

# NA19 Sustavi nelinearnih jednadžbi

Ivan Slapničar

17.prosinca 2018.

## 1 Sustavi nelinearnih jednadžbi

**Problem.** Nađimo rješenje  $\xi = (\xi_1, \xi_2, \dots, \xi_n)$  sustava od  $n$  jednadžbi

$$\begin{aligned}f_1(x) &= 0, \\f_2(x) &= 0, \\&\vdots \\f_n(x) &= 0,\end{aligned}$$

i  $n$  nepoznanica  $x = (x_1, x_2, \dots, x_n)$ . Uz oznaku  $f = (f_1, f_2, \dots, f_n)^T$ , ovaj sustav možemo zapisati kao

$$f(x) = 0.$$

Opisat ćemo *Newtonovu metodu* i tri *kvazi-Newtonove metode*:

2. *Broydenovu metodu*,
3. *Davidon-Fletcher-Powell metodu* i
4. *Broyden-Fletcher-Goldfarb-Schano metodu*.

Sve metode, uz zadanu početnu aproksimaciju  $x^{(0)}$ , generiraju niz točaka  $x^{(n)}$  koji, uz određene uvjete, konvergira prema rješenju  $\xi$ .

**Napomena.** Opisi metoda se nalaze u knjizi [Numerička matematika, poglavlje 4.4](#). Brojevi primjera se odnose na isto poglavlje.

### 1.1 Newtonova metoda

**Jacobijan** ili **Jacobijeva matrica** funkcija  $f$  u točki  $x$  je matrica prvih parcijalnih derivacija

$$J(f, x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \dots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \dots & \frac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \dots & \frac{\partial f_n(x)}{\partial x_n} \end{bmatrix}.$$

Za zadanu početnu aproksimaciju  $x^{(0)}$ , računamo niz točaka

$$x^{(k+1)} = x^{(k)} - s^{(k)}, \quad k = 0, 1, 2, \dots,$$

gdje je  $s^{(k)}$  rješenje sustava

$$J(f, x^{(k)}) \cdot s = f(x^{(k)}).$$

Za računanje Jacobijana koristimo paket `ForwardDiff.jl`. Za crtanje funkcija koristimo paket `PyPlot.jl`.

```
In [1]: using ForwardDiff
        using PyPlot
        using LinearAlgebra
```

```
In [2]: function myNewton(f::Function, J::Function, x::Array{T}, ε::T) where T
        iter=0
        s=ones(T, size(x))
        ζ=x
        while norm(s)>ε && iter<100
            s=J(x)\f(x)
            ζ=x-s
            iter+=1
            x=ζ
        end
        ζ, iter
    end
```

```
Out[2]: myNewton (generic function with 1 method)
```

### 1.1.1 Zadatak 4.4 (a)

Rješenja sustava

$$\begin{aligned} 2(x_1 + x_2)^2 + (x_1 - x_2)^2 - 8 &= 0 \\ 5x_1^2 + (x_2 - 3)^2 - 9 &= 0 \end{aligned}$$

$$\text{su } x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ i } x \approx \begin{bmatrix} -1.18 \\ 1.59 \end{bmatrix}.$$

```
In [3]: # Zadatak 4.4 (a) (Dennis, Schnabel (1996))
fa(x)=[2(x[1]+x[2])^2+(x[1]-x[2])^2-8,5*x[1]^2+(x[2]-3)^2-9]
```

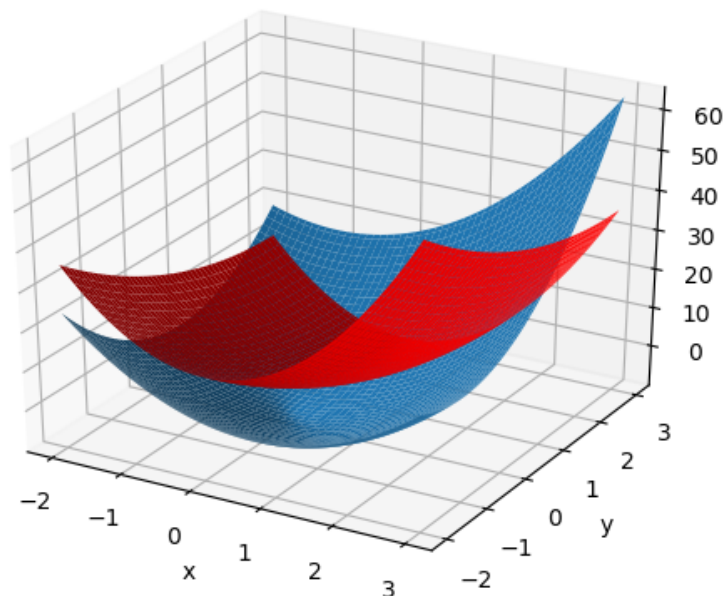
```
Out[3]: fa (generic function with 1 method)
```

```
In [4]: fa([1.0,2])
```

```
Out[4]: 2-element Array{Float64,1}:
 11.0
 -3.0
```

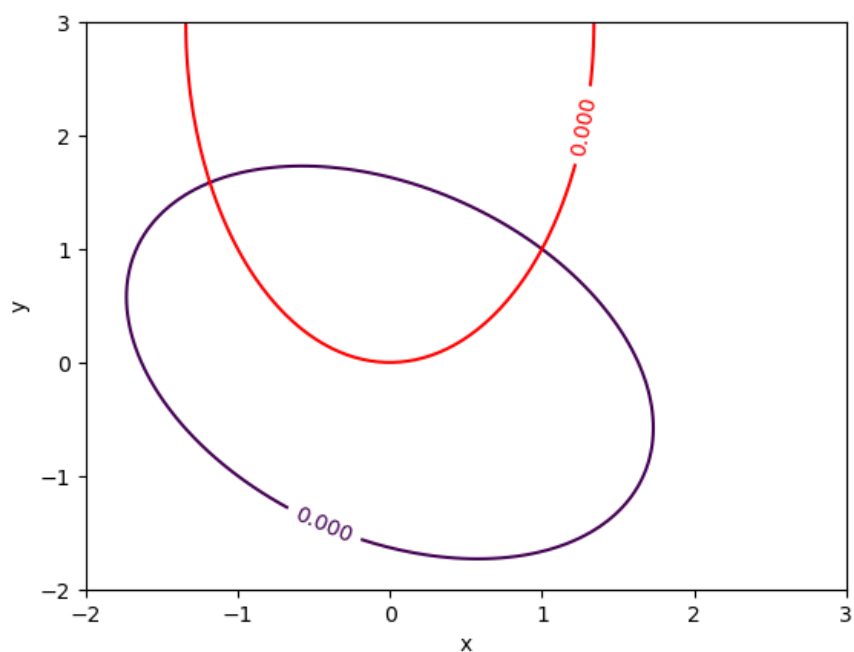
Nacrtajmo funkcije i konture kako bi mogli približno locirati nul-točke:

```
In [5]: # Broj točaka
m=100
X=range(-2,stop=3,length=m)
Y=range(-2,stop=3,length=m)
XY=collect(Iterators.product(X,Y))
# Prva aplikata
Z1=Matrix(getindex.(map(fa,XY),1)')
surf(X,Y,Z1)
# Druga aplikata
Z2=Matrix(getindex.(map(fa,XY),2)')
surf(X,Y,Z2,color="red")
xlabel("x")
ylabel("y")
```



Out[5]: PyObject <matplotlib.text.Text object at 0x7fb8308080f0>

```
In [6]: # Konture (Nivo-krivulje)
C1=contour(X,Y,Z1,levels=[0])
C2=contour(X,Y,Z2,levels=[0],colors="red")
clabel(C1,inline=1, fontsize=10)
clabel(C2,inline=1, fontsize=10)
xlabel("x")
ylabel("y")
```



Out[6]: PyObject <matplotlib.text.Text object at 0x7fb82fa504a8>

```
In [7]: XY
```

```
Out[7]: 100×100 Array{Tuple{Float64,Float64},2}:
 (-2.0, -2.0)      (-2.0, -1.94949)      ...      (-2.0, 3.0)
 (-1.94949, -2.0)  (-1.94949, -1.94949)    (-1.94949, 3.0)
 (-1.89899, -2.0)  (-1.89899, -1.94949)    (-1.89899, 3.0)
 (-1.84848, -2.0)  (-1.84848, -1.94949)    (-1.84848, 3.0)
 (-1.79798, -2.0)  (-1.79798, -1.94949)    (-1.79798, 3.0)
 (-1.74747, -2.0)  (-1.74747, -1.94949)    ...      (-1.74747, 3.0)
```

(-1.69697, -2.0)	(-1.69697, -1.94949)	(-1.69697, 3.0)
(-1.64646, -2.0)	(-1.64646, -1.94949)	(-1.64646, 3.0)
(-1.59596, -2.0)	(-1.59596, -1.94949)	(-1.59596, 3.0)
(-1.54545, -2.0)	(-1.54545, -1.94949)	(-1.54545, 3.0)
(-1.49495, -2.0)	(-1.49495, -1.94949)	... (-1.49495, 3.0)
(-1.44444, -2.0)	(-1.44444, -1.94949)	(-1.44444, 3.0)
(-1.39394, -2.0)	(-1.39394, -1.94949)	(-1.39394, 3.0)
⋮		⋮
(2.44444, -2.0)	(2.44444, -1.94949)	(2.44444, 3.0)
(2.49495, -2.0)	(2.49495, -1.94949)	(2.49495, 3.0)
(2.54545, -2.0)	(2.54545, -1.94949)	... (2.54545, 3.0)
(2.59596, -2.0)	(2.59596, -1.94949)	(2.59596, 3.0)
(2.64646, -2.0)	(2.64646, -1.94949)	(2.64646, 3.0)
(2.69697, -2.0)	(2.69697, -1.94949)	(2.69697, 3.0)
(2.74747, -2.0)	(2.74747, -1.94949)	(2.74747, 3.0)
(2.79798, -2.0)	(2.79798, -1.94949)	... (2.79798, 3.0)
(2.84848, -2.0)	(2.84848, -1.94949)	(2.84848, 3.0)
(2.89899, -2.0)	(2.89899, -1.94949)	(2.89899, 3.0)
(2.94949, -2.0)	(2.94949, -1.94949)	(2.94949, 3.0)
(3.0, -2.0)	(3.0, -1.94949)	(3.0, 3.0)

Vidimo da su nul-točke približno  $x_1 = (-1, 1.5)$  i  $x_2 = (1, 1)$ . Štoviše,  $x_2$  je točno jednaka  $(1, 1)$  (1 iteracija u trećem primjeru). Nadalje, metoda ne mora konvergirati (četvrti primjer).

```
In [8]: x1=[-1.0,0]
        x2=[0.5,1.1]
        Ja(x)=ForwardDiff.jacobian(fa,x)
```

```
Out[8]: Ja (generic function with 1 method)
```

```
In [9]: # Na primjer:
        Ja([1.0,2])
```

```
Out[9]: 2×2 Array{Float64,2}:
 10.0  14.0
 10.0  -2.0
```

```
In [10]: myNewton(fa,Ja,x1,1e-10), myNewton(fa,Ja,x2,1e-10),
         myNewton(fa,Ja,[1.0,1],1e-10), myNewton(fa,Ja,[0.0,0],1e-10)
```

```
Out[10]: (([-1.18347, 1.58684], 8), ([1.0, 1.0], 6), ([1.0, 1.0], 1), ([NaN, NaN], 2))
```

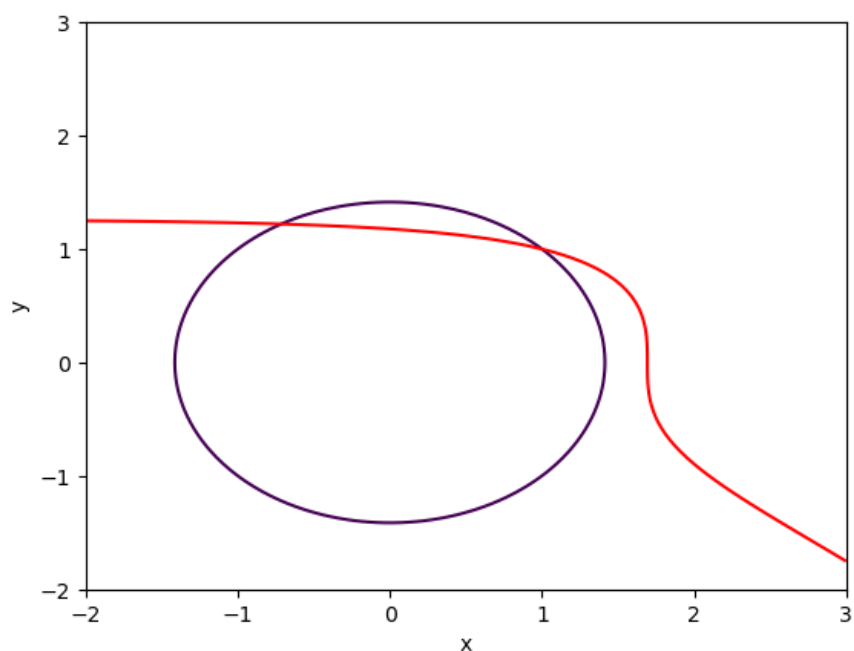
### 1.1.2 Zadatak 4.4 (b)

Rješenja sustava

$$\begin{aligned}x_1^2 - x_2^2 - 2 &= 0 \\e^{x_1-1} + x_2^3 - 2 &= 0\end{aligned}$$

$$\text{su } x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ i } x \approx \begin{bmatrix} -0.71 \\ 1.22 \end{bmatrix}.$$

```
In [11]: # Zadatak 4.4 (b) (Dennis, Schnabel (1996))
fb(x)=[x[1]^2+x[2]^2-2,exp(x[1]-1)+x[2]^3-2]
# Uzet cemo X i Y iz zadataka 4.4 (a)
Z1=Matrix(getindex.(map(fb,XY),1)')
Z2=Matrix(getindex.(map(fb,XY),2)')
C1=contour(X,Y,Z1,levels=[0])
C2=contour(X,Y,Z2,levels=[0],colors="red")
xlabel("x")
ylabel("y")
```



```
Out[11]: PyObject <matplotlib.text.Text object at 0x7fb8300f3a90>
```

```
In [12]: # U knjizi je navedena samo jedna nultočka.
Jb(x)=ForwardDiff.jacobian(fb,x)
myNewton(fb,Jb,[-1.0,1],1e-10), myNewton(fb,Jb,[0.8,1.2],1e-10)
```

```
Out[12]: (([-0.713747, 1.22089], 5), ([1.0, 1.0], 5))
```

### 1.1.3 Zadatak 4.4 (c)

Zadan je problem  $f(x) = 0$ , gdje je

$$f(x) = \begin{bmatrix} x_1 \\ x_2^2 - x_2 \\ e^{x_3} - 1 \end{bmatrix}.$$

Točna rješenja su  $x = (0,0,0)$  i  $x = (0,-1,0)$ . Izračunat ćemo nul-točke s nekoliko početnih aproksimacija.

```
In [13]: # Zadatak 4.4 (c) (Dennis, Schnabel (1996))
         fc(x)=[x[1],x[2]^2+x[2],exp(x[3])-1]
         Jc(x)=ForwardDiff.jacobian(fc,x)
```

```
Out[13]: Jc (generic function with 1 method)
```

```
In [14]: myNewton(fc,Jc,[-1.0,1.0,0.0],1e-10),myNewton(fc,Jc,[1.0,1,1],1e-10),
         myNewton(fc,Jc,[-1.0,1,-10],1e-10),myNewton(fc,Jc,[0.5,-1.5,0],1e-10)
```

```
Out[14]: (([0.0, 0.0, 0.0], 7), ([0.0, 0.0, 7.78375e-17], 7),
         ([0.0, 0.0666667, NaN], 2), ([0.0, -1.0, 0.0], 6))
```

### 1.1.4 Zadatak 4.4 (d)

Zadana je funkcija

$$f(x) = 100(x_2 - x_1)^2 + (1 - x_1)^2.$$

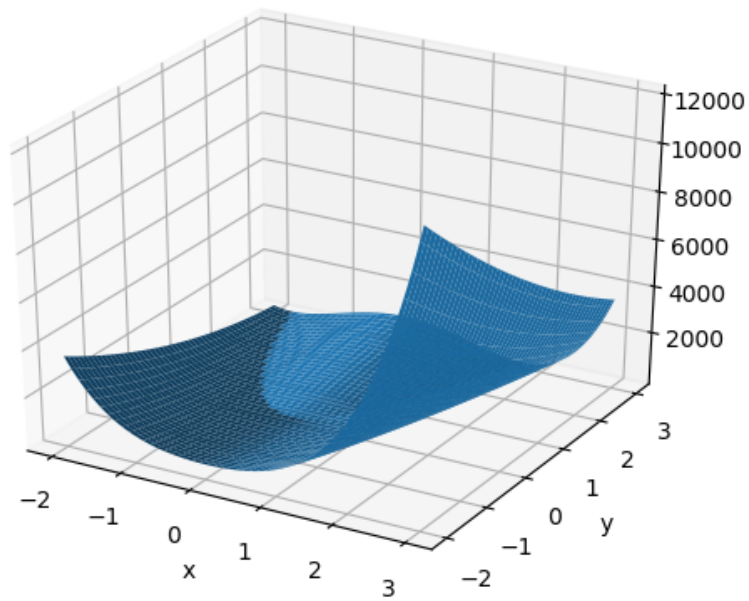
Tražimo moguće ekstreme funkcije, odnosno želimo riješiti jednadžbu

$$\text{grad } f(x) = 0.$$

```
In [15]: # Zadatak 4.4 (d) (Rosenbrock parabolic valley)
         fd1(x)=100(x[2]-x[1]^2)^2+(1-x[1])^2
```

```
Out[15]: fd1 (generic function with 1 method)
```

```
In [16]: # Nacrtajmo funkciju.
         # Uzet cemo X i Y iz zadatka 4.4 (a)
         Z=Matrix(map(fd1,XY)')
         surf(X,Y,Z)
         xlabel("x")
         ylabel("y")
```



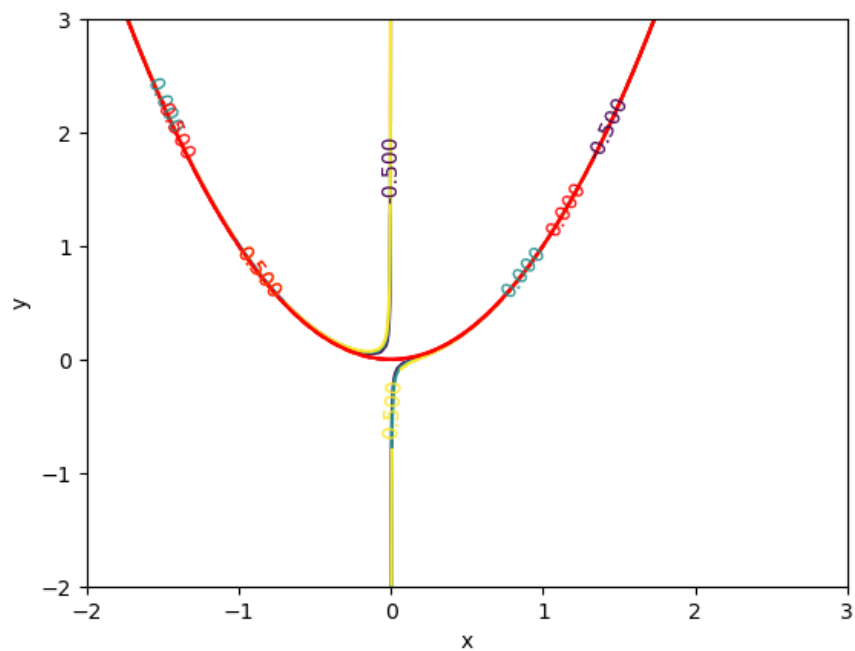
Out[16]: PyObject <matplotlib.text.Text object at 0x7fb82fd78978>

In [17]: *# Funkcija je zahtjevna u smislu određivanje ekstrema*  
 fdg(x)=ForwardDiff.gradient(fd1,collect(x))

Out[17]: fdg (generic function with 1 method)

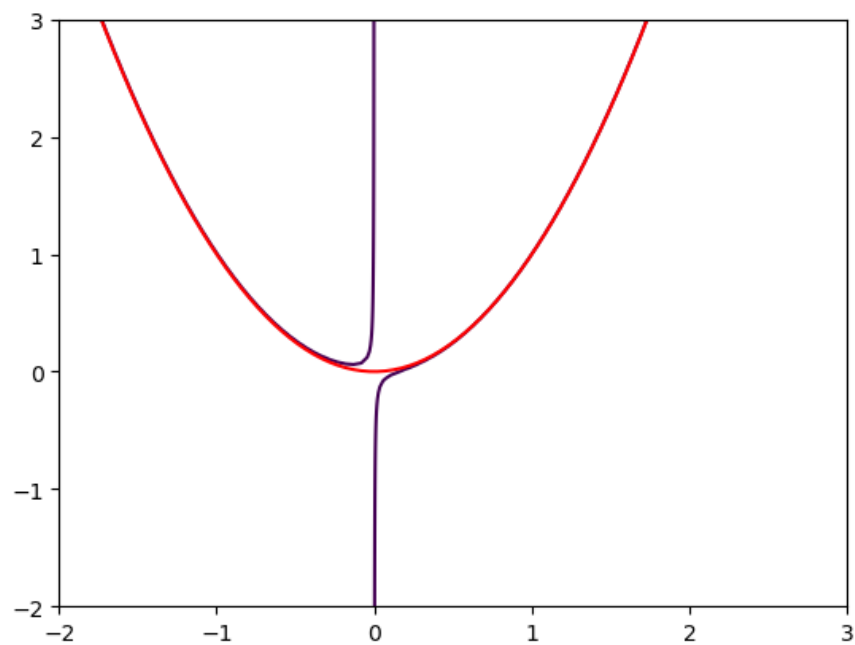
In [18]: Z1=Matrix(getindex.(map(fdg,XY),1)')  
 Z2=Matrix(getindex.(map(fdg,XY),2)')  
 C1=contour(X,Y,Z1,levels=[-0.5,0,0.5])  
 C2=contour(X,Y,Z2,levels=[-0.5,0,0.5],colors="red")  
 clabel(C1,inline=1, fontsize=10)  
 clabel(C2,inline=1, fontsize=10)  
 xlabel("x")  
 ylabel("y")





Out[18]: PyObject <matplotlib.text.Text object at 0x7fb830157860>

```
In [19]: # Pogledajmo samo nivo-krivulju za z=0
C1=contour(X,Y,Z1,levels=[0])
C2=contour(X,Y,Z2,levels=[0],colors="red")
```



Out[19]: PyObject <matplotlib.contour.QuadContourSet object at 0x7fb82ff0a7b8>

Iz kontura vidimo da je primjer numerički zahtjevan, dok analitički lako vidimo da je jedina nultočka  $x_1 = (1, 1)$ .

U ovom primjeru funkcija je zadana kao gradijent skalarne funkcije pa Jacobijevu matricu računamo korištenjem funkcije `ForwardDiff.hessian()` koja računa aproksimaciju matrice drugih parcijalnih derivacija polazne funkcije.

In [20]: `myNewton(fdg,x->ForwardDiff.hessian(fdg,x),[-1.0,2.0],1e-10)`

Out[20]: `([1.0, 1.0], 7)`

### 1.1.5 Zadatak 4.4 (e)

Zadana je funkcija

$$f(x) = \sum_{i=1}^{11} \left( x_3 \cdot \exp \left( -\frac{(t_i - x_1)^2}{x_2} \right) - y_i \right)^2,$$

gdje su brojevi  $(t_i, y_i)$  zadani tablicom:

$i$	1	2	3	4	5	6	7	8	9	10	11
$t_i$	0	1	2	3	4	5	6	7	8	9	10
$y_i$	0.001	.01	.04	.12	.21	.25	.21	.12	.04	.01	.001

Želimo riješiti jednadžbu

$$\text{grad } f(x) = 0.$$

Za razliku od prethodnih zadataka, gdje je kondicija

$$\kappa(J) = O(10)$$

u zadacima (a), (b) i (c) i

$$\kappa(J) = O(1000)$$

u zadatku (d), u ovom zadatku je

$$\kappa(J) > O(10^6)$$

pa je metoda netočna i ne konvergira prema točnom rješenju  $x = (4.93, 2.62, 0.28)$ .

In [21]: `t=collect(0:10)`

`y=[0.001,0.01,0.04,0.12,0.21,0.25,0.21,0.12,0.04,0.01,0.001]`

`fe(x)=sum([( x[3]*exp(-((t[i]-x[1])^2/x[2]))-y[i])^2 for i=1:11])`

```
Out [21]: fe (generic function with 1 method)
```

```
In [22]: # Početna točka je vrlo blizu rješenja
x0=[4.9,2.63,0.28]
fe(x0)
feg(x)=ForwardDiff.gradient(fe,x)
Je(x)=ForwardDiff.hessian(fe,x)
feg(x0), cond(Je(x0))
```

```
Out [22]: ([2.71553e-6, 0.029986, 1.13247], 173703.69351181446)
```

```
In [23]: x1,iter=myNewton(feg,Je,x0,1e-8)
```

```
Out [23]: ([6.50244, 0.00245085, -1.60888e-7], 100)
```

```
In [24]: feg(x1)
```

```
Out [24]: 3-element Array{Float64,1}:
 1.523560685122021e-51
 2.043007334415462e-49
-3.073712817083802e-47
```

```
In [25]: x0=[4.9,2.62,0.28]
x1,iter=myNewton(feg,Je,x0,1e-8)
```

```
Out [25]: ([NaN, NaN, NaN], 13)
```

## 1.2 Broydenova metoda

Za odabranu početnu aproksimaciju  $x_0$  i matricu  $B_0$ , za  $k = 0, 1, 2, \dots$ , računamo redom:

$$\begin{aligned} B_k \cdot s_k &= -f(x_k) \quad (\text{sustav}) \\ x_{k+1} &= x_k + s_k \\ y_k &= f(x_{k+1}) - f(x_k) \\ B_{k+1} &= B_k + \frac{(y_k - B_k s_k) s_k^T}{s_k \cdot s_k} \end{aligned}$$

Na ovaj način izbjegavamo računanje Jacobijeve matrice u svakom koraku. Možemo uzeti  $B_0 = J(x_0)$ , ali i neku drugu matricu.

```
In [26]: function myBroyden(f::Function,B::Matrix,x::Vector{T},ε::T) where T
    iter=0
    s=ones(T,length(x))
    ζ=x
```

```

        while norm(s)>ε && iter<100
            s=-(B\f(x))
            ζ=x+s
            y=f(ζ)-f(x)
            B=B+(y-B*s)*(s/(s*s))'
            x=ζ
            iter+=1
        end
        ζ,iter
    end
end

```

Out[26]: myBroyden (generic function with 1 method)

```

In [27]: # Zadatak 4.4 (a)
x0=[-1.0,0.0]
x1=[1.0,1.5]
myBroyden(fa,Ja(x0),x0,1e-10), myBroyden(fa,Ja(x1),x1,1e-10)

```

Out[27]: (([-1.18347, 1.58684], 12), ([1.0, 1.0], 7))

```

In [28]: # Objasnite ponašanje metode kada za početnu matricu uzmemo jediničnu matricu!
# Nova implementacija funkcije eye()
eye(n)=Matrix{Float64}(I,n,n)
myBroyden(fa,eye(2),x0,1e-10), myBroyden(fa,eye(2),x1,1e-10),
myBroyden(fa,eye(2),[-1,1.5],1e-10)

```

Out[28]: (([-0.0356391, -2.53166], 100), ([0.916022, -3.04547], 100), ([-1.18347, 1.58684], 14))

```

In [29]: # Zadatak 4.4 (b)
x0=[-1.0,1]
x1=[0.8,1.2]
myBroyden(fb,Jb(x0),x0,1e-10),
myBroyden(fb,Jb(x1),x1,1e-10)

```

Out[29]: (([-0.713747, 1.22089], 9), ([1.0, 1.0], 9))

```

In [30]: # Zadatak 4.4 (c)
x0=[-1.0,1,0]
x1=[0.5,-1.5,0]
myBroyden(fc,Jc(x0),x0,1e-10),
myBroyden(fc,Jc(x1),x1,1e-10)

```

Out[30]: (([0.0, 5.96536e-26, 0.0], 9), ([0.0, -1.0, 0.0], 8))

```

In [31]: # Zadatak 4.4 (d)
x0=[-1.0,2]

```

```

x1=[0.8,0.5]
myBroyden(fdg,(x->ForwardDiff.hessian(fd1,x))(x0),x0,1e-10), # ali
myBroyden(fdg,(x->ForwardDiff.hessian(fd1,x))([1,2.0]),x0,1e-10),
myBroyden(fdg,(x->ForwardDiff.hessian(fd1,x))(x1),x1,1e-10)

```

```

Out [31]: (([0.834064, 0.695042], 100), ([1.0, 1.0], 4), ([1.0, 1.0], 29))

```

```

In [32]: # Zadatak 4.4 (e)
x0=[4.9,2.6,0.2]
x1,iter=myBroyden(feg,(x->ForwardDiff.hessian(fe,x))(x0),x0,1e-10)

```

```

Out [32]: ([18.9003, 1.32399, 0.0665517], 6)

```

```

In [33]: feg(x1)

```

```

Out [33]: 3-element Array{Float64,1}:
 1.8552699559685368e-29
-6.235898383995474e-29
-2.0734616070093272e-29

```

### 1.3 Davidon-Fletcher-Powell (DFP) metoda

DFP je optimizacijska metoda koja traži točke ekstrema funkcije  $F : \mathbb{R}^n \rightarrow \mathbb{R}$ , u kojem slučaju je  $f(x) = \text{grad } F(x)$ .

Za odabranu početnu aproksimaciju  $x_0$  i matricu  $H_0$ , za  $k = 0, 1, 2, \dots$ , računamo redom:

$$\begin{aligned}
 s_k &= -H_k f(x_k) \\
 \beta_k &= \arg \min_{\beta} F(x_k + \beta s_k) \\
 s_k &= \beta_k s_k \\
 x_{k+1} &= x_k + s_k \\
 y_k &= f(x_{k+1}) - f(x_k) \\
 H_{k+1} &= H_k + \frac{s_k s_k^T}{y_k \cdot s_k} - \frac{H_k y_k y_k^T H_k}{y_k \cdot (H_k y_k)}.
 \end{aligned}$$

Za matricu  $H_0$  možemo uzeti jediničnu matricu, a za izvršavanje iteracije nije potrebno rješavati sustav linearnih jednačnji, već se sva ažuriranja vrše s  $O(n^2)$  operacija.

Jednodimenzionalnu minimizaciju po pravcu  $x_k + \beta s_k$  računamo tako što metodom bisekcije tražimo nul-točke usmjerene derivacije.

```

In [34]: function mybisection(f::Function,a::T,b::T,ε::T) where T
          fa=f(a)
          fb=f(b)

```

```

x=T
fx=T
if fa*fb>zero(T)
    #return "Incorrect interval"
    if abs(fa)>abs(fb)
        return b,fb,0
    else
        return a,fa,0
    end
end
iter=0
while b-a>ϵ && iter<1000
    x=(b+a)/2.0
    fx=f(x)
    if fa*fx<zero(T)
        b=x
        fb=fx
    else
        a=x
        fa=fx
    end
    iter+=1
    # @show x,fx
end
x,fx,iter
end

```

Out[34]: mybisection (generic function with 1 method)

```

In [35]: function myDFP(f::Function,H::Matrix,x::Vector{T},ϵ::T) where T
    iter=0
    s=ones(T,length(x))
    ζ=x
    while norm(s)>ϵ && iter<50
        s=-H*f(x)
        s0=s/norm(s)
        F(ζ)=f(x+ζ*s)·s0
        β,fx,iterb=mybisection(F,0.0,1.0,10*eps())
        s*=β
        ζ=x+s
        y=f(ζ)-f(x)
        z=H*y
        H=H+(s/(y·s))*s'-(z/(y·z))*z'
        x=ζ
        iter+=1
    end
    ζ,iter
end

```

```
Out [35]: myDFP (generic function with 1 method)
```

**Primjer.** Nađimo točku ekstrema funkcije

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2.$$

Funkcija ima minimum u točki (1, 3).

```
In [36]: fs(x) = (x[1] + 2*x[2]-7)^2 + (2*x[1] + x[2]-5)^2
```

```
Out [36]: fs (generic function with 1 method)
```

```
In [37]: fs([1,2])
```

```
Out [37]: 5
```

```
In [38]: fsg(x)=ForwardDiff.gradient(fs,x)
```

```
Out [38]: fsg (generic function with 1 method)
```

```
In [39]: myDFP(fsg,eye(2),[0.8,2.7],eps())
```

```
Out [39]: ([1.0, 3.0], 4)
```

```
In [40]: # Zadatak 4.4 (d)
         myDFP(fdg,eye(2),[0.9,1.1],1.0e-10)
```

```
Out [40]: ([1.0, 1.0], 9)
```

```
In [41]: # Zadatak 4.4 (e)
         myDFP(feg,eye(3),[4.9,2.6,0.2],1.0e-10)
```

```
Out [41]: ([4.89611, 46.1979, 0.00158913], 25)
```

## 1.4 Broyden-Fletcher-Goldfarb-Schano (BFGS) metoda

BFGS je optimizacijska metoda koja uspješno traži točke ekstrema funkcije  $F : \mathbb{R}^n \rightarrow \mathbb{R}$ , u kojem slučaju je  $f(x) = \text{grad } F(x)$ .

Metoda je slična DFP metodi, s nešto boljim svojstvima konvergencije.

Neka je zadana funkcija  $F : \mathbb{R}^n \rightarrow \mathbb{R}$ , čiji minimum tražimo, i neka je  $f(x) = \text{grad } F(x)$ .

Za odabranu početnu aproksimaciju  $x_0$  i matricu  $H_0$ , za  $k = 0, 1, 2, \dots$ , računamo redom:

$$\begin{aligned}
s_k &= -H_k f(x_k) \\
\beta_k &= \arg \min F(x_k + \beta_k s_k) \\
s_k &= \beta_k s_k \\
x_{k+1} &= x_k + s_k \\
y_k &= f(x_{k+1}) - f(x_k) \\
H_{k+1} &= \left( I - \frac{s_k y_k^T}{y_k \cdot s_k} \right) H_k \left( I - \frac{y_k s_k^T}{y_k \cdot s_k} \right) + \frac{s_k s_k^T}{y_k \cdot s_k}.
\end{aligned}$$

Za matricu  $H_0$  možemo uzeti jediničnu matricu, a za izvršavanje iteracije nije potrebno rješavati sustav linearnih jednačbi, već se sva ažuriranja vrše s  $O(n^2)$  operacija.

Jednodimenzionalnu minimizaciju po pravcu  $x_k + \beta s_k$  računamo tako što metodom bisekcije tražimo nul-točke usmjerene derivacije.

```

In [42]: function myBFGS(f::Function,H::Matrix,x::Vector{T},ε::T) where T
    iter=0
    s=ones(T,length(x))
    ζ=x
    while norm(s)>ε && iter<50
        s=-H*f(x)
        s0=s/norm(s)
        F(ζ)=f(x+ζ*s)·s0
        β,fx,iterb=mybisection(F,0.0,1.0,10*eps())
        s*=β
        ζ=x+s
        y=f(ζ)-f(x)
        z=H*y
        α=y·s
        s1=s/α
        H=H-s1*z'-z*s1'+s1*(y·z)*s1'+s1*s1'
        x=ζ
        iter+=1
    end
    ζ,iter
end

```

Out [42]: myBFGS (generic function with 1 method)

```
In [43]: myBFGS(fsg,eye(2),[0.8,2.7],eps())
```

Out [43]: ([1.0, 3.0], 4)

```
In [44]: # Zadatak 4.4 (d)
myBFGS(fdg,eye(2),[0.9,1.1],1e-10)
```



```
Out [44]: ([1.0, 1.0], 9)
```

```
In [45]: # Zadatak 4.4 (e)
x0=[4.9,2.6,0.2]
x1,iter=myBFGS(feg,eye(3),x0,1e-10)
```

```
Out [45]: ([2.08908, 31226.6, 0.00100056], 50)
```

## 1.5 Julia paketi

Prethodni programi su jednostavne implemntacije navedenih algoritama radi ilustracije. Julia ima paket [NLsolve.jl](#) za rješavanje sustava nelinearnih jednažbi i paket [Optim.jl](#) za nelinearnu optimizaciju.

```
In [46]: using NLsolve
```

```
In [47]: # Zadatak 4.4 (a)
x1=[-1.0,0]
x2=[0.5,1.1]
function fa!(fvec,x)
    fvec[1] = 2(x[1]+x[2])^2+(x[1]-x[2])^2-8
    fvec[2] = 5*x[1]^2+(x[2]-3)^2-9
end
```

```
Out [47]: fa! (generic function with 1 method)
```

```
In [48]: nlsolve(fa!,x1)
```

```
Out [48]: Results of Nonlinear Solver Algorithm
* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [-1.0, 0.0]
* Zero: [-1.18347, 1.58684]
* Inf-norm of residuals: 0.000000
* Iterations: 5
* Convergence: true
* |x - x'| < 0.0e+00: false
* |f(x)| < 1.0e-08: true
* Function Calls (f): 6
* Jacobian Calls (df/dx): 6
```

```
In [49]: nlsolve(fa!,x2)
```

```
Out [49]: Results of Nonlinear Solver Algorithm
* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [0.5, 1.1]
* Zero: [1.0, 1.0]
```

```

* Inf-norm of residuals: 0.000000
* Iterations: 5
* Convergence: true
  *  $|x - x'| < 0.0e+00$ : false
  *  $|f(x)| < 1.0e-08$ : true
* Function Calls (f): 6
* Jacobian Calls (df/dx): 6

```

In [50]: # Zadatak 4.4 (b)

```

function fb!(fvec,x)
    fvec[1] = x[1]^2+x[2]^2-2
    fvec[2] = exp(x[1]-1)+x[2]^3-2
end
nlsolve(fb!,-1.0,1)), nlsolve(fb!, [0.8,1.2])

```

Out[50]: (Results of Nonlinear Solver Algorithm

```

* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [-1.0, 1.0]
* Zero: [-0.713747, 1.22089]
* Inf-norm of residuals: 0.000000
* Iterations: 4
* Convergence: true
  *  $|x - x'| < 0.0e+00$ : false
  *  $|f(x)| < 1.0e-08$ : true
* Function Calls (f): 5
* Jacobian Calls (df/dx): 5, Results of Nonlinear Solver Algorithm
* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [0.8, 1.2]
* Zero: [1.0, 1.0]
* Inf-norm of residuals: 0.000000
* Iterations: 4
* Convergence: true
  *  $|x - x'| < 0.0e+00$ : false
  *  $|f(x)| < 1.0e-08$ : true
* Function Calls (f): 5
* Jacobian Calls (df/dx): 5)

```

In [51]: # Zadatak 4.4 (c)

```

function fc!(fvec,x)
    fvec[1] = x[1]
    fvec[2] = x[2]^2+x[2]
    fvec[3] = exp(x[3])-1
end
nlsolve(fc!,-1.0,1.0,0.0)), nlsolve(fc!, [1.0,1,1]),
nlsolve(fc!,-1.0,1,-10)), nlsolve(fc!, [0.5,-1.5,0])

```

Out[51]: (Results of Nonlinear Solver Algorithm

```

* Algorithm: Trust-region with dogleg and autoscaling

```

```

* Starting Point: [-1.0, 1.0, 0.0]
* Zero: [0.0, 2.32831e-10, 0.0]
* Inf-norm of residuals: 0.000000
* Iterations: 5
* Convergence: true
  *  $|x - x'| < 0.0e+00$ : false
  *  $|f(x)| < 1.0e-08$ : true
* Function Calls (f): 6
* Jacobian Calls (df/dx): 6, Results of Nonlinear Solver Algorithm
* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [1.0, 1.0, 1.0]
* Zero: [0.0, 2.32831e-10, 1.22324e-12]
* Inf-norm of residuals: 0.000000
* Iterations: 5
* Convergence: true
  *  $|x - x'| < 0.0e+00$ : false
  *  $|f(x)| < 1.0e-08$ : true
* Function Calls (f): 6
* Jacobian Calls (df/dx): 6, Results of Nonlinear Solver Algorithm
* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [-1.0, 1.0, -10.0]
* Zero: [0.0, 4.99781e-11, 8.13171e-17]
* Inf-norm of residuals: 0.000000
* Iterations: 20
* Convergence: true
  *  $|x - x'| < 0.0e+00$ : false
  *  $|f(x)| < 1.0e-08$ : true
* Function Calls (f): 21
* Jacobian Calls (df/dx): 8, Results of Nonlinear Solver Algorithm
* Algorithm: Trust-region with dogleg and autoscaling
* Starting Point: [0.5, -1.5, 0.0]
* Zero: [0.0, -1.0, 0.0]
* Inf-norm of residuals: 0.000000
* Iterations: 5
* Convergence: true
  *  $|x - x'| < 0.0e+00$ : false
  *  $|f(x)| < 1.0e-08$ : true
* Function Calls (f): 6
* Jacobian Calls (df/dx): 6)

```

In [52]: `using Optim`

In [53]: `# Zadatak 4.4 (d)`  
`optimize(fdl, [-1.0, 2], BFGS())`

Out[53]: Results of Optimization Algorithm  
 \* Algorithm: BFGS

```

* Starting Point: [-1.0,2.0]
* Minimizer: [0.9999999926685406,0.9999999853370399]
* Minimum: 5.375030e-17
* Iterations: 35
* Convergence: true
  *  $|x - x'| \leq 0.0e+00$ : false
     $|x - x'| = 5.13e-09$ 
  *  $|f(x) - f(x')| \leq 0.0e+00$   $|f(x)|$ : false
     $|f(x) - f(x')| = 1.80e+00$   $|f(x)|$ 
  *  $|g(x)| \leq 1.0e-08$ : true
     $|g(x)| = 2.10e-11$ 
  * Stopped by an increasing objective: false
  * Reached Maximum Number of Iterations: false
* Objective Calls: 102
* Gradient Calls: 102

```

In [54]: # *Zadatak 4.4 (e) - opet ne konvergira prema rješenju*  
optimize(fe,[4.9,2.6,0.2],BFGS())

Out[54]: Results of Optimization Algorithm

```

* Algorithm: BFGS
* Starting Point: [4.9,2.6,0.2]
* Minimizer: [-188.4953862326639,2.147419575851852e6, ...]
* Minimum: 3.572548e-12
* Iterations: 41
* Convergence: true
  *  $|x - x'| \leq 0.0e+00$ : false
     $|x - x'| = 1.87e+05$ 
  *  $|f(x) - f(x')| \leq 0.0e+00$   $|f(x)|$ : false
     $|f(x) - f(x')| = 1.69e-04$   $|f(x)|$ 
  *  $|g(x)| \leq 1.0e-08$ : true
     $|g(x)| = 9.69e-09$ 
  * Stopped by an increasing objective: false
  * Reached Maximum Number of Iterations: false
* Objective Calls: 137
* Gradient Calls: 137

```