Συναρτήσεις - Αναδρομή

Εκτεταμμένα παραδείγματα

Παράδειγμα. (Εύρεση ρίζας με τη μέθοδο διχοτόμησης.) Έχουμε δει πώς μπορούμε να εντοπίσουμε τη ρίζα μίας εξίσωσης *f(x)=0* μεταξύ δύο αριθμών x_1 και x_2. Ας επαναλάβουμε το πρόγραμμα που είχαμε δει χρησιμποιώντας συνάρτηση για την *f(x)*.

```
import math

def fun(x):
        result = math.exp(x) - 2.0
        return result


# Main program
x1 = float(input("Give x1: "))
fx1 = fun(x1)

x2 = float(input("Give x2: "))
fx2 = fun(x2)

print('f(',x1,') =',fx1)
print('f(',x2,') =',fx2)

if fx1*fx2 <= 0:
        print("A root is between ",x1,x2)
        x = (x1+x2)/2.0
        fx = fun(x)
        print("You can find a better estimate")
        print('f(',x,') =',fx)
else:
        print("I cannot locate a root")
```

Θα επεκτείνουμε εδώ αυτή τη μέθοδο για να βρούμε μία ρίζα της εξίσωσης. Θα κάνουμε μία επανάληψη ως εξής:

- Υπολογίζουμε την τιμή της *f(x)* στον μέσον του διαστήματος $[x_1, x_2]$.
- Αν τα *f(x)* και $f(x_1)$ είναι ετερόσημα τότε η ρίζα βρίσκετα στο δίαστημα $[x, x_1]$ (ή $[x_1, x]$). *Αλλιώς, βρίσκεται μεταξύ x και* x_2.
- Ονομάζουμε το νέο διάστημα στο οποίο εντοπίζεται η ρίζα $[x_1, x_2]$ και επαναλαμβάνουμε την διαδικασία.
- Η επανάληψη τερματίζεται όταν το διάστημα εντοπισμού την ρίζας γίνει πολύ μικρό. Λέμε ότι έχουμε μία προσέγγιση της ρίζας, την οποία μπορούμε να κάνουμε όσο καλή θέλουμε.

Η παραπάνω διαδικασία περιλαμβάνεται στην συνάρτηση `findRoot`.

```
def findRoot(x1,x2):
   fx1 = fun(x1); fx2 = fun(x2)
   if fx1*fx2 > 0:
      return None

   while abs(x2-x1)>1e-6:
      x = (x1+x2)/2.0
      fx = fun(x)
      if fx1*fx < 0.0:
         x2 = x; fx2 = fx
      else:
         x1 = x; fx1 = fx
      return x
```

Ένας πλήρες πρόγραμμα το οποίο καλεί την `findRoot` είναι το εξής:

```
# Main program
x1 = float(input("Give x1: "))
fx1 = fun(x1)

x2 = float(input("Give x2: "))
fx2 = fun(x2)

print('f(',x1,') =',fx1)
```

```
print('f(',x2,') =',fx2)
```

Παρατηρήστε ότι το κύριο πρόγραμμα καλεί την συνάρτηση `findRoot`, η οποία καλεί την συνάρτηση `fun`. Για να γράψετε το πλήρες πρόγραμμα πρέπει λοιπόν να γράψετε σε ένα αρχείο πρώτα την `fun` μετά την `findRoot` και τέλος το κύριο πρόγραμμα.

[Δείτε και τις σημειώσεις Μ. Κολουντζάκη για την Εύρεση ρίζας με τη μέθοδο διχοτόμησης.]

Παράδειγμα. Ελάχιστη απόσταση μεταξύ ζευγών σημείων.

Παράδειγμα. Μέγιστος κοινός διαιρέτης.

Αναδρομή

Παράδειγμα (Παραγοντικό). Το παραγοντικό ακεραίου n ορίζεται ως $n! = n \cdot (n-1) \cdot \ldots \cdot 1$ Θα το υπολογίσουμε κάνοντας διαδοχικούς πολλαπλασιασμούς.

```
def factorialIter(n):
    """Assumes n is int > 0
    returns n!"""
    nfact = 1
    while n > 1:
        nfact = n*nfact
        n -= 1
    return nfact
```

Είναι όμως συχνά πιο απλό να σκεφτούμε ότι το παραγοντικό $(n+1)!$ μπορεί να υπολογιστεί άμεσα αν είναι γνωστό το $n!$, αφού $(n+1)! = (n+1) \cdot n!$ Αυτό ορίζει μία αναδρομική σχέση (το $n!$ υπολογίζεται αν γνωρίζουμε το $(n-1)!$, κλπ) και μας επιτρέπει να αναπτύξουμε μία επαγωγική μέθοδο. Η διαδικασία τερματίζεται αν γνωρίζουμε ότι $1! = 1$ (βασική περίπτωση).

Για να γράψουμε έναν κώδικα ο οποίος θα εφαρμόζει την αναδρομική μέθοδο μπορούμε να γράψουμε μία συνάρτηση η οποία θα εφαρμόζει την αναδρομική σχέση για κάθε ακέραιο n.

```
def factorialRecur(n):
```

```
    if n == 1:
        return n
    else:
        return = n*factorialRecur(n-1)
```

Παρατήρηση. Η συνάρτηση `factorialRecur` χρησιμοποιεί την ίδια τη συνάρτηση (δηλαδή, αντίγραφο του εαυτού της) για να μπορέσει να εφαρμόσει την αναδρομική σχέση. Επίσης, υπάρχει μία βασική περίπτωση (`n==1`) για την οποία οι διαδοχικές κλήσεις της συνάρτησης τερματίζονται, αλλιώς θα είχαμε διαδοχικές κλήσεις της συνάρτησης άπειρες φορές.

Παράδειγμα (Ακολουθία Fibonacci). Αν ο πληθυσμός (κάποιου βιολογικού είδους) παρασταθεί με F_i σε κάθε χρονιά i, τότε έχει υποστηριχθεί ότι η αύξηση του πληθυσμού κάποιων ζώων μπορεί να παρασταθεί από την ακολουθία $F_i = F_{i-1} + F_{i-2}$, δηλαδή εξαρτάται από τον πληθυσμό τα δύο προηγούμενα χρόνια. Αυτή είναι μία αναδρομική σχέση. Για την εφαρμογή ενός υπολογισμού χρειαζόμαστε μία βασική περίπτωση: θα υποθέσουμε ότι αρχικά είχαμε μόνο ένα υποκείμενο του βιολογικού είδους, δηλαδή $F_0 = 1, F_1 = 1$.

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return = fib(n-1) + fib(n-2)
```

Παράδειγμα. Μία λίστα λέγεται παλινδρομική αν η σειρά των στοιχείων της είναι η ίδια είτε διαβάζεται από αριστερά προς τα δεξιά είτε αντίστροφα. Ας ελέγξουμε αν μία λίστα είναι παλινδρομική χρησιμοποιώντας αναδρομική κλήση συνάρτησης.

```
def isPalindrome(L):
    if len(L) <= 1:
        return True
    return L[0] == L[-1] and isPalindrome(L[1:-1])
```

Παράδειγμα. [Πηγή: Σημειώσεις Μ. Πλεξουσάκη.] Έστω *L* μια λίστα τα στοιχεία της οποίας μπορεί να είναι με τη σειρά τους λίστες. Για παράδειγμα, θα μπορούσαμε να

έχουμε `L = [1, 2, [2, 3], [4], 6]`. Γράψτε μια συνάρτηση η οποία κατασκευάζει μια «μονοδιάστατη» λίστα, αφαιρεί δηλαδή τυχόν εσωτερικές λίστες.

```
def flatten(L):
    M = []
    for item in L:
        if isinstance(item, list):
            for internalItem in item:
                M.append(internalItem)
        else:
            M.append(item)
    return M
```

Η εντολή `isinstance(item,list)` ελέγχει αν η μεταβλητή `item` είναι τύπου `list`.

Χρησιμοποιώντας αναδρομή μπορούμε να γράψουμε μία πιο κομψή μορφή του προγράμματος. Θα χρειαστούμε την μέθοδο `M.extend(item)` η οποία προσθέτει τα στοιχεία μίας λίστας `item` μετά το τελευταίο στοιχείο της λίστας `L`.

```
def flatten(L):
    M = []
    for item in L:
        if isinstance(item, list):
            M.extend(flatten(item))
        else:
            M.append(item)
    return M
```

Η μορφή αυτή του προγράμματος περιλαμβάνει και την περίπτωση για την οποία μία λίστα περιέχει λίστες οι οποίες επίσης περιέχουν λίστες, π.χ., μπορεί να κάνει μονοδιάστατη την λίστα `L = [1, 2, [2, [3,5]], [4], 6]`.

Παράδειγμα. Δυαδικό ανάπτυγμα φυσικού αριθμού.

Παράδειγμα. Ο αλγόριθμος του Ευκλείδη για το ΜΚΔ.

Παράδειγμα. Υπολογισμός διωνυμικών συντελεστών.