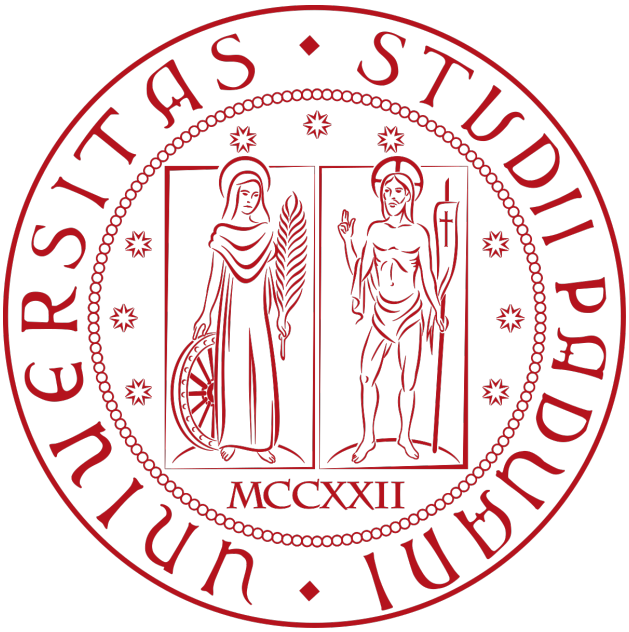
Università degli Studi di Padova



Relazione per:

Progetto 2 Matlab

Realizzata da:
Riccardo Berengan, matricola 2080041
Michele Dioli, matricola 2077629
Gabriele Di Pietro, matricola 2010000

1 Introduzione

Gli autovalori sono le soluzioni dell'equazione $det(A-\lambda I) = 0$. L'autospazio è il sottospazio vettoriale generato da tutti i vettori autovalori λ, insieme al vettore nullo.
Esempio:

$$A = \begin{bmatrix} 4 & 2 \\ 1 & 3 \end{bmatrix}$$

$det(A - \lambda I) = det \begin{bmatrix} 4-\lambda & 2 \\ 1 & 3-\lambda \end{bmatrix} \iff (\lambda - 5)(\lambda - 2) = 0 \iff \lambda_1 = 5 \lambda_2 = 2$
Molteplicità geometrica è la dimensione dell'autospazio associato ad un autovalore quindi $|n - \mathrm{rk}(A - \lambda I)|$ Molteplicità geometrica è la dimensione dell'autospazio associato ad un autovalore quindi $|n - rk(A - \lambda I)|$

2 Calcolo

2.1 Funzione Multigeo

Per il calcolo contiamo gli elementi "nulli" *(minori di una tolleranza)* sulla matrice U ottenuta dalla fattorizzazione LU. Lo si può fare perchè: Una riga nulla in U significa che il sistema lineare di $(A - \lambda I)x = 0$ ha soluzione libera, una riga nulla non dipende dalle altre righe della matrice, le righe nulle indicano che ci sono dipendenze lineari tra le righe. Contiamo quindi gli elementi in diagonale che sono maggiori di una tolleranza, contando così il rango della matrice, lo sottraiamo alla dimensione della matrice e otteniamo la dimensione dell'autospazio dell'autovalore che coincide con la molteplicità geometrica.

2.2 Funzione myobjective

La funzione `myobjective` calcola la funzione f e la funzione g a partire da uno scalare z e da una matrice A quadrata.
Come prima cosa calcoliamo la matrice $B = A - zI$ e poi tramite fattorizzazione LU usando `lu(B)` in Matlab che restituisce la matrice U triangolare superiore e la matrice L triangolare inferiore ed infine la matrice P di permutazione.
$P = \pm 1$ in base al numero di permutazioni effettuate, da qui calcoliamo il determinante di P. Trovato questo calcoliamo il determinante di B come $det(P) \cdot \prod_{i=1}^{n} U_{ii}$, dove U_{ii} sono gli elementi sulla diagonale di U. $f = det(B)$
La funzione dopodichè calcola l'inverso della matrice B e calcola g come l'inverso della traccia *(somma degli elementi sulla diagonale)* di B.

2.3 Funzione Multialg

Consideriamo il polinomio con m molteplicità algebrica $f(x) = (x-\lambda)^m g(x)$ e il metodo di Newton $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ per trovare le radici di $g(x)$. Analizziamo la velocità di convergenza di $f(x) = (x-\lambda)^{m-1} g(x) + (x-\lambda)^m g'(x)$
$f'(x) = 0 \iff x = \lambda$, $m \geq 1$ non è subito alla radice $m(x-\lambda)^{m-1}$ è responsabile per la velocità di convergenza, quindi il metodo di Newton applicato ad una radice di molteplicità $m \geq 1$ ha una velocità lineare non più quadratica.
Abbiamo che la stima dell'errore è ridotta dipendentemente dalla molteplicità della radice $|x_{n+1} - \lambda| \approx |x_n - \lambda|^{\frac{m-1}{m}}$
$\frac{m-1}{m}$ è la velocità di convergenza
(Newton vicino a radici multiple $\Rightarrow |e_{n+1}| \approx C|e_n|^{\frac{m-1}{m}}$ *)*
Calcoliamo l'autovalore con il metodo di Newton, iteriamo fino a un numero di iterazioni e/o per aver raggiunto una tolleranza sensata. Applichiamo Newton al polinomio caratteristico $f(\lambda) = det(A - \lambda I) \Rightarrow \lambda_{n+1} = \lambda_n - \frac{f(\lambda_n)}{f'(\lambda_n)}$. Troviamo così il valore dell'autovalore. Per calcolare la molteplicità algebrica basta dividere l'autovalore per il polinomio caratteristico, e vedere quante volte lo posso fare prima di ottenere un valore di zero.

3 Test

3.1 File `main.m`

Il main permette di definire una matrice di test tramite la modifica del vettore `lambda` e di modificare il punto iniziale per il metodo di newton `l0` e il parametro della tolleranza `toll`.
La matrice per i test viene creata tramite la modifica del parametro `lambda` da qui viene invocata una funzione `creaJacob` che prende in input `lambda` e restituisce la matrice in forma diagonale a blocchi di Jordan, poi viene creata una matrice Q tramite `Q = orth(randn(n))` che genera una matrice ortogonale, ed infine viene calcolata la matrice A usando la struttura $A = Q^t J Q$

3.2 File `test.m`

Il programma `test.m` esegue una serie di test per verificare la correttezza delle funzioni `multialg` e `multigeo`. Queste funzioni sono utilizzate per calcolare gli autovalori di una matrice e le loro rispettive molteplicità algebriche e geometriche. Il programma definisce una serie di casi di test, ciascuno con una matrice di autovalori specificata dal vettore `lambda`, un punto iniziale per il metodo di Newton `l0`, e una tolleranza `toll`.

```
TestCases = {
    struct('lambda', [1,1,1,1,1,4,4,4,4,4,4,4], 'l0', 3.8, '
   toll', 1e-4),
    struct('lambda', [1, 1.01, 1.02, 5, 5.01, 5.02, 10,
   10.001, 10.002, 10.003], 'l0', 10.1, 'toll', 1e-4),
    struct('lambda', [1, 1.01, 1.02, 5, 5.01, 5.02, 10,
   10.001, 10.002, 10.003], 'l0', 1.0125, 'toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6], 'l0', 3.5, 'toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6], 'l0', 3.9, 'toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6,4], 'l0', 3.9, 'toll', 1e
   -4),
    struct('lambda', [1,1,1,4,5,6,4,4], 'l0', 3.5, 'toll', 1e
   -4),
    struct('lambda', [1,1,1,4,5,6,4,4,4], 'l0', 3.7, 'toll',
   1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4], 'l0', 3.3, 'toll'
   , 1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4], 'l0', 3.7, 'toll',
   1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4], 'l0', 3.5, 'toll'
   , 1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4,4], 'l0', 4.3, '
   toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4,4], 'l0', 3.5, '
   toll', 1e-4),
    struct('lambda', [1,1,1,4,5,6,4,4,4,4,4,4], 'l0', 3.4, '
   toll', 1e-4),
    struct('lambda', [4,4,4,4,4,4,4,4], 'l0', 0., 'toll', 1e
   -4),
};
```

1: Test Cases

3.2.1 Esecuzione dei test

Per ogni caso di test, il programma esegue i seguenti passaggi:

1. Stampa il numero del test e i valori di `lambda`, `l0`, e `toll`.
2. Crea una matrice diagonale `J` con i valori di `lambda` e aggiunge 1 agli elementi fuori diagonale per gestire le molteplicità.
3. Genera una matrice ortogonale `Q` e calcola la matrice `A` come `A = Q' * J * Q`.
4. Utilizza la funzione `multialg` per calcolare un autovalore `l` e la sua molteplicità algebrica `m`.
5. Se il metodo di Newton converge, calcola la molteplicità geometrica `k` dell'autovalore utilizzando la funzione `multigeo`.
6. Stampa i risultati del test, inclusi l'autovalore calcolato, la molteplicità algebrica e geometrica, e un messaggio di convergenza.

Il programma fornisce un'analisi dettagliata dei risultati di ciascun test, permettendo di verificare l'accuratezza e l'efficacia delle funzioni implementate.

```
for testID = 1:length(TestCases)
    fprintf('\nExecuting Test %d...\n', testID);
    lambda = TestCases{testID}.lambda;
    fprintf('[%s]\n', sprintf('%g ', lambda));
    l0 = TestCases{testID}.l0;
    fprintf('l0 = %d\n', TestCases{testID}.l0);
    toll = TestCases{testID}.toll;
    fprintf('toll = %d\n', TestCases{testID}.toll);
    J = diag(lambda);
    for i = 2:length(lambda)
        if J(i,i) == J(i-1,i-1)
            J(i-1,i) = 1;
        end
    end
    n = length(lambda);
    Q = orth(randn(n));
    A = Q' * J * Q;
    A1 = A;
    disp(J);
    it = 5;
    maxit = 50;
    [l, m, flag] = multialg(A, l0, toll, it, maxit);
    if flag == 1
        fprintf('Newton convergence.\nCalculated eigenvalue:
   %f\nEstimated algebraic multiplicity: %d\n', l, m);
        l1 = round(l);
        toll = 1e-6;
        k = multigeo(A1, l1, toll);
        fprintf('Geometric multiplicity of %f: %d\n', l1, k);
    else
        fprintf('Non-convergent Method.\nULast calculated
   eigenvalue: %f\nEstimated algebraic multiplicity: %d\n', l
   , m);
    end
    linea = 50;
    fprintf('%s\n', repmat('-', 1, linea));
end
```

2: Esecuzione dei test

3.2.2 Terminal output

Per comodità ci riserviamo di stampare solo il primo test case, gli altri sono simili.

```
Executing Test 1...
[1 1 1 1 1 4 4 4 4 4 4 4 ]
l0 = 3.800000e+00
toll = 1.000000e-04
     1     1     0     0     0     0     0     0     0     0     0     0
     0     1     1     0     0     0     0     0     0     0     0     0
     0     0     1     1     0     0     0     0     0     0     0     0
     0     0     0     1     1     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0     0     0
     0     0     0     0     0     4     1     0     0     0     0     0
     0     0     0     0     0     0     4     1     0     0     0     0
     0     0     0     0     0     0     0     4     1     0     0     0
     0     0     0     0     0     0     0     0     4     1     0     0
     0     0     0     0     0     0     0     0     0     4     1     0
     0     0     0     0     0     0     0     0     0     0     4     1
     0     0     0     0     0     0     0     0     0     0     0     4
Newton convergence.
Calculated eigenvalue: 3.876888
Estimated algebraic multiplicity: 1
Geometric multiplicity of 4.000000: 1
------------------------------------------------------------------------
```

3: Output del primo test case

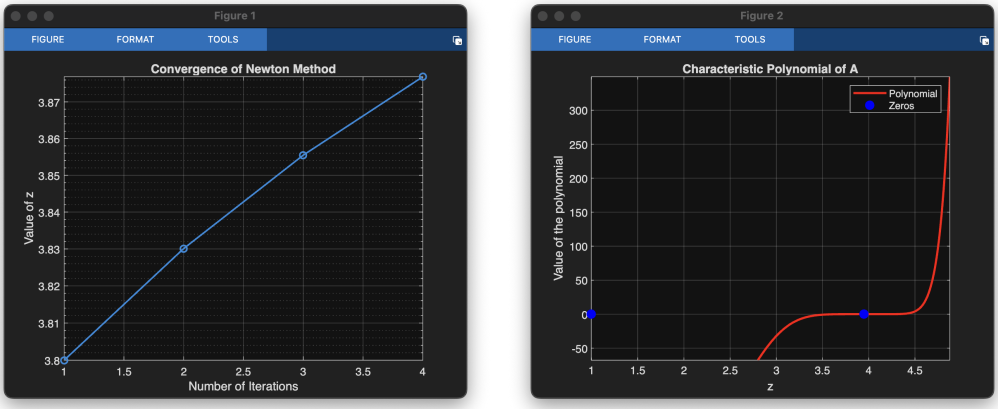


Figure 1: *Convergenza del metodo di Newton e polinomio di A*

Verranno anche mostrate su grafici apparte sia la convergenza del metodo di Newton, sia il polinomio di A.

4 Conclusioni

Possiamo concludere che il software prodotto gestisce bene il problema e può trovare autovalori anche con stime iniziali lontani, infatti il metodo di Newton converge anche con stime iniziali molto distanti (*ad esempio*, `10`$= 3.5$ per $\lambda = 4$).
Il software è stato testato con matrici di varie dimensioni e con molteplicità algebriche e geometriche diverse, e in tutti i casi il metodo di Newton converge, a meno che non si scelga un punto iniziale molto distante dall'autovalore da trovare