

Radarji

Projektna naloga pri predmetu Matematično modeliranje

Peter Mašič, Nik Žunič

April 2025

1 Opis problema

Radi bi onesposobili protizračno obrambo na nekem območju in v ta namen moramo najprej odkriti položaje radarjev raketnih sistemov. Vemo, da ima sovražnik n radarjev razporejenih na lokacijah z neznanimi koordinatami (u_k, v_k) , $k = 1, \dots, n$. Na voljo imamo letala, ki lahko na varni razdalji z neko natančnostjo merijo skupno jakost radarskih signalov, ne pa tudi smeri. Predpostavimo, da letala merijo jakost blizu tal. Vemo, da jakost signala posameznega radarja pada s kvadratom razdalje, v dani točki (x, y) je tako jakost k-tega signala enaka

$$j_k(x, y) = \frac{c}{(x - u_k)^2 + (y - v_k)^2}$$

kjer je c konstanta, ki nam jo zagotovi proizvajalec, ki je pred leti sovražniku prodal radarje, (u_k, v_k) pa je točka, kjer se nahaja k -ti radar.

Zanima nas tudi, kako vpliva število opravljenih meritev, oddaljenost merjenja od radarjev in merske napake merjenja jakosti na natančnost rezultatov.

2 Izpeljava modela

Recimo, da je $c = 1$. V m točkah s znanimi koordinatami (x_i, y_i) , $i = 1, \dots, m$ opravimo meritve z_1, \dots, z_m , ki nam dajo vrednosti za skupno jakost signalov. Približno torej velja

$$\begin{aligned} z_1 &\doteq \sum_{k=1}^n j_k(x_1, y_1) \\ z_2 &\doteq \sum_{k=1}^n j_k(x_2, y_2) \\ &\vdots \\ z_m &\doteq \sum_{k=1}^n j_k(x_m, y_m) \end{aligned}$$

Vsoto jakosti vseh signalov v i -ti točki (x_i, y_i) zapišimo kot funkcijo $V_i(u, v)$, kjer sta u in v vektorja neznanih parametrov u_k in v_k , oz. položajev (u_k, v_k) .

$$V_i(u, v) \doteq \sum_{k=1}^n j_k(x_i, y_i)$$

Naloga je torej rešiti sistem m enačb:

$$\begin{aligned} z_1 &\doteq V_1(u, v) \\ z_2 &\doteq V_2(u, v) \\ &\vdots \\ z_m &\doteq V_m(u, v) \end{aligned}$$

Koliko meritev je potrebno opraviti, da lahko vsaj teoretično določimo položaj n radarjev? Vsaka meritev $V_i(u, v)$ je sestavljena iz vsote n jakosti signalov neznanih točk (u_k, v_k) , torej imamo $2n$ neznank $u_1, \dots, u_n, v_1, \dots, v_n$. Za $2n$ neznank potrebujemo vsaj toliko enačb, torej mora biti število enačb oz. meritev $m \geq 2n$.

Sistem nekoliko preoblikujemo.

$$\begin{aligned} V_1(u, v) - z_1 &\doteq 0 \\ V_2(u, v) - z_2 &\doteq 0 \\ &\vdots \\ V_m(u, v) - z_m &\doteq 0 \end{aligned}$$

V smislu metode najmanjših kvadratov izrazimo sistem m enačb kot funkcijo F takole.

$$F(x, y, u, v, z) = \sum_{i=1}^m (V_i(u, v) - z_i)^2 \doteq 0$$

Kjer sta $x = [x_1 \ x_2 \ \dots \ x_m]^T$ in $y = [y_1 \ y_2 \ \dots \ y_m]^T$ vektorja položajev opravljenih meritev (x_i, y_i) . $u = [u_1 \ u_2 \ \dots \ u_n]^T$ in $v = [v_1 \ v_2 \ \dots \ v_n]^T$ vektorja točk (u_k, v_k) neznanih lokacij radarjev ter $z = [z_1 \ z_2 \ \dots \ z_m]^T$ vektor vrednosti meritev jakosti signala v m točkah (x_i, y_i) .

2.1 Reševanje sistema z gradientno metodo

Funkcijo F želimo minimizirati, zaradi kvadriranja bo vedno $F \geq 0$, torej iščemo globalni minimum funkcije F , za kar rabimo nastaviti parametra u in v , da bo $F \doteq 0$. Sistem lahko rešimo z gradientno metodo [5]. Izberemo začetne približke vektor $x_0 = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}$, kjer sta u_0 in v_0 vektorja z neznanimi koordinatami točke (u_k, v_k) , ter konstanto $\alpha \in \mathbb{R}$. Nato ponavljamo $x_{i+1} = x_i - \alpha * (\text{grad}F)(x_i)$, dokler ni $F \doteq 0$. Gradient funkcije F izračunamo kot:

$$\begin{aligned} \text{grad}F(x, y, u, v, z) &= \left[\frac{\partial F}{\partial u_1}, \frac{\partial F}{\partial u_2}, \dots, \frac{\partial F}{\partial u_n}, \frac{\partial F}{\partial v_1}, \frac{\partial F}{\partial v_2}, \dots, \frac{\partial F}{\partial v_n} \right]^T = \\ &= \begin{bmatrix} \sum_{i=1}^m (2 * (V_i(u, v) - z_i) * \frac{\partial j_1(x_i, y_i)}{\partial u_1}) \\ \sum_{i=1}^m (2 * (V_i(u, v) - z_i) * \frac{\partial j_2(x_i, y_i)}{\partial u_2}) \\ \vdots \\ \sum_{i=1}^m (2 * (V_i(u, v) - z_i) * \frac{\partial j_n(x_i, y_i)}{\partial u_n}) \\ \sum_{i=1}^m (2 * (V_i(u, v) - z_i) * \frac{\partial j_1(x_i, y_i)}{\partial v_1}) \\ \sum_{i=1}^m (2 * (V_i(u, v) - z_i) * \frac{\partial j_2(x_i, y_i)}{\partial v_2}) \\ \vdots \\ \sum_{i=1}^m (2 * (V_i(u, v) - z_i) * \frac{\partial j_n(x_i, y_i)}{\partial v_n}) \end{bmatrix} \end{aligned}$$

Kjer sta

$$\frac{\partial j_k(x_i, y_i)}{\partial u_k} = \frac{2 * (x_i - u_k)}{((x_i - u_k)^2 + (y_i - v_k)^2)^2}, \quad \frac{\partial j_k(x_i, y_i)}{\partial v_k} = \frac{2 * (y_i - v_k)}{((x_i - u_k)^2 + (y_i - v_k)^2)^2}$$

2.2 Implementacija gradientne metode v programskem jeziku Julia[4]

Funkcija $F(x, y, u, v, z)$, je implementirana na sledeči način.

```
function F(z, u, v, x, y)
    return sum((signalStrength(xi, yi, u, v) - zi)^2 for (xi, yi, zi) in zip(x, y, z))
end
```

Kjer je funkcija $\text{signalStrength}(xi, yi, u, v)$ enaka $V_i(u, v)$, ta pa je implementirana takole:

```
function signalStrength(xi, yi, u, v)
    return sum(1 / ((xi - ui)^2 + (yi - vi)^2) for (ui, vi) in zip(u, v))
end
```

x_i in y_i sta parametra, ki ju določa indeks funkcije V_i . Gradient funkcije F je zaradi kompleksnosti razdeljen na več delov, glavna funkcija je:

```

function GradF(z, u, v, x, y)
    du = Float64[]
    dv = Float64[]
    for k in 1:length(u)
        du_k = 0.0
        dv_k = 0.0
        for (xi, yi, zi) in zip(x, y, z)
            du_k += 2 * (signalStrength(xi, yi, u, v) - zi)
            * DSigalStrengthU(xi, yi, u[k], v[k])
            dv_k += 2 * (signalStrength(xi, yi, u, v) - zi)
            * DSigalStrengthV(xi, yi, u[k], v[k])
        end
        push!(du, du_k)
        push!(dv, dv_k)
    end
    return vcat(du, dv)
end

```

Funkciji $DSignalStrengthU(xi, yi, ui, vi)$ ter $DSignalStrengthV(xi, yi, ui, vi)$ sta parcialna odvoda $\frac{\partial j_k(x_i, y_i)}{\partial u_k}$ in $\frac{\partial j_k(x_i, y_i)}{\partial v_k}$

```

function DSigalStrengthU(xi, yi, ui, vi)
    return 2 * (xi - ui) / ((xi - ui)^2 + (yi - vi)^2)^2
end

function DSigalStrengthV(xi, yi, ui, vi)
    return 2 * (yi - vi) / ((xi - ui)^2 + (yi - vi)^2)^2
end

```

Izvajanje gradientne metode opravlja funkcija *gradmet*, ki je podobna tisti, ki smo jo napisali na vajah. Dodatno je predelana, da računa z več neznankami, medtem ko je tista na vajah iskala optimizacijo za le eno neznanko. Izsek kode, ki vsebuje glavni del z iteracijo, je sledeč:

```

function gradmet(z, x, y, alpha, x0; tol = 1e-12, maxit = 10000)
    n = 1
    x1 = x0
    for outer n in 1:maxit
        u0 = x0[1:length(x0) ÷ 2]
        v0 = x0[length(x0) ÷ 2 + 1:end]
        grad = GradF(z, u0, v0, x, y)
        x1 = x0 .- alpha .* grad

        if norm(x1 - x0) < alpha * tol
            break
        end

        x0 = x1
    end
    return (X = x1, n = n)
end

```

2.3 Reševanje sistema z Levenbergovo metodo

Sistem nelinearnih enačb lahko rešimo tudi s Newtonovo metodo [2]. Naj bo $G(\vec{x}) = [V_1(u, v) - z_1 \quad \dots \quad V_m(u, v) - z_m]^T$. Izhajamo iz linearne aproksimacije. Začnemo z začetnim približkom $x_0 = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}$, x_{k+1} je rešitev enačbe

$$L_{x_k}(\vec{x}) = G(\vec{x}) + DG(\vec{x})(\vec{x} - \vec{x}_k) = 0$$

\vec{x}_{k+1} izrazimo kot:

$$\vec{x}_{k+1} = \vec{x}_k - J(\vec{x}_k)^{-1}G(\vec{x}_k)$$

$J(\vec{x})$ je jakobijeva matrika funkcije G , $J(\vec{x}) = DG(\vec{x})$

$$J(x) = \begin{bmatrix} \frac{\partial j_1(x_1, y_1)}{\partial u_1} & \dots & \frac{\partial j_k(x_1, y_1)}{\partial u_k} & \frac{\partial j_1(x_1, y_1)}{\partial v_1} & \dots & \frac{\partial j_k(x_1, y_1)}{\partial v_k} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial j_1(x_m, y_m)}{\partial u_1} & \dots & \frac{\partial j_k(x_m, y_m)}{\partial u_k} & \frac{\partial j_1(x_m, y_m)}{\partial v_1} & \dots & \frac{\partial j_k(x_m, y_m)}{\partial v_k} \end{bmatrix}$$

Izbira začetnega približka je eden izmed največjih problemov pri reševanju sistemov nelinearnih enačb. Napačna izbira začetnega približka lahko ne vodi do tega, da model konvergira do pravilne rešitve, temveč postaja čedalje bolj netočen, ko gremo dlje stran od rešitve. Pri vsakem koraku iteracije se lahko vprašamo, če pripomore k izboljšanju rešitve. Ali se napaka $\|G\|$ zmanjša? Levenbergova metoda [3] poskuša rešiti ta problem, je kombinacija Newtonove metode in gradientne metode [1].

Imamo začetni približek $x_0 = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}$. \vec{x}_{k+1} izrazimo kot:

$$\vec{x}_{k+1} = \vec{x}_k + s_k$$

kjer je s_k , rešitev sistema:

$$(F_k^t F_k + \lambda I) s_k = -F_k^T x_k$$

λ omogoča prehod med Newtonovo iteracijo, ki hitro konvergira proti rešitvi, in med gradientno metodo, za iteracije, ko je predikcija daleč stran od rešitve.

2.4 Implementacija Levenbergove metode v Julii

Za implementacijo Levenbergove metode je uporabljenih kar nekaj funkcij iz gradientne metode, in sicer: $signalStrength(xi, yi, u, v)$ za izračun jakosti signala v točkah (x_i, y_i) , funkciji $DSignalStrengthU(xi, yi, ui, vi)$ in $DSignalStrengthV(xi, yi, ui, vi)$.

Funkcija za Levenbergovo metodo je sledeča:

```
function levenberg(z, x, y, x0; tol=1e-4, maxit=10000)
    X = x0
    k = length(X) ÷ 2
    lambda = 10.0

    for n in 1:maxit
        u = X[1:k]
        v = X[k+1:end]
        r = [signalStrength(xi, yi, u, v) - zi for (xi, yi, zi) in zip(x, y, z)]
        J = jacobian(x, y, X)

        dx = (J' * J + lambda * I(length(X))) \ (-J' * r)
        X_new = X + dx

        u_new = X_new[1:k]
        v_new = X_new[k+1:end]
        r_new = [signalStrength(xi, yi, u_new, v_new) - zi for (xi, yi, zi) in zip(x, y, z)]

        if norm(r_new) < norm(r)
            lambda /= 10
            X = X_new
            push!(koraki, X)
        else
            lambda *= 4
        end
    end
```

```

    if norm(dx) < tol
        break
    end
end

return (X = X, n = length(koraki))
end

```

Zanimiva odseka kode sta

`dx = (J' * J + lambda * I(length(X))) \ (-J' * r)`

kjer rešujemo sistem enačb

$$(F_k^t F_k + \lambda I) s_k = -F_k^T x_k$$

`dx = s_k`, in pa if stavek

```

if norm(r_new) < norm(r)
    lambda /= 10
    X = X_new
    push!(koraki, X)
else
    lambda *= 4
end

```

kjer preverimo, če iteracija poveča $\|G\|$ in zmanjšamo λ , če želimo iteracijo, ki je bolj podobna Newtonovi metodi, oz. povečamo λ , če želimo iteracijo, ki je bolj podobna gradientni metodi.

Jakobijska matrika se izračuna s funkcijo *jacobian*

```

function jacobian(x, y, X)
    k = length(X) ÷ 2
    u = X[1:k]
    v = X[k+1:end]
    J = zeros(length(x), length(X))

    for (j, (xi, yi)) in enumerate(zip(x, y))
        for l in 1:k
            J[j, l] = DSignalStrengthU(xi, yi, u[l], v[l])
            J[j, k + l] = DSignalStrengthV(xi, yi, u[l], v[l])
        end
    end

    return J
end

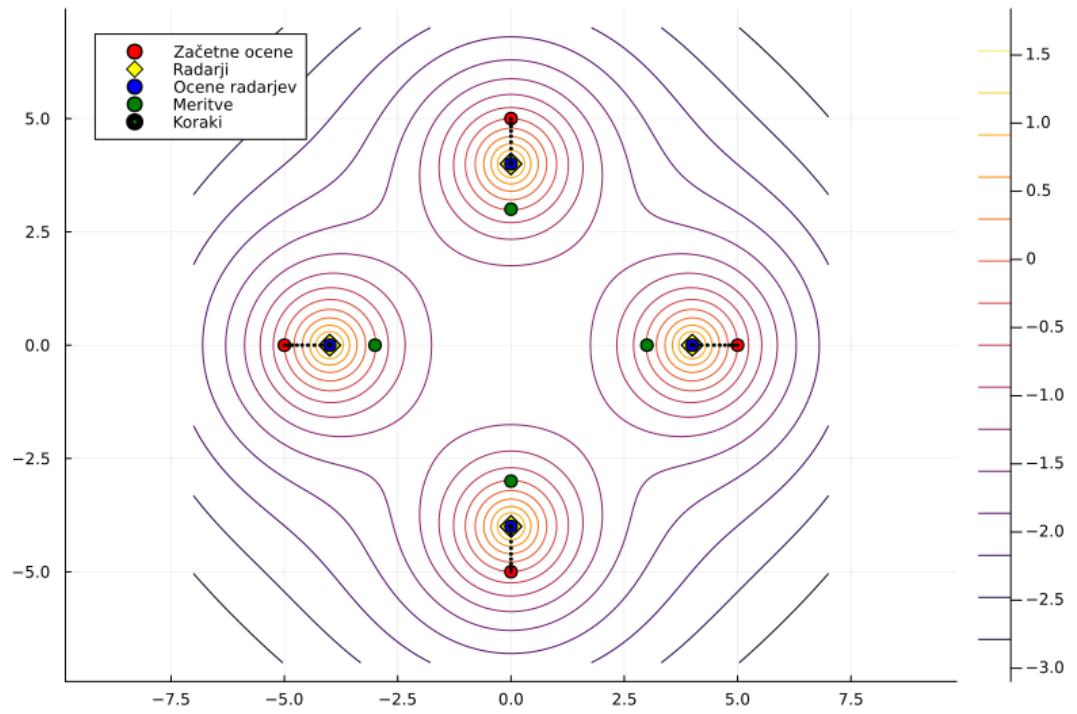
```

3 Simulacije

3.1 Simulacije z gradientno metodo

Naj bodo 4 radarji na lokacijah s koordinatami $(-4, 0)$, $(0, 4)$, $(4, 0)$, $(0, -4)$. Maksimalno število iteracij za gradientno metodo je omejeno na 10000.

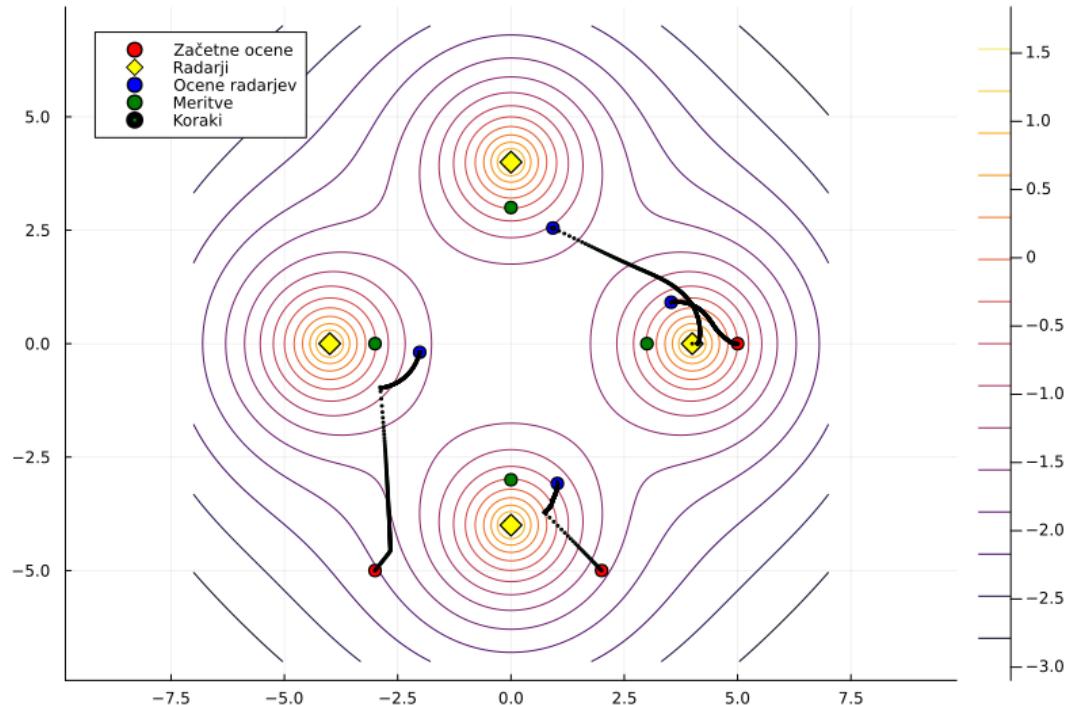
Preverimo, če model deluje na enostavnem testnem primeru, kjer so začetne ocene na lokacijah $(-5, 0)$, $(0, 5)$, $(5, 0)$, $(0, -5)$, 4 meritve pa so na lokacijah $(-3, 0)$, $(0, 3)$, $(3, 0)$, $(0, -3)$.



Število iteracij: 85, Koeficient gradientne metode: 0.2

Slika 1: Prvi testni primer

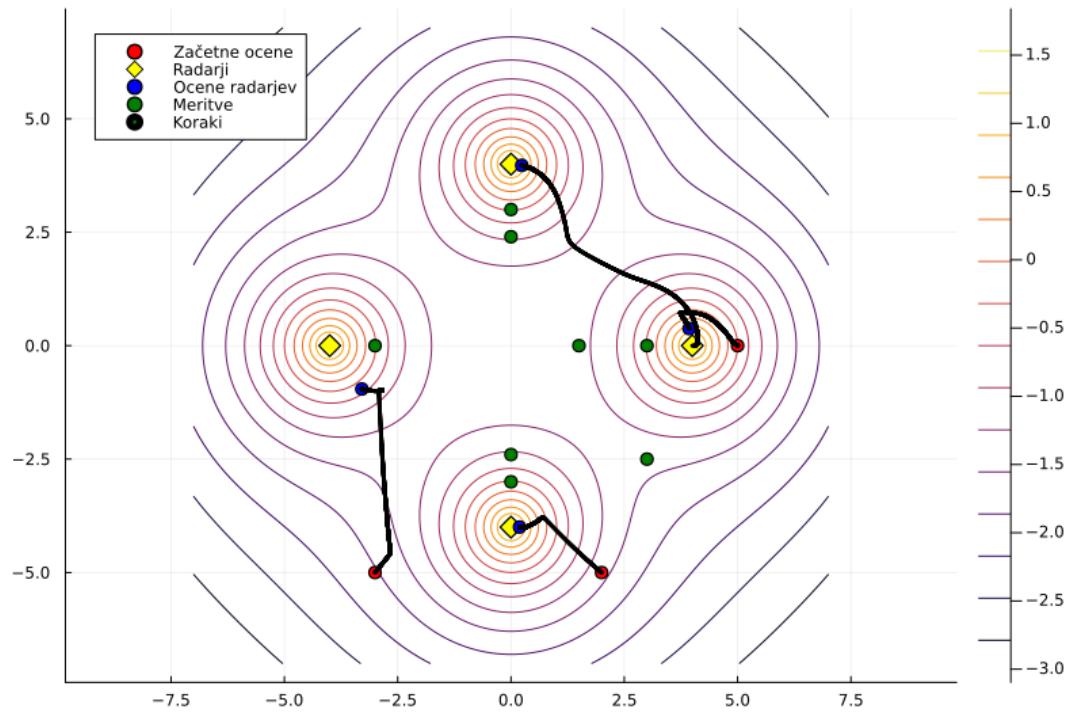
Vidimo, da model konvergira proti pravilni rešitvi. Spremenimo začetne ocene na bolj naključne lokacije, ki niso nujno blizu prave rešitve.



Število iteracij: 10000, Koeficient gradientne metode: 0.2

Slika 2: Simulacija 2

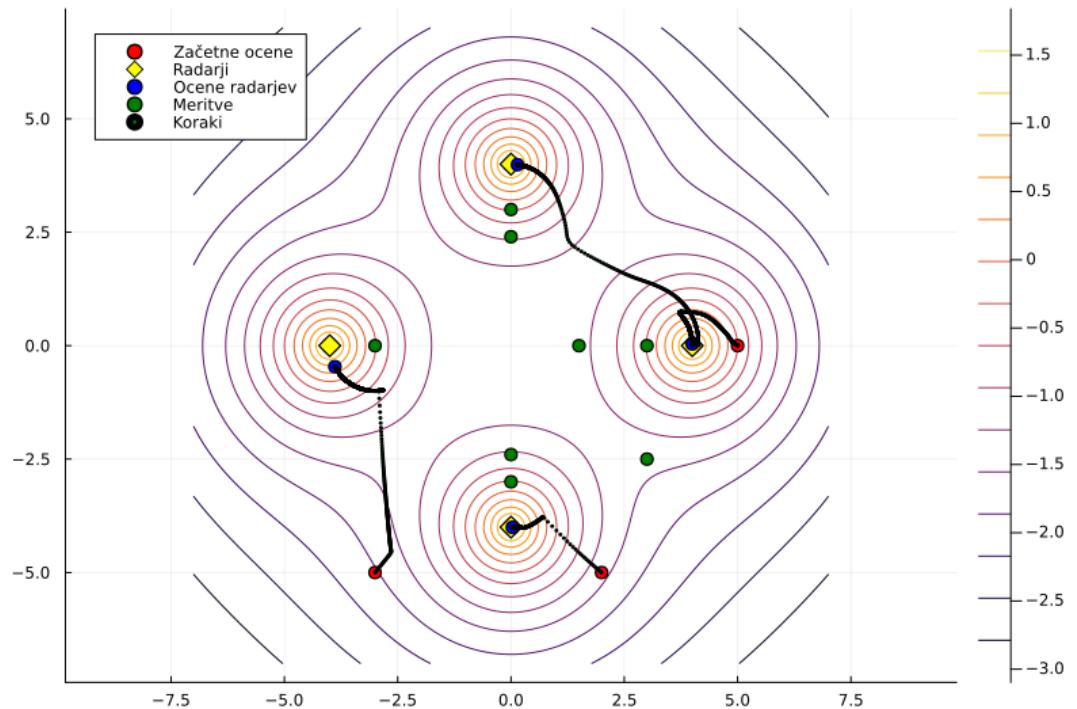
Pravi rešitvi se ne približujemo, prav tako je bilo za izračun narejenih maksimalno število iteracij (10000). Dodamo še 3 dodatne meritve za iste ocene radarjev in iste pravilne položaje.



Število iteracij: 10000, Koeficient gradientne metode: 0.02

Slika 3: Simulacija 3

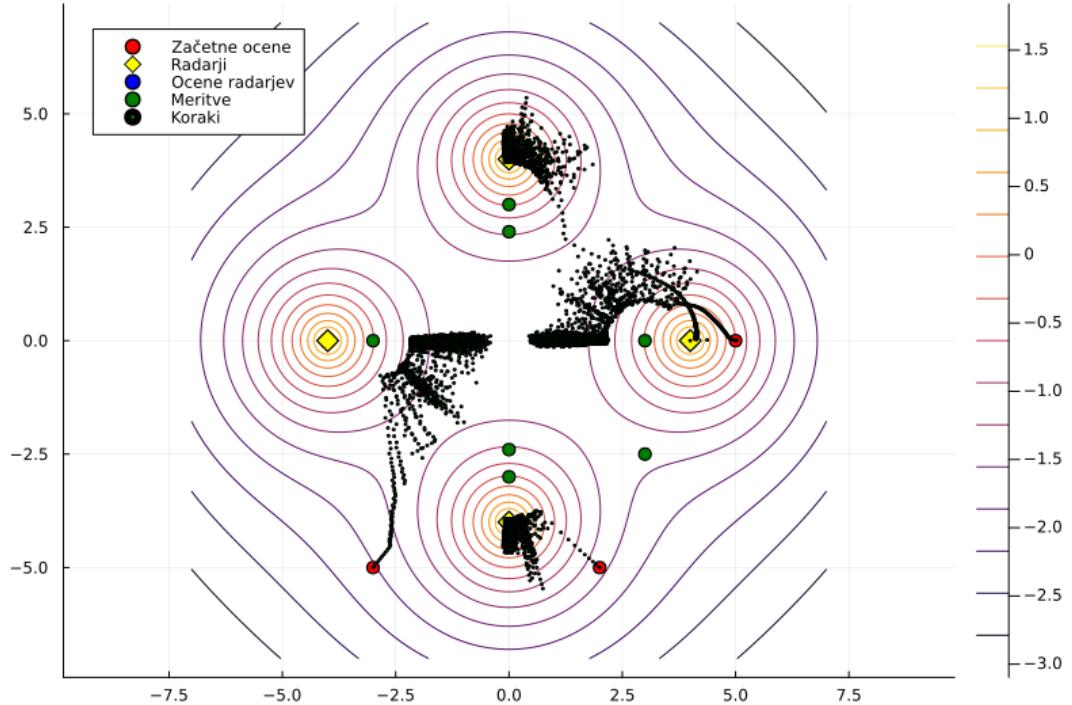
Z več meritvami se bolj približamo pravi rešitvi. Povečamo še koeficient α iz 0,02 na 0,2.



Število iteracij: 10000, Koeficient gradientne metode: 0.2

Slika 4: Simulacija 4

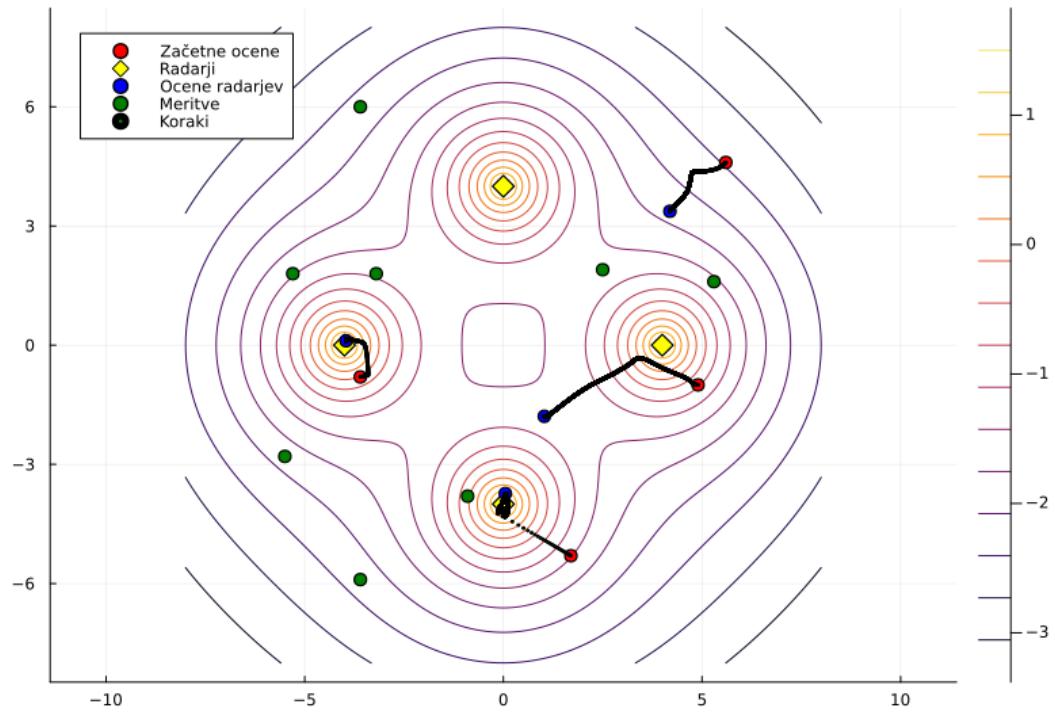
S sedmimi meritvami smo skoraj uganili pravilne položaje 4 radarjev. Na prejšnji simulaciji smo videli, da s premajhnim koeficientom α model ne konvergira najbolje do pravilne rešitve, medtem ko na tej simulaciji s $10 \times$ večjim α , veliko bolje napove pravilne položaje radarjev. Kaj pa če je α prevelik? Recimo da je $\alpha = 0,5$



Število iteracij: 10000, Koeficient gradientne metode: 0.5

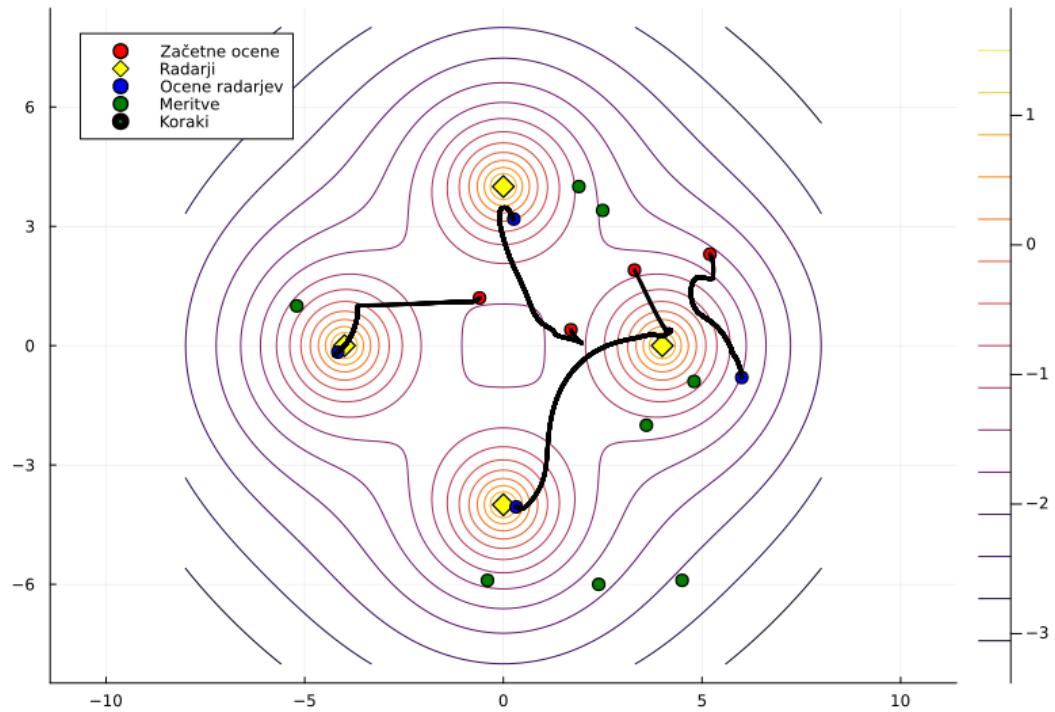
Slika 5: Simulacija 5

S prevelikim koeficientom α model ne konvergira. Naredimo še nekaj simulacij s enakimi pozicijami 4 radarjev, kjer so začetne ocene naključno izbrane, in 8 meritov je narejenih na naključnih lokacijah.



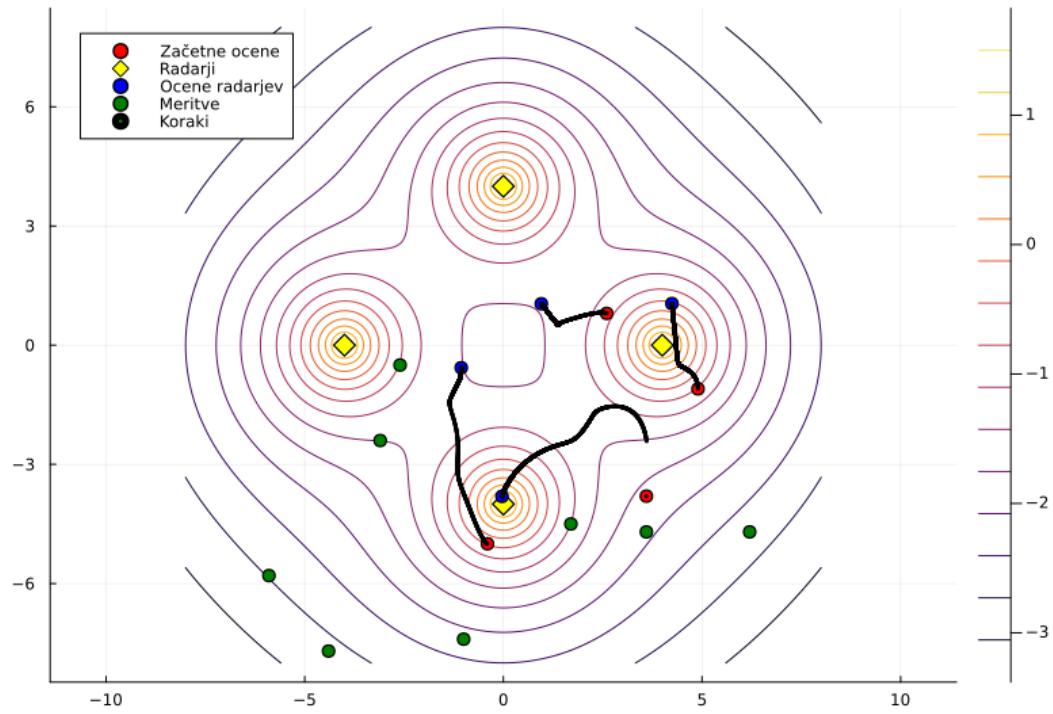
Število iteracij: 10000, Koeficient gradientne metode: 0.2

Slika 6: Simulacija 6



Število iteracij: 10000, Koeficient gradientne metode: 0.2

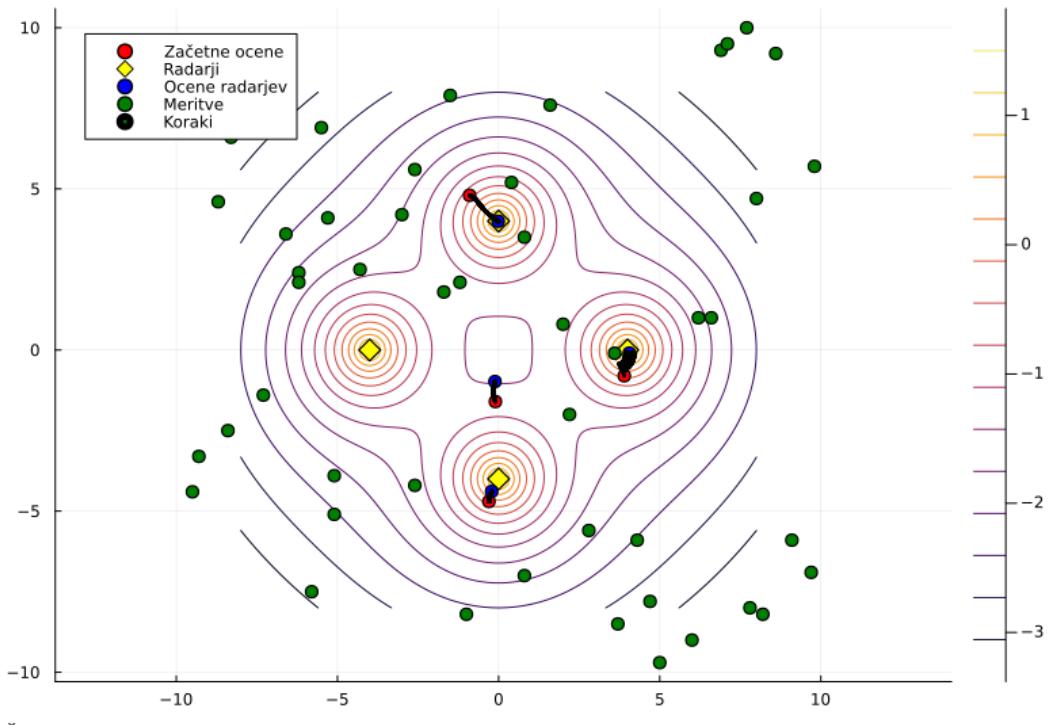
Slika 7: Simulacija 7



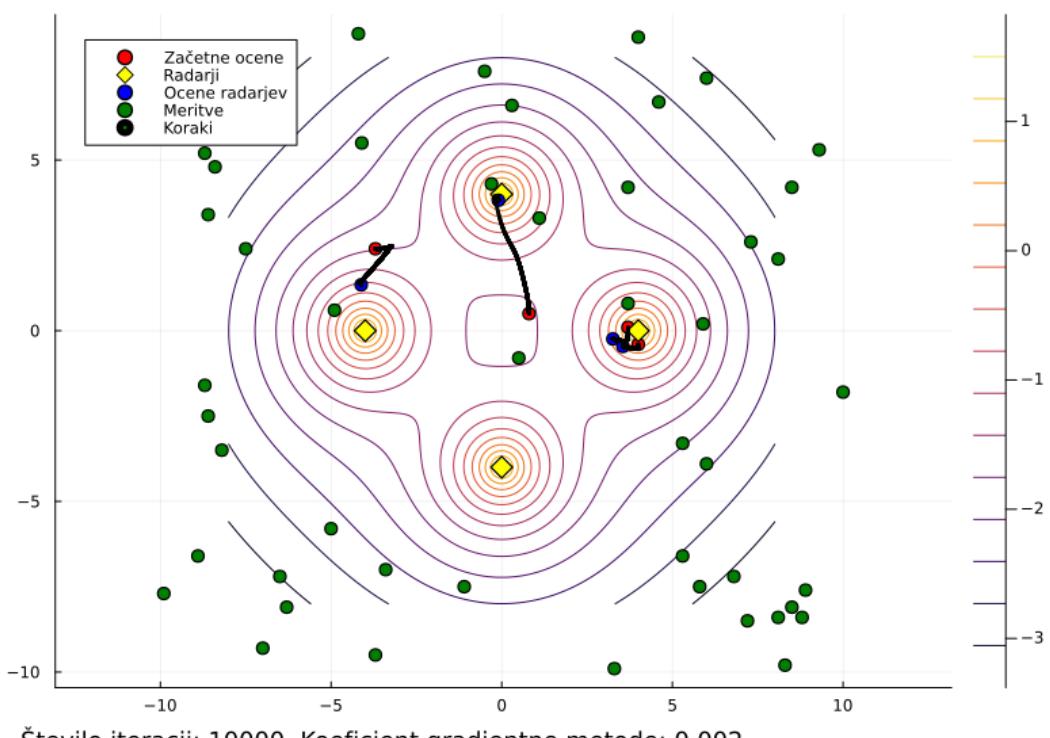
Število iteracij: 10000, Koeficient gradientne metode: 0.2

Slika 8: Simulacija 8

Z osmimi meritvami na naključnih lokacijah, težko napovemo prave lokacije 4 radarjev z naključnimi začetnimi ocenami. Pravilnost rešitve je tudi močno odvisna od začetnih ocen radarjev in lokacij meritev. Vse simulacije so tudi dosegle maksimalno število iteracij. Ali lahko s povečanjem števila meritev še izboljšamo napoved in zmanjšamo število iteracij? Vzemimo 50 naključnih meritev, lokacije meritve imajo x koordinato $x \in [-10, 10]$ in y koordinato $y \in [-10, 10]$.



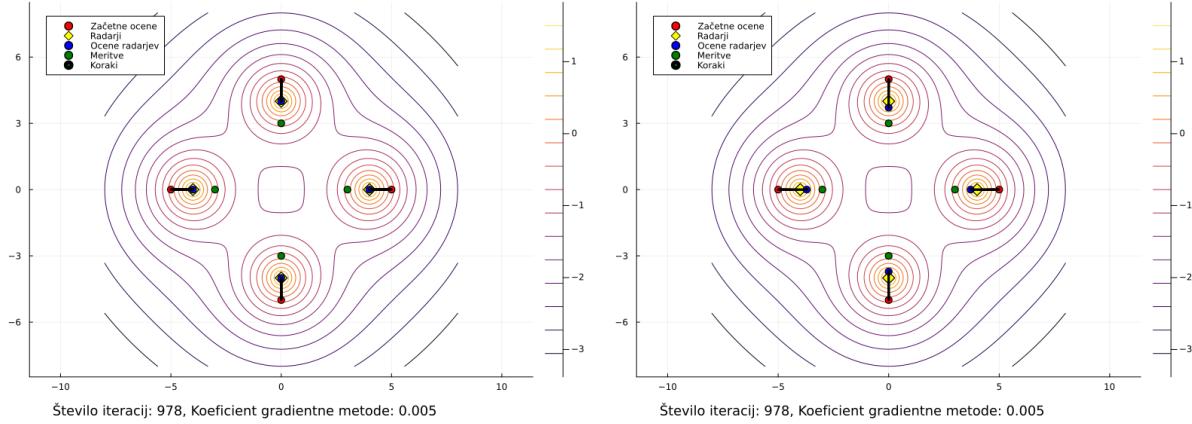
Slika 9: Simulacija 9



Slika 10: Simulacija 10

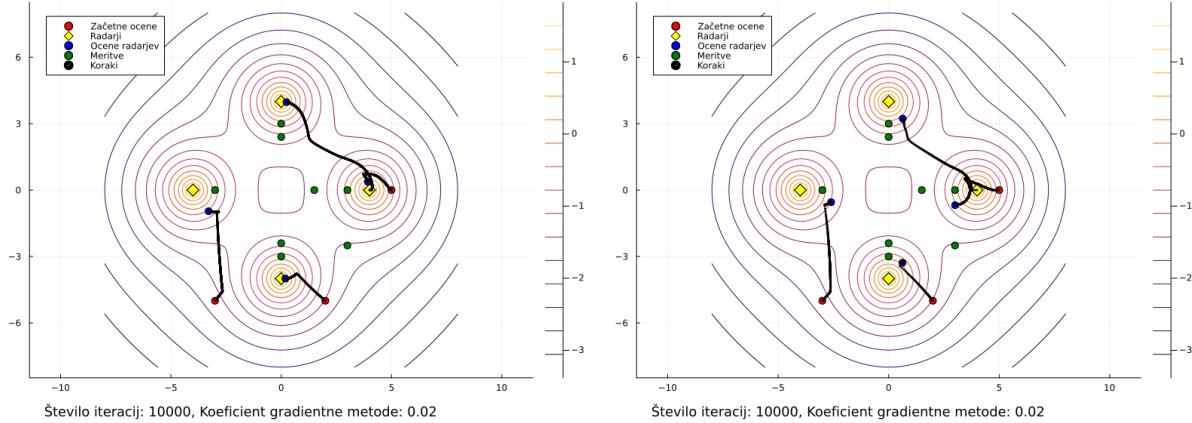
Povečanje števila meritev ne izboljša k boljši predikciji radarjev, prav tako gradientna metoda še vedno doseže maksimalno število iteracij. Pravilnost predikcije je močno odvisna od začetnih ocen radarjev in lokacij opravljenih meritev, povečanje števila opravljenih meritev pa ne vpliva na boljšo predikcijo.

Poglejmo, kako se vede model, če meritve jakosti signalov vsebujejo merske napake. Vsaki meritvi pristejemo ali odstejemo 10% njene vrednosti. Na začetku vzemimo meritve in začetne ocene, ki so blizu rešitve.



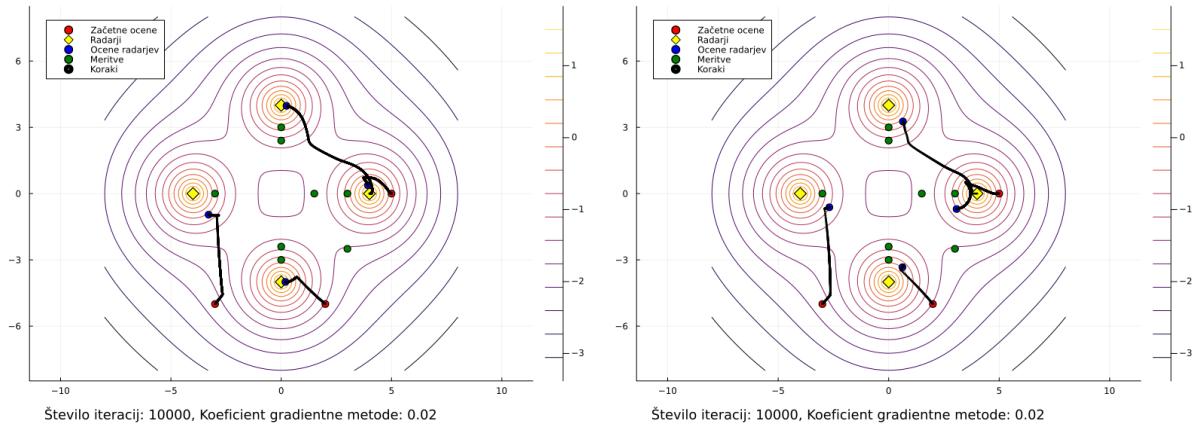
Slika 11: Simulacija brez šuma

Rezultat pri meritvah s šumom nekoliko odstopa od pravega, vendar ne veliko. Kako šum vpliva na meritve, ki niso blizu rešitve in so bolj naključno razporejene, tudi začetne ocene niso blizu rešitev.



Slika 13: Simulacija brez šuma

10% šum vpliva na končni rezultat, še vedno pa kljub napaki začetna ocena konvergira proti pravilni rešitvi. Za isti primer zmanjšamo šum na 2%, ker je zagotovo bolj verjetna merska napaka v realnosti.



Slika 15: Simulacija brez šuma

Tudi 2% šum še vedno vpliva na končni rezultat, začetne predikcije pa še vedno konvergirajo proti pravim lokacijam radarjev.

Slika 12: Simulacija s šumom

Slika 14: Simulacija s šumom

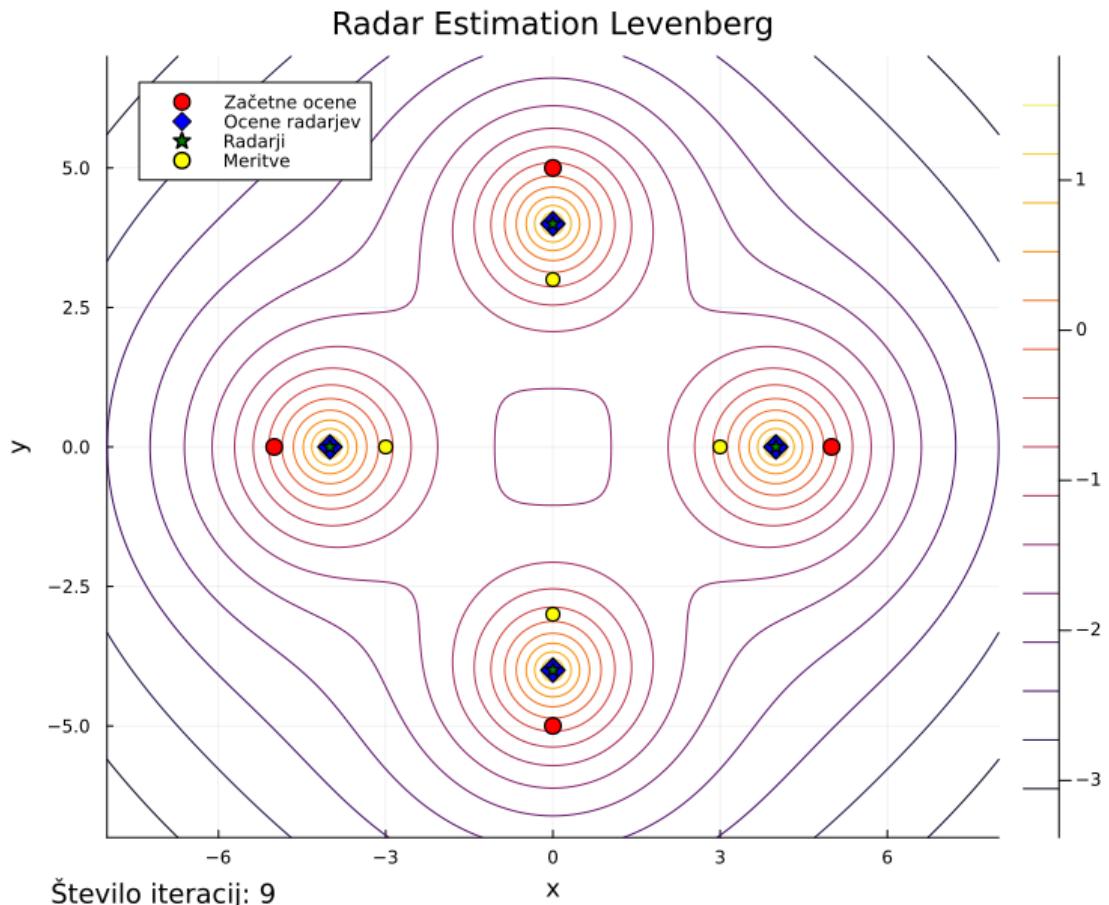
Slika 16: Simulacija s šumom

3.2 Simulacije z Levenbergovo metodo

Naj bodo 4 radarji na lokacijah s koordinatami $(-4, 0)$, $(0, 4)$, $(4, 0)$, $(0, -4)$.

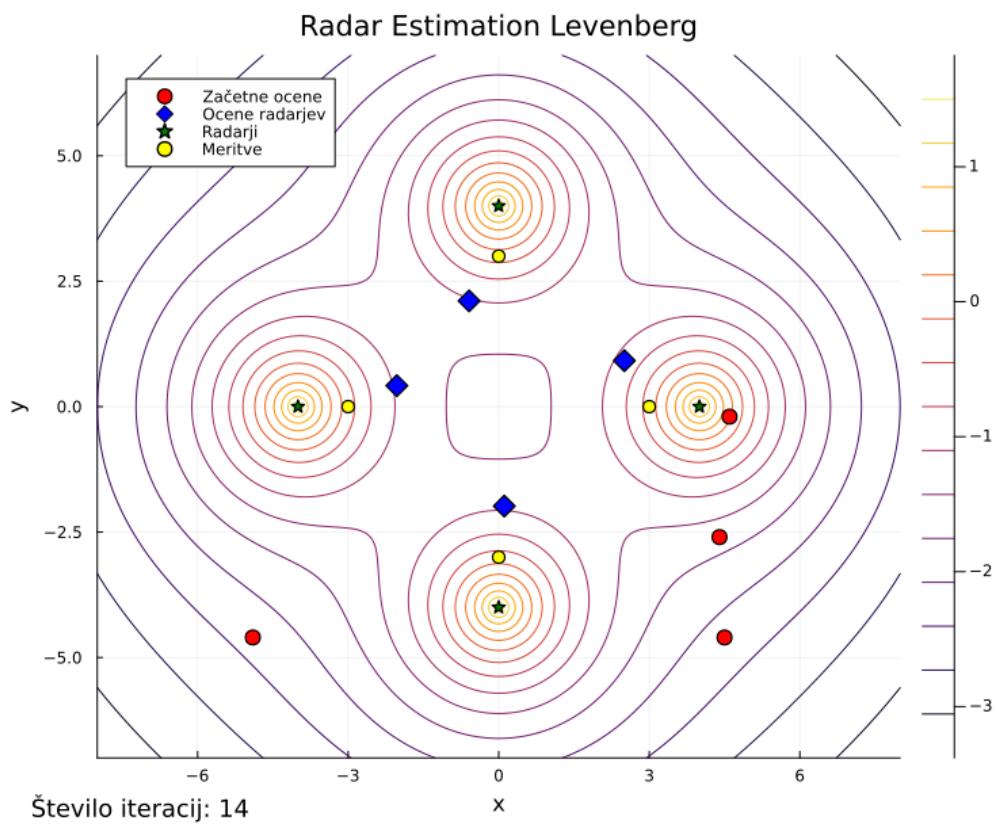
Maksimalno število iteracij za Levenbergovo metodo je omejeno na 10000.

Preverimo, če model deluje na enostavnem testnem primeru, kjer so začetne ocene na lokacijah $(-5, 0)$, $(0, 5)$, $(5, 0)$, $(0, -5)$, 4 meritve pa so na lokacijah $(-3, 0)$, $(0, 3)$, $(3, 0)$, $(0, -3)$.



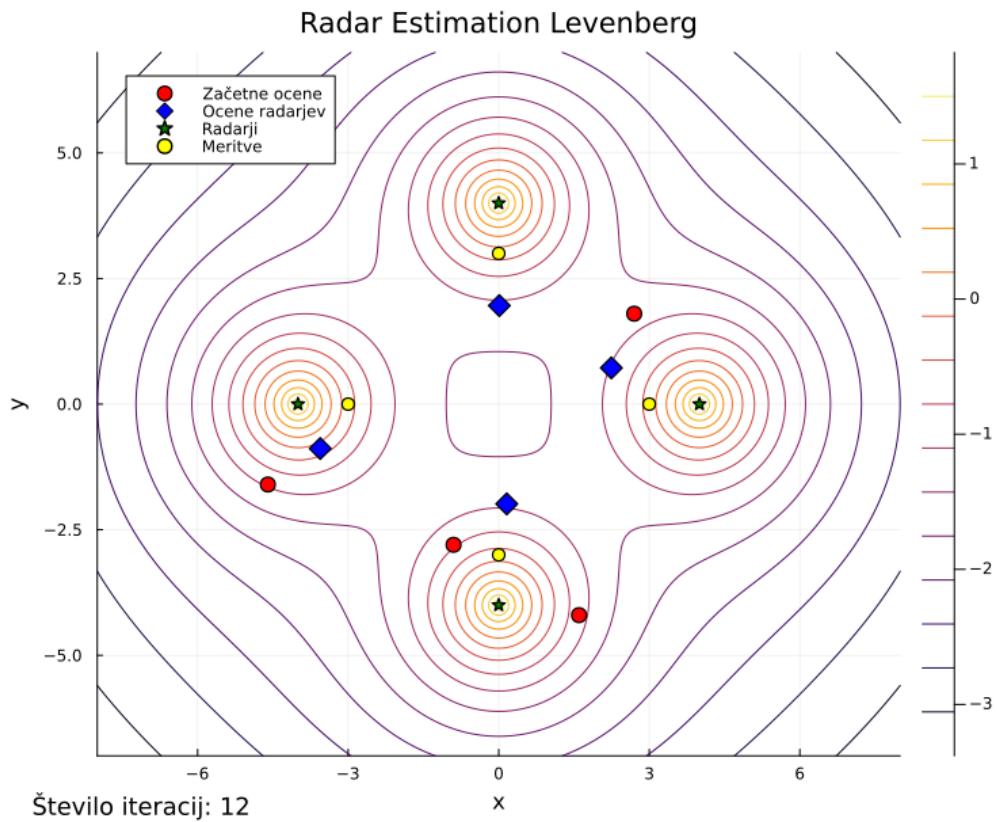
Slika 17: Prvi testni primer

Vidimo, da model konvergira proti pravilni rešitvi. Spremenimo začetne ocene na bolj naključne lokacije, ki niso nujno blizu prave rešitve. Meritve ostanejo enake.



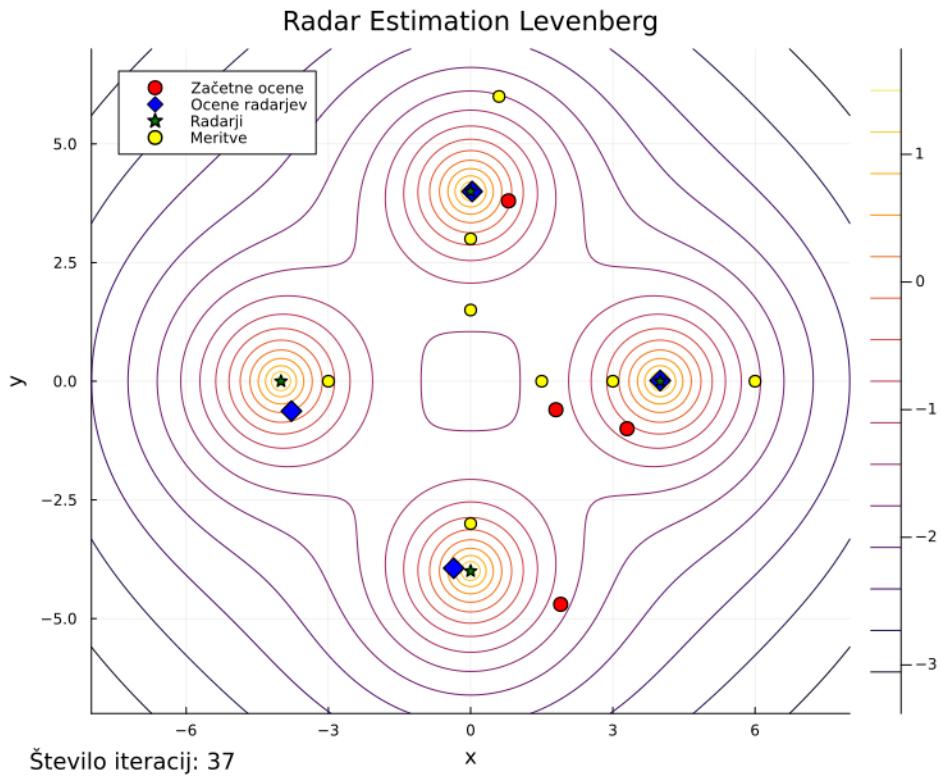
Slika 18: Simulacija 18

Pravi rešitvi se ne približamo dobro. Poskusimo še enkrat.



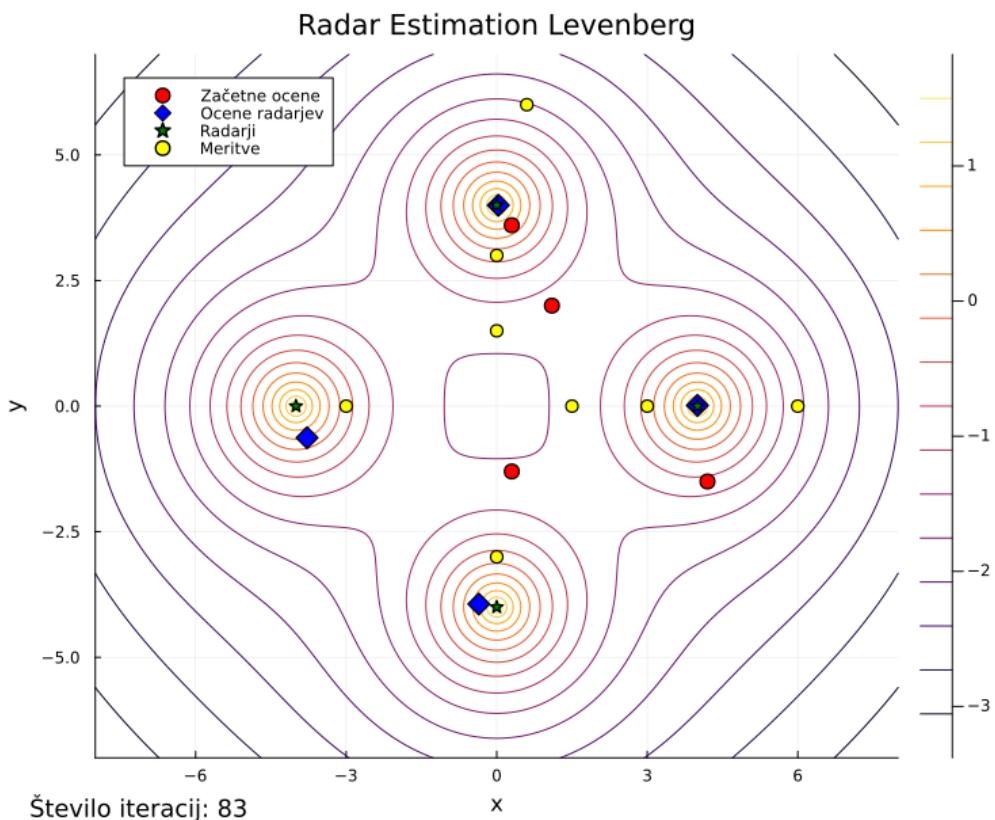
Slika 19: Simulacija 19

Poskusimo povečati število meritev na 8. Začetne ocene so naključno izbrane.



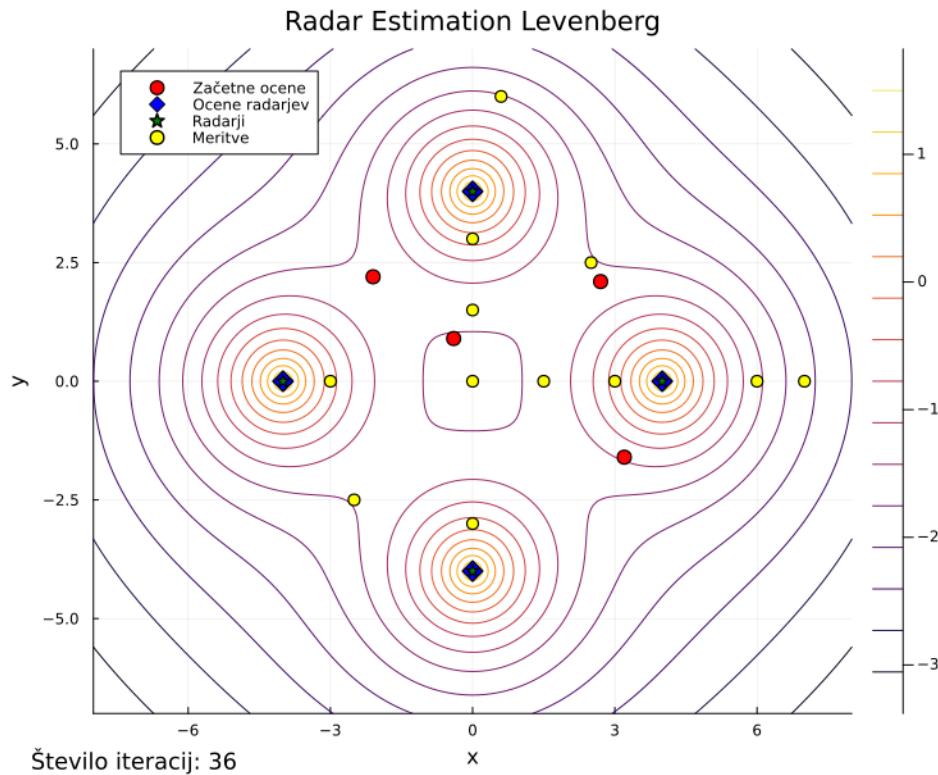
Slika 20: Simulacija 20

Očitno metoda pri večjih številih meritvah bolje deluje. Poskusimo še za en primer.



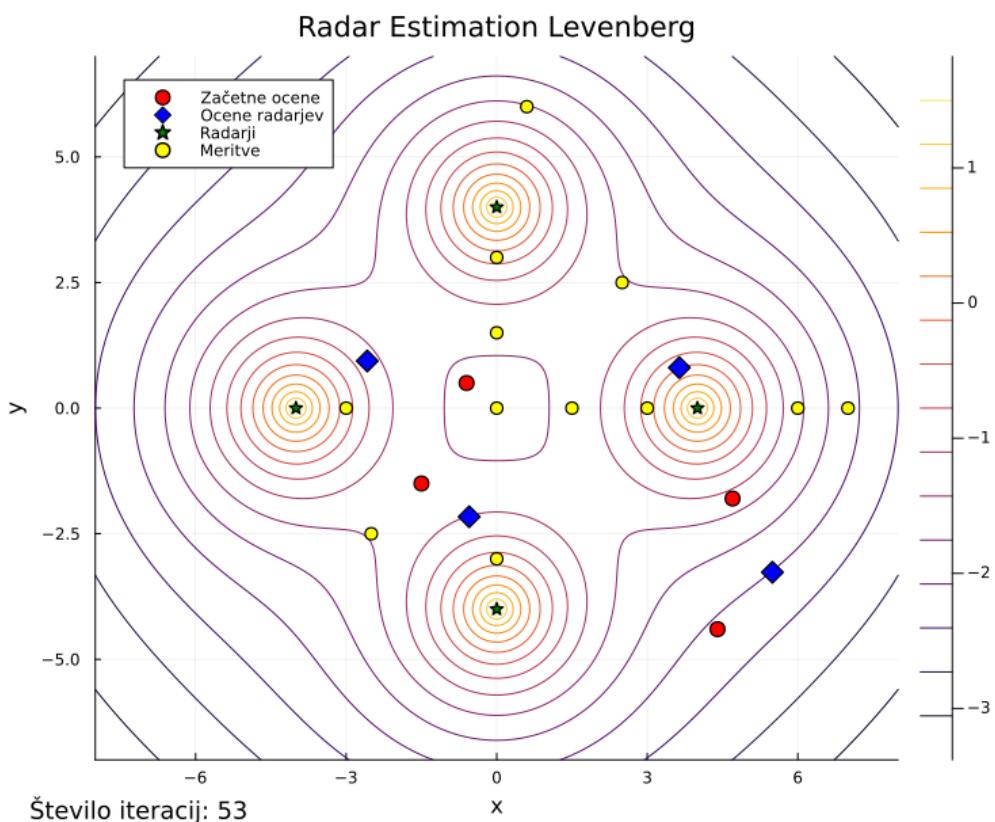
Slika 21: Simulacija 21

Uganemo skoraj vse radarje. Poskusimo povečati število meritev na 12.



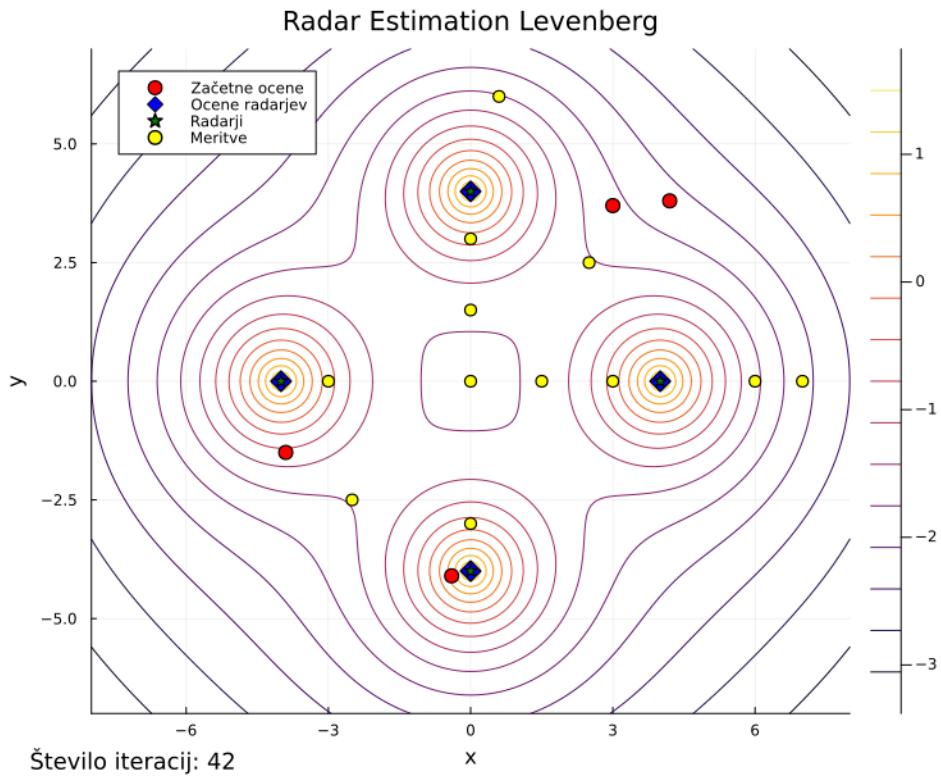
Slika 22: Simulacija 22

Uganemo vse. Poskusimo še za en primer.



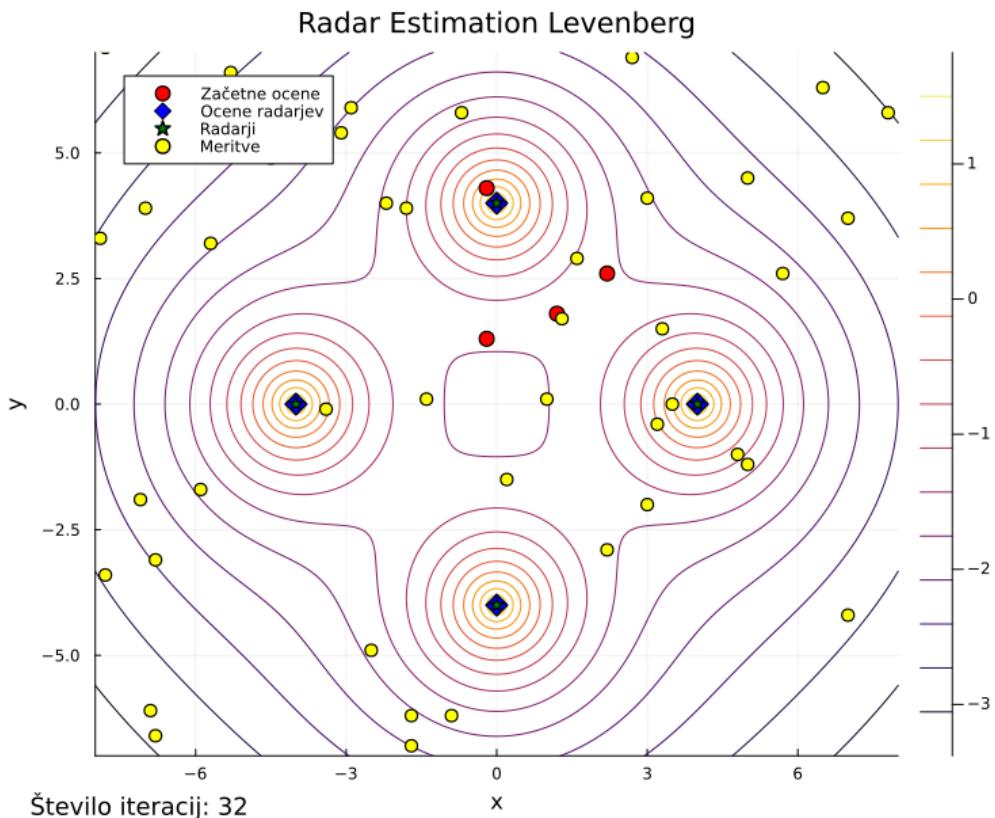
Slika 23: Simulacija 23

Preseneti nas to, da zgrešimo vse štiri radarje. Poskusimo še enkrat.

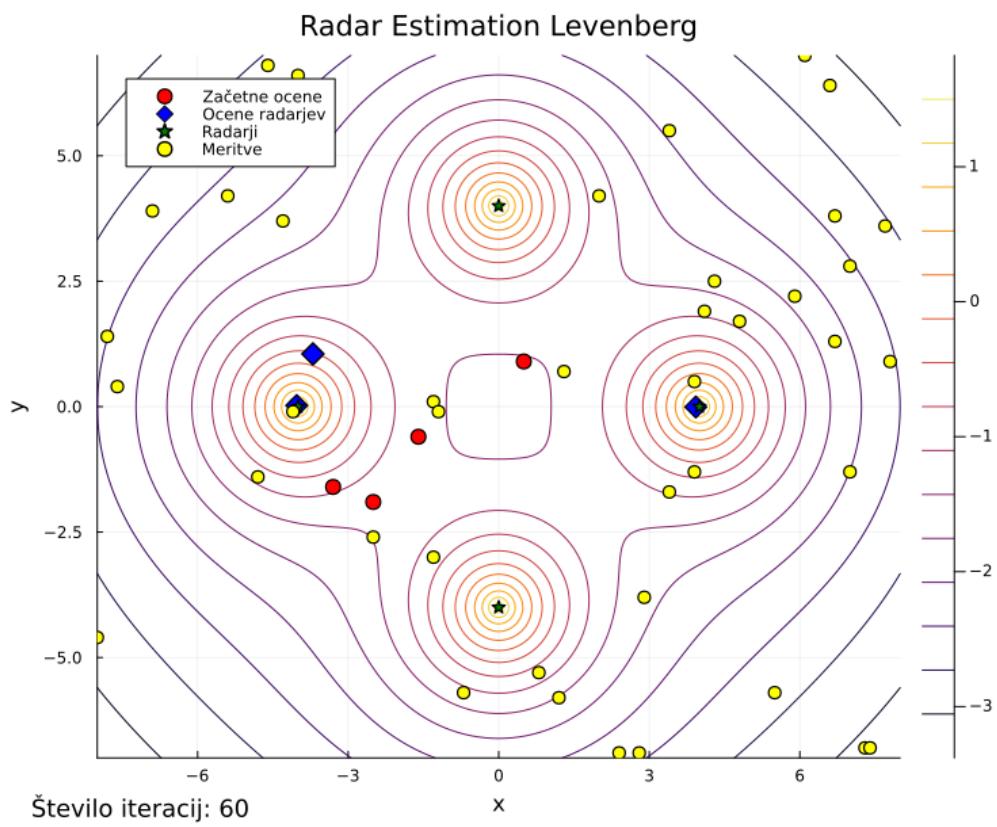


Slika 24: Simulacija 24

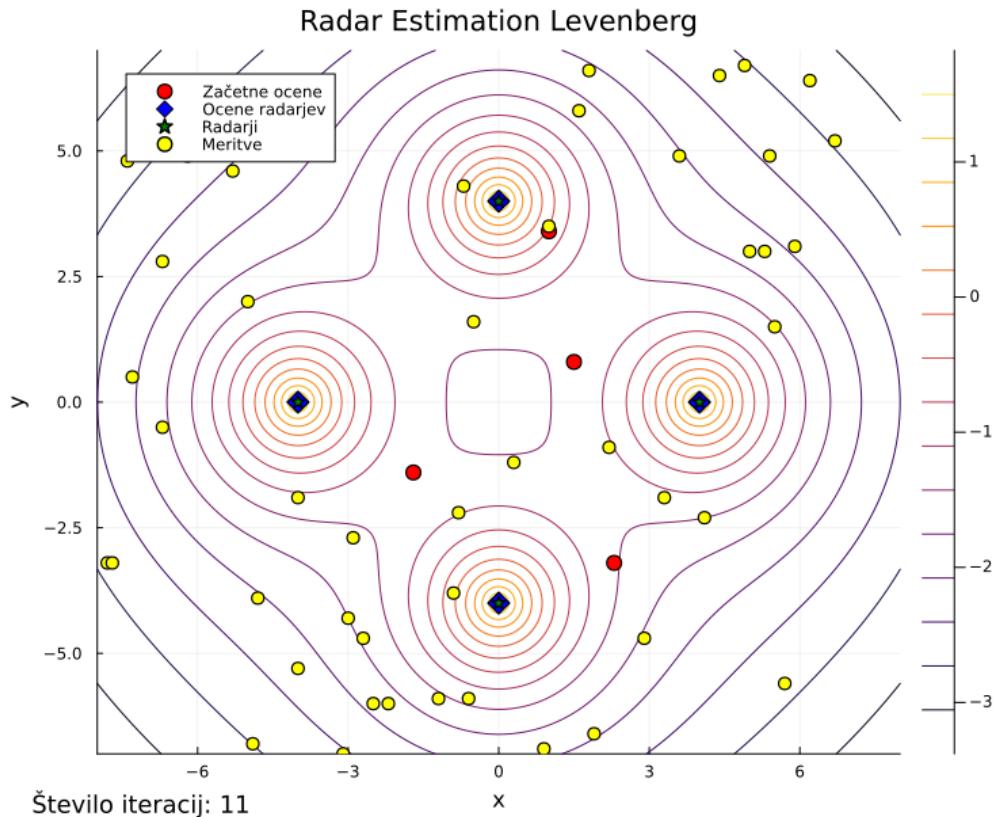
Opazimo, da metoda ni konsistentna pri 12 meritvah. Povečajmo število meritev na 50.



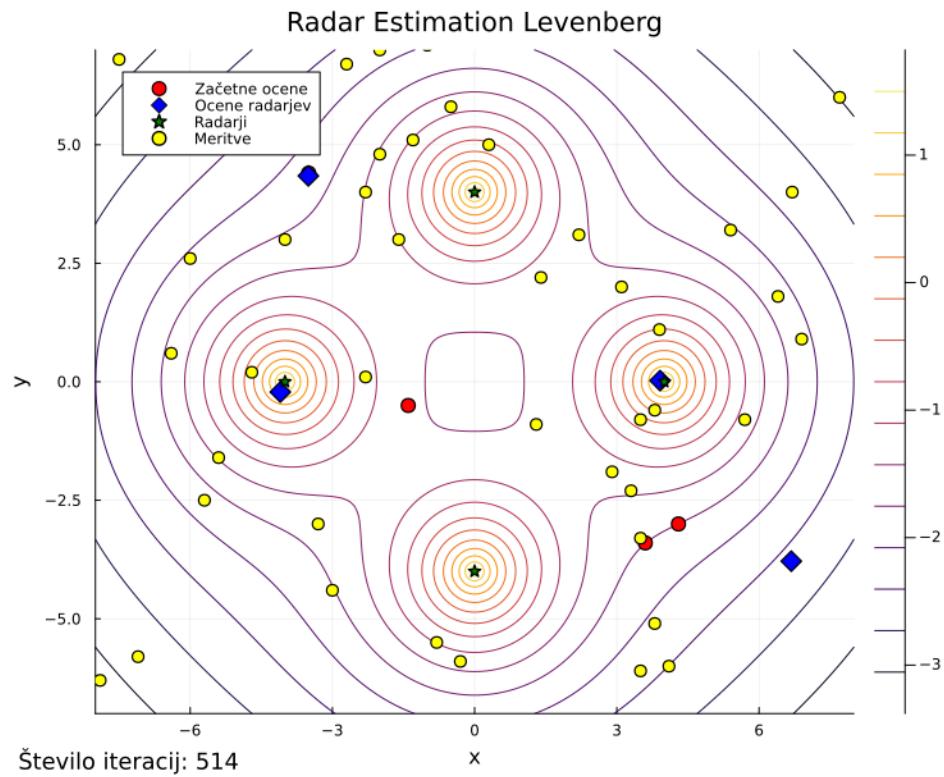
Slika 25: Simulacija 25



Slika 26: Simulacija 26

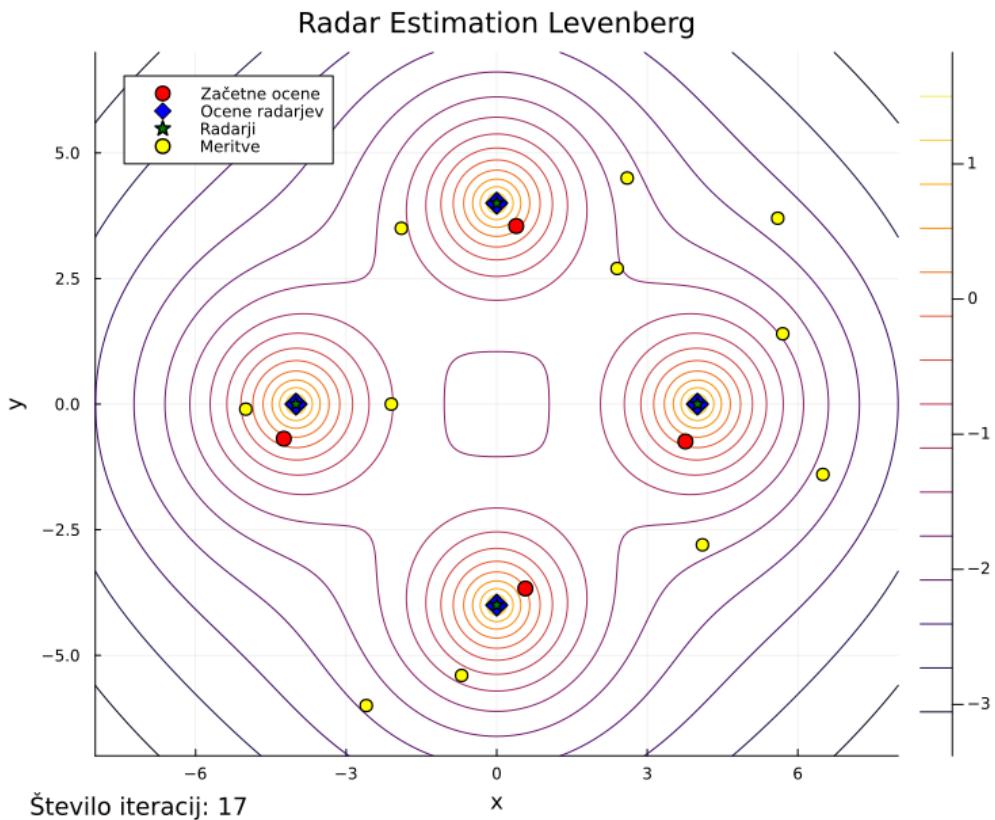


Slika 27: Simulacija 27

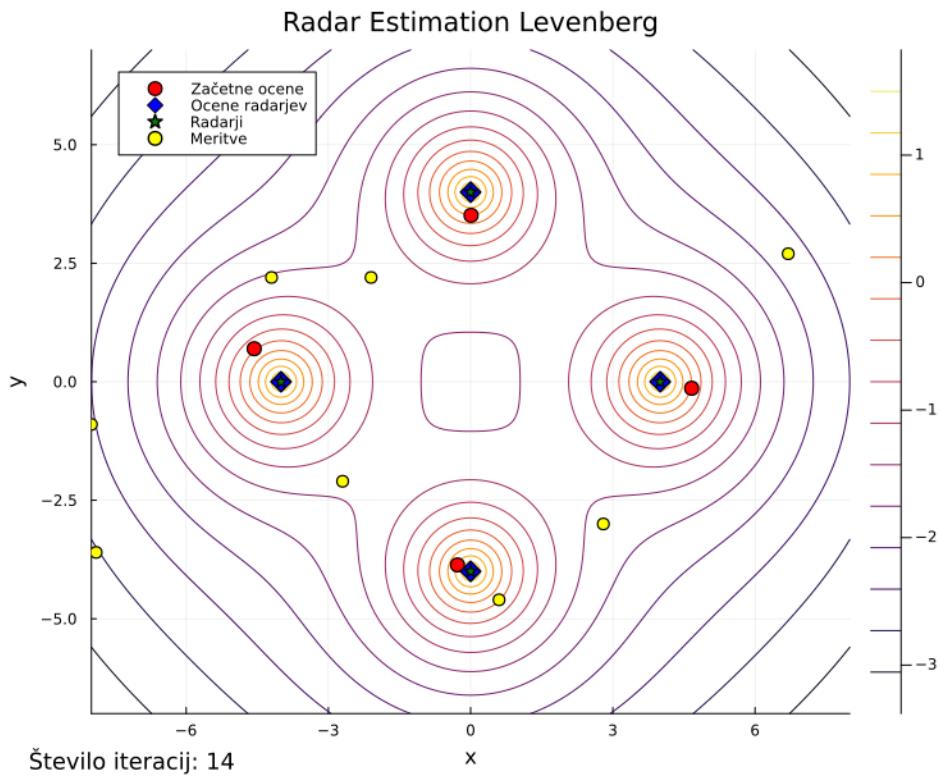


Slika 28: Simulacija 28

Število meritev očitno ne izboljša k boljši predikciji radarjev. Poglejmo, če začetna ocena igra pomembnejšo vlogo. Izberemo začetne ocene, ki so blizu radarjev. Meritev naj bo samo 12.



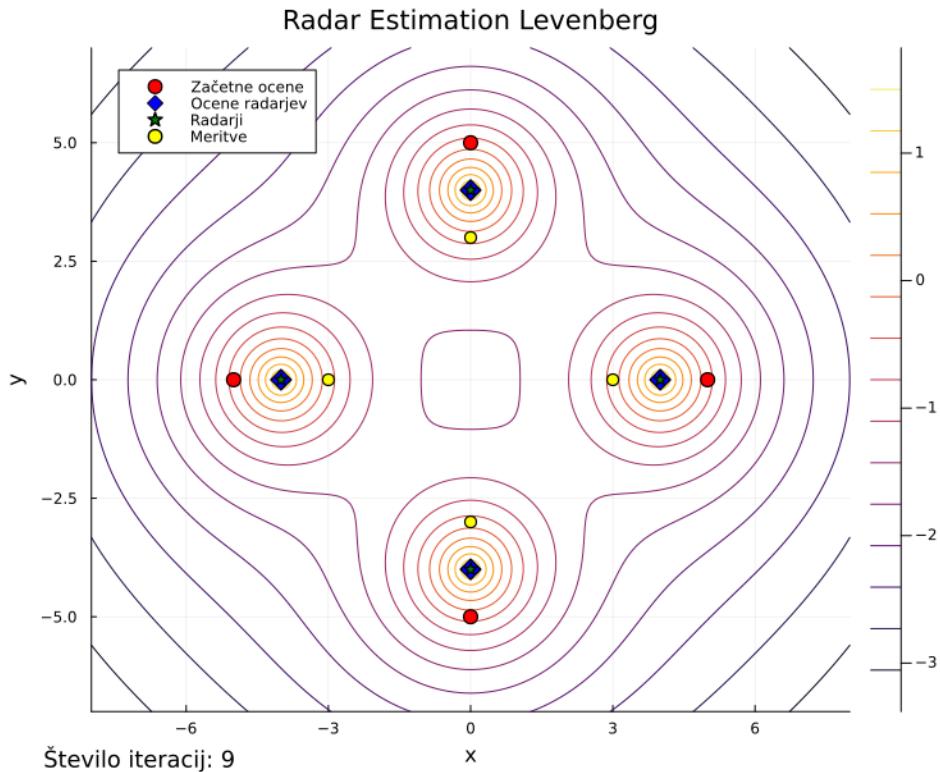
Slika 29: Simulacija 29



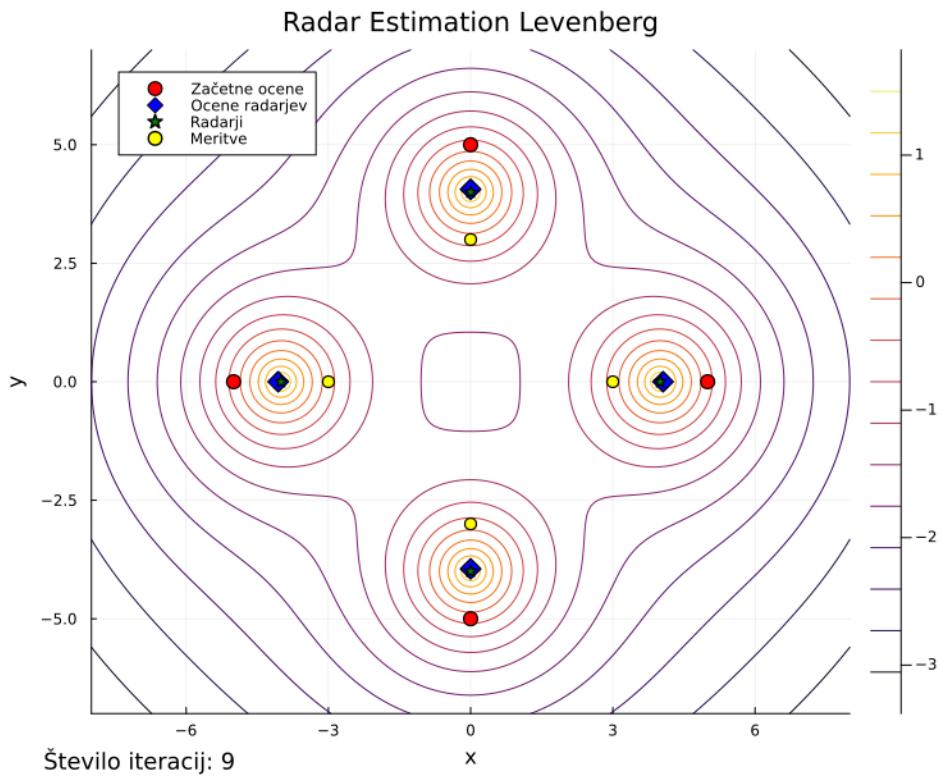
Slika 30: Simulacija 30

Pravilnost predikcije je močno odvisna od začetnih ocen radarjev.

Poglejmo, kako se vede model, če meritve jakosti signalov vsebujejo merske napake. Vsaki meritvi prištejemo ali odštejemo 10% njene vrednosti. Na začetku vzemimo meritve in začetne ocene, ki so blizu rešitve.

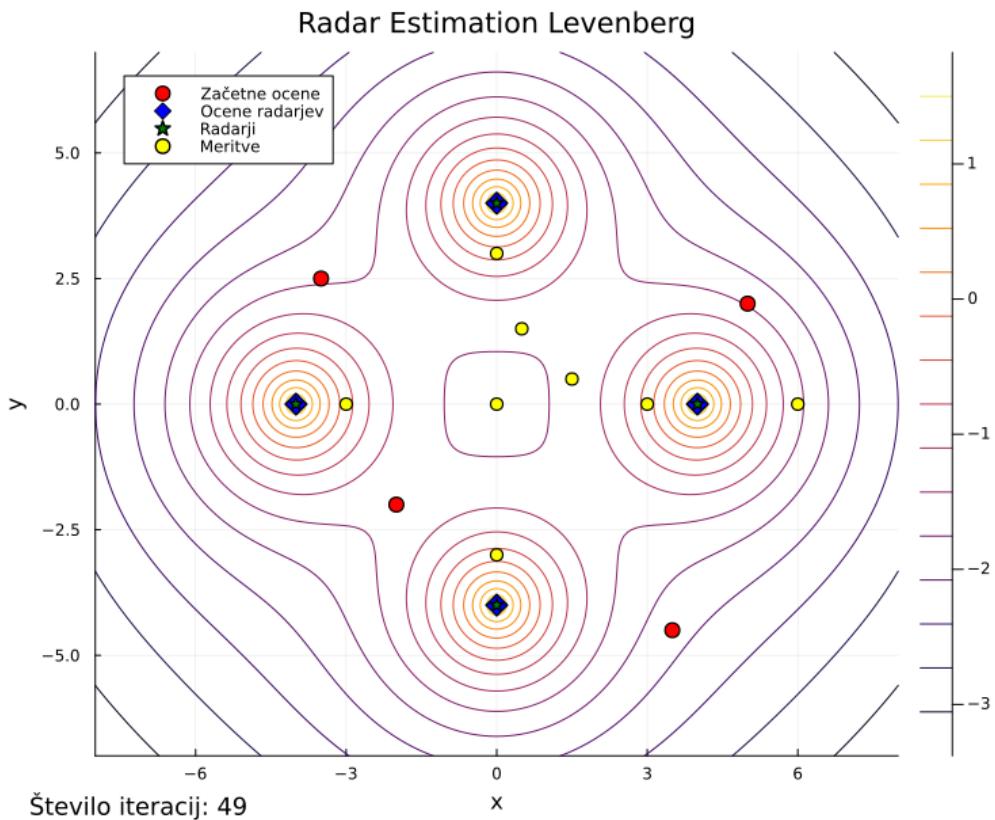


Slika 31: Simulacija brez šuma

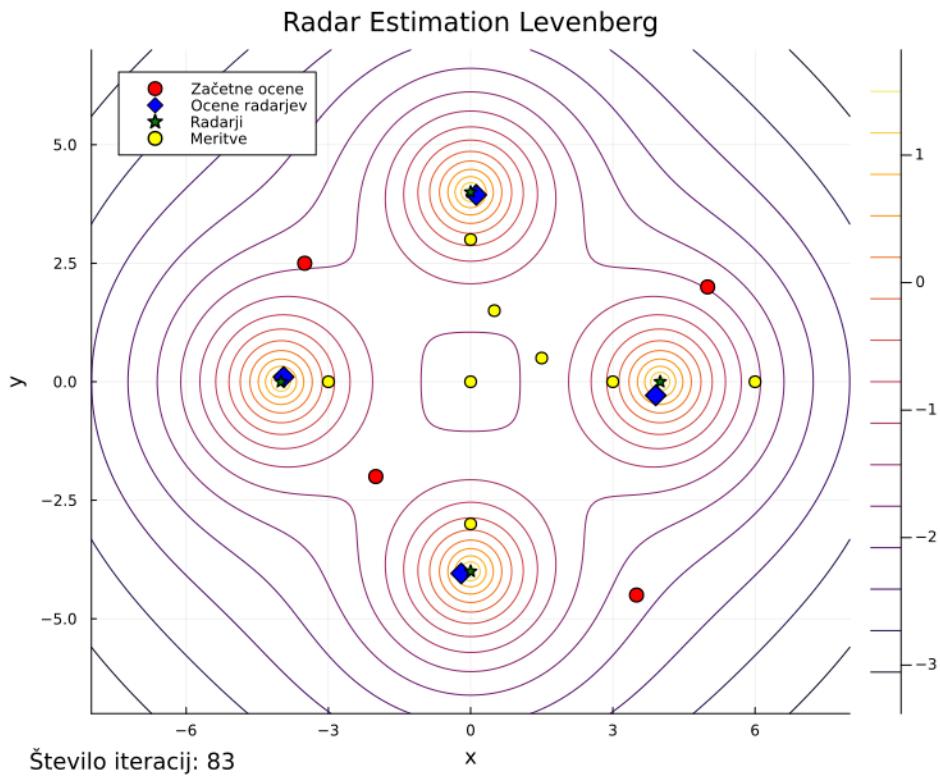


Slika 32: Simulacija s šumom

Rezultat pri meritvah s šumom nekoliko odstopa od pravega, vendar ne veliko. Kako šum vpliva na meritve, ki niso blizu rešitve in so bolj naključno razporejene, tudi začetne ocene niso blizu rešitev.

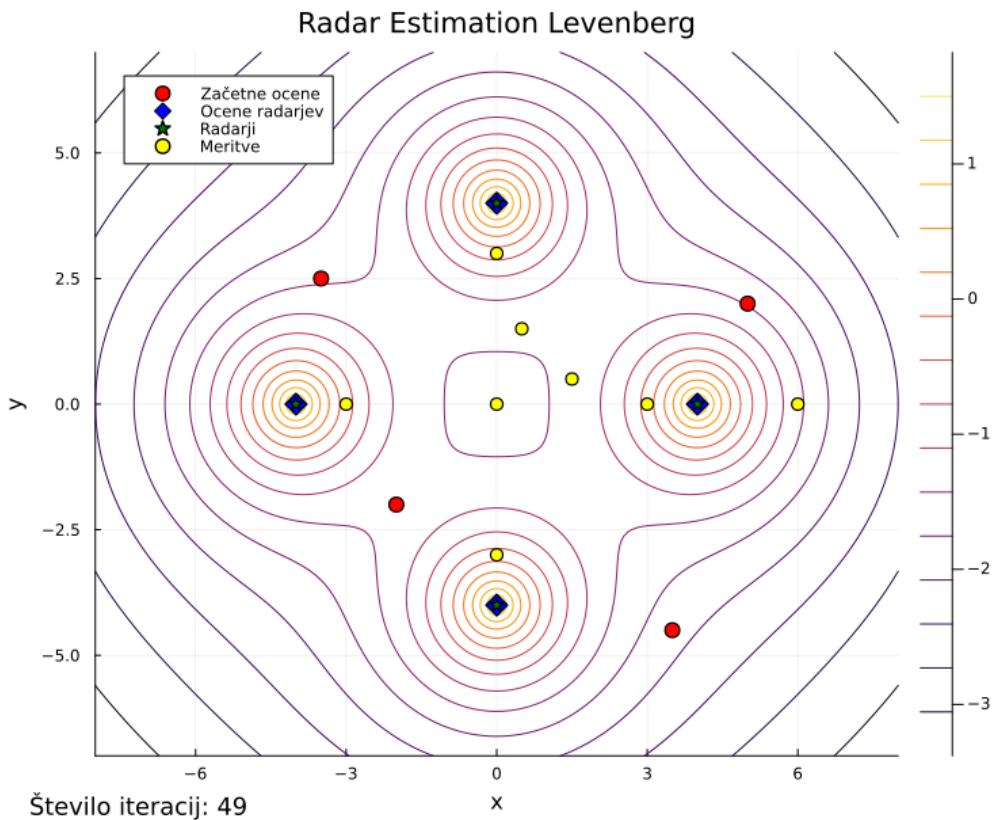


Slika 33: Simulacija brez šuma

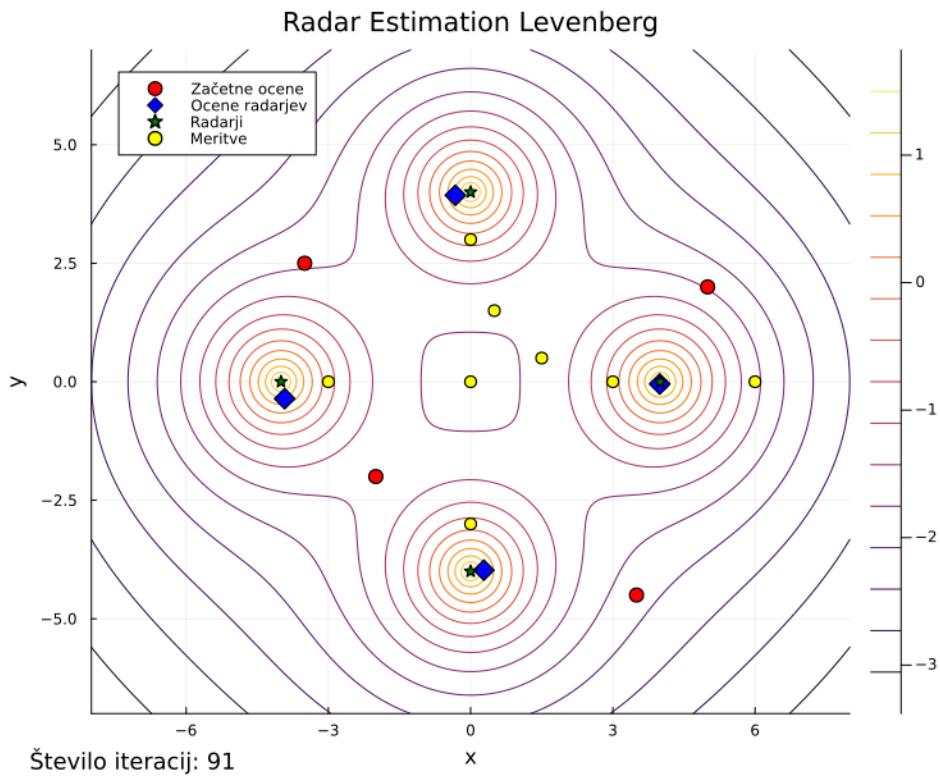


Slika 34: Simulacija s šumom

10% šum vpliva na končni rezultat, še vedno pa kljub napaki začetna ocena konvergira proti pravilni rešitvi. Za isti primer zmanjšamo šum na 2%, ker je zagotovo bolj verjetna merska napaka v realnosti.



Slika 35: Simulacija brez šuma

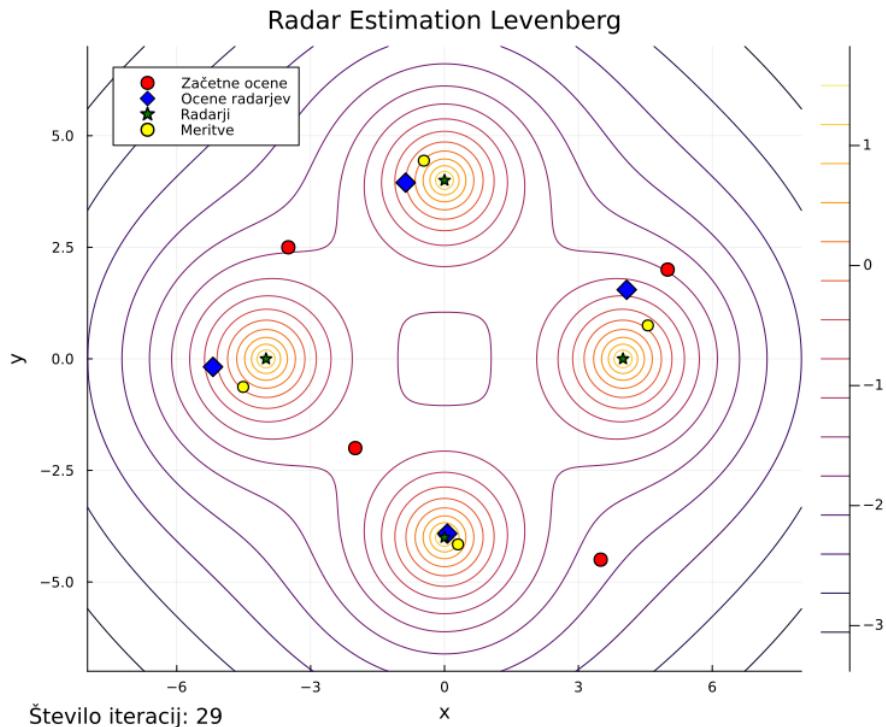


Slika 36: Simulacija s šumom

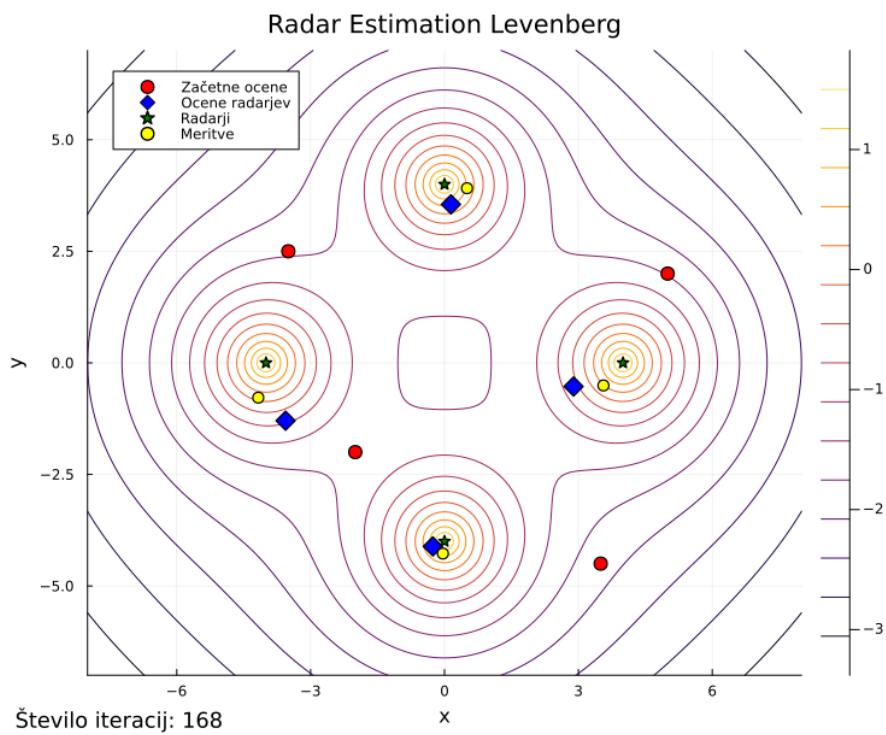
Tudi 2% šum še vedno vpliva na končni rezultat, začetne predikcije pa še vedno konvergirajo proti pravim lokacijam radarjev.

Poglejmo še vpliv povprečnih oddaljenosti meritev od radarskih postaj na natančnost rezultatov. Začetne ocene so konstante. Meritev je 4.

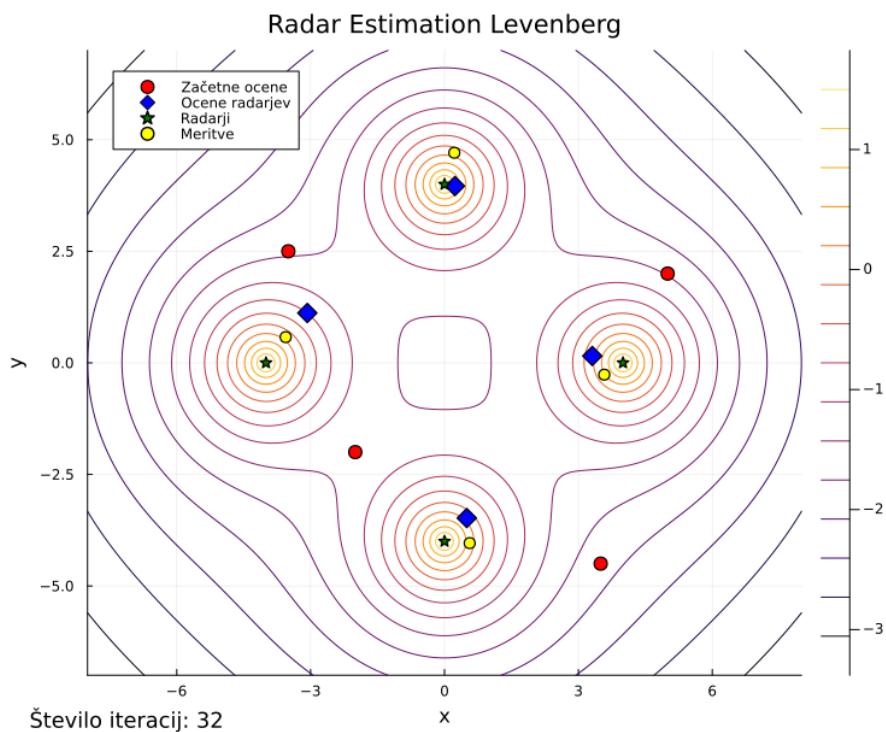
1. Primer: Meritev so blizu radarjev.



Slika 37: Simulacija 37



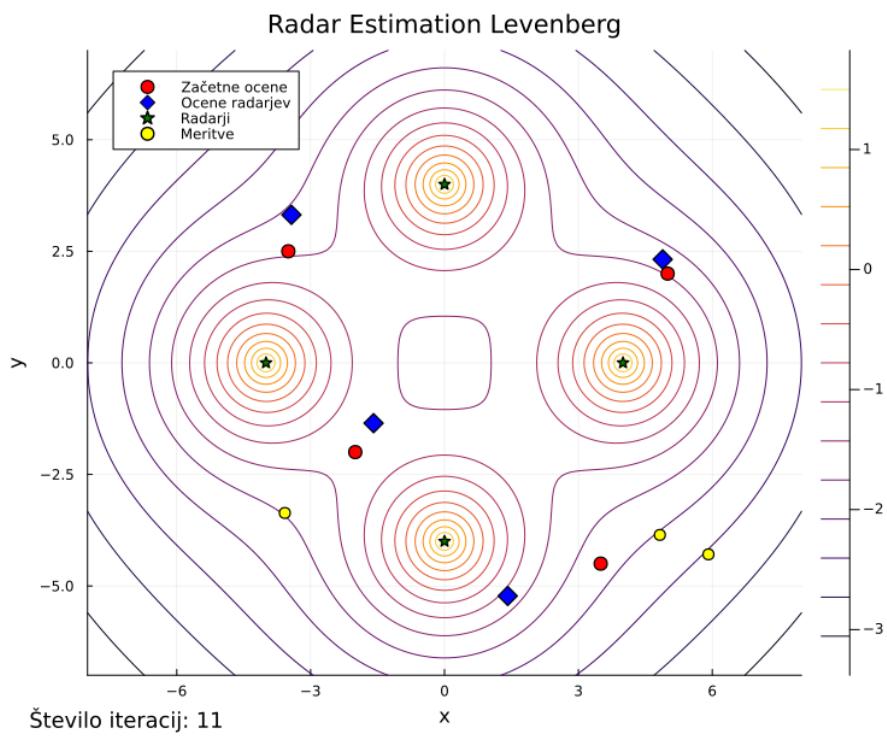
Slika 38: Simulacija 38



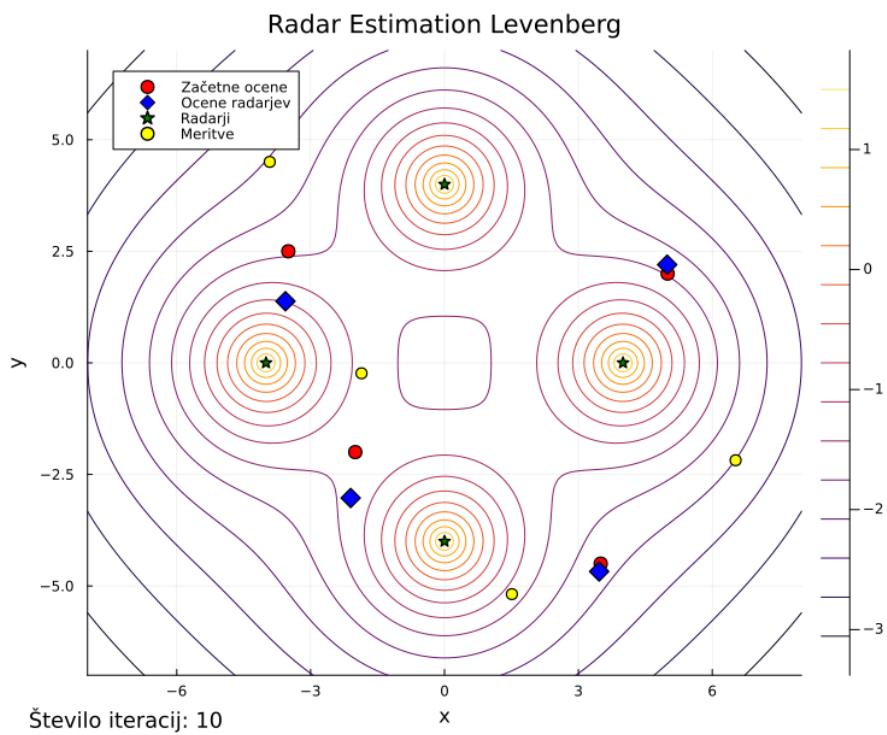
Slika 39: Simulacija 39

Ocene radarjev so dokaj dobre.

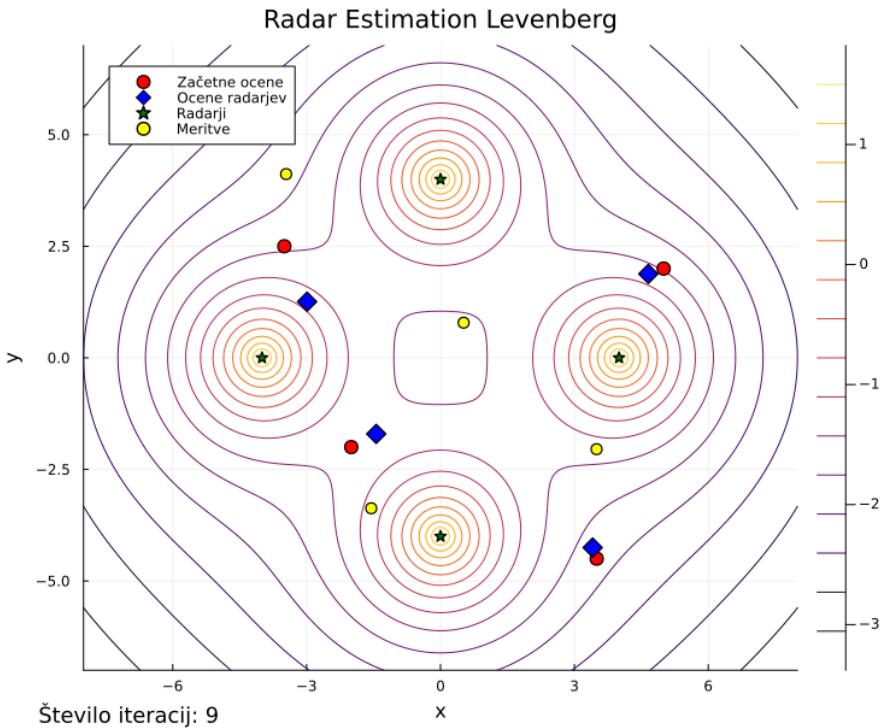
2. Primer: Meritve so oddaljene od radarjev.



Slika 40: Simulacija 40



Slika 41: Simulacija 41



Slika 42: Simulacija 42

Opazimo, da se ocene radarjev približujejo začetnim ocenam.

4 Zaključek

V splošnem je napoved položajev radarjev na podlagi meritov njihovih jakosti težka tako z gradientno metodo kot tudi z Levenbergovo metodo. Pri obeh metodah je pravilnost napovedi močno odvisna od začetnih približkov položajev radarjev. Obe metodi se obneseta, če so začetni približki blizu pravilnih položajev radarjev, takrat z obema metodama približki konvergirajo proti pravilni rešitvi. Če imamo opravljenih malo meritov, potem model najbolje konvergira k pravilni rešitvi, če so tudi meritve opravljene blizu položajev radarjev. Tudi večanje števila meritov ne pripomore nujno k bolj optimalni rešitvi, še vedno je vse odvisno od začetnih približkov, kar dobro prikazujeta simulacija 22 in simulacija 23. Približki so lahko tudi precej daleč od radarjev, pa vseeno konvergirajo k pravilni rešitvi, dober primer sta simulacija 25 in simulacija 3.

Obe metodi imata vključeni konstanti. Pri gradientni metodi je pomembna izbira konstante α , saj tudi če izberemo ustrezne začetne približke, s premajno vrednostjo model ne konvergira k optimalni rešitvi, prevelika konstanta pa model pokvari. S konstanto λ pri Levenbergovi metodi se nisva preveč ukvarjala, uporabila sva nastavitev, ki so priporočene v knjigi [3]. Tukaj je verjetno še možnost za izboljšavo, prednost Levenbergove metode naj bi bila prav v tem, da deluje dobro tudi s slabimi začetnimi približki.

Odzivi obeh metod na šum v meritvah je presenetljiv, šum vpliva na končni rezultat, še vedno pa približki konvergirajo k optimalni rešitvi. Razlika med rezultati, ki imajo šum v meritvah, in tistimi brez šuma, je zelo majhna. Uporabila sva 2% in 10% šum, kar pomeni, da sva vsaki meritvi prištela ali odštela 2% oz. 10% njene vrednosti. Simulacija večjih šumov bi bila zanimiva, a verjetno nesmiselna, saj takih napak verjetno v realnosti ne bi bilo.

Pri vseh simulacijah so bili radarji postavljeni na istih pozicijah. Spreminjanje le-teh v drugačne formacije bi spet odprlo ogromno vprašanj. Na podlagi dosedanjih ugotovitev pa bi bili verjetno pri predikciji spet najbolj uspešni, če smo že pri izbiri začetnih približkov karseda blizu pravih lokacij.

5 Viri

Literatura

- [1] Wikipedia contributors. *Levenberg–Marquardt algorithm*. URL: https://en.wikipedia.org/wiki/Levenberg–Marquardt_algorithm. (accessed: 21.04.2025).
- [2] Tobin A. Driscoll in Richard J. Braun. *Fundamentals of Numerical Computation, Julia edition*. URL: <https://tobydriscoll.net/fnc-julia/nonlineqn/newtonsys.html>. (accessed: 21.04.2025).
- [3] Tobin A. Driscoll in Richard J. Braun. *Fundamentals of Numerical Computation, Julia edition*. URL: <https://tobydriscoll.net/fnc-julia/nonlineqn/quasineutron.html>. (accessed: 21.04.2025).
- [4] The Julia Project. *The Julia Language*. URL: <https://docs.julialang.org/en/v1/>. (accessed: 21.04.2025).
- [5] Neža Mramor Kosta. Polona Oblak. Žiga Virk. Aljaž Zalar. *Mathematical modelling*. Lecture notes. 2024/25.