1 DN1: SOR iteracija za razpršene matrike

Avtor: Timotej Šalamon

V nalogi se ukvarjamo s SOR iteracijo na posebnem tipu matrik - razpršene matrike. Matrika zaradi prostorskih zahtev hrani vrednosti v matrikah `V` in `I`, velikosti n x m. Pri tem velja:

```
V(i,j) = A(i, I(i, j)).
```

```
using Domaca01, LinearAlgebra
```

Definiramo nov tip - Razpršeno matriko A.

```
V = [[1, 2], [3, 4, 5]]
I = [[1, 2], [1, 2, 3]]
A = RazprsenaMatrika(V, I)
```

```
Domaca01.RazprsenaMatrika([[1, 2], [3, 4, 5]], [[1, 2], [1, 2, 3]])
```

Ker gre za poseben tip matrike moramo uvesti nove funkcije za naslenje funkcionalnosti:

Funkcija getindex

```
ix = getIndex(A, 1, 2)
```

```
2
```

Funkcija setindex

```
setindex!(A, 5, 1, 2)
A
```

```
Domaca01.RazprsenaMatrika([[1, 5], [3, 4, 5]], [[1, 2], [1, 2, 3]])
```

Funkcija firstindex

```
first = firstindex(A, 1)
```

```
1
```

Funkcija lastindex

```
last = lastIndex(A, 2)
```

```
3
```

1.1 SOR iteracija

Sedaj lahko izvedemo Successive over-relaxation oz. SOR iteracijo. SOR iteracija se uporablja predvsem za reševanje sistemov `Ax = b`. Pri tem upošteva tudi relaksacijski vektor ω, ki je po navadi v mejah med 0 in 2. Iteracija deluje po naslednji formuli:

x*i^(k+1) = (1 - omega) * x*i^(k) + omega / A*(i, i) * (b*i - sum*((j<i) A*(ij) * x*j^(k+1)) - sum*(j>i) A*(ij) * x*j^(k))

Iteracija se konča ko dosežemo konvergenco po naslednjem kriteriju, kjer za toleranco vzamemo zelo majhno vrednost (npr. 1e-10): `\|Ax^{(k)} - b\|_\infty < \text{toleranca}` Računamo SOR iteracijo za razpršeno matriko A in vektor b za omega 1.25. x0 predstavlja začetni približek.

```
V = [[2.0], [1.0, 1.0], [3.0]]
I = [[1], [2, 3], [3]]
A = RazprsenaMatrika(V, I)

b = [1.0, 4.0, 2.0]
x0 = [0.0, 0.0, 0.0]
omega = 1.25

x, iter = SOR(A, b, x0, omega)
```

```
([0.49999999999954525, 3.3333333332696684, 0.6666666666660603], 20)
```

Izvedemo vlaganje grafa na ravnino / prostor s fizikalno metodo. Če so `(x_i,y_i,z_i)` koordinate vozlišč grafa v prostoru, potem vsaka koordinata posebej zadošča enačbam

```
-st(i) x[i] + sum_(j \in N(i)) x[j] = 0
-st(i) y[i] + sum_(j \in N(i)) y[j] = 0
-st(i) z[i] + sum_(j \in N(i)) z[j] = 0
```

V moji rešitvi sem izvedel vlaganje treh grafov, kjer so grafi podani z vozlišči, robovi in njihovimi vrednosti. Nato sem uporabil funkcijo `ustvariRazsprsenoMatriko`, ki iz teh podatkov ustvari razpršeno matriko.

Primer 1

```
vozlisca = [1, 2, 3]
robovi = [(1, 1), (2, 2), (2, 3), (3, 3)]
vrednosti = [2.0, 1.0, 1.0, 3.0]
A = ustvariRazsprsenoMatriko(vozlisca, robovi, vrednosti)

b = [1.0, 4.0, 2.0]
x0 = [0.0, 0.0, 0.0]

plot, minIteracij, minOmega, rez = optimalnaOmega(A, b, x0, 1)
```

```
(Plot{Plots.GRBackend() n=1}, 10.0, 0.9473684210526315, [0.4999999999999184
5, 3.3333333333523623, 0.6666666666665579])
```

Optimalna vrednost omege

```
minOmega
```

```
0.9473684210526315
```

Minimalno število iteracij

```
minIteracij
```

```
10.0
```

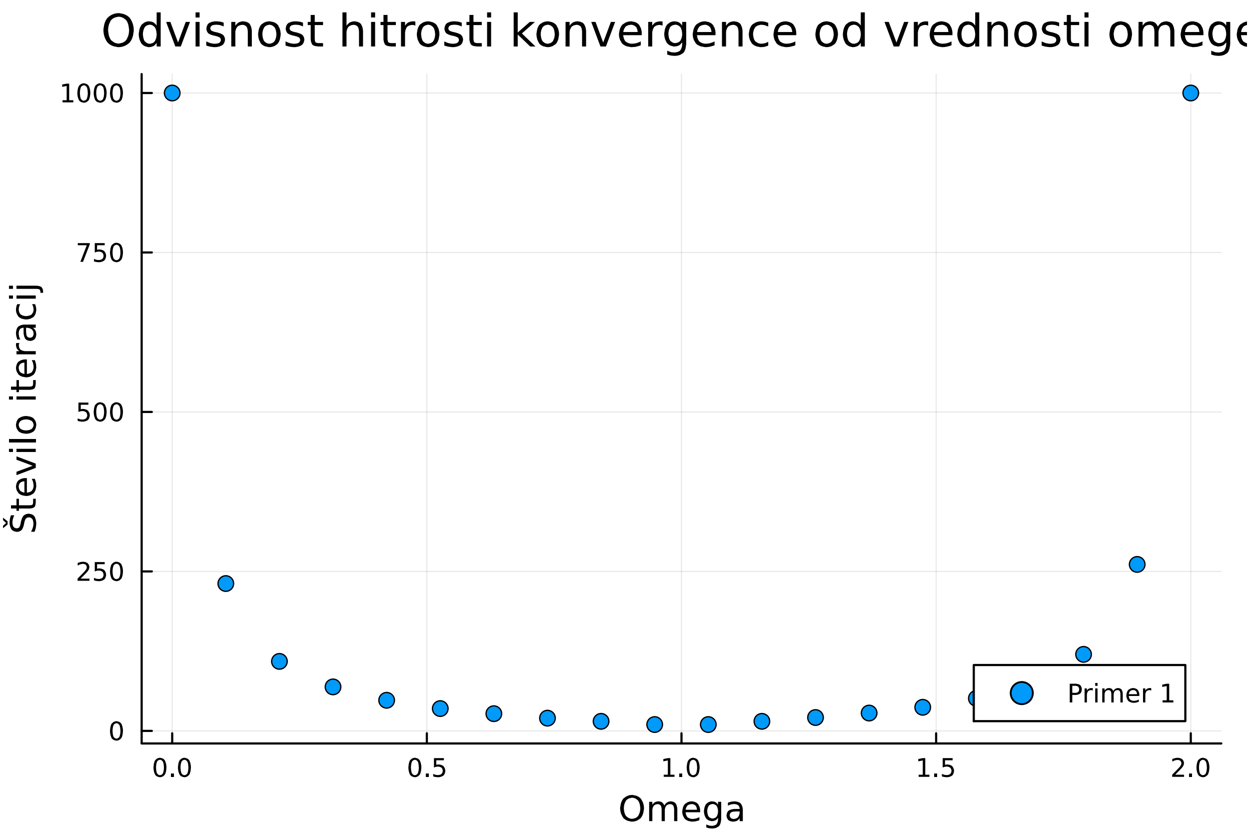
Rezultat

```
rez
```

```
3-element Vector{Float64}:
 0.49999999999991845
 3.3333333333523623
 0.6666666666665579
```

Graf hitrosti konvergence

```
display(plot)
```



Primer 2

```
vozlisca = [1, 2, 3, 4]
robovi = [(1, 1), (2, 2), (2, 3), (3, 3), (3, 1), (4, 3), (4, 4), (4, 2)]
vrednosti = [2.0, 1.0, 1.0, 3.0, 1.0, 4.0, 3.0, 12.0]
B = ustvariRazsprsenoMatriko(vozlisca, robovi, vrednosti)

b = [1.0, 4.0, 2.0, 3.0]
x0 = [0.0, 0.0, 0.0, 0.0]

plot, minIteracij, minOmega, rez = optimalnaOmega(B, b, x0, 2)
```

```
(Plot{Plots.GRBackend() n=2}, 10.0, 0.9473684210526315, [0.4999999999999184
5, 3.4999999999932485, 0.500000000000176, -13.666666666659667])
```

Optimalna vrednost omege

```
minOmega
```

```
0.9473684210526315
```

Minimalno število iteracij

```
minIteracij
```

```
10.0
```

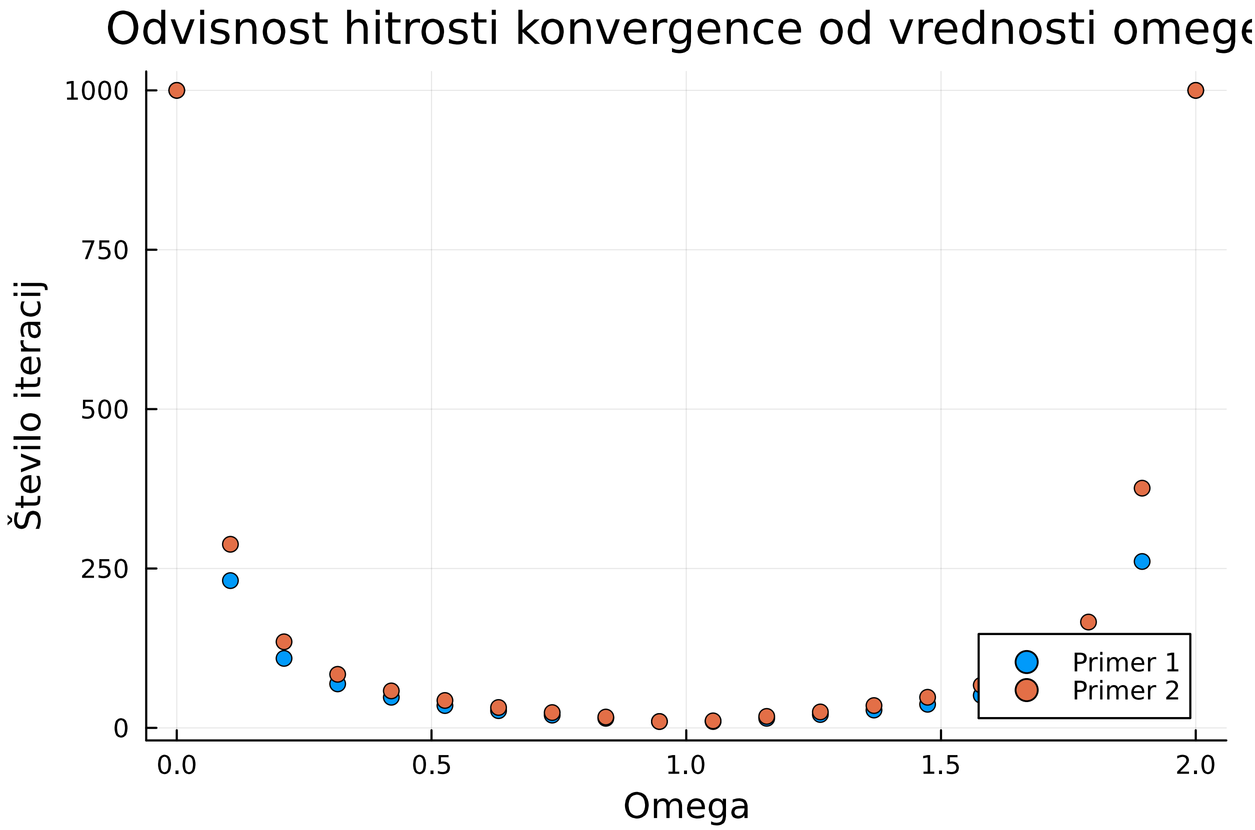
Rezultat

```
rez
```

```
4-element Vector{Float64}:
   0.49999999999991845
   3.4999999999932485
   0.500000000000176
 -13.666666666659667
```

Graf hitrosti konvergence

```
display(plot)
```



Primer 3

```
vozlisca = [1, 2, 3, 4]
robovi = [(1, 1), (2, 2), (2, 3), (3, 3), (3, 4), (3, 1), (4, 3), (4, 4), (4, 2), (4, 1)]
vrednosti = [2.0, 1.0, 1.0, 3.0, 1.0, 4.0, 3.0, 12.0, 1.5, 8.5]
C = ustvariRazsprsenoMatriko(vozlisca, robovi, vrednosti)

plot, minIteracij, minOmega, rez = optimalnaOmega(C, b, x0, 3)
```

```
(Plot{Plots.GRBackend() n=3}, 16.0, 0.9473684210526315, [0.5, 3.78985507252
9117, 0.21014492754805617, -0.6304347826215745])
```

Optimalna vrednost omege

```
minOmega
```

```
0.9473684210526315
```

Minimalno število iteracij

```
minIteracij
```

```
16.0
```

Rezultat

```
rez
```

```
4-element Vector{Float64}:
  0.5
  3.789855072529117
  0.21014492754805617
 -0.6304347826215745
```

Graf hitrosti konvergence

```
display(plot)
```