Εισαγωγή στο Octave

Α. Δροσόπουλος

Περιεχόμενα

1 Εισαγωγή στο Octave **1**

1.1 Εισαγωγικά . . . 1

1.1.1 Εγκατάσταση Octave 4.2.1 . . . 1

1.2 Το Octave σαν «κομπιουτεράκι» . . . 5

1.3 Το Octave σαν περιβάλλον εργασίας . . . 8

1.4 Διανύσματα και πίνακες . . . 9

1.5 Ρίζες πολυωνύμων . . . 15

1.6 Είσοδος/έξοδος δεδομένων από/σε εξωτερικά αρχεία . . . 15

1.7 Είσοδος δεδομένων διαδραστικά . . . 17

1.8 Γραφικές παραστάσεις . . . 18

1.8.1 Απλή γραφική . . . 18

1.8.2 Γραφικές 3 διαστάσεων . . . 19

1.8.3 Γραφική θεωρίας και πειραματικών δεδομένων . . . 20

1.9 Ο Editor και αρχεία script . . . 23

1.10 Functions . . . 23

1.11 Anonymous functions . . . 26

1.12 Εντολές επιλογής και επανάληψης . . . 27

1.12.1 if, else, elseif . . . 27

1.12.2 while και do until . . . 29

1.12.3 for . . . 29

1.13 Λογική πρόσβαση σε στοιχεία . . . 29

1.14 Μέθοδος Newton . . . 31

1.15 Μέθοδος Διχοτόμησης (bisection) . . . 34

1.16 Μέθοδος ελαχίστων τετραγώνων . . . 37

1.17 Μέθοδοι αριθμητικής ολοκλήρωσης . . . 39

1.17.1 Κανόνας Τραπεζίου . . . 39

1.17.2 Κανόνας Simpson . . . 40

1.18 Διαφορικές εξισώσεις . . . 41

1.19 Μελέτη πληθυσμού χώρας . . . 47

1.19.1 Πληθυσμός ΗΠΑ . . . 47

1.19.2 Πληθυσμός Ελλάδος . . . 50

1.20 Παράδειγμα πιο σύνθετου προγράμματος . . . 52

1.21 Ασκήσεις . . . 56

ΑΣΚΗΣΗ 1

Εισαγωγή στο Octave

Εισαγωγή στο πακέτο-εφαρμογή Octave, βασική χρήση απλών εντολών, πράξεις με πραγματικούς και μιγαδικούς αριθμούς, φάκελος εργασίας, αποθήκευση συνεδρίας, διανύσματα και πίνακες, είσοδος και έξοδος δεδομένων από και σε εξωτερικά αρχεία και απλές γραφικές παραστάσεις. Χρήση editor. Προγράμματα script, functions και παραδείγματα εφαρμογών όπου φαίνεται η χρήση των βασικών δομών προγραμματισμού και πινακοποίησης.

1.1 Εισαγωγικά

Τη δεκαετία του 80 κυκλοφόρησε ελεύθερα (public domain) στο Διαδίκτυο ένα πακέτο από FORTRAN υπορουτίνες για εύκολες και γρήγορες πράξεις με πίνακες σε υπολογιστή. Διάφορες εταιρίες βελτίωσαν και αύξησαν τις δυνατότητες του αρχικού πακέτου με την προσθήκη γραφικών και βιβλιοθηκών εξειδικευμένων συναρτήσεων χρήσιμων σε πολλούς τομείς εφηρμοσμένων επιστημών. Η πιο επιτυχημένη εταιρία ήταν και είναι η Mathworks (`http://www.mathworks.com`) και το προϊόν της το MATLAB χρησιμοποιείται σήμερα διεθνώς σε πολλά πανεπιστήμια, ερευνητικά κέντρα και εταιρίες που ασχολούνται με την έρευνα και την τεχνολογία.

Την ίδια δεκαετία ξεκίνησε και το ρεύμα του Ανοικτού Λογισμικού (Open Source) που με τη βοήθεια του Διαδικτύου έδωσε την ευκαιρία σε πολλούς νέους ερευνητές να συνεργαστούν εθελοντικά και να δημιουργήσουν πολλές χρήσιμες εφαρμογές (π.χ. GCC C και C++ compilers και εργαλεία, λειτουργικό Linux, κλπ.).

Το MATLAB είναι αναμφισβήτητα ένα από τα πιο χρήσιμα εργαλεία που μαθαίνει ένας μηχανικός/επιστήμονας και σε όλα σχεδόν τα πανεπιστήμια των τεχνολογικά προηγμένων χωρών του εξωτερικού επιβάλλεται η διδασκαλία του από την αρχή των σπουδών του. Εκτός όμως από το εμπορικό πακέτο MATLAB που η πλήρης έκδοση έχει αρκετά υψηλό κόστος, ερευνητές που χρησιμοποιούν εργαλεία ανοικτού λογισμικού δημιούργησαν άλλα πακέτα/εφαρμογές με παρόμοιες δυνατότητες, που έχουν το πλεονέκτημα να είναι νομίμως δωρεάν. Ένα από αυτά, το Octave (`https://www.gnu.org/software/octave`), από τα Πανεπιστήμια Wisconsin-Madison και Texas της Αμερικής, έχει σχεδιαστεί με στόχο την όσο το δυνατόν πλήρη συμβατότητα με το MATLAB. Διαθέτει πολλές από τις βασικές δυνατότητες του MATLAB και βρίσκεται σε συνεχή ανάπτυξη για περαιτέρω βελτίωση. Τα προγράμματα που θα ετοιμάζουμε παρακάτω μπορούν να τρέξουν σχεδόν αυτούσια και στα δυο περιβάλλοντα.

Ένα συνηθισμένο σενάριο σε πανεπιστήμια του εξωτερικού είναι το πανεπιστήμιο να διαθέτει άδεια χρήσης του MATLAB στο χώρο του πανεπιστημίου και οι σπουδαστές να χρησιμοποιούν το Octave στο σπίτι τους για τις εργασίες τους.

Επισημαίνεται ότι η Mathworks προσφέρει και στην Ελλάδα φοιτητική έκδοση MATLAB με κόστος γύρω στα 90€.

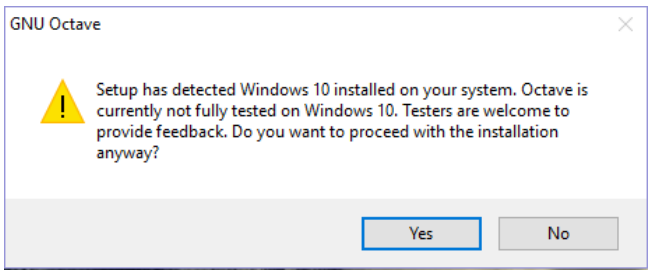
Η τελευταία έκδοση Octave (Μάρτιος 2017) είναι η 4.2.1. Ξεκινούμε με οδηγίες εγκατάστασης της 4.2.1 σε περιβάλλον Windows.

1.1.1 Εγκατάσταση Octave 4.2.1

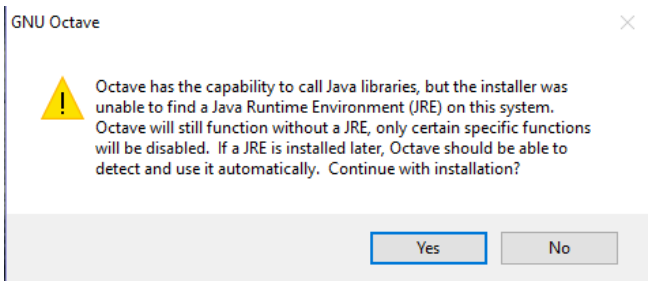
Το επίσημο site του Octave είναι `https://www.gnu.org/software/octave`. Η νέα έκδοση 4.2.1 προσφέρεται σε 32bit (`octave-4.2.1-w32-installer.exe`) και 64bit (`octave-4.2.1-w64-installer.exe`) για Windows από το `https://ftp.gnu.org/gnu/octave/windows/`. Κατεβάζετε αυτή που είναι συμβατή με το λειτουργικό σας. Προσφέρει τη δυνατότητα εργασίας σε δυο περιβάλλοντα, το γραφικό περιβάλλον, GUI (Graphical User Interface) και το περιβάλλον κονσόλας, CLI (Command Line Interface).

Θα χρειαστεί να κατεβάσετε επίσης το ακόλουθο λογισμικό από το διαδίκτυο:

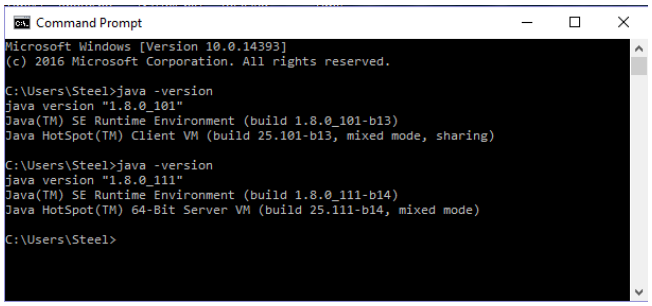
Java JRE: Το κατάλληλο Java Runtime Environment (JRE) για το λειτουργικό σας και την έκδοση Octave που κατεβάσατε, από το `http://java.com`, 32bit ή 64bit. Εδώ χρειάζεται κάποια προσοχή. Μπορεί να έχετε 64bit σύστημα και 32bit browser (περιηγητή). Το αυτόματο κατέβασμα τότε θα σας κατεβάσει 32bit java. Εάν εσείς προσπαθείτε



Σχήμα 1.1: Μήνυμα 1ο και μόνο για λειτουργικό Windows 10. Προειδοποιεί για μη πλήρη υποστήριξη στο συγκεκριμένο Windows. Το δοκίμασα σε Windows 8.1, Pro, 32bit και Windows 10, 64bit και φαίνεται να μην υπάρχει πρόβλημα. Κανένα πρόβλημα επίσης σε Windows 7. Επιλέγετε Yes και συνεχίζετε.



Σχήμα 1.2: Αν δεν έχετε Java στον υπολογιστή σας θα δείτε το παραπάνω μήνυμα που συστήνει πρώτα την εγκατάσταση Java και μετά Octave. Επίσης, αν έχετε Java 32bit και κάνετε εγκατάσταση Octave 64bit θα δείτε το ίδιο μήνυμα. Επιλέγετε No και σταματάτε. Επανέρχεστε όταν εγκαταστήσετε την κατάλληλη Java.



Σχήμα 1.3: Με την παραπάνω εντολή βλέπετε αν έχετε 32 ή 64bit java στην κονσόλα εντολών των Windows. Η πρώτη χρήση δείχνει 32bit και μετά την εγκατάσταση της 64bit java, δείχνει ότι έχουμε 64bit.

να εγκαταστήσετε Octave 64bit θα χρειαστεί να πάτε στο `https://java.com/en/download/manual.jsp` και να κατεβάσετε την java Windows Offline (64-bit) και να την εγκαταστήσετε ανεξάρτητα από τον browser.

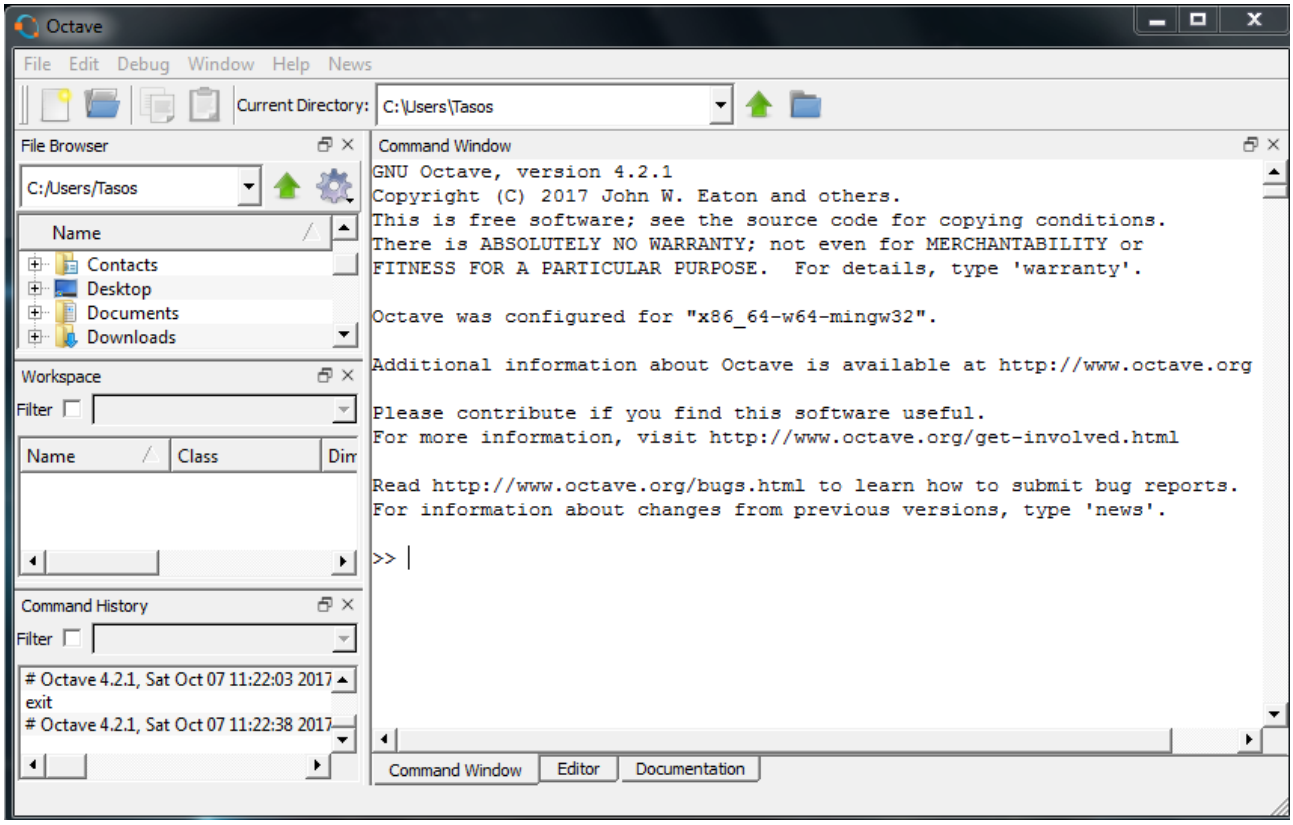
Inkscape: Το πρόγραμμα δημιουργίας και επεξεργασίας γραφικών Inkscape από `https://inkscape.org`. Πηγαίνετε στο DOWNLOAD -> Windows installer και κατεβάστε την 32bit ή 64bit έκδοση που αντιστοιχεί στο δικό σας σύστημα.

Notepad++: Προαιρετικά, μπορείτε να κατεβάσετε τον editor Notepad++, από το `https://notepad-plus-plus.org`, έναν βελτιωμένο editor, που αντικαθιστά τους Notepad και Wordpad editors των Windows. Το Notepad++ εξειδικεύεται για τους προγραμματιστές που γράφουν κώδικα σε πληθώρα γλωσσών και θα σας φανεί χρήσιμο και αλλού. Αν και η έκδοση Octave συμπεριλαμβάνει δικό της editor στο περιβάλλον GUI, στο περιβάλλον CLI θα χρειαστείτε τον Notepad++. Αν και εδώ υπάρχει 32bit και 64bit έκδοση, προτιμήστε την 32bit που προς το παρών είναι πιο πλήρης.

7zip: Προαιρετικά, την εφαρμογή συμπίεσης αρχείων 7zip από το `http://www.7-zip.org`, παρόμοια της Winzip ή Winrar. Δεν την χρειάζεστε αν έχετε κάποια από τις δυο τελευταίες. Η εφαρμογή αυτή σας είναι απαραίτητη όταν ανεβάζετε εργασίες στο eclass για το εργαστήριο.

Εάν αρχίσετε με εγκατάσταση του octave-4.2.1 σε Windows 10 θα δείτε τα μηνύματα των Σχ. 1.1-1.2. Αν δεν έχετε Java αρχίζετε με την εγκατάσταση της Java ακολουθώντας πιστά τις οδηγίες. Αυτό σημαίνει ότι κατεβάζετε πρώτα τον Java installer και τον αποθηκεύετε στο σύστημά σας. Κλείνετε τον browser σας και αρχίζετε την εγκατάσταση. Όταν τελειώσει

θα σας ανοίξει ένα παράθυρο με μήνυμα ότι η εγκατάσταση έγινε σωστά. Συνεχίζετε τότε με εγκατάσταση του Octave-4.2.1. Η εγκατάσταση των Inkscape και Notepad++ είναι και αυτή απλή. Απλώς ακολουθείτε τα βήματα που εμφανίζονται.



Σχήμα 1.4: Octave GUI περιβάλλον για έκδοση 4.2.1. Διακρίνουμε το Command Window (κονσόλα εντολών) τον Editor και την τεκμηρίωση (Documentation) με οδηγίες και online βοήθεια. Διακρίνουμε επίσης τον τρέχοντα φάκελο εργασίας.

Κάθε φορά που αρχίζουμε μια συνεδρία με το Octave το πρόγραμμα «βλέπει» έναν συγκεκριμένο φάκελο (folder) όπου μπορεί να διαβάσει και να αποθηκεύσει αρχεία δεδομένων ή γραφικών παραστάσεων. Ο φάκελος αυτός ονομάζεται *φάκελος εργασίας*. Η αρχική θέση του φακέλου εργασίας εξαρτάται από την έκδοση του Octave και το λειτουργικό σύστημα. Π.χ. σε περιβάλλον Windows για το Octave 4.2.1 βλέπουμε ότι είμαστε στο `C:/Users/Tasos` αν Tasos είναι το όνομα χρήστη (περιοχή File Browser επάνω αριστερά). Αν χρησιμοποιούμε Octave CLI πληκτρολογούμε `pwd` (print working directory) και το αποτέλεσμα εμφανίζεται στην κονσόλα. Χρήσιμο είναι να δημιουργήσετε έναν ξεχωριστό φάκελο, π.χ. `C:\work` και να έχετε το octave να πηγαίνει κατευθείαν εκεί.

Δημιουργείστε πρώτα τον φάκελο `C:\work` και μετά στο Edit -> Preferences -> General δηλώστε τον σαν αρχικό φάκελο εργασίας στον οποίο θα ανοίγει πάντα το πρόγραμμά σας (βλ. Σχ. 1.5). Στο Edit -> Preferences -> Editor σετάρετε επίσης text encoding UTF-8 (βλ. Σχ. 1.6) που θα σας επιτρέψει αργότερα να έχετε και ελληνικούς χαρακτήρες στις γραφικές και σχόλιά σας.

Το Octave έχει τρία διαθέσιμα graphics toolkits, όπως φαίνεται από την εντολή `available_graphics_toolkits`. Παλαιότερα το gnuplot ήταν πιο λειτουργικό από τα άλλα δυο. Τώρα φαίνεται ότι έχει γίνει καλή δουλειά στο qt (που είναι από μόνο του προεπιλεγμένο) και ανταποκρίνεται καλύτερα από το gnuplot στην παρούσα έκδοση. Εάν χρειαστεί να το αλλάξετε για την τρέχουσα συνεδρία χρησιμοποιείτε την παρακάτω εντολή:

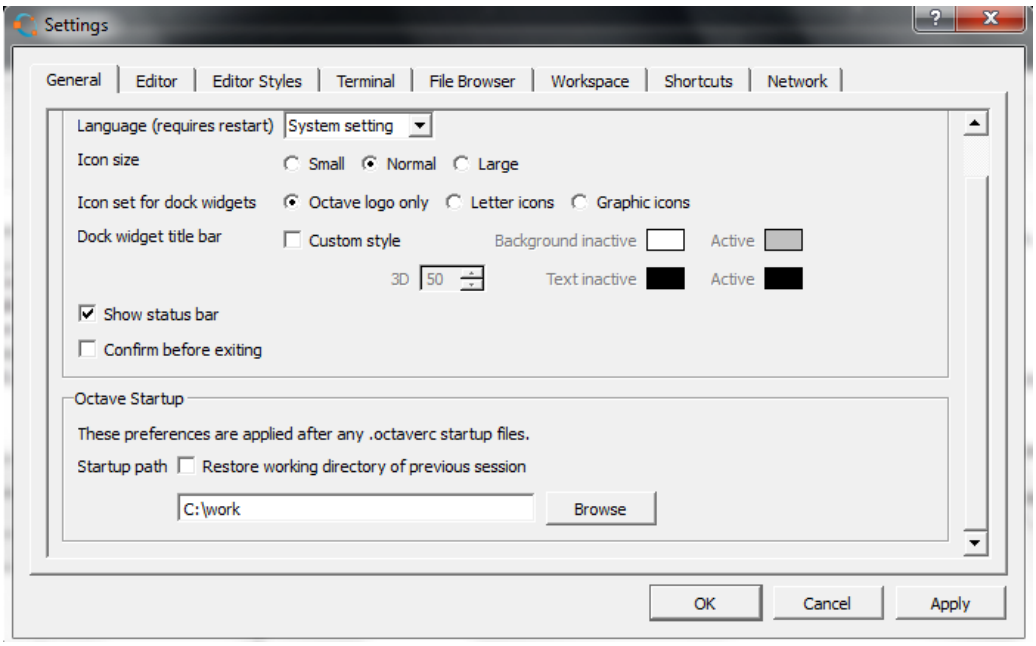
```
graphics_toolkit('gnuplot')
```

και ξαναγυρίζετε στο qt με

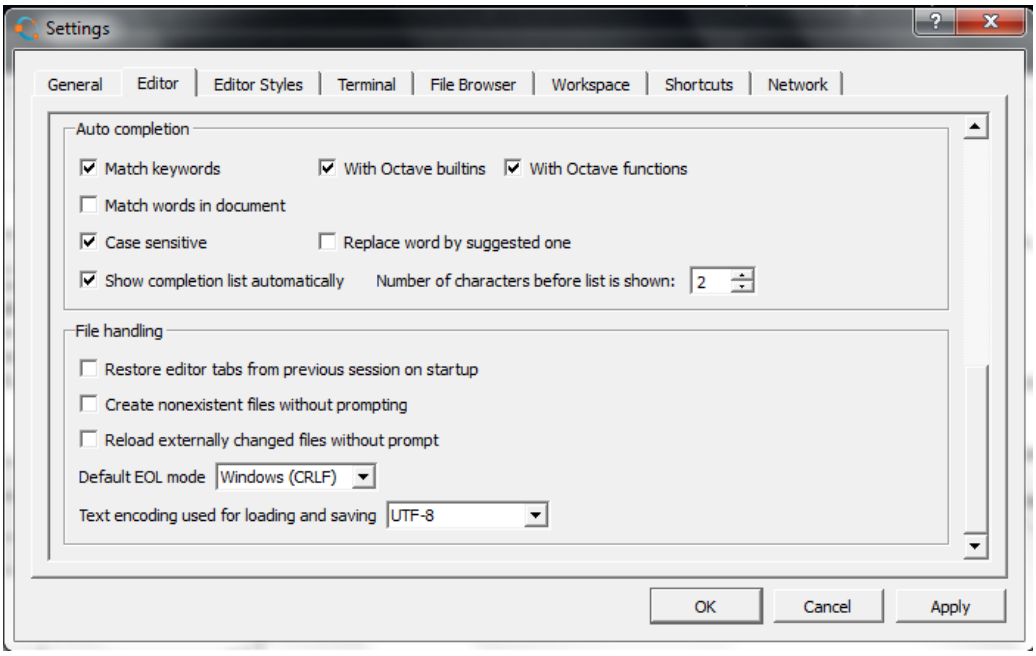
```
graphics_toolkit('qt')
```

Εάν θέλετε μόνιμη αλλαγή πηγαίνετε στο φάκελο `C:\Octave\Octave-4.2.1\share\octave\site\m\startup` και με τον editor ανοίγετε το αρχείο octaverc όπου προσθέτετε στο τέλος την παρακάτω γραμμή:

```
graphics_toolkit('gnuplot')
```



Σχήμα 1.5: Edit -> Preferences, -> General, σετάρισμα αρχικού φακέλου εργασίας.



Σχήμα 1.6: Edit -> Preferences, -> Editor, σετάρισμα UTF-8 text encoding.

Καλή υποστήριξη Ελληνικών σε γραφικές υπάρχει με εξωτερικά αρχεία τύπου svg. Στόχος σε εμάς τους Έλληνες χρήστες είναι να μπορούμε να εισάγουμε γραφικές με ελληνικούς χαρακτήρες σε εφαρμογές επεξεργασίας κειμένου (π.χ. Word, LibreOffice) για οτιδήποτε εργασίες ή αναφορές θέλουμε. Ελληνικούς χαρακτήρες ΔΕΝ βλέπουμε στην κονσόλα. Βλέπουμε όμως στον editor αν τον ρυθμίσουμε όπως παραπάνω.

Ο λόγος εγκατάστασης του inkscape είναι πλέον προφανής. Μας επιτρέπει να ανοίγουμε και να βλέπουμε svg αρχεία. Μας επιτρέπει να κάνουμε edit σε αυτά. Και τέλος μπορούμε να τα μετατρέψουμε σε άλλου τύπου αρχεία, π.χ. png, pdf ή emf μέσα από το ίδιο το πρόγραμμα. Για εισαγωγή σε Word συνιστάται η μορφή emf ή png. Το εναλλακτικό LibreOffice (νομίμως δωρεάν σαν ανοικτό λογισμικό) μπορεί να χειριστεί κατευθείαν svg αρχεία.

1.2 Το Octave σαν «κομπιουτεράκι»

Η πρώτη και ίσως πιο απλή χρήση του Octave είναι να το χρησιμοποιήσει κανείς σαν «κομπιουτεράκι». Όλες οι γνωστές αλγεβρικές πράξεις υποστηρίζονται όπως μπορούμε να δούμε όταν εισάγουμε τις πράξεις που θέλουμε να εκτελέσουμε μετά από το prompt, `>>` , του προγράμματος.

$+$	πρόσθεση
$-$	αφαίρεση
*	πολλαπλασιασμός
^	ύψωση σε δύναμη
/	δεξιά διαίρεση όπου $5/3 = 5 \div 3$
\	αριστερή διαίρεση όπου $5\backslash 3 = 3 \div 5$

Το πρόγραμμα εκτελεί αμέσως κάθε εντολή/πράξη και δίνει το αποτέλεσμα στη μεταβλητή ans

```
>> 5+3
ans =  8
>> 89*5
ans =  445
>> 56-8+9-45.5
ans =  11.500
>> 5/3
ans =  1.6667
>> 5\3
ans =  0.60000
>> 3/5
ans =  0.60000
```

Αν θέλουμε να συνεχίσουμε τους υπολογισμούς μας με το αποτέλεσμα που φαίνεται στην ans, την χρησιμοποιούμε σαν μεταβλητή, η οποία όμως παίρνει μετά τις πράξεις καινούργιο αποτέλεσμα. Μπορούμε επίσης να χρησιμοποιήσουμε μεταβλητές όπως σε γλώσσα Pascal ή C, π.χ. ο νόμος του Ohm σε κάποιο κύκλωμα.

```
>> U=220
U =  220
>> I=2
I =  2
>> R=U/I
R =  110
>> R=R+2
R =  112
>> 5+3
ans =  8
>> R=R+ans
R =  120
```

Εάν κάνετε κάποιο λάθος στη πληκτρολόγηση μπορείτε να πατήσετε το Enter μέχρι να εμφανιστεί το prompt (`>`) και να ξαναπληκτρολογήσετε σωστά τα δεδομένα σας. Μπορείτε επίσης να χρησιμοποιήσετε τα «βελάκια», να επιστρέψετε στο σημείο του λάθους, και να το διορθώσετε. Εάν επαναλαμβάνετε εντολές με ίσως μικρές διαφορές, βολεύει η χρήση του ↑ που φέρνει προηγούμενες εντολές στο prompt (`>>`).

Το Octave χρησιμοποιεί υψηλή ακρίβεια (double precision) για την εκτέλεση αριθμητικών πράξεων, αλλά δείχνει μικρότερο αριθμό δεκαδικών ψηφίων στο αποτέλεσμα (5 σημαντικά ψηφία). Η εντολή format μπορεί να αλλάξει την εμφάνιση του αριθμού των ψηφίων παράστασης σε 15 σημαντικά ψηφία. Π.χ.

```
>> 5/3
ans =  1.6667
>> format('long')
>> 5/3
ans =  1.66666666666667
>> format('short')
>> 5/3
ans =  1.6667
```

Για αριθμούς που είναι δυνάμεις του 10, χρησιμοποιείται η εκθετική μορφή με το σύμβολο e (exponent - εκθέτης), π.χ. $3.245 \times 10^3 =$ `3.245e3`, $-6.736 \times 10^{-6} =$ `-6.736e-6`.

Πίνακας 1.1: Μερικές από τις ενσωματωμένες συναρτήσεις που διαθέτει το Octave

`abs(x)`	απόλυτος τιμή ή μέτρο μιγαδικού αριθμού x
`acos(x)`	τόξο συνημιτόνου x σε rad
`asin(x)`	τόξο ημιτόνου x σε rad
`atan(x)`	τόξο εφαπτομένης x σε rad
`atan2(y,x)`	τόξο εφαπτομένης $\tan^{-1}(y/x)$ σε rad
`conj(x)`	συζυγής μιγαδικός του x
`cos(x)`	συνημίτονο του x όπου x σε rad
`exp(x)`	εκθετικό του x, e^x
`imag(x)`	φανταστικό μέρος του μιγαδικού αριθμού x
`log(x)`	φυσικός λογάριθμος (βάση e) του αριθμού x
`log10(x)`	δεκαδικός λογάριθμος (βάση 10) του αριθμού x
`real(x)`	πραγματικό μέρος του μιγαδικού αριθμού x
`sin(x)`	ημίτονο του x όπου x σε rad
`sqrt(x)`	τετραγωνική ρίζα του x
`tan(x)`	εφαπτομένη του x όπου x σε rad

Υπάρχει διαθέσιμη μια πλούσια συλλογή από ενσωματωμένες συναρτήσεις, πολύ περισσότερες από ένα «κομπιουτεράκι». Στον πίνακα 1.1 φαίνονται μερικές από αυτές.

Η πλήρης συλλογή συναρτήσεων του Octave φαίνεται αν πάτε στο Documentation και ανοίξετε το Function Index. Αν θέλουμε βοήθεια για κάποια συγκεκριμένη εντολή, π.χ. την εντολή `load` μπορούμε επίσης να πληκτρολογήσουμε απλώς `help load` στην κονσόλα. Ανοίγει τότε ο ενσωματωμένος info-reader στην κονσόλα με σύντομες πληροφορίες βοήθειας. Αν η βοήθεια συμπεριλαμβάνει πολλές σελίδες πηγαίνουμε στην επόμενη πληκτρολογώντας το spacebar. Βγαίνουμε από τον info-reader πληκτρολογώντας `q`.

Έστω ότι θέλουμε να βρούμε τις λύσεις της δευτεροβάθμιας $3x^2 + 2x - 6 = 0$. Οι λύσεις της γενικής δευτεροβάθμιας $ax^2 + bx + c = 0$ δίδονται από την σχέση:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Δοκιμάζοντας απλοϊκά:

```
>> a=3; b=2; c=-6;
>> x1 = -b + sqrt(b^2-4*a*c) / 2*a
x1 =  11.077
>> x2 = -b - sqrt(b^2-4*a*c) / 2*a
x2 = -15.077
```

παίρνουμε δυο λύσεις όπου με έλεγχο διαπιστώνουμε ότι είναι λάθος. Το λάθος οφείλεται στην προτεραιότητα των πράξεων. Στα παραπάνω θα υπολογιστεί πρώτα η ρίζα. Το αποτέλεσμα θα προστεθεί ή αφαιρεθεί στο $-b$. Το επόμενο αποτέλεσμα θα διαιρεθεί με το 2 και ότι προκύψει θα πολλαπλασιαστεί με το a. Το σωστό αποτέλεσμα βγαίνει από:

```
>> x1 = (-b + sqrt(b^2-4*a*c)) / (2*a)
x1 =  1.1196
>> x1 = (-b - sqrt(b^2-4*a*c)) / (2*a)
x1 = -1.7863
```

όπου χρησιμοποιήσαμε κατάλληλα παρενθέσεις για να γίνουν οι πράξεις με τη σωστή σειρά. Η προτεραιότητα των πράξεων φαίνεται στον πίνακα 1.2.

Αν κάπου έχετε αμφιβολία χρησιμοποιείστε επιπλέον παρενθέσεις.

Ακολουθούν παραδείγματα με μιγαδικούς αριθμούς και τριγωνομετρικές συναρτήσεις. Προσέξτε στις τελευταίες ότι οι γωνίες είναι σε ακτίνια (rad). Η μετατροπή από rad σε μοίρες γίνεται πολλαπλασιάζοντας με $180/\pi$ και από μοίρες σε rad πολλαπλασιάζοντας με $\pi/180$.

```
>> sin(60*pi/180)
ans =  0.86603
>> asin(ans)*180/pi
ans =  60.000
```

Πίνακας 1.2: Προτεραιότητα των πράξεων

`()`	Πρώτα υπολογίζονται οι πράξεις μέσα σε παρενθέσεις
`^`	Ακολουθεί η ύψωση σε δύναμη
`*/`	Πολλαπλασιασμός και διαίρεση έχουν ίδια προτεραιότητα και εκτελούνται από αριστερά προς τα δεξιά
`+-`	Πρόσθεση και αφαίρεση έχουν ίδια προτεραιότητα και εκτελούνται από αριστερά προς τα δεξιά

```
>> z=5+j*3
z =  5 + 3i
>> abs(z)
ans =  5.8310
>> real(z)
ans =  5
>> imag(z)
ans =  3
>> atan2(imag(z),real(z))*180/pi
ans =  30.964
>> [abs(z) angle(z)*180/pi]
ans =
    5.8310   30.9638
```

Η τελευταία εντολή μας μετατρέπει τον μιγαδικό αριθμό από καρτεσιανή σε πολική μορφή με τη συνάρτηση angle. Το αντίστροφο, όπου μετατρέπουμε έναν μιγαδικό από πολική μορφή (φάσορα) σε καρτεσιανή, π.χ. τον $5\ \underline{/45^\circ}$, το επιτυγχάνουμε ως εξής:

```
>> 5*exp(j*45*pi/180)
ans =  3.5355 + 3.5355i
```

Υπενθυμίζεται ότι ένας μιγαδικός αριθμός $z = a + jb = r\ \underline{/\theta} = r\ e^{j\theta}$ όπου $r = \sqrt{a^2 + b^2}$ και θ το όρισμα σε rad. Επομένως $5\ \underline{/45^\circ} = 5\ e^{j45\pi/180}$.

Μια μεταβλητή στο Octave όπως και σε άλλες γλώσσες προγραμματισμού χρησιμοποιείται σαν ενα σύμβολο για μια θέση μνήμης που περιέχει κάποια τιμή. Στα παραπάνω είδαμε ότι μπορούμε να δώσουμε δικά μας ονόματα σε μεταβλητές ή, αν δεν το κάνουμε αυτό, το Octave χρησιμοποιεί μια δική του μεταβλητή, την ans (answer – απάντηση) σαν στιγμιαία θέση μνήμης για το πιο πρόσφατο αποτέλεσμα (την μεταβλητή αυτή μπορούμε αν θέλουμε να την χρησιμοποιήσουμε και εμείς στην αμέσως επόμενη εντολή).

Στις αλγεβρικές μας πράξεις μπορούμε να χρησιμοποιούμε παρενθέσεις με την ίδια λογική που έχουμε μάθει στα μαθηματικά. Αν έχουμε πολλές αλγεβρικές πράξεις στη σειρά, π.χ. `5+3*2/6-4 = 5+1-4=2`, οι πράξεις εκτελούνται από αριστερά προς τα δεξιά ανάλογα με την προτεραιότητά τους. Η ύψωση σε δύναμη έρχεται πρώτη, πολ/σμός και διαίρεση κατόπιν με ίδια προτεραιότητα, και τέλος πρόσθεση και αφαίρεση, με ίδια προτεραιότητα. Μόνο οι παρενθέσεις μπορούν να αλλάξουν την προτεραιότητα.

```
>> 8+3*5
ans =  23
>> (8+3)*5
ans =  55
>> 4^2-12-8/4*2
ans = 0
>> 4^2-12-8/(4*2)
ans =  3
>> 3*4^2+5
ans =  53
>> (3*4)^2+5
ans =  149
>> 27^1/3+32^0.2
ans =  11
>> 1.2*sin(pi^2+log10(5))
ans = -1.0923
>> new=3+ans/2
new =  2.4539
```

Υπενθυμίζεται επίσης ότι το $=$ στον προγραμματισμό έχει διαφορετική σημασία από ότι στα μαθηματικά. Η έκφραση `x=2` σημαίνει ότι η θέση μνήμης που έχει όνομα το σύμβολο μεταβλητής `x` θα πάρει την τιμή 2. Η έκφραση `x=x+2` σημαίνει ότι στη θέση μνήμης που συμβολίζεται με το `x` θα προστεθεί η τιμή 2 και το αποτέλεσμα θα τοποθετηθεί πάλι στη θέση μνήμης `x`. Δηλ. για ότι αλγεβρική έκφραση είναι στα δεξιά του $=$, θα γίνουν οι πράξεις, και το αποτέλεσμα θα αποθηκευτεί στη θέση μνήμης που υπάρχει στα αριστερά του $=$. Που σημαίνει ότι οι εκφράσεις στα δεξιά πρέπει να καταλήγουν σε

κάποιο αριθμητικό αποτέλεσμα και δεν επιτρέπεται στα αριστερά να έχουμε αριθμητικές πράξεις αλλά κάποια μεταβλητή που δηλώνει κάποια θέση μνήμης.

1.3 Το Octave σαν περιβάλλον εργασίας

Όταν αρχίζουμε και εισάγουμε εντολές στην κονσόλα ελέγχου και παίρνουμε διάφορα αποτελέσματα έχουμε αρχίσει μια συνεδρία (session). Γρήγορα γεμίζει το παράθυρο από εντολές και αποτελέσματα. Χρήσιμη εντολή εδώ είναι η `clc` που «καθαρίζει» το παράθυρο χωρίς να αφαιρέσει τίποτα από τη μνήμη του υπολογιστή. Αν οι εντολές που εισάγουμε δημιουργούν μεταβλητές που δεσμεύουν μεγάλο τμήμα της διαθέσιμης μνήμης του υπολογιστή, η εντολή `clear` καθαρίζει όλες τις μεταβλητές. Αν επιλεκτικά θέλουμε να αφαιρέσουμε συγκεκριμένα μια ή περισσότερες απλώς δίνουμε τα ονόματά τους σαν όρισμα στην `clear`. Π.χ. αν θέλουμε να αφαιρέσουμε τις `x1, x2` το κάνουμε με `clear x1, x2`. Ο πίνακας 1.3 συγκεντρώνει μερικές τέτοιες εντολές έτσι ώστε το Octave να γίνει εύχρηστο περιβάλλον εργασίας. Κοιτάξτε στο on-line help για λεπτομέρειες χρήσης των εντολών.

Πίνακας 1.3: Μερικές χρήσιμες εντολές

`clc`	καθαρίζει την οθόνη χωρίς να αφαιρέσει μεταβλητές που έχουμε εισάγει
`clear`	αφαιρεί από τη μνήμη όλες τις μεταβλητές που έχουμε εισάγει
`clear v1,v2,v3`	αφαιρεί μόνο τις συγκεκριμένες μεταβλητές `v1,v2,v3`
`who`	συνοπτικά όλες οι μεταβλητές που είναι φορτωμένες στη μνήμη
`whos`	με λεπτομέρεια όλες οι μεταβλητές που είναι φορτομένες στη μνήμη
`pwd`	παρουσιάζει το φάκελο εργασίας που βλέπει το Octave
`chdir` ή `cd`	αλλάζει το φάκελο εργασίας που βλέπει το Octave
`diary`	κρατάει αρχείο από μια συνεδρία (session)
`dir` ή `ls`	δίνει λίστα περιεχομένων του φακέλου εργασίας

Για να αλλάξουμε την τοποθεσία του φακέλλου εργασίας σε πριβάλλον CLI χρησιμοποιούμε την εντολή `chdir` ή `cd` με παράμετρο τον καινούργιο φάκελο, π.χ. `chdir('C:\work')`. Σε GUI περιβάλλον μπορούμε εύκολα να τον αλλάξουμε με τον ενσωματωμένο File Browser επάνω αριστερά.

Προσοχή. Σε περιβάλλον CLI εάν θέλουμε να δημιουργήσουμε δικό μας φάκελο εργασίας μέσα στο φάκελο `C:\work` με όνομα π.χ. 1234 και να μεταφερθούμε εκεί, το κάνουμε ως εξής:

```
>> cd C:\work
>> mkdir 1234
ans =  1
>> cd 1234
```

Καλό είναι να γίνεται πάντα επιβεβαίωση με την εντολή `pwd` για το ποιος είναι ο τρέχων φάκελος εργασίας. Με τον File Browser τα παραπάνω απλουστεύονται. Δοκιμάστε να αλλάξετε φάκελο εργασίας καθώς και να δημιουργήσετε καινούργιο φάκελο εργασίας. Το ποιος είναι ο τρέχων φάκελος εργασίας φαίνεται αμέσως.

Εάν τώρα θέλουμε να αποθηκεύσουμε κάποια συνεδρία από πράξεις μας σε κάποιο αρχείο, με όνομα π.χ. `tata`, πληκτρολογούμε την εντολή `diary('tata')`. Οτιδήποτε ακολουθεί κατόπιν είτε από δική μας πληκτρολόγηση είτε από αποτελέσματα που δίνει το Octave αποθηκεύονται στο αρχείο `tata` στον τρέχοντα φάκελο εργασίας. Προσοχή εδώ. Πολλά από τα ενδιάμεσα αποτελέσματα δεν γράφονται αμέσως στο αρχείο αλλά παραμένουν στην μνήμη RAM του υπολογιστή. Αν θέλουμε να τα αδειάσουμε από τη μνήμη στο αρχείο `tata` πρέπει να το κλείσουμε πρώτα με την εντολή `diary` χωρίς παραμέτρους. Μπορούμε κατόπιν να δούμε το αρχείο στον φάκελο εργασίας και να το ανοίξουμε π.χ. με τον Notepad++ (περιβάλλον CLI) ή διπλό-κλικ στο όνομα του αρχείου στον File Browser. Το αρχείο είναι απλό αρχείο κειμένου και ανοίγει με οποιοδήποτε ASCII editor ή ακόμα και με το Word. Προσοχή όμως και εδώ γιατί τα προγράμματα επεξεργασίας κειμένου όπως το Word τροποποιούν το περιεχόμενό του. Αν είναι για ετοιμασία κάποιας εργασίας σε ηλεκτρονική μορφή που θα εκτυπωθεί σε έντυπο, αυτό επιτρέπεται. Αν είναι για δημιουργία κάποιου script προγράμματος Octave αυτό απαγορεύεται. Το κατάλληλο εργαλείο στη δεύτερη περίπτωση είναι ένας editor (π.χ. Notepad++). Τα diary αρχεία είναι χρήσιμα για αποθήκευση συνεδρίας και κατόπιν, με editing και καθάρισμα έτσι ώστε να μείνουν οι καθαρά εκτελέσιμες εντολές, για φτιάξιμο προγραμμάτων script.

Το παρακάτω παράδειγμα δείχνει την αλλαγή φακέλλου εργασίας από την κονσόλα εντολών καθώς και το πως αντιδρά το πρόγραμμα όταν προσπαθούμε να αλλάξουμε φάκελο όταν αυτός δεν υπάρχει.

```
>> pwd
ans = C:\work
>> chdir fakelos
error: fakelos: No such file or directory
>> mkdir fakelos
ans =  1
>> chdir fakelos
>> pwd
ans = C:\work\fakelos
```

Παράδειγμα για να κρατήσουμε αρχείο μιας συνεδρίας:

```
>> diary('tata2')
>> 5+3*34
ans =  107
>> e=5*exp(3)
e =  100.43
>> diary
```

Κάντε διπλό-κλικ στο όνομα του αρχείου tata2 στο File Browser και θα δείτε τι έχει αποθηκευτεί εκεί.

1.4 Διανύσματα και πίνακες

Ο κύριος λόγος που προγράμματα σαν το Octave έχουν μεγάλη επιτυχία τα τελευταία χρόνια είναι το γεγονός ότι χειρίζονται διανύσματα και πίνακες σαν να είναι απλές μεταβλητές. Με τις ενσωματωμένες μάλιστα ρουτίνες πράξεων (βασισμένες στις γνωστές βιβλιοθήκες BLAS και LAPACK) μεταξύ πινάκων και διανυσμάτων, οι πράξεις είναι απλές, λιτές στην εμφάνιση και γρήγορες. Αυτό εξυπηρετεί την γρήγορη και σωστή ανάπτυξη προγραμμάτων και εφαρμογών και την δοκιμή νέων τεχνικών και μεθόδων.

Ένα διάνυσμα είναι απλώς μια ακολουθία αριθμών διατεταγμένων με κάποια συγκεκριμένη σειρά. Ακολουθεί παράδειγμα διανύσματος γραμμής

```
>> x=[0,1,3,6]
x =
        0        1        3        6
>> y = [6 3 1 0]
y =
        6        3        1        0
>> z = x + y
z =
        6        4        4        6
>> size(x)
ans =
        1        4
>> size(y)
ans =
        1        4
>> size(z)
ans =
        1        4
```

και διανύσματος στήλης

```
>> a=[1 3 -5]'
a =
        1
        3
       -5
>> size(a)
ans =
        3        1
>> b=[3; -6; 10]
b =
        3
       -6
       10
>> size(b)
ans =
        3        1
```

Χρησιμοποιούμε `[ ]` για να ορίσουμε τέτοιες ακολουθίες και χωρίζουμε τα στοιχεία τους με κόμμα ή κενό. Προσέξτε τη χρήση της συνάρτησης `size` που μας δίνει τις γραμμές/στήλες των διανυσμάτων. Προσέξτε επίσης τους δυο τρόπους για

την κατασκευή διανύσματος στήλης. Τον ένα με το ' και τον άλλο με το ;. Το ' αντιμεταθέτει γραμμές στήλες με τον μιγαδικό συζυγή. Για πραγματικούς αριθμούς δεν διαφέρει από τον ;.

Μπορούμε επίσης να κατασκευάσουμε διανύσματα από άλλα διανύσματα φτάνει να ταιριάζουν οι εκάστοτε διαστάσεις.

```
>> a=[1 2 3];
>> b=[4 5];
>> c=[a -b]
c =
        1        2        3       -4       -5
>> d=[-10; 3]
d =
      -10
        3
>> a
a =
        1        2        3
>> e=[a d]
error: horizontal dimensions mismatch (1x3 vs 2x1)
```

Όπως έχετε παρατηρήσει μέχρι τώρα αν ορίσουμε κάποια μεταβλητή ή κάνουμε πράξεις με διάφορες μεταβλητές το Octave παρουσιάζει αμέσως την μεταβλητή ή το αποτέλεσμα με την εκάστοτε τιμή της/του. Για την περίπτωση που θέλουμε να αποφύγουμε την «ηχώ» των ενδιαμέσων αποτελεσμάτων χρησιμοποιούμε το σύμβολο ; στο τέλος της έκφρασης όπως στο παραπάνω παράδειγμα.

Ένας εναλλακτικός τρόπος κατασκευής διανυσμάτων είναι με το σύμβολο/τελεστή : όπου δίνουμε αρχική και τελική τιμή (υπονοείται βήμα 1) ή, αρχική τιμή, βήμα και τελική τιμή.

```
>> x=1:10
x =
 Columns 1 through 9:
        1        2        3        4        5        6        7        8        9
 Column 10:
       10
>> y=0:0.1:1
y =
 Columns 1 through 9:
  0.00000  0.10000  0.20000  0.30000  0.40000  0.50000  0.60000  0.70000  0.80000
 Columns 10 and 11:
  0.90000  1.00000
>> Ts=0.2
Ts =  0.20000
>> t=0:Ts:1
t =
  0.00000  0.20000  0.40000  0.60000  0.80000  1.00000
>> x=-1:0.1:1
x =
 Columns 1 through 8:
   -1.0000   -0.9000   -0.8000   -0.7000   -0.6000   -0.5000   -0.4000   -0.3000
 Columns 9 through 16:
   -0.2000   -0.1000    0.0000    0.1000    0.2000    0.3000    0.4000    0.5000
 Columns 17 through 21:
    0.6000    0.7000    0.8000    0.9000    1.0000
>> x=-1:0.15:1
x =
 Columns 1 through 8:
   -1.0000   -0.8500   -0.7000   -0.5500   -0.4000   -0.2500   -0.1000    0.0500
 Columns 9 through 14:
    0.2000    0.3500    0.5000    0.6500    0.8000    0.9500
```

Βλέπουμε ότι με τον τελεστή : αρχίζουμε από το πρώτο στοιχείο και προχωράμε με το βήμα που έχουμε ορίσει μέχρι το τελευταίο. Το τελευταίο συμπεριλαμβάνεται εάν είναι πολλαπλάσιο του βήματος. Αλλιώς, όχι.

Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση linspace όπου θέτουμε αρχική και τελική τιμή και τον αριθμό των στοιχείων. Εδώ δεν χρειάζεται να υπολογίσουμε το βήμα. Η συνάρτηση θα υπολογίσει μια ομοιόμορφη απόσταση μεταξύ των στοιχείων και αρχίζοντας από το πρώτο με το βήμα που υπολόγισε προχωρά μέχρι και το τελευταίο. Το τελευταίο στοιχείο συμπεριλαμβάνεται στην ακολουθία.

```
>> linspace(-1,1,21)
ans =
 Columns 1 through 8:
  -1.00000  -0.90000  -0.80000  -0.70000  -0.60000  -0.50000  -0.40000  -0.30000
 Columns 9 through 16:
  -0.20000  -0.10000   0.00000   0.10000   0.20000   0.30000   0.40000   0.50000
 Columns 17 through 21:
   0.60000   0.70000   0.80000   0.90000   1.00000
```

```
>> linspace(-1,1,14)
ans =
 Columns 1 through 7:
  -1.000000  -0.846154  -0.692308  -0.538462  -0.384615  -0.230769  -0.076923
 Columns 8 through 14:
   0.076923   0.230769   0.384615   0.538462   0.692308   0.846154   1.000000
```

Ένας πίνακας δυο διαστάσεων μπορεί να οριστεί ως εξής:

```
>> A=[3 5 1 -4; 4 91 2 4; 0 4 -8 2]
A =
        3        5        1       -4
        4       91        2        4
        0        4       -8        2
```

όπου ορίσαμε έναν πίνακα 3×4, και όπως βλέπουμε χωρίζουμε τις γραμμές με το σύμβολο `;`.

Μπορούμε επίσης να αντιμεταθέσουμε γραμμές και στήλες με το σύμβολο `'`

```
>> A'
ans =
        3        4        0
        5       91        4
        1        2       -8
       -4        4        2
```

Τα παραπάνω ισχύουν και για πίνακες με μιγαδικούς αριθμούς, μόνο που τότε το σύμβολο `'` δείχνει ότι στην πραγματικότητα γίνεται ερμητιανή αντιμετάθεση (Hermitian Transpose) δηλ. στην αντιμετάθεση αντικαθιστούνται οι αριθμοί με τον μιγαδικό συζυγή τους. Αν θέλουμε απλή αντιμετάθεση χρησιμοποιούμε την συνάρτηση `conj` που υπολογίζει τον μιγαδικό συζυγή, ή το σύνθετο σύμβολο `.'`

```
>> x=[3 4+j*5 -2-j*9]
x =
        3 +      0i        4 +      5i       -2 -      9i
>> x'
ans =
        3 -      0i
        4 -      5i
       -2 +      9i
>> conj(x)'
ans =
        3 +      0i
        4 +      5i
       -2 -      9i
>> x.'
ans =
        3 +      0i
        4 +      5i
       -2 -      9i
```

Έστω τώρα ότι θέλουμε να λύσουμε το γραμμικό σύστημα

$$\begin{aligned} 3x_1 - 4x_2 + 5x_3 &= 0 \\ x_1 + 9x_2 + 2x_3 &= 3 \\ -4x_1 + 9x_2 + x_3 &= 1 \end{aligned}$$

Ορίζουμε τον πίνακα των συντελεστών `A`, το γνωστό διάνυσμα στήλης `b` και η λύση θα είναι το διάνυσμα στήλης `x`.

```
>> A=[3 -4 5; 1 9 2; -4 9 1]
A =
        3       -4        5
        1        9        2
       -4        9        1
>>  b=[0; 3; 1]
b =
        0
        3
        1
>>  x=inv(A)*b
x =
   0.4017094
   0.2905983
  -0.0085470
>> y=A\b
```

```
y =
   0.4017094
   0.2905983
  -0.0085470
```

όπου βλέπουμε ότι μπορούμε να χρησιμοποιήσουμε τη σχέση `x=inv(A)*b` ή την `x=A\b` για να βρούμε το `x`. Μπορούμε επίσης να κάνουμε μια επαλήθευση του αποτελέσματος αφαιρώντας το σύστημα Ax από το γνωστό διάνυσμα b

```
>>  A*x-b
ans =
  -2.0817e-16
  -4.4409e-16
   0.0000e+00
```

και βλέπουμε ότι η απάντηση είναι της τάξης 10^{-16}, πρακτικά μηδέν, όπως πρέπει να είναι. Είναι πιθανόν, στο δικό σας σύστημα, να δείτε λίγο διαφορετικά, αλλά και πάλι πολύ μικρά νούμερα.

Με τα διανύσματα και τους πίνακες βρίσκουμε πολύ χρήσιμες τις συναρτήσεις `length` και `size`. Η πρώτη μας δίνει το μήκος ενός διανύσματος (δηλ. τον αριθμό των στοιχείων του) και η δεύτερη το μέγεθος ενός διανύσματος ή πίνακα (δηλ. τον αριθμό των γραμμών και στηλών).

```
>> y=0:0.2:2
y =
 Columns 1 through 9:
  0.00000  0.20000  0.40000  0.60000  0.80000  1.00000  1.20000  1.40000  1.60000
 Columns 10 and 11:
  1.80000  2.00000
>> length(y)
ans =  11
>> size(y)
ans =
        1       11
>> A=[3 -9 0 12; 9 -2 -4 0; -1 9 5 2]
A =
        3       -9        0       12
        9       -2       -4        0
       -1        9        5        2
>> size(A)
ans =
        3        4
```

Οι δείκτες στα διανύσματα και στους πίνακες ξεκινούν από την τιμή 1 μέχρι το μέγεθος της εκάστοτε γραμμής ή στήλης. Αν θέλουμε κάποιο συγκεκριμένο στοιχείο το αναφέρουμε με τον ή τους δείκτες του.

```
>> x=0:0.1:1
x =
 Columns 1 through 9:
  0.00000  0.10000  0.20000  0.30000  0.40000  0.50000  0.60000  0.70000  0.80000
 Columns 10 and 11:
  0.90000  1.00000
>>  x(3)
ans =  0.20000
>> x(8)
ans =  0.70000
>> A
A =
        3       -9        0       12
        9       -2       -4        0
       -1        9        5        2
>> A(2,3)
ans = -4
>> A(4,2)
error: A(I,J): row index out of bounds; value 4 out of bound 3
>> A(3,1)
ans = -1
```

Μπορούμε επίσης να χρησιμοποιήσουμε κατάλληλα τους δείκτες και να παράγουμε άλλα διανύσματα ή πίνακες. Π.χ.

```
>> u=1:10
u =
 Columns 1 through 9:
        1        2        3        4        5        6        7        8        9
 Column 10:
       10
>> u(1:2:10)
```

```
ans =
        1        3        5        7        9
>> u(3:2:8)
ans =
        3        5        7
>> A
A =
        3       -9        0       12
        9       -2       -4        0
       -1        9        5        2
>> A(1:3,1:3)
ans =
        3       -9        0
        9       -2       -4
       -1        9        5
>> A(2,1:3)
ans =
        9       -2       -4
>> A(3,:)
ans =
       -1        9        5        2
>>  A(:)
ans =
        3
        9
       -1
       -9
       -2
        9
        0
       -4
        5
       12
        0
        2
```

Βλέπουμε ότι μπορούμε να πάρουμε συγκεκριμένα υποσύνολα από διανύσματα ή πίνακες φτάνει να μπορούμε να τα περιγράψουμε κατάλληλα με τους δείκτες τους. Μπορείτε να περιγράψετε εσείς τι ακριβώς συμβαίνει στο παραπάνω παράδειγμα;

Επειδή η ανάγκη δημιουργίας διανυσμάτων με συγκεκριμένη αρχική/τελική τιμή και βήμα σε γραμμική ή λογαριθμική κλίμακα είναι πολύ συχνή (ειδικά σε γραφικές παραστάσεις) μπορούμε να χρησιμοποιήσουμε τις παρακάτω συναρτήσεις:

```
>> linspace(0,1,20)
ans =
 Columns 1 through 8:
   0.00000   0.05263   0.10526   0.15789   0.21053   0.26316   0.31579   0.36842
 Columns 9 through 16:
   0.42105   0.47368   0.52632   0.57895   0.63158   0.68421   0.73684   0.78947
 Columns 17 through 20:
   0.84211   0.89474   0.94737   1.00000
>> logspace(0,3,15)
ans =
 Columns 1 through 7:
      1.0000      1.6379      2.6827      4.3940      7.1969     11.7877     19.3070
 Columns 8 through 14:
     31.6228     51.7947     84.8343    138.9495    227.5846    372.7594    610.5402
 Column 15:
   1000.0000
```

Πληκτρολογήστε `help linspace` και `help logspace` και προσπαθείστε να καταλάβετε τι σημαίνουν οι παράμετροι στις παραπάνω συναρτήσεις.

Πρόσθεση και αφαίρεση με πίνακες προϋποθέτει ίσες διαστάσεις.

```
>> a=[2 4 8]; b=[3 2 2];
>> a+b
ans =
        5        6       10
>> a-b
ans =
       -1        2        6
>> c=[-1; 8; 3]
c =
       -1
        8
        3
>> a+c
ans =
```

```
        1        3        7
       10       12       16
        5        7       11

error: operator +: nonconformant arguments (op1 is 1x3, op2 is 3x1)
>> A=[3 -1 0; 2 4 -1; 8 7 4]
A =
        3       -1        0
        2        4       -1
        8        7        4
>> B=[4 2 8; 6 -3 -5; 1 4 2]
B =
        4        2        8
        6       -3       -5
        1        4        2
>> A+B
ans =
        7        1        8
        8        1       -6
        9       11        6
>> C=[B c]
C =
        4        2        8       -1
        6       -3       -5        8
        1        4        2        3
>> A-C
error: operator -: nonconformant arguments (op1 is 3x3, op2 is 3x4)
```

Πολλαπλασιαμός σταθεράς με πίνακα σημαίνει πολ/σμός όλων των στοιχείων του πίνακα με τη σταθερά.

```
>> 2*A
ans =
        6       -2        0
        4        8       -2
       16       14        8
>> (-3)*A
ans =
       -9        3       -0
       -6      -12        3
      -24      -21      -12
```

Το εσωτερικό γινόμενο δυο διανυσμάτων προϋποθέτει ότι το πρώτο είναι διάνυσμα γραμμής και το δεύτερο διάνυσμα στήλης με ίσο πλήθος στοιχείων:

```
>> a=[1 2 3]
a =
        1        2        3
>> b=[4; 5; 6]
b =
        4
        5
        6
>> a*b
ans =  32
```

Υπάρχει και η συνάρτηση `dot` που υπολογίζει το εσωτερικό γινόμενο δυο διανυσμάτων καθώς και η συνάρτηση `cross` για το εξωτερικό γινόμενο. Η τελευταία είναι μόνο για διανύσματα με τρία στοιχεία. Επιβεβαιώστε τα παρακάτω και κάνετε και τους δικούς σας πειραματισμούς.

```
>> a=[3 1 -4]
a =
        3        1       -4
>> b=[-2 9 1]
b =
       -2        9        1
>> dot(a,b)
ans = -1
>> cross(a,b)
ans =
       37        5       29
```

Πολ/σμός πίνακα $A(m,n)$ επί πίνακα $B(p,q)$ σημαίνει πίνακα $C(m,q)$ εάν $n=p$.

```
>> A=[3 -1; 2 4; 8 7]
A =
        3       -1
```

```
         2        4
         8        7
>> B=[-1 0; 7 2]
B =
        -1        0
         7        2
>> A*B
ans =
       -10       -2
        26        8
        41       14
```

Συχνά όμως μας ενδιαφέρει όχι ο κανονικός πολ/σμός μεταξύ δυο διανυσμάτων αλλά ο πολ/σμός ή διαίρεση στοιχείο προς στοιχείο.

```
>> a=[2 4 8]
a =
        2        4        8
>> b=[3 2 2]
b =
        3        2        2
>> a*b
error: operator *: nonconformant arguments (op1 is 1x3, op2 is 1x3)
>> a.*b
ans =
        6        8       16
>> a./b
ans =
  0.66667  2.00000  4.00000
```

Αυτό επιτυγχάνεται με την προσθήκη μιας τελείας πριν το σύμβολο πολ/σμού ή διαίρεσης. Η παραπάνω ιδιότητα είναι πολύ χρήσιμη στην πινακοποίηση πράξεων. Επεκτείνεται επίσης και στην ύψωση σε δύναμη όλων των στοιχείων ενός πίνακα.

Επισημαίνεται ότι το πιο κοινό λάθος όταν σχεδιάζετε ένα πρόγραμμα για να λύσετε κάποιο πρόβλημα στο Octave είναι κάποιο μπέρδεμα με τις γραμμές / στήλες διανυσμάτων ή πινάκων. Καλό είναι πάντα να επιβεβαιώνετε με την εντολή `size` ότι οι γραμμές / στήλες είναι αυτές που νομίζετε και όχι το ανάποδο.

1.5 Ρίζες πολυωνύμων

Μπορούμε να ορίσουμε ένα πολυώνυμο στο Octave σαν ένα διάνυσμα με στοιχεία τους συντελεστές του πολυωνύμου αρχίζοντας από τον συντελεστή του πιο υψηλόβαθμου όρου. Π.χ. για το πολυώνυμο $p(x) = -5 + 7x - 8x^2 + 4x^3$ έχουμε

```
>> c=[4 -8 7 -5]
c =
        4       -8        7       -5
>> roots(c)
ans =
  1.38802 + 0.00000i
  0.30599 + 0.89829i
  0.30599 - 0.89829i
>> format('long')
>>  roots(c)
ans =

  1.388022717514269 + 0.000000000000000i
  0.305988641242865 + 0.898294261774783i
  0.305988641242865 - 0.898294261774783i
```

Η συνάρτηση `roots` μας δίνει τις ρίζες του και όπως βλέπουμε έχουμε μια πραγματική ρίζα και δυο συζυγείς μιγαδικές.

1.6 Είσοδος/έξοδος δεδομένων από/σε εξωτερικά αρχεία

Το Octave είναι χρήσιμο «εργαλείο» αλλά τίποτα παραπάνω από ένα «κομπιουτεράκι» αν δεν μπορούμε να έχουμε είσοδο/έξοδο δεδομένων από/σε εξωτερικά αρχεία.

Με την εντολή `load` μπορούμε να διαβάσουμε κάποιο εξωτερικό αρχείο το οποίο είναι σε μορφή text. Ένα κοινό παράδειγμα είναι να έχουμε κάποιες μετρήσεις από κάποιο μέγεθος συναρτήσει του χρόνου, π.χ. τάση σε κάποια αντίσταση.

Έστω ότι έχουμε ένα τέτοιο αρχείο, με όνομα toto1.d. Οι μετρήσεις είναι 100 τον αριθμό και καταχωρημένες σε δυο στήλες (πρώτη στήλη χρόνος σε msec, δεύτερη στήλη τάση σε Volt).

Μπορούμε να διαβάσουμε τις μετρήσεις με την εντολή `load`. Το Octave αποθηκεύει στον πίνακα toto1 (ίδιο όνομα με το όνομα του αρχείου) όλα τα δεδομένα. Η εντολή size μας δίνει τις διαστάσεις του πίνακα, δηλ. αριθμό γραμμών και στηλών.

```
>> load toto1.d
>> toto1
toto1 =

        2       11
        3       16
        4       21
        5       26
        6       31
...
>> size(toto1)
ans =
      100        2
```

Αν θέλουμε τις τιμές του χρόνου και της τάσης σε διαφορετικά διανύσματα για περαιτέρω επεξεργασία μπορούμε να το κάνουμε ως εξής:

```
>> t=toto1(:,1)
t =
        2
        3
        4
        5
        6
...
>> v=toto1(:,2)
v =

       11
       16
       21
       26
       31
...
```

Η άνω/κάτω τελεία στον πρώτο δείκτη (που αντιστοιχεί στις γραμμές) σημαίνει ότι παίρνουμε όλες τις γραμμές. Ο δεύτερος δείκτης αντιστοιχεί στην στήλη. Επομένως `toto1(:,1)` σημαίνει ότι παίρνουμε όλες τις γραμμές από την πρώτη στήλη ενώ `toto1(:,2)` σημαίνει ότι παίρνουμε όλες τις γραμμές από την δεύτερη στήλη. Προφανώς εάν το αρχείο έχει παραπάνω από δυο στήλες μπορούμε με ανάλογο τρόπο να ξεχωρίσουμε όποια στήλη θέλουμε.

Υπάρχουν και άλλοι τρόποι να διαβάσουμε εξωτερικά αρχεία. Κοιτάξτε το help της εντολής load. Υπάρχουν ομοίως πολλοί τρόποι να γράψουμε δεδομένα σε εξωτερικά αρχεία. Κοιτάξτε το help της εντολής save.

Παρουσιάζεται εδώ ένας από αυτούς τους τρόπους:

Έστω ότι έχουμε δυο διανύσματα x και y και θέλουμε να γράψουμε τις τιμές τους σε κάποιο αρχείο ofile.

```
>> x=1:10
x =
 Columns 1 through 9:
        1        2        3        4        5        6        7        8        9
 Column 10:
       10
>> y=x.^2
y =
 Columns 1 through 9:
        1        4        9       16       25       36       49       64       81
 Column 10:
      100
>> save -binary ofile x y
```

Η παραπάνω εντολή save σε binary format σώζει τις μεταβλητές x και y στο αρχείο ofile. Για να διαβάσουμε ένα τέτοιο αρχείο σε άλλη συνεδρία όπου δεν υπάρχουν στο χώρο εργασίας οι μεταβλητές x και y ακολουθούμε την εξής διαδικασία:

```
>> load -binary ofile
>> x
x =
 Columns 1 through 9:
```

```
       1        2        3        4        5        6        7        8        9
 Column 10:
      10
>> y
y =
 Columns 1 through 9:
       1        4        9       16       25       36       49       64       81
 Column 10:
     100
```

Αυτόματα φορτώθηκε το αρχείο και εμφανίστηκαν οι μεταβλητές x και y. Πειραματιστείτε με δικές σας παραλλαγές για 1 ή 3 ή 4 μεταβλητές, διανύσματα και πίνακες.

Αναφέρεται επίσης ο τρόπος να έχουμε binary αρχεία συμβατά μεταξύ octave και matlab με save -mat ofile.mat και load ofile.mat όπου χρειάζεται η κατάληξη .mat στα αρχεία. Και εδώ αποθηκεύονται όσες μεταβλητές δηλώνονται ή όλες όσες υπάρχουν στον χώρο εργασίας όπως προηγουμένως.

Παρουσιάζεται επίσης ένας τρόπος που απαιτεί γνώση scripts και for loops (τα μαθαίνετε στα επόμενα). Το παρακάτω script δείχνει τις εντολές όπου ανοίγουμε ένα εξωτερικό αρχείο για γράψιμο και γράφουμε μέσα από for loop στοιχείο προς στοιχείο τις τιμές των διανυσμάτων x και y χρησιμοποιώντας τη συνάρτηση fprintf.

```
fid=fopen('ffile.d','w')
for i=1:length(x)
  fprintf(fid,"%10.2f %10.2f\n",x(i),y(i))
end
fclose(fid)
```

Το αποτέλεσμα στο αρχείο ffile.d είναι:

```
  1.00       1.00
  2.00       4.00
  3.00       9.00
  4.00      16.00
  5.00      25.00
  6.00      36.00
  7.00      49.00
  8.00      64.00
  9.00      81.00
 10.00     100.00
```

Η `fopen` από το file open ανοίγει ένα αρχείο με όνομα την πρώτη παράμετρο της συνάρτησης `fopen` όπου το όνομα είναι χαρακτήρες string μέσα σε απλά ή διπλά «αυτάκια». Το `'w'` σημαίνει ότι το αρχείο που ανοίγουμε είναι για γράψιμο (write). Κοιτάξτε στο `help fopen` για περισσότερες λεπτομέρειες. Η μεταβλητή `fid` είναι ο file descriptor που αντιστοιχεί στο συγκεκριμένο αρχείο και μπαίνει σαν παράμετρος στην fprintf(fid, ...) που μας επιτρέπει να έχουμε formatted print (μορφοποιημένη εκτύπωση). Αυτό γίνεται με ένα string ακολουθούμενο από τις μεταβλητές που θέλουμε να εκτυπώσουμε (στο παράδειγμα `x(i), y(i)`.) Μέσα στο string έχουμε «συνθηματικά» που αντιστοιχούν στον τρόπο που θέλουμε να εμφανιστεί η τιμή των μεταβλητών. Στο συγκεκριμένο παράδειγμα τα «συνθηματικά» είναι `%10.2f`, δηλ. πραγματικοί αριθμοί float σε χώρο 10 θέσεων με 2 δεκαδικά ψηφία. Το `\n` είναι χαρακτήρας ελέγχου που λέει πήγαινε στην επόμενη γραμμή (πειραματιστείτε αφαιρώντας τον). Όταν τελειώνετε με το αρχείο καλό είναι να το κλείνετε με την `fclose` για να μην υπάρξει πιθανότητα παραμόρφωσης του περιεχομένου. Μερικά ακόμα παραδείγματα με την printf (η μόνη διαφορά με fprintf είναι ότι γράφει στην κονσόλα, όχι σε αρχείο).

```
octave:15> a=10.3452
a =  10.345
octave:17> printf("This is a = %8.3f Volts\n",a)
This is a =   10.345 Volts
octave:19> printf("Color %s, number %d, hex %#x, float %5.2f\n", "red", 123456, 255, 3.14159)
Color red, number 123456, hex 0xff, float  3.14
octave:20> printf("This is a = %10.2e\n",a)
This is a =   1.03e+01
```

Για περισσότερες λεπτομέρειες κοιτάξτε στο `https://en.wikipedia.org/wiki/Printf_format_string`.

Αυτός ο τρόπος δίνει περισσότερο έλεγχο στη μορφοποίηση των δεδομένων και η φυσική του θέση είναι εδώ. Μπορείτε να επανέλθετε όταν έχετε κατανοήσει τις απαιτούμενες εντολές.

1.7 Είσοδος δεδομένων διαδραστικά

Υπάρχει και η δυνατότητα διαδραστικής εισόδου δεδομένων. Π.χ. όταν θέλουμε το πρόγραμμα να μας ρωτήσει τι τιμή θα δώσουμε σε κάποια μεταβλητή ή κάποια πληροφορία όπως το όνομά μας, χρησιμοποιούμε την εντολή input. Προσοχή,

στα παρακάτω μπορεί να έχετε πρόβλημα με τα ελληνικά στην κονσόλα σας. Αλλάξτε τότε σε λατινικούς χαρακτήρες και επανέρχεστε όταν μάθετε να φτιάχνετε script προγράμματα με τον editor όπου τα ελληνικά δουλεύουν.

```
%Δοκιμή προγράμματος διαδραστικής εισόδου δεδομένων

radius = input('δώστε την ακτίνα: ')
area = pi*radius^2;
volume = (4/3)*pi*radius^3;

onoma = input('δώστε το όνομά σας: ','s');
disp(onoma);
disp(['Ο ',onoma,' έδωσε ακτίνα radius = ',num2str(radius),' m']);
disp(['και υπολογίστηκε εμβαδόν κύκλου = ',num2str(area),' m^2']);
disp(['και όγκος κύκλου = ',num2str(volume),' m^3']);


δώστε την ακτίνα: 45
radius =  45
δώστε το όνομά σας: Tasos
Tasos
Ο Tasos έδωσε ακτίνα radius = 45 m
και υπολογίστηκε εμβαδόν κύκλου = 6361.7251 m^2
και όγκος κύκλου = 381703.5074 m^3
```

Η input δεν έχει πρόβλημα με αριθμητικές τιμές και το string στην συνάρτηση εμφανίζεται για να μας ετοιμάσει να εισάγουμε δεδομένα. Μπορούμε να δώσουμε και κάποιο string στην input φτάνει να το δηλώσουμε με την δεύτερη παράμετρο s. Το πρόβλημα που παρατήρησα είναι ότι δέχεται λατινικούς χαρακτήρες και όχι ελληνικά.

Με την άλλη συνάρτηση disp βλέπουμε άλλον έναν τρόπο εξόδου δεδομένων μόνο που πρέπει να είναι string. Οι αριθμητικές τιμές μετατρέπονται σε string με την συνάρτηση num2str και μπορούμε να «κολλήσουμε» πολλές μαζί με τον τρόπο που φαίνεται στο παράδειγμα.

1.8 Γραφικές παραστάσεις

1.8.1 Απλή γραφική

Το Octave δεν προσφέρει μόνο ένα περιβάλλον για να κάνουμε πράξεις αλλά επίσης και τη δυνατότητα να απεικονίζουμε τα αποτελέσματά μας με μια μεγάλη ποικιλία γραφικών παραστάσεων. Όπως λένε οι Άγγλοι «μια εικόνα αξίζει όσο χίλιες λέξεις».

Έστω ότι θέλουμε να δούμε γραφικά τη συνάρτηση $f(x) = 5e^{-x/2}\sin(x)$. Διαλέγουμε αυθαίρετα για την ανεξάρτητη μεταβλητή x το διάστημα από 0 έως 10 (εάν θέλουμε κάτι άλλο μπορούμε μετά να το αλλάξουμε) με βήμα 0.1. Ορίζουμε μετά το διάνυσμα των τιμών της συνάρτησης και όπως βλέπουμε χρειαζόμαστε τον τελεστή `.*` για να πολλαπλασιάσουμε στοιχείο προς στοιχείο τα διανύσματα $5e^{-x/2}$ και $\sin(x)$.

```
x=0:0.1:10;
y=5*exp(-x/2).*sin(x);
plot(x,y); grid;
title("This is ένας Ελληνικός τίτλος");
xlabel("χρόνος [sec]"); ylabel("τάση [Volt]");
```

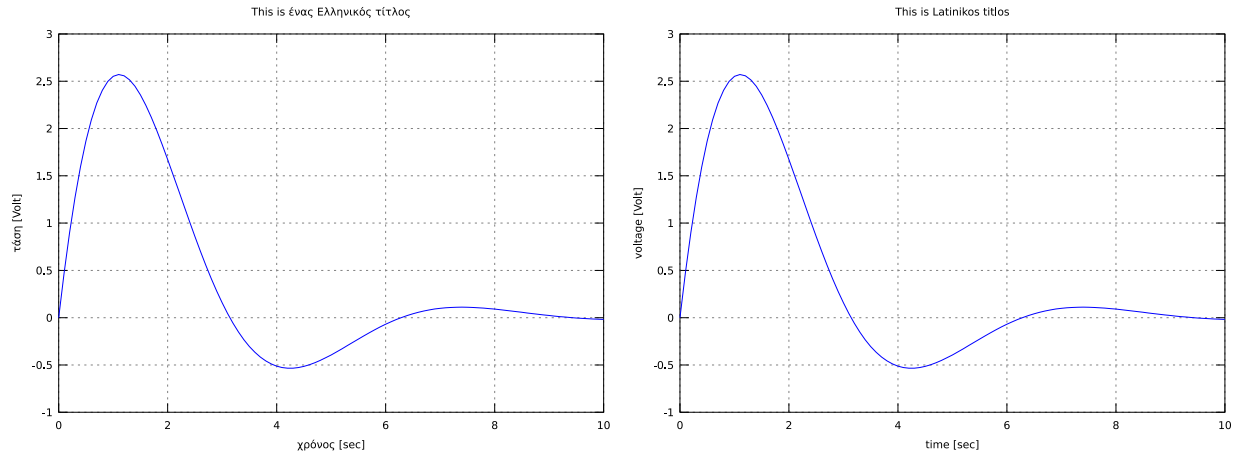
Είναι πιθανόν στο σύστημά σας να μην έχετε υποστήριξη Ελληνικών στην κονσόλα σας (θα έχετε στον editor). Αντικαταστήσετε τότε τους παραπάνω ελληνικούς χαρακτήρες με λατινικούς και συνεχίστε.

```
x=0:0.1:10;
y=5*exp(-x/2).*sin(x);
plot(x,y); grid;
title("This is Latinikos titlos");
xlabel("time [sec]"); ylabel("voltage [Volt]");
```

Η εντολή `plot(x,y)` παρουσιάζει τη γραφική παράσταση των δυο διανυσμάτων (σχ. 1.7) και οι εντολές `title, xlabel, ylabel` και `grid` προσθέτουν τίτλο, υπότιτλους και πλέγμα.

Τις γραφικές παραστάσεις δεν τις θέλουμε απλώς να τις βλέπουμε, αλλά και να τις ενσωματώνουμε σε αρχεία επεξεργασίας κειμένου για εργασίες ή δημοσιεύσεις. Η εντολή για να αποθηκεύσουμε μια γραφική (εφόσον φυσικά είναι έτοιμη) σε εξωτερικό αρχείο τύπου π.χ. png με όνομα grafiki.png στον τρέχοντα φάκελο εργασίας είναι:

```
print grafiki.png
```



Σχήμα 1.7: Απλή γραφική παράσταση από το Octave με ελληνικούς ή λατινικούς χαρακτήρες. Οι παραπάνω παραστάσεις έχουν βγει από μετατροπή αρχείων svg σε pdf και έχουν ενσωματωθεί εδώ.

Png είναι η επέκταση του αρχείου. Η επέκταση του αρχείου βέβαια θα φανεί εφόσον ο υπολογιστής είναι ρυθμισμένος έτσι ώστε να φαίνεται (κοιτάξτε στα Folder Options). Προφανώς αντί για όνομα αρχείου grafiki μπορείτε να χρησιμοποιήσετε όνομα της επιλογής σας.

Στην έκδοση 4.2.1, για υποστήριξη ελληνικών, ο πιο κατάλληλος τύπος γραφικών με δυνατότητα να υποστηρίξει λατινικούς και ελληνικούς χαρακτήρες είναι ο svg.

```
print grafiki.svg
```

Svg (scalable vector graphics) είναι ένας καινούργιος τύπος γραφικών, vector graphics, με πολύ καλές δυνατότητες. Vector graphics στην πράξη σημαίνει ότι η γραφική μπορεί να τροποποιηθεί σε μέγεθος μετά την εισαγωγή της σε πρόγραμμα επεξεργασίας κειμένου (π.χ. σμίκρυνση, επιμήκυνση με το ποντίκι) χωρίς να αλλοιωθεί η ποιότητα. Σε αντίθεση, οι γραφικές bitmap graphics, παρουσιάζουν αλλοίωση στην ποιότητα αν τροποποιηθούν από τις αρχικές τους διαστάσεις. Το κατάλληλο πρόγραμμα επεξεργασίας γραφικών που δίνει τη δυνατότητα να ανοίξετε, να δείτε, να επεξεργαστείτε και να μετατρέψετε αρχεία svg σε άλλες μορφές, π.χ. png, pdf ή emf είναι το inkscape.

Το online help με την εντολή `help print` δείχνει τους διάφορους τύπους γραφικών που υποστηρίζονται. Φυσικά, όλοι οι τύποι γραφικών υποστηρίζουν καλά τους λατινικούς χαρακτήρες. Συνιστάται πάντα να αποφεύγετε τη χρήση της μορφής bmp γιατί το μέγεθος είναι τεράστιο και το όφελος μηδέν.

Σημείωση:

Ο όρος figure αναφέρεται στο παράθυρο γραφικών και το plot στην ίδια την γραφική παράσταση που δημιουργείται με την αντίστοιχη εντολή και μπορεί να περιέχει περισσότερες της μιας γραφικές παραστάσεις. Κοιτάξτε στο help για λεπτομέρειες π.χ. σχετικά με πάχος γραμμών, τύπο (συνεχείς, διακεκομμένες), χρώμα και σύμβολα.

1.8.2 Γραφικές 3 διαστάσεων

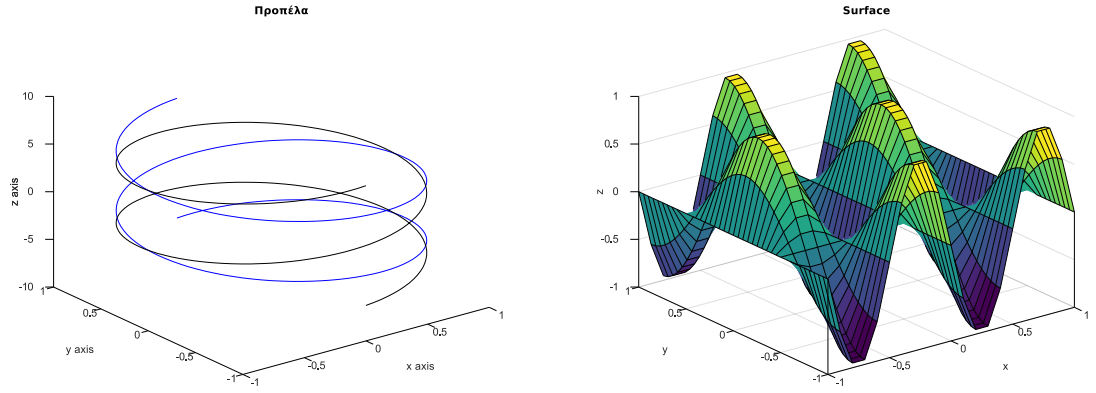
Δίνεται εδώ και μια επισκόπηση με γραφικές 3 διαστάσεων.

Μια παραμετρική ελικοειδής γραμμή όπως αυτή που διαγράφει π.χ. μια προπέλα φαίνεται στην αρχή του παρακάτω παραδείγματος. Με τις hold on, hold off μπορούμε να κρατήσουμε το περιεχόμενο της πρώτης γραφικής έτσι ώστε να μην διαγραφεί από την δεύτερη γραφική που διαγράφει την προπέλα ανάποδα. Φαίνεται επίσης ο τρόπος που δίνουμε μπλε χρώμα στην πρώτη και μαύρο στην δεύτερη. Φαίνεται επίσης και ένας εναλλακτικός τρόπος με την saveas να έχουμε εξωτερικό αρχείο της γραφικής μας.

```
clear all; close all;
t = linspace(-2*pi,2*pi,200);
plot3(sin(t), cos(t), t, 'b-');
title('Προπέλα'); xlabel('x axis'); ylabel('y axis'); zlabel('z axis');
hold on
plot3(sin(t), -cos(t), t, 'k-');
hold off

saveas(gca,'myplot3.svg','svg');
```

Για μια 3-διάστατη επιφάνεια ορίζουμε πρώτα ένα πλέγμα με τα xy σημεία με την meshgrid, τα σημεία της επιφάνειας Z και τέλος την ίδια την επιφάνεια με την surf.

```
clear all; close all;
x = linspace(-1,1,21);
y = linspace(-1,1,41);
[X,Y] = meshgrid(x,y);
Z = sin(2*pi*X) .* cos(pi*Y);
surf(X,Y,Z)
title('Surface'); xlabel('x'); ylabel('y'); zlabel('z');

print myplot4.svg
```



Σχήμα 1.8: Γραφική παράσταση παραμετρικής καμπύλης σε 3 διαστάσεις καθώς και μια επιφάνεια.

1.8.3 Γραφική θεωρίας και πειραματικών δεδομένων

Ένα άλλο κοινό πρόβλημα είναι να έχουμε ορισμένες πειραματικές μετρήσεις καθώς και τη μαθηματική σχέση που πρέπει να ακολουθούν από τη θεωρία και να θέλουμε την απεικόνησή τους στην ίδια γραφική. Π.χ.την ισχύ σε μια αντίσταση όταν μεταβάλλεται η τάση στα άκρα της. Έστω ότι η τιμή της αντίστασης είναι $R = 2.2\ \mathrm{k\Omega}$. Η θεωρία λέει ότι πρέπει

$$P = \frac{V^2}{R}$$

και οι πειραματικές μετρήσεις φαίνονται στον πίνακα 1.4.

Πίνακας 1.4: Πειραματικές μετρήσεις ισχύος έναντι τάσης

V [V]	P [W]
5	0.01
20	0.2
30	0.5
50	1.0
70	2.1

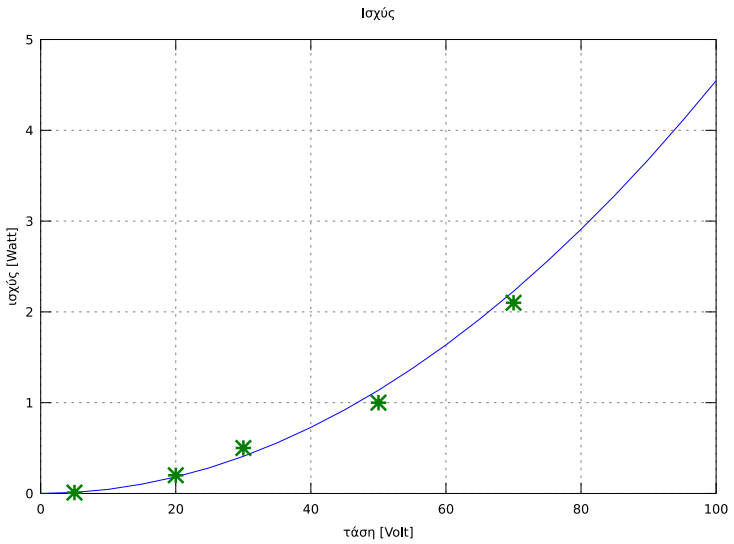
Ο παρακάτω κώδικας δημιουργεί την γραφική στο σχ. 1.9 όπου φαίνονται οι πειραματικές μετρήσεις ως σημεία (μέγεθος και τύπος συμβόλου μπορεί να αλλάξει - κοιτάξτε το help) και η θεωρητική καμπύλη ως συνεχής γραμμή.

```
R=2.2e3;
V=0:5:100;
P=V.^2/R;
Vp=[5 20 30 50 70];
Pp=[0.01 0.2 0.5 1 2.1];
plot(V,P,Vp,Pp,'*')
grid; title("Ισχύς"); xlabel("τάση [Volt]"); ylabel("ισχύς [Watt]");
```

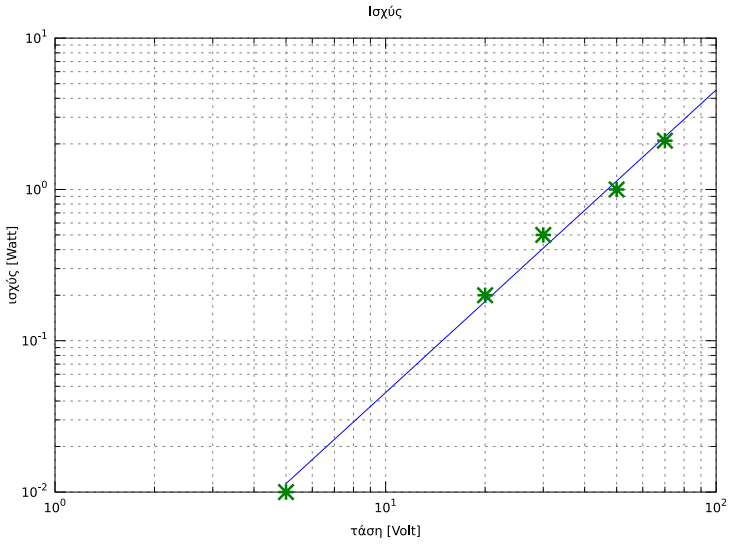
Ενδιαφέρον έχει αν θέλουμε τη γραφική σε λογαριθμική κλίμακα. Αντικαθιστούμε απλώς την plot από την loglog, δηλ.

```
loglog(V,P,Vp,Pp,'*')
```

και το αποτέλεσμα φαίνεται στο σχ. 1.10.



Σχήμα 1.9: Γραφική παράσταση για σύγκριση θεωρίας και πειράματος, μετρήσεων ισχύος έναντι τάσης σε μια αντίσταση.



Σχήμα 1.10: Η ίδια γραφική σε λογαριθμική κλίμακα.

Τα διανύσματα `Vp` και `Pp` περιέχουν τις πειραματικές μετρήσεις ενώ τα `V` και `P` περιέχουν σημεία από τη θεωρητική σχέση. Στη γραφική επιλέγουμε τα θεωρητικά σημεία να τα έχουμε σε συνεχή γραμμή ενώ τα πειραματικά με διακεκριμένα σημεία με το σύμβολο `*`. Προσέξτε τη χρήση της `plot` σε αυτή την περίπτωση.

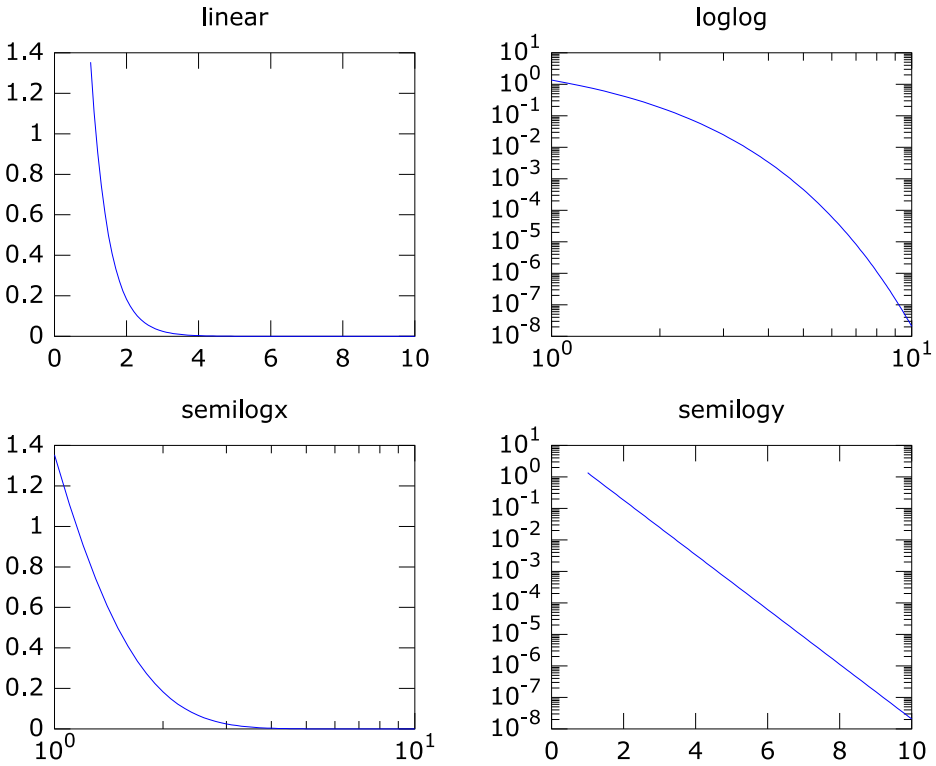
Σημειώνεται εδώ ότι αν θέλετε να έχετε πολλά παράθυρα γραφικών ανοικτά συγχρόνως, χρησιμοποιείστε την εντολή `figure` πριν την εκάστοτε `plot`. Αναφέρεται επίσης και η εντολή `subplot` όπου μπορείτε να δημιουργήσετε περισσότερες γραφικές παραστάσεις σε ένα παράθυρο. Αν θέλετε, ψάξτε στο online help καθώς και στο διαδίκτυο για παραδείγματα.

Υπάρχουν και ορισμένες άλλες παραλλαγές για γραφικές παραστάσεις όταν το εύρος τιμών που χρειάζεται να παρασταθεί είναι μεγάλο. Αυτές είναι οι λογαριθμικές `loglog`, `semilogx`, `semilogy`. Στην `loglog` και οι δυο άξονες είναι σε λογαριθμική κλίμακα. Στην `semilogx` ο οριζόντιος άξονας είναι σε λογαριθμική κλίμακα ενώ ο κατακόρυφος σε γραμμική. Στην `semilogy` ο οριζόντιος άξονας είναι σε γραμμική κλίμακα ενώ ο κατακόρυφος σε λογαριθμική.

Έχουμε για παράδειγμα

```
x=1:0.1:10;
y=10*exp(-2*x);
```

```
subplot(2,2,1); plot(x,y); title('linear')
subplot(2,2,2); loglog(x,y); title('loglog')
subplot(2,2,3); semilogx(x,y); title('semilogx')
subplot(2,2,4); semilogy(x,y); title('semilogy')
figure
subplot(111)
semilogy(x,y); title('semilogy'); grid
```



Σχήμα 1.11: Γραφική παράσταση 2×2 subplot.

όπου χωρίζουμε ένα παράθυρο γραφικής στα τέσσερα, 2×2. Με την εντολή `subplot(2,2,1)` πριν το κάλεσμα της εντολής `plot` για γραφική, η γραφική πάει στο πρώτο τέταρτο. Με την `subplot(2,2,2)` η γραφική πάει στο δεύτερο τέταρτο στην πρώτη γραμμή. Με τις άλλες, γεμίζει και η δεύτερη γραμμή. Όταν τελειώνουμε με τις subplot καλούμε την εντολή `subplot(111)` για να ξαναγυρίσουμε στο περιβάλλον μιας γραφικής ανά παράθυρο. Επανασχεδιάζουμε την κατόπιν την τελευταία γραφική που φαίνεται ευθεία γραμμή.

Η αρχική σχέση είναι $y = 10\exp(-2x)$. Η semilogy έχει τον κατακόρυφο άξονα λογαριθμικό και φαίνεται ευθεία γραμμή. Αυτό σημαίνει ότι από τη γραφική παράσταση μπορούμε να υπολογίσουμε την κλίση και να δούμε αν συμφωνεί με τη σχέση. Αυτός είναι ο τρόπος που επεξεργαζόμαστε πειραματικά δεδομένα που ακολουθούν κάποιο εκθετικό νόμο για να δούμε αν συμφωνούν με την θεωρητική σχέση.

Λογαριθμίζοντας (αυτή είναι η σχέση που απεικονίζεται στη semilogy γραφική) έχουμε:

$$\log_{10} y = \log_{10}\left(10\, e^{-2x}\right) = \log_{10} 10 - 2x \log_{10} e$$

Μπορούμε να κάνουμε zoom στη γραφική (δεξιό κουμπί στο ποντίκι) και παρατηρούμε ότι $x = 1$ αντιστοιχεί στην τιμή 1.35635 και $x = 5$ αντιστοιχεί στην τιμή 0.000448105. Επανερχόμαστε στην προηγούμενη εικόνα (χωρίς zoom) πληκτρολογώντας p. Ή, αν έχουμε κάνει πολλά zoom, πληκτρολογούμε u για επιστροφή στην αρχική εικόνα.

Λογαριθμίζοντας τις αντίστοιχες τιμές (οι οποίες είναι της μεταβλητής y) για να γίνουν τιμές της μεταβλητής $\log_{10} y$ έχουμε 0.13237 και -3.3486 αντίστοιχα. Η κλίση είναι

$$\frac{-3.3486 - 0,13237}{5 - 1} = -0.87025$$

και συμφωνεί αρκετά καλά με την τιμή $-2\log_{10} e = -0.86859$.

Κάπως έτσι δουλεύουμε όταν έχουμε εκθετικά και δυνάμεις. Δοκιμάστε και εσείς να παρουσιάσετε δεδομένα σε λογαριθμική κλίμακα. Θα σας φανεί χρήσιμη και η συνάρτηση logspace (αντίστοιχη της linspace).

Μερικές ακόμα χρήσιμες εντολές. Η `clf` καθαρίζει το τρέχων παράθυρο γραφικών από τη γραφική που έχει. Η `close` κλείνει το τρέχων παράθυρο γραφικών. Η `close all` κλείνει όλα τα ανοικτά παράθυρα γραφικών. Η `clear all` καθαρίζει τη μνήμη από όλες τις μεταβλητές.

1.9 Ο Editor και αρχεία script

Χρησιμοποιώντας το Octave σαν «κομπιουτεράκι» είναι βολικό μόνο για απλά προβλήματα. Το περιβάλλον του πακέτου αυτού είναι «άμεσο» (interpretive) που σημαίνει ότι κάθε εντολή που πληκτρολογούμε εκτελείται αμέσως. Επομένως αν έχουμε ήδη εκτελέσει κάποιο αριθμό εντολών και θέλουμε να αλλάξουμε κάποια παράμετρο σε κάποια προηγούμενη εντολή, αυτό σημαίνει ότι πρέπει να ξανατρέξουμε όλες τις ενδιάμεσες εντολές μια-μια. Βλέπουμε ότι χρειάζεται ένας καλύτερος τρόπος αλληλεπίδρασης. Αυτό επιτυγχάνεται όταν συγκεντρώνουμε τις εκτελέσιμες εντολές σε ένα αρχείο που συνηθίζεται να ονομάζεται script. Στην ουσία πρόκειται για αρχείο προγράμματος για το περιβάλλον Octave. Το Octave αντιστοιχεί την κατάληξη `.m` σε τέτοιου είδους αρχεία (όπως το MATLAB). Μπορούμε επίσης να δημιουργήσουμε script αρχεία που να παίρνουν στην είσοδο παραμέτρους και μεταβλητές και να δίνουν στην έξοδο αποτελέσματα. Αυτά είναι συναρτήσεις (functions).

Στο περιβάλλον Octave GUI κάνοντας κλικ στον Editor πληκτρολογούμε το script πρόγραμμά μας και το αποθηκεύουμε στο αρχείο tata.m (κουμπί Save File As στη λωρίδα εργαλείων του Editor). Το αρχείο είναι ορατό στον File Browser αριστερά. Κάνουμε κλικ στο Command Window και πληκτρολογούμε το όνομα του αρχείου χωρίς την κατάληξη. Το πρόγραμμα τρέχει και εμφανίζεται το αποτέλεσμα.

Το περιεχόμενο του tata.m είναι:

```
% Παράδειγμα κώδικα για δευτεροβάθμια εξίσωση
a=2; b=-1; c=3;            % οι συντελεστές
delta=b^2-4*a*c;           % η διακρίνουσα
x1=(-b+sqrt(delta))/(2*a); % η μια λύση
x2=(-b-sqrt(delta))/(2*a); % η άλλη λύση
x=[x1; x2]                 % και οι δυο λύσεις μαζί
```

και το αποτέλεσμα:

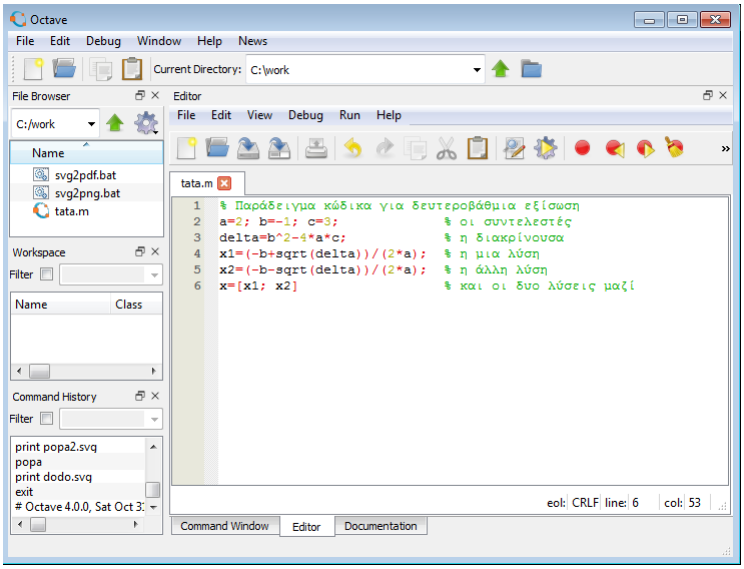
```
>> tata
x =
   0.2500 + 1.1990i
   0.2500 - 1.1990i
```

Στα σχ. 1.12 και 1.13 βλέπουμε εικόνες του editor και της κονσόλας (Command Prompt) με τα παραπάνω. Έστω τώρα ότι θέλουμε να αλλάξουμε τους συντελεστές. Αν πληκτρολογούσαμε τις παραπάνω γραμμές στην κονσόλα, η αλλαγή στην γραμμή 2 σημαίνει ότι πρέπει να επαναεισάγουμε και να ξανατρέξουμε όλες τις άλλες εντολές που ακολουθούν τη 2 (το περιβάλλον του Octave είναι «άμεσης ανταπόκρισης», interpretive, δηλ. όλες οι εντολές εκτελούνται αμέσως μετά την είσοδό τους). Φανταστείτε αν έχουμε δεκάδες ή εκατοντάδες τέτοιες εντολές πόσο κόπο παίρνει. Με τον editor δεν έχουμε τέτοιο πρόβλημα. Ότι αλλαγή κάνουμε, αυτόματα τρέχει όλο το πρόγραμμα. Επιπλέον, έχουμε το πρόγραμμα σε αποθηκευμένο αρχείο και μπορούμε να το πάρουμε σπίτι μας, να το τρέξουμε σε κάποια επόμενη συνεδρία ή να το εισάγουμε σε κάποιο πρόγραμμα επεξεργασίας κειμένου και να το παρουσιάσουμε σε εργασία μας.

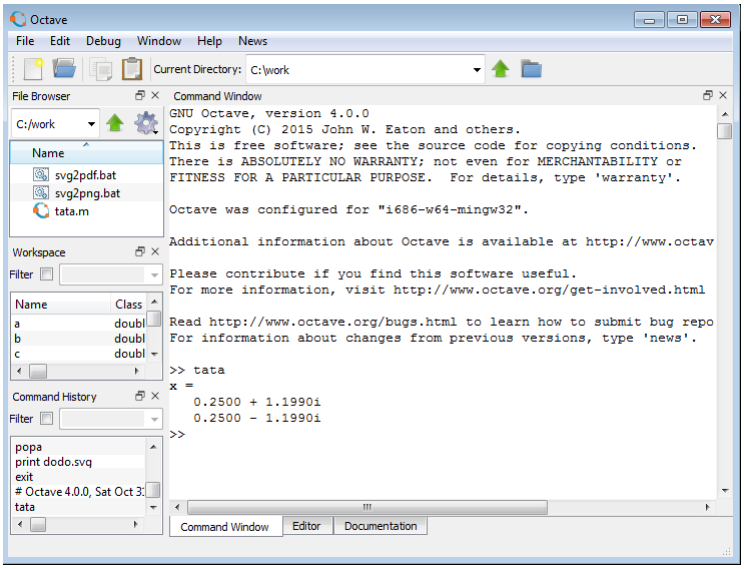
1.10 Functions

Το παραπάνω πρόβλημα μπορεί να γραφεί και σαν συνάρτηση (function). Γράφουμε τις παρακάτω εντολές

```
function [x] = trionimo(a, b, c)
% trionimo
% Βρίσκει ρίζες τριωνύμου ax^2+bx+c=0
% Οι ρίζες, πραγματικές ή μιγαδικές, είναι στο διάνυσμα x=[x1,x2]

  delta=b^2-4*a*c;
  x1=(-b+sqrt(delta))/(2*a);
  x2=(-b-sqrt(delta))/(2*a);
  x=[x1; x2];
end
```



Σχήμα 1.12: Απλό παράδειγμα κώδικα δευτεροβάθμιας στον editor. Διακρίνουμε τους αριθμούς εντολών στη στήλη αριστερά καθώς και τα σχόλια που μπορούν να είναι είτε στα Ελληνικά είτε στα Αγγλικά.



Σχήμα 1.13: Εικόνα της κονσόλας μετά το τρέξιμο του script όπου φαίνονται τα αποτελέσματα.

και τις αποθηκεύουμε στο αρχείο trionimo.m στο φάκελο εργασίας. Κατόπιν, μέσα από τη κονσόλα ή σε άλλο script μπορούμε να καλέσουμε την παραπάνω συνάρτηση σαν

```
>> x=trionimo(2,-1,3)
x =
   0.2500 + 1.1990i
   0.2500 - 1.1990i
```

Αυτό γενικεύεται και σε περισσότερες συναρτήσεις και βοηθά πολύ σε μεγαλύτερα προγράμματα. Αρχεία συναρτήσεων m αρχίζουν με τη λέξη function. Ακολουθεί η μεταβλητή ή μεταβλητές εξόδου. Το σύμβολο =. Το όνομα της συνάρτησης, το οποίο πρέπει να είναι ίδιο με το όνομα του αρχείου. Και οι μεταβλητές εισόδου σαν ορίσματα της συνάρτησης. Αμέσως μετά τον ορισμό της συνάρτησης είναι καλό να υπάρχουν σχόλια που να εξηγούν σκοπό και χρήση. Αυτά μπορούν να εμφανιστούν στην κονσόλα αν πληκτρολογήσετε help και το όνομα του αρχείου m που περιέχει τη συνάρτηση.

```
>> help trionimo
'trionimo' is a function from the file C:\work\trionimo.m

 trionimo
 Βρίσκει ρίζες τριωνύμου ax^2+bx+c=0
```

```
 Οι ρίζες, πραγματικές ή μιγαδικές, είναι στο διάνυσμα x=[x1,x2]

Additional help for built-in functions and operators is
available in the on-line version of the manual.  Use the command
‘doc <topic>’ to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.octave.org and via the help@octave.org
mailing list.
```

Είναι πιθανόν να μην εμφανίζονται τα Ελληνικά στο συστημά σας. Μπορείτε όμως να τα διαβάσετε στον Editor και να θυμηθείτε τι κάνατε «κάποτε» με αυτό το πρόγραμμα.

Στο τέλος της συνάρτησης μην ξεχνάτε την εντολή end.

Αντί να έχουμε ξεχωριστό αρχείο για τη συνάρτησή μας και να έχουμε διαφορετικό πρόγραμμα να την καλεί, μπορούμε να τα ενσωματώσουμε όλα μαζί σε ένα αρχείο φτάνει να έχουμε την εντολή `clear all` στην αρχή.

```
% tritest.m

clear all

function [x] = trionimo(a, b, c)
  delta=b^2-4*a*c;
  x1=(-b+sqrt(delta))/(2*a);
  x2=(-b-sqrt(delta))/(2*a);
  x=[x1; x2];
end

x=trionimo(2,-1,3)
```

με αποτέλεσμα

```
x =
   0.2500 + 1.1990i
   0.2500 - 1.1990i
```

Συνηθίζεται στην αρχή κάθε προγράμματος script να έχουμε τις εντολές

```
clear all; close all;
```

έτσι ώστε να καθαρίζεται η μνήμη από προηγούμενα αποτελέσματα και να κλείνουν όλα τα προηγούμενα παράθυρα γραφικών.

Σε σύνθετα προβλήματα με πολλές γραμμές κώδικα είναι πολύ εύκολο να γίνει κάποιο λάθος (τις πιο πολλές φορές τυπογραφικό). Μπορούμε τότε να διακόπτουμε το πρόγραμμα με την εντολή `pause` και να επιβεβαιώνουμε ότι το πρόγραμμα δουλεύει εντάξει μέχρι εκείνο το σημείο. Οι ενέργειες αυτές αποτελούν ένα πρώτο στάδιο debugging. Συνιστάται όταν γράφετε ένα σύνθετο πρόγραμμα να το κάνετε σε μικρά στάδια, τρέχοντάς το ενδιάμεσα με έλεγχο των ενδιαμέσων αποτελεσμάτων. Έτσι είναι πιο εύκολο να εντοπιστούν σφάλματα.

Παράδειγμα 1.1 *Φτιάξτε συνάρτηση με όνομα odd_index που παίρνει είσοδο πίνακα Min και επιστρέφει πίνακα Mout που περιέχει μόνο τις περιττές γραμμές και στήλες του Min.*

Απάντηση:

```
function Mout = odd_index(Min)
  Mout = Min(1:2:end,1:2:end);
end
```

Φτιάχνουμε τον Mout παίρνοντας γραμμές και στήλες ξεκινώντας από το 1 με βήμα 2 μέχρι το τέλος. Μπορούμε να αποθηκεύσουμε τον παραπάνω κώδικα σε αρχείο με όνομα odd_index.m. Ελέγχουμε τη σωστή εκτέλεση φτιάχοντας το script, π.χ. tata.m όπου η randi είναι συνάρτηση που δίνει τυχαίους ακέραιους:

```
M = randi(5,6)
Mout = odd_index(M)
```

και έχουμε την απάντηση:

```
M =
        3        3        3        4        4        5
        1        5        4        3        5        5
        5        1        2        3        4        3
        2        3        2        1        1        5
        5        5        2        5        3        3
        3        1        3        1        4        3
Mout =
        3        3        4
        5        2        4
        5        2        3
```

Παράδειγμα 1.2 *Φτιάξτε συνάρτηση με όνομα mean_squares που παίρνει είσοδο έναν θετικό ακέραιο nn και επιστρέφει το άθροισμα των τετραγώνων από το 1 έως το nn.*

Απάντηση:

Ένας τρόπος:

```
function mm = mean_squares(nn)
  mm = mean((1:nn).^2);
end
```

Δεύτερος:

```
function mm = mean_squares(nn)
  x = 1:nn;
  sum = 0;
  for i=1:nn
    sum = sum + i^2;
  end
  mm = sum/length(x);
end
```

Παράδειγμα 1.3 *Φτιάξτε συνάρτηση με όνομα mtable που παίρνει είσοδο δυο θετικούς ακεραίους και επιστρέφει τον πίνακα πολ/σμού των. Δηλ. το στοιχείο π.χ. i,j του πίνακα επιστροφής ισούται με* `i*j`.

Απάντηση:

```
function [mt] = mtable(n,m)
  mt =  (1:n)' * (1:m);
end
```

με δοκιμή

```
>> mt = mtable(5,4)
mt =
        1        2        3        4
        2        4        6        8
        3        6        9       12
        4        8       12       16
        5       10       15       20
```

1.11 Anonymous functions

Υπάρχει επίσης και η δυνατότητα για «ανώνυμες» συναρτήσεις όπως:

```
f = @(x) x.^3;

x=0:0.1:2;
plot(x,f(x))
```

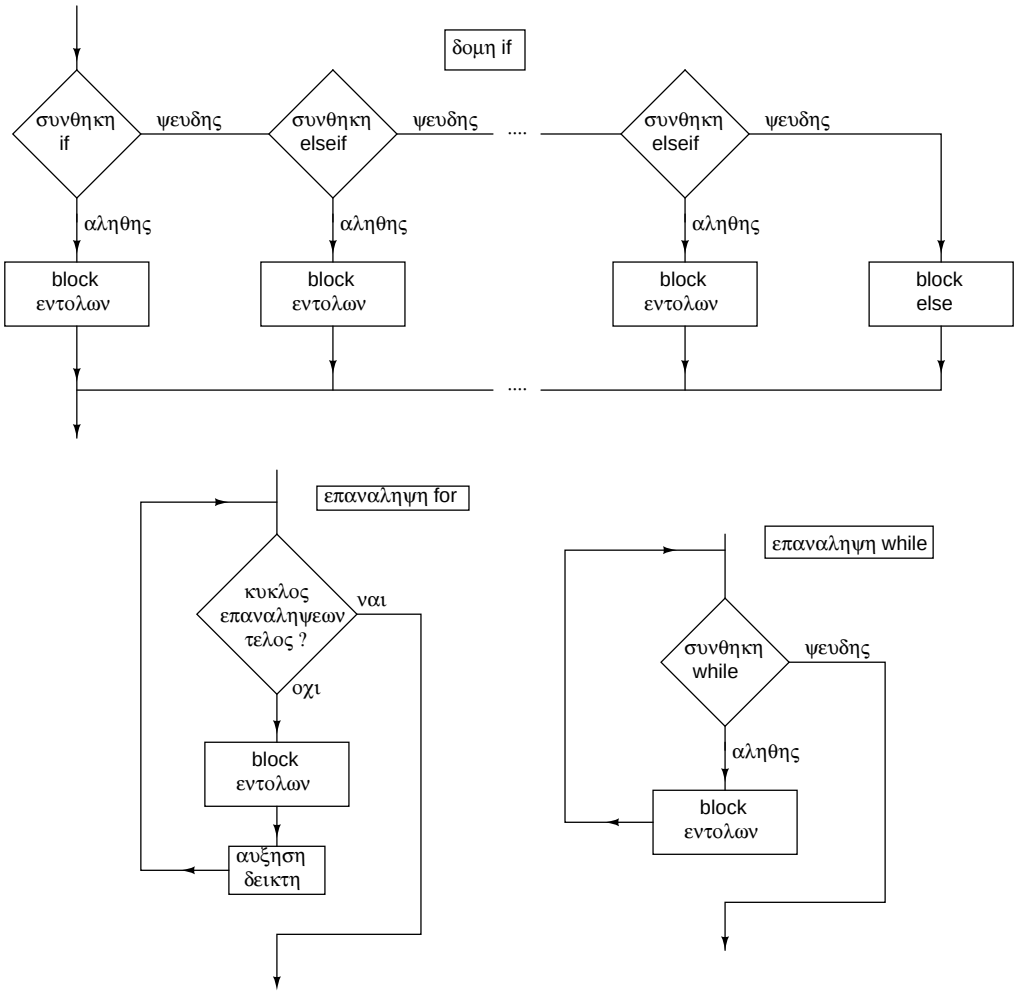
ή

```
f = @(x,y) x.^3+6*sin(y);

x=0:0.1:2;
y=5:0.2:9;
z=f(x,y);
plot(x,z)
```

με γενίκευση και σε παραπάνω μεταβλητές ή παραμέτρους. Ο ορισμός αυτός είναι ιδιαίτερα χρήσιμος για μικρές συναρτήσεις που χρειάζονται απλώς μια γραμμή.

1.12 Εντολές επιλογής και επανάληψης

Το Octave όπως κάθε περιβάλλον και γλώσσα προγραμματισμού περιέχει εντολές επιλογής και επανάληψης.



Σχήμα 1.14: Block διαγράμματα της δομής if, και των επαναλήψεων for και while. Στη δομή if τα else(if) blocks υλοποιούνται μόνο αν το απαιτεί το συγκεκριμένο πρόβλημα.

1.12.1 if, else, elseif

Η κλασσική εντολή επιλογής είναι η if. Εάν κάποια έκφραση, σχέση ή παράσταση είναι αληθής τότε εκτελούνται μια ή περισσότερες συγκεκριμένες εντολές. Αλλιώς, εκτελούνται άλλες.

Η δομή της απλής εντολής if είναι:

if (*expression*)
  statements
end

όπου *expression* είναι μια έκφραση με λογική τιμή αληθή, True, (1) ή ψευδή, False, (0) και *statements* είναι μια ή περισσότερες εντολές ή εκφράσεις που εκτελούνται ή μη ανάλογα με το αν η λογική τιμή είναι αληθής ή ψευδής αντίστοιχα.

Η εντολή if γενικεύεται:

if (*expression*)
  statements
elseif (*expression*)
...
else
  statements
end

όπου οι δυνατές επιλογές είναι περισσότερες. Η τελευταία ομάδα εντολών (statements) με το else εκτελείται όταν όλες οι προηγούμενες συνθήκες είναι ψευδείς.

Οι λογικοί τελεστές σύγκρισης του Octave φαίνονται στον Πίνακα 1.5 και είναι παρόμοιοι με αντίστοιχους τελεστές σε άλλες γλώσσες προγραμματισμού.

Πίνακας 1.5: Οι λογικοί τελεστές σύγκρισης του Octave

`<`	μικρότερο από
`<=`	μικρότερο ή ίσο από
`>`	μεγαλύτερο από
`>=`	μεγαλύτερο ή ίσο από
`==`	ίσο με
`~`	λογικό NOT
`~=`	όχι ίσο με
`&`	λογικό AND στοιχείο προς στοιχείο για πίνακες και διανύσματα
`|`	λογικό OR στοιχείο προς στοιχείο για πίνακες και διανύσματα
`&&`	λογικό AND μόνο για στοιχεία - όχι πίνακες και διανύσματα
`||`	λογικό OR μόνο για στοιχεία - όχι πίνακες και διανύσματα

Ένα απλό παράδειγμα γνωστό σε όλους είναι η επίλυση ενός τριωνύμου. Έστω το τριώνυμο $x^2 - 8x + 15$. Το παρακάτω πρόγραμμα βρίσκει τις λύσεις του ανάλογα με το πρόσημο της διακρίνουσας.

```
a=1; b=-8; c=15;
d=b^2-4*a*c
if (d >= 0)
  x1 = (-b+sqrt(d))/(2*a)
  x2 = (-b-sqrt(d))/(2*a)
else
  x1 = (-b+j*sqrt(abs(d)))/(2*a)
  x2 = (-b-j*sqrt(abs(d)))/(2*a)
end
```

Το αποτέλεσμα όταν τρέχουμε το παραπάνω πρόγραμμα είναι

```
d =  4
x1 =  5
x2 =  3
```

Η διακρίνουσα είναι θετική, άρα εκτελούνται οι δυο πρώτες εντολές και υπολογίζονται οι πραγματικές ριζες 5 και 3.

Έστω τώρα το τριώνυμο $x^2 - 6x + 34$. Η μόνη αλλαγή στο πρόγραμμα είναι οι διαφορετικοί συντελεστές του `a=1; b=-6; c=34;`. Το αποτέλεσμα είναι

```
d = -100
x1 =  3 + 5i
x2 =  3 - 5i
```

Η διακρίνουσα είναι αρνητική, άρα εκτελούνται οι δυο τελευταίες εντολές και υπολογίζονται οι μιγαδικές ριζες $3 + j5$ και $3 - j5$. Επισημαίνεται βέβαια ότι αντίθετα με κλασσικές γλώσσες προγραμματισμού (Pascal, C) το Octave δεν έχει πρόβλημα με αρνητική διακρίνουσα και αυτόματα κάνει το αποτέλεσμα μιγαδικό αριθμό. Το παράδειγμα του τριωνύμου όμως αποτελεί μια καλή επίδειξη της δομής if και γιαυτό το σκοπό χρησιμοποιείται.

1.12.2 while και do until

Η δομή while

while (*expression*)
  statements
end

εκτελεί την ομάδα (block) εντολών (statements) εφόσον η συνθήκη expression είναι αληθής. Π.χ.

```
octave> n=5
n =  5
octave> while n > 2
> disp(n)
> n = n - 1
> end
 5
n =  4
 4
n =  3
 3
n =  2
octave>
```

Στην είσοδο του while η τιμή της μεταβλητής n είναι 5. Η ομάδα εντολών της while εκτυπώνει την τιμή της μεταβλητής (`disp(n)`) και την μειώνει κατά 1. Όταν η λογική παράσταση δεν είναι πλέον αληθής, έχουμε έξοδο από τη δομή. Προφανώς πρέπει να φροντίσουμε έτσι ώστε η τιμή της λογικής παράστασης να μην είναι πάντα αληθής (ξεχνώντας π.χ. την ελάττωση κατά 1). Γιατί;

Αν η παράσταση είναι εξ αρχής ψευδής (π.χ. `n=0`) τότε το while δεν εκτελείται ποτέ.

Σε περίπτωση που θέλουμε η δομή να εκτελεστεί τουλάχιστον μια φορά (π.χ. η τιμή της λογικής παράστασης εξαρτάται από κάτι μέσα στην ομάδα εντολών της δομής), η δομή τροποποιείται στην `do - until`

do
  statements
until (*expression*)

όπου η σύγκριση μετατίθεται στο τέλος και επαναλαμβάνεται μόνον όταν είναι αληθής.

1.12.3 for

Στην επανάληψη `for` η δομή είναι

for δείκτης = αρχή:βήμα:τέλος
  statements
end

όπου γνωρίζουμε εξαρχής πόσες φορές θα γίνει επανάληψη. Αν παραλείψουμε το βήμα, εννοείται η τιμή 1. Π.χ.

```
octave> for i=-2:3
> disp(i)
> end
-2
-1
0
 1
 2
 3
octave>
```

1.13 Λογική πρόσβαση σε στοιχεία

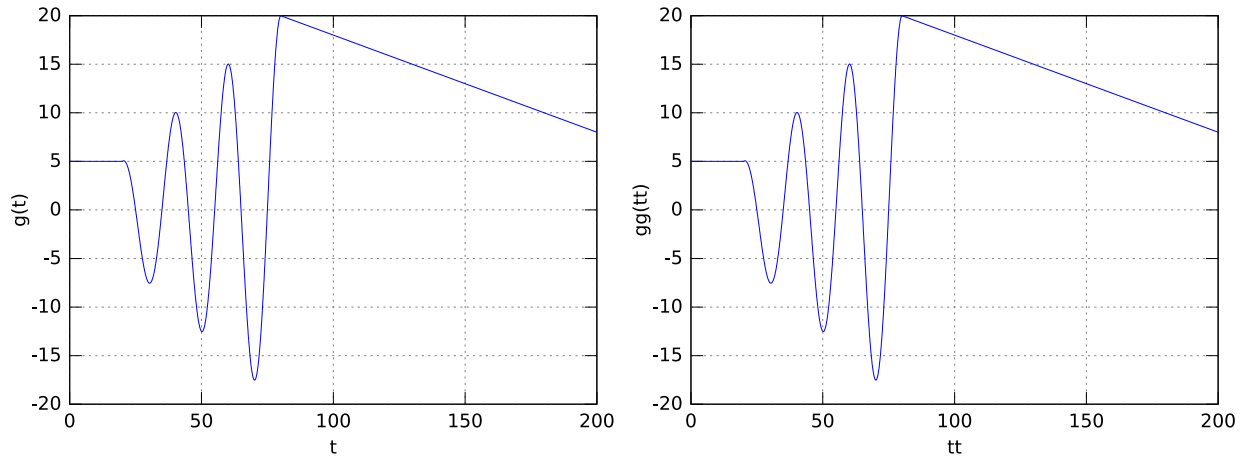
Συχνά είναι αναγκαίο να ορίσουμε ή να έχουμε πρόσβαση σε στοιχεία ενός διανύσματος ή πίνακα που ικανοποιούν κάποιες συνθήκες. Ο κλασσικός τρόπος που προέρχεται από κλασσικές γλώσσες προγραμματισμού είναι να κατασκευάσουμε ένα

βρόγχο επανάληψης και να εξετάσουμε στοιχείο προς στοιχείο το εν λόγω διάνυσμα ή πίνακα με κάποια λογική εντολή αν ικανοποιείται ή όχι η συνθήκη που μας ενδιαφέρει.

Για παράδειγμα, θέλουμε να κωδικοποιήσουμε την παρακάτω συνάρτηση:

$$g(t) = \begin{cases} 5 & \text{για} \quad 0 \leq t < 20 \\ \frac{t}{4}\cos\left(\frac{2\pi t}{20}\right) & \text{για} \quad 20 \leq t < 80 \\ 28 - \frac{t}{10} & \text{για} \quad 80 \leq t < 200 \end{cases}$$

Με τον κλασσικό τρόπο ορίζουμε πόσα σημεία `N` θέλουμε και ένα διάνυσμα `t` που καλύπτει ομοιόμορφα το εύρος τιμών που θέλουμε. Κατόπιν κατασκευάζουμε ένα for loop όπου εξετάζουμε μια-μια τις τιμές του `t` αν βρίσκονται στο εύρος που μας ενδιαφέρει και ορίζουμε το αντίστοιχο τμήμα συνάρτησης $g(t)$. Ο τρόπος που το κάνουμε είναι: ξεκινώντας με ένα κενό διάνυσμα `g` το γεμίζουμε σημείο προς σημείο με τον αντίστοιχο ορισμό του. Ο παρακάτω κώδικας υλοποιεί αυτή τη διαδικασία.



Σχήμα 1.15: Η συνάρτηση $g(t)$ με τον κλασσικό τρόπο (αριστερά) και τον βέλτιστο τρόπο (δεξιά).

```
% Αρχικοποιήσεις και ορισμός συνάρτησης
N=2048;
T=200;
t=linspace(0,T,N+1);
g=[];

for tk=t
  if tk<20
    g = [g 5];
  elseif tk < 80
    g = [g (tk/4)*cos(2*pi*tk/20)];
  else
    g = [g 28-tk/10];
  end
end

plot(t,g); grid; xlabel('t'); ylabel('g(t)');
```

και το αποτέλεσμα είναι στο σχ. 1.15 αριστερά. Ο παραπάνω τρόπος δεν είναι ο βέλτιστος. Φανταστείτε να έχετε μεγάλα διανύσματα ή πίνακες με μερικές χιλιάδες στοιχεία. Πρόσβαση στοιχείο προς στοιχείο θα πάρει πολύ χρόνο. Τι άλλο μπορούμε να κάνουμε; Μπορούμε να χρησιμοποιήσουμε λογικές συνθήκες μέσα στα ίδια τα διανύσματα ή πίνακες και να αποφύγουμε βρόγχους επανάληψης.

Για παράδειγμα, έστω ότι έχουμε ένα διάνυσμα με τυχαίες τιμές, θετικές και αρνητικές και θέλουμε να ξεχωρίσουμε τις θετικές από τις αρνητικές.

```
x = [-2.86188  4.52112  -1.61456  -8.02945  4.02805  -4.98783  1.56886  3.45225]
isNegative = (x<0)
isPositive = (x>0)
```

```
x(isNegative)
x(x<0)
x(isPositive)
x(x>0)

Αποτέλεσμα

x =
  -2.86188  4.52112  -1.61456  -8.02945  4.02805  -4.98783  1.56886  3.45225

isNegative =
        1        0        1        1        0        1        0        0

isPositive =
        0        1        0        0        1        0        1        1

ans =
  -2.86188  -1.61456  -8.02945  -4.98783

ans =
  -2.86188  -1.61456  -8.02945  -4.98783

ans =
  4.52112  4.02805  1.56886  3.45225

ans =
  4.52112  4.02805  1.56886  3.45225
```

Οι εντολές `isNegative` και `isPositive` ορίζουν λογικά διανύσματα με τιμή 1 αν ικανοποιείται κάποια συνθήκη και 0 αν όχι. Τα λογικά αυτά διανύσματα μπορούν να μπουν σαν όρισμα στο διάνυσμα `x`. Αυτόματα εξαλείφονται τα στοιχεία με όρισμα ψευδές (λογική τιμή 0) και εξάγουμε έτσι τα στοιχεία του x που ικανοποιούν τη συνθήκη. Μάλιστα δεν χρειάζεται ξεχωριστή εντολή όπως οι `isNegative` και `isPositive`. Η λογική συνθήκη μπορεί κατευθείαν να μπεί στο όρισμα, `x(x<0)`, `x(x>0)`.

Σύμφωνα με τα παραπάνω, ο βέλτιστος κώδικας που ορίζει την αρχική συνάρτηση $g(t)$ είναι:

```
% Αρχικοποιήσεις και ορισμός συνάρτησης
N=2048;
T=200;
t=linspace(0,T,N+1);

t1 = t (t < 20);
t2 = t ((20 <= t) & (t < 80));
t3 = t (t >= 80);
g1 = 5*ones(1,length(t1));
g2 = (t2/4).*cos(2*pi*t2/20);
g3 = 28-t3/10;
tt = [t1 t2 t3];  % ίδιο με το t
gg = [g1 g2 g3];  % ίδιο με το g

plot(tt,gg); grid; xlabel('tt'); ylabel('gg(tt)');
```

και το αποτέλεσμα είναι στο σχ. 1.15 δεξιά. Βλέπουμε ότι δεν υπάρχει διαφορά στο αποτέλεσμα με τους δυο τρόπους. Απλώς ο βέλτιστος είναι ταχύτερος και πιο απλός στην υλοποίησή του.

1.14 Μέθοδος Newton

Ένα άλλο παράδειγμα εφαρμογής είναι η αριθμητική μέθοδος εύρεσης ριζών συναρτήσεως του Newton. Σύμφωνα με τη μέθοδο, ξεκινάμε με κάποια τιμή της ανεξάρτητης μεταβλητής που είναι κοντά στην πραγματική. Φέρνουμε την εφαπτομένη σε εκείνο το σημείο και βρίσκουμε τη ρίζα της εφαπτομένης (σημείο τομής εφαπτομένης με άξονα x). Η ρίζα αυτή συνήθως είναι πιο κοντά στην πραγματική ρίζα και έτσι επαναλαμβάνουμε τα ίδια μέχρις ότου οι επαναλήψεις οδηγήσουν σε μια τιμή που διαφέρει ελάχιστα από την πραγματική ρίζα. Μαθηματικά, κάθε επανάληψη δίνει

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

και σταματάμε όταν $|x_{n+1} - x_n| < \epsilon$ όπου ϵ είναι ένα όριο που θέτουμε εμείς.

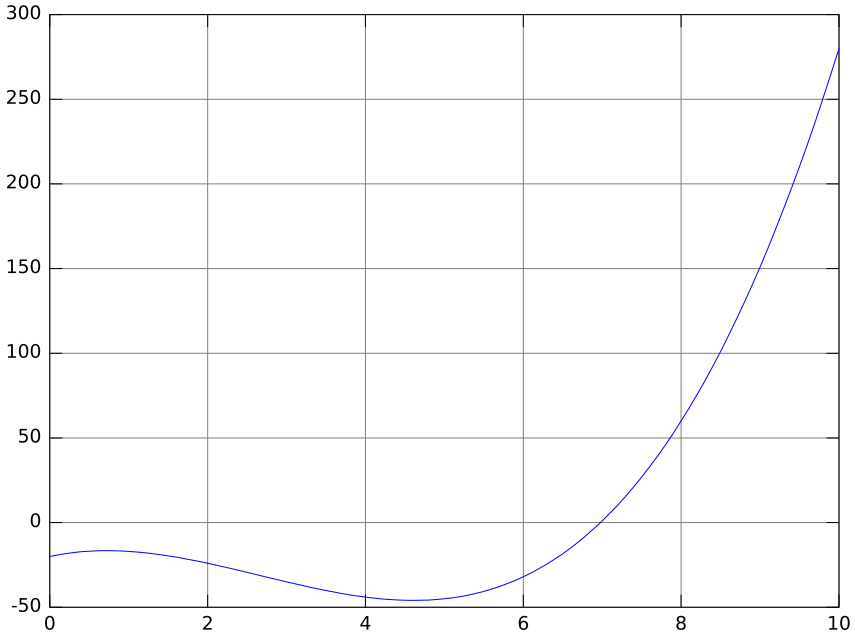
Στο παράδειγμα που ακολουθεί παίρνουμε σαν συνάρτηση την $f(x) = x^3 - 8x^2 + 10x - 20$ με παράγωγο $f'(x) = 3x^2 - 16x + 10$ οπότε ο αλγόριθμος είναι

$$x_{n+1} = x_n - \frac{x_n^3 - 8x_n^2 + 10x_n - 20}{3x_n^2 - 16x_n + 10}$$

και σταματάμε για $\epsilon = 10^{-6}$. Μια γρήγορη γραφική παράσταση

```
x = 0:0.1:10;
y = -20+10*x-8*x.^2+x.^3;
plot(x,y); grid
```

δείχνει ότι μια πραγματική ρίζα είναι κοντά στο 7 και διαλέγουμε αυθαίρετα σαν αρχική τιμή το 5.



Σχήμα 1.16: Πρόχειρη γραφική παράσταση για να δούμε που περίπου βρίσκεται η ρίζα.

Ο παρακάτω κώδικας και αποτέλεσμα

```
clear all; close all;

function [fun] = f(x)
  fun = -20 + 10*x - 8*x.^2 + x.^3;
end

function [funp] = fp(x)
  funp = 10 - 16*x + 3*x.^2;
end

x0 = 5;
for i=0:10
  x1 = x0 - f(x0)./fp(x0);
  eps = abs(x1-x0);
  % για καλύτερη εμφάνιση αποτελεσμάτων
  fprintf('%10d %10.6f %10.6f\n',i,x1,eps)
  x0 = x1;
end
```

```
         0  14.000000   9.000000
         1  10.534759   3.465241
         2   8.432209   2.102550
         3   7.356840   1.075369
         4   7.013652   0.343188
         5   6.978005   0.035647
         6   6.977634   0.000372
         7   6.977634   0.000000
         8   6.977634   0.000000
         9   6.977634   0.000000
        10   6.977634   0.000000
```

δείχνουν ότι ήδη με την 7η επανάληψη έχουμε επιτύχει τον σκοπό μας.

Ο αντίστοιχος κώδικας με while είναι

```
clear all; close all;

function [fun] = f(x)
  fun = -20 + 10*x - 8*x.^2 + x.^3;
end

function [funp] = fp(x)
  funp = 10 - 16*x + 3*x.^2;
end

x0 = 5;
eps = 1e-6;
dif = 10; % για να αρχίσει η επανάληψη
i=0;
while dif>eps
  x1 = x0 - f(x0)./fp(x0);
  dif = abs(x1-x0);
  fprintf('%10d %10.6f %10.6f\n',i,x1,dif)
  x0 = x1;
  i = i+1;
end
```

```
         0  14.000000   9.000000
         1  10.534759   3.465241
         2   8.432209   2.102550
         3   7.356840   1.075369
         4   7.013652   0.343188
         5   6.978005   0.035647
         6   6.977634   0.000372
         7   6.977634   0.000000
```

Εδώ, στην αρχή της επανάληψης ελέγχεται αν η συνθήκη `dif>eps` είναι αληθής. Για να ξεκινήσει η επανάληψη δώσαμε αυθαίρετα μια μεγάλη αρχική τιμή στην μεταβλητή `dif` έτσι ώστε η συνθήκη να αληθεύει. Προσοχή εδώ, αν αντί για `dif` είχαμε απευθείας τη σχέση `abs(x1-x0)` τότε μετά την πρώτη επανάληψη η τιμή θα ήταν μηδέν λόγω της `x0=x1` και δεν θα έτρεχε το πρόγραμμα.

Και η παραλλαγή με την do until

```
clear all; close all;

function [fun] = f(x)
  fun = -20 + 10*x - 8*x.^2 + x.^3;
end

function [funp] = fp(x)
  funp = 10 - 16*x + 3*x.^2;
end

x0 = 5;
eps = 1e-6;
i=0;
do
  x1 = x0 - f(x0)./fp(x0);
  dif = abs(x1-x0);
  fprintf('%10d %10.6f %10.6f\n',i,x1,dif)
  x0 = x1;
  i = i+1;
until dif < eps
```

```
         0  14.000000   9.000000
         1  10.534759   3.465241
         2   8.432209   2.102550
         3   7.356840   1.075369
         4   7.013652   0.343188
         5   6.978005   0.035647
         6   6.977634   0.000372
         7   6.977634   0.000000
```

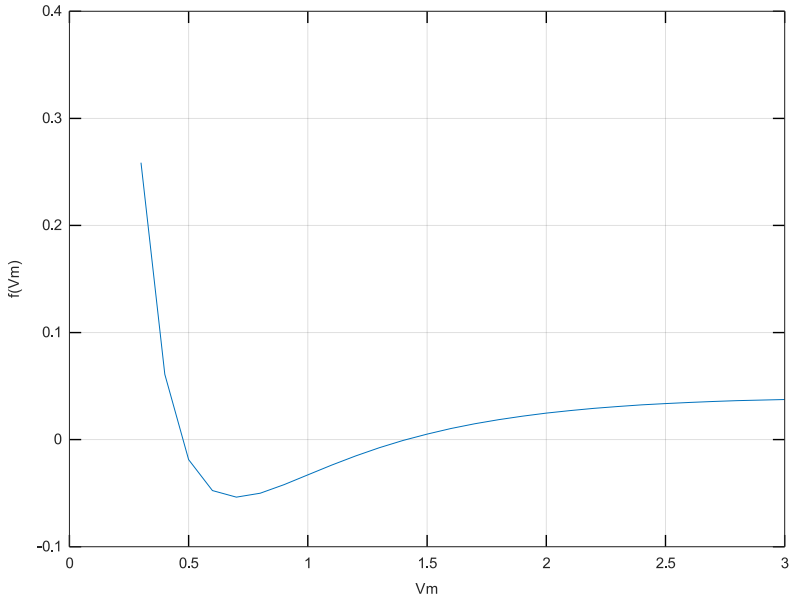
όπου η λογική έκφραση βρίσκεται στο τέλος (άρα η ενδιάμεση δομή εκτελείται τουλάχιστον μια φορά). Η τελευταία δομή είναι ίσως η πιο κατάλληλη για το συγκεκριμένο πρόβλημα.

Στα παραπάνω βλέπουμε ότι μόνο 7 επαναλήψεις αρκούν για να βρούμε τη ρίζα με την επιθυμητή ακρίβεια. Φυσικά, στην πράξη, αν θέλαμε όλες τις ρίζες του πολυωνύμου θα χρησιμοποιούσαμε την συνάρτηση `roots`.

```
c = [1 -8 10 -20];
roots(c)
   6.97763 + 0.00000i
   0.51118 + 1.61400i
   0.51118 - 1.61400i
```

και η πραγματική ρίζα είναι η πρώτη. Η μέθοδος όμως Newton ισχύει και για μη πολυωνυμικές συναρτήσεις και αποτελεί ένα καλό παράδειγμα για δομές επανάληψης.

Παράδειγμα 1.4 *Φτιάξτε ανώνυμη συνάρτηση για την* $f(V_m) = 1/(3V_m) - (5.5/6)\Big[1 - \exp(-1/V_m)\Big]^2$ *όπου η* V_m *μπορεί να είναι αριθμός ή διάνυσμα. Κάντε την γραφική της στο διάστημα* $0.3 \leq V_m \leq 3$*. Πόσες ρίζες φαίνεται να έχει;*



Σχήμα 1.17: Η γραφική της ανώνυμης $f(V_m)$

Την συνάρτηση και τη γραφική της την υλοποιούμε ως εξής:

```
clear all; close all;

f = @(Vm)  (1/3)./Vm - (5.5/6)*(1-exp(-1./Vm)).^2;

Vm = 0.3:0.1:3;
plot(Vm,f(Vm)); grid; xlabel('Vm'); ylabel('f(Vm)');
```

Φαίνεται να έχει 2 ρίζες, μια γύρω στο 0.5 και την άλλη γύρω στο 1.5.

1.15 Μέθοδος Διχοτόμησης (bisection)

Μια άλλη μέθοδος εύρεσης ριζών είναι η μέθοδος Διχοτόμησης (bisection). Είναι πιο απλή και σταθερή, δεν χρειάζεται παραγώγους αλλά είναι πιο αργή. Βασίζεται στο θεώρημα της ενδιάμεσης τιμής όπου αν μια συνάρτηση $f(x)$ έχει ετερόσημες τιμές για κάποιο διάστημα $[a, b]$ τότε υπάρχει ρίζα $x^* \in [a, b]$. Η ρίζα αυτή μπορεί να βρεθεί με διαδοχικές διχοτομήσεις του $[a, b]$ όπου μετά από n διχοτομήσεις το διάστημα που ισχύει το θεώρημα της ενδιάμεσης τιμής είναι τόσο μικρό όσο η επιθυμητή ακρίβεια ϵ για τη ζητούμενη ρίζα.

Ο αλγόριθμος είναι:

1. Έστω πραγματική συνάρτηση $f(x)$ σε διάστημα $[a, b]$ όπου ισχύει το θεώρημα ενδιάμεσης τιμής $f(a) \cdot f(b) < 0$.
2. Διχοτομούμε το διάστημα σε $[a, x_k]$, $[x_k, b]$ όπου $x_k = (a + b)/2$.
3. Ελέγχουμε για ποιο από τα δυο διαστήματα ισχύει ακόμα το θεώρημα ενδιάμεσης τιμής, γιατί μέσα σε αυτό θα βρίσκεται η ρίζα και αντικαθιστούμε το αρχικό $[a, b]$ με αυτό.
4. Εάν $|b - a| < \epsilon$ τότε η επιθυμητή ρίζα είναι το x_k αλλιώς επαναλαμβάνουμε από το βήμα 2 έως ότου $|b - a| < \epsilon$.

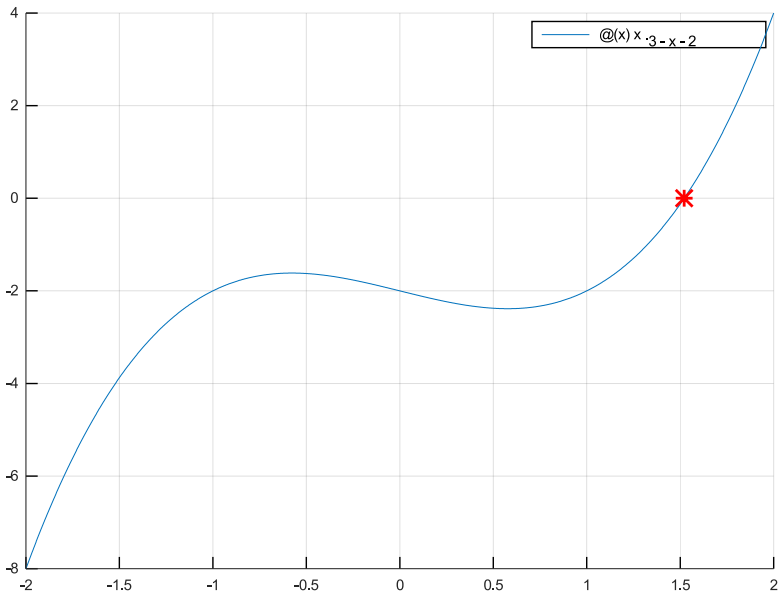
Ένα παράδειγμα για την συνάρτηση $f(x) = x^3 - x - 2$ είναι:

```
clear all; close all;
```

```
function y = bisection(f, a, b, tolerance, max_iter)
% Συνάρτηση bisection για ευρεση ρίζας μεσω διχοτόμησης
% f: η συνάρτηση
% a,b: το διάστημα όπου αναζητείται η ρίζα
% tolerance: η επιθυμιτή ακρίβεια, τυπικά 1e-5 ή 1e-6
% max_iter: ο μέγιστος αριθμός επαναλήψεων

  if(f(a) * f(b) >= 0)
    error("Error: f(a) * f(b) >= 0")
  end

  n_iter = 0;

  printf("\nIter       a               b               c              f(c)\n");
  printf("-------------------------------------------------------------------\n");

  while(n_iter < max_iter)
    c = (a + b) / 2;

    printf("%d \t", n_iter + 1);
    printf("%f \t", a);
    printf("%f \t", b);
    printf("%f \t", c);
    printf("%f \n", f(c));

    y = c;

    if(f(c) == 0 || (b - a) / 2 < tolerance)
      return
    end

    n_iter += 1;

    if(f(c) * f(a) > 0)
      a = c;
    else
      b = c;
    end
  end
end


hold on
fn = @(x) x.^3 - x - 2
fplot(fn, [-2, 2]); grid;
root = bisection(fn, 1, 2, 1e-5, 30)
plot(root, fn(root),"*", "color", "r")
hold off


fn =
@(x) x .^ 3 - x - 2

Iter       a               b               c              f(c)
-------------------------------------------------------------------
1         1.000000         2.000000         1.500000         -0.125000
2         1.500000         2.000000         1.750000         1.609375
3         1.500000         1.750000         1.625000         0.666016
4         1.500000         1.625000         1.562500         0.252197
5         1.500000         1.562500         1.531250         0.059113
6         1.500000         1.531250         1.515625         -0.034054
7         1.515625         1.531250         1.523438         0.012250
8         1.515625         1.523438         1.519531         -0.010971
9         1.519531         1.523438         1.521484         0.000622
10         1.519531         1.521484         1.520508         -0.005179
11         1.520508         1.521484         1.520996         -0.002279
12         1.520996         1.521484         1.521240         -0.000829
13         1.521240         1.521484         1.521362         -0.000103
14         1.521362         1.521484         1.521423         0.000259
15         1.521362         1.521423         1.521393         0.000078
16         1.521362         1.521393         1.521378         -0.000013
17         1.521378         1.521393         1.521385         0.000033
root =  1.5214
```

Παράδειγμα 1.5 *Για την συνάρτηση* $f(V_m) = 1/(3V_m) - (5.5/6)\Big[1 - \exp(-1/V_m)\Big]^2$ *στο διάστημα* $0.3 \leq V_m \leq 3$ *εφαρμόστε τη μέθοδο Διχοτόμησης να βρείτε τις ρίζες της.*

Από το προηγούμενο παράδειγμα είδαμε στη γραφική ότι έχει δυο ρίζες. Μια κοντά στο 0.5 και την άλλη κοντά στο 1.5. Οπότε:



Σχήμα 1.18: Η γραφική της $f(x) = x^3 - x - 2$ και η ρίζα της

```
clear all; close all;
f = @(Vm)  (1/3)./Vm - (5.5/6)*(1-exp(-1./Vm)).^2;
root1 = bisection(f, 0.1, 1, 1e-5, 30)
root2 = bisection(f, 1, 2, 1e-5, 30)
```

```
Iter       a               b               c              f(c)
-----------------------------------------------------------------
1         0.100000         1.000000         0.550000         -0.037171
2         0.100000         0.550000         0.325000         0.191544
3         0.325000         0.550000         0.437500         0.022209
4         0.437500         0.550000         0.493750         -0.015609
5         0.437500         0.493750         0.465625         0.000777
6         0.465625         0.493750         0.479688         -0.007976
7         0.465625         0.479688         0.472656         -0.003748
8         0.465625         0.472656         0.469141         -0.001523
9         0.465625         0.469141         0.467383         -0.000383
10         0.465625         0.467383         0.466504         0.000195
11         0.466504         0.467383         0.466943         -0.000095
12         0.466504         0.466943         0.466724         0.000050
13         0.466724         0.466943         0.466833         -0.000022
14         0.466724         0.466833         0.466779         0.000014
15         0.466779         0.466833         0.466806         -0.000004
16         0.466779         0.466806         0.466792         0.000005
17         0.466792         0.466806         0.466799         0.000000
root1 =  0.46680

Iter       a               b               c              f(c)
-----------------------------------------------------------------
1         1.000000         2.000000         1.500000         0.005190
2         1.000000         1.500000         1.250000         -0.011302
3         1.250000         1.500000         1.375000         -0.002377
4         1.375000         1.500000         1.437500         0.001569
5         1.375000         1.437500         1.406250         -0.000362
6         1.406250         1.437500         1.421875         0.000614
7         1.406250         1.421875         1.414062         0.000128
8         1.406250         1.414062         1.410156         -0.000116
9         1.410156         1.414062         1.412109         0.000006
10         1.410156         1.412109         1.411133         -0.000055
11         1.411133         1.412109         1.411621         -0.000024
12         1.411621         1.412109         1.411865         -0.000009
13         1.411865         1.412109         1.411987         -0.000001
14         1.411987         1.412109         1.412048         0.000002
15         1.411987         1.412048         1.412018         0.000001
16         1.411987         1.412018         1.412003         -0.000000
17         1.412003         1.412018         1.412010         0.000000
root2 =  1.4120
```

1.16 Μέθοδος ελαχίστων τετραγώνων

Ένα κοινό πρόβλημα σε μετρήσεις είναι το ταίριασμα των μετρήσεων με κάποιο θεωρητικό μοντέλο. Η μέθοδος που χρησιμοποιείται σε αυτή τη περίπτωση είναι η μέθοδος των ελαχίστων τετραγώνων. Σύμφωνα με αυτή υπολογίζουμε τη διαφορά από κάθε μέτρηση με την αντίστοιχη θεωρητική και αθροίζουμε τα τετράγωνα όλων αυτών των διαφορών. Το θεωρητικό μοντέλο έχει κάποιες παραμέτρους. Η ελαχιστοποίηση του αθροίσματος των τετραγώνων οδηγεί σε εξισώσεις από τις οποίες υπολογίζονται αυτές οι παράμετροι και έτσι έχουμε ένα θεωρητικό μοντέλο που ταιριάζει όσο γίνεται καλύτερα με τις μετρήσεις μας[1].

Η πιο απλή περίπτωση είναι το μοντέλο μας να είναι γραμμικό, δηλ. να χαρακτηρίζεται από μια ευθεία γραμμή $y = ax+b$. Το ζητούμενο τότε είναι να βρούμε τις τιμές των παραμέτρων a, b έτσι ώστε το άθροισμα των τετραγώνων των διαφορών από n πειραματικά σημεία $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$ να είναι ελάχιστο.

Η διαφορά για το σημείο i είναι

$$\delta_i = y_i - (ax_i + b)$$

και το άθροισμα των τετραγώνων των διαφορών είναι

$$E = \sum_{i=1}^{n} \delta_i^2 = (y_1 - ax_1 - b)^2 + (y_2 - ax_2 - b)^2 + \cdots + (y_n - ax_n - b)^2$$

Η ελαχιστοποίηση του $E(a, b)$ σημαίνει $\partial E/\partial a = 0$ και $\partial E/\partial b = 0$.

$$\frac{\partial E}{\partial a} = 2(y_1 - ax_1 - b)(-x_1) + 2(y_2 - ax_2 - b)(-x_2) + ... + 2(y_n - ax_n - b)(-x_n) = 0$$

$$\frac{\partial E}{\partial b} = 2(y_1 - ax_1 - b)(-1) + 2(y_2 - ax_2 - b)(-1) + ... + 2(y_n - ax_n - b)(-1) = 0$$

Απαλείφοντας το 2 σχηματίζουμε το παρακάτω σύστημα δυο γραμμικών εξισώσεων

$$a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i$$

$$a\sum_{i=1}^{n} x_i + nb = \sum_{i=1}^{n} y_i$$

από όπου μπορούμε να βρούμε τις παραμέτρους a, b.

Πίνακας 1.6: Πειραματικές μετρήσεις για μέθοδο ελαχίστων τετραγώνων

x	0.0	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4	2.7	3.0	3.3	3.6	3.9
y	5.7	4.3	6.0	3.7	7.5	9.5	8.5	9.8	11.5	11.7	10.9	10.5	13.6	14.1

Για εφαρμογή, έστω ότι έχουμε τις μετρήσεις του πιν. 1.6

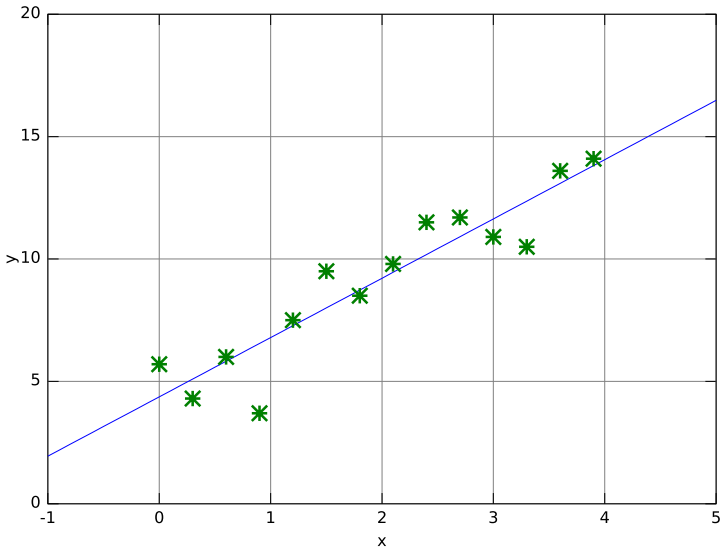
Ο παρακάτω κώδικας

```
xx=0:0.3:3.9;
yy=[5.7 4.3 6 3.7 7.5 9.5 8.5 9.8 11.5 11.7 10.9 10.5 13.6 14.1];
n=length(yy)
A=[sum(xx.*xx) sum(xx) ; sum(xx) length(yy)]
beta = [sum(xx.*yy); sum(yy)]
inv(A)*beta
a=ans(1); b=ans(2);
x=-1:0.1:5;
y=a*x+b;
plot(x,y,xx,yy,'*'); grid;
xlabel('x'); ylabel('y')

n =  14
A =
   73.710   27.300
   27.300   14.000
```

[1] Το κριτίριο του αθροίσματος των τετραγώνων των διαφορών δεν είναι το μοναδικό. Είναι όμως αυτό που υπολογιστικά είναι το πιο απλό.

```
beta =
   297.84
   127.30
ans =
   2.4227
   4.3686
```

μας δίνει το αποτέλεσμα που φαίνεται στο σχ. 1.19.



Σχήμα 1.19: Υπολογισμός ευθείας που διέρχεται από σύνολο σημείων και παρουσιάζει την μικρότερη απόκλιση σύμφωνα με τη μέθοδο ελαχίστων τετραγώνων.

Στην κατηγορία των γραμμικών μοντέλων ανήκουν και τα μοντέλα της μορφής $y = ae^{\rho x}$ εφόσον με λογαρίθμηση, $\ln y = \ln a + \rho x$, μετασχηματίζονται σε γραμμικά.

Το Octave έχει ενσωματωμένη συνάρτηση για πολυωνυμικά μοντέλα. Αυτό σημαίνει ότι μπορεί να μας υπολογίσει το καλύτερο πολυώνυμο που ταιριάζει με τα δεδομένα μας σύμφωνα με το κριτήριο των ελαχίστων τετραγώνων. Δοκιμάζοντάς τη για την περίπτωσή μας (πολυώνυμο 1ου βαθμού) έχουμε

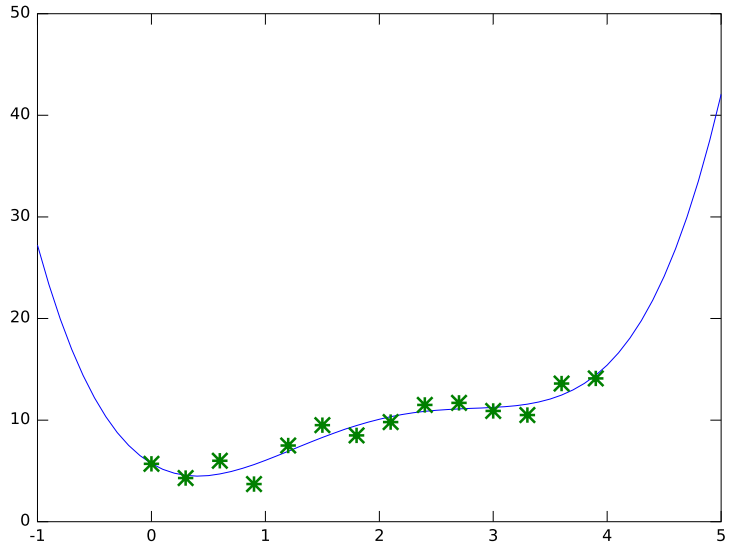
```
xx=0:0.3:3.9;
yy=[5.7 4.3 6 3.7 7.5 9.5 8.5 9.8 11.5 11.7 10.9 10.5 13.6 14.1];
p=polyfit(xx,yy,1)

p =
   2.4227   4.3686
```

τους ίδιους συντελεστές που υπολογίσαμε παραπάνω.

Για πειραματισμό, δοκιμάζουμε πολυώνυμο 5ου βαθμού.

```
xx=0:0.3:3.9;
yy=[5.7 4.3 6 3.7 7.5 9.5 8.5 9.8 11.5 11.7 10.9 10.5 13.6 14.1];
p=polyfit(xx,yy,5);
x=-1:0.1:5;
y=polyval(p,x);
plot(x,y,xx,yy,'*')
```

Στην περιοχή που έχουμε πειραματικά δεδομένα βλέπουμε καλύτερη συμφωνία. Έξω όμως από αυτή τη περιοχή οι αποκλίσεις μπορεί να είναι τεράστιες. Επισημαίνεται επίσης ότι αν πάρουμε πολυώνυμο βαθμού όσος είναι ο αριθμός των δεδομένων μας (ή και μεγαλύτερου), αυτό θα περάσει από όλα τα σημεία. Τι είναι καλύτερο; Εξαρτάται. Αν τα δεδομένα είναι από πειραματικές μετρήσεις, έχουν θόρυβο. Δεν προσπαθούμε τότε να «ταιριάξουμε» τις διακυμάνσεις του θορύβου. Υποτίθεται ότι υπάρχει κάποιο φυσικό μοντέλο που περιγράφει το σύστημά μας και αυτό είναι που προσαρμόζουμε. Αν όμως τα δεδομένα προέρχονται πράγματι από κάποια πολυωνυμική συνάρτηση μεγάλου βαθμού, τότε εντάξει. Απλώς, η δεύτερη περίπτωση είναι απίθανη για συνήθεις εφαρμογές στην πράξη.



Σχήμα 1.20: Υπολογισμός πολυωνύμου 5ου βαθμού για το ίδιο σύνολο σημείων όπως πριν που παρουσιάζει την μικρότερη απόκλιση σύμφωνα με τη μέθοδο ελαχίστων τετραγώνων.

1.17 Μέθοδοι αριθμητικής ολοκλήρωσης

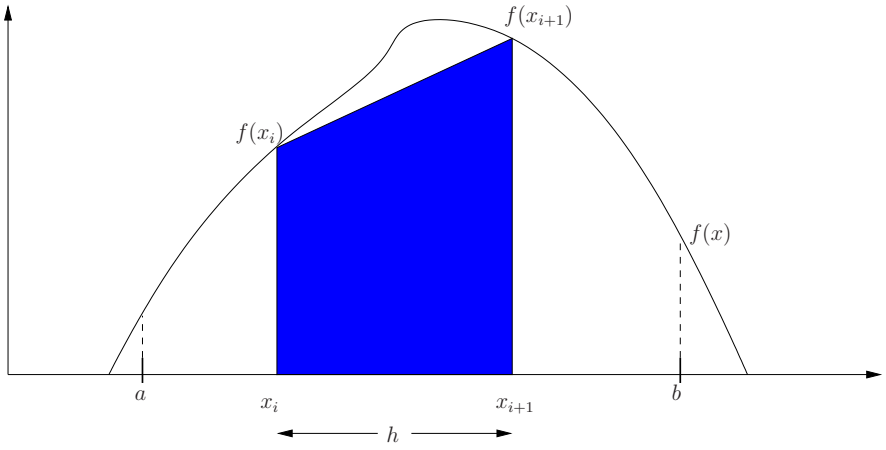
Συχνά στην πράξη θέλουμε να υπολογίσουμε από ένα σύνολο μετρήσεων ή δεδομένων κάποιο ολοκλήρωμα, π.χ. την ισχύ από ένα φάσμα ισχύος. Στην περίπτωση αυτή, συνήθως, δεν έχουμε αναλυτική περιγραφή της συνάρτησης για να υπολογίσουμε το ολοκλήρωμα (ή αυτή που έχουμε είναι αρκετά πολύπλοκη). Τότε καταφεύγουμε στον αριθμητικό υπολογισμό του εμβαδού κάτω από τη γραφική των μετρήσεων. Περιγράφονται δυο μέθοδοι.

1.17.1 Κανόνας Τραπεζίου

Στον κανόνα του τραπεζίου υποδιαιρούμε την περιοχή κάτω από την γραφική σε n τραπέζια ίσου ύψους $h = (b-a)/n$ όπου a, b η αρχή και το τέλος της περιοχής που θέλουμε να υπολογίσουμε το εμβαδόν. Έχουμε τότε:

$$\int_a^b f(x)dx \quad \sim \quad \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_i)\right]$$

όπου $x_0, x_1, ..., x_n$, $f(x_0), f(x_1), ..., f(x_n)$, με $a = x_0$, $b = x_n$.



Σχήμα 1.21: Κανόνας τραπεζίου για αριθμητική ολοκλήρωση.

Για παράδειγμα εφαρμογής έστω ότι θέλουμε την τιμή του ολοκληρώματος $\int_0^4 x^3 dx$.

```
% integ1.m
% Εφαρμόζει μέθοδο τραπεζίου για υπολογισμό ολοκληρώματος
n=10;                                   % αριθμός τραπεζίων
a=0; b=4;                               % αρχή και τέλος
h=(b-a)/n;                              % ύψος τραπεζίων
x=a+[0:n]*h;                            % διάνυσμα x
y=x.^3;                                 % διάνυσμα y
S=(h/2)*(y(1)+y(n+1)+2*sum(y([2:n])))   % το ολοκλήρωμα
```

Η ακριβής τιμή του ολοκληρώματος είναι 64 και βλέπουμε ότι για $n = 10, S = 64.64$ ενώ για $n = 100, S = 64.006$. Δηλ. όσο πιο πολλές υποδιαιρέσεις χρησιμοποιούμε τόσο καλύτερο το αποτέλεσμα. Στην πράξη, όταν δεν έχουμε αναλυτική συνάρτηση για να δημιουργήσουμε τα διανύσματα `x`, `y`, χρησιμοποιούμε μετρήσεις. Σε αυτή την περίπτωση υπάρχει έτοιμη και η συνάρτηση `trapz` του Octave που την καλούμε π.χ. με `S = trapz(x,y)` και μας δίνει το ίδιο αποτέλεσμα.

1.17.2 Κανόνας Simpson

Μεγαλύτερη ακρίβεια έχουμε με τον κανόνα του Simpson όπου αντί για ευθύγραμμο τμήμα μεταξύ σημείων $f(x_i)$, $f(x_{i+1})$, εφαρμόζεται μια παραβολή μεταξύ τριών διαδοχικών σημείων. Ο αριθμός των τμημάτων n πρέπει να είναι άρτιος, $h = (b-a)/n$ όπου a, b η αρχή και το τέλος της περιοχής που θέλουμε να υπολογίσουμε το εμβαδόν. Έχουμε τότε:

$$\int_a^b f(x)dx \quad \sim \quad \frac{h}{3}\left[f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_{2i}) + 4\sum_{i=1}^{n} f(x_{2i-1})\right]$$

όπου $x_0, x_1, ..., x_n$, $f(x_0), f(x_1), ..., f(x_n)$, με $a = x_0$, $b = x_n$.

```
% integ2.m
n=10;           % αριθμός τμημάτων
a=0; b=4;       % αρχή και τέλος
h=(b-a)/n;      % μήκος τμήματος
x=a+[0:n]*h;    % διάνυσμα x
y=x.^3;         % διάνυσμα y
S=(h/3)*(y(1)+y(n+1)+2*sum(y([3:2:n]))+4*sum(y([2:2:n])))  % το ολοκλήρωμα
```

Βλέπουμε ότι και για μικρά n έχουμε την ακριβή τιμή 64. Αυτό ισχύει για όλες τις πολυωνυμικές συναρτήσεις με τη μέθοδο Simpson. Εν γένει χρειαζόμαστε μικρότερο αριθμό τμημάτων από τη μέθοδο τραπεζίου για την ίδια ακρίβεια.

Το Octave έχει και εδώ έτοιμη συνάρτηση για τον αριθμητικό υπολογισμό ολοκληρωμάτων όταν γνωρίζουμε τον μαθηματικό τύπο της προς ολοκλήρωση συνάρτησης. Ο παρακάτω κώδικας δείχνει τη χρήση της `quadv` για την εύρεση του

$$\int_0^4 x^3 dx$$

```
f = inline("x.^3");
S2 = quadv(f,0,4)

S2 =  64
```

Και ένα παράδειγμα για μια πιο δύσκολη συνάρτηση, την $x\sin(1/x)$, από 0 έως 3.

```
clear all

function y = g(x)
  if (x == 0.0)
     y = 0.0;
  else
      y = x.*sin(1./x);
  end
endfunction
S3 = quadv("g",0,3)

S3 =  2.27004890467348
```

Με την `quadv` (που βασίζεται στη μέθοδο Simpson) βλέπουμε ότι το `S2` έχει την ίδια τιμή με τη δική μας υλοποίηση. Για την πιο δύσκολη συνάρτηση (δοκιμάστε να κάνετε τη γραφική της κοντά στο 0 και θα δείτε γιατί τη χαρακτηρίσαμε δύσκολη), βλέπουμε ότι τα καταφέρνει αρκετά καλά (η τιμή της `S3` είναι ακριβής μέχρι 5 σημαντικά ψηφία). Σημειώστε επίσης στον παραπάνω κώδικα τον τρόπο που δίνουμε συναρτήσεις σαν ορίσματα, `inline` για συναρτήσεις απλές, μιας γραμμής, και `function` για πιο σύνθετες. Κοιτάξτε επίσης στο online-help για επιπλέον μεθόδους.

1.18 Διαφορικές εξισώσεις

Ένα άλλο σημαντικό πρόβλημα είναι η επίλυση διαφορικών εξισώσεων. Και εδώ οι διαθέσιμες τεχνικές είναι πολλές και εξαρτώνται από το είδος του προβλήματος και τη συγκεκριμένη διαφορική εξίσωση που το περιγράφει. Θα δούμε εδώ την περίπτωση απλών κανονικών διαφορικών εξισώσεων, με σταθερούς συντελεστές, και δεδομένες αρχικές συνθήκες.

Έστω ότι έχουμε μια συνάρτηση $y(t)$ η οποία είναι λύση της

$$a_n y^{(n)} + a_{n-1} y^{(n-1)} + \cdots + a_1 y' + a_0 y = f(t)$$

όπου $f(t)$ γνωστή συνάρτηση με αρχικές συνθήκες $y(0), y'(0), \ldots, y^{(n-1)}(0)$. Θέτουμε

$$x_1 = y, \quad x_2 = y', \quad x_3 = y'', \quad \ldots, \quad x_n = y^{(n-1)}$$

οπότε

$$x_1' = x_2, \quad x_2' = x_3, \quad x_3' = x_4, \ldots, \quad x_n' = -\frac{a_{n-1}x_n + \cdots + a_1 x_2 + a_0 x_1 - f(t)}{a_n}$$

Με τον τρόπο αυτό ανάγουμε μια n-τάξης διαφορική εξίσωση σε σύστημα n διαφορικών εξισώσεων πρώτης τάξης.

Επομένως το γενικό πρόβλημα είναι η επίλυση του συστήματος:

$$\frac{dx}{dt} = f(x,t) \qquad \text{με} \qquad x(t_0) = x_0 \qquad \text{όπου} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

με t την ανεξάρτητη μεταβλητή από την οποία εξαρτώνται τα στοιχεία του x.

Παράδειγμα 1.6 *Να υπολογιστεί η λύση $y(t)$ για $t \in [1,5]$ της εξίσωσης:*

$$y' + y = 0 \qquad \mu\varepsilon \qquad y(1) = 8$$

Πρώτης τάξεως διαφορική εξίσωση με αναλυτική λύση να βγαίνει εύκολα ότι είναι $y(t) = 8e^{1-t}$. Άρα

$$x_1 = y, \quad x_1' = -x_1$$

και στην ουσία έχουμε απλή εξίσωση, όχι σύστημα, και ο κώδικας είναι:

```
clear all; close all;

function xdot = f(x,t)
  xdot(1) = -x(1);
end

x0 = 8;
t = linspace(1,5,50);
y = lsode("f", x0, t);
yy = 8*exp(1-t);
subplot(121); plot(t,y,t,yy); grid; xlabel('t'); ylabel('y(t)')
subplot(122); plot(t,abs(y-yy')); grid; xlabel('t'); ylabel('διαφορά')
```

Βλέπουμε στο σχ. 1.22 την αριθμητική λύση που μας δίνει ο κώδικας μαζί με την αναλυτική όπου πρακτικά ταυτίζονται. Για να δούμε τη διαφορά παίρνουμε την απόλυτη τιμή της διαφοράς των δυο (προσοχή εδώ γιατί η μια είναι διάνυσμα γραμμής και η άλλη διάνυσμα στήλης) και βλέπουμε ότι είναι της τάξης 10^{-7}. Αν πειραματιστείτε λίγο με αυτό το παράδειγμα θα δείτε ότι πρέπει να αρχίζετε πάντα από την αρχική τιμή. Π.χ. για $t \in [-2,5]$ τι αλλαγή θα κάνετε;

Παράδειγμα 1.7 *Να υπολογιστεί η λύση $y(t)$ για $t \in [0,1]$ της εξίσωσης:*

$$y'' - 4y' + 13 = e^{-5t} \qquad \mu\varepsilon \qquad y(0) = 0, y'(0) = 10$$

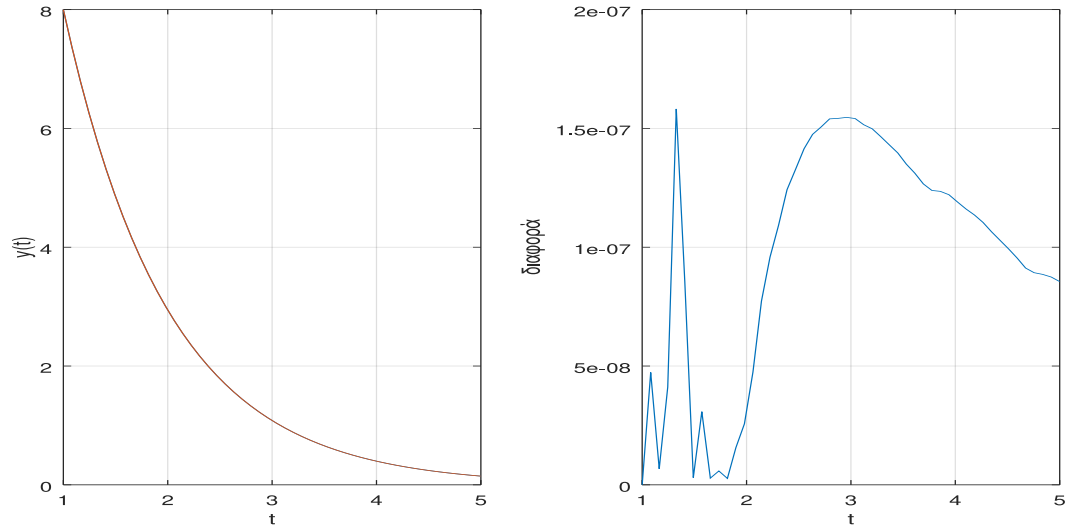
Θέτουμε

$$x_1 = y, \quad x_2 = y'$$

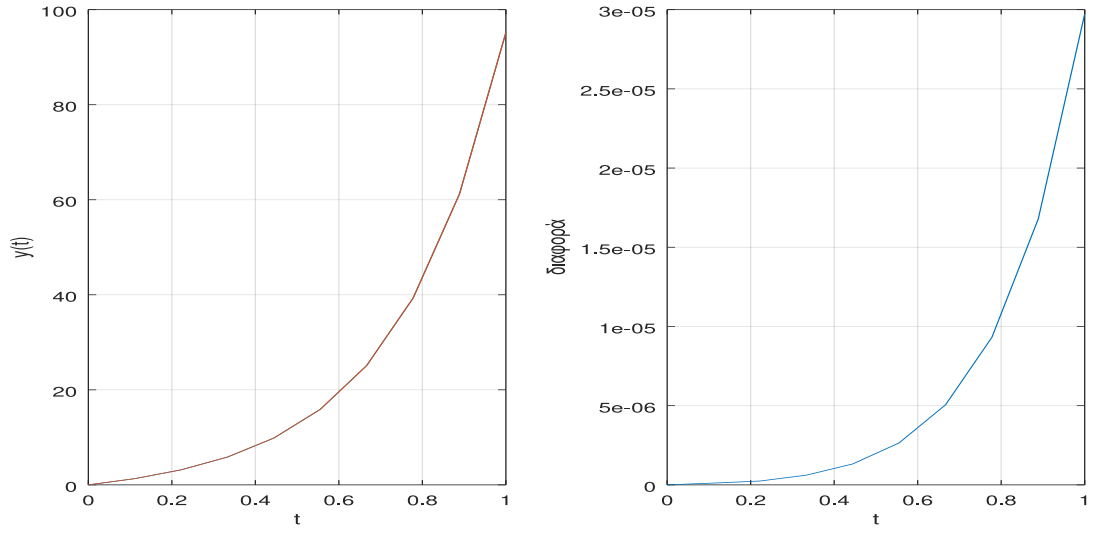
οπότε

$$x_1' = y' = x_2, \quad x_2' = y'' = 4y' - 13 + e^{-5t} = 4x_2 - 13 + e^{-5t}$$

Ο κώδικας octave που λύνει αυτήν την εξίσωση για $t \in [0,1]$ είναι:



Σχήμα 1.22: Αριθμητική και αναλυτική λύση $y(t)$ (αριστερά) όπου δεν φαίνεται διαφορά και η απόλυτος διαφορά αριθμητικής και αναλυτικής λύσης (δεξιά).



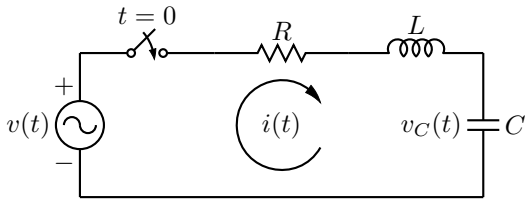
Σχήμα 1.23: Αριθμητική και αναλυτική λύση $y(t)$ αριστερά και η διαφορά τους δεξιά.

```
clear all; close all;

function xdot = f(x,t)
  xdot(1) = x(2);
  xdot(2) = 4*x(2)-13+exp(-5*t);
end

x0 = [0; 10];
t = linspace(0,1,10);
y = lsode("f", x0, t);
yy = (16/720)*exp(-5*t) + (247/144)*exp(4*t) + (2340*t+585)/720 - 51/20;
subplot(121); plot(t,y(:,1),t,yy); grid; xlabel('t'); ylabel('y(t)')
subplot(122); plot(t,abs(y(:,1)-yy')); grid; xlabel('t'); ylabel('διαφορά')
```

Ορίζουμε την συνάρτηση του συστήματος με τη μορφή `xdot = f(x,t)`. Δίνουμε τις αρχικές τιμές στο διάνυσμα στήλης `x0 = [0; 10]` και το χρονικό διάστημα που μας ενδιαφέρει με `t = linspace (0, 1, 10)` και καλούμε την `lsode`. Το αποτέλεσμα θα είναι ένας πίνακας `y`, 10×2, με δυο στήλες, που αντιστοιχούν στις μεταβλητές x_1, x_2 που αναφέραμε παραπάνω. Εμάς μας ενδιαφέρει η `y(:,1)` που αντιπροσωπεύει την λύση $y(t)$. Η εξίσωση είναι αρκετά απλή που μπορούμε

να υπολογίσουμε και την αναλυτική λύση (γραμμή κώδικα με yy). Η γραφική της αριθμητικής και αναλυτικής λύσης φαίνεται στο σχ. 1.23 και βλέπουμε ότι πρακτικά ταυτίζονται. Η διαφορά τους βλέπουμε ότι αυξάνει όσο απομακρυνόμαστε από την αρχική τιμή αν και στο διάστημα που μας ενδιαφέρει είναι μέχρι 3×10^{-5}.

Παράδειγμα 1.8 *Υπολογίστε την τάση και το ρεύμα στα άκρα του πυκνωτή συναρτήσει του χρόνου στο παρακάτω κύκλωμα για* $v(t) = \sin(5t)$ V, $0 \le t \le 5$ s *με αρχικές συνθήκες* $v_C(0) = 0$ V, $i_C(0) = 0$ A *και τιμές στοιχείων* $R = 1.5\ \Omega$, $L = 0.25$ H *και* $C = 0.5$ F.



Σχήμα 1.24: Κύκλωμα με εν σειρά αντίσταση, πηνίο, πυκνωτή.

Ο κανόνας τάσεων του Kirchhoff δίνει

$$L\,\frac{di}{dt} + R\,i + v_C = v$$

όπου $i(t) = i_C(t)$ και $i_C(t) = C dv_C(t)/dt$. Οπότε

$$LC\,\frac{d^2v_C}{dt^2} + RC\frac{dv_C}{dt} + v_C = v \;\Rightarrow\; \frac{d^2v_C}{dt^2} + \frac{R}{L}\frac{dv_C}{dt} + \frac{v_C}{LC} = \frac{v}{LC}$$

Κάνοντας τις απαιτούμενες αντιστοιχίσεις

$$\begin{aligned} x_1 &= v_C & x_1' &= x_2 \\ x_2 &= v_C' & x_2' &= -\frac{R}{L}x_2 - \frac{1}{LC}x_1 + \frac{v}{LC} \end{aligned}$$

με αρχικές συνθήκες $x_1(0) = 0, x_2(0) = 0/C = 0$ έχουμε

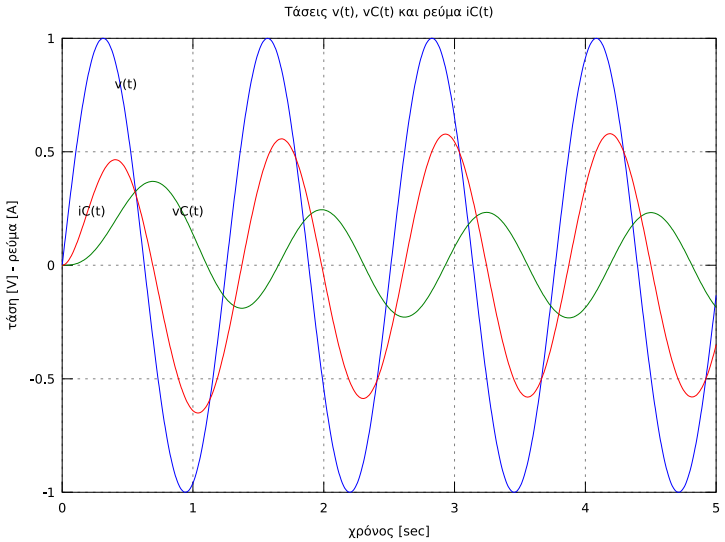
```
clear all; close all;

function xdot = f (x, t)
  R=1.5; L=0.25; C=0.5;
  xdot(1) = x(2);
  xdot(2) = -(R/L)*x(2) - (1/(L*C))*x(1) + sin(5*t)/(L*C);
endfunction

C=0.5;
x0 = [0; 0];
t = linspace (0, 5, 1000);
y = lsode ("f", x0, t);
plot(t,sin(5*t),t,y(:,1),t,C*y(:,2)); grid
title('Τάσεις v(t), vC(t) και ρεύμα iC(t)')
xlabel('χρόνος [sec]'); ylabel('τάση [V] - ρεύμα [A]');

text(0.12,0.24,'iC(t)');
text(0.84,0.24,'vC(t)');
text(0.4,0.8,'v(t)');
```

Οι τελευταίες τρεις εντολές τοποθετούν text στα σημεία της γραφικής που θέλουμε. Τα σημεία αυτά τα βρίσκουμε πρώτα με το «ποντίκι» μας καθώς το μετακινούμε πάνω από τη γραφική. Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε για τον ίδιο σκοπό τη δυνατότητα που μας παρέχει το παράθυρο γραφικών. Στην `plot` η `y(:,1)` είναι η v_C και η `y(:,2)` είναι η v_C', άρα πολλαπλασιάζοντας τη δεύτερη με C έχουμε το ρεύμα $i_C(t)$. Μπορούμε να δούμε τις κυματομορφές τάσης και ρεύματος για τον πυκνωτή του κυκλώματος στο σχ. 1.25.

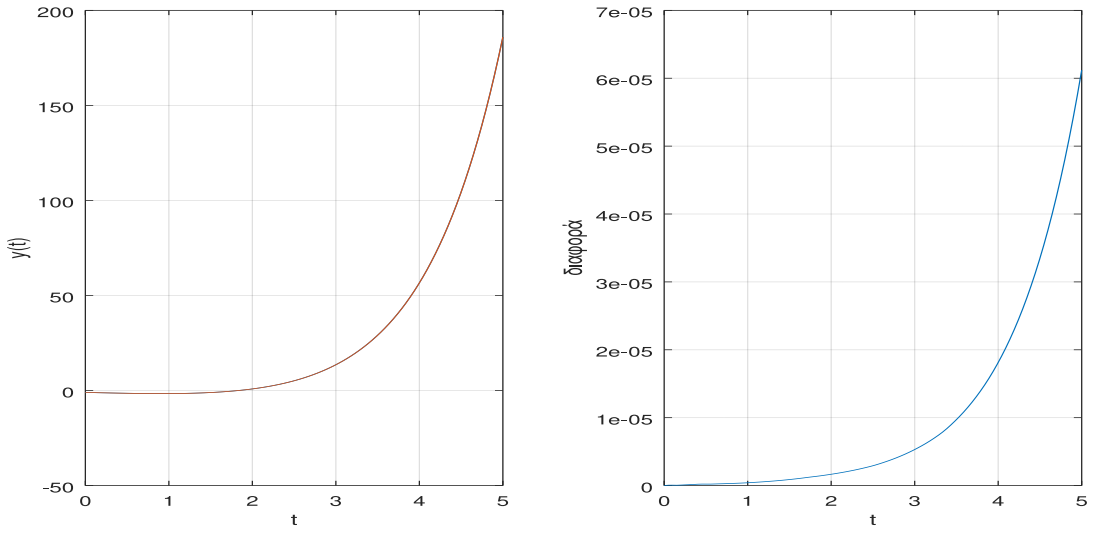
Παράδειγμα 1.9 *Να βρεθεί η λύση της διαφορικής εξίσωσης για* $t \in [0, 5]$

$$\frac{dy}{dt} = y + t^2 + \sin(t) \qquad \textit{με} \qquad y(0) = -1$$

και να συγκριθεί με την αναλυτική λύση $y(t) = 1.5\exp(t) - (t^2 + 2t + 2) - (\sin(t) + \cos(t))/2$.



Σχήμα 1.25: Τάση πηγής $v(t)$, ρεύμα και τάση πυκνωτή $i_C(t)$, $v_C(t)$ αντίστοιχα.



Σχήμα 1.26: Λύση $y(t)$ από την `lsode` μαζί με την αναλυτική λύση.

Εδώ έχουμε

$$\frac{dy}{dt} - y = t^2 + \sin(t)$$

με

$$x_1 = y \qquad x_1' = x_1 + t^2 + \sin(t)$$

και

```
clear all; close all;

function xdot = f (x, t)
  xdot(1) = x(1) + t^2 + sin(t);
endfunction

x0 = -1;
t = linspace (0, 5, 100);
y = lsode ("f", x0, t);
yy = 1.5*exp(t) - (t.^2+2*t+2) - (sin(t)+cos(t))/2;
subplot(121); plot(t,y,t,yy); grid; xlabel('t'); ylabel('y(t)')
subplot(122); plot(t,abs(y-yy')); grid; xlabel('t'); ylabel('διαφορά')
```

Και η λύση στο σχ. 1.26 δείχνει ότι αριθμητική και αναλυτική λύση πρακτικά ταυτίζονται αν και βλέπουμε το σφάλμα να αυξάνει όσο απομακρυνόμαστε από την αρχή.

Παράδειγμα 1.10 *Να βρεθεί η λύση του συστήματος διαφορικών εξισώσεων για* $t \in [0, 10]$

$$dx/dt = 10(y - x)$$
$$dy/dt = -xz + 28x - y$$
$$dz/dt = xy - 8z/3$$

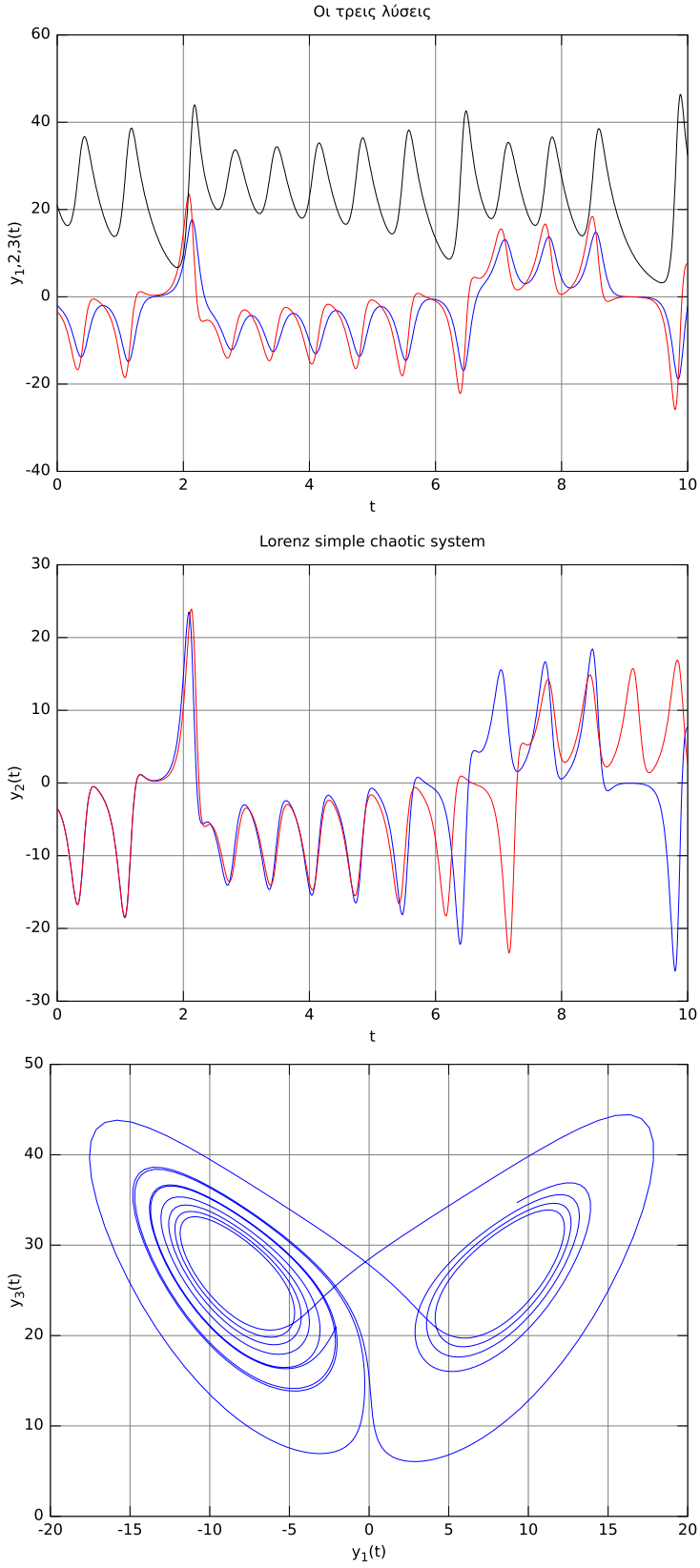
με αρχικές συνθήκες $x(0) = -2, y(0) = -3.5, z(0) = 21$. *Σχεδιάστε τις τρεις συναρτήσεις - λύσεις με διαφορετικά χρώματα (μπλέ, κόκκινο, μαύρο). Κάντε τη γραφική της δεύτερης και κρατήστε το παράθυρο γραφικών. Αλλάξτε την αρχική συνθήκη* $x(0)$ *σε* $x(0) = -2.04$ *και επαναλάβετε τη λύση. Σχεδιάστε πάλι τη γραφική της δεύτερης στο ίδιο παράθυρο που είχατε κρατήσει. Τι παρατηρείτε; Σχεδιάστε σε ξεχωριστό παράθυρο τη λύση της πρώτης συναρτήσει της τρίτης. Τι παρατηρείτε και εδώ;*

Ο παρακάτω κώδικας υπολογίζει τα ζητούμενα.

```
% loren.m
clear all

function xdot = f (x, t)
  xdot(1) = 10*(x(2)-x(1));
  xdot(2) = -x(1)*x(3)+28*x(1)-x(2);
  xdot(3) = x(1)*x(2)-(8/3)*x(3);
endfunction

x0 = [-2; -3.5; 21];
t = linspace (0, 10, 1000);
y = lsode ("f", x0, t);

plot(t,y(:,1),'b',t,y(:,2),'r',t,y(:,3),'k');
grid; title('Οι τρεις λύσεις');
xlabel('t'); ylabel('y_1,2,3(t)');

figure

plot(t,y(:,2),'b');
title('Lorenz simple chaotic system');
xlabel('t'); ylabel('y_2(t)');

hold

x0 = [-2.04; -3.5; 21];
y = lsode ("f", x0, t);

plot(t,y(:,2),'r'); grid;
title('Lorenz simple chaotic system');
xlabel('t'); ylabel('y_2(t)');

hold

figure
plot(y(:,1),y(:,3)); grid;
xlabel('y_1(t)'); ylabel('y_3(t)');
```

Προσέξτε τη χρήση της εντολής `hold` που κρατάει ανοικτό το παράθυρο γραφικών μέχρι να την ξανακαλέσουμε.

Στην πρώτη γραφική παράσταση του Σχ. 1.27 βλέπουμε τις τρεις λύσεις μαζί. Στην δεύτερη γραφική έχουμε την $y(:, 2)$ και για τις δυο περιπτώσεις. Η διαφορά στις αρχικές συνθήκες είναι μικρή, μόνο 2%, και μόνο στην πρώτη συνθήκη. Βλέπουμε στις λύσεις ότι μέχρι $t = 1.5$ η διαφορά είναι αμελητέα. Μετά αρχίζει να υπάρχει απόκλιση που διαρκώς μεγαλώνει και γύρω στο $t = 6$ η δεύτερη λύση έχει διαφοροποιηθεί σημαντικά από την πρώτη. Αυτό είναι σημάδι εμφάνισης χάους. Μικρές διαφορές στις αρχικές συνθήκες οδηγούν σε μεγάλες διαφορές στις λύσεις. Το συγκεκριμένο μάλιστα σύστημα διαφορικών εξισώσεων μοιάζει με τα συστήματα που εξέτασε ο μετεωρολόγος Lorenz τη δεκαετία του 60 σαν παράδειγμα χαοτικών δυναμικών συστημάτων (όπως αυτό της πρόγνωσης του καιρού). Στην τρίτη γραφική παράσταση (χώρος φάσεων) βλέπουμε και το σήμα κατατεθέν ενός χαοτικού συστήματος, τους δυο χαοτικούς πόλους έλξης (Lorenz attractors).



Σχήμα 1.27: Σύστημα χάους Lorenz.

1.19 Μελέτη πληθυσμού χώρας

Στα παρακάτω θα κοιτάξουμε δυο μοντέλα για τον πληθυσμό μιας χώρας.

Μοντέλο Malthus

Κάθε πλυθυσμός, εάν δεν υπάρχουν εμπόδια ανάπτυξης, αυξάνεται εκθετικά (Thomas Malthus, *An Essay on the Principle of Population*, 1798). Μαθηματικά, αυτό σημαίνει ότι η αύξηση πληθυσμού είναι ανάλογη με τον πληθυσμό.

$$\frac{dp}{dt} = kp \quad \Rightarrow \quad p(t) = p_0 e^{k(t-t_0)}$$

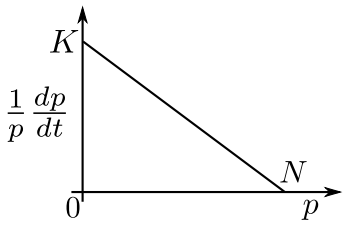
όπου $t = t_0, p = p_0$ οι αρχικές συνθήκες χρόνου και πληθυσμού.

Λογιστικό μοντέλο

Στην πράξη βέβαια εμπόδια υπάρχουν, π.χ. λόγω έλλειψης πόρων. Κάνοντας μια τροποποίηση στο μοντέλο Malthus δεχόμαστε ότι η αναλογία του Malthus ισχύει για μικρούς πληθυσμούς (αρκετοί πόροι). Με την αύξηση του πληθυσμού θεωρούμε ότι ο σχετικός ρυθμός αύξησης ελλατώνεται. Μαθηματικά, ο σχετικός ρυθμός αύξησης είναι:

$$\frac{\left(\frac{dp}{dt}\right)}{p} = \frac{1}{p}\frac{dp}{dt}$$

Το λογιστικό μοντέλο (Σχ. 1.28) θεωρεί ότι η ελάττωση είναι γραμμική, με τον πληθυσμό να μεταβάλλεται από 0 μέχρι μια μέγιστη τιμή N και ο σχετικός ρυθμός αύξησης να ελαττώνεται από την τιμή K στο 0. Η κλίση της ευθείας είναι K/N και το μαθηματικό μοντέλο γίνεται:



Σχήμα 1.28: Λογιστικό μοντέλο γραμμικής ελάττωσης πληθυσμού.

$$\frac{1}{p}\frac{dp}{dt} = K - mp = K - \frac{K}{N}p \quad \Rightarrow \quad \frac{dp}{dt} = Kp\left(1 - \frac{p}{N}\right) \quad \Rightarrow \quad \int_{p_0}^{p} \frac{dp}{p\left(1-\frac{p}{N}\right)} = \int_{t_0}^{t} K dt \quad \Rightarrow$$

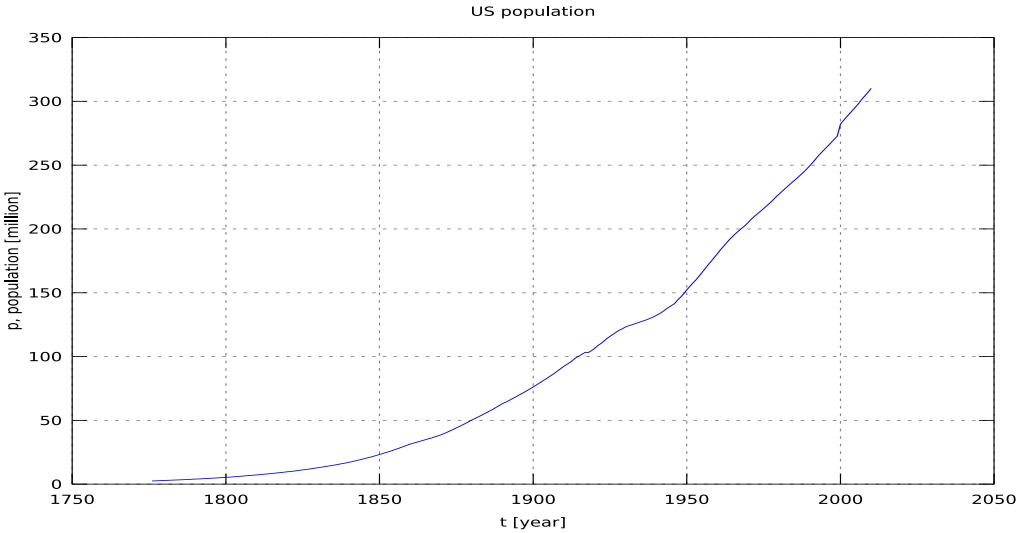
$$p(t) = \frac{N}{1 + \left(\frac{N}{p_0} - 1\right)\exp\left[-K(t-t_0)\right]}$$

1.19.1 Πληθυσμός ΗΠΑ

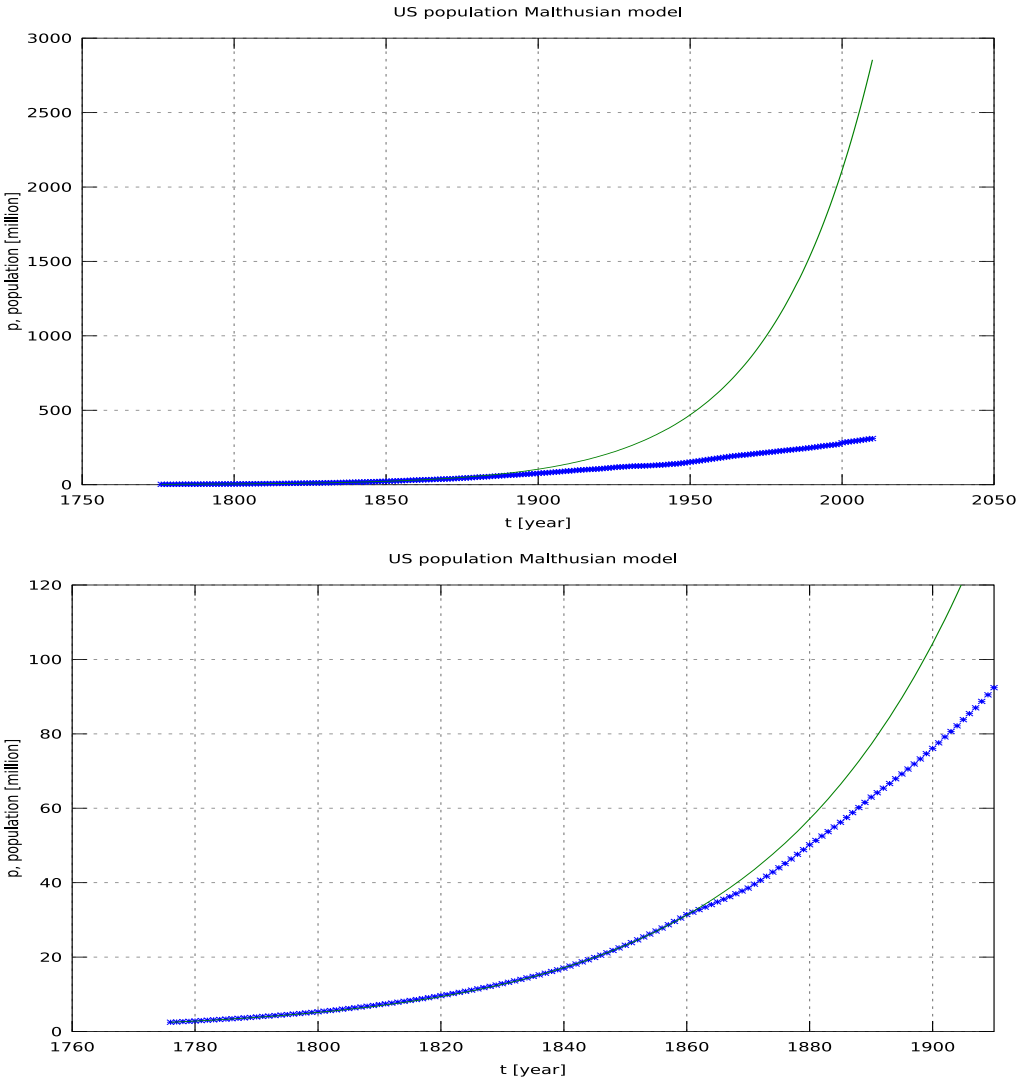
Για εφαρμογή μπορούμε να κοιτάξουμε τον πληθυσμό των ΗΠΑ. Στο διαδίκτυο εύκολα βρίσκει κανείς δεδομένα από το 1776 μέχρι το 2010 (`https://www.google.com/fusiontables/DataSource?dsrcid=225439#rows:id=1`). Παρέχονται εδώ στο αρχείο USpop2.csv.

Ο παρακάτω κώδικας τα διαβάζει και δείχνει την πρώτη γραφική (Σχ. 1.29). Ο χρόνος t είναι σε έτη και ο εκάστοτε πληθυσμός σε εκατομμύρια κατοίκους. Τα δεδομένα είναι σε φθίνουσα σειρά και τα αναστρέφουμε. Η πρώτη γραφική είναι πρόχειρη και την κάνουμε για να σχηματίσουμε μια αρχική ιδέα για τα δεδομένα μας. Τα δεδομένα δείχνει να έχουν τη μορφή της αύξουσας εκθετικής.

```
load USpop2.csv
us = USpop2;
t=us(end:-1:1,1); p=us(end:-1:1,2)*1e-6;
title(”US population”);
xlabel(”t [year]”); ylabel(”p, population [million]”);
```



Σχήμα 1.29: Δεδομένα πληθυσμού ΗΠΑ



Σχήμα 1.30: Δεδομένα πληθυσμού ΗΠΑ με μοντέλο Malthus. Η δεύτερη γραφική είναι zoom της πρώτης στην περιοχή όπου εμφανίζεται απόκλιση από το μοντέλο.

Εφαρμόζοντας το μοντέλο Malthus παίρνουμε δυο σημεία (έτη 1790 και 1800) και υπολογίζουμε τον συντελεστή k

$$\left.\begin{array}{rcl} p_2 & = & p_0 \exp(k(t_2 - t_0)) \\ p_1 & = & p_0 \exp(k(t_1 - t_0)) \end{array}\right\} \quad \Rightarrow \quad k = \frac{1}{t_2 - t_1} \ln\left(\frac{p_2}{p_1}\right) = 0.030087$$

Φτιάχνουμε δεύτερη γραφική (Σχ. 1.30) με τα δεδομένα (σημεία) και το μοντέλο (συνεχής γραμμή). Βλέπουμε πολύ καλή συμφωνία με το μοντέλο μέχρι το 1860-1861. Από την ιστορία ξέρουμε ότι τότε ξέσπασε ο εμφύλιος πόλεμος των ΗΠΑ και εξηγείται έτσι η απόκλιση.

```
t0=t(1)
p0=p(1)
t1=1790
p1=us(us(:,1)==t1,:)(2)*1e-6
t2=1800
p2=us(us(:,1)==t2,:)(2)*1e-6
k=0.1*log(p2/p1)

% Μοντέλο Malthus
plot(t,p,'*', "markersize", 2, t, p0*exp(k*(t-t0)),'-'); grid
title("US population Malthusian model");
xlabel("t [year]"); ylabel("p, population [million]");
```

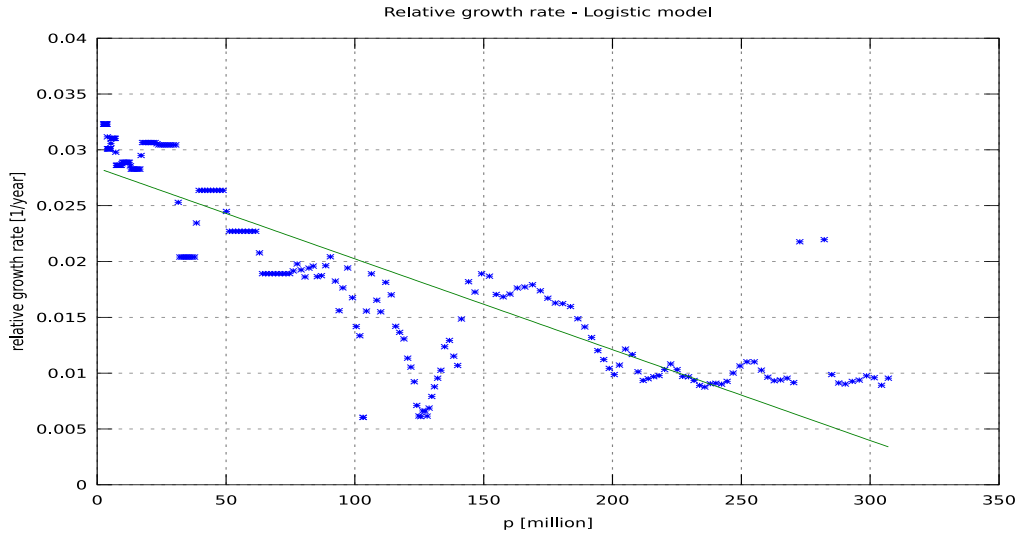
Προχωρούμε στην εφαρμογή του λογιστικού μοντέλου. Χρειαζόμαστε έναν τρόπο να υπολογίσουμε τα K και N του μοντέλου άρα έναν τρόπο να υπολογίσουμε τον σχετικό ρυθμό ελάττωσης. Ανά τρία σημεία δεδομένων, $i-1, i, i+1$, υπολογίζουμε την παράγωγο από τις διαφορές:

$$\frac{1}{p}\frac{dp}{dt} \approx \frac{1}{p_i}\frac{p_{i+1} - p_{i-1}}{t_{i+1} - t_{i-1}}$$

Χρησιμοποιούμε τρία σημεία για να μην εισάγουμε συστηματικό σφάλμα στον υπολογισμό. Στο octave δημιουργούμε την συνάρτηση relg που κάνει αυτόν τον υπολογισμό (σε ξεχωριστό αρχείο με όνομα relg.m). Προφανώς το διάνυσμα rate θα έχει μήκος κατά 2 μικρότερο από τα t, p.

```
function [rate] = relg(t,p)
  N = length(t);
  rate = [];
  for i=2:N-1
    rate(i-1) = (p(i+1)-p(i-1))/(t(i+1)-t(i-1))/p(i);
  end
  rate = rate';
end
```

Σχηματίζουμε την γραφική του σχετικού ρυθμού ελάττωσης έναντι του πληθυσμού (Σχ. 1.31) και βλέπουμε αρκετές διακυμάνσεις. Το μοντέλο ορίζει ευθεία γραμμή και χρησιμοποιούμε μέθοδο ελαχίστων τετραγώνων για τον υπολογισμό της. Το σημείο τομής με τον κατακόρυφο άξονα είναι το K και το σημείο τομής με τον οριζόντιο άξονα είναι το N.

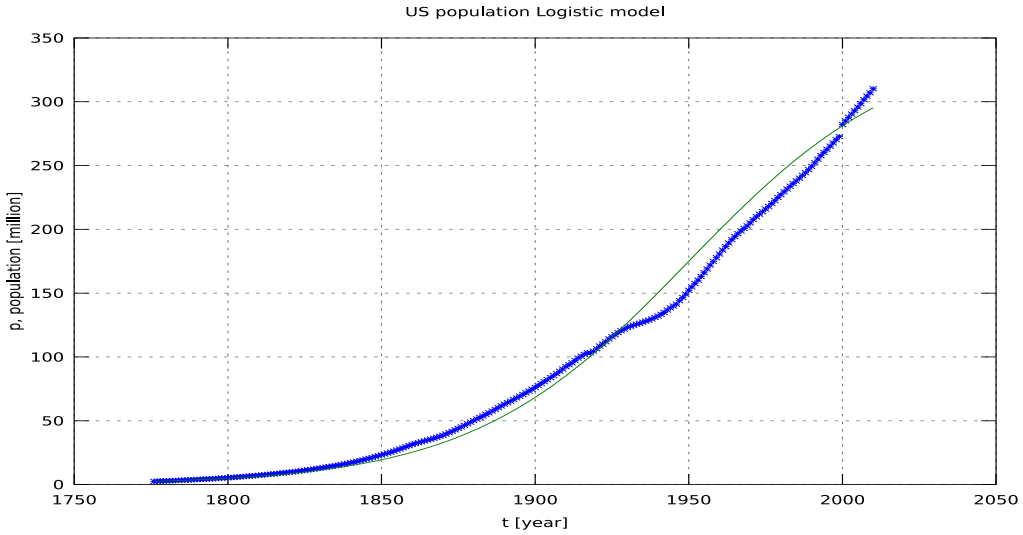


Σχήμα 1.31: Σχετικός ρυθμός ελάττωσης έναντι πληθυσμού και η ευθεία γραμμή με μέθοδο ελαχίστων τετραγώνων.

```
% relative growth rate for logistic model
n = length(t);
rate = relg(t,p);
tt = t(2:n-1);
pp = p(2:n-1);

lsqfit = polyfit(pp,rate,1)
pp2 = lsqfit(2) + lsqfit(1)*pp;

figure()
plot(pp,rate,'*', "markersize", 2,pp,pp2); grid
title("Relative growth rate - Logistic model");
xlabel("p [million]"); ylabel("relative growth rate [1/year]");

K = lsqfit(2)
N = -K/lsqfit(1)

% logistic model
figure()
p3 = N./(1-(1-N/p0).*exp(-K.*(t-t0)));
plot(t,p,'*', "markersize", 2,t,p3); grid
title("US population Logistic model");
xlabel("t [year]"); ylabel("p, population [million]");
```

Υπολογίσαμε $K = 0.028375$ και $N = 348.70$ εκατομμύρια. Η τιμή του N έχει ενδιαφέρον γιατί είναι η μέγιστη σταθερή τιμή για τον πληθυσμό της χώρας σύμφωνα με αυτό το μοντέλο. Τιμή μεγαλύτερη από αυτή εγκυμονεί κινδύνους ισχυρών εσωτερικών κοινωνικών αναταραχών καθώς δεν υπάρχουν αρκετοί πόροι να καλύψουν τις ανάγκες του πληθυσμού. Από το διαδίκτυο βλέπουμε ότι ο τωρινός πληθυσμός των ΗΠΑ είναι 324.4 εκατομμύρια, κοντά στην οριακή τιμή του μοντέλου.

Στην τελευταία γραφική (Σχ. 1.32) έχουμε τα δεδομένα μαζί με το λογιστικό μοντέλο που μόλις υπολογίσαμε. Βλέπουμε καλή συμφωνία στην αρχή με κάποιες μικρές αποκλίσεις αργότερα. Ένας καλός ιστορικός θα μπορούσε να συσχετίσει αυτές τις αποκλίσεις με γεγονότα της εκάστοτε εποχής.



Σχήμα 1.32: Δεδομένα πληθυσμού ΗΠΑ με λογιστικό μοντέλο.

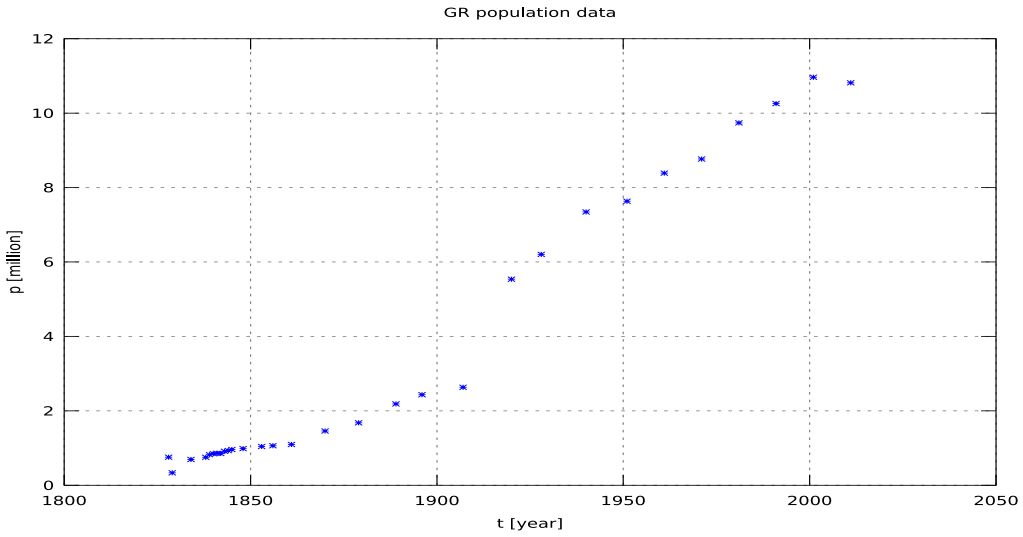
1.19.2 Πληθυσμός Ελλάδος

Σαν Έλληνες προφανώς μας δημιουργείται η περιέργεια να δούμε τι γίνεται με τη δική μας χωρα. Από το διαδίκτυο μπορεί κανείς να αντλήσει πληροφορίες για τις απογραφές πληθυσμού της Ελλάδος (google:απογραφες πληθυσμου στην ελλαδα). Η ΕΛΣΤΑΤ (Ελληνική Στατιστική Αρχή, `http://www.statistics.gr`) παρέχει πολλές και πυκνές πληροφορίες και χάνεται κανείς αν αναζητά μόνο έναν πίνακα πληθυσμού της χώρας ανά τον χρόνο. Η el.wikipedia είναι εδώ πιο χρήσιμη και μπορούμε να βρούμε απογραφές πληθυσμού από το 1828 μέχρι το 2011. Παρέχονται εδώ στο αρχείο GRpop.csv. Υπόψη βέβαια ότι η Ελλάδα ξεκίνησε σαν χώρα τα νεώτερα χρόνια από ένα μικρό κρατίδιο με Πελοπόννησο, Στερεά Ελλάδα και λίγα νησιά, με πολλούς πολέμους και καταστροφές, σε πλήρη αντίθεση με το μοντέλο του Malthus.

Ο παρακάτω κώδικας διαβάζει τα δεδομένα και δείχνει την πρώτη γραφική (Σχ. 1.33). Ο χρόνος t είναι σε έτη και κάνουμε τον πληθυσμό να είναι και εδώ σε εκατομμύρια. Τα δεδομένα είναι κατά αύξουσα σειρά και δεν χρειάζεται αναστροφή.

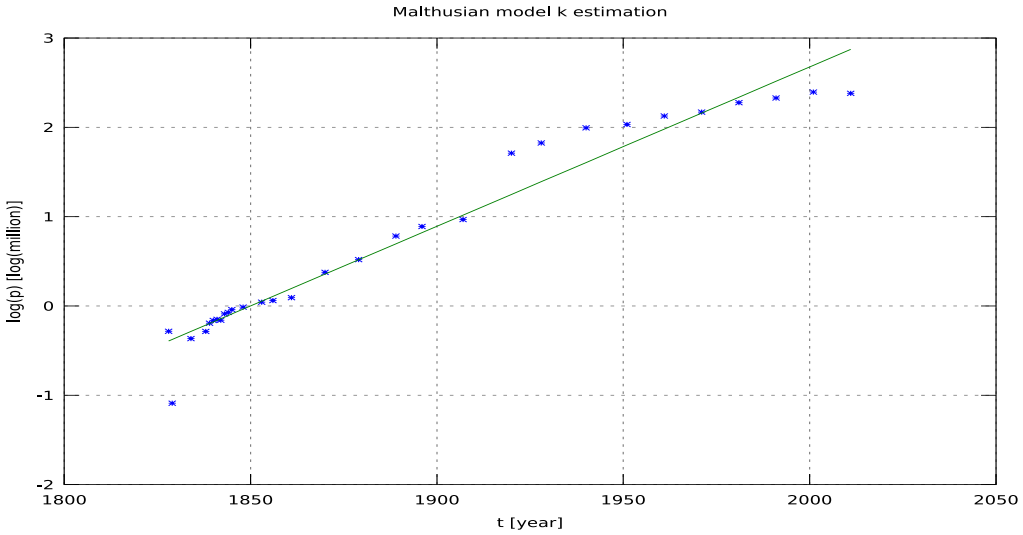
Διακρίνονται δυο περιοχές μεταξύ 1828-1907 και από 1920-2011. Περίοδος Βαλκανικών πολέμων / Μικρασιατική καταστροφή με ότι αυτό συνεπάγεται. Η αύξηση πληθυσμού προέρχεται από την αύξηση εδαφών και την εισροή προσφύγων μετά την Μικρασιατική Καταστροφή.

```
load GRpop.csv
gr = GRpop;

t=gr(:,1); p=gr(:,2)*1e-6;
plot(t,p,'*', "markersize", 2); grid
title("GR population data");
xlabel("t [year]"); ylabel("p [million]");
```



Σχήμα 1.33: Δεδομένα πληθυσμού Ελλάδος

Εδώ δεν μπορούμε να πάρουμε 2 σημεία όπως κάναμε με τις ΗΠΑ και να υπολογίσουμε το k του Malthus. Λογαριθμίζουμε τα δεδομένα και εφαρμόζουμε ελάχιστα τετράγωνα (Σχ. 1.34).



Σχήμα 1.34: Υπολογισμος k του Malthus με ελάχιστα τετράγωνα.

```
figure()
lsqfit = polyfit(t,log(p),1)
pp = lsqfit(2) + lsqfit(1)*t;
plot(t,log(p),'*', "markersize", 2,t,pp); grid
title("Malthusian model k estimation");
xlabel("t [year]"); ylabel("log(p) [log(million)]");
k = lsqfit(1)
```

Η τιμή του k που υπολογίζεται είναι $k = 0.017842$. Εφαρμογή στο μοντέλο Malthus γίνεται με τον κώδικα:

```
t0=t(1)
p0=p(1)
figure()
plot(t,p,'*', "markersize", 2, t, p0*exp(k*(t-t0)),'-'); grid
title("GR population Malthusian model");
xlabel("t [year]"); ylabel("p, population [million]");
```

και το αποτέλεσμα φαίνεται στο Σχ. 1.35. Το πρώτο τμήμα που έχει περισσότερα δεδομένα υπερισχύει του δευτέρου και δείχνει καλή συμφωνία με το μοντέλο Malthus. Στο δεύτερο τμήμα όμως η αντίθεση είναι κραυγαλέα. Είναι σαφές ότι το μοντέλο όπως έχει δεν φαίνεται ικανοποιητικό.

Προχωράμε στο λογιστικό μοντέλο.

```
% relative growth rate for logistic model
n = length(t);
rate = relg(t,p);
tt = t(2:n-1);
pp = p(2:n-1);

lsqfit = polyfit(pp,rate,1)
pp2 = lsqfit(2) + lsqfit(1)*pp;

figure()
plot(pp,rate,'*', "markersize", 2,pp,pp2); grid
title("Relative growth rate - Logistic model");
xlabel("p [million]"); ylabel("relative growth rate [1/year]");

K = lsqfit(2)
N = -K/lsqfit(1)

% logistic model
figure()
p3 = N./(1-(1-N/p0).*exp(-K.*(t-t0)));
plot(t,p,'*', "markersize", 2,t,p3); grid
title("GR population Logistic model");
xlabel("t [year]"); ylabel("p, population [million]");
```

Η συμφωνία μετρήσεων και μοντέλου (σχ. 1.37) είναι καλύτερη αλλά και πάλι όχι τελείως ικανοποιητική. Το ενδιαφέρον βέβαια είναι στη συνολική τιμή πληθυσμού που μπορεί να ζήσει στη χώρα, $N = 13.7$ εκατομμύρια.

Μια παραλλαγή για ίσως καλύτερη συμφωνία είναι να λάβουμε υπόψη και τις εδαφικές μεταβολές. Ανατρέχουμε πάλι στα δεδομένα πληθυσμού και καταγράφουμε και την εκάστοτε έκταση της χώρας. Εφαρμόζουμε το μοντέλο για την πυκνότητα πληθυσμού αντί του πληθυσμού και επαναλαμβάνουμε. Η κρίσιμη τιμή πληθυσμού είναι τώρα $N = 12.264$ εκατομμύρια. Παρόλα αυτά η συμφωνία μετρήσεων και μοντέλου δεν φαίνεται καλύτερη. Η ανταλλαγή πληθυσμών στη Μικρασιατική καταστροφή επηρεάζει πολύ τις μετρήσεις.

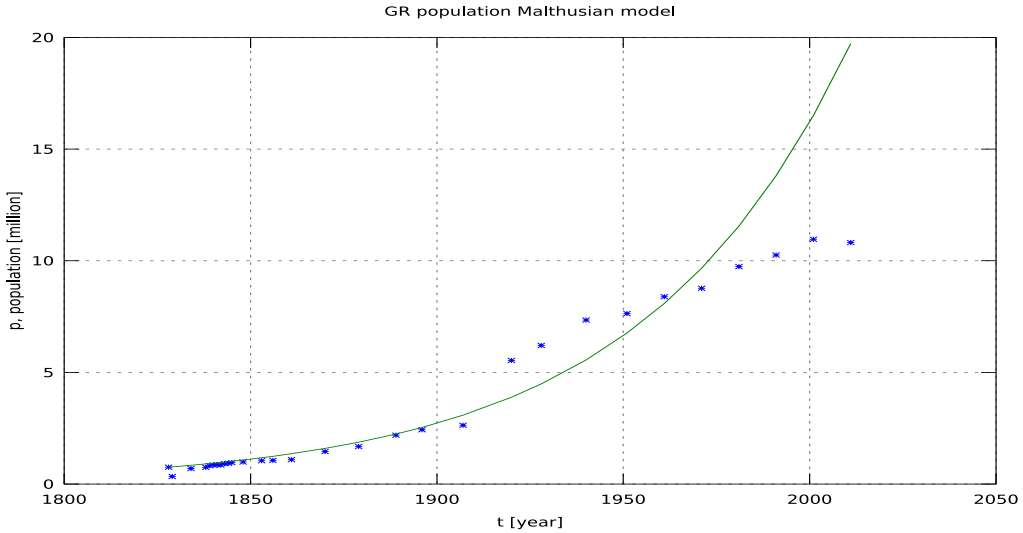
Τελικά, χειριζόμαστε τις δυο ομάδες πληθυσμού (πριν 1910 και μετά) ξεχωριστά. Η συμφωνία τότε μετρήσεων και μοντέλου φαίνεται πολύ καλύτερη. Ένα λεπτό σημείο εδώ είναι η επιλογή t_0, p_0 στο μοντέλο Malthus από τα ελάχιστα τετράγωνα που θα χρειαστούν και στο λογιστικό μοντέλο. Οι κρίσιμες τιμές πληθυσμού είναι τώρα $N_1 = 13.064$ και $N_2 = 12.998$ εκατομμύρια αντίστοιχα.

Η τελευταία απογραφή του 2011 ήταν 10.8 εκατομμύρια και ήταν επιλεκτική ως προς το ποιους κατέγραψε. Τα τελευταία λίγα χρόνια έχουμε επίσης μεγάλη εισροή και εκροή πληθυσμών στη χώρα μας. Είδομεν.

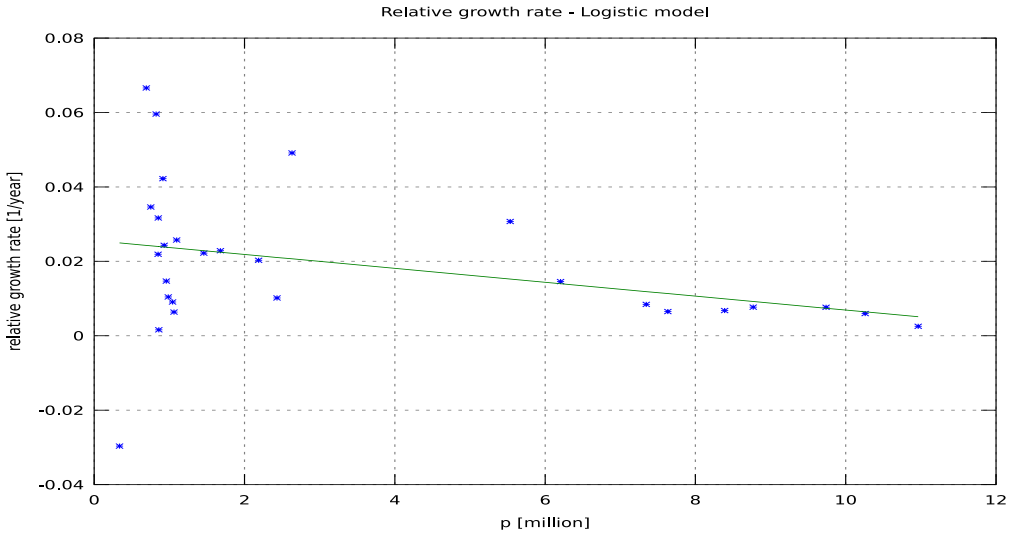
1.20 Παράδειγμα πιο σύνθετου προγράμματος

Βλέπουμε εδώ ένα πιο σύνθετο πρόγραμμα όπου έχουμε «σπάσει» επιμέρους διαδικασίες σε functions και ελέγχουμε τη ροή διαδραστικά. Το πρόγραμμα χρησιμοποιεί την εντολή switch, κατάλληλη για πολλαπλά επιμέρους βήματα μέσα σε while loop που μας επιτρέπει να δοκιμάσουμε τα ίδια βήματα ξανά και ξανά με όποιες παραλλαγές θέλουμε. Δοκιμάστε το και πειραματιστείτε. Χρησιμοποιείστε το help για όποιες εντολές δεν καταλαβαίνετε και εξηγήστε εσείς τι κάνει το πρόγραμμα.

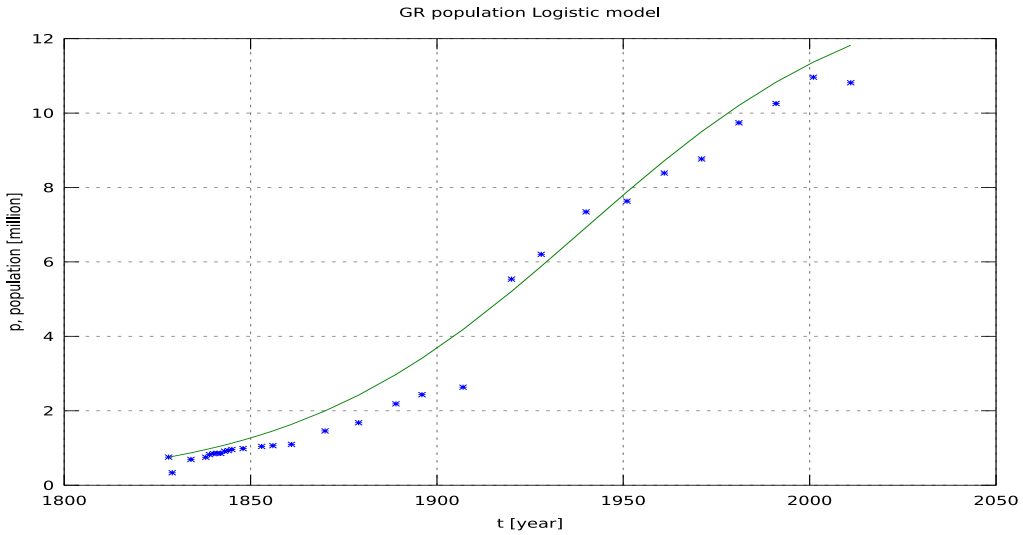
```
% Ορισμός Προγράμματος:
% main:
% choice:
% 1- τι κάνει το πρόγραμμα
% 2- ορισμός ορίων αξόνων στη γραφική
```



Σχήμα 1.35: Πληθυσμός Ελλάδος κατά Malthus



Σχήμα 1.36: Σχετικός ρυθμός ελάττωσης έναντι πληθυσμού και η ευθεία γραμμή με μέθοδο ελαχίστων τετραγώνων.



Σχήμα 1.37: Δεδομένα πληθυσμού Ελλάδος με λογιστικό μοντέλο.

```
% 3- παράσταση έλλειψης όπως ορίζεται από τον χρήστη
% 4- φορτώνει αρχείο με ελλείψεις και τις σχεδιάζει
% 5- ζητά τη γραφική ανάλυση των παραστάσεων - default 200 σημεία
% 6- καθαρίζει την γραφική
% 7- έξοδος

clear all; close all;

Npoints = 200;
figure(1)
hold on

choice = 1;
while choice ~= 7

  switch choice
    case 1
      explainProgram;
    case 2
      myAxes = askAxes;
      modifyAxes (myAxes);
    case 3
      ellipse = askEllipse;
      plotEllipse(ellipse, Npoints);
    case 4
      loadFileAndPlot(Npoints);
    case 5
      Npoints = askPrecision;
    case 6
      clf
  end

  choice = menu('Κάνε τις επιλογές σου', '1 - τι κάνει το πρόγραμμα', ...
                '2 - αλλαγή αξόνων', '3 - ζωγράφισε μια έλλειψη', ...
                '4 - ζωγράφισε ελλείψεις από αρχείο', ...
                '5 - αλλαγή ανάλυσης','6 - καθάρισε τη γραφική', ...
                '7 - έξοδο από πρόγραμμα');

end
```

Οι επιμέρους συναρτήσεις σε εξωτερικά αρχεία είναι:

```
function explainProgram ()
% Αυτή η συνάρτηση εξηγεί το πρόγραμμα

  disp('επιλογές του προγράμματος:')
  disp('  εξήγηση του προγράμματος')
  disp('  ορισμός ορίων αξόνων στη γραφική')
  disp('  παράσταση έλλειψης όπως ορίζεται από τον χρήστη')
  disp('  φορτώνει αρχείο με ελλείψεις και τις σχεδιάζει')
  disp('  ζητά τη γραφική ανάλυση των παραστάσεων - default 200 σημεία')
  disp('  καθαρίζει την γραφική')
  disp('  έξοδος από πρόγραμμα')
end


function [myAxes] = askAxes ()
% Ρωτά χρήστη τα όρια των αξόνων της γραφικής
% Εϊναι στη μορφή [minX maxX minY maxY]

  myAxes = zeros(1,4);
  disp('Δώσε τα όρια των αξόνων της γραφικής')
  myAxes(1) = input('Min X = ');
  myAxes(2) = input('Max X = ');
  myAxes(3) = input('Min Y = ');
  myAxes(4) = input('Max Y = ');
end


function [] = modifyAxes (myAxis)
% Τροποποίησε τους άξονες
  axis(myAxis);
end


function [ellipse] = askEllipse ()
% Ρωτά τον χρήστη για την έλλειψη
% Είναι στη μορφή [c1 c2 a1 a2]

  ellipse = zeros(1,4);
  disp('Δώσε τις παραμέτρους της έλλειψης:')
```

```
  ellipse(1) = input('Κέντρο, συντεταγμένη x = ');
  ellipse(2) = input('Κέντρο, συντεταγμένη y = ');
  ellipse(3) = input('Ημιάξονας x = ');
  ellipse(4) = input('Ημιάξονας y = ');
end


function plotEllipse (ellipse, Npoints)
% σχεδιάζει μια έλλειψη με Npoints σημεία που ορίζονται από την εξίσωση
% τηε έλλειψης (x-c1).^2/a1^2 + (y-c2).^2/a2^2 = 1
% οι παράμετροι δίδονται ως [c1 c2 a1 a2]

  c1 = ellipse(1);
  c2 = ellipse(2);
  a1 = ellipse(3);
  a2 = ellipse(4);

% παράμετρος της έλλειψης
  t = linspace(0,2*pi,Npoints);

% παραμετρική μορφή της έλλειψης
  plot(c1+a1*sin(t), c2+a2*cos(t));
end


function [] = loadFileAndPlot (Npoints)
% φορτώνει αρχείο με παραμέτρους ελλείψεων και τις σχεδιάζει

  AllEllipses = load('cheeseEllipse.txt');

  for ellipse = AllEllipses'
    plotEllipse(ellipse, Npoints);
  end
end


function [Npoints] = askPrecision()
% Προδιόρισε τον αριθμο σημείων

  Npoints = input('Δώσε αριθμό σημείων: ');
end


% εξωτερικό text αρχείο cheeseEllipse.txt
% Ορισμός ελλείψεων
% ellipse: (x-c1).^2/ a1^ + (y-c2).^2/a2 = 1
% κάθε γραμμή αντιπροσωπεύει μια 'ελλειψη
% [c1 c2 a1 a2]
    90, 300, 82, 70
    92, 65, 12, 12
    80, 150, 12, 12
    290, 38, 12, 12
    458, 127, 12, 12
    246, 251, 20,16
    175, 180, 47, 37
    337, 169, 75, 50
    430, 257, 41, 32
    394, 353, 25, 30
    524, 280, 14, 22
    244, 355, 24, 17
    290, 355, 24, 17
```

1.21 Ασκήσεις

Άσκηση 1.1 *Έστω $x = 2, y = 3$. Υπολογίστε τις παρακάτω εκφράσεις και ελέγξτε το αποτέλεσμα με το κομπιουτεράκι σας.*

1) $x + y$	*2)* xy	*3)* x/y	*4)* $\sin x$	*5)* $8\cos xy$
6) $5\tan(x/y)$	*7)* $3\pi x^2$	*8)* e^{-xy^2}	*9)* $\dfrac{yx^3}{x-y}$	*10)* $\dfrac{3x}{2y}$
11) $\dfrac{5}{2}xy$	*12)* $\dfrac{x^4}{x^4-1}$	*13)* $\left(1-\dfrac{1}{x^5}\right)^{-1}$	*14)* $\dfrac{3y}{4x-8}$	*15)* $\dfrac{4(y-5)}{3x-6}$
16) $\dfrac{3x^3}{4xy-8x^2}$	*17)* $6x^5+\dfrac{4}{x}$	*18)* $\dfrac{x}{4^3}+\dfrac{y}{5^6x}$	*19)* $\dfrac{2\sin x}{5}$	*20)* $6(x^{1/3})+5x^{0.62}$

Άσκηση 1.2 *Έστω $x = 2 + j5$, $y = -3 - j7$. Υπολογίστε τις παρακάτω εκφράσεις και ελέγξτε το αποτέλεσμα με το κομπιουτεράκι σας.*

1) $x + y$	*2)* xy	*3)* x/y	*4)* e^x	*5)* $\sqrt{y}$
6) xy^2	*7)* $\dfrac{4(y-5)}{3x-6}$	*8)* e^{-xy^2}	*9)* $\dfrac{3x^3}{4xy-8x^2}$	*10)* $6x^5+\dfrac{4}{x}$

Άσκηση 1.3 *Έστω ότι η μεταβλητή x παίρνει τις τιμές $x = 1, 1.2, 1.4, ...5$. Υπολογίστε το διάνυσμα $y = 7\sin(4x)$. Πόσα στοιχεία έχει αυτό το διάνυσμα και ποιά είναι η τιμή του τρίτου στοιχείου; (Προφανώς εδώ δεν ζητώ να δείξετε όλες τις άλλες τιμές του διανύσματος.)*

Άσκηση 1.4 *Πόσα στοιχεία έχει το διάνυσμα* `[sin(-pi/2):0.05:cos(0)]` *και ποιά είναι η τιμή του 10ου στοιχείου; (Προφανώς εδώ δεν ζητώ να δείξετε όλες τις άλλες τιμές του διανύσματος.)*

Άσκηση 1.5 *Ποιές είναι οι ρίζες των πολυωνύμων $p_1(x) = 13x^3 + 182x^2 - 184x + 2503$ και $p_2(x) = 36x^3 + 12x^2 - 5x + 10$;*

Άσκηση 1.6 *Κάνετε τη γραφική παράσταση της συνάρτησης $T = 6\ln t - 7e^{0.2t}$ στο διάστημα $1 \le t \le 3$. Δώστε τίτλο στην παράσταση και υπότιτλους στους άξονες. Η μεταβλητή T συμβολίζει θερμοκρασία σε βαθμούς $^\circ C$ και η μεταβλητή t, χρόνο σε* min.

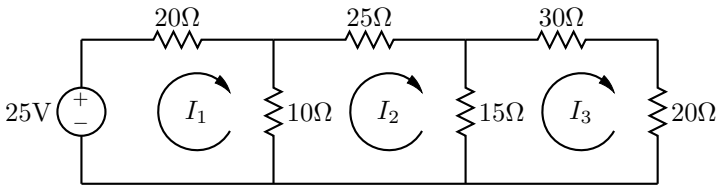
Άσκηση 1.7 *Κάνετε τη γραφική παράσταση των συναρτήσεων $u = 2\log_{10}(60x+1)$ και $v = 3\cos(6x)$, στο ίδιο διάγραμμα, για $0 \le x \le 2$. Δώστε τίτλο στην παράσταση και υπότιτλους στους άξονες. Οι μεταβλητές u, v συμβολίζουν ταχύτητα σε* km/h *και η x, διάστημα σε* km.

Άσκηση 1.8 *Σχεδιάστε τα πολυώνυμα $y = 3x^4 - 6x^3 + 8x^2 + 4x + 90$ και $z = 3x^3 + 5x^2 - 8x + 70$ στην ίδια γραφική παράσταση για $-3 \le x \le 3$. Δώστε τίτλο στην παράσταση και υπότιτλους στους άξονες. Οι μεταβλητές y, z συμβολίζουν ρεύμα σε* mA *και η x, τάση σε* V.

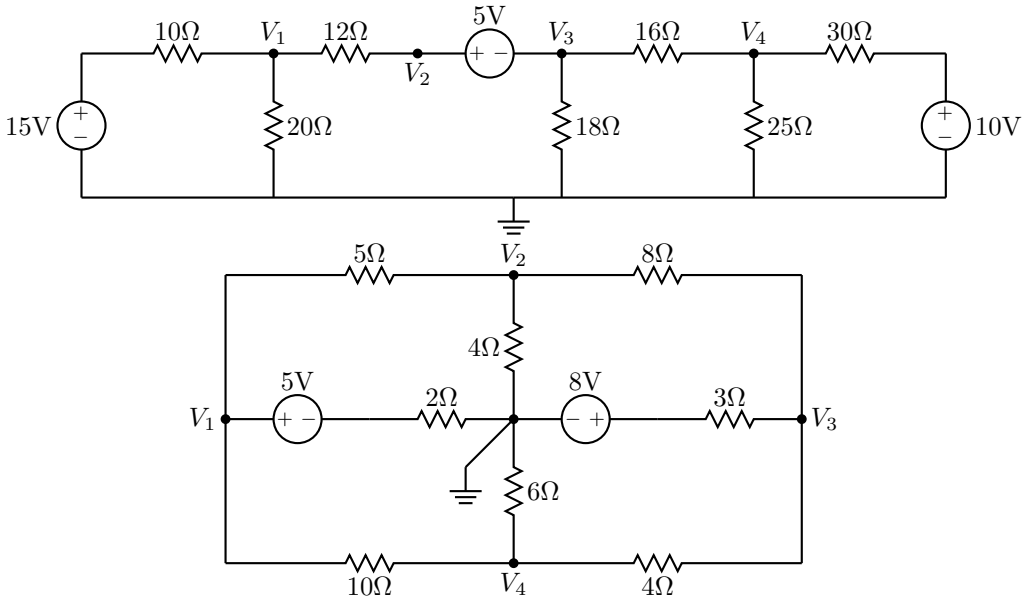
Άσκηση 1.9 *Να λύσετε το παρακάτω σύστημα εξισώσεων.*

$$\begin{aligned} 7x + 14y - 6z &= 95 \\ 12x - 5y + 9z &= -50 \\ -5x + 7y + 15z &= 145 \end{aligned}$$

Άσκηση 1.10 *Να βρείτε τα ρεύματα οφθαλμών στο παρακάτω κύκλωμα.*



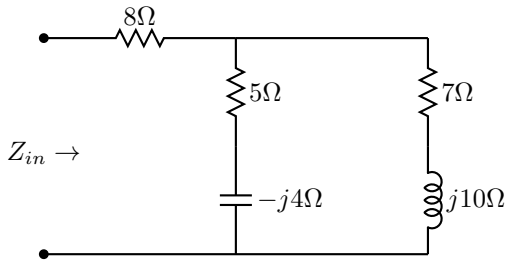
Άσκηση 1.11 *Χρησιμοποιώντας κανόνες Kirchhoff να βρείτε τις κομβικές τάσεις V_1, V_2, V_3, V_4 των παρακάτω κυκλωμάτων.*



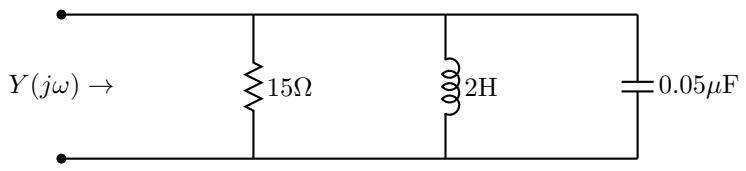
Άσκηση 1.12 *Απλοποιήστε τις παρακάτω εκφράσεις σε καρτεσιανή και πολική μορφή.*

$$Z_1 = 10 + j12 + \frac{(20 + j40)(5 - j15)}{25 + j25} \quad Z_2 = \frac{10(-5 + j13)(4 + j4)}{(1 + j2)(2 + j5)(-5 + j3)}$$
$$Z_3 = 0.2 + j7 + 4.7e^{j0.5} + (2 + j3)e^{-j0.6\pi}$$

Άσκηση 1.13 *Να βρεθεί η σύνθετη αντίσταση εισόδου στο παρακάτω κύκλωμα.*



Άσκηση 1.14 *Να βρεθεί η σύνθετη αγωγιμότητα εισόδου σε πολική μορφή στο παρακάτω κύκλωμα για τις συχνότητες* $1\ \mathrm{kHz}$, $4\ \mathrm{kHz}$, $7\ \mathrm{kHz}$ *και* $10\ \mathrm{kHz}$. *Να σχεδιαστεί κατόπιν η γραφική παράσταση του μέτρου (σε dB) και της φάσης (σε μοίρες) για* $0.5\ \mathrm{kHz} < f < 20\ \mathrm{kHz}$. *Οι γραφικές παραστάσεις να αποθηκευτούν σε κατάλληλα αρχεία γραφικών και να έχουν πλέγμα, τίτλο και κατάλληλους υπότιτλους με διαστάσεις στους άξονες. (Θα σας χρειαστεί η κατάλληλη χρήση των τελεστών* `./` *και* `.*` *για να σαρώσετε όλες τις συχνότητες.)*



Άσκηση 1.15 *Στο φάκελο* `C:\emgt` *υπάρχει το αρχείο* `diode` *το οποίο περιέχει σε δυο στήλες την τάση v_D (πρώτη στήλη) σε V και το ρεύμα i_D (δεύτερη στήλη) σε nA. Να φορτώσετε τα δεδομένα στο Octave και να σχεδιαστεί η κατάλληλη γραφική παράσταση i_D έναντι v_D με πλέγμα, τίτλο και κατάλληλους υπότιτλους με διαστάσεις στους άξονες. Αν*

$$i_D = I_s \exp\left(\frac{v_D}{nV_T}\right)$$

όπου $n = 1.5$, να υπολογείστε από τα δεδομένα τις τιμές των I_s και V_T. (Θα χρειαστεί να λογαριθμήσετε κατάλληλα για να γίνει γραμμική η συνάρτηση και εξετάστε την κλίση της ευθείας καθώς και τα σημεία τομής με τους άξονες.)

Άσκηση 1.16 *Με τη μέθοδο Newton υπολογίστε τη ρίζα ή ρίζες των συναρτήσεων*

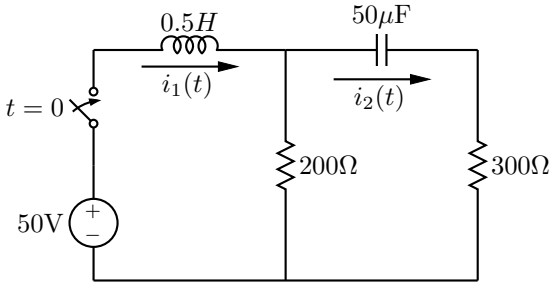
$$f(x) = x^3 - 5.656609x^2 - 0.6765x + 22.348 \qquad \text{και} \qquad g(x) = \cos(x) + 2\sin(x) + x^2$$

Άσκηση 1.17 *Το αρχείο exon.d περιέχει μετρήσεις από κάποιο φυσικό μέγεθος που περιγράφεται θεωρητικά από σχέση της μορφής $a\exp(-\rho x)$. Χρησιμοποείστε τη μέθοδο ελαχίστων τετραγώνων για να εκτιμείστε τις τιμές των παραμέτρων a και ρ και παρουσιάστε τις μετρήσεις (με σημεία) και το θεωρητικό μοντέλο (συνεχής γραμμή) στην ίδια γραφική παράσταση.*

Άσκηση 1.18 *Υπολογίστε τα παρακάτω ολοκληρώματα με τις μεθόδους τραπεζίου και Simpson. Δοκιμάστε επίσης και τις συναρτήσεις* `trapz` *και* `quadv` *του Octave.*

$$\int_0^1 (1-x^2)dx \qquad \int_0^1 \frac{1}{1+x^2}\,dx \qquad \int_{-1}^1 e^{-x^2}dx \qquad \int_0^2 [2+\cos(2\sqrt{x})]dx$$

Άσκηση 1.19 *Στο παρακάτω κύκλωμα υπολογίστε τα ρεύματα $i_1(t), i_2(t)$ για $0 \le t \le 0.1$ sec δοθέντος ότι όλα τα ρεύματα και τάσεις στο κύκλωμα είναι μηδενικά πριν το κλείσιμο του διακόπτη την χρονική στιγμή $t = 0$.*



Άσκηση 1.20 *Λύστε την παρακάτω διαφορική εξίσωση για $0 \le t \le 3$*

$$y'' + 4y' + 5y = 2e^t \qquad \mu\varepsilon \qquad y(0) = y'(0) = 0$$

Άσκηση 1.21 *Λύστε την παρακάτω διαφορική εξίσωση για $0 \le t \le 5$*

$$y'' + 2y' + 13y = te^t \qquad \mu\varepsilon \qquad y(0) = 0, y'(0) = 2$$

Άσκηση 1.22 *Έστω η συνάρτηση $f(x) = \sin(x) + 3\cos(x)$.*

1. *Κάντε τη γραφική της μεταξύ -4 και 4 με βήμα 0.1. Ποια η τιμή της $f'(0)$;*
2. *Θυμηθείτε τον ορισμό της παραγώγου*
$$\lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$
Πάρτε ένα διάνυσμα τιμών h, μεγέθους $N = 100$, από 10^{-20} έως 10^{-2} (θυμηθείτε logspace) και χρησιμοποιώντας τον ορισμό φτιάξτε γραφική σε λογαριθμική κλίμακα (loglog) του απόλυτου σφάλματος μεταξύ της σωστής τιμής $f'(0)$ και αυτής που υπολογίζετε αριθμητικά.
3. *Επαναλάβετε με τον εναλλακτικό ορισμό*
$$\lim_{h\to 0} \frac{f(x+h/2) - f(x-h/2)}{h}$$
4. *Αντί να χρησιμοποιήσετε την συνάρτηση $f(x)$ αυτούσια στον εναλλακτικό ορισμό, κάντε τις διαφορές και απλοποιήσεις για $x \to 0$ και επαναλάβετε.*

Τι παρατηρείτε;

Άσκηση 1.23 *Επιβεβαιώστε τους ισχυρισμούς που αναφέρθηκαν παραπάνω για το ελληνικό μοντέλο όταν το διαχωρίσετε σε δυο ομάδες. Παρουσιάστε τις αντίστοιχες γραφικές των μοντέλων Malthus και λογιστικού με πλήρη αιτιολόγηση.*