Seminarski rad iz predmeta Numerički algoritmi

QR algoritam za nalaženje sopstvenih vrijednosti matrica uz prethodno svođenje matrice na Hessenbergovu normalnu formu

Prezime i ime: Šljivo Kerim

Br. indexa: 18222

Elektrotehnički fakultet Sarajevo

Sažetak/Abstract

Ovaj seminarski rad posvećen je temi *QR algoritam za nalaženje sopstvenih vrijednosti matrica uz prethodno svođenje matrice na Hessenbergovu normalnu formu*. QR algoritam se smatra jednim od najznačajnijih algoritama svih vremena. U ovom radu izložit će se glavne ideje na kojima se temelji rad ovog algoritma, jednostavan pseudokod istog i njegova implementacija u programskom jeziku Julia. Također, algoritam će se iskoristiti u nekim testnim primjerima kako bi se demonstrirao njegov rad kao i njegova uspješnost. Na kraju, diskutovat će se efikasnost ovog algoritma te navesti nedostaci istog.

This seminar paper is dedicated to the topic *QR algorithm for finding the eigenvalues of matrices with the previous reduction of the matrix to Hessenberg's normal form*. The QR algorithm is considered one of the most significant algorithms of all time. This paper will present the main underlying ideas of this algorithm, its simple pseudocode and its implementation in the Julia programming language. Also, the algorithm will be used in some test cases to demonstrate its performance. Finally, we will discuss algorithm's efficiency and its shortcomings.

Teoretski uvod

Sopstvene vrijednost matrice $\mathbf{A}$ su sva rješenja jednačine $det(\mathbf{A}-\lambda\mathbf{I}) = 0$. Pošto rješavanje ove jednačine u općem slučaju nije lagan posao, kao alternativa ručnom nalaženju vrijednosti λ nudi se upravo QR algoritam. Glavna ideja ovog algoritma je sljedeća. Najprije izvršimo QR faktorizaciju matrice $\mathbf{A}$, tj. nađemo matrice $\mathbf{Q}$ i $\mathbf{R}$ takve da vrijedi $\mathbf{A} = \mathbf{QR}$. Zatim formiramo matricu $\mathbf{A}^{(2)} = \mathbf{RQ}$. Sada ponavljamo ovaj postupak, dakle vršimo QR faktorizaciju nad matricom $\mathbf{A}^{(2)}$ i dobijamo matrice $\mathbf{Q}^{(2)}$ i $\mathbf{R}^{(2)}$ takve da vrijedi $\mathbf{A}^{(2)} = \mathbf{Q}^{(2)}\mathbf{R}^{(2)}$ i formiramo matricu $\mathbf{A}^{(3)} = \mathbf{R}^{(2)}\mathbf{Q}^{(2)}$. Dakle, u k-toj iteraciji imamo matricu $\mathbf{A}^{(k)} = \mathbf{Q}^{(k)}\mathbf{R}^{(k)}$ na osnovu koje formiramo matricu $\mathbf{A}^{(k+1)} = \mathbf{R}^{(k)}\mathbf{Q}^{(k)}$. Pod pretpostavkom da su sopstvene vrijednosti matrice $\mathbf{A}$ realne i različite, niz matrica $\mathbf{A}$, $\mathbf{A}^{(2)}$, $\mathbf{A}^{(3)}$, itd. konvergira ka matrici $\mathbf{A}$ u tzv. **Schurovoj formi**. To je gornja trougaona matrica na čijoj se glavnoj dijagonali nalaze sve sopstvene vrijednosti matrice $\mathbf{A}$. Kako bi se ovaj algoritam ubrzao, poželjno je najprije matricu $\mathbf{A}$ svesti na tzv. **Hessenbergovu formu**. Matrica svedena na Hessenbergovu formu je matrica koja nije jednaka polaznoj matrici ali su joj sopstvene vrijednosti očuvane i svi njeni elementi ispod dijagonale koja se nalazi ispod glavne dijagonale su jednaki nuli. U nastavku će se opisati način svođenja matrice na Hessenbergovu formu.

Najprije, formiramo vektore $\mathbf{x}$ i $\mathbf{w}$. Vektor $\mathbf{x}$ se satoji od svih elemenata ispod glavne dijagonale u prvoj koloni matrice $\mathbf{A}$, uključujući i element na glavnoj dijagonali. Vektor $\mathbf{w}$ ima isti broj elemenata kao i vektor $\mathbf{x}$. Njegov prvi element jednak je normi vektora $\mathbf{x}$ sa suprotnim predznakom prvog elementa vektora $\mathbf{x}$ (ovo se radi u cilju izbjegavanja katastrofalnog kraćenja), a svi ostali elementi su jednaki nuli. Sada formiramo vektor $\mathbf{v} = \mathbf{w}-\mathbf{x}$. Dalje, računamo matricu $\mathbf{P} = \frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}$. Zatim formiramo matricu $\mathbf{H} = \mathbf{I} - 2\mathbf{P}$. Ako dobijemo da ova matrica ima manje dimenzije od matrice $\mathbf{A}$ onda dodajemo redove iznad i kolone ispred matrice $\mathbf{H}$ tako da novi elementi budu nule osim elemenata na glavnoj dijagonali koje treba postaviti na 1. Zatim formiramo matricu $\mathbf{A}^{(2)} = \mathbf{HAH}$. Ovaj postupak se nastavlja, samo što u nastavku umjesto matrice $\mathbf{A}$ transformiramo matricu $\mathbf{A}^{(2)}$ s tim da se vektor $\mathbf{x}$ u sljedećoj iteraciji formira na način da se uzmu svi elementi ispod glavne dijagonale iz druge kolone matrice $\mathbf{A}^{(2)}$, uključujući i element na glavnoj dijagonali. Ovaj postupak se vrši sve dok je vektor $\mathbf{x}$ različit od vektora koji sadrži samo jednu komponentu. Dakle, u k-toj iteraciji u kojoj ovaj postupak terminira, dobija se matrica $\mathbf{A}^{(k)}$ koja predstavlja matricu $\mathbf{A}$ u Hessenbergovoj formi. U nastavku ćemo prikazati pseudokod QR algoritma.

$bilo \leftarrow 1$
$k \leftarrow 0$
$\mathbf{P} \leftarrow hessenberg(\mathbf{S})$
while $bilo = 1$ **do**
    $bilo \leftarrow 0$
    $pom \leftarrow qr(P)$
    $Q \leftarrow pom[1]$
    $R \leftarrow pom[2]$
    $P \leftarrow R \cdot Q$
    for i in 1 .. n **do**
        for j in 1 .. n **do**
            if $i > j$ and $P[i][j] \neq 0$ **then**
                $bilo \leftarrow 1$
                break
            end if
        end for
    end for
end while
for i in 1 .. n **do**
    $e[i] \leftarrow P[i][i]$
end for

Osvrnimo se nakratko na ovaj pseudokod. Ulazni podatak je kvadratna matrica **S** veličine n. U matricu **P** se smješta matrica **S** u Hessenbergovoj formi. U pomoćni vektor smještamo matrice **Q** i **R** i u nastavku koda vršimo postupak koji je već opisan. Taj postupak se ponavlja sve dok matrica **P** nije gornja trougaona. Na kraju, sopstvene matrice se smještaju u vektor **e**.

Implementacija algoritma

U nastavku slijedi implementacija algoritma u Julia programskom jeziku:

```
!curl -sSL "https://julialang-s3.julialang.org/bin/linux/x64/1.5/
!tar -xzf julia.tar.gz -C /usr --strip-components 1
!rm -rf julia.tar.gz*
!julia -e 'using Pkg; pkg"add IJulia; precompile"'
!echo "DONE"

using LinearAlgebra
using Pkg;
Pkg.add(url="https://github.com/JuliaMatrices/SpecialMatrices.jl")
using SpecialMatrices
Pkg.add("RowEchelon")
using RowEchelon
```

```
function hessenberg(P)
  n = size(P)[1]
  t = 2
  b = 1

  while t != n
    x = ones(n - t + 1, 1)
    w = zeros(n - t + 1, 1)
    suma = 0
    i = 1

    for j in t : n
      x[i, 1] = P[j, b]
      suma = suma + x[i, 1] ^ 2
      i = i + 1
    end

    w[1, 1] = -sign(x[1, 1]) * sqrt(suma)
    v = w - x

    pom = (transpose(v) * v)[1, 1]
    p = (v * transpose(v)) / pom
    dim = size(p)[1]
    H1 = Matrix{Float64}(I, dim, dim) - 2 * p
    H = Matrix{Float64}(I, n, n)
    i = 0
    j = 0
    while n + 1 - i != dim
      i = i + 1;
      j = j + 1;
    end

    odakle = j;
    for k in 1 : dim
      for l in 1 : dim
        H[i, j] = H1[k, l]
        j = j + 1
      end
      i = i + 1
      j = odakle
    end
    P = H * P * H
    t = t + 1
    b = b + 1
  end
  return P
end
```

```
function QR(S)
  bilo = 1
  k = 0
  P = hessenberg(S)
  n = size(P)[1]
  while bilo == 1
    bilo = 0
    Q, R = qr(P)
    P = R * Q
    for i in 1 : n
      for j in 1 : n
        if i > j && P[i, j] != 0
          i = n
          j = n
          bilo = 1
        end
      end
    end
  end
  e = ones(n, 1)
  for i in 1 : n
    e[i, 1] = P[i, i]
  end
  return e
end
```

Osvrnimo se nakratko na ovaj kod. U funkciji QR poziva se pomoćna funkcija hessenberg koju ćemo ukratko objasniti jer je funkcije već objašnjena u pseudokodu. Funkcija hessenberg prima matricu P koja se svodi na Hessenbergovu formu. U varijabli n se pamti veličina te matrice. Matrice kolone x, w i v se formiraju na već opisan način. Također se koristi funkcija transpose koja vraća transponiranu matricu pa smo u varijabli *pom* sačuvali vrijednost $\mathbf{v}^T\mathbf{v}$. Ostatak koda je već objašnjen u ranijem izlaganju. U sljedećoj cjelini će se demonstrirati primjena algoritma u nekim konkretnim problemima.

Primjene algoritma u praksi

1. Izračunati sopstvene vrijednosti matrice $M = \begin{pmatrix} 12 & 20 & 22 & 3 \\ 1 & 0 & 5 & 21 \\ 2 & 21 & 210 & 20 \\ 2 & 1 & 2 & 20 \end{pmatrix}$

   Možemo iskoristiti Julia funkciju eigvals:

```
M = [12 20 22 3 ; 1 0 5 21 ; 2 21 210 20; 2 1 2 20]
eigvals(M)
4-element Array{Float64,1}:
 0.4686371889939444
```

```
  7.196606553415909
 23.33881041578897
210.99594584180113
```

A možemo iskoristiti i našu funkciju QR:

```
QR(M)
4x1 Array{Float64,1}:
 210.99594584180102
   23.338810415789006
  7.196606553415922
 0.4686371889939501
```

Kao što vidimo rezultati su gotovo identični.

2. Izračunati sopstvene vrijednosti matrice $N = \begin{pmatrix} 1 & 5 & 5 \\ 7 & 2 & 1 \\ 2 & 3 & 2 \end{pmatrix}$

Ponovo možemo iskoristiti Julia funkciju eigvals:

```
N = [1 5 5; 7 2 1; 2 3 2;]
eigvals(N)
3-element Array{Float64,1}:
 -3.77378379654427
 -0.7252952463210942
  9.499079042865365
```

A možemo ponovo iskoristiti i našu funkciju QR:

```
QR(N)
3x1 Array{Float64,1}:
 9.499079042865349
 -3.7737837965442638
 -0.7252952463210944
```

Kao što vidimo rezultati su ponovo gotovo identični.

Zaključak i diskusija

Vremenska kompleksnost QR faktorizacije matrice $\mathbf{A}$ u Hessenbergovoj formi iznosi $O(n^2)$. U slučaju simetrične matrice, njena Hessenbergova forma je također simetrična pa samim tim i tridijagonalna. Ovo vrijedi za sve ostale matrice $\mathbf{A}^{(k)}$. Vremenska kompleksnost QR faktorizacije simetrične tridijagonalne matrice u Hessenbergovoj formi iznosi $O(n)$. Brzina konvergencije zavisi od razdaljine između sopstvenih vrijednosti. Napredniji algoritmi koriste takozvana pomjeranja (engl. *shifted QR algorithm*) kako bi se ta razdaljina povećala i samim tim konvergencija ubrzala. Tako da je to jedan od načina kako bi se ovaj algoritam mogao unaprijediti. Također, jedna od mana ovog algoritma je što on ne računa kompleksne sopstvene vrijednosti. Međutim, za potrebe u kojima se zna da su sopstvene vrijednosti realne, različite i dosta udaljene jedna od druge, ovaj algoritam ima sasvim zadovoljavajuće performanse.

Reference

- Željko Jurić, *Numerički algoritmi*. Univerzitet u Sarajevu, 2018.
- Stoer Josef; Bulirsch Roland, *Introduction to Numerical Analysis* (3. izdanje), Berlin, New York 2002.