Αριθμητική Επίλυση Συνήθων Διαφορικών Εξισώσεων με τη Μέθοδο των Σειρών και Εφαρμογή στο Γενικό Πρόβλημα των Τριών Σωμάτων

Numerical Solution of Ordinary Differential Equations Using Power Series Methods and application to the General Three–Body Problem

Ευθύμιος Κότσιαλος

September 29, 2005

Διπλωματική Εργασία

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Σ.Θ.Ε. – Τμήμα Φυσικής
ΠΜΣ Υπολογιστικής Φυσικής

Επιβλέπων:
Γ. Βουγιατζής, Λέκτορας, Τμήμα Φυσικής
Τομέας Αστρονομίας, Αστροφυσικής και Μηχανικής.

Masters Thesis

Aristotle University of Thessaloniki
Physics Department
Computational Physics Masters Programme

Thesis Supervisor:
G. Voyatzis, Lecturer, Physics Dept.,
Astronomy, Astrophysics & Mechanics Sector.

Στον Πατέρα μου

Δημήτριο Κότσιαλο

1 Εισαγωγή & Καθορισμός του Προβλήματος

1.1 Εισαγωγή.

Η Αριθμητική Ανάλυση, σαν αυτόνομος επιστημονικός κλάδος είναι από τους τελευταίους που αναπτύχθηκαν στις Θετικές Επιστήμες και θεωρείται απόλυτα συνυφασμένη με την εμφάνιση του Ηλεκτρονικού Υπολογιστή, αν και πολλοί ερευνητές, όχι άδικα ίσως, τοποθετούν τις απαρχές της στις αρχές του $20^{ου}$ αιώνα, ή και παλαιότερα [Brezinski & Wuytack, 2001]. Η πλήρης ωρίμανσή της κατέστη δυνατή με την διάδοση των υπολογιστικών συστημάτων και την ολοένα και συμφερότερη σχέση υπολογιστικής ισχύος προς αξία υλικού εξοπλισμού (hardware) και προγραμμάτων (software). Έχοντας διαθέσιμη περισσότερη υπολογιστική δύναμη, οι μέχρι πρότινος αποδεκτές λύσεις δεν ικανοποιούν πλέον και απαιτείται η εξεύρεση νέων τεχνικών και αλγορίθμων για την επιτυχέστερη αντιμετώπιση των προβλημάτων που ανακύπτουν. Πέρα από αυτό, νέα, συνεχώς δυσκολότερα προβλήματα τίθενται στο προσκήνιο. Οι κατηγορίες των προβλημάτων προς αντιμετώπιση συνεχώς διευρύνονται, το οπλοστάσιο των τεχνικών και των μεθόδων που χρησιμοποιούνται για τη λύση τους διαρκώς αυξάνει και μπορεί να ειπωθεί χωρίς υπερβολή ότι κάθε ουσιαστικό βήμα προόδου, είτε συμβαίνει στον τομέα του Υλικού είτε στο Λογισμικό, επιφέρει, σε σύντομο χρονικό διάστημα, αλλαγή και βελτίωση πολλαπλάσιου μεγέθους στο «ανταγωνιστικό» του κομμάτι. Όταν γινόμαστε μάρτυρες μιας θεαματικής προόδου σε κάτι που αφορά π.χ. το Λογισμικό, σχεδόν σίγουρα θα ακολουθήσει μια «επανάσταση» στο Υλικό, η οποία θα αφομοιώσει τα δυνατότερα σημεία της προόδου στο software και θα κάνει τα μέχρι πρότινος «γρήγορα» συστήματα να μοιάζουν απαρχαιωμένα.

Από τους πλέον επαναστατικούς κλάδους της Αριθμητικής Ανάλυσης θεωρείται και αυτός ο οποίος αναπτύσσεται γύρω από το πρόβλημα της επίλυσης Διαφορικών Εξισώσεων με αριθμητικές μεθόδους. Θα μπορούσε να σκεφτεί κανείς ότι στο συγκεκριμένο πεδίο, η Θεωρία και η Πρακτική Εφαρμογή βρίσκονται σε διάσταση, έχοντας αναπτύξει η καθεμιά εντελώς διαφορετικές μεθοδολογίες και τρόπους αντιμετώπισης προβλημάτων. Κάτι τέτοιο όμως, όπως και σε πολλές παρεμφερείς περιπτώσεις, δεν ανταποκρίνεται στην αλήθεια. Οι θεωρητικές μέθοδοι βρίσκονται σε αγαστή συνεργασία με την αριθμητική τους υλοποίηση, έχοντας κοινή συνισταμένη τη Μοντελοποίηση του Προβλήματος. Η συμπληρωματικότητα αυτή είναι η κινητήρια δύναμη που κάνει θεωρητικά οικοδομήματα και τεχνολογικά επιτεύγματα, τα οποία χθες θεωρούνταν ουτοπικά, να είναι κοινός τόπος σήμερα.

Και, φυσικά, η άλλη όψη του νομίσματος, το τίμημα της παραπέρα προόδου, είναι το ότι για να υπάρξει αυτή η πρόοδος, θα πρέπει να συγκεραστούν *και οι δύο κατευθύνσεις μοντελοποίησης, τόσο η θεωρητική, όσο και η υλοποιήσιμη / μετρήσιμη / η δυνάμενη να προγραμματιστεί.*

1.2 Συνήθεις Διαφορικές Εξισώσεις και Προβλήματα Αρχικών Τιμών.

Το ερευνητικό πεδίο με το οποίο θα ασχοληθούμε είναι μέρος του τομέα των διαφορικών εξισώσεων: είναι τα προβλήματα αρχικών τιμών των συνήθων διαφορικών εξισώσεων. Για αυτά έχουν αναπτυχθεί διάφορες μεθοδολογίες επίλυσης, οι οποίες διαφέρουν στη φιλοσοφία, στο σχεδιασμό και την υλοποίησή τους. Υπάρχουν εκτενείς επισκοπήσεις [Aberth, 1998, Brezinski & Wuytack, 2001, Huang & Innanen, 1996], τόσο για «ήπια» (non–stiff) όσο και για «δύσκολα» (stiff) προβλήματα [Jorba & Zou, 2004].

Η μεθοδολογία επίλυσης προβλημάτων αρχικών τιμών ΣΔΕ με τη χρήση αναπτυγμάτων

σε σειρές έχει χρησιμοποιηθεί από πολύ παλαιές εποχές [Barrio et. al., 2003]. Λόγω όμως της δυσκολίας που αυτή παρουσίαζε πολλές φορές στον αναλυτικό υπολογισμό των απαιτούμενων παραγώγων στους όρους των αναπτυγμάτων, δεν χρησιμοποιούνταν ως πρώτη επιλογή, στην προ των Η/Υ εποχή [Roberts, 1975]. Η εντατική χρήση πλέον των υπολογιστών έχει επιφέρει σημαντικές αλλαγές στις προτιμήσεις της υπολογιστικής κοινότητας, τόσο που να μιλάμε πλέον για μία αναβίωση της μεθόδου και το χαρακτηρισμό της ως «μοντέρνα μέθοδος των σειρών»[Jorba & Zou, 2004, Fox, 1984, Murison, 1989, Corliss & Chang, 1982]. Η τεχνολογία που αναπτύσσεται με βάση αυτή [IMA, 1998, ADT, 2005, GNU gsl, 2005] θα μας απασχολήσει στη συνέχεια αυτής της εργασίας και Θα αναπτύξουμε μία μεθοδολογία επίλυσης ΣΔΕ με εφαρμογή σε προβλήματα αρχικών συνθηκών [Hadjifoteinou & Gousidou-Koutita, 1998, Le Guyader, 1993, Χατζηφωτεινού, 2002].

Έχουμε επιλέξει μερικά πρότυπα προβλήματα, τα οποία θα βοηθήσουν στο να καταστεί ευκρινής τόσο η λεπτομερειακή ανάπτυξη της μεθόδου, λόγω της σταδιακά αυξανόμενης πολυπλοκότητάς τους, όσο και η αποτίμηση της αξίας της, εφόσον αυτά είναι επιλύσιμα (το καθένα σε διαφορετικό βαθμό) και με άλλες μεθόδους. Συγκεκριμένα, θα ασχοληθούμε με: 1) το πρόβλημα του αρμονικού ταλαντωτή, 2) το πρόβλημα των δύο σωμάτων και δίνουμε κύρια έμφαση στην επίλυση του προβλήματος των τριών σωμάτων, σε δύο συγκεκριμένες του μορφές: 3α) στο γενικό επίπεδο πρόβλημα και 3β) στο πλήρες γενικό πρόβλημα, κατά τις οποίες καμία από τις εμπλεκόμενες μάζες δε θεωρείται απειροστή ή αγνοήσιμη.

2 Ολοκλήρωση συνήθων διαφορικών εξισώσεων με τη μέθοδο των σειρών

2.1 Θεωρητική Εισαγωγή.

Αντικείμενο της παρούσης εργασίας είναι η μελέτη ορισμένων τεχνικών εύρεσης λύσεων με μορφή δυναμοσειράς σε γραμμικές ομογενείς δεύτερης τάξης συνήθεις διαφορικές εξισώσεις και η υλοποίηση αντίστοιχων υπολογιστικών αλγορίθμων.

Θεωρούμε τη γενική μορφή της γραμμικής ομογενούς δεύτερης τάξης Σ.Δ.Ε.

$$\frac{d^2y}{dx^2} + p(x)\,\frac{dy}{dx} + q(x)\,y = 0\ , \tag{1}$$

όπου οι $p(x)$ και $q(x)$ είναι συναρτήσεις μόνο της ανεξάρτητης μεταβλητής x. Η εγκυρότητα της ανάπτυξης σε σειρά της y γύρω από ένα σημείο x_0, καθορίζεται από τη συμπεριφορά των συναρτήσεων $p(x)$ και $q(x)$ στη γειτονιά του σημείου x_0.

Σκοπός μας είναι να κατασκευάσουμε λύσεις της (1) της μορφής

$$y(x) = x^k \sum_{n=0}^{\infty} \alpha_n x^n = \sum_{n=0}^{\infty} \alpha_n x^{n+k} \tag{2}$$

ή συναφούς μορφής (βλ. παρακάτω, εξ. (4)). Στην εξ. (2), οι αποδεκτές τιμές του k καθορίζονται από τον περιορισμό: $a_0 \neq 0$.

Εάν οι $p(x)$ και $q(x)$ στην (1) είναι πεπερασμένες, μονότιμα ορισμένες και διαφορίσιμες στο x_0, τότε το x_0 καλείται **ομαλό σημείο** ή «σύνηθες σημείο» (regular point). Η εξίσωση (1) λέμε ότι είναι ομαλή στο x_0. Σε αυτή την περίπτωση και τα δύο όρια, $\lim_{x\to x_0} p(x)$ και $\lim_{x\to x_0} q(x)$ υπάρχουν και είναι πεπερασμένα. Εάν όμως ένα τουλάχιστον από αυτά απειρίζεται, λέμε ότι το x_0 είναι **ιδιάζον σημείο** (singular point): η (1) είναι ιδιάζουσα στο x_0.

Έστω ότι το x_0 είναι ιδιάζον σημείο, αλλά **και τα δύο όρια**

$$\lim_{x\to 0}(x-x_0)p(x) \quad \kappa\alpha\iota \quad \lim_{x\to 0}(x-x_0)^2 q(x) \tag{3}$$

υπάρχουν και είναι πεπερασμένα. Τότε λέμε ότι η (1) έχει **κανονική ανωμαλία** (απαλείψιμη ανωμαλία, regular/removable singularity) στο x_0. Εάν όμως τουλάχιστον ένα από τα όρια της (3) απειρίζεται, τότε λέμε ότι υπάρχει **ουσιώδης ανωμαλία** (essential singularity) στο $x = x_0$.

Η παραπάνω κατηγοριοποίηση είναι χρήσιμη, διότι εάν κάποιο σημείο x_0 είναι ομαλό, τότε μπορούμε πάντα να βρούμε **δύο ανεξάρτητες λύσεις** με μορφή γενικευμένης δυναμοσειράς

$$y(x) = \sum_{n=0}^{\infty} \alpha_n (x-x_0)^{n+k} \ , \tag{4}$$

οι οποίες συγκλίνουν για όλες τις τιμές του x, από το x_0 και μέχρι το κοντινότερο ιδιάζον σημείο.

Παραθέτουμε (χωρίς απόδειξη) το παρακάτω

ΘΕΩΡΗΜΑ [Fuch]: Έστω η γενική μορφή της γραμμικής και ομογενούς ΣΔΕ τάξης N:

$$\frac{d^N y}{dx^N} + A_{N-1}(x)\frac{d^{N-1}y}{dx^{N-1}} + \ldots + A_1(x)\frac{dy}{dx} + A_0(x)\, y = 0$$

Τότε:

1. Σε ένα **ομαλό σημείο x_0**, υπάρχουν N *γραμμικά ανεξάρτητες* λύσεις της εξίσωσης.
2. Οι σειρές Taylor για αυτές τις λύσεις, $\sum_{n=0}^{\infty} a_n (x-x_0)^n$, συγκλίνουν τουλάχιστον μέχρι το κοντινότερο ανώμαλο σημείο.
3. Σε ένα **κανονικό ανώμαλο σημείο**, υπάρχει **τουλάχιστον μία** λύση της μορφής (4), η οποία συγκλίνει και αυτή τουλάχιστον μέχρι το κοντινότερο ιδιάζον σημείο.

2.2 Περιορισμοι και εξειδικεύσεις σε προβλήματα Μηχανικής.

Όπως θα δούμε παρακάτω (βλ. Υποενότητα 3.1), στην αναλυτική παράθεση αποτελεσμάτων για το γραμμικό αρμονικό ταλαντωτή, εάν οι συναρτήσεις p και q της (1) είναι απαλλαγμένες από ανώμαλα σημεία, είναι δηλαδή αναλυτικές σε όλο το πεδίο ορισμού τους, η μέθοδος των γενικευμένων δυναμοσειρών μπορεί να απλουστευτεί και ο υπολογιστικός φόρτος να μειωθεί σημαντικά, εάν χρησιμοποιήσουμε απλές δυναμοσειρές.

Η σύγκλιση των δυναμοσειρών είναι εξασφαλισμένη για τα προβλήματα Μηχανικής που θα μας απασχολήσουν στη συνέχεια, διότι τα μέτρα των όρων ανώτερης τάξης βαίνουν ελαττούμενα.

Ακόμη και σε προβλήματα μεγαλύτερης της μίας διάστασης, όπου πιθανόν να υπάρχουν συγκρούσεις (collisions), ανώμαλα σημεία στο χώρο των φάσεων, ανάλογα με τις αρχικές συνθήκες, θα υλοποιήσουμε τη μέθοδο των απλών δυναμοσειρών και όχι των γενικευμένων, όπως αυτές θα προέκυπταν από την εφαρμογή της μεθόδου Frobenius. Το σύνολο των collisions είναι, στα θεωρούμενα προβλήματα, σύνολο μηδενικού μέτρου. Ακόμη και εάν κάποια σύνολα αρχικών συνθηκών οδηγούν σε κρούσεις, υπάρχει πρόβλεψη στον κώδικα,

όπου αποφεύγεται το πρόβλημα σύγκλισης των σειρών και ο ελλειπής καθορισμός των περιοχών σύγκλισης (ROC–Region Of Convergence) και επιτυγχάνεται η βέλτιστη επιλογή βήματος ολοκλήρωσης, σύμφωνα με κάποια κριτήρια.

2.3 Κανονικοποιημένες παράγωγοι και αυτοματοποιημένη διαφόριση.

Εάν $f(x) : x \in I \subset \mathbb{R} \mapsto \mathbb{R}$ είναι μία (λεία) συνάρτηση συμβολίζουμε την n–οστή της παράγωγο ως

$$\frac{d^n}{dx^n} f(x) \equiv f^{(n)}(x) \ . \tag{5}$$

Ορίζουμε ως **κανονικοποιημένη παράγωγο n–οστής τάξης** την

$$f^{[n]}(x) \equiv \frac{1}{n!} f^{(n)}(x) \ . \tag{6}$$

Ο συμβολισμός αυτός μπορεί να επεκταθεί και σε μιγαδικές συναρτήσεις.

Υποθέτουμε τώρα ότι η $f(x)$ μπορεί να εκφραστεί σα συνάρτηση δύο άλλων συναρτήσεων (κλάσεως C^n), $g(x)$ και $h(x)$:

$$f(x) = F(g(x), h(x)) \ , \tag{7}$$

των οποίων οι κανονικοποιημένες παράγωγοι σε κάποιο σημείο x, $g^{[i]}(x)$, $h^{[i]}(x)$, $i = 0, 1, \ldots n$ είναι γνωστές ή μπορεί να υπολογιστούν. Ισχύει τότε η παρακάτω

ΠΡΟΤΑΣΗ : Εάν οι συναρτήσεις g και h είναι κλάσης C^n και $\alpha \in \mathbb{R} - \{0\}$, $\lambda_1, \lambda_2 \in \mathbb{R}$, θα έχουμε:

1. Εάν $f(x) = \lambda_1 g(x) \pm \lambda_2 h(x)$, τότε

$$f^{[n]}(x) = \lambda_1 g^{[n]}(x) \pm \lambda_2 h^{[n]}(x) \tag{8}$$

2. Εάν $f(x) = g(x) \cdot h(x)$, τότε

$$f^{[n]}(x) = \sum_{j=0}^{n} g^{[n-j]}(x) \cdot h^{[j]}(x) \tag{9}$$

3. Εάν $f(x) = \dfrac{g(x)}{h(x)}$, τότε

$$f^{[n]}(x) = \frac{1}{h^{[0]}(x)} \left[g^{[n]}(x) - \sum_{j=1}^{n} h^{[j]}(x) \cdot f^{[n-j]}(x) \right] \tag{10}$$

4. Εάν $f(x) = g^{\alpha}(x)$, τότε

$$f^{[n]}(x) = \frac{1}{n\, g^{[0]}(x)} \sum_{j=0}^{n-1} [n\alpha - j(\alpha + 1)]\, g^{[n-j]}(x) \cdot f^{[j]}(x) \tag{11}$$

5. Εάν $f(x) = e^{g(x)}$, τότε

$$f^{[n]}(x) = \frac{1}{n}\sum_{j=0}^{n-1}(n-j)\cdot g^{[n-j]}(x)\cdot f^{[j]}(x) \tag{12}$$

6. Εάν $f(x) = \ln(g(x))$, τότε

$$f^{[n]}(x) = \frac{1}{g^{[0]}(x)}\left[g^{[n]}(x) - \frac{1}{n}\sum_{j=1}^{n-1}(n-j)\,g^{[j]}(x)\cdot f^{[n-j]}(x)\right] \tag{13}$$

7. Εάν $f(x) = \cos(h(x))$ και $g(x) = \sin(h(x))$, τότε

$$f^{[n]}(x) = -\frac{1}{n}\sum_{j=1}^{n} j g^{[n-j]}(x)\cdot h^{[j]}(x) \quad g^{[n]}(x) = \frac{1}{n}\sum_{j=1}^{n} j f^{[n-j]}(x)\cdot h^{[j]}(x) \tag{14}$$

ΑΠΟΔΕΙΞΗ :

1. $f(x) = \lambda_1 g(x) \pm \lambda_2 h(x) \Rightarrow f^{(n)}(x) = \lambda_1 g^{(n)}(x) \pm \lambda_2 h^{(n)}(x) \Rightarrow$
$\frac{1}{n!}f^{(n)}(x) = \lambda_1\frac{1}{n!}g^{(n)}(x) \pm \lambda_2\frac{1}{n!}h^{(n)}(x) \Rightarrow f^{[n]}(x) = \lambda_1 g^{[n]}(x) \pm \lambda_2 h^{[n]}(x).$ ■

2. Η n–οστή παράγωγος της f, όταν $f = g\cdot h$, δίνεται από τον κανόνα διαφόρισης του Leibniz: $f^{(n)} = \sum_{j=0}^{n}\left(\begin{array}{c} n \\ j \end{array}\right) g^{(n-j)}h^{(j)}$, οπότε θα είναι

$$f^{[n]} = \frac{1}{n!}f^{(n)} = \frac{1}{n!}\sum_{j=0}^{n}\left(\begin{array}{c} n \\ j \end{array}\right) g^{(n-j)}h^{(j)} = \frac{1}{n!}\sum_{j=0}^{n}\frac{n!}{j!(n-j)!}\,g^{(n-j)}h^{(j)} =$$
$$= \sum_{j=0}^{n}\frac{1}{(n-j)!}\,g^{(n-j)}\frac{1}{j!}h^{(j)} = \sum_{j=0}^{n} g^{[n-j]}h^{[j]}$$ ■

3. Είναι $f = \frac{g}{h} \Rightarrow g = f\cdot h$, οπότε, από την (9) είναι

$$g^{[n]} = \sum_{j=0}^{n} f^{[n-j]}h^{[j]} \Rightarrow g^{[n]} = f^{[n]}h^{[0]} + \sum_{j=1}^{n} f^{[n-j]}h^{[j]} \Rightarrow$$
$$f^{[n]} = \frac{1}{h^{[0]}}\left[g^{[n]} - \sum_{j=1}^{n} f^{[n-j]}h^{[j]}\right]$$ ■

4. Είναι
$f = g^{\alpha} \Rightarrow \ln f = \alpha\ln g \Rightarrow (\ln f)' = (\alpha\ln g)' \Rightarrow \frac{f'}{f} = \alpha\frac{g'}{g} \Rightarrow gf' = \alpha g'f.$
Οι n–οστές κανονικοποιημένες παράγωγοι των μελών της τελευταίας εξίσωσης είναι

$$\left(gf'\right)^{[n]} = \sum_{j=0}^{n} g^{[n-j]}\left(f'\right)^{[j]} = \sum_{j=0}^{n} g^{[n-j]}(j+1)f^{[j+1]} = \sum_{j=0}^{n-1} g^{[n-j]}(j+1)f^{[j+1]} + \underbrace{(n+1)gf^{[n+1]}}_{j=n\ term} \tag{15}$$

$$\left(\alpha g'f\right)^{[n]} = \alpha\sum_{j=0}^{n}\left(g'\right)^{[n-j]}f^{[j]} = \alpha\sum_{j=0}^{n}(n-j+1)g^{[n-j+1]}f^{[j]} \tag{16}$$

Για $n+1 \to k$, οι (15) και (16) δίνουν

$$gkf^{[k]} + \sum_{j=0}^{k-2} g^{[k-j-1]}(j+1)f^{[j+1]} = \alpha \sum_{j=0}^{n}(n-j+1)g^{[n-j+1]}f^{[j]} \tag{17}$$

Η (17) για $k-1 \to m+1,\ k \to m+2$ γίνεται

$$gkf^{[k]} = -\sum_{j=0}^{m} g^{[m-j+1]}(j+1)f^{[j+1]} + \alpha \sum_{j=0}^{n}(n-j+1)g^{[n-j+1]}f^{[j]} \tag{18}$$

Για $m \to n,\ n \to k-1$ έχουμε τώρα

$$gkf^{[k]} = \sum_{j=0}^{k-1} \left(\alpha(k-j)f^{[j]} - (j+1)f^{[j+1]}\right)\ g^{[k-j]} \tag{19}$$

οπότε τελικά θα είναι

$$f^{[k]} = \frac{1}{kg} \sum_{j=0}^{k-1} (k\alpha - j(a+1))\ g^{[k-j]}f^{[j]} \tag{20}$$

■

Με παρόμοιο τρόπο αποδεικνύονται και οι υπόλοιπες ιδιότητες 5, 6 και 7.

2.4 Γενική Υπολογιστική Μεθοδολογία.

Ανάγουμε το σύστημα των εξισώσεων μας σε πρωτοβάθμιο, αντικαθιστώντας παραγώγους ανώτερης της πρώτης τάξης με νέες μεταβλητές, όπου αυτό χρειάζεται. Στα προβλήματα που θα μας απασχολήσουν παρακάτω, αυτό σημαίνει ότι τα δεξιά μέλη θα εκφράζουν τους ρυθμούς μεταβολής των θέσεων και των ταχυτήτων των σωμάτων των θεωρούμενων μηχανικών συστημάτων.

Έχοντας τώρα το πρωτοβάθμιο σύστημα διαφορικών εξισώσεων, έστω τάξης n, εισάγουμε τη χρήση k βοηθητικών συναρτήσεων, $k \geq 2n$, με την ισότητα να ισχύει όταν τα δεξιά μέλη μπορούν να εκφραστούν ως γραμμικοί συνδυασμοί των αρχικών n μεταβλητών. Αυτός είναι και ο στόχος μας, να χρησιμοποιήσουμε τόσες U_k, ώστε τα δεξιά μέλη των εξισώσεων που δίνουν τους ρυθμούς μεταβολής των «κανονικών» μεταβλητών, να καθίστανται γραμμικός συνδυασμός των U_k.

Κάθε μη γραμμικότητα θα πρέπει να αντικατασταθεί με μία τουλάχιστον βοηθητική μεταβλητή. Συγκεκριμένα, χρειάζεται ακριβώς μία καινούργια βοηθητική μεταβλητή, εάν η μη γραμμική σχέση ανάγεται σε κάποιον από τους προαναφερθέντες κανόνες ανάπτυξης σε σειρά. Εάν όχι, χρειάζεται να γίνουν ενδιάμεσα βήματα.

Ανάλογα με το πρόβλημα, το k ποικίλλει και εξαρτάται από το εάν μπορούμε να κάνουμε χρήση των κανόνων 1–7, για να εκφράσουμε με συμπαγή τρόπο τις μη γραμμικότητες που υπεισέρχονται. Το συγκεκριμένο πρόβλημα είναι πρόβλημα βελτιστοποίησης έκφρασης σύνθετων παραστάσεων με όσο το δυνατόν λιγότερους πρωτογενείς όρους (primitives) με κανόνες αναγωγής τους 1–7, για προφανείς λόγους υπολογιστικού κόστους. Για αυτό το σημαντικό και περίπλοκο πρόβλημα, έχουν αναπτυχθεί τεχνικές αντιμετώπισης του, τόσο στη θεωρία γραφημάτων, όσο και στη θεωρία κατασκευής μεταγλωττιστών (compiler design) και δεν θα μας απασχολήσει η πλήρης αντιμετώπιση του. Εμφανίζεται και σε άλλους

συναφείς τομείς, με σημαντικότερο αυτόν της αυτοματοποιημένης διαφόρισης [IMA, 1998, ADT, 2005], όπως ήδη έχουμε δει στην προηγούμενη Υποενότητα.

Για το σκοπό μας, για τα παραδείγματα που έχουμε επιλέξει να παρουσιάσουμε, η κατάστρωση του συστήματος είναι σχετικά εύκολη. Αυτό εκπηγάζει από το γεγονός ότι τα προβλήματα αυτά είναι προβλήματα δυναμικής, με σχετικά απλή φύση των δυνάμεων που εμπλέκονται, κάτι που οδηγεί στην απλοποίηση των εξισώσεων που τα περιγράφουν. Αυτό ενισχύεται ακόμα περισσότερο και από τις συμμετρίες που υπεισέρχονται, λόγω δράσεων και αντιδράσεων και από το ότι, όπως ήδη έχουμε αναφέρει παραπάνω, ασχολούμαστε με γραμμικές και ομογενείς συνήθεις διαφορικές εξίσωσεις δεύτερης τάξης.

Με τα πρόσφατα 64–bit υπολογιστικά συστήματα μπορούμε να αυξήσουμε την ακρίβεια των απαιτούμενων υπολογισμών, χωρίς επαναπρογραμματισμό της μεθόδου επίλυσης των ΔΕ με σειρές.

Στόχος μας είναι η υλοποίηση ενός υπολογιστικού πλαισίου, το οποίο να έχει όριο αριθμητικής ακριβείας την ακρίβεια μηχανής (machine precision). Ο κώδικας αυτός θα χρησιμοποιηθεί σε πειράματα προσομοίωσης κίνησης σύνθετων πλανητικών συστημάτων [Le Guyader, 1993, Milani & Nobili, 1988, Murison, 1989, Quinn & Tremaine, 1990, Spallicci et. al., 2005], όπου οι χρόνοι είναι αστρονομικής κλίμακας, μετρούμενοι σε δισεκατομμύρια χρόνια και θέλουμε τα παραγόμενα ενδιάμεσα αποτελέσματα να είναι register–based και όχι να υπάρχουν διαδικασίες μεταφοράς ορισμάτων από και προς τη μνήμη του Η/Υ (memory–to–memory operations), να μην υπολογίζονται δηλαδή τα ενδιάμεσα αποτελέσματα, θέσεις και ταχύτητες, με πράξεις σε δομές δεδομένων στη μνήμη, όπως αναπόφευκτα θα συμβεί εάν ζητήσουμε ακρίβεια υπολογισμών μεγαλύτερη της ακρίβειας μηχανής του υπολογιστικού μας συστήματος, από τις βιβλιοθήκες αριθμητικών υπολογισμών αυθαίρετης ακριβείας [GNU gmp, 2005, GNU gsl, 2005].

2.4.1 Υπολογιστικός πυρήνας – πρόγραμμα οδήγησης – διασύνδεση.

Ο υπολογιστικός πυρήνας είναι η συνάρτηση *do_a_step(...)*, η οποία είναι υπεύθυνη για τη μετάβαση της κατάστασης του συστήματος από ένα αρχικό σημείο στον (επαυξημένο) χώρο των φάσεων σε ένα τελικό. Δέχεται σαν είσοδο το αρχικό σημείο και ένα προτεινόμενο χρονικό βήμα ολοκλήρωσης και παράγει στην έξοδό της ένα καινούριο σημείο και το επόμενο προτεινόμενο βήμα ολοκλήρωσης. Σε αυτή είναι ενσωματωμένα κάποια κριτήρια επιλογής βήματος και διάγνωσης πιθανόν προβληματικών καταστάσεων (π.χ. αδυναμία εκτέλεσης βήματος λόγω μη ικανοποιητικής σύγκλισης των σειρών, αδυναμία επιλογής του επόμενου βήματος γιατί αυτό βρίσκεται κάτω από την κατώτατη αποδεκτή ή πάνω από την ανώτατη αποδεκτή τιμή του, κλπ). Επίσης, η συνάρτηση αυτή παίρνει αποφάσεις για εξοικονόμηση υπολογιστικού φόρτου, βλέποντας το πόσοι όροι απαιτούνται για αποδεκτή σύγκλιση με το συγκεκριμένο χρονικό βήμα και αποφασίζει την αύξηση ή μείωση της τάξης των σειρών που κατασκευάζονται. Η υλοποίησή της στα παρακάτω προβλήματα μπορεί να γίνει περνώντας σε αυτή διάφορα ορίσματα (π.χ. δείκτες σε κατάλληλα διαμορφωμένους πίνακες αρχικών και τελικών σημείων στον χώρο των φάσεων, δείκτη σε βήμα χρονικής ολοκλήρωσης κλπ). Έχει υλοποιηθεί κάθε φορά η πιο πρόσφορη και υπολογιστικά γρηγορότερη υπογραφή της, π.χ. με τη χρήση global μεταβλητών.

Η αριθμητική επίλυση ενός συγκεκριμένου προβλήματος, επιτυγχάνεται με την ενσωμάτωση του υπολογιστικού πυρήνα σε ένα πρόγραμμα οδήγησης (driver program), το οποίο έχει τη γενική μορφή του Αλγόριθμου 1.

Τα παραγόμενα αποτελέσματα μπορούν να αποθηκευτούν για παραπέρα επεξεργασία και γραφική αναπαράσταση σε διάφορες μορφές. Έχουμε επιλέξει τη βιβλιοθήκη γραφικής

```
1 // — Taylor Series ODE Solver Driver Program
2 initialization: specific computational problem setup;
3 while (time_left < time_required) do
4 |   perform a step & keep needed results for further processing;
  |   input  : Point in phase space, recommended integration step
  |   output: New point in phase space, new indicated integration step
5 |   do_a_step(...);
6 |   update time & adjust step for next iteration;
7 end
```

Algorithm 1: Πρόγραμμα οδήγησης του υπολογιστικού πυρήνα.

διασύνδεσης pgplot [Pearson, pgplot 2005], διαδεδομένη στην ακαδημαϊκή κοινότητα, η οποία προσφέρει την επιπλέον προαιρετική ευκολία της αποφυγής εγγραφής δεδομένων στο σκληρό δίσκο και της μετέπειτα επεξεργασίας τους, παρέχοντας την on line, real time απεικόνισή τους στην οθόνη ή σε hard copy.

3 Πρότυπα Προβλήματα.

3.1 Αρμονικός ταλαντωτής, $y'' + \omega^2 y = 0$.

3.1.1 Στοιχεία από τη Θεωρία και αναλυτική προσέγγιση με γενικευμένες δυναμοσειρές.

Η Δ.Ε.

$$\frac{d^2y}{dx^2} + \omega^2 y = 0 \tag{21}$$

έχει γενική λύση την

$$y(x) = A\,\cos(\omega x) + B\,sin(\omega x)\ , \tag{22}$$

όπου A και B είναι (οι δύο απαιτούμενες) αυθαίρετες σταθερές, καθοριζόμενες από τις οριακές συνθήκες του προβλήματος.

Αναζητούμε λύσεις της (21) με μορφή γενικευμένης δυναμοσειράς:

$$y = \sum_{n=0}^{\infty} \alpha_n\, x^{n+k} \tag{23}$$

$$\frac{dy}{dx} = \sum_{n=0}^{\infty} (n+k)\alpha_n\, x^{n+k-1} \tag{24}$$

$$\frac{d^2y}{dx^2} = \sum_{n=0}^{\infty} (n+k)(n+k-1)\alpha_n\, x^{n+k-2} \tag{25}$$

Αντικαθιστώντας τις (23), (24) και (25) στην εξ. (21), βρίσκουμε ότι

$$\sum_{n=0}^{\infty} (n+k)(n+k-1)\alpha_n\, x^{n+k-2} + \omega^2 \sum_{n=0}^{\infty} \alpha_n\, x^{n+k} = 0\ . \tag{26}$$

Η εξίσωση δεικτών (indicial equation) για το k λαμβάνεται από τη μικρότερη δύναμη που εμφανίζεται στην (26) και συγκεκριμένα την x^{k-2}. Αυτή εμφανίζεται μόνο στο αριστερό άθροισμα (για $n=0$) και επομένως ο αντίστοιχος όρος πρέπει να μηδενίζεται:

$$k(k-1)a_0\,x^{k-2}=0\ . \tag{27}$$

Έχουμε επομένως δύο δυνατές τιμές του k, τις $k=0$ και $k=1$, κάτι που θεωρείται τυπικό για εξισώσεις δεύτερης τάξης.

Μετασχηματίζουμε το βωβό δείκτη n στο πρώτο άθροισμα της (26) ως $n-2\to\xi$, οπότε

$$n\to\xi+2,\ \ n+k-2\to\xi+k,\ \ n+k\to\xi+2+k,\ \ n+k-1\to\xi+k+1,\ \ \sum_{n=0}^{\infty}\to\sum_{\xi=-2}^{\infty}$$

και η (26) γράφεται

$$\sum_{\xi=-2}^{\infty}(\xi+k+2)(\xi+k+1)\alpha_{\xi+2}\,x^{\xi+k}+\omega^2\sum_{n=0}^{\infty}\alpha_n\,x^{n+k}=0 \tag{28}$$

Τώρα, θέτοντας $\xi\to n$, παίρνουμε

$$\sum_{n=-2}^{\infty}(n+k+2)(n+k+1)\alpha_{n+2}\,x^{n+k}+\omega^2\sum_{n=0}^{\infty}\alpha_n\,x^{n+k}=0 \tag{29}$$

ή, αναπτύσσοντας τους δύο πρώτους όρους του αριστερού αθροίσματος,

$$\underbrace{k(k-1)a_0x^{k-2}}_{n=-2}+\underbrace{k(k+1)a_1x^{k-1}}_{n=-1}+\sum_{n=0}^{\infty}(n+k+2)(n+k+1)\alpha_{n+2}\,x^{n+k}+\omega^2\sum_{n=0}^{\infty}\alpha_n\,x^{n+k}=0 \tag{30}$$

Λύση για $k=0$

Στην (30) μπορούμε να συγκρίνουμε συντελεστές και να πάρουμε την **αναδρομική σχέση**

$$\alpha_{n+2}=-\frac{\omega^2}{(n+2)(n+1)}\,\alpha_n \tag{31}$$

η οποία έχει τη λύση

$$\alpha_n=\begin{cases}\dfrac{(-\omega^2)^{n/2}a_0}{n!} & n\ \grave{\alpha}\rho\tau\iota o\varsigma\\[2ex] 0 & n\ \pi\varepsilon\rho\iota\tau\tau\acute{o}\varsigma\end{cases} \tag{32}$$

Η ζητούμενη λύση y της (21) είναι

$$y=\alpha_0\sum_{n\ \grave{\alpha}\rho\tau\iota o\varsigma}(-1)^{n/2}(\omega x)^n\frac{1}{n!}=\alpha_0\cos(\omega x) \tag{33}$$

Λύση για $k=1$

Η (29) για $k=1$ δίνει

$$\sum_{n=-2}^{\infty}(n+3)(n+2)\alpha_{n+2}\,x^{n+1}+\omega^2\sum_{n=0}^{\infty}\alpha_n\,x^{n+1}=0 \tag{34}$$

και με το ανάπτυγμα των δύο πρώτων όρων του αριστερού αθροίσματος θα είναι

$$\underbrace{0}_{n=-2} + \underbrace{2a_1}_{n=-1} + \sum_{n=0}^{\infty}(n+3)(n+2)\alpha_{n+2}\, x^{n+1} + \omega^2 \sum_{n=0}^{\infty} \alpha_n\, x^{n+1} = 0\ . \tag{35}$$

Για να ισχύει η (35), θα πρέπει $a_1 = 0$ και η αναδρομική σχέση για τα a_n γίνεται τώρα

$$\alpha_{n+2} = -\frac{\omega^2}{(n+3)(n+2)}\,\alpha_n\ . \tag{36}$$

Από ένα απλό πίνακα τιμών για τα πρώτα n

$n=0$	$a_2 = \dfrac{-\omega^2}{3\cdot 2}\,a_0$
$n=1$	$a_3 = \dfrac{-\omega^2}{4\cdot 3}\,a_1 = 0$
$n=2$	$a_4 = \dfrac{-\omega^2}{5\cdot 4}\,a_2 = \dfrac{-\omega^2}{5\cdot 4}\,\dfrac{-\omega^2}{3\cdot 2} = \dfrac{(-\omega^2)^2}{(4+1)!}\,a_0$
$\vdots$	$\vdots$

επαληθεύουμε ότι η (36) έχει τη λύση

$$\alpha_n = \begin{cases} \frac{(-\omega^2)^{n/2}a_0}{(n+1)!} & n\ \grave{\alpha}\rho\tau\iota o\varsigma \\ \\ 0 & n\ \pi\varepsilon\rho\iota\tau\tau\acute{o}\varsigma \end{cases} \tag{37}$$

Η ζητούμενη λύση y της (21) είναι τώρα

$$y = \alpha_0\, x \sum_{n\ \grave{\alpha}\rho\tau\iota o\varsigma} (-1)^{n/2}(\omega x)^n \frac{1}{(n+1)!} = \frac{\alpha_0}{\omega}\,\sin(\omega x) \tag{38}$$

Παρατηρήσεις

1. Συγκρίνοντας της (22), (33) και (38) παρατηρούμε ότι $A = a_0$, $B = \frac{\alpha_0}{\omega}$.

2. Οι δύο γραμμικά ανεξάρτητες λύσεις στην (22), δηλαδή οι $\cos(\omega x)$ και $\sin(\omega x)$, είναι άρτια και περιττή συνάρτηση της ανεξάρτητης μεταβλητής x, αντίστοιχα. Από τη μορφή της (21) βλέπουμε ότι εάν μία συνάρτηση $f(x)$ την ικανοποιεί, τότε και η $f(-x)$ θα είναι επίσης λύση· ο τελεστής $\dfrac{d^2}{dx^2}$ δεν μεταβάλλεται από την αντικατάσταση $x \to -x$.

3. Τα παραπάνω *δεν επιβάλλουν αναγκαστικά* την απαίτηση κάθε λύση να είναι είτε άρτια είτε περιττή συνάρτηση· μπορούμε όμως να τις εκλέξουμε εμείς ως τέτοιες. Αυτή είναι και η περίπτωση που προέκυψε στις δύο παραπάνω λύσεις με μορφή δυναμοσειράς που κατασκευάσαμε, εξ. (33) και (38). Τα περιττά α_n είναι εντελώς ανεξάρτητα από τα αντίστοιχα άρτια και, όσον αφορά τις λύσεις μας, μπορούν να τεθούν όλα ίσα με μηδέν. Για να έχουμε αμιγώς άρτιες και περιττές λύσεις, η απλούστερη επιλογή είναι να θέσουμε $\alpha_1 = 0$. Εάν $\alpha_1 \neq 0$, δεν παράγονται επιπλέον λύσεις, απλά αναμιγνύονται μερικοί όροι της περίπτωσης με $k = 0$ με αυτούς της περίπτωσης όπου $k = 1$.

3.1.2 Υπολογιστική λύση με χρήση απλής δυναμοσειράς.

Η (21) είναι της μορφής (1), με $p(x) = 0,\ q(x) = \omega^2$. Οι p και q είναι αναλυτικές συναρτήσεις, οπότε δεν αναμένουμε προβλήματα στα αναπτύγματα των δυναμοσειρών και ιδιαιτερότητες στις περιοχές σύγκλισης (βλ. και Υποενότητα 2.2).

Εφαρμόζουμε την μεθοδολογία που παραθέσαμε στην Υποενότητα 2.4:

1. Θεωρούμε την εξίσωση του γραμμικού αρμονικού ταλαντωτή

$$\frac{d^2x}{dt^2} + \omega^2 x(t) = 0\ . \tag{39}$$

   Η αναγωγή της (39) σε πρωτοβάθμιο σύστημα δίνει

$$\begin{aligned} \frac{dx}{dt} &= y \\ \frac{dy}{dt} &= -\omega^2 x \end{aligned} \tag{40}$$

2. Τα δεξιά μέλη των εξισώσεων του συστήματος (40) είναι ήδη γραμμικές συναρτήσεις των $x(t), y(x(t))$, οπότε δε χρειάζονται επιπλέον βοηθητικές συναρτήσεις U_k, πέραν των U_1, U_2 οι οποίες θα χρησιμοποιηθούν στο βασικό βρόχο υπολογισμού.

3. Για απλότητα και χωρίς περιορισμό της γενικότητας, θεωρούμε τις ακόλουθες αρχικές συνθήκες: $x(0) = 0,\ y(0) = 1$.

4. Το σύμπλεγμα βοηθητικών και «κανονικών» μεταβλητών είναι

$$\begin{aligned} U_1 &= -\omega^2 x \\ U_2 &= y \\ x' &= U_2 \\ y' &= U_1 \end{aligned} \tag{41}$$

5. Οι κανονικοποιημένες παράγωγοι υπολογίζονται αναδρομικά από τις σχέσεις

$$\begin{aligned} U_1^{[n]}(t) &= -\omega^2 x^{[n]}(t) \\ U_2^{[n]}(t) &= y^{[n]}(t) \\ x^{[n+1]}(t) &= \frac{1}{n+1} U_2^{[n]}(t) \\ y^{[n+1]}(t) &= \frac{1}{n+1} U_1^{[n]}(t) \end{aligned} \tag{42}$$

6. Κριτήριο αποκοπής της σειράς: εάν το άθροισμα των μέτρων των όρων των θέσεων και ταχυτήτων (όλων δηλαδή των «κανονικών» μεταβλητών των αριστερών μελών του συστήματος (41)) στο τρέχον βήμα επανάληψης, έστω στον όρο τάξης ξ, είναι μικρότερο από μία προκαθορισμένη (μικρή) τιμή *tol*, θεωρούμε ότι η σύγκλιση επιτεύχθηκε.
$$if\ (|x_\xi| + |y_\xi|) \cdot h^{\xi+1} < tol \quad break; \tag{43}$$

7. Επιτέλεση ενός βήματος ολοκλήρωσης: αυτό επιτυγχάνεται με την κλήση του υπολογιστικού πυρήνα του αλγόριθμου, τη συνάρτηση *do_a_step(...)* (βλ. Υποενότητα 2.4).

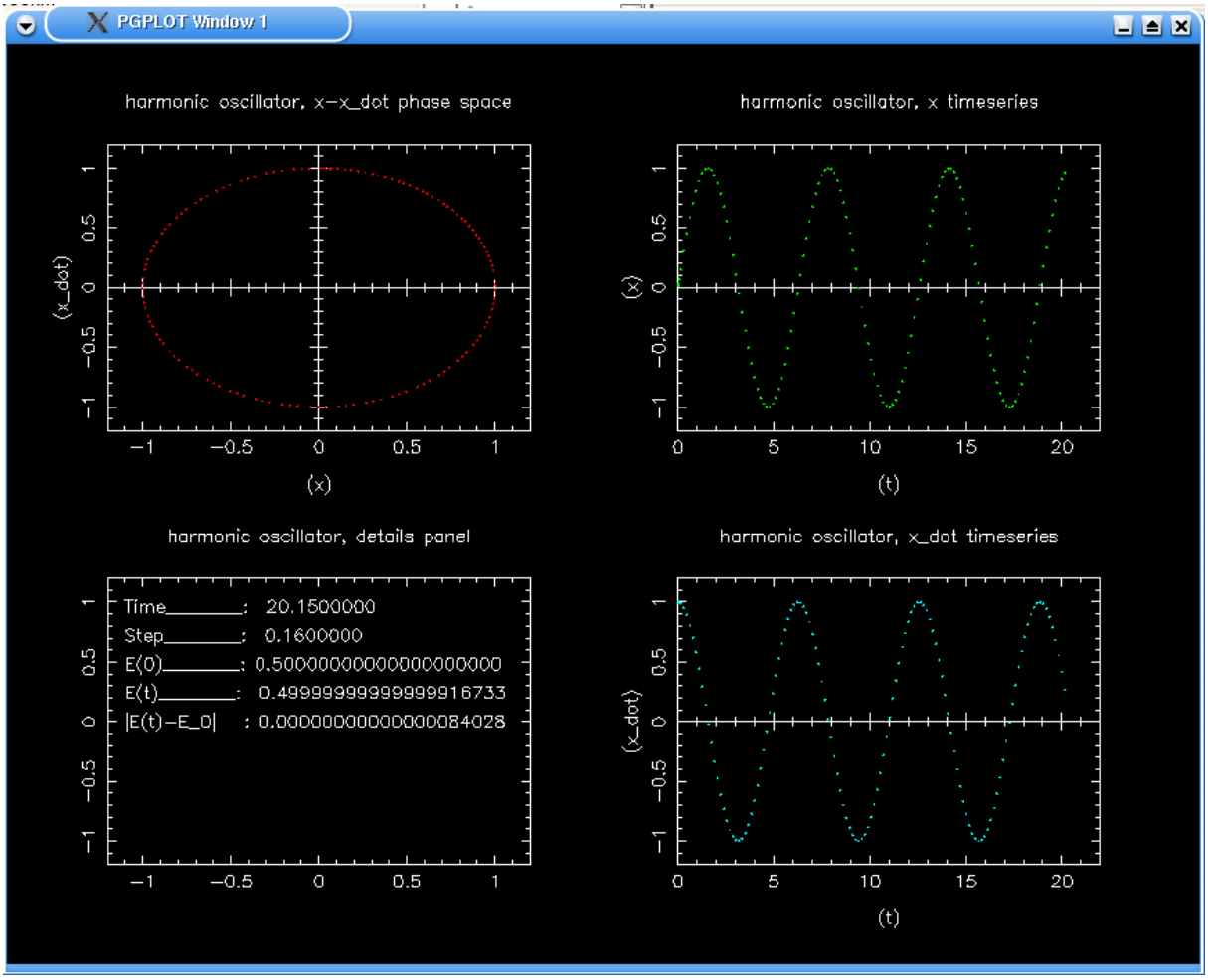
8. Έλεγχος ορθότητας υπολογισμών με τη χρήση του ολοκληρώματος της ενέργειας:
$$E(0) = \frac{1}{2}(y(0))^2 + \frac{1}{2}\omega^2(x(0))^2 \quad vs. \quad E(t) = \frac{1}{2}(y(t))^2 + \frac{1}{2}\omega^2(x(t))^2\ . \tag{44}$$

3.1.3 Κώδικας και αποτελέσματα.

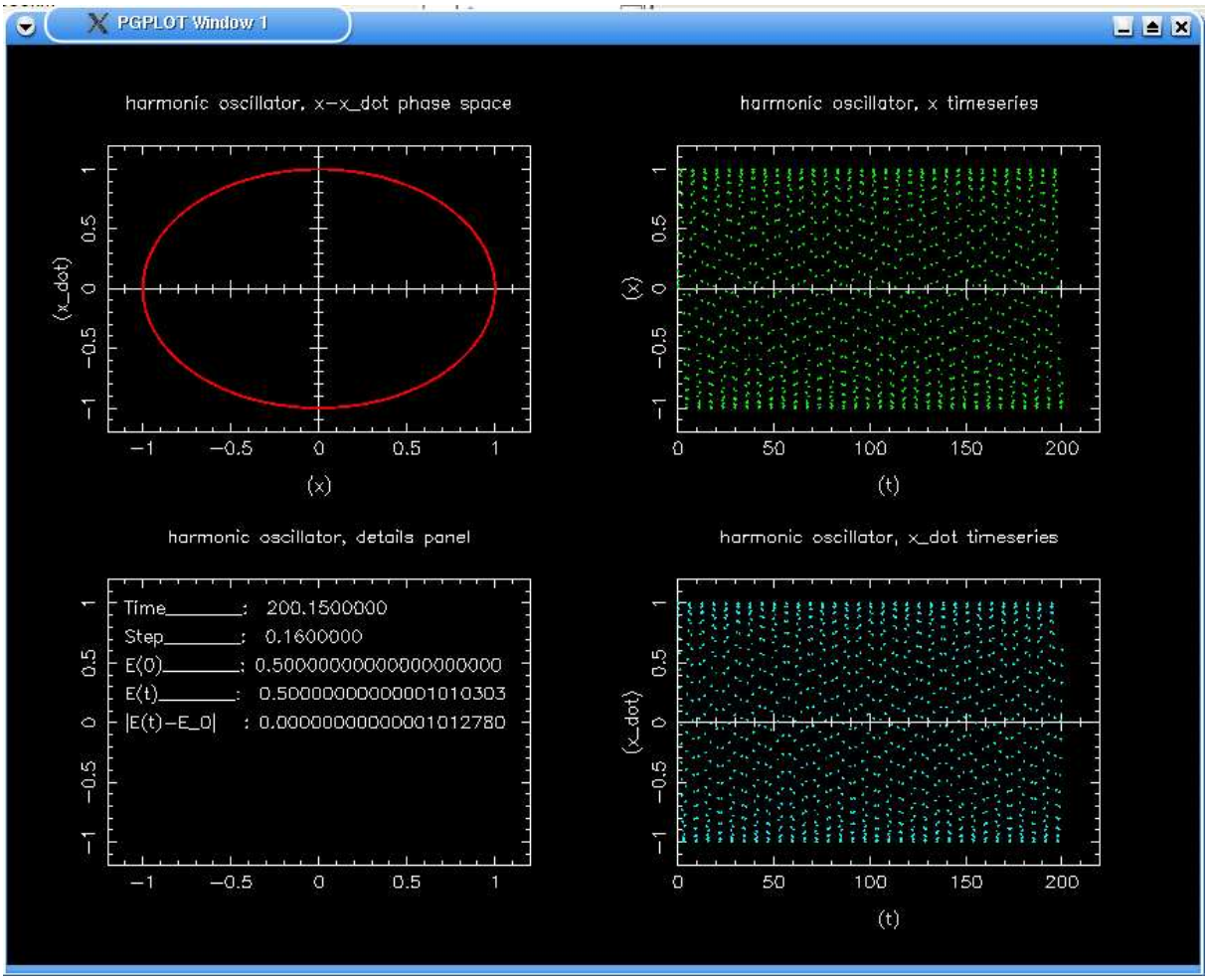
Ένα πρόγραμμα επίλυσης του προβλήματος του γραμμικού αρμονικού ταλαντωτή ακολουθεί στις δύο επόμενες σελίδες. Στο Παράρτημα Α παραθέτουμε τον ίδιο κώδικα, εκτενώς σχολιασμένο και επιπλέον έχουμε υλοποιήσει και μία γραφική διασύνδεση, από την οποία έχουμε επιλέξει να παρουσιάσουμε μερικά στιγμιότυπα, Σχ. 1–2. Στο Σχ 2 παραθέτουμε δύο διαγράμματα, τα οποία απεικονίζουν την εξέλιξη του λάθους των υπολογισμών, όπως αυτό εκφράζεται από τη μεταβολή της τιμής της ενέργειας με το χρόνο ολοκλήρωσης.

```
// Taylor Series solution of the harmonic oscillator
// them - [ekots@uom.gr]
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
// tolerance in x
#define tol 1e-15
#define max(a,b)     (((a) > (b)) ? (a) : (b))
#define min(a,b)     (((a) < (b)) ? (a) : (b))
#define max_n ((int)(40))
#define max_h 0.2L
#define min_h 1e-25
#undef  omega
#define omega (1.00)
#define omega2 (omega*omega)
double u1[max_n], u2[max_n], x[max_n], y[max_n], t, E_0;
//-------------------------------------------------------------------
double Energy(){return 0.5*y[0]*y[0]+0.5*omega2*x[0]*x[0];};
//-------------------------------------------------------------------
double do_a_step(double hstep){
 double h=hstep;
 double new_h;  // new timestep
 int i, order;  // order of Taylor series expansion
 double hpow;
```

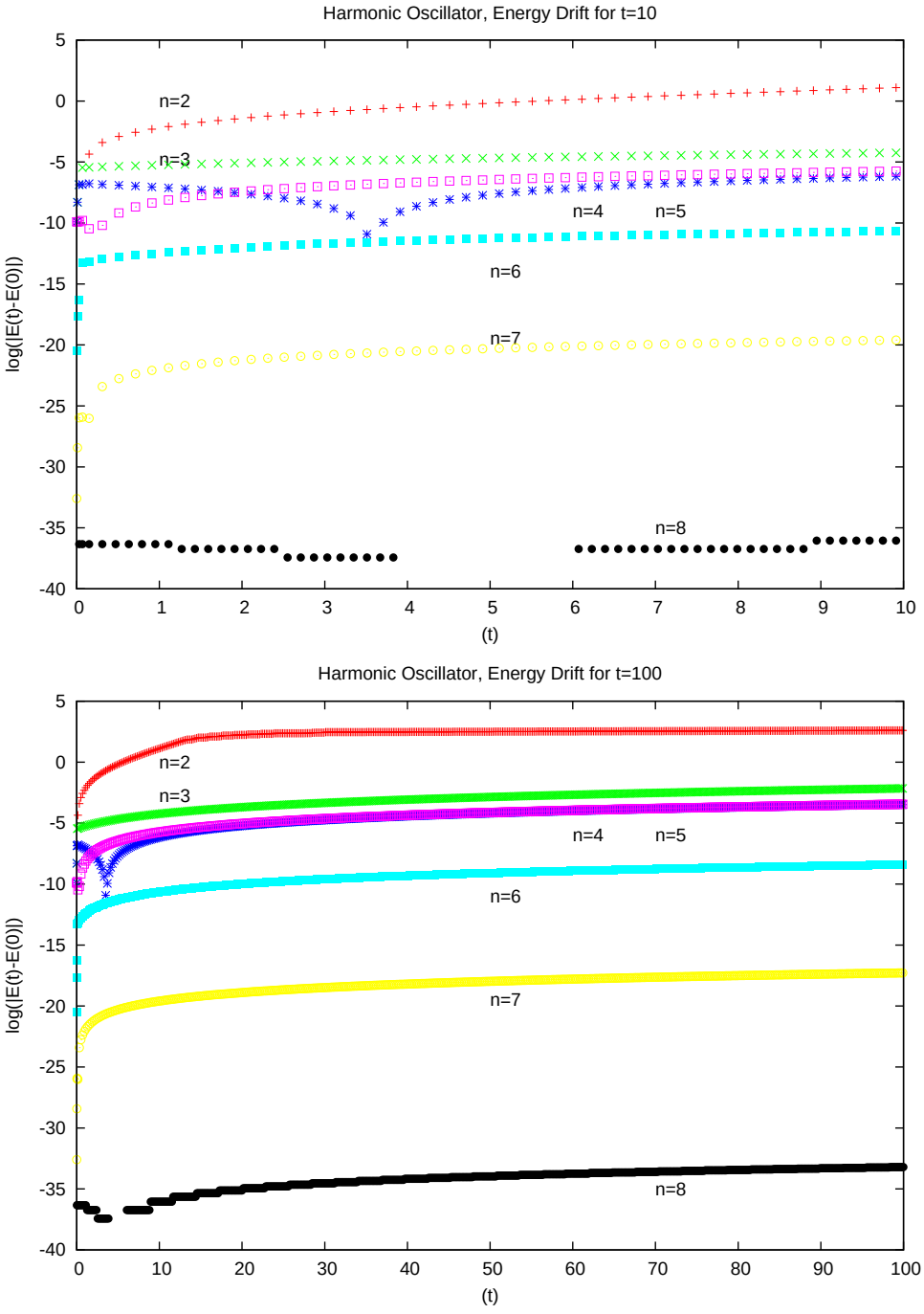
```
 if(h<min_h) {
         printf("Step error at time %lf. Exiting...\n",t);
         exit(1);
 };
 order=max_n; hpow=1.;
 // fill up u1[i], u2[i], x[i+1], y[i+1] matrices
 // up to order max_n
 for (i=0; i<max_n-1; i++){
  hpow*=h;
  u1[i]=(-omega2)*x[i];
  u2[i]=y[i];
  x[i+1]=(1./(i+1.))*u2[i];
  y[i+1]=(1./(i+1.))*u1[i];
  // series has converged ok?
  if(hpow*(fabs(x[i+1])+fabs(y[i+1]))<tol) {order=i+1; break;}
  order=i+1;
 }
 new_h=h;
 if(order<10) new_h=min(2.0*h,max_h);
 if(order>20) new_h=0.5*h;
 if(order==max_n) return -h;
 hpow=1.;  // make Taylor expansion & update x[], y[] matrices
 for(i=1; i<order; i++){
  hpow*=h;
  x[0]+=x[i]*hpow;
  y[0]+=y[i]*hpow;
 };
 printf("\ntime: %15.14lf\t order %d. step: %15.14lf\n",\
                 t,order,h);
 printf("E_0= %15.14lf\tE(t)= %15.14lf\t%15.14lf\n",\
                 E_0,Energy(),fabs(Energy()-E_0));
 return new_h;
}; // end of do_a_step()
//-----------------------------------------------------------------
int main(){
 double time, h, ah;
 double x0,y0;// for pgplot panels

 x0=x[0]= 0.;
 y0=y[0]= 1.;
 time = 200;
 h=0.01;
 t=0.;
 E_0= Energy();
 while(t<time){
  if ((ah=do_a_step(h))>0) {t+=h; h=ah;}
  else {h=-ah;};
 };
 return 0;
};
```



Σχήμα 1: Λύση του προβλήματος του αρμονικού ταλαντωτή. Για αρχικές συνθήκες και τιμές των παραμέτρων, βλ. κείμενο.



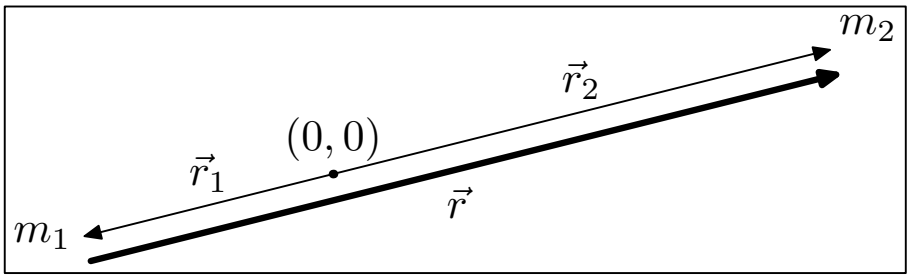
Σχήμα 2: Αρμονικός ταλαντωτής: χρόνος εκτέλεσης $t = 200$ χρονικές μονάδες.



Σχήμα 3: Αρμονικός ταλαντωτής: μετατόπιση (drift) της τιμής της ενέργειας, για διάφορες τάξεις των σειρών που χρησιμοποιούνται στην ολοκλήρωση, για χρόνο $t = 10$ (επάνω) και $t = 100$ χρονικές μονάδες (κάτω).

3.2 Το Πρόβλημα των 2 Σωμάτων (2–body problem).

3.2.1 Στοιχεία από τη Θεωρία και αναλυτικές πράξεις.



Σχήμα 4: Το Πρόβλημα των 2 Σωμάτων.

Η θεωρητική αντιμετώπιση του προβλήματος αναλύεται πλήρως στα [Χατζηδημητρίου, 2000, Hirsch & Smale, 1974]. Εδώ θα μας απασχολήσει ο καθορισμός της υπολογιστικής διαδικασίας.

- Κέντρο μάζας στο $(0,0)$.
- Εξισώσεις της κίνησης:

$$m_1\ddot{\vec{r}}_1 = -Gm_1m_2\frac{\vec{r}_1-\vec{r}_2}{r_{12}^3} \tag{45}$$

$$m_2\ddot{\vec{r}}_2 = -Gm_2m_1\frac{\vec{r}_2-\vec{r}_1}{r_{21}^3} \tag{46}$$

$$\vec{r}_i = \begin{bmatrix} x_i \\ \\ y_i \end{bmatrix}, \quad r_{ij} = r_{ji} = \sqrt{(x_i-x_j)^2+(y_i-y_j)^2}, \quad i,j = 1,2\ . \tag{47}$$

- Σε σύστημα μονάδων όπου $G=1$, θα είναι

$$\ddot{\vec{r}}_1 = -m_2\frac{\vec{r}_1-\vec{r}_2}{r_{12}^3} \tag{48}$$

$$\ddot{\vec{r}}_2 = -m_1\frac{\vec{r}_2-\vec{r}_1}{r_{21}^3} \tag{49}$$

- Με την παρατήρηση από το Σχ. 4 ότι $\vec{r}=\vec{r}_2-\vec{r}_1$, μπορούμε να γράψουμε τις εξισώσεις της κίνησης ως:

$$\ddot{\vec{r}} = -\frac{1}{r^3}\mu\vec{r} \tag{50}$$

  όπου $\mu = m_1+m_2$ και μπορούμε, χωρίς περιορισμό της γενικότητας, να θεωρήσουμε την περίπτωση όπου $m_1+m_2=1$, για απλούστευση του κώδικα και των πράξεων.

- Επομένως, το πρόβλημα αρχικών τιμών που έχουμε να λύσουμε είναι

$$\ddot{\vec{r}} = -\frac{1}{r^3}\vec{r} \tag{51}$$

με $\vec{r}(0) = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$, $\dot{\vec{r}}(0) = \begin{bmatrix} \dot{x}_0 \\ \dot{y}_0 \end{bmatrix}$, $r=\sqrt{x^2+y^2}$, $x=x_2-x_1$, $y=y_2-y_1$.

Εφαρμόζοντας τη μεθοδολογία της Υποενότητας 2.4, ανάγουμε το σύστημα (51) σε πρωτοβάθμιο:

$$\begin{aligned} \dot{x} &= z \\ \dot{y} &= w \\ \dot{z} &= -\frac{x}{r^3} \\ \dot{w} &= -\frac{y}{r^3} \end{aligned} \tag{52}$$

Οι βοηθητικές συναρτήσεις $U_k,\ k = 0 \ldots 8$ που θα χρησιμοποιήσουμε δίνονται από τις

$$\begin{aligned} U_0 &= z \\ U_1 &= w \\ U_2 &= x \\ U_3 &= y \\ U_4 &\rightarrow x^2 \\ U_5 &\rightarrow y^2 \\ U_6 &\rightarrow \left(x^2+y^2\right)^{3/2} \\ U_7 &\rightarrow x/r^3 \\ U_8 &\rightarrow y/r^3 \end{aligned} \tag{53}$$

και οι κανονικοποιημένες παράγωγοι τάξης $i+1$, θα υπολογίζονται για τις θέσεις και τις ταχύτητες από τις σχέσεις

$$\begin{aligned} x^{[i+1]} &= \frac{1}{(i+1)} U_0^{[i]} \\ y^{[i+1]} &= \frac{1}{(i+1)} U_1^{[i]} \\ z^{[i+1]} &= -\frac{1}{(i+1)} U_7^{[i]} \\ w^{[i+1]} &= -\frac{1}{(i+1)} U_8^{[i]} \end{aligned} \tag{54}$$

3.2.2 Κώδικας.

Παραθέτουμε τον υπολογιστικό πυρήνα για το συγκεκριμένο πρόβλημα

```
double do_a_step(double hstep){
 double h=hstep;
 double new_h;  // new timestep
 int i,j,loopsdone;
 double sum=0.;
 double hpow;

 if(h<min_h) {
        printf("Step error at time %lf. Exiting...\n",t);
```

```
        exit(1);
};

loopsdone=max_n;
hpow=1.;
// fill up u[][i], x[i+1], y[i+1], z[i+1], w[i+1] matrices
// up to order max_n
for (i=0; i<max_n-1; i++){
 hpow*=h;

 u[0][i]=z[i];                                          // x'
 u[1][i]=w[i];                                          // y'
 u[2][i]=x[i];
 u[3][i]=y[i];
 sum=0.;
 for (j=0; j<=i; j++) sum+=u[2][i-j]*u[2][j];   //x*x
 u[4][i]=sum;
 sum=0.;
 for (j=0; j<=i; j++) sum+=u[3][i-j]*u[3][j];   //y*y
 u[5][i]=sum;

                                                        // r=sqrt(x^2+y^2)
 if (i==0)
  {u[6][0]=pow((u[4][0]+u[5][0]),3./2.);}       // r^3
 else {
  sum=0.;
  for (j=0; j<=i-1; j++)
   sum+= (1.5*i-j*(2.5))*(u[4][i-j]+u[5][i-j])*u[6][j];
  sum/=i*(u[4][0]+u[5][0]);
  u[6][i]=sum;
 };

 sum=0.;
 for (j=1; j<=i; j++) sum-=u[6][j]*u[7][i-j];   // quotient
 sum+=u[2][i];
 sum/=u[6][0];
 u[7][i]= sum;                                          // x/r^3

 sum=0.;
 for (j=1; j<=i; j++) sum-=u[6][j]*u[8][i-j];   // quotient
 sum+=u[3][i];
 sum/=u[6][0];
 u[8][i]= sum;                                          // y/r^3

 x[i+1]= u[0][i]/(i+1.);
 y[i+1]= u[1][i]/(i+1.);
 z[i+1]=-u[7][i]/(i+1.);
 w[i+1]=-u[8][i]/(i+1.);

 // series has converged ok?
```

```
  if(hpow*(fabs(x[i+1])+fabs(y[i+1])\
 +fabs(z[i+1])+fabs(w[i+1]))<tol)
    {loopsdone=i+1; break;}
  loopsdone=i+1;
 }
 new_h=h;
 if(loopsdone<10) new_h=min(2.0*h,max_h);
 if(loopsdone>20) new_h=0.5*h;
 if(loopsdone==max_n) return -h;

 // make Taylor expansion & update x[], y[], z[], w[] matrices
 hpow=1.;
 for(i=1; i<loopsdone; i++){
    hpow*=h;
        x[0]+=x[i]*hpow;
        y[0]+=y[i]*hpow;
        z[0]+=z[i]*hpow;
        w[0]+=w[i]*hpow;
 };
 printf("\ntime: %15.14lf\t order %d. step: %15.14lf\n",\
                  t,loopsdone,h);
 printf("E_0= %15.14lf\tE(t)= %15.14lf\t%15.14lf\n",\
                  E_0,Energy(),fabs(Energy()-E_0));
 return new_h;
}; // end of do_a_step()
```

- Έχοντας τα $\vec{r}\,|_{t=0}$, $\vec{r}^{(1)}\,|_{t=0}$, μπορούμε να υπολογίσουμε τα $\vec{r}^{(2)}$, $\vec{r}^{(3)}\ldots$ μέχρι την τάξη που χρειάζεται. Οι θέσεις και οι ταχύτητες μετά από κάποιο βήμα h θα δίνονται από τις σχέσεις:

$$\vec{r}(t+h) = \vec{r}(t) + h\vec{r}^{(1)}(t) + \frac{h^2}{2!}\vec{r}^{(2)}(t) + \frac{h^3}{3!}\vec{r}^{(3)}(t) + \ldots$$

$$\dot{\vec{r}}(t+h) = \vec{r}^{(1)}(t) + h\vec{r}^{(2)}(t) + \frac{h^2}{2!}\vec{r}^{(3)}(t) + \frac{h^3}{3!}\vec{r}^{(4)}(t) + \ldots$$

- Από τον όρο του υπολοίπου (remainder term), μπορούμε να καθορίσουμε το τοπικό σφάλμα αποκοπής, εκλέγοντας κατάλληλο βήμα h [Papadakos, 1983].

- Έχοντας βρει το $\vec{r}(t)$ (άρα και το $\dot{\vec{r}}(t)$) για κάποια χρονική στιγμή, μπορούμε να καθορίσουμε τις θέσεις $\vec{r}_1(t), \vec{r}_2(t)$ και τις ταχύτητες $\vec{r}_1(t), \vec{r}_2(t)$ των δύο σωμάτων, από το θεώρημα διατήρησης της ορμής ως προς το κέντρο μάζας και από τη σχέση $\vec{r} = \vec{r}_2 - \vec{r}_1$:

$$\left.\begin{array}{rcl} m_1\vec{r}_1 + m_2\vec{r}_2 &=& 0 \\ \vec{r}_2 - \vec{r}_1 &=& \vec{r} \end{array}\right\} \Rightarrow \vec{r}_1 = -\frac{m_2}{m_1}\vec{r}_2 \Rightarrow \vec{r}_2\left(1+\frac{m_2}{m_1}\right) = \vec{r} \Rightarrow$$

$$\begin{array}{rcl} \vec{r}_2 &=& \dfrac{1}{1+\left(\dfrac{m_2}{m_1}\right)}\vec{r} \;=\; \dfrac{m_1}{\mu}\vec{r} \\ \\ \vec{r}_1 &=& \vec{r}_2 - \vec{r} \end{array} \qquad (55)$$

και αντίστοιχα για τις ταχύτητες θα είναι

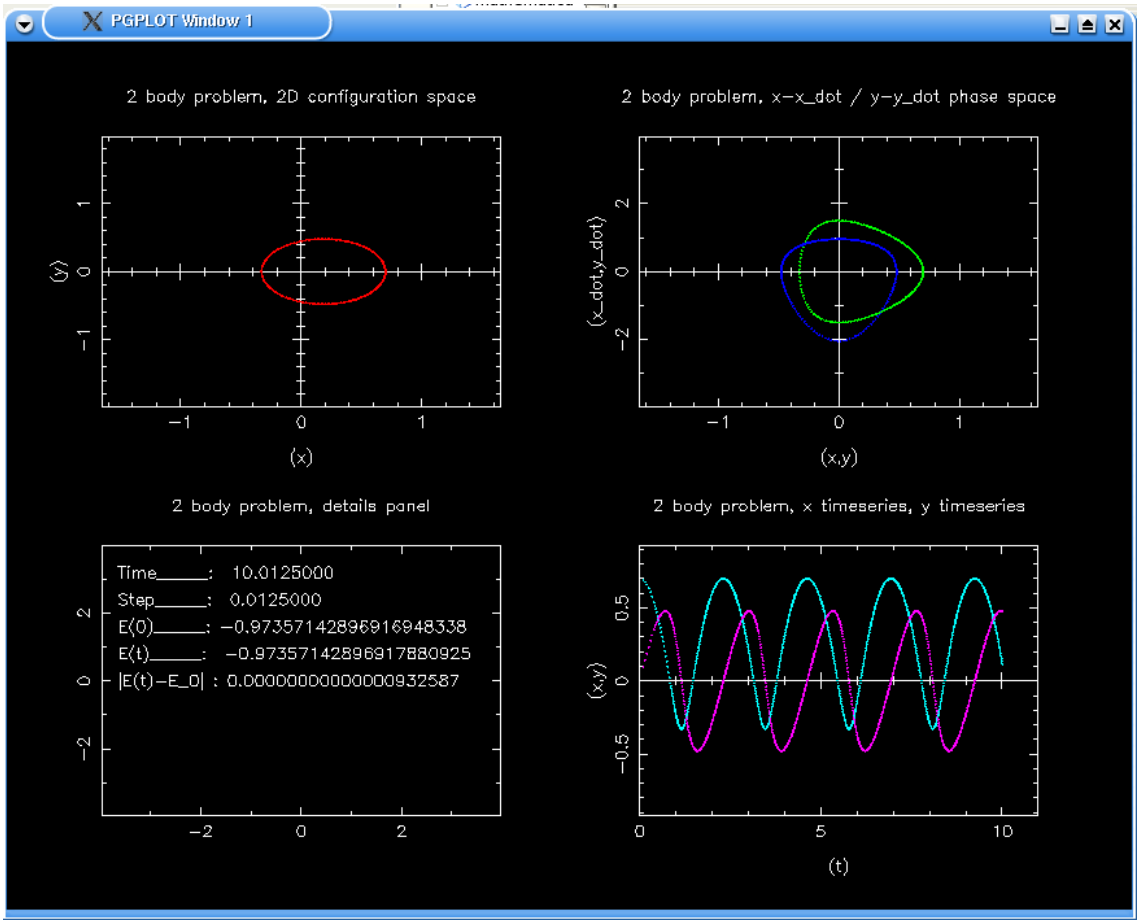
$$\begin{aligned} \dot{\vec{r}}_2 &= \frac{m_1}{\mu}\,\dot{\vec{r}} \\ \dot{\vec{r}}_1 &= \dot{\vec{r}}_2 - \dot{\vec{r}} \end{aligned} \tag{56}$$

- Μπορούμε να ελέγξουμε την αριθμητική ακρίβεια της μεθόδου χρησιμοποιώντας το ολοκλήρωμα της ενέργειας

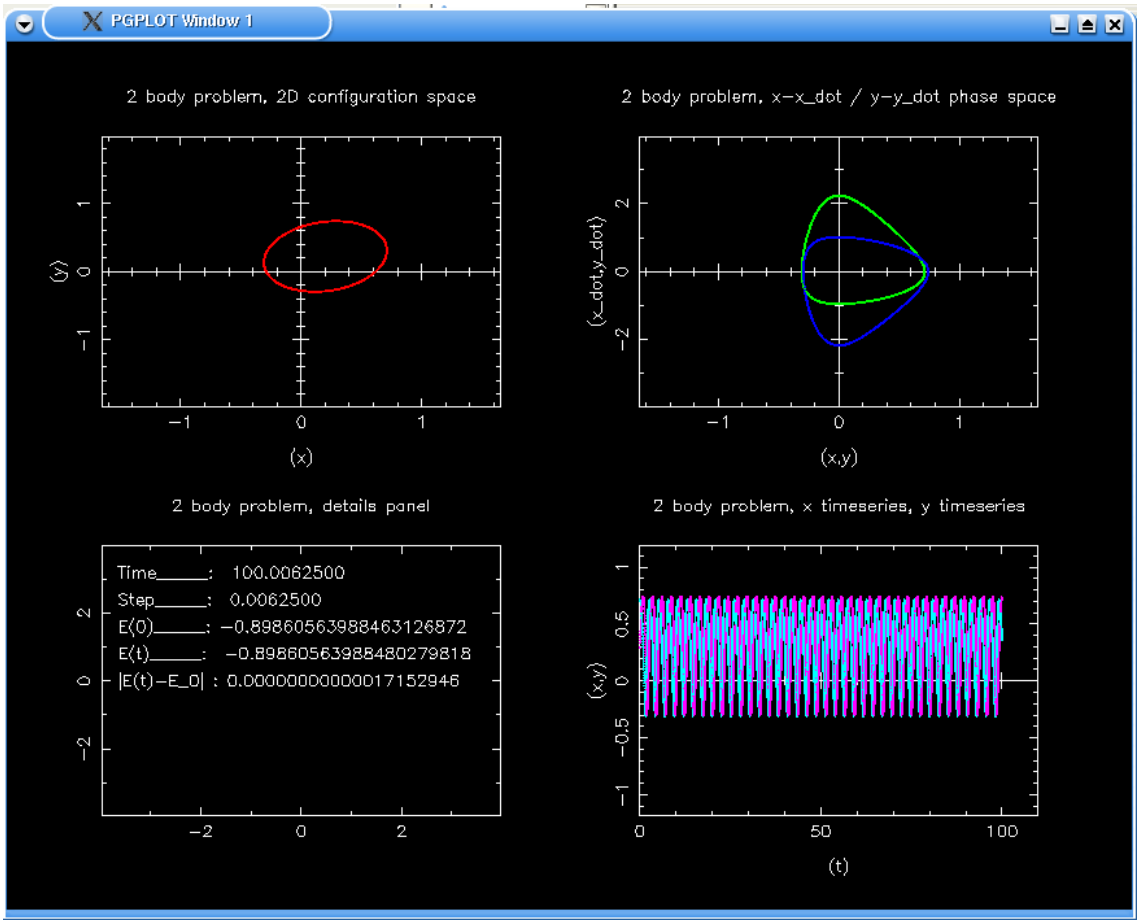
$$E_{ολ} = E_{κιν} + E_{δυν} =$$

$$= \frac{1}{2}m_1\dot{\vec{r}}_1^{\,2}(t) + \frac{1}{2}m_2\dot{\vec{r}}_2^{\,2}(t) - \frac{m_1 m_2}{r_{12}(t)} = \frac{1}{2}m_1\dot{\vec{r}}_1^{\,2}(0) + \frac{1}{2}m_2\dot{\vec{r}}_2^{\,2}(0) - \frac{m_1 m_2}{r_{12}(0)}\ . \tag{57}$$

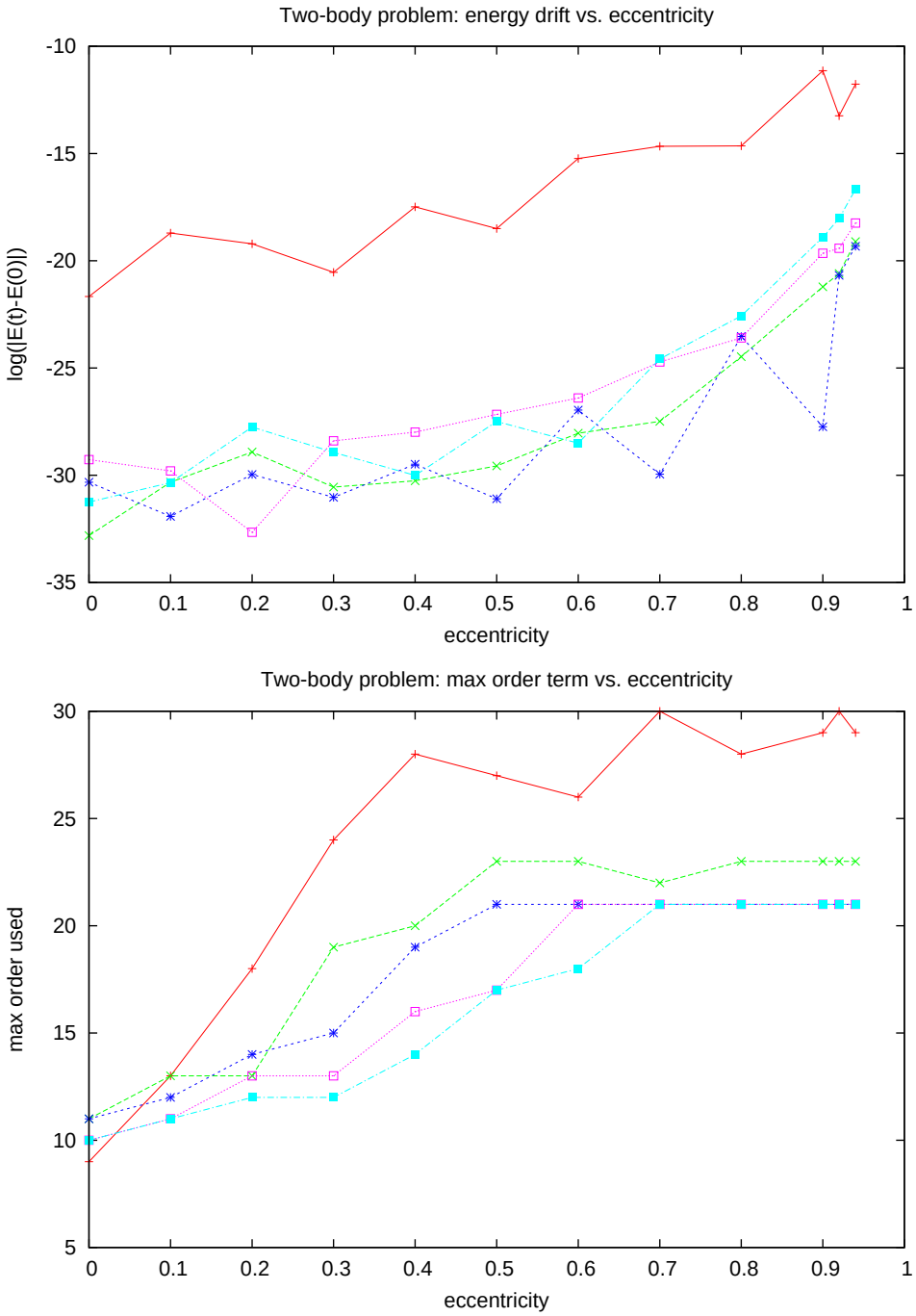
3.2.3 Αποτελέσματα.



Σχήμα 5: Το Πρόβλημα των 2 Σωμάτων. Αρχικές συνθήκες: μεγάλος ημιάξονας 1, εκκεντρότητα 0.3 ($x_0 = 0.7,\ y_0 = 0,\ z_0 = 0,\ w_0 = 0.953939201$).



Σχήμα 6: Το Πρόβλημα των 2 Σωμάτων για $t = 100$. Αρχικές συνθήκες: $x_0 = 0.7,\ y_0 = 0.1,\ z_0 = 0.1,\ w_0 = 0.953939201$.



Σχήμα 7: Το Πρόβλημα των 2 Σωμάτων: μετατόπιση (drift) της ενέργειας σε σχέση με την εκκεντρότητα e (επάνω) και μεταβολή του χρόνου εκτέλεσης με την εκκεντρότητα (κάτω) για χρόνο $T = 10 \cdot 2\pi$.

3.3 Το Πρόβλημα των 3 Σωμάτων (3–body problem).

3.3.1 Στοιχεία από τη Θεωρία και αναλυτικές πράξεις.

Από τα διασημότερα προβλήματα της μηχανικής. Διεξοδική παρουσίαση του προβλήματος γίνεται στα [Szebehely, 1967], [Roy, 1982] και [Goldstein, 1980].

- Κέντρο μάζας στο $(0,0,0)$.
- Εξισώσεις της κίνησης:

$$\begin{bmatrix} \ddot{\vec{r}}_1 \\ \ddot{\vec{r}}_2 \\ \ddot{\vec{r}}_3 \end{bmatrix} = \begin{bmatrix} -\left(\frac{m_2}{r_{12}^3}+\frac{m_3}{r_{13}^3}\right) & \frac{m_2}{r_{12}^3} & \frac{m_3}{r_{13}^3} \\ \frac{m_1}{r_{21}^3} & -\left(\frac{m_1}{r_{21}^3}+\frac{m_3}{r_{23}^3}\right) & \frac{m_3}{r_{23}^3} \\ \frac{m_1}{r_{31}^3} & \frac{m_2}{r_{32}^3} & -\left(\frac{m_1}{r_{31}^3}+\frac{m_2}{r_{32}^3}\right) \end{bmatrix} \cdot \begin{bmatrix} \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{bmatrix} \quad (58)$$

και

$$\vec{r}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}, \ \hat{R} = [R_{ij}],\ R_{ij} = R_{ji} = (\vec{r}_i - \vec{r}_j)\cdot(\vec{r}_i - \vec{r}_j) = (\vec{r}_i - \vec{r}_j)^2 \ , \ i,j = 1,2,3 \ , \quad (59)$$

$$\hat{r} = [r_{ij}],\ r_{ij} = r_{ji} = \sqrt{R_{ij}} = \sqrt{R_{ji}} = \sqrt{(x_i-x_j)^2+(y_i-y_j)^2+(z_i-z_j)^2}\ , \ i,j = 1,2,3 \ . \quad (60)$$

- Χωρίς περιορισμό της γενικότητας, θεωρούμε ότι $m_1 + m_2 + m_3 = 1$.
- Έστω ότι το τρίτο σώμα είναι ο Ήλιος. Οι συνιστώσες της θέσης του τρίτου σώματος προσδιορίζονται από τις αντίστοιχες συνιστώσες των άλλων δύο σωμάτων ως προς το κέντρο μάζας και οι συνιστώσες της ταχύτητάς του βρίσκονται με την εφαρμογή του Θεωρήματος διατήρησης της ολικής ορμής ως προς το κέντρο μάζας.
- Χρησιμοποιώντας τον κανόνα άθροισης του Einstein είναι $m_i\vec{r}_i = 0$ και $m_i\dot{\vec{r}}_i = 0$:

$$\begin{array}{ccc} m_i x_i = 0 & & m_i \dot{x}_i = 0 \\ m_i y_i = 0 & and & m_i \dot{y}_i = 0 \\ m_i z_i = 0 & & m_i \dot{z}_i = 0 \end{array} \quad (61)$$

οπότε

$$\begin{array}{ccc} x_3 = -\frac{1}{m_3}(m_1x_1 + m_2x_2) & & \dot{x_3} = -\frac{1}{m_3}(m_1\dot{x_1} + m_2\dot{x_2}) \\ y_3 = -\frac{1}{m_3}(m_1y_1 + m_2y_2) & and & \dot{y_3} = -\frac{1}{m_3}(m_1\dot{y_1} + m_2\dot{y_2}) \\ z_3 = -\frac{1}{m_3}(m_1z_1 + m_2z_2) & & \dot{z_3} = -\frac{1}{m_3}(m_1\dot{z_1} + m_2\dot{z_2}) \end{array} \quad (62)$$

Έχουμε λοιπόν να υπολογίσουμε, στη γενική περίπτωση, 12 συνολικά μεταβλητές, τις

$$x_1, y_1, z_1, \quad x_2, y_2, z_2, \quad \dot{x_1}, \dot{y_1}, \dot{z_1}, \quad \dot{x_2}, \dot{y_2}, \dot{z_2} \ ,$$

ή 8 εάν περιοριστούμε στο επίπεδο πρόβλημα (θεωρώντας όλα τα $z_i = 0$):

$$x_1, y_1, \quad x_2, y_2, \quad \dot{x_1}, \dot{y_1}, \quad \dot{x_2}, \dot{y_2} \ .$$

- Με το να χρησιμοποιήσουμε το ολοκλήρωμα της κίνησης, $\vec{J}_{o\lambda} = \vec{0}$, προφανώς χάνουμε τη δυνατότητα επαλήθευσης με αυτό των λύσεων που βρίσκουμε. Για τον υπολογισμό του λάθους και τον έλεγχο των αριθμητικών τιμών που προκύπτουν, θα χρησιμοποιήσουμε το ολοκλήρωμα της ενέργειας, $E_{o\lambda}$.

3.3.2 Εξισώσεις και υπολογιστικός πυρήνας για το επίπεδο πρόβλημα των τριών σωμάτων.

$$m_1 + m_2 + m_3 = 1 \tag{63}$$

$$M_{ij} = m_i + m_j = M_{ji} \tag{64}$$

$$\mu_{ij} = \frac{m_i}{m_j} \neq \mu_{ji} = \frac{m_j}{m_i} \tag{65}$$

$$N_{ij} = 1 + \mu_{ij} \neq N_{ji} = 1 + \mu_{ji} \tag{66}$$

$$R_{13} = (N_{13}x_1 + \mu_{23}x_2)^2 + (N_{13}y_1 + \mu_{23}y_2)^2 \tag{67}$$

$$R_{23} = (N_{23}x_2 + \mu_{13}x_1)^2 + (N_{23}y_2 + \mu_{13}y_1)^2 \tag{68}$$

$$R_{12} = (x_1 - x_2)^2 + (y_1 - y_2)^2 \tag{69}$$

$$\ddot{x}_1 = \frac{m_2(x_2 - x_1)}{R_{12}^{3/2}} - \frac{M_{13}x_1 + m_2x_2}{R_{13}^{3/2}} \tag{70}$$

$$\ddot{y}_1 = \frac{m_2(y_2 - y_1)}{R_{12}^{3/2}} - \frac{M_{13}y_1 + m_2y_2}{R_{13}^{3/2}} \tag{71}$$

$$\ddot{x}_2 = \frac{m_1(x_1 - x_2)}{R_{12}^{3/2}} - \frac{M_{32}x_2 + m_1x_1}{R_{23}^{3/2}} \tag{72}$$

$$\ddot{y}_2 = \frac{m_1(y_1 - y_2)}{R_{12}^{3/2}} - \frac{M_{32}y_2 + m_1y_1}{R_{23}^{3/2}} \tag{73}$$

```
//--------------------------------3 body planar core
double do_a_step(double hstep){
 double h=hstep;
 double new_h;  // new timestep
 int i,j,loopsdone=max_n;
 double sum=0.;
 double hpow=1.;

 if(h<min_h) {
        printf("Step error at time %lf. Exiting...\n",t);
        exit(1);
```

```
 };

// fill up u[][i], x[1,2,3][i+1], y[1,2,3][i+1], z[1,2,3][i+1],
//                 xd[1,2,3][i+1], yd[1,2,3][i+1], zd[1,2,3][i+1]
//                 matrices up to order max_n
 for (i=0; i<max_n-1; i++){
  hpow*=h;

  u[ 0][i]=xd[0][i];              //xd1
  u[ 1][i]=xd[1][i];              //xd2
  u[ 2][i]=yd[0][i];              //yd1
  u[ 3][i]=yd[1][i];              //yd2
  u[ 4][i]= x[0][i];              //x1
  u[ 5][i]= x[1][i];              //x2
  u[ 6][i]= y[0][i];              //y1
  u[ 7][i]= y[1][i];              //y2

  sum=0.; for (j=0; j<=i; j++)
          sum+=(u[5][i-j]-u[4][i-j])*(u[5][j]-u[4][j]); //(x2-x1)^2
  u[ 8][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
          sum+=(u[7][i-j]-u[6][i-j])*(u[7][j]-u[6][j]); //(y2-y1)^2
  u[ 9][i]= sum;

  if (i==0){
          u[10][0]=pow(u[ 8][i] + u[ 9][i], 1.5);        //R_12^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[ 8][i-j] + u[ 9][i-j])*u[10][j];
   sum/=(i*(u[ 8][0] + u[ 9][0]));
   u[10][i]=sum;
  };

  sum=0.; for (j=1; j<=i; j++) sum-=u[10][j]*u[11][i-j];// quotient
  sum+=u[5][i]-u[4][i]; sum/=u[10][0];                  // 1a
  u[11][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(N13*u[4][i-j]+mu23*u[5][i-j])*(N13*u[4][j]+mu23*u[5][j]);
  u[12][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(N13*u[6][i-j]+mu23*u[7][i-j])*(N13*u[6][j]+mu23*u[7][j]);
  u[13][i]= sum;

  if (i==0){
```

```
        u[14][0]=pow(u[12][0] + u[13][0], 1.5);          //R_13^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[12][i-j] + u[13][i-j])*u[14][j];
   sum/=(i*(u[12][0] + u[13][0]));
   u[14][i]=sum;
  };

  sum=0.; for (j=1; j<=i; j++) sum-=u[14][j]*u[15][i-j];// quotient
  sum+=M31*u[4][i]+m2*u[5][i]; sum/=u[14][0];              // 1b
  u[15][i]= sum;

//-------------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[10][j]*u[16][i-j];// quotient
  sum+=u[7][i]-u[6][i]; sum/=u[10][0];                    // 2a
  u[16][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[14][j]*u[17][i-j];// quotient
  sum+=M31*u[6][i]+m2*u[7][i];
  sum/=u[14][0];                                          // 2b
  u[17][i]= sum;

//-------------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[10][j]*u[18][i-j];// quotient
  sum+=u[4][i]-u[5][i]; sum/=u[10][0];                    // 3a
  u[18][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(N23*u[5][i-j]+mu13*u[4][i-j])*(N23*u[5][j]+mu13*u[4][j]);
  u[19][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(N23*u[7][i-j]+mu13*u[6][i-j])*(N23*u[7][j]+mu13*u[6][j]);
  u[20][i]= sum;

  if (i==0){
   u[21][0]=pow(u[19][0] + u[20][0], 1.5);          //R_23^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[19][i-j] + u[20][i-j])*u[21][j];
   sum/=(i*(u[19][0] + u[20][0]));
   u[21][i]=sum;
  };

  sum=0.; for (j=1; j<=i; j++) sum-=u[21][j]*u[22][i-j];// quotient
  sum+=M32*u[5][i]+m1*u[4][i];
```

```
  sum/=u[21][0];                                          // 3b
  u[22][i]= sum;

//-------------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[10][j]*u[23][i-j];// quotient
  sum+=u[6][i]-u[7][i];   sum/=u[10][0];
  u[23][i]= sum;                                          // 4a

  sum=0.; for (j=1; j<=i; j++) sum-=u[21][j]*u[24][i-j];// quotient
  sum+=M32*u[7][i]+m1*u[6][i];
  sum/=u[21][0];                                          // 4b
  u[24][i]= sum;

//-------------------------------------------------------------------------

  x[0][i+1]=u[0][i]/(i+1.);
  x[1][i+1]=u[1][i]/(i+1.);
  y[0][i+1]=u[2][i]/(i+1.);
  y[1][i+1]=u[3][i]/(i+1.);

  xd[0][i+1]=(m2*u[11][i]-u[15][i])/(i+1.);
  xd[1][i+1]=(m1*u[18][i]-u[22][i])/(i+1.);
  yd[0][i+1]=(m2*u[16][i]-u[17][i])/(i+1.);
  yd[1][i+1]=(m1*u[23][i]-u[24][i])/(i+1.);

  if(hpow*(fabs( x[0][i+1])+fabs( x[1][i+1])+\
           fabs( y[0][i+1])+fabs( y[1][i+1])+\
           fabs(xd[0][i+1])+fabs(xd[1][i+1])+\
           fabs(yd[0][i+1])+fabs(yd[1][i+1]))\
     <tol) {loopsdone=i+1; break;}
  loopsdone=i+1;
 }

 new_h=h;
 if(loopsdone<10) new_h=min(1.5*h,max_h);
 if(loopsdone>20) new_h=0.5*h;
 if(loopsdone==max_n) return -h;

// update positions & velocities of m1, m2
 hpow=1.;
 for(i=1; i<loopsdone; i++){
  hpow*=h;
  x[0][0]+=   x[0][i]*hpow;
  x[1][0]+=   x[1][i]*hpow;
  y[0][0]+=   y[0][i]*hpow;
  y[1][0]+=   y[1][i]*hpow;
  xd[0][0]+=xd[0][i]*hpow;
  xd[1][0]+=xd[1][i]*hpow;
  yd[0][0]+=yd[0][i]*hpow;
  yd[1][0]+=yd[1][i]*hpow;
```

```
 };
 //update position & velocity of the third body
 x[2][0]=    -(m1* x[0][0]+m2* x[1][0])/m3;
 y[2][0]=    -(m1* y[0][0]+m2* y[1][0])/m3;
 xd[2][0]=   -(m1*xd[0][0]+m2*xd[1][0])/m3;
 yd[2][0]=   -(m1*yd[0][0]+m2*yd[1][0])/m3;

#ifndef pgplot_graphics
 printf("E_0= %3.15lf\tE(t)= %15.14lf\t%15.14lf\n",\
                  E_0,Energy(),fabs(Energy()-E_0));
 printf("time: %5.20lf\t order %d. step: %5.20lf\n",\
                  t,loopsdone,h);
#endif
 return new_h;
}; // end of do_a_step()
```

3.3.3 Εξισώσεις και υπολογιστικός πυρήνας για το πλήρες πρόβλημα των τριών σωμάτων.

$$m_1 + m_2 + m_3 = 1 \tag{74}$$

$$M_{ij} = m_i + m_j = M_{ji} \tag{75}$$

$$\mu_{ij} = \frac{m_i}{m_j} \neq \mu_{ji} = \frac{m_j}{m_i} \tag{76}$$

$$N_{ij} = 1 + \mu_{ij} \neq N_{ji} = 1 + \mu_{ji} \tag{77}$$

$$R_{13} = (N_{13}x_1 + \mu_{23}x_2)^2 + (N_{13}y_1 + \mu_{23}y_2)^2 + (N_{13}z_1 + \mu_{23}z_2)^2 \tag{78}$$

$$R_{23} = (N_{23}x_2 + \mu_{13}x_1)^2 + (N_{23}y_2 + \mu_{13}y_1)^2 + (N_{23}z_2 + \mu_{13}z_1)^2 \tag{79}$$

$$R_{12} = (x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2 \tag{80}$$

$$\ddot{x}_1 = \frac{m_2(x_2 - x_1)}{R_{12}^{3/2}} - \frac{M_{13}x_1 + m_2x_2}{R_{13}^{3/2}} \tag{81}$$

$$\ddot{y}_1 = \frac{m_2(y_2 - y_1)}{R_{12}^{3/2}} - \frac{M_{13}y_1 + m_2y_2}{R_{13}^{3/2}} \tag{82}$$

$$\ddot{z}_1 = \frac{m_2(z_2 - z_1)}{R_{12}^{3/2}} - \frac{M_{13}z_1 + m_2z_2}{R_{13}^{3/2}} \tag{83}$$

$$\ddot{x}_2 = \frac{m_1(x_1 - x_2)}{R_{12}^{3/2}} - \frac{M_{32}x_2 + m_1x_1}{R_{23}^{3/2}} \tag{84}$$

$$\ddot{y}_2 = \frac{m_1(y_1 - y_2)}{R_{12}^{3/2}} - \frac{M_{32}y_2 + m_1y_1}{R_{23}^{3/2}} \tag{85}$$

$$\ddot{z}_2 = \frac{m_1(z_1 - z_2)}{R_{12}^{3/2}} - \frac{M_{32}z_2 + m_1z_1}{R_{23}^{3/2}} \tag{86}$$

```
//-------------------------3 body full core
double do_a_step(double hstep){
 double h=hstep;
```

```
 double new_h;  // new timestep
 int i,j,loopsdone=max_n;
 double sum=0.;
 double hpow=1.;

 if(h<min_h) {
        printf("Step error at time %lf. Exiting...\n",t);
        exit(1);
 };

// fill up u[][i],  x[1,2,3][i+1],  y[1,2,3][i+1],  z[1,2,3][i+1],
//                 xd[1,2,3][i+1], yd[1,2,3][i+1], zd[1,2,3][i+1]
//                 matrices up to order max_n
 for (i=0; i<max_n-1; i++){
  hpow*=h;

  u[ 0][i]=xd[0][i];            //xd1
  u[ 1][i]=xd[1][i];            //xd2
  u[ 2][i]=yd[0][i];            //yd1
  u[ 3][i]=yd[1][i];            //yd2
  u[ 4][i]=zd[0][i];            //zd1
  u[ 5][i]=zd[1][i];            //zd2

  u[ 6][i]= x[0][i];            //x1
  u[ 7][i]= x[1][i];            //x2
  u[ 8][i]= y[0][i];            //y1
  u[ 9][i]= y[1][i];            //y2
  u[10][i]= z[0][i];            //z1
  u[11][i]= z[1][i];            //z2
//-------------------------------------------------------------------
  // construct R_12^(3/2), R_13^(3/2). R_23^(3/2)
  // form 1a, 2a, 1b, 2b, etc.
//-------------------------------------------------------------------
  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(u[ 7][i-j]-u[ 6][i-j])*(u[ 7][j]-u[ 6][j]);     //(x2-x1)^2
  u[12][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(u[ 9][i-j]-u[ 8][i-j])*(u[ 9][j]-u[ 8][j]);     //(y2-y1)^2
  u[13][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(u[11][i-j]-u[10][i-j])*(u[11][j]-u[10][j]);     //(z2-z1)^2
  u[14][i]= sum;
//-------------------------------------------------------------------
  if (i==0){
     u[15][0]=pow(u[12][0]+u[13][0]+u[14][0], 1.5);      //R_12^(3/2)
```

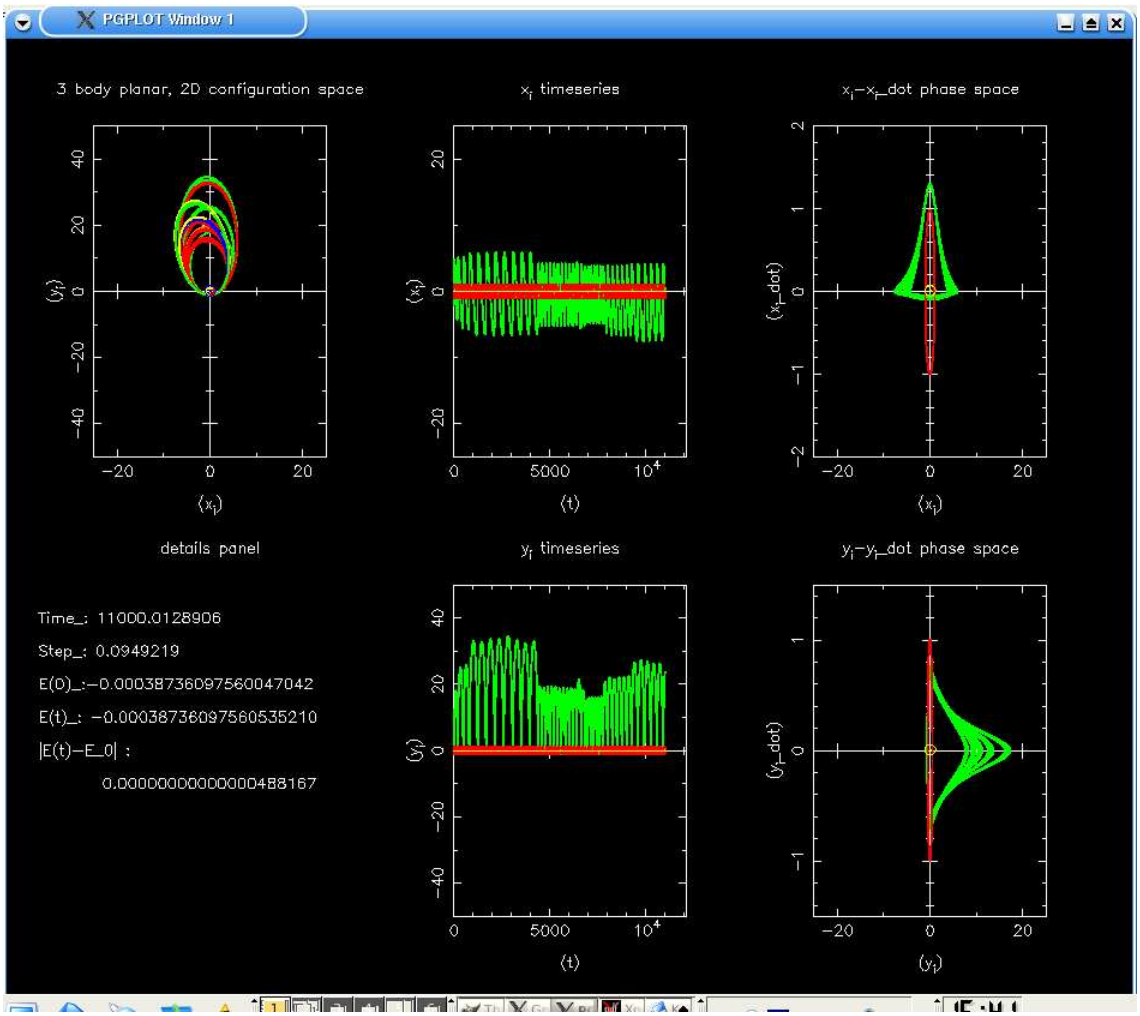
```
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[12][i-j] +\
    u[13][i-j] + u[14][i-j])*u[15][j];
   sum/=(i*(u[12][0] + u[13][0] + u[14][0]));
   u[15][i]=sum;
  };
//-------------------------------------------------------------------------
  sum=0.; for (j=0; j<=i; j++)
   sum+=(N13*u[ 6][i-j]+mu23*u[ 7][i-j])*(N13*u[ 6][j]+mu23*u[ 7][j]);
                                          //(N13*x1+mu23*x2)^2
  u[16][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
   sum+=(N13*u[ 8][i-j]+mu23*u[ 9][i-j])*(N13*u[ 8][j]+mu23*u[ 9][j]);
                                          //(N13*y1+mu23*y2)^2
  u[17][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
   sum+=(N13*u[10][i-j]+mu23*u[11][i-j])*(N13*u[10][j]+mu23*u[11][j]);
                                          //(N13*z1+mu23*z2)^2
  u[18][i]= sum;
//-------------------------------------------------------------------------
  if (i==0){
     u[19][0]=pow(u[16][0] + u[17][0] + u[18][0], 1.5);     //R_13^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[16][i-j] +\
    u[17][i-j] + u[18][i-j])*u[19][j];
   sum/=(i*(u[16][0] + u[17][0] + u[18][0]));
   u[19][i]=sum;
  };
//-------------------------------------------------------------------------
  sum=0.; for (j=0; j<=i; j++)
   sum+=(mu13*u[ 6][i-j]+N23*u[ 7][i-j])*(mu13*u[ 6][j]+N23*u[ 7][j]);
                                          //(mu13*x1+N23*x2)^2
  u[20][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
   sum+=(mu13*u[ 8][i-j]+N23*u[ 9][i-j])*(mu13*u[ 8][j]+N23*u[ 9][j]);
                                          //(mu13*y1+N23*y2)^2
  u[21][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
   sum+=(mu13*u[10][i-j]+N23*u[11][i-j])*(mu13*u[10][j]+N23*u[11][j]);
                                          //(mu13*z1+N23*z2)^2
  u[22][i]= sum;
//-------------------------------------------------------------------------
```

```
  if (i==0){
     u[23][0]=pow(u[20][0] + u[21][0] + u[22][0], 1.5);     //R_23^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[20][i-j] +\
    u[21][i-j] + u[22][i-j])*u[23][j];
   sum/=(i*(u[20][0] + u[21][0] + u[22][0]));
   u[23][i]=sum;
  };
//-------------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[24][i-j];// quotient
  sum+=u[ 7][i]-u[ 6][i]; sum/=u[15][0];                // 1a
  u[24][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[25][i-j];// quotient
  sum+=u[ 9][i]-u[ 8][i]; sum/=u[15][0];                // 2a
  u[25][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[26][i-j];// quotient
  sum+=u[11][i]-u[10][i]; sum/=u[15][0];                // 3a
  u[26][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[27][i-j];// quotient
  sum+=u[ 6][i]-u[ 7][i]; sum/=u[15][0];                // 4a
  u[27][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[28][i-j];// quotient
  sum+=u[ 8][i]-u[ 9][i]; sum/=u[15][0];                // 5a
  u[28][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[29][i-j];// quotient
  sum+=u[10][i]-u[11][i]; sum/=u[15][0];                // 6a
  u[29][i]= sum;
//-------------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[19][j]*u[30][i-j];// quotient
  sum+=M13*u[ 6][i]+m2*u[ 7][i]; sum/=u[19][0];         // 1b
  u[30][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[19][j]*u[31][i-j];// quotient
  sum+=M13*u[ 8][i]+m2*u[ 9][i]; sum/=u[19][0];         // 2b
  u[31][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[19][j]*u[32][i-j];// quotient
  sum+=M13*u[10][i]+m2*u[11][i]; sum/=u[19][0];         // 3b
  u[32][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[23][j]*u[33][i-j];// quotient
  sum+=m1*u[ 6][i]+M23*u[ 7][i]; sum/=u[23][0];         // 4b
  u[33][i]= sum;
```

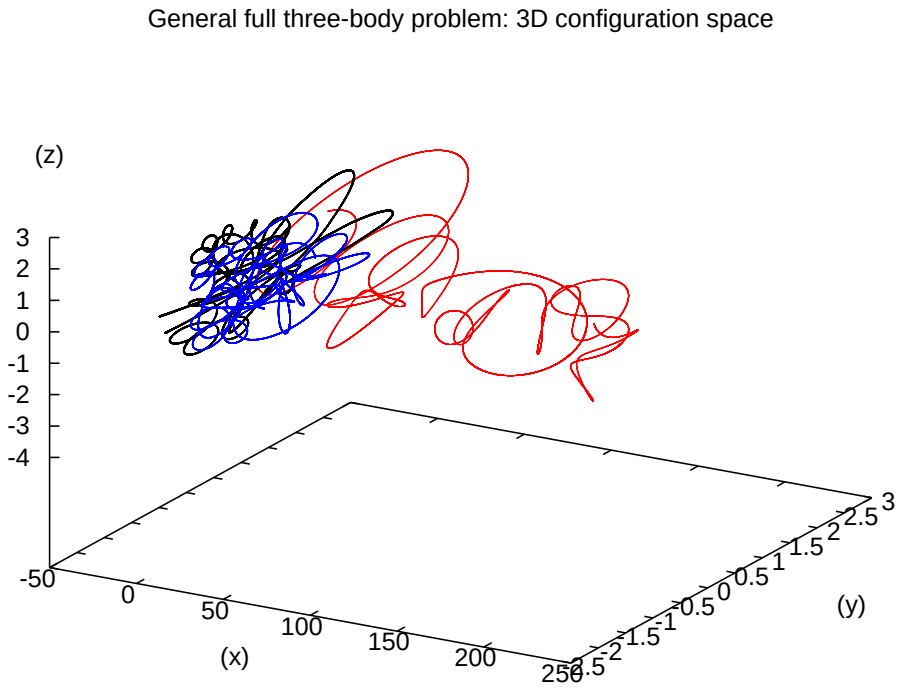
```
  sum=0.; for (j=1; j<=i; j++) sum-=u[23][j]*u[34][i-j];// quotient
  sum+=m1*u[ 8][i]+M23*u[ 9][i]; sum/=u[23][0];          // 5b
  u[34][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[23][j]*u[35][i-j];// quotient
  sum+=m1*u[10][i]+M23*u[11][i]; sum/=u[23][0];          // 6b
  u[35][i]= sum;
//-------------------------------------------------------------------
  x[0][i+1]=u[0][i]/(i+1.);
  x[1][i+1]=u[1][i]/(i+1.);
  y[0][i+1]=u[2][i]/(i+1.);
  y[1][i+1]=u[3][i]/(i+1.);
  z[0][i+1]=u[4][i]/(i+1.);
  z[1][i+1]=u[5][i]/(i+1.);

  xd[0][i+1]=(m2*u[24][i]-u[30][i])/(i+1.);
  yd[0][i+1]=(m2*u[25][i]-u[31][i])/(i+1.);
  zd[0][i+1]=(m2*u[26][i]-u[32][i])/(i+1.);

  xd[1][i+1]=(m1*u[27][i]-u[33][i])/(i+1.);
  yd[1][i+1]=(m1*u[28][i]-u[34][i])/(i+1.);
  zd[1][i+1]=(m1*u[29][i]-u[35][i])/(i+1.);


// ! consider the third body also
  if(hpow*(fabs( x[0][i+1])+fabs( x[1][i+1])+\
           fabs( y[0][i+1])+fabs( y[1][i+1])+\
           fabs( z[0][i+1])+fabs( z[1][i+1])+\
           fabs(xd[0][i+1])+fabs(xd[1][i+1])+\
           fabs(yd[0][i+1])+fabs(yd[1][i+1])+\
           fabs(zd[0][i+1])+fabs(zd[1][i+1])+\
           fabs((m1*x[0][i+1]+m2*x[1][i+1])/m3)+\
           fabs((m1*y[0][i+1]+m2*y[1][i+1])/m3)+\
           fabs((m1*z[0][i+1]+m2*z[1][i+1])/m3)+\
           fabs((m1*xd[0][i+1]+m2*xd[1][i+1])/m3)+\
           fabs((m1*yd[0][i+1]+m2*yd[1][i+1])/m3)+\
           fabs((m1*zd[0][i+1]+m2*zd[1][i+1])/m3))\
      <tol) {loopsdone=i+1; break;}
  loopsdone=i+1;
 }

 new_h=h;
 if(loopsdone<10) new_h=min(1.5*h,max_h);
 if(loopsdone>20) new_h=0.5*h;
 if(loopsdone==max_n) return -h;

// update positions & velocities of m1, m2
 hpow=1.;
 for(i=1; i<loopsdone; i++){
```

```
  hpow*=h;
  x[0][0]+=   x[0][i]*hpow;
  x[1][0]+=   x[1][i]*hpow;
  y[0][0]+=   y[0][i]*hpow;
  y[1][0]+=   y[1][i]*hpow;
  z[0][0]+=   z[0][i]*hpow;
  z[1][0]+=   z[1][i]*hpow;

  xd[0][0]+=xd[0][i]*hpow;
  xd[1][0]+=xd[1][i]*hpow;
  yd[0][0]+=yd[0][i]*hpow;
  yd[1][0]+=yd[1][i]*hpow;
  zd[0][0]+=zd[0][i]*hpow;
  zd[1][0]+=zd[1][i]*hpow;

 };
 //update position & velocity of the third body
 x[2][0]=    -(m1* x[0][0]+m2* x[1][0])/m3;
 y[2][0]=    -(m1* y[0][0]+m2* y[1][0])/m3;
 z[2][0]=    -(m1* z[0][0]+m2* z[1][0])/m3;
 xd[2][0]=   -(m1*xd[0][0]+m2*xd[1][0])/m3;
 yd[2][0]=   -(m1*yd[0][0]+m2*yd[1][0])/m3;
 zd[2][0]=   -(m1*zd[0][0]+m2*zd[1][0])/m3;

#ifndef pgplot_graphics
 printf("E_0= %3.15lf\tE(t)= %15.14lf\t%15.14lf\n",\
                  E_0,Energy(),fabs(Energy()-E_0));
 printf("time: %5.20lf\t order %d. step: %5.20lf\n",\
                  t,loopsdone,h);
#endif
 return new_h;
}; // end of do_a_step()
```

3.3.4 Αποτελέσματα.



Σχήμα 8: Το Επίπεδο Πρόβλημα των 3 Σωμάτων για $t = 11000$. Για αρχικές συνθήκες βλ. αντίστοιχο Παράρτημα.



Σχήμα 9: Το Γενικό Πρόβλημα των 3 Σωμάτων. Για αρχικές συνθήκες βλ. αντίστοιχο Παράρτημα.

4 Συμπεράσματα

Η μεθοδολογία επίλυσης γραμμικών και ομογενών ΣΔΕ δεύτερης τάξης, η οποία αναπτύχθηκε στις προηγούμενες Ενότητες, είναι μια στιβαρή μέθοδος για προβλήματα ΣΔΕ αρχικών συνθηκών. Συνδυάζει τα επιθυμητά χαρακτηριστικά της *(θεωρητικά) αυθαίρετης ακρίβειας*, κάτι που βέβαια περιορίζεται από την ακρίβεια μηχανής που το υπολογιστικό μας σύστημα ή το συνεργαζόμενο Λογισμικό παρέχει, την *ταχύτητα υπολογισμών*, καθώς αποδεικνύεται ταχύτερη από τις περισσότερες συμβατικές μεθόδους ολοκλήρωσης [Fox, 1984, Roberts, 1975] και, κάτι αρκετά σημαντικό για πολυπληθή συστήματα (πολλών βαθμών ελευθερίας), *δίνει τη δυνατότητα βελτιστοποίησης των εσωτερικών βρόχων* του υπολογιστικού της πυρήνα.

Οι παλαιότερες υλοποιήσεις της, επηρεασμένες ίσως από τη φαινομενολογία που ακολουθείται στη θεωρητική ανάπτυξη του θέματος, λάμβαναν υπόψη τους και τους συντελεστές των δυνάμεων του αναπτύγματος. Οι συγκεκριμένοι όροι περιέχουν εκφράσεις με παραγοντικά, και εισάγουν αναπόφευκτα, εάν χρησιμοποιηθούν, σφάλματα αποκοπής και στρογγυλοποίησης, τα οποία μάλιστα γίνονται πιο σημαντικά καθώς η τάξη προσέγγισης αυξάνεται. Η εισαγωγή του συμβολισμού των κανονικοποιημένων παραγώγων έδωσε ικανοποιητική λύση στο συγκεκριμένο θέμα.

Η αυτοματοποιημένη διαφόριση, εντατικά αναπτυσσόμενο ερευνητικό πεδίο, έδωσε νέα ώθηση στις μεθόδους επίλυσης ΣΔΕ με σειρές, καθιστώντας δυνατή την εύκολη υλοποίηση και κατάστρωση σύνθετων προβλημάτων. Βεβαίως, δεν έχουμε φτάσει ακόμη στο στάδιο αυτός ο αυτοματοποιημένος κώδικας να μπορεί να ανταγωνιστεί τον προσεκτικά και δια χειρός προγραμματιστή διαμορφωμένο κώδικα, ιδιαίτερα όταν η αυτοματοποίηση δεν λαμβάνει υπόψη τηςορισμένες παραδοχές, τις οποίες ο σχεδιαστής του υπολογιστικού πυρήνα, στην υλοποίηση που παραθέσαμε, έχει στη διάθεσή του (επιπλέον δεσμοί και ολοκληρώματα του συστήματος των ΔΕ, δυνατότητα απαλοιφής μεταβλητών με βάση αυτά, επαναχρησιμοποίηση βοηθητικών μεταβλητών κλπ.).

Η μέθοδος των σειρών παρέχει ακρίβεια μηχανής. Έτσι, δεν μπορεί απευθείας να συγκριθεί ως προς την απόδοσή της με μία άλλη δημοφιλή κατηγορία προγραμμάτων ολοκλήρωσης, τους συμπλεκτικούς ολοκληρωτές, παρά μόνο έμμεσα. Η μέθοδος των σειρών δεν μπορεί να γίνει «πιο συμπλεκτική».

Περαιτέρω έρευνα και επεξεργασία της μεθοδολογίας επίλυσης η οποία στηρίζεται στις σειρές Taylor μπορεί να γίνει τόσο σε διατηρητκά και Χαμιλτονιανά συστήματα, όσο και σε μη συντηρητικά, απωλεστικά (dissipative), ιδιαίτερα στις περιπτώσεις αρκετών από αυτά που είναι «δημοφιλή» και αποτελούν μέτρο σύγκρισης διαφόρων μεθόδων, τα οποία στα δεξιά τους μέλη έχουν πολυώνυμα (Lorenz, Rössler κ.ά.).

5 Παραρτήματα

5.1 Πρόγραμμα επίλυσης του προβλήματος του αρμονικού ταλαντωτή

```
// Taylor Series solution of the harmonic oscillator
// them - [ekots@uom.gr]
//
// uncomment this to compile with pgplot support
//
//
// build as :
// gcc -c <thisfile>.c
// f77 <thisfile>.o libcpgplot.a libpgplot.a -o <exec_file>
//     -I /usr/X11R6/include -L /usr/X11R6/lib -lX11 -lm
//
#define pgplot_graphics

#ifdef pgplot_graphics
 #include "cpgplot.h"
#endif

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//-----------------------------------------------------------------
//    globals & #defines
//-----------------------------------------------------------------
//
// tolerance in x
#define tol 1e-15

#define max(a,b)    (((a) > (b)) ? (a) : (b))
#define min(a,b)    (((a) < (b)) ? (a) : (b))

#undef  max_n
#define max_n ((int)(40))

#undef  max_h
#define max_h 0.2L

#undef  min_h
#define min_h 1e-25
//-----------------------------------------------------------------
// system: harmonic oscillator, x''+ w*w * x = 0
// this reduces to the first order system
//   x' = y
//   y' = - w*w * x
//
```

```
//    so, we need 4 variables
//
//    u1 = - w*w * x
//    u2 = y
//    x' = u2
//    y' = u1
//
//    ------------------------------
//
//    normalized derivative of a function u:
//
//      [n]         1    (n)
//    u    (t) = ---- u    (t)
//               n!
//
//    ------------------------------
//
//
//       [n]                [n]
//    u1    (t)  = -w*w x     (t)
//
//       [n]          [n]
//    u2    (t)  = y     (t)
//
//      [n+1]         1      [n]
//    x       (t) = ---- u2     (t)
//                  n+1
//
//      [n+1]         1     [n]
//    y       (t) = ---- u1     (t)
//                  n+1
//

#undef  omega
#define omega (1.00)
#define omega2 (omega*omega)

double u1[max_n], u2[max_n], x[max_n], y[max_n];

double t;

double E_0;

//-------------------------------------------------------------------
#ifdef pgplot_graphics
 // for number to text conversion by pgplot
 int nc=10;
 char s[100];

 float XX, YY;
```

```
 char ch=100; // !
 int junk;

 void init_graphics(float ulx, float uly,
                    float urx, float ury,
                    float llx, float lly,
                    float lrx, float lry);
 void draw(float ulx, float uly,
           float urx, float ury,
           float llx, float lly,
           float lrx, float lry,
           float h);
#endif
//-----------------------------------------------------------------
double Energy(){return 0.5*y[0]*y[0]+0.5*omega2*x[0]*x[0];};
//-----------------------------------------------------------------
double do_a_step(double hstep){
 double h=hstep;
 double new_h;  // new timestep
 int i;
 int order;     // order of Taylor series expansion
 double hpow;

 if(h<min_h) {
         printf("Step error at time %lf. Exiting...\n",t);
         exit(1);
 };

 order=max_n;
 hpow=1.;
 // fill up u1[i], u2[i], x[i+1], y[i+1] matrices
 // up to order max_n
 for (i=0; i<max_n-1; i++){
  hpow*=h;

  u1[i]=(-omega2)*x[i];
  u2[i]=y[i];
  x[i+1]=(1./(i+1.))*u2[i];
  y[i+1]=(1./(i+1.))*u1[i];

  // series has converged ok?
  if(hpow*(fabs(x[i+1])+fabs(y[i+1]))<tol) {order=i+1; break;}
  order=i+1;
 }

 new_h=h;
 if(order<10) new_h=min(2.0*h,max_h);
 if(order>20) new_h=0.5*h;
 if(order==max_n) return -h;
```

```
// make Taylor expansion & update x[], y[] matrices
 hpow=1.;
 for(i=1; i<order; i++){
  hpow*=h;
  x[0]+=x[i]*hpow;
  y[0]+=y[i]*hpow;
 };

 printf("\ntime: %15.14lf\t order %d. step: %15.14lf\n",\
                 t,order,h);
 printf("E_0= %15.14lf\tE(t)= %15.14lf\t%15.14lf\n",\
                 E_0,Energy(),fabs(Energy()-E_0));

 return new_h;
}; // end of do_a_step()
//-----------------------------------------------------------------

int main(){
 double time, h, ah;
 double x0,y0;// for pgplot panels

 x0=x[0]= 0.;
 y0=y[0]= 1.;
 time = 200;
 h=0.01;
 t=0.;

 E_0= Energy();

#ifdef pgplot_graphics
 init_graphics((float)(2.4*y0),(float)(2.4*y0),
               (float)(1.1*time),(float)(2.4*y0),
               (float)(2.4*y0),(float)(2.4*y0),
               (float)(1.1*time),(float)(2.4*y0));
#endif

 while(t<time){
  if ((ah=do_a_step(h))>0) {t+=h; h=ah;}
  else {h=-ah;};
#ifdef pgplot_graphics
  draw((float)(2.4*y0),(float)(2.4*y0),
       (float)(1.1*time),(float)(2.4*y0),
       (float)(2.4*y0),(float)(2.4*y0),
       (float)(1.1*time),(float)(2.4*y0),
       (float)h);
#endif
 };

#ifdef pgplot_graphics
  cpgend();
```

```
#endif
 return 0;
};

//=============================================================
//=============================================================
#ifdef pgplot_graphics
 void init_graphics(float ulx, float uly,
                    float urx, float ury,
                    float llx, float lly,
                    float lrx, float lry){
  if(cpgbeg(0, "/xw", 1, 1) != 1)
    exit(EXIT_FAILURE);
  cpgsch(1.7);//determines fontsize
  cpgsubp(2,2);
  cpgenv(-ulx/2.,ulx/2.,-uly/2.,uly/2., 0, 1);
  cpglab("(x)", "(x_dot)",
   "harmonic oscillator, x-x_dot phase space");

  cpgenv(0,urx,-ury/2.,ury/2., 0, 1);
  cpglab("(t)", "(x)",
   "harmonic oscillator, x timeseries");

  cpgenv(-llx/2.,llx/2.,-lly/2.,lly/2., 0, 0);
  cpglab("", "",
   "harmonic oscillator, details panel");

  cpgenv(0,lrx,-lry/2.,lry/2., 0, 1);
  cpglab("(t)", "(x_dot)",
   "harmonic oscillator, x_dot timeseries");

  cpgsci(1);
 };// end of init_graphics(...)

//-------------------------------------------------------------
 void draw(float ulx, float uly,
           float urx, float ury,
           float llx, float lly,
           float lrx, float lry,
           float h){
 cpgpanl(1,1);
 cpgswin(-ulx/2.,ulx/2.,-uly/2.,uly/2.);
 cpgsci(2); cpgpt1((float)x[0],(float)y[0],1);  //x - x_dot

 cpgpanl(2,1);
 cpgswin(0,urx,-ury/2.,ury/2.);
 cpgsci(3); cpgpt1((float)t,(float)x[0],1);     // time - x

 cpgpanl(2,2);
 cpgswin(0,lrx,-lry/2.,lry/2.);
```

```
 cpgsci(5); cpgpt1((float)t,(float)y[0],1);       // time - y

 cpgpanl(1,2);
 cpgswin(-llx/2.,llx/2.,-lly/2.,lly/2.);
 cpgsci(0);
 cpgrect(-0.9*llx/2.,0.9*llx/2.,-0.9*lly/2.,0.9*lly/2.);
 cpgsci(1);
 sprintf(s,"Time_______:  %3.7lf",t);
 cpgmtxt("RV",-22.0,0.9,0.0,s);nc=20;
 sprintf(s,"Step_______:  %3.7lf",h);
 cpgmtxt("RV",-22.0,0.8,0.0,s);nc=20;
 sprintf(s,"E(0)_______: %3.20lf",E_0);
 cpgmtxt("RV",-22.0,0.7,0.0,s);nc=20;
 sprintf(s,"E(t)_______:  %3.20lf",Energy());
 cpgmtxt("RV",-22.0,0.6,0.0,s);nc=20;
 sprintf(s,"|E(t)-E_0|    : %3.20lf",fabs(Energy()-E_0));
 cpgmtxt("RV",-22.0,0.5,0.0,s);nc=20;
 };//end of draw(...)
#endif
```

5.2 Πρόγραμμα επίλυσης του προβλήματος των δύο σωμάτων

```
// Taylor Series solution of the 2 body problem
// them - [ekots@uom.gr]
//

// uncomment this to compile with pgplot support
//
//
// build as :
// gcc -c <thisfile>.c
// f77 <thisfile>.o libcpgplot.a libpgplot.a -o <exec_file>
//     -I /usr/X11R6/include -L /usr/X11R6/lib -lX11 -lm
//
#define pgplot_graphics

#ifdef pgplot_graphics
 #include "cpgplot.h"
#endif

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//-----------------------------------------------------------------------
//    globals & #defines
//-----------------------------------------------------------------------
//
// tolerance in x
#undef  tol
#define tol 1e-25

#define max(a,b)    (((a) > (b)) ? (a) : (b))
#define min(a,b)    (((a) < (b)) ? (a) : (b))

#undef  max_n
#define max_n ((int)(40))

#undef  max_h
#define max_h 0.2L

#undef  min_h
#define min_h 1e-25

double u[10][max_n], x[max_n], y[max_n],z[max_n], w[max_n];

double t, E_0;

//-----------------------------------------------------------------------
#ifdef pgplot_graphics
```

```
 // for number to text conversion by pgplot
 int nc=10;
 char s[100];

 float XX, YY;
 char ch=100; // !
 int junk;

 void init_graphics(float ulx, float uly,
                    float urx, float ury,
                    float llx, float lly,
                    float lrx, float lry);
 void draw(float ulx, float uly,
           float urx, float ury,
           float llx, float lly,
           float lrx, float lry,
           float h);
#endif
//-------------------------------------------------------------------
double Energy()
{
 return 0.5*(z[0]*z[0]+ w[0]*w[0]) - 1.0/sqrt(x[0]*x[0]+y[0]*y[0]);
}
//-------------------------------------------------------------------
double do_a_step(double hstep){
 double h=hstep;
 double new_h;  // new timestep
 int i,j,loopsdone;
 double sum=0.;
 double hpow;

 if(h<min_h) {
         printf("Step error at time %lf. Exiting...\n",t);
         exit(1);
 };

 loopsdone=max_n;
 hpow=1.;
 // fill up u[][i], x[i+1], y[i+1], z[i+1], w[i+1] matrices
 // up to order max_n
 for (i=0; i<max_n-1; i++){
  hpow*=h;

  u[0][i]=z[i];                                          // x'
  u[1][i]=w[i];                                          // y'
  u[2][i]=x[i];
  u[3][i]=y[i];
  sum=0.; for (j=0; j<=i; j++) sum+=u[2][i-j]*u[2][j];   //x*x
  u[4][i]=sum;
  sum=0.; for (j=0; j<=i; j++) sum+=u[3][i-j]*u[3][j];   //y*y
```

```
 u[5][i]=sum;

                                                          // r=sqrt(x^2+y^2
 if (i==0){u[6][0]=pow((u[4][0]+u[5][0]),3./2.);}         // r^3
 else {
  sum=0.;
  for (j=0; j<=i-1; j++)
   sum+= (1.5*i-j*(2.5))*(u[4][i-j]+u[5][i-j])*u[6][j];
  sum/=i*(u[4][0]+u[5][0]);
  u[6][i]=sum;
 };

 sum=0.; for (j=1; j<=i; j++) sum-=u[6][j]*u[7][i-j];   // quotient
 sum+=u[2][i];
 sum/=u[6][0];
 u[7][i]= sum;                                            // x/r^3

 sum=0.; for (j=1; j<=i; j++) sum-=u[6][j]*u[8][i-j];   // quotient
 sum+=u[3][i];
 sum/=u[6][0];
 u[8][i]= sum;                                            // y/r^3

 x[i+1]= u[0][i]/(i+1);
 y[i+1]= u[1][i]/(i+1);
 z[i+1]=-u[7][i]/(i+1);
 w[i+1]=-u[8][i]/(i+1);

 // series has converged ok?
 if(hpow*(fabs(x[i+1])+fabs(y[i+1])+fabs(z[i+1])+fabs(w[i+1]))<tol)
   {loopsdone=i+1; break;}
 loopsdone=i+1;
}
new_h=h;
if(loopsdone<10) new_h=min(2.0*h,max_h);
if(loopsdone>20) new_h=0.5*h;
if(loopsdone==max_n) return -h;

// make Taylor expansion & update x[], y[], z[], w[] matrices
hpow=1.;
for(i=1; i<loopsdone; i++){
   hpow*=h;
        x[0]+=x[i]*hpow;
        y[0]+=y[i]*hpow;
        z[0]+=z[i]*hpow;
        w[0]+=w[i]*hpow;
};
printf("\ntime: %15.14lf\t order %d. step: %15.14lf\n",\
                  t,loopsdone,h);
printf("E_0= %15.14lf\tE(t)= %15.14lf\t%15.14lf\n",\
                  E_0,Energy(),fabs(Energy()-E_0));
```

```
 return new_h;
}; // end of do_a_step()

//-----------------------------------------------------------------
int main(){
 double time,h,ah;
 double x0,y0;// for pgplot panels

 // semimajor axis 1, eccentricity 0.3
 x[0]=    0.7;
 y[0]=    0.1;
 z[0]=    0.1;
 w[0]=    0.953939201;

/*
 // near collision ICs
 x[0]=    0.7;
 y[0]=    0.7;
 z[0]=    0.7;
 w[0]=    0.953939201;
*/


 time = 100;
 h=1e-1;
 t=0.;

 E_0= Energy();

#ifdef pgplot_graphics
 x0=2.*(x[0]+w[0]); y0=1.1*(x[0]+y[0]);
 init_graphics((float)(x0),(float)(1.2*x0),
               (float)(x0),(float)(2.4*x0),
               (float)(2.4*x0),(float)(2.4*x0),
               (float)(1.1*time),(float)(2.4*y0));
#endif

 while(t<time){
  if ((ah=do_a_step(h))>0) { t+=h; h=ah;}
  else { h=-ah;};
#ifdef pgplot_graphics
  draw((float)(x0),(float)(1.2*x0),
       (float)(x0),(float)(2.4*x0),
       (float)(2.4*x0),(float)(2.4*x0),
       (float)(1.1*time),(float)(2.4*y0),h);
#endif
 };

#ifdef pgplot_graphics
 cpgend();
```

```
#endif
 return 0;
};

//=============================================================
//=============================================================
#ifdef pgplot_graphics
 void init_graphics(float ulx, float uly,
                    float urx, float ury,
                    float llx, float lly,
                    float lrx, float lry){
  if(cpgbeg(0, "/xw", 1, 1) != 1)
    exit(EXIT_FAILURE);
//  cpgpap(13,0.8618);
  cpgsch(1.8);//determines fontsize
  cpgsubp(2,2);
//  cpgsubp(3,2);
  cpgenv(-ulx/2.,ulx/2.,-uly/2.,uly/2., 0, 1);
  cpglab("(x)", "(y)",
   "2 body problem, 2D configuration space");

  cpgenv(-urx/2.,urx/2.,-ury/2.,ury/2., 0, 1);
  cpglab("(x,y)", "(x_dot,y_dot)",
   "2 body problem, x-x_dot / y-y_dot phase space");

  cpgsci(0);
  cpgenv(-llx/2.,llx/2.,-lly/2.,lly/2., -1, -1);
  cpgsci(1);
  cpglab("", "",
   "2 body problem, details panel");

  cpgenv(0,lrx,-lry/2.,lry/2., 0, 1);
  cpglab("(t)", "(x,y)",
   "2 body problem, x timeseries, y timeseries");

//  cpgenv(-lrx/2.,lrx/2.,-lry/2.,lry/2., 0, 1);
//  cpglab("(y)", "(y_dot)",
//   "2 body problem, y-y_dot phase space");

  cpgsci(1);
 };// end of init_graphics(...)

//-------------------------------------------------------------
 void draw(float ulx, float uly,
           float urx, float ury,
           float llx, float lly,
           float lrx, float lry,
           float h){
 cpgpanl(1,1);
 cpgswin(-ulx/2.,ulx/2.,-uly/2.,uly/2.);
```

```
 cpgsci(2); cpgpt1((float)x[0],(float)y[0],1);   // x - y

 cpgpanl(2,1);
 cpgswin(-urx/2.,urx/2.,-ury/2.,ury/2.);
 cpgsci(3); cpgpt1((float)x[0],(float)z[0],1);   // x - x_dot
 cpgsci(4); cpgpt1((float)y[0],(float)w[0],1);   // y - y_dot

 cpgpanl(2,2);
 cpgswin(0,lrx,-lry/2.,lry/2.);
 //cpgswin(-lrx/2.,lrx/2.,-lry/2.,lry/2.);
 cpgsci(5); cpgpt1((float)t,(float)x[0],1);      // t - x
 cpgsci(6); cpgpt1((float)t,(float)y[0],1);      // t - y

 cpgpanl(1,2);
 cpgswin(-llx/2.,llx/2.,-lly/2.,lly/2.);
 cpgsci(0);
 cpgrect(-0.9*llx/2.,0.9*llx/2.,-0.9*lly/2.,0.9*lly/2.);
 cpgsci(1);
 sprintf(s,"Time_____:  %3.7lf",t);
 cpgmtxt("RV",-22.0,0.9,0.0,s);nc=20;
 sprintf(s,"Step_____:  %3.7lf",h);
 cpgmtxt("RV",-22.0,0.8,0.0,s);nc=20;
 sprintf(s,"E(0) _____: %3.20lf",E_0);
 cpgmtxt("RV",-22.0,0.7,0.0,s);nc=20;
 sprintf(s,"E(t) _____:  %3.20lf",Energy());
 cpgmtxt("RV",-22.0,0.6,0.0,s);nc=20;
 sprintf(s,"|E(t)-E_0| : %3.20lf",fabs(Energy()-E_0));
 cpgmtxt("RV",-22.0,0.5,0.0,s);nc=20;
 };//end of draw(...)
#endif
```

5.3 Πρόγραμμα επίλυσης του επίπεδου προβλήματος των τριών σωμάτων

```
// Taylor Series solution of the *planar* 3 body problem
// them - [ekots@uom.gr]
//
//
// uncomment this to compile with pgplot support
//
//
// build as :
// gcc -c <thisfile>.c
// f77 <thisfile>.o libcpgplot.a libpgplot.a -o <exec_file>
//     -I /usr/X11R6/include -L /usr/X11R6/lib -lX11 -lm
//
//#define pgplot_graphics

#ifdef pgplot_graphics
 #include "cpgplot.h"
#endif

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//-----------------------------------------------------------------
//    globals & #defines
//-----------------------------------------------------------------
//
// tolerance in x
#undef  tol
#define tol 1e-15

#define max(a,b)     (((a) > (b)) ? (a) : (b))
#define min(a,b)     (((a) < (b)) ? (a) : (b))

#undef max_n
#define max_n ((int)(30))

#undef max_h
#define max_h 0.2

#undef  min_h
#define min_h 1e-26


double u[25][max_n], x[3][max_n],  y[3][max_n],\
                  xd[3][max_n], yd[3][max_n];

double t;
```

```
double m1,m2,m3;
double M31, M32, mu23, mu13, N13, N23;
double E_0;

//-----------------------------------------------------------------
#ifdef pgplot_graphics
 // for number to text conversion by pgplot
 int nc=10;
 char s[100];

 float XX, YY;
 char ch=100; // !
 int junk;
 int x1y1=2, x2y2=3, x3y3=7;

 void init_graphics(float ulx, float uly,
                    float umx, float umy,
                    float urx, float ury,
                    float llx, float lly,
                    float lmx, float lmy,
                    float lrx, float lry);
 int draw(float ulx, float uly,
          float umx, float umy,
          float urx, float ury,
          float llx, float lly,
          float lmx, float lmy,
          float lrx, float lry,
          float h, int duration);
#endif
//-----------------------------------------------------------------
double Energy()
{
        return
    0.5*(\
         m1*(xd[0][0]*xd[0][0] + yd[0][0]*yd[0][0] )+\
         m2*(xd[1][0]*xd[1][0] + yd[1][0]*yd[1][0] )+\
         m3*(xd[2][0]*xd[2][0] + yd[2][0]*yd[2][0] )\
        ) \
    -m1*m2*(1./(sqrt(\
        (x[0][0]-x[1][0])*(x[0][0]-x[1][0])+\
        (y[0][0]-y[1][0])*(y[0][0]-y[1][0])\
                    )))\
    -m1*m3*(1./(sqrt(\
        (x[0][0]-x[2][0])*(x[0][0]-x[2][0])+\
        (y[0][0]-y[2][0])*(y[0][0]-y[2][0])\
                    )))\
    -m2*m3*(1./(sqrt(\
        (x[1][0]-x[2][0])*(x[1][0]-x[2][0])+\
        (y[1][0]-y[2][0])*(y[1][0]-y[2][0])\
                    )));
```

```
}
//-------------------------------------------------------------------
double do_a_step(double hstep){
 double h=hstep;
 double new_h;  // new timestep
 int i,j,loopsdone=max_n;
 double sum=0.;
 double hpow=1.;

 if(h<min_h) {
         printf("Step error at time %lf. Exiting...\n",t);
         exit(1);
 };

// fill up u[][i],  x[1,2,3][i+1],  y[1,2,3][i+1],  z[1,2,3][i+1],
//                 xd[1,2,3][i+1], yd[1,2,3][i+1], zd[1,2,3][i+1]
//                 matrices up to order max_n
 for (i=0; i<max_n-1; i++){
  hpow*=h;

  u[ 0][i]=xd[0][i];              //xd1
  u[ 1][i]=xd[1][i];              //xd2
  u[ 2][i]=yd[0][i];              //yd1
  u[ 3][i]=yd[1][i];              //yd2
  u[ 4][i]= x[0][i];              //x1
  u[ 5][i]= x[1][i];              //x2
  u[ 6][i]= y[0][i];              //y1
  u[ 7][i]= y[1][i];              //y2

  sum=0.; for (j=0; j<=i; j++)
          sum+=(u[5][i-j]-u[4][i-j])*(u[5][j]-u[4][j]); //(x2-x1)^2
  u[ 8][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
          sum+=(u[7][i-j]-u[6][i-j])*(u[7][j]-u[6][j]); //(y2-y1)^2
  u[ 9][i]= sum;

  if (i==0){
          u[10][0]=pow(u[ 8][i] + u[ 9][i], 1.5);        //R_12^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[ 8][i-j] + u[ 9][i-j])*u[10][j];
   sum/=(i*(u[ 8][0] + u[ 9][0]));
   u[10][i]=sum;
  };

  sum=0.; for (j=1; j<=i; j++) sum-=u[10][j]*u[11][i-j];// quotient
  sum+=u[5][i]-u[4][i]; sum/=u[10][0];                  // 1a
  u[11][i]= sum;
```

```
  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(N13*u[4][i-j]+mu23*u[5][i-j])*(N13*u[4][j]+mu23*u[5][j]);
  u[12][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(N13*u[6][i-j]+mu23*u[7][i-j])*(N13*u[6][j]+mu23*u[7][j]);
  u[13][i]= sum;

  if (i==0){
          u[14][0]=pow(u[12][0] + u[13][0], 1.5);          //R_13^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[12][i-j] + u[13][i-j])*u[14][j];
   sum/=(i*(u[12][0] + u[13][0]));
   u[14][i]=sum;
  };

  sum=0.; for (j=1; j<=i; j++) sum-=u[14][j]*u[15][i-j];// quotient
  sum+=M31*u[4][i]+m2*u[5][i]; sum/=u[14][0];              // 1b
  u[15][i]= sum;

//-------------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[10][j]*u[16][i-j];// quotient
  sum+=u[7][i]-u[6][i]; sum/=u[10][0];                     // 2a
  u[16][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[14][j]*u[17][i-j];// quotient
  sum+=M31*u[6][i]+m2*u[7][i];
  sum/=u[14][0];                                           // 2b
  u[17][i]= sum;

//-------------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[10][j]*u[18][i-j];// quotient
  sum+=u[4][i]-u[5][i]; sum/=u[10][0];                     // 3a
  u[18][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(N23*u[5][i-j]+mu13*u[4][i-j])*(N23*u[5][j]+mu13*u[4][j]);
  u[19][i]= sum;

  sum=0.;
  for (j=0; j<=i; j++)
   sum+=(N23*u[7][i-j]+mu13*u[6][i-j])*(N23*u[7][j]+mu13*u[6][j]);
  u[20][i]= sum;
```

```
  if (i==0){
   u[21][0]=pow(u[19][0] + u[20][0], 1.5);                    //R_23^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[19][i-j] + u[20][i-j])*u[21][j];
   sum/=(i*(u[19][0] + u[20][0]));
   u[21][i]=sum;
  };

  sum=0.; for (j=1; j<=i; j++) sum-=u[21][j]*u[22][i-j];// quotient
  sum+=M32*u[5][i]+m1*u[4][i];
  sum/=u[21][0];                                              // 3b
  u[22][i]= sum;

//-------------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[10][j]*u[23][i-j];// quotient
  sum+=u[6][i]-u[7][i];  sum/=u[10][0];
  u[23][i]= sum;                                              // 4a

  sum=0.; for (j=1; j<=i; j++) sum-=u[21][j]*u[24][i-j];// quotient
  sum+=M32*u[7][i]+m1*u[6][i];
  sum/=u[21][0];                                              // 4b
  u[24][i]= sum;

//-------------------------------------------------------------------------

  x[0][i+1]=u[0][i]/(i+1.);
  x[1][i+1]=u[1][i]/(i+1.);
  y[0][i+1]=u[2][i]/(i+1.);
  y[1][i+1]=u[3][i]/(i+1.);

  xd[0][i+1]=(m2*u[11][i]-u[15][i])/(i+1.);
  xd[1][i+1]=(m1*u[18][i]-u[22][i])/(i+1.);
  yd[0][i+1]=(m2*u[16][i]-u[17][i])/(i+1.);
  yd[1][i+1]=(m1*u[23][i]-u[24][i])/(i+1.);

  if(hpow*(fabs( x[0][i+1])+fabs( x[1][i+1])+\
           fabs( y[0][i+1])+fabs( y[1][i+1])+\
           fabs(xd[0][i+1])+fabs(xd[1][i+1])+\
           fabs(yd[0][i+1])+fabs(yd[1][i+1]))\
     <tol) {loopsdone=i+1; break;}
  loopsdone=i+1;
 }

 new_h=h;
 if(loopsdone<10) new_h=min(1.5*h,max_h);
 if(loopsdone>20) new_h=0.5*h;
 if(loopsdone==max_n) return -h;
```

```
// update positions & velocities of m1, m2
 hpow=1.;
 for(i=1; i<loopsdone; i++){
  hpow*=h;
  x[0][0]+=  x[0][i]*hpow;
  x[1][0]+=  x[1][i]*hpow;
  y[0][0]+=  y[0][i]*hpow;
  y[1][0]+=  y[1][i]*hpow;
  xd[0][0]+=xd[0][i]*hpow;
  xd[1][0]+=xd[1][i]*hpow;
  yd[0][0]+=yd[0][i]*hpow;
  yd[1][0]+=yd[1][i]*hpow;
 };
 //update position & velocity of the third body
 x[2][0]=   -(m1* x[0][0]+m2* x[1][0])/m3;
 y[2][0]=   -(m1* y[0][0]+m2* y[1][0])/m3;
 xd[2][0]=  -(m1*xd[0][0]+m2*xd[1][0])/m3;
 yd[2][0]=  -(m1*yd[0][0]+m2*yd[1][0])/m3;

#ifndef pgplot_graphics
  printf("E_0= %3.15lf\tE(t)= %15.14lf\t%15.14lf\n",\
                  E_0,Energy(),fabs(Energy()-E_0));
  printf("time: %5.20lf\t order %d. step: %5.20lf\n",\
                  t,loopsdone,h);
#endif
  return new_h;
}; // end of do_a_step()


int main(){
 double time,h,ah;
 //double x0,y0;// for pgplot panels

 FILE *fil;

 m1=0.00078;
 m2=0.00019;
 m3=1.-m1-m2;

 M31=m3+m1;
 M32=m3+m2;
 mu23=m2/m3;
 mu13=m1/m3;

 N13=1.+mu13;
 N23=1.+mu23;

 x[0][0]= 1.009996829;
 y[0][0]= 0.0;
```

```
 x[1][0]=  -2.069909714;
 y[1][0]=  0.0;

 x[2][0]= -(m1*x[0][0]+m2*x[1][0])/m3;
 y[2][0]= -(m1*y[0][0]+m2*y[1][0])/m3;

 xd[0][0]=   0.0;
 yd[0][0]=   1.004685526;

 xd[1][0]= -0.651;// this one goes off in t~25000
 xd[1][0]=   0.6;
 yd[1][0]= -0.707518398;

 xd[2][0]=  -(m1*xd[0][0]+m2*xd[1][0])/m3;
 yd[2][0]=  -(m1*yd[0][0]+m2*yd[1][0])/m3;

//-------------------------------
 time = 100;
 h=1e-2;
 t=0.;

 E_0= Energy();
 printf("E=%f\n",E_0); getchar();

#ifdef pgplot_graphics
 x0=50.; y0=100.; junk=0;
 srand(4105);
 if(cpgbeg(0, "/xw", 1, 1) != 1)
    exit(EXIT_FAILURE);
 init_graphics((float)(x0),(float)(y0),
               (float)(1.1*time),(float)(x0),
               (float)(x0),(float)(4),
               (float)(10),(float)(10),
               (float)(1.1*time),(float)(y0),
               (float)(x0),(float)(3)
              );
#else
 fil=fopen("ttt.dat","wt");
#endif

 while(t<time){
  if ((ah=do_a_step(h))>0) { t+=h; h=ah; }
  else { h=-ah; };
#ifdef pgplot_graphics
  junk=draw((float)(x0),(float)(y0),
               (float)(1.1*time),(float)(x0),
               (float)(x0),(float)(4),
               (float)(10),(float)(10),
               (float)(1.1*time),(float)(y0),
               (float)(y0),(float)(3),h,junk);
```

```
#else
   fprintf(fil,"%f  %f  %f  %f  %f\n",t,x[0][0], y[0][0], x[1][0], y[1][0]
#endif
  };
#ifdef pgplot_graphics
  getchar();
  cpgend();
#else
  fclose(fil);
#endif
  return 0;
};
//=============================================================
//=============================================================
#ifdef pgplot_graphics
  void init_graphics(float ulx, float uly,
                     float umx, float umy,
                     float urx, float ury,
                     float llx, float lly,
                     float lmx, float lmy,
                     float lrx, float lry){
   cpgsci(1);
   cpgpap(12,0.8618);
   cpgsch(1.8);//determines fontsize
   cpgsubp(3,2);
   cpgenv(-ulx/2.,ulx/2.,-uly/2.,uly/2., 0, 1);
   cpglab("(x\\di\\u)", "(y\\di\\u)",
    "3 body planar, 2D configuration space");

   cpgenv(0,umx,-umy/2.,umy/2., 0, 1);
   cpglab("(t)", "(x\\di\\u)",
    "x\\di\\u timeseries");

   cpgenv(-urx/2.,urx/2.,-ury/2.,ury/2., 0, 1);
   cpglab("(x\\di\\u)", "(x\\di\\u_dot)",
    "x\\di\\u-x\\di\\u_dot phase space");

   cpgsci(0);
   cpgenv(-llx/2.,llx/2.,-lly/2.,lly/2., 0, 1);
   cpgsci(1);
   cpglab("", "",
    "details panel");

   cpgenv(0,lmx,-lmy/2.,lmy/2., 0, 1);
   cpglab("(t)", "(y\\di\\u)",
    "y\\di\\u timeseries");

   cpgenv(-lrx/2.,lrx/2.,-lry/2.,lry/2., 0, 1);
   cpglab("(y\\di\\u)", "(y\\di\\u_dot)",
    "y\\di\\u-y\\di\\u_dot phase space");
```

```
  cpgsci(1); cpgask(0);
 };// end of init_graphics(...)

//-----------------------------------------------------------------------
 int draw(float ulx, float uly,
          float umx, float umy,
          float urx, float ury,
          float llx, float lly,
          float lmx, float lmy,
          float lrx, float lry,
          float h, int duration){
 cpgpanl(1,1);
 cpgswin(-ulx/2.,ulx/2.,-uly/2.,uly/2.);
 if (duration==10000) {
  duration=-1;
  x1y1=(int)(10.*rand()/RAND_MAX); if (x1y1==0||x1y1==1) x1y1+=2;
  x2y2=(int)(10.*rand()/RAND_MAX); if (x2y2==0||x2y2==1) x2y2+=2;
  x3y3=(int)(10.*rand()/RAND_MAX);
  // erase trajectory
  cpgpanl(1,1);
  cpgsci(0);
  cpgrect(-0.99*ulx/2.,0.99*ulx/2.,-0.99*uly/2.,0.99*uly/2.);
  cpgsci(1);
  cpgbox("ABCNST",0,0,"ABCNST",0,0);
 };
 cpgsci(x1y1); cpgpt1((float)x[0][0],(float)y[0][0],1);// x1 - y1
 cpgsci(x2y2); cpgpt1((float)x[1][0],(float)y[1][0],1);// x2 - y2
 cpgsci(x3y3); cpgpt1((float)x[2][0],(float)y[2][0],9);// x3 - y3

 cpgpanl(2,1);
 cpgswin(0,umx,-umy/2.,umy/2.);
 cpgsci(2); cpgpt1((float)t,(float)x[0][0],1);          // t - x1
 cpgsci(3); cpgpt1((float)t,(float)x[1][0],1);          // t - x2
 cpgsci(7); cpgpt1((float)t,(float)x[2][0],1);          // t - x3


 cpgpanl(3,1);
 cpgswin(-urx/2.,urx/2.,-ury/2.,ury/2.);
 cpgsci(2); cpgpt1((float)x[0][0],(float)xd[0][0],1);   // x1 - x1_dot
 cpgsci(3); cpgpt1((float)x[1][0],(float)xd[1][0],1);   // x2 - x2_dot
 cpgsci(7); cpgpt1((float)x[2][0],(float)xd[2][0],9);   // x3 - x3_dot

 cpgpanl(1,2);
 cpgswin(-llx/2.,llx/2.,-lly/2.,lly/2.);
 cpgsci(0);
 //cpgrect(-0.9*llx/2.,0.9*llx/2.,-0.9*lly/2.,0.9*lly/2.);
 cpgrect(-0.99*llx/2.,0.99*llx/2.,-0.99*lly/2.,0.99*lly/2.);
 cpgsci(1);
 sprintf(s,"Time_: %3.7lf",t);
```

```
 cpgmtxt("RV",-18.0,0.9,0.0,s);nc=20;
 sprintf(s,"Step_: %3.7lf",h);
 cpgmtxt("RV",-18.0,0.8,0.0,s);nc=20;
 sprintf(s,"E(0)_:%3.20lf",E_0);
 cpgmtxt("RV",-18.0,0.7,0.0,s);nc=20;
 sprintf(s,"E(t)_: %3.20lf",Energy());
 cpgmtxt("RV",-18.0,0.6,0.0,s);nc=20;
 sprintf(s,"|E(t)-E_0| :");
 cpgmtxt("RV",-18.0,0.5,0.0,s);nc=20;
 sprintf(s,"%3.20lf",fabs(Energy()-E_0));
 cpgmtxt("RV",-14.0,0.4,0.0,s);nc=20;

 cpgpanl(2,2);
 cpgswin(0,umx,-lmy/2.,lmy/2.);
 cpgsci(2); cpgpt1((float)t,(float)y[0][0],1);          // t - y1
 cpgsci(3); cpgpt1((float)t,(float)y[1][0],1);          // t - y2
 cpgsci(7); cpgpt1((float)t,(float)y[2][0],1);          // t - y3

 cpgpanl(3,2);
 cpgswin(-lrx/2.,lrx/2.,-lry/2.,lry/2.);
 cpgsci(2); cpgpt1((float)y[0][0],(float)yd[0][0],1);   // y1 - y1_dot
 cpgsci(3); cpgpt1((float)y[1][0],(float)yd[1][0],1);   // y2 - y2_dot
 cpgsci(7); cpgpt1((float)y[2][0],(float)yd[2][0],9);   // y3 - y3_dot

 return ++duration;
 };//end of draw(...)
#endif
```

5.4 Πρόγραμμα επίλυσης του γενικού προβλήματος των τριών σωμάτων

```
// Taylor Series solution of the *full* 3 body problem
// them - [ekots@uom.gr]
//
//
// uncomment this to compile with pgplot support
//
//
// build as :
// gcc -c <thisfile>.c
// f77 <thisfile>.o libcpgplot.a libpgplot.a -o <exec_file>
//     -I /usr/X11R6/include -L /usr/X11R6/lib -lX11 -lm
//
//#define pgplot_graphics

#ifdef pgplot_graphics
 #include "cpgplot.h"
#endif

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//-----------------------------------------------------------------
//    globals & #defines
//-----------------------------------------------------------------
//
// tolerance in x
#undef  tol
#define tol 1e-25

#define max(a,b)    (((a) > (b)) ? (a) : (b))
#define min(a,b)    (((a) < (b)) ? (a) : (b))

#undef max_n
#define max_n ((int)(60))

#undef  max_h
#define max_h 0.2

#undef  min_h
#define min_h 1e-26

double u[100][max_n], x[3][max_n],  y[3][max_n],  z[3][max_n],\
                    xd[3][max_n], yd[3][max_n], zd[3][max_n];

double t;
double m1,m2,m3;
```

```
double M13, M23, mu23, mu13, N13, N23;
#define M31 (M13)
#define M32 (M23)
//
// notation & scaling:
// ---------------------
//
//  m1 + m2 + m3 = 1.
//
//    M_ij  = m_i + m+j
//   mu_ij  = m_i/m_j     *NEQ*  mu_ji = m_j/m_i
//    N_ij  = 1. + mu_ij  *NEQ*   N_ji = 1. + mu_ji
//
//

double E_0;

//-------------------------------------------------------------
#ifdef pgplot_graphics
 // for number to text conversion by pgplot
 int nc=10;
 char s[100];

 float XX, YY;
 char ch=100; // !
 int junk;
 int x1y1=2, x2y2=3, x3y3=7;

 void init_graphics(float ulx, float uly,
                    float umx, float umy,
                    float urx, float ury,
                    float llx, float lly,
                    float lmx, float lmy,
                    float lrx, float lry);
 int draw(float ulx, float uly,
          float umx, float umy,
          float urx, float ury,
          float llx, float lly,
          float lmx, float lmy,
          float lrx, float lry,
          float h, int duration);
#endif
//-------------------------------------------------------------
double Energy()
{
        return
    0.5*(\
         m1*(xd[0][0]*xd[0][0] + yd[0][0]*yd[0][0] + zd[0][0]*zd[0][0])+\
         m2*(xd[1][0]*xd[1][0] + yd[1][0]*yd[1][0] + zd[1][0]*zd[1][0])+\
         m3*(xd[2][0]*xd[2][0] + yd[2][0]*yd[2][0] + zd[2][0]*zd[2][0])\
```

```
        ) \
    -m1*m2*(1./(sqrt(\
        (x[0][0]-x[1][0])*(x[0][0]-x[1][0])+\
        (y[0][0]-y[1][0])*(y[0][0]-y[1][0])+\
        (z[0][0]-z[1][0])*(z[0][0]-z[1][0])\
                    )))\
    -m1*m3*(1./(sqrt(\
        (x[0][0]-x[2][0])*(x[0][0]-x[2][0])+\
        (y[0][0]-y[2][0])*(y[0][0]-y[2][0])+\
        (z[0][0]-z[2][0])*(z[0][0]-z[2][0])\
                    )))\
    -m2*m3*(1./(sqrt(\
        (x[1][0]-x[2][0])*(x[1][0]-x[2][0])+\
        (y[1][0]-y[2][0])*(y[1][0]-y[2][0])+\
        (z[1][0]-z[2][0])*(z[1][0]-z[2][0])\
                    )));
}
//-------------------------------------------------------------------
double do_a_step(double hstep){
 double h=hstep;
 double new_h;  // new timestep
 int i,j,loopsdone=max_n;
 double sum=0.;
 double hpow=1.;

 if(h<min_h) {
         printf("Step error at time %lf. Exiting...\n",t);
         exit(1);
 };

// fill up u[][i],  x[1,2,3][i+1],  y[1,2,3][i+1],  z[1,2,3][i+1],
//                 xd[1,2,3][i+1], yd[1,2,3][i+1], zd[1,2,3][i+1]
//                 matrices up to order max_n
 for (i=0; i<max_n-1; i++){
  hpow*=h;

  u[ 0][i]=xd[0][i];            //xd1
  u[ 1][i]=xd[1][i];            //xd2
  u[ 2][i]=yd[0][i];            //yd1
  u[ 3][i]=yd[1][i];            //yd2
  u[ 4][i]=zd[0][i];            //zd1
  u[ 5][i]=zd[1][i];            //zd2

  u[ 6][i]= x[0][i];            //x1
  u[ 7][i]= x[1][i];            //x2
  u[ 8][i]= y[0][i];            //y1
  u[ 9][i]= y[1][i];            //y2
  u[10][i]= z[0][i];            //z1
  u[11][i]= z[1][i];            //z2
//-------------------------------------------------------------------
```

```
  // construct R_12^(3/2), R_13^(3/2). R_23^(3/2)
  // form 1a, 2a, 1b, 2b, etc.
//-------------------------------------------------------------------------
  sum=0.; for (j=0; j<=i; j++)
          sum+=(u[ 7][i-j]-u[ 6][i-j])*(u[ 7][j]-u[ 6][j]);//(x2-x1)^2
  u[12][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
          sum+=(u[ 9][i-j]-u[ 8][i-j])*(u[ 9][j]-u[ 8][j]);//(y2-y1)^2
  u[13][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
          sum+=(u[11][i-j]-u[10][i-j])*(u[11][j]-u[10][j]);//(z2-z1)^2
  u[14][i]= sum;
//-------------------------------------------------------------------------
  if (i==0){
     u[15][0]=pow(u[12][0] + u[13][0] + u[14][0], 1.5);     //R_12^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[12][i-j] + u[13][i-j] + u[14][i-j])*u[15][
   sum/=(i*(u[12][0] + u[13][0] + u[14][0]));
   u[15][i]=sum;
  };
//-------------------------------------------------------------------------
  sum=0.; for (j=0; j<=i; j++)
   sum+=(N13*u[ 6][i-j]+mu23*u[ 7][i-j])*(N13*u[ 6][j]+mu23*u[ 7][j]);
                                          //(N13*x1+mu23*x2)^2
  u[16][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
   sum+=(N13*u[ 8][i-j]+mu23*u[ 9][i-j])*(N13*u[ 8][j]+mu23*u[ 9][j]);
                                          //(N13*y1+mu23*y2)^2
  u[17][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
   sum+=(N13*u[10][i-j]+mu23*u[11][i-j])*(N13*u[10][j]+mu23*u[11][j]);
                                          //(N13*z1+mu23*z2)^2
  u[18][i]= sum;
//-------------------------------------------------------------------------
  if (i==0){
     u[19][0]=pow(u[16][0] + u[17][0] + u[18][0], 1.5);     //R_13^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[16][i-j] + u[17][i-j] + u[18][i-j])*u[19][
   sum/=(i*(u[16][0] + u[17][0] + u[18][0]));
   u[19][i]=sum;
  };
//-------------------------------------------------------------------------
```

```
  sum=0.; for (j=0; j<=i; j++)
   sum+=(mu13*u[ 6][i-j]+N23*u[ 7][i-j])*(mu13*u[ 6][j]+N23*u[ 7][j]);
                                          //(mu13*x1+N23*x2)^2
  u[20][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
   sum+=(mu13*u[ 8][i-j]+N23*u[ 9][i-j])*(mu13*u[ 8][j]+N23*u[ 9][j]);
                                          //(mu13*y1+N23*y2)^2
  u[21][i]= sum;

  sum=0.; for (j=0; j<=i; j++)
   sum+=(mu13*u[10][i-j]+N23*u[11][i-j])*(mu13*u[10][j]+N23*u[11][j]);
                                          //(mu13*z1+N23*z2)^2
  u[22][i]= sum;
//-------------------------------------------------------------------
  if (i==0){
     u[23][0]=pow(u[20][0] + u[21][0] + u[22][0], 1.5);     //R_23^(3/2)
  } else {
   sum=0.;
   for (j=0; j<=i-1; j++)
    sum+= ((1.5)*i-j*(2.5))*(u[20][i-j] + u[21][i-j] + u[22][i-j])*u[23][
   sum/=(i*(u[20][0] + u[21][0] + u[22][0]));
   u[23][i]=sum;
  };
//-------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[24][i-j];// quotient
  sum+=u[ 7][i]-u[ 6][i]; sum/=u[15][0];                // 1a
  u[24][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[25][i-j];// quotient
  sum+=u[ 9][i]-u[ 8][i]; sum/=u[15][0];                // 2a
  u[25][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[26][i-j];// quotient
  sum+=u[11][i]-u[10][i]; sum/=u[15][0];                // 3a
  u[26][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[27][i-j];// quotient
  sum+=u[ 6][i]-u[ 7][i]; sum/=u[15][0];                // 4a
  u[27][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[28][i-j];// quotient
  sum+=u[ 8][i]-u[ 9][i]; sum/=u[15][0];                // 5a
  u[28][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[15][j]*u[29][i-j];// quotient
  sum+=u[10][i]-u[11][i]; sum/=u[15][0];                // 6a
  u[29][i]= sum;
//-------------------------------------------------------------------
  sum=0.; for (j=1; j<=i; j++) sum-=u[19][j]*u[30][i-j];// quotient
```

```
  sum+=M13*u[ 6][i]+m2*u[ 7][i]; sum/=u[19][0];          // 1b
  u[30][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[19][j]*u[31][i-j];// quotient
  sum+=M13*u[ 8][i]+m2*u[ 9][i]; sum/=u[19][0];          // 2b
  u[31][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[19][j]*u[32][i-j];// quotient
  sum+=M13*u[10][i]+m2*u[11][i]; sum/=u[19][0];          // 3b
  u[32][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[23][j]*u[33][i-j];// quotient
  sum+=m1*u[ 6][i]+M23*u[ 7][i]; sum/=u[23][0];          // 4b
  u[33][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[23][j]*u[34][i-j];// quotient
  sum+=m1*u[ 8][i]+M23*u[ 9][i]; sum/=u[23][0];          // 5b
  u[34][i]= sum;

  sum=0.; for (j=1; j<=i; j++) sum-=u[23][j]*u[35][i-j];// quotient
  sum+=m1*u[10][i]+M23*u[11][i]; sum/=u[23][0];          // 6b
  u[35][i]= sum;
//-------------------------------------------------------------------
  x[0][i+1]=u[0][i]/(i+1.);
  x[1][i+1]=u[1][i]/(i+1.);
  y[0][i+1]=u[2][i]/(i+1.);
  y[1][i+1]=u[3][i]/(i+1.);
  z[0][i+1]=u[4][i]/(i+1.);
  z[1][i+1]=u[5][i]/(i+1.);

  xd[0][i+1]=(m2*u[24][i]-u[30][i])/(i+1.);
  yd[0][i+1]=(m2*u[25][i]-u[31][i])/(i+1.);
  zd[0][i+1]=(m2*u[26][i]-u[32][i])/(i+1.);

  xd[1][i+1]=(m1*u[27][i]-u[33][i])/(i+1.);
  yd[1][i+1]=(m1*u[28][i]-u[34][i])/(i+1.);
  zd[1][i+1]=(m1*u[29][i]-u[35][i])/(i+1.);


// ! consider the third body also
  if(hpow*(fabs( x[0][i+1])+fabs( x[1][i+1])+\
           fabs( y[0][i+1])+fabs( y[1][i+1])+\
           fabs( z[0][i+1])+fabs( z[1][i+1])+\
           fabs(xd[0][i+1])+fabs(xd[1][i+1])+\
           fabs(yd[0][i+1])+fabs(yd[1][i+1])+\
           fabs(zd[0][i+1])+fabs(zd[1][i+1])+\
           fabs((m1*x[0][i+1]+m2*x[1][i+1])/m3)+\
           fabs((m1*y[0][i+1]+m2*y[1][i+1])/m3)+\
           fabs((m1*z[0][i+1]+m2*z[1][i+1])/m3)+\
           fabs((m1*xd[0][i+1]+m2*xd[1][i+1])/m3)+\
```

```
            fabs((m1*yd[0][i+1]+m2*yd[1][i+1])/m3)+\
            fabs((m1*zd[0][i+1]+m2*zd[1][i+1])/m3))\
      <tol) {loopsdone=i+1; break;}
  loopsdone=i+1;
 }

 new_h=h;
 if(loopsdone<10) new_h=min(1.5*h,max_h);
 if(loopsdone>20) new_h=0.5*h;
 if(loopsdone==max_n) return -h;

// update positions & velocities of m1, m2
 hpow=1.;
 for(i=1; i<loopsdone; i++){
  hpow*=h;
  x[0][0]+=  x[0][i]*hpow;
  x[1][0]+=  x[1][i]*hpow;
  y[0][0]+=  y[0][i]*hpow;
  y[1][0]+=  y[1][i]*hpow;
  z[0][0]+=  z[0][i]*hpow;
  z[1][0]+=  z[1][i]*hpow;

  xd[0][0]+=xd[0][i]*hpow;
  xd[1][0]+=xd[1][i]*hpow;
  yd[0][0]+=yd[0][i]*hpow;
  yd[1][0]+=yd[1][i]*hpow;
  zd[0][0]+=zd[0][i]*hpow;
  zd[1][0]+=zd[1][i]*hpow;

 };
 //update position & velocity of the third body
 x[2][0]=   -(m1* x[0][0]+m2* x[1][0])/m3;
 y[2][0]=   -(m1* y[0][0]+m2* y[1][0])/m3;
 z[2][0]=   -(m1* z[0][0]+m2* z[1][0])/m3;
 xd[2][0]=  -(m1*xd[0][0]+m2*xd[1][0])/m3;
 yd[2][0]=  -(m1*yd[0][0]+m2*yd[1][0])/m3;
 zd[2][0]=  -(m1*zd[0][0]+m2*zd[1][0])/m3;


#ifndef pgplot_graphics
 printf("E_0= %3.15lf\tE(t)= %15.14lf\t%15.14lf\n",\
                E_0,Energy(),fabs(Energy()-E_0));
 printf("time: %5.20lf\t order %d. step: %5.20lf\n",\
                t,loopsdone,h);
#endif
 return new_h;
}; // end of do_a_step()


int main(){
```

```
double time,h,ah;
double x0,y0;// for pgplot panels

FILE *fil;

m1=0.00078;
m2=0.00019;

//this for fancy plots...
m1=1./3.;
m2=1./3.;
//m1=1e-1;
//m2=3e-1;
m3=1.-m1-m2;

M13=m1+m3;
M23=m2+m3;
mu23=m2/m3;
mu13=m1/m3;

N13=1.+mu13;
N23=1.+mu23;



x[0][0]=  1.0;
y[0][0]=  1.0;
z[0][0]=  1.0;

x[1][0]=  -2.;
y[1][0]=   0.;
z[1][0]=  -1.;

x[2][0]= -(m1*x[0][0]+m2*x[1][0])/m3;
y[2][0]= -(m1*y[0][0]+m2*y[1][0])/m3;
z[2][0]= -(m1*z[0][0]+m2*z[1][0])/m3;

xd[0][0]=   0.2;
yd[0][0]=   -0.1;
zd[0][0]=   0.0;

xd[1][0]=   0.01;
yd[1][0]=   0.0;
zd[1][0]=  -0.01;


xd[2][0]=   -(m1*xd[0][0]+m2*xd[1][0])/m3;
yd[2][0]=   -(m1*yd[0][0]+m2*yd[1][0])/m3;
zd[2][0]=   -(m1*zd[0][0]+m2*zd[1][0])/m3;
```

```
//-------------------------------
 time = 200;
 h=1e-2;
 t=0.;

 E_0= Energy();
 printf("E=%f\n",E_0); getchar();

#ifdef pgplot_graphics
 x0=50.; y0=100.; junk=0;
 srand(4105);
 if(cpgbeg(0, "/xw", 1, 1) != 1)
    exit(EXIT_FAILURE);
 init_graphics((float)(x0),(float)(y0),
               (float)(1.1*time),(float)(x0),
               (float)(x0),(float)(4),
               (float)(10),(float)(10),
               (float)(1.1*time),(float)(y0),
               (float)(x0),(float)(3)
              );
#else
 fil=fopen("ttt.dat","wt");
#endif

 while(t<time){
  if ((ah=do_a_step(h))>0) { t+=h; h=ah; }
  else { h=-ah; };
#ifdef pgplot_graphics
  junk=draw((float)(x0),(float)(y0),
               (float)(1.1*time),(float)(x0),
               (float)(x0),(float)(4),
               (float)(10),(float)(10),
               (float)(1.1*time),(float)(y0),
               (float)(y0),(float)(3),h,junk);
#else
  fprintf(fil,"%f %f %f %f %f %f %f %f %f %f %f %f\n",\
                  t, x[0][0], y[0][0], z[0][0], \
                     x[1][0], y[1][0], z[1][0], \
                     x[2][0], y[2][0], z[2][0], \
                    xd[0][0],yd[0][0],zd[0][0], \
                    xd[1][0],yd[1][0],zd[1][0], \
                    xd[2][0],yd[2][0],zd[2][0]);
#endif
 };
#ifdef pgplot_graphics
 getchar();
 cpgend();
#else
 fclose(fil);
#endif
```

```
 return 0;
};
//=============================================================
//=============================================================
#ifdef pgplot_graphics
 void init_graphics(float ulx, float uly,
                    float umx, float umy,
                    float urx, float ury,
                    float llx, float lly,
                    float lmx, float lmy,
                    float lrx, float lry){
  cpgsci(1);
  cpgpap(12,0.8618);
  cpgsch(1.8);//determines fontsize
  cpgsubp(3,2);
  cpgenv(-ulx/2.,ulx/2.,-uly/2.,uly/2., 0, 1);
  cpglab("(x\\di\\u)", "(y\\di\\u)",
   "3 body configuration space");

  cpgenv(0,umx,-umy/2.,umy/2., 0, 1);
  cpglab("(t)", "(x\\di\\u)",
   "x\\di\\u timeseries");

  cpgenv(-urx/2.,urx/2.,-ury/2.,ury/2., 0, 1);
  cpglab("(x\\di\\u)", "(x\\di\\u_dot)",
   "x\\di\\u-x\\di\\u_dot phase space");

  cpgsci(0);
  cpgenv(-llx/2.,llx/2.,-lly/2.,lly/2., 0, 1);
  cpgsci(1);
  cpglab("", "",
   "details panel");

  cpgenv(0,lmx,-lmy/2.,lmy/2., 0, 1);
  cpglab("(t)", "(y\\di\\u)",
   "y\\di\\u timeseries");

  cpgenv(-lrx/2.,lrx/2.,-lry/2.,lry/2., 0, 1);
  cpglab("(y\\di\\u)", "(y\\di\\u_dot)",
   "y\\di\\u-y\\di\\u_dot phase space");

  cpgsci(1); cpgask(0);
 };// end of init_graphics(...)

//-------------------------------------------------------------
 int draw(float ulx, float uly,
          float umx, float umy,
          float urx, float ury,
          float llx, float lly,
          float lmx, float lmy,
```

```
          float lrx, float lry,
          float h, int duration){
cpgpanl(1,1);
cpgswin(-ulx/2.,ulx/2.,-uly/2.,uly/2.);
if (duration==10000) {
 duration=-1;
 x1y1=(int)(10.*rand()/RAND_MAX); if (x1y1==0||x1y1==1) x1y1+=2;
 x2y2=(int)(10.*rand()/RAND_MAX); if (x2y2==0||x2y2==1) x2y2+=2;
 x3y3=(int)(10.*rand()/RAND_MAX);
 // erase trajectory
 cpgpanl(1,1);
 cpgsci(0);
 cpgrect(-0.99*ulx/2.,0.99*ulx/2.,-0.99*uly/2.,0.99*uly/2.);
 cpgsci(1);
 cpgbox("ABCNST",0,0,"ABCNST",0,0);
};
cpgsci(x1y1); cpgpt1((float)x[0][0],(float)y[0][0],1);// x1 - y1
cpgsci(x2y2); cpgpt1((float)x[1][0],(float)y[1][0],1);// x2 - y2
cpgsci(x3y3); cpgpt1((float)x[2][0],(float)y[2][0],9);// x3 - y3

cpgpanl(2,1);
cpgswin(0,umx,-umy/2.,umy/2.);
cpgsci(2); cpgpt1((float)t,(float)x[0][0],1);          // t - x1
cpgsci(3); cpgpt1((float)t,(float)x[1][0],1);          // t - x2
cpgsci(7); cpgpt1((float)t,(float)x[2][0],1);          // t - x3


cpgpanl(3,1);
cpgswin(-urx/2.,urx/2.,-ury/2.,ury/2.);
cpgsci(2); cpgpt1((float)x[0][0],(float)xd[0][0],1);   // x1 - x1_dot
cpgsci(3); cpgpt1((float)x[1][0],(float)xd[1][0],1);   // x2 - x2_dot
cpgsci(7); cpgpt1((float)x[2][0],(float)xd[2][0],9);   // x3 - x3_dot

cpgpanl(1,2);
cpgswin(-llx/2.,llx/2.,-lly/2.,lly/2.);
cpgsci(0);
//cpgrect(-0.9*llx/2.,0.9*llx/2.,-0.9*lly/2.,0.9*lly/2.);
cpgrect(-0.99*llx/2.,0.99*llx/2.,-0.99*lly/2.,0.99*lly/2.);
cpgsci(1);
sprintf(s,"Time_: %3.7lf",t);
cpgmtxt("RV",-18.0,0.9,0.0,s);nc=20;
sprintf(s,"Step_: %3.7lf",h);
cpgmtxt("RV",-18.0,0.8,0.0,s);nc=20;
sprintf(s,"E(0)_:%3.20lf",E_0);
cpgmtxt("RV",-18.0,0.7,0.0,s);nc=20;
sprintf(s,"E(t)_: %3.20lf",Energy());
cpgmtxt("RV",-18.0,0.6,0.0,s);nc=20;
sprintf(s,"|E(t)-E_0| :");
cpgmtxt("RV",-18.0,0.5,0.0,s);nc=20;
sprintf(s,"%3.20lf",fabs(Energy()-E_0));
```

```
 cpgmtxt("RV",-14.0,0.4,0.0,s);nc=20;

 cpgpanl(2,2);
 cpgswin(0,umx,-lmy/2.,lmy/2.);
 cpgsci(2); cpgpt1((float)t,(float)y[0][0],1);          // t - y1
 cpgsci(3); cpgpt1((float)t,(float)y[1][0],1);          // t - y2
 cpgsci(7); cpgpt1((float)t,(float)y[2][0],1);          // t - y3

 cpgpanl(3,2);
 cpgswin(-lrx/2.,lrx/2.,-lry/2.,lry/2.);
 cpgsci(2); cpgpt1((float)y[0][0],(float)yd[0][0],1);   // y1 - y1_dot
 cpgsci(3); cpgpt1((float)y[1][0],(float)yd[1][0],1);   // y2 - y2_dot
 cpgsci(7); cpgpt1((float)y[2][0],(float)yd[2][0],9);   // y3 - y3_dot

 return ++duration;
 };//end of draw(...)
#endif
```

6 Βιβλιογραφία

[1] O. Aberth, "Precise Numerical Methods using C++", Academic Press, London 1998.

[2] A Collection of Automatic Differentiation Tools:
http://www-unix.mcs.anl.gov/autodiff/AD_Tools/index.html

[3] R. Barrio, F. Blesa and M. Lara, "High-precision numerical solution of ODE with higher-order Taylor methods in parallel" *Monografías de la Real Academia de Ciencias de Zaragoza* ***22***, (2003), 67–74.

[4] C. Brezinski, L. Wuytack, "Numerical Analysis in the Twentieth Century "
http://citeseer.ist.psu.edu/557824.html

[5] G. Corliss, Y.F. Chang, "Solving Ordinary Differential Equations Using Taylor Series", *ACM Trans. Mathematical Software Vol.**8**, No. 2*, (1982) 114–144.

[6⊗] Dynamical Systems Homepage:
http://www.dynamicalsystems.org

[7] K. Fox, "Numerical integration of the equations of motion of celestial mechanics", *Celestial Mechanics* ***33***, (1984) 127–142.

[8] GNU Multiple Precision Arithmetic Library Homepage:
http://www.swox.com/gmp/

[9] GNU Scientific Library (gsl) Homepage:
http://www.gnu.org/software/gsl/
Documentation for ODE initial values problem solvers:
http://www.gnu.org/software/gsl/manual/gsl-ref_25.html#SEC391

[10] H. Goldstein, "Classical Mechanics", Addison-Wesley Series in Physics, 2^{nd} ed., 1980.

[11] K.G. Hadjifotinou "Numerical integration of the variational equations of satellite orbits", *Planetary and Space Science* ***50***, (2002) 361–369.

[12] K.G. Hadjifotinou & M. Gousidou-Koutita "Comparison of numerical methods for the integration of natural satellite systems", *Celestial Mechanics and Dynamical Astronomy* ***70***, (1998) 99-113.

[13] M. Hirsch & S. Smale, "Differential Equations, Dynamical Systems, and Linear Algebra", Academic Press, Pure and Applied Mathematics Series, Vol. 60, 1974.

[14] T.Y. Huang, K. Innanen "A survey of multiderivative multistep integrators", *The Astronomical Journal* ***112*** *(3)*, (1996) 1254–1262.

[15] IMA Automatic Differentiation Homepage:
http://www.math.umn.edu/∼santosa/adhome.html

[16] À. Jorba & M. Zou, "A software package for the numerical integration of ODEs by means of high–order Taylor methods", *Experimental Mathematics* ***14***, (2005) 99-117.

[17] Cl. Le Guyader "Solution of the N–body problem expanded into Taylor series of high orders. Applications to the solar system over large time range", *Astron. Astrophys.* ***272***, (1993) 687–694.

[18] A. Milani, A. Nobili, "Integration error over very long time spans", *Celestial Mechanics* ***43***, (1988) 1–34.

[19] M.A. Murison "On an efficient and accurate method to integrate restricted three-body orbits", *The Astronomical Journal* ***97*** *(5)*, (1989) 1496–1509.

[20] D.N. Papadakos, "Generalized F and G series and convergence of the power series solution of the N–body problem", *Celestial Mechanics* ***30***, (1983) 275–282.

[21] PGPLOT Graphics Subroutine Library:
http://www.astro.caltech.edu/∼tjp/pgplot/

[22] Th. Quinn, S. Tremaine "Roundoff error in long-term planetary orbit integrations", *The Astronomical Journal* ***99*** *(3)*, (1990) 1016–1023.

[23] C. Roberts, Jr. "Comments on the application of power series solutions to problems in celestial mechanics", *Celestial Mechanics* ***12***, (1975) 397-407.

[24] A.E. Roy, "Orbital Motion", Hilger Ltd, 2^{nd} ed., Bristol, UK 1982.

[25] A. D A M Spallicci, A. Morbidelli & G. Metris, "The three–body problem and the Hannay angle", *Nonlinearity* ***18***, (2005) 45–54.

[26] V. Szebehely, "Theory of Orbits: The Restricted Problem of Three Bodies", Academic Press, NY 1967.

[27] Ι. Χατζηδημητρίου, «Θεωρητική Μηχανική», Τόμος Α′ (Νευτώνια Μηχανική), $3^{η}$ έκδοση, Γιαχούδη–Γιαπούλη, Θεσσαλονίκη 2000.

Κατάλογος Σχημάτων

Περιεχόμενα