Indice delle Sezioni

1. Orientamento per la Tesi MATLAB

Questa guida è stata creata specificamente per supportarti nell'analisi dei tempi di assemblaggio della tua tesi. Ho esaminato i tuoi codici e ho identificato alcune aree in cui puoi migliorare il tuo approccio a MATLAB.

Cosa troverai in questa guida:

- Spiegazioni dettagliate sui concetti MATLAB che sembrano più problematici nei tuoi script
- Esempi visivi di strutture dati e come manipolarle correttamente
- Soluzioni a errori comuni che ho individuato nei tuoi codici
- Tecniche di visualizzazione per presentare i risultati in modo professionale

I tuoi principali punti di miglioramento:

1. Comprensione dell'orientamento dati (righe vs colonne)
2. Gestione appropriata di vettori, matrici e celle
3. Visualizzazione efficace con grafici personalizzati
4. Organizzazione del codice per chiarezza e manutenibilità

2. Lavorare con Vettori e Matrici

Differenza tra Vettori Riga e Colonna

```
VETTORE RIGA:    [1  2  3  4]
                  |  |  |  |
                  v1 v2 v3 v4
```

```
VETTORE COLONNA:  [1]
                  [2]
                  [3]
                  [4]
```

Creazione di vettori:

```
% Vettore RIGA (1×4) - elementi separati da spazi o virgole
vettore_riga = [1, 2, 3, 4]     % oppure
vettore_riga = [1 2 3 4]

% Vettore COLONNA (4×1) - elementi separati da punto e virgola
vettore_colonna = [1; 2; 3; 4]   % oppure
vettore_colonna = [1, 2, 3, 4]'  % il simbolo ' traspone il vettore
```

Dalla Teoria alla Pratica - Esempio dai Tuoi Codici:

Nel tuo primo codice, hai scritto:

```
tempi_task = [65.49 69.95 88.77 103.87 96.53 117.72 66.05 58.66];
```

Questo è un vettore RIGA di 8 elementi, dove ciascuno rappresenta il tempo di una macchina. È importante notare che:

- La posizione 1 contiene il dato della prima macchina (65.49)
- La posizione 2 contiene il dato della seconda macchina (69.95)
- E così via...

IMPORTANTE: Se nei commenti si dice "Righe = modelli, colonne = task", ma poi i dati sono strutturati diversamente, ciò genera confusione. Assicurati che i commenti e l'implementazione siano coerenti.

Matrici - Pensale come Tabelle

```
MATRICE (3×4):  [1  2  3  4]  ← Riga 1
                [5  6  7  8]  ← Riga 2
                [9 10 11 12]  ← Riga 3
                 ↑  ↑  ↑  ↑
                 C1 C2 C3 C4
```

Creazione di matrici:

```
% Matrice 3×4 (3 righe, 4 colonne)
A = [1, 2, 3, 4;   % Prima riga
     5, 6, 7, 8;   % Seconda riga
     9, 10, 11, 12] % Terza riga
```

Accesso agli elementi:

```
A(2,3)  % Elemento in riga 2, colonna 3 (= 7)
A(1,:)  % Tutta la riga 1 (= [1 2 3 4])
A(:,4)  % Tutta la colonna 4 (= [4; 8; 12])
```

3. Comprendere l'Orientamento dei Dati

Uno degli errori che ho notato nei tuoi codici è la confusione tra come descrivi i dati nei commenti e come li implementi. Vediamo alcuni esempi concreti:

Esempio del Tuo Codice:

```
% Righe = modelli (1--8), colonne = task
tempi_task = [
    65.49 69.95 88.77 103.87 96.53 117.72 66.05 58.66; % somma dei tempi di
montaggio
];
```

Problema: Il commento dice che le righe sono modelli e le colonne sono task, ma hai creato un vettore riga con 8 elementi. Questo crea confusione.

Soluzione corretta: Se i modelli devono essere nelle righe:

```
% Ogni RIGA rappresenta un modello,
% ogni COLONNA rappresenta un tipo di task
tempi_task = [
    65.49, 69.95, 88.77;  % Modello 1: task 1, task 2, task 3
    103.87, 96.53, 117.72; % Modello 2: task 1, task 2, task 3
    66.05, 58.66, 0       % Modello 3: task 1, task 2, task 3
];
```

Oppure, se intendevi creare un vettore con un valore per ciascun modello:

```
% Un valore per ciascun modello (1-8)
tempi_task = [65.49, 69.95, 88.77, 103.87, 96.53, 117.72, 66.05, 58.66];
```

Come Verificare le Dimensioni:

```
size(tempi_task)  % Restituisce [righe, colonne]
length(tempi_task) % Restituisce la dimensione maggiore (utile per vettori)
```

4. Celle e Gestione di Dati Eterogenei

Nel tuo secondo codice, usi correttamente le celle per gestire dati di lunghezza variabile:

```
tempi_assemblaggio = {
    [0.75, 7.68, 2.64, 1.24, 2.74, 2.74, 2.63, 1.38],  % Pre-assemblato 1
    [NaN, NaN, 1.35, NaN, 0.50, 0.50, NaN, NaN],       % Pre-assemblato 2
    % ... altri pre-assemblati
};
```

Celle vs Matrici - Quando Usare Cosa:

Usa CELLE quando:

- Hai elementi di lunghezze diverse
- Devi memorizzare tipi di dati diversi (numeri, testo, ecc.)
- Alcuni elementi possono contenere valori mancanti (come nel tuo caso)

Usa MATRICI quando:

- Tutti gli elementi sono dello stesso tipo
- Hai una struttura regolare (stesso numero di righe e colonne)
- Devi eseguire operazioni matematiche su tutti gli elementi

Accesso agli Elementi delle Celle:

```
% Accede all'intera cella (restituisce un vettore)
tempi_assemblaggio{1}  % Restituisce [0.75, 7.68, 2.64, 1.24, 2.74, 2.74,
2.63, 1.38]

% Accede a un elemento specifico all'interno di una cella
tempi_assemblaggio{1}(3)  % Restituisce 2.64 (terzo elemento del primo pre-
assemblato)
```

ATTENZIONE alla differenza tra:

- `celle{indice}` → Restituisce il CONTENUTO della cella
- `celle(indice)` → Restituisce la CELLA stessa (ancora come cella)

5. Gestione dei Valori Mancanti (NaN)

Nel tuo secondo codice, usi correttamente `NaN` per indicare valori mancanti:

```
[NaN, NaN, 1.35, NaN, 0.50, 0.50, NaN, NaN]  % Pre-assemblato presente solo in alcuni modelli
```

Proprietà importanti di NaN:

- `NaN` significa "Not a Number" (Non un Numero)
- Qualsiasi operazione con `NaN` produce `NaN` (es: `5 + NaN = NaN`)
- Non puoi confrontare `NaN` con operatori normali (es: `NaN == NaN` è sempre falso!)

Come lavorare con NaN:

```
% Rimuovere NaN da un vettore
dati = [1, 2, NaN, 4, NaN, 6];
dati_validi = dati(~isnan(dati))  % Risultato: [1, 2, 4, 6]

% Calcolare statistiche ignorando NaN
media = mean(dati, 'omitnan')     % Equivalente a nanmean(dati)
deviazione = std(dati, 'omitnan') % Equivalente a nanstd(dati)
```

Errore comune - Filtraggio errato:

```
% SBAGLIATO (questo non funziona come aspetti!)
dati_filtrati = dati(dati ~= NaN)  % Non filtra i NaN!

% CORRETTO
dati_filtrati = dati(~isnan(dati))
```

6. Creazione di Grafici Efficaci

I grafici nel tuo codice originale sono funzionali ma basilari. Ecco come migliorarli:

Grafico a Barre con Colori Significativi:

```
% Crea un grafico a barre
figure;  % Crea una nuova figura
b = bar(dati);
title('Titolo del Grafico');
xlabel('Asse X');
ylabel('Asse Y');

% Personalizza i colori in base ai valori
for i = 1:length(dati)
    if dati(i) < soglia_bassa
        b.FaceColor = 'g';  % Verde per valori bassi
    elseif dati(i) > soglia_alta
```

```
        b.FaceColor = 'r';  % Rosso per valori alti
    else
        b.FaceColor = 'y';  % Giallo per valori intermedi
    end
end
```

Barre Raggruppate (come nel tuo primo codice):

```
% Confronto valori "in linea" vs "fuori linea"
dati_confronto = [prod_in_linea; prod_fuori_linea]';  % Nota la
trasposizione!
bar(dati_confronto, 'grouped');
legend('In linea', 'Fuori linea');
```

Attenzione all'orientamento dei dati per bar grouped:

- Per `bar(dati, 'grouped')`, i dati devono essere una matrice dove:
  - Ogni RIGA rappresenta un gruppo (es: una macchina)
  - Ogni COLONNA rappresenta una serie (es: in linea/fuori linea)
- Se i tuoi dati sono orientati diversamente, usa la trasposizione `'`

Aggiungere Linee di Riferimento e Annotazioni:

```
hold on;  % Mantiene il grafico attuale per aggiungere elementi
yline(valore_soglia, '--r', 'Soglia');  % Linea orizzontale tratteggiata
rossa
xline(3, '--b', 'Punto critico');  % Linea verticale tratteggiata blu

% Aggiungere testo ai punti nel grafico
for i = 1:length(dati)
    text(i, dati(i) + offset, sprintf('%.1f', dati(i)), ...
         'HorizontalAlignment', 'center');
end
hold off;  % Termina la modalità di aggiunta
```

7. Calcoli e Analisi Statistica

Coefficiente di Variazione (CV)

Nel tuo secondo codice usi correttamente la formula per il coefficiente di variazione:

```
% Calcolo del coefficiente di variazione (CV)
CV = (deviazione_standard / media) * 100  % In percentuale
```

Questo è un ottimo modo per confrontare la variabilità di serie con diverse unità di misura o diversi valori medi.

Operazioni Vettorizzate

MATLAB eccelle nelle operazioni vettorizzate (operazioni su interi array senza cicli):

```
% Operazioni elemento per elemento
A = [1, 2, 3];
B = [4, 5, 6];

C = A + B        % Addizione: [5, 7, 9]
D = A .* B       % Moltiplicazione: [4, 10, 18]
E = A ./ B       % Divisione: [0.25, 0.40, 0.50]
F = A .^ 2       % Potenza: [1, 4, 9]

% Nel tuo codice, usi correttamente le operazioni vettorizzate:
incidenza = (tempi_premontaggi ./ (tempi_task + tempi_premontaggi)) * 100;
```

Il punto (.) prima dell'operatore è cruciale per le operazioni elemento-per-elemento come moltiplicazione, divisione e potenza.

Errore comune - Dimenticare il punto:

```
% SBAGLIATO (se A e B sono vettori o matrici)
result = A * B   % Questo fa la moltiplicazione matriciale!

% CORRETTO
result = A .* B  % Questo moltiplica elemento per elemento
```

8. Risoluzioni agli Errori Comuni

Errore 1: Dimensioni non Coincidenti

```
Error: Matrix dimensions must agree.
```

Cause comuni:

- Tentare di sommare/sottrarre matrici di dimensioni diverse
- Usare `*` invece di `.*` per moltiplicare elemento per elemento

Soluzione:

- Verifica le dimensioni con `size(A)` e `size(B)`

- Assicurati che le operazioni siano compatibili (es: A + B richiede che A e B abbiano le stesse dimensioni)
- Usa la trasposizione `'` se necessario per allineare le dimensioni

Errore 2: Indice Eccede Dimensioni Matrice

```
Index exceeds matrix dimensions.
```

Cause comuni:

- Tentare di accedere a un elemento che non esiste (es: A(10) quando A ha solo 5 elementi)
- Confondere righe e colonne nell'accesso agli elementi

Soluzione:

- Verifica le dimensioni con `size(A)`
- Usa il giusto indice di accesso: `A(riga, colonna)` per matrici

Errore 3: Confusione tra Contenitore delle Celle e Contenuto

```
Cell contents reference from a non-cell array object.
```

Cause comuni:

- Usare `{}` con un oggetto che non è una cella
- Confondere `()` e `{}`

Soluzione:

- Usa `array(indice)` per array normali
- Usa `celle{indice}` per ottenere il contenuto di una cella

9. Struttura di uno Script Professionale

Per rendere il tuo codice più professionale e facile da mantenere, segui questa struttura:

```
%% TITOLO E DESCRIZIONE DELLO SCRIPT
% Autore: [Tuo Nome]
% Data: [Data]
% Descrizione: Analisi dei tempi di assemblaggio per...

%% INIZIALIZZAZIONE
clc; clear; close all;
```

```
%% PARAMETRI E COSTANTI
% Definisci qui tutte le costanti e i parametri
soglia_bassa = 15;
soglia_alta = 25;

%% DATI DI INPUT
% Inserisci qui i tuoi dati di input
tempi_task = [...];
tempi_premontaggi = [...];

%% CALCOLI PRINCIPALI
% Esegui qui i calcoli principali
incidenza = (tempi_premontaggi ./ (tempi_task + tempi_premontaggi)) * 100;

%% VISUALIZZAZIONE DEI RISULTATI
% Crea qui i tuoi grafici
figure('Name', 'Analisi Incidenza');
bar(incidenza);
title('Incidenza dei pre-assemblati');

%% OUTPUT TESTUALE
% Stampa qui i tuoi risultati
fprintf('Risultati dell''analisi:\n');
for i = 1:length(incidenza)
    fprintf('  Modello %d: %.2f%%\n', i, incidenza(i));
end
```

Vantaggi di questa struttura:

- Il simbolo `%%` crea sezioni navigabili nell'Editor MATLAB
- Puoi eseguire una sezione alla volta usando Ctrl+Enter
- Il codice è più facile da leggere, mantenere e debug

10. Risorse e Comandi Utili

Comandi di Base

```
clc          % Pulisce la Command Window
clear        % Rimuove tutte le variabili
clear nome   % Rimuove solo la variabile 'nome'
close all    % Chiude tutte le figure
help comando % Mostra breve guida per un comando
doc comando  % Apre la documentazione completa
```

Funzioni Utili per la Tua Analisi

Per Calcoli Statistici

```
mean(dati)       % Media
median(dati)     % Mediana
std(dati)        % Deviazione standard
var(dati)        % Varianza
min(dati)        % Valore minimo
max(dati)        % Valore massimo
range(dati)      % Range (max - min)

% Per dati con NaN
nanmean(dati)    % Media ignorando NaN
nanstd(dati)     % Deviazione standard ignorando NaN
```

Per Grafici Avanzati

```
% Grafico a barre con etichette
bar(dati);
set(gca, 'XTickLabel', nomi_macchine);  % Imposta etichette sull'asse X
xtickangle(45);  % Ruota le etichette di 45 gradi (utile se lunghe)

% Aggiungere una griglia
grid on;

% Personalizzare la legenda
legend('Serie 1', 'Serie 2', 'Location', 'northeast');

% Aggiungere testo nei grafici
text(x, y, 'Testo da mostrare');
```

Suggerimenti Specifici per i Tuoi Codici

Per il Primo Codice (Incidenza Tempi)

1. Assicurati di inserire i nomi corretti delle macchine nell'array `nomi_macchine`
2. Controlla che l'ordine dei dati in `tempi_task` e `tempi_premontaggi` corrisponda all'ordine in `nomi_macchine`
3. Puoi modificare le soglie per la classificazione (attualmente 15% e 25%)

Per il Secondo Codice (Variabilità)

1. Se desideri aggiungere nuovi pre-assemblati, aggiungi il nome nell'array `preassemblati` e i tempi corrispondenti in `tempi_assemblaggio`
2. Puoi modificare le soglie per la classificazione della variabilità (attualmente 15% e 20%)
3. Ricorda che i valori NaN indicano pre-assemblati non presenti in certi modelli

Esempio di Applicazione - Calcolo dell'Efficienza

Se volessi estendere l'analisi per calcolare l'efficienza di ogni macchina:

```
% Dati di input
tempi_task = [65.49, 69.95, 88.77, 103.87, 96.53, 117.72, 66.05, 58.66];
nomi_macchine = {'MSR', '3200K', 'Modello3', 'Modello4', 'Modello5',
'Modello6', 'Modello7', 'Modello8'};

% Supponiamo di avere il tempo ideale (standard) per ciascuna macchina
tempi_ideali = [60, 65, 80, 95, 90, 110, 60, 55];

% Calcolo dell'efficienza (in percentuale)
efficienza = (tempi_ideali ./ tempi_task) * 100;

% Visualizzazione dei risultati
figure;
bar(efficienza);
title('Efficienza delle Macchine');
xlabel('Macchina');
ylabel('Efficienza (%)');
xticks(1:length(nomi_macchine));
xticklabels(nomi_macchine);
grid on;

% Aggiungi linea di riferimento al 100%
hold on;
yline(100, '--r', '100% (Ideale)');
hold off;

% Output testuale
for i = 1:length(nomi_macchine)
    fprintf('Macchina %s: Efficienza = %.2f%%\n', nomi_macchine{i},
efficienza(i));
end
```

Buon lavoro con la tua tesi! Questa guida dovrebbe aiutarti a completare i tuoi codici MATLAB e a comprendere meglio i concetti sottostanti.

Risorse Online per Approfondimenti

- Documentazione Ufficiale MATLAB
- MATLAB Answers: community per domande e risposte
- Cody: sfide di programmazione MATLAB per esercitarsi