Relazione del corso di Analisi Numerica

Gianluca Covini, # Matricola 481021, Università di Pavia

6 giugno 2021

1 Metodi iterativi per sistemi lineari

Nella presente sezione andiamo a trattare gli esercizi riguardo ai metodi iterativi per la risoluzione di sistemi lineari, cioè i metodi che permettono di costruire una successione di soluzioni approssimate per un dato sistema $Ax = b$ che tendono alla soluzione esatta del sistema. In particolare, verranno trattati i metodi di Jacobi e Gauss-Seidel e il metodo del gradiente coniugato.

1.1 Esercizio 1

L'esercizio richiede l'implementazione del metodo di Jacobi e del metodo di Gauss-Seidel per la risoluzione di metodi lineari della forma $Ax = b$.

Entrambi questi metodi sono metodi di splitting, cioè metodi stazionari della forma

$$x^{(k+1)} = P^{-1}(Nx^{(k)} + b)$$

dove $A = P - N$. Per l'implementazione si sfrutta la riformulazione equivalente

$$x^{(k+1)} = x^{(k)} + P^{-1}r^{(k)}$$

dove l'elemento $r^{(k)}$ è il residuo della k-esima iterazione.

Il problema si riduce, quindi, alla risoluzione, a ogni iterazione, di un sistema lineare $Pp^{(k)} = r^{(k)}$ dove $p^{(k)}$ è l'elemento da aggiungere a $x^{(k)}$ per ottenere $x^{(k)}$. La risoluzione di questo sistema è computazionalmente molto più economica della risoluzione del sistema $Ax = b$ per via della scelta opportuna della matrice P: nel caso del metodo di Jacobi la matrice P è la matrice diagonale di A; nel metodo di Gauss-Seidel è la matrice triangolare inferiore di A. In questo modo i sistemi da risolvere sono sistemi triangolari che si possono affrontare con metodi diretti non molto costosi.

Fissato un sistema $Ax = b$, le funzioni che andiamo a implementare sono due funzioni che, dati A, b, un valore $x^{(0)}$ iniziale, restituisca un un vettore x, soluzione approssimata del sistema $Ax = b$, ottenuta tramite il metodo di Jacobi. La funzione, inoltre, restituisce anche v, il vettore contenente la norma euclidea del residuo a ogni iterazione e *iter*, il numero delle iterazione effettuate. Vengono inoltre fissati come criteri d'arresto due valori, *maxit*, che indica il massimo numero di iterazioni, e *tol*, che indica la tolleranza sul rapporto tra il valore della norma del k-esimo residuo e la norma del primo residuo $r^{(0)}$.

Metodo di Jacobi:

```
1 function [x, v, iter] = Jacobi(A, b, x0, maxit, tol)
2 P = diag(diag(A));  % matrice diagonale di A
3 x = x0;
4 r0 = b-A*x0;
5 v=zeros(maxit, 1);  % vettore da riempire
6 iter = 1;
7 r = r0;
```

```
8
9  while iter<maxit && norm(r)>tol*norm(r0)
10     r = b-A*x;
11     p = P\r;          % risoluzione del sistema Pp = r
12     x = x+p;
13     v(iter) = norm(r);
14     iter = iter+1;
15 end
16
17 v = v(1:iter);       % taglio vettore in base al n. di
       iterazioni
18 end
```

Metodo di Gauss-Seidel:

```
1  function [x, v, iter] = GaussSeidel(A, b, x0, maxit, tol)
2  P = tril(A);          % matrice trangolare inferiore A
3  x = x0;
4  r0 = b-A*x0;
5  v=zeros(maxit, 1);
6  iter = 1;
7  r = r0;
8
9  while iter<maxit && norm(r)>tol*norm(r0)
10     r = b-A*x;
11     p = P\r;
12     x = x+p;
13     v(iter) = norm(r);
14     iter = iter+1;
15 end
16
17 v = v(1:iter);
18 end
```

Notiamo che l'implementazione dei due metodi differisce solo nella scelta della matrice P. La struttura, poi, è la stessa e prevede per ogni iterazione:

- Calcolo del residuo mediante $r = b - Ax$;
- Risoluzione del sistema $Pp = r$;
- Calcolo dell'iterata successiva mediante $x = x + p$.

1.2 Esercizio 2

Il secondo esercizio chiede di applicare i metodi implementati precedentemente alla risoluzione del sistema lineare $Ax = b$ di dati

$$A = \begin{bmatrix} 7 & 6 & 3 \\ 2 & 5 & -4 \\ -4 & -3 & 8 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}$$

e di rappresentare la storia di convergenza tramite un grafico.

Dalla teoria ci possiamo aspettare di avere effettivamente convergenza: questa informazione ci deriva dal calcolo dei raggi spettrali delle matrici di iterazione. Infatti, dato che i metodi di splitting sono sempre consistenti, possiamo affermare che i metodi implementati convergono se e solo se il raggio spettrale delle matrici di iterazione è strettamente minore di

1. Ricordiamo che la matrice B di iterazione nel caso di metodi di splitting si calcola con la formula $B = P^{-1}N$.

Nel caso del metodo di Jacobi, calcoliamo `B = eye(3)-D\A`, dove D è la matrice diagonale di A e risulta

$$B = \begin{bmatrix} 0 & -0.8571 & -0.4286 \\ -0.4000 & 0 & 0.8000 \\ 0.5000 & 0.3750 & 0 \end{bmatrix}$$

Calcolando, poi, il raggio spettrale ρ di B tramite `max(abs(eig(B)))` otteniamo che $\rho = 0.8661 < 1$.

Ripetendo lo stesso procedimento per il metodo di Gauss-Seidel, con la sola differenza che la matrice B si calcola mediante `B = (D-E)\F`, dove D è la matrice diagonale di A, E è la matrice triangolare strettamente inferiore di A con il segno opposto e F è la matrice triangolare strettamente superiore di A con il segno opposto. Risulta, quindi:

$$B = \begin{bmatrix} 0 & -0.8571 & -0.4286 \\ 0 & 0.3429 & 0.9714 \\ 0 & -0.3000 & 0.1500 \end{bmatrix}$$

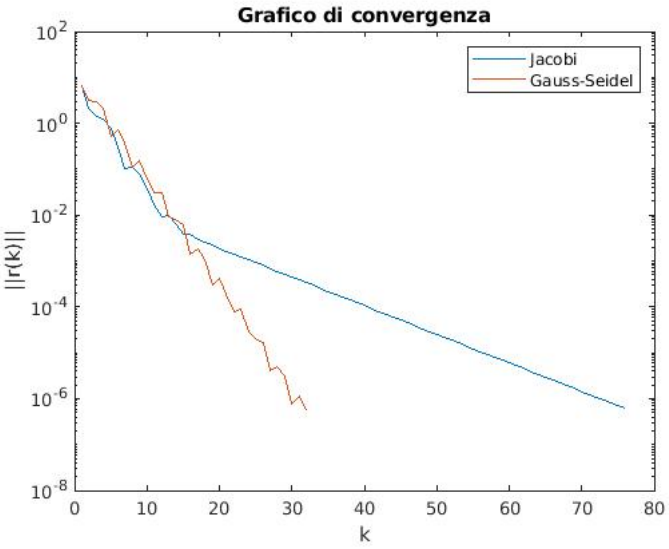
e $\rho = 0.5855 < 1$.

Inoltre, possiamo anche aspettarci che la convergenza non sia monotona nella norma euclidea in quanto la norma euclidea delle matrici di iterazione è maggiore di 1 per entrambi i metodi, 1.1081 e 1.3075 rispettivamente.

Implementazione dell'esercizio:

```
1  A = [7 6 3; 2 5 -4; -4 -3 8];
2  b=[3 -1 2]';
3
4  [~, v1, iter1] = Jacobi(A, b, 0, 1000, 1e-07);
5  t1 = 1:1:iter1;
6  [~, v2, iter2] = GaussSeidel(A, b, 0, 1000, 1e-07);
7  t2 = 1:1:iter2;
8
9  semilogy(t1, v1, t2, v2)
10 title('Grafico di convergenza')
11 xlabel('k')
12 ylabel('||r(k)||')
13 legend('Jacobi', 'Gauss-Seidel')
```

Il grafico che mostra la storia di convergenza è stato realizzato ponendo sull'asse delle x le iterazione e sull'asse delle y le norme euclidee dei residui a ogni iterazione.



Notiamo che effettivamente il grafico conferma le nostre previsioni di convergenza non monotona.

1.3 Esercizio 3

Il terzo esercizio chiede di applicare i due algoritmi al sistema di dati

$$A = \begin{bmatrix} 1 & -1.16 & & \\ 0.16 & \ddots & \ddots & \\ & \ddots & \ddots & -1.16 \\ & & 0.16 & 1 \end{bmatrix}, \quad b = A * \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Le osservazioni che si possono fare sono del tutto analoghe a quelle fatte per l'esercizio precedente: possiamo studiare raggio spettrale e norma euclidea delle matrici di iterazioni per vedere se c'è convergenza e, eventualmente, se questa è monotona.

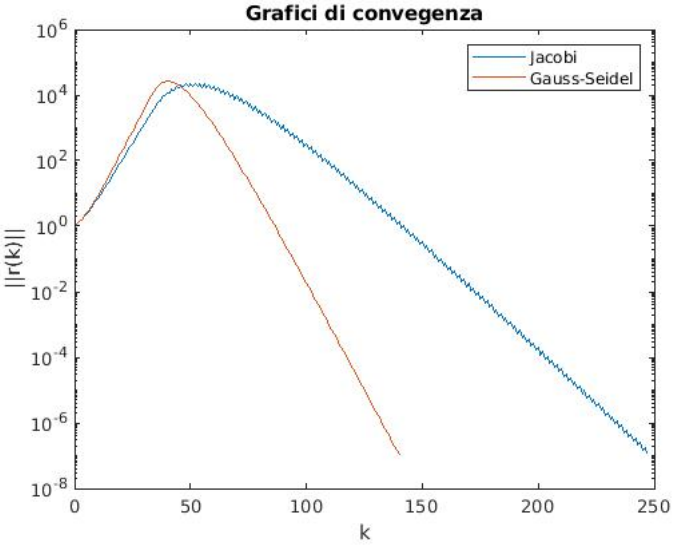
Nel caso del metodo di Jacobi si calcola, come fatto più avanti nell'implementazione, che il raggio spettrale $\rho = 0.8572 < 1$, quindi ci possiamo aspettare che il metodo converga, e che la norma euclidea della matrice è $||B|| = 1.3173 > 1$, il che ci dice che la convergenza non sarà monotona.

Nel caso del metodo di Gauss-Seidel, invece, il raggio spettrale e la norma, calcolati anche in questo caso nell'implementazione dell'esercizio, sono rispettivamente $\rho = 0.7348 < 1$ e $||B|| = 1.3797 > 1$, quindi anche in questo caso avremo convergenza non monotona.

Implementazione dell'esercizio:

```
1  Zu = -1.16*[zeros(29, 1) eye(29); zeros(1, 30)];
2  Zl = 0.16*[zeros(29, 1) eye(29); zeros(1, 30)]';
3  A = eye(30)+Zu+Zl;
4  b = A*ones(30, 1);
5  tol = 1e-07;
6  maxit = 1000;
7  x0 = zeros(30, 1);
8
9  [~, v1, iter1] = Jacobi(A, b, x0, maxit, tol);
10 t1 = 1:1:iter1;
11 [~, v2, iter2] = GaussSeidel(A, b, x0, maxit, tol);
12 t2 = 1:1:iter2;
```

```
13
14 semilogy(t1, v1, t2, v2)
15 title('Grafici di convegenza')
16 xlabel('k')
17 ylabel('||r(k)||')
18 legend('Jacobi', 'Gauss-Seidel')
19
20 D = diag(diag(A));
21 E = -tril(A)+D;
22 F = -triu(A)+D;
23
24 J = eye(30)-D\A;
25 ro_jacobi = max(abs(eig(J)));
26 norm_jacobi = norm(J);
27
28 GS = (D-E)\F;
29 ro_gs = max(abs(eig(GS)));
30 norm_gs = norm(GS);
```



Notiamo che effettivamente il grafico conferma le nostre previsioni di convergenza non monotona.

1.4 Esercizio 4

L'esercizio quattro richiede di scrivere una funzione che implementi il metodo del gradiente coniugato e di usarla per risolvere un sistema dato.

Il metodo del gradiente coniugato è un metodo iterativo per risolvere sistemi lineari simmetrici definiti positivi (SPD). In generale per risolvere sistemi SPD si usano metodi della forma

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}, \quad dove \quad \alpha_k = \frac{p^{(k)t} r^{(k)}}{p^{(k)t} A p^{(k)}}$$

Il metodo del gradiente coniugato si caratterizza per la scelta di

$$p^{(k)} = r^{(k)} + \beta_k p^{(k-1)}, \quad dove \quad \beta_k = -\frac{p^{(k-1)t} A r^{(k)}}{p^{(k-1)t} A p^{(k-1}}$$

intendendo per $r^{(k)}$ il residuo che, nel caso di metodi per sistemi SPD, possiamo calcolare usando la seguente formula che verrà sfruttata anche per l'implementazione:

$$r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}, \quad r^{(0)} = p^{(0)}$$

Per l'implementazione sfruttiamo il fatto che α_k e β_k si esprimono equivalentemente nei seguenti modi:

$$\alpha_k = \frac{||r^{(k)}||_2^2}{p^{(k)t} A p^{(k)}} \tag{1}$$

$$\beta_k = \frac{||r^{(k+1)}||_2^2}{||r^{(k)}||_2^2} \tag{2}$$

Dato il sistema $Ax = b$, la funzione che scriviamo prende in input la matrice A, il vettore b, un dato iniziale $x0$ e, come criteri d'arresto, il numero massimo di iterazioni *maxit* e la tolleranza *tol* sulle norme dei residui. La funzione restituisce, poi, la soluzione approssimata x calcolata con il metodo del gradiente coniugato, il vettore v con le norme dei residui, il numero di iterazioni *iter* e il vettore X contenente tutte le iterate, che servirà per l'esercizio successivo.

Implementazione del metodo del gradiente coniugato:

```
1  function [x, v, iter, X] = CG(A,b,x0,maxit,tol)
2  [n, ~] = size(x0);
3  r0 = b-A*x0;
4  p = r0;
5  r = r0;
6  v = zeros(maxit, 1);
7  v(1) = norm(r0);
8  X = zeros(n, maxit);
9  X(:, 1) = x0;
10 iter = 1;
11 x = x0;
12
13 while iter <= maxit && norm(r) > tol*norm(r0)
14     iter = iter+1;
15     alfa = ((norm(r))^2)/(p'*A*p);
16     x = x+alfa*p;
17     r_old = r;
18     r = r-alfa*A*p;
19     beta = ((norm(r))^2)/(norm(r_old)^2);
20     p = r+beta*p;
21     v(iter) = norm(r);
22     X(:, iter) = x;
23 end
24
25 v = v(1:iter);
26 X = X(:, 1:iter);
27 end
```

Come notiamo, gli step implementati per ogni iterazione sono:

- Calcolo di α con la formula (1);
- Calcolo di x come $x = x + \alpha p$;
- Calcolo di β con la formula (2);

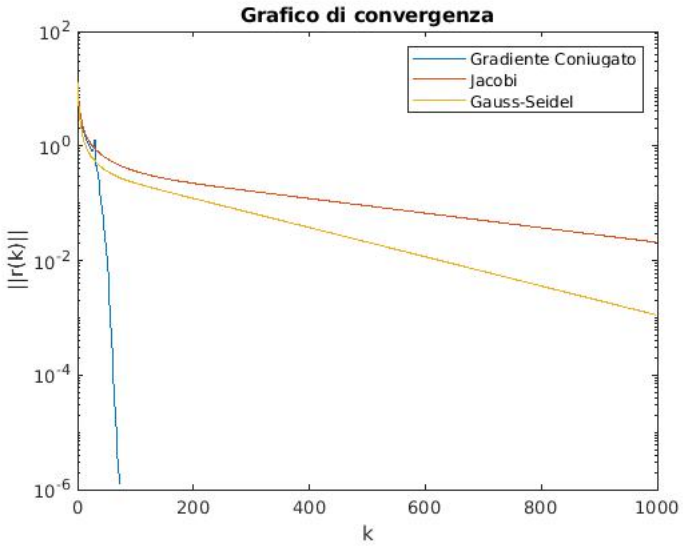
- Calcolo di p come $r + \beta p$;

La seconda parte dell'esercizio chiede, poi, di applicare l'algoritmo alla risoluzione del sistema $Ax = b$, prendendo come A la matrice di Poisson 40 x 40, una matrice sparsa, e $b = A * [1, \ldots, 1]$ e di confrontare graficamente la storia di convergenza con quella ottenuta con i metodi di Jacobi e Gauss-Seidel.

La prima cosa che possiamo notare è che la matrice A essendo una matrice sparsa viene memorizzata da Matlab in maniera differente; infatti, chiamando la matrice A ci vengono indicate solamente le posizioni con entrate non nulle e il valore di tali entrate.

Implementazione dell'esercizio:

```
1  n = 40;
2  A = gallery('poisson', n);
3  b = A*ones(n*n, 1);
4  x0 = zeros(n*n, 1);
5  maxit = 1000;
6  tol = 1e-7;
7
8  [~, vCG, iterCG, ~] = CG(A, b, x0, maxit, tol);
9  tCG = 1:1:iterCG;
10 [~, vJ, iterJ] = Jacobi(A, b, x0, maxit, tol);
11 tJ = 1:1:iterJ;
12 [~, vGS, iterGS] = GaussSeidel(A, b, x0, maxit, tol);
13 tGS = 1:1:iterGS;
14
15 semilogy(tCG, vCG, tJ, vJ, tGS, vGS)
16 title('Grafico di convergenza')
17 xlabel('k')
18 ylabel('||r(k)||')
19 legend('Gradiente Coniugato', 'Jacobi', 'Gauss-Seidel')
```



Notiamo che la convergenza con il metodo del gradiente coniugato è molto più veloce di quella con i metodi di Jacobi e Gauss-Seidel.

1.5 Esercizio 5

L'esercizio richiede di mostrare graficamente la storia di convergenza dell'errore in norma A del metodo del gradiente coniugato per la risoluzione del sistema dell'esercizio precedente con A presa 100 x 100 e confrontarlo con la stima teorica.

Come prima cosa è stato necessario definire la norma A, data come

$$||y||_A = \sqrt{y^t A y}, \quad y \in \mathbb{R}^n \quad A \in \mathbb{R}^{nxn}$$

```
1 function n = normA(y, A)
2 n = sqrt(y'*A*y);
```

La stima teorica che andiamo ora a considerare segue questa formula:

$$\frac{||e^{(k)}||_A}{||e^{(0)}||_A} \leq 2(\frac{\sqrt{K(A)}-1}{\sqrt{K(A)}+1})^k,$$

dove $e^{(k)}$ è l'errore al passo k-esimo e $K(A)$ e il numero di condizionamento relativo del problema $Ax = b$. Ricordiamo, inoltre, che il condizionamento per una matrice simmetrica definita positiva si calcola come

$$K(A) = \frac{\lambda_{max}}{\lambda_{min}},$$

con λ_{max} e λ_{max} autovalore minimo e massimo rispettivamente.

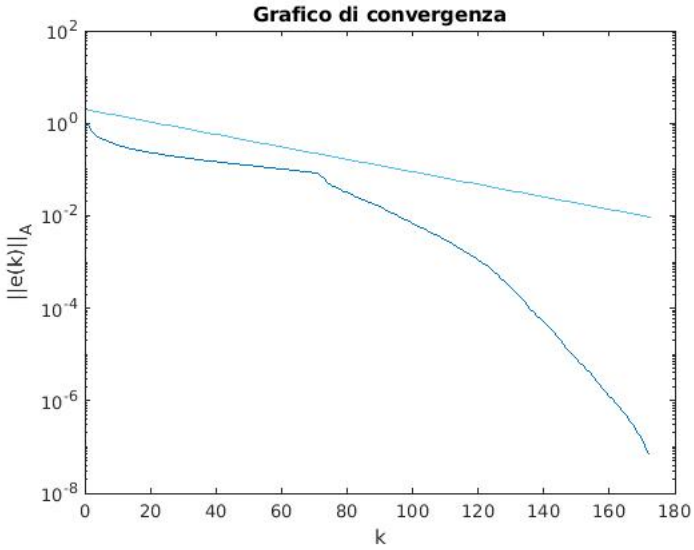
Ci aspettiamo, quindi, un grafico in cui l'errore effettivo sia sempre più piccolo della stima teorica.

L'implementazione è stata fatta tramite un ciclo `for` sul numero di iterazioni abbiamo calcolato a ogni passaggio la norma A dell'errore relativo e il valore della stima teorica in quel ppassaggio e poi rappresentati entrambi in un grafico, ponendo il numero di iterazioni sull'asse x.

Implementazione dell'esercizio:

```
1  n = 100;
2  A = gallery('poisson', n);
3  b = A*ones(n*n, 1);
4  x0 = zeros(n*n, 1);
5  maxit = 1000;
6  tol = 1e-7;
7
8  [x, ~, iter, X] = CG(A, b, x0, maxit, tol);
9  t = 1:1:iter;
10
11 lambda_max = max(eigs(A));
12 lambda_min = eigs(A,1,'smallestabs');
13 K = lambda_max/lambda_min;
14
15 e = zeros(iter);
16 stima = zeros(iter);
17 for k=1:1:iter
18     e(k) = normA(x-X(:, k), A)/normA(x-x0, A);
19     stima(k) = 2*((sqrt(K)-1)/(sqrt(K)+1))^k;
20 end
21
22 semilogy(t, e, t, stima)
23 title('Grafico di convergenza')
```

```
xlabel('k')
ylabel('||e(k)||_A')
```



Notiamo che effettivamente l'errore è sempre più piccolo della stima teorica.

1.6 Esercizio 6

L'ultimo esercizio richiede di risolvere sempre lo stesso sistema $Ax = b$ usando, però, il metodo del gradiente coniugato precondizionato e confrontare il numero di iterazioni necessarie.

Si tratta, in questo caso, di sostituire il problema $Ax = b$ con il problema $PA = Pb$. La matrice P è il precondizionatore e si può scegliere in vari. In questo caso l'esercizio richiede che vengano usati tre precondizionatori:

- Jacobi: $P = D$;
- Gauss-Seidel simmetrico: $P = (D - E)D^{-1}(D - E)^t$;
- Cholesky incompleto: $P =$ `ichol(A)`.

Dove D è la matrice diagonale di A e E la matrice triangolare strettamente inferiore di A con segni opposti.

Per l'implementazione abbiamo usato la funzione built-in di Matlab `pcg`.

Implementazione dell'esercizio:

```
D = diag(diag(A));
E = tril(A, -1);

[~, ~, ~, iterJ] = pcg(A, b, tol, maxit, D);

[~, ~, ~, iterGS] = pcg(A, b, tol, maxit, (D-E)/D, (D-E)');

L = ichol(A);
[~, ~, ~, iterIC] = pcg(A, b, tol, maxit, L, L');
```

Notiamo che le iterazioni ottenute sono le seguenti:

- Gradiente coniugato standard: 173;
- Jacobi: 172;

- Gauss-Seidel: 645;
- Cholesky: 71.

È interessante notare che ciò si accorda perfettamente con la teoria. Possiamo, infatti, stimare la velocità di convergenza osservando il numero di condizionamento delle matrici: la stima dell'errore usata nell'esercizio precedente, infatti, ci dice che la convergenza è tanto più veloce, tanto più è piccolo il condizionamento della matrice del sistema.

Calcolando i numeri di condizionamento di A e di $P^{-1}A$ al variare di P precondizionatore, otteniamo che:

- Il condizionamento di A è molto simile a quello ottenuto con il precondizionatore di Jacobi e sono entrambi dell'ordine di 10^3;
- Il condizionamento con il precondizionatore di Gauss-Seidel è più grande, dell'ordine di 10^4;
- Il condizionamento con il precondizionatore di Cholseky è molto più piccolo ed è circa 4.

2 Integrazione numerica

Nella presente sezione andiamo a trattare le formule di quadratura per l'approssimazione numerica di integrali definiti.

Le formule di quadratura sono formule che approssimano il valore degli integrali definiti nel seguente modo:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} \alpha_i f(x_i)$$

A variare tra i diversi metodi è la scelta dei nodi di quadratura x_i e dei pesi α_i.

2.1 Esercizio 1

Nel primo esercizio si chiede l'implementazione dei metodi di di integrazione composita di punto medio, trapezio e Cavalieri- Simpson.

Questi metodi sono formule interpolatorie: ciò significa che la scelta dei pesi è la stessa ed è

$$\alpha_i = \int_a^b l_i(x)\,dx,$$

dove l_i sono i polinomi della base di Lagrange. A variare tra i vari metodi è il numero di nodi di quadratura, rispettivamente 1, 2 e 3, presi equispaziati all'interno dell'intervallo.

Si chiede di implementare i metodi in integrazione composita: ciò significa che i metodi non sono applicati su tutto l'intervallo $[a, b]$, ma l'intervallo viene iniziale in m sottointervalli di stessa ampiezza su cui viene applicata singolarmente la formula di quadratura.

Le funzioni implementate prendono in input la funzione f da integrare, gli estremi di integrazione a e b e il numero m di sottointervalli per l'integrazione composita.

Metodo del punto medio:

```
1  function I = pto_medio(f, a, b, m)
2  h = (b - a)/m;
3
4  S = 0;
5  for i=1:m
6      x = a + h/2 + (i-1)*h;
7      S = S + f(x);
8  end
9
10 I = h*S;
11 end
```

Metodo del trapezio:

```
1  function I = trapezio(f, a, b, m)
2  h = (b - a)/m;
3
4  S = 0;
5  for i=1:m-1
6      x = a + i*h;
7      S = S + 2*f(x);
8
9  end
10
11 I = h/2 * (f(a) + S + f(b));
12 end
```

Metodo di Cavalieri-Simpson:

```
function I = Cavalieri(f, a, b, m)
h = (b - a)/m;

S = 0;
for i=1:2*m-1
    x = a + i*h/2;

    if rem(i, 2)==0
        S = S + 2*f(x);
    else
        S = S + 4*f(x);
    end

end

I = h/6 * (f(a) + S + f(b));
end
```

L'implementazione in tutti e tre i casi le formule ricavate nella teoria: si calcola prima il nodo successivo x e si somma, poi, $f(x)$ opportunamente pesata; l'approssimazione restituita corrisponde alla somma finale moltiplicata per un peso dipendente dall'ampiezza dei sottointervalli.

L'esercizio chiede, poi, di verificare il grado di esattezza delle formule. Per farlo ci serviamo dell'osservazione che, essendo sia l'integrale che le formule di quadratura lineari, se le formule sono esatte su x^j con j che va da 0 a un certo n, allora il grado di esattezza è almeno n. Basta, poi, verificare la non esattezza per x^{n+1} per dimostrare che il grado è esattamente

La teoria ci dice che il grado di esattezza delle tre formule implementate è rispettivamente 1, 1 e 3. Possiamo, quindi, scrivere uno script che verifica ciò utilizzando l'osservazione fatta sopra.

Verifica del grado di esattezza:

```
f_0 = @(x)(1);
f_1 = @(x)(x);
f_2 = @(x)(x^2);
f_3 = @(x)(x^3);
f_4 = @(x)(x^4);

a = -1;
b = 1;

m = 10000;

eps = 1e-10;

pm_0 = pto_medio(f_0, a, b, m);
pm_1 = pto_medio(f_1, a, b, m);
pm_2 = pto_medio(f_2, a, b, m);

test = [abs(pm_0-2)<eps; abs(pm_1-0)<eps];
if all(test==1)
    disp('Verifica␣punto␣medio␣superata')
end
```

```
22
23 t_0 = trapezio(f_0, a, b, m);
24 t_1 = trapezio(f_1, a, b, m);
25 t_2 = trapezio(f_2, a, b, m);
26
27 test = [abs(t_0-2)<eps; abs(t_1-0)<eps];
28 if all(test==1)
29     disp('Verifica␣trapezio␣superata')
30 end
31
32 c_0 = Cavalieri(f_0, a, b, m);
33 c_1 = Cavalieri(f_1, a, b, m);
34 c_2 = Cavalieri(f_2, a, b, m);
35 c_3 = Cavalieri(f_3, a, b, m);
36 c_4 = Cavalieri(f_4, a, b, m);
37
38 test = [abs(c_0-2)<eps; abs(c_1-0)<eps; abs(c_2-2/3)<eps; abs(
       c_3-0)<eps];
39 if all(test==1)
40     disp('Verifica␣Cavalieri␣superata')
41 end
```

Eseguendo lo script notiamo che effettivamente tutte le verifiche vengono superate.

2.2 Esercizio 2

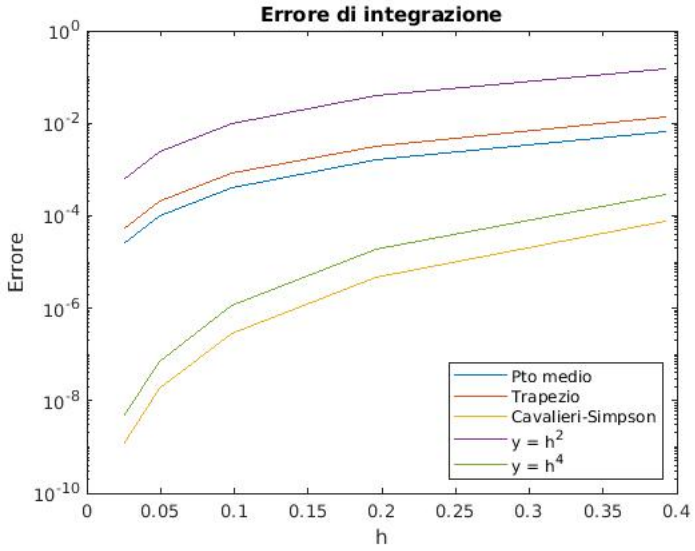
L'esercizio chiede di usare le tre funzioni implementate per approssimare l'integrale e tracciare il grafico degli errori in funzione dell'ampiezza h dei sottointervalli in modo da determinare graficamente l'ordine di convergenza.

Dalla stima dell'errore fornitoci dalla teoria ci possiamo aspettare che il metodo del punto fisso e quello del trapezio abbiano ordine di convergenza 2, mentre il metodo di Cavalieri-Simpson abbia ordine 4.

Implementazione dell'esercizio:

```
1  f = @(x)(x*exp(1)^(-x)*cos(2*x));
2  a = 0;
3  b = 2*pi;
4
5  I = (3*(exp(1)^(-2*pi)-1)-10*pi*exp(1)^(-2*pi))/25;
6
7  % pto medio
8
9  h = zeros(5, 1);
10 E_pm = zeros(5, 1);
11 i = 1;
12 for esp=4:8
13     m = 2^esp;
14     h(i) = (b-a)/m;
15     E_pm(i) = abs(pto_medio(f, a, b, m)-I);
16     i = i+1;
17 end
18
19 % trapezio
20
```

```
h = zeros(5, 1);
E_t = zeros(5, 1);
i = 1;
for esp=4:8
    m = 2^esp;
    h(i) = (b-a)/m;
    E_t(i) = abs(trapezio(f, a, b, m)-I);
    i = i+1;
end

% Cavalieri

h = zeros(5, 1);
E_c = zeros(5, 1);
i = 1;
for esp=4:8
    m = 2^esp;
    h(i) = (b-a)/m;
    E_c(i) = abs(Cavalieri(f, a, b, m)-I);
    i = i+1;
end

semilogy(h, E_pm, h, E_t, h, E_c, h, h.^2, h, (h/3).^4)
legend('Pto medio', 'Trapezio', 'Cavalieri-Simpson', 'y = h^2',
    'y = h^4', 'Location', 'Southeast')
title('Errore di integrazione')
xlabel('h')
ylabel('Errore')
```



Il grafico rappresenta sull'asse delle x l'ampiezza h dell'intervallo per m che raddoppia a ogni passaggio da 0 a 256.

Il risultato è in linea con le aspettative: notiamo, infatti che l'andamento degli errori del metodo del punto medio e del trapezio è lo stesso della funzione h^2, quindi l'ordine di convergenza è 2, mentre il metodo di Cavalieri-Simpson si comporta come h^4, quindi ha ordine 4.

Notiamo, anche, che l'andamento crescente dei grafici degli errori è del tutto giustificato dal fatto che sull'asse delle x è stato posto h. Ricordiamo, infatti, che $h = \frac{b-a}{m}$, quindi per h grandi, il numero di iterazioni m è piccolo, il che significa che la precisione diminuisce, e quindi l'errore aumenta, all'aumentare di h. Peraltro la convergenza si ha per per h che tende a 0.

2.3 Esercizio 3

L'esercizio 3 chiede di calcolare tramite le formule del punto medio e di Cavalieri-Simpson il numero minimo di sottointervalli per approssimare con un errore minore di 10^{-4} l'integrale

$$\int_0^5 \frac{1}{1+(x-\pi)^2}\,dx$$

Per farlo si usa la stima a posteriori dell'errore che ci permette di calcolare l'errore per una formula a $n+1$ punti con $2m$ sottointervalli nel seguente modo:

$$\frac{1}{2^{n+p}-1}(I_{n,2m}(f) - I_{n,m}(f)), \tag{3}$$

con $I_{n,m}(f)$, la formula di quadratura composita a $n+1$ punti su m intervalli; p vale 2 se n pari, 1 se dispari.

Noi utilizzeremo la formula per $n = 0$, nel caso del trapezio, $n = 2$, nel caso di Cavalieri-Simpson. Come m partiremo da 0 e raddoppieremo fino all'ottenimento dell'accuratezza richiesta.

Implementazione dell'esercizio:

```
1  f = @(x)(1/(1+(x-pi)^2));
2  a = 0;
3  b = 5;
4  eps = 1e-4;
5
6  E_pm = 100;
7  esp = -1;
8  n = 0;
9  p = 2;
10 while E_pm > eps
11     esp = esp+1;
12     m = 2^esp;
13     I_2m = pto_medio(f, a, b, 2*m);
14     I_m = pto_medio(f, a, b, m);
15     E_pm = abs((1/(2^(n+p)-1))*(I_2m - I_m));
16 end
17 disp(2*m)
18
19 E_c = 100;
20 esp = -1;
21 n = 2;
22 p = 2;
23 while E_c > eps
24     esp = esp+1;
25     m = 2^esp;
26     I_2m = Cavalieri(f, a, b, 2*m);
27     I_m = Cavalieri(f, a, b, m);
28     E_c = abs((1/(2^(n+p)-1))*(I_2m - I_m));
29 end
```

```
30 disp(2*m)
```

Nell'implementazione il risultato che viene visualizzato è $2m$, poiché la formula (3) restituisce l'errore su $2m$ intervalli.

Il risultato restituito è di 64 iterazioni per la formula del punto fisso e 16 per quella di Cavalieri-Simpson. Ciò è in accordo con il fatto che l'ordine di convergenza del metodo di Cavalieri-Simpson, che è 4, è due volte maggiore dell'ordine di convergenza del metodo di punto medio, che è 2.

2.4 Esercizio 4

L'esercizio 4 richiede di implementare la formula di quadratura di Gauss a due punti sull'intervallo $[-1, 1]$ e di verificarne il grado di esattezza.

Una formula di quadratura di Gauss a n punti è una formula di quadratura avente grado esattezza massimo, cioè avente grado $2n + 1$. Si dimostra che essa si può definire equivalentemente come una formula del tipo

$$\sum_{i=1}^{n} \alpha_i f(x_i)$$

avente come nodi x_i le radici di un polinomio di grado n ortogonale a P_{n-1}, insieme dei polinomi di grado minore o uguale a $n - 1$.

A tal proposito si sfrutta la formula ricorsiva per definire i polinomi $p_n \in P_n$ ortogonali a P_{n-1}:

$$p_{-1}(x) = 0;$$
$$p_0(x) = 1;$$
$$p_{k+1}(x) = xp_k(x) - \beta_k p_{k-1}(x), \quad con \quad \beta_k = \frac{(p_k, p_k)_w}{(p_{k-1}, p_{k-1})_w}.$$

Nel nostro caso dobbiamo calcolare p_2. È immediato vedere che $p_1 = x$. Risulta, quindi,

$$p_2(x) = x^2 - \beta_1$$

Resta da calcolare β_1:

$$(p_1, p_1)_w = \int_{-1}^{1} x^2 \, dx = \frac{2}{3}$$
$$(p_0, p_0)_w = \int_{-1}^{1} 1 \, dx = 2$$

Da cui,

$$\beta_1 = \frac{1}{3}$$

E quindi:

$$p_2(x) = x^2 - \frac{1}{3}$$

Risulta immediato vedere che le radici di p_2 sono $\pm\frac{1}{\sqrt{3}}$.

Per calcolare i pesi α_i sfruttiamo il fatto che una formula di Gauss su due punti ha grado di esattezza 3 e imponiamo l'esattezza della formula sui polinomi di grado 0 e 1.

$$\alpha_0 + \alpha_1 = \int_{-1}^{1} dx = 2$$

$$\alpha_0(-\frac{1}{\sqrt{3}}) + \alpha_1(\frac{1}{\sqrt{3}}) = 0$$

E quindi è immediato vedere che

$$\alpha_0 = 1, \quad \alpha_1 = 1.$$

Noti, quindi, gli x_i e gli α_i, possiamo implementare la formula di quadratura che avrà la forma:

$$I_2(f) = f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}})$$

Implementazione della formula:

```
function I = gauss(f)
I = f(-1/sqrt(3)) + f(1/sqrt(3));
end
```

Scriviamo, ora, per verificare che il grado di esattezza sia proprio 3 uno script analogo a quello fatto nell'esercizio 1 per le altre formule di quadratura.

Verifica del grado di esattezza:

```
f_0 = @(x)1;
I_0 = 2;
Ig_0 = gauss(f_0);
eps = 1e-15;
disp(abs(I_0 - Ig_0) < eps)

f_1 = @(x)x;
I_1 = 0;
Ig_1 = gauss(f_1);
disp(abs(I_1 - Ig_1) < eps)

f_2 = @(x)x^2;
I_2 = 2/3;
Ig_2 = gauss(f_2);
disp(abs(I_2 - Ig_2) < eps)

f_3 = @(x)x^3;
I_3 = 0;
Ig_3 = gauss(f_3);
disp(I_3 == Ig_3)
disp(abs(I_3 - Ig_3) < eps)

f_4 = @(x)x^4;
I_4 = 2/5;
Ig_4 = gauss(f_4);
disp(abs(I_4 - Ig_4) < eps)
```

Eseguendo lo script risulta che si ha esattezza fino all'ordine 3, come previsto

2.5 Esercizio 5

L'esercizio 5 richiede di fornire degli esempi di integrali in cui la funzione integranda non soddisfi le ipotesi di convergenza ottimale per i metodi implementati nell'esercizio 1.

Le condizioni di convergenza ottimale ci sono garantite da alcune stime teoriche sull'errore. Queste stime, tuttavia, richiedono come ipotesi una certa regolarità della funzione integranda:

in particolare la funzione deve essere C^2 nel caso delle formule del punto medio e del trapezio e di classe C^4 nel caso della formula di Cavalieri-Simpson. Possiamo, quindi, cercare come esempi delle funzioni che non possiedano la regolarità richiesta e verificare che l'ordine di convergenza dei metodi è effettivamente più basso.

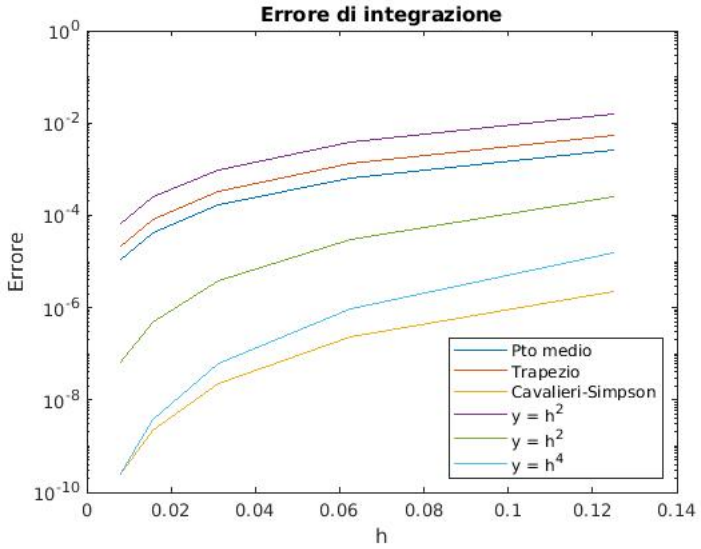
Il primo esempio è la funzione $f(x) = x^{\frac{7}{3}}$, che è C^2, ma non C^3. Ci possiamo, quindi aspettare che i metodi del punto medio e del trapezio abbiano sempre ordine di convergenza 2, mentre il metodo di Cavalieri-Simpson abbia ordine minore di 4.

Lo verifichiamo graficamente sfruttando lo script dell'esercizio 2 opportunamente modificato.

Implementazione del primo esempio:

```
f = @(x)(x^(7/3));
a = -1;
b = 1;

I = 3/10 - (3/10)*(-1)^(10/3);


% pto medio

h = zeros(5, 1);
E_pm = zeros(5, 1);
i = 1;
for esp=4:8
    m = 2^esp;
    h(i) = (b-a)/m;
    E_pm(i) = abs(pto_medio(f, a, b, m)-I);
    i = i+1;
end

% trapezio

h = zeros(5, 1);
E_t = zeros(5, 1);
i = 1;
for esp=4:8
    m = 2^esp;
    h(i) = (b-a)/m;
    E_t(i) = abs(trapezio(f, a, b, m)-I);
    i = i+1;
end

% Cavalieri

h = zeros(5, 1);
E_c = zeros(5, 1);
i = 1;
for esp=4:8
    m = 2^esp;
    h(i) = (b-a)/m;
    E_c(i) = abs(Cavalieri(f, a, b, m)-I);
    i = i+1;
end

```

```
44 semilogy(h, E_pm, h, E_t, h, E_c, h, h.^2, h, (h/2).^3, h, (h
       /2).^4)
45 legend('Pto medio', 'Trapezio', 'Cavalieri-Simpson', 'y = h^2',
       'y = h^4', 'Location', 'Southeast')
46 title('Errore di integrazione')
47 xlabel('h')
48 ylabel('Errore')
```



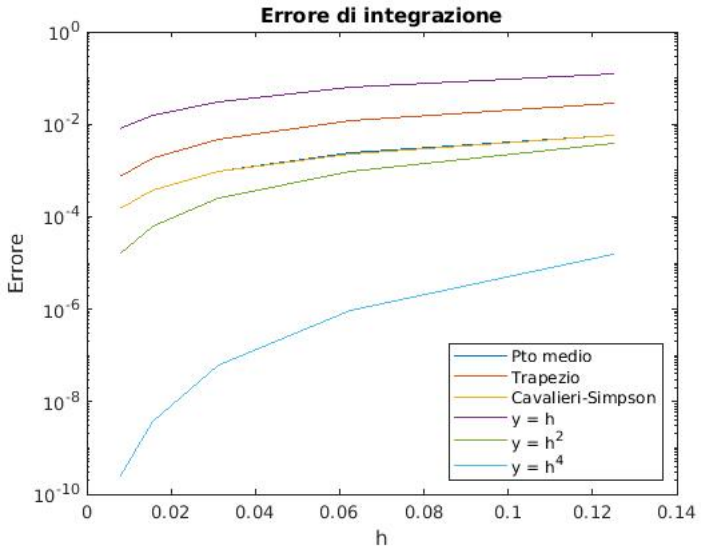
Il grafico rispecchia le nostre previsioni. Infatti, il metodo del punto medio e il metodo del trapezio mantengono ordine di convergenza 2, mentre il metodo di Cavalieri-Simpson ha ordine di convergenza 3, quindi minore del 4 che avrebbe in condizioni di regolarità.

Portiamo anche un secondo esempio di funzione che sia C^1, ma non C^2. In questo caso una funzione da prendere può essere $f(x) = x^{\frac{1}{3}}$. Ci aspettiamo che nessuno dei metodi implementati mantenga lo stesso ordine di convergenza che avrebbe in condizioni di regolarità.

Implementazione del secondo esempio:

```
1  f = @(x)(x^(1/3));
2  a = -1;
3  b = 1;
4
5  I = 3/4 - 3*(-1)^(4/3)/4;
6
7
8  % pto medio
9
10 h = zeros(5, 1);
11 E_pm = zeros(5, 1);
12 i = 1;
13 for esp=4:8
14     m = 2^esp;
15     h(i) = (b-a)/m;
16     E_pm(i) = abs(pto_medio(f, a, b, m)-I);
17     i = i+1;
18 end
19
```

```
20 % trapezio
21
22 h = zeros(5, 1);
23 E_t = zeros(5, 1);
24 i = 1;
25 for esp=4:8
26     m = 2^esp;
27     h(i) = (b-a)/m;
28     E_t(i) = abs(trapezio(f, a, b, m)-I);
29     i = i+1;
30 end
31
32 % Cavalieri
33
34 h = zeros(5, 1);
35 E_c = zeros(5, 1);
36 i = 1;
37 for esp=4:8
38     m = 2^esp;
39     h(i) = (b-a)/m;
40     E_c(i) = abs(Cavalieri(f, a, b, m)-I);
41     i = i+1;
42 end
43
44 semilogy(h, E_pm, h, E_t, h, E_c, h, h, h, (h/2).^2, h, (h/2)
       .^4)
45 legend('Pto␣medio', 'Trapezio', 'Cavalieri-Simpson', 'y␣=␣h', '
       y␣=␣h^2', 'y␣=␣h^4', 'Location', 'Southeast')
46 title('Errore␣di␣integrazione')
47 xlabel('h')
48 ylabel('Errore')
```



Il grafico dell'errore del metodo del punto medio si sovrappone con quello di Cavalieri-Simpson risultando poco visibile.

Notiamo che effettivamente l'ordine di convergenza di tutti i metodi implementati è uno, quindi in tutti i casi minore dell'ordine che si avrebbe in caso di regolarità, che è 2 per i metodi del punto medio e del trapezio e 4 per il metodo di Cavalieri-Simpson.