2h)$ | $D_4(h)$ | |
| $16h$ | $D_1(16h)$ | $D_2(8h)$ | $D_3(4h)$ | $D_4(2h)$ | $D_5(h)$ |

Οι διαγώνιες καταχωρήσεις είναι τιμές μεγαλύτερης τάξης κεντρικών προσεγγίσεων με διαφορές για την παράγωγο $f^{(n)}(x_0)$ όπου $x_0$ είναι το κεντρικό σημείο του διαστήματος των σημείων $x = x_0 - 2^{k-1}h$ και $x = x_0 + 2^{k-1}h$ για $k = 1,2,\ldots,m$. Η προσέγγιση υψηλότερης τάξης $D_k(h)$ έχει σφάλμα αποκοπής $O(h^{2k})$. Εάν το $h$ είναι μικρό, το σφάλμα αποκοπής μειώνεται με μεγάλη ταχύτητα για μεγαλύτερο $k$. Μια βέλτιστη τάξη $m$ υπάρχει στο ελάχιστο του αθροίσματος των σφαλμάτων αποκοπής και στρογγυλοποίησης.

**Παράδειγμα:** Η γραφική παράσταση που ακολουθεί παρουσιάζει τις κεντρικές προσεγγίσεις με διαφορές $D_1(h)$ και $D_2(h)$ της παραγώγου για το ρεύμα $I(t)$ με πλάτος βήματος $h = 10$. Οι μπλε κύκλοι μπορούν να βρεθούν με κεντρικές διαφορές πέντε σημείων $D_2(h)$, οι πράσινοι σταυροί μπορούν να βρεθούν με κεντρικές διαφορές τριών σημείων $D_1(h)$ και η ακριβής παράγωγος $I'(t)$

αντιπροσωπεύεται από την συμπαγή κόκκινη καμπύλη. Η προσέγγιση με διαφορά πέντε σημείων ***D*$_2$*(h)*** είναι εμφανώς πιο ακριβής από την από τη διαφορά τριών σημείων ***D*$_1$*(h)*.**




---

## 7. Αναφορές


1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. https://sites.google.com/a/uni-konstanz.de/na09/Home/methods/fpns
3. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
4. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
5. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
6. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
7. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.

# ΚΕΦΑΛΑΙΟ 13: ΚΑΝΟΝΕΣ ΑΘΡΟΙΣΗΣ ΓΙΑ ΑΡΙΘΜΗΤΙΚΗ ΟΛΟΚΛΗΡΩΣΗ

## Περιεχόμενα



## 1. Κανόνες τραπεζίου, Simpson και μέσου σημείου για προσέγγιση ολοκληρωμάτων:

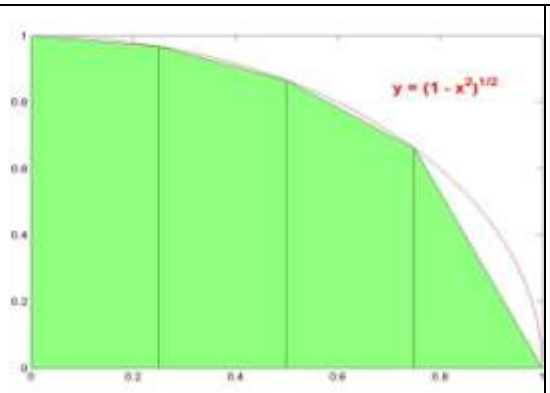
**Πρόβλημα:** Δίνεται ένα σύνολο σημείων δεδομένων**:**

$$(x_0,y_0),\ (x_1,y_1),\ (x_2,y_2),\ \ldots,\ (x_n,y_n)$$

Ας υποθέσουμε ότι τα σημεία δεδομένων αντιπροσωπεύουν μια συνάρτηση $y = f(x)$. Ας υποθέσουμε επίσης ότι τα σημεία δεδομένων είναι ισαπέχοντα με σταθερό πλάτος βήματος $h = x_1 - x_0$. Να βρείτε μια αριθμητική προσέγγιση για το ολοκλήρωμα, η οποία να είναι το σημειωμένο εμβαδό κάτω από την καμπύλη $y = f(x)$ και μεταξύ των άκρων $(x_0,y_0)$ και $(x_n,y_n)$.

**Παράδειγμα:** Τα γραμμικά ηλεκτρικά κυκλώματα μπορούν εύκολα να παραχθούν σε μικρογραφία αν δεν συμπεριλαμβάνουν ογκώδης επαγωγούς. Όταν ένα ρεύμα $I = I(t)$ εφαρμόζεται σε μια θύρα εισόδου ενός δικτύου αντίστασης-πυκνωτή μιας εισόδου, η τάση $V = V(t)$ αναπτύσσεται κατά μήκος όλων των εισόδων τερματικών. Στη χρονική στιγμή $t = T$, η απόδοση της τάσης μπορεί να προσδιοριστεί ως το άθροισμα της πτώσης της τάσης στην αντίσταση (η οποία είναι $R\ I(T)$) και της πτώσης της τάσης στον πυκνωτή (η οποία είναι $V_0 + \int_0^T dtI(t) / C$), όπου $V_0$ είναι η αρχική τάση. Αν το εισαγόμενο ρεύμα είναι $I(t)$ μπορεί να μετρηθεί σε διαφορετικές χρονικές στιγμές $t = t_k$ για $k = 0,1,2,\ldots,n$, έτσι ώστε $t_0 = 0$ and $t_n = T$, τότε το ολοκλήρωμα θα υπολογιστεί αριθμητικά από το δεδομένο σύνολο δεδομένων.

**Λύση:** Η συνάρτηση $y = f(x)$ είναι είτε αναλυτικά ορισμένη ή δίνεται υπό μορφή πίνακα. Η αριθμητική ολοκλήρωση βασίζεται στη χρήση μιας αριθμητικής παρεμβολής $y = P_n(x)$ προσαρμοσμένη στα δεδομένα σημεία δεδομένων και στην αναλυτική ολοκλήρωση του πολυωνύμου $P_n(x)$. Αυτός είναι ο ονομαζόμενος **Newton-Cotes** αλγόριθμος ολοκλήρωσης. Οι πιο σημαντικοί τύποι ολοκλήρωσης Newton-Cotes είναι οι κανόνες του τραπεζίου, του Simpson και του μέσου σημείου.

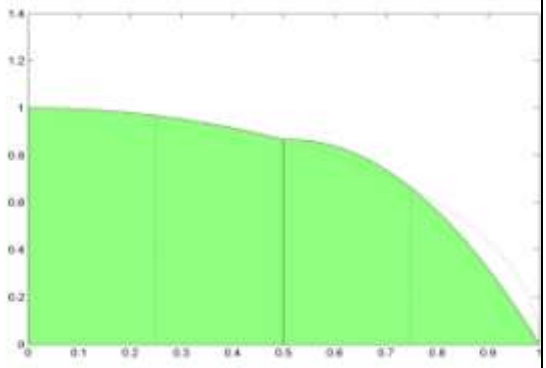



**Κανόνας Τραπεζίου:**

Μία γραμμική παρεμβολή μεταξύ δύο σημείων $(x_0,y_0)$ και $(x_1,y_1)$ προσεγγίζει το εμβαδό κάτω από την καμπύλη $y = f(x)$ με το εμβαδό του τραπεζοειδούς (trapezoidal):

$$\int_{x_0}^{x_1} f(x)dx \approx I_{trapezoidal}(f;x_0,x_1) = \frac{h}{2}(y_1 + y_0)$$

Ο κανόνας του τραπεζίου είναι δημοφιλής στην αριθμητική ολοκλήρωση επειδή είναι μια απλή μέθοδος. Παρόλο που δεν είναι και πολύ ακριβής, μπορεί να ελεγχθεί, διπλασιάζοντας των αριθμό των στοιχειωδών υποσυνόλων (τραπεζοειδών) που χρησιμοποιούνται για να προσεγγίσουμε το ολοκλήρωμα.

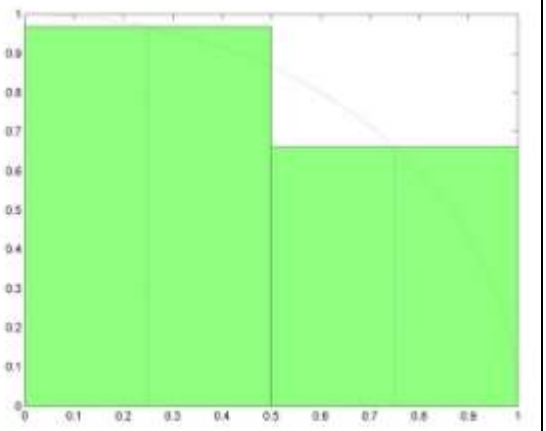


**Ο Κανόνας του Simpson:**

Μία δευτεροβάθμια παρεμβολή μεταξύ των σημείων $(x_0,y_0)$ $(x_1,y_1)$, και $(x_2,y_2)$ προσεγγίζει το εμβαδό κάτω από τη καμπύλη $y = f(x)$ με την περιοχή που βρίσκεται κάτω από τη συνάρτηση παρεμβολής:

$$\int_{x_0}^{x_2} f(x)dx \approx I_{Simpson}(f;x_0,x_2) = \frac{h}{3}(y_0 + 4y_1 + y_2)$$

Ο κανόνας του Simpson είναι δημοφιλής επειδή έχει υψηλή ακρίβεια της αριθμητικής ολοκλήρωσης σε σύγκριση με τον κανόνα του τραπεζίου.



**Κανόνας του μέσου σημείου:**

Μια σταθερή παρεμβολή του σημείου $(x_1,y_1)$, που βρίσκεται στο μέσο του διαστήματος μεταξύ $(x_0,y_0)$ και $(x_2,y_2)$, προσεγγίζει το εμβαδό κάτω από την καμπύλη $y = f(x)$ με το εμβαδό του ορθογωνίου κεντραρισμένο στο μέσο σημείο:

$$\int_{x_0}^{x_2} f(x)dx \approx I_{mid\text{-}point}(f;x_0,x_2) = 2h\,y_1$$

Ο κανόνας του μέσου σημείου είναι δημοφιλής στην αριθμητική ολοκλήρωση συναρτήσεων με ιδιομορφίες στο τέλος του

| | |
|---|---|
| | διαστήματος. Έχει την ίδια ακρίβεια με τον κανόνα του τραπεζίου και συχνά χρησιμοποιείται σε συνδυασμό με τον κανόνα του τραπεζίου για υπολογισμούς ολοκληρωμάτων κοντά σε ιδιομορφίες. |

**Παράδειγμα:**

```
h = 0.1; x0 = 0; x1 = x0+h; x2 = x0+2*h;
% the three data points are taken on [0,1] with equal step size
y0 = sqrt(1-x0^2); y1 = sqrt(1-x1^2); y2 = sqrt(1-x2^2);
yIexact = quad('sqrt(1-x.^2)',x0,x2); % 'exact answer' is computed by
MATLAB
yItrap = h*(y0+y1+y1+y2)/2; % trapezoidal rule for two subintervals
with h
yIsimp = h*(y0+4*y1+y2)/3;  % Simpson rule for two subintervals with h
yImid = 2*h*y1;  % mid-point rule for two subintervals with h
fprintf('Exact = %6.6f\nTrapezoidal = %6.6f\nSimpson = %6.6f\nMid-point
= %6.6f',yIexact,yItrap,yIsimp,yImid);
```

```
Exact = 0.198659
Trapezoidal = 0.198489
Simpson = 0.198658
Mid-point = 0.198997
```

## 2. Σύνθετοι κανόνες άθροισης:

Οι κανόνες άθροισης επεκτείνονται σε πολλαπλά διαστήματα όταν η συνάρτηση $y = f(x)$ αναπαρίσταται από $(n+1)$ σημεία δεδομένων με σταθερό πλάτος βήματος $h$. Ο σύνθετος κανόνας βρίσκεται αθροίζοντας τα εμβαδά όλων των $n$ μεμονωμένων εμβαδών.

- **Σύνθετος κανόνας τραπεζίου:**

$$\int_{x_0}^{x_n} f(x)dx \approx I_{trapezoidal}(f;x_0,x_1,\ldots,x_n) = \frac{h}{2}\left(y_0 + 2y_1 + 2y_2 + \ldots + 2y_{n-1} + y_n\right)$$

- **Σύνθετος κανόνας του Simpson:**

$$\int_{x_0}^{x_n} f(x)dx \approx I_{simpson}(f;x_0,x_1,\ldots,x_n) = \frac{h}{3}\left(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots + 2y_n + 4y_{n-1} + y_n\right)$$

- **Σύνθετος κανόνας μέσου σημείου:**

$$\int_{x_0}^{x_n} f(x)dx \approx I_{mid\text{-}point}(f;x_0,x_1,\ldots,x_n) = 2h\left(y_1 + y_3 + \ldots + y_{n-3} + y_{n-1}\right)$$

Για τους σύνθετους κανόνες Simpson και μέσου σημείου, το ολικό διάστημα ανάμεσα στο $x \in [x_0,x_n]$ πρέπει να διαιρεθεί σε άρτιο αριθμό υποδιαστημάτων.

## 3. Σφάλματα στην αριθμητική ολοκλήρωση:

Η αριθμητική ολοκλήρωση είναι μια πολύ πιο αξιόπιστη μέθοδος σε σύγκριση με την αριθμητική παραγώγιση. Τα σφάλματα στρογγυλοποίησης στον υπολογισμό αθροισμάτων στην αριθμητική ολοκλήρωση είναι πάντα σταθερά. Τα σφάλματα αποκοπής μπορούν να μειωθούν με τη χρήση κανόνων άθροισης για αριθμητική ολοκλήρωση που έχουν μεγαλύτερη ακρίβεια. Θεωρείστε τα ισαπέχοντα σημεία δεδομένων με σταθερό πλάτος βήματος $h = x_2 - x_1 = x_1 - x_0$. Η χρήση της μέθοδου αναπτύγματος του Taylor οδηγεί στα ακόλουθα τοπικά σφάλματα αποκοπής:

- **Κανόνας τραπεζίου:**

$$\int_{x_0}^{x_1} f(x)dx - I_{trapezoidal}(f;x_0,x_1) = -\frac{h^3}{12} f''(x), \quad x \in [x_0,x_1]$$

- **Κανόνας Simpson:**

$$\int_{x_0}^{x_{21}} f(x)dx - I_{Simpson}(f;x_0,x_2) = -\frac{h^5}{90} f''''(x), \quad x \in [x_0,x_2]$$

Παρατηρούμε ότι το σφάλμα αποκοπής του κανόνα Simpson είναι ανάλογο του $h^5$ σε σύγκριση με το σφάλμα αποκοπής του τραπεζίου που είναι της τάξης $h^3$, δηλαδή έχει την τάξη του $O(h^5)$. Το σφάλμα αποκοπής του Simpson είναι επίσης ανάλογο της παραγώγου τετάρτου βαθμού της συνάρτησης $f(x)$ σε ένα εσωτερικό σημείο $x$ του διαστήματος ολοκλήρωσης. Ο κανόνας του Simpson είναι ακριβής για πολυωνυμικές συναρτήσεις $f(x)$ βαθμού $m = 0,1,2,3$ (Γιατί;).

- **Ο κανόνας του μέσου σημείου:**

$$\int_{x_0}^{x_{21}} f(x)dx - I_{mid\text{-}point}(f;x_0,x_2) = \frac{h^3}{3} f''(x), \quad x \in [x_0,x_2]$$

Το σφάλμα αποκοπής του κανόνα του μέσου σημείου συμπεριφέρεται όπως του κανόνα του τραπεζίου.

Το ολικό σφάλμα αποκοπής για σύνθετους κανόνες ολοκλήρωσης βρίσκεται από την άθροιση των τοπικών σφαλμάτων αποκοπής και όλα τα τοπικά σφάλματα στρογγυλοποίησης:

- **Σύνθετος κανόνας τραπεζίου:**

$$e_{trapezoidal} = \left| \int_{x_0}^{x_n} f(x)dx - I_{trapezoidal}(f;x_0,x_1,\ldots,x_n) \right| < M_2 \frac{h^2}{12}(x_n - x_0) + eps\,(x_n - x_0),$$

όπου $M_2 = max\,|f''(x)|$. Το ολικό σφάλμα αποκοπής είναι ανάλογο του μήκους του διαστήματος ολοκλήρωσης και είναι της τάξεως του $O(h^2)$.

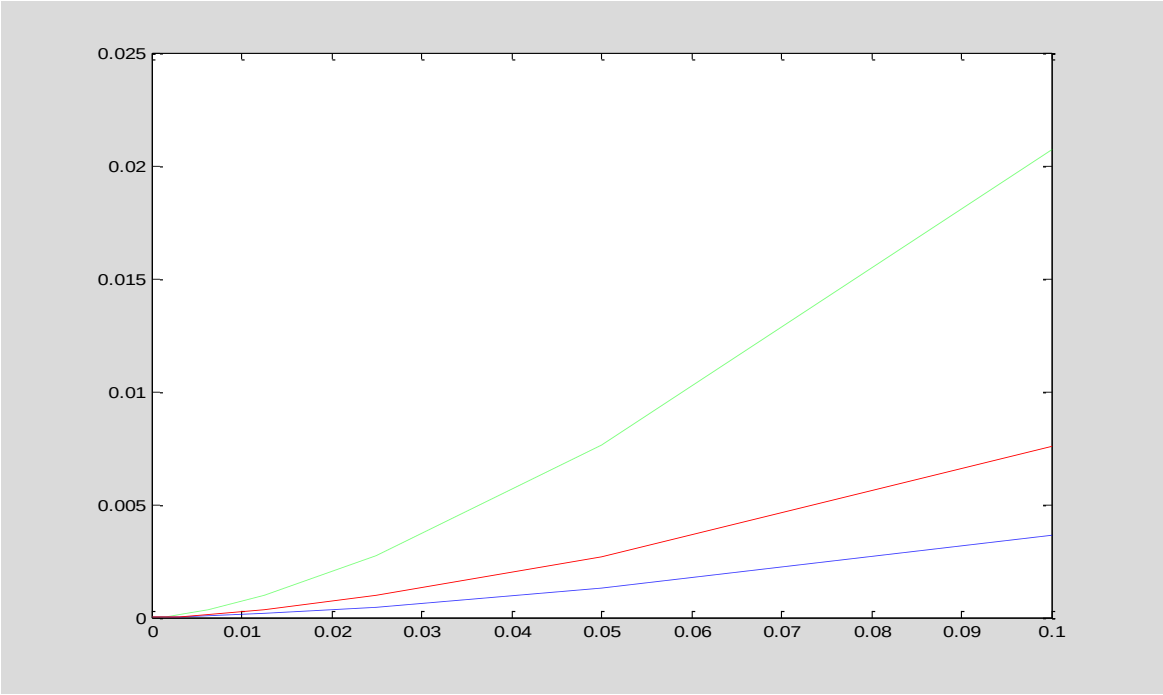
- **Σύνθετος κανόνας του Simpson:**

$$e_{Simpson} = \left| \int_{x_0}^{x_n} f(x)dx \; - I_{Simpson}(f;x_0,x_1,\ldots,x_n) \right| < M_4 \frac{h^4}{180}(x_n - x_0) + eps\,(x_n - x_0),$$

όπου $M_4 = max \mid f''''(x) \mid$. Το ολικό σφάλμα αποκοπής είναι ανάλογο του μήκους του διαστήματος ολοκλήρωσης και είναι της τάξης του $O(h^4)$.

**Παράδειγμα:** Ο παρακάτω κώδικας υπολογίζει τα ολοκληρώματα της συνάρτησης **sqrt(1-x$^2$)** με τις τρεις παραπάνω μεθόδους και το αντίστοιχο σφάλμα αποκοπής μαζί με το γράφημα του.

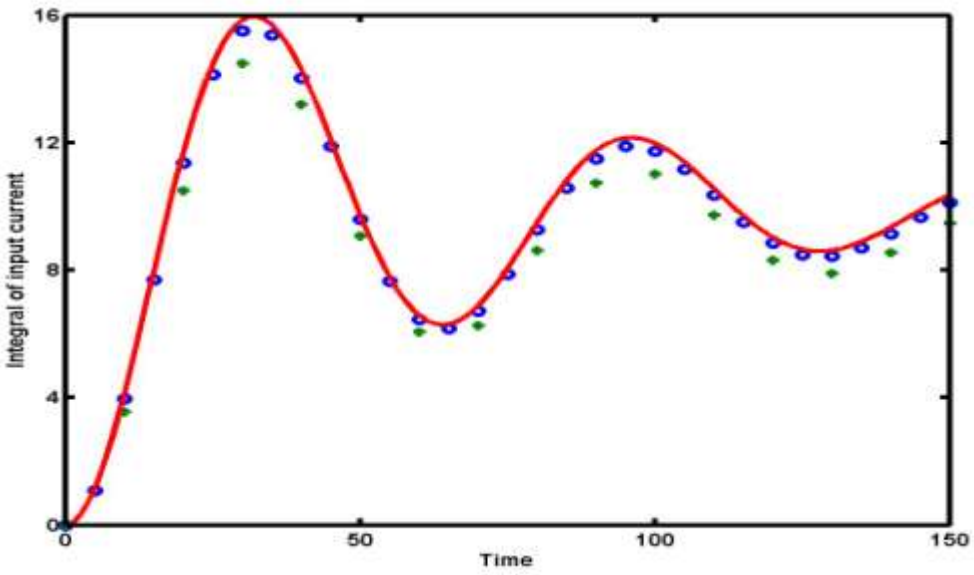
```
h = 0.1; k = 1;
while (h > 0.0000001)
      x = 0 : h : 1; y = sqrt(1.-x.^2); n = length(x)-1;
      yIexact = pi/4; % exact integral, 1/4 of area of a unit disk
  yItrap = h*(y(1)+2*sum(y(2:n))+y(n+1))/2; % composite trapezoidal rule
  yIsimp = h*(y(1)+4*sum(y(2:2:n))+2*sum(y(3:2:n-1))+y(n+1))/3;
  yImid = 2*h*sum(y(2:2:n));  % composite mid-point rule
  eItrap(k) = abs(yItrap-yIexact); eIsimp(k) = abs(yIsimp-yIexact);
  eImid(k) = abs(yImid-yIexact); hst(k) = h; h = h/2; k = k+1;
end,
plot(hst,eItrap,'g:',hst,eIsimp,'b--',hst,eImid,'r');
```




Αν το πλάτος βήματος ***h*** μεταξύ δύο σημείων γίνει μικρότερο, το σφάλμα προσέγγισης του κανόνα άθροισης μειώνεται. Μειώνεται γρηγορότερα με το κανόνα του Simpson και πιο αργά με τους κανόνες του τραπεζίου και του μέσου σημείου. Για παράδειγμα, εάν το ***h*** μειωθεί στο μισό, το ολικό σφάλμα προσέγγισης του κανόνα του Simpson μειώνεται κατά ένα παράγοντα του ***16***, ενώ το ολικό σφάλμα προσέγγισης των κανόνων του τραπεζίου και του μέσου σημείου μειώνονται μόνο κατά ένα παράγοντα του ***4***.

Εφόσον το σφάλμα στρογγυλοποίησης είναι φραγμένο από το διάστημα ολοκλήρωσης πολλαπλασιασμένο με την τιμή της παράμετρου που δηλώνει την ακρίβεια της μηχανής ***eps,*** το σφάλμα της αριθμητικής ολοκλήρωσης μπορεί να μειωθεί σε αυτό το σταθερό καθολικό αριθμό.

**Παράδειγμα:** Οι αριθμητικές προσεγγίσεις του ολοκληρώματος $\int_0^T dtI(t)$ ,όπου ***I(t)*** είναι το ρεύμα σε ένα δίκτυο αντίστασης-πυκνωτή, βρίσκονται με πλάτος βήματος ***h = 10*** (οι πράσινοι σταυροί) και πλάτος βήματος ***h = 5*** (οι μπλε τελείες), σε αντίθεση με το ακριβές ολοκλήρωμα (συμπαγής κόκκινη γραμμή). Το σφάλμα του σύνθετου κανόνα του τραπεζίου μειώνεται με μικρότερο πλάτος βήματος ***h*** (οι μπλε τελείες είναι πιο κοντά στην συμπαγή κόκκινη γραμμή). Η γραφική παράσταση δείχνει επίσης το καθολικό σφάλμα αποκοπής του ολοκληρώματος μεγαλώνει με το μήκος του διαστήματος ***T***.




## 4. Αριθμητική ολοκλήρωση με MATLAB:

- **quad:** υπολογίζει αριθμητικά το ολοκλήρωμα μιας συνάρτησης χρησιμοποιώντας την προσαρμοσμένη αριθμητική ολοκλήρωση του Simpson.
- **quadl:** υπολογίζει αριθμητικά το ολοκλήρωμα μιας συνάρτησης χρησιμοποιώντας την προσαρμοσμένη αριθμητική ολοκλήρωση του Lobatto.
- **dblquad:** υπολογίζει το διπλό ολοκλήρωμα μιας συνάρτησης δύο μεταβλητών σε ένα ορθογώνιο πεδίο

```
  % The function for numerical integration should be written as a
MATLAB M-file
  % The functon should accept a vector argument X and return a vector
result Y
      function [Y] = integrand(X)
            % this M-file sets up a function y = f(x) = sqrt(1 +
exp(x))
            % which is the integrand for the integral to be evaluated
by function "quad"
```

```
            Y = sqrt(1 + exp(X));

      % Compute the integral: I = int_0^2 sqrt(1 + exp(x)) dx
format long; I1 = quad(@integrand,0,2)
% the default tolerance is 10^(-6) for absolute error of numerical
integration
tolerance = 10^(-8); I2 = quad(@integrand,0,2,tolerance)
tolerance = 10^(-12); I3 = quad(@integrand,0,2,tolerance)
I4 = quadl(@integrand,0,2)
h = 0.0001; x = 0 : h : 2; y = feval(@integrand,x); n = length(y)-1;
I5 = h*(y(1)+2*sum(y(2:n))+y(n+1))/2 % composite trapezoidal rule
I6 = h*(y(1)+4*sum(y(2:2:n))+2*sum(y(3:2:n-1))+y(n+1))/3 % composite
Simpson rule
I7 = 2*h*sum(y(2:2:n))  % composite mid-point rule
I1 =        4.00699422404935
I2 =        4.00699422326706
I3 =  4.00699422325470
I4 =        4.00699422322700
I5 =        4.00699422402304
I6 =        4.00699422325470
I7 =    4.00699422171802

W1 = quad('sin(pi*x.^2)',0,1), W2 = quad(inline('sin(pi*x.^2)'),0,1)
% standard MATLAB functions can be used for numerical integration as
strings
W1 =  0.5049
W2 =        0.5049
```

## 5. Η ολοκλήρωση Romberg για υψηλότερου βαθμού τύπους ολοκλήρωσης Newton-Cotes:

Πιο ακριβείς τύποι ολοκλήρωσης με μικρότερο σφάλμα αποκοπής βρίσκονται παρεμβάλλοντας αρκετά σημεία δεδομένων με υψηλότερης τάξης πολυώνυμα παρεμβολής. Για παράδειγμα, το πολυώνυμο παρεμβολής τρίτης τάξεως $\boldsymbol{P_3(x)}$ μεταξύ τεσσάρων σημείων δεδομένων οδηγεί στον κανόνα 3/8 του Simpson:

$$\int_{x_0}^{x_3} f(x)dx \approx I_{Simpson\ 3/8}(f;x_0,x_3) = \frac{3h}{8}(y_0 + 3y_1 + 3y_2 + y_3)$$

Ενώ το πολυώνυμο παρεμβολής τέταρτης τάξεως $\boldsymbol{P_4(x)}$ μεταξύ των πέντε σημείων δεδομένων οδηγεί στον κανόνα του **Booles:**

$$\int_{x_0}^{x_4} f(x)dx \approx I_{Booles}(f;x_0,x_4) = \frac{2h}{45}(7y_0 + 32y_1 + 12y_2 + 32y_3 + 7y_4)$$

Οι υψηλής τάξης τύποι ολοκλήρωσης μπορούν να ανακτηθούν με τη χρήση του αλγόριθμου ολοκλήρωσης Romberg που βασίζεται στον αλγόριθμο παρεκβολής Richardson.

- **Αναδρομικοί τύποι ολοκλήρωσης:**

Ο σύνθετος κανόνας του τραπεζίου έχει ολικό σφάλμα αποκοπής της τάξεως $O(h^2)$. Συμβολίζουμε τον κανόνα του τραπεζίου για το ολοκλήρωμα $f(x)$ μεταξύ των $[x_0,x_n]$ ως $R_1(h)$. Υπολογίστε δύο αριθμητικές προσεγγίσεις του ολοκληρώματος με δύο πλάτη βήματος $h$ και $2h$:

$$\int_{x_0}^{x_n} f(x)\,dx = R_1(h) + \alpha h^2 ; \qquad \int_{x_0}^{x_n} f(x)\,dx = R_1(2h) + 4\alpha h^4;$$

Όπου $\alpha$ είναι άγνωστος συντελεστής του καθολικού σφάλματος αποκοπής. Ο αριθμός των τραπεζοειδών πρέπει να είναι άρτιος για να μπορέσουμε να υπολογίσουμε την αριθμητική προσέγγιση με διπλό πλάτος βήματος $(2h)$. Ακυρώνοντας το σφάλμα αποκοπής της τάξεως $O(h^2)$, ορίζουμε ένα νέο κανόνα ολοκλήρωσης για το ίδιο ολοκλήρωμα:

$$\int_{x_0}^{x_n} f(x)\,dx = \frac{4R_1(h) - R_1(2h)}{3} = R_2(h)$$

Ο νέος κανόνας ολοκλήρωσης $R_2(h)$ για το ίδιο ολοκλήρωμα είναι πιο ακριβής εφόσον το σφάλμα αποκοπής είναι της τάξεως του $O(h^4)$. Στη πραγματικότητα είναι ο σύνθετος κανόνας του Simpson καθώς μπορούμε να τον ελέγξουμε άμεσα. Αν το πλάτος βήματος είναι επαρκώς μικρό, ο σύνθετος κανόνας του Simpson μας δίνει μια πολύ καλύτερη αριθμητική προσέγγιση για το ολοκλήρωμα σε σύγκριση με το σύνθετο κανόνα του τραπεζίου.

Η διαδικασία μπορεί να συνεχιστεί για να βρούμε υψηλής τάξης τύπους ολοκλήρωσης $R_m(h)$ με σφάλμα αποκοπής $O(h^{2m})$. Ο αναδρομικός τύπος για τους τύπους ολοκλήρωσης του Romberg είναι:

$$R_{m+1}(h) = R_m(h) + \frac{R_m(h) - R_m(2h)}{4^k - 1}$$

- **Αριθμητικός αλγόριθμος**

Υπάρχουν δύο τροποποιήσεις του αλγόριθμου ολοκλήρωσης του Romberg: διπλασιάζοντας το πλάτος βήματος $h$ και μειώνοντας στο μισό το πλάτος βήματος. Η πρώτη τροποποίηση χρησιμοποιείται όταν έχουμε διαθέσιμο ένα περιορισμένο αριθμό τιμών δεδομένων. Η δεύτερη τροποποίηση είναι προτιμότερη εάν η συνάρτηση $y = f(x)$ είναι αναλυτικά διαθέσιμη και οι τιμές δεδομένων $(x_k,y_k)$ μπορούν να υπολογιστούν για οποιοδήποτε πλάτος βήματος $h$. Για να υπολογίσουμε τους κανόνες ολοκλήρωσης υψηλού βαθμού του ολοκληρώματος $f(x)$ μεταξύ $x_0 \le x \le x_n$ μέχρι την τάξη $m$, ο αριθμός $n$ πρέπει να ταιριάξει με το $m$ ως: $n = 2^{m-1}$. Σε αυτή τη περίπτωση, υπάρχει επαρκής αριθμός σημείων για να υπολογίσουμε τους κανόνες ολοκλήρωσης χαμηλότερης τάξης με μεγαλύτερα πλάτη βήματος $h, 2h, 4h, 8h, \ldots, (m-1)h$. Οι αριθμητικές προσεγγίσεις των ολοκληρωμάτων μπορούν να τακτοποιηθούν σε ένα πίνακα αναδρομικών τύπων ολοκλήρωσης ξεκινώντας με την πιο απλή προσέγγιση $R_1(h)$ (σύνθετος κανόνας τραπεζίου):

| ***πλάτος βήματος step size*** | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ |
|---|---|---|---|---|---|
| ***h*** | $R_1(h)$ | | | | |
| ***2h*** | $R_1(2h)$ | $R_2(h)$ | | | |
| ***4h*** | $R_1(4h)$ | $R_2(2h)$ | $R_3(h)$ | | |
| ***8h*** | $R_1(8h)$ | $R_2(4h)$ | $R_3(2h)$ | $R_4(h)$ | |
| ***16h*** | $R_1(16h)$ | $R_2(8h)$ | $R_3(4h)$ | $R_4(2h)$ | $R_5(h)$ |

Οι διαγώνιες καταχωρήσεις είναι τιμές υψηλότερης τάξης κανόνων ολοκλήρωσης για το ολοκλήρωμα της $f(x)$ μεταξύ των $x_0 \leq x \leq x_n$. Η προσέγγιση υψηλότερου βαθμού $R_k(h)$ έχει σφάλμα αποκοπής $O(h^{2k})$. Εάν το $h$ είναι μικρό, το σφάλμα αποκοπής μειώνεται γρηγορότερα με μεγαλύτερο $k$. Ωστόσο το σφάλμα στρογγυλοποίησης είναι σταθερό όταν το $k$ έχει μεγαλύτερη τιμή. Ως αποτέλεσμα δεν μπορεί να επιτευχθεί περαιτέρω αύξηση στην ακρίβεια της αριθμητικής ολοκλήρωσης όταν κάποιος αριθμός $m \geq M$. Ο αλγόριθμός μπορεί να προσδιοριστεί όταν το σχετικό σφάλμα είναι μικρότερο από την ανοχή σφάλματος.
.

**Παράδειγμα:** Η ακόλουθη γραφική παράσταση αντιπροσωπεύει τις αριθμητικές προσεγγίσεις $R_1(h)$ και $R_2(h)$ για το ολοκλήρωμα του ρεύματος $I(t)$ με πλάτος βήματος $h = 10$. Οι μπλε κύκλοι βρίσκονται δίπλα στο σύνθετο κανόνα του Simpson $R_2(h)$, οι πράσινοι σταυροί βρίσκονται από το σύνθετο κανόνα του τραπεζίου $R_1(h)$, και το ακριβές ολοκλήρωμα του $I(t)$ είναι σημειωμένο με μία κόκκινη συμπαγή καμπύλη. Ο σύνθετος κανόνας του Simpson $R_2(h)$ είναι εμφανώς πιο ακριβής από τον σύνθετο κανόνα του τραπεζίου $R_1(h)$

## 6. Βιβλιοθήκη Python scipy.intergrate

Το πακέτο **scipy.integrate** υποστηρίζει γνωστούς κανόνες αριθμητικής ολοκλήρωσης και λύσης ορισμένων διαφορικών εξισώσεων Η εντολή help παρέχει πληροφορίες για τους αλγορίθμους που η βιβλιοθήκη περιλαμβάνει:

```
>>> help(integrate)
 Methods for Integrating Functions given function object.

   quad          -- General purpose integration.
   dblquad       -- General purpose double integration.
   tplquad       -- General purpose triple integration.
   fixed_quad    -- Integrate func(x) using Gaussian quadrature of
order n.
   quadrature    -- Integrate with given tolerance using Gaussian
quadrature.
   romberg       -- Integrate func using Romberg integration.

 Methods for Integrating Functions given fixed samples.

   trapz         -- Use trapezoidal rule to compute integral from
samples.
   cumtrapz      -- Use trapezoidal rule to cumulatively compute
integral.
```

```
   simps          -- Use Simpson's rule to compute integral from
samples.
   romb           -- Use Romberg Integration to compute integral from
                     (2**k + 1) evenly-spaced samples.

   See the special module's orthogonal polynomials (special) for
Gaussian
      quadrature roots and weights for other weighting factors and
regions.

 Interface to numerical integrators of ODE systems.

   odeint         -- General integration of ordinary differential
equations.
   ode            -- Integrate ODE using VODE and ZVODE routines.
```

## 7. Υπολογισμοί στο περιβάλλον Python με χρήση της βιβλιοθήκης scipy.integrate

α) Να βρεθεί το ολοκλήρωμα της συνάρτησης bessel $jv(2.5, x)$ στο διάστημα [0,4.5].

Υπολογισμός του

$$I = \int_0^{4.5} J_{2.5}(x)dx.$$

Σημειώστε ότι το Python υποστηρίζει την δημιουργία ανωνύμων συναρτήσεων (δηλαδή συναρτήσεων που δεν δεσμεύονται από ένα όνομα) κατά την διάρκεια εκτέλεσης του προγράμματος, χρησιμοποιώντας το στοιχείο "lambda". Παράδειγμα,

```
>>> def f (x): return x**2
...
>>> print f(8)
64
>>>
>>> g = lambda x: x**2
>>>
>>> print g(8)
64
```

```
>>> from scipy import integrate, special
>>> from numpy import *
>>> result = integrate.quad(lambda x: special.jv(2.5,x), 0, 4.5)
>>> print result
(1.1178179380783249, 7.8663172481899801e-09)
>>> I = sqrt(2/pi)*(18.0/27*sqrt(2)*cos(4.5)-4.0/27*sqrt(2)*sin(4.5)+
    sqrt(2*pi)*special.fresnel(3/sqrt(pi))[0])
>>> print I
1.117817938088701
>>> print abs(result[0]-I)
1.03761443881e-11
```

β) Να υπολογιστεί η συνάρτηση

$$I = \sqrt{\frac{2}{\pi}}\left(\frac{18}{27}\sqrt{2}\cos(4.5) - \frac{4}{27}\sqrt{2}\sin(4.5) + \sqrt{2\pi}\,\mathrm{Si}\left(\frac{3}{\sqrt{\pi}}\right)\right),$$

Όπου

$$\mathrm{Si}(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right)dt$$

Είναι το Fresnel sine ολοκλήρωμα.

Η συνάρτηση **quad** δέχεται άπειρες τιμές ($\infty$) ως όρια ολοκλήρωσης και ορίζονται με το σύμβολο $\pm$ inf . Παρακάτω δίνουμε τις εντολές python που υπολογίζουν το ολοκληρώμα

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n}dt$$

για διάφορες τιμές των παραμέτρων n και x:

```
>>> from scipy.integrate import quad
>>> from scipy import special
>>> from numpy import arange, vectorize, inf, exp
>>> def integrand(t,n,x):
...     return exp(-x*t) / t**n
>>> def expint(n,x):
...     return quad(integrand, 1, Inf, args=(n, x))[0]
>>> vec_expint = vectorize(expint)
>>> vec_expint(3,arange(1.0,4.0,0.5))
array([ 0.1097,  0.0567,  0.0301,  0.0163,  0.0089,  0.0049])
>>> special.expn(3,arange(1.0,4.0,0.5))
array([ 0.1097,  0.0567,  0.0301,  0.0163,  0.0089,  0.0049])
>>> result = quad(lambda x: expint(3, x), 0, inf)
>>> print result
(0.33333333324560266, 2.8548934485373678e-09)
>>> I3 = 1.0/3.0
>>> print I3
0.333333333333
>>> print I3 - result[0]
8.77306560731e-11
```

Πολυδιάστατα ολοκληρώματα μπορούν να υπολογιστούν από την εφαρμογή της συνάρτησης **quad** σε κάθε διάσταση. Συναρτήσεις για τον υπολογισμό διπλών και τριπλών ολοκληρωμάτων είναι **dblquad** και **tplquad**. Ο κώδικας για το

$$I_n = \int_0^\infty \int_1^\infty \frac{e^{-xt}}{t^n}dtdx = \frac{1}{n}$$

είναι:

```
>>> from scipy.integrate import quad, dblquad
>>> def I(n):
...     return dblquad(lambda t, x: exp(-x*t)/t**n, 0, Inf, lambda x:
1, lambda x: Inf)
```

```
>>> print I(4)
(0.25000000000435768, 1.0518245707751597e-09)
>>> print I(3)
(0.33333333325010883, 2.8604069919261191e-09)
>>> print I(2)
(0.49999999999857514, 1.8855523253868967e-09)
```

## 8. Αναφορές


1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. https://sites.google.com/a/uni-konstanz.de/na09/Home/methods/fpns
3. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
4. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
5. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
6. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
7. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.

# ΚΕΦΑΛΑΙΟ 14: ΠΡΟΒΛΗΜΑΤΑ ΑΡΧΙΚΗΣ ΤΙΜΗΣ ΓΙΑ ΔΙΑΦΟΡΙΚΕΣ ΕΞΙΣΩΣΕΙΣ

## Περιεχόμενα



### 1. Ορισμός Προβλήματος

**Πρόβλημα:** Δίνεται μια πρώτου βαθμού διαφορική εξίσωση ορισμένη στο διάστημα $[0,T]$:

$$\boxed{\frac{dy}{dt} = f(t,y)}$$

Βρείτε μια αριθμητική προσέγγιση της λύσης $y = y(t)$ θεωρόντας γνωστή μια αρχική τιμή της:

$$\boxed{y(0) = y_0}$$

**Λύση:** Επιλέξτε μια διαμέριση του διαστήματος $[0,T]$ με σταθερό πλάτος βήματος $h$:

$$\boxed{t_0 = 0; \quad t_k = hk; \quad t_n = T}$$

όπου $n = T/h$ και $k = 1,2,\ldots,n$. Συμβολίστε την αριθμητική προσέγγιση της λύσης $y = y(t)$ στο σημείο $t = t_k$ με $y_k = y(t_k)$. Επειδή, γνωρίζουμε ακριβώς την αρχική τιμή $y_0 = y(0)$, οι αριθμητικές προσεγγίσεις της λύσης $y_k$ στα σημεία $t_k$ υπολογίζονται με κάποιο επαναληπτικό τύπο:

$$\boxed{y_k \rightarrow y_{k+1}}$$

### 2. Προς τα εμπρός, προς τα πίσω και βελτιωμένη μέθοδος Euler

- **Προς τα εμπρός μέθοδος Euler:**

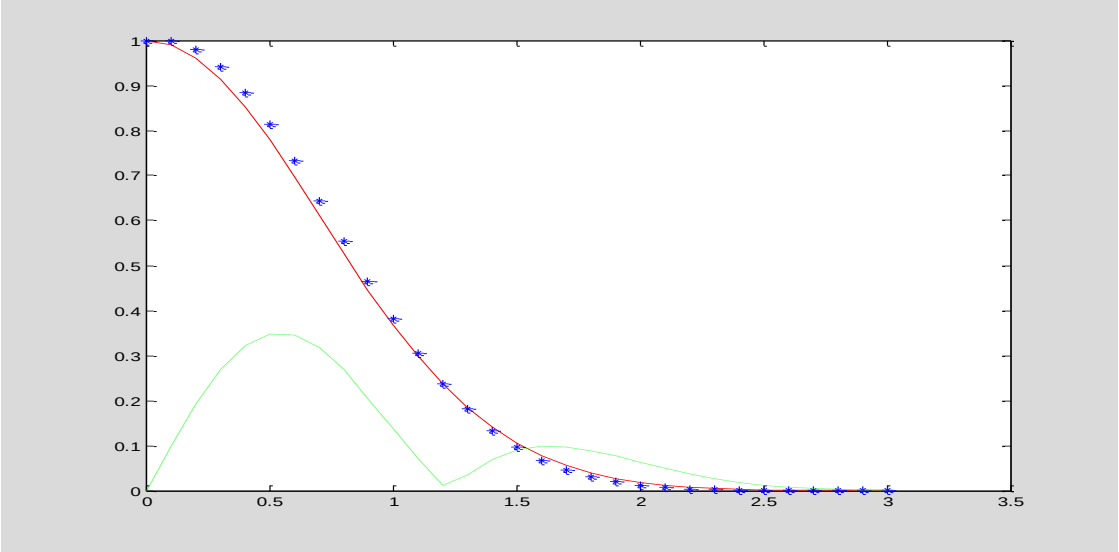
Αντικαθιστώντας την παράγωγο $y'(t)$ στην διαφορική εξίσωση στο $t = t_k$ με την προς τα εμπρός προσέγγιση με διαφορές $(y_{k+1}-y_k)/h$, λαμβάνουμε τον επαναληπτικό τύπο:

$$\boxed{y_{k+1} = y_k + h\, f(t_k,y_k)}$$

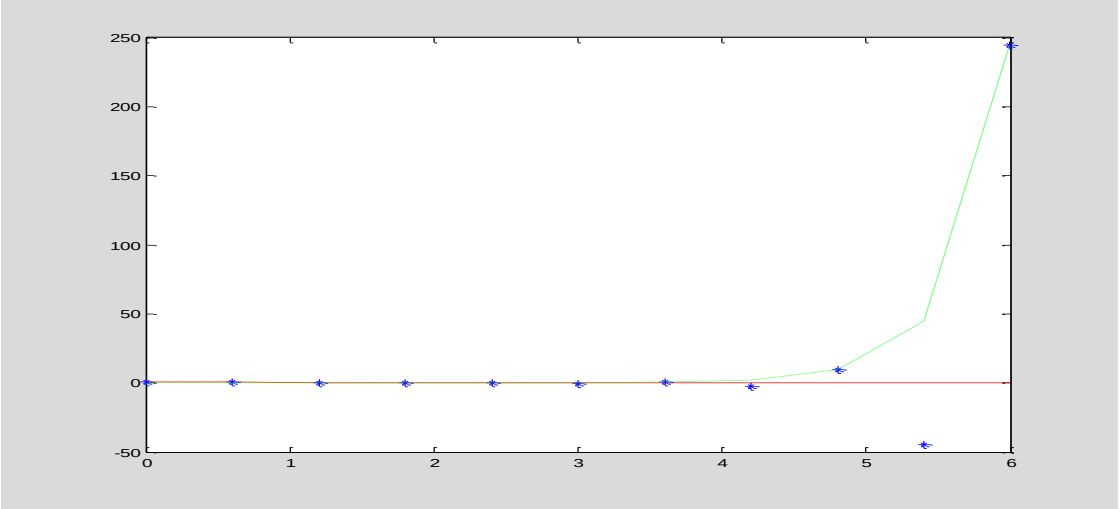
που υπολογίζει την λύση στα διακριτά σημεία $t_k$ που επιλέξαμε, υποθέτοντας ότι η αρχική τιμή $y_0$ είναι γνωστή.

**Παράδειγμα 1:** Ο παρακάτω κώδικας υπολογίζει την προσέγγιση της λύσης του προβλήματος $y'(t) = -2ty(t)$, $y(0) = 1$ και το τοπικό σφάλμα της (δηλαδή στα σημεία $t = t_k$) με την άμμεση μέθοδο του Euler για διαφορετικά βήματα $h$ και παράγει την γραφική τους παράσταση. Σημειώστε ότι η πραγματική λύση του προβλήματος είναι $exp(-t^2)$.

```
h = 0.1; y0 = 1; T = 3; n = T/h;
% differential equation: y' = -2*t*y, y(0) = 1
% exact solution: y(t) = exp(-t^2).
t(1) = 0; y(1) = y0; % the initial point in vectors (t,y)
      for k = 1 : n
                  t(k+1) = t(k) + h;
                  y(k+1) = y(k) + h*(-2*t(k)*y(k));
      end
yExact = exp(-t.^2); eLocal = 10*abs(y-yExact);
plot(t,yExact,'r',t,y,'*b',t,eLocal,':g');
```




```
h = 0.6; y0 = 1; T = 6; n = T/h; t(1) = 0; y(1) = y0;
      for k = 1 : n
                  t(k+1) = t(k) + h; y(k+1) = y(k) + h*(-2*t(k)*y(k));
      end
yExact = exp(-t.^2); eLocal = abs(y-yExact);
plot(t,yExact,'r',t,y,'*b',t,eLocal,':g');
```




Η προς τα εμπρός μέθοδος Euler είναι η πιο απλή μέθοδος για να προσεγγίσουμε τη λύση ενός προβλήματος ΔΕ με αρχικές συνθήκες. Η μέθοδος είναι χαμηλής ακρίβειας και η σύγκλιση της εμφανίζει "αστάθεια", δηλαδή δεν συγκλίνει για ορισμένες τιμές του βήματος ***h***. Η μέθοδος αυτή ανήκει στην κατηγορία των **άμεσων** (explicit) μεθόδων και η σύγκλιση της εξαρτάται από το μέγεθος του βήματος.

- **Προς τα πίσω μέθοδος Euler:**

Αντικαθιστώντας την παράγωγο ***y'(t)*** στην διαφορική εξίσωση στο ***t*** = $t_k$ με την προς τα πίσω προσέγγιση με διαφορές: $(y_k-y_{k-1})/h$ οδηγούμαστε σε ένα μη- γραμμικό επαναληπτικό τύπο:

$$\boxed{y_{k+1} = y_k + h\, f(t_{k+1},y_{k+1})}$$

για την εύρεση της λύσης στα προεπιλεγμένα σημεία $t_k$.

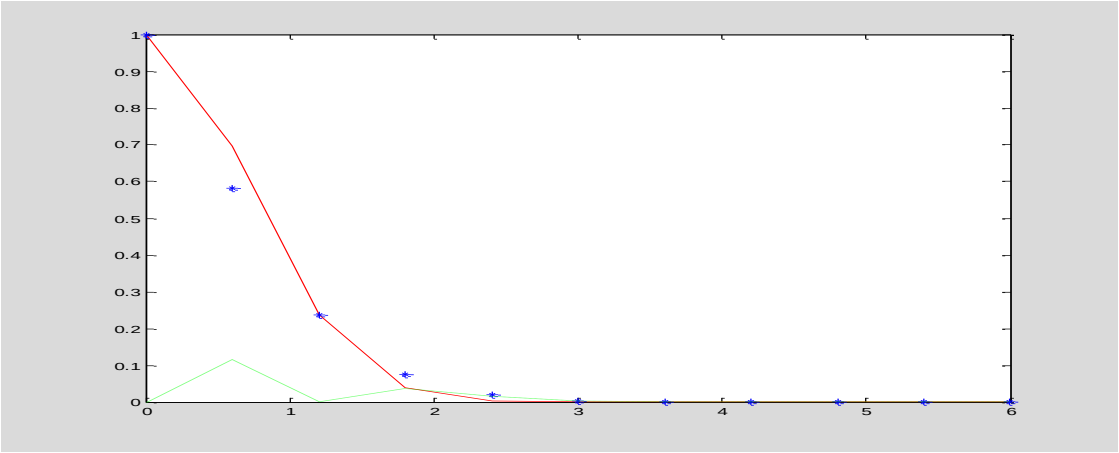
Η ακρίβεια της προς τα πίσω μεθόδου Euler είναι η ίδια με αυτήν της προς τα εμπρός μεθόδου Euler αλλά αυτή η μέθοδος είναι "σταθερή", δηλαδή συγκλίνει χωρίς περιορισμούς στις τιμές του βήματος ***h***. Εφόσον, η δεξιά πλευρά του επαναληπτικού τύπου περιλαμβάνει την άγνωστη τιμή $y_{k+1}$, δηλαδή μπορεί να είναι μη-γραμμική, θα πρέπει να εφαρμοστεί ένας αλγόριθμος εύρεσης ριζών για να προσδιοριστεί η τιμή του $y_{k+1}$ από τον επαναληπτικό τύπο. Η μέθοδος αυτή ανήκει στην κατηγορία των **έμμεσων** (implicit) μεθόδων και η σύγκλιση της **δεν** εξαρτάται από το μέγεθος του βήματος.

**Παράδειγμα 2:** Ο παρακάτω κώδικας υπολογίζει την προσέγγιση της λύσης του προβλήματος $y'(t) = -2ty(t)$, $y(0) = 1$ και το τοπικό σφάλμα της με την έμμεση μέθοδο του Euler και παράγει την γραφική τους παράσταση.

.

```
h = 0.6; y0 = 1; T = 6; n = T/h; t(1) = 0; y(1) = y0;
      for k = 1 : n
                  t(k+1) = t(k) + h;
                  % solve the implicit equation: y(k+1) = y(k) + h*(-
2*t(k+1)*y(k+1))
                  y(k+1) = y(k)/(1 + 2*h*t(k+1));
end, yExact = exp(-t.^2); eLocal = abs(y-yExact);
```

```
plot(t,yExact,'r',t,y,'*b',t,eLocal,':g');
```




- **Βελτιωμένη μέθοδος Euler (Heun):**

Ολοκληρώνοντας τα δύο μέρη της εξίσωσης $y' = f(t,y)$ στο διάστημα $[t_k, t_{k+1}]$ λαμβάνουμε:

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(t,y(t))\,dt$$

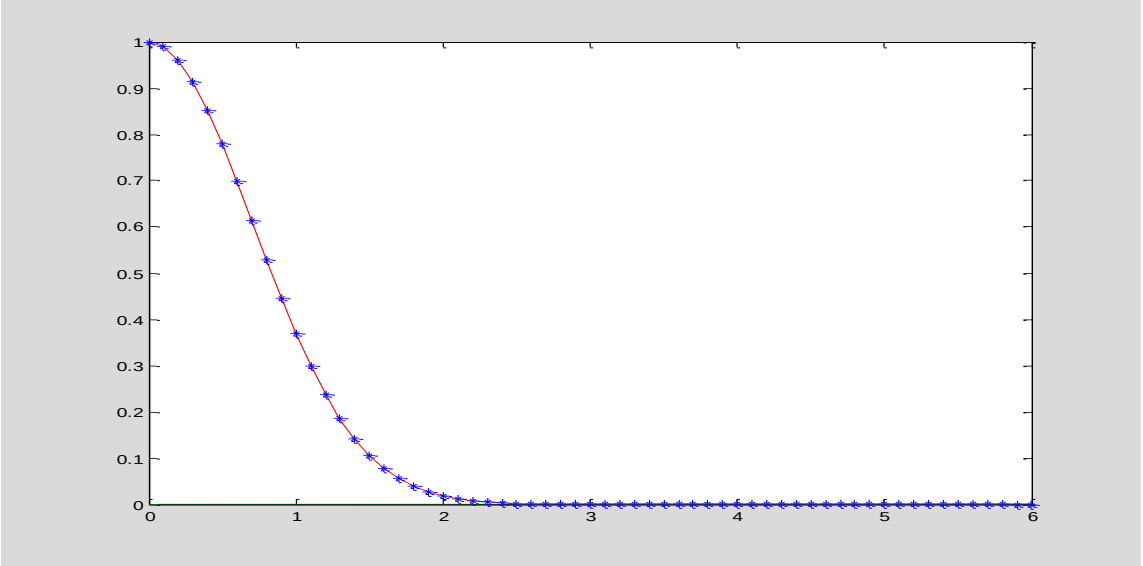
Προσεγγίζοντας το ολοκλήρωμα με τον κανόνα του τραπεζίου λαμβάνουμε τη βελτιωμένη μέθοδο Euler :

$$y_{k+1} = y_k + \frac{h}{2}\left[\, f(t_k,y_k) + f(t_{k+1},y_{k+1}) \,\right]$$

Η βελτιωμένη μέθοδος Euler είναι πιο ακριβής από τις προς τα εμπρός και προς τα πίσω μεθόδους Euler και είναι "σταθερή" χωρίς περιορισμούς στο μήκος του βήματος. Σημειώστε, ότι η εξίσωση είναι γενικώς μη γραμμική, αφού η δεξιά πλευρά του τύπου εξαρτάται από την τιμή $y_{k+1}$. Επομένως, θα πρέπει να χρησιμοποιηθεί ένας αλγόριθμος εύρεσης ριζών για να προσδιοριστεί η τιμή του $y_{k+1}$ σε κάθε βήμα της επαναληπτικής μεθόδου. Η μέθοδος ανήκει στην κατηγορία των **έμμεσων** (implicit) μεθόδων.

**Παράδειγμα 3:** Ο παρακάτω κώδικας υπολογίζει την προσέγγιση της λύσης του προβλήματος $y'(t) = -2ty(t)$, $y(0) = 1$ και το τοπικό σφάλμα της με την βελτιωμένη μέθοδο του Euler και παράγει την γραφική τους παράσταση. Θυμηθείτε ότι η πραγματική λύση του προβλήματος είναι $exp(-t^2)$.

```
h = 0.1; y0 = 1; T = 6; n = T/h; t(1) = 0; y(1) = y0;
      for k = 1 : n
                  t(k+1) = t(k) + h;
                  % solve the implicit equation:
       % y(k+1) = y(k) + h*(-t(k)*y(k)-t(k+1)*y(k+1))
                  y(k+1) = y(k)*(1-h*t(k))/(1 + h*t(k+1));
  end
yExact = exp(-t.^2); eLocal = abs(y-yExact);
plot(t,yExact,'r',t,y,'*b',t,eLocal,':g');
```




Η βελτιωμένη μέθοδο Euler μπορεί να μετατραπεί σε άμεση μέθοδο αν πρώτα προβλέψουμε την τιμή $y_{n+1}$ στη χρονική στιγμή $t_{n+1}$ με την προς τα εμπρός μέθοδο Euler:

$$z_{n+1} = y_n + h\, f(t_n, y_n)$$

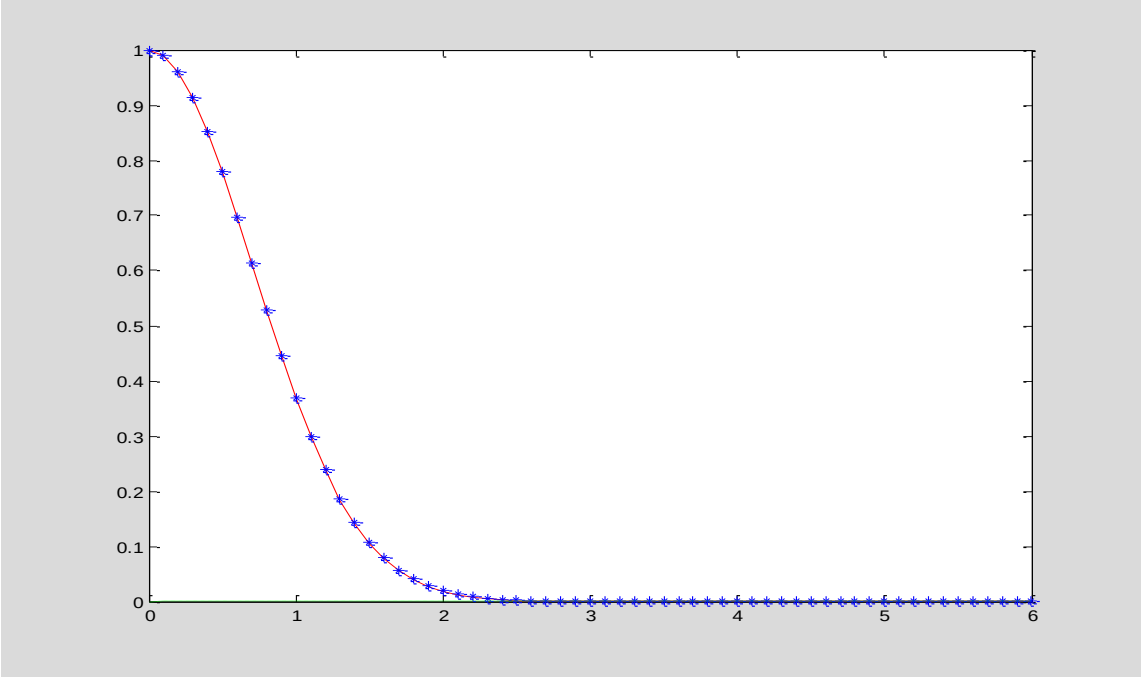
όπου $z_{n+1}$ είναι η προβλεπόμενη τιμή του $y_{n+1}$ και στη συνέχεια διορθώσουμε αυτή τη τιμή με τη βελτιωμένη μέθοδο Euler:

$$y_{k+1} = y_k + \frac{h}{2}\left[\, f(t_k, y_k) + f(t_{k+1}, z_{k+1}) \,\right]$$

Αυτοί οι δύο τύποι πρόβλεψης-διόρθωσης είναι άμεσοι δηλαδή είναι γραμμικοί. Η μέθοδος είναι γνωστή σαν την μέθοδο Heun.

**Παράδειγμα 4:** Ο παρακάτω κώδικας υπολογίζει την προσεγγιστική λύση του προβλήματος $y'(t) = -2ty(t)$, $y(0) = 1$ και το τοπικό σφάλμα της με την μέθοδο του Heun και παράγει την γραφική τους παράσταση.

```
h = 0.1; y0 = 1; T = 6; n = T/h; t(1) = 0; y(1) = y0;
      for k = 1 : n
                  t(k+1) = t(k) + h;
       % prediction:
       z = y(k) + h*(-2*t(k)*y(k));
       % correction:
       y(k+1) = y(k) + h*(-t(k)*y(k)-t(k+1)*z);
   end
yExact = exp(-t.^2); eLocal = abs(y-yExact);
plot(t,yExact,'r',t,y,'*b',t,eLocal,':g');
```




## 3. Ποιο προχωρημένες μέθοδοι για αριθμητικές λύσεις διαφορικών εξισώσεων

- Μέθοδος Runge-Kutta ενός βήματος
- Μέθοδοι Adams-Bashforth πολλών βημάτων

## 4. Επιλυτές MATLAB για ΔΕ μιας μεταβλητής

- **ode23:** επιλυτής Runge-Kutta δεύτερης και τρίτης τάξεως με μεταβλητό πλάτος βήματος
- **ode45:** Επιλυτής Runge-Kutta τέταρτης και πέμπτης τάξεως με μεταβλητό πλάτος βήματος
- **ode113:** Επιλυτής Adams-Bashforth-Moulton μεταβλητής τάξεως από $1^{η}$ εώς $13^{η}$
- **ode15s:** δύσκαμπτος επιλυτής μεταβλητής τάξεως από $1^{η}$ εώς $5^{η}$ με βάση την προς τα πίσω παραγώγιση
- **ode23s, ode23t, ode23tb:** δύσκαμπτος επιλυτής τρίτης και τέταρτης τάξεως με βάση τον κανόνα του τραπεζίου

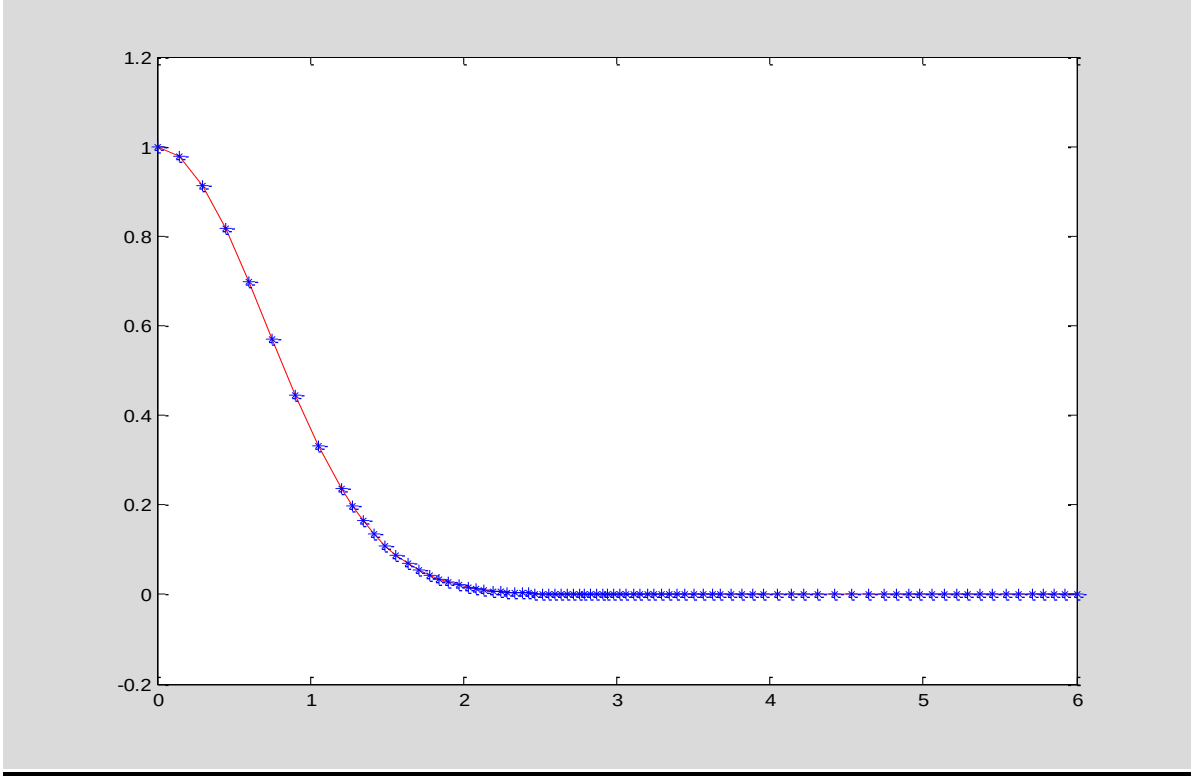
**[t,y] = ODEsolver(odefun,tspan,y0)**

**ODEsolver** = όνομα του επιλυτή ΔΕ
**Odefun** = το όνομα ενός αρχείου ΟΔ (συνάρτηση αρχείου MATLAB)
**tspan** = ο αρχικός και τελικός χρόνος **[0, T]** ή η διαμέριση του χρονικού διαστήματος **[0 : h : T]**
**y0** = η αρχική τιμή της διαφορικής εξίσωσης
**[t,y]** = διανύσματα στήλης για τη λύση ***y = y(t)*** στα χρονικά σημεία μεταξύ **0** και **T**

**Παράδειγμα 5:** Ο παρακάτω κώδικας εφαρμόζει επιλυτές MatLab για την εύρεση της λύσης του προβλήματος του παραδείγματος 1.

```
function [y1] = ODEexample(t,y)
```

```
      % ODE file (M-file function) for right-hand-side of the
differential equation
      % y1 if the derivative of the solution y = y(t) (the right-hand-
side function)
y1 = -2*t*y;

y0 = 1; tspan = [0,6];
[t,y] = ode45(@ODEexample,tspan,y0);
t' % values of time instance where the solution is computed
yExact = exp(-t.^2); plot(t,yExact,'r',t,y,'*b');

  Columns 1 through 11
         0    0.1500    0.3000    0.4500    0.6000    0.7500    0.9000
1.0500    1.2000    1.2725    1.3449
  Columns 12 through 22
    1.4174    1.4898    1.5623    1.6347    1.7072    1.7796    1.8423
1.9050    1.9676    2.0303    2.0830
  Columns 23 through 33
    2.1357    2.1883    2.2410    2.2877    2.3345    2.3812    2.4279
2.4705    2.5131    2.5557    2.5983
  Columns 34 through 44
    2.6378    2.6773    2.7168    2.7563    2.7933    2.8303    2.8673
2.9043    2.9444    2.9846    3.0247
  Columns 45 through 55
    3.0648    3.1094    3.1539    3.1985    3.2431    3.2935    3.3439
3.3943    3.4448    3.5033    3.5619
  Columns 56 through 66
    3.6205    3.6790    3.7493    3.8196    3.8899    3.9602    4.0478
4.1355    4.2231    4.3108    4.4202
  Columns 67 through 77
    4.5295    4.6389    4.7483    4.8259    4.9036    4.9812    5.0589
5.1365    5.2142    5.2918    5.3695
  Columns 78 through 85
    5.4552    5.5410    5.6267    5.7124    5.7843    5.8562    5.9281
6.0000
```

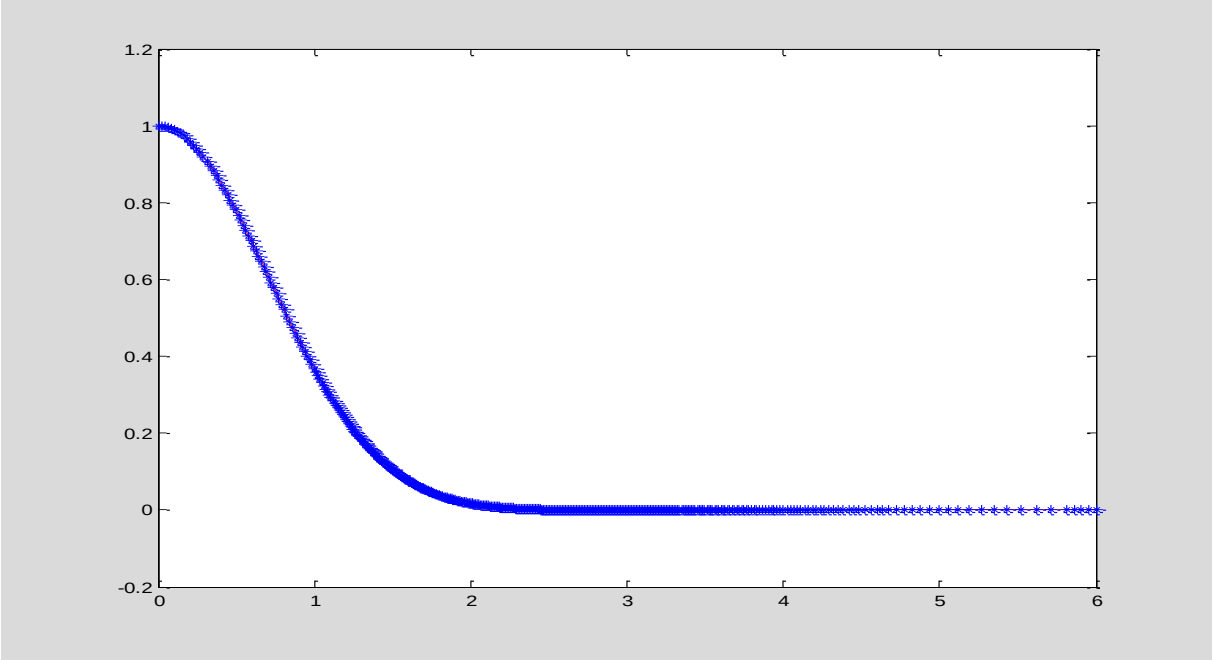



**options = odeset('optionname',optionvalue);**
**[t,y] = ODEsolver(odefun,tspan,y0,options);**
**RelTol** – επιτρεπτή τιμή του σχετικού σφάλματος [θετικό αριθμός π.χ. 1e-3} ]
**AbsTol** – επιτρεπτή τιμή του απόλυτου σφάλματος [ θετικός αριθμός π.χ. 1e-6} ]
**NormControl** - σφάλμα ελέγχου σχετικό με τη νόρμα της λύσης [ on | {off} ]
**MaxStep** – ανώτερο όριο του πλάτους βήματος [θετικός αριθμός]

Επιπλέον παράμετροι της διαφορικής εξίσωσης μπορούν να οριστούν στη συνάρτηση του αρχείου MATLAB (M-file).

**Παράδειγμα 6:** Ο παρακάτω κώδικας εφαρμόζει επιλυτές MatLab για την εύρεση της λύσης του προβλήματος του παραδείγματος 1.

```
function [y1] = ODEexample(t,y,par)
      % ODE file (M-file function) for right-hand-side of the differential equation
      % y1  = the derivative of the solution y = y(t) (the right-hand-side function)
      % par = additional parameter of the differential equation
y1 = -par*t*y;

y0 = 1; tspan = [0,6]; par = 2;
opt = odeset('AbsTol',10^(-10),'RelTol',10^(-8));
[t,y] = ode45(@ODEexample,tspan,y0,opt,par);
yExact = exp(-t.^2); plot(t,yExact,'r',t,y,'*b');
```




## 5. Το πρόβλημα ΔΕ με οριακές συνθήκες (Boundary Value Problem - BVP)

Έστω ***y = y(x)*** είναι η λύση της ΔΕ δεύτερης τάξεως:

$$y'' = g(x,y,y'), \quad alpha < x < beta$$

με συνοριακές τιμές: ***y(alpha) = a, y(beta) = b.***

## 6. Επιλυτής MATLAB BVP:

Η MATLAB έχει την ακόλουθη γενική εντολή για την ενεργοποίηση επιλυτών για την λύση ΔΕ με οριακές συνθήκες:

***[x,y,total] = bvp (alpha,beta,a,b,tol,n,q,g)***

Οι επιπλέον παράμετροι ορίζονται ως εξής:

**tol** = ανώτατο όριο στη διαφορά μεταξύ αριθμητικών προσεγγίσεων
***n*** = αριθμός εσωτερικών σημείων λύσης (m >= 1)
***q*** = μέγιστος αριθμός επαναλήψεων (q >= 1)
***g*** = συμβολοσειρά που περιέχει το όνομα της συνάρτησης g που ορίζει ο χρήστης – παρέχεται από μια συνάρτηση M-file στην μορφή:

***y2 = function g (x,y,y1)***

***total***= επιτρεπτός συνολικός αριθμός επαναλήψεων
***[x,y]***= διάνυσμα στήλη της λύσης ***y = y(x)*** με ***(n+2)*** σημεία δεδομένων, συμπεριλαμβανομένων των οριακών σημείων.

## 7. Υπολογισμοί λύσης ΔΕ στο περιβάλλον Python με χρήση της βιβλιοθήκης SciPy

**Παράδειγμα**: Να υπολογιστεί η λύση της ΔΕ:

$$\frac{d^2w}{dz^2} - zw = 0$$

με αρχικές συνθήκες

$$w(0) = \frac{1}{\sqrt[3]{\frac{2}{3}}\Gamma(\frac{2}{3})},\ \frac{dw(0)}{dz}|_{z=0} = -\frac{1}{\sqrt[3]{\frac{1}{3}}\Gamma(\frac{1}{3})}.$$

Γνωρίζουμε ότι η λύση του παραπάνω προβλήματος είναι η Airy συνάρτηση

$$w = Ai(z),$$

που ορίζεται από την Python συνάρτηση **special.airy**.

Για την εύρεση της αριθμητικής λύσης του παραπάνω προβλήματος, πρώτα μετασχηματίζουμε την εξίσωση σε σύστημα πρώτου βαθμού διαφορικών εξισώσεων, εισάγοντας νέες μεταβλητές $y = [\frac{dw}{dz}, w]$ και $t = z$

$$\frac{dy}{dt} = \begin{bmatrix} ty_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} 0 & t \\ 1 & 0 \end{bmatrix}\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0 & t \\ 1 & 0 \end{bmatrix} y = Ay.$$

$$I = \int_0^{4.5} J_{2.5}(x)dx.$$

$$I = \sqrt{\frac{2}{\pi}}\left(\frac{18}{27}\sqrt{2}\cos(4.5) - \frac{4}{27}\sqrt{2}\sin(4.5) + \sqrt{2\pi}\mathrm{Si}\left(\frac{3}{\sqrt{\pi}}\right)\right),$$

$$\mathrm{Si}(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right)dt.$$

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n}dt.$$

Επομένως, το αρχικό πρόβλημα μετασχηματίζεται στο

$$\frac{dy}{dt} = \begin{bmatrix} ty_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} 0 & t \\ 1 & 0 \end{bmatrix}\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0 & t \\ 1 & 0 \end{bmatrix} y = Ay,$$

όπου,

$$f(y,t) = A(t)y$$

Μπορούμε να αποδείξουμε ότι η αναλυτική λύση του είναι:

$$y(t) = \exp\left(\int_0^t A(\tau)\,d\tau\right) y(0).$$

Ο παρακάτω κώδικας δείχνει την χρήση της Python συνάρτησης `odeint` συμπεριλαμβανομένης και της `Dfun` option που επιτρέπει στο χρήστη να ορίσει το `gradient` (ως προς $y$) της συνάρτησης $f(y,t)$.

```
>>> from scipy.integrate import odeint
>>> from scipy.special import gamma, airy
>>> from numpy import arange
>>> y1_0 = 1.0/3**(2.0/3.0)/gamma(2.0/3.0)
>>> y0_0 = -1.0/3**(1.0/3.0)/gamma(1.0/3.0)
>>> y0 = [y0_0, y1_0]
>>> def func(y, t):
...     return [t*y[1],y[0]]
>>> def gradient(y,t):
...     return [[0,t],[1,0]]
>>> x = arange(0,4.0, 0.01)
>>> t = x
>>> ychk = airy(x)[0]
>>> y = odeint(func, y0, t)
>>> y2 = odeint(func, y0, t, Dfun=gradient)
>>> print ychk[:36:6]
[ 0.355028  0.339511  0.324068  0.308763  0.293658  0.278806]
>>> print y[:36:6,1]
[ 0.355028  0.339511  0.324067  0.308763  0.293658  0.278806]
>>> print y2[:36:6,1]
[ 0.355028  0.339511  0.324067  0.308763  0.293658  0.278806]

ή σε μορφή προγράμματος (script)

from scipy.integrate import odeint
from scipy.special import gamma, airy
from numpy import arange
y1_0 = 1.0/3**(2.0/3.0)/gamma(2.0/3.0)
y0_0 = -1.0/3**(1.0/3.0)/gamma(1.0/3.0)
y0 = [y0_0, y1_0]
def func(y, t):
    return [t*y[1],y[0]]
def gradient(y,t):
    return [[0,t],[1,0]]
x = arange(0,4.0,0.01)
t = x
ychk = airy(x)[0]
y = odeint(func, y0, t)
y2 = odeint(func, y0, t, Dfun=gradient)
print ychk[:36:6]
print y[:36:6,1]
print y2[:36:6,1]

παράγει τα αποτελέσματα
0.35502805  0.33951139  0.32406751  0.30876307  0.29365818  0.27880648]
[ 0.35502805  0.33951138  0.32406749  0.30876306  0.29365817
0.27880648]
```

```
[ 0.35502805  0.33951138  0.32406749  0.30876306  0.29365817
0.27880648]
```

## 8. Αναφορές


1. http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes
2. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
3. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
4. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
5. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
6. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.

# ΚΕΦΑΛΑΙΟ 15: ΤΟ ΑΡΙΘΜΗΤΙΚΟ ΣΥΣΤΗΜΑ ΚΙΝΗΤΗΣ ΥΠΟΔΙΑΣΤΟΛΗΣ ΚΑΙ ΣΦΑΛΜΑΤΑ ΣΤΡΟΓΓΥΛΟΠΟΙΗΣΗΣ

## Περιεχόμενα



Στα μαθηματικά δουλεύουμε με όλο το σύνολο πραγματικών/μιγαδικών αριθμών. Στους επιστημονικούς υπολογισμούς χρησιμοποιούμε ένα πολύ μικρό υποσύνολο των, που ορίζεται από ένα αριθμητικό σύστημα που αναφέρεται σαν *σύστημα κινητής υποδιαστολής*. Σαν συνέπεια, πολλοί σωστοί μαθηματικοί ισχυρισμοί δεν ισχύουν σε *πεπερασμένης ακρίβειας αριθμητικής* που ορίζει το αριθμητικό σύστημα κινητής υποδιαστολής που θα μελετήσουμε. Στις σημειώσεις αυτές θα μελετήσουμε τις "παγίδες" του επιστημονικού υπολογισμού σε *πεπερασμένη ακρίβεια αριθμητική*. Αλλά πρώτα θα δούμε πως ορίζεται το αριθμητικό σύστημα κινητής υποδιαστολής. Το κεφάλαιο αυτό είναι προσαρμογή στα Ελληνικά της σχετικής ύλης του μαθήματος Numerical Algorithms with Applications του Πανεπιστημίου Konstanz της Γερμανίας που βρίσκεται στην ιστοσελίδα https://sites.google.com/a/uni-konstanz.de/na09/Home/methods/fpns και αποτελεί κατά την γνώμη του υπευθύνου των σημειώσεων αυτών μια διαφορετική και εφαρμοσμένη παρουσίαση του αντικειμένου.

## 1. Αριθμητικό σύστημα σταθερής υποδιαστολής

Όποιος έχει εργαστεί με υπολογιστές χειρός (calculators) γνωρίζει το σύστημα παράστασης αριθμών κινητής υποδιαστολής. Οι χαρακτήρες της οθόνης του calculator 2.597 -03 παριστούν τον αριθμό 2.597x10-3 ή 2.596e-3. Το βασικό πλεονέκτημα των του συστήματος κινητής υποδιαστολής είναι ότι μπορούν να παραστούν αριθμούς πολύ μεγαλύτερων τιμών για την συγκεκριμένη «ακρίβεια» της υποτιθέμενης μηχανής. Για παράδειγμα, αν η μηχανή μπορεί να παραστήσει αριθμούς με 6 ψηφία, με 5 ψηφία μετά το «δεκαδικό σημείο», τότε ο μεγαλύτερος αριθμός που μπορεί να παρασταθεί στο σύστημα *σταθερής υποδιαστολής* (*fixed* point system) είναι $9.99999 \approx 10$ και ο μικρότερος $0.00001 \approx 10^{-5}$. Αν αντιστοιχήσουμε δύο από τα έξη ψηφία να παραστούν την δύναμη του δέκα, τότε μπορούμε να παραστούμε αριθμούς με τιμές από $10^{-99}$ μέχρι $10^{99}$.

Το μειονέκτημα της παράστασης αυτής είναι ότι μας δίνει ακρίβεια 4 δεκαδικών ψηφίων. Άρα το αριθμητικό σύστημα κινητής υποδιαστολής χρειάζεται για να παριστά περισσότερους αριθμούς σε σχέση με το απλούστερο σύστημα *σταθερής* υποδιαστολής. Στο σύστημα σταθερής υποδιαστολής, οι αριθμοί που μπορούν να παρασταθούν είναι

$$y = \pm m \times \Delta$$

Όπου *m* είναι θετικός ακέραιος στο διάστημα [*0,m_max*] και *delta* είναι η απόσταση μεταξύ διαδοχικών αριθμών. Στην περίπτωση αυτή ο μέγιστος αριθμός που μπορεί να παρασταθεί είναι $y_{max} = m_{\max} \times \Delta$.

## 2. <u>Αριθμητικό σύστημα κινητής υποδιαστολής</u>

Το σύστημα κινητής υποδιαστολής *F* είναι ένα υποσύνολο των πραγματικών αριθμών όπου τα στοιχεία του ορίζονται από το τύπο:

$$y = \pm(m / \beta^t) \times \beta^e$$

Το σύστημα *F* χαρακτηρίζεται από 4 ακεραίους: *βάση* (beta), *ακρίβεια* (t), και το πεδίο του εκθέτη (emin και emax). Η *mantissa (συντελεστής αναπαράστασης αριθμού) m* και ο *exponent (εκθέτης) e* είναι ακέραιοι που ικανοποιούν:

$$0 \le m \le \beta^t - 1;\ e_{\min} \le e \le e_{\max}$$

Ένας εναλλακτικός τρόπος για την παράσταση του *y* είναι να αναπτύξουμε την mantissa *m* ως προς την βάση *beta*:

$$y = \pm\beta^e \frac{d_1\beta^{t-1} + d_2\beta^{t-2} + ... + d_t\beta^0}{\beta^t} = \pm\beta^e(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + ... + \frac{d_t}{\beta^t}) = f \bullet \beta^e$$

όπου κάθε ψηφίο ικανοποιεί την συνθήκη:

$$0 \le d_i \le \beta - 1$$

Η παράσταση αυτή δικαιολογεί γιατί *f* ονομάζεται *κλάσμα* (*fraction)*. Το *f* περιλαμβάνει την «ακρίβεια t», δηλαδή το πλήθος των ψηφίων του κλάσματος των αριθμών που

αναφέρονται ως τα «σημαντικά» ψηφία του αριθμού (significant figures) και ο εκθέτης, ή κλίμακα του αριθμού, ή τάξη μεγέθους του.

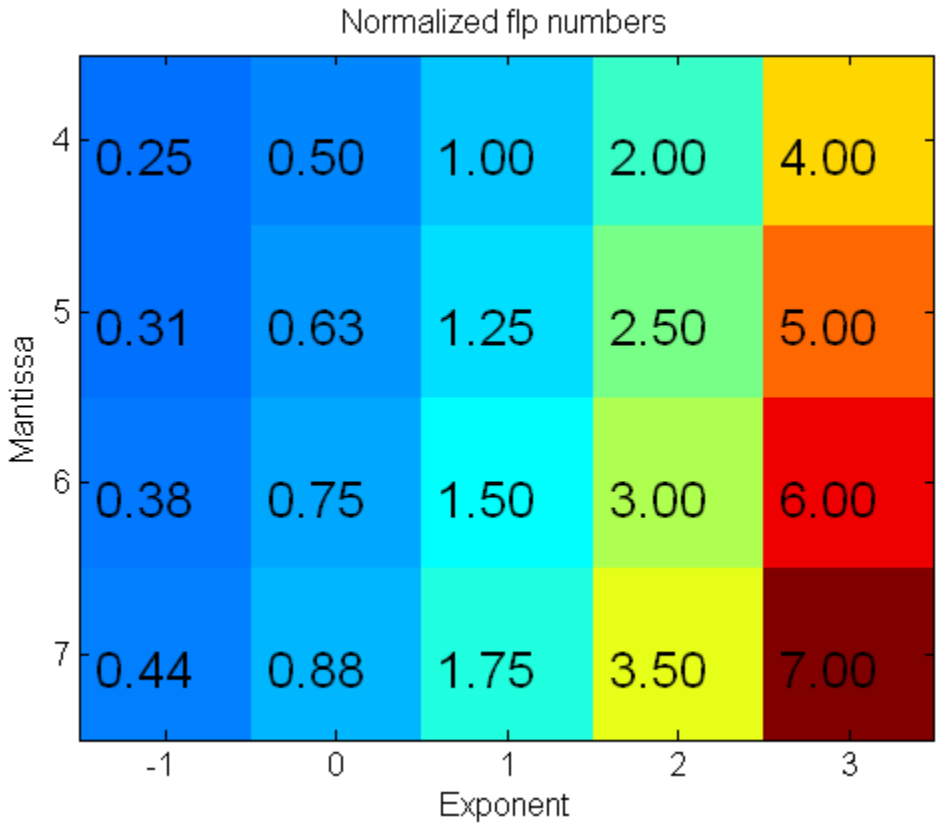
Όμως, η παραπάνω παράσταση δεν είναι μοναδική, αφού κάθε πραγματικός αριθμός ορίζεται από δύο ακεραίους: *m* και *e* (επομένως, κανείς μπορεί να επιλέξει το ένα και να βρει τον άλλο που ικανοποιεί τον τύπο). Για την μοναδική παράσταση κάθε *y* στο σύστημα *ομαλοποιείται* (*normalized )* έτσι ώστε το οδηγό στοιχείο να είναι μη-μηδενικό, δηλαδή υποθέτουμε ότι

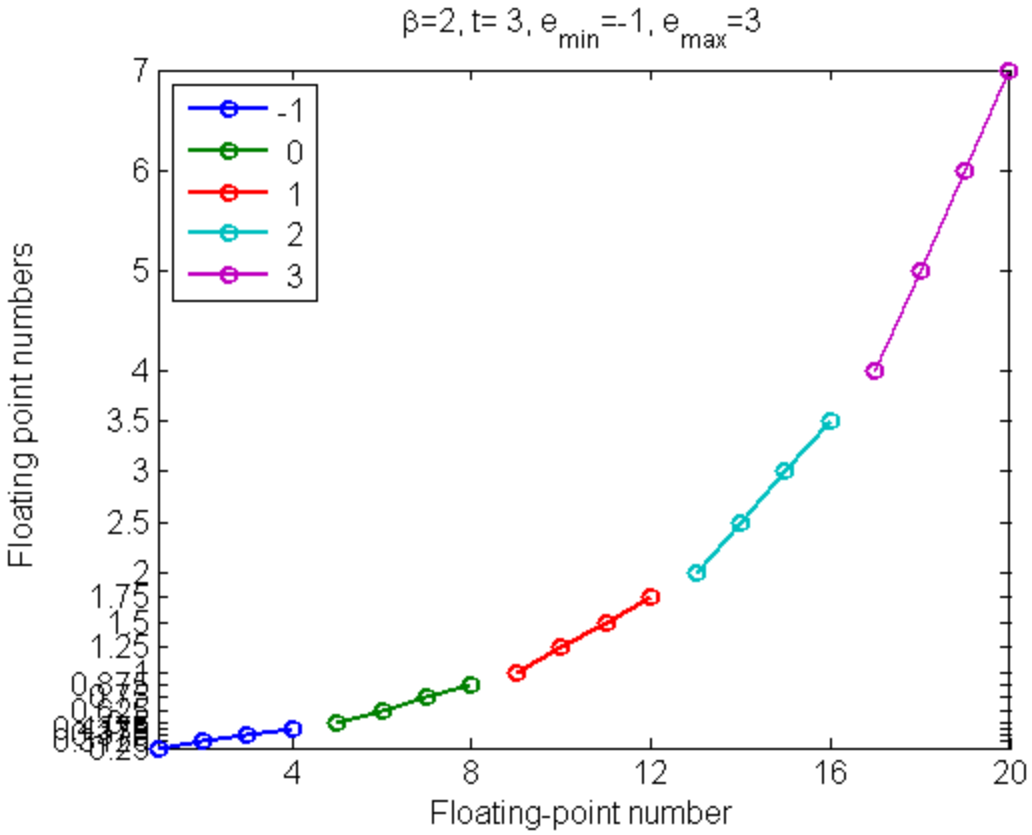
$$d_1 \neq 0 \text{ ή } \beta^{-1} \leq f < 1 \text{ ή } m \geq \beta^{t-1} \ (y \neq 0)$$

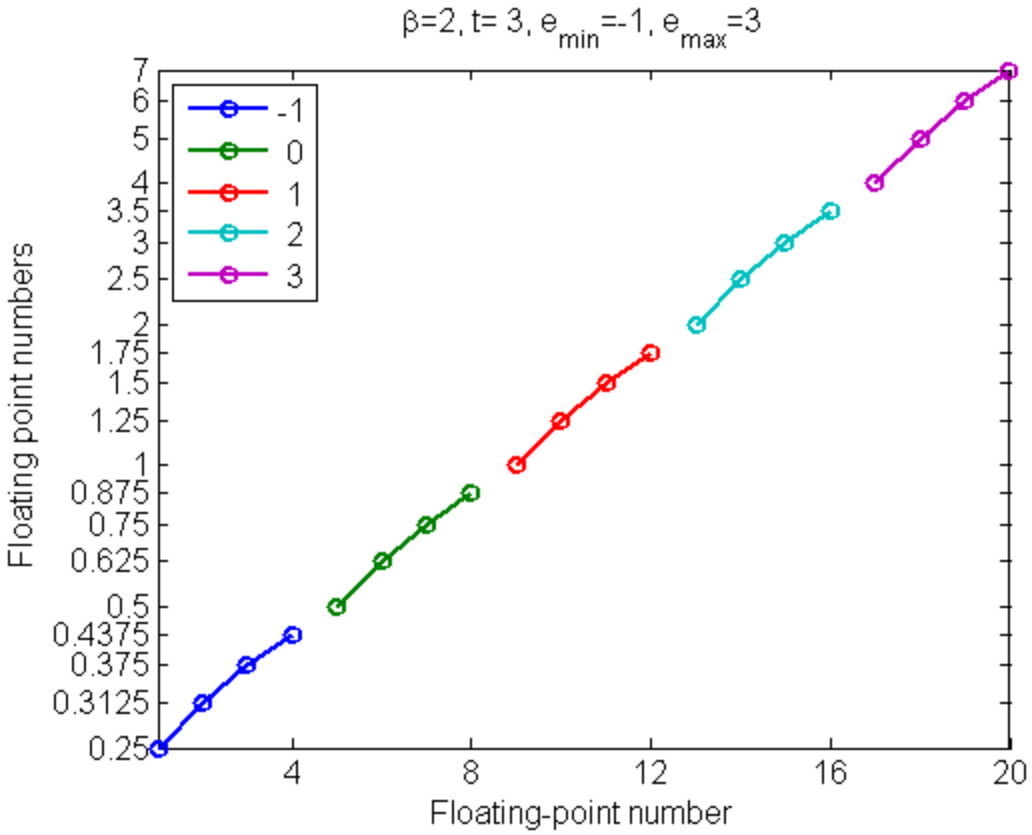
Έτσι για κάθε δυαδικό σύστημα, *beta* = 2, το οδηγό ψηφίο των ομαλοποιημένων αριθμών είναι πάντα 1.

Ας δούμε ένα παράδειγμα κινητής υποδιαστολής:

```
beta = 2, t = 3, emin = -1, emax = 3.
```



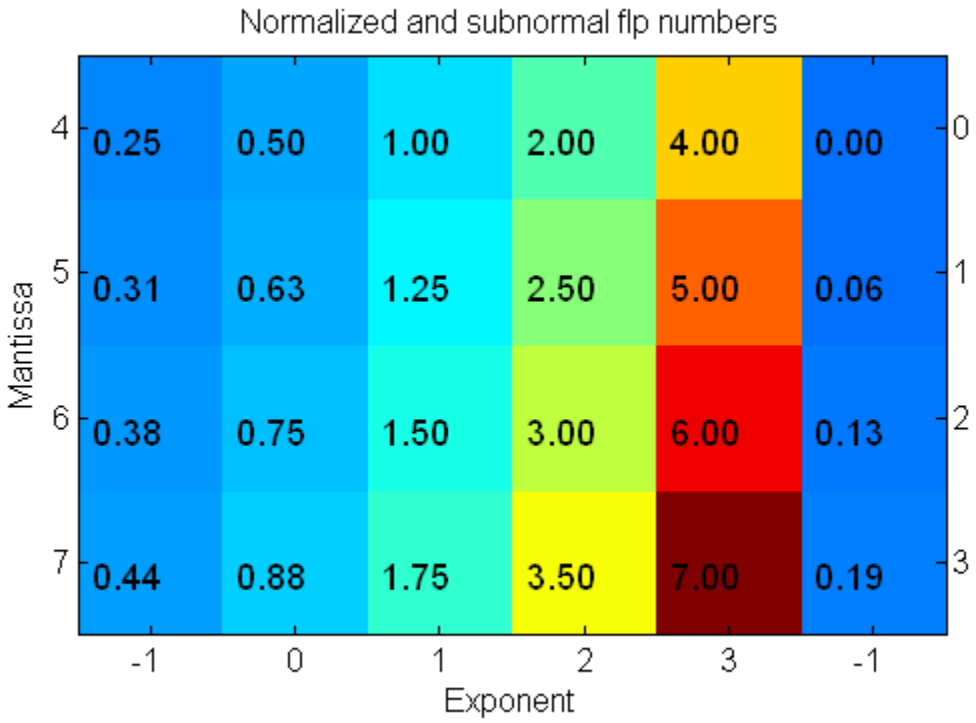

β=2, t= 3, e_min=-1, e_max=3
Floating point numbers
Floating-point number
-1
0
1
2
3
4
8
12
16
20

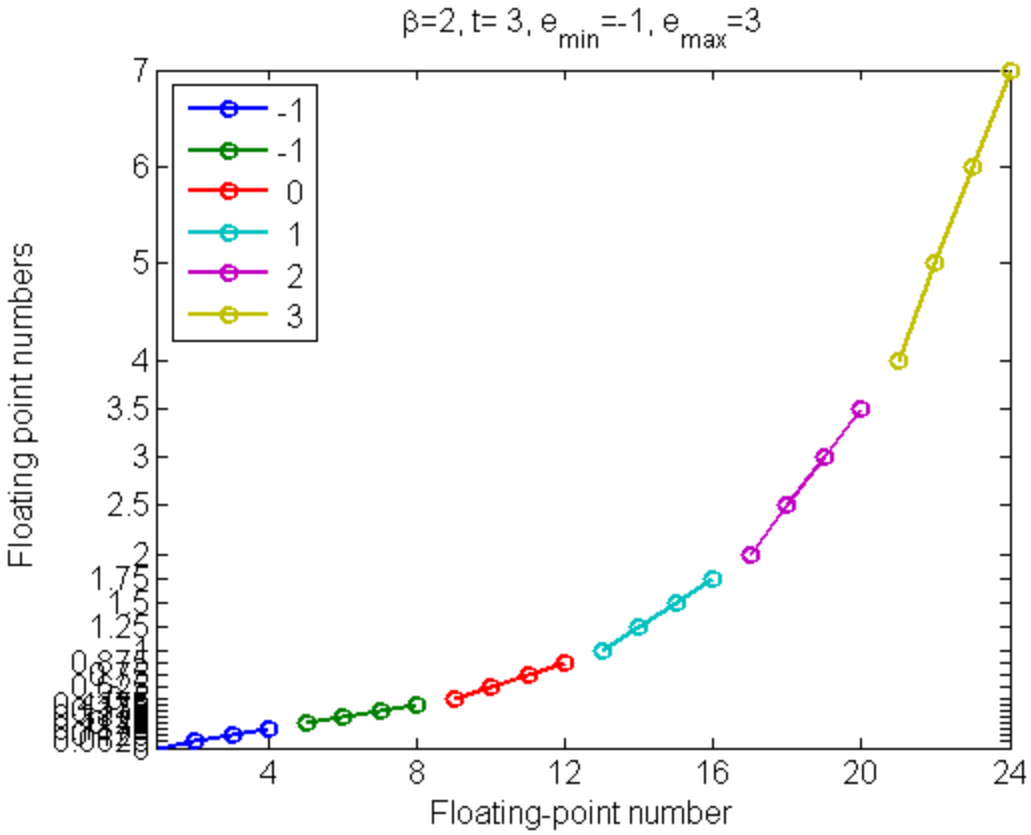

β=2, t= 3, e_min=-1, e_max=3
Floating point numbers
Floating-point number
-1
0
1
2
3
0.25
0.3125
0.375
0.4375
0.5
0.625
0.75
0.875
1
1.25
1.5
1.75
2
2.5
3
3.5
4
5
6
7
4
8
12
16
20

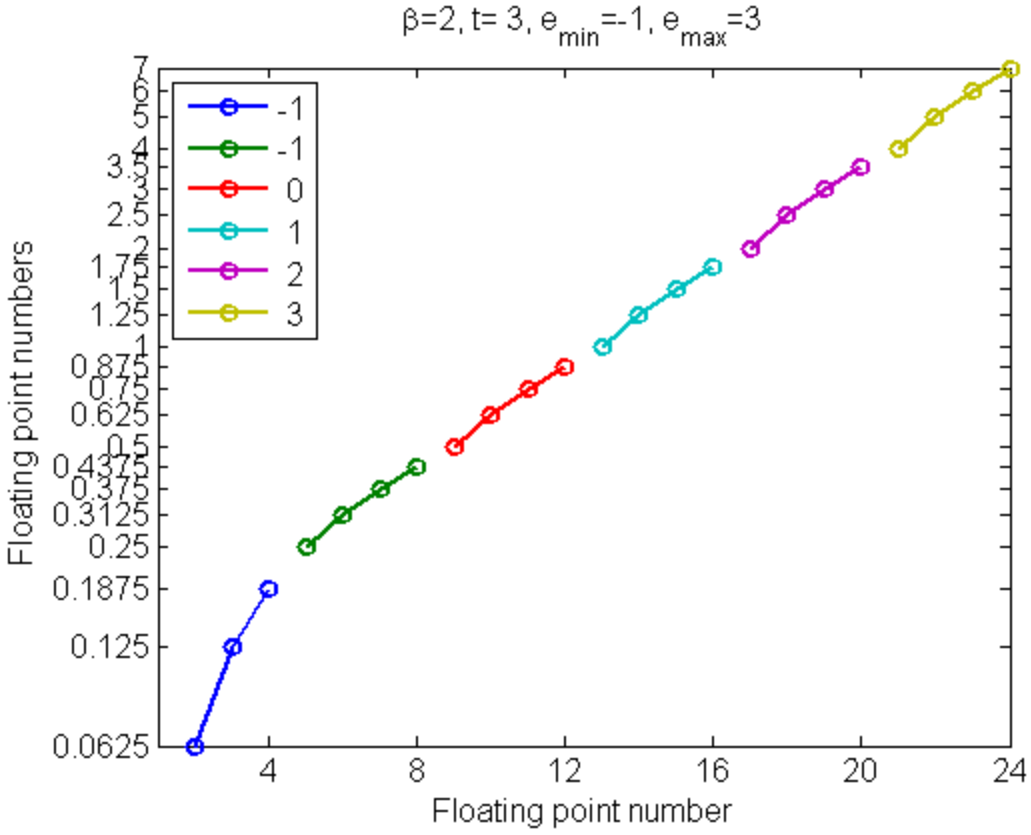
Τα παραπάνω γραφήματα δείχνουν ότι οι αριθμοί κινητής υποδιαστολής είναι ανομοιόμορφα κατανεμημένοι. Δηλαδή, η απόσταση μεταξύ αριθμών δεν είναι σταθερή αλλά μεταβάλλεται κατά 2 για κάθε δύναμη του 2. Η απόσταση μεταξύ αριθμών χαρακτηρίζεται ως ο αριθμό μηχανής - *machine epsilon* (*eps*), δηλαδή, η απόσταση μεταξύ του 1.0 και του επόμενου αριθμού στο σύστημα κινητής υποδιαστολής. Το *eps* στο παραπάνω παράδειγμα αριθμητικού συστήματος είναι 0.25.

Ο μικρότερος θετικός ομαλοποιημένος αριθμός στο παραπάνω παράδειγμα είναι 0.25. Σημειώστε ότι είναι 4 περισσότερες φορές μακρύτερα από το "0" σε σχέση με τον επόμενο μεγαλύτερο αριθμό. Το μεγάλο κενό οφείλεται στο σύστημα ομαλοποίησης – το πρώτο μη-μηδενικό στοιχείο πρέπει να είναι 1. Το σύστημα $F$ μπορεί να επεκταθεί αν συμπεριλάβουμε αριθμούς που έχουν ελάχιστο εκθέτη και πρώτο ψηφίο μηδέν.

Οι νέοι αριθμοί για το παραπάνω παράδειγμα βρίσκονται στην 6η στήλη του πίνακα :




β=2, t= 3, e min =-1, e max =3
Floating point numbers
Floating-point number
-1
-1
0
1
2
3


β=2, t= 3, e min =-1, e max =3
Floating point numbers
Floating point number
-1
-1
0
1
2
3

## 3. IEEE πρότυπο

Το IEEE πρότυπο ορίζει ένα δυαδικό σύστημα αριθμών κινητής υποδιαστολής. Το πρότυπο ορίζει δύο τύπους αριθμών κινητής υποδιαστολής:

| Type | Size | Mantissa (t) | Exponent | emin | emax |
|---|---|---|---|---|---|
| ingle | 32 bits | 23+1 bits | 8 bits | -125 | +128 |
| Double | 64 bits | 52+1 bits | 11 bits | -1021 | +1024 |

Η δυαδική παράσταση κάθε αριθμού στο 32-bit IEEE πρότυπο αποθηκεύεται σε 32 bit λέξη όπου το πρώτο bit διατίθεται για το πρόσημο του κλάσματος, τα επόμενα 8-bits για τον εκθέτη, και τα υπόλοιπα 23 bits για το κλάσμα. Σε αυτό το σύστημα μπορούν να παρασταθούν αριθμοί περίπου από $10^{-38}$ έως $10^{38}$. Η ακρίβεια είναι περίπου 7 «σημαντικά» δεκαδικά ψηφία. Σημειώστε, ότι το οδηγό δυαδικό ψηφίο δεν καταχωρείται αφού είναι γνωστό ότι είναι 1 για τους ομαλοποιημένους αριθμούς και οι αριθμοί αυτού του συστήματος αναφέρονται ως **αριθμοί απλής ακρίβειας**. Στην περίπτωση του 64-bit IEEE προτύπου, δηλαδή **διπλής ακριβείας πρότυπο,** διατίθενται ένα bit για το πρόσημο, 11-bits για τον εκθέτη, και 52-bits για το κλάσμα. Αυτό το πρότυπο μπορεί να παραστήσει αριθμούς από $10^{-307}$ έως $10^{307}$ με ακρίβεια περίπου 15 σημαντικών δεκαδικών ψηφίων.

### Άσκηση 1: Ερωτήματα

- Πόσοι ομαλοποιημένοι αριθμοί υπάρχουν στο IEEE πρότυπο μονής και διπλής ακριβείας;
- Ποια είναι η τιμή του αριθμού μηχανής *eps*;
- Ποιοι είναι οι μικρότεροι ομαλοποιημένοι αριθμοί; Να συγκριθούν οι απαντήσεις σας με το output της συνάρτησης `realmin`.
- Ποιοι είναι οι μεγαλύτεροι ομαλοποιημένοι αριθμοί; Να συγκριθούν οι απαντήσεις σας με το output της συνάρτησης `realmax`.

### Άσκηση 2: Εξαιρέσεις

Να υπολογισθούν οι παρακάτω εκφράσεις στην MATLAB. Είναι οι τιμές αυτές που περιμένατε; (Σημειώστε ότι τα αποτελέσματα δεν συμφωνούν με το IEEE πρότυπο.)

1. 1^inf
2. 2^inf
3. exp(inf), exp(-inf)
4. sign(NaN), sign(-NaN)
5. NaN^0
6. inf^0
7. 1^NaN
8. log(inf), log(-inf), log(0).

## 4. Μονάδα σφάλματος (Unit roundoff)

Μπορούμε να αποδείξουμε ότι κάθε πραγματικός αριθμός που βρίσκετε στο εύρος του συνόλου *F* μπορεί να προσεγγισθεί από ένα αριθμό του *F* με *σχετικό σφάλμα* όχι μεγαλύτερο από το μισό της μηχανής epsilon, u = eps/2. Η τιμή λέγεται *unit roundoff*.

**Άσκηση 3:** Στα περισσότερα αριθμητικά συστήματα κινητής υποδιαστολής μια γρήγορη προσέγγιση του unit roundoff μπορεί να υπολογισθεί από τον αριθμό `3*(4/3-1)-1`. Υπολογίστε την τιμή αυτή σε διπλή (double) και μονή (single) ακρίβειας αριθμητική. Συγκρίνετε τα αποτελέσματα με `eps(1)` και `eps(single(1))`.

```
4/3:            .10101010101010101010101x2^1
1:              .10000000000000000000000x2^1
4/3-1:          .10101010101010101010100x2^-1
|3(4/3-1)|:     .10000000000000000000001x2^1
|3(4/3-1)-1|:   .10000000000000000000000x2^-22
n4 =
  1.1921e-007
```

**Άσκηση 4: Ταλαντευόμενη ακρίβεια (wobbling precision)**

Για ένα αριθμό `x`, η MATLAB συνάρτηση `eps(x)` επιστρέφει την απόσταση στον επόμενο μεγαλύτερο αριθμό κινητής υποδιαστολής. Χρησιμοποιείστε την συνάρτηση αυτή για να παραστήστε γραφικά την σχετική απόσταση από `x` στο επόμενο μεγαλύτερο αριθμό μηχανής για 100 ισαπέχοντες αριθμούς στο διάστημα [1, 16] στο IEEE double και single συστήματα.

Όλοι οι αριθμοί στην MATLAB αποθηκεύονται εσωτερικά ακλουθώντας το διπλής ακρίβειας IEEE πρότυπο. Οι αριθμοί κινητής υποδιαστολής έχουν ακρίβεια περίπου 16 σημαντικών δεκαδικών σημείων και κυμαίνονται στο διάστημα 10-308 έως 10+308.

## 5. Σφάλμα αριθμητική κινητής υποδιαστολής

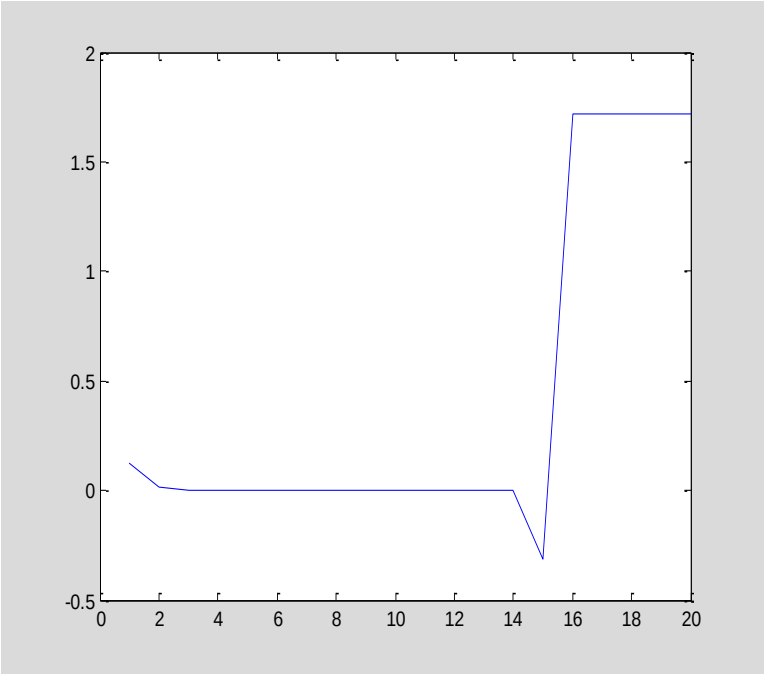
**Άσκηση 5: Στρογγύλευση (Rounding)**

Υπολογίστε την βάση του φυσικού λογαρίθμου από τον ορισμό της:

$$e = \lim_{n\to\infty}(1+\frac{1}{n})^n$$

Για *n = 10^k*, *k* = 1,2,...,20. Κάντε γραφική παράσταση του σφάλματος της προσέγγισης σαν συνάρτησης του *n* . Μπορείτε να εξηγήσετε τα αποτελέσματα;

```
for k=1:20 n=10^k;error(k)= exp(double(1))-(1+double(1/n))^n; end;
plot(error);
```




**Άσκηση 6: Διαγραφή (Cancellation)**

Στην στατιστική, η δειγματική διασπορά των $n$ αριθμών $x_1,...,x_n$ ορίζεται ως:

$$s_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i^2 - \bar{x})^2$$

όπου ο μέσος όρος του δείγματος είναι

$$\bar{x} = \frac{1}{n}(\sum_{i=1}^{n} x_i)^2$$

Ο υπολογισμός της διασποράς από το παραπάνω τύπο απαιτεί τον υπολογισμό του μέσου όρου των δεδομένων και το άθροισμα των τετραγώνων των διαφορών. Ένας εναλλακτικός τύπος που απαιτεί μια χρήση των δεδομένων είναι:

$$s_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2)$$

Υλοποιείστε τους δύο τύπους στην MATLAB και υπολογίστε την διασπορά των `x=10000:10002` σε αριθμητική απλής και διπλής ακρίβειας. Ερμηνεύστε τα αποτελέσματα.

### Άσκηση 7: Αστάθεια χωρίς διαγραφή

Εκτελέστε το παρακάτω αλγόριθμο. Τι δεν πάει καλά;

```
x = 2; M = 60;
for ii=1:M
    x = sqrt(x);
end

for ii=1:M
    x = x^2;
end
x

x =
     1
```

### Άσκηση 8: Διαγραφή σφαλμάτων

Θεωρείστε την συνάρτηση

$$f(x)=\frac{e^x-1}{x}=\sum_{i=0}^{\infty}\frac{x_i}{(i+1)!}$$

που εμφανίζετε σε πολλές εφαρμογές. Θεωρείστε δύο πιθανούς αλγορίθμους για τον υπολογισμό της συνάρτησης:

```
Algorithm 1:
for k=5:16 x=10^(-k);
if x == 0 then
   f = 1;
else
    f = (exp(x)-1)/x;
end;f
end


f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
```

```
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0001
f =
    0.9992
f =
    0.9992
f =
    1.1102
f =
     0
```

```
Algorithm 2:
for k=5:16 x=10^(-k);
y = exp(x);
if y == 1
    f = 1;
else
    f = (y-1)/log(y);
end; f
end

f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
    1.0000
f =
     1
```

Συγκρίνετε τους μαθηματικά ισοδύναμους αλγορίθμους για μικρές τιμές του x:

$$x = 10^{-k},\ k = 5,6,...16$$

σε απλή ακρίβεια. Ποιος από τους αλγορίθμους είναι καλλίτερος; Γιατί;

**Άσκηση 9**

Στην IEEE αριθμητική πόσοι διπλής ακρίβειας αριθμοί υπάρχουν μεταξύ δύο γειτονικών μη μηδενικών απλής ακρίβειας αριθμούς;

**Λύση:**

Σε ένα δυαδικό σύστημα ο αριθμός των ομαλοποιημένων αριθμών κινητής υποδιαστολής μεταξύ δυνάμεων του 2, για κάθε εκθέτη, δίδεται από τον τύπο `2^(t-1)` όπου *t* είναι η ακρίβεια - το μέγεθος σε bits του συντελεστή αναπαράστασης αριθμών (mantissa). Στο IEEE σύστημα αριθμητικής απλής και διπλής ακριβείας, η mantissa έχει μέγεθος 24 και 53 bits, αντίστοιχα. Έτσι για κάθε εκθέτη, υπάρχουν 2^52/2^23 περισσότεροι διπλής ακρίβειας αριθμοί από ότι απλής ακρίβειας αριθμοί, δηλαδή υπάρχουν ***2^29*** (μείον 1) διπλής ακρίβειας αριθμοί μεταξύ κάθε δύο γειτονικών απλής ακρίβειας αριθμών.

Εξηγείστε γιατί ο παρακάτω τύπος δίνει την λύση της άσκησης 9.

```
n = eps(single(1))/eps;k = n - 2^29
```

```
k =
     0
```

**Άσκηση 10**

Γνωρίζουμε ότι ισχύει ο τύπος

$$\sum_{i=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

΄Ενας τρόπος να προσεγγίσουμε το δεύτερο μέρος είναι να υπολογίσουμε το άθροισμα για αυξανόμενες τιμές του *k* έως ότου η τιμή του αθροίσματος δεν θα αλλάξει. Υλοποιείστε την στρατηγική αυτή και υπολογίστε το άθροισμα σε απλή και διπλή ακρίβεια. Πόσα σημαντικά ψηφία ακριβείας παίρνετε σε κάθε περίπτωση;

**Λύση:**

```
clc;
for ii=1:2
    clear k s0 s1 tSum
    switch ii
        case 1
            precision = 'double';
        case 2
            precision = 'single';
    end
    k    = cast(1, precision);
    s0   = cast(0, precision);
    s1   = cast(1, precision);
    tSum = cast(pi, precision)^2/6;   % The true value
```

```
    % Evaluate the sum:
    disp([ upper(precision) ' precision:'] )
    while s1~=s0
        s0 = s1;                    % store the old sum
        k = k + 1;
        s1 = s0 + 1/k^2;            % add the next term
    end

    aErr = abs(s1 - tSum);      % absolute error
    corrDigs = -log10(aErr);

    disp( ['k = ' num2str(k-1) ] );
    disp( ['Absolute error = ' num2str(aErr) ] );
    disp( ['Number of correct digits = ' num2str( round(corrDigs) ) ] )
    disp(' ');
end
```

**Άσκηση 11**

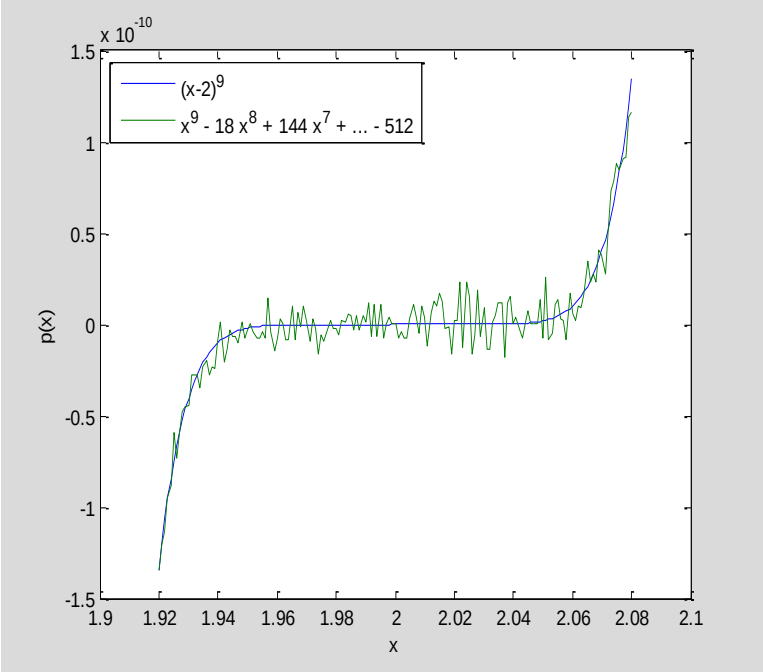
Θεωρείστε το πολυώνυμο :

$$p(x) = (x-2)^9 = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512$$

- Plot p(x) για x=1.920, 1.919, 1.918,..., 2.080, υπολογίζοντας $p$ μέσω των συντελεστών 1, -18, 144, ...
- Παράγετε την ίδια plot ξανά, υπολογίζοντας το $p$ μέσω της έκφρασης (x-2)^9.
- Εξηγείστε τα αποτελέσματα

**Λύση:**

```
x = (1.920:.001:2.080)';
p1 = (x-2).^9;
p2 = x.^9 - 18*x.^8 + 144*x.^7 - 672*x.^6 + 2016*x.^5 - 4032*x.^4 +
5376*x.^3 - 4608*x.^2 + 2304*x - 512;

figure(1);clf;
plot(x,[p1 p2]);
xlabel('x'); ylabel('p(x)');
legend('(x-2)^9', 'x^9 - 18 x^8 + 144 x^7 + ... - 512', 2)
```




**Άσκηση 12**

Ένα σύστημα κινητής υποδιαστολής *F* περιλαμβάνει ένα υποσύνολο ακεραίων.

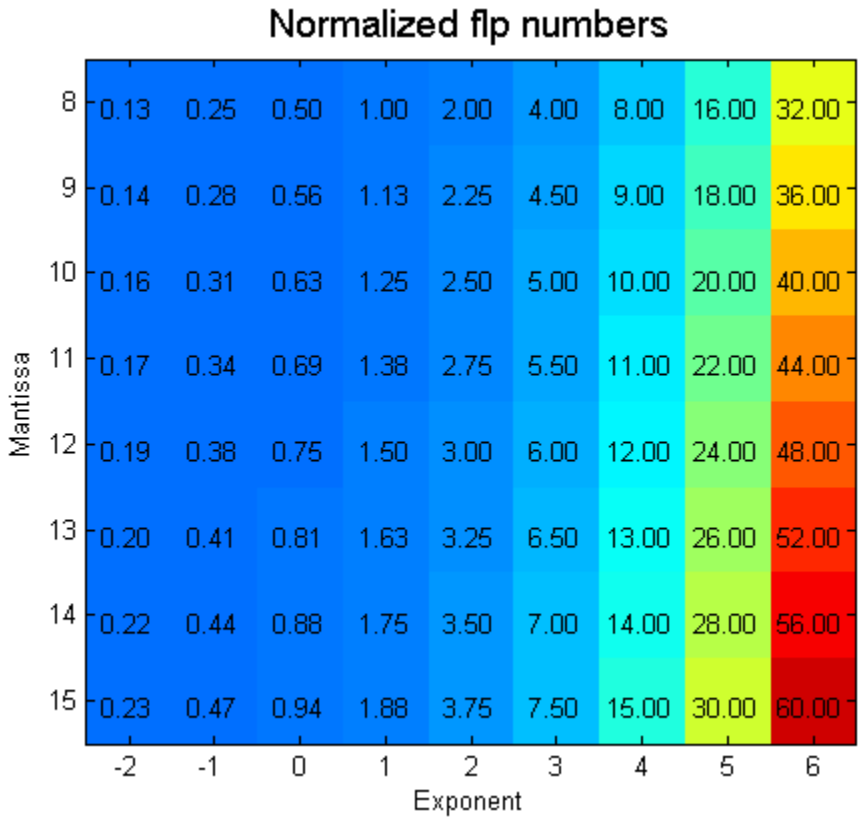
- Να δοθεί ένας τύπος που εκφράζει το μικρότερο θετικό ακέραιο *n* που δεν ανήκει στο *F* .
- Ποιες είναι οι τιμές του *n* για το IEEE πρότυπο μονής και διπλής ακριβείας;
- Να αναπτυχθεί πρόγραμμα που δείχνει ότι *n-3*, *n-2*, και *n-1* ανήκουν στο *F* εκτός του *n*. Το ίδιο να γίνει για τους αριθμούς *n+1*, *n+2*, και *n+3*.

**Λύση:**

Το παρακάτω γράφημα δείχνει του ομαλοποιημένους αριθμούς κινητής υποδιαστολής για το σύστημα με t=4 και e=-2:6. Ποιο ακέραιοι λείπουν;

```
beta = 2;
emin = -2;
emax =  6;
t = 4;
% Some numbers:
M = (beta^t-1) - beta^(t-1) + 1;     % number of distinct mantisses
N = emax - emin + 1;                 % number of distinct exponent
nn = M*N;                                % number of positive normalized
numbers

m = beta^(t-1):beta^t-1;             % mantisses of normalized numbers
y  = fpn(m,emin:emax,beta,t);        % normalized numbers
y2 = fpn(m,emin:emax,beta,t,'bin'); % (in binary format)
% Plot normalized numbers
figure(1);clf; x=magic(nn);viewmatrix(y);axis on;
set(gca, 'XTick', 1:N); set(gca, 'XTickLabel', emin:emax);
```

```
set(gca, 'YTick', 1:M); set(gca, 'YTickLabel', m);
xlabel('Exponent'); ylabel('Mantissa');
title('Normalized flp numbers', 'FontSize', 14);
```




Παρατηρούμε ότι η διαφορά μεταξύ των διαδοχικών αριθμών κινητής υποδιαστολής διπλασιάζεται με την αύξηση του εκθέτη. Συγκεκριμένα, η διαφορά είναι ίση με 2^(*e* - *t*). Αν αυτή η διαφορά γίνει μεγαλύτερη της μονάδος "1", δηλαδή, αν *e* > (*t* + 1), τότε είναι προφανές ότι μερικοί ακέραιοι δεν θα παρίστανται στο σύστημα αυτό. Οι πρώτοι ακέραιοι που δεν παρίστανται δίδονται από τον παρακάτω τύπο:

$$y = \pm m \times 2^{e-t}$$

Είναι ο αριθμός **2^ *t* + 1**. Το παραπάνω φαινόμενο μπορούμε να το επιβεβαιώσουμε και για τους αριθμούς κινητής υποδιαστολής διπλής ακρίβειας. Να κατασκευάσετε 5 διαδοχικούς ακεραίους [2^t-2, 2^t-1, 2^t, 2^t+1, 2^t+2,] και να υπολογίσετε τις 4 διαφορές των διαδοχικών αριθμών. Σημειώστε, ότι στην MATLAB, υπάρχει η συνάρτηση `diff()` που υπολογίζει αυτή την διαφορά και ότι αν όλοι οι αριθμοί μπορούσαν να παρασταθούν με ακρίβεια τότε όλες οι διαφορές θα ήταν «1».

Το output του παρακάτω κώδικα επαληθεύει αυτή την συμπεριφορά.

```
t = 53;
n = 2^t + (-2:2)
diff(n)

n =
  1.0e+015 *
    9.0072    9.0072    9.0072    9.0072    9.0072
ans =
     1     1     0     2
```

## 6. Λογισμικό MatLab

```
function y = fpn(m, e, beta, t, frm)
% Return a floating point number
function [m,e] = real2fpn(r,t)
% Return mantissa and exponent of a real number,
% defined by a binary floating-point system with precision t
function c = chop(x, t)
%CHOP    Round matrix elements.
%        CHOP(X, t) is the matrix obtained by rounding the elements %
of X  to t significant binary places.
%        Default is t = 24, corresponding to IEEE single precision.



function y = fpn(m, e, beta, t, frm)
% Return a floating point number
%
% Usage: y = fpn(m, e, beta, t)
% INPUT:
% m         - mantissa
% e         - exponent
% beta      - basis
% t         - precision
% (optional)
% frm       - {'dec', 'bin'} output format, default 'dec'.
%
% OUTPUT:
% y         - floating point number representation:
%             numeric if the format is 'dec',
%             cell array of strings if 'bin' format.
%
% Examples: fpn(2^52, 1, 2, 53)
%           fpn(2^52, 1, 2, 53, 'bin')
%
% See also: real2fpn.

error(nargchk(4,5,nargin));
% Defaults:
if nargin==4, frm = 'dec'; end

switch lower(frm)
    case 'dec'
        for ii=1:length(e)
            y(:,ii) = m.*beta^(e(ii)-t);
```

```
        end
    case 'bin'
        for ii=1:length(e)
            for jj=1:length(m);
                m2 = dec2bin(m(jj));
                delta = t - length(m2);
                for dum = 0:delta-1
                    m2 = ['0' m2];
                end
                y{jj,ii} = ['.' m2 'x' num2str(beta) '^'
num2str(e(ii))];
            end
        end
    otherwise
        error('Unknown format parameter.');
end

function [m,e] = real2fpn(r,t)
% Return mantissa and exponent of a real number,
% defined by a binary floating-point system with precision t
%
% Usage: [m,e] = real2fpn(r,t)
% INPUT:
% r         - real number
% t         - precision
% OUTPUT:
% m, e      - mantissa and exponent so that, r = m*2^(e-t)
%
% Examples: t=53; [m,e] = real2fpn(1,t), fpn(m,e,2,t)
%
% See also: fpn.

y = abs(r) + (r==0);

e = floor(log2(y)+1);
m = y/2^(e-t);

function c = chop(x, t)
%CHOP    Round matrix elements.
%        CHOP(X, t) is the matrix obtained by rounding the elements of
X
%        to t significant binary places.
%        Default is t = 24, corresponding to IEEE single precision.

if nargin < 2, t = 24; end
[m, n] = size(x);

%  Use the representation:
%  x(i,j) = 2^e(i,j) * .d(1)d(2)...d(s) * sign(x(i,j))

%  On the next line `+(x==0)' avoids passing a zero argument to LOG,
which
%  would cause a warning message to be generated.

y = abs(x) + (x==0);
e = floor(log2(y) + 1);
```

```
c = pow2(round( pow2(x, t-e) ), e-t);
```

### 7. Λογισμικό Python

Το παρακάτω παράδειγμα δείχνει την μετατροπή αριθμών στο διαδικό σύστημα:

```
>>> from convertno2bin import *
>>> int2bin(35)
'00000000000000000000000000000000000000000000000000000000100011
'
>>> float2bin(35.4)
'01000000010000011011001100110011001100110011001100110011001100110011
'
>>> float2sem(35.4)
('0', '10000000100',
'0001101100110011001100110011001100110011001100110011001')
>>> float2dec(35.4)
(0, 5, 1.1062499999999997)
```

```
------------------------------------------------
# module: convertno2bin
# convert an  integer to a binary string (base 2)
# convert a real to binary (base 2)
# returns a triple: binary representation of sign, #
exponent and mantissa
# returns a triple: sign, exponent as integer, #mantissa as
float
```

```
from ctypes import *
def int2bin(n, count=64):
    """returns the binary of integer n, using count number
of digits"""
    return "".join([str((n >> y) & 1) for y in range(count-
1, -1, -1)])

def float2bin(f):
    """returns the binary representation of a floating
point number f"""
    cf = c_double(f)
    pcf = pointer(cf)
    pcf_i = cast(pcf, POINTER(c_int64))
    return int2bin(pcf_i.contents.value)


def float2sem(f):
    """returns a triple: binary representation of sign,
exponent and mantissa"""
    str = float2bin(f)
    return ( str[0:1], str[1:12], str[12:-1])
```

```
def float2dec(f):
    """returns a triple: sign, exponent as integer,
mantissa as float"""
    str = float2bin(f)
    return ( int(str[0:1],2), int(str[1:12],2)-1023, 1.0+
2.0**(-51)*int(str[12:-1],2))
```

## 8. Αναφορές


1. https://sites.google.com/a/uni-konstanz.de/na09/Home/methods/fpns
2. Αριθμητικές Μέθοδοι στην Επιστήμη και Μηχανική, C. Pozrikidis, Εκδόσεις Τζιόλα, 2006
3. Numerical Methods in Engineering with Python, Jaan Kiusalaas, Cambridge University Press, 2005.
4. Numerical Methods for Engineers, with Software and Programming Applications, S.C.Chapra and R.P. Canale, Mc Graw Hill, 2002.
5. Numerical Methods, Software, and Analysis, J.R. Rice, Mc Graw Hill 1983.
6. Applied Numerical Methods, B.C. Carnahan, H.A. Luther, J. O. Wilkes, Krieger Publishing company, 1990.

# ΠΑΡΑΡΤΗΜΑΤΑ

# ΕΙΣΑΓΩΓΗ ΣΤΗΝ MATLAB, PYTHON, & ΕΝΝΟΙΕΣ ΚΑΙ ΑΡ. ΜΕΘΟΔΟΙ ΓΡΑΜΜΙΚΗΣ ΑΛΓΕΒΡΑΣ

## Περιεχόμενα

## Παράρτημα 3: Γραμμική Άλγεβρα: Έννοιες και Αριθμητικές Μέθοδοι ..... 395



Υπάρχουν αρκετές εισαγωγικές παρουσιάσεις των υπολογιστικών περιβαλλόντων MATLAB και PYTHON. Για την πληρότητα των σημειώσεων αυτών παραθέτουμε δύο σύντομες και περιεκτικές παρουσιάσεις αυτών των προγραμματιστικών συστημάτων με έμφαση τους επιστημονικούς υπολογισμούς. Επίσης, περιλαμβάνουμε μια σύντομη γραφική περιγραφή βασικών εννοιών της γραμμικής άλγεβρας χρησιμοποιώντας ένα γραφικό toolbox της MatLab. Τέλος, δίδεται μια σύντομη περιγραφή των αριθμητικών μεθόδων που παρουσιάστηκαν στα κεφάλαια 6, 7, και 8 των σημειώσεων μαζί με ένα σύνολο ασκήσεων.

# Παράρτημα 1: Εισαγωγή στο υπολογιστικό περιβάλλον MatLab

Η παρακάτω ύλη είναι προσαρμογή στα Ελληνικά της ύλης που βρίσκεται στην ιστοσελίδα http://dmpeli.mcmaster.ca/Matlab/Math1J03/LectureNotes.

## 1. Μεταβλητές και ακολουθίες (διανύσματα)

**Γενικές εντολές:**

- **helpwin, helpdesk:** ανοίγει το παράθυρο διαδραστικής βοήθειας
- **help functionname:** δίνει την περιγραφή "***functionname***" στο Παράθυρο Εντολών

```
help date
```

```
 DATE   Current date as date string.
    S = DATE returns a string containing the date in dd-mmm-yyyy
format.

    See also NOW, CLOCK, DATENUM.
```

**dir, ls:** εμφανίζει τα περιεχόμενα των αρχείων και υποκαταλόγων του τρέχοντος καταλόγου εργασίας
**cd c:** αλλάζει τον κατάλογο εργασίας σε "***c:***"
**mkdir dirname:** δημιουργεί ένα νέο υποκατάλογο ***"dirname"***
**who, whos:** δίνει όλες τις αρχικές μεταβλητές στο υπάρχον περιβάλλον
**clock:** επιστρέφει την συστοιχία για [χρόνο,μήνα,μέρα,ώρα,λεπτό,δευτερόλεπτο]

```
  clock
fix(clock) % sets the variable clock as the arrray of integers
```

```
ans =  1.0e+003 *
                2.0020    0.0010    0.0040    0.0140    0.0440
0.0245
ans =
        2002           1           4          14          44
24
```

**diary on, diary off:** καταγράφει κάθε δραστηριότητα από το Παράθυρο Εντολών σε ένα αρχείο που ονομάζεται***"diary***", είναι χρήσιμο για να καταγράφετε το ιστορικό σας για εργασίες και εργαστήρια.
**diary filename:** καταγράφει δραστηριότητες σε ένα αρχείο με το όνομα "***filename***", αν το αρχείο υπάρχει, και προσαρτείται ο νέος κατάλογος.

**Μεταβλητές:**

ονόματα και είδη μεταβλητών δεν είναι απαραίτητο να δηλωθούν

- κάθε αντικείμενο στο MATLAB είναι ένας μιγαδικός πίνακας ή τα παράγωγά του

- οι κλιμακωτές μεταβλητές ορίζονται ως πίνακες 1x1 ακέραιων, πραγματικών ή μιγαδικών τιμών
- οι κλιμακωτές μεταβλητές μπορεί αργότερα να επεκταθούν σε διανύσματα και πίνακες
- τα ονόματα των μεταβλητών δεν θα πρέπει να έρχονται σε αντίθεση με τις λέξεις κλειδιά τις συναρτήσεις και τις εντολές της MATLAB

```
x = 2 % a numerical variable
abc = 'student' % a string variable
```

```
x =             2
abc =           student
```

```
end = 1 % absolutely bad variable
```

```
Error: Missing operator, comma, or semicolon.
```

```
% Relatively bad variables:
sin(2)   % the value of math.function sin(x) at x = 2
sin = 10  % sin becomes the variable name
sin(2) % sin is no longer associated with the name for math.function sin(x)
```

```
ans =    0.9093
sin =    10
???  Index exceeds matrix dimensions.
```

```
% Another example of relatively bad variable:
x = 2 + 3*i  % x is a complex number, since i = sqrt(-1)
i = 2  % i is assigned to another number, 2
x = 2 + 3*i % i is not longer associated with the name for sqrt(-1)
```

```
x =
   2.0000 + 3.0000i
i =
     2
x =
     8
```

```
% Special numbers and variable names:
eps, pi, inf, NaN, date
```

```
ans =
                2.2204e-016
ans =
          3.1416
ans =
                Inf
ans =
          NaN
ans =
     14-Dec-2001
```

- όλοι οι υπολογισμοί στην MatLab είναι διπλής ακριβείας
- **format:** αλλάζει μεταξύ διαφορετικών μορφών/τύπων απόδοσης

```
format short; pi % 4 decimal digits
  format long; pi % 14 decimal digits
  format short e; pi % 4 decimal digits in exponential notations
  format long e; pi % 14 decimal digits in exponential notations
  format hex; pi % hexagonal representation
```

```
  ans =
                  3.1416
  ans =
                      3.14159265358979
  ans =
                      3.1416e+000
  ans =
                3.141592653589793e+000
  ans =
                400921fb54442d18
```

- **clear variablename:** διαγράφει την τιμή του *"variablename"* από τον υπάρχοντα χώρο εργασίας
- **clear all:** διαγράφει τις τιμές από όλες τις μεταβλητές
- **clc** – καθαρίζει το παράθυρο εντολών και μετακινεί το δρομέα στην αρχή

**Αριθμητικές πράξεις:**

"+", "-", "*", "/" – συνηθισμένα σημεία για πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση
"\" – αντίστροφη διαίρεση (3\1 = 1/3 = 0.33333)
"^" – σημείο για τον εκθέτη
Υπάρχουν και περισσότερες λειτουργίες για διανύσματα και πίνακες
";" – η εντολή εκτελείται αλλά το αποτέλεσμα δεν εκτυπώνεται, μια σειρά μπορεί να περιέχει αρκετές εντολές
"," – σημείο που διαχωρίζει τις εντολές, το αποτέλεσμα εκτυπώνεται για κάθε εντολή
"…" – η συνέχεια μιας μεγάλης εντολής σε πολλές σειρές
"%" – σήμα σχολίου, ολόκληρη η σειρά που ακολουθεί το σήμα σχολίου αγνοείται κατά τους υπολογισμούς

```
   % Example of a mini-script:
r = 10; % radius of a sphere
vol = 4*pi
...*r^3/3; % computation of volume of a sphere
r,vol
```

```
   r =
            10
vol =
            12.5664
```

**Διανύσματα ως μονοδιάστατοι πίνακες:**

- **Διανύσματα στήλης: x** = [x1 ; x2 ; … ; xN]
- **Διανύσματα σειράς: x** = [x1, x2, … , xN]

```
x = [ 0, 0.5, 1, 1.5, 2], y = [ 0; 0.5; 1; 1.5; 2]
```

```
x =      0    0.5000    1.0000    1.5000    2.0000
y =      0
    0.5000
    1.0000
    1.5000
    2.0000
```

**Πρόσβαση σε μεμονωμένα στοιχεία: x(j), y(k)**

```
  x(2), y(5)

  ans =     0.5000
ans =   2

x(6)
???  Index exceeds matrix dimensions.
y(0)
??? Index into matrix is negative or zero.
```

**Αλλαγή του μήκους ενός διανύσματος (resize):**

```
x(8) = 4; x % increase the dimension of x to 8 and assign 4 to x(8)

x =    0    0.5000    1.0000    1.5000    2.0000         0         0
4.0000

  y = y(2:4); y
% reduce the dimension of y to 3 and assign y(2),y(3),y(4) to a new
vector

y =
    0.5000
    1.0000
    1.5000
```

**Αλλαγή του τύπου ενός διανύσματος:**

- **transpose: x',y'**

```
z1 = y', z2 = y''

z1 =
    0.5000    1.0000    1.5000
z2 =
    0.5000
    1.0000
    1.5000
```

**Υπολογισμός μήκους και διαστάσεις ενός διανύσματος**

**length:** υπολογίζει τον αριθμό των στοιχείων σε ένα διάνυσμα

**size:** υπολογίζει τη διάσταση ενός διανύσματος, λαμβάνοντας υπόψη την οριζόντια ή την κάθετη δομή του

```
length(z1), length(z2), size(z1), size(z2)

ans =
     3
ans =
     3
ans =
     1     3
```

```
ans =
1
```

**Εισαγωγή διανυσμάτων που ισαπέχουν:**

**type:** x = [ 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1 ]

- **colon vectorization:** x = 0 : 0.1 : 1 *(first element, step size, last element)*

```
x1 = 0 : 0.1 : 0.25 % the last element is not reached
x2 = 0 : 5 % default step size = 1
x3 = 5 : -1 : 0 % the step size can be negative
x4 = 0 : -1 : 1 % non-valid operation produces empty vector
length(x4) % the empty vector x4 has a zero length
```

```
x1 =
         0    0.1000    0.2000
x2 =
     0     1     2     3     4     5
x3 =
     5     4     3     2     1     0
x4 =
   Empty matrix: 1-by-0
ans =
     0
```

**Έμμεσος ορισμός διανυσμάτων**

**έμμεσος τρόπος παραγωγής δεικτών (implicit indexation):** k = 1 : 11; x(1:11) = 0.1*k

```
   y(1:3:12) = 1; y(2:3:12) = 2; y(3:3:12) = 3; y
```

```
y =
     1     2     3     1     2     3     1     2     3     1     2
3
```

**linspace:** x = linspace(0,1,11) **(***first element,last element, number of elements***)**

```
  x1  = linspace(0,0.25,5)
% the built-in function never mismatches the last element
x4  = linspace(0,-1,10) % the vectors are always non-empty
h1 = x1(2) - x1(1)
% the step size can be easily computed (need not to be given)
h4 = x4(2) - x4(1)
```

```
x1 =         0    0.0625    0.1250    0.1875    0.2500
x4 =         0   -0.1111   -0.2222   -0.3333   -0.4444   -0.5556   -
0.6667   -0.7778   -0.8889   -1.0000
h1 =    0.0625
h4 =   -0.1111
```

**Ορισμός βρόγχου ανά στοιχείο αναφοράς (pointwise definition (a for-loop)):**

for k = 1:11, x(k) = (k-1)*0.1, end

**Λειτουργίες διανυσμάτων:**

**Προσθέσεις και αφαιρέσεις διανυσμάτων ίδιου μεγέθους**

```
x = [ 0, 1, 2 ]; y = [ 0.1, 0, 0.1]; z = [ 10, 20 ];
x + y
x - y

ans =    0.1000    1.0000    2.1000
ans =   -0.1000    1.0000    1.9000

x + z

??? Error using ==> +
Matrix dimensions must agree.
```

- **Προσθέσεις και πολλαπλασιαμοί βαθμοτών τιμών με διανύσματα**

```
0.2 + x
0.2 * x

ans =    0.2000    1.2000    2.2000
ans =         0    0.2000    0.4000
```

- **Πολλαπλασιασμοί και διαιρέσεις ίδιου μεγέθους διανυσμάτων ανά στοιχείο των διανυσμάτων**

```
x.*y % main advantage of MATLAB (matrix laboratory)
x./y % no loops for accessing individual elements are required
x.\y % inverse division, the same as y./x

ans =         0         0    0.2000
  Warning: Divide by zero.
ans =     0   Inf    20
Warning: Divide by zero.
ans =       Inf         0    0.0500

x.*z

??? Error using ==> .*
Matrix dimensions must agree.
```

- **Δυνάμεις διανυσμάτων ανά στοιχείο του**

```
  x.^3, y.^(0.1)
     x.^y % the same as x(k)^(y(k)) for any k
   3.^x % the same as 3^(x(k)), the output is a vector of the same
structure as x

ans =     0     1     8
ans =    0.7943         0    0.7943
ans =         0    1.0000    1.0718
ans =     1     3     9
```

- **Προσάρτηση διανυσμάτων**

```
w = [ x,  y,  z]
w = [x'; y'; z']

w =  0    1.0000    2.0000    0.1000         0    0.1000   10.0000
20.0000
```

```
w =      0
    1.0000
    2.0000
    0.1000
         0
    0.1000
   10.0000
   20.0000
```

- **Διαγραφή στοιχείων διανυσμάτων**

```
x(2) = []; x
w(3:6) = []; w

x =     0     2
w =          0
      1
             10
             20
```

## 2. Συμβολοσειρές (Strings) σαν διανύσματα χαρακτήρων

```
str1 = 'Dmitry10';
str1(1:6) % the first three characters are displayed
length(str1) % the length of character vector "str1"
str2 = 'Pelinovsky20';
str3 = [str1, str2] % concatanetion of two string vectors
str4 = [str1(1:6),' ',str2(1:10)]

ans = Dmitry
ans =  8
str3 =      Dmitry10Pelinovsky20
str4 =      Dmitry Pelinovsky
```

## 3. Μαθηματικές συναρτήσεις και M-αρχεία

**Στοιχειώδεις μαθηματικές συναρτήσεις:**

- **sqrt:** συνάρτηση τετραγωνικής ρίζας ( η *MATLAB λειτουργεί γενικά και με πραγματικούς και με μιγαδικούς αριθμούς)*

```
  sqrt(5), sqrt(-5)

ans =    2.2361
ans =    0 + 2.2361i

      % A quadratic equation, a*x^2 + b*x + c = 0, has two solutions
(roots):
      % x = (-b +/- sqrt(b^2 - 4*a*c))/(2*a)
  a = 1; b = 1; c = 10;
      x1 = (-b + sqrt(b^2-4*a*c))/(2*a)
  x2 = (-b - sqrt(b^2-4*a*c))/(2*a)
```

```
     % the two solutions are complex since b^2 < 4*a*c
      a = 1; b = 10; c = 1;
      x1 = (-b + sqrt(b^2-4*a*c))/(2*a)
  x2 = (-b - sqrt(b^2-4*a*c))/(2*a)
           % the two roots are real since b^2 > 4*a*c

x1 =  -0.5000 + 3.1225i
x2 =  -0.5000 - 3.1225i
x1 =   -0.1010
x2 =   -9.8990
```

- **sin, cos, tan:** τριγωνομετρικές συναρτήσεις
- **asin, acos, atan:** αντίστροφες τριγωνομετρικές συναρτήσεις
- **sinh, cosh, tanh:** υπερβολικές συναρτήσεις
- **asinh, acosh, atanh:** αντίστροφες υπερβολικές συναρτήσεις
- **exp, log, log10:** εκθετικές συναρτήσεις

```
  a1 = cos(3-4*i) % MATLAB functions work for complex arguments
   a2 = exp(7+5*i)
   a3 = tan(0:1:5) % MATLAB functions work for vector arguments
      a4 = cosh([ 0,1; -1,-2; 0.1,0.2])
     % MATLAB functions work for matrix arguments, such that
           % the output of MATLAB functions reproduces the input
structure
           % Note that the matrix-vector calls to MATLAB functions are
more efficient
     % computationally than pointwise calls for each element of an array


a1 =
      -27.0349 + 3.8512i
a2 =
       3.1107e+002 -1.0516e+003i
a3 =
         0    1.5574   -2.1850   -0.1425    1.1578   -3.3805
a4 =
    1.0000    1.5431
    1.5431    3.7622
    1.0050    1.0201

   b1 = acos(0.5) % the argument of the standard "acos" is between [-
1,1]
      b2 = acos(5) % the MATLAB "acos" is analytically continued beyond
[-1,1]
      b3 = acosh(5) % imaginary part of acos(5) is the same as real
part of acosh(5)
      b4 = acosh(0.5) % vice verse
      b5 = log(exp(5)) % log(x) is the logarithm of base e
      b6 = log10(1000) % log10(x) is the logarithm of base 10
      format long; b7 = atan(inf) % the range of atan(x) is between [-
pi/2,pi/2]
      pi/2, format short;

b1 =    1.0472
b2 =    0 - 2.2924i
```

```
b3 =    2.2924
b4 =    0 + 1.0472i
b5 =    5
b6 =    3.0000
b7 =   1.57079632679490
ans =  1.57079632679490
```

- **real, imag, conj:** πραγματικά, φανταστικά τμήματα μιγαδικού αριθμού, and the συζυγής μιγαδικός αριθμός
- **abs, angle:** απόλυτη τιμή μιγαδικού αριθμού, η γωνία φάσης ενός μιγαδικού αριθμού

```
      z = 1 + 2*i; % a complex number
      r = abs(z), theta = angle(z) % polar form of a complex number
      comp1 = real(z) - r*cos(theta) % Euler formulas for complex
exponentials
      comp2 = imag(z) - r*sin(theta)
```

```
r =    2.2361
theta =    1.1071
comp1 = -2.2204e-016
comp2 =     0
```

- **round:** στρογγυλοποίηση στον πλησιέστερο ακέραιο
- **floor:** στρογγυλοποίηση στο μικρότερο ακεραίο
- **ceil:** στρογγυλοποίηση στο μεγαλύτερο ακέραιο
- **fix:** στρογγυλοποίηση στο μικρότερο ακέραιο αν είναι θετικό και στο μεγαλύτερο αν είναι αρνητικό
- **sign:** 1 αν είναι θετικό, 0 ή μηδενικό και –1 αν είναι αρνητικό

```
 c1 = round(1.5), c2 = round(1.4999999), c3 = round(-1.5), c4 = round(-
1.4999999)
 d1 = floor(1.9999), d2 = floor(-1.0001), d3 = ceil(1.0001), d4 =
ceil(-1.99999)
 e1 = fix(1.99999), e2 = fix(-1.99999), e3 = sign(1.01), e4 = sign(-
1.01)
```

```
c1 =     2
c2 =     1
c3 =    -2
c4 =    -1
d1 =     1
d2 =    -2
d3 =     2
d4 =    -1
e1 =     1
e2 =    -1
e3 =     1
e4 =    -1
```

- **mod,rem:** υπόλοιπο κατά τη διαίρεση ακεραίων

```
 % mod(x,y) = x - y*ceil(x/y)
  % rem(x,y) = x - y*fix(x/y)
      % rem(x,y) = mod(x,y) if sign(x) = sign(y)
```

```
f1 = mod(5,3), f2 = rem(5,3)
f3 = mod(-5,3), f4 = rem(-5,3)
f5 = mod(5,-3), f6 = rem(5,-3)
f7 = mod(-5,-3), f8 = rem(-5,-3)
```

```
f1 =     2
f2 =     2
f3 =     1
f4 =    -2
f5 =    -1
f6 =     2
f7 =    -2
f8 =    -2
```

- **factorial:** παραγοντισμός ενός ακεραίου
- **factor:** παραγοντοποίηση ενός ακέραιου αριθμού σε περιττούς αριθμούς
- **isprime:** επιστρέφει 1 αν αριθμός είναι περιττός και 0 αν δεν είναι περιττός

```
g1 = factorial(5), g2 = factorial(15), g3 = factor(150), g4 =
isprime(150)
```

```
g1 =   120
g2 =  1.3077e+012
g3 =     2     3     5     5
g4 =     0
```

```
g5 = factorial(5.1),
% the functions are not continued for non-integer values
```

```
??? Error using ==> factorial
N must be a positive integer
```

```
% Remove the numbers divisible by 3 from a vector:
x = [-9, 0, 2, 5, 3, 7, 2, 0, 6, 12, 214342124187]
disp(sprintf(' %12.0f ',x));
m = 0;
for k = 1 : length(x) % looping through the vector x
    if mod(x(k),3) ~= 0 % comparison if x(k) is not divisible
by 3
        m = m + 1;
        y(m) = x(k); % saving x(k) into a dynamically created
vector y
    end
end
x = y
```

```
x =
  1.0e+011 *
   -0.0000         0    0.0000    0.0000    0.0000    0.0000    0.0000
0    0.0000    0.0000    2.1434
           -9            0            2            5            3
7            2            0            6           12  214342124187
x =
     2     5     7     2
```

## 4. Πρότυπο API (Application Programming Interface) συναρτήσεων:

- **sort:** η συνάρτηση ανακατατάσσει στοιχεία ενός διανύσματος σε αύξουσα σειρά

```
x = [2 1 4 10 7 3 1 ]
y = sort(x) % both row-vectors and column-vectors can be sorted
[y,index] = sort(x); % index show how elements of y appear in x: y =
x(ind)
     x(ind)
```

```
x =     2     1     4    10     7     3     1
y =     1     1     2     3     4     7    10
ans =   1     1     2     3     4     7    10
```

- **sum:** η συνάρτηση αθροίζει όλα τα στοιχεία ενός διανύσματος

```
     sum(x), sum(y')
A = [ 1,2,3; -4,-5,-6] % a two-dimensional array (matrix)
  sum(A) % returns the row-vector for sums of each column of A
```

```
ans =    28
ans =    28
A =      1     2     3
        -4    -5    -6
ans =
    -3    -3    -3
```

- **max,min:** βρίσκει το μέγιστο και το ελάχιστο ενός διανύσματος

```
     [maxX,indX] = max(x) % maxX - the maximal element, indX - its
 position in x
  [minX,indX] = min(x) % if the minimum element is multiple, indX is
 the index of the first element
  [maxA,indA] = max(A) % returns the row vectors for max of each column
of A
```

```
maxX =    10
indX =     4
minX =     1
indX =     2
maxA =     1     2     3
indA =     1     1     1
```

- **rand(n):** παράγει έναν πίνακα δευτέρου βαθμού (n by n) τυχαίων αριθμών, και ο κάθε τυχαίος αριθμός ανήκει στο διάστημα [0,1]

```
a1 = rand(1), a2 = rand(1), a3 = rand(1)
```

```
a1 =    0.4565
a2 =    0.0185
a3 =    0.8214
```

- **rand('seed',k):** παράγει τον προσδιορισμό της αρχικής τιμής ενός τυχαίου αριθμού από ένα αρχικό αριθμό(seed number) k (>=1)

NB: αν ο αρχικός αριθμός(seed) είναι ο ίδιος, η ακολουθία τον τυχαίων αριθμών είναι ίδια. Για να παράγουμε μια διαφορετική ακολουθία τυχαίων αριθμών, ο αρχικός αριθμός(seed) επιλέγεται σύμφωνα με τον τυχαίο παράγοντα (δηλ. μετρώντας τη μέρα, το χρόνο σε δευτερόλεπτα, κλπ.)

```
     c = clock % returns a row-vector of length 6
 k = c(2)*c(3)*c(4)*c(5)*c(6) % this product has 3*10^7 combinations
and changes every second during the year
 rand('seed',k); % initialization of the random number generator
depends on the time of computations
 for n = 1 : 10
          m = ceil(12*rand(1)); % randomly choose one the 12 integer
numbers between 1 and 12 for 10 times
          fprintf('the random number is: %3.0f\n',m);
end
```

```
c =  1.0e+003 *
      2.0010    0.0120    0.0280    0.0110    0.0190    0.0023
k =  1.5934e+005
the random number is:  11
the random number is:   4
the random number is:  10
the random number is:   6
the random number is:   3
the random number is:   9
the random number is:   1
the random number is:   6
the random number is:   8
the random number is:   5
```

- **eval('stringName'):** εκτελεί την εντολή που δίνεται από το *"stringName"*

```
     FunctionName = input('Type a function name: ','s');
      % an user is supposed to type a name of valid MATLAB function
     x = 0 : 0.2 : 1;
     strCommand = [ FunctionName, '(x)' ]; % concatanetion of two
strings
     y = eval(strCommand) % evaluate the command, e.g. y = sin(x)
```

```
Type a function name: sin
 y =
         0    0.1987    0.3894    0.5646    0.7174    0.8415
 Type a function name: exp
 y =
    1.0000    1.2214    1.4918    1.8221    2.2255    2.7183
```

```
          % Example of authomatical creating names of files:
 for k = 1 : 1000
          y = exp((k-1)*0.1); % the data y is dynamically created for
each value of k
          if ( k <= 10 )
```

```
                s = sprintf('save file00%1d y -ascii',k-1); % saving
 the data y into file "file00k" for k < 10
           elseif ( k <= 100 )
                s = sprintf('save file0%1d y -ascii',k-1); % saving
 the data y into file "file0k" for 10 <= k < 100
     else
                s = sprintf('save file%1d y -ascii',k-1); % saving
 the data y into file "filek" for 100 <= k < 1000
           end
           eval(s); % executing the string with the command to save
 the data, 1000 files are created in the loop
 end
```

## 5. Μ-αρχεία για διαδικασίες και συναρτήσεις:

Τα αρχεία M-files είναι χρήσιμα για μεγάλες εντολές και διαδικασίες και μπορούν να αποθηκευτούν σε δίσκο και να διορθωθούν όσες φορές χρειαστεί.

**script m-files:**

ένα αρχείο script m-file αντιστοιχεί σε ένα κύριο πρόγραμμα σε άλλες προγραμματιστικές γλώσσες

ένα αρχείο script m-file μπορεί να περιλαμβάνει οτιδήποτε ο χρήστης θα έγραφε σε ένα παράθυρο εντολών

όλα τα σχόλια στην αρχή του script μπορούμε να τα δούμε με on-line βοήθεια

**echo on:** όλες οι δηλώσεις εμφανίζονται στην οθόνη καθώς εκτελούνται

μπορούμε να επικαλεστούμε ένα αρχείο script m-file εντός άλλων script m-files και όλες οι μεταβλητές στον υπάρχοντα χώρο εργασίας μοιράζονται μεταξύ του αρχικού γονικού και του απογόνου m-file.

**Συνάρτηση m-files:**

Η συνάρτηση m-files αντιστοιχεί σε υπόρουτίνες ή συναρτήσεις σε άλλες προγραμματιστικές γλώσσες

Η κάθε συνάρτηση αποθηκεύετε ως μεμονωμένο m-file

Μια συνάρτηση m-file αρχίζει με τη σειρά: *function [y1,y2,…,yN] = functionName(x1,x2,…,xM)*

*y1,y2,…,yN* – N τιμές απόδοσης

*x1,x2,…,xM* – M παράγοντες εισόδου

*functionName* – είναι το ίδιο με το όνομα του m-file

όλα τα σχόλια μετά την πρώτη σειρά (δήλωση της συνάρτησης) μπορούμε να τα δούμε με την on-line βοήθεια ως περιγραφή των MATLAB συναρτήσεων

ο πυρήνας μίας συνάρτησης m-file είναι ένα στιγμιότυπο κώδικα (script) το οποίο χρησιμοποιεί *x1,x2,…,xM*, εκτελεί οποιουσδήποτε ενδιάμεσους υπολογισμούς και τελικά υπολογίζει το *y1,y2,…,yN*

```
 function [mean,stdev] = mean_stdev(x)
  % [mean,stdev] = mean_stdev(x)
      % x is a vector, mean and stdev are scalars
      % the function computes the mean value and the standard deviation
% of a data sample x

      n = length(x); % x is a vector
      if ( n ~= 0 ) % if x is not empty vector
                        mean = sum(x)/n; % compute the mean value of x
```

```
                        varComp = sum((x-mean).^2)/n; % compute the
variance of x
                        stdev = sqrt(varComp); % compute the standard
deviation of x
      end
```

μπορούμε να επικαλεστούμε μια συνάρτηση m-file εντός μιας άλλης συνάρτησης m-file, ή ενός αρχείου κώδικα m-file, ή άμεσα στο παράθυρο εντολών, Η συνάρτηση μοιράζεται μόνο μεταβλητές που δίνονται ως δεδομένα.

```
      x = [1 2 3 4 5 6 7 8 9  10 ];
[mu,st] = mean_stdev(x)
x = 3;
[mu,st] = mean_stdev(x)

mu =    5.5000
st =    2.8723
mu =     3
st =     0
```

```
           mean_stdev(3)
varComp % the variable "varComp" is defined in the function
"mean_stdev",
% but this variable is not accessible in the current working space

ans =
     3
??? Undefined function or variable 'varComp'.
```

Η συνάρτηση της MATLAB m-file θα πρέπει να γενικεύετε για στοιχεία διανυσμάτων και πινάκων.

```
function y = powerfunction(x,n)
      y = x.^n; % the pointwise power ".^" works both for scalars and
vectors

      x = 3; powerfunction(x,2)
x = [ 1 2 3 4 5 6 ]; powerfunction(x,3)

ans =
     9
ans =
     1     8    27    64   125   216
```

**feval:** υπολογίζει μια συνάρτηση m-file, e.g. *[y1,y2,…,yN] = feval(f_name,x1,x2,…,xM)*, όπου *"f_name"* είναι το αναγνωριστικό για τη συνάρτηση; *x1,x2,…,xM* είναι τιμές των δεδομένων και *y1,y2,…,yN* είναι οι τιμές του αποτελέσματος.

```
      % compute a weighted average of a function y = f(x) at three
points a,b,c by using the formula:
```

```
% f_av = (f(a) + 2 f(b) + f(c))/4

function f_av  = average(f,a,b,c)
      a1 = feval(f,a);
b1 = feval(f,b);
c1 = feval(f,c);
      f_av = 0.25*(a1 + 2*b1 + c1);

      w1 = average('sqrt',1,2,3)
% call to the standard MATLAB function y = sqrt(x)
w2 = average('sin',1,2,3)
% call to the standard MATLAB function y = sin(x)
```

```
w1 =
    1.3901
w2 =
    0.7003
```

## 6. I/O λειτουργίες (Reading and Writing Operations)

**Διαδραστικές λειτουργίες με τη χρήση του πληκτρολογίου και του ποντικιού:**

- **input:** παίρνει τα δεδομένα στοιχεία από τον χρήστη, όταν πληκτρολογεί μια τιμή και χτυπάει το πλήκτρο return
  - για αριθμητικά δεδομένα: *z = input('string'), 'όπου 'string' είναι ένα προτρεπτικό μήνυμα.*

```
  r = input('Enter radius of a sphere: ')
```

```
  Enter radius of a sphere: 5
r =
     5
```

για ένα κείμενο δεδομένων: *z = input('string','s'), όπου z είναι το αναγνωριστικό για το κείμενο δεδομένων*

```
z = input('Enter your name: ','s')
length(z)
z(1)
z(6)
```

```
 Enter your name: Dmitry
z =
Dmitry
ans =
      6
ans =
D
ans =
y
```

  - για γενική χρήση: z = *input('string')*, όπου πληκτρολογούμε *variable* για αριθμητικά δεδομένα και *'variable'* για αλφαριθμητικά δεδομένα

```
z1 = 1894 % this is number
```

```
z2= '1894' % this is a string
length(z2), z2(1), z2(4) % z2 is array of four characters
length(z1), z1(1) % z1 is scalar of one numerical value
z3 = str2num(z2) % conversion from string to number
(z1 == z3) % yes, z3 is now the same as z1, i.e. it is the number
z4 = num2str(z1) % conversion from number to string
(z2 == z4) % yes, z4 is now the same as z2, i.e. it is the string
% NB: when array variables are compared, each element is compared
individually
% NB: for comparisons of array variables, they must be of the same
size!
```

```
z1 =        1894
z2 =              1894
ans =       4
ans =             1
ans =       4
ans =       1
ans =     1894
z3 =      1894
ans =     1
z4 =              1894
ans =     1     1     1     1
```

- **disp:** δείχνει αριθμούς, διανύσματα, πίνακες, and strings

```
disp(1234) % 1234 is a number
disp([1, 2, 3, 4]) % [1,2,3,4] is a vector
disp([1, 2; 3, 4]) % [1,2;3,4] is a matrix
disp('MESSAGE: 1234 is a string') % displays a string
```

```
        1234
     1     2     3     4
     1     2
     3     4
MESSAGE: 1234 is a string
```

- **sprintf:** δημιουργεί string αποτελέσματα από οποιαδήποτε δεδομένα, αριθμητικά ή κειμένου

```
 str1 = sprintf('The value of pi equals to %6.3f',pi)
 str2 = sprintf('The integer part of e = exp(1) equal to %6.0f',exp(1))
 % NB: the total width of the printed number is 6 (empty characters are
added)
```

```
  str1 =
       The value of pi equals to  3.142
  str2 =
       The integer part of e = exp(1) equal to      3
```

- **fprintf:** δίνει ως αποτέλεσμα ένα μορφοποιημένο μήνυμα και αριθμούς στο τερματικό του χρήστη ή στο αρχείο

```
fprintf('The value of pi equals to %6.3f\n',pi);
fprintf('The integer part of e = exp(1) equal to %6.0f\n',exp(1));
```

```
The value of pi equals to  3.142
```

```
The integer part of e = exp(1) equal to      3
```

- **formats (the same as in C)**
  - **"%10.2f"** – μορφοποίηση σταθερού σημείου
  - **"%10.2e"** – μορφοποίηση κινητής υποδιαστολής
  - **"%10.2g"** – η καλύτερη, σταθερή ή κινητή μορφή υποδιαστολής
  - **"%10.0f"** – μορφοποίηση σταθερού σημείου για έναν στρογγυλοποιημένο ακέραιο μέρος ενός αριθμού
  - **"%d"** – δείχνει έναν ακέραιο
  - **"%s"** – δείχνει μία συμβολοσειρά

```
x = 123456789.987654321;
x1 = sprintf('%5.2f',x) % x1 displays 2 digits of the fractional
part
          % NB: the total width of the printed value exceeds 5,
    % since the integer part of x has more characters than the format
x2 = sprintf('%5.2e',x) % x2 displays 2 digits after the period
x3 = sprintf('%5.2g',x) % x3 takes the floating point format as best
x4 = sprintf('%5.0f',x) % x4 displays rounded integer number
```

```
 x1 =
      123456789.99
 x2 =
      1.23e+008
 x3 =
      1.2e+008
 x4 =
      123456790
```

- **Ειδικοί Χαρακτήρες:**
  - **'\n':** *χαρακτήρας για αλλαγή σειράς*
  - **'\t':** χαρακτήρας *tab(Οριζοντίου προκαθορισμένου διαστήματος)*
  - **'\r':** *χαρακτήρας επιστροφής*
  - **'\\':** *Εμφάνιση της ανάποδης πλαγίας καθέτου*
  - **'%%':** *χαρακτήρας τις εκατό*

```
fprintf('here is an example\n of a ver\ty fu\t\tnny t\t\t\text!');
```

```
here is an example
 of a ver    y fu        nny t             ext!
```

```
x = 1; y = 12; z = 123; w = 1234;
fprintf('x = %4d,\ty = %4d\nz = %4d,\tw = %4d\n',x,y,z,w);
     % NB: useful formatting of data output
```

```
x =    1,    y =   12
z =  123,    w = 1234
```

**<u>Πως διαβάζουμε ένα ερχείο δεδομένων και πως γράφουμε σ'αυτό:</u>**
**fileID = fopen('filename','w'):** ανοίγει το αρχείο *"filename"* για λειτουργίες γραφής
**fileID:** ένας ακέραιος, ονομάζεται αναγνωριστικό/ταυτότητα αρχείου (είναι μοναδικός στον υπάρχοντα χώρο εργασίας)

**permissions:**
**'r'** - ανάγνωση
**'w'** - γραφή (το δημιουργούμε αν χρειαστεί)
**'a'** - προσάρτηση (το δημιουργούμε αν χρειαστεί**)**
**'r+'** – γραφή και ανάγνωση (δεν το δημιουργούμε)
**'w+'** - αποκοπή ή δημιουργία για ανάγνωση και γραφή
**'a+'** - ανάγνωση και προσάρτηση (το δημιουργούμε αν χρειαστεί**)**
**fclose('fileID'):** κλείνει το αρχείο με το αναγνωριστικό "*fileID*" μετά από όλες τις λειτουργίες
**fprintf(fileID,'textname'):** γράφει το κείμενο "*textname*" μέσα στο αρχείο με το αναγνωριστικό *"fileID*"
**fwrite(fileID,variablename):** γράφει τη μεταβλητή "*variablename*" εντός του αρχείου
**fscanf, fread:** διαβάζει οδηγίες MATLAB για λειτουργίες ανάγνωσης

**<u>Πως αποθηκεύουμε και φορτώνουμε δεδομένα στον χώρο εργασίας:</u>**

**save:** αποθηκεύει τον υπάρχοντα χώρο εργασίας MATLAB σε διπλής ακρίβειας δυαδικό

- **save** – όλες οι υπάρχοντες μεταβλητές αποθηκεύονται στο αρχείο "*matlab.mat*"
- **save filename** – όλες οι υπάρχοντες μεταβλητές αποθηκεύονται στο αρχείο "*filename.mat*"
- **save filename <variablename>** – μόνο η λίστα με τις μεταβλητές στο "*variablename*" αποθηκεύετε στο αρχείο "*filename.mat*" (NB: στη λίστα αυτή δεν γίνετε διαχωρισμός μεταβλητών με κόμματα)
- **save filename <variablename> - ascii :** τα δεδομένα αποθηκεύονται σε μορφή ASCII, το αρχείο μπορούμε να το δούμε από πηγές επεξεργασίας.

- **load:** ανακτά όλες τις αποθηκευμένες μεταβλητές από το αρχείο MATLAB και τις μεταφέρει στον υπάρχοντα χώρο εργασίας
- **load** – φορτώνει όλες τις αποθηκευμένες μεταβλητές από το αρχείο"*matlab.mat*"
- **load filename** – φορτώνει όλες τις αποθηκευμένες μεταβλητές από το αρχείο "*filename.mat*" (NB: το αρχείο θα πρέπει να υπάρχει είτε στο ίδιο φάκελο ή στο ισχύων μονοπάτι)
- **load filename – ascii :** φορτώναι όλα τα δεδομένα από το ερχείο "*filename.mat"* σε μία μεταβλητή που ονομάζεται "*filename*" (NB: το αρχείο "*filename.mat"* θα πρέπει να περιέχει δεδομένα σε μία από τις παρακάτω μορφές)
  - ένας απλός αριθμός
  - διάνυσμα γραμμής
  - διάνυσμα στήλης
  - ένας πίνακας

**Σημείωση:** Αν πολλές μεταβλητές από διαφορετικές μορφές δεδομένων πρόκειται να φορτωθούν, θα πρέπει να ετοιμαστούν σε διαφορετικά ASCII αρχεία δεδομένων! Αυτό το πρόβλημα δεν το έχουμε στη δυαδική μορφή: όλες οι αποθηκευμένες μεταβλητές ανακτούνται ξεχωριστά και αποθηκεύονται σε διαφορετικές μεταβλητές που διατηρούν τα ονόματα των αποθηκευμένων μεταβλητών!

```
x = [ 1, 2, 3, 4 ]; y = [ -1 ; -2 ];
save data1 x y
clear all;
load data1;
x
y
```

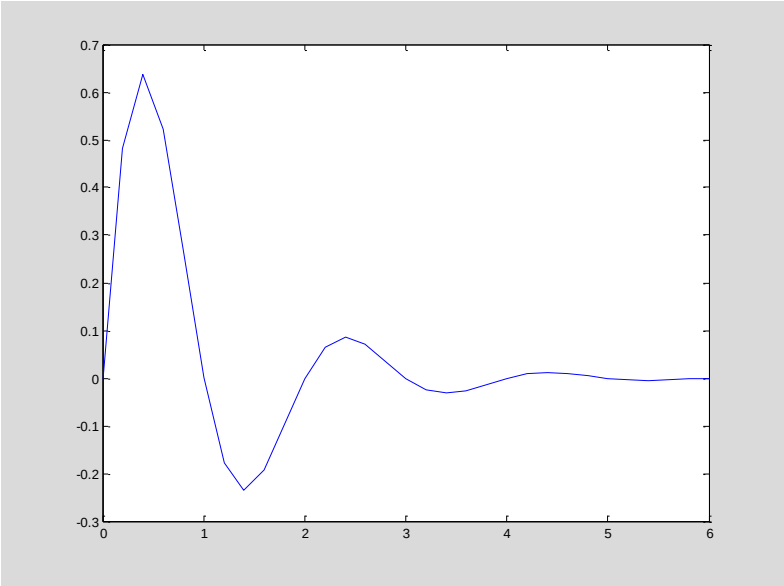
```
x =
     1     2     3     4
y =
    -1
    -2
```

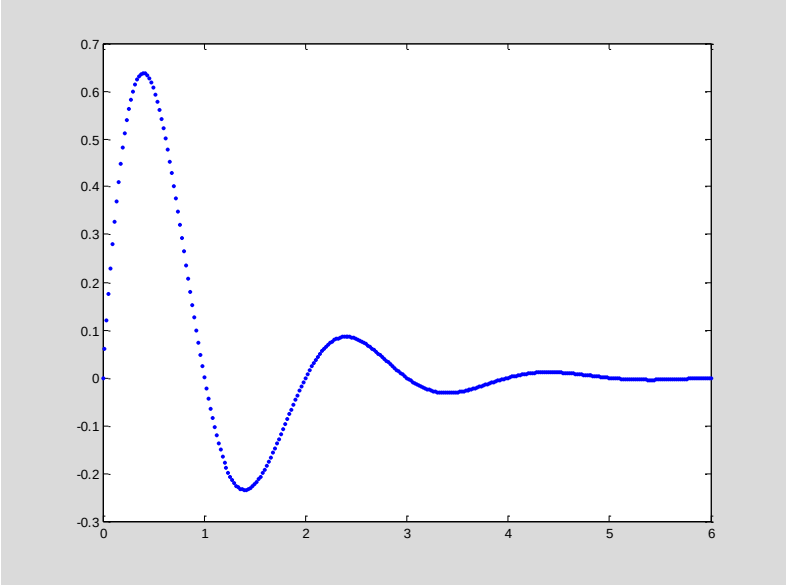
## 7. Σχεδιαστικές διαδικασίες και MATLAB γραφικά

**Σχεδιασμός γραφημάτων συναρτήσεων μίας μεταβλητής:**

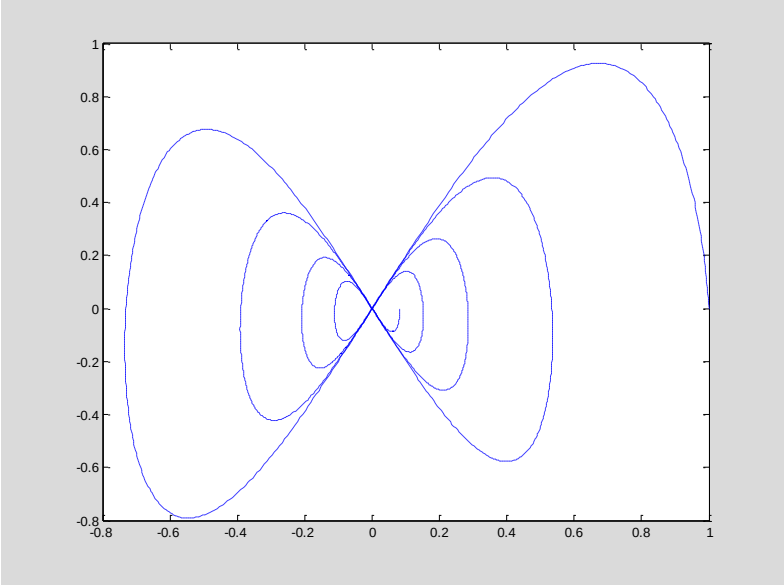
- **plot:** μας δίνει ως αποτέλεσμα ένα διάνυσμα έναντι ενός άλλου διανύσματος, τα διανύσματα πρέπει να είναι της ίδιας δομής και του ίδιου μήκους

```
% plotting of explicit functions of one variables
   x = 0 : 0.2 : 6; % bad resolution
   y = exp(-x).*sin(pi*x); % y has the same structure and the length as the vector x
   plot(x,y); % the graph is connected by solid line
   x = 0 : 0.02 : 6; % better resolution
   y = exp(-x).*sin(pi*x);
   plot(x,y,'.'); % the graph consists of data points
```



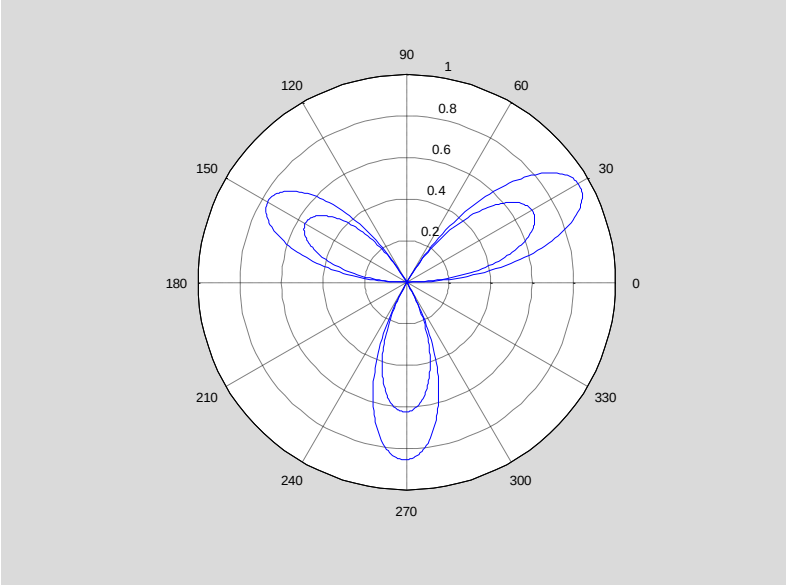



```
% plotting of parametric curves (complex numbers)
      t = 0 : 0.01 : 8*pi;
      z = exp(-0.1*t).*(cos(t) + i*sin(2*t));
      plot(real(z),imag(z),'--'); % the graph is plotted by dashed line
```



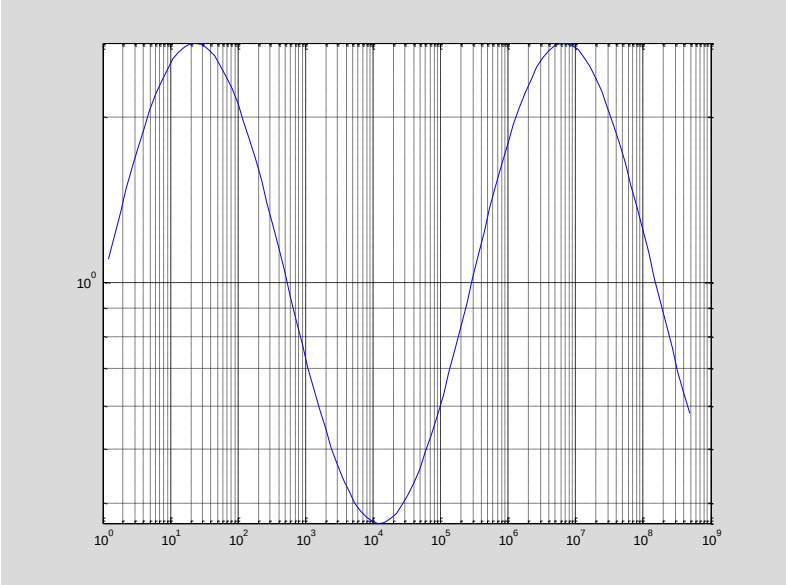

- **polar:** σχεδιάζει ένα γράφημα συνάρτησης με πολικές συντεταγμένες (*ακτίνα με γωνία)*

```
t = 0 : 0.01 : 2*pi;
y = sin(3*t).*exp(-0.1*t);
polar(t,y); grid on;
```




- **loglog:** σχεδιάζει ένα γράφημα συνάρτησης με λογαριθμικές συντεταγμένες *(log(y) με log(x))*

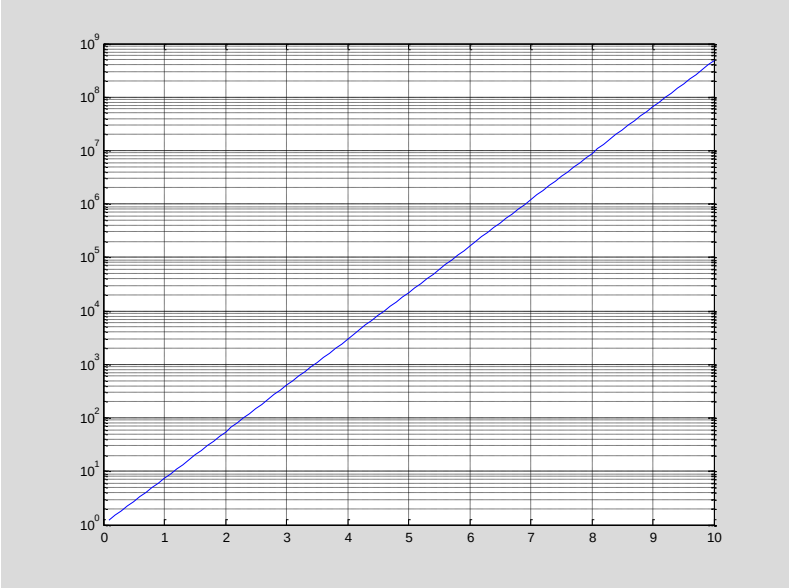
```
t = 0.1 : 0.1 : 10;
x = exp(2*t);
y = exp(sin(t));
loglog(x,y); grid on;
```




- **semilogx,semilogy:** σχεδιάζει ένα γράφημα συνάρτησης σχεδιάζει ένα γράφημα συνάρτησης με ημιλογαριθμικές συντεταγμένες *(log(y) με x ή y με log(x))*

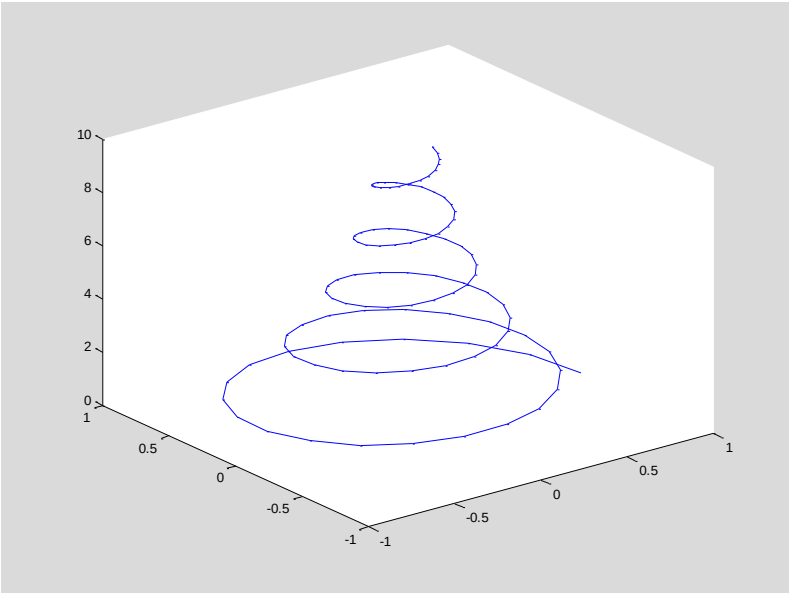
```
t = 0.1 : 0.1 : 10;
```

```
semilogy(t,exp(2*t)); grid on;
```




- **plot3:** σχεδιάζει ένα γράφημα μίας καμπύλης σε τρισδιάστατο χώρο

```
t = 0 : 0.1 : 10;
x = exp(-0.2*t).*cos(pi*t);
y = exp(-0.2*t).*sin(pi*t); plot3(x,y,t);
```




**<u>Αλλαγές στο σχεδιασμό γραφημάτων:</u>**

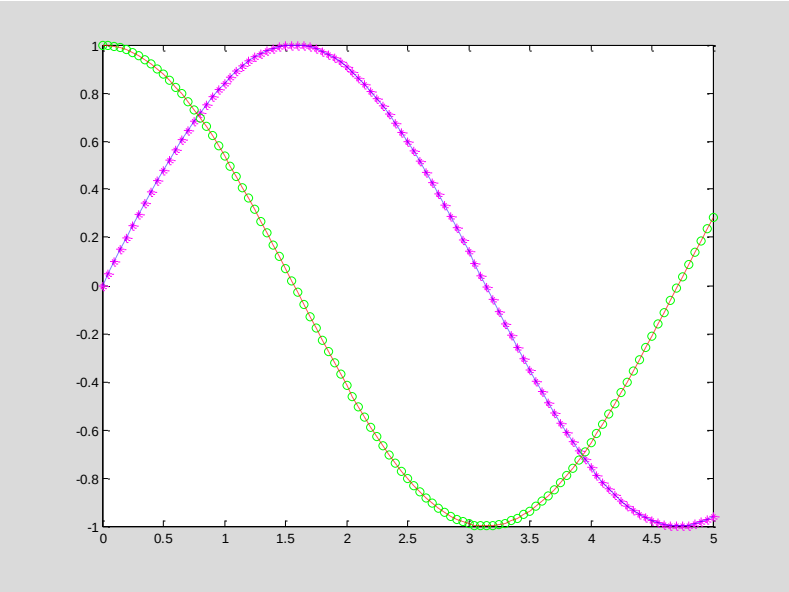
- **σημεία δεδομένων (**δεν υπάρχει ένωση με γραμμές):
  - **'.'** – τελεία, υποδιαστολή
  - **'+'** – συν

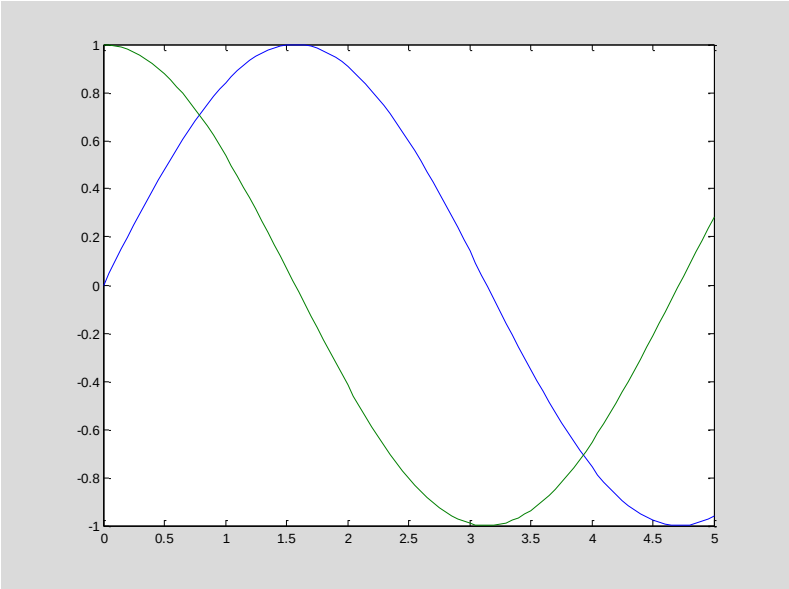
- '*****' – αστέρι
- '**d**' – diamond
- '**o**' – κύκλος
- '**p**' – πεντάγωνο
- '**s**' – τετράγωνο
- '^' – τρίγωνο
- '**x**' – σημείο x

- **Τύπος σειρών** (δεν υπάρχουν μεμονωμένα σημεία):
  - '**-**' – συνεχόμενη γραμμή (αυτόματο)
  - '**--**' – διακεκομμένη γραμμή
  - '**:**' – γραμμή από τελείες
  - '**-.**' – διακεκομμένη γραμμή με τελείες

- **Χρώμα σειρών:**
  - '**r**' – κόκκινο
  - '**y**' – κίτρινο
  - '**m**' – μοβ
  - '**c**' – γαλάζιο
  - '**g**' – πράσινο
  - '**b**' – μπλε
  - '**w**' – άσπρο
  - '**k**' – μαύρο

```
% combination of data marks, line types, and line colors
 x = 0 : 0.05 : 5;
 y = sin(x);
 z = cos(x);
 plot(x,y,'*m',x,y,':b',x,z,'og',x,z,'-.r');
     % check the output and compare with specification of graphs
 plot(x,y,x,z);
  % the line types and colors can be automatically selected for
  each curve by default
```






- **fill:** σχεδιάζει γεμισμένα πολύγωνα με , μπορούμε να την επικαλεστούμε αντί του **"plot"**

- **ετικέτες και τίτλοι:**
  - **xlabel('x-axis')** – Η συμβολοσειρά *'x-axis'* γράφετε ως ετικέτα για τις συντεταγμένες του x
  - **ylabel('y-axis')** – Η συμβολοσειρά *'y-axis'* γράφετε ως ετικέτα για τις συντεταγμένες του y.
  - **title('The graph of y = f(x)')** – Η συμβολοσειρά *'The graph of y = f(x)'* γράφεται ως τίτλος του γραφήματος.
  - **legend('y = f(x)')** – Η συμβολοσειρά *'y = f(x)'* γράφεται σε ένα μικρό υποπαράθυρο και μας δείχνει το είδος γραμμής των δεδομένων σημείων που χρησιμοποιούνται για το σχεδιασμό του γραφήματος της συνάρτησης y = f(x)

- **μέγεθος γραμματοσειράς και πάχος γραμμής:**
  - **plot(x,y,'—','linewidth',3)** – το πάχος της προεπιλεγμένης γραμμής είναι 0.5
  - **xlabel('x-axis','fontsize',14)** – το μέγεθος της προεπιλεγμένης γραμματοσειράς είναι 12pt

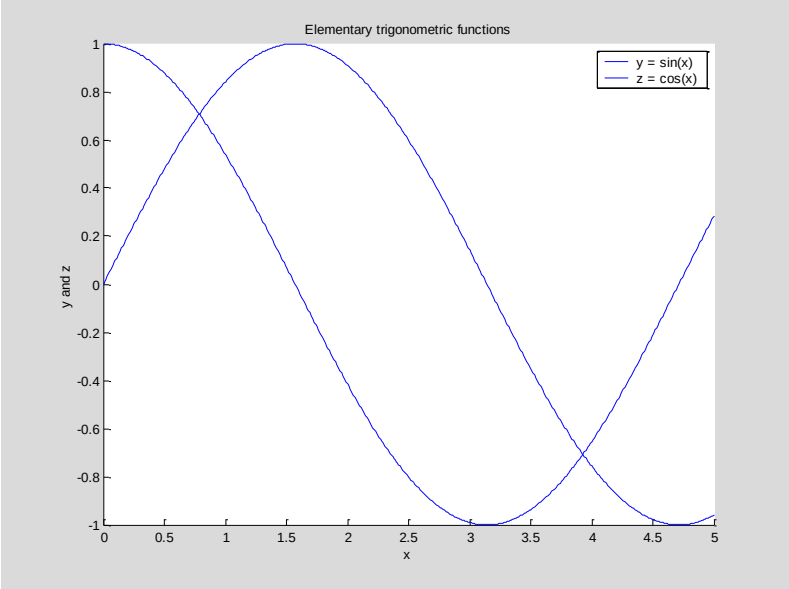
- **κείμενα:**
  - **text(x,y,'TextBody')** – Η συμβολοσειρά *"TextBody"* αποτυπώνεται στο γράφημα, ξεκινώντας από τις απόλυτες συντεταγμένες*(x,y)* pixels(εικονοστοιχείων)
  - **num2str, int2str** – μετατροπή ενός αριθμού *σε* συμβολοσειρά και ενός ακεραίου *σε* συμβολοσειρά , αντίστοιχα
  - **η δομή LATEX ενός κειμένου:**
    - Ελληνικά σύμβολα: \alpha, \beta, \gamma, \epsilon
    - εκθέτες: x^2, y^{-2}
    - δείκτες: x_n, y_{i,j}

```
        strPi = num2str(pi);
    strText = ['The area of the circle, A = ',strPi,'*r^2']
         % concatenation of three strings into the text
    text(5,10,strText,'fontsize',14,'Color','r');
         % setting the properties of the text on a current window

        strText =
    The area of the circle, A = 3.1416*r^2
```

- **άξονες και πλέγματα**
  - **axis([x_min,x_max,y_min,y_max])** – ορίζει τις μέγιστες και ελάχιστες τιμές για τις συντεταγμένες του γραφήματος (*x_min* <= *x* <= *x_max; y_min* <= *y* <= *y_max*)
    - η εντολή "*axis"* πρέπει να χρησιμοποιείται μετά την εντολή *"plot"*
    - οποιαδήποτε γραμμή εκτός ορίων κόβεται
  - **axis off** – αφαιρούνται οι άξονες συντεταγμένων και τα σημεία διαμέρισης του άξονα (tic marks) (*προεπιλογή: axis on*)
  - **axis square** – το γράφημα ξανασχηματίζεται σε τετραγωνική μορφή
  - **grid on** – ένα πλέγμα προστίθεται στο γράφημα (*προεπιλογή: grid off)*
- **several plots on the same graph:**
  - **hold on –** η εντολή επιθέτει αρκετά σημεία στο ίδιο γράφημα, ακόμη κι αν εκτελείται ένα άλλο τμήμα/αρχείο κώδικα *(προεπιλογή: hold off)*
  - **hold off** – προτείνεται η χρήση αυτής της εντολής στο τέλος ενός αρχείου κώδικα, αν η εντολή "hold on" έχει ποτέ χρησιμοποιηθεί σε αυτό το αρχείο κώδικα

```
When several graphs are plotted, it is better to define the
limits for coordinates in advance.  Otherwise, the limits will be
the same as for the first curve. All specifications such as
"axis" can be hold with the command "hold on" before the graphs
are plotted on the figure.
```

```
      x = 0 : 0.01 : 5; y = sin(x); z = cos(x);
      axis([0,5,-1,1]); hold on;
      plot(x,y); hold on; plot(x,z); hold off;
      legend('y = sin(x)','z = cos(x)');
    % the number of arguments correspond to the number of functions
  plotted
           xlabel('x'); ylabel('y and z'); title('Elementary
trigonometric functions');
```




**Λειτουργίες και γραφήματα:**

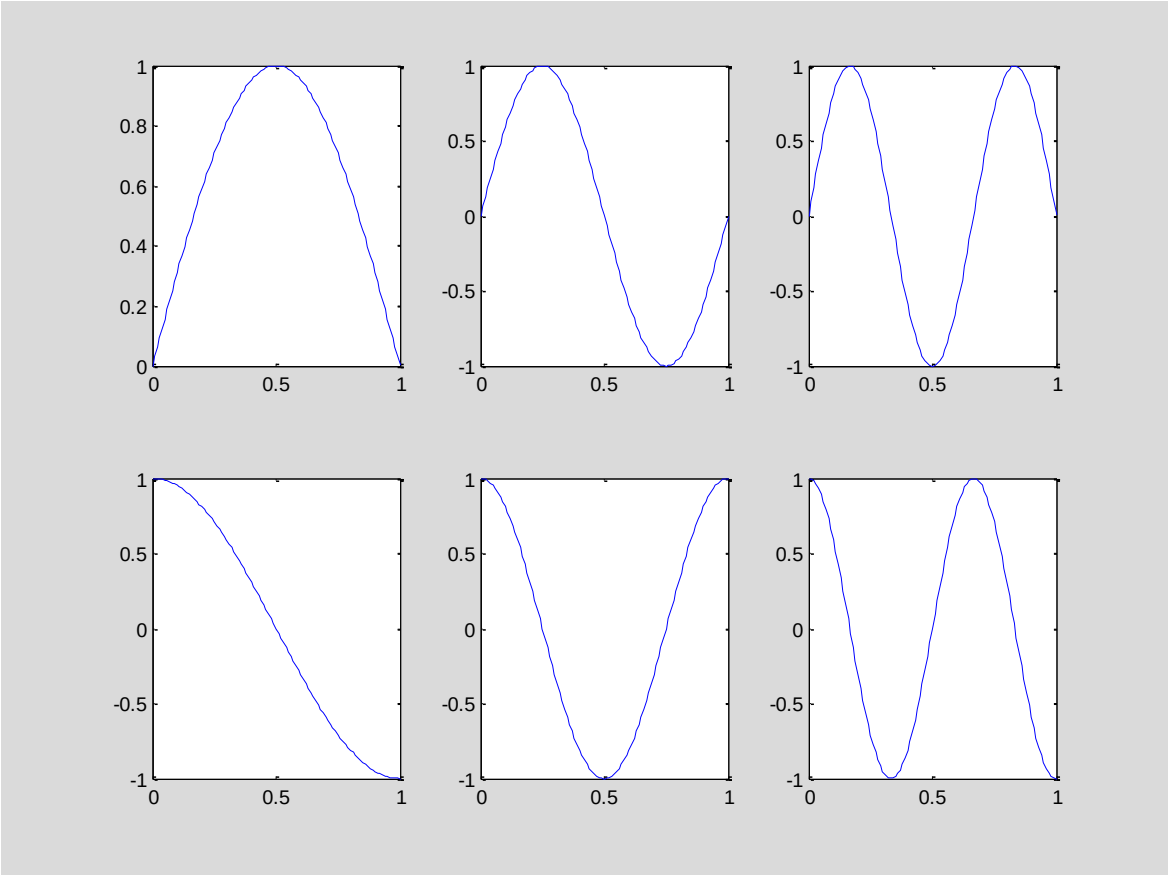
- **figure:** ανοίγει ένα νέο παράθυρο γραφήματος το οποίο έχει τον επόμενο αριθμό από το προηγούμενο παράθυρο
- **figure(n):** φτιάχνει ένα ήδη υπάρχων παράθυρο γραφήματος, (**n**) το υπάρχων παράθυρο; αν το παράθυρο (**n**) δεν υπάρχει, η εντολή ανοίγει ένα νέο παράθυρο γραφήματος αριθμημένο με (**n**) **(NB:** όλες οι εντολές γραφήματος εφαρμόζονται στο υπάρχων παράθυρο)
- **pause:** καθυστερεί την εκτέλεση του κώδικα και περιμένει από τον χρήστη να πατήσει κάποιο κλειδί
- **pause(s):** καθυστερεί την εκτέλεση του κώδικα για ***s*** δευτερόλεπτα
- **close(n):** κλείνει το παράθυρο γραφήματος (**n**)
- **close all**: κλείνει όλα τα παράθυρα γραφημάτων
- **clf**: καθαρίζει το παράθυρο γραφήματος
- **cla**: καθαρίζει τις δεδομένες καμπύλες και ξανασχεδιάζει τους άξονες
- **figure('Position',[pix,piy,pwx,pwy])**: ορίζει το μέγεθος και το σχήμα του υπάρχοντος παραθύρου
    - **pix,piy** – οι οριζόντιες και κάθετες συντεταγμένες από την αριστερή κάτω γωνία του παραθύρου
    - **pwx,pwy** – ο αριθμός των pixels(εικονοστοιχείων) στο πλάτος και το ύψος του παραθύρου
    - **default**: γράφημα ('Position',[232 258 560 420])
- **get(gcf):** δείχνει τις ιδιότητες του υπάρχοντος γραφήματος, όπως μέγεθος, θέση, χάρτη χρώματος και πολλά άλλα
- **set(gcf,'PropertyName',PropertyArray):** αλλάζει την ιδιότητα "*PropertyName*" του υπάρχοντος παραθύρου σύμφωνα με τα δεδομένα του *PropertyArray*

```
set(gcf,'DefaultTextColor','blue');
set(gcf,'PaperPosition',[0.25 2.5 4 3]);
```

- **Property Editor:** σε όλες οι ιδιότητες του υπάρχοντος παραθύρου μπορούν να γίνουν επεξεργασίες διαδραστικά μετά το σχεδιασμό του γραφήματος
- **print:** στέλνει το υπάρχον γράφημα στον προεπιλεγμένο εκτυπωτή

- **print –dps filename:** αποθηκεύει το υπάρχον γράφημα σε ένα αρχείο PostScript *"filename"*
- **print –djpeg filename:** αποθηκεύει το υπάρχον γράφημα σε ένα αρχείο JPEG "*filename*"
- **Window Editor:** περισσότερες επιλογές για τη διάδοση της έρευνας διατίθενται διαδραστικά
- **subplot(m,n,k):** διαιρεί το παράθυρο σε έναν πίνακα m επί n από υποπαράθυρα και τοποθετεί το υπάρχων σχέδιο στο k-th υποπαράθυρο (*k* <= *m***n*)

```
x = 0 : 0.01 : 1;
subplot(2,3,1), plot(x,sin(pi*x));
subplot(2,3,2), plot(x,sin(2*pi*x));
subplot(2,3,3), plot(x,sin(3*pi*x));
subplot(2,3,4), plot(x,cos(pi*x));
subplot(2,3,5), plot(x,cos(2*pi*x));
subplot(2,3,6), plot(x,cos(3*pi*x));
```



## 8. Σφάλματα υπολογισμών στην MATLAB

**Είδη Σφαλμάτων:**

**Σφάλματα στρογγυλοποίησης** (εξαιτίας περιορισμένης αριθμητικής ακρίβειας)
**Σφάλματα αποκοπής (**εξαιτίας της διακριτοποίησης και της αποκοπής σειρών)

**Σφάλματα πειραματισμού (**εξαιτίας των ανακριβειών στις τιμές των δεδομένων)

**Ορισμός Σφαλμάτων:**

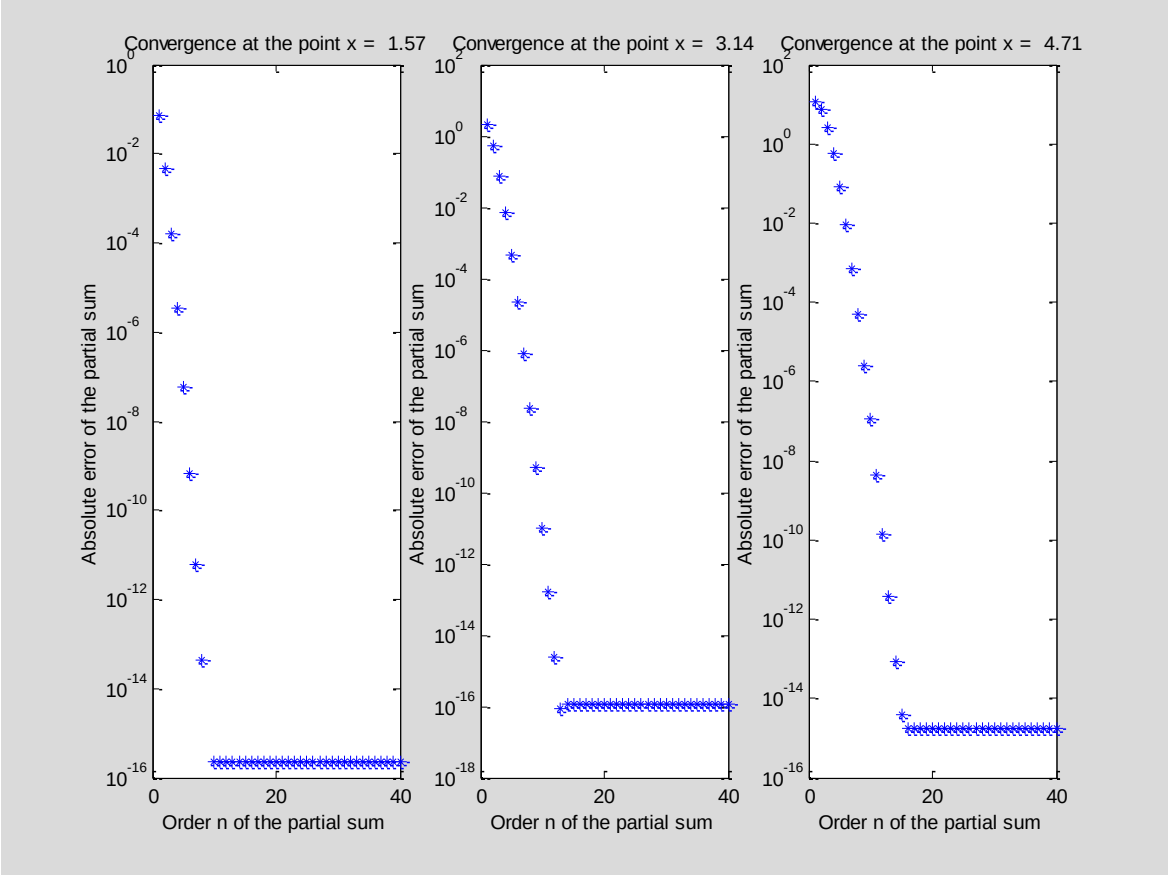
Ας υποθέσουμε ότι: $x_{ex}$ - η ακριβής τιμή του $x$; $x_{ap}$ – η κατά προσέγγιση τιμή που δίνει ο υπολογιστής στο $x$.
**Απόλυτο σφάλμα:** $E_{abs} = | x_{ap} - x_{ex} |$
**Σχετικό Σφάλμα:** $E_{rel} = | x_{ap} - x_{ex} | / | x_{ex} |$

$E_{rel} \sim 10^{-4}$, $| x_{ex} | \sim 1$, τότε $x_{ap}$ έχει σφάλμα στο τέταρτο ψηφίο μετά την τελεία
$E_{rel} \sim 10^{-4}$, $| x_{ex} | \sim 10^{-5}$, then $x_{ap}$ έχει σφάλμα στο ένατο ψηφίο μετά την τελεία
$E_{rel} \sim 10^{-4}$, $| x_{ex} | \sim 10^{5}$, then $x_{ap}$ έχει σφάλμα στο πρώτο ψηφίο μετά την τελεία

**Λάθη αποκοπής σε αντίθεση λαθών στρογγυλοποίησης:**

Ένας υπολογιστής μπορεί να αντιπροσωπεύει μόνο έναν περιορισμένο άθροισμα ή περιορισμένη διαφορά προσέγγισης. Επομένως, η αναπαράσταση του υπολογιστεί πάντα έχει λάθη αποκοπής. Αν προσθέσουμε περισσότερους όρους στο πεπερασμένο άθροισμα ή στις προσεγγίσεις πεπερασμένου αθροίσματος μπορεί να μειωθεί το λάθος αποκοπής αν η σύγκλιση είναι ομοιόμορφη. Ωστόσο, η σύγκλιση δεν βελτιώνεται αφού το λάθος αποκοπής γίνει συγκρίσιμο με το λάθος στρογγυλοποίησης που παράγεται από την κινητή υποδιαστολή. Οι αριθμητικοί υπολογισμοί πάντα βασίζονται στην ανακρίβεια του αριθμητικού συστήματος του υπολογιστή.

```
      % Taylor series for y = sin(x):
  % sin(x) = ∑ (-1)^n*x^(2n+1) / (2n+1)! = x – x^3/3! + x^5/5 - …
  % The Taylor series converges to y = sin(x) for any |x| < inf
  % Computations of sin(x) for n-finite partial sums:
     x = [pi/2,pi,3*pi/2];
     S = x; Term = S;
     y_ex = sin(x);
     for n = 1 : 40
                 Term = -Term.*x.*x/(2*n*(2*n+1));
                 S = S + Term;
                 E_abs(n,:) = abs(S-y_ex);
end
  for k = 1:length(x)
           subplot(1,length(x),k);
           semilogy(E_abs(:,k),'*');
           xlabel('Order n of the partial sum');
           ylabel('Absolute error of the partial sum');
           title(sprintf('Convergence at the point x = %5.2f',x(k)));
  end
```




**Σφάλματα MATLAB:**

- **realmin:** ο μικρότερος θετικός αριθμός με κινητή υποδιαστολή
- **realmax:** ο μέγιστος θετικός αριθμός με κινητή υποδιαστολή

```
realmin, realmax
 % MATLAB gives wider range of numbers compared to double precision
  on a standard workstation

ans =
  2.2251e-308
ans =
  1.7977e+308
```

- **eps:** ακρίβεια μηχανής, η απόσταση από το 1.0 μέχρι την επόμενη τιμή με κινητή υποδιαστολή μεγαλύτερη από το 1.0; το σύστημα του υπολογιστή δεν μπορεί να εκφράσει οποιαδήποτε τιμή μεταξύ [1,1+eps] και στρογγυλοποιεί την τιμή 1 ή 1+eps
- **0.5*eps**: το μέγιστο σεχτικό σφάλμα στρογγυλοποίησης το οποίο σχετίζεται με την αριθμητιμή κινητής υποδιαστολής

```
  eps % MATLAB precision is about the same as the double precision
on a typical workstation
```

```
ans =
  2.2204e-016
```

**Παραδείγματα σφαλμάτων στρογγυλοποίησης:**

Τα σφάλματα στρογγυλοποίησης βρίσκονται σε κάθε αριθμητική λειτουργία κινητής υποδιαστολής, όπως την πρόσθεση, την αφαίρεση, τον πολλαπλασιασμό και την διαίρεση.

- **Υποχείλιση κινητής υποδιαστολής (**x_min is the mashine zero: *if 0 <= x < x_min, then x = 0* **)**

```
 x = 1; q = 0;
while ( x ~= 0 )
       x = x/2;
       q = q + 1;
end
 q % the power for the smallest positive number, when 1 / 2^q = 0
xMin = 1 / 2^q % mashine zero in floating point arithmetic
```

```
q =        1075
xMin =     0
```

- **Υπερχείλυση κινητής υποδιαστολής (**x_max is the mashine infinity: *if x > x_max, then x = inf* **)**

```
 x = 1; q = 0;
while ( x ~= inf )
       x = 2*x;
       q = q + 1;
end
 q % the power for the largest positive number, when 2^q = inf
xMax = 2^q % mashine infinity
x1 = 1/inf % it must be zero
x2 = i*inf % it does not have sense, i.e. NaN
```

```
q =      1024
xMax =   Inf
x1 =     0
x2 =     NaN +    Infi
```

- **Ακρίβεια κινητής υποδιαστολής(** eps is the mashine precision: *if 1 < x < 1 + eps, then x = 1* **)**

```
 x = 1; q = 0; y = 1; z = x + y;
while ( x ~= z )
       y = y/2;
       q = q + 1;
       z = x + y;
end
 q % the power for the smallest positive number, when 1 + 1 / 2^q
= 1
 xEps = 1 + 1 / 2^q % it must be one
```

```
q =      53
```

```
xEps =  1
```

## 9. Δύο διαστάσεων ακολουθίες και πίνακες

Στη γραμμική άλγεβρα, ένας πίνακας είναι μιαη παραλληλόγραμμη διάταξη στοιχείων (αριθμών). Αν ένας πίνακας $A = [\ a_{i,j}\ ]$ έχει $m$ σειρές και $n$ στήλες, τότε έχει $m$-επί-$n$ στοιχεία $a_{i,j}$ όπου ο πρώτος δείκτης $i$ ισχύει για τις σειρές του $A$ και ο δεύτερος δείκτης $j$ ισχύει για τις στήλες του $A$. Οι πίνακες στη MATLAB είναι μία ευθεία και βολική πραγματοποίηση των πινάκων της γραμμικής άλγεβρας.

```
   A = [ 0.1 0.2 0.3 0.4 ; -0.1,-0.2,-0.3,-0.4 ; 0 1 0 1 ]
% Notice that rows in MATLAB matrices are separated by the sign ";"
% Notice that elements at each row can be separated by spaces and
commas

   A =
       0.1000    0.2000    0.3000    0.4000
      -0.1000   -0.2000   -0.3000   -0.4000
            0    1.0000         0    1.0000

A1 = [ 0.1 0.2 0.3 0.4 ; 0 1 0 1 0] ;
      % Number of elements in each row must be identical

??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.
```

Η "**:**" αντιπροσωπεύει στοιχεία μίας σειράς ή μιας στήλης ενός πίνακα της MATLAB.

```
   A(:,1), A(:,4) % columns of A are column-vectors

   ans =
            0.1000
                 -0.1000
                  0
ans =
                 0.4000
            -0.4000
      1.0000

   A(1,:), A(3,:) % rows of A are row-vectors

  ans =
            0.1000    0.2000    0.3000    0.4000
ans =
                  0     1     0     1

      A(1,2), A(3,3) % individual elements of A are scalar variables

   ans =
```

```
            0.2000
   ans =
                0
```

**Λειτουργίες πινάκων:**

**Όλες οι λειτουργίες διανυσμάτων,** όπως οι κατά σημεία προσθέσεις, αφαιρέσεις, πολλαπλασιασμοί και διαιρέσεις και οι πίνακες με εκθέτη ίδιου μεγέθους**.**

```
      A = [ 1 2 3 4 ; -1,-2,-3,-4 ];
B = [ 4 3 2 1 ; -4,-3,-2,-1 ];
C1 = A + B, C2 = A.*B, C3 = A./B, C4 = A.^B
% Pointwise matrix operations are preferable compared to double loops
in i,j
% because of (i) clear mathematical notations, (ii) computational
efficiency
```

```
C1 =
     5     5     5     5
    -5    -5    -5    -5
C2 =
     4     6     6     4
     4     6     6     4
C3 =
    0.2500    0.6667    1.5000    4.0000
    0.2500    0.6667    1.5000    4.0000
C4 =
    1.0000    8.0000    9.0000    4.0000
    1.0000   -0.1250    0.1111   -0.2500
```

**size**: αποδίδει αριθμούς γραμμών και στηλών

```
  [nRow,nCol] = size(A)
```

```
nRow =     2
nCol =     4
```

- **transpose: *A'*,** αποδίδει έναν ανάστροφο πίνακα $A^T$

```
  B = A', size(B)
```

```
 B =  1    -1
      2    -2
      3    -3
     4    -4
 ans =
      4     2
```

- **βασικοί πίνακες:**

- **zeros:** αποδίδει έναν μηδενικό πίνακα του οποίου όλα τα στοιχεία είναι μηδέν

```
zeros(2,6) % a null 2-by-6 matrix
 zeros(3) % a null 3-by-3 matrix
```

```
ans =       0     0     0     0     0     0
            0     0     0     0     0     0
ans =       0     0     0
          0     0     0
            0     0     0
```

- **ones:** απόδίδει έναν πίνακα ου οποίου όλα τα στοιχεία είναι ένα

```
  ones(2,6) % a 2-by-6 matrix of ones
     ones(3) % a 3-by-3 matrix of ones
```

```
ans =
     1     1     1     1     1     1
     1     1     1     1     1     1
ans =
     1     1     1
     1     1     1
     1     1     1
```

- **eye:** αποδίδει έναν μοναδιαίο πίνακα ( διαγώνιο πίνακα)

```
eye(2,6), eye(3)
```

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

- **rand, randn:** αποδίδει έναν τυχαίο πίνακα του οποίου όλατα στοιχεία είναι τυχαίοι αριθμοί

```
  rand(2,6) % random numbers are uniformly distributed in the interval
[0,1]
  randn(3) % random numbers are normally distributed with mean zero and
variance one
```

```
ans =
    0.9501    0.6068    0.8913    0.4565    0.8214    0.6154
    0.2311    0.4860    0.7621    0.0185    0.4447    0.7919
ans =
   -0.4326    0.2877    1.1892
   -1.6656   -1.1465   -0.0376
    0.1253    1.1909    0.3273
```

- **ειδικοί πίνακες:**

- **hilb:** αποδίδει τον πίνακα Hilbert με στοιχεία $a_{i,j} = 1/(i+j-1)$

```
C1 = hilb(6) % Hilbert matrices are examples of ill-conditioned
matrices
```

```
C1 =   1.0000    0.5000    0.3333    0.2500    0.2000
            0.5000    0.3333    0.2500    0.2000    0.1667
            0.3333    0.2500    0.2000    0.1667    0.1429
            0.2500    0.2000    0.1667    0.1429    0.1250
            0.2000    0.1667    0.1429    0.1250    0.1111
```

- **pascal:** outputs the Pascal matrix with integer entries, made up from Pascal's triangle: $a_{i,j} = a_{i-1,j} + a_{i,j-1}$

```
C2 = pascal(6)
      % Pascal matrices produce a table of binomial coefficients
      % Coefficients at the k-th anti-diagonal of Pascal matrices
produce
            % coefficients of the binomial expansion: (x + y)^(k-1)
```

```
C2 =         1     1     1     1     1     1
             1     2     3     4     5     6
       1     3     6    10    15    21
       1     4    10    20    35    56
       1     5    15    35    70   126
       1     6    21    56   126   252
```

**vander:** αποδίδει τον πίνακα Van der Monde του οποίου οι στήλες είναι εκθέτες ενός διανύσματος, i.e. $a_{i,j} = v_i^{n-j}$

```
   C3 = vander([1:6])
```

```
C3 =       1           1           1           1           1
1
          32          16           8           4           2
1
         243          81          27           9           3
1
        1024         256          64          16           4
1
        3125         625         125          25           5
1
        7776        1296         216          36           6
1
```

- **toeplitz:** αποδίδει τους πίνακες Toeplitz

```
r = [ 1 2 3 4 5 6 ]; s = [ -1,-2,-3,-4, -5];
C4 = toeplitz(r,s) % r and s are the column and row vectors of C4
C5 = toeplitz(r)   % the symmetric Toeplitz matrix
rInv = r(length(r):-1:2); rInv = [r(length(r)),rInv];
C6 = toeplitz(r,rInv)' % the circulant matrix
```

```
Warning: Column wins diagonal conflict.
> In c:\progra~1\matlab6\toolbox\matlab\elmat\toeplitz.m at line 18
C4 =
     1    -2    -3    -4    -5
     2     1    -2    -3    -4
     3     2     1    -2    -3
     4     3     2     1    -2
     5     4     3     2     1
     6     5     4     3     2
C5 =
     1     2     3     4     5     6
     2     1     2     3     4     5
     3     2     1     2     3     4
     4     3     2     1     2     3
     5     4     3     2     1     2
     6     5     4     3     2     1
Warning: Column wins diagonal conflict.
> In c:\progra~1\matlab6\toolbox\matlab\elmat\toeplitz.m at line 18
C6 =
     1     2     3     4     5     6
     6     1     2     3     4     5
     5     6     1     2     3     4
     4     5     6     1     2     3
     3     4     5     6     1     2
     2     3     4     5     6     1
```

**hadamard, hankel:** πιό εξειδικευμένοι πίνακες
**πολλαπλασιασμοί πίνακα διανύσματος**

```
A = [ 1 2 3 1 ; 0 1 4 2 ; 3 0 2 3 ], x = [ 1; 2; 1; 1; ]

A =  1     2     3     1
     0     1     4     2
     3     0     2     3
x =         1
      2
      1
      1

  % the number of columns of A must be equal to the number of rows of x
      y1 = A*x  % MATLAB matrix multiplication operator

   % a row-oriented loop:
   [n,m] = size(A);
   for k = 1:n,          y2(k) = A(k,:)*x; end
   y2

   % a column-oriented loop
   y3 = zeros(n,1);
   for j = 1 : m,        y3 = y3 + A(:,j)*x(j); end
   y3

y1 =         9
             8
      8
y2 =     9     8     8
y3 =    9
```

```
     8
     8
```

- **πολλαπλασιασμοί πίνακα με πίνακα:**

Αν ο ***A*** είναι ένας πίνακας ***n***-επι-***m*** , ***B*** είναι ένας πίνακας ***m***-επι-***q*** ,τότε ο παλλαπλασιασμός των πινάκων έχει νόημα και ***C = A*B*** είναι ένας πίνακας ***n***-επί-***q*** ***με στοιχεία***: $c_{i,j} = \sum_{k=1}^{m} a_{i,k}\, b_{k,j}$. Το άλλο γινόμενο ***B*A*** ορίζεται αν ***n = q***. Για τετραγωνικούς πίνακες ίδιου μεγέθους, και τα δύο γινόμενα ορίζονται αλλά δεν είναι ίσα σε μια γενική περίπτωση: ***A*B*** $\neq$ ***B*A.***

```
A = [ -2 1 0 ; 1,-2,1 ; 0,1,-2];
B = [1 0 1 ; 0 1 0; 1 0 1];
C1 = A*B, C2 = B*A
```

```
C1 =
    -2     1    -2
     2    -2     2
    -2     1    -2

C2 =
    -2     2    -2
     1    -2     1
    -2     2    -2
```

```
n = 3;
for k = 1 : n
      for j = 1 : n
            C3(k,j) = A(k,:)*B(:,k); % the dot product algorithm
end
end
C3

C4 = zeros(n,n);
for j = 1 : n
      for p = 1 : n
            C4(:,j) = C4(:,j) + A(:,p)*B(k,j); % the column-oriented
algorithm
      end
end
C4

C5 = zeros(n,n);
for p = 1 : n
      C5 = C5 + A(:,k)*B(k,:); % the outer-product oriented algorithm
end
C5
```

```
C3 =
    -2    -2    -2
    -2    -2    -2
    -2    -2    -2
```

```
C4 =
    -1     0    -1
     0     0     0
    -1     0    -1

C5 =
     0     0     0
     3     0     3
    -6     0    -6
```

Σύνθετοι και ταινιωτοί πίνακες και διάφορες

**<u>Δομή ενός ταινιωτού πίνακα:</u>**

**diag(x):** αποδίδει έναν διαγώνιο πίνακα με στοιχεία του ***x*** στην κύρια διαγώνιο
**diag(x,k**): outputs a matrix with elements of ***x*** at the ***k-th*** diagonal, located above the main diagonal if ***k > 0*** and below the main diagonal if ***k < 0***
**diag(A,k):** outputs the vector standing at the ***k-th*** diagonal of a matrix ***A***

```
   x = [ 1 2 3 4 5 ];
      A1 = diag(x), A2 = diag(x,2), A3 = diag(x,-1)
      v = diag(A2,2)

A1 =      1     0     0     0     0
      0     2     0     0     0
      0     0     3     0     0
      0     0     0     4     0
      0     0     0     0     5

A2 =      0     0     1     0     0     0     0
      0     0     0     2     0     0     0
      0     0     0     0     3     0     0
      0     0     0     0     0     4     0
      0     0     0     0     0     0     5
      0     0     0     0     0     0     0
      0     0     0     0     0     0     0

A3 =      0     0     0     0     0     0
      1     0     0     0     0     0
      0     2     0     0     0     0
      0     0     3     0     0     0
      0     0     0     4     0     0
      0     0     0     0     5     0

v =       1
      2
      3
      4
      5
```

**tril(A) :** αποσπά το χαμηλότερο τριγωνικό τμήμα ενός πίνακα
**tril(A,p):** αποσπά το χαμηλότερο τριγωνικό τμήμα ενός πίνακα πάνω και κάτω από τη p-th διαγώνιο
**triu(A) :** αποσπά το υψηλότερο τριγωνικό τμήμα ενός πίνακα

**triu(A,q):** αποσπά το υψηλότερο τριγωνικό τμήμα ενός πίνακα πάνω και κάτω από τη p-th διαγώνιο

```
A = magic(5)
% fun matrix magic(N) is constructed from the integers
% 1 through N^2 with equal row, column, and diagonal sums

B1 = tril(A,1), B2 = triu(A,-1)
```

```
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
B1 =
    17    24     0     0     0
    23     5     7     0     0
     4     6    13    20     0
    10    12    19    21     3
    11    18    25     2     9
B2 =
    17    24     1     8    15
    23     5     7    14    16
     0     6    13    20    22
     0     0    19    21     3
     0     0     0     2     9
```

**Definition:** **A** is a banded matrix that has lower bandwidth **p** and upper bandwidth **q** if **A** has zeros below **(-p)**-th diagonal and above **q-th** diagonal. If **p = 0, A** is an upper triangular matrix. If **q = 0, A** is a lower triangular matrix. If **p = q = 0, A** is a diagonal matrix. If **p = q = 1, A** is a tri-diagonal matrix.

```
% an example of a tridiagonal matrix for a three-point central
difference:
% u''(x) = (u(x+h) - 2*u(x) + u(x-h))/h^2
x = 1 : 7; % the step size is one, h = 1
n = length(x);
A1 = -2*diag(ones(1,n)) + diag(ones(1,n-1),1) + diag(ones(1,n-1),-1)
A2 = -2*eye(n) + triu(tril(ones(n),1),-1)-eye(n)
      % A1 = A2, the same matrix is generated by two different MATLAB
functions
```

```
A1 =
    -2     1     0     0     0     0     0
     1    -2     1     0     0     0     0
     0     1    -2     1     0     0     0
     0     0     1    -2     1     0     0
     0     0     0     1    -2     1     0
     0     0     0     0     1    -2     1
     0     0     0     0     0     1    -2

A2 =
    -2     1     0     0     0     0     0
```

```
     1    -2     1     0     0     0     0
     0     1    -2     1     0     0     0
     0     0     1    -2     1     0     0
     0     0     0     1    -2     1     0
     0     0     0     0     1    -2     1
     0     0     0     0     0     1    -2
```

**Δομές σύνθετων πινάκων:**

***AA*** = $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$***:*** ένας σύνθετος πίνακας που αποτελείται από τέσσερις άλλους πίνακες ***A,B,C,D***

**Περιορισμοί:** ο πίνακας ***B*** πρέπει να έχει τον ίδιο αριθμό σειρών με τον ***A,*** ο πίνακας ***C*** πρέπει να έχει τον ίδιο αριθμό στηλών με τον ***A*** και ο ***D*** πρεπει να έχει τον ίδιο αριθμό σειρών με τον ***C*** και τον ίδιο αριθμό στηλών με τον ***B.***

```
A = [ 1 2 3 4 ; 1 2 3 4; 1 2 3 4; 1 2 3 4] % A is a 4-by-4 matrix
b = [ 0; 0; 1; 1 ] % b is a column vector
c  = b'; % c is a row-vector
d = 100 % d is a scalar
AA = [ A, b; c, d ]  % a block 5-by-5 matrix

A =          1     2     3     4
       1     2     3     4
       1     2     3     4
       1     2     3     4

b =          0
       0
       1
       1

d =    100

AA =         1     2     3     4     0
       1     2     3     4     0
       1     2     3     4     1
       1     2     3     4     1
       0     0     1     1   100

   b1 = [ 0; 0; 1; 1; 1]; % b has now a mis-match in a number of
columns
AA1 =[ A, b1; c, d ]  % an error in building the block matrix

??? Error using ==> horzcat
All matrices on a row in the bracketed expression must have the
 same number of rows.
```

**Τα υποστοιχεία των σύνθετων πινάκων χάνονται μετά την κατασκευή τους**

```
AA(1,1) % returns only the first element of A, not the first block of
AA
AA(1:4,1:4) % returns the first block of AA, i.e. the matrix A
```

```
ans =
     1
ans =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4
```

```
BB = AA(1:2:5,1:3:5)
% the command returns a block of AA consisting of the first, third, and
fifth rows of AA
% and the first and third columns of AA, i.e. the command returns a 3-
by-2 matrix
```

```
BB =   1     4
       1     4
1
```

**Τα υποστοιχεία των σύνθετων πινάκων μπορούν να δημιουργηθούν ως συστοιχία κυψελών**

```
CC = cell(2,2);
C{1,1} = A; C{1,2} = b; C{2,1} = c; C{2,2} = d;
C, C{1:2,1:2}
```

```
C =
    [4x4 double]    [4x1 double]
    [1x4 double]    [       100]
ans =
     1     2     3     4
     1     2     3     4
     1     2     3     4
     1     2     3     4
ans =
     0     0     1     1
ans =
     0
     0
     1
     1
ans =   100
```

**<u>Προηγμένα αντικείμενα δεδομένων:</u>**

**struct:** δημιουργεί μια δομημένη συστοιχία με πεδία και αριθμητικές ή *αλφαριθμητικές* τιμές

```
PA = struct('point','A','x',5,'y',3) % creates point A(5,3)
PB = struct('point','B','x',1,'y',-1) % creates point B(1,-1)
```

```
PA = point: 'A'
        x: 5
        y: 3
PB = point: 'B'
                    x: 1
      y: -1
```

κατά σημεία δημιουργία στοιχείων μίας δομής

```
PC.point = 'C'; PC.x = 9; PC.y = 4; % creates point C(9,4)
PC

PC =point: 'C'
        x: 9
        y: 4
```

δημιουργία μιας συστοιχίας δομών

```
P = struct('point',char('A','B','C'),'x',[5;1;9],'y',[3;-1;4])
% char('A','B','C') creates separate rows for character strings
'A','B', and 'C'
% different structures in P are located at different rows

P =     point: [3x1 char]
        x: [3x1 double]
        y: [3x1 double]
```

ανακτά και χρησιμοποιεί δεδομένα της δομής και των συστοιχιών των δομών

```
 a1 = PA.point, a2 = PC.x, a3 = PC.y
 a4 = P.point(1), a5 = P.x(:)
      r = sqrt(PA.x^2 + PA.y^2) % computes radius in polar coordinates

a1 =     A
a2 =     9
a3 =     4
a4 =     A
a5 =     5
         1
         9
r =    5.8310
```

Οι συναρτήσεις της MATLAB μπορούν να λειτουργήσουν με δομές στις συστοιχίες εισόδου και εξόδου.

**cell:** ένα δοχείο δεδομένων, που μπορεί να περιέχει οποιοδήποτε είδος πληροφορίας (συστοιχία αριθμών, *αλφαριθμητικά*, δομές, ή κελί)

```
C = cell(2,2);
C{1,1} = rand(3); % a 3-by-3 random matrix is in C{1,1} box
C{1,2} = char('john','dmitry'); % an array of 2 strings is in C{1,2}
box
C{2,1} = PA; % a structure PA is in C{2,1} box
C{2,2} = cell(3,3); % a 3-by-3 nested cell is in C{2,2} box
```

ανακτά πληροφορίες για το είδος δεδομένων και το περιεχόμενο του κελιού

```
C(1,1), C(1,2), C(2,1),C(2,2)

ans =
```

```
    [3x3 double]
ans =
    [2x6 char]
ans =
    [1x1 struct]
ans =
    {3x3 cell}
```

```
D = C{1,1}
```

```
D =    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214
```

## Γραμμική άλγεβρα με MATLAB

**Γραμμικά συστήματα συναρτήσεων *m* με *n* αγνώστους:**

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$$
$$\ldots$$
$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m$$

**m = n:** ένα τεραγωνικό σύστημα με έναν τετραγωνικό πίνακα **A** (συνήθως έχει μία μοναδική λύση)
**m < n:** απροσδιόριστο σύστημα (έχει συνήθως άπειρες λύσεις)
**m > n:** υπερπροσδιορισμένο σύστημα (συνήθως δεν έχει καμία λύση)

**Παράδειγμα:**

$A = \begin{bmatrix} 3 & 2 \\ 1 & -1 \end{bmatrix}$, $b = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$. ***Η μοναδική λύση υπάρχει:*** $x_1 = 0.2$, $x_2 = -0.8$.

**Λειτουργίες και συναρτήσεις της MATLAB:**
"/", "\": διαιρέσεις πινάκων (λύσεις γραμμικού συστήματος)

```
 A = [ 3 2 ; 1, -1]; b = [-1 ; 1];
x = A \ b % the column-oriented solution of a linear system
dif = A*x - b   % check that the solution x is correct
```

```
x =    0.2000
      -0.8000
dif =       0
            0
```

```
AA = [3 1; 2,-1]; bb = [-1,1]; % transposed matrices A and b
xx = bb/AA % the row-oriented solution of a linear system
dif = xx*AA-bb % check that the solution x is correct
```

```
xx =    0.2000   -0.8000
dif =     0     0
```

**inv:** αντιστροφή πινάκων (***B είναι ο ανάστροφος του A if B*A = I)***

```
 A = [ 3 2 ; 1, -1]; B = inv(A), BB = A^(-1)    % two equivalent
operations
      I1 = B*A, I2 = A*B % check that B is the inverse of A

B =    0.2000    0.4000
       0.2000   -0.6000
BB =   0.2000    0.4000
       0.2000   -0.6000
I1 =   1.0000   -0.0000
       0    1.0000
I2 =   1.0000         0
      -0.0000    1.0000
```

**det:** υπολογίζει την ορίζουσα του πίνακα (***det A = $a_{11}$*$a_{22}$ – $a_{12}$*$a_{21}$ for a 2-by-2 matrix A)***
**trace:** υπολογίζει το άθροισμα των διαγώνιων στοιχείων ενός πίνακα ( εγγραφή ίχνους)
**rank:** υπολογίζει τον αριθμό των γραμμικά ανεξάρτητων σειρών και στηλών ενός πίνακα (βαθμό/τάξη)

```
  D = det(A), T = trace(A), R = rank(A)

D =    -5
T =     2
R =     2

  % Alternative computations of solutions of linear systems via A^(-1)
A = [ 3 2 ; 1, -1]; b = [-1 ; 1];
x = inv(A)*b % it is less efficient because of larger computational
time

x =    0.2000
       -0.8000

% Alternative computations of solutions by using the Cramer's rule
      A1 = [ b, A(:,2) ], A2 = [  A(:,1), b], clear x
      x(1) = det(A1)/det(A), x(2) = det(A2)/det(A)

A1 =       -1     2
            1    -1
A2 =       3    -1
      1     1
x =    0.2000
x =    0.2000   -0.8000
```

**rref:** υπολογίσει την ανηγμένη κλιμακωτή μορφή κατά γραμμές ενός πίνακα
**null:** υπολογίζει τη βάση των ιδιοδιανυσμάτων για την ομογενή εξίσωση ***A*x = 0***
**eig:** υπολογίζει τις ιδιοτιμές και τα ιδιοδιανύσματα του γραμμικού προβλήματος ***A*x =*** $\lambda$ ****x***

```
% Alternative computations of solutions by using the augmented matrix
[A,b]
Rrow = rref([A,b])
```

```
Rrow =      1.0000      0           0.2000
            0           1.0000   -0.8000
```

```
[V,D] = eig(A)
% D: diagonal matrix of eigenvalues
% V: fundamental matrix of eigenvectors
% V^(-1)*A*V = D: diagonalization formula
Dif = V^(-1)*A*V - D
```

```
V =    0.9757   -0.4100
       0.2193    0.9121
D =    3.4495         0
            0   -1.4495
Dif =  1.0e-015 *
                 -0.8882    0.2220
                  0.2776         0
```

```
N = null(A) % null-space is the basis of eigenvectors for zero
eigenvalues
```

```
N =   Empty matrix: 2-by-0
```

**flipud:** αποδίδει έναν πίνακα που έχει διατηρήσεις τις στήλες του και έχει ανεστραμμένες τις σειρές του με κατεύθυνση από πάνω προς τα κάτω
**fliplr:** αποδίδει έναν πίνακα που έχει διατηρήσεις τις σειρές του και έχει ανεστραμμένες τις στήλες του με κατεύθυνση από αριστερά προς τα δεξιά
**flipdim:** αποδίδει έναν πίνακα με ανεστραμμένες διαστάσεις (1 – σειρές, 2 – στήλες)

```
A = [ 1 2 3 4 ; 2 1 3 4 ; 3 2 1 4 ]
A1 = flipud(A)
A2 = fliplr(A)
A3 = flipdim(A,1)
A4 = flipdim(A,2)
```

```
A =
     1     2     3     4
     2     1     3     4
     3     2     1     4
A1 =
     3     2     1     4
     2     1     3     4
     1     2     3     4
A2 =
     4     3     2     1
     4     3     1     2
     4     1     2     3
A3 =
     3     2     1     4
     2     1     3     4
     1     2     3     4
A4 =
     4     3     2     1
     4     3     1     2
     4     1     2     3
```

## 10. Λύσεις γραμμικών συστημάτων

- Ένα γραμμικό τετραγωνικό σύστημα γραμμικών εξισώσεων έχει μια μοναδική λύση μόνο αν ***det(A)*** $\neq$ ***0***.
- Όταν ***det(A) = 0,*** το γραμμικό σύστημα μπορεί να έχει άπειρες λύσεις ή καθόλου λύσεις.
- Αν η ***det(A)*** $\neq$ ***0,*** ο ανάστροφος πίνακας ***A^(-1)*** υπάρχει και ***A*** ονομάζεται ομαλός πίνακας.
- Αν ***det(A) = 0, inv(A)*** δεν υπάρχει και ***A*** ονομάζεται μη ομαλός πίνακας. Οι αλγόριθμοι επίλυσης γραμμικής άλγεβρας της MATLAB ***A\b*** or ***b/A*** δεν αποδίδουν κάποια σημαντική λύση, αν ο ***A*** δεν είναι μη ομαλός.

```
   A1 = [ -1,1; -2,2]; b1 = [1 ; 0];
% the system is inconsistent and has no solution
 x1 = A1\b1, B1 = inv(A1), D1 = det(A1)

Warning: Matrix is singular to working precision.
x1 =        Inf
      Inf

Warning: Matrix is singular to working precision.
B1 =    Inf    Inf
      Inf    Inf
D1 =        0



   A2 = [ -1,1; -2,2]; b2 = [1 ; 2];
% the second equation is redundant
% the system has infinitely many solutions: x(2) = 1 + x(1)
 x2 = A2\b2
 A3 = [ -1, 1 ]; b3 = [ 1 ]; % the second equation is removed
 x3 = A3\b3

Warning: Matrix is singular to working precision.
x2 =    Inf
            Inf
x3 =        -1
      0

 % comparison of properties of non-singular matrix A versus singular
matrix A1
S1 = rref(A) % non-singular matrix has the identity matrix in RREF
S2 = rref(A1) % a singular matrix has zeros in one or more rows in RREF
S1 =        1     0
            0     1
S2 =        1    -1
            0     0
R1 = rank(A1) % singular matrices have rank smaller than 2
N1 = null(A1) % the null-space of singular matrices is non-empty
[V1,D1] = eig(A1) % singular matrices have zero eigenvalues
```

```
R1 =         1
N1 =         0.7071
                  0.7071
V1 =   -0.7071   -0.4472
            -0.7071   -0.8944
D1 =     0     0
                  0     1
Rrow1 = rref([A2,b2]) % accurate solution of the system with singular
matrices

Rrow1 =      1    -1    -1
                  0     0     0
```

## 11. Συστήματα κακή κατάστασης (Ill-conditioned) και σποραδικοί πίνακες

Οι αριθμητικοί υπολογισμοί υπόκεινται πάντα σε λάθη στρογγυλοποίησης που προέρχονται εξαιτίας της περιορισμένης ακρίβειας της μηχανής, π.χ. ***eps ~ 10$^{-16}$.*** Το σφάλμα στρογγυλοποίησης μπορεί να μεγαλώσει κατά πολύ για ασθενή συστήματα.

```
   A = hilb(15); % Hilbert matrices are ill-conditioned
        b1 = ones(15,1); % b1 is the vector of ones
        b2 = b1 + 0.01*rand(15,1); % b2 is a small random perturbation
to b1
            x1 = A\b1; y1 = x1'
x2 = A\b2; y2 = x2'
% Two solutions of linear systems with almost equal right-hand-sides
bdif = norm(b2-b1), ydif = norm(y2-y1)

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.543404e-018.
y1 =  1.0e+008 *
    0.0000   -0.0000    0.0007   -0.0097    0.0738   -0.3122    0.7102
-0.5709   -1.0953    3.3002   -2.9619   -0.2353    2.4214   -1.7331
0.4121

Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.543404e-018.
y2 =  1.0e+014 *
   -0.0000    0.0000   -0.0001    0.0015   -0.0157    0.0998   -0.4111
1.1384   -2.1512    2.7591   -2.3245    1.1786   -0.2709   -0.0211
0.0172

bdif =  0.0205
ydif =  4.5367e+014
```

Τα μεγάλα σφάλματα για τα ασθενή συστήματα προκαλούνται από τους μικρούς δείκτες: $x_n = det\ A_n/detA$. If $det\ A \sim 0$, και $det\ A_n$ υπολογίζεται με σφάλμα $e_n$, τότε το απόλυτο σφάλμα μεγιστοποιείται: $|x_n - x_n^{exact}| = e_n/det\ A \sim large$. Οι λύσεις για ασθενή συστήματα γίνονται ανακριβείς με μεγάλο απόλυτο απόλυτο σφάλμα.

## 12. Νόρμες για διανύσματα και πίνακες

- **norm(x):** υπολογίζει το μέτρο για ένα διάνυσμα ***x***

```
x = [ 4 5 ]; % a row 2-vector
n1 = norm(x) % computes a geometric length (2-norm) of a vector:
             % norm(x) = norm(x,2) = sqrt(x(1)^2 + x(2)^2)
n2 = norm(x,1) %  computes a 1-norm of a vector: norm(x,1) = |x(1)| +
|x(2)|
n3 = norm(x,inf) % computes an infinity-norm of a vector:
                 % norm(x,inf) = max(|x(1)|,|x(2)|)

n1 =    6.4031
n2 =     9
n3 =     5
```

- **norm(A):** υπολογίζει το μέτρο για έναν πίνακα ***A***

```
A = [ 3, 2; 1,-1 ]; % a 2-by-2 matrix
n1 = norm(A,'fro') % computes the Frobenius norm of a matrix:
 % norm(A,'fro') = sqrt(sum(diag(A'*A)))
n2 = norm(A,1) % computes the 1-norm of a matrix (the largest column
sum):
  % norm(A,1) = max(sum(abs((A))))
n3 = norm(A,inf) % computes the infinity norm of a matrix (the largest
row sum):
                                          % norm(A,inf) =
max(sum(abs((X'))))

n1 =    3.8730
n2 =     4
n3 =     5
```

**cond(A):** υπολογίζει τους αριθμούς κατάστασης ενός πίνακα σε σχέση με την αντιστροφή.

```
C1 = cond(A,'fro')
C2 = norm(A,'fro')*norm(inv(A),'fro')
           % Properties of the condition number:
% cond(A) = norm(A)*norm(A^(-1)) in any norm
% cond(A) >= 1
C1 =        3.0000
C2 =        3.0000
```

**<u>Κριτήρια για ασθενή γραμμικά συστήματα:</u>**

Δύο θεωρητικά κριτήρια για ασθενή συστήματα ανεξάρτητα από το περιβάλλον υπολογισμού είναι:
μεγάλες τιμές του ***cond(A)***
μικρές τιμές του ***det(A)***

Δύο πρακτικά κριτήρια για ασθενή συστήματα στο δεδομένο υπολογιστικό περιβάλλον είναι:
***cond(A*inv(A)) ~= 1***
***det(A)*det(inv(A)) ~= 1***

Αν η υψηλή ακρίβεια χρησιμοποιείται στο υπολογιστικό περιβάλλον, οι λύσεις για τα γραμμικά συστήματα μπορεί να είναι ακριβείς ακόμη και αν τα θεωρητικά κριτήρια προβλέπουν ότι ο πίνακας συντελεστής ***A*** είναι ασθενής. Αντίθετα, τα πρακτικά κριτήρια προβλέπουν ότι το πραγματικό λάθος στις λύσεις των γραμμικών συστημάτων είναι μεγάλο στο δεδομένο υπολογιστικό περιβάλλον.

Εφόσον το ***inv(A)*** χρησιμοποιείται για λύσεις γραμμικών συστημάτων, υο γινόμενο ***inv(A)*A*** ή ***A*inv(A)*** χρησιμοποιείται για πρακτικά μέτρα κατά του σφάλματος στο δεδομένο υπολογιστικό περιβάλλον. Τα γινόμενα ***inv(A)*A*** ή ***A*inv(A)*** πρέπει να αναπαράγουν έναν μοναδιαίο πίνακα. Η διαφορά μεταξύ του μοναδιαίου πίνακα και του ***inv(A)*A,*** αν αυτή υπάρχει, υποδεικνύει ένα υπολογιστικό σφάλμα στην επίλυση των γραμμικών συστημάτων με τον πίνακα συντελεστή ***A.***

Για ένα δεδομένο ασθενές σύστημα, μία αύξηση στην ακρίβεια θα μειώσει τα σφάλματα των λύσεων του γραμμικού συστήματος. Από την MATLAB χρησιμοποιείται η διπλή ακρίβεια και αυτή επαρκεί για ήπια ασθενή συστήματα. Η τετραπλή ακρίβεια (δεν διατίθεται στη MATLAB) θα πρέπει να χρησιμοποιείται για σοβαρά ασθενή συστήματα.

```
for n = 1 : 15
      A = hilb(n); B = inv(A); C = A*B;
      c1 = cond(A); c2 = det(A); c3 = det(B);
      d1 = c2*c3; d2 = cond(C);
fprintf('n = %2.0f c1 = %3.1e, c2 = %3.1e, d1 = %3.1e, d2
=%3.1e\n',n,c1,c2,d1,d2);
end
```

```
n =  1 c1 = 1.0e+000, c2 = 1.0e+000, d1 = 1.0e+000, d2 =1.0e+000
n =  2 c1 = 1.9e+001, c2 = 8.3e-002, d1 = 1.0e+000, d2 =1.0e+000
n =  3 c1 = 5.2e+002, c2 = 4.6e-004, d1 = 1.0e+000, d2 =1.0e+000
n =  4 c1 = 1.6e+004, c2 = 1.7e-007, d1 = 1.0e+000, d2 =1.0e+000
n =  5 c1 = 4.8e+005, c2 = 3.7e-012, d1 = 1.0e+000, d2 =1.0e+000
n =  6 c1 = 1.5e+007, c2 = 5.4e-018, d1 = 1.0e+000, d2 =1.0e+000
n =  7 c1 = 4.8e+008, c2 = 4.8e-025, d1 = 1.0e+000, d2 =1.0e+000
n =  8 c1 = 1.5e+010, c2 = 2.7e-033, d1 = 1.0e+000, d2 =1.0e+000
n =  9 c1 = 4.9e+011, c2 = 9.7e-043, d1 = 1.0e+000, d2 =1.0e+000
n = 10 c1 = 1.6e+013, c2 = 2.2e-053, d1 = 1.0e+000, d2 =1.0e+000
n = 11 c1 = 5.2e+014, c2 = 3.0e-065, d1 = 1.0e+000, d2 =1.7e+000
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.632091e-017.
n = 12 c1 = 1.8e+016, c2 = 2.9e-078, d1 = 9.4e-001, d2 =2.3e+002
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.348790e-018.
n = 13 c1 = 3.8e+018, c2 = 4.5e-092, d1 = 8.4e-001, d2 =1.4e+004
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.408541e-019.
n = 14 c1 = 4.1e+017, c2 = -3.2e-107, d1 = -3.3e-001, d2 =4.2e+006
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.543404e-018.
n = 15 c1 = 8.5e+017, c2 = -2.2e-120, d1 = 1.6e-001, d2 =2.1e+004
```

**Σχόλιο:** Στο παραπάνω παράδειγμα, ο αριθμός κατάστασης ***c1*** είναι τεράστιος και η ορίζουσα ***c2*** είναι μικρή για το ***n > 5.*** Το πραγματικό σφάλμα είναι ωστόσο ορατό στο δεδομένο υπολογιστικό περιβάλλον, εφόσον τα ***d1*** και ***d2*** δεν διαφέρουν από τις θεωρητικές τιμές: ***d1 = d2 = 1.*** Το υπολογιστικό σφάλμα γίνεται ορατό για ***n > 10***. Οι λύσεις για γραμμικά συστήματα με πίνακες Hilbert το ***A*** γίνεται ανακριβές στο δεδομένο υπολογιστικό περιβάλλον για ***n > 10.***

**Αραιοί Πίνακες:**

Οι αραιοί πίνακες έχουν μηδέν στα περισσότερα στοιχεία τους.
Οι αραιοί πίνακες μπορεί να αντιπροσωπεύονται με μειωμένη αποθήκευση στον υπολογιστή
Όλοι οι πολλαπλασιασμοί πινάκων με τους αραιούς πίνακες με ένα μειωμένο αριθμό λειτουργιών.

**sparse**: μετατρέπει έναν αραιό πίνακα σε μία μορφή αραιής συστοιχίας πετώντας έξω οποιαδήποτε μηδενικά στοιχεία**.** Η MATLAB λειτουργεί με αραιές συστοιχίες όπως και με αυθεντικούς πίνακες αλλά εξοικονομεί αποθηκευτικό χώρο και υπολογιστικό χρόνο καθώς εκμεταλλεύεται την αραιή δομή ενός πίνακα.

```
A = -2*diag(ones(1,5)) + diag(ones(1,4),1) + diag(ones(1,4),-1)
S = sparse(A)
```

```
A =
     -2      1      0      0      0
      1     -2      1      0      0
      0      1     -2      1      0
      0      0      1     -2      1
      0      0      0      1     -2


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

```
b = ones(5,1);
Q1 = b'*A*b % computations of a quadratic form with vector b
Q2 = b'*S*b  % alternative computation of the same quadratic form
% the second computation uses smaller number of floaping point
operations
```

```
Q1 =    -2
Q2 =    -2

      B = A^2
% B is central-difference approximation for fourth-derivative
% u''''(x) = (u(x+2h)-4*u(x+h)+6*u(x)-4*u(x-h)+u(x-2*h))/h^4
% B = A*A ~= A.^2
% Matrix power is not the same as the pointwise power to elements of
matrix
R = S^2
% the same operation can be performed with sparse arrays to save
computer time

B =
     5    -4     1     0     0
    -4     6    -4     1     0
     1    -4     6    -4     1
     0     1    -4     6    -4
     0     0     1    -4     5
R =
   (1,1)         5
   (2,1)        -4
   (3,1)         1
   (1,2)        -4
   (2,2)         6
   (3,2)        -4
   (4,2)         1
   (1,3)         1
   (2,3)        -4
   (3,3)         6
   (4,3)        -4
   (5,3)         1
   (2,4)         1
   (3,4)        -4
   (4,4)         6
   (5,4)        -4
   (3,5)         1
   (4,5)        -4
   (5,5)         5
```

## 13. Γραφική παράσταση συναρτήσεων με δύο μεταβλητές

**Συναρτήσεις σε παραλληλόγραμμα πλέγματα:**

Ας υποθέσουμε ότι έχουμε μία συνάρτηση με δύο μεταβλητές: $z = f(x,y)$ σε παραλληλόγραμμο πεδίο ορισμού:

$D = \{ (x,y): a \le x \le b; c \le y \le d \}$

Ορίστε το διακριτό διάστημα για τον άξονα ***x-axis:***

$[ a, x_1, x_2, \ldots, x_N, b ]$

Ορίστε το διακριτό διάστημα για τον άξονα ***y-axis:***

$[ c, y_1, y_2, \ldots, y_M, d ]$

Η τομή δύο κάθετων γραμμών $x = x_n$ και οριζοντίων γραμμών $y = y_m$ σχηματίζουν ένα παραλληλόγραμμο πλέγμα ***N-επί-M*** εσωτερικών σημείων και (***2*N+2*M***) σημεία ορίων και ***4***

γωνιακά σημεία. Οι εκτιμήσεις της συνάρτησης ***z = f(x,y)*** στα σημεία πλέγματος ορίζει έναν πίνακα ***(N+2)-by-(M+2)*** με τιμές: $z_{\{n,m\}} = f(x_n, y_m)$.

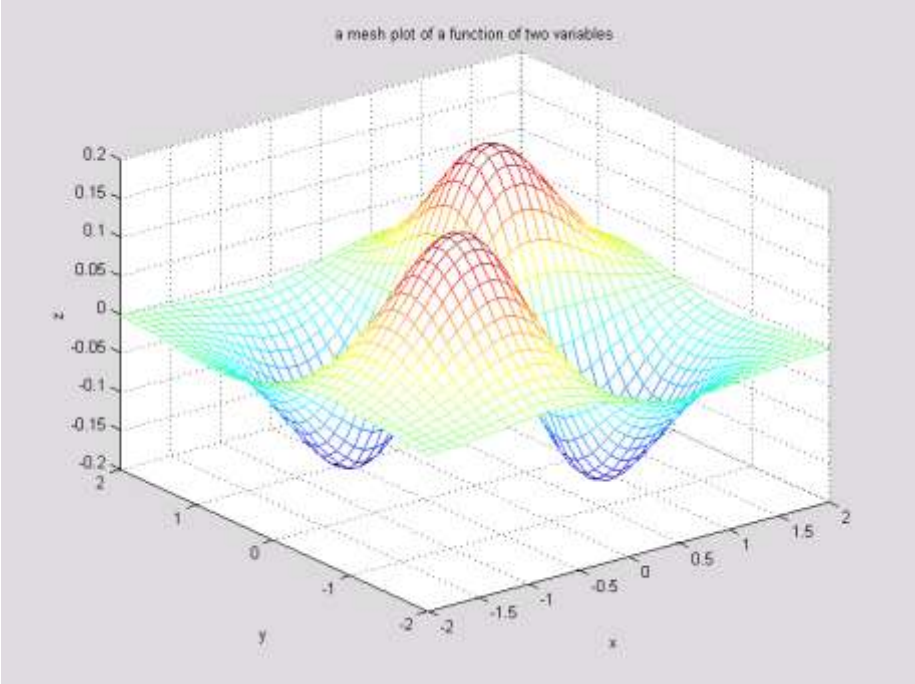
**meshgrid(x,y):** δημιουργεί έναν πίνακα για παραλληλόγραμμα πλέγματα **[X,Y]** από τα διανύσματα **[x,y]**

```
x = 0 : 0.5 : 2;  % a row-vector of 5 points for the x-axis
y = -3 : 1 : 3;  % a row-vector of 7 points for the y-axis
[X,Y] = meshgrid(x,y) % create 5-by-7 matrices for grids of X and Y
      % meshgrid is equivalent to: X = ones(7,1)*x; Y = y'*ones(1,5);
```

```
X         0    0.5000    1.0000    1.5000    2.0000
          0    0.5000    1.0000    1.5000    2.0000
          0    0.5000    1.0000    1.5000    2.0000
          0    0.5000    1.0000    1.5000    2.0000
          0    0.5000    1.0000    1.5000    2.0000
          0    0.5000    1.0000    1.5000    2.0000
          0    0.5000    1.0000    1.5000    2.0000

Y =   -3    -3    -3    -3    -3
       -2    -2    -2    -2    -2
         -1    -1    -1    -1    -1
          0     0     0     0     0
          1     1     1     1     1
          2     2     2     2     2
                3     3     3     3     3
```

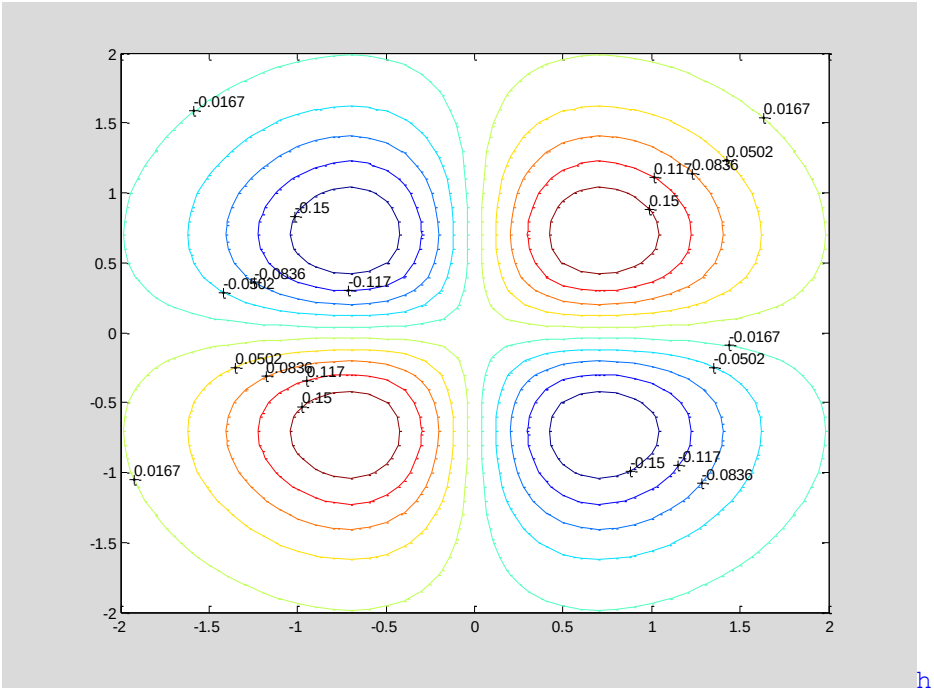
**mesh(x,y,z):** Σχεδιάστε το σχήμα μίας συνάρτησης δύο μεταβλητών σε ένα παραλληλόγραμμο πλέγμα

```
x = -2 : 0.1 : 2; y = -2 : 0.1 : 2;
% z = x.*y.*exp(-x.^2 - y.^2) : example of a function of two variables
% direct coding of the function z = f(x,y) does not work because x,y,z
are row-vectors

[X,Y] = meshgrid(x,y);
Z = X.*Y.*exp(-X.^2-Y.^2);
% pointwise multiplication works now because X and Y are matrices
mesh(X,Y,Z);
title('a mesh plot of a function of two variables');
xlabel('x'); ylabel('y'); zlabel('z');
```




- **contour(x,y,z,label):** σχεδιάζει τις κλειστές καμπύλες μίας συνάρτησης δύο μεταβλητών σε ένα παραλληλόγραμμο πλέγμα
  - **level = L**, όπου ***L*** είναι ο αριθμός των ισαπεχόντων επιπέδων κλειστών καμπυλών μεταξύ των ελάχιστων και μέγιστων τιμών του ***z:*** $z_l = z_{min}+(z_{max}-z_{min})*(l-1)/(L-1)$
  - **level = V,** όπου ***V*** είναι ένα διάνυσμα των επιπέδων κλειστών καμπυλών που δίνεται συγκεκριμένα
  - **clabel(h):** υποσημειώνει αυτόματα τα επίπεδα των κλειστών καμπύλων
  - **clabel(h,'manual'):** επιτρέπει στον χρήστη να εντοπίσει τις θέσεις των ετικετών των επιπέδων κλειστών καμπυλών

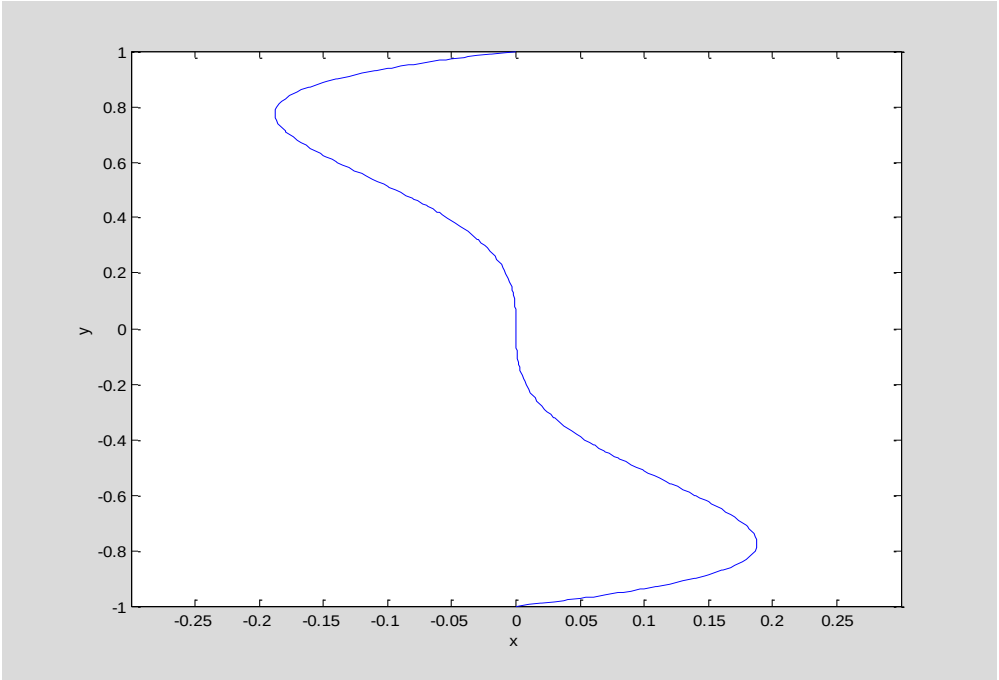
```
h = contour(X,Y,Z,10); clabel(h);
```




```
h =
```

- **contour(x,y,z,[0,0]):** σχεδιάζη μία ρητή συνάρτηση μίας μεταβλητής: ***z = F(x,y) = 0***

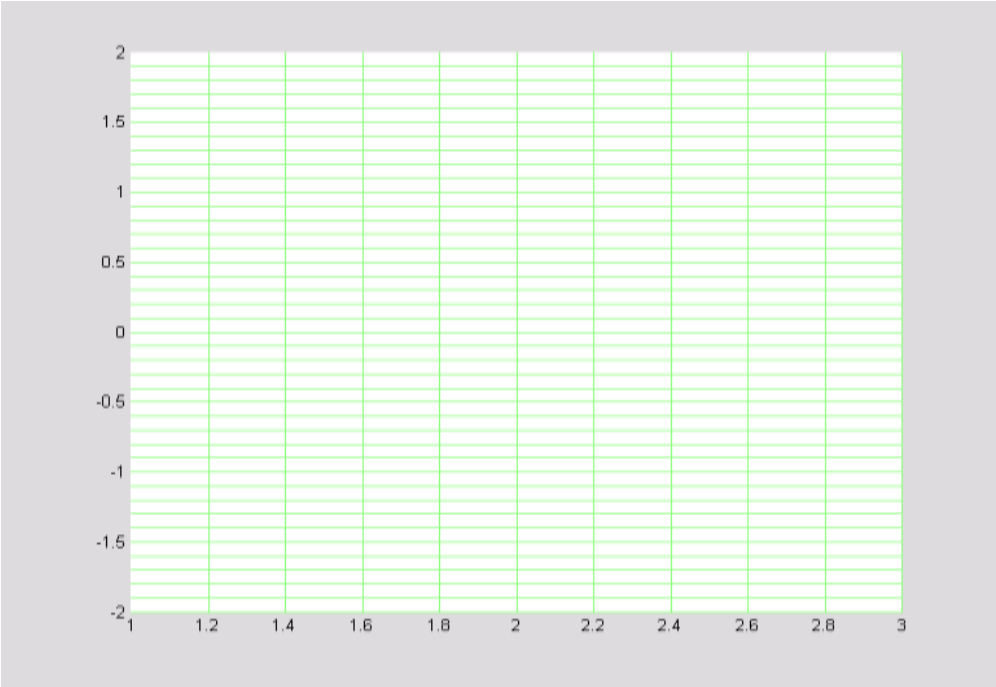
```
 % y^5 - y^3 = tanh(x) : the implicit function y = y(x) to be plotted
 x = -0.3 : 0.01 : 0.3; y = -1 : 0.01 : 1; [X,Y] = meshgrid(x,y);
Z = Y.^5-Y.^3-tanh(X);  contour(x,y,Z,[0,0],'b'); xlabel('x');
ylabel('y');
```




- **mesh(x,y,0*y'*x):** σχεδιάζει ένα παραλληλόγραμμο πλέγμα χωρίς καμία συνάρτηση εντός του πλέγματος

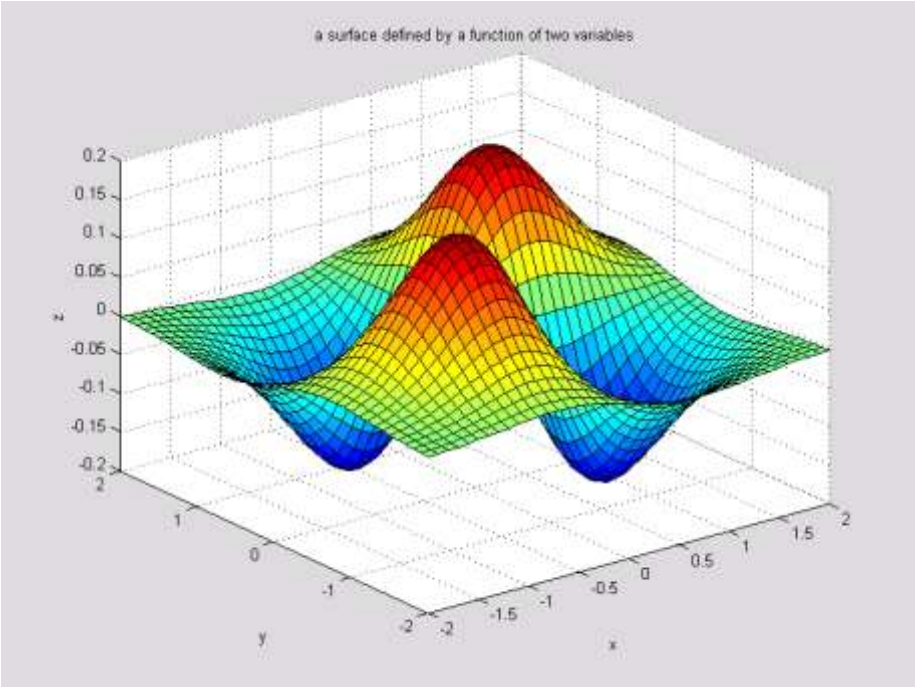
```
x = 1 : 0.2 : 3; y = -2 : 0.1 : 2;
```

```
mesh(x,y,0*y'*x); view([0,0,1]);
```



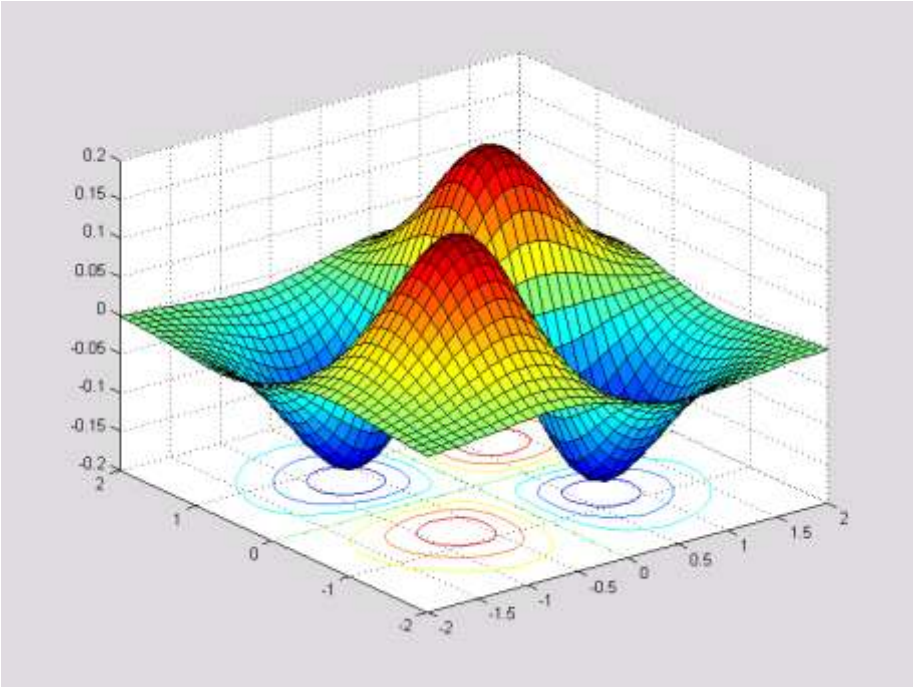

- **surf:** Δημιουργεί μια πολύχρωμη εικόνα μιας δισδιάστατης επιφάνειας

```
x = -2 : 0.1 : 2; y = -2 : 0.1 : 2;
[X,Y] = meshgrid(x,y); Z = X.*Y.*exp(-X.^2-Y.^2);
surf(X,Y,Z); title('a surface defined by a function of two variables');
xlabel('x'); ylabel('y'); zlabel('z');
```



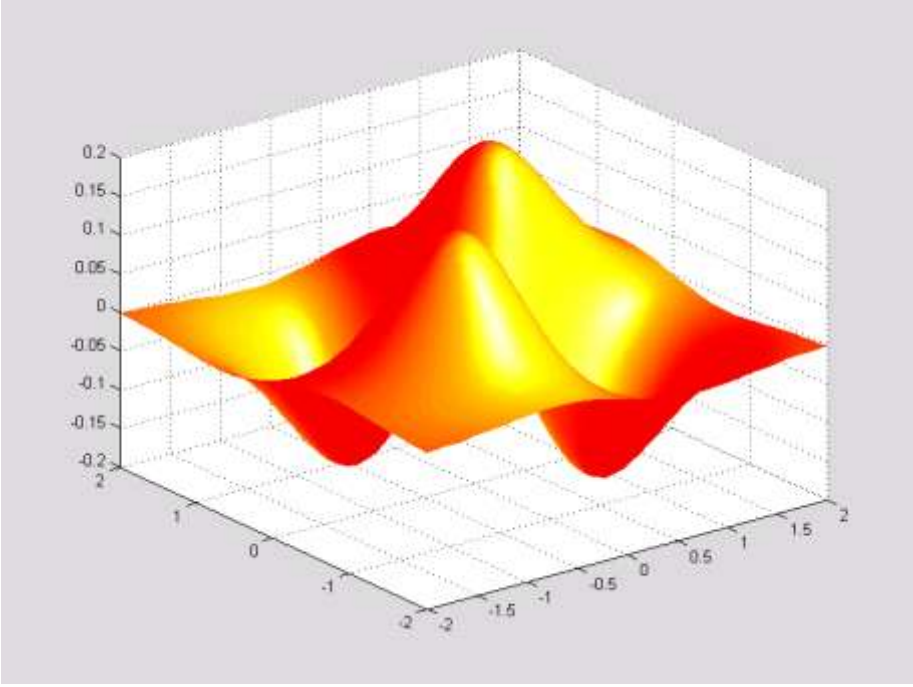
- **meshc, surfc:** σχεδιάζει τις κλειστές καμπύλες της συνάρτησης ***z = f(x,y)*** στο ***(x,y)***-επίπεδο επιπλέον της απόδοσης των συναρτήσεων **"mesh"** και **"surf"**

```
surfc(X,Y,Z)
```



- **surfl:** δημιουργεί μία δισδιάστατη επιφάνεια με φωτισμό

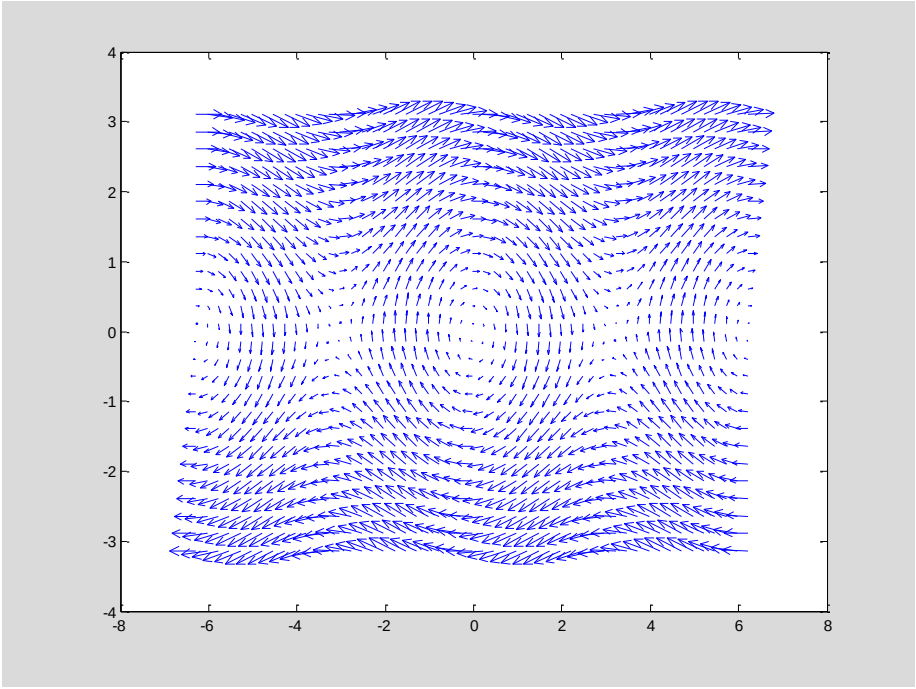
```
surfl(X,Y,Z); colormap hot; shading interp;
```



- **quiver:** σχεδιάζει ένα δισδιάστατο διάνυσμα **[u,v]** στο επίπεδο **[x,y],** όπου **u = f(x,y); y = g(x,y)**

```
% The system of two differential equations:
% dx/dt = f(x,y) = y; dy/dt = g(x,y) = - sin(x)
```

```
% The quiver displays the vector field on the phase plane (x,y)
x = -2*pi:0.25:2*pi; y = -pi:0.25:pi; [X,Y] = meshgrid(x,y);
U = Y; V = -sin(X); quiver(X,Y,U,V,2);
% the last input argument is a scale factor,
% that adjusts the lengths of vectors on the plane
```

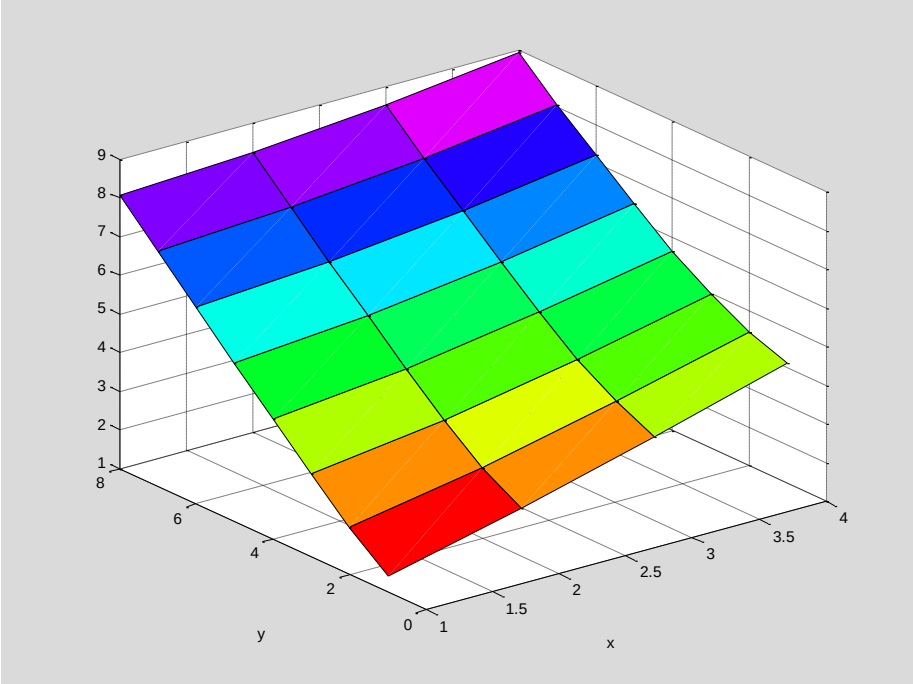



**<u>Περισσότερες λειτουργίες στα τρισδιάστατα γραφήματα:</u>**

**axis([xmin,xmax,ymin,ymax,zmin,zmax]):** ορίζει τα όρια του τρισδιάστατου χώρου

- **xlabel, ylabel, zlabel:** ετικέτες για τα **x, y, z**
- **view([x,y,z]):** αλλάζει την οπτική γωνία του γραφήματος, όπου **[x,y,z]** είναι οι συντεταγμένες ενός θεατή που κοιτάει το αρχικό σημείο **[0,0,0]**
- **view([az,el]):** αλλάζει την οπτική γωνία του γραφήματος, όπου **az** είναι η αζιμούθια γωνία στο επίπεδο ***(x,y)***, το οποίο μετράται από τον αρνητικό άξονα **y**-axis αντίθετα με τη φορά του ρολογιού και **el** είναι η γωνία ανύψωσης από το επίπεδο ***(x,y)***
  - **default: view(-37.5,30)**
- **colormap:** αλλάζει τον προσδιορισμό των χρωμάτων στον άξονα των χρωμάτων
  - **default: colormap(hsv),** όπου **red** ορίζεται να είναι οι χαμηλότερες και υψηλότερες τιμές του **z**, ενώ οι ενδιάμεσες τιμές του ζ είναι ***z***: **red, yellow, green, cyan, blue, magenta, red**
  - η κάθε σειρά του **"mesh"** χρωματίζεται με το χρώμα που προσδιορίζεται από τη μέση τιμή μίας συνάρτησης στα δύο παρακείμενα σημεία της στη γραμμή
  - το κάθε κελί του **"surf"** χρωματίζεται με το χρώμα που προσδιορίζεται από τη μέση τιμή μίας συνάρτησης στα τέσσερα γωνιακά σημεία του κελιού
  - άλλοι χάρτες χρωμάτων: **hot, cold, jet, gray**

```
x = 1:4; y = 1:8; [X,Y] = meshgrid(x,y); Z = sqrt(X.^2 + Y.^2);
colormap(hsv); surf(Z); xlabel('x'); ylabel('y');
```




- **shading:** αλλάζει την οπτική της επιφάνειας
  - **faceted:** μαύρες γραμμές ορίων και τετράπλευρα πλακάκια
  - **flat:** χωρίς γραμμές ορίων
  - **interp:** χωρίς γραμμές ορίων και ομαλές επιφάνειες
- **caxis([zmin,zmax]):** σχεδιάζει τα χρώματα μεταξύ των ελάχιστων και μέγιστων τιμών του **z** που ορίζονται από τον χρήστη

**Σχόλιο:** είναι χρήσιμο να προσδιορίζουμε όλα τα χαρακτηριστικά ενός γραφήματος και να τα διατηρούμε με την εντολή **"hold on"** πριν να σχεδιάσουμε την επιφάνεια, γιατί οι λειτουργίες σχεδίασης είναι χρονοβόρες στον τρισδιάστατο χώρο.

## Παράρτημα 1.1: Γρήγορη εισαγωγή στο υπολογιστικό περιβάλλον MatLab

```
Compact Summary of MATLAB language (MATrix LABoratory)
A mixture of Fortran and C
This compact summary assumes you know either Fortran or C
and can program in at least one of those languages.

MATLAB is an interpreter, not a compiler.
Multiple files may be used in a single execution and files are
interpreted
as needed when the files are in the current workspace.
Files have an extension ".m"
Files are plain ASCII text and may be edited within MATLAB or
created and edited using your favorite editor.
For experimentation, you can type statements into the command window
and
they are interpreted as you type them.
Execute a file in the current workspace using:   run  filename-without-
.m

Each line is a statement.
```

```
The percent character, %, makes the rest of the line a comment.
Ending a statement with a semicolon prevents printing of that statement
during interpreting.
Multiple statements may be on one line when separated by semicolons.
To continue a line, end it with three periods.

Variables must start with a letter and may include upper case, lower
case,
digits and underscore. Case sensitive.
All variables are arrays of some size and dimension.
A single number is a one element one dimensional array.
Variables are not declared. Unless otherwise specified all variables
are
IEEE Standard 754 64-bit floating point. Complex numbers are a pair of
64-bit floating point values.

Key words listed below are not allowed as variables.
Functions and commands should not be used as variables, yet, if used,
can
be "recovered" using the 'clear' command or 'builtin' function.
A few of the many functions and commands are listed below.

Key Words: (all lower case)
  break    case       catch    continue   else        elseif       end
  for      function   global   if         otherwise   persistent
return
  switch   try        while

A few commands to avoid as variables
  clear    format     dir    regexp   close  import  load   pack   save

There are hundreds of functions, here are some you might recognize:
  sin      cos    tan    asin    acos    atan     atan2   exp   log
log2
  sinh     cosh   tanh   asinh   acosh   atanh    sign    abs   fix
log10
  round    floor  ceil   sum     prod    length   rem     mod   eval
feval
  and      or     not    xor     any     all      find    eig
ode23
  bitand   bitor  bitcmp bitxor  exists  isempty
  input    disp   fprintf

There are some special names for values:
  See more constants below, such as   pi  3.14159...

  When following a number with no intervening space:
    i   sqrt(-1)       or i unused and  exp(i*t)
    j   same as i

  In functions: nargin   is number of arguments the caller provided
                nargout  is number of outputs the called requested

  In try - catch: lasterr  is the cause of the error

  (If you define and use any special names or functions, you may get
   the builtin definition back with  clear pi )
```

```
Expressions are built in the normal way from variables and operators.
The standard convention for parenthesis applies.
Subscripts use the parenthesis as in Fortran rather than brackets as in
C.
Because variables may be arrays of any size and dimension, be careful.

The operators in precedence order are:  (precedence number first)
 1   (   )   Grouping parentheses including function calls
 2   .'      Transpose, period apostrophe with no intervening space
 2   .^      Power, element by element, 2 .^ n  pow(2,n) in C, 2**n in
Fortran
 2   '       Complex Conjugate Transpose
 2   ^       Matrix to a power, repeated matrix multiplication
 3   +       Unary Plus
 3   -       Unary Minus
 3   ~       Logical Complement, tilde, ( result is 1 or 0 )
 4   .*      Multiplication, element by element
 4   ./      Right Division, element by element
 4   .\      Left Division, element by element
 4   *       Matrix Multiplication
 4   /       Matrix right division
 4   \       Matrix left division
 5   +       Addition
 5   -       Subtraction
 6   :       Colon Operator  a:b sequence from a to b step 1, a:b:c  c
is step
 7   <       Less Than, element by element, result 1 or 0
 7   <=      Less Than or Equal, element by element, result 1 or 0
 7   >       Greater Than, element by element, result 1 or 0
 7   >=      Greater Than or Equal, element by element, result 1 or 0
 7   ==      Equal, element by element, result 1 or 0 (a new matrix if
matrices compared)
 7   ~=      Not Equal, element by element, result 1 or 0
 8   &       And, element by element, result 1 or 0
 9   |       Or, element by element, result 1 or 0
10   &&      Short Circuit And
11   ||      Short Circuit Or
11   xor     Returns 1 for every element nonzero in only one array,
else 0

Concatenation of strings:
  ['some' ' integer=' int2str(n) ', x=' num2str(x,'%6.3f')]

Constants:
  numbers  123  123.  .123  12.3  12.34e-10  12.34e10  12.34e+10  0
0.0
  string   'abc'  (actually a one dimensional array of characters as
numbers)
  string   s = '(a+b).*c'     can be used later as  eval(s)
  constants:
     pi       3.14159...
     eps      2 .^ -52
     inf      IEEE Inf
     nan      IEEE not a number
     realmax  Largest number
     realmin  Smallest number
```

```
Assignment statements:
  a = 1         % 1.0 in C,  1.0D0 in Fortran
  a = [1 2 3]   % three element row vector
  a = [1, 2, 3] % same three element row vector
  a = [1; 2; 3} % three element column vector
  a = [[1 2 3]; [4 5 6]] % two row, three column matrix

Iteration statements:

  for i = 1:10  % for(i=1; i<=10; i++) in C,  do i = 1,10 in Fortran
    % any number of statements
  end

  for i = 0: Pi/8: Pi  % for(i=0.0; i<=Pi; i=i+Pi/8.0) in C  do i = 0,
Pi, Pi/8.0 in Fortran
    % any number of statements
  end

  while a < b
    % any number of statements
  end

    continue   % causes next iteration

    break      % causes exit from iteration


Conditional statements:

  if a<b
    % any number of statements
  elseif c>d
    % any number of statements
  elseif e == f
    % any number of statements
  else
    % any number of statements
  end


  switch expression
    case 0
    % any number of statements
        case 1
    % any number of statements
        case 9
    % any number of statements
        otherwise
    % any number of statements
  end   % no 'break' needed as in "C"


  try
    % any number of statements
  catch
    % any number of statements
```

```
  end

Functions:

  function variable = function_name(argument(s))
  % comments that will be printed upon        help function_name
  % the function may be in a file             function_name.m
  % the function may be viewed                type function_name
  % the function, in a file, may be executed  run function_name
  %                                  or just  function_name
  %any number of statements, nargin and nargout available
  variable = ...  % compute returned variables

  return  % optional, may be used anywhere, default at end
  end     % optional except when this is a nested function

  % warning! variables defined in outer function are all
  %          available in nested functions. Use of a parameter
  %          name that is the same as a global variable is bad.


Vector of functions and use:
   fun = [@sin; @cos; @log]
   k = some_expression
   x = argument_value
   feval(fun(k), x)

Input and output statements

  Cause a postscript file to be generated for your plot:
  print a_name.ps   % check out other formats in help for print

  k = input('Enter a number: ')

  file_name = input('Enter file name: ', 's')  % no apostrophy from
user

  Any statement that is interpreted that does not end with a semicolon
  will print something. The format for printing is controlled by:

    format short
        format short e
        format short g
        format long
        format long e
        format long g
        format bank         % ddd.dd
        format rat          % rational 7/3
        format hex
        format compact      % eliminates blank lines

    disp('any string')       % prints without the "ans="
    disp(x)                  % prints without the "x="

        use  sprintf  or  fprintf  to do your own formatting
    sprintf(' normal "C" format', variables)  %prints ans = and your
stuff
```

```
    fprintf(' normal "C" format', variables); % prints on screen like
printf

        fid = fopen('your.file', 'w');
        fprintf(' normal "C" format', variables);
        ...
        fclose(fid);  % write to file, no output on screen

    diary output_filename  % non graphic output to screen written to
file
       ...
    diary off              % this keeps appending to file when run
again

Special characters:
  \a  beep
  \b  backspace
  \e  escape
  \f  form feed
  \n  new line
  \r  carriage return
  \t  horizontal tab
  \v  vertical tab
  \o  octal
  \x  hexadecimal

  and, all the LaTeX characters \leq  \pi  {\itt}  etc

Colon Operator:  ":"
  1:5       is a vector  1  2  3  4  5
  1:2:10    is a vector  1  3  5  7  9
  0:pi/8:pi is a vector  0.0   pi/8  pi/4  ...
  100:-2:90 is a vector  100  98  96  94  92  90

  t=(0:3)'  % transpose gives column  0
                                      1
                                      2
                                      3

Matrix definitions and functions

  A = zeros(10)             a vector of 10 zeros
  A = zeros(10,12)          a 10 row 12 column matrix of zeros
  A = zeros(nx, ny, nz, nt) a four dimensional matrix of zeros
  A = ones(nx, ny)          a nx by ny matrix of ones
  A = rand(3, 5, 7)         a 3D matrix of uniformly distributed random
  A = randn(5, ny)          a matrix of normally distributed random
numbers
  A = eye(n)                an n by n identity matrix
  B = inv(A)                B is the matrix inverse of A
  d = det(A)                d is determinant of A
  L = eig(A)                L is eigenvalues
  [V,L] = eig(a)            V are eigenvectors, L is eigenvalues

  [r,c] = size(A)   number of rows and columns of 2D matrix A
  d     = size(A,n) the size of the nth dimension of matrix A
```

```
Sample code to read a file (processing of input line left to
programmer)

  Simple case, one vector of numbers
    fid = fopen('your.file', 'r');
    [v count] = fscanf(fid, '%f', v);
    fclose(fid); % v now has 1:count values

  or, if format processing is needed:

  fid = fopen('your.file', 'r');
  while ~feof(fid)
    line = fget(fid);
    if feof(fid)
      break       % no more input
    end if
    if isempty(line) | strncmp(line, '#', 1)
      continue    % ignore blank lines and comments
    end
    % process line (your code here)
  end
  fclose(fid);

Conversion from "C" to Matlab:
  copy the file xxx.c to xxx.m and do all editing in xxx.m
  Do the global substitute (replace) in the following order:
    */   to   nothing
    /*   to   %
    [    to   (
    ]    to   )
    )(   to   ,     comma
    "    to   '     apostrophe

    printf(  to  fprintf(fid,      then add  fid=fopen('xxx.out', 'w');
                                             sprintf('results in file
xxx.out')

    strip all declarations to  variable = constant;
    It will save deleting characters to substitute (replace)
    int     to  nothing
    double  to  nothing
    float   to  nothing
    char    to  nothing

    double a[20][30];  becomes   a = zeros(20,30);
                       same for all types and number of dimensions
    static int nx=11;  becomes   nx=11;
    #define nx 11      becomes   nx=11;


    I prefer to do each case individually, yet ultimately
    }   to   end
    {   to   nothing

  Statements require some change:
    if(a<b) c=d;  becomes
```

```
    if a<b
      c=d;
    end
    Thus  if(   becomes  if  followed by a space, but not whatif(x) to
whatif x)

    for(i=0; i<n; i++) {    becomes
      stuff
    }
    for i=1:n               note subscript shift, no zero subscript in
Matlab
      stuff
    end

  Functions require some change:
    void func(double x, int y)  becomes
    {
      stuff
    }
    function func(x,y)
      stuff
    end

    int func(int x, double y[]) {  becomes
      stuff
    }
    function value = func(x, y)
      stuff
      value =  % the return "C" value
    end

  In some places you may have to change  *  to   .*   and  /  to  ./

  Continuation is by default in "C", add   ...   ellipsis to unfinished
lines

  Other syntactic changes will be needed. Use Matlab editor and "lint".
    Such as eliminating  #include
    Changing # define func(x,y) ...   to  function value = func(x,y)
...


Plotting:
  given vectors x and y of same length
  plot(x,y); % generates plot
  plot(x,y1, x,y2, x,y3); % plots three curves
  plot(x,y1,'r', x,y2,'.', x,y3,'-'); % plots red, dot, dash curves

  hold on;  % makes more 'plot' commands add to same plot
  hold off; % allows new plot to start

Plot modifiers go AFTER the plot command:
  xlabel('some label for x axis');
  ylabel('y axis label');
  zlabel('if z axis used');
  axis([xmin xmax  ymin ymax]);
  axis([xmin xmax  ymin ymax  zmin zmax]);
```

```
  xlim([xmin xmax]);
  ylim([ymin ymax]);
  title(['title at top of plot, case=' int2num(n)]);
  text(x,y,'some text'); % 2D plot
  text(x,y,z, '3D text in plot');

  figure(n); % starts a new figure, typically n=1, 2, 3 ...
             % in a loop. All draw on screen in same place,
             %  move then to see earlier plots

Miscellaneous warnings:
  Beware the "workspace"
    Running a  .m  file, editing the  .m  file, running again
    will have previous results stored. e.g. the size and stuff
    in old arrays. Thus, reducing the size of the array still
    will plot the old stuff.

    Avoid this problem by always making all  .m  files start
    function some_name % optional comment would be nice
      % your MATLAB code here
    end

  Use Edit menu and occasionally clear Workspace, Command Window and
  Command History. I play it safe and usually put these two statement
  at the front of each main function:
      clear
      format compact

  If you are knowledgeable computer users, create you own directory
  or directories rather than to use MATLAB's  "work"  directory.
```

# Παράρτημα 2: Εισαγωγή στην Python

## 1. Εγκατάσταση της Python

Ο ευκολότερος τρόπος να εγκαταστήσετε την βασική έκδοση του Python software στα windows είναι να τρέξετε "all-in-one" πακέτο http://download.enthought.com/epd_free/epd_free-7.2-2-win-x86.msi. Το πακέτο αυτό σας παρέχει το βασικό περιβάλλον Python και τις βιβλιοθήκες scipy, numpy και την γραφική βιβλιοθήκη matplotlib που είναι η προτεινόμενη προσθήκη (add-on) γραφικών για το Python, η οποία είναι σχεδόν όμοια με αυτή της matlab.

Άλλες βασικές πηγές σχετικά με την Python μπορείτε να τις βρείτε στα παρακάτω links:

Επίδειξη Python.
Βασική ιστοσελίδα του Python.
Scipy και Numpy (βασικά πακέτα).
Εισαγωγή στο Numpy για χρήστες Matlab.

Το βασικό περιβάλλον python και οι βιβλιοθήκες scipy, numpy, και matplotlib είναι πραγματικά όλα όσα θα χρειαστείτε για αυτό το μάθημα. Πολλές από τις βασικές λειτουργίες για σύμβολα που χρησιμοποιούνται στα περιβάλλοντα του maple ή του matlab έχουν υλοποιηθεί στο πακέτο της python SymPy που περιέχετε στην παραπάνω εγκατάσταση. Ένα άλλο πρόγραμμα εγκατάστασης για τις ανάγκες σας είναι αυτό που δίνεται από την ActiveState.

## 2. Διαδραστικός Συντάκτης (Editor) Προγραμμάτων

Κάποιοι έχουν δικές τους προτιμήσεις όσον αφορά τους συντάκτες προγραμματισμού. Μπορείτε να χρησιμοποιήσετε αυτό που προτιμάτε, υπάρχουν όμως κάποιοι συντάκτες που διατίθονται δωρεάν οι οποίοι είναι ιδιαίτερα προσαρμοσμένοι για τις ανάγκες του περιβάλλοντος Python. Προτείνουμε το Wing IDE 101 για ανάπτυξη και διόρθωση και το Ipython για το αποτελεσματικό τρέξιμο των προγραμμάτων και τη διαδραστικότητα με το διερμηνευτής (interpreter) Python. Τα Windows έχουν το δικό τους ενσωματωμένο περιβάλλον προγραμματισμού που μπορεί να είναι αρκετό για τις περισσότερες ανάγκες.

## 3. Βοήθεια για προγραμματισμό στο Python

Πληροφορίες για το προγραμματιστικό περιβάλλον Python μπορείτε να βρείτε στα παρακάτω links:

- Επίδειξη του Python
- Βασική ιστοσελίδα του Python.
- Scipy και Numpy (βασικά πακέτα)
- Εισαγωγή στο Numpy για χρήστες Matlab.
- Εισαγωγή για προγραμματιστές σε Python

Βουτιά στο Python online βιβλίο και κώδικας
Βοήθεια για το Python για επιστήμονες
SciPy cookbook λύσεις για βασικά προβλήματα επιστημονικού προγραμματισμού
Κατάλογος αναφορών που συγκρίνουν το Python με το Matlab

## 4. Εισαγωγή στο περιβάλλον Python

Το περιβάλλον Python έχει σημαντικά πλεονεκτήματα σε σύγκριση με άλλα προγραμματιστικά περιβάλλοντα/γλώσσες προγραμματισμού:

- Το Python είναι "open source" που σημαίνει ότι είναι *ελεύθερο λογισμικό*. Συμπεριλαμβάνεται στις πιο συνηθισμένες εκδόσεις του Linux.
- Το Python είναι διαθέσιμο σε όλα τα βασικά λειτουργικά συστήματα (Linux, Unix, Windows, MacOS, κλπ). Ένα πρόγραμμα που έχει γραφτεί για ένα λειτουργικό σύστημα τρέχει χωρίς μετατροπές σε όλα τα λειτουργικά συστήματα.
- Το Python είναι ευκολότερο στην εκμάθηση και παράγει κώδικα που διαβάζεται ευκολότερα σε σχέση με άλλες γλώσσες.
- Το Python και οι επεκτάσεις του εγκαθίσταται εύκολα.

Η ανάπτυξη του Python έχει επηρεαστεί φανερά από τη Java και τη C++, έχει όμως και σημαντική ομοιότητα με το MATLAB® . Το Python υλοποιεί τις πιο βασικές έννοιες αντικειμενοστραφών γλωσσών προγραμματισμού όπως κλάση, μέθοδος, κληρονομικότητα, κλπ. Θα παραλείψουμε όλες αυτές τις έννοιες και θα χρησιμοποιήσουμε το Python σαν διαδραστική γλώσσα.

Για να καταλάβουμε τις ομοιότητες ανάμεσα στο MATLAB και το Python, ας κοιτάξουμε τον κώδικα που έχει γραφτεί στις δυο γλώσσες για τη λύση γραμμικών εξισώσεων `Ax=b` με την μέθοδο απαλοιφής του Gauss. Η συνάρτηση που έχει γραφτεί με MATLAB είναι:

```
function [x,det] = gaussElimin(a,b)
n = length(b);
for k = 1:n-1
     for i = k+1:n
          if a(i,k) ~= 0
               lam = a(i,k)/a(k,k);
               a(i,k+1:n) = a(i,k+1:n) - lam*a(k,k+1:n);
               b(i)= b(i) - lam*b(k);
          end
     end
end
det = prod(diag(a));
for k = n:-1:1
     b(k) = (b(k) - a(k,k+1:n)*b(k+1:n))/a(k,k);
end
x = b;
```

Η αντίστοιχη συνάρτηση στο Python είναι:

```
from scipy import dot
def gaussElimin(a,b):
     n = len(b)
     for k in range(0,n-1):
          for i in range(k+1,n):
               if a[i,k] != 0.0:
                    lam = a [i,k]/a[k,k]
                    a[i,k+1:n] = a[i,k+1:n] -
               lam*a[k,k+1:n]
                    b[i] = b[i] - lam*b[k]
     for k in range(n-1,-1,-1):
          b[k] = (b[k] - dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
     return b
```

Η εντολή `from numpy import dot` λέει στον διερμηνευτή της Python να φορτώσει τη συνάρτηση `dot` (που υπολογίζει το γινόμενο δυο διανυσμάτων) της ενότητας `numpy`. Η πράξη άνω κάτω τελεία (:), γνωστή στο Python ως χειριστής κοψίματος (*slicing operator)*, λειτουργεί με τον ίδιο τρόπο όπως και στο MATLAB και στην Fortran90, δηλαδή ορίζει ένα τμήμα ενός πίνακα.

Η εντολή `for k=1:n-1` στο MATLAB δημιουργεί ένα βρόγχο που εκτελείται με `k=1,2,…,n-1`. Ο ίδιος βρόχος στο Python εμφανίζεται σαν `for k in range (n-1)`. Εδώ η συνάρτηση `range(n-1)` δημιουργεί μια λίστα `[0,1,...,n-2];k` λαμβάνει τα στοιχεία της λίστας. Οι διαφορές στις τιμές του *k* στα δύο περιβάλλοντα είναι οι εγγενείς μετατοπίσεις (offset) που υποθέτουν τα δύο συστήματα. Στο Python όλες οι λίστες δεικτών έχουν *μηδενική μετατόπιση (zero offset)*, που σημαίνει ότι ο δείκτης του πρώτου στοιχείου της ακολουθίας είναι πάντα `0`. Αντίθετα, η εγγενής μετατόπιση στο MATLAB είναι 1.

Επίσης, σημειώστε ότι το Python δεν έχει δήλωση τέλους (end statements) που κλείνει ένα τμήμα κώδικα (π.χ. βρόγχους, λογικές εκφράσεις, ρουτίνες, κλπ). Το σώμα ενός τμήματος κώδικα ορίζεται από τις εσοχές του (indentation) και είναι αναπόσπαστο τμήμα της σύνταξης του Python.

Όπως και το MATLAB το Python διακρίνει τα κεφαλαία από τα μικρά γράμματα (case sensitive). Έτσι, τα ονόματα *n* και *N* αντιπροσωπεύουν διαφορετικά αντικείμενα.

## 5. Πως να αποκτήσετε το Python

Ο διερμηνευτής του Python μπορεί να κατεβεί από την ιστοσελίδα της γλώσσας Python www.python.org. Συνήθως, περιλαμβάνεται σε έναν ωραίο συντάκτη (editor) κώδικα που λέγεται *Idle* που υποστηρίζει την εκτέλεση προγραμμάτων. Για τον επιστημονικό προγραμματισμό χρειαζόμαστε επίσης και την ενότητα (module) *NumPy* που περιλαμβάνει διάφορα προγράμματα (ρουτίνες) για πράξεις πινάκων. Το περιεχόμενο της βιβλιοθήκης NumPy, μπορείτε να το πληροφορηθείτε από την ιστοσελίδα http://www.stsci.edu/resources/software_hardware/NumPy ή την εντολή help(NumPy).

Η παραπάνω ιστοσελίδα δίνει αναφορές σε σχετική βιβλιογραφία. Αν χρησιμοποιείτε Linux ή MacOS είναι πολύ πιθανό το Python να είναι ήδη εγκατεστημένο στον υπολογιστή σας (όμως πρέπει να κατεβάσετε και σε αυτή την περίπτωση το NumPy). Για την περίπτωση των windows το πακέτο www.enthought.com περιλαμβάνει την

παραπάνω βιβλιοθήκη. Επίσης, θα χρειαστείτε την βιβλιοθήκη numarray η οποία έχει αντικατασταθεί από την scipy.

Καλά είναι να αποκτήσετε και επιπλέον έντυπο υλικό για συμπλήρωση της βιβλιογραφίας που θα βρείτε στο διαδίκτυο.

## 6. Το βασικό Python

## 6.1 Μεταβλητές

Στις περισσότερες γλώσσες προγραμματισμού το όνομα μιας μεταβλητής αντιπροσωπεύει την τιμή ενός συγκεκριμένου τύπου που αποθηκεύεται σε συγκεκριμένη θέση μνήμης. Η τιμή μπορεί να αλλάξει, αλλά όχι ο τύπος. Αυτό δεν ισχύει στο Python, όπου οι μεταβλητές *αποκτούν τύπο δυναμικά*. Η επόμενη επαναλαμβανόμενη εντολή του διερμηνευτή του Python δείχνει ακριβώς αυτό (>>> είναι ο δείκτης δρομέα ( `prompt`) στο Python):

```
>>> b = 2 # b is integer type
>>> print b
2
>>> b = b * 2.0 # Now b is float type
>>> print b
4.0
```

Η καταχώρηση `b=2` δημιουργεί σύνδεση ανάμεσα στο όνομα `b` και στην ακέραια (*integer)* τιμή `2`. Η επόμενη δήλωση υπολογίζει το `b * 2.0` και συνδέει το αποτέλεσμα με το `b`. Η αρχική σύνδεση του *ακεραίου* `2` καταστρέφεται. Τώρα το b αναφέρεται στην *πραγματική τιμή– κινητής υποδιαστολής* `4.0`.

Το (#) δείχνει την αρχή ενός *σχολίου* - όλοι οι χαρακτήρες ανάμεσα στο # και το τέλος της γραμμής αγνοούνται από τον διερμηνευτή.

## 6.2 Συμβολοσειρές (Strings)

Μια συμβολοσειρά (string) είναι μια σειρά χαρακτήρων που περιλαμβάνεται ανάμεσα σε μονά ή διπλά εισαγωγικά. Οι συμβολοσειρές *ενώνονται* με την πράξη +, ενώ το *κοφτήρι (:)* χρησιμοποιείται για να αποσπάσει κομμάτι μιας συμβολοσειράς. Για παράδειγμα:

```
>>> string1 = 'Press return to exit'
>>> string2 = 'the program'
>>> print string1 + ' ' + string2 # Concatenation
Press return to exit the program
>>> print string1[0:12] # Slicing
Press return
```

Η συμβολοσειρά είναι αντικείμενο που είναι αμετάτρεπτο *(immutable)* – οι χαρακτήρες του δεν μπορούν να μετατραπούν με εντολή ανάθεσης και έχει ένα ορισμένο μήκος. Προσπάθεια να παραβιαστεί η ιδιότητα της μη μετάλλαξης θα οδηγήσει σε `TypeError` όπως φαίνεται παρακάτω:

```
>>> s = 'Press return to exit'
>>> s[0] = 'p'
Traceback (most recent call last):
     File ''<pyshell#1>'', line 1, in ?
          s[0] = 'p'
TypeError: object doesn't support item assignment
```

## 6.3 Πλειάδες (Tuples)

Μια πλειάδα (*tuple)* είναι μια σειρά από *αυθαίρετα αντικείμενα* τα οποία διαχωρίζονται με κόμμα και περικλείονται με παρενθέσεις. Αν μια πλειάδα περιλαμβάνει ένα και μόνο αντικείμενο, οι παρενθέσεις μπορούν να παραληφθούν. Οι πλειάδες υποστηρίζουν τις ίδιες πράξεις με τις συμβολοσειρές. Επίσης, οι πλειάδες δεν μπορούν να μεταλλαχθούν. Εδώ δίνεται ένα παράδειγμα όπου μια πλειάδα ονόματι `rec` περιέχει μια άλλη πλειάδα `(6,23,68)`:

```
>>> rec = ('Smith','John',(6,23,68)) # This is a tuple
>>> lastName,firstName,birthdate = rec # Unpacking the
tuple
>>> print firstName
John
>>> birthYear = birthdate[2]
>>> print birthYear
68
>>> name = rec[1] + ' ' + rec[0]
>>> print name
John Smith
>>> print rec[0:2]
('Smith', 'John')
```

## 6.4 Λίστες

Μια λίστα είναι παρόμοια με μια πλειάδα, όμως *μπορεί να μεταλλαχθεί (is mutable)*, έτσι ώστε τα στοιχεία της και το μήκος της μπορούν να μετατραπούν. Μια λίστα προσδιορίζεται με το κλείσιμο της σε τετραγωνισμένες παρενθέσεις (brackets). Εδώ δίνονται δειγματοληπτικά πράξεις που μπορούν να εφαρμοστούν σε λίστες:

```
>>> a = [1.0, 2.0, 3.0] # Create a list
>>> a.append(4.0) # Append 4.0 to list
>>> print a
[1.0, 2.0, 3.0, 4.0]
```

```
>>> a.insert(0,0.0) # Insert 0.0 in position 0
>>> print a
[0.0, 1.0, 2.0, 3.0, 4.0]
>>> print len(a) # Determine length of list
5
>>> a[2:4] = [1.0, 1.0] # Modify selected elements
>>> print a
[0.0, 1.0, 1.0, 1.0, 1.0, 4.0]
```

Αν το *a* είναι ένα μεταλλάξιμο αντικείμενο, όπως μια λίστα, η εντολή ανάθεσης `b = a` δεν οδηγεί σε ένα νέο αντικείμενο *b* αλλά απλά δημιουργεί αναφορά στο *a*. Έτσι, οποιεσδήποτε αλλαγές στο `b` αντιπροσωπεύονται και στο a. Για να δημιουργηθεί ένα νέο ανεξάρτητο αντίγραφο του *a*, χρησιμοποιήστε την εντολή `c = a[:]` όπως φαίνεται παρακάτω.

| | |
|---|---|
| + | Πρόσθεση |
| - | Αφαίρεση |
| * | Πολλαπλασιασμός |
| / | Διαίρεση |
| ** | Ύψωση σε δύναμη |
| % | Διαίρεση (modular division) |

```
>>> a = [1.0, 2.0, 3.0]
>>> b = a # 'b' is an alias of 'a'
>>> b[0] = 5.0 # Change 'b'
>>> print a
[5.0, 2.0, 3.0] # The change is
reflected in 'a'
>>> c = a[:] # 'c' is an independent
copy of 'a'
>>> c[0] = 1.0 # Change 'c'
>>> print a
[5.0, 2.0, 3.0] # 'a' is not affected by the change
```

Πίνακες μπορούν να αντιπροσωπευθούν σαν ένθετες (nested) λίστες στις οποίες κάθε σειρά είναι ένα στοιχείο της λίστας:

```
>>> a = [[1, 2, 3], \
        [4, 5, 6], \
        [7, 8, 9]]
>>> print a[1] # Print second row (element 1)
[4, 5, 6]
>>> print a[1][2] # Print third element of second row
6
```

Η ανάποδη κάθετος (backslash \) είναι ο *χαρακτήρας συνέχειας (continuation character)* του Python. Θυμηθείτε ότι οι ακολουθίες του Python έχουν μηδενική μετατόπιση έτσι ώστε το `a[0]` να αναφέρεται στο πρώτο όρο της ακολουθίας , το `a[1]` στο δεύτερο όρο, κλπ. Δεν χρησιμοποιούμε λίστες για αριθμητικούς πίνακες εκτός από πολύ λίγες εξαιρέσεις. Είναι πολύ πιο πρακτικό να χρησιμοποιούμε *πίνακες* που δίνονται από την ενότητα (module) `numpy`. Οι πίνακες θα συζητηθούν αργότερα.

Αριθμητικές Πράξεις

Το Python υποστηρίζει τις συνήθεις αριθμητικές πράξεις:

Μερικές από αυτές τις πράξεις ορίζονται επίσης και σε συμβολοσειρές και σειρές (ακολουθίες) όπως φαίνεται παρακάτω.

```
>>> s = 'Hello '
>>> t = 'to you'
>>> a = [1, 2, 3]
>>> print 3*s # Repetition
Hello Hello Hello
>>> print 3*a # Repetition
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> print a + [4, 5] # Append elements
[1, 2, 3, 4, 5]
>>> print s + t # Concatenation
Hello to you
>>> print 3 + s # This addition makes no sense
Traceback (most recent call last):
     File ''<pyshell#9>'', line 1, in ?
          print n + s
TypeError: unsupported operand types for +: 'int' and 'str'
```

Το Python 2.0 και μεταγενέστερες εκδόσεις έχουν επίσης και *ενισχυμένες πράξεις ανάθεσης (augmented assignment operators)*, όπως `a+=b`, που είναι γνωστές στους χρήστες της C. Οι ενισχυμένες πράξεις και αντίστοιχες αριθμητικές εκφράσεις φαίνονται στον παρακάτω πίνακα:

| | |
|---|---|
| `a += b` | `a = a + b` |
| `a -= b` | `a = a - b` |
| `a *= b` | `a = a*b` |
| `a /= b` | `a = a/b` |
| `a **= b` | `a = a**b` |
| `a %= b` | `a = a%b` |

## 6.6 Πράξεις Σύγκρισης

Οι πράξεις σύγκρισης επιστρέφουν 1 για true και 0 για false. Οι πράξεις αυτές είναι

| | |
|---|---|
| < | Λιγότερο |
| > | Περισσότερο |
| <= | Λιγότερο ή ίσο |

| | |
|---|---|
| >= | Περισσότερο ή ίσο |
| == | Ίσο |
| != | Άνισο |

Οι αριθμοί με διαφορετικό τύπο (integer, floating point, κλπ) μετατρέπονται σε κοινό τύπο πριν τη σύγκριση. Διαφορετικά αντικείμενα με διαφορετικούς τύπους θεωρούνται διαφορετικά μεταξύ τους. Εδώ υπάρχουν μερικά παραδείγματα:

```
>>> a = 2 # Integer
>>> b = 1.99 # Floating point
>>> c = '2' # String
>>> print a > b
1
>>> print a == c
0
>>> print (a > b) and (a != c)
1
>>> print (a > b) or (a == b)
1
```

## 6.7 Λογικές εκφράσεις (Conditionals)

Η δομή `if`

```
if όρος:
     σώμα
```

Εκτελεί ένα σώμα εντολών (που πρέπει να παρουσιαστεί σε μορφή εσοχής) αν ο όρος επιστρέφει true. Αν ο όρος επιστρέφει false, το σώμα παραλείπεται. Το `if` μπορεί να ακολουθηθεί από ένα οποιοδήποτε αριθμό από `elif` (συντόμευση για το else if)

```
elif όρος:
     σώμα
```

που λειτουργούν με τον ίδιο τρόπο. Η πρόταση `else`

```
else:
     σώμα
```

να εκτελεστούν αν καμία από τις προτάσεις `if-elif` δεν είναι true. Η συνάρτηση `sign_of_a_below` δείχνει τη χρήση των λογικών εντολών:

```
def sign_of_a(a):
     if a < 0.0:
```

```
          sign = 'negative'
     elif a > 0.0:
          sign = 'positive'
     else:
          sign = 'zero'
     return sign
a = 1.5
print 'a is ' + sign_of_a(a)
```

Η εκτέλεση του προγράμματος δίνει το αποτέλεσμα

```
a is positive
```

## 6.8 Βρόχοι

Η δομή `while`

```
while όρος:
     σώμα
```

εκτελεί ένα σώμα εντολών που έχει παρατεθεί στο πρόγραμμα σε εσοχή (indented) εφόσον ο "όρος" είναι true. Μετά την εκτέλεση του σώματος, ο "όρος" αξιολογείται ξανά. Αν εξακολουθεί "όρος" να είναι true, το σώμα εκτελείται ξανά. Η διαδικασία αυτή επαναλαμβάνεται μέχρι ο "όρος" να γίνει false. Η πρόταση `else`

```
else:
     σώμα
```

μπορεί να χρησιμοποιηθεί για να οριστεί το σώμα των εντολών που πρέπει να εκτελεστεί εφόσον ο "όρος" είναι false. Εδώ φαίνεται ένα παράδειγμα που δημιουργεί τη λίστα

[1, 1/2, 1/3, ...]:

```
nMax = 5
n = 1
a = [] # Create empty list
while n < nMax:
     a.append(1.0/n) # Append element to list
     n = n + 1
print a
```

Το αποτέλεσμα του προγράμματος είναι

```
[1.0, 0.5, 0.33333333333333331, 0.25]
```

Συναντήσαμε ήδη την εντολή `for`. Η εντολή αυτή απαιτεί ένα στόχο και μία σειρά (συνήθως λίστα) πάνω στην οποία ο στόχος επαναλαμβάνεται. Η μορφή της δομής είναι

```
for στόχος in σειρά:
     σώμα
```

Μπορείτε να προσθέσετε μια πρόταση `else` που εκτελείται όταν ο βρόχος `for` έχει τελειώσει. Το προηγούμενο πρόγραμμα θα μπορούσε να γραφτεί με δομή `for` ως

```
nMax = 5
a = []
for n in range(1,nMax):
     a.append(1.0/n)
print a
```

Εδώ *n* είναι ο στόχος και η λίστα είναι το `[1,2, ...,nMax-1]`, που δημιουργείται καλώντας τη συνάρτηση `range`.

Οποιοσδήποτε βρόχος μπορεί να τερματιστεί με μια εντολή `break`. Αν υπάρχει πρόταση `else` που σχετίζεται με το βρόγχο δεν εκτελείται. Το παρακάτω πρόγραμμα, που ψάχνει για ένα όνομα σε μια λίστα δείχνει τη χρήση του `break` και του `else` σε συνδυασμό με ένα βρόγχο `for`:

```
list = ['Jack', 'Jill', 'Tim', 'Dave']
name = eval(raw_input('Type a name: ')) # Python input
prompt
for i in range(len(list)):
     if list[i] == name:
          print name,'is number',i + 1,'on the list'
     break
else:
     print name,'is not on the list'
```

Αυτά είναι τα αποτελέσματα εκτέλεσης του προγράμματος:

```
Type a name: 'Tim'
Tim is number 3 on the list
Type a name: 'June'
June is not on the list
```

## 6.9 Μετατροπή Τύπου

Αν μια αριθμητική πράξη αφορά αριθμούς με διαφορετικούς τύπους, οι αριθμοί αυτόματα μετατρέπονται σε κοινό τύπο πριν εκτελεστεί η εντολή. Οι μετατροπές τύπων μπορούν να επιτευχθούν με τις παρακάτω συναρτήσεις:

| | |
|---|---|
| `int(a)` | Μετατρέπει το *a* σε integer |
| `long(a)` | Μετατρέπει το *a* σε long integer |
| `float(a)` | Μετατρέπει το *a* σε floating point |
| `complex(a)` | Μετατρέπει το a σε μιγαδικό *a+0j* |
| `complex(a,b)` | Μετατρέπει το a και b σε μιγαδικό *a+bj* |

Οι παραπάνω συναρτήσεις δουλεύουν επίσης και για μετατροπή συμβολοσειρών (string) σε αριθμό εφόσον ο χαρακτήρας αντιστοιχεί σε αριθμό. Μετατροπή από πραγματικό (float) σε ακέραιο (integer) γίνεται μέσω αποκοπής (truncation), και όχι μέσω στρογγυλοποίησης. Εδώ είναι μερικά παραδείγματα:

```
>>> a = 5
>>> b = -3.6
>>> d = '4.0'
>>> print a + b
1.4
>>> print int(b)
-3
>>> print complex(a,b)
(5-3.6j)
>>> print float(d)
4.0
>>> print int(d) # This fails: d is not Int type
Traceback (most recent call last):
     File ''<pyshell#7>'', line 1, in ?
          print int(d)
ValueError: invalid literal for int(): 4.0
```

## 6.10 Μαθηματικές Συναρτήσεις

Το βασικό περιβάλλον Python υποστηρίζει μόνο μερικές μαθηματικές συναρτήσεις που είναι οι εξής:

| | |
|---|---|
| `abs(a)` | Απόλυτη τιμή του `a` |
| `max(`*σειρά*`)` | Το μεγαλύτερο στοιχείο της *σειράς* |
| `min(`*σειρά*`)` | Το μικρότερο στοιχείο της *σειράς* |
| `round(a,n)` | Στρογγυλοποίηση του a σε n δεκαδικά |
| `cmp(a,b)` | Επιστρέφει:<br>-1 αν `a < b`<br>0 αν `a = b`<br>1 αν `a > b` |

Οι περισσότερες μαθηματικές συναρτήσεις είναι διαθέσιμες μέσα από την ενότητα `math`.

## 6.11 Διάβασμα Εισόδου

Η βασική συνάρτηση για να δεχθεί ένα πρόγραμμα είσοδο από το χρήστη είναι

```
raw_input(prompt)
```

δείχνει το prompt και μετά διαβάζει μια γραμμή εισόδου που μετατρέπεται σε ακολουθία χαρακτήρων. Η μετατροπή μια ακολουθίας χαρακτήρων σε αριθμητική τιμή χρησιμοποιεί την συνάρτηση:

```
eval(string)
```

Το παρακάτω πρόγραμμα δείχνει τη χρήση αυτών των συναρτήσεων:

```
a = raw_input('Input a: ')
print a, type(a) # Print a and its type
b = eval(a)
print b,type(b) # Print b and its type
```

Η συνάρτηση `type(a)` επιστρέφει την τιμή του αντικειμένου `a`. Είναι πολύ χρήσιμο εργαλείο για την αναζήτηση λαθών(debugging). Το παρακάτω πρόγραμμα έτρεξε δυο φορές με τα εξής αποτελέσματα:

```
Input a: 10.0
10.0 <type 'str'>
10.0 <type 'float'>

Input a: 11**2
11**2 <type 'str'>
121 <type 'int'>
```

Ένας εύκολος τρόπος για να εισαχθεί ένα αριθμός και να αντιστοιχιστεί στη μεταβλητή *a* είναι:

```
a = eval(raw_input(prompt))
```

## 6.12 Τύπωση Εξόδου

Η έξοδος μπορεί να επιδειχθεί με την εντολή `print`:

```
print αντικείμενο1, αντικείμενο2, …
```

που μετατρέπει τα αντικείμενα *αντικείμενο1, αντικείμενο2,* κλπ σε ακολουθίες χαρακτήρων και τα τυπώνει στην ίδια γραμμή διαχωρισμένα με κενά. Ο χαρακτήρας νέα γραμμή (*newline)* `'\n'` μπορεί να χρησιμοποιηθεί για να δημιουργήσει μια νέα γραμμή. Για παράδειγμα,

```
>>> a = 1234.56789
>>> b = [2, 4, 6, 8]
>>> print a,b
1234.56789 [2, 4, 6, 8]
>>> print 'a =',a, '\nb =',b
a = 1234.56789
b = [2, 4, 6, 8]
```

Η πράξη % μπορεί να χρησιμοποιηθεί για να δημιουργήσει μια πλειάδα (tuple). Η μορφή της εντολής μετατροπής είναι

```
'%μορφή1 %μορφή2…' % tuple
```

όπου *μορφή1, μορφή2, ...* είναι οι προσδιορισμοί μορφής για κάθε αντικείμενο στις πλειάδες (tuple). Οι συνήθεις προσδιορισμοί μορφής είναι

| | |
|---|---|
| wd | Integer |
| w.df | Floating point |
| w.de | Δύναμη |

όπου *w* είναι το πλάτος του πεδίου και *d* είναι ο αριθμός των δεκαδικών ψηφίων. Η έξοδος στοιχίζεται δεξιά στο προσδιορισμένο πεδίο και γεμίζεται με κενά (υπάρχει δυνατότητα αλλαγής της στοίχισης και του γεμίσματος). Εδώ είναι μερικά παραδείγματα:

```
>>> a = 1234.56789
>>> n = 9876
>>> print '%7.2f' % a
1234.57
>>> print 'n = %6d' % n # Pad with 2 spaces
n = 9876
>>> print 'n = %06d' %n # Pad with 2 zeroes
n = 009876
>>> print '%12.4e %6d' % (a,n)
 1.2346e+003 9876
```

## 6.13 Έλεγχος Λαθών

Όταν γίνεται ένα λάθος κατά την εκτέλεση ενός προγράμματος εγείρεται μια *εξαίρεση (exception)*. Εξαιρέσεις μπορούν να πιαστούν με εντολές try και except

```
try:
     κάνε κάτι
except λάθος:
     κάνε κάτι άλλο
```

όπου *λάθος* είναι το όνομα μιας προ-δημιουργημένης εξαίρεσης του Python. Αν η εξαίρεση *λάθος* δεν εγερθεί εκτελείται το σώμα try. Αλλιώς εκτελείται το σώμα except. Όλες οι εξαιρέσεις μπορούν να πιαστούν παραλείποντας το *λάθος* από την εντολή except.

Εδώ βρίσκεται μια εντολή που εγείρει την εξαίρεση ZeroDivisionError:

```
>>> c = 12.0/0.0
Traceback (most recent call last):
```

```
     File ''<pyshell#0>'', line 1, in ?
          c = 12.0/0.0
ZeroDivisionError: float division
```

Το λάθος μπορεί να πιαστεί με

```
try:
     c = 12.0/0.0
except ZeroDivisionError:
     print 'Division by zero'
```

## 7. Συναρτήσεις και Ενότητες (Modules)

### Συναρτήσεις

Η δομή μιας συνάρτησης στο Python είναι

```
def όνομα_συνάρτησης (παράμετρος, παράμετρος2,…):
     εντολές
     return τιμές_επιστροφής
```

όπου *παράμετρος1, παράμετρος2, …* είναι οι παράμετροι. Μια παράμετρος μπορεί να είναι οποιοδήποτε αντικείμενο του Python, συμπεριλαμβανομένης μιας συνάρτησης. Οι παράμετροι μπορούν να πάρουν προεπιλεγμένες τιμές (default values). Στην περίπτωση αυτή, η παράμετρος της συνάρτησης είναι προαιρετική. Αν η εντολή `return` ή οι *τιμές_επιστροφής* παραληφθούν, η συνάρτηση επιστρέφει το αντικείμενο null.

Το παρακάτω παράδειγμα υπολογίζει τα δύο πρώτα παράγωγα του `arctan(x)` με πεπερασμένες διαφορές:

```
from math import arctan
def finite_diff(f,x,h=0.0001): # h has a default value
     df =(f(x+h) - f(x-h))/(2.0*h)
     ddf =(f(x+h) - 2.0*f(x) + f(x-h))/h**2
     return df,ddf
x = 0.5
df,ddf = finite_diff(arctan,x) # Uses default value of h
print 'First derivative =',df
print 'Second derivative =',ddf
```

Σημειώστε ότι στο `arctan` δίνεται σαν παράμετρος το `finite_diff`. Η έξοδος του προγράμματος είναι

```
First derivative = 0.799999999573
Second derivative = -0.639999991892
```

Αν ένα αντικείμενο που μπορεί να μεταλλαχθεί, όπως μια λίστα, περαστεί σε μια συνάρτηση σαν παράμετρος και το μετατρέψει, οι αλλαγές θα εμφανιστούν στο πρόγραμμα που την έχει καλέσει. Αυτό μπορούμε να το δούμε στο παράδειγμα:

```
def squares(a):
     for i in range(len(a)):
          a[i] = a[i]**2
a = [1, 2, 3, 4]
squares(a)
print a
```

Η έξοδος είναι:

```
[1, 4, 9, 16]
```

## 8. Μαθηματικές Ενότητες (Mathematical Modules)

## 8.

### 1. Η ενότητα math

Οι περισσότερες μαθηματικές συναρτήσεις δεν υπάρχουν στο βασικό Python αλλά γίνονται διαθέσιμες με τη φόρτωση της ενότητας (module) `math`. Υπάρχουν 3 τρόποι για να αποκτήσει κανείς πρόσβαση στην ενότητα. Η εντολή

```
from math import *
```

Φορτώνει *όλους* τους ορισμούς συναρτήσεων από την ενότητα `math` στην τρέχουσα συνάρτηση. Η χρήση αυτής της μεθόδου δε συνιστάται γιατί όχι μόνο είναι σπάταλη αλλά μπορεί να οδηγήσει και σε συγκρούσεις με ορισμούς που έχουν φορτωθεί από άλλες ενότητες.

Μπορείτε να φορτώσετε επιλεγμένους ορισμούς με το

```
from math import συνάρτηση1, συνάρτηση2,…
```

όπως φαίνεται παρακάτω:

```
>>> from math import log,sin
>>> print log(sin(0.5))
-0.735166686385
```

Η τρίτη μέθοδος, που χρησιμοποιείται από τους περισσότερους προγραμματιστές, είναι να κάνει κανείς την ενότητα διαθέσιμη μέσω

```
import math
```

Οι συναρτήσεις στην ενότητα μπορούν τότε να χρησιμοποιηθούν με τη χρήση του ονόματος της ενότητας σαν πρόθεμα:

```
>>> import math
>>> print math.log(math.sin(0.5))
-0.735166686385
```

Τα περιεχόμενα της ενότητας μπορούν να εκτυπωθούν με κλήση του `dir(module)`. Εδώ φαίνεται πως μπορεί κανείς να αποκτήσει μια λίστα των συναρτήσεων της ενότητας `math`:

```
>>> import math
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan',
 'atan2', 'ceil', 'cos', 'cosh', 'e', 'exp', 'fabs',
 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
 'log10', 'modf', 'pi', 'pow', 'sin', 'sinh', 'sqrt',
 'tan', 'tanh']
```

Οι περισσότερες συναρτήσεις είναι γνωστές στους προγραμματιστές. Σημειώστε ότι η ενότητα περιλαμβάνει δυο σταθερές: $\pi$ και $e$.

## 2. Ενότητα cmath

Η ενότητα `cmath` δίνει πολλές συναρτήσεις που βρίσκονται στην ενότητα `math` αλλά αυτές δέχονται μιγαδικούς αριθμούς. Οι συναρτήσεις της ενότητας είναι:

```
['__doc__', '__name__', 'acos', 'acosh', 'asin', 'asinh',
 'atan', 'atanh', 'cos', 'cosh', 'e', 'exp', 'log',
 'log10', 'pi', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']
```

Εδώ είναι παραδείγματα αριθμητικής με μιγαδικούς αριθμούς:

```
>>> from cmath import sin
>>> x = 3.0 -4.5j
>>> y = 1.2 + 0.8j
>>> z = 0.8
>>> print x/y
(-2.56205313375e-016-3.75j)
>>> print sin(x)
(6.35239299817+44.5526433649j)
>>> print sin(z)
(0.7173560909+0j)
```

**1.**

**2.**

**3.**

**4.**

**5.**

**6.**

**7.**

**8.**

**8.1**

**8.2**

## *8.3*Η Ενότητα *numpy*

**1.**

**2.**

**3.**

**4.**

**5.**

**6.**

**7.**

**8.**

## 8.4Γενικές Πληροφορίες

Η ενότητα `numpy` δεν είναι κομμάτι της τυποποιημένης έκδοσης του Python. Όπως ειπώθηκε παραπάνω, πρέπει κανείς να την βρει ανεξάρτητα και να την εγκαταστήσει (η εγκατάσταση είναι εύκολη). Η ενότητα εισάγει *πίνακες* (*array objects)* που είναι παρόμοιες με τις λίστες αλλά μπορεί κανείς να τους επεξεργαστεί με πολλές συναρτήσεις που συμπεριλαμβάνονται στην ενότητα. Το μέγεθος ενός πίνακα δεν μπορεί να μεταλλαχθεί και δεν επιτρέπονται κενά στοιχεία.

Η πλήρης λίστα συναρτήσεων στο `numpy` είναι πολύ μακριά για να δοθεί στον ολότητα της. Η παρακάτω λίστα περιορίζεται μόνο στις πιο συνήθεις συναρτήσεις.

```
['Complex', 'Complex32', 'Complex64', 'Float',
'Float32', 'Float64', 'abs', 'arccos',
'arccosh', 'arcsin', 'arcsinh', 'arctan',
'arctan2', 'arctanh', 'argmax', 'argmin',
'cos', 'cosh', 'diagonal', 'dot', 'e', 'exp',
'floor', 'identity', 'innerproduct', 'log',
'log10', 'matrixmultiply', 'maximum', 'minimum',
'numarray', 'ones', 'pi', 'product' 'sin', 'sinh',
'size', 'sqrt', 'sum', 'tan', 'tanh', 'trace',
'transpose', 'zeros']
```

### 9. Δημιουργία ενός Πίνακα

Πίνακες μπορούν να δημιουργηθούν με πολλούς τρόπους. Ένας από αυτούς είναι να χρησιμοποιηθεί η συνάρτηση `array` που μετατρέπει μια λίστα σε πίνακα :

```
array (λίστα, type = ορισμός_τύπου)
```

Εδώ είναι δυο παραδείγματα που δημιουργούν ένα πίνακα 2x2 με στοιχεία πραγματικούς (floating point):

```
>>> from numarray import array,Float
>>> a = array([[2.0, -1.0],[-1.0, 3.0]])
>>> print a
[[ 2. -1.]
 [-1. 3.]]
>>> b = array([[2, -1],[-1, 3]],type = Float)
>>> print b
[[ 2. -1.]
 [-1. 3.]]
```

Άλλες διαθέσιμες συναρτήσεις είναι

```
zeros ((διάσταση1, διάσταση2), type = ορισμός_τύπου)
```

```
ones ((διάσταση1, διάσταση2), type = ορισμός_τύπου)
```

που γεμίζει τον πίνακα με 1. Ο προεπιλεγμένος τύπος είναι και στις δυο περιπτώσεις το `Int`.

Τέλος, υπάρχει η συνάρτηση

```
arrange(από, μέχρι, αύξηση)
```

που λειτουργεί όπως και η συνάρτηση `range`, αλλά επιστρέφει πίνακα αντί για λίστα. Εδώ είναι παραδείγματα για δημιουργία πινάκων:

```
>>> from numpy import arange,zeros,ones,Float
>>> a = arange(2,10,2)
>>> print a
[2 4 6 8]
>>> b = arange(2.0,10.0,2.0)
>>> print b
[ 2. 4. 6. 8.]
>>> z = zeros((4))
>>> print z
[0 0 0 0]
>>> y = ones((3,3),type= Float)
>>> print y
[[ 1. 1. 1.]
 [ 1. 1. 1.]
 [ 1. 1. 1.]]
```

## 9.1Πρόσβαση και Αλλαγή Στοιχείων Πινάκων

Αν το *a* είναι ένας πίνακας δευτέρας τάξης (rank-2) τότε το `a[i,j]` δίνει πρόσβαση στο στοιχείο της γραμμής `i` και στήλης `j`, όπου `a[i]` αναφέρεται στη γραμμή i. Τα στοιχεία ενός πίνακα μπορούν να αλλαχθούν με ανάθεση όπως φαίνεται παρακάτω.

```
>>> from numarray import *
>>> a = zeros((3,3),type=Float)
>>> a[0] = [2.0, 3.1, 1.8] # Change a row
>>> a[1,1] = 5.2 # Change an element
>>> a[2,0:2] = [8.0, -3.3] # Change part of a row
>>> print a
[[ 2. 3.1 1.8]
 [ 0. 5.2 0. ]
 [ 8. -3.3 0. ]]
```

## 9.2Πράξεις Πινάκων

Οι αριθμητικές πράξεις δουλεύουν διαφορετικά σε πίνακες από ότι σε πλειάδες και λίστες – η πράξη *μεταδίδεται* (is *broadcast*) σε όλα τα στοιχεία του πίνακα. Δηλαδή, η πράξη εφαρμόζεται σε κάθε στοιχείο του πίνακα. Μερικά παραδείγματα:

```
>>> from numpy import array
>>> a = array([0.0, 4.0, 9.0, 16.0])
>>> print a/16.0
[ 0. 0.25 0.5625 1. ]
>>> print a - 4.0
[ -4. 0. 5. 12.]
```

Οι μαθηματικές συναρτήσεις που είναι διαθέσιμες μέσω του `numpy` επίσης μεταδίδονται, όπως φαίνεται παρακάτω

```
>>> from numpy import array,sqrt,sin
>>> a = array([1.0, 4.0, 9.0, 16.0])
```

```
>>> print sqrt(a)
[ 1. 2. 3. 4.]
>>> print sin(a)
[ 0.84147098 -0.7568025 0.41211849 -0.28790332]
```

Συναρτήσεις που εισάγονται μέσα από την ενότητα `math` μπορούν να χρησιμοποιηθούν σε ανεξάρτητα στοιχεία αλλά όχι στον ίδιο τον πίνακα. Εδώ είναι ένα παράδειγμα:

```
>>> from numpy import array
>>> from math import sqrt
>>> a = array([1.0, 4.0, 9.0, 16.0])
>>> print sqrt(a[1])
2.0
>>> print sqrt(a)
Traceback (most recent call last):
...
TypeError: Only rank-0 arrays can be cast to floats.
```

## 9.3Συναρτήσεις Πινάκων

Υπάρχουν πολλές συναρτήσεις πινάκων στο `numpy` που εκτελούν πράξεις πινάκων. Εδώ είναι μερικά παραδείγματα:

```
>>> from numarray import *
>>> a = array([[ 4.0, -2.0, 1.0], \
              [-2.0, 4.0, -2.0], \
              [ 1.0, -2.0, 3.0]])
>>> b = array([1.0, 4.0, 2.0])
>>> print dot(b,b) # Dot product
21.0
>>> print matrixmultiply(a,b) # Matrix multiplication
[ -2. 10. -1.]
>>> print diagonal(a) # Principal diagonal
[ 4. 4. 3.]
>>> print diagonal(a,1) # First subdiagonal
[-2. -2.]
>>> print trace(a) # Sum of diagonal elements
11.0
>>> print argmax(b) # Index of largest element
1
>>> print identity(3) # Identity matrix
[[1 0 0]
 [0 1 0]
 [0 0 1]]
```

## 9.4Αντιγραφή Πινάκων

Εξηγήσαμε παραπάνω ότι αν το `a` είναι ένα αντικείμενο που μπορεί να μεταλλαχθεί, όπως μια λίστα, η εντολή ανάθεσης `b = a` δεν έχει σαν αποτέλεσμα ένα νέο αντικείμενο *b*, αλλά απλά δημιουργεί μια νέα αναφορά στο *a*, που ονομάζεται *deep*

*copy*. Αυτό ισχύει και για τους πίνακες. Για να δημιουργήσει κανείς ένα ανεξάρτητο αντίγραφο του πίνακα `a` πρέπει να χρησιμοποιήσει τη μέθοδο `copy` της ενότητας `numpy`:

```
a = a.copy()
```

### 10. Πεδίο (Scoping) Μεταβλητών

Ο *χώρος ονομάτων (namespace)* είναι ένα λεξικό που περιέχει τα ονόματα των μεταβλητών και τις τιμές τους. Οι χώροι ονομάτων δημιουργούνται και ανανεώνονται αυτόματα καθώς τρέχει ένα πρόγραμμα. Υπάρχουν τρία επίπεδα χώρων ονομάτων στο Python:

- Τοπικός χώρος ονομάτων, που δημιουργείται όταν καλείται μια συνάρτηση. Περιέχει μεταβλητές που περνιούνται στη συνάρτηση σαν παράμετροι και μεταβλητές που δημιουργούνται μέσα στη συνάρτηση. Ο χώρος ονομάτων καταστρέφεται όταν τελειώσει η εκτέλεση της συνάρτησης. Αν μια μεταβλητή δημιουργείται μέσα σε μια συνάρτηση, το πεδίο της είναι ο τοπικός χώρος ονομάτων της συνάρτησης. Δεν είναι ορατή έξω από τη συνάρτηση.
- Ένας γενικός χώρος ονομάτων δημιουργείται όταν φορτώνεται μια ενότητα. Κάθε ενότητα έχει το δικό της χώρο ονομάτων. Μεταβλητές που ανατίθενται μέσα σε ένα γενικό χώρο ονομάτων δεν είναι ορατές από καμία συνάρτηση μέσα στην ενότητα.
- Ένας ενσωματωμένος χώρος ονομάτων δημιουργείται όταν ξεκινά ο interpreter. Περιέχει τις συναρτήσεις που συμπεριλαμβάνονται στον interpreter του Python. Σε αυτές οι συναρτήσεις μπορεί να έχει πρόσβαση οποιαδήποτε μονάδα προγράμματος.

Όταν ένα όνομα συναντάται κατά τη διάρκεια εκτέλεσης μιας συνάρτησης, ο interpreter προσπαθεί να το επιλύσει ψάχνοντας με την εξής σειρά: (1) στον τοπικό χώρο ονομάτων (2) στο γενικό χώρο ονομάτων, και (3) στον ενσωματωμένο χώρο ονομάτων. Αν ένα όνομα δεν μπορεί να επιλυθεί το Python εγείρει την εξαίρεση `NameError`.

Εφόσον οι μεταβλητές που βρίσκονται στο γενικό χώρο ονομάτων είναι ορατές σε συναρτήσεις μέσα στην ενότητα, δεν είναι απαραίτητο να περαστούν στη συνάρτηση σαν παράμετροι, όπως δείχνει το επόμενο πρόγραμμα.

```
def divide():
     c = a/b
     print 'a/b =',c
a = 100.0
b = 5.0
divide()
>>>
   a/b = 20.0
```

Σημειώστε, ότι η μεταβλητή `c` δημιουργείται μέσα στη συνάρτηση `divide` και γι' αυτό το λόγο δεν είναι προσβάσιμη από εντολές εκτός της συνάρτησης. Άρα μια προσπάθεια για να μεταφερθεί η εντολή `print` εκτός της συνάρτησης αποτυχαίνει:

```
def divide():
     c = a/b
a = 100.0
b = 5.0
divide()
print 'a/b =',c
>>>
Traceback (most recent call last):
     File ''C:\Python22\scope.py'', line 8, in ?
          print c
NameError: name 'c' is not defined
```

### 11. Γράφοντας και Διορθώνοντας Προγράμματα

Όταν ανοίγεται ο συντάκτης *Idle* του Python, ο χρήστης βλέπει το >>> που δείχνει ότι ο συντάκτης βρίσκεται σε διαδραστική κατάσταση. Οποιαδήποτε εντολή τυπωθεί μέσα στον συντάκτη εκτελείται αμέσως με το πάτημα του πλήκτρου εισόδου (enter). Το διαδραστικό περιβάλλον είναι ένας καλός τρόπος για να μάθει κανείς την γλώσσα με πειραματισμό και για να δοκιμάσει νέες ιδέες προγραμματισμού.

Ανοίγοντας ένα νέο παράθυρο βάζει το Idle σε κατά δέσμη κατάσταση (batch mode), που επιτρέπει το τύπωμα και σώσιμο προγραμμάτων. Επίσης, μπορεί κανείς να χρησιμοποιήσει ένα επεξεργαστή κειμένου (text editor) για να εισάγει γραμμές προγράμματος, όμως το Idle έχει χαρακτηριστικά που είναι συγκεκριμένα για το Python, όπως την κωδικοποίηση λέξεων κλειδιών (keywords) με χρώματα και την αυτόματη εσοχή κειμένου (indentation) που διευκολύνουν τη δουλειά. Πριν τρέξει ένα πρόγραμμα πρέπει να σωθεί σαν αρχείο Python με την επέκταση `.py`, για παράδειγμα `myprog.py`. Το πρόγραμμα μπορεί τότε να εκτελεστεί εκτελόντας την εντολή `python myprog.py`. Στα Windows, ένα διπλό κλικ στο εικονίδιο του προγράμματος δουλεύει επίσης. Όμως προσέξτε: το παράθυρο του προγράμματος κλείνει αμέσως μετά την εκτέλεση πριν προλάβετε να διαβάσετε το αποτέλεσμα. Για να το αποφύγετε αυτό τελειώστε το πρόγραμμα με τη γραμμή

```
raw_input('press return')
```

Κάνοντας διπλό κλικ στο εικονίδιο του προγράμματος δουλεύει επίσης στο Unix και το Linux αν η πρώτη γραμμή του προγράμματος υποδεικνύει τη διαδρομή (path) προς τον διερμηνευτή (interpreter) του Python (ή σε μια δέσμη ενεργειών κελύφους (shell script) που δίνει μια σύνδεση στο Python). Η διαδρομή πρέπει να προηγείται από τα σύμβολα `#!`. Στον υπολογιστή μου η διαδρομή είναι `/usr/bin/python`.

```
#!/usr/bin/python
```

Σε συστήματα πολλαπλών χρηστών η διαδρομή συνήθως είναι `/usr/local/bin/python`.

Όταν μια ενότητα φορτωθεί σε ένα πρόγραμμα για πρώτη φορά με μια εντολή `import`, μεταφράζεται (compiled) σε κώδυκα byte (bytecode) και γράφεται σε ένα αρχείο με επέκταση `.pyc`. Την επόμενη φορά που το πρόγραμμα τρέχει, ο διερμηνευτής φορτώνει το bytecode αντί για το αρχικό αρχείο Python. Αν έχουν γίνει ενδιάμεσες αλλαγές στην ενότητα, αυτή μεταφράζεται ξανά αυτόματα. Το πρόγραμμα μπορεί να τρέξει επίσης και μέσα από το Idle με χρήση του μενού edit/run script, όμως στην περίπτωση αυτή δεν γίνονται αυτόματες επανα-μεταφράσεις ενοτήτων, εκτός και αν το υπάρχον αρχείο bytecode σβηστεί και το παράθυρο του προγράμματος κλειστεί και ξανανοιχτεί.

Τα μηνύματα λάθους του Python μπορεί μερικές φορές να προκαλέσουν σύγχυση όπως φαίνεται από το παρακάτω παράδειγμα:

```
from numpy import array
a = array([1.0, 2.0, 3.0]
print a
raw_input('press return')
```

The output is

```
File ''C:\Python22\test_module.py'', line 3
     print a
           ^
SyntaxError: invalid syntax
```

Ποιό θα μπορούσε να είναι το λάθος στη σειρά `print a`; Η απάντηση είναι τίποτα. Το πρόβλημα είναι στην πραγματικότητα στην προηγούμενη σειρά, όπου το κλείσιμο της παρένθεσης λείπει, με αποτέλεσμα η εντολή να μη είναι πλήρης. Σαν αποτέλεσμα, ο διερμηνευτής βλέπει την τρίτη γραμμή σαν συνέχεια της δεύτερης γραμμής έτσι ώστε να προσπαθεί να ερμηνεύσει την εντολή

```
 a = array([1.0, 2.0, 3.0]print a
```

Το μάθημα είναι το εξής: όταν έχουμε ένα `SyntaxError` πρέπει να κοιτάξουμε την προηγούμενη γραμμή ως το πιθανό πρόβλημα. Μπορεί έτσι να αποφύγει κανείς πολύ κούραση και απογοήτευση.

Είναι καλή ιδέα να βάζει κανείς σχόλια στις ενότητες προσθέτοντας ένα σχόλιο (*docstring)* στην αρχή κάθε ενότητας. Το σχόλιο, που περιλαμβάνεται σε κλειστά τριπλά εισαγωγικά, μπορεί να εξηγεί τι κάνει η ενότητα. Εδώ είναι ένα παράδειγμα που σχολιάζει την ενότητα `error` (χρησιμοποιούμε αυτή την ενότητα σε πολλά από τα προγράμματα μας):

```
## module error
''' err(string).
     Prints 'string' and terminates program.
'''
import sys
def err(string):
     print string
```

```
    raw_input('Press return to exit')
    sys.exit()
```

Το σχόλιο μιας ενότητας μπορεί να τυπωθεί με την εντολή

```
print module_name.__doc__
```

Για παράδειγμα, το σχόλιο του `error` δείχνεται με

```
>>> import error
>>> print error.__doc__
err(string).
    Prints 'string' and terminates program.
```

## 12. Βιβλιοθήκες για Επιστημονικούς Υπολογισμούς στην Python

### 12.1 Γραμμική Άλγεβρα βιβλιοθήκη NumP

**Linear algebra (numpy.linalg)**

**Matrix and vector products**

| | |
|---|---|
| **dot(a, b[, out])** | **Dot product of two arrays.** |
| **vdot(a, b)** | **Return the dot product of two vectors.** |
| **inner(a, b)** | **Inner product of two arrays.** |
| **outer(a, b)** | **Compute the outer product of two vectors.** |
| **tensordot(a, b[, axes])** | **Compute tensor dot product along specified axes for arrays >= 1-D.** |
| **einsum(subscripts, *operands[, out, dtype, ...])** | **Evaluates the Einstein summation convention on the operands.** |
| **linalg.matrix_power(M, n)** | **Raise a square matrix to the (integer) power *n*.** |
| **kron(a, b)** | **Kronecker product of two arrays.** |

**Decompositions**

| | |
|---|---|
| **linalg.cholesky(a)** | **Cholesky decomposition.** |
| **linalg.qr(a[, mode])** | **Compute the qr factorization of a matrix.** |
| **linalg.svd(a[, full_matrices, compute_uv])** | **Singular Value Decomposition.** |

**Matrix eigenvalues**

| | |
|---|---|
| **linalg.eig(a)** | **Compute the eigenvalues and right eigenvectors of a square array.** |
| **linalg.eigh(a[, UPLO])** | **Return the eigenvalues and eigenvectors of a Hermitian or symmetric matrix.** |
| **linalg.eigvals(a)** | **Compute the eigenvalues of a general matrix.** |
| **linalg.eigvalsh(a[, UPLO])** | **Compute the eigenvalues of a Hermitian or real symmetric matrix.** |

**Norms and other numbers**

| | |
|---|---|
| **linalg.norm(x[, ord])** | **Matrix or vector norm.** |
| **linalg.cond(x[, p])** | **Compute the condition number of a matrix.** |
| **linalg.det(a)** | **Compute the determinant of an array.** |
| **linalg.slogdet(a)** | **Compute the sign and (natural) logarithm of the determinant of an array.** |
| **trace(a[, offset, axis1, axis2, dtype, out])** | **Return the sum along diagonals of the array.** |

**Solving equations and inverting matrices**

| | |
|---|---|
| **linalg.solve(a, b)** | **Solve a linear matrix equation, or system of linear scalar equations.** |
| **linalg.tensorsolve(a, b[, axes])** | **Solve the tensor equation a x = b for x.** |
| **linalg.lstsq(a, b[, rcond])** | **Return the least-squares solution to a linear matrix equation.** |

| | |
|---|---|
| **linalg.inv(a)** | **Compute the (multiplicative) inverse of a matrix.** |
| **linalg.pinv(a[, rcond])** | **Compute the (Moore-Penrose) pseudo-inverse of a matrix.** |
| **linalg.tensorinv(a[, ind])** | **Compute the 'inverse' of an N-dimensional array.** |

## 12.2 Περιεχόμενο βιβλιοθήκης SciPy για γραμμική άλγεβρα

Linear Algebra (scipy.linalg)
Matrix Class
Basic routines
Finding Inverse
Solving linear system
Finding Determinant
Computing norms
Solving linear least-squares problems and pseudo-inverses
Generalized inverse
Decompositions
Eigenvalues and eigenvectors
Singular value decomposition
LU decomposition
Cholesky decomposition
QR decomposition
Schur decomposition
Matrix Functions
Exponential and logarithm functions
Trigonometric functions
Hyperbolic trigonometric functions
Arbitrary function
Special matrices

## 13. Γλώσσα Python σε συντομία

```
Compact summary of python
This is not intended to be the complete language.

name your file          xxx.py

execute your file       python xxx.py

on MS Windows only      pythonw xxx.py
when using Tk, wx, Qt

execute your file and save output in a file
                        python xxx.py > xxx.out

# rest of line is a comment, # sign makes the rest of a line a comment
```

```
"""                     starts a long comment, can be many lines
"""                     # ends the long comment

Indentation must remain the same on a sequence of statements.
(statements starting next indentation, end with a colon : )

Indent after each compound statement such as if, while, for, def ...

print "hello from python" # quotes make strings, writes to display
print 'hello from python' # apostrophes make strings, no character type
print "hello",
print "from python"       # because of previous comma, writes same one
line
                          # see files  hello.py  and  hello.out

print x                   # prints the value of x, any type
print "x=",
print x                   # prints one line, e.g. x= 7.25e-27

Strings may use either " or '  (same on both ends)

Open a file for reading or writing, f is any variable name
f=open("myfile.dat", "r")  # open for reading
f=open("myfile.dat", "w")  # open for writing
f.close()                  # close the file
a=f.read()                 # read entire file into string a
a=readline()               # read one line, including \ n, into string
a
f.seek(5)                  # seek five
                           # see files  testio.py  and  testio.out

Reserved words:

and    as      assert   break   class    continue   def    del
elif   else    except   exec    finally  for        from   global
if     import  in       is      lambda   not        or     pass
print  raise   return   try     while    with       yield

Operators:

y=7      # the usual assignment statements with normal precedence
z=5.3
x=y+z
x=-y
x+=1     # in place of x++;

the list of operators is:   +   -   *  **   /   //   %   arithmetic
                           <<  >>   &   |   ^    ~       shift logical
                            <   >  <=  >=  !=  and  or   comparison

Delimiters include:  (    )    [    ]    {    }    @     grouping
                     ,    :    .    `    ;              separating
                     =   +=   -=   *=   /=  //=   %=     operate store
                    &=   |=   ^=  >>=  <<=  **=          operate store

Special characters:  '    "    #    \
Not used:            $    ?
```

```
Numbers:

1 1234567890123456789  1.3  7.5e-200  0xFF  5.9+7.2j

Boolean:  True   False   constants

Tuples:
x=2, 3, 5, 7, 11
y=(2, 3, 5, 7, 11)

Lists:
x=[2, 3, 5, 7, 11]

An array of 10 elements, initialized to zeros:
z=[0 for i in range(10)]
z[0]=z[9]  # lowest and highest subscripts

A 2D array of 3 by 4 elements, initialized to zeros:
a=[[0 for j in range(4)] for i in range(3)]
a[0][0]=a[2][3] # lowest and highest subscripts a[i][j]



import math      # see files  testmath.py  and  testmath.out
a=math.sin(b)
print "b=%f, sin(b)=%f" % (b, a) # formatted output %d, %10.5f, %g ...

import decimal   # big decimal
a=decimal.Decimal("39.25e-200")

import random
a=random.randint(1,10)

import NumPy     # latest numerics package
import Numarray  # older numerics package   Multiarray ?
import Numeric   # very old numerics package


Three equivalent ways to call QString:
from PyQt4.QtCore import * # all directly visible
x=QString()

import PyQt4.QtCore as P   # any short name
x=P.QString()

import PyQt4
x=PyQt4.QtCore.QString()   # fully named

You can have your own program in many files.
e.g. file spiral_trough.py  has a  function  spiral   def spiral(x,y)
     in test_spiral.py    from spiral_trough import spiral
                          z=spiral(x1,y1)



                # control statements  zero or null is True
```

```
                #                       anything else is False
if a<b:         # <=  for less than or equal
  statements
elif c>d:       # >=  for greater than or equal
  statements
elif e==f:      # !=  for not equal
  statements
else:
  statements
                # unindent to terminate

for i in range(n):  # i=0, 1, 2, ... , n-1

for i in range(3,6): # i=3, 4, 5    not 6

for i in range(5,0,-1): # i=5, 4, 3, 2, 1  not 0

while boolean_expression: # || is "or"  && is "and"  True, False

"break" exits loop or "if" statement.
"continue" goes to next iteration in a loop.

Function definition and call:

def aFunction(x, y, z): # inputs of any type
  # do something, compute a and b
  return a,b

# somewhere later
   m, n = aFunction(p, q, r) # none, one, or multiple value return

one or several functions defined in a file
# funct_to_import.py   has two functions defined, funct and rabc

def funct(n):
  return n

a,b,c=0,0,0
def rabc():
  a=1
  b="rabc string"
  c=[3, 4, 5, 6]
  return a,b,c

In another file, call funct and rabc
# test_import_funct.py   import  funct_to_import and run funct and rabc
import funct_to_import

print  funct_to_import.funct(3)

print  funct_to_import.rabc()
x,y,z = funct_to_import.rabc()
print x
print y
print "z[1]=",
print z[1]
```

Η βιβλιοθήκη SciPy και NumPy http://docs.scipy.org/doc/scipy/reference/index.html

Η βιβλιοθήκη SciPy (προφέρεται “Sigh Pie”) είναι ανοιχτό λογισμικό για μαθηματικούς, επιστήμονες, και μηχανικούς. Παρακάτω περιγράφεται το περιεχόμενο της βιβλιοθήκης.

SciPy Tutorial
Introduction
Basic functions in Numpy (and top-level scipy)
Special functions (**scipy.special**)
Integration (**scipy.integrate**)
Optimization (**scipy.optimize**)
Interpolation (**scipy.interpolate**)
Fourier Transforms (**scipy.fftpack**)
Signal Processing (**scipy.signal**)
Linear Algebra (**scipy.linalg**)
Sparse Eigenvalue Problems with ARPACK
Compressed Sparse Graph Routines **scipy.sparse.csgraph**
Statistics (**scipy.stats**)
Multi-dimensional image processing (**scipy.ndimage**)
File IO (**scipy.io**)
Weave (**scipy.weave**)
API - importing from Scipy
Release Notes
**Αναφορές**
Clustering package (**scipy.cluster**)
Constants (**scipy.constants**)
Discrete Fourier transforms (**scipy.fftpack**)
Integration and ODEs (**scipy.integrate**)
Interpolation (**scipy.interpolate**)
Input and output (**scipy.io**)
Linear algebra (**scipy.linalg**)
Miscellaneous routines (**scipy.misc**)
Multi-dimensional image processing (**scipy.ndimage**)
Orthogonal distance regression (**scipy.odr**)
Optimization and root finding (**scipy.optimize**)
Signal processing (**scipy.signal**)
Sparse matrices (**scipy.sparse**)
Sparse linear algebra (**scipy.sparse.linalg**)
Compressed Sparse Graph Routines (**scipy.sparse.csgraph**)
Spatial algorithms and data structures (**scipy.spatial**)
Special functions (**scipy.special**)
Statistical functions (**scipy.stats**)
Statistical functions for masked arrays (**scipy.stats.mstats**)
C/C++ integration (**scipy.weave**)
NumPy βιβλιοθήκη
http://docs.scipy.org/doc/numpy/reference/index.html

Array objects

# Παράρτημα 3: Γραμμική Άλγεβρα: Έννοιες και Αριθμητικές Μέθοδοι

Το παράρτημα αυτό είναι προσαρμογή στα Ελληνικά μέρος της ύλης του μαθήματος «αριθμητικοί μέθοδοι) που βρίσκεται στην ιστοσελίδα http://www.inf.unikonstanz.de/cgip/lehre/na_08/index.shtml.en.
Το λογισμικό που χρησιμοποιείται στο παράρτημα αυτό μαζί με την βιβλιοθήκη drawToolbox αναφέρεται στο τέλος του παραρτήματος όπου δίδεται και η διεύθυνση για λήψη (download) του.

## 1. Περιγραφή του drawToolbox με παραδείγματα

Εισαγωγή στο drawToolbox
Το drawToolbox είναι μια συλλογή από συναρτήσεις του MATLAB που διευκολύνουν την αναπαράσταση απλών 2-διάστατων/ 3-διάστατων γεωμετρικών αντικειμένων όπως διανύσματα, επίπεδα, και γραμμές. Το εργαλείο γράφτηκε για να οπτικοποιεί γεωμετρικές ιδέες σε σχέση με τις μεθόδους της αριθμητικής γραμμικής άλγεβρας.

Υπάρχουν 6 συναρτήσεις στο drawToolbox:

- **drawVector** – Αναπαριστά 2-διάστατα/ 3-διάστατα διανύσματα σε άξονες.
- **drawPlane** – Αναπαριστά 2-διάστατα/ 3-διάστατα επίπεδα.
- **drawSpan** – Σχεδιάζει γραμμές (2 διαστάσεων)/ επίπεδα (3 διαστάσεων) με βάση ένα (2-διάστατο/ 3-διάστατο) ή δύο 3-διάστατα διανύσματα V.
- **drawLine** – Αναπαριστά 2-διάστατες/ 3-διάστατες γραμμές.
- **drawMesh** – Αναπαριστά πολυωνυμικά 3-διάστατα πλέγματα.
- **drawAxes** – Σχεδιάζει 2-διάστατες/ 3-διάστατες γραμμές αξόνων.

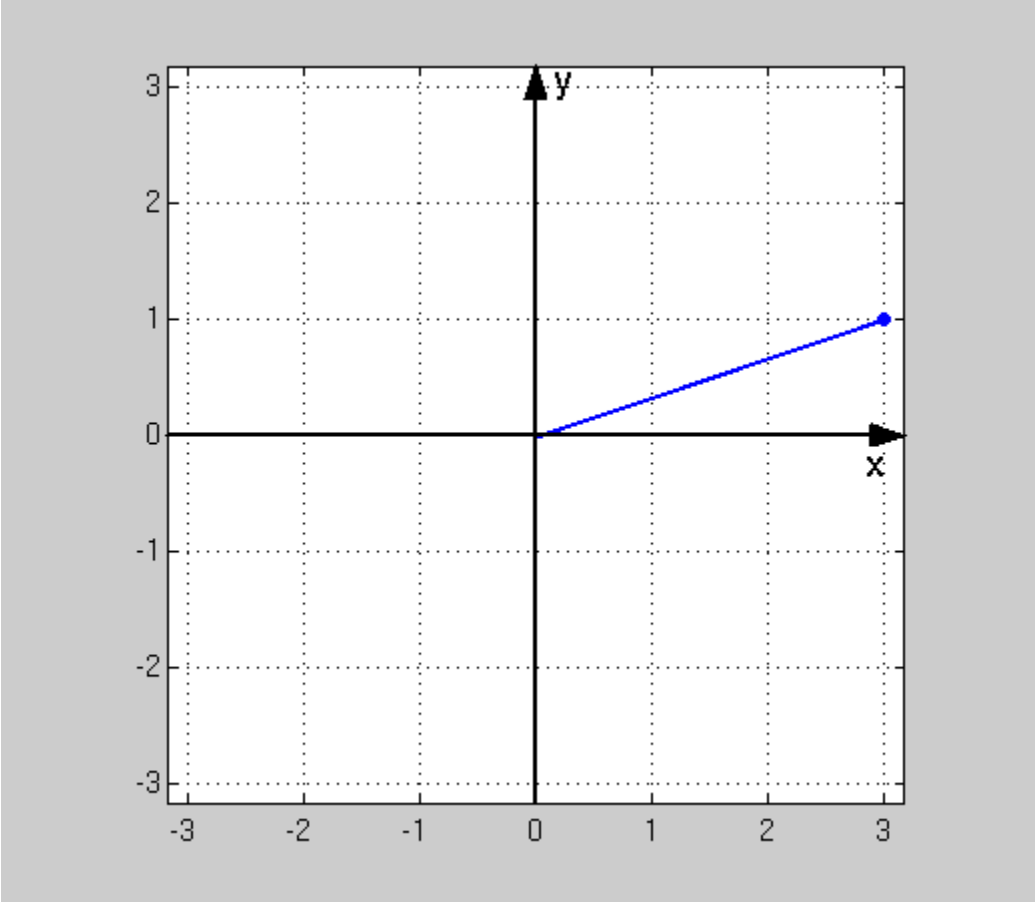
Μπορείτε να βρείτε πληροφορίες για αυτές τις συναρτήσεις με τον συνήθη τρόπο πληκτρολογώντας help function, π.χ., help drawPlane, στο command window της MatLab. Αλλά τώρα, ας δούμε με μια γρήγορη ματιά την χρήση των συναρτήσεων με μερικά απλά παραδείγματα.

### 1.1. drawVector

**Παράδειγμα σε 2 διαστάσεις:** Ας σχεδιάσουμε ένα 2-διάστατο διάνυσμα **a** = [3 1]':

```
a = [3 1]'
figure(1); clf;
drawVector(a)
```

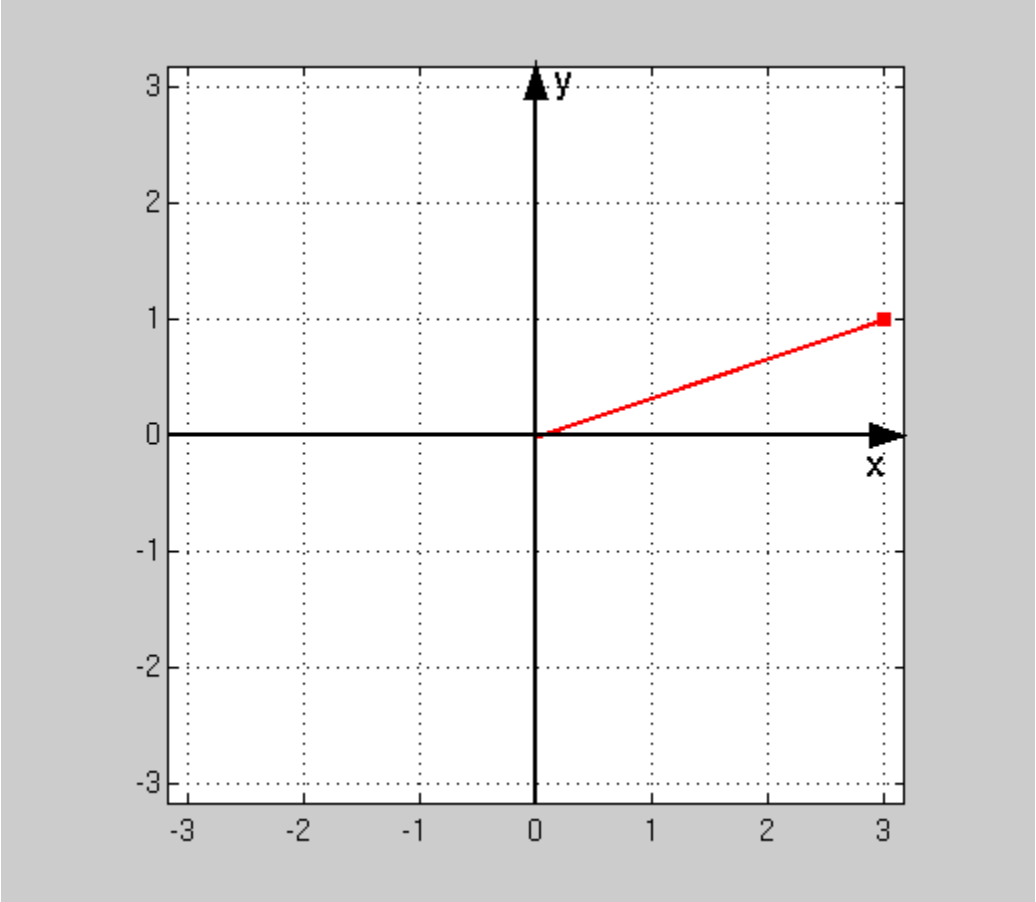
```
a =

   3
   1
```




Παρατηρείστε τους αριθμημένους άξονες. Προστέθηκαν αυτόματα.

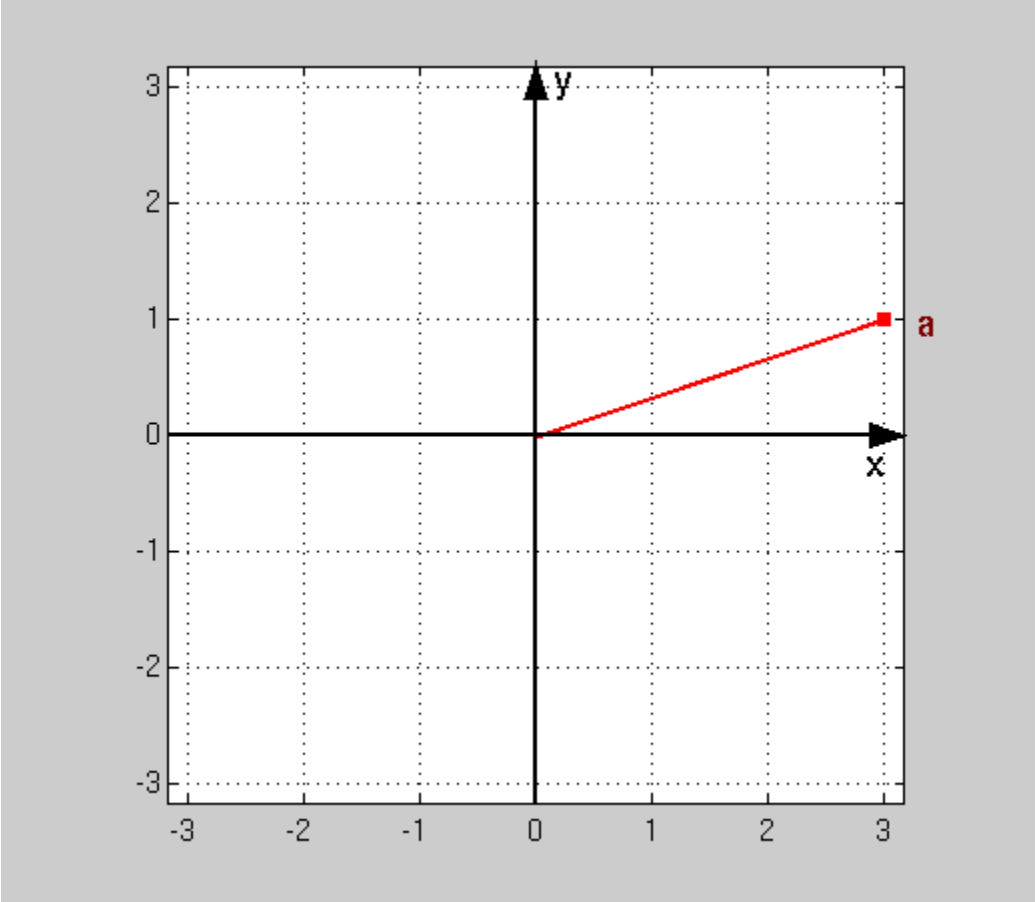
Το προεπιλεγόμενο χρώμα του διανύσματος είναι το μπλε και το προεπιλεγμένο σύμβολο η τελεία. Μπορείτε εύκολα να κάνετε αλλαγές, για παράδειγμα το χρώμα να γίνει κόκκινο και το σύμβολο να γίνει ένα τετράγωνο.

```
drawVector(a, 'rs')
```



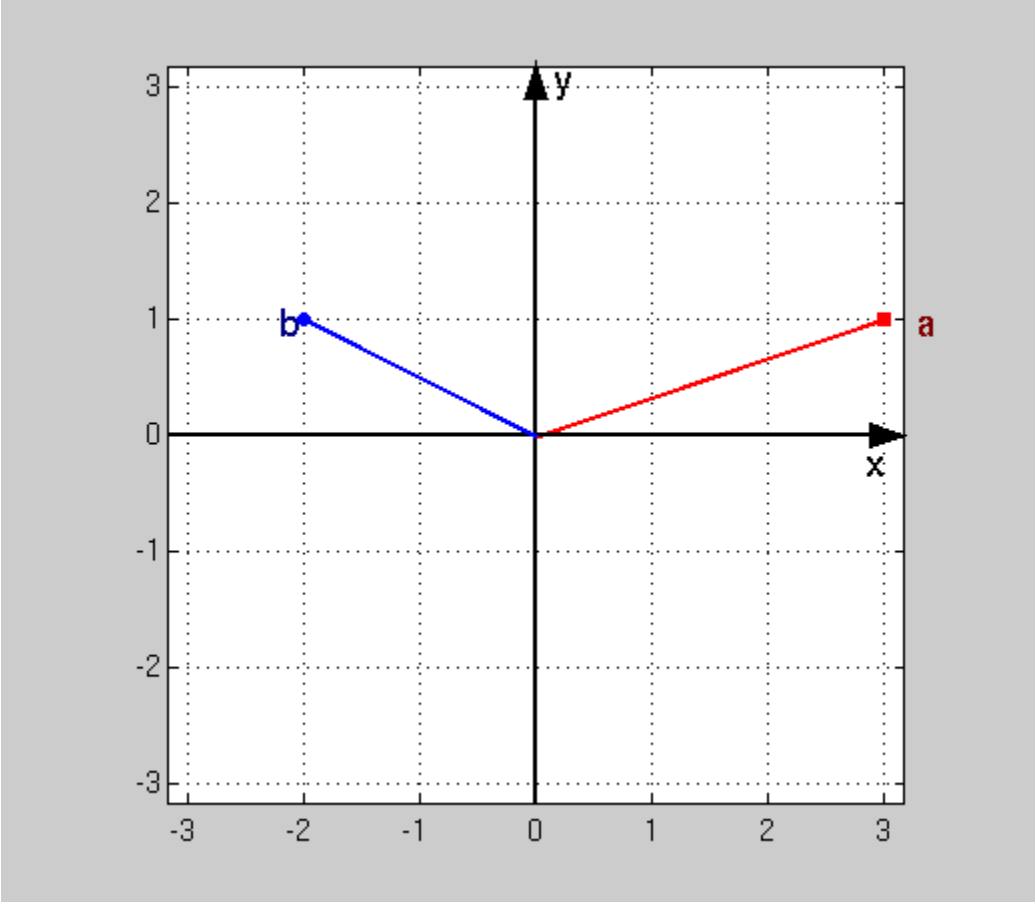

Είναι επίσης πιθανόν να ονομάσετε το διάνυσμα, δίνοντας τον τίτλο μέσα σε αγκύλες:

```
drawVector(a, 'rs', {'a'})
```



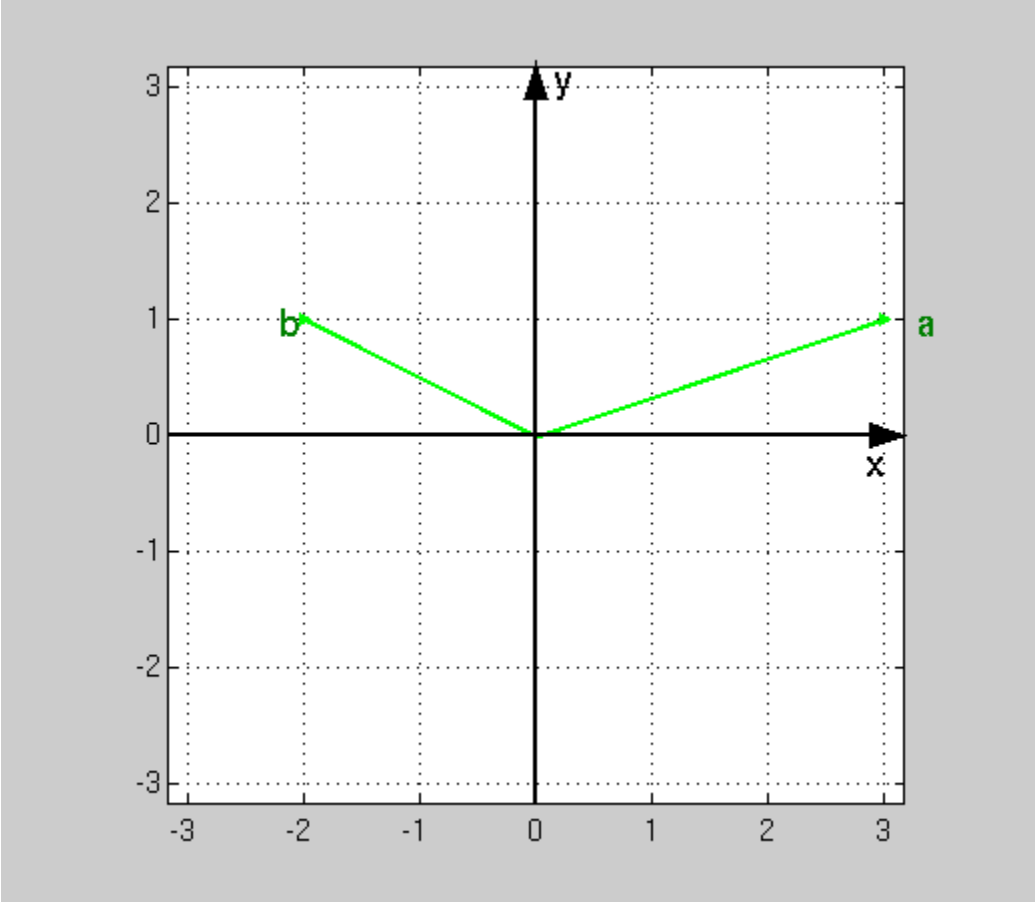

Για να σχεδιάσετε ένα δεύτερο διάνυσμα, π.χ., **b** = [-2 1]' μπορείτε να κάνετε τα παρακάτω:

```
b = [-2 1]';
hold on;
drawVector(b, {'b'})
hold off;
```




Αλλά υπάρχει και καλύτερος τρόπος να το κάνετε αυτό σε μια γραμμή:

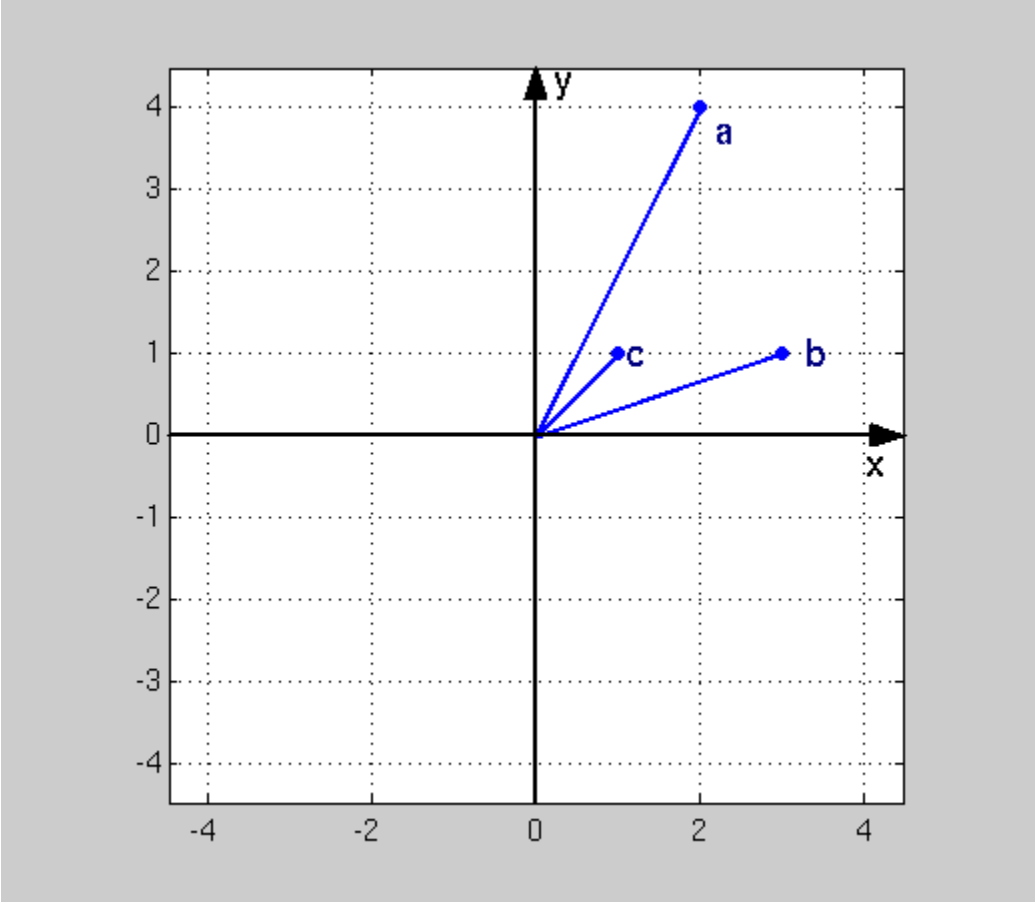
```
figure(1); clf;
drawVector([a b], 'g>', {'a', 'b'});
```




Εντούτοις, τώρα θα πρέπει να χειριστούμε και τα ξεχωριστά χρώματα και σύμβολα των δύο διανυσμάτων.

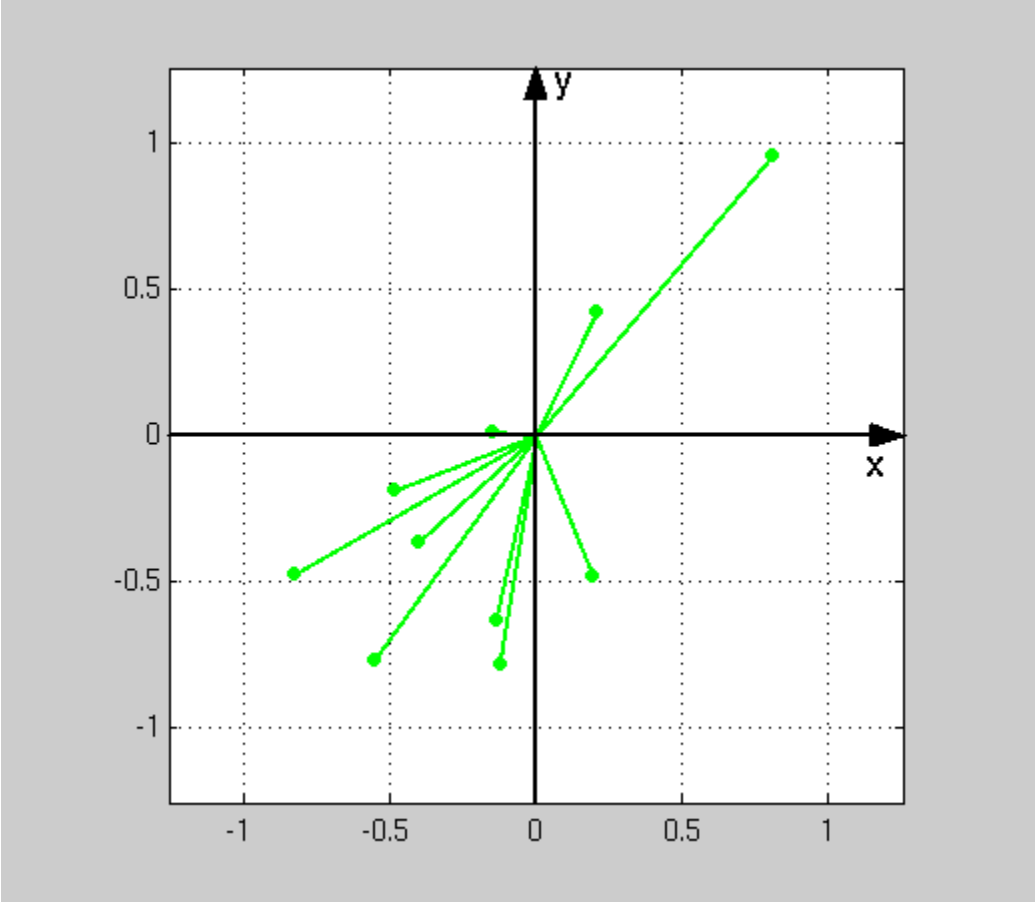
Με παρόμοιο τρόπο μπορείτε να αναπαραστήσετε περισσότερα διανύσματα, βάζοντας τα σαν στήλες ενός πίνακα. Πχ.:

```
A = [ 2 3 1 ;
    4 1 1 ];
drawVector(A, {'a', 'b', 'c'});
```




Ή χρησιμοποιώντας έναν τυχαίο πίνακα (επαναλάβετε το στοιχείο πολλές φορές):

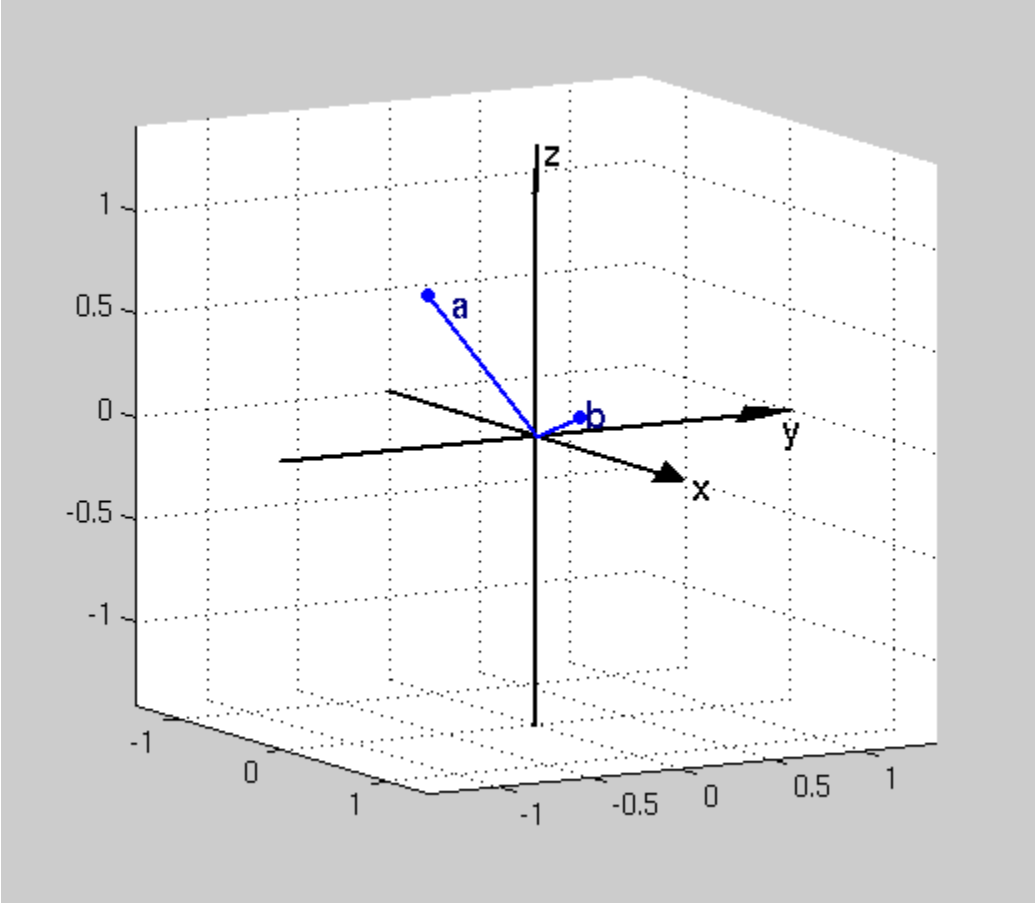
```
A = rand(2, 10)*2 - 1;
drawVector(A, 'go');
```




Είναι ξεκάθαρο γιατί έχουμε πολλαπλασιασμό με 2 και offset -1?

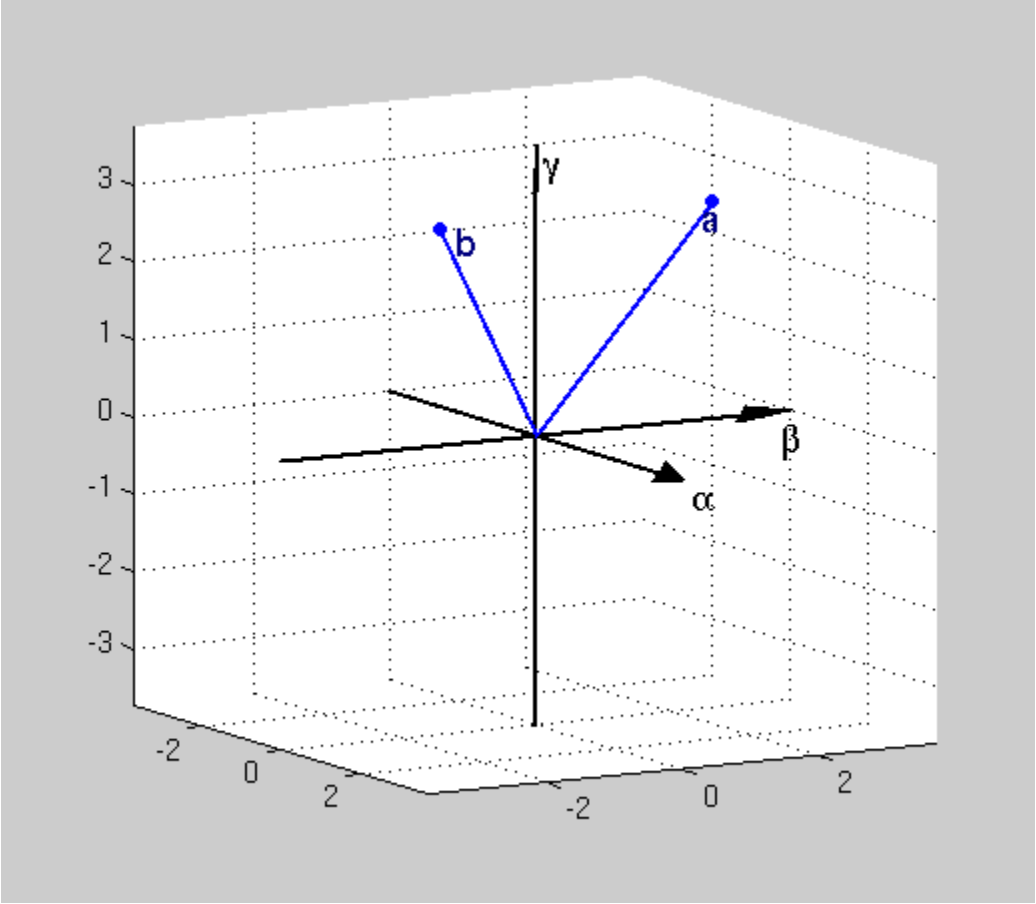
**Παράδειγμα σε 3 διαστάσεις:** Το ίδιο μπορείτε να κάνετε και σε 3 διαστάσεις. Ας σχεδιάσουμε δύο τυχαία 3-διάστατα διανύσματα: A = rand(**3**, 2)*2 - 1;

```
A = rand(3, 2)*2 - 1;
figure(1); clf;
drawVector(A, {'a', 'b'});
view(60,10)
```




Και πάλι οι άξονες προστίθενται αυτόματα. Αν θέλετε να βάλετε ετικέτες στους άξονες χρησιμοποιείστε την προαιρετική παράμετρο, 'AxesLabels':

```
A = [ 1  1;
     2 -2
     3  3 ];
figure(1); clf;
drawVector(A, {'a', 'b'}, 'AxesLabels', {'\alpha','\beta','\gamma'});
view(60,10)
```



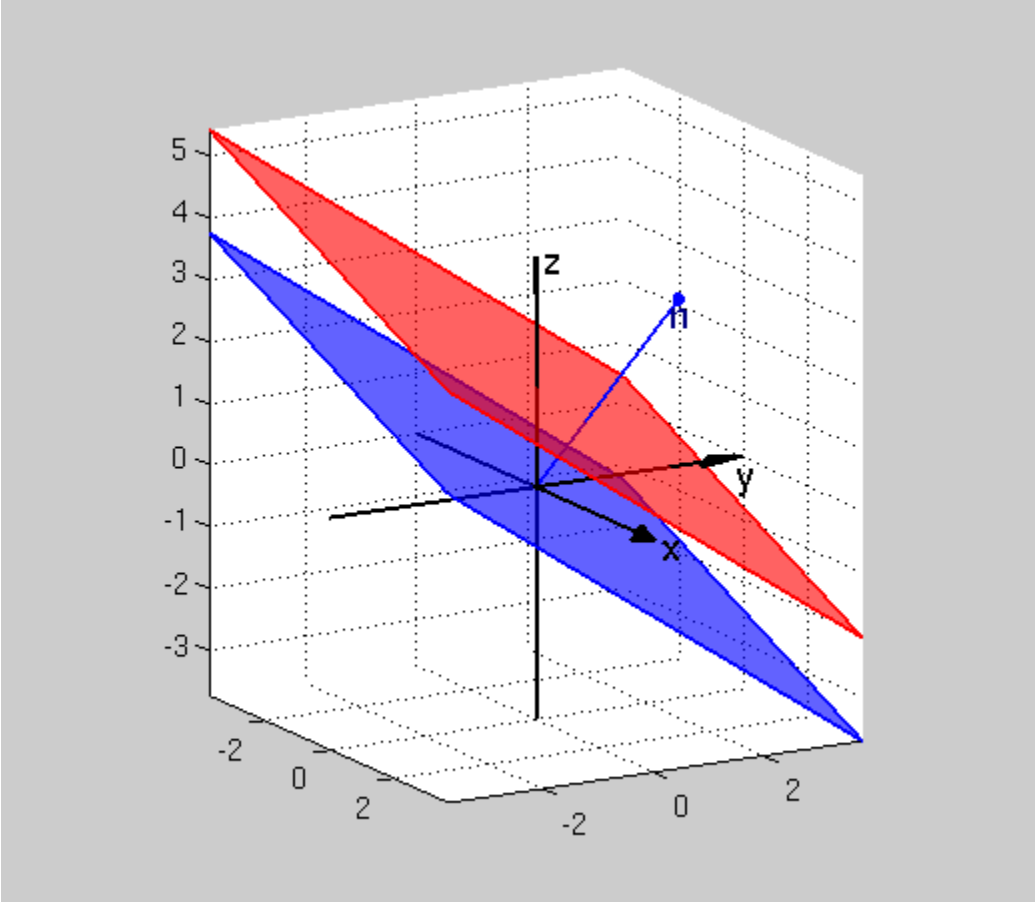

## 1.2. drawPlane

Ένα επίπεδο ορίζεται με ένα διάνυσμα, **n**, το οποίο καθορίζει τον προσανατολισμό του επιπέδου και ένα βαθμοτό, *d*, που ορίζει το μέγεθος του επιπέδου από την αρχή και προς την κατεύθυνση του **n**. Για τα δεδομένα **n** και *d*, όλα τα σημεία (π.χ., διανύσματα **x**), που ικανοποιούν την παρακάτω εξισώση ανήκουν στο επίπεδο:

$$n \bullet x + d = 0$$

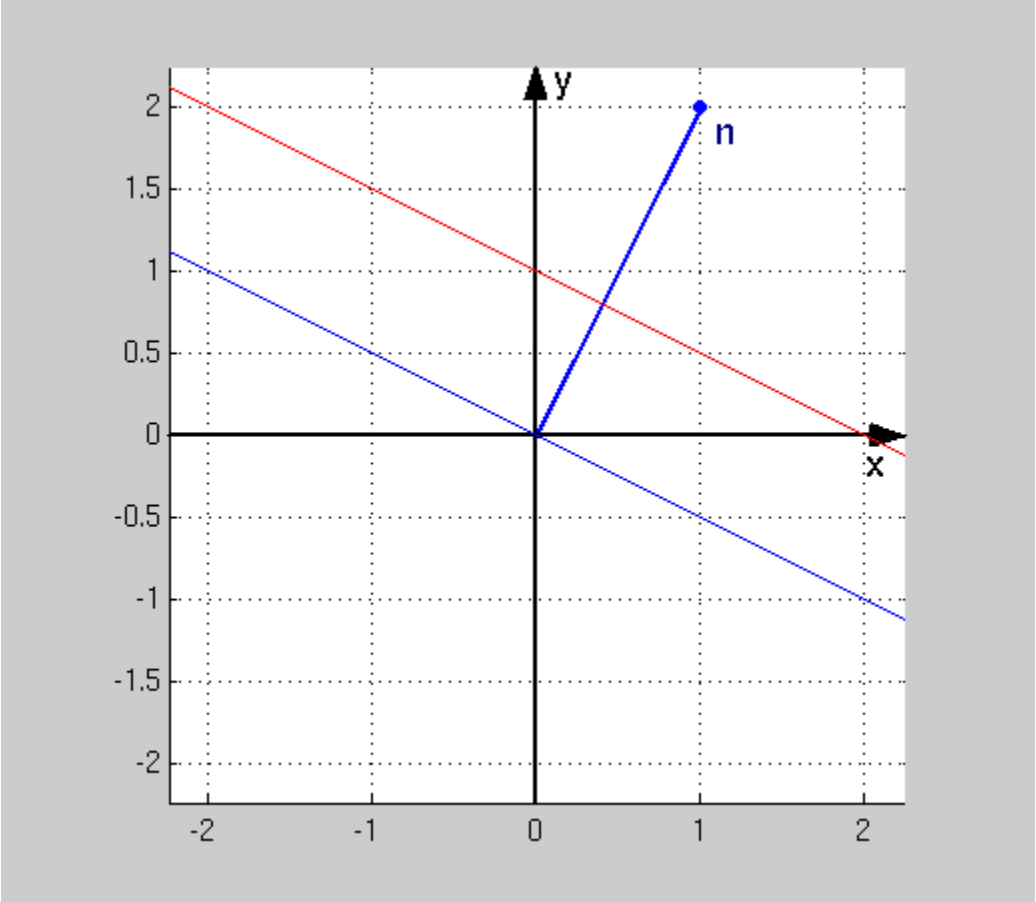
Με την συνάρτηση drawPlane(), μπορείτε εύκολα να οπτικοποιήσετε επίπεδα σε 2 και 3 διαστάσεις. Ας ξεκινήσουμε με 3 διαστάσεις. Έχουμε τον παρακάτω κώδικα:

```
n = [1 2 3]';
figure(1); clf; hold on;
drawVector(n, {'n'});      % the normal
drawPlane(n);              % unshifted plane, comes through the origin
drawPlane(n, 5, 'r');      % plane shifted parallel by "5"
view(60,15)
hold off;
```




Σε 2-διάστατο χώρο, το επίπεδο είναι μια ευθεία γραμμή. Παρακάτω, είναι το ίδιο παράδειγμα όπως πάνω απλά σε 2 διαστάσεις:

```
n = [1 2]';
figure(1); clf; hold on;
drawVector(n, {'n'});      % the normal
drawPlane(n);              % unshifted plane, comes through the origin
drawPlane(n, 2, 'r');      % plane shifted parallel by "2"
hold off;
```

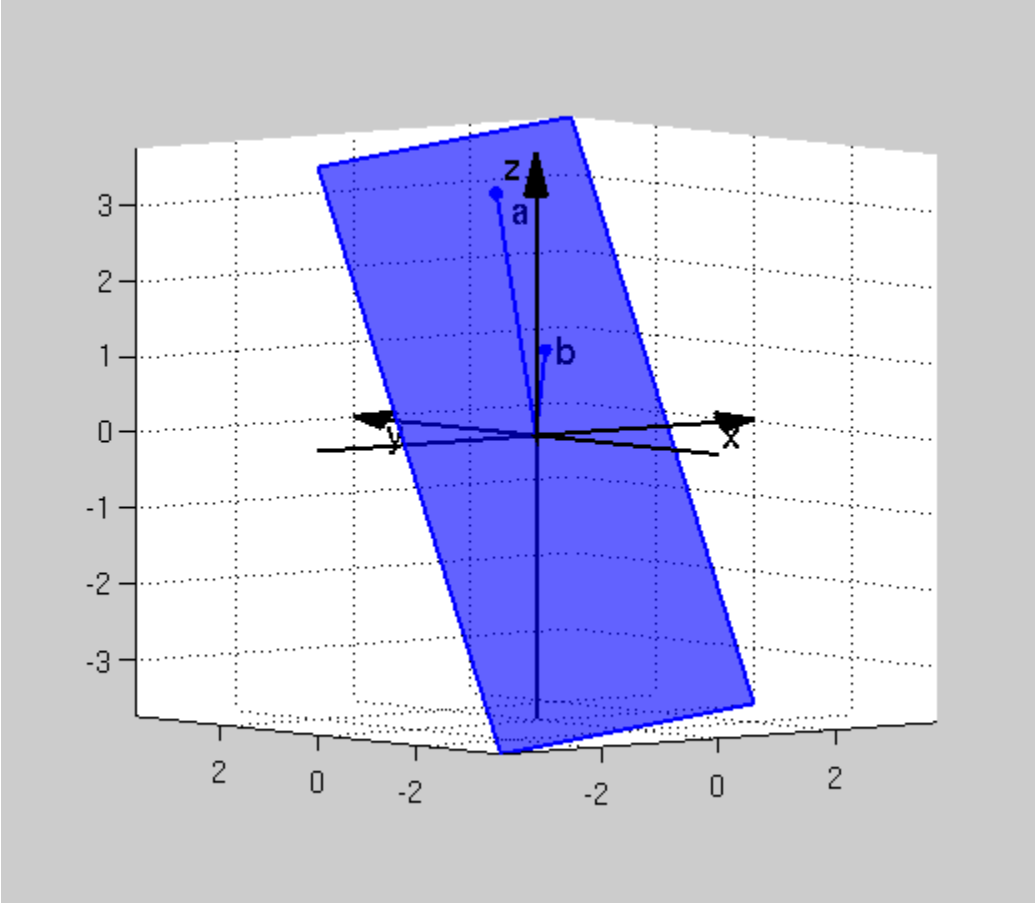



## 1.3. drawSpan

Υπάρχει ακόμη μία δυνατότητα να καθορίσουμε ένα (υπέρ-)επίπεδο σε έναν N-διάστατο χώρο: θεωρείστε N-1 διάστατα διανύσματα τα οποία εκτείνονται στο επίπεδο, δηλαδή, διανύσματα που βρίσκονται στο επίπεδο. Η ρουτίνα drawSpan() οπτικοποιεί επίπεδα σε 2 και 3 διαστάσεις με τον ίδιο τρόπο.

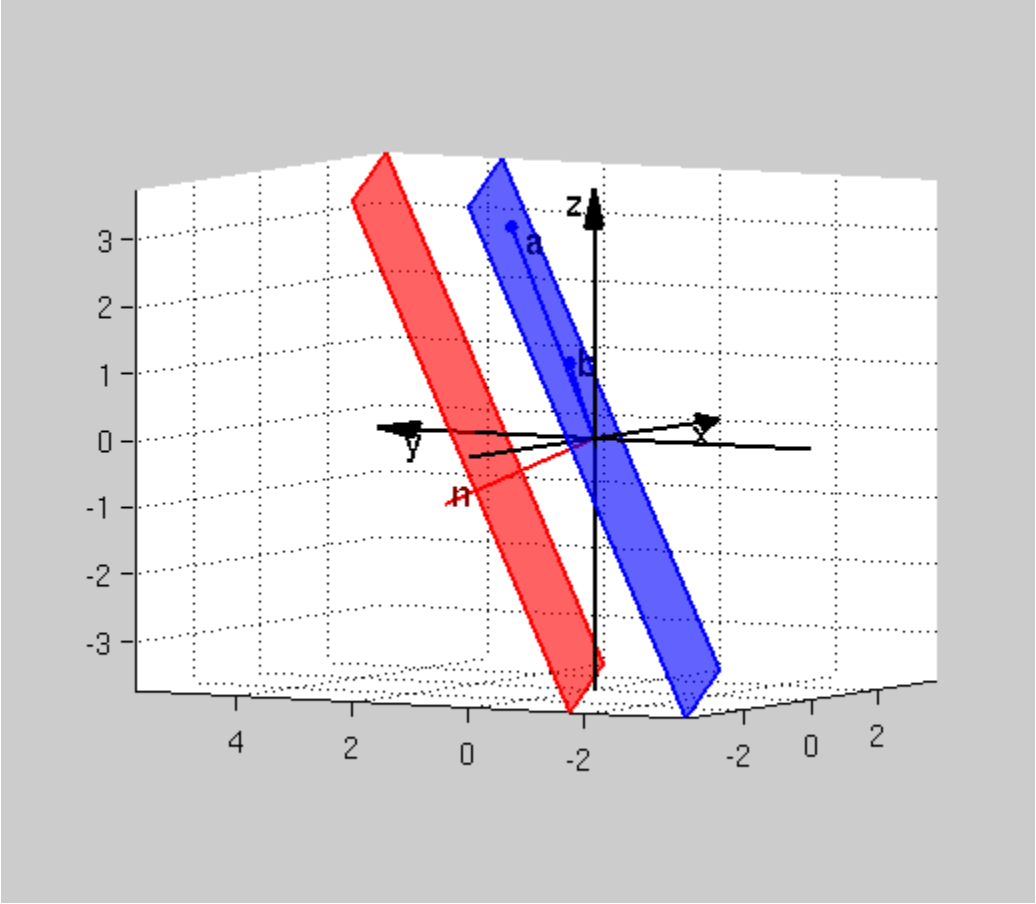
**Παράδειγμα σε 3 διαστάσεις:**

```
a = [1 2 3]'; b = [1 1 1]';        % Two 3D-vector
figure(1);clf; hold on;
drawVector([a b], {'a','b'});
drawSpan([a b], 'b')
view(-40,5)
hold off;
```



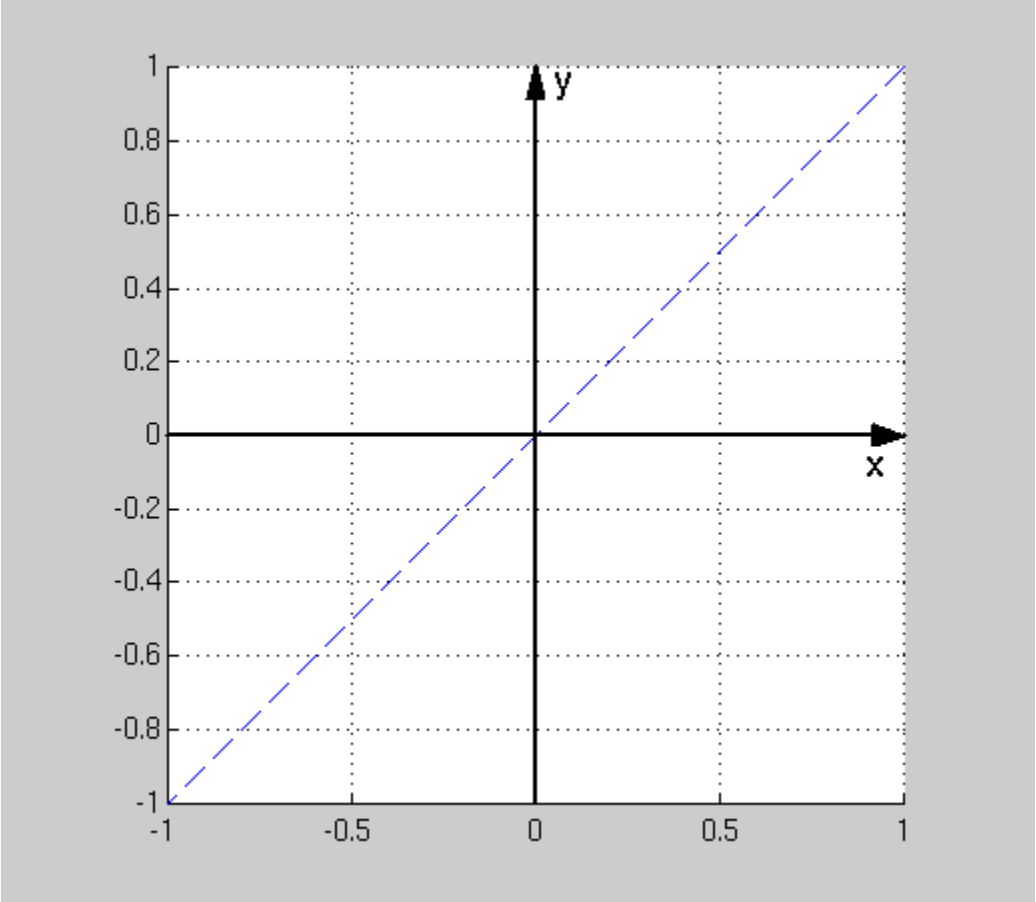

**Σημείωση:** Ένα επίπεδο, που ορίζεται από διανύσματα πάντα περιέχει μία αρχή. Από μαθηματικής πλευράς, είναι ένας γραμμικός υποχώρος. Για να μετατοπίσετε το επίπεδο, πρέπει να υπολογίσετε το αρχικό και να χρησιμοποιήσετε την συνάρτηση drawPlane():

```
n = cross(a,b);                 % The normal
figure(1);hold on;
drawVector(n, {'n'}, 'r');
drawPlane(n, 4, 'r')             % Plot red plane
view(-60,5)
hold off;
```




**Παράδειγμα σε 2 διαστάσεις**: Ένα επίπεδο σε 2 διαστάσεις ορίζεται με ένα 2-διάστατο διάνυσμα:

```
a = [1 1];
figure(1); clf;
drawSpan(a);
```

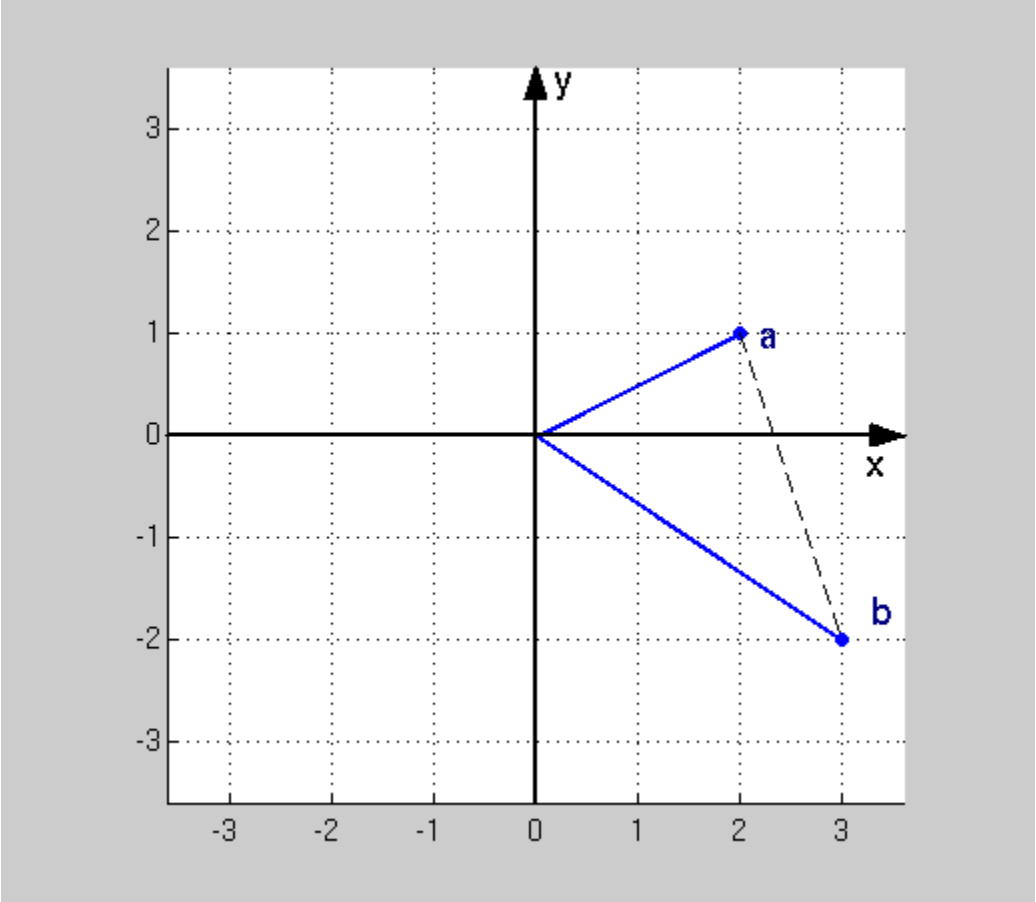



## 1.4. drawLine

Όπως φαίνεται και από το όνομα, αυτή η συνάρτηση σχεδιάζει μια γραμμή (2 ή 3 διαστάσεων) μεταξύ δύο σημείων (δηλαδή, διανυσμάτων). Σαν προεπιλογή, η γραμμή είναι διακεκομμένη χρώματος μαύρου, αλλά αυτά μπορούν εύκολα να αλλάξουν. Εδώ είναι ένα παράδειγμα:

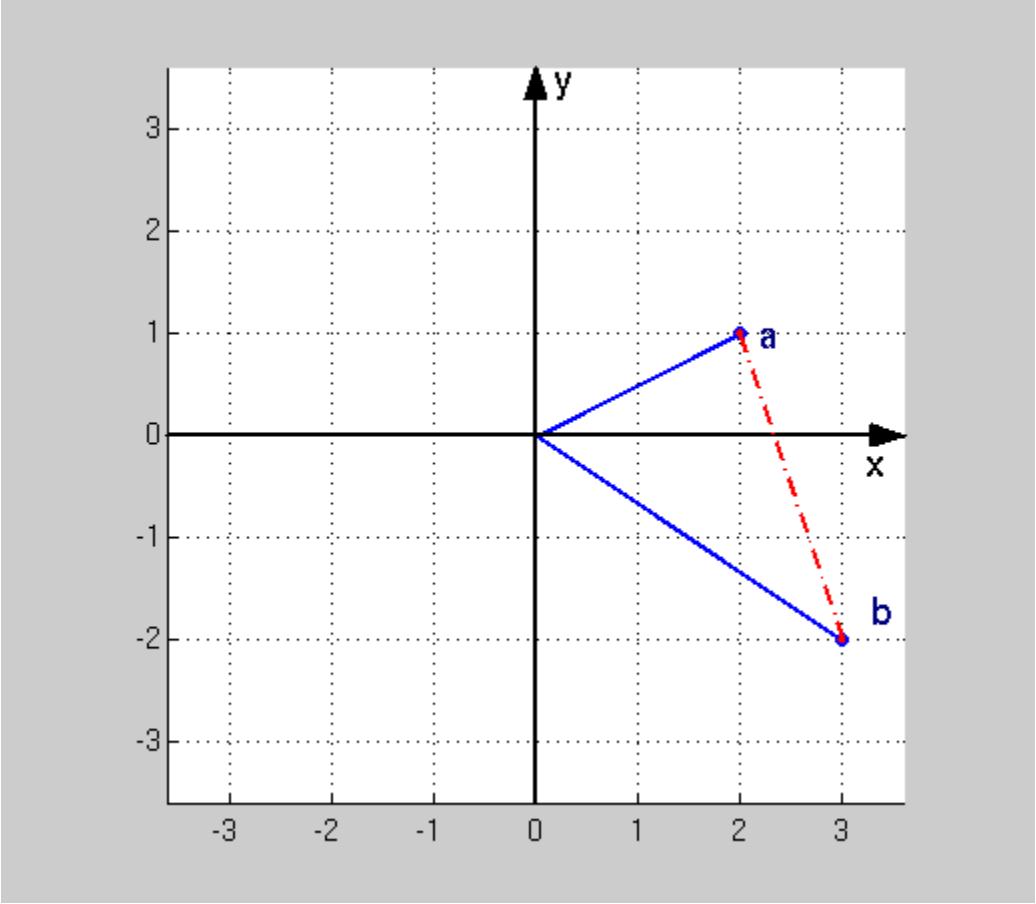
**Παράδειγμα σε 2 διαστάσεις:**

```
a = [2 1]'; b = [3 -2]';          % Two 2D-vectors
figure(1);clf; hold on;
drawVector([a b], {'a','b'});      % Plot the vectors
drawLine([a b]);                   % Draw a line between them
hold off;
```



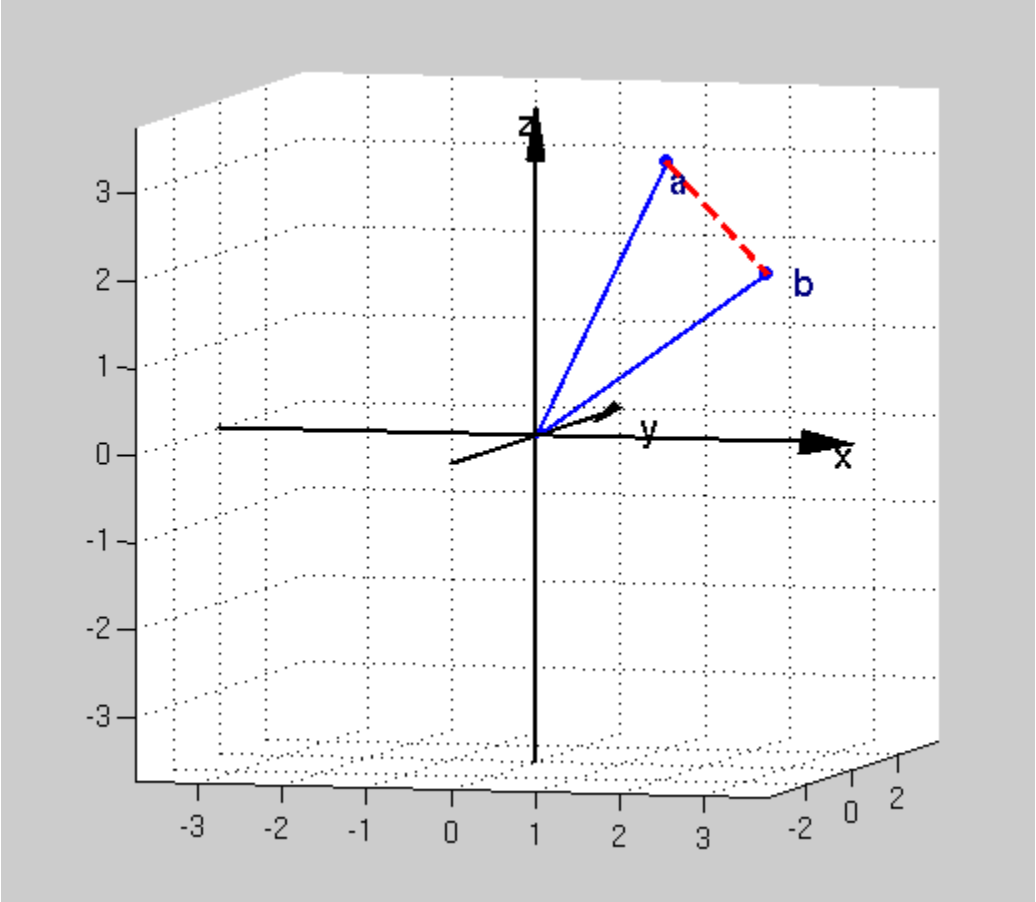

Αλλάξτε τον τύπο της γραμμής (δείτε help drawLine).

```
figure(1);clf; hold on;
drawVector([a b], {'a','b'});      % Plot the vectors
drawLine([a b], 'r2-.');           % Draw a line between them
hold off;
```




**Παράδειγμα σε 3 διαστάσεις:**

```
a = [1 2 3]'; b = [3 -1 2]';        % Two 3D-vectors
figure(1);clf; hold on;
drawVector([a b], {'a','b'});       % Plot the vectors
drawLine([a b], 'r3');              % Draw a line between them
view(15,5);
hold off;
```

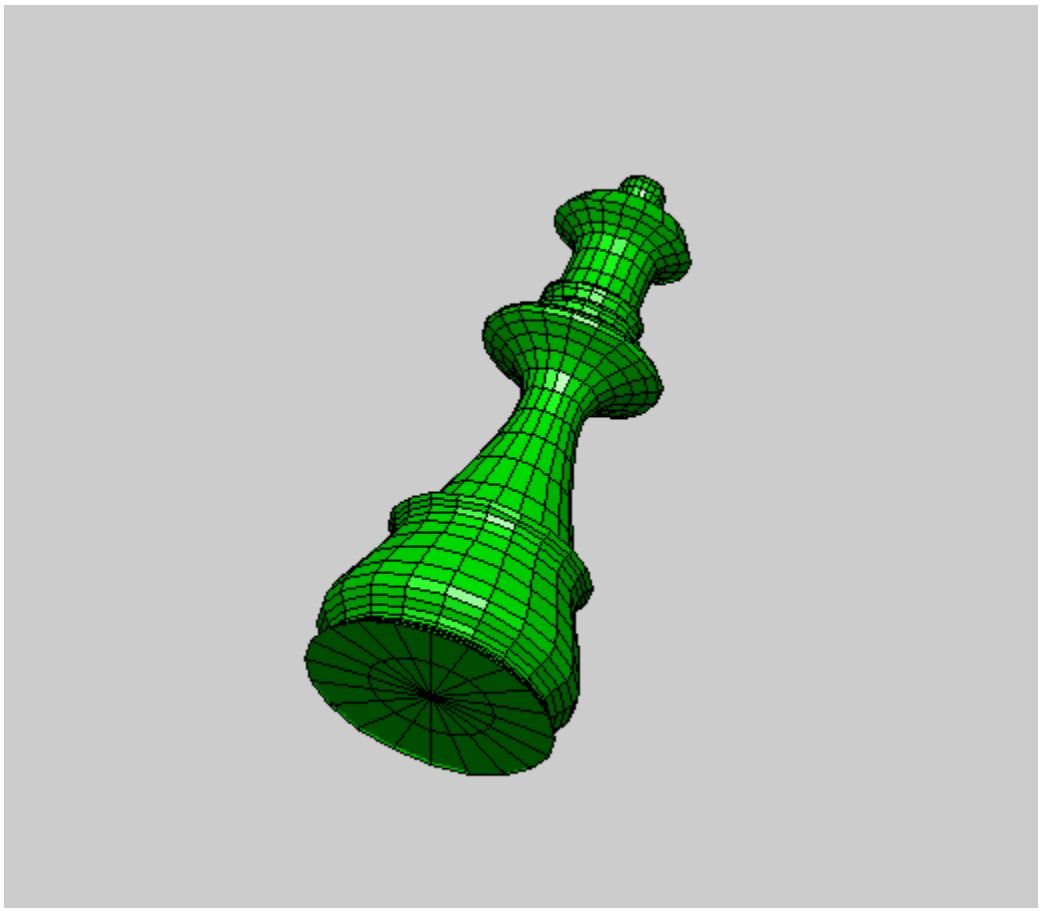



## 1.5. drawMesh

Αυτή η συνάρτηση αναπαριστά ένα 3-διάστατο πολυωνυμικό πλέγμα. Ένα «πλέγμα» ορίζεται με δύο πίνακες: **V** και **F**. Ο πρώτος έχει διαστάσεις *n*-επί-3 και περιέχει τις 3-διάστατες συντεταγμένες των *n* «διαστάσεων». Ο δεύτερος *m*-επί-k πίνακας των «επιφανειών» ορίζει την συνδεσιμότητα των αξόνων: κάθε μία από τις *m* γραμμές αναφέρεται σε κάθε επιφάνεια και περιέχει τους *αριθμούς* (δηλαδή, θετικούς ακεραίους) των αξόνων. Διαφορετικές επιφάνειες μπορεί να έχουν διαφορετικό αριθμό αξόνων, με *k* να είναι ο μέγιστος αριθμός των αξόνων στις επιφάνειες. Το παρακάτω παράδειγμα κάνει ξεκάθαρο αυτό το θέμα:

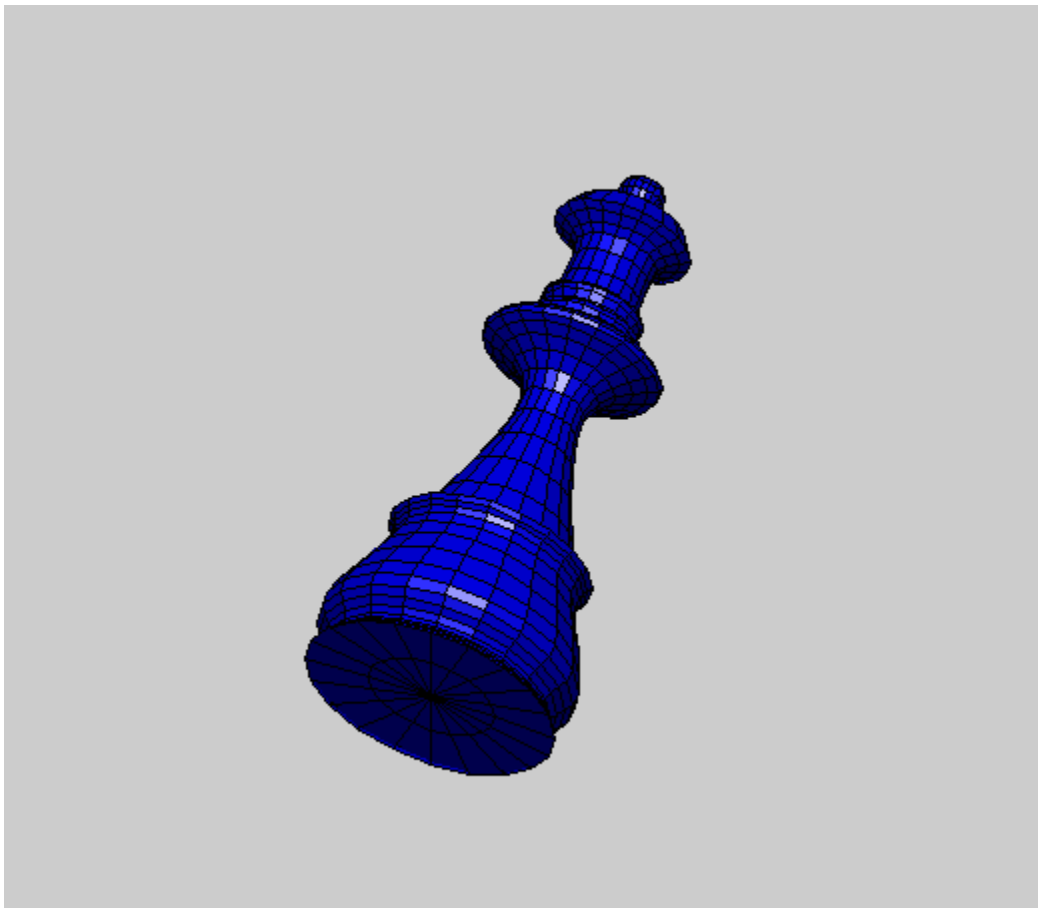
**Παράδειγμα 1: Γραφική παράσταση επιφάνειας**

```
load('queen.mat');              % Load the vertex and face arrays
figure(1); clf;
drawMesh(vertex, face)          % Plot the mesh
view(20, 60)
```



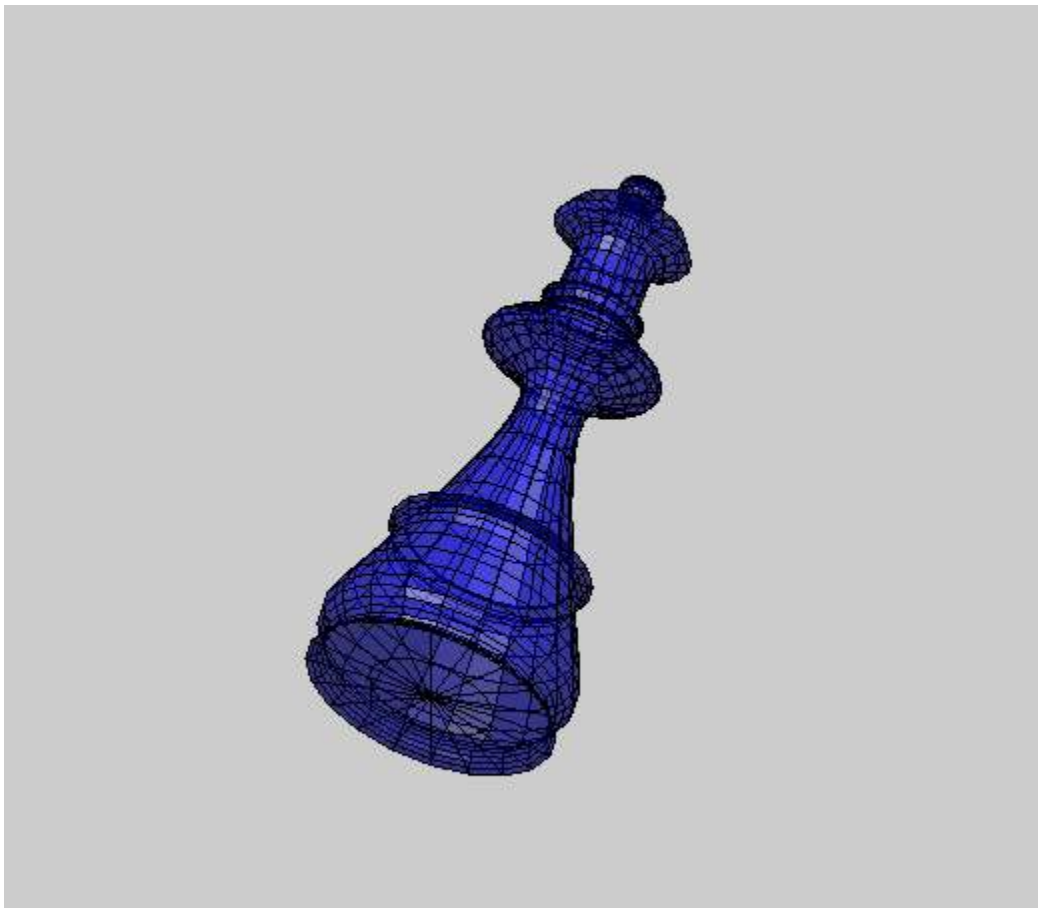
Φυσικά, μπορείτε να αλλάξετε το χρώμα του πλέγματος:

```
figure(1); clf;
drawMesh(vertex, face, 'b')        % Plot the mesh in blue color
view(20, 60)
```



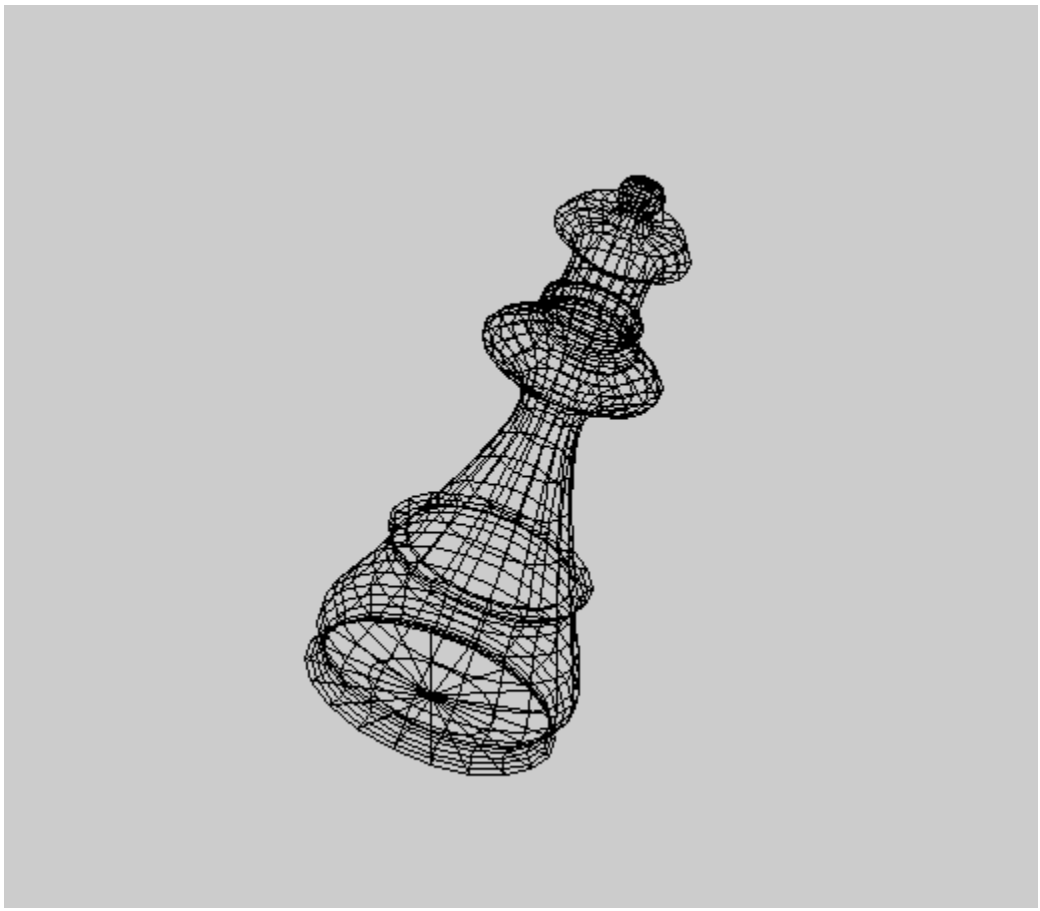
... και να αλλάξετε την διαφάνεια:

```
figure(1); clf;
drawMesh(vertex, face, 'b', .5)     % Semitransparent mesh
view(20, 60)
```



**Παράδειγμα 2: Γραφική παράσταση Wire-frame**

```
figure(1); clf;
drawMesh(vertex, face, 'wire')      % wire-frame plot
view(20, 60)
```

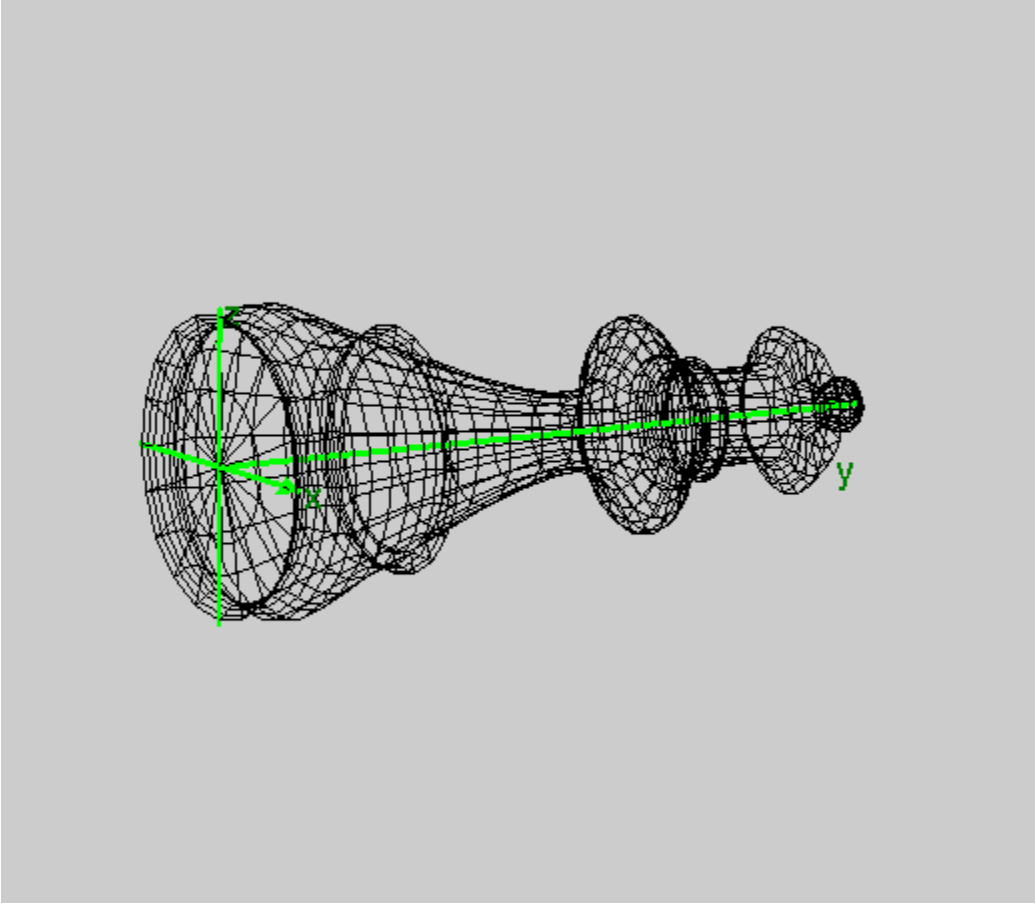


## 1.6. drawAxes

Αυτή η συνάρτηση δεν χρησιμοποιείτε σχεδόν ποτέ απευθείας, αλλά καλείται μαζί με άλλες συναρτήσεις του εργαλείου. Σχεδιάζει τους άξονες "xy" (2 διαστάσεις) ή "xyz" (3 διαστάσεις) στο τρέχων σχήμα. Απαιτεί δύο υποχρεωτικές παραμέτρους: αριθμός διαστάσεων, d = {2,3}, και χρώμα αξόνων.

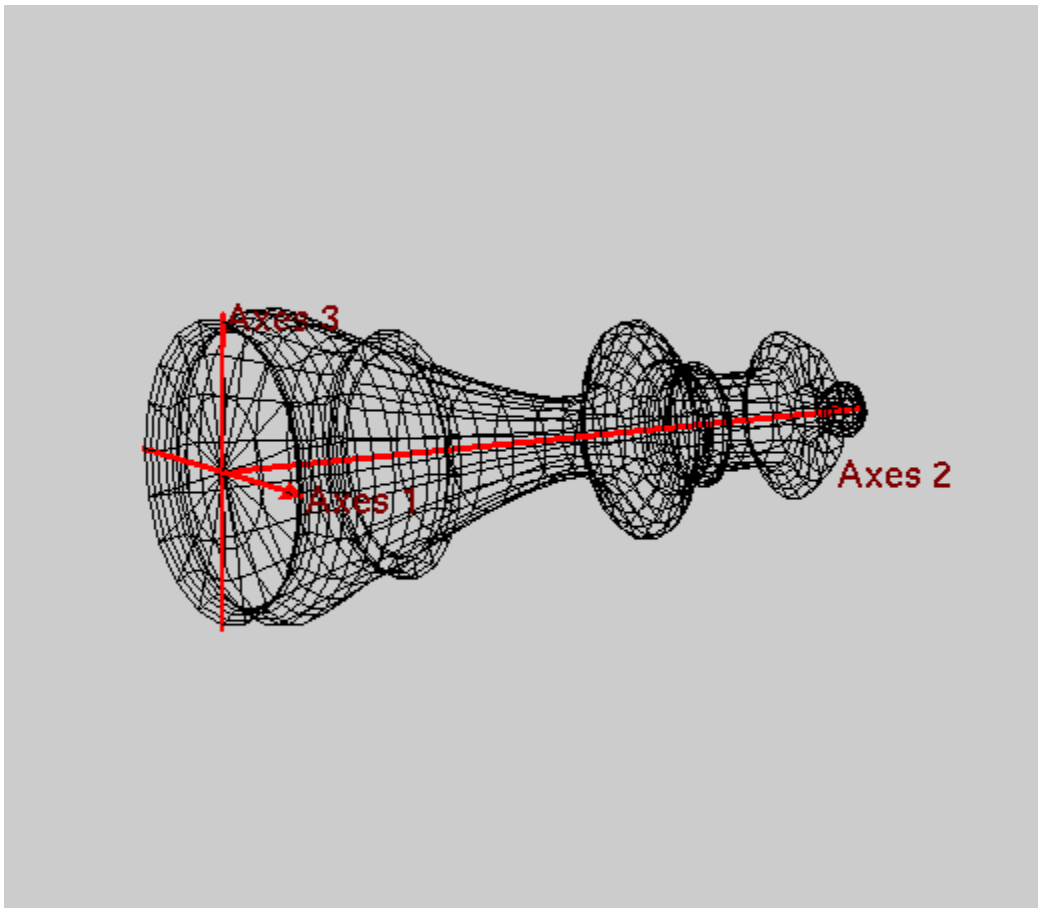
**Παράδειγμα σε 3 διαστάσεις:** Σταθείτε στο προηγούμενο παράδειγμα

```
load('queen.mat');                % load the vertex and face arrays
figure(1); clf;
drawMesh(vertex, face, 'wire');
drawAxes(3, 'g');                 % Draw green axes
view(60, 10)
```



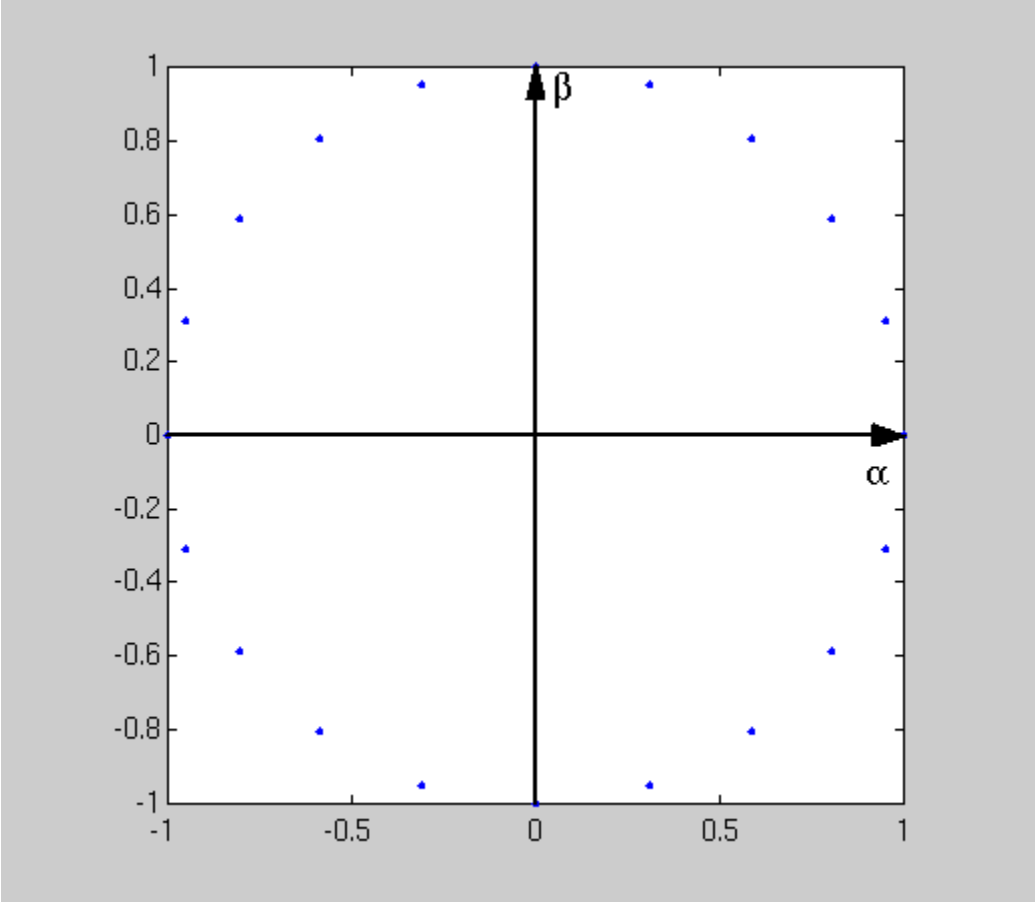

Μερικές φορές μπορείτε να έχετε διαφορετικές ετικέτες αξόνων αντί για τα προεπιλεγμένα "xyz". Μπορεί να γίνει με την drawAxes βάζοντας μία ακόμη (προαιρετική) παράμετρο, έναν πίνακα κελιών:

```
figure(1); clf;
drawMesh(vertex, face, 'wire');
drawAxes(3, 'r', {'Axes 1','Axes 2','Axes 3'}); % Name axes
view(60, 10)
```




**Παράδειγμα σε 2 διαστάσεις:**

```
figure(1); clf;
plot(exp(2*pi*i*(1:20)/20), '.');
drawAxes(2, 'k', {'\alpha','\beta'});
```




Έτσι τελειώνει η εισαγωγή.

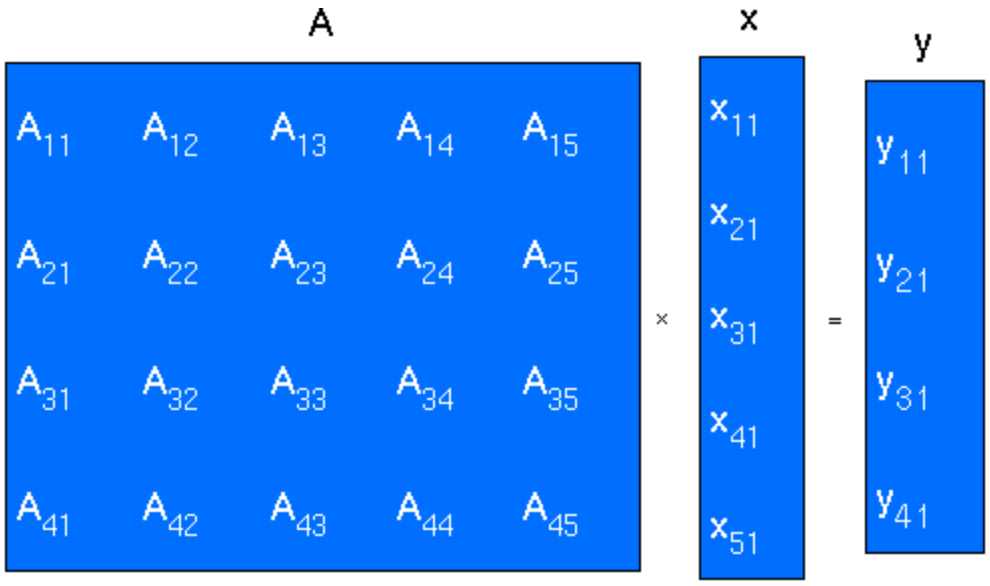
## 2. Γινόμενο πίνακα με διάνυσμα

Ένας πίνακας και ένα διάνυσμα είναι ακολουθείες (arrays) $2^{ων}$ και $1^{ας}$ διάστασης, έτσι ώστε να υπάρχουν δύο τρόποι για τον πολλαπλασιασμό πίνακα-διανύσματος. Κάποιος μπορεί να δει αυτόν το πολλαπλασιασμό σαν μια συλλογή από γινόμενα βαθμοτών επί διανυσμάτων ή σαν ένα γραμμικό συνδυασμό στηλών/ γραμμών του πίνακα. Ανάλογα με τις περιστάσεις, βρίσκουμε τον κατάλληλο τρόπο.

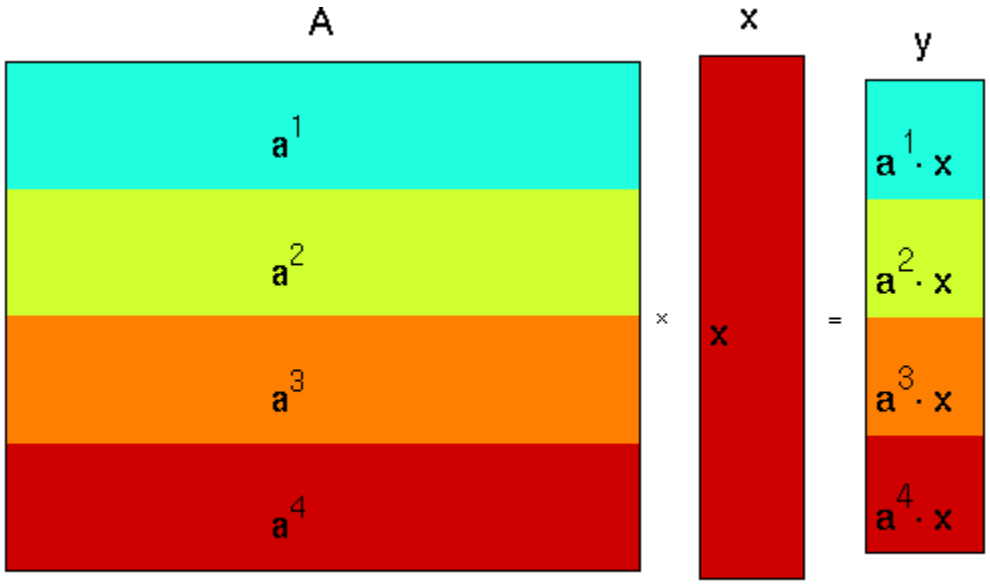
### 2.1. Συλλογή κλιμακωτών γινομένων

Αυτός είναι πιθανώς ο ποιό γνωστός τρόπος για τον πολλαπλασιασμό πίνακα-διανύσματος, **Ax**: ο πίνακας απεικονίζεται με γραμμές και το διάνυσμα ως στήλη.

Ας δούμε ένα παράδειγμα πολλαπλασιασμού ενός πίνακα 4-επί-5 με ένα διάνυσμα 5 στοιχείων:




Ας δούμε τον πίνακα να αποτελείται από γραμμές; δηλαδή 4 διανύσματα *1-επί-5* το ένα πάνω από στο άλλο. Σε αυτή την περίπτωση, ο πολλαπλασιασμός του πίνακα με το διάνυσμα-στήλη, **x**, μπορεί να υπολογισθεί σαν μια σειρά από (τέσσερα) βαθμοτά γινόμενα, δηλαδή των διανυσμάτων-γραμμών του **A** με το διάνυσμα **x**. Κάθε ένα από τα βαθμοτά γινόμενα είναι το αντίστοιχο στοιχείο του διανύσματος **y**. Επομένως, το διάνυσμα **y** παράγεται *στοιχείο προς στοιχείο*. Η παρακάτω υλοποίηση παρουσιάζει αυτή την ιδέα:

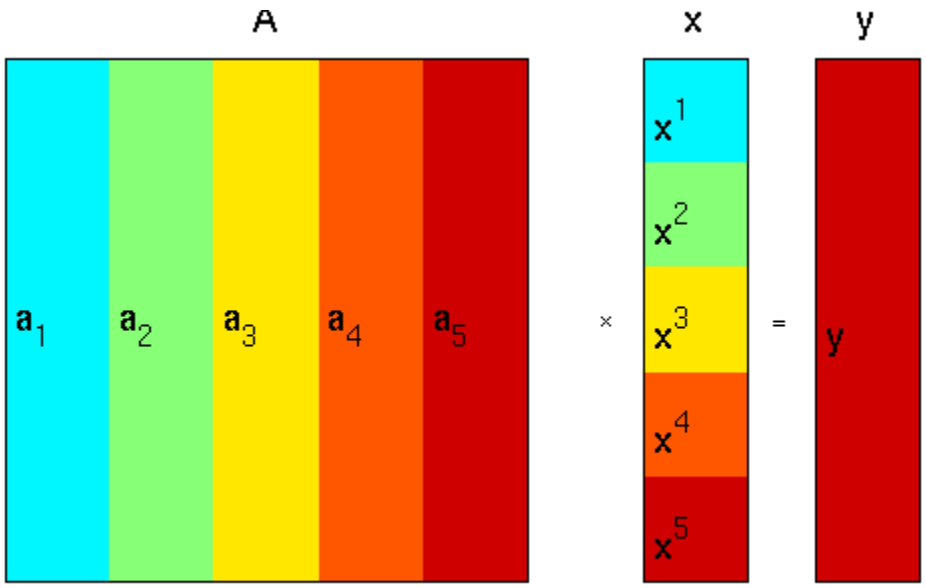


**Πότε προτιμάται αυτός ο τρόπος;**

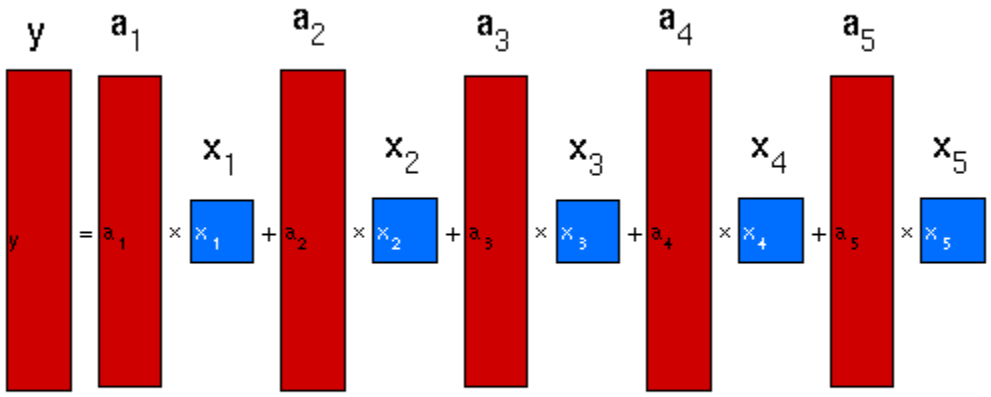
Υποθέστε πως έχουμε ένα 4-διάστατο χώρο και θέλουμε να ελέγξουμε ότι το διάνυσμα 5 στοιχείων, **x**, είναι κάθετο σε ένα 4-διάστατο χώρο. Έτσι μπορείτε να επιλέξετε οποιαδήποτε 4 γραμμικά ανεξάρτητα διανύσματα, να τα βάλετε σαν γραμμές σε ένα πίνακα **A** και τσεκάρετε αν το γινόμενο **Ax** δίνει ένα μηδενικό διάνυσμα.

## 2.2. Γραμμικός συνδυασμός των στηλών του πίνακα

Αυτή η μέθοδος είναι λιγότερη γνωστή (στους μαθητές), αλλά είναι χρήσιμη μέθοδος για τον πολλαπλασιασμό πίνακα με διάνυσμα. Ας θεωρήσουμε έναν πίνακα *5-επί-5* απεικονιζόμενο συναρτήσει των 5 στηλών του και ένα διάνυσμα 5 στοιχείων απεικονιζόμενο συναρτήσει των 5 γραμμών του.




Τότε, το διάνυσμα στήλης, **y**, που προκύπτει είναι ένας γραμμικός συνδυασμός (άθροισμα με βάρη) των στηλών του **A**, όπου τα στοιχεία του **x** είναι τα αντίστοιχα βάρη.



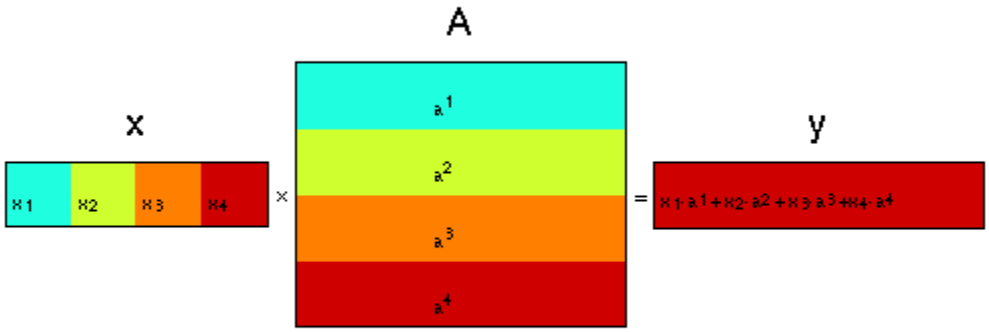

**Πότε προτιμάται αυτός ο τρόπος;**

- Πρώτα από όλα, βλέποντας το **Ax** με αυτό τον τρόπο, είναι ξεκάθαρο, ότι ο πίνακας απεικονίζει κάθε διάνυσμα **x** στον χώρο στηλών του πίνακα.
- Δεύτερον, εάν οι στήλες του **A** είναι γραμμικά ανεξάρτητες, τότε σχηματίζουν μια βάση του χώρου στηλών, και μπορούμε εύκολα να δούμε ότι τα στοιχεία του **x** και είναι οι συντεταγμένες του **y** ως προς την βάση.

## 2.3. Γραμμικός συνδυασμός των γραμμών του πίνακα

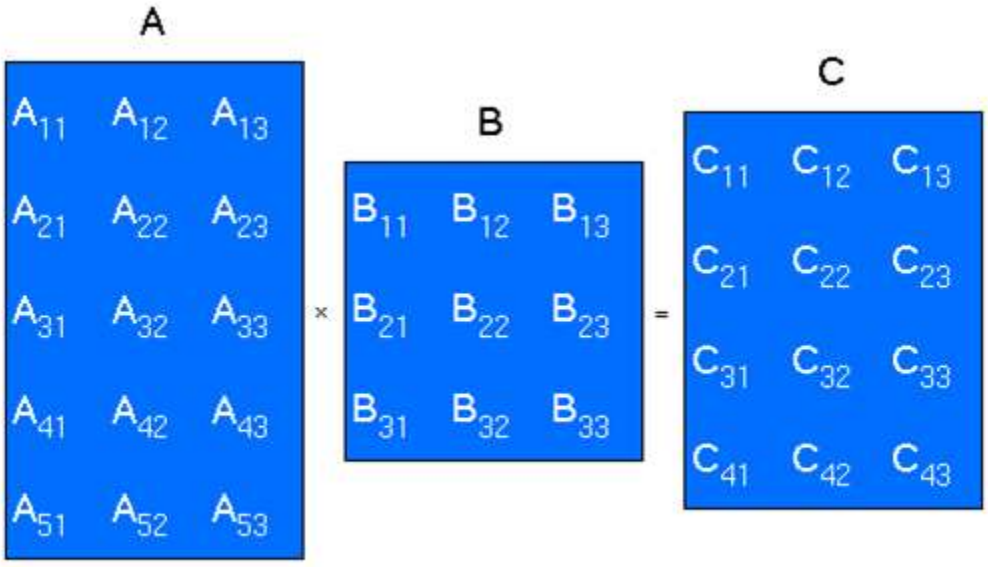
Μερικές φορές μας ενδιαφέρει περισσότερο ο συνδυασμός γραμμών από τον συνδυασμό στηλών του πίνακα. Αυτό επιτυγχάνεται με τον πολλαπλασιασμό *διανύσματος-πίνακα* **xA**, όπου **x** είναι ένα διάνυσμα-γραμμής και τον πίνακα τον βλέπουμε σαν ένα σύνολο γραμμών. Το σχήμα παρακάτω υλοποιεί αυτή την ιδέα.

Το διάνυσμα γραμμής **y** που προκύπτει είναι ένας γραμμικός συνδυασμός των γραμμών του **A** με τους συντελεστές που είναι τα στοιχεία του **x**:

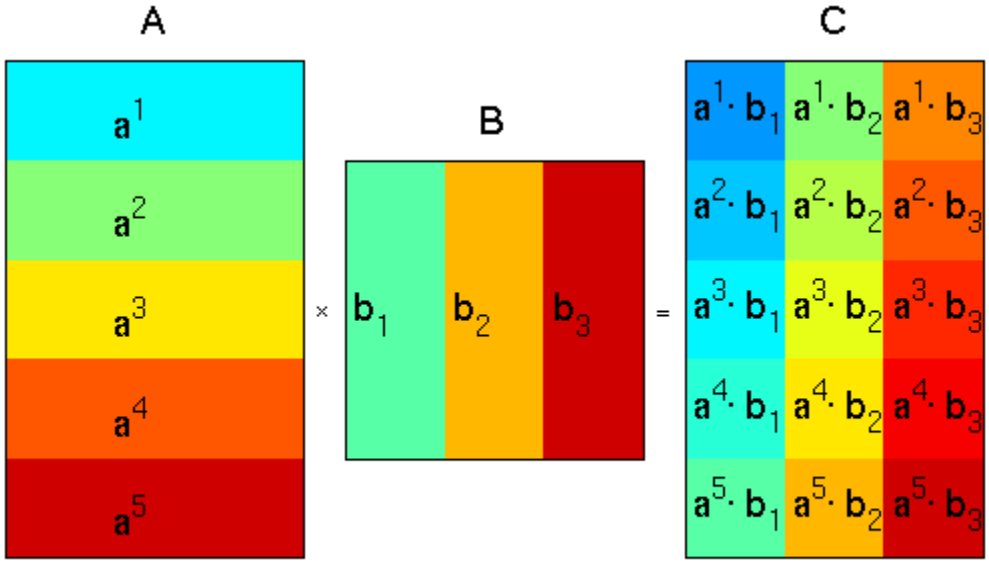



Γινόμενο πίνακα με πίνακα

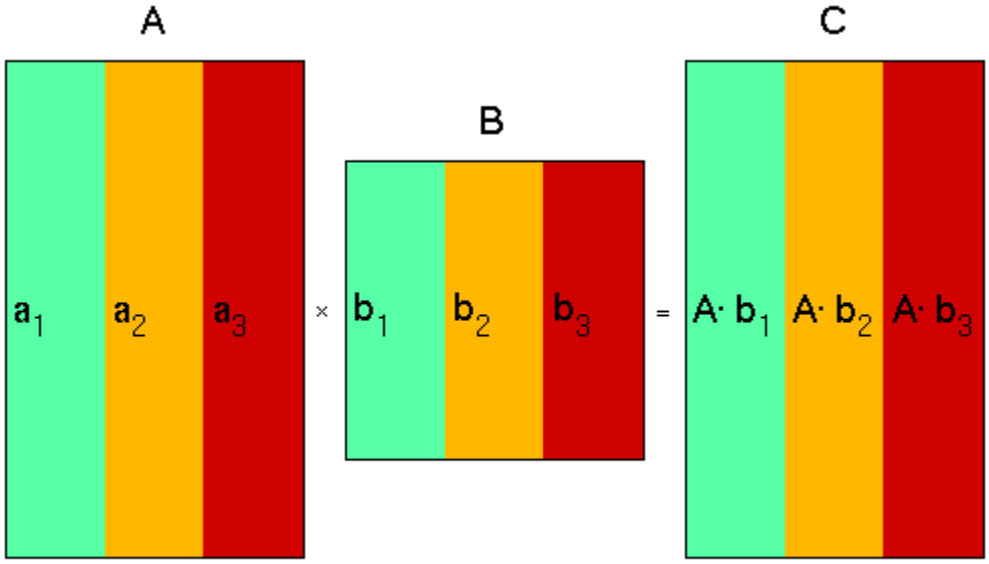
Υπάρχουν 4 τρόποι για να «δούμε» το γινόμενο δύο πινάκων, καθώς κάθε έναν από τους δύο πίνακες μπορούμε να τους δούμε σαν στήλες ή γραμμές. Ανάλογα με τις περιστάσεις, επιλέγουμε τον κατάλληλο τρόπο.



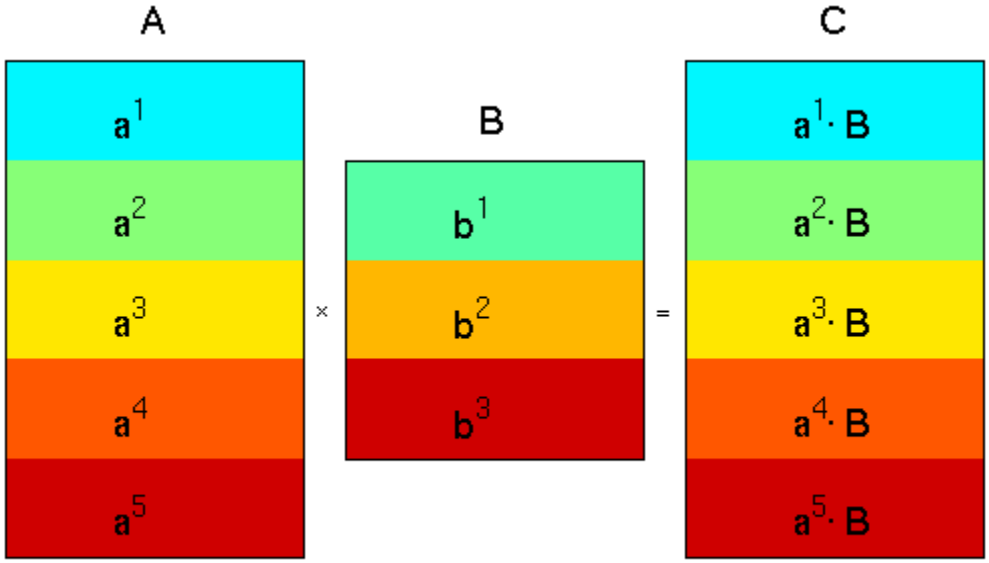
## 2.4. Εσωτερικά γινόμενα διανυσμάτων (γραμμή - στήλη)



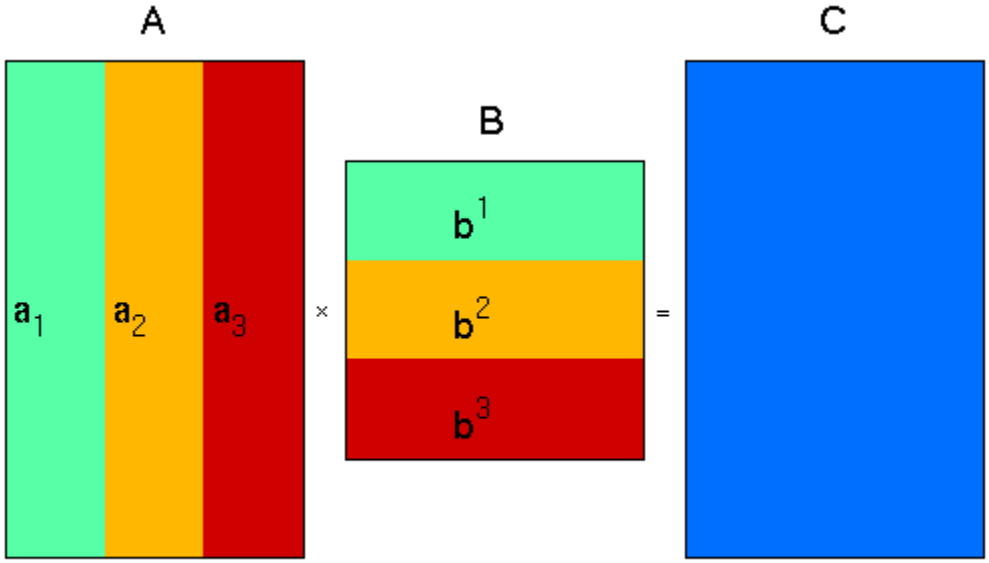

## 2.5. Γινόμενα πίνακα-διανύσματος (στήλη - στήλη)

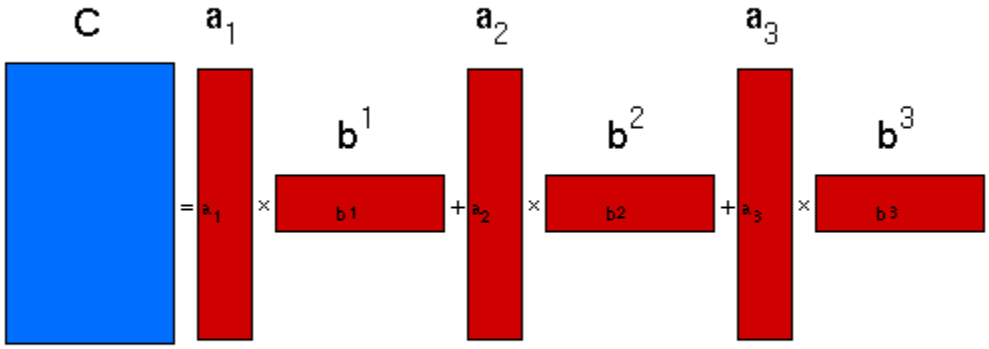


## 2.6. Γινόμενα διανύσματος-πίνακα (στήλη - στήλη)



## 2.7. Εξωτερικά γινόμενα διανύσματος (στήλη - γραμμή)




Ορθογώνιοι πίνακες

1. Ορθογώνια διανύσματα
Ένα ζευγάρι διανυσμάτων **x** και **y** είναι ορθογώνια εάν το *εσωτερικό γινόμενο* των είναι μηδέν.

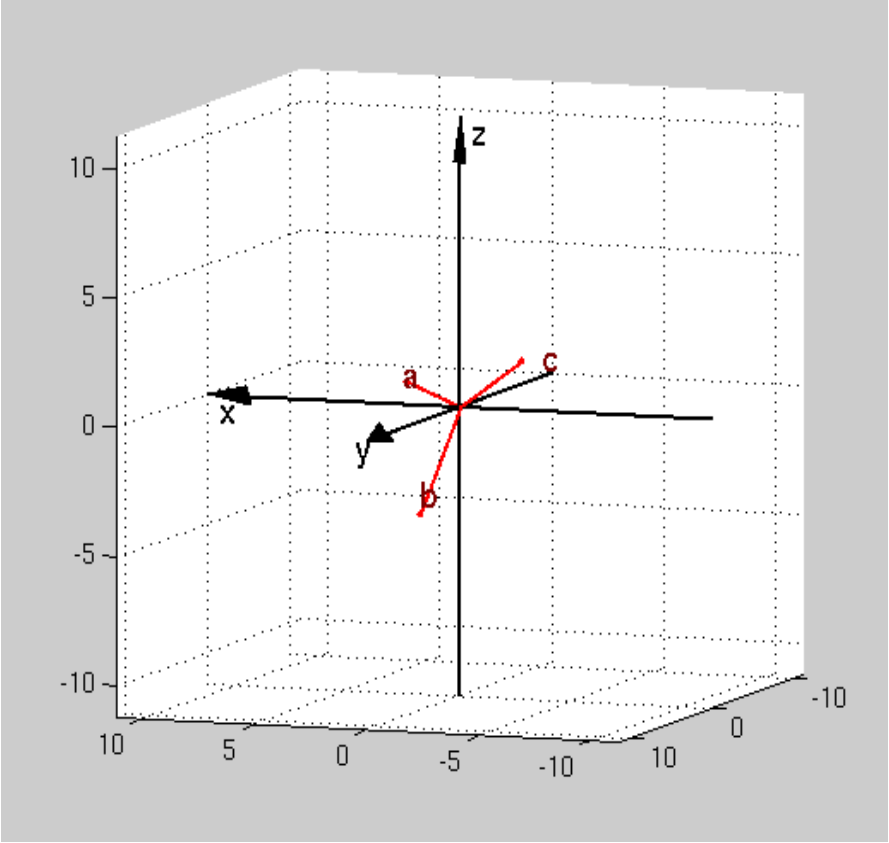
**Άσκηση 1:** Δίνεται ένα διάνυσμα a = [2 1 1]', βρείτε ένα ορθογώνιο διάνυσμα **b**. Είναι μοναδικό?

Ένα σύνολο από μη μηδενικά διανύσματα *S* καλείται *ορθογώνιο* εάν να στοιχεία του είναι ορθογώνια ανά ζεύγη, δηλαδή,

$x, y \in S, x \neq y \Rightarrow x^* y = 0.$

**Άσκηση 2:** Θεωρείστε 2 διανυσμάτα **a** και **b** του ορθογωνίου συνόλου της προηγούμενης Άσκησης. Επεκτείνετε το σύνολο με επιπλέον διανύσματα έτσι ώστε το σύνολο να παραμένει ορθογώνιο. Πόσα διανύσματα μπορείτε να προσθέσετε; Αναπαραστήστε τα.

```
a = [2 1 1]';
b = [1 2 -4]';
c = cross(a,b);
figure(1); clf;
drawVector([a b c], 'r.', {'a','b','c'});
view(-160, 7)
```




2. Ορθοκανονικά Σύνολα και Ορθοκανονικοί Πίνακες

Ένα σύνολο διανυσμάτων *S* είναι *ορθοκανονικό* εάν είναι ορθογώνιο και, επιπλέον όλα τα στοιχεία του έχουν μέγεθος ένα (ως προς κάποια νόρμα):

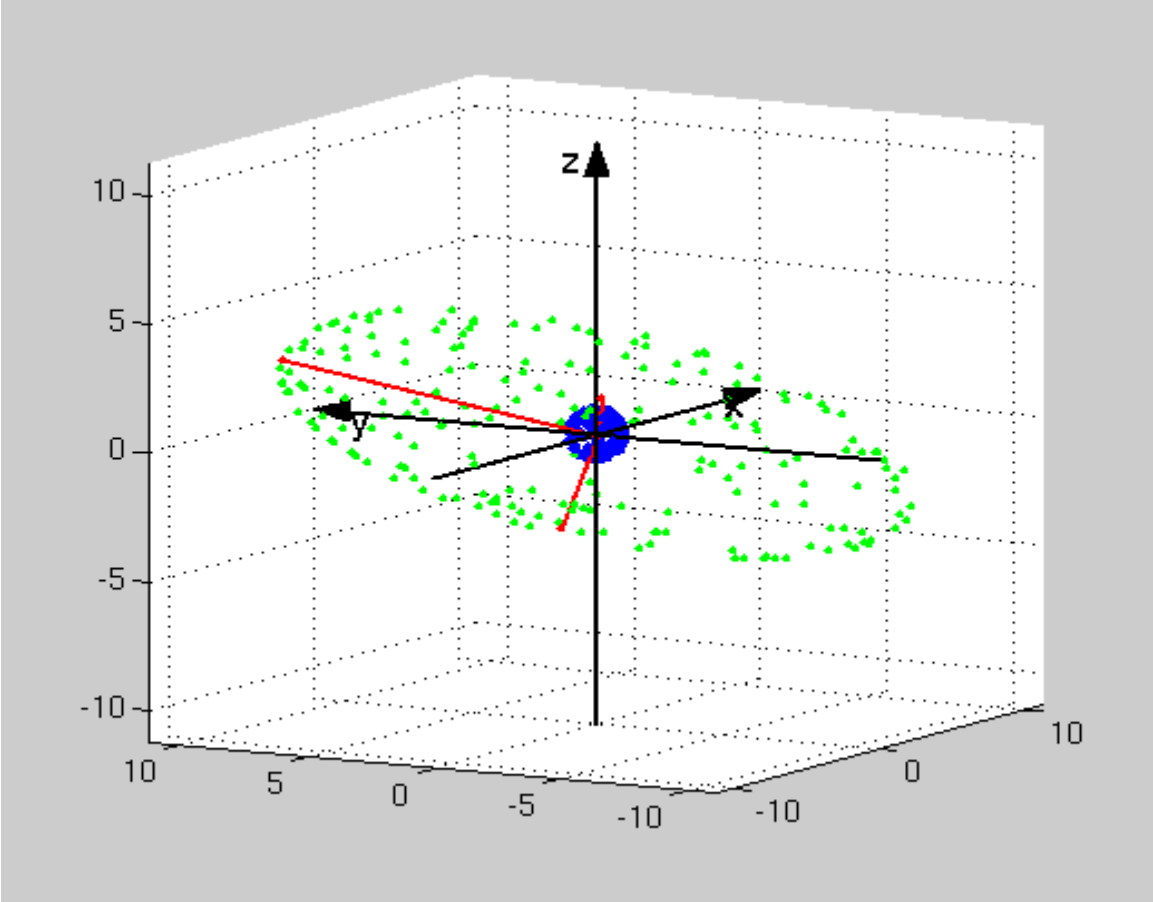
$||x|| = 1, x \in S.$

Ένας πίνακας *m-επί-n* είναι ορθογώνιος (ορθοκανονικός) εάν οι στήλες του αποτελούν ένα ορθογώνιο (ορθοκανονικό) σύνολο.

**Άσκηση 3:** Κατασκευάστε έναν ορθογώνιο πίνακα 3-επί-3 από το ορθογώνιο σύνολο των διανυσμάτων που δημιουργήσατε στην Άσκηση 2. Δείξτε τον μετασχηματισμό της μοναδιαίας σφαίρας για τον πίνακα. Στο ίδιο σχήμα, αναπαραστήστε τις στήλες του

πίνακα. Τι σημασία έχουν οι στήλες του πίνακα; Τι μπορείτε να πείτε για την νόρμα 2 του πίνακα;

```
A = [a b c];
X = sampleUnitSphere(3,200);
AX = A*X;
figure(1); clf; hold on;
plot3( X(1,:), X(2,:), X(3,:),'b.'); view(-60,10)
plot3(AX(1,:),AX(2,:),AX(3,:),'g.');
drawVector(A, 'r.')
hold off; axis square equal; grid on;
```




## 3. Ορθομοναδιαίοι πίνακες

Εάν οι στήλες του τετραγωνικού **μιγαδικού** πίνακα *m-επί-m* αποτελούν ένα ορθοκανονικό σύνολο, τότε ο πίνακας λέγεται ορθομοναδιαίος.

**Άσκηση 4:** Για τον ορθογώνιο πίνακα της Άσκησης 3, το οποίο συμβολίζουμε **B**, βρείτε έναν πίνακα 3-επί-3 **C** ο οποίος όταν πολλαπλασιάζεται με το **B** δίνει έναν ορθομοναδιάιο πίνακα **A**: A = B*C. Οπτικοποιείστε την μοναδιαία σφαίρα που μετασχηματίζεται από τον **A**. Οπτικοποιήσετε την μοναδιαία σφαίρα που μετασχηματίζεται για του πίνακες **AB**, **AAB** και **AAAB**. Τι μπορείτε να πείτε για τις ιδιότητες του **A**;

```
B = [a b c];
C = diag([1/norm(a) 1/norm(b) 1/norm(c)]);
A = B*C;
```

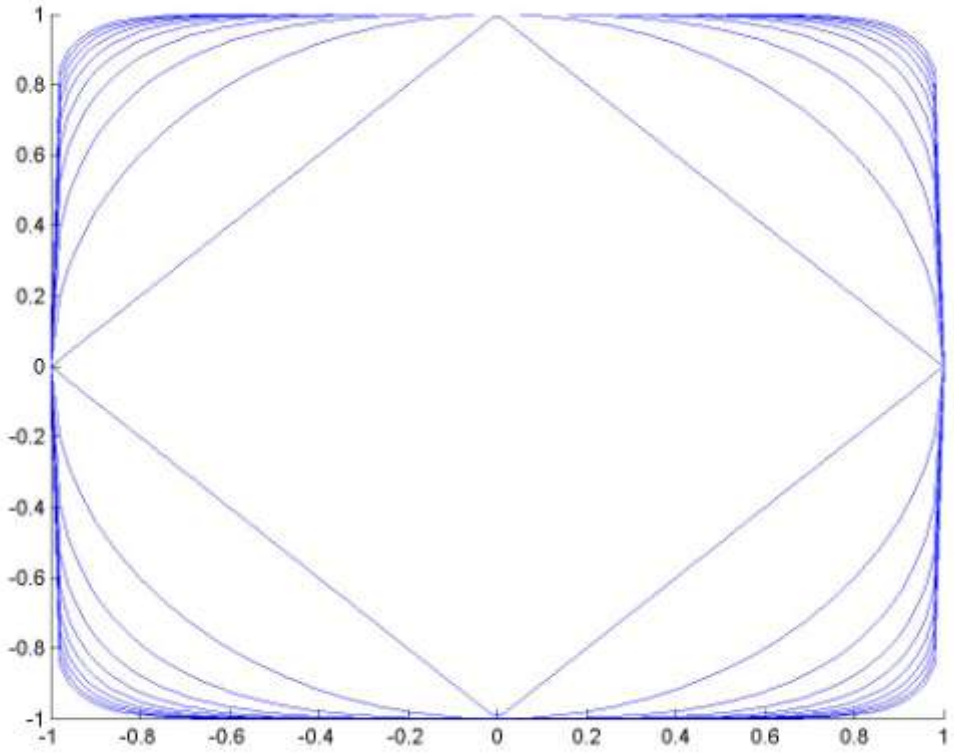
**Άσκηση 5:** Θεωρείστε τον ορθογώνιο πίνακα A = [1 -1; 1 1]. Αναπαραστήστε τις στήλες σε κανονικοποιημένη βάση. Μπορείτε να βρείτε τον αντίστροφο του **A** μόνο από το σχήμα; Ορθογωνοποιείστε τον **A** και προσθέστε τις ορθογώνιες στήλες στο σχήμα. Ποιος είναι ο αντίστροφος του **A** τώρα;

**Τέσσερις υποχώροι πινάκων: χώρος στηλών, χώρος γραμμών, μηδενικός χώρος και αριστερός μηδενικός χώρος.**

```
%% Four matrix subspaces
%% 1. Column space or range
% What is the domain and range of this matrix? Provide an illustration.
% For illustration you might need to augment the |x| by a 3d component and
% set to zero.
A = [ 1 2;
     0 3
     1 0 ];
x = [-3 2]';
figure(1); clf; hold on;
drawVector([ [x;0] A*x], {'x', 'Ax'});
%%
% Plot the domain
%%
% Plot the range
hold off
%% 2. Raw space
% Do the same analysis for the B = A'. What is the dimension of the column
% space of B?
B = A'
x = [-3 2 1]';
figure(1); clf; hold on;
drawVector([ x [B*x; 0] ], {'x', 'Bx'});
%%
% MATLAB: rank(B)
%%
% Plot the column space
hold off
%%
% The column space of *B* is the raw space of *A*.
%% 3. Null space
% Lets continue with the matrix B. Find a nontrivial vector x satisfying |B*x = 0|.
%%
% Plot the x and Bx.
figure(1); clf; hold on;
drawVector([x, [B*x;0] ], {'x', 'Bx'})
%%
% Is |x| a unique vector? What is the nullspace of B, what is its dimension? Draw it.
hold off
%%
% MATLAB: null(B)
%% 4. Left null space
% It is the null space of B'. function norms
```

## 4. Νόρμες Διανυσμάτων

```
%% 1. Vector norms
% Write a function that plots unit ball for a given norm p. Use this
% function to plot unit ball for p = 1:10;
figure(1); clf; hold on
for p=1:10
   plotUnitBall(p)
end
hold off

function plotUnitBall(p)
N = 100;
x = linspace(-1,1, N);
y = (1 - abs(x).^p).^(1/p);
x = [x x(end-1:-1:1)];
y = [y -y(end-1:-1:1)];
plot(x,y);
end
```




## 5. Νόρμες Πινάκων

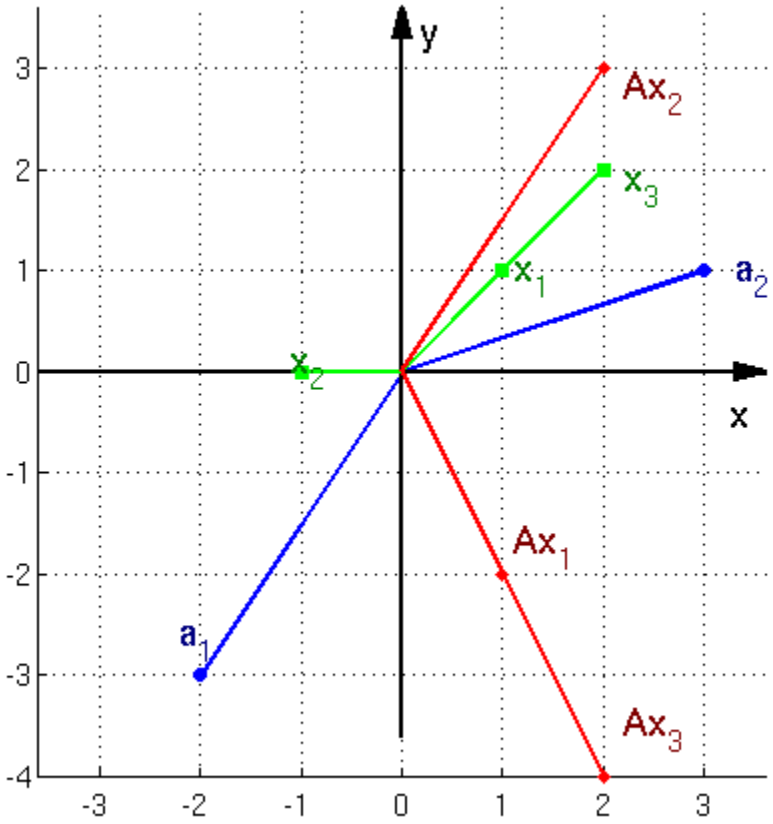
Στην προηγούμενη ενότητα, είδαμε πως ο πίνακας είναι μια γραμμική απεικόνιση του διανύσματος στον χώρο στηλών και μάθαμε για ένα σημαντικό βαθμωτό χαρακτηριστικό του πίνακα, την *τάξη* του. Τώρα, θα δούμε ακόμη ένα βαθμωτό χαρακτηριστικό του

πίνακα, την *νόρμα* πινάκων. Αλλά πρώτα, πρέπει να ξαναθυμηθούμε το γινόμενο πίνακα-διανύσματος.

1. Πολλαπλασιασμός πίνακα-διανύσματος

Ακόμη ένας τρόπος να δούμε τον πίνακα να δρα πάνω σε ένα διάνυσμα είναι να τον δούμε σαν συνδυασμό δύο πράξεων: **περιστροφή και κλιμάκωση**. Για τον γενικό πίνακα, οι γωνίες της περιστροφής και οι συντελεστές κλιμάκωσης είναι διαφορετικοί για διαφορετικά διανύσματα: εξαρτώνται από τις κατευθύνσεις των δεδομένων διανυσμάτων. Το παρακάτω παράδειγμα παρουσιάζει αυτή την ιδέα:

```
A = [-2  3;
    -3  1 ];
X = [1 -1 2;
    1  0 2 ];

figure(1); clf; hold on;
drawVector(A, {'a_1', 'a_2'});
drawVector(X, 'gs', {'x_1','x_2','x_3'});
drawVector(A*X, 'rd', {'Ax_1','Ax_2','Ax_3'});
hold off;
```




Το παραπάνω σχήμα προτείνει τον πιο άμεσο τρόπο για να δούμε τις ιδιότητες του πίνακα: παράγετε ένα σύνολο από διανύσματα εισόδου, **X**, και δείτε τις εικόνες τους, **AX**. Παρόλο που αυτός δεν είναι ο καλύτερος τρόπος για γενικούς πίνακες *m-επί-m*, οι 2-διάστατες και 3-διάστατες περιπτώσεις είναι εύκολες.

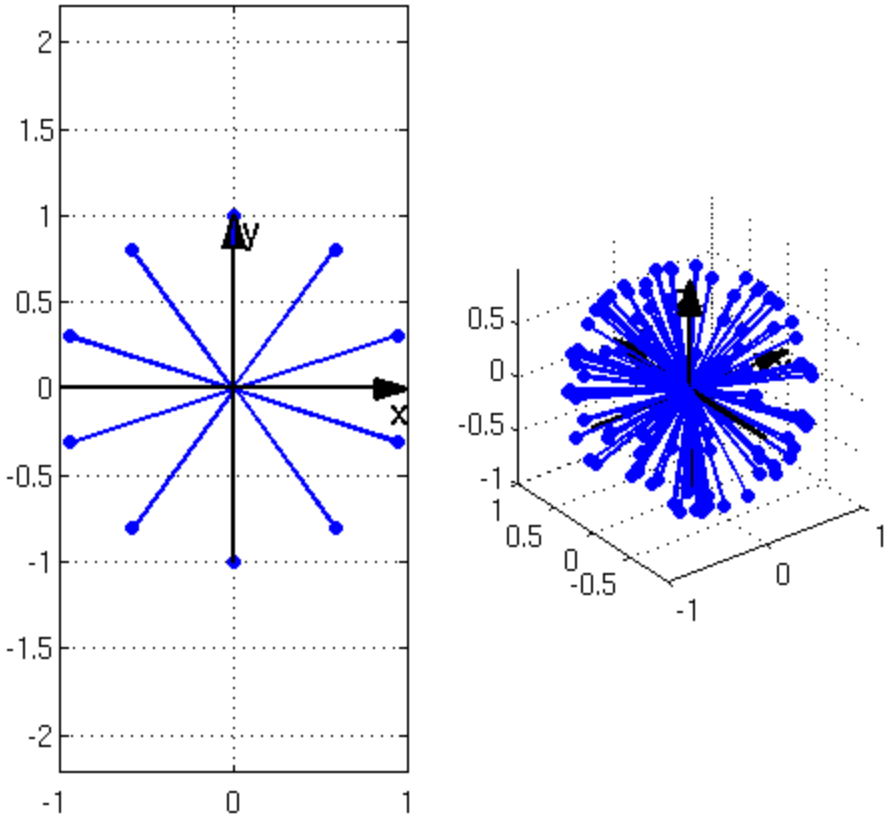
2. sampleUnitSphere()

Καθώς η περιστροφή και η κλιμάκωση εξαρτώνται μόνο από την διεύθυνση εισόδου, μπορούμε να περιορίσουμε τον χώρο του πίνακα της μοναδιαίας σφαίρας σαν ένα σύνολο,

$x \in R^d$ , $d = 2,3$ , $||x||_2 = 1$

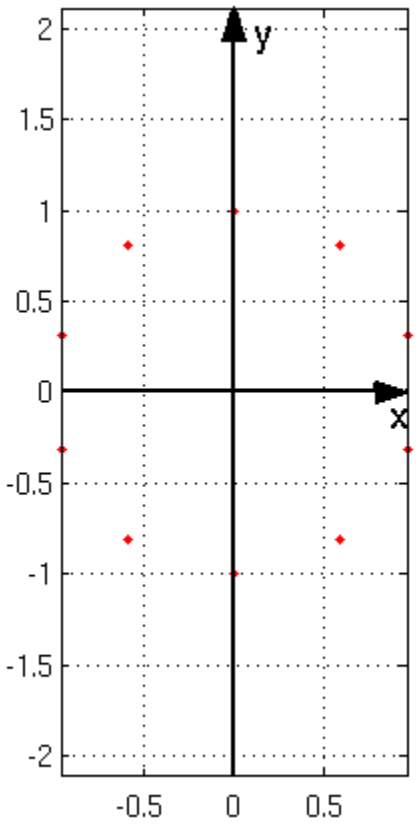
Μπορείτε εύκολα να δημιουργήσετε ένα τέτοιο σύνολο από *n* διανύσματα με την συνάρτηση sampleUnitSphere(d,n). Δείτε help sampleUnitSphere.

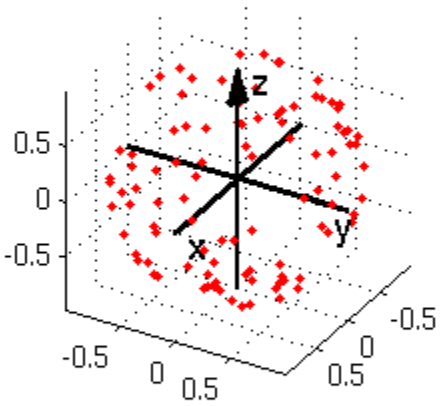
```
X2 = sampleUnitSphere(2,10);
X3 = sampleUnitSphere(3,100);
figure(1); clf;
subplot(1,2,1); drawVector(X2); axis square equal;
subplot(1,2,2); drawVector(X3); axis square equal;
```




Εντούτοις, με αυτά τα πολλά διανύσματα είναι καλύτερο να αποφύγουμε τις γραμμές χρησιμοποιώντας τις συναρτήσεις του MATLAB plot ή plot3:

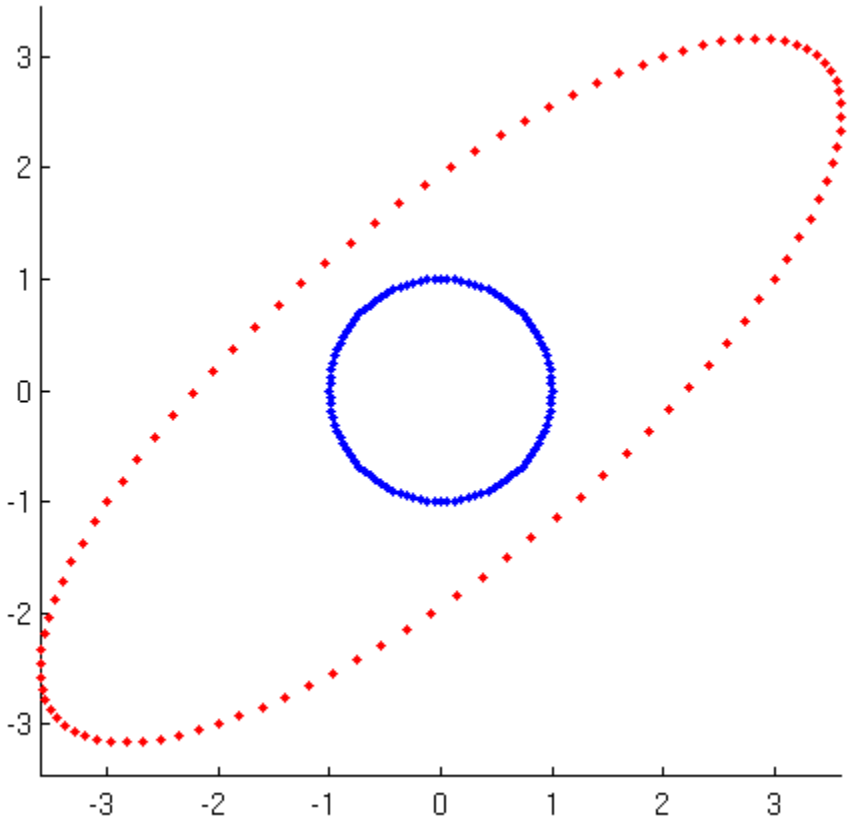
```
figure(1); clf;
subplot(1,2,1);  plot(X2(1,:),X2(2,:), 'r.'); axis square equal;
drawAxes(2,'k'); grid on;
subplot(1,2,2); plot3(X3(1,:),X3(2,:),X3(3,:),'r.'); axis square equal;
drawAxes(3,'k'); grid on;
view(120,30)
```




Ας επιστρέψουμε τώρα στον πρώτο πίνακα, και ας δούμε την εικόνα του στην 2-διάστατης σφαίρα:

```
A = [-2  3;
    -3  1 ];
X = sampleUnitSphere(2, 100);
AX = A*X;
figure(1); clf; hold on;
plot( X(1,:), X(2,:),'b.');
plot(AX(1,:),AX(2,:),'r.');
hold off
axis square equal
```




Υπάρχει ένα όχι απλό θεώρημα της γραμμικής άλγεβρας το οποίο αποδεικνύει πως ο μετασχηματισμός της μοναδιαίας σφαίρας είναι μια *υπερέλλειψη*.

**Άσκηση 1:** Θεωρείστε δύο πίνακες B = [1 3; 1 1] και C = reshape(1:4,2,2), των οποίων τις ιδιότητες συγκρίνουμε γραφικά (σε ένα σχήμα) με τις ιδιότητες του πίνακα A παραπάνω. Προσθέστε τίτλο στο σχήμα. Ποια είναι η διαφορά αυτών των πινάκων; μπορείτε να δείτε ποιος από τους πίνακες είναι μη – αντιστρέψιμος (singular);

**Άσκηση 2:** Να συγκριθεί ο πίνακας **B** από την προηγούμενη Άσκηση με τους πίνακες **-B** και **B'**. Συμβαδίζουν τα αποτελέσματα της ανάλυσής σας με τις προσδοκίες σας; (Πρώτα, πρέπει να προσδιορίσετε τις «προσδοκίες» σας).

**Άσκηση 3:** Περισσότερες πληροφορίες για έναν πίνακα μπορούμε να έχουμε αν αναλύσουμε τις δυνάμεις των πινάκων, π.χ., επαναλαμβανόμενη εφαρμογή ενός πίνακα πάνω σε ένα διάνυσμα: Ax, AAx, AAAx, .... Ας μελετήσουμε τις δυνάμεις των πινάκων **B** και **A** από τις προηγούμενες εργασίες. Δώστε ξανά την εικόνα των *επαναλαμβανόμενων* μετασχηματισμών τις μοναδιαίας σφαίρας για τους πίνακες. Δείξτε μέχρι και 6 μετασχηματισμούς (συμβουλή: χρησιμοποιείστε βρόγχους for). Χρησιμοποιείστε διαφορετικά σχήματα για κάθε πίνακα. Τα αποτελέσματα για τον κάθε πίνακα είναι διαφορετικά; Γιατί;

**Άσκηση 4:** Ερευνήστε γραφικά τις ιδιότητες των δύο 3-διάστατων πινάκων: A = [1 3 -2; 8 0 1; 2 2 6 ] and B = reshape(1:9,3,3). Μπορείτε να δείτε τις τάξεις των πινάκων; Οπτικοποιήσετε τον χώρο στηλών του μη αντιστρέψιμου πίνακα.

3. p-νόρμα διανύσματος με βάρη

Γνωρίζετε ήδη την πιο βασική κλάση για τις νόρμες διανυσμάτων, τις *p*-νόρμες. Άλλες σημαντικές νόρμες διανυσμάτων είναι οι *p-νόρμες με βάρη*. Δεδομένου ενός αντιστρέψιμου πίνακα **W**, η *p*-νόρμα με βάρη ορίζεται σαν:

$||x||_W = ||Wx||$

**Άσκηση 5:** Μπορείτε να βρείτε τον *μοναδιαίο* διάνυσμα **x** με την μεγαλύτερη 2-νόρμα με βάρος τον πίνακα A = diag([1 2 3]; Τι συμβαίνει αν χρησιμοποιήσετε ως βάρος τον πίνακα **A** από την προηγούμενη εργασία; Είναι πιο δύσκολο να βρούμε το μεγαλύτερο διάνυσμα, αλλά μπορείτε να ορίσετε το ελάχιστο μέγεθος με βάρη του **A;**

4. Νόρμες πινάκων

Αν έχετε λύση την Άσκηση 5, θα έχετε βρεί την 2-νόρμα του πίνακα A. Οι «induced» νόρμες είναι συνήθως νόρμες πινάκων. Η πιο σημαντική νόρμα πίνακα η οποία δεν είναι induced από ένα διάνυσμα είναι η *νόρμα Frobenius*.

4.1 Νόρμες Induced

Δεδομένων δύο p-norms διανυσμάτων, μια induced νόρμα ενός γενικού πίνακα *m-επί-n* **A** ορίζεται όπως παρακάτω:

$$||A||_{(m,n)} = \sup_{x \in C^n} \frac{||Ax||_{(m)}}{||x||_{(n)}} = \sup_{x \in C^n, ||x||=1} ||Ax||_{(m)}$$

Με άλλα λόγια, η induced νόρμα πίνακα είναι ο μέγιστος παράγοντας με τον οποίο ο A μπορεί να «τεντώσει» (ή να «συμπιέσει») ένα διάνυσμα **x.**

4.2 Νόρμα Frobenius

Η νόρμα *Frobenius* ή *Hilbert-Schmidt* ορίζεται ως εξής

$$||A||_F = (\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}^2|)^{1/2}$$

Παρατηρείστε πως είναι ίδια με την 2-νόρμα ενός διανύσματος που παράγεται από έναν πίνακα διάστασης *mn*. Στο MATLAB:

$||A||_F = \mathtt{norm}(A(:), 2)$

**Άσκηση 6:** Όπως οι νόρμες διανυσμάτων, οι νόρμες πινάκων μπορούν να χρησιμοποιηθούν για να περιορίσουμε το γινόμενο των πινάκων. Για τα παραδείγματα και τους πίνακες **A** και **B** από την Άσκηση 4, αποδείξτε το για κάθε νόρμα πίνακα:

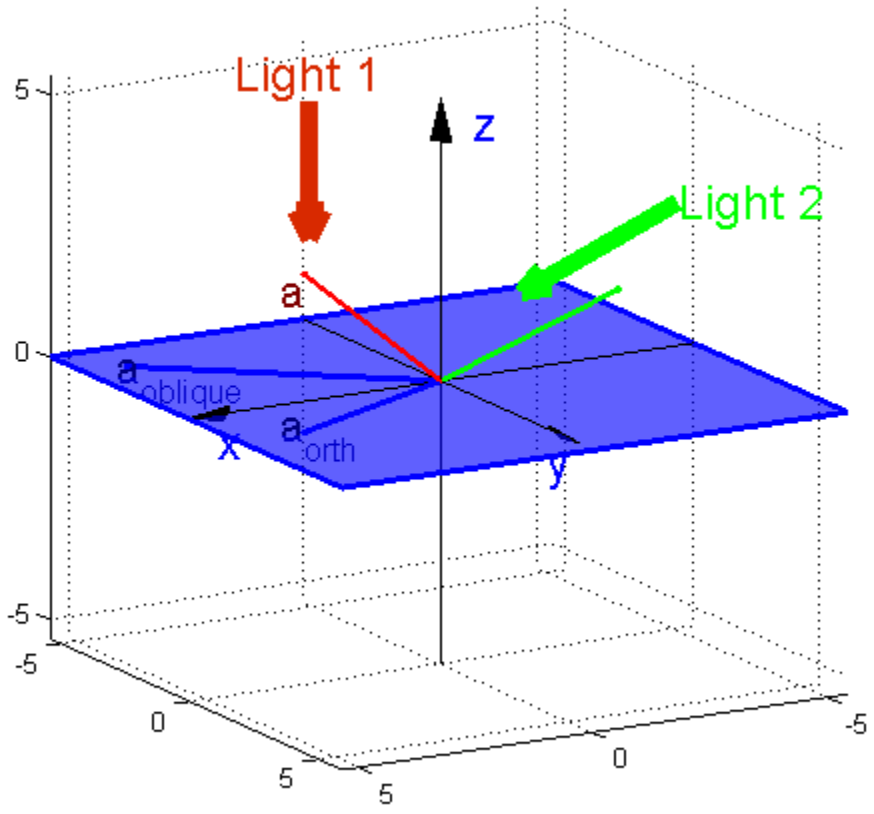
$||AB|| \leq ||A|| \cdot ||B||$.

## 6. <u>Προβολές</u>

Σε αυτή την ενότητα θα μάθουμε δύο τύπους προβολών: μια γενική *πλάγια προβολή* και την ειδική περίπτωση της *ορθογώνιας προβολής*. Η παρακάτω εικόνα παρουσιάζει και τις δύο περιπτώσεις. Ένα διάνυσμα a = [4 2 3]' φωτίζεται με δύο φωτινές πηγές απο δύο διαφορετικές κατευθύνσεις: α) κατά μήκος του άξονα *z* και β) κατά μήκος του πράσινου διανύσματος. Επομένως, το **a** έχει δύο σκιές στο επίπεδο *xy*. Η σκιά που προέρχεται από το πρώτο φως είναι κάθετο στο επίπεδο *xy* και είναι ορθογώνια προβολή του **a** στο επίπεδο. Η σκιά από το δεύτερο φώς πέφτει πάνω στο επίπεδο με πλάγια γωνία και είναι η πλάγια προβολή του **a**. Ποια από τις προβολές είναι πιο κοντά στο **a**;

```
A = [1 0;
   0 1;
   0 0];                % The 2-by-3 matrix
```

```
a = [4 2 3]';                % Vector to project
n = [-.5 6 3]';              % Vector for the oblique projection

figure(1); clf; hold on;     % Plot the vectors
drawVector(a, 'r.', {'a'});
drawVector(n, 'g.')

P_orth = A*A';                          % Orthogonal projector
P_oblique = A*[1 0 0;0 1 0]*inv([A n]);  % Oblique projector
a_orth    = P_orth   *a;                % Orthogonal projection
a_oblique = P_oblique*a;                % Oblique projection
% Plot the projections of vector a
drawVector([a_orth a_oblique], 'b.', {'a_{orth}', 'a_{oblique}'})
drawSpan(A)
hold off
view(150, 15)
```




## 6.1. Ορθογώνιες προβολές

**Άσκηση 1:** Θεωρείστε τους πίνακες A = [1 0; 0 1; 1 0] και B = |1 2; 0 1; 1 0]. Απαντήστε στις παρακάτω ερωτήσεις (δοκιμάστε την πρώτη κάνοντας υπολογισμούς με το χέρι).

- Ποια είναι η ορθογώνια προβολή **P** στο χώρο στηλών του **A**, και ποια είναι η εικόνα του διανύσματος a = [1 2 3]' ως προς **P;**
- Ίδια ερώτηση για τον **B**.

Οπτικοποιήστε τα αποτελέσματα.

## 6.2. Πλάγιες προβολές

**Άσκηση 2:** Για τους ίδιους πίνακες **A** και **B** και για το διάνυσμα **a** από την Άσκηση 1, βρείτε τις *πλάγιες* προβολές των οποίων οι προβολές των 3-επί-1 διανυσμάτων στους χώρους στηλών των **A** και **B** πάνω στο διάνυσμα **a**. Οπτικοποιήστε τις προβολές των διανυσμάτων.

**Άσκηση 3**
Εάν **P** είναι η ορθογώνια προβολή, τότε το **I - 2P** είναι τελεστής. Αποδείξτε το παραπάνω αλγεβρικά και δώστε την γεωμετρική ερμηνεία. Βρείτε δύο παραδείγματα σε 2 και 3 διαστάσεις και δώστε τις υλοποιήσεις τους.
**Άσκηση 4**
Θεωρείστε **E** έναν *m-επί-m* πίνακα ο οποίος αφαιρεί το «άρτιο τμήμα» ενός διανύσματος *m*-διάστασης:
$Ex = (x + Fx)/2$
όπου **F** είναι ένας *m-επί-m* πίνακας που μετασχηματίζει x = [x1... xm]' στο [xm ... x1]'. Υπολογίστε τους πίνακες **F** και **E** και απαντήστε στις παρακάτω ερωτήσεις χρησιμοποιώντας μόνο αριθμητικά επιχειρήματα:
Είναι η **E** ορθογώνια προβολή, πλάγια προβολή ή όχι προβολή;

### 7. <u>Αντίστροφος πίνακας</u>

Ορισμός
Για να έχουμε έναν αντίστροφο, ένας *m*-επί-m πίνακας πρέπει να είναι πλήρους τάξης, δηλαδή, όλες οι στήλες του είναι γραμμικά ανεξάρτητες και δημιουργούν μια πλήρη βάση για το χώρο *m*-διαστάσεων. Επιπλέον, κάθε *m*-διάνυσμα μπορεί να αναπαρασταθεί σαν γραμμικός συνδυασμός των στηλών. Συγκεκριμένα, το κανονικοποιημένο μοναδιαίο διάνυσμα, **e_j**, με "1" στην *j*-στη είσοδο και μηδέν αλλού, μπορεί να γραφτεί:
$e_j = A \cdot x$
Το διάνυσμα **x** είναι η *j* στήλη του αντίστροφου του **A**. Εάν ορίσουμε τον αντίστροφο πίνακα σαν
$A^{-1}$ τότε είναι γνωστό ότι
$A^{-1}A=AA^{-1}=I$
όπου I είναι ο ταυτοτικός πίνακας και μπορεί να παραχθεί με την ρουτίνα του MatLab eye(m).
Αξίζει να σημειώσουμε πως η λύση της εξίσωσης:
$A \cdot x = b$
μπορεί να ορισθεί σαν το διάνυσμα των συντελεστών της επέκτασης του **b** στις στήλες του **A**, και μπορεί να βρεθεί αν εφαρμόσουμε τον αντίστροφο του **A** πάνω στο **b**:

$x = A^{1} \cdot b$

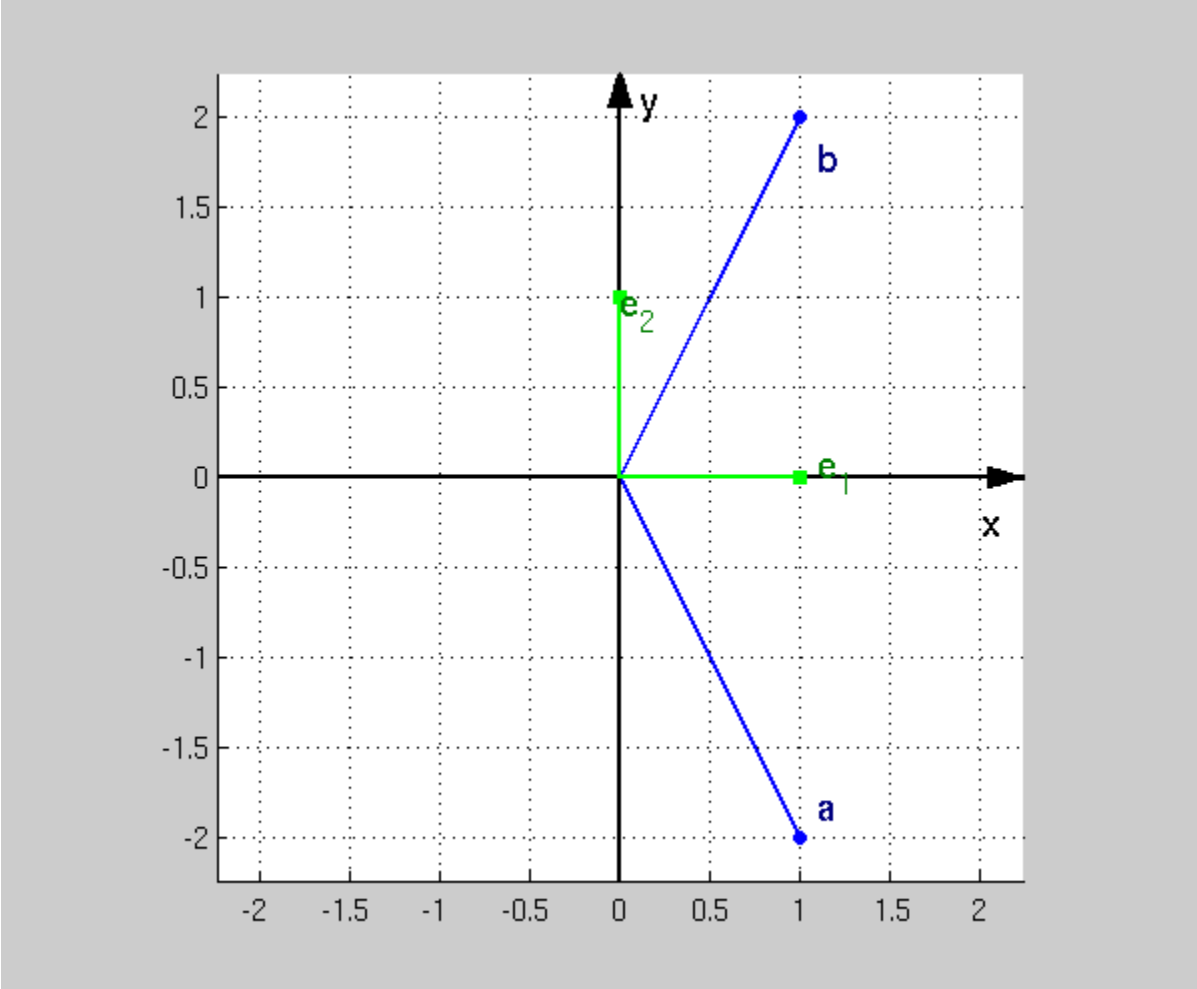
Παράδειγμα 2 διαστάσεων

Θεωρείστε τον πίνακα A = [1 1; -2 2].

```
A = [  1 1;
    -2 2 ];
```

Σχεδιάστε τις στήλες και την κανονικοποιημένη βάση του:

```
e1 = [1;0]; e2 = [0;1];
figure(1); clf; hold on;
drawVector(A, {'a','b'});
drawVector([e1 e2], 'gs', {'e_1', 'e_2'})
hold off;
```

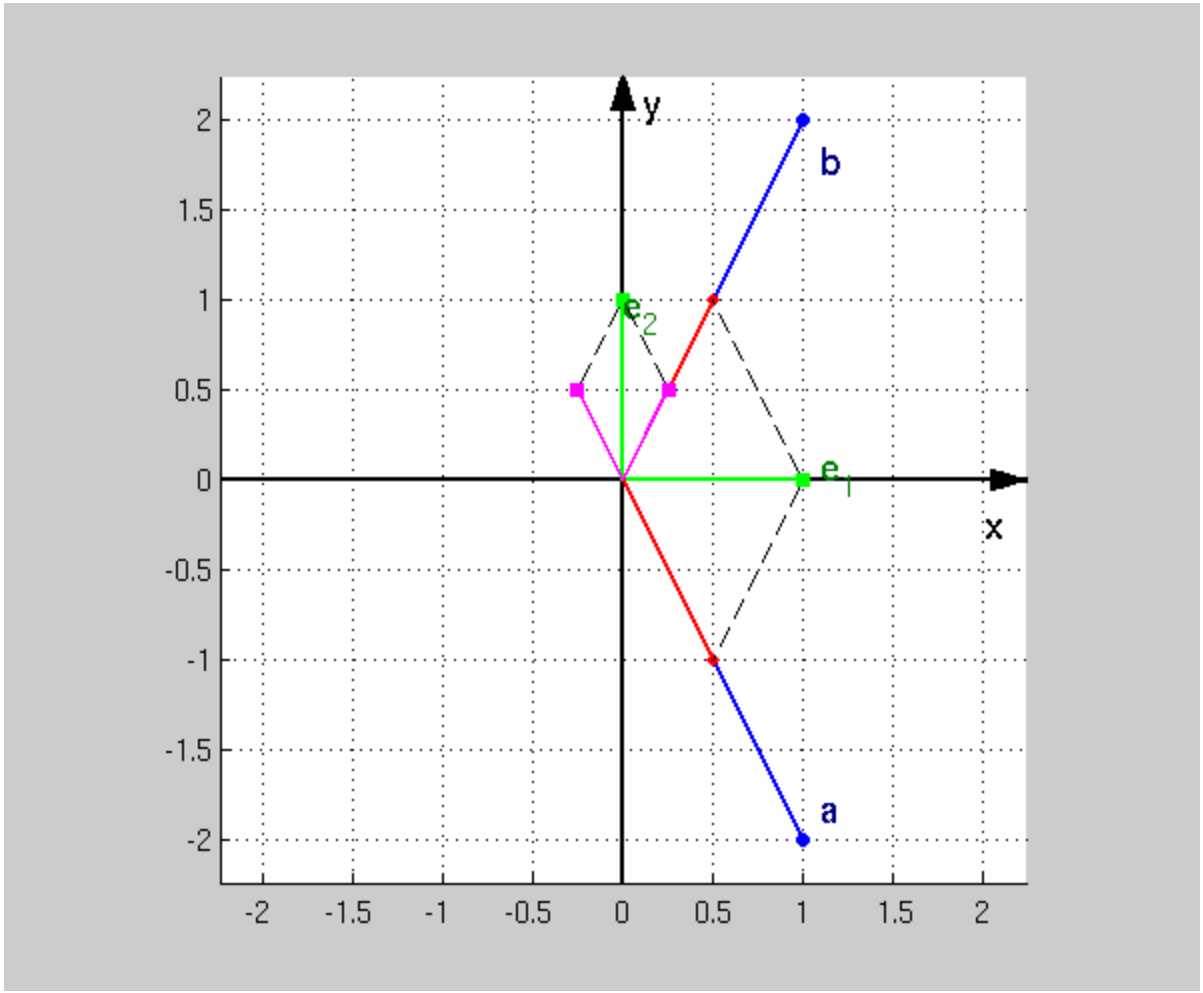



**Άσκηση 1:** Μπορείτε να μαντέψετε τα στοιχεία της πρώτης στήλης του αντίστροφου πίνακα γεωμετρικά; Μπορείτε να κάνετε το ίδιο για την δεύτερη στήλη; Μπορείτε να βρείτε τα πρόσημα των στοιχείων του;

**Άσκηση 2:** Υπολογίστε τον αντίστροφο με την συνάρτηση του MATLAB inv() και ελέγξτε την απάντηση σας στην Άσκηση 1. Δεδομένου του αντίστροφου πίνακα, βρείτε τους συντελεστές του **e_1** για τις στήλες του **A** και σχεδιάστε τις *πλάγιες* προβολές του **e_1**. Το σχήμα σας θα πρέπει να μοιάζει με το παρακάτω.

```
invA = inv(A)              % Compute inverse of A
P1 = A*diag(invA(:,1));    % Find the *oblique projections* of e_1
P2 = A*diag(invA(:,2));    % and of e_2 onto the columns of A

figure(1); hold on;
% Plot the projections
drawVector(P1, 'rd')
drawVector(P2, 'ms')
drawLine([e1 P1(:,1)]);    % Indicate the projection lines for e_1
drawLine([e1 P1(:,2)]);
drawLine([e2 P2(:,1)]);    % and for e_2
drawLine([e2 P2(:,2)]);
hold off
```

```
invA =

    0.5000   -0.2500
    0.5000    0.2500
```



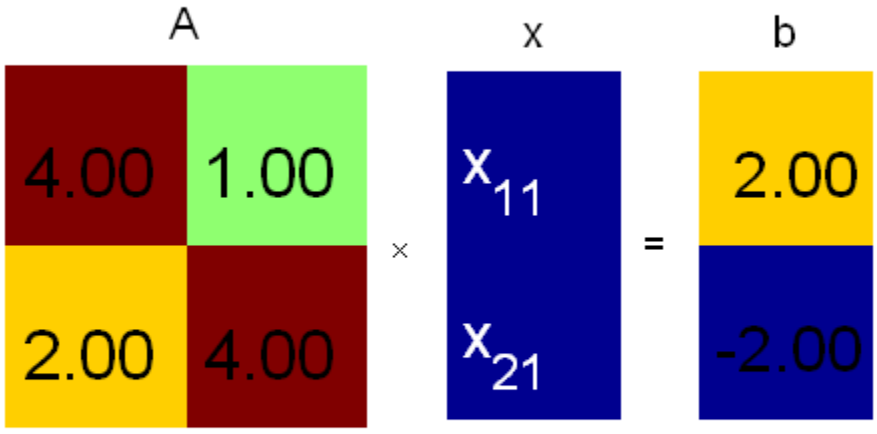

## 8. Παραγοντοποίηση LU

Σε αυτή την ενότητα, θα μελετήσουμε τις μεθόδους για την επίλυση των συστημάτων των γραμμικών εξισώσεων (ΕΓΕ). Ξεκινάμε με την αρχαιότερη, που δεν είναι και η καλύτερη μέθοδος: Μέθοδος απαλοιφής του Gauss. Πρώτα θα δώσουμε μια γεωμετρική απεικόνιση της λύσης ενός γραμμικού συστήματος και μετά θα περιγράψουμε την λύση ενός τριγωνικού συστήματος, που είναι εύκολο να επιλυθεί.

## 8.1. Σύστημα γραμμικών εξισώσεων (ΣΓΕ)

Τα ΣΓΕ στην μορφή πίνακα ορίζονται από μια εξίσωση: **Ax** = **b**, όπου ο **A** είναι ένας *m-επί-n* πίνακας και **b** και **x** είναι *m*- και *n*-διανύσματα, αντίστοιχα. Σε αυτό την ενότητα, θα χειριστούμε μόνο τετραγωνικούς πίνακες *m-επί-m* matrices, .π.χ.,

```
A = [ 4 1;
      2 4 ];
b = [2 -2]';
xnn = [nan,nan]';
figure(1);clf;
dispMEq('A*x=b', A,xnn, b)
```




Συμβολικά, η λύση ενός ΣΓΕ γράφεται σαν **x** = inv(**A**)b. Εντούτοις, πρακτικά το **x** δεν βρίσκεται σχεδόν ποτέ υπολογίζοντας τον αντίστροφο του **A**. Αλλά, ο **A** (και το **b**) μετατρέπεται σε μια παρόμοια και πιο απλή μορφή και έτσι η λύση βρίσκεται πιο εύκολα. Θα μάθουμε διάφορες τέτοιες μεθόδους αναγωγής που διαφοροποιούνται ανάλογα με την γεωμετρική τους ερμηνεία.

## 8.2. Δύο γεωμετρικές ερμηνείες της λύση ενός ΣΓΕ

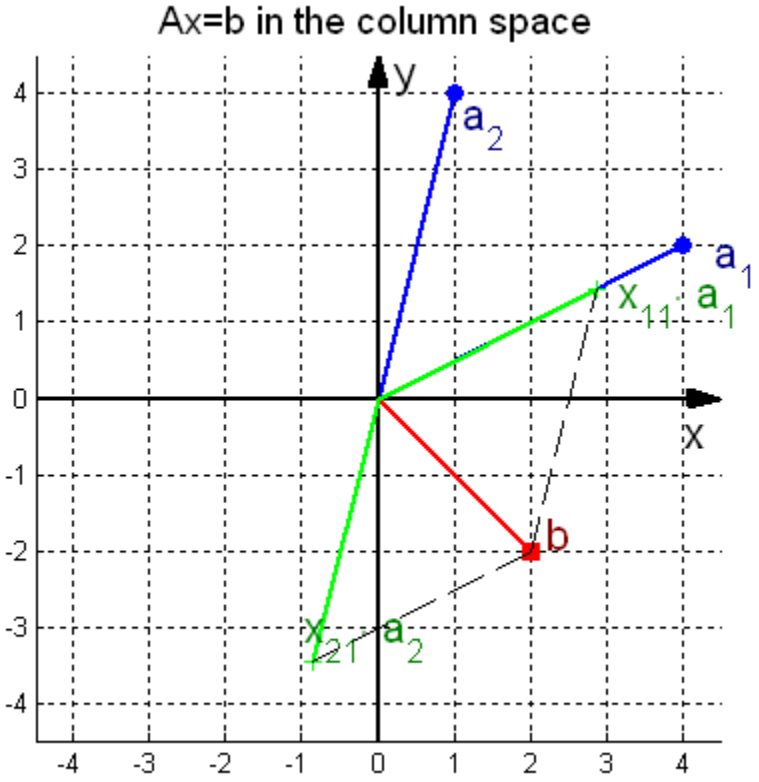
Υπάρχουν δύο τρόποι να δείτε ένα ΣΓΕ **Ax** = **b**: κατά γραμμές και κατά στήλες.

### *8.2.1 Ερμηνεία στο χώρο στηλών του A*

Θεωρούμε το διάνυσμα **b** σαν γραμμικός συνδυασμός των στηλών του **A**, και την λύση, **x**, να περιέχει τα βάρη του γραμμικού συνδυασμού. Η παρακάτω εικόνα παρουσιάζει την ιδέα του συστήματος για το προηγούμενο παράδειγμα.

```
A = [ 4 1;2 4 ];
b = [2 -2]';
x = A\b;
figure(2); clf; hold on;
drawVector(A, {'a_1', 'a_2'})
drawVector(b, 'rs', {'b'});
Pb = A*diag(x);
drawVector(Pb, 'g+', {'x_{11}\cdot a_1', 'x_{21}\cdot a_2'});
drawLine([b Pb(:,1)]); drawLine([b Pb(:,2)]);hold off;
title('Ax=b in the column space', 'FontSize', 14);
```

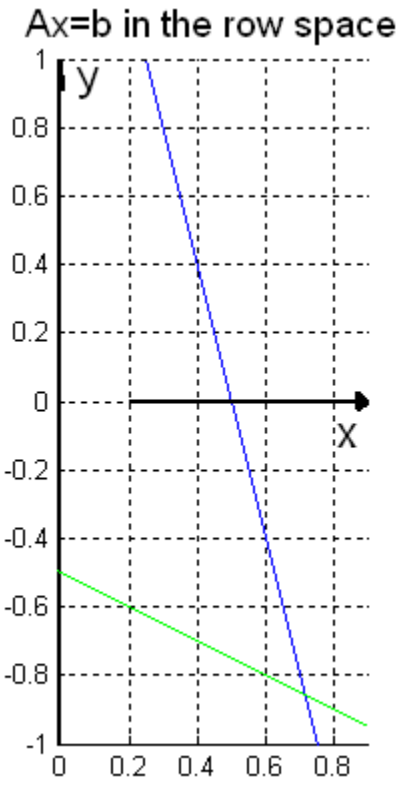
```
ans =
   p: [171.0043 174.0043]
   l: [172.0043 175.0043]
   t: [173.0043 176.0043]
```



### *8.2.2 Ερμηνεία στο χώρο γραμμών του A*

Εναλλακτικά, κάθε γραμμή του **Ax** = **b** μπορεί να θεωρηθεί ως η εξίσωση που ορίζει ένα επίπεδο στο χώρο των *n*-διαστάσεων. Η λύση **x** είναι το σημείο όπου όλα τα *m* επίπεδα τέμνονται. Στο παράδειγμά μας, μπορούμε να οπτικοποιήσουμε την λύση του ΣΓΕ με τον παρακάτω τρόπο:

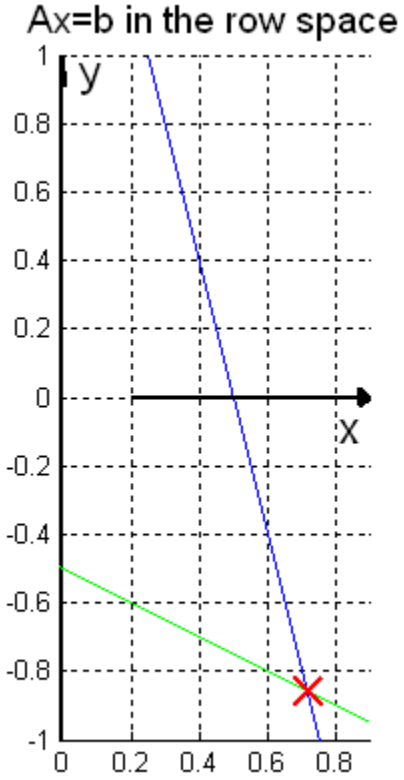
```
A = [ 2 4;
    3 6 ];
b = [4 7]';
figure(3); clf; hold on;
drawPlane(A(1,:), b(1), 'b');
drawPlane(A(2,:), b(2), 'g');
hold off
title('Ax=b in the row space', 'FontSize', 14);
```




Όπου η λύση είναι το σημείο που ικανοποιεί όλες τις εξισώσεις.

```
A = [ 2 4;
    3 6 ];
b = [4 7]';
x = A\b;
figure(3); clf; hold on;
drawPlane(A(1,:), b(1), 'b');
```

```
drawPlane(A(2,:), b(2), 'g');
hold off title('Ax=b in the row space', 'FontSize', 14);
figure(3); hold on;
plot(x(1), x(2), 'rx', 'MarkerSize', 16, 'LineWidth', 2)hold off;
```



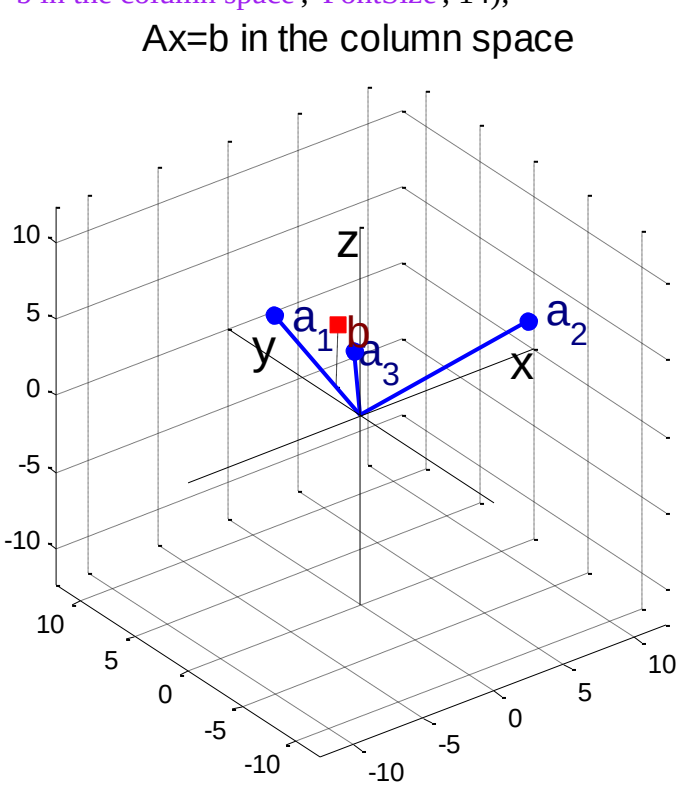

**Άσκηση 1:** Δεδομένου του πίνακα A = [2 4; 3 6] και του διανύσματος b = [4 7], οπτικοποιήστε το ΣΓΕ **Ax** = **b** στον χώρο στηλών και γραμμών.

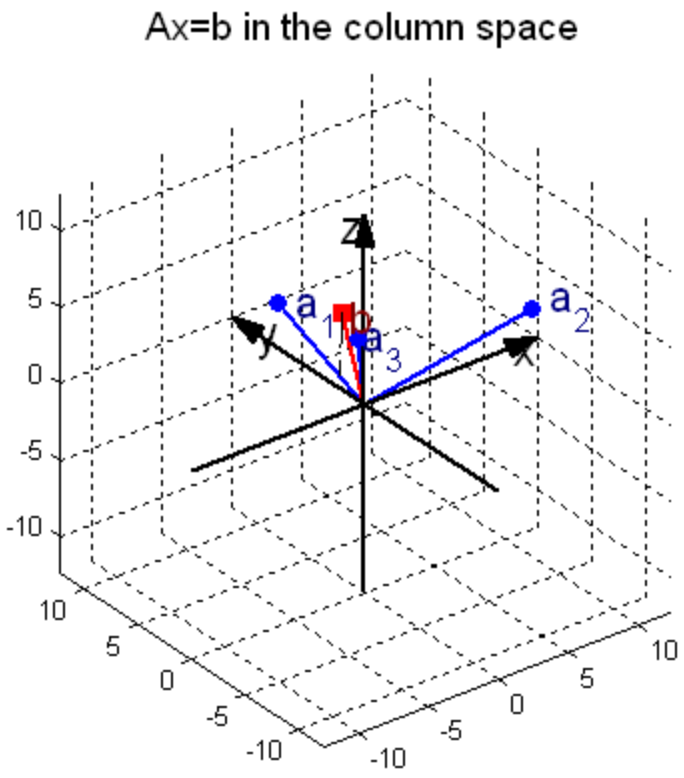
```
A = [ 2 4;
    3 6 ];
b = [4 7]';
```

**Άσκηση 2:** Η Alice αγοράζει τρία μήλα, δώδεκα μπανάνες και έναν ανανά για $2.36. Ο Bob αγοράζει δώδεκα μήλα και δύο ανανάδες για $5.26. Η Carol αγοράζει δύο μπανάνες και τρείς ανανάδες για $2.77. κατασκευάστε δύο γεωμετρικές αναπαραστάσεις για το «οικονομικό» πρόβλημα και βρείτε πόσο κοστίζει το κάθε φρούτο;

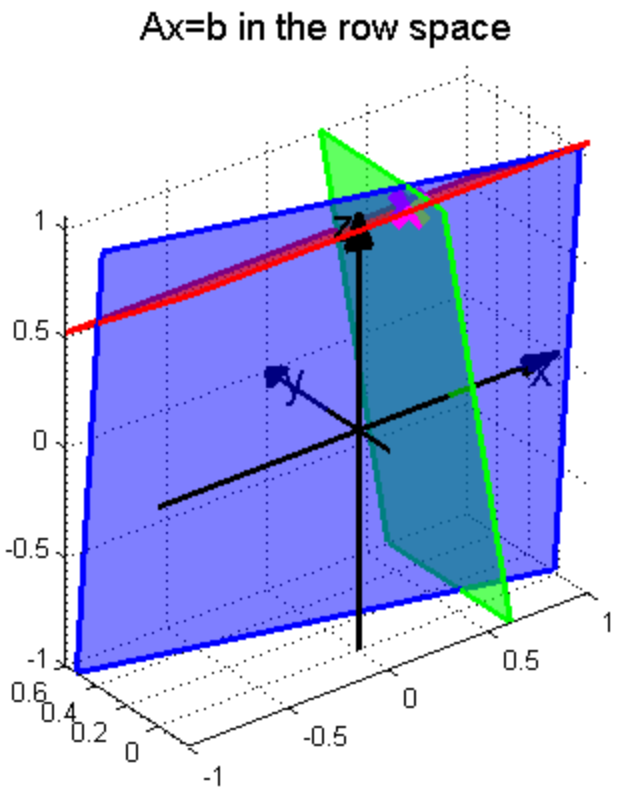
```
A = [3  12 1;
    12  0 2;
    0  2  3];
b = [2.36 5.26 2.77]';
x = A\b;
Pb = A*diag(x);
figure(1);clf;hold on;
drawVector(A, {'a_1','a_2','a_3'});
drawVector(b, 'rs', {'b'});
```

```
drawLine([b Pb(:,1)]);
hold off;title('Ax=b in the column space', 'FontSize', 14);
```






```
figure(3); clf; hold on;
drawPlane(A(1,:), b(1), 'b');
drawPlane(A(2,:), b(2), 'g');
drawPlane(A(3,:), b(3), 'r');
plot3(x(1),x(2),x(3), 'mx', 'LineWidth', 4, 'MarkerSize', 15)
hold off
title('Ax=b in the row space', 'FontSize', 14);
```




3. Ειδικοί τύποι ΣΓΕ: τριγωνικά συστήματα

Μερικοί τύποι ΣΓΕ μπορούν εύκολα να επιλυθούν με γεωμετρική αναπαράσταση.

**Άνω τριγωνικά συστήματα**
**Άσκηση 3:** Συγκρίνετε τον «χώρο στηλών» και τον «χώρο γραμμών» των παρακάτω 2-επί-2 ΣΓΕ: **Ux** = **b**, U = [4 1; 0 3.5] και b = [2 -3]'. Ποια αναπαράσταση είναι πιο κατάλληλη; βρείτε την λύση **x** χωρίς να υπολογίσετε τον αντίστροφο του **U** και προσθέστε την στο σχήμα.

```
A = [ 4 1   ;
      0 3.5 ];
b = [2 -3]';
figure(3); clf; hold on;
drawPlane(A(1,:), b(1), 'b');
drawPlane(A(2,:), b(2), 'g');
hold off title('Ax=b in the row space', 'FontSize', 14);
figure(3); hold on;
plot(x(1), x(2), 'rx', 'MarkerSize', 16, 'LineWidth', 2);
hold off;
```

```
A = [ 4 1   ;
      0 3.5 ];
b = [2 -3]';
```

```
x = A\b;
figure(2); clf; hold on;
drawVector(A, {'a_1', 'a_2'})
drawVector(b, 'rs', {'b'});
Pb = A*diag(x);
drawVector(Pb, 'g+', {'x_{11}\cdot a_1', 'x_{21}\cdot a_2'});
drawLine([b Pb(:,1)]); drawLine([b Pb(:,2)]);hold off;
title('Ax=b in the column space', 'FontSize', 14);
```

**Άσκηση 4:** Η διαδικασία με την οποία μπορείτε να βρείτε την λύση του συστήματος από την προηγούμενη Άσκηση καλείται «όπισθεν αντικατάσταση» και υλοποιείται στην συνάρτηση backSub.m. Κατεβάστε αυτή την συνάρτηση από το eclass. Δεδομένου του πίνακα U = [2 1 1 0; 0 1 1 1; 0 0 2 2; 0 0 0 2], γράψτε ένα script που να υπολογίζει τον αντίστροφο με την βοήθεια στης συνάρτησης backSub.

```
U = [ 2 1 1 0;
      0 1 1 1;
      0 0 2 2;
      0 0 0 2 ];
I=eye(4);
for j=1:4
   b=I(1:4,j);
   x(1:4,j)=backSub(U,b);
end
V=Ux;
display(x);
display(V);
```

**Κάτω τριγωνικά συστήματα**

Ομοίως, το κάτω-τριγωνικό σύστημα μπορεί να επιλυθεί με «μπροστά αντικατάσταση». Η μέθοδος αυτή υλοποιείται με την συνάρτηση forSub.m.

## 9. Γκαουσιανή απαλοιφή χωρίς οδήγηση (αντιμετάθεση γραμμών)

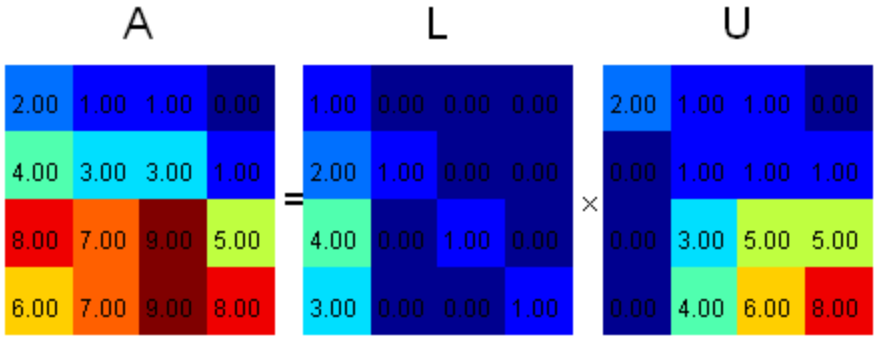
Η μέθοδος παραγοντοποίησης της πίνακα σε ένα γινόμενο άνω και κάτω τριγωνικών πινάκων καλείται Γκαουσιανή απαλοιφή. Για τετραγωνικούς πίνακες, υλοποιείται στην συνάρτηση slu.m και είναι διαθέσιμη στο eclass.

**Άσκηση 5:** Η συνάρτηση slu έχει μια προαιρετική παράμετρο «1» ή «0» η οποία, όταν είναι «1», της δείχνει όλα τα ενδιάμεσα βήματα.

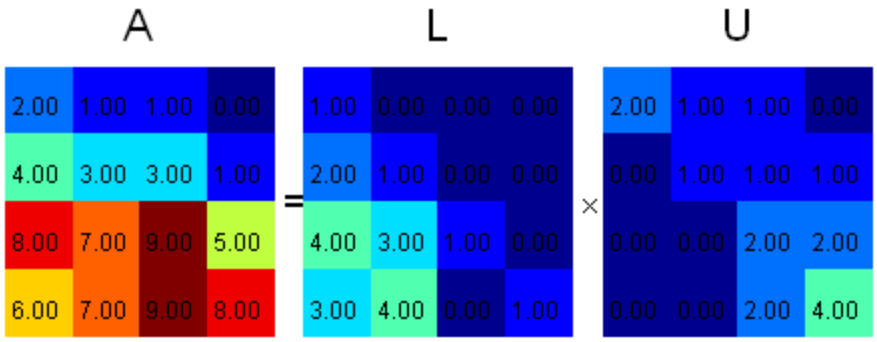
- Δοκιμάστε την με τον πίνακα **A** που ορίζεται παρακάτω.
- Τροποποιήστε την συνάρτηση για να δείχνει αντί το **A** = **LU** σε κάθε βήμα να δείχνει το **MA** = **U** **χωρίς** να υπολογίζει τον αντίστροφο του **L**, δηλαδή, χωρίς την χρήση της inv() ή της forSub(), ή οποιδήποτε άλλου πολλαπλασιασμού με πίνακα.

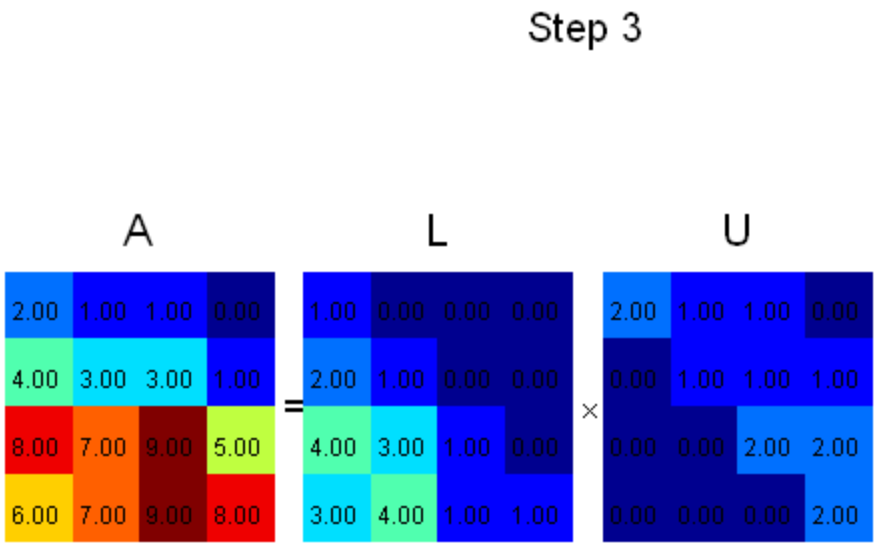
```
A = [ 2 1 1 0;
      4 3 3 1;
      8 7 9 5;
      6 7 9 8 ];
[L, U] = slu(A,1);
```

Step 1


A
L
U
2.00 1.00 1.00 0.00
4.00 3.00 3.00 1.00
8.00 7.00 9.00 5.00
6.00 7.00 9.00 8.00
=
1.00 0.00 0.00 0.00
2.00 1.00 0.00 0.00
4.00 0.00 1.00 0.00
3.00 0.00 0.00 1.00
×
2.00 1.00 1.00 0.00
0.00 1.00 1.00 1.00
0.00 3.00 5.00 5.00
0.00 4.00 6.00 8.00

Step 2


A
L
U
2.00 1.00 1.00 0.00
4.00 3.00 3.00 1.00
8.00 7.00 9.00 5.00
6.00 7.00 9.00 8.00
=
1.00 0.00 0.00 0.00
2.00 1.00 0.00 0.00
4.00 3.00 1.00 0.00
3.00 4.00 0.00 1.00
×
2.00 1.00 1.00 0.00
0.00 1.00 1.00 1.00
0.00 0.00 2.00 2.00
0.00 0.00 2.00 4.00




**Άσκηση 6:** Λύστε το ΣΓΕ **Ax** = **b** για τον πίνακα **A** που ορίζεται στην Άσκηση 5, και το διάνυσμα b = (2:2:8)' εφαρμόζοντας την Γκαουσιανή απαλοιφή πρώτα και την μπροστά και πίσω αντικατάσταση μετά.

## 10. Γκαουσιανή απαλοιφή με οδήγηση (αντιμετάθεση γραμμών ή στηλών)

**Άσκηση 7:** Στο MATLAB, η παραγοντοποίηση LU υλοποιείται με την συνάρτηση lu. Υπολογίστε την παραγοντοποίηση LU του πίνακα **A** που ορίζεται πάνω και συγκρίνετε τους πίνακες **L** και **U** τους οποίους πήρατε από την συνάρτηση slu. Διαφέρουν; Εάν ναι, γιατί; Πως μπορείτε να τροποποιήσετε τον **A** για να δίνουν τα ίδια αποτελέσματα οι συναρτήσεις slu() και lu();

## 11. Επίλυση γραμμικών συστημάτων με παραγοντοποίηση QR

Μέχρις εδώ, μάθαμε δύο μεθόδους για την επίλυση συστημάτων γραμμικών εξισώσεων, **Ax=b**:

i. Πολλαπλασιασμός με τον ανάστροφο: **x=A'b**, εάν ο **A** έχει **ορθογώνιες στήλες**.

ii. Γκαουσιανή απαλοιφή για γενικούς αντιστρέψιμους πίνακες, όπου ο **A** μετατρέπεται σε άνω τριγωνικό πίνακα **U**.

Η πρώτη μέθοδος είναι πολύ απλή, αλλά είναι εφαρμόσιμη για ειδικούς (ορθογώνιους) πίνακες μόνο. Η Γκαουσιανή απαλοιφή είναι γενική αλλά έχει δύο μεγάλα μειονεκτήματα:

- Μπορεί να είναι «ασταθής αριθμητικά» για ορισμένους πίνακες
- Η ανάλυση σφάλματος για τον LU-αλγόριθμο είναι αρκετά δύσκολη.

Παρόλα τα μειονεκτήματα, η Γκαουσιανή απαλοιφή χρησιμοποιείται εδώ και 200 χρόνια. Μόνο με την ανάπτυξη των ψηφιακών υπολογιστών, εμφανίστηκε ο τρίτος αλγόριθμος, ο οποίος συνδυάζει τα πλεονεκτήματα από τους δύο προγενέστερους: παραγοντοποιεί έναν γενικό πίνακα **A** σε ένα γινόμενο ενός ορθογώνιου πίνακα και ενός άνω-τριγωνικού πίνακα: **A=QR**.

**Άσκηση 1:**

Στο MATLAB, η παραγοντοποίηση QR ενός πίνακα **A** υπολογίζεται με την συνάρτηση [Q,R]=qr(A). Κατεβάστε την συνάρτηση solveLinearSystem.m από το link. Η συνάρτηση επιλύει το σύστημα **Ax=b** χρησιμοποιώντας μια από τις 2 μεθόδους που βασίζονται είτε στην παραγοντοποίηση LU- είτε στην παραγοντοποίηση QR. Αυτές οι μέθοδοι ορίζονται σε ένα μπλοκ «switch» που δεν έχει υλοποιηθεί ακόμη. Η Άσκηση σας είναι να:

- Υλοποιήσετε τις δύο μεθόδους.
- Κατασκευάσετε μερικά τυχαία γραμμικά συστήματα και να συγκρίνετε τις λύσεις που παράγονται από τις μεθόδους.
- Ελέγξετε τις μεθόδους σε ειδικούς πίνακες *60-επί-60*.

## 12. Τριγωνική ορθογωνοποίηση Gram-Schmidt

Υπάρχουν δύο (μαθηματικά ισοδύναμες, αλλά αριθμητικά διαφορετικές) υλοποιήσεις της διαδικασίας ορθογωνοποίησης του Gram-Schmidt: κλασική και τροποποιημένη. Υλοποιούνται στα αρχεία cgs.m και mgs.m , αντίστοιχα, και είναι διαθέσιμες στο κεφάλαιο 8.

**Άσκηση 2:**

Οι συναρτήσεις cgs() και mgs() παίρνουν μία προαιρετική παράμετρο "0", "1" ή "2" που ορίζει αν θα φαίνονται τα ενδιάμεσα βήματα της συνάρτησης κατά την εκτέλεση: 0 – για να μην φαίνονται, 1 ή 2 – για να φαίνονται, δηλαδή, κάθε βήμα μετασχηματισμού θα εμφανίζεται. Εκτελέστε τα παρακάτω πειράματα:

12. ***Γεωμετρική αναπαράσταση της κλασικής διαδικασίας Gram-Schmidt***: τρέξτε A=rand(2); [Q,R]=cgs(A,1); και A=rand(3); [Q,R]=cgs(A,1);. Σιγουρευτείτε πως καταλαβαίνετε τα σχήματα που παράγονται.

13. ***Διαφορά μεταξύ της κλασικής και της τροποποιημένης διαδικασίας Gram-Schmidt***: για να δείτε πως εμφανίζονται στους υπολογισμούς οι Q και R τρέξτε cgs() και mgs() σε μεγαλύτερους πίνακες: A = rand(5); [Q1,R1] = cgs(A,1); [Q2,R2] = mgs(A,1).
14. ***Τριγωνική ορθογωνοποίηση***: ο πίνακα Q εμφανίζεται σαν το αποτέλεσμα της σειράς του πολλαπλασιασμού του A με συγκεκριμένους άνω τριγωνικούς πίνακες: **A R1 R2 R3 ... Rn = Q**. θέστε την προαιρετική παράμετρο του mgs() "2" για να δείτε την επαναληπτική διαδικασία: A=rand(5); [Q,R]=mgs(A,2).

## 13. Ορθογώνια τριγωνοποίηση Householder

Στην διαδικασία Gram-Schmidt, ο βασικός στόχος είναι να ορθογωνοποιήσουμε τις στήλες του **A**. Ο **A** διαδοχικά μετασχηματίζεται σε έναν ορθογώνιο πίνακα και ο **R** εμφανίζεται σαν το γινόμενο της διαδικασίας της ορθογωνοποίησης.

Ο Householder επέλεξε έναν εναλλακτικό τρόπο: ο στόχος του είναι να μετατρέψει τον **A** σε έναν άνω τριγωνικό πίνακα εφαρμόζοντας ένα τελεστή στις στήλες του.

### Άσκηση 3: Αναπαράσταση του Householder σε 2 διαστάσεις

Κατεβάστε το demo-script HouseholderDemo2D.m από link. Τρέξτε και εξετάστε τον κώδικα μέχρι να είστε σίγουροι πως καταλαβαίνετε κάθε ένα από τα βήματα. Η παραγοντοποίηση QR του Housholder υλοποιείται στην συνάρτηση hr.m που είναι διαθέσιμη στο λογισμικό του παραρτήματος. Ομοίως με τις συναρτήσεις cgs() και mgs(),η hr() παίρνει ένα προαιρετικό όρισμα το οποίο επιτρέπει να εμφανίζεται κάθε βήμα της μετατροπής.

### Άσκηση 4: Αναπαράσταση του Householder σε 3 διαστάσεις

Για να δείτε πως η διαδικασία τριγωνοποίησης του Hoseholder μπορεί να γίνει γενική σε μεγαλύτερες διαστάσεις, τρέξτε την συνάρτηση hr() έτσι ώστε να φαίνονται τα ενδιάμεσα βήματα σε έναν πίνακα 3-επί-3 ή σε μεγαλύτερο πίνακα, π.χ., A = rand(3); R = hr(A,1);.

## 14. SVD παραγοντοποίηση πίνακα

Σύμφωνα με την SVD παραγοντοποίηση ενός *m-επί-n* πίνακα **A**, ο πίνακας μπορεί να αναπαρασταθεί σαν το άθροισμα πινάκων τάξης ένα:

$$A = U\Sigma V^T = \sum_{j=1}^{n} \sigma_j u_j v_j^T$$

$$f(x)$$

$$\in_{machine}$$

$$\frac{\left\| f(x) - f(x) \right\|}{\left\| f(x) \right\|} = O(\in_{machine})$$

$$\frac{\left\| x - x \right\|}{\left\| x \right\|} = O(\in_{machine})$$

όπου *u* και *v* είναι τα αριστερά και δεξιά ιδιάζοντα διανύσματα και *σ* είναι οι ιδιάζουσες τιμές. Η τάξη προσέγγισης *r* του πίνακα είναι το αποκομμένο *(truncated)* άθροισμα:

$$A = \sum_{j=1}^{n} \sigma_j u_j v_j^T$$

Σε αυτή την ενότητα, θα δούμε γιατί η SVD έχει αυτή την χρήσιμη ιδιότητα. Η ιδέα-κλειδί είναι η ***αλλαγή της βάσης***.

### 15. Αλλαγή βάσης στον χώρο γραμμών

Μία υπενθύμιση. Η SVD παραγοντοποιεί έναν πίνακα **A** σε ένα γινόμενο από τρεις πίνακες:

$$A = U\Sigma V^T$$

όπου **U** και **V** είναι ορθοκανονικοί, αριστερός και δεξιός, πίνακες ιδιοανυσμάτων, και ο διαγώνιος πίνακας **Σ** με ιδιοτιμές τιμές στην διαγώνιο.

### 16. Βελτίωση ενός προβλήματος και σταθερότητα ενός αλγορίθμου

Εδώ μας αφορούν δύο θεμελιώδη ζητήματα της αριθμητικής ανάλυσης: βελτίωση και σταθερότητα. Εάν το πρόβλημα είναι κακής κατάστασης (ill-conditioned) τότε η ακρίβεια του *κάθε* αλγόριθμου που μπορεί να χρησιμοποιήσουμε για την επίλυση των προβλημάτων θα είναι φτωχή. Εντούτοις, ακόμη και αν το πρόβλημα είναι καλής κατάστασης (well-conditioned), το αποτέλεσμα μπορεί να είναι ανακριβές εάν ένας ασταθής αριθμητικός αλγόριθμος χρησιμοποιηθεί.

## 16.1. Βελτίωση

Η ακρίβεια κάθε αλγόριθμου οριοθετείται από τον αριθμό βελτίωσης του προβλήματος *k(x)* που ικανοποιεί την παρακάτω σχέση όπου *f(x)* είναι το πρόβλημα και $\tilde{f}(x)$ η προσέγγιση του, και $\in_{machine}$ η ακρίβεια μηχανής:

$$\frac{\left\| f(x) - \tilde{f}(x) \right\|}{\left\| f(x) \right\|} = O(k(x) \in_{machine}) \quad (1)$$

Ένα πρόβλημα καλείται καλής κατάστασης (well-conditioned), εάν ο αριθμός βελτίωσης είναι μικρός, δηλαδή της τάξης του 10, 100 ή 1000, και κακής κατάστασης (ill-conditioned) εάν ο αριθμός βελτίωσης είναι μεγάλος: της τάξης του 10^6 - 10^10, και μεγαλύτερος.

Η εξίσωση (1) μπορεί να μεταφραστεί σαν ο αριθμός των ακριβεί στοιχείων (***the number of correct digits***) στην προσέγγιση του προβλήματος. Υποθέστε ότι ξέρουμε το *x* με την υψηλότερη δυνατή ακρίβεια στο δεκαδικό σύστημα, δηλαδή, το σχετικό σφάλμα στο *x* δεν είναι μεγαλύτερο από το μοναδιαίο σφάλμα στογγύλευσης (***unit roundoff*** ). Η ακρίβεια μηχανής, *eps* ορίζεται στο κεφάλαιο 14. Στο IEEE αριθμητικό πρότυπο η διπλή ακρίβεια σημαίνει ότι η τιμή του *x* είναι ακριβής στα πρώτα 15 ψηφία (μετά το δεκαδικό) και το 16ο ψηφίο έχει σφάλμα στρογγύλευσης σφάλμα της τάξης του *eps* = 2.22e-16.

Σημειώστε, ότι πρακτικά η τιμή του *x* είναι αποτέλεσμα υπολογισμών με ακρίβεια αρκετά μικρότερη από το *eps*. Εάν το *x* είναι γνωστό για μέχρι, π.χ., 6 σωστά δεκαδικά ψηφία (που είναι αρκετά ακριβής υπολογισμός), η λύση του προβλήματος *f(x)* με αριθμό βελτίωσης 1e+6 δεν θα εμφανίσει καθόλου σωστά ψηφία. Πιθανόν να μην έπρεπε να λύσουμε ένα τέτοιο πρόβλημα καθόλου.

**Άσκηση 1:**

Σαν παράδειγμα κακής κατάστασης προβλήματος μπορείτε να θεωρήσετε ένα σύστημα γραμμικών εξισώσεων της μορφής **Rx = b**, όπου ο **R** είναι ένας άνω τριγωνικός *m-επί-m* τυχαίος πίνακας. Μπορεί να παραχθεί μέσω του MATLAB ως R = triu(rand(m)). Σύμφωνα με την παραπάνω φόρμουλα, το *x* (δηλαδή, τα δεδομένα του προβλήματος) είναι ο **R** και το **b**, και ισχύει *f(x)*= *f*(**R,b**) = **x** .

- Αναπαραστήστε τον *αριθμό βελτίωσης* του πίνακα σαν συνάρτηση του μεγέθους του, *m*, για *m* = 1, 2, ..., 100. Χρησιμοποιείστε την συνάρτηση του MATLAB cond() για τον υπολογισμό των αριθμών βελτίωσης.
- Για *m*=50 ορίστε την *ground truth*: x = rand(m,1), b = R*x. τώρα, επιλύστε το σύστημα με όπισθεν αντικατάσταση: x1 = backSub(R,b), και υπολογίστε το σχετικό σφάλμα της λύσης σας. Ποιος είναι ο αριθμός βελτίωσης του προβλήματος;

Ποιο είναι το όριο του σχετικού σφάλματος της λύσης; Πως αυγκρίνεται με την λύση που υπολογίσατε;

- Συγκρίνετε τα στοιχεία της πραγματικής και της υπολογισμένης λύσης. Πόσα ψηφία χάνονται;
- Κατασκευάστε δύο πρόσθετα διανύσματα **b2** και **b3** διασπώντας το **b** με rand(m,1)*1e-6. Λύστε το αντίστοιχο πρόβλημα για **x2** και **x3** και σημειώστε τις σχετικές αλλαγές για **b2 - b** και **b3 - b** και τα σχετικά σφάλματα για **x2** και **x3.** Τι έχετε να πείτε για την σχετική αλλαγή **x2 - x3**;

## 16.2 Σταθερότητα και προς τα πίσω σταθερότητα

Ένας αλγόριθμος για ένα πρόβλημα *f(x)* καλείται σταθερός για κάθε *x* αν ισχύουν οι σχέσεις:

$$\frac{\left\| f(x) - f(x) \right\|}{\left\| f(x) \right\|} = O(\in_{machine})$$

και,

$$\frac{\left\| x - x \right\|}{\left\| x \right\|} = O(\in_{machine})$$

**Άσκηση 2: (μη)σταθερότητα της Γκαουσιανής απαλοιφής**
Θεωρείστε ένα πρόβλημα παραγοντοποίησης LU για τον παρακάτω πίνακα:

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

Αυτός ο πίνακας είναι καλής κατάστασης και είναι πλήρη τάξης. Παρόλα αυτά, η Γκαουσιανή απαλοιφή αποτυγχάνει στο πρώτο βήμα: το πρώτο οδηγό στοιχείο είναι A(1,1) = 0.

Αντικαταστήσετε το A(1,1) με έναν πολύ μικρό αριθμό 1e-16. Τώρα, η Γκαουσιανή απαλοιφή δεν αποτυγχάνει. Προσδιορίσετε την παραγοντοποίηση LU του πίνακα A ακριβώς με το χέρι και υπολογίστε την λύση με τον αλγόριθμο [L,U] = slu(). Είναι οι πίνακες **L** και **U** που υπολογίσατε ίδιοι με του ακριβείς πίνακες;

Υπολογίστε ξανά τον **A** από την παραγοντοποίηση: **A1 = LU**. Είναι ίδιος με τον αρχικό **A**;

Δοθέντος του δεξιού διανύσματος b=[1 0]', λύστε το σύστημα **LUx = b**. Είναι η λύση σωστή; Ποια είναι η σωστή λύση, δηλαδή η λύση του αρχικού προβλήματος;

Βασισμένη στην παρατήρηση σας, η Γκαουσιανή απαλοιφή είναι σταθερός αλγόριθμος; Προς τα πίσω σταθερός; Ή καθόλου σταθερός;

**Άσκηση 3: σταθερότητα της Γκαουσιανής απαλοιφής με άξονα**

Στην Άσκηση 2 της 4, ο παράγων αύξησης της παραγοντοποίησης LU ορίζεται. Είναι ένα μέτρο σταθερότητας της Γκαουσιανής απαλοιφής: εάν ο παράγοντας αύξησης είναι μεγάλος τότε η παραγοντοποίηση είναι ασταθής. Αυτή η άσκηση παρουσιάζει την χειρότερη περίπτωση αστάθειας: ο παράγοντας αύξησης είναι της τάξης του 2^m, όπου *m* είναι η διάσταση αυτού του ειδικού πίνακα. Στην πράξη, εντούτοις, ένας τόσο μεγάλος παράγοντας αύξησης δεν υπάρχει. (Γι'αυτό η Γκαουσιανή απαλοιφή είναι τόσο δημοφιλής). Ο στόχος αυτής της άσκησης είναι να παρουσιάσουμε αυτό το φαινόμενο εφαρμόζοντας πειράματα παραγοντοποίησης σε τυχαίους πίνακες.

- Για m = 10:100, παράγεται έναν τυχαίο πίνακα, υπολογίστε την παραγοντοποίηση LU και υπολογίστε τον παράγοντα αύξησης. Αναπαραστήστε τον παράγοντα της αύξησης σε semilog σχήμα σαν συνάρτηση του *m*.
- Εκτιμήστε το άνω όριο για τον παράγοντα αύξησης σαν συνάρτηση του *m*, δηλαδή, βρείτε την έκφραση στις παρενθέσεις που ικανοποιούν την εξίσωση:

$\rho = O(?)$.

**17. <u>Λογισμικό</u>**
([https://sites.google.com/a/uni-konstanz.de/na09/Home/software](https://sites.google.com/a/uni-konstanz.de/na09/Home/software))

**Numerical Methods**

| **Systems of linear equations: direct methods** | |
|---|---|
| 1. Backward substitution | backSub.m |
| 2. Forward substitution | forSub.m |
| 3. LU factorization without pivoting | slu.m |
| 4. LU factorization with pivoting | splu.m |
| 5. Linear system solver | solveLinearSystem.m |
| 6. Classical and modified Gram-Schmidt | cgs.m, mgs.m |
| 7. QR by Householder reflections | hr.m |
| 8. Least squares system solver | solveLSS.m |
| **Systems of linear equations: iterative methods** | |
| 9. Steepest descent | sd.m |
| 10. Conjugate gradient | cg.m |
| **Eigenvalue decomposition** | |

**Misc Utils**

| **Floating point number system** | |
|---|---|
| 1. Floating point number given mantissa and exponent | fpn.m |
| 2. Real to floating point number | real2fpn.m |
| 3. Rounding to | |

**drawLA Toolbox**

The drawLA Toolbox was created to facilitate visualization of some basic concepts of Linear Algebra. It is a collection of MATLAB functions for easy plotting of 2D/3D vectors, planes, lines and spheres, and... displaying matrix equations.

**Download**

| | |
|---|---|
| 11. Power iteration method | eigpowit.m |
| **Nonlinear methods** | |
| 12. Newton's method | nlnewton.m |
| 13. Conjugate gradient minimizer | congrad.m |