Università degli Studi di Cagliari

FACOLTÀ DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Elettrica ed Elettronica

Tesina seminario matematica applicata avanzata

Implementazione in linguaggio C del metodo delle potenze

Candidato:
Roberto Carboni

Relatore:
Giuseppe Rodriguez

Anno Accademico 2017-2018

Indice

Capitolo 1

Metodo delle potenze

1.1 Introduzione al metodo

Il metodo delle potenze è un metodo iterativo per il calcolo approssimato degli autovalori di una data matrice A. In particolare, questo metodo, approssima l'autovalore di modulo massimo (o di modulo minimo se si usa A^{-1}, modificando opportunamente l'algoritmo).
Il metodo converge se: A è diagonalizzabile, con autovalori tali che $|\lambda_1| > |\lambda_i|$, con $i = 2, \ldots, n$ e il vettore iniziale x^0 ha una componente non nulla lungo l'autovettore v_1 (corrispondente a λ_1)

1.2 L'algoritmo

Dato in ingresso un vettore $x^{(0)}$, una tolleranza τ(ossia la precisione richiesta), un numero massimo di iterazioni N e la matrice A, l'algoritmo è:

1. $q^{(0)}=\frac{x^{(0)}}{\|x^{(0)}\|_2}$
2. k=0
3. $\lambda^{(0)} = 0$
4. repeat
    1. k=k+1
    2. $x^k = Aq^{(k-1)}$
    3. $q^{(k)} = \frac{x^k}{\|x^{(k)}\|_2}$
    4. $\lambda^k = (q^k)^T A q^k$
5. until $|\ \lambda^{(k)}\text{-}\lambda^{(k-1)}\ | < |\ \tau\ \ \lambda^{(k)}\ |$ or $k > N$.

Dove x^0 può essere scelto arbitrariamente, tale che non vìoli le condizioni del paragrafo precedente.
Nelle applicazioni successive verrà scelto sia un x^0 di soli uno, che un x^0 di numeri reali casuali.

Capitolo 2

Implementazione

Una possibile implementazione in linguaggio c, di cui adesso riporto solo la funzione main, mentre le altre saranno riportate nell'appendice, è la seguente:

```
int main() {
    double A[N][N], x[N], q[N], temp[N];
    double lambda=0, lambdak=0, norm=0;
    int k=0, i;
    read_v(x);                          //x prende i valori da matlab
    read_m(A);                          //A prende i valori da matlab
    norm=norm2(x);                      //1
    product(q, norm, x);                //1
    do {
        lambda=0;
        lambdak=lambda;
        k=k+1;                          //4.1
        product_m(x, A, q);             //  2
        norm=norm2(x);                  //  3
        product(q, norm, x);            //  3
        product_m(temp, A, q);          //  4 temp=A*q
        for(i=0; i<N; i++){             //  4 lambda=q'*temp
            lambda+=temp[i]*q[i];
        }
        write(lambda);                  //scrive su un file gli autovalori
    }while( ( abs((lambda-lambdak)) > (TAU*abs(lambda)) && k<NMAX) ); //5
    return 0;
}
```

Dove TAU è la tolleranza e NMAX è il numero massimo di iterazioni. Questi parametri vengono usati nel ciclo do-while come criterio di stop.
I numeri a destra del codice si riferiscono ai punti dell'algoritmo visti nel capitolo precedente.

Capitolo 3

Applicazioni

I vettori e le matrici utilizzate sono stati creati in matlab con i seguenti comandi:

```
d=diag([rand((N-1), 1)'lambda]);
q=orth(rand(N));
a=q'*d*q;
x=ones(N, 1);
```

Con il primo comando viene creata una matrice diagonale *d*, di cui *lambda* è l'autovalore di modulo massimo, mentre con il secondo si crea una matrice ortogonale di numeri random. Sfruttando una proprietà delle matrici simili, secondo cui due matrici simili hanno gli stessi autovalori, si ottiene la matrice *a* che avrà , quindi, *lambda* come autovalore di modulo massimo. Infine, con l'ultimo comando, si crea un vettore di soli uno.

3.1 Caso 1

Come primo caso si testerà l'algoritmo usando matrici 4x4, con l'autovalore di modulo massimo ben separato dagli altri.
Gli autovalori esatti, delle matrici usate, sono riportati nella Tabella 3.1.

Test	**Autovalori esatti matrice A**
1	[0.9595 0.6557 0.0357 ***5.0000***]
2	[0.3404 0.5853 0.2238 ***10.0000***]
3	[0.2638 0.1455 0.1361 ***50.0000***]
4	[0.3532 0.8212 0.0154 ***100.0000***]
5	[0.5870 0.2077 0.3012 ***500.0000***]
6	[0.3 0.4 0.6 ***1000***]

Tabella 3.1: Vettori degli autovalori esatti della matrice A

Test	Numero iterazioni	Autovalori ottenuti
1	5	4.626779 4.992776 4.999868 4.999997 ***5.000000***
2	3	9.949960 9.999845 ***10.000000***
3	2	49.997167 ***50.000000***
4	2	99.997772 ***100.000000***
5	2	499.983789 ***500.000005***
6	2	999.999852 ***999.999995***

Tabella 3.2: Output del programma

Come visto dalla Tabella 3.2 l'algoritmo converge in tutti i casi. Si può inoltre notare, dal numero di iterazioni, che più l'autovalore di modulo massimo è separato dagli altri e meno iterazioni compierà l'algoritmo prima di convergere.

3.2 Caso 2

In questo caso si verificherà cosa succede se l'autovalore di modulo massimo ha molteplicità algebrica maggiore di uno.
Come vettore iniziale, questa volta, si è utilizzato un vettore random generato da Matlab:

[0.2435 0.9293 0.3500 0.1966 0.2511]

Mentre per la matrice, questa volta di dimensione 5x5, è stato utilizzato lo stesso procedimento del caso precedente.
Nella Tabella 3.3 sono riportati gli autovalori esatti delle matrici.

Test	Autovalori esatti matrice A
1	[0.1067 0.9619 0.0046 ***5.0000 5.0000***]
2	[0.5678 0.0759 0.0540 ***10.0000 10.0000***]
3	[0.9746 0.9591 0.6650 ***50.0000 50.0000***]

Tabella 3.3: Vettori degli autovalori esatti della matrice A

Test	Numero iterazioni	Autovalori ottenuti
1	4	[4.996302 4.999886 4.999996 ***5.000000***]
2	3	[9.999056 9.999998 ***10.000000***]
3	3	[49.979057 49.999991 ***49.999999***]

Tabella 3.4: Output del programma

Dalla Tabella 3.4 si nota come la molteplicità algebrica maggiore di uno non comprometta il funzionamento dell'algoritmo, poiché il metodo converge ai valori della Tabella 3.3.

3.3 Caso 3

Come visto nel Paragrafo 2.1, le variabili sono state dichiarate come double, mentre in questo paragrafo verrà osservato il risultato nel caso in cui si usino variabili float.
Questi test verranno confrontati con i dati ottenuti nel Paragrafo 3.1 (riportati nella Tabella 3.2).
Nella Tabella 3.5 sono riportati gli output del programma con variabili double (gli stessi dei test numero 2, 4 e 6 della Tabella 3.2); Mentre nella Tabella 3.6 vi sono gli output del programma con variabili float.

Test 1	**Test 2**	**Test 3**
9.949960	99.997772	999.999852
9.999845	100.000000	999.999995
10.000000	100.000000	999.999995

Tabella 3.5: Test con variabili Double

Test 1	**Test 2**	**Test3**
9.949960	99.997772	999.999939
9.999845	100.000000	1000.000061
10.000000	100.000008	999.999939
10.000000	100.000008	1000.000061
10.000000	100.000000	1000.000122
10.000000	100.000008	999.999939
10.000000	100.000000	1000.000061
10.000000	100.000015	1000.000122

Tabella 3.6: Test con variabili Float

Dalla Tabella 3.6 si può notare come l'algoritmo converga solo per autovalori piccoli, mentre per autovalori più grandi oscilla tra valori prossimi ai risultati ottenuti nella Tabella 3.5. Questo , appunto, è dovuto all'uso delle variabili float, che essendo variabili a 4 byte (32 bit) hanno una minor precisione rispetto alle double, che sono da 8 byte (64 bit). Si ha quindi una propagazione dell'errore che causa queste oscillazioni nel risultato.

Appendice A

Codice sorgente

Di seguito riporto l'implementazione delle funzioni usate nel main visto nel Capitolo 2:

```c
#include <stdio.h>
#include <math.h>

#define N 4       //grandezza matrice
#define NMAX 40   //numero max iterazioni
#define TAU 1e-5 //precisione


double norm2(double x[]) {
//norma_2 di un vettore
     int i;
     double x2=0;

     for(i=0; i<N; i++){
         x2+=pow(x[i], 2);
     }
     x2=pow(x2, 0.5);
     return x2; //norma_2 del vettore x
}


void product(double q[], double norm, double x[]){
//inverso di una costante per vettore colonna
    int i;
    for(i=0; i<N; i++) {
        q[i]=x[i]*(1/norm);
    }
}
```

```
31  void product_m(double x[], double A[][N], double q[]){
32  //matrice per vettore colonna
33      int i, j;
34      for(i=0; i<N; i++){
35          x[i]=0;
36      }
37      for(i=0; i<N; i++){
38          for(j=0; j<N; j++){
39              x[i]+=A[i][j]*q[j];
40          }
41      }
42  }
43
44
45  double abs(double x){
46      if(x<0)
47          return -x;
48      else
49          return x;
50  }
```

```
53  void read_v(double x[]){
54      FILE *FP;
55      int i;
56      FP=fopen("vettore_rand_matlab.txt", "r");
57      for(i=0; i<N; i++){
58          fscanf(FP, "%lf", &x[i]);
59      }
60      fclose(FP);
61  }
62
63
64  void read_m(double A[][N]){
65      FILE *FP;
66      int i,j;
67      FP=fopen("matrice_matlab.txt","r");
68      for(i=0; i<N; i++){
69          for(j=0; j<N; j++){
70              fscanf(FP, "%lf", &A[i][j]);
71          }
72      }
73      fclose(FP);
74  }
75
76
77  void write(double lambda){
78      FILE *FP;
79      FP=fopen("autovalore_c.txt","a+");
80      fprintf(FP, "%lf\n", lambda);
81      fclose(FP);
82  }
```