

1 Metoda konjugiranih gradientov s predpogojevanjem

Avtor: Luka Bajić

1.1 Opis problema

Metoda konjugiranih gradientov je postopek za reševanje linearnega sistema enačb $Ax = b$, ob predpostavki, da je matrika A pozitivno definitna. V našem primeru se ukvarjamo z razpršenimi matrikami, torej matrikami, ki imajo večino elementov ničelnih. Da izboljšamo hitrost konvergence metode konjugiranih gradientov, uporabimo predpogojevanje z matriko $M = LL^T$, kjer spodnjetrokotno matriko L dobimo z nepopolnim razcepom Choleskega.

Ker imamo opravka z razpršenimi matrikami, je s stališča prostorske zahtevnosti smiselno, da jih hranimo v posebni strukturi, ki hrani samo neničelne elemente in sestoji iz treh vektorjev: vektor vrednosti v matriki, vektor indeksov po vrsticah in vektor indeksov po stolpcih. Na primer za sledečo matriko:

$$A = \begin{bmatrix} 7 & 1.1 & 0 & 0 & 0 \\ 1.1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 3 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 3 & 0 & 4.2 \end{bmatrix}$$

hranimo vektorje

$$\begin{aligned} I &= [1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 4 \quad 5 \quad 5] \\ J &= [1 \quad 2 \quad 1 \quad 2 \quad 3 \quad 5 \quad 4 \quad 3 \quad 5] \\ V &= [7 \quad 1.1 \quad 1.1 \quad 2 \quad 3 \quad 3 \quad 0.5 \quad 3 \quad 4.2] \end{aligned}$$

1.2 Opis rešitve

Implementacija ponuja tri metode: *nep_chol*, ki izračuna nepopolni razcep Choleskega za podano matriko A , *conj_grad_baseline*, ki izvede metodo konjugiranih gradientov nad podano matriko A in vektorjem desnih strani b in vrne iskan x , in *conj_grad*, ki prav tako vrne x , le da metodo konjugiranih gradientov izvede s predpogojevanjem s podano spodnjetrokotno matriko L .

1.2.1 Nepopolni razcep Choleskega

1.2.2 Metoda konjugiranih gradientov brez predpogojevanja

Iščemo rešitev sistema $Ax = b$, kjer je matrika A simetrična in pozitivno definitna. Za začetni približek x_0 vzamemo kar ničelni vektor in izračunamo $r_0 = b - Ax_0$. Nastavimo še $p_0 = x_0$ in nato ponavljamo sledečo iteracijo:

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k - \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

Postopek zaključimo kadar je norma vektorja r_{k+1} manjša od zelene tolerance.

1.2.3 Metoda konjugiranih gradientov s predpogojevanjem

Predpogojevanje izvajamo z matriko $M = LL^T$, kjer spodnjetrokotno matriko L dobimo z nepopolnim razcepom Choleskega. Rešiti moramo sistem $Mz = r$, vendar ker nočemo računati inverza M^{-1} (iz vidika časovne zahtevnosti), lahko najprej s premo substitucijo rešimo sistem $La = r$ in nato z obratno substitucijo rešimo sistem $L^T z = a$. Tako prema kot obratna substitucija imata časovno zahtevnost $O(n^2)$, kar je bistveno hitrejša od računanja inverza.

Zgornji postopek moramo ponoviti v vsakem koraku iteracije, in sicer na sledeč način: najprej kot začetno vrednost p_0 nastavimo rešitev sistema $Mz_0 = r_0$, korake pa izvajamo na sledeč način:

$$\alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k - \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

Rešimo sistem $Mz_{k+1} = r_{k+1}$.

$$\beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$$

$$p_{k+1} = z_{k+1} + \beta_k p_k$$

Zaustavitveni pogoj ostane enak kot pri osnovni metodi.

1.3 Primer uporabe

Spodnji izsek kode demonstrira razliko med metodo konjugiranih gradientov brez predpogojevanja in s predpogojevanjem.

```
using MKG, Plots, LinearAlgebra, SparseArrays

I = [1., 1, 2, 2, 3, 3, 4, 5, 5]
J = [1., 2, 1, 2, 3, 5, 4, 3, 5]
V = [7., 1.1, 1.1, 2, 3, 3, 0.5, 3, 4.2]
A = sparse(I, J, V)
b = [2, 3, -5, 1, 0.2]
L = nep_chol(A)
x1, it1, res1 = conj_grad_baseline(A, b, vrniresid=true)
x2, it2, res2 = conj_grad(A, b, L, vrniresid=true)

println("MKG brez predpogojevanja se zaustavi po $it1 korakih.")
println("MKG s predpogojevanjem se zaustavi po $it2 korakih.")
```

MKG brez predpogojevanja se zaustavi po 5 korakih.

MKG s predpogojevanjem se zaustavi po 1 korakih.

Kot vidimo že na relativno majhnem primeru, predpogojevanje bistveno izboljša konvergenco.

V spodnjem izseku si ogledamo še primer z bistveno večjo matriko, ki jo zgeneriramo psevdonaključno: za vse diagonalne elemente zgeneriramo naključna števila iz nekega intervala, za preostale elemente pa z neko manjšo verjetnostjo (npr. 0.1) zgeneriramo manjša števila, da ohranimo dominantnost diagonale. Vsako izmed teh števil zapišemo tako na indeks (i, j) kot (j, i) , da ohranimo simetričnost. Ta postopek sicer ne zagotavlja pozitivne definitnosti zgenerirane matrike, vendar se empirično izkaže kot dovolj dober za potrebe te demonstracije.

Z zastavico *vrniresid* nam metodi vrneti tabelo residualov (oziroma neskončnih norm vektorjev r_{k+1}) na posameznem koraku iteracije. Residuale izrišemo s paketom Plots in tako dobimo vizualno primerjavo hitrosti konvergence med metodama.

```
n=700
I = [1.0]
J = [1.0]
V = [rand()*10.0]
for i=2:n
    push!(V, rand()*80.0)
    push!(I, i)
    push!(J, i)

    if rand() < 0.1
        r = rand()
        push!(V, r)
        push!(V, r)
        r1 = rand(1:n)
        r2 = rand(1:n)
        while r1==i || r2==i # poskrbimo, da ne prepisemo diagonale
            r1 = rand(1:n)
            r2 = rand(1:n)
        end
        push!(I, r1)
        push!(J, r2)
        push!(I, r2)
        push!(J, r1)
    end
end
```

```

    end
end
A = sparse(I, J, V)
b = rand(n)
x1, it1, res1 = conj_grad_baseline(A, b, vrniresid=true, tol=10e-20)
L = nep_chol(A)
x2, it2, res2 = conj_grad(A, b, L, vrniresid=true, tol=10e-20)

plot(res1, label="brez predpogojevanja", title="Primerjava residualov")
plot!(res2, label="s predpogojevanjem", title="Primerjava residualov")

```

Primerjava residualov

