

Προγραμματισμός με Εφαρμογές
στην
Επιστήμη του Μηχανικού
Μέρος Α΄

Για το αντίστοιχο μάθημα του
4ου εξαμήνου της Σχολής ΕΜΦΕ
Εθνικό Μετσόβιο Πολυτεχνείο
Κωνσταντίνος Ν. Αναγνωστόπουλος

31 Αυγούστου 2010

Περιεχόμενα

1	Ο Υπολογιστής	1
1.1	Το Λειτουργικό Σύστημα	3
1.1.1	Filesystem	4
1.1.2	Εντολές	11
1.1.3	Αναζητώντας Βοήθεια	14
1.2	Εργαλεία Επεξεργασίας Κειμένου – Φίλτρα	16
1.3	Ο Καλύτερος Φίλος του Ανθρώπου	21
1.3.1	Καλώντας τον Emacs	22
1.3.2	Αλληλεπιδρώντας με τον Emacs	24
1.3.3	Βασική Επεξεργασία Κειμένου	27
1.3.4	Κόβοντας και ράβοντας	29
1.3.5	Παράθυρα	31
1.3.6	Αρχεία και Buffers	32
1.3.7	Modes	33
1.3.8	Βοήθεια στον Emacs	34
1.3.9	Παραμετροποίηση του Emacs	36
1.3.10	Ελληνικά στον Emacs	37
1.4	Η Γλώσσα Προγραμματισμού: Fortran 77	38
1.4.1	Τα Στοιχειώδη	38
1.4.2	Οι λεπτομέρειες	49
1.5	Κοιτάζοντας τα Αποτελέσματα	55
1.6	Turbo:Σενάρια Φλοιού	60
2	Περιγραφή της Κίνησης	75
2.1	Κίνηση στο Επίπεδο	75
2.1.1	Απεικόνιση των Δεδομένων	85
2.1.2	Άλλα Παραδείγματα	88
2.2	Κίνηση στο Χώρο	100

2.3	Κίνηση μετ' Εμποδίων	111
2.3.1	Το Μονοδιάστατο Κουτί	112
2.3.2	Σφάλματα	120
2.3.3	Το Δισδιάστατο Κουτί	125
2.4	Εφαρμογές	130
2.5	Ασκήσεις	142
3	Κίνηση Σωματιδίου	147
3.1	Αριθμητική Ολοκλήρωση Εξισώσεων Νεύτωνα	147
3.2	Πρελούδιο: Μέθοδοι Euler	148
3.3	Μέθοδοι Runge–Kutta	162
3.3.1	Προγραμματισμός της Runge–Kutta 4ης τάξης	167
3.4	Σύγκριση των Μεθόδων	172
3.5	Ο Αρμονικός Ταλαντωτής με Απόσβεση και Εξωτερική Δύναμη.	175
3.6	Το Εκκρεμές με Απόσβεση και Εξωτερική Δύναμη.	185
3.7	Παράρτημα: Στη Μέθοδο Euler–Verlet	194
3.7.1	Παράρτημα: Runge–Kutta 2ης τάξης	196
3.8	Ασκήσεις	199
4	Κίνηση στο Επίπεδο	203
4.1	Runge–Kutta στις δύο διαστάσεις.	203
4.2	Βολές στο Βαρυτικό Πεδίο της Γης.	209
4.3	Κίνηση Πλανητών.	216
4.4	Σκέδαση.	222
4.4.1	Σκέδαση Rutherford.	226
4.4.2	Σκέδαση σε Άλλα Δυναμικά Πεδία.	234
4.5	Ασκήσεις	241
5	Ηλεκτροστατική	245
5.1	Ηλεκτροστατικό Πεδίο Σημειακών Ηλεκτρικών Φορτίων	245
5.2	Το Πρόγραμμα – Ορεκτικά και ... επιδόρπιο	248
5.3	Το Πρόγραμμα - Το Κυρίως Πιάτο	262
5.4	Το Πρόγραμμα - Σύνοψη	269
5.5	Ηλεκτροστατικό Πεδίο στο Κενό	276
5.6	Αποτελέσματα	285
5.7	Εξίσωση Poisson	286
5.8	Ασκήσεις	293

Ο τόμος αυτός έχει γραφτεί με την υπόθεση ότι ο αναγνώστης **εκτελεί τις εντολές και τις διαδικασίες που περιγράφονται ταυτόχρονα με την ανάγνωση του κειμένου**. Αν αυτό δε γίνεται, η βοήθεια από το κείμενο θα είναι εξαιρετικά ελλειπής.

Στην ιστοσελίδα του μαθήματος¹ παρέχεται συνοδευτικό λογισμικό το οποίο, μεταξύ άλλων, περιέχει τους κώδικες που περιγράφονται στο κείμενο.

Μερικές συμβάσεις: Κείμενο με γραμματοσειρά όπως η παρακάτω αφορά εντολές στον υπολογιστή, είσοδο και έξοδο προγραμμάτων, κώδικα σε Fortran ή άλλη γλώσσα, ονόματα αρχείων:

```
> echo Hello world
Hello world
```

Όταν μια γραμμή αρχίζει με το χαρακτήρα ‘προτροπής’ (prompt)

```
>
```

όπως παραπάνω, είναι μία *εντολή* που δίνουμε στη γραμμή εντολών του φλοιού. Η δεύτερη γραμμή παραπάνω είναι αυτά που τυπώνει η εντολή στην κονσόλα. Το `file.f` είναι το όνομα ενός αρχείου.

Παρακάτω δίνονται τα περιεχόμενα ενός αρχείου Fortran:

```
program add

  z = 1.0
  y = 2.0
  x = z + y
  print *, x

end
```

¹<http://mycourses.ntua.gr/courses/SEMF1079/>

Τι χρειάζεστε για να δουλέψετε στον υπολογιστή σας:

- Ένα λειτουργικό σύστημα τύπου GNU/Linux και τα βασικά εργαλεία του.
- Ένα μεταγλωττιστή (compiler) για τη γλώσσα Fortran. Ο μεταγλωττιστής gfortran διατίθεται ελεύθερα υπό άδεια ελεύθερου λογισμικού².
- Ένα προηγμένο πρόγραμμα επεξεργασίας κειμένου κατάλληλο για προγραμματιστές, όπως ο Emacs³
- Ένα καλό πρόγραμμα, κατάλληλο για ανάλυση δεδομένων, για να κάνετε γραφικές παραστάσεις όπως το gnuplot⁴.
- Το φλοιό tcsh⁵.
- Τα προγράμματα gawk⁶, grep, sort, cat, head, tail, less. Βεβαιωθείτε ότι είναι στη διάθεσή σας.

Αν έχετε μια διανομή GNU/Linux εγκατεστημένη στον υπολογιστή σας, η εγκατάσταση του παραπάνω λογισμικού γίνεται συνήθως πολύ εύκολα με τον διαχειριστή πακέτων της διανομής. Α.χ. σε μια διανομή τύπου Debian (Ubuntu, ...) με τις απλές εντολές

```
> sudo apt-get install tcsh emacs gnuplot gnuplot-doc gfortran gawk  
> sudo apt-get install gawk-doc binutils textutils manpages-dev coreutils
```

θα βρείτε όλο το λογισμικό έτοιμα εγκατεστημένο στον υπολογιστή σας.

Αν δε θέλετε να εγκαταστήσετε μια διανομή GNU/Linux στον υπολογιστή σας, έχετε τις εξής εναλλακτικές δυνατότητες:

- Χρησιμοποιήστε το cloud computing του ΕΜΠ⁷. Θα εξαρτάστε από την ταχύτητα του δικτύου.

²<http://www.gfortran.org>

³<http://www.gnu.org/software/emacs/>

⁴<http://www.gnuplot.info>

⁵<http://www.tcsh.org>

⁶<http://www.gnu.org/software/gawk>

⁷<https://cloudfront.central.ntua.gr/>

- Εκκινήστε τον υπολογιστή σας από το DVD ή usb stick με μια live διανομή, όπως από του Ubuntu⁸. Αυτή η επιλογή δε θα αλλοιώσει τίποτα στον υπολογιστή σας, αν και θα τρέχει αργά.
- Εγκαταστήστε στα Microsoft Windows τη διανομή Cygwin⁹. Είναι μια πολύ καλή επιλογή για όσους έχουν εθιστεί στη Microsoft.

Ευχαριστώ τους/τις φοιτητές/τριες που μου υπέδειξαν διορθώσεις στο κείμενο και ιδιαίτερα τον κ. Στέφανο Κουτσομπή.

⁸<http://www.ubuntu.com>

⁹<http://www.cygwin.com>

Κεφάλαιο 1

Ο Υπολογιστής

Σκοπός του κεφαλαίου αυτού είναι να θέσει τα θεμέλια για την ανάπτυξη δεξιοτήτων χρήσης των υπολογιστικών εργαλείων που θα χρησιμοποιήσουμε στη μελέτη των υπολογιστικών προβλημάτων που παρουσιάζονται στα επόμενα κεφάλαια. Δεν έχει σκοπό να κάνει πλήρη και εις βάθος παρουσίαση, είναι μάλλον πρακτική εκμάθηση μέσω παραδειγμάτων. Άλλωστε υπάρχουν πολλές πλήρης και παιδαγωγικές παρουσιάσεις του υλικού που θα παρουσιάσουμε σε πολλά βιβλία ελεύθερα διαθέσιμα στο διαδίκτυο ή/και σε βιβλία τα οποία ... έχουν κάποιο τίμημα. Παρακολουθήστε τη βιβλιογραφία και την ιστοσελίδα του μαθήματος¹.

Όπως σε κάθε περιβάλλον εργασίας ενός υπολογιστικού προγράμματος, είναι ανάγκη να γίνουν επιλογές. Αυτές εξαρτώνται από τις συγκεκριμένες ανάγκες του προγράμματος: Απαιτήσεις αριθμητικής αποτελεσματικότητας, μικρή/μεγάλη ομάδα εργασίας, πολυπλοκότητα κώδικα, ανάγκες για αναβαθμίσεις ... αναμνήσεις από το μέλλον.

Εμείς εδώ θα διαλέξουμε να πάρουμε ένα άρωμα από τις ανάγκες ενός προγράμματος με κατεύθυνση επιστημονική/υπολογιστική. Ενός προγράμματος με μεγάλες ανάγκες σε εκμετάλλευση των υπολογιστικών πόρων για γρήγορους αριθμητικούς υπολογισμούς και για ευέλικτη ανάλυση (...πολλών) δεδομένων. Ένα τέτοιο περιβάλλον που προσφέρει ευελιξία, αξιοπιστία, απλότητα², δυνατό εργαλεία για ανάλυση δεδομένων και μεταγλώττιση προγραμμάτων και που να προσφέρει στο χρήστη να κάνει αποδοτικότερη χρήση των υπολογιστικών πόρων του συστήματός του είναι η ομάδα λειτουργικών συστημάτων Unix. Η

¹<http://mycourses.ntua.gr/courses/SEMFE1079/>

²Ο συγγραφέας ποτέ δεν μπόρεσε να κατανοήσει γιατί άλλα δημοφιλή λειτουργικά συστήματα θεωρούνται...απλούστερα.

σύγχρονη, δημοφιλής και ελεύθερα διαθέσιμη έκδοση τέτοιου συστήματος είναι το GNU/Linux³, μια προσπάθεια η οποία πραγματοποιήθηκε χάρη στην εθελοντική δουλειά εκατομμυρίων προγραμματιστών παγκοσμίως και που βασίστηκε στην ιδέα του Ελεύθερου Λογισμικού (όχι με την έννοια ‘τσάμπα’ αλλά με την έννοια της ελεύθερης διακίνησης ιδεών στο λογισμικό) που θεμελίωσε ο Richard Stallman⁴.

Η γλώσσα προγραμματισμού που θα διαλέξουμε είναι η Fortran 77. Μερικοί λόγοι για την επιλογή είναι ότι η γλώσσα (ή οι εξελίξεις αυτής Fortran 90, 95, ...) αυτή είναι προσανατολισμένη σε αριθμητικές εφαρμογές και χρησιμοποιείται ευρέως σε επιστημονικές και μηχανικές συνεργασίες. Είναι απλή και οι μεταγλωττιστές κάνουν βελτιστοποίηση, παραλληλοποίηση και διανυσματοποίηση αποτελεσματικότερα. Υπάρχουν πολλές, καλές και δοκιμασμένες βιβλιοθήκες με μαθηματικό λογισμικό από τις οποίες μερικές είναι ελεύθερα διαθέσιμες. Φυσικά η γλώσσα αυτή δεν προσφέρεται για πολύπλοκες διεργασίες που έχουν σχέση με το λειτουργικό σύστημα (διαχείριση αρχείων, επεξεργασία δεδομένων κλπ) και επεξεργασία κειμένου αλλά το κενό καλύπτεται εύκολα με το συνδυασμό χρήσης των εργαλείων του συστήματος. Επίσης είναι απλή στη δομή της, οπότε ο αναγνώστης δε θα δυσκολευτεί να κάνει απλούς υπολογισμούς ακόμα και αν δεν έχει προηγούμενη εμπειρία προγραμματισμού. Τέλος είναι μαθηματικά προσανατολισμένη: Έχει απλή, κτισμένη μέσα της, χρήση μιγαδικών αριθμών και μαθηματικών συναρτήσεων, βιβλιοθήκες διαθέσιμες για υπολογισμούς διαφορετικής ακρίβειας και αποτελεσματικότερη διαχείριση της μνήμης του υπολογιστή. Μπορεί κανείς εύκολα να διαχειριστεί αριθμητικά δεδομένα σε ψηφιακή μορφή (unformatted) που είναι γρηγορότερο και διατηρεί την ακρίβεια των πραγματικών αριθμών. Η απλότητά της και η ... ηλικία της κάνει τους αντίστοιχους μεταγλωττιστές να κάνουν την καλύτερη διαθέσιμη βελτιστοποίηση και παραλληλοποίηση του κώδικα σε σύγκριση με όλες τις άλλες γλώσσες. Έχει το μειονέκτημα ότι η μνήμη καθορίζεται στατικά και δε διαθέτει pointers, κάτι όμως που δεν αποτελεί συνήθως σοβαρό εμπόδιο σε αριθμητικούς υπολογισμούς. Είναι γλώσσα δομημένου (procedural) και όχι αντικειμενοστραφούς (object oriented) προγραμματισμού, κάτι όμως που δεν είναι μειονέκτημα σε αριθμητικούς υπολογισμούς καθώς ο πολύπλοκος αντικειμενοστραφής προγραμματισμός μπορεί να οδηγήσει εύκολα σε προγραμματιστικά λάθη ως προς τη βελτιστοποίηση εκτέλεσης του προγράμματος από τον προγραμματιστή ή/και τον μεταγλωττιστή (compiler). Γλώσσες αντικειμενοστραφείς όπως

³www.gnu.org

⁴www.stallman.org

οι C++/Java θα προτιμηθούν σε προγράμματα που ο χρόνος/μνήμη δεν είναι σημαντικοί περιορισμοί αλλά όπου ο χρόνος προγραμματισμού λόγω μεγέθους του προγράμματος ή/και της ομάδας προγραμματισμού ή λόγω φορητότητας (portability) σε διαφορετικές πλατφόρμες είναι σημαντικότερος.

Η Fortran77 όπως και οι Fortran90, C, C++, Java είναι γλώσσες που μεταγλωττίζονται από ένα μεταγλωττιστή. Μια άλλη κατηγορία γλωσσών προγραμματισμού είναι οι ερμηνευόμενες (interpreted) όπως είναι οι perl, Basic, awk, shell programming, Macsyma, Mathematica, Matlab, Octave, Maple, Οι ερμηνευτές των γλωσσών αυτών ερμηνεύουν το πρόγραμμα γραμμή-γραμμή κάτι που δεν επιτρέπει την ανάλυση του προγράμματος που κάνει ο μεταγλωττιστής που οδηγεί σε βελτιστοποιήσεις που κάνουν το πρόγραμμα να τρέχει γρηγορότερα. Οι γλώσσες αυτές είναι απλούστερες στη χρήση (λ.χ. με μία εντολή `Inverse[A]` ή `1/A` παίρνουμε τον αντίστροφο ενός πίνακα κάτι που χρειάζεται περισσότερη δουλειά σε μία γλώσσα όπως η Fortran, C, ...) αλλά γίνονται απαγορευτικά αργές για απαιτητικά προβλήματα. Ο χρόνος προγραμματισμού τους όμως είναι πολύ μικρότερος και ο προγραμματιστής θα πρέπει να θεωρήσει αν μπορεί να λύσει το πρόβλημά του με τη βοήθειά τους προτού αρχίσει να σχεδιάζει ένα πρόγραμμα σε μία γλώσσα όπως η Fortran.

Τέλος αρκετές από τις εντολές του λειτουργικού συστήματος που θα συζητήσουμε παρακάτω, ερμηνεύονται έτσι μόνο από το φλοιό `tcsh`. Αυτή είναι μία ακόμα από τις επιλογές μας και δε θα αναλύσουμε τις διαφορές με άλλους φλοιούς έτσι ώστε η παρουσίαση να μη γίνει πολυπλοκότερη από όσο χρειάζεται.

1.1 Το Λειτουργικό Σύστημα

Έχετε βρεθεί στην κατάσταση να θέλετε να λύσετε ένα πρόβλημα και το πολυδιαφημισμένο και ακριβοπληρωμένο λογισμικό σας που ‘ψήνει και καφέ’ να μην μπορεί να κάνει αυτό που αρχικά δεν προβλέψατε ότι θα ήταν αναγκαίο να γίνει; Η λύση σε αυτό το πρόβλημα είναι ένα περιβάλλον όπου οι διεργασίες που απλές ή πολύπλοκες επαναλαμβάνονται συχνά, να καταμερίζονται σε διαφορετικά εργαλεία. Το περιβάλλον αυτό επιτρέπει εύκολα το συνδυασμό των δυνατοτήτων όλων αυτών των εργαλείων οπότε κάθε φορά που θέλετε να κάνετε κάτι καινούργιο αρκεί να συνδυάσετε με διαφορετικό τρόπο τα εργαλεία αυτά για να πετύχετε τον σκοπό σας. Και αν πάλι βρεθείτε στην κατάσταση να σας λείπει ένα εργαλείο τότε να χρειαστείτε απλά να δημιουργήσετε/αποκτήσετε μόνο το εργαλείο αυτό και όχι ... ολόκληρο το γκαράζ!

Αυτή είναι η βασική φιλοσοφία των λειτουργικών συστημάτων τύπου Unix. Ένα άλλο χαρακτηριστικό του συστήματος είναι ότι οτιδήποτε στο σύστημα είναι ... αρχείο. Είτε πρόκειται για δεδομένα σε μορφή κειμένου, είτε εκτελέσιμο πρόγραμμα σε γλώσσα μηχανής, είτε σκληρός δίσκος, συσκευή, οθόνη, κάρτα ήχου ... Άρα το πρώτο που πρέπει να κατανοήσουμε είναι πως δομείται το σύστημα αρχείων (filesystem).

1.1.1 Filesystem

Καταρχήν σε κάθε αρχείο μας οδηγεί ένα ... μονοπάτι (path). Υπάρχουν δύο τρόποι να γράψουμε ένα path. Το σχετικό (relative) και το απόλυτο (absolute). Δύο παραδείγματα είναι:

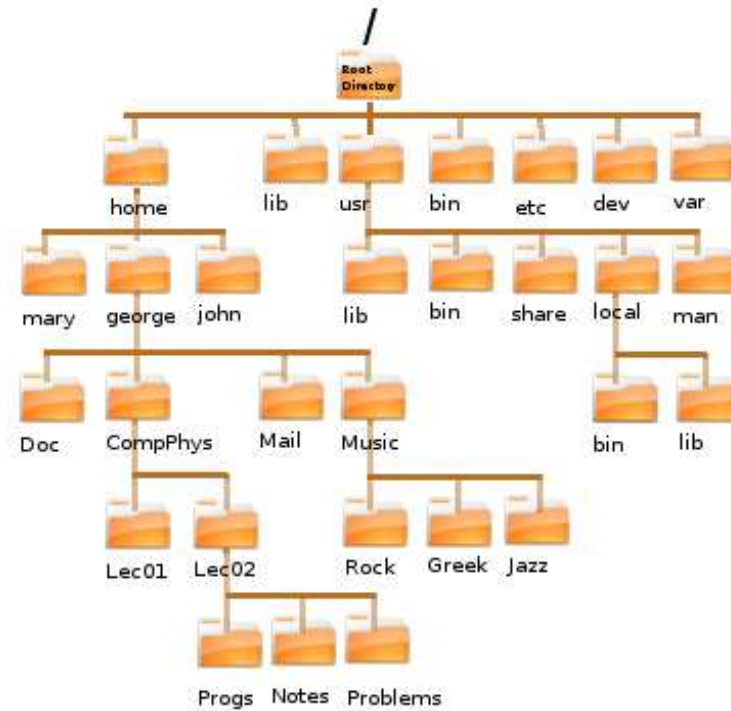
```
bin/RungeKutta/rk.exe
/home/george/bin/RungeKutta/rk.exe
```

Στα παραπάνω και τα δύο μπορεί να αναφέρονται στο ίδιο αρχείο, μπορεί όμως και σε διαφορετικό. Εξαρτάται 'που είμαστε'. Αν 'είμαστε' στον κατάλογο /home/george/ τότε αναφερόμαστε στο ίδιο αρχείο. Αν είμαστε στον κατάλογο /home/john/ ή /home/george/CompPhys/ τότε όχι. Στις τελευταίες περιπτώσεις από το relative path γίνεται αναφορά στα αρχεία /home/john/bin/RungeKutta/rk.exe και /home/george/CompPhys/bin/RungeKutta/rk.exe αντίστοιχα. Πώς τα ξεχωρίζουμε; Το absolute path αρχίζει πάντα από τον χαρακτήρα / ενώ το relative path όχι.

Παραπάνω το 'είμαστε' αναφέρεται σε μια θέση στο σύστημα των αρχείων που ονομάζεται 'τρέχων κατάλογος' (current or working directory). Σε κάθε διεργασία στο λειτουργικό σύστημα αντιστοιχεί ένας τρέχων κατάλογος. Το σύστημα αρχείων στο Unix είναι ενιαίο. Ακόμα και αν πρόκειται για διαφορετικούς σκληρούς δίσκους, συστήματα αρχείων που συνδέονται στον υπολογιστή μας μέσω δικτύου, το CD/DVD, ο εξωτερικός USB δίσκος, τα αρχεία για αλληλεπίδραση με hardware (οθόνη, ποντίκι, modem,) όλα αναρτώνται στο ίδιο λογικά σύστημα αρχείων. Ο χρήστης/διαχειριστής έχει απόλυτη ελευθερία να τα βάλει εκεί που αυτή/ός θέλει⁵.

⁵Αυτό δίνει μια δυνατή αίσθηση ελευθερίας, από την άλλη είναι η αρχή του ... χάους. Ιστορικά αυτό δημιούργησε ένα πύργο της Βαβέλ για συστήματα Unix που αποτέλεσε και έναν από τους κύριους λόγους που άλλα, σαφώς κατώτερης ποιότητας, λειτουργικά συστήματα επικράτησαν στην αγορά των PC

Το filesystem χτίζεται πάνω στη ρίζα του (root) σαν ένα ανάποδο δέντρο. Το σύμβολο του root είναι η / Δηλ. έχουμε, ξεκινώντας από το root φτιά-



Σχήμα 1.1: Το filesystem στο Unix. Στην κορυφή έχουμε τη ρίζα (root directory) του συστήματος αρχείων, τον κατάλογο /. Κάθε κατάλογος περιέχει αρχεία, μεταξύ των οποίων και υποκαταλόγους. Έχει ένα και μοναδικό γονεϊκό κατάλογο (parent directory) που συμβολίζεται με .. (δύο τελείες). Ο / έχει για γονεϊκό κατάλογο τον εαυτό του.

χνουμε καταλόγους και μέσα στους καταλόγους υποκαταλόγους κ.ο.κ. Κάθε κατάλογος χρειάζεται να γνωρίζει το ... γονιό του (parent directory) και τα αρχεία που περιέχει (και από αυτά μερικά μπορεί να είναι υποκατάλογοι - κι αυτοί αρχεία είναι).

Όπως είπαμε στο Unix έχουμε ελευθερία να βάλουμε τα αρχεία μας όπου θέλουμε, αλλά ευτυχώς υπάρχουν μερικές συμβάσεις που μπορούμε να περιμένουμε ότι στα περισσότερα συστήματα θα ακολουθούνται. Έτσι στον κατάλογο /home συνήθως βρίσκουμε τις προσωπικές περιοχές (home directories) των χρηστών, στον /etc αρχεία παραμετροποίησης λειτουργίας συστήματος (sy-

stem configuration files), σε καταλόγους με όνομα `bin` τα εκτελέσιμα αρχεία προγραμμάτων `binaries`, σε καταλόγους με όνομα `lib` βιβλιοθήκες προγραμμάτων.

Μερικές σημαντικές συμβάσεις για θέσεις στο filesystem είναι η `.` (τελεία = ο τρέχων κατάλογος - current directory), η `..` (δύο τελείες = ο 'γονεϊκός' κατάλογος - parent directory) και η `~` (περισπωμένη = προσωπική περιοχή χρήστη - home directory). Έστω για παράδειγμα ότι είμαστε ο χρήστης `john` στον τρέχοντα κατάλογο `/home/john/Mail/Kostas`. Τότε τα παρακάτω paths αναφέρονται στο ίδιο αρχείο `/home/john/Books/Comp.doc`:

```
../..../Books/Comp.doc
~/Books/Comp.doc
~john/Books/Comp.doc
../..../Books/Comp.doc
```

Εισάγουμε τώρα παρακάτω τις βασικές εντολές για να πλοηγούμαστε στο filesystem⁶. Η εντολή `cd` (change directory) μας αλλάζει τοποθεσία στο filesystem ενώ η `pwd` μας αναφέρει πού βρισκόμαστε:

```
> cd /usr/bin
> pwd
/usr/bin
> cd /usr/local/lib
> pwd
/usr/local/lib
> cd
> pwd
/home/konstant
> cd -
> pwd
/usr/local/lib
> cd ../../
> pwd
/usr
```

⁶Οι εντολές που αρχίζουν με `>` είναι εντολές που δίνονται από τη γραμμή εντολών και φυσικά ο αρχικός χαρακτήρας `>` δεν είναι μέρος της εντολής. Οι γραμμές χωρίς `>` είναι το κείμενο που τυπώνει η εντολή στο `stdout` (τερματικό)

Το όρισμα της εντολής `cd` είναι ένα absolute ή relative path στο οποίο (αν είναι σωστό και έχουμε την άδεια πρόσβασης) ‘μεταβαίνουμε’⁷. Εξαιρέσεις είναι να μη δοθεί όρισμα (πάμε στο home directory) ή ο χαρακτήρας `-` (πάμε εκεί που βρισκόμασταν πριν). Η εντολή `mkdir` δημιουργεί καινούργιους καταλόγους ενώ η `rmdir` τους σβήνει αν είναι άδειοι. Δοκιμάστε:

```
> mkdir new
> mkdir new/01
> mkdir new/01/02/03
mkdir: cannot create directory 'new/01/02/03': No such file or directory
> mkdir -p new/01/02/03
> rmdir new
rmdir: 'new': Directory not empty
> rmdir new/01/02/03
> rmdir new/01/02
> rmdir new/01
> rmdir new
```

Προσέξτε πως η `mkdir` δεν μπορεί να δημιουργήσει καταλόγους δύο επίπεδα πιο κάτω ενώ η `mkdir -p` μπορεί. Ο ‘διακόπτης’ `-p` αλλάζει τον τρόπο λειτουργίας της εντολής αυτής.

Για να δούμε τα περιεχόμενα ενός καταλόγου χρησιμοποιούμε την εντολή `ls`:

```
> ls
BE.eps  Byz.eps  Programs      srBE_xyz.eps  srB_xyz.eps
B.eps   Bzy.eps  srBd_xyz.eps  srB_xy.eps
> ls Programs
Backup          rk3_Byz.f  rk3.f
plot-commands  rk3_Bz.f   rk3_g.f
```

Με την πρώτη εντολή βλέπουμε τα περιεχόμενα του καταλόγου που βρισκόμαστε, ενώ στη δεύτερη (προφανώς το αρχείο `Programs` είναι υποκατάλογος) τα περιεχόμενα του καταλόγου που βάζουμε στην εντολή σαν όρισμα. Ένας άλλος τρόπος να δώσουμε την εντολή είναι

```
> ls -l
total 252
```

⁷ Δηλαδή αλλάζει τον current directory της διεργασίας.

```

-rw-r--r--  1 konstant users 24284 May  1 12:08 BE.eps
-rw-r--r--  1 konstant users 22024 May  1 11:53 B.eps
-rw-r--r--  1 konstant users 29935 May  1 13:02 Byz.eps
-rw-r--r--  1 konstant users 48708 May  1 12:41 Bzy.eps
drwxr-xr-x  4 konstant users  4096 May  1 23:38 Programs
-rw-r--r--  1 konstant users 41224 May  1 22:56 srBd_xyz.eps
-rw-r--r--  1 konstant users 23187 May  1 21:13 srBE_xyz.eps
-rw-r--r--  1 konstant users 24610 May  1 20:29 srB_xy.eps
-rw-r--r--  1 konstant users 23763 May  1 20:29 srB_xyz.eps

```

Ο ‘διακόπτης’ (switch) `-l` κάνει την εντολή `ls` να συμπεριφερθεί διαφορετικά. Μας δίνει τα περιεχόμενα του current directory μαζί με χρήσιμες πληροφορίες για τα αρχεία που περιέχει. Η πρώτη στήλη έχει κωδικοποιημένες τις άδειες χρήσης για κάθε αρχείο (βλ. παρακάτω). Η δεύτερη των αριθμό των συνδέσμων (links) των αρχείων. Η τρίτη το όνομα του χρήστη στον οποίο ανήκουν τα αρχεία. Η τέταρτη την ομάδα (group) του αρχείου⁸. Η πέμπτη το μέγεθος του αρχείου σε bytes = 8 bits. Οι επόμενες 3 το χρόνο τελευταίας μετατροπής του αρχείου. Και τέλος το όνομα του αρχείου.

Οι άδειες πρόσβασης `r`, `w`, `x` είναι άδειες πρόσβασης για **read**, **write**, **execute**. Όποιος έχει άδεια `r` έχει άδεια να διαβάσει και αντιγράψει ένα αρχείο. Όποιος έχει άδεια `w` μπορεί να μεταβάλλει τα περιεχόμενα ενός αρχείου. Όποιος έχει άδεια `x` μπορεί να εκτελέσει ένα αρχείο ως πρόγραμμα⁹. Ειδικά για τους καταλόγους, για να μπορεί ο χρήστης/ομάδα/κόσμος να ‘μπει’ σε έναν κατάλογο με την εντολή `cd` πρέπει να έχει άδεια `x`. Για να μπορέσει να σβήσει ένα αρχείο πρέπει να έχει άδεια `w` στον κατάλογο που ανήκει.

Οι άδειες χωρίζονται σε τρεις ομάδες: Ο χρήστης (user- θέσεις 2-4), η ομάδα (group- θέσεις 5-7) και ο υπόλοιπος κόσμος (others-θέσεις 8-10). Έτσι για παράδειγμα

```

-rw-r--r--
-rwxr-----
drwx--x--x

```

Στην πρώτη περίπτωση ο χρήστης έχει άδεια **read**, **write** αλλά όχι **execute** και η ομάδα/κόσμος έχει μόνο άδεια **read**. Στη δεύτερη ο χρήστης έχει άδεια

⁸Ένας χρήστης μπορεί να ανήκει σε πολλές ομάδες για να διευκολύνεται η συνεργασία με διαφορετικές ομάδες χρηστών. Φυσικά κάθε ομάδα μπορεί να έχει πολλούς χρήστες για μέλη.

⁹Φυσικά το αν θα μπορέσει να εκτελεστεί σωστά είναι ευθύνη του χρήστη

`read`, `write`, `execute`, η ομάδα άδεια `read` και ο κόσμος τίποτα. Στην τρίτη ο χρήστης έχει άδεια `read`, `write`, `execute`, η ομάδα/κόσμος άδεια `execute`. Ειδικά στην τρίτη βρίσκουμε το χαρακτήρα `d` στην πρώτη θέση που δηλώνει ότι το αρχείο είναι κατάλογος (directory). Η πρώτη αυτή θέση όταν είναι ‘κατειλημμένη’ δηλώνει αρχείο ειδικού τύπου.

Οι άδειες πρόσβασης αλλάζουν με την εντολή `chmod`:

```
> chmod u+x file
> chmod og-w file1 file2
> chmod a+r file
```

Με την πρώτη εντολή ο χρήστης ($u \equiv \text{user}$) παίρνει (+) άδεια `x` στο αρχείο `file`. Με τη δεύτερη ο κόσμος ($o \equiv \text{others}$) και η ομάδα ($g \equiv \text{group}$) χάνουν (-) άδεια `w` ενώ στην τρίτη όλοι ($a \equiv \text{all}$) αποκτούν άδεια πρόσβασης `r`.

Τελειώνουμε την παράγραφο αυτή αναφέροντας μερικές ακόμα βασικές εντολές που αναφέρονται στη διαχείριση των αρχείων. Η εντολή `cp` (copy) φτιάχνει αντίγραφα αρχείων:

```
> cp file1.f file2.f
> cp file1.f file2.f file3.f Programs
```

Η πρώτη εντολή αντιγράφει τα δεδομένα του αρχείου `file1.f` σε ένα καινούργιο αρχείο `file2.f` αν αυτό δεν υπάρχει ήδη, ή αντικαθιστά το αρχείο `file2.f` από ένα καινούργιο με τα περιεχόμενα του `file1.f`. Η δεύτερη αντιγράφει τα αρχεία `file1.f` `file2.f` `file3.f` στον κατάλογο `Programs` (αν δεν είναι κατάλογος εισπράττουμε ... παράπονα).

Η εντολή `mv` (move) ‘μετακινεί’ ή μετονομάζει αρχεία:

```
> mv file1.f file2.f
> mv file1.f file2.f file3.f Programs
```

Η πρώτη εντολή έχει ως αποτέλεσμα να μετονομάσει το αρχείο `file1.f` σε `file2.f`. Η δεύτερη εντολή μετακινεί τα αρχεία `file1.f` `file2.f` `file3.f` στον κατάλογο `Programs`

Τέλος η εντολή `rm` (remove) διαγράφει αρχεία¹⁰. Η εντολή αυτή δε ‘χαρίζει κάστανά’. Όταν το αρχείο διαγράφεται, το λειτουργικό σύστημα δεν μπορεί να το επαναφέρει. Προσοχή λοιπόν

¹⁰ Στην πραγματικότητα αφαιρεί ‘links’ (συνδέσεις στο filesystem μιάς διαμέρισης-partition) αρχείων. Ένα αρχείο μπορεί να έχει ένα ή περισσότερα links στην ίδια διαμέριση ενός filesystem. Ένα αρχείο θεωρείται διαγραμμένο όταν αφαιρεθούν όλα τα links του.

```
> ls
file1.f file2.f file3.f file4.csh
> rm file1.f file2.f file3.f
> ls
file4.csh
```

τα αρχεία `file1.f` `file2.f` `file3.f` δεν υπάρχουν πια για το λειτουργικό σύστημα¹¹. Για να είμαστε πιο προσεκτικοί μπορούμε να χρησιμοποιήσουμε το διακόπτη `-i`. Τότε η εντολή ζητάει επιβεβαίωση πριν την καταστροφή:

```
> rm -i *
rm: remove regular file 'file1.f'? y
rm: remove regular file 'file2.f'? y
rm: remove regular file 'file3.f'? y
rm: remove regular file 'file4.csh'? n
> ls
file4.csh
```

Στην τελευταία γραμμή απαντήσαμε αρνητικά και έτσι το αρχείο `file4.csh` δεν διαγράφηκε.

Η εντολή `rm` δε διαγράφει καταλόγους. Χρησιμοποιήστε την εντολή `rmdir` για τη διαγραφή άδειων καταλόγων. Για να διαγράψετε καταλόγους με περιεχόμενα χρησιμοποιήστε την εντολή¹² `rm -r`. Λ.χ. έστω ότι έχουμε στους καταλόγους `dir1` και `dir1/dir2` τα αρχεία:

```
./dir1
./dir1/file2.f
./dir1/file1.f
./dir1/dir2
./dir1/dir2/file3.f
```

Οι εντολές

```
> rm dir1
rm: cannot remove 'dir1': Is a directory
> rm dir1/dir2
rm: cannot remove 'dir1/dir2': Is a directory
> rmdir dir1
```

¹¹ Αυτό δε σημαίνει ότι τα δεδομένα τους δεν υπάρχουν πια στο δίσκο...

¹² Ένα `rm -r *` και τα δεδομένα σας αποτελούν ιστορία...

```
rmdir: dir1: Directory not empty
> rmdir dir1/dir2
rmdir: dir1/dir2: Directory not empty
> rm -r dir1
```

Με την τελευταία εντολή όλα τα παραπάνω αρχεία διαγράφονται.

1.1.2 Εντολές

Οι εντολές στο Unix είναι, όπως είπαμε, αρχεία με άδεια πρόσβασης **x** (**execute**). Όταν στη γραμμή εντολών γράψουμε μία πρόταση λ.χ.

```
> ls -l test.f test.dat
```

ο φλοιός (το πρόγραμμα με το οποίο ο χρήστης αλληλεπιδρά με το λειτ. σύστημα) την ερμηνεύει ως εξής: Η πρόταση χωρίζεται σε λέξεις και πρώτη λέξη (**ls**) ερμηνεύεται ως εντολή. Οι υπόλοιπες περνάνε στην εντολή ως τα ορίσματά της. Κατά σύμβαση, λέξεις που αρχίζουν από το χαρακτήρα **-** (λ.χ. **-l**, **--help**, **--version**, **-03**) έχουν συνήθως ειδική ερμηνεία και ονομάζονται ‘διακόπτες’ (**options**, **switches**) και κάνουνε το πρόγραμμα να εκτελείται με διαφορετικό τρόπο ανάλογα με τις τιμές τους. Είδαμε ήδη τη διαφορά με το πρόγραμμα **ls** που ανάλογα με το αν το καλούμε ως **‘ls’** ή **‘ls -l’** τα αποτελέσματα τυπώνονται με διαφορετικό τρόπο.

Για να εκτελεστεί η εντολή **ls** ο φλοιός αναζητεί ένα αρχείο με το όνομα **ls** που να έχει άδεια πρόσβασης **x**. Για να καταλάβουμε πώς γίνεται η αναζήτηση αυτή πρέπει να εξηγήσουμε τι είναι οι μεταβλητές φλοιού και οι μεταβλητές περιβάλλοντος. Αυτές έχουν ένα όνομα που δίνεται από μια ακολουθία χαρακτήρων και οι τιμές τους λαμβάνονται προτάσσοντας το χαρακτήρα **\$** στο όνομά τους. Έτσι η μεταβλητή με το όνομα **PATH** έχει τιμή **\$PATH**. Οι τιμές των μεταβλητών περιβάλλοντος τίθενται με την εντολή¹³ **setenv** για τις μεταβλητές περιβάλλοντος και με την εντολή **set** για τις μεταβλητές φλοιού:

```
> setenv MYVAR test-env
> set myvar = test-shell
> echo $MYVAR $myvar
test-env test-shell
```

¹³Η εντολή **setenv** είναι ειδική για το φλοιό **tcsh**. Στο φλοιό **bash** αρκεί να καθορίσετε την τιμή με ένα **=**: **MYVAR=test-env**.

Δύο μεταβλητές των οποίων αναλαμβάνει ο φλοιός να τις ορίσει σωστά στο περιβάλλον του χρήστη είναι οι `PATH` και `path`:

```
>echo $path
/usr/local/bin /usr/bin /bin /usr/X11/bin
>echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/X11/bin
```

Βλέπουμε ότι η τιμή τους (που ο χρήστης μπορεί να αλλάξει!) αποτελείται από συνιστώσες που είναι διαδρομές στο σύστημα αρχείων. Στην πρώτη περίπτωση οι συνιστώσες χωρίζονται από κενό ενώ στη δεύτερη από : (άνω-κάτω τελεία).

Έτσι, επιστρέφοντας στην ερώτηση πώς βρίσκει ο φλοιός την εντολή `ls`, θα είναι ήδη φανερό πως ψάχνει κάθε συνιστώσα της τιμής της μεταβλητής `path` μέχρι να τη βρει. Αν είστε περίεργοι, δώστε τις εντολές

```
> which ls
/bin/ls
> ls -l /bin/ls
-rwxr-xr-x 1 root root 93560 Sep 28 2006 /bin/ls
```

από όπου είναι προφανές ότι το ζητούμενο αρχείο είναι το `/bin/ls`. Αν η διαδικασία αποτύχει, ο φλοιός δίνει μήνυμα σφάλματος. Αν πετύχει, το πρόγραμμα φορτώνεται από το λειτουργικό σύστημα στη μνήμη για εκτέλεση. Τα ορίσματα περνάνε στην εντολή ώστε αυτή να τα ερμηνεύσει όπως έχει προγραμματιστεί. Στην εντολή

```
> ls -l test.f test.dat
```

το όρισμα `-l` είναι διακόπτης που ερμηνεύεται από την εντολή να δώσει long listing των αρχείων. Τα ορίσματα `test.f` και `test.dat` ερμηνεύονται από την εντολή ως τα αρχεία που θα αναζητήσει για να μας δώσει πληροφορίες.

Μία σημαντική πληροφορία στην ερμηνεία των ορισμάτων είναι η χρήση ‘μπαλαντέρ’ (wildcard):

```
> ls -l *.f *.dat
```

θα κάνει το φλοιό να αναπτύξει τα αστεράκια πριν να περάσει τα ορίσματα στο πρόγραμμα σε οποιαδήποτε ακολουθία χαρακτήρων δίνει ένα υπάρχον αρχείο. Έτσι αν ο κατάλογος που βρισκόμαστε περιέχει τα αρχεία `test.f`, `test1.f`, `myprog.f`, `test.dat`, `hello.dat` η εντολή που θα ‘δει’ το λειτουργικό είναι


```
> ls -l myprog.f test1.f test.f hello.dat test.dat
```

Αυτό συμβαίνει για οποιαδήποτε άλλη εντολή.

Σε κάθε εντολή συναρτάται η καθιερωμένη είσοδος `stdin` (standard input) η καθιερωμένη έξοδος `stdout` (standard output) και η καθιερωμένη έξοδος σφαλμάτων `stderr` (standard error). Αυτές είναι συμβάσεις για αρχεία στα οποία το πρόγραμμα μπορεί να διαβάσει ή να τυπώνει δεδομένα. Όταν ο χρήστης δουλεύει σε ένα τερματικό, όλες οι παραπάνω θεωρούνται αρχικά να είναι το τερματικό¹⁴. Δηλ. μια εντολή που διαβάσει δεδομένα από το `stdin`, αυτά ο χρήστης θα τα εισάγει μέσω του τερματικού τυπώνοντάς τα με το πληκτρολόγιο. Αν μια εντολή τυπώνει στο `stdout` ή στο `stderr` αυτά τυπώνονται στο τερματικό.

Η δυνατότητα που δίνει μεγάλη ευελιξία στο χρήστη να χειριστεί τις εντολές είναι η δυνατότητα επαναορισμού των παραπάνω αρχείων. Ο χρήστης μπορεί να τα ορίσει να είναι οποιοδήποτε αρχείο. Ο επαναορισμός του `stdout` γίνεται με το σύμβολο `>`.

```
> ls
file1.f  file2.f  file3.f  file4.csh
> ls > results
> ls
file1.f  file2.f  file3.f  file4.csh  results
```

Στην πρώτη εντολή βλέπουμε τα περιεχόμενα του καταλόγου. Στη δεύτερη επαναορίζουμε το `stdout` να είναι το αρχείο `results`. Μετά την εκτέλεση της εντολής παρατηρούμε τη δημιουργία του αρχείου `results` το οποίο περιέχει σε δεδομένα τα ονόματα των αρχείων `file1.f file2.f file3.f file4.csh`. Αν το αρχείο `results` δεν υπάρχει δημιουργείται, αν υπάρχει τα περιεχόμενα του καταστρέφονται και αντικαθίστανται από το `stdout` της εντολής. Για να επισυνάψουμε (append) τα δεδομένα του `stdout` στο τέλος ενός ήδη υπάρχοντος αρχείου, χρησιμοποιούμε το σύμβολο `>>`. Έτσι αν μετά από τις παραπάνω εντολές εκτελέσουμε

```
> ls >> results
```

τότε τα περιεχόμενα του αρχείου `results` θα είναι

```
file1.f  file2.f  file3.f  file4.csh
file1.f  file2.f  file3.f  file4.csh  results
```

¹⁴Σας θυμίζουμε ότι για το Unix τα πάντα είναι αρχεία.

Ο επαναορισμός του `stdin` γίνεται με το σύμβολο `<` ενώ του `stderr` με το σύμβολο `>&`¹⁵. Σχετικά παραδείγματα θα δούμε στη παράγραφο 1.2.

Είναι δυνατόν το `stdin/stdout` μιας εντολής να οριστεί να είναι το `stdout/stdin` μιας άλλης εντολής. Με τον τρόπο αυτό μπορούν να συνδυαστούν οι λειτουργίες διαφορετικών εντολών έτσι ώστε να παράγουν αποτελέσματα για τα οποία θα χρειαζόταν να γράψουμε ένα αρκετά πολύπλοκο πρόγραμμα για να τα πάρουμε. Η διαδικασία αυτή λέγεται ‘διασωλήνωση’ (pipring) και χρησιμοποιείται κυρίως για τη δημιουργία ισχυρών φίλτρων. Για το σκοπό αυτό χρησιμοποιείται το σύμβολο `|`

```
> cmd1 | cmd2 | cmd3 | ... | cmdN
```

Με την παραπάνω πρόταση το `stdout` της εντολής `cmd1` γίνεται `stdin` της εντολής `cmd2`, το `stdout` της εντολής `cmd2` γίνεται `stdin` της εντολής `cmd3` κοκ. Σχετικά παραδείγματα θα δούμε στη παράγραφο 1.2.

1.1.3 Αναζητώντας Βοήθεια

Το Unix απέκτησε τη φήμη λειτουργικού συστήματος μη φιλικού προς το χρήστη. Τίποτα δεν απέχει περισσότερο από την πραγματικότητα. Παρόλο που έχει μια αρχική δυσκολία, η οποία λύνεται αν ο χρήστης μεθοδικά διαβάσει και εξασκηθεί στις βασικές εντολές του συστήματος, στη συνέχεια όλες οι πληροφορίες για να κάνει ο χρήστης οτιδήποτε είναι διαθέσιμη online¹⁶.

Το κλειδί για άνετη πλεύση σε αυτό το ταξίδι είναι να μάθει ο χρήστης να χρησιμοποιεί το σύστημα βοήθειας που παρέχεται εντός και εκτός συστήματος. Οι περισσότερες εντολές παρέχουν βασικές πληροφορίες από μόνες τους. Από τη γραμμή εντολών, για τυχαία εντολή `cmd` δοκιμάστε:

```
> cmd --help
> cmd -h
> cmd -help
> cmd -\?
```

Για παράδειγμα δώστε την εντολή `ls --help`. Αν είναι εφαρμογή παραθυρική αρχίστε από το σχεδόν πάντοτε διαθέσιμο menu ‘Help’. Μη φοβηθείτε να διαβάσετε...

¹⁵Το `>&` ισχύει μόνο για το φλοιό `tesh`. Για άλλους φλοιούς (`bash`, `sh`, ...) διαβάστε τη σχετική βοήθεια.

¹⁶Σε αντίθεση με άλλα δημοφιλή λειτουργικά συστήματα τα οποία είναι ‘μαύρα κουτιά’.

Ας υποθέσουμε πως έχουμε ακούσει κάτι για μια εντολή που λέγεται `printf` ή κάτι τέτοιο τέλος πάντων. Το πρώτο σύστημα βοήθειας είναι τα `man pages`. Αυτό είναι ένα σύστημα από `help files` που τις αναζητούμε με την εντολή `man`:

```
> man printf
```

Η εντολή `info` δίνει περισσότερες πληροφορίες σε μορφή ‘βιβλίου’ με βασικές δυνατότητες ξεφυλλίσματος (`browsing`).

```
> info printf
```

οι εντολές

```
> man -k printf
```

```
> whatis printf
```

μας πληροφορούν ότι υπάρχουν και άλλες, πιθανώς σχετιζόμενες εντολές `fprintf`, `fwprintf`, `wprintf`, `sprintf`... Ειδικά το αποτέλεσμα της δεύτερης το παραθέτουμε γιατί είναι διδακτικό:

```
> whatis printf
```

```
printf          (1)  - format and print data
printf          (1p) - write formatted output
printf          (3)  - formatted output conversion
printf          (3p) - print formatted output
printf [builtins] (1) - bash built-in commands, see bash(1)
```

Η δεύτερη στήλη είναι το ‘τμήμα’ (`section`) των `man pages` στο οποίο αναφέρεται η εντολή. Η πρόσβαση στα τμήματα γίνεται δίνοντας το σαν όρισμα στην εντολή:

```
> man 1 printf
```

```
> man 1p printf
```

```
> man 3 printf
```

```
> man 3p printf
```

```
> man bash
```

δίνει πρόσβαση στις αντίστοιχες πληροφορίες. Στο τμήμα ένα βρίσκουμε το `printf` ως ‘κοινή εντολή’, στο τμήμα 3 ως συνάρτηση της γλώσσας C. Άλλα τμήματα είναι το 2 (εντολές διαχείρισης συστήματος), 4, 5, 8 κλπ. Περιηγηθείτε στον κατάλογο `/usr/share/man/` για να δείτε με τα μάτια σας περισσότερα.

Δίνοντας την εντολή

```
> printf --help
```

παίρνουμε πάλι αρκετή πληροφορία. Η εντολή

```
> locate printf
```

μας δείχνει πολλά σχετικά αρχεία στο σύστημα. Οι εντολές

```
> which printf
```

```
> where printf
```

μας δίνει πληροφορία για το πού βρίσκονται τα αρχεία-προγράμματα που εκτελούνται όταν δίνεται η εντολή `printf`.

Μια άλλη σημαντική ευκολία που μας προσφέρει ο φλοιός είναι η ‘συμπλήρωση εντολών’. Μπορούμε να γράψουμε μέρος του ονόματος μιας εντολής και να πατήσουμε το συνδυασμό πλήκτρων `[Ctrl-d]`¹⁷ (δηλ. ταυτόχρονα το πλήκτρο `Ctrl` και το πλήκτρο `d`). Τότε ο φλοιός θα μας συμπληρώσει όλες τις εντολές των οποίων το όνομα αρχίζει με τα γράμματα που έχουμε ήδη γράψει¹⁸:

```
> pri[Ctrl-d]
```

```
printfm    printf  printenv  printnodetest
```

Δοκιμάστε λ.χ. την εντολή `x[Ctrl-d]` και θα μάθετε (σχεδόν) τα πάντα για τις εντολές διαθέσιμες στο παραθυρικό σύστημα `X`: `xterm`, `xeyes`, `xclock`, `xcalc`,

Τέλος, μεγάλη πηγή πληροφοριών είναι το διαδίκτυο. Google your blues... και θα εκπλαγείτε πόσοι άλλοι έχουν ασχοληθεί με το πρόβλημά σας.

1.2 Εργαλεία Επεξεργασίας Κειμένου – Φίλτρα

Για την ανάλυση των δεδομένων που θα παράγουμε χρειαζόμαστε εργαλεία τα οποία να επεξεργάζονται ευέλικτα αρχεία κειμένου. Μερικά εργαλεία που μπορούν να φτιάξουν περίπλοκα και ισχυρά φίλτρα είναι τα προγράμματα `cat`, `less`, `head`, `tail`, `grep`, `sort` και `awk`. Ας αναφέρουμε και τα προγράμματα `perl` και `sed` για τον αναγνώστη που ενδιαφέρεται να πλουτίσει το οπλοστάσιό του παρόλο που δε θα τα περιγράψουμε εδώ λόγω χώρου.

¹⁷Στο φλοιό `bash` πατήστε ένα ή δύο `[Tab]`.

¹⁸Με τον ίδιο τρόπο γίνεται και συμπλήρωση ονομάτων αρχείων. Γράψτε μερικώς το όνομα ενός αρχείου στο όρισμα μιας εντολής και πατήστε `[Tab]` ή `[Ctrl-d]`.

Ας υποθέσουμε ότι έχουμε το αρχείο με δεδομένα με όνομα `data` με τα περιεχόμενα μιας αποθήκης τροφίμων και το κοστολόγιό τους:

```
bananas 100 pieces 1.45
apples   325 boxes  1.18
pears    34 kilos  2.46
bread    62 kilos  0.60
ham       85 kilos  3.56
```

Η εντολή

```
> cat data
```

απλά τυπώνει τα περιεχόμενα στο `stdout`. Η εντολή παίρνει τα αρχεία από το όρισμα της εντολής ή αν δε δοθούν το `stdin` και τυπώνει τα περιεχόμενά τους στο `stdout`. Αφού αυτά μπορεί να επαναοριστούν η εντολή

```
> cat < data > data1
```

παίρνει τα περιεχόμενα του αρχείου `data` από το `stdin` και τα τυπώνει στο `stdout` που εδώ έχει επαναοριστεί να είναι το αρχείο `data1`. Η εντολή έχει ισοδύναμο αποτέλεσμα με την

```
> cp data data1
```

Η εντολή

```
> cat data data1 > data2
```

τυπώνει πρώτα τα περιεχόμενα του `data` και μετά του `data1` μέσα στο αρχείο `data2`.

Η εντολή

```
> less data
```

τυπώνει στο `stdout` τα περιεχόμενα του `data` σελίδα-σελίδα. Φυσικά εδώ τα αρχεία είναι μικρά, δοκιμάστε με ένα μεγαλύτερο του οποίου τα περιεχόμενα δεν χωρούν σε μια οθόνη του τερματικού. Πατήστε `[space]` για να προχωρήσετε μια σελίδα, `[b]` για να γυρίστε πίσω μια σελίδα και τα πάνω/κάτω βελάκια για να προχωρήσετε μια γραμμή. Με `[g]` πάτε στην αρχή του αρχείου και με `[G]` στο τέλος. Με `[h]` παίρνετε βοήθεια και με `[q]`... αναχωρείτε (`quit`).

Με τις εντολές

```
> head -n 1 data
bananas 100 pieces 1.45
> tail -n 2 data
bread    62 kilos  0.60
ham      85 kilos  3.56
> tail -n 2 data | head -n 1
bread    62 kilos  0.60
```

παίρνουμε την πρώτη γραμμή του αρχείου `data`, τις δύο τελευταίες και την δεύτερη από το τέλος αντίστοιχα. Προσέξτε πώς με `piping` των δύο εντολών τις συνδυάσαμε για να φτιάξουμε το φίλτρο ‘τύπωσε τη δεύτερη γραμμή από το τέλος’.

Η εντολή `sort` τυπώνει τα περιεχόμενα του αρχείου κατά αύξουσα διάταξη των γραμμών, όπου η σύγκριση γίνεται χαρακτήρα-χαρακτήρα (όχι αριθμητικά):

```
> sort data
apples  325 boxes  1.18
bananas 100 pieces 1.45
bread    62 kilos  0.60
ham      85 kilos  3.56
pears    34 kilos  2.46
```

Για αντίστροφη διάταξη δοκιμάστε την εντολή `sort -r data`. Για να διατάξουμε τα περιεχόμενα συγκρίνοντας τους αριθμούς στη δεύτερη στήλη χρησιμοποιούμε το διακόπτη `-k 2` (=δεύτερη στήλη) και `-n` (=αριθμητική – numerical – διάταξη):

```
> sort -k 2 -n data
pears    34 kilos  2.46
bread    62 kilos  0.60
ham      85 kilos  3.56
bananas 100 pieces 1.45
apples  325 boxes  1.18
```

Αν αμελήσω το διακόπτη `-n` οι γραμμές συγκρίνονται με βάση τους χαρακτήρες της λέξης στη δεύτερη στήλη:

```
> sort -k 2 data
bananas 100 pieces 1.45
apples  325 boxes  1.18
```

```
pears      34 kilos  2.46
bread      62 kilos  0.60
ham        85 kilos  3.56
```

Η τελευταία στήλη έχει αριθμούς με υποδιαστολή (όχι ακέραιους). Για να κάνουμε τη διάταξη με βάση την αξία τέτοιων αριθμών βάζουμε το διακόπτη `-g`:

```
> sort -k 4 -g data
bread      62 kilos  0.60
apples     325 boxes  1.18
bananas    100 pieces 1.45
pears      34 kilos  2.46
ham        85 kilos  3.56
```

Η εντολή `grep` αναλύει ένα αρχείο κειμένου γραμμή-γραμμή αναζητώντας μια ακολουθία χαρακτήρων που έχουμε ζητήσει. Κάθε τέτοια γραμμή που βρίσκει την τυπώνει στο `stdout`:

```
> grep kilos data
pears      34 kilos  2.46
bread      62 kilos  0.60
ham        85 kilos  3.56
```

τυπώνει κάθε γραμμή που έχει το 'kilos'. Αν θέλουμε να τυπώνει κάθε γραμμή που δεν περιέχει το kilos προσθέτουμε το διακόπτη `-v`:

```
> grep -v kilos data
bananas    100 pieces 1.45
apples     325 boxes  1.18
```

Η ακολουθία χαρακτήρων που αναζητούμε μπορεί να είναι ένα regular expression. Για να περιγράψουμε τα θηρία αυτά, θέλουμε μισό βιβλίο... Μερικά παραδείγματα:

```
> grep ^b data
bananas    100 pieces 1.45
bread      62 kilos  0.60
> grep '0$' data
bread      62 kilos  0.60
> grep '3[24]' data
apples     325 boxes  1.18
pears      34 kilos  2.46
```

Η πρώτη τυπώνει τις γραμμές που αρχίζουν από `b` (αγνοεί την 2η γραμμή), η δεύτερη αυτές που τελειώνουν σε `0` (αγνοεί την πρώτη γραμμή) ενώ η τρίτη γραμμές που περιέχουν τις ακολουθίες χαρακτήρων `32` ή `34` (αγνοεί την τελευταία γραμμή).

Το πιο δυνατό όμως εργαλείο για ανάλυση είναι το πρόγραμμα `awk`. Στην πιο απλή του χρήση, αναλύει το αρχείο γραμμή-γραμμή και ορίζει μεταβλητές `$1`, `$2`, ... στις οποίες αποθηκεύει την τιμή της πρώτης, δεύτερης, ... λέξης της γραμμής. Στη μεταβλητή `$0` αποθηκεύει όλη τη γραμμή ενώ η μεταβλητή `NF` μετράει τον αριθμό των λέξεων στη γραμμή. Η μεταβλητή `NR` μετράει τις γραμμές που έχει επεξεργαστεί μέχρι στιγμής.

Ένα πρόγραμμα `awk` μπορεί να γραφτεί στη γραμμή εντολών. Είναι εντολές που περιλαμβάνονται ανάμεσα σε αγκύλες `{ ... }` εκτελούνται για κάθε γραμμή του αρχείου. Ειδική περίπτωση αποτελούν οι εντολές που γράφονται μέσα στο κατασκευάσμα `BEGIN{ ... }` και `END{ ... }` που είναι εντολές που εκτελούνται μια φορά πριν την επεξεργασία και μετά την επεξεργασία των γραμμών του αρχείου. Για παράδειγμα η εντολή:

```
> awk '{print $1,"total value= ",$2*$4}' data
bananas total value= 145
apples total value= 383.5
pears total value= 83.64
bread total value= 37.2
ham total value= 302.6
```

τυπώνει το είδος (1η στήλη= `$1`) και την συνολική αξία του ποσότητα (2η στήλη= `$2`) $\times$ αξία μονάδας (4η στήλη= `$4`). Άλλα παραδείγματα είναι

```
> awk '{value += $2*$4}END{print "Total= ",value}' data
Total= 951.94
> awk '{av += $4}END{print "Average Price= ",av/NR}' data
Average Price= 1.85
> awk '{print $2^2 * sin($4) + exp(-$4)}' data
```

Στην πρώτη εντολή υπολογίζουμε τη συνολική αξία των προϊόντων: Σε κάθε γραμμή προσθέτουμε (`+=`) στη μεταβλητή `value` την συνολική αξία του προϊόντος. Στο τέλος (`END{ ... }`) τυπώνουμε το άθροισμα που συσσωρεύσαμε στο τέλος του αρχείου. Η δεύτερη εντολή τυπώνει τη μέση τιμή των τιμών. Με τον ίδιο τρόπο προσθέτουμε στη μεταβλητή `av` την τιμή κάθε προϊόντος (2η στήλη= `$2`) και στο τέλος τυπώνουμε το σύνολο δια τον αριθμό των προϊόντων

(=αρ. γραμμών = NR). Η τελευταία εντολή κάνει μια αυθαίρετη αριθμητική πράξη: Τυπώνει το τετράγωνο της δεύτερης στήλης επί το ημίτονο της τέταρτης και προσθέτει το εκθετικό της -4ης στήλης.

Οι δυνατότητες των παραπάνω εργαλείων δεν εξαντλείται σε ένα μικρό κεφάλαιο. Διαβάστε τις man και info pages και θα μάθετε να τις κάνετε να ψήνουν και ... καφέ!

1.3 Ο Καλύτερος Φίλος του Ανθρώπου

Όχι, δεν είμαστε της φιλοζωικής, για editors¹⁹ μιλάμε... Δεν υπερβάλλουμε όμως, για έναν προγραμματιστή που προγραμματίζει αρκετές ώρες κάθε μέρα, το περιβάλλον και τα εργαλεία επεξεργασίας του κειμένου των εντολών προγραμματισμού καθορίζουν κατά ένα σημαντικό ποσοστό τη συνολική ...ποιότητα της ζωής του/της. Και όπως βλέπετε είμαστε αρκετά προσεκτικοί στη διατύπωση: Δε μιλάμε για προγράμματα επεξεργασίας κειμένου εγγράφων (λ.χ. Open Office²⁰) που δίνουν έμφαση στη φόρμα του κειμένου αλλά για επεξεργαστές απλού κειμένου που αποτελείται από σχέτους (χωρίς φόρμα) χαρακτήρες που 'διαβάζονται'. Παραδείγματα απλών τέτοιων επεξεργαστών στο Linux είναι οι επεξεργαστές gedit, vi, pico, nano κλπ που θα μπορούσε κανείς να χρησιμοποιήσει εναλλακτικά για την επεξεργασία του κώδικα στα προγράμματα που παρουσιάζουμε στο μάθημα. Με αυτούς μπορεί κάποιος εύκολα να επεξεργαστεί απλά προγράμματα έχοντας βασικές λειτουργίες επεξεργασίας κειμένου (editing). Υπάρχουν λειτουργίες σε ένα επεξεργαστή κειμένου που κάνει τον προγραμματισμό ανετότερο και βοηθά στην ... υγιεινή κρατώντας μακριά τα ενοχλητικά ... έντομα! Λ.χ. η αναγνώριση από τον επεξεργαστή των εντολών της γλώσσας προγραμματισμού, των μεταβλητών, των δομικών στοιχείων επιτρέπει την 'όμορφη' παρουσίασή τους με κατάλληλο χρωματισμό ή/και font, επισημαίνει σφάλματα όταν δεν κλείνουν παρενθέσεις ή οι εντολές δεν μπαίνουν στο σωστό σημείο στο αρχείο του προγράμματος κλπ. Ένας 'πολύγλωσσος' και 'πολυμορφικός' επεξεργαστής κειμένου με πολλές δυνατότητες και ευκολίες για τον προγραμματιστή είναι ο GNU Emacs editor²¹. Ο Emacs είναι ανοιχτό λογισμικό, διατίθεται ελεύθερα και μπορεί να εγκατασταθεί σε λειτουργικό σύστημα

¹⁹editor= πρόγραμμα επεξεργασίας κειμένου

²⁰Υπάρχει κι άλλο;

²¹<http://www.gnu.org/software/emacs/> (main site), <http://www.emacswiki.org/> (expert tips), <http://en.wikipedia.org/wiki/Emacs> (general info)

Linux, Mac και MS Windows. Ο χρήστης μπορεί να τον προγραμματίσει²² να εκτελεί απλές αλλά και σύνθετες λειτουργίες της αρεσκείας του/της καθώς και να έχει μια σχεδόν ολοκληρωμένη αλληλεπίδραση με το λειτουργικό σύστημα και πολλές από τις εφαρμογές που βρίσκονται σε αυτό. Ο πιο προχωρημένος χρήστης μπορεί λ.χ. να επεξεργαστεί ένα αρχείο σε γλώσσα Fortran να το μεταγλωττίσει και να το διορθώσει με τη βοήθεια του debugger δίνοντας εντολές μέσα από τον Emacs.

Για τον προγραμματισμό πολύπλοκων προγραμμάτων με πολλές χιλιάδες γραμμές κώδικα και πολύπλοκο συσχετισμό διεργασιών είναι συνηθισμένο να χρησιμοποιούνται εξειδικευμένα περιβάλλοντα προγραμματισμού. Αυτά προσφέρουν στον προγραμματιστή ολοκληρωμένες λύσεις για τον προγραμματισμό σε μια γλώσσα (λ.χ. C++, Java κλπ) ενσωματώνοντας σε ένα απλό interface και τις λειτουργίες μεταγλωττισμού, debugging, βοήθειας κλπ. Το μειονέκτημα σε αυτά είναι η εξειδίκευση που περιορίζει την ελευθερία του προγραμματιστή ως προς την επιλογή γλώσσας, βιβλιοθηκών, λειτουργικού συστήματος και συνήθως έχουν ακριβές άδειες χρήσης. Είναι επίσης δύσκολη η μεταφορά των εργασιών ενός προγραμματιστή από έναν υπολογιστή σε έναν άλλο και φυσικά η επεξεργασία του προγράμματος από διαφορετικά περιβάλλοντα προγραμματισμού. Η πολύπλοκη και εξειδικευμένη παραμετροποίησή τους συνήθως 'δένει' τον προγραμματιστή και το πρόγραμμα με το συγκεκριμένο πακέτο περιβάλλοντος προγραμματισμού.

1.3.1 Καλώντας τον Emacs

Στη γραμμή εντολών πληκτρολογήστε:

```
> emacs &
```

Προσέξτε το χαρακτήρα `&` στο τέλος της εντολής. Χωρίς αυτόν το prompt του φλοιού δεν επιστρέφει και δεν μπορούμε να δώσουμε άλλη εντολή από το φλοιό. Με αυτόν η εντολή (όπως και κάθε εντολή την οποία τελειώνουμε με το `&`) πάει στο 'υπόβαθρο' (background) δηλ. ξεκινάει μία διεργασία ανεξάρτητη από το φλοιό η οποία λειτουργεί ακόμα και αν η διεργασία του φλοιού τερματιστεί.

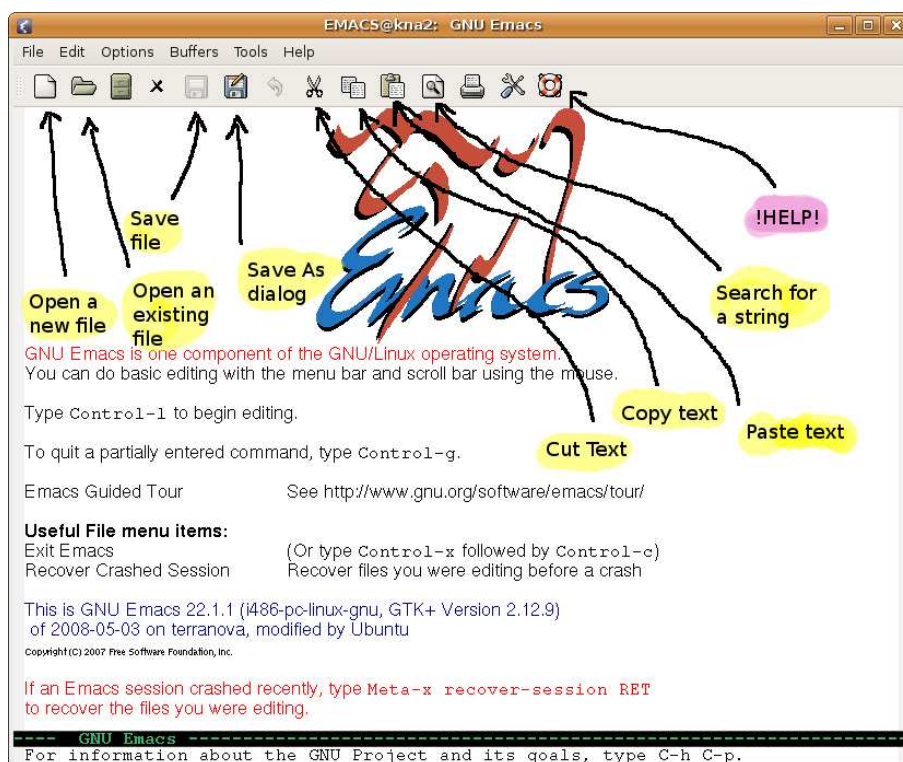
Τα παραπάνω ισχύουν όταν έχουμε παραθυρικό περιβάλλον και τότε ο Emacs ξεκινάει στο δικό του ανεξάρτητο παράθυρο. Μπορούμε όμως να τρέχουμε

²²Ο Emacs είναι γραμμένος σε μια διάλεκτο της γλώσσας προγραμματισμού Lisp που λέγεται Elisp. Για προγραμματισμό απλών λειτουργιών δεν απαιτείται λεπτομερής γνώση της γλώσσας αυτής.

τον Emacs και σε ένα απλό τερματικό, είτε για γρήγορη επεξεργασία κειμένου είτε γιατί δε διαθέτουμε παραθυρικό περιβάλλον²³ αλλά μόνο κονσόλα. Στην τελευταία περίπτωση απλά παραλείπουμε το & στο τέλος της εντολής, ενώ αν έχουμε παραθυρικό περιβάλλον και θέλουμε ο Emacs να τρέξει στην κονσόλα δίνουμε την εντολή

```
> emacs -nw
```

και ο Emacs θα ξεκινήσει μέσα στην κονσόλα.

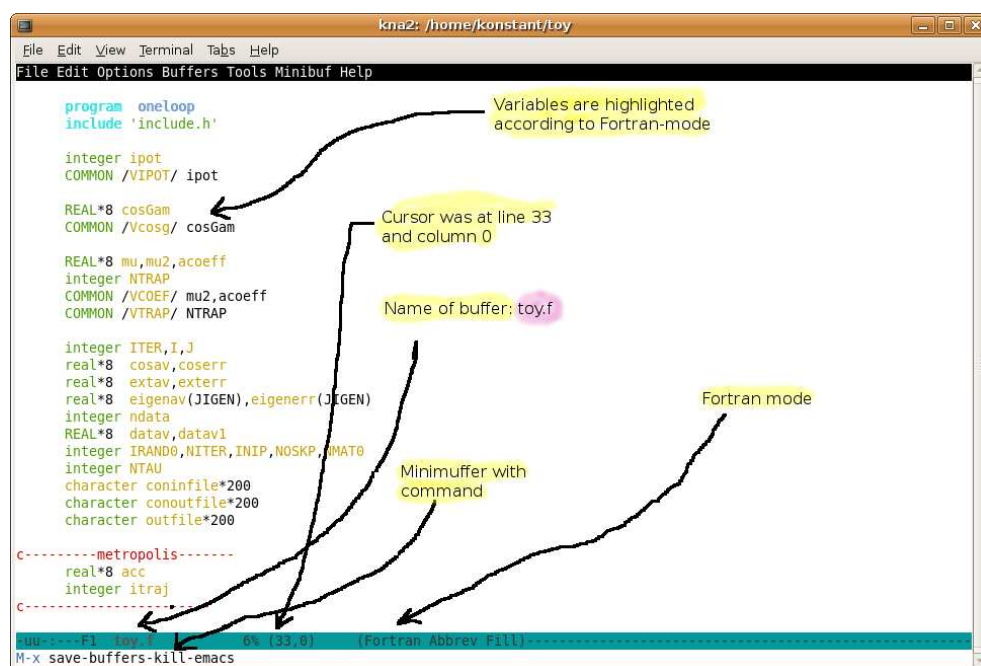


Σχήμα 1.2: Το παράθυρο του Emacs σε ένα παραθυρικό περιβάλλον. Φαίνονται και επεξηγούνται τα βασικά κουμπιά λειτουργίας του.

²³ Αυτό μπορούμε να το καταλάβουμε δίνοντας την εντολή `echo $DISPLAY` και αν πάρουμε το μήνυμα σφάλματος `DISPLAY: Undefined variable.` τότε δεν έχουμε σύνδεση με παραθυρικό περιβάλλον (X server). Αλλιώς θα πάρουμε την τιμή `:0.0`, `localhost:10.0` κλπ.

1.3.2 Αλληλεπιδρώντας με τον Emacs

Με τον Emacs αλληλεπιδρούμε με διάφορους τρόπους. Οι ‘νεοσύλλεκτοι’ θα προτιμήσουν τα κουμπιά και τα μενού που προσφέρει που συνήθως έχουν διασθητική μορφή και ονόματα που συναντά στους περισσότερους επεξεργαστές κειμένου. Αλλά για να χρησιμοποιήσει κανείς τις προχωρημένες δυνατότητες του Emacs είναι καλό να συνηθίσει τις άλλες μορφές αλληλεπίδρασης που είναι συντομεύσεις πλήκτρων και εκτέλεση εντολών με το όνομά τους από τη γραμμή εντολών του Emacs, το minibuffer²⁴.



Σχήμα 1.3: Ο Emacs στην κονσόλα. Στην εικόνα έχουμε μεταβεί στο minibuffer πληκτρολογώντας M-x και έχουμε γράψει την εντολή `save-buffers-exit-emacs` η οποία τερματίζει τον Emacs αφού σώσει τα μεταβληθέντα από την επεξεργασία buffers. Η ίδια εντολή δίνεται ισοδύναμα πληκτρολογώντας C-x C-c. Φαίνεται η mode line στην οποία, ανάμεσα σε άλλα, είναι γραμμένο το όνομα του αρχείου/buffer (toy.f), το ποσοστό του buffer που είναι ορατό στο παράθυρο (6%), η γραμμή και η στήλη που βρίσκεται το σημείο που επεξεργαζόμαστε (33,0) και το editing mode που βρίσκεται ο buffer (Fortran mode (Fortran), Abbreviation mode (Abbrev), Auto Fill mode (Fill)).

Η εντολές που δίνονται με συντομεύσεις από το πληκτρολόγιο είναι συνδυα-

²⁴Είναι και ο πιο απλός τρόπος αλληλεπίδρασης όταν καλούμε τον Emacs στην κονσόλα.

σμός πλήκτρων που πατά κάποιος σε συνδυασμό με τα πλήκτρα **Ctrl** (**Control key**) και **Alt**. Θα ακολουθήσουμε την εξής σύμβαση: Όταν γράφουμε ένα συνδυασμό πλήκτρων αρχίζοντας με **C-** θα εννοούμε ότι τα πλήκτρα που ακολουθούν πατιούνται ταυτόχρονα με το **Control key**, ενώ αν γράφουμε **M-** θα εννοούμε ότι τα πλήκτρα που ακολουθούν πατιούνται ταυτόχρονα με το **Alt key**²⁵. Μερικές εντολές συντομεύονται από μια ακολουθία από δύο ή παραπάνω χαρακτήρες. Λ.χ. πατώντας **C-x C-c** (δηλ. κρατάμε πατημένο το **Ctrl key** και ταυτόχρονα πατάμε το **x** και μετά κρατώντας πατημένο το **Ctrl key** πατάμε το **c**) δίνουμε την εντολή να βγούμε από τον Emacs ενώ πατώντας **C-x 2** (δηλ. κρατάμε πατημένο το **Ctrl key** και ταυτόχρονα πατάμε το **x** και μετά αφήνουμε το **Ctrl key** και πατάμε το **2**) δίνουμε την εντολή να χωριστεί το παράθυρο του buffer που βρισκόμαστε σε δύο ίσα μέρη.

Οι πιο χρήσιμες συντομεύσεις είναι οι **M-x** (πατάμε το **Alt** και κρατώντας το πατημένο πατάμε το **x**) και η **C-g**. Η πρώτη μας οδηγεί στο minibuffer από όπου μπορούμε να δώσουμε μία εντολή με το όνομά της. Για παράδειγμα δώστε την εντολή **save-buffers-exit-emacs** που απλά θα τερματίσει τη συνεδρία του Emacs. Η δεύτερη είναι το ‘κουμπί SOS’ που διακόπτει οτιδήποτε κάνει ο Emacs (λ.χ. αν κάποια εντολή κολλήσει, δώσουμε λάθος εντολή κλπ): Πατώντας **C-g** ο Emacs σταματάει οποιαδήποτε διεργασία κάνει και επιστρέφει στο buffer που εργαζόμαστε. Λ.χ. αν πατήστε κατά λάθος **M-x** και βρεθείτε στο minibuffer χωρίς να το θέλετε πατήστε **C-g** για να ακυρώσετε τη διαδικασία και να επιστρέψετε στο buffer που επεξεργάζασατε.

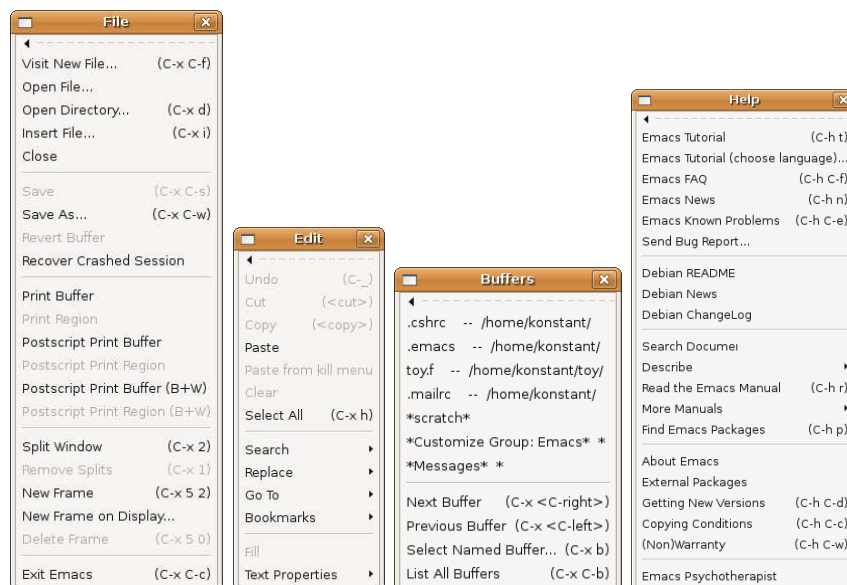
Είναι επίσης χρήσιμο να ορίσουμε συμβάσεις που να υποδηλώνουν τι κάνουμε με το ποντίκι. Με **Mouse-1**, **Mouse-2**, **Mouse-3** υποδηλώνουμε ένα απλό κλικ με το αριστερό, μεσαίο²⁶, και δεξί κουμπί αντίστοιχα. Με **Drag-Mouse-1** υποδηλώνουμε ότι κρατάμε το αριστερό κουμπί διαρκώς κρατημένο και ταυτόχρονα σέρνουμε το ποντίκι.

Ανακεφαλαιώνουμε συνοψίζοντας τους δυνατούς τρόπους να δίνουμε μία εντολή στον Emacs. Θεωρούμε λ.χ. την εντολή που ανοίγει ένα καινούργιο αρχείο σε ένα buffer:

- Από το εικονίδιο που μοιάζει με λευκό χαρτί επάνω αριστερά στη γραμμή

²⁵Στη γλώσσα του Emacs το **M-** είναι το “Meta” key το οποίο βρίσκεται εκτός από το **Alt** και στο **Esc** (**Escape key**). Στην περίπτωση αυτή, σε αντίθεση με το **Alt**, το **Esc** το πατάμε πρώτα και το αφήνουμε και μετά πατάμε τα επόμενα πλήκτρα. Αυτό μπορεί να είναι και η πιο απλή μας επιλογή σε ορισμένα ‘χαζά’ τερματικά. Αν και αυτό δε δουλεύει – μάλλον σπάνιο στην εποχή μας – δοκιμάστε το **C-[**

²⁶Αν το ποντίκι δεν έχει μεσαίο κουμπί πατάμε τη ροδέλα. Αν δεν έχει ροδέλα τότε το αριστερό και το δεξί κουμπί ταυτόχρονα.



Σχήμα 1.4: Τα βασικά μενού που συναντά κανείς στον Emacs σε παραθυρικό περιβάλλον. Βλέπουμε τις βασικές εντολές και σε παρένθεση μας υπενθυμίζεται η αντίστοιχη συντόμευση πληκτρολογίου. Α.χ. η εντολή File → Visit New File μπορεί να δοθεί από το πληκτρολόγιο πληκτρολογώντας C-x C-f. Σημειώστε τις εντολές File → Visit New File (άνοιγμα αρχείου), File→Save (εγγραφή αλλαγών του buffer στο αντίστοιχο αρχείο, File→Exit Emacs (κλείσιμο Emacs), File → Split Window (χωρισμός παραθύρου στα δύο), File→New Frame (άνοιγμα νέου παραθύρου) και φυσικά τις γνωστές εντολές Cut, Copy, Paste, Undo από το Edit menu. Από το μενού Buffers μπορούμε να επιλέξουμε διαφορετικά buffers με τα περιεχόμενα των άλλων αρχείων που επεξεργαζόμαστε. Στους καινούργιους χρήστες συστήνουμε να δουν το Emacs Tutorial και Read Emacs Manual στο Help menu.

των κουμπιών του Σχήματος 1.2

- Από την εντολή μενού File→Visit New File.
- Από τη συντόμευση πληκτρολογίου C-x C-f
- Από εντολή στο minibuffer: M-x find-file

Ο πρώτος τρόπος είναι διαθέσιμος για τις πολύ βασικές εντολές, ο δεύτερος για περισσότερες, ο τρίτος για τις περισσότερες (αλλά όχι όλες) και ο τέταρτος για όλες τις εντολές που είναι διαθέσιμες για διαδραστική χρήση.

1.3.3 Βασική Επεξεργασία Κειμένου

Για να επεξεργαστούμε ένα αρχείο, ο Emacs τοποθετεί τα περιεχόμενά του σε ένα buffer. Το buffer είναι ένα κομμάτι της μνήμης όπου *αντιγράφονται* τα περιεχόμενα ενός αρχείου και όχι το ίδιο το αρχείο. Για να καταγραφούν οι αλλαγές στα περιεχόμενα ενός buffer πρέπει να τις ‘σώσουμε’, δηλ. ο Emacs να γράψει το buffer πίσω στο αρχείο. Μέχρι να γίνει αυτό το αρχικό αρχείο μένει ανέπαφο²⁷. Ο Emacs μπορεί να έχει ανοιχτά πολλά buffers τα οποία όταν συνδέονται με ένα αρχείο έχουν από προεπιλογή το ίδιο όνομα του αρχείου²⁸. Το όνομα ενός buffer φαίνεται στη mode line του Emacs όπως φαίνεται στο Σχήμα 1.3. Ο κύκλος επεξεργασίας ενός αρχείου συνοψίζεται στα εξής σημεία:

- Διάβασμα των περιεχομένων του αρχείου σε ένα buffer.
- Αλλαγή από το χρήστη των περιεχομένων του buffer.
- Εγγραφή των δεδομένων του buffer πίσω στο αρχείο.

Φυσικά αν το αρχείο δεν υπάρχει και δημιουργείται εξ’ αρχής, το πρώτο βήμα παραλείπεται.

Το σημείο στο οποίο βρισκόμαστε νοητά και εισάγουμε κείμενο λέγεται ‘το σημείο’ (point). Αυτό καταδεικνύεται από το δρομέα (cursor) που τυπικά είναι ένα κόκκινο τετραγωνάκι που αναμοσβήνει²⁹. Κάθε buffer έχει μία θέση που ονομάζεται ‘το σημάδι’ (the mark) το οποίο μαζί με το σημείο ορίζει σε κάθε παράθυρο την ‘περιοχή’ (the region). Αυτή είναι μια νοητή περιοχή κειμένου σε κάθε παράθυρο όπου μπορούν να δράσουν οι συναρτήσεις του Emacs (λ.χ. αποκοπή, αντιγραφή, αλλαγή κεφαλαίων σε μικρά γράμματα, έλεγχος ορθογραφίας κλπ). Την περιοχή τη θέτουμε ορίζοντας το σημάδι (mark) επιλέγοντας ένα σημείο και πληκτρολογώντας C-SPC³⁰ (ή στο minibuffer M-x

²⁷ Αν χάσουμε μία συνεδρία του Emacs είναι δυνατόν να ανακτήσουμε μέρος των αλλαγών που κάναμε. Μπορούμε να χρησιμοποιήσουμε την εντολή M-x `recover-file` ή να αναζητήσουμε ένα αρχείο στο δίσκο με όνομα ίδιο με αυτό του αρχείου που επεξεργαζόμαστε ανάμεσα σε δύο #. Λ.χ. το buffer του αρχείου `file.f` σώζεται αυτόματα και περιοδικά στο αρχείο `#file.f#`

²⁸ Αυτό δεν είναι αναγκαστικό. Μπορεί ο χρήστης να αλλάξει το όνομα ενός buffer χωρίς να αλλάξει το αρχείο με το οποίο συνδέεται. Επίσης αν ανοίξουμε αρχεία που έχουν το ίδιο όνομα (λ.χ. `index.html`) που βρίσκονται σε διαφορετικούς καταλόγους, τότε αυτά ονομάζονται από προεπιλογή `index.html`, `index.html<2>`, `index.html<3>`, ...

²⁹ Το σημείο είναι πάντα μεταξύ χαρακτήρων όχι πάνω σε αυτούς. Ο δρομέας είναι πάνω στο χαρακτήρα αμέσως δεξιά από το σημείο. Κάθε παράθυρο έχει ένα σημείο οπότε κάθε buffer μπορεί να έχει περισσότερα από ένα σημεία αν εμφανίζεται σε διαφορετικά παράθυρα.

³⁰ Πατάμε ταυτόχρονα το Control key και το Space bar

`set-mark-command`). Μετακινώντας το δρομέα στο σημείο που θέλουμε ορίζουμε την επιθυμητή περιοχή. Εναλλακτικά με το `Drag-Mouse-1` (κρατάμε αριστερό κουμπί ποντικιού πατημένο και σέρνουμε το ποντίκι) μαρκάριουμε μία περιοχή. Το σημάδι μπορεί να τεθεί και με `Mouse-3` δηλ με απλό κλικ του δεξιού πλήκτρου του ποντικιού (άρα `Mouse-1 Mouse-3` ορίζει μία περιοχή θέτοντας πρώτα το σημείο και μετά το σημάδι).

Ανοίγουμε ένα αρχείο με την εντολή `C-x C-f` και πληκτρολογώντας το όνομά του. Αν το αρχείο υπάρχει βλέπουμε τα περιεχόμενά του στο buffer που δημιουργείται, αλλιώς παίρνουμε ένα άδειο buffer. Τότε:

- Πλοηγούμαστε στο buffer με τα βελάκια του πληκτρολόγιου. Εναλλακτικά με τις εντολές `C-n`, `C-p`, `C-f` και `C-b`.
- Αν έχουμε πολλές ‘σελίδες’ προχωράμε μία-μία σελίδα με `Page Up`, `Page Dn` από το πληκτρολόγιο. Εναλλακτικά με τις εντολές `C-v`, `M-v`
- Εισάγουμε κείμενο απλά πληκτρολογώντας το.
- Σβήνουμε τους χαρακτήρες που βρίσκονται πίσω από το σημείο με το `Backspace` και αυτούς που είναι μπροστά με το `Delete`. Με την εντολή `C-d` σβήνουμε τον μπροστινό χαρακτήρα.
- Σβήνουμε ολόκληρη τη γραμμή που είναι μπροστά από το σημείο με `C-k`
- Ανοίγουμε μια καινούργια γραμμή με `Enter` ή `C-o`
- Πάμε στην αρχή της γραμμής με το πλήκτρο `Home` και στο τέλος της με το πλήκτρο `End`. Εναλλακτικά με `C-a` και `C-e` αντίστοιχα.
- Πάμε στην αρχή του buffer με το πλήκτρο `C-Home` και στο τέλος με `C-End`. Εναλλακτικά με τις εντολές στο minibuffer: `M-x beginning-of-buffer` και `M-x end-of-buffer`.
- Πάμε σε μια γραμμή που θέλουμε με `M-x goto-line`. Στην προτροπή της εντολής δίνουμε τον αριθμό της γραμμής που θέλουμε να πάμε.
- Αναζητούμε κείμενο μπροστά από το σημείο με την εντολή `C-s`. Πληκτρολογούμε το κείμενο μέχρι να το βρούμε. Για να βρούμε το ίδιο κείμενο ξανά (και ξανά) πληκτρολογούμε `C-s` όσες φορές χρειαστεί. Το ίδιο κάνουμε με το `C-r` για να αναζητήσουμε κείμενο πίσω από το σημείο.

- Αν θέλουμε να γράψουμε Ελληνικά, διαβάζουμε την Παράγραφο 1.3.10, σελ. 37

Μόλις τελειώσουμε σώζουμε τις αλλαγές που κάναμε με την εντολή `C-s` ή από το εικονίδιο ‘δισκέτα’ ή από το μενού `File→Save`. Επίσης η εντολή στο minibuffer είναι `M-x save-buffer`.

1.3.4 Κόβοντας και ράβοντας

Για πιο προχωρημένη επεξεργασία ακολουθούμε τις παρακάτω οδηγίες:

- **SOS: Undo!**. Πολλές από τις παρακάτω αλλαγές μπορεί να είναι καταστροφικές. Ο Emacs έχει επαναλαμβανόμενο undo με μεγάλη μνήμη. Πατώντας `C-/` επανειλημμένα προσεγγίζουμε την προηγούμενη κατάσταση που βρισκόμαστε. Εναλλακτικά με `C-x` u και από το μενού `Edit→Undo`. Θυμίζουμε ότι το `C-g` σταματάει οποιαδήποτε λειτουργία του Emacs και έτσι μπορεί να μας γλυτώσει από μέρος μίας καταστροφής αν διακόψουμε το έγκλημα κατά τη διάρκεια που εκτελείται.
- **Αποκοπή κειμένου με το ποντίκι**: Κάνουμε κλικ **Mouse-1** στην αρχή του κειμένου που θέλουμε να αποκόψουμε μετά κλικ **Mouse-3** στο τέλος του κειμένου. Αμέσως μετά ένα δεύτερο κλικ **Mouse-3** και ... πάει (στην πραγματικότητα το κείμενο αντιγράφεται στο Kill ring³¹ και είναι διαθέσιμο για επικόλληση).
- **Αποκοπή κειμένου από το πληκτρολόγιο**: Επιλέγουμε το αρχικό σημείο και θέτουμε το σημάδι (mark) με `C-SPC`. Μετακινούμε το δρομέα αμέσως μετά το σημείο που θέλουμε να θέσουμε ως τέλος της περιοχής. Πληκτρολογούμε `C-w`.
- **Αντιγραφή κειμένου με το ποντίκι**: Σέρνουμε το ποντίκι **Drag-Mouse-1** από την αρχή ως το τέλος. Η περιοχή ‘φωτίζεται’. Εναλλακτικά όπως στην αποκοπή χωρίς το δεύτερο κλικ στο τέλος: **Mouse-1** στην αρχή και **Mouse-3** στο τέλος.
- **Αντιγραφή κειμένου με το πληκτρολόγιο**: `C-SPC` στην αρχή, μετακινούμαστε το τέλος της περιοχής και μετά `M-w`.

³¹ Ας πούμε το clipboard του Emacs

- Επικόλληση κειμένου με το ποντίκι: Στο σημείο που θέλουμε να κάνουμε την εισαγωγή, πατάμε το μεσαίο κουμπί³².
- Επικόλληση κειμένου με το πληκτρολόγιο: Στο σημείο που θέλουμε να κάνουμε εισαγωγή C-y
- Επικόλληση κειμένου που είχαμε αντιγράψει παλιότερα: Μία εύκολη επιλογή είναι από το μενού `Edit→Paste from kill menu` και επιλέγουμε το κείμενο που θέλουμε να επικολλήσουμε. Από το πληκτρολόγιο όπως πριν C-y και αμέσως μετά M-y επανειλημμένα μέχρι να εμφανιστεί το κείμενο που θέλουμε.
- Εισαγωγή ολόκληρου αρχείου: Αν θέλουμε να εισάγουμε τα περιεχόμενα ενός ολόκληρου αρχείου πληκτρολογούμε C-x i στο σημείο που θέλουμε να γίνει η εισαγωγή. Εναλλακτικά με την εντολή M-x `insert-file`.
- Εισαγωγή ολόκληρου buffer: Αν θέλουμε να εισάγουμε τα περιεχόμενα ενός ολόκληρου buffer το κάνουμε με την εντολή M-x `insert-buffer`.
- Αντικατάσταση κειμένου: Με την εντολή M-x `query-replace` αντικαθιστούμε διαδραστικά μία ακολουθία χαρακτήρων με μία άλλη. Στην ερώτηση αν θέλουμε να γίνει η αντικατάσταση απαντούμε y (ναι), n (όχι), q (στοπ). Με , (κόμμα) γίνεται μία αντικατάσταση και σταματάει. Αν είμαστε σίγουροι ότι θέλουμε να γίνουν όλες οι αντικαταστάσεις εκτελούμε την εντολή M-x `replace-string`.
- Αντικατάσταση κεφαλαίων-μικρών γραμμάτων: Επιλέγουμε μία περιοχή που θέλουμε να γίνει η αλλαγή και εκτελούμε μία από τις εντολές M-x `upcase-region`, M-x `capitalize-region`, M-x `downcase-region`.

Το `Edit` μενού έχει πολλές από τις παραπάνω λειτουργίες για τους νεοσύλλεκτους.

Τίποτα επίσης δε μας εμποδίζει οι αποκοπές, αντιγραφές, επικολλήσεις να γίνονται από το ένα παράθυρο στο άλλο ακόμα και αν πρόκειται για buffer συνδεδεμένα με διαφορετικά αρχεία.

³²Αν δεν έχει, τη ροδέλα ή αριστερό και δεξί ταυτόχρονα.

1.3.5 Παράθυρα

Πολλές φορές είναι βολικό να επεξεργαζόμαστε το ίδιο ή διαφορετικά αρχεία σε διαφορετικά παράθυρα. Το ‘παράθυρο’ (window) στον Emacs αναφέρεται σε διαφορετικές περιοχές του ίδιου παραθύρου με την έννοια που δίνουμε σε ένα παραθυρικό περιβάλλον. Ο Emacs μπορεί να χωρίσει ένα παράθυρο σε ένα ή περισσότερα παράθυρα οριζόντια ή κάθετα. Μελετήστε το Σχήμα 1.5 στη σελίδα 65 στο οποίο επεξηγούνται οι βασικές έννοιες. Επίσης μπορεί να ανοίξει ένα διαφορετικό παράθυρο με την έννοια του παραθυρικού περιβάλλοντος τα οποία λέγονται ‘πλαίσια’ (frames)³³. Θα κρατήσουμε αυτή την ορολογία όταν αναφερόμαστε στον Emacs.

- Τοποθέτηση δρομέα στο κέντρο παραθύρου και καθαρισμός παραθύρου από σκουπίδια: `C-1`
- Χωρισμός παραθύρου στα δύο οριζόντια: `C-x 2`
- Χωρισμός παραθύρου στα δύο κάθετα: `C-x 3`
- Κατάργηση των άλλων παραθύρων: `C-x 1`
- Κατάργηση του τρέχοντος παραθύρου: `C-x 0`
- Μετακίνηση δρομέα σε άλλο παράθυρο: Με `Mouse-1` ή `C-x o`
- Αλλαγή μεγέθους παραθύρων: Με το ποντίκι `Drag-Mouse-1` στις διαχωριστικές γραμμές τους. Με το πληκτρολόγιο `C-^` αλλάζει την οριζόντια διάσταση και `C-}` την κάθετη.
- Καινούργιο πλαίσιο: `C-x 5 2`
- Κατάργηση πλαισίου: `C-x 5 0`
- Μετακίνηση δρομέα σε άλλο πλαίσιο: Με το ποντίκι `Mouse-1` ή με `C-x 5 o`.

³³Να σημειώσουμε πως όταν θέλετε να επεξεργαστείτε ένα ή περισσότερα αρχεία είναι καλό να ανοίγετε καινούργια πλαίσια στην ίδια συνεδρία του Emacs και όχι να ξεκινάτε κάθε φορά τον Emacs από την αρχή. Μια καινούργια διαδικασία Emacs τραβάει πόρους από το σύστημά σας και δεν επικοινωνεί με μία άλλη.

Διαφορετικά παράθυρα μπορείτε να έχετε και όταν ο Emacs τρέχει στην κονσόλα, κάτι που μπορεί να είναι η μεγαλύτερη ευλογία για προχωρημένη επεξεργασία κειμένου όταν δε βρίσκεστε σε παραθυρικό περιβάλλον. Φυσικά τότε δεν μπορείτε να έχετε διαφορετικά πλαίσια.

1.3.6 Αρχεία και Buffers

- Άνοιγμα αρχείου: `C-x C-f` ή `M-x find-file`.
- Σώσιμο αλλαγών του buffer σε αρχείο: `C-x C-s` ή `M-x save-buffer`.
Αν θέλουμε ταυτόχρονα να βγούμε από τον Emacs `C-x C-c` ή `M-x save-buffers-kill-emacs` (στο μενού `File→Save` ή εικονίδιο με δισκέτα).
- Σώσιμο αλλαγών σε buffer σε άλλο αρχείο: `C-x C-w` ή `M-x write-file` (στο μενού `File→Save As` ή στο εικονίδιο με δισκέτα και μολύβι).
- Σώσιμο όλων των buffers στα αρχεία τους: `C-x s` ή `M-x save-some-buffers`.
- Σύνδεση ενός buffer με ένα (άλλο) αρχείο: `M-x set-visited-file-name`.
- Κατάργηση buffer: `C-x k`
- Αλλαγή buffer στο παράθυρο που βρισκόμαστε: `C-x b`
- Προβολή λίστας buffer: Από το μενού `Buffers` ή με την εντολή `C-x C-b`. Στη δεύτερη περίπτωση πατώντας `Enter` δίπλα σε ένα buffer το εμφανίζουμε στο παράθυρο. Υπάρχουν εντολές διαχείρισης των buffers που μπορείτε να βρείτε από τη βοήθεια του Emacs (αν βάλετε το δρομέα στο παράθυρο αυτό πληκτρολογήστε `C-h m`)
- Ανάκτηση δεδομένων από επεξεργασμένο buffer: Αν χάσατε τη συνεδρία ενός Emacs μην απελπίζεστε. Στην καινούργια συνεδρία πληκτρολογήστε `M-x recover-file` και ακολουθήστε τις οδηγίες. Η εντολή `M-x recover-session` επαναφέρει όλα τα αρχεία που επεξεργαζόσαστε μαζί.
- Αρχεία ασφαλείας: Όταν σώζετε ένα buffer σε ένα αρχείο, το παλιό αρχείο γίνεται αντίγραφο ασφαλείας. Αν το αρχείο έχει όνομα `myfile` το αντίγραφο ασφαλείας έχει όνομα `myfile~`. Είναι δυνατόν να παραμετροποιήσετε τον Emacs να κρατάει πολλές ‘εκδόσεις’ (versions) των αλλαγών που κάνετε. Σας παραπέμπουμε στην τεκμηρίωση.

- Πλοήγηση και δράσεις σε καταλόγους: `C-x d` ή `M-x dired`. Μπορείτε να δράσετε στα αρχεία του καταλόγου (άνοιγμα, διαγραφή, αλλαγή ονομασίας, αντιγραφή κλπ) με τις ανάλογες εντολές (μέσα από το παράθυρο του `dired` δώστε την εντολή `C-h m` και διαβάστε τη σχετική τεκμηρίωση).

1.3.7 Modes

Σε κάθε buffer που επισκεπτόμαστε ο Emacs μπορεί να βρίσκεται σε διαφορετικά modes (όχι ένα, αλλά πολλά). Σε διαφορετικά modes οι συντομεύσεις των εντολών από το πληκτρολόγιο μπορεί αν είναι διαφορετικές, ο χρωματισμός των δομικών στοιχείων του buffer διαφορετικός κλπ. Υπάρχουν major modes που είναι μοναδικές για κάθε buffer αλλά και minor modes που μπορεί να συνυπάρχουν αρμονικά μαζί με άλλες major και minor modes. Ο Emacs μπορεί να ξεκινά αυτόματα μία major και μία ή περισσότερες minor modes ανάλογα με το όνομα ή/και το περιεχόμενο του αρχείου που επισκεπτόμαστε. Μπορούμε όμως και εμείς ρητά να επιλέξουμε και να επιβάλλουμε τις modes που επιθυμούμε με τις κατάλληλες εντολές.

Οι modes οι οποίες είναι ενεργές σε ένα buffer σημειώνονται μέσα σε παρένθεση στη mode line (βλ. Σχήμα 1.3 και 1.5).

- `M-x fortran-mode`: Μας ενδιαφέρει ιδιαίτερα στο μάθημα αυτό. Αφορά αρχεία με εντολές στη γλώσσα Fortran και τα πιο χρήσιμα χαρακτηριστικά τις είναι η τοποθέτηση του δρομέα στο κατάλληλο σημείο πατώντας το πλήκτρο TAB, ο χρωματισμός των εντολών και των μεταβλητών, η αναγνώριση των δομικών στοιχείων του προγράμματος (subroutines, if, do loops, comments, statement labels κλπ). Μια άλλη ενδιαφέρουσα λειτουργία είναι να πάρει τις εντολές μιας ολόκληρης περιοχής και να τις κάνει σχόλια (comments) δηλ. ανενεργές.
- `M-x c-mode`: Για κώδικα γραμμένο στη γλώσσα C. Ανάλογες modes για γλώσσες προγραμματισμού είναι οι `java-mode`, `perl-mode`, `awk-mode`, `python-mode`, `makefile-mode`, `octave-mode`, `mathematica-mode` και άλλες.
- `M-x latex-mode`: Για την επεξεργασία αρχείων που έχουν κείμενο σε $\text{\LaTeX}$.
- `M-x text-mode`: Για την επεξεργασία απλών (`.txt`) αρχείων κειμένου.
- `M-x fundamental-mode`: Η βασική mode, όταν δεν υπάρχει καλύτερη...

Minor modes που παρουσιάζουν ενδιαφέρον είναι οι

- **M-x auto-fill-mode:** Όταν μία γραμμή γίνεται πολύ μακριά την κόβει αυτόματα. Σχετική είναι η εντολή **M-x fill-paragraph**.
- **M-x overwrite-mode:** Αντί να εισάγονται οι χαρακτήρες ανάμεσα στους υπάρχοντες, γράφονται από πάνω τους. Δίνοντας την εντολή ξανά αλλάζουμε στην προηγούμενη κατάσταση.
- **read-only mode:** Όταν θέλουμε να επισκεφτούμε ένα πολύτιμο αρχείο που δε θέλουμε να αλλάξουμε κατά λάθος τα περιεχόμενά του μπαίνοντας σε αυτή τη mode ο Emacs απαγορεύει τις αλλαγές. Αυτό μπορεί να γίνει ανοίγοντας ένα αρχείο με την εντολή **C-x C-r (M-x find-file-read-only)** ή/και να εναλλάσσουμε τη mode με την εντολή **C-x C-q (M-x toggle-read-only)**. Βλέπε Σχήμα 1.5, το buffer `jack.c` που σημειώνεται στο mode line με `%%`.
- **M-x flyspell-mode:** Άμεσος έλεγχος ορθογραφίας.
- **M-x font-lock-mode:** Χρωματισμός δομικών στοιχείων του buffer. Συνήθως είναι η προεπιλογή, αν όχι την ενεργοποιούμε (ή την απενεργοποιούμε) με την εντολή.

Σε παραθυρικό περιβάλλον έχουμε τη δυνατότητα να επιλέξουμε modes από την mode line. Με το Mouse-3 πάνω στο όνομα μιας mode μας δίνονται επιλογές για την (απ)ενεργοποίηση minor modes. Με το Mouse-1 μπορούμε να (απ)ενεργοποιήσουμε την read-only mode κάνοντας κλικ αριστερά στο `:%%` ή `--` αντίστοιχα. Βλέπε Σχήμα 1.5.

1.3.8 Βοήθεια στον Emacs

Ο Emacs έχει πολύ λεπτομερή online τεκμηρίωση. Στους νέους χρήστες συστήνεται να ακολουθήσουν τις οδηγίες στο emacs tutorial το οποίο εκπαιδεύει το χρήστη στις βασικές εντολές χρήσης και επεξεργασίας κειμένου. Αυτό γίνεται με την εντολή **C-h t** ή από το μενού **Help→Emacs Tutorial**. Αφεθείτε στις οδηγίες και είναι ... διασκεδαστικό. Η man page του (εντολή **man emacs**) έχει συνοπτικές πληροφορίες, κυρίως για τον τρόπο που καλείται ο Emacs από τη γραμμή εντολών.

Πολύ εκτενής τεκμηρίωση βρίσκεται στις `info pages`³⁴. Η χρήση του `info` είναι κεφάλαιο βιβλίου από μόνη της, αλλά στο παραθυρικό περιβάλλον του Emacs είναι σχετικά απλή. Με την εντολή `C-h r` (μενού `Help→Read the Emacs Manual`) ανοίγουμε απευθείας τη σελίδα του Emacs. Πατώντας τα πλήκτρα `SPC` και `Backspace` διαβάζουμε την τεκμηρίωση σελίδα-σελίδα. Αλλά στο `info` έχουμε υπερσυνδέσμους όπως στην πλοήγηση στο διαδίκτυο. Με `Mouse-1` επιλέγετε ένα σύνδεσμο και με τα βελόνια στα εικονίδια μπορείτε να πάτε στον προηγούμενο/επόμενο σύνδεσμο όπως και στην προηγούμενη θέση που είσαστε. Πατώντας πλήκτρα δίνετε εντολές στο `info`, λ.χ. πατώντας `d` βρίσκεστε στον κεντρικό κατάλογο του `info` και μπορείτε να δείτε όλες τις εφαρμογές που έχουν `info` τεκμηρίωση. Με την εντολή `g (info)` (πληκτρολογήστε τους χαρακτήρες όπως τους βλέπετε με τις παρενθέσεις) βρίσκεστε στην τεκμηρίωση του `info` και εκεί μπορείτε να μάθετε την προχωρημένη χρήση της για να διαβάσετε αποτελεσματικά την τεκμηρίωση των εφαρμογών.

Ο Emacs είναι δομημένος διαισθητικά και φιλικά προς το χρήστη. Τα περισσότερα θα τα μάθετε όχι από προσεκτική μελέτη του εγχειρίδιου χρήσης αλλά από τα ίδια τα ονόματα των εντολών και τη συνοπτική τεκμηρίωσή τους. Όλες οι εντολές του Emacs αποτελούνται από ολόκληρες λέξεις που χωρίζονται με ένα - που σχεδόν σχηματίζουν πλήρεις προτάσεις.

- **auto completion εντολών:** Οι εντολές αυτοσυμπληρώνονται στο minibuffer πατώντας το πλήκτρο `TAB`. Για παράδειγμα, μεταβείτε στο minibuffer πληκτρολογώντας `M-x`. Πληκτρολογήστε λ.χ. `capi`[`TAB`] και η εντολή συμπληρώνεται σε `capitalize-`. Πατώντας το [`TAB`] άλλη μια φορά ανοίγει ένα buffer με τις δυνατές επιλογές: `capitalize-region` και `capitalize-word`. Πληκτρολογήστε `r`[`TAB`] και η εντολή συμπληρώνεται στη μοναδική `capitalize-region`. Για να δείτε όλες της εντολές που αρχίζουν από `s` (...πολλές) πληκτρολογήστε `M-x s`[`TAB`][`TAB`]. Επιλέξτε με το ποντίκι το buffer `*Completions*` και πλοηγηθείτε στις δυνατότητες. Από τα ονόματα των εντολών θα πάρετε ιδέες. Αν έχετε χρόνο πληκτρολογήστε `M-x` [`TAB`][`TAB`] και όλες οι διαθέσιμες εντολές θα είναι στο ... buffer σας!
- **Τεκμηρίωση συντομεύσεων πληκτρολογίου:** Δεν ξέρετε τι κάνει η εντολή `C-s`; Κανένα πρόβλημα... Πληκτρολογήστε `C-h k` και μετά `C-s`. Αντί-

³⁴ Αν προτιμάτε έγγραφα-βιβλία σε μορφή PDF επισκεφτείτε την ιστοσελίδα του Emacs www.gnu.org/software/emacs/ και επιλέξτε `Documentation`. Αυτός ο σύνδεσμος τώρα είναι ο www.gnu.org/software/emacs/manual/emacs.html από όπου μπορείτε να κατεβάσετε το εγχειρίδιο χρήσης των 600 περίπου σελίδων!

στροφα, ξεχάσατε με ποια πλήκτρα συντομεύεται η εντολή `save-buffers`; Πληκτρολογήστε `C-h w` και μετά την εντολή.

- Τεκμηρίωση συναρτήσεων: Ψάχνετε μια εντολή λ.χ. `save-κάτι-που-ξέχασα`; Πληκτρολογήστε `C-h f` και μετά `save-[TAB]` να δείτε τις επιλογές. Με Mouse-2 επιλέξτε την εντολή που σας ενδιαφέρει ή πληκτρολογήστε/συμπληρώστε με `[TAB]` το υπόλοιπο όνομα (λ.χ. `save-buffers-kill-emacs`) και στο buffer `*Help*` θα διαβάσετε την τεκμηρίωση της εντολής.
- Τεκμηρίωση μεταβλητών: Με `C-h v` μπορείτε να μάθετε την τιμή και τη λειτουργία των μεταβλητών στον Emacs.
- Τεκμηρίωση `apropos`: Δεν θυμάστε ακριβώς το όνομα της εντολής; Κάνένα πρόβλημα... Πληκτρολογήστε `C-h a` και μια λέξη (λ.χ. `save`) και θα δείτε όλες τις εντολές που περιέχουν τη λέξη αυτή. Περισσότερες πληροφορίες θα πάρετε με `C-h d`
- Τεκμηρίωση `modes`: Πληκτρολογώντας `C-h m` μέσα από ένα buffer παίρνετε πληροφορίες για τις `modes` που είναι ενεργοποιημένες για το buffer. Θα δείτε εκεί τις ειδικές εντολές που δένονται με τις συντομεύσεις πληκτρολογίου.
- Τεκμηρίωση `info`: `C-h i`
- Ξεχάσατε τα παραπάνω ή θέλετε να μάθετε και άλλα; Πληκτρολογήστε `C-h ?` και πλοηγηθείτε στις επιλογές που σας δίνονται.

1.3.9 Παραμετροποίηση του Emacs

Ο Emacs έχει δυνατότητα παραμετροποίησης σε οποιοδήποτε βάθος: Από την απλή σύνδεση πλήκτρων με εντολές που θέλουμε να συντομεύσουμε μέχρι τον προγραμματισμό πολύπλοκων λειτουργιών στη γλώσσα `Elisp`. Ο πιο διαδεδομένος τρόπος για τον μέσο χρήστη είναι να εισάγει τις κατάλληλες εντολές στο αρχείο `~/.emacs` στην προσωπική του περιοχή. Ο Emacs διαβάζει και εκτελεί τις εντολές αυτές πριν ξεκινήσει. Παράδειγμα ενός τέτοιου αρχείου με ενδεικτικές λειτουργίες είναι το παρακάτω:

```
; Define F1 key to save the buffer
(global-set-key [f1] 'save-buffer)
; Define Control-c s to save the buffer
```



```
(global-set-key "\C-cs" 'save-buffer)
; Define Meta-s (Alt-s) to interactively search forward
(global-set-key "\M-s" 'isearch-forward)
; Define M-x is          to interactively search forward
(defalias 'is          'isearch-forward)
; Define M-x fm          to set fortran-mode for the buffer
(defun fm() (interactive) (fortran-mode))
; Define M-x sign        to sign my name
(defun sign() (interactive) (insert "Konstantinos Anagnostopoulos"))
```

Στα περιεχόμενα του παραπάνω αρχείου τα ελληνικά ερωτηματικά ; ορίζουν το υπόλοιπο της γραμμής να είναι σχόλια. Οι πρώτες τρεις εντολές δεσμεύουν τα πλήκτρα `F1`, `C-c s` και `M-s` σε συγκεκριμένες συναρτήσεις-εντολές. Η επόμενη δείχνει πώς να ορίσουμε ψευδώνυμο (alias) μιας εντολής που χρησιμοποιούμε συχνά. Οι τελευταίες δύο ορίζουν δύο πολύ απλές συναρτήσεις (`fm`) και (`sign`) που μπορούμε να τις καλέσουμε από το minibuffer όπως αναφέρεται στα σχετικά σχόλια.

Για περισσότερα παραδείγματα αναζητήστε στο Google: “emacs .emacs file” για να δείτε τα αρχεία που χρησιμοποιούν άλλοι χρήστες.

Επίσης είναι δυνατόν να παραμετροποιήσετε τον Emacs από το μενού `Options` → `Customize Emacs`.

Για τη σε βάθος εκμάθηση της γλώσσας `Elisp` σας παραπέμπουμε στο Emacs Lisp Reference Manual στη διεύθυνση www.gnu.org/software/emacs/manual/elisp.html

1.3.10 Ελληνικά στον Emacs

Με πολλή συντομία περιγράφουμε πώς γίνεται να επεξεργαστούμε αρχεία με ελληνικούς χαρακτήρες. Εδώ ο χρήστης πρέπει να προσδιορίσει αν οι ελληνικοί χαρακτήρες θα αναπαρίστανται από τους 8-bit χαρακτήρες του συστήματος `iso8859-7`³⁵ ή από τους πιο διαδεδομένους 16-bit Unicode χαρακτήρες.

Για να μπορέσουμε να διαβάσουμε αρχεία με χαρακτήρες Unicode πρέπει ο Emacs σε ένα παραθυρικό περιβάλλον να ξεκινήσει με μία κατάλληλη γραμματοσειρά Unicode (UTF)³⁶. Αν αυτό δεν είναι η προεπιλογή επιλέγουμε εμείς μία γραμματοσειρά³⁷. Μια επιλογή δίνεται από την παρακάτω εντολή:

³⁵Χρήσιμοι σε προγράμματα όπως το `LaTeX` με τη χρήση του `Babel`.

³⁶Αν καλούμε τον Emacs στην κονσόλα, θα πρέπει η κονσόλα να εμφανίζει χαρακτήρες Unicode

³⁷Με την εντολή `xlsfonts | grep iso10646 | less` βλέπουμε τις διαθέσιμες γραμματο-

```
> emacs -fn -misc-fixed-medium-r-normal--18-120-100-100-c-90-iso10646-1 &
```

Στη συνέχεια μπορούμε να εισάγουμε ελληνικούς χαρακτήρες χρησιμοποιώντας την αλλαγή της μεθόδου πληκτρολογίου του παραθυρικού περιβάλλοντος (λ.χ. πληκτρολογώντας **Alt-Shift**) όπως και σε οποιαδήποτε άλλη εφαρμογή. Εναλλακτικά (αν λ.χ. δεν είμαστε σε UTF περιβάλλον) με την εντολή **C-\ (M-x toggle-input-method)** και εισάγοντας –μόνο την πρώτη φορά– ‘greek’ στο minibuffer εναλλάσσουμε από αγγλικά σε ελληνικά.

Για τους 8-bit χαρακτήρες τύπου ISO8859-7 καλούμε τον Emacs με την ανάλογη γραμματοσειρά. Μία επιλογή είναι

```
emacs -fn -misc-fixed-medium-r-normal--18-120-100-100-c-90-iso8859-7 &
```

Αφού ανοίξουμε το αρχείο που επιθυμούμε σε ένα buffer μπορούμε από το μενού να διαλέξουμε περιβάλλον γλώσσας **Options->Mule(Multilingual environment)->Set Language Environment->Greek** (ή στο minibuffer **M-x set-language-environment**) και επιλέγει μέθοδο εισαγωγής χαρακτήρων **Options->Mule(Multilingual environment)->Select Input Method-> ‘greek’** (ή στο minibuffer **M-x toggle-input-method**). Στη συνέχεια με τη συντόμευση εντολής **C-** εναλλάσσουμε από αγγλικά σε ελληνικά.

1.4 Η Γλώσσα Προγραμματισμού: Fortran 77

Στην παράγραφο αυτή θα αναφέρουμε τα απολύτως απαραίτητα που χρειάζεται να ξέρετε προκειμένου να αρχίσετε να γράφετε και να τρέχετε προγράμματα σε γλώσσα Fortran 77. Δεν πρόκειται για συστηματική εκμάθηση της γλώσσας, αλλά για μια πρακτική προσέγγιση μέσω παραδειγμάτων.

Για να ωφεληθεί ο/η αναγνώστης/τρια από το κεφάλαιο αυτό πρέπει απαραίτητα να γράφει τα προγράμματα και να τα εκτελεί στον υπολογιστή του/της.

1.4.1 Τα Στοιχειώδη

Το πρώτο πρόγραμμα που γράφει κανείς σε μια καινούργια γλώσσα ή/και υπολογιστικό περιβάλλον, είναι ένα ‘Hello World’ πρόγραμμα, το οποίο απλά

τοσειρές Unicode ενώ με την εντολή `xlsfonts | grep iso8859-7|less` τις διαθέσιμες ελληνικές 8-bit γραμματοσειρές ISO8859-7.

τυπώνει στο `stdout` αυτή τη φράση. Καταφέροντας να δει τη φράση αυτή τυπωμένη, έχει κάνει τη μισή δουλειά που χρειάζεται για να προγραμματίσει στο περιβάλλον αυτό. Το εν λόγω πρόγραμμα σε Fortran 77 το γράφουμε σε ένα αρχείο `hello.f` ως εξής:

```

      program hello_world

C      print a message
      print *, 'Hello world!'

      end                                !this is the end

```

Οι εντολές στη Fortran 77 είναι ακολουθίες χαρακτήρων που γράφουμε από την 7η μέχρι και την 72η στήλη. Η γλώσσα αυτή έχει στο συντακτικό της αυτή την ιδιομορφία³⁸ όπου είναι σημαντικό σε ποια στήλη γράφουμε κάτι. Οτιδήποτε γραφτεί μετά την 72η στήλη αγνοείται. Αν γράψετε το χαρακτήρα³⁹ `C` στην πρώτη στήλη, τότε όλη η γραμμή αγνοείται από το μεταγλωττιστή και μπορείτε να γράψετε οτιδήποτε ως σχόλιο (comment). Η τρίτη σειρά του παραπάνω προγράμματος είναι ένα σχόλιο σύμφωνα με αυτό το συντακτικό. Οι στήλες 2-5 καθώς και η 6η έχουν και αυτές ειδικό ρόλο τον οποίο θα δούμε αργότερα.

Η κύρια είσοδος σε ένα πρόγραμμα καθορίζεται από την εντολή `program name` όπου `name` είναι ό,τι θέλουμε αρκεί να αποτελείται από αλφαριθμητικούς χαρακτήρες και ορισμένους άλλους, όπως ο `_`. Το τέλος του προγράμματος, όπως και κάθε αυτοδύναμης ενότητας του προγράμματος (υπορουτίνες, συναρτήσεις), καθορίζεται από την εντολή `end`. Στη σχετική γραμμή παραπάνω παρατηρούμε ότι γράψαμε `!this is the end`. Αυτός είναι ένας άλλος τρόπος να γράφουμε σχόλια στο πρόγραμμα, οτιδήποτε μετά το `!` αγνοείται οπότε γράφουμε ότι θέλουμε.

Στην τέταρτη γραμμή είναι το 'ζουμί': Η εντολή `print` είναι ο απλούστερος τρόπος να τυπώσουμε κάτι στο `stdout`. Προσέξτε το `*,` που είναι μέρος του συντακτικού και φυσικά δεν τυπώνεται... Για τη Fortran 77 τα κεφαλαία/μικρά γράμματα είναι ισοδύναμα και θα μπορούσαμε να γράφουμε `PRINT`, `Print`, Η φράση που θέλουμε να τυπώσουμε είναι μια ακολουθία χαρακτήρων που περικλείεται από μονά εισαγωγικά.

³⁸ Αυτό προέρχεται από την 'αρχαιότητα' όπου οι εντολές εισάγονταν στη μνήμη του υπολογιστή γραμμένες σε ξεχωριστές κάρτες με τρύπες!

³⁹ ή `*`.

Για να τρέξει το πρόγραμμα, πρέπει να μεταφραστεί σε γλώσσα μηχανής. Τη δουλειά αυτή την αναλαμβάνει ο μεταγλωττιστής (compiler)⁴⁰. Σε κάθε σύστημα το πρόγραμμα αυτό μπορεί να έχει διαφορετικό όνομα ή ακόμα και ο προγραμματιστής να έχει περισσότερες από μία επιλογές. Πρέπει να ενημερωθείτε από το διαχειριστή του συστήματος ή τα σχετικά εγχειρίδια. Τυπικά ονόματα τέτοιων προγραμμάτων είναι f77, f90, g77, ifort, gfortran, Η πρώτη μας δουλειά είναι να μελετήσουμε με προσοχή τα εγχειρίδια χρήσης. Εκεί μαθαίνουμε πώς να χρησιμοποιήσουμε τις δυνατότητές του με τον καλύτερο τρόπο για το δικό μας πρόγραμμα (λ.χ. βελτιστοποίηση - optimization)

Στο δικό μας σύστημα θα χρησιμοποιήσουμε τον f77. Η εντολή που θα δώσουμε για τη μεταγλώττιση είναι⁴¹

```
> f77 hello.f -o hello
```

Ο διακόπτης -o ορίζει ο μεταγλωττισμένος κώδικας να γραφτεί στο αρχείο hello. Αν η μεταγλώττιση είναι επιτυχής τότε το πρόγραμμα τρέχει με την εντολή:

```
> ./hello
Hello world!
```

όπου το ./ το βάλαμε για να προσδιορίσουμε ρητά πως εκτελούμε το πρόγραμμα στο αρχείο hello που βρίσκεται στον τρέχοντα κατάλογο (.).

Ας δοκιμάσουμε τώρα να κάνουμε ένα απλό υπολογισμό, την περίμετρο και το εμβαδόν ενός κύκλου ακτίνας R. Για το λόγο αυτό θα χρειαστούμε να χρησιμοποιήσουμε μεταβλητές τύπου REAL. Στο αρχείο area_01.f πληκτρολογούμε

```
C      file: area_01.f
      program circle_area

      PI = 3.14159265358979
      R  = 4.0
      print *, 'Perimeter= ', 2.0*PI*R
      print *, 'Area=      ', PI*R**2

      end
```

⁴⁰Στην πραγματικότητα αυτή η δουλειά γίνεται σε διάφορα στάδια και από διαφορετικά προγράμματα. Περισσότερα στην τάξη...

⁴¹Θυμίζουμε στον αναγνώστη ότι γραμμές που αρχίζουν με > είναι εντολές που δίνουμε στη γραμμή εντολών. Οτιδήποτε άλλο είναι output του προγράμματος.

Στο παραπάνω πρόγραμμα ορίσαμε τις τιμές των δύο μεταβλητών R , PI στη 3η και 4η γραμμή. Το ότι οι μεταβλητές είναι τύπου `REAL` καθορίζεται από το όνομα της μεταβλητής. Η Fortran 77 έχει implicit rules για να το καθορίζει. Σύμφωνα με αυτούς, μεταβλητές που το όνομά τους αρχίζει από i , j , k , l είναι τύπου `INTEGER` (ακέραιοι) ενώ κάθε άλλη είναι τύπου `REAL`. Αλλαγή γίνεται μόνο αν δηλώσουμε ρητά τον τύπο μιας μεταβλητής όπως θα δείξουμε αργότερα⁴². Στην 5η και 6η γραμμή κάνουμε τον υπολογισμό $2\pi R$ και πR^2 κατευθείαν στο όρισμα της εντολής `print`. Οι τελεστές πολλαπλασιασμού και δύναμης είναι `*` και `**` αντίστοιχα. Προσέξτε ότι στις σταθερές 2.0 και 4.0 βάλαμε ρητά την υποδιαστολή. Αν τις παραλείψουμε οι σταθερές είναι τύπου `INTEGER` και αν δεν είναι αυτό που θέλουμε το αποτέλεσμα μπορεί να μας ... καταπλήξει⁴³. Αν υποθέσουμε ότι το πρόγραμμα είναι αποθηκευμένο στο αρχείο `area_01.f`, οι εντολές μεταγλωττισμού και εκτέλεσης του προγράμματος είναι

```
> f77 area_01.f -o area
> ./area
Perimeter=      25.13274
Area=          50.26548
```

Ας δοκιμάσουμε τώρα μια επαναλαμβανόμενη διεργασία. Ας κάνουμε τον παραπάνω υπολογισμό για 10 διαφορετικούς κύκλους ακτίνας $R_i = 1.28 + i$, $i = 1, \dots, 10$. Τις ακτίνες θα τις αποθηκεύσουμε σε ένα array `R(10)` τύπου `REAL`. Το αρχείο `area_02.f`:

```
C      file: area_02.f
      program circle_area

      dimension R(10)

      PI      = 3.14159265358979
      R(1)    = 2.28
      do i = 2, 10
        R(i) = R(i-1) + 1.0
      enddo

      do i = 1, 10
        perimeter = 2*PI*R(i)
```

⁴²Μπορούμε να αλλάξουμε τους κανόνες αυτούς με την εντολή `implicit`.

⁴³Δοκιμάστε την εντολή `print *, 2.0/4.0, 2/4` στο παραπάνω πρόγραμμα.

```

      area      = PI*R(i)**2
      print *,i,') R= ',R(i),' area= ',area,' perimeter= ',perimeter
    enddo

  end

```

Η εντολή `dimension R(10)` ορίζει ένα array με διάσταση 10. Στη Fortran τα στοιχεία arrays αναφέρονται με ένα δείκτη που παίρνει τιμές από 1 μέχρι τη διάσταση του array (εδώ 10). Άρα `R(4)` είναι το τέταρτο στοιχείο του `R`.

Μεταξύ των εντολών

```

    do i = 2, 10
      ...
    enddo

```

περιέχονται εντολές που εκτελούνται επαναληπτικά με την ακέραια μεταβλητή `i` να παίρνει τιμή από 2 έως 10 με βήμα ⁴⁴1. Η εντολή

```

      R(i) = R(i-1) + 1.0

```

ορίζει την ακτίνα με δείκτη `i` να είναι κατά 1 μεγαλύτερη από την προηγούμενη. Για να είναι σωστή η επαγωγή θα πρέπει να ορίσουμε την τιμή της `R(1)` πριν αρχίσει το `do loop`. Μετά από αυτή την εξήγηση, νομίζω πως μπορεί εύκολα να γίνει κατανοητό τι γίνεται στο δεύτερο `do loop` του προγράμματος. Ο αναγνώστης θα πρέπει να δοκιμάσει το παραπάνω πρόγραμμα και να πειραματιστεί κάνοντας μικροαλλαγές.

Ας μετατρέψουμε τώρα το παραπάνω πρόγραμμα έτσι ώστε ο χρήστης να δίνει τις ακτίνες του κύκλου, το πρόγραμμα να υπολογίζει τις ακτίνες και τα εμβαδά και να γράφει το αποτέλεσμα σε ένα αρχείο. Άρα το πρόγραμμα πρέπει να πάρει input από το χρήστη τα μέτρα των ακτίνων $R_i, i = 1, \dots, 10$. Γράφουμε στο αρχείο `area_03.f`:

```

C      file: area_03.f
      program circle_area

      implicit none

      integer    N

```

⁴⁴Το βήμα μπορεί να αλλάξει λ.χ. `do i=0,12,4` τρέχει για `i=0,4,8,12` και η `do i=10,6,-2` για `i=10,8,6` αντίστοιχα.

```

real      PI
parameter(N=10)
parameter(PI=3.14159265358979)
real      R(N)
real      area,perimeter
integer   i

do i=1,N
  print*, 'Enter radius of circle: '
  read(5,*)R(i)
  print*, 'i= ',i,' R(i)= ',R(i)
enddo

open(UNIT=13,FILE='AREA.DAT')

do i = 1,N
  perimeter = 2*PI*R(i)
  area      = PI*R(i)**2
  write(13,*)i,' R= ',R(i),' area= ',area,
*           ' perimeter= ',perimeter
enddo

close(13)

end

```

Παρατηρήστε τώρα ότι η πρώτη εντολή που δίνουμε είναι η `implicit none`. Αυτό δηλώνει ότι δε θέλουμε να χρησιμοποιήσουμε τους `implicit` κανόνες της Fortran αλλά θέλουμε να υποχρεώσουμε τον εαυτό μας να δηλώσει ρητά κάθε μεταβλητή του προγράμματος. Αυτό σημαίνει πως θα μας πάρει λίγο παραπάνω χρόνο να πληκτρολογήσουμε τους ορισμούς αλλά σας υπόσχομαι ότι αυτός ο κόπος δεν συγκρίνεται με τίποτα με τον πόνο να βρει κάποιος δύσκολα σφάλματα στο πρόγραμμα που οφείλονται σε μικρά ορθογραφικά λάθη στα ονόματα των μεταβλητών⁴⁵. Θα ακολουθήσουμε αυτή την πρακτική σε ολόκληρο το βιβλίο.

Μετά από αυτή την εντολή ακολουθούν οι δηλώσεις των μεταβλητών. Οι μεταβλητές `N`, `i` δηλώνονται ως `integer` ενώ οι `PI`, `area`, `perimeter`, `R(N)`

⁴⁵ Ποια η διαφορά στο όνομα της μεταβλητής `p11` από την `p11`;

ως *real*. Οι *N*, *PI* δηλώνονται να είναι παράμετροι (*parameter*) των οποίων η τιμή δεν μπορεί να αλλάξει στη ροή του προγράμματος.

Μετά τις δηλώσεις των μεταβλητών ακολουθούν οι εκτελέσιμες εντολές. Το πρώτο *do loop* δεν έχει τίποτα καινούργιο εκτός από την εντολή

```
read(5,*)R(i)
```

Με την εντολή αυτή διαβάζουμε από το *stdin* την τιμή της μεταβλητής *R(i)*. Ο χρήστης πρέπει να την τυπώσει στο τερματικό και να πατήσει το πλήκτρο *[Enter]*. Μπορούμε με την ίδια εντολή να διαβάσουμε περισσότερες από μία μεταβλητές.

Για να τυπώσουμε δεδομένα σε ένα αρχείο στο σκληρό δίσκο πρέπει να συναρτήσουμε στο όνομα του αρχείου ένα *UNIT* που μπορεί να είναι οποιοσδήποτε ακέραιος μέσα σε κάποια όρια που καθορίζονται από το σύστημα⁴⁶. Η συνάρτηση αυτή γίνεται με την εντολή *open* και μετά μπορούμε να γράφουμε με την εντολή *write(unit_number,*)...*⁴⁷. Όταν τελειώσουμε κλείνουμε το αρχείο με την εντολή *close* και μπορούμε να χρησιμοποιήσουμε τον ίδιο αριθμό για άλλο αρχείο. Η λογική ροή είναι δηλαδή

```
open(UNIT=13,FILE='AREA.DAT')
...
write(13,*) ....
...
close(13)
```

Το όνομα του αρχείου καθορίζεται από το όρισμα *FILE='AREA.DAT'* της εντολής *open*.

Το τελευταίο που παρατηρούμε είναι η γραμμή

```
write(13,*)i,') R= ',R(i),' area= ',area,
*      ' perimeter= ',perimeter
```

που μας δείχνει πώς να συνεχίζουμε μια μακριά εντολή στην επόμενη γραμμή. Αρχεί να βάλουμε οποιοδήποτε χαρακτήρα μας αρέσει στην 5η στήλη (εδώ το **)* και η ζωή ... συνεχίζεται (και μάλιστα όσο μας αρέσει!).

Το επόμενο βήμα είναι να μάθουμε πώς να χωρίζουμε το πρόγραμμά μας σε λογικά διαφορετικές διαδικασίες οι οποίες μπορεί να επαναλαμβάνονται πολλές φορές στο πρόγραμμά μας. Θα δείξουμε τη διαδικασία της υπορουτίνας

⁴⁶Μπορείτε με ασφάλεια να χρησιμοποιήσετε από 10-99. Το 5 είναι το *stdin*, το 6 το *stdout* και το 0 το *stderr*.

⁴⁷Δοκιμάστε τι γίνεται αν γράψετε σε ένα *UNIT* χωρίς να έχετε ανοίξει ένα αρχείο...

(subroutine) ορίζοντας τον υπολογισμό του εμβαδού και της περιφέρειας του κύκλου να γίνεται από την subroutine `area_of_circle`. Ορίστε τι γράφουμε μέσα στο αρχείο `area_04.f`:

```
C      file: area_04.f
      program circle_area

      implicit none

      integer    N
      real       PI
      parameter(N=10)
      parameter(PI=3.14159265358979)
      real       R(N)
      real       area,perimeter
      integer    i

      do i=1,N
        print*,'Enter radius of circle: '
        read(5,*)R(i)
        print*,'i= ',i,' R(i)= ',R(i)
      enddo

      open(UNIT=13,FILE='AREA.DAT')

      do i = 1,N
        call area_of_circle(R(i),perimeter,area)
        write(13,*)i,') R= ',R(i),' area= ',area,
*          'perimeter= ',perimeter
      enddo

      close(13)

      end

      subroutine area_of_circle(R,L,A)
      implicit none
```

```

real R,L,A
real PI,PI2
parameter(PI=3.14159265358979)
parameter(PI2 = 2.0*PI)

L= PI2*R
A= PI*R*R

return
end

```

Οι αλλαγές που κάναμε αφορούν καταρχήν το κυρίως πρόγραμμα. Οι υπολογισμοί της περιμέτρου και του εμβαδού αντικαταστάθηκαν από τη γραμμή

```
call area_of_circle(R(i),perimeter,area)
```

Η εντολή `call` κάνει αυτό που λέει: καλεί τη διαδικασία που ορίζεται στην υπορουτίνα `area_of_circle`. Τα `(R(i),perimeter,area)` είναι τα ορίσματα της υπορουτίνας. Το `R(i)` είναι μεταβλητή εισόδου η οποία παρέχει δεδομένα για να κάνει τον υπολογισμό η υπορουτίνα. Οι `perimeter,area` είναι οι μεταβλητές εξόδου στις οποίες κατά την έξοδό της η υπορουτίνα μας δίνει τα αποτελέσματα. Ο προγραμματιστής της υπορουτίνας πρέπει να μας δώσει σαφείς οδηγίες για τις μεταβλητές εισόδου/εξόδου έτσι ώστε να χρησιμοποιήσουμε σωστά την υπορουτίνα.

Η υπορουτίνα προγραμματίζεται ανάμεσα στις δηλώσεις

```

subroutine area_of_circle(R,L,A)
...
end

```

Τα ορίσματα `R,L,A` ορίζονται στην υπορουτίνα και τα ονόματά τους δεν είναι αναγκαστικό να είναι τα ίδια με αυτά που χρησιμοποιούμε για να καλέσουμε την υπορουτίνα. Δηλώνονται ρητά με τις δηλώσεις `real R,L,A`. Οι μεταβλητές περνούν *by reference* το οποίο σε απλά ελληνικά σημαίνει πως οποιαδήποτε αλλαγή στις τιμές τους μέσα στην υπορουτίνα, αλλάζει και τις αντίστοιχες τιμές στο πρόγραμμα που την κάλεσε. Άρα με τις εντολές `L= PI2*R`, `A= PI*R*R` πετυχαίνουμε αυτό που θέλουμε, δηλ. να επιστρέψουμε στο χρήστη της υπορουτίνας την περίμετρο και το εμβαδόν κύκλου ακτίνας `R`. Τέλος με την εντολή `return` επιστρέφουμε τον έλεγχο στο πρόγραμμα που κάλεσε την υπορουτίνα. Οι μεταβλητές `PI`, `PI2` είναι 'ιδιωτικές' της `area_of_circle` και


```

C      This is the function that computes the discriminant (diakrinousa)
C      A function returns a value. This value is assigned with the
C      statement:
C      Discriminant = <value>
C      i.e. we simply assign anywhere in the program a variable with the
C      name of the function.
C
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      real function Discriminant(a,b,c)
C      implicit none
C      real a,b,c
C
C      Discriminant = b**2 - 4.0 * a * c
C
C      end
C
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C      The subroutine that computes the roots.
C
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      subroutine roots(a,b,c,x1,x2)
C      implicit none
C      real a,b,c
C      real x1,x2
C      real D, Discriminant
C
C      if(a .eq. 0.0) stop 'roots: a=0'
C
C      D = Discriminant(a,b,c)
C      if(D.ge.0.0)then
C        D = sqrt(D)
C      else
C        print *, 'roots: Sorry, cannot compute roots, D<0=',D
C        stop
C      endif
C
C      x1 = (-b + D)/(2.0*a)

```

```
x2 = (-b - D)/(2.0*a)
```

```
end
```

Το πρόγραμμα ζητάει τους συντελεστές του τριωνύμου $ax^2 + bx + c$. Ελέγχει ότι είναι καλά ορισμένο $a > 0$ και αν όχι σταματάει το πρόγραμμα με την εντολή **stop**. Στη συνέχεια υπολογίζει τη διακρίνουσα (discriminant) $D = b^2 - 4ac$ καλώντας τη συνάρτηση **Discriminant(a,b,c)**. Η συνάρτηση (function) διαφέρει από τη subroutine στο ότι καλείται απευθείας (χωρίς την εντολή **call**) και επιστρέφει μια τιμή της οποίας ο τύπος πρέπει να δηλωθεί όπως οποιαδήποτε άλλη μεταβλητή (**real Discriminant**). Στη συνέχεια ξεχωρίζουμε τις γνωστές περιπτώσεις με τη δομή

```
if(D .gt. 0.0 )then
...
else if (D .eq. 0.0) then
...
else
...
endif
```

όπου παρατηρούμε και τους τελεστές σύγκρισης **.gt.** (greater than-αυστηρά μεγαλύτερο) και **.eq.** (equal-ίσο)⁴⁸.

Η συνάρτηση **Discriminant** πρέπει να δηλωθεί τι τύπου τιμή επιστρέφει (εδώ **real**) καθώς και τα ορίσματά της όπως και για τη subroutine. Η τιμή που επιστρέφει καθορίζεται τοποθετώντας την σε μια μεταβλητή με όνομα ίδιο με αυτό της συνάρτησης:

```
real function Discriminant(a,b,c)
...
Discriminant = b**2 - 4.0 * a * c
...
end
```

1.4.2 Οι λεπτομέρειες

Την παράγραφο αυτή μπορείτε να την αγνοήσετε την πρώτη φορά που διαβάζετε αυτό το κεφάλαιο. Σκοπός είναι περισσότερο να χρησιμεύσει σαν αναφορά όταν

⁴⁸ Παρόμοιοι τελεστές είναι **.lt.**, **.ge.**, **.le.** (μικρότερο, μεγαλύτερο ή ίσο, μικρότερο ή ίσο) και **.ne.**, **.and.**, **.or.** (μη ίσο, λογικό και, λογικό ή)

θα έχετε απορίες στα επόμενα κεφάλαιο.

Ξεκινάμε αναφέροντας και άλλους ενδιαφέροντες τύπους μεταβλητών. Στο παρακάτω πρόγραμμα δείχνουμε πώς να χρησιμοποιήσετε μεταβλητές τύπου CHARACTER, πραγματικούς διπλής ακρίβειας REAL*8 και μιγαδικούς αριθμούς μονής COMPLEX και διπλής ακρίβειας COMPLEX*16:

```

program f77_vars
implicit none

character *10 string

real      x
real*8    x8 !equivalent to: double precision x8
C  Complex Numbers:
complex    z
complex*16 z16 !double precision

C  A string array:
string = 'Hello World!' !string smaller size than necessary
print *, 'A string: ', string
C  Reals with increasing accuracy: Determine PI
x  = 4.0 *atan(1.0 )
x8 = 4.0D0*atan(1.0D0) !Use D for double    precision exponent
print *, 'x4= ',x,' x8= ',x8

C  Complex numbers:
z = (2.0,1.0)*cexp((3.0,-1.0)) ! z= (2 + i) * e^(3-i)
print *, 'z= ',z,' Re(z)= ',REAL(z),' Im(z)= ',AIMAG(z),
*      ' |z|= ',ABS(z),' z*= ',CONJG(z)

C  Complex numbers of double precision:
z16 = (2.0D0,1.0D0)*cdexp((3.0D0,-1.0D0))
print *, 'z= ',z16,' Re(z)= ',DBLE(z16),' Im(z)= ',DIMAG(z16),
*      ' |z|= ',CDABS(z16),' z*= ',DCONJG(z16)

end

```

Τα σημεία που πρέπει να προσέξουμε στο παραπάνω πρόγραμμα είναι:

- Οι μεταβλητές τύπου **CHARACTER** δηλώνονται με το μέγεθός τους, εδώ 10 χαρακτήρες βάζοντας ***10**. Αν περάσουμε το όριο αυτό οι παραπάνω χαρακτήρες ... κόβονται.
- Όταν χρησιμοποιούμε σταθερές στις μεταβλητές διπλής ακρίβειας βάζουμε πάντα τον εκθέτη έστω και αν είναι 0. Ο εκθέτης υποδηλώνεται με το γράμμα **D** αντί του **E** που χρησιμοποιείται για τις μεταβλητές **REAL**. Αλλιώς η σταθερά χάνει την επιθυμητή ακρίβεια.
- Οι συναρτήσεις για μεταβλητές διπλής ακρίβειας συνήθως παίρνουν ένα έξτρα **D** στο όνομά τους (**exp**→**dexp**, **ABS**→**DABS**) ενώ οι αντίστοιχες για μιγαδικούς ένα έξτρα **C** (**DABS**→**CDABS**, **exp**→**cexp** κλπ). Τρέξτε το πρόγραμμα και παρατηρήστε την αυξημένη ακρίβεια υπολογισμού του π και του $z = (2 + i)e^{3-i}$ χρησιμοποιώντας μεταβλητές διπλής ακρίβειας.

Ένα άλλο σημαντικό στοιχείο της γλώσσας που παραλείψαμε στην προηγούμενη παράγραφο είναι η κοινή χρήση μεταβλητών από διαφορετικά μέρη του προγράμματος. Μία μεταβλητή που ορίζεται σε ένα υποπρόγραμμα (**main program**, **subroutine**, **function**) είναι τοπική και διαφορετικά υποπρογράμματα δεν μπορούν να έχουν πρόσβαση σε αυτή. Για να αποκτήσουμε πρόσβαση σε κοινό σημείο της μνήμης όπου αποθηκεύουμε τις τιμές των μεταβλητών χρησιμοποιούμε την εντολή **COMMON**. Δείτε το παρακάτω παράδειγμα:

```
C -----
      program f77_common
      implicit none
      real k1,k2,k3
      common /CONSTANTS/k1,k2

      k1=1.0
      k2=1.0
      k3=1.0
      print *, 'main: k1= ',k1,' k2= ',k2,' k3= ',k3
      call s1 !prints k1 and k2 but not k3
      call s2 !changes the value of k2 but not k3
      print *, 'main: k1= ',k1,' k2= ',k2,' k3= ',k3

      end
C -----
```

```

subroutine s1()
implicit none
real k1,k2,k3
common /CONSTANTS/k1,k2

print *, 's1: k1= ',k1,' k2= ',k2,' k3= ',k3
end

C -----
subroutine s2()
implicit none
real k1,k2,k3
common /CONSTANTS/k1,k2

k2 = 2.0
k3 = 2.0
end

```

Το **COMMON** block εδώ έχει το όνομα **CONSTANTS** και μπορούμε να αναφερόμαστε σε αυτό από οποιαδήποτε υπορουτίνα ή συνάρτηση του προγράμματος. Στην πραγματικότητα δείχνει σε ένα συγκεκριμένο σημείο της μνήμης και εδώ δεσμεύουμε το χώρο για δύο μεταβλητές τύπου **REAL** τις **k1**, **k2**. Οι μεταβλητές αυτές διαβάζονται και αλλάζουν τιμές από τις υπορουτίνες **s1** και **s2** ενώ η **k3** παρόλο που έχει κοινό όνομα και στο κύριο πρόγραμμα και στις υπορουτίνες, αναφέρεται σε διαφορετικές μεταβλητές κάθε φορά. Το πρόγραμμα τυπώνει:

```

main: k1=    1.000000      k2=    1.000000      k3=    1.000000
s1:   k1=    1.000000      k2=    1.000000      k3=   -2.8117745E-05
main: k1=    1.000000      k2=    2.000000      k3=    1.000000

```

Ένα από τα αδύναμα σημεία της Fortran είναι η περιορισμένη δυνατότητα να χειριστούμε εύελικτα το Input/Output (I/O). Για το λόγο αυτό θα χρησιμοποιήσουμε άλλα προγράμματα όπως **awk**, **perl** ή προγράμματα στη γλώσσα **C**. Ακόμα όμως και η Fortran έχει χειρισμό του I/O αλλά όντας επιστημονικά προσανατολισμένη αυτός αφορά κυρίως την ακρίβεια παρουσίασης των αριθμών. Αν έχετε να χειρισθείτε κείμενο καλύτερα διαλέξτε μια άλλη γλώσσα προγραμματισμού... Μέχρι στιγμής οι μόνες εντολές που χρησιμοποιήσαμε για I/O είναι οι προκαθορισμένες χρησιμοποιώντας **print ***, **write(,*)**, **read(,*)**. Αλλά το ***** μπορεί να αντικατασταθεί με εντολές φορμά σύμφωνα με το παρακάτω παράδειγμα:


```

program f77_format
implicit none
integer i
real    x, a(10)
real*8  x8

i  = 123456
x  = 2.0 *atan2(1.0,0.0)
print '(A5,I6,F12.7)', 'x,i= ',i,x
x8 = 2.0D0*atan2(1.0D0,0.0D0)
write(6,'(F18.16,E24.17,G24.17,G24.17)') x8,x8,
*      1.0D15*x8,1.0D18*x8
write(6,'(3F20.16)') x8,x8/2.0,cos(x8)
write(6,'(200F12.6)')(a(i), i=1,10)
end

```

Προσέξτε τις παρενθέσεις μέσα στα εισαγωγικά: (A5,I6,F12.7) είναι εντολή φορμά για την εντολή `print` και δίνει οδηγίες για την εκτύπωση τριών μεταβλητών: A είναι για CHARACTER, I για INTEGER και F για REAL. Οι αριθμοί αμέσως μετά το γράμμα υποδηλώνουν των αριθμό των χαρακτήρων που θα χρησιμοποιηθούν για την εκτύπωση. Προσοχή! Αν δεν είναι αρκετές οι θέσεις εκτύπωσης η Fortran θα αρνηθεί να κάνει την εκτύπωση και θα τυπώσει μια σειρά από *, τα αστεράκια του τρόμου⁴⁹. Και στις θέσεις αυτές πρέπει να συνυπολογίσετε των αριθμό των δεκαδικών ψηφίων, την υποδιαστολή, το πρόσημο, τα ψηφία και το πρόσημο του εκθέτη... Μην είστε τσιγκούνηδες λοιπόν, δώστε άπλετο χώρο και μπορεί να σας χρειαστεί... Εδώ A5 υποδηλώνει CHARACTER που θα τυπωθεί σε 5 θέσεις χαρακτήρων, I6 INTEGER 6 θέσεων και F12 REAL 12 χαρακτήρων. Μετά την υποδιαστολή στο F12.7 υποδηλώνουμε πόσα δεκαδικά ψηφία θέλουμε να τυπωθούν.

Στην εντολή φορμά (F18.16,E24.17,G24.17,G24.17) δίνουμε οδηγίες για την εκτύπωση μιας μεταβλητής διπλής ακρίβειας. Στην καλύτερη περίπτωση έχουμε περίπου 16 δεκαδικά ψηφία ακρίβεια οπότε δεν έχει νόημα να κρατάμε παραπάνω (σε ένα υπολογισμό συνήθως χάνουμε ακρίβεια). Με την εντολή F θέλει προσοχή: Αν χρειαστεί εκθέτης για την αναπαράσταση, ο αριθμός δε θα τυπωθεί και συνήθως το αποφεύγουμε εκτός αν είμαστε σίγουροι ότι δε χρειάζεται εκθέτης. Η επιλογή E γράφει τον αριθμό πάντα σε επιστημονική μορφή με

⁴⁹ Σχεφτείτε μετά από ένα επίπονο υπολογισμό να πάτε να δείτε τα πολυπόθητα αποτελέσματα μόνο για να ανακαλύψετε ότι κάνατε λάθος στην εντολή του φορμά...

εκθέτη. Η επιλογή G γράφει τον αριθμό χωρίς εκθέτη αν δε χρειάζεται και με εκθέτη αν χρειάζεται. Οι αριθμοί έχουν την ίδια έννοια όπως και πριν. Στην εντολή φορμά (3F20.16) δείχνουμε πώς δίνουμε ένα πολλαπλασιαστικό παράγοντα 3 στην εκτύπωση των REAL*8. Και στην τελευταία δείχνουμε πώς να τυπώνουμε ένα μεγάλο διάνυσμα σε μία γραμμή: `write(6,'(200F12.6)')(a(i), i=1,10)`. Ο πολλαπλασιαστικός παράγοντας μπορεί να είναι μεγαλύτερος από αυτόν που θα χρησιμοποιήσουμε. Το πρόγραμμα τυπώνει (τη δεύτερη γραμμή τη διπλώσαμε για να φαίνεται):

```
x,i= 123456    3.1415927
3.1415926535897931  0.31415926535897931E+01  3141592653589793.0
                                     0.31415926535897933E+19
3.1415926535897931  1.5707963267948966 -1.0000000000000000
0.000000    0.000000    0.000000  ....
```

Οι εντολές φορμά μπορούν να μοιράζονται με ... παραπομπές. Αν βάλουμε έναν αριθμό 1-99999 από τη 1η έως και την 5η στήλη αυτός είναι μία 'ετικέτα' στην εντολή που ακολουθεί (labeled statement). Αν η εντολή αυτή είναι εντολή **FORMAT** τότε μπορούμε να αναφερθούμε σε αυτή με τον αριθμό της από τις εντολές **PRINT**, **WRITE** και **READ**. Έτσι πολλές εντολές I/O μπορούν να χρησιμοποιούν την ίδια εντολή **FORMAT** αν τυπώνουν με τον ίδιο τρόπο. Το παρακάτω πρόγραμμα κάνει ακριβώς ό,τι και το παραπάνω με τη μόνη διαφορά ότι χρησιμοποιούμε labeled statements και εντολές **FORMAT**:

```
program f77_format
implicit none
integer i
real    x, a(10)
real*8  x8

i  = 123456
x  = 2.0 *atan2(1.0,0.0)
print 100,'x,i= ',i,x
x8 = 2.0D0*atan2(1.0D0,0.0D0)
write(6,123) x8,x8,
*      1.0D15*x8,1.0D18*x8
write(6,4444) x8,x8/2.0,cos(x8)
write(6,9999)(a(i), i=1,10)
100  FORMAT(A5,I6,F12.7)
```

```
123  FORMAT(F18.16,E24.17,G24.17,G24.17)
4444 FORMAT(3F20.16)
9999 FORMAT(200F12.6)
      end
```

Τέλος ο/η αναγνώστης/τρια θα πρέπει να μελετήσει τις διαθέσιμες συναρτήσεις της Fortran 77 (intrinsic functions) που δίνονται στον Πίνακα 1.2 της σελίδας 67.

1.5 Κοιτάζοντας τα Αποτελέσματα

Η γραφική απεικόνιση των δεδομένων είναι αναπόσπαστο μέρος της ποιοτικής αλλά και ποσοτικής κατανόησης της πληροφορίας που περιέχουν. Ένα καλό και ελεύθερα διαθέσιμο πρόγραμμα που παράγει γραφήματα υψηλής ποιότητας στις δύο και τρεις διαστάσεις είναι το `gnuplot`. Τα ειδικότερα πλεονεκτήματά του έναντι άλλων εφαρμογών είναι η ευελιξία στη χρήση του από τη γραμμή εντολών αλλά και μέσα από άλλα προγράμματα και οι μεγάλες δυνατότητες στο χειρισμό και μετασχηματισμό των δεδομένων. Έχει τη δικιά του στοιχειώδη γλώσσα προγραμματισμού του και, όπου αυτή δε φτάνει, πολύπλοκες διαδικασίες μπορούν να γίνουν χρησιμοποιώντας άλλες εφαρμογές. Ο χρήστης έχει απευθείας πρόσβαση σε πολλές μαθηματικές συναρτήσεις και σε συνάρτηση προσαρμογής των δεδομένων (fitting). Διαθέτει διαδραστικά τερματικά όπου με το ποντίκι ο χρήστης μπορεί να μετασχηματίζει τα γραφήματα. Η παράγραφος αυτή είναι εξαιρετικά συνοπτική παρουσιάζοντας τα εργαλεία που είναι απολύτως απαραίτητα για τα παρακάτω κεφάλαια. Για περισσότερες πληροφορίες παραπέμπουμε στην ιστοσελίδα του `gnuplot` <http://gnuplot.info/> και ειδικότερα στη σελίδα με την Demo Gallery <http://gnuplot.sourceforge.net/demo/> όπου θα βρείτε αμέσως πώς γίνεται η εργασία που σας ενδιαφέρει.

Για να ξεκινήσετε το `gnuplot` δίνετε την εντολή όπως φαίνεται παρακάτω:

```
> gnuplot
```

```
G N U P L O T
```

```
Version 4.2 patchlevel 2
```

```
last modified 31 Aug 2007
```

```
System: Linux 2.6.24-16-generic
```

```
Copyright (C) 1986 - 1993, 1998, 2004, 2007
```

Thomas Williams, Colin Kelley and many others

Type 'help' to access the on-line reference manual.

The gnuplot FAQ is available from <http://www.gnuplot.info/faq/>

Send bug reports and suggestions to

<<http://sourceforge.net/projects/gnuplot>>

Terminal type set to 'wxt'

gnuplot>

Παραπάνω δείχνεται το μήνυμα καλωσορίσματος και στην τελευταία γραμμή φαίνεται το prompt του προγράμματος από το οποίο περιμένει να του πληκτρολογήσουμε μία εντολή και να τη δώσουμε πατώντας το **Enter**. Στη συνέχεια όταν θα γράφουμε το prompt αυτό θα υπονοούμε πως το ακολουθούν εντολές που δίνονται στο gnuplot.

Το γράφημα μιας συνάρτησης δίνεται απλά με την εντολή `plot`. Το σύμβολο `x` εννοείται πως είναι η ανεξάρτητη μεταβλητή⁵⁰. Έτσι η εντολή

gnuplot> plot x

κάνει τη γραφική παράσταση της $y = f(x) = x$ (ευθεία κλίσης 1). Για να κάνουμε ταυτόχρονα τις γραφικές παραστάσεις περισσότερων συναρτήσεων απλά τις γράφουμε μαζί ως εξής:

gnuplot> plot [-5:5] [-2:4] x, x**2, sin(x), besj0(x)

Παραπάνω γίνεται το γράφημα των συναρτήσεων x , x^2 , $\sin x$, $J_0(x)$. Στις αγκύλες `[:]` βάζουμε τα όρια της γραφικής παράστασης στον άξονα x και y αντίστοιχα. Το `[-5:5]` καθορίζει το x να μεταβάλλεται από -5 έως $+5$, ενώ το `[-2:4]` καθορίζει το y να μεταβάλλεται από -2 έως $+4$. Αν σε κάποιες θέσεις δε βάλουμε αριθμό, τότε το gnuplot βάζει τα όρια αυτόματα: `[1:] [:5]` καθορίζει το κάτω όριο το x να είναι το 1 και το άνω όριο στο y να είναι το 5, ενώ τα απροσδιόριστα άνω και κάτω όρια αφήνονται στα ... χέρια του gnuplot.

Συχνά θα θέλουμε να κάνουμε τη γραφική παράσταση δεδομένων που δίνονται από διακριτά ζεύγη (x_i, y_i) . Τα δεδομένα αυτά τα τοποθετούμε σε αρχεία σε στήλες. Ας υποθέσουμε πως το αρχείο με τα δεδομένα μας ονομάζεται `data` και τα περιεχόμενά του είναι:

⁵⁰ Αλλάζει με την εντολή `set dummy t` για να γίνει λ.χ. `t` η ανεξάρτητη μεταβλητή.

```
# x  y1  y2
0.5 1.0 0.779
1.0 2.0 0.607
1.5 3.0 0.472
2.0 4.0 0.368
2.5 5.0 0.287
3.0 6.0 0.223
```

Η πρώτη γραμμή αρχίζει με το χαρακτήρα # και το gnuplot την αγνοεί (σχόλια για μας). Για να κάνουμε τη γραφική παράσταση της 2ης στήλης συναρτήσει της 1ης δίνουμε απλά την εντολή:

```
gnuplot> plot "data" using 1:2 with points
```

Το όνομα του αρχείου `data` δίνεται ανάμεσα σε εισαγωγικά ενώ μετά την εντολή `using` δίνουμε τις στήλες που θα αντιστοιχούν στον άξονα x και y αντίστοιχα (1:2= στήλη 1 τα x_i και στήλη 2 τα y_i). Η εντολή `with points` αναπαριστά τα ζεύγη (x_i, y_i) με σημεία.

Η εντολή

```
gnuplot> plot "data" using 1:3 with lines
```

κάνει τη γραφική παράσταση της 3ης στήλης συναρτήσει της 1ης και τα ζεύγη (x_i, y_i) ενώνονται με ευθύγραμμα τμήματα.

Οι γραφικές παραστάσεις μπορούν να συνδυαστούν:

```
gnuplot> plot "data" using 1:3 with points, exp(-0.5*x)
```

```
gnuplot> replot "data" using 1:2
```

```
gnuplot> replot 2*x
```

Στην πρώτη γραμμή κάνουμε μαζί τη γραφική παράσταση της 1ης και 3ης στήλης του αρχείου `data` μαζί με τη συνάρτηση $e^{-x/2}$. Στη δεύτερη γραμμή προσθέτουμε με την εντολή `replot` στην ίδια γραφική παράσταση τα σημεία της 1ης και 3ης στήλης. Και στην 3η βάζουμε μαζί και τη γραφική παράσταση της συνάρτησης $2x$.

Η εντολή `using` έχει πολλές δυνατότητες. Αν αντί για αριθμούς βάλουμε μαθηματικές εκφράσεις ανάμεσα σε παρενθέσεις (δηλ `using (...):(...)`) τότε το gnuplot τις υπολογίζει για κάθε σημείο και βάζει τα αποτελέσματα στη γραφική παράσταση. Για να μπει η τιμή μιας στήλης στη μαθηματική έκφραση χρησιμοποιούμε το ίδιο συντακτικό με την `awk`, δηλ $\$i$ αναφέρεται στη στήλη $i=1,2,3,\dots$. Παραδείγματα:

```
gnuplot> plot "data" using 1:($2*sin($1)*$3) with points
gnuplot> replot 2*x*sin(x)*exp(-x/2)
```

Κάνει τη γραφική παράσταση της 1ης στήλης με την αντίστοιχη τιμή της έκφρασης $y_i \sin(x_i) z_i$, όπου x_i, y_i, z_i οι τιμές της 1ης, 2ης και 3ης στήλης αντίστοιχα. Η δεύτερη γραμμή τοποθετεί στο σχήμα και τη γραφική παράσταση της συνάρτησης $2x \sin(x) e^{-x/2}$.

```
gnuplot> plot "data" using (log($1)):(log($2**2))
gnuplot> replot 2*x+log(4)
```

Κάνει τη γραφική παράσταση του φυσικού λογάριθμου της 1ης στήλης με το φυσικό λογάριθμο του τετραγώνου της 2ης.

Από το gnuplot μπορούμε να κάνουμε τη γραφική παράσταση των δεδομένων που τυπώνει στο `stdout` οποιοδήποτε πρόγραμμα εκτελείται από το φλοιό. Έστω ότι έχουμε ένα πρόγραμμα με όνομα `area` που τυπώνει στο `stdout` την ακτίνα και το εμβαδόν ενός κύκλου:

```
> ./area
R=      3.280000      area=      33.79851
R=      6.280000      area=     123.8994
R=      5.280000      area=      87.58257
R=      4.280000      area=      57.54895
```

Τα δεδομένα είναι στην 2η και 4η στήλη του `stdout` και μπορούμε να τα δούμε γραφικά από το gnuplot με την εντολή:

```
gnuplot> plot "< ./area" using 2:4
```

Δηλαδή στη θέση του ονόματος του αρχείου βάζουμε το όνομα της εντολής με το χαρακτήρα `<` να προηγείται. Μπορούμε να συνδυάσουμε εντολές μέσω `pipe` και να παράγουμε πολύπλοκα αποτελέσματα. Λ.χ.

```
gnuplot> plot "< ./area|sort -g -k 2|awk '{print log($2),log($4)}'" \
            using 1:2
```

όπου τα δεδομένα που αναπαρίστανται γραφικά είναι το αποτέλεσμα ενός φίλτρου τριών εντολών: Αυτή που παράγει τα δεδομένα ακτίνα-εμβαδόν όπως παραπάνω, η δεύτερη `sort` που τα διατάσσει ανάλογα με την αριθμητική τιμή της 2ης στήλης και η τρίτη `awk` που τυπώνει το λογάριθμο της 2ης στήλης και το

λογάριθμο της 4ης. Παρατηρήστε πώς τώρα χρησιμοποιούμε την εντολή `using 1:2` αφού η τελευταία εντολή τυπώνει τα δεδομένα σε δύο μόνο στήλες.

Για να σώσουμε τις γραφικές μας παραστάσεις σε αρχεία που μπορούμε να φυλάξουμε και πιθανώς να δημοσιεύσουμε, πρέπει να αλλάξουμε το `'terminal'` που χρησιμοποιεί ο `gnuplot` σε ένα οδηγό που μεταφράζει τη γραφική παράσταση σε μία γλώσσα που καταλαβαίνουν άλλα προγράμματα που δείχνουν εικόνες (λ.χ. PDF, postscript, jpeg, png, gif κλπ). Κατευθύνοντας την 'έξοδο' του `terminal` σε ένα αρχείο πετυχαίνουμε το ζητούμενο. Για παράδειγμα

```
gnuplot> plot "data" using 1:3
gnuplot> set terminal jpeg
gnuplot> set output "data.jpg"
gnuplot> replot
gnuplot> set output
gnuplot> set terminal wxt
```

Η πρώτη γραμμή κάνει τη γραφική παράσταση στο τερματικό ώστε να τη δούμε. Η δεύτερη καθορίζει πως το γράφημα θα σωθεί σε μορφή JPEG και η τρίτη το όνομα του αρχείου που θα το αποθηκεύσουμε. Στην τέταρτη επαναλαμβάνουμε το τελευταίο γράφημα (εδώ αυτό της 1ης γραμμής) και στην πέμπτη κλείνουμε το αρχείο `data.jpg` (μην το ξεχάσετε!). Η τελευταία γραμμή επιβάλλει η επόμενη γραφική παράσταση να γίνει πάλι στο τερματικό.

Συνήθως γραφικές παραστάσεις υψηλής ποιότητας αποθηκεύονται στη γλώσσα Postscript. Επιλέξτε `set terminal postscript` και `set output "data.ps"` στην περίπτωση αυτή.

Λίγα λόγια για τις τρισδιάστατες γραφικές παραστάσεις. Οι επόμενες εντολές δείχνουν πώς με την εντολή `splot` μπορείτε να δείτε τη γραφική παράσταση της συνάρτησης $f(x, y) = e^{-x^2-y^2}$. Με το ποντίκι μπορείτε να την περιστρέψετε και να τη δείτε υπό διαφορετική γωνία.

```
gnuplot> set pm3d
gnuplot> set hidden3d
gnuplot> set size ratio 1
gnuplot> set isosamples 50
gnuplot> splot [-2:2][-2:2] exp(-x**2-y**2)
```

Αν έχετε δεδομένα στη μορφή (x_i, y_i, z_i) και θέλετε να τα αναπαραστήσετε γραφικά στη μορφή $z_i = f(x_i, y_i)$ τακτοποιήστε τα σε ένα αρχείο της μορφής

```

-1 -1 2.000
-1  0 1.000
-1  1 2.000

0 -1 1.000
0  0 0.000
0  1 1.000

1 -1 2.000
1  0 1.000
1  1 2.000

```

Προσέξτε πώς βάζουμε μία κενή γραμμή κάθε φορά που αλλάζει η τιμή του x . Αν ονομάσετε το αρχείο αυτό `data3`, δείτε τη γραφική παράσταση με τις εντολές:

```

gnuplot> set pm3d
gnuplot> set hidden3d
gnuplot> set size ratio 1
gnuplot> splot "data3" with lines

```

Κλείνουμε με δύο λόγια για τη γραφική παράσταση που δίνεται από παραμετρικές εξισώσεις. Στις δύο διαστάσεις θεωρούμε τις καμπύλες $(x(t), y(t))$ και στις τρεις επιφάνειες $(x(u, v), y(u, v), z(u, v))$. Με τις παρακάτω εντολές κάνουμε τη γραφική παράσταση του κύκλου $(\sin t, \cos t)$ και της σφαίρας $(\cos u \cos v, \cos u \sin v, \sin u)$:

```

gnuplot> set parametric
gnuplot> plot sin(t),cos(t)
gnuplot> splot cos(u)*cos(v),cos(u)*sin(v),sin(u)

```

1.6 Turbo: Σενάρια Φλοιοού

Η γλώσσα Fortran θα φανεί σε κάποιον που έχει συνηθίσει κάποια γλώσσα με περισσότερες δυνατότητες (C, C++, Java, ...) πως είναι δύσχρηστη όταν κάποιος θέλει να κάνει πολύπλοκες διαδικασίες που αφορούν το σύστημα και δε θα είχε άδικο. Όταν όμως χρησιμοποιήσει τα προγράμματα που γράφει σε Fortran σε συνδυασμό με τα πανίσχυρα εργαλεία που του παρέχει το λειτουργικό

σύστημα τα προβλήματα αυτά ξεπερνιόνται και έτσι μπορεί κανείς να χρησιμοποιήσει τα πλεονεκτήματα της γλώσσας χωρίς να ανησυχεί για διαχειριστικές και συχνά τετριμμένες εργασίες.

Για να αποφύγει κανείς την επαναλαμβανόμενη διαδικασία εκτέλεσης των ίδιων εντολών (που εμπεριέχει και τον κίνδυνο σφάλματος), μπορεί τις εντολές που θέλει να δώσει να τις κωδικοποιήσει μέσα σε ένα αρχείο. Αυτό ονομάζεται σενάριο φλοιού (shell script) και την πιο απλή μορφή του μπορεί να είναι απλά μια σειρά από εντολές. Γράφουμε στο αρχείο `script01.csh`:

```
#!/bin/tcsh -f
f77 area_01.f -o area
./area
f77 area_02.f -o area
./area
f77 area_03.f -o area
./area
f77 area_04.f -o area
./area
```

Η πρώτη γραμμή (ακριβώς!!) αρχίζει με `#!/bin/tcsh -f` που ερμηνεύεται από το λειτουργικό σύστημα ώστε να εκτελεστούν οι εντολές από το φλοιό `/bin/tcsh`. Στη συνέχεια απλά γράφουμε τις εντολές μεταγλώττισης και εκτέλεσης των προγραμμάτων που μελετήσαμε στην προηγούμενη παράγραφο. Για να τρέχουμε τις εντολές στο αρχείο αυτό δίνουμε τις εντολές:

```
> chmod u+x script01.csh
> ./script.csh
```

Την πρώτη εντολή τη δίνουμε μόνο την πρώτη φορά που γράφουμε το αρχείο - ο έτσι ώστε να δώσουμε άδεια πρόσβασης εκτελέσιμου αρχείου στο χρήστη. Όλα ωραία, εκτός του ότι πρέπει να δίνουμε τις 10 ακτίνες του κύκλου στα προγράμματα που τις ζητούν. Μια λύση είναι να γράψουμε τα δεδομένα σε ένα αρχείο `Input` και να δίνουμε την εντολή

```
./area < Input
```

οπότε δεν χρειάζεται να παρέχουμε τα δεδομένα διαδραστικά. Υπάρχει και πιο συμπαγής λύση, να βάλουμε τα περιεχόμενα των δεδομένων σε ένα 'Here Document', ένα αρχείο το οποίο ο χρήστης μπορεί να φανταστεί ότι υπάρχει αλλά δεν χρειάζεται να ανησυχεί για τη διαχείρισή του. Το συντακτικό, λίγο

στρυφνό για αρχή αλλά συνηθίζεται (και γίνεται και εθισμός...) είναι ως εξής (στο αρχείο `script02.csh`):

```
#!/bin/tcsh -f
f77 area_04.f -o area
./area <<EOF
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
EOF
```

δηλ. το πρόγραμμα `./area` παίρνει `stdin` από τα περιεχόμενα μεταξύ των γραμμών⁵¹:

```
./area <<EOF
...
EOF
```

Η δύναμη του shell scripting είναι οι ικανότητες προγραμματισμού που παρέχει: Ορισμός μεταβλητών, loops, conditionals, ... Οι μεταβλητές ορίζονται όπως οι μεταβλητές φλοιού που αναφέραμε στην παράγραφο 1.1.2. Η τιμή μιας μεταβλητής `name` είναι `$name` και μπορούμε να τη θέσουμε με την εντολή `set name = value`. Ένα array μπορεί να οριστεί με την εντολή

```
set R = (1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0)
```

και η πρόσβαση στα δεδομένα γίνεται με το συντακτικό `$R[1] ... $R[10]`

Ας δούμε τώρα ένα πιο ... προχωρημένο σενάριο:

```
#!/bin/tcsh -f

set files = (area_01.f area_02.f area_03.f area_04.f)
```

⁵¹Το EOF μπορεί να είναι οποιαδήποτε ακολουθία χαρακτήρων.

```

set R      = (1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0)

echo "Hello $USER Today is " `date`
foreach file ($files)
  echo "# ----- Working on file $file "
  f77 $file -o area
  ./area <<EOF
$R[1]
$R[2]
$R[3]
$R[4]
$R[5]
$R[6]
$R[7]
$R[8]
$R[9]
$R[10]
EOF
  echo "# ----- Done "
  if( -f AREA.DAT ) cat AREA.DAT
end

```

Οι πρώτες γραμμές με τις εντολές `set` θέτουν τις τιμές των μεταβλητών `files` (4 τιμές) και `R` (10 τιμές). Η εντολή `echo` απλά ‘αντηχεί’ στο `stdout` το όρισμά της. Εδώ ο φλοιός αναπτύσσει στο όρισμα "Hello \$USER Today is " `date` την τιμή της μεταβλητής `USER` η οποία είναι μεταβλητή περιβάλλοντος που το λειτουργικό σύστημα θέτει να είναι το όνομα χρήστη. Στη συνέχεια στη το `date` αντικαθίσταται από το `stdout` της εντολής `date`, λ.χ. Thu May 24 22:01:40 EEST 2007.

Στη συνέχεια αρχίζει το `foreach` loop:

```

foreach file ($files)
  ...
end

```

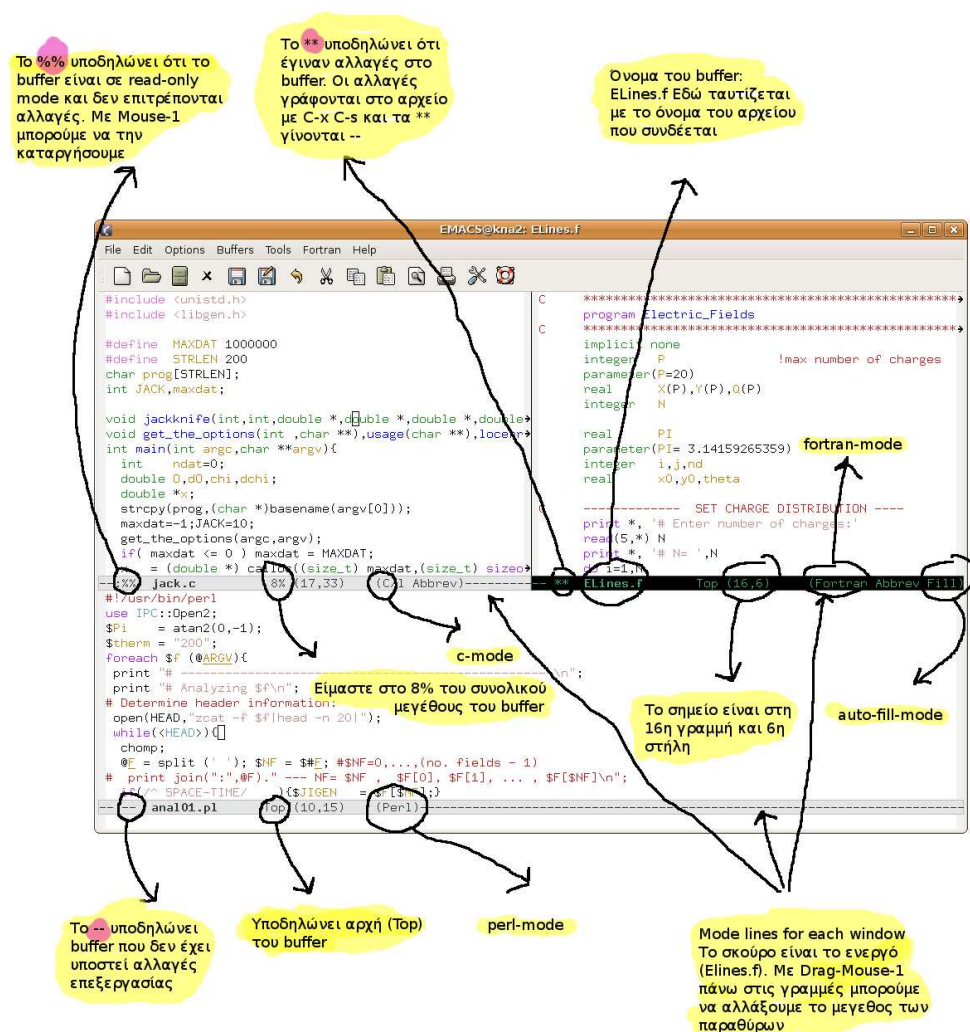
Η μεταβλητή `$files` αναπτύσσεται στις 4 τιμές της (τα ονόματα των αρχείων Fortran `area_01.f`, `area_02.f`, `area_03.f`, `area_04.f`) και ο βρόχος εκτελείται μια φορά για κάθε τιμή. Κάθε φορά η τιμή της μεταβλητής `file`

είναι η εκάστοτε τιμή της `files`. Άρα η εντολή `f77 $file -o area` θα μεταγλωττίσει κάθε φορά ένα από τα παραπάνω 4 αρχεία και στη συνέχεια θα εκτελέσει το εκάστοτε πρόγραμμα `./area`.

Η τελευταία γραμμή στο βρόχο

```
if( -f AREA.DAT ) cat AREA.DAT
```

είναι ένα if-conditional: Εκτελεί την εντολή `cat AREA.DAT` μόνο αν η συνθήκη `-f AREA.DAT` είναι αληθής, δηλ. το αρχείο `AREA.DAT` υπάρχει.



Σχήμα 1.5: Το παράθυρο του Emacs χωρίστηκε εδώ σε τρία παράθυρα. Ο χωρισμός έγινε πρώτα οριζόντια (C-x 2) και μετά κάθετα (C-x 3). Σέρνοντας το ποντίκι Drag-Mouse-1 πάνω στις οριζόντιες (mode lines) και κάθετη διαχωριστικές γραμμές μπορούμε να αλλάξουμε τα μεγέθη των παραθύρων. Τονίζουμε τις χρήσιμες πληροφορίες που βρίσκει κανείς στο mode line κάθε παράθυρου. Κάθε παράθυρο έχει το σημείο του (point) και ο δρομέας (cursor) βρίσκεται στο ενεργό παράθυρο (εδώ στο ELines.f). Παρατηρήστε πώς σημειώνεται το ανέπαφο buffer (--), το επεξεργασμένο (*) και αυτό που είναι σε read only mode (%). Με κλικ με Mouse-1 πάνω στα %% τα αλλάζουμε σε -- και το αντίστροφο. Με δεξί κλικ Mouse-3 πάνω στο όνομα της mode μπορούμε να ενεργοποιήσουμε επιλογή από minor modes. Με αριστερό κλικ Mouse-1 έχουμε πρόσβαση σε εντολές σχετικές με την mode.

<code>awk</code>	search for and process patterns in a file,
<code>cat</code>	display, or join, files
<code>cd</code>	change working directory
<code>chmod</code>	change the access mode of a file
<code>cp</code>	copy files
<code>date</code>	display current time and date
<code>df</code>	display the amount of available disk space
<code>diff</code>	display the differences between two files
<code>du</code>	display information on disk usage
<code>echo</code>	echo a text string to output
<code>find</code>	find files
<code>grep</code>	search for a pattern in files
<code>gzip</code>	compress files in the gzip (.gz) format (gunzip to uncompress)
<code>head</code>	display the first few lines of a file
<code>kill</code>	send a signal (like KILL) to a process
<code>locate</code>	search for files stored on the system (faster than find)
<code>less</code>	display a file one screen at a time
<code>ln</code>	create a link to a file
<code>lpr</code>	print files
<code>ls</code>	list information about files
<code>man</code>	search information about commands in man pages
<code>mkdir</code>	create a directory
<code>more</code>	display a file one screen at a time
<code>mv</code>	move and/or rename a file
<code>ps</code>	report information on the processes run on the system
<code>pwd</code>	print the working directory
<code>rm</code>	remove (delete) files
<code>rmdir</code>	remove (delete) a directory
<code>sort</code>	sort and/or merge files
<code>tee</code>	copy stdout to one or more files
<code>tail</code>	display the last few lines of a file
<code>tar</code>	store or retrieve files from an archive file
<code>top</code>	dynamic real-time view of processes
<code>wc</code>	counts lines, words and characters in a file
<code>whatis</code>	list man page entries for a command
<code>where</code>	show where a command is located in the path (alternatively: <code>whereis</code>)
<code>which</code>	locate an executable program using "path"
<code>who</code>	report who is logged in and what processes are running
<code>zip</code>	create compressed archive in the zip format (.zip)
<code>unzip</code>	get contents of zip archive

Πίνακας 1.1: Περίληψη βασικών εντολών στο Unix.

Πίνακας 1.2: Βασικές συναρτήσεις (intrinsic functions) της Fortran 77. Η δεύτερη και τρίτη στήλη του πίνακα αναφέρονται στον τύπο μεταβλητής που δέχεται/επιστρέφει η συνάρτηση αντίστοιχα. Το C αντιστοιχεί σε μεταβλητή τύπου CHARACTER, το D αντιστοιχεί σε μεταβλητή τύπου REAL*8, το I αντιστοιχεί σε μεταβλητή τύπου INTEGER, το L αντιστοιχεί σε μεταβλητή τύπου LOGICAL, το R αντιστοιχεί σε μεταβλητή τύπου REAL, το X αντιστοιχεί σε μεταβλητή τύπου COMPLEX. Αν είναι πολλά γράμματα μαζί σημαίνει ότι μπορεί να έχουμε περισσότερους από ένα τύπους μεταβλητής. Αν τα γράμματα χωρίζονται με κόμμα, τότε η συνάρτηση παίρνει περισσότερα του ενός ορίσματα. Έτσι λ.χ. η συνάρτηση ABS παίρνει σαν όρισμα REAL*8, INTEGER ή REAL (DIR), και το αποτέλεσμα είναι αντίστοιχα REAL*8, INTEGER, REAL (DIR). Η συνάρτηση CMPLX όταν παίρνει δύο ορίσματα τύπου REAL*8, INTEGER ή REAL (DIR,DIR), επιστρέφει στην έξοδο έναν COMPLEX (X). Όταν παίρνει ένα όρισμα τύπου REAL*8, INTEGER, REAL ή COMPLEX (DIRX), επιστρέφει στην έξοδο έναν COMPLEX (X). Από την ιστοσελίδα <http://www.obliquity.com/computer/fortran/intrinsic.html>.

Συνάρτηση	Όρισμα(τα)	Αποτέλεσμα	Περιγραφή
ABS	X	R	modulus of a complex number
ABS	DIR	DIR	absolute value of a number
ACOS	DR	DR	arccosine of a number
AIMAG	X	R	imaginary part of a complex number
AINTE	DR	DR	truncates fractional part but preserves data type
ANINT	DR	DR	rounds to nearest whole number but preserves data type
ASIN	DR	DR	arcsine of a number
ATAN	DR	DR	arctangent of a number
ATAN2	DR,DR	DR	arctangent of arg1 divided by arg2 resolved into the correct quadrant
CMPLX	DIRX	X	converts to the COMPLEX data type
CMPLX	DIR,DIR	X	converts to the COMPLEX data type arg1 + i arg2
CONJG	X	X	complex conjugate of a complex number
COS	DRX	DRX	cosine of an angle in radians
COSH	DR	DR	hyperbolic cosine
DBLE	DIRX	D	converts to the DOUBLE PRECISION data type

Συνεχίζεται στην επόμενη σελίδα...

Πίνακας 1.2: Συνέχεια...

Συνάρτηση	Όρισμα(τα)	Αποτέλεσμα	Περιγραφή
DIM	DIR, DIR	DIR	if $\text{arg1} > \text{arg2}$, then returns $\text{arg1} - \text{arg2}$; otherwise 0
DPROD	R, R	D	double precision product of two single precision numbers
EXP	DRX	DRX	exponential
INT	DIRX	I	converts to the INTEGER data type by truncation
LOG	DRX	DRX	natural logarithm
LOG10	DRX	DRX	common logarithm
MAX	DIR, DIR, . . .	DIR	maximum value of arguments
MIN	DIR, DIR, . . .	DIR	minimum value of arguments
MOD	DIR, DIR	DIR, DIR	arg1 modulo arg2
NINT	DR	I	converts to the INTEGER data type by rounding
REAL	X	R	real part of a complex number
REAL	DIR	R	converts to the REAL data type
SIGN	DIR, DIR	DIR	if $\text{arg2} < 0$, then returns $-\text{arg1}$; else $+\text{arg1}$
SIN	DRX	DRX	sine of an angle in radians
SINH	DR	DR	hyperbolic sine
SQRT	DRX	DRX	square root
TAN	DR	DR	tangent of an angle in radians
TANH	DR	DR	hyperbolic tangent

Πίνακας 1.3: Περίληψη βασικών εντολών στο Emacs.

Leaving Emacs	
suspend Emacs (or iconify it under X)	C-z
exit Emacs permanently	C-x C-c
Files	
read a file into Emacs	C-x C-f
save a file back to disk	C-x C-s
save all files	C-x s
insert contents of another file into this buffer	C-x i
replace this file with the file you really want	C-x C-v
write buffer to a specified file	C-x C-w
toggle read-only status of buffer	C-x C-q
Getting Help	
The help system is simple. Type C-h (or F1) and follow the directions. If you are a first-time user, type C-h t for a tutorial .	
remove help window	C-x 1
scroll help window	C-M-v
apropos: show commands matching a string	C-h a
describe the function a key runs	C-h k
describe a function	C-h f
get mode-specific information	C-h m
Error Recovery	
abort partially typed or executing command	C-g
recover files lost by a system crash	M-x recover-session
undo an unwanted change	C-x u , C-_ or C-/
restore a buffer to its original contents	M-x revert-buffer
redraw garbaged screen	C-l
Incremental Search	
search forward	C-s
search backward	C-r
regular expression search	C-M-s
exit incremental search	RET
abort current search	C-g
Use C-s or C-r again to repeat the search in either direction. If Emacs is still searching, C-g cancels only the part not matched.	

Συνεχίζεται στην επόμενη σελίδα...

Πίνακας 1.3: Συνέχεια...

Motion		
entity to move over	backward	forward
character	C-b	C-f
word	M-b	M-f
line	C-p	C-n
go to line beginning (or end)	C-a	C-e
sentence	M-a	M-e
paragraph	M-{	M-}
page	C-x [	C-x]
sexp	C-M-b	C-M-f
function	C-M-a	C-M-e
go to buffer beginning (or end)	M-<	M->
scroll to next screen	C-v	
scroll to previous screen	M-v	
scroll left	C-x <	
scroll right	C-x >	
scroll current line to center of screen	C-u C-l	
Killing and Deleting		
entity to kill	backward	forward
character (delete, not kill)	DEL	C-d
word	M-DEL	M-d
line (to end of)	M-O C-k	C-k
kill region	C-w	
copy region to kill ring	M-w	
yank back last thing killed	C-y	
replace last yank with previous kill	M-y	
Marking		
set mark here	C-@ or C-SPC	
exchange point and mark	C-x C-x	
mark paragraph	M-h	
mark page	C-x C-p	
mark entire buffer	C-x h	

Συνεχίζεται στην επόμενη σελίδα...

Πίνακας 1.3: Συνέχεια...

Query Replace		
interactively replace a text string	M-% <i>or</i> M-x query-replace	
using regular expressions	M-x query-replace-regexp	
Valid responses in query-replace mode are:		
replace this one, go on to next	SPC	
replace this one, don't move	,	
skip to next without replacing	DEL	
replace all remaining matches	!	
exit query-replace	RET	
Buffers		
select another buffer	C-x b	
list all buffers	C-x C-b	
kill a buffer	C-x k	
Multiple Windows		
When two commands are shown, the second is a similar command for a frame instead of a window.		
delete all other windows	C-x 1	C-x 5 1
split window, above and below	C-x 2	C-x 5 2
delete this window	C-x 0	C-x 5 0
split window, side by side	C-x 3	
scroll other window	C-M-v	
switch cursor to another window	C-x o	C-x 5 o
select buffer in other window	C-x 4 b	C-x 5 b
display buffer in other window	C-x 4 C-o	C-x 5 C-o
find file in other window	C-x 4 f	C-x 5 f
find file read-only in other window	C-x 4 r	C-x 5 r
run Dired in other window	C-x 4 d	C-x 5 d
find tag in other window	C-x 4 .	C-x 5 .
grow window taller	C-x ^	
shrink window narrower	C-x {	
grow window wider	C-x }	
Formatting		
indent current line (mode-dependent)	TAB	

Συνεχίζεται στην επόμενη σελίδα...

Πίνακας 1.3: Συνέχεια...

insert newline after point	C-o
fill paragraph	M-q
set fill column	C-x f
Case Change	
uppercase word	M-u
lowercase word	M-l
capitalize word	M-c
uppercase region	C-x C-u
lowercase region	C-x C-l
The Minibuffer	
The following keys are defined in the minibuffer.	
complete as much as possible	TAB
complete up to one word	SPC
complete and execute	RET
show possible completions	?
fetch previous minibuffer input	M-p
fetch later minibuffer input or default	M-n
abort command	C-g
Type C-x ESC ESC to edit and repeat the last command that used the minibuffer. Type F10 to activate menu bar items on text terminals.	
Spelling Check	
check spelling of current word	M-\$
check spelling of all words in region	M-x ispell-region
check spelling of entire buffer	M-x ispell-buffer
Shells	
execute a shell command	M-!
run a shell command on the region	M-
filter region through a shell command	C-u M-
start a shell in window *shell*	M-x shell
Info – Getting Help Within Emacs	
enter the Info documentation reader	C-h i
find specified function or variable in Info	C-h S
Moving within a node:	

Συνεχίζεται στην επόμενη σελίδα...

Πίνακας 1.3: Συνέχεια...

scroll forward	SPC
scroll reverse	DEL
Moving between nodes:	
next node	n
previous node	p
move up	u
select menu item by name	m
follow cross reference (return with 1)	f
return to last node you saw	l
return to directory node	d
go to top node of Info file	t
go to any node by name	g
Other:	
run Info tutorial	h
look up a subject in the indices	i
search nodes for regexp	s
quit Info	q

Κεφάλαιο 2

Περιγραφή της Κίνησης

Στο κεφάλαιο αυτό θα δείξουμε πώς να προγραμματίσουμε απλές εξισώσεις τροχιάς ενός σωματιδίου και πώς να κάνουμε βασική ανάλυση των αριθμητικών αποτελεσμάτων. Χρησιμοποιούμε απλές μεθόδους απεικόνισης των τροχιών. Στην παράγραφο 2.3 μελετάμε την επίδραση των συστηματικών σφαλμάτων που υπεισέρχονται σε απλά αριθμητικά πρότυπα της κίνησης σωματιδίου που σκεδάζεται πάνω σε σκληρά και αμετακίνητα εμπόδια.

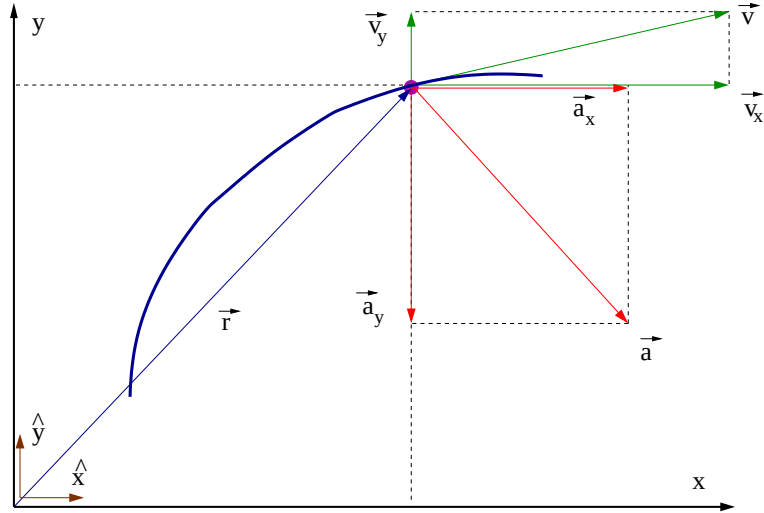
2.1 Κίνηση στο Επίπεδο

Σωματίδιο κινείται στο επίπεδο και η θέση του περιγράφεται σε ένα καρτεσιανό σύστημα συντεταγμένων από τις συντεταγμένες $(x(t), y(t))$ η οποία σε συνάρτηση του χρόνου δίνει την εξίσωση της τροχιάς του σωματιδίου. Το διάνυσμα θέσης του σωματιδίου είναι το $\vec{r}(t) = x(t) \hat{x} + y(t) \hat{y}$, όπου $\hat{x}$ και $\hat{y}$ είναι τα μοναδιαία διανύσματα στους άξονες x και y αντίστοιχα. Το διάνυσμα της ταχύτητας είναι το $\vec{v}(t) = v_x(t) \hat{x} + v_y(t) \hat{y}$ όπου

$$\begin{aligned} \vec{v}(t) &= \frac{d\vec{r}(t)}{dt} \\ v_x(t) &= \frac{dx(t)}{dt} \quad v_y(t) = \frac{dy(t)}{dt}, \end{aligned} \quad (2.1)$$

Η επιτάχυνση $\vec{a}(t) = a_x(t) \hat{x} + a_y(t) \hat{y}$ δίνεται από τις σχέσεις

$$\begin{aligned} \vec{a}(t) &= \frac{d\vec{v}(t)}{dt} = \frac{d^2\vec{r}(t)}{dt^2} \\ a_x(t) &= \frac{dv_x(t)}{dt} = \frac{d^2x(t)}{dt^2} \quad a_y(t) = \frac{dv_y(t)}{dt} = \frac{d^2y(t)}{dt^2}. \end{aligned} \quad (2.2)$$

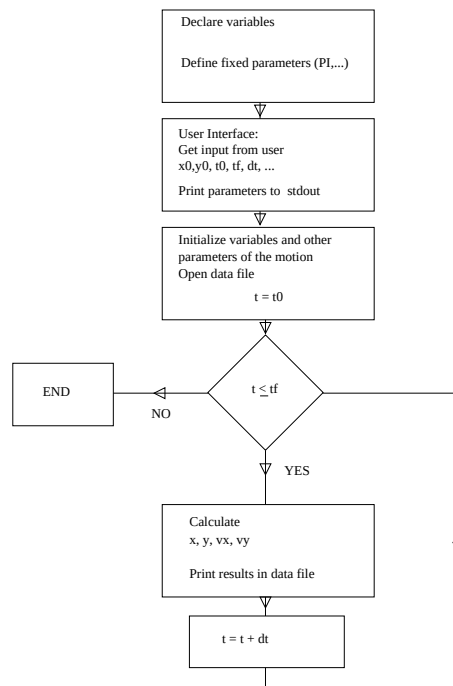


Σχήμα 2.1: Τροχιά κινητού υλικού σημείου στο επίπεδο. Φαίνονται τα διανύσματα θέσης $\vec{r}$, ταχύτητας $\vec{v}$ και επιτάχυνσης $\vec{a}$ και οι αντίστοιχες καρτεσιανές συντεταγμένες στο επιλεγμένο σύστημα αναφοράς σε ένα σημείο της τροχιάς.

Στην παράγραφο αυτή θα υποθέσουμε ότι μας δίνονται οι συναρτήσεις $(x(t), y(t))$. Από αυτές, παραγωγίζοντας σύμφωνα με τις παραπάνω σχέσεις, μπορούμε να πάρουμε την ταχύτητα και την επιτάχυνση. Σκοπός μας είναι να γράφουμε απλά προγράμματα τα οποία θα υπολογίζουν τις τιμές των συναρτήσεων αυτών σε ένα χρονικό διάστημα $[t_0, t_f]$ όπου t_0 η αρχική χρονική στιγμή και t_f η τελική. Η συνεχείς συναρτήσεις $x(t), y(t), v_x(t), v_y(t)$ θα προσεγγίζονται από μια διακριτή ακολουθία τιμών των συναρτήσεων στις χρονικές στιγμές $t_0, t_0 + \delta t, t_0 + 2\delta t, t_0 + 3\delta t, \dots$ έτσι ώστε όλες οι τιμές $t_0 + n\delta t \leq t_f$ ¹.

Αρχίζουμε σχεδιάζοντας το πρότυπο (template) ενός προγράμματος που θα κάνει τον παραπάνω υπολογισμό. Κάνοντας αυτό το σχεδιασμό προσεκτικά, το μόνο που θα μας απασχολεί κάθε φορά που θέλουμε να μελετήσουμε την κίνηση ενός άλλου κινητού θα είναι η ειδική εφαρμογή των εξισώσεων κίνησης. Στο Σχήμα 2.2 φαίνεται το λογικό διάγραμμα των βασικών λειτουργιών του προγράμματος. Αυτό μας βοηθάει να έχουμε μία εποπτική εικόνα όταν προγραμματίζουμε τις λεπτομέρειες και μας βοηθάει να ελέγξουμε τη βασική λογική

¹Μπορεί η μεγαλύτερη από τις τιμές αυτές να είναι μικρότερη από t_f και η t_f να μη συμπεριλαμβάνεται στην ακολουθία.



Σχήμα 2.2: Λογικό διάγραμμα ενός τυπικού προγράμματος των εξισώσεων κίνησης.

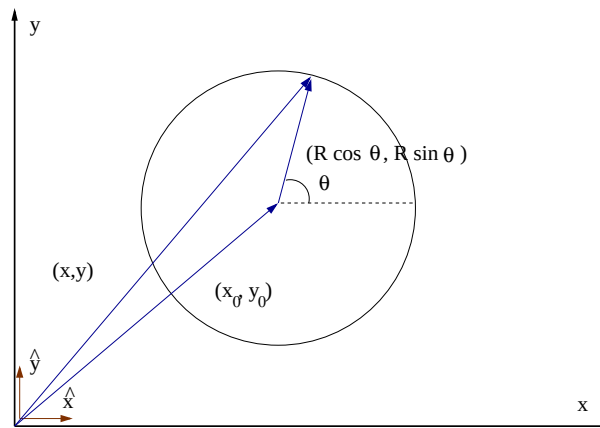
του αλγόριθμου που θα χρησιμοποιήσουμε. Στα ορθογώνια παραλληλόγραμμα σκιαγραφούμε τις σημαντικές εντολές των δομικών στοιχείων του προγράμματος ενώ στους ρόμβους σημειώνουμε τις διακλαδώσεις (branching points) του προγράμματος που προκύπτουν από τις δυνατές τιμές μιας λογικής πρότασης. Με γραμμές ενώνουμε τη λογική σειρά με την οποία εκτελούνται οι διαδικασίες.

Το πρώτο κομμάτι του προγράμματος κάνει τις απαραίτητες δηλώσεις των τύπων των μεταβλητών που θα χρησιμοποιήσουμε. Αν υπάρχουν παράμετροι που παραμένουν σταθερές κατά τη διάρκεια της εκτέλεσης του προγράμματος (λχ. $\pi = 3.1459\dots$, $g = 9.81$, κλπ) τις ορίζουμε εδώ. Στη συνέχεια το πρόγραμμα αλληλεπιδρά με το χρήστη (user interface) και ζητάει τις μεταβλητές παραμέτρους που καθορίζει ο χρήστης: $x_0, y_0, t_0, t_f, dt \dots$. Το πρόγραμμα τυπώνει αυτές τις τιμές στο `stdout` για να μπορεί ο χρήστης να τις ελέγξει ως προς την ορθότητα και να τις αποθηκεύσει για αναφορά στα δεδομένα του. Πριν τον κύριο υπολογισμό, ο χρήστης πρέπει να δώσει τις κατάλληλες αρχικές τιμές σε ορισμένες μεταβλητές, ειδικά στο χρόνο $t = t_0$.

Ο κύριος υπολογισμός γίνεται σε ένα βρόχο (loop) ο οποίος εκτελείται ό-

σο ο χρόνος $t \leq t_f$. Υπολογίζονται οι τιμές της θέσης και της ταχύτητας $x(t), y(t), v_x(t), v_y(t)$ και τυπώνονται μαζί με το χρόνο t σε ένα αρχείο. Εδώ θα κάνουμε τη σημαντική για μας σύμβαση στο φορμά της εξόδου. Αυτό είναι σημαντικό ώστε να έχουμε ενιαίο λογισμικό ανάλυσης των αποτελεσμάτων. Ορίζουμε σε κάθε γραμμή του αρχείου αυτού οι πρώτες πέντε στήλες να είναι οι τιμές t, x, y, v_x, v_y . Μπορούν να υπάρχουν και άλλες στήλες αν χρειαστεί στο ειδικό πρόβλημα που θα μελετάμε, αλλά οι πρώτες πέντε θα είναι πάντα αυτές.

Αφού γίνει αυτό ειδικευόμαστε στο πρόβλημα που θα μελετήσουμε. Ας θεωρήσουμε αρχικά την περίπτωση ενός υλικού σημείου το οποίο εκτελεί ομαλή κυκλική κίνηση. Θεωρούμε το κέντρο του κύκλου (x_0, y_0) , την ακτίνα R και τη γωνιακή ταχύτητα ω να είναι οι βασικές παράμετροι που προσδιορίζουν την κίνηση. Η θέση πάνω στον κύκλο μπορεί τότε να προσδιοριστεί από τη γωνία θ όπως φαίνεται στο Σχήμα 2.3. Θα ορίσουμε την αρχική χρονική στιγμή t_0 να είναι $\theta = 0$.



Σχήμα 2.3: Η τροχιά του υλικού σημείου που εκτελεί την ομαλή κυκλική κίνηση που υπολογίζει το πρόγραμμα `Circle.f`.

Έτσι οι εξισώσεις που δίνουν τη θέση του κινητού κάθε χρονική στιγμή είναι

$$\begin{aligned} x(t) &= x_0 + R \cos(\omega(t - t_0)) \\ y(t) &= y_0 + R \sin(\omega(t - t_0)) . \end{aligned} \quad (2.3)$$

Παραγωγίζοντας ως προς t παίρνουμε την ταχύτητα

$$\begin{aligned} v_x(t) &= -\omega R \sin(\omega(t - t_0)) \\ v_y(t) &= \omega R \cos(\omega(t - t_0)) , \end{aligned} \quad (2.4)$$

και την επιτάχυνση

$$\begin{aligned} a_x(t) &= -\omega^2 R \cos(\omega(t - t_0)) = -\omega^2(x(t) - x_0) \\ a_y(t) &= -\omega^2 R \sin(\omega(t - t_0)) = -\omega^2(y(t) - y_0) . \end{aligned} \quad (2.5)$$

Από τις παραπάνω εξισώσεις παρατηρούμε τις γνωστές γεωμετρικές σχέσεις $\vec{R} \cdot \vec{v} = 0$ ($\vec{R} \equiv \vec{r} - \vec{r}_0$, $\vec{v} \perp \vec{R}$, $\vec{v}$ εφαπτόμενο στην τροχιά) και $\vec{a} = -\omega^2 \vec{R}$ ($\vec{R}$ και $\vec{a}$ αντιπαράλληλα, $\vec{a} \perp \vec{v}$).

Μπορούμε τώρα να σχεδιάσουμε τη δομή των δεδομένων για το πρόγραμμα που θα γράψουμε, η οποία στην περίπτωση μας είναι πολύ απλή. Η σταθερή γωνιακή ταχύτητα ω του υλικού σημείου αποθηκεύεται στην πραγματική (REAL) μεταβλητή `omega`. Το κέντρο του κύκλου (x_0, y_0) , η ακτίνα R και η γωνία θ αντιστοιχούν στις REAL μεταβλητές `x0`, `y0`, `R`, `theta`. Οι χρονικές στιγμές που υπολογίζουμε τη θέση και ταχύτητα του κινητού καθορίζονται από τις παραμέτρους $t_0, t_f, \delta t$ που αντιστοιχούν στο πρόγραμμα στις REAL μεταβλητές `t0`, `tf`, `dt`. Η τρέχουσα θέση του κινητού $(x(t), y(t))$ υπολογίζεται και αποθηκεύεται στις REAL μεταβλητές `x`, `y` και η στιγμιαία του ταχύτητα $(v_x(t), v_y(t))$ στις REAL μεταβλητές `vx`, `vy`. Οι δηλώσεις αυτές γίνονται στην αρχή του προγράμματος με τις εντολές:

```
real x0,y0,R,x,y,vx,vy,t,t0,tf,dt,PI
real theta,omega
parameter(PI=3.1415927)
```

όπου ορίσαμε και την τιμή² του $\pi = 3.1415927$ με την εντολή `parameter`.

Το διαδραστικό με το χρήστη κομμάτι του προγράμματος (user interface) ζητάει από το χρήστη τις τιμές των παραμέτρων που του δίνουμε τη δυνατότητα να προσδιορίζει: `omega`, `x0`, `y0`, `R`, `t0`, `tf`, `dt`. Αρχικά το πρόγραμμα τυπώνει ένα μήνυμα προτροπής (prompt) στο χρήστη προσδιορίζοντας τη μεταβλητή που ζητά να διαβάσει. Αυτό γίνεται με απλές `print` εντολές. Η ανάγνωση των τιμών των παραμέτρων γίνεται από το `stdin` με εντολές `read(5,*)`. Το

²Θυμίζουμε στον αναγνώστη ότι οι μεταβλητές REAL μονής ακρίβειας (4 bytes) έχουν ακρίβεια περίπου 7 δεκαδικών ψηφίων.

‘5’ είναι που προσδιορίζει να διαβαστεί μία γραμμή από το `stdin`. Αφού διαβαστούν οι παράμετροι, το πρόγραμμα τυπώνει τις τιμές που διάβασε στο `stdout`. Αυτό γίνεται για να αποφευχθούν τυπογραφικά λάθη και για να ενημερωθεί ο χρήστης για την τιμή που πραγματικά διάβασε το πρόγραμμα στη μνήμη του υπολογιστή. Επίσης, οδηγώντας το `stdout` σε ένα αρχείο στο σκληρό δίσκο, ο χρήστης μπορεί να αποθηκεύσει τα δεδομένα που εισήγαγε για μελλοντική αναφορά και για χρήση στην ανάλυση των αποτελεσμάτων του:

```
print *, '# Enter omega:'
read(5,*)omega
print *, '# Enter center of circle (x0,y0) and radius R:'
read(5,*)x0,y0,R
print *, '# Enter t0,tf,dt:'
read(5,*)t0,tf,dt
print *, '# omega= ',omega
print *, '# x0= ',x0,' y0= ',y0
print *, '# t0= ',t0,' tf= ',tf,' dt= ',dt
```

Στη συνέχεια το πρόγραμμα θέτει την απαραίτητη αρχική κατάσταση για να γίνει ο υπολογισμός. Αυτό, εκτός από το να θέσει την τιμή του αρχικού χρόνου $t = t_0$, περιλαμβάνει και βασικό έλεγχο για τη νομιμότητα των παραμέτρων που εισήγαγε ο χρήστης. Οι έλεγχοι είναι απαραίτητοι γιατί όταν γράφουμε ένα πρόγραμμα κάνουμε ορισμένες υποθέσεις απαραίτητες για την ορθή λειτουργία του προγράμματος που μπορεί ο χρήστης από σφάλμα ή άγνοια να μην έχει σεβαστεί. Όταν, για παράδειγμα, γράφουμε την έκφραση $2.0*PI/omega$ υποθέτουμε ότι η τιμή του `omega` είναι μη μηδενική ώστε να γίνει η διαίρεση χωρίς να προκύψει μοιραίο σφάλμα κατά την εκτέλεση του προγράμματος. Το πρόγραμμά μας για να λειτουργήσει όπως το σχεδιάσαμε θα απαιτήσουμε να έχουμε $R > 0$ και $\omega > 0$. Αυτό θα το ελέγξουμε με μία εντολή `if` και αν δεν πληρούνται οι υποθέσεις σταματάμε τελείως το πρόγραμμα με την εντολή `stop`³. Το πρόγραμμα επίσης θα ανοίξει το αρχείο `Circle.dat` στο οποίο θα αποθηκεύσουμε τις υπολογισμένες τιμές της θέσης και ταχύτητας του κινητού.

```
if(R .le. 0.0) stop 'Illegal value of R'
if(omega .le. 0.0) stop 'Illegal value of omega'
print *, '# T= ',2.0*PI/omega
```

³Παρατηρήστε πως δεν ελέγξαμε όλες τις υποθέσεις που κάνουμε στο πρόγραμμα. Προσθέστε εσείς τις αναγκαίες σχετικές εντολές.

```
open(unit=11,file='Circle.dat')
t = t0
```

Αν $R \leq 0$ ή $\omega \leq 0$, τότε εκτελούνται οι αντίστοιχες εντολές `stop`. Μετά την εντολή `stop` βάζουμε ένα πληροφοριακό μήνυμα στο χρήστη, ώστε να ξέρει γιατί σταμάτησε το πρόγραμμα, το οποίο τυπώνεται στο `stdout` όταν εκτελείται η εντολή `stop`. Επίσης υπολογίζουμε και τυπώνουμε την περίοδο της κυκλικής κίνησης $T = 2\pi/\omega$. Στην εντολή `open` επιλέξαμε το `unit 11` για να γράφουμε στο `Circle.dat`. Η επιλογή του αριθμού αυτού είναι ελεύθερη για τον προγραμματιστή που μπορεί αν είναι ήσυχος αν διαλέξει έναν οποιοδήποτε αριθμό από 10 έως 99 που δε χρησιμοποιείται ήδη.

Τώρα είμαστε στη φάση που μπορεί να γίνει ο υπολογισμός. Αυτό θα γίνει με ένα βρόχο της μορφής

```
t = t0
do while(t .le. tf)
    .....
    t = t + dt
enddo
```

Η πρώτη εντολή (που τη δείξαμε και παραπάνω) θέτει την αρχική τιμή του χρόνου. Οι εντολές που περιλαμβάνονται ανάμεσα στο `do while( συνθήκη )` και `enddo` εκτελούνται όσο η συνθήκη είναι αληθής. Προσέξτε τη σημασία της εντολής `t = t + dt` χωρίς την οποία ο βρόχος θα εκτελείται επ' αόριστον. Στο ειδικό πρόβλημα που μελετάμε, οι εντολές που μπαίνουν στη θέση των υπολογίζουν τη θέση και την ταχύτητα και τις τυπώνουν στο αρχείο `Circle.dat`:

```
theta = omega * (t-t0)
x = x0+R*cos(theta)
y = y0+R*sin(theta)
vx = -omega*R*sin(theta)
vy = omega*R*cos(theta)
write(11,*)t,x,y,vx,vy
```

Στον υπολογισμό χρησιμοποιούμε τις συναρτήσεις `sin` και `cos` που είναι χτισμένες μέσα στη Fortran 77. Χρησιμοποιούμε την ενδιάμεση μεταβλητή `theta` για να υπολογίσουμε τη φάση $\theta(t) = \omega(t - t_0)$. Με την εντολή `write(11,*)` γράφουμε στο `unit 11` το οποίο συνδέσαμε παραπάνω χρησιμοποιώντας την εντολή `open` με το αρχείο `Circle.dat`.

Το πρόγραμμα το πληκτρολογούμε στο αρχείο `Circle.f`. Η κατάληξη `‘.f’` υποδηλώνει στο μεταγλωττιστή ότι το αρχείο περιέχει κώδικα στη γλώσσα Fortran 77. Για να το μεταγλωττίσουμε και να το τρέξουμε δίνουμε τις εντολές:

```
> f77 Circle.f -o cl
> ./cl
```

Με το διακόπτη `-o cl` δίνουμε την εντολή στο μεταγλωττιστή `f77` να ονομάσει το εκτελέσιμο αρχείο⁴ `cl`. Η δεύτερη εντολή (`./cl`) φορτώνει τις μεταγλωττισμένες εντολές στη μνήμη του υπολογιστή για εκτέλεση. Στη συνέχεια το πρόγραμμά μας ζητάει τα δεδομένα και εκτελεί τον υπολογισμό. Μια πλήρης τυπική συνεδρία εκτέλεσης του προγράμματος δείχνεται παρακάτω:

```
> f77 Circle.f -o cl
> ./cl
# Enter omega:
1.0
# Enter center of circle (x0,y0) and radius R:
1.0 1.0 0.5
# Enter t0,tf,dt:
0.0 20.0 0.01
# omega= 1.
# x0= 1. y0= 1. R= 0.5
# t0= 0. tf= 20. dt= 0.009999999978
# T= 6.28318548
```

Οι γραμμές που αρχίζουν από `#` τυπώνονται από το πρόγραμμα ενώ οι γραμμές με αριθμούς χωρίς `#` είναι οι αριθμητικές τιμές των παραμέτρων που εισάγει ο χρήστης. Αφού πληκτρολογήσει τα δεδομένα, ο χρήστης χρησιμοποιεί το πλήκτρο **Enter** για να τα εισάγει στη μνήμη του υπολογιστή που χρησιμοποιεί το πρόγραμμα. Εδώ $\omega = 1.0$, $x_0 = y_0 = 1.0$, $R = 0.5$, $t_0 = 0.0$, $t_f = 20.0$ και $\delta t = 0.01$.

Το πρόγραμμα μπορείτε να το εκτελέσετε πολλές φορές για διαφορετικές τιμές των παραμέτρων με τη βοήθεια του editor. Σε ένα αρχείο με όνομα λ.χ. `Circle.in` τυπώστε τα δεδομένα που θέλετε να εισάγετε στο πρόγραμμα:

```
1.0                omega
```

⁴Θυμίζουμε ότι αν δε βάλουμε το διακόπτη αυτό το εκτελέσιμο αρχείο θα ονομαστεί από το μεταγλωττιστή `a.out` από προεπιλογή.

```
1.0 1.0 0.5 (x0, y0) , R
0.0 20.0 0.01 t0 tf dt
```

Σε κάθε γραμμή του αρχείου βάζουμε τις τιμές των παραμέτρων που διαβάζει κάθε εντολή `read` με τη σωστή σειρά. Αφού δώσουμε όλες τις παραμέτρους, η Fortran 77 προχωράει στην επόμενη εντολή και η επόμενη εντολή `read` θα διαβάσει μια νέα σειρά από το αρχείο. Έτσι δίπλα από τις παραμέτρους βάζουμε σχόλια για το χρήστη που του θυμίζουν τη σειρά που διαβάζονται οι μεταβλητές και που το πρόγραμμα θα αγνοήσει. Το πρόγραμμα τώρα μπορούμε να το τρέξουμε με την εντολή:

```
> ./c1 < Circle.in > Circle.out
```

Η εντολή `./c1` τρέχει τις εντολές που περιέχονται στο εκτελέσιμο αρχείο `c1`. Το `< Circle.in` οδηγεί τα περιεχόμενα του αρχείου `Circle.in` στο standard input (`stdin`) του προγράμματος, εξαναγκάζοντας το `c1` να διαβάσει τα δεδομένα από το αρχείο αυτό. Το `> Circle.out` οδηγεί το `stdout` του προγράμματος `c1` στο αρχείο `Circle.out` του οποίου τα περιεχόμενα μπορούμε να τα δούμε, αφού τρέξουμε το πρόγραμμα, με την εντολή `cat`:

```
> cat Circle.out
# Enter omega:
# Enter center of circle (x0,y0) and radius R:
# Enter t0,tf,dt:
# omega= 1.
# x0= 1. y0= 1. R= 0.5
# t0= 0. tf= 20. dt= 0.00999999978
# T= 6.28318548
```

Κλείνουμε την παράγραφο αυτή παραθέτοντας για διευκόλυνση του αναγνώστη ολόκληρο το πρόγραμμα που πληκτρολογήσαμε στο αρχείο `Circle.f`:

```
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      File Circle.f
C      Constant angular velocity circular motion
C      Set (x0,y0) center of circle, its radius R and omega.
C      At t=t0, the particle is at theta=0
C      -----
      program Circle
      implicit none
```

```

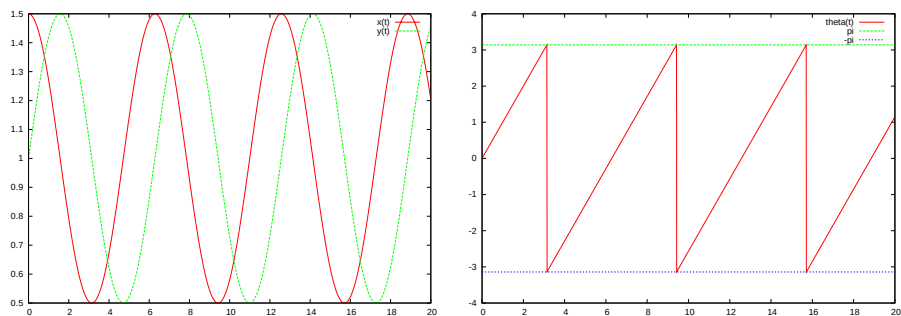
C -----
C Declaration of variables
C real x0,y0,R,x,y,vx,vy,t,t0,tf,dt,PI
C real theta,omega
C parameter(PI=3.1415927)
C -----
C Ask user for input:
C print *,'# Enter omega:'
C read(5,*)omega
C print *,'# Enter center of circle (x0,y0) and radius R:'
C read(5,*)x0,y0,R
C print *,'# Enter t0,tf,dt:'
C read(5,*)t0,tf,dt
C print *,'# omega= ',omega
C print *,'# x0= ',x0,' y0= ',y0,' R= ',R
C print *,'# t0= ',t0,' tf= ',tf,' dt= ',dt
C -----
C Initialize
C if(R .le. 0.0) stop 'Illegal value of R'
C if(omega .le. 0.0) stop 'Illegal value of omega'
C print *,'# T= ',2.0*PI/omega
C open(unit=11,file='Circle.dat')
C -----
C Compute:
C t = t0
C do while(t .le. tf)
C   theta = omega * (t-t0)
C   x = x0+R*cos(theta)
C   y = x0+R*sin(theta)
C   vx = -omega*R*sin(theta)
C   vy = omega*R*cos(theta)
C   write(11,*)t,x,y,vx,vy
C   t = t + dt
C enddo
C close(11)
C end

```


2.1.1 Απεικόνιση των Δεδομένων

Οι γραφικές παραστάσεις των αποτελεσμάτων που πήραμε στην προηγούμενη παράγραφο μελετώνται με τη βοήθεια του προγράμματος `gnuplot`. Υπενθυμίζουμε ότι τα δεδομένα το πρόγραμμά μας τα αποθηκεύει στο αρχείο `Circle.dat` σε πέντε στήλες: Η 1η είναι ο χρόνος t , η 2η και 3η οι συντεταγμένες x , y και η 4η και 5η οι συντεταγμένες της ταχύτητας v_x , v_y . Τα διαγράμματα $x(t)$ και $y(t)$ παράγονται από τις εντολές (που δίνονται μέσα από το `gnuplot`) και μπορείτε να τα δείτε στο (αριστερό) Σχήμα 2.4:

```
gnuplot> plot "Circle.dat" using 1:2 with lines title "x(t)"
gnuplot> replot "Circle.dat" using 1:3 with lines title "y(t)"
```



Σχήμα 2.4: Τα διαγράμματα $(x(t), y(t))$ (αριστερά) και $\theta(t)$ (δεξιά) των δεδομένων που παράγονται από το πρόγραμμα `Circle.f` για $\omega = 1.0$, $x_0 = y_0 = 1.0$, $R = 0.5$, $t_0 = 0.0$, $t_f = 20.0$ και $\delta t = 0.01$.

Η δεύτερη εντολή (`replot`) βάζει τη δεύτερη γραφική παράσταση μαζί με την πρώτη.

Ας δούμε τώρα πώς μπορούμε να φτιάξουμε τη γραφική παράσταση της γωνίας $\theta(t)$. Αυτό μπορούμε να το κάνουμε μέσα από το `gnuplot`, χωρίς να γράψουμε κάποιο καινούργιο πρόγραμμα, από τα δεδομένα μέσα στο αρχείο `Circle.dat`. Παρατηρούμε ότι $\theta(t) = \tan^{-1}((y - y_0)/(x - x_0))$. Η συνάρτηση `atan2` (που υπάρχει και στη Fortran) είναι διαθέσιμη στο `gnuplot`⁵. Για να βρούμε πώς δουλεύει χρησιμοποιούμε τη βοήθεια στο `gnuplot`:

⁵Όπως και όλες οι μαθηματικές συναρτήσεις που υπάρχουν στη μαθηματική βιβλιοθήκη της γλώσσας C. Δώστε την εντολή `help functions` για να δείτε σχετικά.

```
gnuplot> help atan2
```

The 'atan2(y,x)' function returns the arc tangent (inverse tangent) of the ratio of the real parts of its arguments. 'atan2' returns its argument in radians or degrees, as selected by 'set angles', in the correct quadrant.

Άρα αρκεί να καλέσουμε τη συνάρτηση αυτή με εντολή της μορφής $\text{atan2}(y-y_0, x-x_0)$. Στην περίπτωση μας $x_0=y_0=1$ ενώ τα x, y είναι αντίστοιχα στη 2η και 3η στήλη κάθε γραμμής του αρχείου `Circle.dat`. Θα φτιάξουμε μία κατάλληλη έκφραση στην εντολή `using` στο `gnuplot` όπως και στη σελίδα 58 όπου $\$2$ η τιμή της 2ης και $\$3$ η τιμή της 3ης στήλης:

```
gnuplot> x0 = 1 ; y0 = 1
gnuplot> plot "Circle.dat" using 1:(atan2($3-y0,$2-x0)) \
with lines title "theta(t)",pi,-pi
```

Η δεύτερη εντολή δίνεται σε μία γραμμή την οποία τη σπάσαμε με το χαρακτήρα `\` ώστε να χωράει στο κείμενο⁶. Προσέξτε πώς ορίσαμε τις τιμές των μεταβλητών x_0, y_0 μέσα στο `gnuplot` και τις χρησιμοποιήσαμε στην έκφραση $\text{atan2}(\$3-x_0, \$2-y_0)$ αντί (ισοδύναμα) να γράφαμε $\text{atan2}(\$3-1, \$2-1)$. Επίσης μαζί με τη γραφική παράσταση της $\theta(t)$ κάνουμε και τις γραφικές παραστάσεις των σταθερών συναρτήσεων $f_1(t) = \pi$, $f_2(t) = -\pi$ ώστε να ελέγξουμε τα όρια των τιμών της $\theta(t)$. Η μεταβλητή⁷ `pi` στο `gnuplot` έχει ορισμένη εσωτερικά την τιμή π . Το αποτέλεσμα μπορείτε να το δείτε στο (δεξί) Σχήμα 2.4.

Οι συνιστώσες των ταχυτήτων $(v_x(t), v_y(t))$ σα συνάρτηση του χρόνου και η τροχιά του υλικού σημείου $\vec{r}(t)$ μπορούν να απεικονιστούν με τις εντολές:

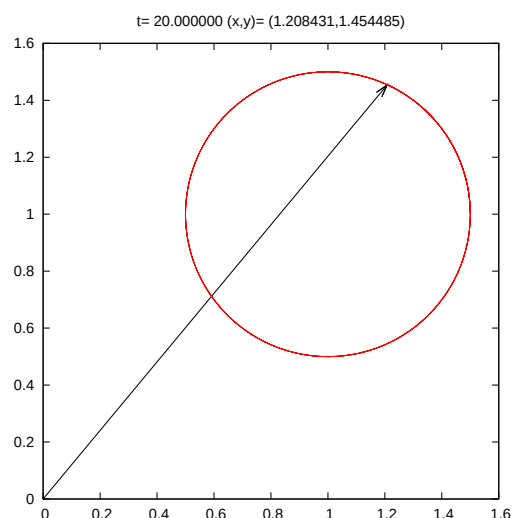
```
gnuplot> plot "Circle.dat" using 1:4 title "v_x(t)" with lines
gnuplot> replot "Circle.dat" using 1:5 title "v_y(t)" with lines
gnuplot> plot "Circle.dat" using 2:3 title "x-y" with lines
```

Η τελευταία εντολή τοποθετεί τα σημεία x, y στο επίπεδο αφού αυτά βρίσκονται στις στήλες 2 και 3 αντίστοιχα.

Κλείνουμε αυτήν την παράγραφο δείχνοντας πώς μπορούμε να δούμε το υλικό σημείο να κινείται κάνοντας στοιχειώδες animation με το `gnuplot`. Για το σκοπό αυτό έχουμε ένα αρχείο `animate2D.gnu` στο συνοδευτικό λογισμικό το οποίο θα πρέπει να αντιγράψετε μέσα στον κατάλογο που έχετε το αρχείο των

⁶ Αυτό επιτρέπεται να το κάνετε και μέσα στο πρόγραμμα `gnuplot` αν το θέλετε.

⁷ Δώστε την εντολή `show variables` για να δείτε τις ορισμένες μεταβλητές στο `gnuplot`.



Σχήμα 2.5: Η τροχιά του υλικού σημείο όπως απεικονίζεται κάθε χρονική στιγμή από το πρόγραμμα `animate2D.gnu` του συνοδευτικού λογισμικού που τρέχει μέσα από το `gnuplot`. Φαίνεται το διάνυσμα θέσης και στον τίτλο του διαγράμματος η χρονική στιγμή t και αντίστοιχη θέση (x,y) . Τα δεδομένα είναι από το πρόγραμμα `Circle.f` που περιγράφεται στο κείμενο.

δεδομένων σας `Circle.dat` και από όπου θα δώσετε και την εντολή `gnuplot`. Δεν είναι σκοπός εδώ να σας εξηγήσουμε πώς δουλεύει αλλά πώς να το χρησιμοποιείτε⁸. Στο Σχήμα 2.5 φαίνεται το τελικό αποτέλεσμα. Οι εντολές είναι απλές. Αρκεί να ορίσουμε από ποιο αρχείο να διαβάσουμε τα δεδομένα⁹, τον αρχικό χρόνο απεικόνισης t_0 της τροχιάς, τον τελικό t_f καθώς και το βήμα dt . Οι χρόνοι αυτοί δεν είναι ανάγκη να είναι οι ίδιοι με αυτούς που βάλαμε στα δεδομένα όταν τρέχαμε το πρόγραμμα που βρίσκεται στο αρχείο `Circle.f`. Έτσι μπορούμε να ρυθμίσουμε πόσο αργά ή γρήγορα θα ‘τρέχει’ το υλικό σημείο στην οθόνη μας μεταβάλλοντας το dt . Ή αν θέλουμε να απεικονίσουμε ένα συγκεκριμένο χρονικό διάστημα μπορούμε να μεταβάλλουμε τα t_0 , t_f . Οι εντολές που δίνουμε είναι οι

```
gnuplot> file = "Circle.dat"
```

⁸Φυσικά μπορείτε να δείτε τα περιεχόμενά του και να μαντέψετε πώς δουλεύει, είναι αρκετά απλό.

⁹Μπορεί να είναι οποιοδήποτε αρχείο του οποίου η 1η, 2η και 3η στήλη έχει το χρόνο t και x και y συνιστώσες της θέσης του υλικού σημείο στο επίπεδο αντίστοιχα.

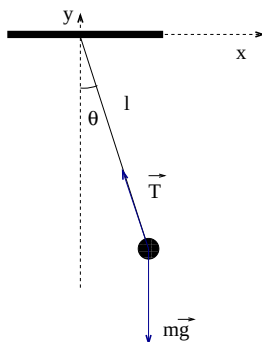
```
gnuplot> set xrange [0:1.6]; set yrange [0:1.6]
gnuplot> t0 = 0; tf = 20 ; dt = 0.1
gnuplot> load "animate2D.gnu"
```

Η πρώτη εντολή καθορίζει το αρχείο από όπου το `animate2D.gnu` θα διαβάσει τα δεδομένα από την 1η-3η στήλη του. Η δεύτερη εντολή καθορίζει τις παραμέτρους του χρόνου που θα χρησιμοποιηθούν στο animation (αρχικός, τελικός και βήμα). Η τρίτη ορίζει τα όρια στους άξονες x και y . Και η τέταρτη είναι η εντολή που 'τρέχει' το animation. Αν θέλετε να ξανατρέξει το animation αρκεί να δώσετε την τελευταία εντολή όσες φορές θέλετε. Τις τρεις πρώτες τις δίνετε όταν θέλετε να αλλάξετε τις αντίστοιχες παραμέτρους. Λ.χ. αν θέλετε να τρέξετε το animation στα ίδια δεδομένα με τη 'μισή ταχύτητα' αρκεί να ξαναορίσετε το $dt=0.05$, να επαναφέρετε το χρόνο $t0=0$ και να δώσετε μόνο τις εντολές

```
gnuplot> t0 = 0; dt = 0.05
gnuplot> load "animate2D.gnu"
```

2.1.2 Άλλα Παραδείγματα

Ας εφαρμόσουμε όσα κάναμε για την απλή κυκλική κίνηση και σε διαφορετικά παραδείγματα κίνησης στο επίπεδο. Το πρώτο πρόβλημα που θα παρουσιάσουμε είναι αυτό του απλού εκκρεμούς που εκτελεί μικρές ταλαντώσεις γύρω από την κατακόρυφο και που φαίνεται στο Σχήμα 2.6. Η κίνηση περιγράφεται από



Σχήμα 2.6: Το απλό εκκρεμές του οποίου η κίνηση για $\theta \ll 1$ περιγράφεται από το πρόγραμμα `SimplePendulum.f`.

τη χρονική εξάρτηση του μοναδικού βαθμού ελευθερίας του συστήματος, της

γωνίας $\theta(t)$. Η κίνηση είναι περιοδική με γωνιακή συχνότητα $\omega = \sqrt{g/l}$ και περίοδο $T = 2\pi/\omega$. Η γωνιακή ταχύτητα υπολογίζεται από τη σχέση $\dot{\theta} \equiv d\theta/dt$ και παίρνουμε:

$$\begin{aligned}\theta(t) &= \theta_0 \cos(\omega(t - t_0)) \\ \dot{\theta}(t) &= -\omega\theta_0 \sin(\omega(t - t_0))\end{aligned}\quad (2.6)$$

Με τις παραπάνω σχέσεις έχουμε επιλέξει τις αρχικές συνθήκες $\theta(0) = \theta_0$, $\dot{\theta}(0) = 0$. Για να γράψουμε τις εξισώσεις της τροχιάς στο καρτεσιανό σύστημα συντεταγμένων του Σχήματος 2.6 χρησιμοποιούμε τις σχέσεις

$$\begin{aligned}x(t) &= l \sin(\theta(t)) \\ y(t) &= -l \cos(\theta(t)) \\ v_x(t) &= \frac{dx(t)}{dt} = l\dot{\theta}(t) \cos(\theta(t)) \\ v_y(t) &= \frac{dy(t)}{dt} = l\dot{\theta}(t) \sin(\theta(t))\end{aligned}\quad (2.7)$$

Αυτές είναι οι ανάλογες των εξισώσεων (2.3) και (2.4) για την περίπτωση της ομαλής κυκλικής κίνησης. Έτσι ο σχεδιασμός του προγράμματος γίνεται με παρόμοιο τρόπο. Η τελική του μορφή, την οποία βρίσκουμε στο αρχείο `SimplePendulum.f` στο συνοδευτικό λογισμικό είναι:

```
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   File SimplePendulum.f
C   Set pendulum original position at theta0 with no initial speed
C   -----
C   program SimplePendulum
C   implicit none
C   -----
C   Declaration of variables
C   real l,g,x,y,vx,vy,t,t0,tf,dt,PI
C   real theta,theta0,dtheta_dt,omega
C   parameter(PI=3.1415927,g=9.81)
C   -----
C   Ask user for input:
C   print *,'# Enter l: '
C   read(5,*)l
C   print *,'# Enter theta0:'
```

```

      read(5,*)theta0
      print *,'# Enter t0,tf,dt:'
      read(5,*)t0,tf,dt
      print *,'# l= ',l ,' theta0= ',theta0
      print *,'# t0= ',t0,' tf= ',tf,' dt= ',dt
C -----
C   Initialize
      omega = sqrt(g/l)
      print *,'# omega= ',omega,' T= ',2.0*PI/omega
      open(unit=11,file='SimplePendulum.dat')
C -----
C   Compute:
      t = t0
      do while(t .le. tf)
         theta = theta0*cos(omega*(t-t0))
         dtheta_dt = -omega*theta0*sin(omega*(t-t0))
         x = l*sin(theta)
         y = -l*cos(theta)
         vx = l*dtheta_dt*cos(theta)
         vy = l*dtheta_dt*sin(theta)
         write(11,100)t,x,y,vx,vy,theta,dtheta_dt
         t = t + dt
      enddo
      close(11)
100  FORMAT(7G15.7)
      end

```

Τα μόνα σημεία που θα σημειώσουμε στον αναγνώστη είναι ότι την επιτάχυνση της βαρύτητας g τη θέτουμε σταθερή στο πρόγραμμα και ο χρήστης μπορεί να καθορίσει το μήκος του εκκρεμούς l και ότι το αρχείο των δεδομένων περιέχει εκτός από τις στήλες 1-5 και άλλες 2 στις οποίες μπορούμε να δούμε τη γωνία και στιγμιαία γωνιακή ταχύτητα του εκκρεμούς. Στην εντολή `write(11,100)` παραπέμπουμε σε κατάλληλη εντολή `FORMAT` έτσι ώστε τα δεδομένα να τυπώνονται σε μία γραμμή. Δείτε τη συζήτηση στη σελίδα 52.

Μια απλή συνεδρία μελέτης του προβλήματος συνοψίζεται παρακάτω:

```

> f77 SimplePendulum.f -o sp
> ./sp
# Enter l:

```

```

1.0
# Enter theta0:
0.314
# Enter t0,tf,dt:
0 20 0.01
# l=      1.      theta0=  0.31400001
# t0=     0.      tf=     20. dt=    0.009999999978
# omega=  3.132092 T=     2.0060668
> gnuplot
gnuplot> plot "SimplePendulum.dat" using 1:2 with lines title "x(t)"
gnuplot> plot "SimplePendulum.dat" using 1:3 with lines title "y(t)"
gnuplot> plot "SimplePendulum.dat" using 1:4 with lines title "v_x(t)"
gnuplot> replot "SimplePendulum.dat" using 1:5 with lines title "v_y(t)"
gnuplot> plot "SimplePendulum.dat" using 1:6 with lines title "theta(t)"
gnuplot> replot "SimplePendulum.dat" using 1:7 with lines title "theta'(t)"
gnuplot> plot [-0.6:0.6][-1.1:0.1] "SimplePendulum.dat" \
                                using 2:3 with lines title "x-y"

gnuplot> file = "SimplePendulum.dat"
gnuplot> t0=0;tf=20.0;dt=0.1
gnuplot> set xrange [-0.6:0.6];set yrange [-1.1:0.1]
gnuplot> load "animate2D.gnu"

```

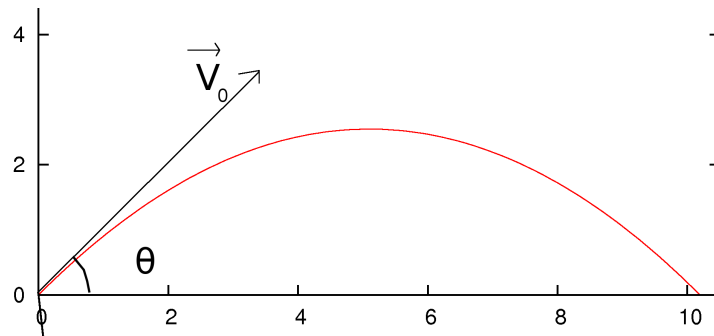
Το επόμενο πρόβλημα θα είναι η μελέτη της βολής υλικού σημείου κοντά στην επιφάνεια της γης¹⁰ αγνοώντας όλες τις δυνάμεις τριβής του αέρα. Όπως γνωρίζουμε, η τροχιά του σωματιδίου και η ταχύτητά του δίνονται από τις παραμετρικές εξισώσεις με παράμετρο το χρόνο:

$$\begin{aligned}
 x(t) &= v_{0x}t \\
 y(t) &= v_{0y}t - \frac{1}{2}gt^2 \\
 v_x(t) &= v_{0x} \\
 v_y(t) &= v_{0y} - gt
 \end{aligned} \tag{2.8}$$

όπου έχουμε υποθέσει τις αρχικές συνθήκες $x(0) = y(0) = 0$, $v_x(0) = v_{0x} = v_0 \cos \theta$ και $v_y(0) = v_{0y} = v_0 \sin \theta$ σύμφωνα με το Σχήμα 2.7.

Η δομή του προγράμματος είναι παρόμοια με τα προηγούμενα. Διαλέγουμε ο χρήστης να εισάγει το μέτρο της αρχικής ταχύτητας και τη γωνία θ σε

¹⁰Δηλ. $\vec{g}$ = σταθ. και η επίδραση της δύναμης Coriolis αμελητέα.



Σχήμα 2.7: Βολή υλικού σημείου κοντά στην επιφάνεια της γης με $x(0) = y(0) = 0$, $v_x(0) = v_{0x} = v_0 \cos \theta$ και $v_y(0) = v_{0y} = v_0 \sin \theta$.

μοίρες που σχηματίζει με την οριζόντια διεύθυνση. Εδώ παίρνουμε το χρόνο $t_0 = 0$. Το πρόγραμμα υπολογίζει τις v_{0x} και v_{0y} και τις τυπώνει στο χρήστη στο `stdout`. Τα δεδομένα σώζονται στο αρχείο `Projectile.dat`. Το πρόγραμμα δίνεται ολόκληρο παρακάτω και βρίσκεται στο αρχείο `Projectile.f` του συνοδευτικού λογισμικού.

```
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      File Projectile.f
C      Shooting a projectile near the earth surface. No air resistance
C      Starts at (0,0), set (v0,theta).
C      -----
C      program Projectile
C      implicit none
C      -----
C      Declaration of variables
C      real x0,y0,R,x,y,vx,vy,t,tf,dt,PI,g
C      real theta,v0x,v0y,v0
C      parameter(PI=3.1415927,g=9.81)
C      -----
C      Ask user for input:
C      print *,'# Enter v0,theta (in degrees):'
C      read(5,*)v0,theta
C      print *,'# Enter tf,dt:'
C      read(5,*) tf,dt
C      print *,'# v0= ',v0,' theta= ',theta,'o (degrees)'
```



```

      print *,'# t0= ',0.0,' tf= ',tf,' dt= ',dt
C -----
C Initialize
  if( v0 .le. 0.0) stop 'Illegal value of v0<=0'
  if( theta .le. 0.0 .or. theta .ge. 90.0)
*    stop 'Illegal value of theta'
  theta = (PI/180.0)*theta !convert to radians
  v0x   = v0*cos(theta)
  v0y   = v0*sin(theta)
  print *,'# v0x = ',v0x,' v0y= ',v0y
  open(unit=11,file='Projectile.dat')
C -----
C Compute:
  t = 0.0
  do while(t .le. tf)
    x = v0x * t
    y = v0y * t - 0.5*g*t*t
    vx = v0x
    vy = v0y - g*t
    write(11,*)t,x,y,vx,vy
    t = t + dt
  enddo
  close(11)
end

```

Για διευκόλυνση του αναγνώστη, δίνουμε πάλι τις εντολές μιας τυπικής συνεδρίας μελέτης του προβλήματος:

```

> f77 Projectile.f -o pj
> ./pj
# Enter v0,theta (in degrees):
10 45
# Enter tf,dt:
1.4416 0.001
# v0= 10.0000000 theta= 45.000000 o (degrees)
# t0= 0.0000000 tf= 1.4416000 dt= 1.00000005E-03
# v0x = 7.0710678 v0y= 7.0710678
> gnuplot
gnuplot> plot "Projectile.dat" using 1:2 with lines title "x(t)"

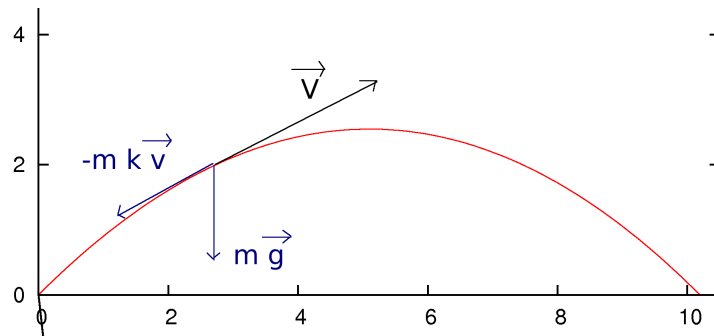
```

```

gnuplot> replot "Projectile.dat" using 1:3 with lines title "y(t)"
gnuplot> plot "Projectile.dat" using 1:4 with lines title "v_x(t)"
gnuplot> replot "Projectile.dat" using 1:5 with lines title "v_y(t)"
gnuplot> plot "Projectile.dat" using 2:3 with lines title "x-y"
gnuplot> file = "Projectile.dat"
gnuplot> set xrange [0:10.3];set yrange [0:10.3]
gnuplot> t0=0;tf=1.4416;dt=0.05
gnuplot> load "animate2D.gnu"

```

Στη συνέχεια ας δούμε το σύστημα όταν στο υλικό σημείο δρα ομαλή αντίσταση από ένα ρευστό της μορφής $\vec{F} = -mk\vec{v}$, δηλ. μια δύναμη που είναι αντίθετη με την ταχύτητα σε κάθε χρονική στιγμή. Οι λύσεις των εξισώσεων κίνησης



Σχήμα 2.8: Οι δυνάμεις που ασκούνται στο σωματίδιο του Σχήματος 2.7 όταν υποθέσουμε και την ύπαρξη αντίστασης του αέρα ανάλογης του μέτρου της ταχύτητας $\vec{F} = -mk\vec{v}$.

$$\begin{aligned}
 a_x &= \frac{dv_x}{dt} = -kv_x \\
 a_y &= \frac{dv_y}{dt} = -kv_y - g
 \end{aligned}
 \tag{2.9}$$

με αρχικές συνθήκες $x(0) = y(0) = 0$, $v_x(0) = v_{0x} = v_0 \cos \theta$ και $v_y(0) = v_{0y} = v_0 \sin \theta$ έχουν λύση¹¹

$$\begin{aligned} v_x(t) &= v_{0x} e^{-kt} \\ v_y(t) &= \left(v_{0y} + \frac{g}{k}\right) e^{-kt} - \frac{g}{k} \\ x(t) &= \frac{v_{0x}}{k} (1 - e^{-kt}) \\ y(t) &= \frac{1}{k} \left(v_{0y} + \frac{g}{k}\right) (1 - e^{-kt}) - \frac{g}{k} t \end{aligned} \quad (2.10)$$

Οι εξισώσεις αυτές προγραμματίζονται με παρόμοιο τρόπο όπως και στην περίπτωση της απουσίας της αντίστασης του αέρα. Η μόνη διαφορά είναι ότι ο χρήστης εισάγει τη σταθερά k στα δεδομένα και φυσικά η μορφή των εξισώσεων. Το πρόγραμμα βρίσκεται στο αρχείο `ProjectileAirResistance.f` στο συνοδευτικό υλικό και παρατίθεται αυτούσιο παρακάτω:

```
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   File ProjectileAirResistance.f
C   Shooting a projectile near the earth surface with air resistance
C   Starts at (0,0), set k, (v0,theta).
C   -----
C   program ProjectileAirResistance
C   implicit none
C   -----
C   Declaration of variables
C   real x0,y0,R,x,y,vx,vy,t,tf,dt,PI,g,k
C   real theta,v0x,v0y,v0
C   parameter(PI=3.1415927,g=9.81)
C   -----
C   Ask user for input:
C   print *,'# Enter k, v0,theta (in degrees):'
C   read(5,*)k, v0,theta
C   print *,'# Enter tf,dt:'
C   read(5,*) tf,dt
C   print *,'# k = ',k
C   print *,'# v0= ',v0,' theta= ',theta,'o (degrees)'
C   print *,'# t0= ',0.0,' tf= ',tf,' dt= ',dt
```

¹¹Ο αναγνώστης καλείται να αποδείξει τις εξισώσεις (2.10) σαν άσκηση.

```

C -----
C Initialize
  if( v0 .le. 0.0) stop 'Illegal value of v0<=0'
  if( k .le. 0.0) stop 'Illegal value of k <=0'
  if( theta .le. 0.0 .or. theta .ge. 90.0)
*    stop 'Illegal value of theta'
  theta = (PI/180.0)*theta !convert to radians
  v0x = v0*cos(theta)
  v0y = v0*sin(theta)
  print *, '# v0x = ',v0x,' v0y= ',v0y
  open(unit=11,file='ProjectileAirResistance.dat')
C -----
C Compute:
  t = 0.0
  do while(t .le. tf)
    x = (v0x/k)*(1.0-exp(-k*t))
    y = (1.0/k)*(v0y+(g/k))*(1.0-exp(-k*t))-(g/k)*t
    vx = v0x*exp(-k*t)
    vy = (v0y+(g/k))*exp(-k*t)-(g/k)
    write(11,*)t,x,y,vx,vy
    t = t + dt
  enddo
  close(11)
end

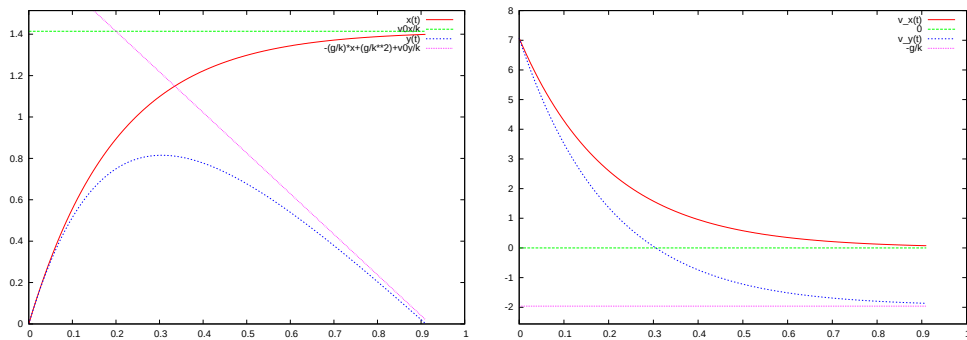
```

Δίνουμε πάλι τις εντολές μιας τυπικής συνεδρίας μελέτης του προβλήματος:

```

> f77 ProjectileAirResistance.f -o pja
> ./pja
  # Enter k, v0,theta (in degrees):
5.0 10.0 45
  # Enter tf,dt:
0.91 0.001
  # k = 5.
  # v0= 10. theta= 45.o (degrees)
  # t0= 0. tf= 0.910000026 dt= 0.00100000005
  # v0x = 7.07106781 v0y= 7.07106781
> gnuplot
gnuplot> v0x = 10*cos(pi/4) ; v0y = 10*sin(pi/4) ;g = 9.81 ; k = 5

```

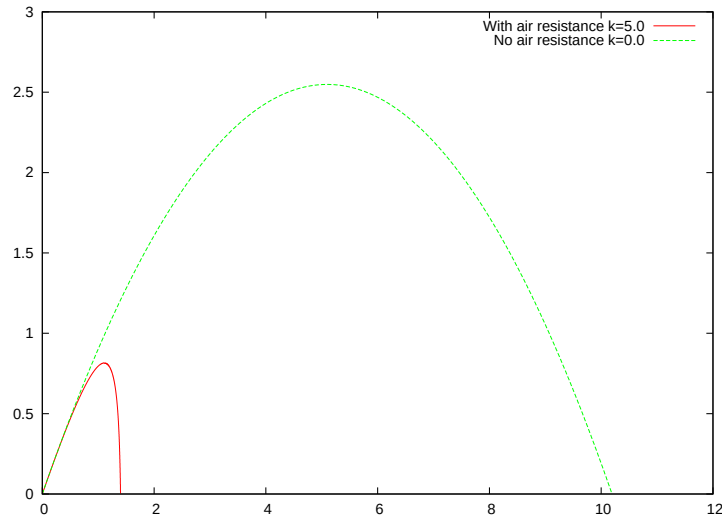


Σχήμα 2.9: Τα διαγράμματα $x(t), y(t)$ (αριστερά) και $v_x(t), v_y(t)$ (δεξιά) των δεδομένων που παράγονται από το πρόγραμμα `ProjectileAirResistance.f` για $k = 5.0$, $v_0 = 10.0$, $\theta = \pi/4$, $t_f = 0.91$ και $\delta t = 0.001$. Φαίνονται και οι ασύμπτωτες των συναρτήσεων καθώς $t \rightarrow \infty$.

```
gnuplot> plot [:][:v0x/k+0.1] "ProjectileAirResistance.dat" \
    using 1:2 with lines title "x(t)",v0x/k
gnuplot> replot "ProjectileAirResistance.dat" \
    using 1:3 with lines title "y(t)",-(g/k)*x+(g/k**2)+v0y/k
gnuplot> plot [:][-g/k-0.6:] "ProjectileAirResistance.dat" \
    using 1:4 with lines title "v_x(t)",0
gnuplot> replot "ProjectileAirResistance.dat" \
    using 1:5 with lines title "v_y(t)",-g/k
gnuplot> plot "ProjectileAirResistance.dat" \
    using 2:3 with lines title "With air resistance k=5.0"
gnuplot> replot "Projectile.dat" \
    using 2:3 with lines title "No air resistance k=0.0"
gnuplot> file = "ProjectileAirResistance.dat"
gnuplot> set xrange [0:1.4];set yrange [0:1.4]
gnuplot> t0=0;tf=0.91;dt=0.01
gnuplot> load "animate2D.gnu"
```

Όπως και παραπάνω διπλώσαμε τις εντολές που δε χωρούσαν σε δύο γραμμές. Ορίσαμε τις μεταβλητές `gnuplot` $v0x$, $v0y$, g και k να έχουν τις τιμές που χρησιμοποιήσαμε όταν τρέξαμε το πρόγραμμα. Μπορούμε έτσι και κατασκευάζουμε τις ασύμπτωτες των συναρτήσεων¹². Τα αποτελέσματα φαίνονται στα

¹²Θυμίζουμε στον αναγνώστη ότι η ανεξάρτητη μεταβλητή από προεπιλογή στο `gnuplot` είναι η x .



Σχήμα 2.10: Οι τροχιές των βλημάτων που βάλλονται με $v_0 = 10.0$, $\theta = \pi/4$ όταν δεν υπάρχει αντίσταση του αέρα και όταν υπάρχει με $k = 5.0$.

Σχήματα 2.9 και 2.10.

Το τελευταίο παράδειγμα που θα μελετήσουμε στην παράγραφο αυτή είναι αυτό του ανισότροπου αρμονικού ταλαντωτή. Αυτό είναι ένα υλικό σημείο υπό την επίδραση τις δύναμης

$$F_x = -m\omega_1^2 x \quad F_y = -m\omega_2^2 y \quad (2.11)$$

όπου οι ‘σταθερές των ελατηρίων’ $k_1 = m\omega_1^2$ και $k_2 = m\omega_2^2$ είναι διαφορετικές στις δύο ορθογώνιες κατευθύνσεις που ορίζονται από τους καρτεσιανούς άξονες x και y . Οι λύσεις των εξισώσεων αυτών για $x(0) = A$, $y(0) = 0$, $v_x(0) = 0$ και $v_y(0) = \omega_2 A$ είναι

$$\begin{aligned} x(t) &= A \cos(\omega_1 t) & y(t) &= A \sin(\omega_2 t) \\ v_x(t) &= -\omega_1 A \sin(\omega_1 t) & v_y(t) &= \omega_2 A \cos(\omega_2 t) \end{aligned} \quad (2.12)$$

Για ορισμένες σχέσεις μεταξύ των κυκλικών συχνοτήτων ω_1 και ω_2 οι καμπύλες των τροχιών του υλικού σημείου είναι κλειστές και αυτοτέμνονται σε ένα συγκεκριμένο αριθμό από σημεία. Θα αφήσουμε τον αναγνώστη να βρει τις σχέσεις αυτές και να τις επιβεβαιώσει από τα αριθμητικά αποτελέσματα που θα προκύψουν από το παρακάτω πρόγραμμα, το οποίο βρίσκεται στο αρχείο `Lissajoux.f` στο συνοδευτικό λογισμικό.

```

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   File Lissajous.f
C   Lissajous curves (special case)  $x(t) = \cos(o_1 t)$ ,  $y(t) = \sin(o_2 t)$ 
C   -----
C   program Lissajous
C   implicit none
C   -----
C   Declaration of variables
C   real x0,y0,R,x,y,vx,vy,t,t0,tf,dt,PI
C   real o1,o2,T1,T2
C   parameter(PI=3.1415927)
C   -----
C   Ask user for input:
C   print *,'# Enter omega1 and omega2:'
C   read(5,*)o1,o2
C   print *,'# Enter tf,dt:'
C   read(5,*)tf,dt
C   print *,'# o1= ',o1, ' o2= ',o2
C   print *,'# t0= ',0.0, ' tf= ',tf, ' dt= ',dt
C   -----
C   Initialize
C   if(o1.le.0.0 .or. o2.le.0.0) stop 'Illegal omega1 or omega2<=0'
C   T1 = 2.0*PI/o1
C   T2 = 2.0*PI/o2
C   print *,'# T1= ',T1, ' T2= ',T2
C   open(unit=11,file='Lissajous.dat')
C   -----
C   Compute:
C   t = 0.0
C   do while(t .le. tf)
C     x = cos(o1*t)
C     y = sin(o2*t)
C     vx = -o1*sin(o1*t)
C     vy = o2*cos(o2*t)
C     write(11,*)t,x,y,vx,vy
C     t = t + dt
C   enddo
C   close(11)

```

end

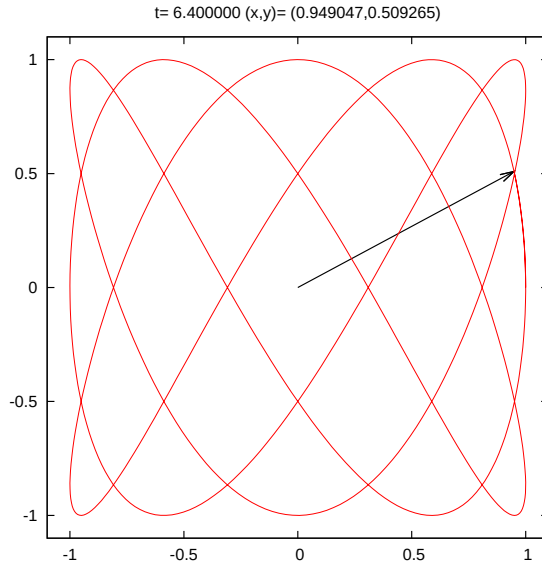
Στο παραπάνω πρόγραμμα έχουμε θέσει $A = 1$. Ο χρήστης εισάγει τις δύο συχνότητες ω_1 και ω_2 και τους αντίστοιχους χρόνους. Εύκολα μπορούμε να το μελετήσουμε με τις εντολές

```
> f77 Lissajous.f -o lsj
> ./lsj
# Enter omega1 and omega2:
3 5
# Enter tf,dt:
10.0 0.01
# o1= 3. o2= 5.
# t0= 0. tf= 10. dt= 0.009999999978
# T1= 2.09439516 T2= 1.2566371
>gnuplot
gnuplot> plot "Lissajous.dat" using 1:2 with lines title "x(t)"
gnuplot> replot "Lissajous.dat" using 1:3 with lines title "y(t)"
gnuplot> plot "Lissajous.dat" using 1:4 with lines title "v_x(t)"
gnuplot> replot "Lissajous.dat" using 1:5 with lines title "v_y(t)"
gnuplot> plot "Lissajous.dat" using 2:3 with lines title "x-y for 3:5"
gnuplot> file = "Lissajous.dat"
gnuplot> set xrange [-1.1:1.1];set yrange [-1.1:1.1]
gnuplot> t0=0;tf=10;dt=0.1
gnuplot> load "animate2D.gnu"
```

Στο Σχήμα 2.11 δείχνουμε το αποτέλεσμα για την τροχιά του ασύμμετρου αρμονικού ταλαντωτή με $\omega_1 = 3$ και $\omega_2 = 5$.

2.2 Κίνηση στο Χώρο

Στην παράγραφο αυτή θα κάνουμε μια απλή γενίκευση των μεθόδων που περιγράψαμε στις προηγούμενες παραγράφους για να μελετήσουμε την κίνηση ενός υλικού σωματιδίου στο χώρο. Οι μόνες αλλαγές θα είναι η προσθήκη μιας ακόμια εξίσωσης για τη συντεταγμένη $z(t)$ και τη συνιστώσα της ταχύτητας $v_z(t)$ και η μέθοδος απεικόνισης των τροχιών στο **gnuplot**. Τα προγράμματα που θα γράψουμε θα έχουν παρόμοια δομή με αυτά που γράψαμε για την κίνηση σωματιδίου στο επίπεδο.



Σχήμα 2.11: Οι τροχιά του ασύμμετρου αρμονικού ταλαντωτή με $\omega_1 = 3$ και $\omega_2 = 5$.

Το πρώτο παράδειγμα που θα εξετάσουμε είναι το κωνικό εκκρεμές του Σχήματος 2.12. Αυτό κινείται στο επίπεδο xy με σταθερή γωνιακή ταχύτητα ω . Οι εξισώσεις κίνησης δίνονται από τις σχέσεις

$$T_z = T \cos \theta = mg \quad T_{xy} = T \sin \theta = m\omega^2 r, \quad (2.13)$$

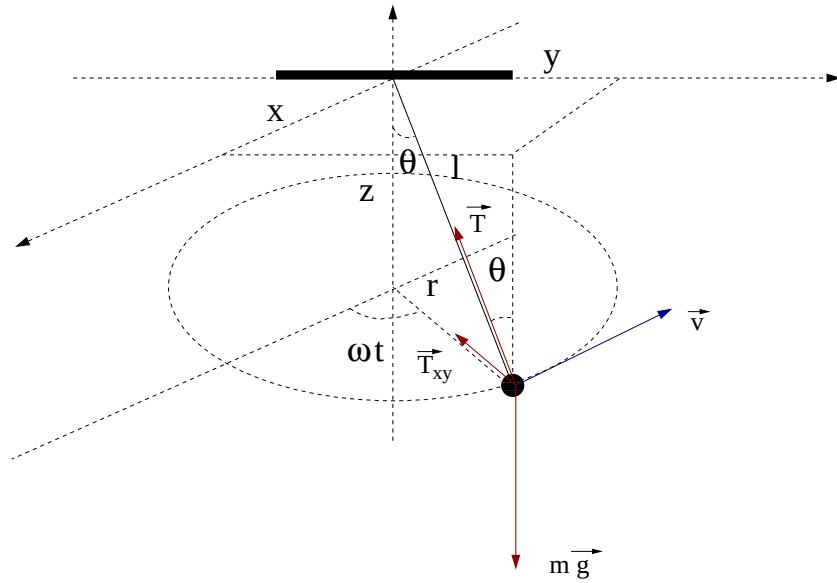
όπου φυσικά $r = l \sin \theta$. Λύνοντας τις παραπάνω εξισώσεις¹³ βρίσκουμε:

$$\begin{aligned} x(t) &= r \cos \omega t \\ y(t) &= r \sin \omega t \\ z(t) &= -l \cos \theta \end{aligned} \quad (2.14)$$

όπου στις παραπάνω αντικαθιστούμε τις τιμές

$$\begin{aligned} \cos \theta &= \frac{g}{\omega^2 l} \\ \sin \theta &= \sqrt{1 - \cos^2 \theta} \\ r &= \frac{g \sin \theta}{\omega^2 \cos \theta} \end{aligned} \quad (2.15)$$

¹³Φυσικά επιλέγοντας τις κατάλληλες αρχικές συνθήκες (γράψτε τις...).



Σχήμα 2.12: Το κωνικό εκκρεμές του προγράμματος ConicalPendulum.f.

Για τις ταχύτητες έχουμε

$$\begin{aligned} v_x &= -r\omega \sin \omega t \\ v_y &= r\omega \cos \omega t \\ v_z &= 0 \end{aligned} \quad (2.16)$$

Από τα παραπάνω προκύπτει ότι

$$\omega \geq \omega_{\min} = \sqrt{\frac{g}{l}}, \quad (2.17)$$

και ότι καθώς $\omega \rightarrow \infty$, $\theta \rightarrow \pi/2$.

Στο πρόγραμμα που θα γράψουμε ο χρήστης δίνει τις παραμέτρους l και ω και τους χρόνους t_f και βήμα χρόνου δt (παίρνουμε $t_0 = 0$). Η σύμβαση για την έξοδο των δεδομένων από το πρόγραμμα είναι ότι γράφονται σε ένα αρχείο, μία γραμμή ανά χρονική στιγμή, όπου οι 7 πρώτες στήλες είναι οι τιμές των t , x , y , z , v_x , v_y και v_z . Επειδή η γραμμή είναι μεγάλη και δε θέλουμε να τη σπάσει η Fortran, δίνουμε την κατάλληλη εντολή **FORMAT**. Δείτε τη συζήτηση στη σελίδα 52. Το πρόγραμμα παρατίθεται παρακάτω:

```

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   File ConicalPendulum.f
C   Set pendulum angular velocity omega and display motion in 3D
C   -----
C   program ConicalPendulum
C   implicit none
C   -----
C   Declaration of variables
C   real l,g,r,x,y,z,vx,vy,vz,t,tf,dt,PI
C   real theta,cos_theta,sin_theta,omega
C   parameter(PI=3.1415927,g=9.81)
C   -----
C   Ask user for input:
C   print *,'# Enter l,omega: '
C   read(5,*)l,omega
C   print *,'# Enter tf,dt:'
C   read(5,*)tf,dt
C   print *,'# l= ',l, ', omega= ',omega
C   print *,'# T= ',2.0*PI/omega,' omega_min= ',sqrt(g/l)
C   print *,'# t0= ',0.0,' tf= ',tf,' dt= ',dt
C   -----
C   Initialize
C   cos_theta = g/(omega*omega*l)
C   if( cos_theta .ge. 1) stop 'cos(theta)>= 1'
C   sin_theta = sqrt(1.0-cos_theta*cos_theta)
C   z = -g/(omega*omega)!they remain constant throughout
C   vz= 0.0 !the motion
C   r = g/(omega*omega)*sin_theta/cos_theta
C   open(unit=11,file='ConicalPendulum.dat')
C   -----
C   Compute:
C   t = 0.0
C   do while(t .le. tf)
C     x = r*cos(omega*t)
C     y = r*sin(omega*t)
C     vx = -r*sin(omega*t)*omega
C     vy = r*cos(omega*t)*omega
C     write(11,100)t,x,y,z,vx,vy,vz

```

```

        t = t + dt
    enddo
    close(11)
100  FORMAT(20G15.7)
    end

```

Για να το μεταγλωττίσουμε και να το τρέξουμε κάνουμε τα γνωστά:

```

> f77 ConicalPendulum.f -o cpd
> ./cpd
# Enter l,omega:
1.0 6.28
# Enter tf,dt:
10.0 0.01
# l=      1. omega=      6.28000021
# T=      1.00050724 omega_min=    3.132092
# t0=     0. tf=     10. dt=     0.00999999978

```

Τα αποτελέσματα θα τα βρούμε στο αρχείο `ConicalPendulum.dat`. Για να δούμε τις γραφικές παραστάσεις των συναρτήσεων $x(t)$, $y(t)$, $z(t)$, $v_x(t)$, $v_y(t)$, $v_z(t)$ εκτελούμε τις γνωστές εντολές μέσα από το `gnuplot`:

```

> gnuplot
gnuplot> plot "ConicalPendulum.dat" u 1:2 with lines title "x(t)"
gnuplot> replot "ConicalPendulum.dat" u 1:3 with lines title "y(t)"
gnuplot> replot "ConicalPendulum.dat" u 1:4 with lines title "z(t)"
gnuplot> plot "ConicalPendulum.dat" u 1:5 with lines title "v_x(t)"
gnuplot> replot "ConicalPendulum.dat" u 1:6 with lines title "v_y(t)"
gnuplot> replot "ConicalPendulum.dat" u 1:7 with lines title "v_z(t)"

```

Το αποτέλεσμα φαίνεται στο Σχήμα 2.13.

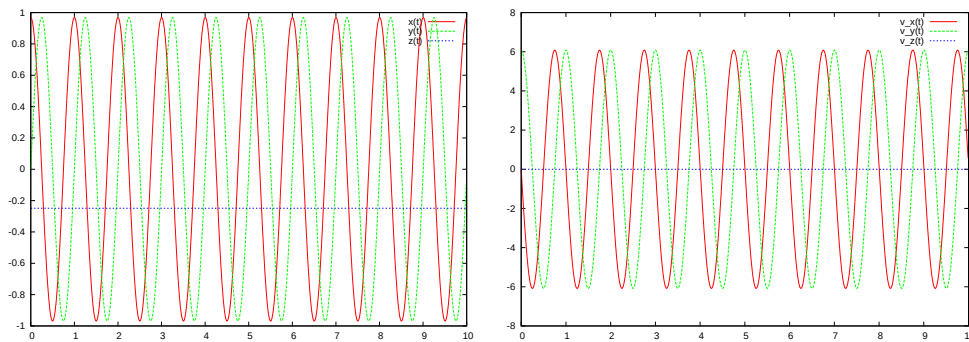
Για να δούμε την τρισδιάστατη τροχιά στο χώρο, θα χρησιμοποιήσουμε την εντολή `splot` στο `gnuplot`:

```

gnuplot> splot "ConicalPendulum.dat" u 2:3:4 with lines title "r(t)"

```

Το αποτέλεσμα φαίνεται στο Σχήμα 2.14. Μπορούμε να κάνουμε κλικ πάνω στην τροχιά και να περιστρέψουμε την καμπύλη ώστε να τη δούμε από διαφορετικές οπτικές γωνίες. Μπορούμε να αλλάξουμε τα όρια στους άξονες ορίζοντάς τα ρητά στην εντολή `splot`:



Σχήμα 2.13: Οι γραφικές παραστάσεις των συναρτήσεων $x(t), y(t), z(t), v_x(t), v_y(t), v_z(t)$ προγράμματος `ConicalPendulum.f` για $\omega = 6.28, l = 1.0$.

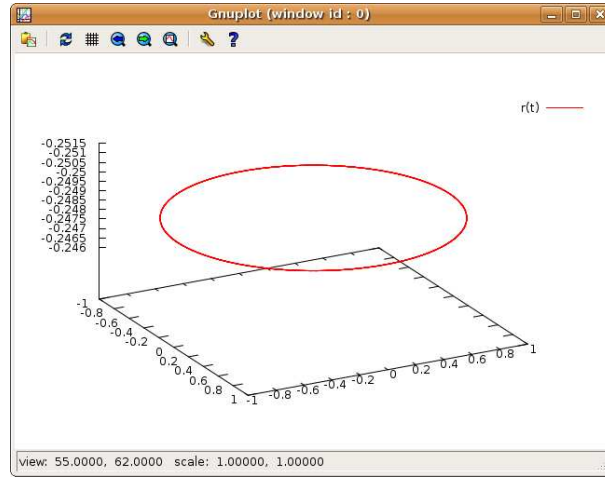
```
gnuplot> splot [-1.1:1.1][-1.1:1.1][-0.3:0.0] "ConicalPendulum.dat" \
          using 2:3:4 with lines title "r(t)"
```

Στο συνοδευτικό λογισμικό περιλαμβάνεται και το αρχείο `animate3D.gnu` με το οποίο μπορούμε να κάνουμε απλά animations της τροχιάς της κίνησης ενός υλικού σημείου στο χώρο. Η εντολές που πρέπει να δώσουμε είναι ανάλογες με αυτές που δίναμε στην περίπτωση του `animate2D.gnu` με τη μόνη διαφορά ότι καλό είναι να ορίσουμε τα όρια και στον άξονα των z . Για να δούμε την τροχιά του κωνικού εκκρεμούς από τα δεδομένα που παρήγαγε παραπάνω, δίνουμε τις εντολές:

```
gnuplot> set xrange [-1.1:1.1];set yrange [-1.1:1.1];set zrange [-0.3:0]
gnuplot> t0=0;tf=10;dt=0.1
gnuplot> load "animate3D.gnu"
```

Το αποτέλεσμα το βλέπουμε στο Σχήμα 2.15. Περισσότερο να πούμε πως το πρόγραμμα `animate3D.gnu` μπορεί να χρησιμοποιηθεί πάνω σε οποιοδήποτε αρχείο που περιέχει δεδομένα της μορφής `t x y z` στις πρώτες τέσσερις στήλες του. Απλά αλλάζουμε την τιμή της μεταβλητής `file` στο όνομα του αρχείου που θα μελετήσουμε. Όπως και με το `animate2D.gnu` μπορούμε να αλλάξουμε μόνο όποιες από τις μεταβλητές `file, t0, tf, dt` είναι αναγκαίο πριν επαναλάβουμε το animation με την εντολή `load "animate3D.gnu"`.

Στη συνέχεια θα μελετήσουμε την τροχιά ενός φορτισμένου σωματιδίου που εισέρχεται σε ένα ομογενές μαγνητικό πεδίο $\vec{B} = B\hat{z}$ τη χρονική στιγμή $t_0 = 0$ από τη θέση $\vec{r}_0 = x_0\hat{x}$ με ταχύτητα $\vec{v}_0 = v_{0y}\hat{y} + v_{0z}\hat{z}$ όπως φαίνεται στο



Σχήμα 2.14: Η γραφική παράσταση της τροχιάς $\vec{r}(t)$ του υλικού σημείου του προγράμματος ConicalPendulum.f για $\omega = 6.28$, $l = 1.0$. Φαίνεται το παράθυρο του gnuplot όπου μπορούμε να κάνουμε κλικ πάνω στην τροχιά και να περιστρέψουμε την καμπύλη ώστε να τη δούμε από διαφορετικές οπτικές γωνίες. Κάτω αριστερά βλέπουμε την οπτική διεύθυνση που δίνεται αντίστοιχα από τις γωνίες $\theta = 55.0$ μοίρες (γωνία με άξονα z) και $\phi = 62$ μοίρες (γωνία με άξονα x), όμοια με τις σφαιρικές συντεταγμένες (θ, ϕ) .

Σχήμα 2.16. Στο φορτίο εξασκείται από το μαγνητικό πεδίο η δύναμη Lorentz $\vec{F} = q(\vec{v} \times \vec{B}) = qBv_y\hat{x} - qBv_x\hat{y}$. Οι εξισώσεις κίνησης είναι

$$\begin{aligned} a_x &= \frac{dv_x}{dt} = \omega v_y & \omega &\equiv \frac{qB}{m} \\ a_y &= \frac{dv_y}{dt} = -\omega v_x. \end{aligned} \quad (2.18)$$

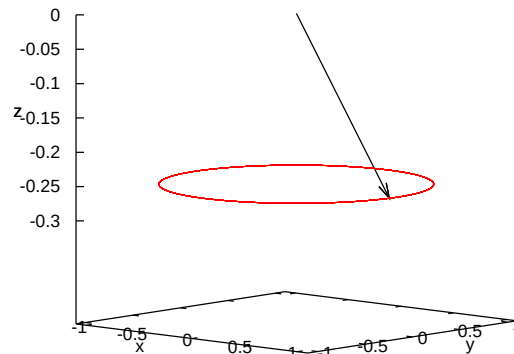
Ολοκληρώνοντας τις παραπάνω εξισώσεις και λαμβάνοντας υπ'οψη τις αρχικές συνθήκες παίρνουμε

$$\begin{aligned} v_x(t) &= v_{0y} \sin \omega t \\ v_y(t) &= v_{0y} \cos \omega t. \end{aligned} \quad (2.19)$$

Με μία ακόμα ολοκλήρωση παίρνουμε τη θέση του σωματιδίου σε συνάρτηση με το χρόνο

$$\begin{aligned} x(t) &= \left(x_0 + \frac{v_{0y}}{\omega}\right) - \frac{v_{0y}}{\omega} \cos \omega t = x_0 \cos \omega t \\ y(t) &= \frac{v_{0y}}{\omega} \sin \omega t = -x_0 \sin \omega t \quad \text{με} \quad x_0 = -\frac{v_{0y}}{\omega}, \end{aligned} \quad (2.20)$$

t= 10.100000 (x,y,z)=(0.964311,-0.090732,-0.248742)

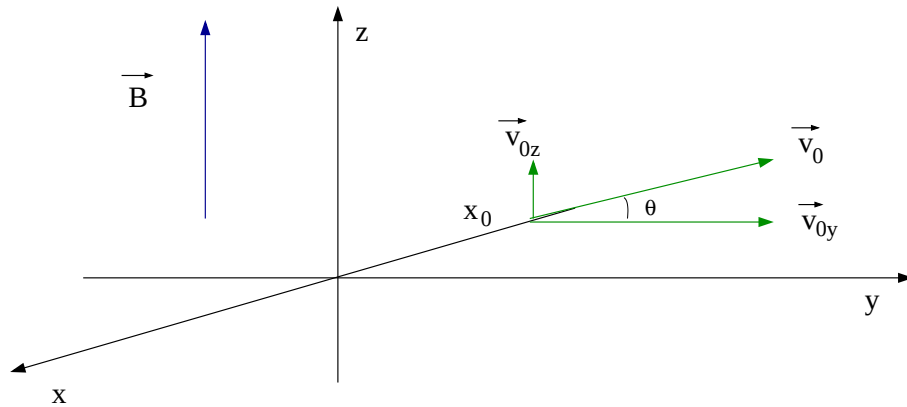


Σχήμα 2.15: Η τροχιά $\vec{r}(t)$ του υλικού σημείου του προγράμματος `ConicalPendulum.f` για $\omega = 6.28$, $l = 1.0$ όπως φαίνεται με το πρόγραμμα `animate3D.gnu`. Στον τίτλο βλέπουμε την τρέχουσα χρονική στιγμή και τις συντεταγμένες του υλικού σημείου.

όπου κάναμε την επιλογή $x_0 = -v_{0y}/\omega$ για να κάνουμε το κέντρο της κυκλικής τροχιάς να συμπίπτει με την αρχή των αξόνων. Το γεωμετρικό σχήμα της τροχιάς είναι μια σπείρα με ακτίνα $R = -x_0$ και βήμα $v_{0z}T = 2\pi v_{0z}/\omega$.

Με τα παραπάνω είναι εύκολο τώρα να γράψουμε ένα πρόγραμμα που να υπολογίζει την τροχιά του παραπάνω φορτίου. Οι παράμετροι που θα εισάγει ο χρήστης είναι το μέτρο της ταχύτητας v_0 , τη γωνία θ σε μοίρες (βλ. Σχήμα 2.16) και τη συχνότητα ω . Προφανώς θα έχουμε $v_{0y} = v_0 \cos \theta$ και $v_{0z} = v_0 \sin \theta$. Η αρχική θέση υπολογίζεται τότε από την $x_0 = -v_{0y}/\omega$. Το πρόγραμμα δίνεται αυτούσιο παρακάτω και μπορείτε να το βρείτε στο αρχείο `ChargeInB.f` στο συνοδευτικό λογισμικό:

```
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      File ChargeInB.f
C      A charged particle of mass m and charge q enters a magnetic
C      field B in +z direction. It enters with velocity
C      v0x=0,v0y=v0 cos(theta),v0z=v0 sin(theta), 0<=theta<pi/2
```



Σχήμα 2.16: Σωματίδιο τη χρονική στιγμή $t_0 = 0$ στη θέση $\vec{r}_0 = x_0 \hat{x}$ με ταχύτητα $\vec{v}_0 = v_{0y} \hat{y} + v_{0z} \hat{z}$ μέσα σε ομογενές μαγνητικό πεδίο $\vec{B} = B \hat{z}$.

```

C      at the position x0=-v0y/omega, omega=q B/m
C
C      Enter v0 and theta and see trajectory from
C      t0=0 to tf at step dt
C      -----
C      program ChargeInB
C      implicit none
C      -----
C      Declaration of variables
C      real x,y,z,vx,vy,vz,t,tf,dt,PI
C      real x0,y0,z0,v0x,v0y,v0z,v0
C      real theta,omega
C      parameter(PI=3.1415927)
C      -----
C      Ask user for input:
C      print *,'# Enter omega: '
C      read(5,*)omega
C      print *,'# Enter v0, theta (degrees):'
C      read(5,*)v0,theta
C      print *,'# Enter tf,dt:'
C      read(5,*)tf,dt
C      print *,'# omega= ',omega ,' T= ',2.0*PI/omega

```



```

      print *,'# v0=      ',v0,      ' theta= ',theta,'o (degrees)'
      print *,'# t0=      ',0.0,      ' tf=      ',tf,' dt= ',dt
C -----
C Initialize
      if(theta.lt.0.0 .or. theta.ge.90.0)stop 'Illegal 0<theta<90'
      theta = (PI/180.0)*theta !convert to radians
      v0y   = v0*cos(theta)
      v0z   = v0*sin(theta)
      print *,'# v0x= ',0.0,' v0y= ',v0y,' v0z= ',v0z
      x0    = - v0y/omega
      print *,'# x0= ',x0, ' y0= ',0.0,' z0= ',0.0
      print *,'# xy plane: Circle with center (0,0) and R= ',ABS(x0)
      print *,'# step of helix: s=v0z*T= ',v0z*2.0*PI/omega
      open(unit=11,file='ChargeInB.dat')
C -----
C Compute:
      t     = 0.0
      vz    = v0z
      do while(t .le. tf)
        x   = x0*cos(omega*t)
        y   = -x0*sin(omega*t)
        z   = v0z*t
        vx  = v0y*sin(omega*t)
        vy  = v0y*cos(omega*t)
        write(11,100)t,x,y,z,vx,vy,vz
        t   = t + dt
      enddo
      close(11)
100  FORMAT(20G15.7)
      end

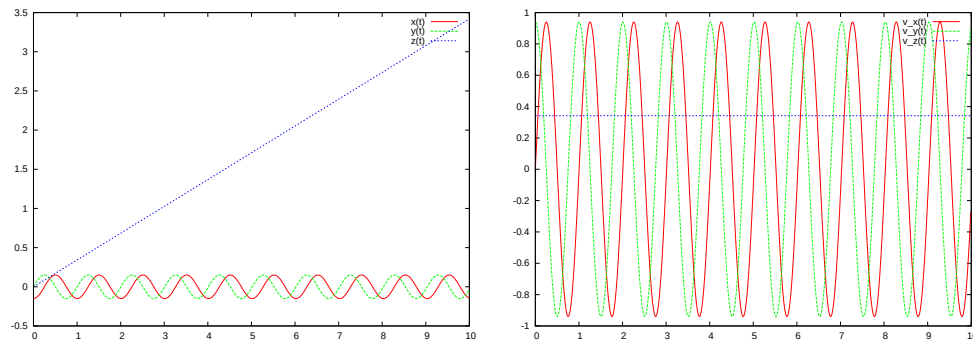
```

Παραθέτουμε εδώ τις εντολές μιας τυπικής συνεδρίας τα αποτελέσματα της οποίας δείχνονται στο Σχήμα 2.17 και στο Σχήμα 2.18.

```

> f77 ChargeInB.f -o chg
> ./chg
# Enter omega:
6.28

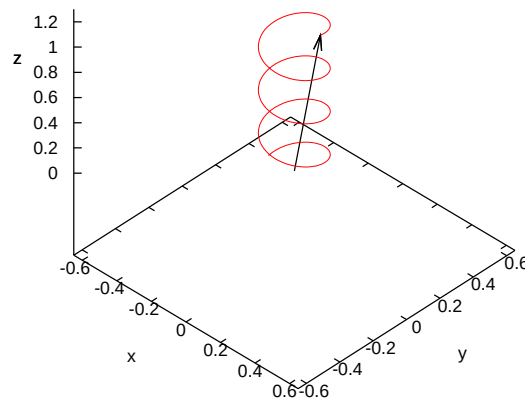
```



Σχήμα 2.17: Οι γραφικές παραστάσεις των συναρτήσεων $x(t)$, $y(t)$, $z(t)$, $v_x(t)$, $v_y(t)$, $v_z(t)$ του προγράμματος ChargeInB.f για $\omega = 6.28$, $x_0 = 1.0$, $\theta = 20$ μοίρες.

```
# Enter v0, theta (degrees):
1.0 20
# Enter tf,dt:
10 0.01
# omega=      6.28000021 T=      1.00050724
# v0=         1. theta=    20.o (degrees)
# t0=         0. tf=      10. dt=    0.00999999978
# v0x=    0. v0y=    0.939692616 v0z=    0.342020124
# x0=   -0.149632573 y0=    0. z0=    0.
# xy plane: Circle with center (0,0) and R=    0.149632573
# step of helix: s=v0z*T=    0.342193604
> gnuplot
gnuplot> plot "ChargeInB.dat" u 1:2 with lines title "x(t)"
gnuplot> replot "ChargeInB.dat" u 1:3 with lines title "y(t)"
gnuplot> replot "ChargeInB.dat" u 1:4 with lines title "z(t)"
gnuplot> plot "ChargeInB.dat" u 1:5 with lines title "v_x(t)"
gnuplot> replot "ChargeInB.dat" u 1:6 with lines title "v_y(t)"
gnuplot> replot "ChargeInB.dat" u 1:7 with lines title "v_z(t)"
gnuplot> splot "ChargeInB.dat" u 2:3:4 with lines title "r(t)"
gnuplot> file = "ChargeInB.dat"
gnuplot> set xrange [-0.65:0.65];set yrange [-0.65:0.65];\
set zrange [0:1.3]
gnuplot> t0=0;tf=3.5;dt=0.1
gnuplot> load "animate3D.gnu"
```

t= 3.500000 (x,y,z)=(0.149623,0.001671,1.197069)



Σχήμα 2.18: Η τροχιά $\vec{r}(t)$ του υλικού σημείου του προγράμματος `ChargeInB.f` για $\omega = 6.28$, $v_0 = 1.0$, $\theta = 20$ μοίρες όπως φαίνεται με το πρόγραμμα `animate3D.gnu`. Στον τίτλο βλέπουμε την τρέχουσα χρονική στιγμή και τις συντεταγμένες του υλικού σημείου.

2.3 Κίνηση μετ' Εμποδίων

Στην παράγραφο αυτή θα μελετήσουμε την κίνηση ενός σωματιδίου η οποία είναι απλή και συνεχής σε πεπερασμένα διαστήματα χρόνου, αλλά σε συγκεκριμένες χρονικές στιγμές υπόκειται σε ελαστική κρούση σε ένα τοίχωμα αμετακίνητο και αδιαπέραστο. Η κρούση αυτή θα δούμε ότι περιγράφεται προσεγγιστικά από τους αλγόριθμους που θα προγραμματίσουμε. Έτσι θα μελετήσουμε για πρώτη φορά τα συστηματικά σφάλματα που εισάγονται από τους αλγόριθμους που χρησιμοποιούμε¹⁴ όταν φτιάχνουμε ένα μοντέλο για το σύστημα που μελετάμε.

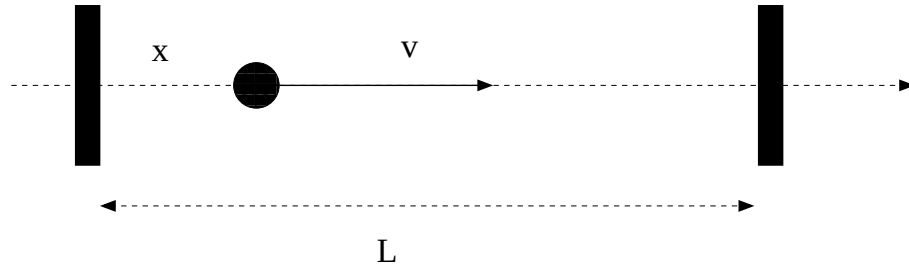
¹⁴Και στις προηγούμενες παραγράφους υπήρχε (μικρό) συστηματικό σφάλμα στα συστήματα που μελετήσαμε, το οποίο οφειλόταν όμως στα σφάλματα των αριθμητικών πράξεων που εκτελούσε ο υπολογιστής - αναπαράσταση πραγματικών, ακρίβεια πράξεων κινητής υποδιαστολής κλπ. Οι αλγόριθμοι ήταν 'ακριβείς'.

2.3.1 Το Μονοδιάστατο Κουτί

Το πιο απλό παράδειγμα που θα μελετήσουμε είναι η κίνηση ενός σωματιδίου σε ένα ‘μονοδιάστατο κουτί’. Το σωματίδιο κινείται ελεύθερα πάνω στον άξονα των x στο διάστημα $0 < x < L$ όπως φαίνεται στο Σχήμα 2.19. Όταν φτάνει στα άκρα του διαστήματος, ανακλάται στα τοιχώματα και η ταχύτητά του αναστρέφεται. Η δυναμική του ενέργεια είναι

$$V(x) = \begin{cases} 0 & 0 < x < L \\ +\infty & \text{αλλού} \end{cases}, \quad (2.21)$$

που έχει σχήμα ενός απειρόβαθου πηγαδιού δυναμικού. Η δύναμη $F = -dV(x)/dx = 0$ μέσα στο κουτί, όπου το σωματίδιο κινείται ελεύθερα, και $F = \pm\infty$ στα τοιχώματα όπου ανακλάται.



Σχήμα 2.19: Σωματίδιο σε μονοδιάστατο κουτί με τοιχώματα στο $x = 0$ και $x = L$.

Το σύστημα είναι πολύ απλό. Αρκεί να δώσουμε την αρχική θέση του σωματιδίου x_0 και την αρχική του ταχύτητα v_0 (το πρόσημό της δηλώνει και την αρχική κατεύθυνση κίνησης). Όσο δε συγκρούεται με τα τοιχώματα η κίνηση είναι ευθύγραμμη και ομαλή και ισχύουν οι απλές σχέσεις

$$\begin{aligned} x(t) &= x_0 + v_0 t \\ v(t) &= v_0. \end{aligned} \quad (2.22)$$

Επίσης για οποιαδήποτε μεταβολή του χρόνου δt , έτσι ώστε να μην υπάρχει σύγκρουση με τα τοιχώματα μέσα στο χρονικό διάστημα $(t, t + \delta t)$, ισχύει (ακριβώς, όχι προσεγγιστικά)

$$\begin{aligned} x(t + \delta t) &= x(t) + v(t)\delta t \\ v(t + \delta t) &= v(t). \end{aligned} \quad (2.23)$$

Θα μπορούσαμε λοιπόν να χρησιμοποιήσουμε τις παραπάνω σχέσεις για να γράψουμε το πρόγραμμά μας και όταν το σωματίο συγκρούεται με κάποιο από τα τοιχώματα να αντιστρέψουμε την ταχύτητα: $v(t) \rightarrow -v(t)$. Η πηγή των δυσκολιών μας κρύβεται στη λέξη ‘όταν’. Αφού στον υπολογιστή το χρονικό διάστημα δt είναι αναγκαστικά πεπερασμένο δεν μπορούμε να εντοπίσουμε τη χρονική στιγμή της σύγκρουσης με ακρίβεια μεγαλύτερη από δt : Αν λ.χ. χρησιμοποιώντας τις Σχέσεις (2.23) το σωματίδιο τη χρονική στιγμή $t + \delta t$ ξεπεράσει το τοίχωμα η κρούση θα μπορούσε να ‘έχει συμβεί οποιαδήποτε χρονική στιγμή μέσα στο διάστημα $(t, t + \delta t)$. Ο αλγόριθμος όμως, θα αλλάξει τη φορά της ταχύτητας τη χρονική στιγμή $t + \delta t$ εισάγοντας ένα συστηματικό σφάλμα στον υπολογισμό. Πιο συγκεκριμένα, αυτό μπορεί να γίνει με το βρόχο

```
do while(t .le. tf)
  x = x + v * dt
  t = t + dt
  if(x .lt. 0.0 .or. x .gt. L) v = -v
enddo
```

όπου η συνθήκη σύγκρουσης δίνεται στην προτελευταία γραμμή με τη λογική πρόταση ‘Αν x μικρότερο του 0 ή το x μεγαλύτερο του L ’ όπου φαίνεται η αδυναμία του αλγόριθμου να ελέγξει την ακριβή χρονική στιγμή της κρούσης.

Ας δούμε τώρα ολόκληρο το πρόγραμμα το οποίο βρίσκεται στο αρχείο `box1D_1.f` του συνοδευτικού λογισμικού. Ο χρήστης μπορεί να καθορίσει το μήκος του κουτιού L και τις αρχικές συνθήκες x_0 , v_0 . Δίνει τον αρχικό και τελικό χρόνο της κίνησης t_0 , t_f καθώς και το βήμα υπολογισμού dt και ο υπολογισμός ... ξεκινάει:

```
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      File box1D_1.f
C      Motion of a free particle in a box  0<x<L
C      Use integration with time step dt: x = x + v*dt
C      -----
C      program box1D
C      implicit none
C      -----
C      Declaration of variables
C      real  L,x0,v0,t0,tf,dt,t,x,v
C      -----
C      Ask user for input:
```

```

print *,'# Enter L:'
read(5,*)L
print *,'# L = ',L
if( L .le. 0.0) stop 'L must be positive.'
print *,'# Enter x0,v0:'
read(5,*)x0,v0
print *,'# x0= ',x0,' v0= ',v0
if(x0 .lt. 0.0 .or. x0 .gt. L) stop 'illegal value of x0.'
if(v0 .eq. 0.0 ) stop 'illegal value of v0 = 0.'
print *,'# Enter t0,tf,dt:'
read(5,*)t0,tf,dt
print *,'# t0= ',t0,' tf= ',tf,' dt= ',dt
C -----
C Initialize
t = t0
x = x0
v = v0
open(unit=11,file='box1D_1.dat')
C -----
C Compute:
do while(t .le. tf)
  write(11,*)t,x,v
  x = x + v*dt
  t = t + dt
  if(x .lt. 0.0 .or. x .gt. L) v = -v
enddo
close(11)
end

```

Τα δεδομένα τα βρίσκουμε σε τρεις στήλες στο αρχείο `box1D_1.dat`. Η μεταγλώττιση και το τρέξιμο γίνεται κατά τα γνωστά, ενώ η γραφική παράσταση της τροχιάς με το `gnuplot`:

```

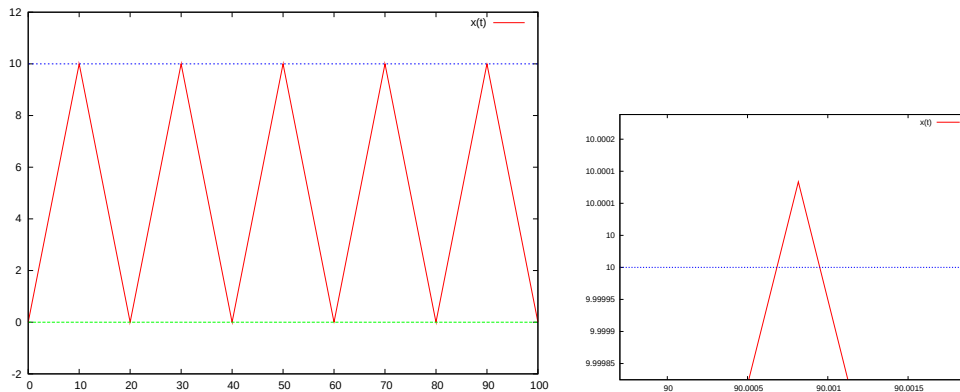
> f77 box1D_1.f -o box1
> ./box1
# Enter L:
10
# L = 10.
# Enter x0,v0:

```

```

0 1.0
# x0= 0. v0= 1.
# Enter t0,tf,dt:
0 100 0.01
# t0= 0. tf= 100. dt= 0.009999999978
> gnuplot
gnuplot> plot "box1D_1.dat" using 1:2 with lines title "x(t)",\
                                0 notitle,L notitle
gnuplot> plot [:][-1.2:1.2] "box1D_1.dat" \
                                using 1:3 with lines title "v(t)"

```



Σχήμα 2.20: Η τροχιά $x(t)$ σωματίδιου σε μονοδιάστατο κουτί με $L = 10$, $x_0 = 0.0$, $v_0 = 1.0$, $\delta t = 0.01$. Στο δεξί διάγραμμα μεγενθύνουμε λεπτομέρεια όταν $t \approx 90$. Φαίνονται τα συστηματικά σφάλματα στον προσδιορισμό της χρονικής στιγμής της κρούσης $t_k = 90$ και της μέγιστης τιμής της $x(t)$, $x_m = L = 10.0$.

Τα αποτελέσματα για την τροχιά $x(t)$ δείχνονται στο Σχήμα 2.20. Είναι φανερό η επίδραση του συστηματικού σφάλματος στα αποτελέσματα. Με τις απλές αρχικές συνθήκες που επιλέξαμε, οι κρούσεις συμβαίνουν κάθε $T/2 = L/v = 10$ μονάδες χρόνου, άρα στο κομμάτι του διαγράμματος που μεγενθύνουμε στο αριστερό του σχήματος 2.20, η αντιστροφή της κίνησης θα έπρεπε να είχε συμβεί όταν $t = 90$, $x = L = 10$.

Ο αναγνώστης θα έχει ήδη καταλάβει ότι το παραπάνω πρόβλημα λύνεται παίρνοντας δt αρκετά μικρό, οπότε μπορούμε αν έχουμε την απαραίτητη υπολογιστική δύναμη να κάνουμε το σφάλμα όσο μικρό θέλουμε. Ναι, αυτό μέχρι ένα σημείο είναι σωστό. Προσέξτε όμως μια αδυναμία στον αλγόριθμο που

επιλέξαμε: Η επόμενη θέση καθορίζεται από την πράξη $x+v*dt$. Τι θα γίνει αν παίρνοντας το dt αρκετά μικρό, το γινόμενο $v*dt$ γίνει περισσότερο από επτά τάξεις μεγέθους μικρότερο από την τρέχουσα τιμή του x ; Αφού το x αναπαρίσταται από μεταβλητή τύπου **REAL**, η ακρίβεια στην αναπαράστασή του είναι περίπου επτά δεκαδικά ψηφία. Άρα η πράξη $x+v*dt$ θα μας δώσει πίσω το x , άρα ο αλγόριθμός μας θα ‘κολλήσει’ και το κινητό θα μένει στο ίδιο σημείο¹⁵. Αυτό όσο και να προσπαθούμε δε διορθώνεται¹⁶. Η μόνη μας ελπίδα είναι να επινοήσουμε έναν καλύτερο αλγόριθμο. Λ.χ θεωρήστε τη σχέση που δίνει τη θέση του κινητού στην ευθύγραμμη ομαλή κίνηση:

$$x(t) = x_0 + v_0(t - t_0). \quad (2.24)$$

Ας χρησιμοποιήσουμε την παραπάνω σχέση για τα τμήματα της κίνησης μεταξύ των συγκρούσεων. Το μόνο που έχουμε να κάνουμε είναι σε κάθε σύγκρουση με ένα από τα τοιχώματα να αντιστρέφουμε τη v_0 , να ορίζουμε το x_0 να είναι η τρέχουσα θέση του κινητού και t_0 να είναι η τρέχουσα χρονική στιγμή. Αυτό επιτυγχάνεται με το βρόχο:

```
t = t0
do while(t .le. tf)
  x = x0 + v0*(t-t0)
  if( x .lt. 0.0 .or. x .gt. L)then
    x0 = x
    t0 = t
    v0 = -v0
  endif
  t = t + dt
enddo
```

Στον παραπάνω αλγόριθμο δε γλιτώνουμε από το σφάλμα προσδιορισμού της στιγμής της κρούσης αλλά δεν έχουμε το πρόβλημα ‘αστάθειας’ που είχε ο προηγούμενος όταν $dt \rightarrow 0$. Παρακάτω δίνουμε το πρόγραμμα που χρησιμοποιεί τον παραπάνω αλγόριθμο που μπορείτε να βρείτε στο συνοδευτικό λογισμικό στο αρχείο `box1D_2.f`:

¹⁵Και η σχέση $t=t+dt$ πάσχει από ανάλογο πρόβλημα, αλλά αυτό λύνεται εύκολα (δες άσκηση).

¹⁶Φυσικά θα μπορούσαμε να διαλέξουμε μεταβλητές **REAL*8**, **REAL*16**, Αυτό μπορεί να μας λύσει ένα συγκεκριμένο πρόβλημα για δεδομένο **tf**, αλλά ανάλογο πρόβλημα ακρίβειας θα συναντήσουμε αν υπολογίσουμε την τροχιά για (αρκετά) μεγαλύτερο **tf**.


```

C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   File box1D_2.f
C   Motion of a free particle in a box  $0 < x < L$ 
C   Use constant velocity equation:  $x = x_0 + v_0 \cdot (t - t_0)$ 
C   Reverse velocity and redefine  $x_0, t_0$  on boundaries
C   -----
C   program box1D
C   implicit none
C   -----
C   Declaration of variables
C   real L,x0,v0,t0,tf,dt,t,x,v
C   -----
C   Ask user for input:
C   print *,'# Enter L:'
C   read(5,*)L
C   print *,'# L = ',L
C   if( L .le. 0.0) stop 'L must be positive.'
C   print *,'# Enter x0,v0:'
C   read(5,*)x0,v0
C   print *,'# x0= ',x0,' v0= ',v0
C   if(x0 .lt. 0.0 .or. x0 .gt. L) stop 'illegal value of x0.'
C   if(v0 .eq. 0.0 ) stop 'illegal value of v0 = 0.'
C   print *,'# Enter t0,tf,dt:'
C   read(5,*)t0,tf,dt
C   print *,'# t0= ',t0,' tf= ',tf,' dt= ',dt
C   -----
C   Initialize
C   t = t0
C   open(unit=11,file='box1D_2.dat')
C   -----
C   Compute:
C   do while(t .le. tf)
C     x = x0 + v0*(t-t0)
C     write(11,*)t,x,v0
C     if( x .lt. 0.0 .or. x .gt. L)then
C       x0 = x
C       t0 = t
C       v0 = -v0

```



```

C      Motion of a free particle in a box   $0 < x < L$ 
C      Computation of analytical solution
C      -----
C      program box1D
C      implicit none
C      -----
C      Declaration of variables
C      real  L,x0,v0,t0,tf,dt,t,x,v,Th,a
C      -----
C      Ask user for input:
C      print *,'# Enter L:'
C      read(5,*)L
C      print *,'# L = ',L
C      if( L .le. 0.0) stop 'L must be positive.'
C      print *,'# Enter x0,v0:'
C      read(5,*) a,v0 !notice, x0=a is the original position
C      print *,'# x0= ',a ,' v0= ',v0
C      if(a .lt. 0.0 .or. a .gt. L) stop 'illegal value of x0.'
C      if(v0 .eq. 0.0 ) stop 'illegal value of v0 = 0.'
C      print *,'# Enter t0,tf,dt:'
C      read(5,*)t0,tf,dt
C      print *,'# t0= ',t0,' tf= ',tf,' dt= ',dt
C      -----
C      Initialize
C      Th = L/ABS(v0) !Th = T/2 = half period
C      t  = t0
C      x  = a
C      v  = v0
C      if( v0 .gt. 0.0 ) then
C          t0 = -a/v0
C          x0 = 0.0
C      else
C          t0 = (L-a)/v0
C          x0 = L
C      endif
C      open(unit=11,file='box1D_3.dat')
C      -----
C      Compute:

```

```

do while(t .le. tf)
  write(11,*)t,x,v
  t = t + dt
  if( (t-t0) .gt. Th )then
    t0 = t0 + Th
    v = -v
    if( v .gt. 0.0)then
      x0 = 0.0
    else
      x0 = L
    endif !if( v .gt. 0.0)
  endif !if( (t-t0) .gt. Th )
  x = x0 + v*(t-t0)
enddo !do while(t .le. tf)
close(11)
end

```

Σας το αφήνουμε για άσκηση να επιβεβαιώσετε πως ο παραπάνω αλγόριθμος αντιστοιχεί στην πλήρη αναλυτική λύση του προβλήματος.

2.3.2 Σφάλματα

Ας δούμε με λίγο περισσότερη λεπτομέρεια την επίδραση των συστηματικών σφαλμάτων στα αποτελέσματα που πήραμε από τη μελέτη του απλού προβλήματος που περιγράψαμε στην προηγούμενη παράγραφο. Χρησιμοποιήσαμε δύο αλγόριθμους, ο πρώτος υλοποιήθηκε στο πρόγραμμα που γράψαμε στο αρχείο `box1D_1.f` και ο δεύτερος στο `box1D_2.f`. Θα αναφερόμαστε σε αυτούς στην παράγραφο αυτή ως η ‘μέθοδος 1’ και ‘μέθοδος 2’ αντίστοιχα. Στο αρχείο `box1D_3.f` προγραμματίσαμε την ακριβή λύση της κίνησης του υλικού σημείου στο μονοδιάστατο κουτί.

Τα σφάλματα που θα δούμε μπορούμε να τα χωρίσουμε σε δύο κατηγορίες. Σε αυτά που οφείλονται στα συστηματικά σφάλματα στην προσέγγιση της κίνησης και σε αυτά που οφείλονται στην πεπερασμένη προσέγγιση των πραγματικών αριθμών και των πράξεων ανάμεσά τους από τον υπολογιστή. Θα δώσουμε έμφαση στην πρώτη κατηγορία γιατί αυτή είναι συνήθως αυτή που επικρατεί σε πολλά ενδιαφέροντα αριθμητικά προβλήματα στις φυσικές επιστήμες.

Το σφάλμα στις μεθόδους 1 και 2 οφείλεται στη διακριτοποίηση του χρόνου. Αυτό έχει σαν αποτέλεσμα οι αλγόριθμοι να μη βλέπουν την ακριβή χρονική

στιγμή της κρούσης. Είναι λοιπόν αναμενόμενο το σφάλμα αυτό να ελαττώνεται όταν η διακριτοποίηση του χρόνου γίνεται ολοένα και λεπτότερη.

Όταν υπολογίζουμε μια ποσότητα με μία από τις μεθόδους αυτές, λ.χ. τη θέση του κινητού μια συγκεκριμένη χρονική στιγμή, το αποτέλεσμα που παίρνουμε δεν έχουμε ιδέα αν είναι ορθό και πόση είναι η ακρίβεια προσδιορισμού του. Η πρώτη απόπειρα στην προσπάθεια ελέγχου αυτού το προβλήματος είναι να μελετήσουμε το αποτέλεσμα που πήραμε πόσο εξαρτάται από την παράμετρο που καθορίζει τη λεπτότητα της διακριτοποίησης του χρόνου: Αν καταφέρουμε να δείξουμε ότι το αποτέλεσμά μας συγκλίνει με ανεκτή ακρίβεια σε ένα όριο καθώς $\delta t \rightarrow 0$, αποκτάμε αυτοπεποίθηση πως η τιμή αυτή είναι σωστή στη δεδομένη ακρίβεια.

Δυστυχώς η μέθοδος αυτή δε δουλεύει αλγοριθμικά: Καθώς $\delta t \rightarrow 0$, το σφάλμα διακριτοποίησης του χρόνου ελαττώνεται αλλά άλλα σφάλματα μπορεί να γίνονται σημαντικά. Αναφέραμε στην προηγούμενη παράγραφο ότι η συσσώρευση σφαλμάτων στην απεικόνιση και στις πράξεις μεταξύ των πραγματικών αριθμών γίνεται σημαντικότερη στο όριο αυτό. Άρα πρέπει η μελέτη αυτή να γίνει προσεκτικά.

Αν έχουμε τη δυνατότητα να μελετήσουμε το πρόβλημα με ένα διαφορετικό αλγόριθμο, η σύγκλιση των αποτελεσμάτων των διαφορετικών μεθόδων συνεισφέρει στη βεβαιότητα της ορθότητας των υπολογισμών. Σε αντίθεση με ένα τυπικό πρόβλημα που έχουμε να λύσουμε, στο συγκεκριμένο έχουμε την ‘πολυτέλεια’ να συγκρίνουμε τα αποτελέσματά μας και με την ακριβή λύση του προβλήματος. Σε τέτοιες περιπτώσεις έχουμε τη δυνατότητα να αντιστρέψουμε το πρόβλημα και να χρησιμοποιήσουμε την ακριβή λύση για να μελετήσουμε τη σχετική ακρίβεια και αποτελεσματικότητα διαφορετικών προσεγγιστικών μεθόδων.

Ας δοκιμάσουμε να δούμε πώς μπορούμε να κάνουμε την ανάλυση της σύγκλισης των αποτελεσμάτων καθώς $\delta t \rightarrow 0$. Για κάθε μέθοδο θα καθορίσουμε όλες τις παραμέτρους εκτός από το δt . Στην ανάλυση παρακάτω θα χρησιμοποιήσουμε $L = 10$, $v_0 = 1.0$, $x_0 = 0.0$, $t_0 = 0.0$, $t_f = 95.0$, άρα το υλικό σημείο θα πρέπει να κάνει μία κρούση ανά 10 μονάδες χρόνου. Θα πάρουμε διαδοχικά μικρότερες τιμές του δt και θα υπολογίσουμε την τελική θέση του κινητού $x(t \approx 95)^{18}$ σε συνάρτηση του δt . Θα μελετήσουμε αν, πόσο καλά και πόσο γρήγορα συγκλίνει αυτή η τιμή σε ένα όριο καθώς $\delta t \rightarrow 0^{19}$.

Η ανάλυση αποτελείται από πολλές επαναλαμβανόμενες εντολές: Η μετα-

¹⁸Προσέξτε το $\approx$!

¹⁹Φυσικά εδώ γνωρίζουμε ότι θα πρέπει να πάρουμε $x(95) = 5$.

γλώττιση του κώδικα, ο καθορισμός των παραμέτρων, το τρέξιμο και ο υπολογισμός τις τιμές $x(t \approx 95)$ για πολλές τιμές του δt . Σε ένα αρχείο `box1D_anal.in` γράφουμε τις τιμές των παραμέτρων που θα εισάγουμε στο πρόγραμμα:

```
10          L
0 1.0      x0 v0
0 95  0.05 t0 tf dt
```

Στη συνέχεια μεταγλωττίζουμε το πρόγραμμα

```
> f77 box1D_1.f -o box
```

και το τρέχουμε με την εντολή

```
> cat box1D_anal.in | ./box
```

που στέλνει τα περιεχόμενα του αρχείου `box1D_anal.in` στο `stdin` του προγράμματος `./box`. Το αποτέλεσμα που ζητάμε είναι στην τελευταία γραμμή του αρχείου `box1D_1.dat`:

```
> tail -n 1 box1D_1.dat
94.9511948  5.45000267 -1.
```

Η τρίτη τιμή είναι αυτή της ταχύτητας και δε μας ενδιαφέρει. Σε ένα αρχείο, λ.χ. `box1D_anal.dat` τοποθετούμε το δt και τις δύο πρώτες τιμές από την παραπάνω εντολή. Αλλάζουμε την τιμή του $\delta t \rightarrow \delta t/2$ στο αρχείο `box1D_anal.in` 12 φορές μέχρι να γίνει ίση με 0.000012 και επαναλαμβάνουμε²⁰. Επαναλαμβάνουμε τη διαδικασία²¹ για τη μέθοδο 2 και για την ακριβή μέθοδο τοποθετώντας τα αποτελέσματα για το $x(t \approx 95)$ σε νέες στήλες στο αρχείο `box1D_anal.dat`. Το αποτέλεσμα είναι

```
# -----
# dt      t1_95    x1(95)    x2(95)    x3(95)
# -----
0.050000 94.95119 5.450003 5.550126 5.048808
0.025000 94.97849 5.275011 5.174837 5.021509
0.012500 94.99519 5.124993 5.099736 5.004811
```

²⁰Ο σεφ προτείνει: Δοκιμάστε την εντολή `sed 's/0.05/0.025/' box1D_anal.in | ./box` αλλάζοντας το 0.025 με την τιμή του δt που επιθυμείτε να μελετήσετε.

²¹Σπεσιαλιτέ: Δείτε το σενάριο φλοιού `box1D_anal.csh` στο συνοδευτικό λογισμικό για ιδέες αυτοματοποίησης της διαδικασίας.

```

0.006250 94.99850 4.987460 5.063134 5.001502
0.003125 94.99734 5.021894 5.035365 5.002666
0.001563 94.99923 5.034538 5.017764 5.000772
0.000781 94.99939 4.919035 5.011735 5.000607
0.000391 94.99979 4.695203 5.005493 5.000212
0.000195 95.00000 5.434725 5.001935 5.000003
0.000098 94.99991 5.528124 5.000745 5.000093
0.000049 94.99998 3.358000 5.000330 5.000020
0.000024 94.99998 2.724212 5.000232 5.000014
0.000012 94.99999 9.240705 5.000158 5.000011

```

όπου τη δεύτερη γραμμή την προσθέσαμε ως σχόλιο (ο `gnuplot` την αγνοεί λόγω του `#`) που μας θυμίζει τι αντιπροσωπεύει κάθε στήλη.

Η μελέτη της σύγκλισης μπορεί να φανεί εποπτικά στο αριστερό Σχήμα 2.21. Η 1η μέθοδος βελτιώνει την ακρίβειά της μεγιστοποιώντας την όταν $\delta t \approx 0.01$ ενώ για $\delta t < 0.0001$ το σφάλμα γίνεται $> 10\%$ και η μέθοδος δίνει άχρηστα αποτελέσματα. Η 2η μέθοδος έχει πολύ καλύτερη συμπεριφορά από την 1η. Για να προσπαθήσουμε να καταλάβουμε το λόγο.

Παρατηρούμε ότι καθώς μικραίνει το δt η τελική τιμή του t πλησιάζει την αναμενόμενη $t_f = 95$. Γιατί όμως δεν παίρνουμε $t = 95$, ειδικά όταν $t/\delta t$ είναι ακέραιος αριθμός; Επίσης παρατηρούμε στην 9η γραμμή των δεδομένων του πίνακα ($dt=0.000195$): Αφού $95/0.000195$ δεν είναι ακέραιος, γιατί $t = 95$; Ακόμα χειρότερα: Αν θεωρητικά αναμένουμε να γίνουν $95/\delta t$ βήματα πόσα γίνονται στην πραγματικότητα; Κάθε φορά που κάνετε μια μέτρηση μετρήστε πόσες γραμμές έχει το αρχείο `box1D_1.dat` με την εντολή²²

```
> wc -l box1D_1.dat
```

και συγκρίνετε με τον αναμενόμενο αριθμό. Το αποτέλεσμα είναι ενδιαφέρον:

```

# -----
#  dt    N      NO
# -----
0.050000 1900    1900
0.025000 3800    3800
0.012500 7601    7600
0.006250 15203   15200
0.003125 30394   30400

```

²²Εναλλακτικά, βάλτε ένα μετρητή μέσα στο βρόχο του προγράμματος.

```

0.001563 60760    60780
0.000781 121751   121638
0.000391 243753   242966
0.000195 485144   487179
0.000098 962662   969387
0.000049 1972589  1938775
0.000024 4067548  3958333
0.000012 7540956  7916666

```

όπου η 2η στήλη έχει τον αριθμό των βημάτων που έγιναν και η 3η τον αριθμό των βημάτων που έπρεπε να γίνουν. Παρατηρούμε ότι η ακρίβεια *ελαττώνεται* όταν μικραίνουμε το δt και στο τέλος η διαφορά είναι περίπου 5%! Ειδικά στην τελευταία γραμμή, αν οι πράξεις γίνονταν με μεγαλύτερη ακρίβεια θα έπρεπε ο τελικός χρόνος να ήταν $t_f = 0.000012 \times 7916668 \approx 90.5$ κάτι που αλλάζει δραματικά το αποτέλεσμα σε μια περιοδική κίνηση με περίοδο συγκρινόμενη με το σφάλμα στο χρόνο... Επειδή η 1η μέθοδος προχωράει το χρόνο στις εξισώσεις κίνησης ανάλογα με τον αριθμό των βημάτων καταλαβαίνουμε ότι στην πραγματικότητα η τιμή που παίρνουμε είναι για άλλο χρόνο από αυτόν που νομίζουμε.

Άρα μια σημαντική πηγή σφαλμάτων είναι το σφάλμα συσσώρευσης στον υπολογισμό του χρόνου που γίνεται μεγαλύτερο καθώς μικραίνει το δt . Πώς θα μπορούσαμε να βελτιώσουμε τη συμπεριφορά αυτή; Μια σημαντική βελτίωση μπορεί να γίνει αν αντί να υπολογίζουμε το χρόνο προσθετικά, τον υπολογίζουμε πολλαπλασιαστικά. Έστω i ένας μετρητής των βημάτων που έχουμε κάνει. Τότε

```

C      t = t  + dt      ! Not accurate,    avoid
      t = t0 + i*dt     ! Better accuracy, prefer

```

Ο κύριος βρόχος του προγράμματος στο `box1D_1.f` γίνεται:

```

t = t0
x = x0
v = v0
i = 0
do while(t .le. tf)
  write(11,*)t,x,v
  i = i  + 1
  x = x  + v*dt

```

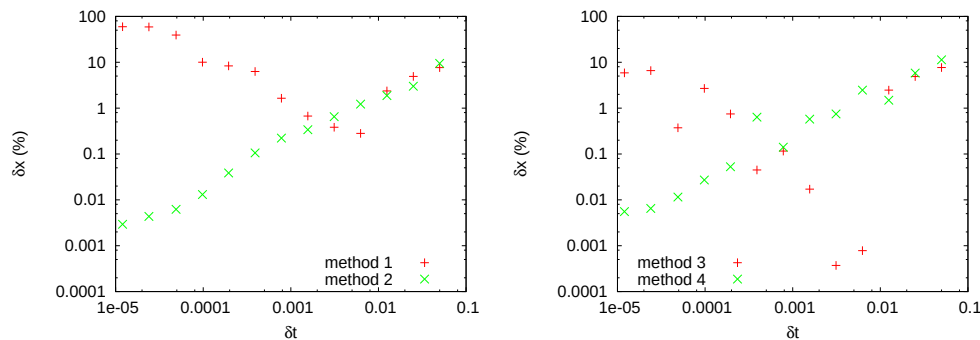


```

t = t0 + i*dt
if(x .lt. 0.0 .or. x .gt. L) v = -v
enddo

```

Το πλήρες πρόγραμμα θα το βρείτε στο αρχείο `box1D_4.f` του συνοδευτικού λογισμικού. Ας ονομάσουμε τη μέθοδο αυτή ‘μέθοδο 3’. Παρόμοια αλλαγή κάνουμε και στο αρχείο `box1D_2.f` που θα βρείτε στο `box1D_5.f` και καλούμε τη μέθοδο αυτή ‘μέθοδο 4’. Τέλος τον ακριβή αλγόριθμο του αρχείου `box1D_3.f` τον μετατρέπουμε στο αρχείο `box1D_6.f`. Επαναλαμβάνουμε την παραπάνω ανάλυση και βρίσκουμε ότι το πρόβλημα ακριβούς προσδιορισμού του χρόνου πρακτικά εξαφανίζεται. Το αποτέλεσμα της ανάλυσης του σφάλματος το δείχνουμε στο δεξί Σχήμα 2.21. Η μέθοδος 2 και 4 δεν παρουσιάζει σημαντική διαφορά. Στη μέθοδο 1 και 3 η διαφορά είναι δραματική με το σφάλμα να μειώνεται μέχρι και περισσότερο από 10 φορές. Το πρόβλημα του αυξανόμενου



Σχήμα 2.21: Το επί % σφάλμα $\delta x = 2|x_i(95) - x(95)|/|x_i(95) + x(95)| \times 100$ όπου $x_i(95)$ η τιμή που υπολογίζει η μέθοδος $i = 1, 2, 3, 4$ και $x(95)$ η τιμή που υπολογίζει ο ακριβής αλγόριθμος σύμφωνα με το κείμενο.

σφάλματος καθώς μειώνουμε το δt δεν εξαφανίζεται. Αλλά τώρα καταλαβαίνουμε ότι προέρχεται από τη συσσώρευση σφάλματος στην επαναληπτική σχέση $x = x + v \cdot dt$. Αυτόν τον τύπο σφάλματος είναι δυσκολότερο να το βελτιώσουμε και παρουσιάζεται συχνά σε μεθόδους επίλυσης διαφορικών εξισώσεων που θα μελετήσουμε στο επόμενο κεφάλαιο.

2.3.3 Το Δισδιάστατο Κουτί

Ας γενικεύσουμε τη μελέτη μας όταν το υλικό σημείο κινείται στις δύο διαστάσεις, στο επίπεδο. Το σωματίο είναι περιορισμένο μέσα σε ένα κουτί $0 < x <$

L_x , $0 < y < L_y$ και σχεδιάζεται ελαστικά πάνω στα τοιχώματά του. Βρίσκεται δηλαδή σε ένα ορθογώνιο απειρόβαθο πηγάδι δυναμικού. Ο χρήστης τοποθετεί τη χρονική στιγμή t_0 το υλικό σημείο σε μια αρχική θέση (x_0, y_0) με αρχική ταχύτητα (v_{0x}, v_{0y}) και το πρόγραμμα υπολογίζει την τροχιά που ακολουθεί το υλικό σημείο μέχρι το χρόνο t_f με βήμα δt . Μία τέτοια τροχιά φαίνεται στο Σχήμα 2.23.

Θα γενικεύσουμε τον αλγόριθμο που χρησιμοποιήσαμε στο πρόγραμμα `box2D_1.f` για να μελετήσουμε την κίνηση στο μονοδιάστατο κουτί. Αν είναι γνωστή η θέση και η ταχύτητα του κινητού σε μια χρονική στιγμή t , τότε τη χρονική στιγμή $t + \delta t$ η θέση και η ταχύτητά του θα δίνεται από τις σχέσεις

$$\begin{aligned} x(t + \delta t) &= x(t) + v_x(t)\delta t \\ y(t + \delta t) &= y(t) + v_y(t)\delta t \\ v_x(t + \delta t) &= v_x(t) \\ v_y(t + \delta t) &= v_y(t). \end{aligned} \quad (2.25)$$

Η σύγκρουση με τα τοιχώματα προτυποποιείται με ανάκλαση της κάθετης στα τοιχώματα συνιστώσας της ταχύτητας όταν η αντίστοιχη συντεταγμένη περάσει τα όρια του τοίχου. Φυσικά αυτό εισάγει το συστηματικό σφάλμα που συζητήσαμε στην προηγούμενη παράγραφο. Αποφεύγουμε το συστηματικό σφάλμα συσσώρευσης στον υπολογισμό του χρόνου εισάγοντας έναν μετρητή βημάτων i , και έτσι το κεντρικό κομμάτι του προγράμματος είναι:

```
i = i + 1
t = t0 + i * dt
x = x + vx*dt
y = y + vy*dt
if(x .lt. 0.0 .or. x .gt. Lx) vx = -vx
if(y .lt. 0.0 .or. y .gt. Ly) vy = -vy
```

Παραθέτουμε το πλήρες πρόγραμμα παρακάτω το οποίο θα βρείτε στο αρχείο `box2D_1.f`. Εκτός από τα εισαγωγικά και τις αρχικοποιήσεις, εισάγαμε και δύο μετρητές κρούσεων με τα τοιχώματα `nx` και `ny`:

```
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      File box2D_1.f
C      Motion of a free particle in a box  0<x<Lx 0<y<Ly
C      Use integration with time step dt: x = x + vx*dt y=y+vy*dt
C      -----
```

```

      program box2D
      implicit none
C -----
C Declaration of variables
      real*8  Lx,Ly,x0,y0,v0x,v0y,t0,tf,dt,t,x,y,vx,vy
      integer i,nx,ny
C -----
C Ask user for input:
      print *,'# Enter Lx,Ly:'
      read(5,*)Lx,Ly
      print *,'# Lx = ',Lx,' Ly= ',Ly
      if( Lx .le. 0.0) stop 'Lx must be positive.'
      if( Ly .le. 0.0) stop 'Ly must be positive.'
      print *,'# Enter x0,y0,v0x,v0y:'
      read(5,*)x0,y0,v0x,v0y
      print *,'# x0= ',x0,' y0= ',y0,' v0x= ',v0x,' v0y= ',v0y
      if(x0 .lt. 0.0 .or. x0 .gt. Lx) stop 'illegal value of x0.'
      if(y0 .lt. 0.0 .or. y0 .gt. Ly) stop 'illegal value of y0.'
      if(v0x**2+v0y**2.eq. 0.0      ) stop 'illegal value of v0 = 0.'
      print *,'# Enter t0,tf,dt:'
      read(5,*)t0,tf,dt
      print *,'# t0= ',t0,' tf= ',tf,' dt= ',dt
C -----
C Initialize
      i  = 0
      nx = 0
      ny = 0
      t  = t0
      x  = x0
      y  = y0
      vx = v0x
      vy = v0y
      open(unit=11,file='box2D_1.dat')
C -----
C Compute:
      do while(t .le. tf)
        write(11,*)t,x,y,vx,vy
        i = i  + 1

```

```

t = t0 + i *dt
x = x  + vx*dt
y = y  + vy*dt
if(x .lt. 0.0 .or. x .gt. Lx) then
  vx = -vx
  nx = nx + 1
endif
if(y .lt. 0.0 .or. y .gt. Ly) then
  vy = -vy
  ny = ny + 1
endif
enddo
close(11)
print *, '# Number of collisions:'
print *, '# nx= ',nx,' ny= ',ny
end

```

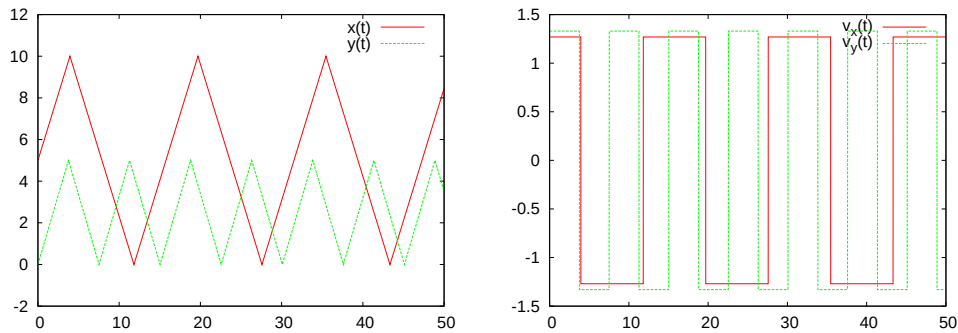
Μια τυπική συνεδρία μελέτης μιας τροχιάς δίνεται παρακάτω:

```

> f77 box2D_1.f -o box
> ./box
# Enter Lx,Ly:
10.0 5.0
# Lx = 10. Ly= 5.
# Enter x0,y0,v0x,v0y:
5.0 0.0 1.27 1.33
# x0= 5. y0= 0. v0x= 1.27 v0y= 1.33
# Enter t0,tf,dt:
0 50 0.01
# t0= 0. tf= 50. dt= 0.01
# Number of collisions:
# nx= 6 ny= 13
> gnuplot
gnuplot> plot "box2D_1.dat" using 1:2 with lines title "x (t)"
gnuplot> replot "box2D_1.dat" using 1:3 with lines title "y (t)"
gnuplot> plot "box2D_1.dat" using 1:4 with lines title "vx(t)"
gnuplot> replot "box2D_1.dat" using 1:5 with lines title "vy(t)"
gnuplot> plot "box2D_1.dat" using 2:3 with lines title "x-y"

```

Παρατηρούμε την τελευταία γραμμή εξόδου από το πρόγραμμα: Το υλικό σημείο ανακλάται από τα κάθετα τοιχώματα του κουτιού 6 φορές ($nx=6$) και με τα οριζόντια 13 ($ny=13$). Με τις εντολές αυτές παίρνουμε τις γραφικές παραστάσεις που φαίνονται στα Σχήματα 2.22 και 2.23.



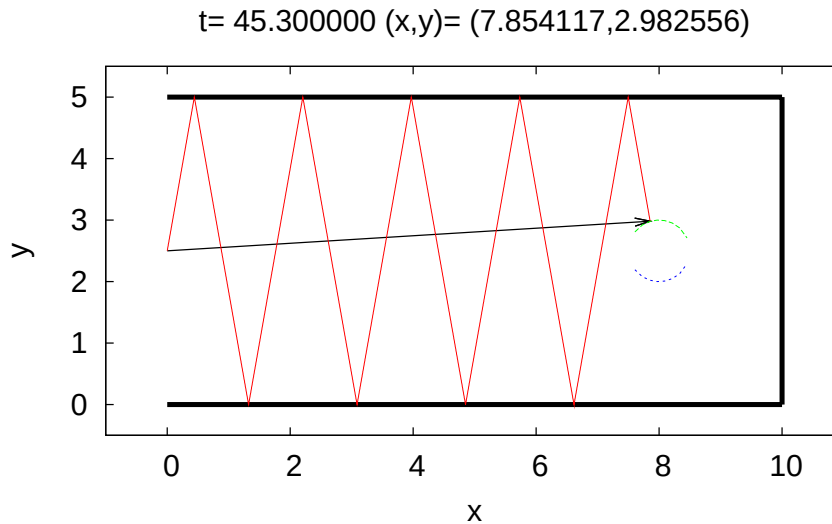
Σχήμα 2.22: Τα αποτελέσματα για το υλικό σημείο που κινείται μέσα σε δισδιάστατο κουτί που παίρνουμε από το πρόγραμμα `box2D_1.f`. Οι παράμετροι είναι $L_x = 10$, $L_y = 5$, $x_0 = 5$, $y_0 = 0$, $v_{0x} = 1.27$, $v_{0y} = 1.33$, $t_0 = 0$, $t_f = 50$, $\delta t = 0.01$.

Για να κάνουμε animation της τροχιάς αντιγράφουμε το αρχείο `box2D_animate.gnu` από το συνοδευτικό λογισμικό στον κατάλογο που έχουμε τα δεδομένα μας και δίνουμε τις εντολές στο `gnuplot`:

```
gnuplot> file = "box2D_1.dat"
gnuplot> Lx = 10 ; Ly = 5
gnuplot> t0 = 0 ; tf = 50; dt = 1
gnuplot> load "box2D_animate.gnu"
gnuplot> t0 = 0 ; dt = 0.5; load "box2D_animate.gnu"
```

Στην τελευταία γραμμή επαναλάβουμε το animation στη ... μισή ταχύτητα. Μπορείτε να χρησιμοποιήσετε επίσης το ήδη γνωστό πρόγραμμα `animate2D.gnu` που παρουσιάσαμε στην παράγραφο 2.1.1. Οι αλλαγές που κάναμε στο `box2D_animate.gnu` απλά προσθέτουν ένα σχέδιο του κουτιού και υπολογίζουν αυτόματα τα όρια σχεδιασμού της γραφικής παράστασης. Επίσης, το διάνυσμα που παρακολουθεί την κίνηση του σωματιδίου δεν είναι το διάνυσμα θέσης, αλλά αυτό που ενώνει την αρχική με την τελική θέση του.

Το επόμενο βήμα είναι να ελέγξουμε την ακρίβεια των αποτελεσμάτων μας. Αυτό γίνεται στο πνεύμα της αντίστοιχης παρουσίασης του μονοδιάστατου προβλήματος και αφήνεται σαν άσκηση για τον αναγνώστη.



Σχήμα 2.24: Η τροχιά του υλικού σημείου που παράγεται από το πρόγραμμα MiniGolf.f με παραμέτρους αυτές που δίνονται στο κείμενο. Απεικονίζεται η στιγμή της ... επιτυχίας! Τη χρονική στιγμή $t = 45.3$ το υλικό σημείο μπαίνει στην περιοχή της τρύπας με κέντρο $(8, 2.5)$ και ακτίνα 0.5 .

```

C      File MiniGolf.f
C      Motion of a free particle in a box  0<x<Lx 0<y<Ly
C      The box is open at x=0 and has a hole at (xc,yc) of radius R
C      Ball is shot at (0,Ly/2) with speed v0, angle theta (degrees)
C      Use integration with time step dt: x = x + vx*dt y=y+vy*dt
C      Ball stops in hole (success) or at x=0 (failure)
C      -----
C      program MiniGolf
C      implicit none
C      -----
C      Declaration of variables
C      real*8  Lx,Ly,x0,y0,v0x,v0y,t0,tf,dt,t,x,y,vx,vy
C      real*8  v0,theta,xc,yc,R,R2,PI
C      integer i,nx,ny
C      character*7 result

```

```

parameter(PI=3.14159265358979324D0)
C -----
C Ask user for input:
print *,'# Enter Lx,Ly:'
read(5,*)Lx,Ly
print *,'# Lx = ',Lx,' Ly= ',Ly
if( Lx .le. 0.0) stop 'Lx must be positive.'
if( Ly .le. 0.0) stop 'Ly must be positive.'
print *,'# Enter hole position and radius: (xc,yc), R:'
read(5,*)xc,yc,R
print *,'# (xc,yc)= ( ',xc,' , ',yc,' ) R= ',R
print *,'# Enter v0, theta(degrees):'
read(5,*)v0,theta
print *,'# v0= ',v0,' theta= ',theta,' degrees'
if(v0 .le. 0.0D0 ) stop 'illegal value of v0.'
if(ABS(theta).ge. 90.0D0) stop 'illegal value of theta.'
print *,'# Enter dt:'
read(5,*)dt
print *,'# dt= ',dt
C -----
C Initialize
t0 = 0.0D0
x0 = 0.00001D0 ! small but non-zero
y0 = Ly/2.0
R2 = R*R
theta = (PI/180.0D0)*theta
v0x = v0*cos(theta)
v0y = v0*sin(theta)
print *,'# x0= ',x0,' y0= ',y0,' v0x= ',v0x,' v0y= ',v0y
i = 0
nx = 0
ny = 0
t = t0
x = x0
y = y0
vx = v0x
vy = v0y
open(unit=11,file='MiniGolf.dat')

```



```

C -----
C Compute:
do while( .TRUE. ) !forever!
  write(11,*)t,x,y,vx,vy
  i = i + 1
  t = t0 + i*dt
  x = x + vx*dt
  y = y + vy*dt
  if(x .gt. Lx) then
    vx = -vx
    nx = nx + 1
  endif
  if(y .lt. 0.0 .or. y .gt. Ly) then
    vy = -vy
    ny = ny + 1
  endif
  if(x .le. 0.0D0)then
    result = 'Failure'
    goto 11
  endif
  if( ((x-xc)*(x-xc)+(y-yc)*(y-yc)) .le. R2)then
    result = 'Success'
    goto 11
  endif
enddo
11 continue !break loop here
close(11)
print *, '# Number of collisions:'
print *, '# Result= ',result,' nx= ',nx,' ny= ',ny
end

```

Για να το τρέξουμε κάνουμε τα συνηθισμένα:

```

> f77 MiniGolf.f -o mg
> ./mg
# Enter Lx,Ly:
10 5
# Lx = 10. Ly= 5.
# Enter hole position and radius: (xc,yc), R:

```

```

8 2.5 0.5
# (xc,yc)= ( 8. , 2.5 ) R= 0.5
# Enter v0, theta(degrees):
1 80
# v0= 1. theta= 80. degrees
# Enter dt:
0.01
# dt= 0.01
# x0= 1.E-05 y0= 2.5 v0x= 0.173648178 v0y= 0.984807753
# Number of collisions:
# Result= Success nx= 0 ny= 9

```

Φτιάξτε τα διαγράμματα της θέσης και της ταχύτητας με το χρόνο καθώς και της τροχιάς κατά τα γνωστά. Για διασκέδαση του αναγνώστη παρέχουμε και ειδικό πρόγραμμα animation (μπορείτε να χρησιμοποιήσετε και το `animate2D.gnu`). Αντιγράψτε από το συνοδευτικό λογισμικό το αρχείο `MiniGolf_animate.gnu`, ξεκινήστε το `gnuplot` και δώστε τις εντολές:

```

gnuplot> file = "MiniGolf.dat"
gnuplot> Lx = 10;Ly = 5
gnuplot> xc = 8; yc = 2.5 ; R = 0.5
gnuplot> t0 = 0; dt = 0.1
gnuplot> load "MiniGolf_animate.gnu"

```

Το Σχήμα 2.24 αποτελεί αδιάψευστη μαρτυρία της επιτυχίας μας!

Η επόμενη εφαρμογή μας θα είναι τρισδιάστατη. Θα μελετήσουμε την κίνηση ενός υλικού σημείου το οποίο κινείται μέσα σε ένα κύλινδρο ακτίνας R και ύψους L . Η κρούσεις στα τοιχώματα και τις βάσεις του κυλίνδρου είναι ελαστικές και ο κύλινδρος είναι αμετακίνητος και απαραμόρφωτος. Στο πρόγραμμα που θα γράψουμε παίρνουμε τον κύλινδρο να έχει τον άξονα συμμετρίας τον άξονα των z . Η μία βάση του τοποθετείται στο επίπεδο $z = 0$ και η άλλη στο $z = L$. Η διάταξη φαίνεται στο Σχήμα 2.26.

Το πρόβλημα δεν παρουσιάζει καμία δυσκολία σε σχέση με τα προηγούμενα όσο αφορά την κίνηση στον άξονα των z , το σώματι ανακλάται πάνω στα επίπεδα τοιχώματα των βάσεων του κυλίνδρου. Το μόνο καινούργιο είναι η προβολή της κίνησης στο επίπεδο $x - y$, όπου το υλικό σημείο φαίνεται να κινείται σε ένα επίπεδο και να συγκρούεται ελαστικά στο εσωτερικό ενός κύκλου ακτίνας R και κέντρο πάνω στον άξονα των z . Η κατάσταση περιγράφεται στο Σχήμα

2.25. Κατά την ελαστική ανάκλαση του σωματίου απλά $v_r \rightarrow -v_r$ ενώ η v_θ μένει ή ίδια. Η ταχύτητα του σωματιδίου πριν την χρούση είναι

$$\begin{aligned}\vec{v} &= v_x \hat{x} + v_y \hat{y} \\ &= v_r \hat{r} + v_\theta \hat{\theta}\end{aligned}\quad (2.26)$$

ενώ μετά

$$\begin{aligned}\vec{v}' &= v'_x \hat{x} + v'_y \hat{y} \\ &= -v_r \hat{r} + v_\theta \hat{\theta}\end{aligned}\quad (2.27)$$

Από τις σχέσεις

$$\begin{aligned}\hat{r} &= \cos \theta \hat{x} + \sin \theta \hat{y} \\ \hat{\theta} &= -\sin \theta \hat{x} + \cos \theta \hat{y},\end{aligned}\quad (2.28)$$

και τις σχέσεις $v_r = \vec{v} \cdot \hat{r}$, $v_\theta = \vec{v} \cdot \hat{\theta}$, παίρνουμε

$$\begin{aligned}v_r &= v_x \cos \theta + v_y \sin \theta \\ v_\theta &= -v_x \sin \theta + v_y \cos \theta.\end{aligned}\quad (2.29)$$

Αντιστρέφοντάς τις παίρνουμε

$$\begin{aligned}v_x &= v_r \cos \theta - v_\theta \sin \theta \\ v_y &= v_r \sin \theta + v_\theta \cos \theta.\end{aligned}\quad (2.30)$$

Με το μετασχηματισμό $v_r \rightarrow -v_r$ η νέα ταχύτητα σε καρτεσιανές συντεταγμένες θα είναι

$$\begin{aligned}v'_x &= -v_r \cos \theta - v_\theta \sin \theta \\ v'_y &= -v_r \sin \theta + v_\theta \cos \theta.\end{aligned}\quad (2.31)$$

Ο μετασχηματισμός $v_x \rightarrow v'_x$, $v_y \rightarrow v'_y$ θα υλοποιηθεί σε μια υπορουτίνα `reflectVonCircle(vx,vy,x,y,xc,yc,R)`. Στην είσοδο της δίνουμε την αρχική ταχύτητα (v_x, v_y) , το σημείο χρούσης (x, y) , το κέντρο του κύκλου (x_c, y_c) και την ακτίνα του κύκλου²³ R . Στην έξοδο, τα (v_x, v_y) έχουν αντικατασταθεί με τις νέες τιμές²⁴ (v'_x, v'_y) .

Το πρόγραμμα που προκύπτει θα το βρείτε στο αρχείο `Cylinder3D.f`:

²³Προφανώς ιδανικά θα ισχύει $R^2 = (x - x_c)^2 + (y - y_c)^2$ αλλά επειδή υπάρχει συστηματικό σφάλμα στη θέση της χρούσης, επιλέγουμε το R να δίνεται.

²⁴Επίσης το υλικό σημείο όπως θα δούμε τοποθετείται ακριβώς πάνω στον κύκλο.


```

C -----
C   Ask user for input:
   print *, '# Enter R, L:'
   read(5,*)R,L
   print *, '# R= ',R,' L= ',L
   if( R .le. 0.0) stop 'R must be positive.'
   if( L .le. 0.0) stop 'L must be positive.'
   print *, '# Enter x0,y0,z0,v0x,v0y,v0z:'
   read(5,*)x0,y0,z0,v0x,v0y,v0z
   rxy = DSQRT(x0*x0+y0*y0)
   print *, '# x0 = ',x0 ,' y0 = ',y0 ,' z0= ',z0, ' rxy= ',rxy
   print *, '# v0x= ',v0x,' v0y= ',v0y,' v0z= ',v0z
   if(rxy .gt. R                ) stop 'illegal value of rxy > R'
   if(z0 .lt. 0.0D0             ) stop 'illegal value of z0 < 0'
   if(z0 .gt. L                ) stop 'illegal value of z0 > L'
   if(v0x**2+v0y**2+v0z**2.eq. 0.0 ) stop 'illegal value of v0 = 0.'
   print *, '# Enter t0,tf,dt:'
   read(5,*)t0,tf,dt
   print *, '# t0= ',t0,' tf= ',tf,' dt= ',dt
C -----
C   Initialize
   i  = 0
   nr = 0
   nz = 0
   t  = t0
   x  = x0
   y  = y0
   z  = z0
   vx = v0x
   vy = v0y
   vz = v0z
   R2 = R*R
   xc = 0.0D0 !center of circle which is the projection of the
   yc = 0.0D0 !cylinder on the xy plane
   open(unit=11,file='Cylinder3D.dat')
C -----
C   Compute:
   do while(t .le. tf)

```

```

write(11,100)t,x,y,z,vx,vy,vz
i = i + 1
t = t0 + i *dt
x = x + vx*dt
y = y + vy*dt
z = z + vz*dt
if(z .lt. 0.0 .or. z .gt. L) then
  vz = -vz          ! reflection on cylinder caps
  nz = nz + 1
endif
r2xy = x*x+y*y
if( r2xy .gt. R2)then
  call reflectVonCircle(vx,vy,x,y,xc,yc,R)
  nr = nr + 1
endif
enddo
close(11)
print *, '# Number of collisions:'
print *, '# nr= ',nr,' nz= ',nz

100 FORMAT(100G28.16)
end

C -----
C =====
C -----

subroutine reflectVonCircle(vx,vy,x,y,xc,yc,R)
implicit none
real*8 vx,vy,x,y,xc,yc,R
real*8 theta,cth,sth,vr,vth

theta = atan2(y-yc,x-xc)
cth   = cos(theta)
sth   = sin(theta)

vr    = vx*cth + vy *sth
vth   = -vx*sth + vy *cth

vx    = -vr*cth - vth*sth !reflect vr -> -vr

```

```

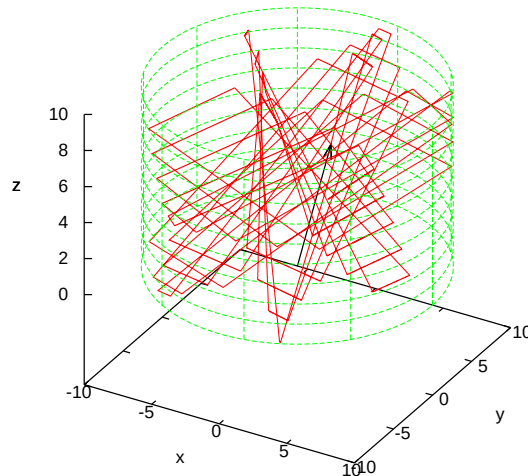
vy      = -vr*sth + vth*cth

x       = xc + R*cth      !put x,y on the circle
y       = yc + R*sth
end

```

Να επισημάνουμε μόνο τις ακόλουθες λεπτομέρειες: Χρησιμοποιούμε καταρχήν τη συνάρτηση `atan2` για τον προσδιορισμό της γωνίας `theta`. Η συνάρτηση αυτή με ορίσματα `atan2(y,x)` επιστρέφει τη γωνία $\theta = \tan^{-1}(y/x)$ στο σωστό τεταρτημόριο που βρίσκεται το σημείο (x, y) . Η γωνία που ζητάμε δίνεται από την εντολή `atan2(y-yc, x-xc)`. Στη συνέχεια εφαρμόζουμε τις σχέσεις (2.29) και (2.31), ενώ στις τελευταίες δύο γραμμές επιβάλλουμε μετά την κρούση το υλικό σημείο να είναι πάνω στο σημείο του κύκλου $(x_c + R \cos \theta, y_c + R \sin \theta)$.

t= 500.000000 (x,y,z)=(2.227212,0.469828,7.088600)



Σχήμα 2.26: Η τροχιά του σωματιδίου που ανακλάται στο εσωτερικό κυλίνδρου με $R = 10$, $L = 10$ σύμφωνα με το πρόγραμμα `Cylinder3D.f`. Έχουμε επιλέξει $\vec{r}_0 = 1.0\hat{x} + 2.2\hat{y} + 3.1\hat{z}$, $\vec{v}_0 = 0.93\hat{x} - 0.89\hat{y} + 0.74\hat{z}$, $t_0 = 0$, $t_f = 500.0$, $\delta t = 0.01$.

Η μεταγλώττιση και το τρέξιμο γίνεται κατά τα γνωστά:

```
> f77 Cylinder3D.f -o cl
```

```

> ./cl
# Enter R, L:
10.0 10.0
# R= 10. L= 10.
# Enter x0,y0,z0,v0x,v0y,v0z:
1.0 2.2 3.1 0.93 -0.89 0.74
# x0 = 1. y0 = 2.2 z0= 3.1 rxy= 2.41660919
# v0x= 0.93 v0y= -0.89 v0z= 0.74
# Enter t0,tf,dt:
0.0 500.0 0.01
# t0= 0. tf= 500. dt= 0.01
# Number of collisions:
# nr= 33 nz= 37

```

Για να φτιάξουμε τις γραφικές παραστάσεις της θέσης/ταχύτητας συναρτήσει του χρόνου δίνουμε τις εντολές `gnuplot` κατά τα γνωστά:

```

gnuplot> file="Cylinder3D.dat"
gnuplot> plot file using 1:2 with lines t " x(t)",\
           file using 1:3 with lines t " y(t)",\
           file using 1:4 with lines t " z(t)"
gnuplot> plot file using 1:5 with lines t "v_x(t)",\
           file using 1:6 with lines t "v_y(t)",\
           file using 1:7 with lines t "v_z(t)"

```

Μπορούμε να δούμε και την απόσταση του κινητού από τον άξονα του κυλίνδρου $r(t) = \sqrt{x(t)^2 + y(t)^2}$ σε συνάρτηση με το χρόνο με την εντολή

```

gnuplot> plot file using 1:(sqrt($2**2+$3**2)) with lines title "r(t)"

```

Για να φτιάξουμε το σχήμα τις τροχιές μαζί με τον κύλινδρο δίνουμε τις εντολές

```

gnuplot> L = 10 ; R = 10
gnuplot> set urange [0:2.0*pi]
gnuplot> set vrange [0:L]
gnuplot> set parametric
gnuplot> splot file using 2:3:4 with lines notitle,\
           R*cos(u),R*sin(u),v notitle

```


Με την εντολή `set parametric` μπορούμε να φτιάξουμε τη γραφική παράσταση μιας δισδιάστατης επιφάνειας της μορφής $\vec{r}(u, v) = x(u, v) \hat{x} + y(u, v) \hat{y} + z(u, v) \hat{z}$. Ο κύλινδρος (χωρίς τις βάσεις του) δίνεται από τις παραμετρικές εξισώσεις $\vec{r}(u, v) = R \cos u \hat{x} + R \sin u \hat{y} + v \hat{z}$ με $u \in [0, 2\pi)$, $v \in [0, L]$.

Για το animation της τροχιάς αντιγράψτε το αρχείο `Cylinder3D_animate.gnu` στον κατάλογο που κάνετε την ανάλυση και μέσα από το `gnuplot` δώστε τις εντολές

```
gnuplot> R=10;L=10;t0=0;tf=500;dt=10;load "Cylinder3D_animate.gnu"
```

βάζοντας φυσικά τις παραμέτρους που χρησιμοποιήσατε εσείς. Το αποτέλεσμα φαίνεται στο Σχήμα 2.26.

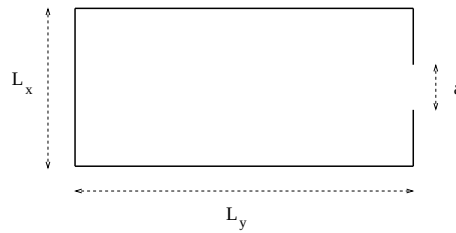
2.5 Ασκήσεις

- 2.1 Στο πρόγραμμα `Circle.f` δώστε τις εντολές που απαιτούνται ώστε το πρόγραμμα να τυπώνει τον αριθμό των πλήρων κύκλων που διέγραψε το κινητό.
- 2.2 Στο πρόγραμμα `Circle.f` προσθέστε όλα τα αναγκαία τεστ στις παραμέτρους που εισάγει ο χρήστης, έτσι ώστε το πρόγραμμα να είναι εγγυημένο ότι θα τρέξει ομαλά. Κάνετε το ίδιο και στα άλλα προγράμματα που δίνονται στο κεφάλαιο αυτό.
- 2.3 Γλιτικό σημείο κινείται πάνω σε κύκλο με κέντρο την αρχή των αξόνων με σταθερή γωνιακή ταχύτητα ω . Τη χρονική στιγμή $t_0 = 0$ το σωματίο βρίσκεται στο σημείο (x_0, y_0) . Γράψτε το πρόγραμμα `CircularMotion.f` που να απεικονίζει την τροχιά. Ο χρήστης θα δίνει στην είσοδο τα $\omega, x_0, y_0, t_0, t_f, \delta t$. Στην έξοδο θα παίρνει τα δεδομένα σύμφωνα με το πρόγραμμα `Circle.f`.
- 2.4 Μετατρέψτε το πρόγραμμα `SimplePendulum.f` έτσι ώστε ο χρήστης να μπορεί να δίνει και μη μηδενική ταχύτητα ως αρχική συνθήκη.
- 2.5 Μελετήστε το όριο $k \rightarrow 0$ στο πρόβλημα της βολής που δίνεται από τις Σχέσεις (2.10). Αναπτύξτε το $e^{-kt} = 1 - kt + \frac{1}{2!}(kt)^2 + \dots$ και κρατήστε τους όρους που δε μηδενίζονται στο όριο $k \rightarrow 0$. Στη συνέχεια κρατήστε τους όρους με την αμέσως μικρότερη δύναμη του k . Προγραμματίστε τις εξισώσεις αυτές σε ένα αρχείο `ProjectileSmallAirResistance.f`. Θεωρήστε τις αρχικές συνθήκες $\vec{v}_0 = \hat{x} + \hat{y}$ και υπολογίστε αριθμητικά το βεληνεκές της τροχιάς με τα δύο προγράμματα `ProjectileSmallAirResistance.f`, `ProjectileAirResistance.f`. Προσδιορίστε τις τιμές του k για τις οποίες τα αποτελέσματά σας συμφωνούν με ακρίβεια καλύτερη από 5%.
- 2.6 Προγραμματίστε την κίνηση της βολής όταν η δύναμη της αντίστασης του αέρα έχει μέτρο ανάλογο του τετραγώνου της ταχύτητας. Συγκρίνετε το βεληνεκές της τροχιάς που προκύπτει με αυτό που υπολογίζεται από το πρόγραμμα `ProjectileAirResistance.f` για τις παραμέτρους του Σχήματος 2.10.
- 2.7 Κάνετε τις απαραίτητες μετατροπές στο πρόγραμμα `Lissajous.f` ώστε ο χρήστης να μπορεί να δίνει διαφορετικό πλάτος και αρχική φάση στην

κίνηση που γίνεται στη διεύθυνση του άξονα των y . Μελετήστε τις τροχιές όταν το πλάτος είναι ίσο και η διαφορά φάσης είναι $\pi/4, \pi/2, \pi, -\pi$ και το ίδιο όταν το πλάτος στη διεύθυνση του άξονα των y είναι διπλάσιο.

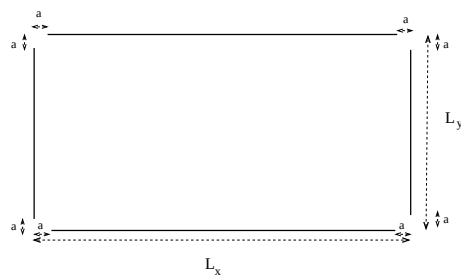
- 2.8 Κάνετε τις απαραίτητες μετατροπές στο πρόγραμμα `ProjectileAirResistance.f` ώστε να μπορεί να συμπεριλάβει και τη μελέτη της περίπτωσης $k = 0$.
- 2.9 Κάνετε τις απαραίτητες μετατροπές στο πρόγραμμα `ProjectileAirResistance.f` ώστε να υπολογίστε την κίνηση του υλικού σημείου στο χώρο. Κάνετε τα διαγράμματα της θέσης/ταχύτητας συναρτήσει του χρόνου και της τρισδιάστατης τροχιάς με την εντολή `splot` στο `gnuplot`. Χρησιμοποιήστε το πρόγραμμα `animate3D.gnu` για να κάνετε animation της τροχιάς.
- 2.10 Κάνετε τις απαραίτητες μετατροπές στο πρόγραμμα `ChargeInB.f` ώστε να υπολογίζει τον αριθμό των πλήρων στροφών που έχει κάνει η προβολή της τροχιάς του φορτίου στο επίπεδο $x - y$.
- 2.11 Μεταβάλλετε το πρόγραμμα `box1D_1.f` έτσι ώστε να τυπώνει στο `stdout` τον αριθμό των χρούσεων του υλικού σημείου στο αριστερό τοίχωμα, στο δεξί και το συνολικό.
- 2.12 Επαναλάβετε το ίδιο για το πρόγραμμα `box1D_2.f`. Συμπληρώστε σε δύο στήλες στον πίνακα της Σελ. 122 το συνολικό αριθμό των χρούσεων που προκύπτουν και σχολιάστε.
- 2.13 Στο πρόγραμμα `box1D_1.f` επιλέξτε $L = 10$, $v_0 = 1$. Ελαττώστε το dt μέχρι το υλικό σημείο να σταματήσει να κινείται. Για ποια τιμή του dt συμβαίνει αυτό; Αυξήστε το $v_0 = 10, 100$. Μέχρι ποια τιμή του dt μπορείτε να φτάσετε τώρα; Γιατί;
- 2.14 Στο πρόγραμμα `box1D_1.f` αλλάξτε τις δηλώσεις `REAL` $\rightarrow$ `REAL*8` και στις σταθερές προσθέστε τον εκθέτη `D0` (λ.χ. $0.0 \rightarrow 0.0D0$). Μελετήστε τη διαφορά στα αποτελέσματα που πήρατε στην ανάλυση της παραγράφου 2.3.2. Επαναλάβετε την άσκηση 2.13. Τι παρατηρείτε;
- 2.15 Κάντε το ίδιο στα προγράμματα `box1D_2.f`, `box1D_3.f` και επαναλάβετε την ανάλυση της παραγράφου 2.3.2.
- 2.16 Μεταβάλλετε το πρόγραμμα `box1D_1.f` ώστε να μελετήσετε την περίπτωση που το κινητό συγκρούεται μη ελαστικά με τα τοιχώματα του κουτιού, έτσι ώστε $v' = -ev$, $0 < e \leq 1$.

- 2.17 Μεταβάλλετε το πρόγραμμα `box2D_1.f` ώστε να μελετήσετε την περίπτωση που το κινητό συγκρούεται μη ελαστικά με τα τοιχώματα του κουτιού, έτσι ώστε $v'_x = -ev_x$, $v'_y = -ev_y$, $0 < e \leq 1$.
- 2.18 Χρησιμοποιείτε τη μέθοδο υπολογισμού του χρόνου που βρίσκεται στα προγράμματα `box1D_4.f`, `box1D_5.f`, `box1D_6.f` και αναπαράγετε τα αποτελέσματα του Σχήματος 2.21.
- 2.19 Κινητό πέφτει ελεύθερα στην κατακόρυφη διεύθυνση ξεκινώντας από ηρεμία από ύψος h . Στο δάπεδο υφίσταται μη ελαστική κρούση έτσι ώστε μετά την κρούση $v'_y = -ev_y$ με $0 < e \leq 1$ παράμετρος. Μελετήστε γράφοντας κατάλληλο πρόγραμμα την κίνηση για $e = 0.1, 0.5, 0.9, 1.0$.
- 2.20 Γενικεύστε το προηγούμενο πρόγραμμα για $\vec{v}_0 = v_{0x} \hat{x}$. Κάνετε το animation της τροχιάς.
- 2.21 Μελετήστε την κίνηση υλικού σημείου στο κουτί του Σχήματος 2.27. Μετρήστε σε κάθε βολή πόσες κρούσεις παίρνετε μέχρι το υλικό σημείο να βγει από το κουτί.

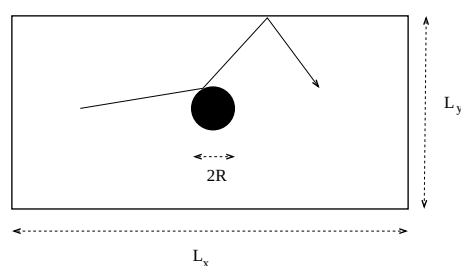


Σχήμα 2.27: Άσκηση 2.21.

- 2.22 Μελετήστε την κίνηση υλικού σημείου στο ‘μπιλιάρδο’ του Σχήματος 2.28. Μετρήστε σε κάθε βολή πόσες κρούσεις έχετε μέχρι να πετύχετε μία ‘τρύπα’. Το πρόγραμμα να σημειώνει σε ποια τρύπα μπήκε η μπίλια.
- 2.23 Προγραμματίστε την κίνηση ενός υλικού σημείου μέσα στο επίπεδο κουτί του Σχήματος 2.29. Στο κέντρο του υπάρχει κύκλος πάνω στον οποίο το υλικό σημείο σχεδιάζεται ελαστικά (Υποδ.: Χρησιμοποιείτε την υπορουτίνα `reflectVonCircle` του προγράμματος `Cylinder3D.f`).

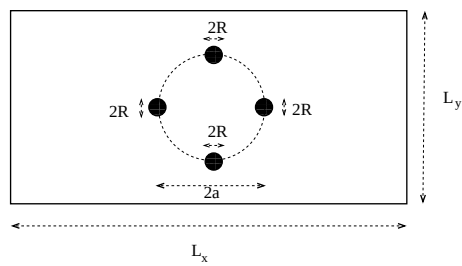


Σχήμα 2.28: Άσκηση 2.22.

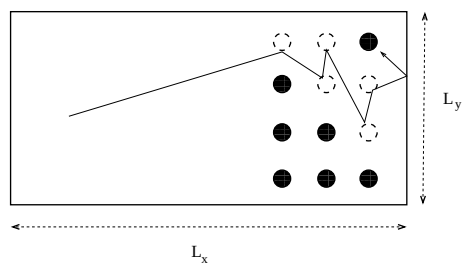


Σχήμα 2.29: Άσκηση 2.23.

- 2.24 Τοποθετήστε στο κουτί της προηγούμενης άσκησης 4 κύκλους πάνω στους οποίους θα ανακλάται το υλικό σημείο (Σχήμα 2.30).
- 2.25 Θεωρήστε τη διάταξη του Σχήματος 2.31. Κάθε φορά που το υλικό σημείο σκεδάζεται σε ένα κύκλο, ο κύκλος 'εξαφανίζεται'. Η κίνηση σταματάει όταν όλοι οι κύκλοι εξαφανιστούν. Κάθε φορά που το υλικό σημείο χτυπάει τον αριστερό τοίχο χάνετε ένα πόντο. Προσπαθήστε να διαλέξετε τροχιές που ελαχιστοποιούν τους χαμένους πόντους.



Σχήμα 2.30: Άσκηση 2.24.



Σχήμα 2.31: Άσκηση 2.25.

Κεφάλαιο 3

Κίνηση Σωματιδίου

Στο κεφάλαιο αυτό μελετάται αριθμητικά η επίλυση των κλασικών εξισώσεων κίνησης μονοδιάστατων μηχανικών συστημάτων, όπως λ.χ. αυτή του σημειακού σωματιδίου σε μια ευθεία, του απλού εκκρεμούς κλπ. Γίνεται εισαγωγή σε μεθόδους αριθμητικής ολοκλήρωσης διαφορικών εξισώσεων με αρχικές συνθήκες και ιδιαίτερα στη μέθοδο Runge–Kutta 4ης τάξης. Τέλος, μελετώνται τα συστήματα του αρμονικού ταλαντωτή και του απλού εκκρεμούς με απόσβεση και οδήγηση από εξωτερική χρονοεξαρτημένη δύναμη. Το τελευταίο σύστημα είναι μη γραμμικό και γίνεται μια εισαγωγή στις χαοτικές ιδιότητές του.

3.1 Αριθμητική Ολοκλήρωση Εξισώσεων Νεύτωνα

Το πρόβλημα της λύσης των εξισώσεων κίνησης σωματιδίου υπό την επίδραση δυνάμεων που δίνονται από το νόμο του Νεύτωνα μπορούν να γραφτούν υπό τη μορφή

$$\frac{d^2\vec{x}}{dt^2} = \vec{a}(t, \vec{x}, \vec{v}), \quad (3.1)$$

όπου

$$\vec{a}(t, \vec{x}, \vec{v}) \equiv \frac{\vec{F}}{m} \quad \vec{v} = \frac{d\vec{x}}{dt}. \quad (3.2)$$

Η κλάση των προβλημάτων που θα θεωρήσουμε είναι προβλήματα αρχικών τιμών, δηλ. δίνονται οι αρχικές συνθήκες

$$\vec{x}(t_0) = \vec{x}_0 \quad \vec{v}(t_0) = \vec{v}_0, \quad (3.3)$$

οι οποίες προσδιορίζουν μία μοναδική λύση $\vec{x}(t)$. Η διαφορικές εξισώσεις (3.1) είναι δεύτερης τάξης ως προς τις συναρτήσεις $\vec{x}(t)$. Για την αριθμητική λύση τους είναι βολικό να ανάγουμε τις εξισώσεις αυτές σε ένα σύστημα από διπλάσιο αριθμό εξισώσεων πρώτου βαθμού:

$$\frac{d\vec{x}}{dt} = \vec{v} \quad \frac{d\vec{v}}{dt} = \vec{a}(t, \vec{x}, \vec{v}). \quad (3.4)$$

Ειδικά θα ενδιαφερόμαστε για την κίνηση σωματιδίου πάνω στην ευθεία (1 διάσταση) και το επίπεδο (2 διαστάσεις) οπότε το σύστημα των εξισώσεων γίνεται

$$\frac{dx}{dt} = v \quad \frac{dv}{dt} = a(t, x, v) \quad \text{1-διάσταση}, \quad (3.5)$$

$$\begin{aligned} \frac{dx}{dt} &= v_x & \frac{dv_x}{dt} &= a_x(t, x, v_x, y, v_y) \\ \frac{dy}{dt} &= v_y & \frac{dv_y}{dt} &= a_y(t, x, v_x, y, v_y), \end{aligned} \quad \text{2-διαστάσεις} \quad (3.6)$$

3.2 Πρελούδιο: Μέθοδοι Euler

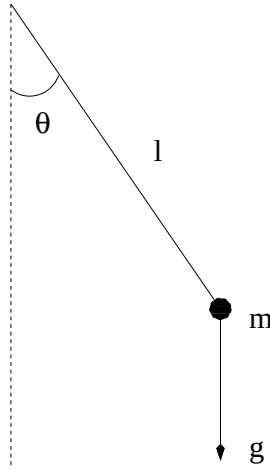
Για να πάρουμε μια πρώτη γεύση του προβλήματος θα μελετήσουμε το πρόβλημα του εκκρεμούς μήκους l μέσα σε ομογενές πεδίο βαρύτητας g (Σχήμα 3.1). Οι εξισώσεις κίνησης δίνονται από το σύστημα διαφορικών εξισώσεων

$$\begin{aligned} \frac{d^2\theta}{dt^2} &= -\frac{g}{l} \sin \theta \\ \frac{d\theta}{dt} &= \omega, \end{aligned} \quad (3.7)$$

που εύκολα ανάγεται στο σύστημα πρώτης τάξης

$$\begin{aligned} \frac{d\theta}{dt} &= \omega \\ \frac{d\omega}{dt} &= -\frac{g}{l} \sin \theta, \end{aligned} \quad (3.8)$$

Το παραπάνω σύστημα πρέπει να γραφτεί σε διακριτή μορφή έτσι ώστε να επιτευχθεί η αριθμητική του επίλυση με τη βοήθεια υπολογιστή. Ο πιο απλός τρόπος είναι να θεωρήσουμε την ολοκλήρωση του συστήματος από τον αρχικό



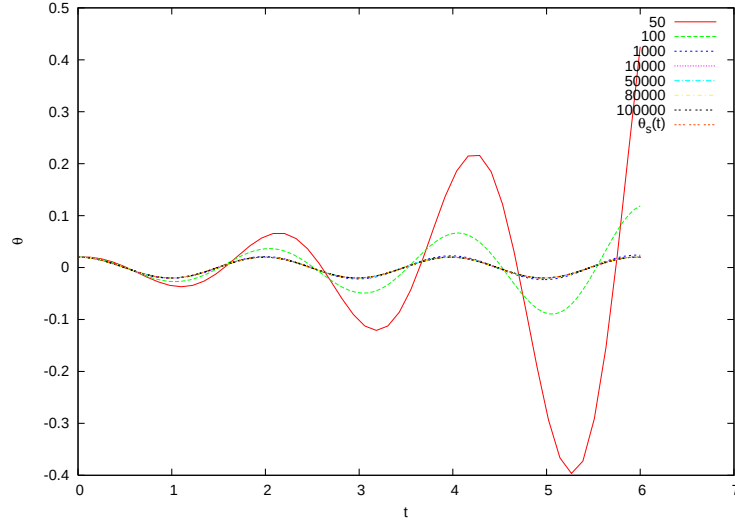
Σχήμα 3.1: Το εκκρεμές μήκους l μέσα σε ομογενές πεδίο βαρύτητας g .

χρόνο $t_0 = 0$ μέχρι τελικό χρόνο t_f χωρίζοντας το χρονικό διάστημα $t_f - t_0$ σε $N - 1$ ίσα διαστήματα πλάτους¹ $\Delta t \equiv h$, όπου $h = (t_f - t_0)/(N - 1)$ και προσεγγίζοντας τις παραγώγους από τις σχέσεις $(x_{n+1} - x_n)/\Delta t \approx x'_n$:

$$\begin{aligned}\omega_{n+1} &= \omega_n + \alpha_n \Delta t \\ \theta_{n+1} &= \theta_n + \omega_n \Delta t.\end{aligned}\tag{3.9}$$

όπου $\alpha = -(g/l) \sin \theta$ η γωνιακή επιτάχυνση. Η μέθοδος αυτή ακούει στο όνομα ‘μέθοδος Euler’. Το σφάλμα που εισάγεται από τη μέθοδο σε κάθε βήμα είναι της τάξης του $(\Delta t)^2$. Πράγματι αυτό προκύπτει από απλή ανάπτυξη κατά Taylor γύρω από το σημείο t_n αγνοώντας όλους τους όρους από τη δεύτερη παράγωγο και πάνω. Κατά την ολοκλήρωση από t_0 σε t_f , το συνολικό σφάλμα είναι τάξης Δt ! Ο λόγος είναι ότι τα σφάλματα αυτά προστίθενται σε κάθε βήμα και αφού ο αριθμός των βημάτων $N \sim 1/\Delta t$ το συνολικό σφάλμα είναι $\sim (\Delta t)^2 \times (1/\Delta t) = \Delta t$. Η μέθοδος Euler λέμε ότι είναι μια μέθοδος πρώτης τάξης και ως εκ τούτου έχει περιορισμένη ακρίβεια. Σαν να μην έφτανε αυτό η μέθοδος αυτή έχει προβλήματα ευστάθειας, ειδικά σε προβλήματα που παρουσιάζουν περιοδικότητα. Η μέθοδος είναι ασύμμετρη γιατί χρησιμοποιεί για την προώθηση της λύσης πληροφορία για τη συνάρτηση μόνο στην αρχή του διαστήματος $(t, t + \Delta t)$. Με μια απλή παραλλαγή παίρνουμε τη μέθοδο Euler–Cromer η οποία παρουσιάζει βελτιωμένη συμπεριφορά ως προς την ευστάθεια

¹Έχουμε N διακριτούς χρόνους $t_0, t_1, \dots, t_{N-2}, t_{N-1} \equiv t_f$



Σχήμα 3.2: Σύγκλιση της μεθόδου Euler για το απλό εκκρεμές με περίοδο $T \approx 1.987$ ($\omega^2 = 10.0$) για διαφορετικές τιμές του βήματος χρόνου Δt που καθορίζεται από τον αριθμό των βημάτων $\text{steps} = 50 - 100,000$. Η λύση είναι για $\theta_0 = 0.2$, $\omega_0 = 0.0$ και συγκρίνεται με τη γνωστή λύση για μικρές γωνίες με $\alpha(t) \approx -(g/l)\theta$.

αν και αυτή είναι πρώτης τάξης με συνολικό σφάλμα $\sim \Delta t$. Για το λόγο αυτό χρησιμοποιούμε για την προώθηση της γωνίας θ την καινούργια τιμή της γωνιακής ταχύτητας

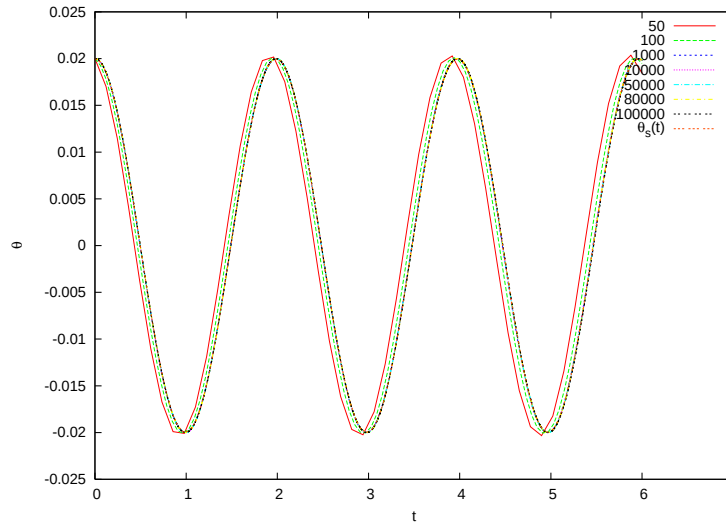
$$\begin{aligned}\omega_{n+1} &= \omega_n + \alpha_n \Delta t \\ \theta_{n+1} &= \theta_n + \omega_{n+1} \Delta t.\end{aligned}\tag{3.10}$$

Μια μέθοδος που βελτιώνει τους παραπάνω αλγόριθμους ως προς το σφάλμα διακριτοποίησης είναι ο αλγόριθμος Euler–Verlet ο οποίος δίνει ολικό σφάλμα² $\sim (\Delta t)^2$. Αυτός δίνεται από τις εξισώσεις

$$\begin{aligned}\theta_{n+1} &= 2\theta_n - \theta_{n-1} + \alpha_n (\Delta t)^2 \\ \omega_n &= \frac{\theta_{n+1} - \theta_{n-1}}{2\Delta t}.\end{aligned}\tag{3.11}$$

Η μέθοδος Euler–Verlet (3.11) είναι μια μέθοδος δύο βημάτων, αφού για την προώθηση της λύσης είναι αναγκαίο να γνωρίζουμε την τιμή της συνάρτησης σε δύο προηγούμενα βήματα. Άρα πρέπει να καθορίσουμε προσεκτικά τις

²Δείτε το Παράρτημα 3.7 για τις λεπτομέρειες.



Σχήμα 3.3: Σύγκλιση της μεθόδου Euler-Cromer, παρόμοια με το Σχήμα 3.2. Παρατηρούμε πως η μέθοδος συγκλίνει πολύ γρηγορότερα από την Euler.

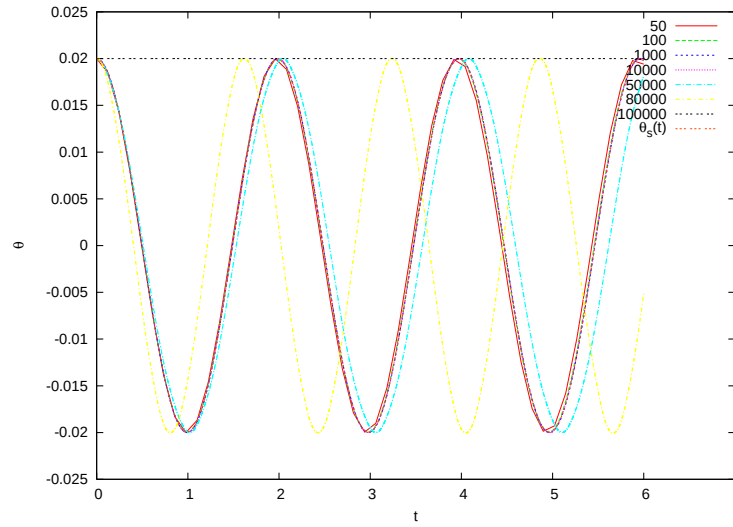
αρχικές συνθήκες για τα δύο πρώτα βήματα. Για το λόγο αυτό χρησιμοποιούμε τον αλγόριθμο Euler για να προωθήσουμε τις αρχικές συνθήκες ένα βήμα πίσω. Αν $\theta_1 = \theta(t_0)$, $\omega_1 = \omega(t_0)$ είναι οι αρχικές συνθήκες, τότε ορίζουμε

$$\theta_0 = \theta_1 - \omega_1 \Delta t + \frac{1}{2} \alpha_1 (\Delta t)^2. \quad (3.12)$$

Επίσης στο τελευταίο βήμα θα πρέπει να πάρουμε

$$\omega_N = \frac{\theta_N - \theta_{N-1}}{\Delta t}. \quad (3.13)$$

Παρόλο που η μέθοδος έχει μικρότερο συνολικό σφάλμα από τη μέθοδο Euler, το πρόβλημά της είναι ότι παρουσιάζεται ασταθής. Στη δεύτερη των εξισώσεων (3.11) η γωνιακή ταχύτητα προκύπτει από το λόγο δύο μικρών αριθμών, εκ των οποίων ο παρονομαστής είναι η διαφορά δύο μεγάλων αριθμών. Για μικρό χρόνο Δt , η πληροφορία βρίσκεται αρκετά δεκαδικά ψηφία πέρα από τα αρχικά και αφού ο υπολογιστής έχει πεπερασμένη ακρίβεια σε κάποια στιγμή η ακρίβεια της πληροφορίας γίνεται πολύ μικρή ή και χάνεται τελείως. Στην πρώτη των εξισώσεων (3.11), ο όρος $\alpha_n \Delta t^2$ είναι κατά μία τάξη ως προς Δt μικρότερος από τον αντίστοιχο όρο $\alpha_n \Delta t$ της μεθόδου Euler. Μειώνοντας το Δt , γρήγορα

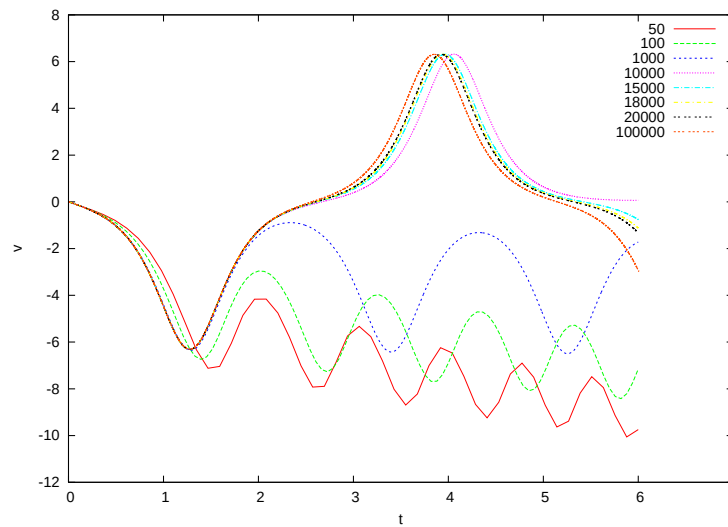


Σχήμα 3.4: Σύγκλιση της μεθόδου Euler-Verlet, παρόμοια με το Σχήμα 3.2. Παρατηρούμε πως η μέθοδος συγκλίνει πολύ γρηγορότερα από την Euler αλλά τα σφάλματα στρογγυλοποίησης κάνουν τη μέθοδο άχρηστη για $\text{steps} \gtrsim 50,000$ (προσέξτε τι γίνεται για $\text{steps} = 100,000$. Γιατί;).

έχουμε $\alpha_n \Delta t^2 \ll 2\theta_n - \theta_{n-1}$ και η ακρίβεια της μεθόδου εκμηδενίζεται λόγω της πεπερασμένης ακρίβειας των πραγματικών αριθμών στη μνήμη του υπολογιστή.

Ο προγραμματισμός των μεθόδων αυτών είναι ιδιαίτερα απλός. Γράφουμε ένα πρόγραμμα που θα κάνει σύγκριση και των τριών μεθόδων, Euler, Euler-Cromer και Euler-Verlet. Για το λόγο αυτό το κυρίως πρόγραμμα είναι απλά ένα interface με το χρήστη και για τους υπολογισμούς καλούνται διαφορετικές υπορουτίνες `euler`, `euler_cromer` και `euler_verlet`. Ο χρήστης απλά πρέπει να προγραμματίσει την κοινή σε όλους του υπολογισμούς συνάρτηση `accel(x)` που δίνει τη γωνιακή επιτάχυνση συναρτήσει της γωνίας θ (εδώ η μεταβλητή `REAL x`). Στην παράγραφο αυτή παίρνουμε `accel(x) = -10.0 * sin(x)`.

Η δομή των δεδομένων είναι πολύ απλή: Σε τρία arrays `REAL T(P)`, `X(P)` και `V(P)` αποθηκεύονται αντίστοιχα οι χρόνοι t_n , γωνιακές θέσεις θ_n και γωνιακές ταχύτητες ω_n για $n = 1, \dots, \text{steps}$. Ο χρήστης προσδιορίζει το χρονικό διάστημα της ολοκλήρωσης από $t_i = 0$ σε $t_f = Tfi$ καθώς και τον αριθμό των βημάτων ολοκλήρωσης `steps` τα οποία πρέπει να είναι λιγότερα από το μέγεθος `P` των arrays. Δίνει τις αρχικές συνθήκες $\theta_0 = Xin$ και $\omega_0 = Vin$. Στη συνέχεια οι υπορουτίνες των μεθόδων καλούνται και στην είσοδο τους

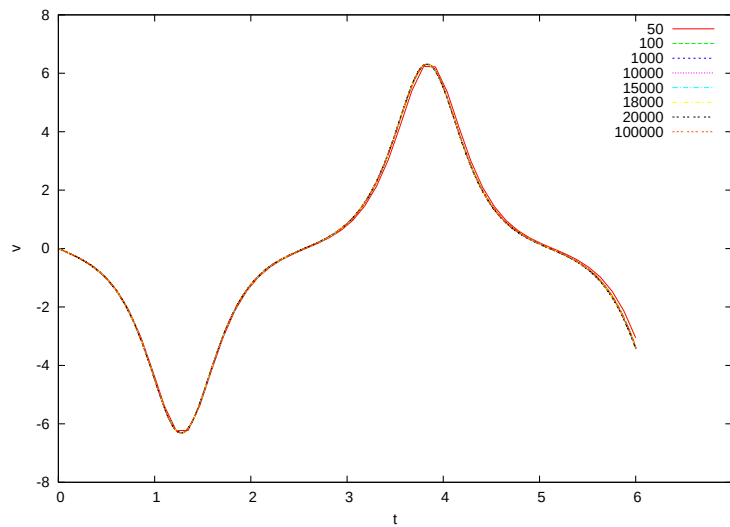


Σχήμα 3.5: Σύγκλιση της μεθόδου Euler για το απλό εκκρεμές παρόμοια με το Σχήμα 3.2 αλλά για $\theta_0 = 3.0$, $\omega_0 = 0.0$. Εδώ δείχνουμε τη συμπεριφορά της γωνιακής ταχύτητας και παρατηρούμε μεγάλη αστάθεια για $\text{steps} \lesssim 1,000$.

παρέχουμε τις αρχικές συνθήκες, διάστημα ολοκλήρωσης και αριθμό βημάτων $X_{in}, V_{in}, T_{fi}, \text{steps}$ ενώ στην έξοδο μας παρέχουν τα αποτελέσματα αποθηκευμένα στα arrays T, X, V . Στη συνέχεια τα αποτελέσματα τυπώνονται στα αρχεία `euler.dat`, `euler_cromer.dat` και `euler_verlet.dat`.

Οι υπορουτίνες υπολογισμού ακολουθούν τη διαδικασία καθορισμού των αρχικών συνθηκών, υπολογισμού του χρόνου $\Delta t \equiv h = T_{fi}/(\text{steps} - 1)$ και εκτελούν τους βρόχους που θα προωθούν τη λύση με βήμα Δt . Σε κάθε βήμα τα αποτελέσματα αποθηκεύονται στα arrays T, X, V . Έτσι το απλό τμήμα του κώδικα

```
T(1) = 0.0
X(1) = Xin
V(1) = Vin
h     = Tfi/(steps-1)
do i = 2,steps
  T(i) = T(i-1)+h
  X(i) = X(i-1)+V(i-1)*h
  V(i) = V(i-1)+accel(X(i-1))*h
enddo
```



Σχήμα 3.6: Σύγκλιση της μεθόδου Euler-Cromer, παρόμοια με το Σχήμα 3.5. Παρατηρούμε πως η μέθοδος συγκλίνει πολύ γρηγορότερα από την Euler.

εκτελεί τις απαραίτητες εντολές για τη μέθοδο Euler. Λίγη προσοχή πρέπει να δοθεί στη μέθοδο Euler-Verlet όπου πρέπει να καθοριστούν τα δύο πρώτα βήματα, καθώς και το τελευταίο για την ταχύτητα:

```

T(1)      = 0.0
X(1)      = Xin
V(1)      = Vin
X0        = X(1) - V(1) * h + accel(X(1)) * h*h/2.0
T(2)      = h
X(2)      = 2.0*X(1) - X0 + accel(X(1)) * h*h
do i      = 3,steps
.....
enddo
V(steps)= (X(steps)-X(steps-1))/h

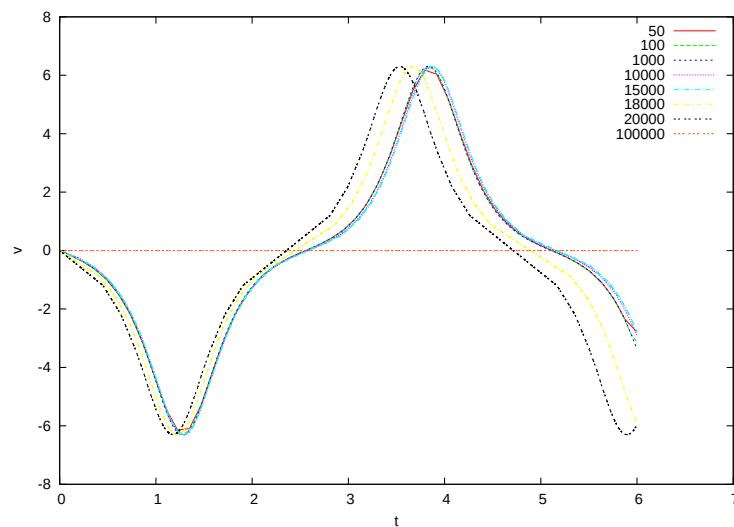
```

Για διευκόλυνση του αναγνώστη παραθέτουμε ολόκληρο το πρόγραμμα παρακάτω:

```

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Program to integrate equations of motion for accelerations

```



Σχήμα 3.7: Σύγκλιση της μεθόδου Euler-Verlet, παρόμοια με το Σχήμα 3.5. Παρατηρούμε πως η μέθοδος συγκλίνει πολύ γρηγορότερα από την Euler αλλά τα σφάλματα στρογγυλοποίησης κάνουν τη μέθοδο πολύ γρήγορα ασταθής για $\text{steps} \gtrsim 18,000$.

```

C      which are functions of x with the method of Euler, Euler-Cromer
C      and Euler-Verlet.
C      The user sets initial conditions and the subroutines return
C      X(t) and V(t)=dX(t)/dt in arrays T(1..STEPS),X(1..STEPS),V(1..STEPS)
C      The user provides number of integration STEPS and the final
C      time TFI. Initial time is assumed to be t_0=0 and the integration
C      step h = TFI/(STEPS-1)
C      The user programs a real function accel(x) which gives the
C      acceleration dV(t)/dt as function of X.
C      NOTE: T(1) = 0 T(STEPS) = TFI and there are STEPS-1 additional
C      steps after the initial point
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      program diff_eq_euler
C      implicit none                                ! We force ourselves to declare all variables
C      integer P                                    ! The size of the arrays, should be larger
C      parameter(P=110000)                          ! than number of steps
C      real T(P),X(P),V(P)                          ! time t,x(t),v(t)=dx/dt
C      real Xin,Vin,Tfi                             ! initial conditions

```

```

integer steps,i
!t_0 = 0.0

C   The user provides initial conditions X_0,V_0 final time t_f and
C   number of steps:
print *, 'Enter X_0,V_0,t_f,number of steps (t_0=0):'
read(5,*)Xin,Vin,Tfi,steps
C   This check is necessary to avoid memory violations:
if(steps .ge. P )then
    print *, 'steps must be strictly less than P. steps,P= ',steps,P
    stop
endif

C   Xin= X(1), Vin=V(1), T(1)=0 and the routine gives evolution in
C   T(2..STEPS), X(2..STEPS), V(2..STEPS) which we print in a file
call euler(Xin,Vin,Tfi,steps,T,X,V)
open(unit=20,file="euler.dat") !filename euler.dat given here
do i=1,steps
C   Each line in data file has time, position, velocity:
    write(20,*) T(i),X(i),V(i)
enddo
close(20) !we close the unit to be reused below

C   We repeat everything for each method
call euler_cromer(Xin,Vin,Tfi,steps,T,X,V)
open(unit=20,file="euler_cromer.dat")
do i=1,steps
    write(20,*) T(i),X(i),V(i)
enddo
close(20)

call euler_verlet(Xin,Vin,Tfi,steps,T,X,V)
open(unit=20,file="euler_verlet.dat")
do i=1,steps
    write(20,*) T(i),X(i),V(i)
enddo
close(20)

```



```

end

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Function which returns the value of acceleration at
C      position x used in the integration subroutines
C      euler, euler_cromer and euler_verlet
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      real function accel(x)
C      implicit none
C      real x
C      accel = -10.0*sin(x)
C      end

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Driver routine for integrating equations of motion
C      using the Euler method
C      Input:
C      Xin=X(1), Vin=V(1) -- initial condition at t=0,
C      Tfi the final time and steps the number of steps of integration
C      (the initial point is counted as the first one)
C      Output:
C      The arrays T(1..steps), X(1..steps), V(1..steps) which
C      gives x(t_i)=X(i), dx/dt(t_i)=V(i), t_i=T(i) i=1..steps
C      where for i=1 we have the initial condition.
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      subroutine euler(Xin,Vin,Tfi,steps,T,X,V)
C      implicit none
C      integer P
C      parameter(P=110000)
C      real T(P),X(P),V(P) !time t,x(t),v(t)=dx/dt
C      real Xin,Vin,Tfi
C      integer steps,i
C      real h,accel      !**Note: we have to declare the function accel**
C      Initial conditions set here:
C      T(1) = 0.0
C      X(1) = Xin
C      V(1) = Vin

```

```

C      h is the time step Dt
C      h      = Tfi/(steps-1)
C      do i = 2,steps
C          T(i) = T(i-1)+h          ! time advances by Dt
C          X(i) = X(i-1)+V(i-1)*h    ! advancement and storage of position
C          V(i) = V(i-1)+accel(X(i-1))*h !... and velocity. Here we call accel(
C      enddo

C      end

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Driver routine for integrating equations of motion
C      using the Euler-Cromer method
C      Input:
C      Xin=X(1), Vin=V(1) -- initial condition at t=0,
C      Tfi the final time and steps the number of steps of integration
C      (the initial point is counted as the first one)
C      Output:
C      The arrays T(1..steps), X(1..steps), V(1..steps) which
C      gives x(t_i)=X(i), dx/dt(t_i)=V(i), t_i=T(i) i=1..steps
C      where for i=1 we have the initial condition.
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      subroutine euler_cromer(Xin,Vin,Tfi,steps,T,X,V)
C      implicit none
C      integer P
C      parameter(P=110000)
C      real T(P),X(P),V(P) !time t,x(t),v(t)=dx/dt
C      real Xin,Vin,Tfi
C      integer steps,i
C      real h,accel

C      T(1) = 0.0
C      X(1) = Xin
C      V(1) = Vin
C      h      = Tfi/(steps-1)
C      do i = 2,steps
C          T(i) = T(i-1)+h
C          V(i) = V(i-1)+accel(X(i-1))*h

```

```

      X(i) = X(i-1)+V(i)*h  !here is the difference compared to Euler
    enddo

  end

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Driver routine for integrating equations of motion
C      using the Euler - Verlet method
C      Input:
C      Xin=X(1), Vin=V(1) -- initial condition at t=0,
C      Tfi the final time and steps the number of steps of integration
C      (the initial point is counted as the first one)
C      Output:
C      The arrays T(1..steps), X(1..steps), V(1..steps) which
C      gives x(t_i)=X(i), dx/dt(t_i)=V(i), t_i=T(i) i=1..steps
C      where for i=1 we have the initial condition.
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      subroutine euler_verlet(Xin,Vin,Tfi,steps,T,X,V)
      implicit none
      integer P
      parameter(P=110000)
      real T(P),X(P),V(P) !time t,x(t),v(t)=dx/dt
      real Xin,Vin,Tfi
      integer steps,i
      real h,g_over_l
      parameter(g_over_l=10.0)
      real h2,X0,o2h
      real accel

C      Initial conditions set here:
      T(1)    = 0.0
      X(1)    = Xin
      V(1)    = Vin
      h       = Tfi/(steps-1)    ! time step
      h2      = h*h               ! time step squared
      o2h     = 0.5/h             ! h/2
C      We have to initialize one more step: X0 corresponds to 'X(0)'
      X0      =      X(1)    - V(1) * h + accel(X(1))  *h2/2.0

```

```

      T(2)      = h
      X(2)      = 2.0*X(1) - X0          + accel(X(1)) *h2
C      Now i starts from 3:
      do i      = 3,steps
      T(i)      = T(i-1)+h
      X(i)      = 2.0*X(i-1) - X(i-2)    + accel(X(i-1))*h2
      V(i-1)    = o2h * (X(i)-X(i-2))
      enddo
C      Notice that we have one more step for the velocity:
      V(steps)= (X(steps)-X(steps-1))/h

      end

```

Η μεταγλώττιση και το τρέξιμο του προγράμματος γίνονται πολύ απλά με τις εντολές:

```

> f77 euler.f -o euler
> ./euler
Enter X_0,V_0,t_f,number of steps (t_0=0):
0.2 0.0 6.0 1000
> ls euler*.dat
euler_cromer.dat  euler.dat  euler_verlet.dat
> head -n 5 euler.dat
0.000000      0.2000000      0.000000
6.0060062E-03 0.2000000      -1.1932093E-02
1.2012012E-02 0.1999283      -2.3864185E-02
1.8018018E-02 0.1997850      -3.5792060E-02
2.4024025E-02 0.1995700      -4.7711499E-02

```

Η τελευταία εντολή μας δείχνει τις 5 πρώτες γραμμές του αρχείου `euler.dat` όπου βλέπουμε τις 3 στήλες με το χρόνο, θέση και ταχύτητα που δίνει η μέθοδος. Για να δούμε γραφικά τα αποτελέσματα μπορούμε να χρησιμοποιήσουμε το πρόγραμμα `gnuplot`³:

```

> gnuplot
      G N U P L O T
      Version 4.0 patchlevel 0

```

³www.gnuplot.info

last modified Thu Apr 15 14:44:22 CEST 2004

```
.....
gnuplot> plot "euler.dat" using 1:2 with lines
gnuplot> plot "euler.dat" using 1:3 with lines
```

κάνει απλές γραφικές παραστάσεις των θέσεων και ταχυτήτων αντίστοιχα συναρτήσει του χρόνου. Στην τελευταία μπορούμε να προσθέσουμε και τα αποτελέσματα των άλλων μεθόδων δίνοντας στη συνέχεια τις εντολές:

```
gnuplot> replot "euler_cromer.dat" using 1:3 with lines
gnuplot> replot "euler_verlet.dat" using 1:3 with lines
```

Τα αποτελέσματα φαίνονται στα Σχήματα 3.2–3.7. Παρατηρούμε ότι η μέθοδος Euler είναι ασταθής εκτός αν πάρουμε το βήμα χρόνου πάρα πολύ μικρό. Η μέθοδος Euler–Verlet έχει πολύ καλύτερη συμπεριφορά. Τα αποτελέσματα συγκλίνουν γρήγορα και παραμένουν σταθερά και για μεγάλο αριθμό βημάτων $\text{steps} \sim 100,000$. Η μέθοδος Euler–Verlet συγκλίνει γρήγορα αλλά σύντομα παρατηρούμε τα σφάλματα συσσώρευσης γίνονται φανερά εξαιτίας της Σχέσης (3.11). Το φαινόμενο αυτό είναι εντονότερο για αρχικές συνθήκες με μεγάλη αρχική γωνιακή απόκλιση, όπως φαίνεται στο Σχήμα 3.7. Στα Σχήματα 3.2–3.4, όπου η αρχική γωνιακή απόκλιση είναι μικρή, συγκρίνουμε τη λύση που παίρνουμε με τη λύση για το αρμονικό εκκρεμές ($\sin(\theta) \approx \theta$):

$$\begin{aligned}\alpha(\theta) &= -\frac{g}{l}\theta \equiv -\Omega^2\theta \\ \theta(t) &= \theta_0 \cos(\Omega t) + (\omega_0/\Omega) \sin(\Omega t) \\ \omega(t) &= \omega_0 \cos(\Omega t) - (\theta_0\Omega) \sin(\Omega t),\end{aligned}\tag{3.14}$$

όπου παρατηρούμε ταύτιση για τις τιμές του Δt που οι μέθοδοι συγκλίνουν. Με τον τρόπο αυτό ελέγχουμε τον κώδικα για πιθανά σφάλματα και για την ορθότητα των αποτελεσμάτων. Για το γράφημα των παραπάνω συναρτήσεων μπορούμε να δώσουμε τις παρακάτω εντολές στο πρόγραμμα gnuplot:

```
gnuplot> set dummy t #makes t the independent variable in functions
gnuplot> omega2 = 10
gnuplot> X0 = 0.2
gnuplot> V0 = 0.0
gnuplot> omega = sqrt(omega2)
gnuplot> x(t) = X0 * cos(omega * t) +(V0/omega)*sin(omega*t)
gnuplot> v(t) = V0 * cos(omega * t) -(omega*X0)*sin(omega*t)
gnuplot> plot x(t), v(t)
```

Η σύγκριση των αποτελεσμάτων με τα θεωρητικά, ιδιαίτερα όταν οι διαφορές δεν διακρίνονται με γυμνό μάτι ή όταν η ποσοτική ανάλυση είναι επιθυμητή, μπορεί να γίνει αναπαριστώντας γραφικά τις διαφορές των υπολογισθέντων από τις θεωρητικές τιμές. Οι σχετικές γραφικές παραστάσεις γίνονται με τις εντολές:

```
gnuplot> plot "euler.dat" using 1:($2-x($1)) with lines
gnuplot> plot "euler.dat" using 1:($3-v($1)) with lines
```

Οι εντολή `using 1:($2-x($1))` σε απλά ελληνικά λέει ‘Κάνε τη γραφική παράσταση χρησιμοποιώντας στο άξονα των y την τιμή της 2ης στήλης του `euler.dat` μείον την τιμή της συνάρτησης $x(t)$ για t ίσο με την αντίστοιχη τιμή της πρώτης στήλης’. Με τον τρόπο αυτό (και με μικρή μετατροπή για να υπολογίζουμε την απόλυτη τιμή της διαφοράς) φτιάχνουμε τα Σχήματα 3.11-3.14.

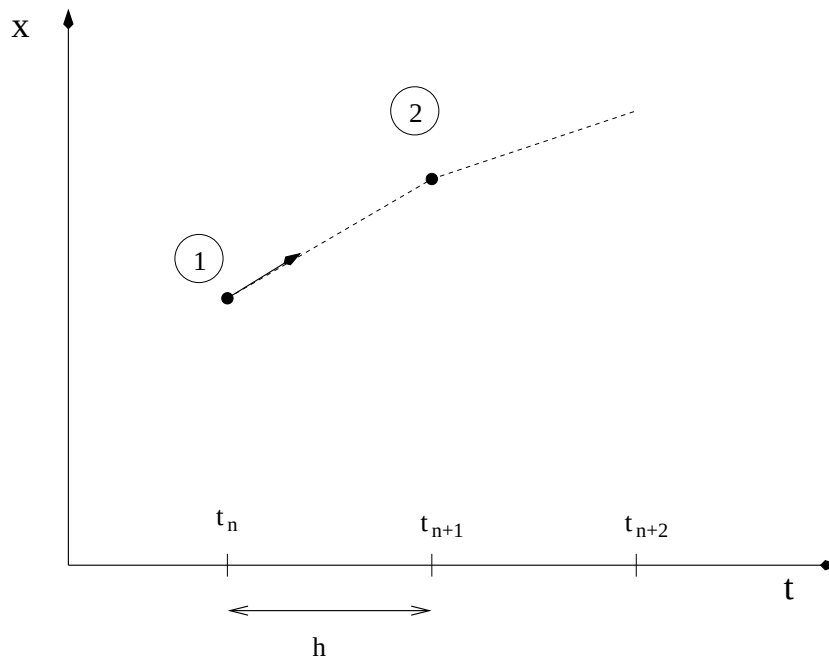
3.3 Μέθοδοι Runge–Kutta

Στην προηγούμενη παράγραφο είδαμε μία μέθοδο πεπερασμένων διαφορών ενός βήματος πρώτης τάξης, τη μέθοδο Euler. Αυτό σημαίνει πως όταν προσεγγίζουμε την ολοκλήρωση με N διακριτά βήματα από χρόνο t_i σε χρόνο t_f με βήμα $\Delta t \equiv h = (t_f - t_0)/N$ το σφάλμα διακριτοποίησης είναι τάξης $\sim \mathcal{O}(h)$. Γεννάται το ερώτημα αν είναι δυνατόν να βρεθεί αλγόριθμος ολοκλήρωσης ο οποίος να κάνει τα σφάλματα να είναι ανώτερης τάξης. Μια κλάση τέτοιων μεθόδων είναι οι μέθοδοι Runge–Kutta. Οι μέθοδοι αυτοί είναι επαγωγικοί ενός βήματος, δηλ. η επόμενη θέση προκύπτει απλά από τη γνώση της προηγούμενης. Αυτό είναι σε αντίθεση με μεθόδους δύο ή πολλαπλών βημάτων όπως η μέθοδος Euler–Verlet όπου η γνώση της επόμενης θέσης απαιτεί να γνωρίζουμε τη θέση του σωματιδίου για δύο προηγούμενα βήματα. Η μέθοδος Runge–Kutta τάξης p έχει σφάλμα $\sim \mathcal{O}(h^p)$. Αυτό σημαίνει ότι σε κάθε βήμα εισάγεται σφάλμα διακριτοποίησης τάξης $\sim \mathcal{O}(h^{p+1})$ αφού τότε το σφάλμα μετά από $N = (t_f - t_0)/\Delta t$ βήματα θα είναι τάξης

$$\sim \mathcal{O}(h^{p+1}) \times N = \mathcal{O}(h^{p+1}) \times \frac{t_f - t_0}{\Delta t} \sim \mathcal{O}(h^{p+1}) \times \frac{1}{h} = \mathcal{O}(h^p). \quad (3.15)$$

Ας θεωρήσουμε για απλότητα το πρόβλημα με μια άγνωστη συνάρτηση $x(t)$ η οποία εξελίσσεται στο χρόνο σύμφωνα με τη διαφορική εξίσωση:

$$\frac{dx}{dt} = f(t, x). \quad (3.16)$$



Σχήμα 3.8: Η γεωμετρία του βήματος της μεθόδου 1ης τάξης που δίνεται από την Εξίσωση (3.17) .

Ας δούμε πρώτα μία μέθοδο πρώτης τάξης. Η πιο αφελής προσέγγιση θα ήταν να πάρουμε την παράγωγο να μας δίνει την πεπερασμένη διαφορά, δηλ.

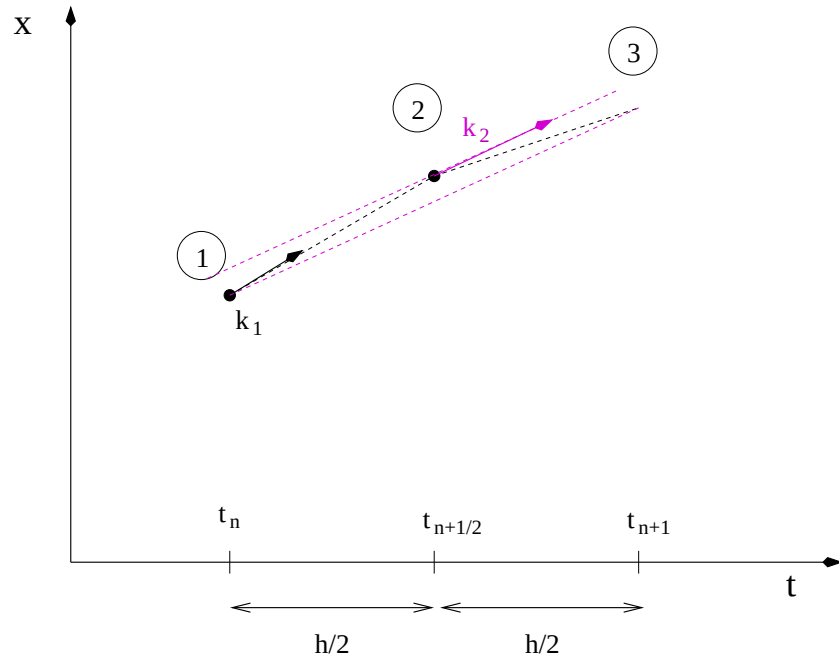
$$\frac{dx}{dt} \approx \frac{x_{n+1} - x_n}{\Delta t} = f(t_n, x_n) \Rightarrow x_{n+1} = x_n + h f(t_n, x_n) \quad (3.17)$$

Αναπτύσσοντας κατά Taylor βλέπουμε ότι το σφάλμα σε κάθε βήμα είναι $\mathcal{O}(h^2)$, άρα το σφάλμα για την εξέλιξη από $t_i \rightarrow t_f$ είναι $\mathcal{O}(h)$. Πράγματι

$$x_{n+1} = x(t_n + h) = x_n + h \frac{dx}{dt}(x_n) + \mathcal{O}(h^2) = x_n + h f(t_n, x_n) + \mathcal{O}(h^2). \quad (3.18)$$

Η γεωμετρία του βήματος φαίνεται στο Σχήμα 3.8. Επιλέγεται το σημείο 1 και από εκεί με γραμμική επέκταση στην κατεύθυνση της παραγώγου $k_1 \equiv f(t_n, x_n)$ προσδιορίζουμε το σημείο x_{n+1} .

Βελτίωση της μεθόδου προκύπτει αν πάρουμε ένα ενδιάμεσο σημείο 2. Αυτή η διαδικασία φαίνεται στο Σχήμα 3.9 και έγκειται στο να πάρουμε το ενδιάμεσο σημείο 2 στο μέσο του διαστήματος (t_n, t_{n+1}) με γραμμική προέκταση



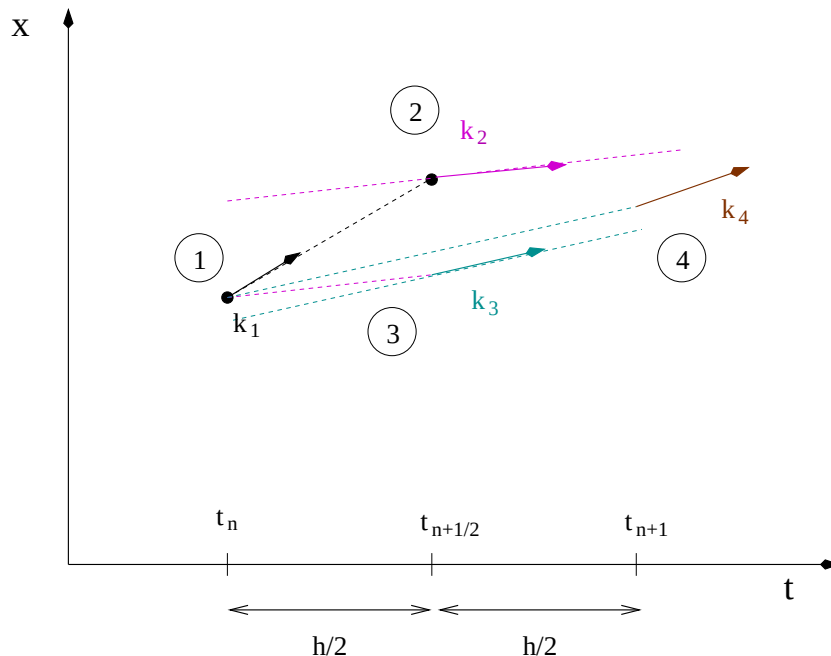
Σχήμα 3.9: Η γεωμετρία του βήματος της μεθόδου 2ης τάξης που δίνεται από την Εξίσωση (3.19) .

από το x_n χρησιμοποιώντας την κλίση που δίνεται από την παράγωγο στο x_n $k_1 \equiv f(t_n, x_n)$. Στη συνέχεια χρησιμοποιούμε ως εκτιμητή της παραγώγου στο διάστημα αυτό την κλίση στο σημείο 2 δηλ. $k_2 \equiv f(t_{n+1/2}, x_{n+1/2}) = f(t_n + h/2, x_n + (h/2)k_1)$ και τη χρησιμοποιούμε για να προεκτείνουμε γραμμικά από το x_n στο x_{n+1} . Συνοψίζοντας έχουμε

$$\begin{aligned} k_1 &= f(t_n, x_n) \\ k_2 &\equiv f\left(t_n + \frac{h}{2}, x_n + \frac{h}{2} k_1\right) \\ x_{n+1} &= x_n + h k_2. \end{aligned} \tag{3.19}$$

Αυτό που θα δείξουμε είναι ότι παρόλο που χρειάζεται να υπολογίσουμε τη συνάρτηση f δύο φορές σε κάθε βήμα, διπλασιάζοντας ουσιαστικά τον υπολογιστικό χρόνο, το σφάλμα στο βήμα (3.19) είναι $\mathcal{O}(h^3)$ άρα το συνολικό σφάλμα είναι $\mathcal{O}(h^2)$, οπότε αναγκαστικά η (3.19) θα υπερτερήσει της (3.17) σε ακρίβεια ακόμα και αν συγκριθούν σε δεδομένο υπολογιστικό χρόνο, δηλ. η

(3.19) σε βήμα h και η (3.17) σε $h/2$.⁴



Σχήμα 3.10: Η γεωμετρία του βήματος της μεθόδου 4ης τάξης που δίνεται από την Εξίσωση (3.20) .

Η βελτίωση γίνεται περισσότερο αισθητή με τη μέθοδο Runge-Kutta 4ης τάξης. Στην περίπτωση αυτή έχουμε 4 υπολογισμούς τη συνάρτησης f αλλά το συνολικό σφάλμα είναι τώρα $O(h^4)$, οπότε για τους ίδιους λόγους η μέθοδος θα υπερτερήσει τελικά σε ακρίβεια της (3.19)⁵. Η διαδικασία που θα ακολουθήσουμε εξηγείται γεωμετρικά στο Σχήμα 3.10. Χρησιμοποιούμε τώρα 3 ενδιάμεσα σημεία για την προώθηση από το x_n στο x_{n+1} . Αρχικά χρησιμοποιώντας την κλίση που δίνεται από την παράγωγο στο x_n $k_1 \equiv f(t_n, x_n)$, βρίσκουμε το ενδιάμεσο σημείο 2 στο μέσο του διαστήματος $(t_n, t_{n+1} = t_n + h)$ δηλ. $x_2 = x_n + (h/2)k_1$. Υπολογίζουμε την παράγωγο της συνάρτησης στο σημείο 2 δηλ. $k_2 \equiv f(t_n + h/2, x_n + (h/2)k_1)$ και τη χρησιμοποιούμε για να προεκτείνουμε γραμμικά από το x_n στο ενδιάμεσο σημείο 3, πάλι στο μέσο του

⁴Όπως φαίνεται στην πράξη, η (3.17) πάσχει και από προβλήματα σταθερότητας περισσότερο από την (3.19) οπότε συνίσταται να αποφεύγεται.

⁵Πάντα; Χμμμμ, όχι πάντα! Μεγαλύτερης τάξης δεν σημαίνει αναγκαστικά και μεγαλύτερης ακρίβειας αν και τις περισσότερες φορές αυτό είναι σωστό.

διαστήματος (t_n, t_{n+1}) , δηλ. $x_3 = x_n + (h/2)k_2$. Υπολογίζουμε την παράγωγο της συνάρτησης $k_3 \equiv f(t_n + h/2, x_n + (h/2)k_2)$ και τη χρησιμοποιούμε για να προεκτείνουμε προς το σημείο 4, το οποίο τώρα το παίρνουμε στο άκρο του διαστήματος δηλ. με $t_4 = t_n + h$, οπότε παίρνουμε $x_4 = x_n + hk_3$. Κάνουμε ένα τέταρτο υπολογισμό της παραγώγου $k_4 \equiv f(t_n + h, x_n + hk_3)$ και χρησιμοποιούμε και τις 4 παραγώγους k_1, k_2, k_3 και k_4 ως εκτιμητές της παραγώγου της συνάρτησης. Εκείνο που έδειξαν οι Runge–Kutta είναι ότι το σφάλμα διακριτοποίησης σε κάθε βήμα στην προώθηση της συνάρτησης γίνεται $\mathcal{O}(h^5)$ αν πάρουμε:

$$\begin{aligned} k_1 &= f(t_n, x_n) \\ k_2 &= f(t_n + \frac{h}{2}, x_n + \frac{h}{2} k_1) \\ k_3 &= f(t_n + \frac{h}{2}, x_n + \frac{h}{2} k_2) \\ k_4 &= f(t_n + h, x_n + h k_3) \\ x_{n+1} &= x_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned} \quad (3.20)$$

Ο δεύτερος όρος στην τελευταία εξίσωση είναι ένας μέσος όρος των 4 παραγώγων με το κατάλληλο βάρος ώστε να πετύχουμε την εξουδετέρωση των σφαλμάτων μέχρι τάξης h^5 .

Τέλος θα κλείσουμε συζητώντας μία απορία που πιθανώς θα έχει δημιουργηθεί στον αναγνώστη: Συζητήσαμε πώς είναι δυνατόν κανείς να μειώσει τα σφάλματα διακριτοποίησης χρησιμοποιώντας αλγόριθμους των οποίων τα σφάλματα αυτά μειώνονται κατα το δυνατόν γρηγορότερα με το βήμα στο χρόνο h , συνήθως σαν $\sim h^p$. Άρα μπορεί κανείς να υποθέσει ότι CPU χρόνου επιτρέποντος -το οποίο δεν αποτελεί σοβαρό εμπόδιο στις απλές περιπτώσεις- μπορούμε να πλησιάσουμε αυθαίρετα κοντά στην αναλυτική λύση σε επίπεδο ακρίβειας μηχανής. Αυτό όμως δεν είναι σωστό. Μια άλλη κατηγορία σφαλμάτων είναι τα σφάλματα στρογγυλοποίησης τα οποία προστίθενται σε κάθε βήμα εφαρμογής της μεθόδου. Αυτά συσσωρεύονται ανάλογα με τον αριθμό των βημάτων, οπότε για πολύ μικρό h , άρα και πολύ μεγάλο αριθμό βημάτων, αυτά θα γίνουν μεγαλύτερα από την επιθυμητή ακρίβεια. Αυτή η κατηγορία σφαλμάτων εξαρτάται από το hardware, τη γλώσσα προγραμματισμού ή/και το μεταγλωττιστή και τέλος από τον αλγόριθμο. Για το τελευταίο ας δώσουμε ένα παράδειγμα: Έστω ότι θέλουμε να υπολογίσουμε την παράγωγο μίας συνάρτησης από τη

σχέση

$$f'(t) = \frac{f(t+h) - f(t)}{h},$$

παίρνοντας το h αυθαίρετα μικρό. Αν υποθέσουμε ότι η παράγωγος και οι τιμές της συνάρτησης είναι πεπερασμένοι αριθμοί $\sim \mathcal{O}(1)$, τότε ο αριθμητής θα πρέπει να είναι $\sim \mathcal{O}(h)$. Όταν το h γίνει της τάξης της ακρίβειας των REAL (ή των REAL*8 κλπ) τότε ο αριθμητής που είναι η διαφορά δύο αριθμών περίπου ίσων και της τάξης της μονάδας θα αρχίσει να χάνει σημαντικά σε ακρίβεια σε σχέση με την πραγματική τιμή μέχρι που θα είναι ένας άχρηστος αριθμός. Αυτό συμβαίνει γενικά όταν αφαιρούμε αριθμούς περίπου ίσους ή όταν προσθέτουμε αριθμούς που η τάξη μεγέθους τους διαφέρει περισσότερο από την ακρίβεια του υπολογιστή. Άρα και όταν χρησιμοποιούμε τη σχέση

$$x_{n+1} = x_n + h\mathcal{O}(x_n)$$

για πολύ μικρό βήμα h και $\mathcal{O}(x)$ της τάξης μεγέθους του x_n , τα σφάλματα στρογγυλοποίησης θα είναι σημαντικά και θα αυξάνουν προσθετικά με τον αριθμό των βημάτων.

3.3.1 Προγραμματισμός της Runge-Kutta 4ης τάξης

Ας συζητήσουμε τώρα τον προγραμματισμό της μεθόδου Runge-Kutta 4ης τάξης για την περίπτωση της κίνησης ενός σωματιδίου σε μία διάσταση. Για το λόγο αυτό θα πρέπει να ολοκληρώσουμε το σύστημα διαφορικών εξισώσεων (3.5) που είναι ένα σύστημα εξισώσεων για τις δύο συναρτήσεις του χρόνου $x_1(t) \equiv x(t)$ και $x_2(t) \equiv v(t)$ για τις οποίες έχουμε

$$\frac{dx_1}{dt} = f_1(t, x_1, x_2) \quad \frac{dx_2}{dt} = f_2(t, x_1, x_2) \quad (3.21)$$

Στην περίπτωση αυτή η μέθοδος Runge-Kutta 4ης τάξης που δίνεται στην

εξίσωση (3.20) γενικεύεται ως εξής:

$$\begin{aligned}
 k_{11} &= f_1(t_n, x_{1,n}, x_{2,n}) \\
 k_{21} &= f_2(t_n, x_{1,n}, x_{2,n}) \\
 k_{12} &= f_1(t_n + \frac{h}{2}, x_{1,n} + \frac{h}{2} k_{11}, x_{2,n} + \frac{h}{2} k_{21}) \\
 k_{22} &= f_2(t_n + \frac{h}{2}, x_{1,n} + \frac{h}{2} k_{11}, x_{2,n} + \frac{h}{2} k_{21}) \\
 k_{13} &= f_1(t_n + \frac{h}{2}, x_{1,n} + \frac{h}{2} k_{12}, x_{2,n} + \frac{h}{2} k_{22}) \\
 k_{23} &= f_2(t_n + \frac{h}{2}, x_{1,n} + \frac{h}{2} k_{12}, x_{2,n} + \frac{h}{2} k_{22}) \\
 k_{14} &= f_1(t_n + h, x_{1,n} + h k_{13}, x_{2,n} + h k_{23}) \\
 k_{24} &= f_2(t_n + h, x_{1,n} + h k_{13}, x_{2,n} + h k_{23}) \\
 x_{1,n+1} &= x_{1,n} + \frac{h}{6}(k_{11} + 2k_{12} + 2k_{13} + k_{14}) \\
 x_{2,n+1} &= x_{2,n} + \frac{h}{6}(k_{21} + 2k_{22} + 2k_{23} + k_{24}). \tag{3.22}
 \end{aligned}$$

Ο προγραμματισμός της μεθόδου είναι απλός. Στο κυρίως πρόγραμμα έχουμε απλό interface με το χρήστη και του ζητάμε τα απαραίτητα δεδομένα: Το χρόνο ολοκλήρωσης από $t_0 = T0$ έως $t_f = TF$ και τον αριθμό των βημάτων $N = STEPS+1$. Τις αρχικές συνθήκες $x_1(t_0) = X10$, $x_2(t_0) = X20$. Η δομή των δεδομένων είναι απλή: Τρία arrays $T(P)$, $X1(P)$, $X2(P)$ αποθηκεύουν τις τιμές του χρόνου $t_1, t_2, \dots, t_{STEPS+1}$ και τις αντίστοιχες τιμές των συναρτήσεων $x_1(t_i)$ και $x_2(t_i)$. Το πρόγραμμα καλεί την υπορουτίνα $RK(T, X1, X2, T0, TF, X10, X20, STEPS, PSIZE)$ η οποία είναι ο οδηγός της μεθόδου, δηλ. οδηγεί την καρδιά του προγράμματος την υπορουτίνα $RKSTEP(t, x1, x2, dt)$ η οποία εφαρμόζει τους τύπους (3.22) και προωθεί τις τιμές των συναρτήσεων $x1$, $x2$ τη χρονική στιγμή t κατά ένα βήμα $h = dt$. Κάθε βήμα, αφού οριστούν οι αρχικές συνθήκες καταγράφεται στην RK στα arrays T , $X1$ και $X2$. Όταν η RK επιστρέψει τον έλεγχο στο κυρίως πρόγραμμα, τα αποτελέσματα είναι καταχωρημένα στα T , $X1$ και $X2$, τα οποία τυπώνονται στο αρχείο `rk.dat`. Παρακάτω παραθέτουμε το πρόγραμμα για να διευκολύνουμε τον αναγνώστη:

```

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Program to solve a 2 ODE system using Runge-Kutta Method
C      User must supply derivatives

```



```

real function f1(t,x1,x2)
implicit none
real t,x1,x2
f1=x2          !dx1/dt= v = x2
end

real function f2(t,x1,x2)
implicit none
real t,x1,x2
f2=-10.0D0*x1  !dx2/dt=dv/dt=a harmonic oscillator
end

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      RK(T,X1,X2,T0,TF,X10,X20,STEPS,PSIZE) is the driver
C      for the Runge-Kutta integration routine RKSTEP
C      Input: Initial and final times T0,T1
C              Initial values at t=T0  X10,X20
C              Number of steps of integration STEPS
C              Size of arrays T,X1,X2
C      Output: real arrays T(PSIZE),X1(PSIZE),X2(PSIZE) where
C      T(1) = T0 X1(1) = X10 X2(1) = X20
C              X1(i) = X1(at t=T(i)) X2(i) = X2(at t=T(i))
C      T(STEPS+1)=TF
C      Therefore we must have PSIZE>STEPS
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      subroutine RK(T,X1,X2,T0,TF,X10,X20,STEPS,PSIZE)
C      implicit none
C      integer STEPS,PSIZE
C      real T(PSIZE),X1(PSIZE),X2(PSIZE),T0,TF,X10,X20
C      real dt
C      real TS,X1S,X2S !values of time and X1,X2 at given step
C      integer i

C      Some checks:
C      if(STEPS .le. 1 )then
C          print *,'rk: STEPS must be >= 1'
C          stop
C      endif

```

[illegible]

```

implicit none
real t,x1,x2,dt
real f1,f2
real k11,k12,k13,k14,k21,k22,k23,k24
real h,h2,h6

h =dt                                !h =dt, integration step
h2=0.5D0*h                           !h2=h/2
h6=0.16666666666666666666D0*h       !h6=h/6

k11=f1(t,x1,x2)
k21=f2(t,x1,x2)
k12=f1(t+h2,x1+h2*k11,x2+h2*k21)
k22=f2(t+h2,x1+h2*k11,x2+h2*k21)
k13=f1(t+h2,x1+h2*k12,x2+h2*k22)
k23=f2(t+h2,x1+h2*k12,x2+h2*k22)
k14=f1(t+h ,x1+h *k13,x2+h *k23)
k24=f2(t+h ,x1+h *k13,x2+h *k23)

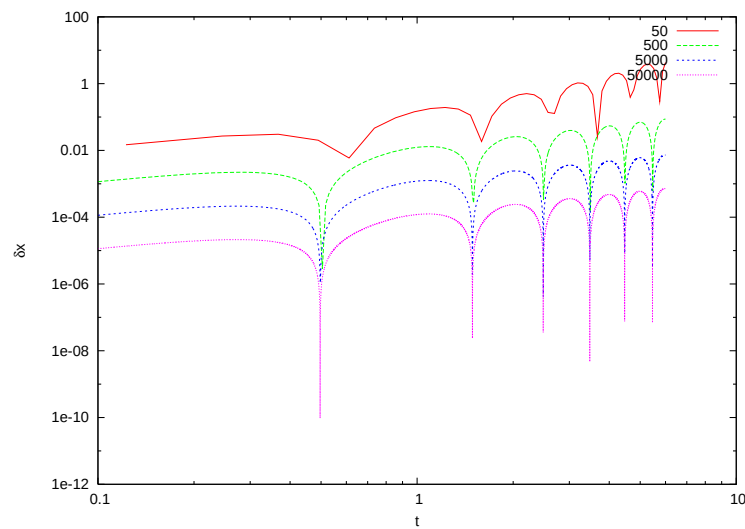
t =t+h
x1=x1+h6*(k11+2.0D0*(k12+k13)+k14)
x2=x2+h6*(k21+2.0D0*(k22+k23)+k24)

end

```

3.4 Σύγκριση των Μεθόδων

Στην παράγραφο αυτή θα κάνουμε έλεγχο ορθότητας των προγραμμάτων μας και θα μετρήσουμε την ακρίβειά τους ως προς τα ολικά σφάλματα διακριτοποίησης. Το πιο απλό τεστ στο οποίο μπορούμε να τα υποβάλλουμε είναι να συγκρίνουμε τα αριθμητικά αποτελέσματα σε ένα σύστημα για το οποίο έχουμε γνωστή την αναλυτική λύση. Θα διαλέξουμε να το κάνουμε για την περίπτωση του απλού αρμονικού ταλαντωτή. Οι αλλαγές που θα κάνουμε στα προγράμματα είναι μικρές και θα τις αναφέρουμε περιληπτικά. Κατ' αρχήν μετατρέπουμε όλες τις μεταβλητές **REAL** σε διπλής ακρίβειας **REAL*8**. Απλά αλλάζουμε τις δηλώσεις στα κατάλληλα σημεία του προγράμματος και προσθέτουμε ένα **D0** σε



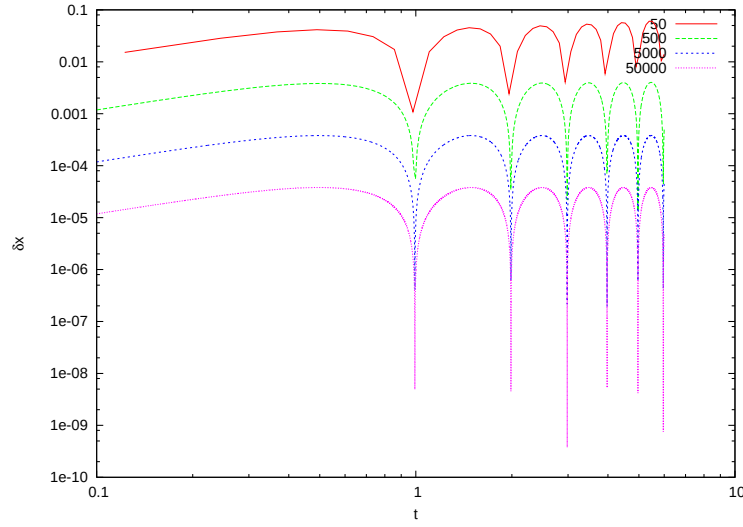
Σχήμα 3.11: Η απόκλιση της αριθμητικής λύσης με τη μέθοδο Euler από την αναλυτική για τον απλό αρμονικό ταλαντωτή. Οι παράμετροι είναι $\omega^2 = 10$, $t_0 = 0$, $t_f = 6$, $x(0) = 0.2$, $v(0) = 0$ και ο αριθμός των βημάτων είναι $N = 50, 500, 5,000, 50,000$. Παρατηρούμε ότι κάθε φορά το σφάλμα περίπου υποδεκαπλασιάζεται σύμφωνα με την αναμενόμενη ακρίβεια της μεθόδου $\sim \mathcal{O}(\Delta t)$.

όλες τις σταθερές (λ.χ. $0.5 \rightarrow 0.5D0$ κλπ). Στη συνέχεια μετατρέπουμε τις συναρτήσεις της επιτάχυνσης σε αυτή του αρμονικού ταλαντωτή $a = -\omega^2 x$ και παίρνουμε $\omega^2 = 10$ ($T \approx 1.987$). Έτσι στο πρόγραμμα `euler.f` έχουμε

```
real*8 function accel(x)
implicit none
real*8 x
accel = -10.0D0*x
end
```

ενώ στο πρόγραμμα `rk.f` έχουμε

```
real*8 function f2(t,x1,x2)
implicit none
real*8 t,x1,x2
f2=-10.0D0*x1
end
```



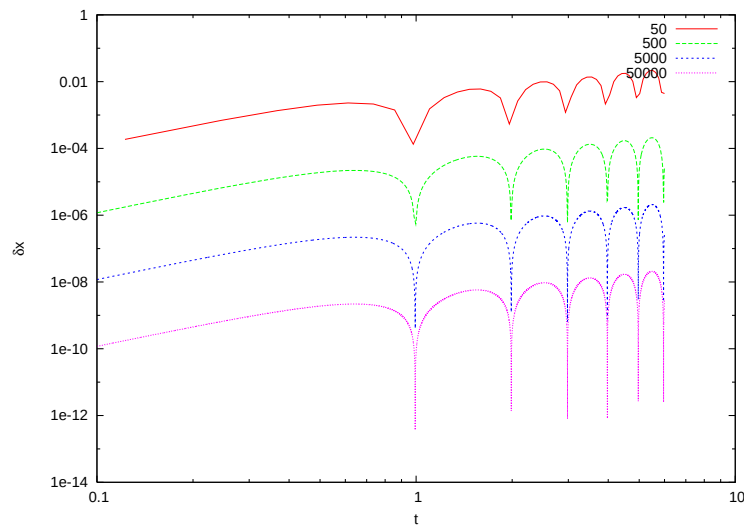
Σχήμα 3.12: Παρόμοια με το Σχήμα 3.11 για τη μέθοδο Euler-Cromer. Το σφάλμα περίπου υποδεκαπλασιάζεται σύμφωνα με την αναμενόμενη ακρίβεια της μεθόδου $\sim \mathcal{O}(\Delta t)$.

Στη συνέχεια τρέχουμε τα προγράμματα για δεδομένο χρονικό διάστημα από $t_0 = 0$ σε $t_f = 6$ με αρχικές συνθήκες $x_0 = 0.2$, $v_0 = 0$ και μεταβάλλουμε το χρονικό βήμα Δt αλλάζοντας τον αριθμό των βημάτων `steps`. Στη συνέχεια συγκρίνουμε τη λύση που παίρνουμε με αυτή του απλού αρμονικού ταλαντωτή

$$\begin{aligned} a(x) &= -\omega^2 x \\ x_h(t) &= x_0 \cos(\omega t) + (v_0/\omega) \sin(\omega t) \\ v_h(t) &= v_0 \cos(\omega t) - (x_0 \omega) \sin(\omega t), \end{aligned} \quad (3.23)$$

και μελετούμε τη σχέση των αποκλίσεων $\delta x(t) = |x(t) - x_h(t)|$ και $\delta v(t) = |v(t) - v_h(t)|$ με το βήμα Δt . Τα αποτελέσματά μας φαίνονται στα Σχήματα 3.11–3.14. Παρατηρούμε πως για τις μεθόδους Euler και Euler-Cromer τα σφάλματα είναι τάξης $\mathcal{O}(\Delta t)$ όπως είχαμε προβλέψει, αλλά για τη δεύτερη τα σφάλματα είναι αριθμητικά μικρότερα από αυτά της πρώτης έχοντας πιο ευσταθή συμπεριφορά για το συγκεκριμένο περιοδικό σύστημα. Για τη μέθοδο Euler-Verlet το σφάλμα βρίσκεται να είναι τάξης $\mathcal{O}(\Delta t^2)$ ενώ για την Runge-Kutta είναι τάξης $\mathcal{O}(\Delta t^4)$.

Μία άλλη μέθοδος για να ελένξουμε την ορθότητα των αποτελεσμάτων μας είναι να βρούμε μια διατηρούμενη ποσότητα όπως η ενέργεια, ορμή, στροφορμή κλπ και να εξετάζουμε αν αυτή αποκλίνει από την αρχική της τιμή. Στην



Σχήμα 3.13: Παρόμοια με το Σχήμα 3.11 για τη μέθοδο Euler-Verlet. Το σφάλμα περίπου υποεκατονταπλασιάζεται σύμφωνα με την αναμενόμενη ακρίβεια της μεθόδου $\sim \mathcal{O}(\Delta t^2)$.

περίπτωσή μας υπολογίζουμε τη μηχανική ενέργεια

$$E = \frac{1}{2}mv^2 + \frac{1}{2}m\omega^2 x^2 \quad (3.24)$$

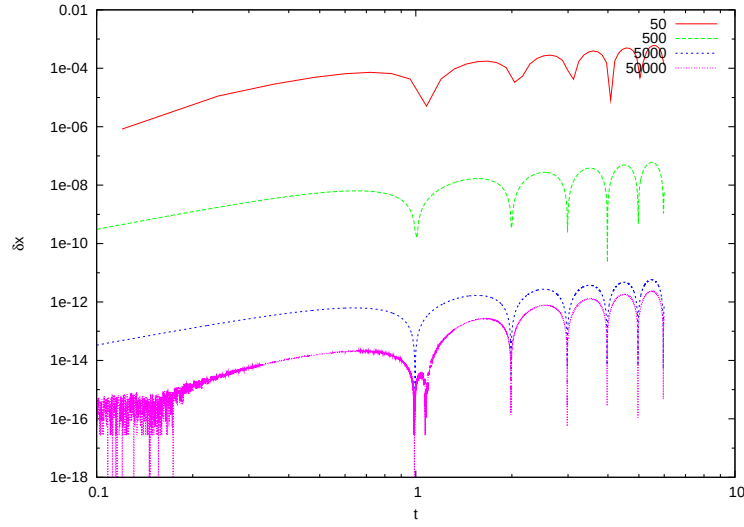
σε κάθε βήμα και από αυτή την απόκλιση $\delta E = |E - E_0|$. Τα αποτελέσματα φαίνονται στα σχήματα 3.15–3.18.

3.5 Ο Αρμονικός Ταλαντωτής με Απόσβεση και Εξωτερική Δύναμη.

Στην παράγραφο αυτή θα μελετήσουμε τον απλό αρμονικό ταλαντωτή του οποίου η κίνηση υπόκειται σε απόσβεση ανάλογη της ταχύτητάς του και σε εξωτερική δύναμη, την οποία για απλότητα θα πάρουμε να έχει ημιτονοειδή εξάρτηση από το χρόνο.

$$\frac{d^2x}{dt^2} + \gamma \frac{dx}{dt} + \omega_0^2 x = a_0 \sin \omega t, \quad (3.25)$$

με $F(t) = ma_0 \sin \omega t$ και ω η κυκλική συχνότητα της οδηγούσας δύναμης.



Σχήμα 3.14: Παρόμοια με το Σχήμα 3.11 για τη μέθοδο Runge-Kutta 4ης τάξης. Το σφάλμα περίπου μειώνεται κατά 10^{-4} σύμφωνα με την αναμενόμενη ακρίβεια της μεθόδου $\sim \mathcal{O}(\Delta t^4)$. Στα 50,000 βήματα γίνονται φανερά τα σφάλματα στρογγυλοποίησης.

Ας θεωρήσουμε αρχικά το σύστημα με $a_0 = 0$. Οι πραγματικές λύσεις της διαφορικής εξίσωσης⁶ που είναι πεπερασμένες για $t \rightarrow +\infty$ δίνονται, διακρίνοντας τις περιπτώσεις,

$$x_0(t) = c_1 e^{-(\gamma + \sqrt{\gamma^2 - 4\omega_0^2})t/2} + c_2 e^{-(\gamma - \sqrt{\gamma^2 - 4\omega_0^2})t/2}, \quad \gamma^2 - 4\omega_0^2 > 0, \quad (3.26)$$

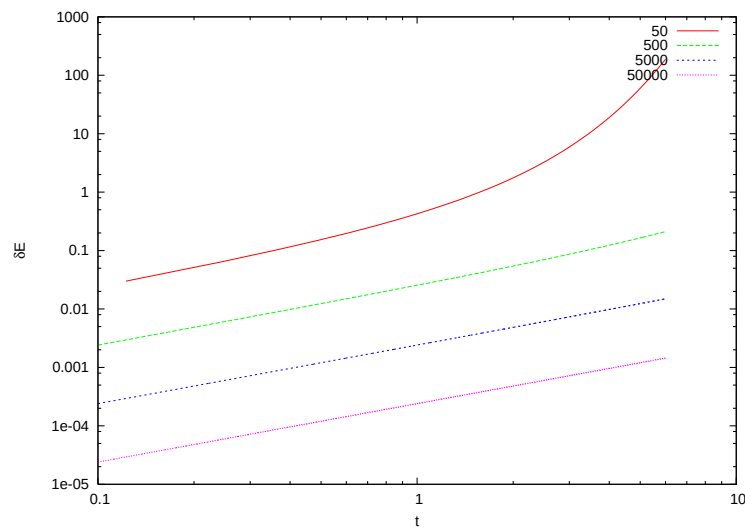
$$x_0(t) = c_1 e^{-\gamma t/2} + c_2 e^{-\gamma t/2} t, \quad \gamma^2 - 4\omega_0^2 = 0, \quad (3.27)$$

$$x_0(t) = c_1 e^{-\gamma t/2} \cos\left(\sqrt{-\gamma^2 + 4\omega_0^2} t/2\right) + c_2 e^{-\gamma t/2} \sin\left(\sqrt{-\gamma^2 + 4\omega_0^2} t/2\right), \quad \gamma^2 - 4\omega_0^2 < 0, \quad (3.28)$$

Στην τελευταία περίπτωση η λύση ταλαντώνεται με πλάτος που φθίνει εκθετικά με το χρόνο.

Για την περίπτωση που $a_0 > 0$, η γενική λύση προκύπτει από το άθροισμα μιας ειδικής λύσης $x_s(t)$ και της λύσης της ομογενούς εξίσωσης $x_0(t)$. Μια

⁶Προκύπτουν εύκολα αντικαθιστώντας τη δοκιμαστική λύση $x(t) = Ae^{-\Omega t}$ και λύνοντας ως προς Ω .



Σχήμα 3.15: Παρόμοια με το Σχήμα 3.11 στην περίπτωση της ενέργειας για τη μέθοδο Euler.

ειδική λύση προκύπτει από τη δοκιμαστική λύση $x_s(t) = A \sin \omega t + B \cos \omega t$ την οποία αντικαθιστούμε στην (3.25) και λύνουμε για τα A και B . Βρίσκουμε ότι

$$x_s(t) = \frac{a_0 [(\omega_0^2 - \omega^2) \cos \omega t + \gamma \omega \sin \omega t]}{(\omega_0^2 - \omega^2)^2 + \omega^2 \gamma^2} \quad (3.29)$$

και

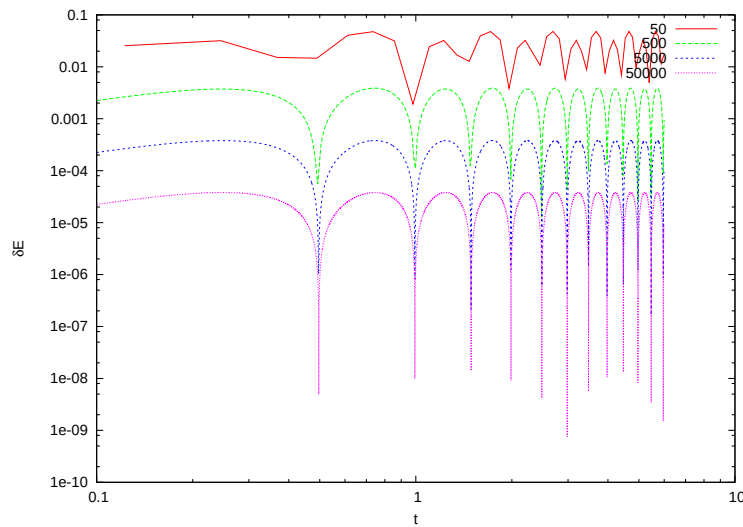
$$x(t) = x_0(t) + x_s(t). \quad (3.30)$$

Η λύση $x_0(t)$ είναι εκθετικά φθίνουσα με το χρόνο και τελικά επικρατεί η $x_s(t)$. Η μόνη περίπτωση που αυτό δεν ισχύει είναι στην περίπτωση του συντονισμού χωρίς απόσβεση $\omega = \omega_0$, $\gamma = 0$. Η λύση στην περίπτωση αυτή βρίσκεται εύκολα να είναι η

$$x(t) = c_1 \cos \omega t + c_2 \sin \omega t + \frac{a_0}{4\omega^2} (\cos \omega t + 2(\omega t) \sin \omega t). \quad (3.31)$$

Οι δύο πρώτοι όροι είναι αυτοί του απλού αρμονικού ταλαντωτή, ενώ ο τελευταίος όρος αυξάνει το πλάτος της μετατόπισης γραμμικά με το χρόνο καταδεικνύοντας τη συνεχή ροή ενέργειας από την εξωτερική δύναμη στον ταλαντωτή.

Ο προγραμματισμός του συστήματος γίνεται με απλή μετατροπή του κώδικα `rk.f`. Οι βασικές ρουτίνες `RK(T,X1,X2,T0,TF,X10,X20,STEPS,PSIZE)` και



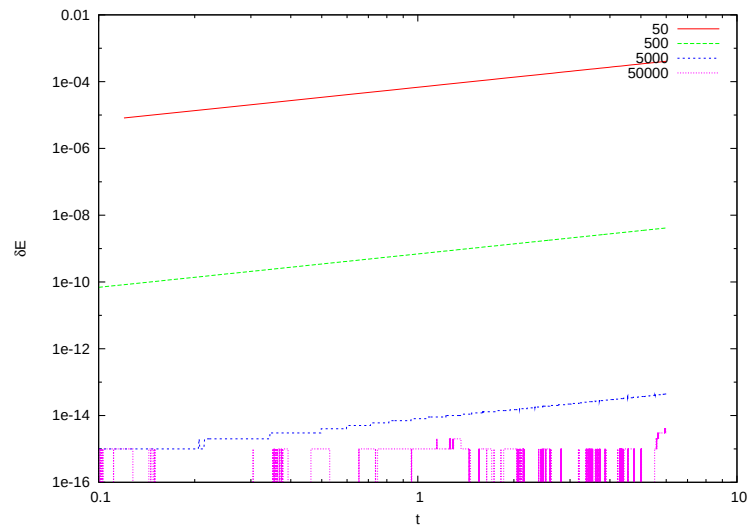
Σχήμα 3.16: Παρόμοια με το Σχήμα 3.11 στην περίπτωση της ενέργειας για τη μέθοδο Euler-Cromer.

`RKSTEP(t,x1,x2,dt)` μένουν ως έχουν και αλλάζει απλά το interface με το χρήστη. Εισάγουμε τις βασικές παραμέτρους ω_0 , ω , γ , a_0 διαδραστικά στην καθιερωμένη είσοδο (standard input). Επειδή θέλουμε να τις χρησιμοποιήσουμε στη συνάρτηση `f2(t,x1,x2)` που δίνει την επιτάχυνση χωρίς να αλλάξουμε τη δομή των οδηγών της μεθόδου Runge-Kutta, πρέπει να τις ορίσουμε σε κοινή θέση στη μνήμη για το κυρίως πρόγραμμα και την εν λόγω συνάρτηση. Αυτό στη `FORTRAN 77` γίνεται με τη χρήση `COMMON BLOCKS`. Αυτά ορίζονται μετά τη δήλωση των μεταβλητών και ορίζουν για αυτές μια συγκεκριμένη θέση στη μνήμη στις οποίες αποθηκεύονται οι τιμές τους. Το κομμάτι κώδικα

```
real*8      omega_0,omega,gamma,a_0,omega_02,omega2
common /params/omega_0,omega,gamma,a_0,omega_02,omega2
```

σε οποιαδήποτε ρουτίνα δίνει πρόσβαση στη μνήμη στη 'θέση' `params` στην οποία αποθηκεύονται οι τιμές των μεταβλητών⁷. Το μόνο άλλο σημείο που χρήζει προσοχής στο πρόγραμμα είναι η συνάρτηση της επιτάχυνσης `f2(t,x1,x2)` η οποία τώρα έχει όρο που εξαρτάται από την ταχύτητα που εδώ συμβολίζουμε με τη μεταβλητή `x2`:

⁷Στην πραγματικότητα ο προγραμματιστής μπορεί να αποθηκεύσει ό,τι τύπου μεταβλητή θέλει σε διαφορετικές υπορουτίνες και έτσι να ρυθμίσει την παροχή μνήμης με 'οικονομία'. Εμείς μόνο για ευκολία χρησιμοποιούμε τις ίδιου τύπου μεταβλητές με τα ίδια ονόματα.



Σχήμα 3.18: Παρόμοια με το Σχήμα 3.11 στην περίπτωση της ενέργειας για τη μέθοδο Runge-Kutta 4ης τάξης. Για μεγάλο αριθμό βημάτων το σφάλμα είναι σφάλμα στρογγυλοποίησης.

```
implicit none
integer P
parameter(P=110000)
real*8 T0,TF,X10,X20
integer STEPS,PSIZE
real*8 T(P),X1(P),X2(P)
real*8 Energy
real*8      omega_0,omega,gamma,a_0,omega_02,omega2
common /params/omega_0,omega,gamma,a_0,omega_02,omega2
integer i
```

```
C      Input:
print *, 'Runge-Kutta Method for DLO Integration'
print *, 'Enter omega_0, omega, gamma, a_0:'
read(5,*) omega_0,omega,gamma,a_0
omega_02 = omega_0*omega_0
omega2    = omega  *omega
print *, 'omega_0= ',omega_0,' omega= ', omega
```


3.5. Ο ΑΡΜΟΝΙΚΟΣ ΤΑΛΑΝΤΩΤΗΣ ΜΕ ΑΠΟΣΒΕΣΗ ΚΑΙ ΕΞΩΤΕΡΙΚΗ ΔΥΝΑΜΗ.181

```

print *, 'gamma= ',gamma, ' a_0= ',a_0
print *,'Enter STEPS,T0,TF,X10,X20:'
read(5,*) STEPS,T0,TF,X10,X20
print *,'No. Steps= ',STEPS
print *,'Time: Initial T0 =',T0,' Final TF=',TF
print *,'          X1(T0)=' ,X10,' X2(T0)=' ,X20

```

C The Calculation:

```

PSIZE=P
call RK(T,X1,X2,T0,TF,X10,X20,STEPS,PSIZE)

```

C Output:

```

open(unit=11,file='dlo.dat')
write(11,*)'# Damped Linear Oscillator - dlo'
write(11,*)'# omega_0= ',omega_0,' omega= ', omega,
*      ' gamma= ',gamma,' a_0= ',a_0
do i=1,STEPS+1
  Energy = 0.5D0*X2(i)*X2(i)+0.5D0*omega_02*X1(i)*X1(i)
  write(11,*)T(i),X1(i),X2(i),Energy
enddo
close(11)

```

end

C CCC

C The functions f1,f2(t,x1,x2) provided by the user

C CCC

```

real*8 function f1(t,x1,x2) !velocity function
implicit none
real*8 t,x1,x2
f1=x2          !dx1/dt= v = x2
end

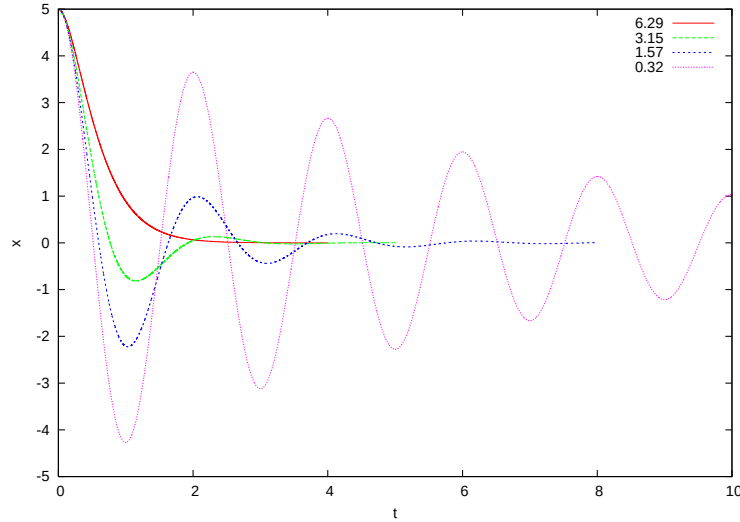
```

```

real*8 function f2(t,x1,x2) !acceleration function
implicit none
real*8          omega_0,omega,gamma,a_0,omega_02,omega2
common /params/omega_0,omega,gamma,a_0,omega_02,omega2
real*8 t,x1,x2

```

```
f2=-omega_02*x1-gamma*x2+a_0*dcos(omega*t)
end
```

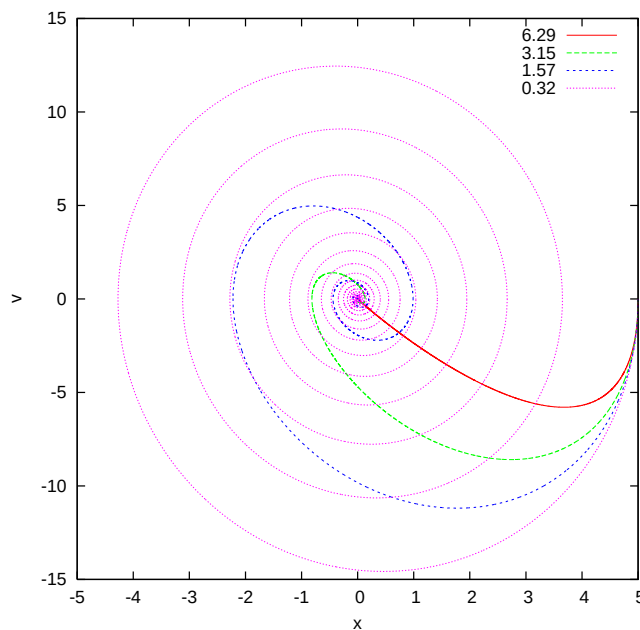


Σχήμα 3.19: Η θέση συναρτήσει του χρόνου για τον αρμονικό ταλαντωτή με απόσβεση για διαφορετικές τιμές του συντελεστή απόσβεσης γ με $\omega_0 = 3.145$.

Τα αποτελέσματα φαίνονται στα Σχήματα 3.19–3.22. Στο Σχήμα 3.19 παρατηρούμε τη μετάβαση από την φάση που η κίνηση αποσβέννυται χωρίς ταλάντωση για $\gamma > 2\omega_0$ στη φάση που το σύστημα ταλαντώνεται με εκθετικά φθίνων με το χρόνο πλάτος για $\gamma < 2\omega_0$. Η εκθετική μείωση του πλάτους φαίνεται στο Σχήμα 3.21, ενώ η εξάρτηση της περιόδου T του όρου ταλάντωσης (δηλ. του ορίσματος στον (συν)ημιτονοειδή όρο) από το συντελεστή απόσβεσης γ στο Σχήμα 3.22. Το εν λόγω σχήμα προκύπτει από τη Σχέση (3.28) την οποία γράφουμε στη μορφή

$$4\omega_0^2 - \left(\frac{2\pi}{T}\right)^2 = \gamma^2. \quad (3.32)$$

Το δεξί μέλος της εξίσωσης τοποθετείται στον οριζόντιο άξονα ενώ στον κάθετο τοποθετούμε το αριστερό. Η παραπάνω σχέση προβλέπει ότι οι δύο ποσότητες είναι ίσες και οι μετρήσεις πρέπει να βρίσκονται πάνω στη διαγώνιο $y = x$. Οι μετρήσεις για την ‘περίοδο’ T παίρνονται μετρώντας το χρόνο μεταξύ δύο διαδοχικών ακρότατων ($v = 0$) στην τροχιά $x(t)$ (βλ. Σχήμα 3.19).

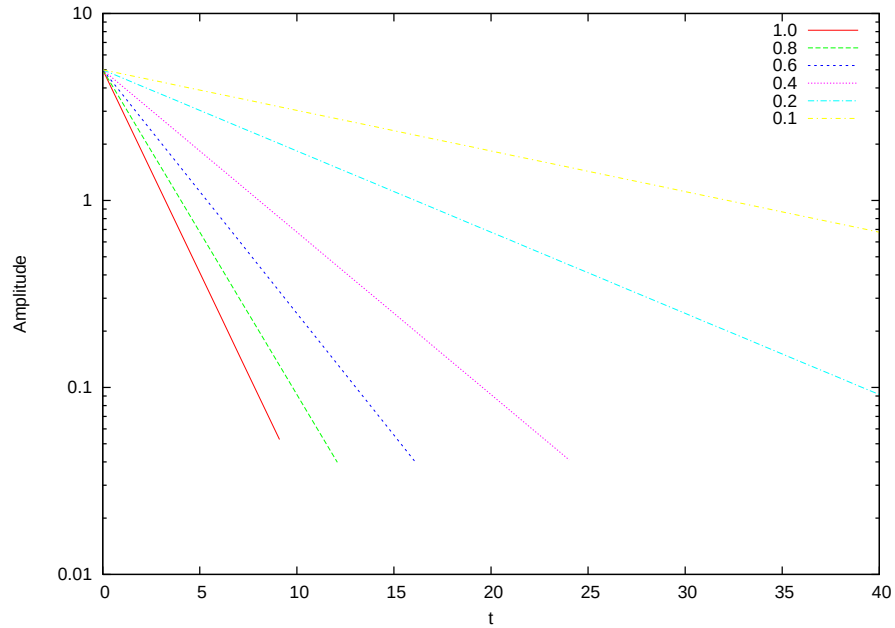


Σχήμα 3.20: Η τροχιά στο χώρο των φάσεων για τον αρμονικό ταλαντωτή με απόσβεση για διαφορετικές τιμές του συντελεστή απόσβεσης γ με $\omega_0 = 3.145$. Παρατηρούμε την ύπαρξη ελκυστή στο $(x, v) = (0, 0)$ στον οποίο καταλήγει στο σύστημα για $t \rightarrow +\infty$.

Τέλος σημαντικό είναι να μελετήσουμε το Σχήμα 3.20 στο οποίο βλέπουμε την τροχιά του συστήματος στο χώρο των φάσεων⁸. Το σύστημα για $t \rightarrow +\infty$ καταλήγει στο σημείο $(0, 0)$ για οποιαδήποτε τιμή της σταθεράς γ . Το σημείο αυτό για το σύστημα είναι ένας ‘ελκυστής’ (attractor).

Στη συνέχεια προσθέτουμε την εξωτερική δύναμη και μελετούμε την απόκριση του συστήματος σε αυτή. Το πρώτο που παρατηρούμε στο Σχήμα 3.23 είναι ότι το σύστημα μετά από μια παροδική κατάσταση (transient state) η οποία εξαρτάται από τις αρχικές συνθήκες, καταλήγει σε μία σταθερή κατάσταση η οποία δεν εξαρτάται από τις αρχικές συνθήκες. Στη συγκεκριμένη περίπτωση αυτό προβλέπεται εύκολα από τις Σχέσεις (3.26) – (3.28) όπου αφού οι εκθετικοί όροι γίνουν αμελητέοι, επικρατεί ο όρος $x_s(t)$ της (3.29). Ο τελευταίος

⁸Για την ακρίβεια ο χώρος των φάσεων είναι ο χώρος των θέσεων-ορμών, αλλά στην περίπτωσή μας η διαφορά είναι τετριμμένη.

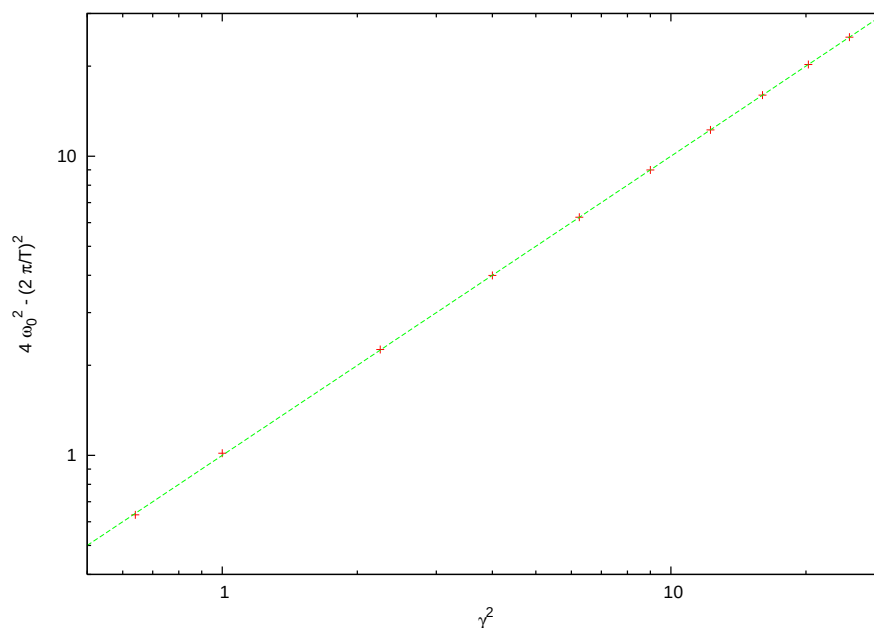


Σχήμα 3.21: Το πλάτος ταλάντωσης για τον αρμονικό ταλαντωτή με απόσβεση για διαφορετικές τιμές του συντελεστή απόσβεσης γ με $\omega_0 = 3.145$. Παρατηρούμε την εκθετική μείωση του πλάτους με το χρόνο.

μπορεί να γραφτεί στη μορφή

$$\begin{aligned} x(t) &= x_0(\omega) \cos(\omega t + \delta(\omega)) \\ x_0(\omega) &= \frac{a_0}{\sqrt{(\omega_0^2 - \omega^2)^2 + \gamma^2 \omega^2}}, \quad \tan \delta(\omega) = \frac{\omega \gamma}{\omega^2 - \omega_0^2}. \end{aligned} \quad (3.33)$$

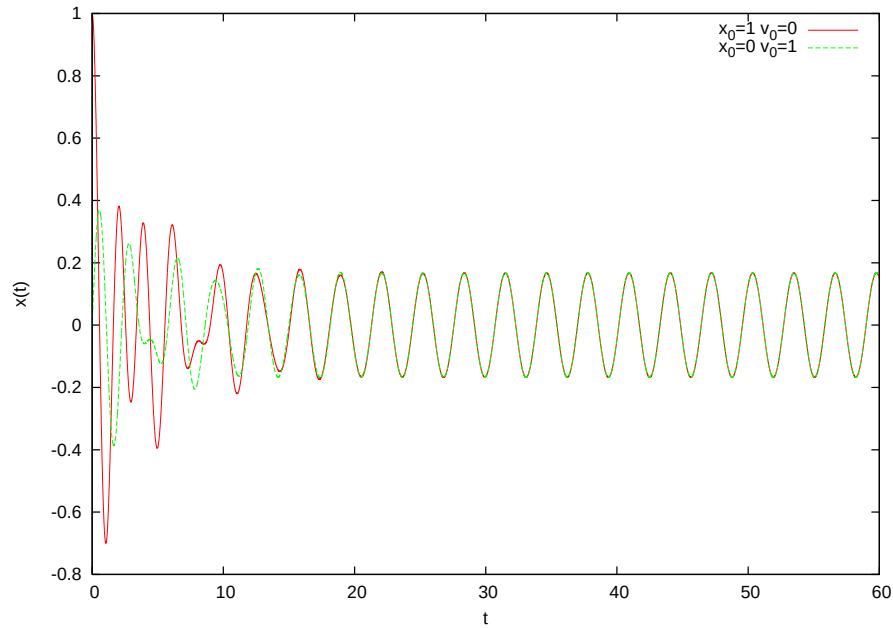
Την παραπάνω σχέση την επιβεβαιώνουμε στο Σχήμα 3.24 όπου μελετάμε την εξάρτηση του πλάτους $x_0(\omega)$ από τη γωνιακή συχνότητα της εξωτερικής δύναμης. Τέλος μελετάμε την τροχιά του συστήματος στο χώρο των φάσεων. Παρατηρούμε την ύπαρξη ελκυστή για το σύστημα, δηλ. την ύπαρξη υπόχωρου πάνω στον οποίο κινείται το σύστημα όταν $t \rightarrow +\infty$ ο οποίος είναι ανεξάρτητος των αρχικών συνθηκών. Στην περίπτωση αυτή ο ελκυστής είναι καμπύλη μιας διάστασης σε αντίθεση με την περίπτωση που δεν έχουμε εξωτερική δύναμη και ο ελκυστής είναι μηδενοδιάστατο σημείο (βλ. Σχήμα 3.20).



Σχήμα 3.22: Η περίοδος ταλάντωσης για τον αρμονικό ταλαντωτή με απόσβεση για διαφορετικές τιμές του συντελεστή απόσβεσης γ με $\omega_0 = 3.145$. Στους άξονες επιλέγουμε τις κατάλληλες ποσότητες για τη γραφική επιβεβαίωση της Εξίσωσης (3.28), δηλ. $(2\pi/T)^2 = 4\omega_0^2 - \gamma^2$. Τα σημεία είναι οι μετρήσεις ενώ η ευθεία γραμμή είναι η θεωρητική πρόβλεψη, δηλ. η διαγώνιος $y = x$

3.6 Το Εκκρεμές με Απόσβεση και Εξωτερική Δύναμη.

Στην παράγραφο αυτή θα μελετήσουμε ένα μη γραμμικό δυναμικό σύστημα με μη-τετριμμένες ιδιότητες. Θα ανακαλύψουμε πως πολλά ενδιαφέροντα δυναμικά συστήματα στη φύση, παρ' όλο που η κίνησή τους καθορίζεται από ντετερμινιστικούς νόμους και δεν υπάρχει πρόβλημα αρχής να προσδιοριστεί η κατάστασή τους σε αυθαίρετους χρόνους δεδομένων των αρχικών συνθηκών, η συμπεριφορά τους είναι χαοτική και πολύ σύντομα είναι πρακτικά αδύνατο να κάνουμε προβλέψεις σε βάθος χρόνου. Αυτό δίνεται από το εκκρεμές στο ομογενές πεδίο βαρύτητας της γης με δύναμη απόσβεσης ανάλογη της ταχύτητας του εκκρεμούς και μια εξωτερική οδηγούσα δύναμη, κατακόρυφη στη διεύθυνση και



Σχήμα 3.23: Η περίοδος ταλάντωσης για τον αρμονικό ταλαντωτή με απόσβεση και εξωτερική δύναμη για διαφορετικές αρχικές συνθήκες. Έχουμε πάρει $\omega_0 = 3.145$, $\omega = 2.0$, $\gamma = 0.5$ και $a_0 = 1.0$. Παρατηρούμε ότι το σύστημα παρουσιάζει μια μεταβατική κατάσταση μετά το πέρας της οποίας ταλαντώνεται σύμφωνα με τη σχέση $x(t) = x_0(\omega) \cos(\omega t + \delta)$.

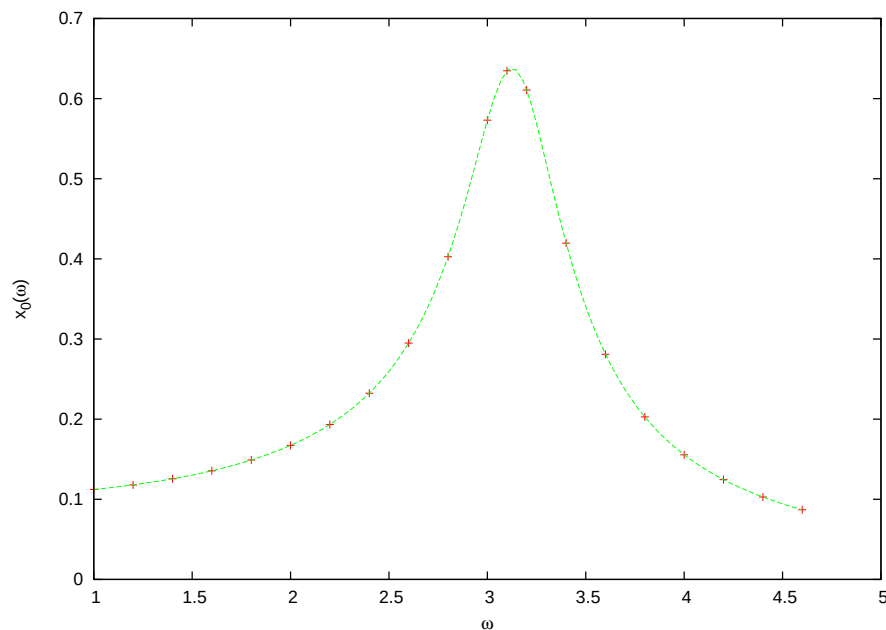
μέτρο/φορά που μεταβάλλεται συνημιτονοειδώς με το χρόνο:

$$\frac{d^2\theta}{dt^2} + \gamma \frac{d\theta}{dt} + \omega_0^2 \sin \theta = -2A \cos \omega t \sin \theta. \quad (3.34)$$

Στην παραπάνω εξίσωση θ είναι η γωνία με την κατακόρυφο, γ ο συντελεστή απόσβεσης, $\omega_0^2 = g/L$ η φυσική κυκλική συχνότητα του εκκρεμούς και ω , $2A$ η κυκλική συχνότητα και το πλάτος της εξωτερικής γωνιακής επιτάχυνσης που προκαλείται από την εξωτερική δύναμη.

Όταν δεν υπάρχει εξωτερική δύναμη το σύστημα έχει εξ' αιτίας των αποσβέσεων, ελκυστή το σημείο $(\theta, \dot{\theta}) = (0, 0)$. Αυτό θα συνεχίσει να συμβαίνει καθώς αυξάνουμε το A από μηδέν και ο ελκυστής παραμένει σταθερός για αρκετά μικρό A . Για κάποια τιμή A_c ο ελκυστής γίνεται ασταθής και η συμπεριφορά του συστήματος γίνεται πολυπλοκότερη. Αυτή θα μελετηθεί λεπτομερέστερα σε επόμενο κεφάλαιο, εδώ θα κάνουμε μια εισαγωγική προσέγγιση έχοντας κατά νου ότι το σύστημα αυτό παρουσιάζει ενδιαφέρον.

3.6. ΤΟ ΕΚΚΡΕΜΕΣ ΜΕ ΑΠΟΣΒΕΣΗ ΚΑΙ ΕΞΩΤΕΡΙΚΗ ΔΥΝΑΜΗ.187



Σχήμα 3.24: Το πλάτος ταλάντωσης $x_0(\omega)$ για τον αρμονικό ταλαντωτή με απόσβεση και εξωτερική δύναμη. Έχουμε πάρει $\omega_0 = 3.145$, $\gamma = 0.5$ και $a_0 = 1.0$. Παρατηρούμε συντονισμό για $\omega \approx \omega_0$. Τα σημεία είναι οι μετρήσεις μας και η συνεχής γραμμή η θεωρητική πρόβλεψη (3.33).

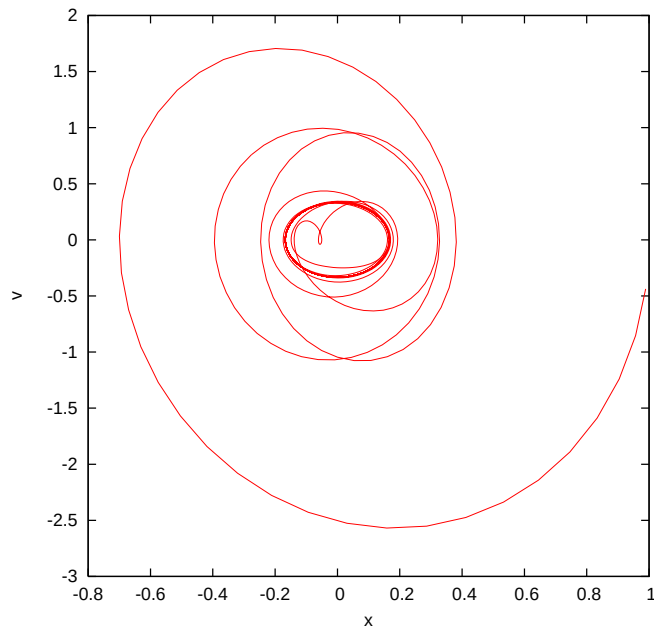
Ο προγραμματισμός του συστήματος γίνεται με τετριμμένες αλλαγές του προγράμματος `dlo.f`. Οι μετατροπές στο πρόγραμμα φαίνονται παρακάτω, έχοντας κατά νου πως $X1 \leftrightarrow \theta$, $X2 \leftrightarrow \dot{\theta}$, $a_0 \leftrightarrow A$. Το πρόγραμμα το αποθηκεύουμε στο αρχείο `fdp.f` (`fdp`= Forced Damped Pendulum). Οι εντολές ανάμεσα στις τελίτσες είναι πανομοιότυπες με τα προγράμματα `dlo.f`, `rk.f`.

```

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Program to solve Forced Damped Pendulum
C      using 4th order Runge-Kutta Method
C      Output is written in file fdp.dat
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

program fdp_solve
implicit none
integer P

```



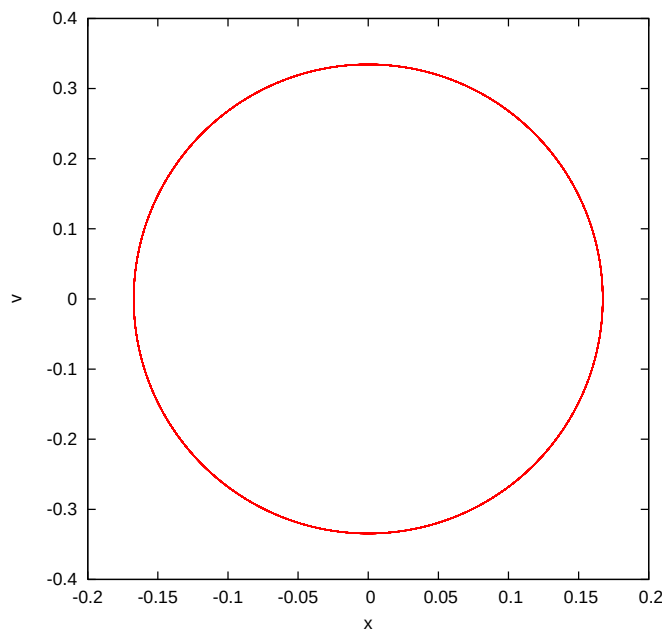
Σχήμα 3.25: Η τροχιά στο χώρο των φάσεων για τον αρμονικό ταλαντωτή με απόσβεση και εξωτερική δύναμη. Έχουμε πάρει $\omega_0 = 3.145$, $\omega = 2.0$, $\gamma = 0.5$ και $a_0 = 1.0$.

```

parameter(P=110000)
...
    Energy = 0.5D0*X2(i)*X2(i)+omega_02*(1.0D0-dcos(X1(i)))
...
.....
    real*8 function f2(t,x1,x2)
    implicit none
    real*8          omega_0,omega,gamma,a_0,omega_02,omega2
    common /params/omega_0,omega,gamma,a_0,omega_02,omega2
    real*8 t,x1,x2
    f2=-(omega_02+2.0D0*a_0*dcos(omega*t))*dsin(x1)-gamma*x2
    end
.....
    subroutine RKSTEP(t,x1,x2,dt)
...
    real*8 h,h2,h6,pi,pi2

```


3.6. ΤΟ ΕΚΚΡΕΜΕΣ ΜΕ ΑΠΟΣΒΕΣΗ ΚΑΙ ΕΞΩΤΕΡΙΚΗ ΔΥΝΑΜΗ.189



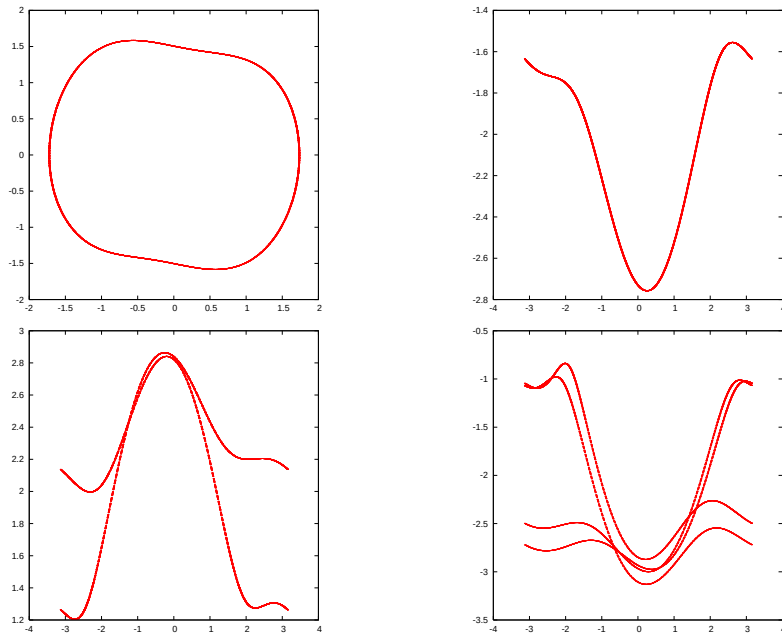
Σχήμα 3.26: Η τροχιά στο χώρο των φάσεων για τον αρμονικό ταλαντωτή με απόσβεση και εξωτερική δύναμη για $t > 100$. Έχουμε πάρει $\omega_0 = 3.145$, $\omega = 2.0$, $\gamma = 0.5$ και $a_0 = 1.0$. Παρατηρούμε τον ελκυστή που τώρα είναι καμπύλη (έλλειψη) μιας διάστασης.

```
parameter(pi =3.14159265358979324D0)
parameter(pi2=6.28318530717958648D0)
...
x1=x1+h6*(k11+2.0D0*(k12+k13)+k14)
x2=x2+h6*(k21+2.0D0*(k22+k23)+k24)
if( x1 .gt.  pi) x1 = x1 - pi2
if( x1 .lt. -pi) x1 = x1 + pi2
end
```

Τις τελευταίες γραμμές στο πρόγραμμα τις προσθέσαμε ώστε να κρατήσουμε τη γωνία στο διάστημα $[-\pi, \pi]$.

Για να μελετήσουμε τις ιδιότητες του συστήματος θα θέσουμε $\omega_0 = 1$, $\omega = 2$, και $\gamma = 0.2$ εκτός αν αναφέρουμε ρητώς διαφορετικά. Η φυσική περίοδος του εκκρεμούς είναι $T_0 = 2\pi/\omega_0 = 2\pi \approx 6.28318530717958648$ ενώ αυτή της εξωτερικής δύναμης $T = 2\pi/\omega = \pi \approx 3.14159265358979324$. Το σύστημα για $A < A_c$ με $A_c \approx 0.18$ έχει σταθερό ελκυστή το σημείο $(\theta, \dot{\theta}) = (0, 0)$

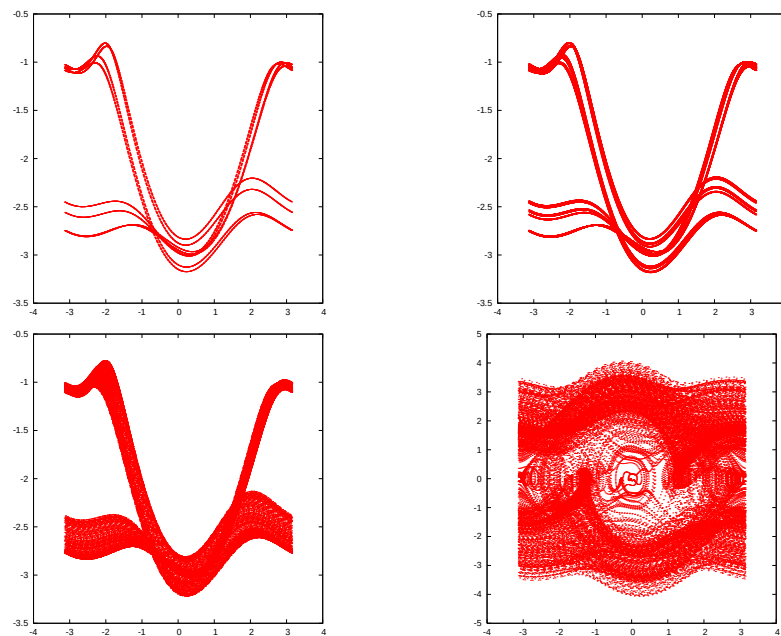
ενώ για $A_c < A < 0.71$ ο ελκυστής είναι κλειστή καμπύλη. Η περίοδος της ταλάντωσης βρίσκεται να είναι διπλάσια αυτής της εξωτερικής δύναμης. Για $0.72 < A < 0.79$ ο ελκυστής είναι ανοιχτή καμπύλη μια και το εκκρεμές εκτελεί ολόκληρους κύκλους στη σταθερή του κατάσταση. Η περίοδος στο διάστημα αυτό γίνεται ίση με αυτή της εξωτερικής δύναμης. Για $0.79 < A \lesssim 1.033$ η περίοδος διπλασιάζεται διαδοχικά για κρίσιμες τιμές του A , η τροχιά όμως συνεχίζει να είναι περιοδική. Για μεγαλύτερες τιμές του A η τροχιά παύει να είναι περιοδική και το σύστημα έχει χαοτική συμπεριφορά. Για $A \approx 3.1$ βρίσκουμε το σύστημα να έχει πάλι περιοδική κίνηση ενώ για $A \approx 3.8 - 4.448$ να έχουμε το φαινόμενο διπλασιασμού της περιόδου. Για $A \approx 4.4489$ έχουμε καθαρή χαοτική συμπεριφορά κ.ο.κ. Τα αποτελέσματα αυτά περιγράφονται στα Σχήματα 3.27–3.29.



Σχήμα 3.27: Τροχιά στο χώρο των φάσεων για το εκκρεμές με απόσβεση και εξωτερική δύναμη. Έχουμε πάρει $\omega_0 = 1.0$, $\omega = 2.0$, $\gamma = 0.2$ και $A = 0.60, 0.72, 0.85, 1.02$. Παρατηρούμε το φαινόμενο του διπλασιασμού της περιόδου.

Για την ανάλυση της συμπεριφοράς του εκκρεμούς και κυρίως για τη διεύκλυνση της διάκρισης μεταξύ περιοδικής και χαοτικής συμπεριφοράς μπορεί κανείς να μελετήσει τα λεγόμενα διαγράμματα Poinaré. Στα διαγράμματα αυτά

3.6. ΤΟ ΕΚΚΡΕΜΕΣ ΜΕ ΑΠΟΣΒΕΣΗ ΚΑΙ ΕΞΩΤΕΡΙΚΗ ΔΥΝΑΜΗ.191

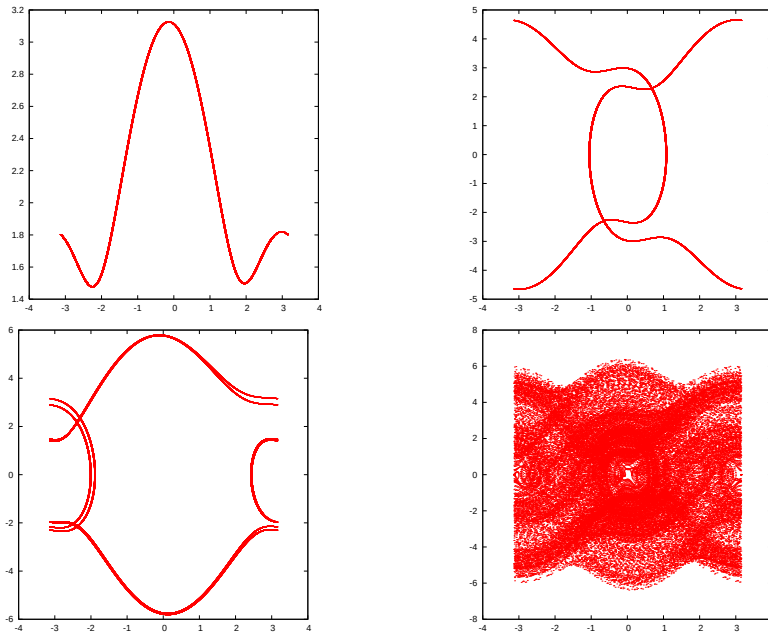


Σχήμα 3.28: Τροχιά στο χώρο των φάσεων για το εκκρεμές με απόσβεση και εξωτερική δύναμη. Έχουμε πάρει $\omega_0 = 1.0$, $\omega = 2.0$, $\gamma = 0.2$ και $A = 1.031, 1.033, 1.04, 1.4$. Παρατηρούμε την εμφάνιση χαοτικής συμπεριφοράς.

τοποθετούμε ένα σημείο στο χώρο των φάσεων κάθε φορά που ο χρόνος είναι ακέραιο πολλαπλάσιο της περιόδου της εξωτερικής δύναμης. Με τον τρόπο αυτό αν η κίνηση είναι περιοδική με περίοδο ίση με την περίοδο της εξωτερικής δύναμης θα έχουμε ένα σημείο στο διάγραμμα και γενικότερα θα έχουμε n σημεία αν η περίοδος είναι n -πολλαπλάσιο της $T = 2\pi/\omega$. Οπότε κανείς περιμένει όταν παρατηρήται το φαινόμενο διπλασιασμού της περιόδου, το διάγραμμα Poincaré να αποκτά επί πλέον μεμονωμένα σημεία ενώ όταν η συμπεριφορά είναι χαοτική, τα σημεία να ανήκουν σε έναν υπόχωρο του χώρου των φάσεων που να έχει πολυπλοκότερη δομή. Αυτό μπορούμε εύκολα να το προγραμματίσουμε στον κώδικά μας στο `fdo.f` ή εναλλακτικά να πάρουμε τη σχετική πληροφορία από το αρχείο εξόδου `fdo.dat` με το πρόγραμμα `awk` το οποίο τρέχουμε από τη γραμμή εντολών⁹:

```
awk -v o=$omega -v s=$STEPS -v tf=$TF \
  'BEGIN{T=6.283185307179/o;dt=tf/s;} $1%T<dt{print $2,$3}' fdp.dat
```

⁹Η εντολή μπορεί να γραφτεί σε μία γραμμή χωρίς το τελικό `\` της πρώτης γραμμής



Σχήμα 3.29: Τροχιά στο χώρο των φάσεων για το εκκρεμές με απόσβεση και εξωτερική δύναμη. Έχουμε πάρει $\omega_0 = 1.0$, $\omega = 2.0$, $\gamma = 0.2$ και $A = 1.568, 3.8, 4.44, 4.5$. Παρατηρούμε την παύση και επανεμφάνιση χαοτικής συμπεριφοράς.

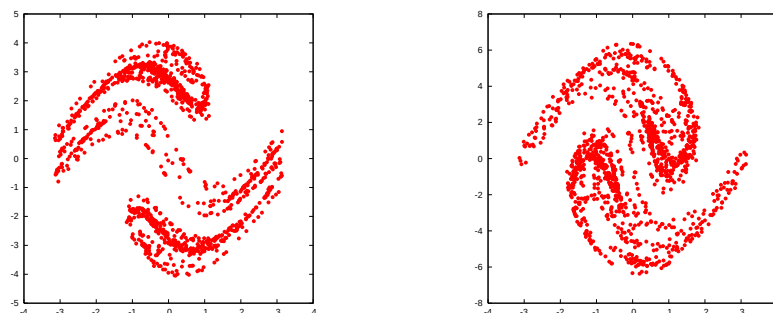
όπου ω , ω_0 , γ οι τιμές της κυκλικής συχνότητας ω , αριθμού βημάτων ολοκλήρωσης και τελικού χρόνου t_f ¹⁰. Στο πρόγραμμα υπολογίζουμε την περίοδο T και το βήμα χρόνου dt . Στη συνέχεια τυπώνουμε εκείνες τις γραμμές του αρχείου των οποίων ο χρόνος είναι ακέραιο πολλαπλάσιο της περιόδου με ακρίβεια χρόνου dt ¹¹. Αυτό γίνεται με την πράξη modulo $1 \% T < dt$ που είναι TRUE όταν το υπόλοιπο της διαίρεσης της πρώτης στήλης (1) του αρχείου `fdo.dat` έχει υπόλοιπο διαίρεσης με την περίοδο T μικρότερο από dt . Τα αποτελέσματα για τη χαοτική φάση φαίνονται στο Σχήμα 3.30.

Κλείνουμε τη μελέτη μας με την παρουσίαση μιας ακόμα έννοιας που μας βοηθάει στην ανάλυση των ιδιοτήτων του εκκρεμούς. Αυτή είναι η έννοια της 'λεκάνης του ελκυστή' (basin of an attractor) η οποία είναι το σύνολο των

¹⁰ Αντικαταστήστε με απλές αριθμητικές τιμές αν δεν ξέρετε να χρησιμοποιήτε μεταβλητές φλοιού.

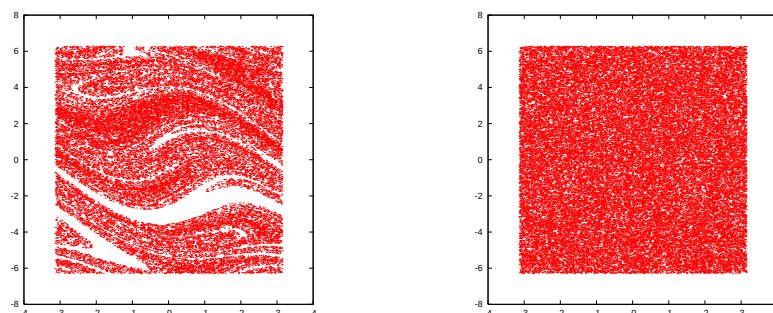
¹¹ Φυσικά αυτό κάνει τα μεμονομένα σημεία να γίνονται συγκεχυμένα στο διάγραμμα Poincaré.

3.6. ΤΟ ΕΚΚΡΕΜΕΣ ΜΕ ΑΠΟΣΒΕΣΗ ΚΑΙ ΕΞΩΤΕΡΙΚΗ ΔΥΝΑΜΗ.193



Σχήμα 3.30: Διάγραμμα Poincaré για το εκκρεμές με απόσβεση και εξωτερική δύναμη όταν παρουσιάζει χαοτική συμπεριφορά. Έχουμε πάρει $\omega_0 = 1.0$, $\omega = 2.0$, $\gamma = 0.2$ και $A = 1.4, 4.5$.

αρχικών συνθηκών στο χώρο των φάσεων που οδηγούν το σύστημα στο συγκεκριμένο ελκυστή. Στην περίπτωση μας το εκκρεμές για $A > 0.79$ εκτελεί κυκλική κίνηση είτε με θετική είτε με αρνητική φορά (μετα την παρέλευση της παροδικής φάσης φυσικά) η οποίες αποτελούν τους δύο ελκυστές του συστήματος. Παίρνοντας ένα μεγάλο δείγμα από αρχικές συνθήκες και σημειώνοντας το πρόσημο της γωνιακής ταχύτητας μετά την παρέλευση της παροδικής φάσης παίρνουμε το Σχήμα 3.31. Στην περιοδική φάση διακρίνουμε περιοχές, των οποίων τα περιγράμματα δεν είναι καθαρά, οι οποίες δεν οδηγούν στο συγκεκριμένο ελκυστή.



Σχήμα 3.31: Basin of attractor για το εκκρεμές με απόσβεση και εξωτερική δύναμη. Έχουμε πάρει $\omega_0 = 1.0$, $\omega = 2.0$, $\gamma = 0.2$ και $A = 0.85, 1.4$. Διακρίνουμε την περιοδική από τη χαοτική συμπεριφορά.

3.7 Παράρτημα: Στη Μέθοδο Euler–Verlet

Η Σχέσεις (3.11) προκύπτουν από το ανάπτυγμα κατά Taylor

$$\begin{aligned}\theta(t + \Delta t) &= \theta(t) + (\Delta t)\theta'(t) + \frac{(\Delta t)^2}{2!}\theta''(t) + \frac{(\Delta t)^3}{3!}\theta'''(t) + \mathcal{O}((\Delta t)^4) \\ \theta(t - \Delta t) &= \theta(t) - (\Delta t)\theta'(t) + \frac{(\Delta t)^2}{2!}\theta''(t) - \frac{(\Delta t)^3}{3!}\theta'''(t) + \mathcal{O}((\Delta t)^4).\end{aligned}$$

Προσθέτοντας και αφαιρώντας κατά μέλη παίρνουμε

$$\begin{aligned}\theta(t + \Delta t) + \theta(t - \Delta t) &= 2\theta(t) + (\Delta t)^2\theta''(t) + \mathcal{O}((\Delta t)^4) \\ \theta(t + \Delta t) - \theta(t - \Delta t) &= 2(\Delta t)\theta'(t) + \mathcal{O}((\Delta t)^3)\end{aligned}\tag{3.35}$$

που δίνουν

$$\begin{aligned}\theta(t + \Delta t) &= 2\theta(t) - \theta(t - \Delta t) + (\Delta t)^2\alpha(t) + \mathcal{O}((\Delta t)^4) \\ \omega(t) &= \frac{\theta(t + \Delta t) - \theta(t - \Delta t)}{2(\Delta t)} + \mathcal{O}((\Delta t)^2)\end{aligned}\tag{3.36}$$

που είναι οι σχέσεις (3.11).

Στις προσομοιώσεις το σημαντικό είναι το συνολικό σφάλμα που συσσωρεύεται μετά από τα $N - 1$ βήματα της ολοκλήρωσης. Ειδικά για τη συγκεκριμένη μέθοδο πρέπει να δούμε τα σφάλματα που συσσωρεύονται ιδιαίτερα προσεκτικά:

- Το σφάλμα στην ταχύτητα $\omega(t)$ δε συσσωρεύεται γιατί υπολογίζεται από τη διαφορά των θέσεων $\theta(t + \Delta t) - \theta(t - \Delta t)$.
- Στη θέση το σφάλμα συσσωρεύεται ως εξής: Έστω $\delta\theta(t)$ το συνολικό σφάλμα που έχει συσσωρευτεί από την ολοκλήρωση από χρόνο t_0 έως t . Τότε σύμφωνα με τα αναπτύγματα (3.36) το σφάλμα στο πρώτο βήμα είναι $\delta\theta(t_0 + \Delta t) = \mathcal{O}((\Delta t)^4)$. Τότε¹²

$$\begin{aligned}\theta(t_0 + 2\Delta t) &= 2\theta(t_0 + \Delta t) - \theta(t_0) + \Delta t^2\alpha(t_0 + \Delta t) + \mathcal{O}((\Delta t)^4) \Rightarrow \\ \delta\theta(t_0 + 2\Delta t) &= 2\delta\theta(t_0 + \Delta t) - \delta\theta(t_0) + \mathcal{O}((\Delta t)^4) \\ &= 2\mathcal{O}((\Delta t)^4) - 0 + \mathcal{O}((\Delta t)^4) \\ &= 3\mathcal{O}((\Delta t)^4)\end{aligned}$$

¹²Θυμίζουμε ότι η επιτάχυνση $\alpha(t)$ δίνεται, οπότε $\delta\alpha(t) = 0$.

Στα επόμενα βήματα παίρνουμε

$$\begin{aligned}\theta(t_0 + 3\Delta t) &= 2\theta(t_0 + 2\Delta t) - \theta(t_0 + \Delta t) + \Delta t^2 \alpha(t_0 + 2\Delta t) + \mathcal{O}((\Delta t)^4) \Rightarrow \\ \delta\theta(t_0 + 3\Delta t) &= 2\delta\theta(t_0 + 2\Delta t) - \delta\theta(t_0 + \Delta t) + \mathcal{O}((\Delta t)^4) \\ &= 6\mathcal{O}((\Delta t)^4) - \mathcal{O}((\Delta t)^4) + \mathcal{O}((\Delta t)^4) \\ &= 6\mathcal{O}((\Delta t)^4)\end{aligned}$$

$$\begin{aligned}\theta(t_0 + 4\Delta t) &= 2\theta(t_0 + 3\Delta t) - \theta(t_0 + 2\Delta t) + \Delta t^2 \alpha(t_0 + 3\Delta t) + \mathcal{O}((\Delta t)^4) \Rightarrow \\ \delta\theta(t_0 + 4\Delta t) &= 2\delta\theta(t_0 + 3\Delta t) - \delta\theta(t_0 + 2\Delta t) + \mathcal{O}((\Delta t)^4) \\ &= 12\mathcal{O}((\Delta t)^4) - 3\mathcal{O}((\Delta t)^4) + \mathcal{O}((\Delta t)^4) \\ &= 10\mathcal{O}((\Delta t)^4)\end{aligned}$$

Και επαγωγικά, αν $\delta\theta(t_0 + (n-1)\Delta t) = \frac{(n-1)n}{2}\mathcal{O}((\Delta t)^4)$ στο επόμενο βήμα παίρνουμε

$$\begin{aligned}\theta(t_0 + n\Delta t) &= 2\theta(t_0 + (n-1)\Delta t) - \theta(t_0 + (n-2)\Delta t) + \Delta t^2 \alpha(t_0 + (n-1)\Delta t) \\ &\quad + \mathcal{O}((\Delta t)^4) \Rightarrow \\ \delta\theta(t_0 + n\Delta t) &= 2\delta\theta(t_0 + (n-1)\Delta t) - \delta\theta(t_0 + (n-2)\Delta t) + \mathcal{O}((\Delta t)^4) \\ &= 2\frac{(n-1)n}{2}\mathcal{O}((\Delta t)^4) - \frac{(n-2)(n-1)}{2}\mathcal{O}((\Delta t)^4) + \mathcal{O}((\Delta t)^4) \\ &= \frac{n(n+1)}{2}\mathcal{O}((\Delta t)^4)\end{aligned}$$

Άρα τελικά

$$\delta\theta(t_0 + n\Delta t) = \frac{n(n+1)}{2}\mathcal{O}((\Delta t)^4) \sim \frac{1}{\Delta t^2}\mathcal{O}((\Delta t)^4) \sim \mathcal{O}((\Delta t)^2) \quad (3.37)$$

Άρα το ολικό σφάλμα είναι $\mathcal{O}((\Delta t)^2)$.

Για πληρότητα αναφέρουμε και τον αλγόριθμο Velocity Verlet ή μέθοδο Leapfrog. Στην περίπτωση αυτή χρησιμοποιούμε ρητά την ταχύτητα:

$$\begin{aligned}\theta_{n+1} &= \theta_n + \omega_n \Delta t + \frac{1}{2}\alpha_n \Delta t^2 \\ \omega_{n+\frac{1}{2}} &= \omega_n + \frac{1}{2}\alpha_n \Delta t \\ \omega_{n+1} &= \omega_{n+\frac{1}{2}} + \frac{1}{2}\alpha_{n+1} \Delta t.\end{aligned} \quad (3.38)$$

Στο τελευταίο βήμα χρειαζόμαστε την επιτάχυνση a_{n+1} οπότε πρέπει αυτή να εξαρτάται μόνο από τη θέση θ_{n+1} και όχι από την ταχύτητα.

Οι μέθοδοι Verlet είναι δημοφιλείς σε προσομοιώσεις molecular dynamics, κυρίως συστημάτων με πολλά σωματίδια. Έχουν το ιδιαίτερο προσόν ότι υλοποιούνται σχετικά εύκολα οι περιορισμοί (constraints) στους οποίους υπόκειται τα σωματίδια που αποτελούν το σύστημα.

3.7.1 Παράρτημα: Runge–Kutta 2ης τάξης

Στην παράγραφο αυτή θα δείξουμε με δύο τρόπους γιατί η επιλογή του ενδιάμεσου σημείου 2 στην Εξίσωση (3.17) μειώνει το σφάλμα κατά μία δύναμη του βήματος h . Όπως θα φανεί, η επιλογή του μέσου του διαστήματος για το σημείο 2 δεν είναι τυχαία (οπότε λ.χ. το σημείο με $t = t_n + 0.4h$ δεν θα είχε το ίδιο αποτέλεσμα). Πράγματι από τη σχέση

$$\frac{dx}{dt} = f(t, x) \Rightarrow x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} f(t, x) dt. \quad (3.39)$$

Αναπτύσσοντας κατά Taylor γύρω από το σημείο $(t_{n+1/2}, x_{n+1/2})$ παίρνουμε

$$f(t, x) = f(t_{n+1/2}, x_{n+1/2}) + (t - t_{n+1/2}) \frac{df}{dt}(t_{n+1/2}) + \mathcal{O}(h^2). \quad (3.40)$$

Οπότε

$$\begin{aligned} & \int_{t_n}^{t_{n+1}} f(t, x) dt \\ &= f(t_{n+1/2}, x_{n+1/2})(t_{n+1} - t_n) + \frac{df}{dt}(t_{n+1/2}) \frac{(t - t_{n+1/2})^2}{2} \Big|_{t_n}^{t_{n+1}} \\ & \quad + \mathcal{O}(h^2)(t_{n+1} - t_n) \\ &= f(t_{n+1/2}, x_{n+1/2})h + \frac{df}{dt}(t_{n+1/2}) \left\{ \frac{(t_{n+1} - t_{n+1/2})^2}{2} - \frac{(t_n - t_{n+1/2})^2}{2} \right\} \\ & \quad + \mathcal{O}(h^2)h \\ &= f(t_{n+1/2}, x_{n+1/2})h + \frac{df}{dt}(t_{n+1/2}) \left\{ \frac{h^2}{2} - \frac{(-h)^2}{2} \right\} + \mathcal{O}(h^3) \\ &= f(t_{n+1/2}, x_{n+1/2})h + \mathcal{O}(h^3). \end{aligned} \quad (3.41)$$

Παρατηρούμε ότι για την εξαφάνιση του όρου $\mathcal{O}(h)$ είναι αναγκαία η τοποθέτηση του βοηθητικού σημείου στο χρόνο $t_{n+1/2}$.

Η επιλογή αυτή δεν είναι μοναδική. Αυτό μπορεί να φανεί και από μια διαφορετική ανάλυση του αναπτύγματος κατά Taylor. Αναπτύσσοντας τώρα γύρω από το σημείο (t_n, x_n) παίρνουμε

$$\begin{aligned}
 x_{n+1} &= x_n + (t_{n+1} - t_n) \frac{dx_n}{dt} + \frac{1}{2} (t_{n+1} - t_n)^2 \frac{d^2 x_n}{dt^2} + \mathcal{O}(h^3) \\
 &= x_n + hf_n + \frac{h^2}{2} \frac{df_n}{dt} + \mathcal{O}(h^3) \\
 &= x_n + hf_n + \frac{h^2}{2} \left(\frac{\partial f_n}{\partial t} + \frac{\partial f_n}{\partial x} \frac{dx_n}{dt} \right) + \mathcal{O}(h^3) \\
 &= x_n + hf_n + \frac{h^2}{2} \left(\frac{\partial f_n}{\partial t} + \frac{\partial f_n}{\partial x} f_n \right) + \mathcal{O}(h^3), \tag{3.42}
 \end{aligned}$$

όπου για συντομία θέσαμε $f_n \equiv f(t_n, x_n)$, $\frac{dx_n}{dt} \equiv \frac{dx}{dt}(x_n)$ κ.ο.κ. Ορίζουμε

$$\begin{aligned}
 k_1 &= f(t_n, x_n) = f_n \\
 k_2 &= f(t_n + ah, x_n + bhk_1) \\
 x_{n+1} &= x_n + h(c_1 k_1 + c_2 k_2). \tag{3.43}
 \end{aligned}$$

και θα προσδιορίσουμε τις συνθήκες έτσι ώστε το σφάλμα οι όροι $\mathcal{O}(h^2)$ της τελευταίας εξίσωσης να ταυτίζονται με αυτούς της (3.42). Αναπτύσσοντας την k_2 παίρνουμε

$$\begin{aligned}
 k_2 &= f(t_n + ah, x_n + bhk_1) \\
 &= f(t_n, x_n + bhk_1) + ha \frac{\partial f}{\partial t}(t_n, x_n + bhk_1) + \mathcal{O}(h^2) \\
 &= f(t_n, x_n) + hb k_1 \frac{\partial f}{\partial x}(t_n, x_n) + ha \frac{\partial f}{\partial t}(t_n, x_n) + \mathcal{O}(h^2) \\
 &= f_n + h \left\{ a \frac{\partial f_n}{\partial t} + b k_1 \frac{\partial f_n}{\partial x} \right\} + \mathcal{O}(h^2) \\
 &= f_n + h \left\{ a \frac{\partial f_n}{\partial t} + b f_n \frac{\partial f_n}{\partial x} \right\} + \mathcal{O}(h^2) \tag{3.44}
 \end{aligned}$$

Αντικαθιστώντας στην (3.43) παίρνουμε

$$\begin{aligned}
 x_{n+1} &= x_n + h(c_1 k_1 + c_2 k_2) \\
 &= x_n + h \left\{ c_1 f_n + c_2 f_n + c_2 h \left(a \frac{\partial f_n}{\partial t} + b f_n \frac{\partial f_n}{\partial x} \right) + \mathcal{O}(h^2) \right\} \\
 &= x_n + h(c_1 + c_2) f_n + \frac{h^2}{2} \left((2c_2 a) \frac{\partial f_n}{\partial t} + (2c_2 b) f_n \frac{\partial f_n}{\partial x} \right) \\
 &\quad + \mathcal{O}(h^3).
 \end{aligned} \tag{3.45}$$

Αρκεί λοιπόν να επιλέξουμε

$$\begin{aligned}
 c_1 + c_2 &= 1 \\
 2c_2 a &= 1 \\
 2c_2 b &= 1.
 \end{aligned} \tag{3.46}$$

Επιλογή $c_1 = 0$, $c_2 = 1$, $a = b = 1/2$ μάς δίνει τη μέθοδο (3.19). Άλλες επιλογές στη βιβλιογραφία είναι $c_2 = 1/2$ και $c_2 = 3/4$.

3.8 Ασκήσεις

- 3.1 Αποδείξτε ότι το συνολικό σφάλμα στη μέθοδο Euler-Cromer είναι τάξης Δt
- 3.2 Αναπαράγετε τα αποτελέσματα των σχημάτων 3.11–3.18
- 3.3 Βελτιώστε τον υπολογισμό του χρόνου στα προγράμματα των μεθόδων Euler, Euler-Cromer, Euler-Verlet, Runge-Kutta ώστε να μην έχουμε προσθετική συσσώρευση σφαλμάτων στο χρόνο όταν η παράμετρος h είναι πολύ μικρή (λ.χ. στις εντολές $T(i)=T(i-1)+h$). Επαναλάβετε την ανάλυση της προηγούμενης άσκησης.
- 3.4 Μεταβάλλετε τα προγράμματα των μεθόδων Euler, Euler-Cromer, Euler-Verlet, Runge-Kutta ώστε `REAL`→`REAL*8` (προσοχή στις σταθερές: να της ορίσετε ρητά να είναι διπλής ακρίβειας προσθέτοντας τον εκθέτη `D0`). Επαναλάβετε την ανάλυση της προηγούμενης άσκησης.
- 3.5 Να επαναλάβετε τη σύγκριση των μεθόδων Euler, Euler-Cromer, Euler-Verlet, Runge-Kutta για τα παρακάτω συστήματα των οποίων η αναλυτική λύση είναι γνωστή:
- (α') Σωματίδιο που πέφτει ελεύθερα στο ομογενές πεδίο βαρύτητας. Θεωρήστε $v(0) = 0$, $m = 1$, $g = 10$.
 - (β') Σωματίδιο που πέφτει στο ομογενές πεδίο βαρύτητας μέσα σε ρευστό από το οποίο δέχεται δύναμη $F = -kv$. Θεωρήστε $v(0) = 0$, $m = 1$, $g = 10$, $k = 0.1, 1.0, 2.0$. Υπολογίστε την οριζική ταχύτητα του σωματιδίου αριθμητικά και συγκρίνετέ τη με τη θεωρητική της τιμή.
 - (γ') Επαναλάβετε για δύναμη αντίστασης $F = -kv^2$.
- 3.6 Μελετήστε το σύστημα του αρμονικού ταλαντωτή με απόσβεση

$$\frac{d^2x}{dt^2} + \gamma \frac{dx}{dt} + \omega_0^2 x = 0. \quad (3.47)$$

Πάρτε $\omega_0 = 3.145$, $\gamma = 0.5$ και υπολογίστε την ενέργεια συναρτήσει του χρόνου. Είναι η τιμή της μονοτονική; Γιατί; (Δείξτε ότι $d(E/m)/dt = -\gamma v^2$). Επαναλάβετε για $\gamma = 4, 5, 6, 7, 8$. Πότε το σύστημα ταλαντώνεται και πότε όχι; Υπολογίστε αριθμητικά την κρίσιμη τιμή του γ για την

οποία το σύστημα μετατρέπεται από ταλαντούμενο σε μή ταλαντούμενο. Συγκρίνετε το αποτέλεσμα σας με την αναλυτική λύση.

3.7 Αναπαράγετε τα αποτελέσματα που οδηγούν στα Σχήματα 3.19–3.22.

3.8 Αναπαράγετε τα αποτελέσματα που οδηγούν στα Σχήματα 3.23–3.26. Υπολογίστε αριθμητικά τη φάση $\delta(\omega)$ και επιβεβαιώστε τη Σχέση (3.33).

3.9 Θεωρήστε το μοντέλο για μια κούνια να είναι ο αρμονικός ταλαντωτής με απόσβεση ο οποίος υπόκειται σε περιοδική δύναμη η οποία είναι στιγμιαία ώθηση συχνότητας ω . Ορίστε το ‘στιγμιαία’ να είναι η ώθηση να δίνει επιτάχυνση a_0 το κατάλληλο χρονικό διάστημα διάρκειας Δt και 0 στα υπόλοιπα βήματα. Υπολογίστε το πλάτος $x_0(\omega)$ για $\omega_0 = 3.145$ και $\gamma = 0.5$.

3.10 Θεωρήστε την εξωτερική δύναμη να είναι το ‘μισό συνημίτονο’ δίνοντας επιτάχυνση στον ο αρμονικό ταλαντωτή με απόσβεση

$$a(t) = \begin{cases} a_0 \cos \omega t & \cos \omega t > 0 \\ 0 & \cos \omega t \leq 0 \end{cases}$$

Μελετήστε τη μετάβαση του συστήματος στη σταθερή κατάσταση και υπολογίστε το πλάτος $x_0(\omega)$ για $\omega_0 = 3.145$ και $\gamma = 0.5$.

3.11 Θεωρήστε την εξωτερική δύναμη να δίνει επιτάχυνση στον ο αρμονικό ταλαντωτή με απόσβεση

$$a(t) = \frac{1}{\pi} + \frac{1}{2} \cos \omega t + \frac{2}{3\pi} \cos 2\omega t - \frac{2}{15\pi} \cos 4\omega t$$

Μελετήστε τη μετάβαση του συστήματος στη σταθερή κατάσταση και υπολογίστε το πλάτος $x_0(\omega)$ για $\omega_0 = 3.145$ και $\gamma = 0.5$. Συγκρίνετε τα αποτελέσματά σας με αυτά της προηγούμενης άσκησης. Τι συμπεραίνετε;

3.12 Γράψτε ένα πρόγραμμα που να προσομοιώνει ταυτόχρονα N όμοιους αρμονικούς ταλαντωτές. Πάρτε $N = 20$ και δώστε τυχαίες αρχικές συνθήκες σε κάθε ένα από αυτούς και παρακολουθήστε τις τροχιές τους στο χώρο των φάσεων. Παρατηρήστε αν οι τροχιές τέμνονται και εξηγήστε τα αποτελέσματά σας.

- 3.13 Στην προηγούμενη άσκηση τοποθετήστε τους $N = 20$ αρμονικούς ταλαντωτές στο χώρο των φάσεων σε ένα μικρό τετράγωνο με κέντρο την αρχή των αξόνων. Παρακολουθήστε την εξέλιξη του συστήματος στο χρόνο. Αλλάζει το σχήμα με το χρόνο; Αλλάζει το εμβαδόν; Εξηγήστε...
- 3.14 Επαναλάβετε την προηγούμενη άσκηση όταν υπάρχει απόσβεση με $\gamma = 0.5$. Πάρτε $\omega_0 = 3.145$.
- 3.15 Στην περίπτωση του εκκρεμούς με απόσβεση και εξωτερική δύναμη, πάρτε $\omega = 2$, $\omega_0 = 1.0$, $\gamma = 0.2$ μελετήστε την παροδική φάση στα διαγράμματα $\theta(t)$, $\dot{\theta}(t)$ για $A = 0.1, 0.5, 0.79, 0.85, 1.03, 1.4$.
- 3.16 Στην περίπτωση του εκκρεμούς με απόσβεση και εξωτερική δύναμη, πάρτε $\omega = 2$, $\omega_0 = 1.0$, $\gamma = 0.2$ μελετήστε τις τροχιές στο χώρο της φάσης για $A = 0.1, 0.19, 0.21, 0.25, 0.5, 0.71, 0.79, 0.85, 1.02, 1.031, 1.033, 1.05, 1.08, 1.1, 1.4, 1.8, 3.1, 3.5, 3.8, 4.2, 4.42, 4.44, 4.445, 4.447, 4.4488$.
- 3.17 Αναπαράγετε τα Σχήματα 3.30.
- 3.18 Αναπαράγετε τα Σχήματα 3.31.

Κεφάλαιο 4

Κίνηση στο Επίπεδο

4.1 Runge–Kutta στις δύο διαστάσεις.

Στις δύο διαστάσεις, το πρόβλημα αρχικών τιμών που έχουμε να λύσουμε δίνεται από το σύστημα (3.6)

$$\begin{aligned}\frac{dx}{dt} &= v_x & \frac{dv_x}{dt} &= a_x(t, x, v_x, y, v_y) \\ \frac{dy}{dt} &= v_y & \frac{dv_y}{dt} &= a_y(t, x, v_x, y, v_y) .\end{aligned}\quad (4.1)$$

Ο κώδικας που θα τρέχει τη μέθοδο Runge–Kutta 4ης τάξης προκύπτει με μικρές μετατροπές του κώδικα `rk.f`. Κατ' αρχήν για διευκόλυνση της μελέτης πολλών δυνάμεων ξεχωρίζουμε τον κοινό κώδικα με το user interface και τον αλγόριθμο της μεθόδου από τις συναρτήσεις της επιτάχυνσης που αλλάζουν ανάλογα με τη δύναμη σε ξεχωριστά αρχεία. Στο αρχείο `rk2.f` τοποθετούμε τα πρώτα και σε αρχείο `rk_XXX.f` τα δεύτερα. XXX είναι ακολουθία χαρακτήρων που ταυτοποιούν τη δύναμη λ.χ. `rk2_hoc.f` έχει την επιτάχυνση του αρμονικού ταλαντωτή, `rk2_g.f` την επιτάχυνση από ομογενές πεδίο βαρύτητας $\vec{g} = -g\hat{j}$ κ.ο.κ.

Στον κώδικα στο `rk2.f` κάνουμε μερικές μικροαλλαγές στο user interface. Τοποθετούμε δυο απροσδιόριστες σταθερές σύζευξης `k1`, `k2` που μπορούν να δοθούν διαδραστικά από το χρήστη και να προσδιορίσουν το μέγεθος της δύναμης που ασκείται κάθε φορά στο σώμα. Τοποθετούνται σε common block

```
double precision k1,k2
common /couplings/k1,k2
```

το οποίο θα ‘φαίνεται’ και από τις συναρτήσεις επιτάχυνσης $f3$, $f4$ και της ενέργειας $energy$. Ο χρήστης πρέπει τώρα να παρέχει τις αρχικές συνθήκες και για τις δύο συντεταγμένες στο επίπεδο x , y . Αυτές αντιστοιχούν στις μεταβλητές $X10 \leftrightarrow x_0$, $X20 \leftrightarrow y_0$, $V10 \leftrightarrow v_{x0}$, $V20 \leftrightarrow v_{y0}$, ενώ οι συναρτήσεις του χρόνου αντιστοιχούν στα arrays $X1(P) \leftrightarrow x(t)$, $X2(P) \leftrightarrow y(t)$, $V1(P) \leftrightarrow v_x(t)$, $V2(P) \leftrightarrow v_y(t)$. Η ολοκλήρωση γίνεται όπως και πριν καλώντας

```
call RK(T,X1,X2,V1,V2,T0,TF,X10,X20,V10,V20,STEPS,PSIZE)
```

και στο αρχείο `rk2.dat` αποθηκεύουμε τα αποτελέσματα μαζί με τη συνολική μηχανική ενέργεια που υπολογίζεται από τη συνάρτηση $energy(t,x1,x2,v1,v2)$ η οποία βρίσκεται στο ίδιο αρχείο με τις επιταχύνσεις μια και η μορφή της εξαρτάται από τον τύπο της δύναμης:

```
open(unit=11,file='rk2.dat')
do i=1,STEPS+1
  write(11,*)T(i),X1(i),X2(i),V1(i),V2(i),
*      energy(T(i),X1(i),X2(i),V1(i),V2(i))
enddo
```

Τέλος αλλαγές πρέπει να γίνουν στον κώδικα της βασικής υπορουτίνας `RKSTEP(t,x1,x2,x3,x4,dt)` λόγω του μεγαλύτερου αριθμού μεταβλητών στο πρόβλημα. Παραθέτουμε ολόκληρο τον κώδικα για να διευκολύνουμε τον αναγνώστη¹:

```
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Program to solve a 4 ODE system using Runge-Kutta Method
C      User must supply derivatives
C      dx1/dt=f1(t,x1,x2,x3,x4) dx2/dt=f2(t,x1,x2,x3,x4)
C      dx3/dt=f3(t,x1,x2,x3,x4) dx4/dt=f4(t,x1,x2,x3,x4)
C      as real*8 functions
C      Output is written in file rk2.dat
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      program rk2_solve
C      implicit none
C      integer P
C      parameter(P=510000)
C      double precision T0,TF,X10,X20,V10,V20
```

¹double precision είναι συνώνυμο με real*8.


```

integer STEPS,PSIZE
double precision T(P),X1(P),X2(P),V1(P),V2(P)
integer i
double precision k1,k2
common /couplings/k1,k2
double precision energy

C      Input:
print *, 'Runge-Kutta Method for 4-ODEs Integration'
print *, 'Enter coupling constants:'
read(5,*) k1,k2
print *, 'k1= ',k1,' k2= ',k2
print *, 'Enter STEPS,T0,TF,X10,X20,V10,V20:'
read(5,*) STEPS,T0,TF,X10,X20,V10,V20
print *, 'No. Steps= ',STEPS
print *, 'Time: Initial T0 =',T0,' Final TF=',TF
print *, '          X1(T0)=',X10,' X2(T0)=',X20
print *, '          V1(T0)=',V10,' V2(T0)=',V20

C      The Calculation:
PSIZE=P
call RK(T,X1,X2,V1,V2,T0,TF,X10,X20,V10,V20,STEPS,PSIZE)

C      Output:
open(unit=11,file='rk2.dat')
do i=1,STEPS+1
  write(11,*)T(i),X1(i),X2(i),V1(i),V2(i),
*      energy(T(i),X1(i),X2(i),V1(i),V2(i))
enddo
close(11)

end

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      The velocity functions f1,f2(t,x1,x2,v1,v2)
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
double precision function f1(t,x1,x2,v1,v2)
implicit none

```

```

double precision t,x1,x2,v1,v2
f1=v1          !dx1/dt= v1
end

double precision function f2(t,x1,x2,v1,v2)
implicit none
double precision t,x1,x2,v1,v2
f2=v2          !dx2/dt= v2
end

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      RK(T,X1,X2,V1,V2,T0,TF,X10,X20,V10,V20,STEPS,PSIZE) is the driver
C      for the Runge-Kutta integration routine RKSTEP
C      Input: Initial and final times T0,T1
C              Initial values at t=T0  X10,X20,V10,V20
C              Number of steps of integration STEPS
C              Size of arrays T,X1,X2,V1,V2
C      Output: real arrays T(PSIZE),X1(PSIZE),X2(PSIZE),
C              V1(PSIZE),V2(PSIZE) where
C      T(1) = T0 X1(1) = X10 X2(1) = X20 V1(1) = V10 V2(1) = V20
C              X1(i) = X1(at t=T(i)) X2(i) = X2(at t=T(i))
C              V1(i) = V1(at t=T(i)) V2(i) = V2(at t=T(i))
C      T(STEPS+1)=TF
C      Therefore we must have PSIZE>STEPS
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      subroutine RK(T,X1,X2,V1,V2,T0,TF,X10,X20,V10,V20,STEPS,PSIZE)
C      implicit none
C      integer STEPS,PSIZE
C      double precision T (PSIZE),T0,TF
C      double precision X1(PSIZE),X2(PSIZE),X10,X20
C      double precision V1(PSIZE),V2(PSIZE),V10,V20
C      double precision dt
C      double precision TS,X1S,X2S !values of time and X1,X2 at given step
C      double precision V1S,V2S
C      integer i

C      Some checks:
C      if(STEPS .le. 1 )then

```



```

C      real function f1(t,x1,x2,x3,x4)
C      real function f2(t,x1,x2,x3,x4)
C      real function f3(t,x1,x2,x3,x4)
C      real function f4(t,x1,x2,x3,x4)
C      Given initial point (t,x1,x2) the routine advances it
C      by time dt.
C      Input : Initial time t      and function values x1,x2,x3,x4
C      Output: Final   time t+dt and function values x1,x2,x3,x4
C      Careful!: values of t,x1,x2,x3,x4 are overwritten...
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      subroutine RKSTEP(t,x1,x2,x3,x4,dt)
      implicit none
      double precision t,x1,x2,x3,x4,dt
      double precision f1,f2,f3,f4
      double precision k11,k12,k13,k14,k21,k22,k23,k24
      double precision k31,k32,k33,k34,k41,k42,k43,k44
      double precision h,h2,h6

      h =dt                                !h =dt, integration step
      h2=0.5D0*h                            !h2=h/2
      h6=0.1666666666666666666666666666D0*h !h6=h/6

      k11=f1(t,x1,x2,x3,x4)
      k21=f2(t,x1,x2,x3,x4)
      k31=f3(t,x1,x2,x3,x4)
      k41=f4(t,x1,x2,x3,x4)

      k12=f1(t+h2,x1+h2*k11,x2+h2*k21,x3+h2*k31,x4+h2*k41)
      k22=f2(t+h2,x1+h2*k11,x2+h2*k21,x3+h2*k31,x4+h2*k41)
      k32=f3(t+h2,x1+h2*k11,x2+h2*k21,x3+h2*k31,x4+h2*k41)
      k42=f4(t+h2,x1+h2*k11,x2+h2*k21,x3+h2*k31,x4+h2*k41)

      k13=f1(t+h2,x1+h2*k12,x2+h2*k22,x3+h2*k32,x4+h2*k42)
      k23=f2(t+h2,x1+h2*k12,x2+h2*k22,x3+h2*k32,x4+h2*k42)
      k33=f3(t+h2,x1+h2*k12,x2+h2*k22,x3+h2*k32,x4+h2*k42)
      k43=f4(t+h2,x1+h2*k12,x2+h2*k22,x3+h2*k32,x4+h2*k42)

      k14=f1(t+h ,x1+h *k13,x2+h *k23,x3+h *k33,x4+h2*k43)

```

```

k24=f2(t+h ,x1+h *k13,x2+h *k23,x3+h *k33,x4+h2*k43)
k34=f3(t+h ,x1+h *k13,x2+h *k23,x3+h *k33,x4+h2*k43)
k44=f4(t+h ,x1+h *k13,x2+h *k23,x3+h *k33,x4+h2*k43)

t =t+h
x1=x1+h6*(k11+2.0D0*(k12+k13)+k14)
x2=x2+h6*(k21+2.0D0*(k22+k23)+k24)
x3=x3+h6*(k31+2.0D0*(k32+k33)+k34)
x4=x4+h6*(k41+2.0D0*(k42+k43)+k44)

end

```

4.2 Βολές στο Βαρυτικό Πεδίο της Γης.

Θεωρούμε αρχικά σωματίδιο υπό την επίδραση δύναμης που του προσδίδει επιτάχυνση $\vec{g} = -mg \hat{j}$

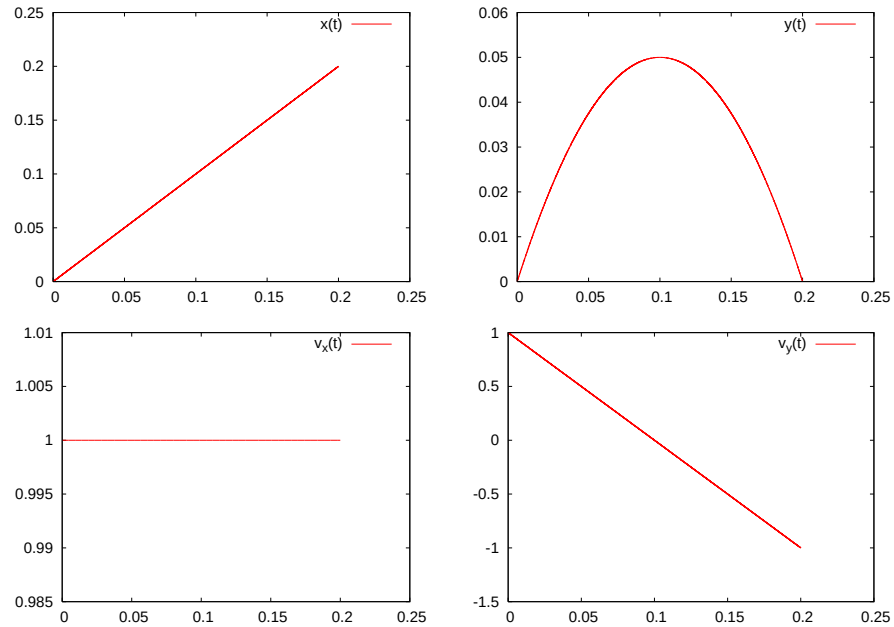
$$\begin{aligned}
 x(t) &= x_0 + v_{0x}t \\
 y(t) &= y_0 + v_{0y}t - \frac{1}{2}gt^2 \\
 v_x(t) &= v_{0x} \\
 v_y(t) &= v_{0y} - gt \\
 a_x(t) &= 0 \\
 a_y(t) &= -g
 \end{aligned} \tag{4.2}$$

Το σωματίδιο, όπως γνωρίζουμε καλά, κινείται πάνω σε μια παραβολή στην οποία εμείς απλά διαλέγουμε το σημείο στο οποίο τοποθετείται αρχικά το σωματίο:

$$\begin{aligned}
 (y - y_0) &= \left(\frac{v_{0y}}{v_{0x}} \right) (x - x_0) - \frac{1}{2} \frac{g}{v_{0x}^2} (x - x_0)^2 \\
 &= \tan \theta (x - x_0) - \frac{\tan^2 \theta}{4h_{\max}} (x - x_0)^2,
 \end{aligned} \tag{4.3}$$

όπου $\tan \theta = v_{0y}/v_{0x}$ και $h_{\max}$ η γωνία υπό την οποία βάλλεται το σωματίο και το μέγιστο ύψος που φτάνει το σωματίο ως προς το αρχικό σημείο βολής.

Κωδικοποιούμε την επιτάχυνση $a_x(t) = 0$ $a_y(t) = -g$ ($a_x \leftrightarrow \mathbf{f3}$, $a_y \leftrightarrow \mathbf{f4}$) καθώς και τη συνολική μηχανική ενέργεια στο αρχείο `rk2_g.f`:



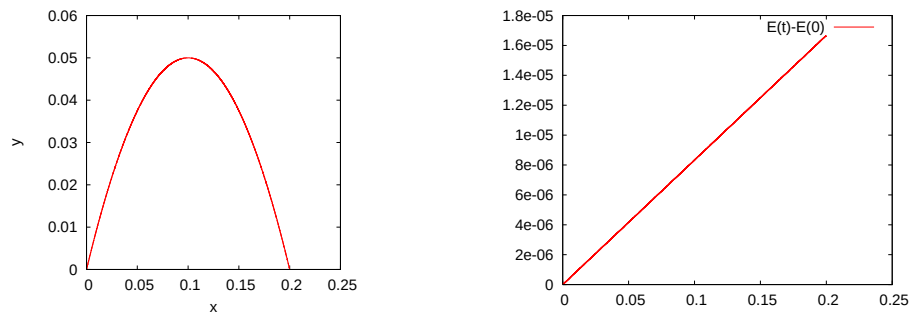
Σχήμα 4.1: Βολή στο βαρυτικό πεδίο με ένταση $\vec{g} = -10.0\hat{j}$ και αρχική ταχύτητα $\vec{v}_0 = \hat{i} + \hat{j}$. Δίνονται τα διαγράμματα $x(t)$, $y(t)$, $v_x(t)$, $v_y(t)$.

```

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      The acceleration functions f3,f4(t,x1,x2,v1,v2) provided by the user
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Free fall in constant gravitational field with
C      g = -k2
C      double precision function f3(t,x1,x2,v1,v2)
C      implicit none
C      double precision t,x1,x2,v1,v2
C      double precision k1,k2
C      common /couplings/k1,k2
C      f3=0.0D0      !dx3/dt=dv1/dt=a1
C      end

C      double precision function f4(t,x1,x2,v1,v2)
C      implicit none
C      double precision t,x1,x2,v1,v2

```



Σχήμα 4.2: Βολή στο βαρυτικό πεδίο με ένταση $\vec{g} = -10.0\hat{j}$ και αρχική ταχύτητα $\vec{v}_0 = \hat{i} + \hat{j}$. Φαίνεται η παραβολική τροχιά που ακολουθεί το σωματίδιο. Στο διπλανό σχήμα παρακολουθούμε την απόκλιση της μηχανικής ενέργειας του σωματιδίου από την αρχική της τιμή.

```
double precision k1,k2
common /couplings/k1,k2
f4=-k1      !dx4/dt=dv2/dt=a2
end

double precision function energy(t,x1,x2,v1,v2)
implicit none
double precision t,x1,x2,v1,v2
double precision k1,k2
common /couplings/k1,k2
energy = 0.5*(v1*v1+v2*v2) + k1*x2
end
```

Στη συνέχεια παραθέτουμε τη σειρά εντολών που δίνει ο χρήστης για να υπολογίσει την τροχιά

```
> f77 -O2 rk2.f rk2_g.f -o rk2
> ./rk2
Runge-Kutta Method for 4-ODEs Integration
Enter coupling constants:
10.0 0.0
k1= 10.000000000000000 k2= 0.0000000000000000
Enter STEPS,T0,TF,X10,X20,V10,V20:
```

```
20000 0.0 0.2 0.0 0.0 1.0 1.0
```

```
No. Steps= 20000
```

```
Time: Initial T0 = 0.0000000000000000 Final TF= 0.2000000000000000
```

```
X1(T0)= 0.0000000000000000 X2(T0)= 0.0000000000000000
```

```
V1(T0)= 1.0000000000000000 V2(T0)= 1.0000000000000000
```

Στη συνέχεια επεξεργαζόμαστε τα αποτελέσματά μας αναλύοντας τα δεδομένα από το αρχείο `rk2.dat` με το πρόγραμμα `gnuplot`:

```
> gnuplot
gnuplot> set terminal x11 1
gnuplot> plot "rk2.dat" using 1:2 with lines title "x(t)"
gnuplot> set terminal x11 2
gnuplot> plot "rk2.dat" using 1:3 with lines title "y(t)"
gnuplot> set terminal x11 3
gnuplot> plot "rk2.dat" using 1:4 with lines title "vx(t)"
gnuplot> set terminal x11 4
gnuplot> plot "rk2.dat" using 1:5 with lines title "vy(t)"
gnuplot> set terminal x11 5
gnuplot> plot "rk2.dat" using 1:($6-1.0) with lines title "E(t)-E(0)"
gnuplot> set terminal x11 6
gnuplot> set size square
gnuplot> set title "Trajectory"
gnuplot> plot "rk2.dat" using 2:3 with lines notit
```

Τα αποτελέσματα φαίνονται στο Σχήματα 4.1 και 4.2. Παρατηρούμε μικρή αύξηση της ενέργειας που μας δίνει και το μέτρο της ακρίβειας της μεθόδου.

Με τη βοήθεια του `gnuplot` μπορούμε να φτιάξουμε κινούμενα σχέδια της τροχιάς. Για το λόγο αυτό ομαδοποιούμε μερικές εντολές του `gnuplot` σε αρχείο σεναρίου, έστω στο `rk2_animate.gpl`

```
icount = icount+skip
plot "<cat -n rk2.dat" \
    using 3:($1<= icount ? $4: 1/0) with lines notitle
# pause 1
if(icount < nlines ) reread
```

Το παραπάνω αρχείο υποθέτει ότι όταν τρέξουμε το `gnuplot` έχουμε αρχικοποιήσει τις μεταβλητές `icount`, `skip`, `nlines` να είναι οι τιμές του αρχικού

αριθμού γραμμών του αρχείου `rk2.dat` που θα μπουν στο διάγραμμα, ο αριθμός γραμμών που θα προστίθενται σε κάθε καινούργιο πλαίσιο που σχεδιάζεται στα κινούμενα σχέδια και ο συνολικός αριθμός γραμμών που περιέχει το αρχείο ώστε να σταματήσει η διαδικασία. Η ιδέα είναι ότι οι εντολές του αρχείου διαβάζονται από το `gnuplot` κάνοντας ένα `plot` και αν πληρήται το κριτήριο του `if` το αρχείο ξαναδιαβάζεται με την εντολή `reread`. Ας εξηγήσουμε την γραμμή με την εντολή `plot`: Το 'αρχείο' "`<cat -n rk2.dat`" είναι το standard output της εντολής `cat -n rk2.dat` η οποία τυπώνει στο standard output το αρχείο `rk2.dat` βάζοντας στην πρώτη στήλη τον αριθμό γραμμής που διαβάζεται. Έτσι η εντολή `plot` διαβάζει δεδομένα στα οποία η πρώτη στήλη είναι ο αριθμός γραμμής, η δεύτερη ο χρόνος, η τρίτη η συντεταγμένη x , η τέταρτη η συντεταγμένη y κ.ο.κ. Η γραμμή

```
using 3:($1<= icount ? $4: 1/0)
```

λέει να χρησιμοποιηθεί η 2η στήλη στον οριζόντιο άξονα και αν η πρώτη στήλη είναι μικρότερη από την τιμή της μεταβλητής `icount` να μπει στον κατακόρυφο άξονα η τιμή της 4ης στήλης αλλιώς τίποτα (βάζοντας κάτι που δεν είναι νόμιμο, όπως διαίρεση με το 0 κάνει το `gnuplot` να αγνοήσει το συγκεκριμένο σημείο). Με τον τρόπο αυτό καθώς η τιμή της μεταβλητής `icount` αυξάνει τοποθετούμε στο διάγραμμα περισσότερα σημεία της τροχιάς δημιουργώντας την ψευδαίσθηση της κίνησης. τη γραμμή με την εντολή `pause` την έχουμε βάλει στα σχόλια. Αν τα κινούμενα είναι πολύ γρήγορα για σας, βγάλτε το χαρακτήρα του σχολίου `#` και αντικαταστήστε τη μονάδα με τον αριθμό δευτερολέπτων που θέλετε να σταματάει κάθε πλαίσιο. Για να χρησιμοποιήσουμε το σενάριο αυτό από το `gnuplot` δίνουμε τις εντολές

```
> gnuplot
gnuplot> icount = 10
gnuplot> skip    = 200
gnuplot> nlines = 20000
gnuplot> load "rk2_animate.gpl"
```

Τα παραπάνω σενάρια θα τα βρείτε στο συνοδευτικό λογισμικό του κεφαλαίου. Εκεί θα βρείτε και σενάρια φλοιού τα οποία θα σας βοηθήσουν να αυτοματοποιήσετε πολλές από τις εντολές που περιγράψαμε παραπάνω. Περιγράφουμε τη χρήση δύο από αυτών. Πρώτα το σενάριο `rk2_animate.csh`:

```
> rk2_animate.csh -h
Usage: rk2_animate.csh -t [sleep time] -d [skip points] <file>
```

Default file is rk2.dat

Other options:

- x: set lower value in xrange
- X: set lower value in xrange
- y: set lower value in yrange
- Y: set lower value in yrange
- r: automatic determination of x-y range

```
> rk2_animate.csh -r -d 500 rk2.dat
```

Η τελευταία γραμμή πραγματοποιεί τα κινούμενα σχέδια με πλαίσια που κάθε φορά έχουν 500 παραπάνω σημεία, ενώ τα όρια των πλαισίων υπολογίζονται αυτόματα από το σενάριο με το διακόπτη `-r`. Ο διακόπτης `-h` δίνει σύντομες οδηγίες για τη χρήση του σεναρίου, μια σύμβαση που την ακολουθούμε συχνά στα σεναρία/προγράμματα που γράφουμε.

Ένα πιο πλήρες σενάριο που κάνει όλες τις δουλειές είναι το `rk2.csh`. Οδηγίες χρήσης παίρνουμε με την εντολή

```
> ./rk2.csh -h
```

Usage: rk2.csh -f <force> k1 k2 x10 x20 v10 v20 STEPS t0 tf

Other Options:

- n Do not animate trajectory

Available forces (value of <force>):

- | | | |
|-----------------------------|--------------------------|---|
| 1: ax=-k1 | ay= -k2 y | Harmonic oscillator |
| 2: ax= 0 | ay= -k1 | Free fall |
| 3: ax= -k2 vx | ay= -k2 vy - k1 | Free fall + air resistance ~ v |
| 4: ax= -k2 v vx | ay= -k2 v vy - k1 | Free fall + air resistance ~ v ² |
| 5: ax= k1*x1/r ³ | ay= k1*x2/r ³ | Coulomb Force |
| | | |

όπου φαίνεται ότι έχουμε την επιλογή να τρέξουμε το πρόγραμμα με διαφορετικές δυνάμεις που επιλέγονται με το διακόπτη `-f`. Στην υπόλοιπη γραμμή εντολών δίνουμε τα δεδομένα εισόδου για το πρόγραμμα `rk2.f`, τις σταθερές ζεύξης `k1`, `k2`, τις αρχικές συνθήκες `x10`, `x20`, `v10`, `v20` και τις συνθήκες ολοκλήρωσης `STEPS`, `t0`, `tf`. Έτσι για παράδειγμα οι εντολές

```
> rk2.csh -f 2 -- 10.0 0.0 0.0 0.0 1.0 1.0 20000 0.0 0.2
> rk2.csh -f 1 -- 16.0 1.0 0.0 1.0 1.0 0.0 20000 0.0 6.29
> rk2.csh -f 5 -- 10.0 0.0 -10 0.2 10. 0.0 20000 0.0 3.00
```

μας δίνουν την κίνηση του σωματιδίου στο πεδίο βαρύτητας που μελετήσαμε ως τώρα, την κίνηση ανομοιογενούς αρμονικού ταλαντωτή ($\mathbf{k1} = a_x = -\omega_1^2 x$, $\mathbf{k2} = a_y = -\omega_2^2 y$) και τη σκέδαση φορτίου σε πεδίο Coulomb - δοκιμάστε τα! Ελπίζω να σας δημιουργηθεί και η περιέργεια να δείτε ‘μέσα’ στα σενάρια έτσι ώστε να τα μεταβάλλετε και δημιουργήτε από μόνοι/ες σας. Από μένα μερικές οδηγίες για τους τεμπέληδες: Αν θελήσετε να προσθέσετε μια δική σας δύναμη στο ρεπερτόριο του σεναρίου ακολουθήστε τη συνταγή: Προγραμματίστε τη δύναμή σας σε αρχείο με όνομα `rk2_myforce.f` σύμφωνα με τις προδιαγραφές του `rk2_g.f`. Επεξεργαστήτε το αρχείο `rk2.csh` και αλλάξτε τη γραμμή

```
set forcecode = (hoc g vg v2g cb)
```

σε

```
set forcecode = (hoc g vg v2g cb myforce)
```

(φυσικά μπορεί η μεταβλητή `$forcecode` να έχει και άλλες εγγραφές στο σενάριο αλλά αυτό δεν θα σας εμποδίσει). Μετρήστε σε ποιά σειρά έχετε βάλει το `myforce`, εδώ την 6η, και τρέξτε την εντολή με το διακόπτη `-f 6` όπου το 6 αντικαταστήστε το με τη σειρά στο δικό σας σενάριο (οι τελίτσες οι δικιές σας σταθ. ζεύξης και αρχικές συνθήκες):

```
> rk2.csh -f 6 -- .....
```

Ας μελετήσουμε τώρα την επίδραση της αντίστασης του αέρα ή ενός ρευστού στην πτώση/βολή του σωματιδίου. Για μικρές ταχύτητες η αντίσταση γίνεται ανάλογη της ταχύτητας και έχουμε $\vec{F}_r = -mk\vec{v}$ οπότε

$$\begin{aligned} a_x &= -kv_x \\ a_y &= -kv_y - g \end{aligned} \quad (4.4)$$

παίρνοντας

$$\begin{aligned} x(t) &= x_0 + \frac{v_{0x}}{k} (1 - e^{-kt}) \\ y(t) &= y_0 + \frac{1}{k} \left(v_{0y} + \frac{g}{k} \right) (1 - e^{-kt}) - \frac{g}{k} t \\ v_x(t) &= v_{0x} e^{-kt} \\ v_y(t) &= \left(v_{0y} + \frac{g}{k} \right) e^{-kt} - \frac{g}{k}, \end{aligned} \quad (4.5)$$

από όπου διαβάζουμε την κίνηση του σωματιδίου με ορισκή ταχύτητα $v_y(+\infty) = -g/k$ ($x(+\infty) = \text{σταθ.}$, $y(+\infty) \sim t$).

Ο προγραμματισμός της επιτάχυνσης καταγράφεται στο αρχείο ($k1 \leftrightarrow g$, $k2 \leftrightarrow k$) `rk2_vg.f`:

```
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      The acceleration functions f3,f4(t,x1,x2,v1,v2) provided by the user
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Free fall in constant gravitational filed with
C      ax = -k2 vx    ay = -k2 vy - k1
      double precision function f3(t,x1,x2,v1,v2)
      implicit none
      double precision t,x1,x2,v1,v2
      double precision k1,k2
      common /couplings/k1,k2
      f3=-k2*v1    !dx3/dt=dv1/dt=a1
      end

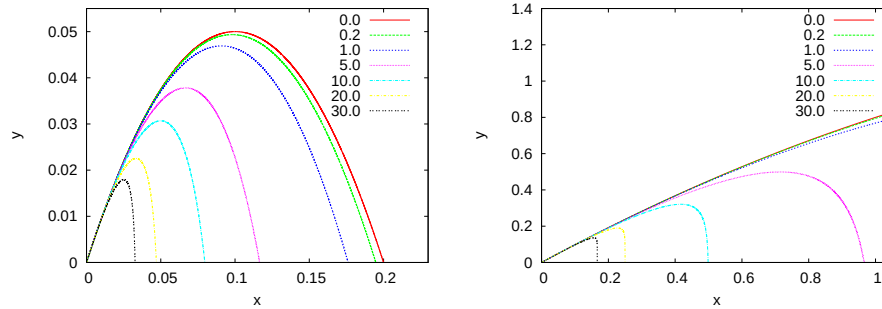
      double precision function f4(t,x1,x2,v1,v2)
      implicit none
      double precision t,x1,x2,v1,v2
      double precision k1,k2
      common /couplings/k1,k2
      f4=-k2*v2-k1    !dx4/dt=dv2/dt=a2
      end
```

Τα αποτελέσματα καταγράφονται στα Σχήματα 4.3 όπου φαίνεται η επίδραση της αυξανόμενης αντίστασης στην τροχιά του σωματιδίου. Στο Σχήμα 4.4 δίνεται για σύγκριση η επίδραση δύναμης $\vec{F}_r = -mkv^2\hat{v}$.

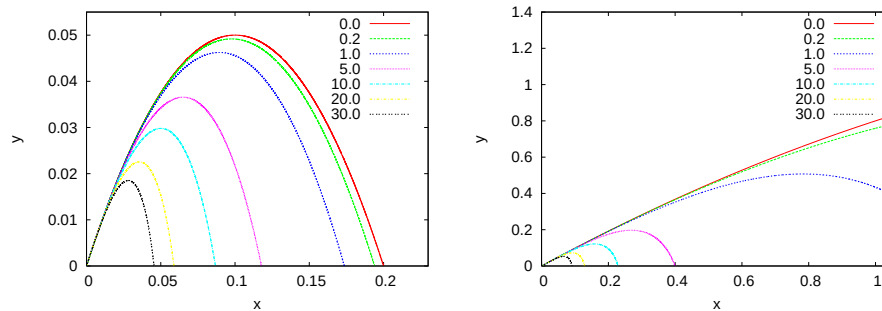
4.3 Κίνηση Πλανητών.

Θα θεωρήσουμε το απλό πλανητικό μοντέλο του 'ήλιου' με μάζα M και ενός πλανήτη 'γη' με μάζα m έτσι ώστε $m \ll M$. Ο νόμος του Νεύτωνα μας δίνει ότι η επιτάχυνση της 'γης' δίνεται από τη σχέση

$$\vec{a} = \vec{g} = -\frac{GM}{r^2}\hat{r} = -\frac{GM}{r^3}\vec{r} \quad (4.6)$$



Σχήμα 4.3: Τροχιά σωματιδίου που βάλλεται στο σταθερό βαρυτικό πεδίο της γής $\vec{g} = -10\hat{j}$ υπό την επίδραση αντίστασης ρευστού $a_r = -k\vec{v}$ για $k = 0, 0.2, 1, 5, 10, 20, 30$. Αριστερά έχουμε $\vec{v}(0) = \hat{i} + \hat{j}$ ενώ δεξιά $\vec{v}(0) = 5\hat{i} + 5\hat{j}$.



Σχήμα 4.4: Τροχιά σωματιδίου που βάλλεται στο σταθερό βαρυτικό πεδίο της γής $\vec{g} = -10\hat{j}$ υπό την επίδραση αντίστασης ρευστού $a_r = -kv^2\hat{v}$ για $k = 0, 0.2, 1, 5, 10, 20, 30$. Αριστερά έχουμε $\vec{v}(0) = \hat{i} + \hat{j}$ ενώ δεξιά $\vec{v}(0) = 5\hat{i} + 5\hat{j}$.

Θυμίζουμε στον αναγνώστη ότι $G = 6.67 \times 10^{-11} \frac{\text{m}^3}{\text{kg} \cdot \text{sec}^2}$, $M = 1.99 \times 10^{30} \text{kg}$, $m = 5.99 \times 10^{24} \text{kg}$. Επίσης όταν η υπόθεση $m \ll M$ δεν είναι ικανοποιητική, τότε το πρόβλημα των δύο σωμάτων ανάγεται σε αυτό του ενός χρησιμοποιώντας την ανηγμένη μάζα

$$\frac{1}{\mu} = \frac{1}{m} + \frac{1}{M}.$$

Η δύναμη της βαρύτητας είναι κεντρική με αποτέλεσμα να διατηρήται η στροφορμή $\vec{L} = \vec{r} \times \vec{p}$. Αυτό σημαίνει ότι η κίνηση γίνεται πάνω σε ένα επίπεδο και

μπορούμε να πάρουμε τον άξονα των z έτσι ώστε

$$\vec{L} = L_z \hat{k} = m(xv_y - yv_x). \quad (4.7)$$

Η δύναμη είναι διατηρητική και η ενέργεια

$$E = \frac{1}{2}mv^2 - \frac{GmM}{r} \quad (4.8)$$

διατηρήται. Αν πάρουμε την αρχή των αξόνων να είναι το κέντρο της δύναμης, τότε οι εξισώσεις κίνησης (4.6) γίνονται

$$\begin{aligned} a_x &= -\frac{GM}{r^3} x \\ a_y &= -\frac{GM}{r^3} y \end{aligned} \quad (4.9)$$

με $r^2 = x^2 + y^2$. Οι εξισώσεις αυτές είναι ένα σύστημα δύο συζευγμένων διαφορικών εξισώσεων ως προς τις συναρτήσεις $x(t)$, $y(t)$. Οι λύσεις είναι κωνικές τομές που είναι είτε έλλειψη (δεσμευμένη τροχιά - ‘πλανήτης’), παραβολή (για τη λεγόμενη ‘ταχύτητα διαφυγής’ ή υπερβολή (σκέδαση)).

Για την περίοδο περιστροφής των πλανητών ισχύει ο τρίτος νόμος του Κέπλερ

$$T^2 = \frac{4\pi^2}{GM} a^3 \quad (4.10)$$

όπου εδώ a είναι ο μεγάλος ημιάξονας της ελλειπτικής τροχιάς και b ο μικρός ημιάξονας. Το πόσο ‘πλατιά’ είναι η έλλειψη χαρακτηρίζεται από την εκκεντρότητα της τροχιάς

$$e = \sqrt{1 - \frac{b^2}{a^2}}, \quad (4.11)$$

η οποία είναι 0 για τον κύκλο και τείνει προς τη 1 όταν την ‘πατάμε’ να γίνει ευθεία. Σε απόσταση ea από το κέντρο της έλλειψης βρίσκονται οι εστίες της F_1 και F_2 . Αυτές έχουν την ιδιότητα ότι κάθε σημείο P της τροχιάς έχει

$$PF_1 + PF_2 = 2a. \quad (4.12)$$

Για να προγραμματίσουμε τη δύναμη του Νεύτωνα γράφουμε στο αρχείο `rk2_cb.f`:

```

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      The acceleration functions f3,f4(t,x1,x2,v1,v2) provided by the user
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Motion in Coulombic/Newtonian potential:
C      ax= k1*x1/r^3 ay= k1*x2/r^3
      double precision function f3(t,x1,x2,v1,v2)
      implicit none
      double precision t,x1,x2,v1,v2
      double precision k1,k2
      common /couplings/k1,k2
      double precision r2,r3
      r2=x1*x1+x2*x2
      r3=r2*dsqrt(r2)
      if(r3.gt.0.0D0)then
        f3=k1*x1/r3          !dx3/dt=dv1/dt=a1
      else
        f3=0.0D0
      endif
      end

      double precision function f4(t,x1,x2,v1,v2)
      implicit none
      double precision t,x1,x2,v1,v2
      double precision k1,k2
      common /couplings/k1,k2
      double precision r2,r3
      r2=x1*x1+x2*x2
      r3=r2*dsqrt(r2)
      if(r3.gt.0.0D0)then
        f4=k1*x2/r3          !dx4/dt=dv2/dt=a2
      else
        f4=0.0D0
      endif
      end

      double precision function energy(t,x1,x2,v1,v2)
      implicit none
      double precision t,x1,x2,v1,v2

```

```

double precision k1,k2
common /couplings/k1,k2
double precision r
r=dsqrt(x1*x1+x2*x2)
if( r .gt. 0.0D0)then
  energy = 0.5*(v1*v1+v2*v2) + k1/r
else
  energy = 0.0D0
endif
end

```

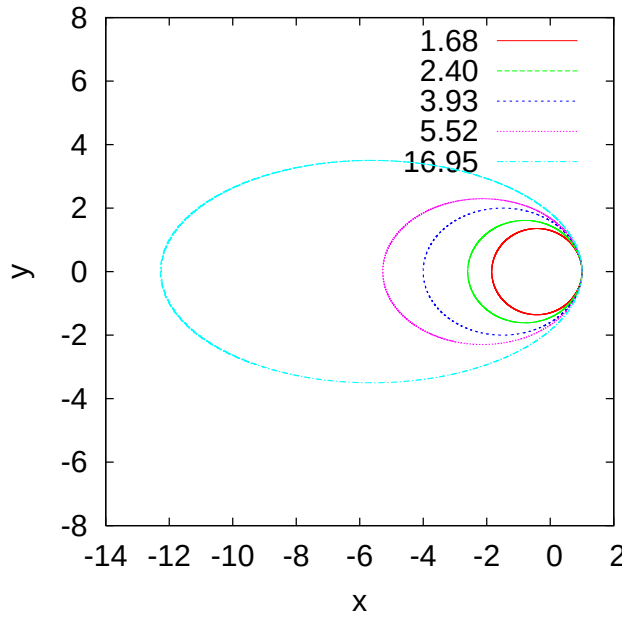
Στο παραπάνω πρόγραμμα $k1 = -GM$ και έχουμε προσέξει την περίπτωση το σωματίο να προσκρούσει στο ιδιάζον σημείο $(0,0)$, το κέντρο της δύναμης. Προφανώς ό ίδιος κώδικας μπορεί να χρησιμοποιηθεί και για το ηλεκτροστατικό πεδίο Coulomb με $k1 = qQ/4\pi\epsilon_0 m$.

Κατ' αρχήν μελετάμε τροχιές οι οποίες είναι δέσμιες. Διαλέγουμε $GM = 10$, $x(0) = 1.0$, $y(0) = 0$, $v_{0x} = 0$ και v_{0y} μεταβλητό. Μετράμε την περίοδο και το μήκος των ημιαξόνων της έλλειψης. Προκύπτει ο Πίνακας 4.1. Μερικές από

v_{0x}	$T/2$	$2a$
3.2	1.032	2.051
3.4	1.282	2.371
3.6	1.682	2.841
3.8	2.398	3.598
4.0	3.9288	5.002
4.1	5.5164	6.272
4.2	8.6952	8.481
4.3	16.95	13.256
4.35	28.168	18.6
4.38	42.81	24.58
4.40	61.8	31.393
4.42	99.91	43.252

Πίνακας 4.1: Τα αποτελέσματα για την περίοδο και τον μεγάλο ημιάξονα της ελλειπτικής τροχιάς πλανητικής κίνησης για $GM = 10$, $x(0) = 1.0$, $y(0) = 0$, $v_{0y} = 0$.

τις τροχιές φαίνονται στο σχήμα 4.5 όπου φαίνεται η εξάρτηση του μεγέθους της έλλειψης από την περίοδο. Στο Σχήμα 4.6 επιβεβαιώνουμε τον 3ο νόμο του Κέπλερ, Σχέση (4.10) .



Σχήμα 4.5: Τροχιές πλανήτη για $GM = 10$, $x(0) = 1.0$, $y(0) = 0$, $v_{0y} = 0$ και $v_{0x} = 3.6$, 3.8, 4.0, 4.1, 4.3. Αναγράφονται οι αντίστοιχες ημιπερίοδοι.

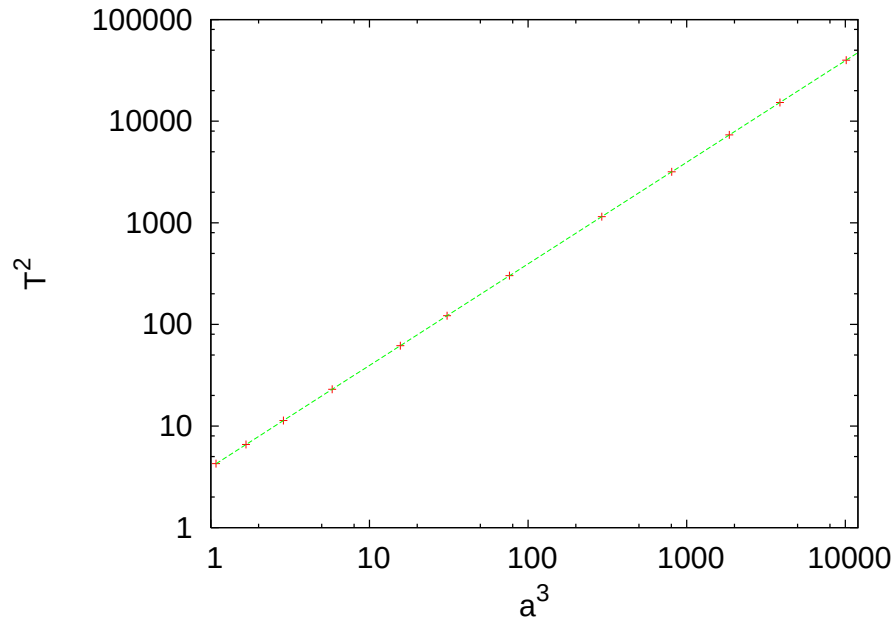
Πώς θα μπορούσαμε να προβλέψουμε το νόμο του Κέπλερ χωρίς να γνωρίζαμε το αποτέλεσμα εκ των προτέρων; Αν πάρουμε το λογάριθμο και στα δύο μέλη της Εξίσωσης (4.10) προκύπτει:

$$\ln T = \frac{3}{2} \ln a + \frac{1}{2} \ln \left(\frac{4\pi^2}{GM} \right) \quad (4.13)$$

Άρα σε ένα διάγραμμα των σημείων $(\ln a, \ln T)$ τα σημεία πρέπει να βρίσκονται πάνω σε μια ευθεία. Με τη μέθοδο των ελαχίστων τετραγώνων μπορούμε να υπολογίσουμε το συντελεστή κατεύθυνσης και το σημείο τομής των αξόνων που θα πρέπει να είναι $\frac{3}{2}$ και $1/2 \ln(4\pi^2/GM)$ αντίστοιχα. Το αφήνουμε σαν άσκηση για τον αναγνώστη.

Σε περίπτωση που η αρχική ταχύτητα του σωματιδίου υπερβεί την ταχύτητα διαφυγής v_e το σωματίδιο ξεφεύγει από την επίδραση του πεδίου βαρύτητας. Αυτό γίνεται όταν η μηχανική του ενέργεια (4.8) είναι 0 ή όταν

$$v_e^2 = \frac{2GM}{r}, \quad (4.14)$$



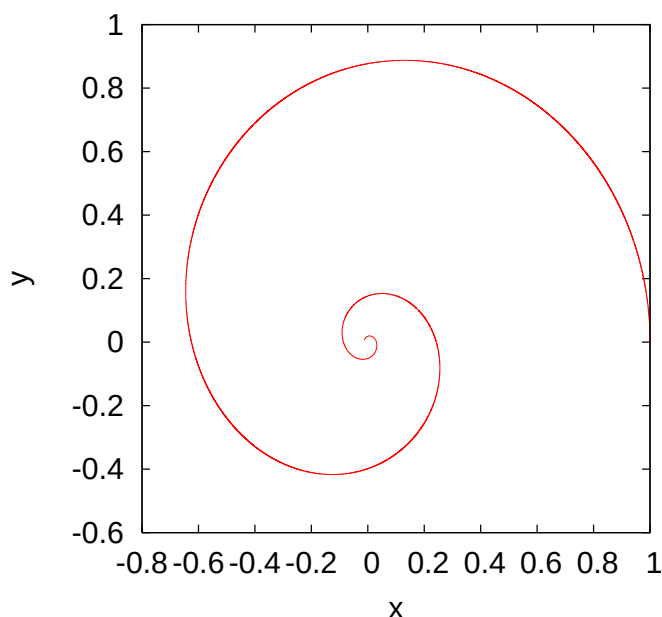
Σχήμα 4.6: Ο τρίτος νόμος του Κέπλερ για $GM = 10$. Τα σημεία είναι οι μετρήσεις από τον Πίνακα 4.1 και η συνεχής γραμμή η αναλυτική λύση (4.10) .

που στην περίπτωση που εξετάζουμε με $GM = 10$ παίρνουμε $v_e \approx 4.4721 \dots$. Δοκιμάστε με πόση ακρίβεια μπορείτε να την προσδιορίσετε αριθμητικά παίρνοντας την αρχική ταχύτητα v_{0x} να πλησιάζει την παραπάνω τιμή από μεγαλύτερες και μικρότερες τιμές και βλέποντας τότε περνάτε από κλειστή σε ανοιχτή τροχιά. Επαναλάβετε τη διαδικασία ενκυβοτίζοντας τα διαστήματα της αρχικής ταχύτητας για τα οποία η ενέργεια αλλάζει πρόσημο.

4.4 Σκέδαση.

Στην παράγραφο αυτή θεωρούμε σκέδαση σωματιδίων από ένα κεντρικό δυναμικό ². Υποθέτουμε ότι το δυναμικό αυτό έχει τροχιές που ξεκινούν από το άπειρο και καταλήγουν στο άπειρο, στο οποίο τα σωματίδια κινούνται ελεύθερα από την επίδραση της δύναμης. Έτσι αρχικά τα σωματίδια κινούνται ελεύθερα προς την περιοχή αλληλεπίδρασης με το κέντρο της δύναμης μέσα στην οποία

²Διαβάστε το κεφάλαιο 4 του [;]



Σχήμα 4.7: Σπειροειδής τροχιά σωματιδίου που κινείται υπό την επίδραση κεντρική δύναμης $\vec{F} = -k/r^3 \hat{r}$.

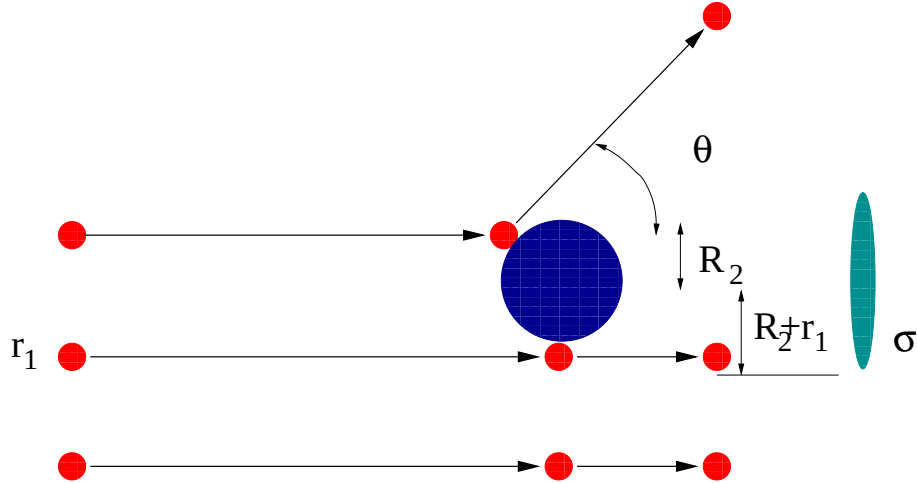
αλλάζουν κατεύθυνση και κινούνται πάλι έξω από αυτή σε διαφορετική διεύθυνση. Λέμε τότε ότι το σωματίο σκεδάστηκε και ότι η γωνία μεταξύ της αρχικής και τελικής διεύθυνσης της ταχύτητας είναι η γωνία σκέδασης θ . Το ενδιαφέρον στην περίπτωση αυτή έγκειται στο γεγονός ότι από την κατανομή της γωνίας σκέδασης μιας δέσμης σωματιδίων μπορούμε να πάρουμε χρήσιμη πληροφορία για το δυναμικό σκέδασης. Αυτή η τεχνική χρησιμοποιείται κατά κόρο στους σημερινούς επιταχυντές για την μελέτη των θεμελιωδών αλληλεπιδράσεων των στοιχειωδών σωματιδίων. Ο ακριβής προσδιορισμός της μορφής του δυναμικού σκέδασης με τον τρόπο αυτό αποτελεί μέρος της θεωρίας αντίστροφης σκέδασης..

Για να κατανοήσουμε τους ορισμούς είναι χρήσιμο να θεωρήσουμε τη σκέδαση μικρών σκληρών σφαιρών ακτίνας r_1 από άλλες σκληρές σφαίρες ακτίνας

R_2 . Το δυναμικό αλληλεπίδρασης³ είναι δηλαδή:

$$V(r) = \begin{cases} 0 & r > R_2 + r_1 \\ \infty & r < R_2 + r_1 \end{cases} \quad (4.15)$$

Υποθέτουμε ότι τα σωματίδια της δέσμης δεν αλληλεπιδρούν μεταξύ τους και



Σχήμα 4.8: Σκέδαση σκληρών σφαιρών. θ είναι η γωνία σκέδασης. Δεξιά φαίνεται η συνολική ενεργός διατομή σ .

ότι κατά τη σκέδαση κάθε σωματίο αλληλεπιδρά μόνο με ένα κέντρο σκέδασης του στόχου. Έστω J η πυκνότητα ροής ή ένταση της δέσμης⁴ και A η διατομή της δέσμης. Έστω ότι ο στόχος έχει n σωματίδια ανά μονάδα επιφάνειας. Η ενεργός διατομή της αλληλεπίδρασης είναι $\sigma = \pi(r_1 + R_2)^2$ όπου r_1 και R_2 οι ακτίνες των σκεδαζομένων σφαιρών και των στόχων αντίστοιχα (βλ. Σχήμα (4.8)): όλες οι σφαίρες έξω από την επιφάνεια αυτή στη δέσμη δεν σκεδάζονται από το συγκεκριμένο στόχο. Η συνολική ενεργός διατομή που παρουσιάζουν όλα τα κέντρα αλληλεπίδρασης του στόχου είναι

$$\Sigma = nA\sigma, \quad (4.16)$$

όπου nA είναι ο συνολικός αριθμός των κέντρων του στόχου που βρίσκονται μέσα στην δέσμη. Κατά μέσο όρο ο ρυθμός σκέδασης, δηλ. ο αριθμός των

³Λέγεται δυναμικό σκληρού πυρήνα (hard core potential).

⁴Ο αριθμός των σωματιδίων που περνούν μια επιφάνεια κάθετη στη δέσμη ανά μονάδα χρόνου και μονάδα επιφάνειας.

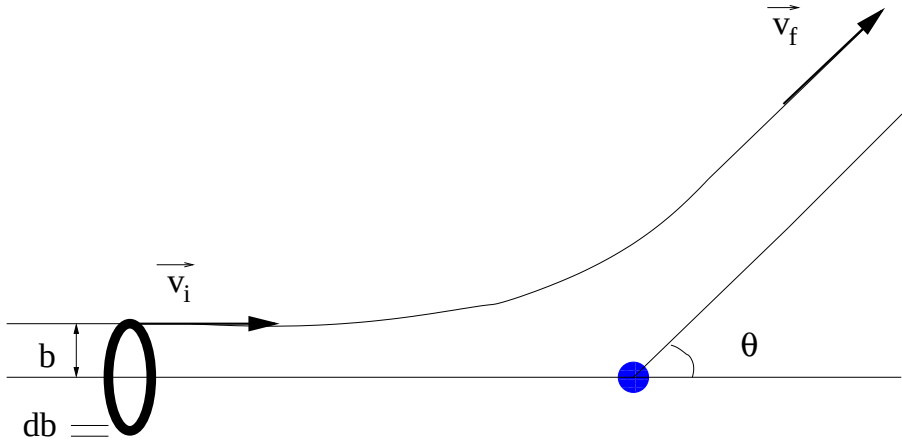
σκεδάσεων ανά μονάδα χρόνου θα είναι

$$N = J\Sigma = JnA\sigma. \quad (4.17)$$

Η παραπάνω εξίσωση αποτελεί και τον ορισμό της συνολικής ενεργούς διατομής της αλληλεπίδρασης για οποιαδήποτε άλλη περίπτωση σκέδασης που πληρεί τις βασικές υποθέσεις που κάναμε, οι οποίες μας οδηγούν ότι αυτή είναι η ποσότητα που εξαρτάται από το είδος της αλληλεπίδρασης. Η διαφορική ενεργός διατομή ορίζεται από τη σχέση

$$dN = JnA\sigma(\theta) d\Omega, \quad (4.18)$$

όπου dN ο αριθμός των σωματιδίων ανά μονάδα χρόνου που σκεδάζονται μέσα στη στερεά γωνία $d\Omega$. Η συνολική ενεργός διατομή είναι



Σχήμα 4.9: Σωματίδια της δέσμης που περνούν μέσα από το δακτύλιο $2\pi b db$ σκεδάζονται μέσα στη στερεά γωνία $d\Omega = 2\pi \sin\theta d\theta$.

$$\sigma_{tot} = \int_{\Omega} \sigma(\theta) d\Omega = \int \sigma(\theta) \sin\theta d\theta d\phi = 2\pi \int \sigma(\theta) \sin\theta d\theta. \quad (4.19)$$

Στην τελευταία σχέση χρησιμοποιήσαμε την κυλινδρική συμμετρία της αλληλεπίδρασης ως προς τον άξονα της κρούσης. Καταλήγουμε στη σχέση

$$\sigma(\theta) = \frac{1}{nAJ} \frac{dN}{2\pi \sin\theta d\theta}. \quad (4.20)$$

Αυτή η σχέση μπορεί να χρησιμοποιηθεί πειραματικά για τη μέτρηση της διαφορικής ενεργούς διατομής μετρώντας το ρυθμό ανίχνευσης σωματιδίων μέσα

σε δύο κώνους που ορίζονται από τις γωνίες θ και $\theta + d\theta$. Τη σχέση αυτή θα χρησιμοποιήσουμε και στον αριθμητικό υπολογισμό της $\sigma(\theta)$.

Για να προσδιορίσουμε τη διαφορική ενεργό διατομή από μια θεωρία, μπορούμε να ακολουθήσουμε την εξής γενική διαδικασία. Έστω ότι σωματίο βάλλεται προς το στόχο όπως φαίνεται στο Σχήμα 4.9. b ονομάζεται η παράμετρος κρούσης και η τελική γωνία θ εξαρτάται από αυτή. Άρα το μέρος της δέσμης που σχεδιάζεται σε γωνίες μεταξύ θ και $\theta + d\theta$ βρίσκεται σε ένα κυκλικό δαχτυλίδι ακτίνας $b(\theta)$, πάχους db και εμβαδού $2\pi b db$. Αφού έχουμε ένα σωματίο στο στόχο $nA = 1$. Ο αριθμός των σωματιδίων ανα μονάδα χρόνου που περνούν μέσα από το δαχτυλίδι είναι $J2\pi b db$, άρα

$$2\pi b(\theta) db = -2\pi \sigma(\theta) \sin \theta d\theta \quad (4.21)$$

(το $-$ οφείλεται στο γεγονός ότι όταν το b αυξάνει το θ μικραίνει). Από τη γνώση του δυναμικού μπορούμε να υπολογίσουμε το $b(\theta)$ οπότε προκύπτει η $\sigma(\theta)$. Αντίστροφα, αν μετρήσουμε τη $\sigma(\theta)$, μπορούμε να προσδιορίσουμε την $b(\theta)$.

4.4.1 Σκέδαση Rutherford.

Η σκέδαση φορτισμένου σωματιδίου φορτίου q ('ηλεκτρονίου') μέσα σε δυναμικό Coulomb πολύ βαρύτερου σημειακού ηλεκτρικού φορτίου Q ('πυρήνας') ονομάζεται σκέδαση Rutherford. Στην περίπτωση αυτή το δυναμικό αλληλεπίδρασης είναι

$$V(r) = \frac{1}{4\pi\epsilon_0} \frac{Q}{r}, \quad (4.22)$$

το οποίο προσδίδει επιτάχυνση $\vec{a}$ στο σωματίδιο ίση με

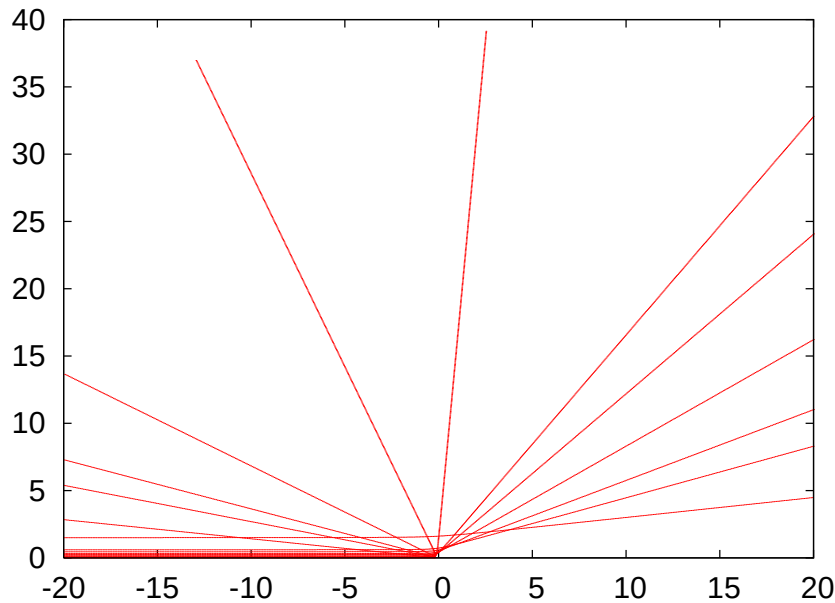
$$\vec{a} = \frac{qQ}{4\pi\epsilon_0 m} \frac{\hat{r}}{r^2} \equiv \alpha \frac{\vec{r}}{r^3}. \quad (4.23)$$

Η ενέργεια του σωματιδίου είναι $E = \frac{1}{2}mv^2$ και η στροφορμή του $l = mvb$ όπου εδώ $v \equiv |\vec{v}_i|$. Η σχέση μεταξύ της παραμέτρου κρούσης και της γωνίας σκέδασης βρίσκεται να είναι

$$b(\theta) = \frac{\alpha}{v^2} \cot \frac{\theta}{2}, \quad (4.24)$$

όπου σε συνδυασμό με την (4.21) προκύπτει ότι

$$\sigma(\theta) = \frac{\alpha^2}{4} \frac{1}{v^4} \sin^{-4} \frac{\theta}{2}. \quad (4.25)$$



Σχήμα 4.10: Τροχιές σκέδασης Rutherford. Θέσαμε $\kappa_1 \equiv \frac{qQ}{4\pi\epsilon_0 m} = 1$ στο αρχείο `rk2_cb.f` και μελετήσαμε τις τροχιές για $b = 0.08, 0.015, 0.020, 0.035, 0.080, 0.120, 0.200, 0.240, 0.320, 0.450, 0.600, 1.500$. Το σωματίο τοποθετήθηκε αρχικά στη θέση $x(0) = -50$ και του δόθηκε αρχική ταχύτητα $v = 3$. Ο αριθμός των βημάτων στην ολοκλήρωση είναι 1000 για χρόνο από 0 έως 30.

Αρχικά εξετάζουμε τις τροχιές σκέδασης. Τα αποτελέσματα φαίνονται ποιοτικά στο Σχήμα 4.10 στην περίπτωση που τα φορτισμένα σωματίια έχουν ομώνυμα φορτία. Ανάλογο σχήμα προκύπτει και για ετερόνυμα φορτία, προσοχή πρέπει να δοθεί στην ακρίβεια της μεθόδου για μικρές παραμέτρους χρούσης $b < 0.2$ (και τις υπόλοιπες παραμέτρους όπως στο Σχήμα 4.10) όπου η γωνία σκέδασης γίνεται ≈ 1 . Πολύ μεγαλύτερος αριθμός βημάτων απαιτείται για την επίτευξη ικανοποιητικής ακρίβειας. Βρίσκουμε ότι η ποσότητα που είναι δείκτης που δείχνει την σύγκλιση των αριθμητικών αποτελεσμάτων με αυτά της Σχέσης (4.24) είναι η ενέργεια, η οποία πρέπει να διατηρήται κατά την χρούση. Αυτό θα μας χρησιμεύσει όταν θα μελετήσουμε δυναμικά για τα οποία δεν έχουμε αναλυτική λύση.

Για να μελετήσουμε ποσοτικά τα αποτελέσματά μας αυξάνουμε την ακρίβεια έτσι ώστε να πετύχουμε ικανοποιητική σύγκλιση των αναλυτικών και αριθμητικών αποτελεσμάτων. Καταρτίζουμε έτσι τον Πίνακα

b	θ_n	θ_a	$\Delta E/E$	STEPS
0.008	2.9982	2.9978	$1.06 \cdot 10^{-5}$	5000
0.020	2.7861	2.7854	$6.25 \cdot 10^{-5}$	5000
0.030	2.6152	2.6142	$1.29 \cdot 10^{-4}$	5000
0.043	2.4043	2.4031	$2.31 \cdot 10^{-4}$	5000
0.056	2.2092	2.2079	$3.27 \cdot 10^{-4}$	5000
0.070	2.0184	2.0172	$4.07 \cdot 10^{-4}$	5000
0.089	1.7918	1.7909	$4.70 \cdot 10^{-4}$	5000
0.110	1.5814	1.5808	$4.82 \cdot 10^{-4}$	5000
0.130	1.4147	1.4144	$4.59 \cdot 10^{-4}$	5000
0.160	1.2138	1.2140	$3.97 \cdot 10^{-4}$	5000
0.200	1.0137	1.0142	$3.08 \cdot 10^{-4}$	5000
0.260	0.8070	0.8077	$2.04 \cdot 10^{-4}$	5000
0.360	0.5979	0.5987	$1.07 \cdot 10^{-4}$	5000
0.560	0.3910	0.3917	$3.83 \cdot 10^{-5}$	5000
1.160	0.1905	0.1910	$5.58 \cdot 10^{-6}$	5000

Πίνακας 4.2: Γωνίες σκέδασης στη σκέδαση Rutherford. Θέσαμε $\kappa_1 \equiv \frac{qQ}{4\pi\epsilon_0 m} = 1$ στο αρχείο `rk2_cb.f` και μελετήσαμε τις τροχιές για τις τιμές του b που φαίνονται στη στήλη 1. θ_n είναι η γωνία σκέδασης που υπολογίζεται αριθμητικά και θ_a το αποτέλεσμα της Σχέσης (4.24). $\Delta E/E$ είναι η ποσοστιαία μεταβολή της ενέργειας λόγω συστηματικών σφαλμάτων της μεθόδου και στην τελευταία στήλη ο αριθμός των βημάτων ολοκλήρωσης για χρόνο από 0 έως 30. Το σωματίο τοποθετήθηκε αρχικά στη θέση $x(0) = -50$ και του δόθηκε αρχική ταχύτητα $v = 3$.

Θα περιγράψουμε τώρα ένα τρόπο για τον υπολογισμό της ενεργούς διατομής χρησιμοποιώντας τη Σχέση (4.20). Εναλλακτικά θα μπορούσε να χρησιμοποιηθεί η (4.21) υπολογίζοντας την κατάλληλη παράγωγο αριθμητικά. Αυτό όμως το αφήνουμε για άσκηση στον ανήσυχο και επιμελή αναγνώστη. Ο υπολογισμός που θα κάνουμε μοιάζει να είναι ‘πειραματικός’. Τοποθετούμε ‘ανιχνευτή’ που ‘ανιχνεύει’ τα σωματίδια που σκεδάζονται από θ μέχρι $\theta + \delta\theta$. Για το λόγο αυτό χωρίζουμε το διάστημα $[0, \pi]$ σε N_b βινς έτσι ώστε $\delta\theta = \pi/N_b$. Κάνουμε ‘πειράματα’ σκέδασης μεταβάλλοντας την παράμετρο σκέδασης $b \in [b_m, b_M]$ με βήμα δb . Λόγω της συμμετρίας το προβλήματος, κρατάμε το ϕ σταθερό, οπότε δεδομένο θ αντιστοιχεί σε κώνο ανοίγματος θ και κορυφή το κέντρο σκέδασης. Παρατηρούμε σε ποια γωνία θ σκεδάζεται το σωματίο με το συγκεκριμένο b και καταχωρούμε τον αριθμό των σωματιδίων ανά μονάδα χρόνου $\delta N \propto b\delta b$, μια και αυτός είναι ανάλογος του εμβαδού του δακτυλιδιού ακτίνας b . Απομένει

b	θ_n	θ_a	$\Delta E/E$	STEPS
0.020	0.577	2.785	0.47	150000
0.030	2.466	2.614	0.22	150000
0.043	2.333	2.403	$8.62 \cdot 10^{-2}$	150000
0.056	2.180	2.208	$2.88 \cdot 10^{-2}$	150000
0.070	2.006	2.017	$1.07 \cdot 10^{-2}$	150000
0.089	1.779	1.791	$8.94 \cdot 10^{-3}$	60000
0.110	1.570	1.581	$7.07 \cdot 10^{-3}$	30000
0.130	1.406	1.414	$5.25 \cdot 10^{-3}$	20000
0.160	1.198	1.214	$8.63 \cdot 10^{-3}$	5000
0.200	1.007	1.014	$3.71 \cdot 10^{-3}$	5000
0.260	0.8057	0.8077	$1.40 \cdot 10^{-3}$	5000
0.360	0.5988	0.5987	$4.32 \cdot 10^{-4}$	5000
0.560	0.3923	0.3917	$9.40 \cdot 10^{-5}$	5000
1.160	0.1913	0.1910	$8.61 \cdot 10^{-6}$	5000

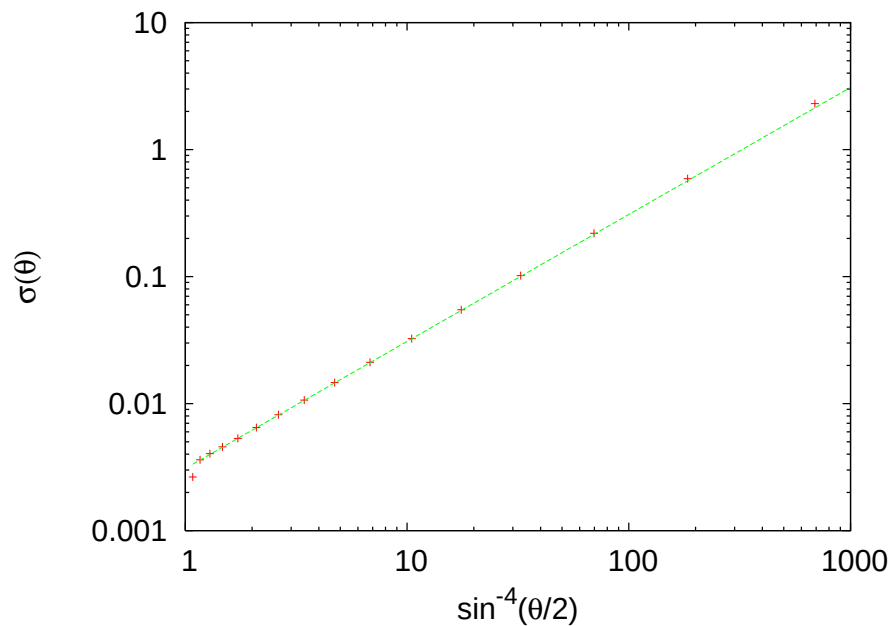
Πίνακας 4.3: Αποτελέσματα όμοια με αυτά του Πίνακα 4.2. Η μόνη διαφορά είναι ότι η σκέδαση είναι μεταξύ ετερόνυμων φορτίων με $\frac{qQ}{4\pi\epsilon_0 m} = -1$. Φαίνεται η δυσκολία που αντιμετωπίζει η μέθοδος για μικρές παράμετρους κρούσης.

να υπολογίσουμε τη ροή J η οποία είναι ο συνολικός αριθμός των σωματιδίων ανά μονάδα χρόνου που δεν είναι άλλος από $J \propto \sum_i b \delta b$ (στο λόγο $\delta N/J$ η σταθερά αναλογίας και το δb απλοποιούνται) και τη στερεά γωνία $2\pi \sin(\theta) \delta\theta$. Τέλος, εύκολα χρησιμοποιείται η σχέση (4.19) για τον υπολογισμό της συνολικής ενεργούς διατομής σ_{tot} . Ο προγραμματισμός της διαδικασίας γίνεται μεταβάλλοντας απλά το κυρίως πρόγραμμα του `rk2.f` το οποίο καταγράφουμε στο αρχείο `scatter.f`.

```

C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Program that computes scattering cross-section of a central
C      force on the plane. The user should first check that the
C      parameters used, lead to a free state in the end.
C      ** X20 is the impact parameter b **
C      A 4 ODE system is solved using Runge-Kutta Method
C      User must supply derivatives
C      dx1/dt=f1(t,x1,x2,x3,x4) dx2/dt=f2(t,x1,x2,x3,x4)
C      dx3/dt=f3(t,x1,x2,x3,x4) dx4/dt=f4(t,x1,x2,x3,x4)

```

Σχήμα 4.12: Διαφορική ενεργός διατομή για τη σκέδαση Rutherford όπως στο Σχήμα 4.11. Η συνεχής ευθεία γραμμή είναι η $1/(4 \times 3^4)x$ από όπου είναι εμφανής η συναρτησιακή μορφή της $\sigma(\theta)$.

```

integer Nbins,index
parameter (Nbins = 20 )
double precision PI,rad2deg,angle,dangle,bins(Nbins),Npart
parameter (PI      =3.14159265358979324D0)
parameter (rad2deg=180.0D0/PI)
parameter (dangle =PI/Nbins)
double precision R,density,dOmega,sigma,sigmatot
C
Input:
print *, 'Runge-Kutta Method for 4-ODEs Integration'
print *, 'Enter coupling constants:'
read(5,*) k1,k2
print *, 'k1= ',k1,' k2= ',k2
print *, 'Enter STEPS,T0,TF,X10,X20,V10,V20:'
read(5,*) STEPS,T0,TF,X10,X20,V10,V20
print *, 'Enter final impact parameter X20F and step dX20:'

```

```

read(5,*) X20F,dX20
print *,'No. Steps= ',STEPS
print *,'Time: Initial T0 =',T0, ' Final TF=',TF
print *,'
           X1(T0)=' ,X10, ' X2(T0)=' ,X20
print *,'
           V1(T0)=' ,V10, ' V2(T0)=' ,V20
print *,'Impact par X20F  =' ,X20F,' dX20  =' ,dX20

open(unit=11,file='scatter.dat')
do i=1,Nbins
  bins(i) = 0.0d0
enddo
C   The Calculation:
PSIZE  = P
Npart  = 0.0D0
X20    = X20 + dX20/2.0D0 !starts in middle of first interval
do while (X20 .lt. X20F )
  call RK(T,X1,X2,V1,V2,T0,TF,X10,X20,V10,V20,STEPS,PSIZE)

C   Take absolute value due to symmetry:
  angle = DABS(atan2(V2(STEPS),V1(STEPS)))
C   Output: The final angle. Check if almost constant
  write(11,*) '@ ', X20, angle,
$      DABS(atan2(V2(STEPS-50),V1(STEPS-50)))
$      ,k1/V10**2/tan(angle/2.0D0)

C   Update histogram:
  index      = int(angle/dangle)+1
C   Number of incoming particles per unit time
C   is proportional to radius of ring
C   of radius X20, the impact parameter:
  bins(index) = bins(index) + X20 !db is cancelled from density
  Npart       = Npart       + X20 !<-- i.e. from here
  X20         = X20         + dX20
enddo

C   Print scattering cross section:
R          = X20          !beam radius
density    = Npart/(PI*R*R) !beam flux density J

```

```

    sigmatot = 0.0D0          !total cross section
    do i=1,Nbins
        angle = (i-0.5D0)*dangle
        dOmega = 2.0D0*PI*dsin(angle)*dangle !d(Solid Angle)
        sigma = bins(i)/(density*dOmega)
        if(sigma.gt.0.0D0) write(11,*) 'ds= ',angle,angle*rad2deg,sigma
        sigmatot = sigmatot + sigma*dOmega
    enddo
    write(11,*) 'sigmatot= ',sigmatot
    close(11)

end

```

Η μεταγλώττιση γίνεται όπως και με την περίπτωση το `rk2.f` ενώ τα αποτελέσματα βρίσκονται στο αρχείο `scatter.dat`. Έτσι για να παράγουμε τα αποτελέσματα των Σχημάτων 4.11 και 4.12 εκτελούμε τις εντολές:

```

> f77 scatter.f rk2_cb.f -o scatter
> scatter
Runge-Kutta Method for 4-ODEs Integration
Enter coupling constants:
1.0 0.0
k1= 1.0000000000000000 k2= 0.0000000000000000
Enter STEPS,T0,TF,X10,X20,V10,V20:
5000 0 30 -50 0.02 3 0
Enter final impact parameter X20F and step dX20:
1 0.0002
No. Steps= 5000
Time: Initial T0 = 0.0000000000000000 Final TF= 30.00000000000000
      X1(T0)= -50.0000000000000000 X2(T0)= 2.0000000000000000E-002
      V1(T0)= 3.0000000000000000 V2(T0)= 0.0000000000000000
Impact par X20F = 1.0000000000000000 dX20 = 2.0000000000000000E-004

```

και ακολούθως βλέπουμε τα αποτελέσματα με το `gnuplot`:

```

> gnuplot
gnuplot> set log
gnuplot> plot [:1000] "<grep ds= scatter.dat" u ((sin($2/2))**(-4)):(($4) notit,\

```

```
(1./(4.*3.**4))*x notit
gnuplot> unset log
gnuplot> set log y
gnuplot> plot [:] "<grep ds= scatter.dat" u 2:4 notit, \
(1./(4.*3.**4))*(sin(x/2))**(-4) notit
```

Τα αποτελέσματα που παίρνουμε είναι σε πολύ καλή συμφωνία με τα αναμενόμενα από την αναλυτική έκφραση (4.25), οπότε έχουμε αρκετή αυτοπεποίθηση να μελετήσουμε διαφορετικά δυναμικά στις επόμενες παραγράφους και στις ασκήσεις.

4.4.2 Σκέδαση σε Άλλα Δυναμικά Πεδία.

Ας εξετάσουμε πρώτα τη σκέδαση από μία δύναμη της μορφής

$$\vec{F} = f(r) \hat{r}, \quad f(r) = \begin{cases} \frac{1}{r^2} - \frac{r}{a^3} & r \leq a \\ 0 & r > a \end{cases}, \quad (4.26)$$

η οποία είναι ένα απλό μοντέλο της σκέδασης ποζιτρονίου e^+ (θετικό φορτίο $+e$) με άτομο υδρογόνου που αποτελείται από θετικά φορτισμένο πυρήνα (θετικό φορτίο $+e$) που περιβάλλεται από νέφος ηλεκτρονίου αντίθετου φορτίου. Φυσικά έχουμε θέσει τις κλίμακες έτσι ώστε $m_{e^+} = 1$ και $e^2/4\pi\epsilon_0 = 1$. Στην περίπτωση αυτή δεν έχουμε αναλυτική λύση οπότε θα χρησιμοποιήσουμε αριθμητικές μεθόδους για τον υπολογισμό των συναρτήσεων $b(\theta)$, $\sigma(\theta)$ καθώς και της συνολικής ενεργού διατομής σ_{tot} .

Η δυναμική ενέργεια δίνεται από τη σχέση:

$$f(r) = -\frac{dV(r)}{dr} \Rightarrow V(r) = \frac{1}{r} + \frac{r^2}{2a^2} - \frac{3}{2a}. \quad (4.27)$$

όπου επιλέξαμε $V(r) = 0$ για $r \geq a$. Ο προγραμματισμός της δύναμης γίνεται εύκολα στο αρχείο `rk_hy.f`:

```
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      The acceleration functions f3,f4(t,x1,x2,v1,v2) provided by the user
C      CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Motion in hydrogen atom + positron:
C      f(r) = 1/r^2-r/k1^3
C      ax= f(r)*x1/r ay= f(r)*x2/r
```

```

double precision function f3(t,x1,x2,v1,v2)
implicit none
double precision t,x1,x2,v1,v2
double precision k1,k2
common /couplings/k1,k2
double precision r2,r,fr
r2=x1*x1+x2*x2
r =dsqrt(r2)
if(r .le.k1 .and. r2.gt.0.0D0)then
  fr = 1/r2-r/k1**3
else
  fr = 0.0D0
endif

if(fr.gt.0.0D0 .and. r .gt.0.0D0)then
  f3=fr*x1/r          !dx3/dt=dv1/dt=a1
else
  f3=0.0D0
endif
end

```

```

double precision function f4(t,x1,x2,v1,v2)
implicit none
double precision t,x1,x2,v1,v2
double precision k1,k2
common /couplings/k1,k2
double precision r2,r,fr
r2=x1*x1+x2*x2
r =dsqrt(r2)
if(r .le.k1 .and. r2.gt.0.0D0)then
  fr = 1/r2-r/k1**3
else
  fr = 0.0D0
endif

if(fr.gt.0.0D0 .and. r .gt.0.0D0)then
  f4=fr*x2/r          !dx3/dt=dv1/dt=a1
else

```

```

f4=0.0D0
endif
end

double precision function energy(t,x1,x2,v1,v2)
implicit none
double precision t,x1,x2,v1,v2
double precision k1,k2
common /couplings/k1,k2
double precision r,Vr
r=dsqrt(x1*x1+x2*x2)
if( r .le.k1 .and. r .gt.0.0D0)then
  Vr = 1/r + 0.5D0*r*r/k1**3 - 1.5D0 / k1
else
  Vr = 0.0D0
endif
energy = 0.5D0*(v1*v1+v2*v2) + Vr
end

```

Τα αποτελέσματα δίνονται στα Σχήματα 4.13–4.14. Βρίσκουμε ότι $\sigma_{tot} = \pi a^2$ (Άσκηση 4.9).

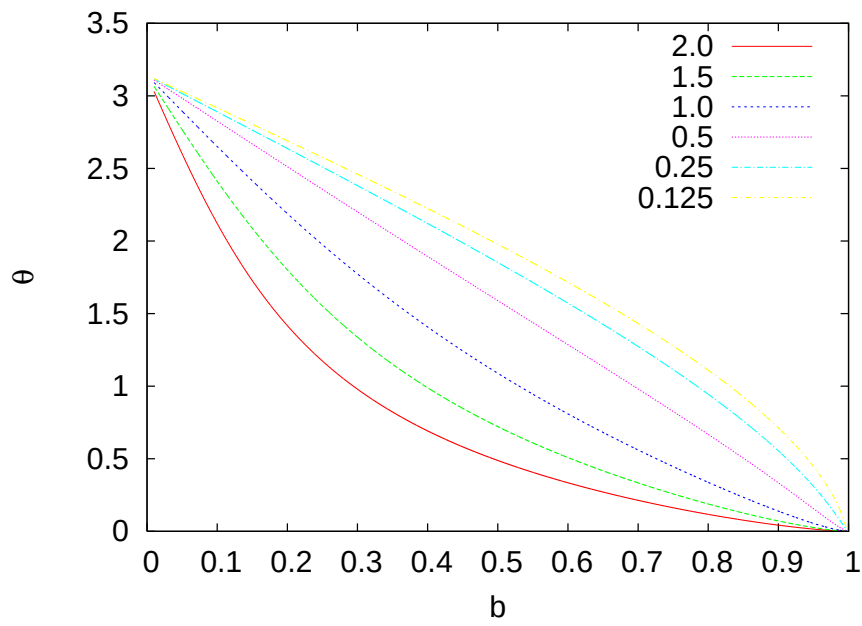
Ένα άλλο δυναμικό που παρουσιάζει ενδιαφέρον είναι το δυναμικό Yukawa ως φαινομενολογικό μοντέλο πυρηνικών αλληλεπιδράσεων:

$$V(r) = k \frac{e^{-r/a}}{r}. \quad (4.28)$$

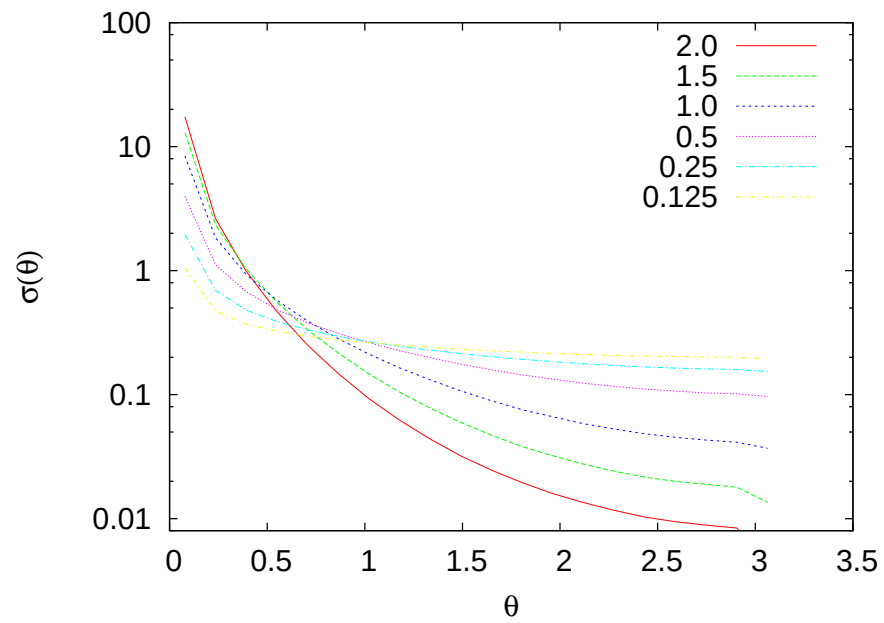
Το πεδίο αυτό μπορεί να χρησιμοποιηθεί και ως μοντέλο ενεργούς αλληλεπιδράσεως των ηλεκτρονίων στα μέταλλα (Thomas–Fermi) ή το δυναμικό Debye στο κλασικό πλάσμα. Η δύναμη που ασκείται σε ένα σώμα υπό την επίδραση του δυναμικού αυτού είναι:

$$\vec{F}(r) = f(r) \hat{r}, \quad f(r) = k \frac{e^{-r/a}}{r^2} \left(1 + \frac{r}{a} \right) \quad (4.29)$$

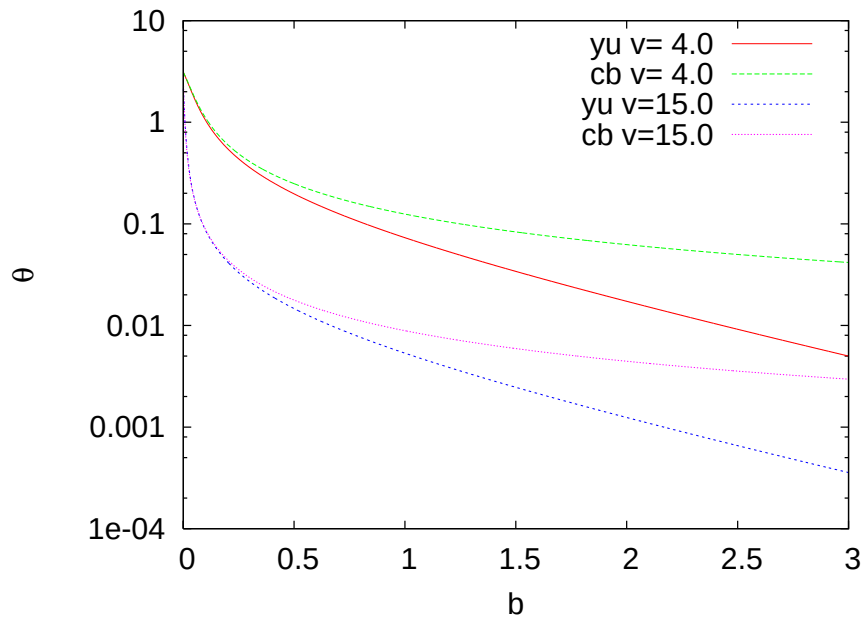
Ο προγραμματισμός της δύναμης γίνεται στο αρχείο `rk2_yu.f` κατά απόλυτη αναλογία με την προηγούμενη περίπτωση. Τα αποτελέσματα φαίνονται στα Σχήματα 4.15–4.16.



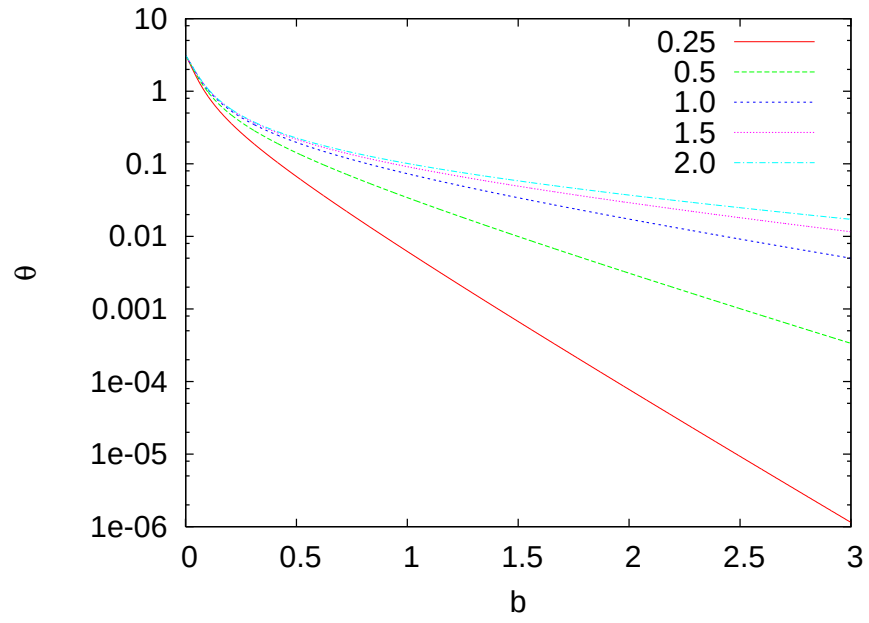
Σχήμα 4.13: Η συνάρτηση $b(\theta)$ για το δυναμικό της Σχέσης (4.27) για διαφορετικές τιμές της αρχικής ταχύτητας v . Έχουμε επιλέξει $a = 1$ και η ολοκλήρωση γίνεται σε 4000 βήματα από $t_i = 0$ έως $t_f = 40$ με $x(0) = -5$. Δίνεται συγκριτικά η σχέση (4.24) της σκέδασης Rutherford από τις καμπύλες μαρκαρισμένες ως cb .



Σχήμα 4.14: Η συνάρτηση $\sigma(\theta)$ για το δυναμικό της Σχέσης (4.27) για διαφορετικές τιμές της αρχικής ταχύτητας v . Έχουμε επιλέξει $a = 1$ και η ολοκλήρωση γίνεται σε 4000 βήματα από $t_i = 0$ έως $t_f = 40$ με $x(0) = -5$.



Σχήμα 4.15: Η συνάρτηση $b(\theta)$ για το δυναμικό Yukawa για διαφορετικές τιμές της αρχικής ταχύτητας v . Έχουμε επιλέξει $a = 1$, $k = 1$ και η ολοκλήρωση γίνεται σε 5000 βήματα από $t_i = 0$ έως $t_f = 30$ με $x(0) = -50$.



Σχήμα 4.16: Η συνάρτηση $b(\theta)$ για το δυναμικό Yukawa για διαφορετικές τιμές της ‘έκτασης’ a της δύναμης. Έχουμε επιλέξει $v = 4.0$, $k = 1$ και η ολοκλήρωση γίνεται σε 5000 βήματα από $t_i = 0$ έως $t_f = 30$ με $x(0) = -50$.

4.5 Ασκήσεις

- 4.1 Αναπαράγετε τα αποτελέσματα των Σχημάτων 4.3 και 4.4. Συγκρίνετε τα αποτελέσματά σας με την γνωστή αναλυτική λύση.
- 4.2 Προγραμματίστε τη δύναμη που αισθάνεται φορτισμένο σωματίδιο σε ομογενές μαγνητικό πεδίο $\vec{B} = B\hat{k}$ και μελετήστε την τροχιά του για $\vec{v}(0) = v_{0x}\hat{i} + v_{0y}\hat{j}$. Για $x(0) = 1, y(0) = 0, v_{0y} = 0$ υπολογίστε την ακτίνα της τροχιάς και φτιάξτε γράφημα της σχέσης ακτίνας τροχιάς και v_{0x} . Συγκρίνετε με αυτό που αναμένετε από τον αναλυτικό υπολογισμό. (Μη σχετικιστικός υπολογισμός)
- 4.3 Μελετήστε τον ανομοιογενή αρμονικό ταλαντωτή $a_x = -\omega_1^2 x, a_y = -\omega_2^2 y$. Αναπαράγετε τις καμπύλες Lissajoux παίρνοντας $x(0) = 0, y(0) = 1, v_x(0) = 1, v_y(0) = 0, t_f = 2\pi, \omega_2^2 = 1, \omega_1^2 = 1, 2, 4, 9, 16, \dots$. Τι γίνεται όταν $\omega_1^2 \neq n\omega_2^2$;
- 4.4 Αναπαράγετε τα αποτελέσματα του Πίνακα 4.1 και των Σχημάτων 4.5 και ;;. Φτιάξτε διάγραμμα των ποσοτήτων $\ln a - \ln T$ και υπολογίστε την κλίση της ευθείας που θα προκύψει με τη μέθοδο των ελαχίστων τετραγώνων. Είναι αυτή που περιμένετε; Υπολογίστε το σημείο τομής με τον κατακόρυφο άξονα και συγκρίνετε το αποτέλεσμά σας με το αναμενόμενο.
- 4.5 Υπολογίστε τη στροφορμή σε κάθε βήμα ολοκλήρωσης στην πλανητική κίνηση και μελετήστε αν πράγματι διατηρήται. Δείξτε (αναλυτικά) ότι η διατήρηση της στροφορμής συνεπάγεται ότι το διάνυσμα θέσης του πλανήτη σαρώνει επιφάνεια με σταθερό ρυθμό.
- 4.6 Υπολογίστε την ταχύτητα διαφυγής v_e για $GM = 10.0, y(0) = 0.0, v_x(0) = 0$ συναρτήσει της αρχικής απόστασης $x_0 = x(0)$. Φτιάξτε διάγραμμα των ποσοτήτων $\ln x_0 - \ln v_e$ και υπολογίστε την κλίση και το σημείο τομής με τον κατακόρυφο άξονα. Συγκρίνετε τα αποτελέσματά σας με τα αναμενόμενα. (Υποδ.: Παρατηρήστε ότι η ενέργεια για $v_0 > v_e$ γίνεται αρνητική.)
- 4.7 Εξετάστε αν για την κλειστή τροχιά του πλανήτη με $GM = 10.0, x(0) = 1, y(0) = 0.0, v_x(0) = 0, v_y(0) = 4$ ισχύει η ορίζουσα ιδιότητα της έλλειψης $F_1P + F_2P = 2a$. Ως σημείο F_1 θα πάρετε το κέντρο της δύναμης και αφού προσδιορίσετε αριθμητικά το a θα πάρετε ως F_2 το σημείο που είναι συμμετρικό ως προς το κέντρο της έλλειψης.

- 4.8 Θεωρήστε την κίνηση πλανητών σύμφωνα με την προηγούμενη άσκηση. Εφαρμόστε στιγμιαία ώθηση στην εφαπτόμενη διεύθυνση της τροχιάς αφού ο πλανήτης εκτελέσει περίπου $1/4$ της τροχιάς του. Πόσο ευσταθής είναι η τροχιά στην ώθηση (δηλ. ποιά είναι η εξάρτηση της τροχιάς από το μέγεθος/διάρκεια της ώθησης); Επαναλάβετε την ανάλυση όταν η ώθηση είναι στην κάθετη διεύθυνση.
- 4.9 Θεωρήστε το δυναμικό σκέδασης ποζιτρονίου-υδρογόνου της σχέσης (4.26). Φτιάξτε τη γραφική παράσταση της συνάρτησης $f(r)$ καθώς και της $V(r)$ για διαφορετικές τιμές του a . Υπολογίστε αριθμητικά τη συνολική ενεργό διατομή σ_{tot} και δείξτε ότι είναι ίση με πa^2 .
- 4.10 Θεωρήστε το δυναμικό Morse που χρησιμοποιείται σε μοντέλα διατομικών μορίων:

$$V(r) = D (\exp(-2\alpha r) - 2 \exp(-\alpha r)) \quad (4.30)$$

με $D, \alpha > 0$. Λύστε αριθμητικά το πρόβλημα αρχικά στη μία διάσταση και συγκρίνετε με τις γνωστές αναλυτικές λύσεις για ενέργεια $E < 0$:

$$x(t) = \frac{1}{\alpha} \ln \left\{ \frac{D - \sqrt{D(D - |E|)} \sin(\alpha t \sqrt{2|E|/m} + C)}{|E|} \right\} \quad (4.31)$$

με τη σταθερά ολοκλήρωσης να δίνεται σα συνάρτηση της αρχικής θέσης και της ενέργειας από

$$C = \sin^{-1} \left[\frac{D - |E|e^{\alpha x_0}}{\sqrt{D(D - |E|)}} \right] \quad (4.32)$$

Η κίνηση είναι περιοδική με περίοδο που εξαρτάται από την ενέργεια $T = (\pi/\alpha) \sqrt{2m/|E|}$. Για $E > 0$ έχουμε

$$x(t) = \frac{1}{\alpha} \ln \left\{ \frac{\sqrt{D(D + E)} \cosh(\alpha t \sqrt{2E/m} + C) - D}{|E|} \right\} \quad (4.33)$$

ενώ για $E = 0$

$$x(t) = \frac{1}{\alpha} \ln \left\{ \frac{1}{2} + \frac{D\alpha^2}{m}(t + C)^2 \right\}. \quad (4.34)$$

Στις τελευταίες σχέσεις η σταθερά ολοκλήρωσης C δίνεται από άλλη σχέση και όχι από την (4.32). Μελετήστε την κίνηση στο χώρο των φάσεων

$(x, \dot{x})$ και μελετήστε τη μετάβαση από ανοιχτές σε κλειστές τροχίες του συστήματος.

- 4.11 Στην προηγούμενη άσκηση θεωρήστε τον όρο του ενεργού δυναμικού $V_{eff}(r) = l^2/2mr^2$ ($l \equiv |\vec{L}|$). Κάνετε τη γραφική παράσταση της συνάρτησης $V_{tot}(r) = V(r) + V_{eff}(r)$ για $D = 20$, $\alpha = 1$, $m = 1$, $l = 1$, φυσικά για $r > 0$. Προσδιορίστε τη θέση ισορροπίας και την ενέργεια ιονισμού.

Μελετήστε αριθμητικά τις λύσεις $x(t)$, $y(t)$, $y(x)$, $r(t)$ στο επίπεδο για $E > 0$, $E = 0$, και $E < 0$. Στην τελευταία περίπτωση μελετήστε και το πρόβλημα σκέδασης, υπολογίζοντας αριθμητικά τη συνάρτηση $b(\theta)$, $\sigma(\theta)$ και τη συνολική ενεργό διατομή σ_{tot} .

- 4.12 Θεωρήστε τη δύναμη $\vec{F}(r) = f(r) \hat{r}$ όπου $f(r) = 24(2/r^{13} - 1/r^7)$ η οποία είναι μοντέλο μοριακού δυναμικού. Υπολογίστε το δυναμικό $V(r)$ και κάνετε τη γραφική παράσταση της συνάρτησης $V_{tot}(r) = V(r) + V_{eff}(r)$. Προσδιορίστε τη θέση ισορροπίας και την ενέργεια ιονισμού.

Μελετήστε το πρόβλημα σκέδασης, υπολογίζοντας αριθμητικά τη συνάρτηση $b(\theta)$, $\sigma(\theta)$ και τη συνολική ενεργό διατομή σ_{tot} . Πόσο εξαρτάται ο υπολογισμός σας από την ελάχιστη γωνία σκέδασης;

- 4.13 Μελετήστε την κίνηση σωματιδίου υπό την επίδραση ελκτικής κεντρικής δύναμης $\vec{F} = -k/r^3 \hat{r}$. Εξετάστε με ποιές αρχικές συνθήκες παίρνετε σπειροειδή τροχιά.

- 4.14 Υπολογίστε τη συνολική διαφορική διατομή σ_{tot} αναλυτικά και υπολογιστικά για την σκέδαση Rutherford. Τι παρατηρήτε στην αριθμητικά υπολογισμένη τιμή καθώς μεταβάλλετε τα όρια της ολοκλήρωσης;

- 4.15 Γράψτε πρόγραμμα που θα υπολογίζει την τροχιά φορτισμένου σωματιδίου όταν αυτό κινείται σε ηλεκτρικό πεδίο Coulomb που δημιουργείται από N ακίνητα σημειακά ηλεκτρικά φορτία.

Κεφάλαιο 5

Ηλεκτροστατική

Στο κεφάλαιο αυτό θα μελετήσουμε αριθμητικά το ηλεκτροστατικό πεδίο στατικής κατανομής φορτίων. Στην πρώτη παράγραφο μελετάμε τις δυναμικές γραμμές και ισοδυναμικές επιφάνειες κατανομής σημειακών ηλεκτρικών φορτίων στο επίπεδο. Στη δεύτερη παράγραφο εξετάζονται προβλήματα συνεχών κατανομών φορτίου, πάλι στο επίπεδο. Γίνεται αριθμητική λύση προβλημάτων συνοριακών τιμών χρησιμοποιώντας μεθόδους τύπου (over)relaxation.

5.1 Ηλεκτροστατικό Πεδίο Σημειακών Ηλεκτρικών Φορτίων

Έστω N σημειακά ηλεκτρικά φορτία Q_i βρίσκονται σε σταθερές θέσεις στο επίπεδο που δίνονται από τα διανύσματα θέσης τους $\vec{r}_i$, $i = 1, \dots, N$. Ο νόμος του Coulomb μας δίνει την τιμή του ηλεκτρικού πεδίου

$$\vec{E}(\vec{r}) = \frac{1}{4\pi\epsilon_0} \sum_{i=1}^N \frac{Q_i}{|\vec{r} - \vec{r}_i|^2} \hat{\rho}_i \quad (5.1)$$

όπου $\hat{\rho}_i = (\vec{r} - \vec{r}_i)/|\vec{r} - \vec{r}_i|$ το μοναδιαίο διάνυσμα στη διεύθυνση του $\vec{r} - \vec{r}_i$. Οι συνιστώσες του πεδίου είναι

$$\begin{aligned} E_x(x, y) &= \frac{1}{4\pi\epsilon_0} \sum_{i=1}^N \frac{Q_i(x - x_i)}{((x - x_i)^2 + (y - y_i)^2)^{3/2}} \\ E_y(x, y) &= \frac{1}{4\pi\epsilon_0} \sum_{i=1}^N \frac{Q_i(y - y_i)}{((x - x_i)^2 + (y - y_i)^2)^{3/2}}, \end{aligned} \quad (5.2)$$

Το ηλεκτροστατικό δυναμικό στη θέση $\vec{r}$ είναι

$$V(\vec{r}) = V(x, y) = \frac{1}{4\pi\epsilon_0} \sum_{i=1}^N \frac{Q_i}{((x - x_i)^2 + (y - y_i)^2)^{1/2}} \quad (5.3)$$

και ισχύει

$$\vec{E}(\vec{r}) = -\vec{\nabla}V(\vec{r}) \quad (5.4)$$

Οι δυναμικές γραμμές είναι οι ολοκληρωτικές καμπύλες του διανυσματικού πεδίου $\vec{E}$, δηλ. σε απλά Ελληνικά¹ οι καμπύλες εκείνες που εφάπτονται σε κάθε σημείο του επιπέδου με το διάνυσμα-τιμή του ηλεκτρικού πεδίου. Μια σύμβαση που ακολουθείται στο σχεδιασμό των δυναμικών γραμμών, είναι ότι η πυκνότητα τους ανά μονάδα επιφάνειας είναι ανάλογη του μέτρου του $\vec{E}$. Δηλ. ο αριθμός των δυναμικών γραμμών που τέμνει μια επιφάνεια είναι ανάλογη της ροής Φ_E του ηλεκτρικού πεδίου που τη διαπερνά.

Οι ισοδυναμικές επιφάνειες είναι ο γεωμετρικός τόπος των σημείων εκείνων όπου η συνάρτηση του δυναμικού έχει σταθερή τιμή. Η σχέση (5.4) μας λέει πως πυκνές ισοδυναμικές επιφάνειες (που σημαίνουν γρήγορη χωρική μεταβολή του δυναμικού) συνεπάγονται ισχυρό ηλεκτρικό πεδίο στην περιοχή και αντίστροφα. Επίσης ότι η διεύθυνση του ηλεκτρικού πεδίου είναι κάθετη στις ισοδυναμικές επιφάνειες σε κάθε σημείο (η διεύθυνση ταχύτερης μεταβολής του V) και με φορά αυτή της μείωσης του δυναμικού. Φυσικά στην περίπτωση που περιοριζόμαστε στο επίπεδο που βρίσκεται η κατανομή των φορτίων, οι ισοδυναμικές επιφάνειες αντιστοιχούν στις καμπύλες που είναι η τομή τους με το εν λόγω επίπεδο.

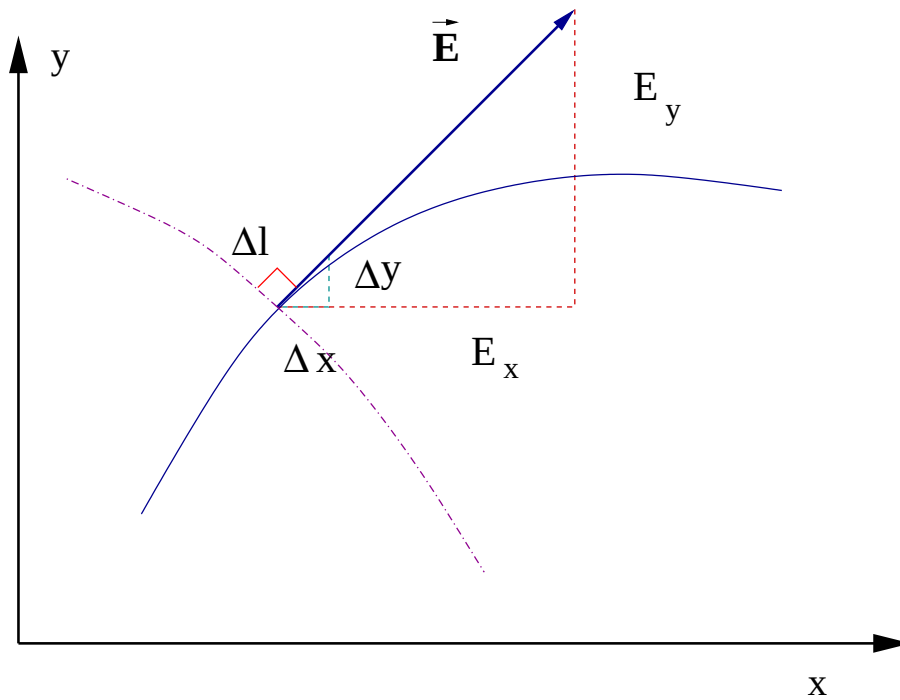
Στον υπολογισμό φυσικά δεν μπορούμε να λύσουμε το πρόβλημα στο συνεχές. Η συνεχής καμπύλη που θέλουμε να υπολογίσουμε θα προσεγγιστεί με ένα μεγάλο αλλά πεπερασμένο αριθμό από μικρά ευθύγραμμο τμήματα. Η ιδέα περιγράφεται στο Σχήμα 5.1: Το μικρό ευθύγραμμο τμήμα Δl είναι στη διεύθυνση του ηλεκτρικού πεδίου οπότε από τα όμοια τρίγωνα παίρνουμε

$$\Delta x = \Delta l \frac{E_x}{E} \quad \Delta y = \Delta l \frac{E_y}{E}, \quad (5.5)$$

όπου $E \equiv |\vec{E}| = \sqrt{E_x^2 + E_y^2}$.

Για τον υπολογισμό των ισοδυναμικών καμπύλων χρησιμοποιούμε την ιδιότητα που έχουν οι δυναμικές γραμμές να είναι παντού κάθετες στις ισοδυναμικές επιφάνειες (εδώ καμπύλες). Άρα αν $(\Delta x, \Delta y)$ δίνει την εφαπτόμενη στην

¹...it's greek to me!



Σχήμα 5.1: Σε κάθε σημείο του χώρου η δυναμικές γραμμές εφάπτονται με το διάνυσμα της έντασης του ηλ. πεδίου ενώ οι ισοδυναμικές επιφάνειες (εδώ καμπύλες) το έχουν κάθετο. Προσεγγίζοντας τη συνεχή καμπύλη με το ευθ. τμήμα Δl έχουμε $\Delta y/\Delta x = E_y/E_x$.

ισοδυναμική γραμμή, τότε το $(-\Delta y, \Delta x)$ είναι σε κάθετη κατεύθυνση αφού $(\Delta x, \Delta y) \cdot (-\Delta y, \Delta x) = -\Delta x \Delta y + \Delta y \Delta x = 0$. Οπότε για τις ισοδυναμικές καμπύλες προκύπτει η εξίσωση

$$\Delta x = -\Delta l \frac{E_y}{E} \quad \Delta y = \Delta l \frac{E_x}{E}. \quad (5.6)$$

Μπορούμε τώρα να σχεδιάσουμε μια αλγοριθμική διαδικασία που θα μας επιτρέψει τον προσεγγιστικό υπολογισμό των δυναμικών και ισοδυναμικών καμπύλων: Επιλέγουμε αρχικό σημείο από το οποίο περνάει η (μοναδική) δυναμική γραμμή ή ισοδυναμική επιφάνεια που επιθυμούμε. Από την κατανομή των φορτίων υπολογίζουμε το ηλεκτρικό πεδίο χρησιμοποιώντας τις σχέσεις (5.2). Επιλέγοντας αρκετά μικρό βήμα Δl μετακινούμαστε στη διεύθυνση $(\Delta x, \Delta y)$

$$x \rightarrow x + \Delta x \quad y \rightarrow y + \Delta y, \quad (5.7)$$

χρησιμοποιώντας τις σχέσεις (5.5) ή (5.5) ανάλογα με την περίπτωση. Επαναλαμβάνουμε τη διαδικασία μέχρι να τελειώσει ο σχεδιασμός για ένα μεγάλο αριθμό βημάτων. Το κριτήριο για αυτό θα το καθορίσει ο προγραμματιστής ή ο χρήστης ανάλογα με τις ανάγκες του υπολογισμού, λ.χ. η δυναμική γραμμή φεύγει έξω από τα όρια σχεδιασμού ή πλησιάζει κοντύτερα σε ένα φορτίο από μια ελάχιστη απόσταση.

5.2 Το Πρόγραμμα – Ορεκτικά και ... επιδόρπιο

Ο βιαστικός, αλλά και ελαφρά καταρτισμένος αναγνώστης μπορεί να συνεχίσει κατευθείαν στην παράγραφο 5.4 και να επιστρέψει εδώ αργότερα για διευκρινήσεις. Εκεί θα βρείτε την τελική και πλήρη μορφή του προγράμματος το οποίο μπορείτε να αντιγράψετε καθώς και συνοπτικές οδηγίες χρήσης του.

Για τον προγραμματισμό του αλγόριθμου που περιγράψαμε στην προηγούμενη παράγραφο, θα χωρίσουμε τις διαδικασίες σε τέσσερις ξεχωριστές και ανεξάρτητες μεταξύ τους ομάδες οι οποίες έχουν καλά ορισμένους στόχους και μπορούν να χρησιμοποιηθούν σε διαφορετικά τμήματα ενός προγράμματος.

- Το κυρίως πρόγραμμα: Διεπαφή (interface) με το χρήστη. Εισαγωγή δεδομένων και επεξεργασία αποτελεσμάτων.
- **subroutine eline(xin,yin,X,Y,Q,N)**: Υπολογίζει τη δυναμική γραμμή που περνάει από το σημείο *xin,yin*. Στην είσοδο ο χρήστης δίνει το σημείο *xin,yin* και τα δεδομένα *N*, *X(N)*, *Y(N)*, *Q(N)* όπως και πριν. Στην έξοδο η υπορουτίνα τυπώνει στην καθιερωμένη έξοδο (**standard output** - συνήθως η οθόνη ή όπου αλλού ο χρήστης επανακατευθύνει) τις συντεταγμένες της δυναμικής γραμμής που περνάει από το σημείο *xin,yin* και σταματάει είτε πολύ κοντά σε κάποιο άλλο φορτίο ή εκτός της περιοχής σχεδίασης (εδώ μια μέγιστη απόσταση από την αρχή των αξόνων). Καλεί την υπορουτίνα **efield** για τον υπολογισμό του ηλεκτρικού πεδίου και την **mdist** για τον υπολογισμό της ελάχιστης απόστασης από τα ηλεκτρικά φορτία.
- **subroutine epotline(xin,yin,X,Y,Q,N)**: Υπολογίζει την ισοδυναμική καμπύλη που περνάει από το σημείο *xin,yin*. Στην είσοδο ο χρήστης δίνει το σημείο *xin,yin* και τα δεδομένα *N*, *X(N)*, *Y(N)*, *Q(N)* όπως

και πριν. Στην έξοδο η υπορουτίνα τυπώνει στην καθιερωμένη έξοδο τις συντεταγμένες της ισοδυναμικής καμπύλης που περνάει από το σημείο x_{in}, y_{in} και σταματάει είτε όταν κλείσει αρκετά κοντά στο αρχικό σημείο² ή όταν βγει εκτός της περιοχής σχεδίασης. Καλεί την υπορουτίνα `efield` για τον υπολογισμό του ηλεκτρικού πεδίου και την `mdist` για τον υπολογισμό της ελάχιστης απόστασης από τα ηλεκτρικά φορτία.

- **subroutine `efield(x0,y0,X,Y,Q,N,Ex,Ey)`:** Καλείται από τις παραπάνω ρουτίνες με τις οποίες πρέπει να συνδεθεί (linked). Υπολογίζει το ηλεκτρικό πεδίο E_x, E_y στη θέση x_0, y_0 . Στην είσοδο ο χρήστης παρέχει τον αριθμό N και τις θέσεις των φορτίων που αποθηκεύονται στα arrays $X(N), Y(N)$ και το μέγεθος των φορτίων στο array $Q(N)$. Επίσης δίνει τη θέση x_0, y_0 στην οποία θα υπολογιστεί το ηλεκτρικό πεδίο. Στην έξοδο ο χρήστης παίρνει τις συνιστώσες του ηλεκτρικού πεδίου E_x, E_y .
- **subroutine `mdist(x0,y0,X,Y,N,rmin,rmax)`:** Καλείται από τις παραπάνω ρουτίνες με τις οποίες πρέπει να συνδεθεί. Υπολογίζει την ελάχιστη και μέγιστη απόσταση του σημείου x_0, y_0 από τα φορτία που βρίσκονται στις θέσεις $X(N), Y(N)$. Στην είσοδο ο χρήστης παρέχει τον αριθμό των φορτίων N , τις θέσεις τους $X(N), Y(N)$ και το εν λόγω σημείο. Στην έξοδο η ρουτίνα δίνει την ελάχιστη και μέγιστη απόσταση $rmin, rmax$.

Η ιδέα είναι να γράψουμε τον κώδικα για τις υπορουτίνες και να τις χρησιμοποιήσουμε μετά με διαφορετικό τρόπο στο κυρίως πρόγραμμα για να φτιάξουμε ισοδυναμικές/ισοδυναμικές γραμμές. Αρχίζουμε με το κυρίως πρόγραμμα. Σε αυτό ο χρήστης πρέπει να ορίσει τις μεταβλητές $N, X(N), Y(N), Q(N)$ που αποτελούν τη δομή των δεδομένων μας. Θα πρέπει να ζητήσει από το χρήστη τις σχετικές πληροφορίες ('διεπαφή με χρήστη') και χρησιμοποιώντας τις να καλέσει τις σχετικές ρουτίνες που υλοποιούν τον υπολογισμό. Τα arrays X, Y, Q ορίζονται να έχουν μέγεθος μεγαλύτερο από οποιοδήποτε αριθμό N σκεφτόμαστε να χρησιμοποιήσουμε. Οπότε θα τα ορίσουμε να έχουν μέγεθος $P=20$ οπότε θα μελετήσουμε το ηλεκτρικό πεδίο μέχρι 20 σημειακών φορτίων. Έτσι μια μινιμαλιστική προσέγγιση θα είναι να γράψουμε σε ένα αρχείο `ELines.f` ένα πρόγραμμα της μορφής ('πρώτη έκδοση' - version 1:

```
C *****
  program Electric_Fields
```

²Θυμάστε φαντάζομαι πως οι ισοδυναμικές επιφάνειες, άρα και οι καμπύλες, είναι κλειστές.

```

C      *****
      implicit none
      integer    P                !max number of charges
      parameter(P=20)
      real       X(P),Y(P),Q(P)
      integer    N

C      -----  SET CHARGE DISTRIBUTION  -----
      N      =  2
      X(1) =  1.0
      Y(1) =  0.0
      Q(1) =  1.0
      X(2) = -1.0
      Y(2) =  0.0
      Q(2) = -1.0

C      -----  DRAWING LINES  -----
      call eline(0.0, 0.5,X,Y,Q,N)
      call eline(0.0, 1.0,X,Y,Q,N)
      call eline(0.0, 1.5,X,Y,Q,N)
      call eline(0.0, 2.0,X,Y,Q,N)
      call eline(0.0,-0.5,X,Y,Q,N)
      call eline(0.0,-1.0,X,Y,Q,N)
      call eline(0.0,-1.5,X,Y,Q,N)
      call eline(0.0,-2.0,X,Y,Q,N)

      end

```

Στο πρώτο κομμάτι του προγράμματος, η εντολή

```
program Electric_Fields
```

ορίζει ότι αυτό το κομμάτι του κώδικα είναι η 'κύρια' ρουτίνα από την οποία αρχίζει η εκτέλεση του προγράμματος. Αυτό σταματάει στην εντολή

```
end
```

Το πρώτο κομμάτι του προγράμματος είναι οι δηλώσεις των μεταβλητών οι οποίες αρχίζουν με την εντολή

```
implicit none
```

η οποία μας υποχρεώνει να δηλώσουμε ρητά κάθε μεταβλητή και όχι να αφήσουμε το μεταγλωττιστή να το κάνει αυτόματα με βάση τους εσωτερικούς κανόνες της Fortran 77. Αυτό δεν είναι υποχρεωτικό, αλλά το συστήνουμε να είναι η πάγια πρακτική στα προγράμματά σας ώστε να αποφύγετε δύσκολα να βρεθούν σφάλματα (bugs) που οφείλονται σε ορθογραφικά λάθη κατά την πληκτρολόγηση του προγράμματος. Οι δηλώσεις προηγούνται των εκτελέσιμων εντολών. Τα μεγέθη των arrays καθορίζονται από την παράμετρο P τύπου INTEGER στην οποία δίνουμε την τιμή 20 μέσω της εντολής

```
parameter(P=20)
```

Αυτό δηλώνει πως η τιμή της παραμέτρου δεν μπορεί να αλλάξει κατά τη διάρκεια του προγράμματος. Το ότι τα arrays X, Y, Q έχουν μέγεθος P καθορίζεται από την εντολή

```
real      X(P),Y(P),Q(P)
```

η οποία ταυτόχρονα δηλώνει ότι είναι τύπου REAL. Στην επόμενη γραμμή ορίζουμε και τη μεταβλητή N τύπου INTEGER, το αριθμό των φορτίων στο πρόβλημα.

Στο δεύτερο κομμάτι του προγράμματος, θέτουμε τον αριθμό των φορτίων N=2 και τις αντίστοιχες θέσεις και μεγέθη φορτίων. Έτσι οι εντολές

```
C      -----  SET CHARGE DISTRIBUTION  ----
N      =   2
X(1)   =   1.0
Y(1)   =   0.0
Q(1)   =   1.0
X(2)   =  -1.0
Y(2)   =   0.0
Q(2)   =  -1.0
```

ορίζουν στο πρόβλημα 2 ίσα και αντίθετα φορτία $Q(1) = -Q(2) = 1.0$ στις θέσεις (1,0) και (-1,0) αντίστοιχα. Οι επόμενες γραμμές καλούν την υπορουτίνα **eline** να κάνει τον υπολογισμό για 8 δυναμικές γραμμές που περνούν αντίστοιχα από τα σημεία $(0, \pm 1/2)$, $(0, \pm 1)$, $(0, \pm 3/2)$, $(0, \pm 2)$:

```
C      -----  DRAWING LINES  -----
call eline(0.0, 0.5,X,Y,Q,N)
call eline(0.0, 1.0,X,Y,Q,N)
call eline(0.0, 1.5,X,Y,Q,N)
```

```
call eline(0.0, 2.0,X,Y,Q,N)
call eline(0.0,-0.5,X,Y,Q,N)
call eline(0.0,-1.0,X,Y,Q,N)
call eline(0.0,-1.5,X,Y,Q,N)
call eline(0.0,-2.0,X,Y,Q,N)
```

Οι εντολές τυπώνουν τις συντεταγμένες των σημείων των δυναμικών γραμμών στο standard output τις οποίες ο χρήστης θα επεξεργαστεί περαιτέρω.

Για τον υπολογισμό των ισοδυναμικών καμπύλων ο κώδικας είναι ακριβώς ο ίδιος αντικαθιστώντας `call eline` → `call epotline`.

Με τον τρόπο αυτό η κατανομή των φορτίων και οι δυναμικές γραμμές που ζητούνται είναι 'σκληρά προγραμματισμένες' (hard coded) και κάθε φορά που θέλουμε καινούργιες γραμμές ή να προσθέσουμε/αφαιρέσουμε/μετακινήσουμε φορτία θα πρέπει να επαναμεταγλωττίσουμε τον κώδικα και μετά να τον ξανατρέξουμε.

Έστω ότι το πρόγραμμα για την `eline` και τις ρουτίνες από τις οποίες εξαρτάται έχουν ήδη προγραμματιστεί³ μέσα στο ίδιο αρχείο `ELines.f`. Τότε ο χρήστης θα πρέπει να μεταγλωττίσει τον κώδικα με την εντολή⁴

```
> f77 ELines.f -o el
```

Με την εντολή αυτή ο μεταγλωττιστής (compiler) `f77` της Fortran 77 μετατρέπει τις εντολές της γλώσσας σε οδηγίες γλώσσας μηχανής οι οποίες αποθηκεύονται στο αρχείο `el` μέσω του διακόπτη `-o el`. Το πρόγραμμα τώρα μπορεί να τρέξει με την εντολή

```
> ./el > el.out
```

όπου με το χαρακτήρα `>` επαναορισμού του `standard output` μεταφέρουμε τα αποτελέσματα που τυπώνει το πρόγραμμα στο αρχείο `el.out`. Για να δούμε τα αποτελέσματα, χρησιμοποιούμε το πρόγραμμα `gnuplot`⁵:

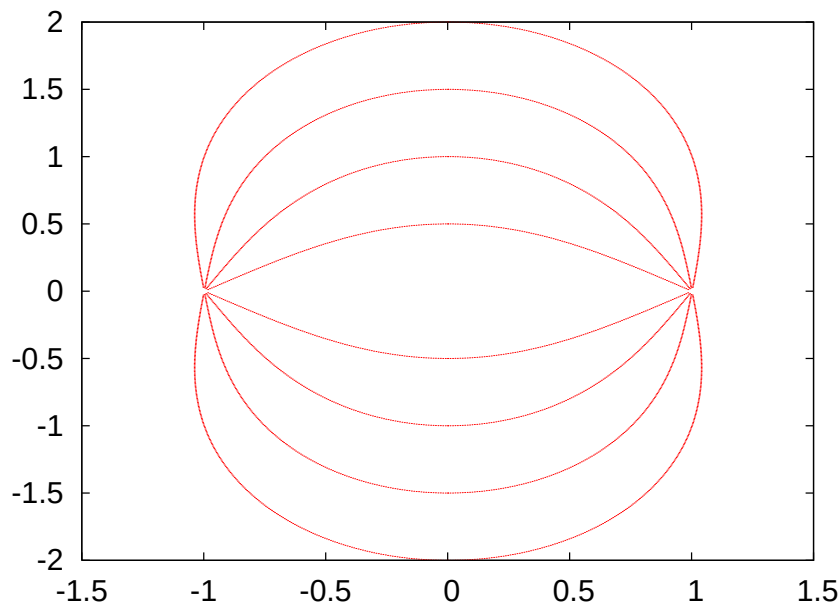
```
> gnuplot
gnuplot> plot "el.out" with dots
```

³Επιστρέψτε δηλ. να εκτελέσετε τις εντολές αυτές αφού συζητήσουμε τον προγραμματισμό των `eline`, `efield`, `mdist` ή χρησιμοποιήστε τις από το συνοδευτικό λογισμικό.

⁴Οι εντολές αυτές εξαρτώνται από το διαθέσιμο μεταγλωττιστή του χρήστη και το λειτουργικό του σύστημα. Αναφερόμαστε σε ένα τυπικό σύστημα τύπου Unix όπως τα GNU/Linux.

⁵www.gnuplot.info

Στην πρώτη γραμμή δίνουμε την εντολή `gnuplot` από τη γραμμή εντολών ενώ στη δεύτερη αφού το πρόγραμμα ξεκινήσει και μας δώσει prompt (το `gnuplot>`) δίνουμε την εντολή `plot` του προγράμματος με τα ορίσματα που αναφέρονται παραπάνω. Το αποτέλεσμα φαίνεται στο Σχήμα 5.2.



Σχήμα 5.2: Μερικές δυναμικές γραμμές ηλ. πεδίου δύο αντίθετων φορτίων που σχεδιάζουμε με το πρόγραμμα `ELines.f` στην ... πρώτη του έκδοση.

Ας γράψουμε τώρα το πρόγραμμα δίνοντας στο χρήστη την ευκαιρία να μελετήσει ευκολότερα διαφορετικές κατανομές φορτίων και επιλογή δυναμικών γραμμών. Η ιδέα είναι ο χρήστης να παρέχει διαδραστικά τον αριθμό και τη θέση/μέγεθος των ηλεκτρικών φορτίων καθώς και τον αριθμό και αρχικά σημεία των δυναμικών γραμμών. Ο πρώτος στόχος πετυχαίνεται αλλάζοντας τον κώδικα στο σημείο που θέτει την κατανομή φορτίων ως εξής:

```
C      ----- SET CHARGE DISTRIBUTION -----
      print *, '# Enter number of charges:'
      read(5,*) N
      print *, '# N= ',N
      do i=1,N
```

```

print *, '# Charge: ', i
print *, '# Position and charge: (X,Y,Q):'
read(5,*) X(i),Y(i),Q(i)
print *, '# (X,Y)= ', X(i),Y(i), ' Q= ',Q(i)
enddo

```

Η πρώτη εντολή ζητάει από το χρήστη τον αριθμό των φορτίων της κατανομής, το οποίο διαβάζει από το standard input (το οποίο είναι by default η γραμμή εντολών του φλοιού). Το 5 στην εντολή read υποδηλώνει το unit 5 το οποίο στα περισσότερα λειτουργικά συστήματα ορίζεται να είναι το standard input. Το * είναι εντολή format της Fortran 77 και απλά λέει το format να το αφήσουμε στο ... έλεος του compiler. Στη συνέχεια αρχίζουμε ένα βρόχο do ο οποίος θα τρέξει N φορές αυξάνοντας κάθε φορά την τιμή της μεταβλητής i από 1 έως N με βήμα 1. Για κάθε τιμή του i διαβάζουμε τις τιμές της θέσης/μέγεθος φορτίου και τις αποθηκεύουμε στις αντίστοιχες θέσεις των arrays X(i), Y(i), Q(i). Τα αποτελέσματα τυπώνονται για να τα βλέπει ο χρήστης και να ελέγχει ότι πέρασαν σωστά στη μνήμη του υπολογιστή.

Ο σχεδιασμός των δυναμικών γραμμών γίνεται αλλάζοντας τον κώδικα στο σημείο σχεδίασης των γραμμών ως εξής:

```

C ----- DRAWING LINES -----
print *, '# How many lines to draw? '
read(5,*) draw
do i=1,draw
  print *, '# Initial point (x0,y0): '
  read(5,*) x0,y0
  call eline(x0,y0,X,Y,Q,N)
enddo

```

Ζητείται από το χρήστη να εισάγει τον αριθμό των δυναμικόγραμμών που θέλει να σχεδιαστούν και αυτός αποθηκεύεται στη μεταβλητή draw. Στη συνέχεια ο βρόχος do εκτελείται draw φορές και κάθε φορά αφού ζητηθεί από το χρήστη το αρχικό σημείο σχεδίασης της γραμμής, καλείται η υπορουτίνα eline η οποία κάνει το σχεδιασμό.

Για να τρέξει το πρόγραμμα (πάλι υποθέτοντας ότι η υπορουτίνα eline και οι ρουτίνες από τις οποίες εξαρτάται έχουν ήδη προγραμματιστεί) μεταγλωττίζουμε με την εντολή f77 τον κώδικα του αρχείου ELines.f το τρέχουμε όπως πριν. Η διαφορά είναι ότι τώρα αλληλεπιδρούμε με το πρόγραμμα και εισάγουμε δεδομένα από τη γραμμή εντολών (εδώ ένα φορτίο $q = 1.0$ στην αρχή των αξόνων):

```

> f77 ELines.f -o el
> ./el
# Enter number of charges:
1
# N=          1
# Charge:          1
# Position and charge: (X,Y,Q):
0.0 0.0 1.0
# (X,Y)=      0.000000      0.000000      Q=      1.000000
# How many lines to draw?
1
# Initial point (x0,y0):
0.1 0.1
  9.2928931E-02  9.2928931E-02
  8.5857861E-02  8.5857861E-02
  7.8786790E-02  7.8786790E-02
....

```

Τι κερδίσαμε; Αν το δοκιμάσετε θα πείτε πως η μορφή αυτή του προγράμματος δεν είναι εύχρηστη αφού τα αποτελέσματα που τυπώνονται στην οθόνη δεν είναι εύκολο να τα επεξεργαστούμε (δοκιμάστε...). Ας υποθέσουμε όμως ότι θέλετε να κάνετε εντατική μελέτη του προβλήματος. Γράψτε με τον editor της αρεσκείας σας⁶ σε ένα αρχείο **Input** τα δεδομένα που θέλετε να εισάγετε ως εξής:

2	N: Number of Charges
1.0 0.0 1.0	(X,Y,Q): Position and charge
-1.0 0.0 -1.0	(X,Y,Q): Position and charge
8	Number of lines to draw
0.0 0.5	x0,y0: Initial point of line
0.0 1.0	x0,y0: Initial point of line
0.0 1.5	x0,y0: Initial point of line
0.0 2.0	x0,y0: Initial point of line
0.0 -0.5	x0,y0: Initial point of line
0.0 -1.0	x0,y0: Initial point of line
0.0 -1.5	x0,y0: Initial point of line
0.0 -2.0	x0,y0: Initial point of line

⁶Λ.χ. τον emacs (www.gnu.org/software/emacs/)

Αν δώσετε τώρα την εντολή ⁷

```
./el < Input > el.out
```

το αποτέλεσμα θα είναι πανομοιότυπο με αυτό στην πρώτη έκδοση του προγράμματος, και το μελετούμε ακριβώς όπως πριν με το `gnuplot`⁸. Η διαφορά τώρα είναι πως για να μελετήσουμε περισσότερες δυναμικές γραμμές ή/και διαφορετικές κατανομές φορτίων, αρκεί να αλλάζουμε τα περιεχόμενα του αρχείου `Input` και να επαναλάβουμε την παραπάνω εντολή χωρίς να χρειαστεί να επαναμεταγλωττίσουμε το πρόγραμμα και χωρίς να χρειάζεται ο χρήστης να γνωρίζει πώς το πρόγραμμα έχει γραφτεί. Προσέξτε πως η Fortran 77 αφού διαβάσει τα δεδομένα που χρειάζεται, αγνοεί το περιεχόμενο της υπόλοιπης γραμμής του αρχείου `Input`. Αυτό το χρησιμοποιούμε για να γράψουμε κατατατοπιστικά σχόλια για το χρήστη.

Ο παραπάνω κώδικας ('δεύτερη έκδοση' - version 2) παρατίθεται για ευκολία του αναγνώστη παρακάτω:

```
C      *****
      program Electric_Fields
C      *****
      implicit none
      integer    P                !max number of charges
      parameter(P=20)
      real       X(P),Y(P),Q(P)
      integer    N

      integer    i,j,draw
      real       x0,y0
C      ----- SET CHARGE DISTRIBUTION -----
      print *, '# Enter number of charges:'
      read(5,*) N
      print *, '# N= ',N
      do i=1,N
         print *, '# Charge: ',i
         print *, '# Position and charge: (X,Y,Q):'
         read(5,*)  X(i),Y(i),Q(i)
         print *, '# (X,Y)= ', X(i),Y(i), ' Q= ',Q(i)
```

⁷To '< Input' επαναορίζει το standard input να είναι το αρχείο με όνομα `Input`.

⁸Οι γραμμές στο αρχείο `el.out` που αρχίζουν από # είναι σχόλια για το `gnuplot`.

```

      enddo
C      ----- DRAWING LINES -----
      print *, '# How many lines to draw? '
      read(5,*) draw
      do i=1,draw
        print *, '# Initial point (x0,y0): '
        read(5,*) x0,y0
        call eline(x0,y0,X,Y,Q,N)
      enddo
      end

```

Ήδη ο προσεκτικός αναγνώστης θα έχει ασκηθεί αρκετά ώστε να καταλάβει πως για να φτιάξει μια ωραία εικόνα από αντιπροσωπευτικές δυναμικές γραμμές χρειάζεται αρκετό χρόνο για να το κάνει... Πως θα μπορούσαμε, τουλάχιστον αρχικά να πάρουμε μια εικόνα και να αφήσουμε τις λεπτομέρειες για το πρόγραμμα της δεύτερης έκδοσης; Για τις δυναμικές γραμμές η απάντηση είναι εύκολη: Αρκετά κοντά σε ένα σημειακό ηλεκτρικό φορτίο, το ηλεκτρικό πεδίο είναι κατά πολύ καλή προσέγγιση ισοτροπικά ακτινικό. Ο αριθμός των δυναμικών γραμμών που ξεκινούν/καταλήγουν από/σε ένα σημειακό θετικό/αρνητικό ηλεκτρικό φορτίο είναι ανάλογος του μεγέθους του φορτίου. Έτσι αρκεί να πάρουμε για αρχικά σημεία έναν αριθμό ανάλογο του φορτίου από ισότροπα κατανομημένα σημεία πάνω σε ένα αρκετά μικρό κυκλάκι γύρω από κάθε φορτίο της κατανομής. Παρακάτω παραθέτουμε τον κώδικα ('τρίτη έκδοση' - version 3) για φορτία ίσα σε μέγεθος και αφήνουμε ως άσκηση στον αναγνώστη την περίπτωση φορτίων που είναι διαφορετικού μεγέθους:

```

C      *****
      program Electric_Fields
C      *****
      implicit none
      integer    P                !max number of charges
      parameter(P=20)
      real       X(P),Y(P),Q(P)
      integer    N

      real       PI
      parameter(PI= 3.14159265359)
      integer    i,j,nd
      real       x0,y0,theta

```

```

C      ----- SET CHARGE DISTRIBUTION -----
print *, '# Enter number of charges:'
read(5,*) N
print *, '# N= ',N
do i=1,N
  print *, '# Charge: ',i
  print *, '# Position and charge: (X,Y,Q):'
  read(5,*) X(i),Y(i),Q(i)
  print *, '# (X,Y)= ', X(i),Y(i), ' Q= ',Q(i)
enddo

C      ----- DRAWING LINES -----
C      We draw 2*nd field lines around each charge
nd      = 6
do i = 1,N
  do j = 1,(2*nd)
    theta = (PI/nd)*j
    x0     = X(i) + 0.1 * cos(theta)
    y0     = Y(i) + 0.1 * sin(theta)
    call   eline(x0,y0,X,Y,Q,N)
  enddo
enddo

end

```

Θα παρατηρήσατε ήδη πως η μόνη αλλαγή που κάναμε είναι στο κομμάτι σχεδιασμού των γραμμών. Θέτουμε τον αριθμό των αρχικών γραμμών γύρω από κάθε φορτίο να είναι 12 με την εντολή `nd = 6` και μετά γύρω από κάθε φορτίο (`do i = 1,N`) καλούμε την `eline` με αρχικά σημεία (`x0,y0`) πάνω σε ένα κύκλο κέντρου (`X(i),Y(i)`) και ακτίνας 0.1. Αυτό γίνεται `2*nd` φορές σε σημεία που καθορίζονται από τη γωνία `theta=(PI/nd)*j`.

Για να το τρέξουμε, γράφουμε ένα αρχείο `Input` με περιεχόμενα την κατανομή των φορτίων. Για παράδειγμα, για μια κατανομή τεσσάρων φορτίων $q_i = \pm 1$ πάνω στις κορυφές ενός τετραγώνου γράφουμε:

```

4      N: Number of charges
1  1  -1      (X,Y,Q): Position and charge
-1  1   1      (X,Y,Q): Position and charge
1 -1   1      (X,Y,Q): Position and charge

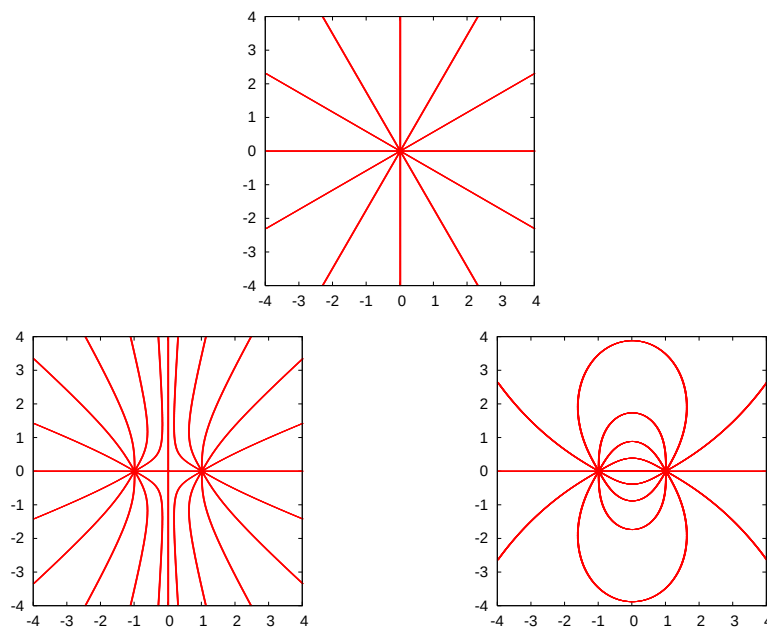
```

-1 -1 -1 (X,Y,Q): Position and charge

και μετά δίνουμε τις εντολές:

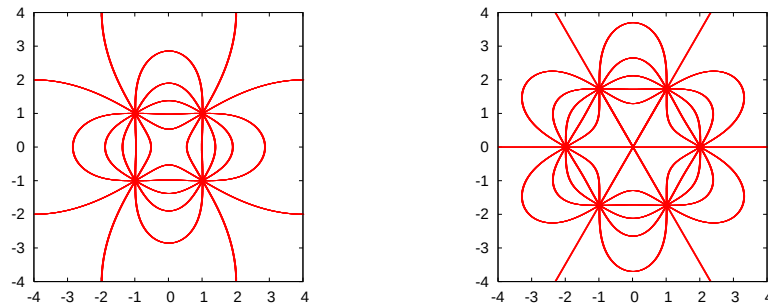
```
> f77 ELines.f -o el
> ./el < Input > el.out
> gnuplot
gnuplot> plot "el.out" with dots
```

Στα Σχήματα 5.3-5.4 δείχνουμε τα αποτελέσματα για μερικές κατανομές φορτίου. Αφήνουμε σαν άσκηση στον αναγνώστη να βρει από ποιές κατανομές προκύπτουν και να αναπαράγει τα σχήματα αυτά για εξάσκηση.



Σχήμα 5.3: Δυναμικές γραμμές ηλεκτρικού πεδίου κατανομής σημειακών φορτίων που σχεδιάζουμε με το πρόγραμμα ELines.f.

Με παρόμοιο τρόπο γράφουμε πρόγραμμα και για τις ισοδυναμικές καμπύλες. Η μόνη διαφορά είναι αν θέλουμε τα αρχικά σημεία να επιλέγονται αυτόματα, θα πρέπει να επινοήσουμε ένα νέο αλγόριθμο που να ζωγραφίζει τις δυναμικές γραμμές για δεδομένες πτώσεις δυναμικού. Λόγω της πολυπλοκότητας του αλγόριθμου, αυτό αφήνεται σαν άσκηση για τον αναγνώστη (δες σχετική άσκηση



Σχήμα 5.4: Δυναμικές γραμμές ηλεκτρικού πεδίου κατανομής σημειακών φορτίων που σχεδιάζουμε με το πρόγραμμα ELines.f.

με οδηγίες). Εμείς στον παρακάτω κώδικα επιλέγουμε για αρχικά σημεία ισαπέχοντα σημεία πάνω σε ένα τετραγωνικό πλέγμα. Ο κώδικας αποθηκεύεται στο αρχείο EPotential.f:

```
C *****
program Electric_Potential
C *****
implicit none
integer    P                !max number of charges
parameter(P=20)
real       X(P),Y(P),Q(P)
integer    N

real       PI
parameter(PI= 3.14159265359)
integer    i,j,nd
real       x0,y0,rmin,rmax,L

print *, '# Enter number of charges:'
read(5,*)  N
print *, '# N= ',N
do i=1,N
  print *, '# Charge: ',i
  print *, '# Position and charge: (X,Y,Q):'
  read(5,*)  X(i),Y(i),Q(i)
```



```

        print *, '# (X,Y)= ', X(i), Y(i), ' Q= ', Q(i)
    enddo

C      ----- DRAWING LINES -----
C      We draw lines passing through an equally
C      spaced lattice of N=(2*nd+1)x(2*nd+1) points
C      in the square -L<= x <= L, -L<= y <= L.
        nd      = 4
        L        = 1.0
        do i = -nd, nd
            do j = -nd, nd
                x0 = i*(L/nd)
                y0 = j*(L/nd)
                call mdist(x0,y0,X,Y,N,rmin,rmax)
C      we avoid getting too close to a charge:
                if(rmin .gt. L/(nd*10) )
*           call epotline(x0,y0,X,Y,Q,N)
            enddo
        enddo
    end

```

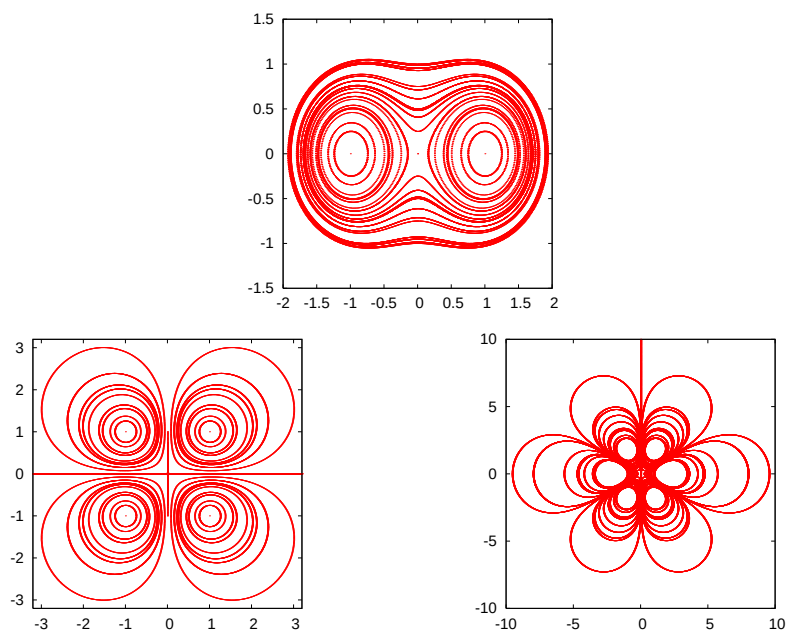
Ο κώδικας είναι πανομοιότυπος με τον προηγούμενο στο πρώτο και δεύτερο μέρος του. Στο τρίτο μέρος, όπου ζητείται ο σχεδιασμός των καμπύλων, καλείται τώρα η υπορουτίνα `epotline` να κάνει το σχεδιασμό. Όλα τα υπόλοιπα αναφέρονται στον προσδιορισμό των αρχικών σημείων που γίνεται ως εξής: Επιλέγουμε τον αριθμό των σημείων του πλέγματος με την εντολή `nd=4`, η οποία δίνει $(2*nd+1)*(2*nd+1) = 25$ σημεία. Με την εντολή `L=1.0` καθορίζουμε τα όρια του πλέγματος να είναι το τετράγωνο $(1,1)$, $(-1,1)$, $(-1,-1)$, $(1,-1)$. Στη συνέχεια, για κάθε σημείο του πλέγματος $(x0,y0)$ υπολογίζουμε την ισοδυναμική καμπύλη που περνάει από αυτό με την προϋπόθεση το σημείο αυτό να μην είναι πολύ κοντά σε ένα από τα φορτία. Αυτό γίνεται καλώντας την υπορουτίνα `mdist` από την οποία παίρνουμε την ελάχιστη απόσταση `rmin` του σημείου και βάζοντας κάτω όριο σε αυτή τον αριθμό `L/(nd*10)`. Για να το τρέξουμε δίνουμε τις εντολές:

```

> f77 EPotential.f -o ep
> ./ep < Input > ep.out
> gnuplot
gnuplot> plot "ep.out" with dots

```

Μερικά από τα αποτελέσματα δείχνονται στο Σχήμα 5.5.



Σχήμα 5.5: Ισοδυναμικές καμπύλες ηλεκτρικού πεδίου κατανομής σημειακών φορτίων που σχεδιάζουμε με το πρόγραμμα EPotential.f.

5.3 Το Πρόγραμμα - Το Κυρίως Πιάτο

Στην παράγραφο αυτή περιγράφουμε τη λειτουργία και τον προγραμματισμό των κυρίων κομματιών του προγράμματος: Τις υπορουτίνες υπολογισμού των καμπύλων `eline`, `epotline`, του ηλεκτρικού πεδίου `efield` και της ελάχιστης απόστασης από τα φορτία `mdist`.

Αρχίζουμε από την ρουτίνα `eline`. Ο χρήστης την καλεί με την εντολή

```
call eline(x0,y0,X,Y,Q,N)
```

δίνοντας στην είσοδο το αρχικό σημείο της γραμμής (x_0, y_0) , τον αριθμό των φορτίων N καθώς και τις θέσεις των φορτίων $(X(N), Y(N))$ και μεγέθη των φορτίων $Q(N)$. Για να σχεδιαστεί η δυναμική γραμμή χρειάζονται το βήμα Δl της σχέσης (5.5) καθώς και τα όρια σχεδίασης των γραμμών. Τα όρια αυτά είναι δύο: Πρώτα όταν πλησιάζουμε πολύ κοντά σε ένα φορτίο. Τότε το

ηλεκτρικό πεδίο τείνει στο άπειρο και αυτό θα δημιουργήσει προβλήματα. Το βήμα Δl είναι μια καλή κλίμακα μικρού μήκους, μια και αυτή είναι έτσι και αλλιώς το όριο της προσέγγισης της συνεχούς καμπύλης. Θα θέσουμε το όριο προσέγγισης σε $2\Delta l$. Επίσης δε θέλουμε να απομακρυνθούμε πολύ από την κατανομή, οπότε θέτουμε μια αυθαίρετη μέγιστη απόσταση από την αρχή των αξόνων `max_dist=20.0`. Άλλες προβληματικές καταστάσεις που πρέπει να προβλέψουμε, είναι να μηδενιστεί το ηλεκτρικό πεδίο οπότε το αποτέλεσμα του υπολογισμού στη Σχέση (5.5) γίνεται απροσδιόριστο⁹. Τέλος, παίρνοντας $\Delta l > 0$ μετακινούμαστε πάνω στη δυναμική γραμμή μόνο προς την κατεύθυνση του ηλεκτρικού πεδίου καταλήγοντας πάντα πάνω σε ένα αρνητικό φορτίο ή στη μέγιστη απόσταση (γιατί;). Για να σχεδιάσουμε τη δυναμική γραμμή ολόκληρη, θα επαναλάβουμε τον υπολογισμό από το ίδιο αρχικό σημείο με $\Delta l < 0$.

Η κωδικοποίηση της διαδικασίας φαίνεται παρακάτω:

```
subroutine eline(xin,yin,X,Y,Q,N)
implicit none
integer N
real X(N),Y(N),Q(N),xin,yin,x0,y0
real step !absolute value of step on field line
parameter(step=0.01)
real max_dist !max distance of field line
parameter(max_dist=20.0)
integer i,direction
real rmin,rmax,r,dx,dy,d1
real Ex,Ey,E
do direction = -1,1,2 !direction= +/- 1
  d1 = direction * step
  x0 = xin
  y0 = yin
  dx = 0.0
  dy = 0.0
  call mdist(x0,y0,X,Y,N,rmin,rmax)
  do while(rmin .gt. (2.0*step) .and. rmax .lt. max_dist)
    print *,x0,y0
```

⁹Το αποτέλεσμα των πράξεων θα εξαρτηθεί από τη σειρά με την οποία γίνεται. Νομίζω πως ανάλογα με τον μεταγωγτιστή, αν και πώς γίνεται βελτιστοποίηση, τον επεξεργαστή, το λειτ. σύστημα και τις βιβλιοθήκες το αποτέλεσμα θα είναι διαφορετικό. Αν γνωρίζετε καλύτερα, πείτε μου...

```

C      We evaluate the E-field at the midpoint: This reduces
C      systematic errors
      call efield(x0+0.5*dx,y0+0.5*dy,X,Y,Q,N,Ex,Ey)
      E = sqrt(Ex*Ex+Ey*Ey)
      if( E .le. 1.0e-10 ) goto 11 !stop calculation
      dx = dl*Ex/E
      dy = dl*Ey/E
      x0 = x0 + dx
      y0 = y0 + dy
      call mdist(x0,y0,X,Y,N,rmin,rmax)
      enddo                                !do while()
11      continue                          !come here if E=0
      enddo                                !do direction = -1,1,2
      end

```

Στο πρώτο κομμάτι του κώδικα δηλώνονται οι μεταβλητές. Το μόνο καινούργιο που έχουμε να πούμε είναι η δήλωση

```
real      X(N),Y(N),Q(N)
```

αντί για $X(P), Y(P), Q(P)$. Αυτό δεν πειράζει σε τίποτα, αρκεί φυσικά ο προγραμματιστής να έχει ήδη ελέγξει ότι $N \leq P$. Τα arrays X, Y, Q περνάνε στην υπορουτίνα 'βψ ρεφερενσε', δηλ. δίνεται η θέση στη μνήμη στην οποία είναι αποθηκευμένα και όχι 'βψ αλυε'. Στο αρχικό πρόγραμμα που δηλώθηκαν, ζητήθηκε και ο φυσικός χώρος στη μνήμη του υπολογιστή όπου θα αποθηκευτούν τα δεδομένα. Τώρα αυτό θεωρείται δεδομένο. Το μέγεθος του βήματος Δl καθώς και η μέγιστη απόσταση σχεδιασμού καθορίζονται με τις δύο δηλώσεις

```
parameter(step=0.01)
parameter(max_dist=20.0)
```

και οι τιμές αυτές θα πρέπει να καθοριστούν προσεκτικά από τον προγραμματιστή ανάλογα με τη ζητούμενη ακρίβεια επίλυσης του προβλήματος.

Στο κυρίως πρόγραμμα παρατηρούμε τον εξωτερικό βρόχο

```

do direction = -1,1,2      !direction= +/- 1
  dl = direction * step
  ...
enddo                      !do direction = -1,1

```

αλλάζει την κατεύθυνση κίνησης πάνω στη δυναμική γραμμή. Η εντολή `do direction = -1,1,2` εκτελεί το βρόχο για `direction` από -1 έως 1 με βήμα 2 . Δηλ. θα εκτελεστεί δύο φορές για `direction =  $\pm 1$` . Άρα το βήμα `dl` έχει κάθε φορά διαφορετικό πρόσημο.

Στη συνέχεια οι εντολές `x0 = xin, y0 = yin` ορίζουν το αρχικό σημείο της δυναμικής γραμμής. `(x0, y0)` είναι το εκάστοτε σημείο της δυναμικής γραμμής το οποίο τυπώνουμε στο `standard output` με την εντολή `print *`. `(dx, dy)` είναι το βήμα μετακίνησης πάνω στη δυναμική γραμμή, έτσι ώστε `(x0, y0)  $\rightarrow$  (x0+dx, y0+dy)` μετά από κάθε υπολογισμό, και αρχικοποιούνται στην τιμή `(0.0, 0.0)`. Ο κύριος σχεδιασμός γίνεται στον εσωτερικό βρόχο

```
call mdist(x0,y0,X,Y,N,rmin,rmax)
do while(rmin .gt. (2.0*step) .and. rmax .lt. max_dist)
...
call mdist(x0,y0,X,Y,N,rmin,rmax)
enddo                                !do while()
```

ο οποίος εκτελείται όταν η συνθήκη `rmin .gt. (2.0*step) .and. rmax .lt. max_dist` έχει την τιμή `.TRUE.`, είναι δηλ. αληθής. Αυτό ισχύει όσο η ελάχιστη απόσταση από όλα τα φορτία στο εκάστοτε σημείο που βρισκόμαστε δεν έχει γίνει μικρότερη ή ίση από `2.0*step` και η μέγιστη απόσταση από οποιοδήποτε φορτίο παραμένει μικρότερη από `max_dist`¹⁰. Οι μέγιστες και ελάχιστες αποστάσεις προσδιορίζονται με κάλεσμα στην υπορουτίνα `mdist` την οποία θα μελετήσουμε παρακάτω.

Το ηλεκτρικό πεδίο που θα χρησιμοποιηθεί στη Σχέση (5.5) υπολογίζεται καλώντας `efield(x0+0.5*dx,y0+0.5*dy,X,Y,Q,N,Ex,Ey)`. Τα δύο πρώτα ορίσματα της υπορουτίνας είναι το σημείο στο οποίο ζητούμε το ηλεκτρικό πεδίο, και αυτό επιλέγεται να είναι το `(x0+dx/2,y0+dy/2)` αντί για το `(x0,y0)`. Αυτό γίνεται για να μειώσουμε το συστηματικό σφάλμα από τη διακριτοποίηση του προβλήματος παίρνοντας συνεισφορά από το μέσο του διαστήματος `(x0,x0+dx)` και `(y0,y0+dy)` και όχι από τό ένα άκρο μόνο. Αυτό θα καταστεί σαφέστερο στο κεφάλαιο 3 όπου θα μελετήσουμε την ολοκλήρωση παρόμοιων προβλημάτων διαφορικών εξισώσεων με αρχικές τιμές, στο πλαίσιο της ολοκλήρωσης των εξισώσεων κίνησης.

Μετά τον υπολογισμό του ηλεκτρικού πεδίου, η εφαρμογή της σχέσης (5.5) γίνεται απλά με τις εντολές

¹⁰Εδώ μπορείτε να βάλετε και `rmin .lt. max_dist`.

```

E = sqrt(Ex*Ex+Ey*Ey)
dx = dl*Ex/E
dy = dl*Ey/E
x0 = x0 + dx
y0 = y0 + dy

```

Ο έλεγχος των παθολογικών καταστάσεων $E=0.0$ και $dx=dy=0.0$ εδώ γίνεται με την εντολή

```

if( E .le. 1.0e-10 ) goto 11 !stop calculation

```

όπου όταν το μέτρο του πεδίου γίνει πολύ μικρό σταματάει ο υπολογισμός βγαίνοντας από το βρόχο με την εντολή `goto 11` η οποία μας παραπέμπει στην εντολή με ετικέτα (label) 11 που εδώ είναι η

```

11      continue                !come here if E=0

```

η οποία απλώς συνεχίζει τη ροή του προγράμματος. Ομολογουμένως δεν είμαστε πολύ προσεκτικοί χάρην απλότητας και ο αναγνώστης καλείται να θεραπεύσει ορισμένες παθολογικές καταστάσεις σε σχετική άσκηση.

Με παρόμοιο τρόπο προγραμματίζουμε και την `epotline`. Ο σχετικός κώδικας παρατίθεται παρακάτω:

```

subroutine epotline(xin,yin,X,Y,Q,N)
implicit none
integer    N
real       X(N),Y(N),Q(N),xin,yin,x0,y0
real       step                !absolute value of step on equipot line
parameter(step=0.02)
real       max_dist            !max distance of equipot line
parameter(max_dist=20.0)
integer    i
real       r,dx,dy,dl
real       Ex,Ey,E

dl = step
x0 = xin
y0 = yin
dx = 0.0
dy = 0.0

```

```

      r = step                                !in order to start loop
      do while( r .gt. (0.9*dl) .and. r .lt. max_dist)
        print *,x0,y0
C      We evaluate the E-field at the midpoint: This reduces
C      systematic errors
        call efield(x0+0.5*dx,y0+0.5*dy,X,Y,Q,N,Ex,Ey)
        E = sqrt(Ex*Ex+Ey*Ey)
        if( E .le. 1.0e-10 ) goto 11 !stop calculation
        dx = dl*Ey/E
        dy = -dl*Ex/E
        x0 = x0 + dx
        y0 = y0 + dy
        r = sqrt((x0-xin)**2+(y0-yin)**2)
      enddo                                !do while()
11    continue                                !come here if E=0
      end

```

Οι διαφορές με το προηγούμενο πρόγραμμα είναι λίγες: Οι ισοδυναμικές καμπύλες είναι κλειστές, άρα τις διασχίζουμε μόνο κατά μία φορά και το κριτήριο τερματισμού του υπολογισμού είναι να φτάσουμε αρκετά κοντά στο αρχικό σημείο:

```

      do while( r .gt. (0.9*dl) .and. r .lt. max_dist)
        ...
      enddo                                !do while()

```

Ο παραπάνω βρόχος εκτελείται μέχρι η απόσταση από το αρχικό σημείο να γίνει μικρότερη από $0.9*dl$ ή να φύγουμε εκτός της περιοχής σχεδιασμού. Τα dx , dy υπολογίζονται σύμφωνα με την (5.6) :

```

      dx = dl*Ey/E
      dy = -dl*Ex/E

```

Η υπορουτίνα `efield` προγραμματίζεται απλά εφαρμόζοντας τους τύπους (5.2) ¹¹:

```

      subroutine efield(x0,y0,X,Y,Q,N,Ex,Ey)
      implicit none

```

¹¹Βελτιώστε το πρόγραμμα ώστε να λαμβάνει υπόψη του την περίπτωση $r_i = 0$.

```

integer  N
real     x0,y0,dx,dy,X(N),Y(N),Q(N),Ex,Ey
integer  i
real     r3,xi,yi

Ex = 0.0
Ey = 0.0
do i=1,N
  xi = x0-X(i)
  yi = y0-Y(i)
  r3 = (xi*xi+yi*yi)**(-1.5)
  Ex = Ex + Q(i)*xi*r3
  Ey = Ey + Q(i)*yi*r3
enddo
end

```

Τέλος, η υπορουτίνα `mdist` που χρησιμοποιούμε αρκετά στα παραπάνω υπολογίζει την ελάχιστη και μέγιστη απόσταση `rmin` και `rmax` από ένα δεδομένο σημείο (x_0, y_0) :

```

subroutine mdist(x0,y0,X,Y,N,rmin,rmax)
implicit none
integer  N
real     X(N),Y(N),x0,y0,rmin,rmax
integer  i
real     r

rmax = 0.0
rmin = 1000.0
do i = 1,N
  r = sqrt((x0-X(i))**2 + (y0-Y(i))**2)
  if(r.GT.rmax) rmax = r
  if(r.LT.rmin) rmin = r
enddo
end

```

Εδώ οι αρχικές τιμές των `rmin` και `rmax` πρέπει να οριστούν προσεκτικά ανάλογα με τα όρια σχεδίασης των δυν. γραμμών.

5.4 Το Πρόγραμμα - Σύνοψη

Στην παράγραφο αυτή παρατίθενται για διευκόλυνση του αναγνώστη ολόκληρα τα προγράμματα των δύο τελευταίων παραγράφων καθώς και συνοπτικές οδηγίες για τη μεταγλώττιση και ανάλυση των αποτελεσμάτων. Μπορείτε αν θέλετε πρώτα να αντιγράψετε τα προγράμματα στα αντίστοιχα αρχεία και να εκτελέσετε τις εντολές συλλογής και ανάλυσης των δεδομένων που δίνονται εδώ χωρίς εξήγηση και μετά να επιστρέψετε στις προηγούμενες παραγράφους για βαθύτερη κατανόηση των πεπραγμένων.

Πρώτα δίνουμε τα περιεχόμενα του αρχείου `ELines.f`:

```

C      *****
program Electric_Fields
C      *****
      implicit none
      integer      P                !max number of charges
      parameter(P=20)
      real          X(P),Y(P),Q(P)
      integer       N

      real          PI
      parameter(PI= 3.14159265359)
      integer       i,j,nd
      real          x0,y0,theta

C      -----  SET CHARGE DISTRIBUTION  -----
      print *, '# Enter number of charges:'
      read(5,*) N
      print *, '# N= ',N
      do i=1,N
         print *, '# Charge: ',i
         print *, '# Position and charge: (X,Y,Q):'
         read(5,*)  X(i),Y(i),Q(i)
         print *, '# (X,Y)= ', X(i),Y(i), ' Q= ',Q(i)
      enddo

C      -----  DRAWING LINES  -----
C      We draw 2*nd field lines around each charge
      nd      = 6

```

```

do i = 1,N
  do j = 1,(2*nd)
    theta = (PI/nd)*j
    x0 = X(i) + 0.1 * cos(theta)
    y0 = Y(i) + 0.1 * sin(theta)
    call eline(x0,y0,X,Y,Q,N)
  enddo
enddo

end

subroutine eline(xin,yin,X,Y,Q,N)
implicit none
real X(N),Y(N),Q(N),xin,yin,x0,y0
integer N
real step !absolute value of step on field line
parameter(step=0.01)
real max_dist !max distance of field line
parameter(max_dist=20.0)
integer i,direction
real rmin,rmax,r,dx,dy,dl
real Ex,Ey,E
do direction = -1,1,2 !direction= +/- 1
  dl = direction * step
  x0 = xin
  y0 = yin
  dx = 0.0
  dy = 0.0
  call mdist(x0,y0,X,Y,N,rmin,rmax)
  do while(rmin .gt. (2.0*step) .and. rmax .lt. max_dist)
    print *,x0,y0
C    We evaluate the E-field at the midpoint: This reduces
C    systematic errors
    call efield(x0+0.5*dx,y0+0.5*dy,X,Y,Q,N,Ex,Ey)
    E = sqrt(Ex*Ex+Ey*Ey)
    if( E .le. 1.0e-10 ) goto 11 !stop calculation
    dx = dl*Ex/E

```

```

        dy = dl*Ey/E
        x0 = x0 + dx
        y0 = y0 + dy
        call mdist(x0,y0,X,Y,N,rmin,rmax)
    enddo                                !do while()
11  continue                            !come here if E=0
enddo                                !do direction = -1,1,2
end

```

```

subroutine efield(x0,y0,X,Y,Q,N,Ex,Ey)
implicit none
integer N
real    x0,y0,dx,dy,X(N),Y(N),Q(N),Ex,Ey
integer i
real    r3,xi,yi

Ex = 0.0
Ey = 0.0
do i= 1,N
    xi = x0-X(i)
    yi = y0-Y(i)
C  Exercise: Improve code so that xi*xi+yi*yi=0 is taken care of
    r3 = (xi*xi+yi*yi)**(-1.5)
    Ex = Ex + Q(i)*xi*r3
    Ey = Ey + Q(i)*yi*r3
enddo

end

```

```

subroutine mdist(x0,y0,X,Y,N,rmin,rmax)
implicit none
integer N
real    X(N),Y(N),x0,y0,rmin,rmax
integer i
real    r

```

```

rmax = 0.0
rmin = 1000.0
do i = 1,N
  r = sqrt((x0-X(i))**2 + (y0-Y(i))**2)
  if(r.GT.rmax) rmax = r
  if(r.LT.rmin) rmin = r
enddo

end

```

Στη συνέχεια δίνουμε τα περιεχόμενα του αρχείου EPotential.f:

```

C *****
program Electric_Potential
C *****
implicit none
integer P !max number of charges
parameter(P=20)
real X(P),Y(P),Q(P)
integer N

real PI
parameter(PI= 3.14159265359)
integer i,j,nd
real x0,y0,rmin,rmax,L

print *, '# Enter number of charges:'
read(5,*) N
print *, '# N= ',N
do i=1,N
  print *, '# Charge: ',i
  print *, '# Position and charge: (X,Y,Q):'
  read(5,*) X(i),Y(i),Q(i)
  print *, '# (X,Y)= ', X(i),Y(i), ' Q= ',Q(i)
enddo

C ----- DRAWING LINES -----
C We draw lines passing through an equally

```

```

C      spaced lattice of N=(2*nd+1)x(2*nd+1) points
C      in the square -L<= x <= L, -L<= y <= L.
      nd      = 4
      L       = 1.0
      do i = -nd,nd
        do j = -nd,nd
          x0 = i*(L/nd)
          y0 = j*(L/nd)
          print *, '# @ ', i, j, L/nd, x0, y0
          call mdist(x0, y0, X, Y, N, rmin, rmax)
C      we avoid getting too close to a charge:
          if(rmin .gt. L/(nd*10) )
*          call epotline(x0, y0, X, Y, Q, N)
        enddo
      enddo
      enddo
      end

      subroutine epotline(xin, yin, X, Y, Q, N)
      implicit none
      integer      N
      real          X(N), Y(N), Q(N), xin, yin, x0, y0
      real          step                !absolute value of step on equipot line
      parameter(step=0.02)
      real          max_dist            !max distance of equipot line
      parameter(max_dist=20.0)
      integer      i
      real          r, dx, dy, dl
      real          Ex, Ey, E

      dl = step
      x0 = xin
      y0 = yin
      dx = 0.0
      dy = 0.0
      r = step                !in order to start loop
      do while( r .gt. (0.9*dl) .and. r .lt. max_dist)
        print *, x0, y0

```

```

C      We evaluate the E-field at the midpoint: This reduces
C      systematic errors
      call efield(x0+0.5*dx,y0+0.5*dy,X,Y,Q,N,Ex,Ey)
      E = sqrt(Ex*Ex+Ey*Ey)
      if( E .le. 1.0e-10 ) goto 11 !stop calculation
      dx = dl*Ey/E
      dy = -dl*Ex/E
      x0 = x0 + dx
      y0 = y0 + dy
      r = sqrt((x0-xin)**2+(y0-yin)**2)
enddo                                !do while()
11  continue                          !come here if E=0
end

subroutine efield(x0,y0,X,Y,Q,N,Ex,Ey)
implicit none
integer  N
real     x0,y0,dx,dy,X(N),Y(N),Q(N),Ex,Ey
integer  i
real     r3,xi,yi

Ex = 0.0
Ey = 0.0
do i=1,N
  xi = x0-X(i)
  yi = y0-Y(i)
C      Exercise: Improve code so that xi*xi+yi*yi=0 is taken care of
  r3 = (xi*xi+yi*yi)**(-1.5)
  Ex = Ex + Q(i)*xi*r3
  Ey = Ey + Q(i)*yi*r3
enddo
end

subroutine mdist(x0,y0,X,Y,N,rmin,rmax)
implicit none
integer  N

```

```

real      X(N),Y(N),x0,y0,rmin,rmax
integer   i
real      r

rmax = 0.0
rmin = 1000.0
do i = 1,N
  r = sqrt((x0-X(i))**2 + (y0-Y(i))**2)
  if(r.GT.rmax) rmax = r
  if(r.LT.rmin) rmin = r
enddo

end

```

Τέλος υπενθυμίζουμε στον αναγνώστη πώς να τα τρέξει και να δει τα αποτελέσματα. Μεταγλωττίζουμε το πρόγραμμα με την εντολή

```

> f77 ELines.f      -o el
> f77 EPotential.f -o ep

```

Στη συνέχεια γράφουμε σε ένα αρχείο `Input` τα δεδομένα της κατανομής των φορτίων. Για παράδειγμα:

```

4           N: Number of charges
 1  1  -1    (X,Y,Q): Position and charge
-1  1   1    (X,Y,Q): Position and charge
 1 -1   1    (X,Y,Q): Position and charge
-1 -1  -1    (X,Y,Q): Position and charge

```

Τα αποτελέσματα τα παίρνουμε με τις εντολές:

```

> ./el < Input > el.dat
> ./ep < Input > ep.dat
> gnuplot
gnuplot> plot "el.dat" with dots
gnuplot> plot "ep.dat" with dots

```

Καλή ... διασκέδαση!

5.5 Ηλεκτροστατικό Πεδίο στο Κενό

Η παράγραφος αυτή βασίζεται στο κεφ. 10 της [;].

Θεωρούμε ηλεκτρικό πεδίο το οποίο είναι ανεξάρτητο του χρόνου σε περιοχή του χώρου άδεια από ηλεκτρικά φορτία. Οι εξισώσεις του Maxwell στην περίπτωση αυτή είναι η εξίσωση Gauss

$$\vec{\nabla} \cdot \vec{E}(x, y, z) = \frac{\partial E_x}{\partial x} + \frac{\partial E_y}{\partial y} + \frac{\partial E_z}{\partial z} = 0, \quad (5.8)$$

μαζί με την εξίσωση που ορίζει το ηλεκτροστατικό δυναμικό¹²

$$\vec{E}(x, y, z) = -\vec{\nabla}V(x, y, z). \quad (5.9)$$

Οι εξισώσεις (5.8) και (5.9) μας δίνουν μία εξίσωση Laplace για τη $V(x, y, z)$:

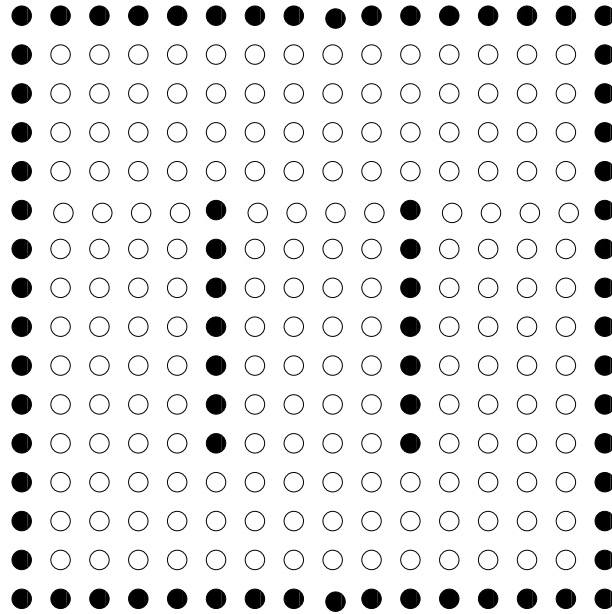
$$\nabla^2 V(x, y, z) = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = 0. \quad (5.10)$$

Η λύση της παραπάνω εξίσωσης είναι ένα πρόβλημα συνοριακών συνθηκών: Ζητούμε το δυναμικό $V(x, y, z)$ σε μία περιοχή του χώρου $\mathcal{S}$ η οποία περιβάλλεται από το όριο της, μία κλειστή επιφάνεια $\partial\mathcal{S}$. Όταν το δυναμικό είναι γνωστό πάνω στην $\partial\mathcal{S}$ η λύση της (5.10) είναι μοναδική και το δυναμικό και κατ'επέκταση το ηλεκτρικό πεδίο προσδιορίζεται παντού στην $\mathcal{S}$.

Για απλότητα παρακάτω θεωρούμε το πρόβλημα ανηγμένο στο επίπεδο, οπότε $V = V(x, y)$. Στην περίπτωση αυτή ο τελευταίος όρος στην (5.10) απουσιάζει, η $\mathcal{S}$ είναι συμπαγές τμήμα του επιπέδου και η $\partial\mathcal{S}$ είναι μια κλειστή καμπύλη.

Για να μελετήσουμε το πρόβλημα αριθμητικά, προσεγγίζουμε το χώρο με ένα τετραγωνικό πλέγμα με το δυναμικό να ορίζεται μόνο σε N πλεγματικές θέσεις. Η προσέγγιση της συνεχούς λύσης επιτυγχάνεται όταν η διακριτικοποίηση αυτή εκλεπτύνεται παίρνοντας τον αριθμό των πλεγματικών σημείων στο όριο $N \rightarrow \infty$. Η καμπύλη $\partial\mathcal{S}$ προσεγγίζεται από τις πλεγματικές θέσεις που αποτελούν το όριο του πλέγματος και πιθανώς από άλλα σημεία στο εσωτερικό στα οποία η τιμή του δυναμικού είναι δεδομένη. Το φυσικό σύστημα που αντιστοιχεί στο μοντέλο μας είναι μια διάταξη από αγωγούς των οποίων το δυναμικό διατηρείται σταθερό και ζητείται το ηλεκτρικό πεδίο που δημιουργείται στον περιβάλλοντα χώρο τους. Μία τέτοια διάταξη φαίνεται στο σχήμα 5.6.

¹²Ισοδύναμη με την εξίσωση $\vec{\nabla} \times \vec{E} = 0$.



Σχήμα 5.6: Πλέγμα που αντιστοιχεί στη διάταξη δύο παράλληλων επίπεδων αγωγών μέσα σε γειωμένο μεταλλικό κουτί. Οι μαύρες πλεγματικές θέσεις αντιστοιχούν σε σημεία σταθερού δυναμικού (όχι κατ' ανάγκη το ίδιο), ενώ οι άσπρες σε σημεία του κενού χώρου.

Για να βούμε την εξίσωση πεπερασμένων διαφορών που θα προσεγγίζει την Εξίσωση (5.10), θεωρούμε την ανάπτυξη κατά Taylor:

$$\begin{aligned}
 V(x + \delta x, y) &= V(x, y) + \frac{\partial V}{\partial x} \delta x + \frac{1}{2} \frac{\partial^2 V}{\partial x^2} (\delta x)^2 + \dots \\
 V(x - \delta x, y) &= V(x, y) - \frac{\partial V}{\partial x} \delta x + \frac{1}{2} \frac{\partial^2 V}{\partial x^2} (\delta x)^2 + \dots \\
 V(x, y + \delta y) &= V(x, y) + \frac{\partial V}{\partial y} \delta y + \frac{1}{2} \frac{\partial^2 V}{\partial y^2} (\delta y)^2 + \dots \\
 V(x, y - \delta y) &= V(x, y) - \frac{\partial V}{\partial y} \delta y + \frac{1}{2} \frac{\partial^2 V}{\partial y^2} (\delta y)^2 + \dots
 \end{aligned}$$

Παίρνοντας το άθροισμα και των δύο μελών της παραπάνω εξίσωσης, $\delta x = \delta y$

και αγνοώντας τους όρους ... παίρνουμε

$$\begin{aligned} & V(x + \delta x, y) + V(x - \delta x, y) + V(x, y + \delta y) + V(x, y - \delta y) \\ &= 4V(x, y) + (\delta x)^2 \left(\frac{1}{2} \frac{\partial^2 V}{\partial x^2} + \frac{1}{2} \frac{\partial^2 V}{\partial y^2} \right) + \dots \\ &\approx 4V(x, y). \end{aligned} \quad (5.11)$$

Θεωρώντας τις συντεταγμένες των πλεγματοειδών σημείων να δίνονται από ακέραιους (i, j) η παραπάνω εξίσωση γίνεται:

$$V(i, j) = \frac{1}{4}(V(i - 1, j) + V(i + 1, j) + V(i, j - 1) + V(i, j + 1)), \quad (5.12)$$

η οποία ερμηνεύεται με απλό τρόπο: Το δυναμικό στη θέση (i, j) είναι απλά ο μέσος όρος του δυναμικού των πλησιεστέρων γειτόνων του¹³. Ο αλγόριθμος που θα ακολουθήσουμε εντάσσεται στη γενική κατηγορία μεθόδων 'χαλάρωσης' (relaxation methods) και τα βασικά του βήματα είναι:

1. Καθορίζεται το μέγεθος τετραγωνικού πλέγματος με L πλεγματοειδείς θέσεις σε κάθε πλευρά.
2. Καθορίζονται οι πλεγματοειδείς θέσεις που έχουν καθορισμένο δυναμικό και δίνεται η τιμή του δυναμικού στις θέσεις αυτές.
3. Καθορίζεται αυθαίρετη τιμή για το δυναμικό στις υπόλοιπες θέσεις. Μια καλή επιλογή θα δώσει γρηγορότερη σύγκλιση προς τη σωστή συνάρτηση $V(x, y)$. Μια κακή επιλογή αργή σύγκλιση ή/και σύγκλιση προς λανθασμένη λύση.
4. Σε διαδοχικά επαναλαμβανόμενα 'περάσματα' (sweeps) του πλέγματος επικεπτόμαστε κάθε πλεγματοειδή θέση και υπολογίζουμε το δυναμικό V_{av} από τη σχέση (5.12) και ορίζουμε αυτή να είναι η νέα τιμή του δυναμικού στη θέση αυτή.
5. Η διαδικασία σταματάει όταν σε δύο διαδοχικά περάσματα η μεταβολή του δυναμικού είναι ικανοποιητικά μικρή. Υποθέτουμε δηλ. ότι έχουμε πρακτικά επιτύχει τη σύγκλιση στη σωστή λύση.

¹³Αυτό συνδέει το πρόβλημα άμεσα με προβλήματα διάχυσης!

Η προσεκτική χρήση του παραπάνω αλγόριθμου απαιτεί να τον μελετήσουμε με διαφορετικά κριτήρια 'ικανοποιητικά μικρής' διαφοράς και διαφορετικές αρχικές συνθήκες.

Παρακάτω παραθέτουμε πρόγραμμα που χρησιμοποιεί τον παραπάνω αλγόριθμο για τον υπολογισμό του ηλεκτροστατικού δυναμικού δύο παράλληλων επίπεδων μεταλλικών αγωγών που βρίσκονται μέσα σε ένα γειωμένο τετράγωνο μεταλλικό αγωγίμο κουτί. Το πλέγμα φαίνεται στο Σχήμα 5.6, όπου οι μαύρες κουκίδες αναπαριστούν τους αγωγούς. Το κουτί έχει δυναμικό $V = 0$ ενώ οι αγωγοί δυναμικό V_1 και V_2 αντίστοιχα. Ο χρήστης στην είσοδο δίνει τις τιμές V_1 και V_2 καθώς, το μέγεθος L του πλέγματος και την ακρίβεια σύγκλισης που επιθυμεί. Η τελευταία καθορίζεται ποσοτικά από ένα μικρό αριθμό ϵ . Η μέγιστη μεταβολή στο δυναμικό ανάμεσα σε δύο διαφορετικά περάσματα του πλέγματος θα πρέπει να είναι μικρότερη από ϵ για επίτευξη σύγκλισης της μεθόδου.

Η δομή των δεδομένων είναι απλή. Χρειαζόμαστε ένα πραγματικό array $V(L,L)$ με αποθηκευμένες τις τιμές του δυναμικού και ένα logical array $isConductor(L,L)$ το οποίο καθορίζει αν μια πλεγματική θέση έχει καθορισμένο δυναμικό ('αγωγός' = .TRUE.) ή όχι ('κενός χώρος' = .FALSE.).

Το κυρίως πρόγραμμα αφού διαβάσει τα δεδομένα από το χρήστη καλεί τρεις υπορουτίνες:

1. `initialize_lattice(V,isConductor,L,V1,V2):`

Τα δεδομένα εισόδου της ρουτίνας είναι το δυναμικό V_1 του αριστερού αγωγού και το δυναμικό V_2 του δεξιού αγωγού, καθώς και το μέγεθος του πλέγματος L . Οι μεταβλητές εξόδου είναι τα arrays $V(L,L)$ και $isConductor(L,L)$ τα οποία στην έξοδο έχουν τις αρχικές τιμές: V_1 στον αριστερό αγωγό, V_2 στο δεξί, 0 στο κουτί καθώς και την αυθαίρετη τιμή 0 για τις υπόλοιπες πλεγματικές θέσεις. Η γεωμετρία της διάταξης είναι κωδικοποιημένη και ο χρήστης που θα θέλει να τη μεταβάλλει (λ.χ. μεταβολή απόστασης αγωγών, προσθήκη/αφαίρεση αγωγού κλπ) θα πρέπει να επέμβει στον κώδικα και να κάνει τις απαραίτητες μετατροπές.

2. `laplace(V,isConductor,L,epsilon):`

Η καρδιά του προγράμματος. Στην είσοδο παρέχουμε τα αρχικοποιημένα arrays $V(L,L)$, $isConductor(L,L)$ και τη διάστασή τους L , καθώς και την επιθυμητή ακρίβεια σύγκλισης $epsilon$. Στην έξοδο παίρνουμε την τελική λύση $V(L,L)$ για περεταίρω επεξεργασία. Στη ρουτίνα αυτή υπολογίζουμε τη μέση τιμή του δυναμικού V_{av} των πλησιέστερων γειτόνων

[illegible]


```
C      real *8 V(L,L): array to be printed
C      OUTPUT:
C      no output
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    subroutine print_results(V,L)
    implicit none
    integer L
    real *8 V(L,L)

    integer i,j

    open(unit=11,file="data")
    do i=1,L
        do j =1,L
            write(11,*)i,j,V(i,j)
        enddo
        write (11,*)'' !print empty line for gnuplot, separate isolines
    enddo

end
```

5.6 Αποτελέσματα

Το παραπάνω πρόγραμμα το αποθηκεύουμε στο αρχείο LaplaceEq.f. Η μεταγλώττιση του προγράμματος γίνεται με τον μεταγλωττιστή f77¹⁵. Το εκτελέσιμο αρχείο το ονομάζουμε lf και το δημιουργούμε με την επιλογή -o lf του μεταγλωττιστή. Παρακάτω φαίνεται πώς τρέχει το πρόγραμμα για L=31 από τη γραμμή εντολών όταν ο χρήστης καθορίσει V1= 100.0 V2= -100.0 epsilon= 0.01. Οι εντολές στο >f77... και >./lf δίνονται στη γραμμή εντολών του φλοιού. Μετα φαίνεται το input/output του προγράμματος:

```
> f77 LaplaceEq.f -o lf
> ./lf
Enter V1,V2:
100 -100
```

¹⁵Στο σύστημα Linux αυτός είναι ο GNU fortran compiler που διατίθεται ελεύθερα σε πολλά λειτουργικά συστήματα.

```

Enter epsilon:
0.01
Starting Laplace:
Grid Size= 31
Conductors set at V1= 100. V2= -100.
Relaxing with accuracy epsilon= 0.01
1 err= 33.3333333
2 err= 14.8148148
3 err= 9.87654321
.....
110 err= 0.0106860904
111 err= 0.0101182476
112 err= 0.00958048937

```

Το πρόγραμμα κάνει 112 περάσματα στο πλέγμα μέχρι το μέγιστο σφάλμα να γίνει $0.00958048937 < 0.01$. Τα αποτελέσματα αποθηκεύονται στο αρχείο `data`. Για να δούμε τα αποτελέσματα χρησιμοποιούμε το πρόγραμμα `gnuplot`. Παρακάτω η εντολή `>gnuplot` δίνεται στο φλοιό, ενώ όταν το prompt είναι `gnuplot>` η εντολή δίνεται στη γραμμή εντολών του `gnuplot`:

```

> gnuplot
gnuplot> set pm3d
gnuplot> set hidden3d
gnuplot> set size ratio 1
gnuplot> splot "data" with lines

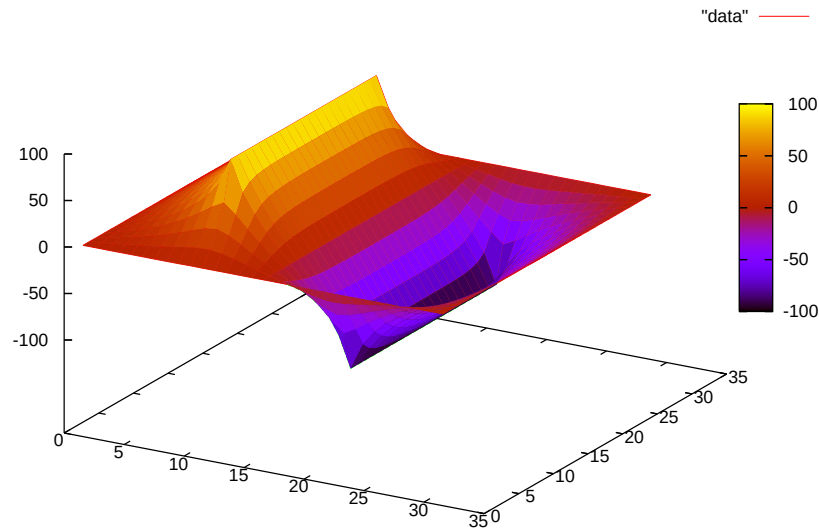
```

Τα αποτελέσματα φαίνονται στο Σχήμα 5.7

5.7 Εξίσωση Poisson

Τέλος ας αναφέρουμε περιληπτικά τη λύση του προβλήματος που μελετήσαμε στις προηγούμενες παραγράφους όταν υπάρχει στην περιοχή που μελετάμε κατανομή στατικού ηλεκτρικού φορτίου που δίνεται από την πυκνότητα φορτίου $\rho(\vec{r})$. Στην περίπτωση αυτή η εξίσωση Laplace για το δυναμικό γίνεται εξίσωση Poisson:

$$\nabla^2 V = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = -4\pi\rho(x, y, z) \quad (5.13)$$



Σχήμα 5.7: Η λύση της εξίσωσης (5.10) από το πρόγραμμα LaplaceEq.f για $L=31$, $V_1=100$, $V_2=-100$, $\epsilon=0.01$.

Η μορφή της εξίσωσης στο επίπεδο πλέγμα που μελετήσαμε γίνεται

$$V(i, j) = \frac{1}{4}(V(i-1, j) + V(i+1, j) + V(i, j-1) + V(i, j+1) + \rho(i, j)), \quad (5.14)$$

και είναι πάρα πολύ απλό να μετατρέψουμε τον κώδικα της προηγούμενης παραγράφου έτσι ώστε να μελετήσουμε κατανομές φορτίου που επιθυμούμε.

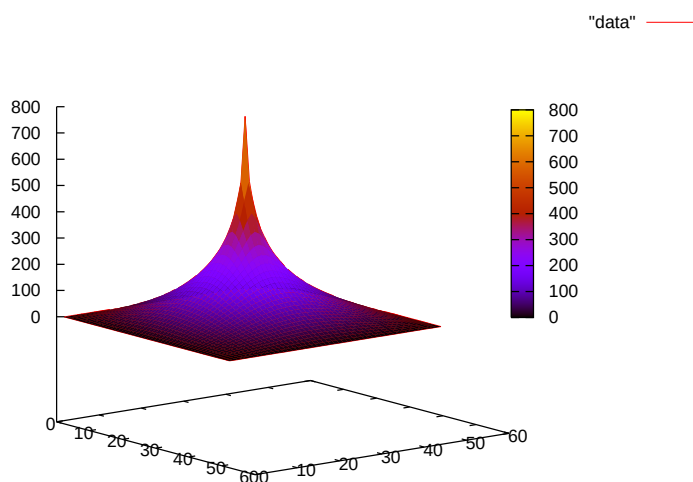
Παρακάτω παραθέτουμε το πρόγραμμα PoissonEq.f που λύνει την (5.14) για ομοιόμορφη κατανομή του φορτίου (Σχήμα 5.10). Ο αναγνώστης καλείται να αναπαράγει το σχήμα αυτό, καθώς και τα 5.8, 5.9.

```

program laplace_sq
implicit none

integer P
parameter(P=51)
logical isConductor(P,P)
real *8 V(P,P),rho(P,P),Q

```



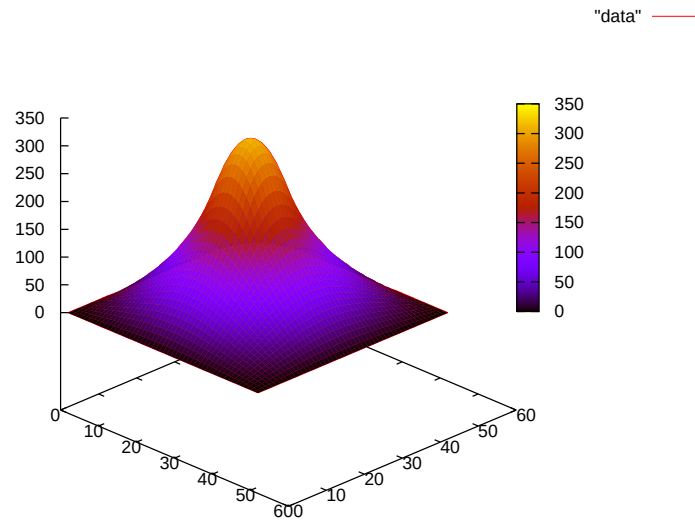
Σχήμα 5.8: Η λύση της εξίσωσης (5.13) από το πρόγραμμα Poisson.f για $L=51$, $V=0$ στο σύνορο και το φορτίο $Q=1000$ συγκεντρωμένο σε ένα σημείο.

```

real *8 V1,V2,V3,V4,epsilon
integer L

L = P
print *, 'Enter V1,V2,V3,V4:'
read(5,*) V1,V2,V3,V4
print *, 'Enter total charge Q:'
read(5,*) Q
print *, 'Enter epsilon:'
read(5,*) epsilon
print *, 'Starting Laplace:'
print *, 'Grid Size= ',L
print *, 'Boundaries set at V1= ',V1,' V2= ',V2,' V3= ',V3,' V4= ',
*      V4
print *, 'Relaxing with accuracy epsilon= ',epsilon

```



Σχήμα 5.9: Η λύση της εξίσωσης (5.13) από το πρόγραμμα Poisson.f για $L=51$, $V=0$ στο σύνορο και το φορτίο $Q=1000$ κατανεμημένο ομοιόμορφα σε ένα μικρό τετράγωνο πλάτους 10 πλεγματικών σημείων.

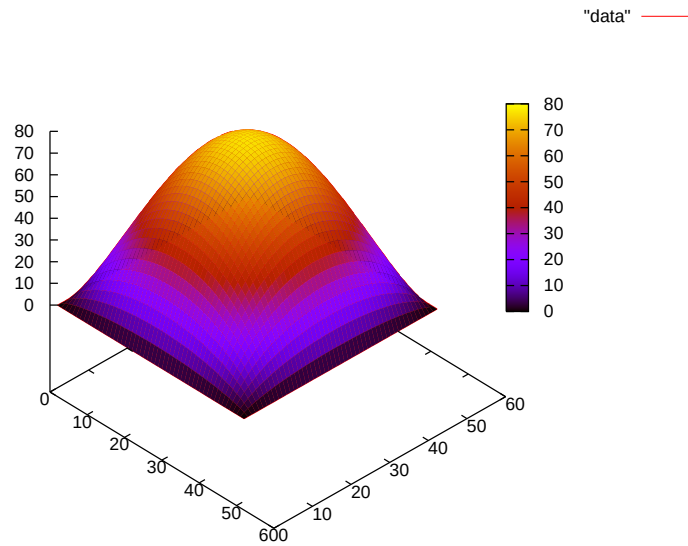
```

call initialize_lattice(V,isConductor,rho,L,V1,V2,V3,V4,Q)
call laplace(V,isConductor,rho,L,epsilon)
call print_results(V,L)

end

subroutine initialize_lattice(V,isConductor,rho,L,V1,V2,V3,V4,Q)
implicit none
integer L
logical isConductor(L,L)
real *8 V(L,L),rho(L,L)
real *8 V1,V2,V3,V4,Q
real *8 Area
integer i,j

```



Σχήμα 5.10: Η λύση της εξίσωσης (5.13) από το πρόγραμμα Poisson.f για $L=51$, $V=0$ στο σύνορο και το φορτίο $Q=1000$ κατανεμημένο ομοιόμορφα σε όλες τις εσωτερικές πλεγματικές θέσεις.

```

integer L1,L2

C      Initialize to 0 and .FALSE.
do j=1,L
  do i=1,L
    V      (i,j) = 0.0D0
    rho    (i,j) = 0.0D0
    isConductor(i,j) = .FALSE.
  enddo
enddo

C      We set the boundary to be a conductor:
do i=1,L
  isConductor(1,i) = .TRUE.
  isConductor(i,1) = .TRUE.

```

```

        isConductor(L,i) = .TRUE.
        isConductor(i,L) = .TRUE.
        V          (1,i) =  V1
        V          (i,L) =  V2
        V          (L,i) =  V3
        V          (i,1) =  V4
    enddo

C      We set the points with non-zero charge
C      A uniform distribution at the interior
    L1 = 2
    L2 = L-1
    Area = (L2-L1+1)*(L2-L1+1)
    do j=L1,L2
        do i=L1,L2
            rho(i,j) = Q/Area
        enddo
    enddo

end

subroutine laplace(V,isConductor,rho,L,epsilon)
implicit none
integer L
logical isConductor(L,L)
real *8 V(L,L),rho(L,L)
real *8 epsilon

integer i,j,icount
real *8 Vav,error,dV

icount = 0
do while (.TRUE.)
    error = 0.0D0
    do j=2,L-1
        do i=2,L-1
C      We change the voltage only for non conductors:
            if( .NOT. isConductor(i,j))then

```

```

      Vav = ( V(i-1,j)+V(i+1,j)+V(i,j+1)+V(i,j-1)+rho(i,j)) * 0.25D0
      dV = DABS(V(i,j)-Vav)
      if(error .LT. dV ) error = dV !maximum error
      V(i,j) = Vav
    endif
  enddo
enddo
icount = icount + 1
print *,icount,' err= ',error

  if( error .LT. epsilon) return
enddo

end

subroutine print_results(V,L)
implicit none
integer L
real *8 V(L,L)

integer i,j

open(unit=11,file="data")
do i=1,L
  do j =1,L
    write(11,*)i,j,V(i,j)
  enddo
  write (11,*)'' !for gnuplot, separate isolines
enddo

end

```


5.8 Ασκήσεις

- 5.1 Φτιάξτε τις εικόνες των δυναμικών και ισοδυναμικών γραμμών που βρίσκονται στην παράγραφο 5.2.
- 5.2 Στις προηγούμενες κατανομές φορτίων κάνετε όλα τα φορτία θετικά. Φτιάξτε τις εικόνες των δυναμικών και ισοδυναμικών γραμμών. Μετά δώστε στα φορτία αυτά διαφορετική τιμή και επαναλάβετε.
- 5.3 Γιατί το πρόγραμμα `ELines.f` κολλάει σε κατανομή ίσων φορτίων πάνω σε κορυφές τετραγώνων; Πώς διορθώνεται η παθολογία αυτή;
- 5.4 Μεταβάλλετε το πρόγραμμα `ELines.f` ώστε ο αριθμός των δυναμικών γραμμών που ξεκινάνε από ένα φορτίο να είναι ανάλογος του μέτρου του.
- 5.5 Βελτιώστε το πρόγραμμα `EPotential.f` έτσι ώστε οι ισοδυναμικές καμπύλες να ζωγραφίζονται με πυκνότητα ανάλογη του ηλεκτρικού πεδίου. Υπόδειξη:
- (α') Γράψτε υπορουτίνα που να υπολογίζει το δυναμικό $V(x, y)$ στο σημείο (x, y) .
 - (β') Από κάθε φορτίο παίρνουμε μια ευθεία στην ακτινική διεύθυνση και υπολογίζουμε το δυναμικό πάνω σε αυτή σε σημεία που απέχουν μικρή απόσταση Δl .
 - (γ') Υπολογίζουμε τη μέγιστη/ελάχιστη τιμή του δυναμικού V_{max}/V_{min} και από εκεί τις τιμές του δυναμικού πάνω στις ισοδυναμικές καμπύλες που θα σχεδιάσουμε. Αν λ.χ. αποφασίσετε να σχεδιάζετε 5 ισοδ. καμπύλες, πάρτε $\delta V = (V_{max} - V_{min})/4$ και $V_i = V_{min} + i\delta V$ $i = 1, \dots, 4$.
 - (δ') Επαναλάβετε το δεύτερο βήμα. Όταν το δυναμικό παίρνει τιμή (σχεδόν) ίση με μια από αυτές που διαλέξατε στο τρίτο βήμα, σχεδιάστε την δυναμική γραμμή.
- 5.6 Μελετήστε το ηλεκτρικό πεδίο που προκύπτει από το πρόγραμμα `LaplaceEq.f` για:
- (α') $L=31, V1=100, V2=100$
 - (β') $L=31, V1=100, V2=0$

και φτιάξτε τα αντίστοιχα σχήματα.

- 5.7 Μελετήστε το ηλεκτρικό πεδίο που προκύπτει από το πρόγραμμα LaplaceEq.f για:

(α') $V_1=100$, $V_2=100$

(β') $V_1=100$, $V_2=100$

(γ') $V_1=100$, $V_2=0$

για $L=31, 61, 121, 241, 501$ και φτιάξτε τα αντίστοιχα σχήματα. Μεταβάλλετε την ακρίβεια προσδιορισμού της λύσεις $\epsilonpsilon=0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001$. Πως εξαρτάται ο αριθμός περασμάτων N του πλέγματος από το $\epsilonpsilon$; Φτιάξτε γραφική παράσταση $N(\epsilonpsilon)$ στην οποία θα τοποθετήσετε τις καμπύλες $N(\epsilonpsilon)$ για τα διαφορετικά L που χρησιμοποιήσατε.

- 5.8 Μελετήστε το ηλεκτρικό πεδίο τετράγωνου αγωγού του οποίου κάθε πλευρά έχει διαφορετικό δυναμικό V_1, V_2, V_3, V_4 . Επαναλάβετε τη μελέτη που κάνατε στην προηγούμενη άσκηση για τις περιπτώσεις:

(α') $V_1=10$, $V_2=5$, $V_3=10$, $V_4= 5$

(β') $V_1=10$, $V_2=0$, $V_3=0$, $V_4= -10$

(γ') $V_1=10$, $V_2=0$, $V_3=0$, $V_4= 0$

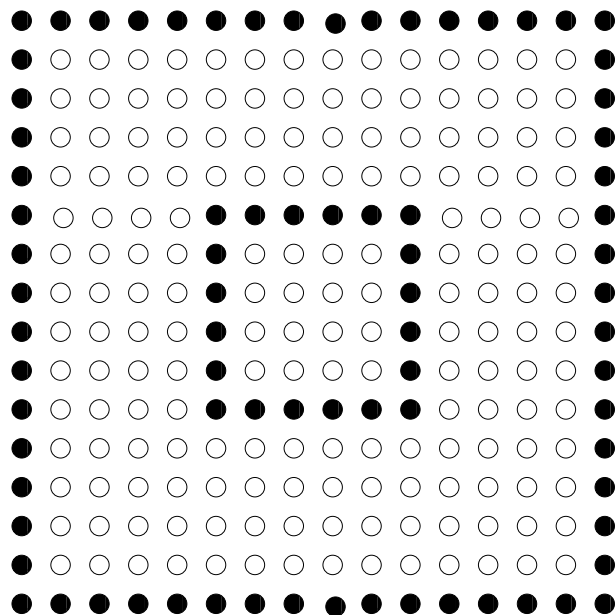
- 5.9 Μελετήστε το ηλεκτρικό πεδίο συστήματος τετράγωνων αγωγών που βρίσκεται ο ένας μέσα στον άλλο όπως στο Σχήμα 5.11. Ο κάθε αγωγός έχει πλευρά L_1, L_2 και βρίσκεται σε δυναμικό V_1, V_2 αντίστοιχα. Πάρτε $L_2= L_1/5$ και επαναλάβετε την μελέτη της προηγούμενης άσκησης για $V_1=10$, $V_2=-10$ και $L_1= 25, 50, 100, 200$.

- 5.10 Υπολογίστε αριθμητικά τη χωρητικότητα $C = Q/V$ του συστήματος της προηγούμενης άσκησης όταν $V_1 = V$, $V_2 = -V$. Για να υπολογίσετε το φορτίο Q μπορείτε να υπολογίσετε την επιφανειακή πυκνότητα φορτίου σ από τη σχέση

$$\sigma = \frac{E_n}{4\pi},$$

όπου E_n η κάθετη συνιστώσα του ηλεκτρικού πεδίου στην επιφάνεια την οποία θα προσεγγίσετε από τη σχέση

$$E_n = -\frac{\delta V}{\delta r},$$



Σχήμα 5.11: Η διάταξη αγωγών που ζητήται να μελετηθεί το ηλεκτρικό πεδίο στην Άσκηση 5.9.

όπου δV είναι η διαφορά δυναμικού μεταξύ ενός σημείου του αγωγού και του σημείου που είναι πλησιέστερος γείτονάς του. Με τον τρόπο αυτό υπολογίστε το συνολικό φορτίο και κάθε αγωγό. Αν αυτά είναι αντίθετα και κατ' απόλυτη τιμή ίσα με Q τότε προσδιορίζεται η χωρητικότητα C .

(α') Κάνετε τον παραπάνω υπολογισμό για $L1=25, 75$.

(β') Μεταθέστε τον κεντρικό αγωγό 1 πλεγματική θέση αριστερά. Είναι τα δύο φορτία ίσα τώρα; Δοκιμάστε να μελετήσετε την απόκλιση αυτή καθώς μετατοπίζουμε τον κεντρικό αγωγό προς τα αριστερά.

5.11 Στο σύστημα της προηγούμενης άσκησης μελετήστε τη συνάρτηση $Q(V)$. Βεβαιωθείτε ότι η χωρητικότητα είναι ανεξάρτητη της διαφοράς δυναμικού. Πάρτε $L1=25, 50$, $V1 = -V2 = 1, 2, 5, 10, 15, 20, 25$. Εξετάστε την εξάρτηση των αποτελεσμάτων σας για διαφορετικές τιμές της ακρίβειας ϵ . Είναι δυνατόν για αρκετά μικρό ϵ να πάρετε $C = \text{σταθ.}$;

5.12 Αναπαράγετε τα Σχήματα 5.8, 5.9 και 5.10. Στην πρώτη περίπτωση συγ-

κρίνετε τη λύση που θα πάρετε με αυτή που δίνεται από το δυναμικό σημειακού ηλεκτρικού φορτίου. Φτιάξτε ανάλογα σχήματα για μικρότερες τιμές του φορτίου και βάζοντας διαφορετικές τιμές του δυναμικού στο σύνορο του πλέγματος.