Analisi numerica

Relazione di Laboratorio

Tiziano Bertoncello, Marco Ambrogio Bergamo, Chiara Catalano

Anno 2023-2024

Indice

Elenco dei listati

Parte I
Metodi diretti per sistemi lineari

Esercizio 1

Esercizio. Scrivere una funzione `tri_solve` che prenda in input una matrice triangolare A e un vettore b e calcoli la soluzione del sistema lineare $Ax = b$. Tale funzione deve gestire sia il caso triangolare superiore che triangolare inferiore (si usino eventualmente le funzioni `istriu` e `istril`).

```
function[x] = tri_solve(A, b)
[n, m] = size(A); %verifico che A sia nxn
x = zeros(n, 1); %inizializzo un vettore colonna n-dim

if n~= m
    warning('Solo matrici quadrate')
    return
end

if min(abs(diag(A))) == 0       %controllo che nessun elemento sulla
    error('Sistema singolare')  %diagonale sia nullo
end

 if istriu(A) % Se la matrice A è triangolare superiore
        x(n) = b(n) / A(n, n);
        for j = n-1:-1:1  %diminuisce con passo -1 fino a 1
            x(j) = (b(j) - A(j, j+1:n) * x(j+1:n)) / A(j, j);
        end

elseif istril(A) % Se la matrice A è triangolare inferiore
    x(1) = b(1) / A(1, 1);
    for j = 2:n
        x(j) = (b(j) - A(j, 1:j-1) * x(1:j-1)) / A(j,j);
    end

else
    x = NaN;
    fprintf('Non ho un sistema triangolare')
end

end
```

Listato 1: Funzione tri_solve

Commento. La funzione controlla di avere a che fare con un sistema triangolare e gestisce sia il caso della triangolare superiore sia quella inferiore, restituendo il vettore colonna della soluzione.

Esercizio 2

Esercizio. Scrivere due funzioni, `lu_nopiv` e `lu_piv`, che calcolino la fattorizzazione LU senza e con pivoting di una matrice, rispettivamente

```
function [L, U] = lu_nopiv(A)
[n, m] = size(A);

if n~=m
    error('A non è una matrice quadrata')
end

U = A;        %inizializzo U e L
L = eye(n);
for k = 1:n-1
    if A(k,k) == 0   %controllo che gli el. pivotali siano non nulli
        error('Elemento pivotale nullo')
    end
    for i = k+1:n   %nelle righe successive alla kesima, definisco l_ik
        L(i,k) = U(i,k) / U(k,k);    %e sottraggo alla iesima riga
        U(i, k:n) = U(i, k:n) - L(i,k)*U(k, k:n);  %l_ik*kesima riga
    end
end
end
```

Listato 2: lu_nopiv

Commento. Per ogni riga k (che va da 1 a $n-1$) si controlla se l'elemento pivotale (diagonale) $A(k,k)$ sia non nullo, se è zero, viene generato un errore. Definisco il moltiplicatore l_{ik} al variare di i dalla riga $k+1$ a n e sottraggo alle righe dalla $k+1$ a n della matrice U (che è la A trasformata di volta in volta) $l_{ik} \cdot k$-esima riga.

```
function [P, L, U, num] = lu_piv(A)
[n, m] = size(A);

if n~=m
    error('A non è una matrice quadrata')
end

U = A;      %inizializzo le matrici P,L,U
L = eye(n);
P = eye(n);
num = 0; %numero di scambi righe per il calcolo del det(P)

for k = 1:n-1
    [value, index] = max(abs(U(k:n,k))); %indice riga dell'elemento max
    index = index + k -1;               %"value" a_ik della kima colonna
    if index ~= k
        num = num+1; %aggiorno il numero di scambi

        U([index, k], k:n) = U([k, index], k:n); %scambio per P,L,U la porzione
        L([index, k], 1:k-1) = L([k, index], 1:k-1); %di riga interessata
        P([index, k], :) = P([k, index], :);   %nel pivoting

    end


    for i = k+1:n    %per il resto eseguo lu_nopiv
        L(i, k) = U(i, k) / U(k, k);
        U(i, k:n) = U(i, k:n) - L(i, k) * U(k, k:n);
    end
end
end
```

Listato 3: lu_piv

Commento. Utilizziamo pivoting parziale perché comporta costo computazionale minore rispetto a quello totale. Per evitare errori di arrotondamento si sceglie come elemento pivotale l'elemento di modulo massimo della colonna $A(k)(k:n,k)$. Nel codice troviamo sia il `value` che il suo indice riga, indicato con `index`. A questo punto correggiamo l'indice e continuiamo l'implementazione del codice in cui invertiamo le porzioni di riga coinvolte nel pivoting ed eseguiamo fattorizzazione LU senza pivoting nel resto.

Esercizio 3

Esercizio. Scrivere una funzione che, presa in input una matrice $A \in \mathbb{R}^{n \times n}$ non singolare, ne calcoli l'inversa usando la sua fattorizzazione LU.

```
function [X] = lu_inv(A)
[n,m] = size(A);

if n~=m
    error('A non è una matrice quadrata')
end

if det(A) == 0
    error('Matrice A singolare')
end

[P, L, U] = lu_piv(A);  %eseguo la fattorizzazione LU con pivoting
X = zeros(n);   %inizializzo l'inversa

for i = 1:n
    y = tri_solve(L, P(:,i));  %per tutti i vettori colonna risolvo il sist.
    X(:, i) = tri_solve(U, y); %triangolare Ly_i = Pe_i e Ux_i = y_i
end
```

Listato 4: lu_inv

Commento. Chiamiamo X la matrice inversa e sfruttiamo la fattorizzazione $PA = LU$. Le colonne della matrice X sono tali da risolvere il duplice sistema triangolare $Ly_i = Pe_i$ e $Ux_i = y_i$, per le quali utilizziamo la funzione già implementata `tri_solve`.

Esercizio 4

Esercizio. Scrivere una funzione che calcola il determinante di una matrice A usando la fattorizzazione $PA = LU$. Per calcolare il determinate di P, si modifichi la funzione scritta al punto 2) in modo da contare il numero di scambi di righe effettuati

```
function [d] = determinante(A)

[P, L, U, num] = lu_piv(A);


d= (-1)^num*prod(diag(U)); %uso della funzione prod presente in matlab
                           %per il prodotto degli elementi del vettore
end
```

Listato 5: Determinante

Commento. Il $\det(A) = (-1)^{\delta} \prod_{i=1}^{n} u_{ii}$. Dato che U è una matrice triangolare superiore, il suo determinante si calcola come prodotto degli elementi sulla diagonale estratti dal comando `diag(U)`; ne calcoliamo il prodotto con `prod()` e moltiplichiamo per $(-1)^{\square}$ ($\square$ = numero di scambi righe in P).

Esercizio 5

Esercizio. Scrivere due funzioni, `sgs` e `mgs`, che calcolino la fattorizzazione QR ridotta di una matrice, usando rispettivamente il metodo di Gram-Schmidt e di Gram-Schmidt modificato.

```
function [Q, R] = sgs(A)
[m,n] = size(A); %matrice mxn
R = zeros(n); %inizializzo R, mentre costruisco Q man mano


for j = 1:n     %per ogni colonna j considero le colonne i < j
    Q(:, j) = A(:, j);
    for i = 1:j-1
        R(i,j) = Q(:,i)'*A(:, j); %prodotto scalare, altrimenti con dot
        Q(:,j) = Q(:,j) - R(i,j)*Q(:,i);
    end
    R(j,j) = norm(Q(:,j), 2);  %norma della colonna Q_j
    Q(:, j) = Q(:, j) / R(j,j);  %e normalizzo la colonna in Q
end
end
```

Listato 6: Fattorizzazione QR con Gram-Schmidt

```
function [Q, R] = mgs(A)
[m,n] = size(A); %matrice mxn
R = zeros(n);  %inizializzo Q e R
Q = A;

for i = 1:n    %per ogni colonna i
    R(i,i) = norm(Q(:,i), 2);    %ne calcolo la norma euclidea
    Q(:, i) = Q(:, i) / R(i,i);  % e la normalizzo
    for j= i+1:n                 %considero le colonne j > i
        R(i,j) = Q(:,i)'*Q(:,j);   %prod scalare tra Q_i e Q_j
        Q(:,j) = Q(:, j) - R(i,j)*Q(:,i);  %aggiorno Q_j
    end
end
end
```

Listato 7: Fattorizzazione QR con Gram-Schmidt modificato

Commento. Entrambi i codici implementano il metodo di Gram-Schmidt per la fattorizzazione QR di una matrice A. Il primo esegue il metodo standard, inizializzando una matrice R da 0 e costruendo Q colonna per colonna: la prima colonna è la prima colonna di A normalizzata, mentre a seguire ogni nuova colonna viene costruita imponendo l'ortogonalità con tutte le precedenti e normalizzata.
Lo svantaggio di questo metodo è l'instabilità, per questo viene usato il metodo di Gram-Schmidt modificato: ogni volta che si calcola una proiezione la si sottrae immediatamente riducendo così l'accumulo di errori numerici.

Esercizio 6

Esercizio. Sia

- $m = 50, n = 12$
- $f(t) = \cos(4t)$
- Si definiscano i punti $t_i = \frac{i-1}{m-1} \quad i = 1, \ldots, m$
- b il vettore di componenti $b_i = f(t_i)$
- A la matrice del problema ai minimi quadrati che si ottiene approssimando con un polinomio di grado $n-1$ la sequenza di punti $(t_i, b_i) \quad i = 1, \ldots, m$

Si risolva e si visulizzi (con tutti i 16 decimali) la soluzione x del problema, calcolata con i seguenti metodi:

1. Formazione e soluzione del sistema di equazioni normali, usando il comando MATLAB `\`.
2. Fattorizzazione QR calcolata con `sgs`.
3. Fattorizzazione QR calcolata con `mgs`.
4. Fattorizzazione QR calcolata con `qr` (funzione built-in di Matlab che utilizza il metodo di Householder).

I calcoli precedenti generano quattro liste di 12 coefficienti. Prendendo come soluzione di riferimento quella ottenuta con l'ultimo metodo, si cancellino in ogni lista le cifre decimali che appaiono errate (affette da errori di arrotondamento). Quali metodi si mostrano più instabili?

```
m = 50;
n = 12;

f = @(x) cos(4*x);
t = linspace(0, 1, m);
b = f(t)';
A = zeros(m, n);
for i = 1:n
    A(:,i) = (t.^(i-1))';
end

 C = zeros(n, 4);  %matrice con le 4 liste di coefficienti
%devo risolvere tAAx = tAb

%soluzione col comando \ di Matlab
c_1 = (A'*A) \ (A'*b);
C(:, 1) = c_1;

%soluzione tramite fatt. QR con sgs
[Q, R] = sgs(A);
y = Q' *b;
c_2 = tri_solve(R, y);
%x_2 = linsolve(R, Q' *b);
C(:, 2) = c_2;

%soluzione tramite fatt. QR con mgs
[Q, R] = mgs(A);
y = Q' *b;
c_3 = tri_solve(R, y);
%x_3 = linsolve(R, Q' *b);
C(:, 3) = c_3;

%soluzione qr
[Q, R] = qr(A);
y = Q' *b;
c_4 = linsolve(R, y);
C(:, 4) = c_4;

%matrice delle differenze
D = abs ( C(: , 1:3) - repmat ( C(: , 4) , 1 , 3) );
                        %crea una matrice 1x3 contenente la 4 colonna di C
                        % ripetuta nelle 3 colonne


format long
```

```
disp('Coefficienti')
disp(C)
disp("Matrice delle differenze tra i coefficienti dei vari metodi e quelli qr")
disp(D)
```

Listato 8: Esercizio_6

Commento. Lo script calcola le soluzioni c_i al problema risolvendo per i quattro metodi `\`, `sgs`, `mgs` e `qr` i sistemi lineari

$$A^T \cdot A \cdot c_i = A^T \cdot b \implies \cancel{R^T} \cdot \cancel{Q^T} \cdot \cancel{Q} \cdot R \cdot c_i = \cancel{R^T} \cdot Q^T \cdot b \implies \boxed{Rc_k = Q^T b}$$

Lo script resituisce dunque in output la matrice $C = [c_1\ c_2\ c_3\ c_4]$ con nelle colonne i coefficienti calcolati con i diversi metodi

$$C = \begin{bmatrix} 0.999999980992711 & 1.000013182519530 & 0.999999998520678 & 1.000000000996605 \\ 0.000005276190967 & -0.002257207967712 & 0.000000305080723 & -0.000000422742855 \\ -8.000192129130278 & -7.939131604863162 & -8.000008537233985 & -7.999981235690834 \\ 0.002753271593689 & -0.651916745053513 & 0.000083096216035 & -0.000318763166674 \\ 10.646135038183751 & 14.271195521078631 & 10.666357171895211 & 10.669430795444505 \\ 0.090461942277623 & -11.567831187500790 & 0.000038863810680 & -0.013820286072909 \\ -5.940682754250898 & 17.068086616660683 & -5.686340458426423 & -5.647075632526641 \\ 0.459108364221840 & -27.857195008923703 & -0.003456843263779 & -0.075316015199692 \\ 1.065546102285269 & 22.365631340117201 & 1.608753414490871 & 1.693606953048568 \\ 0.466150171806121 & -8.657750217256378 & 0.068459156451883 & 0.006032116225040 \\ -0.565318882477130 & 1.289403471510039 & -0.400264201059355 & -0.374241706482950 \\ 0.122390009747441 & 0.028098490547490 & 0.092734415580960 & 0.088040576606074 \end{bmatrix}$$

da cui, considerando le differenze delle prime tre colonne dalla quarta, abbiamo $D = [|c_1 - c_4|\ |c_2 - c_4|\ |c_3 - c_4|\ ]$

$$D = \begin{bmatrix} 0.000000020003895 & 0.000013181522925 & 0.000000002475928 \\ 0.000005698933822 & 0.002256785224857 & 0.000000727823578 \\ 0.000210893439444 & 0.060849630827672 & 0.000027301543151 \\ 0.003072034760363 & 0.651597981886839 & 0.000401859382709 \\ 0.023295757260755 & 3.601764725634126 & 0.003073623549295 \\ 0.104282228350532 & 11.554010901427882 & 0.013859149883588 \\ 0.293607121724257 & 22.715162249187323 & 0.039264825899782 \\ 0.534424379421533 & 27.781878993724010 & 0.071859171935913 \\ 0.628060850763299 & 20.672024387068632 & 0.084853538557697 \\ 0.460118055581082 & 8.663782333481418 & 0.062427040226843 \\ 0.191077175994180 & 1.663645177992989 & 0.026022494576405 \\ 0.034349433141367 & 0.059942086058584 & 0.004693838974886 \end{bmatrix}$$

Vediamo che il metodo di Gram-Schmidt standard è più instabile del metodo modificato, in accordo con la teoria.

Parte II
Zeri di funzioni

Esercizio 1

Esercizio. Implementare il metodo di bisezione e di Newton-Raphson e testare le function per la ricerca di una delle radici della funzione $f(x) = e^x - x^2 - \sin x - 1$ in $[-2, 2]$.

```
function[c,fc,iter] = Bisezione(f,a,b,tol,itmax)
    c = a;
    fc = f(a);
    iter = 0;

    for i = 1:itmax
        if abs(fc) < tol
            break
        else
            c = (a+b)/2;
            fc = f(c);
            iter = i;

            if f(a)*f(c) < 0
                b = c;
            else
                a = c;
            end
        end
    end
```

Listato 9: Bisezione

```
%testo la funzione bisezione che ho scritto
f = @(x)(exp(x)-(x.^2)-sin(x)-1);

fplot(f,[-2,2]); grid on
[c,fc,iter] = Bisezione(f,1,1.5,1e-12,100)  %non ho messo il ';' per far stampare a terminale
    i risultati
```

Listato 10: Prova bisezione

```
function [alpha, fa, iter] = NewtonRaphson(f,df,x0,tol,itmax)
    delta = 1;
    for i = 0:itmax
        alpha = x0;
        fa = f(x0);
        iter = i;
        if abs(delta) <= tol | df(x0) == 0    %nota: il metodo potrebbe sbagliare se la
    funzione ha un punto stazionario diverso dalla radice in accordo con la teoria
            break
        else
            delta = -fa/(df(x0));
            x0 = x0+delta;
        end
    end
end
```

Listato 11: Newton-Raphson

```
f = @(x)(exp(x)-(x.^2)-sin(x)-1);
df = @(x)(exp(x)-(2*x)-cos(x));

fplot(f,[-2,2]); grid on
[c,fc,iter] = NewtonRaphson(f,df,2,1e-12,100)
```

Listato 12: Prova Newton-Raphson

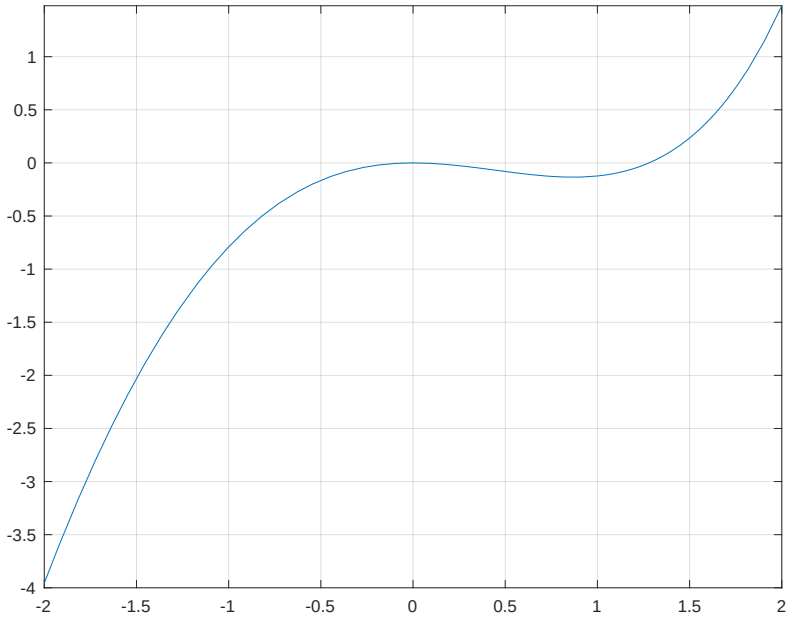
Commento. Vediamo che ricercando l'unica soluzione all'interno dell'intervallo $[1, 1.5]$, partendo da 1, con il metodo di bisezione otteniamo un risultato entro la tolleranza fissata in 37 iterazioni. Tale radice è

$$c = 1.279701331000979 \qquad f(c) = -1.298960938811433e^{-14}$$

Con il metodo di Newton-Raphson, invece, con le medesime premesse, otteniamo un risultato accettabile in sole 7 iterazioni, con risultato

$$c = 1.279701331000996 \qquad f(c) = -1.110223024625157e^{-16}$$

Osserviamo graficamente che i risultati sono corretti:



Esercizio 2

Esercizio. Si consideri la funzione $f(x) = (x-1)\log x$. Si applichino il metodo di Newton-Raphson ed il metodo di Newton-Raphson modificato per la ricerca della radice doppia $\alpha = 1$ e si confrontino graficamente gli ordini di convergenza dei due metodi.

```
function [alpha, fa, iter] = NewtonRaphsonMod(f,df,x0,tol,itmax)
    delta = 1;
    for i = 0:itmax
        alpha = x0;
        fa = f(x0);
        iter = i;
        if abs(delta) <= tol | df(x0) == 0                    %nota: il metodo potrebbe
    sbagliare se la funzione ha un punto stazionario diverso dalla radice in accordo con la
    teoria
            break
        else
            delta = (-2)*(fa/(df(x0)));
            x0 = x0+delta;
        end
    end
end
```

Listato 13: Newton-Raphson modificato

```
f = @(x)((x-1)*(log(x)));
df = @(x)((x-1)/x + log(x));

fplot(f,[0.5,5]); grid on
[c,fc,iter] = NewtonRaphsonMod(f,df,5,1e-12,10)
```

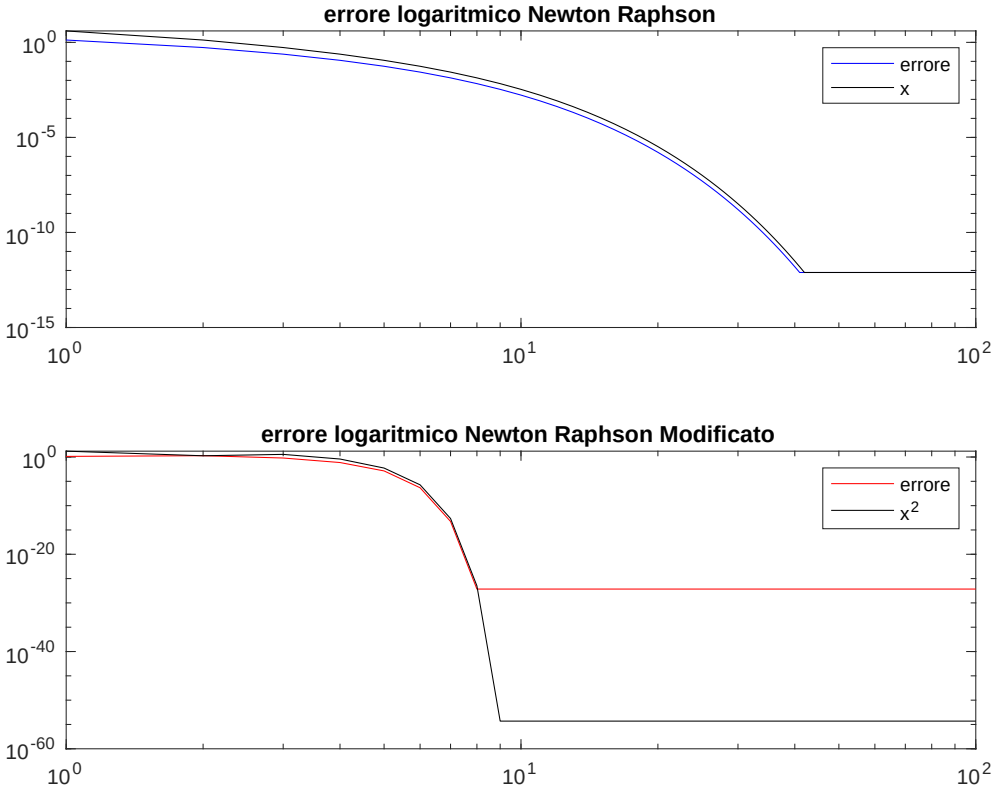
Listato 14: Prova Newton-Raphson modificato

Commento. Come nell'esercizio precedente, verifichiamo che il metodo implementato funzioni.

```
f = @(x)((x-1)*(log(x)));
df = @(x)((x-1)/x + log(x));

%inizializzo le liste
ErrNR = [];
ErrNRmod = [];
xnorm = [];
xmod = [];
x = [];

%riempio le liste in funzione di itmax

for i = 1:100
    ErrNR(i) = abs(1-NewtonRaphson(f,df,5,1e-12,i));
    ErrNRmod(i) = abs(1-NewtonRaphsonMod(f,df,5,1e-12,i));
    xnorm(i) = abs(1-NewtonRaphson(f,df,5,1e-12,i-1));
    xmod(i) = abs(1-NewtonRaphsonMod(f,df,5,1e-12,i-1));
    x(i) = i;
end

%faccio i grafici
figure
tiledlayout(2,1)
loglog(nexttile,x,ErrNR,'-b');
hold on
loglog(x,xnorm,'-k')
title('errore logaritmico Newton Raphson')
hold off
legend({'errore', 'x'})

loglog(nexttile,x,ErrNRmod, '-r')
hold on
loglog(x,xmod.^2, '-k')
title('errore logaritmico Newton Raphson Modificato')
hold off
legend({'errore', 'x^2'})
```

Listato 15: Ordini di convergenza

Commento. Confrontiamo graficamente gli ordini di convergenza dei due metodi nella ricerca della radice doppia e osserviamo, in accordo con la teoria, che il metodo di Newton-Raphson modificato è più veloce, essendo di ordine 2 rispetto al metodo standard, che è di ordine 1.
I grafici, che rappresentano l'andamento dell'errore rispetto alle iterazioni compiute, diventano stazionari dopo aver raggiunto la tolleranza fissata.



Parte III
Integrazione

Esercizio 1

Esercizio. Si implementino le formule di quadratura composite del punto medio, dei trapezi, di Cavalieri-Simpson e di Gauss-Legendre a due nodi. Si scriva la seguente function: `I=quadratura(f,a,b,m,metodo)`. In seguito testare la function quadratura calcolando gli integrali:

$$\int_{-1}^{2} x^4 dx = \frac{33}{5}, \qquad \int_{-\pi/2}^{\pi/2} \cos x dx = 2, \qquad \int_{0}^{1} e^x dx = e - 1$$

```
function integrale = IntegraPuntoMedio (f, a, b, m)
    x = linspace (a, b, 2*m+1);
    h = (b-a)/m;
    integrale = h*sum(f(x(2:2:2*m)));
    end
```

Listato 16: Punto medio

```
function integrale = IntegraTrapezi(f, a, b, m)
    x = linspace (a, b, m+1);
    h = (b-a)/m;
    if m >= 2
        integrale = (h/2)*(f(a)+2*sum(f(x(2:m)))+f(b));
    else
        integrale = (h/2)*(f(a)+f(b));
    end
```

Listato 17: Trapezi

```
function integrale = IntegraCavalieriSimpson (f, a, b, m)
    h = (b-a)/m;
    x = linspace(a, b, 2*m+1);
    if m >= 2
        integrale = (h/6)*(f(a)+2*sum(f(x(3:2:2*m-1)))+4*sum(f(x(2:2:2*m)))+f(b));
    else
        integrale = (h/6)*(f(a)+4*f((a+b)/2)+f(b));
    end
```

Listato 18: Cavalieri-Simpson

```
function integrale = IntegraGaussLegendre(f,a,b)
    integrale = (b-a)*(0.5)*(f((b-a)*(2*sqrt(3))^(-1)+(a+b)/2)+f((b-a)*(-2*sqrt(3))^(-1)+(a+b)
    /2));
end
```

Listato 19: Gauss-Legendre

```
function I = Quadratura(f,a,b,m,metodo)
    if metodo == 1
        I = IntegraPuntoMedio(f,a,b,m);
    elseif metodo == 2
        I = IntegraTrapezi(f,a,b,m);
    elseif metodo == 3
        I = IntegraCavalieriSimpson(f,a,b,m);
    elseif metodo == 4
        I = IntegraGaussLegendre(f,a,b);
    else
        error("inserire un metodo di integrazione valido");
    end
```

Listato 20: Function `quadratura`

```
%definisco le funzioni
f = @(x)(x.^(4));
g = @(x)(cos(x));
h = @(x)(exp(x));

%provo la funzione quadratura con la prima funzione
```

```
i = 1;
disp("Al variare del metodo di integrazione, l'integrale della prima funzione risulta:");
while i < 5
    disp(Quadratura(f,-1,2,100,i));
    i = i+1;
end
%provo la funzione quadratura con la seconda funzione
j = 1;
disp("Al variare del metodo di integrazione, l'integrale della seconda funzione risulta:");
while j < 5
    disp(Quadratura(g,-pi/2, pi/2, 100,j));
    j = j+1;
end
%provo la funzione quadratura con la terza funzione
k = 1;
disp("Al variare del metodo di integrazione, l'integrale della terza funzione risulta:");
while k < 5
    disp(Quadratura(h,0,1,100,k));
    k = k+1;
end
```

Listato 21: Calcolo degli integrali

Commento. Dopo aver implementato i quattro metodi, definiamo la function `Quadratura`, che ci permette di calcolare l'integrale approssimato di una funzione scegliendo gli estremi di integrazione, il numero di suddivisioni dell'intervallo e il metodo, tra quelli implementati. Successivamente usiamo la function `Quadratura` per calcolare gli integrali approssimati delle funzioni date.
I risultati degli integrali, al variare del metodo, sono:

	Prima funzione	Seconda funzione	Terza funzione
Punto medio	6.598650070875000	2.000082249070986	1.718274668972308
Trapezi	6.602699919000000	1.999835503887444	1.718296147450418
Cavalieri-Simpson	6.600000020250000	2.000000000676472	1.718281828465011
Gauss-Legendre	5.250000000000003	1.935819574651137	1.717896378007504

Esercizio 2

Esercizio. Si scriva uno script per valutare al variare di m (numero di intervalli della suddivisione) e del metodo usato, l'errore di integrazione $E_{\text{metodo},m}$:

$$E_{\text{metodo},m} = |I(f) - \mathcal{I}_{\text{metodo},m}(f)|$$

Si riporti in un grafico in scala logaritmica l'errore in funzione di m.

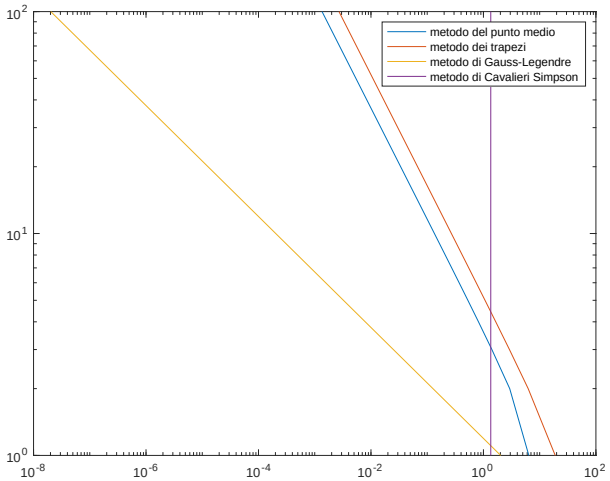
```
%definisco la funzione E che mi dà l'errore in funzione del metodo e di
%m, assumo di usare x^4
function E = Erroref(metodo,m)
    E = abs((33/5) - Quadratura(@(x)(x.^(4)),-1,2,m,metodo));
end
```

Listato 22: Errore su f

```
m = (1:100);

EMediof = [];
ETrapf = [];
EGLf = [];
ECSf = [];

Intf = 33/5;

for i = 1:100
    EMediof(i) = abs(Intf - Quadratura(@(x)(x.^(4)),-1,2,i,1));
    ETrapf(i) = abs(Intf - Quadratura(@(x)(x.^(4)),-1,2,i,2));
    EGLf(i) = abs(Intf - Quadratura(@(x)(x.^(4)),-1,2,i,3));
    ECSf(i) = abs(Intf - Quadratura(@(x)(x.^(4)),-1,2,i,4));
end

%creo il grafico per gli errori relativo alla prima funzione
figure
title("Errori in scala logaritmica per la prima funzione")
plot(EMediof, m);
hold on
plot(ETrapf,m);
plot(EGLf,m);
plot(ECSf,m);
legend(["metodo del punto medio", "metodo dei trapezi", "metodo di Gauss-Legendre", "metodo di
    Cavalieri Simpson"])
set(gca , 'XScale', 'log', 'YScale', 'log') ;
hold off
```

Listato 23: Grafici degli errori

Commento. Dopo aver definito la funzione `Erroref`, che trova l'errore nel calcolo dell'integrale della prima funzione al variare del metodo e del numero di suddivisioni, riportiamo i risultati ottenuti:



Avendo implementato il metodo di Gauss-Legendre a due nodi, quindi indipendente dal numero di suddivisioni, il grafico relativo è costante.

Parte IV
Equazioni differenziali ordinarie

Esercizio 1

Esercizio. Si scrivano due funzioni Matlab che implementino i metodi di Eulero esplicito e di Heun per la risoluzione del seguente problema di Cauchy:

$$\begin{cases} y'(t) = f(t, y(t)) \quad t \in [t_0, t_f] \\ y(t_0) = y_0 \end{cases}$$

Tali funzioni devono prendere in input

t_0	istante iniziale
t_f	istante finale
y_0	dato iniziale
f	funzione
N	numero di sottointervalli

e restituire un vettore $\boldsymbol{u} = (u_0, \dots, u_N)$ con i valori u_j, $j = 0, \dots, N$ della soluzione approssimata nei nodi.

```
function u = EuleroEsplicito(a, b, y0, f, N)
    u = [y0];
    h = (b-a)/N;
    for j = 1:N
        u(j+1) = u(j) + h*f(a+(j-1)*h, u(j));
    end
end
```

Listato 24: Metodo di Eulero esplicito

```
function u = Heun(a, b, y0, f, N)
    u = [y0];
    h = (b-a)/N;
    for j = 1:N
        x = f(a+(j-1)*h, u(j));
        u(j+1) = u(j) + h*0.5*(x + f(a+j*h, u(j)+h*x));
    end
end
```

Listato 25: Metodo di Heun

Esercizio. Studiare graficamente l'ordine di convergenza dell'errore

$$\max_j |u_j - y(t_j)|$$

risolvendo il seguente problema di Cauchy

$$\begin{cases} y'(t) = \sin(t)(1 + \cos(t) - y(t)) \quad t \in [0, 1] \\ y(0) = 3 \end{cases}$$

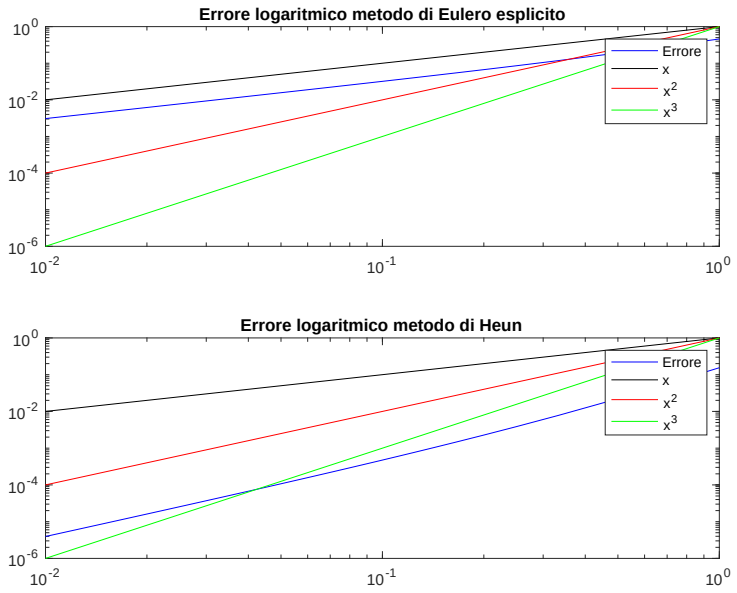
la cui soluzione esatta è $y(t) = 2 + \cos(t)$

```
f = @(t,y)(sin(t)*(1+cos(t)-y));
y = @(t)(2 + cos(t));

a = 0;
b = 1;
y0 = 3;

%faccio i grafici

figure
tiledlayout(2,1)

%eulero esplicito
e = [];
```

```
x = [];
for i = 1:100
    N = i;
    h = (b-a)/N;
    x(i) = h;
    t = linspace(a,b,N+1);
    u = y(t)-EuleroEsplicito(a,b,y0,f,N);
    e(i) = max(abs(u));
end

plot(nexttile, x, e, '-b');
hold on
plot(x,x,'-k');
plot(x,x.^2,'-r')
plot(x,x.^3,'-g');
title('Errore logaritmico metodo di Eulero esplicito')
legend({'Errore', 'x', 'x^2', 'x^3'});
set(gca, 'XScale', 'log', 'YScale', 'log');
hold off

%Heun
e = [];
x = [];
for i = 1:100
    N = i;
    h = (b-a)/N;
    x(i) = h;
    t = linspace(a,b,N+1);
    u = y(t) - Heun(a,b,y0,f,N);
    e(i) = max(abs(u));
end

plot(nexttile,x,e,'-b');
hold on
plot(x,x,'-k');
plot(x,x.^2,'-r');
plot(x,x.^3,'-g');
title('Errore logaritmico metodo di Heun')
legend({'Errore', 'x', 'x^2', 'x^3'});
set(gca, 'XScale', 'log','YScale', 'log');
hold off
```

Listato 26: Ordine di convergenza dell'errore (esplicito)

Commento. In accordo con la teoria risulta che il metodo di Eulero esplicito è del primo ordine mentre quello di Heun è di ordine 2.



Esercizio 2

Esercizio. Si svolga l'esercizio precedente considerando i metodi di Eulero implicito e di Crank-Nicolson. In entrambi i metodi si approssimi u_{j+1} con un metodo di tipo punto fisso, precisamente

$$\text{Eulero implicito: } \begin{cases} z^{(0)} = u_j + hf(t_j, u_j) \\ z^{(k+1)} = u_j + h(f(t_{j+1}, z^{(k)}) \end{cases}$$

$$\text{Crank-Nicolson: } \begin{cases} z^{(0)} = u_j + hf(t_j, u_j) \\ z^{(k+1)} = u_j + \frac{h}{2}(f(t_j, u_j) + f(t_{j+1}, z^{(k)})) \end{cases}$$

Si usino, come criterio d'arresto per il metodo di punto fisso, le condizioni

$$\frac{\left|z^{(k+1)} - z^{(k)}\right|}{\left|z^{(k+1)}\right|} < \text{tol} \qquad \text{e} \qquad k < \text{maxit}$$

(si fissi ad esempio $\text{tol} = 10^{-6}$, $\text{maxit} = 100$).

```
function u = EuleroImplicito(a,b,y0,f,N,tol,maxit)
    h = (b-a)/N;   %definisco il passo h
    u = [y0]; % inizializzo la lista di valori
    %riempio la lista
    for j = 1:N
        g = @(x)(u(j) + h*f(a+(j)*h,x));%funzione per applicare il metodo di punto fisso
        %applico il metodo di punto fisso
        zk = u(j);
        for i = 1:maxit
            zk = g(zk);
            if abs((zk-g(zk))/g(zk))<tol
                break
            end
        end
        u(j+1) = zk; %aggiungo alla lista il valore trovato con il metodo di punto fisso
    end
end
```

Listato 27: Eulero implicito

```
function u = CrankNicolson(a,b,y0,f,N,tol,maxit)
    h = (b-a)/N; %definisco il passo tra due nodi
    u = [y0]; %inizializzo la lista di valori
    %riempio la lista
    for j =  1:N
        g = @(x)(u(j)+h*(0.5)*(f(a+(j-1)*h,u(j))+f(a+j*h,x))); %funzione per applicare il
    metodo di punto fisso
        %applico il metodo di punto fisso
        zk = u(j);
        for i = 1:maxit
            zk = g(zk);
            if abs((zk-g(zk))/g(zk)) < tol
                break
            end
        end
        u(j+1) = zk; %aggiungo alla lista il valore trovato
    end
end
```

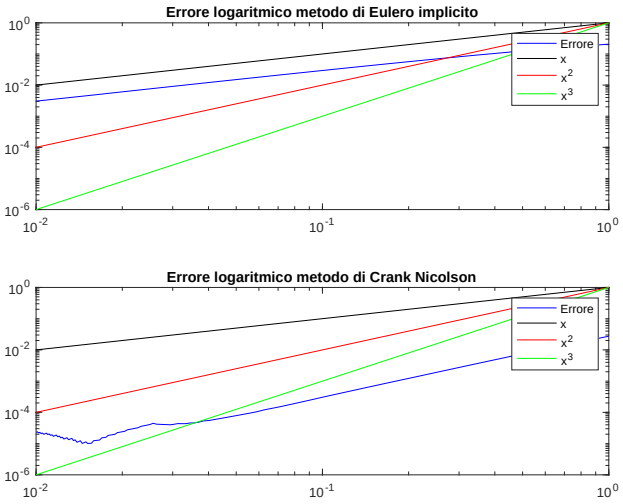
Listato 28: Crank-Nicolson

```
f = @(t,y)(sin(t)*(1+cos(t)-y));
y = @(t)(2 + cos(t));

a = 0;
b = 1;
y0 = 3;
tol = 10^(-6);
maxit = 100;

%faccio i grafici

figure
tiledlayout(2,1)
```

```
14
15 %Eulero Implicito
16 e = [];
17 x = [];
18 for i = 1:100
19     N = i;
20     h = (b-a)/N;
21     x(i) = h;
22     t = linspace(a,b,N+1);
23     u = y(t)-EuleroImplicito(a,b,y0,f,N,tol,maxit);
24     e(i) = max(abs(u));
25 end
26
27 plot(nexttile, x, e, '-b');
28 hold on
29 plot(x,x,'-k');
30 plot(x,x.^2,'-r')
31 plot(x,x.^3,'-g');
32 title('Errore logaritmico metodo di Eulero implicito')
33 legend({'Errore', 'x', 'x^2', 'x^3'});
34 set(gca, 'XScale', 'log', 'YScale', 'log');
35 hold off
36
37 %Crank Nicolson
38 e = [];
39 x = [];
40 for i = 1:100
41     N = i;
42     h = (b-a)/N;
43     x(i) = h;
44     t = linspace(a,b,N+1);
45     u = y(t) - CrankNicolson(a,b,y0,f,N,tol,maxit);
46     e(i) = max(abs(u));
47 end
48
49
50 plot(nexttile,x,e,'-b');
51 hold on
52 plot(x,x,'-k');
53 plot(x,x.^2,'-r');
54 plot(x,x.^3,'-g');
55 title('Errore logaritmico metodo di Crank Nicolson')
56 legend({'Errore', 'x', 'x^2', 'x^3'});
57 set(gca, 'XScale', 'log','YScale', 'log');
58 hold off
```

Listato 29: Ordine di convergenza dell'errore (implicito)

Commento. In accordo con la teoria risulta che il metodo di Eulero implicito è del primo ordine mentre quello di Crank-Nicholson è di ordine 2.



Esercizio 3

Esercizio. Si applichi il metodo di Eulero esplicito al seguente problema di Cauchy

$$\begin{cases} y'(t) = -5y(t) & t \in [0, 8] \\ y(0) = 1 \end{cases}$$

per $N = 19, 20, 21$ e per ciascun caso si riportino sullo stesso grafico la soluzione esatta e quella approssimata. Si commentino i risultati ottenuti.

```
f = @(t,y)((-5)*y);
y = @(t)(exp((-5)*t));

a = 0;
b = 8;
y0 = 1;

N1 = 19;
N2 = 20;
N3 = 21;

figure
tiledlayout(3,1)
%per N1:
approx = EuleroEsplicito(a,b,y0,f,N1);
x = linspace(a,b,N1+1);
esatt = y(x);

plot(nexttile, x, esatt, '-b');
hold on
plot(x, approx, '-k');
title('grafico per N = 19');
legend({'soluzione esatta', 'soluzione approssimata'});

hold off

%per N2:
approx = EuleroEsplicito(a,b,y0,f,N2);
x = linspace(a,b,N2+1);
esatt = y(x);

plot(nexttile, x, esatt, '-b');
hold on
plot(x, approx, '-k');
title('grafico per N = 20');
legend({'soluzione esatta', 'soluzione approssimata'});

hold off

%per N3:
approx = EuleroEsplicito(a,b,y0,f,N3);
x = linspace(a,b,N3+1);
esatt = y(x);

plot(nexttile, x, esatt, '-b');
hold on
plot(x, approx, '-k');
title('grafico per N = 21');
legend({'soluzione esatta', 'soluzione approssimata'});

hold off
```

Listato 30: Grafici approssimati

Commento. Al variare di N, uso il metodo di Eulero esplicito per creare la successione che approssima la soluzione del problema di Cauchy dato e la rappresento nel grafico. Notiamo che per $N = 19$, il metodo diverge; per N = 20, non converge; per $N = 21$, converge.

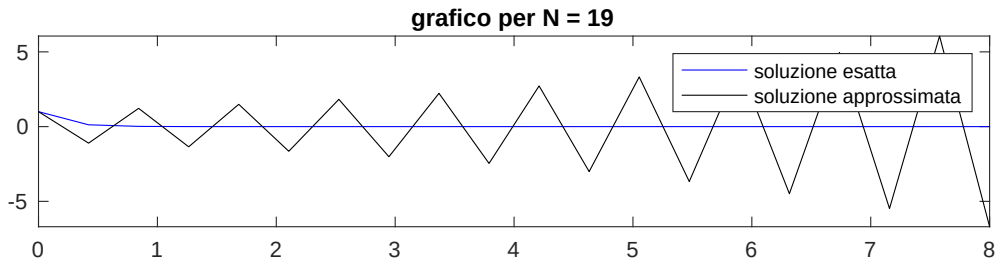


grafico per N = 19
soluzione esatta
soluzione approssimata
5
0
-5
0
1
2
3
4
5
6
7
8

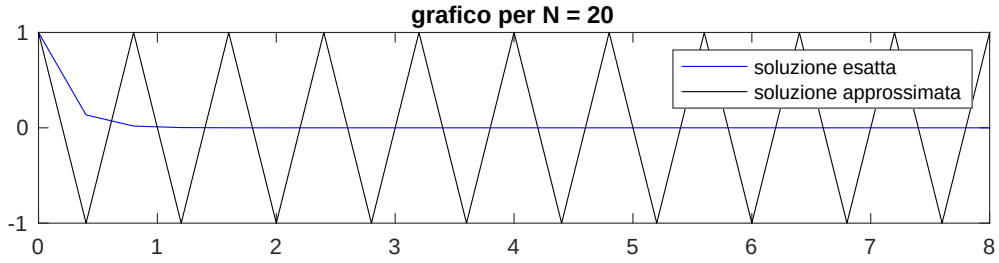


grafico per N = 20
soluzione esatta
soluzione approssimata
1
0
-1
0
1
2
3
4
5
6
7
8

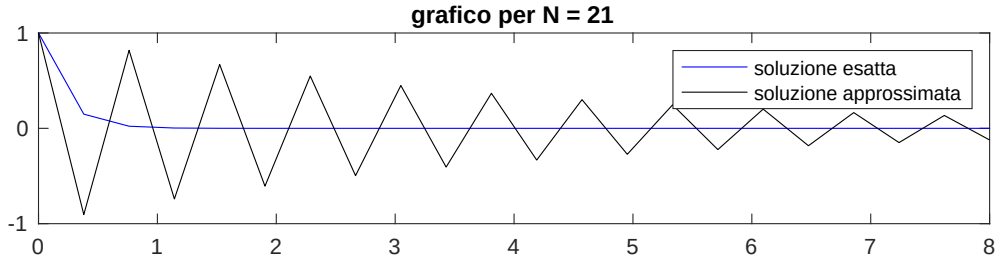


grafico per N = 21
soluzione esatta
soluzione approssimata
1
0
-1
0
1
2
3
4
5
6
7
8