PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **17** (1989/1990)
Številka 3
Strani 182–189

Matija Lokar:

IŠČEMO NIČLO

Ključne besede: matematika, numerična analiza, numerične metode, reševanje enačb, regula falsi, bisekcija.

Elektronska verzija: http://www.presek.si/17/982-Lokar-0.pdf

IŠČEMO NIČLO

Eno od zelo pogostih opravil v matematiki je iskanje ničel določene funkcije. Iščemo točko x, za katero velja

$$f(x) = 0$$

Točki x, ki zadošča gornjemu pogoju, pogosto rečemo tudi *koren enačbe.* Pred takim opravilom ste se znašli (ali pa se še boste), ko ste iskali rešitve kvadratne enačbe

$$ax^2 + bx + c = 0$$

No, že od tam veste, da lahko take točke sploh ni, ali pa jih je lahko več. Na primer: enačba

$$x^2 - 5x + 6 = 0$$

ima dve rešitvi in sicer $x = 3$ in $x = 2$, enačbi

$$x^2 + 1 = 0$$

ne zadošča nobeno realno število, enačbi

$$x^2 - 4x + 4 = 0$$

pa ustreza le $x = 2$. Vemo tudi, da kvadratno enačbo brez težav uženemo. Izračunamo

$$D = b^2 - 4ac$$

in če je D pozitiven ($D > 0$), sta dve rešitvi, če je D negativen, rešitve v realnem ni, če pa je D enak nič, je natanko ena rešitev. Z iskanjem rešitve ni težav, saj dobro poznamo obrazec

$$x_1 = \frac{-b + \sqrt{D}}{2a}$$

$$x_2 = \frac{-b - \sqrt{D}}{2a}$$

Zaplete se le, če enačba ni prava kvadratna enačba, ampak je $a = 0$. Enačba je potem še enostavnejša, linearna

$$bx + c = 0$$

Ta ima le eno rešitev in sicer $x = -c/b$, če le b ni enak nič. Če pa je, potem imamo dve možnosti. Kadar je tudi c enak nič nam enačbo reši

vsak x, saj bomo na levi vedno dobili 0. Če pa c ni nič, pač rešitve ni.

Rešitve kvadratne enačbe $ax^2 + bx + c = 0$					
$a = 0$			$a \neq 0$		
$b = 0$		$b \neq 0$	$D = b^2 - 4ac$		
$c = 0$	$c \neq 0$		$D < 0$	$D = 0$	$D > 0$
vsa realna os	ni rešitve	ena rešitev	ni rešitve v realnem	ena rešitev	dve rešitvi

Napišimo program, ki nam bo rešil poljubno kvadratno enačbo. Zapišimo ga kar v BASICU.

```
 10 REM
 20 REM RESEVANJE KVADRATNE ENACBE
 30 REM
 40 INPUT 'Vnesi koeficiente A, B in C',A,B,C
 50 IF A = 0 THEN GOTO 110 : REM linearna enacba
 60 D = B*B - 4*A*C
 70 IF D < 0 THEN PRINT 'Ni realne resitve' : STOP
 80 IF D = 0 THEN PRINT 'Resitev je ';-B/(2*A) : STOP
 90 X1 = (-B + SQR(D))/(2*A) : X2 = (-B - SQR(D))/(2*A)
100 PRINT 'Resitvi sta ';X1;' in ';X2 : STOP
110 REM linearna enacba
120 IF B <> 0 THEN PRINT 'Resitev je ';-C/B : STOP
130 IF C = 0 THEN PRINT 'Resitev so vsi x' : STOP
140 PRINT 'Ni resitve'
```

Brž preiskusimo program. Deluje prav! No pa vtipkajmo za koeficiente še 1, -5000.002, 10. Brž se lahko prepričamo, da sta rešitvi 5000 in 0.002. Kaj pa naš program? Glej ga šmenta! Rešitvi, ki ju je izračunal sta 5000 in 0.001953. Napaka res ni velika, ampak le zakaj je prišlo do nje? In zakaj je ena rešitev točna, druga pa ne? Težava je v tem, ker računalnik ne more računati povsem natančno, ampak vedno na določeno število mest. Ponavadi napaka ni velika, ampak napake imajo ponavadi mlade. In tako lahko le majhna napaka na začetku povzroči, da je končni rezultat povsem napačen. In kje se je vtihotapila napaka? Poglejmo. $\sqrt{(-5000.002)^2 - 4 \cdot 10 \cdot 1} \approx -5000.002$ In ko odštevamo dve približno enaki števili, pride končno število mest, ki jih upošteva računalnik, še posebej do veljave. Mogoče bo vaš računalnik gornji problem izračunal

pravilno. Vendar ne bo potrebno veliko poskušanja, da boste našli primer, ko rešitvi ne bosta povsem pravi.

Na srečo se pri reševanju kvadratne enačbe tovrstnim napakam lahko izognemo. Kot smo videli, je bila ena rešitev točna. Je to vedno X1? No, če je b negativen, da. Če pa je b pozitivno število, bo do odštevanja približno enakih števil lahko prišlo prav pri računanju X1 in bo X2 pravilen. Kako pa točno izračunati drugi koren. Pomagamo si z Vietovimi obrazci. Eden med njimi pravi, da je produkt rešitev kvadratne enačbe enak c/a

$$x_1 \cdot x_2 = c/a$$

V programu vrstico 90 nadomestimo z

```
85 IF B > 0 THEN X1 = (-B - SQR(D)) / (2*A) : GOTO 95
90 X1 = (-B + SQR(D)) / (2*A)
95 X2 = C/(A*X1)
```

Vse lepo in prav. Kaj pa če bi radi našli rešitev za poljubno funkcijo in ne le kvadratno? Npr.

$$x^4 - \frac{2}{\sqrt{x^2-1}} + 4 = 0$$

Ali pa še kakšno zapletenejšo funkcijo? Saj ni treba, da bi znali poiskati vse rešitve. Ampak vsaj kakšno bi pa že radi našli.

Najenostavnejša metoda je verjetno poizkušanje. Izberemo si točko, izračunamo vrednost funkcije in postopek ponavljamo toliko časa, dokler ne naletimo na točko, kjer je funkcijska vrednost res enaka nič. Vendar s tako metodo ne bomo zadovoljni. Preveč je odvisna od naše sreče pri izbiri točk. In kaj če slučajno take točke sploh ni? Kdaj se ustaviti?

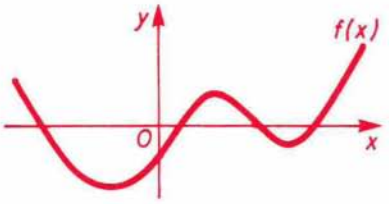
No, če je funkcija dovolj 'lepa', obstaja postopek, ki nas vedno pripelje do rešitve. Postopku rečemo **bisekcija** ali **metoda razpolavljanja**. S pomočjo nje poiščemo rešitev enačbe

$$f(x) = 0$$

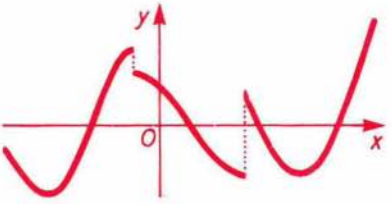
za poljubno zvezno funkcijo f. Kaj pa sploh je to zvezna funkcija? Enostavno. To je funkcija, katere slika ni nikjer strgana, nikjer prekinjena. Npr. na sliki 1 je zvezna funkcija, funkcija na sliki 2 pa ni zvezna.

Iščemo točke, kjer slika funkcije **zadene** koordinatno os x.

Pa še nekaj potrebujemo za začetek postopka. Levo in desno mejo intervala, kjer iščemo ničlo funkcije. Meji moramo postaviti tako, da je



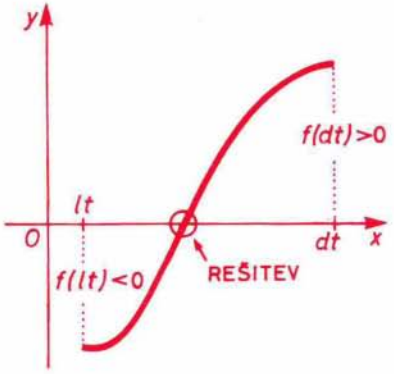
Slika 1: Zvezna funkcija



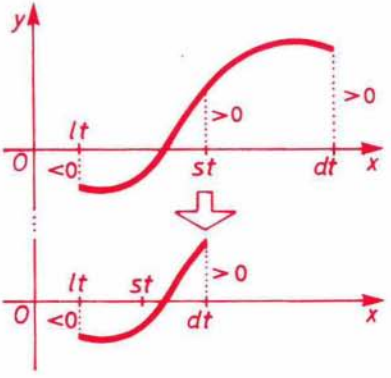
Slika 2: Nezvezna funkcija

funkcija v mejnih točkah nasprotnega predznaka. Torej, če je na levem krajišču pozitivna, mora biti na desnem negativna ali obratno. Na ta način zagotovimo, da x, ki reši gornjo enačbo res obstaja. Kratek razmislek nas prepriča o tem. Če smo na negativni strani koordinatne osi x in moramo priti na pozitivno stran, pri tem pa se moramo gibati po neprekinjeni poti, je očitno, da bomo morali vsaj enkrat prečkati koordinatno os. Tista točka pa je že iskana rešitev (slika 3)

Kako pa bomo prišli do točke, kjer prečkamo os ? Vzemimo srednjo točko in poglejmo vrednost funkcije le tam. Če je enaka nič, smo jo že našli. Drugače ima pozitivno ali negativno vrednost. Sedaj glede na predznak spremenimo bodisi levo, bodisi desno mejo, tako da je ima funkcija na krajiščih še vedno nasprotni predznak. Postopek ponavljamo, dokler ne najdemo rešitve. Slika 4 bo verjetno dovolj zgovorna



Slika 3



Slika 4: Bisekcija

Zapišimo še program, tokrat v pascalu

```
program test_bisek;

var   a,b : real;   { interval }
      nat : real;   { natancnost}
      n : integer;  { iteracij }
      ok : integer;
      nicla : real;
{ ------------------------------------------------ }
function f(x:real) : real;
{ funkcija, katere niclo iscemo }
begin
  f := x*x - sqrt(x) - 1
end;
{ ------------------------------------------------ }

procedure  bisek (lt,dt,nat : real; var n : integer;
                  var nic : real;
                  var ok  : integer );

  { z metodo bisekcije poisce niclo funkcije f(x).
    lt : leva tocka intervala
    dt : desna tocka intervala
    nat : natancnost
    n   : maksimalno stevilo iteracij
          v rezultatu dejansko stevilo

    nic : izracunana nicla, ce ok = 0
    ok = 1 : f(lt) * f(dt) >= 0
       = 2 : n premajhen                              }

var
  st  : real;          {srednja tocka}
  it  : integer;       { stevec iteracij }
  nad : Boolean;


begin
   if f(lt) * f(dt) > 0 then  ok := 1 {napacno izbran interval}
   else begin
     { pravilno predznacena funkcija }
     it := 1; nad := true;
     while (it <= n) and nad do begin
       st := lt + (dt - lt)/2;      { izbira naslednje tocke }
```

```
      if (f(st) = 0) or ((dt - lt)/2 < nat) then begin
        { nasli smo niclo }
        ok := 0;
        nic := st;
        nad := false;
      end {if}
      else begin
        it := it + 1;
        if f(lt)*f(st) > 0 then lt := st  { spremenimo levo }
        else dt := st                     { ali desno mejo  }
      end {else}
    end; {while}
    if it > n then ok := 2      { prevec iteracij }
  end; {if}
  n := it;
end; { bisek }

begin
    write(' levo krajisce  ' ); readln(a);
    write(' desno          ' ); readln(b);
    write(' maks. iteracij ' ); readln(n);
    write(' natancnost     ' ); readln(nat);
    writeln;
    bisek(a,b,nat,n,nicla,ok);
    case ok of
     0 : begin writeln(' nicla je ',nicla);
               writeln('iteracij : ',n)
         end;
     1 : writeln(' napacen interval ',f(a),f(b));
     2 : writeln(' premalo iteracij ')
    end; { case }
end.
```

Če gledamo postopek s strogo matematičnega stališča, moramo s postopkom prenehati, ko pridemo do točke, kjer velja $f(x) = 0$. V programu pa prenehamo, ko je izpolnjen pogoj

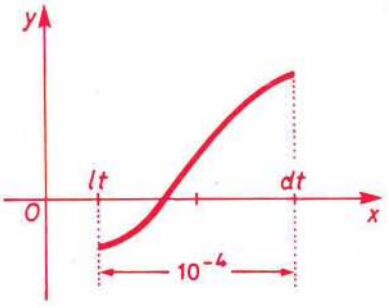
```
(f(st) = 0) or ((dt - lt)/2 < nat)
```

kjer je **nat** željena natančnost. Na sliki namreč vidimo, da se lahko zgodi, da sta leva in desna meja že povsem blizu skupaj, funkcija pa v srednji točki še vedno ni enaka nič. Vendar ne bi imelo smisla, da s postopkom

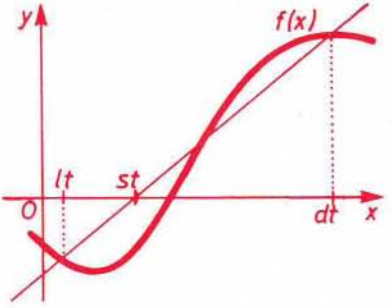
nadaljujemo, saj se izračunana rešitev od prave razlikuje za kvečjemu toliko, kot je konstanta *nat*.

Bisekcija ima to ugodno lastnost, da nas, brž ko smo si izbrali ustrezen začetni interval, vedno pripelje do rešitve (seveda, če je funkcija na izbranem intervalu res zvezna). Njena slaba lastnost je edino v tem, da do rešitve pridemo dokaj počasi, še posebej, če je začetni interval velik. Na vsakem koraku ga namreč le razpolovimo. Od tod izvira tudi ime postopka, **bi-sekcija**. Zato so se mnogi trudili, da bi našli postopek, ki bi tudi vedno pripeljal do rešitve, a hitreje. No žal takega postopka ni. Obstajajo sicer metode, ki so hitrejše, a mora imeti funkcija na izbranem intervalu še druge lastnosti. Taka postopka sta npr. Newtonova in sekantna metoda, ki si ju bomo ogledali kdaj drugič. Danes pa poglejmo še postopek, ki je podoben bisekciji in nas lahko hitreje pripelje do rešitve. Obstajajo pa žal primeri, kjer se obnaša slabše kot bisekcija. To je **metoda lažnega mesta** ali **metoda regula falsi**.

Ta metoda se od bisekcije razlikuje le v izbiri točke, kjer poskusimo najti ničlo. Pri bisekciji je to razplovišče intervala, pri metodi regula falsi pa točka, kjer premica skozi točki $(lt, f(lt))$ in $(dt, f(dt))$ seka x os.



Slika 5



Slika 6: Izbira točke pri metodi Regula falsi

V programu moramo zato le spremeniti mesto, kjer si izbiramo naslednjo točko. Namesto

```
st := lt + (dt - lt)/2;      { izbira naslednje tocke }
```

moramo uporabiti

```
st := dt - f(dt)*(dt - lt)/(f(dt) - f(lt));
```