Elementi di Informatica e Applicazioni Numeriche T

Script e Funzioni
in Matlab pre-2016b

Limitazioni di Matlab pre-2016b

Matlab pre-2016b non permette di definire funzioni in script

A però volte una funzione è utile solo per un determinato problema

- E.g. regressione di dati, definizione di equazioni non lineari...

In questo caso definire un file di funzione rende le cose più complicate

È possibile aggirare il problema?

Sì! Il modo più semplice consiste in:

- Trattare l'intero script come una funzione
- Definire la "vera" funzione come ausiliaria

Funzioni Ausiliarie

Matlab permette di definire più funzioni in un unico file

Per esempio, nel file `somma_mat.m`:

```
function s = somma_mat(M) % Funzione principale
    s = 0;
    for C = M
        s = s + somma_col(M);
    end
end
function s = somma_col(C) % Funzione _ausiliaria_
    s = 0;
    for v = C
        s = s + v;
    end
end
```

Funzioni Ausiliarie

Matlab permette di definire più funzioni in un unico file

La funzione principale:

- Compare come prima istruzione del file
- Si chiama come il file
- È l'unica a poter essere chiamata dall'esterno

Le funzioni ausiliarie:

- Compaiono dopo quella principale
- Possono avere un nome arbitrario
- Ne potete definire quante ne volete

Funzioni e Script in Matlab pre-2016b

Supponiamo di avere uno script del genere:

```
% Lo script vero e proprio
clear all
W = rand(1, 1000);
res = my_sum(W)

% Definizione di funzione
function s = my_sum(V)
    s = 0;
    for v in V
        s = s + v;
    end
end
```

Funzioni e Script in Matlab pre-2016b

Possiamo convertirlo in una funzione!

```
function my_sum_test() % niente valore di ritorno
    clear all
    W = rand(1, 1000);
    res = my_sum(W)
end

function s = my_sum(V)
    s = 0;
    for v in V
        s = s + v;
    end
end
```

Funzioni e Script in Matlab pre-2016b

`clear all` è inutile (il record di attivazione è inizialmente vuoto)

```
function my_sum_test()
    W = rand(1, 1000);
    res = my_sum(W)
end

function s = my_sum(V)
    s = 0;
    for v in V
        s = s + v;
    end
end
```

Inseriamo tutto questo in un file di funzione **`my_sum_test.m`**

Funzioni e Script in Matlab pre-2016b

Per eseguire lo "script", dobbiamo chiamarlo come funzione

...Del resto, a questo punto è una funzione

```
my_sum_test() % esegue lo script
```

Nota: se vogliamo chiamare una funzione funzione senza parametri:

- Matlab permette di omettere le parentesi "`()`"...
- ...Quindi possiamo chiamare il nostro script/funzione anche con:

```
my_sum_test
```

- Dà l'illusione di aver chiamato uno script
- In realtà, abbiamo chiamato una funzione (senva passarvi alcunché)

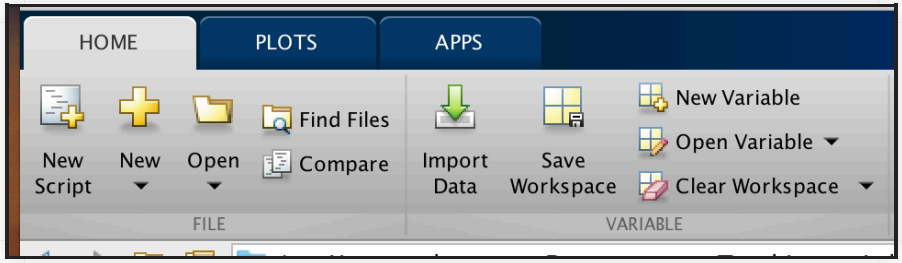
Elementi di Informatica e Applicazioni Numeriche T

Caricamento di Dati da File

Importazione di Fogli di Calcolo

Matlab permette di importare facilmente dati da file

- Basta cliccare su "import data" nella barra degli strumenti:



- Poi va selezionato il file (e.g. excel o file di testo)...
- ...quindi il range di dati che ci interessa...
- ...e va specificato a quali variabili associarlo

Guardate la registrazione della lezione per vedere come si fa!

Funzioni `save` e `load`

Si può anche salvare su file il contenuto di una variabile

Si usa la funzione **`save`**:

```
save() % Anche senza parentesi
save(<nome file>)
save(<nome file>, <nomi di variabile>)
```

Qualche commento sui parametri:

- "**`<nome file>`**" indica il nome del file
- "**`<nome file>`**" è una stringa, ossia una porzione di testo
- Per indicare le stringhe in Matlab si usano gli apici singoli

E.g. nome file: **`'i_miei_dati.mat'`**

Funzioni `save` e `load`

Si può anche salvare su file il contenuto di una variabile

Si usa la funzione **`save`**:

```
save() % Anche senza parentesi
save(<nome file>)
save(<nome file>, <nomi di variabile>)
```

Qualche commento sui parametri:

- "**`<nomi di variabile>`**" è di nuovo una stringa
- Contiene i nomi delle variabili da salvare, separate da "**`,`**"

E.g. **`'M, A, x'`**

Funzioni `save` e `load`

Si può anche salvare su file il contenuto di una variabile

Si usa la funzione **`save`**:

```
save() % Anche senza parentesi
save(<nome file>)
save(<nome file>, <nomi di variabile>)
```

Esempi e comportamento:

- **`save()`** o **`save`** salva tutte le variabili in **`matlab.mat`**
- **`save('dati.mat')`** salva tutte le variabili in **`dati.mat`**
- **`save('dati.mat', 'A, x')`** salva **`A`** e **`x`** in **`dati.mat`**

Funzioni `save` e `load`

Per caricare i dati salvati si usa `load`

```
load()
load(<nome file>)
```

Il comportamento è il seguente:

- **`load`** o **`load()`** carica tutte le variabili da **`matlab.mat`**
- **`load('dati.mat')`** carica tutte le variabili da **`dati.mat`**
- Le variabili vengono caricate nell'ambiente corrente

Ci sono anche altri modi di importare/esportare dati

- Noi però non li vedremo

Elementi di Informatica e Applicazioni Numeriche T

Una Prova: Analisi di Dati di Potenza

Una Prova: Analisi di Dati di Potenza

Scaricate dal sito del corso lo "start-kit" di queste lezione

- Il file **`P.mat`** contenente misurazioni di un contatore elettrico

Predisponete un file (di funzione) **`analyze_power.m`**

- Definite nel file una funzione (ausiliaria):

```
function s = my_sum(V)
```

- La funzione deve sommare gli elementi di **`V`**

Definite poi del codice di test nella funzione principale **`analyze_power`**

- Caricate il file **`P.mat`** (creerà la variabile **`P`**)
- Eseguite **`my_sum`** su **`P`** e visualizzate il risultato

Un Esercizio per Fare Pratica

Una possibile soluzione (nel file `analyze_power.m`):

```
function analyze_power()
    load('P.mat');
    res = my_sum(P)
end

function s = my_sum(V)
    s = 0;
    for w = V
        s = s + w;
    end
end
```

Elementi di Informatica e Applicazioni Numeriche T

Compilatori, Interpreti e Matlab

Compilatori, Interpreti e Matlab

Matlab è un linguaggio interpretato

- Le funzioni predefinite, però...
- Sono realizzate con un diverso linguaggio, che è compilato

Di conseguenza:

- Usare le istruzioni predefinite è molto più efficiente
- Se possibile, vanno preferite rispetto ad utilizzare cicli **`for`** e **`while`**

Proviamo ad effettuare l'analisi dei dati di potenza in due modi:

- Con la funzione codificata nell'esercizio precedente
- Utilizzando le funzioni predefinite (inclusa l'indicizzazione)

Compilatori, Interpreti e Matlab

Usiamo un grossa quantità di dati casuali invece di quelli nel file

```
function analyze_power2()
    P = rand(1, 1e8); % 100,000,000 elementi!
    tic;  % Inizia a misurare il tempo
    my_sum(P)
    toc   % Visualizza il tempo passato
    tic;
    sum(P)
    toc
end
```

- La nostra `my_sum` ci mette $\sim 1.7s$ (sul mio PC)
- Mentra `sum(P)` impiega $\sim 0.044s$ (sul mio PC)

Elementi di Informatica e Applicazioni Numeriche T

Esercizio: Elementi Non Nulli

Esercizio: Elementi Non Nulli

Matlab fornisce la funzione:

```
function I = find(X)
```

- che restituisce gli indici degli elementi diversi da `0`

Nel file di script **`es_find`** , definite la funzione (ausiliaria):

```
function I = my_find(X)
```

- Che replichi tale funzionalità.

Scrivede del codice di test nella funzione principale

- Utilizzate un vettore definito a mano
- Infatti, **`rand`** non genera mai `0`

Elementi di Informatica e Applicazioni Numeriche T

Esercizio: Fattoriale degli Elementi di un Vettore

Esercizio: Fattoriale di un Vettore

Matlab fornisce la funzione:

```
function F = factorial(V)
```

Che restituisce un vettore con il fattoriale di ogni numero in $\mathbf{v}$

- Gli elementi di $\mathbf{v}$ devono essere numeri interi
- Il fattoriale $n!$ di un numero n è definito come:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ \prod_{i=1}^{n} i & \text{se } n > 0 \end{cases}$$

Esercizio: Fattoriale di un Vettore

Si definisca un file di funzione `es_factorial`, contenente:

Una funzione ausiliaria:

```
function F = my_factorial(V)
```

- Che replichi tale il comportamento di **`factorial`**

Si verifichi il funzionamento:

- Scrivendo le istruzioni di test nella funzione principale
- Si esegua **`my_factorial`** e **`factorial`** su un vettore scelto a mano
- Si confrontino i risultati

Elementi di Informatica e Applicazioni Numeriche T

Esercizio: Moltiplicazioni Ripetute

Esercizio: Moltiplicazioni Ripetute

Si definisca un file di funzione `es_dec2bin_f`, contenente:

Una funzione ausiliaria

```
function z = dec2bin_f(x)
```

Che, dato un numero reale x in $]0, 1[$:

- Restituisca la sua rappresentazione binaria (vettore di 0/1)...
- ...Ottenuta mediante il metodo delle moltiplicazioni ripetute
- Si generino esattamente 10 cifre binarie

Si noti che non è necessario rappresentare il separatore "`.`"

Esercizio: Moltiplicazioni Ripetute

La parte intera di un numero positivo si può ottenere con

```
floor(<dato>)
```

- `floor` restituisce l'arrotondamento all'intero più piccolo

Si effettuino dei test nella funzione principale

- In particolare, per il numero 0.390625
- Il risultato deve essere (10 cifre):

$$(\quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0)$$

Elementi di Informatica e Applicazioni Numeriche T

Esercizio: Somma di Matrici

Esercizio: Somma di Matrici

Si definisca un file di funzione `es_msum`, contenente:

Una funzione ausiliaria

```
function C = msum(A, B)
```

Che calcoli a somma di due matrici

- Si eviti di utilizzare l'operatore + applicato a vettori
- In altre parole: sommate gli elementi uno ad uno

Si verifichi il funzionamento:

- Scrivendo il codice di test nella funzione principale
- Si confrontino i risultati con quelli di "+", su matrici casuali

Elementi di Informatica e Applicazioni Numeriche T

Esercizio: Indicizzazione
con Valori Logici

Esercizio: Indicizzazione con Valori Logici

Abbiamo visto che Matlab permette di accedere ad un vettore con:

```
V(I)
```

- `V` è un vettore
- `I` è un vettore di valori logici

Nel file di funzione **`es_index2.m`**, definite la funzione (ausiliaria):

```
function W = my_index2(V, I)
```

Che replichi la stessa funzionalità

- Verificate il funzionamento come al solito

Elementi di Informatica e Applicazioni Numeriche T

Esercizio: Elementi Distinti

Esercizio: Elementi Distinti

Matlab fornisce la funzione:

```
function U = unique(X)
```

- **`X`** che restituisce gli elementi distinti del vettore **`X`**

Nel file di funzione **`es_unique.m`**, definite una la funzione (ausiliaria):

```
my_unique(X, Y)
```

Che replichi la stessa funzionalità

- Matlab ordina anche l'array di uscita: noi non lo faremo
- Verificate il funzionamento con un vettore costruito a mano