# מציאת שורשים

### הגדרת הבעיה

שיטות למציאת שורש (ערך שמאפס את הפונקציה( עבור פונקציה מכל סוג ,כאשר השורש המדויק מסומן באחד מהסימונים הבאים .

**שיטת החציה:** .

בהנתן ששורש מסוים של פונקציה נתון בקטע  שבו פונקציה מחליפה סימן אזי על פי משפט ערך הביניים קיים ערך שבו הפונקציה מתאפסת , לכן נמצא את השורש המקורב על פי האלגוריתם הבא:

1. חצה את הקטע לשתיים כך ש
2. אם  אזי השורש המקורב הוא c

אחרת בדוק

אם  אזי a=c

אחרת b=c.

3. חזור ל 1

בצורה זו הקטע מצטמצם עד לבחירת הקטע המינימאלי במסגרת ההתניה על השורש או מגבלות המחשב.

יתרון: תמיד מתכנס ,קצב קבוע (אחרי *n* איטרציות גודל הקטע קטן פי ).

חסרונות: התכנסות איטית מדי, צריכים לדעת מראש קטע שבו פונקציה מחליפה סימן.

שימוש אופייני: שלב ראשון של פתרון. אם שיטה מהירה יותר לא מתכנסת מהניחוש הראשוני הקיים, מתקרבים לשורש בשיטת חציה ואז חוזרים לשיטה מהירה יותר.

****

**import** sys  
  
**def** f(x):  
 **return** x\*\*2 - 4  
  
**def** bisection(a, b, epsilon):  
 m = (a + b) / 2.0  
 **while** abs(a - b) > epsilon:  
 **if** f(m) == 0:  
 **return** m  
 **elif** f(a) \* f(m) < 0:  
 b = m  
 **else**:  
 a = m  
 m = (a + b) / 2.0  
 **return** m  
  
  
print (**'The root is: '**,)  
print (bisection(0,3,0.0001))

**שיטת Newton – Raphson:** פותרים משוואה . בנקודה  מקרבים את הפונקציה ע"י המשיק ומוצאים את הנקודה הבאה כחיתוך בין משיק לציר *x*: . השיטה מתכנסת עם סדר 2 ,השיטה איננה מתכנסת אם הערך של הנגזרת בנקודה שווה לאפס .

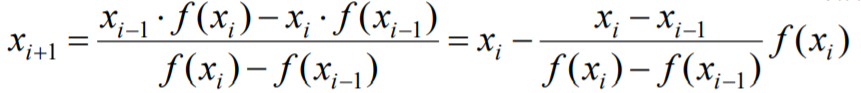
יתרונות וחסרונות: השיטה הכי יעילה באזורים קרובים לשורש. באזורים רחוקים היא עלולה להתבדר. אסור שהנגזרת תהיה שווה לאפס.

.****

**from** math **import** sin, cos  
  
  
**def** newton\_rapson(func, x=0, it=1, h=0.00000001):  
 *"""func is a function  
 x is the starting guess  
 it is number of iterations  
 h is the accuracy factor of the derivative"""* **def** der(f, h):  
 **return lambda** x0: (f(x0 + h) - f(x0)) / h  
  
 **if** der(func, h)(x) == 0:  
 x = 1  
 **for** i **in** range(it):  
 fd = func(x)  
 ft = der(func, h)(x)  
 x = x - (fd / ft)  
 **return** x  
  
  
**def** f1(x):  
 **return** sin(x) - cos(x)  
  
  
print(f1(newton\_rapson(f1, 0, 100))) *# should be ~0*

**שיטת המיתר (Secant):** זוהי שיטה איטרטיבית למצוא שורשים של פונקציה רציפה בעלת משתנה אחד. נתונה פונקציה ונרצה לחשב את השורשים של המשוואה מהצורה .

בשיטה זו מחברים את שתי הנקודות האחרונות ע"י ישר ומוצאים חיתוך בין ישר זה לציר *x*.  
בהינתן שתי נקודות xi וxi-1 מוצאים את xi+1 , כך מוצאים את הישר העובר ב- xi וxi-1 ולוקחים את xi+1 להיות חיתוכו עם ציר ה-X , הנוסחה:



חסרונות: השיטה לא תמיד מתכנסת, אבל אם היא מתכנסת אז מהר יותר: סדר התכנסות בערך 1.6.

יתרונות: לעומת Newton – Raphson לא צריכים לחשב נגזרת (זה יכול להיות קשה). כתוצאה מזה בכל איטרציה מחשבים רק את הפונקציה

****

xI ו xI\_1 הינם ניחושים התחלתיים.   
נסמן את eS גודל קטן כלשהו המהווה רמת מובהקות מהו הפער בין שני הקבוצות.  
amtIterations כמות האיטרציות ככל שנרבה באיטרציות כך נקבל ערך קרוב יותר לשורש.

**def** f(x):  
 **return** x\*\*2 - 2  
  
go = **True**count = 0  
  
print(**"Enter your two guesses:"**)  
xI = float(input(**"Guess 1:"**))  
xI\_1 = float(input(**"Guess 2: "**))  
amtIterations = float(input(**"Enter the amount:"**))  
eS = float(input(**"Enter your pre-specified relative error of tolerance:"**))  
  
**while** go:  
 fXI = f(xI)  
 fXI\_1 = f(xI\_1)  
  
 t = xI  
  
 xI = xI - ((fXI \* (xI - xI\_1))/(fXI - fXI\_1))  
  
 xI\_1 = t  
  
 eA = float(abs((xI - xI\_1)/xI) \* 100)  
  
 count += 1  
  
 print(**"Estimated root (gas oil on dipstick): "**, xI)  
 print(**"Approximate error:"**, eA)  
 print()  
  
 **if**(count < amtIterations) **or** (eA > eS):  
 go = **True  
 else**:  
 go = **False** print(**"Terminate"**)