Υπολογιστική Φυσική

Βασιλείου Σ. Γερογιάννη

Καθηγητή Τμήματος Φυσικής Πανεπιστημίου Πατρών

Πατρα 2016

Κεφάλαιο 1

Εισαγωγικά

1.1 Ακρίβεια πραγματικών και μιγαδικών σταθερών και μεταβλητών στην **Fortran**

Εκτός από την απλή και διπλή ακρίβεια ([1], Κεφ. 2, §9· επίσης [2], §4), η Fortran G95 [3] διαθέτει και την «υψηλή ακρίβεια» (high precision) με `KIND=10`. Αυτή εφαρμόζεται σε πραγματικές και μιγαδικές σταθερές και μεταβλητές εξασφαλίζοντας ακρίβεια ~ 20 δεκαδικών ψηφίων.

Ακολουθεί ένα πρόγραμμα, στο οποίο εμφανίζονται μεταβλητές διαφόρων τύπων και ειδών.

```
      PROGRAM TYPES_AND_KINDS
      IMPLICIT REAL(KIND=4) (A-C,T)
      IMPLICIT REAL(KIND=8) (D-H)
      IMPLICIT REAL(KIND=10) (O-S,U-W)
      IMPLICIT COMPLEX(KIND=10) (X-Z)
      INTEGER :: FLAG_ONE,FLAG_TWO,FORES
      REAL(KIND=4) :: DURATION
      REAL(KIND=8) :: STEP_ONE,STEP_TWO
      REAL(KIND=10) :: ELN,LENGTH,LINEAR_MOMENTUM,MASS
      COMPLEX(KIND=10) :: RADIUS_1,RADIUS_2,RADIUS_3
! Three dots indicate ommitted code inside a program unit
      ...
      T_START=1.2E1_4
      T_END=2.4E1_4
      DISTANCE=7.842376395554E3_8
! The mathematical constant pi
      PI=4*ATAN(1._10)
! The base e of the natural logarithms
      ELN=EXP(1._10)
      RADIUS_1=CMPLX(1.71_10,3.44_10,10)
      ...
      END PROGRAM TYPES_AND_KINDS
```

Προγράμματα όπως αυτό γράφονται (και τροποποιούνται) πιο εύκολα με τη βοήθεια ακεραίων «ονομασμένων σταθερών» (named constants) που ορίζονται με την εντολή `PARAMETER` ([1], Κεφ. 1, §5· επίσης [2], §4).

Ακολουθεί μία τροποποίηση του προηγούμενου προγράμματος, στην οποία γίνονται εύκολα αλλαγές σε τύπους και είδη.

```
      PROGRAM TYPES_AND_KINDS
      IMPLICIT REAL(KIND=4) (A-C,T)
```

```
IMPLICIT REAL(KIND=8) (D-H)
IMPLICIT REAL(KIND=10) (O-S,U-W)
IMPLICIT COMPLEX(KIND=10) (X-Z)
INTEGER,PARAMETER :: KA=4,KB=8,KD=10
INTEGER :: FLAG_ONE,FLAG_TWO,FORES
REAL(KIND=KA) :: DURATION
REAL(KIND=KB) :: STEP_ONE,STEP_TWO
REAL(KIND=KD) :: ELN,LENGTH,LINEAR_MOMENTUM,MASS
COMPLEX(KIND=KD) :: RADIUS_1,RADIUS_2,RADIUS_3
...
T_START=1.2E1_KA
T_END=2.4E1_KA
DISTANCE=7.842376395554E3_KB
PI=4*ATAN(1._KD)
ELN=EXP(1._KD)
RADIUS_1=CMPLX(1.71_KD,3.44_KD,KD)
...
END PROGRAM TYPES_AND_KINDS
```

Άν π.χ. χρειάζεται να μετατρέψουμε όλες τις μεταβλητές διπλής ακρίβειας σε υψηλής ακρίβειας, μαζί με τις αριθμητικές τιμές τους, πρέπει να αλλάξουμε την τιμή της ακέραιας ονομασμένης σταθεράς KB από '8' σε '10'. Πρέπει όμως να γίνει και μία αλλαγή 'με το χέρι', δηλαδή το είδος KIND=8 στη δεύτερη εντολή IMPLICIT πρέπει να γραφτεί KIND=10. Αυτό οφείλεται στο γεγονός ότι δεν επιτρέπεται να γράψουμε πριν από την εντολή (ή τις εντολές) IMPLICIT οποιαδήποτε άλλη εντολή δήλωσης τύπου, οπότε δεν είναι δυνατό να προτάξουμε την εντολή ορισμού του KD ώστε να εφαρμόζεται και στην (στις) IMPLICIT.

1.2 Υποπρογράμματα modules

Ένας τρόπος να απλοποιήσουμε ακόμη περισσότερο τη διαδικασία τροποποίησης, όπως προηγουμένως, είναι να χρησιμοποιήσουμε το τρίτο είδος υποπρογραμμάτων που διαθέτει η Fortran. Συγκεκριμένα, εκτός από τις υπορουτίνες (subroutines) και τις συναρτήσεις (functions) ([1], Κεφ. 4· επίσης [2], §11), υπάρχουν και τα υποπρογράμματα modules ([2], §21.6). Ένα υποπρόγραμμα module αρχίζει με την εντολή MODULE ακολουθούμενη από το όνομά του. Τελειώνει με την εντολή END ακολουθούμενη προαιρετικά από τη λέξη MODULE και το όνομά του.

Τα modules έχουν δύο βασικά χαρακτηριστικά:

1. Περιέχουν «μή εκτελέσιμες εντολές» (nonexecutable statements) και, ενδεχομένως, ένα ή περισσότερα «εσωτερικά υποπρογράμματα» (internal subprograms). Τα εσωτερικά υποπρογράμματα, εφόσον υπάρχουν, γράφονται μετά από την εντολή CONTAINS· κατά τα άλλα, ακολουθούν τους κανόνες των συνηθισμένων υποπρογραμμάτων. **Μεμονωμένες εκτελέσιμες εντολές δεν επιτρέπονται μέσα σε ένα module**.

2. Γράφονται πριν από το κύριο πρόγραμμα. Δηλαδή ένα «πλήρες πρόγραμμα Fortran», ή όπως αλλοιώς λέγεται «πακέτο Fortran», ή απλώς (χωρίς τους χαρακτηρισμούς «κύριο» και «Fortran») «πρόγραμμα» (complete Fortran program, Fortran package, program), το οποίο περιέχει modules, αρχίζει με αυτά, συνεχίζεται με το κύριο πρόγραμμα, και τελειώνει με τα υποπρογράμματα. Το κύριο πρόγραμμα ή ένα υποπρόγραμμα, που χρειάζεται κάποιο από τα modules, πρέπει να το 'καλέσει' όχι με την εντολή CALL, αλλά με την εντολή USE· πρόκειται δηλαδή, στην κυριολεξία, για 'χρήση' και όχι για 'κλήση' ενός module.

Ένα απλό module, που αυτοματοποιεί μετατροπές του είδους των διαφόρων

μεταβλητών στο κύριο πρόγραμμα της προηγούμενης ενότητας, εμφανίζεται στο ακόλουθο πρόγραμμα.

```
      MODULE SIMPLE
      INTEGER,PARAMETER :: KA=4,KB=8,KD=10
      END MODULE SIMPLE
!---------
      PROGRAM TYPES_AND_KINDS
      USE SIMPLE
      IMPLICIT REAL(KIND=KA) (A-C,T)
      IMPLICIT REAL(KIND=KB) (D-H)
      IMPLICIT REAL(KIND=KD) (O-S,U-W)
      IMPLICIT COMPLEX(KIND=KD) (X-Z)
      INTEGER :: FLAG_ONE,FLAG_TWO,FORES
      REAL(KIND=KA) :: DURATION
      REAL(KIND=KB) :: STEP_ONE,STEP_TWO
      REAL(KIND=KD) :: ELN,LENGTH,LINEAR_MOMENTUM,MASS
      COMPLEX(KIND=KD) :: RADIUS_1,RADIUS_2,RADIUS_3
      ...
      T_START=1.2E1_KA
      T_END=2.4E1_KA
      DISTANCE=7.842376395554E3_KB
      PI=4*ATAN(1._KD)
      ELN=EXP(1._KD)
      RADIUS_1=CMPLX(1.71_KD,3.44_KD,KD)
      ...
      END PROGRAM TYPES_AND_KINDS
```

1.3 Κύριο πρόγραμμα με υποπρογράμματα

Στην περίπτωση που, κατά την επίλυση ενός προβλήματος, εμπλέκεται αρκετές φορές η ίδια υπολογιστική διαδικασία, τότε είναι προτιμότερο να αποχωρισθεί από το σώμα του κύριου προγράμματος ο αντίστοιχος κώδικας και να λάβει τη μορφή ενός ανεξάρτητου υποπρογράμματος.

Έστω π.χ. ότι σε ένα πρόβλημα πρέπει να υπολογίσουμε διάφορα εσωτερικά γινόμενα διανυσμάτων. Ακολουθεί ένα πρόγραμμα, στο οποίο φαίνεται η διαδικασία ανάγνωσης δεδομένων και υπολογισμού του εσωτερικού γινομένου με χρήση μίας υπορουτίνας.

```
      MODULE KINDS_ETC
      INTEGER,PARAMETER :: KD=10
      INTEGER,PARAMETER :: NDIM=16
      END MODULE KINDS_ETC
!---------
      PROGRAM COMPUTE_DOTPRODUCTS
      USE KINDS_ETC
      REAL(KIND=KD) :: A(NDIM),B(NDIM),DP
      EXTERNAL DOTPRO
      ...
      PRINT*,' NUMBER OF VECTOR COMPONENTS ? '
      READ*,N
      PRINT*,' COMPONENTS OF VECTOR A ? '
      READ*,A
      PRINT*,' COMPONENTS OF VECTOR B ? '
      READ*,B
      CALL DOTPRO(N,A,B,DP)
      PRINT*,' DOT PRODUCT = ',DP
      ...
      END PROGRAM COMPUTE_DOTPRODUCTS
!---------
```

```
      SUBROUTINE DOTPRO(NOFC,VECTOR_1,VECTOR_2,RESULT)
! Use module kinds_etc or set the integer named constant kd equal to the
! required kind, coinciding to that of the main program. So, convert one
! of the two following statements into comment.
      USE KINDS_ETC
!     INTEGER,PARAMETER :: KD=10
      REAL(KIND=KD) :: VECTOR_1(NOFC),VECTOR_2(NOFC),RESULT
      RESULT=0
      DO I=1,NOFC
          RESULT=RESULT+VECTOR_1(I)*VECTOR_2(I)
      END DO
      END SUBROUTINE DOTPRO
```

Το κύριο πρόγραμμα διαβάζει δεδομένα για τα δύο διανύσματα και καλεί την υπορουτίνα DOTPRO που υπολογίζει το εσωτερικό γινόμενό τους.

Έχοντας σχεδιάσει και γράψει εμείς οι ίδιοι αυτό το πρόγραμμα, γνωρίζουμε τον τρόπο με τον οποίο το κύριο πρόγραμμα καλεί το υποπρόγραμμα. Άν, όμως, πρόκειται να το δώσουμε σε τρίτους για να το χρησιμοποιήσουν, τότε πρέπει να γράψουμε οδηγίες για τη σωστή «οδήγηση» (driving, driver) της υπορουτίνας DOTPRO. Οι οδηγίες πρέπει να είναι περιληπτικές, σαφείς, και κυρίως να εξηγούν ποιές είναι οι «μεταβλητές εισόδου» (input) και ποιές οι «μεταβλητές εξόδου» (output). Εστιάζοντας σε αυτό, μπορούμε να γράψουμε τις ακόλουθες οδηγίες.

...

ΕΙΣΟΔΟΣ

NOFC = πλήθος συνιστωσών των δοσμένων διανυσμάτων.

VECTOR_1 = μονοδιάστατος πίνακας NOFC στοιχείων, τύπου REAL και είδους KD.

VECTOR_2 = μονοδιάστατος πίνακας NOFC στοιχείων, τύπου REAL και είδους KD.

ΕΞΟΔΟΣ

RESULT = το εσωτερικό γινόμενο των VECTOR_1 και VECTOR_2, θεωρουμένων ως διανυσμάτων, τύπου REAL και είδους KD.

ΠΑΡΑΤΗΡΗΣΕΙΣ

Το είδος KD πρέπει να συμπίπτει με το αντίστοιχο είδος του κύριου προγράμματος. Αυτό επιτυγχάνεται είτε με τη χρήση του module που χρησιμοποιεί και το κύριο πρόγραμμα, είτε με τον ορισμό μίας ακέραιας ονομασμένης σταθεράς KD μέσα στην υπορουτίνα, στην οποία δίνεται τιμή ίδια με αυτή του κύριου προγράμματος. Ανάλογα με την επιλογή του, ο χρήστης πρέπει να μετατρέψει σε σχόλιο (comment) τη μία από τις δύο πρώτες εντολές.

...

1.4 Η Βιβλιοθήκη SLATEC

Όταν πρόκειται να γράψουμε ένα πρόγραμμα για την επίλυση ενός προβλήματος, συνήθως κάνουμε χρήση έτοιμων υποπρογραμμάτων που υπάρχουν στις μεγάλες «βιβλιοθήκες λογισμικού» (software libraries). Τέτοια βιβλιοθήκη είναι και η «Μαθηματική Βιβλιοθήκη SLATEC» (SLATEC Common Mathematical Library, SLATEC CML) [4]. Πρόκειται για μία εκτεταμένη βιβλιοθήκη «ανοικτού πηγαίου κώδικα» (open source code) σε γλώσσα Fortran, η οποία αποτελεί «ελεύθερο λογισμικό» (free software), ή, όπως αλλοιώς λέμε, εντάσσεται στην «δημόσια περιοχή λογισμικού» (public-domain software).

Η φιλική έκδοση της SLATEC, που περιέχεται στο συνοδευτικό CD, διαθέτει το έγγραφο περιήγησης και επιλογής gams.html. Το περιεχόμενο, προς το οποίο μπορεί να μας κατευθύνει αυτό, περιγράφεται αναλυτικά στο έγγραφο περιήγησης toc.html ή, εναλλακτικά, στο έγγραφο κειμένου toc.html, που περιέχονται και αυτά στο συνοδευτικό CD.

1.5 Αναζήτηση και επιλογή υποπρογραμμάτων στην SLATEC

Έστω ότι σε ένα πρόβλημα χρειάζεται να υπολογίσουμε όλες τις ρίζες ενός πολυωνύμου με πραγματικούς συντελεστές. Τότε η διαδικασία αναζήτησης και επιλογής ενός κατάλληλου υποπρογράμματος έχει ως εξής.
1. Ανοίγουμε το έγγραφο gams.html. και επιλέγουμε την κατηγορία "F. Solution of nonlinear equations".
2. Το έγγραφο αυτό μάς κατευθύνει στις υποπεριπτώσεις τής F. Εκεί επιλέγουμε την υποκατηγορία "F1. Single equation".
3. Οδηγούμαστε στις υποπεριπτώσεις τής F1. Εκεί επιλέγουμε την υποκατηγορία "F1A1. Polynomial".
4. Οδηγούμαστε στις υποπεριπτώσεις τής F1A1. Στην υποκατηγορία "F1A1A. Real coefficients", στην οποία κατατάσσεται το πρόβλημά μας, διαπιστώνουμε ότι μπορούμε να επιλέξουμε μεταξύ των υπορουτινών `RPQR79`, και `RPZERO`.
5. Διαβάζοντας τις οδηγίες τους, διαπιστώνουμε ότι η πρώτη μετασχηματίζει το πρόβλημα στο αντίστοιχο τού υπολογισμού των ιδιοτιμών της λεγόμενης «συνοδεύουσας μήτρας» (companion matrix) ενός πολυωνύμου. Με την υπόθεση ότι δεν επιθυμούμε να εμπλέξουμε μήτρες στην επίλυση του προβλήματός μας, επιλέγουμε την υπορουτίνα `RPZERO`.
6. Οδηγούμαστε στον φάκελο `_rpzero`, οποίος περιέχει όλο το απαραίτητο υλικό για τη χρήση αυτής της υπορουτίνας. Λεπτομερής περιγραφή για τα επόμενα βήματα χρήσης θα δοθεί στη συνέχεια.

1.6 Υπολογισμός του σφάλματος στρογγύλευσης μηχανής

Το «σφάλμα στρογγύλευσης μηχανής» (machine roundoff error, MRE) ορίζεται ως η μεγαλύτερη ποσότητα, U, που όταν προστίθεται στη μονάδα την αφήνει αναλλοίωτη λόγω στρογγύλευσης. Η ποσότητα αυτή είναι διαφορετική για την απλή ακρίβεια `KIND=4`, τη διπλή ακρίβεια `KIND=8`, και την υψηλή ακρίβεια `KIND=10`. Ο αλγόριθμος υπολογισμού του MRE έχει ως εξής.
1. Αρχικά δίνεται στο `U` η τιμή 0.5.
2. Στο εσωτερικό μίας ατέρμονος δομής `DO`, η ποσότητα `U` διαιρείται σε κάθε επανάληψη δια του 2 και εκχωρείται η τρέχουσα τιμή $1 + U$ στη μεταβλητή `R`.
3. Συνεχίζονται οι επαναλήψεις όσο ισχύει η συνθήκη `R > 1`.
4. Όταν τερματίζεται η ατέρμων δομή `DO`, η μεταβλητή `U` έχει γίνει ίση με το MRE.

Ο αλγόριθμος αυτός υλοποιείται στο ακόλουθο πρόγραμμα.

```
MODULE MODULE_KINDS
INTEGER,PARAMETER :: KD=4
REAL(KIND=KD),PARAMETER :: PI=4*ATAN(1._KD)
REAL(KIND=KD),PARAMETER :: ELN=EXP(1._KD)
```

```
      END MODULE MODULE_KINDS
!---------
      PROGRAM MRE
      USE MODULE_KINDS
      REAL(KIND=KD) :: R,U
      U=0.5_KD
      DO
       U=U/2._KD
       R=1._KD+U
       IF (R>1._KD) THEN
           CYCLE
       ELSE
           PRINT*
           PRINT*,' MRE                   = ',U
           PRINT*
           PRINT*,' PI  IN SAME PRECISION = ',PI
           PRINT*
           PRINT*,' ELN IN SAME PRECISION = ',ELN
           EXIT
       END IF
      END DO
      END PROGRAM MRE
```

Το κύριο πρόγραμμα εκτυπώνει επίσης τις τιμές των μαθηματικών σταθερών π και e (βάση των φυσικών λογαρίθμων), έτσι ώστε ο χρήστης να έχει μία ρεαλιστική εικόνα της συγκεκριμένης ακρίβειας. **Αξιοσημείωτος είναι ο τρόπος με τον οποίο ορίζονται οι αντίστοιχες πραγματικές ονομασμένες σταθερές** `PI` **και** `ELN`. Επίσης, πρέπει να προσεχθεί ιδιαίτερα το γεγονός ότι, σε κάθε επανάληψη, η τρέχουσα τιμή `1+U` εκχωρείται στη μεταβλητή `R`, που είναι ίδιου (τύπου και) είδους με την `U`. Στη συνέχεια, ο έλεγχος για τις επιπτώσεις της στρογγύλευσης πραγματοποιείται επί της μεταβλητής `R`, και όχι επί του αθροίσματος `1+U`.

Άν ελέγχαμε απευθείας τη συμπεριφορά του αθροίσματος `1+U`, γράφοντας

```
...
IF (1._KD+U > 1._KD) THEN
    CYCLE
ELSE
...
```

τότε, αντί να υπολογίσουμε το σφάλμα στρογγύλευσης μηχανής για τη συγκεκριμένη ακρίβεια `KD`, θα υπολογίζαμε το **σφάλμα στρογγύλευσης του μαθηματικού επεξεργαστή**. Το σφάλμα αυτό αναφέρεται συνήθως ως «σφάλμα στρογγύλευσης μηχανής κατά την εκτέλεση» (machine roundoff error on the fly, MREF).

Όπως έχει γραφτεί, το πρόγραμμα υπολογίζει το σφάλμα στρογγύλευσης μηχανής για την απλή ακρίβεια `KIND=4`. Αλλάζοντας την τιμή της ακέραιας ονομασμένης σταθεράς `KD` σε '8' ή '10', επιτυγχάνουμε τον υπολογισμό του σφάλματος αυτού στη διπλή ακρίβεια ή στην υψηλή ακρίβεια, αντίστοιχα.

1.7 Υπολογιστική ταχύτητα

Η μέτρηση του «χρόνου εκτέλεσης» (execution time, computation time) μίας υπολογιστικής διαδικασίας επιτυγχάνεται με χρήση της «εσωτερικής υπορουτίνας» (Fortran library subroutine, intrinsic subroutine) `CPU_TIME(T)`, όπου η μεταβλητή `T` είναι απλής ακρίβειας. Χρειάζεται να καλέσουμε την υπορουτίνα αυτή στην

αρχή και στο τέλος της υπολογιστικής διαδικασίας, οπότε ο χρόνος εκτέλεσης είναι ίσος με τη διαφορά του τελικού μείον τον αρχικό χρόνο.

Ακολουθεί ένα πρόγραμμα, στο οποίο γίνεται χρήση του προηγούμενου σχήματος χρονομέτρησης.

```
      MODULE MODULE_KINDS
      INTEGER,PARAMETER :: KA=4,KB=8,KD=10
      END MODULE MODULE_KINDS
!---------
      PROGRAM TRY
      USE MODULE_KINDS
      ...
      REAL(KIND=KA) :: T0,T1,DURATION
      ...
      CALL CPU_TIME(T0)
! Required computational process
      ...
      CALL CPU_TIME(T1)
      DURATION=T1-T0
      PRINT*,' DURATION = ',DURATION
      ...
      END PROGRAM TRY
```

Η «υπολογιστική ταχύτητα» (computational speed) ενός υπολογιστή μπορεί να εκφρασθεί με αρκετούς τρόπους. Ένας απλός τρόπος είναι να μετρήσουμε τον χρόνο που χρειάζεται για να εκτελεσθεί ένα μεγάλο πλήθος κλήσεων (και αντίστοιχων υπολογισμών) μίας τυπικής «εσωτερικής συνάρτησης» (Fortran library function, intrinsic function)· συνήθως επιλέγουμε τη συνάρτηση του υπερβολικού ημιτόνου, `SINH`, επειδή έχει σχετικά μεγάλο (άρα καλώς μετρήσιμο) χρόνο υπολογισμού. Στη συνέχεια, μπορεί να γίνει αναγωγή του χρόνου εκτέλεσης σε υπολογιστική ταχύτητα με διαίρεση του πλήθους `N` των κλήσεων δια του χρόνου εκτέλεσης, `N/DURATION`· το αποτέλεσμα αυτό εκφράζει «πλήθος κλήσεων ανα δευτερόλεπτο».

Στο πρόγραμμα που ακολουθεί γίνεται εκτίμηση της υπολογιστικής ταχύτητας για ένα πλήθος επαναλήψεων καθοριζόμενο από τον χρήστη.

```
      MODULE MODULE_KINDS
      INTEGER,PARAMETER :: KA=4,KD=4
      REAL(KIND=KD),PARAMETER :: PI=4*ATAN(1._KD)
      END MODULE MODULE_KINDS
!---------
      PROGRAM B01SINH
      USE MODULE_KINDS
      IMPLICIT REAL(KIND=KD) (A-H,O-Z)
      REAL(KIND=KA) :: DURATION,TM0,TM1
      PRINT *,' ITERATIONS: ITER ? '
      READ*,ITER
      PRINT*
      CALL CPU_TIME(TM0)
      DO I=1,ITER
          X=I*PI/(I+1)
          Y=SINH(X)
      END DO
      CALL CPU_TIME(TM1)
      DURATION=TM1-TM0
      PRINT *
      PRINT *,' EXECUTION TIME (SECONDS)                    = ',DURATION
      PRINT *
      PRINT *,' COMPUTATIONAL SPEED (EVALUATIONS PER SECOND)= ',        &
     &                                                      ITER/DURATION
      END PROGRAM B01SINH
```

Όπως έχει γραφτεί, το πρόγραμμα βρίσκει την υπολογιστική ταχύτητα για την απλή ακρίβεια `KIND=4`. Αλλάζοντας την τιμή της ακέραιας ονομασμένης σταθεράς `KD` σε '8' ή '10', επιτυγχάνουμε τον υπολογισμό τής ταχύτητας αυτής στη διπλή ακρίβεια ή στην υψηλή ακρίβεια, αντίστοιχα.

Κεφάλαιο 2

Ρίζες Μή Γραμμικών Αλγεβρικών Εξισώσεων

2.1 Πολυωνυμικές εξισώσεις

Οι εξισώσεις της μορφής

$$p_n(x) = 0, \tag{2.1}$$

όπου $p_n(x)$ πολυώνυμο βαθμού n με πραγματικούς ή μιγαδικούς συντελεστές, ονομάζονται πολυωνυμικές εξισώσεις. Αυτές είναι μή γραμμικές αλγεβρικές εξισώσεις, πλην της περίπτωσης $n = 1$.

Στη βιβλιογραφία, τα πολυώνυμα παριστάνονται με διαφόρους τρόπους. Συνήθως γίνεται διακρίση της περίπτωσης που οι ανιόντες πολυωνυμικοί συντελεστές αντιστοιχούν στις ανιούσες δυνάμεις της ανεξάρτητης μεταβλητής x, και της περίπτωσης που αυτοί αντιστοιχούν στις κατιούσες δυνάμεις του x. Στην πρώτη περίπτωση, ένα πολυώνυμο βαθμού n έχει τη μορφή

$$p_n(x) = a_1x^0 + a_2x^1 + \cdots + a_{n+1}x^n = \sum_{i=1}^{n+1} a_i x^{i-1}. \tag{2.2}$$

Η δεικτοδότηση των $n + 1$ συντελεστών αρχίζει από '1'. Η μορφή αυτή λέγεται «$A_1 - A$ πολυωνυμική μορφή» (ascending coefficients, ascending powers). Παλαιότερα, στις γλώσσες προγραμματισμού, δεν ήταν δυνατό να αρχίζει η δεικτοδότηση των πινάκων (arrays) από '0'. Σήμερα, που αυτό είναι εφικτό, μπορούμε να χρησιμοποιήσουμε και στον προγραμματισμό την εναλλακτική μορφή

$$p_n(x) = a_0x^0 + a_1x^1 + \cdots + a_nx^n = \sum_{i=0}^{n} a_i x^i, \tag{2.3}$$

η οποία λέγεται «$A_0 - A$ πολυωνυμική μορφή».

Στη δεύτερη περίπτωση, ένα πολυώνυμο βαθμού n γράφεται

$$p_n(x) = a_1x^n + a_2x^{n-1} + \cdots + a_{n+1}x^0 = \sum_{i=1}^{n+1} a_i x^{n+1-i}. \tag{2.4}$$

Εδώ η δεικτοδότηση των $n+1$ συντελεστών αρχίζει πάλι από '1', αλλά η ανεξάρτητη μεταβλητή x εμφανίζεται με κατιούσες δυνάμεις. Η μορφή αυτή αναφέρεται ως

«$A_1 - D$ πολυωνυμική μορφή» (ascending coefficients, discending powers). Η εναλλακτική μορφή με τη δεικτοδότηση να αρχίζει από '0' γράφεται

$$p_n(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_n x^0 = \sum_{i=0}^{n} a_i x^{n-i} \tag{2.5}$$

και αναφέρεται ως «$A_0 - D$ πολυωνυμική μορφή».

Όταν οι πολυωνυμικοί συντελεστές εμφανίζονται με κατιόντες δείκτες, τότε: (1) υπάρχει ταύτιση με το σχήμα $A - A$ άν οι δυνάμεις του x είναι κατιούσες· (2) υπάρχει ταύτιση με το σχήμα $A - D$ άν οι δυνάμεις του x είναι ανιούσες.

Άν ένας αλγόριθμος έχει γραφτεί έτσι ώστε να διαχειρίζεται ένα πολυώνυμο σε μορφή $A_1 - D$, οι συντελεστές όμως του πολυωνύμου έχουν δοθεί σε μορφή $A_0 - A$, τότε, για να κάνουμε χρήση του αλγορίθμου, πρέπει να βρούμε τους συντελεστές της μορφής $A_1 - D$. Έστω

$$p_n(x) = a_0 + a_1 x^1 + \cdots + a_n x^n \tag{2.6}$$

το πολυώνυμο όπως έχει δοθεί, και

$$p_n(x) = b_1 x^n + b_2 x^{n-1} + \cdots + b_{n+1}, \tag{2.7}$$

το πολυώνυμο όπως εισάγεται στον αλγόριθμο. Η σύγκριση των δύο σχέσεων δείχνει ότι $b_1 = a_n$, $b_2 = a_{n-1}$, $\ldots$, $b_{n+1} = a_0$.

Ακολουθεί ένα πρόγραμμα που εφαρμόζει τη μετατροπή των πολυωνυμικών συντελεστών.

```
      MODULE SIMPLE
      INTEGER,PARAMETER :: KD=10
      END MODULE SIMPLE
!---------
      PROGRAM CONVERT_COEFFICIENTS
      USE SIMPLE
      INTEGER,PARAMETER :: NDIM=64,NDP1=NDIM+1
      REAL(KIND=KD) :: A(0:NDIM),B(1:NDP1),RESULT,X
      ...
      PRINT*,' N,X ? '
      READ*,N,X
      NP1=N+1
      PRINT*,' A ? '
      READ*,A(0:N)
      B(1:NP1)=A(N:0:-1)
! Or, equivalently,
!     DO I=1,NP1
!         B(I)=A(NP1-I)
!     END DO
      CALL ALGORITHM(NP1,B,X,RESULT)
      ...
      END PROGRAM CONVERT_COEFFICIENTS
!---------
      SUBROUTINE ALGORITHM(NP1,B,X,R)
      USE SIMPLE
      REAL(KIND=KD) :: B(NP1),R,X
      ...
      END SUBROUTINE ALGORITHM
```

2.2 Ταχύς υπολογισμός πολυωνύμων

Έστω το πολυώνυμο τρίτου βαθμού

$$p_3(x) = a_0 + a_1 * x + a_2 * x^2 + a_3 * x^3. \tag{2.8}$$

Φαίνεται εύκολα ότι, για να υπολογισθεί η τιμή του σε δοσμένο x, χρειάζονται 3 προσθέσεις, $L_+ = 3$, 3 πολλαπλασιασμοί, $L_* = 3$, και δύο υψώσεις σε δύναμη, $L_{**} = 2$. Οι τελευταίες αναλύονται σε 1 πολλαπλασιασμό για το $x^2 = x*x$ και σε 2 πολλαπλασιασμούς για το $x^3 = x*x*x$. Οπότε το πλήθος των πολλαπλασιασμών από τις υψώσεις σε δύναμη είναι

$$L_{**} = \sum_{i=1}^{3-1} i = \frac{1}{2}(3-1)\,3 = 3[*]. \tag{2.9}$$

Γενικεύοντας το αποτέλεσμα για ένα πολυώνυμο βαθμού n, έχουμε

$$L(p_n(x)) = L_+ + L_* + L_{**} = n[+] + n[*] + \frac{1}{2}(n-1)\,n[*] = n[+] + \frac{1}{2}\,n\,(n+1)[*]. \tag{2.10}$$

Επανερχόμαστε τώρα στο $p_3(x)$ και το γράφουμε ως εξής

$$p_3(x) = a_0 + x*(a_1 + x*(a_2 + x*a_3)). \tag{2.11}$$

Η μορφή αυτή λέγεται «εμφωλευμένη δομή πολυωνυμικού πολλαπλασιασμού» (βλέπε π.χ. [7], §1.3, Eq. (1.21)), ή και «σχήμα Horner» παρά το γεγονός ότι στη βιβλιογραφία οι δύο αυτές έννοιες δεν είναι πάντοτε ταυτόσημες (βλέπε π.χ. [7], §1.3, Eqs. (1.22)–1.25). Παρατηρούμε ότι για τον υπολογισμό του σχήματος Horner χρειάζονται 3[+] και 3[∗], δηλαδή απαλείφονται οι πολλαπλασιασμοί από τις υψώσεις σε δύναμη. Γενικεύοντας, συμπεραίνουμε ότι για τον υπολογισμό ενός πολυωνύμου βαθμού n σε σχήμα Horner χρειάζεται το ακόλουθο πλήθος πράξεων

$$L_{\rm H}(p_n(x)) = L_{{\rm H},+} + L_{{\rm H},*} = n[+] + n[*]. \tag{2.12}$$

Η παράγωγος του πολυωνύμου $p_3(x)$ γράφεται διαδοχικά

$$p_3'(x) = a_1 + 2a_2x + 3a_3x^2 = a_1 + x*((2*a_2) + x*(3*a_3)). \tag{2.13}$$

Εδώ εφαρμόζεται το σχήμα Horner στην παράγωγο, που είναι πολυώνυμο δευτέρου βαθμού με συντελεστές $d_0 = a_1$, $d_1 = 2a_2$, $d_2 = 3a_3$. Η δεύτερη παράγωγος του $p_3(x)$ γράφεται διαδοχικά

$$p_3''(x) = 2a_2 + 6a_3x = (2*a_2) + x*(6*a_3). \tag{2.14}$$

Εφαρμόζεται και πάλι το σχήμα Horner στη δεύτερη παράγωγο, που είναι πολυώνυμο πρώτου βαθμού με συντελεστές $f_0 = d_1 = 2a_2$, $f_1 = 2d_2 = 6a_3$. Τα αποτελέσματα αυτά γενικεύονται εύκολα για ένα πολυώνυμο βαθμού n, οπότε η παράγωγός του είναι πολυώνυμο βαθμού $n-1$, και η δεύτερη παράγωγός του είναι πολυώνυμο βαθμού $n-2$.

Εφαρμογή του σχήματος Horner γίνεται στο ακόλουθο πρόγραμμα (βλέπε και [1], Κεφ. 3, §8, p. 81)

```
      MODULE SIMPLE
      INTEGER,PARAMETER :: KD=10
      END MODULE SIMPLE
!---------
      PROGRAM HORNER_SCHEME
      USE SIMPLE
      INTEGER,PARAMETER :: NDIM=64,NDM1=NDIM-1,NDM2=NDIM-2
      REAL(KIND=KD) :: A(0:NDIM),D(0:NDM1),DD(0:NDM2)
```

```
      REAL(KIND=KD) :: DDPOLY,DPOLY,POLY,X
      ...
      PRINT*,' N,X ? '
      READ*,N,X
      NM1=N-1
      NM2=N-2
      PRINT*,' A ? '
      READ*,A(0:N)
! The value of the polynomial at x is denoted by poly.
      POLY=A(N)
      DO I=N-1,0,-1
          POLY=POLY*X+A(I)
      END DO
! The value of the polynomial derivative at x is denoted by dpoly.
! The coefficients of the polynomial derivative are denoted by d(i).
      DO I=0,NM1
          D(I)=(I+1)*A(I+1)
      END DO
      DPOLY=D(NM1)
      DO I=NM1-1,0,-1
          DPOLY=DPOLY*X+D(I)
      END DO
! The value of the second derivative at x is denoted by ddpoly.
! The coefficients of the second derivative are denoted by dd(i).
      DO I=0,NM2
          DD(I)=(I+1)*D(I+1) ! Equivalently, (I+1)*(I+2)*A(I+2)
      END DO
      DDPOLY=DD(NM2)
      DO I=NM2-1,0,-1
          DDPOLY=DDPOLY*X+DD(I)
      END DO
      ...
      END PROGRAM HORNER_SCHEME
```

Άν θέλουμε να κάνουμε χρήση ενός υποπρογράμματος function, τότε μπορούμε να γράψουμε το ακόλουθο πρόγραμμα.

```
      MODULE SIMPLE
      INTEGER,PARAMETER :: KD=10
      END MODULE SIMPLE
!---------
      PROGRAM HORNER_SCHEME_AS_FUNCTION
      USE SIMPLE
      INTEGER,PARAMETER :: NDIM=64,NDM1=NDIM-1,NDM2=NDIM-2
      REAL(KIND=KD) :: A(0:NDIM),D(0:NDM1),DD(0:NDM2)
      REAL(KIND=KD) :: DDPOLY,DPOLY,FPOLY,POLY,X
      EXTERNAL FPOLY
      ...
      PRINT*,' N,X ? '
      READ*,N,X
      NM1=N-1
      NM2=N-2
      PRINT*,' A ? '
      READ*,A(0:N)
      POLY=FPOLY(N,A,X)        ! Polynomial value at x
      DO I=0,NM1
          D(I)=(I+1)*A(I+1)
      END DO
      DPOLY=FPOLY(NM1,D,X)     ! Derivative value at x
      DO I=0,NM2
          DD(I)=(I+1)*D(I+1)   ! Equivalently, (I+1)*(I+2)*A(I+2)
      END DO
      DDPOLY=FPOLY(NM2,DD,X)   ! Second derivative value at x
```

```
      ...
      END PROGRAM HORNER_SCHEME_AS_FUNCTION
!---------
      FUNCTION FPOLY(N,A,X)
      USE SIMPLE
      REAL(KIND=KD) :: A(0:N),FPOLY,X
      FPOLY=A(N)
      DO I=N-1,0,-1
          FPOLY=FPOLY*X+A(I)
      END DO
      END FUNCTION FPOLY
```

Τέλος, επειδή αρκετοί αλγόριθμοι χρειάζονται ταυτόχρονα τις τιμές p, p', p'' σε δοσμένο x, μπορούμε να μετατρέψουμε τη συνάρτηση σε υπορουτίνα και να χρησιμοποιήσουμε τον ακόλουθο 'λακωνικό' κώδικα.

```
      MODULE SIMPLE
      INTEGER,PARAMETER :: KD=10
      END MODULE SIMPLE
!---------
      PROGRAM HORNER_SCHEME_AS_SUBROUTINE
      USE SIMPLE
      INTEGER,PARAMETER :: NDIM=64
      REAL(KIND=KD) :: A(0:NDIM)
      REAL(KIND=KD) :: DDPOLY,DPOLY,POLY,X
      EXTERNAL POLYNOMIAL
      ...
      PRINT*,' N,X ? '
      READ*,N,X
      PRINT*,' A ? '
      READ*,A(0:N)
      CALL POLYNOMIAL(N,A,X,POLY,DPOLY,DDPOLY)
      ...
      END PROGRAM HORNER_SCHEME_AS_SUBROUTINE
!---------
      SUBROUTINE POLYNOMIAL(N,A,X,P,DP,DDP)
      USE SIMPLE
      REAL(KIND=KD) :: A(0:N),DDP,DP,P,X
      P=A(N)
      DP=0
      DDP=0
      DO I=N-1,0,-1
          DDP=DDP*X+DP
          DP=DP*X+P
          P=P*X+A(I)
      END DO
      DDP=2*DDP
      END SUBROUTINE POLYNOMIAL
```

2.3 Σημαντικά θεωρήματα της άλγεβρας για τις ρίζες των πολυωνυμικών εξισώσεων

Υπάρχουν στην άλγεβρα πολλά θεωρήματα που ασχολούνται με τις πολυωνυμικές εξισώσεις. Θα αναφέρουμε εδώ τα πιο σημαντικά θεωρήματα.

Το πρώτο θεώρημα είναι 'διάσημο' στον χώρο των μαθηματικών. Έχει μακρόχρονη ιστορία με αρκετούς εξαίρετους μαθηματικούς να έχουν ασχοληθεί με αυτό.
1. Θεμελιώδες θεώρημα της άλγεβρας. Κάθε πολυωνυμική εξίσωση $p_n(x) = 0$ έχει ακριβώς n πραγματικές ή/και μιγαδικές ρίζες.

Πρέπει να διευκρινισθεί εδώ ότι ότι κάθε ρίζα συνεισφέρει στο συνολικό πλήθος n των ριζών κατά τη λεγόμενη «πολλαπλότητα» (multiplicity) αυτής. Στο πολυώνυμο βαθμού 6, π.χ., που προκύπτει από την εκτέλεση των πράξεων στην παράσταση

$$p_6(x) = (x-a)(x-b)(x-b)(x-c)(x-c)(x-c), \tag{2.15}$$

η ρίζα a είναι «απλή ρίζα» (single root) ή «ρίζα πολλαπλότητας 1» (root of multiplicity 1), η ρίζα b είναι «διπλή ρίζα» (double root) ή «ρίζα πολλαπλότητας 2» (root of multiplicity 2), και η ρίζα c είναι «τριπλή ρίζα» (triple root) ή «ρίζα πολλαπλότητας 3» (root of multiplicity 3). Συνεπώς το συνολικό πλήθος των ριζών του είναι: 1 (από την a) + 2 (από την b) + 3 (από την c) = 6 (όπως προβλέπει το θεμελιώδες θεώρημα).

Γενικά, μία ρίζα a της πολυωνυμικής εξίσωσης $p_n(x) = 0$ έχει πολλαπλότητα m άν το $p_n(x)$ μπορεί να γραφτεί ως

$$p_n(x) = (x-a)^m\, q(x), \tag{2.16}$$

όπου το πολυώνυμο $q(x)$ είναι βαθμού $n-m$. Τότε ισχύει το ακόλουθο θεώρημα.

2. Θεώρημα για τις πολλαπλότητες των ριζών. Άν μία ρίζα a της πολυωνυμικής εξίσωσης $p_n(x) = 0$ έχει πολλαπλότητα m, τότε ισχύει

$$p_n(a) = p_n'(a) = p_n''(a) = \cdots = p_n^{(m-1)}(a) = 0, \qquad p_n^{(m)}(a) \neq 0. \tag{2.17}$$

Ένα άλλο σημαντικό θεώρημα εστιάζει στα πραγματικά πολυώνυμα, δηλαδή στα πολυώνυμα με πραγματικούς συντελεστες.

3α. Θεώρημα των πραγματικών συντελεστών. Άν το πολυώνυμο $p_n(x)$ είναι πραγματικό, τότε για κάθε (ενδεχόμενη) μιγαδική ρίζα $a + i\,b$ της εξίσωσης $p_n(x) = 0$, υπάρχει μία ακόμη μιγαδική ρίζα, ίση με τη συζυγή της, $a - i\,b$.

Από το θεώρημα αυτό προκύπτει ένα πόρισμα πολύ χρήσιμο στη φυσική.

3β. Πόρισμα για πραγματικά πολυώνυμα περιττού βαθμού. Ένα πραγματικό πολυώνυμο περιττού βαθμού έχει μία τουλάχιστον πραγματική ρίζα.

Πράγματι, αφού οι (ενδεχόμενες) μιγαδικές ρίζες εμφανίζονται κατά ζεύγη, απομένει μία τουλάχιστον ρίζα a που δεν ανήκει σε ένα ζεύγος· άρα είναι πραγματική ρίζα, $a \in \mathbb{R}$. Επειδή, στη συντριπτική πλειονότητά τους, οι φυσικές ποσότητες θεωρούνται πραγματικές, το να περιγράφεται σε ένα σύστημα μία ποσότητα ως ρίζα πραγματικού πολυωνύμου περιττού βαθμού, σημαίνει ότι μπορεί να λάβει τουλάχιστον μία πραγματική τιμή, δηλαδή ότι υπάρχει λύση με φυσική σημασία.

Σε σχέση με τα πραγματικά πολυώνυμα, υπάρχει ένα απλό θεώρημα που διευκολύνει την εύρεση μίας ή δύο αρχικών ριζών τους.

4. Θεώρημα των πραγματικών συντελεστών με άθροισμα μηδέν, ή με συνδυασμούς αθροισμάτων μηδέν. Άν το πραγματικό πολυώνυμο $p_n(x)$ έχει άθροισμα συντελεστών μηδέν, τότε έχει ως μία ρίζα του το $+1$. Άν, επιπρόσθετα, τα επιμέρους αθροίσματα αρτίων και περιττών συντελεστών στη μορφή $A_0 - A$ είναι μηδέν (ο συντελεστής a_0 θεωρείται άρτιος), τότε έχει ως ρίζες του τις ± 1. Συμπληρωματικά, άν το άθροισμα των περιττών συντελεστών είναι ίσο με το άθροισμα των αρτίων συντελεστών, τότε έχει ως μία ρίζα του το -1.

Ακολουθεί ένα θεώρημα πολύ χρήσιμο για τους σχετικούς αλγορίθμους της αριθμητικής ανάλυσης, αφού 'οριοθετεί' την περιοχή στην οποία βρίσκονται όλες οι ρίζες ενός πολυωνύμου.

5. Θεώρημα του τόπου των ριζών. Ο τόπος των ριζών μίας πολυωνυμικής εξίσωσης $p_n(x) = 0$ με συντελεστές $a_0, a_1, \ldots, a_n$, δοσμένους κατά τις ανιούσες

δυνάμεις της ανεξάρτητης μεταβλητής x, είναι κύκλος με κέντρο το σημείο $(0,0)$ και ακτίνα r ίση με τη μέγιστη των τιμών $1+|a_0/a_n|$, $1+|a_1/a_n|$, $\ldots$, $1+|a_{n-1}/a_n|$.

Άς πάρουμε ως παράδειγμα το πολυώνυμο

$$p_7(x) = 1 - 2x - 3x^2 - 4x^3 - 5x^4 - 6x^5 - 7x^6 + x^7. \tag{2.18}$$

Εύκολα φαίνεται ότι ο μέγιστος λόγος $|a_i/a_7|$, $i = 0, 1, \ldots, 6$, είναι ο $|a_6/a_7| = 7$, οπότε $r = 1+7 = 8$. Συνεπώς κάθε μία από τις επτά ρίζες του πολυωνύμου αυτού κείται στην κυκλική περιοχή με κέντρο το σημείο $(0,0)$ και ακτίνα 8.

2.4 Πολυωνυμική απομείωση

Άν το a είναι ρίζα της εξίσωσης

$$p_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = 0, \tag{2.19}$$

τότε το πολυώνυμο $p_n(x)$ μπορεί να γραφτεί ως $p_n(x) = (x-a)q_{n-1}(x)$ (Εξ. 2.16). Το πολυώνυμο $q_{n-1}(x)$ βαθμού $n-1$ είναι το αποτέλεσμα της «τέλειας διαίρεσης» (exact division)

$$q_{n-1}(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1} = \frac{p_n(x)}{(x-a)}. \tag{2.20}$$

Εκτελώντας τις πράξεις στην παράσταση $(x-a)q_{n-1}(x)$, βρίσκουμε

$$p_n(x) = b_{n-1}x^n + (b_{n-2} - b_{n-1}a)x^{n-1} + \cdots + (b_0 - b_1a)x - b_0a. \tag{2.21}$$

Συγκρίνοντας με τους πολυωνυμικούς συντελεστές του $p_n(x)$ (Εξ. 2.19), επιβεβαιώνουμε ότι

$$b_{n-1} = a_n, b_{n-2} = a_{n-1}+b_{n-1}a, \ldots, b_k = a_{k+1}+b_{k+1}a,\ k = n-3,\ \ldots, 0. \tag{2.22}$$

Ο αλγόριθμος αυτός λέγεται «συνθετική διαίρεση» (synthetic division) και υλοποιείται με τη μορφή υπορουτίνας στο ακόλουθο πρόγραμμα.

```
      MODULE SIMPLE
      INTEGER,PARAMETER :: KD=10
      END MODULE SIMPLE
!---------
      PROGRAM DRV_SYNTHETIC_DIVISION
      USE SIMPLE
      INTEGER,PARAMETER :: NDIM=64,NDM1=NDIM-1
      REAL(KIND=KD) :: A(0:NDIM),B(0:NDM1)
      REAL(KIND=KD) :: ROOT_A
      EXTERNAL SYNTHETIC_DIVISION
      ...
      PRINT*,' N,ROOT_A ? '
      READ*,N,ROOT_A
      PRINT*,' A ? '
      READ*,A(0:N)
      CALL SYNTHETIC_DIVISION(N,A,B,ROOT_A)
      ...
      END PROGRAM DRV_SYNTHETIC_DIVISION
!---------
      SUBROUTINE SYNTHETIC_DIVISION(N,PC,QC,ROOT)
      USE SIMPLE
      REAL(KIND=KD) :: PC(0:N),QC(0:N-1),ROOT
```

```
QC(N-1)=PC(N)
DO I=N-2,0,-1
    QC(I)=PC(I+1)+QC(I+1)*ROOT
END DO
END SUBROUTINE SYNTHETIC_DIVISION
```

Η αναγωγή του προβλήματος της εύρεσης των n ριζών της εξίσωσης $p_n(x) = 0$ σε πρόβλημα εύρεσης των $n-1$ ριζών της εξίσωσης $q_{n-1}(x) = 0$, με την υπόθεση ότι έχει καταστεί γνωστή (με οποιονδήποτε τρόπο) μία ρίζα a της αρχικής εξίσωσης, ονομάζεται «πολυωνυμική απομείωση» (polynomial deflation). Αυτή αποτελεί τη βάση πολλων αλγορίθμων της αριθμητικής ανάλυσης που επιχειρούν να εντοπίσουν όλες τις ρίζες ενός πολυωνύμου.

Άς πάρουμε ως παράδειγμα το πολυώνυμο

$$p_4(x) = -2 + 2x + x^2 - 2x^3 + x^4. \tag{2.23}$$

Το άθροισμα των συντελεστών του είναι μηδέν· συνεπώς μία ρίζα του είναι το $+1$ (§2.3, Θεώρημα 4). Θέτοντας $a = +1$ στον αλγόριθμο της συνθετικής διαίρεσης, βρίσκουμε τους συντελεστές b_i του πολυωνύμου $q_3(x)$,

$$\begin{aligned} b_3 &= a_4 = 1, \\ b_2 &= a_3 + b_3 * a = -2 + 1 * 1 = -1, \\ b_1 &= a_2 + b_2 * a = 1 - 1 * 1 = 0, \\ b_0 &= a_1 + b_1 * a = 2 + 0 * 1 = 2. \end{aligned} \tag{2.24}$$

Οπότε

$$q_3(x) = 2 - x^2 + x^3. \tag{2.25}$$

Παρατηρούμε ότι το άθροισμα των αρτίων συντελεστών τού $q_3(x)$ είναι ίσο με το άθροισμα των περιττών συντελεστών του, οπότε μία ρίζα του είναι το -1 (§2.3, Θεώρημα 4). Θέτοντας $b = -1$ στον αλγόριθμο της συνθετικής διαίρεσης, βρίσκουμε τους συντελεστές c_i του πολυωνύμου $r_2(x) = q_3(x)/(x-b)$,

$$\begin{aligned} c_2 &= b_3 = 1, \\ c_1 &= b_2 + c_2 * b = -1 + 1 * (-1) = -2, \\ c_0 &= b_1 + c_1 * b = 0 - 2 * (-1) = 2. \end{aligned} \tag{2.26}$$

Οπότε

$$r_2(x) = 2 - 2x + x^2. \tag{2.27}$$

Τέλος, το πολυώνυμο βαθμού 2 (δηλ. το τριώνυμο) $r_2(x)$ έχει τις συζυγείς μιγαδικές ρίζες $c = 1+i$, $d = 1-i$, όπως επιβεβαιώνεται από τον γνωστό αναλυτικό τύπο της άλγεβρας (με το τριώνυμο να παριστάνεται ως $r_2(x) = \alpha x^2 + \beta x + \gamma$)

$$x = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha} = \frac{-2\gamma}{\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}} \tag{2.28}$$

2.5 Μή γραμμικές αλγεβρικές εξισώσεις (όχι απαραίτητα πολυωνυμικές): εντοπισμός μίας ρίζας

2.6 Μέθοδος της διχοτόμησης

2.6.1 Περιγραφή

Η «μέθοδος της διχοτόμησης» (bisection method, Bolzano's method) (βλέπε π.χ. [7], §4.1· επίσης [8], §2.8.1) στοχεύει στην προσέγγιση $\bar{x}$ μίας πραγματικής ρίζας ξ της εξίσωσης $f(x) = 0$, όταν είναι γνωστό ότι στο διάστημα $[a, b] = \mathbb{G}_0 \subset \mathbb{R}$ η συνάρτηση $f(x)$ αλλάζει πρόσημο μία μόνον φορά, οπότε ισχύει $f(a)\, f(b) < 0$. Σημειωτέον ότι, άν η $f(x)$ είναι μονότονη στο διάστημα $\mathbb{G}_0$, τότε η ισχύς της $f(a)\, f(b) < 0$ συνεπάγεται τη μοναδική αλλαγή προσήμου της $f(x)$ στο $\mathbb{G}_0$. Το διάστημα $\mathbb{G}_0$ αναφέρεται ως «διάστημα εγκιβωτισμού» (bracketing interval) μίας και μόνον μίας ρίζας ξ.

Η μέθοδος της διχοτόμησης συνίσταται στην αντικατάσταση του $\mathbb{G}_0$ με ένα νέο διάστημα εγκιβωτισμού, $\mathbb{G}_1$, μήκους ίσου με το ήμισυ του αρχικού, $L(\mathbb{G}_1) = L(\mathbb{G}_0)/2 = (b-a)/2$, επειδή ως ένα εκ των άκρων του επιλέγεται το μέσο $c = (a+b)/2$ που αντικαθιστά το a όταν $f(a)\, f(c) > 0$, ή το b όταν $f(b)\, f(c) > 0$. Οπότε παράγεται μία ακολουθία διαστημάτων εγκιβωτισμού που τερματίζεται στο $\mathbb{G}_n$, του οποίου το μήκος $L(\mathbb{G}_n)$ ικανοποιεί ένα συγκεκριμένο κριτήριο τερματισμού. Τότε το τελευταίο c είναι η ζητούμενη προσέγγιση $\bar{x}$ της ρίζας ξ.

2.6.2 Ο αλγόριθμος

Ο αλγόριθμος που υλοποιεί τη μέθοδο της διχοτόμησης περιγράφεται ως εξής.
Step 1. Δίδονται τα $\texttt{A} = a$, $\texttt{B} = b$, $\texttt{XPREV} = b$, η συνάρτηση $\texttt{F(X)} = f(x)$, μία σχετική ανοχή $\texttt{RE} = \tau_{\mathrm{R}}$, μία απόλυτη ανοχή $\texttt{AE} = \tau_{\mathrm{A}}$, μία ανοχή $\texttt{TOLF} = \tau_{\mathrm{F}}$ ως προς το σφάλμα μηδενισμού της $f(x)$, και ένα άνω όριο επαναλήψεων $\texttt{MAXIT} = N_{\max}$ που επιτρέπεται να πραγματοποιήσει ο αλγόριθμος.
Step 2. Ελέγχεται άν ισχύει $|f(a)| < \tau_{\mathrm{F}}$ ή $|f(b)| < \tau_{\mathrm{F}}$, οπότε $\bar{x} = a$ ή $\bar{x} = b$, αντίστοιχα, και ο αλγόριθμος τερματίζεται· στην περίπτωση αυτή, το ακέραιο flag IFLAG λαμβάνει την τιμή '2'.
Step 3. Επαναλαμβάνεται η ακόλουθη δομή DO, έως ότου ικανοποιηθεί το κριτήριο τερματισμού $|\delta x| = \texttt{ABS(X} - \texttt{XPREV)} <= \texttt{RE} * \texttt{ABS(X)} + \texttt{AE}$, οπότε το IFLAG λαμβάνει την τιμή '1', ή το κριτήριο τερματισμού $|f(x)| = \texttt{ABS(F(X))} < \mathrm{TOLF}$, οπότε το IFLAG λαμβάνει την τιμή '2'. Άν πραγματοποιηθούν MAXIT επαναλήψεις χωρίς να ικανοποιηθεί ένα από τα προηγούμενα κριτήρια, τότε ο αλγόριθμος τερματίζεται με $\texttt{IFLAG} = 3$.

```
ITERATE: DO K=1,MAXIT
    X=(A+B)/2
    IF (ABS(X-XPREV) <= RE*ABS(X)+AE) THEN
        IFLAG=1
        EXIT ITERATE
    END IF
    IF (ABS(F(X)) < TOLF) THEN
        IFLAG=2
        EXIT ITERATE
    ELSE IF (F(A)*F(X) > 0) THEN
        A=X
    ELSE
        B=X
```

```
        END IF
        IF (K == MAXIT) THEN
            IFLAG=3
            EXIT ITERATE
        END IF
        XPREV=X
    END DO ITERATE
```

2.6.3 Εφαρμογές

1. Δίδεται η συνάρτηση

$$f(x) = \sqrt{2} - 1.111x + \ln(x), \tag{2.29}$$

όπου το 'ln' παριστάνει τον φυσικό λογάριθμο. Είναι γνωστό ότι η μή γραμμική αλγεβρική εξίσωση $f(x) = 0$ έχει μία πραγματική ρίζα στο διάστημα $[1, 2]$. Να κάνετε χρήση της μεθόδου της διχοτόμησης για τον υπολογισμό της ρίζας αυτής.

2. Δίδεται η συνάρτηση

$$g(x) = 1.001 \exp(x) + 1.005\, x^2 - 0.9\, x - 4.905. \tag{2.30}$$

Είναι γνωστό ότι η μή γραμμική αλγεβρική εξίσωση $g(x) = 0$ έχει μία πραγματική ρίζα στο διάστημα $[1, 2]$. Να κάνετε χρήση της μεθόδου της διχοτόμησης για τον υπολογισμό της ρίζας αυτής.

3. Δίδεται το πολυώνυμο

$$p_7(x) = x^7 - 28x^6 + 322x^5 - 1960x^4 + 6769x^3 - 13132x^2 + 13068x - 5040, \tag{2.31}$$

Η πολυωνυμική εξίσωση $p_7(x) = 0$ έχει τις ακόλουθες πραγματικές ρίζες

$$x_1 = 1,\ x_2 = 2,\ x_3 = 3,\ x_4 = 4,\ x_5 = 5,\ x_6 = 6,\ x_7 = 7. \tag{2.32}$$

Να χρησιμοποιήσετε τη μέθοδο της διχοτόμησης για να επαληθεύσετε όλες τις ρίζες του πολυωνύμου (2.31).

2.7 Μέθοδος του Newton

2.7.1 Περιγραφή

Για την προσέγγιση $\bar{x}$ μίας ρίζας ξ της εξίσωσης $f(x) = 0$, η «μέθοδος του Newton» ή «μέθοδος των Newton–Raphson» (βλέπε π.χ. [7], §4.2· επίσης [8], §2.5), εδράζεται σε μία αρχική εκτίμηση x_0. Η μέθοδος υπολογίζει μία καλύτερη προσέγγιση της ρίζας, η οποία προκύπτει ως η ρίζα x_1 της ευθείας που άγεται εφαπτομενικά της καμπύλης $f(x)$ στο σημείο $f(x_0)$· συνεπώς έχει κλίση ίση με $f'(x_0)$. Τότε ισχύει $x_1 = x_0 - f(x)/f'(x)$. Η επανάληψη της διαδικασίας παράγει τις διαδοχικά καλύτερες προσεγγίσεις $x_2, x_3, \ldots, x_{n+1}$,

$$x_{n+1} = x_n - \frac{f(x)}{f'(x)}, \tag{2.33}$$

έως ότου το τρέχον ζεύγος x_n, x_{n+1} ικανοποιήσει ένα συγκεκριμένο κριτήριο τερματισμού. Τότε το x_{n+1} είναι η ζητούμενη προσέγγιση $\bar{x}$ της ρίζας ξ.

2.7.2 Ο αλγόριθμος

Η μέθοδος υλοποιείται με τον ακόλουθο αλγόριθμο.
Step 1. Δίδονται τα `X0` $= x_0$, οι συναρτήσεις `F(X)` $= f(x)$ και `DF(X)` $= f'(x)$, μία σχετική ανοχή `RE` $= \tau_{\rm R}$, μία απόλυτη ανοχή `AE` $= \tau_{\rm A}$, ένα άνω όριο επαναλήψεων `MAXIT` $= N_{\max}$ που επιτρέπεται να πραγματοποιήσει ο αλγόριθμος, και `TOLDF` $= \tau_{\rm D}$ μία ανοχή ως προς το πόσο επιτρέπεται να προσεγγίσει το μηδέν η $|f'(x)|$.
Step 2. Επαναλαμβάνεται η ακόλουθη δομή `DO`, έως ότου ικανοποιηθεί το κριτήριο τερματισμού $|\delta x|$ = `ABS(X1 - X0)` $<=$ `RE` $*$ `ABS(X1)` $+$ `AE`, οπότε το `IFLAG` λαμβάνει την τιμή '1', ή το κριτήριο τερματισμού $|f'(x)|$ = `ABS(DF(X))` $<$ TOLDF, οπότε το `IFLAG` λαμβάνει την τιμή '2'. Άν πραγματοποιηθούν `MAXIT` επαναλήψεις χωρίς να ικανοποιηθεί ένα από τα προηγούμενα κριτήρια, τότε ο αλγόριθμος τερματίζεται με `IFLAG` $= 3$.

```
ITERATE: DO I=1,MAXIT
    DFVAL=DF(X0)
    IF (ABS(DFVAL) < TOLDF) THEN
        X=X0
        IFLAG=2
        EXIT ITERATE
    END IF
    X1=X0-F(X0)/DFVAL
    IF (ABS(X1-X0) < RE*ABS(X1)+AE) THEN
        X=X1
        IFLAG=1
        EXIT ITERATE
    END IF
    IF (I == MAXIT) THEN
        X=X1
        IFLAG=3
        EXIT ITERATE
    END IF
    X0=X1
END DO ITERATE
```

2.7.3 Εφαρμογές

Να χρησιμοποιήσετε τη μέθοδο του Newton για τον υπολογισμό των ριζών που υποδείχθηκαν στην §2.6.3 για τις μή γραμμικές αλγεβρικές εξισώσεις $f(x) = 0$, $g(x) = 0$, και $p_7(x) = 0$.

2.8 Μέθοδος του Müller

2.8.1 Περιγραφή

Για την προσέγγιση $\bar{x}$ μίας ρίζας ξ της εξίσωσης $f(x) = 0$, η «μέθοδος του Müller» (βλέπε π.χ. [8], §3.3.2) χρησιμοποιεί τρεις αρχικές εκτιμήσεις x_0, x_1, x_2, προσδιορίζει το τριώνυμο $r_2(x - x_2)$ που διέρχεται από τα τρία σημεία $(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$, και υπολογίζει μία ρίζα του, $r_2(x - x_2) = 0$, με χρήση του τύπου (2.28). Η διαδικασία επαναλαμβάνεται με τις τιμές $x_0 = x_1$, $x_1 = x_2$, $x_2 = x$, έως ότου ικανοποιηθεί ένα συγκεκριμένο κριτήριο τερματισμού. Τότε το τελευταίο x είναι η ζητούμενη προσέγγιση $\bar{x}$ της ρίζας ξ.

Το τριώνυμο παριστάνεται εδώ ως $r_2(x) = A(x - x_2)^2 + B(x - x_2) + \Gamma$. Αφού

διέρχεται από τα σημεία $(x_0, f(x_0))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$, πρέπει να ισχύει

$$\begin{aligned} f(x_0) &= A(x_0 - x_2)^2 + B(x_0 - x_2) + \Gamma, \\ f(x_1) &= A(x_1 - x_2)^2 + B(x_1 - x_2) + \Gamma, \\ f(x_2) &= A(x_2 - x_2)^2 + B(x_2 - x_2) + \Gamma = \Gamma, \end{aligned} \tag{2.34}$$

Η λύση του συστήματος των τριών εξισώσεων (2.34) για τους τρεις αγνώστους A, B, και Γ, δίνει

$$\begin{aligned} \Gamma &= f(x_2), \\ B &= \frac{(x_0 - x_2)^2 \left[f(x_1) - f(x_2)\right] - (x_1 - x_2)^2 \left[f(x_0) - f(x_2)\right]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)}, \\ A &= \frac{(x_1 - x_2) \left[f(x_0) - f(x_2)\right] - (x_0 - x_2) \left[f(x_1) - f(x_2)\right]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)}. \end{aligned} \tag{2.35}$$

Μπορούμε να χρησιμοποιήσουμε τον τύπο (2.28) για την εύρεση μίας ρίζας του τριωνύμου r_2, επιλέγοντας το πρόσημο της ποσότητας $\Delta = \sqrt{B^2 - 4A\Gamma}$ έτσι ώστε να συμφωνεί με το πρόσημο του συντελεστή B· συγκεκριμένα, επιλέγουμε ως παρονομαστή την ποσότητα $E = B + \mathrm{sgn}(B)\,\Delta$, οπότε

$$h = x - x_2 = \frac{-2\,\Gamma}{E}, \qquad x = x_2 - \frac{2\,\Gamma}{E}. \tag{2.36}$$

Με το πρόσημο που επιλέξαμε, ο παρονομαστής E έχει τη μέγιστη δυνατή απόλυτη τιμή· συνεπώς το x απέχει την ελάχιστη δυνατή απόσταση από το x_2. Επαναλαμβάνουμε τη διαδικασία με εκτιμήσεις $x_0 = x_1$, $x_1 = x_2$, $x_2 = x$, έως ότου εκπληρωθεί ένα κριτήριο τερματισμού.

2.8.2 Ο αλγόριθμος

Η μέθοδος υλοποιείται με τον ακόλουθο αλγόριθμο.
Step 1. Δίδονται τα `X0` $= x_0$, `X1` $= x_1$, `X2` $= x_2$, η συνάρτηση `F(X)` $= f(x)$ μία σχετική ανοχή `RE` $= \tau_{\mathrm{R}}$, μία απόλυτη ανοχή `AE` $= \tau_{\mathrm{A}}$, μία ανοχή `TOLE` $= \tau_{\mathrm{E}}$ ως προς το πόσο επιτρέπεται να προσεγγίσει το μηδέν ο παρονομαστής `E`, και ένα άνω όριο επαναλήψεων `MAXIT` $= N_{\max}$ που επιτρέπεται να πραγματοποιήσει ο αλγόριθμος.
Step 2. Επαναλαμβάνεται η ακόλουθη δομή `DO`, έως ότου ικανοποιηθεί το κριτήριο τερματισμού $|h| = |\delta x| =$ `ABS(H)` $<=$ `RE` $*$ `ABS(X2)` $+$ `AE`, οπότε το `IFLAG` λαμβάνει την τιμή '1', ή το κριτήριο τερματισμού `ABS(E)` $<$ TOLE, οπότε το `IFLAG` λαμβάνει την τιμή '2'. Αν πραγματοποιηθούν `MAXIT` επαναλήψεις χωρίς να ικανοποιηθεί ένα από τα προηγούμενα κριτήρια, τότε ο αλγόριθμος τερματίζεται με `IFLAG` $= 3$.

```
ITERATE: DO I=1,MAXIT
    H1=X1-X0
    H2=X2-X1
    G=F(X2)
    D1=(F(X1)-F(X0))/H1
    D2=(G-F(X1))/H2
    D=(D2-D1)/(H2+H1)
    B=D2+H2*D
    DC=SQRT(B*B-4*G*D)
    IF (ABS(B-DC) < ABS(B+DC)) THEN
        E=B+DC
    ELSE
        E=B-DC
```

```
        END IF
        IF (ABS(E) < TOLE) THEN
            X=X2
            IFLAG=2
            EXIT ITERATE
        END IF
        H=-2*G/E
        X=X2+H
        IF (ABS(H) < RE*ABS(X2)+AE) THEN
            IFLAG=1
            EXIT ITERATE
        END IF
        IF (I == MAXIT) THEN
            IFLAG=3
            EXIT ITERATE
        END IF
        X0=X1
        X1=X2
        X2=X
    END DO ITERATE
```

2.8.3 Εφαρμογές

1. Να χρησιμοποιήσετε τη μέθοδο του Müller για τον υπολογισμό των ριζών που υποδείχθηκαν στην §2.6.3 για τις μή γραμμικές αλγεβρικές εξισώσεις $f(x) = 0$, $g(x) = 0$, και $p_7(x) = 0$.

2. Δίδεται το μιγαδικό πολυώνυμο

$$\begin{aligned} p_5(x) &= (25.632 + 0.164\,i) + (4.09 - 20.954\,i)\,x - (14.5528 + 3.4724\,i)\,x^2 \\ &- (3.4274 - 4.0758\,i)\,x^3 + (3.2074 + 1.7322\,i)\,x^4 + x^5. \end{aligned} \tag{2.37}$$

Μία και μόνον μία από τις ρίζες της εξίσωσης $p_5(x) = 0$, της μορφής

$$\omega = \sigma + \tau\,i, \tag{2.38}$$

έχει $\sigma > 0$ και $\tau > 0$. Αυτή παριστάνει 'ιδιοτιμή' κυμάτων βαρύτητας ενός αστέρα νετρονίων με 'ιδιοπερίοδο' σ και 'ίδιο' ή 'χαρακτηριστικό' χρόνο απόσβεσης τ, μετρούμενα σε κατάλληλες μονάδες. Να χρησιμοποιήσετε τη μέθοδο του Müller, σε συνεργασία με τον αλγόριθμο της πολυωνυμικής απομείωσης μέσω της συνθετικής διαίρεσης (§2.4), για να υπολογίσετε όλες τις ρίζες. Στη συνέχεια, να υποδείξετε τη ρίζα ω που έχει τη συγκεκριμένη φυσική σημασία. Γιατί, κατά τη γνώμη σας, οι άλλες τέσσερεις ρίζες στερούνται φυσικής σημασίας;

2.9 Περιεχόμενο του φακέλου Drpzero

Ο φάκελος Drpzero περιλαμβάνει το «κύριο πρόγραμμα» (main program, driver) `drv_rpzero.for`, το αρχείο `rpzero_and_dependencies.for`, το αρχείο `rpzero_and_infoENGR.for` και τον φάκελο _rpzero.

Ο φάκελος _rpzero ανήκει στην SLATEC. Περιλαμβάνει τα επεξηγηματικά έγγραφα `rpzero.txt, depends.txt`, το αρχείο `rpzero.f` με περιεχόμενο την υπορουτίνα `RPZERO`, και τον φάκελο depends.

Ο φάκελος depends περιλαμβάνει τα αρχεία `cpevl.f` με περιεχόμενο την υπορουτίνα `CPVL` και `cpzero.f` με περιεχόμενο την υπορουτίνα `CPZERO`.

2.9.1 Το αρχείο rpzero_and_dependencies.for

Το rpzero_and_dependencies.for περιλαμβάνει το αρχείο rpzero.f και τα παρελκόμενα αρχεία (dependencies) cpevl.f, cpzero.f. Το ότι τα δύο τελευταία είναι παρελκόμενα του πρώτου διευκρινίζεται στο έγγραφο depends.txt.

Η σύνθεση των τριών αρχείων σε ένα επιτυγχάνεται με τον κειμενογράφο Crimson Editor [5] . Πρώτα ανοίγουμε με τον Crimson Editor και τα τρία αρχεία συγχρόνως· στη συνέχεια, με τις διαδικασίες «αντιγραφή» (copy) και «επικόλληση» (paste) επικολλούμε το δεύτερο αρχείο στο τέλος του πρώτου και το τρίτο αρχείο στο τέλος των πρώτου–δεύτερου. Απομένει να αποθηκεύσουμε (Save As) το αρχείο που προκύπτει ως προϊόν αυτής της σύνθεσης με το όνομα rpzero_and_dependencies.for, το οποίο υποδηλώνει το περιεχόμενό του.

2.9.2 Το κύριο πρόγραμμα DRV_RPZERO

Το αρχείο drv_rpzero.for περιέχει το κύριο πρόγραμμα DRV_RPZERO. Το πρόγραμμα αυτό έχει ως σκοπό την ορθή οδήγηση της υπορουτίνας RPZERO και των παρελκομένων της και γράφεται μετά από ενδελεχή μελέτη των οδηγιών που περιέχονται στο έγγραφο rpzero.txt (στα αγγλικά) ή στο έγγραφο rpzero_and_infoENGR.for (στα αγγλικά και στα ελληνικά). Μόνον η πλήρης κατανόηση την σχετικών οδηγιών μπορεί να εγγυηθεί τη συγγραφή ενός κύριου προγράμματος χωρίς λάθη.

2.9.3 Μεταγλώττιση και σύνδεση

Η «μεταγλώττιση» (compilation, compiling) και «σύνδεση» (link, linking) των απαραίτητων αρχείων σε ένα «εκτελέσιμο πρόγραμμα» (executable program, executable file) γίνεται με τον «μεταγλωττιστή» (compiler) και «συνδέτη» (linker) Fortran G95.

Τη διαδικασία διευκολύνουν τα αρχεία δέσμης (batch files) g95local_set.bat και g95exe.bat που είναι τα αντίστοιχα των g95set.bat, και g95net.bat, τα οποία χρησιμοποιούμε στο Υπολογιστικό Κέντρο. Πρώτα εκτελούμε το αρχείο δέσμης g95local_set.bat,

```
> g95local_set
```

ώστε να γνωρίζει το λειτουργικό σύστημα σε ποιούς φακέλους βρίσκεται το υλικό που προέκυψε από την εγκατάσταση του μεταγλωττιστή Fortran G95. Στη συνέχεια, εκτελούμε το αρχείο δέσμης g95exe.bat με τα ακόλουθα ορίσματα

```
> g95exe drv_rpzero rpzero_and_dependencies.for cmlutils_mod.for
```

οπότε, καλώς εχόντων των πραγμάτων, δημιουργείται το εκτελέσιμο πρόγραμμα drv_rpzero.exe.

2.9.4 Τα αρχεία cmlutils.f και cmlutils_mod.for

Το αρχείο cmlutils.f περιλαμβάνει υπορουτίνες και συναρτήσεις, οι οποίες εμπλέκονται στη λειτουργία όλων σχεδόν των αρχείων της SLATEC. Όπως μπορούμε να διαπιστώσουμε από το έγγραφο depends.txt, τα παρελκόμενα της υπορουτίνας RPZERO που περιέχονται στο cmlutils_mod.f αναφέρονται ως "generic dependencies (routines in the file cmlutils.f)"· στη συγκεκριμένη περίπτωση, τέτοιο παρελκόμενο είναι μόνον η ακέραια συνάρτηση I1MACH. Στο ίδιο έγγραφο, οι υπορουτίνες CPVL και CPZERO αναφέρονται ως "specific dependencies (routines NOT in the file cmlutils.f)".

Αντί του πρωτότυπου (original) αρχείου `cmlutils.f` της SLATEC, χρησιμοποιούμε το τροποποιημένο (modified) αρχείο `cmlutils_mod.for`, το οποίο ισχύει για υπολογιστές με τα πιο συνηθισμένα (στις μέρες μας) λειτουργικά συστήματα. Δεδομένου ότι, όταν εργαζόμαστε με την SLATEC, το αρχείο αυτό χρειάζεται σχεδόν πάντοτε στη διαδικασία μεταγλώττισης–σύνδεσης, μπορούμε να δημιουργήσουμε τον «μεταγλωττισμένο κώδικα» (object code, object file — τα αρχεία αυτά έχουν κατάληξη '.o' ή '.obj') `cmlutils_mod.o` με χρήση της εντολής

```
> g95 -c cmlutils_mod.for
```

όπου με την επιλογή (option) `-c` καλούμε τον μεταγλωττιστή, χωρίς να καλούμε μετά από αυτόν και τον συνδέτη. Ως αποτέλεσμα, έχουμε την δημιουργία του αντίστοιχου μεταγλωττισμένου κώδικα `cmlutils_mod.o`. Μπορούμε λοιπόν να χρησιμοποιήσουμε το αρχείο αυτό στη θέση του αρχείου `cmlutils_mod.for`,

```
> g95exe drv_rpzero rpzero_and_dependencies.for cmlutils_mod.o
```

Κεφάλαιο 3

Αριθμητική Παραγώγιση

3.1 Αριθμητικός υπολογισμός παραγώγου συνάρτησης

3.1.1 Περιγραφή του βασικού αλγόριθμου

Για τον αριθμητικό υπολογισμό τής παραγώγου $f'(x)$ μίας συνάρτησης $f(x)$ σε δοσμένη τιμή τής ανεξάρτητης μεταβλητής x, μπορούμε να χρησιμοποιήσουμε έναν αλγόριθμο που προκύπτει άμεσα από τον ορισμό της παραγώγου

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{3.1}$$

Συγκεκριμένα, αφού αδυνατούμε να 'μιμηθούμε' στην πράξη την έννοια τού ορίου, προσεγγίζουμε την παράγωγο επιλέγοντας ένα μικρό «βήμα» h (δηλαδή μία τιμή τού h στη γειτονιά του μηδενός),

$$f'(x) \doteq \frac{f(x+h) - f(x)}{h} \equiv D(f(x); h), \quad h \in \Gamma(0). \tag{3.2}$$

Η ποσότητα $D(f(x); h)$ ονομάζεται «αριθμητική παράγωγος τής $f(x)$ με βήμα h». Το σύμβολο '$\doteq$' σημαίνει ότι το δεξιό μέλος τής σχέσης (3.2) είναι μία θεωρητική προσέγγιση του αριστερού μέλους (δηλαδή είναι 'θεωρητικά περίπου ίσο' με το αριστερό μέλος). Το πόσο 'μικρό' επιλέγεται το h στην πράξη, δηλαδή το πόσο ευρεία θεωρείται η γειτονιά του μηδενός $\Gamma(0)$, εξαρτάται από την τάξη μεγέθους τής φυσικής ποσότητας και της παραγώγου της που εμπλέκονται στο συγκεκριμένο πρόβλημα. Για δοσμένη τιμή $x = a$ της ανεξάρτητης μεταβλητής, η αριθμητική προσέγγιση (3.2) λαμβάνει τη μορφή

$$D(f(a); h) = \frac{f(a+h) - f(a)}{h}. \tag{3.3}$$

3.1.2 Αλγόριθμος με επαναληπτικότητα, έλεγχο αριθμητικής συμπεριφοράς, και εκτίμηση σφάλματος

Μπορούμε να μετατρέψουμε τον αλγόριθμο που περιγράψαμε στην §3.1.1 σε «επαναληπτικό αλγόριθμο», ξεκινώντας τον υπολογισμό με ένα αρχικό βήμα h_0 και, στη

συνέχεια, επαναλαμβάνοντας με υποδιπλασιασμένο βήμα $h_1 = h_0/2$ στην πρώτη επανάληψη, $h_2 = h_1/2 = h_0/4$ στην δεύτερη επανάληψη, κ.ο.κ., μέχρι ένα καθορισμένο μέγιστο πλήθος επαναλήψεων $N_{\max}$.

Επιπρόσθετα, μπορούμε να εισάγουμε έναν «έλεγχο αριθμητικής συμπεριφοράς» του αλγόριθμου με το ακόλουθο «κριτήριο αριθμητικής ευστάθειας». Φθάνοντας στη δεύτερη επανάληψη, έχουμε στη διάθεσή μας τη διαφορά $d_{(2,1)} = |D_2 - D_1|$ της τρέχουσας εκτίμησης D_2 από την προηγούμενη D_1 και την αντίστοιχη $d_{(1,0)} = |D_1 - D_0|$ της προηγούμενης εκτίμησης D_1 από την πρό-προηγούμενη D_0. Ο αλγόριθμος είναι ευσταθής άν ισχύει $d_{(1,0)} > d_{(2,1)}$, οπότε μπορούμε να συνεχίσουμε με την επόμενη επανάληψη· διαφορετικά είναι ασταθής, οπότε δεχόμαστε ως αριθμητική προσέγγιση τής παραγώγου την τιμή D_1.

Γενικά, στη n-οστή επανάληψη το κριτήριο αριθμητικής ευστάθειας έχει την ακόλουθη μορφή. Υπολογίζουμε τη διαφορά $d_{(n,n-1)} = |D_n - D_{n-1}|$ της τρέχουσας εκτίμησης D_n από την προηγούμενη D_{n-1} και την αντίστοιχη $d_{(n-1,n-2)} = |D_{n-1} - D_{n-2}|$ της προηγούμενης εκτίμησης D_{n-1} από την πρό-προηγούμενη D_{n-2}. Ο αλγόριθμος είναι ευσταθής άν ισχύει

$$d_{(n-1,n-2)} > d_{(n,n-1)}, \tag{3.4}$$

οπότε μπορούμε να συνεχίσουμε με την επόμενη επανάληψη· διαφορετικά είναι ασταθής, οπότε δεχόμαστε ως αριθμητική προσέγγιση τής παραγώγου την τιμή D_{n-1} και η διαδικασία τερματίζεται.

Ο αλγόριθμος, εφόσον παραμένει αριθμητικά ευσταθής, μπορεί να τερματισθεί όχι αναγκαστικά στις $N_{\max}$ επαναλήψεις, αλλά μόλις ικανοποιηθεί ένα κατάλληλο «κριτήριο εκτίμησης σφάλματος». Αναγκαίο είναι να έχει δοθεί μία «ανοχή σφάλματος» τ, η οποία καθορίζεται από το εκάστοτε πρόβλημα (τάξη μεγέθους συνάρτησης και παραγώγου), αλλά και από ορισμένους πρακτικούς περιορισμούς όπως είναι το «σφάλμα στρογγύλευσης τής μηχανής». Τότε η διαδικασία μπορεί να τερματισθεί στην τρέχουσα επανάληψη n, εφόσον ικανοποιούνται: (1) το κριτήριο αριθμητικής ευστάθειας (Εξ. (3.4)), και (2) το κριτήριο εκτίμησης σφάλματος

$$|d_{(n,n-1)}| < \tau. \tag{3.5}$$

Συνεπώς, με την εισαγωγή των δύο κριτηρίων, ο αλγόριθμος τερματίζεται όταν ισχύει μία από τις ακόλουθες τρεις περιπτώσεις.

Case 1. Όταν εκτελούνται οι $N_{\max}$ επαναλήψεις, με τον αλγόριθμο να παραμένει αριθμητικά ευσταθής (Εξ. (3.4)), αλλά χωρίς να έχει ικανοποιηθεί το κριτήριο εκτίμησης σφάλματος (Εξ. (4.23)).

Case 2. Όταν μετά από n επαναλήψεις, όπου $n \leq N_{\max}$, κατά τις οποίες ο αλγόριθμος παραμένει αριθμητικά ευσταθής (Εξ. (3.4)), ικανοποιείται το κριτήριο εκτίμησης σφάλματος (Εξ. (4.23)).

Case 3. Όταν μετά από n επαναλήψεις, όπου $n \leq N_{\max}$, ο αλγόριθμος καθίσταται αριθμητικά ασταθής (δηλαδή παύει να ισχύει η Εξ. (3.4)).

Ακολουθεί ένα πρόγραμμα, στο οποίο χρησιμοποιείται ο επαναληπτικός αλγόριθμος με έλεγχο αριθμητικής συμπεριφοράς και εκτίμηση σφάλματος.

```
      MODULE KINDS
      INTEGER,PARAMETER :: KD=8
      REAL(KIND=KD) :: A,B,C,D,EXACT
      END MODULE KINDS
!---------
      PROGRAM DRV_DERIV_NUM
      USE KINDS
```

```
      IMPLICIT REAL(KIND=KD) (A-H,O-Z)
      INTEGER :: REPEAT
      EXTERNAL DERIV_NUM,F
!
      LOOP: DO
          PRINT*,' A, B, C, D ? '
          READ*,A,B,C,D
          PRINT*,' TOLERANCE, ITERMAX ? '
          READ*,TOLERANCE,ITERMAX
          PRINT*,' X, H_INITIAL? '
          READ*,X,H_INITIAL
          VALUE=DERIV_NUM(F,X,H_INITIAL,TOLERANCE,ITERMAX,IFLAG,DO)
          PRINT*
          PRINT*,' NUMERICAL VALUE    = ',VALUE
          PRINT*,' EXACT VALUE        = ',EXACT
          PRINT*,' PERCENT DIFFERENCE = ',100*ABS((VALUE-EXACT)/EXACT)
          PRINT*,' IFLAG = ',IFLAG
          PRINT*
          PRINT*,' REPEAT: Y/N -> 1/0 ? '
          READ*,REPEAT
          IF (REPEAT==0) THEN
              EXIT LOOP
          ELSE
              CYCLE LOOP
          END IF
      END DO LOOP
      END PROGRAM DRV_DERIV_NUM
!---------
      FUNCTION DERIV_NUM(F,X,H_GIVEN,TOL,ITERMAX,IFLAG,DO)
      USE KINDS
      IMPLICIT REAL(KIND=KD) (A-H,O-Z)
      EXTERNAL F
!
      H=H_GIVEN
      DO=(F(X+H)-F(X))/H
      IF (ITERMAX==0) THEN
          IFLAG=1
          DERIV_NUM=DO
          RETURN
      END IF
      H=H/2
      D1=(F(X+H)-F(X))/H
      IF (ITERMAX==1) THEN
          IFLAG=1
          DERIV_NUM=D1
          RETURN
      END IF
!
      ITER=1
      DIFNOW=ABS(D1-DO)
      LOOP: DO
          ITER=ITER+1
          DOPRE=DO
          DO=D1
          DIFPRE=DIFNOW
          H=H/2
          D1=(F(X+H)-F(X))/H
          DIFNOW=ABS(D1-DO)
          IF (DIFNOW < DIFPRE) THEN
              IF (DIFNOW < TOL) THEN
                  IFLAG=1
                  DERIV_NUM=D1
```

```
                RETURN
            END IF
            IF (ITER >= ITERMAX) THEN
                IFLAG=2
                DERIV_NUM=D1
                RETURN
            END IF
            CYCLE LOOP
        ELSE
            IFLAG=0
            DERIV_NUM=D0
            D0=D0PRE
            RETURN
        END IF
    END DO LOOP
    END FUNCTION DERIV_NUM
!---------
    FUNCTION F(X)
    USE KINDS
    IMPLICIT REAL(KIND=KD) (A-H,O-Z)
    F=A+B*X+C*X**2+D*X**3
    EXACT=B+2*C*X+3*D*X**2
    END FUNCTION F
!---------
```

Το υποπρόγραμμα `FUNCTION DERIV_NUM(F,X,H0,TOL,MAXIT,IFLAG)` είναι αυτό που υλοποιεί τον αλγόριθμο. Το **πρώτο όρισμα** `F` είναι το όνομα τής συνάρτησης $f(x)$, της οποίας πρέπει να υπολογίσουμε την παράγωγο $f'(x)$ σε δοσμένη τιμή `X` (**δεύτερο όρισμα**) της ανεξάρτητης μεταβλητής x. Το **τρίτο όρισμα** `H0` είναι μία δοσμένη τιμή για το αρχικό βήμα h_0. Το **τέταρτο όρισμα** `TOL` είναι μία δοσμένη τιμή για την ανοχή σφάλματος τ. Το **πέμπτο όρισμα** `MAXIT` είναι μία δοσμένη τιμή για το μέγιστο πλήθος επαναλήψεων $N_{\max}$. Το **έκτο όρισμα** `IFLAG` είναι μία ακέραια μεταβλητή που λαμβάνει την τιμή '2' όταν η `DERIV_NUM` τερματίζεται με ισχύουσα την **Case 1**· την τιμή '1' όταν η `DERIV_NUM` τερματίζεται με ισχύουσα την **Case 2**· και, τέλος, την τιμή '0' όταν η `DERIV_NUM` τερματίζεται με ισχύουσα την **Case 3**.

3.1.3 Τύπος σφάλματος για τον αριθμητικό υπολογισμό τής παραγώγου

Από το ανάπτυγμα Taylor, με το υπόλοιπο στην f'',

$$f(a+h) = f(a) + hf'(a) + \frac{h^2}{2} f''(\xi), \quad \xi \in (a, a+h), \tag{3.6}$$

λύνοντας ως προς $f'(a)$ βρίσκουμε

$$f'(a) = \frac{f(a+h) - f(a)}{h} - \frac{h}{2} f''(\xi). \tag{3.7}$$

Από τη σύγκριση αυτής με την Εξ. (3.3), συμπεραίνουμε ότι το σφάλμα τής αριθμητικής εκτίμησης τής παραγώγου δίδεται από τον τύπο

$$E_{D(f(a);h)} = \left| -\frac{h}{2} f''(\xi) \right|, \quad \xi \in (a, a+h). \tag{3.8}$$

Άν, αντί για τον προσεγγιστικό τύπο (3.2), χρησιμοποιήσουμε τον τύπο

$$f'(a) \doteq \frac{f(a+h) - f(a-h)}{2\,h} \equiv D(f(a); \pm h) \tag{3.9}$$

τότε μπορεί να αποδειχθεί ότι το σφάλμα τής αριθμητικής προσέγγισης (3.9) είναι (βλέπε π.χ. [7], Eq. (7.72))

$$E_{D(f(a);\pm h)} = \left| -\frac{h^2}{2}\, f'''(\xi) \right|, \quad \xi \in (a-h, a+h). \tag{3.10}$$

3.2 Αριθμητικός υπολογισμός τής 2ης παραγώγου μίας συνάρτησης

Για την 2η παράγωγο $f''(x)$ μίας συνάρτησης $f(x)$ σε δοσμένη τιμή $x = a$ της ανεξάρτητης μεταβλητής, μπορούμε να χρησιμοποιήσουμε τον προσεγγιστικό τύπο (3.3) για την παράγωγο τής συνάρτησης $f'(a)$,

$$\begin{aligned} f''(a) &\doteq \frac{D(f(a+h); h) - D(f(a); h)}{h} \doteq \frac{f(a) - 2f(a+h) + f(a+2h)}{h^2} \\ &\equiv D^{(2)}(f(a); h). \end{aligned} \tag{3.11}$$

Η ποσότητα $D^{(2)}(f(a); h)$ ονομάζεται «αριθμητική 2η παράγωγος τής $f(a)$ με βήμα h». Εναλλακτικά, και σύμφωνα με την αριθμητική προσέγγιση (3.9), μπορούμε να χρησιμοποιήσουμε τον τύπο

$$f''(a) \doteq \frac{f(a-h) - 2f(a) + f(a+h)}{h^2} \equiv D^{(2)}(f(a); \pm h). \tag{3.12}$$

Μπορεί να αποδειχθεί ότι το σφάλμα τής $D^{(2)}(f(a); \pm h)$ δίδεται από τον τύπο (βλέπε π.χ. [7], Eq. (7.79))

$$E_{D^{(2)}(f(a);\pm h)} = \left| -\frac{h^2}{12}\, f^{(4)}(\xi) \right|, \quad \xi \in (a-h, a+h). \tag{3.13}$$

Κεφάλαιο 4

Αριθμητική Ολοκλήρωση

4.1 Αριθμητικός υπολογισμός ορισμένου ολοκληρώματος

Για τον αριθμητικό υπολογισμό τού ορισμένου ολοκληρώματος

$$I(f) = \int_a^b f(x)dx \tag{4.1}$$

χρησιμοποιούμε συνήθως τους ακόλουθους αλγόριθμους (κανόνες).

4.2 Κανόνας τού τραπεζίου

Η γεωμετρική αρχή τού «κανόνα τού τραπεζίου» (βλέπε π.χ. [7], §7.1· επίσης [8], §15.3.1) φαίνεται στο Σχήμα 4.1. Συγκεκριμένα, το εμβαδόν τού χωρίου που περικλείεται από το γραφικό τής συνάρτησης $f(x)$ μεταξύ των ορίων a και b της ολοκλήρωσης — το οποίο συμπίπτει με την αριθμητική τιμή τού ορισμένου ολοκληρώματος $I(f)$ (Εξ. (4.1)) — προσεγγίζεται από το εμβαδόν τού τραπεζίου με βάση $(b-a)$ και ύψη $\upsilon_a = f(a)$ και $\upsilon_b = f(b)$,

$$I(f) \doteq T(f) = \frac{1}{2}(b-a)[f(a)+f(b)]. \tag{4.2}$$

Η σχέση αυτή μπορεί να περιγραφεί ισοδύναμα ως προσέγγιση τής $f(x)$ με το «γραμμικό πολυώνυμο» (πολυώνυμο 1ου βαθμού) $p_1(x)$ που παρεμβάλλει την $f(x)$ στα σημεία a και b,

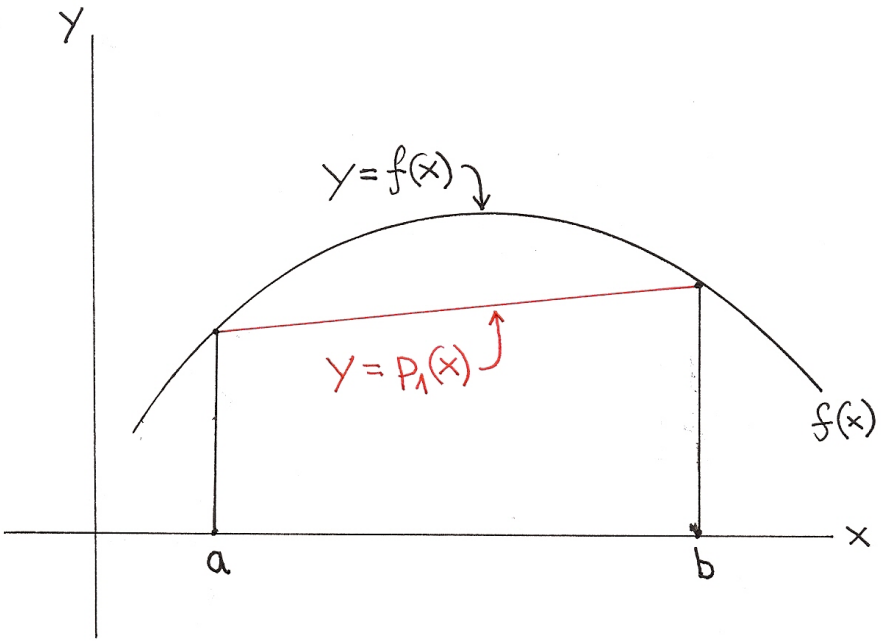
$$p_1(x) = a_0 + a_1 x, \tag{4.3}$$

οπότε

$$p_1(a) = a_0 + a_1 a = f(a), \quad p_1(b) = a_0 + a_1 b = f(b). \tag{4.4}$$

Από την επίλυση του συστήματος (4.4) προκύπτει ότι

$$a_1 = \frac{f(b)-f(a)}{b-a}, \quad a_0 = f(a) - \left[\frac{f(b)-f(a)}{b-a}\right] a \tag{4.5}$$



Σχήμα 4.1: Γεωμετρική ερμηνεία τού κανόνα τού τραπεζίου.

Συνεπώς το παρεμβάλλον γραμμικό πολυώνυμο $p_1(x)$ λαμβάνει τη μορφή

$$p_1(x) = f(a) + \left[\frac{f(b) - f(a)}{b - a}\right](x - a). \tag{4.6}$$

Το σφάλμα $E_{\rm T}$ στον κανόνα τού τραπεζίου δίνεται από τη σχέση (βλέπε π.χ. [6], Eq. (7.26))

$$E_{\rm T} = \left| -\frac{f''(\xi)(b-a)^3}{12} \right|, \quad \xi \in (a, b). \tag{4.7}$$

Ώστε το σφάλμα μπορεί να μειωθεί άν μειωθεί το μήκος του διαστήματος ολοκλήρωσης. Στην πράξη, αυτό επιτυγχάνεται με την υποδιαίρεση τού $(b - a)$ σε n υποδιαστήματα, έκαστο μήκους

$$h = \frac{b-a}{n}, \tag{4.8}$$

οριζόμενα από τα $n + 1$ σημεία (τετμημένες)

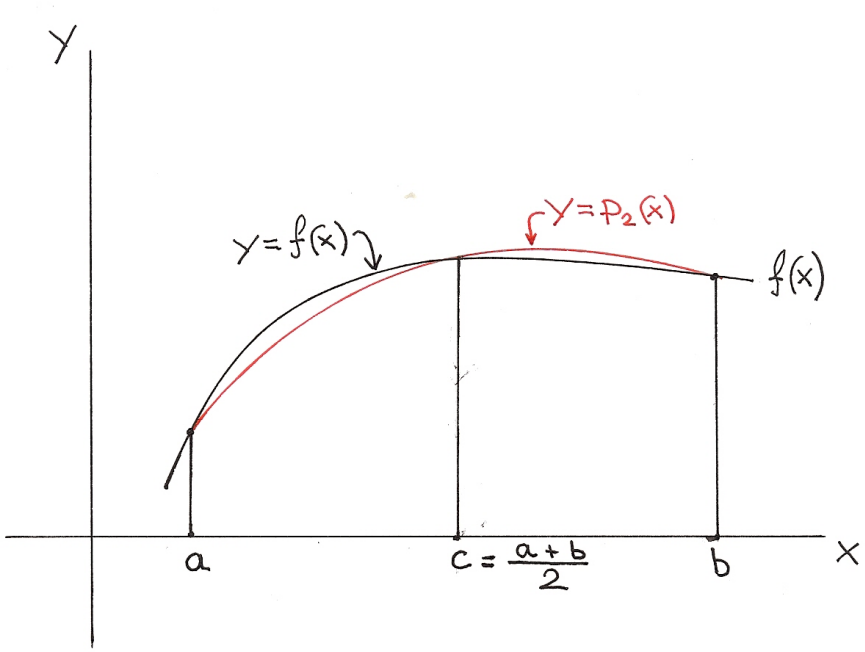
$$x_i = a + ih, \quad i = 0, 1, \ldots, n. \tag{4.9}$$

Με εφαρμογή τού κανόνα τού τραπεζίου σε κάθε ένα υποδιάστημα, αναδιάταξη των εμπλεκόμενων όρων και ορισμένες απλοποιήσεις, προκύπτει ο «σύνθετος κανόνας

τού τραπεζίου» για n υποδιαστήματα,

$$T_n(f) = h\left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{f(x_n)}{2}\right]. \tag{4.10}$$

4.3 Κανόνας τού Simpson



Σχήμα 4.2: Γεωμετρική ερμηνεία τού κανόνα τού Simpson.

Η γεωμετρική αρχή τού «κανόνα τού Simpson» (βλέπε π.χ. [7], §7.1· επίσης [8], §15.3.2) φαίνεται στο Σχήμα 4.2. Συγκεκριμένα, το εμβαδόν τού χωρίου που περικλείεται από το γραφικό τής συνάρτησης $f(x)$ μεταξύ των ορίων a και b της ολοκλήρωσης προσεγγίζεται από το εμβαδόν τού γραφικού τού πολυωνύμου 2ου βαθμού $p_2(x)$, το οποίο παρεμβάλλει την $f(x)$ στα σημεία a, $c = \frac{a+b}{2}$, και b,

$$p_2(x) = a_0 + a_1 x + a_2 x^2. \tag{4.11}$$

Οπότε ισχύουν οι σχέσεις

$$p_2(a) = a_0 + a_1 a + a_2 a^2 = f(a), \tag{4.12}$$

$$p_2(c) = a_0 + a_1 c + a_2 c^2 = f(c), \tag{4.13}$$

$$p_2(b) = a_0 + a_1 b + a_2 b^2 = f(b), \tag{4.14}$$

και οι συντελεστές a_0, a_1, a_2 είναι οι λύσεις τού συστήματος (4.12)–(4.14). Μετά από αναδιάταξη των εμπλεκόμενων όρων και ορισμένες απλοποιήσεις, βρίσκουμε

$$I(f) \doteq S(f) = \frac{h}{3}\,[f(a) + 4f(c) + f(b)]\,, \tag{4.15}$$

όπου

$$h = \frac{b-a}{2}, \tag{4.16}$$

οπότε και $h = c-a = b-c$. Ώστε για την εφαρμογή τού κανόνα τού Simpson, είναι απαραίτητο να διαιρεθεί το αρχικό διάστημα ολοκλήρωσης σε δύο υποδιαστήματα, έκαστο μήκους h (Εξ. (4.16)).

Το σφάλμα $E_{\rm S}$ στον κανόνα τού Simpson δίνεται από τη σχέση (βλέπε π.χ. [6], Eq. (7.28))

$$E_{\rm S} = \left| -\frac{f^{(4)}(\xi)[(b-a)/2]^5}{90} \right|, \quad \xi \in (a,b). \tag{4.17}$$

Ώστε το σφάλμα μπορεί να μειωθεί άν μειωθεί το μήκος του διαστήματος ολοκλήρωσης. Στην πράξη, αυτό επιτυγχάνεται με την υποδιαίρεση τού $(b-a)$ σε $n = 2m$ υποδιαστήματα (**δηλαδή για την εφαρμογή τού κανόνα τού Simpson το πλήθος n των υποδιαστημάτων πρέπει να είναι άρτιος ακέραιος**), έκαστο μήκους

$$h = \frac{b-a}{n}, \tag{4.18}$$

οριζόμενα από τα $n+1$ σημεία (τετμημένες)

$$x_i = a + ih, \quad i = 0, 1, \ldots, n, \tag{4.19}$$

οπότε $x_0 = a$ και $x_n = b$.

Η εφαρμογή τού κανόνα τού Simpson γίνεται σε κάθε ζεύγος γειτονικών υποδιαστημάτων. Συγκεκριμένα, για το πρώτο ζεύγος $[x_0, x_1]$ και $[x_1, x_2]$ τον ρόλο τού 'τοπικού' a στην Εξ. (4.15) έχει το x_0, τον ρόλο τού 'τοπικού' c το x_1, και τον ρόλο τού 'τοπικού' b το x_2. Για το δεύτερο ζεύγος $[x_2, x_3]$ και $[x_3, x_4]$ τον ρόλο τού 'τοπικού' a στην Εξ. (4.15) έχει το x_2, τον ρόλο τού 'τοπικού' c το x_3, και τον ρόλο τού 'τοπικού' b το x_4· κ.ο.κ., μέχρι το τελευταίο ζεύγος $[x_{n-2}, x_{n-1}]$ και $[x_{n-1}, x_n]$. Μετά από αναδιάταξη των εμπλεκόμενων όρων και ορισμένες απλοποιήσεις, προκύπτει ο «σύνθετος κανόνας τού Simpson» για n υποδιαστήματα,

$$S_n(f) = \frac{h}{3}\,[f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]\,. \tag{4.20}$$

4.4 Επαναληπτικότητα και εκτίμηση σφάλματος

Μπορούμε να μετατρέψουμε τούς αλγόριθμους (κανόνες) που περιγράψαμε στις §§4.2–4.3 σε επαναληπτικούς αλγόριθμους, ξεκινώντας τον υπολογισμό με αρχικό πλήθος υποδιαστημάτων $n = 2$ και, στη συνέχεια, επαναλαμβάνοντας με διπλασιασμό τού πλήθους τής προηγούμενης επανάληψης,

$$n_j = 2^j, \quad j = 1, 2, \ldots, J_{\max}, \tag{4.21}$$

όπου $J_{\max}$ το μέγιστο επιτρεπτό πλήθος επαναλήψεων. Το αντίστοιχο μήκος h των υποδιαστημάτων είναι

$$h_j = \frac{b-a}{n_j} = \frac{b-a}{2^j} = \frac{x_n - x_0}{2^j}, \quad j = 1, 2, \ldots, J_{\max}. \tag{4.22}$$

Επιπρόσθετα, μπορούμε να εισάγουμε ένα «κριτήριο εκτίμησης σφάλματος», με «ανοχή σφάλματος» τ, η οποία καθορίζεται από το εκάστοτε πρόβλημα (τάξη μεγέθους συνάρτησης και παραγώγου), αλλά και από ορισμένους πρακτικούς περιορισμούς όπως είναι το «σφάλμα στρογγύλευσης τής μηχανής». Τότε η διαδικασία μπορεί να τερματισθεί στην τρέχουσα επανάληψη j, εφόσον ικανοποιείται το κριτήριο εκτίμησης σφάλματος

$$|d_{(j,j-1)}| = |D_j - D_{j-1}| < \tau, \tag{4.23}$$

όπου D_j είναι η τρέχουσα εκτίμηση τού σύνθετου κανόνα τού τραπεζίου, $D_j = T_{n_j}$, ή του σύνθετου κανόνα τού Simpson, $D_j = S_{n_j}$, και D_{j-1} η αντίστοιχη προηγούμενη εκτίμηση. Ώστε ο αλγόριθμος μπορεί να τερματισθεί όχι αναγκαστικά μετά από τις $J_{\max}$ επαναλήψεις, αλλά μόλις ικανοποιηθεί το κριτήριο (4.23).

Συνεπώς, ο αλγόριθμος τερματίζεται όταν ισχύει μία από τις ακόλουθες δύο περιπτώσεις.

Case 1. Όταν εκτελούνται οι $J_{\max}$ επαναλήψεις χωρίς να έχει ικανοποιηθεί το κριτήριο εκτίμησης σφάλματος (Εξ. (4.23)).

Case 2. Όταν μετά από j επαναλήψεις, όπου $j \leq J_{\max}$, ικανοποιείται το κριτήριο εκτίμησης σφάλματος (Εξ. (4.23)).

4.5 Μέθοδος του Gauss

Έστω ότι διαθέτουμε έναν κανόνα ολοκλήρωσης αρκούντως ακριβή για πολυώνυμα $p_k(x)$ βαθμού $k \lesssim 10$· τότε ο ίδιος κανόνας μπορεί να χρησιμοποιηθεί και για άλλες συναρτήσεις $f(x)$, εφόσον αυτές προσεγγίζονται ικανοποιητικά από τα πολυώνυμα $p_k(x)$. Συγκεκριμένα, ο κανόνας έχει τη γενική μορφή

$$I(f) \doteq I_n(f) = \sum_{j=1}^{n} w_j f(x_j) \tag{4.24}$$

όπου οι «κόμβοι» (σημεία, τετμημένες) $x_1, \ldots, x_n$ και τα «βάρη» $w_1, \ldots, w_n$ είναι λύσεις ενός συστήματος αλγεβρικών εξισώσεων. Το σύστημα αυτό επιλύεται άπαξ.

Στην πράξη, χρησιμοποιείται συνήθως η «μέθοδος Gauss των οκτώ σημείων» (για τα αντίστοιχα σημεία και βάρη, βλέπε π.χ. [7], Table 7.6). Η μαθηματική βιβλιοθήκη SLATEC διαθέτει την υπορουτίνα `DGAUS8`, **διπλής ακρίβειας**, με επικεφαλίδα

```
SUBROUTINE DGAUS8(FUN,A,B,ERR,ANS,IERR)
```

η οποία εφαρμόζει αυτόν τον κανόνα. `FUN` είναι η προς ολοκλήρωση συνάρτηση, `A` το αριστερό όριο ολοκλήρωσης, `B` το δεξιό όριο ολοκλήρωσης, `ERR` κατάλληλη ανοχή σφάλματος, `ANS` η υπολογιζόμενη τιμή τού ορισμένου ολοκληρώματος $I(f)$ (Εξ. (4.1)), `IERR` ακέραια μεταβλητή που λαμβάνει την τιμή '1' όταν (πιθανότατα!) έχει ικανοποιηθεί το κριτήριο του σφάλματος και '2' όταν αυτό δεν έχει ικανοποιηθεί.

Κεφάλαιο 5

Αριθμητική Επίλυση Συνήθων Διαφορικών Εξισώσεων

5.1 Προβλήματα αρχικών τιμών

Μία «συνήθης διαφορική εξίσωση» (ΣΔΕ) (ordinary differential equation, ODE) πρώτης τάξης εκφράζεται ως εξίσωση που συσχετίζει την παράγωγο y' της συνάρτησης $y(x)$ με μία συνάρτηση $f(x, y)$ της ανεξάρτητης μεταβλητής x και της εξηρτημένης μεταβλητής y,

$$y' = f(x, y), \tag{5.1}$$

όπου, στην απλούστερη περίπτωση, η f μπορεί να είναι της μορφής $f = f(x)$ ή $f = f(y)$.

Στα «προβλήματα αρχικών τιμών» (ΠΑΤ) (initial value problems, IVP), ζητείται η επίλυση της ΣΔΕ (5.1) στο «διάστημα ολοκλήρωσης» (που λέγεται και «διάστημα επίλυσης»)

$$x \in I_{\text{ivp}} = [x_0 = a, x_{\text{up}} = b] \tag{5.2}$$

όταν είναι γνωστή η τιμή τής y στο αρχικό σημείο $x_0 = a$,

$$y(x_0) = y_0. \tag{5.3}$$

Για την αριθμητική επίλυση των ΣΔΕ χρησιμοποιούμε συνήθως τους ακόλουθους αλγόριθμους (μεθόδους).

Βιβλιογραφία

[1] Γερογιάννη, Β. Σ., *Η Γλώσσα Προγραμματισμού Fortran*, Σημειώσεις Πανεπιστημίου Πατρών (2012).

[2] Whitlock, S. T, and Boman, E., *FORTRAN, An Introductory Course*, Stanford University (1996). Copyright (C) 1996, 1995 by Stanford University. All rights reserved.

[3] http://www.g95.org; licensed under the GNU General Public License, http://www.gnu.org/licenses/gpl.html.

[4] http://netlib.cs.utk.edu.

[5] http://www.crimsoneditor.com, Release 3.70, Freeware. Copyright (C) Ingyu Kang. All rights reserved.

[6] Conte, S. D., and De Boor, C., *Elementary Numerical Analysis*, McGraw-Hill International Book Company, London (1981).

[7] Atkinson, K., *Elementary Numerical Analysis*, John Wiley & Sons, New York (1985).

[8] Engeln-Müllges, G., and Uhlig, F., *Numerical Algorithms with Fortran*, Springer-Verlag, Berlin Haidelberg (1996).