# Attività Matlab per Crediti D

Matteo Marinelli

March 5, 2021

## Indice



## 1 Note

Per la risoluzione dei problemi di test ho deciso di scrivere uno script, per ognuno di questi, che sia di carattere generale; cioè uno script composto di due fasi: una di settaggio dei parametri del problema ed una di risoluzione algoritmica del problema stesso sui dati immessi nella fase precedente. In tal modo sarà poi possibile riusare lo stesso script per la risoluzione del problema su dati differenti (Semplicemente, immetto i dati opportuni all'esecuzione corrente nella fase di settaggio parametri).

Le soluzioni di tutti e tre i problemi di test sono dei codici MATLAB che ho sviluppato e che sono visibili nella sezione 'Appendice' della presente relazione.
Si avvisa il lettore circa la lunghezza degli script appena detti, poiché ciò potrebbe rendere la relazione non di facile (piacevole) lettura, anche se 'facili' sono gli esercizi che gli script risolvono.

## 2 Problema 1

La densità con cui sono distribuiti i nodi d'interpolazione all'interno del nostro intervallo $[0, 1]$ varia in modo graduale con l'allontanarsi dallo 0 degli assi del grafico riportato in figura 1. Da quello che è possibile vedere, infatti, i primi 3 nodi sono tra loro molto vicini, mentre i restanti sono distribuiti in maniera tale che la distanza dal nodo loro più prossimo tenda ad aumentare con l'aumento della loro distanza dallo 0. Questo "vincola" il polinomio interpolante ad essere in qualche modo molto vicino alla funzione $f(x)$ nella prima parte del grafico, mentre porta, man mano che ci si allontana dalla regione iniziale, ad avere come effetto contrario quello che il polinomio stesso tenda a sfuggire sempre di più questa vicinanza che ha con la funzione $f(x)$.
La parte drammatica di questo discorso è raggiunta in prossimità del punto 0.9 del nostro del grafico, in cui la distanza tra polinomio interpolante $p$ e funzione $f(x)$ supera le 10 unità.

Per apprezzare meglio gli errori possiamo guardare la tabella 2 corrispondente alla parte dell'output dedita alla risoluzione del punto (a), restituita dallo script sviluppato.

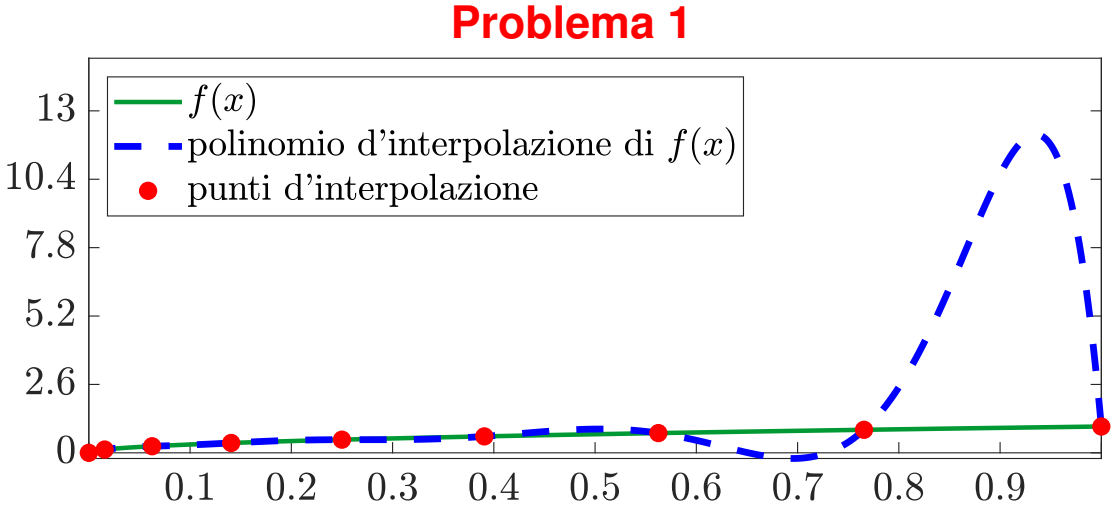


Figura 1: Grafico richiesto dal punto (b) del problema 1.

Questa tabella delle differenze, infatti, raggiunge il suo massimo (in valore assoluto) di 10.731478361986419 in penultima posizione, corrispondente al punto $\zeta_{20} = \frac{19}{20} = 0.95$

| differenze |
|---|
| 0 |
| 0.009373456935820 |
| −0.016624898598359 |
| 0.006265159516694 |
| 0.026059100541981 |
| −0.000000000000000 |
| −0.046798842893449 |
| −0.052843679514482 |
| 0.019043791981465 |
| 0.136657922266043 |
| 0.195969221000565 |
| 0.070222900207983 |
| −0.298665479678432 |
| −0.793827451939212 |
| −1.047857448417112 |
| −0.461689802877413 |
| 1.600121563949947 |
| 5.337600132745780 |
| 9.648720381277604 |
| 10.731478361986419 |
| −0.000000000000199 |

Un'ultima nota: lo script per la risoluzione del problema test 1 è stato congegnato in modo da produrre in automatico una figura che sia presentabile come quella inserita in questa relazione, utilizzando i comandi per l'abbellimento automatico dei grafici. Questo può essere comodo quando si vuole eseguire lo script senza avere la presente relazione, e quindi la relativa figura, a portata di mano; infatti, la figura prodotta in automatico dallo script sarà tale da non mostrare alcun bisogno di essere abbellita manualmente nel relativo editor delle figure.

# 3 Problema 2

Prima di procedere con i dettagli relativi alle soluzioni dei punti del problema 2, vorrei porre l'accento sulle modalità con cui questi risultati sono stati raggiunti. In particolare, ho modificato il codice che implementa la formula dei trapezi, nonché il codice della funzione *nepsilon*, in modo che queste due function possano ricevere in input vettori, e restituire in output dei vettori di risultati corrispondenti all'esecuzione dell'algoritmo standard su ciascuno degli elementi di tali/tale vettori/vettore. Ad esempio, per quel che riguarda la formula dei trapezi ho deciso di pormi nel caso generale in cui il quarto parametro della function corrisponde in realtà ad un vettore di interi positivi, relativi agli ordini della formula dei trapezi con cui si vuole approssimare l'integrale definito della funzione $f(x)$ (parametro 1), nell'intervallo di estremi a (parametro 2) e b (parametro 3). Ci tengo a sottolineare ciò in quanto il codice, ad una prima occhiata, potrebbe risultare poco leggibile, soprattutto per il fatto che dietro ad un assegnamento del tipo

```
I_ns = formulaTrapezi(@(x)exp(x), a, b, ns);
```

si nascondono in realtà, in un'unica esecuzione della function chiamata, più esecuzioni della function in versione standard e dunque più risultati da memorizzare nella variabile

```
I_ns
```

Venendo a noi, la risoluzione del punto (a) consiste nella function *nepsilon* sviluppata all'interno dello script relativo al problema test 2 e utilizzata per la risoluzione del punto (b).
La tabella richiesta dal punto (b), invece, è mostrata di seguito ( nonché riprodotta in automatico dallo script ad ogni sua esecuzione, mediante il tipo di dato che MATLAB ci mette a disposizione per le rappresentazioni in forma tabellare)

| $n(\epsilon)$ | $I_{n(\epsilon)}$ | $I$ | $\lvert I - I_{n(\epsilon)}\rvert$ | $\epsilon$ |
|---|---|---|---|---|
| 2 | 1.75393109246483 | 1.71828182845905 | 0.0356492640057799 | 0.1 |
| 5 | 1.72400561978279 | 1.71828182845905 | 0.00572379132374246 | 0.01 |
| 16 | 1.71884112857999 | 1.71828182845905 | 0.000559300120948958 | 0.001 |
| 48 | 1.71834397651311 | 1.71828182845905 | 6.2148054068345e-0 | 0.0001 |
| 151 | 1.71828810844886 | 1.71828182845905 | 6.2799898117305e-06 | 1e-05 |
| 476 | 1.71828246043305 | 1.71828182845905 | 6.31974002462954e-07 | 1e-06 |
| 1506 | 1.71828189159303 | 1.71828182845905 | 6.31339851508983e-08 | 1e-07 |
| 4760 | 1.71828183477879 | 1.71828182845905 | 6.31974050868678e-09 | 1e-08 |
| 15051 | 1.71828182909114 | 1.71828182845905 | 6.32092156394037e-10 | 1e-09 |
| 47595 | 1.71828182852224 | 1.71828182845905 | 6.31910079817999e-11 | 1e-10 |

Come si può vedere dalla figura, si hanno (ovviamente) migliori approssimazioni dell'integrale per valori alti dell'ordine della formula dei trapezi con cui si decide di approssimare. Si vede inoltre come sia $|I - I_{n(\epsilon)}| \leq \epsilon$ per ognuno degli $\epsilon$ riportati nella relativa colonna, come giustamente deve essere! (altrimenti avremmo ottenuto un errore nella risoluzione del problema).

Per quello che riguarda il punto (c), invece, abbiamo la seguente tabella

| $I_2$ | $I_4$ | $I_8$ | $I_{16}$ | $I$ |
|---|---|---|---|---|
| 1.75393109246483 | 1.72722190455752 | 1.7205185921643 | 1.71884112857999 | 1.71828182845905 |

Ancora una volta è possibile osservare come all'aumentare dell'ordine della formula dei trapezi aumenti l'accuratezza dell'approssimazione fornita.

Per il punto (d), invece, abbiamo la tabella seguente

| | |
|---|---|
| $I_2$ | 1.75393109246483 |
| $I_4$ | 1.72722190455752 |
| $I_8$ | 1.7205185921643 |
| $I_{16}$ | 1.71884112857999 |
| $I$ | 1.71828182845905 |
| $p(0)$ | 1.71828182846039 |

Quello che scopriamo è che $p(0)$ è la migliore delle approssimazioni fornite per l'integrale

$$\int_0^1 e^x \, dx$$

Notiamo infatti una precisione esatta fino alla $10^a$ cifra decimale!

# 4 Problema 3

Per risolvere il punto (a) del problema ho usato (banalmente) il comando
MATLAB 'backslash'

```
x = A\b
```

Il valore esatto della soluzione del sistema lineare $Ax = b$ è stato poi usato per fornire la tabella richiesta nel punto (b).
Venendo al punto (b), si può osservare, guardando la tabella seguente, come il metodo di Jacobi (con vettore d'innesco nullo) stia per convergere alla soluzione esatta del sistema lineare detto sopra già dopo le prime 10 iterazioni.

| $x^{(0)}$ | $x^{(1)}$ | $x^{(2)}$ | $x^{(3)}$ | $x^{(4)}$ | $x^{(5)}$ | $x^{(6)}$ | $x^{(7)}$ | $x^{(8)}$ | $x^{(9)}$ | $x^{(10)}$ | $x$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2.6 | 1.2095 | 0.8971 | 0.9535 | 1.0038 | 1.0054 | 1.0005 | 0.9995 | 0.9998 | 1.0000 | 1 |
| 0 | 2.2857 | 2.3238 | 2.0163 | 1.9698 | 1.9925 | 2.0019 | 2.0011 | 1.9999 | 1.9998 | 1.9999 | 2 |
| 0 | 2.3333 | 3.0952 | 3.1079 | 3.0054 | 2.9899 | 2.9975 | 3.0006 | 3.0003 | 2.9999 | 2.9999 | 3 |

Tabella 5: problema 3, punto (b): Ho troncato le approssimazioni alla $4^a$ cifra decimale per ragioni di spazio.

Per quanto riguarda il punto (c), invece, veniva chiesto di costruire una tabella che, per ogni precisione $\epsilon \in \{10^{-1}, ..., 10^{-10}\}$, riportasse, tra le altre cose, il numero $K_\epsilon$ di iterazioni necessarie al metodo di Jacobi per convergere entro la precisione $\epsilon$. Siccome ci veniva richiesto di risolvere il problema utilizzando i programmi MATLAB implementati per la risoluzione degli esercizi proposti a lezione, e siccome tra questi avevo definito la function che implementa il metodo di Jacobi in maniera tale che prendesse in input:

- A: una matrice quadrata
- b: un vettore di termini noti relativo al sistema lineare da risolvere
  Ax = b
- toll: una soglia di precisione relativa alla condizione d'arresto della
  function rispetto al criterio d'arresto del residuo con tolleranza, appunto, 'toll'
- x0: un vettore di innesco
- Nmax: un numero massimo di iterazioni consentite al programma
- norma: la norma rispetto alla quale si considera il criterio d'arresto del residuo.

e tornasse in output:

- xk : primo vettore (della successione di vettori generata) che soddisfa il criterio di arresto del residuo relativamente alla tolleranza 'toll' e norma 'norma', oppure, l'ultimo vettore calcolato dal programma: $x^{(Nmax)}$, qualora nessun vettore generato dovesse soddisfare il criterio di arresto sopra citato
- k : il numero di iterazioni eseguite dal programma
- rk_norm: la norma 'norma' dell'ultimo residuo calcolato

ho innanzitutto fatto decidere all'utente (nella FASE DI SETTAGGIO) la norma 'norma' da utilizzare (per il problema 3 viene considerata la norma infinito: "Inf"), dopodiché ho inteso, per "convergere entro la precisione $\epsilon$", una convergenza con tolleranza $\epsilon$ rispetto al criterio d'arresto del residuo in cui viene utilizzata come norma la nostra 'norma'.

La tabella restituita dallo script, per la risoluzione del punto (c), è la seguente:

| $K_\epsilon$ | $x^k$ | $x$ | $\|\|x^k - x\|\|_\infty$ | $\epsilon$ |
|---|---|---|---|---|
| 5 | 1.00384580498866, 1.9925882734046, 2.98996220710506, | 1, 2, 3 | 0.010037792894936 | 0.1 |
| 7 | 1.00059155598747, 2.00113829114412, 3.00066112370874 | 1, 2, 3 | 0.00113829114412178 | 0.01 |
| 9 | 0.999850215496629, 1.99987549455804, 2.99999668725137 | 1, 2, 3 | 0.000149784503370665 | 0.001 |
| 11 | 1.00002078563287, 2.00000967542883, 2.99999302515454 | 1, 2, 3 | 2.07856328726663e-05 | 0.0001 |
| 13 | 0.999997916786298, 1.99999966138707, 3.00000132192754 | 1, 2, 3 | 2.08321370165354e-06 | 1e-05 |
| 15 | 1.00000014243814, 1.99999995026036, 2.99999983785042 | 1, 2, 3 | 1.62149583093907e-07 | 1e-06 |
| 17 | 0.999999997929445, 2.00000001305538, 3.00000001450418 | 1, 2, 3 | 1.45041774146648e-08 | 1e-07 |
| 19 | 0.999999998732847, 1.99999999817649, 2.99999999921073 | 1, 2, 3 | 1.82350601285464e-09 | 1e-08 |
| 21 | 1.00000000025679, 2.00000000018404, 2.99999999997724 | 1, 2, 3 | 2.56787702213046e-10 | 1e-09 |
| 23 | 0.999999999967475, 1.99999999998728, 3.00000000001331 | 1, 2, 3 | 3.25247606625112e-11 | 1e-10 |

Ricordo al lettore che tutte le tabelle nella presente relazione sono visualizzabili anche eseguendo gli script in MATLAB.

# 5 Appendice

## 5.1 Codice-Problema 1

```
%     author: Matteo Marinelli
%     date: 03/03/2021
%     version: 1
%     note: Script per la risoluzione del Problema test 1;
%           questo codice é composto di due parti:
%               1. una pensata per la risoluzione del problema in generale,
%                  in cui, cioé, si risolve il problema sui dati forniti
%                  nella prima parte del presente script, qualunque siano
%                  i valori di questi dati (i dati, quindi, possono anche
%                  essere diversi da quelli forniti nel testo)
%               2. e una in cui appunto si definiscono i particolari dati
%                  su cui si desidera far risolvere il problema al presente
%                  script durante la sua prossima esecuzione
%                  (FASE DI SETTAGGIO).

%                 1. FASE DI SETTAGGIO


% IMPORTANTE!!!:
% Scrivere la funzione, per successive immissioni, in modo tale che agisca
% puntualmente sui vettori (i.e.: quando hai una funzione "complessa",
% ovvero una funzione scritta come combinazione di funzioni pi semplici
% mediante operatori aritmetici, assicurati di usare le operazioni in modo
% puntuale qualora la loro natura non fosse giá puntuale di per s;
% ad esempio: se hai * usa .*)
f = @(x)sqrt(x);
% qui sopra ho una funzione "semplice", quindi il discorso di
% prima non si applica
```

```
% Nodi d'interpolazione
vx = (0:8).^2/64;

% estremo sinistro dell'intervallo [a,b]
a = 0;

% estremo destro dell'intervallo [a,b]
b = 1;

% numero di punti in [a,b] utilizzati per i grafici
m = 1000;

% Vettore (RIGA) di punti in [a,b] in cui valutare la
% differenza tra polinomio di interpolazione e funzione da
% interpolare.
ts = (0:20)/20;

% passo di graduazione sull'asse delle x
dex = 0.1;

% passo di graduazione sull'asse delle y
dey = 2.5;


%                 2. Parte "Generale"


% Calcolo delle valutazioni di f nei nodi d'interpolazione vx
vy = f(vx);

% a pts verrá assegnato il vettore che contiene le valutazioni nei punti
% in ts del polinomio (p) d'interpolazione di f sui nodi vx
pts = valutazionePolinomioInterpolazione(vx, vy, ts);

differenze = pts - f(ts); % Vettore risultato delle differenze

% Trasformo il risultato (differenze) in un vettore 'colonna'
differenze = transpose(differenze);

differenze %#ok<NOPTS>

%       PLOTTING
x = linspace(a, b, m); % Griglia di punti in [a,b]

% vettore delle valutazioni della funzione f ( sqrt(x) per il problema test
% 1) nei punti 'campione' dell'intervallo [0,1]
y = f(x);

% vettore delle valutazioni del polinomio iterpolante p nei punti 'campione'
% dell'intervallo [0, 1]
pol = valutazionePolinomioInterpolazione(vx, vy, x);

figure;
hold on;
```

```
% setting del plot della funzione y = f(x)
hp = plot(x, y, '-', 'LineWidth', 2);
% decido di dare a questo plot una colorazione verde scuro
set(hp, 'color', [0, 0.6, 0.2] );

% setting del plot del polinomio d'interpolazione
hp = plot(x, pol, '--', 'LineWidth', 3);
set(hp, 'color', 'Blue');

% setting del plot dei punti d'interpolazione
hp = plot(vx, vy, 'r.');
set(hp, 'MarkerSize', 25);

%   ABBELLIMENTO AUTOMATICO DELLA FIGURA PRODOTTA
title('Problema 1', 'color', 'r');
box on;
set(gca, 'XAxisLocation', 'origin', 'YAxisLocation', 'origin');
set(gca,'TickLabelInterpreter','latex');

% Settaggio del font
set(gca, 'fontsize', 18);

% Settaggio della legenda
legend({'$f(x)$', 'polinomio d''interpolazione di $f(x)$', 'punti d''interpolazione'});

legend('Interpreter', 'latex');
legend('Location', 'northwest');

% Graduazione e stile degli assi

% OSS: Se voglio un passo delta (ad esempio = 0.1) negli assi graduati, ho
% che il passo rappresenta l'ampiezza di ciascun sottointervallo delimitato
% tra i punti campione che su quell'asse prendo per formare la 'scala'.
% Ora, l'ampiezza di ciasun sottointervallo (con i sottointervalli tutti di
% ampiezza uniforme) é proprio (b-a)/n, con n il numero di sottointervalli
% formati dall'asse graduato. Quindi passo = (b-a)/n --> n = (b-a)/passo, e
% #punti = n + 1 = (b-a)/passo  + 1.
ticks_x = linspace(a, b, ( ((b-a)/dex) + 1) );
xticks(ticks_x);
inizio  = f(a);
massimo = ceil(max(max(pol, y)));
ticks_y = linspace(inizio,  massimo, ( ((massimo-inizio)/dey) + 1) );
yticks(ticks_y);
set(gca, 'ticklabelinterpreter', 'latex');


%       Function Utilizzate nello Script

function pts = valutazionePolinomioInterpolazione(vx, vy, vt)


%     author: Matteo Marinelli
%     date: 28/02/2021
```

```
%     version: 2

%     input
%       vx: vettore dei nodi d'interpolazione
%       vy: vettore delle valutazioni di una certa funzione f sui nodi vx
%       vt: vettore dei punti in cui valutare il polinomio p interpolante f
%       sui nodi vx
%
%     Output
%       pts: vettore delle valutazioni di p nei punti vt


%inizializzazioni
n = length(vx);
m = length(vt);
pts = zeros(1, m); %inizializzazione del vettore di output


% Parte1 dell'algoritmo: calcolo delle differenze divise
% (indipendente dai punti t)
dds = differenzEdivise(vx, vy);

% Parte2 dell'algoritmo (dipendente dai punti t)
for j = 1:m
    t = vt(j);
    hs = dds(n);
    for i = n-1:-1:1
        hs = dds(i) + (t - vx(i))*hs;
    end
    pts(j) = hs;
end

end

function dds = differenzEdivise(vx, vy)

%     author: Matteo Marinelli
%     date: 05/03/2021
%     version: 3

%     Input
%       vx: vettore (RIGA) dei nodi d'interpolazione
%       vy: vettore (RIGA) delle valutazioni di una certa funzione f
%           sui nodi vx
%
%
%      Output
%       dds: vettore (RIGA) dei coefficienti del polinomio che interpola,
%            nella sua forma di Newton, la funzione f sui nodi vx



%               Flusso Eccezionale
```

```
% Le cardinalitá "dell'insieme" dei nodi di interpolazione e delle valutazioni
% della funzione f in quei nodi devono ovviamente coincidere!
if(length(vx) ~= length(vy))
    err('La cardinalitá dell''insieme dei nodi di interpolazione e dell''insieme delle
    valutazioni della funzione f in quei nodi devono coincidere!')




end


%           Flusso Standard

% quanti sono i nodi di interpolazione
n = length(vx);

% matrice n*n inizializzata a '0', corrispondente alla tabella delle
% differenze divise
tabella = zeros(n);


%           Calcolo della tabella delle differenze divise

% inizializzazione prima colonna
tabella(1:n, 1) = transpose(vy);

% costruzione del resto della tabella
for i = 1:n
    for j = 2:i
        % in (i,j) l'ultimo punto é xi e il penultimo é xj-1, quindi se levo
        % l'ultimo elemento vado in riga j-1 e colonna (j-2)+1 dato che in
        % colonna j ho f[x1,...,xj-1, xi] per ogni indice di riga i.
        tabella(i, j) = (tabella(i, j-1) - tabella(j-1, (j-2)+1))/(vx(i) - vx(j-1));

        % la quantitá assegnata sopra é = f[x1, ..., xj-1, xi], con j < i.
    end
end

%           Calcolo del vettore dei coefficienti della forma di Newton
dds = transpose(diag(tabella));

end
```

## 5.2 Codice-Problema 2

```
%     author: Matteo Marinelli
%     date: 05/03/2021
%     version: 1
%     note: Script per la risoluzione del Problema test 2;
%           questo codice é composto di due parti:
%               1. una pensata per la risoluzione del problema in generale,
%                  in cui, cioé, si risolve il problema sui dati forniti
```

```
%                   nella prima parte del presente script, qualunque siano
%                   i valori di questi dati (i dati, quindi, possono anche
%                   essere diversi da quelli forniti nel testo)
%                2. e una in cui appunto si definiscono i particolari dati
%                   su cui si desidera far risolvere il problema al presente
%                   script durante la sua prossima esecuzione
%                   (FASE DI SETTAGGIO).



%               FASE DI SETTAGGIO

% l'estremo sinistro dell'intervallo di integrazione
a = 0;

% l'estremo destro   dell'intervallo di integrazione
b = 1;

% Nomi desiderati per le colonne della tabella del punto (c)
colNames = {'I2', 'I4', 'I8', 'I16', 'I'};

% Nomi desiderati per le colonne della tabella da costruire nel punto (d)
colNames2 = {'I2', 'I4', 'I8', 'I16', 'I', 'p0'};


% vettore delle precisioni fornite dal
% punto (b) del testo.
epss = 10.^(-(1:10));

% vettore degli ordini delle formule
% dei trapezi citate nel punto (c)
ns   = 2.^(1:4);

% Punto in cui calcolare il polinomio d'interpolazione del
% punto (d) del Problema.
pt_ptd = 0;


%               Parte 'Generale'

fprintf("\nRisoluzione del problema 2\n\n");
%--------------------------------------------------------------------------

% Punto (a)
fprintf("Risoluzione punto (a):\n");
fprintf("Il punto (a) serve per la risoluzione del punto (b).\nLa risoluzione del punto (a)
    consiste nella function 'nepsilon' compresa nel presente script\n\n");




%--------------------------------------------------------------------------
```

```
% Punto (b)
fprintf("Risoluzione punto (b):\n");

% Inizializzazioni

% Lunghezza del vettore delle precisioni fornite
% dal problema.
m  = length(epss);



% Costruzione Tabella

% Valore esatto dell'integrale fornito dal problema
I = exp(b) - exp(a);

% La colonna (VETTORE) della tabella richiesta con i valori esatti
% dell'integrale
I = linspace(I, I, m);

% Vettore dei n(eps) per ogni eps in epss vettore di precisioni desiderate.
n_epss = nepsilon(a, b, epss);

% Vettore delle approssimazioni di I mediante le formule dei trapezi di
% ordine n_eps per ogni n_eps in n_epss
I_ns  = formulaTrapezi(@(x)exp(x), a, b, n_epss);

% Vettore degli errori |I-In|
err_I_In = abs(I - I_ns);

% Colonne della tabella
I        = transpose(I);
n_epss   = transpose(n_epss);
I_ns     = transpose(I_ns);
err_I_In = transpose(err_I_In);
epss     = transpose(epss);


table(epss, n_epss, I_ns, I, err_I_In)
%-------------------------------------------------------------------------

% Punto (c)
fprintf("Risoluzione punto (c):\n");

I    = exp(b) - exp(a);
I_ns = formulaTrapezi(@(x)exp(x), a, b, ns);

I_ns(length(I_ns)+1) = I; % Aggiungo un elemento: I

% Creazione e visualizzazione della tabella
array2table(I_ns, 'VariableNames', colNames)
%-------------------------------------------------------------------------
```

```
% Punto (d)
fprintf("Risoluzione punto (d):\n");

% Questi sono i nodi d'interpolazione corrispondenti ai passi di
% discretizzazione al quadrato delle formule dei trapezi di ordine n per
% approssimare l'integrale definito di f in [a,b], per ogni n in ns.
% (per ns si guardi la parte di SETTAGGIO)
num_nodes = length(ns);

% vettore dei nodi d'interpolazione
% citati nel punto d
vx   = ((b-a)./(ns)).^(2);

% Vettore delle approssimazioni mediante
% formula dei trapezi di ordine n > 1. (Ho
% tolto il valore esatto I).
vy   =  I_ns(1:length(I_ns)-1);

p0   =  valutazionePolinomioInterpolazione(vx, vy, pt_ptd);

I_ns(length(I_ns)+1) = p0;

array2table(I_ns, 'VariableNames', colNames2)
%-------------------------------------------------------------------------



%               FUNCTION UTILIZZATE

%-------------------------------------------------------------------------
function nepss = nepsilon(a, b, tolls)

%     author: Matteo Marinelli
%     date: 05/03/2021
%     version: 2
%     note: Codice per la risoluzione del punto (a) del secondo problema di
%           test. Il codice é pensato per un uso generale in cui passo al
%           programma estremi e tolleranze di valore qualunque.

%     Input
%       a:  l'estremo sinistro di un intervallo contenuto in R
%       b:  l'estremo destro di un intervallo   contenuto in R
%   tolls:  vettore delle soglie di precisione desiderate per
%           l'approssimazione, mediante formula dei trapezi di ordine n generico,
%           del valore esatto dell'integrale definito della funzione
%           exp(x) in [a,b].
%
%     Output
%       neps: un vettore di costanti neps in R t.c.
%             per ogni eps in tolls |I - In| <= eps per ogni n >=
%             neps. Dove con I intendiamo il valore esatto dell'integrale
%             definito di exp(x) in [a,b] e con In intendiamo
%             l'approssimazione di questo mediante la formula dei trapezi
%             di ordine n.
```

```
%

% nel caso fornissi gli estremi dell'intervallo nell'ordine sbagliato...
if( a > b )
    t = a;
    a = b;
    b = t;
end

% OSS: |D(D(exp(x)))| = exp(x) <= exp(b) per ogni x in [a,b]


C = sqrt( ( ((b-a)^3)*exp(b) )/12 ); % Parte indipendente da 'tolls'

% N.B: L'operatore puntuale './' fa sí che venga
% restituito un vettore in neps
nepss = ceil(C./sqrt(tolls));

end


function I_ns = formulaTrapezi(f, a, b, ns)

%     author: Matteo Marinelli
%     date: 05/03/2021
%     version: 3

%     Input
%       a: l'estremo sinistro di un intervallo [a,b]
%       b: l'estremo destro   di un intervallo [a,b]
%       ns: vettore di numeri interi positivi, relativi al numero di
%           sottointervalli di equal misura in cui suddividere l'intervallo
%           [a,b]; ovvero relativi all'ordine di una formula dei trapezi.
%       f: una certa funzione integrabile nell'intervallo di estremi a,b
%
%     Output:
%       I_ns: vettore delle approssimazioni dell'integrale definito di f
%             nell'intervallo di estremi a, b, mediante le formule dei
%             trapezi di ordine n, per ogni n in ns.


hs = (b-a)./ns; % Passi di discretizzazione (VETTORE)

nh = length(hs);
I_ns = zeros(1, nh);

c = (f(a) + f(b))/2;

for i = 1:nh
    sums = sum(f( a + (1:ns(i)-1)*hs(i) ));

    % Valore approssimato dalla formula dei trapezi di ordine ns(i)
    % dell'integrale assegnato
    I_ns(i) = hs(i)*(c + sums);
```

```
end


end

function pts = valutazionePolinomioInterpolazione(vx, vy, vt)


%     author: Matteo Marinelli
%     date: 28/02/2021
%     version: 2

%     input
%       vx: vettore dei nodi d'interpolazione
%       vy: vettore delle valutazioni di una certa funzione f sui nodi vx
%       vt: vettore dei punti in cui valutare il polinomio p interpolante f
%       sui nodi vx
%
%     Output
%       pts: vettore delle valutazioni di p nei punti vt


%inizializzazioni
n = length(vx);
m = length(vt);
pts = zeros(1, m); %inizializzazione del vettore di output


% Parte1 dell'algoritmo: calcolo delle differenze divise
% (indipendente dai punti t)
dds = differenzEdivise(vx, vy);

% Parte2 dell'algoritmo (dipendente dai punti t)
for j = 1:m
    t = vt(j);
    hs = dds(n);
    for i = n-1:-1:1
        hs = dds(i) + (t - vx(i))*hs;
    end
    pts(j) = hs;
end

end

function dds = differenzEdivise(vx, vy)

%     author: Matteo Marinelli
%     date: 05/03/2021
%     version: 3

%     Input
%       vx: vettore (RIGA) dei nodi d'interpolazione
%       vy: vettore (RIGA) delle valutazioni di una certa funzione f
%           sui nodi vx
```

```
%
%
%       Output
%        dds: vettore (RIGA) dei coefficienti del polinomio che interpola,
%             nella sua forma di Newton, la funzione f sui nodi vx



%               Flusso Eccezionale

% Le cardinalitá "dell'insieme" dei nodi di interpolazione e delle valutazioni
% della funzione f in quei nodi devono ovviamente coincidere!
if(length(vx) ~= length(vy))
    err('La cardinalitá dell''insieme dei nodi di interpolazione e dell''insieme delle
    valutazioni della funzione f in quei nodi devono coincidere!')




end


%           Flusso Standard

% quanti sono i nodi di interpolazione
n = length(vx);

% matrice n*n inizializzata a '0', corrispondente alla tabella delle
% differenze divise
tabella = zeros(n);


%           Calcolo della tabella delle differenze divise

% inizializzazione prima colonna
tabella(1:n, 1) = transpose(vy);

% costruzione del resto della tabella
for i = 1:n
    for j = 2:i
        % in (i,j) l'ultimo punto é xi e il penultimo é xj-1, quindi se levo
        % l'ultimo elemento vado in riga j-1 e colonna (j-2)+1 dato che in
        % colonna j ho f[x1,...,xj-1, xi] per ogni indice di riga i.
        tabella(i, j) = (tabella(i, j-1) - tabella(j-1, (j-2)+1))/(vx(i) - vx(j-1));

        % la quantitá assegnata sopra é = f[x1, ..., xj-1, xi], con j < i.
    end
end

%           Calcolo del vettore dei coefficienti della forma di Newton
dds = transpose(diag(tabella));

end
```

### 5.3 Codice-Problema 3

```
%     author: Matteo Marinelli
%     date: 05/03/2021
%     version: 1
%     note: Script per la risoluzione del Problema test 3;
%           questo codice é composto di due parti:
%               1. una pensata per la risoluzione del problema in generale,
%                  in cui, cioé, si risolve il problema sui dati forniti
%                  nella prima parte del presente script, qualunque siano
%                  i valori di questi dati (i dati, quindi, possono anche
%                  essere diversi da quelli forniti nel testo)
%               2. e una in cui appunto si definiscono i particolari dati
%                  su cui si desidera far risolvere il problema al presente
%                  script durante la sua prossima esecuzione
%                  (FASE DI SETTAGGIO).


%           FASE DI SETTAGGIO


% Sistema Lineare Ax = b da risolvere
A = [5,1,2; -1,7,1; 0,1,-3];
b = [13; 16; -7];

% Dati per il metodo di Jacobi

% Vettore d'innesco
x0 = [0; 0; 0];

% Le prime n_itera iterazioni che si vogliono far fare al
% metodo nel punto (a).
n_itera = 10;

% Vettore delle precisioni citate nel punto (c) della
% traccia del problema test 3.
epss = 10.^(-(1:10));

% la norma che si vuole utilizzare nel punto (c)
norma = Inf;

% I nomi desiderati per le colonne della tabella da costruire per la
% risoluzione del punto (c) del problema test 3.
ColNamesc = {'Keps_s', 'xe_s', 'xx_s', 'norm_Inf_err_s', 'eps_s__AKA_precisioni'};

%-----------------------------------------------------------------


%           PARTE 'GENERALE'

% Punti (a) & (b)
fprintf("\nRisoluzione dei punti (a) & (b):\n")
```

```
x = A\b; % soluzione esatta del sistema

n = length(A);

% inizializzazione della matrice che conterrá le prime n_itera iterazioni
% del metodo di Jacobi
xs = zeros(n, n_itera);


% Calcolo delle prime n = n_itera iterazioni del metodo di Jacobi
% ogni iterazione andrá ad occupare una colonna nella matrice xs
for i = 1:n_itera
    [xs(:, i), ~, ~]  = jacobi(A, b, 0, x0, 2, i);
    % Se metto toll = 0, allora avr che o esco prima, ma in questo caso
    % soluzione esatta del sistema e quella calcolata al passo j < i
    % coincideranno, dunque anche se vado avanti fino all'iterazione i la
    % soluzione calcolata resterá la stessa ( perché se il sistema converge
    % allora é per forza consistente, e dalla consistenza ricavo l'osservazione
    % che le soluzioni calcolate resteranno invariate nelle successive
    % iterazioni del metodo), oppure esco esattamente all'iterazione i-esima
    % perché i é il numero massimo di iterazioni consentite al metodo di
    % Jacobi e non convergo esattamente (cioé con tolleranza nulla)
    % prima di i iterazioni
    % (la soluzione potrebbe convergere esattamente anche all'iterazione i,
    %  in ogni caso oltre i iterazioni non posso andare).
    % Si nota, intoltre, che usare la norma 2, piuttosto che la norma Inf o
    % la norma 1, é indifferente quando invoco la function che implementa
    % il metodo di Jacobi su una tolleranza di valore nullo.
    % Questo perch, dalla proprietá di positivitá delle norme, si ha che
    % la norma di un vettore é sempre >= 0, e inoltre é 0 sse il vettore
    % stesso é 0, indipendentemente dalla particolare norma utilizzata.
end

tabella = [x0, xs, x];

tabella %#ok<NOPTS>
%-------------------------------------------------------------------------

% Punto (c)
fprintf("Risoluzione punto (c):\n")

% inizializzazioni 0
[~, n] = size(A);    % Quante componenti avranno i vettori soluzione del sistema
rows = length(epss); % Quante righe ci saranno nella tabella

% inizializzazioni 1
Kepss = zeros(rows, 1); % Vettore colonna dei Keps per ogni eps in epss
xes   = zeros(rows, n); % matrice in cui ogni riga corrisponde ad una soluzione
                        % approssimata xeps calcolata dal metodo di Jacobi,
                        % con eps in epss la precisione relativa a xeps
xx    = zeros(rows, n); % ogni riga corrisponde alla soluzione esatta

norm_err = zeros(rows, 1); % Ogni riga corrisponde alla norma 'norma'
                           % (si guardi la fase di SETTAGGIO)
```

```
                                % della differenza tra la soluzione esatta x
                                % e la soluzione approssimata xeps (relativa
                                % alla precisione eps in epss).

x = A\b; % Soluzione esatta del sistema (La ripeto per leggibilita')

%       COSTRUZIONE TABELLA

for i = 1:rows
% So che mettere 'Inf' ( quarto parametro di jacobi) é un rischio, ma noi
% stiamo lavorando nell'ipotesi che il metodo converga con precisione
% 'epss(i)' alla soluzione desiderata in un numero "ragionevole" di passi;
% altrimenti non potrei soddisfare la richiesta del punto (c), problema 3.
    [xt, k, ~] = jacobi(A, b, epss(i), x0, norma, Inf);

    Kepss(i, 1) = k;

    % Perché xt é un vettore 'colonna' e a me serve un vettore 'riga'
    xes(i, :) = transpose(xt);

    % Stessa ragione di su
    xx(i, :)   = transpose(x);

    norm_err(i, 1) = norm(x-xt, norma);
end

precisioni = transpose(epss);

tabella = table(Kepss, xes, xx, norm_err, precisioni, 'VariableNames', ColNamesc);

tabella %#ok<NOPTS>
%-------------------------------------------------------------------------


%-------------------------------------------------------------------------

%...       FUNCTION UTILIZZATE
%-------------------------------------------------------------------------

function [xk, k, rk_norm] = jacobi(A, b, toll, x0, norma, Nmax)

%     author: Matteo Marinelli
%     date: 28/02/2021
%     version: 2

%     Input
%       A: una matrice quadrata
%       b: un vettore di termini noti relativo al sistema lineare da
%          risolvere Ax = b
%       toll: una soglia di precisione relativa alla condizione d'arresto
%             della function rispetto al criterio d'arresto del residuo con
%             tolleranza, appunto, 'toll'
%       x0:   un vettore di innesco
%       Nmax: un numero massimo di iterazioni consentite al programma
```

```
%      norma: la norma rispetto alla quale si considera il criterio
%             d'arresto del residuo.
%
%     Output
%       xk :  primo vettore (della successione di vettori generata) che
%             soddisfa il criterio di arresto del residuo relativamente
%             alla tolleranza 'toll' e norma 'norma', oppure,
%             l'ultimo vettore calcolato dal programma: (x^Nmax), qualora
%             nessun vettore generato dovesse soddisfare il criterio di
%             arresto sopra citato
%        k :   il numero di iterazioni eseguite dal programma
%  rk_norm:   la norma 'norma' dell'ultimo residuo calcolato

xk = x0;
rk = b - A*xk;
rk_norm = norm(rk, norma);
b_norm = norm(b, norma);
norm_rb = rk_norm/b_norm;
D = diag(diag(A)); % Precondizionatore

k = 0;
while( norm_rb > toll && k < Nmax)
    zk = D\rk;
    xk = xk + zk;
    rk = b - A*xk;
    rk_norm = norm(rk, norma);
    norm_rb = rk_norm/b_norm;
    k = k + 1;
end

end


function cnd = condizionamento(A, norma)

%    Input
%       A: una matrice quadrata
%
%    Output
%       cnd: il condizionamento di A in norma 'norma'

cnd = norm(A, norma)*norm(inv(A), norma);

end
```