

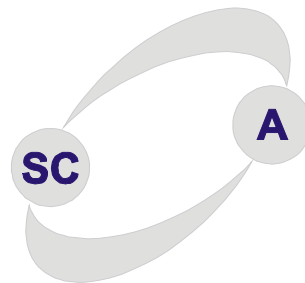
Predavanje

Predmet

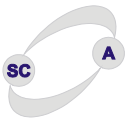
Metodi optimizacije

Tema

Statička optimizacija, numeričke metode jednodimenzione optimizacije



2019 godina



Statička optimizacija, numeričke metode jednodimenzione optimizacije

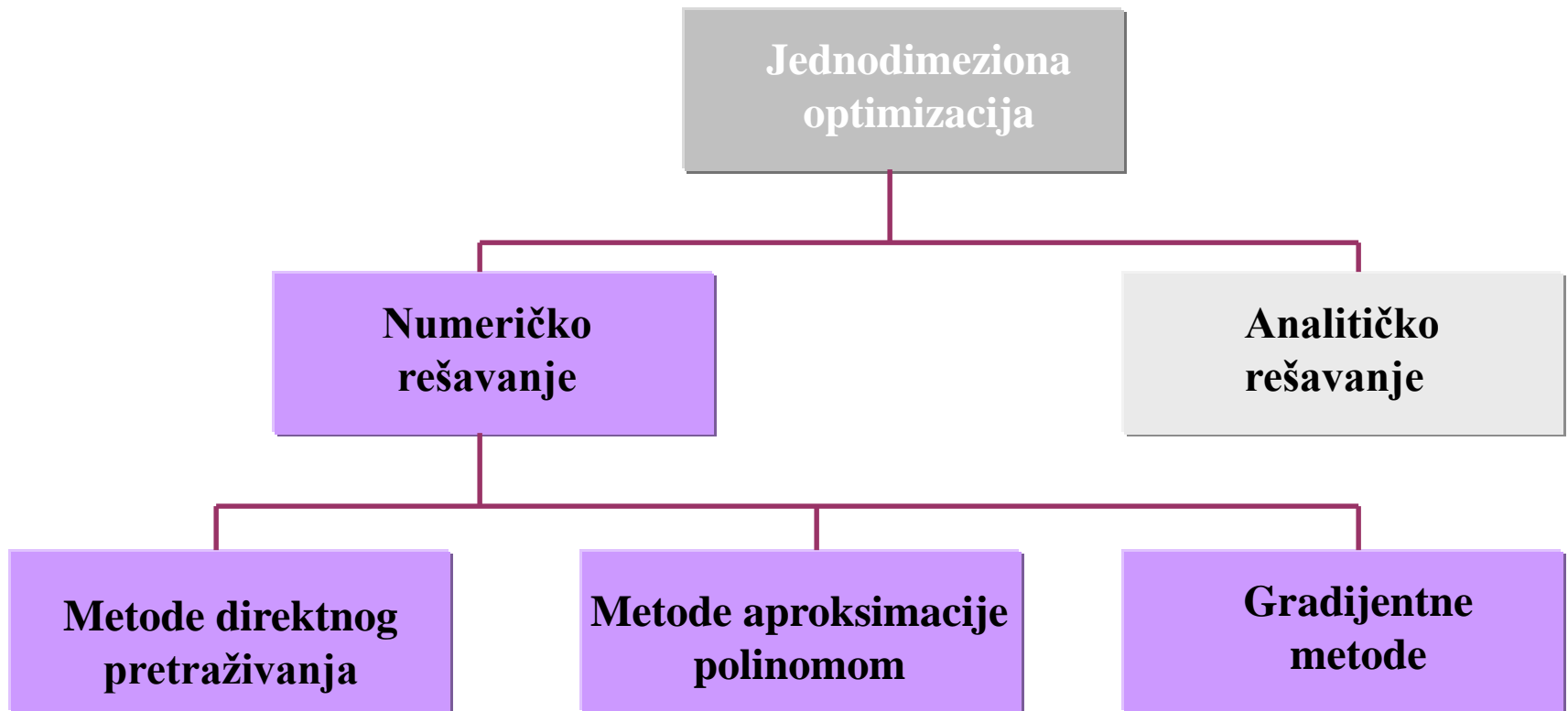
Da li su jednodimenzioni problemi laki? **(Ponekad)**

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

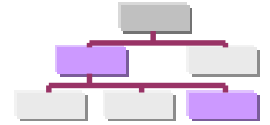
ali,

1. Postoji određen broj važnih jednodimenzionih problema.
2. Predstavljaju osnovu višedimenzione 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*

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

The graph shows a coordinate system with a horizontal x-axis and a vertical y-axis. A solid red curve, labeled $f(x)$, represents a function. It has a minimum point marked with a vertical dashed line extending to the x-axis at x^* . A dashed blue line segment, labeled $g(x)$, represents a subgradient. It is shown for two points, x_0 and x_1 , both of which are greater than x^* . Vertical dashed lines connect x_0 and x_1 to the x-axis. The subgradient $g(x)$ is shown as a dashed blue line segment that is tangent to the curve $f(x)$ at x_0 and x_1 .

$$g'(x_1) = 0$$

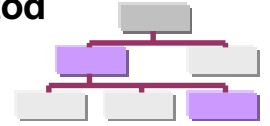
$$f'(x_0) + f''(x_0)(x_1 - x_0) = 0$$

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



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

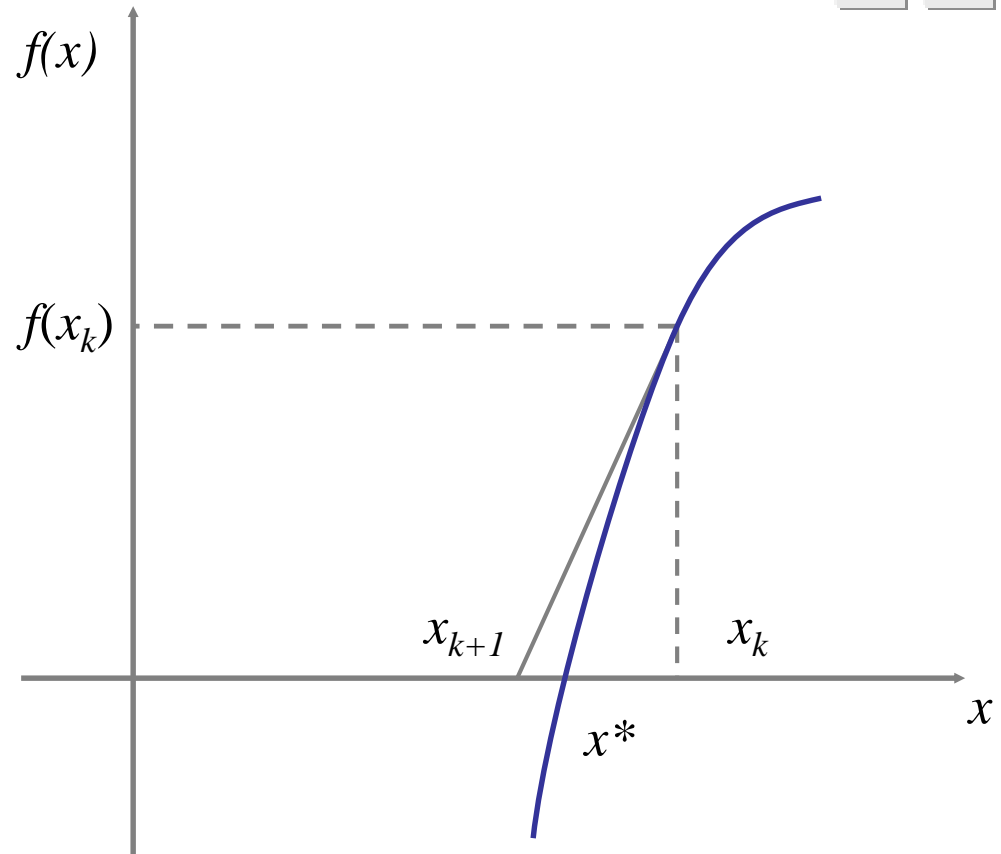
Newton-Raphson Metod

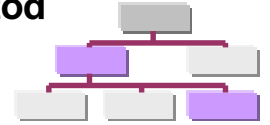


$$f'(x_k) = \frac{f(x_k) - f(x_{k+1})}{x_k - x_{k+1}}$$

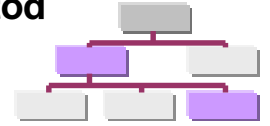
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

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





```
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
```



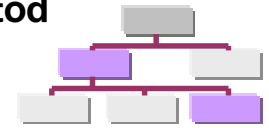
- Početno pogađanje $x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$

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

- Kriterijum zaustavljanja $\varepsilon_{n+1} = \left| \frac{x_{n+1} - x_n}{x_{n+1}} \right|$

Da li je uvek ovo kriterijum zaustavljanja





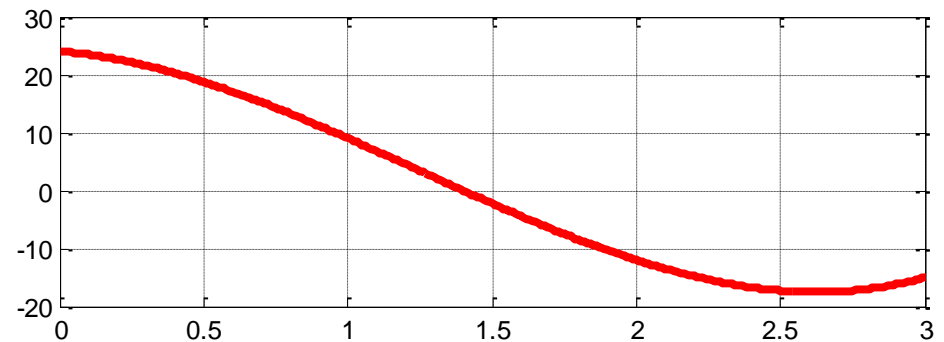
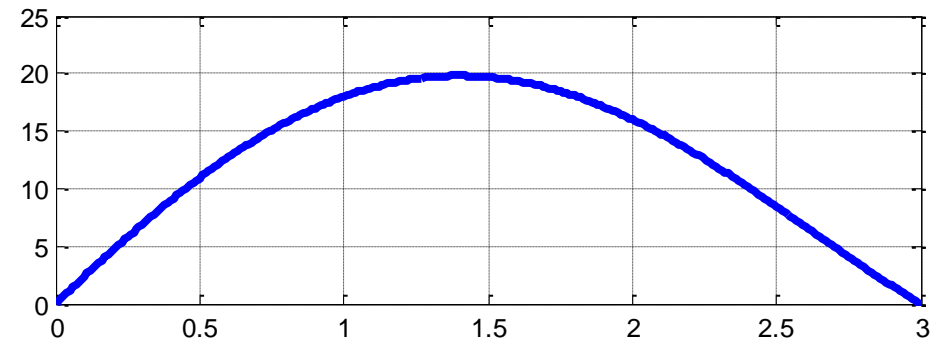
Primer

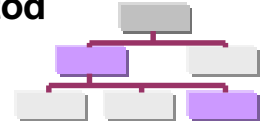
$$f(x) = x^4 - 5x^3 - 2x^2 + 24x$$

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

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

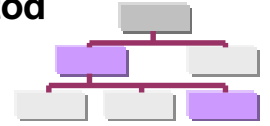
$$x = [0, 3]$$



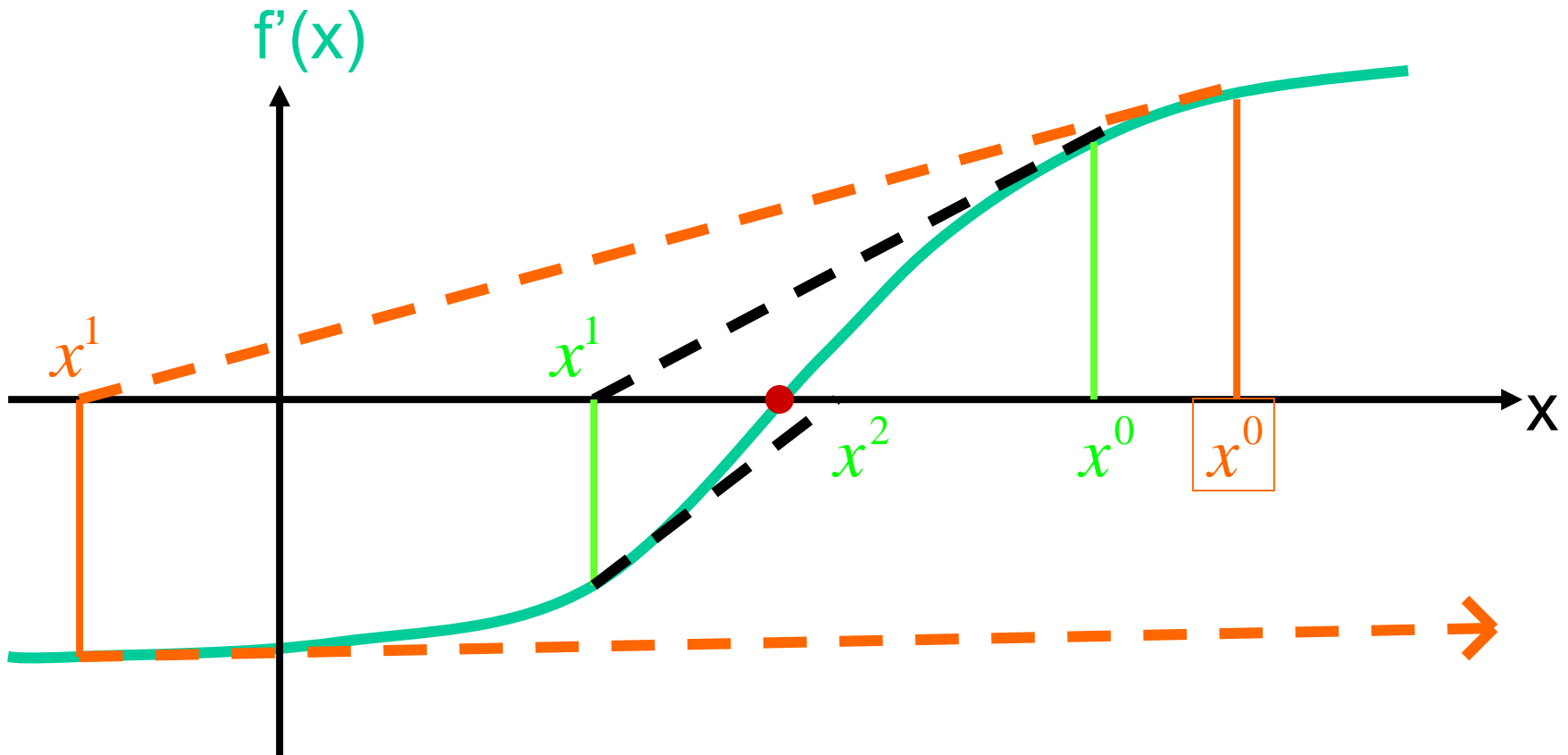


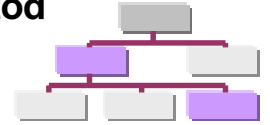
Primer

```
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
```

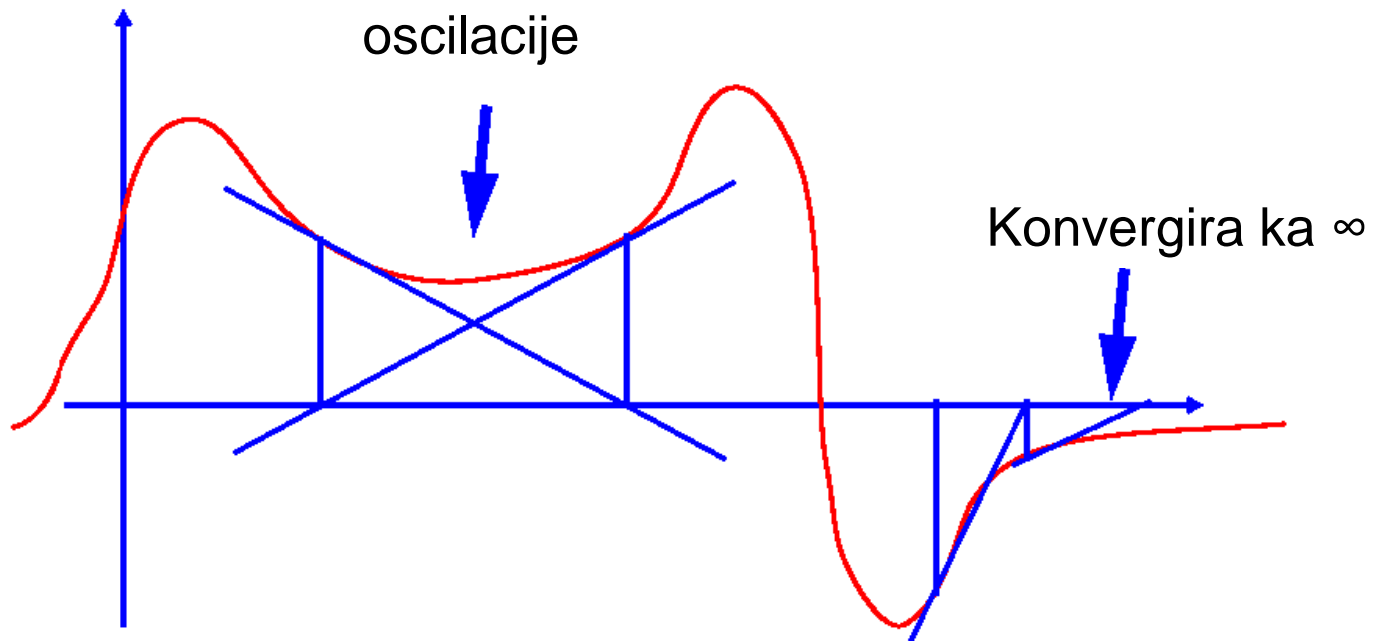


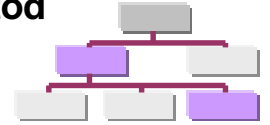
Konvergencija zavisi od dobrog početnog pogađanja





Konvergencija zavisi od dobrog početnog pogađanja





domaći

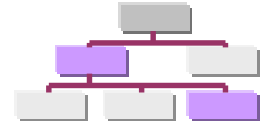
$$f(x)' = (x-1)^3 = 0$$

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

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

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

Gradijentne metode



Metod Sečice

- Ako se drugi izvod zameni konačnom razlikom,

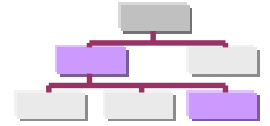
$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}$$

Newton-va metoda postaje

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

- tačke x_0, x_1 su proizvoljne

Metod Sečice

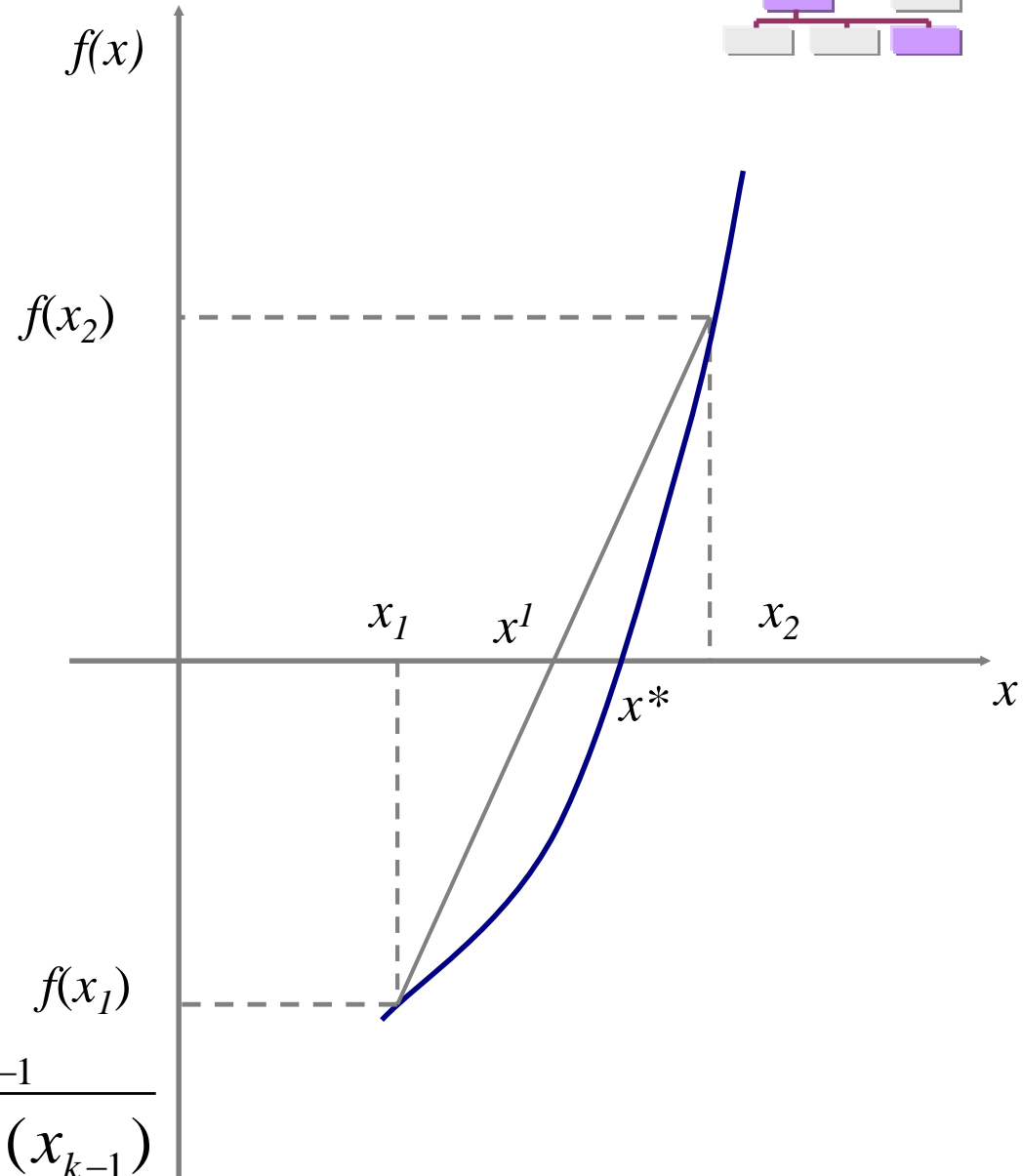


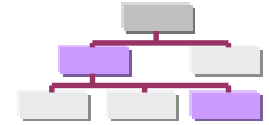
$$\frac{x^1 - x_1}{-f(x_1)} = \frac{x_2 - x_1}{f(x_2) - f(x_1)}$$

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

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

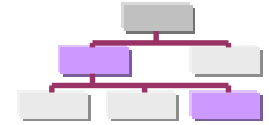
$$x_{k+1} = x_k - f'(x_k) \frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})}$$



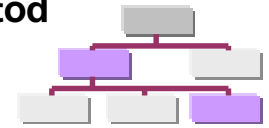


```
def secica(x1,x0,tol):
    x_pre=x0
    x_ppre=math.inf
    x_novo=x1
    iter=0
    while(abs(x_novo-x_pre)>tol):
        iter+=1
        x_ppre = x_pre
        x_pre = x_novo
        x_novo=x_pre-dfunc(x_pre)*(x_pre-x_ppre)/(dfunc(x_pre)-
dfunc(x_ppre))

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

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



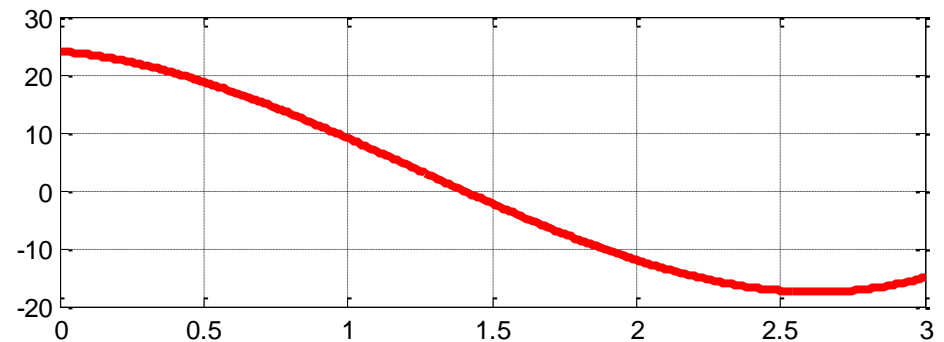
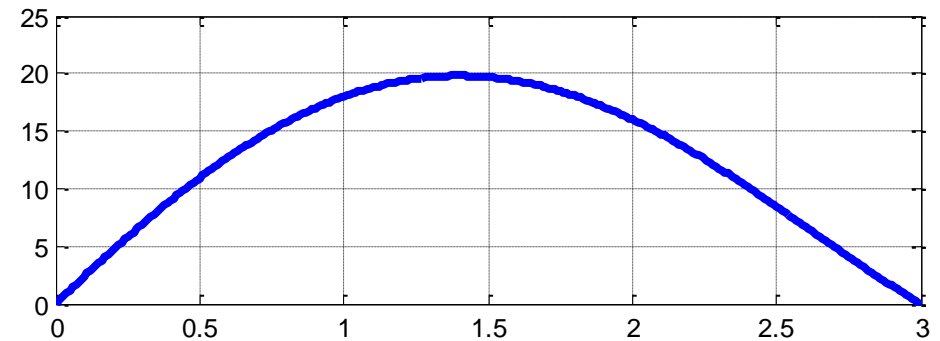
Primer

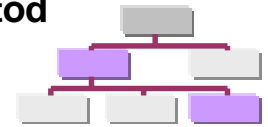
$$f(x) = x^4 - 5x^3 - 2x^2 + 24x$$

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

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

$$x = [0, 3]$$





```
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
```

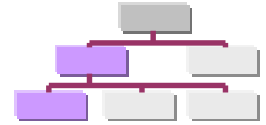
```
init_guess1=0
```

```
init_guess2=3
```

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

```
1.3989324663002147 19.80161281065917 5
```

Metode direktnog pretraživanja

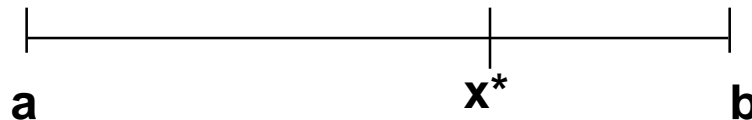


**Metod direktnog pretraživanja su u osnovi metode
jednodimenzione optimizacije**

Smatraju ih “kičmom” nelinearnih optimizacionih algoritama

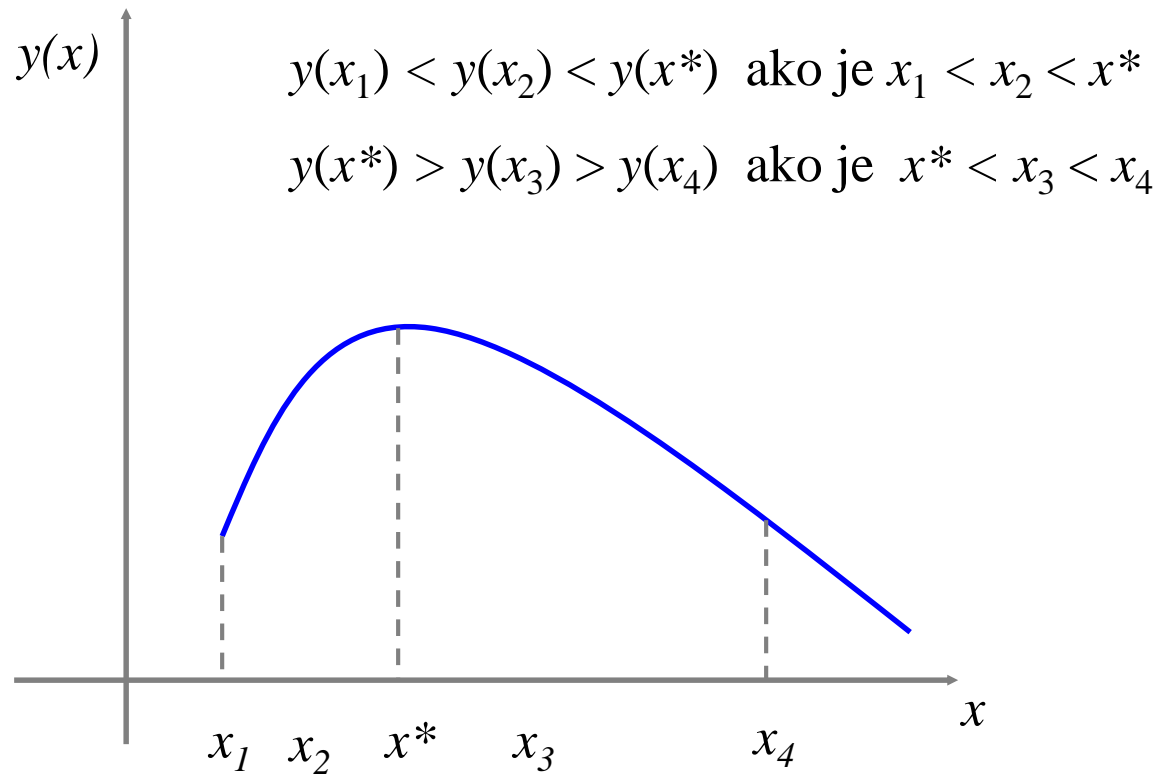
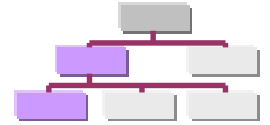
Svode se na pretraživanje zatvorenog intervala

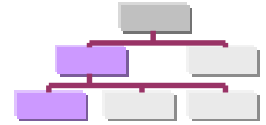
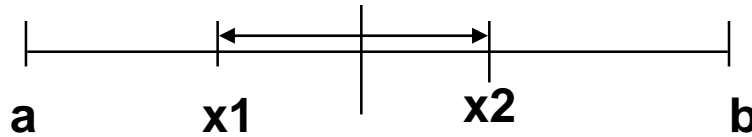
Često pretpostavljamo da je funkcija unimodalna



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

unimodalnost





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

0) pretpostaviti interval $[a, b]$

1) naći po nekoj formuli (npr $x_1 = a + (b-a)/2 - \varepsilon/2$ i $x_2 = a + (b-a)/2 + \varepsilon/2$ gde je ε rezolucija).

2) porediti $f(x_1)$ i $f(x_2)$

3) Ako je $f(x_1) < f(x_2)$ tada eliminišemo $x > x_2$ i $b = x_2$

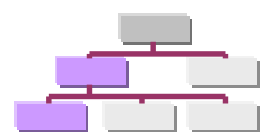
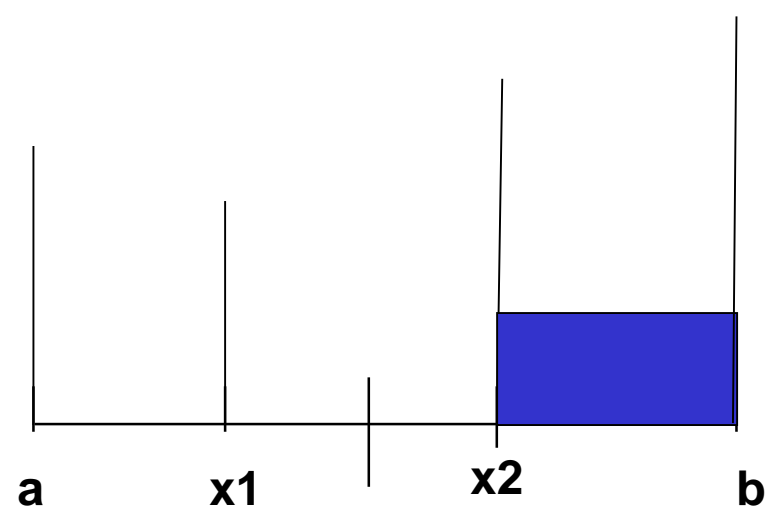
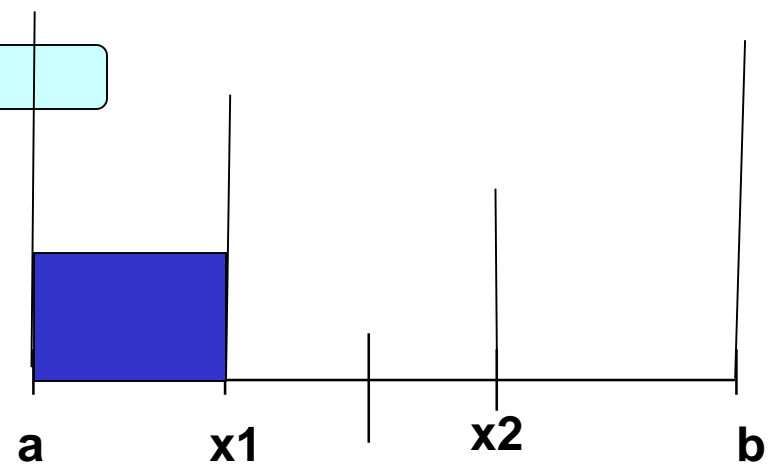
Ako $f(x_1) > f(x_2)$ tada eliminišemo $x < x_1$ i $a = x_1$

Ako $f(x_1) = f(x_2)$ tada biramo novi par tačaka

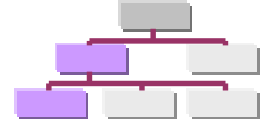
4) Nastaviti dok interval ne bude $< 2 \varepsilon$



Koju oblast eliminišemo



Metod direktnog pretraživanja

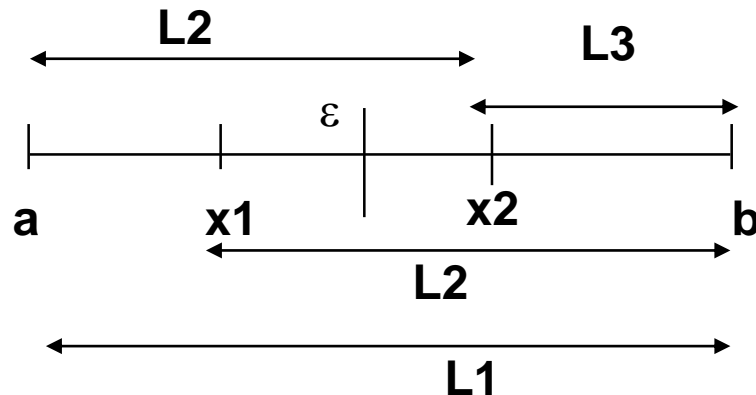


Fibonačijev Metod

Fibonačijevi brojevi:

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

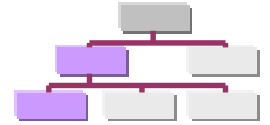
$$F_n = F_{n-1} + F_{n-2}$$



$$L_1 = L_2 + L_3$$

Može se pokazati

$$L_n = (L_1 + F_{n-2} \epsilon) / F_n$$



1200 te...

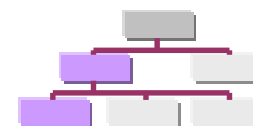
Fibonači je postavio pitanje...



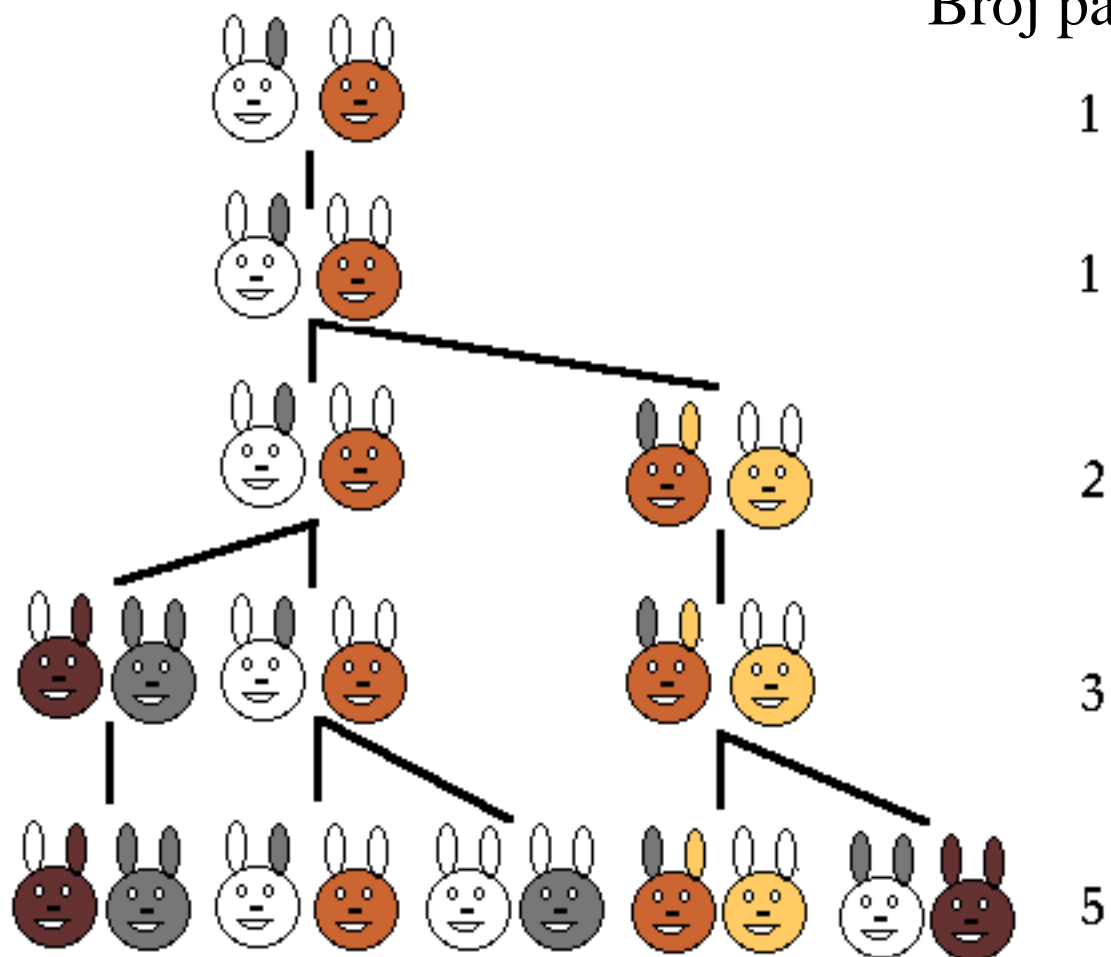
Leonardo Pisano
(1170-1250)

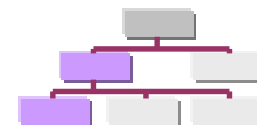
Pretpostavimo da se tek rođeni par, mužjak i ženka, zečeva stavi u polje. Zečevi su u stanju da se razmnožavaju posle mesec dana i na kraju drugog meseca ženka može da dobije novi par zečeva. Pretpostavimo da zečevi ne umiru i da ženka uvek daje novi par (muško-žensko) svaki mesec od drugog meseca. Koliko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

Fibonačijev metod



Broj parova





1. Odrediti interval $L_0 [a, b]$ ($a < b$), koji sadrži tačku x^* i specificirati rezoluciju-tačnost aproksimacije $\varepsilon > 0$. 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 $f(x_1) \leq f(x_2)$ tada eliminišemo $x > x_2$ i $b = x_2$, $x_2 = x_1$, $x_1 = a + b - x_1$.

Ako $f(x_1) > f(x_2)$ tada eliminišemo $x < x_1$ i $a = x_1$, $x_1 = x_2$, $x_2 = a + b - x_2$

Ako $f(x_1) = f(x_2)$ tada biramo novi par tačaka

```
def fibonaci_metod(a,b,tol):  
    #Fibonacci - jev postupak minimizacije funkcije funkcije jedne  
    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 ((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  $(b - a) < \text{tol}$ 
```

```
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
```

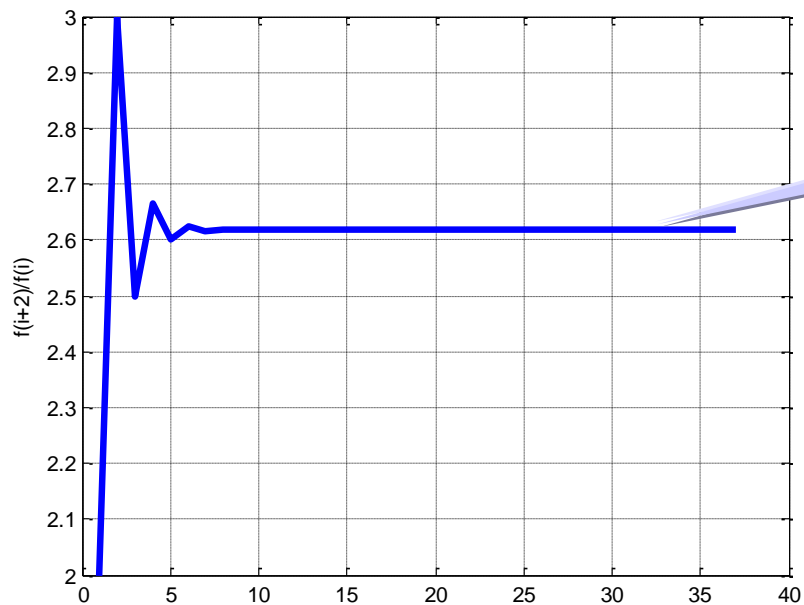
```
a=0
```

```
b=3
```

```
tol=0.0001
```

```
xopt,fopt,n=fibonacci_metod(a,b,tol)
```

```
1.398938923395339 -19.801612810191763 24
```



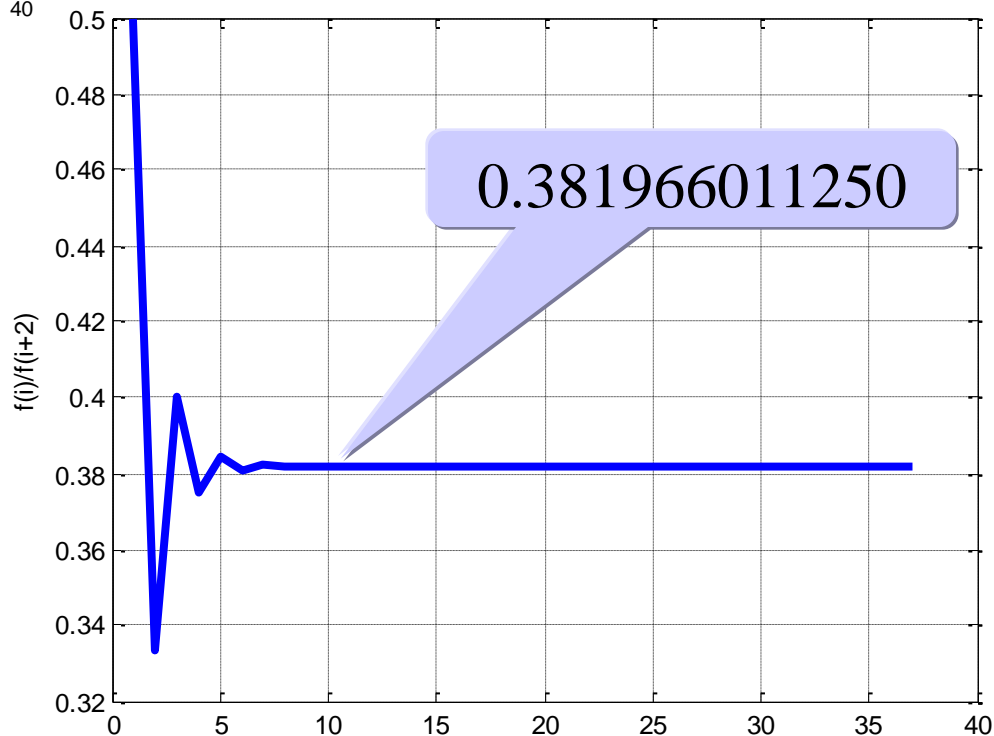
2.618033988749

$F(i)/F(i+2)$

$F(i+2)/F(i)$

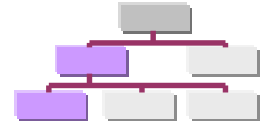
Posle 15 brojeva podudaraju
se na 5 decimalnih mesta

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



0.381966011250

Metod direktnog pretraživanja



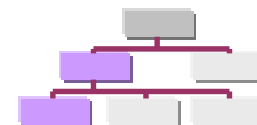
Metod Zlatnog Preseka

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

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

4. Dok nije $(b-a) < \varepsilon$

$$x_2 = a + b - x_1$$

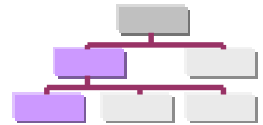
Odrediti $f(x_1)$ i $f(x_2)$, pa napraviti novi set tačaka a i b po principu:

Ako je $f(x_1) \leq f(x_2)$ tada eliminišemo $x > x_2$ i $b = x_2$, $x_2 = x_1$, $x_1 = a + c^*(b - a)$

Ako $f(x_1) > f(x_2)$ tada eliminišemo $x < x_1$ i $a = x_1$, $x_1 = x_2$, $x_2 = b - c^*(b - a)$

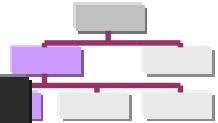
Ako $f(x_1) = f(x_2)$ tada biramo novi par tačaka

Metod Zlatnog Preseka



```
def zlatni_presek_metod(a,b,tol):  
    # Zlatni presek postupak minimizacije funkcije 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
```

Metod Zlatnog Preseka



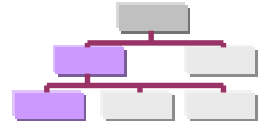
```
## Korak 2 - Iterativno smanjujemo interval dok ne  
zadovoljimo zadatu preciznost
```

```
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
```

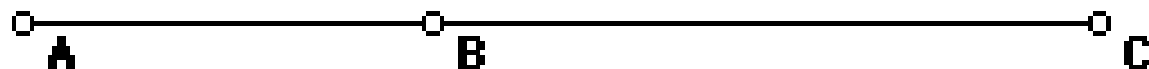
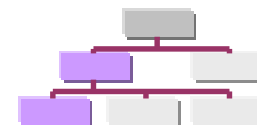
Metod Zlatnog Preseka



```
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 3
```

Zašto se zove Zlatni Presek

Metod Zlatnog Preseka



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

Proveriti za domaći

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

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

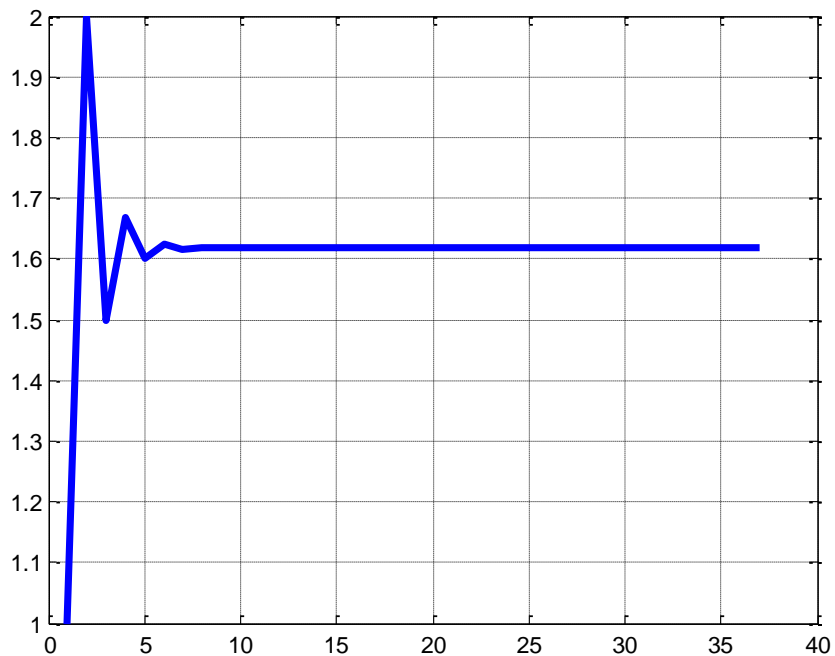
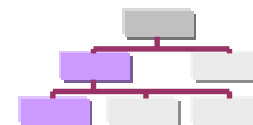
Rešiti kvadratnu jednačinu:

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

Kako je x negativnu vrednost eliminišemo pa dobijamo

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

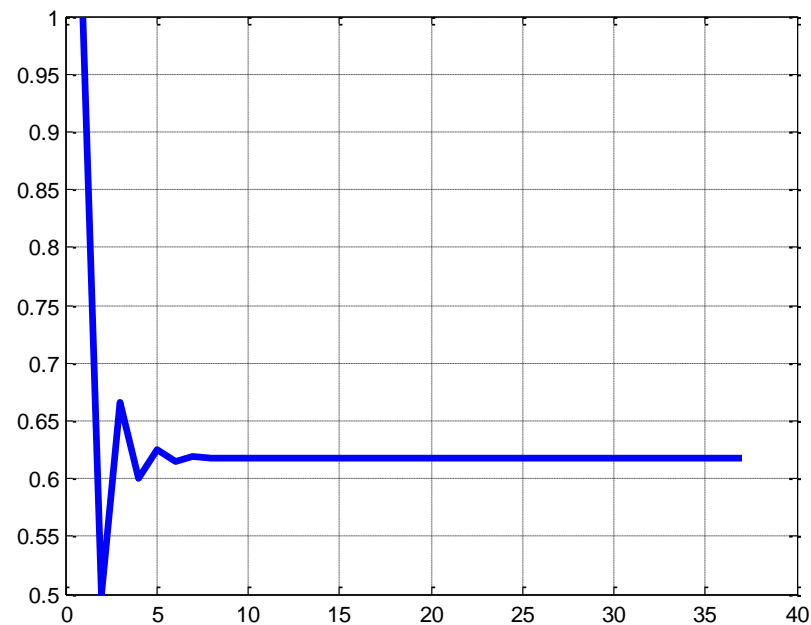
Metod Zlatnog Preseka



$F(i+1)/F(i)$

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

$F(i)/F(i+1)$



Simbolički

- Phi - “Fì” - Φ

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

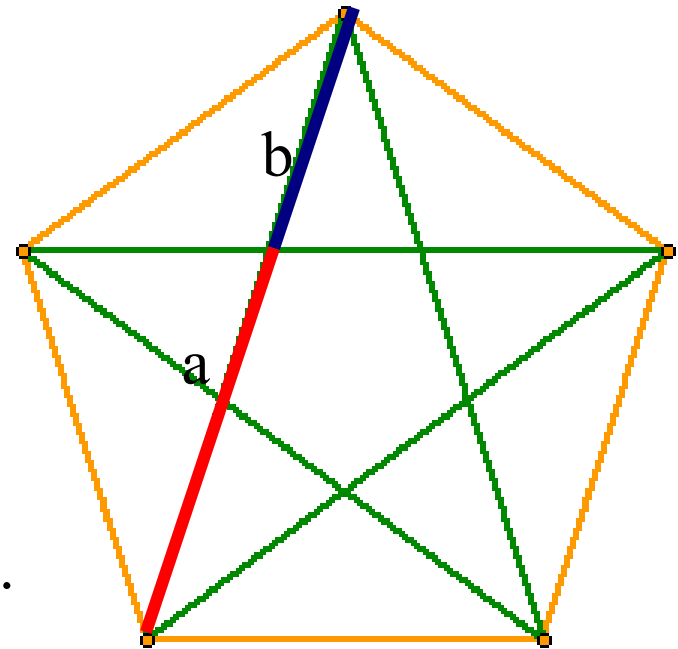
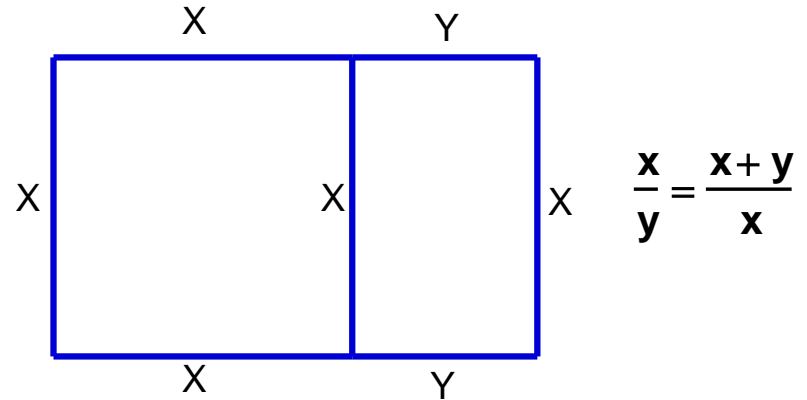
- phi - “fì” - ϕ , φ

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

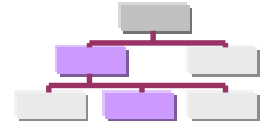
- Tau - τ

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

Geometrijski



Metode aproksimacije polinomom

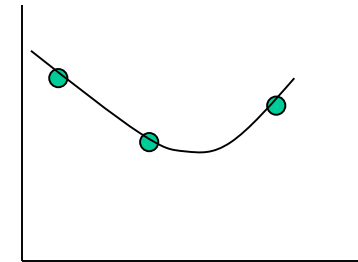
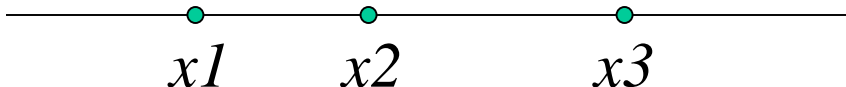


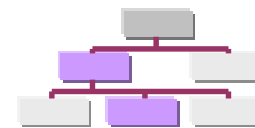
Osnovna ideja ovih metoda

f -ja se aproksimira polinomom $y(x)$ na intervalu I koji sadrži optimum, odredi se minimum $\min y(x)=x_{opt}$, u okolini x_{opt} 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

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

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

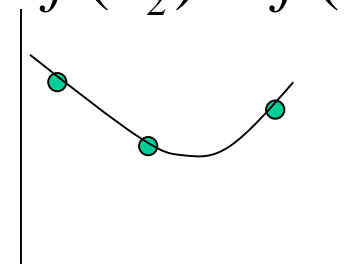
$$a + bx_3 + cx_3^2 = f(x_3)$$

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

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

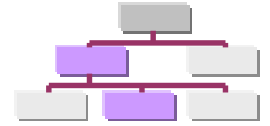
- sada x_{opt} i dve susedne tačke od x_1, x_2, x_3 formiraju novu trojku i postupak se nastavlja. Uporediti x_{opt} i x_2 manja od njih dve je nova x_2 a tačke levo i desno čine x_1 i x_3 .

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



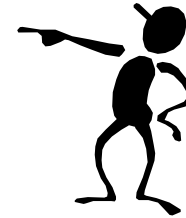
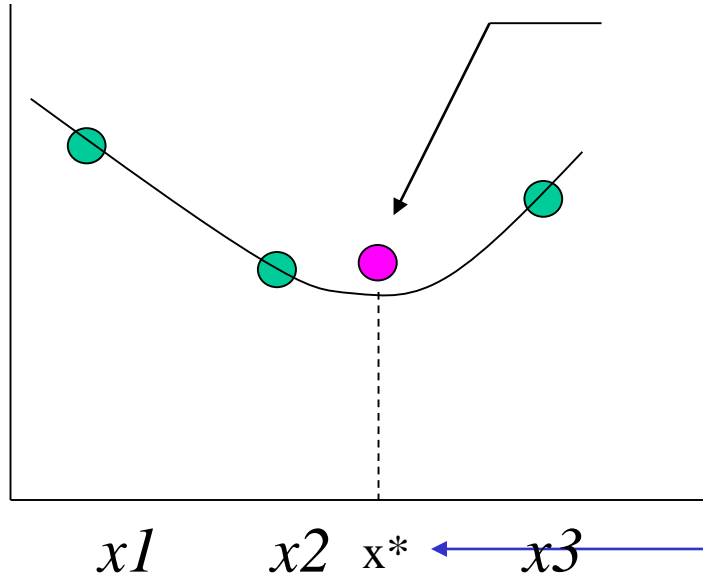
algoritam

Metod Parabole



$$y(\mathbf{x}) = \mathbf{a} + \mathbf{b} \mathbf{x} + \mathbf{c} \mathbf{x}^2$$

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



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

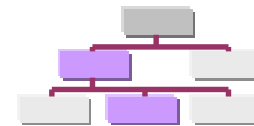
```

```
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)
```

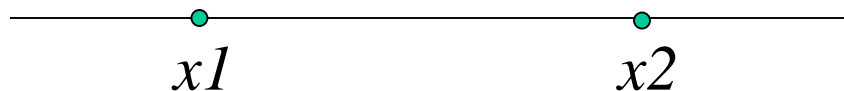
```
1.3989682254173135 -19.801612796291295 2
```

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

$$y(x)=a+bx+cx^2+dx^3 \quad f'(x_1) < 0 \quad f'(x_2) > 0 \quad x_1 < x_2$$

- Koeficijenti se mogu odrediti na 2 načina:

- poznavanjem $f(x)$ u 4 tačke ili
- poznavanjem $f(x)$ i $f'(x)$ u 2 tačke.

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

$$b+2cx_i+3dx_i^2=f'(x_i), \quad i=1,2$$

- 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 $|f(x_{opt}) - y(x_{opt})| \leq \xi$

```

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 $[x_1, x_2]$
- brža je od metode parabole, ali zahteva više računarskih operacija (stepen konvergencije je superlinearan)
- Minimum $y(x)$ na intervalu $[x_1, x_2]$ se može izračunati direktno

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

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

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