Numeričke metode

Aleksandar Maksimović
IRB

Iterativne metode

Računanje nultočaka nelinearnih funkcija jedan je od najčešćih zadataka primijenjene matematike. Općenito, neka je zadana funkcija

$$f : I \to \mathbb{R},$$

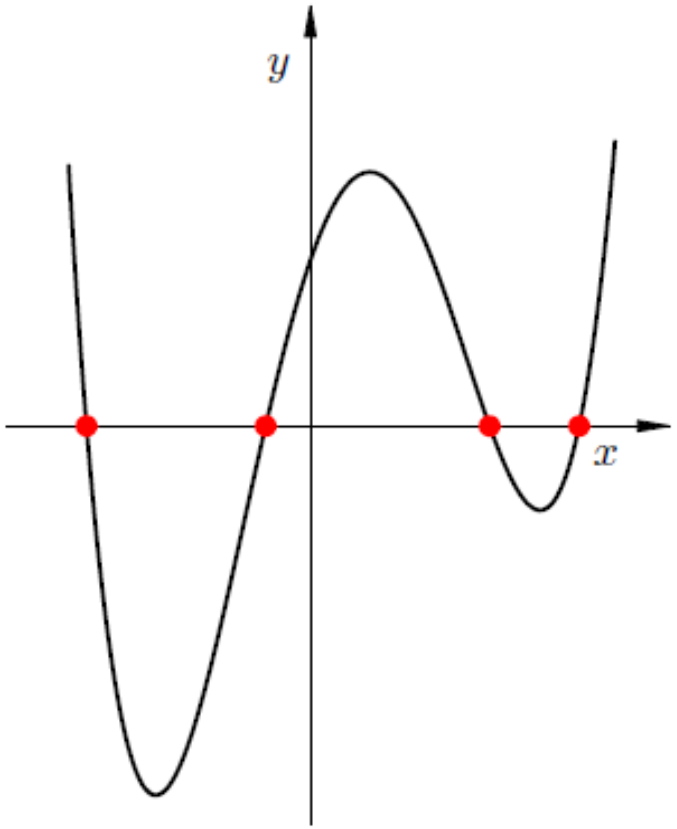
gdje je I neki interval. Tražimo sve one $x \in I$ za koje je

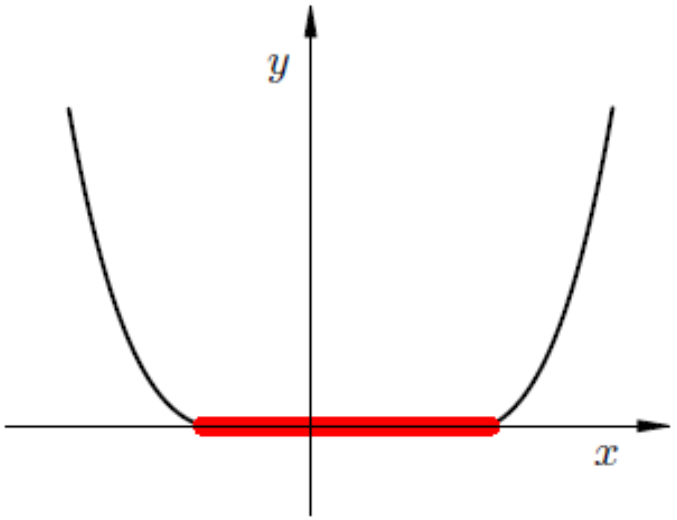
$$f(x) = 0.$$

Takve točke x zovu se rješenja, korijeni pripadne jednadžbe ili nultočke funkcije f.

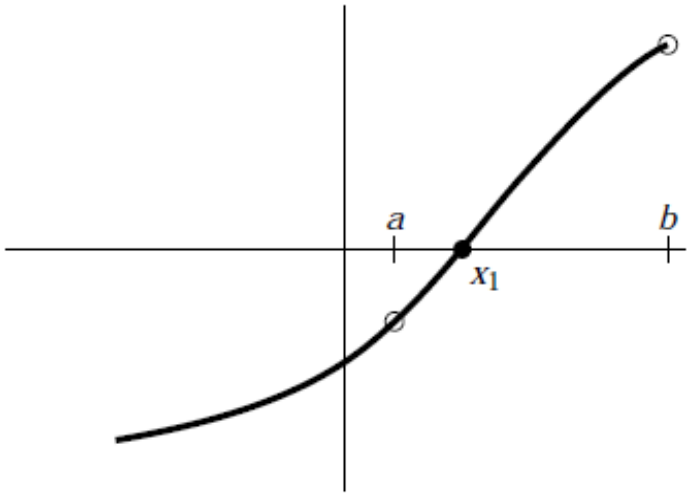
U pravilu, pretpostavljamo da je f **neprekidna** na I i da su joj nultočke izolirane. U protivnom postojao bi problem konvergencije.

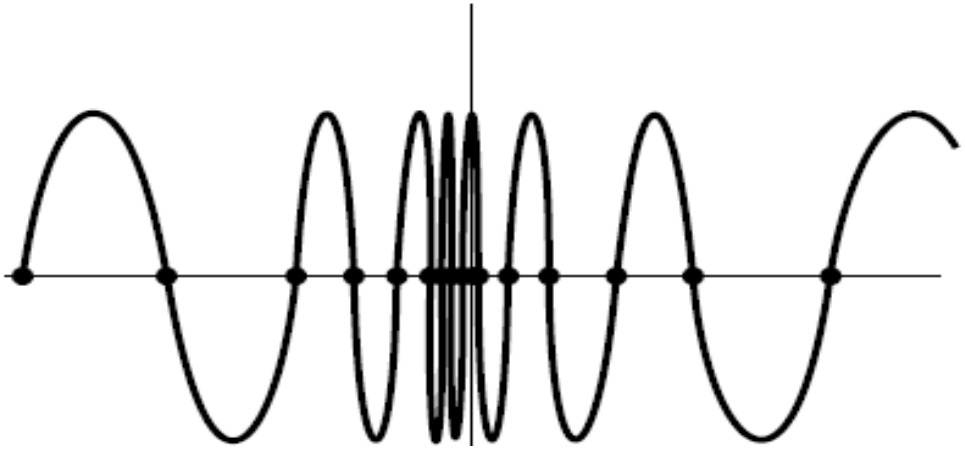
Iterativne metode

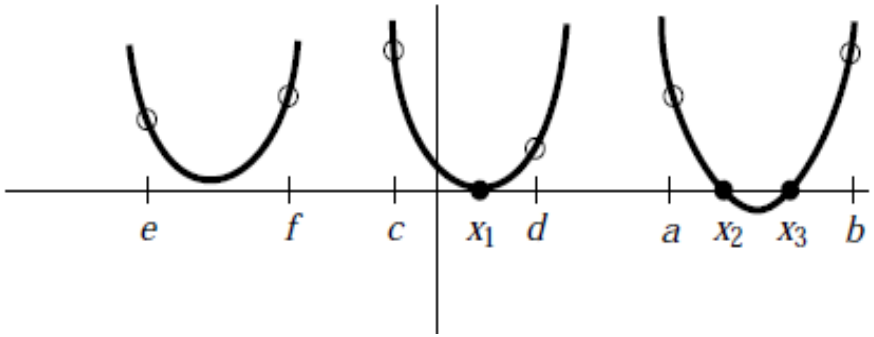


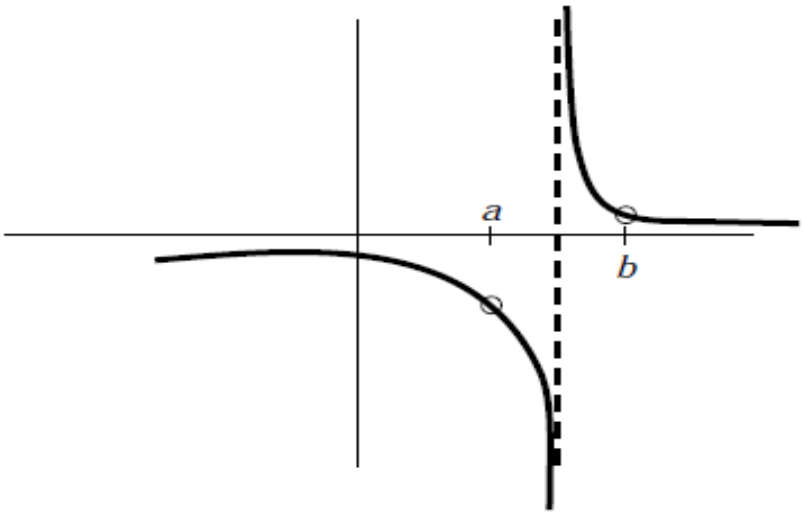


Primjeri funkcija









Iterativne metode

Traženje nultočki na zadanu točnost sastoji se od dvije faze.

1. Izolacije jedne ili više nultočki, tj. nalaženje intervala I unutar kojeg se nalazi bar jedna nultočka. Ovo je teži dio posla i obavlja se na temelju analize toka funkcije.
2. Iterativno nalaženje nultočke na traženu točnost.

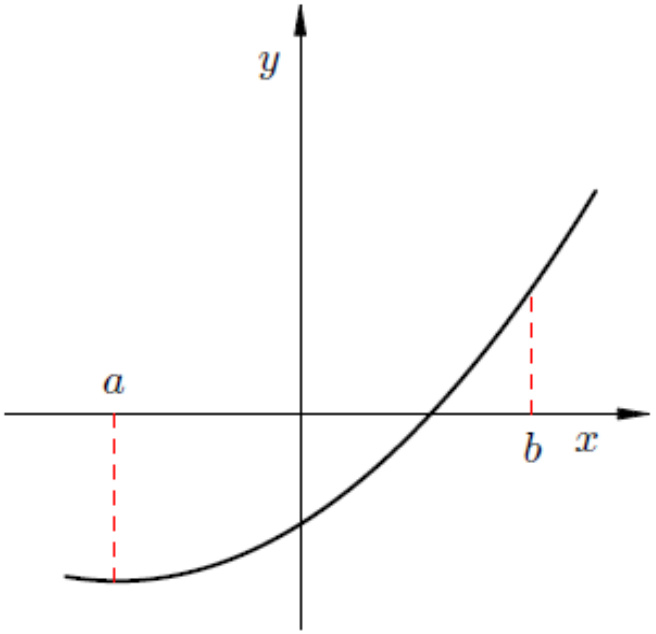
Postoji mnogo metoda za nalaženje nultočaka nelinearnih funkcija na zadanu točnost. One se bitno razlikuju po tome hoće li uvijek konvergirati, tj. imamo li sigurnu konvergenciju ili ne i po brzini konvergencije. Uobičajen je slučaj da brze metode nemaju sigurnu konvergenciju, dok je sporije metode imaju.

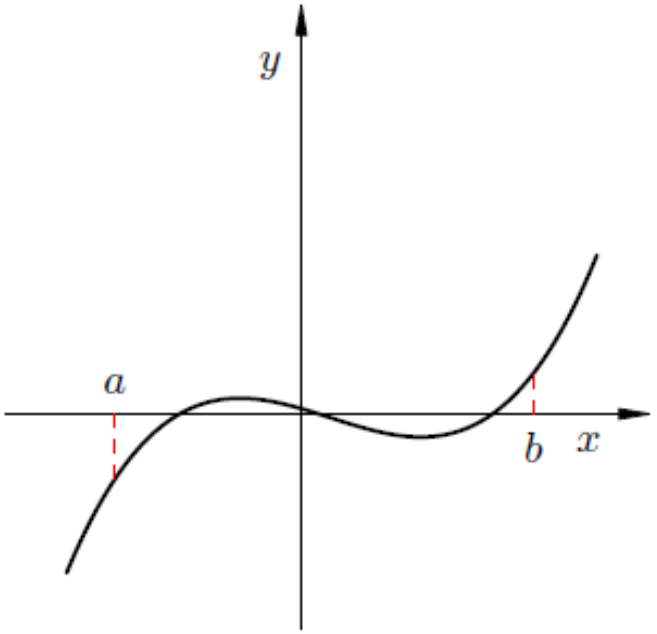
metoda bisekcije

Osnovna pretpostavka za primjenu algoritma raspolavljanja je **neprekidnost** funkcije f na intervalu $[a, b]$ i uvjet

$$f(a) \cdot f(b) < 0.$$

Prethodna relacija znači da funkcija f ima na intervalu $[a, b]$ **barem jednu** nultočku.

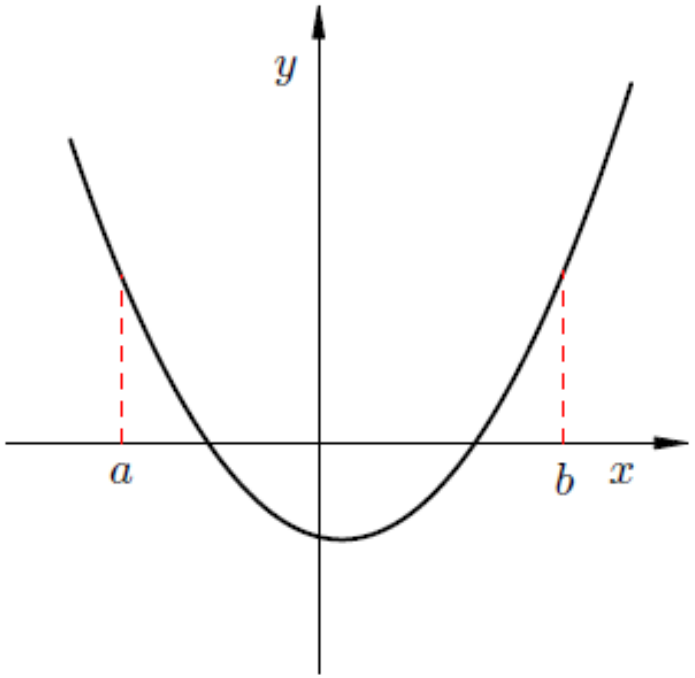


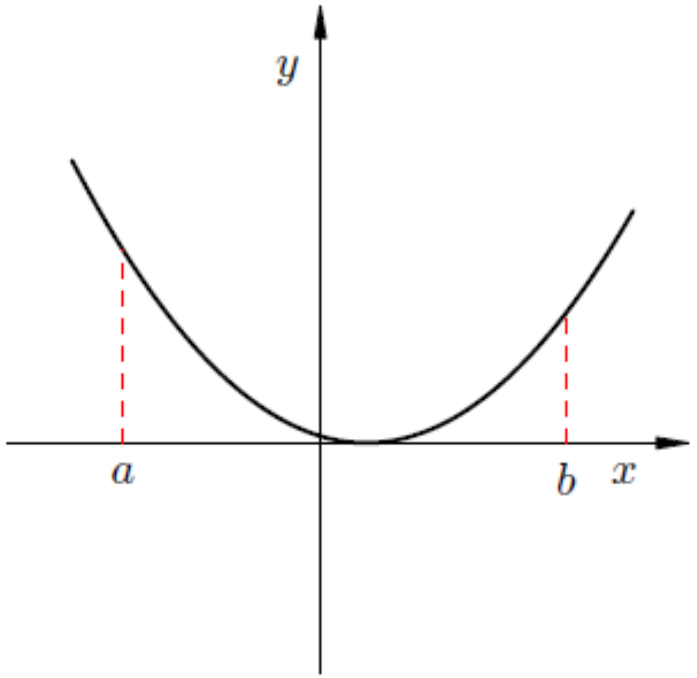


metoda bisekcije

S druge strane, ako je

$$f(a) \cdot f(b) > 0,$$





metoda bisekcije

Algoritam raspolavljanja je vrlo jednostavan. Označimo s α pravu nultočku funkcije, a zatim s $a_0 := a$, $b_0 := b$ i x_0 polovište $[a_0, b_0]$, tj.

$$x_0 = \frac{a_0 + b_0}{2}.$$

Neka je $n \geq 1$. U n-tom koraku algoritma konstruiramo interval $[a_n, b_n]$ kojemu je duljina polovina duljine prethodnog intervala, ali tako da je nultočka ostala unutar intervala $[a_n, b_n]$.

Konstrukcija intervala $[a_n, b_n]$ sastoji se u raspolavljanju intervala $[a_{n-1}, b_{n-1}]$ točkom x_{n-1} i to tako da je

$$a_n = x_{n-1},\ b_n = b_{n-1} \quad \text{ako je} \quad f(a_{n-1}) \cdot f(x_{n-1}) > 0,$$
$$a_n = a_{n-1},\ b_n = x_{n-1} \quad \text{ako je} \quad f(a_{n-1}) \cdot f(x_{n-1}) < 0.$$

Postupak zaustavljamo kad je $|\alpha - x_n| \leq \varepsilon.$

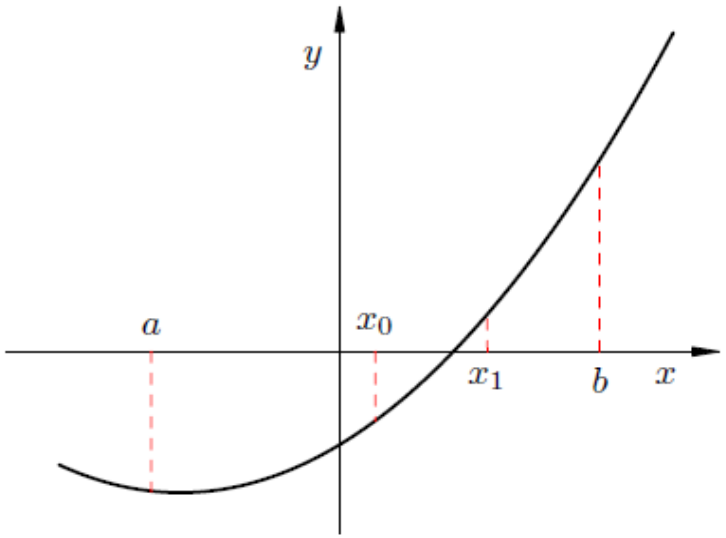
metoda bisekcije

budući da je x_n polovište intervala $[a_n, b_n]$, a $\alpha \in [a_n, b_n]$, onda je

$$|\alpha - x_n| \leq b_n - x_n,$$

pa je dovoljno postaviti zahtjev

$$b_n - x_n \leq \varepsilon.$$



metoda bisekcije

Algoritam 8.2.1 (Metoda raspolavljanja)

$x := (a + b)/2;$
while $b - x > \varepsilon$ **do**
  begin;
  if $f(x) * f(b) < 0.0$ **then**
    $a := x$
  else
    $b := x;$
  $x := (a + b)/2;$
  end;
{ Na kraju je $x \approx \alpha$. }

Aleksandar Maksimović,
Institute Ruđer Bošković, Bijenička c. 54

metoda bisekcije

Iz konstrukcije metode lako se izvodi pogreška n-te aproksimacije nultočke. Vrijedi

$$|\alpha - x_n| \leq b_n - x_n = \frac{1}{2}(b_n - a_n) = \frac{1}{2^2}(b_{n-1} - a_{n-1}) = \cdots = \frac{1}{2^{n+1}}(b-a).$$

$$\frac{1}{2^{n+1}}(b-a) \leq \varepsilon. \qquad \text{a zatim logaritmiranjem nejednakosti}$$

$$n \geq \frac{\log(b-a) - \log\varepsilon}{\log 2} - 1, \quad n \in \mathbb{N}_0.$$

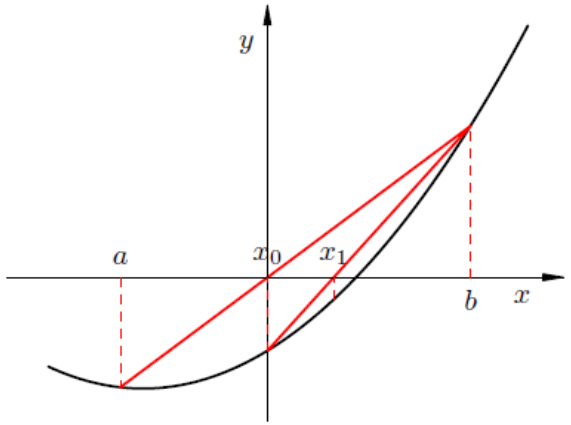
Opaska: U NR koristi se n-1 aproksimacija, tj. $n = \log_2 \frac{\epsilon_0}{\epsilon}$

Regula falsi

Aproksimirajmo funkciju f pravcem koji prolazi točkama $(a, f(a))$, $(b, f(b))$. Njegova je jednadžba

$$y - f(b) = \frac{f(a) - f(b)}{a - b}(x - b), \quad \text{odnosno} \quad y - f(a) = \frac{f(b) - f(a)}{b - a}(x - a).$$

Nultočku α funkcije f možemo aproksimirati nultočkom tog pravca, točkom x_0. Nakon toga, pomaknemo ili točku a ili točku b u x_0, ali tako da nultočka α ostane unutar novodobivenog intervala. Postupak ponavljamo sve dok ne postignemo željenu točnost.



Regula falsi

Točka x_0 dobiva se jednostavno iz jednadžbe pravca, pa je

$$x_0 = b - f(b)\,\frac{b-a}{f(b)-f(a)} = a - f(a)\,\frac{a-b}{f(a)-f(b)},$$

$$x_0 = b - \frac{f(b)}{f[a,b]} = a - \frac{f(a)}{f[a,b]}, \qquad f[a,b] = \frac{f(b)-f(a)}{b-a},$$

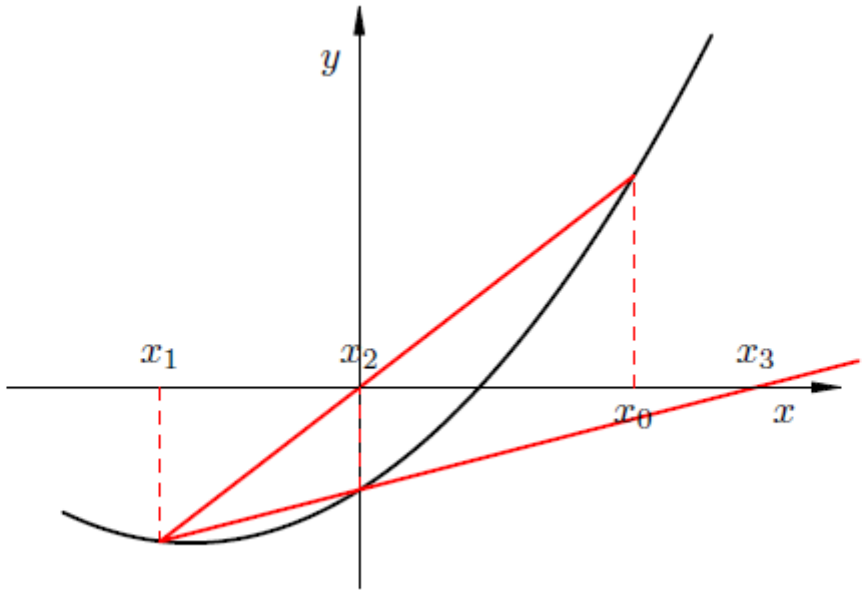
Metoda sekante

Ako graf funkcije f aproksimiramo sekantom, slično kao kod *regule falsi*, samo ne zahtijevamo da nultočka funkcije f ostane "zatvorena" unutar posljednje dvije iteracije, dobili smo metodu sekante. Time smo izgubili svojstvo sigurne konvergencije, ali se nadamo da će metoda, kad konvergira konvergirati brže nego *regula falsi*.

Počinjemo s dvije početne točke x_0 i x_1 i povlačimo sekantu kroz $(x_0, f(x_0))$, $(x_1, f(x_1))$. Ta sekanta siječe os x u točki x_2. Postupak nastavljamo povlačenjem sekante kroz posljednje dvije točke $(x_1, f(x_1))$ i $(x_2, f(x_2))$. Formule za metodu sekante dobivaju se iteriranjem početne formule za *regulu falsi*, tako da dobivamo

$$x_{n+1} = x_n - f(x_n)\,\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Metoda sekante



Primijetite da je treće iteracija izašla izvan početnog intervala, pa metoda sekante ne mora konvergirati. Jednako tako, da smo "prirodno" numerirali prve dvije točke, tako da je $x_0 < x_1$, imali bismo konvergenciju prema rješenju.

Metoda sekante

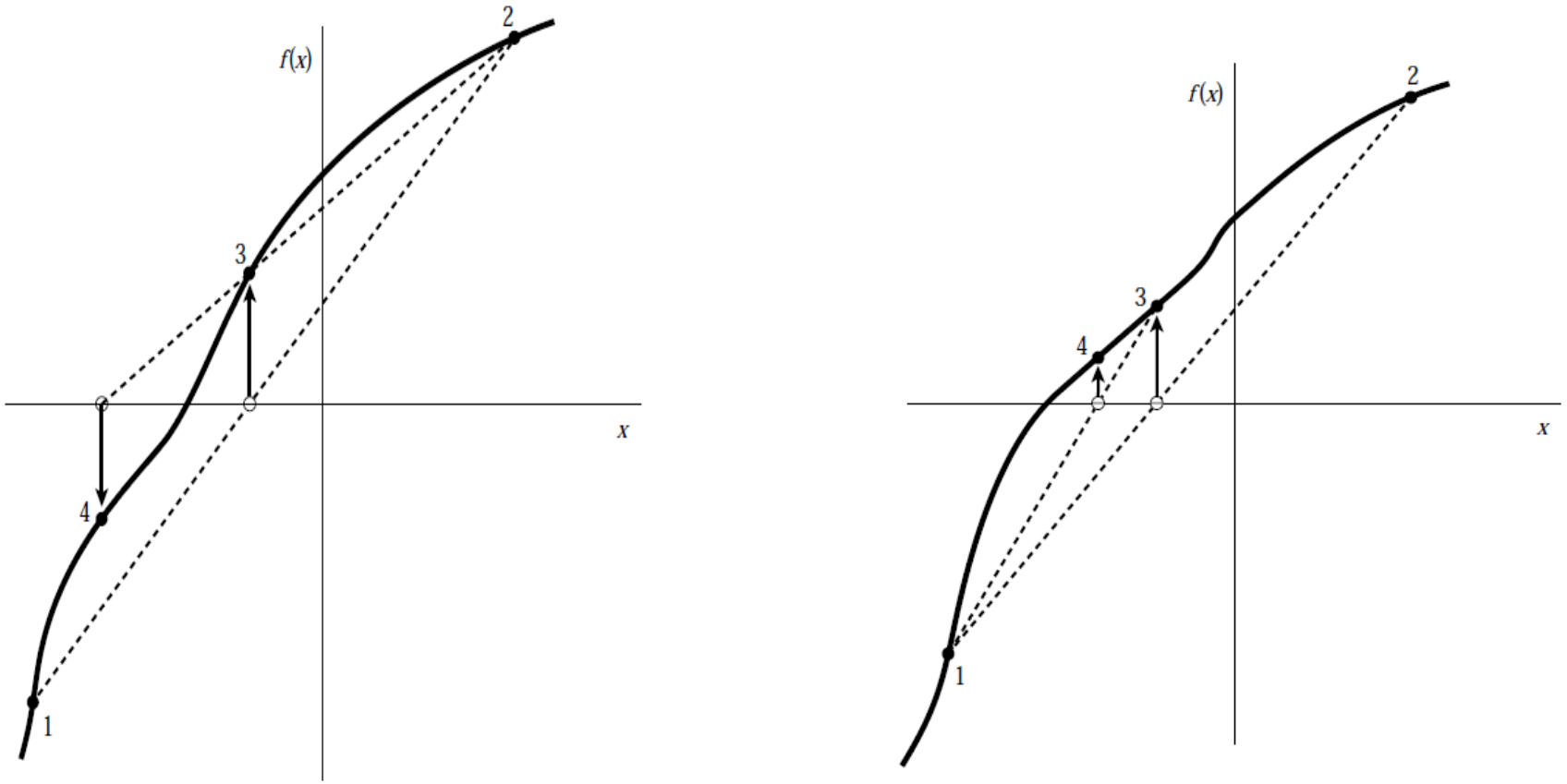
Kod metode sekante postoji nekoliko problema. Prvi je da može divergirati ako početne aproksimacije nisu dobro odabrane. Drugi problem se može javiti zbog kraćenja u brojniku i (posebno) nazivniku kvocijenta

$$\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})},$$

kad $x_n \to \alpha$. Osim toga, budući da iteracije ne "zatvaraju" nultočku s obje strane nije lako reći kad treba zaustaviti iterativni proces.

Konačno, primijetimo da je za svaku iteraciju metode sekante potrebno samo jednom izvrednjavati funkciju f i to u točki x_n, jer $f(x_{n-1})$ čuvamo od prethodne iteracije.

regula falsi i sekanta

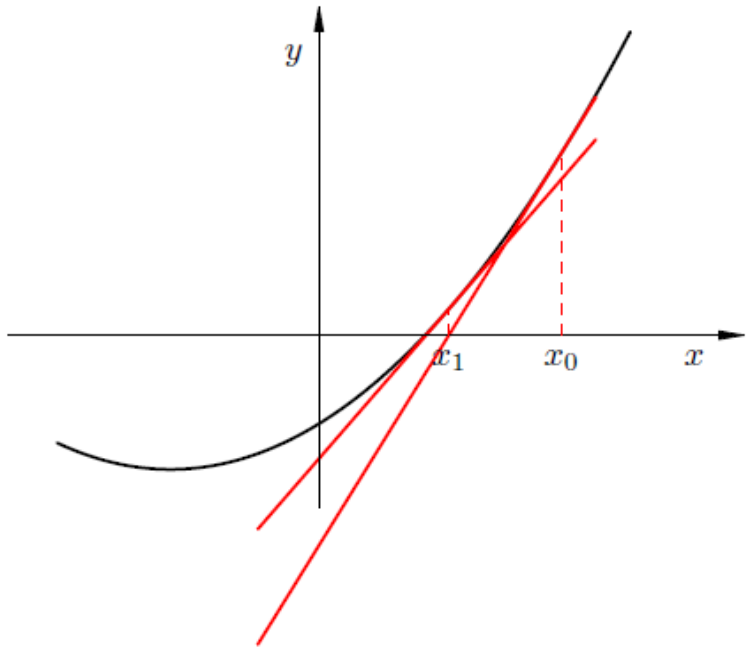


Metoda sekante i regula falsi metoda

Metoda tangente

Ako graf funkcije f umjesto sekantom, aproksimiramo tangentom, dobili smo metodu tangente ili Newtonovu metodu. Slično kao i kod sekante, time smo izgubili svojstvo sigurne konvergencije, ali se nadamo da će metoda brzo konvergirati.

Pretpostavimo da je zadana početna točka x_0. Ideja metode je povući tangentu u točki $(x_0, f(x_0))$ i definirati novu aproksimaciju x_0 u točki gdje ona siječe os x.



Metoda tangente

Geometrijski izvod je jednostavan. U točki x_n napiše se jednadžba tangente i pogleda se gdje siječe os x. Jednadžba tangente je

$$y - f(x_n) = f'(x_n)(x - x_n),$$

odakle izlazi da je nova aproksimacija $x_{n+1} := x$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Primijetite da je prethodna formula usko vezana uz metodu sekante, jer je

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Hibridna Brent-Dekker metoda

Brent–Dekkerova metoda smišljena je kao metoda koja će imati sigurnu konvergenciju, a nadamo se da će konvergirati brže nego metoda sekante, u najboljem slučaju kvadratično. Ona **ne zahtijeva** računanje derivacija, pa ako joj je red konvergencije u prosjeku bolji od sekante, možemo očekivati da će metoda po brzini biti slična Newtonovoj, ali će imati sigurnu konvergenciju.

Metoda se sasoji od tri dijela, koje grubo možemo opisati kao inverznu kvadratnu interpolaciju, metodu sekante i metodu bisekcije. Algoritam počinje metodom sekante koja generira treću točku. Ako se prema nekim kriterijima ta točka prihvaća kao dobra, možemo nastaviti raditi s kvadratnom interpolacijom kroz posljednje tri točke, ali inverznom (uloga x i y zamijenjena) i time dobivamo četvrtu točku.

Ako je treća točka odbačena kao loša, radi se jedan korak metode bisekcije. Drugim riječima, metoda se "vrti" između svoja tri sastavna dijela, a mi se nadamo da će rijetko koristiti bisekciju.

podprogrami

SUBROUTINE zbrac(func,x1,x2,succes)

Given a function func and an initial guessed range x1 to x2, the routine expands the range geometrically until a root is bracketed by the returned values x1 and x2 (in which case succes returns as .true.) or until the range becomes unacceptably large (in which case succes returns as .false.).

int zbrac(float (*func)(float), float *x1, float *x2)

root is bracketed by the returned values x1 and x2 (in which case zbrac returns 1) or until the range becomes unacceptably large (in which case zbrac returns 0).

SUBROUTINE zbrak(fx,x1,x2,n,xb1,xb2,nb)

Given a function fx dened on the interval from x1-x2 subdivide the interval into n equally spaced segments, and search for zero crossings of the function. nb is input as the maximum number of roots sought, and is reset to the number of bracketing pairs xb1(1:nb), xb2(1:nb) that are found.

void zbrak(float (*fx)(float), float x1, float x2, int n, float xb1[], float xb2[], int *nb)

podprogrami

FUNCTION rtbis(func,x1,x2,xacc)

PARAMETER (JMAX=40) Maximum allowed number of bisections.

Using bisection, find the root of a function func known to lie between x1 and x2. The root, returned as rtbis, will be refined until its accuracy is ±xacc.

float rtbis(float (*func)(float), float x1, float x2, float xacc)

FUNCTION rtflsp(func,x1,x2,xacc)

Using the false position method, find the root of a function func known to lie between x1 and x2. The root, returned as rtflsp, is refined until its accuracy is ±xacc.

float rtflsp(float (*func)(float), float x1, float x2, float xacc)

FUNCTION rtsec(func,x1,x2,xacc)

PARAMETER (MAXIT=30) Maximum allowed number of iterations.

Using the secant method, find the root of a function func thought to lie between x1 and x2. The root, returned as rtsec, is refined until its accuracy is ±xacc

float rtsec(float (*func)(float), float x1, float x2, float xacc)

podprogrami

FUNCTION zbrent(func,x1,x2,tol)

Using Brent's method, find the root of a function func known to lie between x1 and x2.

The root, returned as zbrent, will be refined until its accuracy is tol.

Parameters: Maximum allowed number of iterations, and machine floating-point precision.

float zbrent(float (*func)(float), float x1, float x2, float tol)

FUNCTION rtnewt(funcd,x1,x2,xacc)

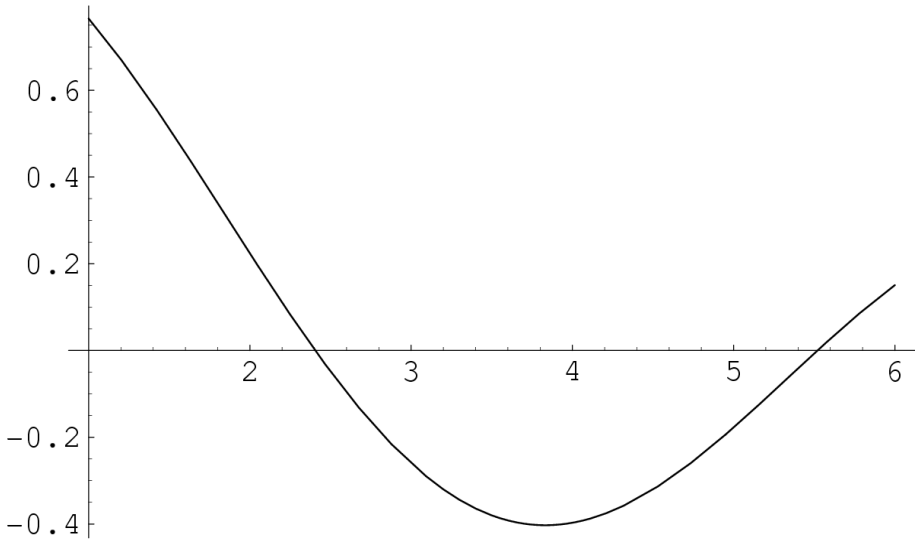
PARAMETER (JMAX=20) Set to maximum number of iterations.

Using the Newton-Raphson method, find the root of a function known to lie in the interval [x1; x2]. The root rtnewt will be refined until its accuracy is known within xacc. funcd is a user-supplied subroutine that returns both the function value and the first derivative of the function at the point x.

float rtnewt(void (*funcd)(float, float *, float *), float x1, float x2, float xacc)

NR primjeri

Besselova funkcija 0-tog reda prve vrste $J_n(x)$




```
for (i=1;i<=10;i++) {
                    x1=i;
                    x2=x1+1.0;
                    succes=zbrac(fx,&x1,&x2);
          .....
```

Bracketing values: Function values:

X1	X2	BESSJ0(X1)	BESSJ0(X2)
1.00	3.60	0.765198	-0.391769
2.00	3.00	0.223891	-0.260052

```
f77 -o Fzbrac xzbrac.for zbrac.for bessj0.for
 gcc -o Czbrac xzbrac.c zbrac.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
f77 -o Fzbrak xzbrak.for zbrak.for bessj0.for
gcc -o Czbrak xzbrak.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
```

NR primjeri

```
gcc -o Czbrac xzbrac.c zbrac.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
f77 -o Fzbrak xzbrak.for zbrak.for bessj0.for
```

```
xb1=vector(1,NBMAX);
        xb2=vector(1,NBMAX);
        zbrak(fx,X1,X2,N,xb1,xb2,&nb);
```

	lower	upper	F(lower)	F(upper)
Root 1	1.9800	2.4700	0.2354	-0.0334
Root 2	5.4100	5.9000	-0.0378	0.1220

```
gcc -o Crtbis xrtbis.c rtbis.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/-lm
f77 -o Frtbis xrtbis.for  rtbis.for zbrak.for bessj0.for
```

```
xb1=vector(1,NBMAX);
xb2=vector(1,NBMAX);
zbrak(fx,X1,X2,N,xb1,xb2,&nb);
for (i=1;i<=nb;i++) {
  xacc=(1.0e-6)*(xb1[i]+xb2[i])/2.0;
   root=rtbis(fx,xb1[i],xb2[i],xacc);
```

	x	F(x)
Root 1	2.404827	-0.5610E-06
Root 2	5.520077	-0.4607E-06

Analogni primjer za metodu sekante i false position

```
f77 -o Frtsec xrtsec.for  rtsec.for zbrak.for bessj0.for
gcc -o Crtsec xrtsec.c rtsec.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
f77 -o Frtflsp xrtflsp.for  rtflsp.for zbrak.for bessj0.for
gcc -o Crtflsp xrtflsp.c rtflsp.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm
```

NR primjeri

f77 -o Fzbrent xzbrent.for zbrent.for zbrak.for bessj0.for

gcc -o Czbrent xzbrent.c zbrent.c zbrak.c bessj0.c nrutils/nrutil.c -I nrutils/ -lm

./Czbrent

Roots of bessj0:

		x	f(x)
root	1	2.404826	-0.000000
root	2	5.520078	0.000000

```
xb1=vector(1,NBMAX);
        xb2=vector(1,NBMAX);
        zbrak(fx,X1,X2,N,xb1,xb2,&nb);
        for (i=1;i<=nb;i++) {
                tol=(1.0e-6)*(xb1[i]+xb2[i])/2.0;
                root=zbrent(fx,xb1[i],xb2[i],tol);
```

f77 -o Frtnewt xrtnewt.for rtnewt.for zbrak.for bessj0.for bessj1.for

gcc -o Crtnewt xrtnewt.c rtnewt.c zbrak.c bessj0.c bessj1.c nrutils/nrutil.c -I nrutils/ -lm

Isti princip za metodu Newtona:

Roots of bessj0:

		x	f(x)
root	1	2.404825	0.000000
root	2	5.520078	0.000000
root	3	8.653728	0.000000

Primjer

Treći korijen broja 1.5 može se prikazati pomoću jedandžbe $x^3-1=0$, odnosno nalaženje nultočke funkcije $f(x)=x^3-1$.

Metoda bisekcije:

f77 -o Fbis trtbis.for rtbis.for zbrak.for

gcc -o Cbis trtbis.c rtbis.c zbrak.c nrutils/nrutil.c -I nrutils/ -lm

Newtonova metoda:

f77 -o Fnewt trtnewt.for rtnewt.for zbrak.for

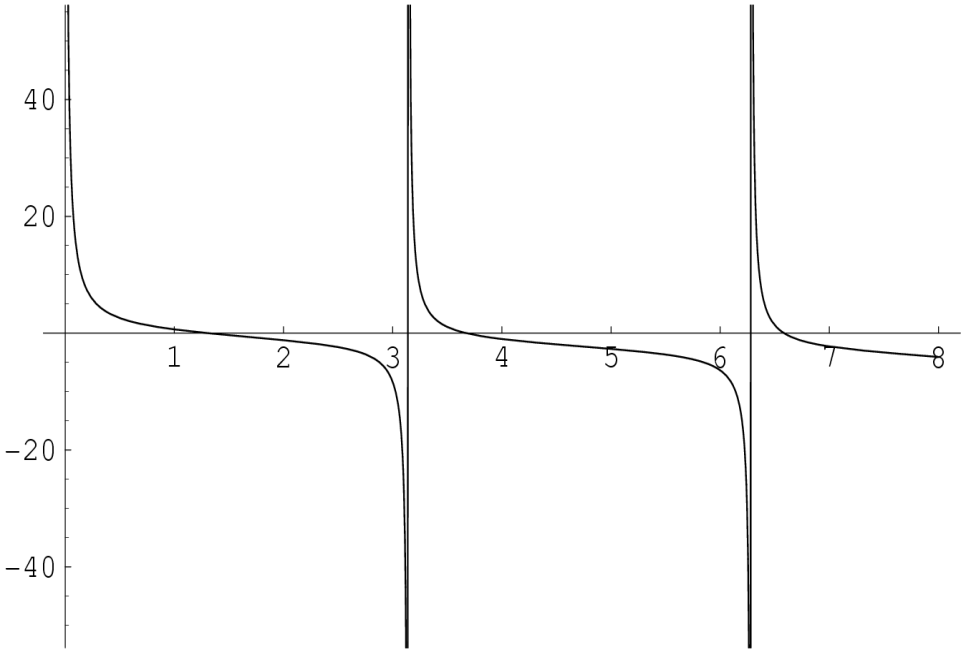
gcc -o Cnewt trtnewt.c rtnewt.c zbrak.c nrutils/nrutil.c -I nrutils/ -lm

Obratite pažnju na definiciju prve derivacije kod ove metode.

Zadatak za praktikum

Prilikom transporta neutrona u štapu dolazi do transcendentalne jednadžbe. O korijenima jednadžbe ovise kritične duljine, za štap duljine L jednadžba je

$$\cot(\ell x) = \frac{x^2 - 1}{2x}.$$



Graf prikazuje funkciju za vrijednost L=1.

Odredi prve dvije najmanje pozitivne nultočke.

Koristite graf kako bi odredili interval za traženje nultočke.

cot=1/tan, csc=1/sin, f' - prva derivacija

f'(x)=-csc(x)^2+(x^2-1)/(2*x^2)-1

Pošaljite rezultat i source na mail Aleksandar.Maksimovic@irb.hr

Literatura

- Online literatura:
  - Numerička matematika-osnovni udžbenik, PMF, projekt mzt.
  - Numerical Recipes in C
  - Numerical Recipes in Fortran
- L. F. Shampine, R. C. Allen, Jr., S. Pruess: FUNDAMENTALS OF NUMERICAL COMPUTING, John Wiley & Sons, Inc. (1997)
- George Em Karniadakis and Robert M. Kirby II: Parallel Scientific Computing in C++ and MPI, Cambridge University Press.