1
Αριθμητική Κινητής Υποδιαστολής και Συμβολική Υπολογιστική Άλγεβρα

Δημήτριος Χριστόπουλος[1,2]

[1]Εθνικό Καποδιστριακό Πανεπιστήμιο Αθηνών, Τμήμα Οικονομικών Επιστημών
[2]dchristop@econ.uoa.gr

Άνοιξη 2011

Σημειώσεις Εργαστηρίου Γραμμικών Μαθηματικών[3].

[3]Οι ηλεκτρονικές σημειώσεις που ακολουθούν περιέχουν υπερσυνδέσεις, με ένα απλό κλικ, εσωτερικά ή εξωτερικά του κειμένου.

Περιεχόμενα

1 Αριθμητική Κινητής Υποδιαστολής και Συμβολική Υπολογιστική Άλγεβρα **3**

1.1 Οι αριθμοί κινητής υποδιαστολής 3

1.2 Το σφάλμα στις στοιχειώδεις αριθμητικές πράξεις 6

1.2.1 Η δημιουργία του σφάλματος στρογγύλευσης (rounding) . . 6

1.2.2 Το πρόβλημα της αναπαράστασης ενός δεκαδικού αριθμού από τον πλησιέστερο ρητό 9

1.2.3 Η διάδοση (propagation) του σφάλματος στρογγύλευσης . . 14

1.3 Η Συμβολική Υπολογιστική Άλγεβρα 16

1.4 Ασκήσεις 20

1 Αριθμητική Κινητής Υποδιαστολής και Συμβολική Υπολογιστική Άλγεβρα

1.1 Οι αριθμοί κινητής υποδιαστολής

Από την εμφάνιση των πρώτων υπολογιστών έχουν εισαχθεί οι λεγόμενοι αριθμοί κινητής υποδιαστολής, η γενική μορφή των οποίων είναι:

$$x = \pm\,(0.d_1 d_2 \dots d_t) \cdot \beta^e \tag{1}$$

Το β είναι η βάση του αριθμητικού συστήματος ($\beta = 2, 10$ ή 16) και ο εκθέτης e παίρνει τιμές από L έως U ανάλογα με το πρότυπο που χρησιμοποιείται. Το t είναι το πλήθος των σημαντικών ψηφίων και ονομάζεται 'mantissa'. Το σύστημα λειτουργίας των υπολογιστών είναι το λεγόμενο δυαδικό (binary), το οποίο ουσιαστικά διαθέτει δύο καταστάσεις, 0 :*δεν διέρχεται ηλεκτρικό ρεύμα* και 1 :*διέρχεται ηλεκτρικό ρεύμα* από κάποιο τρανζίστορ, το οποίο αποτελεί τον δομικό λίθο κάθε επεξεργαστή. Η ανωτέρω ποσότητα πληροφορίας 0 ή 1 ονομάζεται 1 bit. Το πρότυπο που χρησιμοποιείται σήμερα από τα περισσότερα προγράμματα αριθμητικής κινητής υποδιαστολής και από όλες σχεδόν τις γλώσσες προγραμματισμού για αριθμητικές πράξεις είναι το *IEEE 754-2008*[1] και πιο συγκεκριμένα το πρότυπο *binary64*. Σε αυτό το πρότυπο διπλής ακρίβειας, έχουμε ότι ένας αριθμός καταλαμβάνει $64\,bits = 8\,bytes$ στην μνήμη του υπολογιστή:

- 1 bit καταλαμβάνει το πρόσημο $0 \to (-1)^0 = +1$ ή $1 \to (-1)^1 = -1$
- 11 bits καταλαμβάνει ο εκθέτης του 2
- 52 bits καταλαμβάνει η mantissa

Επίσης έχουμε ότι $\beta = 2, L = -1022, U = +1023$ το οποίο αντιστοιχεί σε 16 σημαντικά ψηφία στην συνηθισμένη δεκαδική αναπαράσταση των αριθμών. Σε κάθε πρότυπο $\{\beta, t, L, U\}$ ο πλησιέστερος αριθμός στο μηδέν είναι ο αριθμός $.1 \cdot \beta^L$ και ο πλησιέστερος αριθμός στο $+\infty$ είναι ο β^U. Στην πράξη πάντως, παίρνοντας το δεκαδικό σύστημα, έχουμε ότι $-\infty \approx -10^{308}$, $0 \approx 10^{-308}$ και $\infty \approx 10^{308}$. Παραδείγματα αριθμών μηχανής:

- Με βάση το $\beta = 10$

$$14.75 = 10 + 4 + \frac{7}{10} + \frac{5}{100} = \mathbf{1} \cdot 10^1 + \mathbf{4} \cdot 10^0 + \mathbf{7} \cdot 10^{-1} + \mathbf{5} \cdot 10^{-2} = (14.75)_{10}$$

- Με βάση το $\beta = 2$

$$14.75 = 8{+}4{+}2{+}\frac{1}{2}{+}\frac{1}{4} = \mathbf{1}{\cdot}2^3{+}\mathbf{1}{\cdot}2^2{+}\mathbf{1}{\cdot}2^1{+}\mathbf{0}{\cdot}2^0{+}\mathbf{1}{\cdot}2^{-1}{+}\mathbf{1}{\cdot}2^{-2} = (1110.11)_2$$

[1] Institute of Electrical and Electronics Engineers, περισσότερες λεπτομέρειες εδώ.

Όταν πληκτρολογείτε τον αριθμό -12345.678 σε έναν υπολογιστή με $t = 8$ αυτός μετατρέπεται αμέσως εσωτερικά στην μορφή $-.12345678 \cdot 10^5$.

Επίσης πρέπει να έχετε υπόψιν σας ότι το πλήθος των σημαντικών ψηφίων t ενός υπολογιστή είναι ανεξάρτητο του τρόπου γραφής του αριθμού λ.χ. στην οθόνη. Σαν παράδειγμα ο αριθμός 12.345678 σε έναν υπολογιστή με $t = 8$ μπορεί να εμφανιστεί είτε σαν $.12345678 \cdot 10^{+2}$ είτε σαν $1.2345678 \cdot 10^{+1}$. Το τελευταίο είναι αυτό που έχουμε συνηθίσει να λέμε *επιστημονική γραφή*, αλλά δεν πρέπει να ξεχνάτε ότι τα σημαντικά ψηφία είναι 8. Εάν στο MATLAB γράψετε τον παραπάνω αριθμό ενώ έχετε βάλει πρώτα την εντολή >>format long η οποία δείχνει τους αριθμούς με $t = 16$ σημαντικά ψηφία θα δείτε σαν απάντηση:

```
ans =

   12.345677999999999
```

Πόσα από τα ανωτέρω ψηφία μπορούμε να εμπιστευθούμε; Η απάντηση είναι 8 ξεκινώντας από το 1 ενώ το τελευταίο προκύπτει με στρογγυλοποίηση, δηλ.:
12.345677999999999 = **12.345678**000000000
(με έντονη γραφή δείχνουμε τα σημαντικά ψηφία). Στο ίδιο πρόγραμμα εάν είχαμε επιλέξει >>format long e ο ίδιος αριθμός θα εμφανιζόταν σωστά ως:

```
ans =

     1.234567800000000e+001
```

Τώρα ο υπολογιστής εμφανίζει τον αριθμό με 8 σημαντικά ψηφία, ενώ τα υπόλοιπα 8 μέχρι τα 16 διαθέσιμα παίρνουν την τιμή 0. Βλέπετε λοιπόν ότι για να έχετε κομψά αποτελέσματα πρέπει να γνωρίζετε πως αντιλαμβάνεται ένας υπολογιστής κινητής υποδιαστολής τους αριθμούς. Επίσης είναι φανερό ότι δεν είναι δυνατός ο χειρισμός ενός αριθμού της μορφής 10^{309} γιατί αντιστοιχεί στο $+\infty \to Inf$. Σε αυτές τις περιπτώσεις το πρότυπο *binary64* έχει συγκεκριμένο τρόπο χειρισμού, π.χ.:

```
>> Inf+25

ans =

   Inf
```

Το MATLAB έχει τις εντολές

```
>> realmin

ans =

    2.225073858507201e-308

>> realmax

ans =

    1.797693134862316e+308
```

που δίνουν τον μικρότερο και τον μεγαλύτερο θετικό αριθμό, $2.225073858507201 \cdot 10^{-308}$ και $1.797693134862316 \cdot 10^{+308}$ αντίστοιχα. Εάν κάποιος θεωρεί ότι είναι ικανοποιημένος με αριθμούς σε αυτό το εύρος, τότε πιθανόν να μην κατανοεί τον σκοπό συγγραφής του παρόντος κεφαλαίου. Θα δούμε όμως ότι το περιορισμένο εύρος των αριθμών μηχανής δεν είναι το μοναδικό τους πρόβλημα.

Ένα σημαντικό ερώτημα είναι το εξής:

Πόσο κοντά μπορούμε να πλησιάσουμε σε έναν αριθμό μηχανής;

Για να απαντήσουμε σε αυτό το ερώτημα ξεκινάμε από έναν αριθμό π.χ. από το 1 και προσθέτουμε κάθε φορά έναν αριθμό $\epsilon > 0$ ο οποίος διαρκώς μικραίνει μέχρι τελικά στους αριθμούς μηχανής του υπολογιστή να ισχύσει $1 + \epsilon = 1$, δηλ. τότε το ϵ που θα έχουμε βρει θα είναι η μικρότερη απόσταση στην οποία μπορούμε να πλησιάσουμε κοντά στο 1. Στο MATLAB γράφουμε τις ακόλουθες εντολές:

```
>> e=1;
>> while (1+e)>1
em=e;
e=e/2;
end
>> em

em =

    2.220446049250313e-016
```

Αυτό που βρήκαμε είναι το *έψιλον της μηχανής* και είναι εξαιρετικά σημαντικό διότι καθορίζει το σφάλμα διακριτοποίησης όλων των αριθμητικών μεθόδων. Πρέπει να σημειωθεί ότι ακριβώς το ίδιο ϵ διαθέτει λ.χ. το Mathematica [2], το οποίο είναι $\$MachineEpsilon = 2.22045 * 10^{-16}$.

Χρησιμοποιώντας όμως στο Mathematica τις εντολές:

[2] Σήμα κατατεθέν της Wolfram Research, Inc., Champaign, Illinois, USA.

e = 1; i = 0; While[1 + e >1, e = e/(10^308); i++]; {e, i} και διακόπτοντας την εκτέλεση μετά από 20.717 sec βρίσκουμε, σε $i = 3930$ βήματα, $e = 1. \cdot 10^{-1210440}$, δηλ. πρακτικά μπορούμε να πλησιάζουμε οσοδήποτε κοντά στο 1 αρκεί να διαθέτουμε υπολογιστική ισχύ/χρόνο. Αυτό είναι ένα πρώτο μεγάλο άλμα των μεθόδων της Υπολογιστικής Συμβολικής Άλγεβρας έναντι των μεθόδων της Αριθμητικής Κινητής Υποδιαστολής.

1.2 Το σφάλμα στις στοιχειώδεις αριθμητικές πράξεις

Θα αναφέρουμε συνοπτικά χωρίς αποδείξεις ορισμένα χρήσιμα συμπεράσματα σχετικά με το σφάλμα που γίνεται όταν κάνουμε τις συνήθεις πράξεις με αριθμούς μηχανής. Ο αναγνώστης που ενδιαφέρεται για περαιτέρω εμβάθυνση καλείται να ανατρέξει είτε στα βιβλία Ακρίβης & Δουγαλής (2005), Stoer & Bulirsch (2002) , Hildebrand, Prausnitz & Scott (1970) είτε στο άρθρο Goldberg (1991).

1.2.1 Η δημιουργία του σφάλματος στρογγύλευσης (rounding)

Όταν ένας αριθμός είναι άθροισμα δυνάμεων του 2, τότε είναι λογικό να αναμένουμε ότι στην δυαδική αναπαράσταση δεν θα έχουμε σφάλμα, αρκεί να βρίσκεται εντός των ορίων αριθμών μηχανής του προτύπου μας. Το πρόβλημα είναι ότι δεν χρησιμοποιούμε στην καθημερινή ζωή το δυαδικό, αλλά το δεκαδικό σύστημα. Εάν ανατρέξετε στο διαδικτυακό εργαλείο της IEEE για την μετατροπή αριθμών σε γλώσσα μηχανής και εισάγετε ορισμένους στοιχειώδεις ρητούς αριθμούς όπως το $\frac{1}{3} = .333333333333333\ldots$ θα δείτε ότι δεν αναπαρίσταται ακριβώς, αλλά καταλαμβάνει όλα τα διαθέσιμα σημαντικά ψηφία[3] με την μορφή $0.010101010101\ldots$. Αναγκαστικά το πρότυπο binary64 θα προσεγγίσει τον αριθμό με βάση το τελευταίο σημαντικό ψηφίο του. Είθισται αυτό να γίνεται με στρογγυλοποίηση (rounding) προς τον πλησιέστερο ακέραιο με τον κανόνα του '5', δηλ. οτιδήποτε είναι <5 γίνεται 0 και οτιδήποτε είναι ≥ 5 γίνεται 10, δηλ. αυξάνεται ο αριθμός στο επόμενο δεκαδικό ψηφίο. Εάν $t = 8$ και είμαστε στο δεκαδικό σύστημα τότε ο αριθμός $+\mathbf{0.12345678}4 \rightarrow +0.12345678$, ενώ ο αριθμός $+\mathbf{0.12345678}5 \rightarrow +0.12345679$.

Κάθε αριθμός x που κατ' απόλυτο τιμή βρίσκεται στο διάστημα $\left[.1 \cdot \beta^L, \beta^U\right]$ προσεγγίζεται από τον πλησιέστερο σε αυτόν διαθέσιμο αριθμό μηχανής, συμβολικά $fl(x)$. Το σχετικό σφάλμα αυτής της προσέγγισης μπορεί να δειχθεί ότι είναι:

$$|\epsilon_x| = \left|\frac{fl(x) - x}{x}\right| \leq \frac{1}{2}\beta^{1-t} \tag{2}$$

βλέπε Ακρίβης & Δουγαλής (2005) ή Stoer & Bulirsch (2002). Επίσης μπορούμε να αποκόπτουμε τα δεκαδικά ψηφία που 'περισσεύουν', οπότε ένας αριθμός στρογ-

[3] Πηγαίνετε στο πλαίσιο "Significand" του "Double precision (64 bits)" και πολλαπλασιάστε τον αριθμό με 2^{-2} όπου -2 ο εκθέτης.

γυλεύεται στον πλησιέστερο αριθμό μηχανής και τότε το σχετικό σφάλμα είναι:

$$|\epsilon_x| = \left|\frac{fl(x) - x}{x}\right| \leq \beta^{1-t} \tag{3}$$

Ορίζουμε την ποσότητα *μοναδιαίο σφάλμα στρογγύλευσης* ως εξής:

$$u = \begin{cases} \frac{1}{2}\beta^{1-t} & \text{στρογγύλευση} \\ \beta^{1-t} & \text{αποκοπή} \end{cases} \tag{4}$$

Με όλους τους παραπάνω ορισμούς, αν θεωρήσουμε την πράξη *, ο αριθμός μηχανής που θα προκύψει στο τέλος κάνοντας την πράξη $x * y$ στον υπολογιστή θα είναι $z = fl(fl(x) * fl(y))$.

Παράδειγμα 1.1. *Ας θεωρήσουμε έναν υπολογιστή όπου* $\beta = 10, t = 3$ *και τους αριθμούς μηχανής* $x = 1 = .1 \cdot 10^{+1}, y = .003 = 3 \cdot 10^{-3} = .3 \cdot 10^{-2}, z = .004 = 4 \cdot 10^{-3} = .4 \cdot 10^{-2}$. *Τότε έχουμε ότι* $y + z = .7 \cdot 10^{-2} = 7 \cdot 10^{-3} = .007$, *άρα* $x + (y + z) = 1.007 = .101 \cdot 10^{+1} = 1.01$, *λόγω στρογγύλευσης του 7 προς τα πάνω. Όμως έχουμε ακόμη ότι:* $x + y = 1.003 = .100 \cdot 10^{+1} = 1.00$, *λόγω στρογγύλευσης του 3 προς τα κάτω, άρα* $(x+y)+z = 1.003 = 1.00 \neq x + (y + z)$, *δηλ. η πρόσθεση παύει να είναι προσεταιριστική.*

Περισσότερα παραδείγματα πάνω στα 'παράδοξα' των αριθμών μηχανής μπορείτε να βρείτε στα αγγλικά κάνοντας κλικ κι εδώ.

Μπορούμε να υπολογίσουμε τα σφάλματα στρογγύλευσης στις βασικές πράξεις και να τα συγκρίνουμε με το u, δηλ. το *μοναδιαίο σφάλμα στρογγύλευσης*. Τότε έχουμε συνοπτικά τα ακόλουθα συμπεράσματα:

1. Πολλαπλασιασμός ή Διαίρεση
   Στον πολλαπλασιασμό ή στην διαίρεση δύο αριθμών το σχετικό σφάλμα που προκύπτει είναι μικρότερο ή ίσο του $3u$.

2. Πρόσθεση ομοσήμων αριθμών
   Στην πρόσθεση ομοσήμων αριθμών το σχετικό σφάλμα είναι μικρότερο ή ίσο του $2u$.

3. Πρόσθεση ετεροσήμων αριθμών (αφαίρεση)
   Στην πρόσθεση ετεροσήμων αριθμών το σχετικό σφάλμα είναι μικρότερο ή ίσο του $\frac{|x|+|y|}{|x+y|}2u$.

Βλέπουμε λοιπόν ότι σε κάθε περίπτωση που θα χρειαστεί να κάνουμε αφαίρεση αριθμών αρκετά κοντινών μεταξύ τους, επειδή τότε $|x + y| \approx 0$, το ανωτέρω κλάσμα του σχετικού σφάλματος γίνεται πολύ μεγάλο, με αποτέλεσμα την απώλεια ή καταστροφή της όποιας ακρίβειας. Ας υποθέσουμε ότι κάποιος θέλει να κάνει την πράξη $\sqrt{x} - \sqrt{y}$ με $x \approx y$. Πως θα αποφύγουμε το μεγάλο σφάλμα που προκύπτει ;

Παράδειγμα 1.2. *Ορίστε στο MATLAB τους αριθμούς* $x = 10025.01562 = (100.125^2)$, $y = 10024.81538 = (100.124^2)$, *οπότε προφανώς* $\sqrt{x} - \sqrt{y} = 0.001 = 1. \cdot 10^{-3}$ *και δοκιμάστε να βρείτε το αποτέλεσμα.*

Μόλις κάνετε την πράξη στο MATLAB βρίσκετε:

$$\sqrt{x} - \sqrt{y} = \mathbf{9.99955055982}664 \cdot 10^{-4}$$

Ας πολλαπλασιάσουμε και διαιρέσουμε με την 'συζυγή ποσότητα' $\sqrt{x} + \sqrt{y}$ *και ας υπολογίσουμε:*

$$\sqrt{x} - \sqrt{y} = \frac{x - y}{\sqrt{x} + \sqrt{y}} = \mathbf{9.99955055982}08786 \cdot 10^{-4}$$

Δηλαδή έχουμε μία οριακή βελτίωση της ακρίβειας από το 10^o *δεκαδικό ψηφίο και μετά.*
Εάν κάνουμε τις αντίστοιχες πράξεις στο Mathimatica βρίσκουμε:

$$\sqrt{x} - \sqrt{y} = \mathbf{0.000999955055982}664$$

$$\sqrt{x} - \sqrt{y} = \frac{x - y}{\sqrt{x} + \sqrt{y}} = \mathbf{0.000999955055982}08789$$

Τα αποτελέσματα είναι σχεδόν ίδια. Για να αποφύγουμε αυτό το σφάλμα στο Mathimatica πρέπει να μετατρέψουμε τους αριθμούς σε ρητούς και κατόπιν να κάνουμε συμβολικά πράξεις με απόλυτη ακρίβεια:

$$x = \left(100 + \frac{125}{1000}\right)^2 = \frac{641601}{64}$$

$$y = \left(100 + \frac{124}{1000}\right)^2 = \frac{626550961}{62500}$$

$$\sqrt{x} - \sqrt{y} = \frac{1}{1000} = 0.001$$

Το τέχνασμα της μετατροπής σε ρητό πρέπει να εφαρμόσουμε για να κάνουμε με απόλυτη ακρίβεια την ίδια πράξη και στα προγράμματα Axiom [4] και wxMaxima [5]. Εάν χρησιμοποιήσουμε το Maple [6] τότε μπορούμε να έχουμε άμεσα το απόλυτα ακριβές αποτέλεσμα, χωρίς να χρειαστεί πρώτα να μετατρέψουμε τους αριθμούς x, y σε ρητούς.

[4] Ελεύθερο λογισμικό, μπορείτε να το 'κατεβάσετε' από εδώ ή εδώ, ανάλογα με το λειτουργικό που χρησιμοποιείτε.

[5] Ελεύθερο λογισμικό, μπορείτε να το 'κατεβάσετε' από εδώ.

[6] Σήμα κατατεθέν της Waterloo Maple Inc., Waterloo, Ontario, CANADA.

1.2.2 Το πρόβλημα της αναπαράστασης ενός δεκαδικού αριθμού από τον πλησιέστερο ρητό

Εάν χρησιμοποιήσουμε στο MATLAB την εντολή >>format rat ώστε να μετατρέψουμε τους αριθμούς σε ρητούς θα έχουμε:

$$x = 10025.01562 = \frac{641601}{64}$$

$$y = 10024.81538 = \frac{110273}{11}$$

$$\sqrt{x} - \sqrt{y} = \frac{22}{22001} = 9.999550559882664 \cdot 10^{-4}$$

Δηλαδή δεν κερδίσαμε τίποτα σε ακρίβεια. Επίσης είδαμε ότι το MATLAB δεν κατάφερε να κάνει με ακρίβεια πράξεις απλών ρητών, όπως ο υπολογισμός του αριθμού y ανωτέρω, κάνοντας μάλιστα ένα απόλυτο σφάλμα:

$$\left|y^{(mat)} - y\right| = \left|\frac{110273}{11} - \frac{626550961}{62500}\right| = \frac{1929}{687500} = 0.002805818182$$

Σε αυτό το σημείο ας συγκρίνουμε το MATLAB με ένα πρόγραμμα που είναι 'κλώνος' του και λέγεται Octave[7]. Κάνοντας τις ίδιες πράξεις το αποτέλεσμα είναι πιο ακριβές, δεδομένου ότι το σφάλμα είναι τρεις τάξεις μεγέθους μικρότερο:

$$\left|y^{(oct)} - y\right| = \left|\frac{651613}{65} - \frac{626550961}{62500}\right| = \frac{7}{812500} = 0.000008615385$$

Πρόκειται άραγε για κάποιο software **bug**, δηλαδή για σφάλμα ή αστοχία του πηγαίου κώδικα του MATLAB ή έχουμε κάποια αστοχία του προτύπου *IEEE 754-2008 / binary64 ;* Για να απαντήσουμε σε αυτό το ερώτημα θα πάμε ξανά στο διαδικτυακό εργαλείο μετατροπής δεκαδικών σε δυαδικούς αριθμούς και θα δούμε ότι:

$$100.125 = 64 + 32 + 4 + \tfrac{125}{1000} = \tfrac{801}{8} = 2^6 + 2^5 + 2^2 + \tfrac{1}{8} = 2^6 + 2^5 + 2^2 + 2^{-3}$$
$$100.125 = (1100100.0010000000000000000000000000000000000000000000)_2$$
$$100.124 = (1100100.0001111110111110011101101100100010110100001110)_2$$

Εάν πάρουμε τώρα αυτήν την δυαδική αναπαράσταση, κάνουμε τις πράξεις για όλες τις δυνάμεις του 2 που έχουν συντελεστή 1 και αθροίσουμε βλέπουμε ότι οι αριθμοί έχουν την ρητή προσέγγιση:

$$100.125 = \tfrac{801}{8}$$
$$100.124 = \tfrac{3522800071022215}{35184372088832}$$

[7]Ελεύθερο λογισμικό το οποίο μπορείτε να 'κατεβάσετε' από εδώ με κλικ και 'αποθήκευση'.

Το δεύτερο αποτέλεσμα σε οποιαδήποτε εφαρμογή ή πρόγραμμα και αν το υπολογίσουμε θα δώσει ακριβώς 100.124. Η διαφορά των ρητών προσεγγίσεων είναι για τα δύο προγράμματα:

$$\begin{aligned}\left|(100.125)^{(mat)} - (100.125)\right| &= \left|\frac{12115}{121} - \frac{3522800071022215}{35184372088832}\right| \\ &= \frac{140737488335}{4257309022748672} = .00003305785123 \\ \left|(100.124)^{(oct)} - (100.124)\right| &= \left|\frac{25031}{250} - \frac{3522800071022215}{35184372088832}\right| \\ &= \frac{21}{4398046511104000} = .4774847184 \cdot 10^{-14}\end{aligned}$$

Μπορούμε να κάνουμε έναν γενικότερο έλεγχο για τον τρόπο με τον οποίο το MATLAB ή το Octave μετατρέπουν απλούς αριθμούς σε ρητούς. Δημιουργούμε τους αριθμούς:

$$x_i = 100 + \frac{100 + i}{1000}, i = 1, 2, \ldots, 30 \tag{5}$$

Στον Πίνακα 1 φαίνονται οι αριθμοί x_i καθώς και οι αντίστοιχες προσεγγίσεις τους.

Αν υπολογίσουμε την μέση τιμή και την τυπική απόκλιση των λαθών $e_i^{(mat)} = x_i - x_i^{(mat)}$ και $e_i^{(oct)} = x_i - x_i^{(oct)}$ έχουμε ότι:

$$\begin{aligned}\overline{e}^{(mat)} &= -.1950433939 \cdot 10^{-5},\ \sigma^{(mat)} = 0.00004089444627 \\ \overline{e}^{(oct)} &= -.4256585548 \cdot 10^{-5},\ \sigma^{(oct)} = 0.00002144390164\end{aligned}$$

Εάν κάνουμε το ενδεικνυόμενο στατιστικό F-test βλέπουμε ότι απορρίπτεται η μηδενική υπόθεση της ισότητας των διακυμάνσεων σε επίπεδο στατιστικής σημαντικότητας 0.08%, επομένως πράγματι η διασπορά σφάλματος του MATLAB είναι μεγαλύτερη από την διασπορά σφάλματος του Octave.
Επίσης κάνοντας το στατιστικό t-test βλέπουμε ότι γίνεται δεκτή η μηδενική υπόθεση της ισότητας των μέσων τιμών (το σφάλμα τύπου α είναι 78.5%), επομένως πράγματι η μέση τιμή σφάλματος του MATLAB είναι ίση με την μέση τιμή σφάλματος του Octave.
Εάν αντί για τους αριθμούς x_i της 5 είχαμε επιλέξει τους αριθμούς:

$$y_i = 100 + \frac{96 + 2i}{2^{10}}, i = 0, 1, \ldots, 29 \tag{6}$$

οι οποίοι είναι άθροισμα δυνάμεων του 2, δηλ. αναπαρίστανται ακριβώς στο δυαδικό σύστημα, άρα δεν υπάρχει σφάλμα[8] στο πρότυπο *IEEE 754-2008 / binary64* , θα έπρεπε να έχουμε μηδενικό σφάλμα στα προγράμματά μας, πράγμα που επιτυγχάνεται μόνον στο Octave, όπως φαίνεται στον Πίνακα 2.

[8] Μπορείτε να εξηγήσετε γιατί ;

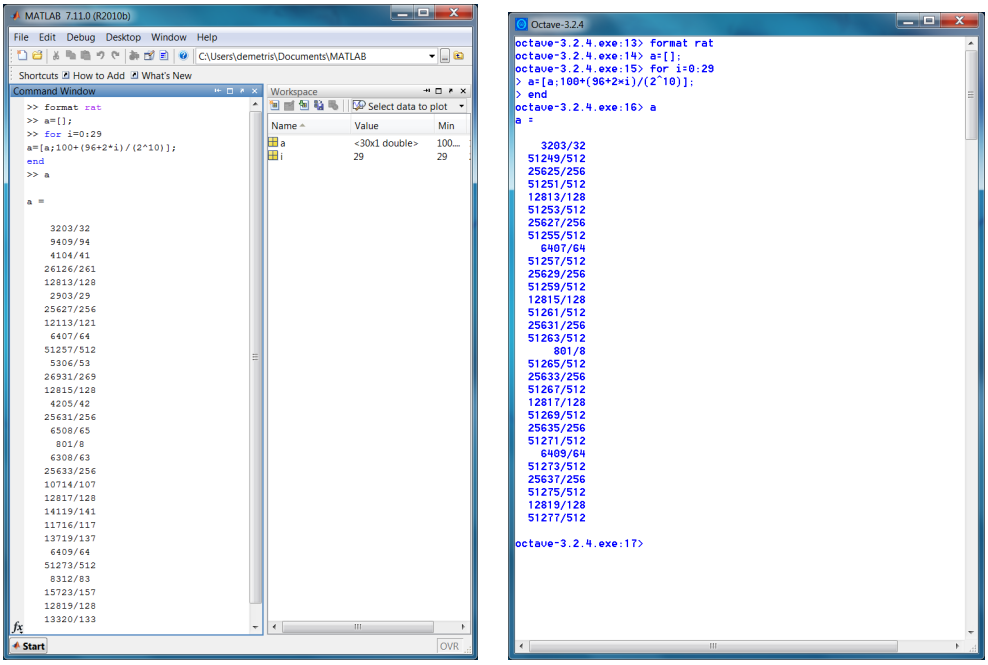
Πίνακας 1: Η αναπαράσταση των ρητών αριθμών $x_i = 100 + \frac{100+i}{1000}, i = 1, \ldots, 30$

x_i	$x_i^{(mat)}$	$x_i^{(oct)}$	$x_i - x_i^{(mat)}$	$x_i - x_i^{(oct)}$
$\frac{100101}{1000}$	$\frac{9910}{99}$	$\frac{9910}{99}$	$-\frac{1}{99000}$	$-\frac{1}{99000}$
$\frac{50051}{500}$	$\frac{4905}{49}$	$\frac{50051}{500}$	$-\frac{1}{24500}$	0
$\frac{100103}{1000}$	$\frac{6807}{68}$	$\frac{23324}{233}$	$\frac{1}{17000}$	$-\frac{1}{233000}$
$\frac{12513}{125}$	$\frac{12513}{125}$	$\frac{12513}{125}$	0	0
$\frac{20021}{200}$	$\frac{20021}{200}$	$\frac{20021}{200}$	0	0
$\frac{50053}{500}$	$\frac{6607}{66}$	$\frac{50053}{500}$	$-\frac{1}{16500}$	0
$\frac{100107}{1000}$	$\frac{24326}{243}$	$\frac{24326}{243}$	$\frac{1}{243000}$	$\frac{1}{243000}$
$\frac{25027}{250}$	$\frac{25027}{250}$	$\frac{25027}{250}$	0	0
$\frac{100109}{1000}$	$\frac{5506}{55}$	$\frac{21123}{211}$	$-\frac{1}{11000}$	$-\frac{1}{211000}$
$\frac{10011}{100}$	$\frac{10011}{100}$	$\frac{10011}{100}$	0	0
$\frac{100111}{1000}$	$\frac{100111}{1000}$	$\frac{901}{9}$	0	$-\frac{1}{9000}$
$\frac{12514}{125}$	$\frac{12514}{125}$	$\frac{12514}{125}$	0	0
$\frac{100113}{1000}$	$\frac{6207}{62}$	$\frac{17720}{177}$	$\frac{3}{31000}$	$\frac{1}{177000}$
$\frac{50057}{500}$	$\frac{11413}{114}$	$\frac{50057}{500}$	$-\frac{1}{28500}$	0
$\frac{20023}{200}$	$\frac{8710}{87}$	$\frac{20023}{200}$	$\frac{1}{17400}$	0
$\frac{25029}{250}$	$\frac{6908}{69}$	$\frac{25029}{250}$	$\frac{1}{17250}$	0
$\frac{100117}{1000}$	$\frac{9411}{94}$	$\frac{54764}{547}$	$-\frac{1}{47000}$	$-\frac{1}{547000}$
$\frac{50059}{500}$	$\frac{16119}{161}$	$\frac{50059}{500}$	$-\frac{1}{80500}$	0
$\frac{100119}{1000}$	$\frac{4205}{42}$	$\frac{47957}{479}$	$-\frac{1}{21000}$	$\frac{1}{479000}$
$\frac{2503}{25}$	$\frac{2503}{25}$	$\frac{2503}{25}$	0	0
$\frac{100121}{1000}$	$\frac{15719}{157}$	$\frac{28134}{281}$	$\frac{-3}{157000}$	$\frac{1}{281000}$
$\frac{50061}{500}$	$\frac{4105}{41}$	$\frac{50061}{500}$	$\frac{1}{20500}$	0
$\frac{100123}{1000}$	$\frac{6508}{65}$	$\frac{18723}{187}$	$-\frac{1}{13000}$	$\frac{1}{187000}$
$\frac{25031}{250}$	$\frac{12115}{121}$	$\frac{25031}{250}$	$\frac{1}{30250}$	0
$\frac{801}{8}$	$\frac{801}{8}$	$\frac{801}{8}$	0	0
$\frac{50063}{500}$	$\frac{12716}{127}$	$\frac{50063}{500}$	$\frac{1}{63500}$	0
$\frac{100127}{1000}$	$\frac{6308}{63}$	$\frac{6308}{63}$	$\frac{1}{63000}$	$\frac{1}{63000}$
$\frac{12516}{125}$	$\frac{12516}{125}$	$\frac{12516}{125}$	0	0
$\frac{100129}{1000}$	$\frac{3104}{31}$	$\frac{3104}{31}$	$-\frac{1}{31000}$	$-\frac{1}{31000}$
$\frac{10013}{100}$	$\frac{10013}{100}$	$\frac{10013}{100}$	0	0

Πίνακας 2: Η αναπαράσταση των ρητών αριθμών $y_i = 100 + \frac{96+2i}{2^{10}}, i = 0, \ldots, 29$

y_i	$y_i^{(mat)}$	$y_i^{(oct)}$	$y_i - y_i^{(mat)}$	$y_i - y_i^{(oct)}$
$\frac{3203}{32}$	$\frac{3203}{32}$	$\frac{3203}{32}$	0	0
$\frac{51249}{512}$	$\frac{9409}{94}$	$\frac{51249}{512}$	$-\frac{1}{24064}$	0
$\frac{25625}{256}$	$\frac{4104}{41}$	$\frac{25625}{256}$	$\frac{1}{10496}$	0
$\frac{51251}{512}$	$\frac{26126}{261}$	$\frac{51251}{512}$	$-\frac{1}{133632}$	0
$\frac{12813}{128}$	$\frac{12813}{128}$	$\frac{12813}{128}$	0	0
$\frac{51253}{512}$	$\frac{2903}{29}$	$\frac{51253}{512}$	$\frac{1}{14848}$	0
$\frac{25627}{256}$	$\frac{25627}{256}$	$\frac{25627}{256}$	0	0
$\frac{51255}{512}$	$\frac{12113}{121}$	$\frac{51255}{512}$	$-\frac{1}{61952}$	0
$\frac{6407}{64}$	$\frac{6407}{64}$	$\frac{6407}{64}$	0	0
$\frac{51257}{512}$	$\frac{51257}{512}$	$\frac{51257}{512}$	0	0
$\frac{25629}{256}$	$\frac{5306}{53}$	$\frac{25629}{256}$	$\frac{1}{13568}$	0
$\frac{51259}{512}$	$\frac{26931}{269}$	$\frac{51259}{512}$	$-\frac{1}{137728}$	0
$\frac{12815}{128}$	$\frac{12815}{128}$	$\frac{12815}{128}$	0	0
$\frac{51261}{512}$	$\frac{4205}{42}$	$\frac{51261}{512}$	$\frac{1}{10752}$	0
$\frac{25631}{256}$	$\frac{25631}{256}$	$\frac{25631}{256}$	0	0
$\frac{51263}{512}$	$\frac{6508}{65}$	$\frac{51263}{512}$	$-\frac{1}{33280}$	0
$\frac{801}{8}$	$\frac{801}{8}$	$\frac{801}{8}$	0	0
$\frac{51265}{512}$	$\frac{6308}{63}$	$\frac{51265}{512}$	$-\frac{1}{32256}$	0
$\frac{25633}{256}$	$\frac{25633}{256}$	$\frac{25633}{256}$	0	0
$\frac{51267}{512}$	$\frac{10714}{107}$	$\frac{51267}{512}$	$\frac{1}{54784}$	0
$\frac{12817}{128}$	$\frac{12817}{128}$	$\frac{12817}{128}$	0	0
$\frac{51269}{512}$	$\frac{14119}{141}$	$\frac{51269}{512}$	$\frac{1}{72192}$	0
$\frac{25635}{256}$	$\frac{11716}{117}$	$\frac{25635}{256}$	$-\frac{1}{29952}$	0
$\frac{51271}{512}$	$\frac{13719}{137}$	$\frac{51271}{512}$	$-\frac{1}{70144}$	0
$\frac{6409}{64}$	$\frac{6409}{64}$	$\frac{6409}{64}$	0	0
$\frac{51273}{512}$	$\frac{51273}{512}$	$\frac{51273}{512}$	0	0
$\frac{25637}{256}$	$\frac{8312}{83}$	$\frac{25637}{256}$	$-\frac{1}{21248}$	0
$\frac{51275}{512}$	$\frac{15723}{157}$	$\frac{51275}{512}$	$-\frac{1}{80384}$	0
$\frac{12819}{128}$	$\frac{12819}{128}$	$\frac{12819}{128}$	0	0
$\frac{51277}{512}$	$\frac{13320}{133}$	$\frac{51277}{512}$	$\frac{1}{68096}$	0

Καταλήγουμε λοιπόν στο συμπέρασμα ότι ο τρόπος που χρησιμοποιεί εσωτερικά το MATLAB για να μετατρέψει έναν δεκαδικό στον πλησιέστερο ρητό δεν είναι ακριβής, διότι αδυνατεί να παραστήσει ρητούς που εκφράζονται ακριβώς στο πρότυπο *IEEE 754-2008 / binary64* με απόλυτη ακρίβεια στο δικό του περιβάλλον εργασίας. Το Octave από την άλλη πλευρά, μολονότι δεν είναι πάντα ακριβές στους ρητούς, εντούτοις αναπαριστά πάντα τους ακριβείς ρητούς αριθμούς μηχανής με απόλυτη ακρίβεια στο περιβάλλον εργασίας του.
Βέβαια, το τελικό αποτέλεσμα δεν είναι αρκετά διαφορετικό, αλλά πρέπει πάντα να είμαστε προσεκτικοί στο μέγεθος του σφάλματος στρογγύλευσης στα ενδιάμεσα στάδια, διότι το φαινόμενο της *απόσβεσης σφάλματος* (error damping) δεν είμαστε σίγουροι ότι έχει πραγματοποιηθεί στην ακολουθία πράξεων μηχανής που εξετάζουμε κάθε φορά. Η σύγκριση των αποτελεσμάτων των δύο προγραμμάτων φαίνεται στο 1.



(α′) MATLAB (β′) Octave

Σχήμα 1: Σύγκριση απλών πράξεων ρητών σε MATLAB και Octave.

Εκτός από την ανωτέρω περίπτωση η οποία ουσιαστικά αναφέρεται σε αρνητικές δυνάμεις του 2, μπορείτε να βείτε κάποια άλλη περίπτωση για την οποία να υπάρχει πλήρης αναπαράσταση από το Octave αλλά όχι από το MATLAB; Εάν βρείτε, τότε μην διστάσετε να επικοινωνήσετε με τον γράφοντα[9], ώστε να υπάρξει πληρέστερη

[9] Δημήτριος Θ. Χριστόπουλος, dchristop@econ.uoa.gr

διερεύνηση του θέματος.

1.2.3 Η διάδοση (propagation) του σφάλματος στρογγύλευσης

Ενδιαφερόμαστε να εξετάσουμε πως 'μεταδίδεται' το σφάλμα στρογγύλευσης των αριθμών που συμμετέχουν σε μία σειρά από πράξεις μηχανής. Είδαμε ότι δεν ισχύει η προσεταιριστική ιδιότητα στην πρόσθεση αριθμών μηχανής. Παρακάτω θα δούμε αναλυτικά πως μπορεί να εξηγηθεί και αυτό το φαινόμενο. Εάν έχουμε μία συνάρτηση ν μεταβλητών $f(x_1, x_2, \ldots, x_\nu)$ τότε γνωρίζουμε ότι η μεταβολή της μπορεί προσεγγιστικά να παρασταθεί ως:

$$\Delta f \approx \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 + \ldots + \frac{\partial f}{\partial x_\nu} dx_\nu \tag{7}$$

Κάθε παράσταση που πρέπει να υπολογιστεί με αριθμούς μηχανής την θεωρούμε ως συνάρτηση πολλών μεταβλητών και ορίζουμε σαν απόλυτο σφάλμα Δf την διαφορά $fl(f) - f$ ή $|fl(f) - f|$.
Ως σχετικό σφάλμα ϵ_f ορίζουμε το πηλίκο $\frac{fl(f)-f}{f}$ ή $\left|\frac{fl(f)-f}{f}\right|$, συνήθως εκφρασμένο ως ποσοστό %.
Επίσης μπορούμε να αποδείξουμε, αλλά δεν είναι σκόπιμο να το κάνουμε εδώ [10], ότι για το σχετικό σφάλμα της ανωτέρω συνάρτησης έχουμε ότι:

$$\epsilon_f \approx \sum_{i=1}^{\nu} \frac{x_i}{f} \frac{\partial f}{\partial x_i} \epsilon_{x_i} = \frac{x_1}{f} \frac{\partial f}{\partial x_1} \epsilon_{x_1} + \frac{x_2}{f} \frac{\partial f}{\partial x_2} \epsilon_{x_2} + \ldots + \frac{x_\nu}{f} \frac{\partial f}{\partial x_\nu} \epsilon_{x_\nu} \tag{8}$$

Παράδειγμα 1.3. *Σαν γενικό παράδειγμα θεωρούμε τον υπολογισμό των ριζών της δευτεροβάθμιας εξίσωσης :*

$$\frac{1}{2}x^2 + \beta x - \frac{1}{2}\gamma = 0 \tag{9}$$

Οι ρίζες της 9 είναι:

$$\begin{aligned} \rho_1 &= -\beta + \sqrt{\beta^2 + \gamma} \\ \rho_2 &= -\beta - \sqrt{\beta^2 + \gamma} \end{aligned} \tag{10}$$

Θεωρώντας την συνάρτηση:

$$f(\alpha, \beta) = -\beta + \sqrt{\beta^2 + \gamma} \tag{11}$$

[10] Ο παρατηρητικός αναγνώστης ας προσέξει ότι ο τύπος που γράψαμε δεν είναι τίποτα άλλο παρά ένα άθροισμα ελαστικοτήτων ως προς όλες τις μεταβλητές.

Υπολογίζουμε το απόλυτο σφάλμα για την ρίζα ρ_1:

$$\Delta f = \left(-1 + \frac{\beta}{\sqrt{\beta^2+\gamma}}\right) d\beta + \left(\frac{1}{2\sqrt{\beta^2+\gamma}}\right) d\gamma \tag{12}$$

Από την μορφή του σφάλματος 12 παρατηρούμε ότι θα έχουμε σίγουρα πρόβλημα όταν ισχύει $\gamma \approx -\beta^2$. Υπολογίζουμε και το σχετικό σφάλμα της αριθμητικής προσέγγισης για την ρίζα ρ_1:

$$\begin{aligned}\epsilon_f &= \left(\frac{-\beta}{\sqrt{\beta^2+\gamma}}\right)\epsilon_\beta + \left(\frac{\gamma}{2\left(-\beta+\sqrt{\beta^2+\gamma}\right)\sqrt{\beta^2+\gamma}}\right)\epsilon_\gamma \\ &= \frac{-\beta}{\sqrt{\beta^2+\gamma}}\epsilon_\beta + \frac{\beta+\sqrt{\beta^2+\gamma}}{2\sqrt{\beta^2+\gamma}}\epsilon_\gamma\end{aligned} \tag{13}$$

λόγω και της 11. Επομένως πάλι έχουμε πρόβλημα ακρίβειας όταν $\gamma \approx -\beta^2$.

Παράδειγμα 1.4. *Να μελετηθεί η πράξη $f(x,y) = x+y+z$ ως προς την διάδοση του σχετικού σφάλματος των x, y, z.*
Έχουμε ότι $f(x,y,z) = x+y+z$ και το σχετικό σφάλμα διάδοσης υπολογίζεται εύκολα:

$$\epsilon_f = \frac{x}{x+y+z}\epsilon_x + \frac{y}{x+y+z}\epsilon_y + \frac{z}{x+y+z}\epsilon_z \tag{14}$$

Από την μορφή 14 βλέπουμε ότι κάθε φορά που ένας προσθετέος είναι μεγάλος σχετικά με το συνολικό άθροισμα, τότε το σφάλμα του αντίστοιχου προσθετέου μεγενθύνεται στο τελικό σφάλμα. Γι αυτό το λόγο πρέπει να αποφεύγουμε να προσθέτουμε πολύ μεγάλους με πολύ μικρούς αριθμούς.
Λόγω αυτού του γεγονότος ερμηνεύεται και το μεγάλο σφάλμα που είδαμε να κάνει το MATLAB στον υπολογισμό του ρητού $y = \left(100 + \frac{124}{1000}\right)^2$ στην προηγούμενη υπο-ενότητα 1.2.1.
Επίσης τώρα μπορούμε να ερμηνεύσουμε και την μη ύπαρξη της προσεταιριστικότητας στην πρόσθεση αριθμών μηχανής. Εάν $x+y >> z$ και ο όρος $x+y$ έχει αυτοτελώς μεγάλο σφάλμα στρογγύλευσης, τότε το σχετικό αφάλμα διάδοσης στο σφάλμα της πράξης $(x+y)+z$ θα είναι $\frac{x+y}{x+y+z}\epsilon_{x+y} + \frac{z}{x+y+z}\epsilon_z$ και μπορεί να είναι μεγαλύτερο από το $\frac{x}{x+y+z}\epsilon_x + \frac{y+z}{x+y+z}\epsilon_{y+z}$ που αντιστοιχεί στην πράξη $x+(y+z)$.
Το άλλο συμπέρασμα που προκύπτει είναι ότι πρέπει παντα να κάνουμε την πρόσθεση του μικρότερου με τον μεγαλύτερο αριθμό κι όχι το αντίστροφο, γιατί τότε το τελικό σφάλμα είναι μικρότερο. Σε ένα άθροισμα πολλών προσθετέων λοιπόν, ο αλγόριθμος που θα πρέπει να αναπτύξουμε θα είναι τέτοιος ώστε να κάνει τις πρόσθέσεις σε αύξουσα σειρά των προσθετέων και όχι σε φθίνουσα σειρά.

Ανακεφαλαιώνοντας παραθέτουμε συγκεντρωτικά τους κανόνες που πρέπει να ακολουθούνται ώστε να να υπάρχουν οι μικρότερες δυνατές απώλειες ακρίβειας.

Γενικοί Κανόνες χειρισμού αριθμών μηχανής.

1. Χρησιμοποιούμε αλγεβρικές ταυτότητες, τεχνάσματα καθώς και τριγωνομετρικές ταυτότητες για να μετατρέψουμε μία αφαίρεση σε συνδυασμό άλλων πράξεων.

2. Αποφεύγουμε να προσθέτουμε έναν μεγάλο με έναν μικρό αριθμό.

3. Όταν έχουμε να υπολογίσουμε ένα άθροισμα, αναδιατάσσουμε τους όρους έτσι ώστε η πρόσθεση να γίνεται με αυξανόμενους προσθετέους.

4. Όταν θέλουμε να λύσουμε ένα πρόβλημα σε ένα πλέγμα αριθμών, φροντίζουμε οι αριθμοί αυτοί να είναι παραστάσιμοι με απόλυτη ακρίβεια στο δυαδικό σύστημα.

5. Όταν σχεδιάζουμε έναν αλγόριθμο λαμβάνουμε υπόψιν μας όλες τις ανωτέρω παρατηρήσεις.

Γενικότερα αν θεωρήσουμε τις πράξεις μηχανής $f(x, y)$ που ακολουθούν και υπολογίσουμε το σχετικό σφάλμα, θα έχουμε τον Πίνακα 3 από τον οποίο βλέπουμε ότι η μόνη πράξη που πρακτικά μπορεί να καταστρέψει την συνολική ακρίβεια ενός αλγορίθμου πράξεων μηχανής είναι η πρόσθεση. Εάν πάρουμε απόλυτες τιμές, τότε για την προσθεση έχουμε ότι:

$$|\epsilon_f| \leq \left|\frac{x}{x+y}\right| |\epsilon_x| + \left|\frac{y}{x+y}\right| |\epsilon_y|$$

Εάν ο ένας προσθετέος είναι μικρός μεν σε σύγκριση με τον άλλο, έχει μεγάλο σχετικό σφάλμα όμως, τότε είναι δυνατόν να έχουμε απαλοιφή σφάλματος error damping, στο συνολικό αποτέλεσμα. Εάν όμως προσθέτουμε αριθμούς με διαφορετικό πρόσημο, δηλ. κάνουμε αφαίρεση, τότε υπάρχει μεγάλη πιθανότητα ο όρος $x + y$ στους παρονομαστές να γίνει πολύ μεγάλος κι έτσι να υπάρξει σημναντική απώλεια ακρίβειας σε μία και μόνον πράξη.

1.3 Η Συμβολική Υπολογιστική Άλγεβρα

Το 1954 εμφανίστηκε από την IBM[11] η αρχαιότερη γλώσσα προγραμματισμού, η **FOR**mula**TRAN**slation η οποία εξακολουθεί να υποστηρίζεται ακόμη. Έναν χρόνο μετά άρχισε να δημιουργείται στα πλαίσια ενός προγράμματος υλοποίησης παραγώγισης συναρτήσεων μέσω υπολογιστή η όχι και τόσο γνωστή γλώσσα προγραμματισμού **LIS**t**P**rocessing (LISP). Η υλοποίηση της πρώτης έκδοσης ξεκίνησε το

[11]International Business Machines, Armonk, New York, USA.

$f(x,y)$	ϵ_f
$x \cdot y$	$\epsilon_x + \epsilon_y$
x/y	$\epsilon_x - \epsilon_y$
$\sqrt{x}$	$\frac{1}{2}\epsilon_x$
$x \pm y$	$\frac{x}{x\pm y}\epsilon_x \pm \frac{y}{x\pm y}\epsilon_y \ (x \pm y \neq 0)$

Πίνακας 3: Σχετικό σφάλμα διάδοσης βασικών πράξεων αριθμών μηχανής

1958 μέσα από το πρόγραμμα Τεχνητής Νοημοσύνης του Πανεπιστημίου M.I.T.[12]. Πρόκειται βασικά για μία γλώσσα που χειρίζεται *Δομές Λιστών* με περιεχόμενο είτε αριθμητικό είτε αλφαβητικό, η οποία μπορεί να ορίζει και να διαχειρίζεται *αναδρομικές συναρτήσεις*.

Ένα παράδειγμα, το οποίο βρίσκεται εδώ, με τον κώδικα (αριστερά) και το αποτέλεσμα (δεξιά) είναι αυτό που ακολουθεί:

$$\text{(PLUS X (TIMES 3 Y) Z)} \rightarrow x + 3y + z$$

Ένα από τα πρώτα προγράμματα που προέκυψαν από την εφαρμογή αυτής της γλώσσας ήταν το Reduce το 1970, το οποίο από το 2008 είναι ελεύθερο λογισμικό. Το 1971 ξεκίνησε την σταδιοδρομία του και το Axiom, το οποίο έγινε ελεύθερο λογισμικό το 2001. Πάλι από το M.I.T. στα τέλη της δεκαετίας του ΄60 έχουμε την εμφάνιση του *DOE Macsyma*, το οποίο είναι ο κοινός πρόγονος των Mathimatica και Maple. Μετά το 1998 οδήγησε στο ελεύθερο λογισμικό Maxima, το οποίο υποστηρίζεται πλέον ως wxMaxima. Ακόμη πιο αναλυτική παρουσίαση για την γλώσσα LISP μπορεί κανείς να βρει εδώ.

Πρέπει να τονίσουμε ότι μολονότι όλα τα εμπορικά Συστήματα Υπολογιστικής Άλγεβρας (**Computer Algebra Systems - CAS**) έχουν την ρίζα τους σε εφαρμογές που αναπτύχθηκαν σε LISP, εντούτοις σήμερα είναι γραμμένα κατά το μεγαλύτερο μέρος τους σε γλώσσα προγραμματισμού C,C++. Για παράδειγμα το Mathimatica, όπως αναφέρεται εδώ, είναι γραμμένο περίπου κατά 50% από C,C++,LISP, (γλώσσα Mathimatica + αλγεβρικοί υπολογισμοί), ενώ το υπόλοιπο 50% είναι το λεγόμενο περιβάλλον Γραφικής Αλληλεπίδρασης Χρήστη (Graphical User Interface - GUI), το οποίο είναι γραμμένο σε γλώσσα Mathimatica και σε JAVA[13]. Το Maple είναι χωρισμένο σε δύο περιβάλλοντα, το λεγόμενο κλασσικό φύλλο εργασίας, το οποίο 'τρέχει' σε C και το τυπικό φύλλο εργασίας το οποίο εργάζεται με JAVA.

Το MATLAB μέχρι το 2008 είχε ενσωματωμένο πυρήνα (kernel) του Maple για να υποστηρίζει το λεγόμενο 'Κουτί Εργαλείων Συμβολικών' υπολογισμών, δηλαδή το "Symbolics Toolbox", αλλά έκτοτε αγόρασε το MuPad, απορρόφησε την εταιρεία

[12]Massachusetts Institute of Technology, Cambridge, Massachusetts, U.S.A..

[13]Σήμα που ανήκει σήμερα στην Oracle Corporation , Redwood Shores, California, USA

που το παρήγαγε και τώρα πλέον οι συμβολικοί υπολογισμοί στο MATLAB γίνονται με το MuPad.

Εκτός από τη δυνατότητα συμβολικών πράξεων όπως $a + a = 2a$, όλα τα προγράμματα CAS έχουν όπως ήδη παρατηρήσαμε την λεγόμενη *αριθμητική οιασδήποτε ακρίβειας* (arbritrary - precision arithmitic) ή *άπειρης ακρίβειας αριθμητική* (infinite - precision arithmetic), δηλ. μπορούμε να κάνουμε πράξεις με οιαδήποτε ακρίβεια δεκαδικών ψηφίων επιθυμούμε, αρκεί να διαθέτουμε επαρκή υπολογιστική ισχύ και χρόνο. Σημαντικές παρατηρήσεις:

- Όλα τα προγράμματα CAS αποφεύγουν να μετατρέπουν άμεσα άρρητους αριθμούς, όπως λ.χ. $\pi, e, \sqrt{2}$ ή μη ρητές συναρτήσεις, π.χ. $sin(1), e^{-1}, \ln(2)$ σε αριθμούς κινητής υποδιαστολής, αλλά δίνουν το αποτέλεσμα σαν συνάρτηση αυτών.

- Επαφίεται κατόπιν στον χρήστη να ζητήσει δεκαδική αναπαράσταση με όσα δεκαδικά ψηφία επιθυμεί εκείνος.

Για παράδειγμα αν γράψουμε στο Mathimatica:

```
Sin[2*Pi*Sqrt[3]] + Exp[-5]*Log[7]
```

αυτό θα δώσει το ακριβές αλγεβρικό αποτέλεσμα:

$$\frac{Log[7]}{e^5} + Sin\left[2\sqrt{3}\pi\right]$$

Αν όμως γράψουμε:

```
Sin[2*Pi*Sqrt[3]] + Exp[-5]*Log[7] // N
```

αυτό θα δώσει το προσεγγιστικό αριθμητικό αποτέλεσμα:

```
-0.980536
```

Υπάρχουν δύο μεγάλες οικογένειες πακέτων λογισμικού τα οποία κάνουν συμβολικές πράξεις:

- Τα λεγόμενα εμπορικά πακέτα, τα οποία πωλούνται ως προϊόντα λογισμικού και συνήθως διαθέτουν και μία φοιτητική έκδοση με μειωμένη τιμή. Συνήθως δεν επιδέχονται αλλαγές στον πηγαίο κώδικά τους. Μερική δυνατότητα επέμβασης υπάρχει στο Maple. Σε αυτήν την κατηγορία ξεχωρίζουν τα προγράμματα Mathimatica και Maple.

- Τα λεγόμενα 'ελεύθερα προγράμματα' ή αλλιώς 'λογισμικό ανοιχτού κώδικα', τα οποία διατίθενται χωρίς χρέωση μέσω διαδικτύου και επιδέχονται αλλαγές στον πηγαίο κώδικά τους. Σε αυτήν την κατηγορία ξεχωρίζουν το wxMaxima (το οποίο υποστηρίζει και ελληνικό περιβάλλον εργασίας) και το Sage[14]. Το τελευταίο πολλές φορές ανταγωνίζεται σε ταχύτητα τα εμπορικά προγράμματα, αλλά είναι κυρίως γραμμένο για το λειτουργικό σύστημα Linux[15] και δεν είναι τόσο φιλικό με τα Windows[16]

Περισσότερες λεπτομέρειες για τα Συστήματα Υπολογιστικής Άλγεβρας (CAS) καθώς επίσης και μία σύγκριση μεταξύ τους μπορείτε να βρείτε εδώ. Τελειώνοντας την συνοπτική παρουσίαση των (CAS) αναφέρουμε τα βασικά πλεονεκτήματά τους έναντι των παραδοσιακών πορογραμμάτων αριθμητικής κινητής υποδιαστολής.

Πλεονεκτήματα των Συστημάτων Υπολογιστικής Άλγεβρας (CAS)

1. Είναι ικανά να κάνουν πράξεις ρητών αριθμών με απόλυτη ακρίβεια.

2. Υπάρχει η δυνατότητα επιλογής της επιθυμητής ακρίβειας δεκαδικών ψηφίων στους άρρητους αριθμούς, π.χ. το $\pi = 3.1415926253\ldots$ με όσα δεκαδικά ψηφία επιθυμούμε.

3. Δυνατότητα συμβολικών πράξεων,π.χ. $x + x = 2x, \int x dx = \frac{x^2}{2}$.

4. Μπορούν να λύσουν ακριβώς σύνθετα μαθηματικά προβλήματα, π.χ. Διαφορικές Εξισώσεις.

Το μεγάλο πλεονέκτημα βέβαια των προγραμμάτων αριθμητικής κινητής υποδιαστολής όπως το MATLAB είναι η ταχύτητα επεξεργασίας μεγάλου μεγέθους αριθμητικών πινάκων, κάτι που είναι ιδιαίτερα χρήσιμο στην επεξεργασία εικόνας, ήχου και οπουδήποτε θέλουμε να επεξεργαστούμε μεγάλο όγκο δεδομένων. Αυτός είναι και ο λόγος που ακόμα και τα (CAS) όπως το Maple λ.χ. έχουν αναπτύξει περιβάλλον διασύνδεσης με το MATLAB για την ταχύτατη υλοποίηση μετασχηματισμών όπως ο Ταχύς Μετασχηματισμός Fourier - (F.F.T.).

Τέλος, ένα από τα δυνατά σημεία του πλήρους πακέτου MATLAB είναι η ικανότητα διασύνδεσης του προγράμματος με κάθε είδους περιφερειακά: μικρόφωνα, κάμερες, ηλεκτρονικά μικροσκόπια και κάθε είδους ηλεκτρονικά επιστημονικά όργανα. Με αυτόν τον τρόπο και την χρήση του κατάλληλου "Toolbox" είναι δυνατή η εισαγωγή, επεξεργασία και μοντελοποίηση ταυτόχρονα με ένα πρόγραμμα.

[14] Ελεύθερο λογισμικό, μπορείτε να το 'κατεβάσετε' από εδώ.
[15] Σήμα κατατεθέν του Φινλανδού Linus Torvalds.
[16] Σήμα κατατεθέν της Microsoft Corporation, Redmond, Washington, USA.

1.4 Ασκήσεις

1. Εισάγετε στο MATLAB έχοντας format rat τους ρητούς αριθμούς:

$$p = 100 + \frac{24}{1000} = 100.024, p = 100 + \frac{23}{1000} = 100.023$$

και κάνετε τις πράξεις $p+q, p-q, pq, \frac{p}{q}$. Υπολογίστε τα απόλυτα σφάλματα των άνω πράξεων, έχοντας κάνει με το χέρι ή με ένα CAS τις πράξεις με απόλυτη ακρίβεια. Τι παρατηρείτε; Πως εξηγείτε το γεγονός ότι το MATLAB αναπαρέστησε σωστά μόνον τον p;.

2. Γνωρίζουμε ότι η παρακάτω σειρά MacLaurin για την εκθετική συνάρτηση:

$$e^{-x} = 1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + \frac{1}{24}x^4 - \frac{1}{120}x^5 + \frac{1}{720}x^6 - \frac{1}{5040}x^7 + \ldots$$

συγκλίνει στην πραγματική τιμή του e^{-x} για κάθε πραγματικό αριθμό x και θέλουμε να υπολογίσουμε τον αριθμό e^{-5} παίρνοντας τους ανωτέρω 8 πρώτους όρους της. Να κάνετε τις πράξεις με το MATLAB και να συγκρίνετε το αποτέλεσμα με την τιμή $exp(-5)$ του ίδιου προγράμματος. Κατόπιν να κάνετε τις πράξεις για $x = 5$ στην ακόλουθη ισοδύναμη παράσταση:

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7 + \ldots}$$

Μπορείτε να εξηγήσετε γιατί τώρα το αποτέλεσμα είναι κοντά στο πραγματικό;

3. Χρησιμοποιώντας πρώτα fromat rat κι έπειτα fromat long e να περάσετε στο MATLAB τους παρακάτω ρητούς αριθμούς:

$$\alpha = 1063 + \frac{431}{1201}, \beta = 10 + \frac{225}{100000}, \gamma = 17 + \frac{333}{10007}$$

Να κάνετε τις πράξεις με απόλυτη ακρίβεια και να συγκρίνετε αυτό που βρήκατε με την έξοδο του MATLAB. Είστε ικανοποιημένοι από το αποτέλεσμα;

4. Χρησιμοποιώντας κατάλληλες αλγεβρικές ή τριγωνομετρικές ταυτότητες να βρείτε τρόπους υπλογισμού των παρακάτω παραστάσεων στο MATLAB ώστε να έχουμε την μικρότερη δυνατή απώλεια ακρίβειας:

   (α′) $sin(u+x) - sin(u), |x| << 1$

   (β′) $1 - cosx, |x| << 1$

   (γ′) $\ln(x) - \ln(y), x, y >> 1$

   (δ′) $e^{x-y}, x, y >> 1$

5. Θεωρήστε ότι στο τριώνυμο $x^2 - 2\beta \cdot x + \gamma$ ισχύει $\beta, \gamma > 0$ και $\beta^2 >> \gamma$. Αφού εξηγήσετε γιατί ο τύπος των ριζών τριωνύμου θα εμφανίσει προβλήματα ακρίβειας, να βρείτε έναν άλλο ισοδύναμο τύπο ο οποίος να μην παρουσιάζει τέτοια προβλήματα. Σαν εφαρμογή, να λύσετε το τριώνυμο με $\beta = 10^6, \gamma = 10$ χρησιμοποιώντας το MATLAB και με τους δύο τρόπους. Πότε παρατηρείτε μεγαλύτερη ακρίβεια;

Αναφορές

Ακρίβης, Γ. & Δουγαλής, Β. (2005), *Εισαγωγή στην αριθμητική ανάλυση, 2η έκδοση*, Πανεπιστημιακές Εκδόσεις Κρήτης.

Goldberg, D. (1991), 'What every computer scientist should know about floating-point arithmetic', *ACM Comput. Surv.* **23**, 5–48.
URL: `http://doi.acm.org/10.1145/103162.103163`

Hildebrand, J., Prausnitz, J. & Scott, R. (1970), *Regular and Related Solutions*, van Nostrand Reinhold Company, New York.

Stoer, J. & Bulirsch, R. (2002), *Introduction to Numerical Analysis*, Texts in Applied Mathematics; 12, third edn, Springer, New York.