Dispense ed esercizi per il Laboratorio di Calcolo Numerico

Elena Gaburro, elenagaburro@gmail.com

AA 2016-2017

Questi appunti non hanno alcuna pretesa di completezza. Sono solo alcune note ed esercizi che affiancano il corso di Laboratorio di Calcolo Numerico. Sono inoltre da considerarsi in perenne "under revision" e pertanto possono contenere discrepanze, inesattezze o errori.

Gli esercizi proposti durante l'esame scritto saranno ispirati a quelli presentati in queste pagine: alcune parti potranno essere simili, altre saranno delle variazioni. Potrà essere richiesto di saper commentare l'algoritmo implementato e i risultati ottenuti. E' permesso portare con sé dispense, appunti e i propri codici durante la prova scritta in laboratorio (è quindi vivamente consigliato lo svolgimento di tutti gli esercizi con commenti dettagliati!).

Indice

Capitolo 1

Introduzione a MATLAB

Il nome MATLAB è acronimo di MATrix LABoratory. Originariamente MATLAB è stato sviluppato come ambiente interattivo per il calcolo matriciale ad alto livello. La principale caratteristica è che non opera con numeri, ma con *matrici*: i vettori e i numeri sono considerati casi particolari di matrici.

Attualmente MATLAB è utilizzato anche come calcolatrice scientifica evoluta, ambiente grafico, e linguaggio di programmazione.

1.1 Linguaggi interpretati e linguaggi compilati

I primi programmi per calcolatori elettronici sono stati scritti direttamente in codice macchina, nella forma di sequenze di bit riconoscibili dal calcolatore come istruzioni, ma molto rapidamente i programmatori hanno riconosciuto l'esigenza di definire linguaggi di programmazione più facili da capire e da utilizzare da parte degli umani. Le istruzioni numeriche sono state quindi sostituite da istruzioni scritte in modo più vicino al linguaggio naturale. Chiaramente, tali linguaggi non sono immediatamente comprensibili al calcolatore che quindi non è in grado di eseguirne direttamente le istruzioni. Per superare questo problema, sono state ideate due soluzioni.

La prima soluzione è quella di trasformare i programmi in codice comprensibile dal calcolatore *prima* di tentarne l'esecuzione. Questo processo di trasformazione si chiama *compilazione* e viene effettuato con l'ausilio di un programma che si chiama *compilatore*. La seconda soluzione invece consiste nell'installare sul calcolatore un programma, chiamato *interprete*, che è in grado di eseguire *direttamente* le istruzioni dei programmi scritti nel linguaggio di programmazione di alto livello attuando, per ogni tipo di costrutto del linguaggio interpretato, le azioni appropriate.

La compilazione e l'interpretazione presentano entrambe vantaggi e svantaggi e, per questo motivo, non è possibile stabilire a priori quale delle 2 soluzioni sia da preferire. Di solito, la compilazione consente di ottenere programmi che vengono eseguiti *più velocemente* rispetto all'interpretazione perché non richiedono l'intermediazione dell'interprete. L'approccio interpretato invece ha come vantaggio fondamentale una maggiore semplicità e velocità nella scrittura dei programmi. E' possibile infatti per il programmatore provare *incrementalmente* il programma *durante* la sua scrittura in modo da verificare immediatamente il comportamento di ciascuna istruzione.

MATLAB è un esempio di linguaggio interpretato. Per questo motivo per esempio in MATLAB non occorre dichiarare le variabili. Esse risultano definite nel punto in cui vengono utilizzate per la prima volta. Inoltre, il loro tipo è *dinamico*, nel senso che esso

può cambiare durante l'esecuzione del programma per effetto di assegnamento di valori che appartengono a tipi diversi.

1.2 Ambienti di MATLAB

L'interfaccia principale di MATLAB è composta da diverse finestre che è possibile affiancare, spostare, ridurre a icona, ridimensionare e così via. Le finestre più usate sono *Command Window*, *Workspace*, *Current Folder*, *Command History* e la finestra dell'*Editor*.

1.2.1 Command Window

In questa finestra è possibile inserire comandi che verranno immediatamente eseguiti non appena si clicca invio. Inoltre è in questa finestra che appaiono gli output dei programmi che lanciamo.

Per usare questa finestra come *prompt* dei comandi (cioè per navigare tra le cartelle, vedere il loro contenuto ecc..) si possono digitare i seguenti comandi e poi cliccare invio:

pwd Per conoscere la cartella attuale in cui ci si trova.

cd Per muoversi tra cartelle: seguito da '.' torna alla cartella al livello precedente; per far completare automaticamente il nome della cartella di destinazione si può schiacciare il tasto TAB.

ls Per vedere il contenuto della cartella.

clc Pulisce la Command Window, cioè non si vedrà più il testo digitato, però tutto ciò che è stato fatto rimane comunque in memoria.

La Command Window si può usare anche come semplice calcolatrice: si possono fare le normali operazioni tra numeri (somma $+$, differenza $-$, moltiplicazione $*$, divisione $/$, elevamento a potenza$\hat{}$e parentesi ()), oppure si possono assegnare valori a delle variabili e usare i nomi delle variabili nei calcoli. Se alla fine di una riga di comando si mette il punto e virgola ';' si sopprime l'output, però il comando viene ugualmente eseguito.

La Command Window è molto comoda perché si vede immediatamente l'effetto di ogni comando, però non è possibile tenere una traccia ordinata del codice che si sta scrivendo. Per questo invece di scrivere direttamente qui, in generale, scriveremo in un file a parte, attraverso l'editor di testo di Matlab. Prima di passare all'editor vediamo rapidamente le altre interfacce di MATLAB.

Per vedere quali variabili sono attualmente in uso si può usare il comando `who` oppure, per avere informazioni più dettagliate si può usare `whos`. Le stesse informazioni si trovano anche nello *Workspace*.

Inoltre è possibile utilizzare i comandi `lookfor`, `help` e `doc` per avere informazioni sulle varie funzioni implementate in MATLAB.

Workspace

Finestra nella quale si possono vedere tutte le variabili memorizzate. Con un doppio click sopra una di esse si apre l'array editor (tipo foglio di Excel) dove si può leggere (e anche modificare) il contenuto delle variabili.

Current Folder

E' la prima cartella in cui MATLAB cerca il file da eseguire. Ci viene mostrato tutto il suo contenuto.

Commnd History

Rimane traccia di tutti i comandi digitati nella Command Window, in ordine cronologico.

1.2.2 Editor, creare un nuovo SCRIPT

E' la finestra nella quale scriviamo i nostri codici. Ci permette di realizzare diversi tipi di file. Oggi ci occupiamo degli **script**. Per aprirne uno nuovo: andare su `NUOVO > SCRIPT` (in alto a sinistra, la sequenza esatta dipende dalla versione di MATLAB utilizzata).

Tutto ciò che scriviamo in uno SCRIPT si potrebbe anche scrivere direttamente nella Command Window, ma scrivendolo qui possiamo organizzarlo meglio. E' il tipo più semplice di file MATLAB che possiamo generare perché non contiene nè variabili di ingresso nè di uscita, e per esempio ci serve per automatizzare azioni che MATLAB dovrà eseguire più volte

Per scrivere dei **commenti** basta far precedere ogni riga di commento dal simbolo '%'.

Ogni file può essere salvato in qualsiasi cartella. L'estensione dei file MATLAB è `.m`. E' importante scegliere bene il **nome del file**. In particolare i nomi dei file di MATLAB:

- Devono necessariamente cominciare con un carattere alfabetico (non possono cominciare con un numero).
- Possono contenere lettere e numeri (esempio Esercizio1.m, Es2Lezione1.m); MATLAB è *case-sensitive* cioè distingue tra lettere minuscole e maiuscole.
- Non possono contenere segni di punteggiatura, a parte l'underscore '_' (in particolare non si può usare il normale trattino '-').
- Sarebbe meglio evitare di sovrascrivere funzioni MATLAB già esistenti (per esempio è meglio non chiamare un proprio file `cos.m` perché esso già rappresenta un comando di MATLAB).

Quando **si comincia un nuovo file** è bene scrivere un commento chiaro che descriva il suo scopo. Inoltre è comodo poter pulire lo schermo, cancellare i valori precedentemente assegnati alle variabili, e chiudere le eventuali finestre aperte (per esempio quelle dei grafici): a tale scopo si possono usare rispettivamente i comandi `clc`, `clear all`, e `close all`.

Per **ottenere informazioni sui comandi**: se si conosce già il nome del comando e si vuole avere qualche dettaglio in più su come funziona si può digitare `help` seguito dal nome del comando. In questo modo si ottengono delle informazioni sintetiche (spesso sufficienti); per informazioni ancora più dettagliate si può digitare `doc` seguito dal nome del comando (in questa parte di documentazione sono spesso presenti anche degli esempi).

Se invece non si conosce il nome specifico del comando si può usare `lookfor` seguito da una parola chiave (Es: `lookfor cosine`).

Per quanto riguarda le **variabili** $MATLAB$ non richiede alcun tipo di dichiarazione o dimensionamento. A differenza dei normali linguaggi di programmazione (C, Pascal, Fortran) non occorre dichiarare le variabili. L'assegnazione coincide con la dichiarazione.

Esercizio .1 (Primi comandi MATLAB, svolto a lezione)**.** Creare un nuovo `script` MATLAB. Scrivere al suo interno qualche commento riguardante i nomi che si possono dare ai file. Salvare questo file con il nome di `Esercizio1_PrimiComandi.m`. Provare i comandi nominati in classe (`pi, exp, 1i, format, lookfor, help, doc`) le funzioni goniometriche ed esponenziali.

1.3 Vettori e matrici

Per creare un **vettore** esistono diverse possibilità:

- Mediante inserimento diretto di ogni sua componente:

```
a = [3, 12, 25, 4];   % se gli elementi del vettore sono separati da ...
b = [3 12 25 4];      % ... virgole o spazi vuoti -> vettore RIGA
c = [3; 12; 25; 4];   % separando con punti e virgola creo vettori COLONNA
c';   % il comando ' traspone il vettore (da riga a colonna o viceversa)
```

- Come l'elenco ordinato di N nodi equispaziati nell'intervallo $[a, b]$ (estremi inclusi):

```
a = 2; b = 7; N = 250;
x = linspace(a,b,N);
```

- Come l'elenco dei punti che si trovano partendo dall'estremo a di un intervallo, muovendosi con passo p fino a raggiungere l'altro estremo (b) dell'intervallo (il passo può anche essere negativo):

```
a = 2; b = 6; p = 0.5;
x = a:p:b;   % genera [2  2.5  3  3.5  4  4.5  5  5.5  6]
a = 100; b = 95; p = -1;
y = a:p:b;   % genera [100  99  98  97  96  95]
```

- Casi particolari

```
u = ones(4,1)   % vettore con 4 righe e 1 colonna contenente tutti 1
z = zeros(1,7)  % vettore con 1 riga e 7 colonne contenente tutti 0
t = 3*ones(4,1) % vettore con 4 righe e 1 colonna contenente tutti 3
```

Dato un vettore x è poi possibile

- Determinarne la lunghezza `length(x)`;
- Accedere alla sua componente i-esima `x(i)`;
- Modificare la componente i-esima, es. `x(i) = 50`;
- Accedere e/o modificare le componenti dalla i alla h, es:

```
x = 1:1:6;            % genera [1 2 3 4 5 6]
x(3:5) = [55,66,77]; % x diventa [1 2 55 66 77 6]
```

- Accedere all'ultima componente del vettore (senza doverne conoscere la lunghezza) `x(end)`;
- Fare varie operazioni, come *somma*, *sottrazione*, *moltiplicazione divisione* componente per componente (con i comandi $+$, $-$, $.*$ e $./$), o anche il prodotto scalare (cioè dati due vettori riga `a` e `b`, si fa $\texttt{a} * \texttt{b}'$). Le operazioni puntuali, componente per componente, sono molto importanti perché ci permettono di evitare cicli in molte occasioni.

Per creare delle **matrici** si procede o in modo analogo ai vettori (per inserimento diretto, o mediante `ones, zeros`) oppure si possono creare matrici particolari con comandi come `eye, diag`. Su di esse si possono applicare per esempio i seguenti comandi

- `length, size`: restituiscono rispettivamente la dimensione maggiore e entrambe le dimensioni di una matrice.
- `det, inv, eig` per ottenere il determinante, l'inversa, gli autovalori e gli autovettori di una matrice (guardare l'help di MATLAB per maggiori dettagli)

Esercizio .2 (Matrici e Vettori, svolto a lezione)**.** Definire vari vettori e matrici attraverso le modalità viste sopra. Provare ad eseguire alcune operazioni.

Le matrici poi si possono manipolare in vario modo, vediamo qualche esempio attraverso il prossimo esercizio guidato.

Esercizio .3 (Matrici, da svolgere autonomamente 20 min)**.**

a. Creare i vettori $v = [0, \pi, 2\pi]$ e $w = [0, 1, 2]$. Calcolarne la somma, il prodotto scalare. Calcolarne il prodotto e il quoziente componente per componente. Moltiplicare w per 10. Calcolare il seno e il coseno di v.

b. Creare il vettore riga $z1$ contenete tutti 1, di dimensione 1000. Creare il vettore colonna $z2$ contenente tutti i numeri pari interi tra 1 e 2000. Verificarne le dimensioni con gli appositi comandi e eseguire il prodotto scalare.

c. Costruire la matrice $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 3 & 3 & 3 \\ 6 & 5 & 4 & 3 \end{bmatrix}$, in 2 modi diversi e verificare che le matrici fatte siano uguali. Usare il comando `size` per trovare la dimensione di A.

```
% Mediante inserimento diretto
A <- [1,2,3,4; 3, ... ];
% A partire da 3 vettori
a <- 1:1:4;   b <- ...;
c <- ...
A2 <- [a;b;c]
% Vediamo se sono uguali (== fa il test di uguaglianza)
A == A2  % 1 significa VERO, 0 significa FALSO
% (il confronto viene fatto elemento per elemento)
dimensioniA = size(A)
```

d. Estrarre la prima colonna di A e salvarla; estrarre le sottomatrici (chiamarle rispettivamente B e C) di dimensione 2×2 e 3×3 (che occupano la parte superiore sinistra della matrice A). Calcolare la dimensione degli elementi estratti.

```
% Selezioniamo la 1' colonna di A e la salviamo nel vettore 'k'
k = A(:,1)  % = A(tutte le righe, colonna 1)
dimensioniK = ... ;
% Selezioniamo la sottomatrice di A (2,2) in alto a sinistra (la chiamiamo B)
B = A(1:2,1:2) % = A(righe da 1 a 2, colonne da 1 a 2)
dimensioniB =  ...;
% Selezioniamo la sottomatrice di A (3,3) in alto a sinistra (la chiamiamo C)
C = ...  % = A(righe da 1 a 3, colonne ... )
dim ...
```

e. Porre l'elemento nella seconda riga, terza colonna di A uguale a 55 e, quello nella prima riga, quarta colonna uguale a 33.

```
A(2,3) = 55;
A ... ;
```

f. Estrarre, e salvare in un vettore, la diagonale di C. Creare poi la matrice D che sia una matrice che ha per diagonale il vettore $j = [1, 2, 3, 4]$ e tutti gli altri elementi nulli.

   Suggerimento: guardare l'help del comando `diag`.

g. Calcolare il determinante, l'inversa, gli autovalori e gli autovettori di D.

h. Creare la matrice identità 5×5.

   Suggerimento: comando `eye`.

1.4 Cicli

In *MATLAB* la ripetizione di blocchi di istruzioni per *un numero di volte specificato e in modo incondizionato* viene eseguita tramite l'istruzione di ciclo **FOR ...END** la cui sintassi è

```
for indice = espressione
corpo del ciclo
end
```

Dove `indice` è una quantità che assume diversi valori a seconda di `espressione`, e `end` segna la fine del blocco di istruzioni da ripetere.
In `espressione` possiamo scrivere sequenze di indici arbitrarie, ad esempio
`for i = [2, 3, 5, 7, 11, 13, 17, 19]`
oppure indicare intervalli di numeri successivi, ad esempio
`for i = 1:100` (se il passo non è indicato, quello di default è 1),
o anche andare in ordine decrescente, ad esempio `for i=100:-1:1`.

Se si ha la necessità di ripetere una o più istruzioni fintanto che una condizione sarà verificata non sapendo a priori il numero di ripetizioni si usa l'istruzione **WHILE ...END**, la cui sintassi è:

```
while condizione
blocco di istruzioni
end
```

Dove blocco di istruzioni verrà eseguito fintanto che condizione risulta vera.

1.4.1 Il calcolo di π

Per il calcolo di π esistono alcuni importanti algoritmi non tutti convergenti per motivi di instabilità. Di seguito diamo cenno di 2 di questi algoritmi.

Esercizio .4 (Algoritmo di Archimede, 15 min)**.** Mediante questo algoritmo, π è approssimato con l'area del poligono regolare di 2^n lati inscritto nella circonferenza di raggio 1 (che ha area uguale a π). Indicando con b_i il numero di lati dell'i-esimo poligono regolare iscritto, $s_i = \sin\left(\frac{\pi}{2^i}\right)$ e A_i la corrispondente area, l'algoritmo si può così descrivere:

$$
\begin{aligned}
&b_1 \leftarrow 2; \quad s_1 \leftarrow 1;\\
&\text{for } i \leftarrow 1 \text{ to } n\\
&\quad A_i \leftarrow b_{i-1}\, s_{i-1};\\
&\quad s_i \leftarrow \sqrt{\frac{1-\sqrt{1-s_{i-1}^2}}{2}}\\
&\quad b_i = 2b_{i-1}\\
&\text{end for}
\end{aligned}
$$

Provare prima con n piccolo (es. $n = 10$). Poi provare con $n = 30$: commentare i problemi che appaiono alla luce delle considerazioni fatte nel capitolo successivo. Può aiutare il calcolo di `s2` $= s^2$ e la stampa di questa tabella

```
fprintf('Iter = %2.0f \t s = %2.1e \t s2 = %2.1e \t b = %2.1e \t A = %4.3e \n', ...
          i, s, s2, b, A);
```

Esercizio .5 ($4arctan(1)$, da svolgere autonomamente, 15 min)**.** $\pi = 4\arctan(1)$. Usando l'espansione di Taylor di $\arctan(1)$, π è approssimato con la formula:

$$
arctan(1) = \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7}\right).
$$

Indicando con q_i la somma al passo i, e volendo arrestare l'algoritmo quando la differenza tra due iterate successive è minore di un certo valore`tol`, l'algoritmo si può descrivere come segue

Inizializzare $q_1 \leftarrow 1, i \leftarrow 1$
Inizializzare tol e err
Ciclo WHILE finchè err $>$ tol
    incrementare il contatore i
    $q_i = q_{i-1} + \dfrac{(-1)^{i-1}}{2i-1}$
    calcolare la differenza tra q_i e q_{i-1}
end WHILE
stampare il risultato

Suggerimento: servirsi di una variabile $qOld$ e di un'altra $qNew$ per poterle avere entrambe quando si deve calcolare l'errore, e poi sovrascriverle correttamente.

Esercizio .6 (Evitare i cicli for quando possibile, da svolgere autonomamente, 20 min)**.** In $MATLAB$ è possibile evitare la maggior parte dei cicli `for` utilizzando gli strumenti "furbi" che ci sono forniti così da semplificare e velocizzare i codici.

Creare 2 vettori a scelta `a,b` di 10000 elementi ciascuno. Calcolare

$$s = \sum_{i=1}^{10000} a(i) * b(i)$$

sia usando che non usando il ciclo `for`. Confrontare il tempo impiegato.

Suggerimenti:

- Per non usare il ciclo si può utilizzare il comando `sum` applicato ai 2 vettori moltiplicati tra loro puntualmente;
- Per confrontare i tempi di esecuzione si possono usare i comandi `tic` e `toc`. Il comando `tic` avvia il cronometro, e il comando `toc` lo arresta

```
% definizione dei vettori a e b ...
somma1 = 0; somma2 = 0;  % inizializzazione dei vettori contenenti i risultati
tic
% 1' metodo di calcolo
toc
tic
% 2' metodo di calcolo
toc
```

Esercizio .7 (Grafici, autonomamente)**.** Tracciare il grafico della funzione $\sin x$ per $x \in [-\pi, \pi]$. Colorarlo di verde. Mettere la legenda, le etichette agli assi e dare un titolo al grafico.

Suggerimenti: Definire un vettore x di 100 punti tra $-\pi$ e π. Sia ora $y = sin(x)$. Digitare `plot(x,y,'*g')`. MATLAB disegna un punto di colore verde (green) per ogni coppia di coordinate x e y.

Provare anche `plot(x,y,'-m')`. Per altre combinazioni di simboli e colori vedere il `doc` del comando `plot`.

Per legenda titolo ed etichette usare i seguenti comandi: `legend('sin(x)'), xlabel('x'), ylabel('y'), title('Ecco il grafico di sin(x)')`.

L'esame...

L'esame per la parte di laboratorio consisterà nella soluzione di 3 esercizi attraverso l'implementazione di codice in MATLAB e la motivazione dei risultati ottenuti.

Per ciascun esercizio sarà necessario scrivere un file principale che eventualmente ne invocherà altri. I **file principali** di ciascun esercizio devono chiamarsi rispettivamente `main1.m, main2.m, main3.m`. All'interno di **ciascun file** deve essere riportato il vostro **nome, cognome, numero di matricola**.

Al termine dell'esame dovete inviare una **mail** all'indirizzo **elena.gaburro@unitn.it**, **dal vostro indirizzo istituzionale**, con:
OGGETTO: Esame calcolo numerico, nome, cognome e numero di matricola
ALLEGATO: un **unico file zip** (con nome uguale alla vostra matricola) contenente **tutti** i file destinati alla correzione.
Chi volesse **ritirarsi** deve comunque mandare una mail con nome, cognome, numero di matricola e il testo "MI RITIRO" (se si vuole mantenere un voto ottenuto in un appello precedente ciò deve essere segnalato esplicitamente nella mail).

Chi consegna l'esame **annulla automaticamente** qualsiasi voto ottenuto negli appelli precedenti.

È concesso l'uso di appunti, dispense e dei propri codici.
È vietato l'uso di cellulari ed è vietato comunicare con i compagni in qualsiasi modo.

Le immagini devono essere chiaramente interpretabili.
Il codice deve essere ordinato, indentato e ogni passaggio deve essere adeguatamente motivato.

NOME: .....

COGNOME: .....

NUMERO DI MATRICOLA: .....

È obbligatorio **riconsegnare il testo dell'esame** al termine della prova.

Per prepararsi all'esame

Bibliografia

[1] S. De Marchi, "Appunti di Calcolo Numerico con codici in Matlab/Octave", 2009

[2] M. Caliari, "Dispense del corso di Laboratorio di Calcolo Numerico", 2008

[3] A. Quarteroni, "Introduzione al Calcolo Scientifico".

[4] A. Morzenti, A. Campi, E. Di Nitto, D. Loiacono, P. Spoletini "Introduzione alla programmazione in Malab", 2011 .

[5] M. Venturin, "Introduzione a Matlab", AA 2008 2009.

[6] V. Comincioli, “Analisi numerica: metodi, modelli, applicazioni”, 2005.