Simulacija sistema opisanog matematičkim modelom – II deo

Modeliranje i simulacija sistema

Numeričko rešavanje sistema nelinearnih algebarskih jednačina

Problem:

$$
\begin{gathered}
f_1(x_1, x_2, \dots, x_n) = 0 \\
f_2(x_1, x_2, \dots, x_n) = 0 \\
\dots. \\
f_n(x_1, x_2, \dots, x_n) = 0
\end{gathered}
$$

Vektorki zapis

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}$$

Pojednostavljen problem jedne promenljive

$$f(x) = 0$$

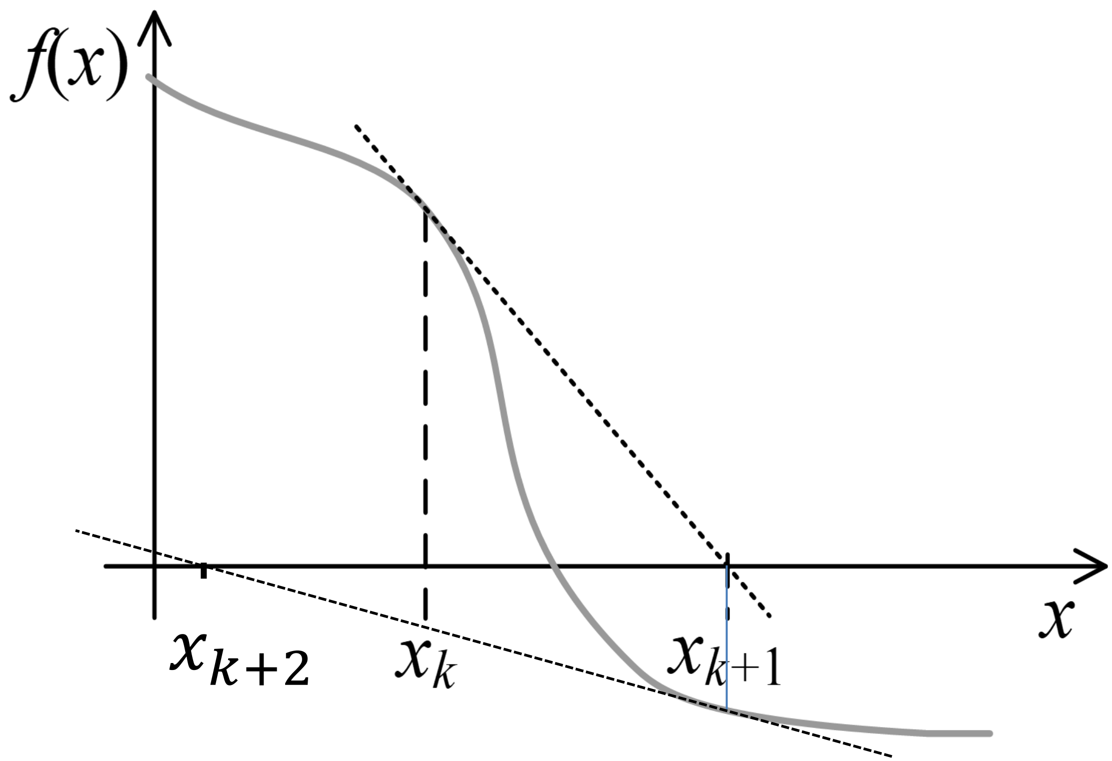
Njutn-Rapsonov postupak

Njutnov metod (poznat i kao Njutn-Rapsonov postupak):

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \qquad k = 0,1,\ldots$$

Njutn-Rapsonov postupak

- u tekućoj tački x_k se odredi tangenta na krivu f i nova tačka x_{k+1} se nalazi na preseku tangente i x-ose.



Testiranje završetka algoritma

- $|\Delta x_k| < \varepsilon_x$ ili
- $\frac{1}{2} f(x_k)^2 < \varepsilon_J$ ili
- $k > k_{max}$

Primer: Julia funkcija

```
using LinearAlgebra

function NjutnRapson(fun, x0, epsx, epsJ, kmax)
    # Njutn-Rapson: radi za jednu promenljivu
    k = 0                  # broj iteracija
    x = x0                 # početno pogađanje
    Δx = J = Inf;
    while norm(Δx) > epsx && J > epsJ && k < kmax
        (f, fprim) = fun(x)
        Δx = -f / fprim
        x += Δx
        J = 0.5 * f * f
        k += 1
    end
    return x, J, k
end;
```

Primer

Izračunati $\sqrt{a}$, za dato a.

Rešenje:

$$x = \sqrt{a}$$
$$x^2 - a = 0$$
$$f(x) = x^2 - a = 0$$
$$f'(x) = 2x$$

```
function modelSqrt(x,a)
   f=x^2-a
   fprim = 2x
   return f, fprim
end

u = NjutnRapson((x) -> modelSqrt(x,4.0),
1.0, eps(), eps(), 10);
```

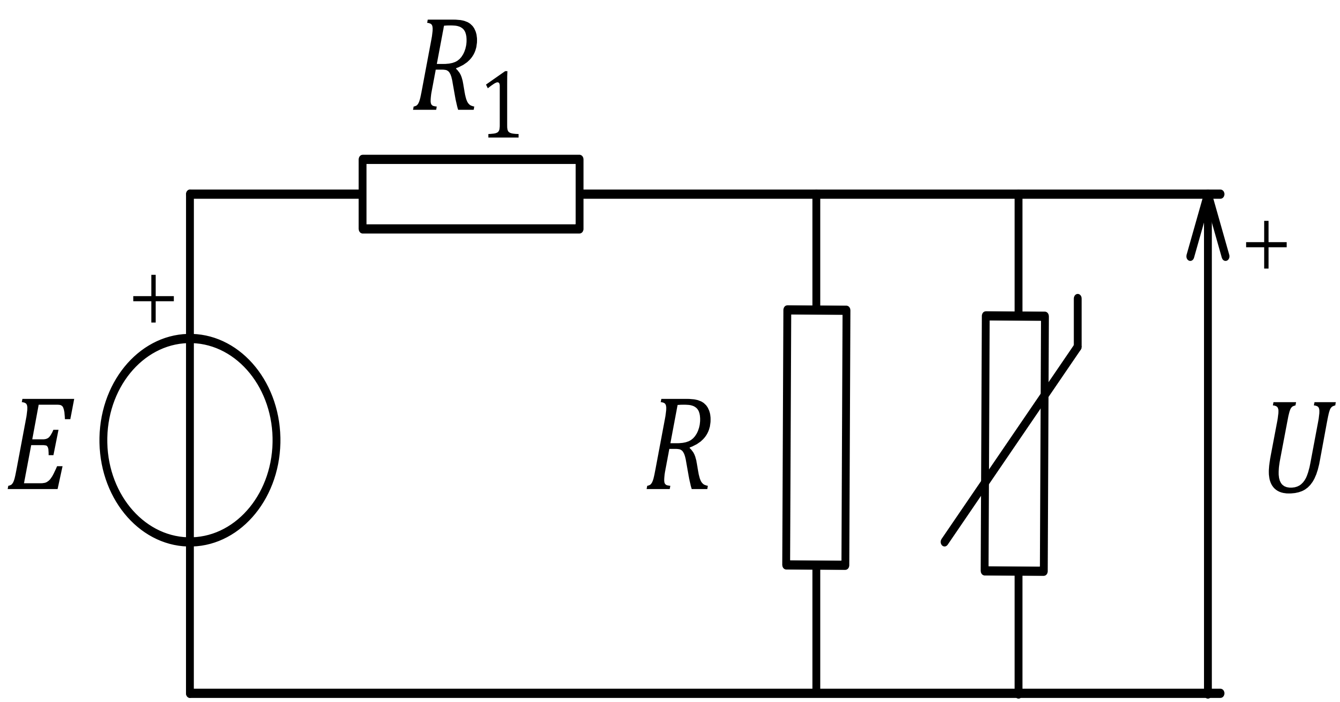
function NjutnRapson(fun, x0, epsx, epsJ, kmax)

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$
$$x_{k+1} = x_k - \frac{x_k^2 - a}{2x}$$
$$x_{k+1} = 0.5\left(x_k - \frac{a}{f'(x_k)}\right)$$

Rešenje za: $a = 4,\ x_0 = 1, \ldots$

1	2.500000000000000000000000000000
2	2.049999999999999982236431605997 5
3	2.000609756097560865129025842180
4	2.00000092922294747666001057951
5	2.0000000000000002220446049250313
6	2.000000000000000000000000000000

Primer: Potrošač otpornosti $R = 1000\,\Omega$ i tiritni otpornik (varistor) čija je karakteristika $I_v = AU^a$, gde je $A = 10^{-10}$ i $a = 3.5$, vezani su paralelno. Na njih je redno povezan otpornik otpornosti $R_1 = 1000\,\Omega$ i generator promenljive elektromotorne sile E i zanemarljive unutrašnje otpornosti. Ako se E generatora menja od $1\ kV$ do $10\ kV$, sa korakom od $0.5\ kV$, izračunati promenu napona na potrošaču i prikazati je na dijagramu. Takođe, prikazati dijagram promene struje kroz tiritni otpornik u zavisnosti od E generatora.

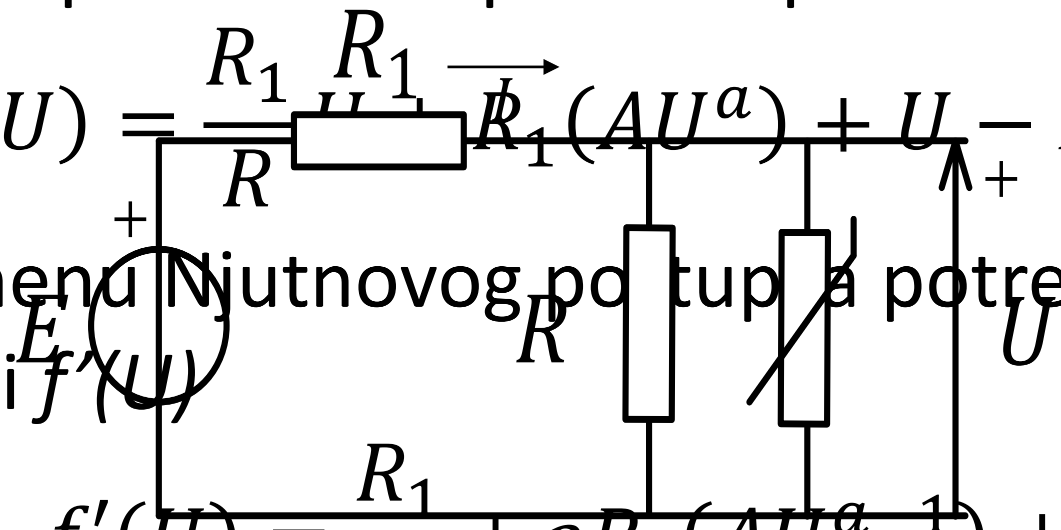


- Omov zakon $E = IR_1 + U$
- I Kirhofovog zakon - struja generatora je

$$I = U/R + AU^a$$

- model promene napona na potrošaču U

$$f(U) = \frac{R_1}{R}U + R_1(AU^a) + U - E = 0$$

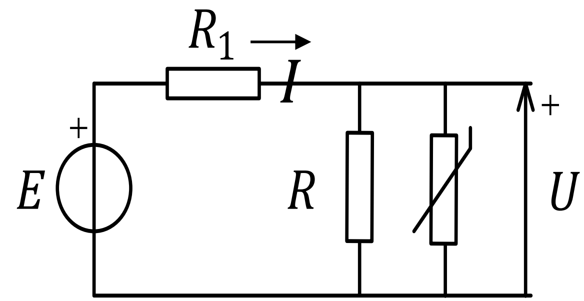


Za primenu Njutnovog postupka potrebno je odrediti $f'(U)$

$$f'(U) = \frac{R_1}{R} + \mathrm{a}R_1(AU^{a-1}) + 1$$

$$f(U) = \frac{R_1}{R}U + R_1(AU^a) + U - E = 0$$

$$f'(U) = \frac{R_1}{R} + aR_1(AU^{a-1}) + 1$$

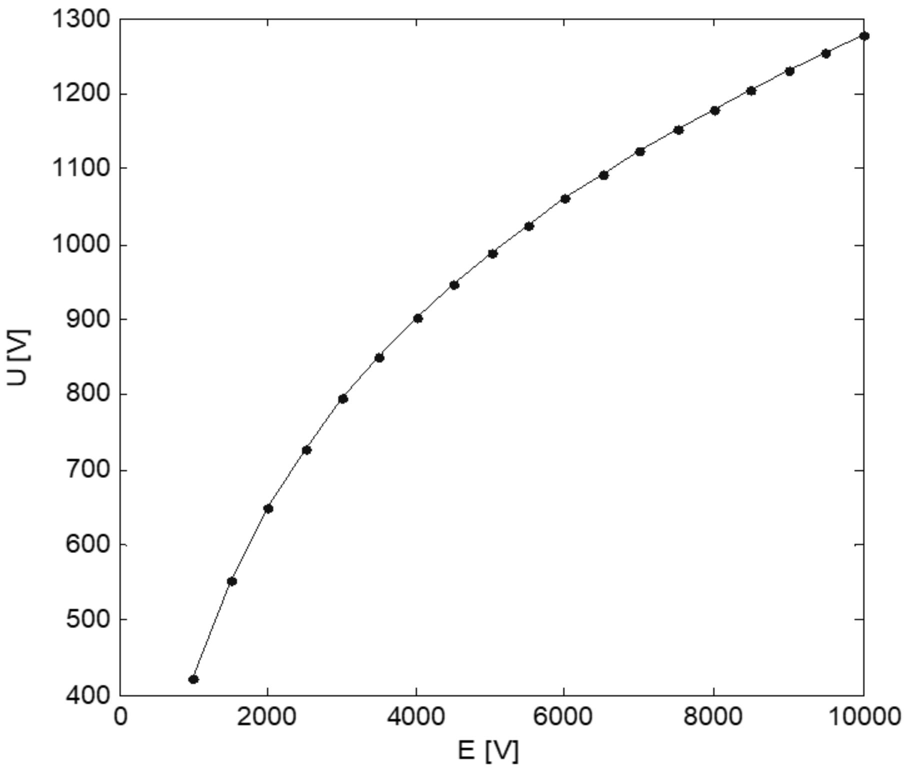


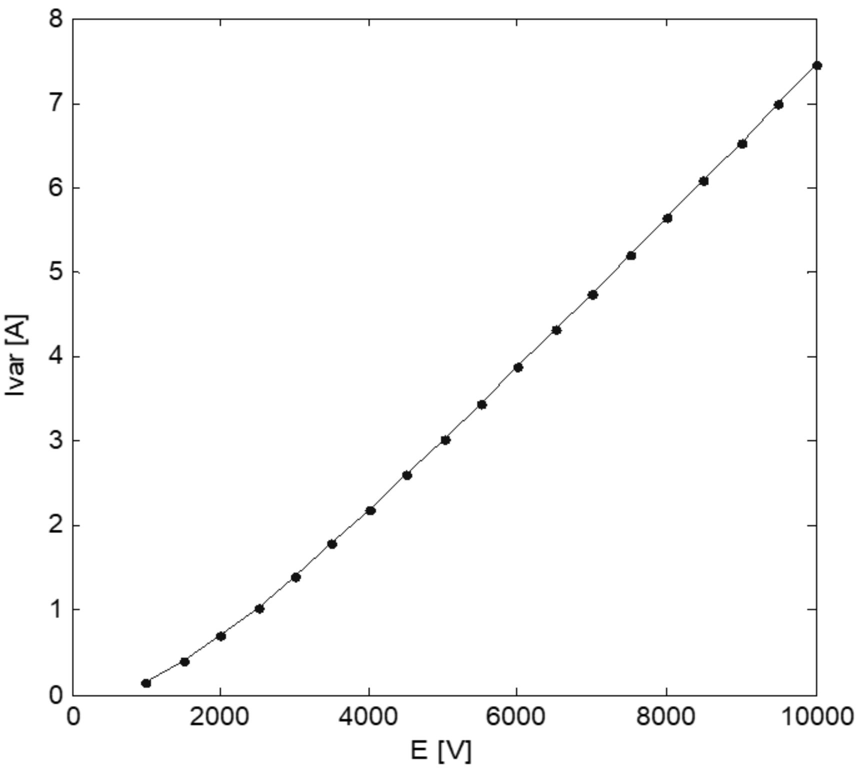

```
function modelVar(U,E)
    R=1000; R1=1000; A=1e-10; a=3.5;    # parametri el. kola
    VarI(U) = A*U .^ a                  # struja varistora
    f = R1/R*U + R1*VarI(U) + U - E     # jednačina koja se rešava f(U)=0
    fprim = R1/R + a*R1*VarI(U)/U + 1   # izvod: df/dU
    return (f, fprim)
end

E = 1000:500:10000;
U = [];
for e=E
    u, _ = NjutnRapson((x) -> modelVar(x,e), 1000, 1e-3, 1e-4, 100);
    push!(U, u);
end
[E U]
```

```
19×2 Array{Any,2}:
  1000   422.493
  1500   552.184
  2000   649.972
  2500   728.425
  3000   794.183
  3500   851.016
  4000   901.236
  4500   946.355
  5000   987.415
  5500  1025.16
  6000  1060.16
  6500  1092.82
  7000  1123.48
  7500  1152.39
  8000  1179.79
  8500  1205.83
  9000  1230.66
  9500  1254.42
 10000  1277.2
```

function NjutnRapson(fun, x0, epsx, epsJ, kmax)

Grafički prikaz rezultata





Gaus-Njutnov postupak

Traženje minimuma funkcije

$$\mathcal{J}(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(\boldsymbol{x})$$

Razvijanje funkcije $f_i(\boldsymbol{x}_{k+1})$ u Tejlorov red u okolini tačke $\boldsymbol{x}_k$

$$f_i(\boldsymbol{x}_{k+1}) \approx f_i(\boldsymbol{x}_k) + \left.\frac{\partial f_i}{\partial x_1}\right|_{\boldsymbol{x}_k} \Delta x_1 + \left.\frac{\partial f_i}{\partial x_2}\right|_{\boldsymbol{x}_k} \Delta x_2 + \cdots + \left.\frac{\partial f_i}{\partial x_n}\right|_{\boldsymbol{x}_k} \Delta x_n$$

Tada se funkcija cilja u $k+1$ iteraciji može izraziti kao

$$\mathcal{J}(\boldsymbol{x}_{k+1}) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(\boldsymbol{x}_{k+1}) \approx \frac{1}{2}\sum_{i=1}^{m}\left(\frac{\partial f_i}{\partial x_1}\Delta x_1 + \frac{\partial f_i}{\partial x_2}\Delta x_2 + \cdots + \frac{\partial f_i}{\partial x_n}\Delta x_n + f_i(\boldsymbol{x}_k)\right)^2$$

Potrebno je odrediti

$$\Delta x_j = x_j(k+1) - x_j(k), j = 1,2, \dots, n$$

(podsećanje) Kriterijum optimalnosti metode najmanjih kvadrata

$$\min_{\boldsymbol{x}} \frac{1}{2} \sum_{k=1}^{m} e_k^2 = \min_{\boldsymbol{x}} \frac{1}{2} \sum_{k=1}^{m} (a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n - b_k)^2$$

Analogija sa kriterijumom: min $\mathcal{J}(\boldsymbol{x}_{k+1})$

$$\min_{\boldsymbol{x}_{k+1}} \frac{1}{2} \sum_{i=1}^{m} f_i^2(\boldsymbol{x}_{k+1}) \approx \min_{\Delta \boldsymbol{x}} \frac{1}{2} \sum_{i=1}^{m} \left(\frac{\partial f_i}{\partial x_1} \Delta x_1 + \frac{\partial f_i}{\partial x_2} \Delta x_2 + \cdots + \frac{\partial f_i}{\partial x_n} \Delta x_n + f_i(\boldsymbol{x}_k) \right)^2$$

gde su: $a_{ij} = \frac{\partial f_i}{\partial x_j}$, $x_j = \Delta x_j$, $-b_i = f_i(\boldsymbol{x}_k)$

Stoga se rešenje za Δx može dobiti metodom najmanjih kvadrata.

Do minimuma $\mathcal{J}(\boldsymbol{x})$ se dolazi iterativno gde je $\Delta x_k \equiv \Delta x$, tj,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}_k, \qquad k = 0,1,2, \ldots$$

Definiše se Jakobijan $\boldsymbol{J}$ i vrednost funkcija kao vektor $\boldsymbol{f}$

$$\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{x}_k) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \cdots & & & \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \qquad \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \cdots \\ f_m(\boldsymbol{x}) \end{bmatrix}$$

Tada se kriterijum optimalnosti može zapisati u vektorskom obliku

$$\mathcal{J}(\boldsymbol{x}_{k+1}) \approx \left(\boldsymbol{J}(\boldsymbol{x}_k) \cdot \Delta\boldsymbol{x}_k + \boldsymbol{f}(\boldsymbol{x}_k)\right)^T \cdot \left(\boldsymbol{J}(\boldsymbol{x}_k) \cdot \Delta\boldsymbol{x}_k + \boldsymbol{f}(\boldsymbol{x}_k)\right)$$

gde se traži $\Delta\boldsymbol{x}_k$ za koje je $\mathcal{J}$ minimalno.

Iz uslova za minnimum $\nabla_{\Delta\boldsymbol{x}}\mathcal{J} = \boldsymbol{0}$

$$\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{x}_k) \cdot \left(\boldsymbol{J}(\boldsymbol{x}_k) \cdot \Delta\boldsymbol{x}_k + \boldsymbol{f}(\boldsymbol{x}_k)\right) = \boldsymbol{0}$$

dobija se

$$\Delta\boldsymbol{x}_k = -\left(\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{x}_k) \cdot \boldsymbol{J}(\boldsymbol{x}_k)\right)^{-\boldsymbol{1}} \cdot \boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{x}_k) \cdot \boldsymbol{f}(\boldsymbol{x}_k)$$

$$\Delta\boldsymbol{x} = -(\boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{J}(\boldsymbol{x}))^{-1}\boldsymbol{J}(\boldsymbol{x})^T\boldsymbol{f}(\boldsymbol{x})$$

Test kraja

- Iterativan postupak
  - kod linearnog problema – rešenje u jednom koraku
  - Kod nelinearnih problema

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$$

- Uslovi zaustavljanja:
  - $\|\Delta\boldsymbol{x}_k\| < \varepsilon_x$, ili
  - $\mathcal{J}(\boldsymbol{x}_k) < \varepsilon_J$, ili
  - $k > k_{max}$

Implemenatacija Gaus-Njutn algoritma

$$\Delta \boldsymbol{x} = -(\boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{J}(\boldsymbol{x}))^{-1} \boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{f}(\boldsymbol{x})$$

$$\mathcal{J}(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(x) = \frac{1}{2}\boldsymbol{f}^{\boldsymbol{T}}(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})$$

```
function GausNjutn(fun, x0; epsx=1e-6, epsJ=1e-6,
                                  kmax=100, logs=false)
   k = 0
   x = x0
   Δx = J = Inf
   stat = []
   while norm(Δx) > epsx && J > epsJ && k < kmax
      (f, S) = fun(x)
      Δx = -(S'*S) \ (S'*f)
      x += Δx
      J = 0.5 * f' * f
      k += 1
      if logs
         push!(stat, x)
      end
   end
   return (x, J, k, stat)
end;
```

PRIMER Napisati program za numeričko rešavanje skupa jednačina

$$f_1 = 2x_1 - 3x_2 - e^{-x_1} = 0$$
$$f_2 = -x_1 + 2x_2 - e^{-x_2} = 0$$

- Rešenje: $J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2 + e^{-x_1} & -3 \\ -1 & 2 + e^{-x_2} \end{bmatrix}$

```
function test(x)
   x1, x2 = x
   F = [2*x1-3*x2-exp(-x1)
        -x1+2*x2-exp(-x2)]
   J = [2+exp(-x1)  -3
        -1          2+exp(-x2)]

   return F, J
end
```

```
x, J, k = GausNjutn(test, [0.0; 0.0], logs=true);
```

k	J	x_1	x_2
1	1.0000	1.0000	0.6667
2	0.0839	1.4963	0.9358
3	0.0009	1.5582	0.9688
4	0.0000	1.5589	0.9692

Gradijentni algoritam

$$\mathcal{J}(\boldsymbol{x}_{k+1}) = \mathcal{J}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) \approx \mathcal{J}(\boldsymbol{x}_k) + \nabla^T\mathcal{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k$$

$$\mathcal{J}(\boldsymbol{x}_0) > \mathcal{J}(\boldsymbol{x}_1) > \cdots > \mathcal{J}(\boldsymbol{x}_k) > \mathcal{J}(\boldsymbol{x}_{k+1}) > \cdots$$

$$\mathcal{J}(\boldsymbol{x}_{k+1}) - \mathcal{J}(\boldsymbol{x}_k) < 0$$

$$\Delta\boldsymbol{x}_k = -h\nabla\mathcal{J}(\boldsymbol{x}_k), \qquad h > 0$$

$$\mathcal{J}(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{f}^T(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x})$$

$$\nabla\mathcal{J}(\boldsymbol{x}) = \nabla\boldsymbol{f}^T(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{J}^T\boldsymbol{f}(\boldsymbol{x})$$

Implementacija gradijentnog algoritma

```
function NajstrmijiPad(fun, x0, h; epsx=1e-6, epsJ=1e-6, kmax=100, logs=false)
    k = 0           # broj iteracija
    x = x0
    Δx = J = Inf
    stat = []
    while norm(Δx) > epsx && J > epsJ && k < kmax
        (f, S) = fun(x)
        Δx = -h*(S'*f)
        x += Δx
        J = 0.5 * f' * f
        k += 1
        if logs
            push!(stat, x)
            ispis(k, J, x)
        end
    end
    return (x, J, k, stat)
end;
```

```
x, J, k = NajstrmijiPad(test, [0.0; 0.0], 0.05, kmax=1000);
```

```
(x, J, k) = ([1.556588944229349, 0.9677019214952504], 9.804252666984152e-7, 446)
```

Levenberg–Markartov algoritam

$$\Delta \boldsymbol{x}_k = -(\boldsymbol{J}^T \boldsymbol{J} + \lambda_k \boldsymbol{I})^{-1} \boldsymbol{J}^T \boldsymbol{f}(\boldsymbol{x}_k)$$

$$\lambda_{k+1} = \begin{cases} \nu \lambda_k, & \mathcal{J}(\boldsymbol{x}_k) > \mathcal{J}(\boldsymbol{x}_{k-1}) \\ \dfrac{1}{\nu} \lambda_k, & \mathcal{J}(\boldsymbol{x}_k) \leq \mathcal{J}(\boldsymbol{x}_{k-1}) \end{cases}, \qquad \nu > 1$$

$$\lambda_k \to \infty: \;\; \Delta \boldsymbol{x}_k = -\frac{1}{\lambda_k} \boldsymbol{J}^T \boldsymbol{f}(\boldsymbol{x}_k)$$

$$\lambda_k \to 0: \;\; \Delta \boldsymbol{x}_k = -(\boldsymbol{J}^T \boldsymbol{J})^{-1} \boldsymbol{J}^T \boldsymbol{f}(\boldsymbol{x}_k)$$

```
function LevenbergMarkart(fun, x0; h=0.5, epsx=1e-6, epsJ=1e-6, kmax=100, logs=false)
    #norm(x) = x'*x
    k = 0                  # broj iteracija
    x = x0
    Δx = Js = Inf          # prethodna vrednost J
    λ = 1/h
    ni = 1.2
    stat = []
    while norm(Δx) > epsx && Js > epsJ && k < kmax
        (f, S) = fun(x)
        Δx = -((S'*S) + λ*I) \ S'*f
        J = 0.5 * f' * f
        k += 1
        if J > Js
            λ = λ * ni
        else
            λ = λ / ni
            x += Δx
        end
        Js = J
        if logs
            push!(stat, x)
        end
    end
    return (x, Js, k, stat)
end;
```

```
(x,J,k) = LevenbergMarkart(test, [0.0, 0.0], logs=true);
x =
    1.5586
    0.9689
J =
  1.8619e-007
k =
    16
```

k	J	λ	x_1	x_2
1	1.0000	1.6667	0.4167	0.2500
2	0.4077	1.3889	0.6940	0.4234
3	0.1989	1.1574	0.9001	0.5542
4	0.1032	0.9645	1.0614	0.6564
5	0.0542	0.8038	1.1901	0.7377
6	0.0280	0.6698	1.2926	0.8023
7	0.0139	0.5582	1.3732	0.8529
8	0.0065	0.4651	1.4348	0.8915
9	0.0028	0.3876	1.4802	0.9199
10	0.0011	0.3230	1.5119	0.9397
11	0.0004	0.2692	1.5327	0.9528
12	0.0001	0.2243	1.5454	0.9607
13	0.0000	0.1869	1.5526	0.9652
14	0.0000	0.1558	1.5562	0.9675
15	0.0000	0.1298	1.5579	0.9685
16	0.0000	0.1082	1.5586	0.9689