Οι εντολές COMMON και PARAMETER

- Οι εντολές αυτές είναι μή εκτελέσιμες και δεν είναι απαραίτητες σε διάφορα προγράμματα.
- Η ανάγκη τους όμως παρουσιάζεται σε μεγάλα και πολύπλοκα προγράμματα. Ιδιαίτερα η εντολή COMMON εμφανίζεται κυρίως σε προγράμματα πολλών subroutines όπου πληροφορίες πρέπει να περάσουν από ένα υποπρόγραμμα σε άλλο.
- Οι κανόνες γραφής και χρήσης τους διαφέρουν αρκετά από υπολογιστή σε υπολογιστή. Ωστόσο είναι πολύ χρήσιμες

COMMON και COMMON blocks

- Οι περισσότερες γλώσσες προγραμματισμού υποστηρίζουν τις λεγόμενες γενικές μεταβλητές (**global variables**). Μια μεταβλητή θεωρείται γενική όταν μπορεί να χρησιμοποιηθεί από διάφορα τμήματα του προγράμματος (main program, functions, subroutines). Είναι σημαντικές για να περάσουν πληροφορίες από μια μονάδα του προγράμματος σε άλλη. Ότι έχουμε δει μέχρι τώρα δεν είναι γενικές μεταβλητές. Ορίζονταν μόνο σε ένα μέρος του προγράμματος.

- COMMON blocks δίνουν την ευκαιρία να ορίσουμε αυτές τις γενικές μεταβλητές διευκολύνοντας έτσι το μοίρασμα της ίδιας πληροφορίας με πολλές μονάδες του προγράμματος χωρίς να χρειάζεται να τις περάσουμε σα παραμέτρους των διαφόρων functions ή subroutines. Το πλεονέκτημα είναι ότι δεν χρειάζεται να γράφουμε πολλές παραμέτρους αν θέλουμε να μοιράζονται με διάφορα μονάδες του προγράμματος.

- Το COMMON block μας δίνει μεγάλη ευκολία αλλά πρέπει να χρησιμοποιηθεί με πολύ μεγάλη προσοχή. Αυτό γιατί με την απουσία των γενικών μεταβλητών μπορούμε να ξέρουμε τη σχέση μιας μονάδας του προγράμματος με το υπόλοιπο πρόγραμμα. Όλες οι μεταβλητές περνάνε σαν παράμετροι και ξέρουμε τι κάνουν. Με τις γενικές μεταβλητές οι τιμές τους μπορούν να αλλάξουν σε κάποια μονάδα του προγράμματος και μετά να περάσουν σε κάποια άλλη.

Common blocks

- Ένα COMMON block είναι μια ομάδα γενικών μεταβλητών που μπορεί να μοιραστεί μεταξύ δύο τουλάχιστον μονάδων του προγράμματος Ονομάζεται block γιατί όλες οι μεταβλητές που ανήκουν στο COMMON block αποθηκεύονται στην ίδια περιοχή μνήμης του υπολογιστή.
- Υπάρχουν πολλοί τρόποι για να ορίσουμε ένα COMMON block.
- Ο ακόλουθος τρόπος είναι καλό να ακολουθείται για αποφυγή λαθών:
- Για να δημιουργήσουμε ένα COMMON block πρέπει πρώτα να δηλώσουμε όλες τις μεταβλητές που ανήκουν σ' αυτό: π.χ. REAL A(5,3)
- Μετά πρέπει να δημιουργήσουμε ένα COMMON block με το να δώσουμε την εντολή COMMON ακολουθούμενη από το όνομα που θέλουμε να δώσουμε στο block και κατόπιν τη λίστα όλων των μεταβλητών που ανήκουν στο block. Αυτός είναι ο ορισμός του COMMON block. (Πρέπει να υπάρχει σε όλες τις μονάδες του προγράμματος που χρησιμοποιούν τις μεταβλητές του COMMON block).

```
COMMON / MATRIX / A
```

- Το COMMON block πετυχαίνει το ακόλουθο: Παρόλο που το όνομα της μεταβλητής Α εμφανίζεται σε διαφορετικά σημεία του προγράμματος ωστόσο είναι αποθηκευμένη σε μια μόνο περιοχή μνήμης και μοιράζεται σε όλα τα υποπρογράμματα. Κάθε μονάδα του προγράμματος που περιέχει το COMMON/MATRIX θα έχει πρόσβαση στη μεταβλητή Α.
- Το όνομα πρέπει να είναι διαφορετικό από όλα τα υπόλοιπα COMMON blocks του προγράμματος και ονόματα υποπρογραμμάτων.

Παράδειγμα

```
C This subroutine prints out the global array
      SUBROUTINE PRNT
      REAL A(5,5)
      COMMON /MATRIX/ A
      INTEGER I, J
      DO I = 1, 5
        WRITE (*,*) (A(I,J), J=1,5)
      ENDDO
      END
```

```
C This subroutine computes the square
C root of every element in the global array.
      SUBROUTINE ROOT
      REAL A(5,5)
      COMMON /MATRIX/ A
      INTEGER I, J
      DO I = 1, 5
        DO J = 1, 5
          A(I,J) = SQRT(A(I,J))
        ENDDO
      ENDDO
      CALL PRNT
      END
```

```
C The main program initializes the global array
      PROGRAM COMM
      REAL A(5,5)
      COMMON /MATRIX/ A
      INTEGER I, J
      DO I = 1, 5
        DO J = 1, 5
          A(I,J) = I*(J+2.0)
        ENDDO
      ENDDO
      CALL ROOT
      STOP
      END
```

Κανόνες για COMMON blocks – INCLUDE files

- Χρησιμοποιείται πάντα το ίδιο όνομα μεταβλητών στον ορισμό του COMMON block στις διάφορες μονάδες του προγράμματος.
- Χρησιμοποιείται πάντα τον ίδιο τύπο μεταβλητών που ανήκουν στο COMMON block. (π.χ. όχι κάποτε ορισμένες σα REAL και άλλοτε διαφορετικού τύπου)
- Η λίστα των μεταβλητών που βρίσκονται στον ορισμό του COMMON block πρέπει να δίνεται πάντοτε με την ίδια σειρά στα διάφορα υποπρογράμματα που χρησιμοποιούν το COMMON block και να είναι του ίδιου αριθμού κάθε φορά.

  π.χ. REAL A(5)

  COMMON / MATRIX / A

  Σε άλλο υποπρόγραμμα δεν μπορούμε να ορίσουμε

  REAL A(10)

  REAL X

  COMMON /MATRIX/A ← Λάθος (στο υποπρόγραμμα ο Α έχει μέγεθος 10 όχι 5)

  COMMON /MATRIX/A,X ← Λάθος (η Χ δεν υπάρχει στο αρχικό common)
- Μη ακολουθία των παραπάνω κανόνων προϋποθέτει γνώση του τρόπου με τον οποίο η FORTRAN οργανώνει την μνήμη... και τις περισσότερες φορές σε λάθη.
- Είναι πολλές φορές χρήσιμο να γράφουμε τις μεταβλητές και τον ορισμό του COMMMON block σε ένα αρχείο με την ονομασία <onoma>.INC και να το συμπεριλαμβάνουμε στην αρχή κάθε μονάδας που χρησιμοποιεί το COMMON block

PROGRAM COMM
INCLUDE 'MTRX.INC'

Το αρχείο MTRX.INC περιέχει

REAL A(5)
COMMON/ MATRX/ A

Η εντολή PARAMETER

- Η εντολή αυτή μας δίνει ένα τρόπο με τον οποίο μπορούμε να αντιστοιχήσουμε το όνομα μιας παραμέτρου του προγράμματος με μια σταθερά
- Είδαμε σε κάποια προβλήματα ότι χρειαζόμασταν την τιμή του p (p=3.141592654) το οποίο το χρησιμοποιήσαμε αρκετές φορές στο πρόγραμμα. Αντί να πληκτρολογούμε κάθε φορά το νούμερο μπορούμε να αντιστοιχήσουμε μια φορά μόνο τη τιμή στη μεταβλητή PI και κατόπιν να αναφερόμαστε σ' αυτό με το όνομα PI.
- Ωστόσο αν το PI ήταν μόνο μια μεταβλητή θα μπορούσαμε να κάνουμε πράξεις με αυτό και ακόμα να αλλάξουμε την τιμή του μέσω μιας εντολής ανάθεσης όπως κάνουμε με όλες τις μεταβλητές Δηλαδή

```
REAL A, B, PI
PI = 3.141592654
:
:
PI = 4*A/B
:
:
```

Η τιμή του PI αλλάζει. Αυτό μπορεί να προκλήθηκε και τυχαία ενώ θέλαμε να γράψουμε LI=4*A/B (P και L είναι κοντά στο πληκτρολόγιο).
Αυτό δεν βρίσκεται από το compiler.
Η Fortran μας βοηθά με τη χρήση της εντολής PARAMETER που πρέπει να ακολουθεί μια δήλωση τύπου μεταβλητής

Η εντολή PARAMETER – Τρόπος χρήσης

```
REAL PI, C, CHARGE
PARAMETER(PI=3.14159254, C=2.997925)
PARAMETER(CHARGE = 1.6021917)
```

- Ποιο το πλεονέκτημα της εντολής PARAMETER?

  Δεν μπορείς να αλλάξεις αργότερα μέσα στο πρόγραμμα την τιμή της παραμέτρου με μια εντολή ανάθεσης. Δηλαδή η τιμή για τις μεταβλητές PI, C και CHARGE είναι πλέον αμετάβλητες στο πρόγραμμα και ο compiler θα δηλώσει κάποιο μήνυμα αν συναντήσει κάποια εντολή που προσπαθεί να αλλάξει κάποια από τις μεταβλητές που δηλώθηκαν σαν σταθερές PARAMETER
- Μια εντολή PARAMETER μπορεί να περιέχει και μια αριθμητική έκφραση, ώστε κάποιες απλές αριθμητικές πράξεις μπορούν να χρησιμοποιηθούν όταν ορίζουμε τη τιμή μια σταθεράς PARAMETER, π.χ.

```
REAL PI, TWOPI, RADIAN
PARAMETER(PI=3.14159254, TWOPI=2.*PI, RADIAN=360./PI)
```

- Η εντολή είναι χρήσιμη για να δηλώνουμε το μέγεθος ενός πίνακα π.χ.

```
INTEGER NDIM
PARAMETER (NDIM=100)
REAL     MATRX(NDIM)
```

Η εντολή DATA

- Η εντολή DATA εξυπηρετεί τις περιπτώσεις που θέλουμε να καθορίσουμε τιμές μεταβλητών χωρίς ο υπολογιστής να διαβάσει κάποια στοιχεία από κάποιο αρχείο ή το πληκτρολόγιο.
- Η γενική μορφή της εντολής είναι:

  DATA M / Σ /

  όπου M είναι της μορφής $m_1, m_2, \ldots, m_n$ μεταβλητές

  και Σ είναι της μορφής $\sigma_1, \sigma_2, \ldots, \sigma_n$ σταθερές

  Η σημασία είναι: δώσε στην μεταβλητή m_1 την τιμή σ_1 κ.ο.κ

- Το πλήθος μεταβλητών (που χωρίζονται με κόμμα) θα πρέπει να είναι ίσο με το πλήθος των σταθερών που επίσης χωρίζονται με κόμμα.

Παράδειγμα:

```
DATA AX, AY, AZ / 10.0, -2.0, 3.1/
```

έχει σαν αποτέλεσμα η μεταβλητή AX=10.0, η AY=-2.0 και η AZ=3.1

```
REAL X(5)
DATA X/1.1, 2.1, 3.1, 4.1, 5.1/
```

Στην περίπτωση αυτή γεμίζουμε τα στοιχεία ενός πίνακα X(5) με τις τιμές ώστε μετά θα έχουμε X(1)=1.1, X(2)=2.1, ... X(5)=5.5

```
DATA B,C,D,E/3.1,3*2.5/
```

Η B=3.1 και C=D=E=2.5 (3*2.5 δηλώνει 3 φορές 2.5)

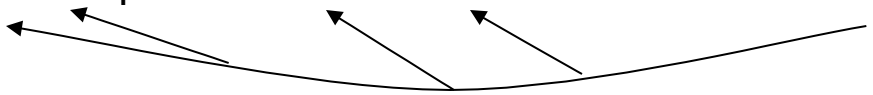
Λογικές μεταβλητές

- Πολύ συχνά συναντούμε περιπτώσεις που χρειαζόμαστε δυαδική πληροφορία της μορφής ON/OFF, TRUE/FALSE, YES/NO και στις περιπτώσεις αυτές χρησιμοποιούμε τις μεταβλητές τύπου LOGICAL

  LOGICAL FLAG

Οι λογικές μεταβλητές παίρνουν μόνο 2 τιμές:

FLAG=.TRUE. ή FLAG=.FALSE.



Προσοχή στις (.)

Έχουμε ήδη δει τις λογικές μεταβλητές από τις εντολές IF: IF (A.EQ.B) THEN
Η λογική έκφραση (A.EQ.B) επιστρέφει μια τιμή η οποία είναι TRUE/FALSE και η οποία προσδιορίζει τη ροή που θα ακολουθηθεί. Α η ποσότητα είναι TRUE τότε (THEN) εκτελούμε το επόμενη εντολή, διαφορετικά (ELSE) κάνουμε κάτι άλλο.
Κατά το ίδιο τρόπο τα ακόλουθα ισχύουν:

```
LOGICAL   FLAG
FLAG = .TRUE.
:
IF (FLAG) THEN
:
ELSE
:
ENDIF
```

Μαθηματικές μέθοδοι – Βασικές ιδέες

- Οι μαθηματικές μέθοδοι χρησιμοποιούν αριθμούς για την προσομοίωση μαθηματικών διεργασιών που με την σειρά τους συνήθως χρησιμοποιούνται για να προσομοιώσουν καταστάσεις που συναντιόνται στη καθημερινή ζωή.
- Η επιλογή μιας εξίσωσης ή αλγόριθμου για την επίλυση ενός προβλήματος επηρεάζει όχι μόνο τους υπολογισμούς αλλά και την κατανόηση των αποτελεσμάτων που θα πάρουμε.
- Οι βασικές ιδέες που προσπαθούν να εφαρμόσουν και να επιλύσουν:

- Εύρεση ενός πλήθους μεθόδων και σχολιασμός για την εφαρμογή τους σε γενικές περιπτώσεις
- Μελέτη οικογενειών όμοιων προβλημάτων και σύνδεση της μιας οικογένειας με άλλη αποφεύγοντας μεμονωμένους αλγόριθμους ή εξισώσεις
- Υπολογισμός ή αποφυγή σφάλματος στρογγυλοποίησης.

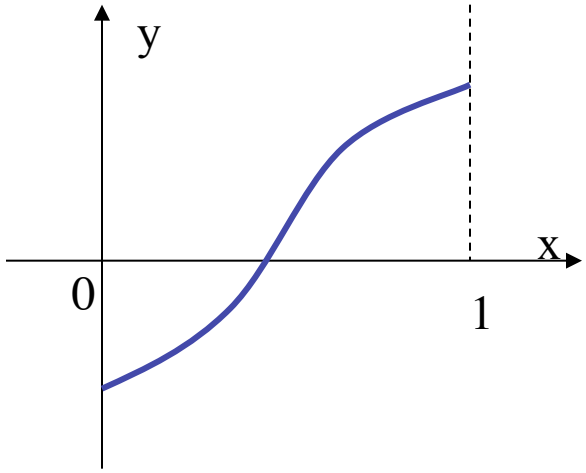
  Η μεγαλύτερη απώλεια σημαντικότητας σε νούμερα συμβαίνει όταν 2 αριθμοί περίπου ίδιου μεγέθους αφαιρούνται με αποτέλεσμα τα περισσότερα από τα αρχικά σημαντικά ψηφία αναιρούνται.
- Υπολογισμός σφάλματος αποκοπής: Ο υπολογιστής έχει πεπερασμένη ταχύτητα και μπορεί να κάνει πεπερασμένο αριθμό πράξεων σε ορισμένο χρονικό διάστημα
- Ανατροφοδότηση (feedback) – σταθερότητα

Σφάλματα υπολογισμών

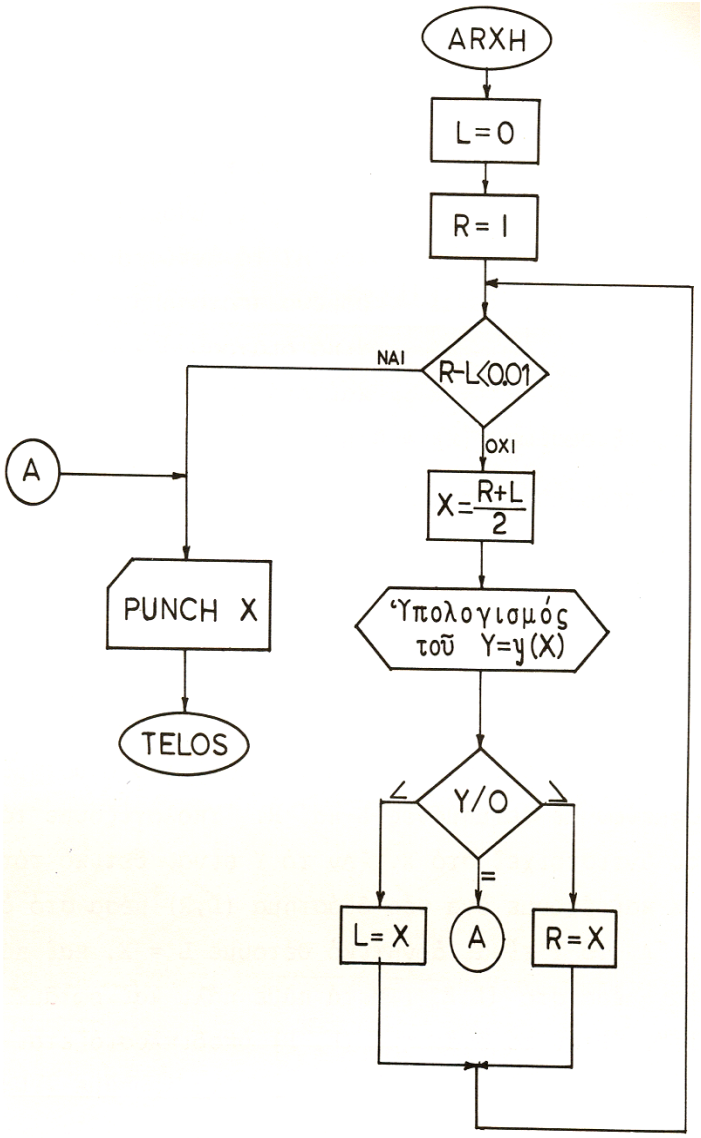
- Ανθρώπινα λάθη
  - Πολλές φορές σφάλματα ερμηνείας ή τυπογραφικά λάθη
- Τυχαία σφάλματα
  - Φυσικά φαινόμενα (κοσμική ακτινοβολία, διακοπή/διαταραχή ρεύματος)
- Σφάλμα προσέγγισης και αποκοπής
- Σφάλμα λόγω υπερχείλισης ή υποχείλισης
- Σφάλμα στρογγυλοποίησης

Εύρεση ριζών – Bisection Method

- Η γραφική παράσταση του παρακάτω σχήματος μιας μονότονης συναρτήσεως $y(x)$ κόβει τον άξονα των x σ' ένα σημείο μεταξύ $x=0$ και $x=1$ Η αναλυτική εξάρτηση του y από το x δεν είναι γνωστή.
  Για κάθε x όμως στο διάστημα $0 \le x \le 1$ μπορούμε να βρούμε με τον Η/Υ το αντίστοιχο y με ένα δεδομένο πρόγραμμα.
  Θέλουμε το πρόγραμμα που να βρίσκει τη ρίζα x_0 της εξίσωσης $y(x) = 0$ με προσέγγιση 0.01.



Εύρεση ριζών – Bisection Method



Στην αρχή ορίζουμε 2 αριθμούς $L=0$ και $R=1$ (όρια)
Οι L και R θα αλλάξουν πολλές τιμές αλλά πάντα

$$L \le x_0 \le R$$

Ρωτάμε αν η διαφορά $R-L \le 0.01$ (προσέγγιση)
Την πρώτη φορά η διαφορά είναι 1.

Προχωρούμε προς τα κάτω και ορίζουμε σα x το μέσο του διαστήματος μεταξύ των σημείων με τετμημένες L και R.

Υπολογίζουμε το y (με το Η/Υ) που αντιστοιχεί στο x.

Αν το $y > 0$, θέτουμε $R=x$ και έχουμε ένα νέο διάστημα (L,R) μέσα στο οποίο είναι το x_0.

Αν το $y < 0$ θέτουμε $L=x$. Πάλι x_0 είναι στο διάστημα (L,R). Ρωτάμε $R-L < 0.01$?

Έτσι υποδιπλασιάζουμε το διάστημα (L,R) συνεχώς έως $R-L<0.01$ οπότε και κρατάμε την τελευταία τιμή του x για τον οποίο $y=0$.

Bisection program

Τμήμα κώδικα που εφαρμόζει τη μέθοδο της διχοτόμησης φαίνεται παρακάτω

```
Read*, low_x
Read*, up_x
Half = (up_x – low_x) / 2
Do while (half .gt. Epsi)
    mid_x = (up_x + low_x)/2
    if (funct(mid_x) * funct(low_x) .lt. 0) then
      up_x = mid_x
    else
      low_x = mid_x
    endif
    half = (up_x – low_x)/2
Enddo
```

Σύγκλιση της μεθόδου:

Αν το αρχικό διάστημα είναι ε_i το σφάλμα θα είναι $\varepsilon_i/2$.
Μετά από κάθε επανάληψη το αρχικό σφάλμα διαιρείται με 2 οπότε
μετά από N επαναλήψεις το σφάλμα είναι $\varepsilon_f = \varepsilon_i/2^N$ όπου ε_f η επιθυμητή ακρίβεια

Επομένως ο αριθμός των βημάτων για την επίτευξη της επιθυμητής ακρίβειας είναι

$$N = \log\left(\varepsilon_i/\varepsilon_f\right)/\log(2)$$

Εύρεση ριζών – Newton's Method (Newton-Rhapson)

Μέθοδος εύρεσης λύσης μη γραμμικής εξίσωσης.

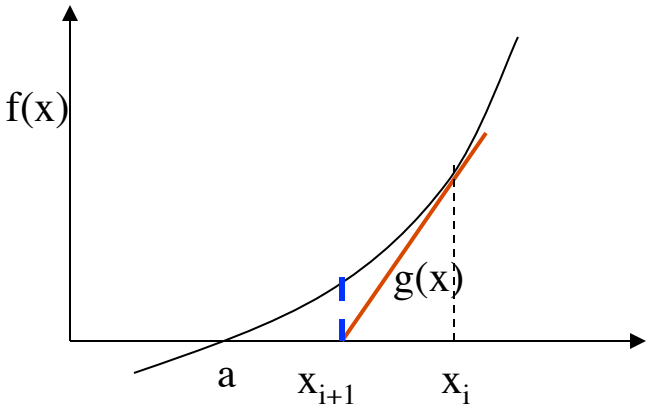
Συγκλίνει αν η αρχική προσεγγιστική τιμή για την ρίζα είναι κοντά στην πραγματική τιμή.

Η σύγκλιση είναι ανάλογη του τετραγώνου του αρχικού σφάλματος $\varepsilon_i = x - x_i$

Γρηγορότερη από τη μέθοδο της διχοτόμησης

Μειονέκτημα ----> Πρέπει να υπολογισθεί η παράγωγος της μη γραμμικής συνάρτησης $g(x)=f'(x)$

Η μέθοδος στηρίζεται στην χρησιμοποίηση των παραγώγων $f'(x)$ της συνάρτησης $f(x)$ για ταχύτερη σύγκλιση στην εύρεση των ριζών της $f(x)=0$



Η βασική ιδέα: Μια συνεχής συνάρτηση, παραγωγίσιμη συνάρτηση $f(x)$ που έχει συνεχή δεύτερη παράγωγο, μπορεί να αναπτυχθεί κατά Taylor ως προς ένα σημείο x_n που είναι κοντά στη ρίζα. Έστω ότι η πραγματική ρίζα αυτή είναι α

$$f(a) = f(x_n) + (a - x_n) f'(x_n) + (a - x_n)^2 \frac{f''(x_n)}{2!} + \cdots$$

Εφόσον x=α είναι ρίζα της f, τότε f(α) = 0 οπότε από τους 2 πρώτους όρους του αναπτύγματος έχουμε:

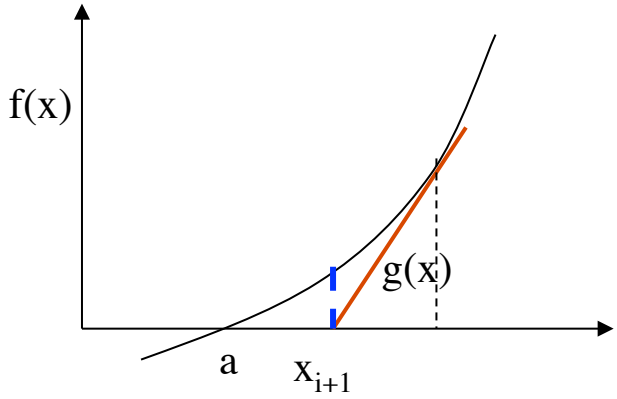
$$0 = f(a) = f(x_n) + (a - x_n) f'(x_n) \quad \Rightarrow a = x_n - \frac{f(x_n)}{f'(x_n)}$$

Εύρεση ριζών – Newton's Method

Επομένως χρειαζόμαστε f(x) και f'(x) για την μέθοδο.

Κάθε επανάληψη της μεθόδου βρίσκει: $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$

Σφάλμα: $x_{k+1} - a = x_k - a - \dfrac{f(x_k)}{f'(x_k)} \Rightarrow \varepsilon_{k+1} = \varepsilon_k + \dfrac{f(x_k)}{f'(x_k)}$ (A)



Χρησιμοποιώντας τους 3 πρώτους όρους από Taylor:

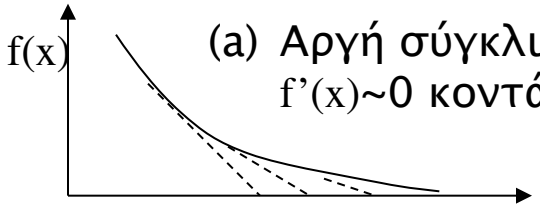
$$0 = f(a) = f(x_k) + (a - x_k) f'(x_k) + (a - x_k)^2 \frac{f''(x_k)}{2}$$

$$\Rightarrow f(x_k) = -(a - x_k) f'(x_k) - (a - x_k)^2 \frac{f''(x_k)}{2}$$

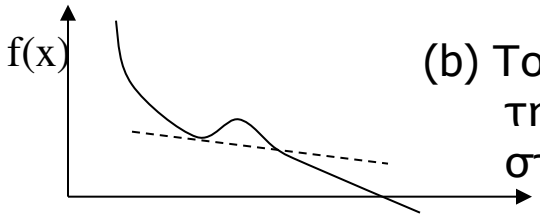
$$\Rightarrow f(x_k) = -\varepsilon_k f'(x_k) - \varepsilon_k^2 \frac{f''(x_k)}{2}$$ Αντικαθιστώντας στην (A)

Ρυθμός σύγκλισης: $\varepsilon_{k+1} = -\dfrac{\varepsilon_k^2 f''(x_k)}{2 f'(x_k)}$

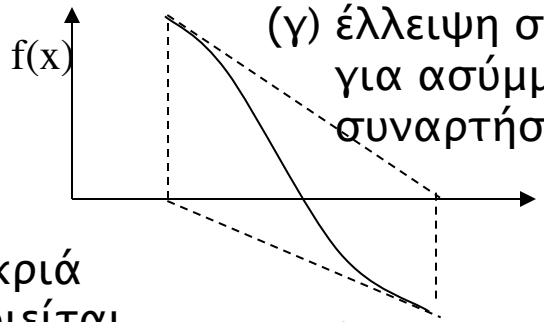
Προβλήματα της μεθόδου:



(a) Αργή σύγκλιση προς τη ρίζα όταν f'(x)~0 κοντά στη ρίζα



(b) Τοπικό ακρότατο στέλνει μακριά την τιμή x_{k+1} που χρησιμοποιείται στην επόμενη επανάληψη



(γ) έλλειψη σύγκλισης για ασύμμετρες συναρτήσεις

$f(x + a) = -f(x - a)$

Μέθοδος Newton's - πρόγραμμα

```
Error = func(x) / deriv(x)
Do while (abs(error) .gt. Epsi)
    x = x – error
    error = func(x) / deriv(x)      ! Οπου θα πρέπει να οριστούν οι συναρτήσεις deriv και func
enddo
```