Η Ομάδα μας 57
Κωνσταντινος Χουλιαρας, 1505
Ιωαννης Ασταλ, 1448

Εργασία Τρίτη

1 Ευκλείδιος αλγόριθμος και συνεχή κλάσματα για ρητούς αριθμούς

Τα συνεχή κλάσματα παίζουν σημαντικό ρόλο σε διάφορες περιοχές των Μαθηματικών. Οπως θα δούμε, τα συνεχή κλάσματα αποτελούν την βάση της πιο γρήγορης και αποτελεσματικής μεθόδου για την απομόνωση των πραγματικών ριζών πολυωνύμων με ρητούς συντελεστές. Σε αυτήν την ενότητα εισάγουμε τα συνεη κλάσματα με την βοήθεια του Ευκλείδειου αλγορίθμου.

Εστω λοιπόν R μία Ευκλείδεια περιοχή, $\alpha_{0,}\, \alpha_1 \in R$ και $q_{i,}\, \alpha_i \in R,\ 1 \leqslant i \leqslant L$ τα πηλίκα και υπόλοιπα στον κλασσικό Ευκλείδειο αλγόριθμο για α_0, α_1. Τότε, απαλείφοντας διαδοχικά τα υπόλοιπα έχουμε:

$$\frac{\alpha_0}{\alpha_1} = \frac{q_1 \cdot \alpha_1 + \alpha_2}{a_1} = q_1 + \frac{\alpha_2}{\alpha_1} = q_1 + \frac{1}{\frac{a_1}{a_2}} = q_1 + \frac{1}{q_2 + \frac{a_3}{a_2}} =$$

$$= q_1 + \frac{1}{q_2 + \frac{1}{\frac{a_2}{a_3}}} = \cdots = q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \frac{1}{\ddots}}}$$

Αυτό είναι το ανάπτυγμα του $\frac{\alpha_0}{\alpha_1}$ σε ***συνεχή κλάσματα*** (continued fractions expansion) και συμβολίζεται με $\frac{a_0}{a_1} = \{q_1, q_2, \ldots, q_L\}$. Οι αριθμοί $q_1,\ q_2,\ \ldots.,\ q_l$ λέγονται ***μερικά πηλίκα*** (partial quotients) και είνται όλοι του θετικοί. Στην συνέχεια το ανάπτυγμα ενός αριθμού σε συνεχή κλάσματα θα σθμβολίζεται ως $\{c_1, c_2, \ldots., c_L\}$. Αν χρησιμοποιήσουμε τους πρώτους k αριθμούς $\{c_1,\ \ldots.,\ c_k\}$, $k < L$, για να προσεγγίσουμε το $\frac{\alpha_0}{\alpha_1}$, τότε έχουμε την k-στή συγκλίνουσα (k-th convergent) του $\frac{a_0}{a_1}$. Στο Xcas έχουμε την συνάρτηση convert(_, confrac)

```
> convert(8/5, confrac)
```

$$[1, 1, 1, 2]$$

Σημειώνουμε πως ο Ευκλείδειος αλγόριθμος μπορεί να χρησιμοποιηθεί μόνο για το ανάπτυγμα ρητών αριθμών σε συνεχή κλάσματα.

Ασκηση 1. Να τροποποιήσετε τον Ευκλείδειο αλγόριθμο του Project_1 ώστε να σας επιστρέφει την λίστα των μερικών πηλίκων και να αναπτύξετε σαν παράδειγμα το κλάσμα $\frac{21}{13}$ σε συνεχή κλάσματα.

Λύση 1.

```
>>> from sympy import ZZ
    from sympy.matrices.matrices import  *
    from fractions import Fraction

    Cf_list = []
    def new_euclid(klasma):

        klasma = klasma.replace("/", " ")
        klasma = klasma.split(" ")
        a = int( klasma[0])
        b = int(klasma[1])
        Cf_list = CF_euclid(a,b)
        print(Cf_list)

    def CF_euclid(a, b):

        if b > a:
            return CF_euclid(b, a)

        if (ZZ.rem(a, b)) == 0:
            Cf_list.append((ZZ.quo(a,b)))
            return Cf_list

        Cf_list.append((ZZ.quo(a,b)))
        return CF_euclid(b, (ZZ.rem(a, b)))
```

```
>>> new_euclid("21/13")
  [1, 1, 1, 1, 1, 2]
>>>
```

Σχόλιο 1. Η python , τουλάχιστον στο επίπεδο που εμείς την ξέρουμε, δεν δέχεται σαν όρισμα σε συνάρτηση κλάσμα αλλά το μεταφράζει σε αριθμό κινητής υποδιαστολής(float). Για να παρακάμψουμε λοιπόν αυτή την αδυναμία, δίνουμε στη συνάρτησή μας το κλάσμα σε μορφή string και έπειτα από αυτό κρατάμε τους αριθμούς ώστε να δωθούν στον ευκλείδιο αλγόριθμο.

2 Συνεχή κλάσματα για πραγματικούς αριθμούς

Για την εύρεση του αναπτύγματος σε συνεχή κλάσματα ενός πραγματικού αριθμού α εργαζόμαστε ως εξής: Θέτουμε $\alpha_1 = \alpha,\ c_1 = \lfloor a_1 \rfloor,\ \beta_2 = \alpha_1 - c_1,$ $\alpha_2 = \frac{1}{\beta_2}$, και γενικά $c_i = \lfloor a_i \rfloor, \beta_{i+1} = a_i - c_i, a_{i+1} = \frac{1}{\beta_{i+1}}$. Προσέξτε πως για όλα τα i έχουμε $0 \leqslant \beta_i \leqslant 1$, και το ανάπτυγμα σταματάει οταν $\beta_i = 0$, που συμβαίνει εάν και μόνον εάν $a \in \mathbb{Q}$.

Περισσότερες πληροφορίες για τα συνεχή κλάσματα βρίσκονται στο αρχείο 34_EA+CF_P.nb, που ανοίγει με το Wolfram CDF Player (free download).

Ασκηση 2. Να γράψετε ένα πρόγραμμα και να υπολογίσετε το ανάπτυγμα σε συνεχή κλάσματα των αριθμών π και $\sqrt{3}$.

Λύση 2. Καθώς το β_i και για το π και για $\sqrt{3}$ δε θα μηδενίσει επιλέξαμε να αφήσουμε το πρόγραμμα να τρέξει για 500 επαναλήψεις. Δοκιμάσαμε και για 10000 αλλά το Texmacs σταματούσε να λειτουργεί.

```
>>> from sympy import floor, pi
    from sympy.matrices.matrices import  *

    def Cont_Fractions(float_numb):

        CF_list = []
        a1 = float(float_numb)
        print(a1)
        i = 0
        b2 = 1
        while i < 500 and b2 != 0 :
            c1 = floor(a1)
            CF_list.append(c1)
            b2 = a1 - c1
            a1= 1 / b2
            i = i + 1
        print(CF_list)


>>> Cont_Fractions(sqrt(3))

   1.73205080757
   [1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
   2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 13, 13, 12, 1, 1,
   17, 1, 1, 5, 15, 6, 57, 1, 1, 1, 3, 66, 1, 8, 1, 1, 4,
   1, 1, 1, 9, 3, 1, 2, 2, 1, 1, 2, 2, 3, 22, 1, 20, 15,
   1, 4, 3, 1, 2, 2, 3, 1, 1, 1, 51, 1, 1, 1, 1, 1, 4, 2,
   1, 1, 1, 1, 3, 3, 1, 1, 2, 3, 95, 2, 1, 49, 1, 1, 1, 1,
   1, 4, 3, 3, 13, 5, 1, 43, 3, 1, 2, 2, 14, 5, 27, 1, 2,
   3, 1, 5, 3, 1, 2, 12, 2, 19, 2, 1, 12, 3, 16, 1, 14, 1,
   6, 1, 27, 1, 1, 1, 1, 10, 2, 1, 8, 1, 1, 1, 2, 2, 1, 1,
   14, 1, 17, 17, 1, 1, 7, 9, 254, 1, 1, 1, 1, 4, 1, 10,
   2, 1, 2, 3, 5, 4, 10, 5, 19, 1, 13, 15, 1, 1, 4, 2, 1,
   228, 1, 1, 13, 1, 2, 3, 4, 1, 3, 5, 1, 3, 2, 3, 1, 4,
   1, 2, 1, 5, 1, 3, 1, 1, 1, 32, 1, 1, 2, 19, 28, 1, 12,
   3, 6, 1, 3, 1, 3, 5, 1, 4, 1, 4, 1, 2, 2, 20, 1, 3, 3,
   3, 2, 2, 2, 12, 1, 20, 1, 7, 11, 1, 5, 7, 1, 1, 145, 1,
   1, 1, 67, 104, 3, 1, 2, 2, 18, 2, 2, 13, 1, 56, 1, 2,
   1, 2, 1, 41, 1, 24, 1, 4, 2, 5, 3, 2, 6, 1, 4, 1, 6, 1,
```

```
12, 7, 1, 1, 2, 1, 2, 1, 1, 1, 18, 1, 1, 3, 1, 1, 6, 3,
11, 3, 6, 1, 1, 1, 1, 1, 1, 1, 49, 1, 1, 7, 2, 97, 4,
3, 12, 4, 2, 1, 8, 2, 1, 2, 69, 1, 56, 8, 1, 2, 27, 1,
2, 6, 29, 4, 1, 1, 1, 4, 2, 1, 1, 1, 1, 17, 1, 5, 11,
2, 1, 3, 204, 1, 1, 2, 1, 6, 1, 33, 1, 6, 1, 2, 1, 3,
2, 2, 1, 2, 2, 1, 35, 7, 1, 1, 2, 1, 1, 21, 1, 1, 2, 2,
6, 1, 1, 2, 2, 4, 1, 2, 1, 82, 1, 1, 1, 7, 2, 3, 12,
17, 60, 2, 1, 3, 1, 1, 1, 8, 6, 1, 1, 1, 11, 3, 6, 1,
4, 1, 10, 9, 5, 1, 2, 1, 4, 1, 1, 6, 2, 6, 2, 1, 10,
59, 1, 19, 1, 2, 1, 7, 7, 4, 14, 2, 3, 3, 2, 5, 1, 3,
5, 1, 1, 2, 1, 13, 5, 5, 18, 1, 7, 1, 3, 3, 12, 6, 4,
47, 1, 6, 1, 14, 14, 1, 1, 1, 3, 3, 2, 1, 4, 1, 10, 1,
1]
```

```
>>> Cont_Fractions(pi)
```

```
3.14159265359
[3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 3, 3, 23,
1, 1, 7, 4, 35, 1, 1, 1, 2, 3, 3, 3, 3, 1, 1, 14, 6, 4,
5, 1, 7, 1, 5, 1, 1, 3, 18, 2, 1, 2, 4, 2, 96, 2, 3, 2,
1, 1, 6, 1, 6, 2, 5, 64, 1, 2, 3, 1, 17, 5, 1, 12, 3,
2, 1, 1, 1, 1, 2, 2, 1, 4, 1, 1, 2, 2, 22, 1, 2, 1, 6,
1, 16, 1, 2, 3, 2, 4, 2, 5, 2, 3, 1, 1, 3, 2, 1, 7, 6,
4, 4, 3, 1, 61, 20, 11, 4, 1, 1, 4, 3, 1, 1, 3, 2, 1,
2, 1, 13, 2, 12, 2, 1, 1, 1, 1, 3, 1, 1, 1, 5, 10, 8,
9, 4, 1, 5, 1, 1, 2, 4, 1, 7, 3, 5, 4, 66, 13, 3, 1, 1,
6, 32, 1, 5, 4, 4, 6, 1, 2, 4, 1, 1, 1, 1, 2, 2, 1, 1,
1, 7, 2, 1, 2, 92, 2, 1, 5, 4, 2, 13, 2, 1, 1, 22, 2,
1, 3, 4, 6, 1, 22, 11, 3, 1, 1, 2, 2, 5, 1, 14, 8, 10,
3, 2, 1, 5, 8, 4, 7, 2, 4, 2, 1, 2, 1, 2, 1, 1, 5, 1,
3, 1, 2, 2, 2, 1, 1, 4, 2, 14, 1, 1, 6, 2, 2, 1, 1, 2,
1, 15, 2, 3, 2, 3, 53, 56, 4, 2, 1, 7, 1, 55, 1, 2, 7,
2, 9, 1, 46, 2, 15, 37, 7, 34, 1, 2, 1, 5, 1, 1, 2, 2,
4, 1, 2, 4, 1, 1, 2, 1, 9, 5, 3, 3, 4, 2, 6, 2, 2, 2,
3, 5, 1, 1, 4, 2, 21, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 3,
1, 33, 1, 10, 2, 1, 8, 4, 3, 1, 1, 1, 6, 1, 1, 1, 15,
1, 2, 4, 264, 4, 1, 2, 1, 27, 1, 10, 1, 23, 1, 4, 7, 1,
4, 2, 5, 4, 4, 6, 4, 2, 1, 2, 8, 1, 6, 6, 1, 1, 6, 4,
3, 1, 2, 151, 1, 1, 22, 1, 4, 2, 2, 2, 1, 72, 1, 1, 6,
85, 3, 1, 1, 1, 3, 4, 2, 3, 4, 7, 3, 16, 1, 1, 5, 1, 1,
```

```
3, 1, 1, 3, 2, 2, 3, 5, 24, 3, 2, 1, 6, 22, 1, 259, 5,
4, 2, 1, 1, 3, 3, 13, 1, 4, 1, 47, 31, 1, 6, 5, 95, 1,
1, 1, 1, 1, 1, 1, 1, 4, 2, 2, 5, 1, 7, 1, 2, 26, 1, 2,
593, 27, 5, 2, 7, 1, 1, 1, 2, 9, 1, 2, 2, 1, 2, 1, 1,
1, 1, 2, 2, 2, 1, 4, 1, 1, 1, 7, 3, 51, 1, 3, 1, 7, 1,
6, 1, 1, 1, 1, 1, 5, 1, 1, 129, 1]
```

3 Πάνω και κάτω φράγματα στις τιμές των πραγματικών ριζών πολυωνύμων

Στο αρχείο *Linear and Quadratic Complexity Bounds on the Values of the Positive Roots of Polynomials*.pdf υπάρχουν οι αλγόριθμοι με γραμμική πολυπλοκότητα C (Cauchy), K (Kioutselidis), FL (first-λ), LM (Local Max), και οι αντίστοιχοι αλγόριθμοι με τετραγωνική πολυπλοκότητα CQ, KQ, FLQ και LMQ. Αν εξαιρέσουμε την μέθοδο C, του Cauchy, που την έχετε όλοι προγραμματίσει για την 2η Εργασία μας μένουν 7 μέθοδοι με αρίθμηση $1 \Rightarrow K$, $2 \Rightarrow$ FL, $3 \Rightarrow$ LM, $4 \Rightarrow$ CQ, $5 \Rightarrow$ KQ, $6 \Rightarrow$ FLQ, 7ή$0 \Rightarrow$ LMQ.

Ασκηση 3. Η κάθε ομάδα να προγραμματίσει την μέθοδο που προκύπτει απο τον τύπο Z.rem(αριθμός ομάδας, 7)

$$\mathrm{Z.rem}(57, 7) = 1$$

Τη μέθοδο αυτή θα χρησιμοποιήσετε για τον υπολογισμό πάνω ή κάτω φράγματος στις τιμές των πραγματικών ριζών πολυωνύμων. Δοκιμάστε την στο πολυώνυμο $x^3 + 10^{100}x^2 - 10^{100}x - 1$ για τον υπολογισμό πάνω φράγματος

Λύση 3. Η λύση που ακολουθεί βασίζεται στη μέθοδο του Κιουστελίδη με εφαρμογή του Θεωρήματος 5 που παρουσιάζεται στο αρχείο Linear and Quadratic Complexity Bounds on the Values od Positive Roots of Polynomials.pdf (σελ. 7)

```
>>>
>>> from sympy import symbols
    from sympy.polys import *

    x = symbols('x')

    def K_method (f):

        d = degree(f,x)
        Coef_list = []
        Neg_list = []
        Coef_list = Poly(f, x).coeffs()
        i = d
        Neg_el = 0

        while i >= 0 :

            if Coef_list[i] < 0:
                Neg_list.append(Coef_list[i])
                Neg_el = Neg_el + 1
            else :
                Neg_list.append(0)

            i = i - 1

        for i in range (Neg_el):
            k = d - i
            rtn =
    (((-Neg_list[i])/(Coef_list[0]*2**(-k)))**(1/k))

        print("The upper bound is: ")
        print(float(rtn))


>>> fop = x**3+10**100*x**2-10**100*x-1
>>> K_method(f)

   Traceback (most recent call last):
   NameError: name 'f' is not defined
```

```
>>>
```

4 Μέθοδοι για την απομόνωση των πραγματικών ριζών με το θεώρημα του Vincent

Οπως βλέπετε από το αρχείο, *Swansea_2018.pdf*, από το θεώρημα του *Vincent* προκύπτουν 3 μέθοδοι: δέο μέθοδοι διχοτόμησης ***VCA*** & ***VAG*** και μία μέθοδος συνεχών κλασμάτων ***VAS***.

Ασκηση 4. Να προγραμματίσετε και τις τρεις αυτές μεθόδους και να τις συγκρίνετε ως προς την ταχύτητα απομόνωσης των ριζών.Να υπολογίσετε (απομόνωση + προσέγγιση) τις ρίζες των 2 πολυωνύμων καθώς επίσης και την πολλαπλότητα τους:

$$64x^7 - 112x^5 + 56x^3 - 7x$$

και

$$128x^8 - 256x^6 + 160x^4 - 32x^2 + 1.$$

Λύση 4.

```
>>> import timeit
    from sympy import symbols
    from sympy.polys import *
    from sympy.polys.dispersion import *
    from sympy.matrices.matrices import  *
    from sympy.solvers import *
    from sympy import *


    x = symbols('x')
    lim_list = []
    root_list = []
    lim_list2 = []
    lim_list3 = []
```

```
>>> #euresi ano oriou
    def upperbound(f):

        fspace = [[],[]]
        fspace=intervals(f)
        d = degree(f,x)

        for i in range (d):
            for j in range (1):
                oria = str(fspace[i][j])
                oria = oria.replace("(", "")
                oria = oria.replace(")", "")
                oria = oria.replace(" ", "")
                oria = oria.split(",")

        oria = str(oria)
        oria = oria.replace(","," ")
        oria = oria.replace("'","")
        oria = oria.replace("[","")
        oria = oria.replace("]"," ")
        oria = oria.split(" ")
        ub1 = int(oria[2])
        return (ub1)
```

```
>>> #ypologismos allagon prosimoy
    def sighChanges(InList):

        length = len(InList)
        prev = 0
        changes = 0

        for i in range (length):
            if InList[i] < 0:
                signof = -1
            else:
                signof = +1

            if signof == -prev:
                changes = changes + 1
                prev = signof
            else:
                prev = signof

        return(changes)
```

```
>>> def VAG(f,a,b):

        d = degree(f,x)
        vari = 0
        coeffs_list = []

        fvar = ((x + 1)**d * f.subs({x : (a + b*x)/(x +
    1)})).simplify()
        coeffs_list = Poly(fvar,x).all_coeffs()
        vari = sighChanges(coeffs_list)

        if vari == 0 :
            return
        elif vari == 1 :
            lim_list2.append([a,b])
            return

        m = expand((a + b)/ 2).simplify()

        fm = expand((f.subs({x : m})).simplify())
        if fm != 0 :
            newb = m
            VAG(f ,a,newb)
            newa = m
            VAG(f ,newa,b)
```

```
>>> # def VAS(fz,fM):
    #
    #     d = degree(f,x)
    #     vari = 0
    #     coeffs_list = []
    #
    #     coeffs_list = Poly(fz,x).all_coeffs()
    #     vari = sighChanges(coeffs_list)
    #     if vari == 0 :
    #         return
    #     elif vari == 1 :
    #         a = expand((fM.subs({x : 0})).simplify())
    #         b = upperbound(fM)
    #         lim_list3.append([a,b])
    #         return
    #     lb = lowerbound(fz)
    #     print(lb)
    #     if lb >=1 :
    #         fz = expand((fz.subs({x : x +
    1})).simplify())
    #         fM = expand((fM.subs({x : x +
    lb})).simplify())
    #
    #     p01 = expand(((x+1)**d * fz.subs({x : 1 /(x +
    1)})).simplify())
    #     M01  = expand((fM.subs({x : 1 /(x +
    1)})).simplify())
    #     m = expand((fM.subs({x : 1})).simplify())
    #     p1ap = expand((fz.subs({x : (x+1)
    })).simplify())
    #     M1ap = expand((fM.subs({x : (x+1)})).simplify())
    #     p1 = expand((fz.subs({x : 1})).simplify())
    #     if p1ap != 0 :
    #         VAS(p01,M01)
    #         VAS(p1ap ,M1ap)
    #
    #
```

```
>>> def VCA(f,a,b):

        d = degree(f,x)
        vari = 0
        coeffs_list = []

        fvar = ((x + 1)**d * f.subs({x : 1 / (x +
    1)})).simplify()

        coeffs_list = Poly(fvar,x).all_coeffs()
        vari = sighChanges(coeffs_list)
        m = expand((a + b)/ 2).simplify()

        if vari == 0 :
            return
        elif vari == 1 :
            lim_list.append([a,b])
            return

        f0_12 = expand((2**d * f.subs({x : x /
    2})).simplify())

        f12_1 = expand((2**d * f.subs({x : (x + 1) /
    2})).simplify())

        f1_2 = expand((f12_1.subs({x :
    (0.5)})).simplify())

        if f1_2 != 0 :

            newb = m
            VCA(f0_12,a,newb)
            newa = m
            VCA(f12_1 ,newa,b)

>>> def findroots(foz,listoo):

        length = len(listoo)
        i = 0
        middle = 0

        while i < length :
            interval = listoo[i]
            interval = str(interval)
            interval = interval.replace("[", "")
```

```
>>> def All_methods(fz):
        ub = upperbound(fz)
        lb = 0
        fvca = (fz.subs({x : ub*x})).simplify()

        start = timeit.default_timer()
        VCA(fvca,lb,ub)
        stop = timeit.default_timer()

        start2 = timeit.default_timer()
        VAG(fz,lb,ub)
        stop2 = timeit.default_timer()

    #     start3 = timeit.default_timer()
    #     M = x
    #     VAS(fz,M)
    #     stop3 = timeit.default_timer()

        start1 = timeit.default_timer()
        findroots(fz,lim_list)
        stop1 = timeit.default_timer()

        print("Method␣VCA:")
        print("Time␣of␣VCA␣to␣find␣intervals␣(seconds):␣",
    stop-start)
        print("\nMethod␣VAG:")
        print("Time␣of␣VAG␣to␣find␣intervals␣(seconds):",
    stop2-start2)
    #     print("\nMethod␣VAS:")
    #
    print("Time␣of␣VAS␣to␣find␣intervals␣(seconds):␣",
    stop3-start3)
    #

        print("\nThe␣roots␣with␣10^-6␣tolerance␣is:",root_list)
        print('Calculating␣time␣to␣find␣positive␣roots␣(seconds):␣',
    stop1-start1)

>>> f1 = 64*x**7-112*x**5+56*x**3-7*x
    All_methods(f1)
   Method VCA:
```

```
Time of VCA to find intervals (seconds):
2.197025090000011

Method VAG:
Time of VAG to find intervals (seconds):
2.583577961000003

The roots with 10^-6 tolerance is: [0.1950904130935669,
0.5555702447891235, 0.8314696550369263,
0.9807852506637573, 0.43388378620147705,
0.7499998807907104, 0.7818313837051392,
0.9749280214309692, 0.43388378620147705,
0.7818313837051392, 0.9749280214309692]
Calculating time to find positive roots (seconds):
0.43479175300001316
>>>
```

```
>>> f2 = 128*x**8-256*x**6+160*x**4-32*x**2+1
    All_methods(f2)
```

```
Method VCA:
Time of VCA to find intervals (seconds):
3.3074066919999723

Method VAG:
Time of VAG to find intervals (seconds):
2.9584715379999693

The roots with 10^-6 tolerance is: [0.1950904130935669,
0.5555702447891235, 0.8314696550369263,
0.9807852506637573]
Calculating time to find positive roots (seconds):
0.5497769030000086
```

Σημείωση 2. Τη μέθοδο VAS την έχουμε μέσα σε σχόλια καθώς δε προλαβαμε να προγραμματίσουμε κάποια μέθοδο υπολογισμού κάτω ορίου. Παρ'όλα αυτά θεωρούμε οτι σαν αλγόριθμος είναι σωστός και έπρεπε εστώ και με μορφή σχολίου να συμπεριληφθεί