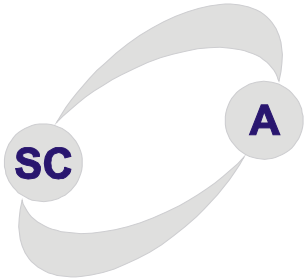
Predavanje

Predmet

Metodi optimizacije

Tema

Statička optimizacija, numeričke metode jednodimenzione optimizacije



2022 godina

Statička optimizacija, numeričke metode jednodimenzione optimizacije

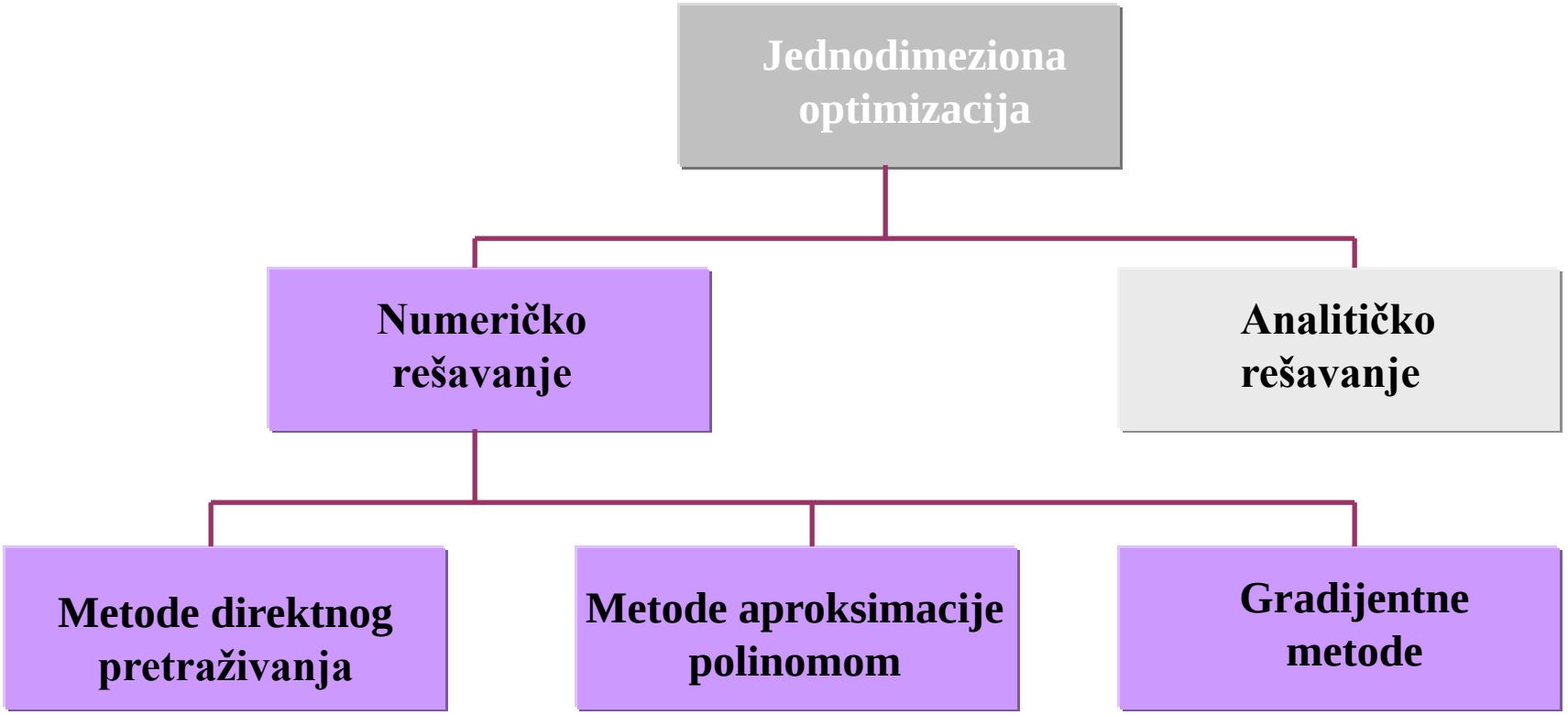
Da li su jednodimenzioni problemi laki? (Ponekad)

Da li se metodi višedimenzione optimizacije, mogu koristiti kod sistema sa jednom promenljivom? (Da)

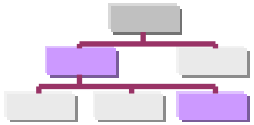
ali,

1. **Postoji određen broj važnih jednodimenzionih problema.**
2. **Predstavljaju osnovu višediemnzione numeričke optimizacije.**
3. **Jednodimenziona optimizacija je često sastavni deo složenih optimizacionih problema i softvera!**

Statička optimizacija, jednodimenziona optimizacija



Gradijentne metode



Osnovna ideja ovih metoda

Naći stacionarne tačke funkcije ($f'(x)=0$)

(ako je funkcija diferencijabiln do reda koji nam je potreban).

Newton-Raphson Metod

$f(x)$ *se razvija u Taylor-ov red oko* x_0

$$f(x) \approx g(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2$$

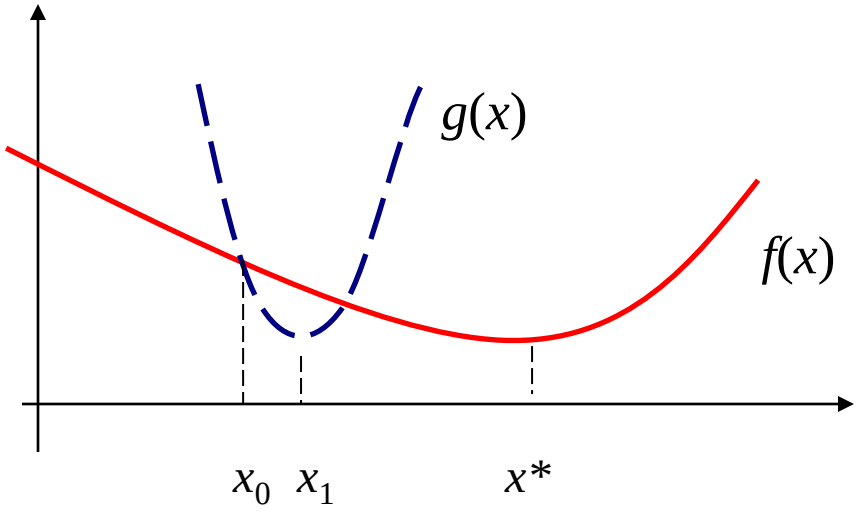
$g(x)$ *je parabola čiji je ekstrem u* x_1,

a uzima se kao polazna tačka za sledeću iteraciju

$f(x)$ *se razvija u Taylor-ov red oko* x_0

$$f(x) \approx g(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2$$

$g(x)$ *je parabola čiji je ekstrem u* x_1,
a uzima se kao polazna tačka za sledeću iteraciju



$$g'(x_1) = 0$$

$$f'(x_0) + f''(x_0)(x_1 - x_0) = 0$$

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

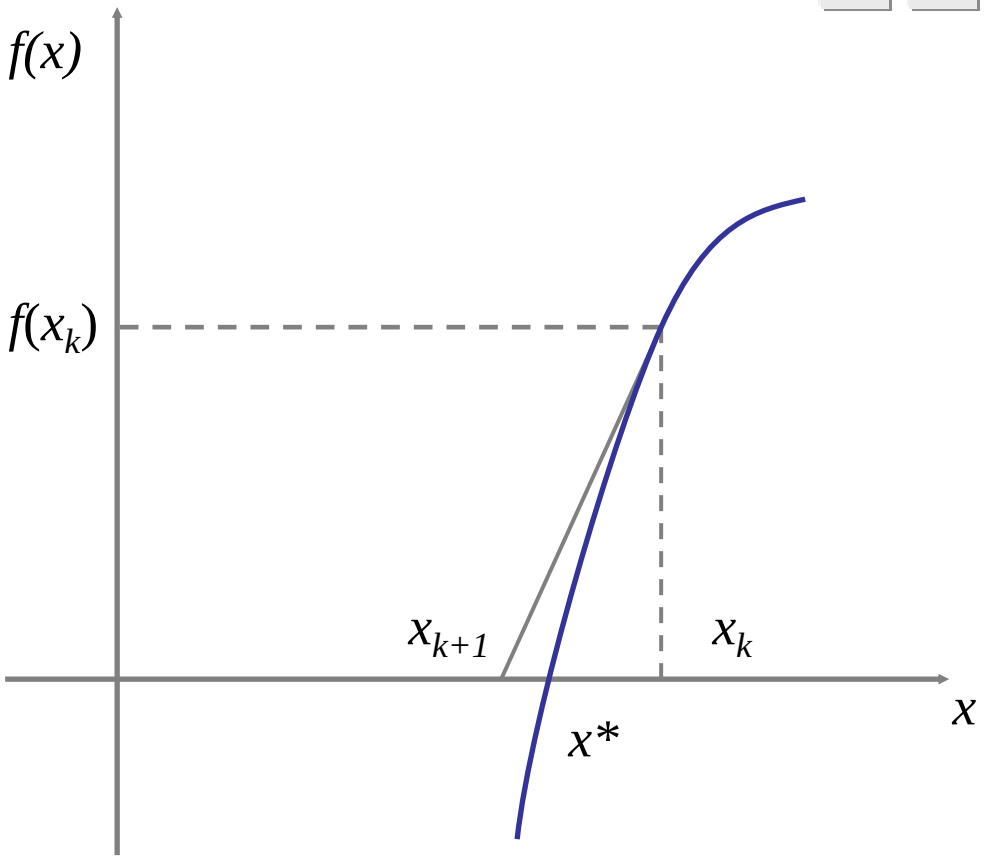
$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

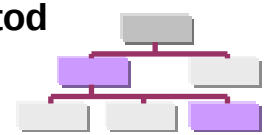
Newton-Raphson Metod

$$f'(x_k) = \frac{f(x_k)}{x_k - x_{k+1}}$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

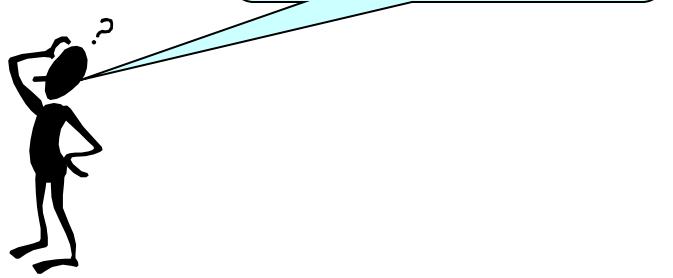
$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$





```
def newtonRaphson(x0,tol):
    x_novo=x0;
    x_pre=math.inf
    iter=0
    while(abs(x_pre-x_novo)>tol):
        iter+=1
        x_pre=x_novo
        x_novo=x_pre-dfunc(x_pre)/ddfunc(x_pre)

    xopt=x_novo
    fopt=func(xopt)
    return xopt,fopt,iter
```

- Početno pogađanje $$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$$
- Iterativni postupak $$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$
- Kriterijum zaustavljanja $$\varepsilon_{n+1} = \left| \frac{x_{n+1} - x_n}{x_{n+1}} \right|$$



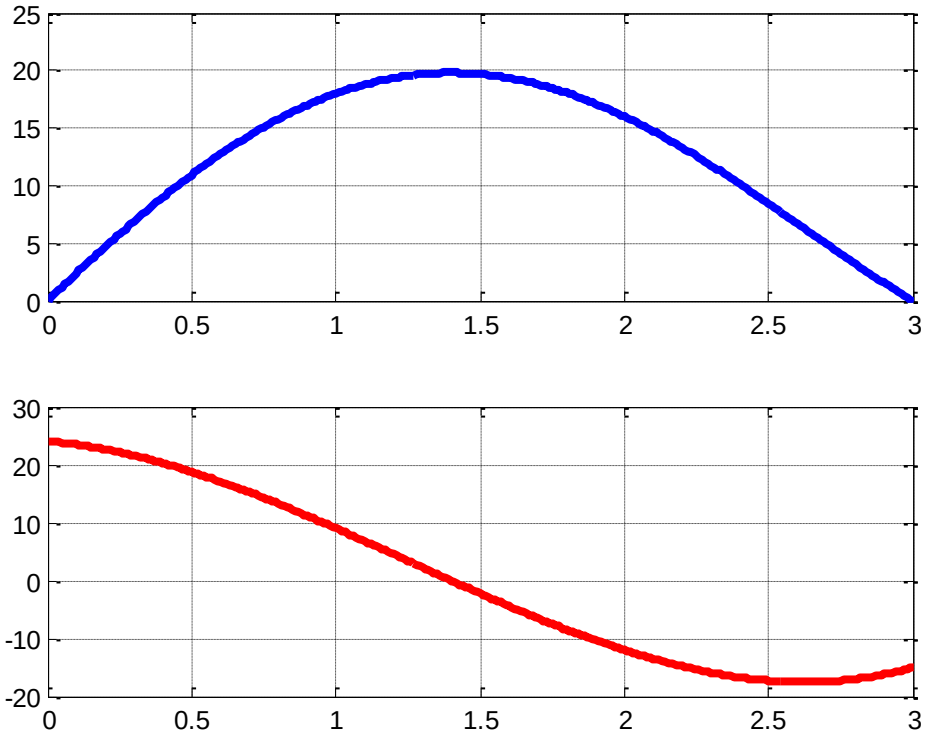
Primer

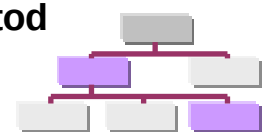
$$f(x) = x^4 - 5x^3 - 2x^2 + 24x$$

$$f'(x) = 4x^3 - 15x^2 - 4x + 24$$

$$f''(x) = 12x^2 - 30x - 4$$

$$x = [0,3]$$

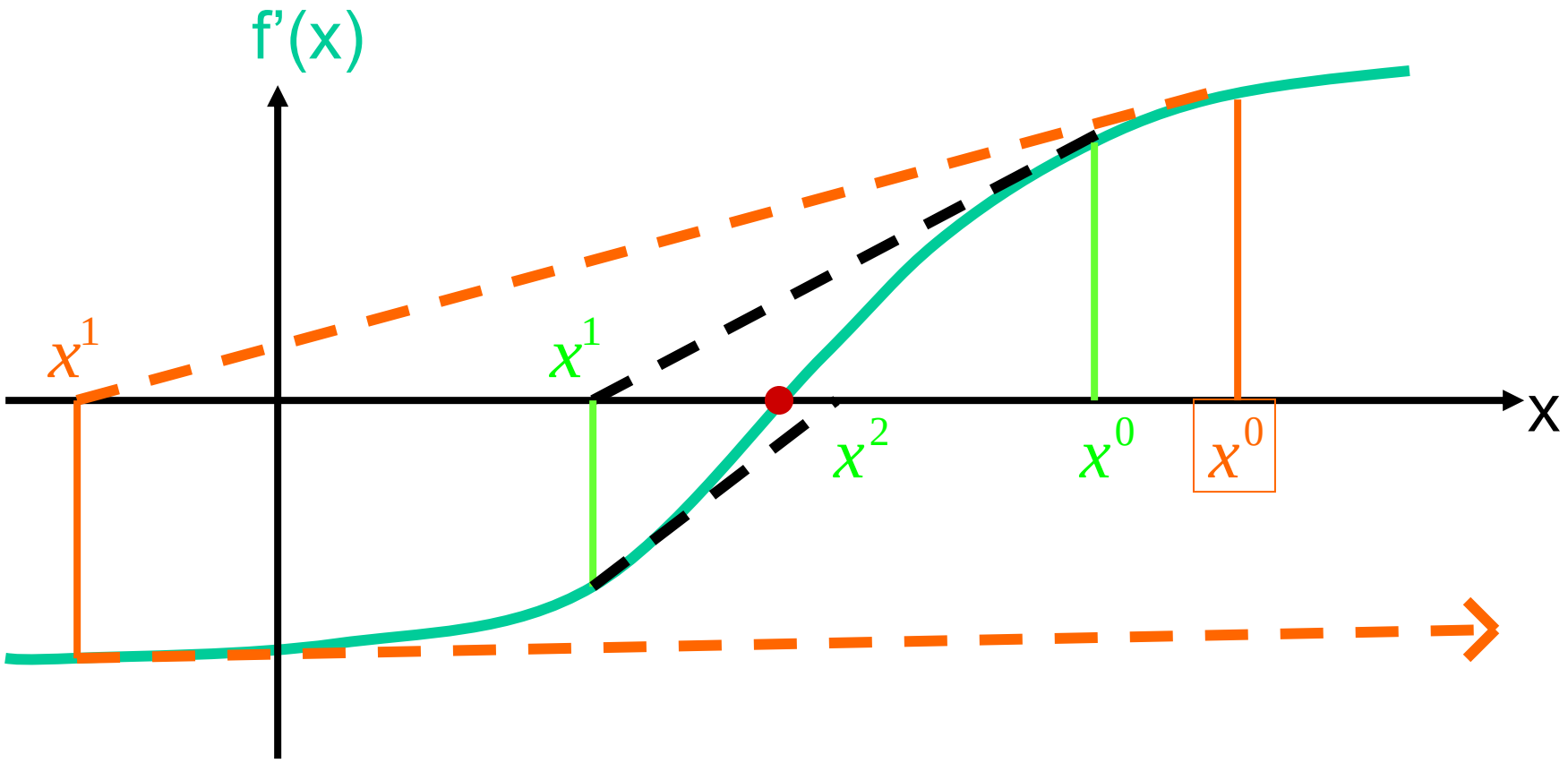




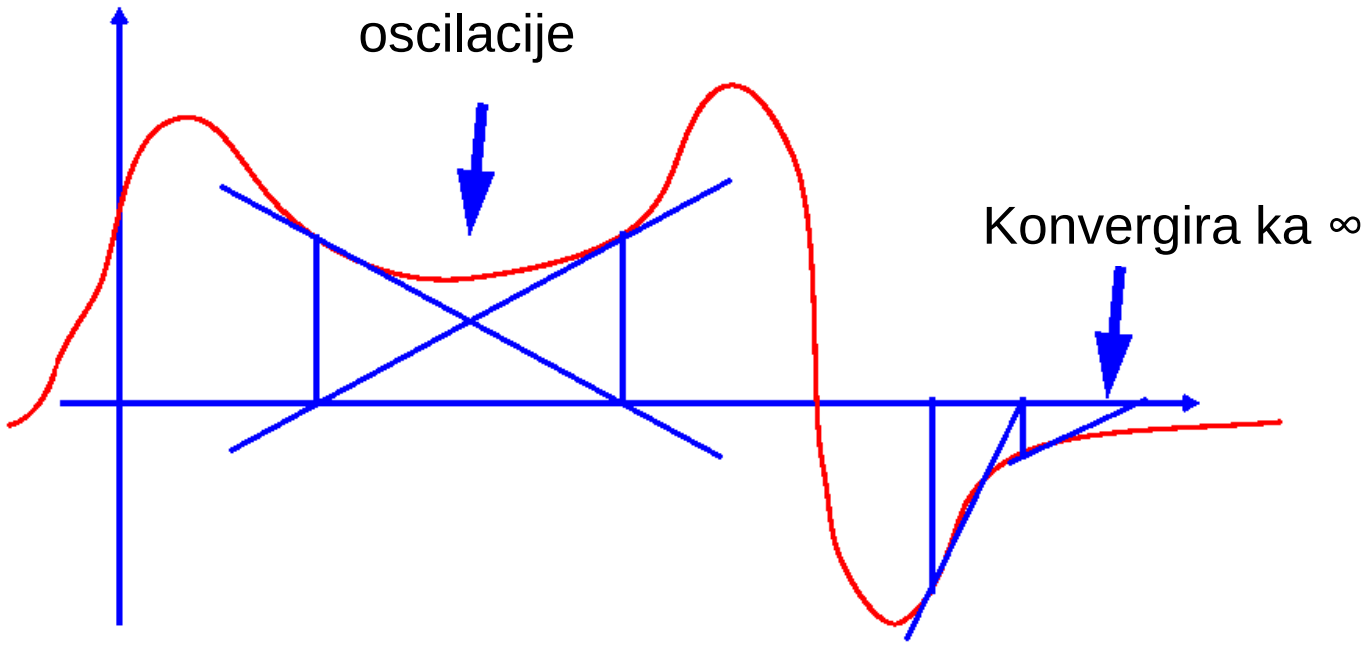
Primer

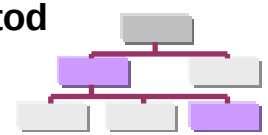
```
def func(x):
    f=x**4-5*x**3-2*x**2+24*x
    return f

def dfunc(x):
    f=4*x**3-15*x**2-4*x+24
    return f
def ddfunc(x):
    f=12*x**2-30*x-4
    return f


##################################################
tol=0.0001
init_guess=1
[xopt,fopt,iter]=newtonRaphson(init_guess,tol)
print(xopt,fopt,iter)

1.3989324753691192 19.801612810659165 3
```

Konvergencija zavisi od dobrog početnog pogađanja



Konvergencija zavisi od dobrog početnog pogađanja





domaći

$$f(x)' = (x-1)^3 = 0$$

$$f(x)' = x^3 - 0.03x^2 + 2.4\text{x}10^{-6} = 0$$

$$f(x)' = Sin\ x = 0$$

$$f(x)' = x^2 + 2 = 0$$

Gradijentne metode

Metod Sečice

- Ako se drugi izvod zameni konačnom razlikom,

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}$$

Newton-va metoda postaje

$$x_{k+1} = x_k - f'(x_k)\frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})} \qquad k=1,2,\ldots$$

- tačke x_0, x_1 su proizvoljne

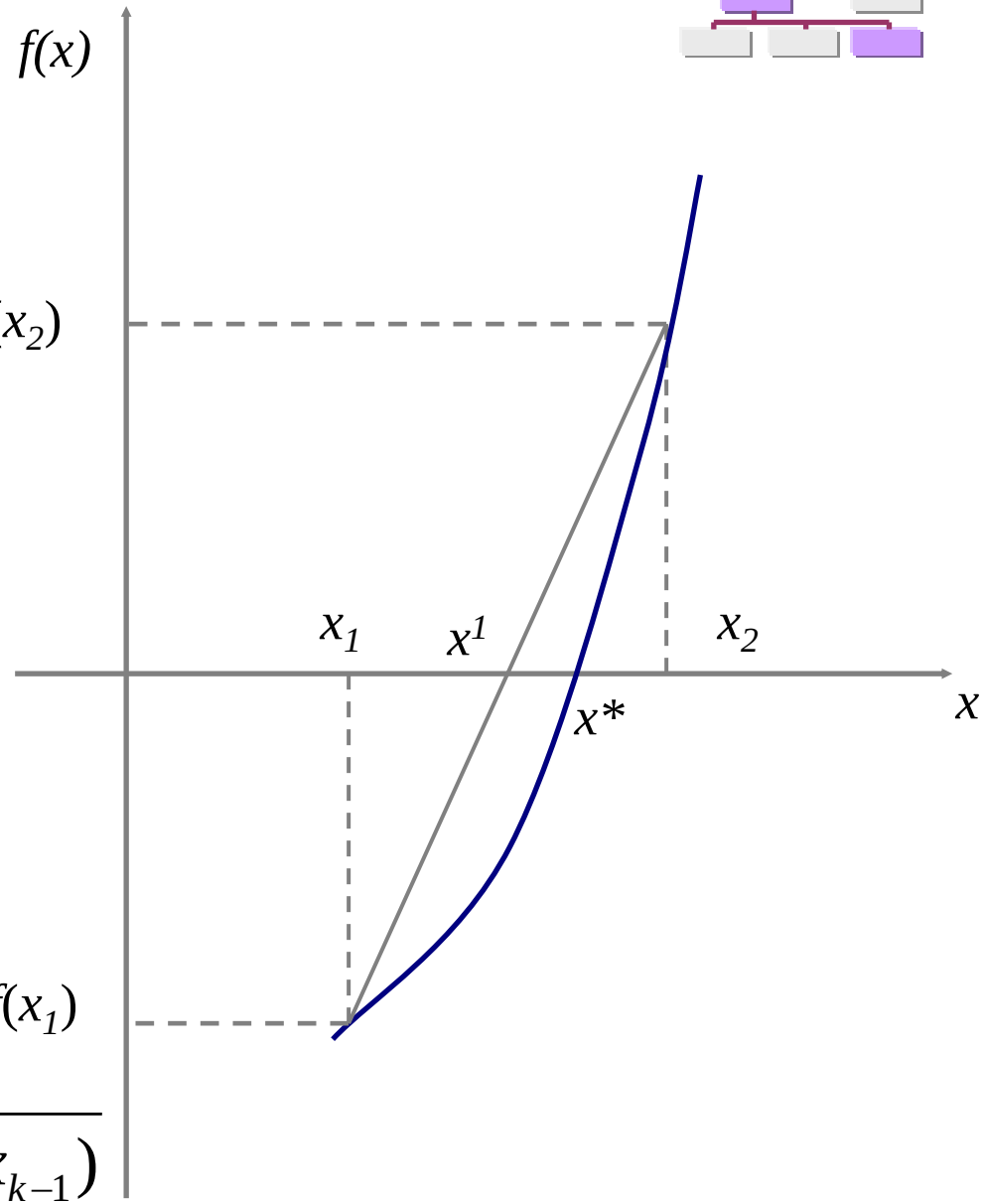
Metod Sečice

$$\frac{x^1 - x_1}{-f(x_1)} = \frac{x_2 - x_1}{f(x_2) - f(x_1)}$$

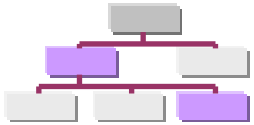
$$x^1 = x_1 - \frac{x_2 - x_1}{f(x_2) - f(x_1)} f(x_1)$$

$$x^1 = x_1 - \frac{x_2 - x_1}{f'(x_2) - f'(x_1)} f'(x_1)$$

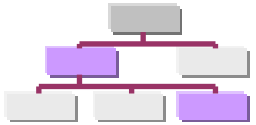
$$x_{k+1} = x_k - f'(x_k) \frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})}$$



Metod Sečice



```
def secica(x0,x1,tol):
    x_pre=x0
    x_ppre=math.inf
    x_novo=x1
    iter=0
    while(abs(x_novo-x_pre)>tol):
        iter+=1
        x_ppre = x_pre
        x_pre = x_novo
        x_novo=x_pre-dfunc(x_pre)*(x_pre-x_ppre)/(dfunc(x_pre)-
dfunc(x_ppre))

    xopt=x_novo
    fopt=func(xopt)
    return xopt,fopt,iter
```



- Početno pogađanje $x_2 = x_1 - f'(x_1)\dfrac{x_1 - x_0}{f'(x_1) - f'(x_0)}$
- Iterativni postupak $x_{k+1} = x_k - f'(x_k)\dfrac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})}$
- Kriterijum zaustavljanja $\varepsilon_{n+1} = \left|\dfrac{x_{n+1} - x_n}{x_{n+1}}\right|$

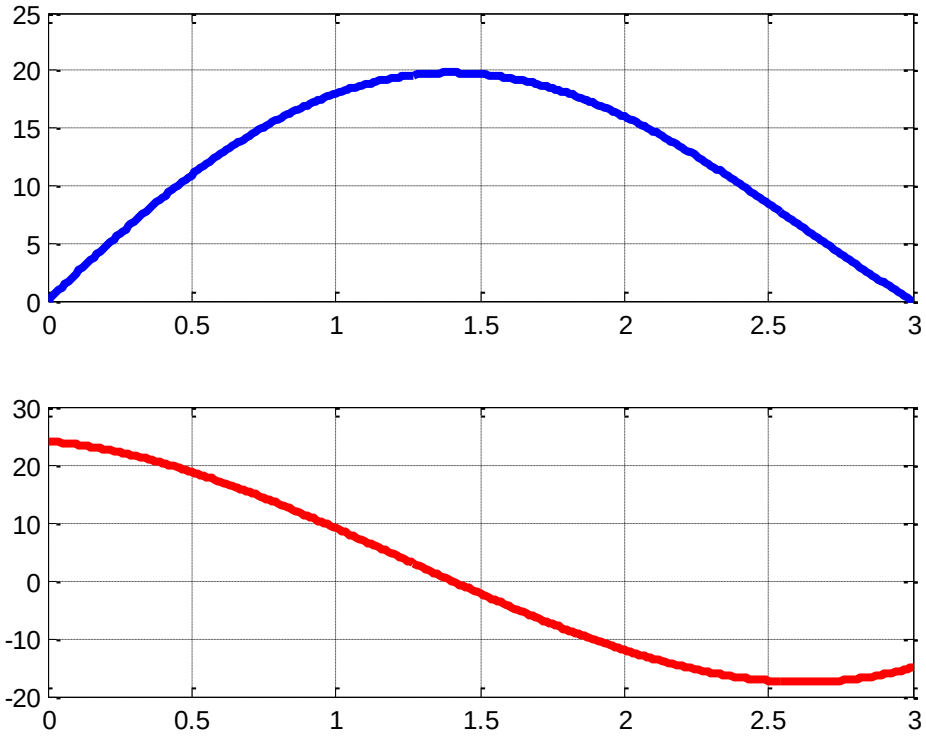
Primer

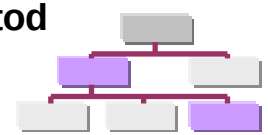
$$f(x) = x^4 - 5x^3 - 2x^2 + 24x$$

$$f'(x) = 4x^3 - 15x^2 - 4x + 24$$

$$f''(x) = 12x^2 - 30x - 4$$

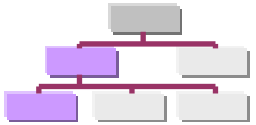
$$x = [0,3]$$





```
def func(x):
    f=x**4-5*x**3-2*x**2+24*x
    return f

def dfunc(x):
    f=4*x**3-15*x**2-4*x+24
    return f

##################################################
tol=0.0001
init_guess1=0
init_guess2=3

[xopt,fopt,iter]=secica(init_guess1,init_guess2,tol)
print(xopt,fopt,iter)

1.3989324753831391 19.8016128106591657
```

Metode direktnog pretraživanja

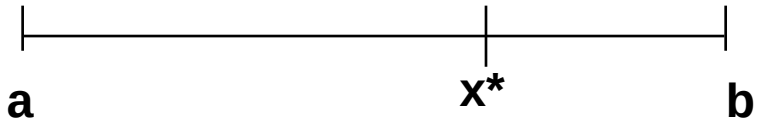


Metod direktnog pretraživanja su u osnovi metode jednodimenzione optimizacije

Smatraju ih "kičmom" nelinearnih optimizacionih algoritama

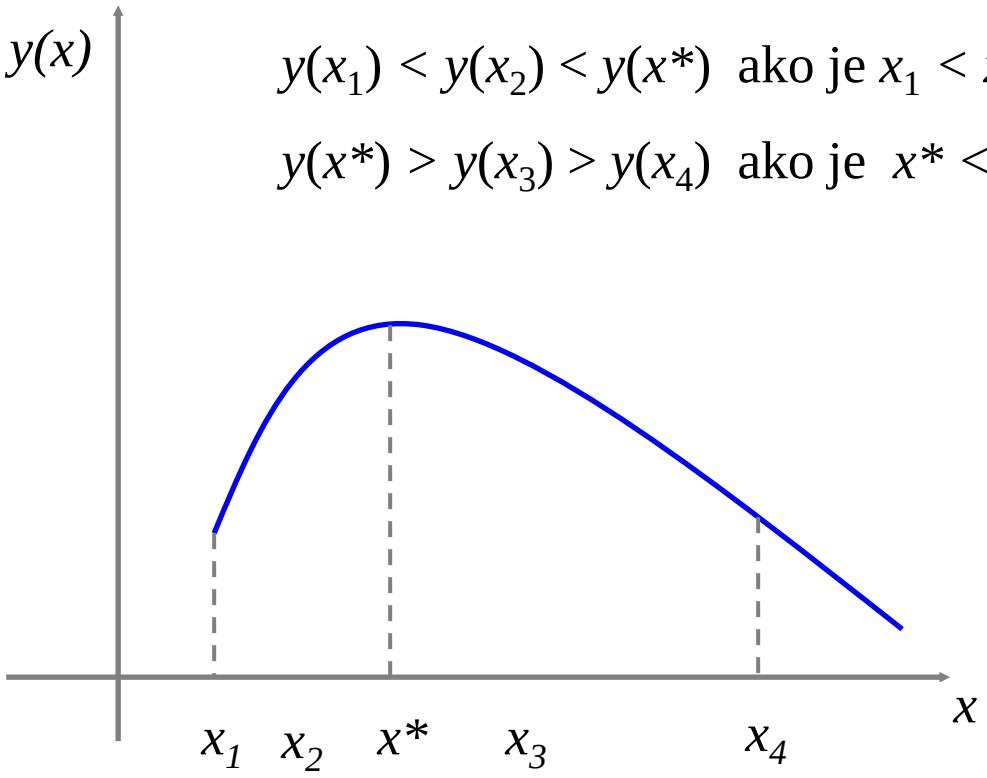
Svode se na pretraživanje zatvorenog intervala

Često pretpostavljamo da je funkcija unimodalna



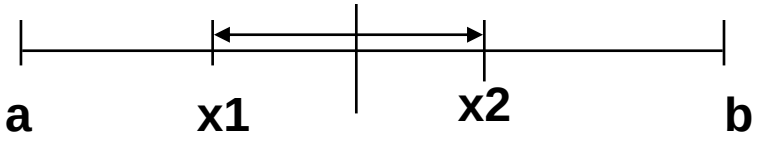
Detaljno pretraživanje zahteva $N = (b-a)/\varepsilon + 1$ proračuna u interavalu sa slike, pri čemu je ε rezolucija.

unimodalnost



$y(x_1) < y(x_2) < y(x^*)$ ako je $x_1 < x_2 < x^*$

$y(x^*) > y(x_3) > y(x_4)$ ako je $x^* < x_3 < x_4$



Pretraga da bi se našao **min** $f(x)$:

0) pretpostaviti interval [a,b]

1) naći po nekoj formuli (npr x1 = a + (b-a)/2 - ε/2 i x2 = a+(b-a)/2 + ε/2 gde je ε rezolucija).

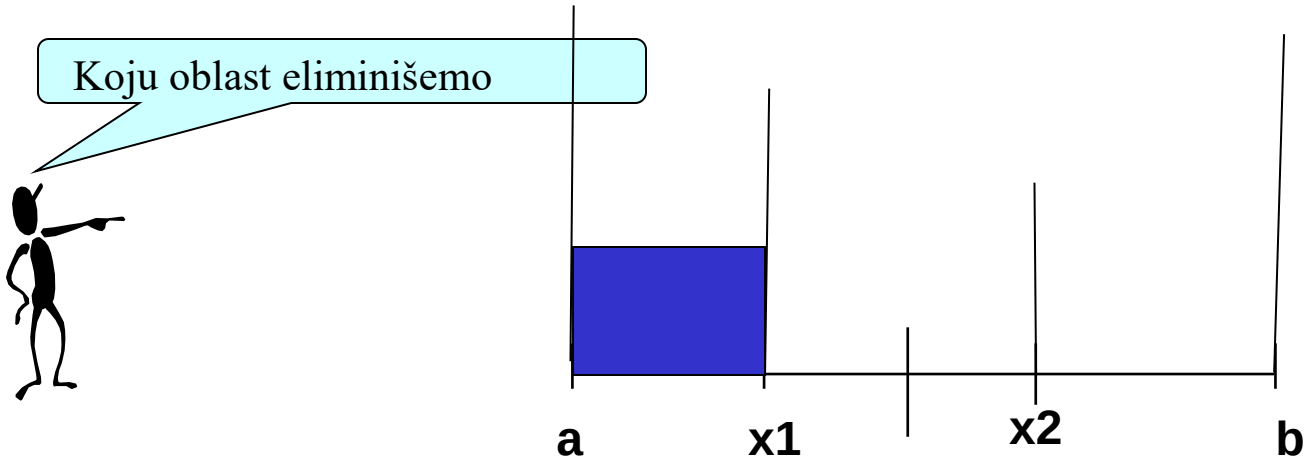
2) porediti f(x1) i f(x2)

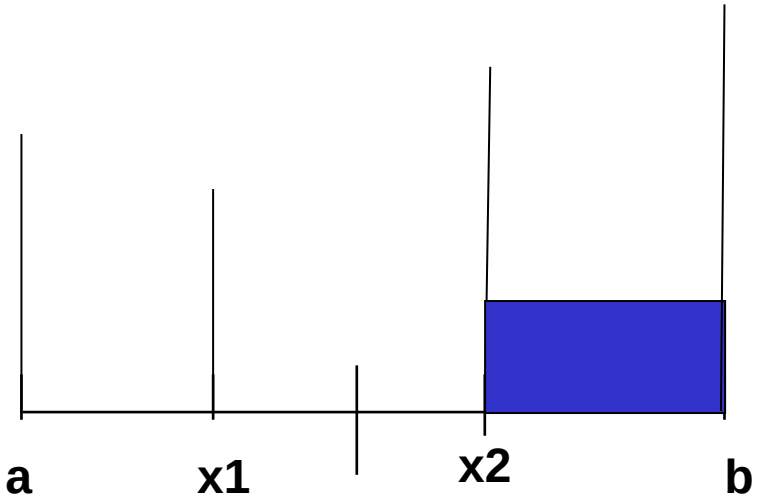
3) Ako je f(x1) < f(x2) tada eliminišemo x > x2 i b = x2

Ako f(x1) > f(x2) tada eliminišemo x < x1 i a = x1

Ako f(x1) = f(x2) tada biramo novi par tačaka

4) Nastaviti dok interval ne bude < 2 ε


Koju oblast eliminišemo
a
x1
x2
b


a
x1
x2
b

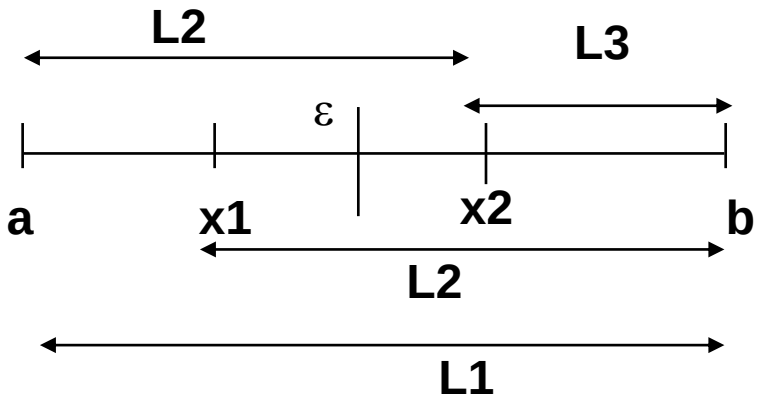
Metod direktnog pretraživanja

Fibonačijev Metod

Fibonačijevi brojevi:

1,1,2,3,5,8,13,21,34,.. odnosno , suma dva prethodna

$F_n = F_{n-1} + F_{n-2}$



L1 = L2 + L3

Može se pokazati

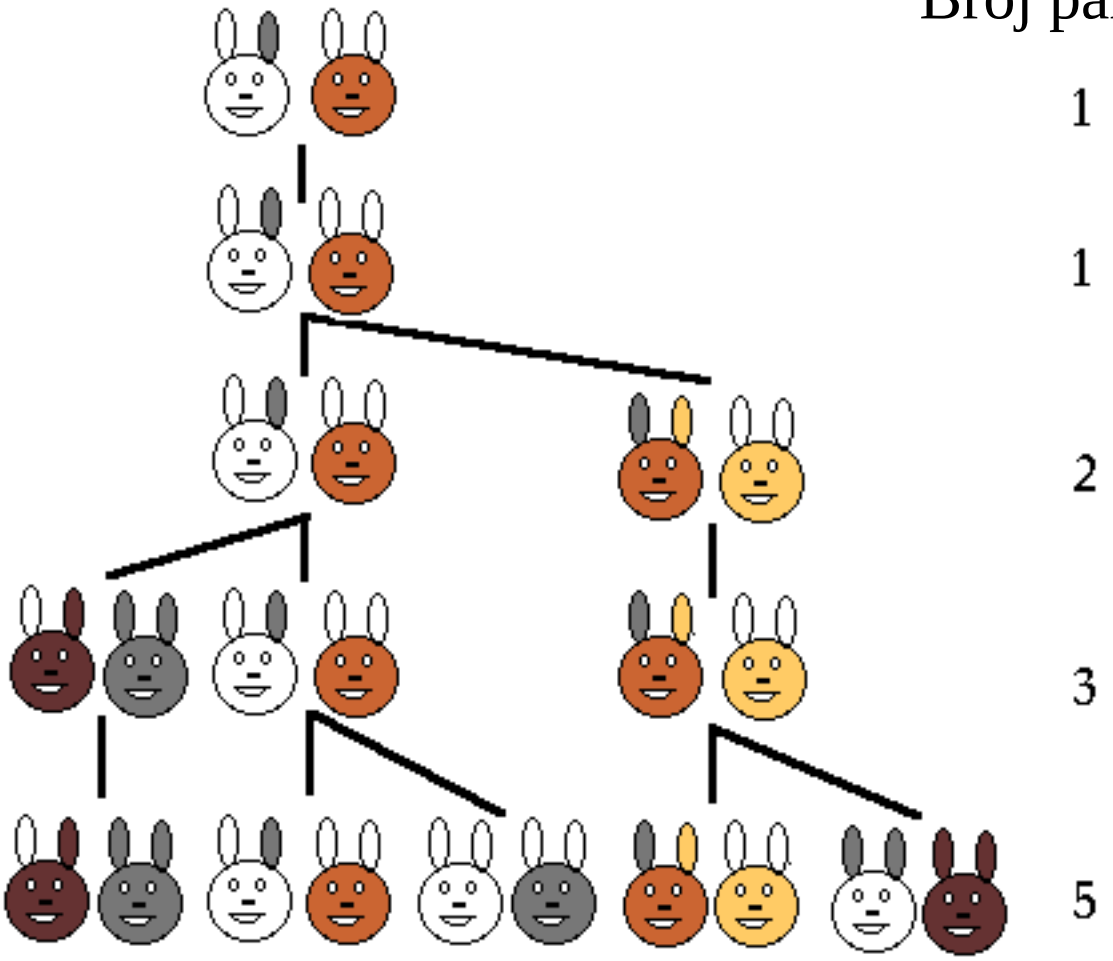
$L_n = (L1 + F_{n-2}\,\varepsilon) / F_n$

1200 te...

Fibonači je postavio pitanje…

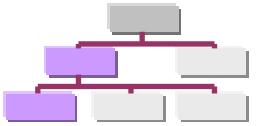
Leonardo Pisano (1170-1250)

Pretpostavimo da se tek rođeni par, mužjak i ženka, zečeva stavi u polje. Zečevi su u stanju da se razmnožavaju posle mesec dana i na kraju drugog meseca ženka može da dobije novi par zečeva. Pretpostavimo da zečevi ne umiru i da ženka uvek daje novi par (muško-žensko) svaki mesec od drugog meseca. Koliko će parova biti na kraju prve godine.


Broj parova
1
1
2
3
5

algoritam

Fibonačijev metod



1. Odrediti interval L_0 [a,b] (a<b), koji sadrži,tačku x* i specificirati rezoluciju-tačnost aproksimacije ε>0. Interval
2. Odrediti najmanji prirodan broj *n* koji zadovoljava uslov:

$$F_n > \frac{1}{\varepsilon}(b-a) \quad \text{ili} \quad F_n > \frac{L_0}{\varepsilon}$$

3. Izračunati prvi interval (prva iteracija)

$$x_1 = a + \frac{F_{n-2}}{F_n}(b-a)$$

$$x_2 = a + b - x_1$$

4. Izračunati k ti interval i ponavljati postupak (korak 4.) sve do k=n

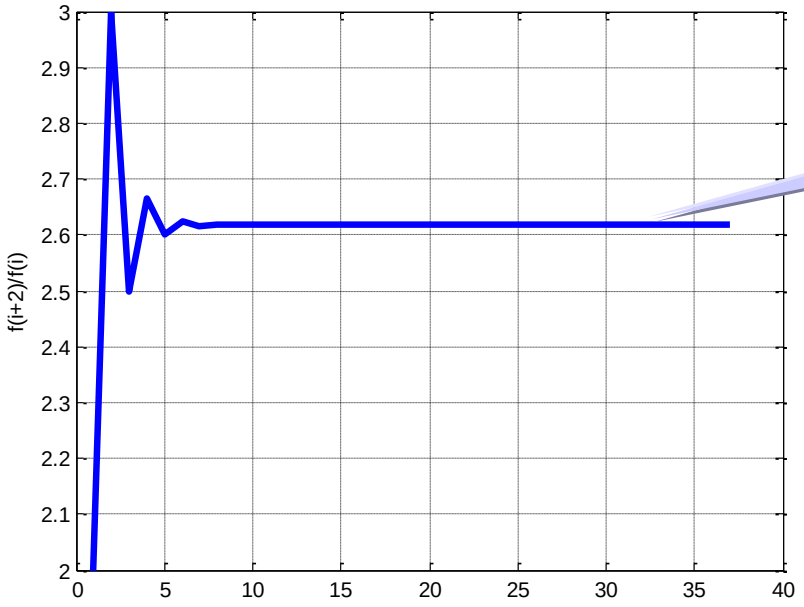
Odrediti $f(x_1)$ i $f(x_2)$, pa napraviti novi set tačaka a i b po principu:

Ako je f(x1) <= f(x2) tada eliminišemo x > x2 i b = x2, x2=x1, x1 = a + b - x1.
Ako f(x1) > f(x2) tada eliminišemo x < x1 i a = x1, x1=x2, x2 = a + b - x2
Ako f(x1) = f(x2) tada biramo novi par tačaka

```
def fibonaci_metod(a,b,tol):
    #Fibonacci - jev postupak minimizacije funckcije FUNCjedne
promenljive.
    # Funkcija mora biti unimodalna nad intervalom[a, b].
    # Tol je trazena sirina intervala u kome se nalazi minimum.
    ## Korak 1 - Trazimo najmanji broj n koji zadovoljava uslov
    n=1
    while (abs(b-a)/tol)>fibonaci_broj(n):
        n+=1

    ## Korak 2 - Odredjujemo pocetne tacke

    x1 = a + fibonaci_broj(n-2)/fibonaci_broj(n)*(b-a)
    x2 = a +b-x1
```

```
## Korak 3 - Iteracije
    # Radimo n - 1 iteracija, posle cega je (b - a) < tol

for i in range(2,n+1):

    if func(x1)<=func(x2):
        b=x2
        x2=x1
        x1=a+b-x2
    else:
        a=x1
        x1=x2
        x2=a+b-x1
if func(x1)<func(x2):
    xopt=x1
    fopt=func(x1)
else:
    xopt=x2
    fopt=func(x2)

return xopt,fopt,n
```

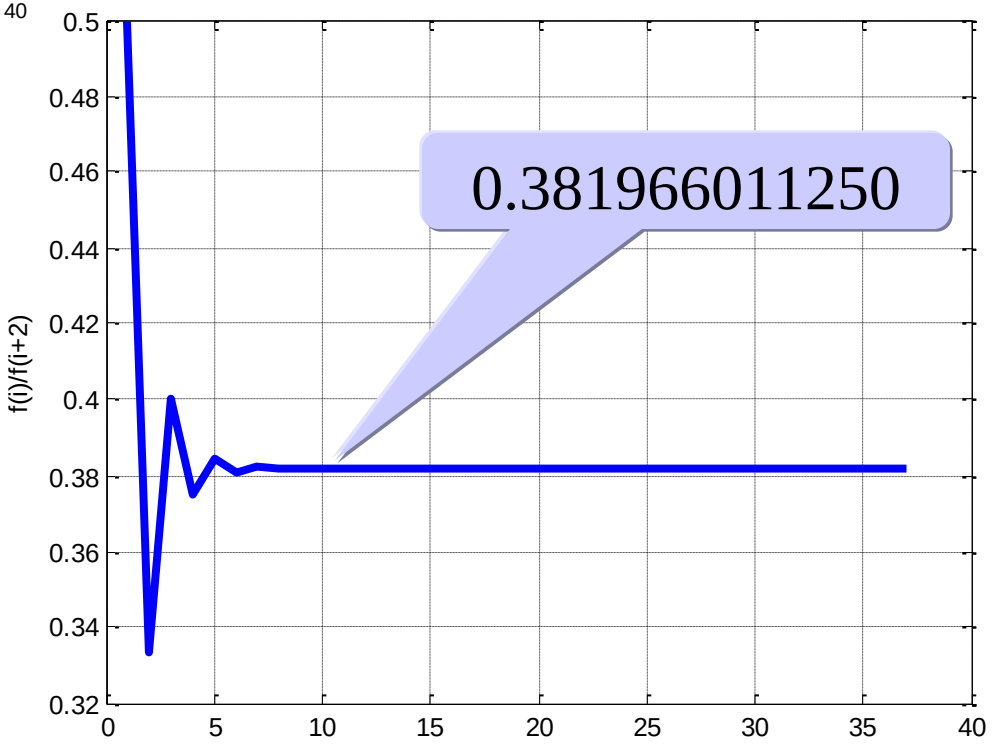
```
def func(x):
    f=-1*(x**4-5*x**3-2*x**2+24*x)
    return f

a=0
b=3
tol=0.0001

xopt,fopt,n=fibonaci_metod(a,b,tol)


1.398938923395339 -19.801612810191763 24
```



2.618033988749

F(i)/F(i+2)

F(i+2)/F(i)

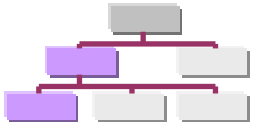


0.381966011250

Posle 15 brojeva podudaraju se na 5 decimalnih mesta

$$\frac{F_n}{F_{n+2}} \approx \frac{3-\sqrt{5}}{2}$$

Metod direktnog pretraživanja



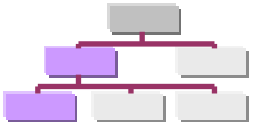
Metod Zlatnog Preseka

Jedan od nedostataka Fibonačijeve metode sastoji se u tome da se odredi broj iteracija *n* koji garantuje da tačka optimuma leži unutar željenog intervala.

Da bi to izbegli stavljamo da je odnos

$$\frac{F_n}{F_{n+2}} \approx c = \frac{3-\sqrt{5}}{2} = 0.38197$$

Tada algoritam postaje



1. Odrediti interval [a,b] (a<b), koji sadrži,tačku x* i specificirati rezoluciju-tačnost aproksimacije $\varepsilon>0$.
2. Izračunati

$$c = \frac{3-\sqrt{5}}{2} \approx 0.38197$$

3. Izračunati prvi interval

$$x_1 = a + c(b-a)$$

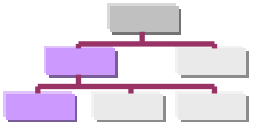
$$x_2 = a + b - x_1$$

4. Dok nije (b-a)< ε

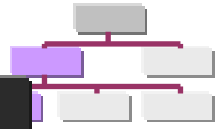
Odrediti $f(x_1)$ i $f(x_2)$, pa napraviti novi set tačaka a i b po principu:

Ako je f(x1) <= f(x2) tada eliminišemo x > x2 i b = x2, x2=x1, x1 = a + c*(b - a)
Ako f(x1) > f(x2) tada eliminišemo x < x1 i a = x1, x1 = x2, x2 = b - c*(b - a)
Ako f(x1) = f(x2) tada biramo novi par tačaka

Metod Zlatnog Preseka

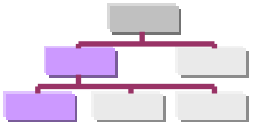


```
def zlatni_presek_metod(a,b,tol):
    # Zlatni presek postupak minimizacije funckcije FUNCjedne
promenljive.
    # Funkcija mora biti unimodalna nad intervalom[a, b].
    # Tol je trazena sirina intervala u kome se nalazi minimum.
    ## Korak 1 - Odredjujemo početni tačku
    # Odredjivanje konstante zlatnog preseka
    c=(3-math.sqrt(5))/2
    # Pocetne tacke
    x1 = a + c*(b-a)
    x2 = a +b-x1
    n=1
```



```
## Korak 2 - Iterativno smanjujemo interval dok ne
zadovoljimo zadatu preciznost
while (b-a)>tol:
    n+=1
    if func(x1)<=func(x2):
        b=x2
        x1=a + c*(b-a)
        x2=a+b-x1
    else:
        a=x1
        x1 = a + c * (b - a)
        x2 = a + b - x1

if func(x1)<func(x2):
    xopt=x1
    fopt=func(x1)
else:
    xopt=x2
    fopt=func(x2)

return xopt,fopt,n
```

Metod Zlatnog Preseka



```
def func(x):
    f=-1*(x**4-5*x**3-2*x**2+24*x)
    return f
a=0
b=3
tol=0.0001

xopt,fopt,n=zlatni_presek_metod(a,b,tol)
print(xopt,fopt,n)

1.398935135585186 -19.801612810579616 23
```

Zašto se zove Zlatni Presek

$$\frac{AC}{BC} = \frac{BC}{AB}$$

Proveriti za domaći

Ako : AC=1 BC=x AB=1-x

Tada je: $\frac{1}{x} = \frac{x}{1-x}$

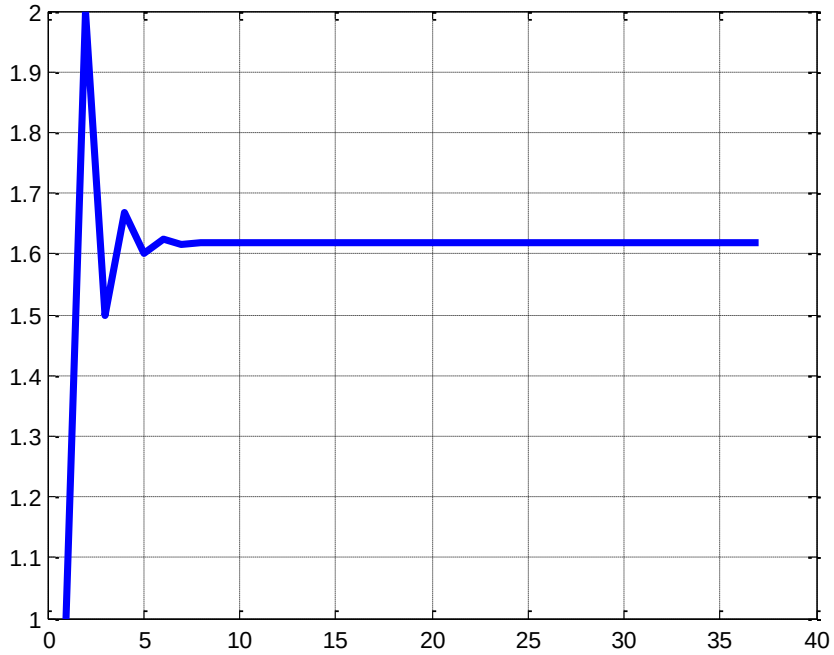
Rešiti kvadratnu jednačinu:

$$x^2 = 1 - x \quad 0 = x^2 + x - 1 \quad x = \frac{-1 \pm \sqrt{5}}{2}$$

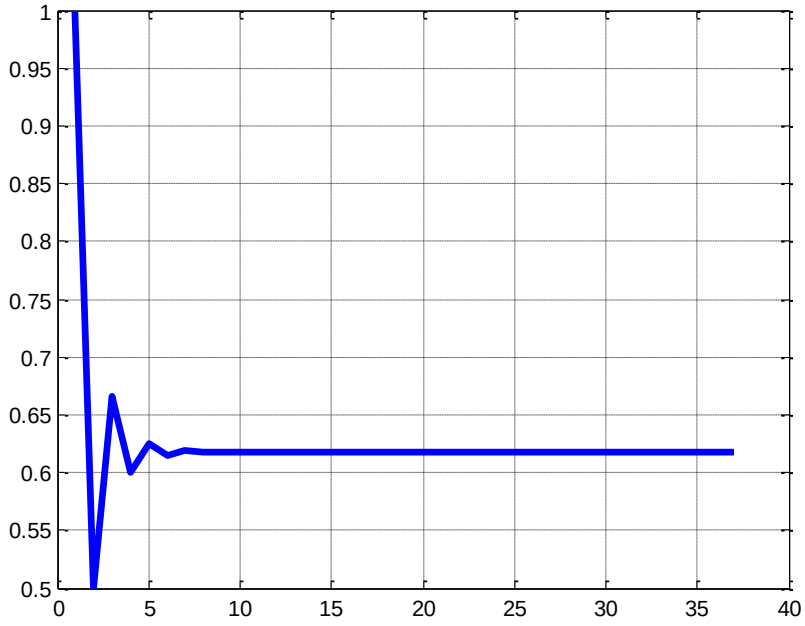
Kako je x negativnu vrednost eliminišemo pa dobijamo

$$x = \frac{\sqrt{5}-1}{2} = 0.618034$$

Metod Zlatnog Preseka



F(i+1)/F(i)



F(i)/F(i+1)

$$\frac{F_n}{F_{n+1}} \approx \frac{\sqrt{5}-1}{2}$$

Simbolički

- Phi - “Fi” - Φ

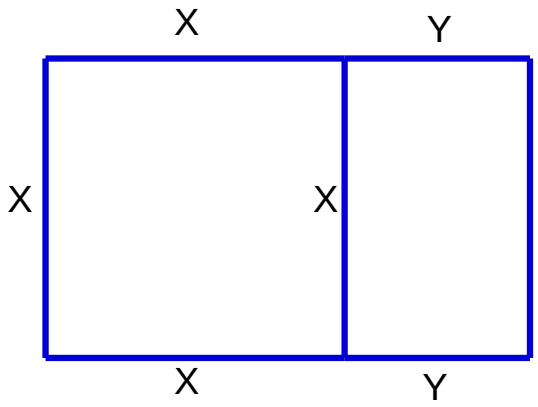
$$\Phi = \frac{\sqrt{5}+1}{2} = 1.6180...$$

- phi - “fi” - ϕ, φ

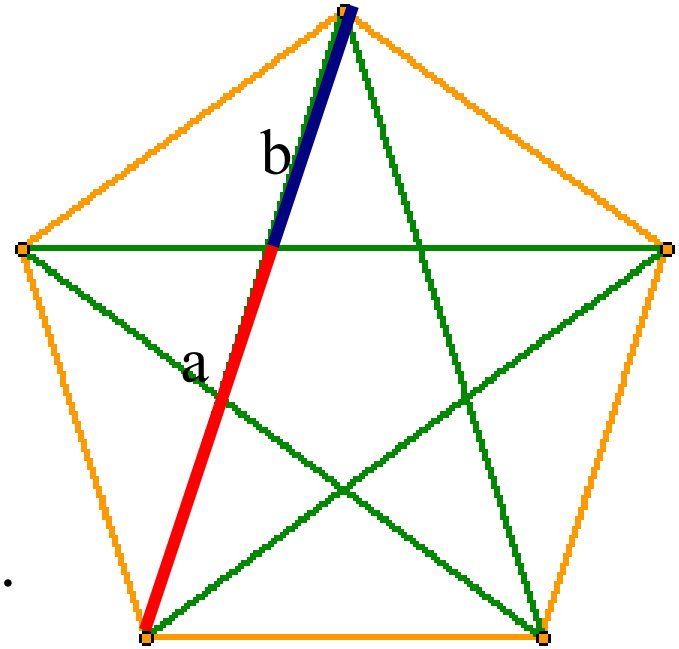
$$\phi = \frac{\sqrt{5}-1}{2} = 0.6180...$$

- Tau - τ

Geometrijski



$$\frac{x}{y} = \frac{x+y}{x}$$



$$\Phi = \frac{a}{b} = 1.6180\ldots$$

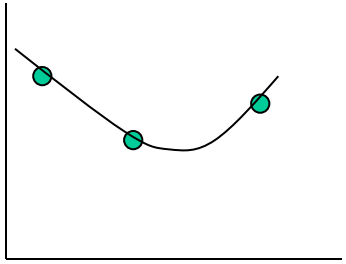
Metode aproksimacije polinomom

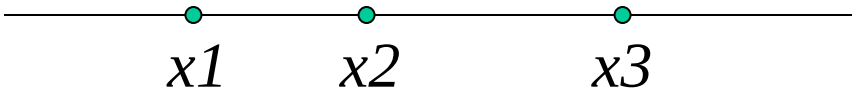
Osnovna ideja ovih metoda

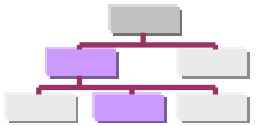
f-ja se aproksimira polinomom *y(x)* na intervalu I koji sadrži optimum, odredi se minimum *min y(x)=xopt*, u okolini *xopt* se formira novi interval (manji od prethodnog) i vrši se nova aproksimacija

Metod Parabole

$$y(x) = a + b\,x + c\,x^2$$

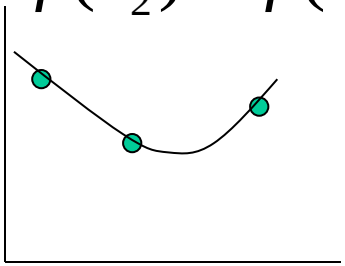






$$y(x) = a + b\,x + c\,x^2$$

- traže se tri $x_1<x_2<x_3$ tačke tako da je $f(x_1) \geq f(x_2) \leq f(x_3)$ tada je i $x_1<x^*<x_3$



- Reši se sistem jednačina po *a*, *b*, *c*

  $a+bx_1+cx_1^2=f(x_1)$

  $a+bx_2+cx_2^2=f(x_2)$

  $a+bx_3+cx_3^2=f(x_3)$

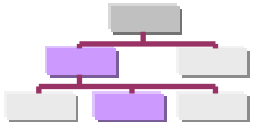
- Uslov minimuma parabole: $y'(x)=0$ da je

$$x_{opt} = -\frac{b}{2c}$$

- sada x_{opt} i dve susedne tačke od x_1, x_2, x_3 formiraju novu trojku i postupak se nastavlja. Uporediti x_{opt} i x_2 manja od njih dve je nova x_2 a tačke levo i desno čine x_1 i x_3.
- postupak se prekida kada je $\left| f(x_{opt}) - y(x_{opt}) \right| \leq \xi$

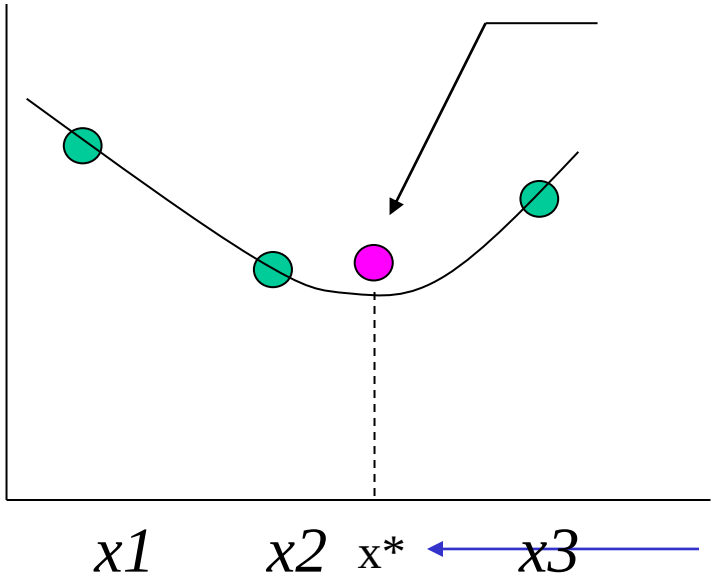
algoritam

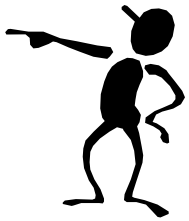
Metod Parabole



$$y(x) = a + b\,x + c\,x^2$$

$$f(x_1) \geq f(x_2) \leq f(x_3)$$





Optimum aproksimacije

$x_2 = x_2$

$x_1 = x_1$

$x_3 = x^*$

$$x^* = \frac{1}{2}\frac{(x_2^2 - x_3^2) f_1 + (x_3^2 - x_1^2) f_2 + (x_1^2 - x_2^2) f_3}{(x_2 - x_3) f_1 + (x_3 - x_1) f_2 + (x_1 - x_2) f_3}$$

```
import numpy as np
import numpy.linalg as lin

def parabola(x1,x3, tol):
    X=np.array([x1, (x1+x3)/2, x3]).transpose()
    pom=np.array([1, 1, 1]).transpose()
    Y=np.array([pom, X, X*X]).transpose()
    F = np.linspace(0, 0, len(X))
    for i in range(0, len(X), 1):
        F[i] = func(X[i])
    abc=lin.solve(Y,F)
    x=-abc[1]/2/abc[2]
    fx=func(x)
    n=0
    while np.abs(np.dot([1, x, x**2],abc)-fx)>tol:
        if (x>X[1]) and  (x<X[2]):
            if (fx<F[1]) and (fx<F[2]):
                X=np.array([X[1], x, X[2]])
                F=np.array([F[1], fx, F[2]])
            elif (fx>F[1]) and (fx<F[2]):
                X = np.array([X[0],X[1], x])
                F = np.array([F[0],F[1], fx])
            else:
                print('Greska')
```

```
    elif (x>X[0]) and  (x<X(2)):
        if (fx < F[0]) and (fx < F[1]):
            X = np.array([X[0], x, X[2]])
            F = np.array([F[0], fx, F[2]])
        elif (fx > F[1]) and (fx < F[0]):
            X = np.array([x, X[1], X[2]])
            F = np.array([fx, F[1], F[2]])
        else:
            print('Greska')
    else:
            print('x lezi van granica')

    pom = np.array([1, 1, 1]).transpose()
    Y = np.array([pom, X, X* X]).transpose()
    F = np.linspace(0, 0, len(X))
    for i in range(0, len(X), 1):
        F[i] = func(X[i])
    abc = lin.solve(Y, F)
    x = -abc[1]/2/abc[2]
    fx = func(x)
    n=n+1
return x,fx,n
```

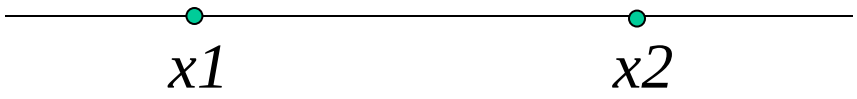
```
def func(x):
    f=-1.*(x**4-5*x**3-2*x**2+24*x)

    return f

tol=0.0001
init_guess=1
[xopt,fopt,n]=parabola(0,2, 0.0001)
print(xopt,fopt,n)
```

```
1.3989682254173135 -19.801612796291295 2
```

Metode aproksimacije polinomom

Kubna Metoda

$$y(x)=a+bx+cx^2+dx^3$$



$$f'(x_1)<0 \quad f'(x_2)>0 \quad x_1<x_2$$

algoritam

- aproksimira *f(x)* polinomom 3. reda
  $y(x)=a+bx+cx^2+dx^3$ $f'(x_1)<0 \quad f'(x_2)>0 \quad x_1<x_2$
- Koeficijenti se mogu odrediti na 2 načina:
  - poznavanjem *f(x)* u 4 tačke ili
  - poznavanjem *f(x)* i *f'(x)* u 2 tačke.

    $a+bx_i+cx_i^2+dx_i^3=f(x_i)$, i=1,2

    $b+2cx_i+3dx_i^2=f'(x_i)$, i=1,2
- Kod rešavanja *y'(x)* =0 dobijaju se dva rešenja, a uzima se ono koje leži u intervalu $[x_1,x_2]$ i ima manju vrednost (za minimum)
  Ako je $f'(x_{opt})<0$, $x_1=x_{opt}$
  inače $x_2=x_{opt}$

- postupak se prekida kada je $\left|f(x_{opt})-y(x_{opt})\right|\le\xi$

```
import numpy as np
import numpy.linalg as lin
import math
def kubna(x1,x2, tol):
    X=np.array([x1, x2])
    Y=np.array([[1,x1,x1**2,x1**3],[1,x2,x2**2,x2**3],[0, 1,
2*x1,3*x1**2],[0, 1, 2*x2,3*x2**2]])
    F = np.array([func(x1),func(x2), dfunc(x1), dfunc(x2)  ])
    abcd=lin.solve(Y,F)
    b=abcd[1]
    c = abcd[2]
    d = abcd[3]
    D=math.sqrt(4*c*c-12*b*d)
    xa=(-2*c-D)/6/d
    xb=(-2*c+D)/6/d
    if func(xa)<func(xb):
        x=xa
    else:
        x=xb
    fx=func(x)
    n=0
```

```
while np.abs(np.dot([1, x, x**2,x**3],abcd)-fx)>tol:
    if func(xa) <func(xb):
        x2=x
    else:
        x1=x
    Y = np.array([[1, x1, x1 ** 2, x1 ** 3], [1, x2, x2 ** 2, x2 ** 3],
[0, 1, 2 * x1, 3 * x1 ** 2],
                  [0, 1, 2 * x2, 3 * x2 ** 2]])

    F = np.array([func(x1), func(x2), dfunc(x1), dfunc(x2)])
    abcd = lin.solve(Y, F)
    b = abcd[1]
    c = abcd[2]
    d = abcd[3]
    D = math.sqrt(4 * c * c - 12 * b * d)
    xa = (-2 * c - D) / 6 / d
    xb = (-2 * c + D) / 6 / d
    if func(xa) < func(xb):
        x = xa
    else:
        x = xb
    n+=1
    fx=func(x)
return x,fx,n
```

```
def func(x):
    f=-1.*(x**4-5*x**3-2*x**2+24*x)

    return f
def dfunc(x):
    f=-1*(4*x**3-15*x**2-4*x+24)
    return f
tol=0.0001
init_guess=1
[xopt,fopt,n]=kubna(0,2, 0.00001)
print(xopt,fopt,n)
```

```
1.3971493814365605 -19.801577064478614 30
```

Osobine kubne metode

- optimum uvek leži u [*x1,x2*]
- brža je od metode parabole, ali zahteva više računarskih operacija (stepen konvergencije je superlinearan)
- Minimum *y(x)* na intervalu [*x1,x2*] se može izračunati direktno

$$x^* = x_2 - \frac{f_2' + w - z}{f_2' - f_1' + 2w}(x_2 - x_1)$$

$$z = 3\frac{f_1 - f_2}{x_2 - x_1} + f_1' + f_2'$$

$$w = \sqrt{z^2 - f_1' \cdot f_2'}$$