



UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

SCUOLA DI INGEGNERIA

**CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E
DELLE TECNOLOGIE DELL'INFORMAZIONE**

RELAZIONE PROGETTO MNCC

DOCUMENTAZIONE MATSOL

Docente:

Ch.mo Prof. Raffaele Fresa

Studenti:

Michael Pio Stolfi 68787

Rocco Samuele Tancredi 64366

Ivan Scarano 69156

ANNO ACCADEMICO 2023-2024

Indice

1. <u>Introduzione</u>	01
1.1. <u>Specifica del progetto</u>	
1.2. <u>Contesto del progetto</u>	
1.3. <u>Strumenti utilizzati</u>	
2. <u>Sviluppo</u>	05
2.1. <u>Costruzione del modello COMSOL</u>	
2.2. <u>Implementazione delle funzioni MATLAB</u>	
2.3. <u>Progettazione della Command Line Interface (CLI)</u>	
3. <u>Risultati</u>	20
3.1. <u>Presentazione dei risultati</u>	
4. <u>Documentazione delle funzioni</u>	22
4.1. <u>Riepilogo delle funzioni</u>	
4.2. <u>Funzioni raggruppate per compito</u>	
4.3. <u>Documentazione</u>	

Capitolo 1

Introduzione

L'obiettivo principale del progetto è stato quello di sviluppare un applicativo MATLAB che permettesse l'estrazione, l'elaborazione e il salvataggio di una serie di informazioni rilevanti da un qualsiasi modello COMSOL di interesse.

1.1 Specifica del progetto

La specifica di progetto prevedeva che l'applicativo mettesse a disposizione dell'utente tramite opportune elaborazioni, sia nel workspace base di MATLAB sia, nel caso di elaborazioni lunghe, su un file salvato sul disco, i seguenti dati del modello COMSOL di interesse:

- 1) Le matrici di incidenza
- 2) Le funzioni di forma
- 3) Le matrici Jacobiane
- 4) Le matrici di Trasformazione

Per quanto riguarda il primo punto, l'applicativo avrebbe dovuto estrarre/generare le seguenti matrici di incidenza:

- 1) Nodi-Elementi
- 2) Nodi-Facce
- 3) Nodi-Facce di Frontiera
- 4) Nodi-Lati
- 5) Facce-Elementi
- 6) Lati-Elementi
- 7) Lati-Facce
- 8) Lati-Facce di Frontiera
- 9) Domini-Elementi
- 10) Facce di Frontiera Dominio-Facce di Frontiera Elementi

Per quanto riguarda invece il secondo punto esso avrebbe dovuto mettere a disposizione le funzioni di forma della famiglia di Lagrange, utilizzate da

COMSOL per tutti i possibili tipi di elementi di mesh ovvero: tetraedro, piramide, prisma, esaedro; sia per il primo che per il secondo ordine.

Per quanto riguarda il terzo e quarto punto esso avrebbe dovuto estrarre/generare le matrici Jacobiane e le matrici di Trasformazione che permettono la conversione dell'elemento di mesh locale nell'elemento di mesh globale.

1.2 Contesto del progetto

Lo sviluppo di un'applicazione in MATLAB, integrata con COMSOL tramite LiveLink, risulta essere particolarmente utile in contesti di ricerca e sviluppo di modelli ingegneristici complessi. Questo tipo di integrazione facilita l'accesso a informazioni chiave per la comprensione e l'analisi dei modelli fisici simulati.

Partendo col dire che uno dei vantaggi che tale progetto comporta è quello di automatizzare molte delle operazioni che altrimenti richiederebbero un intervento manuale, risparmiando tempo e riducendo il margine d'errore. COMSOL è uno strumento estremamente potente per la simulazione multi-fisica, ma la sua interfaccia nativa può risultare poco pratica per utenti che necessitano di estrarre specifici elementi del modello o che necessitano di processare informazioni in modo personalizzato e automatizzato. Questo è particolarmente utile quando i risultati di COMSOL devono essere rielaborati ulteriormente o integrati in workflow più complessi. Il LiveLink tra COMSOL e MATLAB fornisce quindi un mezzo per adattare l'ambiente di simulazione alle esigenze particolari dell'utente, senza dover dipendere unicamente dalle funzionalità della sua interfaccia nativa. Sviluppare un'applicazione che permette di automatizzare l'estrazione e l'elaborazione dei dati significa non solo aumentare l'efficienza analitica, ma anche assicurare che ogni passo del processo sia facilmente replicabile.

Un altro vantaggio è la possibilità di trasferire i dati tra ambienti di simulazione. Il fatto che COMSOL sia un ambiente chiuso in cui la maggior parte delle operazioni avviene in maniera trasparente all'interfaccia grafica rende difficile esportare e utilizzare i dati al di fuori del suo contesto. Il progetto in questione permette di superare queste barriere, consentendo di sfruttare i dati estratti tra differenti ambienti di simulazione, calcolo e/o analisi, rendendo il flusso di lavoro più fluido e flessibile.

1.3 Strumenti utilizzati

In questo progetto sono stati impiegati diversi strumenti. Di seguito, ne viene fornita una descrizione dei principali:

1) MATLAB

MATLAB è stato lo strumento principale per lo sviluppo dell'applicativo. Si tratta di un ambiente di programmazione avanzato e di un linguaggio di calcolo scientifico ampiamente utilizzato per l'analisi numerica, la simulazione e lo sviluppo di algoritmi. Nello specifico, MATLAB è stato utilizzato per:

- Implementare le funzioni di estrazione dei dati dai modelli COMSOL;
- Eseguire l'elaborazione e la visualizzazione dei dati estratti;
- Salvare i risultati sia nel workspace base di MATLAB che in file esterni per archiviazione e/o analisi successiva.

2) COMSOL Multiphysics

COMSOL Multiphysics è un software per la simulazione multi-fisica che consente di modellare e risolvere problemi in una vasta gamma di campi ingegneristici, tra cui meccanica strutturale, termica, fluidodinamica e elettromagnetismo. In questo progetto, COMSOL è stato utilizzato come piattaforma di simulazione per generare i modelli fisici da cui sono stati estratti i dati. I modelli COMSOL contengono informazioni di interesse come:

- Matrici di incidenza;
- Funzioni di forma degli elementi di mesh;
- Matrici Jacobiane e di trasformazione;
- Proprietà dei materiali utilizzati nei modelli.

3) COMSOL Server

COMSOL Server è il server che ha permesso l'interazione tra COMSOL e

MATLAB ed è il cuore dell'interfaccia di comunicazione LiveLink. Questo strumento consente di interfacciare direttamente i modelli di COMSOL con l'ambiente MATLAB. Tramite il LiveLink, è stato possibile:

- Accedere alle strutture di dati interne dei modelli COMSOL e portarle in MATLAB;
- Eseguire simulazioni in modalità batch direttamente da MATLAB.

Capitolo 2

Sviluppo

Lo sviluppo del progetto si è articolato principalmente in tre fasi: la costruzione del modello COMSOL, l'implementazione delle funzioni MATLAB e infine la progettazione della Command Line Interface (CLI).

La prima fase, indispensabile per il proseguo del progetto, è consistita nella costruzione di un modello COMSOL che fosse il più generale possibile, in termini di numero di componenti, mesh, fisiche, studi e step contenuti. Questo era fondamentale per permettere poi la costruzione di un applicativo che fosse il più generale possibile e robusto a scenari articolati e/o alla presenza di mesh complesse.

La seconda fase, che è il cuore del progetto, è consistita nello sviluppo di un gran numero di funzioni MATLAB, che eseguono tutte le elaborazioni necessarie alla estrazione/generazione delle informazioni di interesse. Queste essendo state pensate per essere generiche ed efficienti hanno spesso delle intestazioni piuttosto complesse, e richiedono il passaggio di numerosi parametri per funzionare correttamente.

La terza e ultima fase, è consistita nello sviluppo del main che è la sede di tutta la logica della Command Line Interface (CLI). La CLI è l'applicativo a riga di comando che permette all'utente in maniera semplice ed intuitiva di interfacciarsi con le funzioni descritte precedentemente; l'idea è che l'utente risponda a una serie di domande sulla natura del dato da estrarre ed essa provveda a chiamare correttamente la funzione di interesse.

2.1. Costruzione del modello COMSOL

Come detto in precedenza la costruzione di un modello COMSOL era indispensabile poiché l'applicativo avrebbe avuto come input principale proprio tali modelli, e inoltre era importante che fosse il più generale possibile. Il modello costruito a scopo di sviluppo/test è stato intitolato *component_library_RF.mph*. Esso si compone di 4 componenti, alcuni componenti hanno fisiche multiple, altri componenti hanno mesh multiple, altri hanno anche domini multipli.

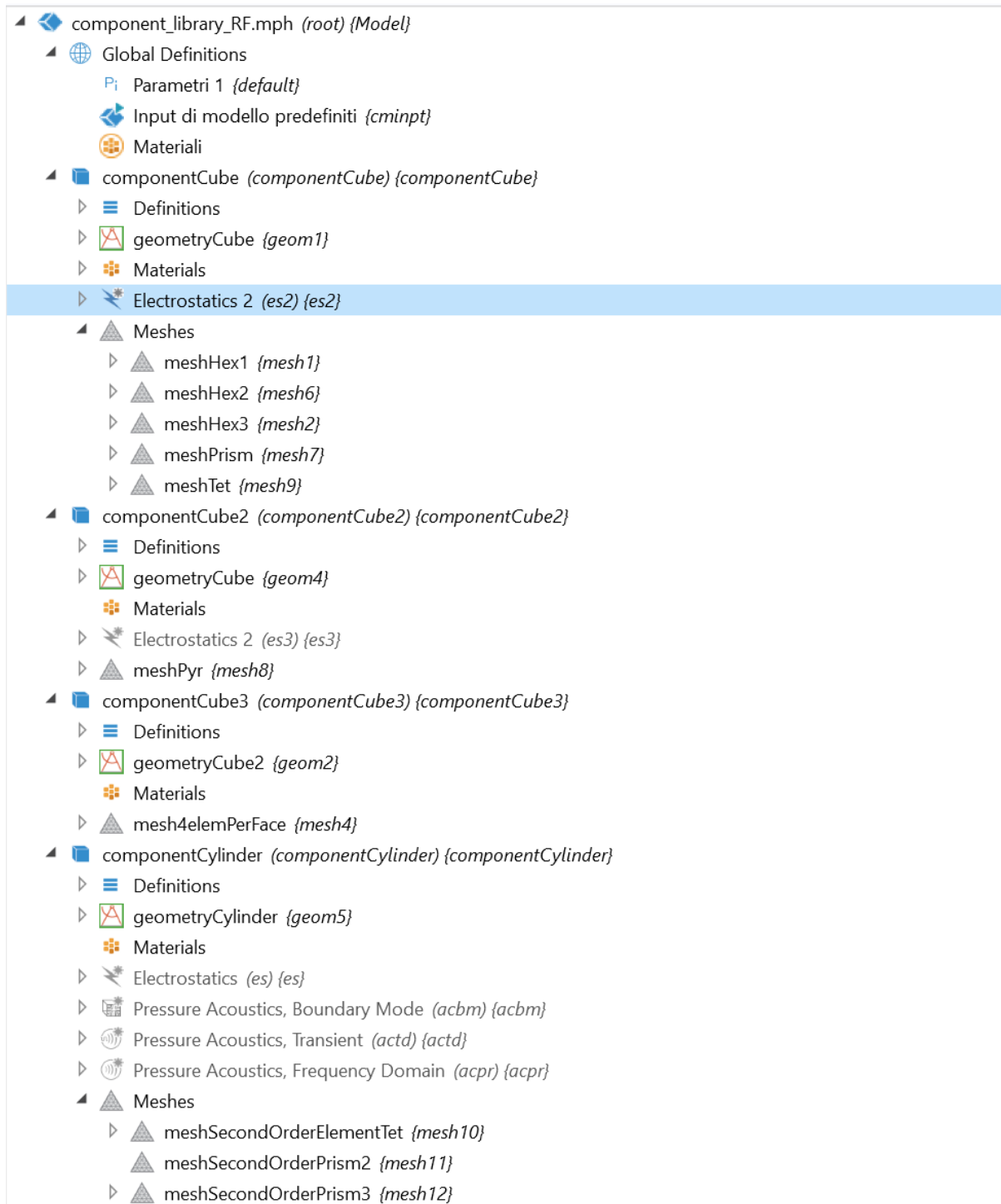


Figura 1: albero del modello con i suoi vari nodi(componenti)

Il componente “componentCube3” è il più banale, esso ha un singolo dominio esaedrico, una singola mesh composta da 8 esaedri del primo ordine, e nessuna fisica; esso è stato fondamentale nelle prime fasi di sviluppo per comprendere come avviene la comunicazione tra MATLAB e COMSOL e inoltre per iniziare l’implementazione delle funzioni di generazione delle matrici di incidenza per elementi di mesh del primo ordine.

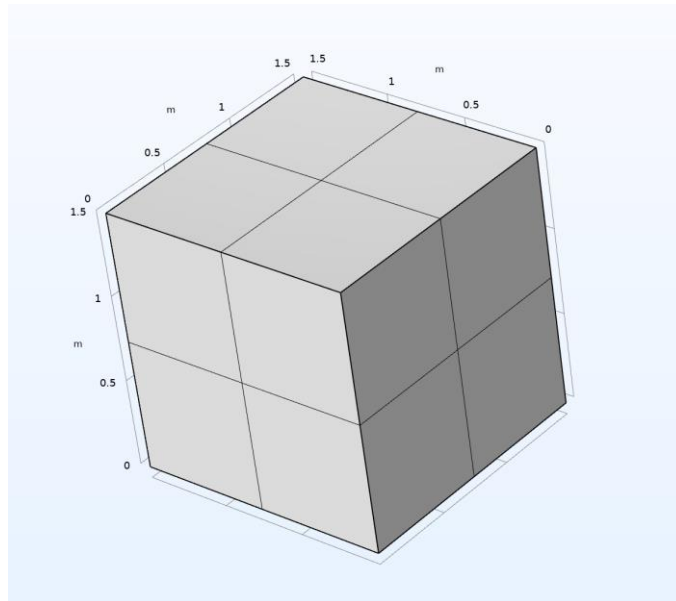


Figura 2: componente *componentCube3*

Il componente “componentCube” è una versione avanzata del precedente, esso ha due domini esaedrici, diverse mesh che coprono 3 dei 4 tipi di elementi di mesh presenti in COMSOL, e una fisica; esso è risultato importante durante la generalizzazione delle funzioni a più domini, e durante l’introduzione della gestione degli elementi del secondo ordine.

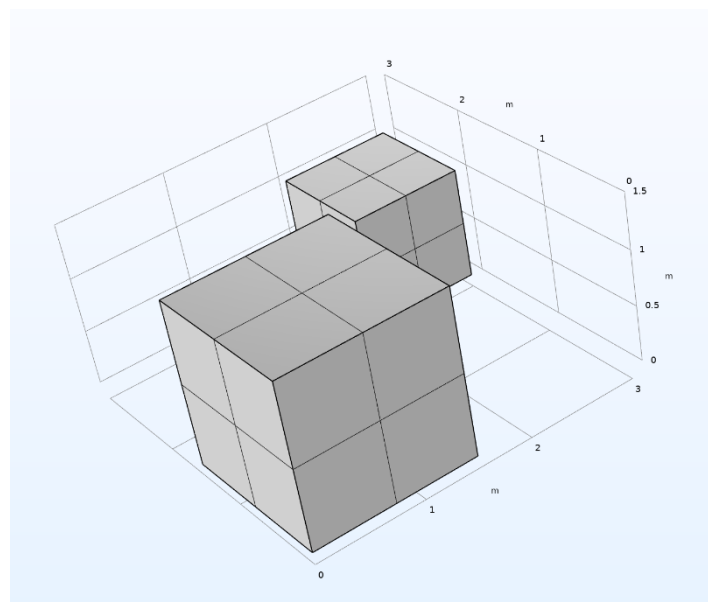


Figura 3: componente *componentCube*

Infine l’ultimo componente di interesse è il componente “componentCube2”, esso ha una geometria ben più complessa degli altri componenti, e ha un dominio che è il risultato dell’operazione di unione di 3 domini differenti; è risultato

indispensabile durante la generalizzazione delle funzioni per la gestione del tipo di elemento di mesh piramidale. L'elemento di mesh piramidale è l'unico elemento di mesh utilizzato da COMSOL che non è possibile inserire volontariamente nella propria mesh, viene invece inserito da COMSOL come elemento di giunzione tra una mesh contenente prismi e una contenente tetraedri. La geometria particolare di questo componente serve appunto a scatenare la creazione di tali elementi di mesh.

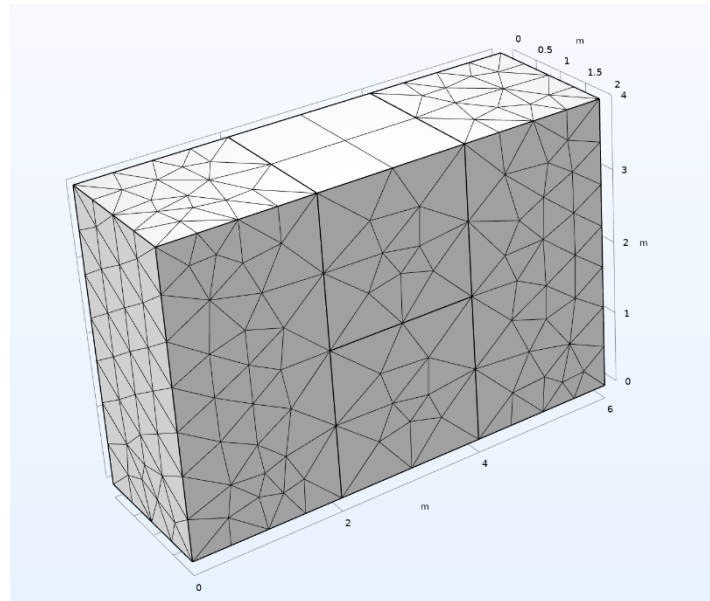


Figura 4: componente componentCube2

Nel modello sono stati inseriti inoltre diversi studi, alcuni con diversi step che vanno a coprire tutti gli studi presenti in COMSOL, questo è servito durante l'implementazione dei controlli sulla mesh e sullo studio selezionato dall'utente tra quelli resi disponibili dalla CLI.

2.1. Implementazione delle funzioni MATLAB

In questo sottoparagrafo verranno analizzate soltanto le principali funzioni MATLAB, seguendo un flusso di esecuzione standard, se si è interessati ai dettagli di una singola funzione fare riferimento al capitolo 4.

Iniziando dal caso in cui l'utente sia interessato a ottenere le matrici di incidenza, interrogando opportunamente la CLI essa scatenerà la chiamata della funzione *createIncidenceMatricesForCLI*. Quest'ultima è un punto di ingresso unico per il calcolo delle matrici di incidenza e si occuperà di chiamare diverse altre

funzioni per ottenere il risultato desiderato, di conseguenza la sua intestazione è piuttosto articolata come è possibile osservare:

```
incidenceMatrices = createIncidenceMatricesForCLI (
    model,
    selectedComponentGeometryTag,
    geometryTagPos,
    meshdata,
    meshdataTypeList,
    searchedString,
    elementsOrder,
    flagsStruct,
    fileName,
    flagFacesEqual)
```

Si rimanda sempre al capitolo 4 per una analisi puntuale dei parametri di ingresso della funzione. Tale funzione si occupa anzitutto di estrarre la matrice delle COORDINATE NODALI tramite il seguente snippet di codice:

```
if elementsOrder == 2
    nodes = double(meshdata.nodes(geometryTagPos).coords);
else
    nodes = meshdata.vertex;
end
% Trasposizione della matrice degli elementi
transposedMatrixNodes = nodes';
```

È interessante notare come la funzione si comporti in maniera differente in relazione all'ordine degli elementi della mesh, questo è dovuto al fatto che la struttura dati *meshdata*, che viene passata alla funzione *createIncidenceMatricesForCLI* viene estrapolata con due chiamate differenti dal server COMSOL a seconda dell'ordine degli elementi della mesh, e quindi può essere composta in maniera differente. Infatti se gli elementi sono del primo ordine viene estrapolata grazie alla seguente istruzione:

```
[~,meshdata] = mphmeshstats(model, selectedMeshTag);
```

mentre se gli elementi sono del secondo ordine viene estrapolata grazie alla seguente istruzione:

```
meshdata = mphxmeshinfo(model);
```

questo è collegato al meccanismo con cui COMSOL gestisce le mesh. In COMSOL tutte le mesh al momento della creazione sono del primo ordine,

quando poi viene aggiunta una fisica, aggiunto uno studio e calcolata una soluzione, in base al tipo di discretizzazione impostato nella sezione della fisica la mesh viene modificata per adattarsi alle necessità della fisica. La funzione *mphmeshstats* restituisce statistiche sulla mesh e informazioni sui dati della mesh prima di qualsiasi elaborazione. Mentre la funzione *mphxmeshinfo* restituisce le informazioni riguardo alla cosiddetta “mesh estesa” ovvero la mesh modificata per adattarsi alla fisica.

Tale funzione si occupa inoltre di estrarre la matrice NODI-ELEMENTI tramite il seguente snippet di codice:

```
meshdataTypePos = strcmp(meshdataTypeList, searchedString);
%N.B.: Come da documentazione gli elementi sono indicizzati da
%      0 quindi bisogna aggiungere 1
if elementsOrder == 2
    try
        if strcmp(searchedString, 'tet')
            elements = double(meshdata.
                               elements(geometryTagPos).
                               tet.nodes+1);
        elseif strcmp(searchedString, 'pyr')
            elements = double(meshdata.
                               elements(geometryTagPos).
                               pyr.nodes+1);
        elseif strcmp(searchedString, 'prism')
            elements = double(meshdata.
                               elements(geometryTagPos).
                               prism.nodes+1);
        elseif strcmp(searchedString, 'hex')
            elements = double(meshdata.
                               elements(geometryTagPos).
                               hex.nodes+1);
        end
    catch
        incidenceMatrices = struct();
        return;
    end
else
    elements = double(meshdata.elem{meshdataTypePos}+1);
end
transposedMatrixElements = elements';
```

Anche in questo caso la funzione si comporta in maniera differente in relazione all'ordine degli elementi della mesh. Inoltre, come denotato nel commento, alle matrici viene sommato elemento per elemento un 1 poiché i nodi che compongono gli elementi in COMSOL sono indicizzati su base 0.

Estrapolate le precedenti due matrici grazie alla API del LiveLink e grazie ad alcune leggere elaborazioni, la funzione si occupa di generare tutte le altre matrici, poiché nessun'altra matrice è estrapolabile (eccezion fatta per la matrice Domini-Elementi). Tutte le altre matrici vengono create sulla base delle informazioni precedentemente recuperate e sulla base delle convenzioni di numerazione degli elementi di mesh adottate da COMSOL. Per fare ciò, la funzione *createIncidenceMatricesForCLI* chiama opportunamente una serie di sei altre funzioni che, come è possibile osservare nell'immagine sottostante:

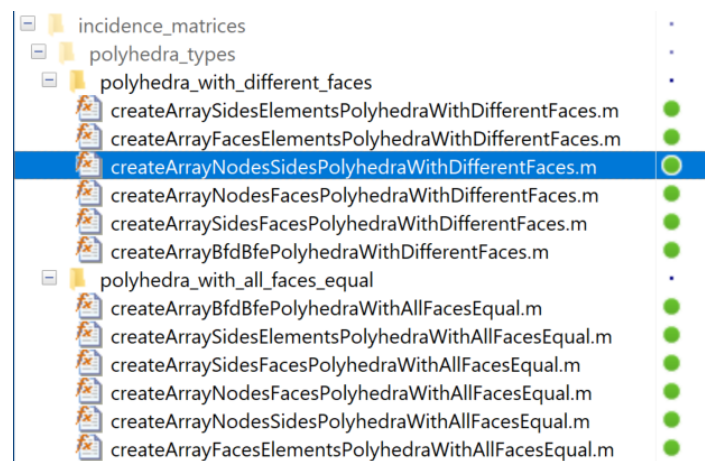


Figura 5: le due famiglie di funzioni di creazione delle matrici

sono divise in due famiglie, ci sono sei funzioni che si occupano della creazione delle altre nove matrici di incidenza nel caso in cui la mesh sia composta di elementi che sono poliedri con tutte le facce uguali, ovvero tetraedri ed esaedri; e ci sono poi le funzioni equivalenti per mesh composte da poliedri aventi facce differenti, come il prismi e la piramidi. Questa distinzione è dovuta alla gestione differente che avviene dei nodi che compongono facce e lati nei due casi.

Analizzando a scopo di esempio il caso in cui l'utente stia cercando di ottenere informazioni sulle matrici di incidenza per un componente avente una mesh composta da esaedri, allora in questa situazione verrà chiamata anzitutto la funzione *createArrayNodesFacesPolyhedraWithAllFacesEqual* avente la seguente intestazione:

```
[arrayNodesFaces, arrayNodesBoundaryFaces]=
createArrayNodesFacesPolyhedraWithAllFacesEqual (
                                                    tableNodesElements,
                                                    elementType,
                                                    elementsOrder)
```

quest'ultima in base al numero di ordine degli elementi della mesh, e in base al tipo di elemento che compone la mesh costruirà la matrici NODI-FACCE seguendo le convenzioni di stile di COMSOL.

```

if elementsOrder == 2
    if strcmp(elementType, 'hex')
        faces = [
            nodes([1, 2, 3, 6, 9, 8, 7, 4, 5]);
            nodes([19, 20, 21, 24, 27, 26, 25, 22, 23]);
            nodes([7, 4, 1, 10, 19, 22, 25, 16, 13]);
            nodes([3, 6, 9, 18, 27, 24, 21, 12, 15]);
            nodes([9, 8, 7, 16, 25, 26, 27, 18, 17]);
            nodes([1, 2, 3, 12, 21, 20, 19, 10, 11]);
        ];
    elseif strcmp(elementType, 'tet')
        faces = [
            nodes([6, 9, 10, 7, 1, 4]);
            nodes([6, 9, 10, 8, 3, 5]);
            nodes([6, 5, 3, 2, 1, 4]);
            nodes([10, 7, 1, 2, 3, 8]);
        ];
    end
else

```

Successivamente nell'ordine verrà chiamata la funzione *createArrayNodesSidesPolyhedraWithAllFacesEqual* avente la seguente intestazione:

```

[arrayNodesSides] =
createArrayNodesSidesPolyhedraWithAllFacesEqual (
                                                    tableNodesFaces,
                                                    elementType,
                                                    elementsOrder)

```

quest'ultima in maniera del tutto simile alla precedente costruirà la matrice NODI-LATI seguendo sempre le convenzioni di stile di COMSOL.

In seguito la funzione *createArrayFacesElementsPolyhedraWithAllFacesEqual* viene chiamata, essa ha la seguente intestazione:

```

arrayFacesElements =
createArrayFacesElementsPolyhedraWithAllFacesEqual (
                                                    tableNodesElements,
                                                    tableNodesFaces,
                                                    elementType)

```

quest'ultima combinando in la matrice NODI-ELEMENTI con la matrice NODI-FACCE costruirà la matrice FACCE-ELEMENTI.

Poi la funzione *createArraySidesElementsPolyhedraWithAllFacesEqual* viene chiamata, essa ha la seguente intestazione:

```
arraySidesElements =  
createArraySidesElementsPolyhedraWithAllFacesEqual(  
                                                    tableNodesElements,  
                                                    tableNodesSides,  
                                                    elementType,  
                                                    elementsOrder)
```

quest'ultima invece combinando in la matrice NODI-ELEMENTI con la matrice NODI-LATI costruirà la matrice LATI-ELEMENTI.

Poi la funzione *createArraySidesFacesPolyhedraWithAllFacesEqual* viene chiamata, essa ha la seguente intestazione:

```
arraySidesFaces =  
createArraySidesFacesPolyhedraWithAllFacesEqual(  
                                                    tableNodesFaces,  
                                                    tableNodesSides,  
                                                    elementType,  
                                                    elementsOrder)
```

quest'ultima invece combinando la matrice NODI-FACCE con la matrice NODI-LATI costruirà la matrice LATI-FACCE.

La matrice DOMINI-ELEMENTI viene gestita direttamente nella funzione *createIncidenceMatricesForCLI*, poiché essa nel caso di elementi del primo ordine è direttamente estrapolabile grazie alla API del LiveLink, mentre nel caso di elementi del secondo ordine il LiveLink attualmente non fornisce alcuna informazione.

```
if elementsOrder == 2  
    cprintf('Keywords', 'Unfortunately, the LiveLink  
                        does not \n');  
    cprintf('Keywords', 'currently provide information  
                        about the \n');  
    cprintf('Keywords', 'domain of membership for  
                        second-order elements, \n');  
    cprintf('Keywords', 'so this matrix cannot be
```

```

                                generated. \n');
else
    arrayDomainsElements = meshdata.
                                elementy{meshdataTypePos};
end

```

Infine per la creazione dell'ultima matrice verrà chiamata la funzione *createArrayBfdBfePolyhedraWithAllFacesEqual*, la quale effettua una serie di calcoli piuttosto complessi e di conseguenza ha la seguente intestazione:

```

arrayBoundaryFacesDomainBoundaryFacesElement =
createArrayBfdBfePolyhedraWithAllFacesEqual(
                                model,
                                tableNodesBoundaryFaces,
                                tableNodalCoordinates,
                                selectedComponentGeometryTag,
                                numberOfBoundary,
                                elementType,
                                elementsOrder)

```

Anche in questo caso si rimanda al capitolo 4 per una analisi puntuale dei parametri di ingresso della funzione. La funzione, una volta eliminati i nodi intermedi in caso di elementi del secondo ordine, per raggiungere l'obiettivo chiama ciclicamente la funzione *checkFaceInDomain* che verifica se una faccia è contenuta in un dominio e questa funzione a sua volta chiama a cascata altre funzioni di utilità. Così facendo si riesce a costruire la matrice *FACCE_FRONTIERA_DOMINIO-FACCE_FRONTIERA_ELEMENTO*.

Infine se richiesto dall'utente le matrici create in precedenza, che volta per volta sono state aggiunte a una struttura dati, vengono salvate sul disco nella struttura dati e tramite la funzione di utilità *saveToJson*, che appunto si occupa di salvarle nella cartella predefinita di progetto ovvero *saved_matrices* in un file formato json che ha il vantaggio di essere leggibile dall'uomo. Se si volesse inoltre caricare in un proprio script MATLAB uno di questi file, frutto dell'elaborazione di MATSOL, viene messa a disposizione la funzione di utilità *loadFromJson* che, come è intuibile dal nome, fa il passo inverso ovvero impostato il nome del file di interesse carica la struttura dati in formato json nel workspace base di MATLAB.

Si continua l'analisi precedente prendendo in considerazione il caso in cui l'utente sia interessato a ottenere le funzioni di forma. La creazione delle funzioni di forma dipende da alcune matrici di incidenza, di conseguenza, interrogando opportunamente la CLI, essa scatenerà la chiamata della funzione

createIncidenceMatricesForCLI in un primo momento e *createShapeFunctions* successivamente. Quest'ultima è un punto di ingresso unico per il calcolo delle funzioni di forma e si presenta come segue:

```
tableShapeFunctions = createShapeFunctions(
    incidenceMatrices,
    elementsType,
    elementsOrder)
```

Tale funzione si occuperà di verificare l'ordine dell'elemento mediante una istruzione di controllo if – else e il tipo di elemento. Sulla base di queste informazioni avverrà la generazione delle opportune shape functions. Considerando come esempio il caso in cui l'utente stia cercando di ottenere le funzioni di forma per un componente avente una mesh composta da esaedri, del primo ordine.

```
% Definizione delle coordinate locali (dell'elemento master)
masterElement = [ 0 0 0; 1 0 0; 0 1 0; 1 1 0;
                  0 0 1; 1 0 1; 0 1 1; 1 1 1];

% Calcolo le funzioni di forma per un elemento esaedrico lineare
syms xi eta zeta
N = @(xi_i, eta_i, zeta_i)(...
    (((1-xi)^(1-xi_i)) * xi^xi_i) * (((1-eta)^(1-eta_i)) *
    eta^eta_i) * (((1-zeta)^(1-zeta_i)) * zeta^zeta_i));
```

Nel codice mostrato sopra viene definito in primis l'elemento master dell'esaedro. In seguito con il comando *syms*, si definiscono le variabili simboliche ξ , η e ζ , che rappresentano le coordinate locali di un punto all'interno dell'elemento. Queste variabili simboliche non rappresentano numeri specifici, ma espressioni o equazioni matematiche che possono essere manipolate algebricamente. *N* è una funzione anonima che calcola l'espressione della funzione di forma per ogni nodo dell'elemento esaedrico. Tale funzione dipende dalle coordinate locali ξ , η e ζ che vengono passate come parametri. Infine i parametri vengono sostituiti con i valori delle coordinate del master element mediante il seguente ciclo for:

```
for i = 1:numNodes
    xi_i = masterElement(i,1);
    eta_i = masterElement(i,2);
    zeta_i = masterElement(i,3);

    shapeFunction{i} = N(xi_i, eta_i, zeta_i);
end
```

La definizione delle funzioni di forma è in alcuni casi differente, come per il tetraedro di second'ordine. Di seguito si riporta il codice.

```
syms xi eta zeta

shapeFunction{1} = (1-xi-eta-zeta)*(1-(2*xi)-(2*eta)-(2*zeta));
shapeFunction{2} = (4*xi)*(1-xi-eta-zeta);
shapeFunction{3} = xi*((2*xi)-1);
shapeFunction{4} = (4*eta)*(1-xi-eta-zeta);
shapeFunction{5} = 4*xi*eta;
shapeFunction{6} = eta*((2*eta)-1);
shapeFunction{7} = (4*zeta)*(1-xi-eta-zeta);
shapeFunction{8} = 4*xi*zeta;
shapeFunction{9} = 4*eta*zeta;
shapeFunction{10} = zeta*((2*zeta)-1);
```

Si introduce ora il caso in cui l'utente sia interessato a ottenere le matrici Jacobiane. Una matrice Jacobiana è una matrice che rappresenta come una regione dello spazio viene "trasformata" (ruotata e scalata) da una funzione. Più precisamente descrive come le coordinate locali ξ , η , ζ vengono trasformate nelle coordinate globali x , y , z (operazione di mappatura). L'applicazione dovrà restituire le matrici Jacobiane per ogni singolo elemento contenuto nel progetto COMSOL preso in esame. Al fine di ottenere le matrici Jacobiane è necessario creare alcune matrici di incidenza e le funzioni di forma; pertanto, interrogando la CLI, dovrà avvenire all'inizio la chiamata della funzione *createIncidenceMatricesForCLI*, successivamente la chiama di *createShapeFunctions* e infine la chiamata di *createJacobianMatrices*. Quest'ultima è il punto di ingresso unico per il calcolo delle Jacobiane. Si procede ad analizzarne il contenuto, partendo dall'intestazione:

```
tableJacobianMatrices = createJacobianMatrices(
                                                    incidenceMatrices,
                                                    tableShapeFunctions)
```

Nella prima parte di codice si estrapolano le matrici, gli elementi e le coordinate degli elementi necessarie, tramite le seguenti istruzioni.

```
% Tabella mesh-nodi
mesh_elements = incidenceMatrices.arrayNodesElements;

% Tabella nodi-coordinate
coord_nodes = incidenceMatrices.arrayNodalCoordinates;

for i = 1: size(coord_nodes,1)
    for j = 1:size(coord_nodes,2)
```

```

        if (0 < coord_nodes(i,j)) && (coord_nodes(i,j) < 1e-15)
            coord_nodes(i,j) = double(0);
        end
    end
end

%% Costruzione della tabella elementi di mesh-coordinate nodali

% Numero di elementi di mesh
numElements = size(mesh_elements, 1);

% Numero di nodi per elemento di mesh
numNodes = size(mesh_elements, 2);

% Preallico un array per salvare le coordinate di una mesh
coord_element = zeros(numNodes,3);

% preallico un cell array per tener traccia delle coordinate di
tutti gli elementi di mesh
coord_mesh = cell(size(mesh_elements,1),1);

%Popolazione degli array
for i = 1:size(mesh_elements,1)
    for j = 1:size(mesh_elements,2)
        node = mesh_elements(i,j);
        coord = coord_nodes(node,:);
        coord_element(j,:) = coord(1,:);
    end
    coord_mesh{i} = coord_element;
    coord_element(:, :) = [];
end
end

```

Avviene in seguito la chiamata alle funzioni di forma, convertendole da table a stringhe e da stringhe in variabili simboliche.

```

stringShapeFunctions = table2array(tableShapeFunctions);
shapeFunctions = sym(zeros(size(stringShapeFunctions)));
for i = 1 : size(shapeFunctions,1)
    shapeFunctions(i) = str2sym(stringShapeFunctions(i));
end

```

In conclusione, si generano le derivate delle funzioni di forma rispetto alle coordinate locali grazie alle quali sarà possibile calcolare la matrice Jacobiana per ogni elemento.

```

dN_dxi = sym(zeros(1,numNodes));
dN_deta = sym(zeros(1,numNodes));
dN_dzeta = sym(zeros(1,numNodes));

syms xi eta zeta
% Calcolo le derivate per costruire la jacobiana

```

```

for i = 1:size(shapeFunctions, 1)
    dN_dxi(1,i) = diff(shapeFunctions(i),xi);
    dN_deta(1,i) = diff(shapeFunctions(i),eta);
    dN_dzeta(1,i) = diff(shapeFunctions(i),zeta);
end

% Definisco le coordinate locali
local_x = 0.5;
local_y = 0.5;
local_z = 0;

% Calcolo le derivate nel punto stabilito
dN_dxi = subs(dN_dxi,{xi,eta,zeta},{local_x,local_y,local_z});
dN_deta = subs(dN_deta,{xi,eta,zeta},{local_x,local_y,local_z});
dN_dzeta = subs(dN_dzeta,{xi,eta,zeta},{local_x,local_y,local_z});

%Calcolo delle derivate globali (matrice Jacobiana)
Jacobian = cell(numElements,1);
Jacobian_info = cell (numElements,1);
tableJacobianMatrices = table();

for i = 1 : numElements
    X = coord_mesh{i}(:,1);
    Y = coord_mesh{i}(:,2);
    Z = coord_mesh{i}(:,3);

    J = zeros(3, 3);
    J(1, 1) = sum(dN_dxi * X) ;
    J(1, 2) = sum(dN_deta * X) ;
    J(1, 3) = sum(dN_dzeta * X);

    J(2, 1) = sum(dN_dxi * Y) ;
    J(2, 2) = sum(dN_deta * Y) ;
    J(2, 3) = sum(dN_dzeta * Y);

    J(3, 1) = sum(dN_dxi * Z);
    J(3, 2) = sum(dN_deta * Z);
    J(3, 3) = sum(dN_dzeta * Z);

    det_J = det(J);
    inv_J = pinv(J);
    det_inv = det(inv_J);

    Jacobian{i} = J;
    Jacobian_info{i} = {array2table(J), det_J,
                        array2table(inv_J), det_inv};
    variableNames = {'Jacobian',
                    'Determinant of the Jacobian',
                    'Inverse of the Jacobian',
                    'Determinant of the Inverse of the Jacobian'};
    tableJacobianMatrices(i,:) = cell2table(
                                                Jacobian_info{i},
                                                'VariableNames',
                                                variableNames);
end

```

Si presenta l'ultimo caso, quello in cui l'utente è interessato a creare le matrici di trasformazione. Una matrice di trasformazione rappresenta una trasformazione geometrica completa (rotazione, scalatura e traslazione) e strutturalmente viene definita esattamente come la matrice Jacobiana con l'aggiunta di una dimensione ulteriore. La colonna che viene aggiunta in ultima posizione comprende i termini di traslazione lungo gli assi e nel codice, questa colonna, viene identificata dall'array *refCoords*; mentre la riga aggiunta è convenzionalmente una riga di tutti 0 con un 1 finale in ultima posizione. Anche in questo caso, per la creazione delle matrici di trasformazione, si dovranno generare alcune matrici di incidenza, le funzioni di forma e le matrici Jacobiane degli elementi. Le funzioni che vengono eseguite in ordine di chiamata sono *createIncidenceMatricesForCLI*, *createShapeFunctions*, *createJacobianMatrices* e infine la funzione di interesse ovvero *createTransformationMatrices*. L'intestazione si presenta come segue:

```
transformationMatrices = createTransformationMatrices(
                                                    incidenceMatrices,
                                                    tableJacobian)
```

Il codice di *createTransformationMatrices* è molto simile a quello per la creazione delle matrici jacobiane, verrà analizzato quindi per differenze, la parte in cui differisce principalmente è la seguente:

```
% Calcolo la matrice di trasformazione
transformation_matrices = cell(size(mesh_elements,1),1);

Jacobian = table2array(tableJacobian(:, 1));

for i = 1 : numElements
    refCoord = [coord_mesh{i}(1,1) coord_mesh{i}(1,2)
                coord_mesh{i}(1,3)];
    T = [table2array(Jacobian{i, 1}), refCoord'; 0 0 0 1];
    T = round(T,10);
    transformation_matrices{i} = T;
end

coord_mesh = coord_mesh';
nodeLabels = strcat('e_', string(1:size(coord_mesh, 2)));
TransformationMatrices = cell2table(
                                transformation_matrices',
                                'VariableNames',
                                nodeLabels,
                                "RowNames",
                                "Trans Matr");
```

2.3. Progettazione della Command Line Interface (CLI)

In questo sottoparagrafo verranno analizzate soltanto le principali feature della CLI e come essa è stata pensata e progettata, seguendo un flusso di esecuzione standard, se si è interessati ai dettagli di una singola funzione è necessario fare riferimento al capitolo 4. L'ultimo passo dello sviluppo dell'applicativo è stato proprio lo sviluppo della Command Line Interface o CLI, ovvero una interfaccia a riga di comando che permettesse di orchestrare tutte le funzioni che si occupano dell'elaborazione e che permettesse all'utente di interfacciarsi in maniera intuitiva con l'applicativo. Per avviare la CLI è sufficiente fare il run del main dell'applicativo, la prima cosa che si presenterà all'utente è lo splash screen come da immagine sottostante:

[illegible]

Figura 6: splash screen CLI

Quest'ultimo fornisce una serie di informazioni generali sull'applicativo, tra cui il numero di versione e una descrizione del suo possibile utilizzo.

MATSOL è pesantemente basato sul COMSOL LiveLink for MATLAB, non può funzionare in assenza di esso, infatti il secondo passo è proprio un tentativo di stabilire una connessione con il server COMSOL come è possibile osservare nell'immagine sottostante:

```
=====
Please wait while the connection to the Comsol Server is checked...
Connection successfully established!
=====
```

Figura 7: tentativo connessione con server COMSOL

Nel caso in cui il tentativo dia esito negativo la CLI avvisa l'utente dell'accaduto, e termina.

Il primo passo che richiede una interazione dell'utente è il terzo, ovvero la selezione del modello. In quest'ultimo verrà aperta una finestra del File Explorer che permetterà all'utente di selezionare in maniera comoda il modello di interesse.

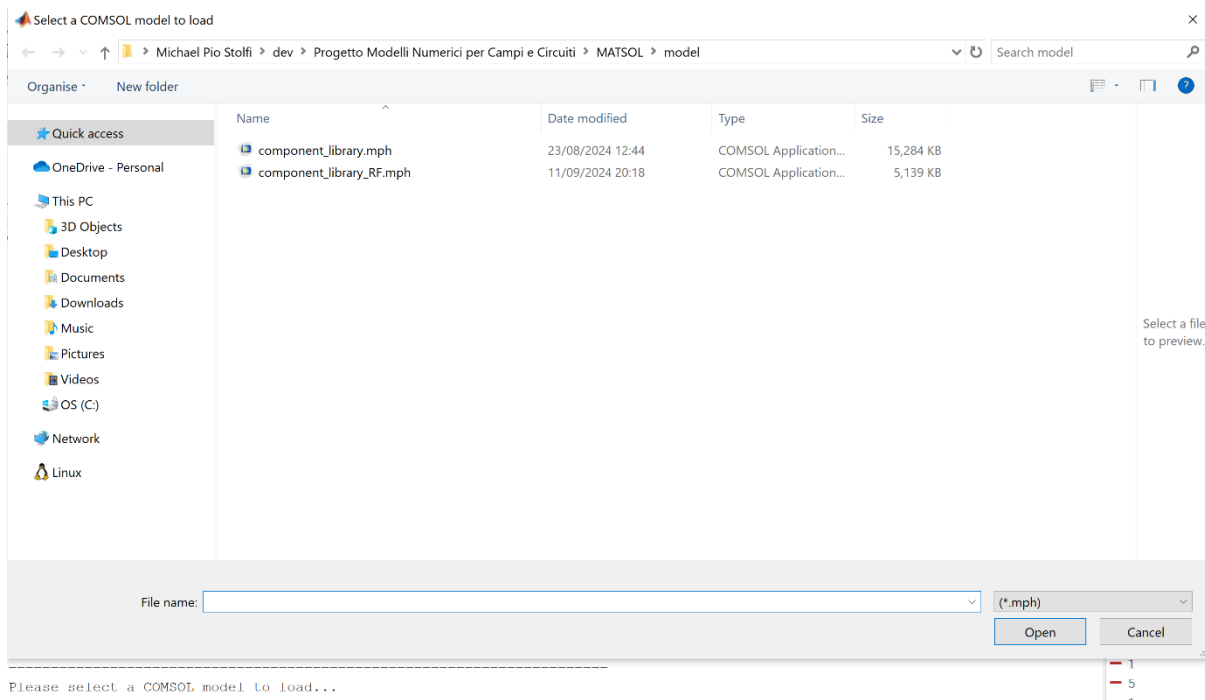


Figura 8: selezione modello

Nel caso in cui l'utente chiuda la finestra senza selezionare alcun modello, oppure selezioni un file con formato diverso dal “.mph”, oppure il modello selezionato sia corrotto allora la CLI avvisa l'utente dell'accaduto, e termina.

Il passo successivo è la selezione del componente, MATSOL scansiona il modello selezionato, preleva tutti i componenti e permette all'utente di selezionarne uno di interesse come illustrato nell'immagine sottostante:

```
=====
Please select the component of interest from those available:

1) componentCube
2) componentCube2
3) componentCube3
4) componentCylinder

Selection-> |
```

Figura 9: selezione del componente

Nel caso in cui il modello non possenga ancora alcun componente, la CLI avvisa l'utente dell'accaduto, e termina.

Il seguente passo è la selezione della mesh, MATSOL scansiona il componente selezionato, preleva tutte le mesh associate ad esso, e permette all'utente di selezionarne una di interesse come illustrato nell'immagine sottostante:

```
=====
Please select the mesh of interest from those available:

1) meshHex1
2) meshHex2
3) meshHex3
4) meshPrism
5) meshTet

Selection-> |
```

Figura 10: selezione della mesh

Nel caso in cui il componente non possenga ancora alcuna mesh, la CLI avvisa l'utente dell'accaduto, e termina.

Il seguente passo è la selezione dello studio, MATSOL scansiona il modello selezionato, preleva tutti gli studi presenti, e permette all'utente di selezionarne uno di interesse come illustrato nell'immagine sottostante:

```
=====
Please select the study of interest from those available:

1) std1
2) std2
3) std3
4) std4
5) std5
6) std6
7) std7
8) std8
9) std9
10) std10
11) std11

Selection->
```

Figura 11: selezione dello studio

Nel caso in cui il componente non possenga ancora alcuno studio, la CLI avvisa l'utente dell'accaduto, e termina.

Il seguente passo è la selezione dello step, MATSOL scansiona lo studio selezionato, preleva tutte gli step presenti, e permette all'utente di selezionarne uno di interesse come illustrato nell'immagine sottostante:

```
=====
Please select the step of interest from those available:

1) eig
2) timod

Selection-> |
```

Figura 12: selezione dello step

Nel caso in cui lo studio non possenga ancora alcuno step, la CLI avvisa l'utente dell'accaduto, e termina.

Successivamente avvengono una serie di controlli sui dati appena inseriti, se ad esempio nello step selezionato per il componente selezionato non si è inserita alcuna mesh, allora la cosa viene notificata all'utente e l'applicazione termina. Se invece nello step seleziona per il componente selezionato si è inserita una mesh differente da quella inserita precedentemente allora la cosa viene notificata all'utente e l'applicazione termina. Poi viene verificato se per il componente selezionato sono presenti le funzioni di forma, in caso della loro assenza viene suggerito all'utente di impostare una fisica per il componente di interesse e l'applicazione termina. Poi viene verificato se è presente una soluzione per il modello selezionato, in caso della sua assenza ancora una volta la cosa viene notificata all'utente e l'applicazione termina. Se invece una soluzione attiva è presente ma è allegata a uno studio differente da quello selezionato la cosa viene notificata all'utente e l'applicazione termina.

In seguito a tutte le verifiche precedenti avviene la valutazione del numero di ordine degli elementi della mesh selezionata, passo cruciale per tutte le valutazioni che avvengono nelle funzioni, come visto nel sottoparagrafo precedente.

```
=====
Please wait while the mesh element order number is evaluated...
Evaluation completed!
=====
```

Figura 13: valutazione del numero d'ordine degli elementi della mesh

Dopo tutta la serie di inserimenti precedenti, in cui si sono delineati gli oggetti di interesse per l'utente, avviene la selezione del tipo di dato che si vuole far

estrarre/generare da MATSOL. Quindi viene mostrato un menù di selezione dei servizi di estrazione/generazione attualmente disponibili in MATSOL come illustrato nell'immagine sottostante:

```
=====
Please select the data you want to extract/generate from the model:

1) Incidence Matrices
2) Shape Functions
3) Jacobian Matrices
4) Transformation Matrices

Selection->
```

Figura 14: selezione del servizio di estrazione/generazione

Una volta selezionato il servizio di interesse, la CLI entrerà nella selezione interna al servizio, comportandosi in maniera differente da servizio a servizio.

Nel caso in cui si sia selezionato il servizio di estrazione/generazione delle matrici di incidenza allora verrà mostrato all'utente un ulteriore menù di selezione in cui potrà inserire a quale matrice di incidenza è interessato tra quelle presenti.

```
=====
Please select which matrix you want to extract/generate:

1) NODES-FACES
2) NODES-BOUNDARY_FACES
3) NODES-SIDES
4) FACES-ELEMENTS
5) SIDES-ELEMENTS
6) SIDES-FACES
7) SIDES-BOUNDARY_FACES
8) DOMAINS-ELEMENTS
9) BOUNDARY_FACES_DOMAINS-BOUNDARY_FACES_ELEMENTS
10) ALL

Selection-> |
```

Figura 15: selezione matrice

Si noti il fatto che sono esenti da selezione la matrice di COORDINATE NODALI e la matrice NODI-ELEMENTI, il motivo è legato al fatto che tali matrici, insieme alla matrice DOMINI-ELEMENTI, sono le uniche totalmente estratte tramite il LiveLink, che subiscono leggere elaborazioni e che sono fondamentali alla generazione di tutte le altre matrici. Conseguentemente tali matrici vengono sempre estratte e saranno presenti sempre nella struttura dati di output, qualsiasi matrice sia selezionata dall'utente.

Si noti inoltre come vi sia la possibilità di generare tutte le matrici, questa scelta è sconsigliata nel caso di mesh con molti elementi poiché può portare a tempi di elaborazione complessivi molto lunghi.

Infine la CLI chiede all'utente un'ultima selezione che riguarda la scelta di salvare o non salvare anche su disco la struttura dati contenente le matrici precedentemente selezionate.

```
=====
Wants the matrix to be saved on disk too:

    1) YES
    2) NO

Selection-> |
```

Figura 16: scelta di salvataggio

Se viene scelto di salvare su disco allora, come detto precedentemente, il risultato del lavoro di MATSOL verrà salvato, oltre che nel workspace base di MATLAB, anche nella cartella predefinita di progetto *saved_matrices* in un file formato json leggibile dall'uomo. Tale scelta è altamente consigliata nel caso di elaborazioni lunghe e complesse poiché permette di effettuare i calcoli una volta, per poi poterli sfruttare successivamente senza alcuna preoccupazione di preservare le variabili nel workspace.

Qualsiasi sia la matrice selezionata, la CLI di MATSOL analizza il tipo di elementi presenti nella mesh selezionata e avvia una o più esecuzioni della funzione *createIncidenceMatricesForCLI*, salvando poi opportunamente una o più strutture dati contenenti le suddette matrici.

Nel caso in cui si sia selezionato il servizio di estrazione/generazione delle shape functions la CLI restituirà direttamente la tabella delle funzioni di forma nel WORKSPACE a seconda del tipo di elemento di mesh e dell'ordine. Come nel caso precedente, scegliendo i servizi di creazione delle Matrici Jacobiane e delle Matrici di trasformazione, la CLI allocherà direttamente i risultati, in tabelle, nel WORKSPACE, senza ulteriori step.

Infine la CLI pulisce il workspace base di MATLAB, preservando solo le variabili di interesse, e termina.

Capitolo 3

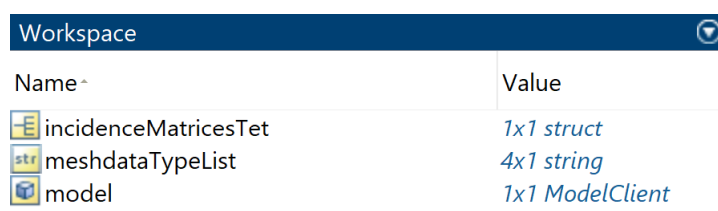
Risultati

In questo capitolo verranno presentati i risultati di una esecuzione di MATSOL. Una esecuzione di MATSOL restituisce due serie di risultati: i risultati fissi, ovvero quei risultati che non cambiano mai e sono presenti a ogni esecuzione, e i risultati variabili, ovvero quei risultati che cambiano in base a quale servizio di estrazione/generazione si è deciso di eseguire.

In base al servizio eseguito vi sarà la possibilità di salvare i risultati dell'elaborazione su disco, questa feature è stata pensata nell'ottica di rendere MATSOL robusto a elaborazioni lunghe e di renderlo facilmente integrabile in altri progetti MATLAB.

3.1 Presentazione dei risultati

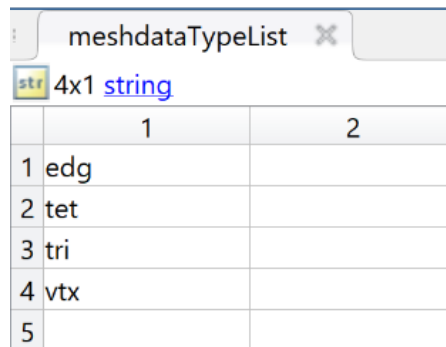
Tra i risultati fissi vi sono quei risultati che non cambiano mai e sono presenti a ogni esecuzione, questo perché possono risultare sempre utili a un utente che vuole interfacciarsi con un modello COMSOL.



Name	Value
incidenceMatricesTet	1x1 struct
meshdataTypeList	4x1 string
model	1x1 ModelClient

Figura 17: esempio di workspace base di MATLAB dopo una esecuzione di MATSOL

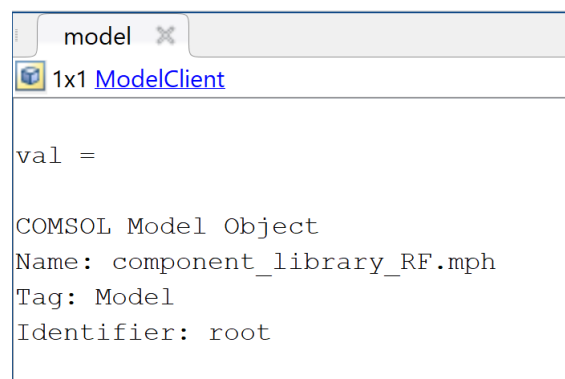
I risultati fissi attualmente presenti in output a una esecuzione di MATSOL vi sono *meshdataTypeList* e *model*. Il primo dei due ovvero *meshdataTypeList* è un array di stringhe contenenti i tag degli elementi che compongono la mesh attualmente in esame e che quindi sono estrapolabili dalla matrice *meshdata*. Vi sono tra essi tag che fanno riferimento a elementi monodimensionali come “vtx”, oppure bidimensionali come “edg” e “tri”, oppure tridimensionali come “tet”.



	1	2
1	edg	
2	tet	
3	tri	
4	vtx	
5		

Figura 18: esempio di stringhe contenute nell'array meshdataTypeList

Il secondo risultato è invece *model* il quale è in sostanza il riferimento a un oggetto di tipo ModelClient molto utile se si vuole ad esempio continuare a navigare l'albero dei nodi che compone il modello in esame, alla ricerca di altri elementi di interesse.



```

model
1x1 ModelClient

val =

COMSOL Model Object
Name: component_library_RF.mph
Tag: Model
Identifier: root

```

Figura 19: esempio di riferimento a un modello selezionato

Tra i risultati variabili invece ci sono quei risultati che cambiano in base a quale servizio di estrazione/generazione si è deciso di eseguire.

Se si è deciso di eseguire il servizio di estrazione/generazione delle matrici di incidenza allora nel workspace di MATLAB sarà presente una o più strutture dati contenenti tutti gli array delle matrici di incidenza richieste per i vari elementi di mesh che compongono la mesh selezionata. Il nome di tali strutture dati può essere uno tra i seguenti: “incidenceMatricesTet”, “incidenceMatricesPyr”, “incidenceMatricesPrism” e “incidenceMatricesHex”. Ovviamente il loro contenuto, come detto, cambia in base a quali matrici di incidenza si è chiesto di generare a MATSOL e si presentano come da immagine sottostante:

incidenceMatricesTet	
1x1 struct with 10 fields	
Field	Value
arrayNodalCoordinates	63x3 double
arrayNodesElements	24x10 double
arrayNodesFaces	60x6 double
arrayNodesBoundaryFaces	24x6 double
arrayNodesSides	98x2 double
arrayFacesElements	24x4 double
arraySidesElements	24x12 double
arraySidesFaces	60x6 double
arraySidesBoundaryFaces	24x6 double
arrayBoundaryFacesDomainBoundaryFacesElement	24x1 double

Figura 20: esempio di struct *incidenceMatricesTet*

Se l'utente sceglie il servizio di creazione delle funzioni di forma il risultato sarà la creazione di una table chiamata *tableShapeFunctions*. Nello specifico caso che segue, la tabella delle funzioni di forma nodali è relativa a una mesh tetraedrica, difatti il suo nome sarà *tableShapeFunctionsTet*.

Workspace	
Name	Value
meshdataTypeList	4x1 string
model	1x1 ModelClient
tableShapeFunctionsTet	10x1 table

Figura 21: workspace base di MATLAB dopo una esecuzione di MATSOL

La struttura della table creata sarà organizzata in modo tale da avere sulle righe i nodi dell'elemento e sull'unica colonna presente le funzioni di forma nodali.

tableShapeFunctionsTet	
10x1 table	
	1 shape_fcn
1 n_1	"(eta + xi + zeta - 1)*(2*eta + 2*xi + 2*zeta - 1)"
2 n_2	"-4*xi*(eta + xi + zeta - 1)"
3 n_3	"xi*(2*xi - 1)"
4 n_4	"-4*eta*(eta + xi + zeta - 1)"
5 n_5	"4*eta*xi"
6 n_6	"eta*(2*eta - 1)"
7 n_7	"-4*zeta*(eta + xi + zeta - 1)"
8 n_8	"4*xi*zeta"
9 n_9	"4*eta*zeta"
10 n_10	"zeta*(2*zeta - 1)"

Figura 212: esempio di struct *incidenceMatricesTet*

Se invece si opta per eseguire il servizio di generazione delle matrici Jacobiane allora nel workspace di MATLAB verrà allocata una tabella chiamata *tableJacobianMatrices*. Questa tabella ha un numero di righe pari al numero di elementi di mesh e 4 colonne. Ciascuna di queste quattro colonne contiene: la matrice Jacobiana dell'elemento in questione, il suo determinante, l'inversa della Jacobiana e il determinante dell'inversa.

tableJacobianMatricesTet				
24x4 table				
	1	2	3	4
	Jacobian	Determinant of the Jacobian	Inverse of the Jacobian	Determinant of the Inverse of the Jacobian
1 e_1	3x3 table	0.8438	3x3 table	1.1852
2 e_2	3x3 table	0.8438	3x3 table	1.1852
3 e_3	3x3 table	0.8438	3x3 table	1.1852
4 e_4	3x3 table	0.8438	3x3 table	1.1852
5 e_5	3x3 table	0.8438	3x3 table	1.1852
6 e_6	3x3 table	0.8438	3x3 table	1.1852
7 e_7	3x3 table	0.8438	3x3 table	1.1852
8 e_8	3x3 table	0.8438	3x3 table	1.1852
9 e_9	3x3 table	0.8438	3x3 table	1.1852
10 e_10	3x3 table	0.8438	3x3 table	1.1852
11 e_11	3x3 table	0.8438	3x3 table	1.1852
12 e_12	3x3 table	0.8438	3x3 table	1.1852
13 e_13	3x3 table	0.8438	3x3 table	1.1852
14 e_14	3x3 table	0.8438	3x3 table	1.1852
15 e_15	3x3 table	0.8438	3x3 table	1.1852
16 e_16	3x3 table	0.8438	3x3 table	1.1852
17 e_17	3x3 table	0.8438	3x3 table	1.1852
18 e_18	3x3 table	0.8438	3x3 table	1.1852
19 e_19	3x3 table	0.8438	3x3 table	1.1852
20 e_20	3x3 table	0.8438	3x3 table	1.1852
21 e_21	3x3 table	0.8438	3x3 table	1.1852
22 e_22	3x3 table	0.8438	3x3 table	1.1852
23 e_23	3x3 table	0.8438	3x3 table	1.1852
24 e_24	3x3 table	0.8438	3x3 table	1.1852

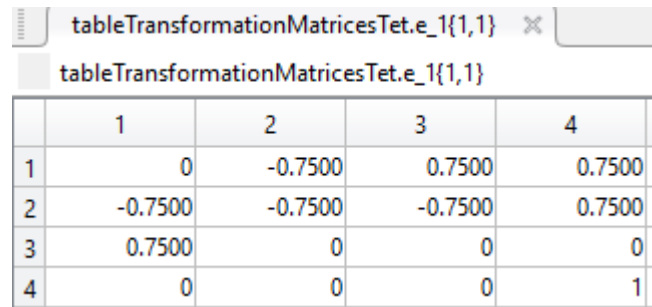
Figura 222: struttura di *tableJacobianMatricesTet*

Ovviamente per accedere alla visualizzazione della matrice Jacobiana e della sua inversa sarà necessario cliccare su una qualsiasi cella della prima e terza colonna. Ad esempio, volendo visualizzare la matrice Jacobiana del primo elemento di questa mesh tetraedrica, il risultato sarà il seguente:

tableJacobianMatricesTet.Jacobian{1,1}				
tableJacobianMatricesTet.Jacobian{1,1}				
	1	2	3	
	J1	J2	J3	
1	0	-0.7500	0.7500	
2	-0.7500	-0.7500	-0.7500	
3	0.7500	0	0	

Figura 23: Matrice Jacobiana dell'elemento 1

In conclusione, selezionando il servizio di generazione della matrice di trasformazione avremmo nel workspace la tabella chiamata *tableTransformationMatrices*. Quest'ultima avrà tante colonne quanti sono gli elementi di mesh e una sola riga corrispondente all'array che contiene la matrice di trasformazione. Anche in questo caso si visualizza la matrice di trasformazione del primo elemento di questa mesh tetraedrica; Il risultato sarà il seguente:



	1	2	3	4
1	0	-0.7500	0.7500	0.7500
2	-0.7500	-0.7500	-0.7500	0.7500
3	0.7500	0	0	0
4	0	0	0	1

Figura 24: Matrice di trasformazione dell'elemento 1

Notare come la matrice jacobiana è compresa nella matrice di trasformazione.

Capitolo 4

Documentazione delle funzioni

In questo capitolo verrà fornita una panoramica documentativa delle varie funzioni che compongono MATSOL. Verranno trattate per scopo, ovvero area di interesse, al momento vi sono sei aree di interesse: cli, incidence_matrices, shape_functions, jacobian_matrices, transformation_matrices e utility.

Nella prima area vi sono tutte le funzioni che permettono il funzionamento della CLI, quindi sono principalmente funzioni di prelevamento e convalida dell'input dall'utente e funzioni di verifica e valutazione di alcuni parametri.

Nella seconda area vi sono tutte le funzioni che permettono tutte le elaborazioni necessarie alla generazione delle matrici di incidenza, che è uno dei servizi offerti da MATSOL. Sono principalmente funzioni di interrogazione del server COMSOL e elaborazione vettorizzata di matrici.

Nella terza area vi è la funzione che permette la generazione delle funzioni di forma. Essa è una funzione di elaborazione, non esegue alcuna estrazione, siccome COMSOL non permette di estrarre alcuna informazione sulle funzioni di forma utilizzate.

Nella quarta e quinta area vi sono le funzioni che si occupano della generazione delle matrici jacobiane e di trasformazione, in ambo i casi si tratta principalmente di funzioni che effettuano elaborazioni sulle matrici estratte con le funzioni della prima area.

Infine nella quinta area vi sono tutte le funzioni di utilità utilizzate in tutte le altre aree. Sono funzioni assortite che fanno calcoli matriciali e vettoriali, stampa su schermo, salvataggio e caricamento da file.

4.1. Riepilogo delle funzioni

[checkConnection](#)

[checkFaceInDomain](#)

[componentPicker](#)

[computePlaneEquationFromPoints](#)

[createArrayBfdBfePolyhedraWithAllFacesEqual](#)

[createArrayBfdBfePolyhedraWithDifferentFaces](#)

[createArrayFacesElementsPolyhedraWithAllFacesEqual](#)

[createArrayFacesElementsPolyhedraWithDifferentFaces](#)

[createArrayNodesFacesPolyhedraWithAllFacesEqual](#)

[createArrayNodesFacesPolyhedraWithDifferentFaces](#)

[createArrayNodesSidesPolyhedraWithAllFacesEqual](#)

[createArrayNodesSidesPolyhedraWithDifferentFaces](#)

[createArraySidesElementsPolyhedraWithAllFacesEqual](#)

[createArraySidesElementsPolyhedraWithDifferentFaces](#)

[createArraySidesFacesPolyhedraWithAllFacesEqual](#)

[createArraySidesFacesPolyhedraWithDifferentFaces](#)

[createIncidenceMatricesForCLI](#)

[createJacobianMatrices](#)

[createShapeFunction](#)

[createTransformationMatrices](#)

[evaluateOrderNumber](#)

[loadFromJson](#)

[meshPicker](#)

[modelPicker](#)

[printSplashScreen](#)

[projectToPlane](#)

[saveToJson](#)

[sortPolygonVertices](#)

[stepPicker](#)

[studyPicker](#)

[validateInput](#)

4.2. Funzioni raggruppate per compito

Funzioni CLI

FUNZIONE	SCOPO
<u>checkConnection</u>	Verificare connessione server COMSOL
<u>componentPicker</u>	Selezione componente
<u>createIncidenceMatricesForCLI</u>	Creazione matrici di incidenza
<u>evaluateOrderNumber</u>	Valutare ordine elementi mesh
<u>meshPicker</u>	Selezionare mesh
<u>modelPicker</u>	Selezionare modello
<u>printSplashScreen</u>	Stampare splash screen
<u>stepPicker</u>	Selezionare step
<u>studyPicker</u>	Selezionare studio
<u>validateInput</u>	Validare input utente

Funzioni matrici di incidenza

FUNZIONE	SCOPO
<u>createArrayBfdBfePolyhedraWithAllFacesEqual</u>	Mat. FD-FE
<u>createArrayFacesElementsPolyhedraWithAllFacesEqual</u>	Mat. F-E
<u>createArrayNodesFacesPolyhedraWithAllFacesEqual</u>	Mat. N-F
<u>createArrayNodesSidesPolyhedraWithAllFacesEqual</u>	Mat N-S
<u>createArraySidesElementsPolyhedraWithAllFacesEqual</u>	Mat. S-E
<u>createArraySidesFacesPolyhedraWithAllFacesEqual</u>	Mat. S-F
<u>createArrayBfdBfePolyhedraWithDifferentFaces</u>	Mat. FD-FE
<u>createArrayFacesElementsPolyhedraWithDifferentFaces</u>	Mat. F-E
<u>createArrayNodesFacesPolyhedraWithDifferentFaces</u>	Mat. N-F
<u>createArrayNodesSidesPolyhedraWithDifferentFaces</u>	Mat N-S
<u>createArraySidesElementsPolyhedraWithDifferentFaces</u>	Mat. S-E
<u>createArraySidesFacesPolyhedraWithDifferentFaces</u>	Mat. S-F

Funzioni shape functions

FUNZIONE	SCOPO
<u>createShapeFunctions</u>	Calcola le funzioni di forma nodali

Funzioni matrici jacobiane

FUNZIONE	SCOPO
<code>createJacobianMatrices</code>	Calcola le matrici jacobiane degli elementi

Funzioni matrici trasformazione

FUNZIONE	SCOPO
<code>createTransformationMatrices</code>	Calcola le matrici di trasformazione degli elementi

Funzioni utilità

FUNZIONE	SCOPO
<code>checkFaceInDomain</code>	Verificare faccia mesh in dominio
<code>computePlaneEquationFromPoints</code>	Calcolare equazione piano dati tre punti
<code>loadFromJson</code>	Caricare da un file Json
<code>projectToPlane</code>	Proiettare i punti su un piano
<code>saveToJson</code>	Salvare su un file Json
<code>sortPolygonVertices</code>	Ordinare vertici poligono verso antiorar.

4.3. Documentazione

`checkConnection`

Descrizione

Questa funzione si occupa di effettuare un tentativo di connessione con un server COMSOL.

Sintassi

```
isConnected = checkConnection()
```

Parametri di Input

Nessun parametro di input

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>isConnected</code>	boolean	Valore booleano che indica se l'istanza MATLAB è collegata a una istanza del server COMSOL tramite il LiveLink

`checkFaceInDomain`

Descrizione

Questa funzione si occupa di verificare se una faccia di un elemento di una mesh è compresa in una faccia del dominio.

Sintassi

```
in_domain = checkFaceInDomain(domain_face, mesh_face)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
<code>domain_face</code>	Nx3 matrix of double	Matrice Nx3 dove ogni riga è un vertice del dominio [x, y, z]
<code>mesh_face</code>	Nx3 matrix of double	Matrice Nx3, dove ogni riga è un vertice dell'elemento della mesh [x, y, z]

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>in_domain</code>	boolean	Valore booleano che indica se la faccia della mesh è all'interno della faccia del dominio

`componentPicker`

Descrizione

Questa funzione si occupa di permettere all'utente la scelta del componente di interesse, prelevando i componenti disponibili nel modello e mostrandoli a schermo prima della scelta.

Sintassi

```
selectedComponent = componentPicker(model)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
model	ModelClient Obj	Oggetto di tipo ModelClient risultato della chiamata <i>mphload</i> sul percorso di un modello COMSOL

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
selectedComponent	ModelNodeClient Obj	Oggetto di tipo ModelNodeClient utile per l'estrazione di geometria e mesh

computePlaneEquationFromPoints

Descrizione

Questa funzione si occupa di calcolare l'equazione del piano passante per tre punti non allineati.

Sintassi

```
plane_eq = computePlaneEquationFromPoints(p1, p2, p3)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
p1, p2, p3	1x3 matrix of double	Punto dello spazio rappresentato come un vettore di coordinate [x, y, z]

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
plane_eq	1x4 matrix of double	Coefficienti dell'equazione del piano passante per i tre punti specificati

`createArrayBfdBfePolyhedraWithAllFacesEqual`

Descrizione

Questa funzione si occupa di generare la matrice **FACCE_FRONTIERA_DOMINIO-FACCE_FRONTIERA_ELEMENTO** per elementi di mesh che sono poliedri con tutte le facce uguali (tetraedri, esaedri).

Sintassi

```
arrayBoundaryFacesDomainBoundaryFacesElement =  
createArrayBfdBfePolyhedraWithAllFacesEqual(  
    model,  
    tableNodesBoundaryFaces,  
    tableNodalCoordinates,  
    selectedComponentGeometryTag,  
    numberOfBoundary,  
    elementType,  
    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
model	ModelClient Obj	Oggetto di tipo ModelClient risultato della chiamata <i>mphload</i> sul percorso di un modello COMSOL
tableNodesBoundaryFaces	NxM matrix of double	Matrice NODI-FACCE_FRONTIERA
tableNodalCoordinates	NxM matrix of double	Matrice COORDINATE NODALI

selectedComponentGeometryTag	string	Stringa contenente il tag della geometria del componente
numberOfBoundary	integer	Numero dei domini che compongono la geometria del componente
elementType	string	Stringa contenente il tipo di elemento che compone la mesh
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arrayBoundaryFacesDomainBoundaryFacesElement	NxM matrix of double	Matrice NxM BFD-BFE

`createArrayBfdBfePolyhedraWithDifferentFaces`

Descrizione

Questa funzione si occupa di generare la matrice `FACCE_FRONTIERA_DOMINIO-FACCE_FRONTIERA_ELEMENTO` per elementi di mesh che sono poliedri con facce differenti tra loro (prismi, piramidi).

Sintassi

```
arrayBoundaryFacesDomainBoundaryFacesElement =
createArrayBfdBfePolyhedraWithDifferentFaces(
    model,
    tableNodesBoundaryFaces,
    tableNodalCoordinates,
    selectedComponentGeometryTag,
    numberOfBoundary,
    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
model	ModelClient Obj	Oggetto di tipo ModelClient risultato della chiamata <i>mphload</i> sul percorso di un modello COMSOL
tableNodesBoundaryFaces	NxM matrix of double	Matrice NODI-FACCE_FRONTIERA
tableNodalCoordinates	NxM matrix of double	Matrice COORDINATE NODALI
selectedComponentGeometryTag	string	Stringa contenente il tag della geometria del componente
numberOfBoundary	integer	Numero dei domini che compongono la geometria del componente
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arrayBoundaryFacesDomainBoundaryFacesElement	NxM matrix of double	Matrice NxM BFD-BFE

`createArrayFacesElementsPolyhedraWithAllFacesEqual`

Descrizione

Questa funzione si occupa di creare la matrice FACCE-ELEMENTI per elementi di mesh che sono poliedri con tutte le facce uguali (tetraedri, esaedri).

Sintassi

```
arrayFacesElements =
createArrayFacesElementsPolyhedraWithAllFacesEqual(
    tableNodesElements,
```

```
tableNodesFaces,  
elementType)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesElements	NxM matrix of double	Matrice NODI-ELEMENTI
tableNodesFaces	NxM matrix of double	Matrice NODI-FACCE
elementType	string	Stringa contenente il tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arrayFacesElements	NxM matrix of double	Matrice FACCE-ELEMENTI

```
createArrayFacesElementsPolyhedraWithDifferentFaces
```

Descrizione

Questa funzione si occupa di creare la matrice FACCE-ELEMENTI per elementi di mesh che sono poliedri con facce differenti tra loro (prismi, piramidi).

Sintassi

```
arrayFacesElements =  
createArrayFacesElementsPolyhedraWithDifferentFaces(  
    tableNodesElements,  
    tableNodesFaces,  
    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesElements	NxM matrix of double	Matrice NODI-ELEMENTI
tableNodesFaces	NxM matrix of double	Matrice NODI-FACCE
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arrayFacesElements	NxM matrix of double	Matrice FACCE-ELEMENTI

```
createArrayNodesFacesPolyhedraWithAllFacesEqual
```

Descrizione

Questa funzione si occupa di creare la matrice NODI-FACCE (sia per tutte le facce, che per le sole facce di frontiera) per tutti gli elementi di mesh che sono poliedri con tutte le facce uguali (tetraedri, esaedri).

Sintassi

```
[arrayNodesFaces, arrayNodesBoundaryFaces] =
createArrayNodesFacesPolyhedraWithAllFacesEqual(
    tableNodesElements,
    elementType,
    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesElements	NxM matrix of double	Matrice NODI-ELEMENTI
elementType	string	Stringa contenente il tipo di elemento che compone la mesh
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arrayNodesFaces	NxM matrix of double	Matrice NODI-FACCE
arrayNodesBoundaryFaces	NxM matrix of double	Matrice NODI-FACCE_FRONTIERA

```
createArrayNodesFacesPolyhedraWithDifferentFaces
```

Descrizione

Questa funzione si occupa di creare la matrice NODI-FACCE (sia per tutte le facce, che per le sole facce di frontiera) per tutti gli elementi di mesh che sono poliedri con facce differenti tra loro (prismi, piramidi).

Sintassi

```
[arrayNodesFaces, arrayNodesBoundaryFaces] =
createArrayNodesFacesPolyhedraWithDifferentFaces (
    tableNodesElements,
    elementType,
    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesElements	NxM matrix of double	Matrice NODI-ELEMENTI
elementType	string	Stringa contenente il tipo di elemento che compone la mesh
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arrayNodesFaces	NxM matrix of double	Matrice NODI-FACCE
arrayNodesBoundaryFaces	NxM matrix of double	Matrice NODI-FACCE_FRONTIERA

```
createArrayNodesSidesPolyhedraWithAllFacesEqual
```

Descrizione

Questa funzione si occupa di creare la matrice NODI-LATI per tutte le facce, per elementi di mesh che sono poliedri con tutte le facce uguali (tetraedri, esaedri).

Sintassi

```
arrayNodesSides =
createArrayNodesSidesPolyhedraWithAllFacesEqual (
    tableNodesFaces,
```

```
elementType,  
elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesFaces	NxM matrix of double	Matrice NODI-FACCE
elementType	string	Stringa contenente il tipo di elemento che compone la mesh
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arrayNodesSides	NxM matrix of double	Matrice NODI-LATI

```
createArrayNodesSidesPolyhedraWithDifferentFaces
```

Descrizione

Questa funzione si occupa di creare la matrice NODI-LATI per tutte le facce, per elementi di mesh che sono poliedri con facce differenti tra loro (prismi, piramidi).

Sintassi

```
arrayNodesSides =  
createArrayNodesSidesPolyhedraWithDifferentFaces (  
                                                    tableNodesFaces,  
                                                    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesFaces	NxM matrix of double	Matrice NODI-FACCE
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
-----------	------	-------------

arrayNodesSides	NxM matrix of double	Matrice NODI-LATI
-----------------	----------------------	-------------------

```
createArraySidesElementsPolyhedraWithAllFacesEqual
```

Descrizione

Questa funzione si occupa di creare la matrice LATI-ELEMENTI, per elementi di mesh che sono poliedri con tutte le facce uguali (tetraedri, esaedri).

Sintassi

```
arraySidesElements =
createArraySidesElementsPolyhedraWithAllFacesEqual(
    tableNodesElements,
    tableNodesSides,
    elementType,
    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesElements	NxM matrix of double	Matrice NODI-ELEMENTI
tableNodesSides	NxM matrix of double	Matrice NODI-LATI
elementType	string	Stringa contenente il tipo di elemento che compone la mesh
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arraySidesElements	NxM matrix of double	Matrice LATI-ELEMENTI

```
createArraySidesElementsPolyhedraWithDifferentFaces
```

Descrizione

Questa funzione si occupa di creare la matrice LATI-ELEMENTI per elementi di mesh che sono poliedri con facce differenti tra loro (prismi, piramidi).

Sintassi

```
arraySidesElements =  
createArraySidesElementsPolyhedraWithDifferentFaces (  
                                                    tableNodesElements,  
                                                    tableNodesSides,  
                                                    elementType,  
                                                    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesElements	NxM matrix of double	Matrice NODI-ELEMENTI
tableNodesSides	NxM matrix of double	Matrice NODI-LATI
elementType	string	Stringa contenente il tipo di elemento che compone la mesh
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arraySidesElements	NxM matrix of double	Matrice LATI-ELEMENTI

```
createArraySidesFacesPolyhedraWithAllFacesEqual
```

Descrizione

Questa funzione si occupa di creare la matrice LATI-FACCE, per elementi di mesh che sono poliedri con tutte le facce uguali (tetraedri, esaedri).

Sintassi

```
arraySidesFaces =  
createArraySidesFacesPolyhedraWithAllFacesEqual (  
                                                    tableNodesFaces,  
                                                    tableNodesSides,  
                                                    elementType,  
                                                    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesFaces	NxM matrix of double	Matrice NODI-FACCE
tableNodesSides	NxM matrix of double	Matrice NODI-LATI
elementType	string	Stringa contenente il tipo di elemento che compone la mesh
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arraySidesFaces	NxM matrix of double	Matrice LATI-FACCE

`createArraySidesFacesPolyhedraWithDifferentFaces`

Descrizione

Questa funzione si occupa di creare la matrice LATI-FACCE, per elementi di mesh che sono poliedri con facce differenti tra loro (prismi, piramidi).

Sintassi

```
arraySidesFaces =  
createArraySidesFacesPolyhedraWithDifferentFaces (  
                                                    tableNodesFaces,  
                                                    tableNodesSides,  
                                                    elementsOrder)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
tableNodesFaces	NxM matrix of double	Matrice NODI-FACCE
tableNodesSides	NxM matrix of double	Matrice NODI-LATI
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
arraySidesFaces	NxM matrix of double	Matrice LATI-FACCE

Descrizione

Questa funzione si occupa di calcolare le matrici di incidenza selezionate per la mesh specificata del modello scelto.

Sintassi

```
incidenceMatrices = createIncidenceMatricesForCLI(  
    model,  
    selectedComponentGeometryTag,  
    geometryTagPos,  
    meshdata,  
    meshdataTypeList,  
    searchedString,  
    elementsOrder,  
    flagsStruct,  
    fileName,  
    flagFacesEqual)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
model	ModelClient Obj	Oggetto di tipo ModelClient risultato della chiamata <i>mphload</i> sul percorso di un modello COMSOL
selectedComponentGeometryTag	string	Stringa contenente il tag della geometria del componente
geometryTagPos	integer	Intero che indica la posizione del tag della geometria di interesse all'interno della lista dei tag di tutte le geometrie del modello
meshdata	struct	Struttura contenente le informazioni relative alla mesh
meshdataTypeList	array of string	Lista dei tipi di elemento di cui vi

		sono informazioni nella struttura meshdata
searchedString	string	Stringa contenente il tipo di elemento di mesh contenuto nella mesh selezionata
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh
flagsStruct	struct	Struttura contenente due valori booleani per ciascuna matrice, utile per pilotare la costruzione delle matrici e il loro salvataggio
fileName	char	Lista di caratteri contenente il percorso e il nome del file in cui salvare sul disco la struct
flagFacesEqual	boolean	Valore booleano che indica la presenza di elementi di mesh con facce tutte uguali, utile per permettere di chiamare la funzione di creazione della famiglia corretta

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
incidenceMatrices	struct	Struttura dati contenente tutti gli array delle matrici di incidenza

createJacobianMatrices

Descrizione

Questa funzione si occupa di calcolare le matrici jacobiane per gli elementi di mesh del modello scelto.

Sintassi

```
tableJacobianMatrices = createJacobianMatrices(  
    incidenceMatrices,  
    tableShapeFunctions);
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
incidenceMatrices	struct	Struttura dati contenente tutti gli array delle matrici di incidenza
tableShapeFunctions	table of string	Struttura dati contenente le funzioni di forma di ogni singolo nodo. Sulle righe ci sono i nodi e sull'unica colonna presente viene mostrata la funzione di forma sotto forma di stringa

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
tableJacobianMatrices	table	Struttura dati contenente informazioni circa le matrici jacobiane degli elementi, il loro determinante, l'inversa delle jacobiane e il determinante dell'inversa

`createShapeFunctions`

Descrizione

Questa funzione si occupa di calcolare le funzioni di forma per ogni nodo.

Sintassi

```
tableShapeFunctions = createShapeFunctions(  
    incidenceMatrices,  
    searchedString,  
    elementsOrder);
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
<code>incidenceMatrices</code>	struct	Struttura dati contenente tutti gli array delle matrici di incidenza
<code>searchedString</code>	string	Stringa contenente il tipo di elemento di mesh selezionato dall'utente
<code>elementsOrder</code>	integer	Numero di ordine del tipo di elemento che compone la mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>tableShapeFunctions</code>	table of string	Struttura dati contenente le funzioni di forma di ogni singolo nodo. Sulle righe ci sono i nodi e sull'unica colonna presente viene mostrata la funzione di forma sotto forma di stringa

`createTransformationMatrices`

Descrizione

Questa funzione si occupa di calcolare le matrici di trasformazione per gli elementi di mesh del modello scelto.

Sintassi

```
tableTransformationMatrices = createTransformationMatrices(  
    incidenceMatrices,  
    tableJacobianMatrices);
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
-----------	------	-------------

<code>incidenceMatrices</code>	struct	Struttura dati contenente tutti gli array delle matrici di incidenza
<code>tableJacobianMatrices</code>	table	Struttura dati contenente informazioni circa le matrici jacobiane degli elementi, il loro determinante, l'inversa delle jacobiane e il determinante dell'inversa

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>tableTransformationMatrices</code>	table	Struttura dati contenente informazioni circa le matrici di trasformazione degli elementi. Ogni colonna corrisponde ad un elemento, mentre nell'unica riga presente vi è un array contenente la matrice di trasformazione

`evaluateOrderNumber`

Descrizione

Funzione che si occupa di valutare il numero di ordine degli elementi di mesh della mesh selezionata.

Sintassi

```
elementsOrder = evaluateOrderNumber(model)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
<code>model</code>	ModelClient Obj	Oggetto di tipo ModelClient risultato della chiamata <i>mphload</i> sul percorso di un modello COMSOL

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
elementsOrder	integer	Numero di ordine del tipo di elemento che compone la mesh

loadFromJson

Descrizione

Questa funzione si occupa di caricare nel workspace base di MATLAB la struttura dati contenente le matrici di incidenza salvata nel file Json.

Sintassi

```
loadFromJson()
```

Parametri di Input

Nessun parametro di input

Parametri di Output

Nessun parametro di output

meshPicker

Descrizione

Questa funzione si occupa di permettere all'utente la scelta della mesh di interesse, prelevando le mesh disponibili nel componente e mostrandoli a schermo prima della scelta.

Sintassi

```
[selectedMesh, selectedMeshTag] = meshPicker(  
                                model,  
                                selectedComponent)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
-----------	------	-------------

<code>model</code>	ModelClient Obj	Oggetto di tipo ModelClient risultato della chiamata <i>mphload</i> sul percorso di un modello COMSOL
<code>selectedComponent</code>	ModelNodeClient Obj	Oggetto di tipo ModelNodeClient utile per l'estrazione di geometria e mesh

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>selectedMesh</code>	MeshSequenceClient Obj	Oggetto di tipo MeshSequenceClient
<code>selectedMeshTag</code>	string	Stringa contenente il tag della mesh selezionata

`modelPicker`

Descrizione

Questa funzione si occupa di permettere all'utente la selezione, dal suo file system locale, del modello di interesse da caricare poi in MATLAB per le elaborazioni.

Sintassi

```
model = modelPicker()
```

Parametri di Input

Nessun parametro di input

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>model</code>	ModelClient Obj	Oggetto di tipo ModelClient risultato della chiamata <i>mphload</i> sul percorso di un modello COMSOL

printSplashScreen

Descrizione

Questa funzione si occupa di stampare a schermo nella command window di MATLAB lo splash screen di MATSOL.

Sintassi

```
printSplashScreen()
```

Parametri di Input

Nessun parametro di input

Parametri di Output

Nessun parametro di output

projectToPlane

Descrizione

Questa funzione si occupa di proiettare i punti dello spazio 3D su un piano definito dalla normale.

Sintassi

```
[points_2D, T] = projectToPlane(points, normal, T)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
points	Nx3 matrix of double	Matrice Nx3, contenente i punti dello spazio rappresentato come un vettore di coordinate [x, y, z] da proiettare
normal	1x3 array of double	vettore che definisce il piano di proiezione
T	4x4 matrix of double	Matrice di rotazione, se fornita, viene utilizzata

		per la trasformazione; altrimenti, viene calcolata
--	--	---

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>points_2D</code>	Nx2 matrix of double	Vettore di punti in 2D, proiettati sul piano specificato
<code>T</code>	4x4 matrix of double	Matrice di rotazione, se fornita, viene utilizzata per la trasformazione; altrimenti, viene calcolata

`saveToJson`

Descrizione

Questa funzione si occupa di salvare la struttura dati contenente le matrici di incidenza in un file Json.

Sintassi

```
saveToJson(incidenceMatrices, fileName)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
<code>incidenceMatrices</code>	struct	Struttura dati contenente tutti gli array delle matrici di incidenza
<code>fileName</code>	char	Lista di caratteri contenente il percorso e il nome del file in cui salvare sul disco la struct

Parametri di Output

Nessun parametro di output

`sortPolygonVertices`

Descrizione

Questa funzione si occupa di ordinare i vertici di un poligono in senso antiorario.

Sintassi

```
sorted_points = sortPolygonVertices(points)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
sorted_points	Nx3 matrix of double	Matrice contenente i punti nello spazio 3D [x,y,z] da ordinare

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
points	Nx3 matrix of double	Matrice contenente i punti nello spazio 3D [x, y, z] ordinati in senso antiorario

stepPicker

Descrizione

Questa funzione si occupa di permettere all'utente la selezione dello step di interesse tra quelli disponibili per lo studio selezionato.

Sintassi

```
[selectedStep, selectedStepTag] = stepPicker(selectedStudy)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
selectedStudy	StudyClient Obj	Oggetto di tipo StudyClient utilizzato per l'estrazione degli step

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>selectedStep</code>	StudyFeatureClient Obj	Oggetto di tipo StudyFeatureClient
<code>selectedStepTag</code>	string	Stringa contenente il tag dello step selezionato

`studyPicker`

Descrizione

Questa funzione si occupa di permettere all'utente la selezione dello studio di interesse tra quelli disponibili per il modello selezionato.

Sintassi

```
[selectedStudy, selectedStudyTag] = studyPicker(model)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
<code>model</code>	ModelClient Obj	Oggetto di tipo ModelClient risultato della chiamata <i>mphload</i> sul percorso di un modello COMSOL

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
<code>selectedStudy</code>	StudyClient Obj	Oggetto di tipo StudyClient
<code>selectedStudyTag</code>	string	Stringa contenente il tag dello studio selezionato

`validateInput`

Descrizione

Questa funzione si occupa di validare l'input inserito dall'utente.

Sintassi

```
choice = validateInput(maxNumberOfChoice)
```

Parametri di Input

PARAMETRO	TIPO	DESCRIZIONE
maxNumberOfChoice	integer	Intero che definisce il numero massimo che l'utente può inserire

Parametri di Output

PARAMETRO	TIPO	DESCRIZIONE
choice	integer	Intero che definisce la scelta dell'utente