Τμημα Μηχανικων Μουσικης Τεχνολογιας και Ακουστικης Τ.Ε., Σχολη Εφαρμοσμενων Επιστημων, Τεχνολογικο Εκπαιδευτικο Ιδρυμα Κρητης

Σημειώσεις στα πλαίσια του μαθήματος

ΤΑ-Π001–Εφαρμοσμένα Μαθηματικά

Στοιχεία από τα εφαρμοσμένα μαθηματικά αξιοποιώντας την Climax στο περιβάλλον του SDE.

Χ. Παναγιωτόπουλος

Χειμερινό εξάμηνο ακαδημαϊκού έτους 2018-2019

Περιεχόμενα

Πρόλογος

Το παρόν τεύχος γράφτηκε κατά το χειμερινό εξάμηνο του ακαδημαϊκού έτους 2018–2019 οπότε και μου είχε ανατεθεί η αυτόνομη διδασκαλία και γενικότερη διεξαγωγή του μαθήματος «ΤΠΑ-Π001–Εφαρμοσμένα Μαθηματικά» του τμήματος Μηχανικών Μουσικής Τεχνολογίας και Ακουστικής της Σχολής Εφαρμοσμένων Επιστημών του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Κρήτης. Η ευκαιρία αυτή μου δόθηκε στα πλαίσια της δράσης «Απόκτηση Ακαδημαϊκής Διδακτικής Εμπειρίας σε Νέους Επιστήμονες Κατόχους Διδακτορικού».

Σε καμία περίπτωση δεν αποτελεί εκτενές οδηγό ούτε πλήρες εγχειρίδιο για το ευρύ πεδίο των Εφαρμοσμένων Μαθηματικών. Από το πεδίο αυτό έχουν επιλεχθεί θέματα που κυρίως άπτονται της ενότητας της Αριθμητικής Ανάλυσης [4] τα οποία και παρουσιάζονται συνοπτικά.

Τα υπολογιστικά εργαλεία που χρησιμοποιήθηκαν είναι το περιβάλλον εργασίας `SDE` και η βιβλιοθήκη `Climax` αμφότερα ανεπτυγμένα σε γλώσσα `Java` με ταυτόχρονη αξιοποίηση της γλώσσας `Groovy`. Σκοπός μας είναι πέρα από το να προτείνουμε την χρήση των συγκεκριμένων εργαλείων (Java, Groovy, Climax, SDE) ως μια εναλλακτική για την διδακτική διαδικασία, ταυτόχρονα να πειραματιστούμε και να ελέγξουμε για την καταλληλότητα και προσαρμοστικότητα τους.

Η παρούσα μορφή δεν μπορεί να θεωρηθεί ολοκληρωμένη, αποτελεί όμως μία ειλικρινή προσπάθεια μιας πρώτης έκδοσης δεδομένου των συνθηκών που διαμορφώθηκαν ύστερα από σχετικά προβλήματα (κατάληψη, συμβάσεις, κ.α.) και καθυστερήσεις κατά την έναρξη του χειμερινού εξαμήνου του 2018–2019 στο τμήμα Μηχανικών Μουσικής Τεχνολογίας και Ακουστικής ειδικά, αλλά και του Τεχνολογικού Εκπαιδευτικού Ιδρύματος Κρήτης γενικότερα.

Χ. Παναγιωτόπουλος
Ιανουάριος 2019

Κεφάλαιο 1

Μαθηματική μοντελοποίηση

Τις τελευταίες δεκαετίες με τη ραγδαία ανάπτυξη των ηλεκτρονικών υπολογιστών η συστηματική αξιοποίηση των εφαρμοσμένων μαθηματικών και των αριθμητικών μεθόδων έχει λάβει εξέχουσα θέση στην διαδικασία επίλυσης προβλημάτων επιστημονικών εφαρμογών[1].

Για να περιγράψει κανείς κάποιο φυσικό φαινόμενο υιοθετεί ή και ορίζει κάποιο κατάλληλο μαθηματικό μοντέλο. Το μαθηματικό αυτό μοντέλο συνήθως αποτελείται από τις γνωστές και άγνωστες μεταβλητές, τις σταθερές παραμέτρους και τις μαθηματικές σχέσεις που τις συνδέουν. Τις περισσότερες φορές οι εξισώσεις που περιγράφουν το μαθηματικό μοντέλο είναι τέτοιας πολυπλοκότητας που είναι πολύ δύσκολο ή και ακατόρθωτο να βρεθεί μια αναλυτική λύση για αυτές.

Το πιο πάνω μαθηματικό μοντέλο, τις περισσότερες φορές, προσεγγίζεται από ένα αριθμητικό μοντέλο η αριθμητική λύση του οποίου επιτυγχάνεται από διαδικασίες που αποτελούνται από πεπερασμένη σειρά ενεργειών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, που στοχεύουν στην επίλυση ενός προβλήματος, γνωστές ως αλγόριθμοι[1].

Είναι φανερό ότι η πιο πάνω διαδικασία αντικαταστάσεων, από το φυσικό μοντέλο στο μαθηματικό, από το μαθηματικό στο αριθμητικό και τέλος η επίλυση του αριθμητικού αυτού μοντέλου, εισάγει κάποια σφάλματα στα οποία θα αναφερθούμε συνοπτικά σε αυτό το κεφάλαιο.

[1] Βλέπε και στην `https://el.wikipedia.org/wiki`, στο αντίστοιχο λήμμα (Αλγόριθμος).

1.1 Σφάλματα

Η έννοια του σφάλματος στο πεδίο των επιστημονικών υπολογισμών κατέχει μια σημαντική θέση στην θεωρία και την ανάλυση των αριθμητικών μεθόδων.

Το σφάλμα της αντικατάστασης του φυσικού μοντέλου από το μαθηματικό μοντέλο δεν είναι εύκολο να ποσοτικοποιηθεί αν και θα το καλούμε εδώ 'σφάλμα μοντελοποίησης'. Η αντικατάσταση του μαθηματικού μοντέλου από το αριθμητικό μοντέλο εισάγει το 'σφάλμα αποκοπής'. Ως παράδειγμα μπορεί κανείς να σκεφτεί την αναλυτική μορφή του αναπτύγματος μια συνάρτησης σε σειρά άπειρου πλήθους για τον αριθμητικό υπολογισμό της οποίας χρησιμοποιούνται πεπερασμένου αριθμού όροι.

Είναι επίσης σύνηθες στους υπολογισμούς μας να αντικαθιστούμε έναν αριθμό με μεγάλο πλήθος δεκαδικών ψηφίων από έναν άλλο με πεπερασμένο πλήθος δεκαδικών ψηφίων. Το σφάλμα που υπεισέρχεται από την προσέγγιση αυτή ονομάζεται 'σφάλμα στρογγυλοποίησης'. Όσο αφορά την στρογγυλοποίηση ενός αριθμού με χρήση m δεκαδικών ψηφίων, το ψηφίο της m-στης δεκαδικής θέσης του προσεγγιστικού αριθμού μένει ως έχει ή αυξάνεται κατά μία μονάδα, ανάλογα με το αν το παραλειπόμενο μέρος είναι μικρότερο ή μεγαλύτερο από μισή μονάδα της m-οστής δεκαδικής τάξης που διατηρείται. Όταν το παραλειπόμενο μέρος είναι ίσο με τη μισή μονάδα τότε αφήνουμε το ψηφίο της m-οστής δεκαδικής θέσης όπως είναι αν αυτό είναι άρτιο ή το αυξάνουμε κατά μια μονάδα αν αυτό είναι περιττό.

Η απολυτή διαφορά του αριθμού x^* που είναι μια προσέγγιση του αριθμού x θα είναι:

$$|e| = |x^* - x|$$

και καλείται 'απόλυτο σφάλμα'. Ο αριθμός αυτός δεν μπορεί να αποδώσει ικανοποιητικά την ποιότητα της προσέγγισης του αριθμού x από τον αριθμό x^* καθώς αυτή θα εξαρτάται από το μέγεθος του αριθμού x. Για το λόγο αυτό χρησιμοποιούμε το 'απόλυτο σχετικό σφάλμα' που είναι:

$$|e|/|x|$$

η προσεγγιστικά:

$$|e|/|x^*|.$$

Τέλος στην πράξη πολλές φορές χρησιμοποιούμε το 'ποσοστιαίο σφάλμα', το οποίο θα συναντάμε καμιά φορά και σαν 'επί της εκατό σφάλμα', ως:

$$100|e|/|x|.$$

Κεφάλαιο 2

Αριθμητική αναζήτηση ριζών και επίλυση εξισώσεων

2.1 Εισαγωγή

Ένα απο τα σημαντικότερα προβλήματα που συναντά κανείς είναι αυτό της επίλυσης κάποιας εξίσωσης της μορφής:

$$f(x) = 0 \tag{2.1}$$

με άλλα λόγια τον υπολογισμό των τιμών της μεταβλητής x για τις οποίες μηδενίζεται η συνάρτηση f. Οι τιμές αυτές της μεταβλητής x ονομάζονται και ρίζες της εξίσωσης (2.1). Όταν η μοναδική ανεξάρτητη μεταβλητή είναι η x αναφερόμαστε σε ένα μονοδιάστατο πρόβλημα. Έχοντας περισσότερες της μιας ανεξάρτητες μεταβλητές $\mathbf{x}$ τότε παραπάνω της μιας εξίσωσης μπορούν να ικανοποιούνται ταυτόχρονα και θα έχουμε ένα πολυδιάστατο πρόβλημα. Αν οι ανεξάρτητες μεταβλητές είναι n στον αριθμό τότε το διάνυσμα $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ θα μπορεί να ικανοποιεί έως και n εξισώσεις $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ του ορίζοντας ένα n-διαστατο πρόβλημα το οποίο και θα γράφεται ως.

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{2.2}$$

Η εξ. (2.2) ουσιαστικά αφορά ένα σύστημα n εξισώσεων με n άγνωστους, τις συντεταγμένες του διανύσματος x. Στο παρόν κεφάλαιο θα μας απασχολήσει η επίλυση της εξίσωσης (2.1). Οι εξισώσεις f που μελετάμε μπορεί να είναι πολυωνυμικές (αλγεβρικές), της μορφής:

$$f(x) = a_\nu x^\nu + a_{\nu-1}x^{\nu-1} + \ldots + a_1 x + + a_0 \tag{2.3}$$

ή υπερβατικές, όταν δηλαδή η άγνωστη μεταβλητή x θα βρισκεται μέσα σε τριγωνομετρική, λογαριθμική, εκθετική κ.τ.λ. συνάρτηση, οπότε και θα έχει μια μορφή όπως:

$$f(x) = x - e^{x^2 - x^3} + x\cos(x)\sinh x^2 \tag{2.4}$$

ή παρόμοια.

Στην πλειονότητα των περιπτώσεων που συναντάμε σε πρακτικά προβλήματα η εξ. (2.1) είναι αδύνατο ή πολύ δύσκολο να λυθεί αναλυτικά. Για το λόγο αυτό έχουν αναπτυχθεί οι προσεγγιστικές μέθοδοι εύρεσης της λύσης. Η προσεγγιστική λύση x^* είναι ένας αριθμός που μόνο προσεγγιστικά ικανοποιεί την εξ. (2.1), δηλαδή ισχύει:

$$|f(x^*)| = \epsilon$$

με το ϵ ένα πολύ μικρό αριθμό κοντά στο μηδέν.

Ένα πολύ βασικό στοιχείο όλων των προσεγγιστικών μεθόδων είναι η απαίτηση να γνωρίζουμε ένα διάστημα στο οποίο θα βρίσκεται τουλάχιστον μια ρίζα της εξίσωσης ή ένα σημείο έναρξης x_{init} το οποίο υποτίθεται να είναι 'κοντά' σε σημείο που είναι ρίζα της εξίσωσης. Για να βρούμε τέτοια διαστήματα σημείων ή κάποια σημεία εκκίνησης βοηθάει συχνά το διάγραμμα της, γνωστής και δεδομένης, συνάρτησης στη μεταβλητή x. Η επιλογή του διαστήματος ή του σημείου εκκίνησης παίζει σημαντικό ρόλο για τη σύγκλιση των μεθόδων στη σωστή ακριβή τιμή της ρίζας της εξίσωσης.

2.1.1 Κριτήρια διακοπής επαναλήψεων

Οι προσεγγιστικές μέθοδοι που περιγράφουμε εδώ για την εύρεση των ριζών μιας εξίσωσης αποτελούν μια επαναληπτική διαδικασία η οποία και θα πρέπει να εκτελεί πεπερασμένο αριθμό κύκλων επανάληψης. Για τη περίπτωση του μονοδιάστατου προβλήματος της ελ. (2.1) τα κυριότερα κριτήρια παύσης της διαδικασίας επαναλήψεων δίνονται πιο κάτω. Σε όλα τα κριτήρια θα πρέπει να ορίσουμε ένα μικρό αριθμό ϵ που θα είναι η ανοχή σφάλματος (tolerance). Ο δείκτης i δείχνει το τρέχον βήμα επανάληψης.

- $|x_i - x_{i-1}| < \epsilon$
- $|f(x_i)| < \epsilon$
- $\frac{|x_i - x_{i-1}|}{x_i} < \epsilon$ με $x_i \neq 0$

2.2 Μέθοδος μέσου σημείου (ή διχοτόμησης)

Η μέθοδος του μέσου σημείου η μέθοδος διχοτόμησης (bisection) βασίζεται στο θεώρημα Bolzano της μέσης τιμής, που λέει ότι, *αν μια συνάρτηση f με πεδίο ορισμού το $[a, b]$ είναι συνεχής σε αυτό και ισχύει $f(a)f(b) < 0$, τότε υπάρχει ένα τουλάχιστον σημείο μηδενισμού της f σε αυτό το διάστημα.* Αν η ρίζα αυτή είναι απλή, τότε ο προσδιορισμός της ρίζας μπορεί να γίνει με τον αλγόριθμο της μεθόδου διχοτόμησης (Αλγόριθμος 1.2.1-1 στην αναφορά [5]).

Αλγόριθμος: Μέθοδος μέσου σημείου (διχοτόμησης)

- Είσοδος: Συνάρτηση $f(x)$ και $[a, b]$, επίπεδο ακρίβειας ϵ.
- Αρχικοποίηση:
  Όρισε $[a_1, b_1] \subset [a, b]$, με $f(a_1)f(b_1) < 0$, θέσε $i = 1$.
- Επαναλήψεις με αυξανόμενο i:
  1. Θέσε $x_i = (a_i + b_i)/2$·
  2. Αν $f(a_i)f(x_i) < 0$, θέσε $b_{i+1} = p_i$, $a_{i+1} = a_i$·
     αλλιώς, θέσε $a_{i+1} = x_i$, $b_{i+1} = b_i$·
  3. Αν $(b_i - a_i)/2 > \epsilon$, θέσε $i = i + 1$ μετάβαση στο βήμα 1
  4. Παύση με $x^* = x_i = (b_{i+1} + a_{i+1})/2$.

  Σημειώσεις:
  Άλλα κριτήρια παύσης (σύγκλισης) είναι δυνατό να επιλεχθούν, π.χ. $|f(x_i)| < \epsilon$.

Εδώ δίνεται σε απλό κώδικα Groovy/Climax η επίλυση της εξίσωσης

$$f(x) = x^3 + 4x^2 - 10 = 0 \tag{2.5}$$

στο διάστημα $[1, 2]$ (βλέπε Παράδειγμα 1.2.1-1 και Πρόγραμμα 1.2.1-1 στην [5]).

```
a=1; b=2
a1=a; b1=b
Nmax=200; eps=1.0e-6
f={x**3+4*x**2-10}
sol=[]
converge=false
iter=1
while(iter<=Nmax && !converge){
```

```
  x=(a1+b1)/2.0
  fx=f(x)
  sol.add(x)
  printf("%d: x= %8.6f and f(x)= %6.4e \n",iter,x,fx)
  if(abs(fx)<eps){
    converge=true
  }else{
    if(fx*f(a)>0){a1=x}else{b1=x}
  }
  iter++
}

thePlot.clear()
pf=new plotfunction(sol)
thePlot.addFunction(pf)
thePlot.setMarker(true)
thePlot.show()
```

2.3 Μέθοδος Newton

Η μέθοδος Newton είναι γνωστή και ως μέθοδος Newton-Raphson. Η μέθοδος θα μπορούσε να παρουσιαστεί με χρήση του αναπτύγματος Taylor. Έστω η f στο διάστημα $[a, b]$ με παραγώγους μέχρι και δεύτερης τάξης συνεχείς σε αυτό το διάστημα, τότε έχουμε:

$$f(x) \approx f(\xi) + (x - \xi)f'(\xi) + \frac{(x - \xi)^2}{2} f''(\xi)$$

Υποθέτοντας ότι η τιμή x^* είναι ρίζα της f τότε:

$$0 \approx f(\xi) + (x^* - \xi)f'(\xi) + \frac{(x^* - \xi)^2}{2} f''(\xi)$$

Θεωρώντας ότι ο αριθμός $|x^* - \xi|$ είναι πολύ μικρός τότε ο όρος $(x - \xi)^2$ μπορεί να παραληφθεί και να έχουμε:

$$0 \approx f(\xi) + (x^* - \xi)f'(\xi)$$

οπότε και λύνοντας ως προς x^*:

$$x^* \approx \xi - \frac{f(\xi)}{f'(\xi)} \tag{2.6}$$

Από την σχέση της εξ. (2.6) μπορεί να προκύψει ο αλγόριθμος της μεθόδου Newton.

Αλγόριθμος: Μέθοδος Newton

- Είσοδος: Συνάρτηση $f(x)$, αρχική τιμή x_{init}, επίπεδο ακρίβειας ϵ.
- Αρχικοποίηση:
  Θέσουμε $i = 1$ και $x_0 = x_{init}$.
- Επαναλήψεις με αυξανόμενο i:
  1. Θέσε $x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$.
  2. Αν $|f(x_i)| > \epsilon$, θέσε $i = i + 1$ και $x_{i-1} = x_i$, μετάβαση στο βήμα 1
  3. Παύση με $x^* = x_i$.

Εδώ δίνεται σε απλό κώδικα Groovy/Climax η επίλυση της εξίσωσης

$$f(x) = x^3 + 2x^2 + 10x - 20 = 0 \tag{2.7}$$

(βλέπε Παράδειγμα 1.4.1-1 στην [5]).

```
x0=2.1d
tol=1.0e-6
Nmax=100
f={it**3+2*it**2+10*it -20}
df={3*it**2+4*it+10}
sol=[]
converge=false
iter=0
sol.add(x0)
fx=f(x0); dfx=df(x0)
printf("%d: x= %8.6f and f(x)= %6.4e \n",iter,x0,fx)
while(iter<=Nmax && !converge){
   iter++
   fx=f(x0); dfx=df(x0)
   x=x0-fx/dfx
   fx=f(x);
   sol.add(x)
   printf("%d: x= %8.6f and f(x)= %6.4e \n",iter,x,fx)
   if(abs(fx)<tol)converge=true
   x0=x
}

thePlot.clear()
pf=new plotfunction(sol)
```

```
25 thePlot.addFunction(pf)
26 thePlot.setMarker(true)
27 thePlot.show()
```

2.4 Μέθοδος της τέμνουσας

Ένα μειονέκτημα της μεθόδου Newton είναι η απαίτηση της ύπαρξης της αναλυτικής μορφής της πρώτης παραγώγου της συνάρτησης $f(x)$, δηλαδή την $f'(x)$. Η μέθοδος της τέμνουσας ή αλλιώς της χορδής (secant method) μπορεί να ιδωθεί ως εναλλακτική της μεθόδου Newton, όπου η τιμή της παραγώγου της συνάρτησης στο σημείο ξ προσεγγίζεται από ένα σχήμα πεπερασμένων διάφορων.

Η βασική επαναληπτική σχέση της μεθόδου μπορεί να γραφτεί ως:

$$x_i = x_{i-1} - f(x_{i-1})\frac{x_{i-1} - x_{i-2}}{f(x_{i-1}) - f(x_{i-2})} \tag{2.8}$$

και όπως είναι εμφανές η μέθοδος απαιτεί δυο αρχικά σημεία x_0 και x_1 τα οποία πρέπει να είναι κοντά στη ριζά της εξίσωσης.

Αλγόριθμος: Μέθοδος τέμνουσας

- Είσοδος: Συνάρτηση $f(x)$, αρχική τιμή x_{init_1} και x_{init_2}, επίπεδο ακρίβειας ϵ.
- Αρχικοποίηση:
  Θεσουμε $i = 1$ και $x_a = x_{init_1}$ και $x_b = x_{init_2}$.
- Επαναλήψεις με αυξανόμενο i:
  1. Θέσε $f_a = f(x_a)$, $f_b = f(x_b)$·
  2. Θέσε $x_i = x_b - f_b\frac{x_b - x_a}{f_b - f_a}$·
  3. Αν $|f(x_i)| > \epsilon$,
     θέσε $i = i + 1$ και $x_a = x_b$, $x_b = x_i$
     μετάβαση στο βήμα 1
  4. Παύση με $x^* = x_i$.

Εδώ δίνεται σε απλό κώδικα Groovy/Climax η επίλυση της εξίσωσης

$$f(x) = \cos x - x = 0 \tag{2.9}$$

(βλέπε Παράδειγμα 1.4.2-1 στην [5]).

```
x0=0.5d
x1=PI/4.0d
tol=1.0e-4
Nmax=100
f={cos(it)-it}
converge=false
iter=0
printf("%d: x= %12.10f and |x0-x1|= %6.4e \n",iter,x,abs(x0-x1))
while(iter<=Nmax && !converge){
  iter++
  fx=f(x0); fx1=f(x1)
  x=x1-fx1*(x1-x0)/(fx1-fx)
  fx=f(x);
  printf("%d: x= %12.10f and |x0-x1|= %6.4e \n",iter,x,abs(x0-x1
    ))
  if(abs(x0-x1)<tol)converge=true
  x0=x1; x1=x
}
```

2.5 Μέθοδος Muller

Στις μέχρι τώρα μεθόδους υποθέταμε πως η ανεξάρτητη μεταβλητή x θα είναι ένας πραγματικός αριθμός όπως άλλωστε και η τιμή της συνάρτησης $f(x)$. Με αυτόν τον τρόπο θα ήταν αδύνατο να λάβουμε τις μιγαδικές ρίζες της εξίσωσης $f(x)$. Ένα απλό παράδειγμα θα ήταν να προσπαθήσουμε να λύσουμε με τη μέθοδο της τέμνουσας (όπως παρατίθεται πιο πάνω) για τη συνάρτηση $f(x) = x^2 + 1$. Θεωρώντας όμως αρχικές τιμές μιγαδικούς αριθμούς η εύρεση αμφότερων πραγματικών και μιγαδικών ριζών είναι δυνατή. Σε αυτή την παράγραφο παρουσιάζουμε μια γενίκευση της μεθόδου της τέμνουσας η οποία είναι δυνατό να εντοπίσει τις μιγαδικές ρίζες της εξίσωσης ακόμα και αν ξεκινήσουμε από πραγματικές αρχικές τιμές. Η μέθοδος είναι γνωστή ως μέθοδος Muller και διαφοροποιείται από την προηγούμενη καθώς αντί της κατασκευής μιας γραμμής που περνάει από δύο σημεία κάνει χρήση μιας παράβολης που περνάει από τρία σημεία. Ως επακόλουθο η μέθοδος απαιτεί τρία αρχικά σημεία. Η επαναληπτική σχέση της μεθόδου μπορεί να γραφτεί ως:

$$x_i = x_{i-1} - \frac{2f(x_{i-1})}{w \pm \sqrt{w^2 - 4f(x_{i-1})f[x_{i-1}, x_{i-2}, x_{i-3}]}} \tag{2.10}$$

στην πιο πάνω σχέση το πρόσημο από το $\pm$ επιλέγεται έτσι ώστε ο παρανομαστής να έχει το μεγαλύτερο μέτρο. Για τον όρο $f[x_{i-1}, x_{i-2}, x_{i-3}]$ γνωστό ως

διαιρεμένη διαφορά (divided differences[1]) βλέπε στη βιβλιογραφία [5]. Στην αλγοριθμική μορφή της μεθόδου θα δώσουμε μια ίσως πιο κατάλληλη μορφή για εφαρμογή σε υπολογιστή που μπορεί να συναντήσει κάνεις στη βιβλιογραφία [6].

Αλγόριθμος: Μέθοδος Muller

- Είσοδος: Συνάρτηση $f(x)$, αρχική τιμή x_{init_1}, x_{init_2} και x_{init_3}, επίπεδο ακρίβειας ϵ.
- Αρχικοποίηση:
  Θέτουμε $i = 1$ και $x_a = x_{init_1}$, $x_b = x_{init_2}$ και $x_c = x_{init_3}$.
- Επαναλήψεις με αυξανόμενο i:
  1. Θέσε $f_a = f(x_a)$, $f_b = f(x_b)$, $f_c = f(x_c)$·
  2. Θέσε $q = \frac{x_c - x_b}{x_b - x_a}$
  3. Θέσε $A = qf_c - q(q+1)f_b + q^2 f_a$
  4. Θέσε $B = (2q+1)f_c - (1+q)^2 f_b + q^2 f_a$
  5. Θέσε $C = (1+q)f_c$
  6. Θέσε $x_i = x_c - (x_c - x_b)\left[\frac{2C}{B \pm \sqrt{B^2 - 4AC}}\right]$·
  7. Αν $|f(x_i)| > \epsilon$,
     θέσε $i = i + 1$ και $x_a = x_b$, $x_b = x_c$ και $x_c = x_i$
     μετάβαση στο βήμα 1
  8. Παύση με $x^* = x_i$.

Ξανά εδώ το πρόσημο από το $\pm$ επιλέγεται έτσι ώστε ο παρανομαστής να έχει το μεγαλύτερο μέτρο.

Εδώ δίνεται σε κώδικα Groovy/Climax η επίλυση της εξίσωσης

$$f(x) = x^3 + 5x^2 + 9x + 45 = 0 \tag{2.11}$$

Το οποίο παράδειγμα έχουμε πάρει από τον επίσημο ιστότοπο[2] της IMSL Fortran Numerical Math Library για να γίνει έλεγχος του αλγόριθμου.

[1] https://en.wikipedia.org/wiki/Divided_differences

[2] https://docs.roguewave.com/imsl/fortran/6.0/math/default.htm?turl=zanly.htm

```
x0=new Complex(10.1d,0.0d); x1=new Complex(11.1d,0.0d)
x2=new Complex(12.1d,0.0d); tol=1.0e-16 ; Nmax=100

c2=new Complex(2.0d,0.0d); c4=new Complex(4.0d,0.0d)
c5=new Complex(5.0d,0.0d); c9=new Complex(9.0d,0.0d)
c45=new Complex(45.0d,0.0d)

rr=[]
f={
  fun=(it.pow(3)+c5*it.pow(2)+c9*it+c45)
  for(i in 0..<rr.size())fun/=(it-rr[i])
  return fun
}

converge=false; iter=0; fx=f(x2)
printf("%d: x= (%8.6f,%8.6f) and f(x)= (%6.4e,%6.4e) accuracy
    =%6.4e\n",iter,x0.re(),x0.im(),fx.re(),fx.im(),fx.abs())
while(iter<=Nmax && !converge){
  iter++;fx0=f(x0); fx1=f(x1); fx2=f(x2)

  q=(x2-x1)/(x1-x0)
  A=q*fx2-q*(c1+q)*fx1+q*q*fx0
  B=(c2*q+c1)*fx2-(c1+q)*(c1+q)*fx1+q*q*fx0
  C=(c1+q)*fx2
  denom=B-(B*B-c4*A*C).sqrt()
  if( denom.abs()< (B+(B*B-c4*A*C).sqrt()).abs() )denom=B+(B*B-
    c4*A*C).sqrt()
  x=x2-(x2-x1)*c2*C/denom

  fx=f(x); acr=fx.abs()
  printf("%d: x= (%8.6f,%8.6f) and f(x)= (%6.4e,%6.4e) accuracy
    =%6.4e\n",iter,x0.re(),x0.im(),fx.re(),fx.im(),fx.abs())
  if(fx.abs()<tol)converge=true
  x0=x1; x1=x2; x2=x
}
```

2.5.1 Εφαρμογές

Εύρεση ριζών χαρακτηριστικής εξίσωσης καμπτικών ταλαντώσεων δοκού παραδοχών Euler-Bernoulli

Η διαφορική εξίσωση που διέπει το πρόβλημα των καμπτικών ταλαντώσεων είναι,

$$EI\frac{\partial^4 u(x,t)}{\partial x^4}+m\frac{\partial^2 u(x,t)}{\partial t^2}=0 \tag{2.12}$$

Πίνακας 2.1: Ρίζες r της χαρακτηριστικής εξ. (2.13) για L=100m

j	αναλυτική προσέγγιση	μέθοδος τέμνουσας	μέθοδος Muller
1	0.0471238898	0.04730041	0.04730041
2	0.0785398163	0.07853205	0.07853205
3	0.1099557429	0.10995608	0.10995608
4	0.1413716694	0.14137165	0.14137165

Η χαρακτηριστική εξίσωση καμπτικών ταλαντώσεων συνεχούς αμφίπακτης δοκού συνολικού μήκους L, προκύπτει να είναι η υπερβατική εξίσωση:

$$\cos(rL)\cosh(rL) - 1 = 0$$

Η αναλυτικά προσεγγιστική λύση[3] της πιο πάνω εξίσωσης δίνει τις ρίζες,

$$r_j = \left(j + \frac{1}{2}\right)\frac{\pi}{L}, \qquad \text{για } j = 1, 2, \ldots, \infty$$

Για μήκος δοκού $L = 100$m να επαληθευθεί το αποτέλεσμα με χρήση κατάλληλης αριθμητικής μεθόδου.

Οι λύσεις των υπολογιστικών μεθόδων, που να σημειώσουμε εδώ εξαρτώνται πολύ από την αρχική τιμή εκτίμησης της ρίζας, βρίσκονται σε συμφωνία μεταξύ τους και είναι ίδιες με την ακριβή αναλυτική λύση που δίνεται στη βιβλιογραφία (π.χ. [2] και [3])

Αξίζει εδώ να σημειωθεί ότι οι αντίστοιχες (κυκλικές) ιδιοσυχνότητες [2] προκύπτουν από τη σχέση,

$$\omega_j = r_j^2\sqrt{\frac{EI}{\varrho AL}} \tag{2.13}$$

Σημειώνεται εδώ πως θα μπορούσαμε να γράψουμε τη πιο πάνω σχέση ως

$$r_j = \sqrt{\omega_j\left(\frac{EI}{\varrho AL}\right)^{-\frac{1}{2}}} \tag{2.14}$$

οπότε και η χαρακτηριστική εξίσωση (2.13) να γραφεί σαν σε συνάρτηση της άγνωστης κυκλικής ιδιοσυχνότητας ω,

$$f(\omega) = \cos(r(\omega)L)\cosh(r(\omega)L) - 1 \tag{2.15}$$

Πίνακας 2.2: Ρίζες ω της χαρακτηριστικής εξ. (2.15) για L=100m

j	αναλυτική προσέγγιση	μέθοδος τέμνουσας	μέθοδος Muller
1	4.822002783	4.8581956825	4.8581956825
2	13.3944521636	13.3918033112	13.3918033112
3	26.2531262788	26.2532872516	26.2532872516
4	43.3980250469	43.3980131361	43.3980131361

Οι ρίζες της οποίας δινονται στον πιν. 2.2.

Στην γενικότερη θεώρηση σύμφωνα με την οποία η δοκός εδράζεται πάνω σε σύστημα συνεχών ελατηρίων (Winkler) ελατηριακής σταθεράς k και λαμβάνοντας υπόψη την επιρροή στην κάμψη κάποιας αξονικής εσωτερικής δύναμης N η διαφορική εξίσωση θα έχει την πιο σύνθετη μορφή,

$$EI\frac{\partial^4 u(x,t)}{\partial x^4} + m\frac{\partial^2 u(x,t)}{\partial t^2} + N\frac{\partial^2 u(x,t)}{\partial^2 x} + ku(x,t) = 0 \tag{2.16}$$

ενώ η χαρακτηριστική εξίσωση θα δίνεται από τη σχέση,

$$\begin{aligned}
&f(\omega) = \\
&(s_1 - s_4)\Big(\big(\exp(s_2L) - \exp(s_4L)\big)\big(s_3\exp(s_3L) - s_4\exp(s_4L)\big) \\
&\qquad -\big(\exp(s_3L) - \exp(s_4L)\big)\big(s_2\exp(s_2L) - s_4\exp(s_4L)\big)\Big) + \\
&(s_2 - s_4)\Big(\big(\exp(s_3L) - \exp(s_4L)\big)\big(s_2\exp(s_2L) - s_4\exp(s_4L)\big) \\
&\qquad -\big(\exp(s_1L) - \exp(s_4L)\big)\big(s_3\exp(s_3L) - s_4\exp(s_4L)\big)\Big) + \\
&(s_3 - s_4)\Big(\big(\exp(s_1L) - \exp(s_4L)\big)\big(s_2\exp(s_2L) - s_4\exp(s_4L)\big) \\
&\qquad -\big(\exp(s_2L) - \exp(s_4L)\big)\big(s_1\exp(s_1L) - s_4\exp(s_4L)\big)\Big)
\end{aligned} \tag{2.17}$$

[3]Η λύση αυτή θεωρείται ακριβής για $j > 5$, βλ. [3].

Πίνακας 2.3: Ρίζες ω της χαρακτηριστικής εξ. (2.17) για συνδυασμούς (N, k)

j	$(0,0)$	$(N,0)$	$(0,k)$	(N,k)
1	4.85819516	6.80749014	21.71861214	21.27962966
2	13.39180203	16.29553232	22.25534033	22.76049286
3	26.25328648	29.58892641	25.51545561	27.15220961
4	43.39801615	46.97780319	34.07246930	36.70425969
5	64.82914901	68.56222904	48.52922747	51.75530993

όπου $s_i = s_i(\omega)$, για i από 1 έως 4. Πιο συγκεκριμένα,

$$s_1 = \sqrt{\frac{N + \sqrt{N^2 + 4EI(\omega^2 m - k)}}{2EI}} \tag{2.18}$$

$$s_2 = -\sqrt{\frac{N + \sqrt{N^2 + 4EI(\omega^2 m - k)}}{2EI}} \tag{2.19}$$

$$s_3 = \sqrt{\frac{N - \sqrt{N^2 + 4EI(\omega^2 m - k)}}{2EI}} \tag{2.20}$$

$$s_4 = -\sqrt{\frac{N - \sqrt{N^2 + 4EI(\omega^2 m - k)}}{2EI}} \tag{2.21}$$

Η αριθμητική επίλυση της εξίσωσης (2.17) για την εύρεση των ριζών της ... για L=100m και m=0.53tn/m καθώς και για συνδυασμούς των N=10000kN και k=250kN/m^2 δίνει τα αποτελέσματα του αντίστοιχου πίνακα.

Κεφάλαιο 3

Αριθμητική επίλυση συστημάτων εξισώσεων

Όπως είδαμε και στο κεφ. 2 ένα πρόβλημα μπορεί να περιγράφεται από N ανεξάρτητες μεταβλητές $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Έστω ότι στο ίδιο πρόβλημα υπάρχουν M γραμμικές εξισώσεις επί των μεταβλητών $\mathbf{x}$, ορίζεται τότε ένα σύστημα γραμμικών εξισώσεων ή αλλιώς ένα γραμμικό σύστημα.

3.1 Γραμμικό σύστημα

Η γενική μορφή ενός γραμμικού συστήματος m-εξισώσεων με n-αγνώστους $x_1, x_2, \ldots, x_n$, είναι:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1j}x_j + \cdots + a_{1n}x_n &= b_1 \\
&\vdots \\
a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ij}x_j + \cdots + a_{in}x_n &= b_i \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mj}x_j + \cdots + a_{mn}x_n &= b_m
\end{aligned} \tag{3.1}$$

Αν $b_i = 0$ για καθε $i = 1, 2, \ldots, m$, τότε το σύστημα λέγεται ομογενές με προφανή (τετριμμένη) λύση την:

$$x_1 = x_2 = \ldots = x_N = 0,$$

ενώ όταν τουλάχιστον ένα b_i διάφορο του μηδέν, τότε το σύστημα λέγεται μη ομογενές.

Με τη βοήθεια των πινάκων το σύστημα μπορεί να γραφτεί ως:

$$\underbrace{\begin{bmatrix} a_{11} & a_{21} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ a_{i1} & a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ a_{m1} & a_{m1} & \dots & a_{mj} & \dots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_m \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_m \end{bmatrix}}_{\mathbf{b}}$$

ή αλλιώς

$$A\mathbf{x} = \mathbf{b}, \tag{3.2}$$

όπου A ο πίνακας των συντελεστών των αγνώστων τάξης (m, n) και $\mathbf{b}$ διάνυσμα τάξης (m), με $A \in \mathbb{R}^{m\times n}$, αντίστοιχα $A \in \mathbb{C}^{m\times n}$ και $\mathbf{b} \in \mathbb{R}^m$, αντίστοιχα $\mathbf{b} \in \mathbb{C}^m$.

3.1.1 Μέθοδοι επίλυσης

Εδώ θα περιοριστούμε στην περίπτωση κατά την οποία έχουμε τόσες εξισώσεις όσος και ο αριθμός των αγνώστων μεταβλητών x_i. Θα θεωρήσουμε δηλαδή ότι ο πινάκας A είναι τετραγωνικός $m = n$. Κατά τα γνωστά, ένα τέτοιο γραμμικό μη ομογενές σύστημα της μορφής της εξ. (3.2), θα έχει μια και μόνο λύση για κάθε $\mathbf{b}$ αν ο πίνακας A είναι αντιστρέψιμος, δηλαδή αν ισχύει $\det(A) \neq 0$, όπου $\det(A)$ η ορίζουσα του A που διαφορετικά θα συμβολίζουμε και ως $|A|$. Η επίλυση του προβλήματος με τον κανόνα Cramer[1] είναι υπολογιστικά μη αποδοτικές και για το λόγο αυτό έχουν αναπτυχθεί μέθοδοι υπολογισμού με λιγότερο απαιτούμενο υπολογιστικό κόστος.

Βασικός διαχωρισμός των υπολογιστικών μεθόδων επίλυσης είναι αυτός που τις κατατάσσει στις:

- άμεσες (direct)
- έμμεσες ή επαναληπτικές (iterative)

3.1.2 Άμεσοι μέθοδοι επίλυσης γραμμικών συστημάτων

Οι άμεσες μέθοδοι οδηγούν στη λύση ύστερα από πεπερασμένο αριθμό βημάτων. Ενώ θεωρητικά δίνουν την ακριβή λύση του προβλήματος δεν αγνοούμε ότι λόγω σφαλμάτων στρογγυλοποίησης ενδέχεται, και είναι αναμενόμενο, να

[1] https://en.wikipedia.org/wiki/Cramer's_rule

υπάρχουν διαφορές. Επιπλέον ένα γραμμικό σύστημα μπορεί να περιεχέι συντελεστές τέτοιων αριθμητικών τιμών που να ορίζουν μιας κακής κατάστασης μητρώο (ill-conditioned matrix), σε αυτή την περίπτωση μπορεί να οδηγηθούμε σε αστάθεια της λύσης.

Μέθοδος απαλοιφής Gauss

Η μέθοδος απαλοιφής του Gauss χωρίς διάταξη (pivoting) παρουσιάζεται εδώ επιλύοντας το πιο κάτω πρόβλημα (βλέπε Παράδειγμα 2.2.2-1 στην [5]).

Έστω το σύστημα,

$$\begin{aligned} 2x_1 + 2x_2 + 4x_3 &= 6 \\ -x_1 + 2x_2 - 3x_3 &= 3 \\ x_1 + 2x_2 - x_3 &= 5 \end{aligned}$$

να λυθεί με τη μέθοδο Gauss.

Στην αναφορά [5] μπορεί να αναζητηθεί και η αντίστοιχη θεωρία καθώς και μια πλήρης περιγραφή. Με χρήση του εκπαιδευτικού κώδικα που ακολουθεί μπορεί κανείς να λύσει οποιοδήποτε κατάλληλο σύστημα εξισώσεων τάξης $n \times n$.

```
n=3
A=new Matrix(n,n)
b=new Matrix(n,1)
x=new Matrix(n,1)

A[0,0]=2.0; A[0,1]=2.0; A[0,2]=4.0; b[0,0]=6.0
A[1,0]=-1.0; A[1,1]=2.0; A[1,2]=-3.0; b[1,0]=3.0
A[2,0]=1.0; A[2,1]=2.0; A[2,2]=-1.0; b[2,0]=5.0

println("matrix A:"); A.print();
println("vector b:"); b.print()


println("Gauss elimination")
println("-----------------------------")
(0..<n-1).each{
  step=it
  println("Step: "+step)
  (step+1..<n).each{
    i=it
    m=A[i,step]/A[step,step]
    println("m"+i+"_"+step+"="+m)
    (0..<n).each{
      j=it
```

```
25          A[i,j]=A[i,j]-A[step,j]*m
26        }
27        b[i,0]=b[i,0]-b[step,0]*m
28      }
29      A.print(); b.print()
30  }
31
32  println("Back substitution")
33  println("----------------------------")
34  (n-1..0).each{
35      i=it
36      c=b[i,0]
37      (i..<n).each{
38          c-=A[i,it]*x[it,0]
39      }
40      x[i,0]=c/A[i,i]
41      println("x"+(i)+"="+x[i,0])
42
43  }
44  println("\n Solution vector x:")
45  println("----------------------------")
46  x.print()
```

Αξίζει να σημειωθεί ότι στον πιο πάνω αλγόριθμο δεν είναι απαραίτητο να είναι μητρώα οι μεταβλητές `A` καθώς και `b, x`. Θα μπορούσαν να είναι απλές διατάξεις του τύπου `double[n][n]` και `double[n]`, αντίστοιχα.

Μέθοδος της LU διαμέρισης (decomposition»

Με χρήση των πολλαπλασιαστών/συντελεστών της μεθόδου Gauss μπορούν να διαμορφωθούν κατάλληλοι πινάκες, ένας άνω τριγωνικός L και ένας κάτω τριγωνικός U, τέτοιοι ώστε,

$$A = LU \tag{3.3}$$

δηλαδή ο αρχικός πινάκας A να διαμεριστεί σε τριγωνικούς επιμέρους πίνακες.

Με τον τρόπο αυτό ορίζεται και η μέθοδος LU διαμέρισης, κατάλληλη όταν έχουμε να λύσουμε συστήματα που έχουν ως πινάκα συντελεστών το ίδιο A αλλά διαφορετικά δεξιά μέλη b.

Μέθοδος Cholesky

Η μέθοδος Cholesky αναφέρεται σε συμμετρικούς πίνακες οι οποίοι μπορούν να διαμεριστούν με χρήση ενός και μόνο τριγωνικού πίνακα L ως,

$$A = LL^T \tag{3.4}$$

όπου L^T ο ανάστροφος πίνακας του L.

3.1.3 Έμμεσοι μέθοδοι επίλυσης γραμμικών συστημάτων

Μέθοδος Jacobi

Η μέθοδος απαλοιφής του Jacobi παρουσιάζεται εδώ επιλύοντας το πιο κάτω πρόβλημα (βλέπε Παράδειγμα 2.3.2-1 στην [5]).

Έστω το σύστημα,

$$\begin{aligned} x_1 - 2x_2 + 3x_3 &= -1 \\ -3x_1 + 9x_2 + 1x_3 &= -5 \\ x_1 - x_2 - 7x_3 &= 15 \end{aligned}$$

να λυθεί με τη μέθοδο Jacobi.

Στην αναφορά [5] μπορεί να αναζητηθεί η αντίστοιχη θεωρία καθώς και μια πλήρης περιγραφή. Με χρήση του εκπαιδευτικού κώδικα που ακολουθεί μπορεί κανείς να λύσει οποιοδήποτε κατάλληλο σύστημα εξισώσεων τάξης $n \times n$.

```
n=3
A=new Matrix(n,n)
b=new Matrix(n,1)
x=new Matrix(n,1)
xp=new Matrix(n,1)

A.set(0,0,5.0); A.set(0,1,-2.0); A.set(0,2,3.0); b.set(0,0,-1.0)
    ; x.set(0,0,0.0)
A.set(1,0,-3.0); A.set(1,1,9.0); A.set(1,2,1.0); b.set(1,0,-5.0)
    ; x.set(1,0,0.0)
A.set(2,0,1.0); A.set(2,1,-1.0); A.set(2,2,-7.0); b.set
    (2,0,15.0); x.set(2,0,0.0)

println("matrix A:"); A.print();
println("vector b:"); b.print()
println("initial vector x:"); x.print()

Nmax=50
tol=10.0e-6
converge=false
iter=0
while(iter<=Nmax && !converge){
  iter++
  (0..<n).each{
```

```
22      i=it
23      c=0.0
24      (0..<i).each{c+=A[i,it]*xp[it,0]}
25      (i+1..<n).each{c+=A[i,it]*xp[it,0]}
26      x[i,0]=(b[i,0]-c)/A[i,i]
27    }
28    norm=0.0
29    (0..<n).each{norm+=sqrt((xp[it,0]-x[it,0])**2); xp[it,0]=x[it
        ,0]}
30    println(iter+": "+x[0,0]+" "+x[1,0]+" "+x[2,0])
31    if(norm<=tol)converge=true
32  }
33
34  println("\n Solution vector x, after "+iter+" iterations :")
35  println("-------------------------------------------------")
36  x.print()
```

Μέθοδος Gauss-Seidel

Ο αλγόριθμος Gauss-Seidel προκύπτει αν στον προηγούμενο κώδικα αλλάξουμε τη γραμμή 24 σε:

```
24  (0..<i).each{c+=A[i,it]*x[it,0]}
```

ο υπόλοιπος κώδικας παραμένει όπως έχει.

3.2 Μη γραμμικό σύστημα

Έστω σύστημα μη γραμμικών εξισώσεων του οποίου οι ανεξάρτητες μεταβλητές είναι n στον αριθμό τότε το διάνυσμα $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ και το οποίο θα ικανοποιεί n εξισώσεις $\mathbf{f} = (f_1, f_2, \ldots, f_n)$:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ \vdots \quad & \ldots \quad \vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

ενώ σε συμπτυγμένη μορφή γραφουμε:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{3.5}$$

Αναζητούμε λοιπόν μια τιμή για το διάνυσμα $\mathbf{x}$, έστω η $\mathbf{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T$ για την οποία θα ισχύει:

$$\mathbf{f}(\mathbf{x}^*) \approx \mathbf{0}. \tag{3.6}$$

Μέθοδος Newton

Σε αναλογία με την ενότητα 2.3 εδώ θα είναι,

$$\mathbf{x}^* = \boldsymbol{\xi} - J^{-1}(\boldsymbol{\xi})\mathbf{f}(\boldsymbol{\xi}) \tag{3.7}$$

με J^{-1} το αντίστροφο του Ιακωβιανού μητρώου (Jacobian),

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \frac{\partial f_n(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

καθώς η εύρεση ενός αντίστροφου μητρώου, ιδιαιτέρα όταν η τάξη n είναι μεγάλη, είναι μια ασύμφορη υπολογιστικά ενέργεια, είναι προτιμητέο να γράψουμε ένα πρόβλημα επίλυσης γραμμικού συστήματος (βλ. ενότητα 3.1.1). Πιο αναλυτικά, γράφουμε τη σχέση της εξ. (3.7) σε μορφή επαναλήψεων με δεικτη k.

$$\mathbf{x}^{k+1} = \mathbf{x}^k - J^{-1}(\mathbf{x}^k)\mathbf{f}(\mathbf{x}^k) \tag{3.8}$$

θέτουμε

$$\mathbf{s}(\mathbf{x}^k) = -J^{-1}(\mathbf{x}^k)\mathbf{f}(\mathbf{x}^k) \tag{3.9}$$

το οποίο ισοδύναμα, αντί με χρήση του αντίστροφου του Ιακωβιανού $J^{-1}(\mathbf{x}^k)$, προκύπτει από την επίλυση του συστήματος:

$$J(\mathbf{x}^k)\mathbf{s}(\mathbf{x}^k) = -\mathbf{f}(\mathbf{x}^k) \tag{3.10}$$

οπότε και δημιουργούμε το απαραίτητο διάνυσμα $\mathbf{s}(\mathbf{x}^k)$ και μπορούμε να γράψουμε την εκτίμηση της λύση στο επόμενο βήμα ως:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}(\mathbf{x}^k). \tag{3.11}$$

Εδώ δίνεται σε κώδικα Groovy/Climax η επίλυση του συστήματος εξισώσεων

$$f_1(x_1, x_2) = \exp(x_1) + x_2 - 1 = 0 f_2(x_1, x_2) = x_1^2 + x_2^2 - 4 = 0 \tag{3.12}$$

το οποίο μπορεί κανείς να βρει ως Παράδειγμα 2.4.4-1 στην [5].

```
n=2
x0=new Matrix(n,1)
tol=1.0e-12
Nmax=100

f={ x ->
  fmat=new Matrix(n,1)
  // code to define fmat
  x1=x[0,0]; x2=x[1,0]
  fmat[0,0]=exp(x1)+x2-1.0
  fmat[1,0]=x1*x1+x2*x2-4.0
  return fmat
}

df={ x ->
  dfmat=new Matrix(n,n)
  // code to define dfmat (Jacobian)
  x1=x[0,0]; x2=x[1,0]
  dfmat[0,0]=exp(x1); dfmat[0,1]=1.0
  dfmat[1,0]=2.0*x1; dfmat[1,1]=2.0*x2
  return dfmat
}

x0[0,0]=1.0; x0[1,0]=-1.7

converge=false
iter=0

fx=f(x0); dfx=df(x0)
printf("%d: x= %8.6f and f(x)= %6.4e,%6.4e \n",iter,x0[0,0],x0
    [1,0],fx[0,0],fx[1,0])

while(iter<=Nmax && !converge){
  iter++
  fx=f(x0); dfx=df(x0)
  s=dfx.inverse()*fx
  x=x0-s
  fx=f(x);
  printf("%d: x= %8.6f, %8.6f and f(x)= %6.4e, %6.4e \n",iter,x0
    [0,0],x0[1,0],fx[0,0],fx[1,0])
  if(fx.norm1()<tol)converge=true
  x0=x
}
```

Ο πιο πάνω κώδικας μπορεί εύκολα να τροποποιηθεί για διαφορετικό αριθμό εξισώσεων, αρα και ανεξάρτητων μεταβλητών, καθώς και να χρησιμοποιηθεί για άλλες συναρτήσεις αρκεί να γωνιάζουμε τις παραγώγους τους ως προς τις

ανεξάρτητες μεταβλητές.

Μέθοδος Broyden

Η μέθοδος Broyden[2] για τα προβλήματα συστημάτων μη γραμμικών εξισώσεων, είναι το ανάλογο της μεθόδου της τέμνουσας (βλ. ενότητα 2.4) που είδαμε στα μονοδιάστατα προβλήματα εύρεσης της ρίζας μιας μη γραμμικής εξίσωσης.

[2] https://en.wikipedia.org/wiki/Broyden's_method

Κεφάλαιο 4

Ελαχιστοποίηση και μεγιστοποίηση συνάρτησης

Η ενότητα αυτή θα μπορούσε να έχει τον πιο περιεκτικό τίτλο της *βελτιστοποίησης*.

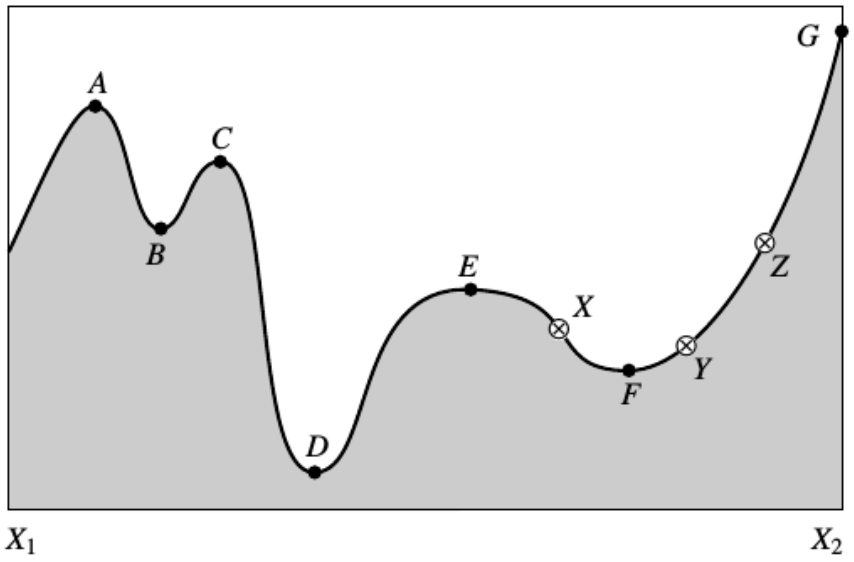
Τα προβλήματα ελαχιστοποίησης μπορούν να τεθούν ως [7], '*ζητούνται οι τιμές των μεταβλητών απόφασης x που ελαχιστοποιούν την αντικειμενική συνάρτηση $f(x)$ κάτω από ένα σύνολο περιορισμών οι οποίοι εκφράζονται ως σχέσεις ισότητας ή ανισότητας*'.

$$\begin{aligned} &\text{ελαχιστοποίηση της} \quad \underset{x}{} && f(x) \\ &\text{κάτω από τις συνθήκες} && \begin{cases} g_i(x) \leq 0, \ i = 1, \ldots, m \\ h_j(x) = 0, \ j = 1, \ldots, p. \end{cases} \end{aligned}$$

ενώ για η μεγιστοποίηση ακολουθεί ακριβώς τον ίδιο ορισμό, αν κανείς αλλάξει τη συνάρτηση $f(x)$ με την αντίθετη της, δηλαδή την $-f(x)$, οπότε και θα καταλήξει στο ταυτόσημο πρόβλημα,

$$\begin{aligned} &\text{μεγιστοποίηση της} \quad \underset{x}{} && -f(x) \\ &\text{κάτω από τις συνθήκες} && \begin{cases} g_i(x) \leq 0, \ i = 1, \ldots, m \\ h_j(x) = 0, \ j = 1, \ldots, p. \end{cases} \end{aligned}$$

Η σχέση των προβλημάτων βελτιστοποίησης με τα αντίστοιχα προβλήματα εύρεσης ριζών είναι μεγάλη και σε πολλές περιπτώσεις το ένα πρόβλημα μπορεί να τεθεί στη μορφή του άλλου.



Σχήμα 4.1: Ακρότατα συνάρτησης σε ένα πεδίο $[X_1, X_2]$. Τα σημεία A, C και E αποτελούν τοπικά μέγιστα. Τα σημεία B και F είναι τοπικά ελάχιστα. Το καθολικό μέγιστο βρίσκεται στο σημείο G το οποίο και όντας στο σύνορο του χωρίου δεν απαιτείται μηδενισμός της παράγωγου της συνάρτησης. Το καθολικό ελάχιστο είναι στο σημείο D. Στο σημείο E οι παράγωγοι μεγαλύτερης της πρώτης τάξης θα είναι μηδενικές, μια συνθήκη στην οποία πολλοί αλγόριθμοι θα εύρισκαν δυσκολίες. Τα σημεία X, Y και Z εσωκλείουν (bracket) το ελάχιστο F, καθώς Y είναι μικρότερο από αμφότερα τα X και Z.

Μια πρώτη κατηγοριοποίηση των προβλημάτων ελαχιστοποίησης (μεγιστοποίησης) είναι αυτή του διαχωρισμού σε αναζήτηση *τοπικών* ακρότατων και σε εκείνη της αναζήτησης *καθολικών* ακρότατων.

Τα προβλήματα βελτιστοποίησης διακρίνονται σε αυτά του γραμμικού προγραμματισμού όταν αμφότερες οι σχέσεις της αντικειμενικής συνάρτησης και των περιορισμών είναι γραμμικές και σε αυτά του μη γραμμικού προγραμματισμού όταν κάποια από τις σχέσεις είναι μη γραμμική.

Σε αυτό το εισαγωγικό κεφάλαιο θα παρουσιάσουμε κάποιες βασικές διαδικασίες και αλγορίθμους βελτιστοποίησης για μονοδιάστατα προβλήματα συναρτήσεων από τον $\mathbb{R}$ στον $\mathbb{R}$ απουσία περιορισμών. Τα προβλήματα βελτιστοποίησης με ισοτικούς ή ανισοτικούς περιορισμούς συνήθως αντιμετωπίζονται με χρήση των πολλαπλασιαστών Lagrange.

4.1 Μέθοδος διχοτόμησης του διαστήματος

Αλγόριθμος: Μέθοδος διχοτόμησης

- Είσοδος: Συνάρτηση $f(x)$ και $[a, b]$, επίπεδο ακρίβειας ϵ. Χρήση πρώτης παραγώγου $f'(x)$.
- Αρχικοποίηση: Θέσε $i = 1$.
- Επαναλήψεις με αυξανόμενο i:
  1. Θέσε $x_i = (a_i + b_i)/2$.
  2. Αν $f'(x_i) = 0$ ή αν $|a_i - b_i| < \epsilon$, πήγαινε στο βήμα 5.
  3. Αν $f'(x_i) > 0$, θέσε $b_{i+1} = x_i$, $a_{i+1} = a_i$ και πήγαινε στο βήμα 1.
  4. Αν $f'(x_i) < 0$, θέσε $a_{i+1} = x_i$, $b_{i+1} = b_i$ και πήγαινε στο βήμα 1.
  5. Παύση με $x^* = x_i$.

Σημειώσεις:
Άλλα κριτήρια παύσης (σύγκλισης) είναι δυνατό να επιλεχθούν, π.χ. $|f(x_i)| < \epsilon$.

Εδώ δίνεται σε απλό κώδικα Groovy/Climax η εύρεση του ελάχιστου της συνάρτησης

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x \tag{4.1}$$

στο διάστημα $[0, 2]$.

```
a=0; b=2
a1=a; b1=b
Nmax=30
tol=1.0e-6
f={x -> x**4-14*x**3+60*x**2-70*x}
df={x -> 4*x**3-3*14*x**2+2*60*x-70}
sol=[]
converge=false
iter=1
while(iter<=Nmax && !converge){
  x=(a1+b1)/2.0
  fx=f(x)
```

```
13    dfx=df(x)
14    sol.add(x)
15    printf("%d: x= %12.8f and f(x)= %12.8e \n",iter,x,fx)
16    if(abs(dfx)<tol || abs(a1-b1)<tol){
17      converge=true
18    }
19    if(dfx<0.0){
20      a1=x
21    }else{
22      b1=x
23    }
24    iter++
25  }
26
27  convergePlot = new PlotFrame()
28  pf=new plotfunction(sol)
29  convergePlot.addFunction(pf)
30  convergePlot.setMarker(true)
31  convergePlot.show()
```

4.2 Μέθοδος της χρυσής τομής

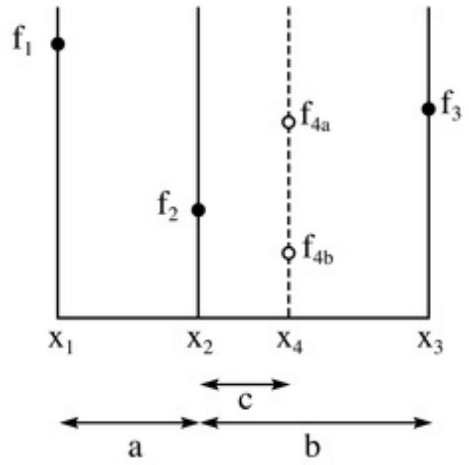
Η μέθοδος της χρυσής τομή[1] αφορά την εύρεση ακρότατων μιας συνάρτησης η οποία έχει ένα μοναδικό ακρότατο σε κάποιο πεδίο ορισμού. Σε αντίθεση από την περίπτωση εύρεση μιας ρίζας όπου δυο τιμές της συνάρτησης με αντίθετο πρόσημο ήταν αρκετές για να αναγνωρίσουμε την ύπαρξη ρίζας στο πεδίο αυτό, σε αυτή τη περίπτωση χρειαζόμαστε τρεις τιμές.

Το σχηματικό διάγραμμα της μεθόδου στην εικ. 4.2 απεικονίζει ένα βήμα της μεθόδου. Οι τιμές της συνάρτησης $f(x)$ βρίσκονται στον κατακόρυφο άξονα και στον οριζόντιο έχουμε την ανεξάρτητη μεταβλητή x. Η συνάρτηση έχει υπολογιστεί στα σημεία $f_1 = f(x_1), f_2 = f(x_2)$ και $f_3 = f(x_3)$. Εφόσον το f_2 είναι μικρότερο από τα f_1 και f_3, είναι φανερό ότι το ελάχιστο υπάρχει στην περιοχή από x_1 στο x_3.

Η επόμενη τιμή της f θα υπολογιστεί σε ένα σημείο στην περιοχή με μεγαλύτερο εύρος από τις $[x_1, x_2]$ και $[x_2, x_3]$, που είναι η δεύτερη. Από το διάγραμμα φαίνεται ότι αν η τιμής της f στο σημείο είναι η f_{4a} τότε το ελάχιστο θα βρίσκεται στην περιοχή $[x_1, x_4]$ και η νέα τριάδα σημείων θα είναι η (x_1, x_2, x_4). Ωστόσο, αν η τιμή της f στο σημείο είναι η f_{4b} τότε το ελάχιστο θα είναι στην περιοχή $[x_2, x_3]$ και η νέα τριάδα σημείων θα είναι η (x_2, x_4, x_3).

Για τον καθορισμό της θέσης x_4 η μέθοδος επιβάλλει το μήκος της περιοχής $[x_1, x_4]$ που είναι $a+c$ και αυτό της περιοχής $[x_2, x_3]$ μήκους b να είναι ίσα. Για

[1] https://en.wikipedia.org/wiki/Golden-section_search



Σχήμα 4.2: Διάγραμμα της αναζήτησης με την χρυσή τομή.

να ισχύει αυτό θα πρέπει $x_4 = x_1 + (x_3 - x_2)$. Συνεχίζει βέβαια να παραμένει το ερώτημα για το που θα τοποθετηθεί το σημείο x_2 σε σχέση με τα x_1 και x_3. Η απάντηση είναι ότι το x_2 θα εντοπίζεται έτσι ώστε αν $f(x_4) = f_{4a}$ να ισχύει η αναλογία,

$$\frac{c}{a} = \frac{a}{b}$$

ενώ αν $f(x_4) = f_{4b}$ να ισχύει η αναλογία,

$$\frac{c}{b-c} = \frac{a}{b}$$

απαλείφοντας το c από τις δυο προηγούμενες εξισώσεις θα έχουμε,

$$\left(\frac{b}{a}\right)^2 - \frac{b}{a} = 1 \xrightarrow{\phi=\frac{b}{a}} \phi^2 - \phi = 1$$

με λύση την,

$$\phi = \frac{1+\sqrt{5}}{2}$$

που είναι ο αριθμός της χρυσής αναλογίας. Οπότε η θέση του σημείου x_2 υπολογίζεται από τη σχέση,

$$x_2 = \frac{x_3 + \phi x_1}{1+\phi}$$

Ως συνθήκη τερματισμού προτείνεται η εξής παρακάτω,

$$|x_3 - x_1| < \epsilon(|x_2| + |x_4|)$$

Εδώ δίνεται σε απλό κώδικα Groovy/Climax η εύρεση του ελάχιστου της συνάρτησης

$$f(x) = x^4 - 14x^3 + 60x^2 - 70x \tag{4.2}$$

στο διάστημα $[0, 2]$.

```
x1=0; x3=2
phi=(1.0+sqrt(5.0))/2.0
Nmax=30
tol=1.0e-12
f={x-> x**4-14*x**3+60*x**2-70*x}
sol=[]
converge=false; x=(x1+x3)/2.0
iter=1
while(iter<=Nmax && !converge){
  x2=(x3+phi*x1)/(1+phi)
  x4=x1+(x3-x2)
  f2=f(x2); f4=f(x4)
  if(f2<f4){x=x2; fx=f2; x3=x4}else{x=x4; fx=f4; x1=x2;}
  sol.add(x)
  if(abs(x3-x1)<tol*(abs(x2)+abs(x4))){
  converge=true
  }
  iter++
}
println("minimum= "+fx+" at x= "+x)
convergePlot = new PlotFrame()
pf=new plotfunction(sol)
convergePlot.addFunction(pf)
convergePlot.setMarker(true)
convergePlot.show()
```

4.3 Μέθοδος Newton

Η μέθοδος Newton που είδαμε και στην ενότητα εύρεσης ριζών, εδώ για τον εντοπισμό ακρότατων, βασίζεται στην επαναληπτική σχέση,

$$x_i = x_{i-1} - \frac{f'(x_{i-1})}{f''(x_{i-1})}$$

με άλλα λόγια στο παρόν πλαίσιο αναζητά τις ρίζες της πρώτης παραγώγου $f'(x)$ της συνάρτησης $f(x)$.

Κεφάλαιο 5

Παρεμβολή

Το πρόβλημα της παρεμβολής συνίσταται στην εύρεση της $f(x)$ όταν γνωρίζουμε τιμές της συνάρτησης για κάποιες δεδομένες τιμές του ορίσματος x. Τα ζεύγη αυτά των δεδομένων τιμών θα συμβολίζονται ως $\big(x_i, f_(x_i)\big)$ για $i = 1, 2, \ldots, n$. Τα σημεία x_i δεν είναι σε καμιά περίπτωση απαραιτήτως ισαπέχοντα. Στο παρόν κεφάλαιο θα θεωρούμε ότι η $f(x)$ είναι μια πολυωνυμική συνάρτηση και η αντίστοιχη παρεμβολή θα αποκαλείται και *πολυωνυμική παρεμβολή*.

Σύμφωνα με το θεώρημα Weierstrass, μια συνεχής συνάρτηση $f(x)$ στο πεδίο $[a, b]$ μπορεί να προσεγγιστεί από ένα πολυώνυμο $P(x)$ με οποιαδήποτε επιθυμητή ακρίβεια ϵ. Δηλαδή υπάρχει $P(x)$ τέτοιο ώστε,

$$|f(x) - P(x)| < \epsilon, \forall x \in [a, b] \tag{5.1}$$

το παραπάνω θεώρημα βέβαια αν και αναφέρεται στην ύπαρξη του πολυωνύμου $P(x)$ δεν μας προσδιορίζει τις σχέσεις κατασκευής και υπολογισμού του.

Μια τέτοια πρώτη απάντηση μπορούμε να λάβουμε από το πολυώνυμο του Taylor, και αντίστοιχα του Maclaurin (βλ. π.χ. στη Wikipedia[1]),

$$f(x) \approx P_n(x) = f(\xi) + \frac{f'(\xi)}{1!}(x - \xi) + \frac{f''(\xi)}{2!}(x - \xi)^2 + \ldots + \frac{f^{(n)}(\xi)}{n!}(x - \xi)^n,$$

όταν το σημείο ξ ανήκει στο πεδίο ορισμού της f. Αντίστοιχα όταν $\xi = 0$,

$$f(x) \approx P_n(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \ldots + \frac{f^{(n)}(0)}{n!}x^n.$$

Είναι σημαντικό να επισημανθεί εδώ ότι η προσέγγιση είναι ακριβής μόνο για τιμές του x πλησίον του ξ.

[1] `https://en.wikipedia.org/wiki/Taylor_series`

5.1 Πολυώνυμα Lagrange

Εστω $x_0, x_1, \ldots, x_n$ ειναι $n+1$ διαφορετικα σημεια ενος διαστηματος $[a, b]$ και $f(x)$ μια πραγματικη συναρτηση της οποιας ειναι γνωστε οι τιμες $f(x_i)$ για καθε $i = 0, 1, \ldots, n$. Η πολυωνυμικη παερμβολη οριζεται απο ενα πολυωνυμο, εστω P_n βαθμου $\leq n$ το οποιο διερχεται απο τα σημεια απο τα σημεια $(x_i, f(x_i))$, δηλαδη $P_n(x_i) = f(x_i)$. Το πολυωνυμο θα δινεται απο τη σχεση,

$$\begin{aligned} P_n(x) &= l_0(x)f(x_0) + l_1(x)f(x_1) + \ldots + l_n(x)f(x_n) \\ &= \sum_{i=0}^{n} l_i(x)f(x_i) \end{aligned} \tag{5.2}$$

οπου

$$\begin{aligned} l_i(x) &= \frac{(x-x_0)(x-x_1\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)} \\ &= \prod_{0\leq j\leq n, j\neq i} \frac{x-x_j}{x_i-x_j} \end{aligned} \tag{5.3}$$

Για τα πολυώνυμα $l_i(x)$ ισχύει η σχέση

$$l_i(x_j) = \delta_{ij} = \begin{cases} 1, & \text{αν } i = j \\ 0, & \text{αν } i \neq j. \end{cases} \tag{5.4}$$

Έστω τα ζεύγη[2] δεδομένων που δίνονται στον πίνακα 5.1, να γίνει η προσέγ-

Πίνακας 5.1: Δεδομένα για τα ζεύγη σημείων $(x,f(x_i))$

x	0	20	40	60	80	100
$f(x)$	26.0	48.6	61.6	71.2	74.8	75.2

γιση της τιμής της συνάρτησης $f(x)$ στο σημείο x_p=55 με χρήση πολυωνύμων Lagrange[3].

[2]Για το αντίστοιχο παράδειγμα σε Matlab/Octave, https://www.youtube.com/watch?v=NZfd-EuBYyo

[3]Script at: https://eclass.teicrete.gr/modules/document/file.php/TA221/climax_files/lagrangepol.climax

```
x=[0,20,40,60,80,100]
fx=[26.0,48.6,61.6,71.2,74.8,75.2]
n=x.size()-1 // should be, n<=x.size()-1
xp=55
sm=0
(0..n).each{
   i=it
   pr=1
   (0..n).each{
      j=it
      if(j!=i)pr*=(xp-x[j])/(x[i]-x[j])
   }
   sm+=fx[i]*pr
}
fxp=sm
println ("value of f("+xp+") is : "+fxp)
```

5.2 Τύπος του Newton

Το πολυώνυμο προσέγγισης $P_n(x)$ μπορεί να γραφτεί με χρήση των διαιρεμένων διαφορών [5], ως εξής:

$$\begin{aligned} P_n(x) = & f[x_0] + f[x_1, x_0](x - x_0) \\ & + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \dots \\ & + f[x_0, x_1, \dots, x_n](x - x_0) \dots (x - x_{n-1}) \end{aligned} \tag{5.5}$$

Η σχέση της εξ. (5.5) είναι γνωστή και ως τύπος παρεμβολής του Newton.

Εδώ θα γίνει η προσέγγιση για τα ζεύγη του πίνακα 5.1 με χρήση των διαιρεμένων διαφορών και του τύπου παρεμβολής του Newton[4].

```
x=[0,20,40,60,80,100]
fx=[26.0,48.6,61.6,71.2,74.8,75.2]
n=x.size()-1
xtab=[]
x.each{xtab.add(it); xtab.add(0.0);}
divdif = new double[2*n+1][n+1]

j=0
for(i=0;i<2*n+1;i+=2){
   divdif[i][0]=fx[j++]
```

[4]Script at: https://eclass.teicrete.gr/modules/document/file.php/TA221/climax_files/divdif.climax

```
11 }
12
13 (1..n).each{
14   k=it
15   (0..(n - k)).each{
16     i=k+it*2
17     divdif[i][k]=(divdif[i-1][k-1]-divdif[i+1][k-1])/(xtab[i-k]-
       xtab[i+k])
18   }
19 }
20 /*
21 * // print divided differences table
22 (0..<2*n+1).each{
23   i=it
24   (0..<n+1).each{
25     j=it
26     printf("%8.6f ",divdif[i][j])
27   }
28   println(" ")
29 }*/
30
31 xp=55
32 pr=1.0
33 sm=divdif[0][0]
34 (1..n).each{
35   pr*=(xp-x[it-1])
36   sm+=divdif[it][it]*pr
37 }
38 fxp=sm
39 println ("value of f("+xp+") is : "+fxp)
40 }
```

5.3 Υπολογισμός συντελεστών πολυώνυμων προσέγγισης

Ο πιο άμεσος και εύκολος σχετικά τρόπος ώστε να υπολογίσουμε τους συντελεστές του πολυωνύμου προσέγγισης είναι από την επίλυση ενός αντίστοιχου γραμμικού συστήματος. Αν γράψουμε το πολυώνυμο $P_n(x)$ ως

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \dots a_n x^n \tag{5.6}$$

τότε με χρήση των δεδομένων για τα ζεύγη σημείων $(x_i, f(x_i)) = (x_i, P_n(x_i))$ μπορούμε να καταστρώσουμε τις πιο κάτω εξισώσεις.

$$\begin{aligned}
&a_0 + x_0 a_1 + x_0^2 a_2 + \cdots + x_0^n a_n = f(x_0)\\
&a_0 + x_1 a_1 + x_1^2 a_2 + \cdots + x_1^n a_n = f(x_1)\\
&\vdots\\
&a_0 + x_n a_1 + x_n^2 a_2 + \cdots + x_n^n a_n = f(x_n)
\end{aligned}$$

Το οποίο σύστημα με τη βοήθεια των πινάκων μπορεί να γραφτεί ως:

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}}_{\mathbf{b}}$$

από την επίλυση του οποίου μπορούμε να υπολογίσουμε το διάνυσμα $\mathbf{x}$ που θα εμπεριέχει τους συντελεστές $a_0, a_1, \dots, a_n$ και συνεπώς να καθορίσουμε το πολυώνυμο P_n της εξ. (5.6).

Κεφάλαιο 6

Αριθμητική ολοκλήρωση

Η προσεγγιστική αριθμητική τιμή[5] του ορισμένου ολοκληρώματος,

$$I(f) = \int_a^b f(x)\,dx \tag{6.1}$$

χρησιμοποιείται κυρίως, όταν

1. λόγω της πολύπλοκης μορφής του τύπου μιας συνάρτησης είναι δύσκολος, ή ακόμα και αδύνατος, ο θεωρητικός υπολογισμός του, και
2. δεν είναι γνωστός ο τύπος της συνάρτησης, αλλά μόνο οι τιμές της σε ορισμένα σημεία.

6.1 Απλοί κανόνες ολοκλήρωσης

Οι κανόνες αυτοί αριθμητικής ολοκλήρωσης ονομάζονται και Newton–Cotes. Ανάλογα με τον θεωρούμενο αριθμό σημείων παρεμβολής έχουμε τους παρακάτω κανόνες.

6.1.1 Κανόνας του ορθογωνίου

Έστω το ορισμένο ολοκλήρωμα

$$I(f) = \int_a^b f(x)\,dx \tag{6.2}$$

όπου η $f(x)$ μια συνεχής συνάρτηση στο $[a, b]$. Στη γενικότερη περίπτωση όπου δεν δίνεται ρητά η αναλυτική μορφή της $f(x)$, θεωρείται ότι θα είναι γνωστές οι

τιμές της σε $n+1$ διαφορετικά σημεία $x_0, x_1, \ldots, x_n$ στο $[a,b]$. Η παρεμβολή του Newton θα είναι,

$$\begin{aligned} f(x) \approx & P_n(x) = f[x_0] + f[x_0,x_1](x-x_0) + \ldots \\ & + f[x_0,x_1,\ldots,x_n](x-x_0)\cdots(x-x_{n-1}) \end{aligned} \tag{6.3}$$

Χρησιμοποιώντας ένα σημείο παρεμβολής, έστω το x_0 τότε προκύπτει,

$$f(x) \approx P_0(x) = f[x_0] = f(x_0) \tag{6.4}$$

οπότε

$$I(f) = \int_a^b f(x)\,dx \approx f(x_0)\int_a^b dx = (b-a)f(x_0) \tag{6.5}$$

που είναι γνωστό ως ο κανόνας του ορθογωνίου.

Ανάλογα με τη θέση x_0 διακρίνονται οι εξής περιπτώσεις:

- για $x_0 = a$, τότε $I(f) = \int_a^b f(x)\,dx \approx (b-a)f(a)$,
- για $x_0 = b$, τότε $I(f) = \int_a^b f(x)\,dx \approx (b-a)f(b)$,
- για $x_0 = (b+a)/2$, τότε $I(f) = \int_a^b f(x)\,dx \approx (b-a)f\left(\frac{a+b}{2}\right)$, που είναι γνωστός ως ο κανόνας του μέσου σημείου.

```
1 Midpoint={f,a,b->
2   h=(b-a)
3   x0=(a+b)/2
4   return h*f(x0)
5 }
```

6.1.2 Κανόνας του τραπεζίου

Αν έχουμε δύο σημεία για την παρεμβολή ($n=1$), οπότε

$$f(x) \approx P_1(x) = f(x_0) + f[x_0,x_1](x-x_0), \tag{6.6}$$

θέτοντας $x_0 = a$, $x_1 = b$ και $h = b-a$

$$I(f) \approx \int_a^b P_1(x)\,dx = \ldots = \frac{h}{2}\big(f(a)+f(b)\big). \tag{6.7}$$

που είναι γνωστό ως ο κανόνας του τραπεζίου.

```
Trapezoidal={f,a,b->
  h=(b-a)
  x0=a; x1=b
  return h*(f(x0)+f(x1))/2.0
}
```

6.1.3 Κανόνας του Simpson

Αν έχουμε τρία σημεία για την παρεμβολή ($n = 2$), οπότε

$$f(x) \approx P_2(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1), \tag{6.8}$$

θέτοντας $x_0 = a$, $x_1 = \frac{a+b}{2}$, $x_2 = b$ και $h = \frac{b-a}{2}$

$$I(f) \approx \int_a^b P_2(x)\,dx = \ldots = \frac{h}{3}\left(f(x_0) + 4f(x_1) + f(x_2)\right). \tag{6.9}$$

που είναι γνωστό ως ο κανόνας του Simpson.

```
Simpson={f,a,b->
  h=(b-a)/2
  x0=a; x1=(a+b)/2; x2=b
  return h*(f(x0)+4*f(x1)+f(x2))/3.0
}
```

6.1.4 Κανόνας του Simpson 3/8

Για τέσσερα σημεία παρεμβολής ($n = 3$), θέτοντας $x_0 = a$, $x_1 = \frac{2a+b}{3}$, $x_2 = \frac{(a+2b)}{3}$, $x_3 = b$ και $h = \frac{b-a}{8}$

$$I(f) \approx \int_a^b P_3(x)\,dx = \ldots = h\left(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right). \tag{6.10}$$

που είναι γνωστό ως ο κανόνας των 3/8 του Simpson.

```
Simpson3_8={f,a,b->
  h=(b-a)/8
  x0=a; x1=(2*a+b)/3; x2=(a+2*b)/3; x3=b
  return h*(f(x0)+3*f(x1)+3*f(x2)+f(x3))
}
```

Αριθμητικό παράδειγμα

Να υπολογιστεί με τους πιο πάνω κανόνες το ολοκλήρωμα

$$\int_0^{1.2} \frac{1}{\sqrt{(1+x^2)}}\, dx \tag{6.11}$$

και αν δοθεί το απόλυτο λάθος για κάθε ένα από αυτούς.

```
f={1/sqrt(1+it**2)}
a=0.0; b=1.2
Ian=1.015973
println("---------------------------------------")
println("Rule Integral Approximation  Error")
println("---------------------------------------")
println "Midpoint "+Midpoint(f,a,b)+" "+abs(Midpoint(f,a,b)-Ian)
println "Trapezoidal  "+Trapezoidal(f,a,b)+"  "+abs(Trapezoidal(
    f,a,b)-Ian)
println "Simpson  "+Simpson(f,a,b)+"  "+abs(Simpson(f,a,b)-Ian)
println "Simpson 3/8  "+Simpson3_8(f,a,b)+" "+abs(Simpson3_8(f,a
    ,b)-Ian)
println("---------------------------------------")
```

Αν εκτελέσουμε το πιο πάνω σκριπτ στο SDE θα λάβουμε ως έξοδο παρόμοια με:

```
---------------------------------------
Rule Integral Approximation Error
---------------------------------------
Midpoint 1.028991510855 0.0130185108550
Trapezoidal 0.9841106397986 0.03186236020131
Simpson 1.0140312205029 0.0019417794970688
Simpson 3/8 1.0152331350974 7.398649025975E-4
---------------------------------------
```

6.2 Σύνθετοι κανόνες ολοκλήρωσης

Για να αυξηθεί η ακρίβεια της αριθμητικής ολοκλήρωσης των απλών κανόνων θα πρέπει κανείς να Θεωρήσει μεγαλύτερη τάξη παρεμβολής μια κατεύθυνση η οποία δυσχεραίνει πολύ τη διαδικασία. Ένας πολύ πιο απλός και άμεσος τρόπος είναι ο διαμερισμός του διαστήματος ολοκλήρωσης $[a, b]$ σε επιμέρους υποδιαστήματα, ο υπολογισμός του ολοκληρώματος σε κάθε ένα από αυτά τα

υποδιαστήματα και η πρόσθεση των τιμών σε κάθε ένα από τα διαστήματα αυτά μεταξύ τους.

Αυτη τη διαδικασία έχουμε γράψει στο παρακάτω σκριπτ.

```
f={1/sqrt(1+it**2)}
a=0.0; b=1.2
Ian=1.015973
num=10
stepwise={If->
  val=0.0
  (1..num).each{
  xs=a+(it-1)*(b-a)/num; xe=xs+(b-a)/num
  val+=If(f,xs,xe)
  }
  return val
}

println("----------------------------------------")
println("Rule Integral Approximation  Error")
println("----------------------------------------")
println "Midpoint "+stepwise(Midpoint)+"  "+abs(stepwise(
    Midpoint)-Ian)
println "Trapezoidal  "+stepwise(Trapezoidal)+" "+abs(stepwise(
    Trapezoidal)-Ian)
println "Simpson  "+stepwise(Simpson)+" "+abs(stepwise(Simpson)-
    Ian)
println "Simpson 3/8  "+stepwise(Simpson3_8)+"  "+abs(stepwise(
    Simpson3_8)-Ian)
println("----------------------------------------")
```

6.3 Κανόνας ολοκλήρωσης Gauss

Υπό κατασκευή ...

Κεφάλαιο 7

Αριθμητική Επίλυση Συνήθων Διαφορικών Εξισώσεων

Προσαρμογή από αντίστοιχες σημειώσεις[1] της καθ. Χρυσούλας Τσόγκα για το μάθημα «ΕΜ 291/M236 – Αριθμητική Επίλυση Συνήθων Διαφορικών Εξισώσεων» του Εαρινού Εξαμήνου 2014-2015 στο τμήμα Μαθηματικών και Εφαρμοσμένων Μαθηματικών του Πανεπιστήμιου Κρήτης.

7.1 Επίλυση προβλήματος αρχικών τιμών με την μέθοδο του Euler

Σκοπός αυτού του εργαστηρίου είναι η παρουσίαση αριθμητικής επίλυσης για Προβλήματα Αρχικών Τιμών (Π.Α.Τ.) με την μέθοδο του Euler. Το πρόβλημα των αρχικών τιμών:

$$\begin{cases} y'(t) = f(t,y), \quad t \in [a,b] \\ y(a) = y_a \end{cases} \tag{7.1}$$

Το επόμενο βήμα μας είναι να προσπαθήσουμε να λύσουμε αριθμητικά το Π.Α.Τ. (7.1) με την μέθοδοEuler, η οποία δίνεται από τον τύπο

$$y^{n+1} = y^n + hf(t^n, y^n), \quad n = 1, 2, \ldots, N, \tag{7.2}$$

[1] `http://users.tem.uoc.gr/~tsogka/Courses/AESDE-spring2015/index.html`

όπου $h = (b-a)/N$ το βήμα μας και y^n η προσέγγιση της λύσης στη χρονική στιγμή t^n, όπου $t^n = t^0 + nh$.

Για να υλοποιήσουμε την μέθοδο του Euler στην Climax, θα δημιουργήσουμε ένα συναρτησιακό αντικείμενο (closure) που θα ονομάσουμε myeuler και προαιρετικά θα μπορούσαμε να το αποθηκεύσουμε σε ένα ξεχωριστό αρχείο, π.χ. το myeuler.climax[2], ο κώδικας του οποίου θα είναι όπως παρακάτω[3].

```
1 myeuler={a,b,y0,N,f->
2 h=(b-a)/N as double
3 y=[]; (0..N).each{y.add(0.0)}
4 y[0]=y0 as double
5 for(n in 1..N)y[n]=y[n-1]+h*f(a+h*(n-1),y[n-1]) as double
6 return y
7 }
```

Όπως βλέπουμε τα ορίσματα του συναρτησιακού αντικειμένου είναι το διάστημα ορισμού $[a, b]$ στο οποίο θα υπολογίσουμε τη λύση, η αρχική συνθήκη y_0, ο αριθμός των βημάτων N και τέλος η συνάρτηση (ή και συναρτησιακό αντικείμενο) f. Στην γραμμή 2 του κώδικα καθορίζεται το βήμα h, στην γραμμή 3 εκχωρούμε μια λίστα (διάταξη) στην μεταβλητή y και ορίζουμε την πρώτη τιμή ίση με την αρχική τιμή (γραμμή 4) $y[0] = y_0$, στην επαναληπτική δομή της γραμμής 5 υλοποιείται η σχέση (7.2). Το συναρτησιακό αντικείμενο τέλος, επιστρέφει στη γραμμή 6, την διάταξη (ArrayList) y που περιέχει τις διακριτές τιμές της προσεγγιστικής λύσης.

7.1.1 Παράδειγμα πρώτο

Θα προσπαθήσουμε τώρα να λύσουμε το παράδειγμα της παρακάτω εξίσωσης συνοδεία της αντίστοιχης αρχικής τιμής:

$$\begin{cases} y'(t) = -y, & t \in [0, T] \\ y(0) = 1.0, \end{cases} \tag{7.3}$$

το οποίο έχει αναλυτική λύση την $y(t) = e^{-t}$. Για να υπολογίσουμε τις προσεγγίσεις της σχέσης (7.2), εκτελούμε την πιο κάτω δέσμη εντολών.

[2] Το όνομα του αρχείου καθώς και η κατάληξη καθορίζονται αυθαίρετα.

[3] Αξίζει να σημειωθεί εδώ ότι στον παραπάνω κώδικα θα μπορούσε να είχε παραληφθεί η οδηγία "as double", σε αυτή όμως την περίπτωση ο προκαθορισμένος τύπος μεταβλητής θα ήταν ο BigDecimal αντί του πιο αποδοτικού αριθμητικού τύπου double, και ως συνέπεια θα ήταν πιθανό να εμφανιστούν αριθμητικά προβλήματα κατά την εκτέλεση.

```
t0=0.0 // define initial time
T=2.0 // define final time
y0=1.0 // initial value
N=16 // number of steps
exact={t->exp(-t)} // exact solution
f={t,y->-y} // rhs
approx=myeuler(t0,T,y0,N,f) // approximate solution returned by
    myeuler
h=(T-t0)/N as double
(0..N).each{println exact(t0+it*h)+" "+approx[it]} // print
    approximation results together with exact solution
```

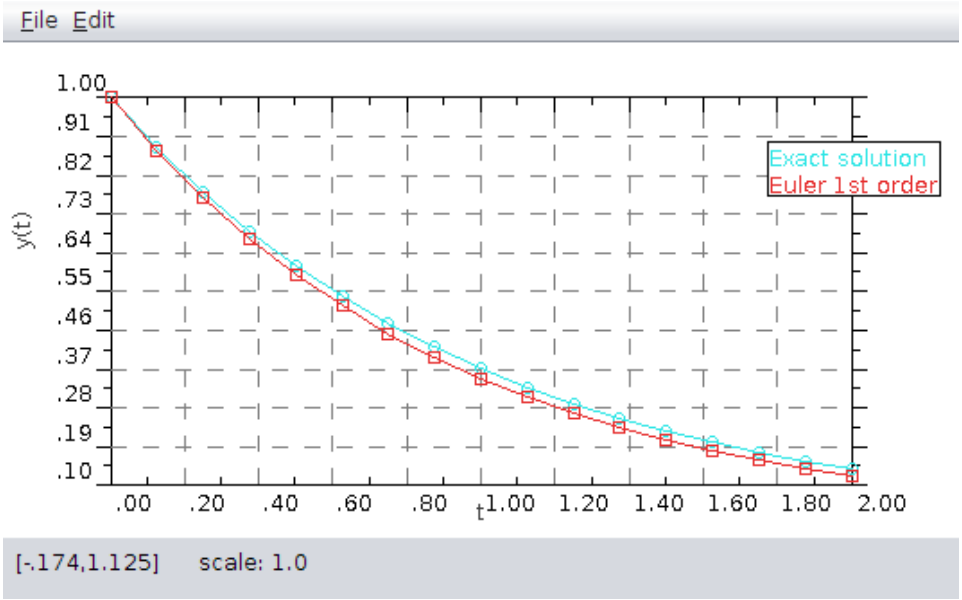
Εναλλακτικά,θα μπορούσαμε να τρέξουμε την συνάρτηση μας χωρίς να αναθέσουμε τα ορίσματα της σε κάποια μεταβλητή και θα την καλούσαμε όπως παρακάτω.

```
approx=myeuler(0.0,2.0,1.0,16,{t,y->-y})
```

7.1.2 Σφάλμα μεθόδου και τάξη ακρίβειας

Υπάρχουν δύο τρόποι για να εξετάσουμε το πόσο καλή είναι η προσεγγιστική μας λύση. Ο πρώτος τρόπος είναι η ποιοτική εξέταση, όπου κανείς μπορεί να παραθέσει τα αποτελέσματα της αναλυτικής λύσης με αυτά της προσεγγιστικής και να τα συγκρίνει. Σε αυτό θα βοηθούσε πολύ η δημιουργία ενός κοινού διαγράμματος στο πεδίο ορισμούυ. Στην περίπτωση της Climax αυτό είναι εφικτό με το αντικείμενο thePlot που είναι ένα προκαθορισμένο στιγμιότυπο της κλάσης PlotFrame. Οι συναρτήσεις που σχεδιάζονται σε ένα PlotFrame είναι στιγμιότυπα της κλάσης plotfunction.

```
thePlot.clear()

fp=new plotfunction(h,approx)
fp.setMarker(true)
fp.setName("Euler 1st order")
thePlot.addFunction(fp)

fp=new plotfunction(linspace(0,N*h,N+1),exact as DoubleFunction)
fp.setMarker(true)
fp.setMarkerStyle(1)
fp.setName("Exact solution")
thePlot.addFunction(fp)

// ------------------------------------
```



Σχήμα 7.1: Προσεγγιστική λύση με τη μέθοδο Euler (κόκκινη γραμμή) μαζί με την αναλυτική λύση (μπλε γραμμή) για την εξίσωση (7.3), για $N = 16$.

```
25 thePlot.setAutoColor(true)
26 thePlot.makeLegend()
27 thePlot.xLabel("t")
28 thePlot.yLabel("y(t)")
29 // ------------------------------------
30 thePlot.show()
```

Το αποτέλεσμα της ομάδας των παραπάνω εντολών είναι το διάγραμμα που φαίνεται στο Σχήμα 7.1

Ο δεύτερος τρόπος είναι να ποσοτικοποιήσουμε την ποιότητα της προσέγγισης υπολογίζοντας το σφάλμα της μεθόδου. Συνήθως, αυτό γίνεται στον τελικό χρόνο όπου υπολογίζουμε το σφάλμα E ως

$$E = |y^N - y(t^N)|.$$

Αυτό που πρέπει να προσέξουμε, είναι ότι η γραφική παράσταση και το σφάλμα που υπολογίσαμε, έγιναν για ένα συγκεκριμένο αριθμό σημείων ...

7.2 Μέθοδος του Euler δεύτερης τάξης

Στο επόμενο βήμα μας είναι να προσπαθήσουμε να λύσουμε αριθμητικά το Π.Α.Τ. (7.1) με την μέθοδοEuler δεύτερης τάξης, η οποία δίνεται από τον τύπο

$$y^{n+1} = y^n + h\left(f(t^n, y^n) + \frac{h}{2}f'(t^n, y^n)\right), \quad n = 1, 2, \ldots, N, \tag{7.4}$$

όπου $h = (b-a)/N$ το βήμα μας και y^n η προσέγγιση της λύσης στη χρονική στιγμή t^n, όπου $t^n = t^0 + nh$.

Παρόμοια του myeuler θα δημιουργήσουμε το συναρτησιακό αντικείμενο myeuler2nd

```
myeuler2nd={a,b,y0,N,f,df->
h=(b-a)/N as double
y=[]; (0..N).each{y.add(0.0)}
y[0]=y0 as double
for(n in 1..N){
xp=a+h*(n-1)
y[n]=y[n-1]+h*(f(xp,y[n-1])+h*df(xp,y[n-1])/2.0) as double
}
return y
}
```

7.3 Μέθοδοι Runge-Kutta

Στη μέθοδο Runge-Kutta δεύτερης τάξης (ή αλλιώς βελτιωμένη Euler μέθοδο) σε κάθε βήμα υπολογίζεται πρώτα μια βοηθητική τιμή ...

```
myrungekutta2nd={a,b,y0,N,f->
h=(b-a)/N as double
y=[] ; (0..N).each{y.add(0.0)}
y[0]=y0 as double
for(n in 1..N){
xp=a+h*(n-1)
ya=y[n-1]+h*f(xp,y[n-1]) as double
y[n]=y[n-1]+h/2*(f(xp,y[n-1])+f(xp,ya)) as double
}
return y
}
```

Τελος στη μεθοδο Runge-Kutta τέταρτης τάξης σε κάθε βήμα ...

```
myrungekutta4th={a,b,y0,N,f->
h=(b-a)/N as double
y=[] ; (0..N).each{y.add(0.0)}
y[0]=y0 as double
for(n in 1..N){
xn=a+h*n
xp=a+h*(n-1)
k1=h*f(xp,y[n-1]) as double
k2=h*f(xp+h/2,y[n-1]+k1/2) as double
k3=h*f(xp+h/2,y[n-1]+k2/2) as double
k4=h*f(xn,y[n-1]+k3) as double
y[n]=y[n-1]+1/6*(k1+2*k2+2*k3+k4) as double
}
return y
}
```

7.3.1 Παράδειγμα δεύτερο

Θα προσπαθήσουμε τώρα να λύσουμε το παράδειγμα της παρακάτω εξίσωσης συνοδεία της αντίστοιχης αρχικής τιμής:

$$\begin{cases} y'(x) = 1/x^2 - y/x - y^2, & x \in [1,2] \\ y(1) = -1.0, \end{cases} \tag{7.5}$$

το οποίο έχει αναλυτική λύση την $y(x) = -1/x$.

```
a=1.0 // define initial time
b=2.0 // define final time
y0=-1.0 // initial value
N=16 // number of steps
exact={x->-1/x} // exact solution
f={x,y->1/x**2-y/x-y**2} // rhs
df={x,y->-2/x**3+y/x-(1/x+2*y)*(1/x**2-y/x-y**2)}
approx=myeuler(a,b,y0,N,f) // approximate solution returned by
    myeuler
approx2nd=myeuler2nd(a,b,y0,N,f,df) // approximate solution
    returned by myeuler2nd
(0..N).each{println exact(a+it*(b-a)/N)+" "+approx[it]+" "+
    approx2nd[it]} // print approximation results together with
    exact solution
```

7.4 Συστήματα διαφορικών εξισώσεων 1ης τάξης

Σκοπός αυτής της ενότητας είναι η παρουσίαση αριθμητικής επίλυσης συστημάτων διαφορικών εξισώσεων (Σ.Δ.Ε.) με χρήση της Climax στο περιβάλλον SDE αξιοποιώντας τον μεταγλωττιστή της Groovy .

Ένα Σ.Δ.Ε. γράφεται ως εξής: Έστω $m \in \mathbb{N}, F : [a, b] \times \mathbb{R}^m \to \mathbb{R}^m$ και $y_0 \in \mathbb{R}^m$. Ζητείται συνάρτηση $y : [a, b] \times \mathbb{R}^m$ που να ικανοποιεί

$$\begin{cases} y'(t) = f(t, y(t)), & t \in [a, b] \\ y(a) = y_0 \end{cases} \tag{7.6}$$

ένα τέτοιο σύστημα λύνεται με αντίστοιχο τρόπο όπως στην περίπτωση των βαθμωτών συναρτήσεων, οπότε η μέθοδος Euler σε αυτή τη περίπτωση γράφεται:

$$y_{(k)}^{n+1} = y_{(k)}^{n} + hf(t^n, y_{(k)}^{n}), \quad n = 0, 2, \ldots, N, \quad k = 0, 2, \ldots, m \tag{7.7}$$

όπου $h = (b-a)/N$ το βήμα μας και $y_{(k)}^{n}$ η προσέγγιση της λύσης της k-ιοστής συνιστώσας στη χρονική στιγμή t^n, όπου $t^n = t^0 + nh$.

Τέτοια συστήματα προκύπτουν πολλές φορές όταν προσπαθούμε να λύσουμε Σ.Δ.Ε. υψηλότερης τάξης την οποία γράφουμε ισοδύναμα ως ένα σύστημα 1ης τάξης. Ένα χαρακτηριστικό παράδειγμα είναι το απλό εκκρεμές που θα παρουσιάσουμε παρακάτω.

7.5 Μετατροπή δευτεροβάθμιας εξίσωσης σε σύστημα πρώτου βαθμού (εξίσωση για το απλό εκκρεμές).

Ένα απλό εκκρεμές αποτελείται από σημειακή μάζα στο άκρο μιας ράβδου μήκους L που στηρίζεται σε κάποιο καρφί (χωρίς τριβή). Αν η βαρύτητα είναι η μόνη δύναμη που ενεργεί τότε η ταλάντωση του εκκρεμούς διαμορφώνεται από την εξίσωση,

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\sin(\theta) \tag{7.8}$$

όπου θ είναι η γωνιακή θέση της ράβδου, με $\theta = 0$ αν η ράβδος κρέμεται κάτω από το καρφί και $\theta = \pi$ αν η ράβδος βρίσκεται ακριβώς πάνω από το καρφί.

Έστω επιπλέον ότι $L = 50cm$ και $g = 9.81m/s^2$. Οι αρχικές συνθήκες είναι

$$\theta(0) = \theta_0, \text{ και } \frac{d\theta}{dt}(0) = 0. \tag{7.9}$$

Εάν η αρχική γωνία δεν είναι πολύ μεγάλη, τότε η προσέγγιση $\sin(\theta) = \theta$ μπορεί να χρησιμοποιηθεί και οδηγεί στο γραμμικό μοντέλο του ταλαντωτή

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\theta \tag{7.10}$$

το οποίο και λύνεται εύκολα αναλυτικά για να προκύψει

$$\theta(t) = \theta_0 \cos(t\sqrt{g/L}). \tag{7.11}$$

Η εξίσωση (7.10) μπορεί να γραφτεί ως σύστημα 1ης τάξης κάνοντας την έξης αλλαγή μεταβλητών: $y_1 = \theta$ και $y_2 = \frac{d\theta}{dt}$, οπότε και προκύπτει το σύστημα

$$\begin{cases} \dfrac{dy_1}{dt} = y_2, \\ \dfrac{dy_2}{dt} = -\dfrac{g}{L}y_1, \end{cases} \tag{7.12}$$

ή αλλιώς

$$\frac{dy}{dt} = Ay \text{ με } y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \text{ και } A = \begin{pmatrix} 0 & 1 \\ -\frac{g}{L} & 0 \end{pmatrix}. \tag{7.13}$$

7.6 Υλοποίηση της μεθόδου Euler για συστήματα

Παίρνοντας αφορμή από το παραπάνω παράδειγμα θα προσπαθήσουμε να υλοποιήσουμε με τη βοήθεια της Climax τη μέθοδο Euler στο σύστημα (7.13). Ξεκινάμε φτιάχνοντας ένα συναρτησιακό αντικείμενο, το οποίο θα μπορούσε να είναι μέσα σε ένα ξεχωριστό αρχείο, που θα περιέχει τον παρακάτω κώδικα.

```
1 myeuler={a,b,y0,N,f1,f2->
2 h=(b-a)/N as double
3 y = new double[2][N+1]
4 y[0][0]=y0[0]
5 y[1][0]=y0[1]
6 for(n in 1..N){
7 tp=a+h*(n-1)
```

```
y[0][n]=y[0][n-1]+h*f1(tp,y[0][n-1],y[1][n-1])
y[1][n]=y[1][n-1]+h*f2(tp,y[0][n-1],y[1][n-1])
}
return y
}
```

Με χρήση του πιο πάνω κώδικα για την μέθοδο Euler και το σύνολο της δέσμης εντολών που ακολουθεί μπορούμε να επιλύσουμε αριθμητικά το πρόβλημα του εκκρεμούς.

```
exact_th={t,th0-> L=0.5; g=9.81; return th0*cos(sqrt(g/L)*t) }
exact_th_t={t,th0-> L=0.5; g=9.81; return -th0*sqrt(g/L)*sin(
    sqrt(g/L)*t) }

f1={t,y1,y2->
L=0.5; g=9.81
return y2
}

f2={t,y1,y2->
L=0.5; g=9.81
return -g*y1/L
}

th0=PI/6.0
y0= new double[2]
y0[0]=th0
y0[1]=0.0

T=5*1.4185
t0=0.0
N=512

approx=myeuler(t0,T,y0,N,f1,f2) // approximate solution returned
    by myeuler
h=(T-t0)/N as double
(0..N).each{println exact_th(t0+it*h,th0)+" "+approx[0][it]}
}
```

Να σημειώσουμε εδώ πως παρόλο που υποβαθμίσαμε την δεύτερης τάξης εξίσωση σε ένα σύστημα δύο εξισώσεων πρώτης τάξης, η λύση που ζητάμε είναι αυτό που αντιστοιχεί στην πρώτη στήλη του διανύσματος y ή, αντίστοιχα, το y_1 όπως έχει δηλωθεί στο σύστημα (7.13).

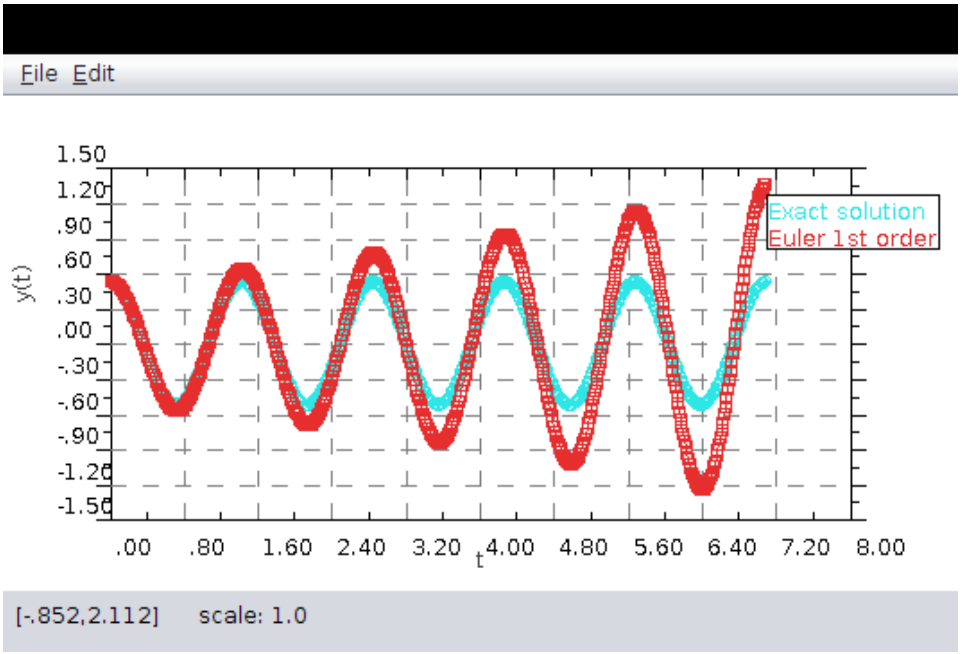
Για τη δημιουργία του διαγράμματος του που φαίνεται στο Σχήμα 7.2, χρησιμοποιούμε τον πιο κάτω κώδικα.

```
thePlot.clear()
approx1=[];(0..(N-1)).each{approx1.add(approx[0][it])}
fp=new plotfunction(h,approx1)
fp.setMarker(true)
fp.setName("Euler 1st order")
thePlot.addFunction(fp)

exact_thp=[]; (0..(N-1)).each{exact_thp.add(exact_th(t0+it*h,th0
    ))}
fp=new plotfunction(h,exact_thp)
fp.setMarker(true)
fp.setMarkerStyle(1)
fp.setName("Exact solution")
thePlot.addFunction(fp)

// ------------------------------------
thePlot.setAutoColor(true)
thePlot.makeLegend()
thePlot.xLabel("t")
thePlot.yLabel("y(t)")
// ------------------------------------
thePlot.show()
```

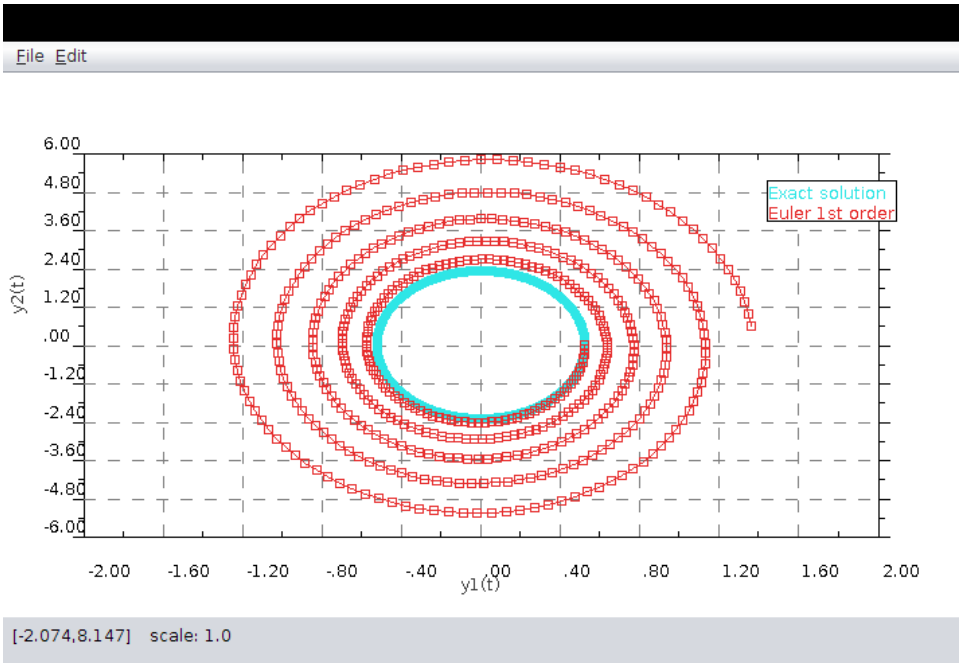
7.7 Διάγραμμα φάσης

Για τη δημιουργία του διαγράμματος (φάσης) που φαίνεται στο Σχήμα 7.3, χρησιμοποιούμε τον πιο κάτω κώδικα.

```
thePlot.clear()
approx0=[];(0..(N-1)).each{approx0.add(approx[0][it])}
approx1=[];(0..(N-1)).each{approx1.add(approx[1][it])}
fp=new plotfunction(approx0,approx1)
fp.setMarker(true)
fp.setName("Euler 1st order")
thePlot.addFunction(fp)

exact_thp=[]; (0..(N-1)).each{exact_thp.add(exact_th(t0+it*h,th0
    ))}
exact_th_tp=[]; (0..(N-1)).each{exact_th_tp.add(exact_th_t(t0+it
    *h,th0))}
fp=new plotfunction(exact_thp,exact_th_tp)
fp.setMarker(true)
fp.setMarkerStyle(1)
fp.setName("Exact solution")
```



Σχήμα 7.2: Προσεγγιστική λύση με τη μέθοδο Euler (κόκκινη γραμμή) μαζί με την αναλυτική λύση (μπλε γραμμή) για την εξίσωση (7.13), για $N = 512$.



Σχήμα 7.3: Διάγραμμα φάσης της λύσης με τη μέθοδο Euler (κόκκινη γραμμή) μαζί με την αναλυτική λύση (μπλε γραμμή) για την εξίσωση (7.13), για $N = 512$.

```
15 thePlot.addFunction(fp)
16
17 // ------------------------------------
18 thePlot.setAutoColor(true)
19 thePlot.makeLegend()
20 thePlot.xLabel("y1(t)")
21 thePlot.yLabel("y2(t)")
22 // ------------------------------------
23 thePlot.show()
```

Το επόμενο βήμα μας είναι να χρησιμοποιήσουμε περισσότερα σημεία για την προσέγγιση της λύσης μας και συγκεκριμένα χρησιμοποιούμε $N = 4096$ σημεία. Τότε παρατηρούμε, όπως φαίνεται και στο Σχήμα 7.4, πως η σπείρα που περιγράφει το διάγραμμα φάσης της προσέγγισής μας είναι πιο συγκεντρωμένο.

Σχήμα 7.4: Διάγραμμα φάσης της λύσης με τη μέθοδο Euler (κόκκινη γραμμή) μαζί με την αναλυτική λύση (μπλε γραμμή) για την εξίσωση (7.13), για $N = 4096$.

Κεφάλαιο 8

Αριθμητική Επίλυση Μερικών Διαφορικών Εξισώσεων

Υπό κατασκευή ...

Προσαρμογή από αντίστοιχες σημειώσεις[1] της καθ. Χρυσούλας Τσόγκα για το μάθημα «MEM253/EM292/M2513 – Αριθμητική Επίλυση Μερικών Διαφορικών Εξισώσεων» του Χειμερινού Εξαμήνου 2015-2016 στο τμήμα Μαθηματικών και Εφαρμοσμένων Μαθηματικών του Πανεπιστήμιου Κρήτης.

8.1 Πεπερασμένες διαφορές

Έστω το πρόβλημα συνοριακών τιμών

$$\begin{cases} -u''(x) + q(x)u(x) = f(x), \quad x \in [a,b] \\ u(a) = u(b) = 0 \end{cases} \tag{8.1}$$

όπου $q(x) \geq 0$ για κάθε $x \in [a,b]$. Θεωρούμε ομοιόμορφο διαμερισμό του διαστήματος $[a,b]$ με βήμα $h = (b-a)/N$, τα σημεία x_i του οποίου δίνονται από τη σχέση

$$x_i = a + ih, \quad i = 0, 1, \ldots, N.$$

Σκοπός μας είναι να υλοποιήσουμε μια αριθμητική μέθοδο η οποία να υπολογίζει μια προσεγγιστική λύση της (8.1) στα σημεία x_i της διαμέρισης, τις οποίες θα συμβολίζουμε με U_i, όπου $U_i \approx u(x_i), \quad i = 0, 1, \ldots, N$.

[1] http://users.tem.uoc.gr/~tsogka/Courses/AEMDE-fall2015/

Η μέθοδος που θα χρησιμοποιήσουμε εδώ για τον υπολογισμό της προσεγγιστικής λύσης ονομάζεται *μέθοδος πεπερασμένων διαφορών* και βασίζεται στην προσέγγιση της παραγώγου από πηλίκα διαφορών, τα οποία προέρχονται από τα αναπτύγματα Taylor.

8.1.1 Υλοποίηση σε Climax

Θα θεωρήσουμε το πρόβλημα

8.2 Μη-ομοιόμορφος διαμερισμός

8.3 Μέθοδοι πεπερασμένων διαφορών για παραβολικά προβλήματα

Σκοπός αυτού του εργαστηρίου είναι η υλοποίηση μιας αριθμητικής μεθόδου για την επίλυση μιας παραβολικής εξίσωσης όπως είναι η εξίσωση της θερμότητας.Θα ξεκινήσουμε θεωρώντας το πρόβλημα αρχικών/συνοριακών τιμών για την εξίσωση της θερμότητας, με ομογενείς συνοριακές συνθήκες τύπου Dirichlet

$$\begin{cases} u_t(t,x) = u_{xx}(t,x), & x \in [a,b], \quad \in [t_0,T_f], \\ u(0,x) = u_0(x), & x \in [a,b], \\ u(t,a) = u(t,b) = 0, & t \in [t_0,T_f], \end{cases} \tag{8.2}$$

8.4 Μέθοδοι πεπερασμένων διαφορών για υπερβολικά προβλήματα

...

Κεφάλαιο 9

Η Groovy και το SDE

Η γλώσσα την οποία χρησιμοποιούμε ώστε να εφοδιάσουμε το περιβάλλον SDE με κατάλληλα εργαλεία για αξιοποίηση των βιβλιοθηκών που συμπεριλαμβάνονται στην Climax είναι η Groovy. Ο πιο απλός τρόπος για να δοκιμάσουμε την Groovy είναι διαδικτυακά με το Groovy web console[1]. Επιπλέον για να χρησιμοποιήσει κανείς είτε την Groovy είτε τις δυνατότητες του συνόλου της βιβλιοθήκης Climax μπορεί να τρέξει «διαδικτυακά» το περιβάλλον SDE[2] μέσω της τεχνολογίας του Java web start.

Για να διαχωρίσουμε τις ενδογενείς μεθόδους της Groovy[3] από δυνατότητες με τις οποίες έχουμε εφοδιάσει τον συνδυασμό των πακέτων που απαρτίζεται από την Climax και/ή το SDE θα δηλώνουμε στο τίτλο των ενοτήτων αυτού του κεφαλαίου με ένα αστερίσκο (*) για την δεύτερη περίπτωση. Σημειώνεται πως σε αυτή την ενότητα δεν θα αναφερθούμε σχεδόν καθόλου στις δυνατότητες αξιοποίησης των υπολογιστικών μεθόδων που περιλαμβάνονται στην βιβλιοθήκη Climax, ενώ περισσότερο θα επικεντρώσουμε στην χρήση της Groovy σαν μια εναλλακτική γλώσσα και για υπολογισμούς και επιστημονικές/εκπαιδευτικές εφαρμογές.

Μια πληρέστερη και ταυτόχρονα συνοπτική περιγραφή μπορεί κανείς να βρεις στην επίσημη ιστοσελίδα της γλώσσας Groovy[4]

---

[1] `https://groovyconsole.appspot.com/`

[2] `http://symplegma.org/`

[3] Σημειώνουμε εδώ ότι η γλώσσα προγραμματισμού Groovy πολλές φορές αναφέρεται και ως ένα υπερσύνολο, ή αλλιώς μια επέκταση, της γλώσσας Java. Κάτω από αυτό το πρίσμα οι φοιτητές του Τ.Ε.Ι. Κρήτης, του τμήματος Μουσικής Τεχνολογίας & Ακουστικής, μπορούν να ανατρέξουν στην ύλη και τις αντίστοιχες σημειώσεις, της καθ. Χρ. Αλεξανδράκης, για το μάθημα «Ειδικά Θέματα Μουσικού Προγραμματισμού»

[4] `http://groovy-lang.org/documentation.html`

9.1 Μεταβλητές

Μεταβλητές (Variables), μπορούν να ονοματοδοτηθούν χρησιμοποιώντας κεφαλαίους ή πεζούς χαρακτήρες σε συνδυασμό με αριθμούς. Αποδεκτά ονόματα μπορεί να έχουν τη μορφή:

```
NetCost, Left2Pay, x3, X3, z25c5
```

Δεν επιτρέπεται να δίνουμε ονόματα τα οποία περιέχουν ειδικούς χαρακτήρες ή μεταβλητές που ξεκινάνε με αριθμό. Για παράδειγμα μη αποδεκτά ονόματα μεταβλητών είναι:

```
Net-Cost, 2pay, %x, *sign
```

Επιπλέον δεν πρέπει να χρησιμοποιηθούν ονόματα τα οποία χρησιμοποιούνται από την ίδια τη Groovy (ή και από το περιβάλλον SDE) όπως για παράδειγμα το PI=3.14159... $\simeq \pi$.

```
1 x=13; y=5*x
2 z=x**2+y
3 println x
4 println "y= "+y
```

Όπως φαίνεται πιο πάνω για να εμφανιστεί μια η τιμή κάποιας μεταβλητής θα πρέπει να χρησιμοποιήσουμε την εντολή `print` ή την `println`. Το ελληνικό ερωτηματικό (;) χρησιμοποιείται για να χωρίσουμε επιμέρους εντολές που δίνονται στην ίδια γραμμή.

9.2 Διατάξεις και πινάκες

Μια πολυ χρήσιμη οντότητα αποτελεί το αντικείμενο της διάταξης που θα χρησιμοποιούμε εδώ πολύ συχνά σε μια από τις δύο συνήθεις μορφές, αυτή του array καθώς και της ArrayList.

```
1 x=[]
2 x.add(1.0); x.add(4.5)
3 println x.getClass()+" "+x[0]
```

Μια χρήσιμη μέθοδος, όπως βλέπουμε και πιο πάνω, με την οποία μπορούμε να ανακτήσουμε το είδος κάποιας μεταβλητής είναι η getClass().

9.3 Εσωτερικές συναρτήσεις

Στις εσωτερικές συναρτήσεις της Groovy συμπεριλαμβάνονται οι τριγωνομετρικές συναρτήσεις σιν, ςος κ.α. καθώς και άλλες συναρτήσεις που χρησιμοποιούνται ευρέως όπως για παράδειγμα οι sqrt, log, exp κ.α.. Για να ακριβολογούμε οι συναρτήσεις αυτές για να είναι διαθέσιμες στην Groovy, όπως και στη Java, θα πρέπει να εισάγουμε (import static java.lang.Math.*) την προκαθορισμενη βιβλιοθήκη Math της Java. Στα πλαίσια της Climax αυτό έχει γίνει εκ των πρότερων ώστε οι συναρτήσεις και οι σταθερές (όπως για παράδειγμα οι PI, E, για τα $\pi \approx 3.14$ και $e \approx 2.718$, αντίστοιχα) να είναι άμεσα διαθέσιμες. Μερικά παραδείγματα χρήσης είναι:

```
1 x=PI**2
2 println sqrt(x)
```

Σε αντίθεση με την Matlab/Octave οι εσωτερικές συναρτήσεις δεν μπορούν να εφαρμοστούν σε διατάξεις ή διανύσματα/πίνακες.

9.4 Δομές ελέγχου

Οι δομές ελέγχου είναι κομμάτια κώδικα τα οποία αφορούν εντολές και διαδικασίες που θα εκτελεστούν ή όχι ανάλογα με το αν ισχύει κάποια συγκεκριμένη συνθήκη ή μια ομάδα συνθηκών. Χρήσιμη ενδογενής μεταβλητή των γλωσσών Java/Groovy είναι η λογική boolean μεταβλητή που παίρνει τις αυτονόητες τιμές true ή false.

Αν σε κάποιο σημείο έχουμε αναθέσει κάποια τιμή στην μεταβλητή x, τότε μπορούμε να κάνουμε ελέγχους σε αυτό, όπως

- x == 2 είναι το x ίσο με 2;
- x ! =2 δεν είναι το x ίσο με 2;
- x > 2 είναι το x μεγαλύτερο από 2;
- x < 2 είναι το x μικρότερο από 2;
- x >=2 είναι το x μεγαλύτερο από ή ίσο με 2;
- x <= 2 είναι το x μικρότερο από ή ίσο με 2;

Ιδιαίτερη προσοχή πρέπει να δοθεί στο γεγονός ότι ο έλεγχος για την ισότητα απαιτεί δύο σύμβολα ισότητας ==. Σε αντίθεση με την Matlab/Octave οι

εσωτερικές συναρτήσεις δεν μπορούν να εφαρμοστούν σε διατάξεις ή διανύσματα/πίνακες.

9.4.1 Δομή ελέγχου if/else

Η δομή ελέγχου if εξετάζει την αλήθεια μιας συνθήκης/πρότασης και προχωρά ή όχι σε κάποια ενέργεια. Η δομή αυτή μπορεί να συνοδεύεται και από ένα ακόλουθο else που δίνει την οδηγία του θα συμβεί αν δεν ισχύει η πρόταση ελέγχου. Αν δεν υπάρχει η επέκταση του else και δεν είναι αληθής η πρόταση τότε δεν θα γίνει καμιά περαιτέρω ενέργεια. Ένα παράδειγμα είναι και το παρακάτω.

```
// Initializing a local variable
int a = 2

//Check for the boolean condition
if (a<100) {
    //If the condition is true print the following statement
    println("The value is less than 100");
} else {
    //If the condition is false print the following statement
    println("The value is greater than 100");
}
```

9.4.2 Δομή ελέγχου switch

Μια άλλη δομή ελέγχου και απόφασης των Java/Groovy είναι η switch μέσω της οποίας για να συμβεί κάτι εξετάζονται επιμέρους περιπτώσεις για μια πρόταση. Ένα παράδειγμα δίνεται πιο κάτω.

```
//initializing a local variable
a = 2

//Evaluating the expression value
switch(a) {
    //There is case statement defined for 4 cases
    // Each case statement section has a break condition to exit
     the loop
    case 1:
        println("The value of a is One");
        break;
    case 2:
```

```
12          println("The value of a is Two");
13          break;
14      default:
15          println("The value is neither One or Two");
16          break;
17  }
```

9.5 Δομές επανάληψης

Μια δομή επανάληψης επαναλαμβάνει μια διαδικασία, ο αριθμός των επαναλήψεων εξαρτάται από την αλήθεια/ισχύ μιας συνθήκης/πρότασης. Η γλώσσα Groovy διαθέτει πλούσιο εύρος σε δομές επανάληψης, ενώ εδώ θα παρουσιάσουμε αυτές που συνηθέστερα θα χρησιμοποιούμε στις εφαρμογές των σημειώσεων αυτών.

9.5.1 Δομή επανάληψης for

Η δομή επανάληψης θα εκτελέσει την εντολή που είναι στο block της 5 φορές για τιμές του i από 0 έως και 4.

```
1  for(int i = 0;i<5;i++) {
2      println(i);
3  }
```

9.5.2 Δομή επανάληψης while

Όσο η συνθήκη ελέγχου είναι αληθής, εκτελούνται οι εντολές μέσα στο block της δομής while. Το αποτέλεσμα στο παράδειγμα που ακολουθεί θα είναι ίδιο με αυτό του παραδείγματος της δομής for που δόθηκε παραπάνω.

```
1  int i = 0;
2  while(i<5) {
3      println(i);
4      count++;
5  }
```

9.5.3 Δομή επανάληψης σε πεδίο τιμών range

Μια πολυ χρήσιμη οντότητα της Groovy είναι και η αριθμοσειρά (range) ακεραίων η οποία μπορεί να οριστεί ως (start..end) και περιλαμβάνει τους αριθμούς από τον start έως και τον end. Σε συνδυασμό με την συνάρτηση each (και με όρισμα ένα συναρτησιακό αντικείμενο (closure), το οποίο παρουσιάζεται σε επόμενη παράγραφο) μπορούμε να ορίσουμε μια δομή επανάληψης όπως στο παράδειγμα που ακολουθεί, με αποτέλεσμα ίδιο με τα πιο πάνω παραδείγματα.

```
1 (0..4).each{println(it)}
```

9.6 Συναρτήσεις

Μια μέθοδος ή αλλιώς συνάρτηση στη Groovy ορίζεται έτσι ώστε να παίρνει ένα ή κάποια ορίσματα και να επιστρέφει κάτι. Το αντικείμενο που θα επιστρέφει μπορεί να είναι και ένα κενό αντικείμενο (void) ή ακόμα και κάποιο απροσδιόριστο αντικείμενο (def). Η συνάρτηση αφού ολοκληρώσει τις διαδικασίες που δίνονται στο σώμα της επιστρέφει την έξοδο της με την εντολή return. Στη περίπτωση που η συνάρτηση έχει οριστεί ως (void) ή (def) η τελευταία return δήλωση μπορεί να παραληφθεί.

Στο παράδειγμα που ακολουθεί θα ορίσουμε και θα χρησιμοποιήσουμε τη συνάρτηση:

$$f(x, y) = \sin(2\pi x)\sin(2\pi y)$$

```
1 double f(double x, double y){
2  return Math.sin(2.0*Math.PI*x)*Math.sin(2.0*Math.PI*y)
3 }
4 println f(0.85,0.2)
```

Ο ορισμός για τους τύπους των παραμέτρων στο όρισμα της συνάρτησης είναι προαιρετικός. Στα πλαίσια του περιβάλλοντος SDE το όνομα της κλάσης Math μπορεί να παραληφθεί. Λαμβάνοντας αυτά υπόψη θα μπορούσαμε να γράψουμε σε πιο συμπτυγμένη μορφή, όπως παρακάτω.

```
1 double f(x, y){
2  return sin(2.0*PI*x)*sin(2.0*PI*y)
3 }
4 println f(0.85,0.2)
```

9.7 Συναρτησιακά αντικείμενα (closures)

Για πολλούς λόγους, η εξήγηση των οποίων ξεπερνά την εμβέλεια και το διδακτικό σκοπό του παρόντος τεύχους, εδώ προτιμάμε τη χρήση του συναρτησιακού αντικείμενου (closure) αντί της συνάρτησης (method), αν και κάποιοι από τους λόγους αυτούς πιθανώς να φανούν αυτονόητοι από το περιεχόμενο αυτών των σημειώσεων. Ένας, ίσως απλοϊκός, τρόπος να αντιληφθεί κανείς το αντικείμενο closure, είναι να το θεωρήσει ως μια συνάρτηση που μπορεί να περάσει ως όρισμα μέσα σε μια άλλη συνάρτηση. Από τον προηγούμενο ορισμό προκύπτει και η ονομασία που εδώ έχουμε επιλέξει στα ελληνικά ως *συναρτησιακό αντικείμενο*. Τα αντικείμενα αυτά είναι και μια από τις βασικές οντότητες, μαζί με τις δυνατότητες δυναμικού προγραμματισμού και του υψηλού βαθμού συμβατότητας με την Java, που καθιστούν την Groovy εξαιρετικά χρήσιμη. Ένα συναρτησιακό αντικείμενο ορίζεται μέσα σε αγκύλες με τις παραμέτρους εισόδου (ορίσματα) να χωρίζονται από το σώμα με το σύμβολο (−>). Τέλος, το συναρτησιακό αντικείμενο μπορεί να καταλήγει με μια δήλωση επιστροφής (return) ή και απουσία αυτής. Στην δεύτερη περίπτωση ως επιστροφή ορίζεται το υπολογισμένο μέγεθος από τη τελευταία διαδικασία που έχει γίνει μέσα στο σώμα του αντικειμένου.

Εδώ ως απλό παράδειγμα θα δώσουμε τον ορισμό της προηγούμενης τριγωνομετρικής συνάρτησης ως συναρτησιακό αντικείμενο.

```
1 f={x, y->
2  sin(2.0*PI*x)*sin(2.0*PI*y)
3 }
4 println f(0.85,0.2)
```

Τέλος αξίζει να σημειωθεί ότι ακόμα και όταν λείπουν οι μεταβλητές εισόδου το συναρτησιακό αντικείμενο της Groovy εχει μια προκαθορισμένη μεταβλητή με την ονομασία it. Αυτό μπορεί να φανεί στο παράδειγμα που ακολουθεί.

```
1 f={sin(2.0*PI*it[0])*sin(2.0*PI*it[1])}
2 println f([0.85,0.2])
```

9.8 Είσοδος και έξοδος δεδομένων σε και από αρχεία

Η Groovy παρέχει με μια ομάδα βοηθητικών μεθόδων για την επικοινωνία με και διαχείριση δεδομένων προς και από αρχεία. Μια βασική ενέργεια είναι αρχικά

να οριστεί ή να δημιουργηθεί αυτό το αρχείο

```
SomeFile = new File("somedirectory/Example.txt")
```

9.8.1 Αποθήκευση δεδομένων

Για να γράψει κανείς σε ένα αρχείο, για παράδειγμα στο SomeFile που δημιουργήσαμε προηγούμενα, μπορεί να χρησιμοποιήσει μεθόδους όπως αυτές στο παράδειγμα που ακολουθεί.

```
SomeFile.write "This is the first line\n"
SomeFile << "This is the second line\n"
SomeFile.leftShift "This is the third line\n"
SomeFile.append "This is the fourth line\n"
println SomeFile.text
```

Η μέθοδος write ουσιαστικά σβήνει ότι έχει γραφτεί στο αρχείο και γράφει από πάνω το αλφαριθμητικό περιεχόμενο του ορίσματος. Η μέθοδος leftShift ή append για την οποία η Groovy υιοθετεί τον συμβολικό τελεστή <<, γράφει το περιεχόμενο του ορίσματος, στο τέλος του αρχείου χωρίς να σβήνει το προηγούμενο περιεχόμενο.

9.8.2 Ανάκτηση δεδομένων

Για να διαβάσουμε δεδομένα από ένα αρχείο ο πιο απλός τρόπος είναι να χρησιμοποιήσουμε την μέθοδο text που επιστρέφει το περιεχόμενο του αρχείου σε μια αλφαριθμητική String μεταβλητή.

```
filetext=SomeFile.text
println filetext
```

ενώ αν θέλουμε μπορούμε να διαβάσουμε το αρχείο γραμμή προς γραμμή τοποθετώντας το σε μια λίστα με χρήση της μεθόδου readLines

```
lines = SomeFile.readLines()
lines.each{println it}
```

9.9 Αρχεία δέσμης εντολών (scripts) *

Η γλώσσα προγραμματισμού Groovy διαθέτει δυνατότητες μιας scripting programming language. Με το όρο script (αρχείο δέσμης εντολών) εννοούμε ένα αρχείο το οποίο περιέχει μια ομάδα εντολών ή οδηγιών σε γραφή που ακολουθεί τις συμβάσεις κάποιας συγκεκριμένης γλώσσας προγραμματισμού, με σκοπό τη διενέργεια συγκεκριμένων διαδικασιών από κάποιο ηλεκτρονικό υπολογιστικό σύστημα. Η Groovy μπορεί να διαβάσει και να διαχειριστεί τέτοια αρχεία που μπορεί να έχουν οποιαδήποτε κατάληξη. Στα πλαίσια του περιβάλλοντος SDE έχει επιλεγεί η σύμβαση τα αντίστοιχα αρχεία να έχουν κατάληξη .climax καθώς βασική βιβλιοθήκη και ταυτόχρονα και κύριος λόγος ανάπτυξης του περιβάλλοντος ήταν η διαχείριση και αξιοποίηση της Java βιβλιοθήκης υπολογιστικών μεθόδων υπό τον τίτλο Climax. Ουσιαστικά ο μηχανισμός που αναλαμβάνει να εκτελέσει τις εντολές που περιέχονται στα script αυτά αρχεία είναι το αντικείμενο GroovyShell.

Στο παράρτημα παραθέτονται ολοκληρωμένα script αρχεία climax τα οποία περιέχουν τα σενάρια (εντολών) για την αντιμετώπιση προβλημάτων που παρουσιάζονται σε αυτές τις σημειώσεις.

9.10 Μητρώα και διανύσματα *

Στο περιβάλλον του SDE ο προκαθορισμένος τύπος που χρησιμοποιείται για τους πίνακες (αλγεβρικά μητρώα) είναι αυτός που μας προσφέρεται από τη βιβλιοθήκη της JAMA : A Java Matrix Package[5]. Όσο αφορά τα διανύσματα εδώ νοούνται ως πίνακες με κατάλληλες διαστάσεις. Ταυτόχρονα υπάρχουν και ενσωματωμένες δυνατότητες επίλυσης γραμμικών συστημάτων και ανάλυσης του ιδιοσυστήματος. Παράδειγμα ορισμού και χρήσης δίνεται στην ομάδα εντολών που ακολουθεί.

```
1  order = 3
2  // define an array double[][]
3  da=new double[order][order]
4
5  for(i in 0..<order){
6      for(j in 0..<order){
7          da[i][j]=random()
8      }
9  }
10
```

[5] http://math.nist.gov/javanumerics/jama/

```
// use the above defined double array to equip a matrix M
M=Matrix(da)

// update array da for another use
for(i in 0..<order){
     for(j in 0..<order){
          da[i][j]=random()
     }
}

// use the above defined double array to equip a matrix K
K=Matrix(da)

M.print()
K.print()

(K+M).print()
(K-M).print()
(K*M).print()
M.inverse().print()
M.transpose().print()
```

Ακολουθεί παράδειγμα με τον κώδικα που απαιτείται για να λύσουμε το γενικευμένο ιδιοπρόβλημα,

$$(K - \lambda M)x = 0$$

```
Eigen= EigenDescomposition(M.inverse()*K)
EigenValues=Eigen.getRealEigenvalues()
EigenVectors=Eigen.getV()

(1..order).each{println EigenValues[it-1]}
EigenVectors.print()
```

Σημειώνεται εδώ πως η μέθοδος getRealEigenvalues() επιστρέφει μια διάταξη (double array) με όρους τις ιδιοτιμές λ_i, ενώ η getV() επιστρέφει ένα πίνακα του οποίου κάθε στήλη περιέχει και ένα ιδιοδιάνυσμα x_i.

9.11 Μιγαδικοί αριθμοί *

Στο περιβάλλον του SDE ο προκαθορισμένος τύπος που χρησιμοποιείται για τους μιγαδικούς αριθμούς είναι αυτός που μας προσφέρεται από την βιβλιοθήκη

της Apache Commons Math[6]. Παράδειγμα ορισμού και χρήσης δίνεται στην ομάδα εντολών που ακολουθεί.

```
c1= new Complex(1.0,2.0)
c2= new Complex(3.0,2.4)
println c1+c2
println c1-c2
println c1*c2
println c1/c2
println c1.conj() // c1.conjugate()
println c1.re() // c1.getReal()
println c1.im() // c1.getImaginary()
println c1.abs() // absolute value of c1
```

9.12 Διαγράμματα (plotting)*

Τα προς σχεδίαση, σε δομή διαγράμματος, αντικείμενα είναι τα plotfunction. Ο καθορισμός (κατασκευή) ενός τέτοιου αντικείμενου απαιτεί την δήλωση των διακριτών τιμών της συνάρτησης (τεταγμένες) μέσω μιας διάταξης αριθμών και προαιρετικά τις τιμές του οριζόντιου άξονα (τετμημένες) οι οποίες αν δεν δηλωθούν θεωρείται ότι αυξάνονται με μοναδιαίο βήμα. Εναλλακτικά, για τον οριζόντιο άξονα μπορεί να δοθεί ένας μόνο αριθμός που θα είναι το επαυξητικό βήμα ξεκινώντας από το μηδέν. Η δομή διαγράμματος[7] που σχεδιάζει τα πιο πάνω αντικείμενα ονομάζεται PlotFrame και μπορεί να φιλοξενεί αυθαίρετο πλήθος plotfunction. Στο παράδειγμα που ακολουθεί σχεδιάζουμε σε ένα κοινό διάγραμμα τις τριγωνομετρικές συναρτήσεις του ημίτονου και συνημίτονου.

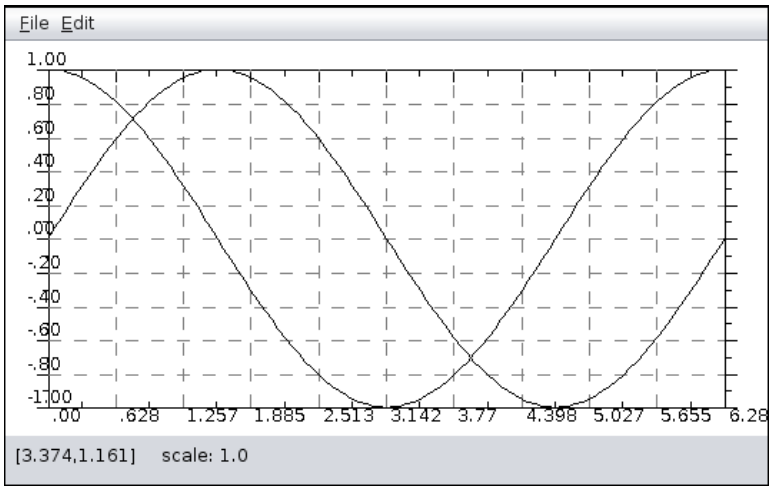
```
trigPlot = new PlotFrame()
n=100; dt=2.0*PI/n
t=new double[n+1]
s=new double[n+1]
c=new double[n+1]
(0..n).each{
    td=dt*it
    t[it]=td
    s[it]=sin(td)
    c[it]=cos(td)
}
```

[6] http://commons.apache.org/proper/commons-math/

[7] Στο περιβάλλον του SDE υπάρχει ένα προκαθορισμένο αντικείμενο δομής διαγράμματος με όνομα μεταβλητής thePlot.

```
trigPlot.addFunction(new plotfunction(dt,s))
trigPlot.addFunction(new plotfunction(t,c))
trigPlot.show()
```

Η χρήση του πιο πάνω κομματιού κώδικα θα εμφανίσει το διάγραμμα όπως περίπου φαίνεται στην πρώτη εικόνα 9.1 που ακολουθεί. Εμπλουτίζοντας (και



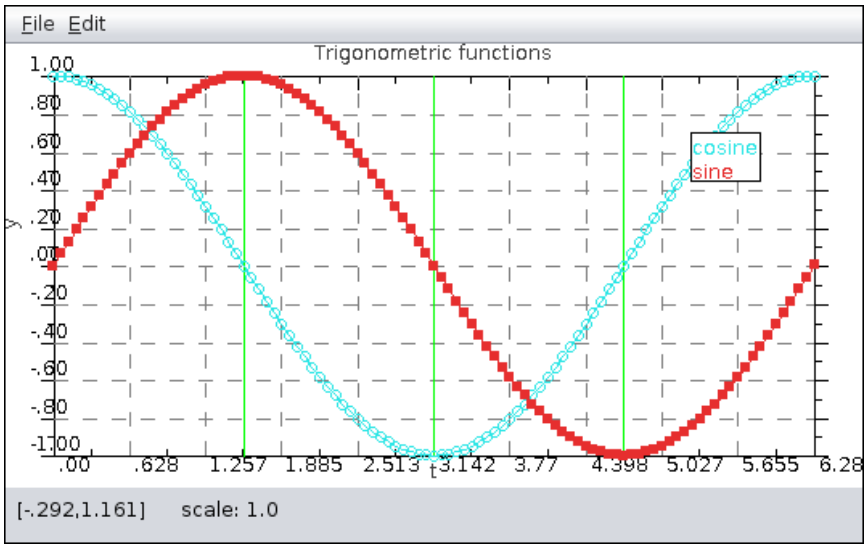
Σχήμα 9.1: Τριγωνομετρικές συναρτήσεις

σε κάποια σημεία τροποποιώντας) τον κωδικά όπως πιο κάτω, θα μπορούσαμε να εφοδιάσουμε περαιτέρω το διάγραμμα με πληροφορίες αλλά και κάποιες δυνατότητες μορφοποίησης.

```
trigPlot = new PlotFrame()
n=100; dt=2.0*PI/n
s=[];c=[]
(0..n).each{s[it]=sin(dt*it);c[it]=cos(dt*it)}
fp=new plotfunction(dt,s); fp.setMarker(true)
fp.setMarkerFill(true); fp.setName("sine")
trigPlot.addFunction(fp)
fp=new plotfunction(dt,c); fp.setMarker(true)
fp.setMarkerStyle(1);   fp.setName("cosine")
trigPlot.addFunction(fp)

trigPlot.setAutoColor(true)
trigPlot.Title("Trigonometric functions")
trigPlot.makeLegend()
trigPlot.xLabel("t")
trigPlot.yLabel("y")
```

```
trigPlot.vline(PI/2,java.awt.Color.green)
trigPlot.vline(PI,java.awt.Color.green)
trigPlot.vline(3*PI/2,java.awt.Color.green)
trigPlot.show()
```

Το αποτέλεσμα θα είναι παρόμοιο με το διάγραμμα της αντίστοιχης εικόνας 9.2.



Σχήμα 9.2: Τριγωνομετρικές συναρτήσεις, εμπλουτισμένο διάγραμμα σε σχέση με αυτό του σχήματος 9.1

9.13 Αναπαραγωγή ήχου*

Η κλάση που έχει διαμορφωθεί στο περιβάλλον του SDE για την αναπαραγωγή ήχου είναι η soundplayer και βασίζεται στο Java Sound API. Προκαθορισμένο αντικείμενο αναπαραγωγής ήχου είναι το στιγμιότυπο της κλάσης soundplayer με όνομα μεταβλητής theSP. Μοναδική μέθοδος αυτής της κλάσης είναι η playsound που στο όρισμα της παίρνει την διάταξη που περιέχει τις τιμές από τις οποίες θα παραχθεί ο ήχος και μία τιμή για την ένταση. Παράδειγμα χρήσης αποτελεί ο κωδικας του πιο κατω σκριπτ.

```
sound=[]
f1=440.0  // Hz
```

```
3  om1=2*PI*f1
4  // ratios r2=f2/f1 and r3=f3/f1
5  r2=4.0; r3=9.8;
6  // participation of each sine
7  c1=1.0; c2=1.0; c3=1.0;
8  om2=r2*om1; om3=r3*om1;
9  freq=44100
10 dt=1.0/freq
11 (0..freq).each{
12      sound.add(c1*sin(om1*it*dt)+c2*sin(om2*it*dt)+c3*sin(om3*it*
       dt))
13 }
14 theSP.playsound(sound)
```

Βιβλιογραφία

[1] Μ. Γουσίδου-Κουτίτα. *Ανώτερα εφαρμοσένα μαθηματκά και αριθμητικές μέθοδοι*. Εκδόσεις Χριστοδουλίδη, Θεσσαλονίκη, 2004.

[2] Graff, Karl F. *Wave motion in elastic solids*. Oxford University Press, 1973.

[3] Χ. Παναγιωτόπουλος και Π. Κολιόπουλος. *Εγχειρίδιο δυναμικής των κατασκευών*. Εκδόσεις Σοφία, Θεσσαλονίκη, 2007.

[4] Σ. Παπαϊωάννου και Χ. Βοζίκης. *Αριθμητική Ανάλυση*. ΣΕΑΒ, Αθήνα, 2015. Διαθέσιμο στο: `http://hdl.handle.net/11419/845`.

[5] Α. Μπράτσος. *Μαθήματα εφαρμοσμένων μαθηματικών*. ΣΕΑΒ, Αθήνα, 2015. Διαθέσιμο στο: `http://hdl.handle.net/11419/438`.

[6] Press, W.H. and Teukolsky, S.A. and Vetterling, W.T. and Flannery, B.P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[7] Α. Πρωτοπαπάς. *Βελτιστοποίηση τεχνικών συστημάτων*. ΣΕΑΒ, Αθήνα, 2015. Διαθέσιμο στο: `http://hdl.handle.net/11419/5906`.