Domača naloga 1: Inverzna potenčna metoda za tridiagonalno matriko

Maksimiljan Vojvoda
Maj 2024

Uvod

V domači nalogi je bilo potrebno implementirati Hessenbergov razcep in inverzno potenčno metodo za izračunanje lastne vrednosti matrike. Za namene računanja je bilo potrebno implementirati še LU razcep in posebne podatkovne strukture, ki zmanjšajo pomnilniško porabo matrik: `Tridiagonalna`, `ZgornjaTridiagonalna` in `SpodnjaTridiagonalna`.

Podatkovne strukture

Zaradi narave problema so matrike pri računanju večinoma napolnjene z samimi 0, razen po treh diagonalah. Temu primerno implementiramo podatkovno strukturo, ki hrani le te tri diagonale, ostale ničle pa obstajajo le implicitno. Poleg tipa `Tridiagonalna` imamo še tipa `ZgornjaTridiagonalna` in `SpodnjaTridiagonalna`, ki hranita le 2 diagonali.

Primer matrike tipa `Tridiagonalna` si lahko predstavljamo kot:

$$\begin{bmatrix} m_{1,1} & m_{1,2} & & & \\ m_{2,1} & m_{2,2} & m_{2,3} & & \\ & m_{3,2} & m_{3,3} & m_{3,4} & \\ & & m_{4,3} & m_{4,4} & \dots \\ & & & \dots & \dots \end{bmatrix}$$

Primer matrike tipa `ZgornjaTridiagonalna` si lahko predstavljamo kot:

$$\begin{bmatrix} m_{1,1} & m_{1,2} & & & \\ & m_{2,2} & m_{2,3} & & \\ & & m_{3,3} & m_{3,4} & \\ & & & m_{4,4} & \dots \\ & & & & \dots \end{bmatrix}$$

Primer matrike tipa `SpodnjaTridiagonalna` si lahko predstavljamo kot:

$$\begin{bmatrix} m_{1,1} & & & & \\ m_{2,1} & m_{2,2} & & & \\ & m_{3,2} & m_{3,3} & & \\ & & m_{4,3} & m_{4,4} & \\ & & & \cdots & \cdots \end{bmatrix}$$

In [ ]:
```
using Domaca01: Tridiagonalna, ZgornjaTridiagonalna, SpodnjaTridiagonalna
using Printf

"""
        Izpiši matriko. Uporabno za nove podatkovne strukture.
"""
function p(A)
        for i=1:size(A, 1)
                for j=1:size(A, 2)

                        if (A[i,j] != 0)
                                @printf "%7.3f " A[i,j]
                        else
                                @printf "        "
                                # @printf "    0   "
                        end
                end
                println()
        end
end
```

Tridiagonalizacija

S pomočjo Housholderjeve transformacije lahko tridiagonaliziramo matriko A velikosti $n \times n$.
Postopek je dokaj preprost kjer postopoma popravljamo matriko A preko interacij $k = 1, 2, ..., n-2$:

$$\alpha = -sgn(a^{(k)}_{k+1,k}) \sqrt{\sum_{n}^{j=k+1} (a^{(k)}_{j,k})^2}$$

$$r = \sqrt{\frac{1}{2} \alpha \left(\alpha - a^{(k)}_{k+1,k}\right)}$$

$$v^{(k)}_1 = v^{(k)}_2 = \cdots = v^{(k)}_k = 0$$

$$v^{(k)}_{k+1} = \frac{a^{(k)}_{k+1,k} - \alpha}{2r}$$

$$v^{(k)}_j = \frac{a^{(k)}_{j,k}}{2r} \quad \forall\, j \in [k+1, n]$$

$$Q^{(k)} = I - 2v^{(k)}(v^{(k)})^T$$

$$A^{(k+1)} = Q^{(k)} A^{(k)} Q^{(k)}$$

Pri tem moramo biti pozorni, da so indeksi iteracije zapisani nad posameznimi simboli, npr $A^{(k+1)}$. Dodatko, funkcija sgn vrne predznak vrednosti, kjer ima 0 tudi predznak 1.

Ena od optimizacij, ki jo lahko naredimo v programu je pri računanju vrednosti $Q^{(k)} = I - 2v^{(k)}(v^{(k)})^T$, saj se tako lahko izognemo računanju korena, če vnaprej izračunamo r^2, do česar pride pri računanju matrike $v^{(k)}(v^{(k)})^T$.

In [ ]:
```
using Domaca01: tridiag

A = [4 1 -2 2 ; 1 2 0 1 ; -2 0 3 -2 ; 2 1 -2 -1.0]
H, Q = tridiag(A)
p(H)
```

```
 4.000  -3.000
-3.000   3.333  -1.667
        -1.667  -1.320   0.907
                 0.907   1.987
```

LU razcep

Za potrebe inverzne potenčne metode potrebujemo tudi LU razcep tridiagonalne matrike. Zaradi strukture tridiagonalne matrike, je ta izračun bistveno bolj preprost kot sicer, saj moramo skrbeti zgolj za 3 diagonale. Poleg tega lahko rezultate vpišemo neposredno kar v matriki L in U brez skrbi za nadaljno gausovo redukcijo.

In [ ]:
```
using Domaca01: lu

A = Tridiagonalna([-6, 2.0], [2, -11, -8.0], [3, 13.0])
L, U = lu(A)

println("A:")
p(A)
println("L:")
p(L)
println("U:")
p(U)
```

```
A:
  2.000   3.000
 -6.000 -11.000  13.000
          2.000  -8.000
L:
  1.000
 -3.000   1.000
         -1.000   1.000
U:
  2.000   3.000
         -2.000  13.000
                  5.000
```

Inverzna potenčna metoda

Z inverzno potenčno metodo lahko iz približka lastne vrednosti izračunamo dejansko lastno vrednost λ matrike. Poleg lastne vrednosti zraven izračunamo še lastni vektor in sicer za približek uporabimo kar vektor naključnih vrednost, say bo skupaj z lastno vrednostjo konvergiral v dejanski lastni vektor in vrednost.

V splošnem se to naredi z iteracijo:

$$x^{(n+1)} = \frac{(A - \lambda I)^{-1} x^{(n)}}{||(A - \lambda I)^{-1} x^{(n)}||}$$

V našem primeru lahko iteracijo specializiramo tako, da izračunamo Hessenbergovo matriko oz. tridiagonaliziramo matriko A. Nato izračunamo LU razcep nad matriko $H - \lambda I$. Pridobljeni LU razcep nato uporabimo v iteraciji $L(Ux^{(n+1)}) = x^{(n)}$.

Iteriranje zaključimo, ko se $||Hx - \lambda x||$ spusti pod poljubno toleranco napake.

In [ ]:
```
using Domaca01: inv_lastni

A = [4.0 1 -2 2 ; 1 2 0 1 ; -2 0 3 -2 ; 2 1 -2 -1]
λ, v = inv_lastni(A, 0) # Najdi najmanjšo lastno vrednost

println("λ = ", λ)
println("v:")
p(v)
```

```
λ = 1.0843644637890046
v:
  1.000
 -0.847
  0.810
 -0.224
```

Lastne vrednosti laplacetove matrike

Laplacetova matrika je matrika, ki ima na glavni diagonali vrednosti -2, na spodnji in zgornji diagonali pa vrednost 1.

In [ ]:
```
function laplace(n::Int)
        return Tridiagonalna(fill(1, n-1), fill(-2, n), fill(1, n-1))
end

function mtrx(A)
        B = zeros(size(A))
        for i=firstindex(A,1):lastindex(A,1)
        for j=firstindex(A,2):lastindex(A,2)
                B[i,j] = A[i,j]
        end
        end
        return B
```

```
end

p(laplace(4))
```

```
-2.000   1.000
 1.000  -2.000   1.000
         1.000  -2.000   1.000
                 1.000  -2.000
```

In [ ]:

```
lap = mtrx(laplace(10))
l,v = inv_lastni(lap, -4)

println(l)
p(v)
```

```
-3.918985947212332
 0.285
-0.546
 0.764
-0.919
 1.000
-1.000
 0.919
-0.764
 0.546
-0.285
```