Poročilo 1. domače naloge

Rok Klobučar
Ljubljana, April 2024

I. Uvod

Ideja ekstrapolirana Gauss-Seidlove iteracija ali SOR(w), kjer je realno število w $\in$ R relaksacijski parameter, je pospešiti GS–iteracijo tako, da nov približek računamo kot uteženo povprečje.

$$x_i^{(k)} = (1-w)\cdot x_i^{(k-1)} + w \cdot x_i^{(k)}$$

pri čemer $x_i^{(m+1)}$ na desni strani enačbe izračunamo iz predpisa za GS-iteracijo:

$$x_i^{(k)} = (1-w)\cdot x_i^{(k-1)} + \frac{w}{a_{ii}}\cdot(b_j - \sum_{j<i}^{n} a_{ij}\cdot x_j^{(k)} - \sum_{j>i}^{n} a_{ij}\cdot x_j^{(k-1)})$$

Če to enačbo preuredimo in zapišemo v matrični obliki dobimo:

$$R_{SOR(w)} = (D - w\cdot L)^{-1}\cdot((1-w)\cdot D + w\cdot U)$$

$$c_{SOR(w)} = w\cdot(D - w\cdot L)^{-1}\cdot b$$

II. Navodilo naloge - SOR iteracija za razpršene matrike

Naj bo A n × n diagonalno dominantna razpršena matrika (velika večina elementov je ničelnih aij=0).

Definirajte nov podatkovni tip RazprsenaMatrika, ki matriko zaradi prostorskih zahtev hrani v dveh matrikah V in I, kjer sta V in I matriki n × m, tako da velja:

$$V(i,j) = A(i, I(i,j))$$

V matriki V se torej nahajajo neničelni elementi matrike A. Vsaka vrstica matrike V vsebuje ničelne elemente iz iste vrstice v A. V matriki I pa so shranjeni indeksi stolpcev teh neničelnih elementov.

Za podatkovni tip RazprsenaMatrika definirajte metode za naslednje funkcije:

Base.getindex,
Base.setindex!,
Base.firstindex,
Base.lastindex
množenje z desne Base.* z vektorjem

Napišite funkcijo x, it = sor(A, b, x0, omega, tol=1e-10), ki reši tridiagonalni sistem Ax=b z SOR iteracijo. Pri tem je x0 začetni približek, tol pogoj za ustavitev iteracije in omega parameter pri SOR iteraciji. Iteracija naj se ustavi, ko je

$$|Ax^{(k)} - b|_{\inf} < tol$$

Metodo uporabite za vožitev grafa v ravnino ali prostor s fizikalno metodo. Če so (xi,yi,zi) koordinate vozlišč grafa v prostoru, potem vsaka koordinata posebej zadošča enačbam

$$\begin{aligned} -st(i)x_i + \textstyle\sum_{j\in N(i)} x_j &= 0, \\ -st(i)y_i + \textstyle\sum_{j\in N(i)} y_j &= 0, \\ -st(i)z_i + \textstyle\sum_{j\in N(i)} z_j &= 0, \end{aligned}$$

kjer je st(i) stopnja i-tega vozlišča, N(i) pa množica indeksov sosednjih vozlišč. Če nekatera vozlišča fiksiramo, bodo ostala zavzela ravnovesno lego med fiksiranimi vozlišči.

Za primere, ki jih boste opisali, poiščite optimalni omega, pri katerem SOR najhitreje konvergira in predstavite odvisnost hitrosti konvergence od izbire ω.

III. Splošna rešitev

Ustvarili smo podatkovno strukturo RazpršenaMatrika Ta vsebuje dve matriki V in I. V hrani neničelne vrednosti, I hrani stolpec pripadajoče neničelne vrednosti. Če je v eni vrstici manj elementov, kot v drugih, jo podložimo (ang. padding), oziroma dopolnimo z ničlami.

Funkcija sor vzame matriko podatkovne strukture RazpršenaMatrika, vektor, začetni približek, relaksacijski parameter ter toleranco. Začetni približek kopiramo v vektor x_n. V vsaki iteracije se sprehodimo čez vse elemente matrike. Če je element ničelen, ga izpustimo, saj nam tej elementi služjio zgolj za zapolnitev matrike. Element se pomnoži s pripadajočim členom vektorja x_n. Rezultat se prišteje k vsoti.

$$vsota = \sum_{i\neq j}^{n} a_{ij}\cdot x_n$$

V primeru, da je element diagonalen, si ga zgolj shranimo in ne izvedemo množenja.

$$diagonalniElement = a_{ii}$$

Tako imamo vse vrednosti za izračun prvega približka:

$$x_i^{(k)} = (1-w)\cdot x_i^{(k-1)} + \frac{w}{a_{ii}}\cdot(b_i - \sum_{i\neq j}^{n} a_{ij}\cdot x_n)$$

Ali drugače:

$$x_i^{(k)} = (1-w)\cdot x_i^{(k-1)} + \frac{w}{diagonalniElement}\cdot(b_i - vsota)$$

To ponavljamo dokler je razlika med prejšnjo in novo vrednostjo večja od tolerance. Ponavljanje se predčasno ustavi, če smo dosegli 10.000 ponovitev, oziroma takrat, ko iteracija ne skonvergira.

Za uporabo so potrebne knjižnice LinearAlgebra, Plots in Test.

IV. Dodatne funkcije

Ustvarili smo funkcijo za iskanje lastnih vrednosti $R_{SOR(w)}$ matrike. Matriko smo razcepili na 3 matrike. Prva matrika D hrani diagonalne elemente, druga matrika U naddiagonalne in tretja matrika L poddiagonalne. Z njimi smo izračunali matriko R in našli njene lastne vrednosti. SOR iteracija bo konvergirala le takrat, ko so vse lastne vrednosti različne od 0.

Ustvarili smo tudi funkcije
Base.getindex,
Base.setindex!,
Base.firstindex,
Base.lastindex.
Zadnji dve sta nam služili, da smo izvedeli, koliko vrstic in stolpcev ima matrika.

Funkcija za množenje z vektorjem z desne Base.product nam je omogočila preverbo rezultata.

$$A \cdot x = b$$

Ali drugače:

$$A \cdot x - b = 0$$

Za izris odvisnosti konvergence od relaksacijskega parametra smo ustvarili funkcijo OdvinostW.

V. Primer

Koda je dostopna na: https://github.com/Hatterok/NM

Vzeli smo matriko A:

```
A = [ -2.0 1.0 1.0 0.0 0.0 0.0;
      1.0 -2.0 1.0 0.0 0.0 0.0;
      0.0 1.0 -2.0 1.0 0.0 0.0;
      0.0 1.0 0.0 -3.0 1.0 0.0;
      0.0 0.0 0.0 1.0 -2.0 1.0;
      0.0 0.0 0.0 1.0 1.0 -2.0;
]
```

Shranili smo jo v strukturo RazpršenaMatrika:

```
B = RazprsenaMatrika(
    [   -2.0 1.0 1.0;
        1.0 -2.0 1.0;
        1.0 -2.0 1.0;
        1.0 -3.0 1.0;
        1.0 -2.0 1.0;
        1.0 1.0 -2.0;
    ],

    [   1 2 3;
        1 2 3;
        2 3 4;
        2 4 5;
        4 5 6;
        4 5 6;
    ]
)
```

Ustvarili smo vektor b:

```
b = [   1.0, 0.0, 0.0,
        0.0, 1.0, 0.0
]
```

Ustvarili smo začetni približek x0:

```
x0 = ones(6) * 1.0
```

Postavili smo toleranco:

```
tol = 1e-10
```

Postavili smo relaksacijski parameter:

```
w = 1.3
```

Izračunali smo lastne vrednosti matrike R:

```
LastnaVrednost = LastneVrednosti(A, w)

6-element Vector{Float64}:
 0.6774947520994041
 0.2133705893066192
 0.3122113037727775
 0.3122113037727775
 0.06760724545872322
 0.7652380364745255
```

Vse so različne od 0. Nadaljujemo naprej.

Izvedli smo SOR iteracijo:

```
xn, korak = sor(B,b,x0,w,tol)

6-element Vector{Float64}:
 -2.3333333330601977
 -1.9999999997511635
 -1.6666666664678809
 -1.333333333197976
 -1.9999999998576787
 -1.666666666537007

 88
```

Izrisali smo odvisnost konvergence od relaksacijskega parametra:

```
OdvisnostW(B, b, x0, w, tol)
```

Glej slika 1.

Preverili smo rezultat z našo in vgrajeno funkcijo. Obakrat dobimo enak rezultat:

```
Base.product(B,xn)-b
A*xn-b

6-element Vector{Float64}:
 -9.864908889767321e-11
 -2.575184510078543e-11
 -1.3377743357523286e-11
 -1.4914292023604503e-11
 -1.9625412406298892e-11
  1.835909202441144e-11
```

Rezultat je približno enak 0, torej je rešitev pravilna.

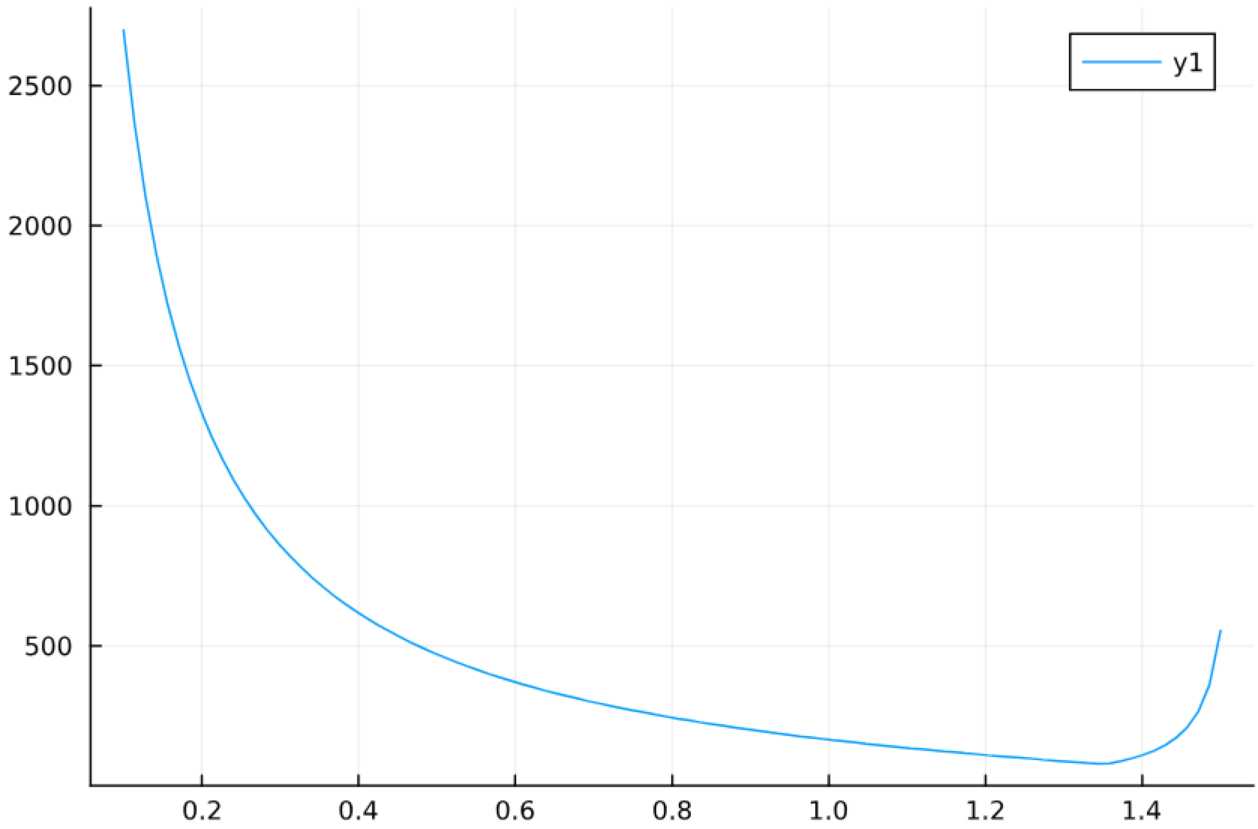

Figure 1. Odvisnost konvergence od relaksacijskega parametra