Analisi di risposta sismica locale su area vasta usando risorse HPC

Abstract
Lo Spoke 5 “Environment & Natural Disasters” del Centro Nazionale in High Performance Computing (HPC), Big Data and Quantum Computing si pone come obiettivo lo sviluppo di metodologie per la modellazione di processi fisici naturali capaci di provocare disastri tali da indurre perdite di vite umane e danni socio-economici. Tra i processi naturali fonte di disastro rientrano i terremoti, che, nel propagarsi all’interno di depositi di terreno, subiscono fenomeni di amplificazione in superficie in grado di provocare danni alle strutture e alle infrastrutture. Di conseguenza, la previsione degli effetti di amplificazione diviene di fondamentale importanza per la riduzione del rischio sismico su aree fortemente esposte. In tale contesto, la modellazione numerica della propagazione sismica su larga scala richiede l’uso di modelli numerici complessi e l’adozione di tecniche di calcolo parallelo attraverso sistemi HPC.
La tesi propone una metodologia operativa di modellazione FEM dinamica lineare e non lineare 1D, 2D e 3D, applicata alla risposta sismica di un pendio naturale ideale. Le simulazioni hanno tenuto conto dell’interazione dinamica solido-fluido attraverso l’implementazione della formulazione u-p e sono state svolte utilizzando i codici di calcolo Plaxis e OpenSees. Quest’ultimo permette la risoluzione di modelli FEM in parallelo attraverso l’uso di infrastrutture di calcolo ad alte prestazioni. La risposta ciclica dei terreni è stata descritta adottando modelli visco-elastici e visco-elasto-plastici con smorzamento di Rayleigh. In particolare, nel codice OpenSees è stato adottato il modello Pressure Independent Multi Yield con incrudimento isotropo e cinematico, mentre in Plaxis si è utilizzato l’Hardening Soil model with small strain stiffness.
Il confronto tra i risultati 1D e 2D ha permesso di validare le previsioni di OpenSees rispetto a quelle di un software agli elementi finiti consolidato nella letteratura scientifica e, più in generale, di identificare i vantaggi ed i limiti dell'adozione di diversi strumenti numerici per la valutazione della risposta sismica di un pendio naturale. Infine, l’elaborazione delle accelerazioni massime di superficie ottenute dall’analisi 3D ha consentito di definire mappe di pericolosità sismica locale per l’intera area oggetto di studio.

In copertina: mappa di amplificazione sismica locale.

DICATECh

2026

Gianluca Cavallo

Analisi di risposta sismica locale su area vasta usando risorse HPC

09

D.R.S.A.T.E.

PhD in Rischio Sviluppo Ambientale Territoriale ed Edilizio

Coordinatore: Prof. Francesco Fiorito

XXXVIII CICLO
Curriculum: CEAR-05/A - Geotecnica

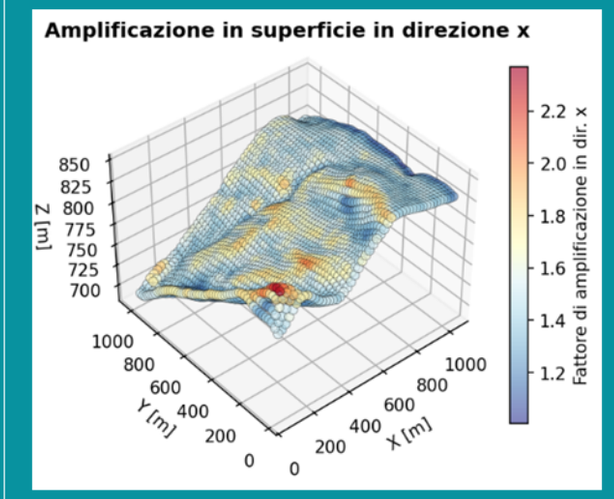
DICATECh
Dipartimento di Ingegneria Civile Ambientale del Territorio Edile e Chimica

Gianluca Cavallo

Analisi di risposta sismica locale su area vasta usando risorse HPC

Prof.ssa Federica Cotecchia
DICATECh
Politecnico di Bari
Prof. Gaetano Elia
DICATECh
Politecnico di Bari

Dott.ssa Annamaria di Lernia
DICATECh
Politecnico di Bari



POLITECNICO DI BARI

09

2026

Abstract
Lo Spoke 5 “Environment & Natural Disasters” del Centro Nazionale in High Performance Computing (HPC), Big Data and Quantum Computing si pone come obiettivo lo sviluppo di metodologie per la modellazione di processi fisici naturali capaci di provocare disastri tali da indurre perdite di vite umane e danni socio-economici. Tra i processi naturali fonte di disastro rientrano i terremoti, che, nel propagarsi all’interno di depositi di terreno, subiscono fenomeni di amplificazione in superficie in grado di provocare danni alle strutture e alle infrastrutture. Di conseguenza, la previsione degli effetti di amplificazione diviene di fondamentale importanza per la riduzione del rischio sismico su aree fortemente esposte. In tale contesto, la modellazione numerica della propagazione sismica su larga scala richiede l’uso di modelli numerici complessi e l’adozione di tecniche di calcolo parallelo attraverso sistemi HPC.
La tesi propone una metodologia operativa di modellazione FEM dinamica lineare e non lineare 1D, 2D e 3D, applicata alla risposta sismica di un pendio naturale ideale. Le simulazioni hanno tenuto conto dell’interazione dinamica solido-fluido attraverso l’implementazione della formulazione u-p e sono state svolte utilizzando i codici di calcolo Plaxis e OpenSees. Quest’ultimo permette la risoluzione di modelli FEM in parallelo attraverso l’uso di infrastrutture di calcolo ad alte prestazioni. La risposta ciclica dei terreni è stata descritta adottando modelli visco-elastici e visco-elasto-plastici con smorzamento di Rayleigh. In particolare, nel codice OpenSees è stato adottato il modello Pressure Independent Multi Yield con incrudimento isotropo e cinematico, mentre in Plaxis si è utilizzato l’Hardening Soil model with small strain stiffness.
Il confronto tra i risultati 1D e 2D ha permesso di validare le previsioni di OpenSees rispetto a quelle di un software agli elementi finiti consolidato nella letteratura scientifica e, più in generale, di identificare i vantaggi ed i limiti dell'adozione di diversi strumenti numerici per la valutazione della risposta sismica di un pendio naturale. Infine, l’elaborazione delle accelerazioni massime di superficie ottenute dall’analisi 3D ha consentito di definire mappe di pericolosità sismica locale per l’intera area oggetto di studio.

In copertina: mappa di amplificazione sismica locale.

DICATECh

2026

Gianluca Cavallo

Analisi di risposta sismica locale su area vasta usando risorse HPC

09

D.R.S.A.T.E.

PhD in Rischio Sviluppo Ambientale Territoriale ed Edilizio

Coordinatore: Prof. Francesco Fiorito

XXXVIII CYCLE
Curriculum: CEAR-05/A - Geotecnica

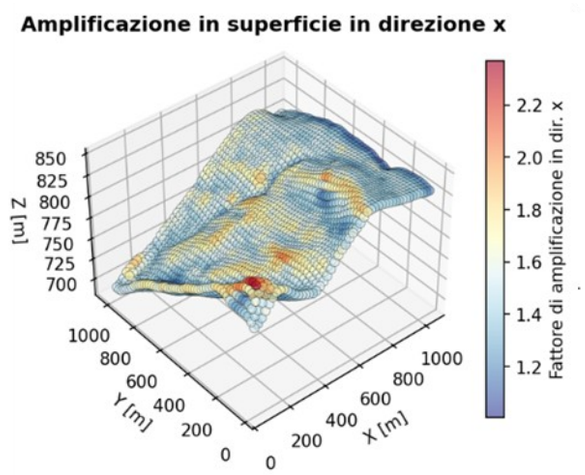
DICATECh
Dipartimento di Ingegneria Civile Ambientale del Territorio Edile e Chimica

Gianluca Cavallo

Analisi di risposta sismica locale su area vasta usando risorse HPC

Prof.ssa Federica Cotecchia
DICATECh
Politecnico di Bari
Prof. Gaetano Elia
DICATECh
Politecnico di Bari

Dott.ssa Annamaria di Lernia
DICATECh
Politecnico di Bari



POLITECNICO DI BARI

09

2026

D.R.S.A.T.E.

POLITECNICO DI BARI

09

2026

PhD in Rischio Sviluppo Ambientale Territoriale ed Edilizio

Coordinatore: Prof. Francesco Fiorito

XXXVIII CICLO
Curriculum: CEAR-05/A - Geotecnica

DICATECh
Dipartimento di Ingegneria Civile Ambientale del Territorio Edile e Chimica

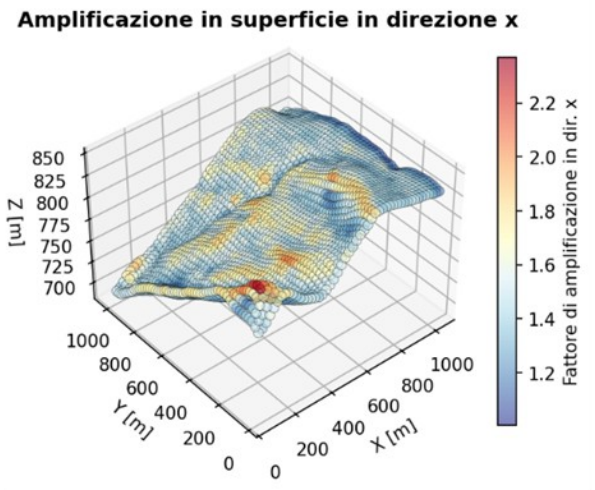
Gianluca Cavallo

Analisi di risposta sismica locale su area vasta usando risorse HPC

Prof.ssa Federica Cotecchia
DICATECh
Politecnico di Bari

Dott.ssa Annamaria di Lernia
DICATECh
Politecnico di Bari

Prof. Gaetano Elia
DICATECh
Politecnico di Bari



D.R.S.A.T.E.

POLITECNICO DI BARI

09

2026

PhD in Risk and Environmental, Territorial and Building Development

Coordinator: Prof. Francesco Fiorito

XXXVIII CYCLE
Curriculum: CEAR-05/A - Geotecnica

DICATECh
Department of Civil, Environmental, Land Building Engineering and Chemistry

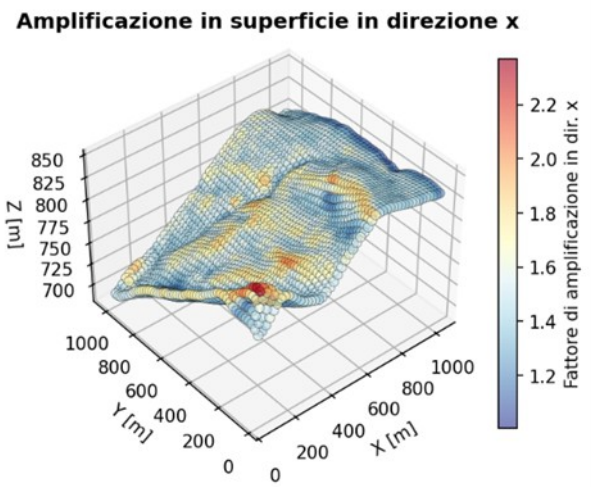
Gianluca Cavallo

Seismic site response analysis over large areas with HPC resources

Prof.ssa Federica Cotecchia
DICATECh
Politecnico di Bari

Dott.ssa Annamaria di Lernia
DICATECh
Politecnico di Bari

Prof. Gaetano Elia
DICATECh
Politecnico di Bari



EXTENDED ABSTRACT

Spoke 5 "Environment & Natural Disasters" of the National Centre for High Performance Computing (HPC), Big Data and Quantum Computing aims to develop methodologies for modelling natural physical processes causing disasters that result in loss of human life and socio-economic damage. Earthquakes are among those natural processes that cause disasters, especially because, after their propagation through soil deposits, surface amplification phenomena are capable of causing damage to structures and infrastructure. Consequently, the accurate prediction of amplification effects becomes fundamentally important for seismic risk reduction in highly exposed areas. In this context, large-scale numerical modelling of seismic wave propagation processes requires the use of complex numerical models and the adoption of parallel computing techniques through HPC systems.

This thesis proposes an operational methodology for 1D, 2D, and 3D linear and nonlinear dynamic FEM modelling applied to the seismic response of an ideal natural slope. The simulations accounted for dynamic solid-fluid interaction through the implementation of the u-p formulation and were performed using the PLAXIS and OPENSEES software. The latter enables the solution of FEM models in parallel through high-performance computing infrastructure. The cyclic response of the soils was described by visco-elastic and visco-elasto-plastic models with Rayleigh damping. Specifically, the Pressure Independent Multi Yield model with isotropic and kinematic hardening was adopted in OPENSEES, while the Hardening Soil model with small strain stiffness was used in PLAXIS.

The comparison between 1D and 2D results allowed the validation of OPENSEES predictions against those of a finite element software well-established in the scientific literature and, more generally, the identification of the advantages and limitations

of adopting different numerical tools for the assessment of the seismic response of a natural slope. Finally, processing the maximum surface accelerations obtained from the 3D analysis enabled the definition of local seismic hazard maps for the entire study area.

keywords

Site response analysis, advanced soil constitutive modelling, FE method, High Performance Computing

EXTENDED ABSTRACT

Lo Spoke 5 “Environment & Natural Disasters” del Centro Nazionale in High Performance Computing (HPC), Big Data and Quantum Computing si pone come obiettivo lo sviluppo di metodologie per la modellazione di processi fisici naturali capaci di provocare disastri tali da indurre perdite di vite umane e danni socio-economici. Tra i processi naturali fonte di disastro rientrano i terremoti, che, nel propagarsi all’interno di depositi di terreno, subiscono fenomeni di amplificazione in superficie in grado di provocare danni alle strutture e alle infrastrutture. Di conseguenza, la previsione degli effetti di amplificazione diviene di fondamentale importanza per la riduzione del rischio sismico su aree fortemente esposte. In tale contesto, la modellazione numerica della propagazione sismica su larga scala richiede l’uso di modelli numerici complessi e l’adozione di tecniche di calcolo parallelo attraverso sistemi HPC.

La tesi propone una metodologia operativa di modellazione FEM dinamica lineare e non lineare 1D, 2D e 3D, applicata alla risposta sismica di un pendio naturale ideale. Le simulazioni hanno tenuto conto dell’interazione dinamica solido-fluido attraverso l’implementazione della formulazione u-p e sono state svolte utilizzando i codici di calcolo PLAXIS e OPENSEES. Quest’ultimo permette la risoluzione di modelli FEM in parallelo attraverso l’uso di infrastrutture di calcolo ad alte prestazioni. La risposta ciclica dei terreni è stata descritta adottando modelli visco-elastici e visco-elasto-plastici con smorzamento di Rayleigh. In particolare, nel codice OPENSEES è stato adottato il modello Pressure Independent Multi Yield con incrudimento isotropo e cinematico, mentre in PLAXIS si è utilizzato l’Hardening Soil model with small strain stiffness.

Il confronto tra i risultati 1D e 2D ha permesso di validare le previsioni di OPENSEES rispetto a quelle di un software agli elementi finiti consolidato nella letteratura

scientifica e, più in generale, di identificare i vantaggi ed i limiti dell'adozione di diversi strumenti numerici per la valutazione della risposta sismica di un pendio naturale. Infine, l'elaborazione delle accelerazioni massime di superficie ottenute dall'analisi 3D ha consentito di definire mappe di pericolosità sismica locale per l'intera area oggetto di studio.

keywords

Risposta sismica locale, modellazione costitutiva avanzata, metodo FEM, HPC

INDICE

INTRODUZIONE

Il presente lavoro di tesi di dottorato riassume le attività di ricerca svolte nell'ambito dello Spoke 5, intitolato "Ambiente e disastri naturali", che afferisce al Centro Nazionale di Ricerca (CN) in HPC, Big Data and Quantum Computing (HPC), finanziato dai fondi europei Next Generation UE - PNRR.

Lo Spoke 5 si propone di sviluppare metodologie per il monitoraggio delle strutture fisiche artificiali (edifici, strade ecc.) e degli ambienti naturali (corsi d'acqua, pendii ecc.) in grado di mitigare i rischi collegati al loro deterioramento. Nell'ambio dello Spoke 5 lavora un gruppo fortemente multidisciplinare, che spazia dalla ricerca di base nelle aree della matematica e della fisica, alle specializzazioni nel campo computazionale avanzato, sino agli ambiti di ricerca nei settori delle scienze della terra e dell'ingegneria civile e geo-ambientale. Inoltre, si intende tradurre i risultati delle ricerche condotte in chiari indicatori di impatto economico per sviluppare e proporre soluzioni per il monitoraggio dell'ambiente che abbiano una ricaduta positiva sull'intera società civile.

L'obiettivo è quello di utilizzare le capacità computazionali del Centro Nazionale per sviluppare i cosiddetti "gemelli digitali" o "digital twin", ovvero dei modelli virtuali di ambienti ed infrastrutture reali che si intendono studiare per prevederne il comportamento in riferimento ai disastri naturali o all'evoluzione delle condizioni ambientali. L'innovazione risiede nella previsione dell'interazione tra i sistemi attraverso l'acquisizione dei dati e l'analisi dei modelli virtuali in tempo reale con una possibile previsione degli effetti.

All'interno dello Spoke 5 le attività sono state suddivise in diverse tematiche, afferenti ad un particolare fenomeno fisico e/o disastro naturale. Nello specifico, sono stati definiti i seguenti gruppi di ricerca:

(0) "Cross phenomena"
(1) "Landslides" + "Floods & Pollutant spilling"
(2) "Earthquakes"
(3) "Volcanic eruptions"
(4) "Environmental sustainability development"

Tra i processi naturali fonte di disastro sopra elencati rientrano i terremoti, che, nel propagarsi all'interno di depositi di terreno, subiscono fenomeni di amplificazione in superficie in grado di esasperare gli effetti in termini di danni alle strutture e alle infrastrutture. Di conseguenza, la previsione degli effetti indotti da tali fenomeni diviene di fondamentale importanza per la riduzione del rischio sismico su aree fortemente esposte.

In generale, lo studio dei fenomeni di propagazione sismica può essere eseguito adottando schemi di calcolo monodimensionali, bidimensionali o tridimensionali, implementati in codici di calcolo agli elementi finiti. In particolare, per la modellazione numerica di tali processi geotecnici su larga scala risulta necessario l'uso di modelli tridimensionali (3D) complessi e l'adozione di tecniche di calcolo parallelo attraverso sistemi sofisticati HPC (High Performance Computing), come quelli forniti dal centro di calcolo RECAS di Bari gestito dall'INFN.

Il principale obiettivo del lavoro di tesi è quello di sviluppare degli strumenti avanzati per la generazione ed implementazione di modelli numerici 3D, finalizzati allo studio della risposta sismica locale su area vasta, in grado di sfruttare le risorse di calcolo HPC per il calcolo in parallelo. In particolare, è stata definita una metodologia per la programmazione ed implementazione di modelli e strumenti di calcolo su sistemi HPC o Cloud Computing messi a disposizione dal centro di calcolo RECAS, alternativi a quelli già presenti in centri avanzati come il SimCenter in California ed in Texas. Al fine di effettuare le analisi in sicurezza sui sistemi Linux a disposizione è stato necessario programmare in autonomia un container Singularity con all'interno il software necessario alla risoluzione dei modelli su unità in parallelo, OPENSEESMP ed OPENSEESSP.

Il centro di calcolo RECAS dell'Istituto Nazionale di Fisica Nucleare (sede di Bari) ha una potenza pari a 4000 singole unità di elaborazione (core) distribuiti su 250 nodi di calcolo e circa 1700 TB di spazio disco che si aggiunge a 128 server, ciascuno dotato di 64 core e 256 GB di RAM, per un totale di più di 8000 core e 3500 TB di spazio disco. Sulla infrastruttura RECAS possono girare contemporaneamente più di 12000 applicazioni. In aggiunta, il data center RECAS-Bari ospita un piccolo cluster

dedicato al calcolo HPC per eseguire applicazioni che richiedono un elevato numero di unità di elaborazione che lavorano in parallelo scambiandosi continuamente informazioni l'una con l'altra. Il cluster HPC consiste di 20 server, ciascuno con 40 core, per complessivi 800 core. Ogni server è dotato di un acceleratore grafico (NVIDIA K40) particolarmente utile per accelerare l'esecuzione di alcuni calcoli. Tutti i server sono interconnessi, con tecnologia InfiniBand a bassa latenza, al fine di aumentare la velocità di scambio dei dati tra gli stessi. Si tratta di un cluster HPC in grado di eseguire applicazioni che usano fino a 800 core in parallelo, potente a sufficienza per eseguire la grande maggioranza delle applicazioni così come per testare algoritmi prima di portarli su calcolatori HPC di maggiori dimensioni.

Partendo dalla modellazione numerica di casi 1D semplificati, per i quali il risultato delle simulazioni può essere validato attraverso il confronto con approcci di comprovata validità, sono stati sviluppati modelli numerici di complessità via via crescente, passando da schemi di calcolo 2D a modelli 3D implementati nel codice agli elementi finiti OPENSEES (McKenna et al., 2000). Traendo ispirazione dal caso reale di un pendio naturale sito a Chieuti (Foggia) in Puglia, l'ultima fase del lavoro è rappresentata dallo sviluppo di un modello numerico 3D per lo studio della risposta sismica locale, che include la complessa topografia dell'area.

La tesi è articolata in 10 capitoli. Nel Capitolo 1 si discute dello stato dell'arte dei software e delle applicazioni open-source maggiormente diffuse per lo studio del problema della propagazione sismica. Attenzione particolare è rivolta alla valutazione della possibilità di tali software di essere scalati per la risoluzione in parallelo dei sistemi lineari. Inoltre, è stata verificata la disponibilità di librerie di modelli costitutivi elasto-plastici avanzati o semi-avanzati, che descrivano accuratamente la risposta ciclica dei terreni. Gli elementi di base per la corretta modellazione del fenomeno di propagazione sismica, come smorzatori e molle puntuali, sono invece a disposizione nella totalità dei software analizzati o facilmente implementabili.

La visualizzazione del risultato per la fase di post-processing, invece, è stata la parte più trascurata in molte applicazioni e lasciata spesso in mano all'utente e alle sue capacità di programmazione e operatività con i sistemi.

La ricerca e la selezione della soluzione più adatta alle esigenze di sviluppo e calcolo sono giustificate dal sistema operativo adottato dal centro di calcolo RECAS che, come anche gli altri servizi di computing avanzato, hanno come base Linux. Infine, le applicazioni trovate e testate sono tutte open-source, per cui l'utente può liberamente accedere al codice sorgente messo a disposizione in genere su piattaforma GitHub ed eventualmente apportare delle modifiche.

Nel Capitolo 2 si procede ad una breve digressione sui sistemi di calcolo parallelo e sui sistemi programmati, al fine di poter utilizzare il software selezionato secondo la modalità di calcolo parallelo. Tale capitolo assume l'accezione più di un glossario che, tuttavia, fornisce il significato e i principi di utilizzo dei sistemi di calcolo adottati.

Nel Capitolo 3 si definiscono le caratteristiche del codice OPENSEES, di un pre- e post-processore programmato in-house in linguaggio Python, secondo una programmazione ad oggetti, noto come OPENSTOOLS, disponibile in rete su piattaforma GitHub [https://www.github.com/gcavpoliba/openstoolsα], e del post-processore OPSTVIS, ricavato da un più vecchio codice presente in rete [https://www.github.com/motiurce/FeView].

Nel Capitolo 4, dopo la scelta dei software e la programmazione di strumenti in grado di preparare l'eseguibile per effettuare il calcolo in parallelo su HPC, si descrive la fase di test e la programmazione di alcuni modelli prototipo, al fine di testare la validità tanto dello strumento programmato quanto la compatibilità con l'infrastruttura di calcolo.

Nel Capitolo 5 è descritto il caso di studio del pendio naturale di Chieuti, cui ci si è ispirati per la modellazione 3D, soffermandosi in particolare sui dati delle prove di laboratorio utilizzati per la calibrazione dei modelli costitutivi selezionati.

Nel Capitolo 6 sono descritti i due modelli costitutivi avanzati scelti per descrivere la risposta dei terreni: il Pressure Independent Multi Yield (PIMY) (Prevost, 1977) e l'Hardening Soil Model with Small Strain (Benz et al., 2009), implementati rispettivamente nei codici OPENSEES e PLAXIS 2D.

Nel Capitolo 7 è illustrata la procedura di calibrazione dei modelli costitutivi; in particolare per il PIMY è stata programmata una applicazione in OPENSEES che

riproduce più prove di taglio semplice ciclico, al fine di definire la curva di decadimento del modulo di rigidezza al taglio e la curva dello smorzamento in funzione del livello di deformazione di taglio imposto.

Nel Capitolo 8 si procede con la simulazione di un semplice caso di propagazione delle onde sismiche all'interno di una colonna di terreno (analisi monodimensionale). Questo caso viene utilizzato per testare la bontà del codice presentato nella prima parte della tesi. L'attenzione è, infatti, rivolta alla impostazione e programmazione delle condizioni al contorno dinamiche e alla scelta del metodo di risoluzione del calcolo.

Nel Capitolo 9 si procede ad analizzare i risultati di un modello 2D eseguito per lo studio della propagazione sismica con riferimento ad una sezione ispirata al caso di studio di Chieuti.

Nel Capitolo 10, infine, si conclude il lavoro con l'analisi della propagazione sismica in un dominio 3D, la cui geometria è ancora ispirata al caso del pendio naturale di Chieuti.

1. SOFTWARE AGLI ELEMENTI FINITI OPEN-SOURCE

In funzione dell'obiettivo preposto dal tema del dottorato di ricerca, si sono analizzati i software più diffusi in rete che siano in grado di risolvere problemi geotecnici al finito mediante la modellazione agli elementi finiti utilizzando più CPU in parallelo.

L'utilizzo di tali software, nella maggior parte dei casi, richiede la conoscenza delle basi della programmazione in TCL o Python.

Code_Aster (https://code-aster.org/) è un software open-source per l'analisi agli elementi finiti sviluppato da Électricité de France (EDF). Nato per rispondere alle esigenze di calcolo strutturale nel settore energetico, è oggi ampiamente utilizzato in ambito ingegneristico per la simulazione numerica di problemi complessi. La sua architettura modulare e la possibilità di personalizzazione lo rendono uno strumento estremamente versatile, capace di affrontare analisi statiche e dinamiche, lineari e non lineari, in ambito termo-meccanico, strutturale e geotecnico.

In particolare, per le analisi geotecniche non lineari in campo dinamico, Code_Aster mette a disposizione una vasta gamma di modelli costitutivi, che permettono di simulare con elevata accuratezza il comportamento dei terreni sotto carichi monotoni e ciclici.

Tra i modelli costitutivi più rilevanti inclusi nella libreria del software, vi sono il Drucker-Prager, il Mohr-Coulomb, il Cam-Clay modificato ed il modello Hujeux; quest'ultimo è particolarmente efficace nella rappresentazione del comportamento ciclico e anisotropo dei terreni. Inoltre, il software include leggi visco-plastiche e dipendenti dal tempo che consentono di modellare fenomeni come la liquefazione, la deformazione accumulata e la dissipazione energetica, integrando effetti di *creep* e rilassamento.

Le capacità di Code_Aster in ambito dinamico si estendono alla definizione di input sismici, all'integrazione temporale implicita ed esplicita, e alla modellazione dell'interazione terreno-struttura, rendendolo uno strumento completo per lo studio di fondazioni, pendii, gallerie e infrastrutture soggette a sollecitazioni dinamiche.

Grazie alla sua natura open-source, alla documentazione tecnica dettagliata e alla possibilità di essere integrato in ambienti grafici come Salome-Meca, Code_Aster

rappresenta una soluzione potente e accessibile per la ricerca e l'applicazione professionale nel campo della geotecnica computazionale.

Nonostante le ottime caratteristiche implementate per la modellazione dei fenomeni geotecnici sismici, non è stato semplice trasferire il software sull'infrastruttura HPC in relazione alla carenza di guide online.

FEniCS (https://fenicsproject.org/) è un software open-source per la risoluzione generica di equazioni alle derivate parziali mediante il metodo degli elementi finiti. Consente di definire problemi matematici in modo compatto e vicino alla formulazione variazionale. La sua compatibilità con Python ne facilita l'integrazione in ambienti scientifici.

Supporta modelli costitutivi elastici lineari e non lineari, visco-elastici e plastici, con possibilità di estensione tramite codice personalizzato. Tra i principali vantaggi si annoverano la flessibilità nella definizione dei problemi, il supporto per mesh non strutturate e la capacità di gestire calcoli distribuiti.

Tuttavia, FEniCS presenta anche diversi limiti: è privo di un'interfaccia grafica, ha una documentazione talvolta frammentaria e richiede una solida preparazione teorica e competenze di programmazione, rendendolo uno strumento potente ma non immediatamente accessibile. Questi aspetti lo rendono particolarmente adatto all'ambito accademico e alla ricerca, ma meno indicato per utenti alle prime armi o per applicazioni industriali standard.

RealESSI (Realistic Modeling and Simulation of Earthquakes, Soils, and Structures and their Interaction - http://sokocalo.engr.ucdavis.edu/~jeremic/ESSI_Simulator/) è un software avanzato per l'analisi agli elementi finiti in ambito geotecnico e strutturale, sviluppato presso l'Università della California a Davis. Esso è progettato per simulazioni tridimensionali in ambito statico e dinamico, sia lineare che non lineare, con particolare attenzione all'interazione terreno-struttura e agli effetti di propagazione sismica.

Tra i principali vantaggi si annoverano:

- una vasta documentazione online fornita in prima persona dal prof. Jeremic,

- la capacità di gestire simulazioni su larga scala;
- l'ottimizzazione per il calcolo parallelo su supercomputer;
- lettura della mesh facilitata da funzioni interne che si interfacciano con il software Gmsh.

Tuttavia, RealESSI presenta anche notevoli difficoltà:

- non dispone di interfaccia grafica;
- non ci sono applicazioni in interfacce grafiche note come GiD;
- l'applicazione per il calcolo parallelo non è disponibile per utenti esterni all'università;
- la comunità di uso e sviluppo è piuttosto scarna in quanto il software non è molto diffuso.

OPENSEES (Open System for Earthquake Engineering Simulation) è un framework open-source sviluppato presso l'Università della California a Berkeley per la simulazione del comportamento di sistemi strutturali e geotecnici soggetti ad azioni sismiche. Scritto principalmente in C++ e integrato con librerie numeriche in Fortran e C, OPENSEES consente la creazione di applicazioni agli elementi finiti sia in modalità seriale che parallela, con un'architettura orientata agli oggetti che ne facilita l'estendibilità e la personalizzazione.

Uno dei principali punti di forza di OPENSEES riguarda principalmente l'ambito strutturale con attenzione anche al problema geotecnico.

La sua vasta libreria di elementi finiti include elementi trave-colonna (sia a plasticità distribuita che concentrata), elementi a fibre, elementi solidi tridimensionali, elementi per l'interazione terreno-struttura, e modelli specifici per fondazioni e smorzatori; inoltre, all'utente è data possibilità di sviluppare codici di programmazione di ogni "oggetto" del framework OPENSEES. Questa varietà consente la modellazione di geometrie complesse e condizioni al contorno articolate, rendendo il software adatto a una vasta gamma di applicazioni ingegneristiche.

Grazie alla stretta interconnessione con l'ambiente della ricerca, il software è aggiornato allo stato dell'arte più recente.

La libreria dei materiali costitutivi di OPENSEES è altrettanto estesa e versatile. I materiali sono suddivisi principalmente in due categorie: *UniaxialMaterial*, per modelli monodimensionali, e *NDMaterial*, per modelli multidimensionali (2D e 3D). Tra i modelli disponibili si trovano quelli per la simulazione del comportamento dell'acciaio e del calcestruzzo, quelli che descrivono la risposta isteretica dei materiali, quelli che descrivono il degrado ciclico, quelli che simulano il comportamento dei materiali geotecnici e quelli per simulazioni termo-meccaniche. Questa ricchezza consente di simulare con precisione il comportamento non lineare, degradante e dissipativo di strutture complesse, rendendo OPENSEES uno strumento ideale per analisi statiche non lineari, analisi pushover e simulazioni dinamiche nel dominio del tempo.

OPENSEES è ampiamente utilizzato per:

- studi di propagazione sismica;
- Performance-Based Seismic Design;
- analisi di interazione terreno-struttura;
- simulazioni di sistemi infrastrutturali complessi.

La sua flessibilità e modularità lo rendono uno strumento consolidato per la ricerca avanzata e per applicazioni ingegneristiche in ambito sismico.

Nonostante la sua potenza, OPENSEES presenta alcune criticità.

La mancanza di un'interfaccia grafica nativa può rappresentare un ostacolo per i nuovi utenti, anche se esistono GUI esterne disponibili online. Inoltre, l'utilizzo del software richiede la scrittura di script in linguaggio TCL o Python, il che implica una certa familiarità con la programmazione.

In conclusione, OPENSEES rappresenta uno strumento potente e flessibile per la simulazione numerica in ingegneria sismica; in particolare, vi è una grande comunità online disponibile a fornire informazioni per problemi sia di modellazione che di funzionamento e configurazione del software.

A conclusione della fase informativa, in ragione della vasta documentazione disponibile online, delle caratteristiche del software e della presenza di supporto tecnico per la programmazione, OPENSEES è il software scelto come strumento per la generazione e modellazione dei modelli complessi di cui si parlerà in seguito. A questo si è

associato l'uso del pre-post processore commerciale STKO e di un pre-post processore "OPENSTOOLS" messo a punto nell'ambito del presente dottorato.

2. SISTEMI DI CALCOLO PARALLELO

Il calcolo parallelo è ormai comunemente utilizzato in tutte le applicazioni su personal computer e smartphone commerciali. I primi, in genere, sono dotati di unità computazionali pari ad un numero che va da 4 in su, in funzione del costo che l'utente è disposto a sostenere. Ovviamente, anche i software più comunemente utilizzati sono programmati per svolgere più operazioni allo stesso istante, per rispondere alla necessità di un uso veloce e di una risposta pronta alla domanda di utilizzo espressa dall'utente.

Lungi dal fornire una trattazione esaustiva circa i moderni sistemi di calcolo parallelo, in questo capitolo si vuole giusto stilare un elenco di termini e significati in modo da trasferire, in una sorta di glossario, una breve descrizione del lessico inerente il calcolo parallelo, oltre che una mappa concettuale circa le librerie comunemente utilizzate e la loro logica di utilizzo.

Tutto questo trova applicazione nella risoluzione dei sistemi di equazioni lineari dei modelli agli elementi finiti, influendo pesantemente sulla performance del calcolo in termini di tempo impiegato per la risoluzione, il quale non segue una legge lineare funzione del numero dei processori impiegati (https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial).

L'architettura fondamentale della maggior parte dei sistemi di calcolo moderni è stata concepita dal matematico ungherese John von Neumann negli anni '40. Questa architettura, nota anche come "stored-program computer", rappresenta un cambiamento rivoluzionario rispetto ai sistemi precedenti, poiché consente di memorizzare sia le istruzioni del programma che i dati nella stessa memoria elettronica, superando il concetto di programmazione attraverso "hard wiring" tipico delle macchine precedenti.

L'architettura von Neumann si articola in quattro componenti fondamentali che costituiscono la base di ogni sistema di elaborazione:

Memoria: unità destinata al deposito di informazioni, costituisce il repository centrale per programmi e dati, implementata attraverso tecnologie ad accesso casuale (RAM) che permettono operazioni di lettura e scrittura. La memoria non distingue

intrinsecamente tra istruzioni e dati, trattando entrambi come sequenze di informazioni codificate.

Unità di Controllo: rappresenta il coordinatore del sistema, responsabile del recupero sequenziale delle istruzioni dalla memoria, della loro decodifica e dell'orchestrazione delle operazioni necessarie per l'esecuzione del programma. Il carattere sequenziale di questa unità costituisce uno degli elementi distintivi dell'architettura von Neumann.

Unità Aritmetico-Logica (ALU): si occupa dell'esecuzione delle operazioni matematiche e logiche fondamentali, costituendo il cuore computazionale del sistema.

Sistema di Input/Output: fornisce l'interfaccia con l'ambiente esterno, consentendo l'interazione con operatori umani e altri sistemi.

È importante notare che anche i sistemi di calcolo parallelo più sofisticati mantengono questa struttura architettonica di base, semplicemente moltiplicando e interconnettendo questi componenti fondamentali. La robustezza e l'universalità di questo design hanno garantito la sua persistenza attraverso decenni di evoluzione tecnologica.

Per classificare i diversi approcci al calcolo parallelo, la comunità scientifica fa riferimento alla tassonomia proposta da Michael Flynn nel 1966, che rimane tuttora uno degli schemi classificativi più utilizzati e riconosciuti nel campo del calcolo parallelo.

La **tassonomia di Flynn** categorizza le architetture multi-processore secondo due dimensioni indipendenti: il flusso di istruzioni (Instruction Stream) e il flusso di dati (Data Stream). Ciascuna di queste dimensioni può assumere due stati distinti: Singolo (Single) o Multiplo (Multiple), generando una matrice 2x2 che definisce quattro categorie fondamentali.

La categoria SISD (**Single Instruction, Single Data**) rappresenta l'architettura seriale tradizionale, dove un singolo flusso di istruzioni opera su un singolo flusso di dati. Durante ogni ciclo di clock, la CPU elabora esclusivamente una istruzione operando su un singolo elemento dati. Questa configurazione garantisce un'esecuzione deterministica e rappresenta il paradigma più antico nell'evoluzione dei sistemi di calcolo. Esempi tipici includono i mainframe di generazioni precedenti, i minicomputer, le workstation tradizionali e i personal computer mono-processore.

L'architettura SIMD (**Single Instruction, Multiple Data)** introduce il primo livello di parallelismo, mantenendo un singolo flusso di istruzioni, ma operando simultaneamente su elementi multipli di dati. Tutte le unità di elaborazione eseguono la medesima istruzione in ogni ciclo di clock, ma ciascuna opera su porzioni differenti del dataset. Questa configurazione si rivela particolarmente efficace per problemi caratterizzati da alta regolarità, come l'elaborazione di immagini e grafica.

L'esecuzione in modalità SIMD è intrinsecamente sincrona e deterministica, manifestandosi principalmente in due varianti: gli Array di Processori e le Pipeline Vettoriali. La maggior parte dei sistemi moderni, specialmente quelli dotati di unità di elaborazione grafica (GPU), implementa istruzioni e unità di esecuzione SIMD.

La categoria MISD (**Multiple Instruction, Single Data)** rappresenta una configurazione teorica, dove multipli flussi di istruzioni operano indipendentemente su un singolo flusso di dati. Nella pratica, pochissimi esempi concreti di questa architettura sono mai stati implementati, rendendo MISD più un caso di studio teorico che una soluzione pratica. Possibili applicazioni potrebbero includere sistemi dove multipli filtri di frequenza operano simultaneamente su un singolo segnale o dove differenti algoritmi crittografici tentano di decifrare contemporaneamente lo stesso messaggio codificato.

L'architettura MIMD (**Multiple Instruction, Multiple Data**) costituisce attualmente la forma più diffusa di calcolo parallelo, caratterizzata da multipli processori che eseguono indipendentemente diversi flussi di istruzioni operando su dataset distinti. Questa configurazione offre la massima flessibilità, permettendo un'esecuzione che può essere sia sincrona che asincrona, deterministica o non-deterministica.

La prevalenza di MIMD nel panorama computazionale contemporaneo è testimoniata dalla sua adozione nella maggior parte dei supercomputer moderni, nei cluster di calcolo parallelo distribuiti, nei sistemi multi-processore SMP e nei personal computer multi-core. È interessante notare che molte architetture MIMD incorporano anche componenti di esecuzione SIMD, creando sistemi ibridi che massimizzano l'efficienza computazionale.

Procedendo nella tassonomia, nella categoria dei componenti hardware abbiamo:

CPU (Central Processing Unit): nelle implementazioni contemporanee, una CPU è costituita da uno o più core, ciascuno rappresentante un'unità di esecuzione distinta con il proprio flusso di istruzioni. I core all'interno di una CPU possono essere organizzati in uno o più socket, ognuno con memoria distinta, e quando una CPU comprende multipli socket, l'infrastruttura hardware tipicamente supporta la condivisione di memoria tra socket.

Nodo: rappresenta un "computer completo" autonomo, solitamente comprendente multiple CPU/processori/core, memoria, interfacce di rete e altri componenti. I nodi vengono interconnessi attraverso reti per costituire supercomputer.

Task: costituisce una sezione logicamente discreta di lavoro computazionale, tipicamente rappresentata da un programma o un insieme di istruzioni simili a un programma che viene eseguito da un processore. Un programma parallelo è costituito da task multipli, che operano simultaneamente su differenti processori.

La terminologia riguardante l'elaborazione invece comprende:

Pipelining: tecnica che suddivide un'attività in passi sequenziali eseguiti da differenti CPU, con input che fluiscono attraverso il sistema in maniera simile a una catena di montaggio. Rappresenta una forma specifica di elaborazione parallela.

Memoria Condivisa (Shared Memory): descrive sia un'architettura hardware dove tutti i processori hanno accesso diretto alla memoria fisica comune, sia un modello di programmazione dove tutti i task paralleli condividono la stessa "visione" della memoria e possono accedere direttamente alle stesse locazioni logiche indipendentemente dalla posizione fisica della memoria.

Multi-processor Simmetrico (SMP): architettura hardware di memoria condivisa, dove multipli processori condividono un singolo spazio di indirizzi e hanno accesso equo a tutte le risorse, incluse memoria e dispositivi di storage.

Memoria Distribuita: nel contesto hardware, si riferisce all'accesso alla memoria basato su rete per memoria fisica non comune. Come modello di programmazione, implica che i task possono "vedere" logicamente solo la memoria della macchina locale e devono utilizzare comunicazioni per accedere alla memoria su altre macchine dove operano altri task.

Nella macrocategoria della comunicazione e sincronizzazione:

Comunicazione: i task paralleli necessitano tipicamente di scambiare dati, operazione che può essere realizzata attraverso varie metodologie, come bus di memoria condivisa o comunicazione di rete.

Sincronizzazione: coordinamento dei task paralleli in tempo reale, spesso associato alle comunicazioni. La sincronizzazione comporta tipicamente l'attesa da parte di almeno un task, potendo quindi incrementare il tempo di esecuzione totale di un'applicazione parallela.

Granularità Computazionale: misura quantitativa o qualitativa del rapporto tra computazione e comunicazione. Si distingue tra *granularità grossolana* (coarse), dove quantità relativamente grandi di lavoro computazionale vengono eseguite tra eventi di comunicazione, e *granularità fine* (fine), dove quantità relativamente piccole di lavoro computazionale sono inframmezzate da frequenti comunicazioni.

Metriche di Performance:

Speedup Osservato: definito come il rapporto tra il tempo di esecuzione seriale e il tempo di esecuzione parallela, rappresenta uno degli indicatori più semplici e utilizzati per valutare le prestazioni di un programma parallelo.

Overhead Parallelo: tempo di esecuzione richiesto specificamente per le operazioni parallele, in contrapposizione a quello dedicato al lavoro utile. Include fattori come tempo di avvio dei task, sincronizzazioni, comunicazioni dati, overhead software imposto da linguaggi paralleli, librerie e sistemi operativi, e tempo di terminazione dei task.

Scalabilità: capacità di un sistema parallelo (hardware e/o software) di dimostrare un aumento proporzionale dello speedup parallelo con l'aggiunta di più risorse. I fattori che contribuiscono alla scalabilità includono caratteristiche hardware (particolarmente bandwidths memoria-CPU e proprietà di comunicazione di rete), algoritmi applicativi, overhead relativo al parallelismo e caratteristiche specifiche dell'applicazione.

Dopo aver affrontato le strutture del calcolo parallelo passiamo a categorizzare **le performance**.

Uno dei principi fondamentali che governa le prestazioni del calcolo parallelo è espresso dalla **Legge di Amdahl**, che definisce il potenziale speedup di un programma in base alla frazione di codice che può essere parallelizzata. Formalmente, lo **speedup** è definito dalla relazione (1):

$$\frac{1}{1-P} \quad (1)$$

dove **P** rappresenta la **frazione parallelizzabile** del codice.

Questa legge rivela immediatamente i limiti intrinseci della parallelizzazione: se nessuna porzione del codice può essere parallelizzata ($P = 0$), lo speedup è 1 (nessun miglioramento); se tutto il codice è parallelizzabile ($P = 1$), lo speedup teorico è infinito; se il 50% del codice può essere parallelizzato, lo speedup massimo è limitato a 2.

Introducendo il numero di processori N che eseguono la frazione parallela del lavoro, la relazione diventa (2):

$$\frac{1}{S+P/N} \quad (2)$$

dove **S** rappresenta la **frazione seriale** del codice.

Questa formulazione evidenzia come i limiti alla scalabilità del parallelismo diventino rapidamente evidenti. I dati empirici dimostrano che anche con frazioni parallele elevate, l'aggiunta di processori produce rendimenti decrescenti.

La valutazione delle prestazioni parallele si articola tradizionalmente in due paradigmi di scaling:

Strong Scaling (Amdahl): mantiene costante la dimensione totale del problema mentre si aumenta il numero di processori. L'obiettivo è risolvere lo stesso problema in minor tempo, con scaling perfetto che implica una riduzione del tempo di esecuzione proporzionale a 1/P rispetto alla versione seriale.

Weak Scaling (Gustafson): mantiene costante la dimensione del problema per processore, mentre si aggiungono processori, risultando in una dimensione totale del problema proporzionale al numero di processori utilizzati. L'obiettivo è risolvere problemi più grandi nello stesso tempo, con scaling perfetto che implica che un problema P volte più grande richieda lo stesso tempo dell'esecuzione mono-processore.

La programmazione parallela introduce intrinsecamente livelli di complessità superiori rispetto alle controparti seriali. Questa complessità non deriva solamente dalla

gestione di multipli flussi di istruzioni simultanei, ma anche dalla necessità di coordinare il flusso di dati tra di essi. I costi di questa complessità si manifestano in ogni fase del ciclo di sviluppo software: progettazione, codifica, debugging, tuning e manutenzione.

Portabilità: nonostante la standardizzazione in varie API (Application Programming Interface), come MPI, OpenMP e POSIX threads, abbia significativamente ridotto i problemi di portabilità rispetto al passato, persistono sfide. Implementazioni diverse degli stessi standard possono variare in dettagli significativi, richiedendo talvolta modifiche al codice per garantire la portabilità. Sistemi operativi e architetture hardware mantengono caratteristiche altamente variabili che possono influenzare la portabilità.

Requisiti di Risorse: l'intento primario della programmazione parallela è ridurre il tempo di esecuzione wall-clock, ma questo richiede maggiori risorse CPU totali. Un codice parallelo che opera per 1 ora su 8 processori utilizza effettivamente 8 ore di tempo CPU. La quantità di memoria richiesta può essere superiore per i codici paralleli rispetto a quelli seriali, a causa della necessità di replicare dati e per gli overhead associati a librerie e sottosistemi di supporto parallelo.

Le **architetture a memoria condivisa**, pur presentando un'ampia varietà implementativa, condividono la caratteristica fondamentale di consentire a tutti i processori l'accesso a tutta la memoria come spazio di indirizzamento globale. Multipli processori possono operare indipendentemente condividendo le stesse risorse di memoria, con modifiche effettuate da un processore che risultano immediatamente visibili a tutti gli altri.

Storicamente, questi sistemi sono stati classificati in base ai tempi di accesso alla memoria, distinguendo tra architetture **UMA** (Uniform Memory Access) e **NUMA** (Non-Uniform Memory Access).

Uniform Memory Access (UMA): rappresentata più comunemente dalle macchine Symmetric Multi-Processor (SMP) è caratterizzata da processori identici con accesso equo e tempi di accesso uniformi alla memoria. Essa è spesso denominata CC-UMA (Cache Coherent UMA), dove la coerenza della cache (“Cache coherent”) significa che, quando un processore aggiorna una locazione di memoria condivisa, tutti gli

altri processori sono informati dell'aggiornamento, con la coerenza garantita ("Cache coherency") a livello hardware.

Non-Uniform Memory Access (NUMA): spesso implementata collegando fisicamente due o più SMP (Symmetric Multi-Processor), dove un SMP può accedere direttamente alla memoria di un altro SMP, ma non tutti i processori hanno tempi di accesso equi a tutte le memorie. L'accesso alla memoria attraverso collegamenti inter-SMP risulta più lento. Quando la coerenza della cache ("Cache coherency") è mantenuta, può essere denominata CC-NUMA.

I vantaggi delle architetture a memoria condivisa includono uno spazio di indirizzamento globale che fornisce una prospettiva di programmazione user-friendly, e condivisione di dati tra task veloce e uniforme grazie alla prossimità della memoria alle CPU.

Gli svantaggi principali comprendono la mancanza di scalabilità tra memoria e CPU, dove l'aggiunta di più CPU può aumentare geometricamente il traffico sul percorso memoria-CPU condivisa, e per sistemi cache-coherent, può aumentare geometricamente il traffico associato alla gestione cache/memoria. Inoltre, la responsabilità della sincronizzazione ricade sul programmatore per garantire l'accesso "corretto" alla memoria globale.

I sistemi a **memoria distribuita**, pur presentando un'ampia variabilità, condividono la caratteristica di richiedere una rete di comunicazione per connettere la memoria inter-processore. Ogni processore dispone della propria memoria locale, con indirizzi di memoria che non mappano ad altri processori, eliminando il concetto di spazio di indirizzamento globale.

Poiché ogni processore opera indipendentemente con la propria memoria locale, le modifiche alla memoria locale non influenzano la memoria di altri processori, rendendo non applicabile il concetto di coerenza della cache. Quando un processore necessita di accedere a dati residenti su un altro processore, è tipicamente responsabilità del programmatore definire esplicitamente come e quando i dati vengono comunicati.

I vantaggi includono scalabilità della memoria proporzionale al numero di processori, accesso rapido di ogni processore alla propria memoria senza interferenze e senza l'overhead di mantenimento della coerenza globale della cache, e cost-effectiveness attraverso l'uso di processori commodity off-the-shelf e networking.

Gli svantaggi comprendono la responsabilità del programmatore per molti dettagli associati alla comunicazione dati tra processori, difficoltà nel mappare strutture dati esistenti basate su memoria globale a questa organizzazione di memoria, e tempi di accesso non uniformi alla memoria con dati residenti su nodi remoti che richiedono più tempo di accesso rispetto ai dati locali.

I computer più grandi e veloci del mondo impiegano attualmente architetture che combinano memoria condivisa e distribuita.

La componente di memoria condivisa può essere rappresentata da macchine a memoria condivisa e/o **unità di elaborazione grafica** (**GPU**), mentre la componente di memoria distribuita è costituita dal networking di più macchine a memoria condivisa/GPU, che conoscono solamente la propria memoria locale.

Questa configurazione ibrida offre vantaggi che includono tutto ciò che è comune alle architetture sia di memoria condivisa che distribuita, con particolare enfasi sulla scalabilità aumentata. Lo svantaggio principale è rappresentato dalla complessità accresciuta per il programmatore.

I **modelli di programmazione parallela** esistono come un'astrazione al di sopra delle architetture hardware e di memoria. È importante riconoscere che questi modelli non sono specifici per un particolare tipo di macchina o architettura di memoria; teoricamente, qualsiasi di questi modelli può essere implementato su qualsiasi hardware sottostante.

Nel **modello a memoria condivisa**, processi/task condividono uno spazio di indirizzi comune, che leggono e scrivono asincronamente. Vari meccanismi come **lock** e **semafori** controllano l'accesso alla memoria condivisa, risolvono contese e prevengono **race condition** e **deadlock.**

Questo rappresenta forse il modello di programmazione parallela più semplice. Dal punto di vista del programmatore, un vantaggio significativo è l'assenza del

concetto di **"ownership" dei dati**, eliminando la necessità di specificare esplicitamente la comunicazione di dati tra task. Tutti i processi vedono e hanno accesso equo alla memoria condivisa, spesso semplificando lo sviluppo del programma.

Uno svantaggio importante in termini di prestazioni è che diventa più difficile comprendere e gestire la località dei dati: mantenere i dati locali al processo che li elabora conserva accessi alla memoria, refresh della cache e traffico bus che si verifica quando più processi utilizzano gli stessi dati.

Il **modello a thread** rappresenta una tipologia di programmazione a memoria condivisa, dove un singolo processo "heavy weight" può avere multipli percorsi di esecuzione concorrenti "light weight". Il programma principale viene pianificato dal sistema operativo nativo, acquisendo tutte le risorse necessarie (processo "heavy weight"), quindi crea più task (thread) che possono essere pianificate ed eseguite nello stesso istante dal sistema operativo.

Ogni thread possiede dati locali, ma condivide anche tutte le risorse del programma principale, risparmiando l'overhead associato alla replicazione delle risorse per ogni thread. I thread comunicano attraverso la memoria globale, richiedendo costrutti di sincronizzazione per garantire che thread multipli non aggiornino simultaneamente lo stesso indirizzo globale.

Le implementazioni principali includono POSIX Threads (Pthreads), specificati dallo standard IEEE POSIX 1003.1c del 1995, che rappresentano una soluzione basata su libreria per linguaggio C con parallelismo molto esplicito, e OpenMP, uno standard industriale basato su direttive del compilatore, multi-piattaforma e disponibile per implementazioni C/C++ e Fortran.

Il **Modello a Memoria Distribuita / Message Passing** è caratterizzato da un insieme di task che utilizzano la propria memoria locale durante la computazione, con task multipli che possono risiedere sulla stessa macchina fisica e/o attraverso un numero arbitrario di macchine. I task scambiano dati attraverso comunicazioni inviando e ricevendo messaggi, con trasferimenti dati che tipicamente richiedono operazioni cooperative tra processi.

L'implementazione di riferimento è rappresentata dal **Message Passing Interface** (MPI), sviluppato dal MPI Forum formatosi nel 1992 con l'obiettivo primario di stabilire un'interfaccia standard per implementazioni message passing. MPI-1 è stato rilasciato nel 1994, seguito da MPI-2 nel 1996 e MPI-3 nel 2012, diventando lo standard industriale de facto per il message passing.

Noto anche come modello Partitioned Global Address Space (PGAS), il **modello data parallel** tratta lo spazio degli indirizzi globalmente, con la maggior parte del lavoro parallelo focalizzato sull'esecuzione di operazioni su un dataset organizzato in strutture comuni come array o cubi. Un insieme di task lavora collettivamente sulla stessa struttura dati, con ogni task operante su una partizione differente.

Implementazioni attuali includono Coarray Fortran, Unified Parallel C (UPC), Global Arrays, X10 e Chapel, ciascuna con caratteristiche specifiche per differenti paradigmi di programmazione.

I modelli ibridi combinano due o più dei modelli precedentemente descritti. Un esempio comune è la combinazione del modello message passing (MPI) con il modello a thread (OpenMP), dove thread eseguono kernel computazionalmente intensivi utilizzando dati locali on-node, mentre le comunicazioni tra processi su nodi differenti avvengono attraverso la rete utilizzando MPI.

Un altro esempio crescente in popolarità è l'uso di MPI con programmazione CPU-GPU, dove task MPI operano su CPU utilizzando memoria locale e comunicando attraverso rete, mentre kernel computazionalmente intensivi vengono trasferiti alle GPU on-node.

Single Program Multiple Data (SPMD): modello di programmazione di alto livello costruibile su qualsiasi combinazione dei modelli paralleli precedenti. Tutti i task eseguono la loro copia dello stesso programma simultaneamente, ma possono utilizzare dati differenti. I programmi SPMD contengono la logica necessaria per permettere a differenti task di eseguire condizionalmente solo quelle parti del programma per cui sono progettati.

Multiple Program Multiple Data (MPMD): simile a SPMD come modello *di alto livello*, però in questo caso i task possono eseguire programmi differenti

simultaneamente utilizzando dati differenti. Le applicazioni MPMD sono meno comuni di quelle SPMD, ma possono essere più adatte per certi tipi di problemi, in particolare quelli che si prestano meglio alla decomposizione funzionale piuttosto che alla decomposizione di dominio.

La progettazione e lo sviluppo di programmi paralleli è stata caratteristicamente un processo molto manuale, con il programmatore tipicamente responsabile sia dell'identificazione che dell'implementazione effettiva del parallelismo. Spesso, lo sviluppo manuale di codici paralleli è un processo time-consuming, complesso, soggetto a errori e iterativo.

I compilatori parallelizzanti operano generalmente in due modalità:

Completamente Automatica: il compilatore analizza il codice sorgente e identifica opportunità di parallelismo, includendo l'identificazione di inibitori al parallelismo e possibilmente una valutazione costo-beneficio sulla effettiva utilità del parallelismo per migliorare le prestazioni.

Diretta dal Programmatore: utilizzando "direttive del compilatore" o **flag** del compilatore, il programmatore dice esplicitamente al compilatore come parallelizzare il codice, possibilmente in congiunzione con un certo grado di parallelizzazione automatica.

Prima di dedicare tempo allo sviluppo di una soluzione parallela, è fondamentale determinare se il problema possa effettivamente essere parallelizzato. Alcuni problemi si prestano naturalmente alla parallelizzazione, mentre altri presentano dipendenze sequenziali intrinseche che limitano o impediscono la parallelizzazione efficace.

L'identificazione degli hotspot del programma è cruciale: la maggior parte dei programmi scientifici e tecnici concentra la maggior parte del lavoro computazionale in poche sezioni specifiche. Strumenti di profiling e analisi delle prestazioni possono assistere in questa identificazione, permettendo di focalizzare gli sforzi di parallelizzazione sulle sezioni critiche ignorando quelle che contribuiscono minimamente all'uso della CPU.

Uno dei primi passi nella progettazione di un programma parallelo è la suddivisione del problema in "chunk" discreti di lavoro distribuibili a task multipli

(**partizionamento**). Esistono due approcci fondamentali: decomposizione di dominio e decomposizione funzionale.

Decomposizione di Dominio: i dati associati al problema vengono decomposti, con ogni task parallelo che opera su una porzione dei dati. Questo approccio può essere implementato attraverso strategie di partizionamento, incluso partizionamento per blocchi, ciclico, o block-ciclico.

Decomposizione Funzionale: il focus è sulla computazione da eseguire piuttosto che sui dati manipolati. Il problema viene decomposto secondo il lavoro che deve essere fatto, con ogni task che esegue una porzione del lavoro complessivo. Questo approccio si adatta bene a problemi divisibili in task differenti.

La necessità di **comunicazioni** tra task dipende dalla natura del problema. Alcuni problemi possono essere decomposti ed eseguiti in parallelo senza nessuna necessità di condivisione dati tra task ("embarrassingly parallel"), mentre la maggioranza delle applicazioni parallele richiede condivisione di dati tra task.

Fattori importanti da considerare includono l'overhead di comunicazione, che implica cicli macchina e risorse utilizzate per impacchettare e trasmettere dati invece che per computazione; la **latenza** vs **larghezza di banda**, dove latenza rappresenta il tempo per inviare un messaggio minimo dal punto A al punto B, mentre la larghezza di banda è la quantità di dati comunicabile per unità di tempo.

La visibilità delle comunicazioni varia tra modelli: nel Message Passing Model le comunicazioni sono esplicite e visibili, mentre nel Data Parallel Model possono essere trasparenti al programmatore. Le comunicazioni possono essere sincrone (blocking) o asincrone (non-blocking), con quest'ultime che permettono l'interleaving di computazione e comunicazione.

La gestione della sequenza di lavoro e dei task che lo eseguono rappresenta una considerazione critica di design per la maggior parte dei programmi paralleli. La **sincronizzazione** può essere un fattore significativo nelle prestazioni del programma e spesso richiede la serializzazione di segmenti del programma.

I tipi principali di sincronizzazione includono:

Barrier: tipicamente coinvolge tutti i task, dove ogni task esegue il proprio lavoro fino a raggiungere la barriera, quindi si arresta. Quando l'ultimo task raggiunge la barriera, tutti i task sono sincronizzati.

Lock/Semaphore: può coinvolgere qualsiasi numero di task, tipicamente utilizzato per serializzare l'accesso a dati globali o sezioni di codice. Solo un task alla volta può possedere il lock.

Operazioni di Comunicazione Sincrona: coinvolge solo task che eseguono operazioni di comunicazione, richiedendo coordinamento con altri task partecipanti alla comunicazione.

Una **dipendenza** esiste tra istruzioni del programma quando l'ordine di esecuzione delle istruzioni influenza i risultati del programma. Una **dipendenza dei dati** risulta dall'uso multiplo delle stesse locazioni di storage da parte di task differenti. Le dipendenze rappresentano uno dei principali inibitori al parallelismo.

Esempi includono dipendenze **loop-carried**, dove il valore di A(J) deve essere calcolato dopo A(J-1), inibendo il parallelismo. In architetture a memoria distribuita, il task 2 deve ottenere il valore di A(J-1) dal task 1 dopo che quest'ultimo completa la sua computazione. In architetture a memoria condivisa, il task 2 deve leggere A(J-1) dopo che il task 1 lo aggiorna.

La gestione delle dipendenze dei dati richiede strategie specifiche: in architetture a memoria distribuita attraverso la comunicazione dei dati richiesti nei punti di sincronizzazione, mentre in architetture a memoria condivisa attraverso la sincronizzazione delle operazioni di lettura/scrittura tra task.

Il **bilanciamento del carico** si riferisce alla pratica di distribuire quantità approssimativamente uguali di lavoro tra i task, mantenendo tutti i task occupati costantemente. Questo aspetto è cruciale per le prestazioni dei programmi paralleli, poiché se tutti i task sono soggetti a un punto di sincronizzazione barrier, il task più lento determinerà le prestazioni complessive.

Per ottenere un bilanciamento efficace del carico, è possibile partizionare equamente il lavoro assegnato a ogni task, particolarmente efficace per operazioni su array/matrici dove ogni task esegue lavoro simile. Per iterazioni di loop dove il lavoro in

ogni iterazione è simile, distribuire equamente le iterazioni attraverso i task risulta vantaggioso.

Quando la quantità di lavoro per task è intenzionalmente variabile o imprevedibile, può essere utile utilizzare un approccio scheduler-task pool, dove ogni task riceve nuovo lavoro dalla coda man mano che completa le assegnazioni precedenti. Esistono codici che prevedono questa particolarità come le **librerie METIS** per la partizione della mesh e il bilanciamento del carico.

La **granularità nel calcolo parallelo** rappresenta una misura qualitativa del rapporto computazione/comunicazione. I periodi di computazione sono tipicamente separati da periodi di comunicazione attraverso eventi di sincronizzazione.

Parallelismo Fine-Grain: caratterizzato da quantità relativamente piccole di lavoro computazionale tra eventi di comunicazione, basso rapporto computazione/comunicazione, facilita il bilanciamento del carico ma implica alto overhead di comunicazione e minori opportunità di miglioramento delle prestazioni.

Parallelismo Coarse-Grain: caratterizzato da quantità relativamente grandi di lavoro computazionale tra eventi di comunicazione/sincronizzazione, alto rapporto computazione/comunicazione, implica maggiori opportunità di aumento delle prestazioni ma è più difficile da bilanciare efficacemente.

La granularità più efficiente dipende dall'algoritmo e dall'ambiente hardware in cui opera. Nella maggior parte dei casi, l'overhead associato a comunicazioni e sincronizzazione è alto relativamente alla velocità di esecuzione, rendendo vantaggiosa la granularità coarse.

Le **operazioni I/O** rappresentano generalmente inibitori al parallelismo, richiedendo ordini di grandezza in più di tempo rispetto alle operazioni di memoria. I sistemi I/O paralleli possono essere immaturi o non disponibili per tutte le piattaforme. In ambienti dove tutti i task condividono lo stesso path, le operazioni di scrittura possono risultare in sovrascrittura di file, mentre le operazioni di lettura possono essere influenzate dalla capacità del file server di gestire più richieste di lettura simultanee.

Tuttavia, esistono sviluppi positivi: file system paralleli come GPFS (IBM Spectrum Scale), Lustre (Intel), HDFS (Apache Hadoop), e PanFS (Panasas) sono

disponibili. L'interfaccia di programmazione I/O parallela per MPI è disponibile dal 1996 come parte di MPI-2, con implementazioni vendor e "free" ora disponibili.

Strategie raccomandabili includono ridurre l'I/O complessivo quanto possibile, utilizzare file system paralleli quando disponibili, scrivere chunk grandi di dati piuttosto che piccoli, utilizzare file meno numerosi e più grandi, confinare l'I/O a porzioni seriali specifiche del job utilizzando comunicazioni parallele per distribuire dati.

Il **debugging** di codici paralleli può essere incredibilmente difficile, particolarmente man mano che i codici scalano verso l'alto. Fortunatamente, esistono debugger eccellenti per assistere, inclusi strumenti per threaded (pthreads e OpenMP), MPI, GPU/accelerator, e applicazioni ibride.

Strumenti disponibili includono TotalView (RogueWave Software), DDT (Allinea), Inspector (Intel), e Stack Trace Analysis Tool (STAT). Tutti questi strumenti hanno una curva di apprendimento associata ma rappresentano risorse inestimabili per lo sviluppo di applicazioni parallele robuste.

L'analisi e il tuning delle prestazioni di programmi paralleli presentano sfide superiori rispetto ai programmi seriali. Strumenti specializzati includono TAU, HPCToolkit, Open|Speedshop, Vampir/Vampirtrace, Valgrind, PAPI, mpiP, e memP, ciascuno ottimizzato per aspetti specifici dell'analisi delle prestazioni parallele.

Il calcolo parallelo rappresenta un paradigma fondamentale per affrontare le crescenti esigenze computazionali della ricerca scientifica e tecnologica contemporanea. La comprensione approfondita dei principi teorici, delle architetture hardware, dei modelli di programmazione e delle tecniche implementative è essenziale per sviluppare soluzioni efficaci e scalabili.

L'evoluzione continua delle architetture hardware, dall'emergere di sistemi many-core all'integrazione di acceleratori specializzati come GPU, richiede un approccio adattivo e flessibile alla programmazione parallela. La padronanza di più paradigmi di programmazione, dalla memoria condivisa al message passing, dai modelli data parallel agli approcci ibridi, diventa quindi cruciale per il successo nell'implementazione di applicazioni parallele performanti.

Le sfide intrinseche del calcolo parallelo - dalla gestione delle dipendenze dei dati al bilanciamento del carico, dalla minimizzazione dell'overhead di comunicazione all'ottimizzazione della granularità computazionale - richiedono una comprensione profonda tanto degli aspetti teorici quanto di quelli pratici.

L'applicazione efficace di tecniche di calcolo parallelo dipende dalla capacità di analizzare criticamente i problemi, identificare opportunità di parallelizzazione, e implementare soluzioni che massimizzino l'efficienza computazionale rispettando i vincoli delle architetture hardware sottostanti.

La tendenza verso architetture sempre più complesse ed eterogenee, combinando CPU multi-core, acceleratori specializzati e sistemi distribuiti, evidenzia l'importanza di approcci ibridi che sfruttino le caratteristiche ottimali di ogni componente. La progettazione di applicazioni parallele future richiederà quindi non solo competenze tecniche approfondite, ma anche capacità di anticipare e adattarsi alle evoluzioni tecnologiche emergenti.

L'investimento nella comprensione e padronanza delle tecniche di calcolo parallelo rappresenta quindi non solo una necessità immediata per affrontare problemi intensivi dal punto di vista computazionale, ma anche una preparazione essenziale per partecipare attivamente all'evoluzione continua del panorama del calcolo ad alte prestazioni.

3. OPENSEES, OPENSTOOLS E OPSTVIS

3.1 DESCRIZIONE DEL SOFTWARE OPENSEES

OPENSEES (Open System for Earthquake Engineering Simulation) rappresenta uno dei framework più avanzati e completi per l'analisi strutturale non lineare. Il software, sviluppato presso l'Università della California a Berkeley, è interamente basato sulla programmazione ad oggetti in linguaggio C++ e fornisce un ambiente di calcolo estremamente flessibile per modellare sistemi strutturali complessi sottoposti a carichi dinamici. La filosofia progettuale di OPENSEES si fonda inoltre sui metodi di analisi e i sistemi di monitoraggio della risposta che consentono agli utenti di combinare diversi algoritmi numerici e modelli costitutivi per affrontare problematiche ingegneristiche di elevata complessità.

Il cuore computazionale di OPENSEES risiede nella sua capacità di operare efficacemente sia in modalità seriale che parallela, adattandosi alle diverse esigenze computazionali. In ambiente seriale, il software è efficacemente dedicato ad analisi di modelli di dimensioni moderate. L'eseguibile per il coding del modello di calcolo è compilato attraverso un interprete TCL/Tk. Tuttavia, la vera forza del framework emerge nell'ambiente parallelo, dove sono disponibili due "interpreter" distinti: OPENSEESSP (Single Parallel Interpreter) per la decomposizione automatica del dominio di modelli molto grandi, e OPENSEESMP (Multiple Parallel Interpreters) per analisi parametriche massive e controllo granulare della parallelizzazione.

L'implementazione parallela di OPENSEES sfrutta l'interfaccia MPI (Message Passing Interface) per garantire portabilità e scalabilità su diverse architetture di calcolo ad alte prestazioni (HPC). La strategia di parallelizzazione adottata nell'interpreter OPENSEESSP si basa su una distribuzione efficiente del carico computazionale attraverso tecniche di decomposizione del dominio, dove il processore master (P0) gestisce l'interpretazione dei comandi e la costruzione del modello, mentre i processori slave eseguono istanze di ActorSubdomain per la risoluzione distribuita del sistema di equazioni. Questa architettura consente una scalabilità ottimale su sistemi che vanno dai cluster locali con decine di processori fino ai supercomputer con un numero superiore

di unità di calcolo, mantenendo un'efficienza computazionale elevata anche attraverso l'ottimizzazione delle comunicazioni inter-processore.

La filosofia open-source di OPENSEES rappresenta un aspetto fondamentale del progetto, dove tutti gli utenti sono potenzialmente sviluppatori e l'architettura software estensibile consente l'integrazione di nuove formulazioni di elementi e modelli costitutivi.

Il codice sorgente completo di OPENSEES è liberamente accessibile attraverso il repository su piattaforma GitHub, permettendo una trasparenza totale degli algoritmi implementati e facilitando la verifica, la validazione e l'estensione delle funzionalità esistenti. Il sistema supporta tre categorie principali di materiali: materiali uniassiali per relazioni costitutive monodimensionali, materiali n-dimensionali per formulazioni in stato piano o tridimensionali, e sezioni che definiscono comportamenti accoppiati. Gli sviluppatori possono implementare nuove classi ereditate dalle classi base del framework, beneficiando automaticamente di tutte le funzionalità di integrazione, comunicazione parallela e gestione della memoria già implementate nel sistema.

La scalabilità del sistema è ulteriormente potenziata dall'integrazione di solutori paralleli avanzati, tra cui **SuperLU, PETSc e MUMPS**, che rappresentano lo stato dell'arte nella risoluzione di sistemi lineari sparsi di grandi dimensioni.

L'ambiente OPENSEES gestisce automaticamente la selezione del solutore più appropriato basandosi sulle caratteristiche del problema e della piattaforma hardware disponibile.

3.2 IL SOLUTORE PARALLELO MUMPS

Il solutore **MUMPS (MUltifrontal Massively Parallel sparse direct Solver)** risulta particolarmente efficace per problemi strutturali complessi, offrendo prestazioni superiori nella fattorizzazione diretta di matrici sparse attraverso algoritmi multi-fronte ottimizzati per architetture parallele.

MUMPS, infatti, rappresenta uno dei solutori diretti più avanzati e performanti per sistemi lineari sparsi di grandi dimensioni, sviluppato durante il progetto europeo PARASOL (1996-1999) da CERFACS, IRIT-ENSEEIHT e RAL. Il solutore è interamente implementato in Fortran 90 e utilizza MPI per il message passing, consentendo la

risoluzione efficiente di sistemi del tipo Ax = b, dove A è una matrice sparsa non simmetrica.

La filosofia progettuale di MUMPS si basa sull'algoritmo multi-fronte, che rappresenta l'evoluzione naturale dell'eliminazione gaussiana per matrici sparse, organizzando la fattorizzazione attraverso un albero di assemblaggio che riflette la struttura di riempimento della matrice.

Il cuore computazionale di MUMPS risiede nella sua capacità di sfruttare simultaneamente due livelli di parallelismo: il parallelismo derivante dalla complessità della matrice sparsa attraverso la decomposizione dell'albero multi-fronte, e il parallelismo disponibile per le operazioni dense sui fronti assemblati.

MUMPS utilizza una tecnica di scheduling dinamico distribuito che consente il pivoting numerico e la migrazione dei task computazionali verso processori meno caricati, ottimizzando automaticamente il bilanciamento del carico durante l'esecuzione. Questa architettura adattiva risulta particolarmente efficace per problemi complessi dove la distribuzione dei gradi di libertà non è uniforme nel dominio computazionale.

L'integrazione di MUMPS in OPENSEES avviene attraverso interfacce ottimizzate che sfruttano appieno le capacità del solutore per problemi di meccanica strutturale. Il solutore supporta fattorizzazioni LU per matrici generali e LDL^T per matrici simmetriche, utilizzando strategie di pivoting sofisticate per mantenere la stabilità numerica anche in presenza di matrici mal condizionate tipiche dell'analisi non lineare.

La gestione della memoria è particolarmente avanzata, con la possibilità di operare in modalità in-core per problemi di dimensioni moderate o out-of-core per modelli estremamente grandi, che eccedono la memoria disponibile.

3.3 LA LIBRERIA METIS

Un aspetto cruciale nell'efficacia computazionale di OPENSEESMP su sistemi paralleli risiede nell'utilizzo delle librerie **METIS (Multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning)**, sviluppate dal Karypis Lab dell'Università del Minnesota.

METIS costituisce un insieme di algoritmi seriali per il partizionamento di grafi, la suddivisione di mesh agli elementi finiti e il calcolo di ordinamenti che riducono il

riempimento delle matrici sparse, basati su approcci multi-livello di *bisezione ricorsiva* (teoria dei grafi).

Nel contesto di OPENSEES, le librerie METIS svolgono un ruolo fondamentale nella fase di pre-processing parallelo, convertendo automaticamente la mesh agli elementi finiti in un grafo equivalente e applicando algoritmi sofisticati per ottenere un partizionamento ottimale del dominio computazionale.

Il **processo di partizionamento** opera attraverso tre fasi distinte: la contrazione progressiva del grafo originale attraverso una sequenza di grafi sempre più piccoli, l'applicazione di algoritmi di partizionamento sul grafo più contratto, e infine la proiezione e raffinamento della soluzione sui grafi di dimensioni crescenti fino al grafo originale.

Questa metodologia garantisce un bilanciamento ottimale del carico computazionale tra i processori, minimizzando simultaneamente la comunicazione inter-processore attraverso la riduzione dei tagli nel grafo.

L'integrazione nativa di METIS in OPENSEES avviene in modo trasparente all'utente, che può beneficiare di questa ottimizzazione automatica senza dover specificare manualmente la decomposizione del dominio.

3.4 IL PRE-POST PROCESSORE STKO

STKO (Scientific ToolKit for OPENSEES) rappresenta l'interfaccia grafica più avanzata disponibile per OPENSEES, trasformando radicalmente l'interazione tra ingegneri, ricercatori e sviluppatori attraverso un ambiente grafico potente, intuitivo e completamente integrato. Questo strumento rivoluzionario supera una delle limitazioni storiche di OPENSEES, ovvero l'assenza di un'interfaccia grafica nativa, fornendo un ambiente di modellazione CAD professionale specificamente ottimizzato per analisi strutturali e geotecniche avanzate.

STKO funziona come pre-processore e post-processore completo, generando automaticamente file di input TCL per OPENSEES e fornendo strumenti di visualizzazione avanzati per l'interpretazione dei risultati.

L'architettura di STKO si basa su un'interfaccia Python programmabile che consente agli utenti di personalizzare ed estendere le funzionalità del software secondo specifiche esigenze di ricerca o professionali. Sfruttando appieno le potenzialità di

OPENSEES, STKO abilita aziende, ricercatori, accademici e professionisti a sbloccare le vaste capacità del software, offrendo strumenti di modellazione geometrica avanzati, definizione intuitiva di proprietà dei materiali, condizioni al contorno sofisticate e sistemi di meshing automatico ottimizzati per diverse tipologie strutturali. La piattaforma integra, inoltre, capacità di importazione da formati CAD standard, facilitando il trasferimento di modelli complessi da software di progettazione commerciali.

L'ambiente di post-processing di STKO fornisce strumenti di visualizzazione scientifica all'avanguardia, consentendo la creazione di rappresentazioni grafiche interattive per campi di spostamento, tensioni, deformazioni e altri parametri di risposta strutturale. La piattaforma supporta animazioni dinamiche della risposta temporale, facilitando l'interpretazione del comportamento strutturale durante eventi sismici o altri carichi dinamici. Questa integrazione sinergica tra le capacità computazionali di OPENSEES e l'interfaccia user-friendly di STKO facilita l'accesso alle analisi strutturali avanzate per coloro che non hanno familiarità nella programmazione di uno dei linguaggi utilizzati dagli interpreter OPENSEES o con sistemi operativi che richiedono configurazioni ad hoc, benché ampiamente supportati.

3.5 SVILUPPO DEL PRE-POST PROCESSORE OPENSTOOLS

Nella fase di apprendimento del codice OPENSEESMP ed OPENSEESSP, prima dell'approccio all'interfaccia STKO, si sono mossi preziosi passi nella programmazione di un pre-post processore in codice Python in grado di svolgere le funzioni di base per la codifica in codice TCL del modello di calcolo e delle fasi di calcolo tipiche dei problemi geotecnici dinamici. Da uno stato embrionale, questo codice è stato ulteriormente sviluppato nell'ambito del presente dottorato fino alla versione 2.0 e prende il nome di **OPENSTOOLS (v. 2.0)** (Cavallo, 2024).

Le librerie sviluppate nascono dall'esigenza di colmare il divario esistente tra la generazione di mesh complesse mediante software specializzati, come Gmsh, e la loro implementazione in ambienti di calcolo agli elementi finiti, come OPENSEES. Infatti, con poche modifiche alle funzioni implementate sarebbe possibile convertire l'applicazione ad altri software che dispongono di un API e la possibilità di importare mesh user-defined.

Con il codice OPENSTOOLS, programmato secondo la tecnica della programmazione ad oggetti, si è quindi in grado di importare la mesh del modello geometrico dal software Gmsh e di dividerla per un numero n di processori superiore a 2, imporre un numero n di modelli costitutivi da libreria per un numero n di domini, applicare le condizioni al contorno di vario genere ed altre funzioni fondamentali per la modellazione dedicata al problema geotecnico sismico. Il limite, allo stato attuale riguarda l'assenza di una interfaccia CAD e la disponibilità di soli 2 tipi di elementi finiti utilizzabili: l'elemento finito esaedrico up a 20 nodi e l'elemento quadrato up a 9 nodi. La scelta di programmare con questi elementi è data dalla possibilità di eseguire analisi dinamiche accoppiate con questi elementi ed associare loro modelli costitutivi avanzati tipo il "Pressure Independent Multi Yield" e il "Pressure Dependent Multi Yield" (Zhaohui, 2000).

Il codice in **APPENDICE 6** è liberamente scaricabile da piattaforma GitHub. Va notato che il codice è ancora in via di sviluppo e il prossimo step sarà l'eliminazione della dipendenza dalle librerie Python OPENSEESPY con l'eliminazione della generazione dell'eseguibile in codice Python.

Tra le altre dipendenze si annovera: Gmsh, Gmsh2opensees e numpy (APPENDICE 6 - py code 1).

Il sistema proposto quindi non si limita a fornire una semplice interfaccia di traduzione, ma implementa una serie di funzionalità avanzate che permettono di gestire in modo efficiente problemi di notevole complessità computazionale, includendo la possibilità di esecuzione parallela su architetture multi-processore.

L'architettura delle librerie OPENSTOOLS è stata progettata seguendo principi di modularità e scalabilità. La struttura fondamentale si basa su una gerarchia di classi che riflette la progressione logica dell'analisi agli elementi finiti.

Al centro dell'architettura troviamo la **classe Model** (APPENDICE 6 - py code 2), che gestisce l'interfaccia con la mesh importata da Gmsh e fornisce i metodi fondamentali per la manipolazione della struttura dati. Questa classe viene arricchita dalla **classe App** (APPENDICE 6 - py code 3), che aggiunge funzionalità specifiche per la gestione dinamica di vincoli e carichi durante le diverse fasi dell'analisi.

La funzione **trova_nodo** (APPENDICE 6 - py code 4) rappresenta un elemento fondamentale nell'architettura multi-processore del sistema. Questa funzione accetta come parametri un dizionario ("dictionary") che mappa i processori agli elementi e ai loro nodi, insieme al valore del nodo da localizzare. Il suo compito consiste nell'identificare quali processori contengono un determinato nodo all'interno dei loro elementi assegnati. L'algoritmo itera attraverso il dizionario dei processori, esaminando per ciascuno la lista degli elementi e verificando la presenza del nodo cercato nelle liste nodali associate. Il risultato viene restituito come un array numpy di interi univoci, garantendo che ogni processore venga identificato una sola volta anche se il nodo appare in multipli elementi gestiti dallo stesso processore. Questa funzionalità risulta essenziale per la corretta generazione degli script paralleli, dove le operazioni sui nodi devono essere eseguite solo dai processori che effettivamente gestiscono quei nodi.

Complementare alla precedente, la funzione **trova_elemento** (APPENDICE 6-py code 5) si occupa di identificare il processore responsabile di un elemento specifico. A differenza della funzione per i nodi, questa restituisce un singolo valore poiché ogni elemento è assegnato univocamente a un processore. L'implementazione attraversa il dizionario dei processori fino a trovare l'elemento cercato, restituendo immediatamente il numero del processore corrispondente. Questa funzione viene utilizzata principalmente durante la generazione degli elementi finiti, quando è necessario assegnare le istruzioni di creazione degli elementi al processore appropriato negli script di esecuzione parallela.

Il costruttore della **classe Model** stabilisce le fondamenta per tutte le operazioni successive sul modello. Durante l'inizializzazione, la classe acquisisce la mappa dei gruppi fisici dalla mesh Gmsh attraverso la variabile globale **PhysGr**, che contiene le informazioni su tutti i gruppi fisici definiti nella mesh, inclusi volumi, superfici, linee e punti. Il costruttore inizializza anche il vettore **uniqueVector**, una struttura dati versatile utilizzata per memorizzare temporaneamente liste di nodi o elementi durante le varie operazioni di processamento. L'attributo **nomeEl** memorizza il nome dell'elemento geometrico correntemente selezionato per il processamento, permettendo alla classe di operare selettivamente su porzioni specifiche della mesh.

Il metodo **setGeoElement** (APPENDICE 6- py code 6) fornisce il meccanismo per selezionare quale gruppo fisico della mesh deve essere processato. Accettando come parametro una stringa che identifica il nome del gruppo fisico, questo metodo aggiorna lo stato interno della classe per focalizzare le operazioni successive sul gruppo specificato. Questa funzionalità è fondamentale per l'applicazione selettiva di proprietà, vincoli o carichi a specifiche regioni del modello, come volumi rappresentanti diversi strati di terreno o superfici che definiscono i confini del dominio.

Il metodo **getElementsVectors** (APPENDICE 6- py code 7) costituisce uno degli elementi più complessi e cruciali della classe Model. La sua funzione principale consiste nell'estrarre tutte le informazioni rilevanti relative agli elementi e ai nodi del gruppo fisico correntemente selezionato. Il metodo implementa una logica differenziata basata sulla dimensionalità del gruppo fisico. Per elementi tridimensionali, utilizza direttamente le funzioni di *Gmsh2OPENSEES* per ottenere i tag degli elementi, i tag dei nodi, i nomi degli elementi e il numero di nodi per elemento. Per elementi di dimensionalità inferiore, il metodo implementa un approccio più articolato, recuperando prima le entità geometriche associate al gruppo fisico e poi estraendo gli elementi da ciascuna entità. Questa distinzione è necessaria per gestire correttamente superfici composte da multiple facce o linee composte da multiple curve, situazioni comuni nelle mesh complesse utilizzate in geotecnica.

Esclusivo della versione multi-processore, il metodo **mDict** (APPENDICE 6- py code 8) implementa l'algoritmo fondamentale per la distribuzione degli elementi tra i processori disponibili. Il metodo inizia creando una nuova istanza della classe Model e impostando il gruppo fisico 'Solid' come target. Dopo aver estratto tutti gli elementi e i loro nodi, costruisce un dizionario che mappa ogni elemento ai suoi nodi costituenti. L'algoritmo calcola quindi il numero di elementi da assegnare a ciascun processore, escludendo il processore. La distribuzione avviene attraverso un doppio ciclo annidato che assegna sequenzialmente blocchi di elementi ai processori, garantendo che ogni processore riceva approssimativamente lo stesso numero di elementi. Il dizionario risultante costituisce la base per tutta la generazione successiva di script paralleli.

Il metodo **setUniqueVector** (APPENDICE 6- py code 9) aggiorna il vettore di lavoro interno della classe con un nuovo insieme di dati. Questo metodo apparentemente semplice svolge un ruolo cruciale nel flusso di lavoro della classe, permettendo il riutilizzo della stessa istanza Model per operazioni successive su diversi insiemi di nodi o elementi. La flessibilità offerta da questo metodo permette di concatenare operazioni complesse senza la necessità di creare nuove istanze della classe.

Complementare al precedente, il metodo **getUniqueVector** (APPENDICE 6- py code 10) fornisce accesso al vettore di lavoro corrente. Questa funzionalità è essenziale per il passaggio di dati tra diverse operazioni e per l'ispezione dello stato interno della classe durante il debugging o la validazione dei risultati intermedi.

La funzione **mRemoveXY** (APPENDICE 6 - py code 11) rimuove simultaneamente i vincoli nelle direzioni x e y, mantenendo eventuali vincoli in z o sulla pressione interstiziale. Questa funzione è particolarmente utile per liberare il movimento nel piano orizzontale, mantenendo i vincoli nella direzione verticale.

La funzione **mTieNodesX** (APPENDICE 6 - py code 12) implementa una variante della funzione mTieNodes, che vincola solo la componente x dello spostamento. Questa specializzazione è utile quando si desidera mantenere la periodicità in una sola direzione, permettendo movimenti relativi nelle altre. L'implementazione segue la stessa logica di identificazione delle coppie di nodi, ma applica equalDOF solo per il DOF 1 (direzione x).

3.5.1 FUNZIONI PER LA MODELLAZIONE 3D

Il metodo **nodeNumCoordVector** (APPENDICE 6 - py code 13) trasforma una lista di tag nodali nelle corrispondenti coordinate spaziali. Per ogni nodo nel vettore unique corren-te, il metodo interroga il modello Gmsh per ottenere le coordinate x, y e z, assem-blando poi queste informazioni in una struttura dati che associa ogni numero di no-do alle sue coordinate tridimensionali. Questa trasformazione è fondamentale per tut-te le operazioni che richiedono informazioni geometriche, come il calcolo di distanze, l'identificazione di nodi corrispondenti su facce opposte, o la verifica di allineamenti.

Il metodo **listOfTagsij** (APPENDICE 6 - py code 14) gestisce la conversione di strutture dati bidimensionali in array monodimensionali di tag unici. Accettando come parametri le dimensioni della matrice di input, il metodo itera attraverso tutti gli elementi della matrice bidimensionale contenuta in uniqueVector, raccogliendo tutti i tag in una lista lineare. Successivamente, applica l'operazione unique di numpy per rimuovere eventuali duplicati, restituendo un array ordinato di tag unici. Questa operazione è particolarmente utile quando si processano liste multiple di nodi provenienti da elementi diversi, garantendo che ogni nodo venga processato una sola volta.

Il metodo **makeUnique** (APPENDICE 6 - py code 15) fornisce una versione semplificata dell'operazione di rimozione duplicati, operando direttamente sul vettore unique corrente senza parametri aggiuntivi. Il metodo converte il vettore in un array numpy, lo appiattisce se necessario, e applica l'operazione unique per eliminare i duplicati. Questa funzionalità viene utilizzata frequentemente nel workflow di processamento per garantire che liste di nodi o elementi non contengano ripetizioni.

Il metodo **nodeCornEdge** (APPENDICE 6 - py code 16) implementa un algoritmo sofisticato per la classificazione dei nodi in elementi di ordine superiore. Il metodo accetta come parametro il tipo di elemento da processare, supportando elementi a 8 nodi, 20 nodi o 9 nodi. Per elementi a 20 nodi, il metodo separa i primi 8 nodi (corner nodes) dagli ultimi 12 (edge nodes), organizzandoli in due liste separate. Questa distinzione è fondamentale per l'applicazione corretta delle condizioni al contorno nelle analisi accoppiate, dove i nodi corner tipicamente includono il grado di libertà della pressione, mentre i nodi edge non lo fanno. Per elementi a 8 nodi, il metodo divide semplicemente i nodi in due gruppi di 4, mentre per elementi a 9 nodi (utilizzati nelle analisi 2D) separa i 4 nodi corner dai 5 nodi edge includendo il nodo centrale.

Il metodo **sendOpsNodes** (APPENDICE 6 - py code 17) gestisce la creazione effettiva dei nodi nel modello OPENSEES. Accettando come parametri il numero di dimensioni spaziali e il numero di gradi di libertà per nodo, il metodo prima configura il modello OPENSEES con le dimensioni appropriate. Per la versione multi-processore, recupera il dizionario di distribuzione dei processori e genera sia i comandi Python che TCL per la creazione dei nodi. Per ogni nodo nel vettore unique, estrae le coordinate e

crea il nodo in OPENSEES. Nella versione parallela, identifica quali processori gestiscono ciascun nodo e genera le istruzioni condizionali appropriate negli script, garantendo che ogni nodo venga creato solo dai processori che effettivamente lo utilizzano. La generazione simultanea di script Python e TCL garantisce la compatibilità con diversi ambienti di esecuzione.

Il metodo **sendOpsFixes** (APPENDICE 6 - py code 18) implementa l'applicazione di vincoli nodali al modello. I parametri x, y, z e p specificano quali gradi di libertà vincolare (1 per vincolato, 0 per libero), con p che rappresenta la pressione del fluido per analisi accoppiate. Il metodo distingue tra nodi con 4 gradi di libertà (includenti la pressione) e nodi con 3 gradi di libertà (solo spostamenti). Per la versione multi-processore, il metodo genera script condizionali che garantiscono l'applicazione dei vincoli solo dai processori appropriati. Una caratteristica importante è l'assegnazione dei vincoli anche al processore 0 per i nodi di bordo, garantendo la consistenza delle condizioni al contorno durante l'analisi parallela.

Si inserisce ora un'altra classe, la **classe App** (APPENDICE 6 - py code 19)

Il metodo **subReacForce** (APPENDICE 6 - py code 20) della classe App implementa una procedura fondamentale per la transizione tra analisi statiche e dinamiche. Durante l'analisi statica iniziale sotto carichi gravitazionali, molti nodi sono vincolati per raggiungere l'equilibrio. Prima di procedere con l'analisi dinamica, questi vincoli devono essere rimossi e sostituiti con forze equivalenti alle reazioni vincolari calcolate. Il metodo inizia richiedendo all'utente di specificare quale elemento geometrico processare e in quale direzione applicare le forze sostitutive. Successivamente, estrae tutti i nodi dell'elemento specificato, li separa in nodi corner e edge, e per ciascun processore genera file contenenti i comandi per applicare carichi equivalenti alle reazioni vincolari. La distinzione tra nodi corner e edge è mantenuta per rispettare i diversi gradi di libertà. L'output consiste in file separati per ogni processore, permettendo l'esecuzione efficiente in ambiente parallelo.

I metodi **mfixSpX, mfixSpY e mfixSpZ** (APPENDICE 6 - py code 21) implementano una strategia sofisticata per l'applicazione di vincoli di spostamento, che preservano la deformazione corrente. Questi metodi sono particolarmente utili nelle analisi

multi-stadio, dove si desidera "congelare" gli spostamenti in una direzione specifica mantenendo la configurazione deformata raggiunta. Ciascun metodo opera su una direzione specifica (X, Y o Z) e richiede all'utente di specificare l'elemento geometrico da vincolare e il tipo di elementi (8-nodi o 20-nodi). Il metodo estrae tutti i nodi rilevanti, separa corner da edge, e genera comandi che applicano vincoli di tipo sp (single-point constraint) che mantengono lo spostamento corrente. Per la versione multi-processore, vengono generati file specifici per ogni processore contenenti solo i vincoli per i nodi di loro competenza.

Il metodo **mRemove** (APPENDICE 6 - py code 22) gestisce la rimozione completa di tutti i vincoli sp da un gruppo di nodi specificato. Il processo inizia con l'identificazione dell'elemento geometrico da cui rimuovere i vincoli. Prima della rimozione, il metodo registra le reazioni vincolari correnti per ciascun nodo, informazione che può essere utilizzata successivamente per applicare forze equivalenti. La rimozione avviene separatamente per nodi corner (con 4 DOF) e edge (con 3 DOF), garantendo che tutti i gradi di libertà appropriati vengano liberati. Nella versione multi-processore, il metodo genera script condizionali che assicurano che ogni processore rimuova solo i vincoli dai nodi di sua competenza, con una gestione speciale per il processore 0 che gestisce i nodi di bordo.

I metodi **mRemoveX** e **mRemoveY** (APPENDICE 6 - py code 23) forniscono una rimozione selettiva dei vincoli, limitata a una specifica direzione. Questi metodi seguono una logica simile a mRemove, ma operano solo sul grado di libertà corrispondente alla direzione specificata. Questa funzionalità è essenziale quando si desidera liberare il movimento in una direzione mantenendo i vincoli nelle altre, situazione comune nelle analisi di risposta sismica, dove in è necessario liberare il movimento nella direzione orizzontale, mantenendo vincoli verticali o viceversa.

La funzione **mDefine** (APPENDICE 6 - py code 24) orchestra la definizione completa di tutti i nodi del modello tridimensionale. La funzione inizia creando un'istanza della classe Model e, per la versione multi-processore, genera il dizionario di distribuzione dei processori. Successivamente, imposta l'elemento geometrico 'Solid' come target ed estrae tutti gli elementi volumetrici dalla mesh. Utilizzando il metodo

nodeCornEdge, separa i nodi in corner ed edge per elementi a 20 nodi up. Per i nodi corner, la funzione applica una serie di trasformazioni, quali estrazione dei tag unici, conversione in coordinate, e creazione in OPENSEES con 4 gradi di libertà. Il processo viene poi ripetuto per i nodi edge, che vengono creati con soli 3 gradi di libertà. Questa separazione è fondamentale per l'implementazione corretta degli elementi up dove la pressione del fluido è definita solo nei nodi corner.

Esclusiva della versione multi-processore, la funzione **boundNodesPid0** (APPENDICE 6 - py code 25) implementa una strategia cruciale per la gestione della comunicazione inter-processore. La funzione identifica tutti i nodi appartenenti alle superfici di bordo del dominio e li assegna esplicitamente al processore 0. Questo approccio centralizzato per i nodi di bordo elimina potenziali conflitti nella gestione delle condizioni al contorno e semplifica l'implementazione di elementi di interfaccia come i dashpot. La funzione itera attraverso tutti i gruppi fisici di superficie, estrae i nodi corner e edge separatamente, e genera script che creano questi nodi esclusivamente sul processore 0.

La funzione **mgetFixedCoord** (APPENDICE 6 - py code 26) fornisce un meccanismo per recuperare informazioni sui vincoli correntemente applicati al modello. La funzione interroga OPENSEES per ottenere la lista di tutti i nodi vincolati; quindi, per ciascuno recupera le coordinate e i gradi di libertà vincolati. Il risultato è una struttura dati tripla contenente i tag dei nodi vincolati, le loro coordinate e i DOF vincolati per ciascun nodo. Questa informazione è essenziale per operazioni che devono preservare o modificare vincoli esistenti.

La funzione **mFix** (APPENDICE 6 - py code 27) implementa l'applicazione interattiva di vincoli a un gruppo di nodi. La funzione richiede all'utente di specificare l'elemento geometrico da vincolare e quali gradi di libertà bloccare. Per ciascun nodo del gruppo selezionato, la funzione verifica prima se esistono vincoli preesistenti attraverso **getFixedDOFs**. Se il nodo è già parzialmente vincolato, la funzione aggiorna solo i gradi di libertà non ancora bloccati, preservando i vincoli esistenti. La logica di applicazione distingue tra nodi corner (4 DOF) e edge (3 DOF), garantendo che i vincoli siano applicati correttamente in base al tipo di nodo.

La funzione **mFixVol** (APPENDICE 6 - py code 28) rappresenta una specializzazione di mFix ottimizzata per l'applicazione di vincoli a elementi volumetrici completi. Mentre mFix può operare su qualsiasi gruppo fisico, mFixVol è specificamente progettata per volumi con elementi a 20 nodi. La funzione segue una logica simile a mFix, ma con ottimizzazioni specifiche per la gestione di grandi numeri di nodi tipici degli elementi volumetrici. L'interazione con l'utente rimane identica, richiedendo la specifica dei gradi di libertà da vincolare.

La funzione **mRec** (APPENDICE 6 - py code 29) configura il sistema di registrazione per l'output dei risultati dell'analisi. La funzione inizia identificando tutti i nodi del modello attraverso l'elemento 'Solid' e salvando la lista in un file nodeInfo.txt per riferimento futuro. Successivamente, configura una serie di recorder OPENSEES per catturare spostamenti, accelerazioni e pressioni nodali, nonché stress e deformazioni negli elementi. Per la versione multi-processore, la funzione genera recorder specifici per ogni processore, limitati ai nodi e agli elementi di loro competenza. Questo approccio distribuito riduce il sovraccarico di I/O e facilita il post-processing parallelo dei risultati.

La funzione **mGenFem20** (APPENDICE 6 - py code 30) gestisce la generazione di elementi brick a 20-8 nodi per analisi accoppiate solido-fluido. La funzione richiede all'utente di specificare il volume per cui generare la mesh, il tag del materiale da assegnare e, per la versione single-processore, i parametri di permeabilità. Per ogni elemento nel volume, la funzione estrae i 20 nodi e implementa un complesso algoritmo di riordinamento per conformarsi alla convenzione di numerazione di OPENSEES. L'algoritmo utilizza il sorting multi-livello delle coordinate per identificare la posizione relativa di ciascun nodo: prima separa i nodi per livello z (bottom/top), poi per coordinata y (front/back), e infine per coordinata x (left/right). Dopo il riordinamento, crea l'elemento 20_8_BrickUP con i parametri appropriati. Per la versione multi-processore, identifica il processore responsabile e genera gli script condizionali corrispondenti.

La funzione **mDiaf** (APPENDICE 6 - py code 31) implementa il concetto di diaframma rigido, essenziale per molte applicazioni geotecniche. La funzione richiede all'utente di identificare un nodo master e una superficie da irrigidire. Tutti i nodi della

superficie vengono quindi vincolati a muoversi solidalmente con il nodo master attraverso vincoli equalDOF nelle tre direzioni traslazionali. Questa condizione è tipicamente applicata alla base del modello, quando si utilizzano dashpot per simulare la radiazione di energia. L'implementazione esclude il nodo master stesso dal vincolo per evitare ridondanze e genera gli script appropriati per l'esecuzione parallela.

La funzione **mDash** (APPENDICE 6 - py code 32) implementa elementi dashpot viscosi per la simulazione di bordi assorbenti nelle analisi dinamiche. La funzione crea due nuovi nodi: uno fisso e uno mobile, posizionati alle coordinate specificate dall'utente. Il nodo mobile viene collegato a un nodo della mesh attraverso equalDOF, mentre un elemento zeroLength con materiale viscoso collega il nodo fisso a quello mobile. Il coefficiente di smorzamento viene calcolato come prodotto della densità del bedrock, della velocità delle onde di taglio e dell'area tributaria. La funzione supporta dashpot unidirezionali o multidirezionali secondo la scelta dell'utente. Per la versione multi-processore, tutti gli elementi del dashpot vengono assegnati al processore che gestisce il nodo della mesh collegato.

La funzione **floatingNodes** (APPENDICE 6 - py code 33) implementa un controllo di qualità fondamentale per la validazione del modello. La funzione identifica nodi che sono stati definiti, ma non sono stati connessi a nessun elemento, situazione che può indicare errori nella generazione della mesh o nella definizione del modello. L'algoritmo raccoglie tutti i nodi referenziati dagli elementi e li confronta con tutti i nodi definiti, restituendo la differenza come lista di nodi "floating". Questa verifica è particolarmente utile durante lo sviluppo e il debugging di modelli complessi.

La funzione **chPerm** (APPENDICE 6 - py code 34) gestisce la modifica dinamica della permeabilità degli elementi durante l'analisi. La funzione richiede all'utente di specificare i valori di permeabilità orizzontale e verticale ed itera attraverso tutti gli elementi del modello creando parametri OPENSEES per hPerm e vPerm. Questi parametri vengono immediatamente aggiornati con i valori specificati. Per la versione multi-processore, la funzione identifica il processore responsabile di ciascun elemento e genera gli script condizionali appropriati. Questa funzionalità è essenziale per simulare variazioni di permeabilità dovute a danneggiamento, consolidazione o altri processi evolutivi.

La funzione **mTieNodes** (APPENDICE 6 - py code 35) implementa condizioni al contorno periodiche attraverso il vincolo di facce opposte del modello. La funzione richiede all'utente di identificare una faccia master e una slave ed estrae tutti i nodi da entrambe le superfici. Per ogni coppia di nodi con la stessa coordinata z (assumendo facce verticali), applica vincoli equalDOF nelle direzioni x e y, permettendo il movimento verticale indipendente. Il processo viene ripetuto separatamente per nodi corner e edge per rispettare i diversi gradi di libertà. Questa funzionalità è fondamentale per la simulazione di mezzi stratificati infiniti in direzione orizzontale.

La funzione **mStage0** (APPENDICE 6 - py code 36) configura ed esegue l'analisi elastica iniziale (stage 0) necessaria per l'inizializzazione gravitazionale dello stato tensionale. La funzione inizia aggiornando lo stage dei materiali a 0, configurando il sistema come elastico lineare. Successivamente, imposta i parametri dell'analisi: constraints di tipo Penalty con rigidezza elevata, test di convergenza NormDispIncr, algoritmo KrylovNewton per la soluzione del sistema non lineare, numerazione Plain o RCM dei gradi di libertà, sistema di equazioni UmfPack o MUMPS, e integratore Newmark con parametri specificati. Per la versione multi-processore, genera script TCL con parametri ottimizzati per l'esecuzione parallela, includendo opzioni specifiche per il solver MUMPS.

La funzione **mRecDyn** (APPENDICE 6 - py code 37) configura recorder specializzati per analisi dinamiche con campionamento temporale controllato. A differenza dei recorder standard, questi utilizzano l'intervallo di tempo recDT per il campionamento, riducendo la dimensione dei file di output per analisi di lunga durata. La funzione configura recorder per spostamenti, velocità, accelerazioni e pressioni nodali, oltre a stress, strain e stato plastico per gli elementi. Per la versione multi-processore, genera recorder distribuiti che operano solo sui nodi e elementi locali a ciascun processore, con particolare attenzione alla sincronizzazione dei timestamp per facilitare il post-processing.

Le funzioni **mNodeInfoTxt, mNodeInfoCornerDat** e **mNodeInfoDat** (APPENDICE 6 - py code 38) gestiscono l'esportazione delle informazioni nodali in diversi formati. mNodeInfoTxt salva una lista semplice dei tag nodali in formato testo, utilizzata

principalmente come riferimento per i recorder. mNodeInfoCornerDat esporta le coordinate dei soli nodi corner in formato DAT con colonne separate per numero nodo e coordinate x, y, z. mNodeInfoDat esporta le coordinate di tutti i nodi nello stesso formato. Questi file sono essenziali per il post-processing e la visualizzazione dei risultati in software esterni.

La funzione **mGIDfile** (APPENDICE 6 - py code 39) genera file di mesh nel formato richiesto dal software GID per il post-processing. La funzione estrae tutti gli elementi e nodi del volume principale e li riorganizza secondo le convenzioni GID. Per elementi a 20 nodi, implementa una mappatura complessa tra l'ordinamento OPENSEES/Gmsh e quello richiesto da GID. Il file risultante include una sezione Coordinates con tutte le coordinate nodali e una sezione Elements con la connettività degli elementi nel formato GID. Questa funzionalità permette l'utilizzo delle potenti capacità di post-processing di GID per l'analisi dei risultati.

La funzione **mRegion20** (APPENDICE 6 - py code 40) definisce regioni di elementi per l'applicazione di parametri di smorzamento Rayleigh. La funzione richiede all'utente di specificare il volume target, il materiale associato e i quattro parametri di Rayleigh (alphaM, betaK, betaKinit, betaKcomm). Genera un comando TCL, che definisce la regione includendo tutti gli elementi del volume specificato e associando i parametri di smorzamento. Questa funzionalità è essenziale per modelli con smorzamento non uniforme o per l'applicazione selettiva di smorzamento a porzioni specifiche del dominio.

3.5.2 FUNZIONI PER LA MODELLAZIONE PIANA

Nel seguito, si definiscono le funzioni e i metodi per l'elemento in stato piano a 9 nodi up; queste funzioni sono disponibili solo in OPENSTOOLSSP, in quanto per la risoluzione dei modelli discretizzati con tali elementi è sufficiente un personal computer con minimo 4 core, che è ormai alla portata di tutti. Tuttavia, il codice può essere facilmente adeguato anche per la versione MP del tool.

La funzione **mDefine9** (APPENDICE 6 - py code 41) gestisce la definizione dei nodi per modelli bidimensionali con elementi a 9 nodi. A differenza della versione 3D, questa funzione crea preliminarmente file loads.TCL e removes.TCL per la gestione

successiva di carichi e rimozione vincoli. La funzione processa l'elemento 'Solid' (che in 2D rappresenta un'area) e separa i nodi corner (con 3 DOF includendo la pressione) dai nodi edge (con 2 DOF). La creazione dei nodi avviene attraverso il metodo specializzato sendOpsNodes9 che gestisce correttamente le coordinate bidimensionali.

I metodi **sendOpsNodes9** e **sendOpsFixes9** (APPENDICE 6 - py code 42) della classe Model sono specializzazioni per la gestione di nodi in analisi bidimensionali. sendOpsNodes9 crea nodi con coordinate bidimensionali, adattando il numero di gradi di libertà al contesto 2D. sendOpsFixes9 applica i vincoli, considerando che in 2D si hanno solo spostamenti lungo x e y, insieme alla pressione interstiziale. Entrambi i metodi generano script TCL appropriati per l'esecuzione, con particolare attenzione alla corretta sintassi per modelli 2D.

La funzione **mFix9** (APPENDICE 6 - py code 43) implementa l'applicazione di vincoli per modelli bidimensionali. La logica è simile a mFix, ma adattata per gestire solo due direzioni di spostamento insieme alla pressione interstiziale. La funzione distingue tra nodi con 2 DOF (solo spostamenti) e nodi con 3 DOF (spostamenti più pressione), applicando i vincoli appropriatamente. La verifica dei vincoli preesistenti e la loro preservazione segue la stessa filosofia della versione 3D ma con le necessarie semplificazioni dimensionali.

La funzione **mGenFem9** (APPENDICE 6 - py code 44) genera elementi quadrangolari a 9 nodi per analisi bidimensionali accoppiate. La funzione accetta come parametro la densità del materiale per calcolare il peso dell'unità di volume. Per ogni elemento, estrae i 9 nodi e implementa un algoritmo di riordinamento specifico per la topologia 2D. L'algoritmo identifica prima i nodi corner, basandosi sulle coordinate minime e massime, poi ordina i nodi edge secondo la loro posizione relativa. La creazione dell'elemento 9_4_QuadUP include tutti i parametri necessari per l'analisi accoppiata, inclusa la permeabilità e il peso dell'unità di volume.

Gli script TCL generati seguono la sintassi specifica per elementi 2D.

La funzione **mDash9** (APPENDICE 6 - py code 45) implementa dashpot viscosi per modelli bidimensionali. La logica è analoga a mDash, ma adattata per operare nello schema 2D. La funzione crea i nodi del dashpot con coordinate bidimensionali e

configura l'elemento zeroLength per operare nelle direzioni x e/o y. Il calcolo del coefficiente di smorzamento considera l'area tributaria appropriata per il contesto 2D. La funzione genera script TCL con la sintassi corretta per modelli bidimensionali, prestando particolare attenzione alla definizione delle direzioni attive del dashpot.

La funzione **mDiaf9** (APPENDICE 6 - py code 46) implementa diaframmi rigidi per modelli bidimensionali. A differenza della versione 3D che vincola tre traslazioni, questa versione vincola solo le due traslazioni nel piano. La funzione è tipicamente utilizzata per vincolare la base del modello 2D, quando si utilizzano dashpot per simulare bordi assorbenti. L'implementazione segue la stessa filosofia della versione 3D, ma con le necessarie semplificazioni dimensionali negli equalDOF.

La funzione **mTieNodes9** (APPENDICE 6 - py code 47) gestisce condizioni al contorno periodiche per modelli bidimensionali. La funzione identifica coppie di nodi su bordi opposti basandosi sulla coordinata y (assumendo bordi verticali) e applica vincoli equalDOF nelle direzioni x e y. Questa implementazione è fondamentale per simulare mezzi stratificati infiniti in direzione orizzontale nei modelli 2D, comuni nelle analisi di risposta sismica di depositi stratificati.

La funzione **mStage02D** (APPENDICE 6 - py code 48) configura l'analisi elastica iniziale per modelli bidimensionali. Mentre la struttura generale è simile a mStage0, i parametri sono ottimizzati per problemi 2D. La funzione configura il modello con ndm=2 e ndf=3, imposta i parametri del solver appropriati per la dimensionalità ridotta, e genera script TCL con sintassi specifica per analisi 2D. I parametri di convergenza e gli algoritmi di soluzione sono calibrati considerando le caratteristiche tipiche dei problemi bidimensionali.

I metodi **subReacForce9** e **mRemove9** (APPENDICE 6 - py code 49) della classe App sono specializzazioni per modelli 2D delle corrispondenti funzioni 3D. subReacForce9 gestisce la sostituzione di reazioni vincolari con forze equivalenti, distinguendo tra nodi con 2 e 3 DOF basandosi sulla lunghezza del vettore delle reazioni. mRemove9 rimuove i vincoli, considerando la dimensionalità ridotta, con particolare attenzione alla corretta gestione dei file loads.TCL e removes.TCL creati durante mDefine9. Entrambi i metodi generano script appropriati per l'esecuzione in ambiente 2D.

La funzione **meleNodeTag2D** (APPENDICE 6 - py code 50) è una utility specializzata per l'estrazione e organizzazione dei nodi da elementi 2D a 9 nodi. La funzione processa la lista lineare di nodi restituita da Gmsh e la riorganizza in una struttura, che associa ogni elemento ai suoi 9 nodi costituenti. Questa riorganizzazione è necessaria perché Gmsh restituisce i nodi degli elementi 2D come una lista continua, piuttosto che come array bidimensionale, richiedendo un processamento aggiuntivo per ricostruire la topologia degli elementi.

La funzione **mGIDfile2D** (APPENDICE 6 - py code 51) genera file di mesh bidimensionali nel formato GID. L'implementazione gestisce le peculiarità degli elementi quad a 9 nodi, incluso il riordinamento necessario per conformarsi alle convenzioni GID per elementi 2D. La funzione utilizza meleNodeTag2D per ottenere la struttura corretta degli elementi, poi implementa l'algoritmo di riordinamento specifico per la topologia 2D. Il file risultante include le sezioni standard Coordinates ed Elements ma con la dichiarazione appropriata per mesh 2D.

3.5.3 STRUMENTI PER LA VISUALIZZAZIONE DEI RISULTATI

Per la visualizzazione dei risultati in output è possibile utilizzare il post-processore GiD o quello di STKO per il formato. mpco inglobato in OPENSEES. Inoltre, si è messo a punto anche uno script per la visualizzazione dei risultati nodali sia in stato piano che in 2D attraverso la libreria Python PyVista (Sullivan & Kaszynski, 2019). L'applicazione prende il nome di OPSTVIS (Cavallo, 2025) e combina le librerie di un'altra applicazione ormai in disuso (non aggiornata) e presente su piattaforma GitHub, chiamata FeView.

OPSTVIS è un sistema completo di visualizzazione e post-processing, sviluppato specificamente per l'elaborazione dei risultati provenienti da analisi numeriche condotte con OPENSEES, di cui vi è un ulteriore sviluppo per l'output parallelo. Tale sistema è utile particolarmente nell'ambito della geotecnica computazionale. Il framework si articola attraverso due moduli principali, denominati **opstvis2D.py** e **opstvis3D.py**, progettati rispettivamente per gestire problemi bidimensionali e tridimensionali con particolare attenzione agli elementi finiti specifici per l'analisi accoppiata fluido-struttura in mezzi porosi saturi (20-8 BrickUP e 9-4 QuadUP). L'architettura del sistema

è stata concepita per garantire massima flessibilità e robustezza nel processamento di dati provenienti anche da simulazioni di grande scala, tipiche delle analisi parallele condotte con OPENSEESMP.

La funzione **OPENSEESTCLRead** (APPENDICE 6 - py code 52) costituisce il fondamento del sistema di acquisizione dati dal modello OPENSEES, implementando un parser specializzato per l'interpretazione dei file di input in formato TCL. Questa funzione accetta tre parametri fondamentali: il percorso del file TCL da analizzare, una stringa di pattern che identifica l'inizio delle righe di interesse, e il numero di colonne da estrarre da ciascuna riga identificata. L'algoritmo procede attraverso una lettura sequenziale del file, esaminando ciascuna riga e verificando se il suo inizio corrisponde al pattern specificato, attraverso un confronto diretto dei caratteri iniziali. Quando viene identificata una corrispondenza, la funzione procede alla tokenizzazione della riga utilizzando gli spazi come delimitatori, estraendo il numero specificato di colonne e accumulandole in una lista temporanea. Al termine della scansione, i dati raccolti vengono riorganizzati in un array NumPy bidimensionale attraverso l'operazione di reshape, garantendo una struttura dati ottimale per le successive elaborazioni numeriche. Questa implementazione risulta particolarmente efficace nell'estrazione di informazioni nodali, elementi finiti, condizioni al contorno e parametri di analisi dai file di configurazione OPENSEES, fornendo un'interfaccia unificata per l'accesso a dati eterogenei presenti nel file TCL.

La funzione **OPENSEESOutputRead** (APPENDICE 6 - py code 53) gestisce l'acquisizione e il parsing dei file di output generati da OPENSEES durante l'esecuzione dell'analisi numerica. L'implementazione sfrutta le capacità di pandas per la lettura efficiente di file strutturati, configurando il parser per interpretare correttamente file con delimitazione a spazi bianchi, caratteristica standard degli output OPENSEES. La funzione implementa meccanismi di robustezza attraverso la gestione delle righe di commento, identificate dal carattere '<', e la possibilità di saltare un numero configurabile di righe iniziali attraverso il parametro skiprows. La scelta di utilizzare pandas, come backend di lettura, garantisce prestazioni ottimali anche per file di output di dimensioni considerevoli, tipici delle analisi dinamiche con elevata risoluzione temporale. I dati letti

vengono immediatamente convertiti in array NumPy, garantendo compatibilità con le operazioni vettoriali successive e minimizzando l'overhead di memoria. Questa funzione risulta fondamentale per l'acquisizione delle storie temporali di spostamento, velocità, accelerazione, pressioni interstiziali e altre variabili di risposta calcolate durante l'analisi.

La funzione **step_static** (APPENDICE 6 - py code 54) implementa un meccanismo specializzato per l'estrazione delle informazioni relative al controllo del carico dall'analisi statica o quasi-statica. Attraverso l'invocazione della funzione OPENSEEST-CLRead con pattern specifico 'set numSteps', la funzione identifica ed estrae le dichiarazioni relative al numero di passi di carico definiti nel file TCL. L'implementazione processa l'array risultante estraendo specificamente la terza colonna, che contiene il valore numerico dei passi di carico. Questa informazione risulta cruciale per la corretta sincronizzazione temporale tra i risultati dell'analisi e la visualizzazione, permettendo di mappare correttamente ciascun set di risultati al corrispondente passo di carico nell'analisi. La funzione gestisce implicitamente la possibilità di multiple definizioni di passi di carico nel file TCL, caratteristica comune nelle analisi multi-fase tipiche della geotecnica, dove differenti fasi di carico possono richiedere differenti discretizzazioni temporali.

La funzione **out_response** (APPENDICE 6 - py code 55) rappresenta un componente critico nell'architettura del sistema, implementando la logica di processamento delle risposte nodali con adattamento automatico alla dimensionalità del problema. La funzione accetta quattro parametri: il file di output da processare, il numero di passi temporali, la dimensionalità del modello (ndm), e il tipo di risposta da estrarre. L'implementazione distingue tra problemi bidimensionali e tridimensionali attraverso una struttura condizionale che adatta il processamento alla specifica configurazione. Per problemi tridimensionali con tipo 'all', la funzione estrae direttamente le tre componenti spaziali dal file di output, riorganizzandole in una matrice tridimensionale attraverso l'operazione di reshape. Nel caso bidimensionale, l'algoritmo implementa una strategia di augmentation dimensionale, estraendo le componenti xy e generando automaticamente una coordinata z nulla per ciascun nodo attraverso la funzione np.repeat. Questa trasformazione garantisce compatibilità con il sistema di visualizzazione PyVista, che

opera nativamente in spazio tridimensionale. La struttura modulare della funzione permette future estensioni per gestire tipi di risposta specifici, come l'estrazione selettiva di componenti o la gestione di output multi-fisici caratteristici delle analisi accoppiate.

La funzione **NodeCoords** (APPENDICE 6 - py code 56) implementa l'algoritmo di estrazione e processamento delle coordinate nodali dal file TCL, adattandosi dinamicamente alla dimensionalità del problema attraverso la variabile ndm. Per modelli tridimensionali, la funzione invoca OPENSEESTCLRead con pattern 'node' e cinque colonne, estraendo l'identificatore del nodo e le tre coordinate spaziali. Le coordinate vengono convertite in formato float per garantire precisione numerica nelle successive operazioni geometriche. Nel caso bidimensionale, l'implementazione diventa più sofisticata, richiedendo una trasformazione geometrica per proiettare il modello planare nello spazio tridimensionale. Dopo l'estrazione delle coordinate xy attraverso la lettura di quattro colonne, l'algoritmo genera un vettore di zeri della lunghezza appropriata e procede alla costruzione iterativa di una lista di coordinate augmented. La scelta di utilizzare un approccio iterativo con successiva conversione in array NumPy attraverso column_stack garantisce flessibilità nella gestione di mesh di dimensioni variabili, mantenendo efficienza computazionale. Questa funzione risulta fondamentale per la corretta ricostruzione della geometria iniziale del modello, servendo come riferimento per il calcolo delle configurazioni deformate durante la visualizzazione.

La funzione **quad_cell** (APPENDICE 6 - py code 57) implementa l'algoritmo di costruzione della connettività per elementi quadrilateri a nove nodi, specificamente gli elementi 9_4_QuadUP utilizzati nell'analisi di mezzi porosi saturi. La funzione accetta due parametri: la lista degli elementi estratta dal file TCL e la lista dei nodi con i loro identificatori. L'implementazione include statement di debug attraverso print per verificare la corretta acquisizione dei dati durante lo sviluppo. L'algoritmo procede iterativamente attraverso ciascun elemento, costruendo tuple che definiscono la connettività secondo il formato richiesto da PyVista per celle di tipo VTK. Ciascuna tupla inizia con il valore 9, indicando il numero di nodi per elemento, seguito dagli indici globali dei nodi ottenuti attraverso la funzione np.argwhere. Questa funzione cerca l'identificatore di ciascun nodo locale nell'array globale dei nodi, restituendo l'indice di posizione che viene

convertito in intero. La complessità dell'implementazione deriva dalla necessità di mappare correttamente i nove nodi dell'elemento quadratico, includendo i nodi vertice, i nodi di bordo e il nodo centrale, secondo l'ordinamento specifico richiesto dal formato VTK. L'output finale è un array NumPy che codifica completamente la topologia della mesh, pronto per essere utilizzato nella costruzione dell'UnstructuredGrid di PyVista.

Le funzioni **cell_type_quad** e **cell_type_quad9** (APPENDICE 6 - py code 58) implementano la generazione dei vettori di tipo cella necessari per la corretta interpretazione della topologia della mesh in PyVista. La funzione cell_type_quad genera un array di valori costanti pari a 9, corrispondente al tipo VTK_QUAD nella libreria VTK, utilizzato per elementi quadrilateri lineari a quattro nodi. Parallelamente, la funzione cell_type_quad9 produce un array di valori 28, corrispondente al tipo VTK_QUADRATIC_QUAD, specifico per elementi quadrilateri quadratici a nove nodi. L'implementazione utilizza la funzione np.repeat per generare efficientemente array della dimensione richiesta, determinata dal numero di elementi nella mesh. Queste funzioni, seppur semplici nella loro implementazione, sono cruciali per garantire la corretta interpretazione geometrica degli elementi di ordine superiore, permettendo a PyVista di applicare le appropriate funzioni di forma per l'interpolazione dei campi scalari e vettoriali sulla superficie degli elementi.

La funzione **hex_cell** (APPENDICE 6 - py code 59) estende il concetto implementato in quad_cell agli elementi esaedrici a venti nodi, specificamente gli elementi 20_8_BrickUP utilizzati per analisi tridimensionali di terreni saturi. L'implementazione segue una struttura analoga, ma con complessità aumentata dovuta al maggior numero di nodi per elemento. Ciascuna tupla di connettività inizia con il valore 20, indicando il numero di nodi dell'elemento esaedrico quadratico, seguito dai venti indici globali dei nodi. L'ordinamento dei nodi segue la convenzione VTK per elementi esaedrici quadratici, richiedendo particolare attenzione nella mappatura dai nodi locali OPENSEES agli indici globali della mesh. L'algoritmo utilizza ripetutamente np.argwhere per localizzare ciascun nodo nell'array globale, con conversione esplicita a intero per garantire compatibilità con il formato richiesto da PyVista. La funzione include diagnostica attraverso print statement per facilitare il debugging della complessa mappatura topologica.

L'implementazione gestisce correttamente i nodi vertice, i nodi di spigolo e i nodi di faccia dell'elemento esaedrico, garantendo la corretta rappresentazione della geometria curvilinea caratteristica degli elementi di ordine superiore.

La funzione **cell_type_hex20** (APPENDICE 6 - py code 60) genera il vettore di identificazione del tipo di cella per elementi esaedrici a venti nodi, producendo un array di valori costanti pari a 25. Questo valore corrisponde al tipo VTK_QUADRATIC_HEXAHEDRON nella nomenclatura VTK, specificamente designato per elementi esaedrici quadratici con nodi ai vertici, al centro degli spigoli e senza nodi al centro delle facce o del volume. L'implementazione attraverso np.repeat garantisce la generazione efficiente di array anche per mesh con migliaia di elementi, caratteristica comune nelle analisi geotecniche tridimensionali. La corretta specificazione del tipo di cella è fondamentale per l'accurata rappresentazione geometrica e per l'interpolazione dei campi di risultati sugli elementi di ordine superiore.

La funzione **calcola_modulo_spostamenti** (APPENDICE 6 - py code 61) implementa l'algoritmo di indicizzazione temporale per l'estrazione dei vettori di spostamento corrispondenti a specifici istanti temporali dell'analisi. La funzione accetta un parametro temporale t e calcola gli indici di inizio e fine nel vettore globale dei risultati, basandosi su un time-step fisso di 0.01 secondi. L'implementazione gestisce il caso speciale del tempo iniziale (t=0) attraverso una struttura condizionale, assegnando direttamente gli indici per il primo blocco di risultati. Per tempi successivi, l'algoritmo calcola l'offset moltiplicando l'indice temporale per il numero di punti della mesh, garantendo l'estrazione del corretto subset di dati. La funzione utilizza slicing NumPy per estrarre efficientemente il vettore di spostamenti corrispondente all'istante temporale richiesto. Questa implementazione assume una struttura di storage dei risultati dove i dati di tutti i nodi per ciascun istante temporale sono memorizzati consecutivamente, convenzione standard negli output OPENSEES per analisi dinamiche. L'algoritmo è ottimizzato per minimizzare le operazioni di copia memoria, utilizzando view NumPy quando possibile.

La funzione **update** (APPENDICE 6 - py code 62) implementa il meccanismo di callback per l'aggiornamento dinamico della visualizzazione in risposta all'interazione

dell'utente con il widget slider. La funzione accetta un parametro temporale t fornito dal widget e orchestra l'aggiornamento completo della scena di visualizzazione. L'implementazione inizia con la definizione di un fattore di scala per l'amplificazione visiva degli spostamenti, necessario per rendere percettibili deformazioni che nella realtà fisica sarebbero dell'ordine dei millimetri o centimetri. L'algoritmo procede alla rimozione degli attori esistenti dalla scena attraverso le chiamate p.remove_actor, prevenendo sovrapposizioni visive e garantendo una transizione pulita tra frame. Successivamente, la funzione invoca calcola_modulo_spostamenti per ottenere il vettore di spostamenti corrispondente al nuovo istante temporale, aggiornando il campo scalare associato alla mesh deformata. La costruzione del vettore di deformazione tridimensionale avviene attraverso operazioni di column stacking, con generazione di componenti nulle per analisi bidimensionali o utilizzo diretto delle tre componenti per problemi tridimensionali. L'aggiornamento della geometria della mesh utilizza somma vettoriale tra la configurazione di riferimento e gli spostamenti scalati, garantendo che la visualizzazione mantenga coerenza geometrica. La funzione gestisce anche l'aggiornamento delle etichette nodali, ricalcolando le loro posizioni basandosi sulla configurazione deformata e aggiungendo un offset verticale per migliorare la leggibilità. L'implementazione conclude con la ri-aggiunta della mesh aggiornata alla scena, specificando tutti i parametri di visualizzazione, inclusi colormap, limiti di scala, e opzioni di rendering.

Il sistema implementa la **generazione di animazioni** attraverso un loop temporale che itera attraverso frame predefiniti, aggiornando progressivamente la configurazione della mesh e catturando ciascun frame per la composizione finale. L'implementazione utilizza il metodo open_gif di PyVista per inizializzare il writer dell'animazione, seguito da un loop che processa nframe in istanti temporali con intervallo dT_nframe. Per ciascun frame, l'algoritmo calcola i nuovi spostamenti, aggiorna la geometria della mesh attraverso somma vettoriale con fattore di scala, e invoca write_frame per catturare lo stato corrente della visualizzazione. La gestione delle etichette nodali richiede particolare attenzione, con rimozione e ri-aggiunta ad ogni frame per garantire il corretto posizionamento rispetto alla geometria deformata. L'implementazione include la specifica di color limits fissi attraverso il parametro clim, garantendo consistenza visiva tra

frame e facilitando l'interpretazione quantitativa dell'animazione. La chiusura del writer attraverso p.close() finalizza la generazione del file GIF, producendo un output direttamente utilizzabile per presentazioni e pubblicazioni.

L'implementazione della modalità interattiva sfrutta il sistema di widget di PyVista per fornire controllo real-time sulla visualizzazione temporale. Il widget slider viene configurato attraverso **add_slider_widget**, specificando la funzione di callback update, il range temporale basato sulla durata totale dell'analisi, e un titolo descrittivo. L'implementazione mantiene due istanze della mesh: una configurazione di riferimento immutabile e una mesh deformata che viene aggiornata dinamicamente. Questa strategia previene l'accumulo di errori numerici che potrebbe derivare da trasformazioni incrementali successive. Il sistema di coordinate per le etichette viene ricalcolato ad ogni aggiornamento per mantenere l'allineamento con la geometria deformata. L'implementazione include parametri di visualizzazione avanzati attraverso il dizionario sargs, configurando l'interattività della barra di scala e permettendo all'utente di modificare dinamicamente i parametri di visualizzazione. La specifica della posizione della camera attraverso cpos garantisce una vista ottimale della geometria, con 'xy' per problemi bidimensionali e 'iso' per visualizzazioni tridimensionali.

Il sistema implementa funzionalità di export attraverso il metodo mesh.save, generando file in formato VTK compatibili con l'ecosistema di software di visualizzazione scientifica. L'implementazione del salvataggio avviene dopo la costruzione completa della mesh e l'assegnazione dei campi scalari, garantendo che il file esportato contenga sia la geometria che i dati di risultato. Il formato VTK legacy ASCII viene utilizzato per massimizzare la compatibilità, permettendo l'importazione diretta in software come ParaView, VisIt, o Mayavi. L'export include automaticamente tutti i campi scalari e vettoriali associati alla mesh, preservando la possibilità di post-processing avanzato in ambienti esterni. Questa funzionalità risulta particolarmente utile per la generazione di visualizzazioni di pubblicazione quality attraverso software specializzati, o per l'integrazione dei risultati in pipeline di analisi più complesse.

L'implementazione del sistema pone particolare attenzione all'efficienza computazionale, cruciale quando si processano risultati di analisi dinamiche con migliaia di

passi temporali e decine di migliaia di gradi di libertà. L'utilizzo sistematico di operazioni vettorizzate NumPy minimizza l'overhead del Python interpreter, sfruttando le ottimizzazioni BLAS/LAPACK sottostanti. La strategia di pre-allocazione degli array e l'uso di view invece di copie dove possibile riduce la pressione sulla memoria e migliora le prestazioni del cache. L'implementazione del parsing dei file TCL attraverso lettura sequenziale invece di caricamento completo in memoria permette di gestire file di configurazione di grandi dimensioni senza impatti significativi sulle risorse di sistema. La struttura modulare delle funzioni facilita il profiling e l'ottimizzazione selettiva dei colli di bottiglia computazionali. Le operazioni di visualizzazione sfruttano l'accelerazione hardware attraverso VTK/OpenGL, delegando il rendering alla GPU quando disponibile. Questa architettura garantisce fluidità nell'interazione anche per mesh complesse, mantenendo frame rate interattivi durante la navigazione temporale dei risultati.

In appendice si allega il codice per gestire i risultati in output degli interpreter OPENSEESMP e OPENSEESSP (APPENDICE 6 - py code 63).

4. TEST IN RECAS CON SVILUPPO DI MODELLI FEM PROTOTIPO 1D, 2D E 3D

Gli strumenti analizzati e presentati nei capitoli precedenti sono stati elaborati per la necessità di interagire con i sistemi ad alte prestazioni messi a disposizione secondo precise modalità di uso del cluster HPC.

Tali sistemi, infatti, tipicamente dotati di sistema operativo Linux non permettono l'esecuzione del codice PLAXIS in quanto dedicato a sistemi operativi Windows.

Sono state proprio queste le motivazioni che hanno richiesto uno sforzo nella programmazione in ragione anche della mancanza o non conoscenza di altri software con una interfaccia grafica che permettessero un primo approccio più agevole.

Come si avrà modo di verificare dal confronto dei risultati nel problema della propagazione monodimensionale illustrati nel Capitolo 8 del presento lavoro di tesi, gli strumenti messi a punto in linguaggio Python si sono rivelati validi.

I modelli codificati con tali strumenti, infatti, applicano condizioni al contorno ed input di vario tipo su un dominio 3D non troppo complesso con risultati paragonabili a quelli del codice PLAXIS.

Pertanto, il codice OPENSTOOLS è stato adottato per la modellazione dei casi più semplici come, ad esempio, il caso della propagazione monodimensionale in ipotesi di free-field e base deformabile.

Il processo completo che parte dalla definizione della mesh fino all'ottenimento di un output è stato composto dalle seguenti fasi:

- implementazione di un container con all'interno l'applicazione per la risoluzione dei modelli FEM 2D e 3D con il software Singularity;
- definizione del file eseguibile contenente il modello FEM risolvibile dall'applicazione OPENSEES con OPENSTOOLS;
- definizione delle caratteristiche in termini di potenza computazionale da richiedere al sistema (numero CPU, memoria RAM…);
- esecuzione ed estrazione finale dell'output;
- post-processing dei risultati con OPENSTOOLS o STKO.

Con gli strumenti a disposizione, sono stati elaborati e risolti in parallelo i seguenti prototipi:

- modelli per l'analisi della propagazione sismica monodimensionale (risolti con 50 CPU);
- pendio piano omogeneo di forma arbitraria risolto con 35 CPU;
- pendio piano bistrato (risolto con 4 CPU);
- cubo multistrato (risolto con 4 CPU);
- modello 3D di un pendio naturale risolto con 60 CPU.

4.1 MODELLO PER L'ANALISI DELLA PROPAGAZIONE SISMICA 1D

Uno dei casi ideali esaminati nella fase di test è la simulazione del fenomeno di propagazione delle onde sismiche in condizioni monodimensionali. Allo scopo, è stato sviluppato un modello numerico 3D (Fig. 1a), che riproduce una colonna monodimensionale di terreno omogeneo, di spessore 40 m, in assenza di acqua, discretizzata con elementi tipo 20_8_BrickUP. Il modello numerico è costituito da 319 elementi con un numero di nodi pari a 2421. La misura media del lato del singolo esaedro è di 0,5 m.

Nella fase dinamica, le condizioni al contorno consistono nel vincolare le facce di normale parallela alla direzione di applicazione del segnale sismico (direzione x nel caso in esame, versore rosso in Fig. 1a) a muoversi della stessa quantità con vincoli di tipo tied-nodes, mentre le facce di normale perpendicolare alla direzione di applicazione del moto (direzione y, versore verde in Fig. 1a) sono libere di muoversi nella sola direzione x (condizione realizzata inserendo dei carrelli lungo x). Alla base della colonna è stata imposta la condizione di base deformabile ("compliant base"), realizzata assegnando contorni assorbenti mediante l'applicazione di smorzatori e applicando la storia temporale delle tensioni tangenziali determinate in funzione del segnale di riferimento (i.e. convoluzione del segnale registrato all'affioramento della formazione rocciosa, Chiaradonna 2022) e delle caratteristiche fisiche e meccaniche del substrato sismico.

La risposta del terreno è stata simulata con un modello visco-elastico lineare, in cui i parametri sono γ (peso dell'unità di volume) pari a 19.32 kN/m^3, V_S (velocità delle onde di taglio) pari a 163.05 m/s e ν (modulo di Poisson) uguale a 0.3; la capacità

dissipativa del terreno è stata introdotta mediante lo smorzamento viscoso dell'1% secondo la formulazione di Rayleigh (Rayleigh, 1945), assegnando α_R e β_R pari a, rispettivamente, 0.09558 e 0.7920E-03 (Laera & Brinkgreve, 2015; Amorosi et al., 2016). Questi parametri sono stati determinati considerando come estremo inferiore la frequenza di risonanza del banco di materiale argilloso e come estremo superiore il prodotto dell'estremo inferiore per il valore n dato dall'arrotondamento alla cifra intera superiore dispari del rapporto tra la frequenza fondamentale del segnale e la frequenza di risonanza del banco (Elia & Rouainia, 2022). Pertanto, i parametri di Rayleigh α_R e β_R sono stati definiti per un fattore di smorzamento dell' 1% per il range di frequenze 1.019 Hz $<$ f $<$ 3 Hz (Laera & Brinkgreve, 2015)

I risultati dell'analisi sono stati confrontati con quelli ottenuti dal modello di una colonna 2D, avente la stessa altezza e larghezza, discretizzato con 400 elementi triangolari lineari a 6 nodi (per un numero complessivo di 2800 nodi) ed implementato nel software PLAXIS 2D.

È stato utilizzato il codice PLAXIS2D per il confronto in quanto nella versione 3D non è possibile imporre i tie-nodes, condizione fondamentale per la simulazione in questione.

L'analisi è stata condotto con il metodo di Newmark in condizione "incodizionatamente stabile" con un passo temporale di 0.002 sec per un totale di 20 sec di segnale.

Il confronto, illustrato in termini di accelerazione nel tempo per un nodo in corrispondenza della estremità superiore della colonna in Fig. 1b, mette in evidenza il perfetto accordo tra i risultati delle due analisi, suggerendo la bontà della modellazione eseguita con l'applicazione di calcolo precedentemente illustrata.

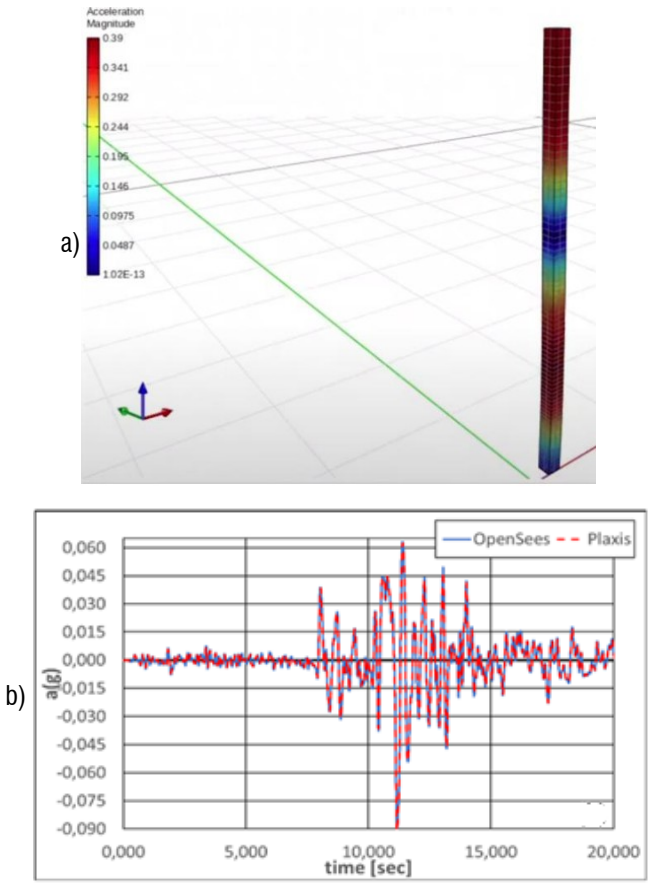


Fig. 1 a) Modello di colonna 3D in OPENSEES (versori: rosso=x, verde=y, blu=z, b) Confronto output OPENSEES – PLAXIS (Cavallo et al., 2024).

4.2 ANALISI DI UN PENDIO OMOGENEO DI FORMA ARBITRARIA

In questo caso è stata eseguita un'analisi di propagazione sismica in un modello 2D di pendio visco-elastico lineare costituito da 536 elementi 9_4_QuadUP con 2272 nodi, la cui esecuzione ha richiesto l'utilizzo di 35 CPU ed un tempo di calcolo di meno di 1 ora.

Il fine dell'operazione è stato di verificare a grandi linee, anche attraverso la sola visione dei contour:

- l'implementazione degli elementi nel codice OPENSTOOLS (condizioni al contorno, carichi, disposizione di nodi e generazione degli elementi finiti);
- il funzionamento del framework impostato su infrastruttura HPC (istruzioni per l'avvio delle analisi, risposta dei processori in merito alla ripartizione degli elementi, verifica del container generato ad hoc);
- l'interazione tra il software e il sistema HPC (corretta esecuzione dei comandi propri del software OPENSEESSP ed OPENSEESMP);
- l'esecuzione delle analisi pre-processate con l'applicazione STKO.

Dopo la definizione della geometria con CAD, questa è stata dapprima discretizzata utilizzando l'applicazione Gmsh in elementi UP piani a nove nodi. Successivamente, questi sono stati elaborati in codice OPENSEES insieme a fasi di calcolo e condizioni al contorno. Infine il codice è stato riscritto in automatico per dividere le istruzioni per il numero di processori selezionato dall'utente (in questo caso 35). Le fasi di calcolo imposte sono 2, una statica di inizializzazione ed una successiva dinamica. Nella fase statica sono stati assegnati vincoli tipici della condizione K_0 anche se la fase è stata risolta in condizioni di "gravity loading". Nella fase dinamica, invece, vi è stata la rimozione dei vincoli laterali, l'attivazione della condizione di compliant base per il contorno di base e l'applicazione dello stesso segnale naturale visto in precedenza (Laera & Brinkrgreve, 2015) convertito in storia di velocità nel tempo. La fase dinamica è stata risolta con l'integrazione nel tempo alla Newmark ed il materiale assunto è di tipo elastico isotropo con gli stessi parametri di Rayleigh visti in precedenza al solo fine di eliminare moti spuri o instabilità dovute a risonanze numeriche nel dominio. Come si può osservare in Fig. 2, non ci sono concentrazioni di spostamenti o valori anomali all'interno del dominio durante la fase dinamica. Inoltre, è stato possibile eseguire il calcolo interamente su infrastruttura RECAS ed estrarne i risultati in output in tempi molto brevi, inferiori all'ora. Pertanto, l'esperimento ha risposto alle domande di funzionalità in parallelo tanto del codice che del metodo di implementazione.

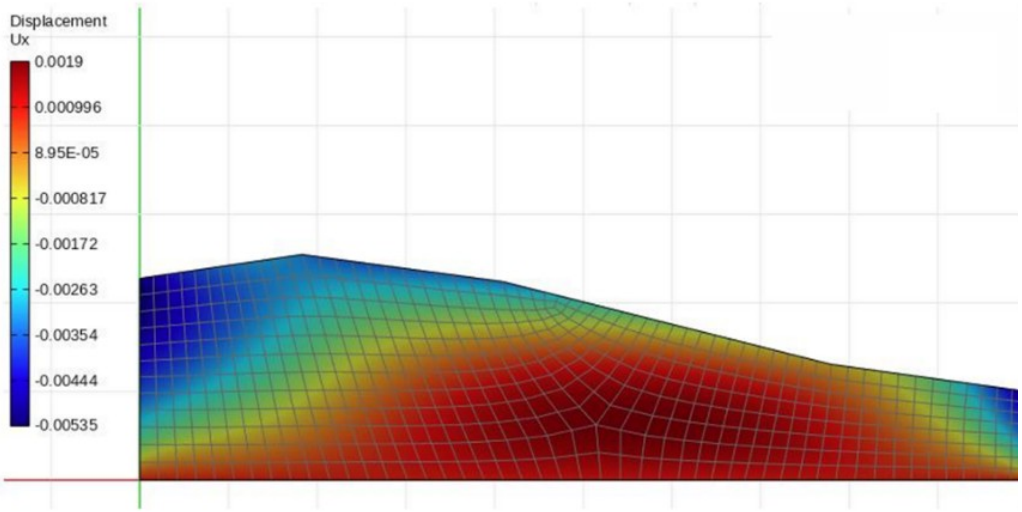


Fig. 2 Pendio visco-elastico, diagramma degli spostamenti orizzontali in fase dinamica relativo ad un generico time frame significativo per la verifica.

4.3 ANALISI DI UN PENDIO BISTRATO E DI UN CUBO MULTISTRATO

Nel presente caso si è testata la capacità di funzionalità delle funzioni dell'applicazione OPENSTOOLS (Cavallo, 2024), che permettono l'inserimento di più strati, utilizzando sempre 4 CPU e differenziando i casi per l'elemento a 20 nodi esaedrico e quello a 9 nodi piano. La falda è coincidente con il bordo superiore del modello (piano campagna) ed alla sua base è stata imposta una condizione di compliant base.

In questi casi vi è stato anche l'utilizzo dei tied-nodes ai bordi verticali di estremità nella direzione del moto consentito, ovvero quello lungo x (vettore rosso in Fig. 3a). Il moto nella direzione verticale è vincolato con carrelli. Lo schema delle fasi di calcolo, della soluzione matriciale, del modello costitutivo elastico isotropo con smorzamento alla Rayleigh è analogo a quello dei casi visti in precedenza.

La Fig. 3a mostra il caso del pendio bistrato, rappresentante un argine su un deposito di fondazione, per cui sono stati usati parametri elastici rappresentativi di due materiali differenti.

La Fig. 3b mostra, invece, il modello 3D formato da 3 scatole complementari: la fondazione più esterna (3), uno strato di interfaccia (2) e un cubo interno (1). Il

risultato finale ha dimostrato il perfetto rispetto delle condizioni al contorno imposte (uguali a quelle dei casi precedenti) ed il corretto funzionamento del modello costitutivo adottato. Inoltre, non vi sono state compenetrazioni o moti spuri delle parti modellate con materiali differenti.

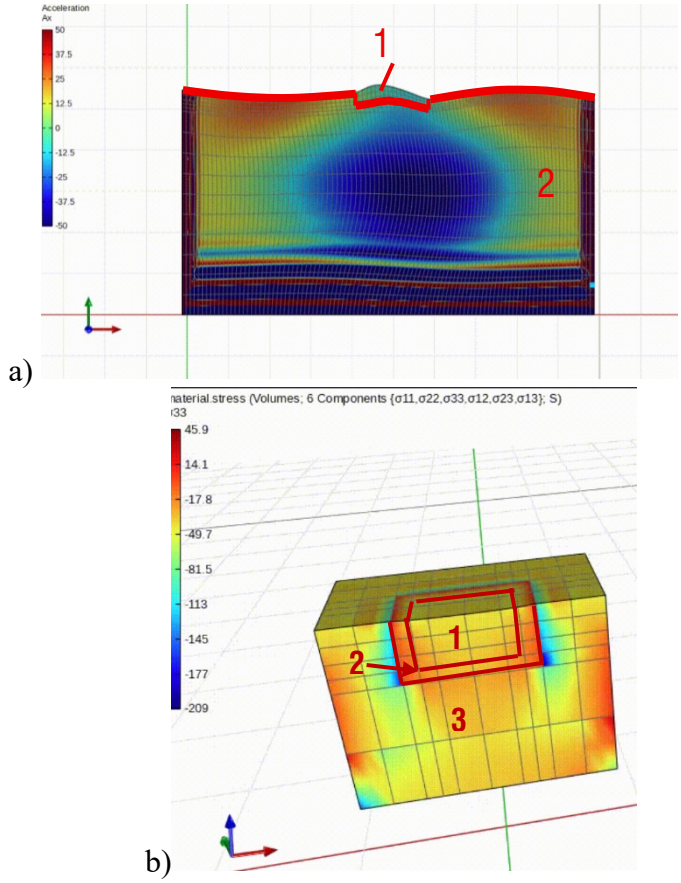


Fig. 3 a) Modello 2D di pendio bistrato, gli strati sono indicati con 1 e 2, b) Modello 3D di un cubo multistrato, gli strati sono indicati con 1, 2 e 3.

4.4 MODELLO 3D DI UN PENDIO NATURALE

Come nei casi precedenti, il fine di questa attività di modellazione e simulazione è stata quella di applicare la procedura definita al caso di un modello 3D di grandi dimensioni, al fine di valutare:

- la procedura di modellazione utilizzando l'applicazione OPENSTOOLS;
- la corretta codifica e applicazione dei vincoli nel software;
- la corretta suddivisione del modello secondo il numero di processori desiderato;
- concentrazioni di spostamenti o irregolarità dovute ad una non corretta imposizione di vincoli e parametri;
- il tempo di risoluzione di tutto il modello.

La Fig. 4 presenta il modello 3D avente la topografia del versante occidentale del comune di Chieuti. Il modello è stato ottenuto con elaborazione GIS (QGIS). Partendo dal DTM (Digital Terrain Model) del territorio del comune di Chieuti (Fig. 4a), sono state estratte le curve di livello con risoluzione di 1 m. Da questo è stato ritagliato un quadrato di misura di circa 1 Km x 1 Km rappresentativo dell'estensione del modello numerico implementato nel codice di calcolo (Fig. 4b).

Il modello 3D implementa un deposito omogeneo, visco-elastico lineare, con falda coincidente con il piano campagna. Le caratteristiche meccaniche assegnate al terreno sono: un modulo di Young pari a 123270 kPa ed un peso dell'unità di volume pari a 19.32 kN/m^3, con un modulo di Poisson pari a 0.2.Tale geometria è stata elaborata successivamente con il software open-source GiD e Salome-Meca con cui si è estrusa la superficie e discretizzato il modello con elementi a 20 nodi compatibili con quelli del software OPENSEESMP. Infine, si è creato il modello numerico finale e l'eseguibile con l'ausilio dell'interfaccia OPENSTOOLS precedentemente presentata con cui si è compilato automaticamente l'eseguibile poi risolto su sistema HPC con 60 CPU. Le condizioni al contorno dinamiche sono tali per cui il modello può scorrere in una sola direzione (x), mentre alla base del modello numerico è stata implementata la condizione di compliant base. A tale base, infine, è stato imposto il segnale sismico degli

esempi precedenti, ma amplificato di due volte al fine di verificare la convergenza dell'analisi anche durante eventi sismici di maggiore intensità.

Nonostante il processo sia andato a buon fine in termini di procedure di risoluzione e calcolo (Fig. 5), nonché di assenza di errore di impostazione per le condizioni al contorno, vi è stato un problema dovuto al set di parametri per definire la compliant base che ha sovra-amplificato il segnale. Successivamente, l'errore è stato individuato e corretto, modificando i coefficienti di calcolo all'interno dell'applicazione OPENSTOOLS.

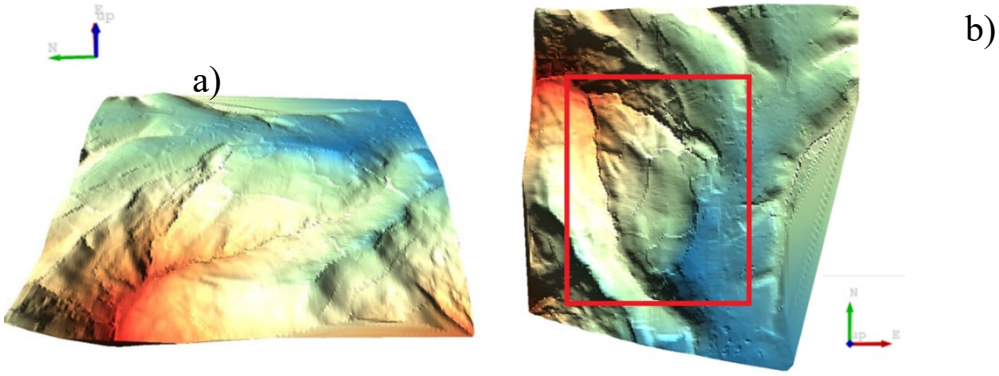
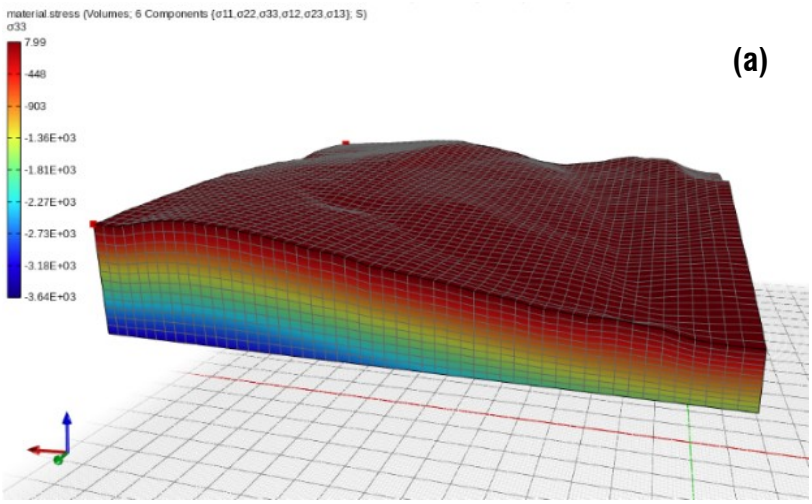


Fig. 4 a) DTM del Comune di Chieuti, b) Area selezionata per la costruzione del modello numerico 3D. I colori indicano la quota s.l.m. più alta (220 m - blu) e quella più bassa (62 m - rosso).



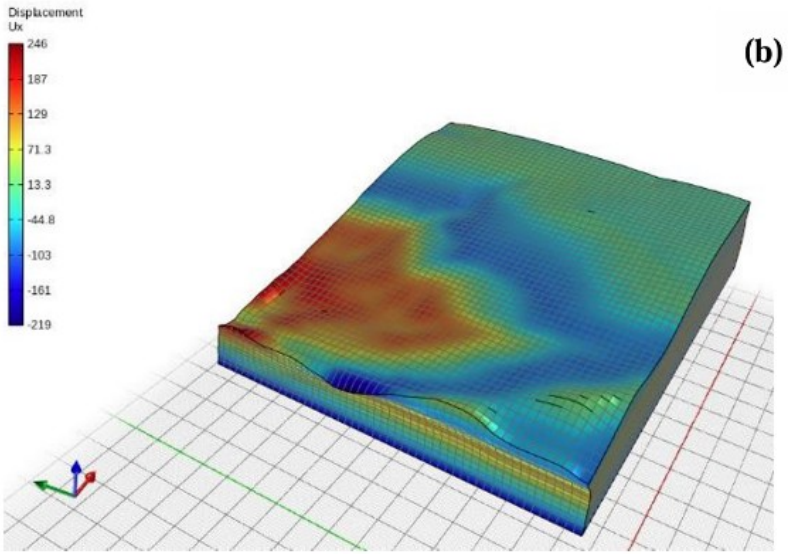


Fig. 5 a) Analisi geostatica 3D del pendio di Chieuti in materiale omogeneo, b) Risultati dell'analisi dinamica in termini di spostamenti nella direzione x.

In conclusione, i test di modellazione effettuati hanno avuto i seguenti obiettivi principali:

- testare le funzioni dell'applicazione OPENSTOOLS al fine della correzione di bug di esecuzione;
- verificare che le ipotesi sulle condizioni al contorno fossero correttamente applicate;
- verificare il corretto setup su sistema HPC del container OPENSEES al fine della corretta esecuzione in parallelo dei codici eseguibili;
- verifica del corretto funzionamento del calcolo parallelo e test dei tempi di esecuzione;

La validazione dell'applicazione con il confronto approfondito in campo visco-elastico e visco-elasto-plastico verrà mostrata nel Capitolo 8 attraverso il confronto dei risultati con il software PLAXIS.

5. IL CASO DI STUDIO: PENDIO DI CHIEUTI

5.1 INTRODUZIONE

Lo studio del fenomeno della propagazione sismica con modelli 2D e 3D sviluppati in codice OPENSEES è stato condotto con riferimento ad un pendio naturale, ben documentato dal punto di vista geotecnico. Si tratta del versante occidentale del comune di Chieuti, sede di un fenomeno franoso di lunga data, per il quale sono disponibili dati di sito e di laboratorio volti alla caratterizzazione geotecnica del sottosuolo (Sonnessa et al., 2023, di Lernia et al., 2023, Tagarelli et al., 2025, Santaloia et al., in prep).

5.2 EVOLUZIONE GEOMORFOLOGICA DEL VERSANTE

Il comune di Chieuti, situato a 221 m s.l.m. in provincia di Foggia, vede nascere le sue origini intorno al XV secolo con immigrati albanesi che si stabilirono tra il 1461 e il 1470 su un colle protetto da mura di cinta costituite da abitazioni allineate alle cui estremità erano presenti delle torri.

Il versante occidentale di Chieuti è frutto dell'emersione e sollevamento della successione marina composta dalle formazioni delle Argille di Montesecco (Qm1) e delle Sabbie di Serracapriola (Qm2) e della sovrastante successione continentale rappresentata dai Conglomerati di Campomarino (Qc1 + Qc2) a seguito di una fase tettonica evolutasi nel Pleistocene da medio a tardo, seguita dallo scavo fluviale ad Ovest del versante, caratterizzato da processi di down-cutting verso Est.

Il centro storico è costruito sopra uno strato continentale di sommità più sottile (formazione dei Conglomerati di Campomarino), composto da ghiaia, intervalli limosi, sabbiosi e argillosi, depositati in ambiente alluvionale. Il versante collinare di Chieuti è formato da una successione marina di avanfossa, rappresentata dalle formazioni delle Sabbie di Serracapriola e delle Argille di Montesecco, sovrastata da depositi continentali. Le Sabbie di Serracapriola possono essere suddivise in due unità, ovvero l'unità inferiore limoso-sabbiosa e l'unità superiore sabbiosa che include intercalazioni locali

di corpi ghiaiosi, mentre la formazione delle Argille di Montesecco consiste in argille marine grigie con livelli limoso-sabbiosi.

Da campagne di rilevamento e studi con metodi avanzati, basati su osservazioni ad alta risoluzione e satellitari (Lidar, InSarr etc..), nella zona di studio si sono desunti molteplici corpi di frana principali e secondari.

Il principale movimento franoso è caratterizzato da un'evoluzione retrogressiva, che si manifesta con meccanismi rotazionali multipli, con particolare riferimento a due corpi diversi aventi piede coincidente e nei pressi del torrente Fico.

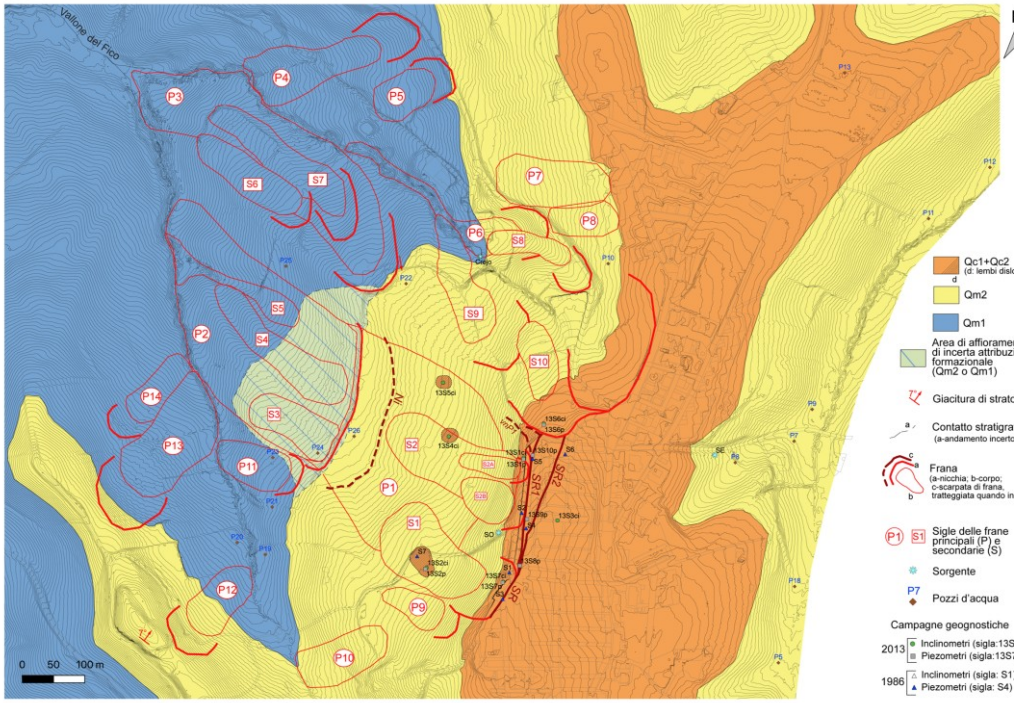


Fig. 6 Carta geologico-geomorfologica del versante occidentale di Chieuti (Santaloia et al., in prep).

Pertanto, dagli studi pregressi eseguiti in merito all'analisi sui fattori predisponenti ed innescanti dei meccanismi profondi, si ritiene che tali cinematismi siano dovuti a processi gravitativi dovuti ad una progressiva erosione asimmetrica della valle del fiume Bivento.

Inoltre, come ulteriore fattore vi è la presenza di una falda che a monte si attesta nei livelli sabbiosi per poi arrivare a valle in un materiale meno permeabile. Il livello piezometrico, infatti, a monte si mantiene intorno ai 20 m da piano campagna nello strato di sabbia, mentre a valle, proprio in corrispondenza del piede, vi è il sollevamento del livello stesso fino a piano campagna.

La sismicità del territorio di Chieuti è caratterizzata da un livello medio-alto a causa della vicinanza a diverse faglie trascorrenti come le faglie di Apricena e Mattinata. L'area di Chieuti è caratterizzata da una PGA compresa tra 0,175g e 0,225g su roccia affiorante, calcolata per un periodo di ritorno di 475 anni sulla base degli studi di pericolosità sismica dell'INGV.

Il sito è stato raramente colpito da eventi sismici di alta intensità, tra i quali i più importanti sono il terremoto di San Severo (30/07/1627), grado VII della scala MCS e il terremoto del Molise del 2002, con Mw 5,9.

5.3 CARATTERIZZAZIONE GEOTECNICA DI BASE

La caratterizzazione geotecnica dei terreni del versante fa riferimento all'ultima campagna effettuata, che risale a dicembre 2020 e marzo 2021. Dal complesso dei dati in possesso quindi si sono definiti quattro diversi materiali corrispondenti alle successioni sedimentarie, ovvero l'Unità 1 che rappresenta i Conglomerati di Campomarino, le sub-Unità 2a e 2b associate rispettivamente alla porzione sabbiosa superiore e alla porzione di sabbie limose inferiori delle Sabbie di Serracapriola e l'Unità 3, che rappresenta le argille marine rigide appartenenti alla formazione delle Argille di Montesecco.

A titolo di esempio, si riportano in Fig. 7 i risultati delle prove TRX-CIU eseguite sul campione C15i/B1, prelevato a 45.25 m ed afferente all'argilla dell'Unità 3. I risultati sperimentali sono illustrati in termini di percorsi tensionali e curve tensione deviatorica-deformazione assiale per i tre provini confezionati dallo stesso campione. Questi provini sono stati consolidati isotropicamente a tre diverse pressioni di confinamento p' pari a:

$p'_1 = 276.89$ kPa

$p'_2 = 549.44$ kPa

$p'_3 = 970$ kPa

I valori a rottura (di picco) dell'invariante deviatorico q sono rispettivamente:

$q_1 = 350.13$ kPa

$q_2 = 455.02$ kPa

$q_3 = 700.4$ kPa

Il comportamento del materiale è quello di un'argilla debolmente sabbiosa leggermente sovraconsolidata.

L'inviluppo a rottura per questo materiale è descritto dai parametri di resistenza efficaci di picco pari a:

$c' = 28.4$ kPa

$\phi' = 21°$

mentre i parametri post-picco determinati sono pari a:

$\phi' = 23°$

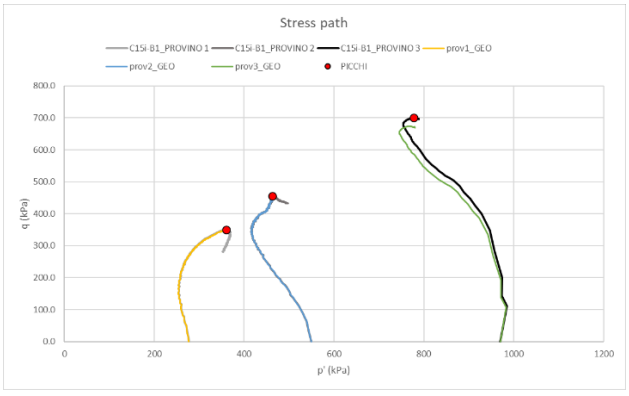


Fig. 7 - Percorsi tensionali nel piano p' - q delle prove TRX-CIU (Santaloia et al., in prep).

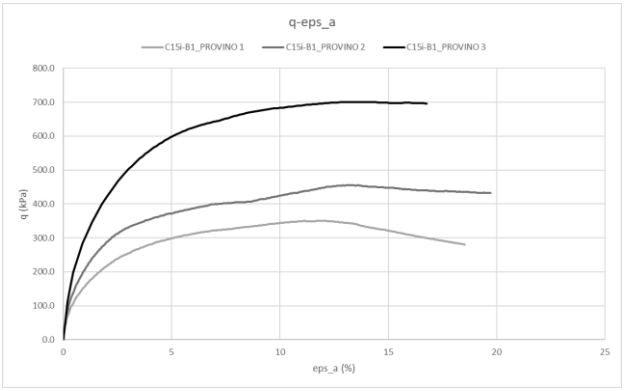


Fig. 8 Curve nel piano q - ε_a delle prove TRX-CIU (Santaloia et al., in prep).

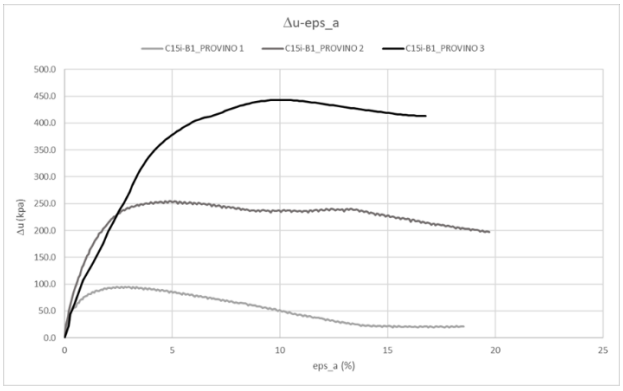


Fig. 9 Curve nel piano Δu - ε_a delle prove TRX-CIU (Santaloia et al., in prep).

Elaborando i risultati di tutte le prove triassiali e di taglio diretto eseguite su numerosi campioni prelevati delle diverse unità litologiche, è stato definito il modello geotecnico di sottosuolo, sintetizzato in Tab. 1.

Tab. 1 - Parametri di resistenza a taglio per le diverse unità del pendio di Chieuti

Unità litostratigrafica	**Valori medi dell'angolo di attrito**
Unità 1	$13.7° < \phi' < 30.8°$
Sub-Unità 2a	$26.3° < \phi' < 41°$
Sub-Unità 2b	$24° < \phi' < 34°$
Unità 3	$20° < \phi' < 27°$

5.4 CARATTERIZZAZIONE GEOTECNICA SISMICA

Per quanto concerne la caratterizzazione geotecnica dinamica, essa è stata eseguita mediante prove in situ e prove di laboratorio. Nello specifico sono state eseguite due prove Down-Hole (DH), in corrispondenza della cresta e a circa metà pendio, e 4 stendimenti sismici MASW 2D, lungo 4 sezioni che attraversano il centro abitato. I risultati delle prove geofisiche sono riportati nelle Fig. 10 (Down-Hole) e Fig. 11 (MASW). La prova DH C3di è stata eseguita all'interno del contesto urbano e raggiunge la profondità massima di 48 m; la prova DH D11di raggiunge invece la profondità di 70 m ed è ubicata all'interno del versante (di Lernia et al., 2023).

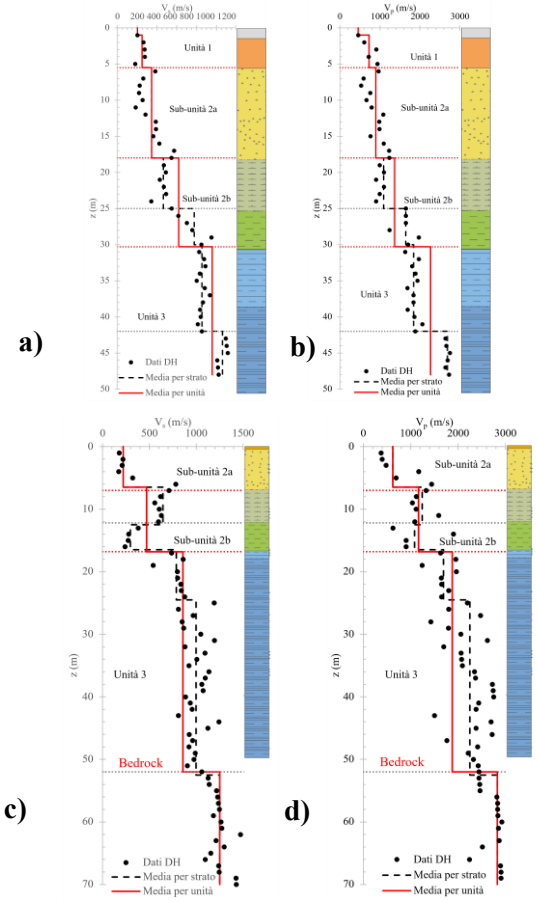


Fig. 10 Profili di V_S e V_P ottenuti dalle prove Down Hole: a) e b) sondaggio C3di, c) e d) sondaggio D11di (di Lernia et al., 2023).

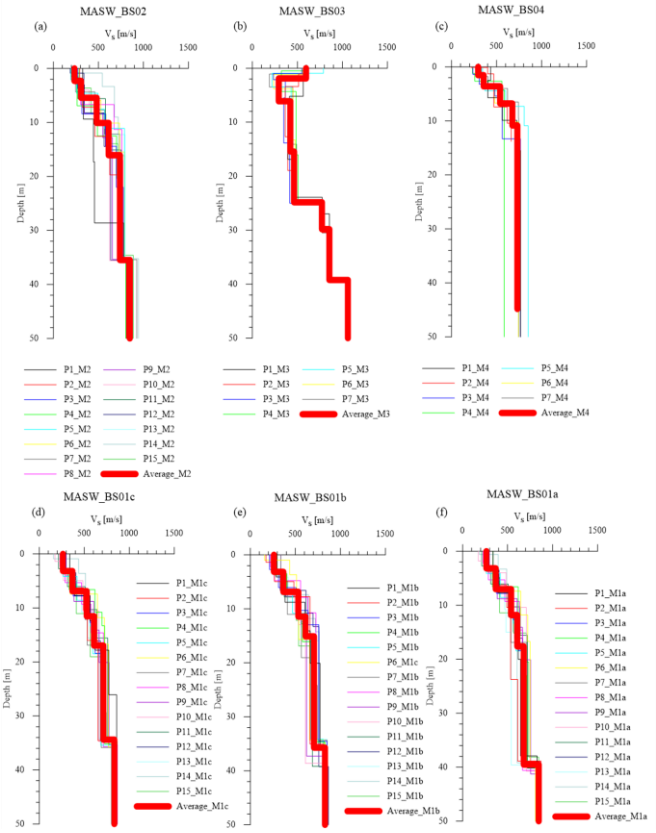


Fig. 11 Log delle prove in situ MASW 2D (di Lernia et al., 2023).

Lungo il foro effettuato in contesto urbano, il C3di Fig. 10, i primi 18 m, caratterizzati da Unità 1 e 2a, mostrano un graduale aumento di velocità con la profondità; mentre la transizione tra il secondo e il terzo strato, tra le Unità 2a e 2b, non è particolarmente marcata. Questo è probabilmente dovuto alla forte eterogeneità del deposito di terreno nella porzione superiore del versante, già riconosciuta da altri log stratigrafici nell'area.

Lungo il foro D11di Fig. 10, i profili di velocità V_S mostrano una tendenza complessivamente crescente con la profondità, eccetto tra 12,2 m e 16,8 m da piano campagna, dove si osserva un'inversione del profilo di velocità delle onde nella porzione di limo sabbioso dell'Unità 2b.

Questa inversione potrebbe essere associata sia alla presenza di inclusioni di materiali più rigidi nello strato superiore sia a processi di disturbo localizzati dove il materiale diventa meno rigido, probabilmente a causa della presenza di una banda di taglio a questa profondità.

Normalizzando i corrispondenti profili del modulo di rigidezza a taglio G0 per la pressione di confinamento p', si può osservare che i terreni nel foro D11di sono complessivamente più rigidi di quelli presenti lungo la verticale C3di (di Lernia et al., 2023).

Con riferimento al profilo stratigrafico D11di, il membro sabbioso dell'Unità 2b (tra 7 e 12,2 m da piano campagna) è caratterizzato da un rapporto G0/p' significativamente più alto del valore valutato lungo il profilo C3di nello stesso strato (tra 18 m e 25 m). Inoltre, la rigidezza normalizzata del membro limoso dell'Unità 2b lungo C3di (tra 25 e 30 m da piano campagna) risulta essere maggiore di quella ottenuta per lo stesso strato di terreno in D11di (tra 12,2 e 16,8 m da piano campagna).

L'analisi dettagliata dei profili di V_S evidenzia che la rigidezza tende ad aumentare con la profondità, partendo da valori di circa 200 m/s, riferibili allo strato di materiale antropico, e raggiungendo valori di circa 800-1000 m/s per gli strati più profondi, coerentemente con i profili di velocità delle onde S del DH. Sotto il primo strato, può essere rilevato un materiale caratterizzato da V_S uguale a 300-400 m/s, che è ascrivibile all'Unità 1, seguito da uno strato di 400-500 m/s di V_S, associato alla sotto-unità 2a. Sotto questo, si incontra un materiale con V_S di 600-700 m/s, riferibile all'Unità 2b, che sovrasta uno strato di terreno caratterizzato da velocità delle onde S superiore a 800 m/s, come previsto per lo strato più profondo dell'Unità 3.

In Tab. 2 è riportata la sintesi del modello geotecnico dinamico.

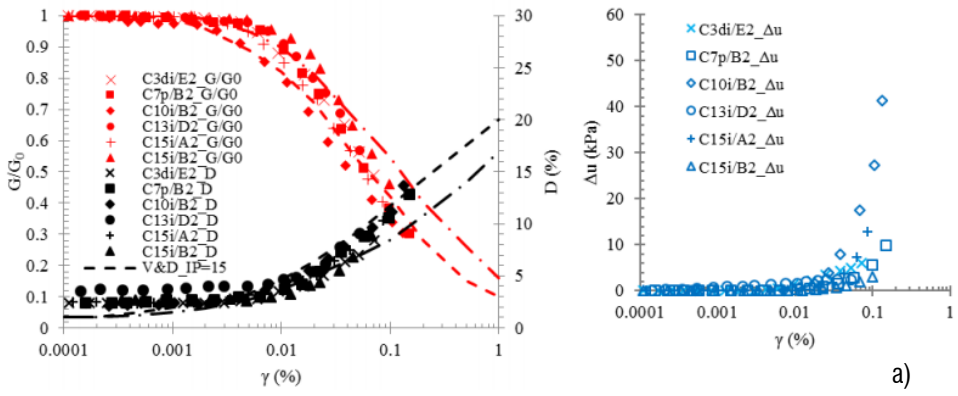
Tab. 2 - Modello geotecnico dinamico per il pendio di Chieuti

Unità litologica	**V_S [m/s]**
Riporto antropico	200
Unità 1	317
Unità 2a	491
Unità 2b	627
Unità 3	871
Bedrock	1250

In funzione dei risultati delle prove DH in situ, si è deciso di considerare come bedrock sismico lo strato con $V_S > 1200$ m/s. Quindi risulta una profondità di 52 m da piano campagna in riferimento al sondaggio D11, mentre ci si riferisce ad una profondità di 40 m per il sondaggio C3di.

Specificatamente, il down-hole D11di ha rilevato un materiale con V_S maggiore di 1200 m/s a una profondità di 52 m dal piano campagna, mentre il test sismico ha identificato una velocità delle onde di taglio di circa 1000 m/s a una profondità di 40 m sotto il centro urbano lungo il C3di. Così, il sottosuolo sotto la cittadina è molto probabilmente caratterizzato da una V_S maggiore di 1000 m/s ad una profondità maggiore di 40 m. Pertanto, la sommità del bedrock sismico è stata definita parallela alla superficie inclinata del terreno a una profondità di 50 m dal piano campagna.

La risposta ciclica dei terreni afferenti alle diverse unità litologiche è stata studiata mediante prove in colonna risonante, eseguite su 8 campioni prelevati da 7 verticali di sondaggio. In Fig. 12 si riportano i risultati delle prove in colonna risonante in termini di curve di decadimento del modulo di rigidezza a taglio G normalizzato rispetto a G_{max} e del fattore di smorzamento con la deformazione di taglio.



a)

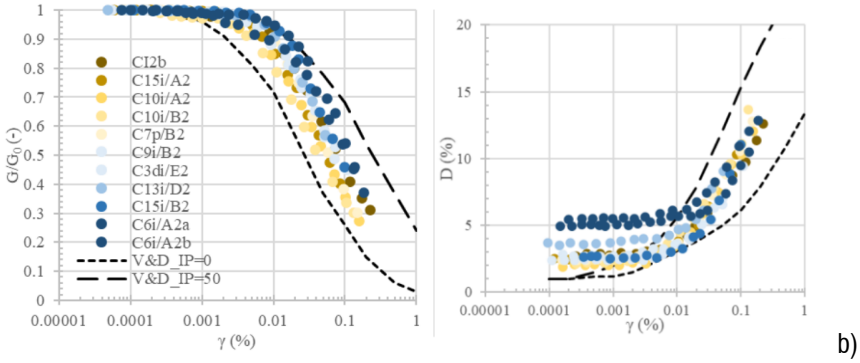


Fig. 12 a) Risultati di prove in colonna risonante per l'Unità 3, b) risultati delle prove in colonna risonante eseguite su diversi campioni (di Lernia et al., 2023).

6. DESCRIZIONE DEI MODELLI COSTITUTIVI

Nel presente capitolo verranno presentati i due modelli costitutivi utilizzati nelle analisi dinamiche con i due codici di calcolo PLAXIS e OPENSEES. Per tali modelli verranno presentati i caratteri salienti e le formulazioni alla loro base, in modo da farne comprendere il funzionamento sotto carichi monotoni e ciclici. Infine, per quanto possibile, si citeranno alcuni brevi aspetti dell'implementazione numerica.

Per comodità di trattazione, si riportano di seguito le ben note condizioni principali che governano la maggior parte dei modelli costitutivi elasto-plastici:

$\varepsilon_{iJ} = \varepsilon_{iJ}^{e} + \varepsilon_{iJ}^{p}$ (composizione del tensore di deformazione totale) (3)

$\sigma_{iJ} = D_{ijkl}\varepsilon_{kl}^{e}$ (modello costitutivo elastico) (4)

$d\varepsilon_{ij}^{p} = d\lambda \frac{\partial g(\sigma_{ij},q)}{\partial \sigma_{ij}}$ (legge di flusso per il tensore di deformazione plastico) (5)

$dq^{*} = d\lambda h^{*}\left(\sigma_{ij}, q^{*}\right)$ (legge di incrudimento) (6)

Per esprimere il moltiplicatore plastico $d\lambda$ si riportano le condizioni di Kuhn-Tucker:

$$f\left(\sigma_{ij}, q^{*}\right) \leq 0 \quad (7)$$

$$d\lambda \geq 0 \quad (8)$$

$$f\, d\lambda = 0 \quad (9)$$

$$df = \frac{\partial f}{\partial \sigma_{ij}} d\sigma_{ij} + \frac{\partial f}{\partial q^{*}} dq^{*} = 0 \text{ (condizione di consistenza)} \quad (10)$$

per cui:

$$d\lambda = \frac{\frac{\partial f}{\partial \sigma_{ij}} D_{ijkl}\, d\varepsilon_{kl}}{\frac{\partial f}{\partial \sigma_{ij}} D_{ijkl} \frac{\partial g}{\partial \sigma_{kl}} + h} \quad (11)$$

$$h = -\frac{\partial f}{\partial q^{*}} h^{*} \quad (12)$$

6.1 Il MODELLO HARDENING SOIL WITH SMALL STRAIN STIFFNESS

Il **modello Hardening Soil with Small Strain Stiffness (HSsmall)** (Benz et al., 2009) rappresenta una estensione del modello elasto-plastico **Hardening soil** (**HS**)

(Schanz et al., 1999), aggiungendo la possibilità di simulare la risposta ciclica dei terreni, attraverso l'introduzione delle curve di decadimento del modulo di rigidezza a taglio e dello smorzamento.

Nello specifico, il modello HS è un modello non lineare elasto-plastico ad incrudimento isotropo, sviluppato da Schanz et al. (1999) per riprodurre la risposta di diverse tipologie di terreno. Esso è caratterizzato dalla presenza di due superfici di snervamento: una superficie di incrudimento a taglio (shear hardening yield surface) ed una di incrudimento a compressione (cap yield surface), che delimitano la regione elastica. La prima è funzione delle deformazioni deviatoriche plastiche e può espandersi fino al criterio di resistenza di Mohr-Coulomb; la seconda è governata dalle deformazioni volumetriche plastiche.

L'effetto principale dell'Hardening Soil è quello di rappresentare il decadimento del modulo di rigidezza a taglio G quando il materiale è sottoposto a carico deviatorico.

Infatti, quando il terreno vergine viene sottoposto a carico deviatorico mostra un abbassamento della rigidezza insieme allo sviluppo di deformazioni plastiche irreversibili. Nel caso speciale di una prova triassiale drenata la relazione tra le deformazioni assiali e le tensioni deviatoriche è ben approssimata da una iperbole.

Il modello inoltre include la dipendenza dei parametri di rigidezza da σ'_1 e dall'esponente m. Pertanto, vi è la possibilità di applicare una rigidezza del materiale variabile con la profondità (ipotesi più verosimile rispetto al semplice Mohr-Coulomb).

Passando alla formulazione matematica del caso generico nello spazio delle tensioni principali, la funzione di snervamento che regola il comportamento a sforzi deviatorici è la seguente (si consideri una tensione σ generica):

$$f_{12}^{s} = \frac{2q_a}{E_i}\frac{(\sigma_1-\sigma_2)}{q_a-(\sigma_1-\sigma_2)} - \frac{2(\sigma_1-\sigma_2)}{E_{ur}} - \gamma^{ps} \quad (13)$$

$$f_{13}^{s} = \frac{2q_a}{E_i}\frac{(\sigma_1-\sigma_3)}{q_a-(\sigma_1-\sigma_3)} - \frac{2(\sigma_1-\sigma_3)}{E_{ur}} - \gamma^{ps} \quad (14)$$

La funzione potenziale plastico invece è definita come segue (Benz, 2006):

$$g_{12}^{s} = \frac{(\sigma_1-\sigma_2)}{2} - \frac{\sigma_1+\sigma_2}{2} sin\,\psi_m \quad (15)$$

$$g_{13}^{s} = \frac{(\sigma_1-\sigma_3)}{2} - \frac{\sigma_1+\sigma_3}{2} sin\,\psi_m \quad (16)$$

Con:

$$sin\,\psi_m = \frac{sin\,\varphi_m - sin\,\varphi_{cs}}{1 - sin\,\varphi_m\,sin\,\varphi_{cs}} \geq 0 \quad (17)$$

$\psi_m = angolo\ di\ dilatanza$

$\varphi_{cs} = angolo\ d'attrito\ di\ stato\ critico$

$\varphi_m = angolo\ d'attrito\ mobilizzato$

Con:

$$\sin\varphi_m = \frac{(\sigma_1 - \sigma_3)}{\sigma_1 + \sigma_3 + 2c\cot\varphi} \quad (18)$$

I moduli di rigidezza E_i ed E_{ur} assumono la seguente formulazione (Benz, 2006):

$$E_i = E_i^{ref}\left(\frac{\sigma_3 + c\,cot\,\varphi}{p^{ref} + c\,cot\,\varphi}\right)^m \quad (19)$$

$$E_{ur} = E_{ur}^{ref}\left(\frac{\sigma_3 + c\,cot\,\varphi}{p^{ref} + c\,cot\,\varphi}\right)^m \quad (20)$$

E_i^{ref}, E_{ur}^{ref} sono i moduli di rigidezza di riferimento misurati alla pressione di riferimento p^{ref}, mentre l'esponente m indica la legge di potenza.

La tensione minima σ_3 è indice dello stato di sforzo attuale agente nel materiale.

Si definisce ora la superficie che delimita l'estensione delle superfici "a taglio". Tale superficie si dispone come una cupola con concavità all'interno del dominio (condizione di Khun-Taker) e la sua equazione è (Benz, 2006):

$$f^c = \frac{\tilde{q}^2}{\alpha^2} - p^2 - p_p^2 \quad (21)$$

dove p è il tensore isotropo, α è una costante interna del materiale che controlla la curvatura della cupola nel piano p-q, p_p è una variabile che controlla la tensione di preconsolidazione invece $\tilde{q}$ è una variabile di tensionale così definita:

$$\tilde{q} = \sigma_1 + (\delta^{-1} - 1)\sigma_2 - \delta^{-1}\sigma_3 \quad (22)$$

$$\delta = \frac{3 - \sin\varphi}{3 + \sin\varphi} \quad (23)$$

Questa variabile serve per adattare le due superfici come in Fig. 13:

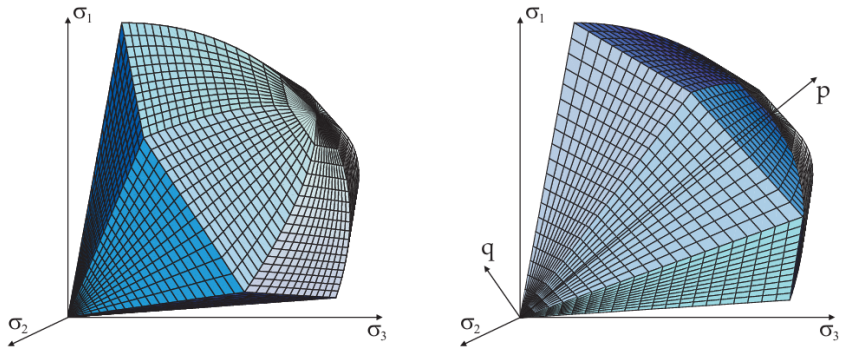


Fig. 13 Rappresentazione nello spazio delle tensioni principali del dominio elastico del modello HS (Brinkgreve et al., 2016).

Per questa superficie la legge di flusso scelta è associata.

Pertanto, si contano globalmente due variabili di stato per le funzioni di incrudimento e per le funzioni potenziale così definite:

$$d\gamma^{ps} = d\lambda^{s} h_{\gamma^{ps}} \quad \text{con:} \quad h_{\gamma^{ps}} = \frac{\partial g}{\partial \sigma_1} - \frac{\partial g}{\partial \sigma_2} - \frac{\partial g}{\partial \sigma_3} = 1 \tag{24}$$

$$d_{P_p} = d\lambda^{c} h_{p_p} \qquad \text{con:} \quad h_{P_P} = 2H \left(\frac{\sigma_3 + c\cot\varphi}{p^{ref} + c\cot\varphi} \right)^{m} p \tag{25}$$

$$d_{P_p} = H \left(\frac{\sigma_3 + c\cot\varphi}{p^{ref} + c\cot\varphi} \right)^{m} d\varepsilon_v^{p} \tag{26}$$

In riferimento alla decomposizione del tensore delle deformazioni volumetriche nelle rispettive parti elastiche e plastiche, H può essere riscritto in funzione delle compressibilità K_s (riguardo la fase di scarico-ricarico) e K_c in relazione alla fase di primo carico:

$$H = \frac{K_S K_C}{K_S - K_C} = \frac{1}{\frac{K_S}{K_C} - 1} K_S \quad \text{con:} \quad K_s = \frac{E_{ur}^{Ref}}{3(1-2v)} \tag{27}$$

Infine, si riporta la definizione del tensore di elasticità lineare:

$$D_{iJkl} = \frac{E_{ur}}{(1+v_{ur})(1-2v_{ur})} + \left((1-2v_{ur})\delta_{ik}\delta_{Jl} + v_{ur}\delta_{jk}\delta_{Jl} \right) \tag{28}$$

Il modello **HSsmall** introduce nel modello HS l'approccio para-elastico, che può essere combinato con formulazioni elasto-plastiche, per descrivere la risposta ciclica dei terreni.

La variazione della rigidezza tangenziale secante, quindi, è introdotta attraverso una formulazione che lega il modulo di rigidezza a taglio secante alla deformazione di taglio secondo una legge iperbolica alla Hardin and Drnevich (Hardin & Drnevich, 1972):

$$\frac{G_S}{G_0} = \frac{1}{1+0.385\left|\frac{\gamma_{Hist}}{\gamma_{0.7}}\right|} \tag{29}$$

In cui $\gamma_{0.7}$ è la deformazione di taglio in corrispondenza della quale il modulo di rigidezza a taglio secante è pari al 70% del suo valore iniziale:

$$G_s = 0.772\, G_0 \tag{30}$$

Per rappresentare il comportamento isteretico del materiale lungo percorsi di carico, scarico e ricarico nel modello è stato integrato l'approccio di Masing (1926). Pertanto, il modulo di rigidezza a taglio iniziale in scarico è uguale a quello iniziale nel percorso di primo carico e la forma del percorso di scarico e ricarico è simile a quella di primo carico ma raddoppiata in misura ovvero:

$$\gamma_{0.7\ reloading} = 2\,\gamma_{0.7\ primo\ carico} \tag{31}$$

Come per HS, in HSsmall è implementata la dipendenza del modulo di rigidezza a taglio iniziale G_0 dalla σ'_3 oltre che dall'esponente m e dall'angolo di attrito φ':

$$G_0 = G_0^{ref}\left(\frac{c\cos(\varphi)-\sigma_3'\sin(\varphi)}{c\cos(\varphi)+p^{ref}\sin(\varphi)}\right)^m \tag{32}$$

Quando la direzione di carico si inverte, la rigidezza guadagna il massimo valore recuperabile, per poi decadere nuovamente quando si applica un carico nella nuova configurazione.

In relazione a questo fenomeno (Benz, 2006) è necessario introdurre nella formulazione matematica una quantità che tenga memoria della storia di deformazione del materiale.

In primo luogo, la storia di deformazione deve essere considerata nello spazio generale delle deformazioni al fine di includere tutto il comportamento 3D.

Passando dallo spazio generale a quello delle deformazioni principali, l'attuale incremento di deformazione è rappresentato come una superfice del II ordine.

Al fine di evitare singolarità a questa superficie si aggiunge una deformazione volumetrica unitaria che nello spazio definito ha la forma di una sfera.

Quindi si definisce un tensore H_{kl} che memorizza la storia di deformazione deviatorica.

Il tasso di deformazione deviatorica attuale è rappresentato con $\dot{e_{kl}}$ risolvendo il problema agli autovalori:

$$\left(\dot{e}_{kl} - \lambda^{(m)}\delta_{kl}\right) - S_i^{(m)} = 0 \tag{33}$$

Quindi l'incremento di deformazione deviatorica trasformato:

$$\dot{e}_{kl} = S_{km}\,\dot{e}_{mn}S_{nl} \tag{34}$$

La storia di deformazione trasformata sarà:

$$H_{kl} = S_{km}H_{mn}S_{nl} \tag{35}$$

Espandendo la storia di deformazione per una quantità unitaria di deformazione volumetrica si rende il tensore H definito positivo simmetrico. La deviazione dalla forma di sfera volumetrica unitaria nello spazio delle deformazioni principali è data dalla diagonale di H_{kl}^{i}.

Ogni direzione principale risponde singolarmente dell'inversione di carico (per cui non vi è accoppiamento tra le direzioni principali).

Per azzerare la storia di carico della i-esima direzione, durante l'inversione di carico si assume una matrice di trasformazione diagonale T_{kl} definita come:

$$T_{11} = \frac{1}{\sqrt{H_{11}+1}}\left(1 + u\left(\lambda^{(1)}H_{11}\right)\left(\sqrt{H_{11}+1} - 1\right)\right) \tag{36}$$

$$T_{22} = \frac{1}{\sqrt{H_{22}+1}}\left(1 + u\left(\lambda^{(2)}H_{22}\right)\left(\sqrt{H_{22}+1} - 1\right)\right) \tag{37}$$

$$T_{33} = \frac{1}{\sqrt{H_{33}+1}}\left(1 + u\left(\lambda^{(3)}H_{33}\right)\left(\sqrt{H_{33}+1} - 1\right)\right) \tag{38}$$

Dove u(x) è uguale ad 1 per $x > 0$ mentre è nulla per $x < 0$

Quindi la storia di deformazione aggiornata H_{kl}^{*} è calcolata come:

$$H_{kl}^{*} = T_{km}(H_{mn} - \delta_{mn})T_{nl} - \delta_{kl} \tag{39}$$

Dalla storia di deformazione aggiornata si definisce una misura scalare della deformazione a taglio (Benz, 2006):

$$\gamma_{hist} = \sqrt{3}\frac{\|\boldsymbol{H}\Delta e\|}{\|\Delta e\|} \quad (40)$$

In cui:

H è un tensore simmetrico che rappresenta la storia di carico deviatorico del materiale

Δe è l'attuale incremento di deformazione deviatorica.

$\boldsymbol{\gamma_{Hist}}$ è la proiezione della storia di deformazione lungo l'attuale direzione di carico. Questa può essere usata per definire la rigidezza del materiale isotropo in forma incrementale. In Fig. 17 si riporta l'algoritmo di calcolo per la definizione della rigidezza tangenziale, estratto da Benz (2006).

La formulazione per il modulo di rigidezza a taglio tangente G_t, derivata dalla formula (27), è riportata nell'Eq. (41):

$$G_t = \frac{G_0}{(1+0.385\frac{\gamma}{\gamma_{0.7}})^2} \quad (41)$$

Il modulo di rigidezza a taglio di scarico e ricarico assume l'espressione:

$$G_{ur} = \frac{E_{ur}}{2(1+\nu_{ur})} \quad (42)$$

Tale modulo definisce il limite inferiore del valore della rigidezza a taglio tangente G_t ed è funzione del Modulo di Young di scarico e ricarico e del modulo di Poisson assegnati dall'utente.

Nel diagramma G-γ l'intersezione della curva di decadimento del modulo di rigidezza tangenziale G_t con il limite inferiore imposto da G_{ur} è definito all'ascissa $\gamma_{cut\text{-}off}$:

$$\gamma_{cut-off} = \frac{1}{0.385}\left(\sqrt{\frac{G_0}{G_{ur}}} - 1\right)\gamma^{0.7} \quad (43)$$

In Fig. 14 e Fig. 15 sono riportate un esempio di curva di decadimento del modulo di rigidezza a taglio tangente e secante derivante dai parametri di input e una risposta meccanica ciclica tipica di un comportamento elasto-plastico alla Masing con un dominio di isteresi allargato. Si notino in particolare i tratti a rigidezza G_0 e G_{ur} (Brinkgreve et al., 2016):

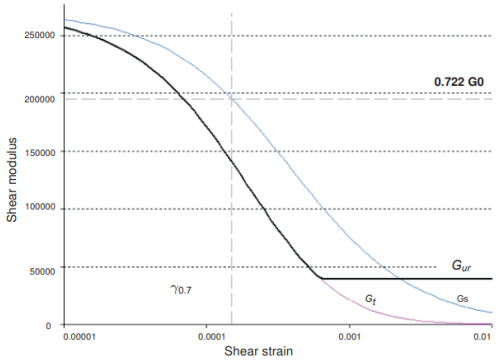


Fig. 14 Esempio di curva di decadimento del modulo tangente e secante (Brinkgreve et al., 2016).

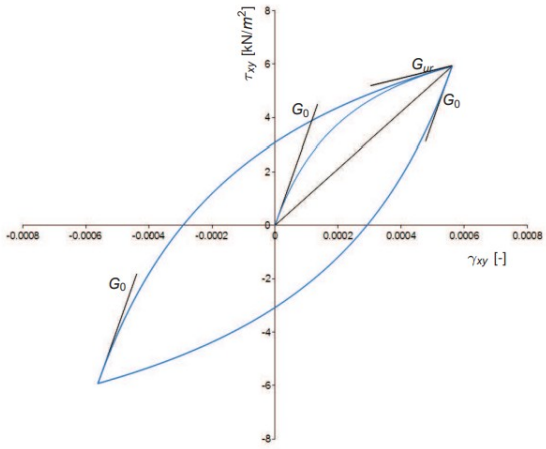


Fig. 15 Parametri di rigidezza in una prova di taglio ciclico (Brinkgreve et al., 2016).

Pertanto, i parametri che l'utente è chiamato a dare in input sono:

- E_{50}^{ref} è il modulo di Young secante di riferimento, preso al 50% del valore di resistenza in una prova triassiale drenata
- E_{oed}^{ref} è il modulo di Young edometrico di riferimento
- E_{ur}^{ref} è il modulo di Young di scarico e ricarico
- ν_{ur} è il modulo di Poisson di scarico e ricarico
- G_0^{ref} è il modulo di rigidezza a taglio di riferimento
- $\gamma_{0.7}$ è il valore dello scorrimento preso in corrispondenza del punto avente 0.722 G_0
- m è la potenza per la definizione della variazione del modulo della rigidezza a taglio con la profondità
- c' coesione efficace

- ϕ' angolo d'attrito efficace

Dagli studi di Alpan (1970) si definisce un rapporto tra il modulo di rigidezza a taglio G_0 e il modulo di rigidezza al taglio di scarico e ricarico G_{ur} secondo il diagramma in Fig. 16:

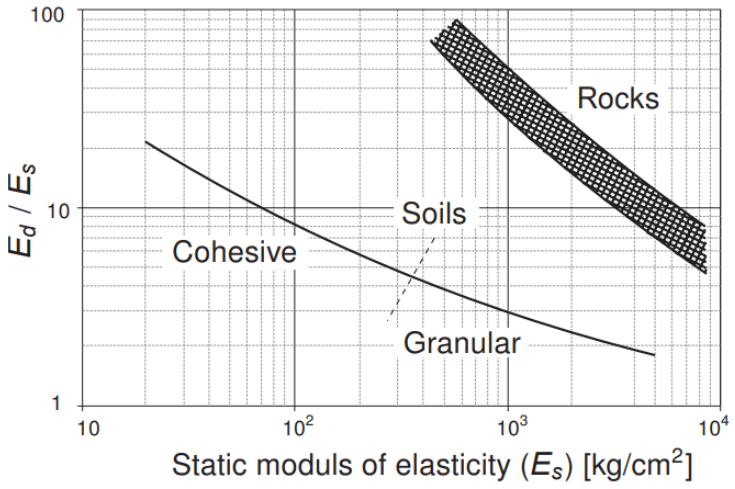


Fig. 16 Relazione tra $E_{dyn} \approx E_0$ ed $E_s \approx E_{ur}$ (Brinkgreve et al., 2016).

Pertanto, per un prima stima dei valori, definito un valore di G_0^{ref} dal valore del rapporto ottenuto incrociando le informazioni sull'abaco in Fig. 16 si ottiene G_{ur}^{ref}. Da questo E_{50}^{ref} è consigliato assumerlo pari alla metà di E_{ur}^{ref} e assumere E_{oed}^{ref} pari a E_{50}^{ref}.

1. Initalize strain history. Default setting is $H_{kl}^0 = 0$.

2. Adapt G_0^{ref} and $\gamma_{0.7}^{\text{ref}}$ to the actual material state (optionally):
   $G_0 = f(G_0^{\text{ref}}, p, e, ...)$ and $\gamma_{0.7} = f(\gamma_{0.7}^{\text{ref}}, p, e, ...)$

3. Calculate shear strain γ_{Hist}:

   a) Transform H_{kl}^i to the strain increment's Eigensystem:
   $\underline{H}_{kl}^i = S_{km} H_{mn}^i S_{nl}$ where $(\Delta e_{kl}^{i+1} - \lambda^{(m)} \delta_{kl}) S_l^{(m)} = 0$

   b) Calculate monotonic shear strain at time i:
   $\gamma_{Hist}^i = \sqrt{3} \frac{\|\underline{\Delta e}_{km}^{i+1} \underline{H}_{ml}^i\|}{\|\underline{\Delta e}_{kl}^{i+1}\|}$

   c) Calculate transformation matrix $\underline{T}_{kl}^i$:

$$\begin{aligned}
\underline{T}_{11} &= \frac{1}{\sqrt{H_{11}+1}} \left(1 + u(\lambda^{(1)} H_{11})(\sqrt{H_{11}+1} - 1)\right) \\
\underline{T}_{22} &= \frac{1}{\sqrt{H_{22}+1}} \left(1 + u(\lambda^{(2)} H_{22})(\sqrt{H_{22}+1} - 1)\right) \\
\underline{T}_{33} &= \frac{1}{\sqrt{H_{33}+1}} \left(1 + u(\lambda^{(3)} H_{33})(\sqrt{H_{33}+1} - 1)\right) \\
\underline{T}_{ij} &= 0 \text{ for } i \neq j
\end{aligned}$$

   where $u(x) = \begin{cases} 0 & for \quad x < 0 \\ 1 & for \quad x \geq 0 \end{cases}$

   d) Update strain history:
   $\underline{H}_{kl}^{i+1} = \underline{T}_{km}^i (\underline{H}_{mn}^i + \delta_{mn}) \underline{T}_{nl}^i + \underline{\Delta e}_{kl}^{i+1} - \delta_{kl}$

   e) Calculate monotonic shear strain at time $i+1$:
   $\gamma_{Hist}^{i+1} = \sqrt{3} \frac{\|\underline{\Delta e}_{km}^{i+1} \underline{H}_{ml}^{i+1}\|}{\|\underline{\Delta e}_{kl}^{i+1}\|}$

4. Calculate tangent stiffness G^{i+1} (unloading/ reloading):
   $G^{i+1} = \frac{G_0}{\gamma_{Hist}^{i+1} - \gamma_{Hist}^i} \left(\frac{\gamma_{Hist}^{i+1}}{1 + \frac{0.385 \gamma_{Hist}^{i+1}}{2\gamma_{0.7}}} - \frac{\gamma_{Hist}^i}{1 + \frac{0.385 \gamma_{Hist}^i}{2\gamma_{0.7}}} \right)$

5. IF $G^{i+1} < \text{Min}\{G^1, G^2, ..., G^i\}$ THEN (primary loading):

   a) Set $\gamma_{0.7} = \gamma_{0.7}/2$

   b) Recalculate stiffness (4)

   END IF

Fig. 17 Algoritmo di calcolo per la definizione della rigidezza tangenziale del modello HS.

6.2 IL MODELLO PRESSURE INDEPENDENT MULTI YIELD

I primi modelli multi-superfice sono stati proposti da Mroz et al. (1979, 1981) e Iwan (1967). In particolare, fino ad allora, questi modelli erano sviluppati per la simulazione dell'effetto isteretico di Baushinger nei metalli. Più tardi sono stati applicati ai terreni con gli studi di Prevost (1977) e Mroz et al. (1979, 1981).

La caratteristica più importante di questi modelli è la capacità di cogliere la storia tensionale recente nelle variabili interne (backstress). I modelli di tipo Small Strain Stiffness basati sulla teoria multi-superficie associano la rigidezza a piccole deformazioni con la superficie di snervamento più interna.

Definendo un appropriato set di leggi di incrudimento per tutte le superfici di snervamento, tali modelli possono dare una previsione realistica anche a deformazioni tangenziali elevate.

Il Pressure Independent Multi Yield (PIMY) model parte da Prevost (1977) il quale ha applicato il concetto di plasticità multi-superficie di Mroz per modellare il comportamento tenso-deformativo dei terreni in relazione al caso del comportamento non drenato delle argille.

In particolare è stata utilizzata la superficie di snervamento di Von Mises nello spazio generale delle tensioni totali sia come criterio di rottura che come superficie di snervamento i-esima interna alla superficie di rottura.

Nell'analisi non drenata applicata si divide la tensione e la deformazione in componenti volumetriche e deviatoriche disaccoppiate, che si riportano in forma tensoriale sintetica come:

$s_{ij} = \sigma_{ij} - \frac{1}{3}\sigma_{kk}\delta_{ij}$ (tensore degli stress deviatorici) (44)

$e_{ij} = \varepsilon_{ij} - \frac{1}{3}\varepsilon_{kk}\delta_{ij}$ (tensore delle deformazioni deviatoriche) (45)

δ_{ij} è il delta di Kronecker

$\sigma_{kk} = tr(\sigma) = \sigma_{11} + \sigma_{22} + \sigma_{33}$ (46)

Il comportamento elastico è modellato come:

$d\varepsilon_{kk} = \frac{d\sigma_{kk}}{3K}$ (47)

$$de_{kk} = \frac{ds_{ij}}{2G} \tag{48}$$

In cui K e G sono le rigidezze a compressione e taglio; inoltre, si richiama l'ipotesi di composizione della deformazione totale come somma di una parte puramente elastica e della complementare puramente plastica.

Richiamando il concetto di *campo a modulo plastico constante* di Mroz (1979,1981), questo è definito nello spazio delle tensioni da un insieme di superfici di snervamento annidate $f_1,\ldots,f_p$, come evidenziato nella Fig. 18:

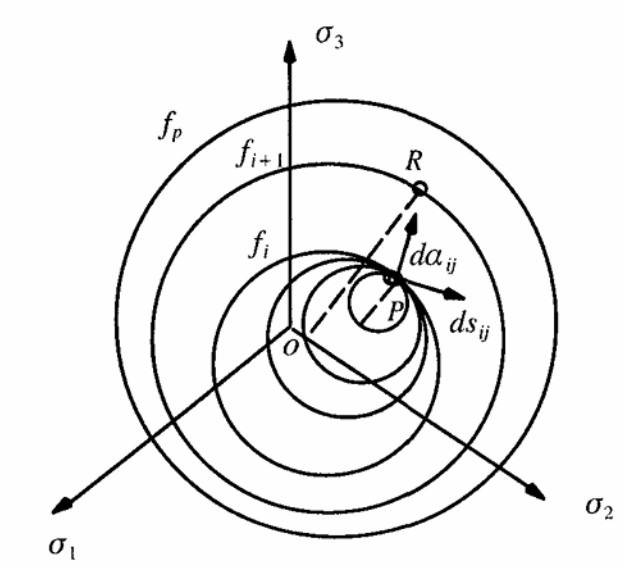


Fig. 18 Rappresentazione delle superfici di snervamento nel piano deviatorico (Prevost, 1977).

Come si osserva nel caso non drenato delle argille, durante lo snervamento dovuto ad un carico non drenato, il comportamento è indipendente dal tensore isotropo e quindi il fenomeno può essere considerato solo dipendente dal tensore deviatorico s_{ij} che quindi è quello che compare nella funzione di snervamento (Prevost, 1977, 1978):

$$f_i = \left\{\frac{3}{2}\left[s_{ij} - \alpha_{ij}^{(i)}\right]\left[s_{ij} - \alpha_{ij}^{(i)}\right]\right\}^{\frac{1}{2}} - k^{(i)} = 0 \tag{49}$$

dove $i = 1,2, \dots, p$ e $\alpha_{ij}^{(i)}$ rappresentano le coordinate del centro della superficie di snervamento i-esima f_i nello spazio delle tensioni deviatoriche.

Per il caso particolare di $\alpha_{ij}^{(i)} = 0$ la funzione diventa:

$f_i = \sqrt{3J_2} - k^{(i)} = 0$ (criterio di Von Mises) (50)

Dato che gli $\alpha_{ij}^{(i)}$ non sono tutti nulli, lo snervamento dei terreni è generalmente anisotropo.

Si definisce anche un modulo plastico $K_p^{(i)}$ associato ad ogni superficie di snervamento.

Inoltre, si assume che il potenziale plastico coincide con la funzione di snervamento; pertanto la legge di flusso è associata.

L'ultima superficie che contiene tutte le altre, f_p, rappresenta il criterio di rottura; quindi gli stati fuori tale superficie non esistono.

La legge di traslazione considerata nel modello è quella di Mroz, secondo cui tutte le superfici possono traslare nello spazio delle tensioni senza cambiare forma o orientazione; inoltre ogni superficie può tangere la successiva e spingerla ma senza intersecarsi.

Nella Fig. 18 si vedono molteplici superfici al termine di un percorso di carico avere tutte il punto P in comune, quindi:

$$\frac{s_{ij}-\alpha_{ij}^{(1)}}{k^{(1)}} = \frac{s_{ij}-\alpha_{ij}^{(2)}}{k^{(2)}} = \dots = \frac{s_{ij}-\alpha_{ij}^{(i-1)}}{k^{(i-1)}} = \frac{s_{ij}-\alpha_{ij}^{(i)}}{k^{(i)}} \quad (51)$$

Il tasso di deformazione plastico è la somma della componente elastica e di quella plastica.

Quella plastica è assunta essere normale alla superficie di snervamento nel punto di tensione, ovvero (Prevost 1977):

$$ds_{ij} = 2G\, de_{ij} - \frac{3\left[2G-K_t^{(i)}\right]}{2}\frac{\left[s_{ij}-\alpha_{ij}^{(i)}\right]}{[k^{(i)}]^2}\,[s_{kl} - \alpha_{kl}^{(i)}]de_{kl} \quad (52)$$

in cui:

$$\frac{1}{K_p^{(i)}} = \frac{1}{K_t^{(i)}} - \frac{1}{2G} \quad (53)$$

Per la determinazione di K_t si rinvia a Prevost (1977).

Quando il punto giace sulla superficie di snervamento f_i e si muove verso la superficie che lo contiene, f_{i+1}, la sua traslazione istantanea è scritta come:

$$d\alpha_{ij}^{(i)} = d\mu\, \overline{PR_{ij}} \tag{54}$$

$$\overline{PR_{ij}} = \frac{k^{i+1}}{k^{(i)}} \left[s_{ij} - \alpha_{ij}^{(i)}\right] - \left[s_{ij} - \alpha_{ij}^{(i+1)}\right] \tag{55}$$

In cui $k^{(i)}$e k^{i+1} sono le misure istantanee delle superfici di snervamento f_i e f_{i+1} per un dato livello di deformazione plastica α. Ovvero le superfici possono espandersi o contrarsi indipendentemente dal loro traslare.

Il modello PIMY è un modello elasto-plastico in cui la plasticità si manifesta solo come risposta tenso-deformativa a sollecitazioni deviatoriche.

La risposta volumetrica è disaccoppiata da quella deviatorica ed è lineare elastica. Il modello è in grado di simulare sia la risposta ad azioni monotone che ad azioni cicliche; tuttavia, la resistenza a taglio non risente delle variazioni dello stato di confinamento.

Il tipo di plasticità implementata è quella basata sulla teoria dei modelli multi-superficie annidati con legge di flusso associata (J2) già descritta in dettaglio. Il modello è particolarmente adatto per simulare la risposta a taglio elasto-plastica non drenata di materiali argillosi.

La superficie di snervamento è del tipo: “Von Mises”:

$$f = \left\{\frac{3}{2}\,(\tau - \alpha) : (\tau - \alpha)\right\}^{\frac{1}{2}} - K = 0 \tag{56}$$

In cui:

τ è il tensore deviatorico

α è il tensore di back-stress e denota il centro della superficie di snervamento nello spazio deviatorico

K è la misura della superficie di snervamento ($\sqrt{(3/2)}$)

Il modello è in grado di simulare la curva di decadimento del modulo a taglio G in funzione dello scorrimento γ secondo la forma iperbolica:

$$\tau = \frac{G\gamma}{1 + \frac{\gamma}{\gamma_r}} \tag{57}$$

in cui:

$$\gamma_r = \gamma_{max}\tau_{max}/(G\gamma_{max} - \tau_{max}) \quad (58)$$

τ e γ sono lo sforzo e la deformazione ottaedrici

G è il modulo di deformazione a piccole deformazioni

τ_{max} è lo sforzo tangenziale di picco che corrisponde allo scorrimento di picco γ_{max}

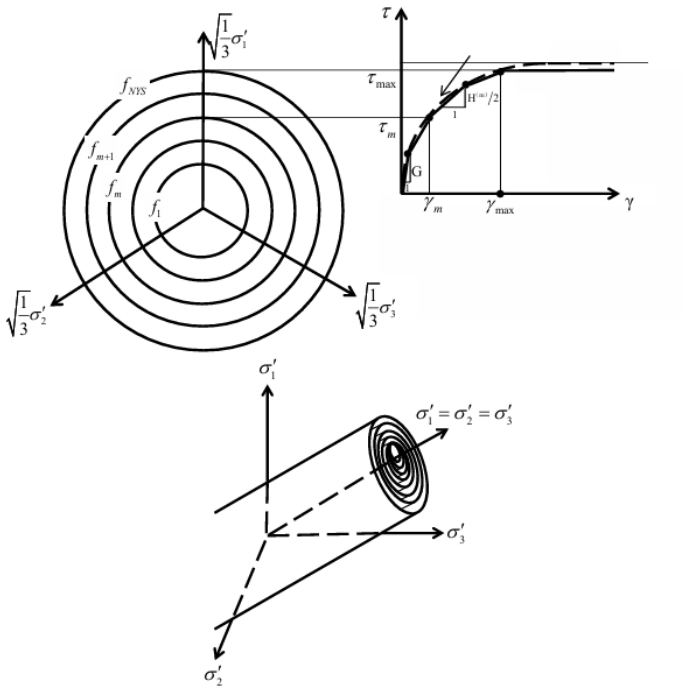


Fig. 19 Formulazione del modello PIMY (Qiu et al., 2019, Yang et al., 2008).

Nell'impostazione dei modelli multi-superficie, la curva dorsale è sostituita passo-passo secondo una approssimazione lineare, secondo cui un tratto di linea rappresenta il dominio di una superficie di snervamento (fi = 0) di misura Ki e caratterizzata da un modulo elasto-plastico tangenziale $H^{(i)}$.

$$H^{(i)} = 2\left(\frac{\tau_{i+1}-\tau_i}{\gamma_{i+1}-\gamma_i}\right) \quad (59)$$

Il modulo plastico tangenziale $H'^{(i)}$ è costante ed è definito come:

$$\frac{1}{H'^{(i)}} = \frac{1}{H^{(i)}} - \frac{1}{2G} \tag{60}$$

Tale modulo (Gu, 2011) è associato con ogni superficie di snervamento.

Il modulo plastico corrispondente alla superficie di snervamento (stato critico) più esterna è imposto pari a 0.

Per calcolare gli incrementi di deformazione plastica è utilizzata una legge di flusso associata.

Nello spazio degli sforzi deviatorici il vettore di incremento di deformazione plastica giace lungo la normale esterna alla superficie di snervamento nel punto di tensione.

Per chiarezza illustrativo riguardo il modello costitutivo si diporta in Fig. 20 un percorso in compressione ed estensione triassiale.

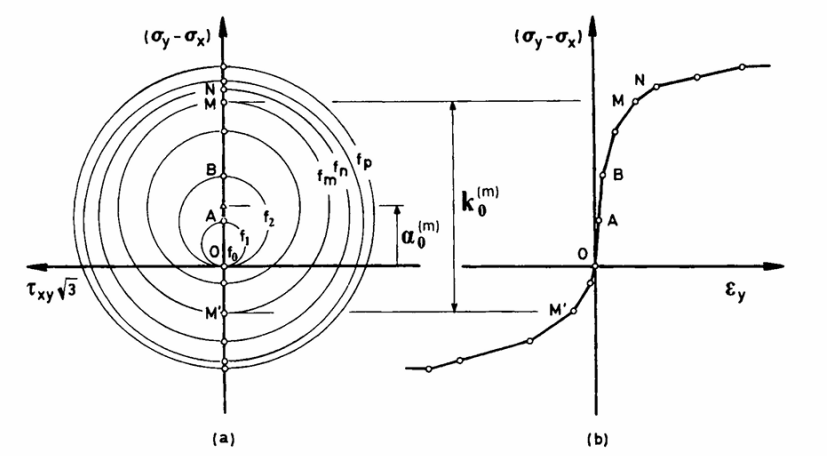


Fig. 20 Prova triassiale monotona in compressione ed estensione a) rappresentazione nel piano deviatorico, b) curva tensione-deformazione (Prevost, 1977).

In Fig. 21 invece si riporta una simulazione di prova triassiale ciclia.

Come è possibile osservare le n-superfici possono cambiare tanto di posizione quanto di diametro. Inoltre la superficie n tange la successiva n+1 in un punto senza intersecarsi.

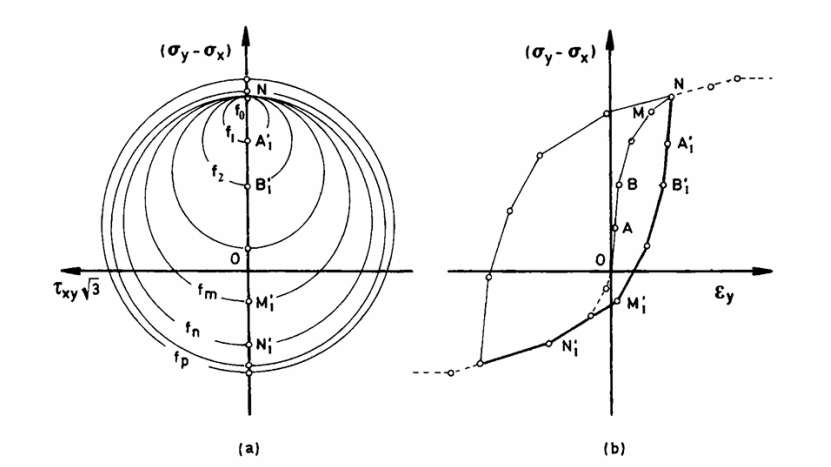


Fig. 21 Prova triassiale ciclica a) rappresentazione nel piano deviatorico, b) risposta ciclica (Prevost, 1977).

7. CALIBRAZIONE DEI MODELLI COSTITUTIVI

I modelli costitutivi sopra descritti sono stati calibrati con riferimento ai risultati sperimentali disponibili per il caso di studio di Chieuti. Essi, infatti, saranno adottati nelle simulazioni delle analisi di risposta sismica locale con i modelli 2D e 3D implementati nei codici PLAXIS e OPENSEES.

Nello specifico, sono stati presi in considerazione i risultati delle prove triassiali consolidate non drenate CIU-TRX eseguite sul provino 2 del campione C15i/B1 (Fig. 7) e delle prove in colonna risonante eseguite sul campione C15i/B2 (Fig. 12a), entrambi afferenti all'Unità 3 delle Argille di Montesecco prelevati alla profondità di circa 45.5 m. Il peso dell'unità di volume γ del terreno è pari a 19.7 kN/m^3, mentre il modulo di rigidezza a taglio iniziale G_0 è pari a 125 MPa, come desunto dai risultati delle prove Down-Hole.

7.1 CALIBRAZIONE DEL MODELLO PIMY

I parametri del modello costitutivo PIMY sono stati selezionati in modo da riprodurre con la migliore accuratezza possibile la risposta a taglio monotona, descritta dalla prova triassiale non drenata TRX-CIU, e la risposta a taglio ciclica, descritta dalle curve di decadimento dedotte dalla prova in colonna risonante.

La procedura di calibrazione dei parametri del modello PIMY è stata eseguita sviluppando dei modelli test, ovvero dei driver, che riproducono da un lato una prova triassiale e dall'altro una prova di taglio semplice ciclico. Nello specifico, i modelli che riproducono la prova triassiale sono costituiti da un singolo elemento finito "brick" a 8 nodi con formulazione u-p (brickUP) di lato 1 m (Zhaohui Y, 2000). La simulazione della prova triassiale è eseguita in due fasi: una prima fase di compressione isotropa, applicando una pressione di confinamento p' pari a 550 kPa, attraverso dei carichi esterni uniformemente distribuiti; una seconda fase puramente deviatorica realizzata applicando un carico verticale di intensità elevata. Il modello utilizzato per questa simulazione codificato in OPENSEES è allegato in **Appendice 1.** In **Appendice 2** è allegato il codice Python (van Rossum et al., 1995), per gestire i dati rinvenienti dall'output dell'analisi eseguita e per plottare i risultati.

La simulazione della prova di taglio semplice ciclico è stata eseguita utilizzando un cubo ad 8 nodi lineare con formulazione u-p (brickUP), con 8 punti di gauss indicati con "gp" seguito dal numero del punto uguale a quello del nodo adiacente. I nodi da 1 a 4 sono i nodi di base mentre quelli da 5 a 8 sono quelli in sommità. L'elemento è stato imposto come saturo.

Le condizioni al contorno prevedono il blocco della traslazione verticale e l'imposizione di un carico isotropo in fase statica. Nella fase dinamica si impedisce il moto dei nodi di base e di testa fuori dai piani orizzontali cui appartengono andando ad assegnare a questi piani due leggi di spostamento sinusoidale in controfase in modo da ottenere il cinematismo di taglio semplice voluto.

Particolare attenzione è stata imposta nel permettere solo spostamenti lungo la dimensione x azzerando ogni altro spostamento nelle altre dimensioni in modo da simulare uno stato piano (nel piano xz).

L'analisi, come citato in precedenza, prevede l'esecuzione di due fasi: una prima fase di applicazione del carico isotropo con p' = 500 kPa, che simula quello a cui è stato sottoposto il provino considerato in cella risonante; una seconda fase di taglio a controllo di deformazione, in cui si applica una distribuzione di spostamenti in sommità e alla base, variabile nel tempo con legge sinusoidale (Fig. 22, in legenda lo scorrimento ottenuto), sia alla base che in sommità, in modo da indurre un livello di deformazione di taglio γ prefissato. Come già descritto nelle condizioni al contorno, in questa fase è consentito solo lo scorrimento lungo la direzione x, mentre ogni spostamento in direzione y è vincolato.

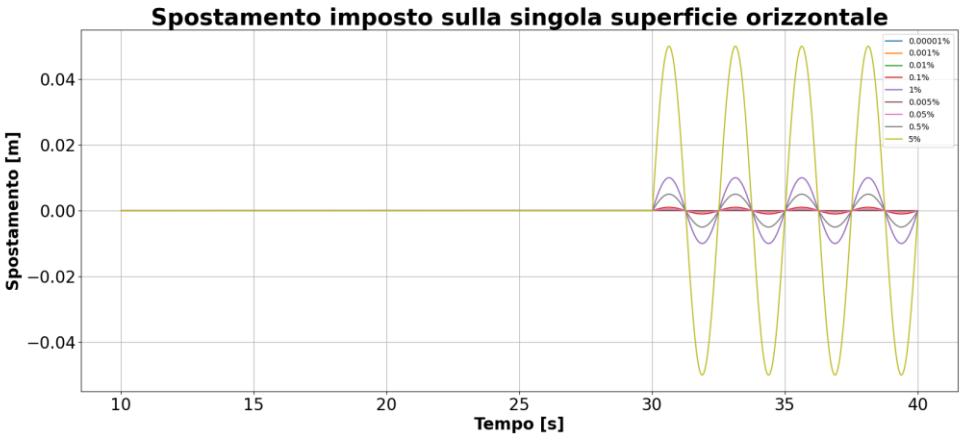


Fig. 22 Funzione di spostamento imposta alla base.

Con l'obiettivo di ricostruire la curva di decadimento del modulo di rigidezza a taglio, sono state eseguite 9 simulazioni di taglio imponendo storie di spostamenti tali da indurre i livelli deformativi γ pari a 0.00001%, 0.001%, 0.005%, 0.01%, 0.05%, 0.1%, 0.5%, 1%, 5%.

Il codice Python della prova ciclica di taglio semplice (CSS) è allegato in **Appendice 3**.

Il set di parametri che meglio approssimano la risposta a taglio monotona e ciclica del terreno sono riportati in Tab. 3.

Tab. 3 - Parametri del modello PIMY

SET 1		**SET 2**	
Parametro	**Valore**	**Parametro**	**Valore**
c_u	270 kPa	c_u	450 kPa
Φ_u	0°	Φ_u	0°
peck shear strain	0.1	peck shear strain	0.2
p'r	500 kPa	p'r	500 kPa
G_0^{ref}	125000 kPa	G_0^{ref}	50000 kPa
d	0	d	0

Particolarità di una tale scelta è la costanza della rigidezza a taglio G con la profondità in quanto $d = 0$; pertanto in relazione alla formulazione del PIMY anche p'r ha una influenza trascurabile sulla curva di decadimento, altra scelta è stata quella di

imporre l'angolo di attrito ϕ' pari a 0° per ottenere un comportamento più facilmente interpretabile. Il riferimento, nel caso di angolo di attrito assegnato, è comunque ad una resistenza non drenata:

$$\tau_f = \frac{2\sqrt{2}sin\varphi}{3-sin\varphi}\ p + \frac{2\sqrt{2}}{3}\ c_u \qquad \text{(Qiu et al., 2019)} \qquad (61)$$

Tale formulazione deriva da quella già ampiamente nota di stato critico per cui si considerano le relazioni:

$q = M\ p$ (equazione della retta di stato critico nel piano p-q) (62)

con $M = \frac{6\ sin\varphi}{3-sin\varphi}$ (inclinazione della retta di stato critico funzione) (63)

Considerando che il PIMY utilizza il criterio di rottura di Von Mises questo può essere scritto come:

$q = \frac{3}{\sqrt{2}}\tau_f$ (criterio di Von Mises) (64)

Per cui, per il termine attritivo funzione solo di ϕ' avremo:

$$\frac{\sqrt{2}}{3}\ \frac{6\ sin\varphi}{3-sin\varphi}\ p \qquad (65)$$

Se invece consideriamo la resistenza non drenata di un materiale puramente coesivo, è valida la relazione:

$$q = 2c_u \qquad (66)$$

Quindi nel nostro caso avendo imposto $\phi = 0$ avremo esclusivamente:

$$\tau_f = \frac{2\sqrt{2}}{3}\ c_u \qquad (67)$$

La coppia di valori p = 500 kPa e $\phi = 27°$ fornisce lo stesso valore di tensione di rottura imponendo $c_u = 0$. Il risultato della simulazione sarà il medesimo essendo la tensione di rottura non drenata impostata il valore di calcolo principale del modello. Il set 1 è stato scelto per la maggior rispondenza del comportamento ciclico in merito alla curva di decadimento del modulo di rigidezza a taglio rispetto al dato sperimentale cercando nel mentre di sostenere una resistenza non drenata fisicamente ammissibile rispetto ai dati sperimentali. Si riporta che la corrispondenza con la curva di decadimento del modulo di rigidezza al taglio si ottiene per un valore di c_u pari a 100 kPa, tuttavia questa opzione si è scartata a causa del valore troppo modesto di resistenza che non si addice ai fini preposti.

Nel set 2, invece a parità delle caratteristiche del modello illustrate per il set 1 si è scelto di ottenere una maggiore rispondenza con la risposta meccanica in prova monotona dell'elemento finito, a parità di condizioni al contorno, riducendo il modulo di rigidezza a taglio di riferimento G_0^{ref} ed aumentando la resistenza non drenata c_u rispetto al set 1.

In dettaglio, nella Fig. 23 si mostra il confronto dei risultati della simulazione numerica della prova triassiale eseguita con i due set di parametri del modello con quelli della prova di laboratorio.

In riferimento al confronto tra curva sperimentale e curva di calcolo nel piano p-q (Fig. 23 a1, a2) si sottolinea che non è possibile la perfetta corrispondenza in quanto non vi è incrudimento a compressione isotropa. Quindi, dato che l'acqua non reagisce a taglio, non si possono sviluppare le pressioni interstiziali che portano lo sbilanciamento del diagramma a sinistra tipico delle argille sovra-consolidate e ci si dovrà accontentare dell'andamento verticale (campo elastico). Infine (Fig. 23 a2) si nota l'incremento di resistenza voluto attraverso l'inserimento di un c_u più alto nel set 2. In Fig. 23 b1 e b2 si nota come la riduzione del modulo di rigidezza a taglio e l'aumento della resistenza e della deformazione a taglio di picco abbiano comportato la perfetta sovrapposizione. Tuttavia, si vedrà nei risultati della simulazione della prova di taglio semplice che la curva di decadimento risultante a tali parametri si discosta di molto da quella sperimentale. Tale comportamento è dovuto ad una carente descrizione dell'incrudimento a pressione isotropa. Si riporta anche il tentativo di assegnare le n superfici attraverso l'imposizione di coppie G-γ prese dalla curva sperimentale ottenendo come risultato un valore di c_u discorde (inferiore a quello sperimentale). In Fig. 23 c1 e c2 invece si riporta l'andamento delle sovrapressioni interstiziali nei 4 nodi di base.

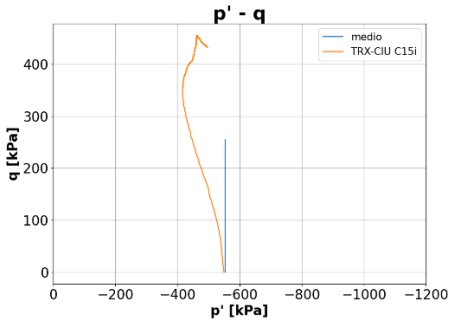
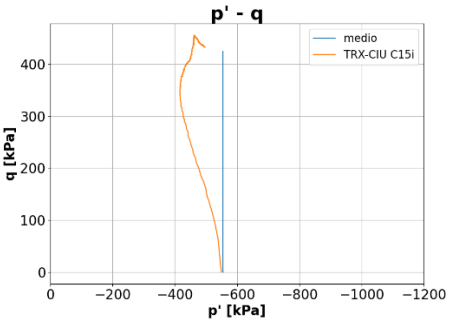
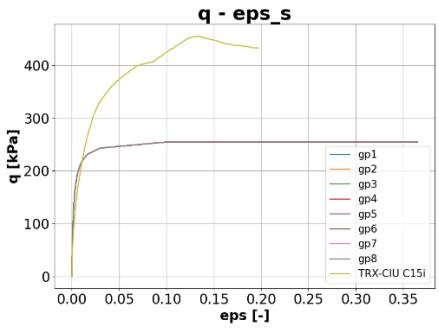
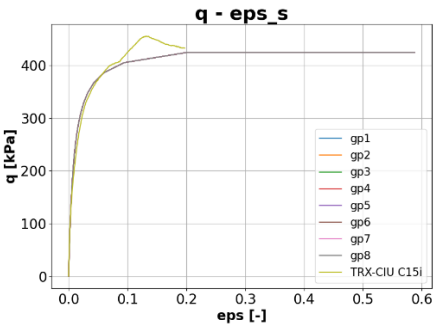
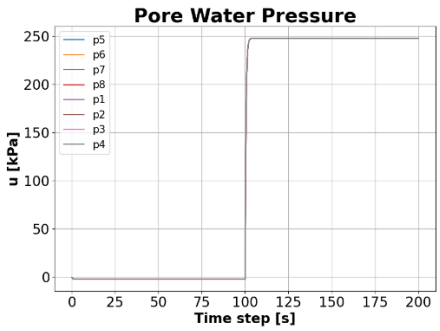
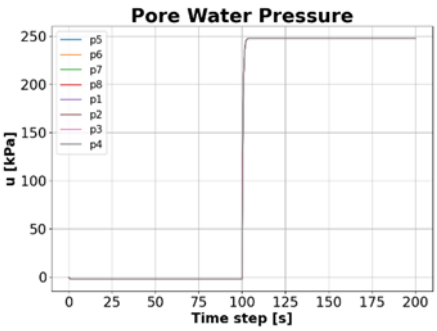
Risultati set 1	Risultati set 2
a1) 	a2) 
b1) 	b2) 
c1) 	c2) 

Fig. 23 Confronto della risposta del modello PIMY con i dati di laboratorio della prova TRX-CIU. I risultati delle prove sperimentali sono indicati in legenda con “TRX-CIU C15i.

I risultati delle simulazioni numeriche della prova di taglio ciclico eseguite con i due set di parametri sono illustrati nelle Fig. 24 in termini di curve tensioni e deformazioni di taglio τ-γ e diagrammi del fattore di smorzamento nel tempo D-t, per ogni livello di deformazione imposto. Il fattore di smorzamento è stato determinato seguendo un approccio energetico (Elia et al., 2021).

Definito come *i* il numero di intervalli come il rapporto tra la durata totale del segnale ed il singolo time-step, l'area incrementale sottesa al tratto i-esimo di risposta meccanica è pari a:

$$\Delta A(t)_i = \Delta\tau(t)_i \, \Delta\gamma(t)_i \tag{68}$$

quindi l'energia dissipata dal ciclo è pari a:

$$E(t_i)_- = \sum_i \Delta A(t)_i \tag{69}$$

Considerando come energia dissipata dal singolo ciclo quella calcolata assumendo come durata totale quella equivalente al singolo ciclo, quindi ad ogni nuovo ciclo i-esimo si azzera l'energia dissipata.

L'energia accumulata invece è pari a:

$$E_+ = 0.5 \, \tau_{max} \, \gamma_{max} \tag{70}$$

In cui τ_{max} è la massima tensione di taglio raggiunta dall'elemento in corrispondenza della massima deformazione di scorrimento γ_{max} prima dell'inversione del ciclo. Quindi si calcola la nota formula dello smorzamento:

$$D(t) = \frac{E(t)_-}{4 \, \pi \, E_+} \tag{71}$$

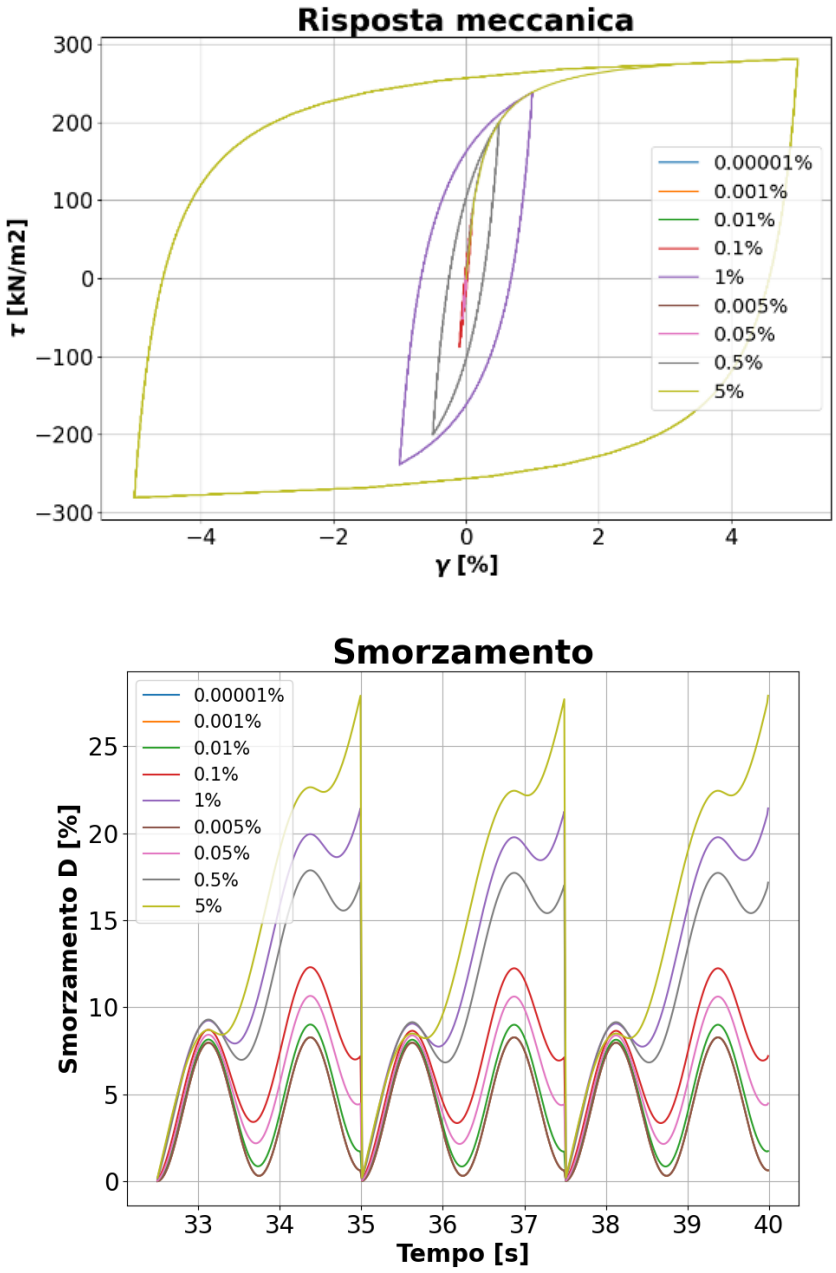


Fig. 24 a) Output della simulazione della prova di taglio ciclico, b) calcolo dello smorzamento.

I risultati numerici delle prove di taglio ciclico sono stati elaborati per descrivere l'evoluzione di modulo di rigidezza secante GS e del fattore di smorzamento D con il livello di deformazione γ. Essi sono confrontati in Fig. 26 con le curve sperimentali relative al campione C15i/B2 e con la curva ottenuta applicando la formula iperbolica alla base del modello costitutivo. Per il set 1, Il confronto mostra come i parametri del modello utili per la simulazione del comportamento meccanico simulato non sono equivalenti a quelli sperimentali. Infatti, come già evidenziato nel caso della prova TRX-CIU (Fig. 23), il modello non incrudisce a pressione isotropa rimanendo elastico con la relativa rigidezza, quindi non è possibile riprodurre una sovra-consolidazione nell'inizializzazione del materiale. Il modello non possiede accoppiamento volumetrico-deviatorico per cui vi possono essere delle difficoltà nella rappresentazione di un materiale complesso e non omogeneo come quello sperimentato. Per il set 2 invece l'aderenza perfetta alla risposta meccanica monotona ha richiesto una riduzione della Gref ed un aumento della resistenza non drenata con un collasso atteso nel piano γ-Gs.

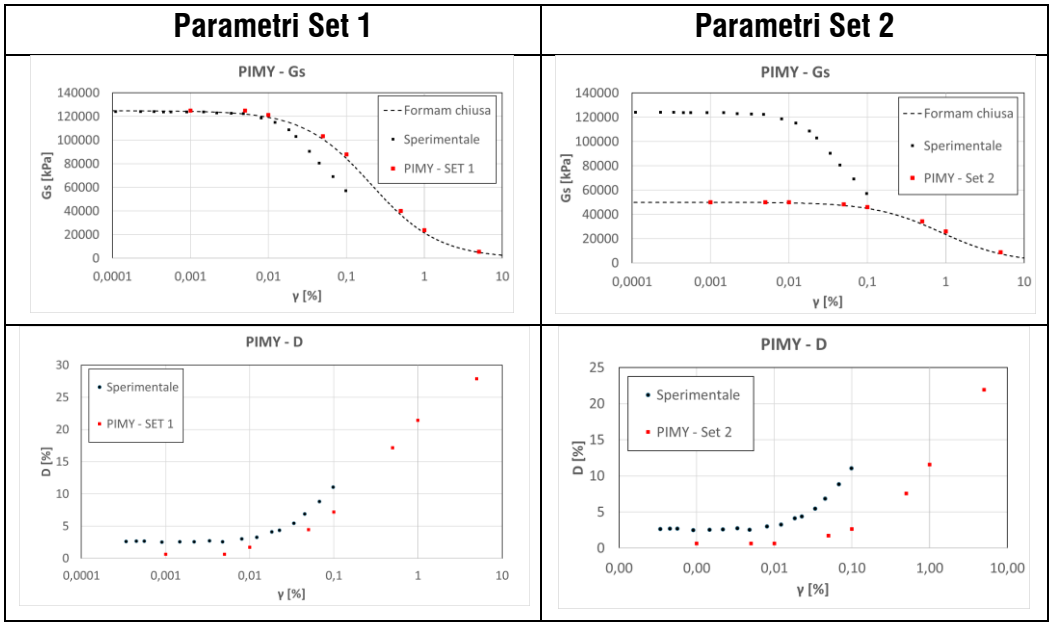


Fig. 25 Confronto tra la previsione del modello PIMY ed i risultati sperimentali: in alto le relazioni nel piano Gs - γ, in basso lo smorzamento D - γ.

Nonostante per limitazioni intrinseche del modello non si sia riuscito a trovare parametri tali da soddisfare tanto la corrispondenza con la risposta statica che con quella ciclica, si è scelto di adottare il set di parametri che si avvicina il più possibile alla curva di decadimento sperimentale.

7.2 CALIBRAZIONE DEL MODELLO HSSMALL

Il modello costitutivo HSsmall è descritto da un numero inferiore di parametri (Cudny & Truty, 2020) costitutivi, che possono essere più facilmente calibrati. Essi, infatti, sono stati impostati utilizzando la formulazione del modello e senza la necessità di simulare le prove sperimentali.

Sulla base delle considerazioni in Amorosi et al. (2016) e di Brinkgreve et al. (2016) si è proceduto nel modo seguente:

1. Definizione dei i parametri γ_{sat} e G_0^{ref} in base alle risultanze sperimentali.
2. Definizione del valore del rapporto tra G_0^{ref} e G^{ref}_{ur} si entra nell'abaco di Alpan (Alpan 1972, da Bringreve RJB 2013) per cui si ottiene il G^{ref}_{ur} (Fig. 16).
3. Sulla base delle note equazioni derivate dalle ipotesi di elasticità isotropa si è ottenuto il valore di E^{ref}_{ur}.
4. $E^{ref}_{oed} = E^{ref}_{ur} / 2$.

Una sintesi dei parametri selezionati è riportata in Tab. 4:

Tab. 4 - Parametri del modello HSsmall

Parametro	**Valore**		**Parametro**	**Valore**	
γ_{sat}	19	kN/m^3	m	0	-
G_0^{ref}	125000	kPa	ρ	1937,4707	kg/m3
G_0^{ref} / G^{ref}_{ur}	20	-	ϕ'	0	rad
G^{ref}_{ur}	6250	kPa	c'	270	kPa
E^{ref}_{ur}	16250	kPa	$\gamma_{0.7}$	0,001	
E^{ref}_{50}	3000	kPa	OCR	1	-
E^{ref}_{oed}	8125	kPa	$\gamma_{cut-off}$	0,009019	-

Il risultato dell'uso dei parametri in Tab. 4 può essere apprezzato dall'andamento della curva di decadimento del modulo di rigidezza al taglio mostrata dalla Fig. 26 in cui si può osservare una buona sovrapposizione tra il risultato del modello PIMY e quello dell'HSsmall.

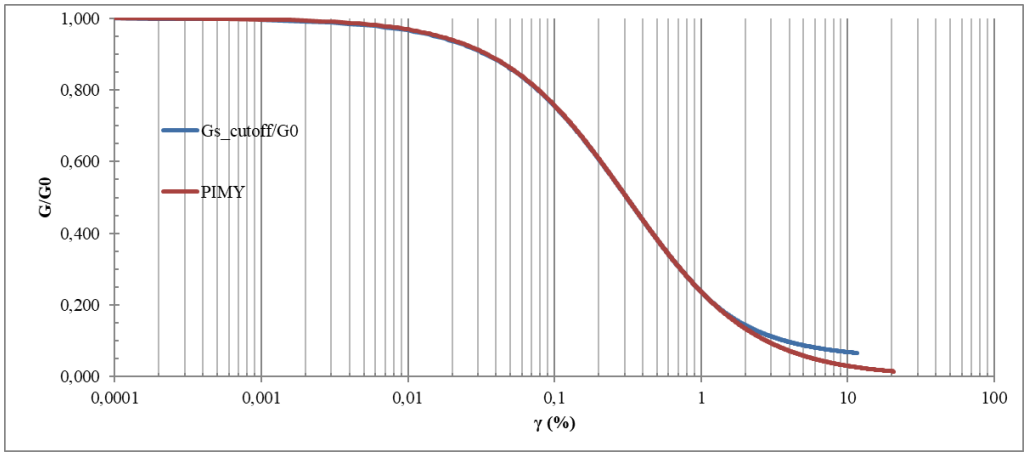


Fig. 26 Calibrazione della curva di decadimento del modulo di rigidezza a taglio del modello HSsmall, confrontato con quella del modello PIMY.

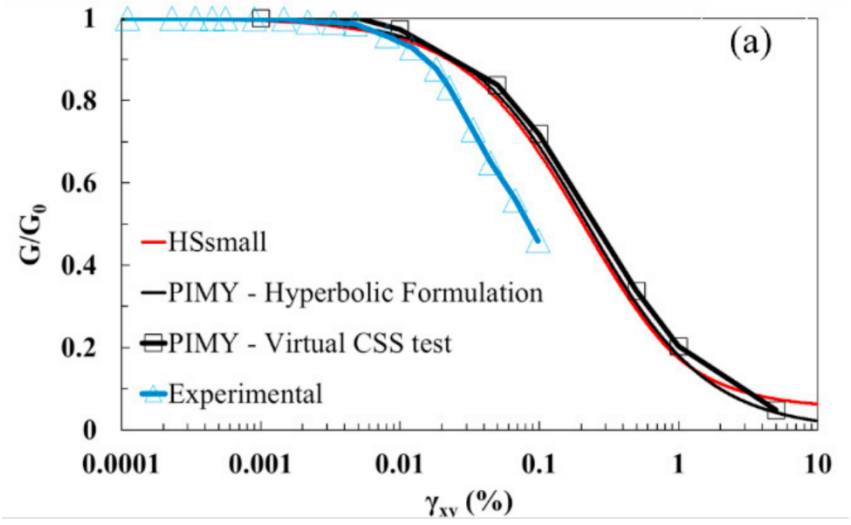


Fig. 27- Sovrapposizione finale delle curve G/G0 (Elia et al,2025)

8. MODELLI 1D PER L'ANALISI DELLA RISPOSTA SISMICA LOCALE: CONFRONTO OPENSEES E PLAXIS

8.1 INTRODUZIONE

Dopo un primo step di approccio e setup del sistema HPC a disposizione per il progetto, la definizione di un container per l'esecuzione di analisi in parallelo con il codice OPENSEES e l'elaborazione di prototipi e codici in linguaggio Python e TCL per il pre-processing e post-processing, lo step successivo si è concretizzato nella definizione di modelli numerici per l'esecuzione di analisi di propagazione sismica in condizioni monodimensionali, implementati nel codice di calcolo OPENSEES. Allo scopo di verificare l'efficacia del procedimento di modellazione e di programmazione, i risultati di queste simulazioni sono confrontati con quelli ottenuti con il codice agli elementi finiti PLAXIS, la cui reputazione è ben consolidata anche in campo accademico.

La verifica della efficacia dei modelli implementati in OPENSEES è stata eseguita sviluppando dapprima modelli 3D visco-elastici. Successivamente, si è passati alla modellazione elasto-plastica non lineare, adottando i due modelli costitutivi descritti nel Capitolo 6: il modello Pressure Independent Multi Yield implementato in OPENSEES e il modello Hardening Soil Small Strain stiffness, implementato in PLAXIS.

8.2 MODELLAZIONE DELLA RISPOSTA SISMICA IN CAMPO ELASTICO

8.2.1. DESCRIZIONE DEI MODELLI NUMERICI

L'analisi della risposta sismica locale in condizioni di campo libero (Chiaradonna, 2022) è stata eseguita con riferimento ad una colonna di terreno secco di spessore pari a 40 m poggiante su uno strato di roccia di spessore pari a 1 m. La colonna, a base quadrata, ha dimensioni pari a 1 m x 1 m e riproduce la condizione di propagazione monodimensionale.

Il terreno e la roccia sono descritti con il modello visco-elastico lineare (VEL), le cui caratteristiche sono riportate in Tab. 5:

Tab. 5 - Parametri del modello visco-elastico lineare

Terreno		**Bedrock**	
Vs	$250\ m/s$	Vs	$800\ m/s$
G	$125000\ kPa$	G	$1300000\ kPa$
E	$325000\ kPa$	E	$3.392 \cdot 10^6\ kPa$
v	0.3	v	0.3
ρ	$1.97 \cdot 10^3\ kg/m^3$	ρ	$2 \cdot 10^3\ kg/m^3$

La capacità dissipativa del materiale è stata introdotta come smorzamento viscoso, secondo la formulazione di Rayleigh (Elia & Rouainia, 2022). Nello specifico, per lo strato di 1 m di roccia alla base si è assunto uno smorzamento pari all' 0.1% imponendo valori pari a $\alpha_R = 0.0114240$ e $\beta_R = 0.02893733\text{e-}3$ per un intervallo di frequenze tra 1 e 10 Hz.

Per lo strato di 40 m di terreno, invece si è imposto uno smorzamento viscoso pari al 2.6 % imponendo fattori $\alpha_R = 0.4514$ e $\beta_R = 0.6579\text{e-}3$ per un intervallo di frequenze tra 1.56 Hz (cfr. Eq. 72) e 11 Hz. Nella scelta dell'intervallo delle frequenze nella formulazione di Rayleigh si è considerato, per il valore massimo, il contenuto in frequenza dei wavelet utilizzati, che non supera gli 11 Hz. Per il valore minimo, invece, si è considerato il valore della frequenza fondamentale del banco.

Il valore dello smorzamento è stato desunto dalle curve delle prove in colonna risonante a piccolissimi livelli di deformazione. Per il caso considerato, la frequenza fondamentale di risonanza del banco è pari a:

$$f_1 = \frac{V_S}{4H} = \frac{250\ m/s}{4 \cdot 40\ m} = 1.56\ Hz \quad (72)$$

Nel codice PLAXIS 2D sono stati utilizzati elementi finiti isoparametrici triangolari a 6 nodi, mentre in OPENSEES sono stati utilizzati elementi esaedrici isoparametrici a 20 nodi con formulazione u-p (Zaohui Y, 2000) di lato 1 m. La procedura di calcolo agli elementi finiti è stata organizzata in due fasi: una prima fase in cui si esegue un'analisi statica in condizioni K_0 di generazione dello stato tensionale iniziale; una seconda fase di analisi dinamica non drenata, in cui si applica l'evento sismico.

Le condizioni al contorno nella fase dinamica per il modello implementato in PLAXIS 2D prevedono vincoli del tipo tied-nodes sulle facce laterali della colonna e vincolo compliant-base alla base del modello, che realizza la condizione di substrato deformabile al fine di evitare la riflessione totale del segnale nel dominio.

Nella modellazione in OPENSEES invece si è considerato un modello numerico 3D, in cui sono state imposte condizioni al contorno di tipo Lysmer-Kuhlemeyer alla base (Løkke & Chopra, 2019, Løkke & Chopra, 2018, Løkke & Chopra, 2017, Løkke & Chopra, 2013), mentre sulle facce laterali, con normale parallela alla direzione del moto, si sono imposti vincoli di tipo tied-nodes. Sulle facce con normale perpendicolare alla direzione del moto si sono imposti vincoli tipo carrello con scorrimento impedito lungo la normale, per ricreare le condizioni di propagazione monodimensionale con moto solo lungo una delle due direzioni orizzontali.

Il segnale di input è un segnale impulsivo di tipo Ricker-Wavelet (Ricker, 1940), imposto alla base del modello in termini di diagramma delle velocità nel tempo. Esso è definito in modo tale da avere la massima ampiezza nell'intorno di frequenza centrale secondo una forma "a campana".

La forma d'onda in termini di accelerazione (*a(t)*) e velocità (*v(t)*) variabili nel tempo è definita dalle equazioni:

$$a(t) = (-2\pi^2 fn^2 t^2 + 1)e^{-\pi^2 fn^2 t^2} \quad (73)$$

$$v(t) = te^{-2\pi^2 fn^2 t^2} \quad (74)$$

in cui:

t è la variabile tempo espressa in s;

f_n è frequenza centrale espressa in Hz.

Le simulazioni della risposta sismica locale sono state eseguite applicando tre wavelet aventi le frequenze centrali pari a 0.5 Hz, 2 Hz e 3 Hz (Fig. 28), scelte in quanto rappresentative del contenuto in frequenza di un segnale sismico reale e tali da includere la frequenza fondamentale del banco.

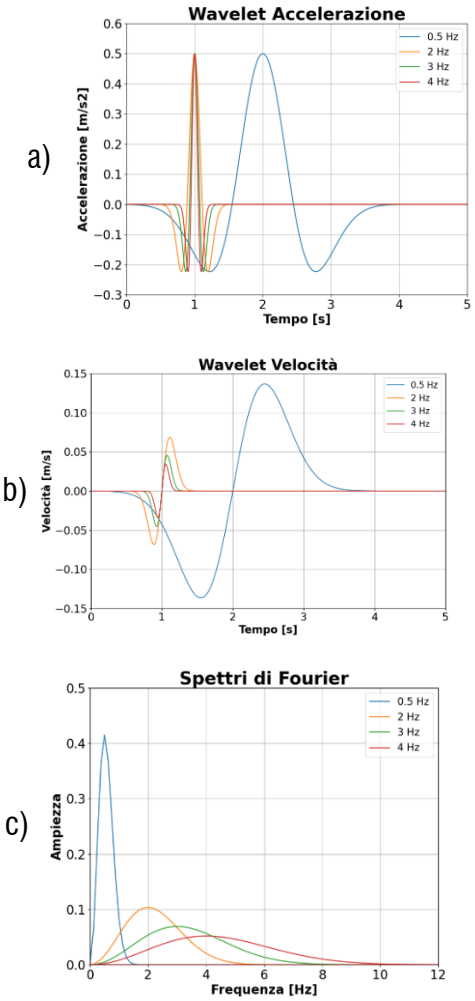


Fig. 28 Wavelet di input: a) accelerazione, b) velocità, c) spettri di Fourier dell'accelerazione.

Nella impostazione del modello numerico, si è adottato il segnale in termini di velocigramma, in quanto requisito di input per il software OPENSEES, oltre che più facilmente governabile in termini di codifica.

Si è cercato inoltre di mantenere, per quanto possibile le stesse caratteristiche nei parametri di integrazione specialmente nel caso non-lineare, ed in particolare quelli utilizzati da PLAXIS, in quanto parametri fissi del codice. Quindi si sono utilizzati il metodo di Newton-Raphson per l'integrazione delle leggi costitutive mentre il metodo di Newmark incondizionatamente stabile per l'integrazione nel tempo (Cui et al., 2024, Zhang et al., 2020).

Tutti i modelli sviluppati in questo capitolo con il codice FEM OPENSEES sono stati risolti in parallelo con 4 cores, utilizzando il proprio personal computer. I codici utilizzati STKO (Petracca et al., 2017) e OPENSTOOLS permettono di scalare anche su più processori, come già testato nelle fasi preliminari su infrastruttura RECAS con l'opportuno setup implementato nell'ambito del presente dottorato. Il massimo numero di CPU utilizzato per la risoluzione di un problema è arrivato a 90 (cfr. Capitolo 4).

Nelle presenti analisi 1D, al fine della validazione numerica, il codice eseguibile è stato sviluppato in due diversi modi: utilizzando il pre- e post-processore STKO e il codice pre-processore OPENSTOOLS, sviluppato nell'ambito del presente dottorato e reso disponibile online sotto licenza open-source.

8.2.2. RISULTATI DELLE ANALISI DINAMICHE VEL

I risultati delle simulazioni numeriche, eseguite con i tre codici PLAXIS (PLX), STKO e OPENSTOOLS (OPS_TLS), sono riportati in termini di:

- accelerogrammi;
- spettri di Fourier degli accelerogrammi ottenuti in corrispondenza di un punto alla base, inteso come estremità inferiore dello strato assunto come bedrock, ed uno in sommità del modello;
- funzione di amplificazione, ottenuta come rapporto tra lo spettro di Fourier dell'accelerogramma registrato alla base del bedrock e quello del segnale registrato in testa al modello in condizioni di "free-field" (piano campagna); Tale indice viene utilizzato per cogliere il risultato

dell'integrazione all'interno della colonna ed effettuare il confronto per la valutazione della performance del codice e della modellazione effettuata.

- profili delle accelerazioni massime lungo la verticale passante per il centro dei modelli.

Nello specifico, in Fig. 29 sono riportati i risultati delle analisi eseguite con il wavelet con frequenza pari a 0.5 Hz; in Fig. 30 quelli relativi all'analisi con wavelet di frequenza pari a 2 Hz; in Fig. 31 quelli relativi all'analisi con wavelet di frequenza pari a 3 Hz; in Fig. 32 quelli relativi all'analisi con wavelet di frequenza pari a 4 Hz.

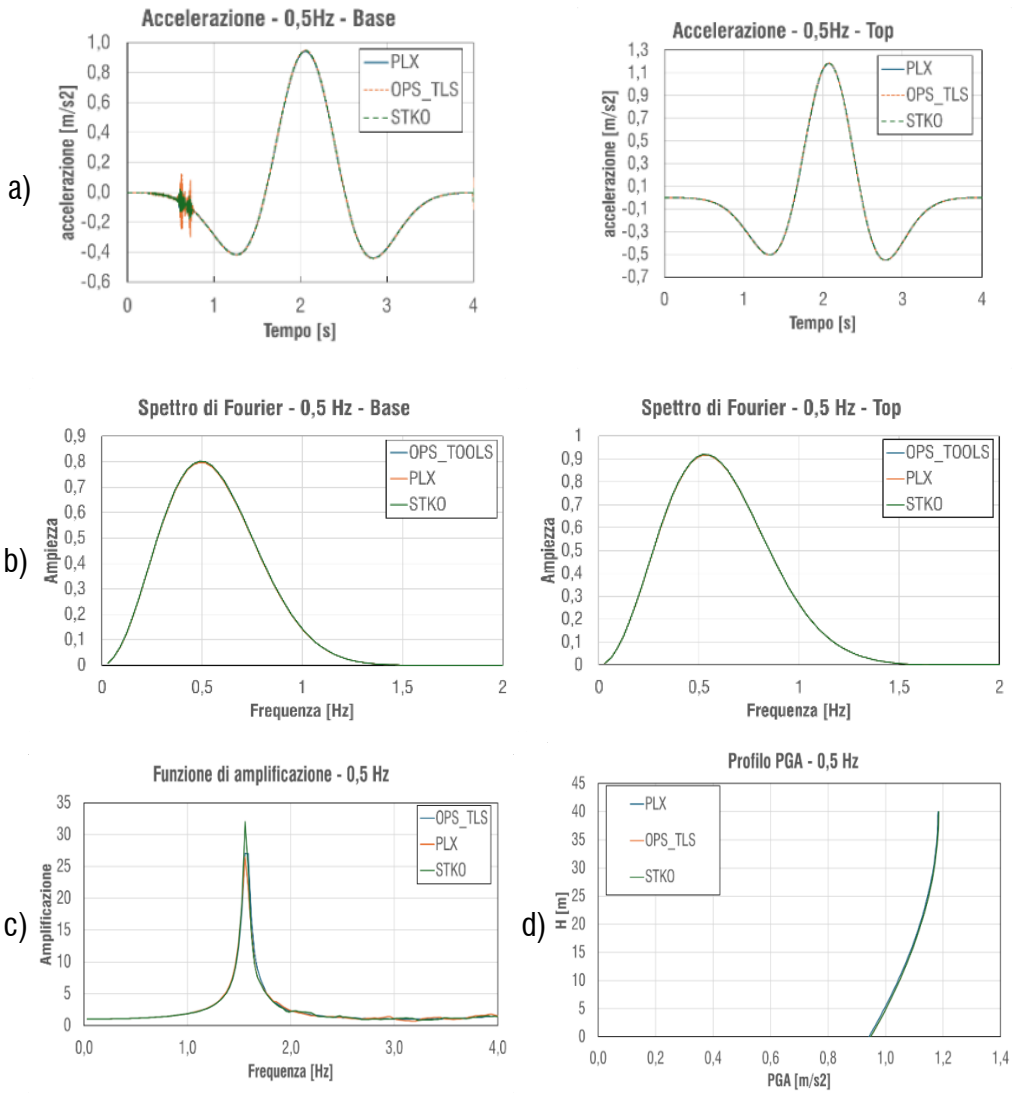


Fig. 29 Confronto propagazione 1D - segnale wavelet di 0.5 Hz.

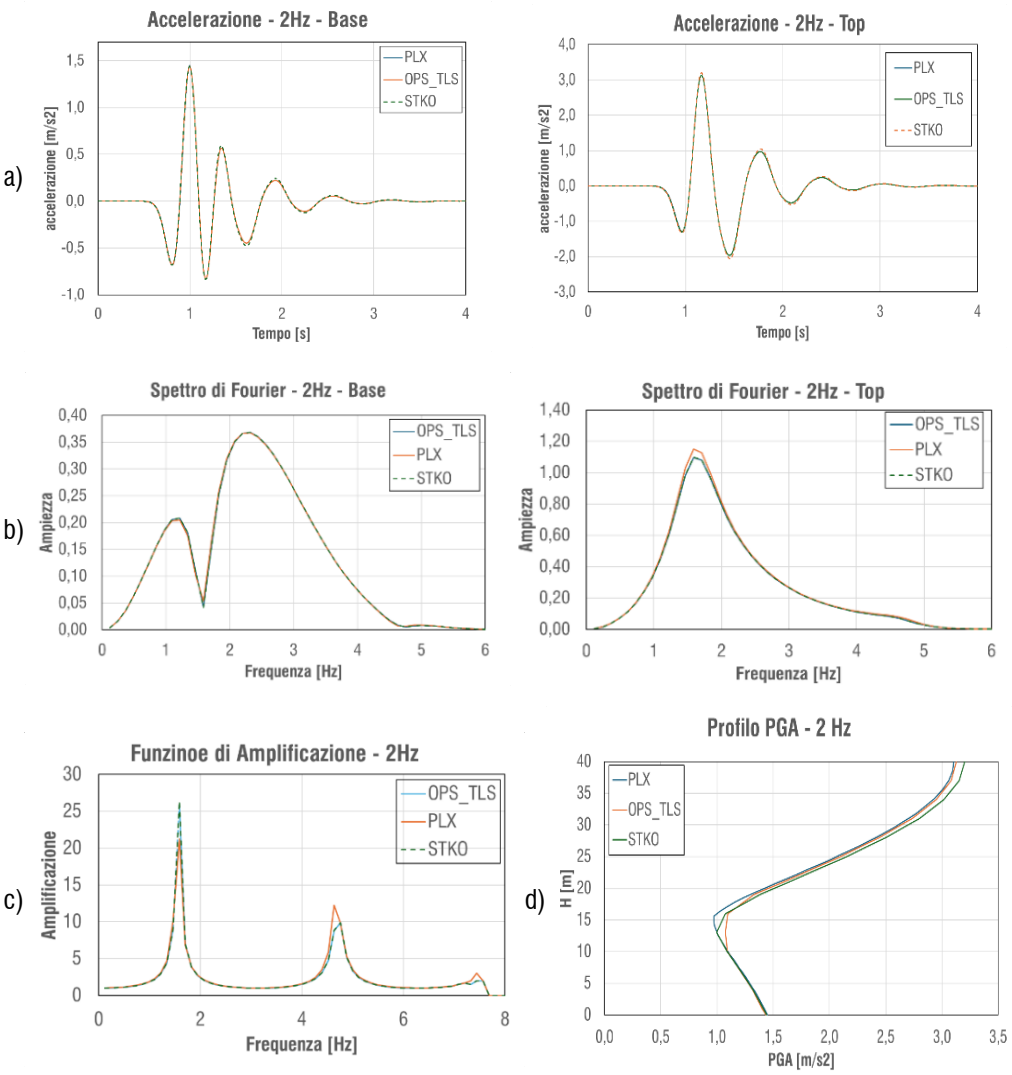


Fig. 30 Confronto propagazione 1D - segnale wavelet di 2 Hz.

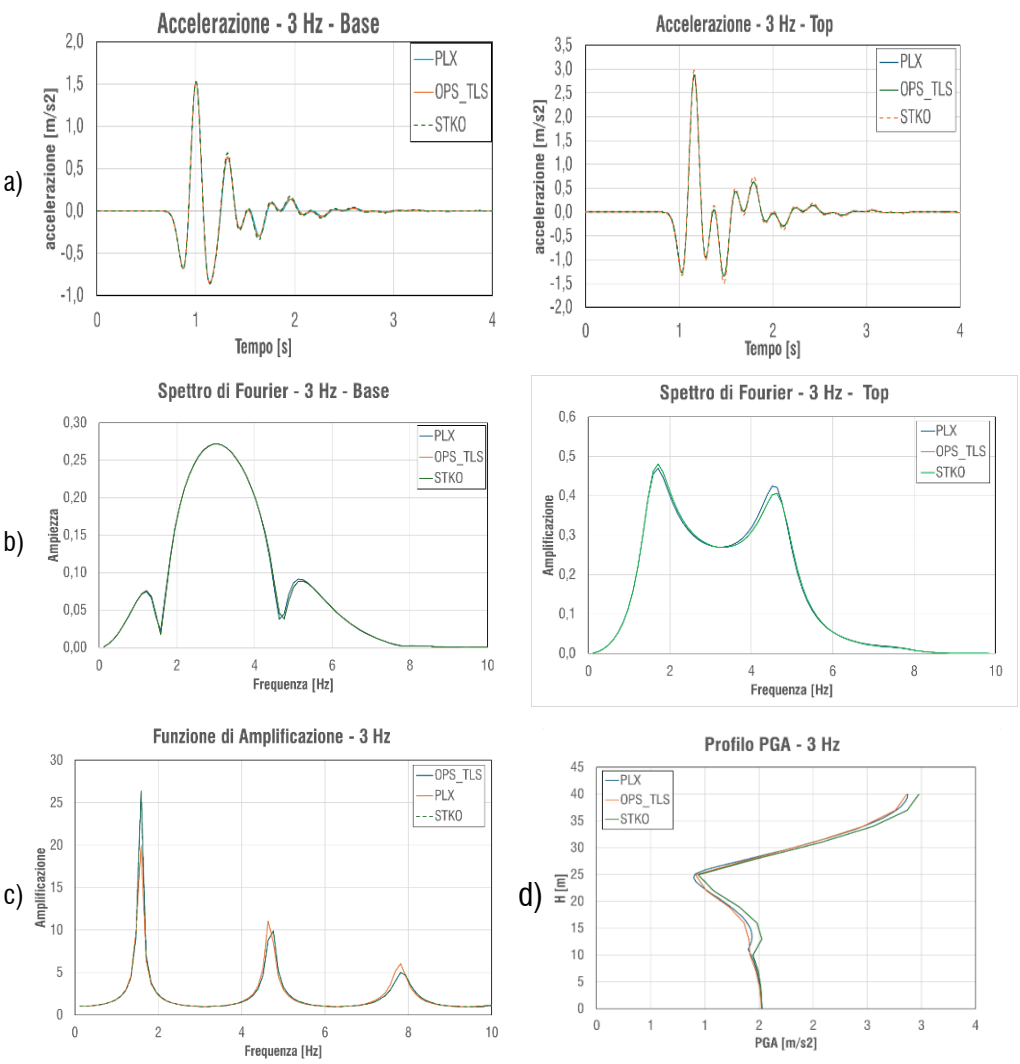


Fig. 31 Confronto propagazione 1D - segnale wavelet di 3 Hz.

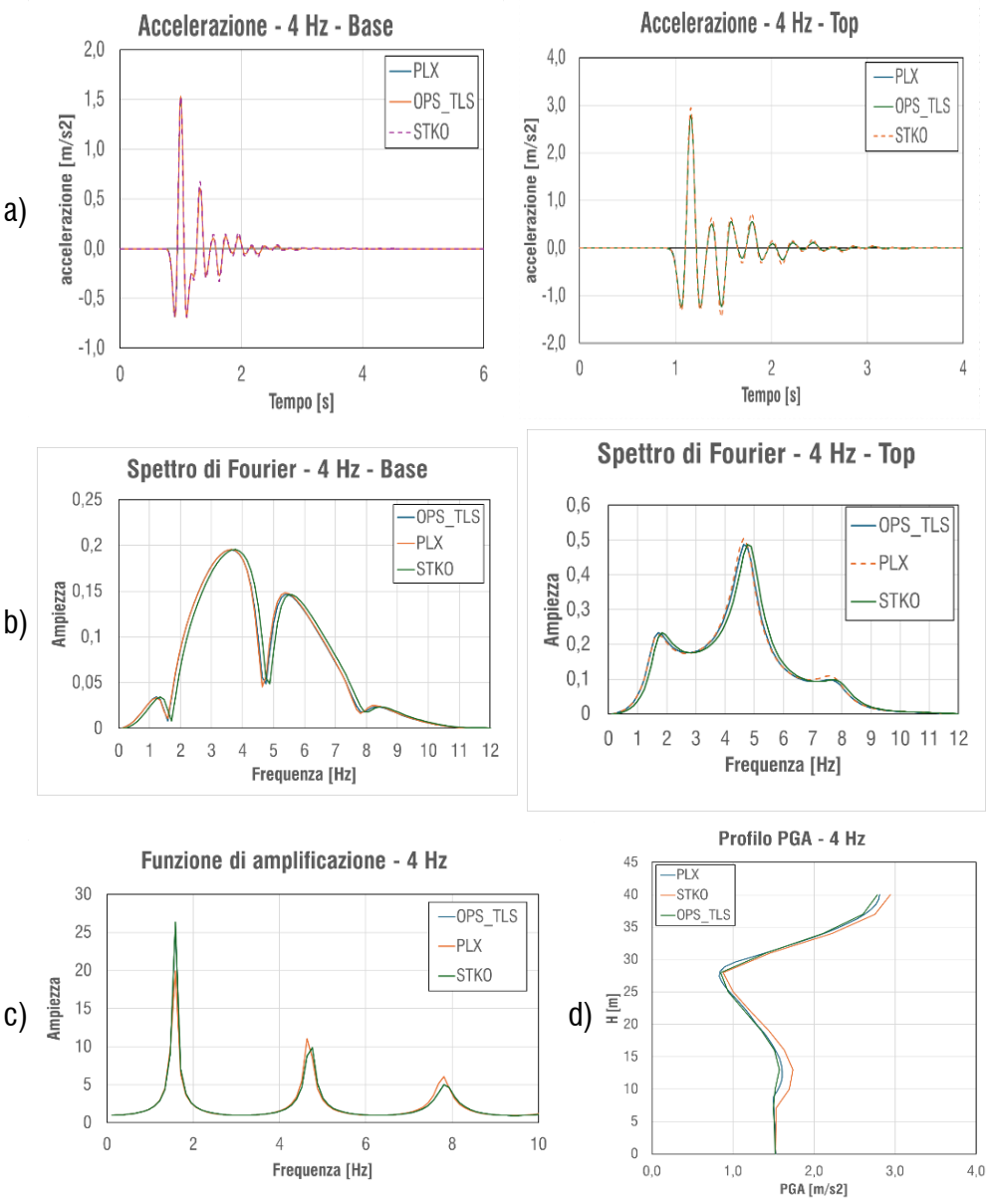


Fig. 32 Confronto propagazione 1D - segnale wavelet di 4 Hz.

Il confronto dei risultati mostra una buona corrispondenza tra gli output della propagazione monodimensionale simulata con 3 modelli ottenuti seguendo 3 approcci differenti:

- utilizzo di 2 solutori diversi: PLAXIS e OPENSEES
- codifica del modello in linguaggio OPENSEES secondo 2 pre-processori differenti: STKO e OPENSTOOLS (programmato nell'ambito del presente dottorato).

I risultati in termini di fattore di amplificazione riportano correttamente un aumento del fattore intorno alla frequenza fondamentale del banco.
Le lievi differenze che si possono notare dipendono da:

- Differente dimensione del problema: in PLAXIS si è modellato uno stato piano mentre in OPENSEES si è modellato in 3D con condizioni al contorno tali da rendere la propagazione monodimensionale.
- Differente tipo di elemento finito: in PLAXIS si è utilizzato il triangolo a 6 nodi, mentre in OPENSEES un esaedro a 20 nodi (20_8 BrickUP)(Zaohui, 2000)

Tuttavia, il differente rumore numerico introdotto con queste differenze è moderato dallo smorzamento viscoso alla Rayleigh introdotto pertanto si dissipano velocemente per le tipiche frequenze colpite che sono quelle alte in genere maggiori di 15 – 20 Hz.
Si fa notare, infatti, il ruolo della qualità della discretizzazione e dell'ordine della funzione di forma nell'integrazione temporale al passo in relazione al problema della propagazione del segnale nel dominio.
Questi parametri, infatti, se non opportunamente dimensionati possono generare rumore numerico e imprecisioni nella soluzione ricercata (Seron et al., 1990).
Inoltre, anche se definita al fine di evitare il taglio delle frequenze e per elementi lineari a 4 nodi, si è rispettata anche la regola sulla dimensione massima dell'elemento finito (Kuhlemeyer & Lysmer, 1973):

$$h_{max} = \frac{V_s}{(6 \div 8) f_{max}} \tag{74}$$

Con h_{max} = dimensione massima dell'elemento finito
V_S = Velocità delle onde di taglio nel mezzo [m/s]
f_{max} = Frequenza massima in input

In particolare, si riporta l'attenzione anche sull'implementazione delle condizioni al contorno di tipo compliant base e tied-nodes nei codici in linguaggio OPENSEES.
Infine, si riporta anche un effetto dello smorzamento alla Rayleigh dovuto alla sua formulazione. In particolare, i fattori di Rayleigh per il terreno sono stati selezionati secondo un intervallo di frequenze compreso tra 1 e 10 Hz per uno smorzamento pari al 2.6%. Tuttavia, la frequenza fondamentale del banco è pari a 1,56 Hz quindi lo smorzamento risultante sarà leggermente inferiore e comunque variabile in funzione delle frequenze eccitate durante l'integrazione nel tempo del segnale. Quindi l'analisi può subire modeste variazioni nel risultato in relazione ai parametri imposti al risolutore e alla discretizzazione del segnale in input. In Fig. 33 si mostra lo smorzamento corrispondente al valore della frequenza fondamentale del banco.

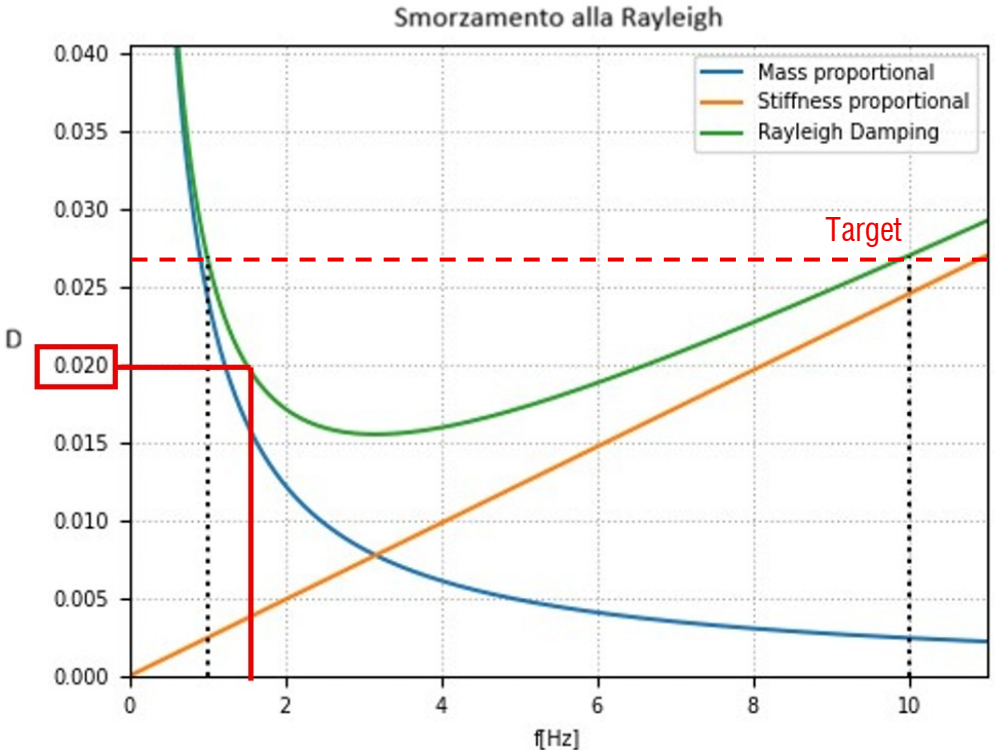


Fig. 33 Andamento dello smorzamento alla Rayleigh funzione della frequenza e valore dello smorzamento corrispondente alla frequenza fondamentale del banco (1,56 Hz)

La codifica del modello è avvenuta in linguaggio OPENSEES secondo 2 pre-processori differenti: STKO e OPENSTOOLS (programmato nell'ambito del presente dottorato).

Considerando l'esperienza comprovata con il software PLAXIS, la coincidenza dei risultati fornisce preziose indicazioni tanto sulla corretta modellazione quanto sul corretto funzionamento delle funzioni implementate in codice Python. Tali funzioni infatti hanno tradotto il modello in linguaggio OPENSEES correttamente in 2 modi differenti e tale da ripercorrere il risultato ottenuto con un software di fama comprovata.

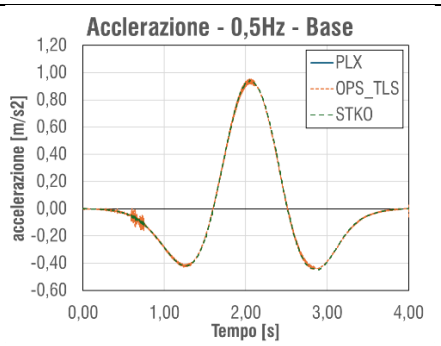
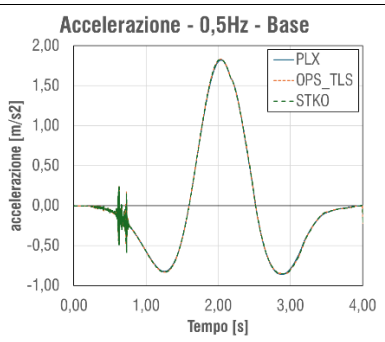
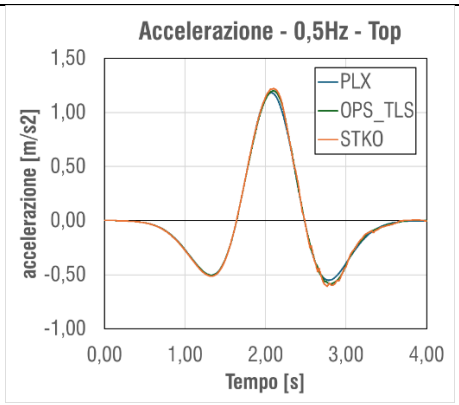
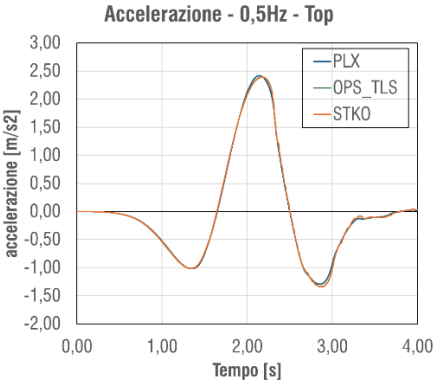
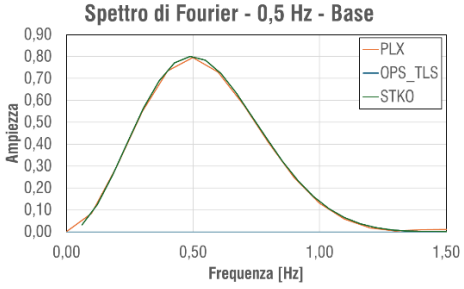
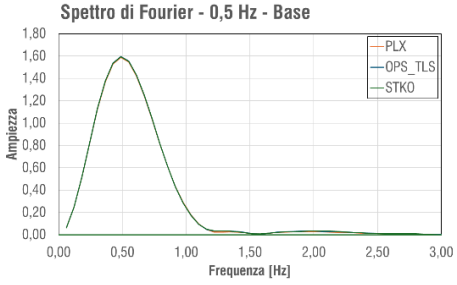
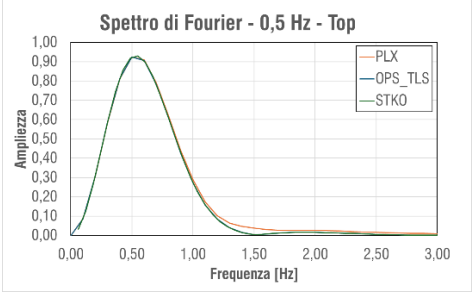
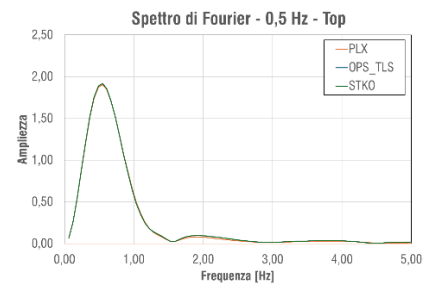
8.3 MODELLAZIONE DELLA RISPOSTA SISMICA IN CAMPO ELASTO-PLASTICO

8.3.1. DESCRIZIONE DEI MODELLI NUMERICI

Allo scopo di verificare la modellazione anche in campo elasto-plastico (EP), sono state eseguite delle analisi di risposta sismica locale con riferimento alla stessa colonna di terreno considerata nelle precedenti analisi, ma adottando per il terreno i modelli costitutivi elasto-plastici HSsmall e PIMY, descritti e calibrati nei capitoli precedenti. In queste simulazioni i tre segnali di input di tipo wavelet sono stati scalati ad un valore di accelerazione massima pari a 0.05g e 0.1g, al fine di mettere in risalto il comportamento elasto-plastico dei materiali con decadimento del modulo di rigidezza a taglio. Le condizioni al contorno e le fasi di calcolo adottate in queste simulazioni sono identiche a quelle descritte per le analisi in campo elastico.

8.3.2. RISULTATI DELLE ANALISI DINAMICHE EP

I risultati delle simulazioni dinamiche elasto-plastiche (EP), eseguite con i tre codici PLAXIS (PLX), STKO e OPENSTOOLS (OPS_TLS), sono riportati in termini di: accelerogrammi e relativi spettri di Fourier della risposta ottenuta in corrispondenza di un punto alla base ed uno in sommità del modello; profili delle accelerazioni massime lungo la verticale passante per il centro dei modelli; curve tensioni-deformazioni di taglio per un punto in prossimità della base ed uno in prossimità della superficie. Nello specifico, in Fig. 33 sono riportati i risultati delle analisi eseguite con il wavelet avente frequenza pari a 0.5 Hz, con PGA pari a 0.05g e 0.1g, rispettivamente; in Fig. 34 quelli relativi all'analisi con wavelet di frequenza pari a 2 Hz, con PGA pari a 0.05g e 0.1g, rispettivamente; in Fig. 35 quelli relativi all'analisi con wavelet di frequenza pari a 3 Hz, con PGA pari a 0.05g e 0.1g, rispettivamente.

Wavelet 0.5 Hz_PGA= 0.05g	Wavelet 0.5 Hz_PGA= 0.1g
	
	
	
	

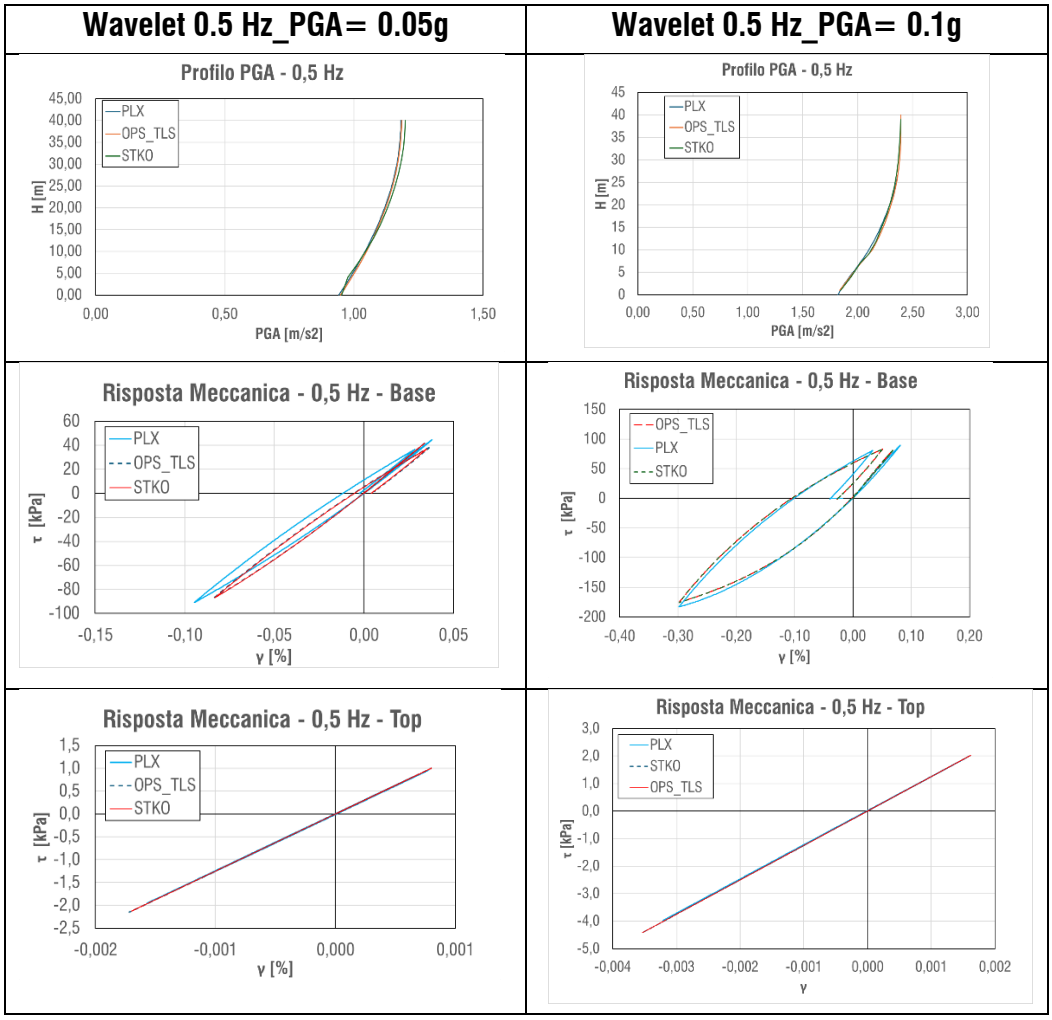
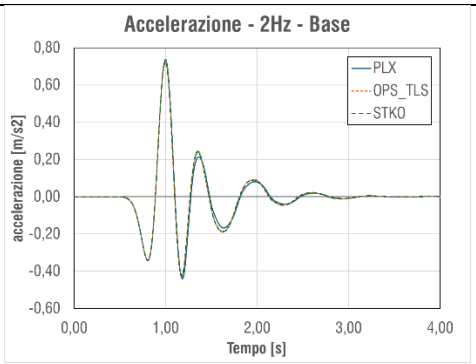
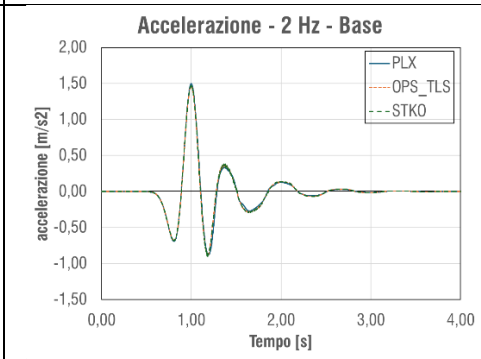
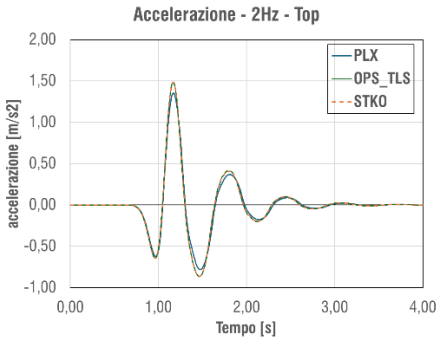
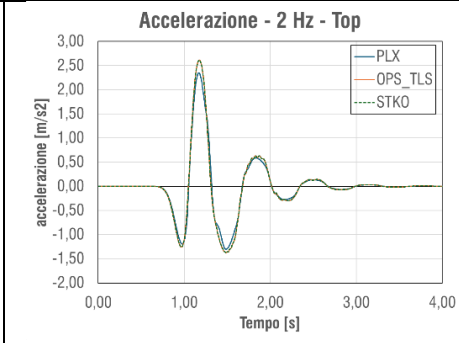
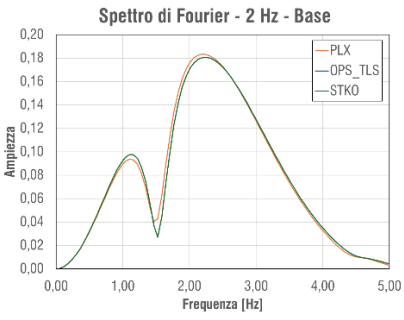
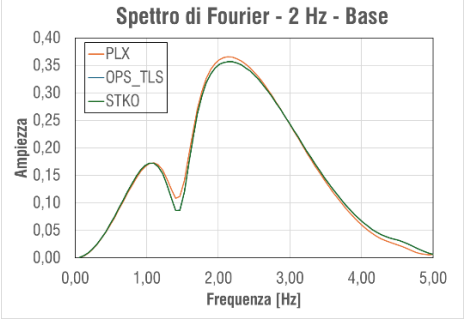
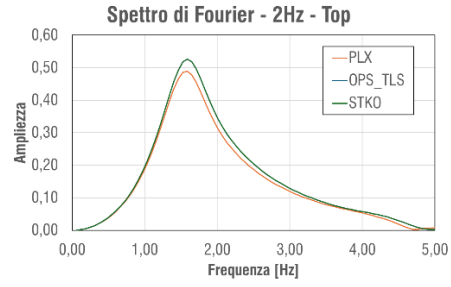
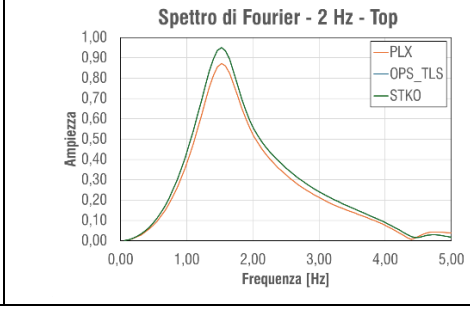


Fig. 34 Analisi di propagazione 1D elasto-plastica - wavelet di 0.5 Hz.

Wavelet 2 Hz_PGA= 0.05g	Wavelet 2 Hz_PGA= 0.1g
	
	
	
	

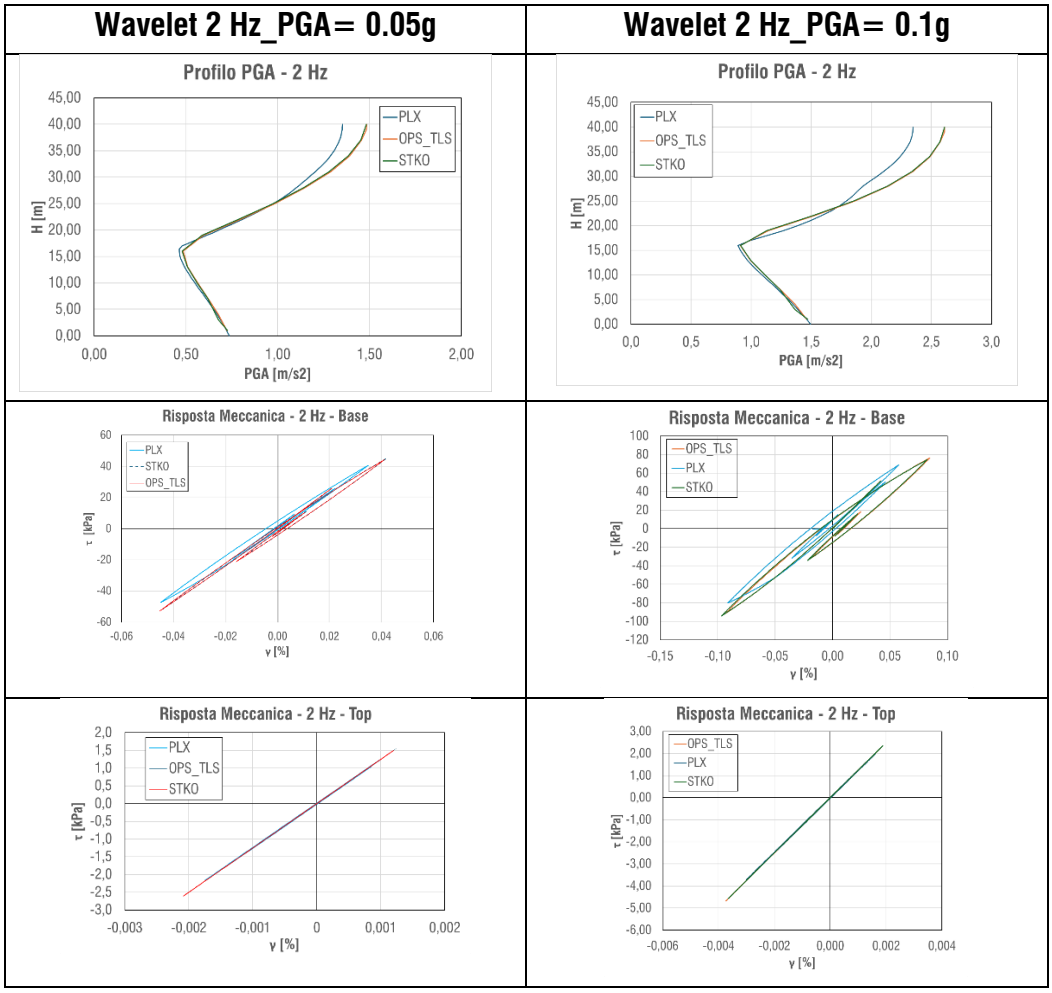
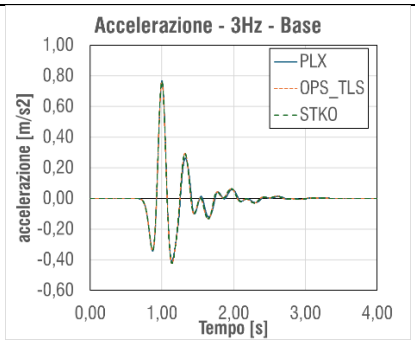
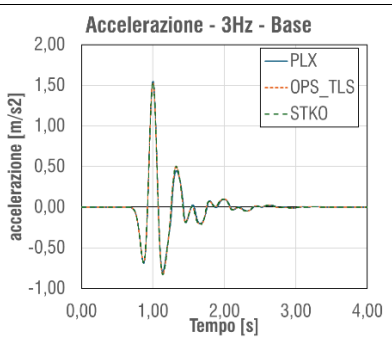
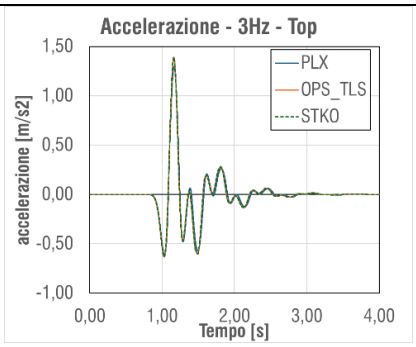
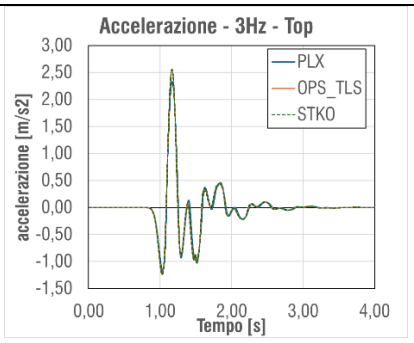
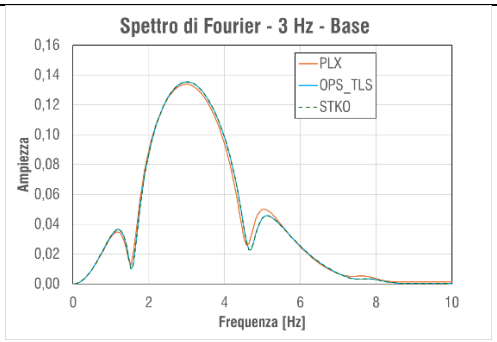
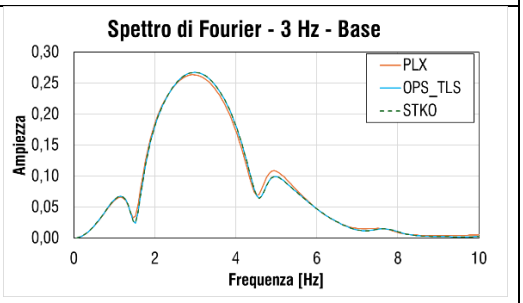
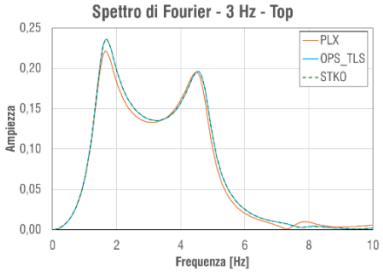
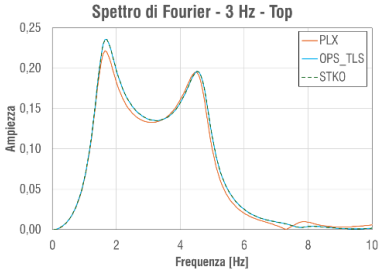


Fig. 35 Analisi di propagazione 1D elasto-plastica - wavelet di 2 Hz.

Wavelet 3 Hz_PGA= 0.05g	Wavelet 3 Hz_PGA= 0.1g
	
	
	
	

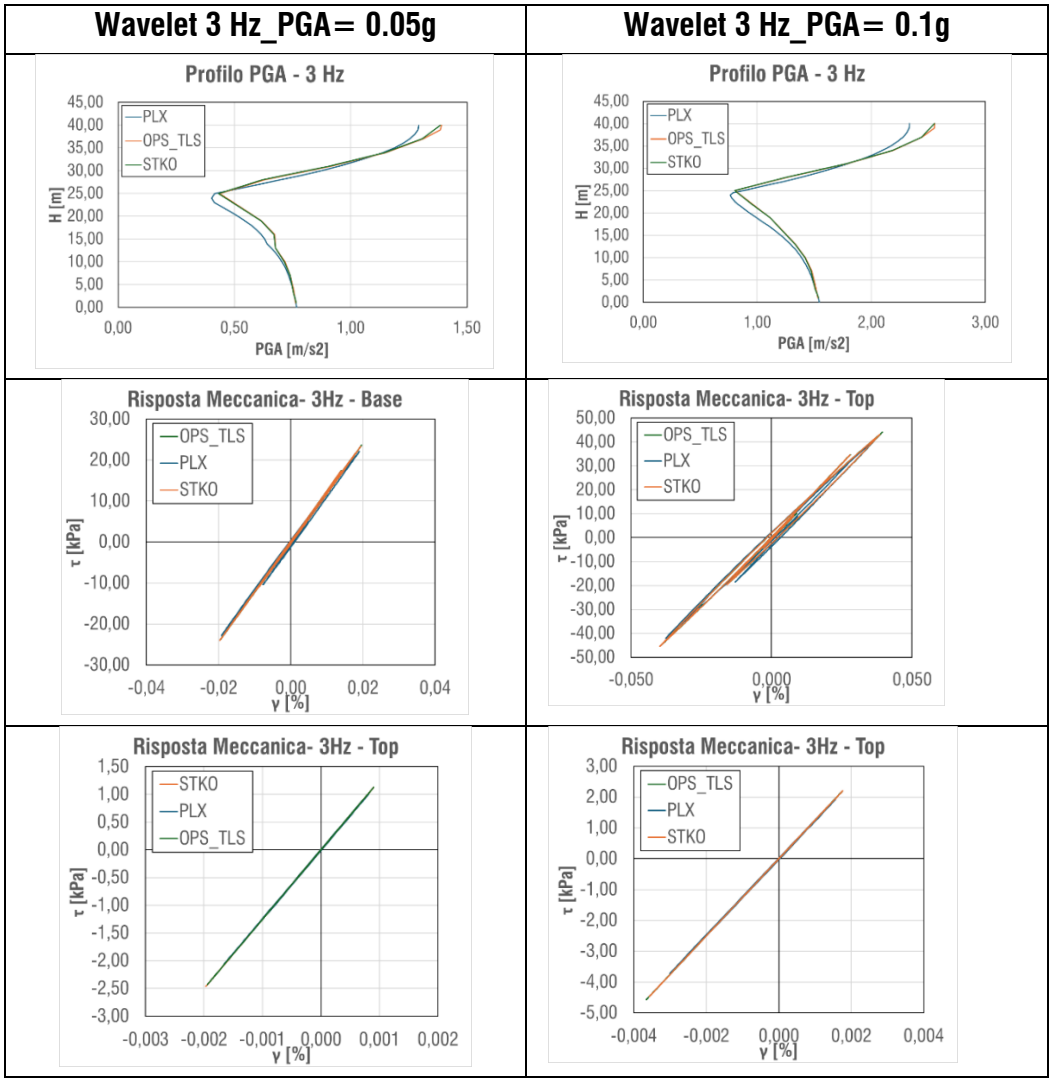


Fig. 36 Analisi di propagazione 1D elasto-plastica - wavelet di 3 Hz.

Il confronto dei risultati mostra un buon accordo dei modelli costitutivi in OPENSEES e PLAXIS a confermare la calibrazione effettuata. Tuttavia, il modello risolto il PLAXIS si è rivelato leggermente più rigido, tale rigidezza trova la sua motivazione tanto in un numero di nodi superiore quanto nella risposta del modello costitutivo HSsmall. Questo ha comportato una lieve discrepanza dei profili di PGA.

9. ANALISI DI PROPAGAZIONE SISMICA 2D

Dopo aver verificato la corretta modellazione nel caso del fenomeno della propagazione sismica monodimensionale, si passa ora alla modellazione del fenomeno di propagazione sismica per il caso bidimensionale nell'ipotesi di stato di deformazione piana. Nello specifico, si è scelto di modellare il fenomeno di risposta sismica locale per un pendio naturale ideale, ispirato al caso del versante occidentale del comune di Chieuti (Foggia) in Puglia (di Lernia et al., 2023, Sonnessa et al., 2023, Tagarelli et al., 2025, Santaloia et al., in prep) sia in campo visco-elastico che visco-elasto-plastico. Per il medesimo pendio, sono stati sviluppati due modelli numerici, implementati in OPENSEES (McKenna et al., 2000) e in PLAXIS 2D (Brinkgreve et al., 2016).

In ragione dell'elevato numero di nodi presenti nella mesh di calcolo del modello implementato in OPENSEES, per l'esecuzione dell'analisi è stata utilizzata l'infrastruttura HPC con un numero di 10 CPU, sfruttando il framework di base per il funzionamento (Cavallo et al., 2024). Tuttavia, è stata possibile anche la risoluzione su personal computer con un numero di 4 CPU, seppur con un maggior tempo computazionale.

I risultati numerici ottenuti con il codice PLAXIS sono stati utilizzati come benchmark per validare la modellazione eseguita con il codice OPENSEES.

9.1 DEFINZIONE DEL MODELLO FEM 2D

Il pendio ideale implementato nelle simulazioni numeriche (Fig. 37) è costituito da due strati, uno di argilla di spessore pari a 50 m, sovrastante il bedrock sismico. Il pendio si estende per una lunghezza di 941.3 m e presenta un'altezza massima di 170.3 m. L'argilla è caratterizzata da resistenza a taglio non drenata c_u di 270 kPa e velocità delle onde di taglio V_S pari a 250 m/s, mentre il bedrock sismico è caratterizzato da V_S pari a 800 m/s. Si assume che il livello di falda sia situato in corrispondenza del piano campagna.

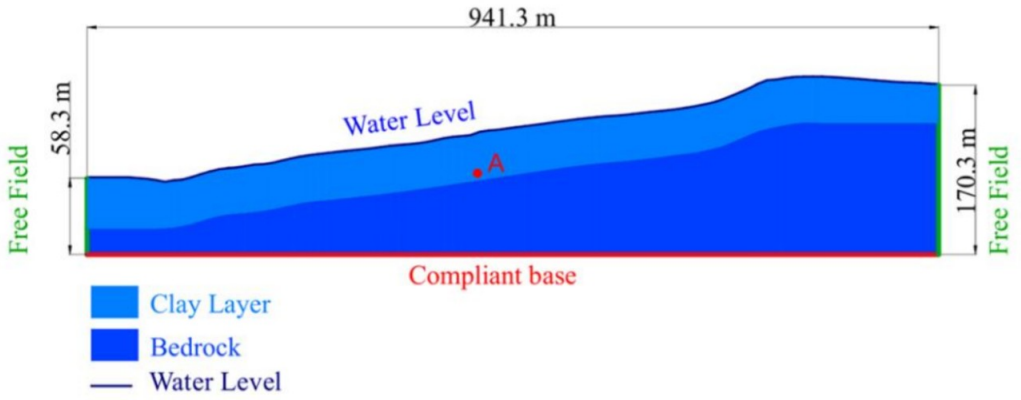


Fig. 37 Modello del pendio implementato nei codici FEM (Elia et al., 2025).

I modelli numerici implementati nei due codici FE sono riportati in Fig. 38 (Elia et al., 2025). Questi sono stati estesi lateralmente di 8 volte l'altezza del lato laterale per evitare qualsiasi interferenza dei contorni verticali con l'area di interesse.

Il modello PLAXIS 2D è stato discretizzato con 3157 elementi triangolari a 15 nodi, per un numero totale di nodi pari a 26258 (Fig. 38a). Invece, il modello di pendio OPENSEES, mostrato in Fig. 38b, consiste di 37500 elementi quadrilaterali a 4 nodi con un punto di Gauss al centro (SSPQuadUP), implementando la formulazione u-p per l'interazione solido-fluido in condizioni dinamiche (McGann et al., 2012).

Questi elementi sono stati selezionati per la loro stabilità numerica nell'integrazione in campo elasto-plastico. Il numero totale di nodi nel modello OPENSEES è pari a 30974. Nel caso delle analisi visco-elastiche con modello elastico isotropo sono stati applicati i parametri in Tab. 5.

Le caratteristiche dei materiali che costituiscono il pendio sono quelle definite nel Capitolo 7. Nello specifico, per il modello numerico implementato in OPENSEES, il comportamento dell'argilla è descritto dal modello costitutivo PIMY, i cui parametri sono quelli relativi al Set 1 di Tab. 3; il bedrock sismico è modellato come mezzo elastico lineare con V_S pari a 800 m/s. Nel modello PLAXIS, la risposta dell'argilla è descritta dal modello HSsmall, i cui parametri sono sintetizzati in Tab. 4.

Ai materiali assegnati agli strati è stato assegnato uno smorzamento viscoso alla Rayleigh, i cui parametri α_R e β_R sono stati determinati assegnando due diversi valori dello smorzamento target ξ.

Nello specifico, per lo strato di argilla il fattore di smorzamento target è assunto pari al 3%, mentre per il bedrock sismico il fattore di smorzamento è pari all'1%.

Per entrambi gli strati le frequenze di controllo sono state imposte pari a $f_1 = 1$ Hz e $f_2 = 10$ Hz. Tale intervallo è stato determinato in funzione di alcune necessità principali (Elia et al., 2021; Elia & Rouainia, 2022):

1. fornire ai materiali modellati un valore di smorzamento per bassi livelli di deformazione in quanto la curva di smorzamento del modello costitutivo nel caso del terreno ha un valore vicino a quello nullo, mentre per il bedrock sismico non vi è alcun smorzamento in quanto elastico;
2. smorzare le alte frequenze, superiori ai 10 Hz, al fine di evitare rumore numerico;
3. evitare effetti di risonanza nel dominio durante l'integrazione del segnale di input.

Per lo strato di argilla quindi si ottiene un fattore $\alpha_R = 0.3427$ e un fattore $\beta_R = 0.0008681$. Per lo strato assunto come bedrock sismico sono stati applicati fattori di Rayleigh pari ad $\alpha_R = 0.011424$ e un fattore $\beta_R = 2.89e\text{-}05$.

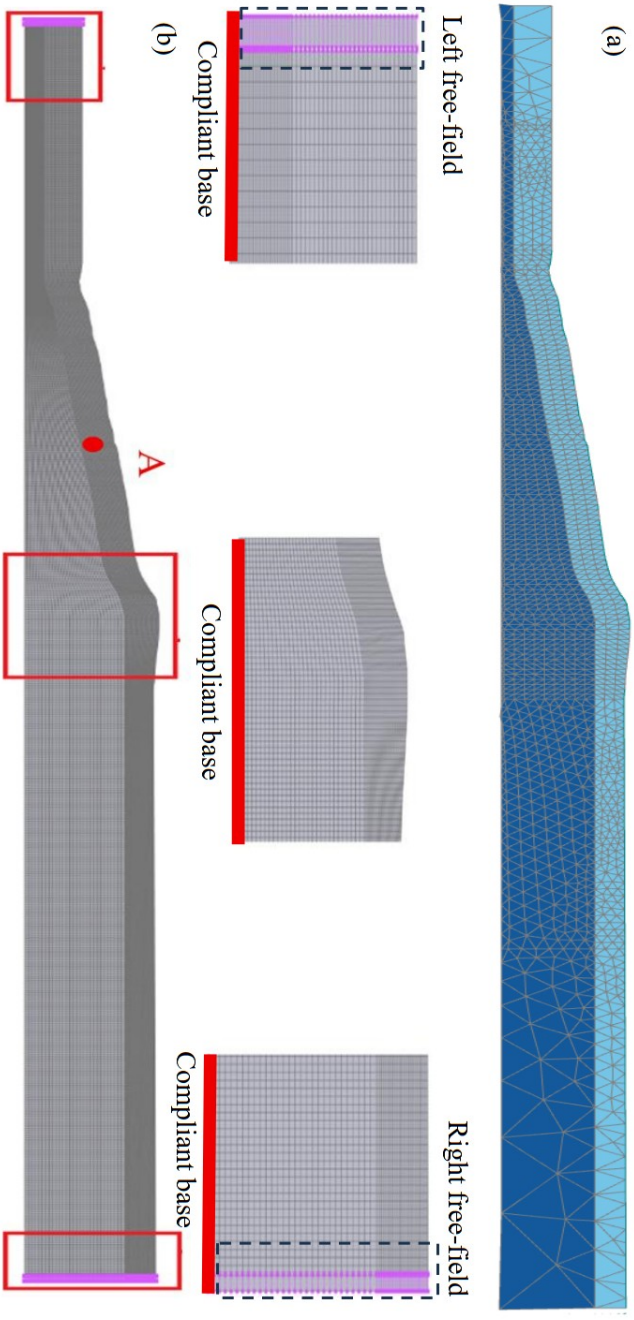


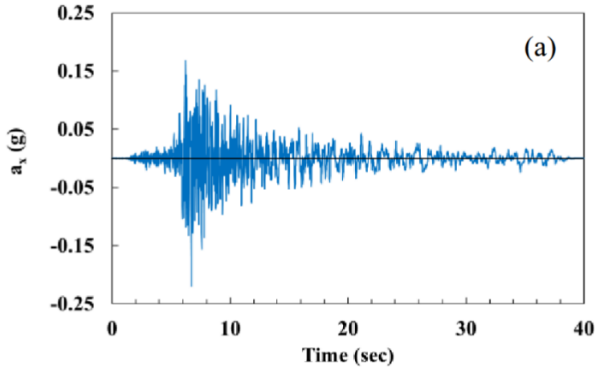
Fig. 38 Modelli FEM del pendio implementati nei due codici: a) PLAXIS 2D, b) OPENSEES (Elia et al., 2025).

Le simulazioni consistono in una fase statica iniziale, durante la quale lo stato tensionale viene inizializzato attraverso la procedura di gravity loading, seguita da una fase dinamica. Durante la fase statica, i bordi verticali del modello sono stati vincolati nella direzione orizzontale, mentre gli spostamenti dei nodi alla base sono nulli. Nella fase dinamica, sono state adottate condizioni al contorno di free-field lungo i lati verticali, mentre è stata imposta una compliant base lungo il bordo inferiore della mesh per simulare un substrato deformabile.

Nel modello OPENSEES, i contorni dinamici sono stati implementati manualmente, aggiungendo due colonne di terreno vincolate internamente con tied-nodes e connesse alla geometria principale con smorzatori lineari distribuiti lungo i bordi verticali del modello FE (Cavallo, 2024). Durante questa fase, i vincoli statici sono stati disattivati (Løkke & Chopra, 2018), mentre l'intera procedura di calcolo è stata eseguita utilizzando il calcolo parallelo con un solutore MUMPS.

Il metodo di integrazione temporale di Newmark è stato impiegato nelle simulazioni dinamiche adottando i coefficienti $\alpha_N = 0.5$ e $\beta_N = 0.25$, che non forniscono alcuno smorzamento numerico.

Il segnale sismico adottato nella fase dinamica è rappresentato dalla storia temporale delle accelerazioni, registrata alla stazione Melanico-Santa Croce di Magliano durante il terremoto verificatosi a Montecilfone (Molise) nell'agosto 2018. Il moto sismico, caratterizzato da PGA di 0.022g, è stato filtrato a una frequenza massima di 20 Hz, amplificato per un fattore di 10, per essere rappresentativo di un evento intenso, ed è stata applicata la “baseline correction” (Fig. 39).



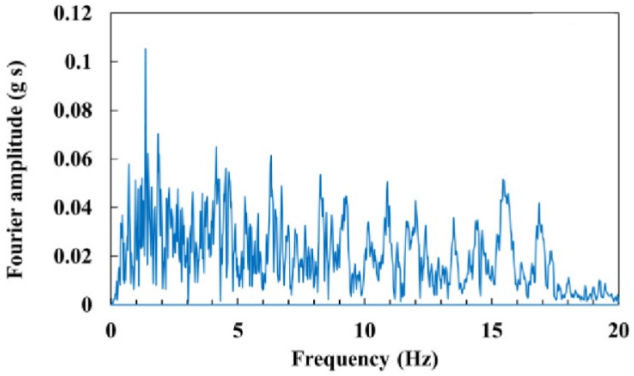


Fig. 39 Segnale in input: a) accelerogramma, b) spettro di Fourier.

Le analisi visco-elastiche e visco-elasto-plastiche dinamiche sono state eseguite per valutare la propagazione sismica del segnale nel dominio 2D e l'influenza della plasticità del terreno sulla propagazione delle onde sismiche nel continuo discretizzato, considerando i diversi modelli costitutivi implementati nei due codici FE.

La risposta dinamica del pendio è stata indagata anche con riferimento a 3 verticali, una di valle (x = 540.0 m), una di monte in corrispondenza della cresta (x = 1250.0) e una a metà tra le due definite (x = 870.0), come mostrato nella Fig. 40.

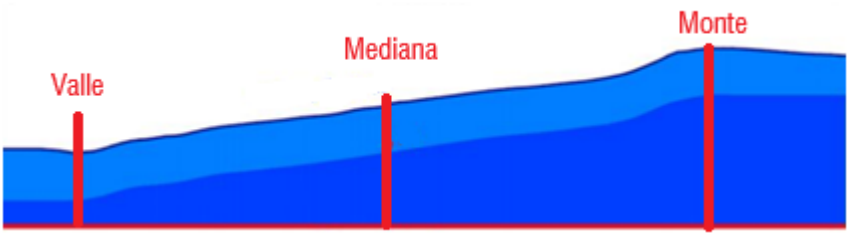


Fig. 40 Rappresentazione delle sezioni di indagine.

Lungo ogni verticale sono stati selezionati 10 punti per ricostruire il profilo delle accelerazioni massime ed il profilo delle deformazioni di taglio massime.
I risultati delle simulazioni numeriche sono illustrati in termini di storie temporali dell'accelerazione e corrispondenti spettri di Fourier, storie temporali della deformazione a taglio e della tensione di taglio, curve tensione-deformazione in campo visco-elastico

prima e visco-elasto-plastico dopo. Infine, sia per il caso elastico che per quello plastico si è definito il profilo delle accelerazioni massime al piano campagna.

9.2 RISULTATI DELLE ANALISI VISCO-ELASTICHE

I risultati delle simulazioni visco-elastiche eseguite con i codici PLAXIS 2D e OPENSEES sono riportati in **Appendice 4**, suddivisi per sezione di indagine (vedi Fig. 38). In particolare, si mostrano:

- storie temporali degli spostamenti orizzontali;
- storie temporali delle accelerazioni orizzontali;
- spettri di Fourier delle accelerazioni orizzontali;
- storie temporali delle deformazioni di taglio (scorrimenti orizzontali);
- diagrammi della risposta meccanica;
- profili delle accelerazioni massime e degli scorrimenti massimi;
- profili delle accelerazioni massime in superficie.

Dal confronto tra le analisi effettuate con i due codici, si nota un accordo più marcato tra i risultati in termini di spostamenti, piuttosto che in termini di accelerazioni e relativi spettri di Fourier.

A titolo di esempio, la Fig. 41 riporta la storia temporale degli spostamenti del punto a piano campagna in corrispondenza della sezione di monte, più critica in quanto luogo di maggiore amplificazione, riflessione e interferenza del segnale sismico.

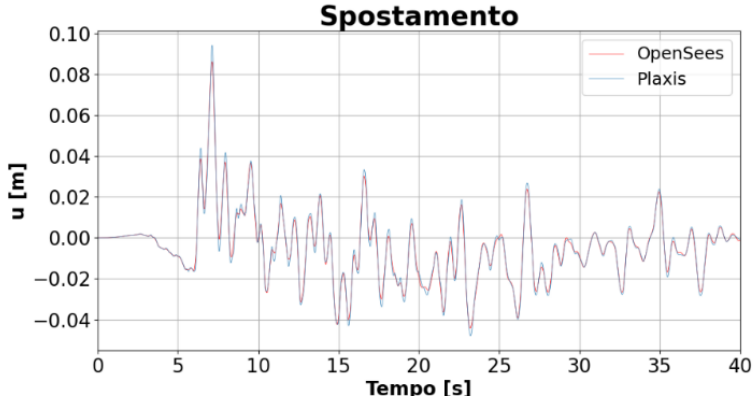


Fig. 41 Diagramma degli spostamenti in corrispondenza del piano campagna della sezione di monte per il caso visco-elastico.

In seguito, invece, si riportano i profili di accelerazione di picco per le sezioni caratteristiche definite in Fig. 42:

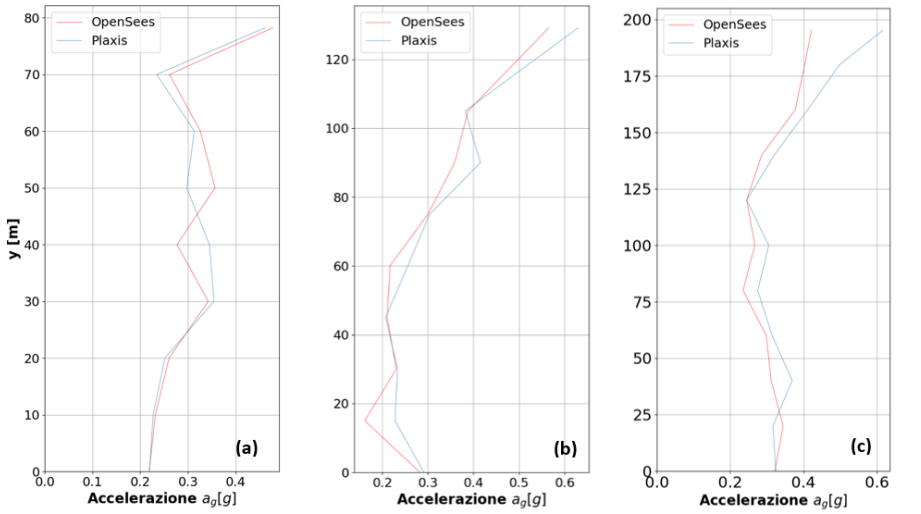
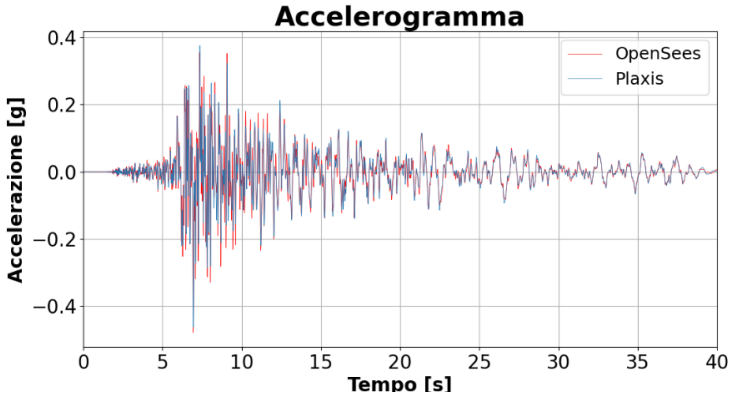
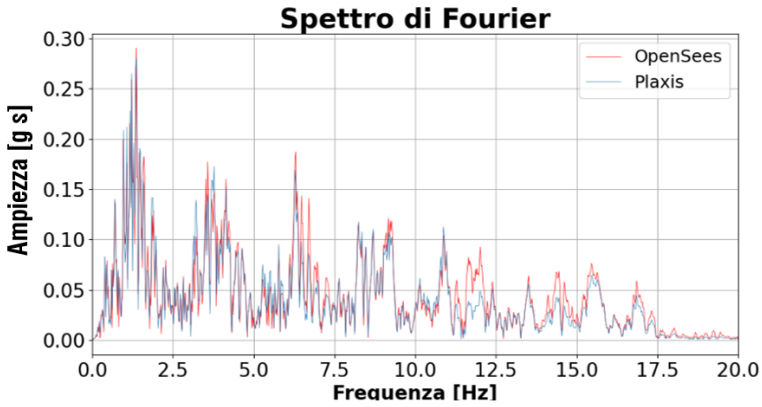


Fig. 42 Profili di accelerazione di picco, (a) sezione di valle, (b) sezione mediana, (c) sezione di monte

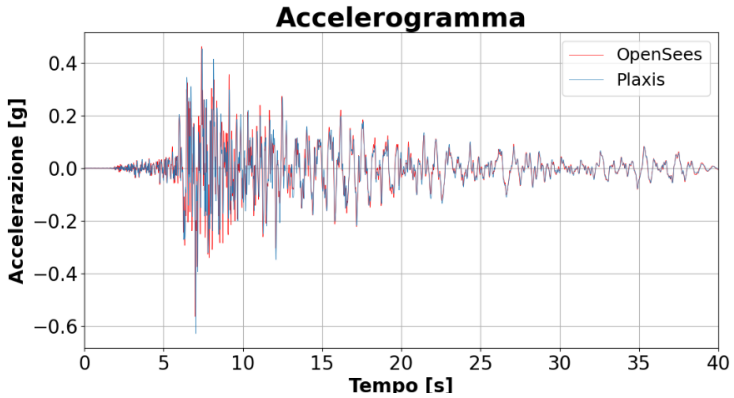
Nonostante questi profili abbiano dimostrato la stessa tendenza dei risultati, maggiori discrepanze tra i risultati sono evidenti nel confronto degli accelerogrammi e relativi spettri, se ne riportano in seguito i diagrammi relativi al punto a piano campagna delle tre sezioni caratteristiche definite:

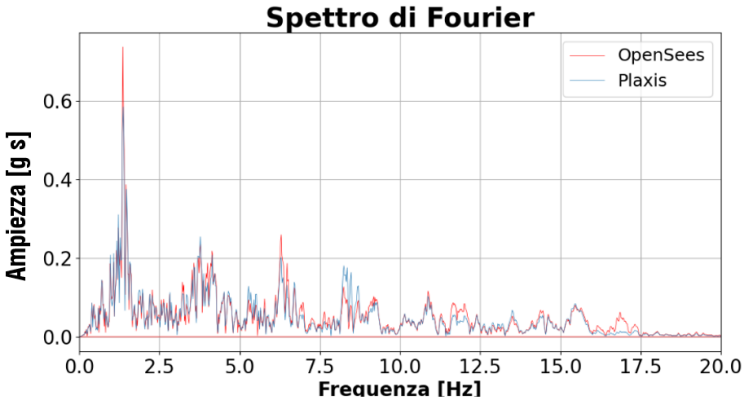
Sezione di valle (x = 540 m)





Sezione mediana (x = 870)





Sezione di monte (x = 1250)

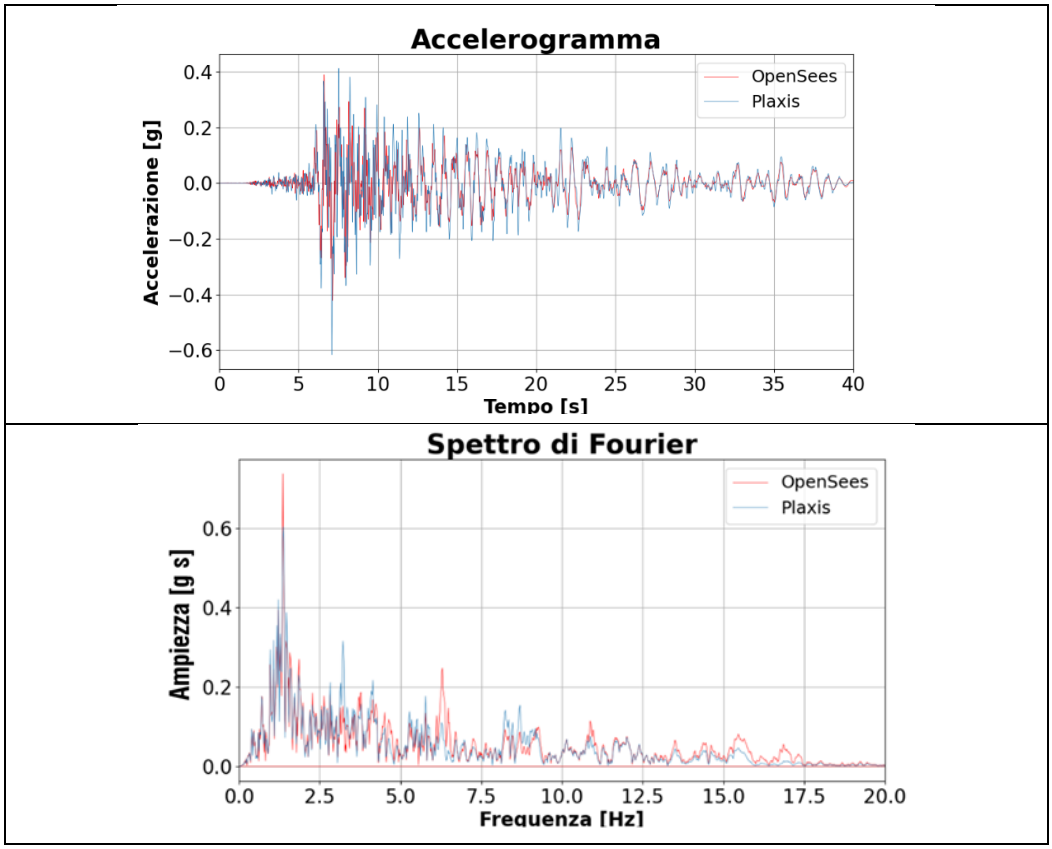


Fig. 43 Accelerogrammi e spettri di fourier per i punti a piano campagna delle sezioni caratteristiche di valle, centro e monte

Il profilo longitudinale delle accelerazioni massime in superficie in Fig. 44 dimostra un marcato disallineamento dei risultati:

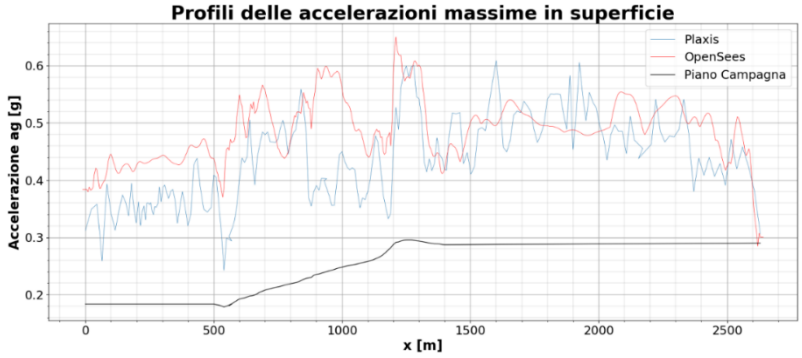


Fig. 44 Profilo longitudinale delle accelerazioni massime in superficie

In merito questo ultimo punto, è nota la forte dipendenza della soluzione dalla discretizzazione della mesh (Schmicker et al. 2014). Infatti, nel modello in OPENSEES sono stati utilizzati elementi finiti di forma quadrata lineari, mentre in PLAXIS sono stati utilizzati elementi finiti triangolari a 15 nodi. Una diversa configurazione della discretizzazione può comportare un risultato non perfettamente coincidente in termini di accelerogrammi e spettri di Fourier, che quindi saranno confrontabili solo in termini di forma ed ordine di grandezza.

Tale divergenza viene attenuata nei casi in cui la geometria del problema è caratterizzata da condizioni di simmetria, come nel caso della condizione di banco orizzontale in condizioni di free-field (condizione K_0) o propagazione monodimensionale (Latijrelle, 1989). Ad analoga considerazione si giunge variando l'ordine di integrazione degli elementi finiti nello stesso software PLAXIS 2D, passando da elementi a 15-nodi ad elementi a 9-nodi in una geometria discretizzata che non presenta simmetrie.

I risultati, non mostrati qui per brevità, mettono in evidenza come le alte frequenze ($f > 10$ Hz) sono quelle che più amplificano questo effetto, producendo rumore numerico e risonanze difficilmente smorzabili.

In ultimo, si deve considerare l'interazione con la falda e la diversa integrazione delle pressioni interstiziali per i tipi di elementi finiti selezionati nei due codici.

9.3 RISULTATI DELLE ANALISI VISCO-ELASTO-PLASTICHE

In **Appendice 5** vengono mostrati i risultati delle analisi visco-elasto-plastiche, eseguite adottando i due modelli costitutivi differenti. Ne consegue che ci si aspettano differenze tra i risultati di propagazione delle due analisi, in quanto il deposito di terreno funge da filtro e modifica in maniera diversa nei due casi il segnale originariamente trasferito dal bedrock sismico.

Come nel caso visco-elastico, la Fig. 40 mostra, a titolo di esempio, il risultato del confronto in termini di serie temporale degli spostamenti del punto in corrispondenza del piano campagna nella sezione di monte.

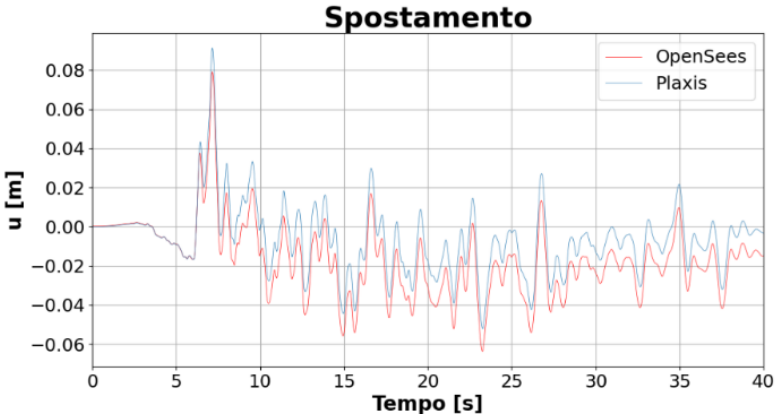


Fig. 45 Diagramma degli spostamenti in corrispondenza del piano campagna della sezione di monte per il caso visco-elasto-plastico.

In aggiunta alle considerazioni relative al confronto dei risultati visco-elastici, nelle analisi visco-elasto-plastiche si aggiunge l'effetto del diverso smorzamento isteretico introdotto dai modelli costitutivi considerati che introduce un differente accumulo di spostamento residuo.

Si riporta a titolo d'esempio l'accelerogramma, il relativo spettro di fourier e la risposta meccanica, per un punto all'interfaccia tra i due materiali per la sezione di valle:

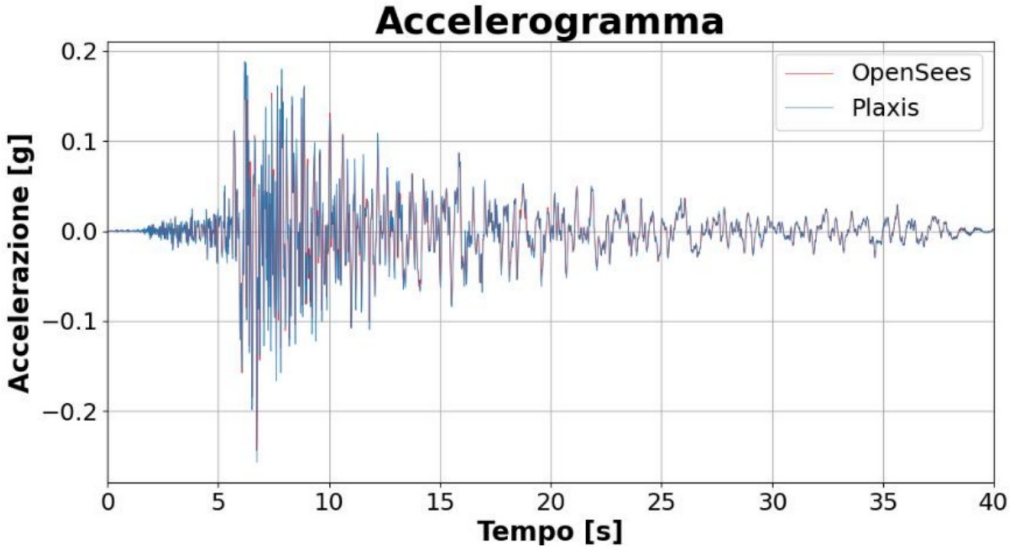


Fig. 46 Confronto degli accelerogrammi in corrispondenza del punto all'interfaccia terreno/bedrock

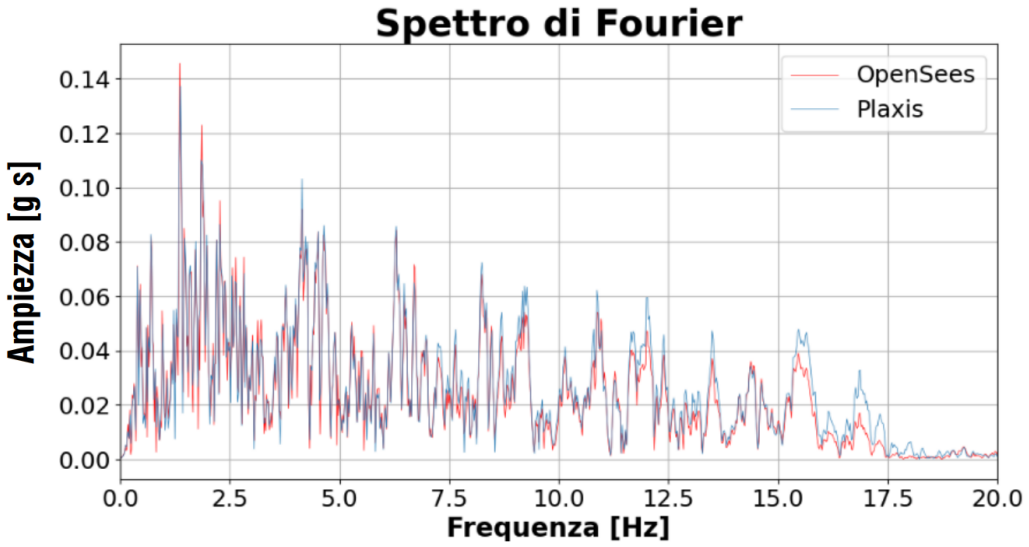


Fig. 47 Confronto degli spettri di risposta in corrispondenza del punto all'interfaccia terreno/bedrock

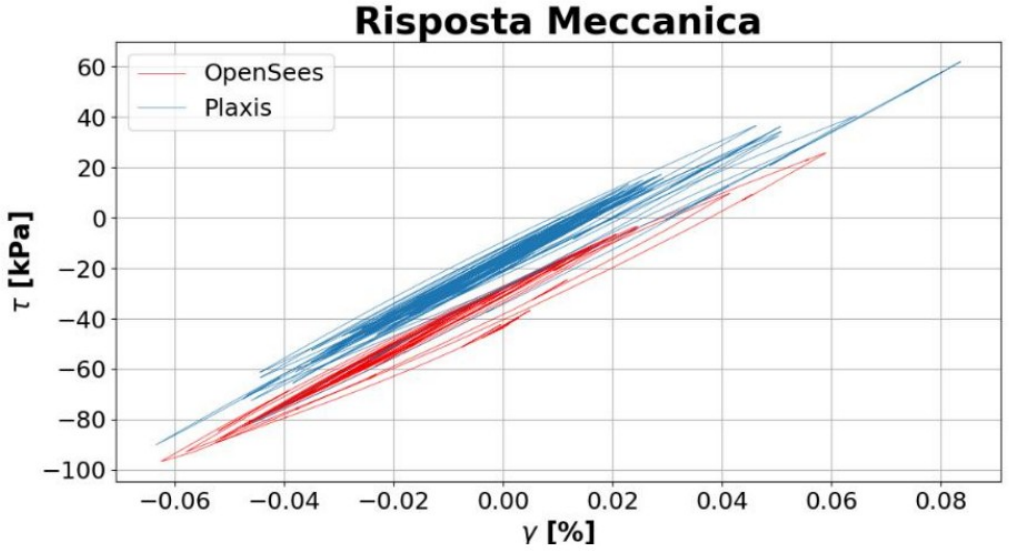


Fig. 48 Confronto della risposta meccanica (PIMY/HSsmall) all'interfaccia terreno/bedrock

Il confronto delle storie temporali dell'accelerazione e dei corrispondenti spettri di Fourier indica un buon accordo tra le due previsioni FEM, sia in termini di ampiezza

che di contenuto in frequenza. In particolare, si osserva che la plasticità introdotta in entrambi i modelli, con i propri modelli costitutivi, ha prodotto lo stesso slittamento delle frequenze fondamentali. Questa deduzione si evince dalla coincidenza dei picchi nel confronto degli spettri e una certa omotetia delle funzioni di amplificazione confrontate in funzione della frequenza pertanto questo dimostra che i modelli elaborati integrano in modo vero-simile e sono quindi paragonabili così come nel caso visco-elastico vi è la coincidenza degli spostamenti (incognita dei metodi compatibili).

Una leggera discrepanza nel modulo dell'ampiezza (g s) può essere osservata ad alte frequenze (> 10 Hz), correlata alle diverse caratteristiche della mesh di elementi finiti e ai modelli costitutivi adottati nonché all'integrazione delle sovrapressioni.

Le storie temporali della deformazione a taglio mostrano chiaramente una netta differenza nella risposta delle due leggi costitutive come si vede nella storia temporale delle deformazioni a taglio all'interfaccia terreno/bedrock nella sezione di valle che si riporta in seguito:

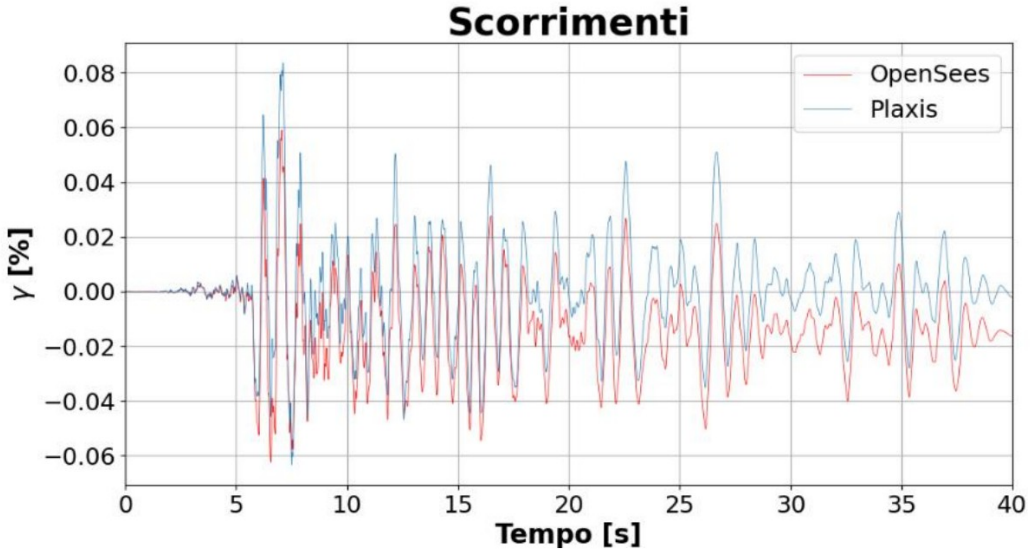


Fig. 49 Confronto della storia temporale delle deformazioni a taglio all'interfaccia terreno/bedrock nella sezione di valle

Infatti, il modello PIMY tende a esibire un comportamento elasto-plastico più pronunciato rispetto al modello HSsmall, che si riflette nell'accumulo di deformazioni plastiche maggiori. Inoltre, le curve tensione-deformazione mostrano cicli di isteresi più sottili previsti dal modello HSsmall, indicativi di una risposta più rigida e meno dissipativa. Al contrario, il modello PIMY fornisce un accumulo più evidente di deformazioni plastiche, che si riflette in cicli di isteresi più ampi e associato smorzamento isteretico più elevato. Si noti che le caratteristiche di incrudimento cinematico del modello PIMY implicano l'esibizione di ratcheting nella risposta dinamica.

A valle delle considerazioni in campo visco-elastico e visco-elasto-plastico si ripropone il confronto dei profili delle accelerazioni massime determinate a piano campagna in campo visco-elastico ed in campo visco-elasto-plastico:

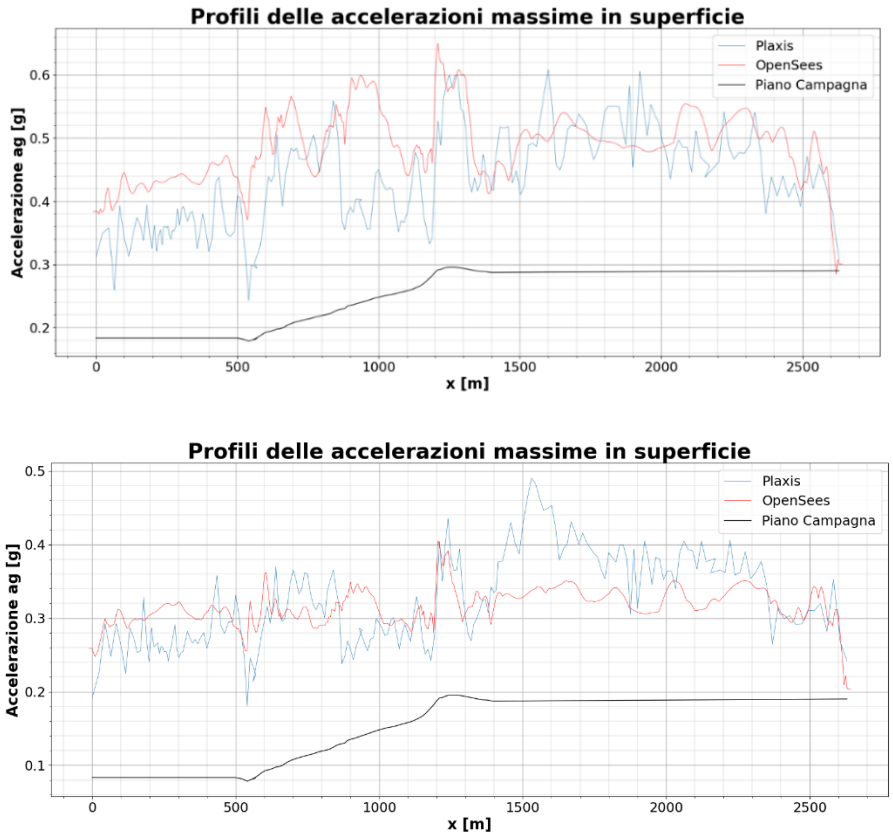


Fig. 50 Confronto dei profili di accelerazione massima in superficie

Il confronto in campo visco-elasto-plastico rispetto al campo visco-elastico dimostra:

- una riduzione del modulo delle accelerazioni massime
- il mantenimento della tendenza del modulo delle accelerazioni nelle zone di amplificazione (positivo e negativa)
- L'allontanamento dei bordi non ha prodotto un andamento regolare del modulo delle accelerazioni massime in superficie tuttavia ha prodotto uno smorzamento degli effetti singolari numerici in quanto in cresta si osserva la coincidenza dei moduli sia in campo elastico che in campo plastico

10. ANALISI DI PROPAGAZIONE SISMICA 3D

A valle delle analisi e degli effetti riscontrati dal confronto dei risultati dei modelli di propagazione 1D e 2D illustrati finora, si passa alla modellazione 3D mediante il codice di calcolo OPENSEES, eseguita ispirandosi al medesimo versante occidentale del comune di Chieuti.

10.1 DEFINIZIONE DEL MODELLO FEM 3D

Il modello numerico è stato costruito partendo dal DTM mostrato in Fig. 4, considerando un volume di area 1000 m x 1000 m ed un'altezza massima di circa 220 m. Esso include 2 strati di terreno, uno rappresentativo dell'Unità 3 delle Argille di Montesecco e l'altro rappresentativo del bedrock sismico. L'interfaccia tra i due strati è stata ottenuta traslando il piano campagna verso il basso ad una profondità di 50 m. Il livello di falda è stato considerato coincidente con il piano campagna. Il comportamento del terreno è simulato con il modello PIMY, calibrato come descritto nel Capitolo 7, mentre per il bedrock sismico si è assunta l'ipotesi di mezzo visco-elastico lineare. Al fine di creare le condizioni di free-field, sono stati estrusi i 4 lati del volume per una lunghezza nella direzione perpendicolare alla faccia i-esima di 200 m che è circa la media dell'altezza variabile per ogni lato. Quindi ogni volume risultato dell'estrusione è stato collegato al dominio di studio per mezzo di smorzatori lineari. Infine, alla base del modello è stata imposta la compliant-base.

Il modello è stato discretizzato con elementi esaedrici a 8 nodi stabilizzati (SSPBrickUP) con formulazione u-p. Il modello 3D, riportato in Fig. 51, conta 95172 elementi e 73682 nodi. Le fasi di calcolo implementate nella simulazione 3D sono le stesse eseguite per la modellazione 2D:

- inizializzazione con gravity loading;
- fase dinamica totalmente accoppiata (cfr. Cap. 9).

In questo caso il modello 3D è stato risolto in parallelo con 4 CPU per un tempo di calcolo di circa 3 ore.

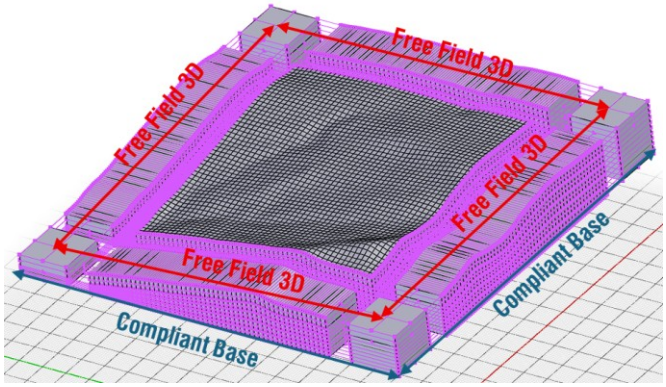


Fig. 51 Modello agli elementi finiti 3D implementato in OPENSEES e condizioni al contorno.

Al fine del calcolo multi-processore o multi-core è necessario dividere il modello matematico nel numero disponibile di core della macchina che eseguirà le operazioni di inversione delle sottomatrici. Quindi la matrice delle rigidezze verrà divisa in n sottomatrici ed ognuna verrà assegnata all' n-esimo processore (o core nel caso dei personal computer).

Lo schema di partizionamento della mesh di calcolo è illustrato in Fig. 52. Tale partizionamento è eseguito e ottimizzato con libreria METIS al fine di evitare sovraccarichi su singoli processori (cfr. Cap 3). Tale procedura velocizza l'inversione della matrice di rigidezza globale effettuando più operazioni nello stesso intervallo di tempo.

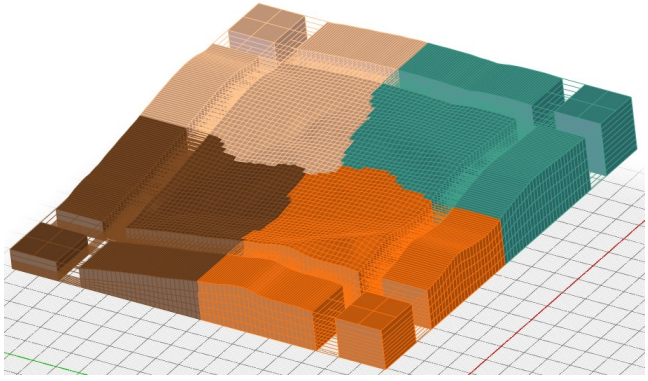
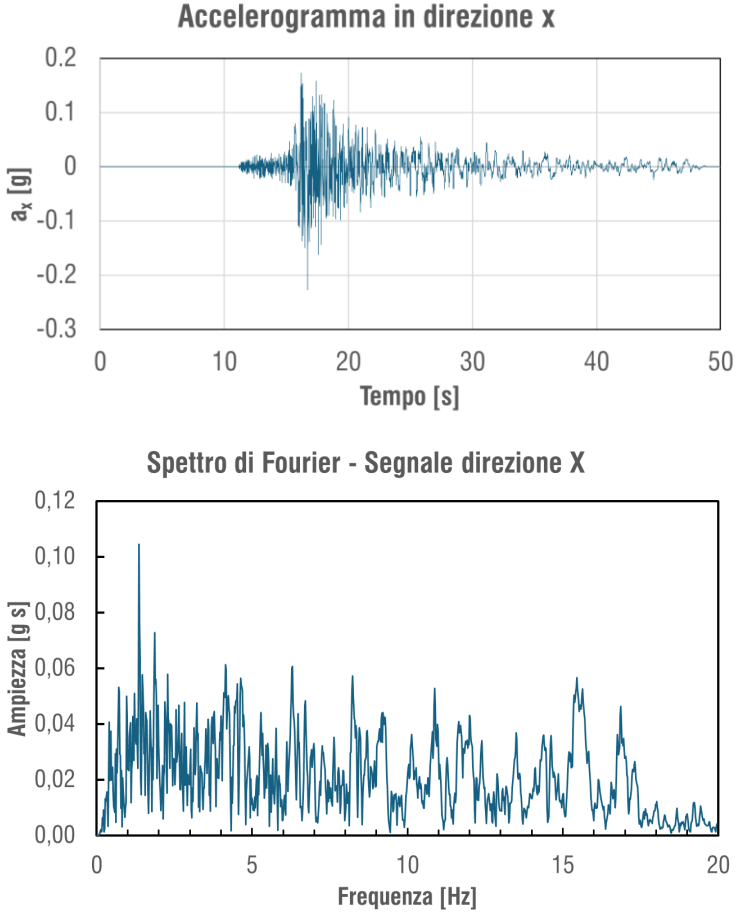


Fig. 52 Schema di partizionamento della mesh per 4 CPU.

Alla base del modello sono state applicate le due componenti nelle direzioni x e y dell'evento registrato alla stazione Melanico-Santa Croce di Magliano durante il terremoto verificatosi a Montecilfone (Molise) nell'agosto 2018. Il moto sismico è caratterizzato da PGA di 0.022g lungo la direzione x e di 0.017g lungo la direzione y. I segnali sono stati filtrati a una frequenza massima di 20 Hz ed amplificati per un fattore di 10, per essere rappresentativi di un evento sismico intenso. Inoltre, è stata applicata la "baseline correction" (Fig. 53).



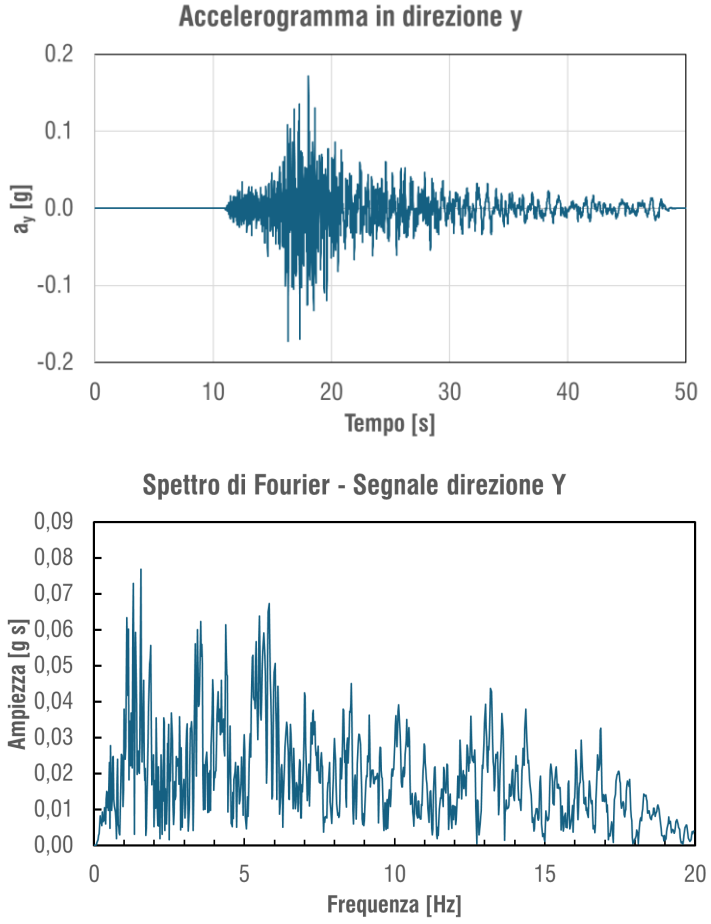


Fig. 53 Accelerogrammi e rispettivi spettri di Fourier assegnati alla base in direzione x e y.

10.2 RISULTATI DELLE SIMULAZIONI NUMERICHE (4 CPU)

In questo paragrafo si mostrano alcuni risultati ottenuti dall'analisi dinamica totalmente accoppiata soggetta ad un segnale alla base bidirezionale.

In Fig. 58 si mostrano le pressioni interstiziali geostatiche che mostrano una distribuzione corretta delle pressioni neutre. Questo risultato indica una corretta modellazione e disposizione della mesh in quanto si è ottenuto un diagramma continuo ed uniforme.

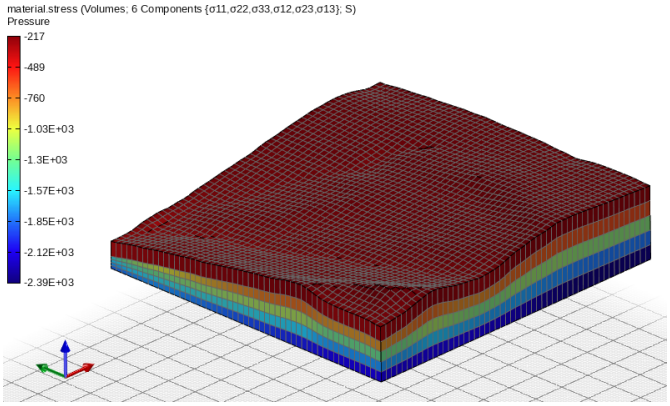
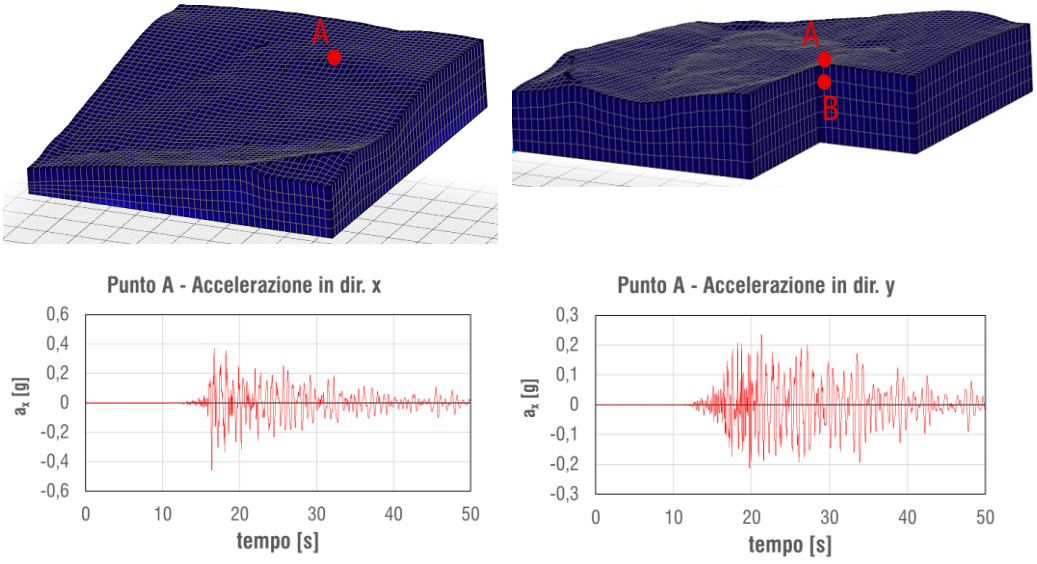


Fig. 54 Distribuzione delle pressioni interstiziali alla fine dell'analisi geostatica.

In relazione ai risultati ottenuti per il caso bidimensionale si sono ottenuti gli output in corrispondenza della zona in cui si ha la maggiore amplificazione. Si riporta quindi il punto A in superficie e il punto B in corrispondenza dell'interfaccia tra i due strati (Fig. 60)



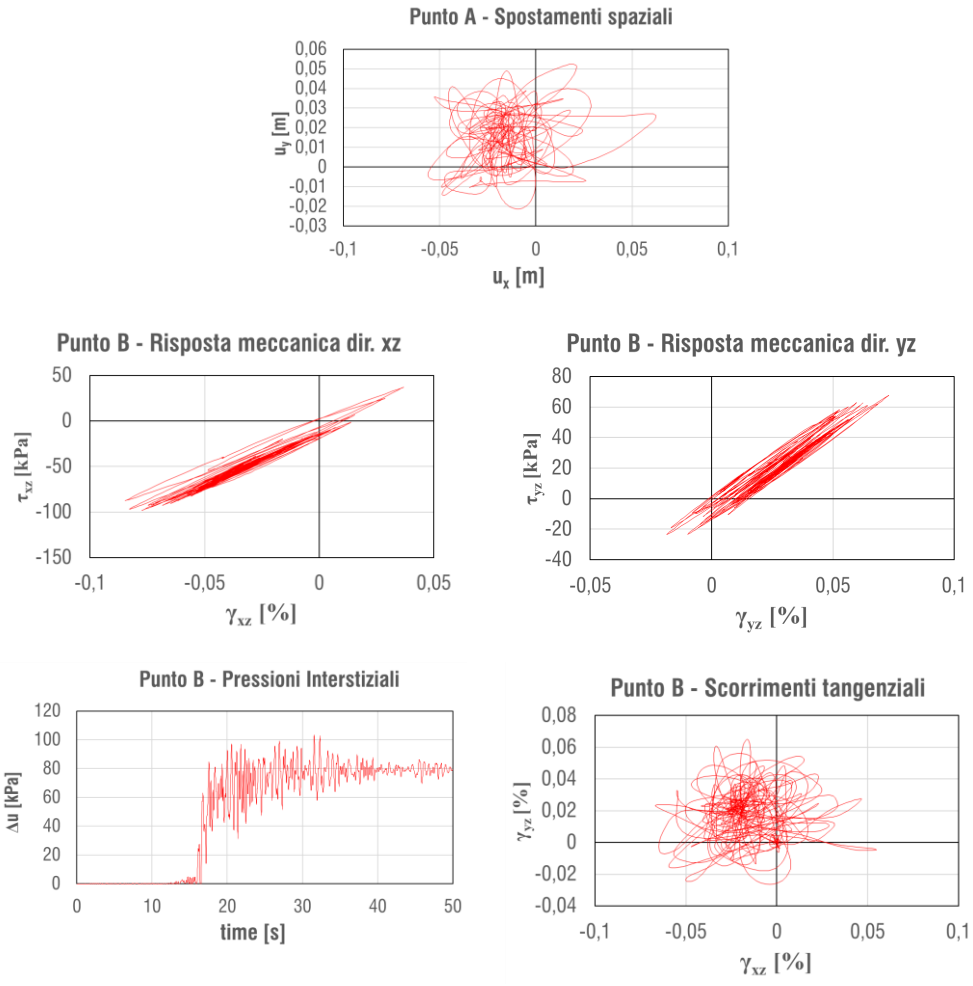


Fig. 55 Output per i punti A e B.

In Fig. 60 si riportano anche le accelerazioni in direzione x e y e gli spostamenti nello spazio del punto A. Per il punto B, in materiale visco-elasto-plastico, si mostrano le risposte meccaniche in direzione x e y, la risposta spaziale delle deformazioni tangenziali e le sovrapressioni interstiziali. I risultati in termini di accelerazioni nel punto in A dimostrano l'assenza di errori e instabilità (spike) indice di una corretta integrazione. I risultati in B dimostrano un accumulo di deformazioni in correlazione ad un accumulo di sovrapressioni interstiziali. Pertanto, il comportamento dimostra una buona modellazione.

Attraverso elaborazione dati si sono ottenute le accelerazioni massime dei punti in superficie per la direzione x e y. Tali accelerazioni sono state normalizzate per le PGA dei segnali assegnati in input rispettivamente per le due direzioni, ottenendo le mappe dei fattori di amplificazione in superficie mostrate nella Fig. 46.

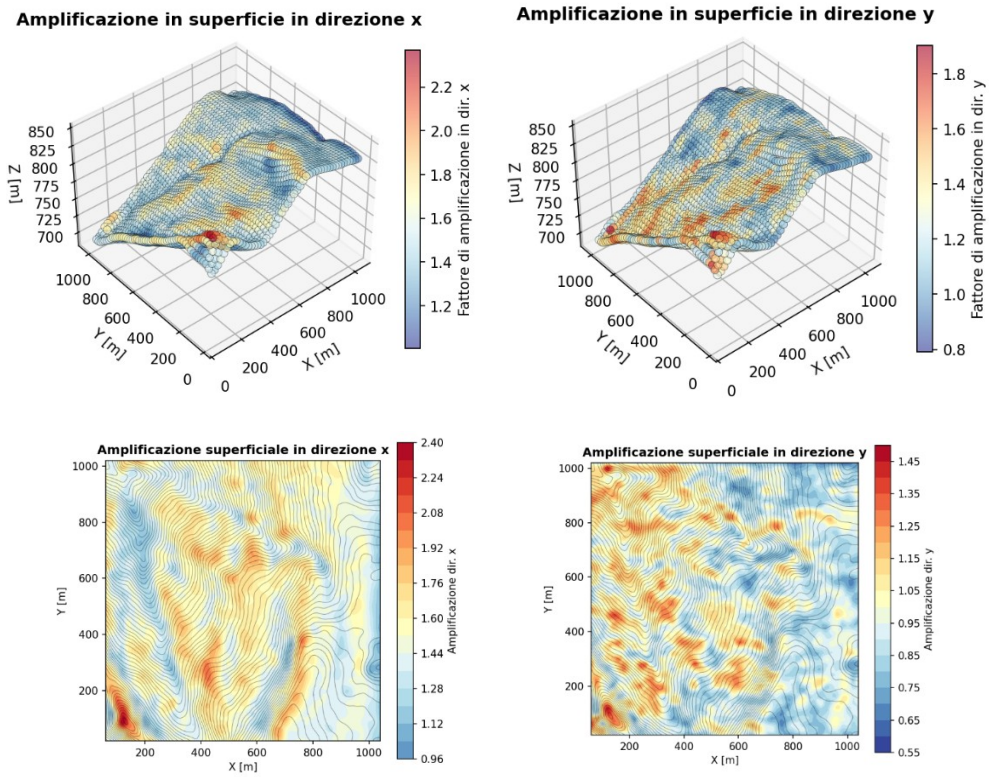


Fig. 56 Visione 3D e mappe delle amplificazioni superficiali in basso.

La mappa delle amplificazioni in direzione x (Fig. 65) rappresenta un buon risultato preliminare, in quanto si nota una bassa amplificazione delle zone di valle ed una più alta in corrispondenza dei promontori.

La mappa dell'amplificazione in direzione y (Fig. 65) mostra valori del fattore di amplificazione di modulo inferiore rispetto a quelli in direzione x.

Tali mappe possono essere utilizzate ai fini dell'analisi di pericolosità in quanto delineano le aree in cui si ha una maggiore amplificazione sismica rispetto ad altre.

Il limite principalmente riscontrabile in questa analisi è rappresentato da una mesh di dimensioni troppo elevate per cui è avvenuto un filtro delle frequenze massime trasmesse in superficie, in base alla seguente relazione:

$$f_{max} = \frac{V_s}{(6 \div 8)h} = \frac{250\ m/s}{8 \cdot 25\ m} = 1.25\ Hz \quad (75)$$

per cui le frequenze superiori a 1.25 Hz non sono rappresentate nei risultati dell'analisi. Quindi si ritiene necessario applicare un forte infittimento della mesh con una richiesta di capacità computazionali superiori.

Nonostante questo limite, in base a considerazioni di carattere qualitativo e dal confronto con i risultati del caso 2D, è comunque possibile riconoscere le zone di compluvio di valle e cresta come zone di amplificazione. Questo effetto è riscontrabile, infatti, in entrambe le mappe mostrate in Fig. 65.

10.3 RISULTATI DELLE SIMULAZIONI NUMERICHE (90 CPU)

In questo paragrafo si mostra lo stesso caso analizzato nel paragrafo precedente ma con una mesh più risoluta e risolto con un numero di CPU pari a 90 su centro di calcolo HPC RECAS. In seguito si mostra il partizionamento della mesh per il numero di CPU imposto al fine della risoluzione.

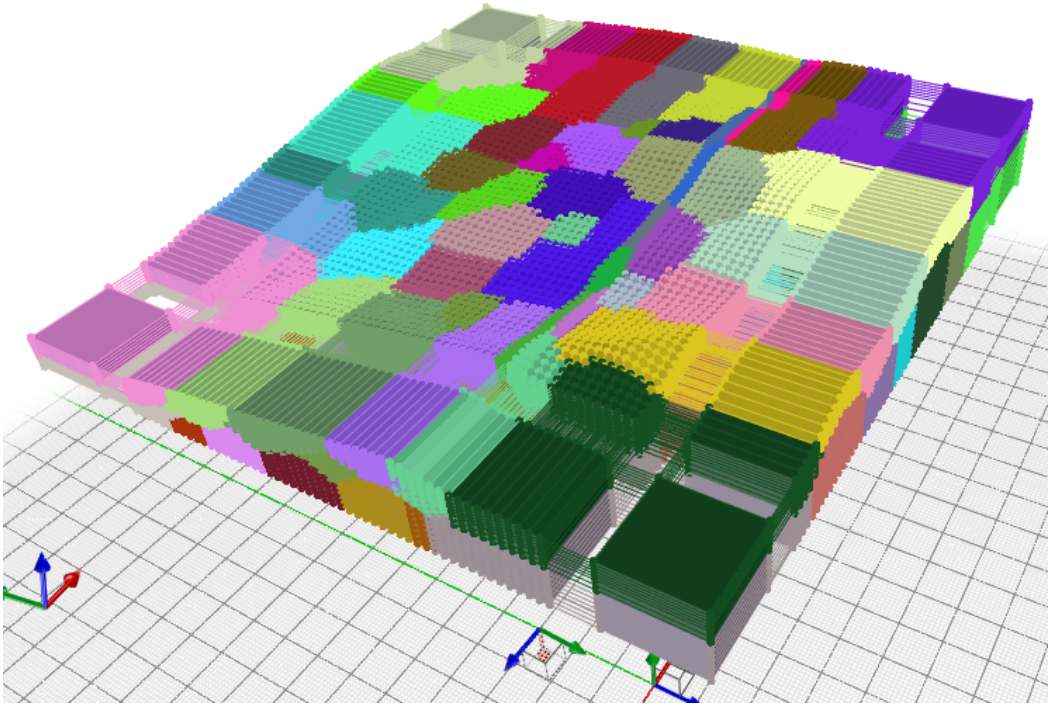


Fig. 57 Partizionamento della mesh per il numero di CPU imposto (90).

In Fig. 58 si mostrano le pressioni interstiziali idrostatiche che confermano una distribuzione corretta delle pressioni neutre. Anche in questo caso, questo risultato indica una corretta modellazione e disposizione della mesh in quanto si è ottenuto un diagramma continuo ed uniforme.

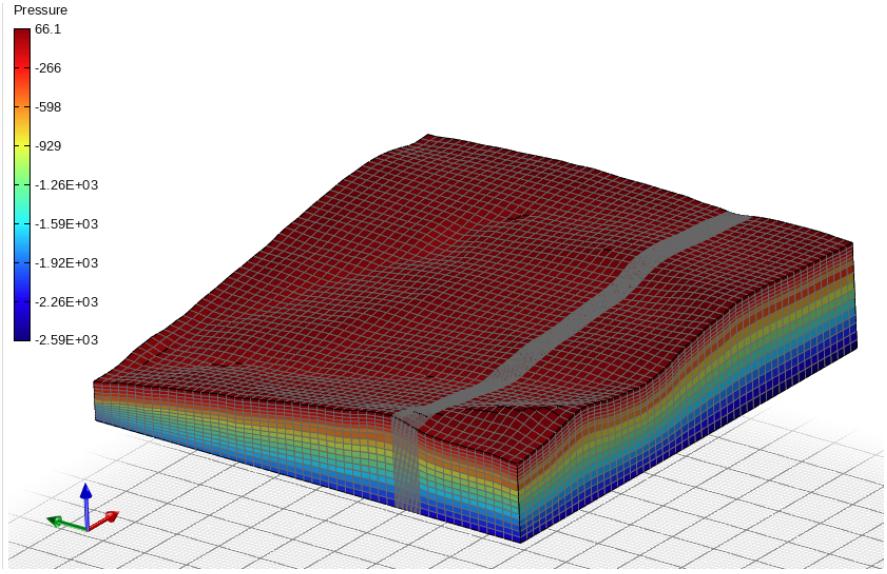


Fig. 58 Distribuzione delle pressioni interstiziali alla fine dell'analisi geostatica.

In relazione ai risultati ottenuti per il caso bidimensionale si sono ottenuti gli output in corrispondenza della zona in cui si ha la maggiore amplificazione.

Si riportano a seguire i risultati cinematici in termini di accelerogrammi, relativi spettri e spostamenti x y dei punti indicati in Fig. 59:

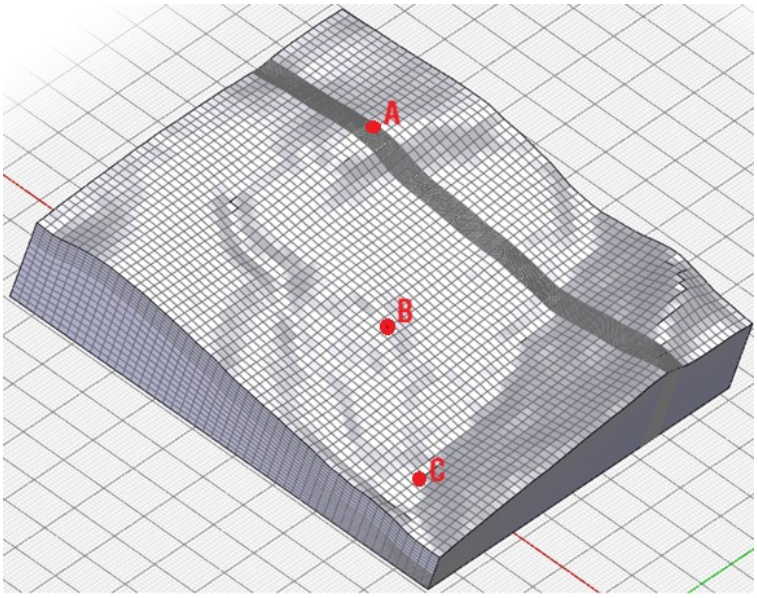


Fig. 59 Selezione dei punti in superficie

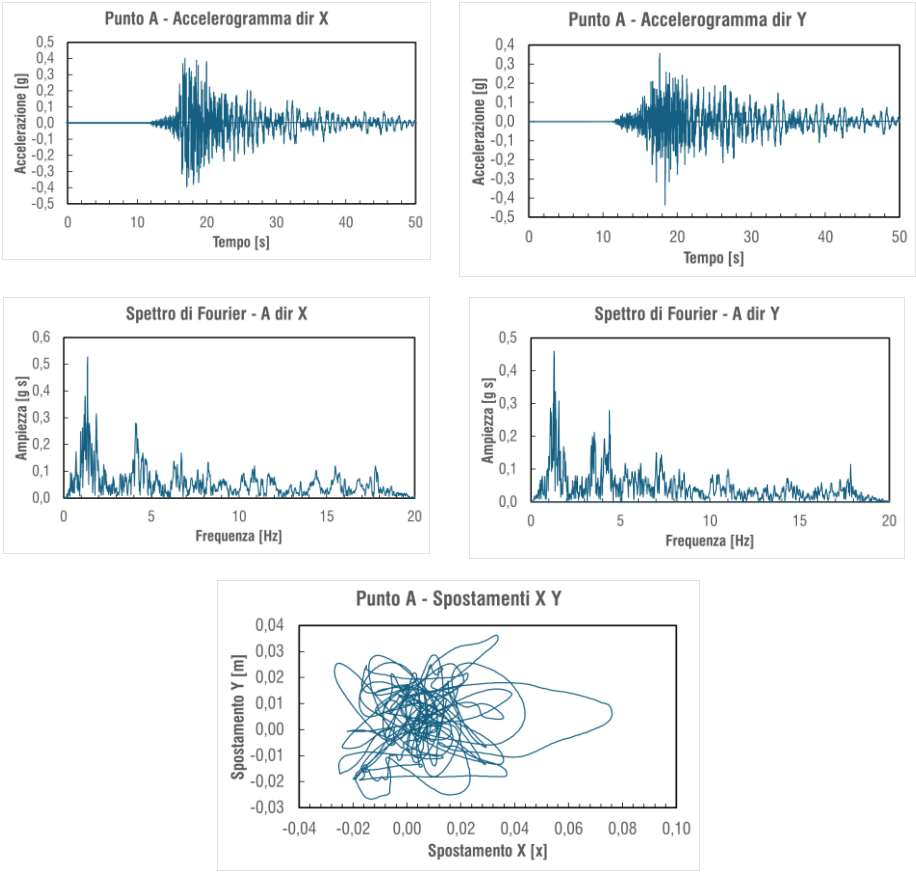
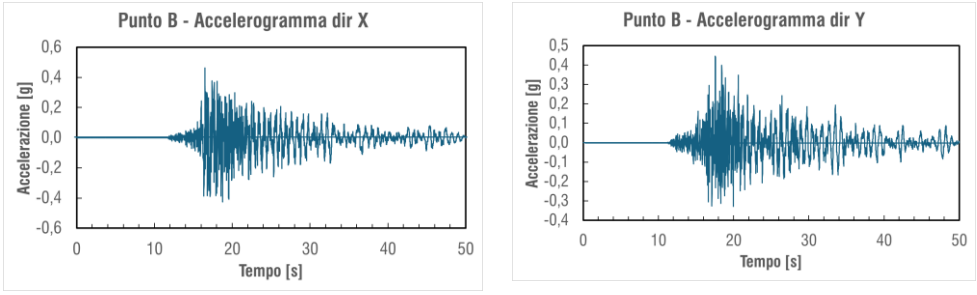
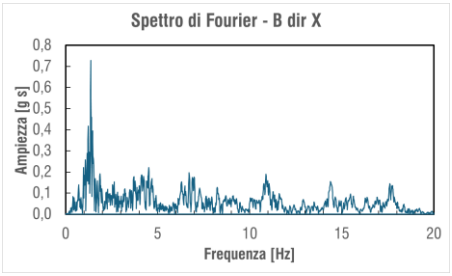
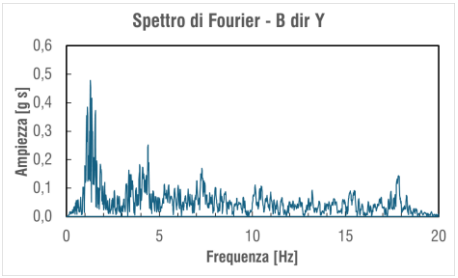


Fig. 60 Output per il punto A







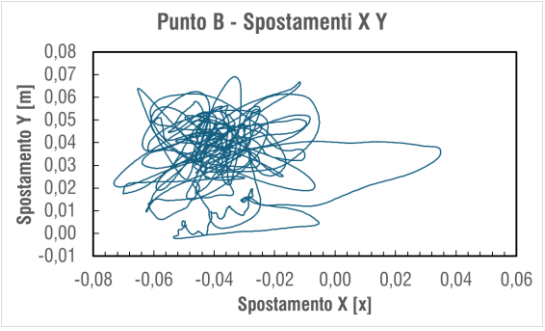
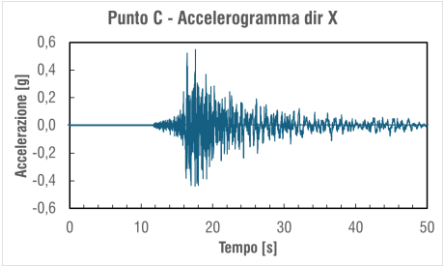
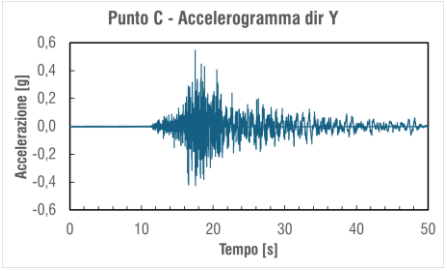
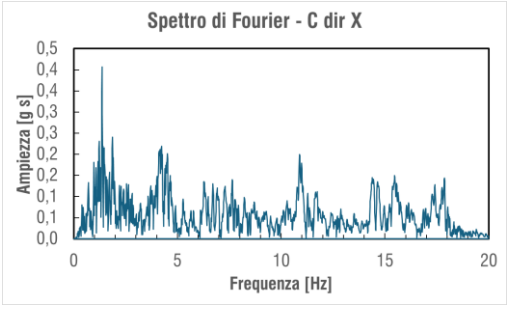
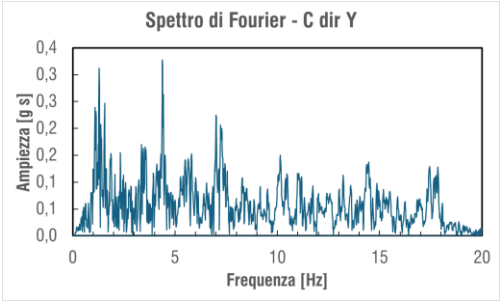


Fig. 61 Output per il punto B









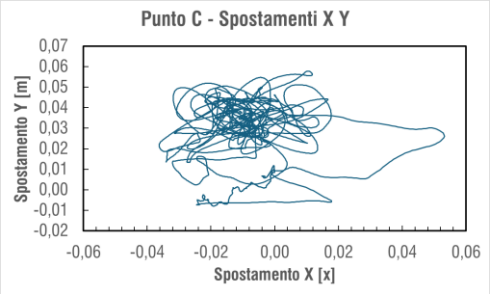


Fig. 62 Output per il punto C

I risultati in termini di accelerazione dimostrano l'assenza di errori e instabilità (spike) indice di una corretta integrazione. Inoltre, si nota l'amplificazione del segnale di input in superficie ed in particolare si amplifica la frequenza: 1,3 – 1,35 Hz. Questo effetto è comune in entrambe le direzioni.

Nella zona di valle, invece, (Punto C) si nota anche l'amplificazione della frequenza 4.39 Hz nella sola direzione Y.

Tali risultati indicano l'influenza della tridimensionalità e del passaggio di strato nella diffusione del segnale all'interno del dominio. Dall'analisi degli spostamenti nel piano si nota una prevalenza dello spostamento lungo X rispetto a quello lungo Y. Infatti, il segnale imposto in input alla base del modello lungo X ha una PGA maggiore del segnale imposto nella direzione Y.

In seguito, si riportano gli output delle risposte meccaniche nelle due direzioni xz ed yz presi nella stessa posizione in pianta come in Fig. 59 ma in corrispondenza dell'interfaccia terreno/bedrock. Tali punti vengono indicati con A', B', C'.

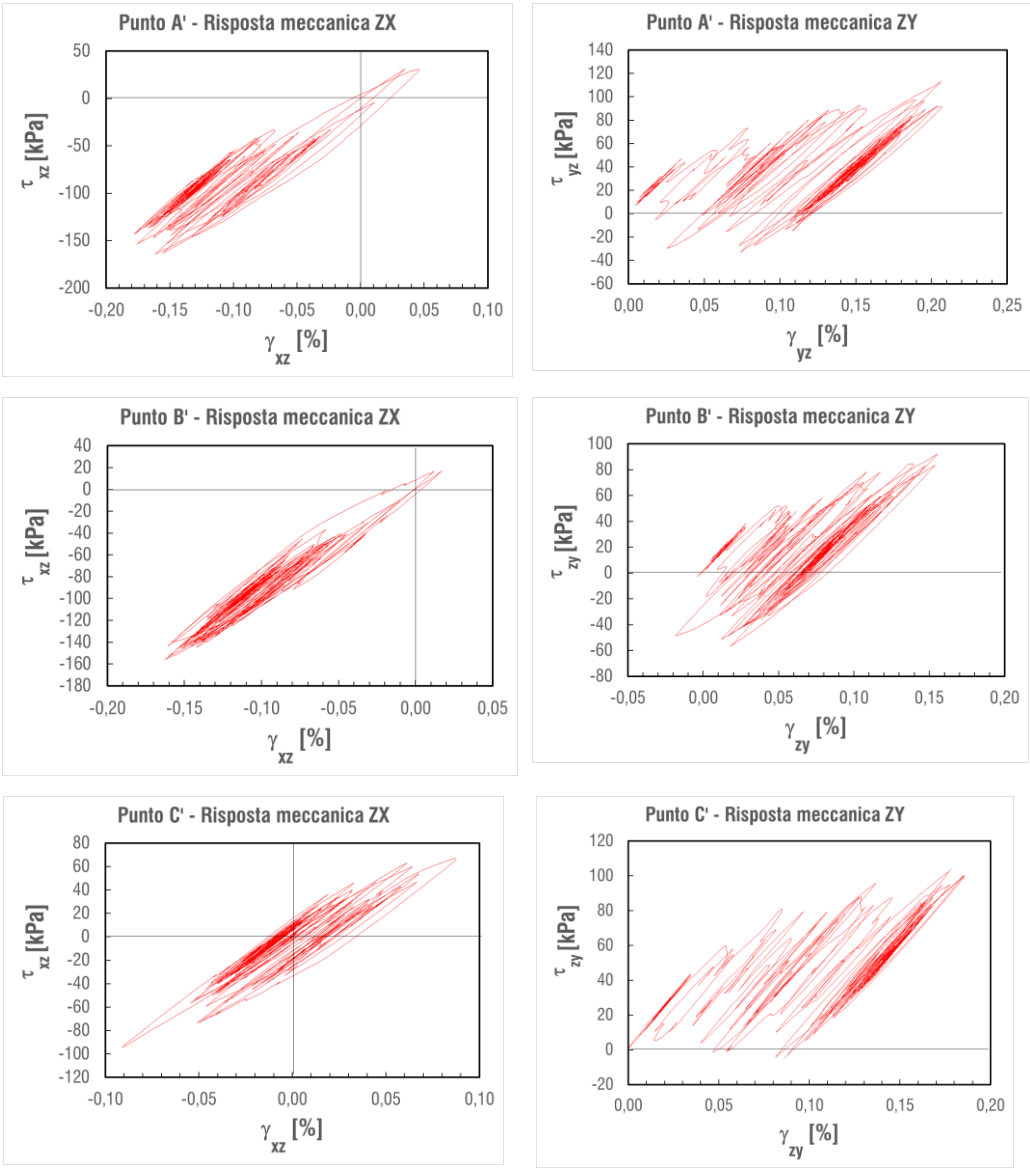
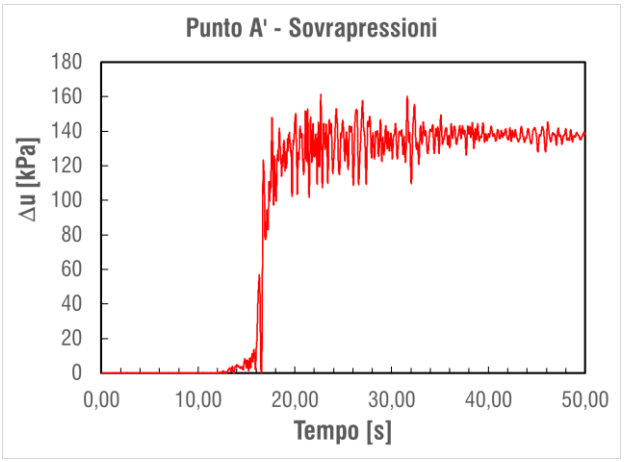
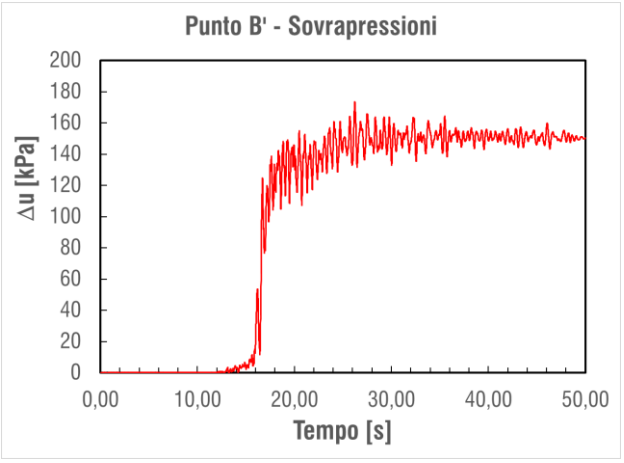


Fig. 63 Risposte meccaniche interfaccia terreno/bedrock

Ciò che emerge dall'analisi degli output in Fig. 63 è una marcata plasticità. I cicli di isteresi sono ampi e quindi dissipativi (effetto voluto). Pertanto, ad uno smorzamento viscoso si aggiunge uno smorzamento isteretico. In particolare si sottolinea l'innesco dell'incrudimento cinematico in entrambe le direzioni con un evidente ratcheting marcato in direzione YZ e cicli di isteresi più piccoli.

In seguito si illustrano le sovrapressioni interstiziali per i punti: A', B' e C'.





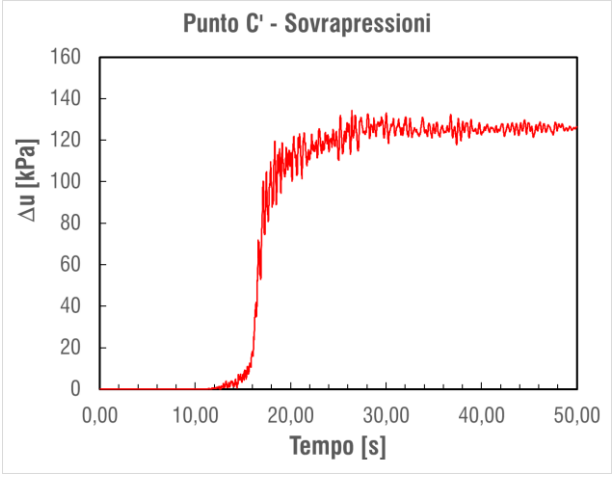


Fig. 64 Sovrapressioni interstiziali interfaccia terreno/bedrock

Dagli output si nota un incremento delle sovrapressioni interstiziali dovuti al passaggio del segnale intorno ai 140 kPa. Tale effetto è dovuto all'interazione terreno-fluido implementato con il modello costitutivo PIMY. In particolare, l'incremento è causato da deformazioni plastiche dovute alla risposta ciclica del materiale in interazione con il segnale in propagazione.

Con questo risultato si completa lo scenario del comportamento meccanico che si voleva ottenere.

Nonostante l'elevato numero di CPU che si ripercuote sulla mole di dati da elaborare, anche in questo caso, si sono ottenute le accelerazioni massime dei punti in superficie per la direzione x e y. Tali accelerazioni sono state normalizzate per le PGA dei segnali assegnati in input rispettivamente per le due direzioni, ottenendo le mappe dei fattori di amplificazione in superficie mostrate nella Fig. 65:

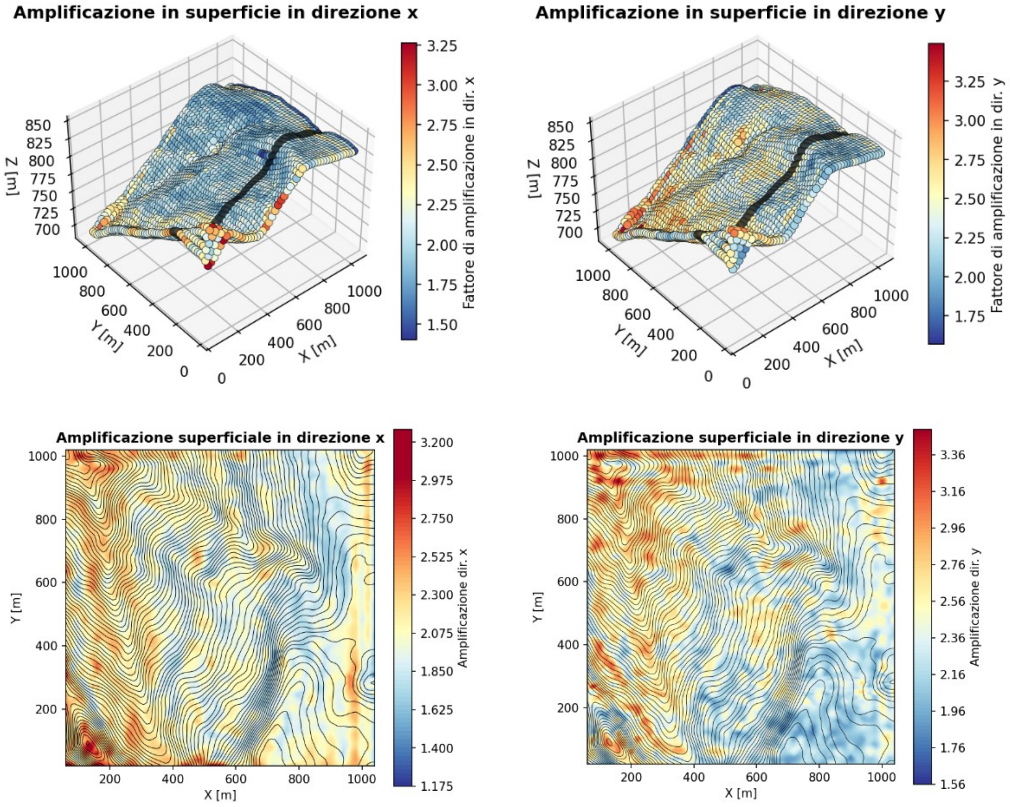


Fig. 65 Visione 3D e mappe delle amplificazioni superficiali in basso.

Come si nota facilmente per la mappa di amplificazione in direzione X, confrontando il risultato con quella del modello meno rifinito, si nota una differenza netta. In questo caso, infatti sono sempre evidenti zone di amplificazione in corrispondenza delle irregolarità geomorfologiche ma con un modulo e una diffusione differente rispetto al caso precedente. Tale effetto indica l'influenza dell'amplificazione di frequenze che nel modello precedente erano state tagliate a causa della dimensione dell'elemento finito.

In questo caso inoltre anche i risultati nella direzione Y iniziano ad annidarsi nelle zone di valli/promontori lasciando la rappresentazione a "macchia di leopardo" del caso precedente.

Pertanto si rimarca il limite rappresentato da una mesh di dimensioni ancora troppo elevate per cui è avvenuto un filtro delle frequenze massime trasmesse in superficie, in base alla seguente relazione:

$$f_{max} = \frac{V_s}{(6 \div 8)h} = \frac{250\ m/s}{8 \cdot 7.4\ m} = 4.22\ Hz \qquad (75)$$

per cui le frequenze superiori a 4.22 Hz con i loro effetti di interazione non sono rappresentate nei risultati dell'analisi.

Quindi, si ritiene necessario applicare un forte infittimento della mesh con una richiesta di capacità computazionali superiori. Da test effettuati tale capacità è pari a circa 200 CPU con una misura del lato dell'elemento di 4 m.

CONCLUSIONI

Il principale obiettivo del presente lavoro di tesi è stata la definizione di una metodologia rigorosa di modellazione FEM dinamica lineare e non lineare 1D, 2D e 3D per la simulazione dei fenomeni di propagazione sismica nei terreni, utilizzando sistemi di calcolo parallelo e HPC per applicazioni su area vasta. In tale ottica, il lavoro ha fornito gli strumenti metodologici, tecnici ed informatici che permettono lo sviluppo completo di analisi volte alla valutazione della pericolosità sismica locale. La previsione degli effetti di amplificazione che avvengono durante la propagazione del segnale all'interno del deposito di terreno è, infatti, di fondamentale importanza per la riduzione del rischio sismico su aree fortemente esposte.

Nella prima parte del lavoro di dottorato, dopo aver preso confidenza con i sistemi informatici ad alte prestazioni (come quelli messi a disposizione dal centro di calcolo RECAS di Bari), sono stati studiati i software ad elementi finiti open-source maggiormente diffusi in ambito scientifico per lo studio del problema della propagazione sismica locale. Tra questi, si è scelto di utilizzare il codice OPENSEES in quanto esso fornisce un ambiente di calcolo estremamente flessibile per modellare sistemi strutturali complessi sottoposti a carichi dinamici ed ha la capacità di operare efficacemente sia in modalità seriale che parallela, adattandosi alle diverse esigenze computazionali dell'utente. Successivamente, sono state sviluppate applicazioni Python per il pre- e post-processing dei dati di simulazioni FEM eseguite con il codice selezionato secondo i principi della programmazione ad oggetti e data analysis. La maggior parte dei codici di questi applicativi sono allegati nelle appendici della tesi e scaricabili liberamente.

Nella seconda parte della tesi relativo alle simulazioni di tipici problemi al finito di geotecnica sismica, è stato adottato un approccio rigoroso che è partito dall'analisi di casi ideali semplici, come quello della propagazione monodimensionale in una colonna di terreno, passando a casi più complessi, come la risposta sismica di un pendio, verificandone i risultati attraverso il confronto con procedure ampiamente consolidate. Il confronto tra i risultati 1D e 2D ha permesso di validare le previsioni di OPENSEES rispetto a quelle di un software agli elementi finiti consolidato nella

letteratura scientifica (PLAXIS), a parità di ipotesi costitutiva adottata per descrivere il comportamento meccanico del terreno.

Parallelamente alle applicazioni al finito, si è approfondito anche l'argomento teorico dei modelli costitutivi avanzati, quali quelli ad incrudimento isotropo (HSsmall implementato in PLAXIS) e ad incrudimento cinematico (PIMY utilizzato in OPENSEES), fornendo una panoramica completa della loro formulazione e degli aspetti di implementazione nei codici FEM.

In generale, l'approccio adottato ha permesso di riconoscere gli ingredienti principali che governano le modellazioni del problema della propagazione sismica locale ed, in particolare, il peso che hanno: le funzioni di forma, le caratteristiche dell'elemento finito in relazione all'accoppiamento idraulico (formulazione u-p), la distribuzione degli elementi sul risultato finale delle analisi, la presenza di simmetrie, l'efficacia dello smorzamento viscoso ed isteretico.

Il capitolo finale della tesi riporta i risultati preliminari di un'analisi 3D dinamica non lineare, eseguita con OPENSEES in parallelo su personal computer, di un'area rappresentativa di un pendio naturale. In tale simulazione, la discretizzazione del modello numerico con elementi finiti non ha potuto essere troppo raffinata, per i limiti di hardware imposti dal pc personale. Tuttavia, gli output ottenuti in corrispondenza di due punti pilota, uno in cresta al pendio e l'altro a 25 m di profondità, risultano promettenti ed evidenziano chiaramente i tipici effetti di amplificazione topografica in corrispondenza delle creste e delle valli (o compluvi). L'elaborazione delle accelerazioni massime di superficie ottenute dall'analisi 3D ha, così, consentito di definire mappe di pericolosità sismica locale per l'intera area oggetto di studio.

Gli sviluppi futuri riguardano principalmente lo sviluppo di modelli FEM 3D più raffinati ed estesi spazialmente per una valutazione più accurata della pericolosità sismica su area vasta.

APPENDICE 1

```
set meshFile [open test.msh w]
set nodesInfo [open nodesInfo.dat w]
set elementInfo [open elementsInfo.dat w]

puts $meshFile "MESH ffBrick dimension 3 ElemType Hexahedra
Nnode 8"
puts $meshFile "Coordinates"
puts $meshFile "#node_number   coord_x   coord_y   coord_z"

puts $meshFile  "1 0.0 0.0 0.0 "
puts $meshFile  "2 1.0 0.0 0.0 "
puts $meshFile  "3 1.0 1.0 0.0 "
puts $meshFile  "4 0.0 1.0 0.0 "
puts $meshFile  "5 0.0 0.0 1.0 "
puts $meshFile  "6 1.0 0.0 1.0 "
puts $meshFile  "7 1.0 1.0 1.0 "
puts $meshFile  "8 0.0 1.0 1.0 "

puts $nodesInfo  "1 0.0 0.0 0.0 "
puts $nodesInfo  "2 1.0 0.0 0.0 "
puts $nodesInfo  "3 1.0 1.0 0.0 "
puts $nodesInfo  "4 0.0 1.0 0.0 "
puts $nodesInfo  "5 0.0 0.0 1.0 "
puts $nodesInfo  "6 1.0 0.0 1.0 "
puts $nodesInfo  "7 1.0 1.0 1.0 "
puts $nodesInfo  "8 0.0 1.0 1.0 "

puts $meshFile "end coordinates"
puts $meshFile "Elements"
puts $meshFile "# element   node1   node2   node3   node4
node5   node6   node7   node8"

puts $elementInfo "1 1 2 3 4 5 6 7 8"

puts $meshFile  "1 1 2 3 4 5 6 7 8"

puts $meshFile "end elements"
close $meshFile
close $elementInfo
close $nodesInfo
######################################## START
########################################
wipe

set ndm 3
```

```
set ndf 4

model BasicBuilder -ndm $ndm -ndf $ndf

set peakShearStrain  0.1
set poisson  0.3

set G1 125000.0

set E1 [expr $G1*(2.0*(1+$poisson))]
set B1 [expr $E1*(3.0*(1-2*$poisson))]

set alpha [expr (1.0/(4.0 + ($B1 + (4.0/3.0)*$G1)))]
puts "alpha = $alpha"

set p_ref 500.0
set frictionAng 0.0
set cohesion 270

set pressDependCoe 0.0

#######################################################
set Bfluid 2.2e6
set fluid1 1
set solid1 10
set fmass 1.0
#set fmass 0.0
#valroe perm testato 1.0e-15
set perm 1.0e-4
#######################################################
set rhoS 1.97
#set rhoS 0.0
set rhoF 1.0
#######################################################
set densityMult 1.0
#######################################################

set massProportionalDamping 1.0
set InitialStiffnessProportionalDamping 1.0

set bUnitWeightX 0
set bUnitWeightY 0
set bUnitWeightZ 0


#######################################################
```

```
nDMaterial PressureIndependMultiYield $solid1 $ndm $rhoS $G1 $B1
$cohesion $peakShearStrain $frictionAng  $p_ref $pressDependCoe
40

nDMaterial InitialStateAnalysisWrapper 2 $solid1 3

node 1 0.0 0.0 0.0
node 4 0.0 1.0 0.0
node 3 1.0 1.0 0.0
node 2 1.0 0.0 0.0
node 5 0.0 0.0 1.0
node 8 0.0 1.0 1.0
node 7 1.0 1.0 1.0
node 6 1.0 0.0 1.0

fix 1 0 0 1 1
fix 4 0 0 1 1
fix 3 0 0 1 1
fix 2 0 0 1 1
fix 5 0 0 0 1
fix 6 0 0 0 1
fix 7 0 0 0 1
fix 8 0 0 0 1


element brickUP 1 1 2 3 4 5 6 7 8 2 $B1 $fmass $perm $perm $perm
$bUnitWeightX $bUnitWeightY $bUnitWeightZ


updateMaterialStage -material 2 -stage 0

set durata 10
set dt 0.1
set numSteps [expr int($durata/$dt)]

puts "INITAL ANALYSIS START"
constraints Penalty 1e18 1e18
numberer RCM
system ProfileSPD
algorithm ModifiedNewton
test NormDispIncr 1.0e-5 600 2
integrator Newmark 0.5 0.25
analysis Transient

InitialStateAnalysis on

set load_f 500.0
```

```
set patternTag 2
set tsTag 1
set linCfact 0.1

set durata 10
set dt 0.1
set numSteps [expr int($durata/$dt)]


timeSeries Constant $tsTag -factor 1
pattern Plain $patternTag $tsTag -factor 1 {

load 1  [expr (1.0/4.0)*$load_f]   [expr (1.0/4.0)*$load_f]
[expr (1.0/4.0)*$load_f] 0
load 2  [expr -(1.0/4.0)*$load_f]  [expr (1.0/4.0)*$load_f]
[expr (1.0/4.0)*$load_f] 0
load 3  [expr -(1.0/4.0)*$load_f]  [expr -(1.0/4.0)*$load_f]
[expr (1.0/4.0)*$load_f] 0
load 4  [expr (1.0/4.0)*$load_f]   [expr -(1.0/4.0)*$load_f]
[expr (1.0/4.0)*$load_f] 0
load 5  [expr (1.0/4.0)*$load_f]   [expr (1.0/4.0)*$load_f]
[expr -(1.0/4.0)*$load_f] 0
load 8  [expr (1.0/4.0)*$load_f]   [expr -(1.0/4.0)*$load_f]
[expr -(1.0/4.0)*$load_f] 0
load 7  [expr -(1.0/4.0)*$load_f]  [expr -(1.0/4.0)*$load_f]
[expr -(1.0/4.0)*$load_f] 0
load 6  [expr -(1.0/4.0)*$load_f]  [expr  (1.0/4.0)*$load_f]
[expr -(1.0/4.0)*$load_f] 0
}

analyze $numSteps $dt

InitialStateAnalysis off

puts "INITAL ANALYSIS END"
recorder Element -ele 1  -file  0stress1.out material 1 stress


updateMaterialStage -material 2 -stage 1
updateMaterialStage -material $solid1 -stage 1

equalDOF 5 6 3
equalDOF 5 7 3
equalDOF 5 8 3

equalDOF 5 6 1
equalDOF 5 7 1
```

```
equalDOF 5 8 1

equalDOF 5 6 2
equalDOF 5 7 2
equalDOF 5 8 2

equalDOF 1 2 3
equalDOF 1 3 3
equalDOF 1 4 3

equalDOF 1 2 1
equalDOF 1 3 1
equalDOF 1 4 1

equalDOF 1 2 2
equalDOF 1 3 2
equalDOF 1 4 2

set n11 [nodeDisp 1 1]
set n21 [nodeDisp 2 1]
set n31 [nodeDisp 3 1]
set n41 [nodeDisp 4 1]
set n51 [nodeDisp 5 1]
set n61 [nodeDisp 6 1]
set n71 [nodeDisp 7 1]
set n81 [nodeDisp 8 1]

set n12 [nodeDisp 1 2]
set n22 [nodeDisp 2 2]
set n32 [nodeDisp 3 2]
set n42 [nodeDisp 4 2]
set n52 [nodeDisp 5 2]
set n62 [nodeDisp 6 2]
set n72 [nodeDisp 7 2]
set n82 [nodeDisp 8 2]

set n13 [nodeDisp 1 3]
set n23 [nodeDisp 2 3]
set n33 [nodeDisp 3 3]
set n43 [nodeDisp 4 3]
set n53 [nodeDisp 5 3]
set n63 [nodeDisp 6 3]
set n73 [nodeDisp 7 3]
set n83 [nodeDisp 8 3]

timeSeries Constant 9  -factor 1.0
pattern Plain 16 9 -factor 1.0 {
```

```
      sp 1 1 $n11
      sp 2 1 $n21
      sp 3 1 $n31
      sp 4 1 $n41

      sp 5 1 $n51
      sp 6 1 $n61
      sp 7 1 $n71
      sp 8 1 $n81

      sp 1 2 $n12
      sp 2 2 $n22
      sp 3 2 $n32
      sp 4 2 $n42

      sp 5 2 $n52
      sp 6 2 $n62
      sp 7 2 $n72
      sp 8 2 $n82

      sp 5 3 $n53
      sp 6 3 $n63
      sp 7 3 $n73
      sp 8 3 $n83

}

analyze 10 1

set durata2 30.0
set dt2 0.01
set numSteps2 [expr int($durata2/$dt2)]
puts "STEPS $numSteps2 dt $dt2 durata $durata2"

set time_dyn factors_dyn.out
timeSeries Path 10 -dt $dt2 -filePath $time_dyn  -factor 1.0 -
startTime 10.0
pattern Plain [expr ($patternTag+1)] 10 -factor 1.0 {

sp 5 1 [expr 1 + $n51]
sp 6 1 [expr 1 + $n61]
sp 7 1 [expr 1 + $n71]
sp 8 1 [expr 1 + $n81]

sp 1 1 [expr -1 + $n11]
sp 2 1 [expr -1 + $n21]
```

```
sp 3 1 [expr -1 + $n31]
sp 4 1 [expr -1 + $n41]

}

puts "spostamento nodo 5 [nodeDisp 5 1]"
recorder Node -file displacement.out -time -dT $dt2 -node 1 2 3
4 5 6 7 8 -dof 1 2 3 disp
recorder Node -file velocity.out -time -dT $dt2 -node 1 2 3 4 5
6 7 8 -dof 1 2 3 vel
recorder Node -file acceleration.out -time -dT $dt2 -node 1 2 3
4 5 6 7 8 -dof 1 2 3 accel
recorder Node -file porePressure.out -time -dT $dt2 -node 1 2 3
4 5 6 7 8 -dof 4 vel

recorder Element -ele 1  -file  stress1.out -time -dT $dt2 mate-
rial 1 stress
recorder Element -ele 1  -file  stress2.out -time -dT $dt2 mate-
rial 2 stress
recorder Element -ele 1  -file  stress3.out -time -dT $dt2 mate-
rial 3 stress
recorder Element -ele 1  -file  stress4.out -time -dT $dt2 mate-
rial 4 stress
recorder Element -ele 1  -file  stress5.out -time -dT $dt2 mate-
rial 5 stress
recorder Element -ele 1  -file  stress6.out -time -dT $dt2 mate-
rial 6 stress
recorder Element -ele 1  -file  stress7.out -time -dT $dt2 mate-
rial 7 stress
recorder Element -ele 1  -file  stress8.out -time -dT $dt2 mate-
rial 8 stress

recorder Element -ele 1  -file  strain1.out -time -dT $dt2 mate-
rial 1 strain
recorder Element -ele 1  -file  strain2.out -time -dT $dt2 mate-
rial 2 strain
recorder Element -ele 1  -file  strain3.out -time -dT $dt2 mate-
rial 3 strain
recorder Element -ele 1  -file  strain4.out -time -dT $dt2 mate-
rial 4 strain
recorder Element -ele 1  -file  strain5.out -time -dT $dt2 mate-
rial 5 strain
recorder Element -ele 1  -file  strain6.out -time -dT $dt2 mate-
rial 6 strain
recorder Element -ele 1  -file  strain7.out -time -dT $dt2 mate-
rial 7 strain
```

```
recorder Element -ele 1  -file  strain8.out -time -dT $dt2 mate-
rial 8 strain

recorder Element -ele 1 -time -file backbone.out -dT $dt2 mate-
rial 4 backbone 500
recorder Element -ele 1 -time -file backbone1.out -dT $dt2 mate-
rial 1 backbone 1.0

rayleigh $massProportionalDamping 0.0 $InitialStiffnessPropor-
tionalDamping 0.0


constraints Transformation
numberer RCM
system ProfileSPD
algorithm ModifiedNewton
test NormDispIncr 1.0E-09 600 2
integrator Newmark 0.5 0.25
analysis Transient
analyze $numSteps2 $dt2

puts "FINE DELLE ANALISI"
wipe
```

APPENDICE 2

```
import math as mth
import numpy as np
import matplotlib.pyplot as plt

def stress_float(lst):
    ''' divide la lista stress in sottoliste per cui
rst[0] è l'istante dt da 0 a durata. mentre rst[0][0] è
    il punto di gauss quindi va da 0 a 7 (essendo 8 i
punti). Solo per stress e strain'''
    l = len(lst)
    m = len(lst[0][0])
    rst = [[[float(x) for x in lst[i][0][j].split()] for i
in range(l)] for j in range(m)]
    return rst

def disp_float(lst):
     ''' divide la lista degli spostamenti. In questo caso
rst[0] è composto dal dt e dalle componenti dello
     spostamento. Applicabile anche alle pressioni, il ri-
sultato sono subliste per gli 8 nodi [dt, p1,..,p8]  '''
     l = len(lst[0]) #dt
     m = len(lst[0][0])
     rst = [[float(x) for x in lst[0][i].split()] for i in
range(l)]
     return rst

####### LEGGIAMO I FILE OUTPUT DEI PUNTI DI GAUSS
var = [i for i in range(1,9)]
displacement = [s for s in [y.readlines() for y in
[open(f'displacement.out','r')]]]
acceleration = [s for s in [y.readlines() for y in
[open(f'acceleration.out','r')]]]
ppressure =  [s for s in [y.readlines() for y in
[open(f'porePressure.out','r')]]]

stress = [s for s in [[y.readlines() for y in
[open(f'stress{k}.out','r')]] for k in range(1,9)]]
strain = [s for s in [[y.readlines() for y in
[open(f'strain{k}.out','r')]] for k in range(1,9)]]
```

```
#####attenzione non applicare più volte la funzione altri-
menti suddivide ulteriormente
stress_flt = stress_float(stress)
strain_flt = stress_float(strain)
disp_flt = disp_float(displacement)
acc_flt = disp_float(acceleration)
pwp = disp_float(ppressure)

stress_flt, il risultato è [[[dt (gp1) compomnenti stress
gp1],[dt (gp2) stress(gp2)]...[dt gp(8)], [dt(i+1),
(gp1)]... stress_flt[0][0] esempio di dt = 0 e punto di
gauss n 1 (lo stesso per strain_flt) il risultato sono le
6 compomnenti dello stress (+eta) o dello strain.
acc_flt [dt, componenti x y z per gli 8 nodi],[dt(i+1),
x(i+1) y(i+1) z(i+1),...] uguale per accelerazione e pwp

 QUERY SUI NODI ##############################
time = [t[0] for t in disp_flt]
array_t = np.array(time)
np.savetxt("time.txt",array_t)
### in acc_x ci sono tutte le componenti degli spostamenti
x per tutti gli 8 punti, quindi abbiamo una lista di
### 8 valori per ogni istante di tempo per tutta la du-
rata.
disp_x = [disp_flt[i][1::3] for i in range(len(disp_flt))]
disp_y = [disp_flt[i][2::3] for i in range(len(disp_flt))]
disp_z = [disp_flt[i][3::3] for i in range(len(disp_flt))]

acc_x = [acc_flt[i][1::3] for i in range(len(acc_flt))]
acc_y = [acc_flt[i][2::3] for i in range(len(acc_flt))]
acc_z = [acc_flt[i][3::3] for i in range(len(acc_flt))]

pwp_t =  [pwp[i][1::] for i in range(len(pwp))]

 QUERY SUI PUNTI DI GAUSS - STRESS
##############################

sigma_xx = [[stress_flt[i][j][1] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
sigma_yy = [[stress_flt[i][j][2] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
```

```
sigma_zz = [[stress_flt[i][j][3] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
sigma_xy = [[stress_flt[i][j][4] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
sigma_yz = [[stress_flt[i][j][5] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
sigma_xz = [[stress_flt[i][j][6] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]

#TOTALI
sxx = np.array(sigma_xx)
syy = np.array(sigma_yy)
szz = np.array(sigma_zz)
sxy = np.array(sigma_xy)
sxz = np.array(sigma_xz)
syz = np.array(sigma_yz)

 QUERY SUI PUNTI DI GAUSS - STRAIN
##############################
eps_xx = [[strain_flt[i][j][1] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
eps_yy = [[strain_flt[i][j][2] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
eps_zz = [[strain_flt[i][j][3] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
eps_xy = [[strain_flt[i][j][4] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
eps_yz = [[strain_flt[i][j][5] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]
eps_xz = [[strain_flt[i][j][6] for i in
range(len(stress_flt))] for j in
range(len(stress_flt[0]))]

exx = np.array(eps_xx)
```

```
eyy = np.array(eps_yy)
ezz = np.array(eps_zz)
exy = np.array(eps_xy)
exz = np.array(eps_xz)
eyz = np.array(eps_yz)

SPOSTAMENTI

 PLOT SPOSTAMENTI PER GLI 8 NODI ####################
disp_z_1 = [x[0] for x in disp_z]
disp_z_2 = [x[1] for x in disp_z]
disp_z_3 = [x[2] for x in disp_z]
disp_z_4 = [x[3] for x in disp_z]

disp_z_down = np.c_[disp_z_1,disp_z_2,disp_z_3,disp_z_4]

disp_z_5 = [x[4] for x in disp_z]
disp_z_6 = [x[5] for x in disp_z]
disp_z_7 = [x[6] for x in disp_z]
disp_z_8 = [x[7] for x in disp_z]

disp_z_top = np.c_[disp_z_5,disp_z_6,disp_z_7,disp_z_8]

fig, ax = plt.subplots(figsize=(10,7))
ax.set_title('Spostamenti Z down')
ax.set_ylabel('spostamento [m]')
ax.set_xlabel('time step [s]')
ax.grid(True)
ax.plot(time,np.c_[disp_z_down],la-
bel=['p1','p2','p3','p4'])
plt.legend()
plt.savefig('pic_disp_z_down.png')
plt.show()

fig, ax2 = plt.subplots(figsize=(10,7))
ax2.set_title('Spostamenti Z top')
ax2.set_ylabel('spostamento [m]')
ax2.set_xlabel('time step [s]')
ax2.grid(True)
#ax.plot(time,np.c_[list1,list2],'g--')
ax2.plot(time,np.c_[disp_z_top],la-
bel=['p5','p6','p7','p8'])
plt.legend()
```

```
plt.savefig('pic_disp_z_top.png')
plt.show()

 PLOT SPOSTAMENTI PER GLI 8 NODI ####################
disp_x_1 = [x[0] for x in disp_x]
disp_x_2 = [x[1] for x in disp_x]
disp_x_3 = [x[2] for x in disp_x]
disp_x_4 = [x[3] for x in disp_x]

disp_x_down = np.c_[disp_x_1,disp_x_2,disp_x_3,disp_x_4]

disp_x_5 = [x[4] for x in disp_x]
disp_x_6 = [x[5] for x in disp_x]
disp_x_7 = [x[6] for x in disp_x]
disp_x_8 = [x[7] for x in disp_x]

disp_x_top = np.c_[disp_x_5,disp_x_6,disp_x_7,disp_x_8]



fig, ax = plt.subplots(figsize=(10,7))
ax.set_title('Spostamenti X down')
ax.set_ylabel('spostamento [m]')
ax.set_xlabel('time step [s]')
ax.grid(True)
ax.plot(time,np.c_[disp_x_down],la-
bel=['p1','p2','p3','p4'])
plt.legend()
plt.savefig('pic_dispo_down.png')
plt.show()

fig, ax2 = plt.subplots(figsize=(10,7))
ax2.set_title('Spostamenti X top')
ax2.set_ylabel('spostamento [m]')
ax2.set_xlabel('time step [s]')
ax2.grid(True)
ax2.plot(time,np.c_[disp_x_top],la-
bel=['p5','p6','p7','p8'])
plt.legend()
plt.savefig('pic_dispo_top.png')
plt.show()

fig, ax3 = plt.subplots(figsize=(10,7))
```

```
ax3.set_title('Spostamenti X top')
ax3.set_ylabel('spostamento [m]')
ax3.set_xlabel('time step [s]')
ax3.grid(True)
plt.xlim(800,850)
ax3.plot(time,np.c_[disp_x_top],la-
bel=['p5','p6','p7','p8'])
plt.xlim(800,850)
plt.legend()
plt.savefig('pic_dispo_top.png')
plt.show()

 PLOT SPOSTAMENTI PER GLI 8 NODI ####################
disp_y_1 = [y[0] for y in disp_y]
disp_y_2 = [y[1] for y in disp_y]
disp_y_3 = [y[2] for y in disp_y]
disp_y_4 = [y[3] for y in disp_y]

disp_y_down = np.c_[disp_y_1,disp_y_2,disp_y_3,disp_y_4]

disp_y_5 = [y[4] for y in disp_y]
disp_y_6 = [y[5] for y in disp_y]
disp_y_7 = [y[6] for y in disp_y]
disp_y_8 = [y[7] for y in disp_y]

disp_y_top = np.c_[disp_y_5,disp_y_6,disp_y_7,disp_y_8]

fig, ax = plt.subplots(figsize=(10,7))
ax.set_title('Spostamenti Y down')
ax.set_ylabel('spostamento [m]')
ax.set_xlabel('time step [s]')
ax.grid(True)
ax.plot(time,np.c_[disp_y_down],la-
bel=['p1','p2','p3','p4'])
plt.legend()
plt.savefig('pic_dispo_down.png')
plt.show()

fig, ax2 = plt.subplots(figsize=(10,7))
ax2.set_title('Spostamenti Y top')
ax2.set_ylabel('spostamento [m]')
ax2.set_xlabel('time step [s]')
ax2.grid(True)
```

```
ax2.plot(time,np.c_[disp_y_top],la-
bel=['p5','p6','p7','p8'])
plt.legend()
plt.savefig('pic_dispo_top.png')
plt.show()

ACCELERAZIONI

 PLOT ACCELERAZIONI PER GLI 8 NODI ####################
acc_x_1 = [x[0] for x in acc_x]
acc_x_2 = [x[1] for x in acc_x]
acc_x_3 = [x[2] for x in acc_x]
acc_x_4 = [x[3] for x in acc_x]

acc_x_down = np.c_[acc_x_1,acc_x_2,acc_x_3,acc_x_4]

acc_x_5 = [x[4] for x in acc_x]
acc_x_6 = [x[5] for x in acc_x]
acc_x_7 = [x[6] for x in acc_x]
acc_x_8 = [x[7] for x in acc_x]

acc_x_top = np.c_[acc_x_5,acc_x_6,acc_x_7,acc_x_8]

fig, ax3 = plt.subplots(figsize=(10,7))
ax3.set_title('Accelerazione X down')
ax3.set_ylabel('a [m/s2]')
ax3.set_xlabel('time step [s]')
ax3.grid(True)
ax3.plot(time,np.c_[acc_x_down],la-
bel=['p1','p2','p3','p4'])
plt.legend()
plt.savefig('pic_acc_down.png')
plt.show()

fig, ax4 = plt.subplots(figsize=(10,7))
ax4.set_title('Accelerazione X top')
ax4.set_ylabel('a [m/s2]')
ax4.set_xlabel('time step [s]')
ax4.grid(True)
ax4.plot(time,np.c_[acc_x_top],la-
bel=['p5','p6','p7','p8'])
plt.legend()
plt.savefig('pic_acc_down.png')
```

```
plt.show()

u1 = [x[0] for x in pwp_t]
u2 = [x[1] for x in pwp_t]
u3 = [x[2] for x in pwp_t]
u4 = [x[3] for x in pwp_t]

uu1 = np.array(u1)
uu2 = np.array(u2)
uu3 = np.array(u3)
uu4 = np.array(u4)

pwp_down = np.c_[u1,u2,u3,u4]

fig, pwp = plt.subplots(figsize=(10,7))
pwp.set_title('Pore Water Pressure Down')
pwp.set_ylabel('u [kPa]')
pwp.set_xlabel('time step [s]')
pwp.grid(True)
pwp.plot(time,np.c_[pwp_down],label=['p1','p2','p3','p4'])
pwp.legend()
plt.savefig('pic_pwp_down.png')

u5 = [x[4] for x in pwp_t]
u6 = [x[5] for x in pwp_t]
u7 = [x[6] for x in pwp_t]
u8 = [x[7] for x in pwp_t]

uu5 = np.array(u5)
uu6 = np.array(u6)
uu7 = np.array(u7)
uu8 = np.array(u8)

uu_m = (1.0/8.0)*(uu1+uu2+uu3+uu4+uu5+uu6+uu7+uu8)

pwp_top = np.c_[u5,u6,u7,u8]

fig, pwp2 = plt.subplots(figsize=(10,7))
pwp2.set_title('Pore Water Pressure Top')
pwp2.set_ylabel('u [kPa]')
pwp2.set_xlabel('time step [s]')
pwp2.grid(True)
```

```
pwp2.plot(time,np.c_[pwp_down],la-
bel=['p5','p6','p7','p8'])
pwp2.legend()
plt.savefig('pic_pwp_top.png')

STRESS

sigma_x =
np.c_[sigma_xx[0],sigma_xx[1],sigma_xx[2],sigma_xx[3],sigm
a_xx[4],sigma_xx[5],sigma_xx[6],sigma_xx[7]]
fig, pwp2 = plt.subplots(figsize=(10,7))
pwp2.set_title('Sigma_xx')
pwp2.set_ylabel(r'$\sigma$_xx'+' '+'[kPa]')
pwp2.set_xlabel('time step [s]')
pwp2.grid(True)
pwp2.plot(time,sigma_x,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp2.legend()
plt.savefig('pic_sigmaxx.png')
#pwp.show()

sigma_y =
np.c_[sigma_yy[0],sigma_yy[1],sigma_yy[2],sigma_yy[3],sigm
a_yy[4],sigma_yy[5],sigma_yy[6],sigma_yy[7]]
fig, pwp2 = plt.subplots(figsize=(10,7))
pwp2.set_title('Sigma_yy')
pwp2.set_ylabel(r'$\sigma$_yy'+' '+'[kPa]')
pwp2.set_xlabel('time step [s]')
pwp2.grid(True)
pwp2.plot(time,sigma_y,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp2.legend()
plt.savefig('pic_sigmayy.png')

sigma_z =
np.c_[sigma_zz[0],sigma_zz[1],sigma_zz[2],sigma_zz[3],sigm
a_zz[4],sigma_zz[5],sigma_zz[6],sigma_zz[7]]
fig, pwp2 = plt.subplots(figsize=(10,7))
pwp2.set_title('Sigma_z')
pwp2.set_ylabel(r'$\sigma$_zz'+' '+'[kPa]')
pwp2.set_xlabel('time step [s]')
pwp2.grid(True)
```

```
pwp2.plot(time,sigma_z,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp2.legend()
plt.savefig('pic_sigmazz.png')

tau_xz =
np.c_[sigma_xz[0],sigma_xz[1],sigma_xz[2],sigma_xz[3],sigm
a_xz[4],sigma_xz[5],sigma_xz[6],sigma_xz[7]]
fig, pwp2 = plt.subplots(figsize=(10,7))
pwp2.set_title('Tau_xz')
pwp2.set_ylabel(r'$\tau$_xz'+' '+'[kPa]')
pwp2.set_xlabel('time step [s]')
pwp2.grid(True)
pwp2.plot(time,tau_xz,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp2.legend()
plt.savefig('pic_sigmaxz.png')

tau_xz_m =
(1.0/8.0)*(sxz[0]+sxz[1]+sxz[2]+sxz[3]+sxz[4]+sxz[5]+sxz[6
]+sxz[7])
fig, pwp2 = plt.subplots(figsize=(10,7))
pwp2.set_title('Tau_xz media')
pwp2.set_ylabel(r'$\tau$_xz'+' '+'[kPa]')
pwp2.set_xlabel('time step [s]')
pwp2.grid(True)
pwp2.plot(time,tau_xz_m,label='gp_medio')
pwp2.legend()
plt.savefig('pic_sigmaxz_medio.png')

tau_yz =
np.c_[sigma_yz[0],sigma_yz[1],sigma_yz[2],sigma_yz[3],sigm
a_yz[4],sigma_yz[5],sigma_yz[6],sigma_yz[7]]
fig, pwp2 = plt.subplots(figsize=(10,7))
pwp2.set_title('Tau_yz')
pwp2.set_ylabel(r'$\tau$_yz'+' '+'[kPa]')
pwp2.set_xlabel('time step [s]')
pwp2.grid(True)
pwp2.plot(time,tau_yz,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp2.legend()
plt.savefig('pic_sigmayz.png')
```

```
tau_xy =
np.c_[sigma_xy[0],sigma_xy[1],sigma_xy[2],sigma_xy[3],sigm
a_xy[4],sigma_xy[5],sigma_xy[6],sigma_xy[7]]
fig, pwp2 = plt.subplots(figsize=(10,7))
pwp2.set_title('Tau_xy')
pwp2.set_ylabel(r'$\tau$_xy'+' '+'[kPa]')
pwp2.set_xlabel('time step [s]')
pwp2.grid(True)
pwp2.plot(time,tau_xy,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp2.legend()
plt.savefig('pic_sigmaxy.png')

eps_x =
np.c_[eps_xx[0],eps_xx[1],eps_xx[2],eps_xx[3],eps_xx[4],ep
s_xx[5],eps_xx[6],eps_xx[7]]
fig, pwp5 = plt.subplots(figsize=(10,7))
pwp5.set_title('epsilon_xx')
pwp5.set_ylabel(r'$\epsilon$_xx'+' '+'[-]')
pwp5.set_xlabel('time step [s]')
pwp5.grid(True)
pwp5.plot(time,eps_x,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp5.legend()
plt.savefig('pic_epsilonxx.png')

eps_y =
np.c_[eps_yy[0],eps_yy[1],eps_yy[2],eps_yy[3],eps_yy[4],ep
s_yy[5],eps_yy[6],eps_yy[7]]
fig, pwp3 = plt.subplots(figsize=(10,7))
pwp3.set_title('epsilon_yy')
pwp3.set_ylabel(r'$\epsilon$_yy'+' '+'[-]')
pwp3.set_xlabel('time step [s]')
pwp3.grid(True)
pwp3.plot(time,eps_y,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp3.legend()
plt.savefig('pic_epsilonyy.png')

eps_z =
np.c_[eps_zz[0],eps_zz[1],eps_zz[2],eps_zz[3],eps_zz[4],ep
s_zz[5],eps_zz[6],eps_zz[7]]
fig, pwp4 = plt.subplots(figsize=(10,7))
```

```
pwp4.set_title('epsilon_zz')
pwp4.set_ylabel(r'$\epsilon$_zz'+' '+'[-]')
pwp4.set_xlabel('time step [s]')
pwp4.grid(True)
pwp4.plot(time,eps_z,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp4.legend()
plt.savefig('pic_epsilonzz.png')

gamma_xz =
np.c_[eps_xz[0],eps_xz[1],eps_xz[2],eps_xz[3],eps_xz[4],ep
s_xz[5],eps_xz[6],eps_xz[7]]
fig, pwp5 = plt.subplots(figsize=(10,7))
pwp5.set_title('gamma_xz')
pwp5.set_ylabel(r'$\gamma$_xz'+' '+'[-]')
pwp5.set_xlabel('time step [s]')
pwp5.grid(True)
pwp5.plot(time,gamma_xz,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp5.legend()
plt.savefig('pic_epsilonxz.png')

gamma_xz_m = (1.0/8.0)*(np.array(eps_xz[0])+np.ar-
ray(eps_xz[1])+np.array(eps_xz[2])+np.ar-
ray(eps_xz[3])+np.array(eps_xz[4])+np.ar-
ray(eps_xz[5])+np.array(eps_xz[6])+np.array(eps_xz[7]))
fig, pwp6 = plt.subplots(figsize=(10,7))
pwp6.set_title('gamma_xz')
pwp6.set_ylabel(r'$\gamma$_xz'+' '+'[-]')
pwp6.set_xlabel('time step [s]')
pwp6.grid(True)
pwp6.plot(time,gamma_xz_m, label='gamma_medio')
pwp6.legend()
plt.savefig('pic_epsilonxz_medio.png')

gamma_yz =
np.c_[eps_yz[0],eps_yz[1],eps_yz[2],eps_yz[3],eps_yz[4],ep
s_yz[5],eps_yz[6],eps_yz[7]]
fig, pwp6 = plt.subplots(figsize=(10,7))
pwp6.set_title('gamma_yz')
pwp6.set_ylabel(r'$\gamma$_yz'+' '+'[-]')
pwp6.set_xlabel('time step [s]')
pwp6.grid(True)
```

```
pwp6.plot(time,gamma_yz,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp6.legend()
plt.savefig('pic_epsilonyz.png')

gamma_xy =
np.c_[eps_xy[0],eps_xy[1],eps_xy[2],eps_xy[3],eps_xy[4],ep
s_xy[5],eps_xy[6],eps_xy[7]]
fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title('gamma_xy')
pwp7.set_ylabel(r'$\gamma$_xy'+' '+'[-]')
pwp7.set_xlabel('time step [s]')
pwp7.grid(True)
pwp7.plot(time,gamma_xy,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp7.legend()
plt.savefig('pic_epsilonxy.png')

STRESS:STRAIN

fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title('epsilon_xx - gamma_xx')
pwp7.set_xlabel(r'$\epsilon$_xx'+' '+'[-]')
pwp7.set_ylabel(r'$\sigma$_xx'+' '+'[kPa]')
pwp7.grid(True)
pwp7.plot(eps_x,sigma_x,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp7.legend()
plt.savefig('pic_rispostaxx.png')

fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title('epsilon_yy - gamma_yy')
pwp7.set_xlabel(r'$\epsilon$_yy'+' '+'[-]')
pwp7.set_ylabel(r'$\sigma$_yy'+' '+'[kPa]')
pwp7.grid(True)
pwp7.plot(eps_y,sigma_y,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp7.legend()
plt.savefig('pic_rispostayy.png')

fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title('epsilon_zz - gamma_zz')
pwp7.set_xlabel(r'$\epsilon$_zz'+' '+'[-]')
```

```
pwp7.set_ylabel(r'$\sigma$_zz'+' '+'[kPa]')
pwp7.grid(True)
pwp7.plot(eps_z,sigma_z,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp7.legend()
plt.savefig('pic_rispostazz.png')

fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title('tau_xz - gamma_xz')
pwp7.set_xlabel(r'$\gamma$_xz'+' '+'[-]')
pwp7.set_ylabel(r'$\tau$_xz'+' '+'[kPa]')
pwp7.grid(True)
pwp7.plot(gamma_xz,tau_xz,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp7.legend()
plt.savefig('pic_rispostaxz.png')

fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title('tau_xz - gamma_xz')
pwp7.set_xlabel(r'$\gamma$_xz'+' '+'[-]')
pwp7.set_ylabel(r'$\tau$_xz'+' '+'[kPa]')
pwp7.grid(True)
pwp7.plot(gamma_xz_m,tau_xz_m,label='valori medi gp')
pwp7.legend()
plt.savefig('pic_rispostaxz_medio.png')

##### EXPORT TAU XZ ####################
for i in range(8):

np.savetxt(f'gamma_xz_gp_{i+1}.txt',np.take(gamma_xz,i,axi
s = 1))

np.savetxt(f'tau_xz_gp_{i+1}.txt',np.take(tau_xz,i,axis =
1))
np.savetxt("tau_xz_medio.txt",tau_xz_m)

#tensioni ottaedriche
p = (1.0/3.0)*(sxx + syy + szz)
q = (1.0/3.0)*np.sqrt((sxx - syy)**2 + (syy - szz)**2 +
(szz - sxx)**2 + (6*(sxy**2 +sxz**2 + syz**2)))

qr_lst = [s for s in [y.readlines() for y in [open(f'C15i-
q.txt','r')]]]
```

```
ppr_lst = [s for s in [y.readlines() for y in
[open(f'C15i-pp.txt','r')]]]
lr = len(qr_lst[0])
mr = len(qr_lst[0][0])
qr_rst = [[float(x) for x in qr_lst[0][i].split()] for i
in range(lr) ]
ppr_rst = [[-float(x) for x in ppr_lst[0][i].split()] for
i in range(lr) ]
qr_plot = [qr_rst[i][0] for i in range(len(qr_rst))]
ppr_plot = [ppr_rst[i][0] for i in range(len(ppr_rst))]

epsa_lst = [s for s in [y.readlines() for y in
[open(f'C15i-epsa.txt','r')]]]
ler = len(epsa_lst[0])
mer = len(epsa_lst[0][0])
epsa_rst = [[float(x) for x in epsa_lst[0][i].split()] for
i in range(ler) ]
epsa_plot = [i[0]/100.0 for i in epsa_rst]

#attenzione certevolte diagramma sfarfalla e da quello er-
rato, ricalcolare tutte le celle
p_plot =
np.c_[p[0]+uu1,p[1]+uu2,p[2]+uu3,p[3]+uu4,p[4]+uu5,p[5]+uu
6,p[6]+uu7,p[7]+uu8]
q_plot = np.c_[q[0],q[1],q[2],q[3],q[4],q[5],q[6],q[7]]
fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title("p' - q")
pwp7.set_xlabel("p"+' '+'[kPa]')
pwp7.set_ylabel('q'+' '+'[kPa]')
pwp7.grid(True)
pwp7.plot(p_plot,q_plot,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp7.invert_xaxis()
pwp7.legend()
plt.savefig('pic_rispostapq.png')

pp_m =
(1.0/8.0)*((p[0]+uu1)+(p[1]+uu2)+(p[2]+uu3)+(p[3]+uu4)+(p[
4]+uu5)+(p[5]+uu6)+(p[6]+uu7)+(p[7]+uu8))
q_m = (1.0/8.0)*(q[0]+q[1]+q[2]+q[3]+q[4]+q[5]+q[6]+q[7])
fig, pwp_7 = plt.subplots(figsize=(10,7))
pwp_7.set_title("p' - q")
pwp_7.set_xlabel("p'"+' '+'[kPa]')
```

```
pwp_7.set_ylabel('q'+' '+'[kPa]')
pwp_7.grid(True)
pwp_7.plot(pp_m,q_m,label='medio')
pwp_7.plot(ppr_plot,qr_plot,label='TRX-CIU C15i')
plt.xlim(-1200,0)
pwp_7.invert_xaxis()
pwp_7.legend()
plt.savefig('pic_rispostapq.png')

np.savetxt("p'_m",pp_m)
np.savetxt('q_m',q_m)

ess = (2.0/3.0)*np.sqrt((exx-eyy)**2+(eyy-ezz)**2+(ezz-
exx)**2+(6*(exy**2+exz**2+eyz**2)))
ess_plot=
np.c_[ess[0],ess[1],ess[2],ess[3],ess[4],ess[5],ess[6],ess
[7]]
ess_m =
(1.0/8.0)*(ess[0]+ess[1]+ess[2]+ess[3]+ess[4]+ess[5]+ess[6
]+ess[7])

fig, pwp_8 = plt.subplots(figsize=(10,7))
pwp_8.set_title("q - eps_s")
pwp_8.set_xlabel("eps"+' '+'[-]')
pwp_8.set_ylabel('q'+' '+'[kPa]')
pwp_8.grid(True)
pwp_8.plot(ess_plot,q_plot,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp_8.plot(epsa_plot,qr_plot,label=['TRX-CIU C15i'])
pwp_8.legend()
plt.savefig('pic_risposta_q_eps.png')

fig, pwp_9 = plt.subplots(figsize=(10,7))
pwp_9.set_title("q - eps_s")
pwp_9.set_xlabel("eps"+' '+'[-]')
pwp_9.set_ylabel('q'+' '+'[kPa]')
pwp_9.grid(True)
pwp_9.plot(ess_m,q_m,label='medio')
pwp_9.plot(-eps_z,q_m,label='eps_z - q_medio')
pwp_9.plot(epsa_plot,qr_plot,label='TRX-CIU C15i')
pwp_9.legend()
plt.savefig('pic_risposta_eps_q_medio.png')
```

```
fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title('tau_yz - gamma_yz')
pwp7.set_xlabel(r'$\gamma$_yz'+' '+'[-]')
pwp7.set_ylabel(r'$\tau$_yz'+' '+'[kPa]')
pwp7.grid(True)
pwp7.plot(gamma_yz,tau_yz,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp7.legend()
plt.savefig('pic_rispostayz.png')

fig, pwp7 = plt.subplots(figsize=(10,7))
pwp7.set_title('tau_xy - gamma_xy')
pwp7.set_xlabel(r'$\gamma$_xy'+' '+'[-]')
pwp7.set_ylabel(r'$\tau$_xy'+' '+'[kPa]')
pwp7.grid(True)
pwp7.plot(gamma_xy,tau_xy,la-
bel=['gp1','gp2','gp3','gp4','gp5','gp6','gp7','gp8'])
pwp7.legend()
plt.savefig('pic_rispostaxy.png')

### BACKBONE CURVE ###
backb1 = open('backbone1.out')
readb1 = backb1.readlines()
lb1 = len(readb1)
list1_1 = [float(i) for i in [readb1[x].split(' ') for x
in range(lb1)][0][3:]]
gamma1 = [float(i) for i in [readb1[x].split(' ') for x in
range(lb1)][0][3:]][::2] #indici 0 2 4 6
Gs1 = [float(i) for i in [readb1[x].split(' ') for x in
range(lb1)][0][4:]][::2]

### BACKBONE CURVE ###
backb = open('backbone.out')
readb = backb.readlines()
lb = len(readb)
list_1 = [float(i) for i in [readb[x].split(' ') for x in
range(lb)][0][3:]]
gamma = [float(i) for i in [readb[x].split(' ') for x in
range(lb)][0][3:]][::2] #indici 0 2 4 6
Gs = [float(i) for i in [readb[x].split(' ') for x in
range(lb)][0][4:]][::2]
```

```
gammaG = [s for s in [y.readlines() for y in [open(f'C15i-
gammaG.txt','r')]]]
GG = [s for s in [y.readlines() for y in [open(f'C15i-
G.txt','r')]]]
lr = len(gammaG[0])
mr = len(gammaG[0][0])
gammaG = [[float(x) for x in gammaG[0][i].split()] for i
in range(lr) ]
GG = [[float(x) for x in GG[0][i].split()] for i in
range(lr) ]

gammaG = [gammaG[i][0] for i in range(len(gammaG))]
GG = [GG[i][0] for i in range(len(GG))]

#### plot backbone
fig, bb = plt.subplots(figsize=(10,7))
bb.set_title('Backbone Curve')
bb.set_ylabel('Gs [kPa]')
bb.set_xlabel('log'+' '+chr(947)+' '+'[-]')
bb.plot(gammaG,GG,label=['sperimentale G0 = 124000 kPa'])
bb.plot(gamma1, Gs1,label=['pp = 1 kpa'])
bb.grid(True)
bb.grid(which='minor')
bb.set_xscale('log')
plt.legend()
# Mostra il grafico
plt.savefig('pic_BackBone1.png')
plt.show()

#### plot backbone
fig, bb1 = plt.subplots(figsize=(10,7))
bb1.set_title('Backbone Curve')
bb1.set_ylabel('Gs [kPa]')
bb1.set_xlabel('log'+' '+chr(947)+' '+'[-]')
bb1.plot(gammaG,GG,label=['sperimentale G0 = 124000 kPa'])
bb1.plot(gamma,Gs,label=["pp' = 500 kpa"])
bb1.grid(True)
bb1.grid(which='minor')
bb1.set_xscale('log')
plt.legend()
# Mostra il grafico
plt.savefig('pic_BackBone.png')
plt.show()
```

```
np.savetxt('Gs1.txt',Gs1)
np.savetxt('Gs.txt',Gs)
np.savetxt('gamma_Gs1.txt',gamma1)
np.savetxt('gamma_Gs.txt',gamma)
```

APPENDICE 3

```
     set meshFile [open test.msh w]
     set nodesInfo [open nodesInfo.dat w]
     set elementInfo [open elementsInfo.dat w]

     puts $meshFile "MESH ffBrick dimension 3 El-
emType Hexahedra Nnode 8"
     puts $meshFile "Coordinates"
     puts $meshFile "#node_number   coord_x   coord_y
coord_z"

     puts $meshFile  "1 0.0 0.0 0.0 "
     puts $meshFile  "2 1.0 0.0 0.0 "
     puts $meshFile  "3 1.0 1.0 0.0 "
     puts $meshFile  "4 0.0 1.0 0.0 "
     puts $meshFile  "5 0.0 0.0 1.0 "
     puts $meshFile  "6 1.0 0.0 1.0 "
     puts $meshFile  "7 1.0 1.0 1.0 "
     puts $meshFile  "8 0.0 1.0 1.0 "

     puts $nodesInfo  "1 0.0 0.0 0.0 "
     puts $nodesInfo  "2 1.0 0.0 0.0 "
     puts $nodesInfo  "3 1.0 1.0 0.0 "
     puts $nodesInfo  "4 0.0 1.0 0.0 "
     puts $nodesInfo  "5 0.0 0.0 1.0 "
     puts $nodesInfo  "6 1.0 0.0 1.0 "
     puts $nodesInfo  "7 1.0 1.0 1.0 "
     puts $nodesInfo  "8 0.0 1.0 1.0 "

     puts $meshFile "end coordinates"
     puts $meshFile "Elements"
     puts $meshFile "# element   node1   node2
node3   node4   node5   node6   node7   node8"

     puts $elementInfo "1 1 2 3 4 5 6 7 8"

     puts $meshFile  "1 1 2 3 4 5 6 7 8"
```

```
     puts $meshFile "end elements"
     close $meshFile
     close $elementInfo
     close $nodesInfo

     ##### START
########################################
     wipe

     set ndm 3
     set ndf 4

     model BasicBuilder -ndm $ndm -ndf $ndf
     #### SET TARATURA

     set peakShearStrain  0.1
     set poisson  0.3

     set G1 125000

     set E1 [expr $G1*(2.0*(1+$poisson))]
     set B1 [expr $E1*(3.0*(1-2*$poisson))]

     set alpha [expr (1.0/(4.0 + ($B1 +
(4.0/3.0)*$G1)))]
     puts "alpha = $alpha"

     set p_ref 500.0
     set frictionAng 0.0
     set cohesion 270
     set pressDependCoe 0.0

     ################################################
####
     set Bfluid 2.2e6
     set fluid1 1
     set solid1 10
     set fmass 1.0
```

```
     #set fmass 0.0
     #valroe perm testato 1.0e-15
     set perm 1.0e-4
     ################################################
####
     set rhoS 1.97
     set rhoF 1.0
     ################################################
####
     set densityMult 1.0
     ################################################
####

     set massProportionalDamping 1.0
     set InitialStiffnessProportionalDamping 1.0

     set bUnitWeightX 0
     set bUnitWeightY 0
     set bUnitWeightZ 0
     ################################################
####
     #C15i

     nDMaterial PressureIndependMultiYield $solid1
$ndm $rhoS $G1 $B1 $cohesion $peakShearStrain $fric-
tionAng  $p_ref $pressDependCoe 40

     nDMaterial InitialStateAnalysisWrapper 2 $solid1
3

     node 1 0.0 0.0 0.0
     node 4 0.0 1.0 0.0
     node 3 1.0 1.0 0.0
     node 2 1.0 0.0 0.0
     node 5 0.0 0.0 1.0
     node 8 0.0 1.0 1.0
     node 7 1.0 1.0 1.0
     node 6 1.0 0.0 1.0
```

```
    fix 1 0 0 1 1
    fix 4 0 0 1 1
    fix 3 0 0 1 1
    fix 2 0 0 1 1
    fix 5 0 0 0 1
    fix 6 0 0 0 1
    fix 7 0 0 0 1
    fix 8 0 0 0 1


    element brickUP 1 1 2 3 4 5 6 7 8 2 $B1 $fmass
$perm $perm $perm $bUnitWeightX $bUnitWeightY $bUnit-
WeightZ


    updateMaterialStage -material 2 -stage 0


    set durata 10
    set dt 0.1
    set numSteps [expr int($durata/$dt)]

    puts "INITAL ANALYSIS START"
    constraints Penalty 1e18 1e18
    numberer RCM
    system ProfileSPD
    algorithm ModifiedNewton
    test NormDispIncr 1.0e-5 600 2
    integrator Newmark 0.5 0.25
    analysis Transient

    InitialStateAnalysis on

    set load_f 500.0
    set patternTag 2
    set tsTag 1
    set linCfact 0.1

    set durata 10
```

```
     set dt 0.1
     set numSteps [expr int($durata/$dt)]


     timeSeries Constant $tsTag -factor 1
     pattern Plain $patternTag $tsTag -factor 1 {

     load 1  [expr (1.0/4.0)*$load_f]   [expr
(1.0/4.0)*$load_f]  [expr (1.0/4.0)*$load_f] 0
     load 2  [expr -(1.0/4.0)*$load_f]  [expr
(1.0/4.0)*$load_f]  [expr (1.0/4.0)*$load_f] 0
     load 3  [expr -(1.0/4.0)*$load_f]  [expr -
(1.0/4.0)*$load_f] [expr (1.0/4.0)*$load_f] 0
     load 4  [expr (1.0/4.0)*$load_f]   [expr -
(1.0/4.0)*$load_f] [expr (1.0/4.0)*$load_f] 0
     load 5  [expr (1.0/4.0)*$load_f]   [expr
(1.0/4.0)*$load_f]  [expr -(1.0/4.0)*$load_f] 0
     load 8  [expr (1.0/4.0)*$load_f]   [expr -
(1.0/4.0)*$load_f] [expr -(1.0/4.0)*$load_f] 0
     load 7  [expr -(1.0/4.0)*$load_f]  [expr -
(1.0/4.0)*$load_f] [expr -(1.0/4.0)*$load_f] 0
     load 6  [expr -(1.0/4.0)*$load_f]  [expr
(1.0/4.0)*$load_f] [expr -(1.0/4.0)*$load_f] 0
     }

     analyze $numSteps $dt
     InitialStateAnalysis off

     puts "INITAL ANALYSIS END"
     #recorder mpco "phase_0.mpco" -N "displacement"
"pressure" -E "material.stress" "material.strain" "ma-
terial.pressure"
     recorder Element -ele 1  -file  0stress1.out ma-
terial 1 stress

     updateMaterialStage -material 2 -stage 1
     updateMaterialStage -material $solid1 -stage 1

     equalDOF 5 6 3
```

```
equalDOF 5 7 3
equalDOF 5 8 3

equalDOF 5 6 1
equalDOF 5 7 1
equalDOF 5 8 1

equalDOF 5 6 2
equalDOF 5 7 2
equalDOF 5 8 2

equalDOF 1 2 3
equalDOF 1 3 3
equalDOF 1 4 3

equalDOF 1 2 1
equalDOF 1 3 1
equalDOF 1 4 1

equalDOF 1 2 2
equalDOF 1 3 2
equalDOF 1 4 2

set n11 [nodeDisp 1 1]
set n21 [nodeDisp 2 1]
set n31 [nodeDisp 3 1]
set n41 [nodeDisp 4 1]
set n51 [nodeDisp 5 1]
set n61 [nodeDisp 6 1]
set n71 [nodeDisp 7 1]
set n81 [nodeDisp 8 1]

set n12 [nodeDisp 1 2]
set n22 [nodeDisp 2 2]
set n32 [nodeDisp 3 2]
set n42 [nodeDisp 4 2]
set n52 [nodeDisp 5 2]
set n62 [nodeDisp 6 2]
set n72 [nodeDisp 7 2]
```

```
set n82 [nodeDisp 8 2]

set n13 [nodeDisp 1 3]
set n23 [nodeDisp 2 3]
set n33 [nodeDisp 3 3]
set n43 [nodeDisp 4 3]
set n53 [nodeDisp 5 3]
set n63 [nodeDisp 6 3]
set n73 [nodeDisp 7 3]
set n83 [nodeDisp 8 3]

timeSeries Constant 9  -factor 1.0
pattern Plain 16 9 -factor 1.0 {

     sp 1 1 $n11
     sp 2 1 $n11
     sp 3 1 $n11
     sp 4 1 $n11

     sp 5 1 $n51
     sp 6 1 $n61
     sp 7 1 $n71
     sp 8 1 $n81

     sp 1 2 $n12
     sp 2 2 $n12
     sp 3 2 $n12
     sp 4 2 $n12

     sp 5 2 $n52
     sp 6 2 $n62
     sp 7 2 $n72
     sp 8 2 $n82

}

analyze 10 1

set durata2 30.0
```

```
     set dt2 0.01
     set numSteps2 [expr int($durata2/$dt2)]
     puts "STEPS $numSteps2 dt $dt2 durata $durata2"

     set time_dyn factors_dyn.out
     timeSeries Path 10 -dt $dt2 -filePath $time_dyn
-factor 1.0 -startTime 10.0
     pattern Plain [expr ($patternTag+1)] 10 -factor
1.0 {

     sp 5 1 [expr 1 + $n51]
     sp 6 1 [expr 1 + $n61]
     sp 7 1 [expr 1 + $n71]
     sp 8 1 [expr 1 + $n81]

     sp 1 1 [expr -1 + $n11]
     sp 2 1 [expr -1 + $n21]
     sp 3 1 [expr -1 + $n31]
     sp 4 1 [expr -1 + $n41]

     }

     puts "spostamento nodo 5 [nodeDisp 5 1]"
     recorder Node -file displacement.out -time -dT
$dt2 -node 1 2 3 4 5 6 7 8 -dof 1 2 3 disp
     recorder Node -file velocity.out -time -dT $dt2
-node 1 2 3 4 5 6 7 8 -dof 1 2 3 vel
     recorder Node -file acceleration.out -time -dT
$dt2 -node 1 2 3 4 5 6 7 8 -dof 1 2 3 accel
     recorder Node -file porePressure.out -time -dT
$dt2 -node 1 2 3 4 5 6 7 8 -dof 4 vel

     recorder Element -ele 1  -file  stress1.out -
time -dT $dt2 material 1 stress
     recorder Element -ele 1  -file  stress2.out -
time -dT $dt2 material 2 stress
     recorder Element -ele 1  -file  stress3.out -
time -dT $dt2 material 3 stress
```

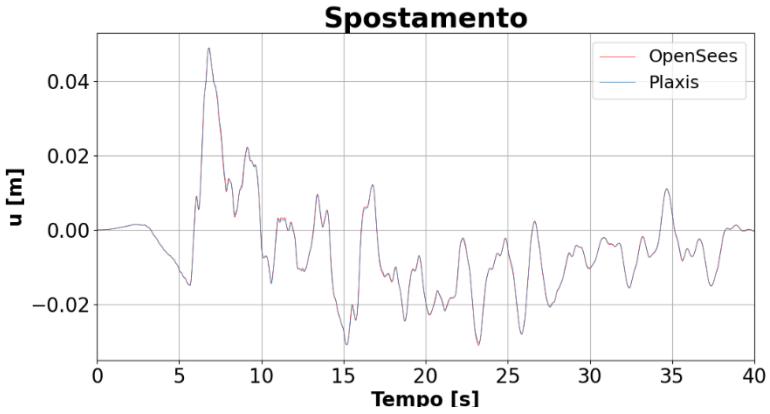
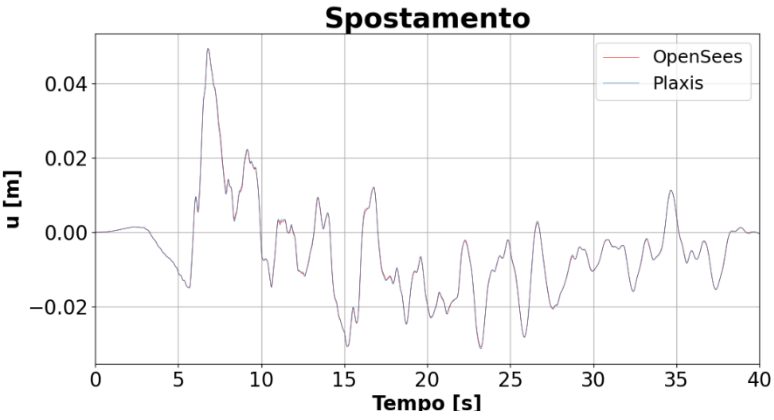
```
     recorder Element -ele 1  -file  stress4.out -
time -dT $dt2 material 4 stress
     recorder Element -ele 1  -file  stress5.out -
time -dT $dt2 material 5 stress
     recorder Element -ele 1  -file  stress6.out -
time -dT $dt2 material 6 stress
     recorder Element -ele 1  -file  stress7.out -
time -dT $dt2 material 7 stress
     recorder Element -ele 1  -file  stress8.out -
time -dT $dt2 material 8 stress

     recorder Element -ele 1  -file  strain1.out -
time -dT $dt2 material 1 strain
     recorder Element -ele 1  -file  strain2.out -
time -dT $dt2 material 2 strain
     recorder Element -ele 1  -file  strain3.out -
time -dT $dt2 material 3 strain
     recorder Element -ele 1  -file  strain4.out -
time -dT $dt2 material 4 strain
     recorder Element -ele 1  -file  strain5.out -
time -dT $dt2 material 5 strain
     recorder Element -ele 1  -file  strain6.out -
time -dT $dt2 material 6 strain
     recorder Element -ele 1  -file  strain7.out -
time -dT $dt2 material 7 strain
     recorder Element -ele 1  -file  strain8.out -
time -dT $dt2 material 8 strain

     recorder Element -ele 1 -time -file backbone.out
-dT $dt2 material 4 backbone 500
     recorder Element -ele 1 -time -file back-
bone1.out -dT $dt2 material 1 backbone 1.0

     rayleigh $massProportionalDamping 0.0 $Ini-
tialStiffnessProportionalDamping 0.0


     constraints Transformation
     numberer RCM
```

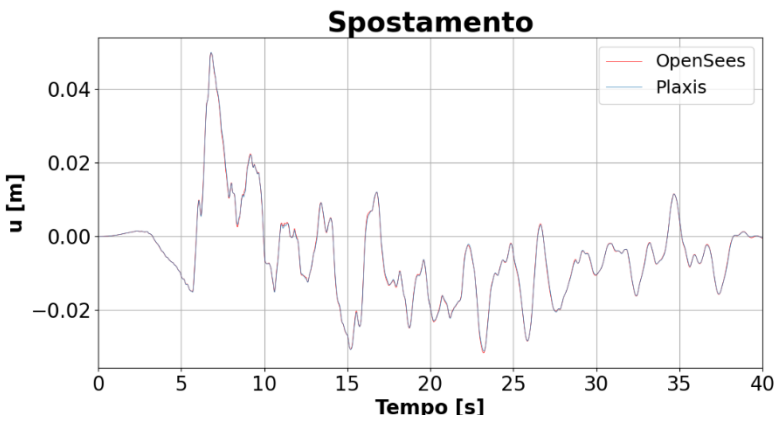
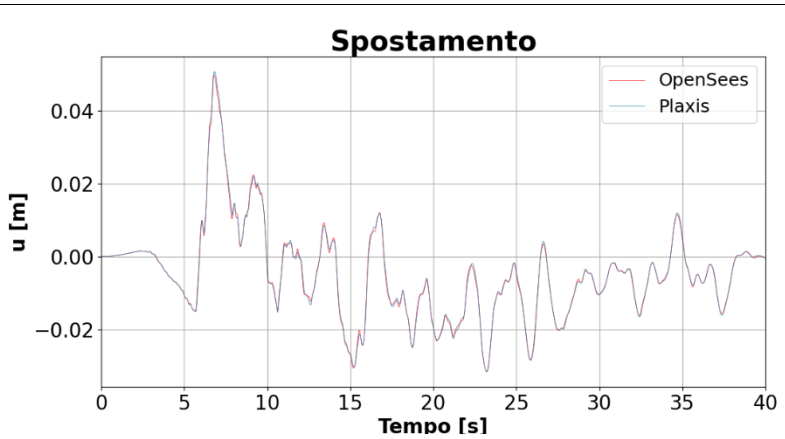
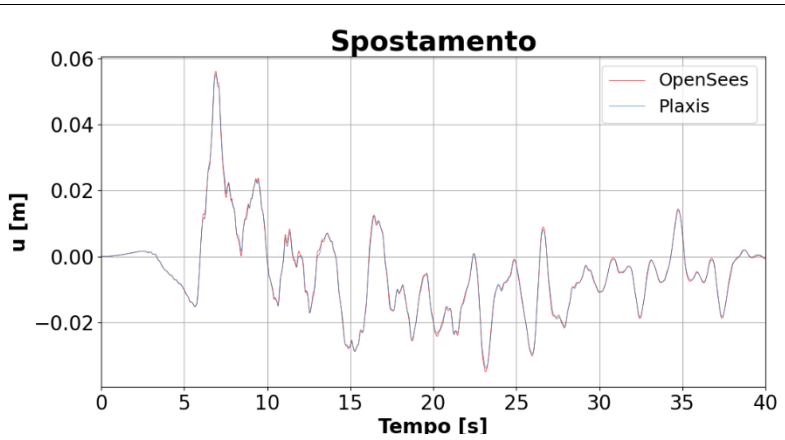
```
#constraints Penalty 1e18 1e18
#numberer Plain
system ProfileSPD
algorithm ModifiedNewton
test NormDispIncr 1.0E-09 600 2
integrator Newmark 0.5 0.25
analysis Transient
analyze $numSteps2 $dt2

puts "FINE DELLE ANALISI"
wipe
```

APPENDICE 4

RISULTATI ANALISI VISCO-ELASTICHE

VERTICALE VALLE (X = 540.0 m)

Punto	Spostamento Orizzontale
Y = 0 m	
Y = 10 m	

Y = 20 m	
Y = 30 m	
Y = 40 m	

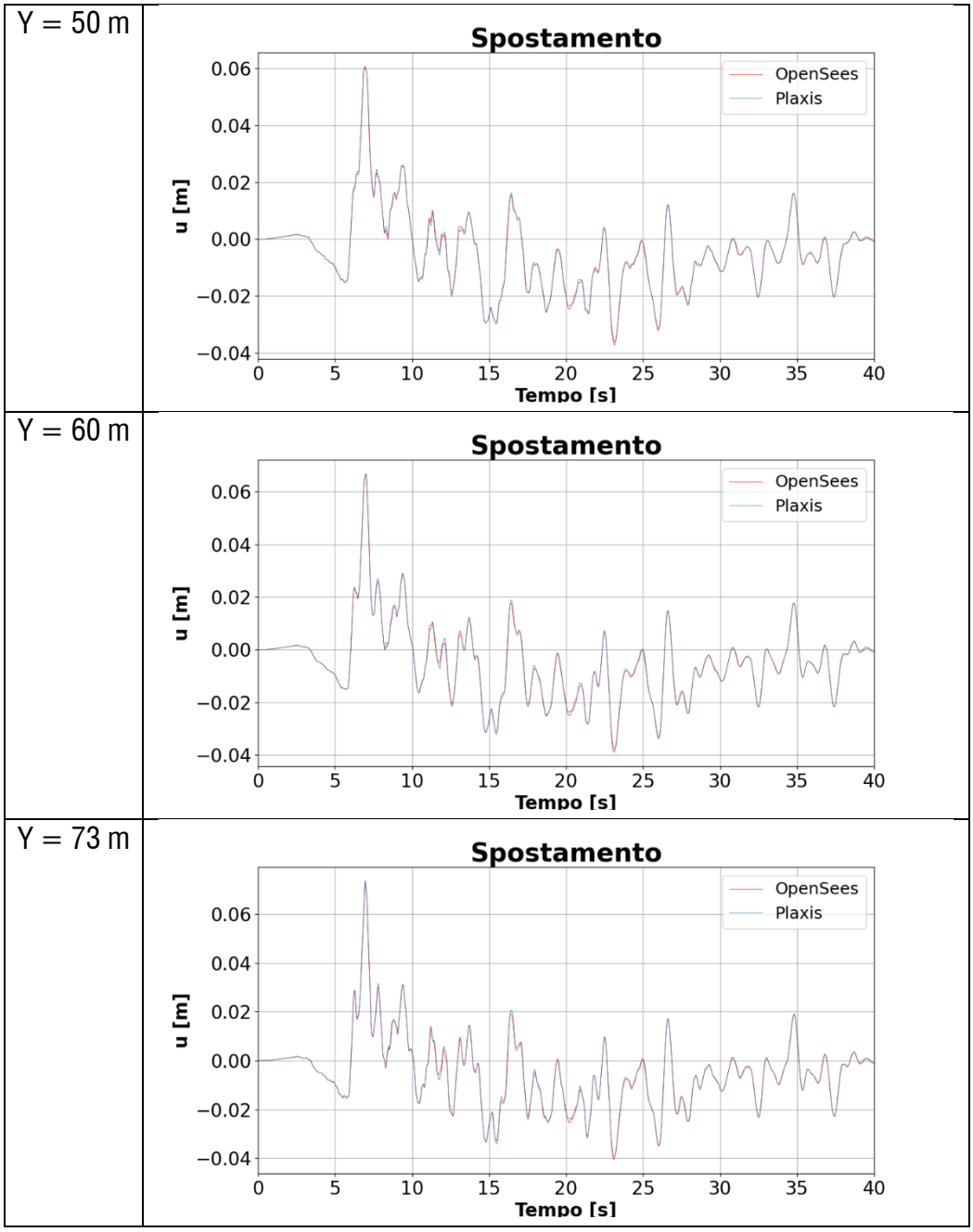
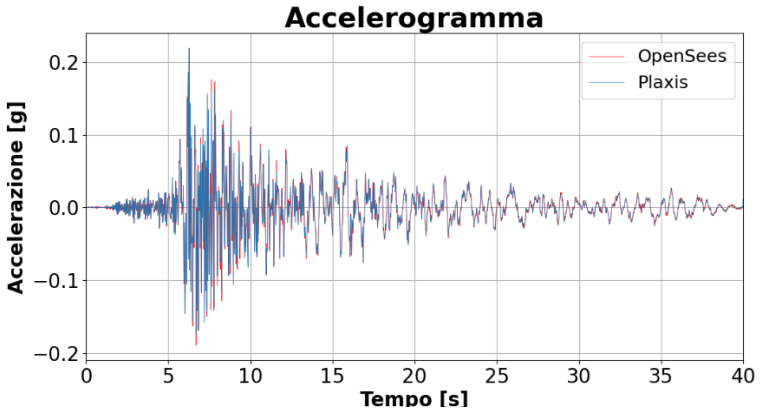
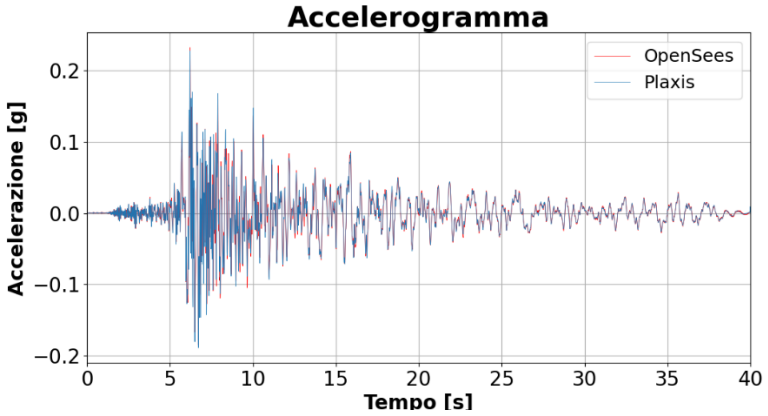
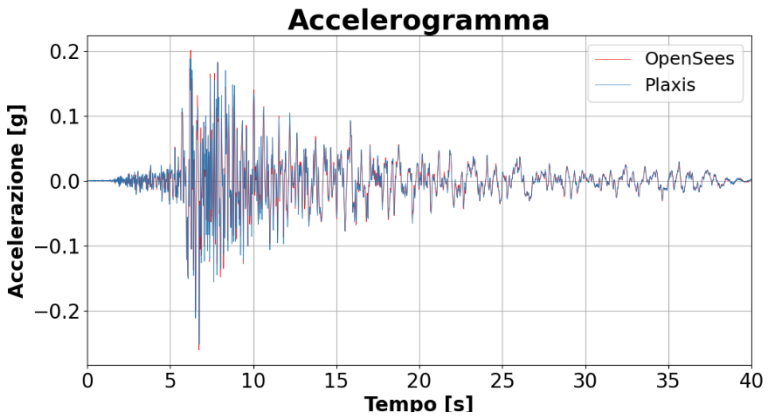
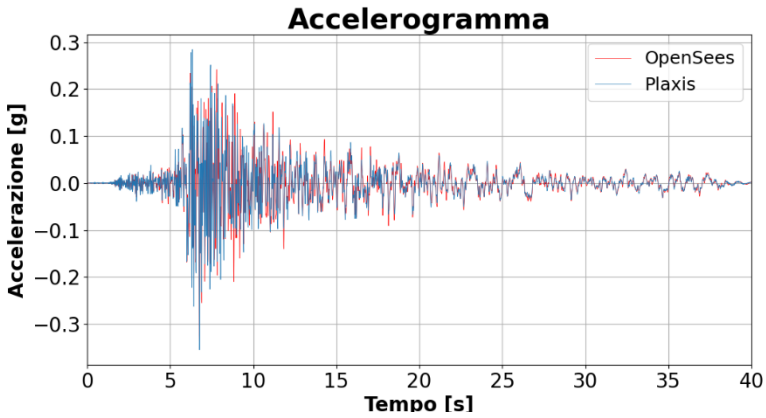
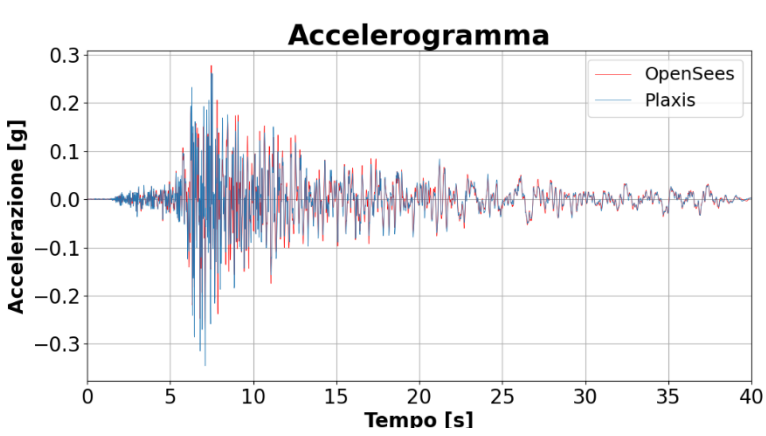
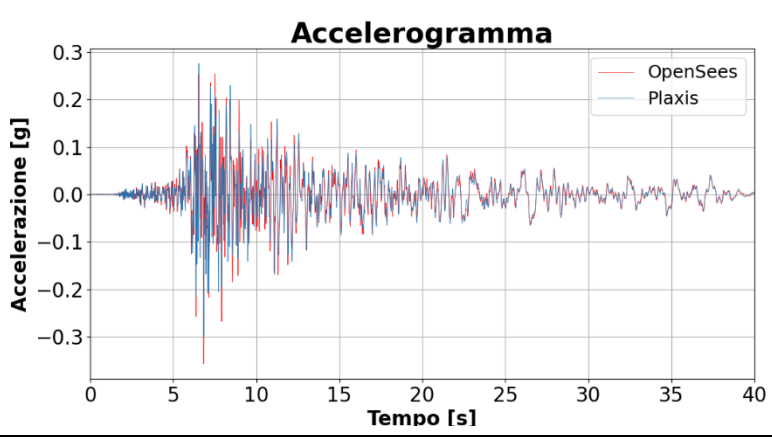


Fig. 66 Sezione X = 540.0 m, spostamenti orizzontali.

Punto	Accelerazione Orizzontale
Y = 0 m	
Y = 10 m	
Y = 20 m	

Y = 30 m	
Y = 40 m	
Y = 50 m	

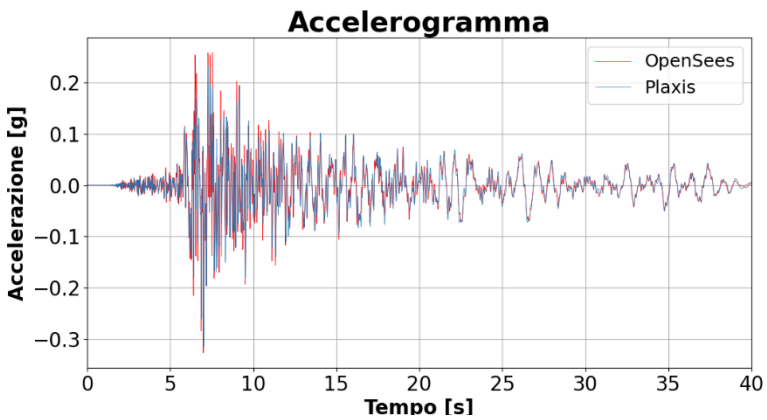
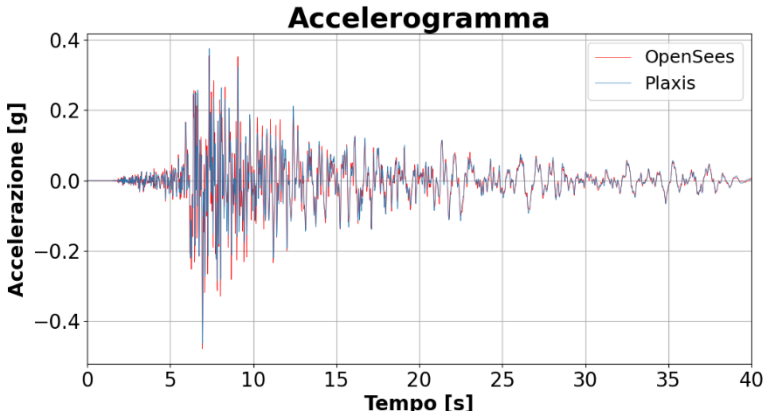
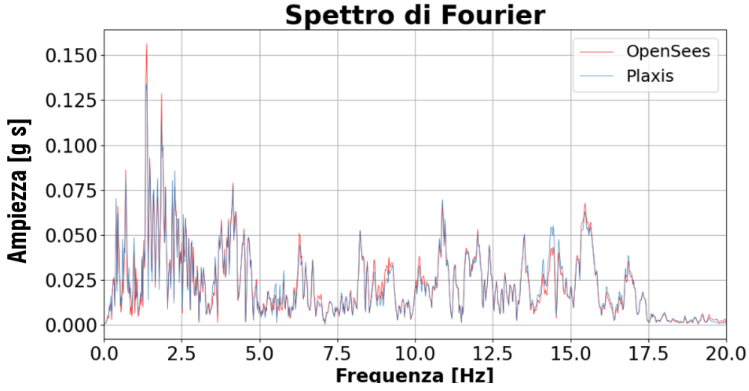
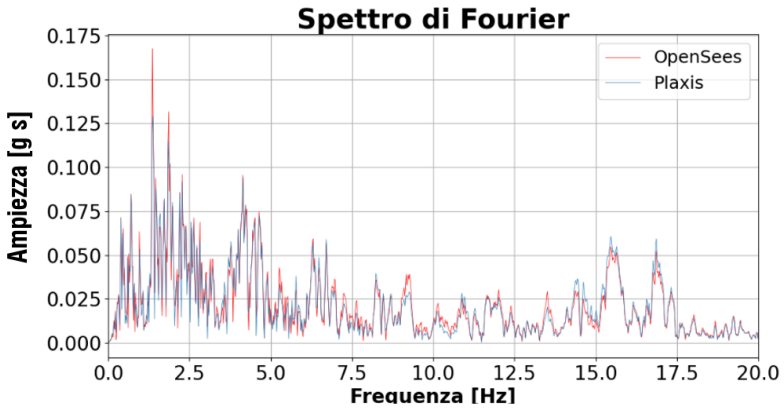
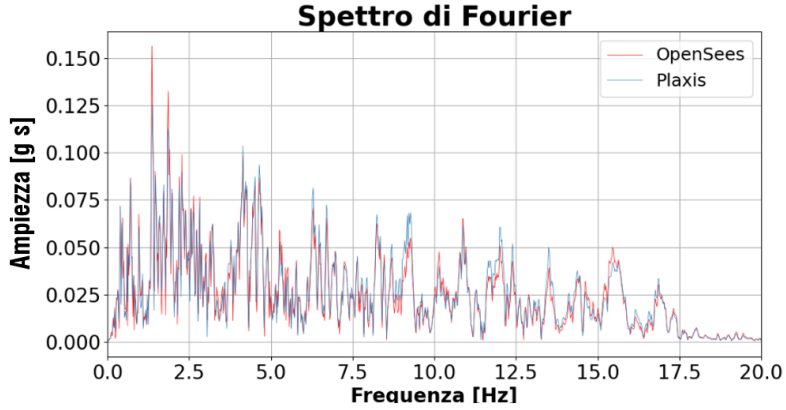
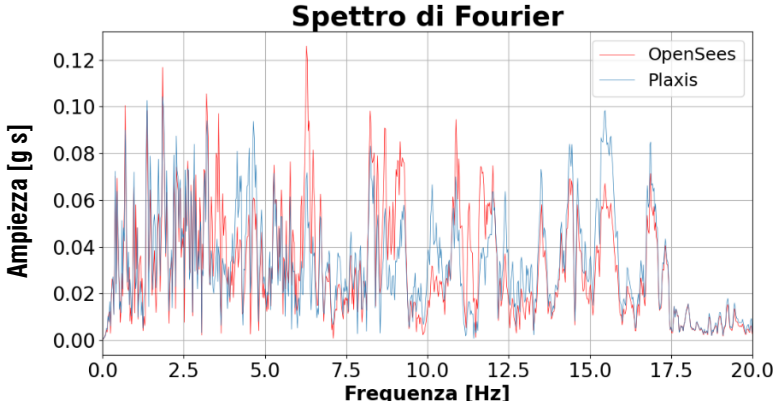
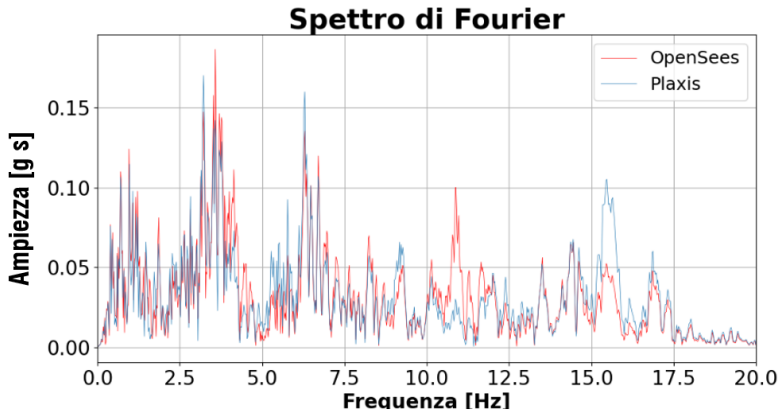
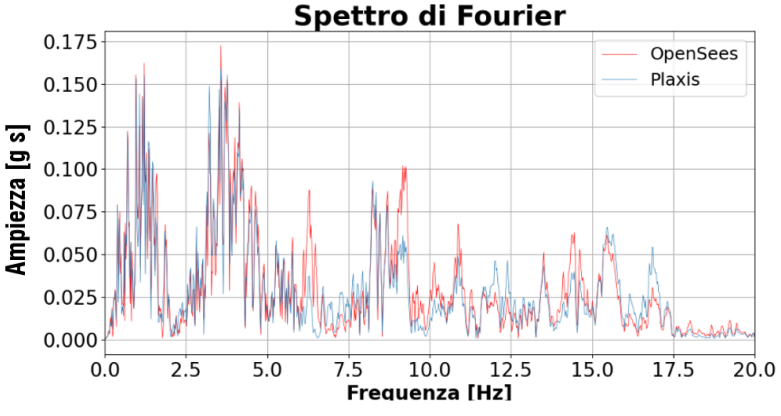
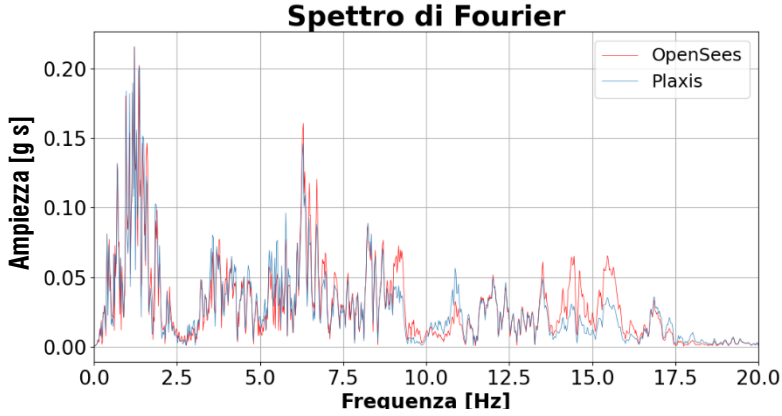
Y = 60 m	
Y = 73 m	

Fig. 67 Sezione X = 540.0 m, accelerazioni orizzontali.

Punto	Spettro di Fourier
Y = 0 m	

Y = 10 m	
Y = 20 m	
Y = 30 m	

Y = 40 m	
Y = 50 m	
Y = 60 m	

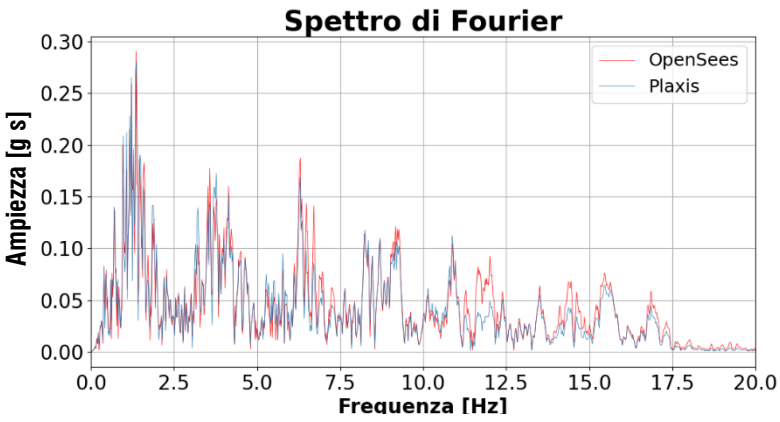
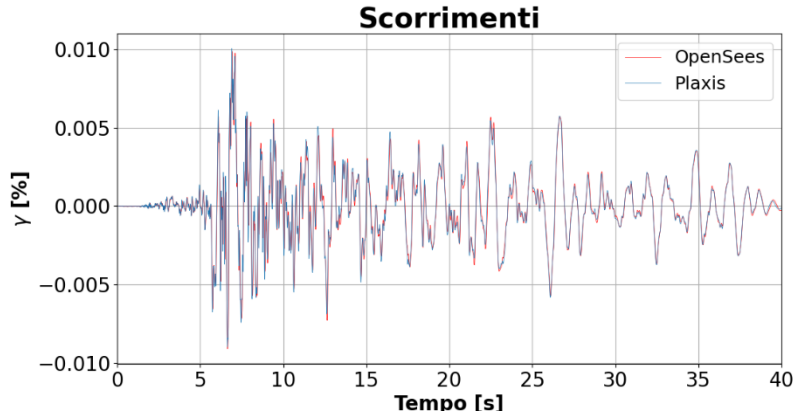
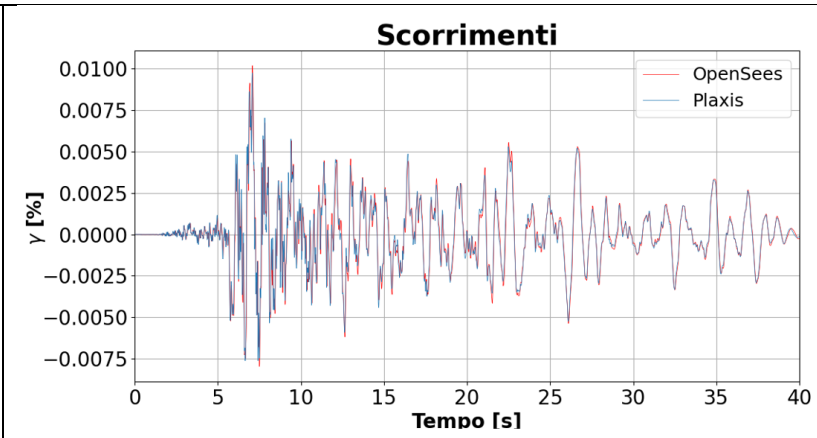
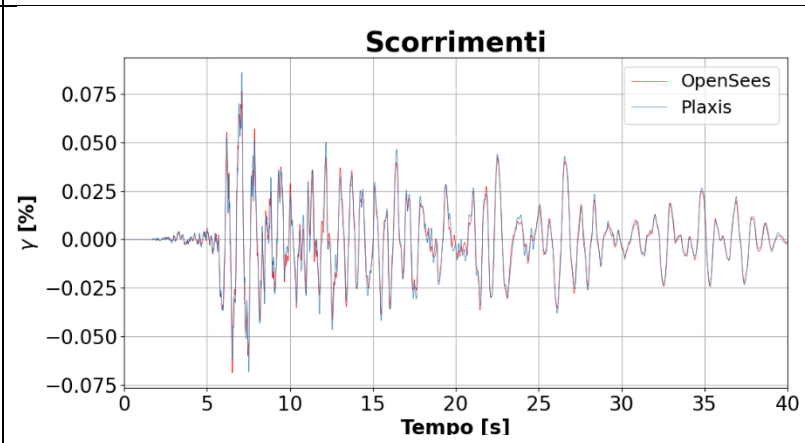
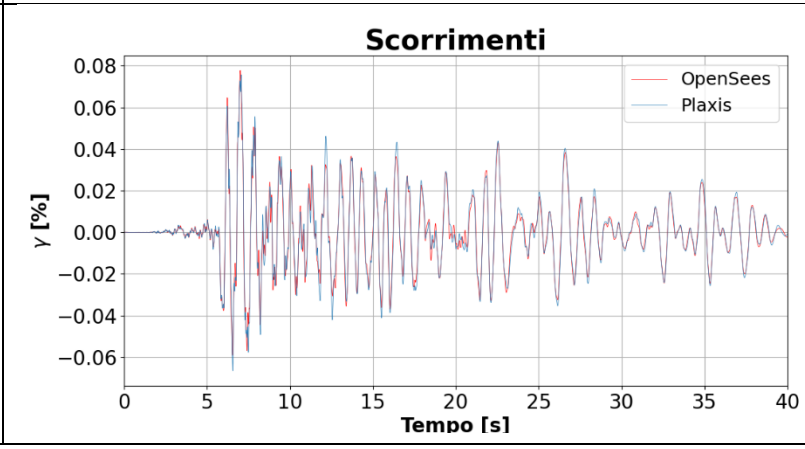
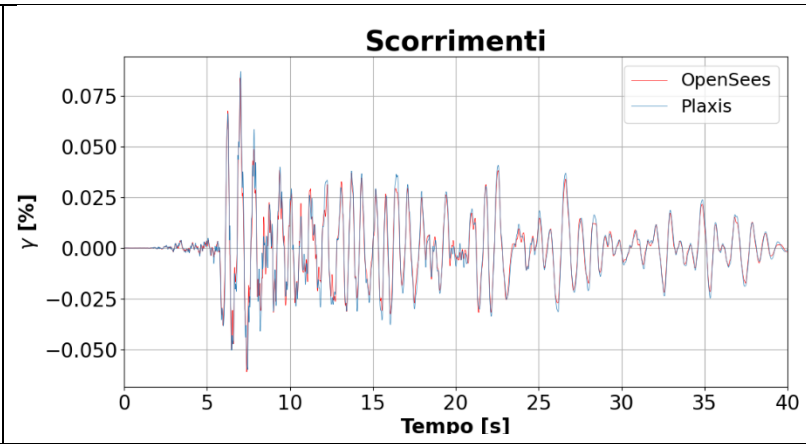
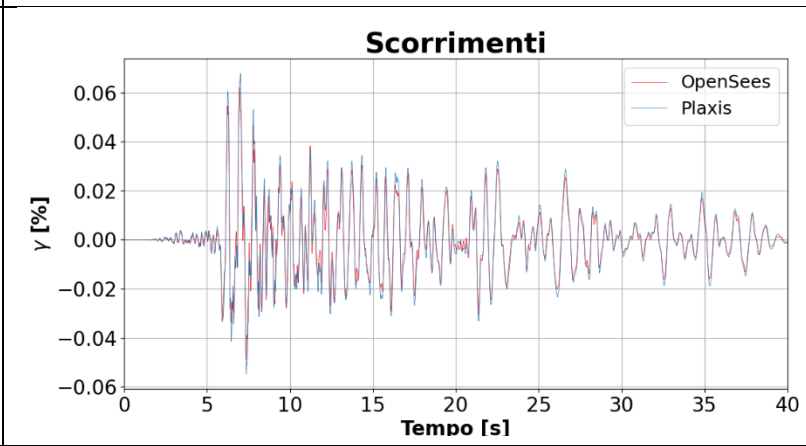
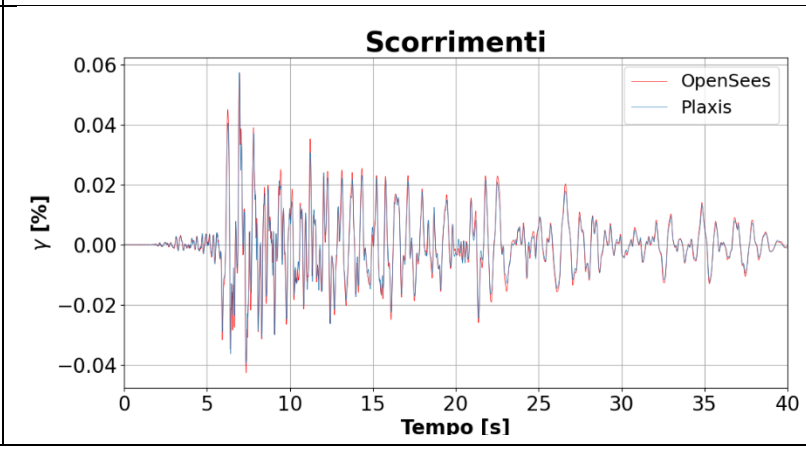
Y = 73 m	

Fig. 68 Sezione X = 540.0 m, spettri di Fourier.

Punto	Scorrimenti orizzontali
Y = 0 m	

Y = 10 m	
Y = 20 m	
Y = 30 m	

Y = 40 m	
Y = 50 m	
Y = 60 m	

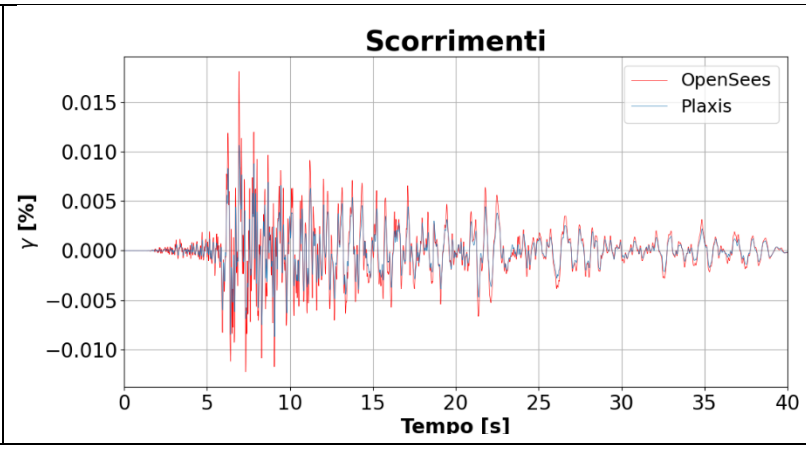
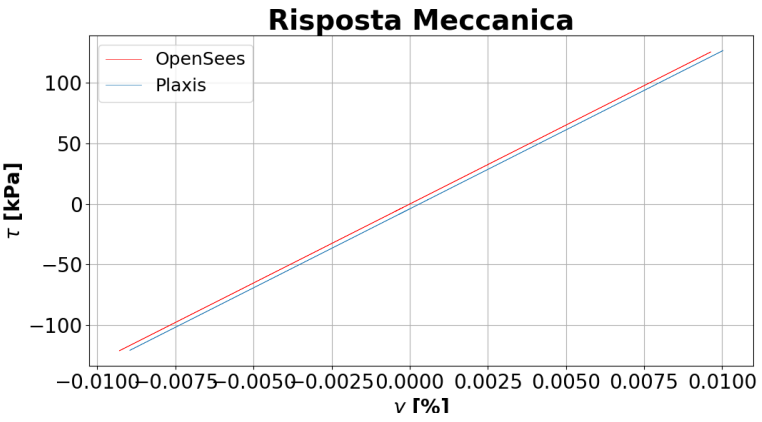
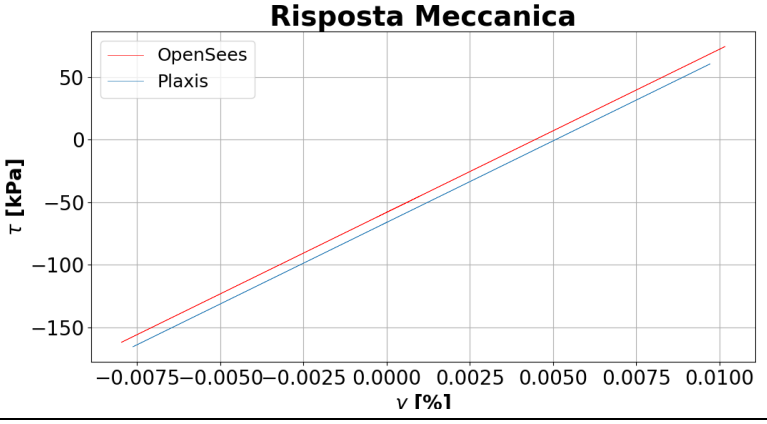
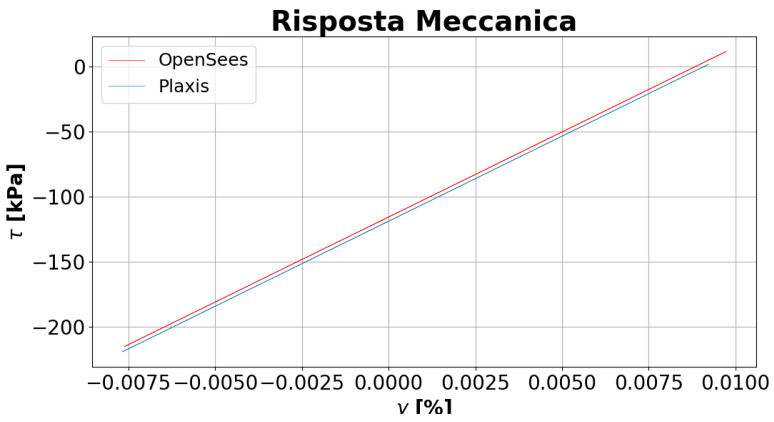
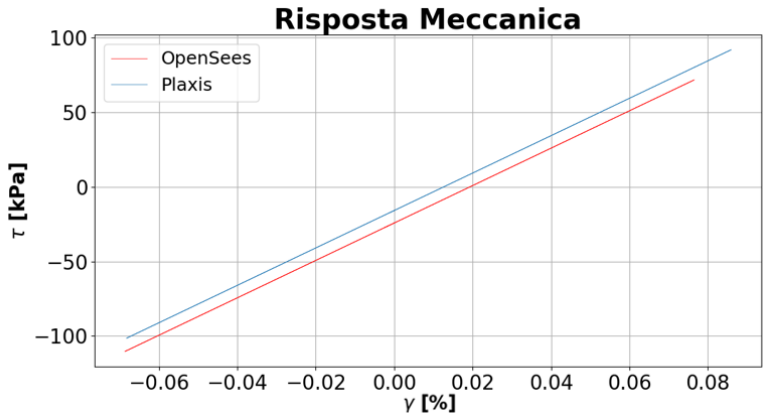
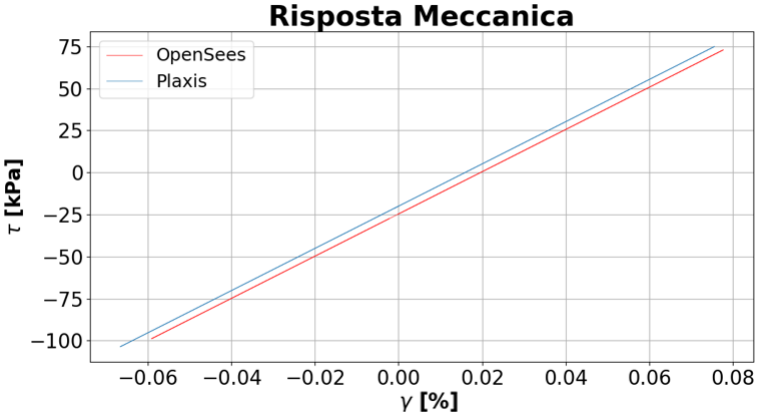
Y = 73 m	

Fig. 69 Sezione X = 540.0 m, scorrimenti orizzontali.

Punto	Risposta meccanica
Y = 0 m	
Y = 10 m	

Y = 20 m	
Y = 30 m	
Y = 40 m	

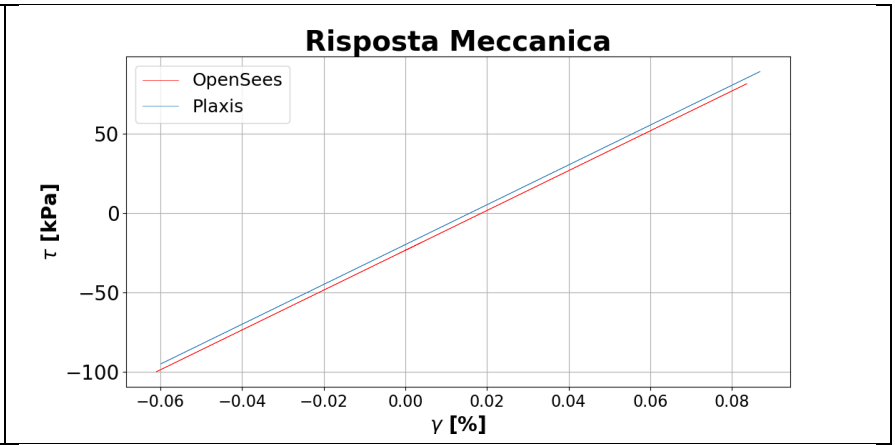
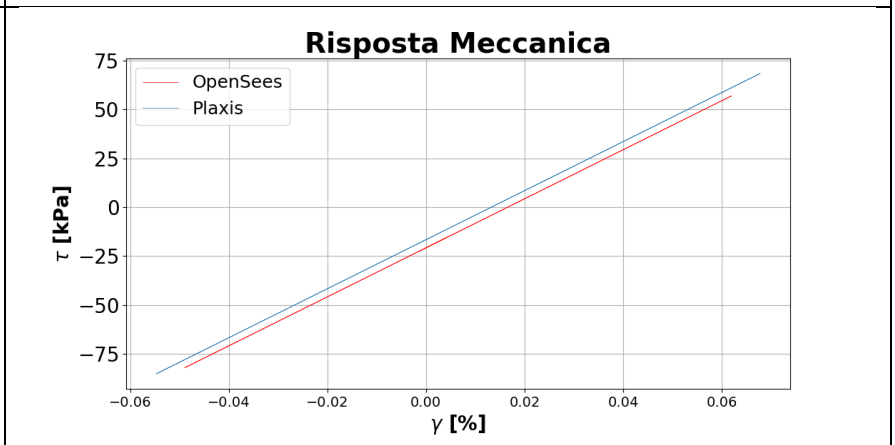
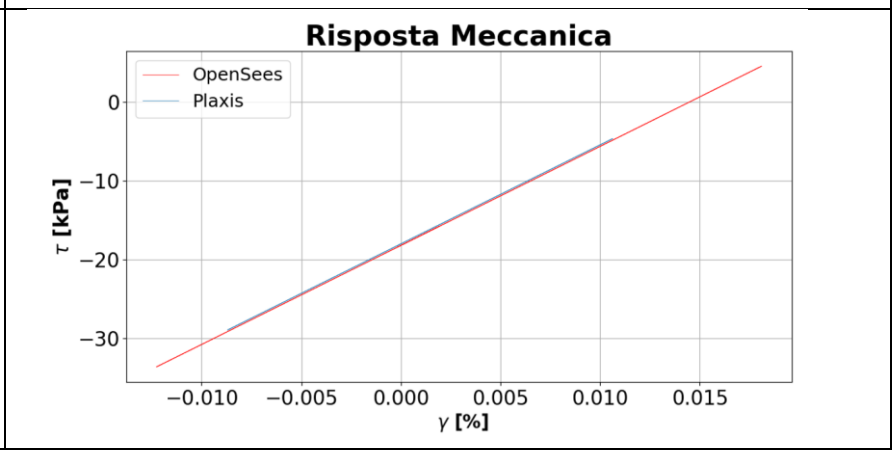
Y = 50 m	
Y = 60 m	
Y = 73 m	

Fig. 70 Sezione X = 540.0 m, risposta meccanica.

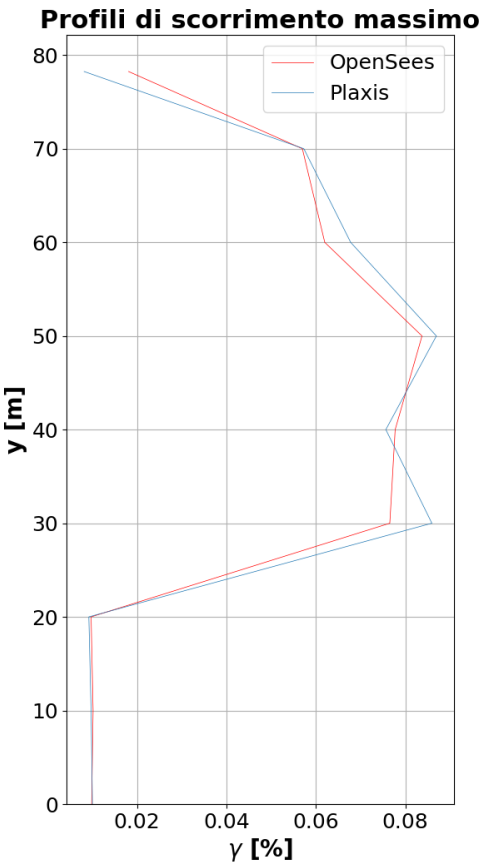
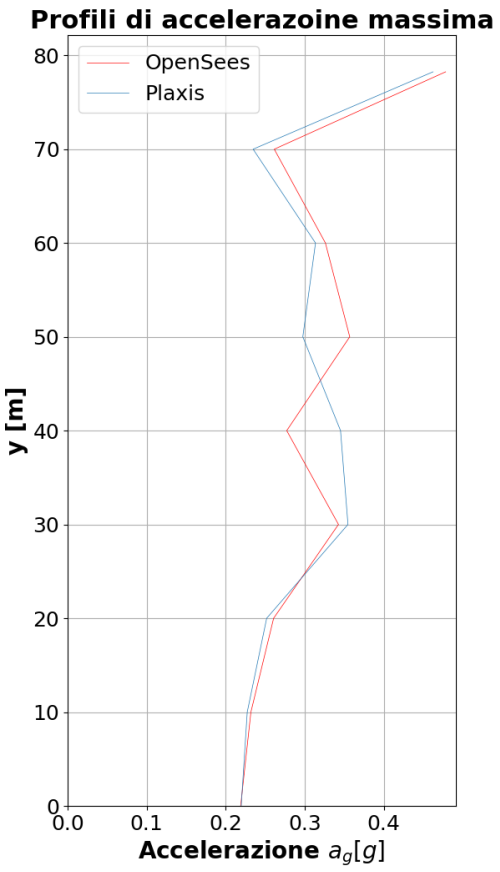
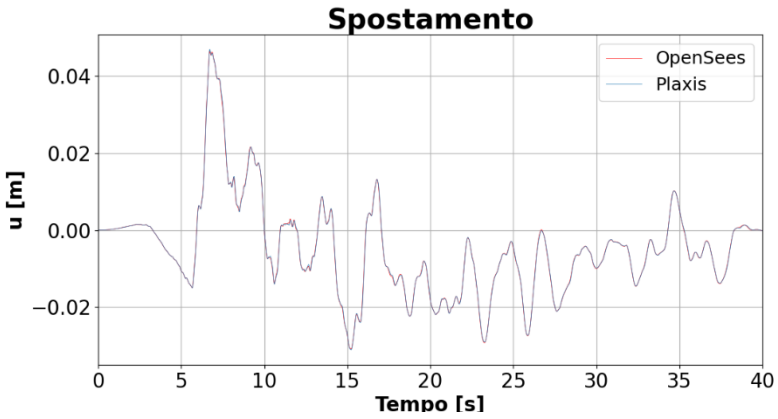
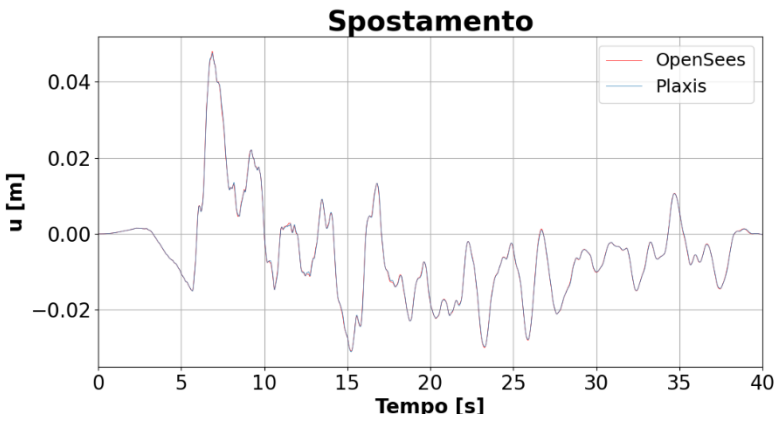
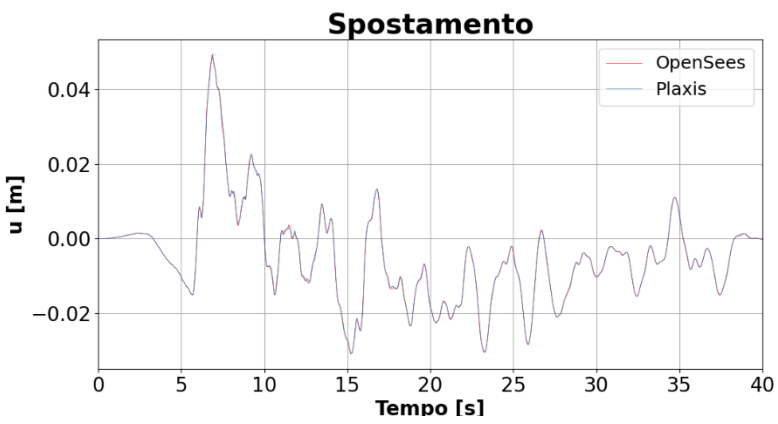
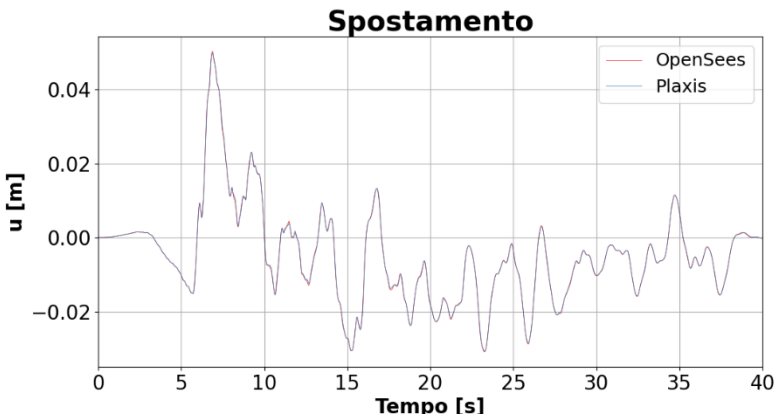
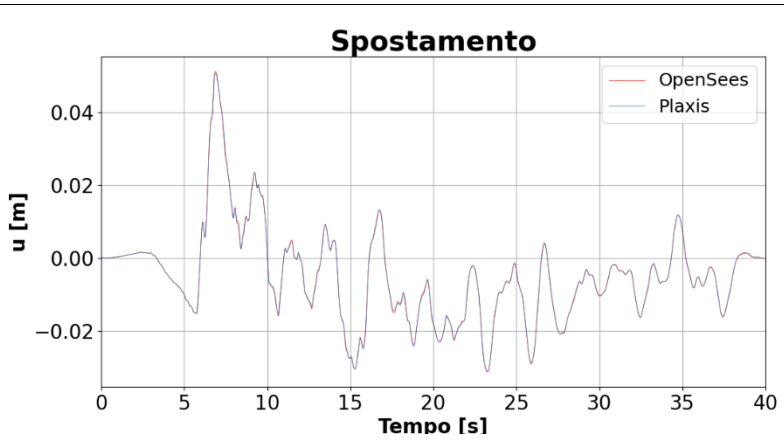
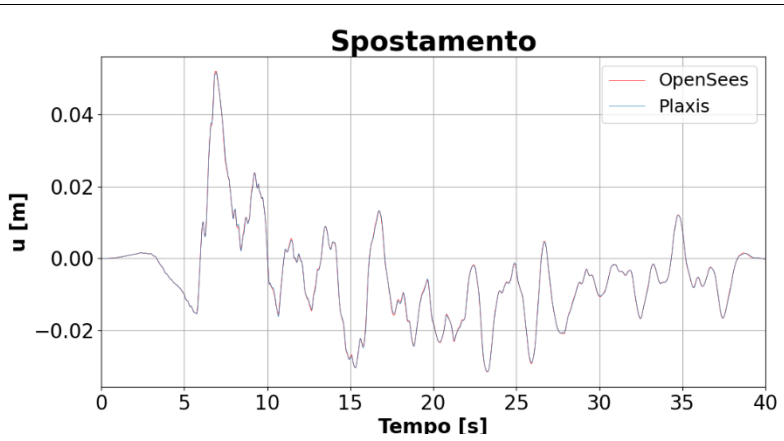
Profilo degli scorrimenti massimi	Profilo delle accelerazioni massime
	

Fig. 71 Sezione X = 540.0 m, profili con la profondità.

VERTICALE MEDIANA (X = 870.0 m)

Punto	Spostamento Orizzontale
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

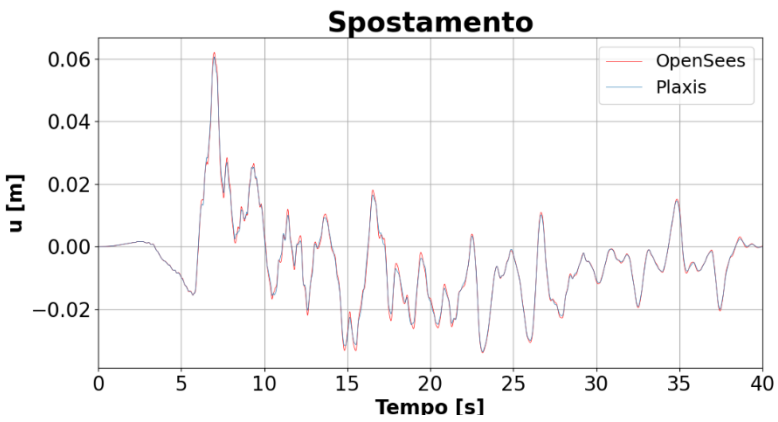
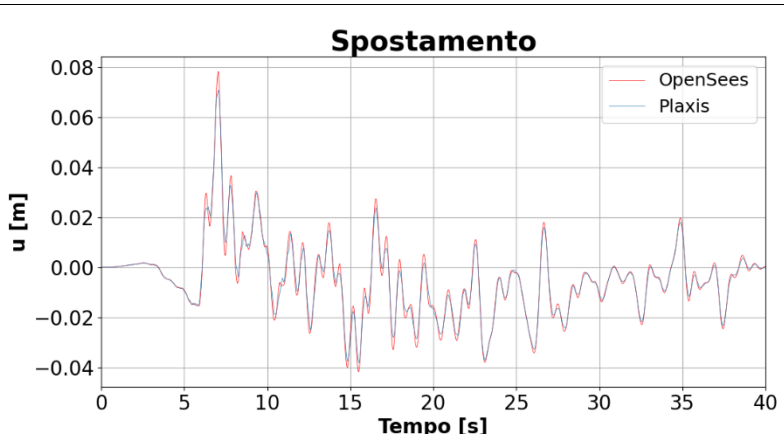
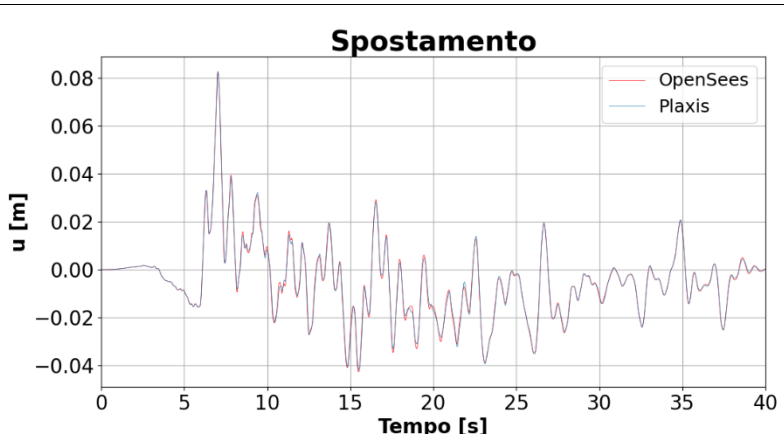
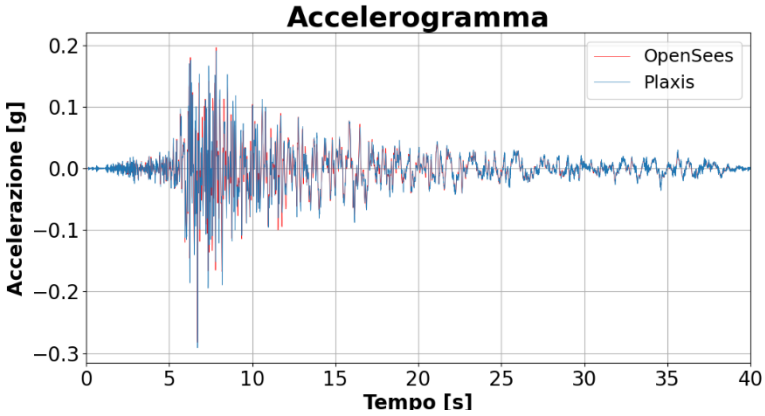
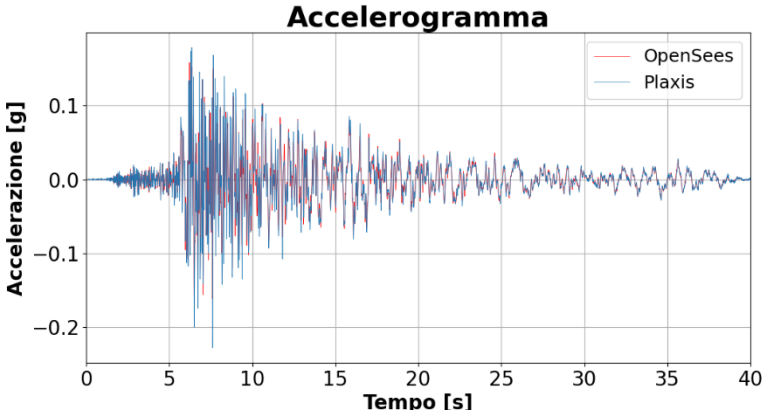
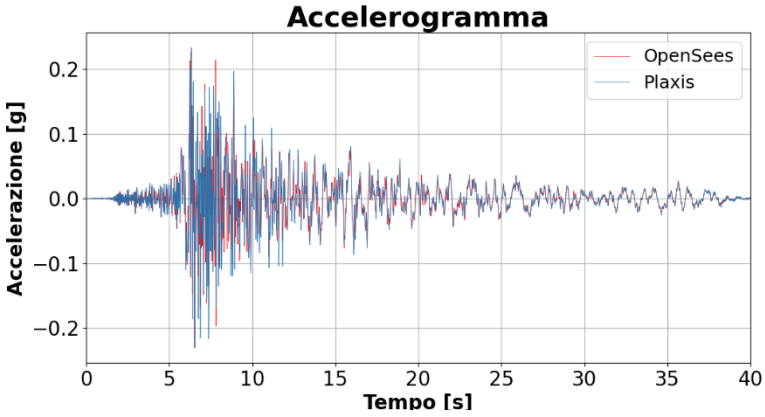
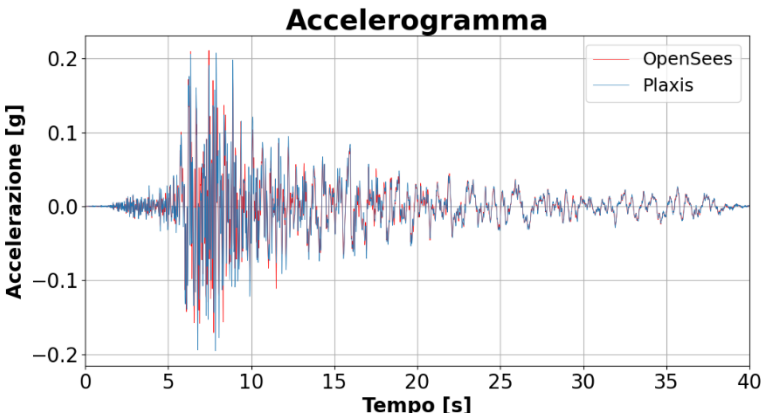
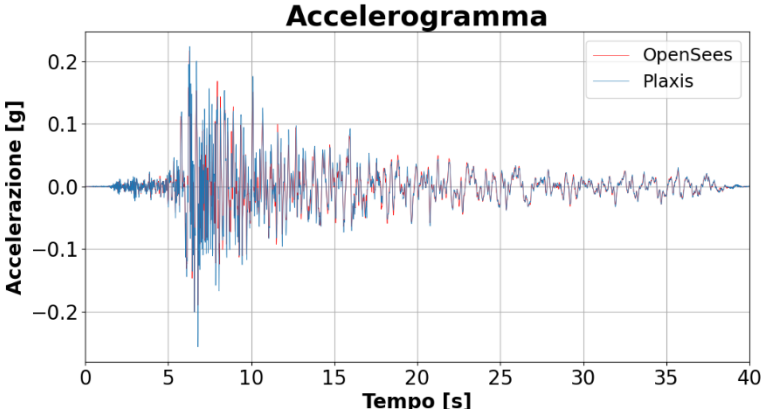
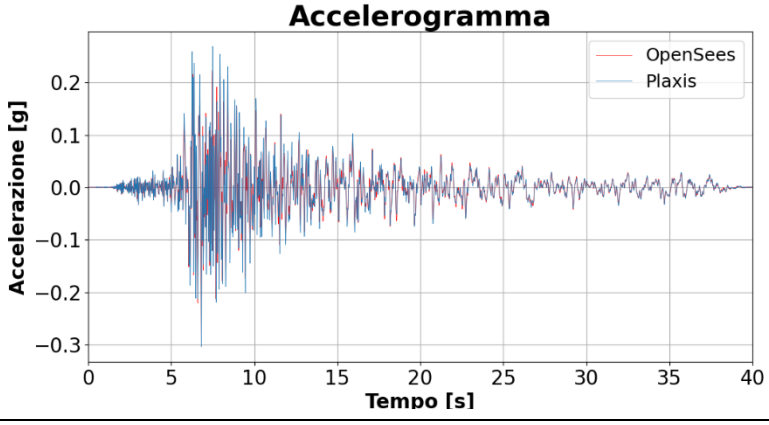
Y = 90 m	
Y = 105 m	
Y = 129 m	

Fig. 72 Sezione X = 870.0 m, spostamenti orizzontali.

Punto	Accelerazione Orizzontale
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

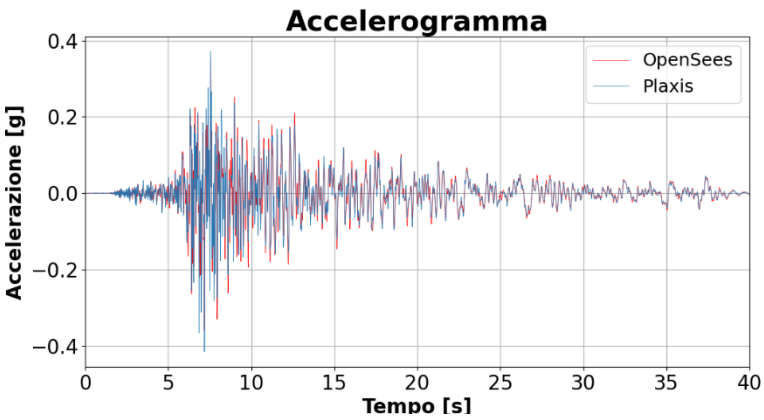
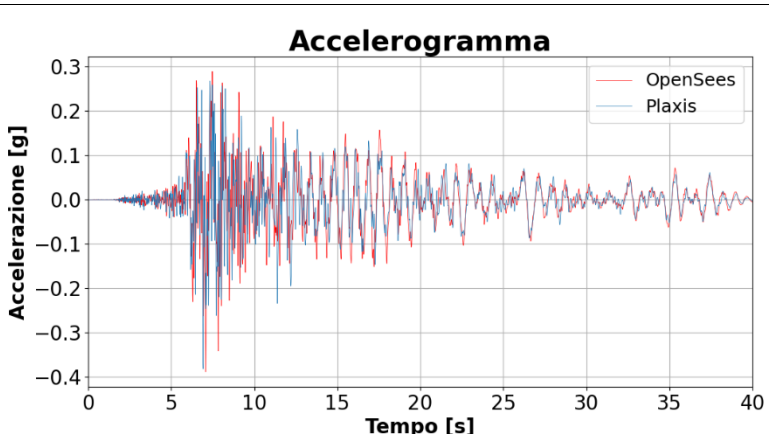
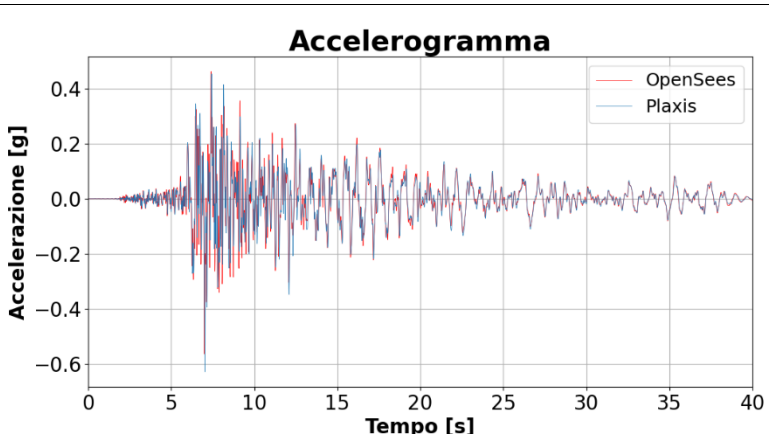
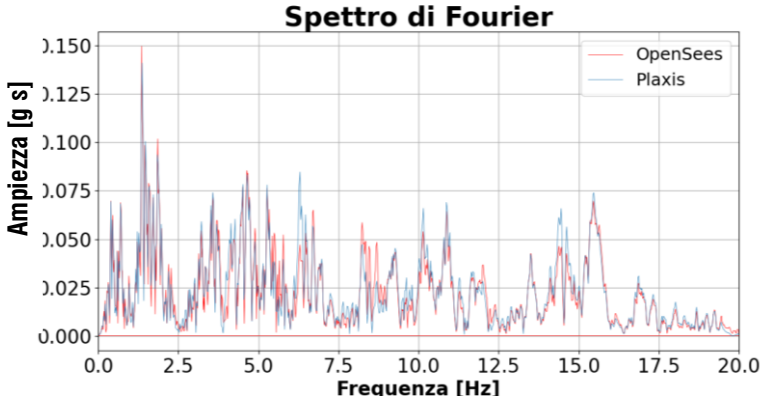
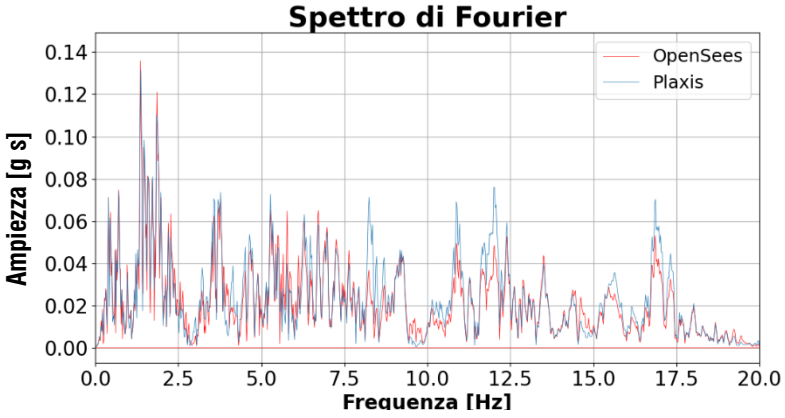
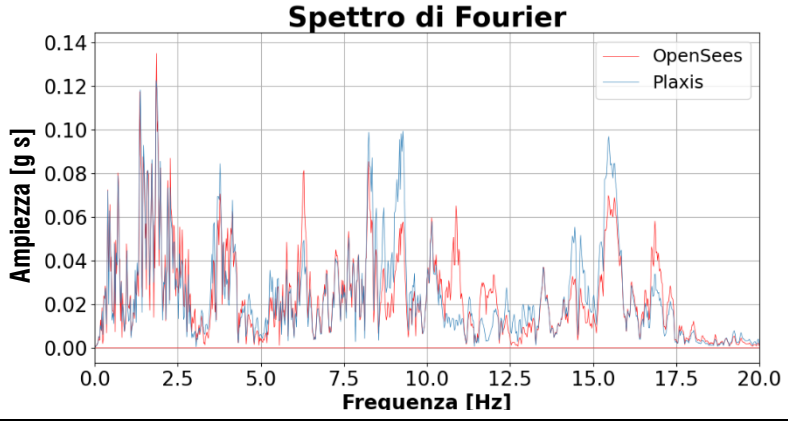
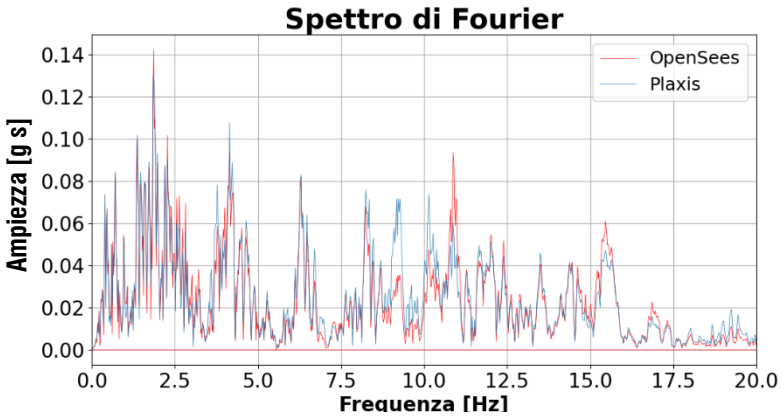
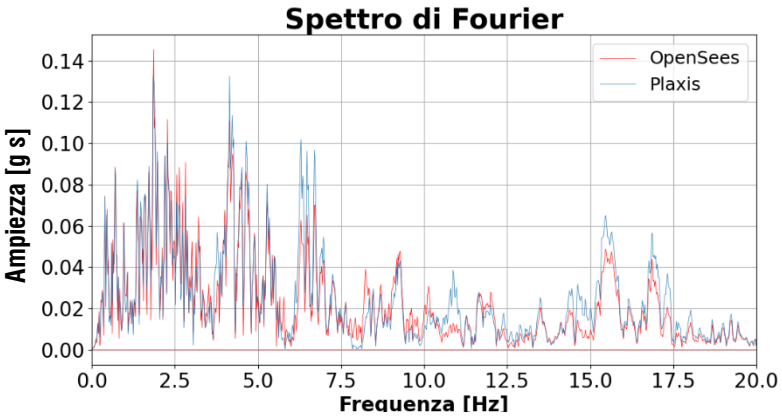
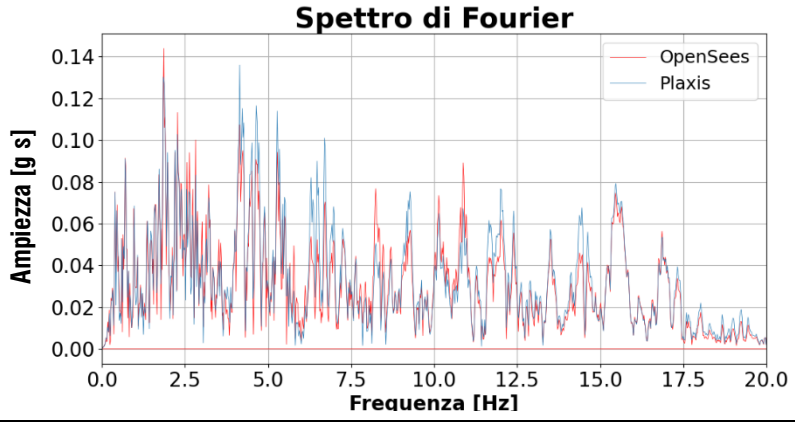
Y = 90 m	
Y = 105 m	
Y = 129 m	

Fig. 73 Sezione X = 870.0 m, accelerazioni orizzontali.

Punto	Spettro di Fourier
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

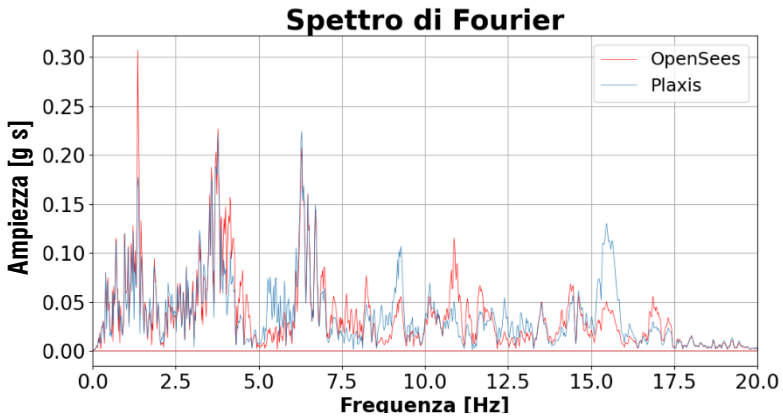
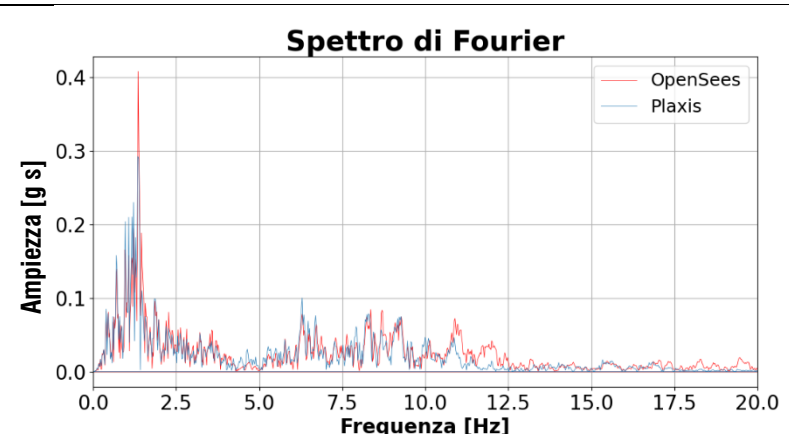
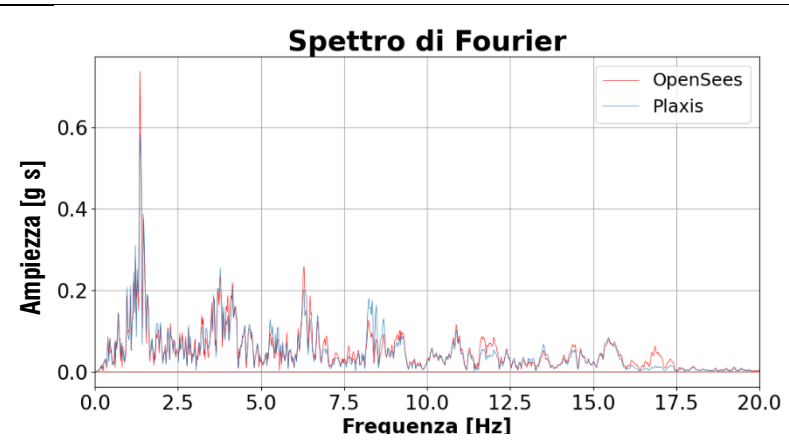
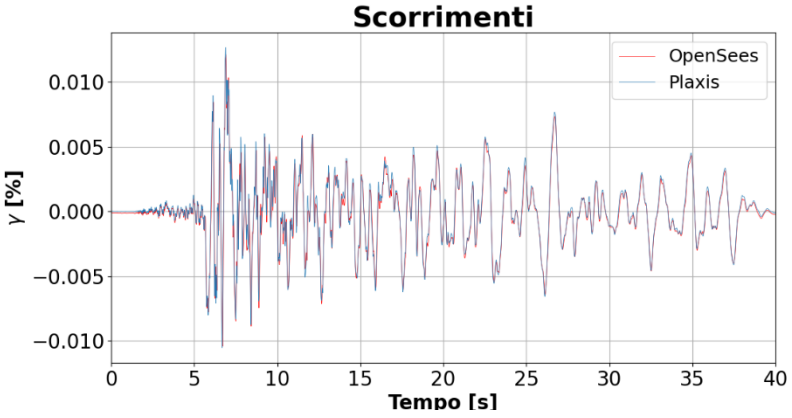
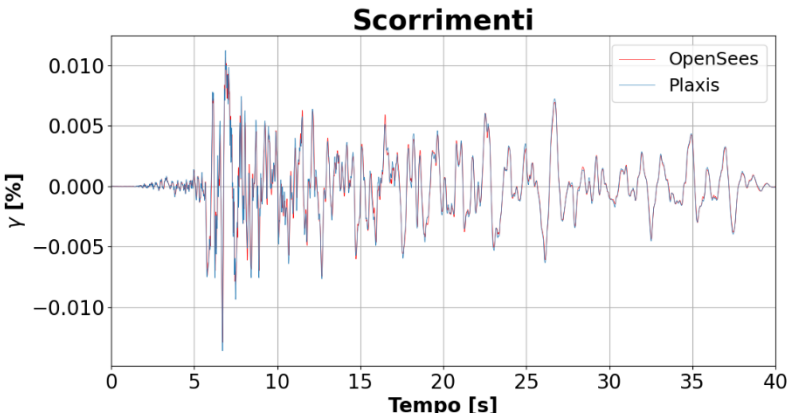
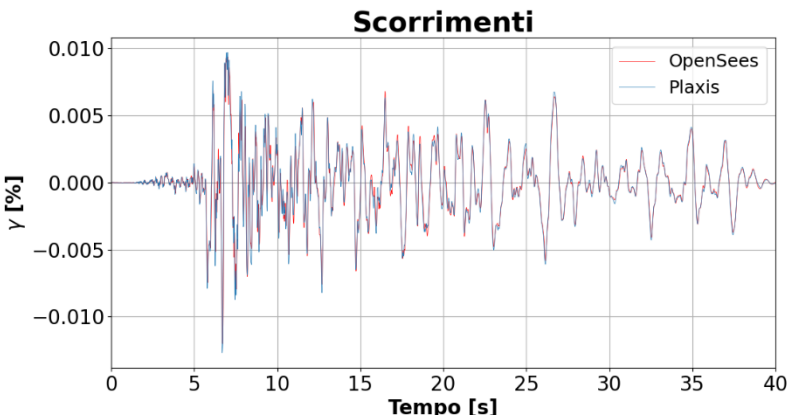
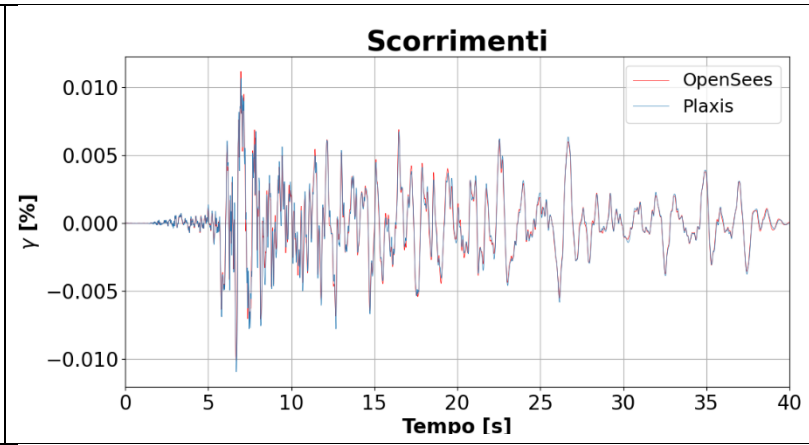
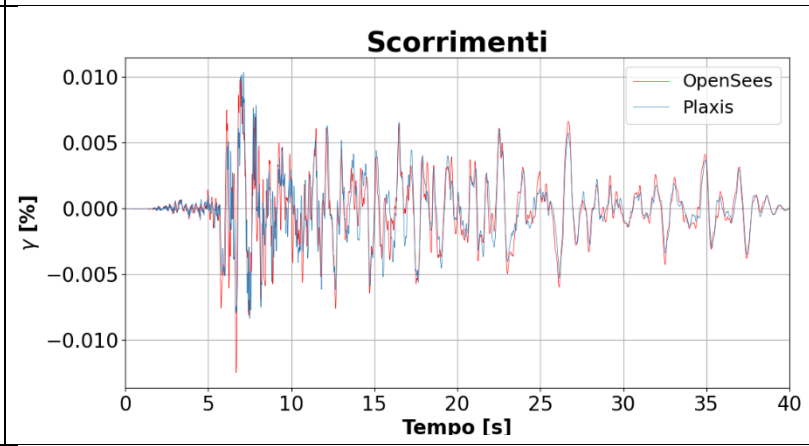
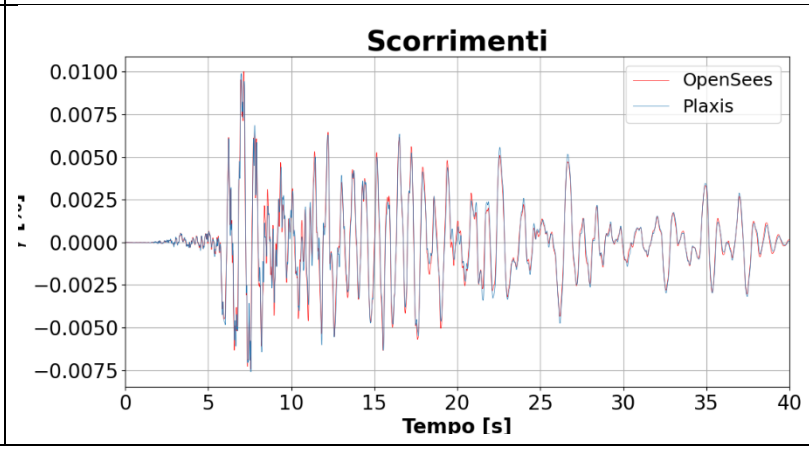
Y = 90 m	
Y = 105 m	
Y = 129 m	

Fig. 74 Sezione X = 870.0 m, spettri di Fourier.

Punto	Scorrimenti orizzontali
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

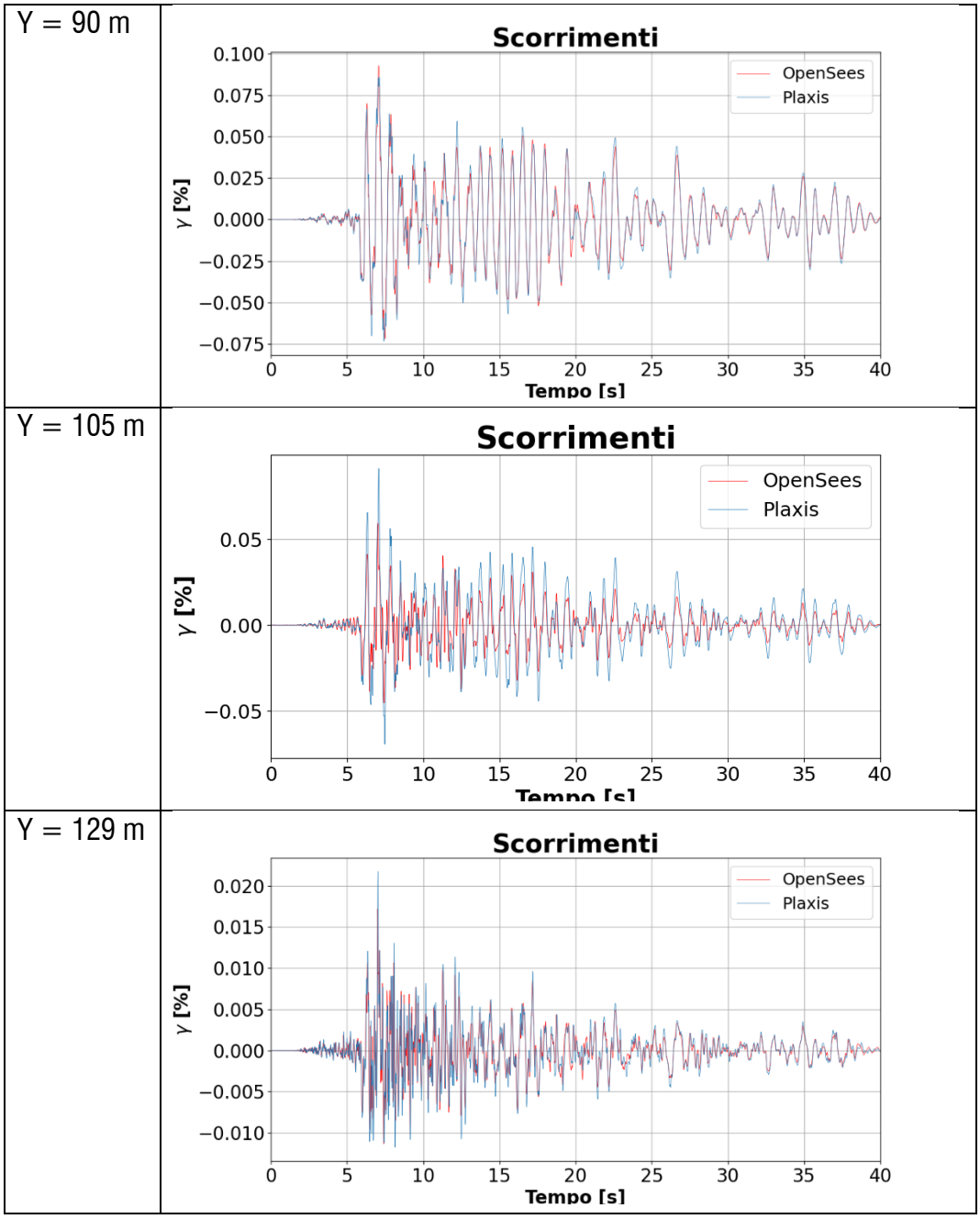
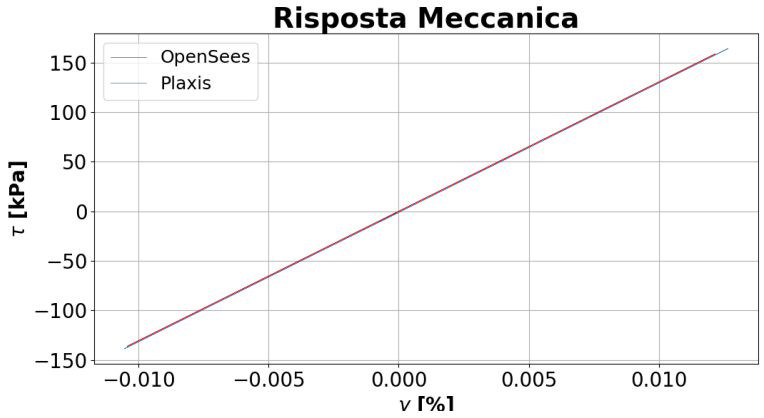
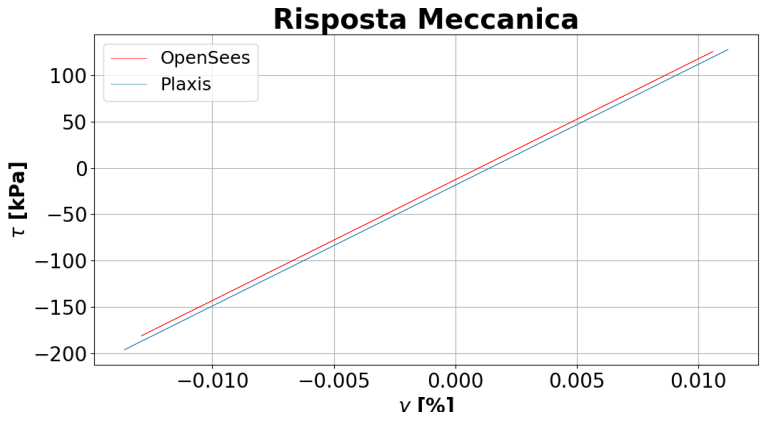
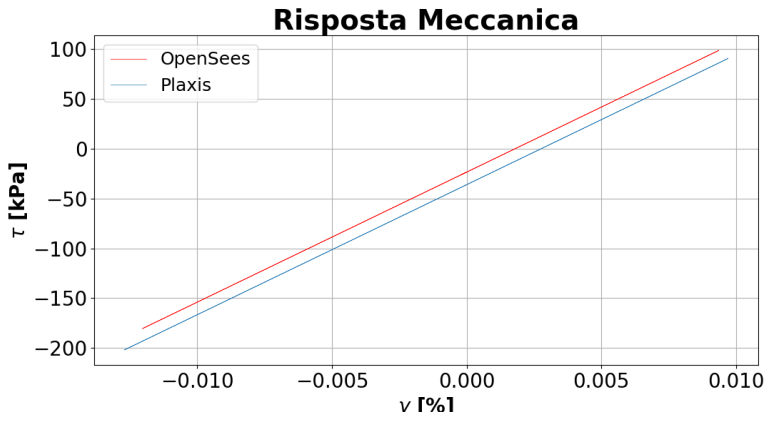
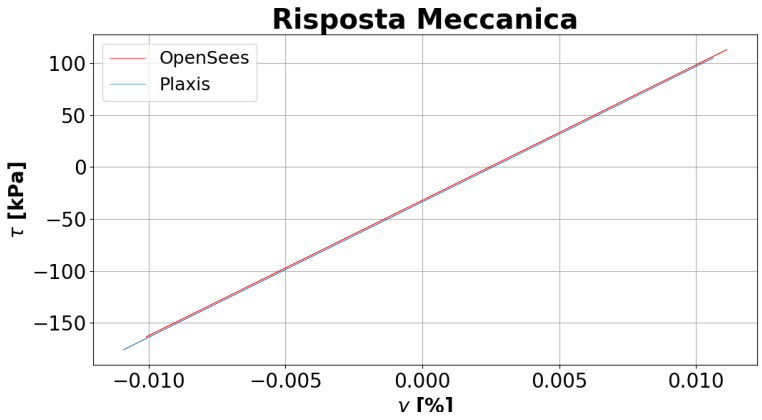
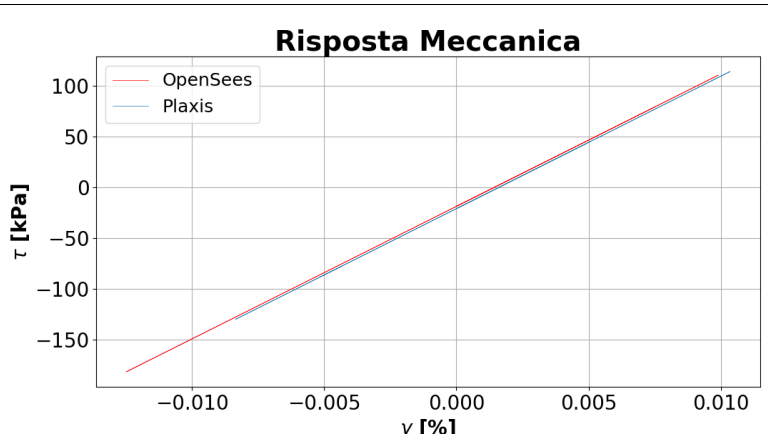
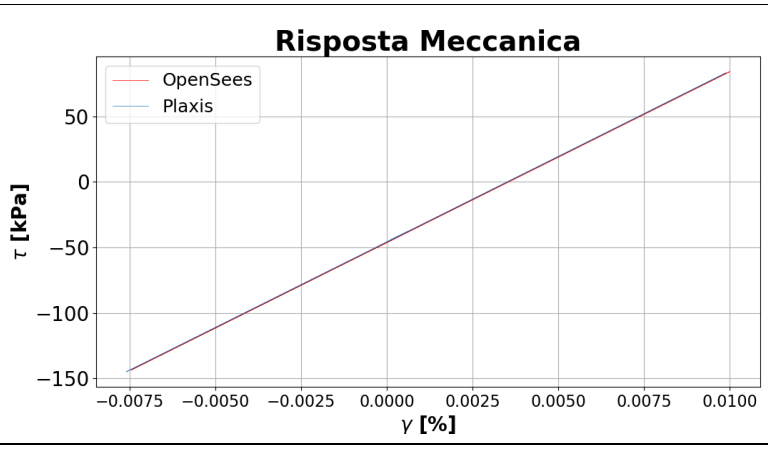


Fig. 75 Sezione X = 870.0 m, scorrimenti orizzontali.

Punto	Risposta meccanica
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

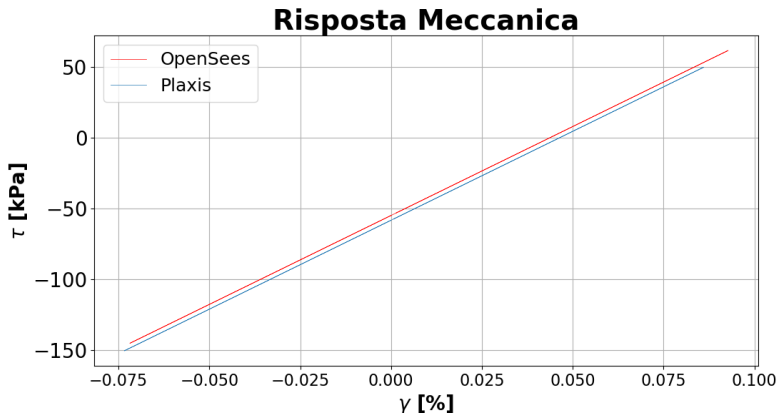
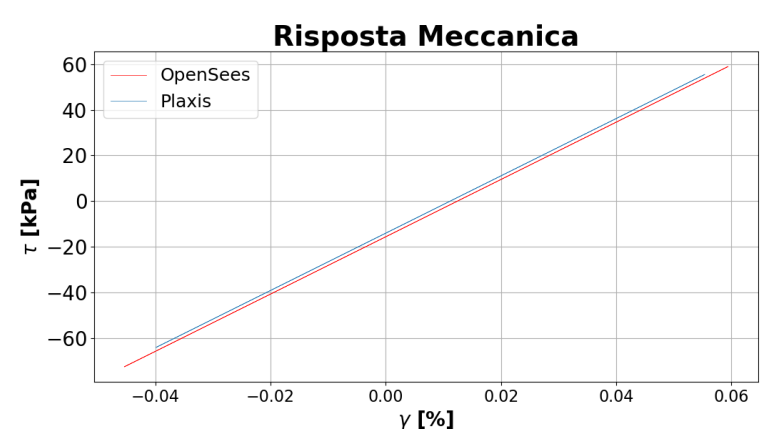
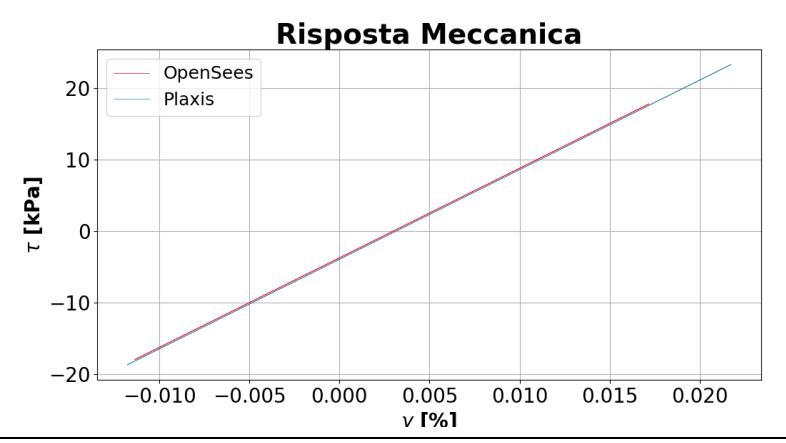
Y = 90 m	
Y = 105 m	
Y = 129 m	

Fig. 76 Sezione X = 870.0 m, risposta meccanica.

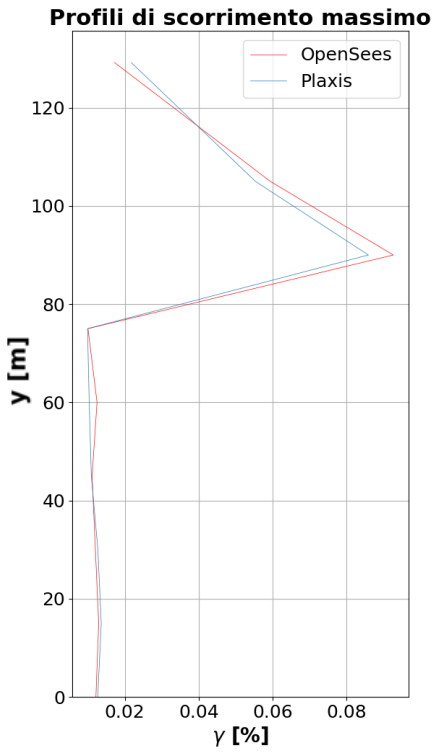
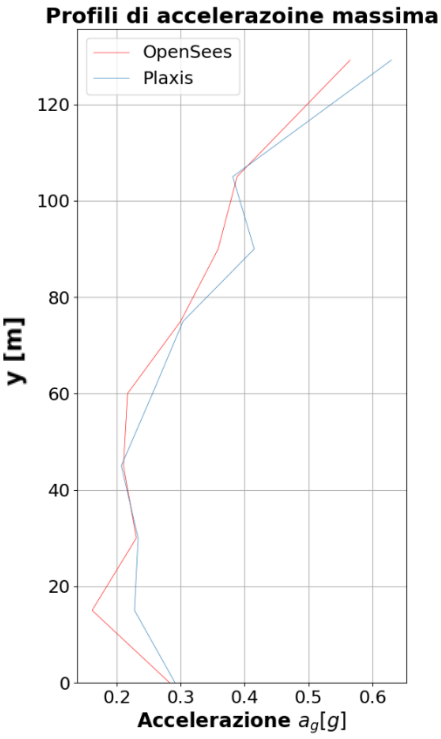
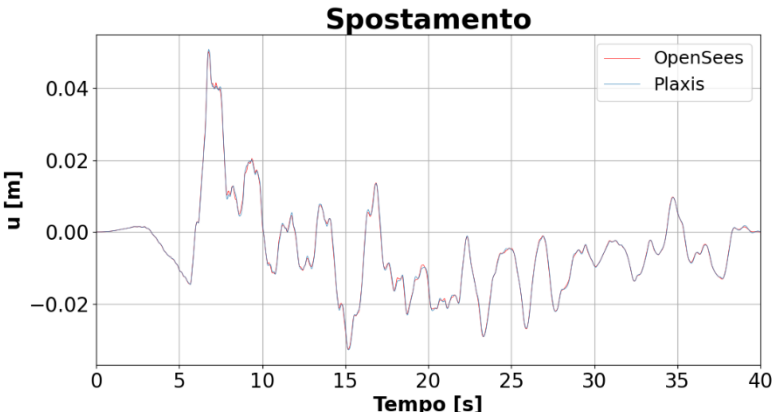
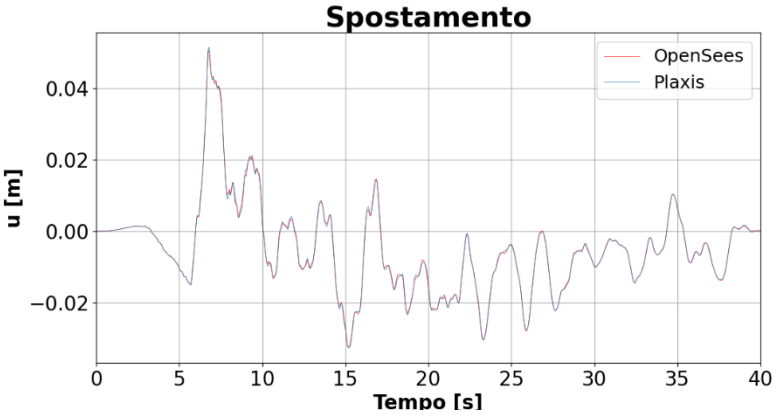
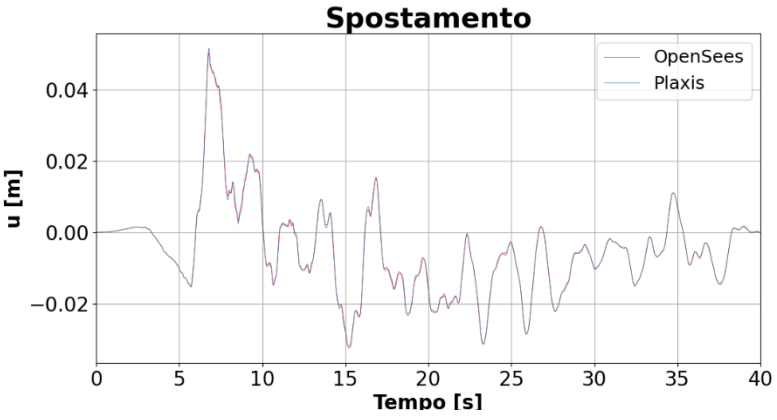
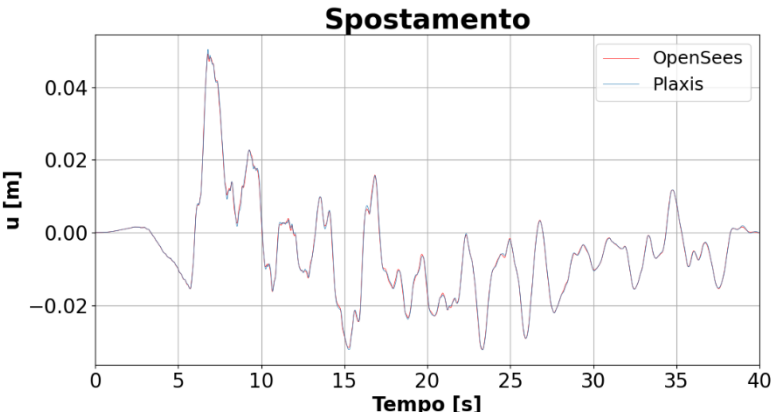
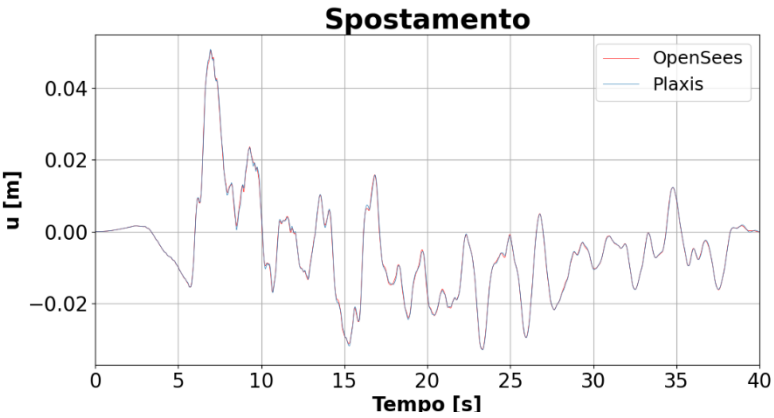
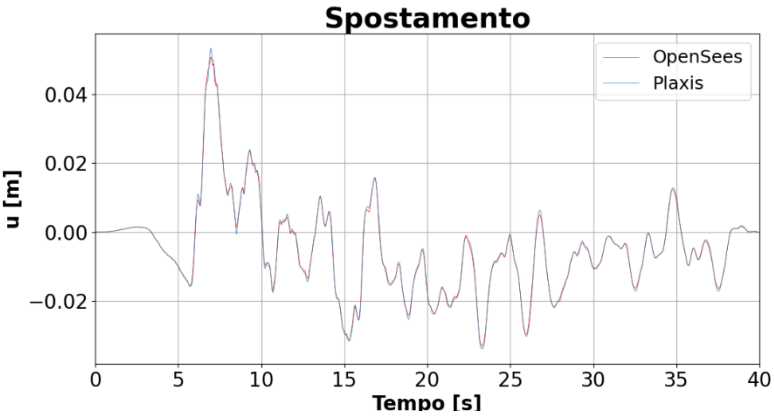
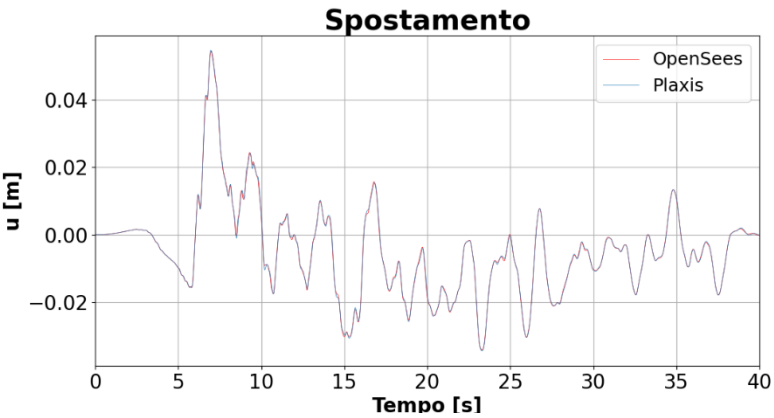
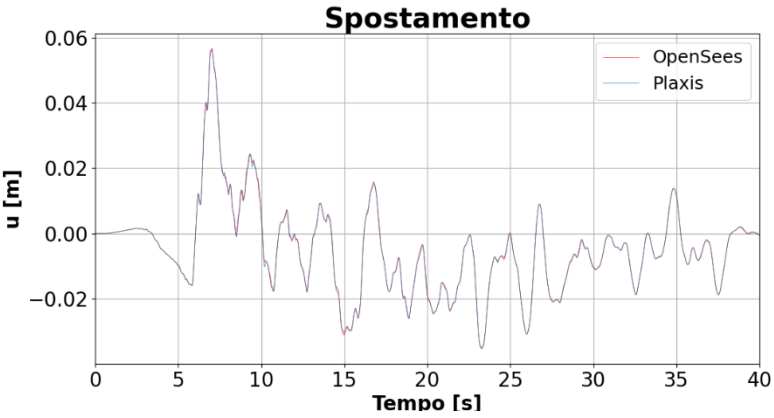
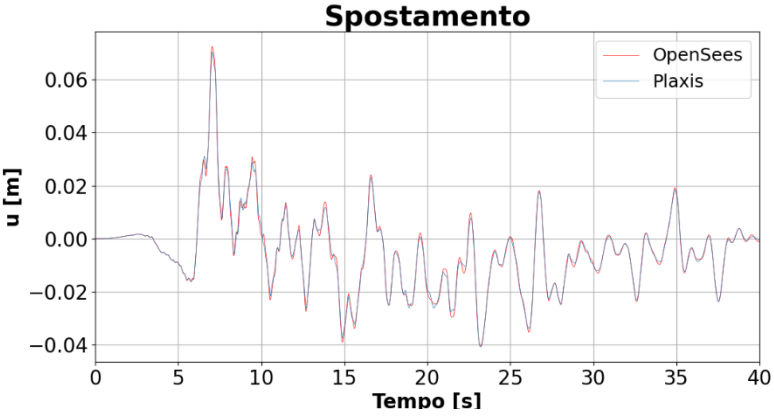
Profilo degli scorrimenti massimi	Profilo delle accelerazioni massime
	

Fig. 77 Sezione X = 870.0 m, profili con la profondità.

VERTICALE DI MONTE (X = 1250.0 m)

Punto	Spostamento Orizzontale
Y = 0 m	
Y = 20 m	
Y = 40 m	

Y = 60 m	
Y = 80 m	
Y = 100 m	

Y = 120 m	
Y = 140 m	
Y = 160 m	

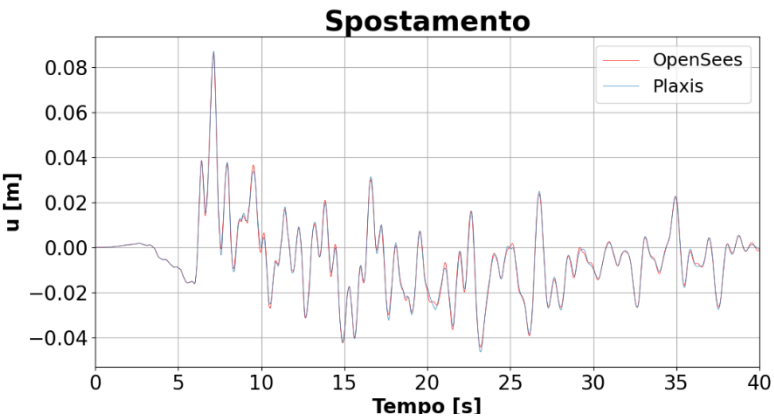
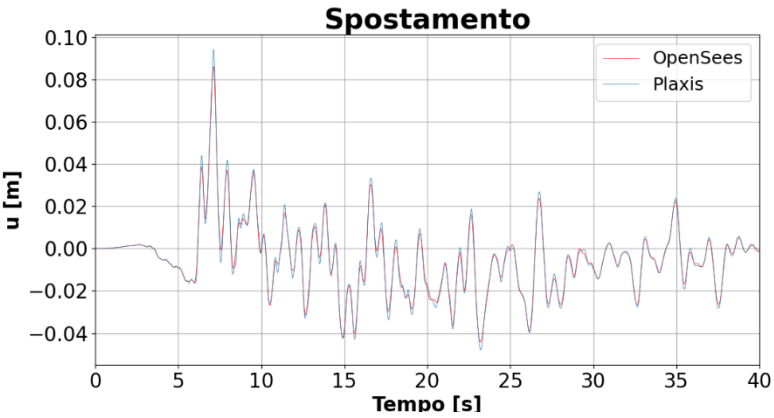
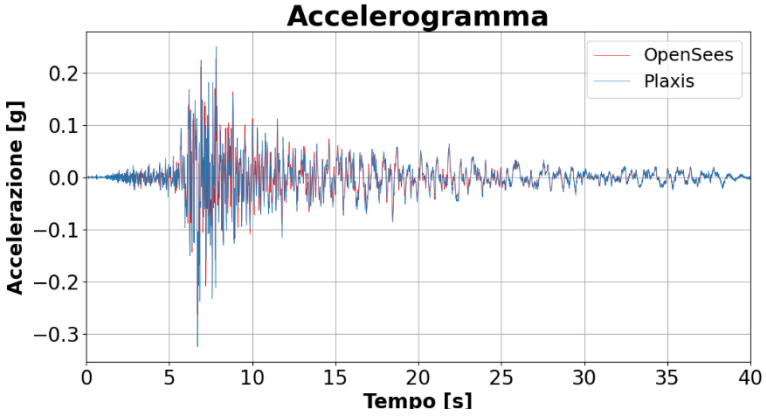
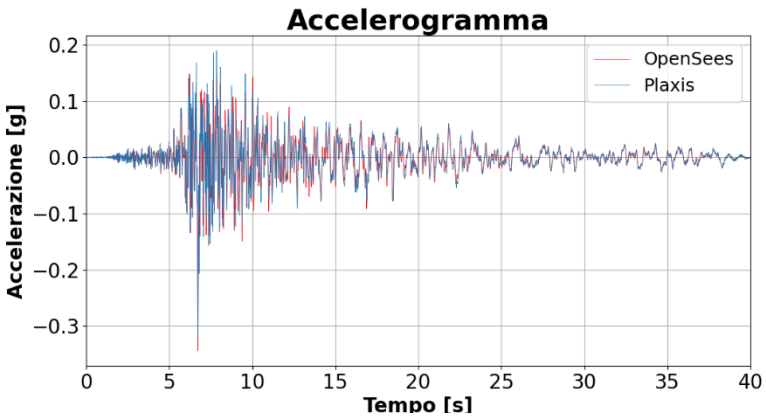
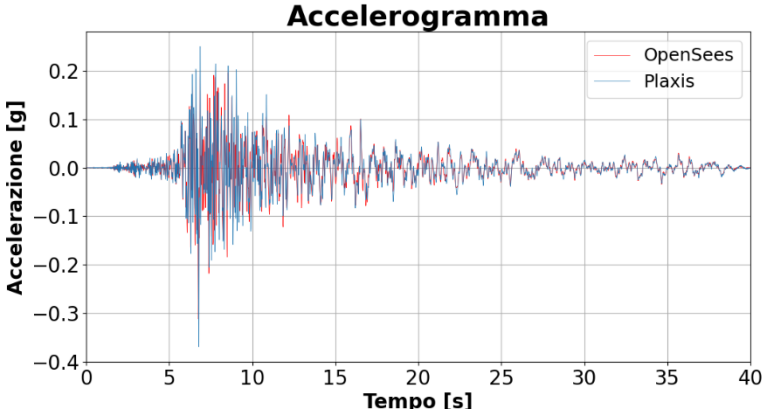
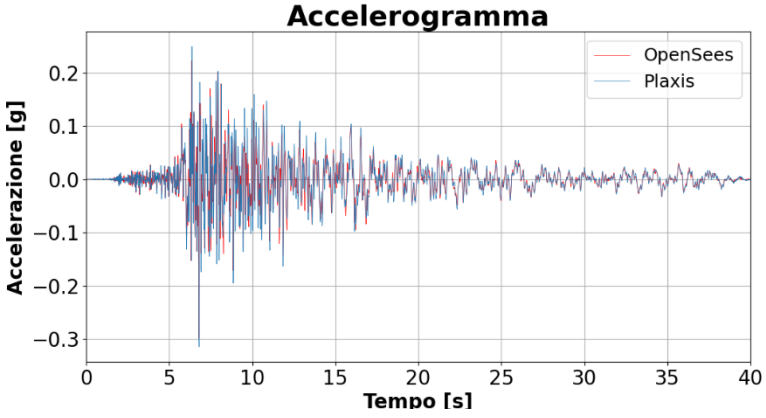
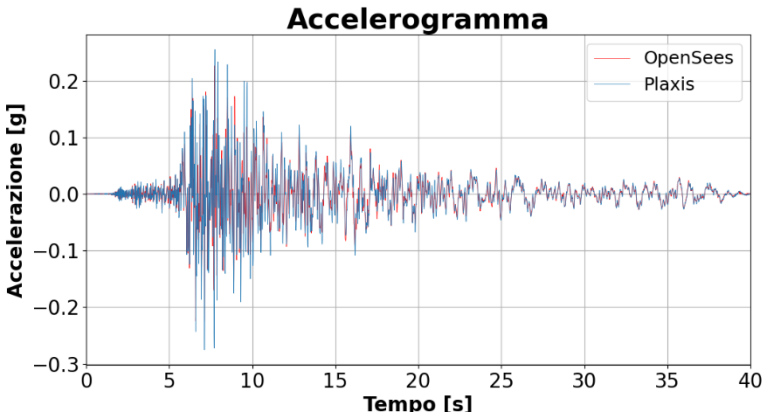
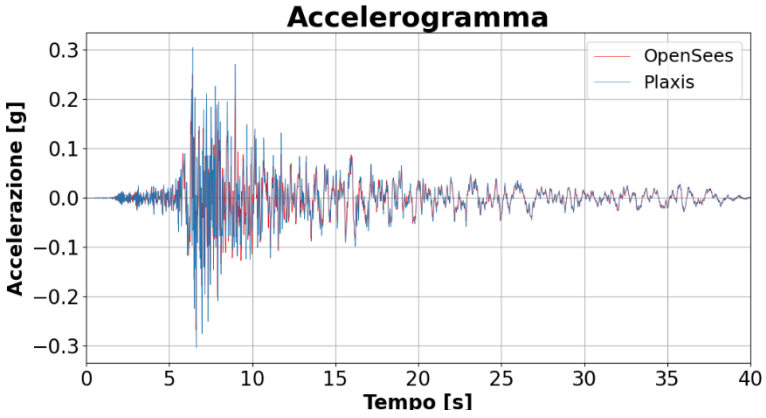
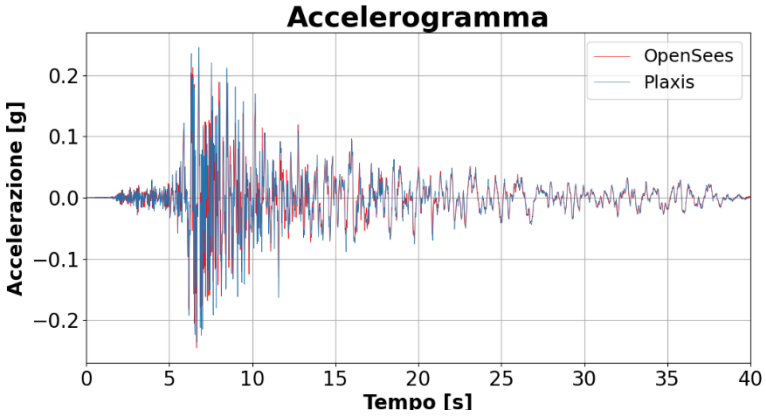
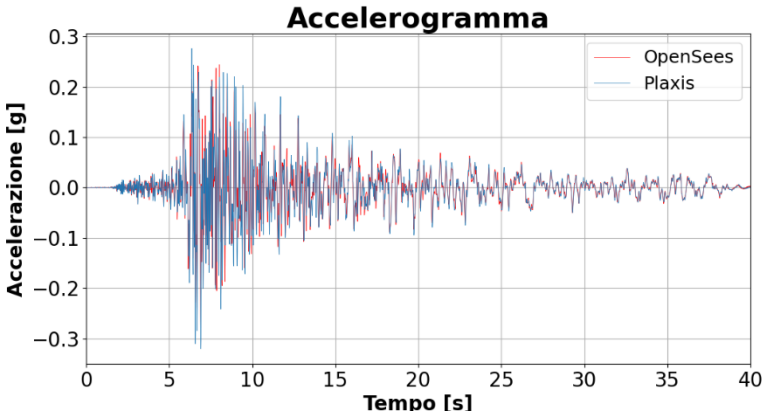
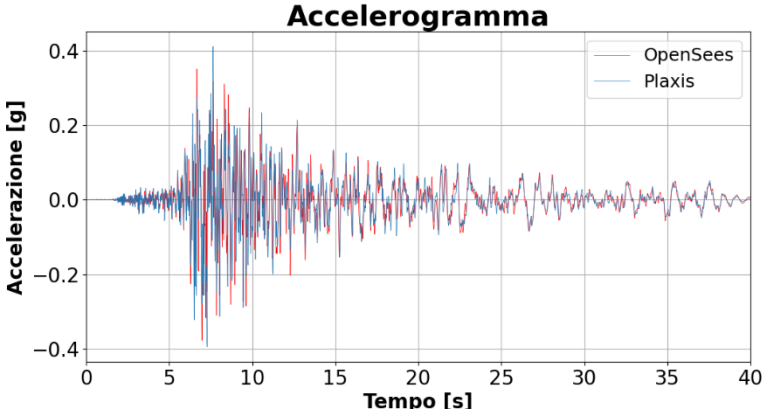
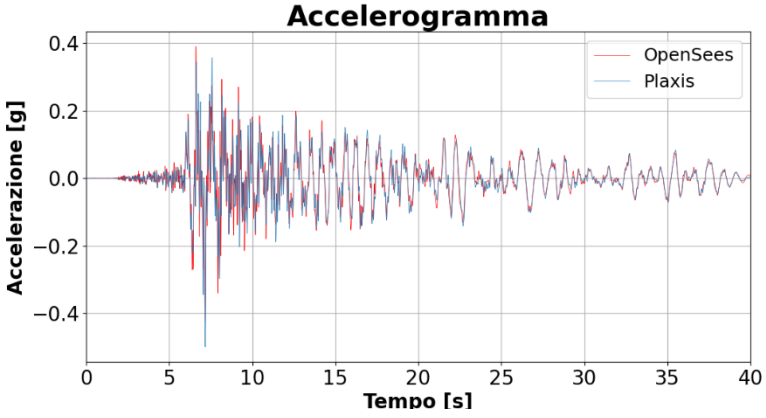
Y = 180 m	
Y = 195 m	

Fig. 78 Sezione X = 1250.0 m, spostamenti orizzontali.

Punto	Accelerazione Orizzontale
Y = 0 m	

Y = 20 m	
Y = 40 m	
Y = 60 m	

Y = 80 m	
Y = 100 m	
Y = 120 m	

Y = 140 m	
Y = 160 m	
Y = 180 m	

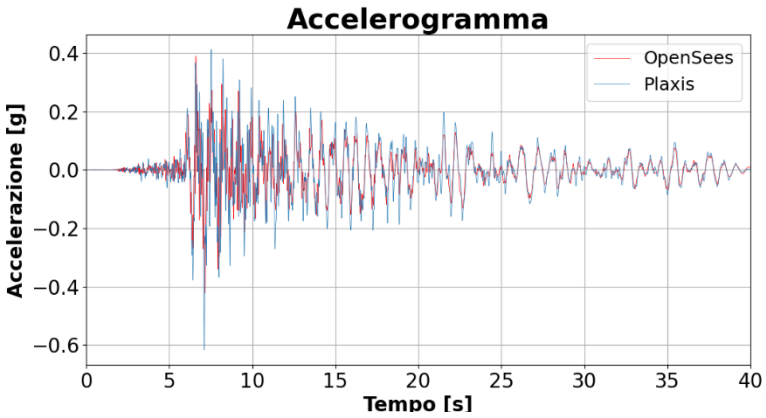
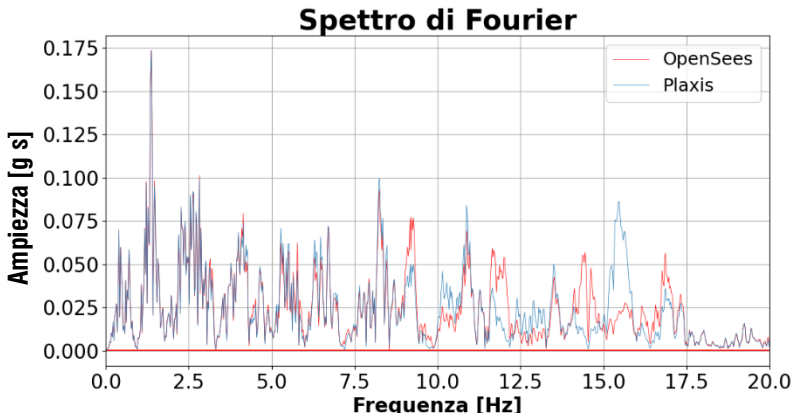
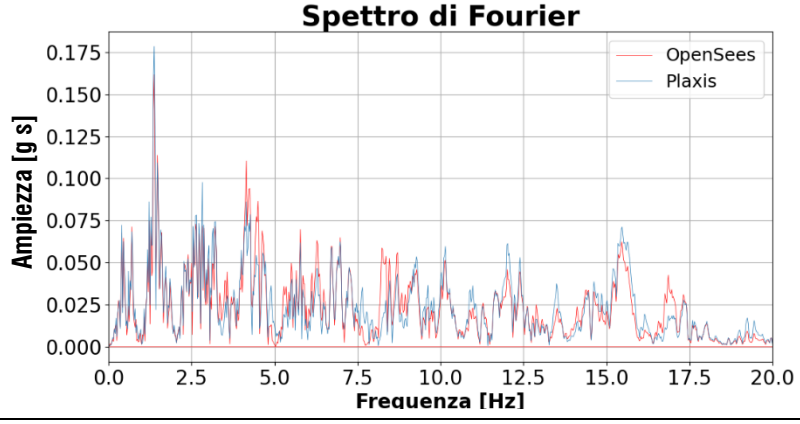
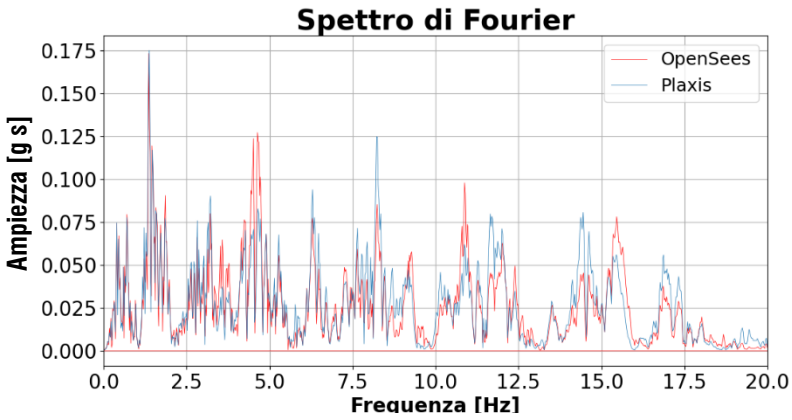
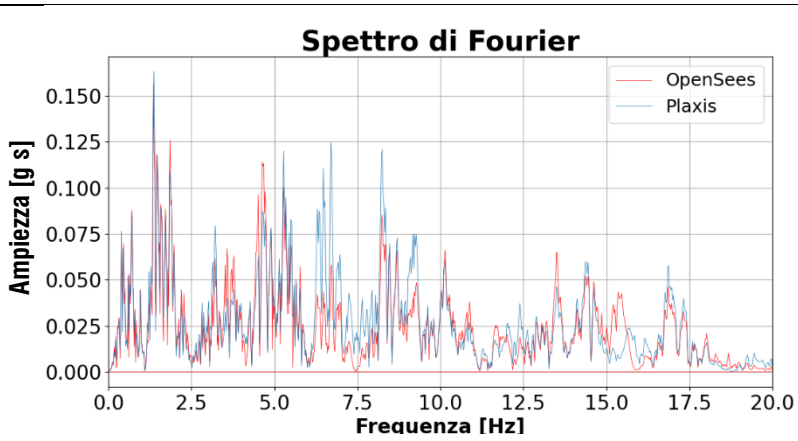
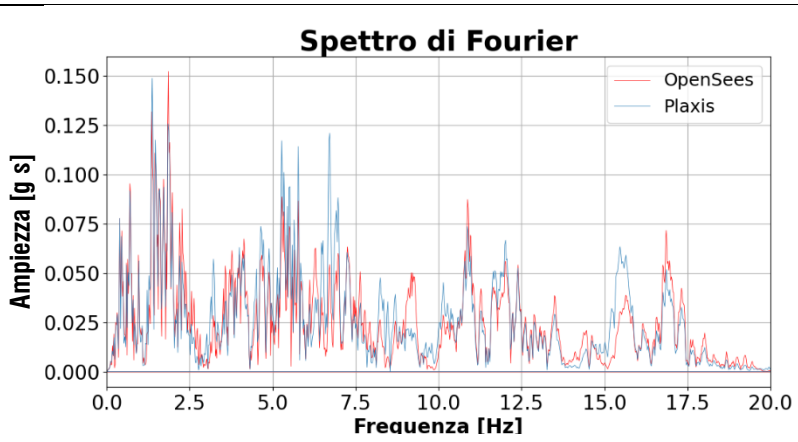
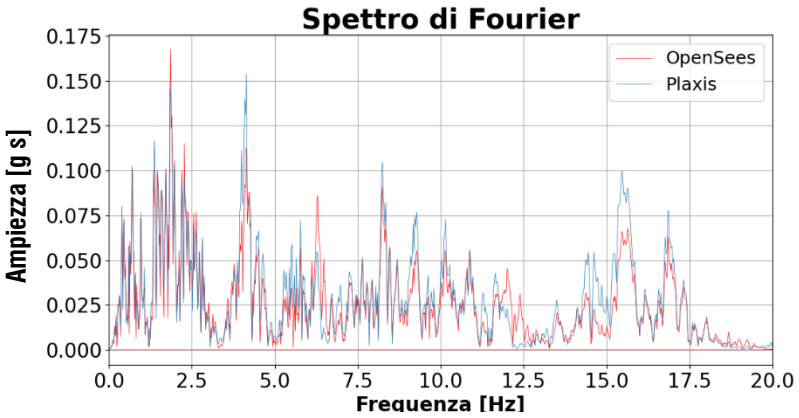
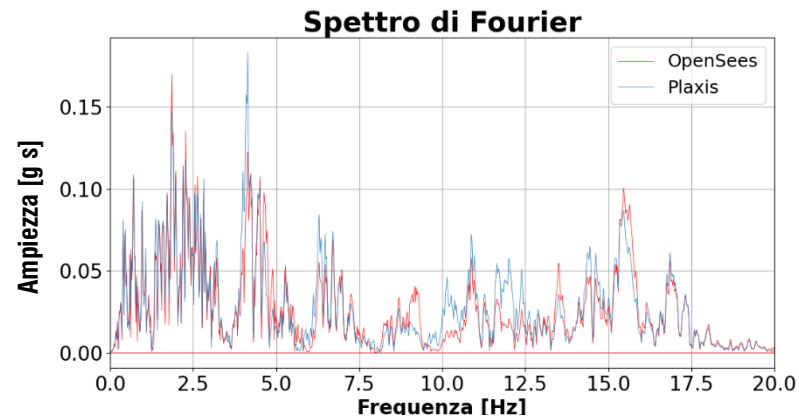
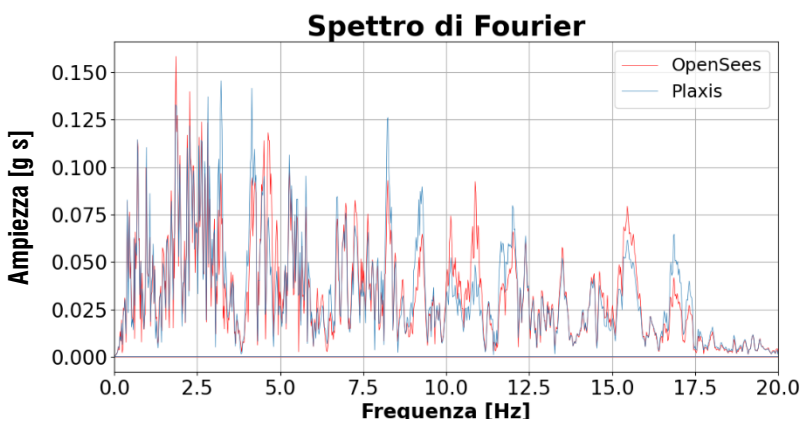
Y = 195 m	

Fig. 79 Sezione X = 1250.0 m, accelerazioni orizzontali.

Punto	Spettro di Fourier
Y = 0 m	
Y = 20 m	

Y = 40 m	
Y = 60 m	
Y = 80 m	

Y = 100 m	
Y = 120 m	
Y = 140 m	

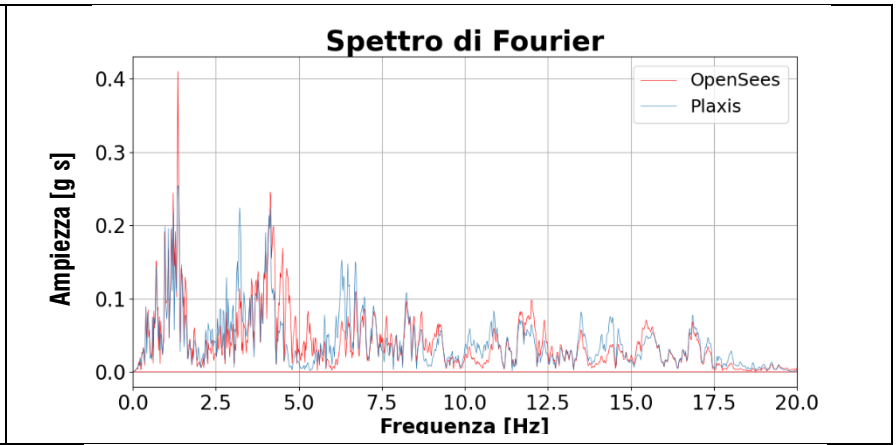
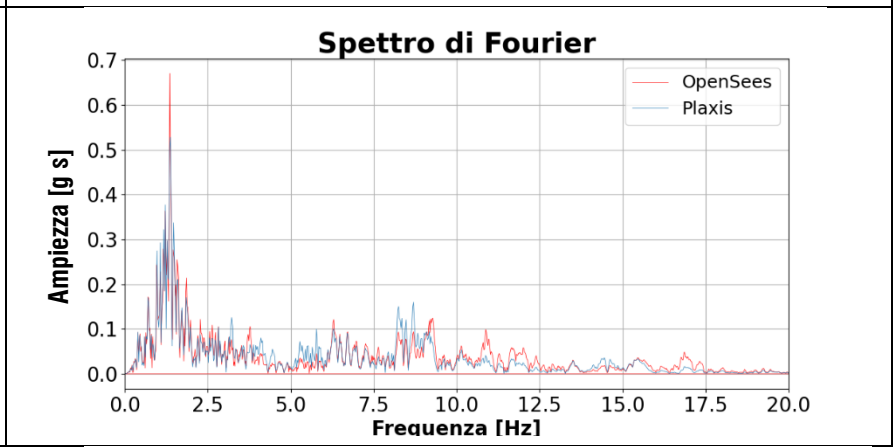
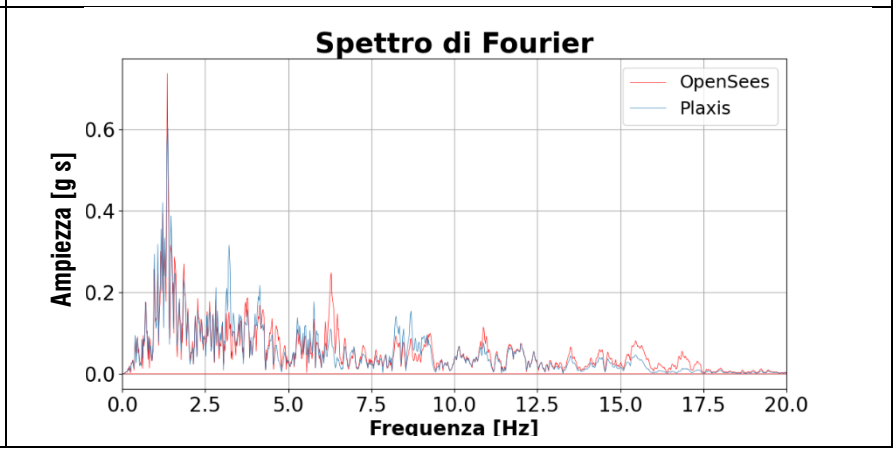
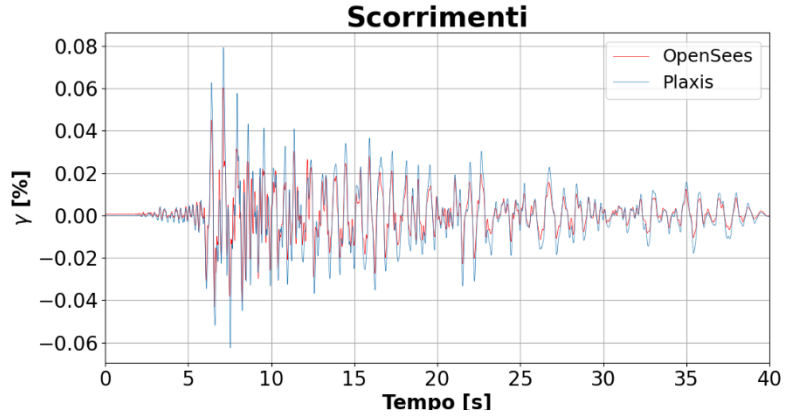
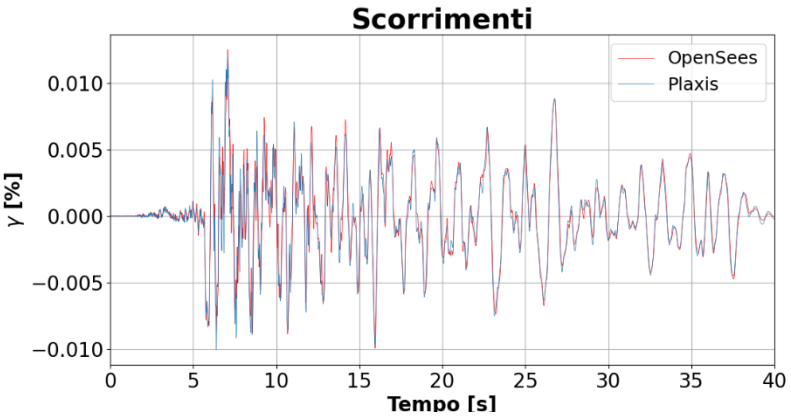
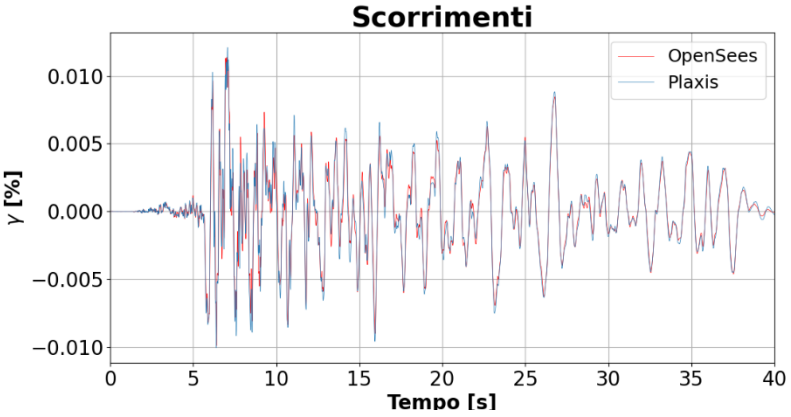
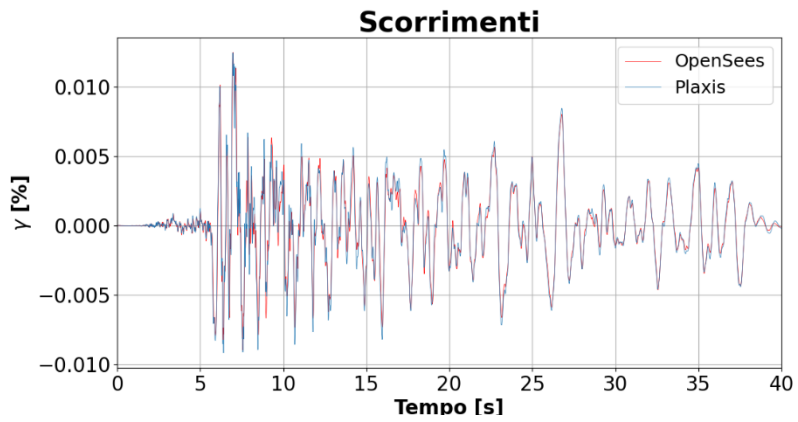
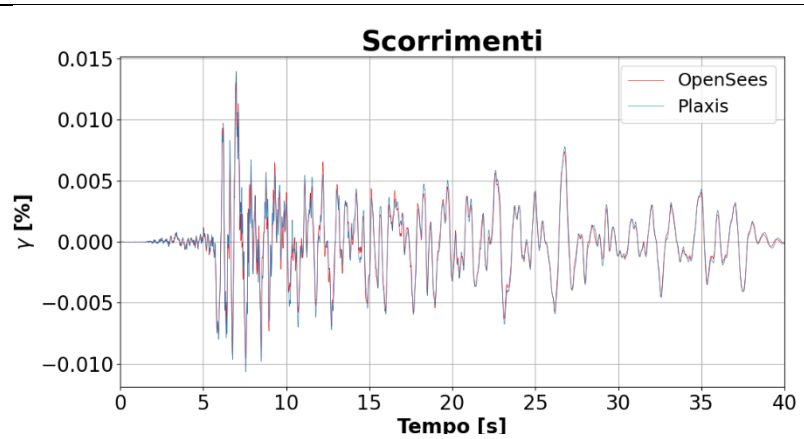
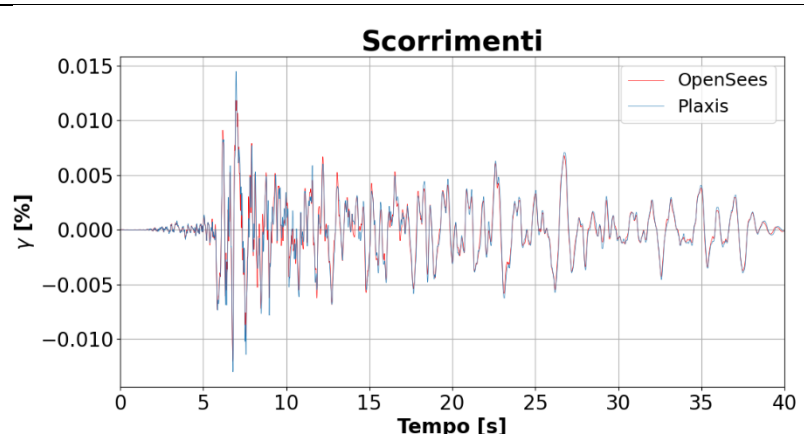
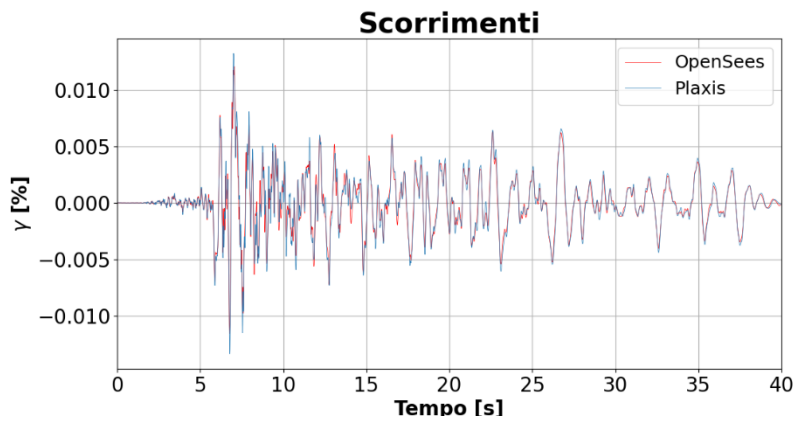
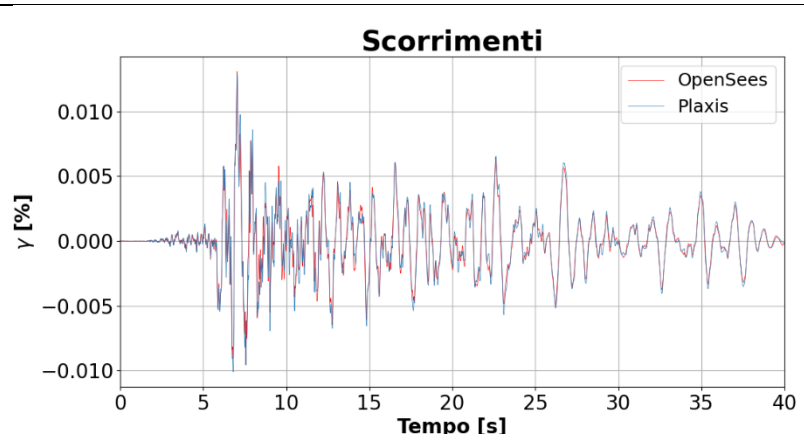
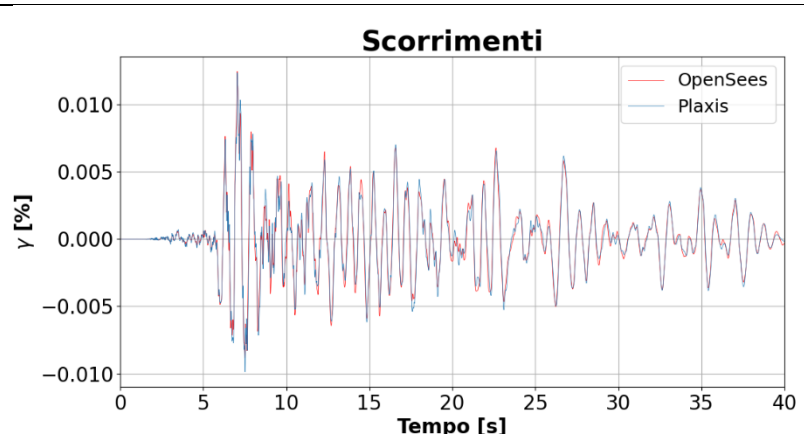
Y = 160 m	
Y = 180 m	
Y = 195 m	

Fig. 80 Sezione X = 1250.0 m, spettri di Fourier.

Punto	Scorrimenti orizzontali
Y = 0 m	
Y = 20 m	
Y = 40 m	

Y = 60 m	
Y = 80 m	
Y = 100 m	

Y = 120 m	
Y = 140 m	
Y = 160 m	

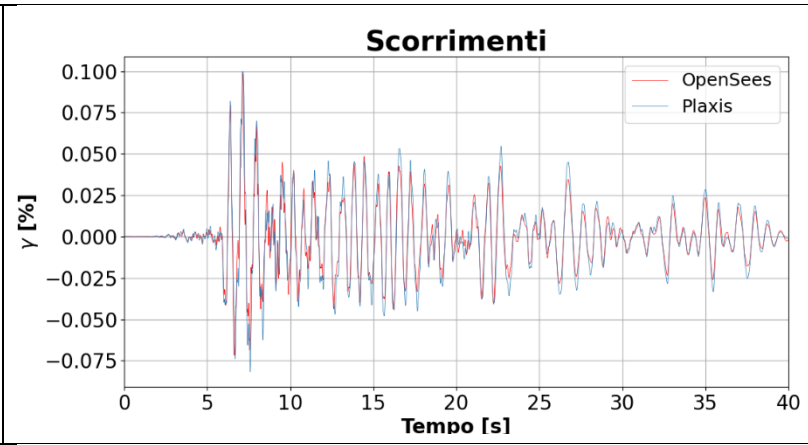
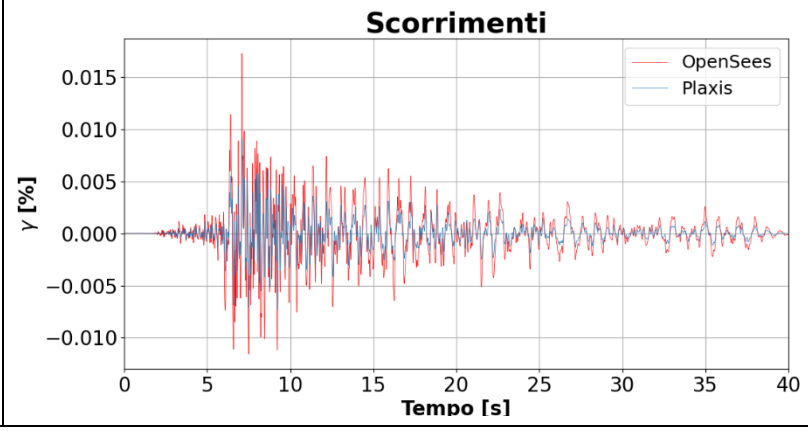
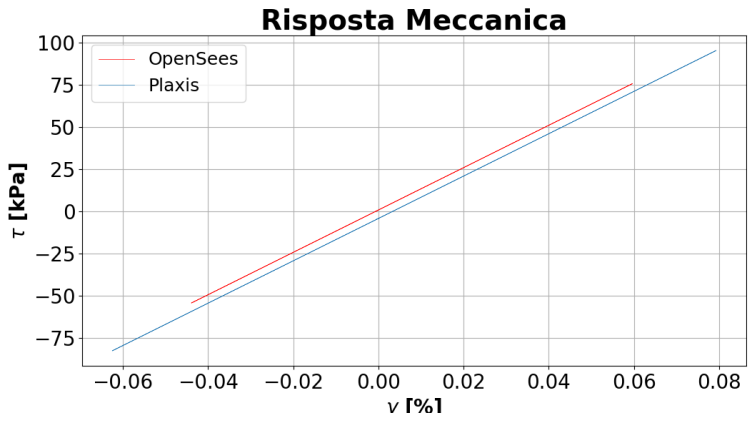
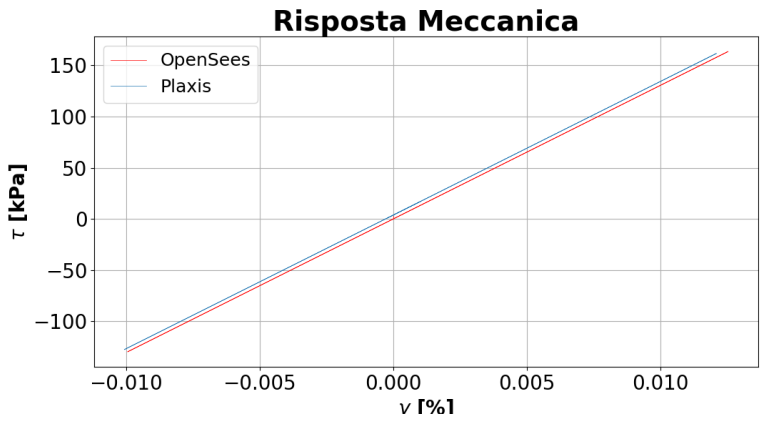
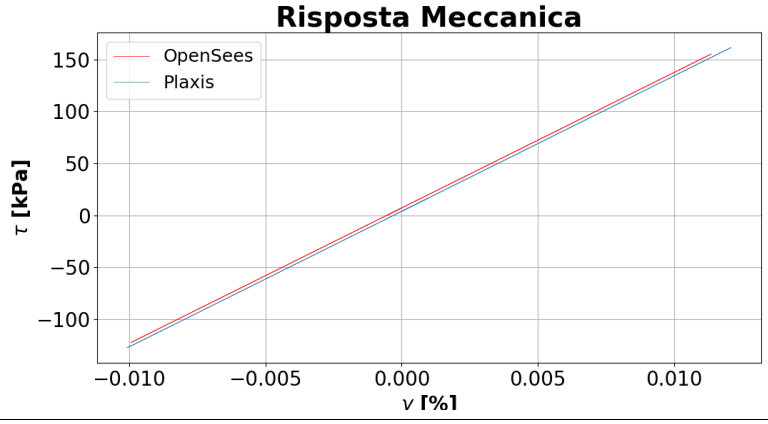
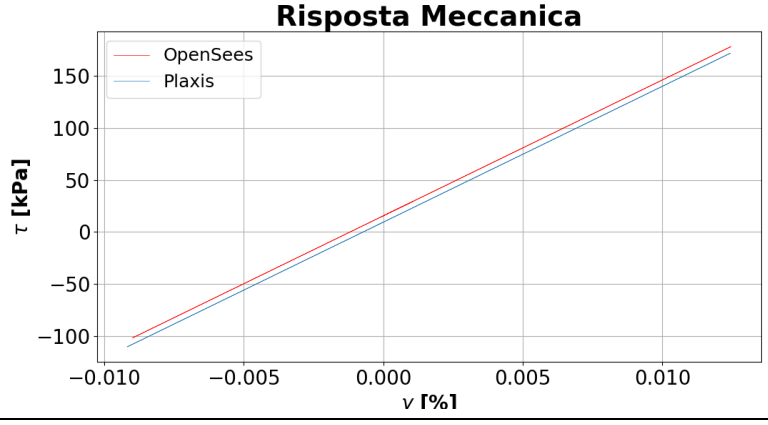
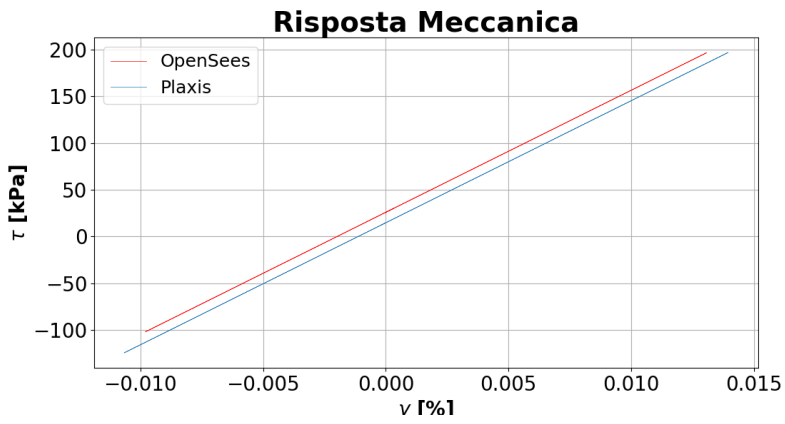
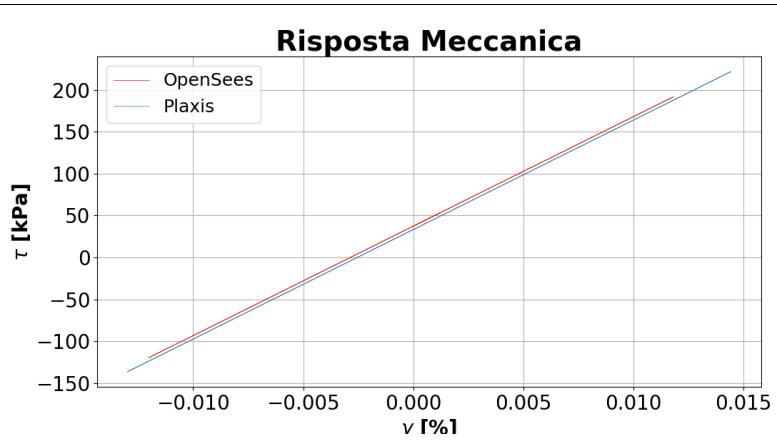
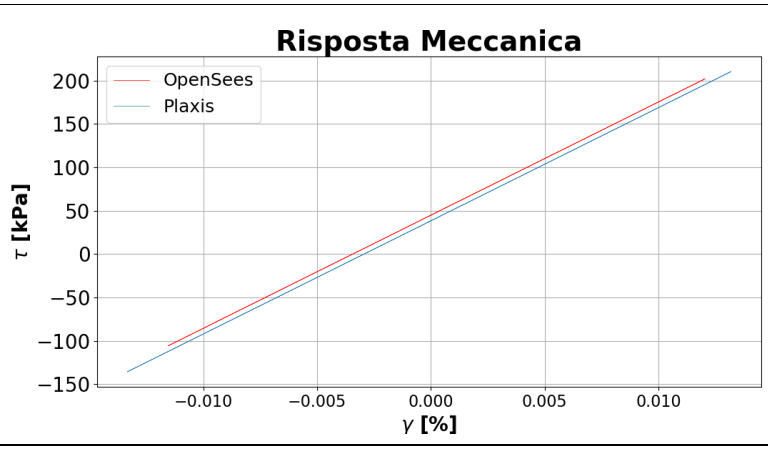
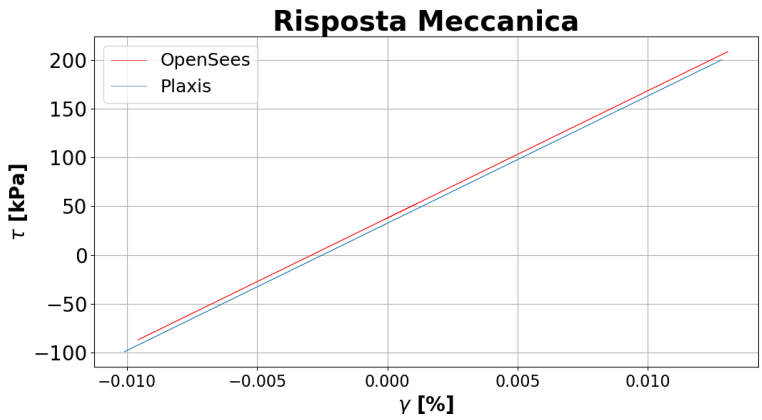
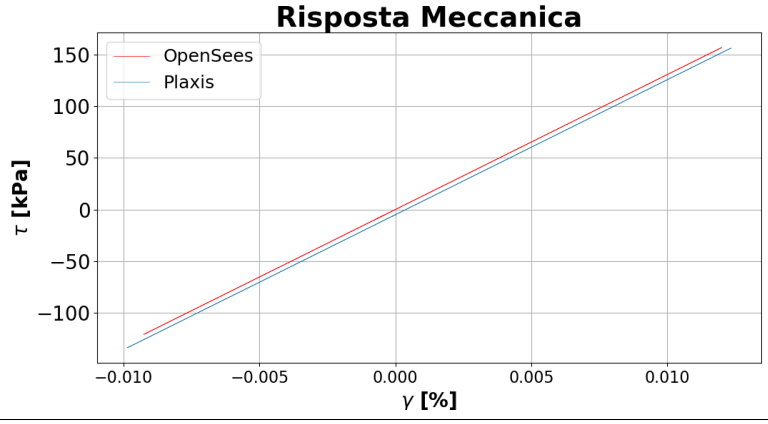
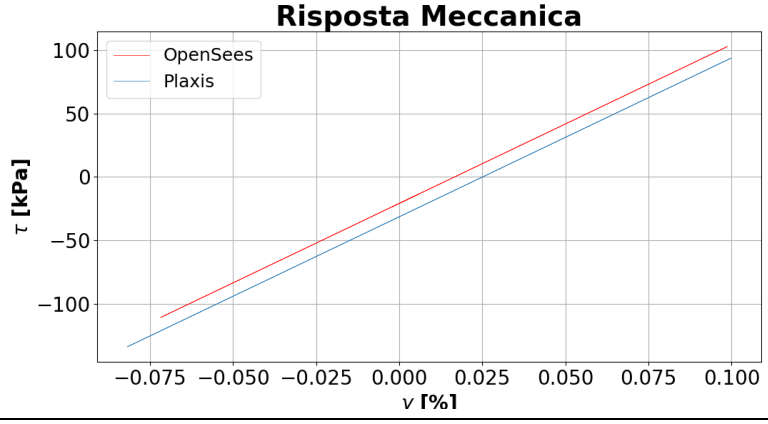
Y = 180 m	
Y = 195 m	

Fig. 81 Sezione X = 1250.0 m, scorrimenti orizzontali.

Punto	Risposta meccanica
Y = 0 m	

Y = 20 m	
Y = 40 m	
Y = 60 m	

Y = 80 m	
Y = 100 m	
Y = 120 m	

Y = 140 m	
Y = 160 m	
Y = 180 m	

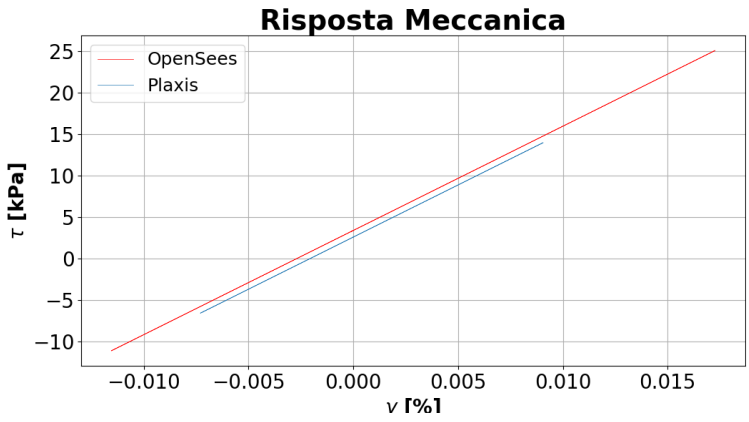
Y = 195 m	

Fig. 82 Sezione X = 1250.0 m, risposta meccanica.

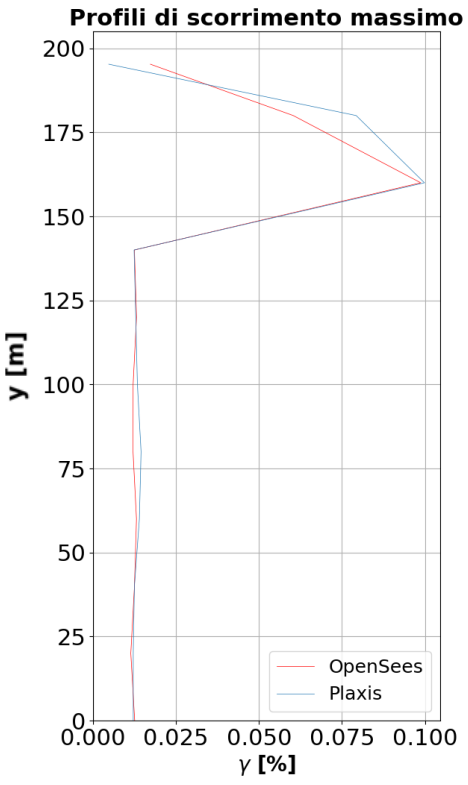
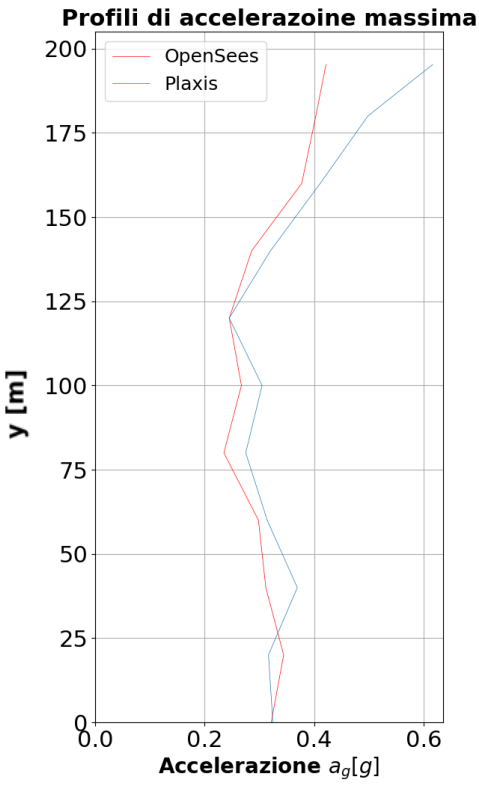
Profilo degli scorrimenti massimi	Profilo delle accelerazioni massime
	

Fig. 83 Sezione X = 1250.0 m, profili con la profondità.

PROFILI ACCELERAZIONI MASSIME IN SUPERFICIE

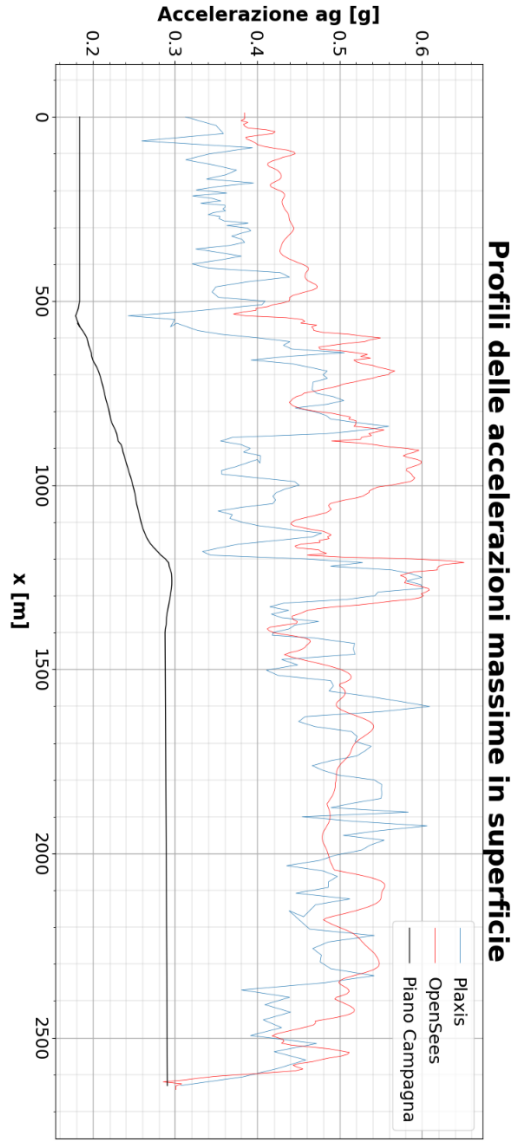
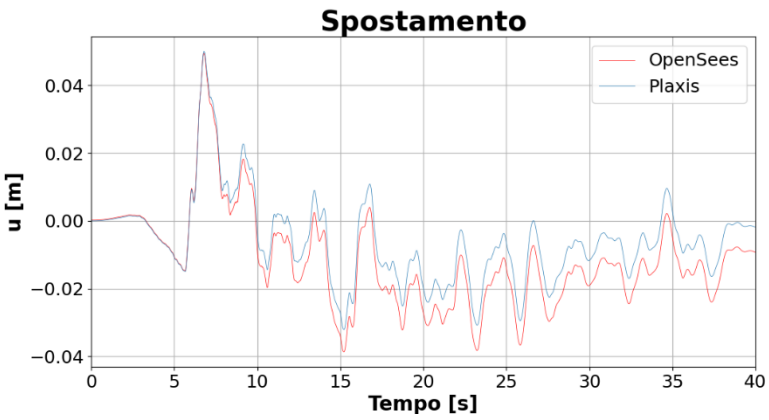
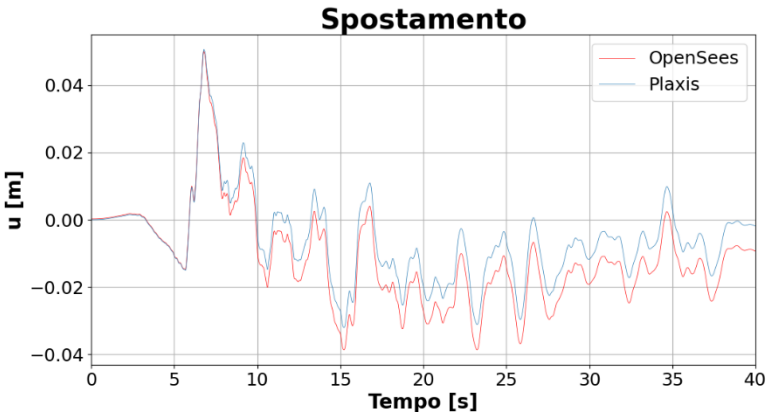


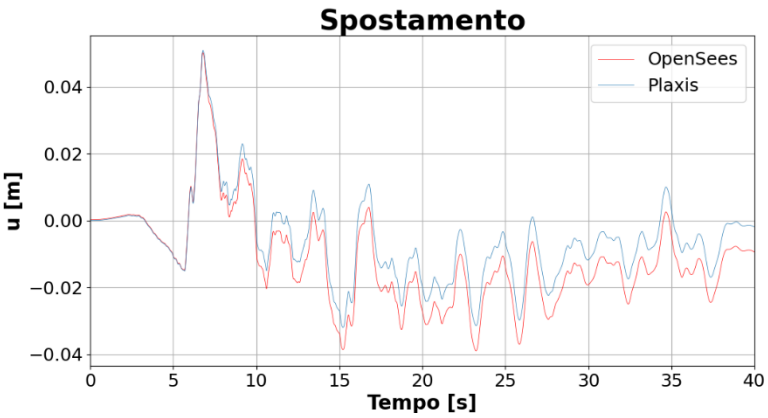
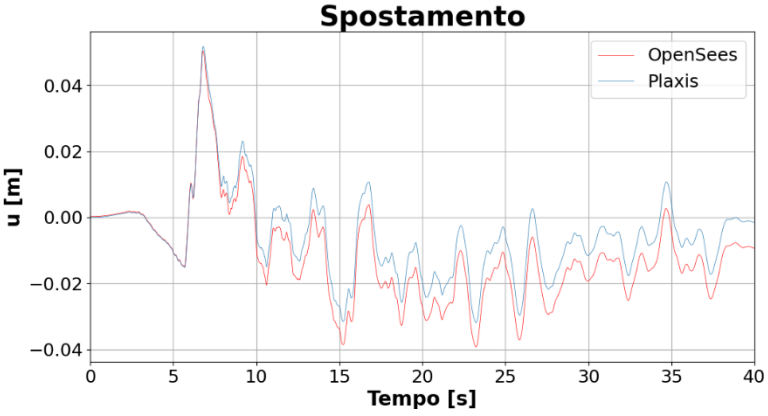
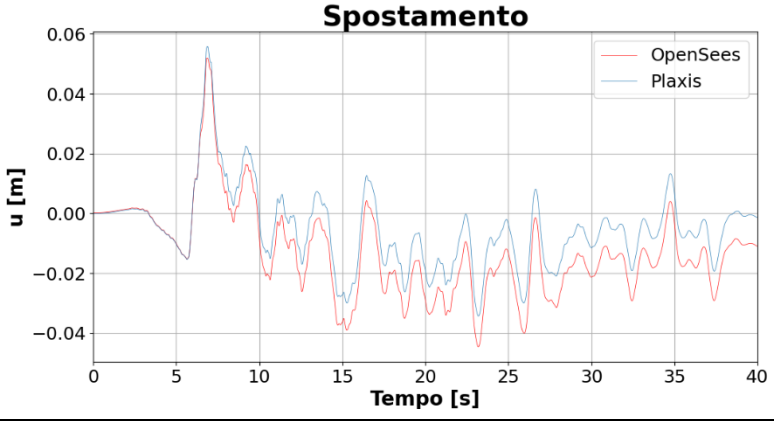
Fig. 84 Profilo longitudinale delle accelerazioni massime in superficie ottenuti dalle analisi visco-elastiche.

APPENDICE 5

RISULTATI ANALISI VISCO-ELASTO-PLASTICHE

VERTICALE VALLE (X = 540.0 m)

Punto	Spostamento Orizzontale
Y = 0 m	
Y = 10 m	

Y = 20 m	
Y = 30 m	
Y = 40 m	

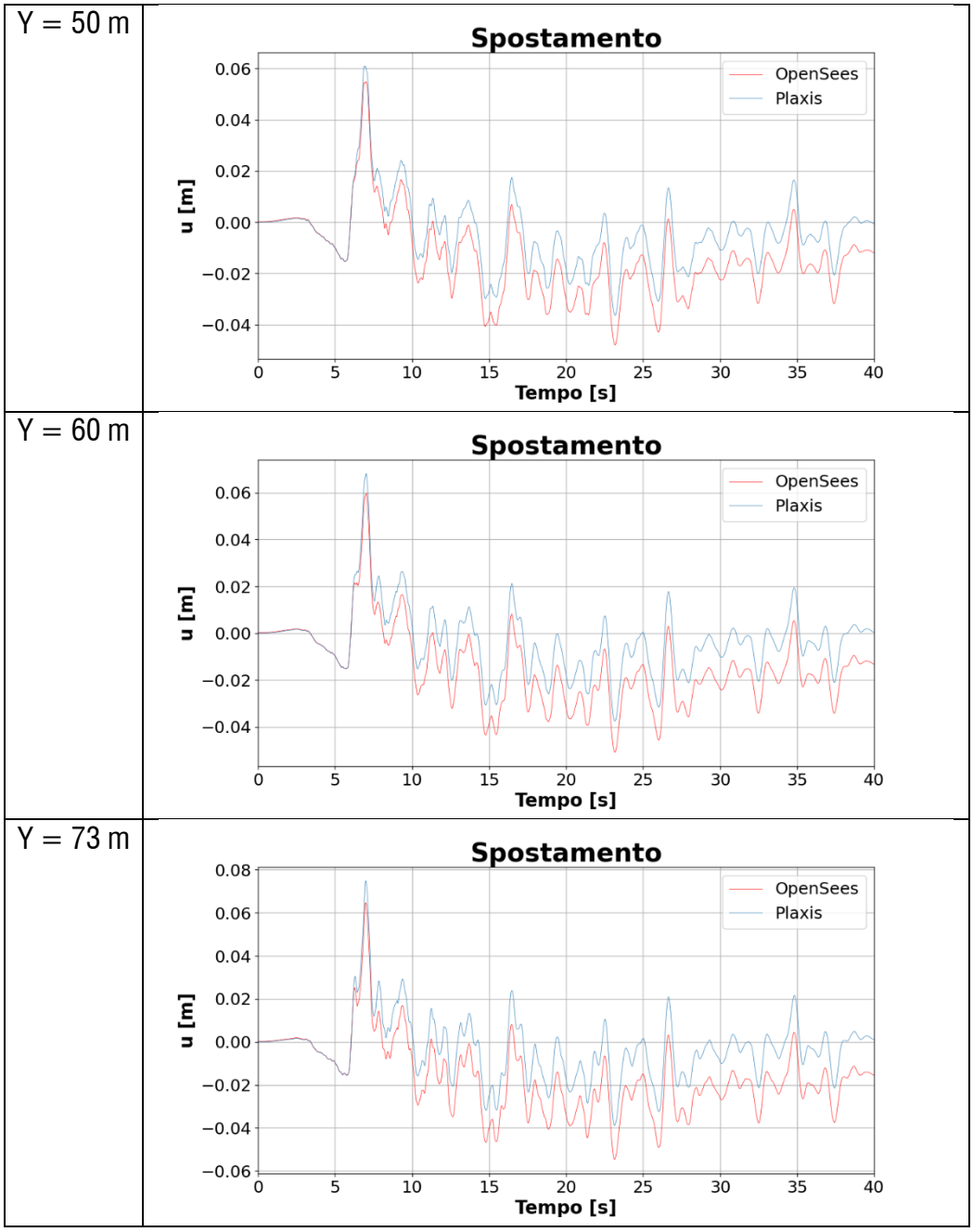
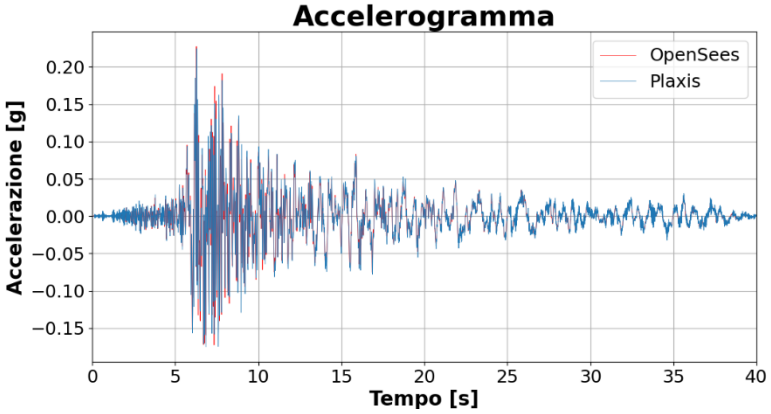
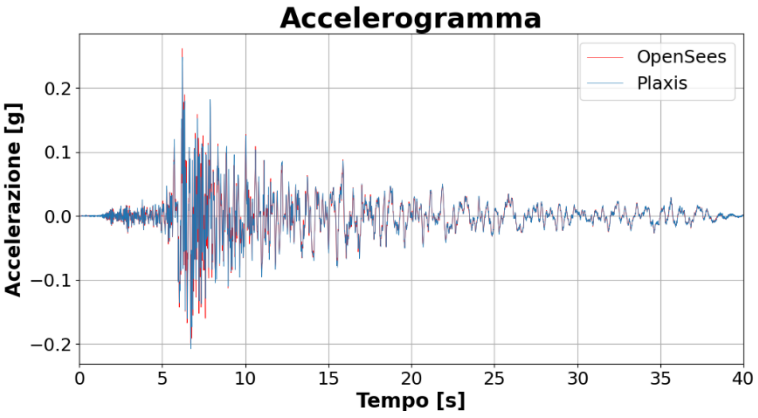
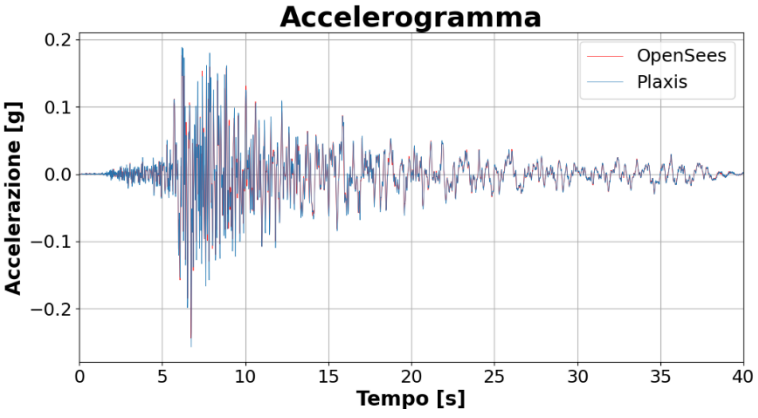
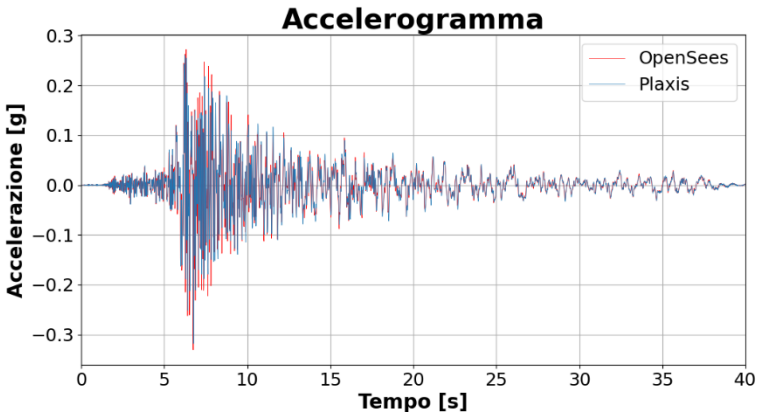
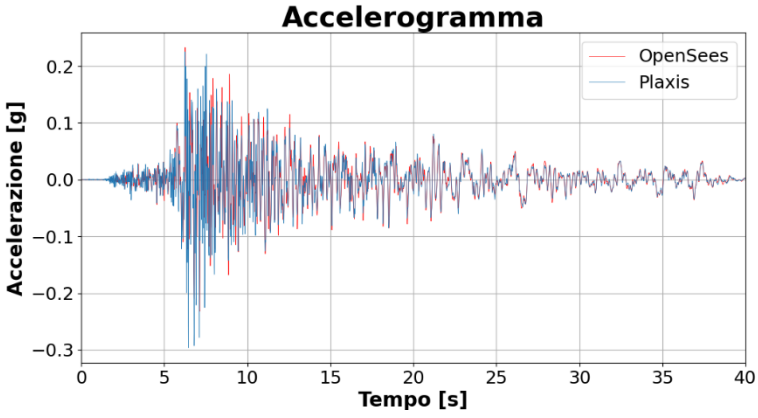
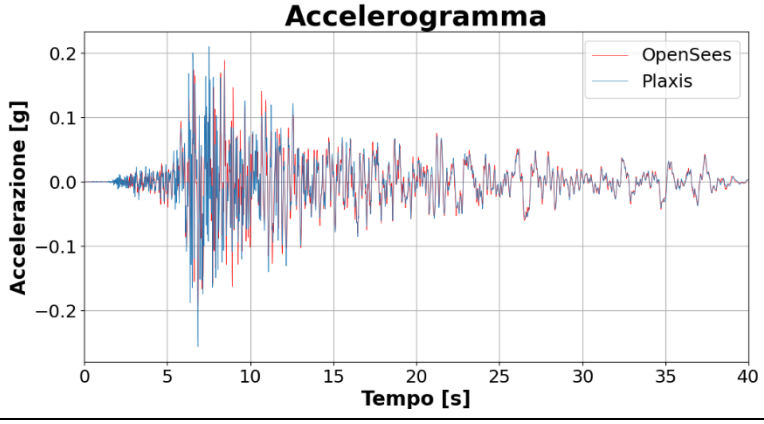


Fig. 85 Sezione X = 540.0 m, spostamenti orizzontali.

Punto	Accelerazione Orizzontale
Y = 0 m	
Y = 10 m	
Y = 20 m	

Y = 30 m	
Y = 40 m	
Y = 50 m	

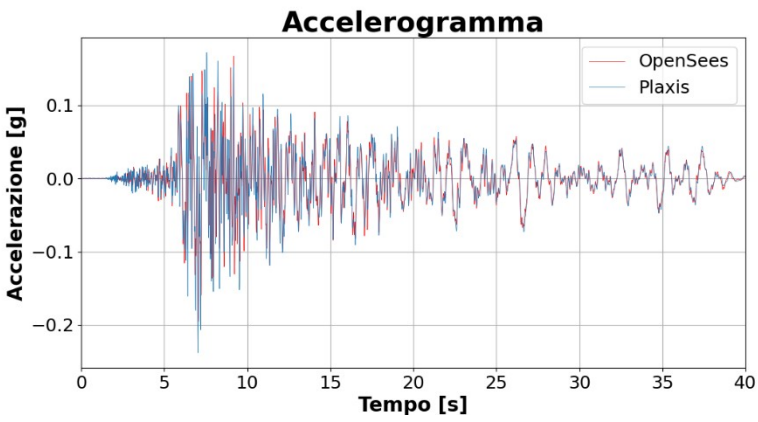
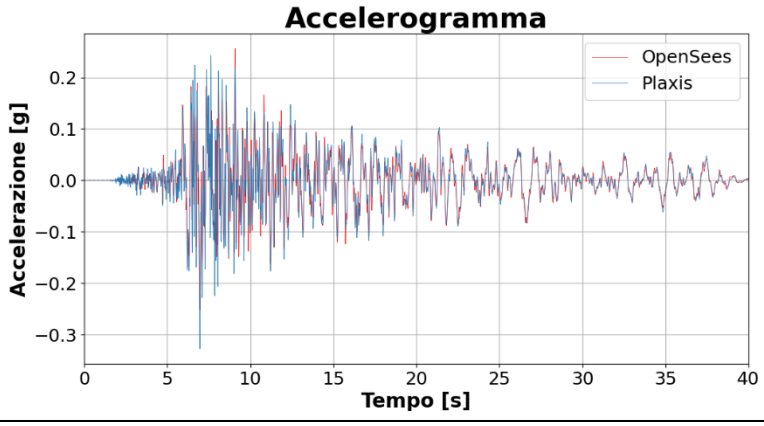
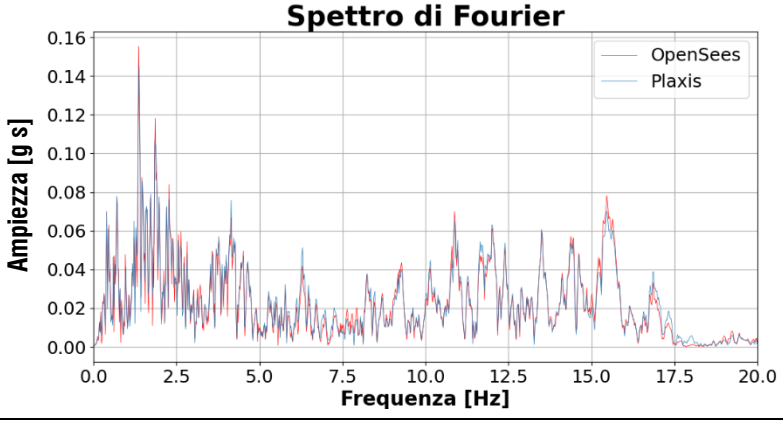
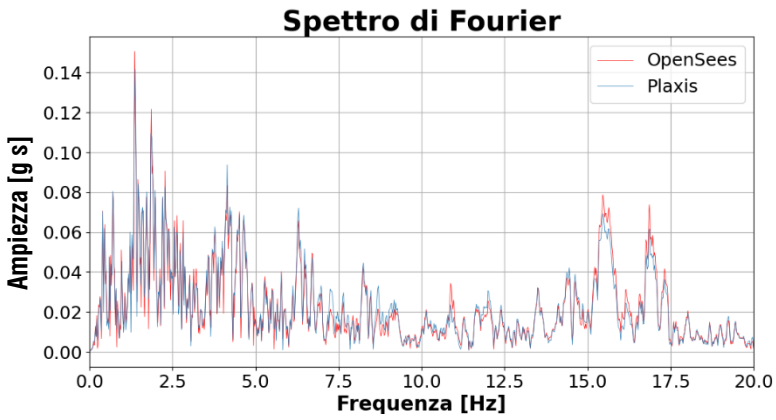
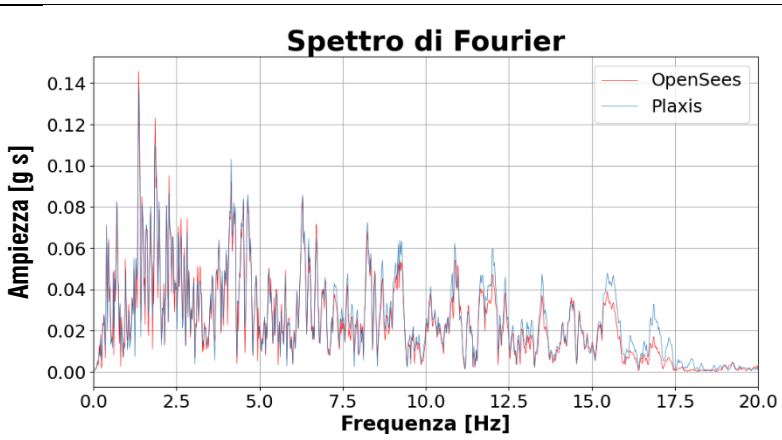
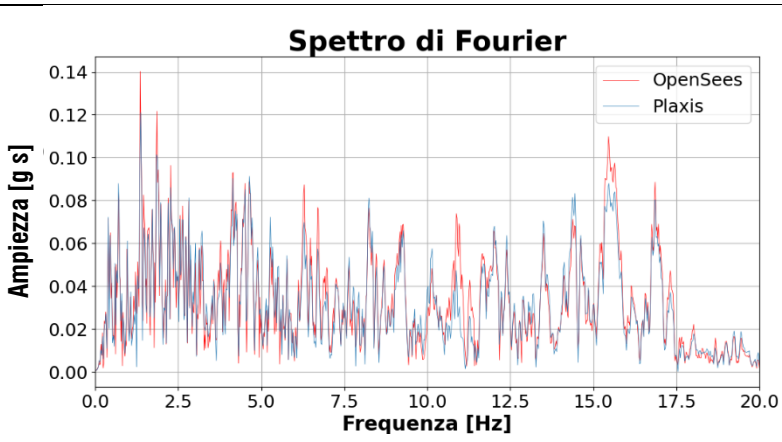
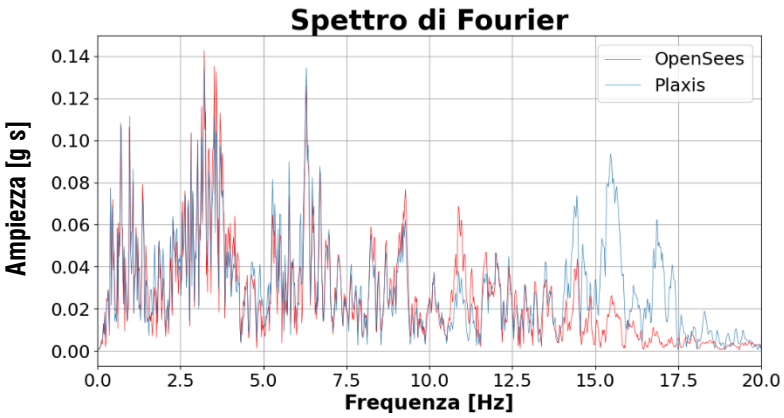
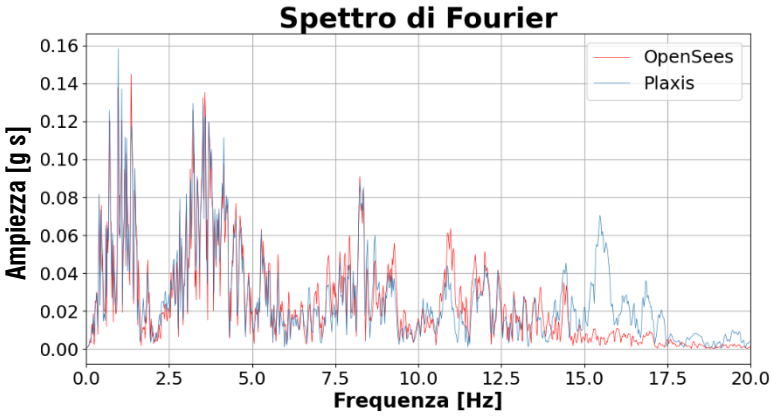
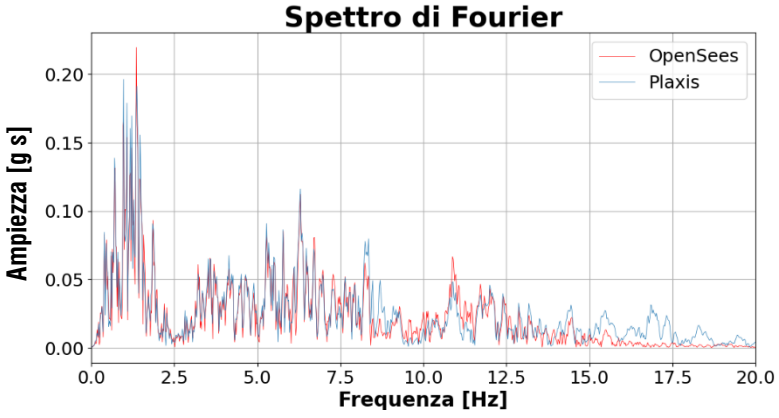
Y = 60 m	
Y = 73 m	

Fig. 86 Sezione X = 540.0 m, accelerazioni orizzontali.

Punto	Spettro di Fourier
Y = 0 m	

Y = 10 m	
Y = 20 m	
Y = 30 m	

Y = 40 m	
Y = 50 m	
Y = 60 m	

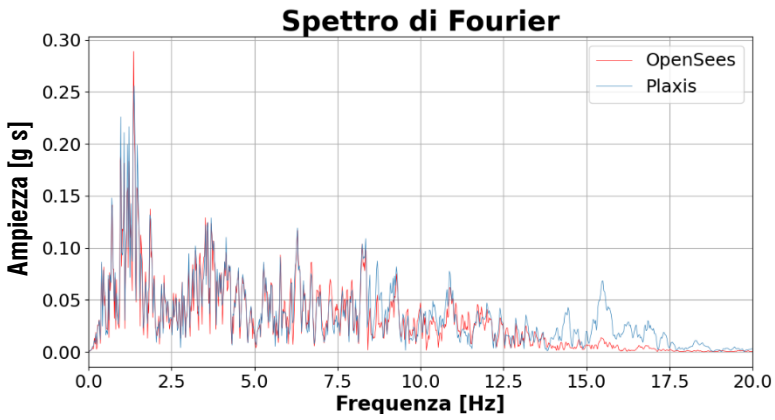
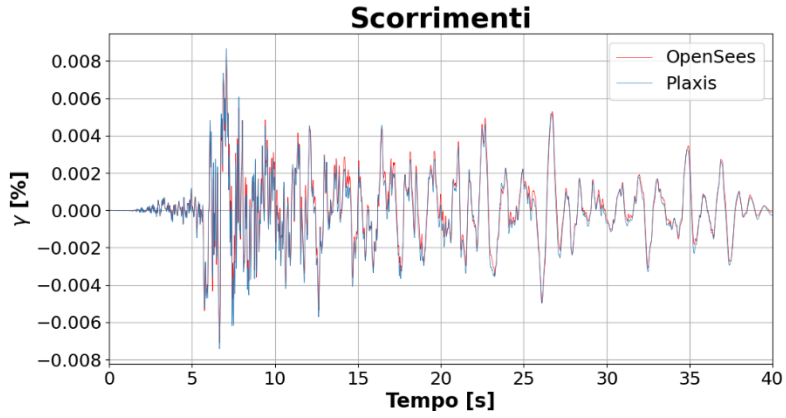
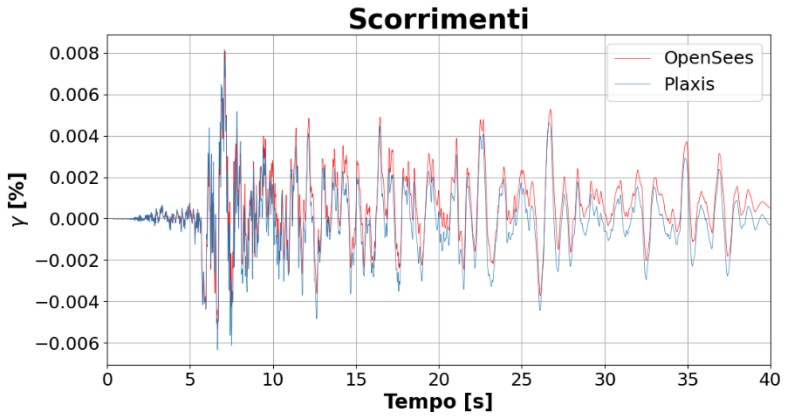
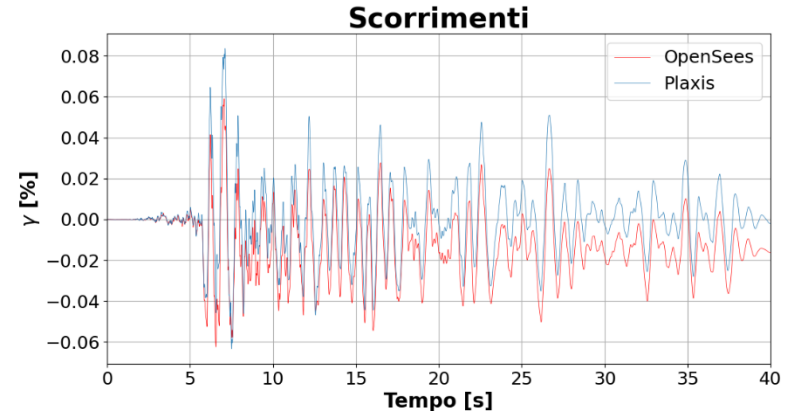
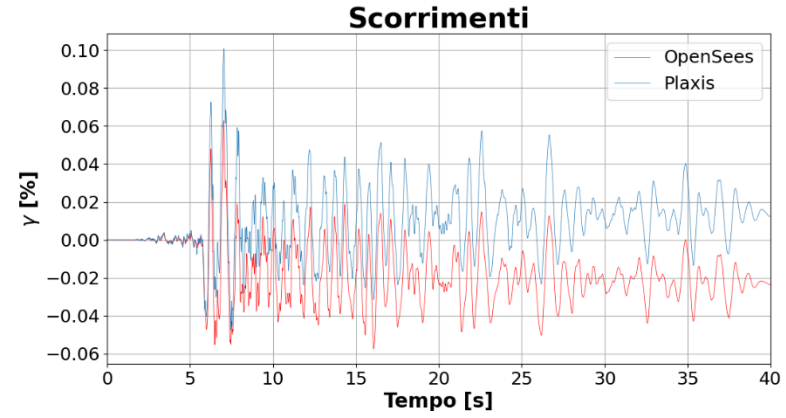
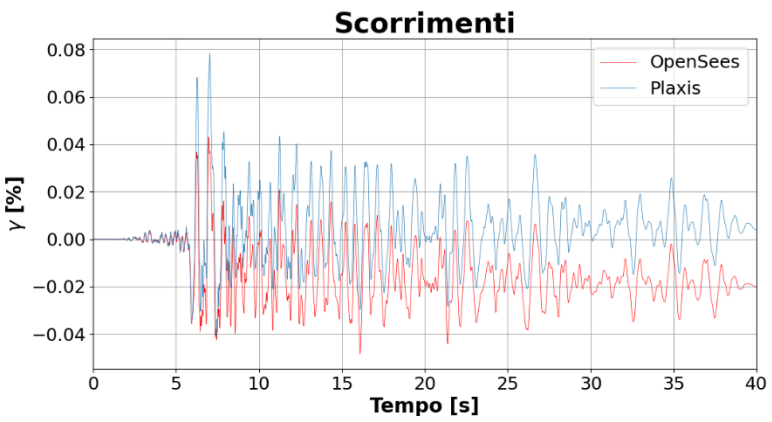
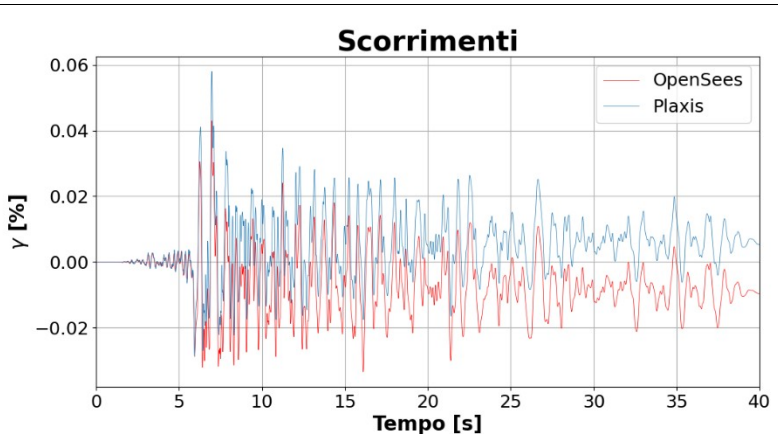
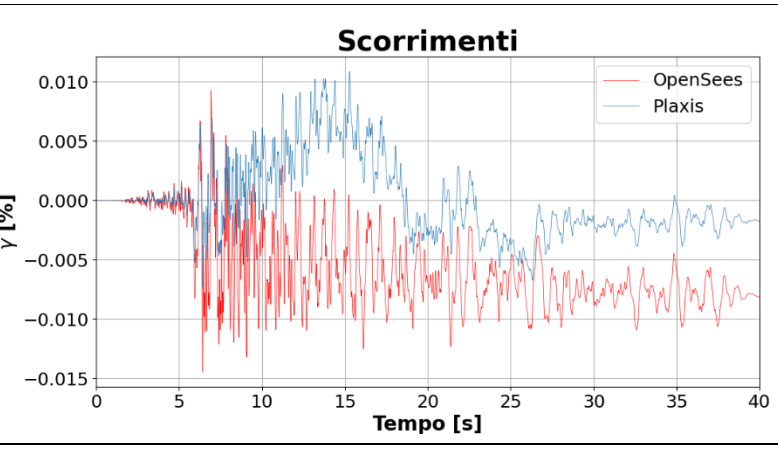
Y = 73 m	

Fig. 87 Sezione X = 540.0 m, spettri di Fourier.

Punto	Scorrimenti orizzontali
Y = 0 m	

Y = 10 m	
Y = 20 m	
Y = 30 m	

Y = 40 m	
Y = 50 m	
Y = 60 m	

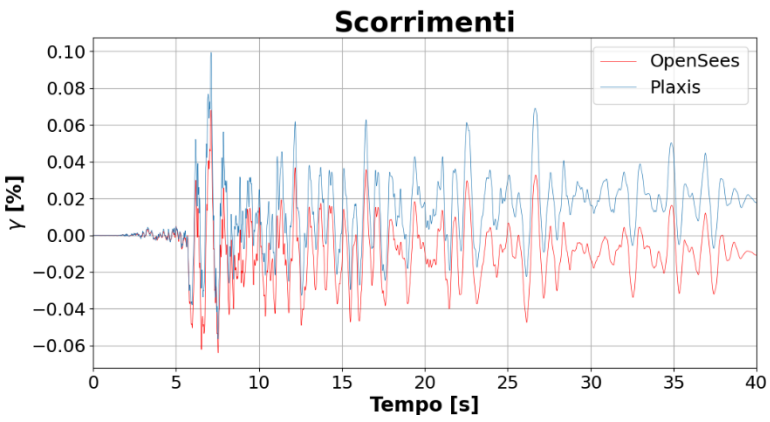
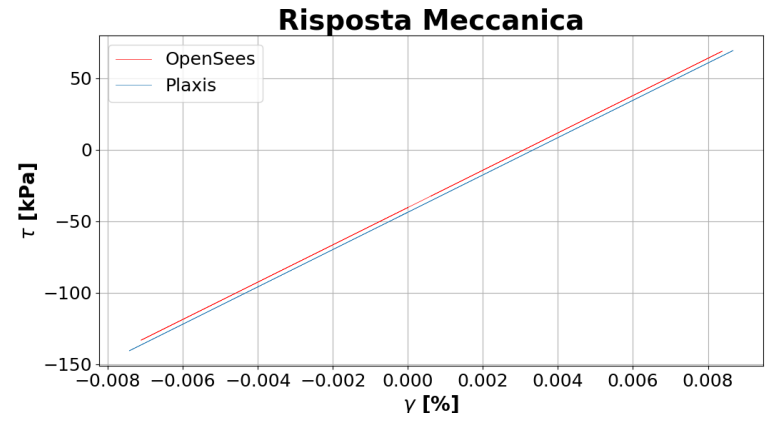
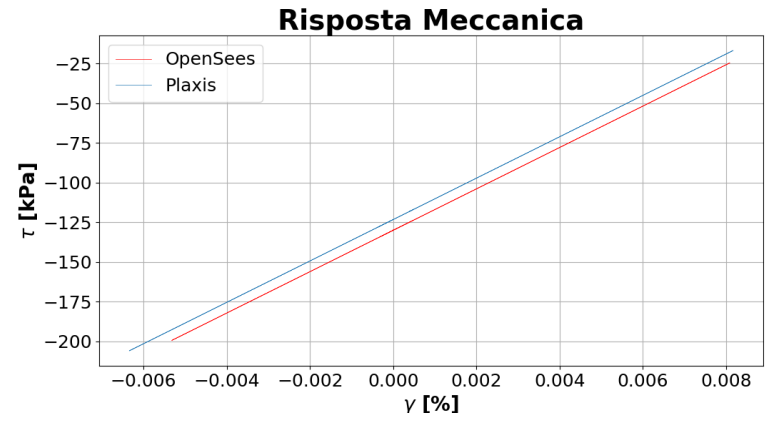
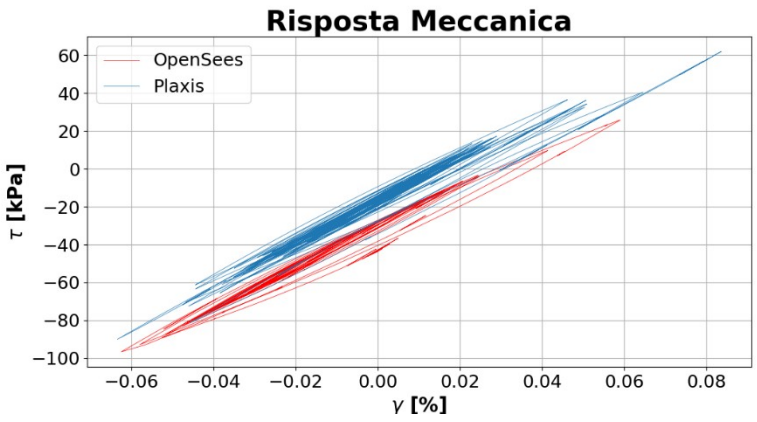
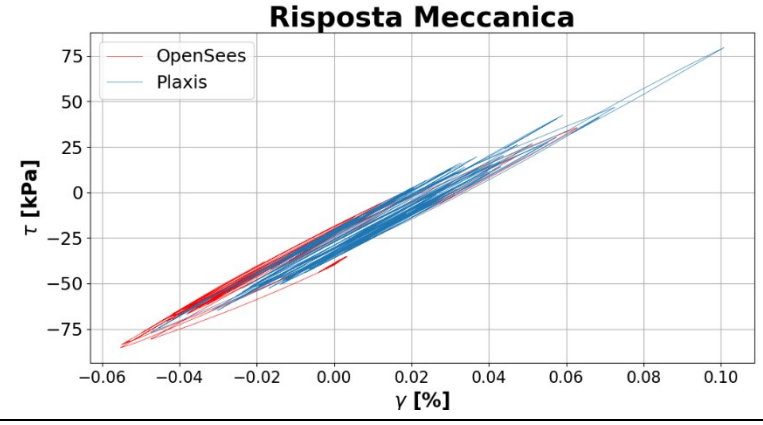
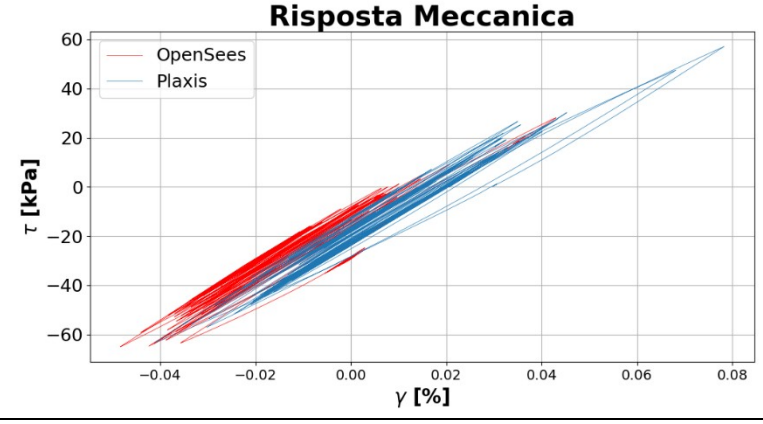
Y = 73 m	

Fig. 88 Sezione X = 540.0 m, scorrimenti orizzontali.

Punto	Risposta meccanica
Y = 0 m	
Y = 10 m	

Y = 20 m	
Y = 30 m	
Y = 40 m	

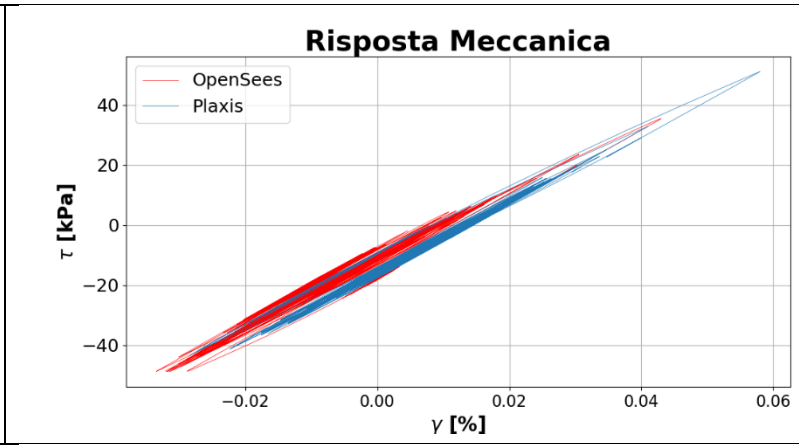
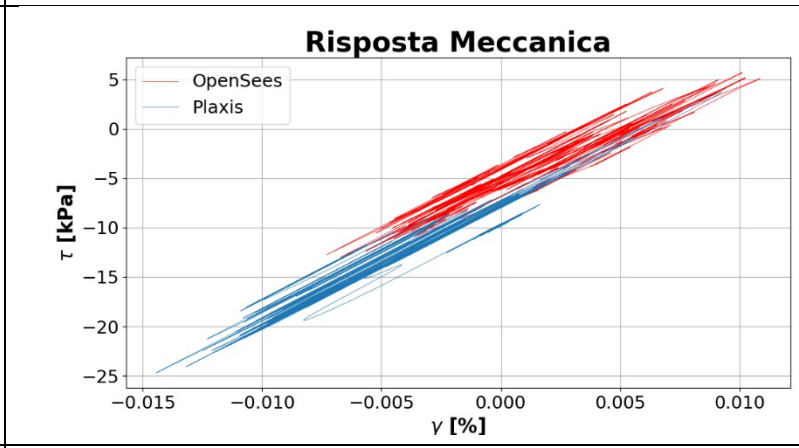
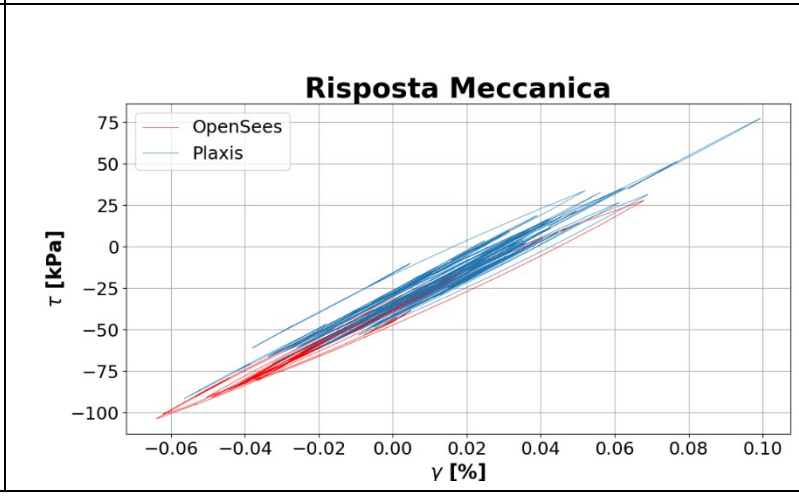
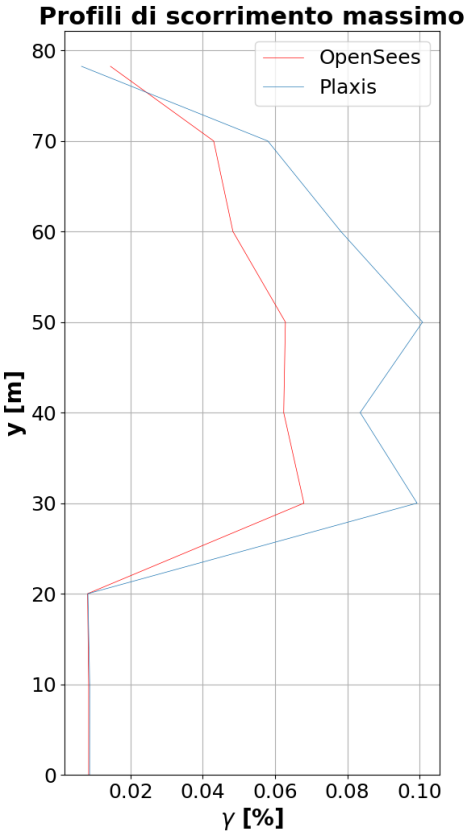
Y = 50 m	
Y = 60 m	
Y = 73 m	

Fig. 89 Sezione X = 540.0 m, risposta meccanica.

Profilo degli scorrimenti massimi	Profilo delle accelerazioni massime



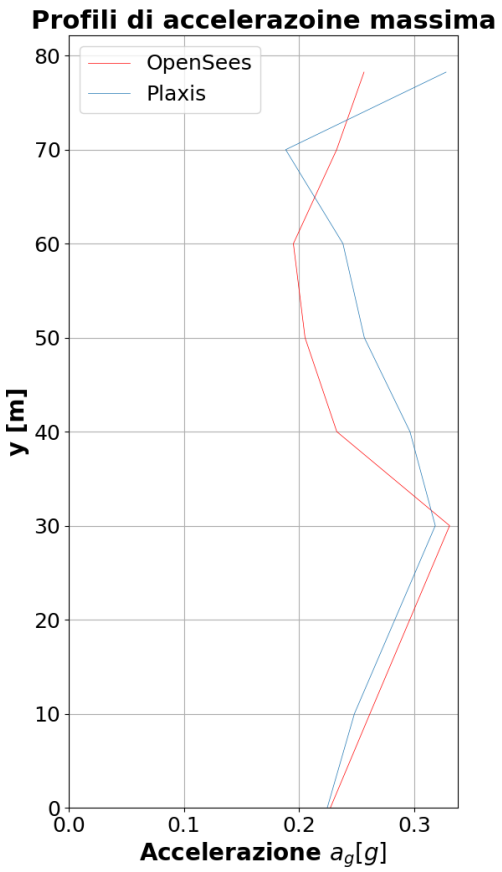
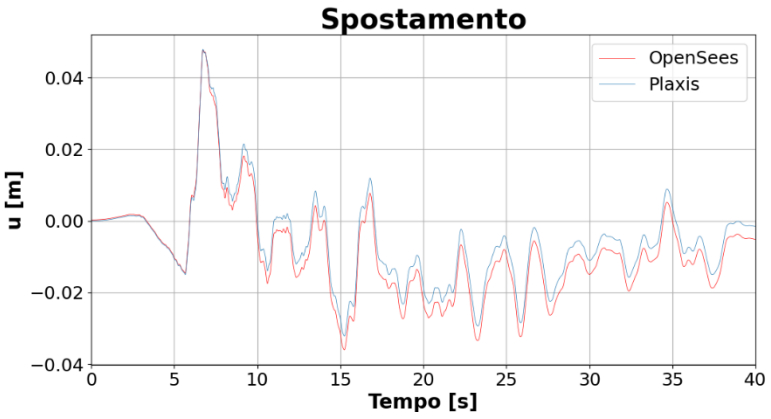
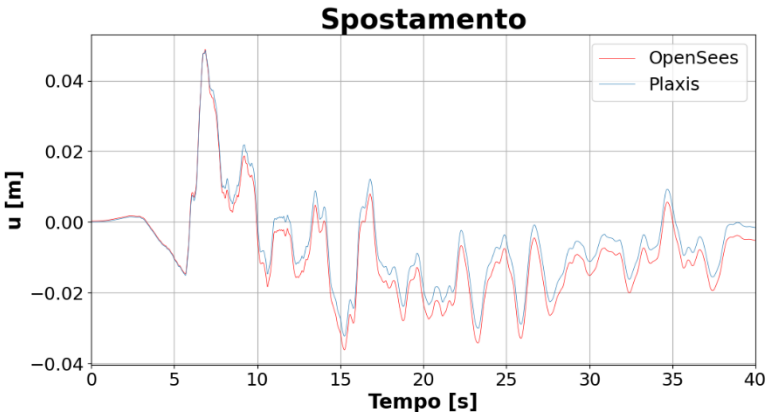
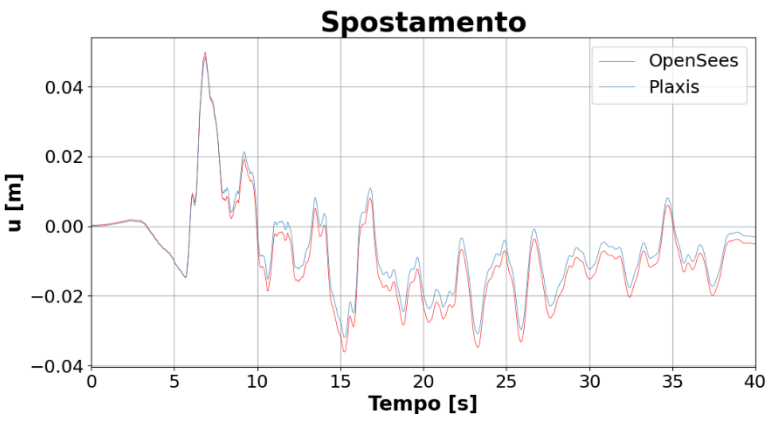
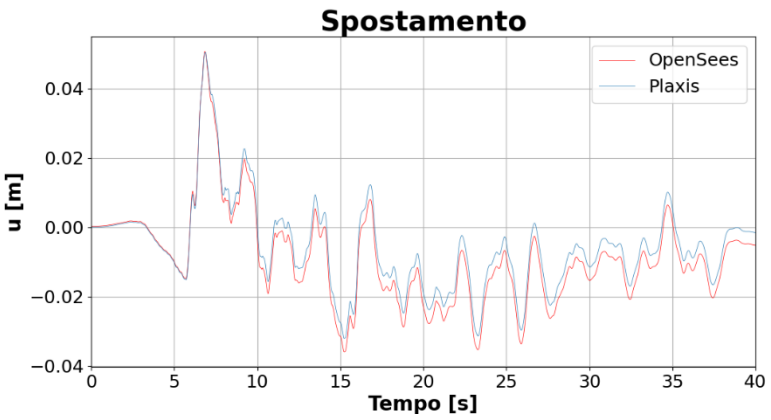
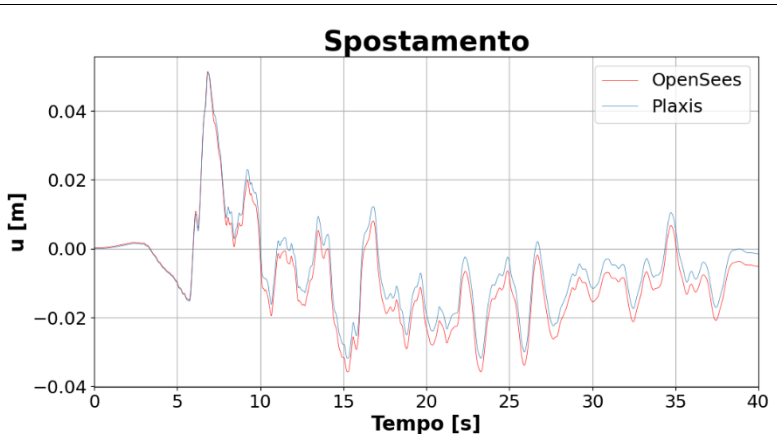
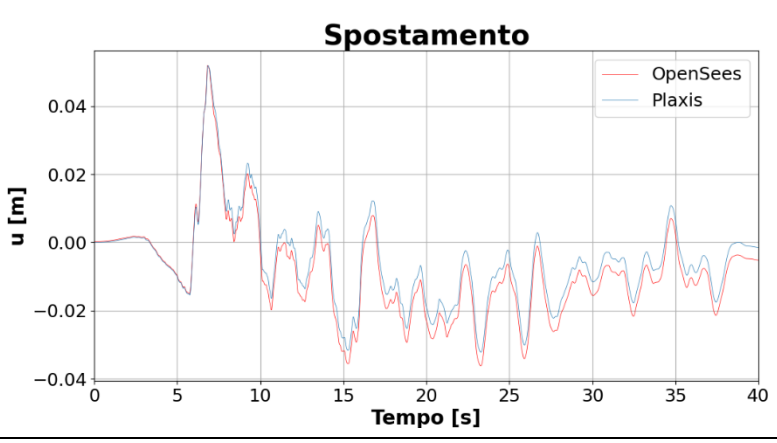


Fig. 90 Sezione X = 540.0 m, profili con la profondità.

VERTICALE MEDIANA (X = 870.0 m)

Punto	Spostamento Orizzontale
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

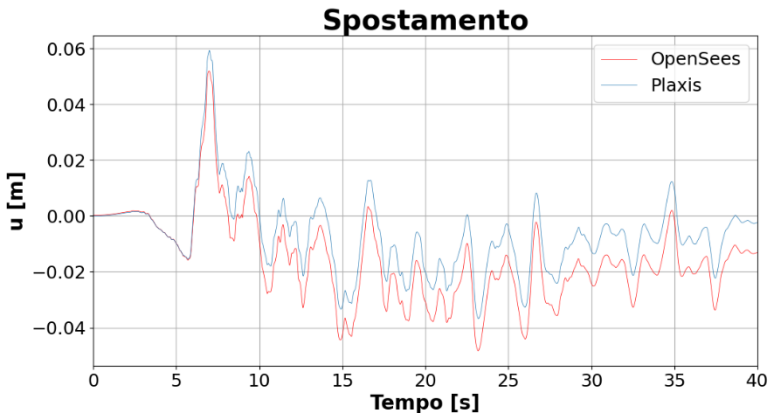
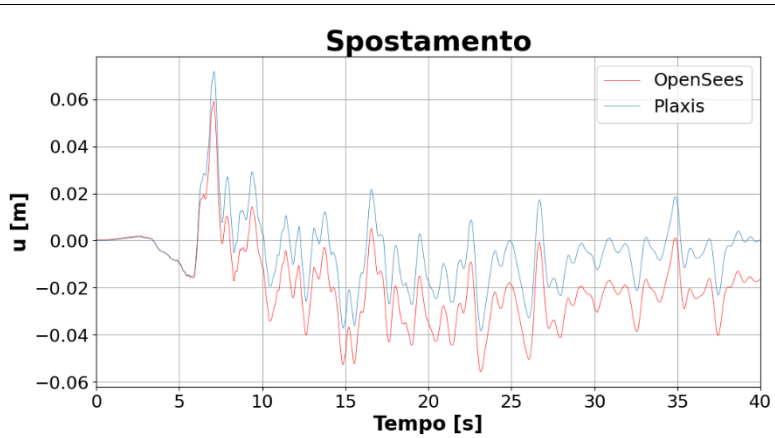
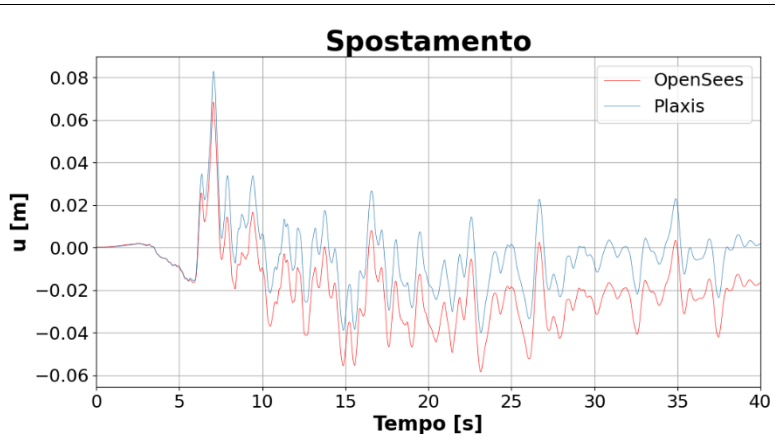
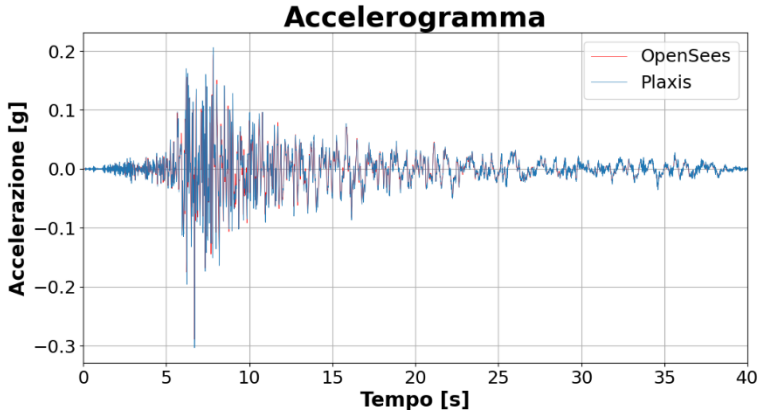
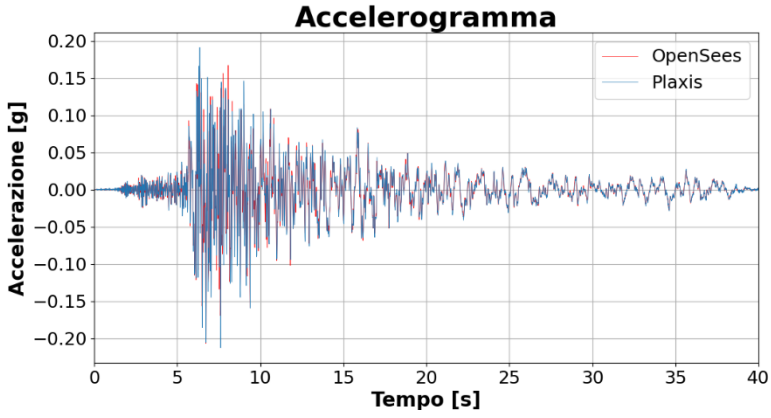
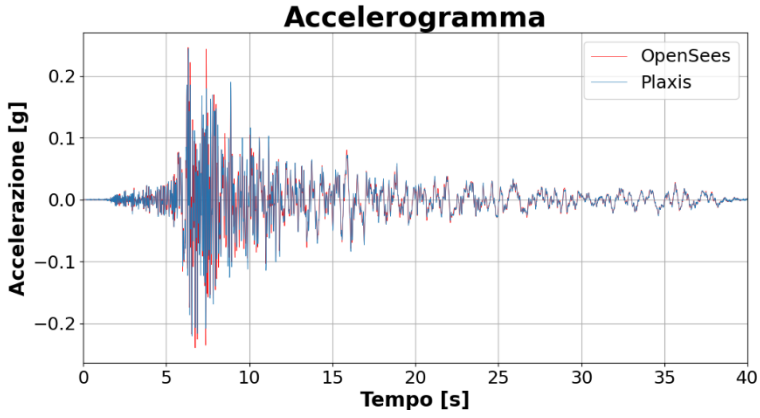
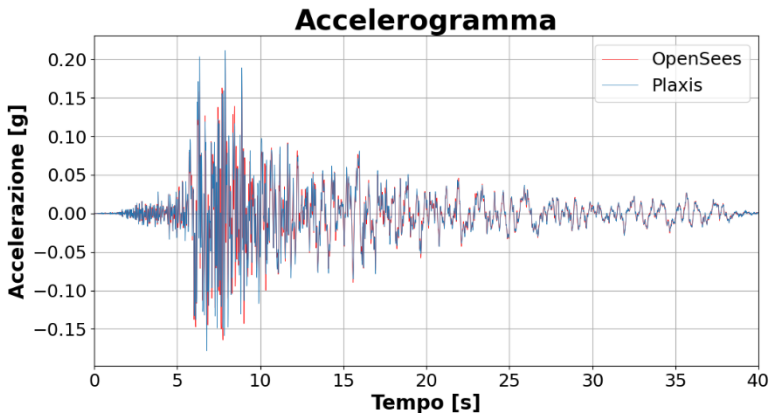
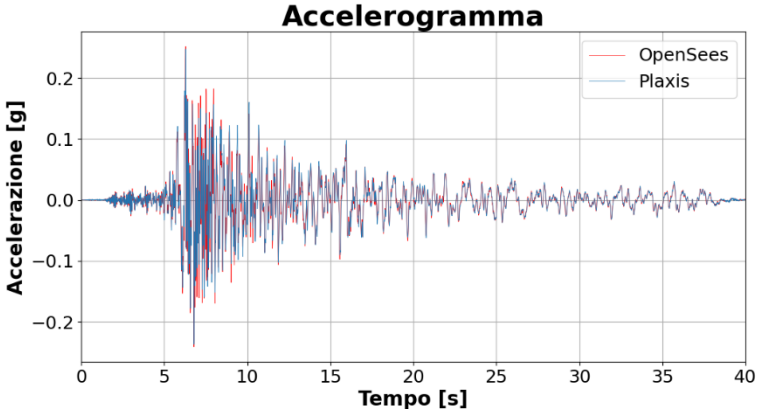
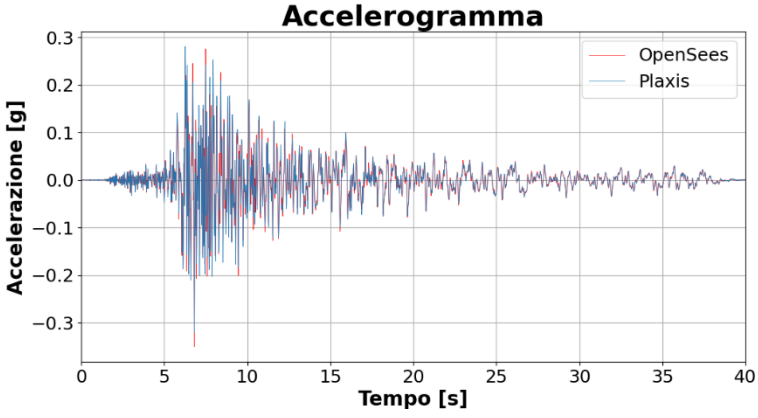
Y = 90 m	
Y = 105 m	
Y = 129 m	

Fig. 91 Sezione X = 870.0 m, spostamenti orizzontali.

Punto	Accelerazione Orizzontale
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

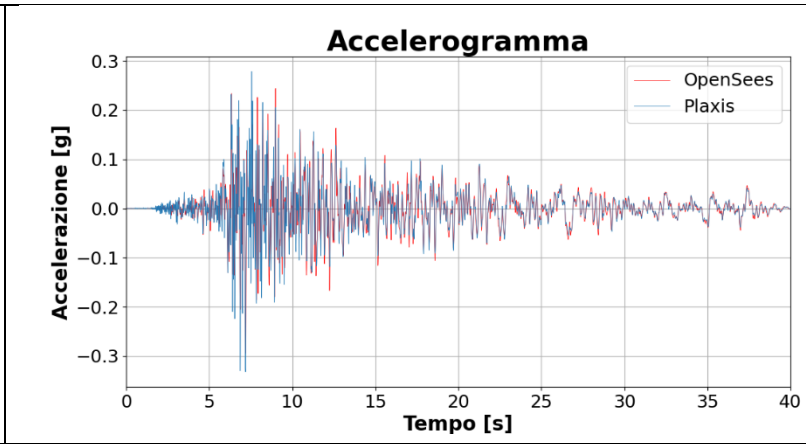
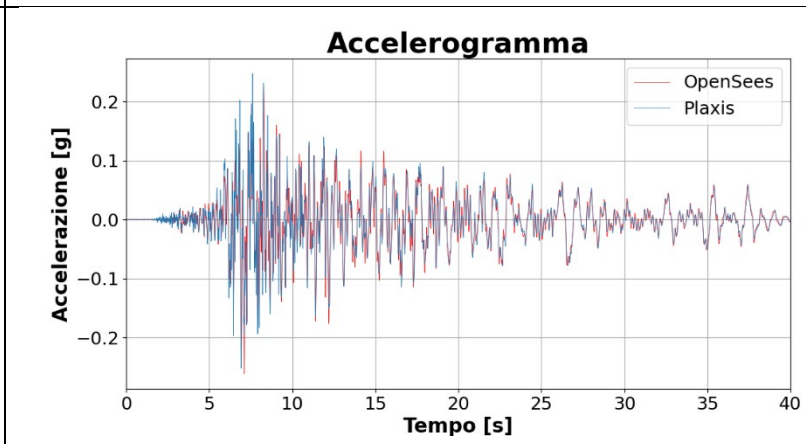
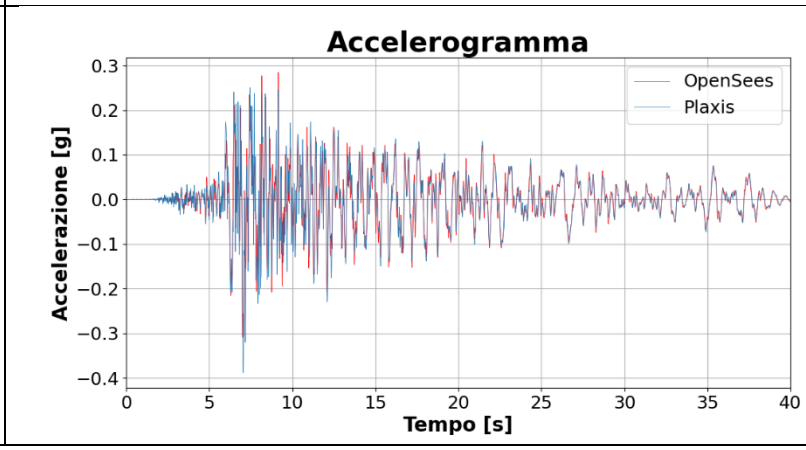
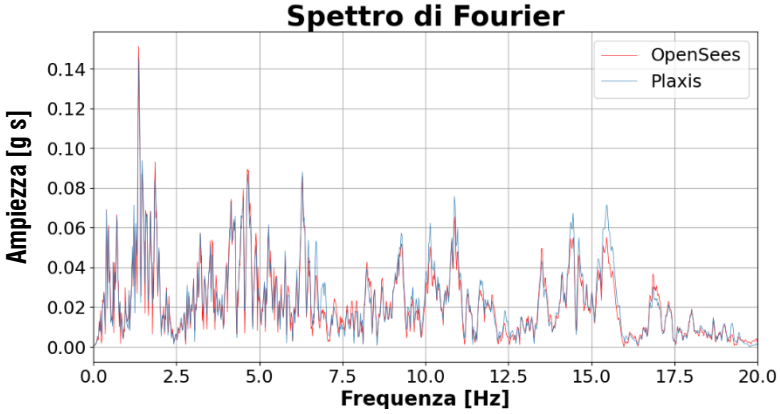
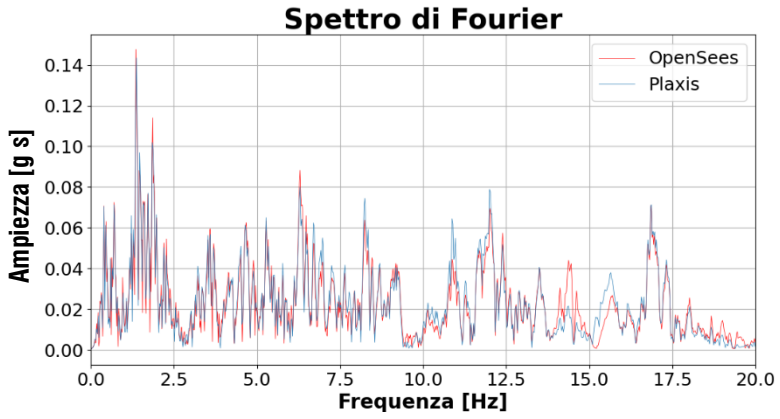
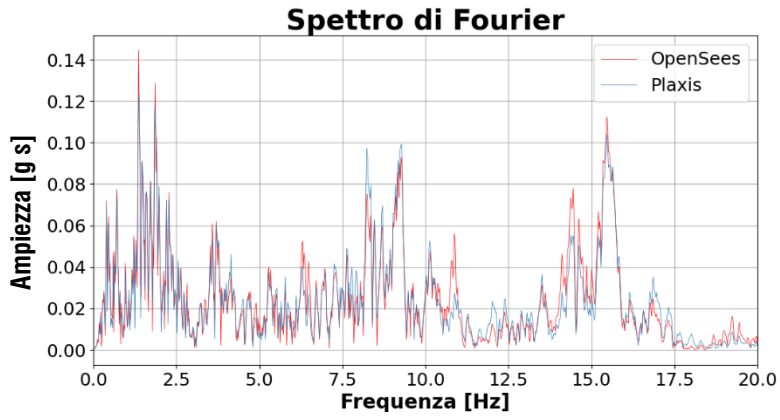
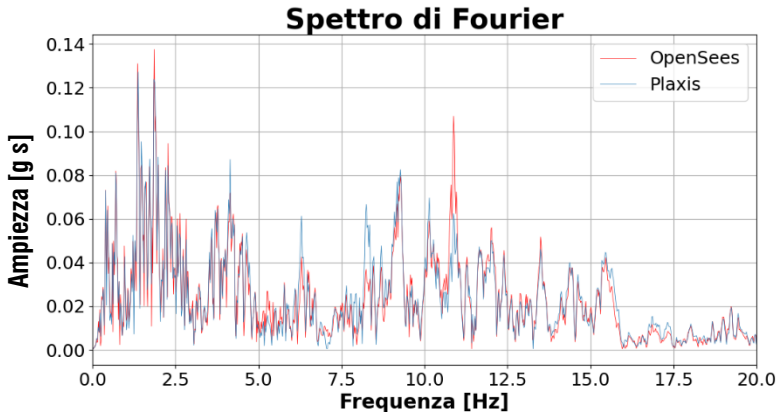
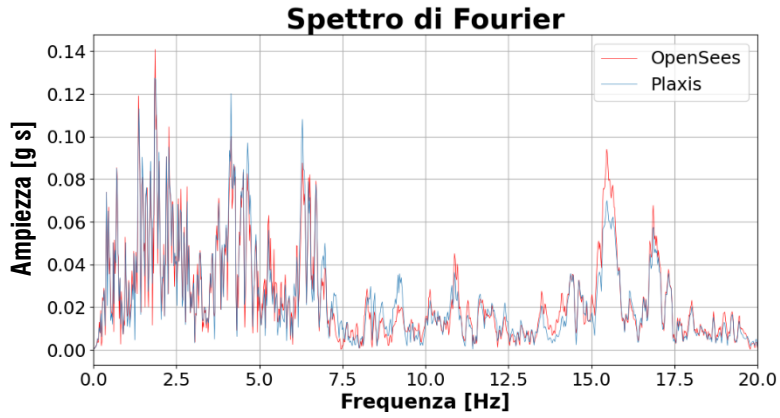
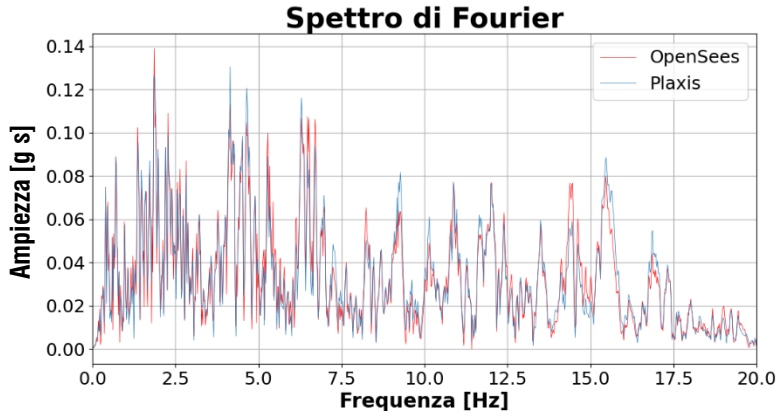
Y = 90 m	
Y = 105 m	
Y = 129 m	

Fig. 92 Sezione X = 870.0 m, accelerazioni orizzontali.

Punto	Spettro di Fourier
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

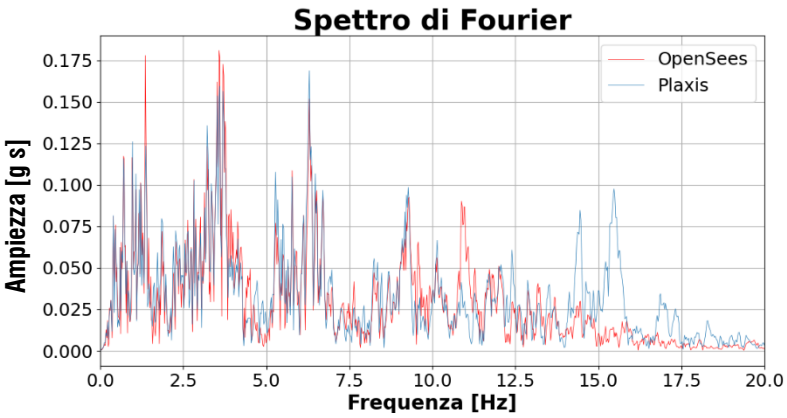
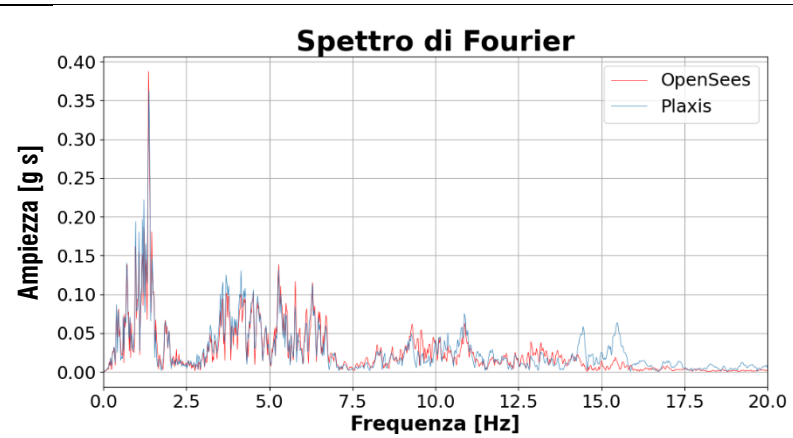
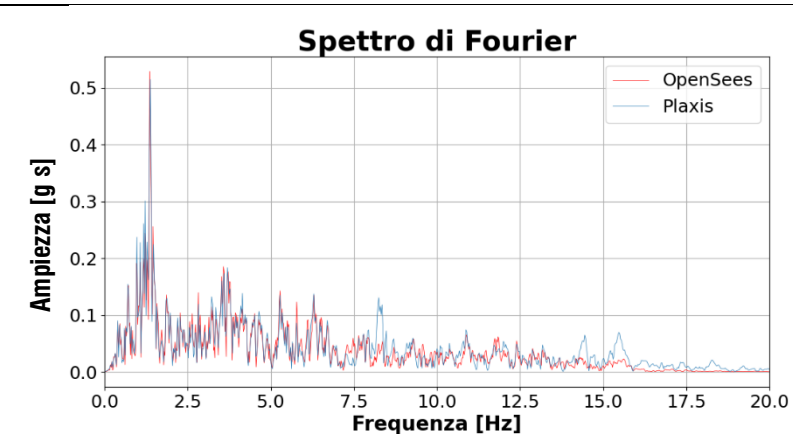
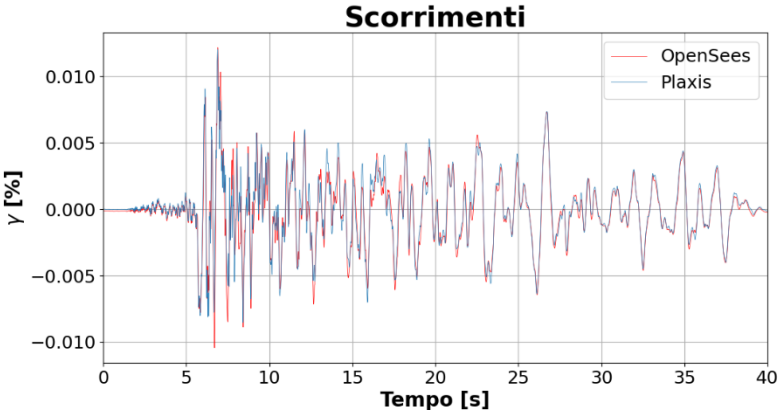
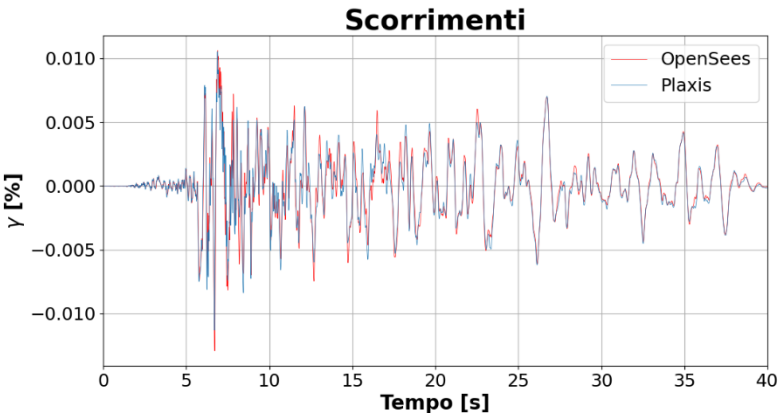
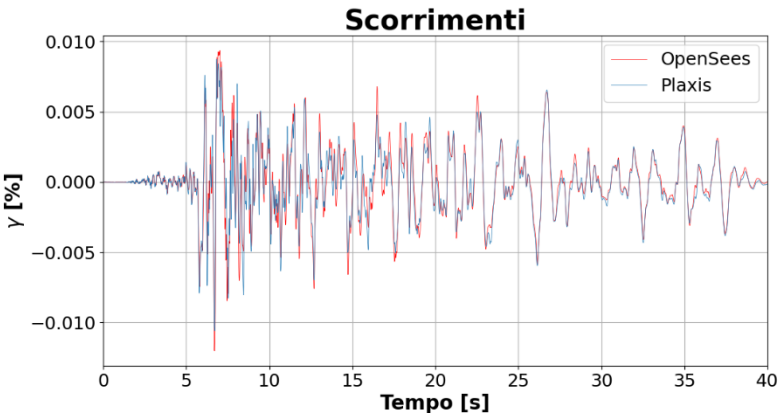
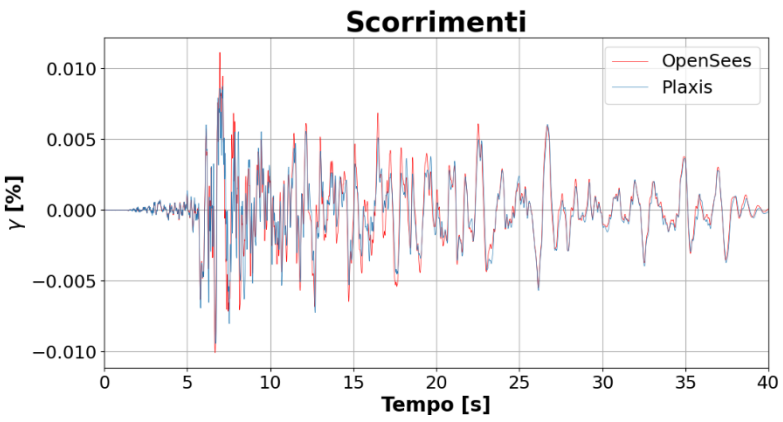
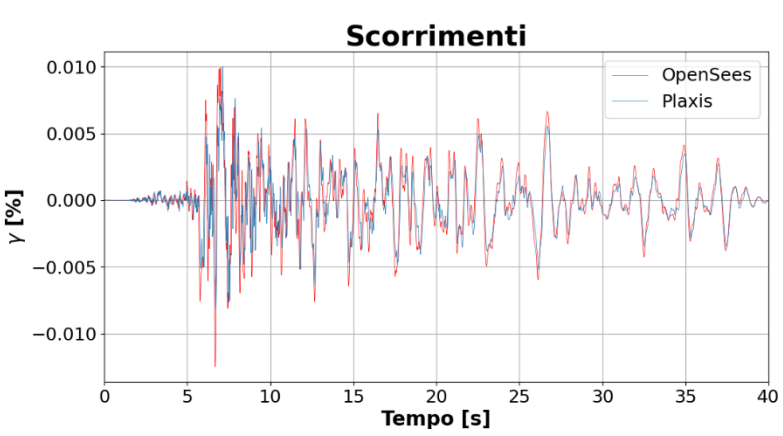
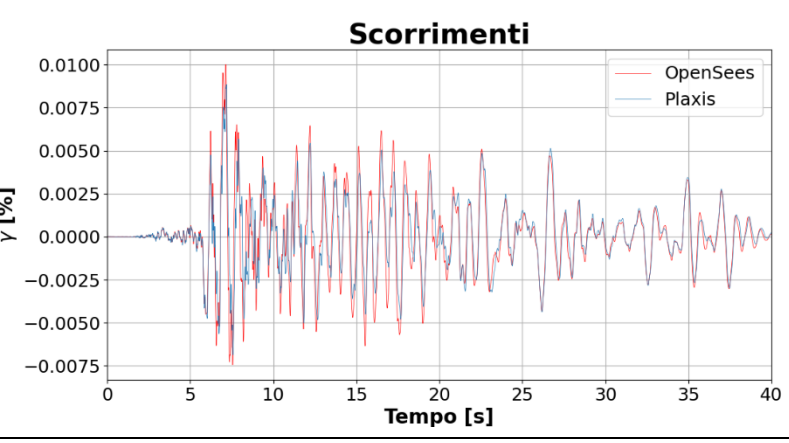
Y = 90 m	
Y = 105 m	
Y = 129 m	

Fig. 93 Sezione X = 870.0 m, spettri di Fourier.

Punto	Scorrimenti orizzontali
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

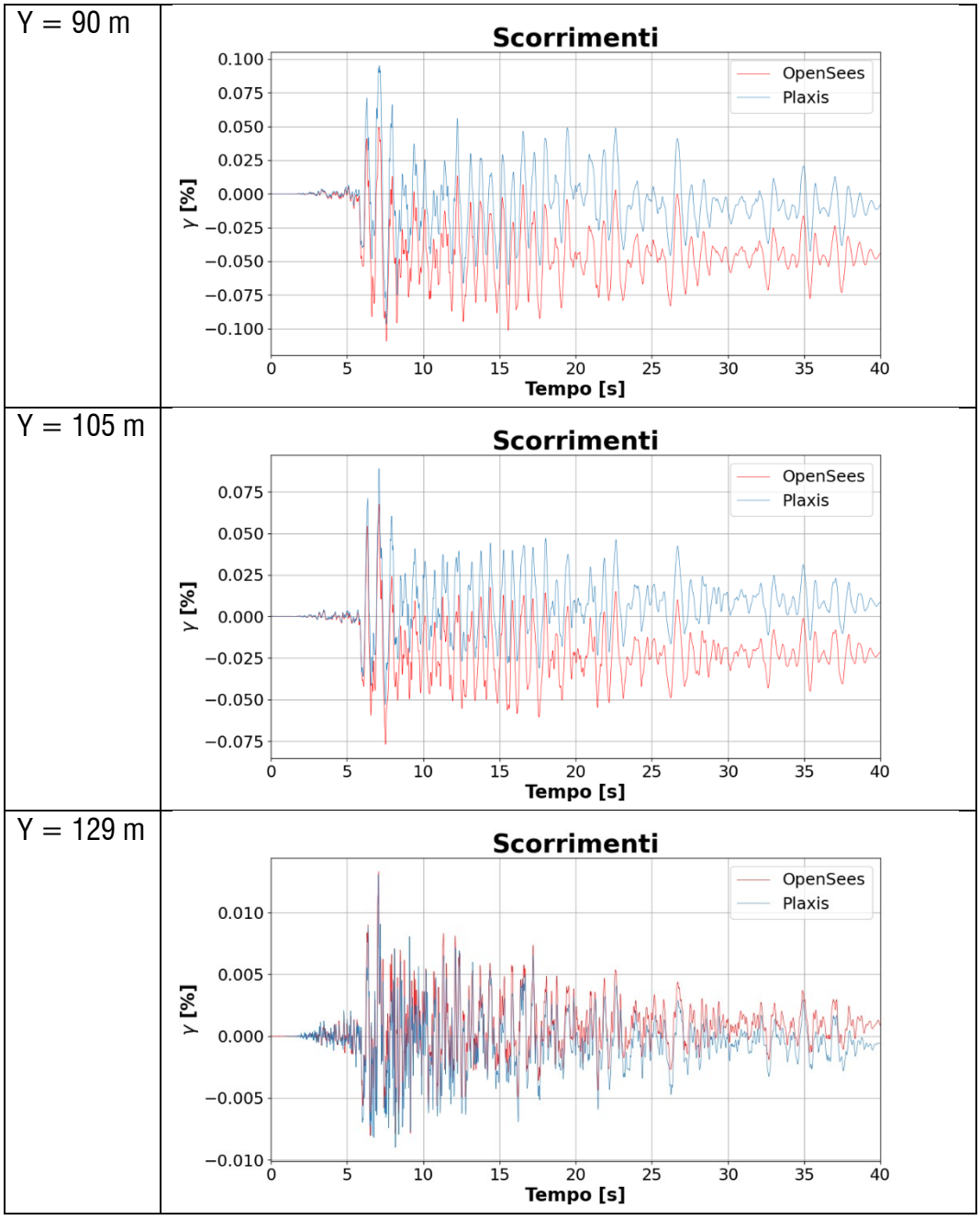
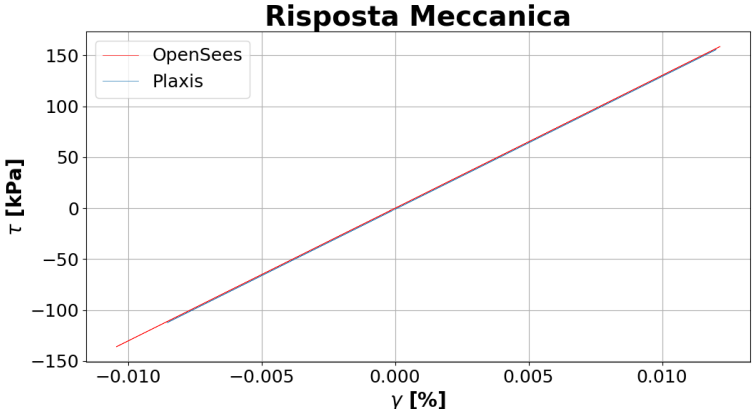
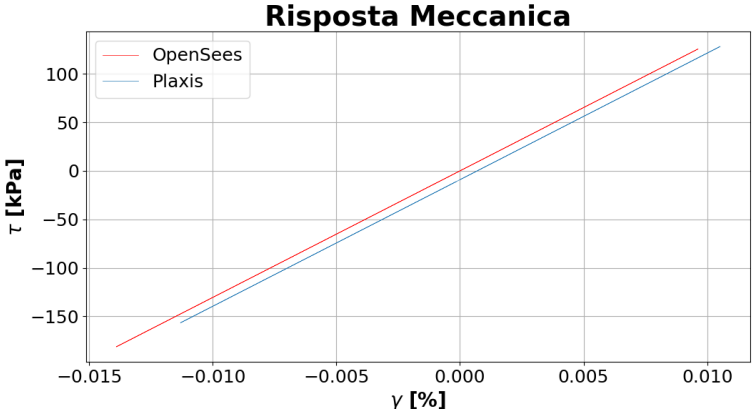
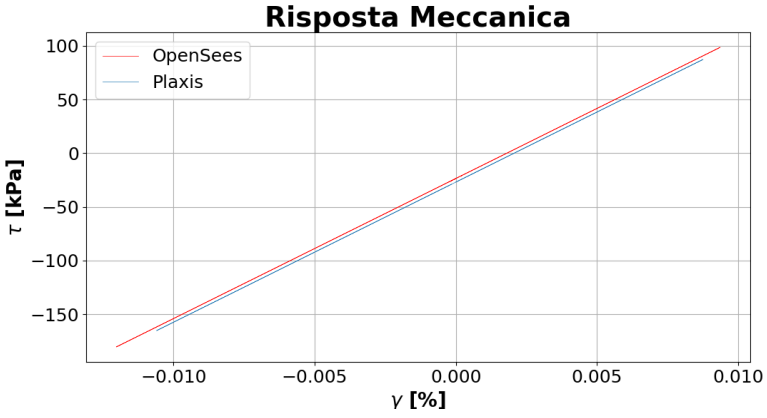
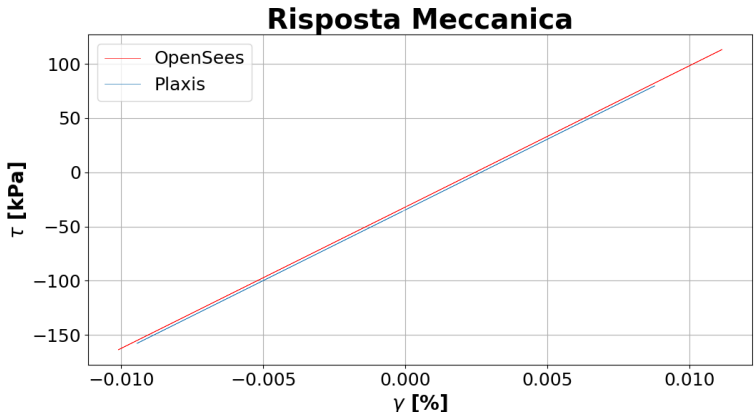
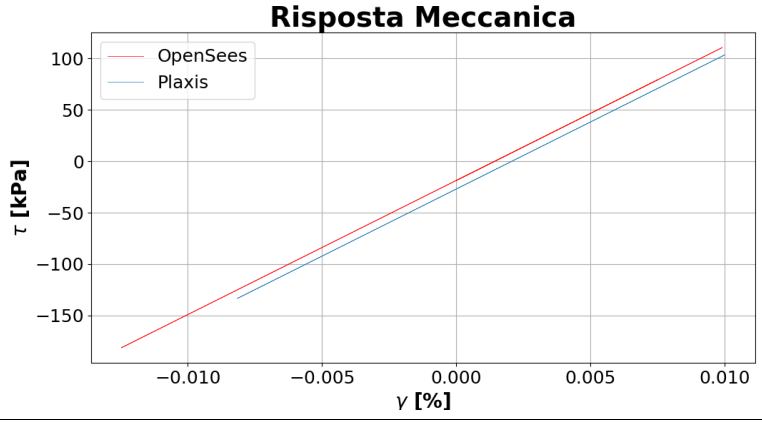
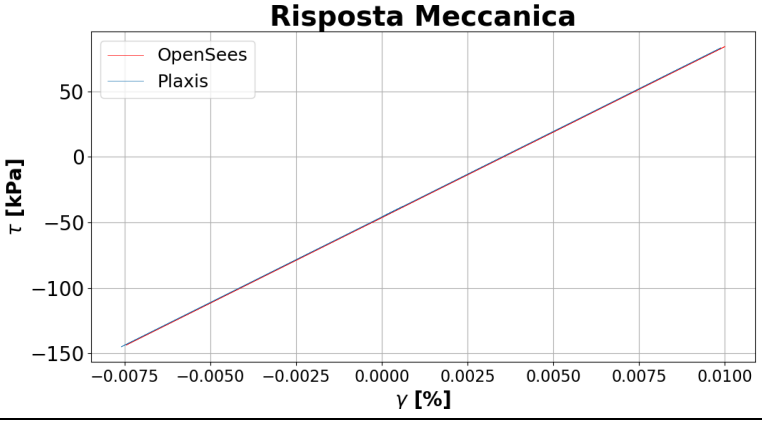


Fig. 94 Sezione X = 870.0 m, scorrimenti orizzontali.

Punto	Risposta meccanica
Y = 0 m	
Y = 15 m	
Y = 30 m	

Y = 45 m	
Y = 60 m	
Y = 75 m	

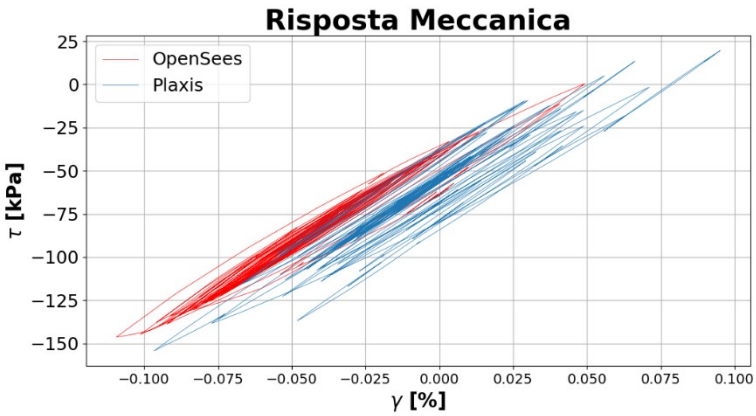
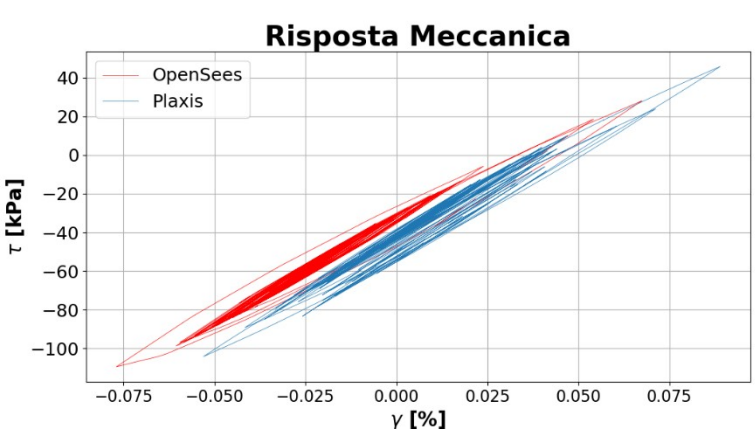
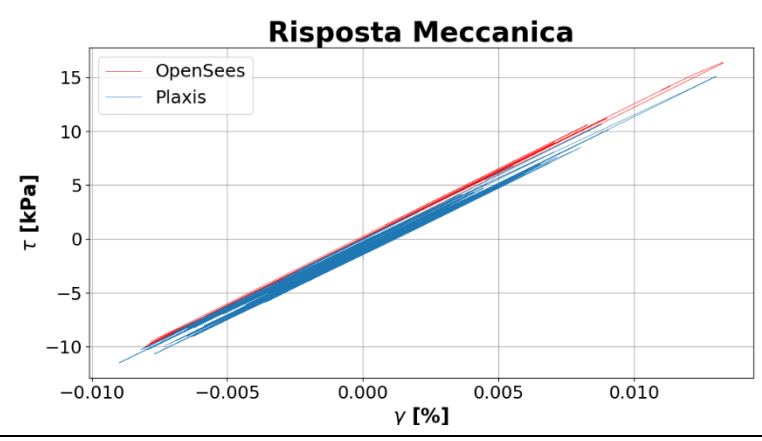
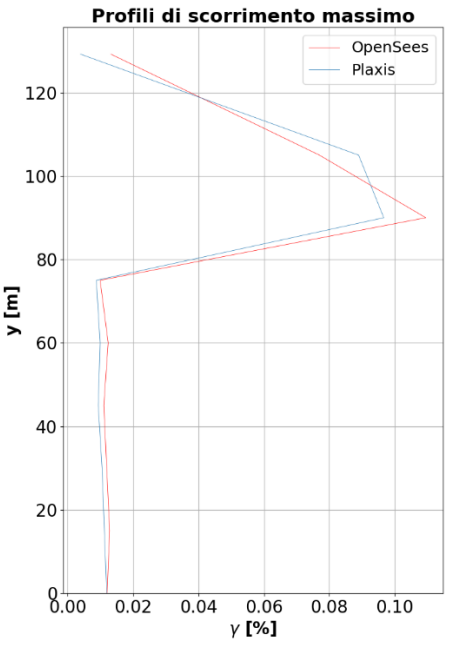
Y = 90 m	
Y = 105 m	
Y = 129 m	

Fig. 95 Sezione X = 870.0 m, risposta meccanica.

Profilo degli scorrimenti massimi	Profilo delle accelerazioni massime



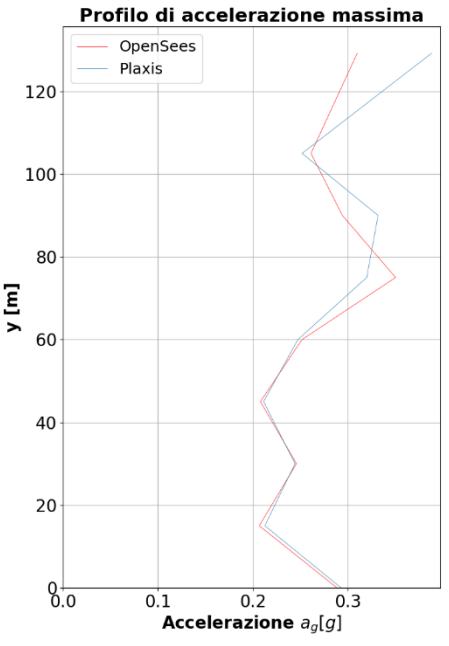
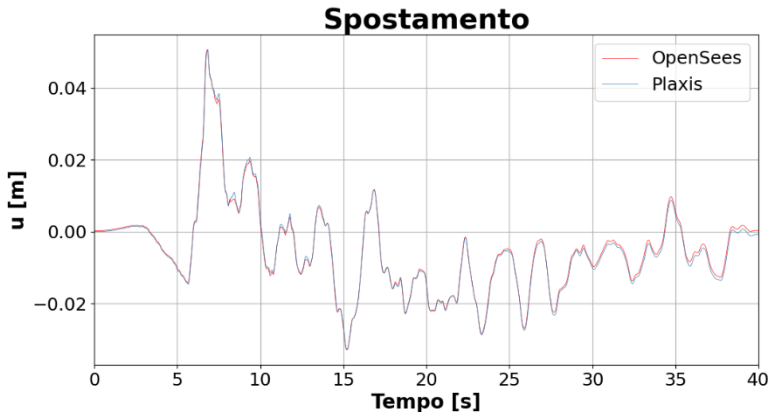
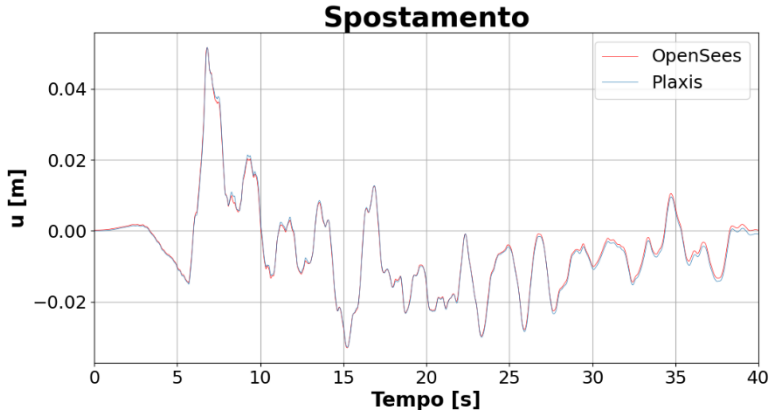
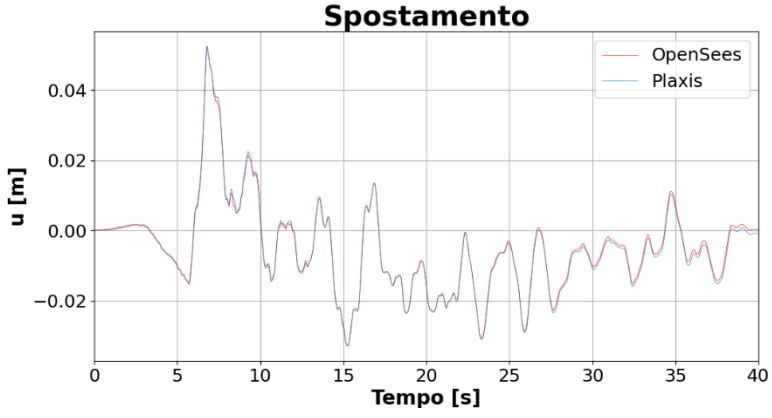
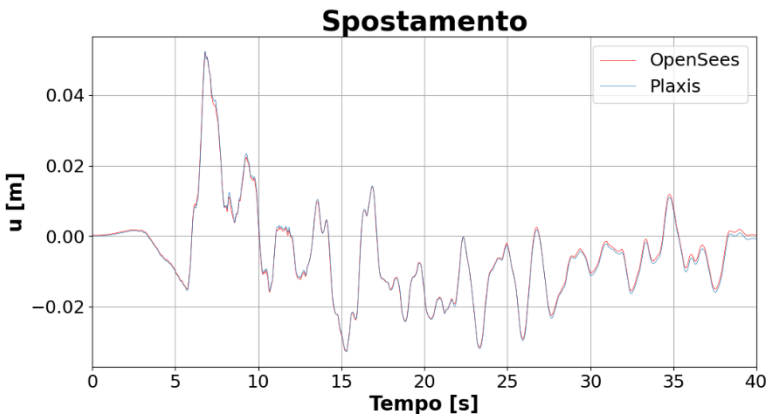
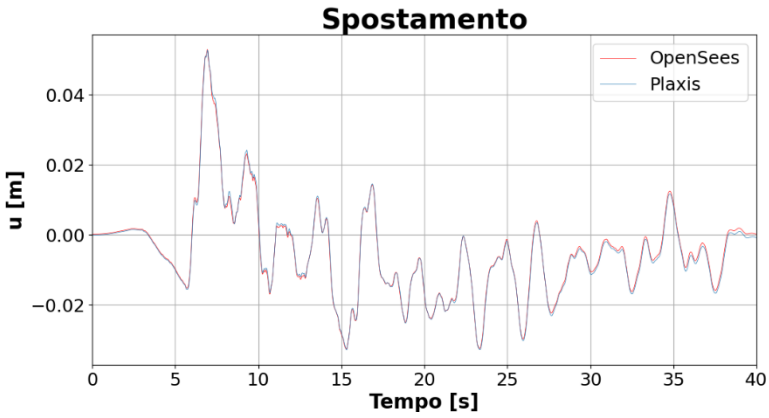
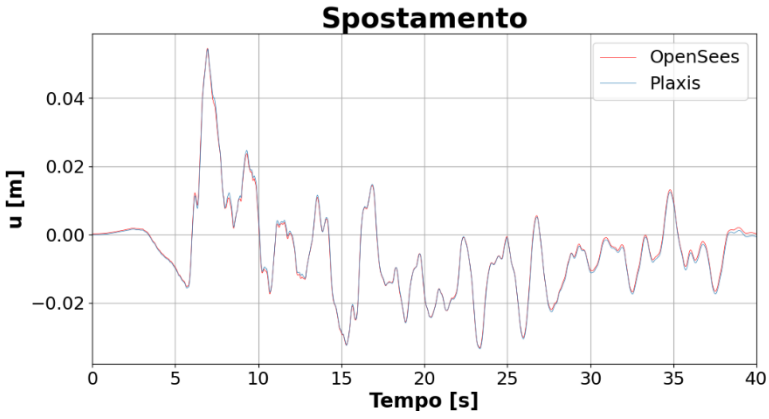
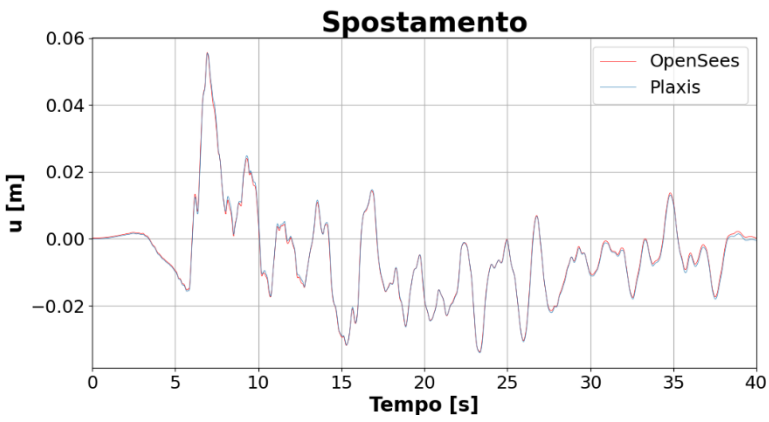
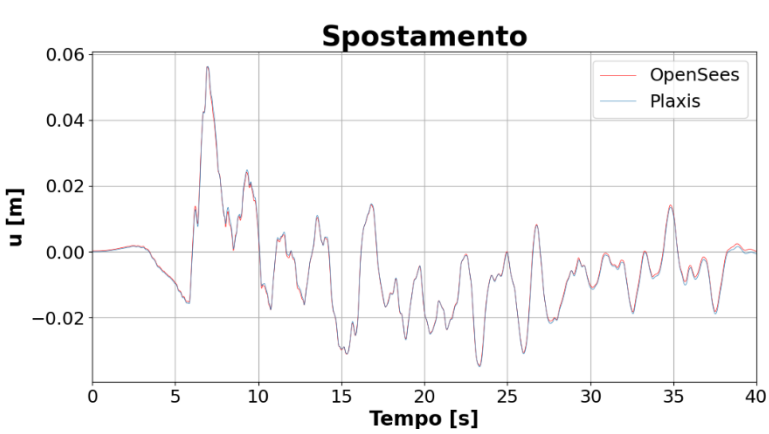
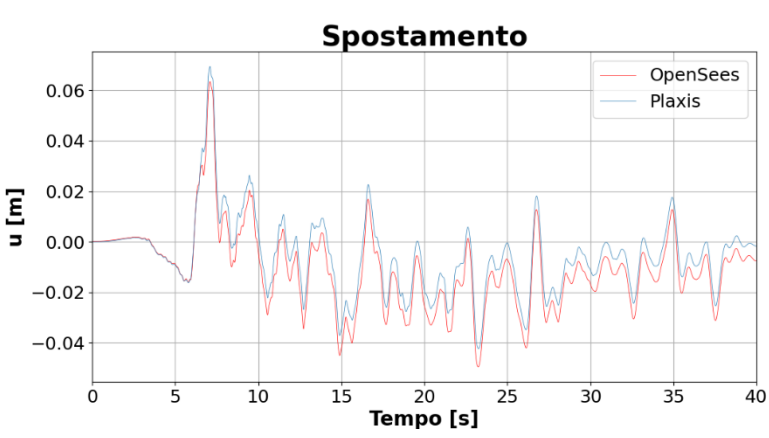


Fig. 96 Sezione X = 870.0 m, profili con la profondità.

VERTICALE DI MONTE (X = 1250.0 m)

Punto	Spostamento Orizzontale
Y = 0 m	
Y = 20 m	
Y = 40 m	

Y = 60 m	
Y = 80 m	
Y = 100 m	

Y = 120 m	
Y = 140 m	
Y = 160 m	

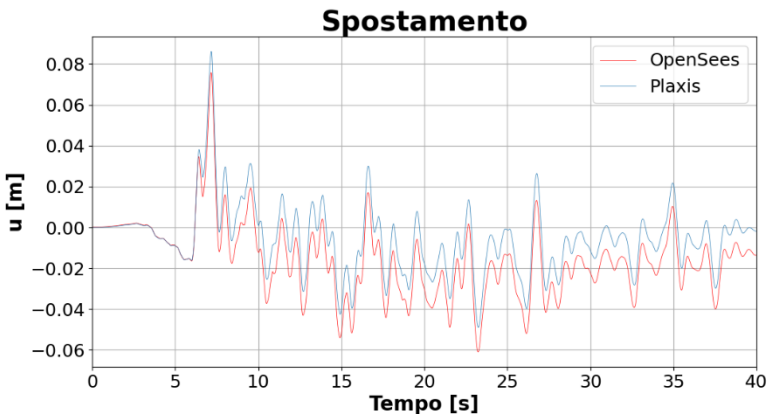
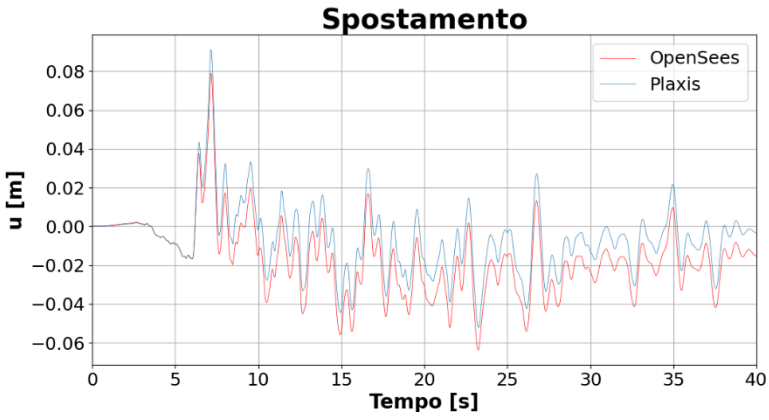
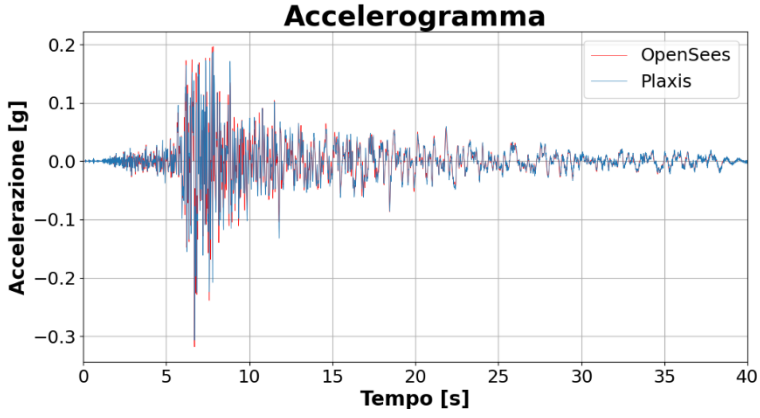
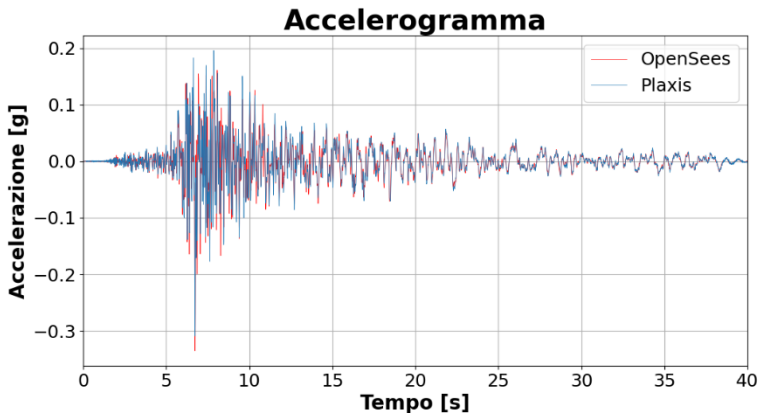
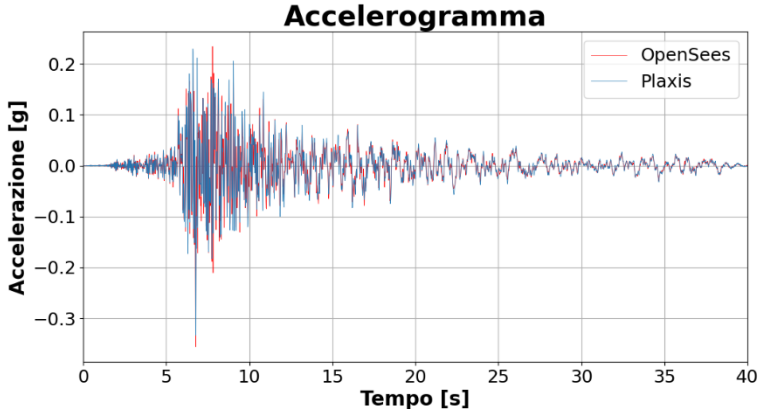
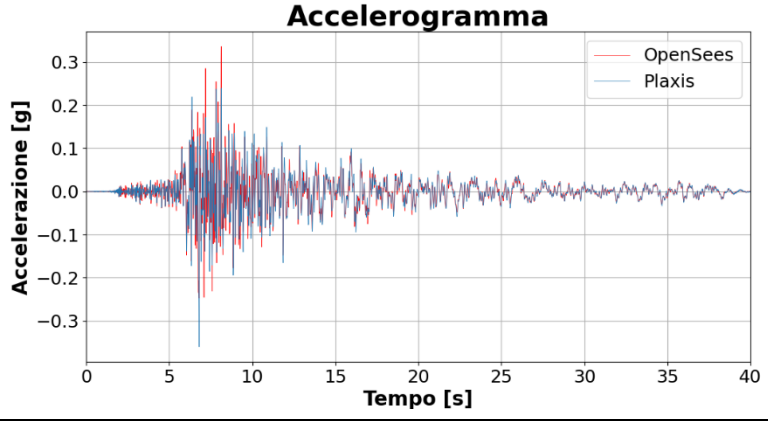
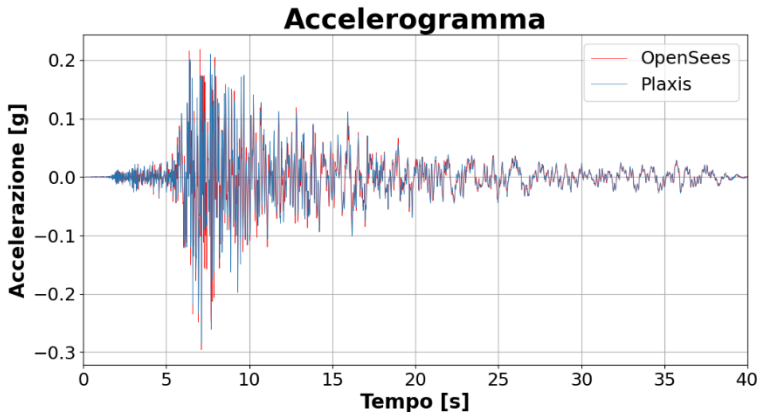
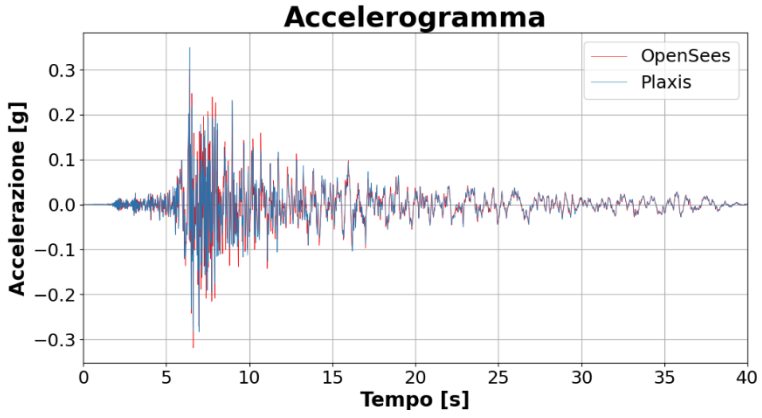
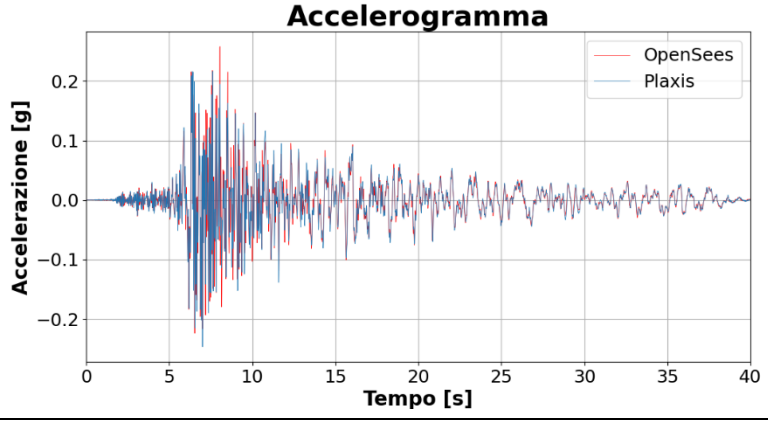
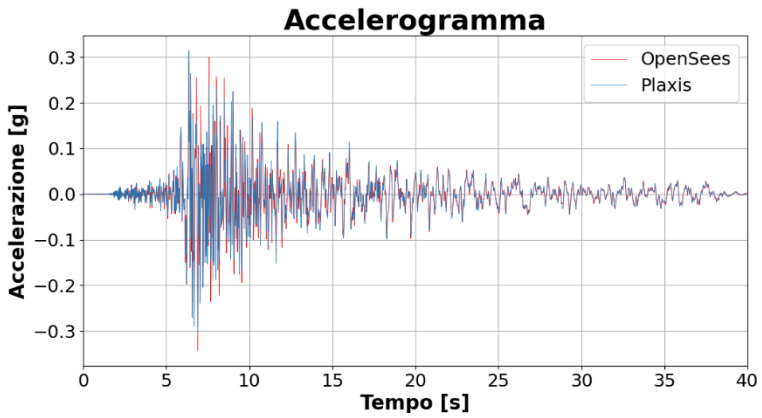
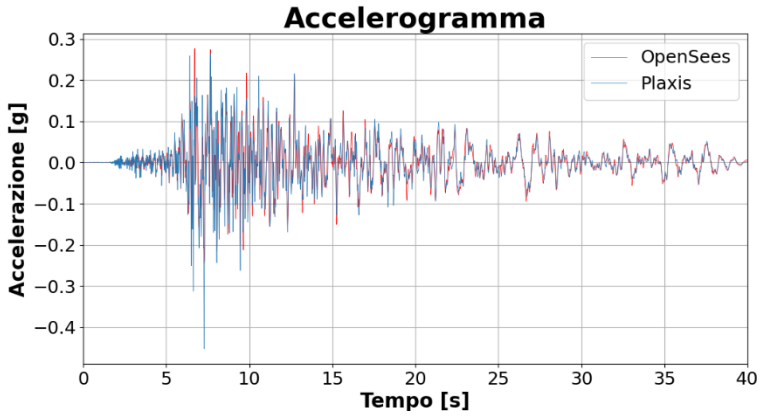
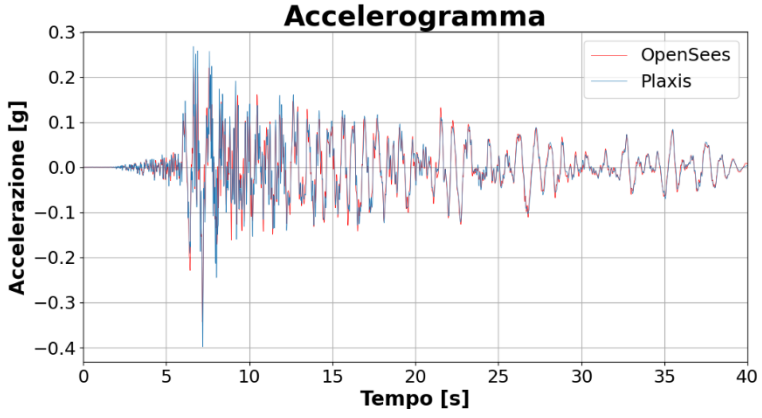
Y = 180 m	
Y = 195 m	

Fig. 97 Sezione X = 1250.0 m, spostamenti orizzontali.

Punto	Accelerazione Orizzontale
Y = 0 m	

Y = 20 m	
Y = 40 m	
Y = 60 m	

Y = 80 m	
Y = 100 m	
Y = 120 m	

Y = 140 m	
Y = 160 m	
Y = 180 m	

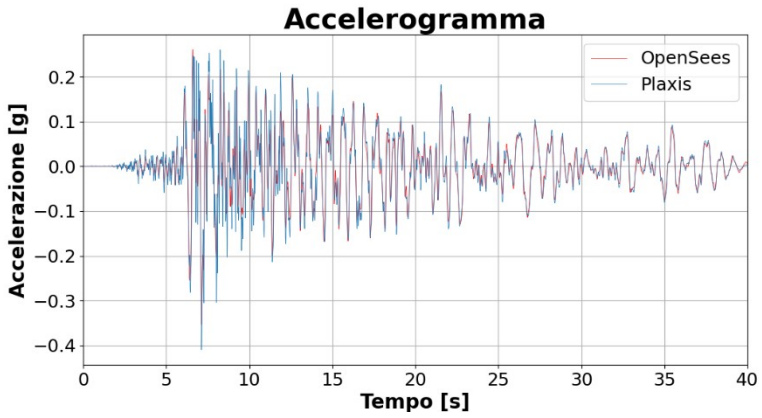
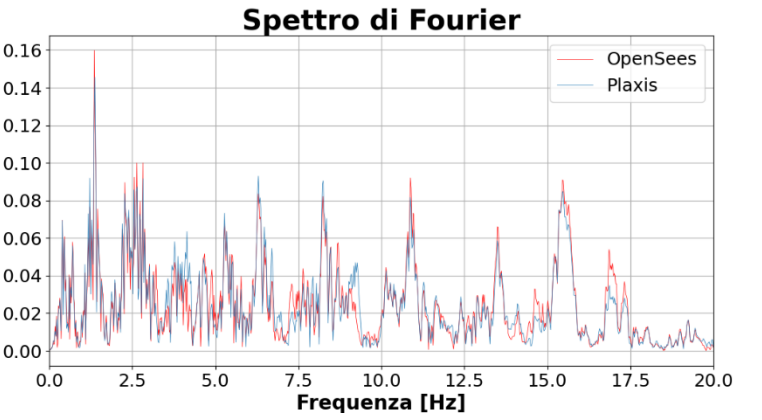
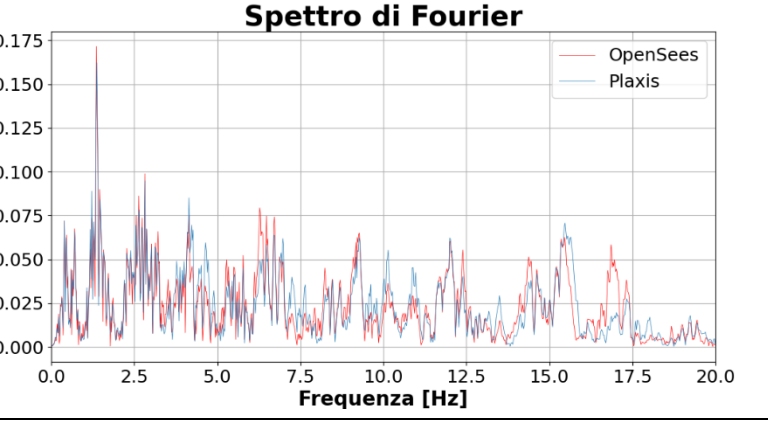
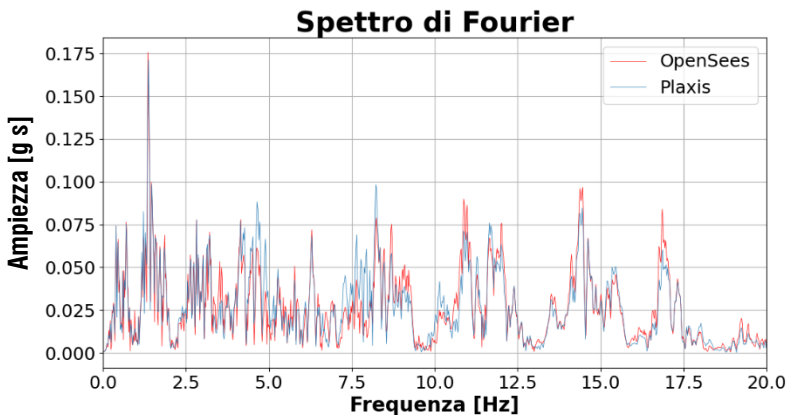
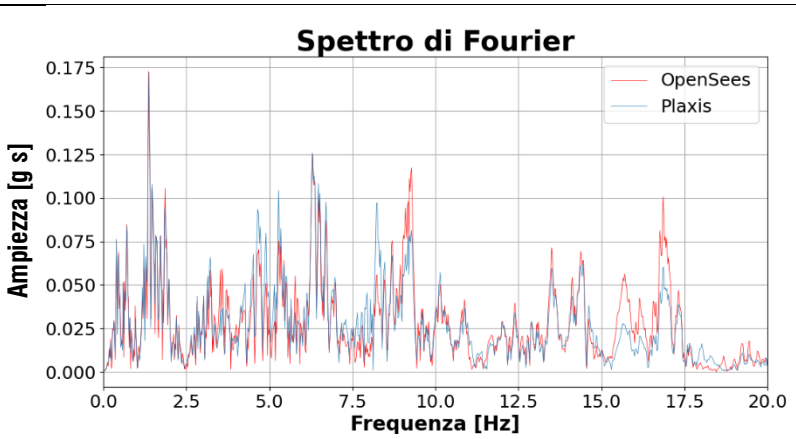
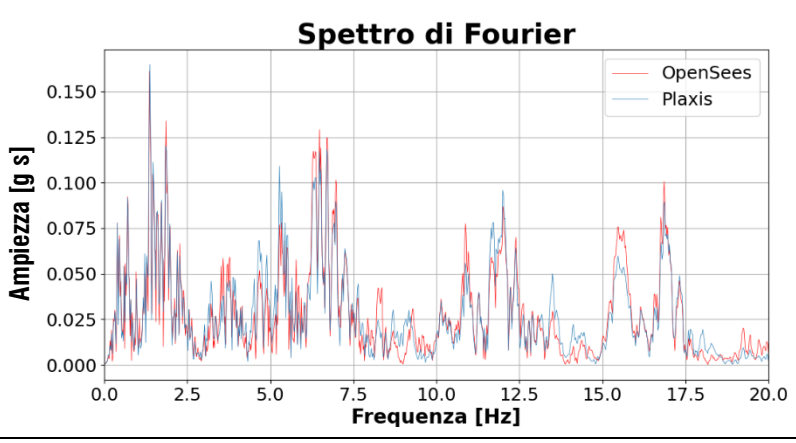
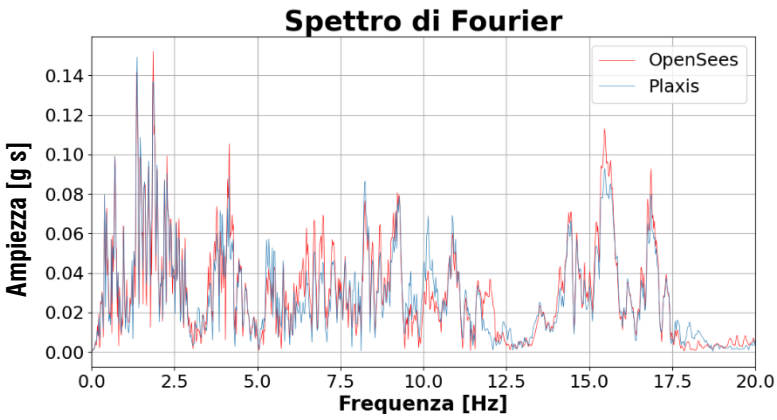
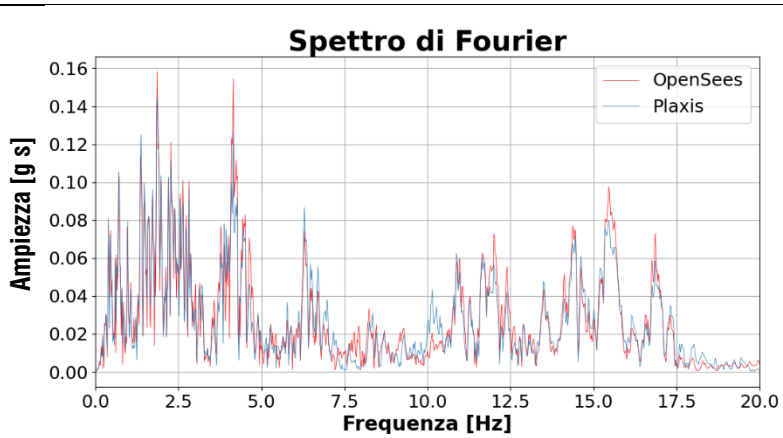
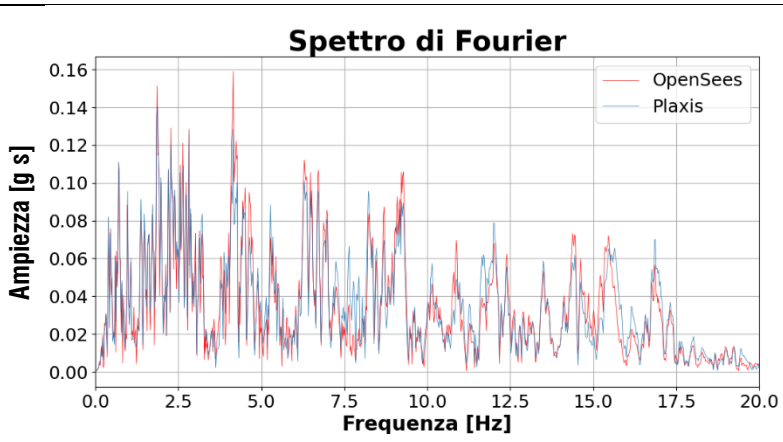
Y = 195 m	

Fig. 98 Sezione X = 1250.0 m, accelerazioni orizzontali.

Punto	Spettro di Fourier
Y = 0 m	
Y = 20 m	

Y = 40 m	
Y = 60 m	
Y = 80 m	

Y = 100 m	
Y = 120 m	
Y = 140 m	

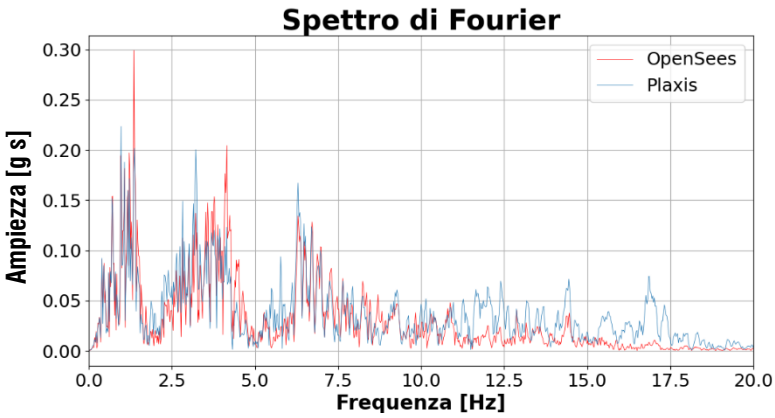
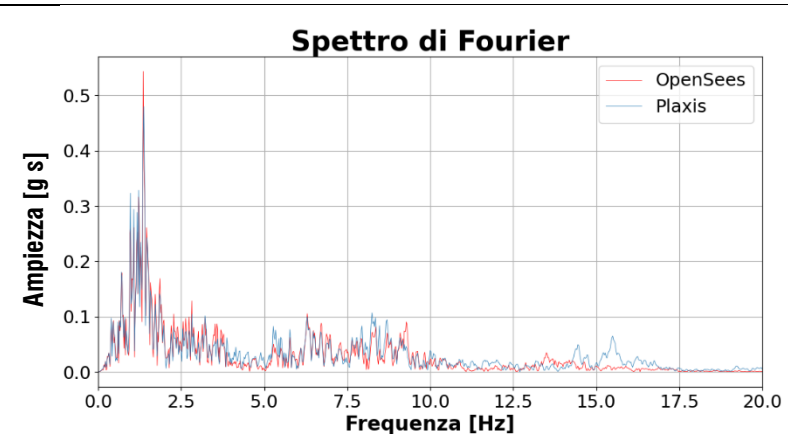
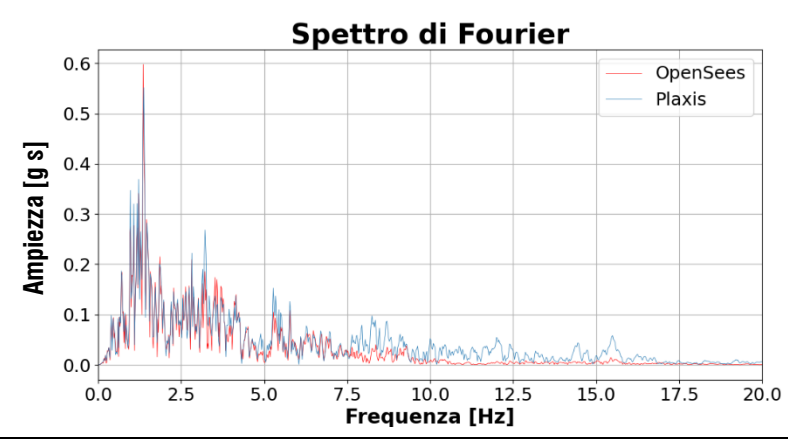
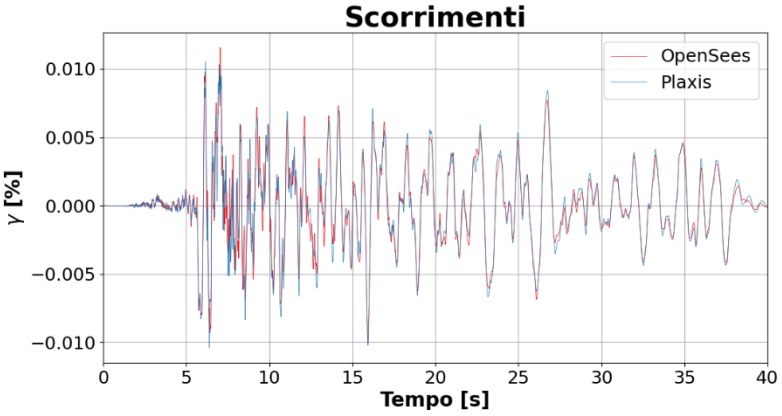
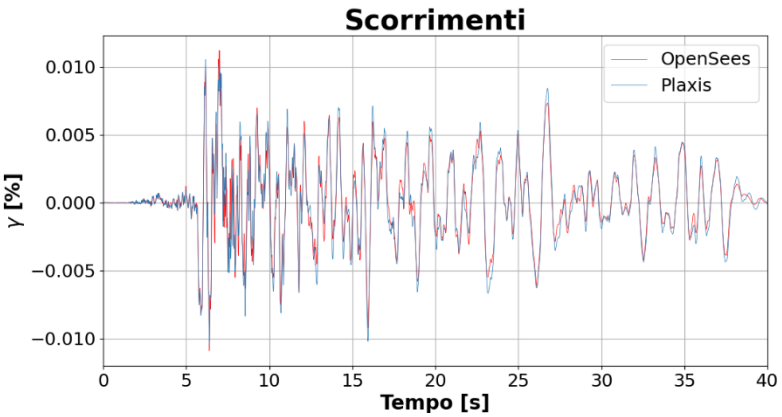
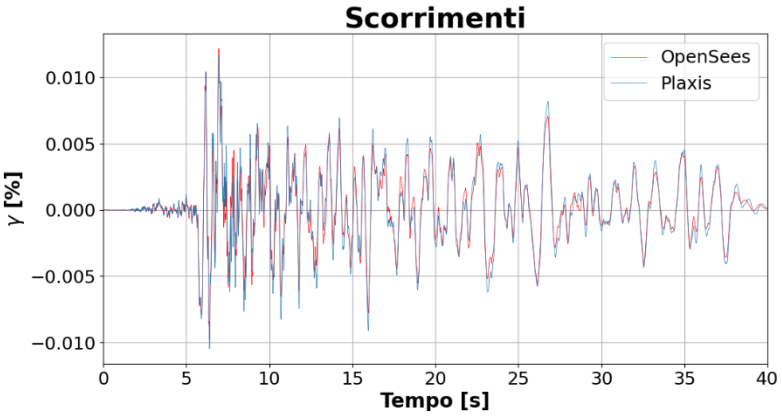
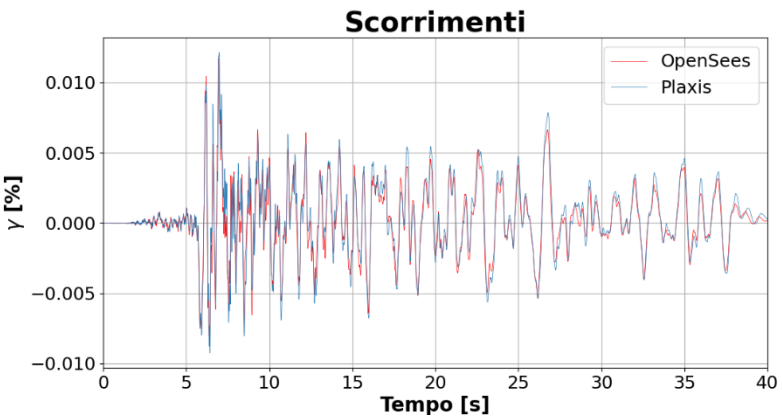
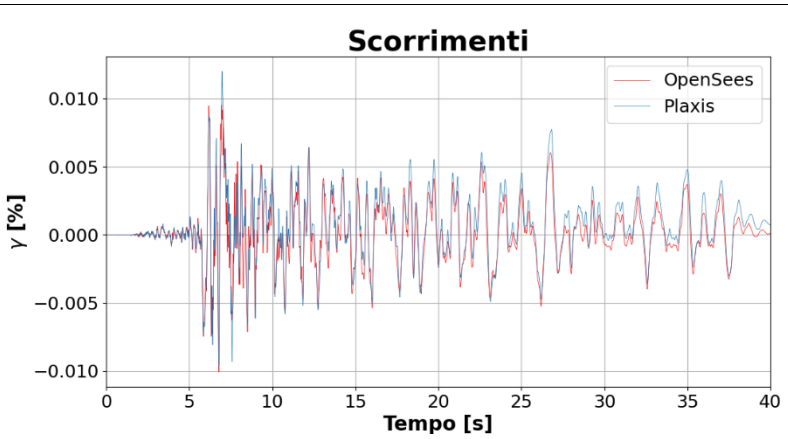
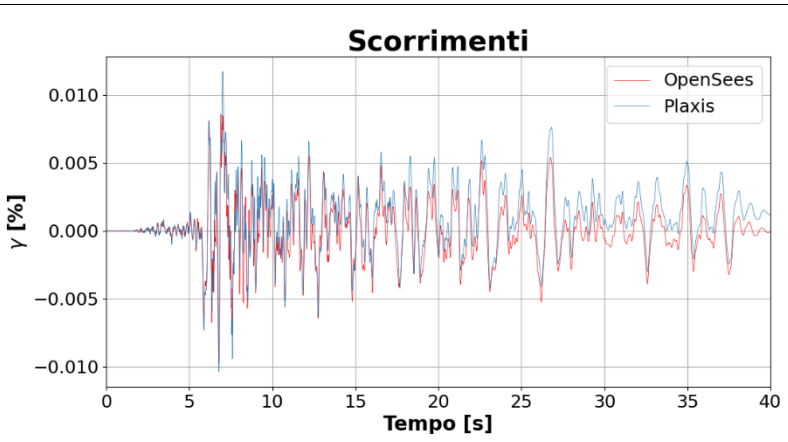
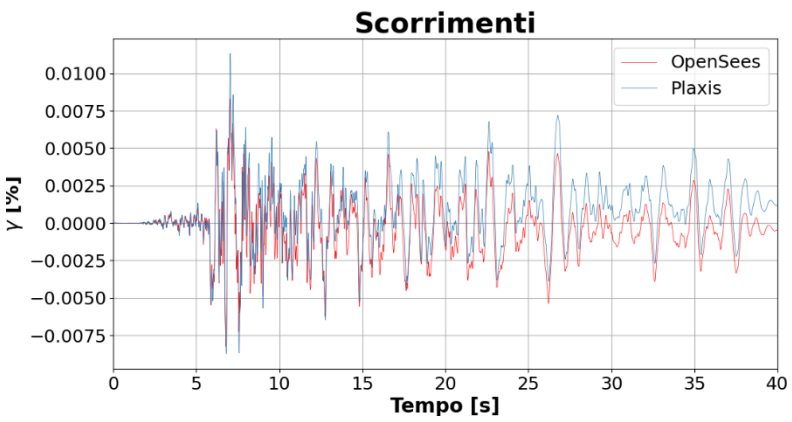
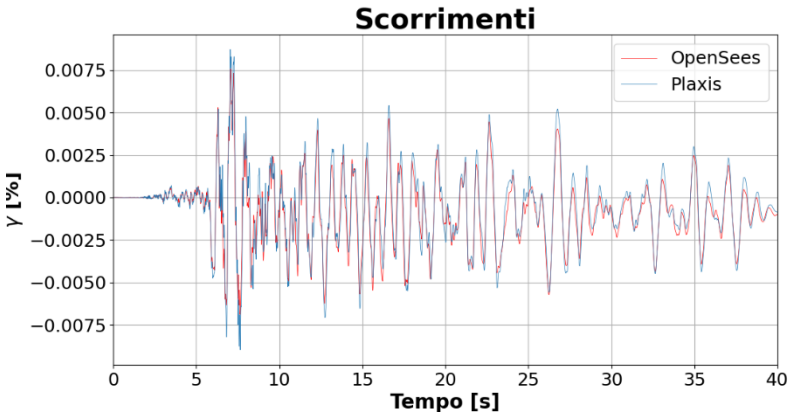
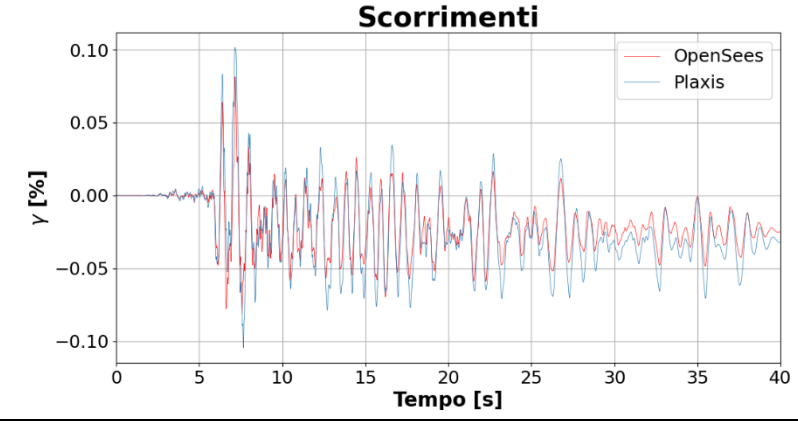
Y = 160 m	
Y = 180 m	
Y = 195 m	

Fig. 99 Sezione X = 1250.0 m, spettri di Fourier.

Punto	Risposta meccanica
Y = 0 m	
Y = 20 m	
Y = 40 m	

Y = 60 m	
Y = 80 m	
Y = 100 m	

Y = 120 m	
Y = 140 m	
Y = 160 m	

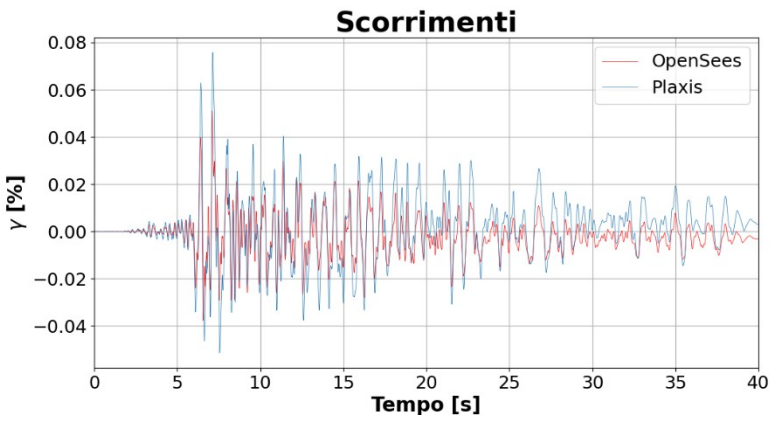
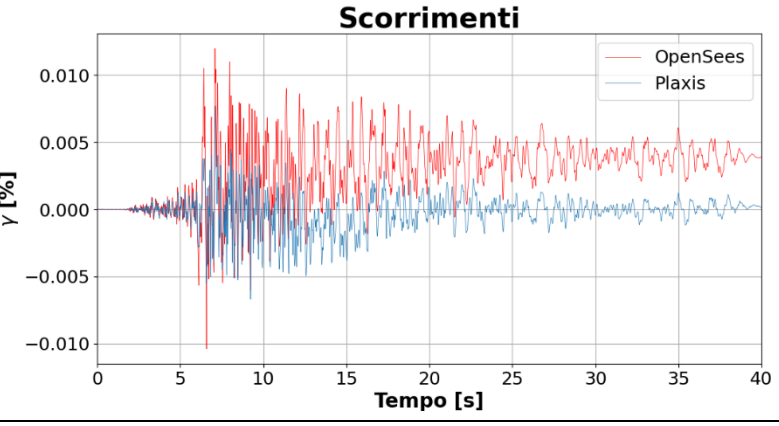
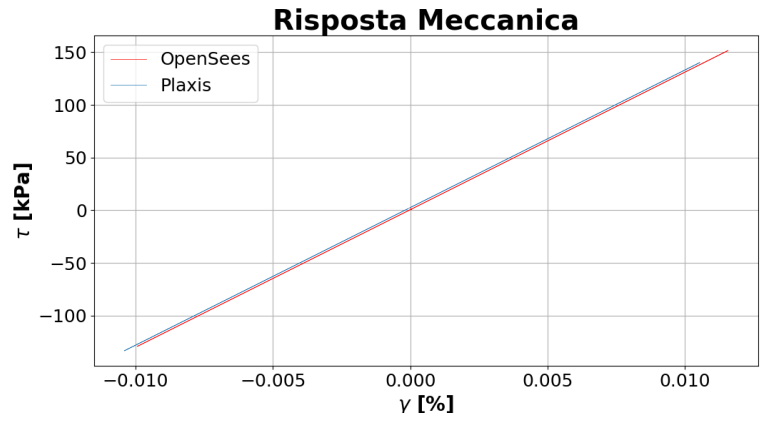
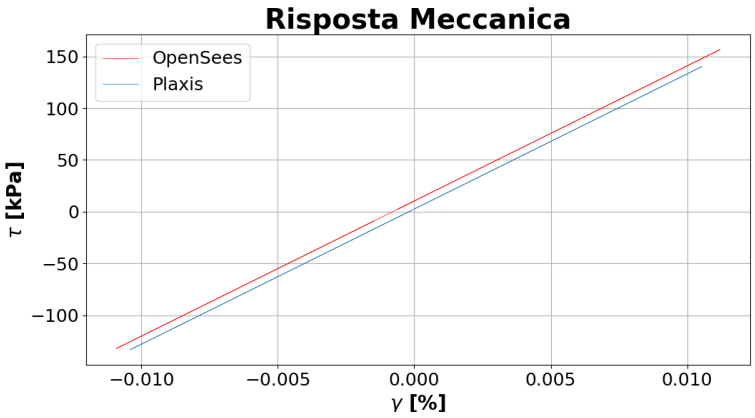
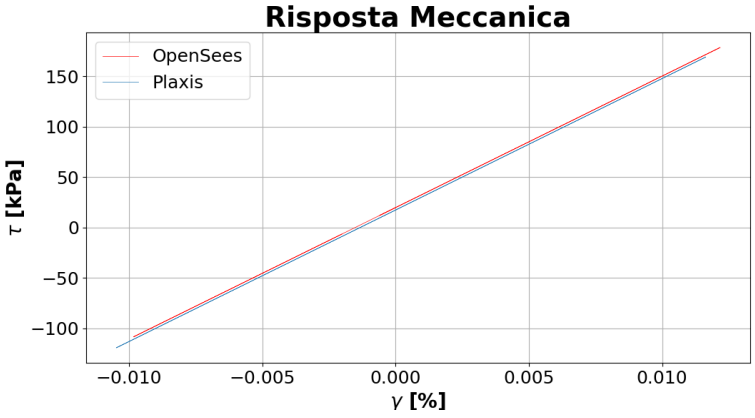
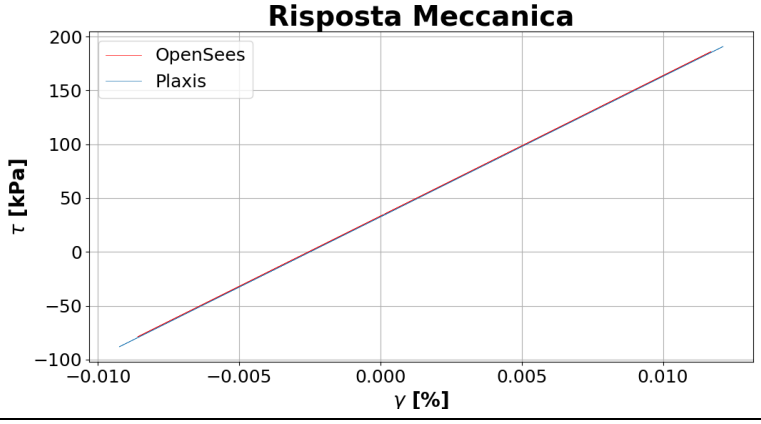
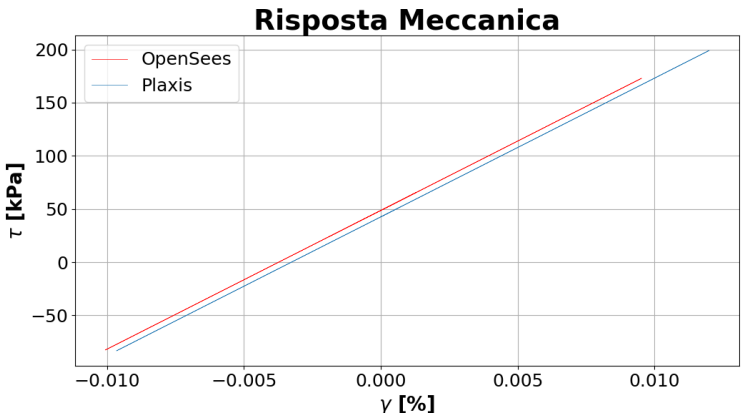
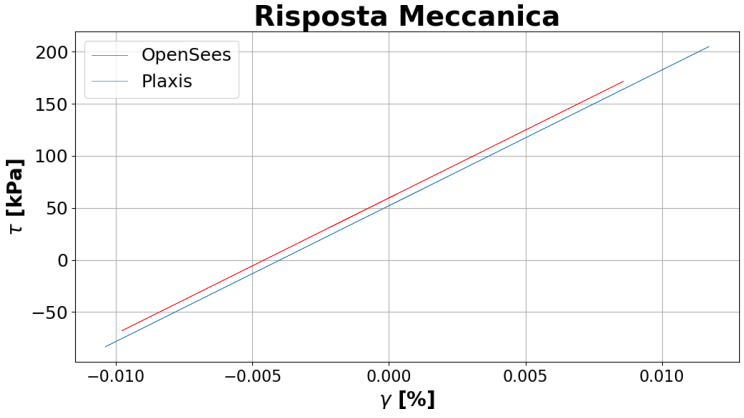
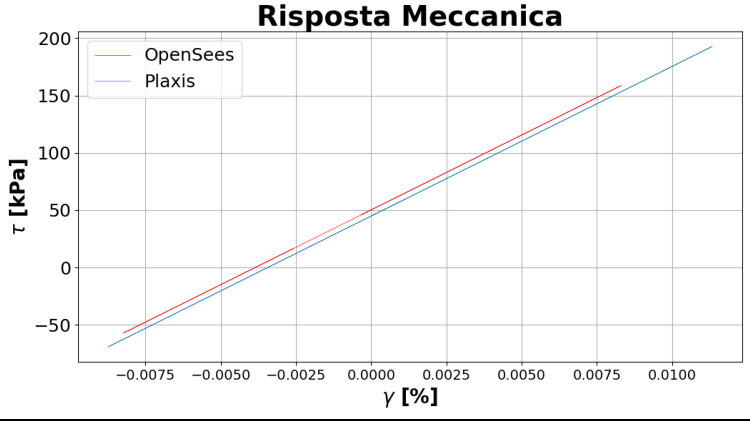
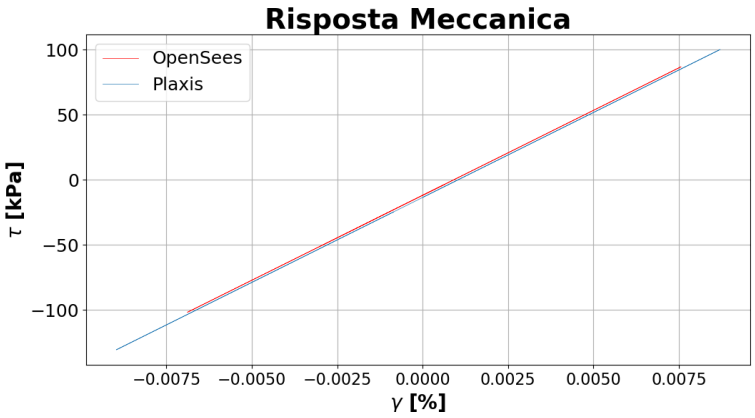
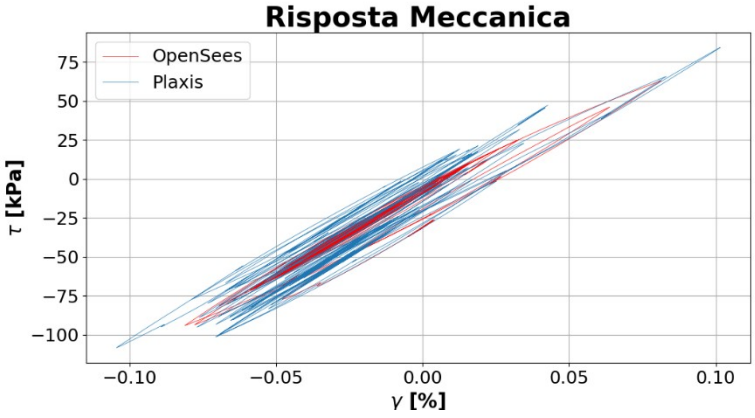
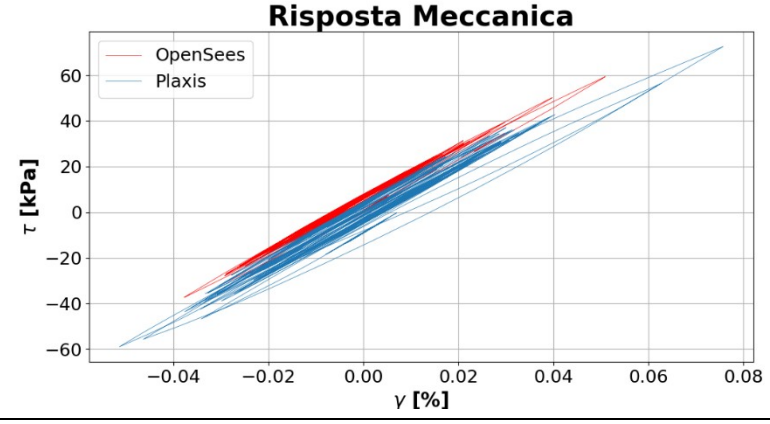
Y = 180 m	
Y = 195 m	

Fig. 100 Sezione X = 1250.0 m, scorrimenti orizzontali.

Punto	Risposta meccanica
Y = 0 m	

Y = 20 m	
Y = 40 m	
Y = 60 m	

Y = 80 m	
Y = 100 m	
Y = 120 m	

Y = 140 m	
Y = 160 m	
Y = 180 m	

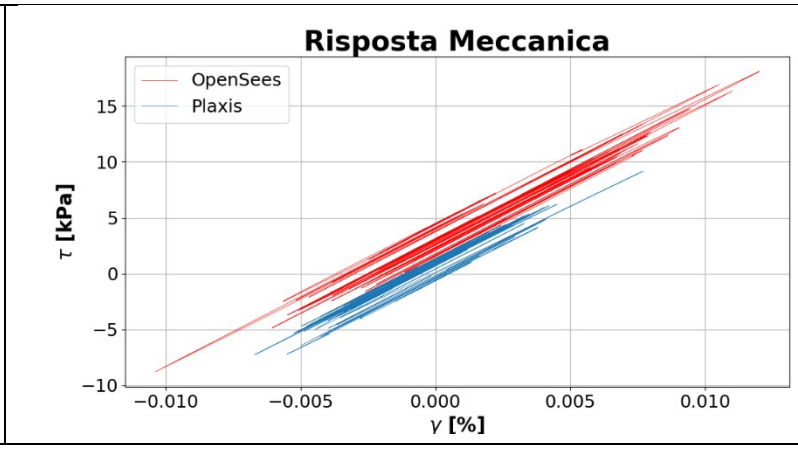
Y = 195 m	

Fig. 101 Sezione X = 1250.0 m, risposta meccanica.

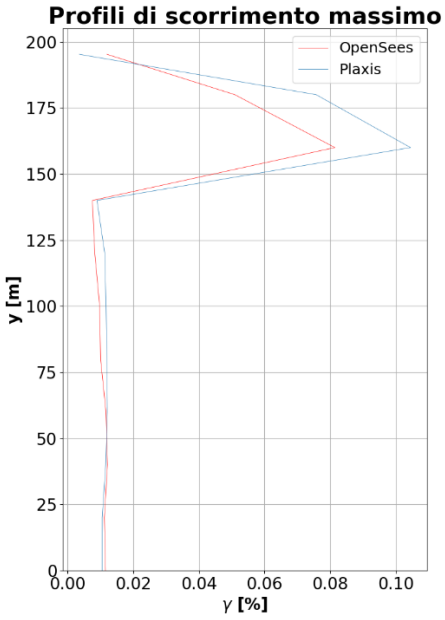
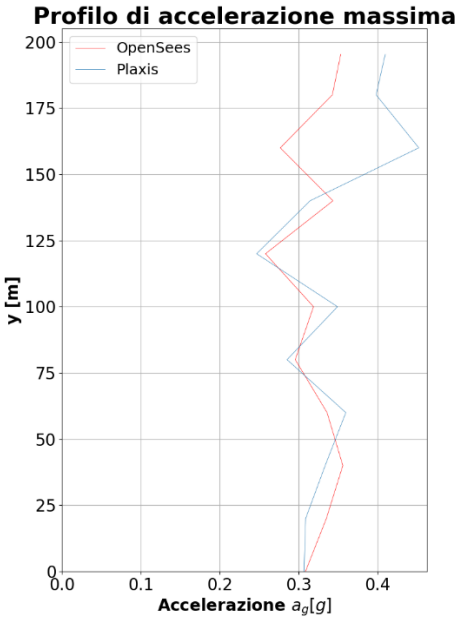
Profilo degli scorrimenti massimi	Profilo delle accelerazioni massime
	

Fig. 102 Sezione X = 1250.0 m, profili con la profondità.

PROFILI ACCELERAZIONI MASSIME IN SUPERFICIE

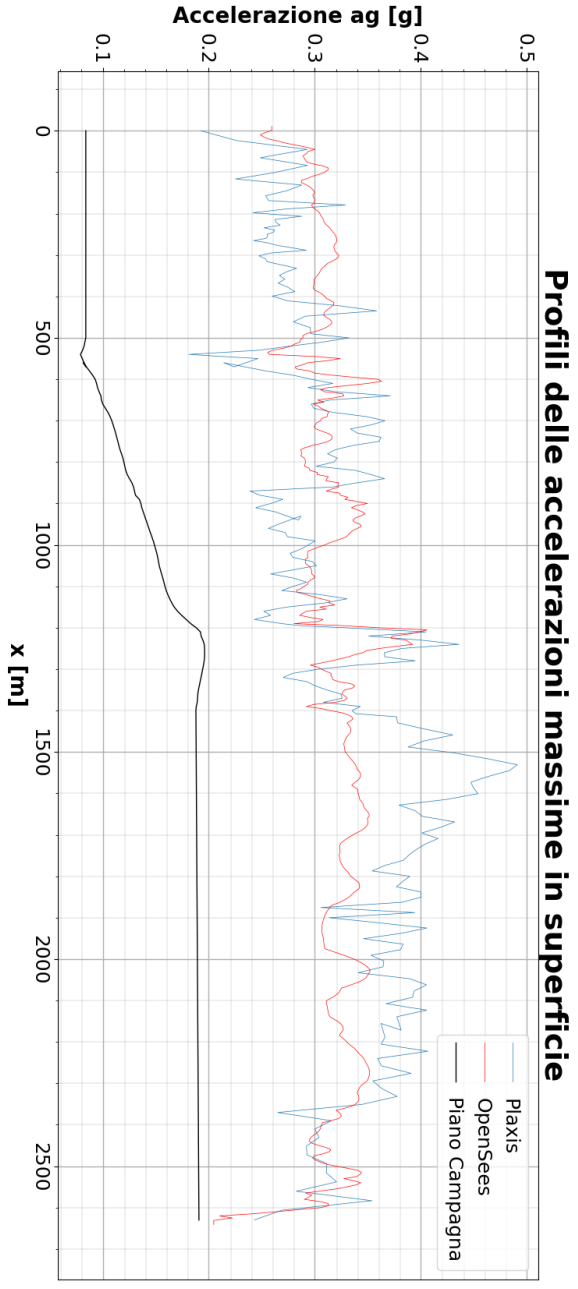


Fig. 103 Profilo longitudinale delle accelerazioni massime in superficie ottenuti dalle analisi visco-elasto-plastiche.

APPENDICE 6

APPENDICE 6 - py code 1 – dipendenze del codice

```
import os
import openseespy.opensees as ops
import Gmsh2opensees as g2o
import numpy as nup
import Gmsh
import math as mm
import time as tt
```

APPENDICE 6 - py code 2 – class Model

```
class Model(object):
    global nodo_mesh
    global PhysGr
    global nop
    PhysGr = g2o.get_physical_groups_map(Gmsh.model)
    print(g2o.get_physical_groups_map(Gmsh.model))

    def __init__(self,uniqueVector = [],nomeEl=str()):
        self.uniqueVector = uniqueVector
        self.nomeEl = nomeEl
        self.PhysGr = PhysGr
        print('Model activated')
```

APPENDICE 6 - py code 3 – class App

```
class App(Model):
    ops.reactions()
    def __init__(self):
        Model.__init__(self)
```

APPENDICE 6 - py code 4 – funzione trova nodo

```
def trova_nodo(dicto,value):
    print('loading....')
    output = []
    for nproc, subdict in dicto.items():
        if subdict == {}:
            break
        else:
            for el_num,nod_list in subdict.items():
                if value in nod_list:
                    output.append(nproc)
    output = nup.unique(nup.array(output,dtype = int))
    return output
```

APPENDICE 6 - py code 5 – funzione trova elemento

```
def trova_elemento(dicto,value):
```

```
            print('loading....')
            for nproc, subdict in dicto.items():
                for el_num,nod_list in subdict.items():
                    if value == el_num:
                        return nproc
```

APPENDICE 6 - py code 6 – funzione setGeoElement

```
def setGeoElement(self,nomeEl):
        self.nomeEl = nomeEl
        print('setGeoElement: geometry element inserted')
```

APPENDICE 6 - py code 7 – funzione getElementssVectors

```
def getElementsVectors(self):
        elementTag=[]
        nodeTag=[]
        entities=[]
        entTag = []
        elementName=[]
        elementNnode=[]
        Tags = []
        LNT = 0
        TagPh = PhysGr[self.nomeEl][1]
        DimPh = PhysGr[self.nomeEl][0]
        if DimPh ==3:
            elementTag, nodeTag, elementName, elementNnode =
g2o.get_elements_and_nodes_in_physi-
cal_group(self.nomeEl,Gmsh.model)
            LNT = len(nodeTag)
            return (elementTag, nodeTag, elementName, element-
Nnode,LNT)
        else:
            entities = Gmsh.model.getEntitiesForPhysi-
calGroup(DimPh, TagPh)
            Tags = Gmsh.model.mesh.getElements(DimPh, enti-
ties[0])
            elementTag = Tags[1][0]
            nodeTag = Tags[2][0]
            entTag, elementTag, nodeTag = Gmsh.model.mesh.getEl-
ements(DimPh, entities[0])
            return (elementTag, nodeTag)
```

APPENDICE 6 - py code 8 – funzione mDict

```
def mDict(self):
        '''vale solo per volumi = 'Solids' '''
        global proc_dict
        print('funzione mDict')
        print(f'nop = {nop}')
        m = Model()
```

```
        m.setGeoElement('Solid')
        elementTag, nodeTag, elementName, elementNnode, LNT =
m.getElementsVectors()
        eleNode = {}
        eleNode = dict(zip(elementTag, nodeTag))

        numero_nodi = len(nodeTag)
        print(f'numero dei nodi:  {numero_nodi}')

        numero_el = len(elementTag)
        print(f'numero di elementi: {numero_el}')


        numero_el_np = 0
        if numero_el%(nop-1)==0:
            numero_el_np = int(numero_el/(nop-1))
        else:
            numero_el_np = int(numero_el/(nop-1)) + 1

        print(f'il numero di el per np è: {numero_el_np}')

        proc_dict = {}
        for i in range(1,nop):
            proc_dict[i] = 0

        i = 1
        j = 0
        k = 0
        jj = 0
        proc_list = {}
        while i < nop:
            proc_dict[i] = {}
            while j < numero_el:
                jj = elementTag[j]
                if k < numero_el_np:
                    proc_dict[i][jj] = eleNode[jj]
                    k = k + 1
                    j = j + 1
                else:
                    k = 0
                    break
            i = i + 1
        return proc_dict
```

APPENDICE 6 - py code 9 – funzione setUniqueVector

```
def setUniqueVector(self,uniqueVector):
        self.uniqueVector = uniqueVector
```

APPENDICE 6 - py code 10 – funzione getUniqueVector

```
def getUniqueVector(self):
      return self.uniqueVector
```

APPENDICE 6 - py code 11 – Funzione mRemoveXY

```
def mRemoveXY(self):
        print('funzione subReacForce')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()
        type_el = str(input('what geoElement do you want to un-
constrain'))
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 4
        reacV1={}
        mem = 0

        for i in f:

            ops.remove('sp',int(i[0]),1)
            ops.remove('sp',int(i[0]),2)


            TCL=open("cubotto.TCL",'a')
            TCL.write(f'remove sp {int(i[0])} 1\nremove sp
{int(i[0])} 2\n')
            TCL.close()

        ##########################

        print('\nEdge\n')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
```

```
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 3
        reacV2={}
        mem = 0

        for i in f:
            ops.remove('sp',int(i[0]),1)
            ops.remove('sp',int(i[0]),2)

            TCL = open("cubotto.TCL",'a')
            TCL.write(f'remove sp {int(i[0])} 1\nremove sp
{int(i[0])} 2\n')
            TCL.close()
```

APPENDICE 6 - py code 12 – Funzione mTieNodesX

```
def mTieNodesX():
        print('funzione mTieNodes')
        m = Model()

        type_el_master1 = str(input('insert master plane geoEle-
ment'))
        b = m.setGeoElement(type_el_master1)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0]) ############ CORNER
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()

        type_el_slave1 = str(input('insert slave plane geoEle-
ment'))
        b = m.setGeoElement(type_el_slave1)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
```

```
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        g = m.nodeNumCoordVector()
        i = 0
        j = 0
        k = 0
        DOF = [1, 2, 3]
        eqDofDic = {}
        for i in f:
            for j in g:
                z_master = i[1][2]
                z_slave = j[1][2]
                if z_master == z_slave:
                    eqDofDic[i[0]] = j[0]
                    ops.equalDOF(int(i[0]), int(j[0]), *DOF)

################################################

                    TCL = open('cubotto.TCL', 'a')
                    TCL.write(f'set DOF1 {DOF[0]}\n')
                    TCL.close()

        b = m.setGeoElement(type_el_master1)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        b = m.setGeoElement(type_el_slave1)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[1])
```

```
        print(len(c[1]))
        print(len(c[1][0]))
        l = len(c[1])
        ll = len(c[1][0])
        d = m.listOfTagsij(l, ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        g = m.nodeNumCoordVector()
        i = 0
        j = 0
        k = 0
        DOF = [1, 2, 3]
        eqDofDic = {}
        for i in f:
            for j in g:
                z_master = i[1][2]
                z_slave = j[1][2]
                if z_master == z_slave:
                    eqDofDic[i[0]] = j[0]
                    ops.equalDOF(int(i[0]), int(j[0]), *DOF)

##################################################
                    TCL = open('cubotto.TCL', 'a')
                    TCL.write(f'set DOF1 {DOF[0]}\n')
                    TCL.write(f'equalDOF {int(i[0])} {int(j[0])}
$DOF1\n')
```

APPENDICE 6 - py code 13 – funzione nodeNumCoordVector

```
def nodeNumCoordVector(self):
        opsVector = []
        collect = []
        coord_l=[]
        num = 0
        x=0
        y=0
        z=0
        j=0
        for j in self.uniqueVector:
            x = Gmsh.model.mesh.getNode(int(j))[0][0]
            y = Gmsh.model.mesh.getNode(int(j))[0][1]
            z = Gmsh.model.mesh.getNode(int(j))[0][2]
            coord_l=[x,y,z]
            num = int(j)
            collect = [num,coord_l]
            opsVector.append(collect)
        return opsVector
```

APPENDICE 6 - py code 14 – funzione listOfTagsij

```
def listOfTagsij(self,l,ll):
        nodeTot = []
        nodeTotU = []
        self.l = l
        self.ll = ll
        for i in range(0, l):
            for j in range(0, ll):
                nodeTot.append(self.uniqueVector[i][j])
        nodeTotU=nup.unique(nup.array(nodeTot, dtype=int).re-
shape(-1))
        return nodeTotU
```

APPENDICE 6 - py code 15 – funzione makeUnique

```
def makeUnique(self):
        getUnique= nup.unique(nup.array(self.uniqueVector,
dtype=int).reshape(-1))
        print('make Unique ok')
        return getUnique
```

APPENDICE 6 - py code 16 – funzione nodeCornEdge

```
def nodeCornEdge(self,type):
        print('insert type = 8-nodi o 20-nodi')
        self.type = type
        j=0
        i=0
        k=0
        n = 0
        nodeNum = []
        nodeVecCorn = []
        nodeVecEdge = []
        mem = []
        elemento = str(type)
        if elemento == '20-nodi':
            p = 1
            l = int(len(self.uniqueVector[p]))
            for i in range(0, l):
                for j in range(0, 20):
                    nodeNum = self.uniqueVector[p][i][j]
                    mem.append(nodeNum)
                    if j == 7:
                        nodeVecCorn.append(mem)
                        mem = []
                    elif j == 19:
                        nodeVecEdge.append(mem)
                        mem = []
        else:
            len1 = len(self.uniqueVector[1][0])
```

```
            for i in range(0, len1):
                n = self.uniqueVector[1][0][i]
                mem.append(n)
                j = j + 1
                if j == 4:
                    nodeVecCorn.append(mem)
                    mem = []
                elif j == 8:
                    nodeVecEdge.append(mem)
                    j = 0
                    mem = []
        return (nodeVecCorn,nodeVecEdge)
```

APPENDICE 6 - py code 17 – funzione sendOpsNodes

```
def sendOpsNodes(self,dim,ndf):
        print('funzione sendOpsNodes')
        self.dim = dim
        self.ndf = ndf
        m = Model()
        proc_dict = m.mDict()
        ops.model("basicBuilder", "-ndm", dim, "-ndf", ndf)

        py = open("cubotto_exe.py","a")
        py.write(f'ops.model("basicBuilder","-ndm",{dim},"-
ndf",{ndf})\n')
        py.close()

        TCL=open("cubotto.TCL","a")
        TCL.write(f'model BasicBuilder -ndm {dim} -ndf {ndf}\n')
        TCL.close()
        l = len(self.uniqueVector)
        for i in range(0,l):
            coord = [self.uniqueVector[i][1][0],self.uniqueVec-
tor[i][1][1],self.uniqueVector[i][1][2]]
            ops.node(int(self.uniqueVector[i][0]),*coord)

            procId = trova_nodo(proc_dict,self.uniqueVec-
tor[i][0])
            for proc in procId:
                py = open("cubotto_exe.py","a")
                py.write(f'if pid == {proc}:\n')
                py.close()

                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == {proc}'+' }'+' {\n')
                TCL.close()
                py = open("cubotto_exe.py","a")
```

```
                py.write(f"    ops.node({int(self.uniqueVec-
tor[i][0])}, {coord[0]}, {coord[1]}, {coord[2]})\n")
                py.close()

                TCL=open("cubotto.TCL","a")
                TCL.write(f'    node {int(self.uniqueVec-
tor[i][0])} {coord[0]} {coord[1]} {coord[2]}\n')
                TCL.write('}\n')
                TCL.close()
```

APPENDICE 6 - py code 18 – funzione sendOpsFixes

```
def sendOpsFixes(self,dim,ndf,x,y,z,p):
        print('funzione sendOpsFixes')
        self.dim = int(dim)
        self.ndf = int(ndf)
        self.x = int(x)
        self.y = int(y)
        self.z = int(z)
        self.p = int(p)
        m = Model()
        proc_dict = m.mDict()
        if ndf == 4:
            ops.model("basicBuilder", "-ndm", dim, "-ndf", ndf)

            py = open("cubotto_exe.py","a")
            py.write(f'ops.model("basicBuilder","-ndm",{dim},"-
ndf",{ndf})\n')
            py.close()

            TCL=open("cubotto.TCL","a")
            TCL.write(f'model BasicBuilder -ndm {dim} -ndf
{ndf}\n')
            TCL.close()

            l = len(self.uniqueVector)
            for i in range(0, l):
                ops.fix(int(self.uniqueVector[i][0]), x, y, z,
p)
                print(f' ndf = 4, nodo: {int(self.uniqueVec-
tor[i][0])}, {x}, {y}, {z}, {p}')

                coord = [self.uniqueVector[i][1][0],self.unique-
Vector[i][1][1],self.uniqueVector[i][1][2]]
                procId = trova_nodo(proc_dict,self.uniqueVec-
tor[i][0])

                j = 0
```

```
                for proc in procId:

                    TCL=open("cubotto.TCL","a")
                    TCL.write("if {"+f' $pid == {proc}'+' }'+'
{\n')
                    #TCL.write(f'    remove sp {int(self.unique-
Vector[i][0])} 1\n    remove sp {int(self.uniqueVector[i][0])}
2\n    remove sp {int(self.uniqueVector[i][0])} 3\n    remove sp
{int(self.uniqueVector[i][0])} 4\n')
                    TCL.write(f'    fix {int(self.uniqueVec-
tor[i][0])} {x} {y} {z} {p}\n')
                    TCL.write('}\n')
                    TCL.close()

                    if j == 0:

                        TCL=open("cubotto.TCL","a")
                        TCL.write("if {"+f' $pid == 0'+' }'+'
{\n')
                        #TCL.write(f'    remove sp
{int(self.uniqueVector[i][0])} 1\n    remove sp
{int(self.uniqueVector[i][0])} 2\n    remove sp
{int(self.uniqueVector[i][0])} 3\n    remove sp
{int(self.uniqueVector[i][0])} 4\n')
                        TCL.write(f'    fix {int(self.uniqueVec-
tor[i][0])} {x} {y} {z} {p}\n')
                        TCL.write('}\n')
                        TCL.close()

                        j = j + 1

        else:
            ops.model("basicBuilder", "-ndm", dim, "-ndf", ndf)

            py = open("cubotto_exe.py","a")
            py.write(f'ops.model("basicBuilder","-ndm",{dim},"-
ndf",{ndf})\n')
            py.close()


            TCL=open("cubotto.TCL","a")
            TCL.write(f'model BasicBuilder -ndm {dim} -ndf
{ndf}\n')
            TCL.close()

            l = len(self.uniqueVector)
            for i in range(0, l):
```

```
                # ops.remove('sp', int(self.uniqueVec-
tor[i][0]),1)
                # ops.remove('sp', int(self.uniqueVec-
tor[i][0]),2)
                # ops.remove('sp', int(self.uniqueVec-
tor[i][0]),3)
                ops.fix(int(self.uniqueVector[i][0]), x, y, z,
p)
                print(f' ndf = 3, nodo: {int(self.uniqueVec-
tor[i][0])}, {x}, {y}, {z}, {p}')

                procId = trova_nodo(proc_dict,self.uniqueVec-
tor[i][0])
                coord = [self.uniqueVector[i][1][0],self.unique-
Vector[i][1][1],self.uniqueVector[i][1][2]]
                j = 0
                for proc in procId:

                    TCL=open("cubotto.TCL","a")
                    TCL.write("if {"+f' $pid == {proc}'+' }'+'
{\n')
                    #TCL.write(f'    remove sp {int(self.unique-
Vector[i][0])} 1\n    remove sp {int(self.uniqueVector[i][0])}
2\n    remove sp {int(self.uniqueVector[i][0])} 3\n')
                    TCL.write(f'    fix {int(self.uniqueVec-
tor[i][0])} {x} {y} {z}\n')
                    TCL.write('}\n')
                    TCL.close()

                    if j == 0:
                        TCL=open("cubotto.TCL","a")
                        TCL.write("if {"+f' $pid == 0'+' }'+'
{\n')
                        #TCL.write(f'    remove sp
{int(self.uniqueVector[i][0])} 1\n    remove sp
{int(self.uniqueVector[i][0])} 2\n    remove sp
{int(self.uniqueVector[i][0])} 3\n')
                        TCL.write(f'    fix {int(self.uniqueVec-
tor[i][0])} {x} {y} {z}\n')
                        TCL.write('}\n')
                        TCL.close()
                        j = j+1
```

APPENDICE 6 - py code 19 – Classe App

```
class App(Model):
    ops.reactions()
    def __init__(self):
```

```
        Model.__init__(self)
```

APPENDICE 6 - py code 20 – Funzione SubReacForce

```
def subReacForce(self):
        print('funzione subReacForce')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()
        proc_dict = m.mDict()

        type_el = str(input('what geoElement do you want to fix
substituting'))
        dim = int(input('what dimension is the force ?: '))
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 4
        reacV1={}
        mem = 0

        k = 0
        while k < nop:
            file=open(f'nodeReaction_{k}.out','w')
            file.write('reactions\n')
            file.close()
            k = k + 1

        for i in f:

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                if dim == 1:
                    file=open(f'nodeReaction_{proc}.out','a')
                    file.write(f'    load {int(i[0])} [nodeReac-
tion {int(i[0])} {dim}] 0.0 0.0 0.0\n')
```

```
                    file.close()
                elif dim == 2:
                    file=open(f'nodeReaction_{proc}.out','a')
                    file.write(f'    load {int(i[0])} 0.0
[nodeReaction {int(i[0])} {dim}] 0.0 0.0\n')
                    file.close()
                elif dim == 3:
                    file=open(f'nodeReaction_{proc}.out','a')
                    file.write(f'    load {int(i[0])} 0.0 0.0
[nodeReaction {int(i[0])} {dim}] 0.0\n')
                    file.close()

        print('\nEdge\n')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 3
        reacV2={}
        mem = 0


        for i in f:

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                if dim == 1:
                    file=open(f'nodeReaction_{proc}.out','a')
                    file.write(f'    load {int(i[0])} [nodeReac-
tion {int(i[0])} {dim}] 0.0 0.0\n')
                    file.close()
                elif dim == 2:
                    file=open(f'nodeReaction_{proc}.out','a')
                    file.write(f'    load {int(i[0])} 0.0
[nodeReaction {int(i[0])} {dim}] 0.0\n')
                    file.close()
                elif dim == 3:
                    file=open(f'nodeReaction_{proc}.out','a')
```

```
                    file.write(f'    load {int(i[0])} 0.0 0.0
[nodeReaction {int(i[0])} {dim}]\n')
                    file.close()
        k = 0
        while k < nop:
            file=open(f'cubotto.TCL','a')
            file.write("if {"+f' $pid == {k}'+' }'+' {\n')
            file.write(f'source nodeReaction_{k}.out\n')
            file.write('}\n')
            file.close()
            k = k + 1
```

APPENDICE 6 - py code 21 – Funzioni mfixSpx mfixSpy mfixSpz

```
def mfixSpX(self):
        print('funzione subReacForce')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()

        proc_dict = m.mDict()

        type_el = str(input('what geoElement do you want to sp:
'))
        type_f_el=str(input('what Element 8-nodi o 20-nodi: '))
        dim = 1
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge(type_f_el)
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 4
        reacV1={}
        mem = 0

        k = 1
        while k < nop:
            file=open(f'sp_x_{k}.out','w')
            file.close()
```

```
            k = k + 1

        for i in f:

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                    file=open(f'sp_{proc}.out','a')
                    file.write(f'    sp {int(i[0])} {dim} [expr
0.0 + [nodeDisp {int(i[0])} {dim}]]\n')
                    file.close()


        print('\nEdge\n')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 3
        reacV2={}
        mem = 0


        for i in f:

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                    file=open(f'sp_{proc}.out','a')
                    file.write(f'    sp {int(i[0])} {dim} [expr
0.0 + [nodeDisp {int(i[0])} {dim}]]\n')
                    file.close()

        k = 1
        while k < nop:
            file=open(f'cubotto.TCL','a')
            file.write("if {"+f' $pid == {k}'+' }'+' {\n')
            file.write(f'source sp_x_{k}.out\n')
            file.write('}\n')
            file.close()
```

```
            k = k + 1

    def mfixSpY(self):
        print('funzione subReacForce')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()

        proc_dict = m.mDict()

        type_el = str(input('what geoElement do you want to sp:
'))
        type_f_el=str(input('what Element 8-nodi o 20-nodi: '))
        dim = 2
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge(type_f_el)
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 4
        reacV1={}
        mem = 0

        k = 1
        while k < nop:
            file=open(f'sp_y_{k}.out','w')
            file.close()
            k = k + 1

        for i in f:

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                    file=open(f'sp_{proc}.out','a')
                    file.write(f'    sp {int(i[0])} {dim} [expr
0.0 + [nodeDisp {int(i[0])} {dim}]]\n')
                    file.close()
```

```
        print('\nEdge\n')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 3
        reacV2={}
        mem = 0


        for i in f:

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                    file=open(f'sp_{proc}.out','a')
                    file.write(f'    sp {int(i[0])} {dim} [expr
0.0 + [nodeDisp {int(i[0])} {dim}]]\n')
                    file.close()

        k = 1
        while k < nop:
            file=open(f'cubotto.TCL','a')
            file.write("if {"+f' $pid == {k}'+' }'+' {\n')
            file.write(f'source sp_y_{k}.out\n')
            file.write('}\n')
            file.close()
            k = k + 1

    def mfixSpZ(self):
        print('funzione subReacForce')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()

        proc_dict = m.mDict()

        type_el = str(input('what geoElement do you want to sp:
'))
        type_f_el=str(input('what Element 8-nodi o 20-nodi: '))
```

```
        dim = 3
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge(type_f_el)
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 4
        reacV1={}
        mem = 0

        k = 1
        while k < nop:
            file=open(f'sp_z_{k}.out','w')
            file.close()
            k = k + 1

        for i in f:

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                    file=open(f'sp_{proc}.out','a')
                    file.write(f'    sp {int(i[0])} {dim} [expr
0.0 + [nodeDisp {int(i[0])} {dim}]]\n')
                    file.close()

        print('\nEdge\n')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
```

```
        dim = 3
        ndf = 3
        reacV2={}
        mem = 0


        for i in f:

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                    file=open(f'sp_{proc}.out','a')
                    file.write(f'    sp {int(i[0])} {dim} [expr
0.0 + [nodeDisp {int(i[0])} {dim}]]\n')
                    file.close()

        k = 1
        while k < nop:
            file=open(f'cubotto.TCL','a')
            file.write("if {"+f' $pid == {k}'+' }'+' {\n')
            file.write(f'source sp_z_{k}.out\n')
            file.write('}\n')
            file.close()
            k = k + 1
```

APPENDICE 6 - py code 22 – Funzione mRemove

```
def mRemove(self):

        print('funzione remove')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()

        proc_dict = m.mDict()

        type_el = str(input('what geoElement do you want to re-
move fixes'))
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l, ll)
        m.setUniqueVector(d)
```

```
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 4
        reacV1 = {}
        mem = 0
        ops.model("basicBuilder", "-ndm", 3, "-ndf", 4)

        TCL = open("cubotto.TCL", "a")
        TCL.write("model BasicBuilder -ndm 3 -ndf 4\n")
        TCL.close()

        py = open("cubotto_exe.py", "a")
        py.write('ops.model(" basicBuilder", " - ndm", 3, " -
ndf ", 4)\n')
        py.close()
        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)
            reacV1[int(i[0])] = mem

            ops.remove('sp',int(i[0]))

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                py = open("cubotto_exe.py", "a")
                py.write(f'if pid == {proc}:\n')
                py.write(f"    ops.remove('sp',{int(i[0])},1)\n\
        ops.remove('sp',{int(i[0])},2)\n\
        ops.remove('sp',{int(i[0])},3)\n")

                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == {proc}'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 1\n    re-
move sp {int(i[0])} 2\n    remove sp {int(i[0])} 3\n')
                TCL.write('}\n')
                TCL.close()

                py.write(f'if pid == 0:\n')

                py.write(f'    ops.remove("sp",{int(i[0])},1)\n\
        ops.remove("sp",{int(i[0])},2)\n\
        ops.remove("sp",{int(i[0])},3)\n')
                py.close()
```

```
                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == 0'+' }'+' {\n')
                TCL.write(f'    remove sp {int(i[0])} 1\n    re-
move sp {int(i[0])} 2\n    remove sp {int(i[0])} 3\n')
                TCL.write('}\n')
                TCL.close()

        print('\nEdge\n')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 3
        reacV2={}
        mem = 0

        ops.model("basicBuilder", "-ndm", 3, "-ndf", 3)
        py = open("cubotto_exe.py", "a")
        py.write('ops.model(" basicBuilder", " - ndm", 3, " -
ndf ", 3)\n')
        py.close()

        TCL = open("cubotto.TCL", "a")
        TCL.write("model BasicBuilder -ndm 3 -ndf 3\n")
        TCL.close()

        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)

            reacV2[int(i[0])] = mem

            ops.remove('sp',int(i[0]))

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:


                py = open("cubotto_exe.py", "a")
                py.write(f'if pid == {proc}:\n')
                py.write(f"    ops.remove('sp',{int(i[0])},1)\n\
```

```
            ops.remove('sp',{int(i[0])},2)\n\
            ops.remove('sp',{int(i[0])},3)\n")



                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == {proc}'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 1\n    re-
move sp {int(i[0])} 2\n    remove sp {int(i[0])} 3\n')
                TCL.write('}\n')
                TCL.close()

                py.write('if pid == 0:\n')
                py.write(f'    ops.remove("sp",{int(i[0])},1)\n\
            ops.remove("sp",{int(i[0])},2)\n\
            ops.remove("sp",{int(i[0])},3)\n')

                py.close()

                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == 0'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 1\n    re-
move sp {int(i[0])} 2\n    remove sp {int(i[0])} 3\n')
                TCL.write('}\n')
                TCL.close()
```

APPENDICE 6 - py code 23 – Funzioni mRemoveX mRemoveY

```
def mRemoveX(self):

        print('funzione remove')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()

        proc_dict = m.mDict()

        type_el = str(input('what geoElement do you want to re-
move fixes'))
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
```

```
        ll = len(c[0][0])
        d = m.listOfTagsij(l, ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 4
        reacV1 = {}
        mem = 0
        ops.model("basicBuilder", "-ndm", 3, "-ndf", 4)

        TCL = open("cubotto.TCL", "a")
        TCL.write("model BasicBuilder -ndm 3 -ndf 4\n")
        TCL.close()

        py = open("cubotto_exe.py", "a")
        py.write('ops.model(" basicBuilder", " - ndm", 3, " -
ndf ", 4)\n')
        py.close()
        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)
            reacV1[int(i[0])] = mem

            ops.remove('sp',int(i[0]))

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                py = open("cubotto_exe.py", "a")
                py.write(f'if pid == {proc}:\n')
                py.write(f"    ops.remove('sp',{int(i[0])},1)")

                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == {proc}'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 1\n')
                TCL.write('}\n')
                TCL.close()

                py.write(f'if pid == 0:\n')

                py.write(f'    ops.re-
move("sp",{int(i[0])},1)\n')
                py.close()

                TCL=open("cubotto.TCL","a")
```

```
                TCL.write("if {"+f' $pid == 0'+' }'+' {\n')
                TCL.write(f'    remove sp {int(i[0])} 1\n')
                TCL.write('}\n')
                TCL.close()

         print('\nEdge\n')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 3
        reacV2={}
        mem = 0

        ops.model("basicBuilder", "-ndm", 3, "-ndf", 3)
        py = open("cubotto_exe.py", "a")
        py.write('ops.model(" basicBuilder", " - ndm", 3, " -
ndf ", 3)\n')
        py.close()

        TCL = open("cubotto.TCL", "a")
        TCL.write("model BasicBuilder -ndm 3 -ndf 3\n")
        TCL.close()

        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)

            reacV2[int(i[0])] = mem

            ops.remove('sp',int(i[0]))

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:


                py = open("cubotto_exe.py", "a")
                py.write(f'if pid == {proc}:\n')
                py.write(f"    ops.re-
move('sp',{int(i[0])},1)\n")
```

```
                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == {proc}'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 1\n')
                TCL.write('}\n')
                TCL.close()

                py.write('if pid == 0:\n')
                py.write(f'    ops.re-
move("sp",{int(i[0])},1)\n')

                py.close()

                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == 0'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 1\n')
                TCL.write('}\n')
                TCL.close()

    def mRemoveY(self):

        print('funzione remove')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()

        proc_dict = m.mDict()

        type_el = str(input('what geoElement do you want to re-
move fixes'))
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l, ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
```

```
        ndf = 4
        reacV1 = {}
        mem = 0
        ops.model("basicBuilder", "-ndm", 3, "-ndf", 4)

        TCL = open("cubotto.TCL", "a")
        TCL.write("model BasicBuilder -ndm 3 -ndf 4\n")
        TCL.close()

        py = open("cubotto_exe.py", "a")
        py.write('ops.model(" basicBuilder", " - ndm", 3, " -
ndf ", 4)\n')
        py.close()
        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)
            reacV1[int(i[0])] = mem

            ops.remove('sp',int(i[0]))

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:

                py = open("cubotto_exe.py", "a")
                py.write(f'if pid == {proc}:\n')
                py.write(f"    ops.re-
move('sp',{int(i[0])},2)\n")

                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == {proc}'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 2\n')
                TCL.write('}\n')
                TCL.close()

                py.write(f'if pid == 0:\n')

                py.write(f'    ops.re-
move("sp",{int(i[0])},2)\n')
                py.close()

                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == 0'+' }'+' {\n')
                TCL.write(f'    remove sp {int(i[0])} 2\n')
                TCL.write('}\n')
                TCL.close()

         print('\nEdge\n')
```

```
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 3
        reacV2={}
        mem = 0

        ops.model("basicBuilder", "-ndm", 3, "-ndf", 3)
        py = open("cubotto_exe.py", "a")
        py.write('ops.model(" basicBuilder", " - ndm", 3, " -
ndf ", 3)\n')
        py.close()

        TCL = open("cubotto.TCL", "a")
        TCL.write("model BasicBuilder -ndm 3 -ndf 3\n")
        TCL.close()

        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)

            reacV2[int(i[0])] = mem

            ops.remove('sp',int(i[0]))

            procId = trova_nodo(proc_dict,int(i[0]))
            for proc in procId:


                py = open("cubotto_exe.py", "a")
                py.write(f'if pid == {proc}:\n')
                py.write(f"    ops.remove('sp',{int(i[0])},2\n")



                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == {proc}'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 2\n')
                TCL.write('}\n')
```

```
                TCL.close()

                py.write('if pid == 0:\n')
                py.write(f'    ops.re-
move("sp",{int(i[0])},2)\n')

                py.close()

                TCL=open("cubotto.TCL","a")
                TCL.write("if {"+f' $pid == 0'+' }'+' {\n')

                TCL.write(f'    remove sp {int(i[0])} 2\n')
                TCL.write('}\n')
                TCL.close()
```

APPENDICE 6 - py code 24 – Funzioni mDefine

```
def mDefine():
        print('funzione mDefine')
        m = Model()
        proc_dict = m.mDict()
        b = m.setGeoElement('Solid')
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('20-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 4
        m.sendOpsNodes(dim,ndf)
        ##
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
```

```
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 3
        ndf = 3
        m.sendOpsNodes(dim,ndf)
```

APPENDICE 6 - py code 25 – Funzione mBoundNodesPid0

```
def boundNodesPid0():
    print('funzione boundNodesPid0')
    print('attenzione al nome dei geoGruppi, dal gruppo si sono
elminati i solidi e le linee, attenzione ai nomi, rieditare
boundNodesPid0')
    m = Model()
    boundNodesCornTag = []
    boundNodesEdgeTag = []

    for i in m.PhysGr:
        type_el = str(i)
        if type_el == 'Solid':
            continue
        elif type_el == 'Edges':
            continue
        elif (type_el == "SX" or type_el == "DX" or type_el ==
"SY" or type_el == "DY" or type_el == "DOWN" or type_el ==
"TOP"):
            b = m.setGeoElement(type_el)
            bb = m.getElementsVectors()
            m.setUniqueVector(bb)
            c = m.nodeCornEdge('8-nodi')
            m.setUniqueVector(c[0])
            print(len(c[0]))
            print(len(c[0][0]))
            l = len(c[0])
            ll = len(c[0][0])
            d = m.listOfTagsij(l,ll)
            m.setUniqueVector(d)
            e = m.makeUnique()
            for i in e:
                    boundNodesCornTag.append(i)

    for i in m.PhysGr:
        type_el = str(i)
        if type_el == 'Solid':
            continue
        elif type_el == 'Edges':
            continue
```

```
        elif (type_el == "SX" or type_el == "DX" or type_el ==
"SY" or type_el == "DY" or type_el == "DOWN" or type_el ==
"TOP"):

            b = m.setGeoElement(type_el)
            bb = m.getElementsVectors()
            m.setUniqueVector(bb)
            c = m.nodeCornEdge('8-nodi')

            m.setUniqueVector(c[1])
            print(len(c[1]))
            print(len(c[1][0]))
            l=len(c[1])
            ll=len(c[1][0])
            d = m.listOfTagsij(l,ll)
            m.setUniqueVector(d)
            e = m.makeUnique()
            for i in e:
                boundNodesEdgeTag.append(i)

    boundNodeCorner = nup.unique(nup.array(boundNodesCornTag))
    boundNodeEdge = nup.unique(nup.array(boundNodesEdgeTag))


    dim = 3
    ndf = 4

    py = open("cubotto_exe.py","a")
    py.write(f'ops.model("basicBuilder","-ndm",{dim},"-
ndf",{ndf})\n')
    py.write('if pid == 0:\n')
    py.close()

    TCL=open("cubotto.TCL","a")
    TCL.write(f'model BasicBuilder -ndm {dim} -ndf {ndf}\n')
    TCL.write("if {"+f' $pid == 0'+' }'+' {\n')
    TCL.close()

    for j in boundNodeCorner:
        x = Gmsh.model.mesh.getNode(int(j))[0][0]
        y = Gmsh.model.mesh.getNode(int(j))[0][1]
        z = Gmsh.model.mesh.getNode(int(j))[0][2]
        num = int(j)
        py = open("cubotto_exe.py","a")
        py.write(f'    ops.node({num},{x},{y},{z})\n')
        py.close()
```

```
        TCL=open("cubotto.TCL","a")
        TCL.write(f'    node {num} {x} {y} {z}\n')
        TCL.close()

    TCL=open("cubotto.TCL","a")
    TCL.write('}\n')
    TCL.close()



    dim = 3
    ndf = 3

    py = open("cubotto_exe.py","a")
    py.write(f'ops.model("basicBuilder","-ndm",{dim},"-
ndf",{ndf})\n')
    py.write('if pid == 0:\n')
    py.close()

    TCL=open("cubotto.TCL","a")
    TCL.write(f'model BasicBuilder -ndm {dim} -ndf {ndf}\n')
    TCL.write("if {"+f' $pid == 0'+' }'+' {\n')
    TCL.close()

    for j in boundNodeEdge:
        x = Gmsh.model.mesh.getNode(int(j))[0][0]
        y = Gmsh.model.mesh.getNode(int(j))[0][1]
        z = Gmsh.model.mesh.getNode(int(j))[0][2]
        num = int(j)
        py = open("cubotto_exe.py","a")
        py.write(f'    ops.node({num},{x},{y},{z})\n')
        py.close()

        TCL=open("cubotto.TCL","a")
        TCL.write(f'    node {num} {x} {y} {z}\n')
        TCL.close()

    TCL=open("cubotto.TCL","a")
    TCL.write('}\n')
    TCL.close()


    print(boundNodeCorner,boundNodeEdge)
    return [boundNodeCorner,boundNodeEdge]
```

APPENDICE 6 - py code 26 – Funzioni mgetFixedCoord

```
def mgetFixedCoord():
        f = ops.getFixedNodes()
        coordTag=[]
        fixedCoord=[]
        Dofs=[]
        for i in f:
            coordTag.append(i)
            coordFixNode = ops.nodeCoord(i)
            fixedCoord.append(coordFixNode)
            g = ops.getFixedDOFs(i)
            Dofs.append(g)
        return [coordTag,fixedCoord,Dofs]
```

APPENDICE 6 - py code 27 – Funzione mFix

```
def mFix():
    print('funzione mFix')
    print(g2o.get_physical_groups_map(Gmsh.model))
    m = Model()

    type_el = str(input('inserisci l elemento geo da vincolare:
'))
    print('inserire 1 per vincolo on 0 per vincolo off')

    DofL = []
    x = int(input('x: 1 0n - 0 off: '))
    f_x = 0
    if x == 1:
        f_x = 1
        DofL.append(f_x)
    else:
        f_x = 0

    y = int(input('y: 1 0n - 0 off: '))
    f_y = 0
    if y == 1:
        f_y = 2
        DofL.append(f_y)
    else:
        f_y = 0

    z = int(input('z: 1 0n - 0 off: '))
    f_z = 0
    if z == 1:
        f_z = 3
        DofL.append(f_z)
    else:
```

```
        f_z = 0

    p = int(input('p: 1 0n - 0 off: '))
    f_p = 0
    if p == 1:
        f_p = 4
        DofL.append(f_p)
    else:
        f_p = 0
    #Vettore dof Locale ok
    fix_mem = [x, y, z, p]
    b = m.setGeoElement(type_el)
    bb = m.getElementsVectors()
    m.setUniqueVector(bb)
    ## 20 nodi
    c = m.nodeCornEdge('8-nodi')
    m.setUniqueVector(c[0])
    print(len(c[0]))
    print(len(c[0][0]))
    l = len(c[0])
    ll = len(c[0][0])
    d = m.listOfTagsij(l, ll)
    m.setUniqueVector(d)
    e = m.makeUnique()
    f = m.nodeNumCoordVector()

    ff = []
    num = 0
    for i in f:
        # i[0] = tag
        # i[1] = coordinate np.float[0.0 1.0 1.0]
        x = fix_mem[0]
        y = fix_mem[1]
        z = fix_mem[2]
        p = fix_mem[3]

        isFixed = ops.getFixedDOFs(i[0])
        flag = 0
        num = 0
        mem = []
        if isFixed == []:
            ff.append(i)
            m.setUniqueVector(ff)
            dim = 3
            ndf = 4
            m.sendOpsFixes(dim, ndf, x, y, z, p)
            ff = []
```

```
            continue
        for j in isFixed:
            if j == 1:
                x = 0
            if j == 2:
                y = 0
            if j == 3:
                z = 0
            if j == 4:
                p = 0
        ff.append(i)
        m.setUniqueVector(ff)
        dim = 3
        ndf = 4
        m.sendOpsFixes(dim, ndf, x, y, z, p)
        ff = []

    print('\nEdge\n')
    m.setUniqueVector(c[1])
    print(len(c[1]))
    print(len(c[1][0]))
    l = len(c[1])
    ll = len(c[1][0])
    d = m.listOfTagsij(l, ll)
    m.setUniqueVector(d)
    e = m.makeUnique()
    f = m.nodeNumCoordVector()
    ff = []

    for i in f:
        # i[0] = tag
        # i[1] = coordinate np.float[0.0 1.0 1.0]
        x = fix_mem[0]
        y = fix_mem[1]
        z = fix_mem[2]
        p = fix_mem[3]
        isFixed = ops.getFixedDOFs(i[0])
        flag = 0
        num = 0
        mem = []
        if isFixed == []:
            ff.append(i)
            m.setUniqueVector(ff)
            dim = 3
            ndf = 3
            m.sendOpsFixes(dim, ndf, x, y,z,0)
            ff = []
```

```
            continue
        for j in isFixed:
            if j == 1:
                x = 0
            if j == 2:
                y = 0
            if j == 3:
                z = 0
        ff.append(i)
        m.setUniqueVector(ff)
        dim = 3
        ndf = 3
        m.sendOpsFixes(dim, ndf, x, y, z,0)
        ff = []
```

APPENDICE 6 - py code 28 – Funzione mFixVol

```
def mFixVol():
    print('funzione mFix')
    print(g2o.get_physical_groups_map(Gmsh.model))
    m = Model()

    type_el = str(input('inserisci l elemento geo da vincolare:
'))
    print('inserire 1 per vincolo on 0 per vincolo off')

    DofL = []
    x = int(input('x: 1 0n - 0 off: '))
    f_x = 0
    if x == 1:
        f_x = 1
        DofL.append(f_x)
    else:
        f_x = 0

    y = int(input('y: 1 0n - 0 off: '))
    f_y = 0
    if y == 1:
        f_y = 2
        DofL.append(f_y)
    else:
        f_y = 0

    z = int(input('z: 1 0n - 0 off: '))
    f_z = 0
    if z == 1:
        f_z = 3
        DofL.append(f_z)
```

```
    else:
        f_z = 0

    p = int(input('p: 1 0n - 0 off: '))
    f_p = 0
    if p == 1:
        f_p = 4
        DofL.append(f_p)
    else:
        f_p = 0
    #Vettore dof Locale ok
    fix_mem = [x, y, z, p]
    b = m.setGeoElement(type_el)
    bb = m.getElementsVectors()
    m.setUniqueVector(bb)
    ## 20 nodi
    c = m.nodeCornEdge('20-nodi')
    m.setUniqueVector(c[0])
    print(len(c[0]))
    print(len(c[0][0]))
    l = len(c[0])
    ll = len(c[0][0])
    d = m.listOfTagsij(l, ll)
    m.setUniqueVector(d)
    e = m.makeUnique()
    f = m.nodeNumCoordVector()

    ff = []
    num = 0
    for i in f:
        # i[0] = tag
        # i[1] = coordinate np.float[0.0 1.0 1.0]
        x = fix_mem[0]
        y = fix_mem[1]
        z = fix_mem[2]
        p = fix_mem[3]

        isFixed = ops.getFixedDOFs(i[0])
        flag = 0
        num = 0
        mem = []
        if isFixed == []:
            ff.append(i)
            m.setUniqueVector(ff)
            dim = 3
            ndf = 4
            m.sendOpsFixes(dim, ndf, x, y, z, p)
```

```
            ff = []
            continue
        for j in isFixed:
            if j == 1:
                x = 0
            if j == 2:
                y = 0
            if j == 3:
                z = 0
            if j == 4:
                p = 0
        ff.append(i)
        m.setUniqueVector(ff)
        dim = 3
        ndf = 4
        m.sendOpsFixes(dim, ndf, x, y, z, p)
        ff = []

    print('\nEdge\n')
    m.setUniqueVector(c[1])
    print(len(c[1]))
    print(len(c[1][0]))
    l = len(c[1])
    ll = len(c[1][0])
    d = m.listOfTagsij(l, ll)
    m.setUniqueVector(d)
    e = m.makeUnique()
    f = m.nodeNumCoordVector()
    ff = []

    for i in f:
        # i[0] = tag
        # i[1] = coordinate np.float[0.0 1.0 1.0]
        x = fix_mem[0]
        y = fix_mem[1]
        z = fix_mem[2]
        p = fix_mem[3]

        isFixed = ops.getFixedDOFs(i[0])
        flag = 0
        num = 0
        mem = []
        if isFixed == []:
            ff.append(i)
            m.setUniqueVector(ff)
            dim = 3
            ndf = 3
```

```
            m.sendOpsFixes(dim, ndf, x, y,z,0)
            ff = []
            continue
        for j in isFixed:
            if j == 1:
                x = 0
            if j == 2:
                y = 0
            if j == 3:
                z = 0
        ff.append(i)
        m.setUniqueVector(ff)
        dim = 3
        ndf = 3
        m.sendOpsFixes(dim, ndf, x, y, z,0)
        ff = []
```

APPENDICE 6 - py code 29 – Funzione mRec

```
def mRec():
    print('funzione mRec')
    print('The code is under developement, please remeber to
change the 3d recorder element range in mRec functions')
    global nodeList

    m = Model()
    proc_dict = m.mDict()
    b = m.setGeoElement('Solid')
    bb = m.getElementsVectors()
    eleNodes = bb[1]
    eleTag = bb[0]
    m.setUniqueVector(eleNodes)
    l = len(eleNodes)
    ll = len(eleNodes[0])
    d = m.listOfTagsij(l, ll)
    m.setUniqueVector(d)
    e = m.makeUnique()
    nup.savetxt('nodeInfo.txt', e)

    load_nodeList = nup.loadtxt('nodeInfo.txt')
    nodeList = []

    for i in range(len(load_nodeList)):
        nodeList.append(int(load_nodeList[i]))

    m.setUniqueVector(eleTag)
    max_value = nup.max(eleTag)
```

```
    min_value = nup.min(eleTag)
    print(f'min_value = {min_value}, max_value = {max_value}')
    ops.recorder('Node', '-xml', 'Gdisplacement.part-$pid.out',
'-time', '-node', *nodeList, '-dof', 1, 2, 3, 'disp')
    ops.recorder('Node', '-xml', 'Gacceleration.part-$pid.out',
'-time', '-node', *nodeList, '-dof', 1, 2, 3, 'accel')
    ops.recorder('Node', '-xml', 'GporePressure.part-$pid.out',
'-time', '-node', *nodeList, '-dof', 4, 'vel')

    for nproc, subdict in proc_dict.items():
        if subdict == {}:
            break
        else:
            py = open("cubotto_exe.py", "a")
            py.write(f'if pid == {nproc}:\n')
            py.close()

            TCL=open("cubotto.TCL","a")
            TCL.write("if {"+f' $pid == {nproc}'+' }'+' {\n')
            TCL.close()

            for el_num,nod_list in subdict.items():
                max_val = int(nup.max(nod_list))
                min_val = int(nup.min(nod_list))
            py = open("cubotto_exe.py", "a")
            py.write(f"    ops.recorder('Node','-xml','Gdis-
placement.part-$pid.out','-time','-nodeRange',{min_val},
{max_val},'-dof', 1, 2, 3, 'disp')\n\
    ops.recorder('Node','-xml','Gacceleration.part-$pid.out','-
time','-nodeRange',{min_val}, {max_val},'-dof', 1, 2, 3, 'ac-
cel')\n\
    ops.recorder('Node','-xml','GporePressure.part-$pid.out','-
time','-nodeRange',{min_val}, {max_val},'-dof', 4, 'vel')\n")
            py.close()

            TCL = open('cubotto.TCL', 'a')
            TCL.write(f'    recorder Node -xml Gdisplace-
ment.part-$pid.out -time -nodeRange {min_val} {max_val} -dof 1 2
3 disp \n\
    recorder Node -xml Gvelocity.part-$pid.out -time -time -
nodeRange {min_val} {max_val} -dof 1 2 3 vel \n\
    recorder Node -xml Gacceleration.part-$pid.out -time -
nodeRange {min_val} {max_val} -dof 1 2 3 accel \n\
    recorder Node -xml Gporepressure.part-$pid.out -time -
nodeRange {min_val} {max_val} -dof 4 vel \n')
            TCL.write('}\n')
            TCL.close()
```

```
    ops.recorder('Element', '-xml', 'Gstress.part-$pid.out', '-
time', '-eleRange', 1, 50000, 'material', '1', 'stress')
    ops.recorder('Element', '-xml', 'Ggauss.part-$pid.out', '-
time', '-eleRange', 1, 50000, 'material', '1', 'gausspoint')
    ops.recorder('Element', '-xml', 'Gstrain.part-$pid.out', '-
time', '-eleRange', 1, 50000, 'material', '1', 'strain')

    for nproc, subdict in proc_dict.items():
        if subdict == {}:
            break
        else:
            el_n_list=[]
            py = open("cubotto_exe.py", "a")
            py.write(f'if pid == {nproc}:\n')
            py.close()

            TCL=open("cubotto.TCL","a")
            TCL.write("if {"+f' $pid == {nproc}'+' }'+' {\n')
            TCL.close()

            for el_num,nod_list in subdict.items():
                el_n_list.append(el_num)
            max_val = int(nup.max(el_n_list))
            min_val = int(nup.min(el_n_list))
        py = open("cubotto_exe.py", "a")
        py.write(f"    ops.recorder('Element','-
file','Gstress.part-$pid.out','-time','-eleRange',
{min_val},{max_val},'material','1','stress')\n\
    ops.recorder('Element','-file','Ggauss.part-$pid.out','-
time','-eleRange', {min_val},{max_val},'materi-
al','1','gausspoint')\n\
    ops.recorder('Element','-file','Gstrain.part-$pid.out','-
time','-eleRange', {min_val},{max_val},'materi-
al','1','strain')\n")
        py.close()

        TCL = open('cubotto.TCL', 'a')
        TCL.write(f'    recorder Element -file Gstress.part-
$pid.out -time -eleRange {min_val} {max_val} -material 1
stress\n\
    recorder Element -file Gstrain.part-$pid.out -time -eleRange
{min_val} {max_val} -material 1 strain\n')
        TCL.write('}\n')
        TCL.close()

    print("OK RECORDERS")
```

```
    return nodeList
```

APPENDICE 6 - py code 30 – Funzione mGenFem20

```
def mGenFem20():
    ops.model("basicBuilder", "-ndm", 3, "-ndf", 4)
    py = open("cubotto_exe.py", "a")
    py.write('ops.model("basicBuilder", "-ndm", 3, "-ndf",
4)\n')
    py.close()

    TCL = open("cubotto.TCL", "a")
    TCL.write("model BasicBuilder -ndm 3 -ndf 4\n")
    TCL.close()

    print('funzione mGenFem20')
    m = Model()
    proc_dict = m.mDict()
    ###
    Volume = str(input('inserisci l elemento geo Solid da model-
lare: '))
    materiale = int(input("inserisci il tag del materiale asso-
ciato al Volume: "))
    b = m.setGeoElement(Volume)
    ###
    bb = m.getElementsVectors()
    eleNodes = bb[1]
    eleTag = bb[0]
    LET = len(bb[0])
    for i in range(0,LET):
        print(f'mGenFem20 - percent completed: {(i*100)/LET}')
        elem = int(eleTag[i])
        nodo1 = int(eleNodes[i][0])
        nodo2 = int(eleNodes[i][1])
        nodo3 = int(eleNodes[i][2])
        nodo4 = int(eleNodes[i][3])
        nodo5 = int(eleNodes[i][4])
        nodo6 = int(eleNodes[i][5])
        nodo7 = int(eleNodes[i][6])
        nodo8 = int(eleNodes[i][7])
        nodo9 = int(eleNodes[i][8])
        nodo10 = int(eleNodes[i][9])
        nodo11 = int(eleNodes[i][10])
        nodo12 = int(eleNodes[i][11])
        nodo13 = int(eleNodes[i][12])
        nodo14 = int(eleNodes[i][13])
        nodo15 = int(eleNodes[i][14])
        nodo16 = int(eleNodes[i][15])
```

```
        nodo17 = int(eleNodes[i][16])
        nodo18 = int(eleNodes[i][17])
        nodo19 = int(eleNodes[i][18])
        nodo20 = int(eleNodes[i][19])

        nodes_l_first =
[nodo1,nodo2,nodo3,nodo4,nodo5,nodo6,nodo7,nodo8,nodo9, nodo10,
nodo11,nodo12, nodo13, nodo14,nodo15,nodo16, nodo17,nodo18,
nodo19, nodo20]

        mem = []
        coordinate = []
        coordinate_x = []
        coordinate_y = []
        coordinate_z = []
        for j in nodes_l_first:
            x = Gmsh.model.mesh.getNode(int(j))[0][0]
            y = Gmsh.model.mesh.getNode(int(j))[0][1]
            z = Gmsh.model.mesh.getNode(int(j))[0][2]
            mem = [j,[x,y,z]]
            coordinate.append(mem)
            coordinate_x.append(x)
            coordinate_y.append(y)
            coordinate_z.append(z)


        x = nup.array(coordinate_x)
        y = nup.array(coordinate_y)
        z = nup.array(coordinate_z)

        x_min = nup.min(x)
        x_max = nup.max(x)
        y_min = nup.min(y)
        y_max = nup.max(y)
        z_min = nup.min(z)
        z_max = nup.max(z)



        coordinate_corner=[]
        for i in range(8):
            coordinate_corner.append(coordinate[i])
        coordinate_o = sorted(coordinate_corner, key = lambda x:
x[1][2])

        #nodi spigoli 1 2 3 4
```

```
        coordinate2=[]
        for i in range(4):
            coordinate2.append(coordinate_o[i])

        nodes_l = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
        s = []
        s = sorted(coordinate2, key = lambda x: x[1][1])
        select = []
        for i in range(2):
            select.append(s[i])
        s1 = []
        s1 = sorted(select,key = lambda x: x[1][0])

        select2 = []
        s2 = []
        for i in range(2,4):
            select2.append(s[i])
        s3 = []
        s3 = sorted(select2,key = lambda x: x[1][0])


        nodes_l[0] = s1[0][0]
        nodes_l[1] = s1[1][0]
        nodes_l[2] = s3[1][0]
        nodes_l[3] = s3[0][0]

        #nodi spigoli 5 6 7 8

        coordinate2=[]
        for i in range(4,8):
            coordinate2.append(coordinate_o[i])


        s = []
        s = sorted(coordinate2, key = lambda x: x[1][1])
        select = []
        for i in range(2):
            select.append(s[i])
        s1 = []
        s1 = sorted(select,key = lambda x: x[1][0])

        select2 = []
        s2 = []
        for i in range(2,4):
            select2.append(s[i])
        s3 = []
        s3 = sorted(select2,key = lambda x: x[1][0])
```

```
        nodes_l[4] = s1[0][0]
        nodes_l[5] = s1[1][0]
        nodes_l[6] = s3[1][0]
        nodes_l[7] = s3[0][0]


        coordinate_mid_total=[]
        for i in range(8,20):
            coordinate_mid_total.append(coordinate[i])
        coordinate_o = sorted(coordinate_mid_total, key = lambda
x: x[1][2])


        #DOWN

        coordinate_mid=[]
        for i in range(4):
            coordinate_mid.append(coordinate_o[i])
        s4 = []
        s4 = sorted(coordinate_mid, key = lambda x: x[1][1])



        select = []
        for i in range(2):
            select.append(s4[i])


        nodes_l[8]=s4[0][0]

        coordinate_mid2 = []
        for i in range(1,3):
            coordinate_mid2.append(s4[i])

        s5 = []
        s5 = sorted(coordinate_mid2,key = lambda x: x[1][0])
        nodes_l[9]=s5[1][0]
        nodes_l[10]=s4[3][0]
        nodes_l[11]=s5[0][0]


        #TOP

        coordinate_mid=[]
        for i in range(8,12):
```

```
            coordinate_mid.append(coordinate_o[i])
        s4 = []
        s4 = sorted(coordinate_mid, key = lambda x: x[1][1])
        select = []
        for i in range(2):
            select.append(s4[i])

        nodes_l[12]=s4[0][0]

        coordinate_mid2 = []
        for i in range(1,3):
            coordinate_mid2.append(s4[i])

        s5 = []
        s5 = sorted(coordinate_mid2,key = lambda x: x[1][0])
        nodes_l[13]=s5[1][0]
        nodes_l[14]=s4[3][0]
        nodes_l[15]=s5[0][0]


        #MID

        coordinate2=[]
        for i in range(4,8):
            coordinate2.append(coordinate_o[i])

        s = []
        s = sorted(coordinate2, key = lambda x: x[1][1])
        select = []
        for i in range(2):
            select.append(s[i])
        s1 = []
        s1 = sorted(select,key = lambda x: x[1][0])

        select2 = []
        s2 = []
        for i in range(2,4):
            select2.append(s[i])
        s3 = []
        s3 = sorted(select2,key = lambda x: x[1][0])


        nodes_l[16] = s1[0][0]
        nodes_l[17] = s1[1][0]
        nodes_l[18] = s3[1][0]
        nodes_l[19] = s3[0][0]
        print(f'elemento: {elem}, nodi: {nodes_l}')
```

```
        ops.element('20_8_BrickUP',elem,*nodes_l, materiale,
2.2e6, 1, 1.0, 1.0, 1.0, 0.0, 0.0,-9.81)
        ################################################
        procId = trova_elemento(proc_dict,elem)
        if elem == []:
            continue
        else:
            py = open("cubotto_exe.py", "a")
            py.write(f'if pid == {procId}:\n')
            py.write(f"    ops.element('20_8_BrickUP',{elem},
{nodes_l[0]}, {nodes_l[1]}, {nodes_l[2]}, {nodes_l[3]},
{nodes_l[4]}, {nodes_l[5]}, {nodes_l[6]}, {nodes_l[7]},
{nodes_l[8]}, {nodes_l[9]}, {nodes_l[10]}, {nodes_l[11]},
{nodes_l[12]}, {nodes_l[13]}, {nodes_l[14]}, {nodes_l[15]},
{nodes_l[16]}, {nodes_l[17]}, {nodes_l[18]}, {nodes_l[19]}, {ma-
teriale}, 2.2e6, 1, 1.0, 1.0, 1.0, 0.0, 0.0,-9.81)\n")
            py.close()

            TCL = open("cubotto.TCL","a")
            TCL.write("if {"+f' $pid == {procId}'+' }'+' {\n')
            TCL.write(f'    element 20_8_BrickUP {elem}
{nodes_l[0]} {nodes_l[1]} {nodes_l[2]} {nodes_l[3]} {nodes_l[4]}
{nodes_l[5]} {nodes_l[6]} {nodes_l[7]} {nodes_l[8]} {nodes_l[9]}
{nodes_l[10]} {nodes_l[11]} {nodes_l[12]} {nodes_l[13]}
{nodes_l[14]} {nodes_l[15]} {nodes_l[16]} {nodes_l[17]}
{nodes_l[18]} {nodes_l[19]} {materiale}  2.2e6 1 1.0 1.0 1.0 0.0
0.0 -9.81\n')
            TCL.write ('}\n')
            TCL.close()
```

APPENDICE 6 - py code 31 – Funzione mDiaf

```
def mDiaf():

    m = Model()

    ops.model("basicBuilder", "-ndm", 3, "-ndf", 3)
    nodo_mesh = int(input('inserisci il nodo selezionato per il
dashpot: '))
    el = str(input('inserisci la superficie da irrigidire: '))
    b = m.setGeoElement(el)
    bb = m.getElementsVectors()
    m.setUniqueVector(bb)
    d = bb[1]
    m.setUniqueVector(d)
    e = m.makeUnique()
    for i in e:
```

```
        if i == nodo_mesh:
            continue
        else:
            ops.equalDOF(int(nodo_mesh), int(i), 1)
            ops.equalDOF(int(nodo_mesh), int(i), 2)
            ops.equalDOF(int(nodo_mesh), int(i), 3)

            py = open("cubotto_exe.py", "a")
            py.write(f'if pid == 0:\n')
            py.write(f'    ops.equalDOF({int(nodo_mesh)},
{int(i)}, 1)\n')
            py.write(f'    ops.equalDOF({int(nodo_mesh)},
{int(i)}, 2)\n')
            py.write(f'    ops.equalDOF({int(nodo_mesh)},
{int(i)}, 3)\n')
            py.close()

            TCL = open("cubotto.TCL", "a")
            TCL.write("if {"+f' $pid == 0'+' }'+' {\n')
            TCL.write("model BasicBuilder -ndm 3 -ndf 3\n")
            TCL.write(f'equalDOF {int(nodo_mesh)} {int(i)} 1\n')
            TCL.write(f'equalDOF {int(nodo_mesh)} {int(i)} 2\n')
            TCL.write(f'equalDOF {int(nodo_mesh)} {int(i)} 3\n')
            TCL.write('}\n')
            TCL.close()
```

APPENDICE 6 - py code 32 – Funzione mDash

```
def mDash():
        ops.model("basicBuilder", "-ndm", 3, "-ndf", 3)
        py = open("cubotto_exe.py", "a")
        py.write('ops.model(" basicBuilder", " - ndm", 3, " -
ndf ", 3)\n')
        py.close()

        TCL = open("cubotto.TCL", "a")
        TCL.write("model BasicBuilder -ndm 3 -ndf 3\n")
        TCL.close()


        print('funzione mDash')
        m = Model()
        proc_dict = m.mDict()
        b = m.setGeoElement('Solid')
        bb = m.getElementsVectors()
        eleNodes = bb[1]
        eleTag = bb[0]
        m.setUniqueVector(eleNodes)
```

```
        l = len(bb[1])
        ll=len(bb[1][0])
        c = m.listOfTagsij(l,ll)
        m.setUniqueVector(c)
        cc = m.makeUnique()
        max_value = nup.max(cc)
        min_value = nup.min(cc)
        dashFree = int(max_value + 1)

        print(f'dashFree = {dashFree}')

        dashFix = int(max_value + 2)
        print(f'dashFix = {dashFix}')

        x = float(input('insert x coord of dashpot in format
0.0'))
        y = float(input('insert y coord of dashpot in format
0.0'))
        z = float(input('insert z coord of dashpot in format
0.0'))
        ops.node(dashFree,x,y,z)
        ops.node(dashFix,x,y,z)
        ops.fix(dashFix,1,1,1)

        f_x = int(input('Free node - insert 1 to constrain on -
0 to constrain off on dir x'))
        f_y = int(input('Free node - insert 1 to constrain on -
0 to constrain off on dir y'))
        f_z = int(input('Free node - insert 1 to constrain on -
0 to constrain off on dir z'))

        ops.fix(int(dashFree),f_x,f_y,f_z)
        nodo_mesh = int(input('write the mesh node tag who want
to assigned dashpot'))

        ops.equalDOF(int(nodo_mesh), int(dashFree),1)

        colArea =   float(input('conn. area of dashpot'))
        rockVS =    float(input('Bedrock VS'))
        rockDen  =  float(input('Bedrock density'))
        dashpotCoeff = rockVS*rockDen*colArea
        tag = int(input('insert uniaxialMaterial tag (> n. geo-
material)'))
        alpha = 1
        ops.uniaxialMaterial('Viscous',tag,dashpotCoeff, alpha)

        m.setUniqueVector(eleTag)
```

```
        max_tag = nup.max(eleTag)
        print('ATTENZIONE PER IL MOMENTO POSSIBILE SOLO LUNGO X
QUINDI PREMERE 1')
        var_dir = int(input('dof of dashpot -dir: 1 along x, 2
along xy, 3 along y'))
        if var_dir == 1:
            dir = [1]
        elif var_dir == 2:
            dir = [1,2]
        elif var_dir == 3:
            dir = [2]
        ops.element('zeroLength', int(max_tag+1), int(dashFix),
int(dashFree), '-mat', tag, '-dir', *dir)

        procId = trova_nodo(proc_dict,int(nodo_mesh))
        for proc in procId:

            py = open("cubotto_exe.py", "a")
            py.write(f'if pid == {proc}:\n')

            TCL=open("cubotto.TCL","a")
            TCL.write("if {"+f' $pid == {proc}'+' }'+' {\n')
            TCL.close()

            py = open("cubotto_exe.py", "a")
            py.write(f'    ops.node({dashFree},{x},{y},{z})\n')
            py.close()

            TCL=open("cubotto.TCL","a")
            TCL.write(f'    node {dashFree} {x} {y} {z}\n')
            TCL.close()

            py = open("cubotto_exe.py", "a")
            py.write(f'    ops.node({dashFix},{x},{y},{z})\n')
            py.close()

            TCL=open("cubotto.TCL","a")
            TCL.write(f'    node {dashFix} {x} {y} {z}\n')
            TCL.close()

            py = open("cubotto_exe.py", "a")
            py.write(f'    ops.fix({dashFix},1,1,1)\n')
            py.close()

            TCL=open("cubotto.TCL","a")
            TCL.write(f'    fix {dashFix} 1 1 1\n')
            TCL.close()
```

```
            py = open("cubotto_exe.py", "a")
            py.write(f'    ops.fix({dash-
Free},{f_x},{f_y},{f_z})\n')
            py.close()

            TCL=open("cubotto.TCL","a")
            TCL.write(f'    fix {dashFree} {f_x} {f_y} {f_z}\n')
            TCL.close()
        ################## ATTENZIONE SI è SPECIFICATA LA DIRE-
ZIONE 1 lungo x QUINDI E POSSIBILE SOLO LUNGO X
            py = open("cubotto_exe.py", "a")
            py.write(f'    ops.equalDOF(int({nodo_mesh}),
int({dashFree}),1)\n')
            py.close()

            TCL=open("cubotto.TCL","a")
            TCL.write(f'    equalDOF {int(nodo_mesh)} {int(dash-
Free)} 1\n')
            TCL.close()

            py = open("cubotto_exe.py", "a")
            py.write(f"    ops.uniaxialMaterial('Vis-
cous',{tag},{dashpotCoeff}, {alpha})\n")
            py.close()
        ################################################
            TCL=open("cubotto.TCL","a")
            TCL.write(f'    uniaxialMaterial Viscous {tag}
{dashpotCoeff} {alpha}\n')
            TCL.close()

        ################################################
            py = open("cubotto_exe.py", "a")
            py.write(f'    dir = {dir}\n')
            py.write(f"    ops.element('zeroLength',
{int(max_tag+1)}, {int(dashFix)}, {int(dashFree)}, '-mat',
{tag}, '-dir', *dir)\n")
            py.close()
        ################################################ ATTEN-
ZIONE SI è IMPOSTO DIREZIONE 1
            TCL=open("cubotto.TCL","a")
            #TCL.write(f'    set dir {dir}\n')
            TCL.write(f'    set dir 1\n')
            TCL.write(f'    element zeroLength  {int(max_tag+1)}
{int(dashFix)} {int(dashFree)} -mat {tag} -dir $dir\n')
            TCL.write("}\n")
```

```
        return [colArea,dashpotCoeff,nodo_mesh]
```

APPENDICE 6 - py code 33 – Funzione floatingNodes

```
def floatingNodes():
    connectedNodes = []
    for ele in ops.getEleTags():
        for nd in ops.eleNodes(ele):
            connectedNodes.append(nd)

    definedNodes = ops.getNodeTags()

    return list(set(connectedNodes) ^ set(definedNodes))
```

APPENDICE 6 - py code 34 – Funzione chPerm

```
def chPerm():
    print('funzione chPerm')
    m = Model()
    proc_dict = m.mDict()
    permh = float(input('insert h permeability'))
    permv = float(input('insert v permeability'))
    eleTags = []
    getParamTags = []

    for i in ops.getParamTags():
        getParamTags.append(i)

    LT = len(getParamTags)
    if getParamTags == []:
        tag = 1
    else:
        tag = int(getParamTags[LT - 1]) + 1

    for i in ops.getEleTags():
        eleTags.append(int(i))

    LT2 = len(eleTags)
    for j in range(0, LT2 - 1):
        i = eleTags[j]
        ops.parameter(int(tag), 'element', i, 'hPerm')
        ops.parameter(int(tag + 1), 'element', i, 'vPerm')
        ################################################
        procId = trova_elemento(proc_dict,i)
        py = open("cubotto_exe.py", "a")
        py.write(f'if pid == {procId}:\n')
        py.write(f"    ops.parameter({int(tag)}, 'element',
{i},'hPerm')\n")
        py.close()
```

```
        TCL = open('cubotto.TCL', 'a')
        TCL.write("if {"+f' $pid == {procId}'+' }'+' {\n')
        TCL.write(f'    parameter {int(tag)} element {i}
hPerm\n')
        TCL.close()

        py = open("cubotto_exe.py", "a")
        py.write(f"    ops.parameter({int(tag + 1)}, 'element',
{i},'vPerm')\n")
        py.close()

        TCL = open('cubotto.TCL', 'a')
        TCL.write(f'    parameter {int(tag + 1)} element {i}
vPerm\n')
        TCL.close()


        ###################################################
        ops.updateParameter(int(tag), permh)
        ops.updateParameter(int(tag + 1), permv)
        ###################################################
        py = open("cubotto_exe.py", "a")
        py.write(f"    ops.updateParame-
ter({int(tag)},{permh})\n")
        py.close()

        TCL = open('cubotto.TCL', 'a')
        TCL.write(f'    updateParameter {int(tag)} {permh}\n')
        TCL.close()

        py = open("cubotto_exe.py", "a")
        py.write(f"    ops.updateParameter({int(tag +
1)},{permv})\n")
        py.close()

        TCL = open('cubotto.TCL', 'a')
        TCL.write(f'    updateParameter {int(tag+1)} {permv}\n')
        TCL.write('}\n')
        TCL.close()


        ###################################################
        tag = tag + 2
```

APPENDICE 6 - py code 35 – Funzione mTieNodes

```
def mTieNodes():
        print('funzione mTieNodes')
        m = Model()
        proc_dict = m.mDict()
        type_el_master1 = str(input('insert master plane geoEle-
ment'))
        b = m.setGeoElement(type_el_master1)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0]) ############ CORNER
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()

        type_el_slave1 = str(input('insert slave plane geoEle-
ment'))
        b = m.setGeoElement(type_el_slave1)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        g = m.nodeNumCoordVector()
        i = 0
        j = 0
        k = 0
        DOF = [1, 2]
        eqDofDic = {}
        for i in f:
            for j in g:
                z_master = i[1][2]
                z_slave = j[1][2]
                if z_master == z_slave:
                    eqDofDic[i[0]] = j[0]
```

```
                    ops.equalDOF(int(i[0]), int(j[0]), *DOF)

################################################

                    py = open("cubotto_exe.py", "a")
                    py.write(f'if pid == 0:\n')
                    py.write(f'    DOF = {DOF}\n')
                    py.write(f'    ops.equalDOF({int(i[0])},
{int(j[0])}, *DOF)\n')
                    py.close()

                    TCL = open('cubotto.TCL', 'a')
                    TCL.write("if {"+f' $pid == 0'+' }'+' {\n')
                    #TCL.write(f'    set DOF {DOF}\n')
                    #TCL.write(f'    equalDOF {int(i[0])}
{int(j[0])} {DOF[0]} {DOF[1]}\n')
                    TCL.write(f'    equalDOF {int(i[0])}
{int(j[0])} {DOF[0]}\n')
                    TCL.write('}\n')
                    TCL.close()

        b = m.setGeoElement(type_el_master1)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()

        b = m.setGeoElement(type_el_slave1)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('8-nodi')
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l = len(c[1])
        ll = len(c[1][0])
        d = m.listOfTagsij(l, ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
```

```
        g = m.nodeNumCoordVector()
        i = 0
        j = 0
        k = 0
        DOF = [1, 2]
        eqDofDic = {}
        for i in f:
            for j in g:
                z_master = i[1][2]
                z_slave = j[1][2]
                if z_master == z_slave:
                    eqDofDic[i[0]] = j[0]
                    ops.equalDOF(int(i[0]), int(j[0]), *DOF)

##################################################

                    py = open("cubotto_exe.py", "a")
                    py.write(f'if pid == 0:\n')
                    py.write(f'    DOF = {DOF}\n')
                    py.write(f'    ops.equalDOF({int(i[0])},
{int(j[0])}, *DOF)\n')
                    py.close()

                    TCL = open('cubotto.TCL', 'a')
                    TCL.write("if {"+f' $pid == 0'+' }'+' {\n')
                    #TCL.write(f'    set DOF {DOF}\n')
                    #TCL.write(f'    equalDOF {int(i[0])}
{int(j[0])} {DOF[0]} {DOF[1]}\n')
                    TCL.write(f'    equalDOF {int(i[0])}
{int(j[0])} {DOF[0]}\n')
                    TCL.write('}\n')
                    TCL.close()

##################################################
```

APPENDICE 6 - py code 36 – Funzione mStage0

```
def mStage0(gamma1,beta1,a0,a1):
    print('funzione mStage0')
    ops.model("basicBuilder", "-ndm", 3, "-ndf", 4)
    ops.updateMaterialStage('-material', 1, '-stage', 0)
    ops.updateMaterialStage('-material', 2, '-stage', 0)

    py = open("cubotto_exe.py", "a")
    py.write('ops.model("basicBuilder","-ndm",3,"-ndf",4)\n')
    py.close()

    TCL = open('cubotto.TCL', 'a')
```

```
    TCL.write('model BasicBuilder -ndm 3 -ndf 4\n')
    TCL.close()

    py = open("cubotto_exe.py", "a")
    py.write("ops.updateMaterialStage('-material', 1, '-stage',
0)\n")
    py.write("ops.updateMaterialStage('-material', 2, '-stage',
0)\n")
    py.close()

    TCL = open('cubotto.TCL', 'a')
    TCL.write('updateMaterialStage -material 1 -stage 0\n')
    TCL.write('updateMaterialStage -material 2 -stage 0\n')
    TCL.close()

    ops.constraints('Penalty', 1.e18, 1.e18)
    ops.test('NormDispIncr', 1.0e-6, 500, 1)
    ops.algorithm('KrylovNewton')
    ops.numberer('Plain')
    ops.system('UmfPack')
    ops.integrator('Newmark', gamma1, beta1)
    ops.analysis('Transient')


    py = open("cubotto_exe.py", "a")
    py.write(f"ops.constraints('Penalty', 1.e18, 1.e18)\n\
ops.test('NormDispIncr', 1.0e-3, 500, 1)\n\
ops.algorithm('ModifiedNewton')\n\
ops.numberer('ParallelRCM')\n\
ops.system('Mumps')\n\
ops.integrator('Newmark', {gamma1}, {beta1})\n\
ops.analysis('Transient')\n\
startT = tt.time()\n\
ops.analyze(1,1)\n")
    py.close()

    TCL = open('cubotto.TCL', 'a')
    TCL.write(f'constraints Transformation\n\
test NormDispIncr 1e-3 100 1\n\
algorithm ModifiedNewton\n\
numberer ParallelPlain\n\
system Mumps -ICNTL14 200\n\
integrator Newmark {gamma1} {beta1}\n\
analysis Transient\n\
analyze 5 100\n')
    TCL.close()
```

APPENDICE 6 - py code 37 – Funzione mRecDyn

```
def mRecDyn(recDT):
    print('funzione mRecDyn')
    m = Model()
    proc_dict = m.mDict()
    print('The code is under developement, please remeber to
change the 3d recorder element range in mRec functions')
    ops.recorder('Node', '-xml', 'displacement.part-$pid.out',
'-time', '-dT', recDT, '-node', *nodeList, '-dof', 1, 2,
3,'disp')
    ops.recorder('Node', '-xml', 'acceleration.part-$pid.out',
'-time', '-dT', recDT, '-node', *nodeList, '-dof', 1, 2, 3,'ac-
cel')
    ops.recorder('Node', '-xml', 'porePressure.part-$pid.out',
'-time', '-dT', recDT, '-node', *nodeList, '-dof', 4, 'vel')

    ops.recorder('Element', '-xml', 'stress.part-$pid.txt', '-
time', '-dT', recDT, '-eleRange', 1, 50000, 'material',
'1','stress')
    ops.recorder('Element', '-xml', 'strain.part-$pid.txt', '-
time', '-dT', recDT, '-eleRange', 1, 50000, 'material',
'1','strain')
    ops.recorder('Element', '-xml', 'strain.part-$pid.txt', '-
time', '-dT', recDT, '-eleRange', 1, 50000, 'material',
'1','plastic')

    ops.recorder('Element', '-xml', 'stress.part-$pid.out', '-
time', '-dT', recDT, '-eleRange', 1, 50000, 'material',
'1','stress')
    ops.recorder('Element', '-xml', 'strain.part-$pid.out', '-
time', '-dT', recDT, '-eleRange', 1, 50000, 'material',
'1','strain')
    ops.recorder('Element', '-xml', 'plastic.part-$pid.out', '-
time', '-dT', recDT, '-eleRange', 1, 50000, 'material',
'1','plastic')


    for nproc, subdict in proc_dict.items():
        if subdict == {}:
            break
        else:
            el_n_list=[]
            py = open("cubotto_exe.py", "a")
            py.write(f'if pid == {nproc}:\n')
            py.close()

            TCL = open('cubotto.TCL', 'a')
```

```
            TCL.write("if {"+f' $pid == {nproc}'+' }'+' {\n')
            TCL.close()

            for el_num,nod_list in subdict.items():
                el_n_list.append(el_num)
                max_val = int(nup.max(nod_list))
                min_val = int(nup.min(nod_list))
            ### NODI #######
            py = open("cubotto_exe.py", "a")
            py.write(f"    ops.recorder('Node','-xml','displace-
ment.part-$pid.out','-time', '-dT',recDT,'-
nodeRange',{min_val},{max_val},'-dof', 1, 2,3, 'disp')\n\
    ops.recorder('Node','-xml','acceleration.part-$pid.out','-
time', '-dT',recDT,'-nodeRange',{min_val},{max_val},'-dof', 1,
2,3, 'accel')\n\
    ops.recorder('Node','-xml','porePressure.part-$pid.out','-
time', '-dT',recDT,'-nodeRange',{min_val},{max_val},'-dof', 4,
'vel')\n")
            py.close()

            TCL = open('cubotto.TCL', 'a')
            TCL.write(f'    eval "recorder Node -xml displace-
ment.part-$pid.out -time -dT {recDT} -nodeRange {min_val}
{max_val} -dof 1 2 3 disp" \n\
    eval "recorder Node -xml velocity_{min_val}_{max_val}.part-
$pid.out -time -dT {recDT} -nodeRange {min_val} {max_val} -dof 1
2 3 vel" \n\
    eval "recorder Node -xml accelera-
tion_{min_val}_{max_val}.part-$pid.out -time -dT {recDT} -
nodeRange {min_val} {max_val} -dof 1 2 3 accel" \n\
    eval "recorder Node -xml porepres-
sure_{min_val}_{max_val}.part-$pid.out -time -dT {recDT} -
nodeRange {min_val} {max_val} -dof 4 vel" \n')
            #TCL.write('}\n')
            TCL.close()

            ##ELEMENTI###
            max_val = int(nup.max(el_n_list))
            min_val = int(nup.min(el_n_list))
            py = open("cubotto_exe.py", "a")
            py.write(f"    ops.recorder('Element','-
xml','stress.part-$pid.out','-time','-eleRange',
{min_val},{max_val},'material','1','stress')\n\
    ops.recorder('Element','-xml','gauss.part-$pid.out','-
time','-eleRange', {min_val},{max_val},'materi-
al','1','gausspoint')\n\
```

```
    ops.recorder('Element','-xml','strain.part-$pid.out','-
time','-eleRange', {min_val},{max_val},'materi-
al','1','strain')\n")
            py.close()

            TCL = open('cubotto.TCL', 'a')
            TCL.write(f'    eval "recorder Element -xml
stress_{min_val}_{max_val}.part-$pid.out -time -dT {recDT} -el-
eRange {min_val} {max_val} -material 1 stress"\n\
    eval "recorder Element -xml strain_{min_val}_{max_val}.part-
$pid.out -time -dT {recDT} -eleRange {min_val} {max_val} -mate-
rial 1 strain"\n')
            TCL.write('}\n')
            TCL.close()
```

APPENDICE 6 - py code 38 – Funzioni mNodeInfoTxt mNodeInfoCornerDat mNodeInfoDat

```
def mNodeInfoTxt():
    global nodeList

    m = Model()
    print('Attenzione chiamare con Solid l elemento di volume')
    b = m.setGeoElement('Solid')
    bb = m.getElementsVectors()
    eleNodes = bb[1]
    eleTag = bb[0]
    m.setUniqueVector(eleNodes)
    l = len(eleNodes)
    ll = len(eleNodes[0])
    d = m.listOfTagsij(l, ll)
    m.setUniqueVector(d)
    e = m.makeUnique()
    nup.savetxt('nodeInfo.txt', e)

def mNodeInfoCornerDat():
    m = Model()
    b = m.setGeoElement('Solid')
    bb = m.getElementsVectors()
    m.setUniqueVector(bb)
    c = m.nodeCornEdge('20-nodi')
    m.setUniqueVector(c[0])
    print(len(c[0]))
    print(len(c[0][0]))
    l = len(c[0])
    ll = len(c[0][0])
    d = m.listOfTagsij(l,ll)
    m.setUniqueVector(d)
    e = m.makeUnique()
```

```
    f = m.nodeNumCoordVector()
    n = open("nodeInfoCorner.dat","w")
    for i in f:
        xCoord =  i[1][0]
        yCoord =  i[1][1]
        zCoord =  i[1][2]
        n.write(f"{i[0]} {xCoord}    {yCoord}    {zCoord}\n")
    n.close()


def mNodeInfoDat():
    m = Model()
    b = m.setGeoElement('Solid')
    bb = m.getElementsVectors()
    m.setUniqueVector(bb[1])
    c = m.makeUnique()
    m.setUniqueVector(c)
    d=m.nodeNumCoordVector()
    n=open("nodeInfo.dat","w")
    for i in d:
     xCoord =  i[1][0]
     yCoord =  i[1][1]
     zCoord =  i[1][2]
     n.write(f"{i[0]} {xCoord}    {yCoord}    {zCoord}\n")
    n.close()
```

APPENDICE 6 - py code 39 – Funzione mGidFile

```
def mGIDfile():
    m = Model()
    b = m.setGeoElement('Solid')
    bb = m.getElementsVectors()
    eleNodes = bb[1]
    eleTag = bb[0]
    LET = len(bb[0])
    print(LET)

    mesh=open("mesh4GID.msh","w")
    element=open("element4GID.dat","w")
    mesh.write("MESH dimension 3 ElemType Hexahedra Nnode 20\n")
    mesh.write("Coordinates\n")
    mesh.write("#node_number   coord_x   coord_y   coord_z\n")
    m.setUniqueVector(bb[1])
    c = m.makeUnique()
    m.setUniqueVector(c)
    d=m.nodeNumCoordVector()
    for i in d:
         xCoord =  i[1][0]
```

```
          yCoord =  i[1][1]
          zCoord =  i[1][2]
    mesh.write(f"{i[0]} {xCoord}    {yCoord}    {zCoord}\n")
    mesh.write("end coordinates\n")
    mesh.write("Elements\n")

    mesh.write("# element   nodo1  nodo2   nodo3   nodo4   nodo5
nodo6   nodo7   nodo8   nodo9   nodo10   nodo11   nodo12
nodo13   nodo14   nodo15   nodo16   nodo17   nodo18   nodo19
nodo20\n")

    for i in range(0,LET):
            elem = int(eleTag[i])
            nodo1 = int(eleNodes[i][4])
            nodo2 = int(eleNodes[i][5])
            nodo3 = int(eleNodes[i][1])
            nodo4 = int(eleNodes[i][0])
            nodo5 = int(eleNodes[i][7])
            nodo6 = int(eleNodes[i][6])
            nodo7 = int(eleNodes[i][2])
            nodo8 = int(eleNodes[i][3])
            nodo9 = int(eleNodes[i][16])
            nodo10 = int(eleNodes[i][12])
            nodo11 = int(eleNodes[i][8])
            nodo12 = int(eleNodes[i][10])
            nodo13 = int(eleNodes[i][19])
            nodo14 = int(eleNodes[i][14])
            nodo15 = int(eleNodes[i][13])
            nodo16 = int(eleNodes[i][15])
            nodo17 = int(eleNodes[i][17])
            nodo18 = int(eleNodes[i][18])
            nodo19 = int(eleNodes[i][11])
            nodo20 = int(eleNodes[i][9])
            nodes_l =
[nodo1,nodo2,nodo3,nodo4,nodo5,nodo6,nodo7,nodo8,nodo9, nodo10,
nodo11,nodo12, nodo13, nodo14,nodo15,nodo16, nodo17,nodo18,
nodo19, nodo20]
            mesh.write(f"{elem} {nodo1} {nodo2} {nodo3} {nodo4}
{nodo5} {nodo6} {nodo7} {nodo8} {nodo9} {nodo10} {nodo11}
{nodo12} {nodo17} {nodo18} {nodo19} {nodo20} {nodo13} {nodo14}
{nodo15} {nodo16} \n")
            element.write(f"{elem} {nodo1} {nodo2} {nodo3}
{nodo4} {nodo5} {nodo6} {nodo7} {nodo8} {nodo9} {nodo10}
{nodo11} {nodo12} {nodo17} {nodo18} {nodo19} {nodo20} {nodo13}
{nodo14} {nodo15} {nodo16}\n")
    mesh.write("end elements")
```

```
    element.close()
    mesh.close()
```

APPENDICE 6 - py code 40 – Funzione mRegion2D

```
def mRegion20():
    print('funzione mRegion20')
    m = Model()
    Volume = str(input('inserisci l elemento geo Solid da model-
lare: '))
    materiale = int(input("inserisci il tag del materiale asso-
ciato al Volume: "))
    alphaM = float(input("inserisci il valore di alphaM: "))
    betaK = float(input("inserisci il valore di betaK: "))
    betaKinit = float(input("inserisci il valore di betaKinit:
"))
    betaKcomm = float(input("inserisci il valore di betaKcomm:
"))
    b = m.setGeoElement(Volume)
    bb = m.getElementsVectors()
    eleNodes = bb[1]
    eleTag = bb[0]
    LET = len(bb[0])

    TCL = open("cubotto.TCL","a")
    TCL.write(f'region {materiale} -ele ')

    for i in range(0,LET):
        print(f'mRegion20 - percent completed: {(i*100)/LET}')
        elem = int(eleTag[i])
        TCL = open("cubotto.TCL","a")
        TCL.write(f' {elem} ')
    TCL.close()

    TCL = open("cubotto.TCL","a")
    TCL.write(f'-rayleigh {alphaM} {betaK} {betaKinit} {be-
taKcomm} ')
    TCL.close()
```

APPENDICE 6 - py code 41 – Funzione mDefine9

```
def mDefine9():
        TCL1 = open("loads.TCL",'w')
        TCL1.close()
        TCL2 = open("removes.TCL",'w')
        TCL2.close()
        m = Model()
```

```
        b = m.setGeoElement('Solid')
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        c = m.nodeCornEdge('9-nodi')
        m.setUniqueVector(c[0])
        print(len(c[0]))
        print(len(c[0][0]))
        l = len(c[0])
        ll = len(c[0][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 2
        ndf = 3
        m.sendOpsNodes9(dim,ndf)
        ##
        m.setUniqueVector(c[1])
        print(len(c[1]))
        print(len(c[1][0]))
        l=len(c[1])
        ll=len(c[1][0])
        d = m.listOfTagsij(l,ll)
        m.setUniqueVector(d)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 2
        ndf = 2
        m.sendOpsNodes9(dim,ndf)
```

APPENDICE 6 - py code 42 – Funzione mSendOpsNode9 mSendOpsFixes9

```
def sendOpsNodes9(self,dim,ndf):
        self.dim = dim
        self.ndf = ndf
        m = Model()
        ops.model("basicBuilder", "-ndm", dim, "-ndf", ndf)
        l = len(self.uniqueVector)
        TCL = open("cubotto.TCL", "a")
        TCL.write(f'model BasicBuilder -ndm {dim} -ndf {ndf}\n')
        TCL.close()
        for i in range(0,l):
            coord = [self.uniqueVector[i][1][0],self.uniqueVec-
tor[i][1][1]]
            ops.node(int(self.uniqueVector[i][0]),*coord)
```

```
            TCL=open("cubotto.TCL","a")
            TCL.write(f'node {int(self.uniqueVector[i][0])}
{coord[0]} {coord[1]}\n')
            TCL.close()

def sendOpsFixes9(self,dim,ndf,x,y,p):
        self.dim = int(dim)
        self.ndf = int(ndf)
        self.x = int(x)
        self.y = int(y)
        self.p = int(p)
        if ndf == 3:
            ops.model("basicBuilder", "-ndm", dim, "-ndf", ndf)
            ###################################

            TCL=open("cubotto.TCL","a")
            TCL.write(f'model BasicBuilder -ndm {dim} -ndf
{ndf}\n')
            TCL.close()

            ###################################
            l = len(self.uniqueVector)
            print(self.uniqueVector)
            for i in range(0, l):
                ###################################
                coord = [self.uniqueVector[i][1][0],self.unique-
Vector[i][1][1],self.uniqueVector[i][1][2]]

                TCL=open("cubotto.TCL","a")
                print(f'fix function dim 3 - percent completed:
{(i*100)/l}')
                TCL.write(f'remove sp {int(self.uniqueVec-
tor[i][0])} 1\nremove sp {int(self.uniqueVector[i][0])} 2\nre-
move sp {int(self.uniqueVector[i][0])} 3\n')
                TCL.write(f'fix {int(self.uniqueVector[i][0])}
{x} {y} {p}\n')
                TCL.write(f'puts "nodo {int(self.uniqueVec-
tor[i][0])} remove and fixed ---> ok"\n')
                TCL.close()
                ###################################
                ops.remove('sp', int(self.uniqueVector[i][0]),1)
                ops.remove('sp', int(self.uniqueVector[i][0]),2)
                ops.remove('sp', int(self.uniqueVector[i][0]),3)
                ops.fix(int(self.uniqueVector[i][0]), x, y, p)
        else:
            ops.model("basicBuilder", "-ndm", dim, "-ndf", ndf)
            ###################################
```

```
            TCL=open("cubotto.TCL","a")
            TCL.write(f'model BasicBuilder -ndm {dim} -ndf
{ndf}\n')
            TCL.close()


            ###################################
            l = len(self.uniqueVector)
            print(self.uniqueVector)
            for i in range(0, l):
                ops.remove('sp', int(self.uniqueVector[i][0]),1)
                ops.remove('sp', int(self.uniqueVector[i][0]),2)
                ops.remove('sp', int(self.uniqueVector[i][0]),3)
                ops.fix(int(self.uniqueVector[i][0]), x, y)
                ###################################

                coord = [self.uniqueVector[i][1][0],self.unique-
Vector[i][1][1],self.uniqueVector[i][1][2]]
                TCL=open("cubotto.TCL","a")
                print(f'fix function dim 3 - percent completed:
{(i*100)/l}')
                TCL.write(f'remove sp {int(self.uniqueVec-
tor[i][0])} 1\nremove sp {int(self.uniqueVector[i][0])} 2\n')
                TCL.write(f'fix {int(self.uniqueVector[i][0])}
{x} {y}\n')
                TCL.write(f'puts "nodo {int(self.uniqueVec-
tor[i][0])} remove and fixed ---> ok"\n')
                TCL.close()
```

APPENDICE 6 - py code 43 – Funzione mFix9

```
def mFix9():
    print(g2o.get_physical_groups_map(Gmsh.model))
    m = Model()
    fixed = mgetFixedCoord()
    type_el = str(input('inserisci l elemento geo da vincolare:
'))
    print('inserire 1 per vincolo on 0 per vincolo off')
    DofL = []
    x = int(input('x: 1 0n - 0 off: '))
    f_x = 0
    if x == 1:
        f_x = 1
        DofL.append(f_x)
    else:
        f_x = 0
```

```
y = int(input('y: 1 0n - 0 off: '))
f_y = 0
if y == 1:
    f_y = 2
    DofL.append(f_y)
else:
    f_y = 0

p = int(input('p: 1 0n - 0 off: '))
f_p = 0
if p == 1:
    f_p = 3
    DofL.append(f_p)
else:
    f_p = 0

# Vettore dof Locale ok
b = m.setGeoElement(type_el)
bb = m.getElementsVectors()
# 0 element tag - 1 node tag
c = m.setUniqueVector(bb[1])
d = m.makeUnique()
d1 = []
d2 = []
for i in d:
    dnr = ops.nodeReaction(int(i))
    l = len(dnr)
    if l == 2:
        d1.append(i)

e = m.setUniqueVector(d1)
f = m.nodeNumCoordVector()
ff = []

for i in f:
    flag = 0
    mem = []
    if fixed == [[], [], []]:
        ff.append(i)
        continue
    for j in fixed[1]:
        if flag == 1:
            break
        if i[1] == j:
            for k in fixed[2]:
                if flag == 1:
                    break
```

```
                    for kk in k:
                        if flag == 1:
                            break
                        for l in DofL:
                            if kk == l:
                                flag = 1
                                break
        if flag == 0:
            ff.append(i)

    m.setUniqueVector(ff)

    dim = 2
    ndf = 2
    p2 = 0

    m.sendOpsFixes9(dim, ndf, x, y, p2)

    for i in d:
        dnr = ops.nodeReaction(int(i))
        l = len(dnr)
        if l == 3:
            d2.append(i)

    e = m.setUniqueVector(d2)
    f = m.nodeNumCoordVector()
    ff = []

    for i in f:
        flag = 0
        mem = []
        if fixed == [[], [], []]:
            ff.append(i)
            continue
        for j in fixed[1]:
            if flag == 1:
                break
            if i[1] == j:
                for k in fixed[2]:
                    if flag == 1:
                        break
                    for kk in k:
                        if flag == 1:
                            break
                        for l in DofL:
                            if kk == l:
                                flag = 1
```

```
                                    break
        if flag == 0:
            ff.append(i)

    m.setUniqueVector(ff)

    dim = 2
    ndf = 3
    m.sendOpsFixes9(dim, ndf, x, y, p)
```

APPENDICE 6 - py code 44 – Funzione mGenFem9

```
def mGenFem9(rho):
    Volume = str(input('inserisci l elemento geo Solid da model-
lare: '))
    g = -9.81
    gamma_tot = g*rho
    print(f'gamma_tot: {gamma_tot}')
    m = Model()
    materiale = int(input("inserisci il tag del materiale asso-
ciato al Volume: "))
    b = m.setGeoElement(Volume)
    m.setUniqueVector(b)
    bb = m.getElementsVectors()
    eleNodes = bb[1][0]
    eleTag = bb[0][0]
    LET = len(bb[0][0])
    k = 0
    for i in range(0, LET):
        print(f'mGenFem9 - percent completed: {(i * 100) /
LET}')
        elem = int(eleTag[i])
        k = i*9
        nodo1 = int(eleNodes[k])
        nodo2 = int(eleNodes[k+1])
        nodo3 = int(eleNodes[k+2])
        nodo4 = int(eleNodes[k+3])
        nodo5 = int(eleNodes[k+4])
        nodo6 = int(eleNodes[k+5])
        nodo7 = int(eleNodes[k+6])
        nodo8 = int(eleNodes[k+7])
        nodo9 = int(eleNodes[k+8])
        nodes_l_first = [nodo1, nodo2, nodo3, nodo4, nodo5,
nodo6, nodo7, nodo8, nodo9]

        mem = []
        coordinate = []
        coordinate_x = []
```

```
        coordinate_y = []
        for j in nodes_l_first:
            x = Gmsh.model.mesh.getNode(int(j))[0][0]
            y = Gmsh.model.mesh.getNode(int(j))[0][1]
            mem = [j,[x,y]]
            coordinate.append(mem)
            coordinate_x.append(x)
            coordinate_y.append(y)
        #print(f'coordinate: {coordinate}')

        x = nup.array(coordinate_x)
        y = nup.array(coordinate_y)

        x_min = nup.min(x)
        #print(f'x_min: {x_min}')
        x_max = nup.max(x)
        #print(f'x_max: {x_max}')
        y_min = nup.min(y)
        #print(f'y_min: {y_min}')
        y_max = nup.max(y)
        #print(f'y_max: {y_max}')

        #nodi spigoli 1 2 3 4
        coordinate2=[]
        for i in range(4):
            coordinate2.append(coordinate[i])

        nodes_l = [0,0,0,0,0,0,0,0,0]
        s = []
        s = sorted(coordinate2, key = lambda x: x[1][1])
        select = []
        for i in range(2):
            select.append(s[i])
        s1 = []
        s1 = sorted(select,key = lambda x: x[1][0])

        select2 = []
        s2 = []
        for i in range(2,4):
            select2.append(s[i])
        s3 = []
        s3 = sorted(select2,key = lambda x: x[1][0])


        nodes_l[0] = s1[0][0]
        nodes_l[1] = s1[1][0]
        nodes_l[3] = s3[0][0]
```

```
        nodes_l[2] = s3[1][0]

        coordinate_mid=[]
        for i in range(4,8):
            coordinate_mid.append(coordinate[i])


        s4 = []
        s4 = sorted(coordinate_mid, key = lambda x: x[1][1])
        select = []
        for i in range(2):
            select.append(s4[i])
        print(f'ordine solo secondo y: {s4}')
        #print(f'select_mid: {select}')
        nodes_l[4]=s4[0][0]

        coordinate_mid2 = []
        for i in range(1,3):
            coordinate_mid2.append(s4[i])
        #print(f'coordinate_mid2: {coordinate_mid2}')

        s5 = []
        s5 = sorted(coordinate_mid2,key = lambda x: x[1][0])
        print(f'vettore dei tre moschettieri: {s5}')
        nodes_l[5]=s5[1][0]
        print(f'nodo 6: {s5[1][0]}')
        nodes_l[7]=s5[0][0]
        print(f'nodo 7: {s5[0][0]}')
        nodes_l[6]=s4[3][0]
        print(f'nodo 8: {s4[3][0]}')

        nodes_l[8] = nodes_l_first[8]

        print(f'nodo9: {nodes_l[8]}')


        print(f'elemento: {elem}, nodi: {nodes_l}')
        ops.element('9_4_QuadUP', elem, *nodes_l, 1, 1, 2.2e6,
1.0, 1.0, 1.0,0.0,g)
        TCL = open("cubotto.TCL","a")
        TCL.write(f'element 9_4_QuadUP {elem} {nodes_l[0]}
{nodes_l[1]} {nodes_l[2]} {nodes_l[3]} {nodes_l[4]} {nodes_l[5]}
{nodes_l[6]} {nodes_l[7]} {nodes_l[8]} {materiale} 1 2.2e6 1 1.0
1.0 0.0 {g}\n')
        TCL.close()
```

APPENDICE 6 - py code 45 – Funzione mDash9

```
def mDash9():
    ops.model("basicBuilder", "-ndm", 2, "-ndf", 2)
    m = Model()
    b = m.setGeoElement('Solid')
    bb = m.getElementsVectors()
    eleNodes = bb[1][0]
    eleTag = bb[0][0]
    print(f"eleTag = {eleTag}")
    m.setUniqueVector(eleNodes)
    cc = m.makeUnique()
    max_value = nup.max(cc)
    min_value = nup.min(cc)
    dashFree = int(max_value + 1)
    print(f'dashFree = {dashFree}')
    dashFix = int(max_value + 2)
    print(f'dashFix = {dashFix}')
    x = float(input('insert x coord of dashpot in format 0.0'))
    y = float(input('insert y coord of dashpot in format 0.0'))
    ops.node(dashFree, x, y)
    ops.node(dashFix, x, y)
    ops.fix(dashFix, 1, 1)
    f_x = int(input('Free node - insert 1 to constrain on - 0 to
constrain off on dir x'))
    f_y = int(input('Free node - insert 1 to constrain on - 0 to
constrain off on dir y'))
    ops.fix(int(dashFree), f_x, f_y)
    nodo_mesh = int(input('write the mesh node tag who want to
assigned dashpot'))
    ops.equalDOF(int(nodo_mesh), int(dashFree), 1)
    colArea = float(input('conn. area of dashpot'))
    rockVS = float(input('Bedrock VS'))
    rockDen = float(input('Bedrock density'))
    dashpotCoeff = rockVS * rockDen * colArea
    tag = int(input('insert uniaxialMaterial tag (> n. geo-mate-
rial)'))
    alpha = 1
    ops.uniaxialMaterial('Viscous', tag, dashpotCoeff, alpha)
    m.setUniqueVector(eleTag)
    max_tag = nup.max(eleTag)
    var_dir = int(input('dof of dashpot -dir: 1 along x, 2 along
xy, 3 along y'))
    if var_dir == 1:
        dir = [1]
    elif var_dir == 2:
        dir = [1, 2]
    elif var_dir == 3:
```

```
        dir = [2]
    print(f"zeroLength: tag {max_tag + 1} dashFix {dashFix}
dashFree {dashFree} dir {dir}")
    ops.element('zeroLength', int(max_tag + 1), int(dashFix),
int(dashFree), '-mat', tag, '-dir', *dir)

    TCL=open("cubotto.TCL","a")
    TCL = open('cubotto.TCL', 'a')
    TCL.write('model BasicBuilder -ndm 2 -ndf 2\n')
    TCL.write(f'node {dashFree} {x} {y}\n')
    TCL.close()


    TCL=open("cubotto.TCL","a")
    TCL.write(f'node {dashFix} {x} {y} \n')
    TCL.close()


    TCL=open("cubotto.TCL","a")
    TCL.write(f'fix {dashFix} 1 1 \n')
    TCL.close()


    TCL=open("cubotto.TCL","a")
    TCL.write(f'fix {dashFree} {f_x} {f_y} \n')
    TCL.close()

    TCL=open("cubotto.TCL","a")
    TCL.write(f'equalDOF {int(nodo_mesh)} {int(dashFree)} 1\n')
    TCL.close()

    ###################################################
    TCL=open("cubotto.TCL","a")
    TCL.write(f'uniaxialMaterial Viscous {tag} {dashpotCoeff}
{alpha}\n')
    TCL.close()

    ###################################################
    TCL=open("cubotto.TCL","a")
    #TCL.write(f'set dir {dir}')
    TCL.write(f'set dir 1\n')
    TCL.write(f'element zeroLength  {int(max_tag+1)} {int(dash-
Fix)} {int(dashFree)} -mat {tag} -dir $dir\n')
    TCL.close()

    print(f'colArea: {colArea} dashpotCoeff: {dashpotCoeff} nodo
mesh: {nodo_mesh}')
```

```
    return [colArea, dashpotCoeff, nodo_mesh]
```

APPENDICE 6 - py code 46 – Funzione mDiaf9

```
def mDiaf9():
    m = Model()

    print('equalDof sia in x che in y')
    ops.model("basicBuilder", "-ndm", 2, "-ndf", 2)
    nodo_mesh = int(input('inserisci il nodo selezionato per il
dashpot o il punto master: '))
    el = str(input('inserisci la linea da irrigidire: '))
    b = m.setGeoElement(el)
    bb = m.getElementsVectors()
    m.setUniqueVector(bb)
    d = bb[1]
    m.setUniqueVector(d)
    e = m.makeUnique()
    for i in e:
        if i == nodo_mesh:
            continue
        else:
            ops.equalDOF(int(nodo_mesh), int(i), 1)
            ops.equalDOF(int(nodo_mesh), int(i), 2)


            TCL = open("cubotto.TCL", "a")
            TCL.write(f'equalDOF {int(nodo_mesh)} {int(i)} 1\n')
            TCL.write(f'equalDOF {int(nodo_mesh)} {int(i)} 2\n')
            TCL.close()
```

APPENDICE 6 - py code 47 – Funzione mTieNodes9

```
def mTieNodes9():
    m = Model()
    type_el_master1 = str(input('insert master plane geoEle-
ment'))
    b = m.setGeoElement(type_el_master1)
    bb = m.getElementsVectors()
    # 0 element tag - 1 node tag
    c = m.setUniqueVector(bb[1])
    d = m.makeUnique()
    m.setUniqueVector(d)
    e = m.makeUnique()
    f = m.nodeNumCoordVector()

    type_el_slave1 = str(input('insert slave plane geoElement'))
    b = m.setGeoElement(type_el_slave1)
    bb = m.getElementsVectors()
```

```
    m.setUniqueVector(bb)
    c = m.setUniqueVector(bb[1])
    d = m.makeUnique()
    m.setUniqueVector(d)
    e = m.makeUnique()
    g = m.nodeNumCoordVector()
    i = 0
    j = 0
    k = 0
    DOF = [1, 2]
    eqDofDic = {}
    for i in f:
        for j in g:
            y_master = i[1][1]
            y_slave = j[1][1]
            if y_master == y_slave:
                eqDofDic[i[0]] = j[0]
                ops.equalDOF(int(i[0]), int(j[0]), *DOF)
                ################################################

                TCL = open('cubotto.TCL', 'a')
                TCL.write(f'set DOF1 {DOF[0]}\n')
                TCL.write(f'set DOF2 {DOF[1]}\n')
                TCL.write(f'equalDOF {int(i[0])} {int(j[0])}
$DOF1 $DOF2\n')
                TCL.close()

################################################
```

APPENDICE 6 - py code 48 – Funzione mStage02D

```
def mStage02D(gamma1,beta1,a0,a1):
    print('##################GRAVITY ELASTIC LOAD-
ING#####################################')
    ops.model("basicBuilder", "-ndm", 2, "-ndf", 3)

    ops.updateMaterialStage('-material', 1, '-stage', 0)
    ops.constraints('Penalty', 1.e18, 1.e18)
    ops.test('NormDispIncr', 1.0e-6, 500, 1)

    ops.algorithm('KrylovNewton')
    ops.numberer('Plain')
    ops.system('ProfileSPD')
    ops.integrator('Newmark', gamma1, beta1)
    ops.analysis('Transient')

    startT = tt.time()
```

```
    TCL = open('cubotto.TCL', 'a')
    TCL.write('model BasicBuilder -ndm 2 -ndf 3\n')
    TCL.write(f'constraints Penalty 1.0E16 1.0E16\n\
test NormDispIncr 1e-3 100 1\n\
algorithm ModifiedNewton\n\
numberer RCM\n\
system Mumps\n\
integrator Newmark {gamma1} {beta1}\n\
#rayleigh {a0} {a1} 0.0 0.0\n\
analysis Transient\n\
analyze 10 0.1\n')
    TCL.close()

    return startT
```

APPENDICE 6 - py code 49 – mSubReacForce9 mRemove9 function

```
def subReacForce9(self):
        print('WARNING::::::ATTENZIONE AI GRADI DI LIBERTA')
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()
        type_el = str(input('what geoElement do you want to un-
constrain'))
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        # 0 element tag - 1 node tag
        c = m.setUniqueVector(bb[1])
        d = m.makeUnique()
        d1 = []
        d2 = []

        for i in d:
            dnr = ops.nodeReaction(int(i))
            l = len(dnr)
            if l == 3:
                d1.append(i)
            elif l == 2:
                d2.append(i)
        m.setUniqueVector(d1)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)

        dim = 2
        ndf = 3
        reacV1 = {}
        mem = 0
```

```
        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)

            reacV1[int(i[0])] = mem
            #ops.load(int(i[0]), mem[0], mem[1], 0.0)
            ops.load(int(i[0]), mem[0], 0.0, 0.0)
            ops.remove('sp',int(i[0]),1)
            ops.remove('sp',int(i[0]),2)
            #ops.remove('sp',int(i[0]),3)
            TCL1 = open("loads.TCL",'a')
            #TCL1.write(f'load {int(i[0])} {mem[0]} {mem[1]}
0.0\n')
            TCL1.write(f'load {int(i[0])} {mem[0]} 0.0 0.0\n')
            TCL1.close()

            TCL2 = open("removes.TCL",'a')
            TCL2.write(f'remove sp {int(i[0])} 1\nremove sp
{int(i[0])} 2\nremove sp {int(i[0])} 3\n')
            TCL2.close()

        m.setUniqueVector(d2)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 2
        ndf = 2
        reacV2 = {}
        mem = 0
        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)

            reacV2[int(i[0])] = mem
            #ops.load(int(i[0]), mem[0], mem[1], mem[2])
            #ops.load(int(i[0]), mem[0], mem[1])
            ops.load(int(i[0]), mem[0], 0.0)
            ops.remove('sp',int(i[0]),1)
            ops.remove('sp',int(i[0]),2)
            ops.remove('sp',int(i[0]),3)

            TCL1 = open("loads.TCL",'a')
            #TCL1.write(f'load {int(i[0])} {mem[0]} {mem[1]}\n')


            TCL1.write(f'load {int(i[0])} {mem[0]} 0.0\n')
            TCL1.close()

            TCL2 = open("removes.TCL",'a')
```

```
            TCL2.write(f'remove sp {int(i[0])} 1\nremove sp
{int(i[0])} 2\n')
            TCL2.close()

def mRemove9(self):
        print(g2o.get_physical_groups_map(Gmsh.model))
        m = Model()
        type_el = str(input('what geoElement do you want to un-
constrain'))
        b = m.setGeoElement(type_el)
        bb = m.getElementsVectors()
        m.setUniqueVector(bb)
        # 0 element tag - 1 node tag
        c = m.setUniqueVector(bb[1])
        d = m.makeUnique()
        d1 = []
        d2 = []

        for i in d:
            dnr = ops.nodeReaction(int(i))
            l = len(dnr)
            if l == 3:
                d1.append(i)
            elif l == 2:
                d2.append(i)
        m.setUniqueVector(d1)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)

        dim = 2
        ndf = 3
        reacV1 = {}
        mem = 0
        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)
            ops.model("basicBuilder","-ndm",2,"-ndf",3)
            reacV1[int(i[0])] = mem
            #ops.load(int(i[0]), mem[0], mem[1], 0.0)
            #ops.load(int(i[0]), mem[0], 0.0, 0.0)
            ops.remove('sp',int(i[0]),1)
            ops.remove('sp',int(i[0]),2)
            #ops.remove('sp',int(i[0]),3)


            TCL2 = open("removes.TCL",'a')
```

```
            #TCL2.write(f'remove sp {int(i[0])} 1\nremove sp
{int(i[0])} 2\nremove sp {int(i[0])} 3\n')
            TCL2.write(f'remove sp {int(i[0])} 1\nremove sp
{int(i[0])} 2\n')
            TCL2.close()

        print('\nEdge\n')

        m.setUniqueVector(d2)
        e = m.makeUnique()
        f = m.nodeNumCoordVector()
        m.setUniqueVector(f)
        dim = 2
        ndf = 2
        reacV2 = {}
        mem = 0
        for i in f:
            mem = ops.nodeReaction(int(i[0]), -1)
            ops.model("basicBuilder","-ndm",2,"-ndf",2)
            reacV2[int(i[0])] = mem
            #ops.load(int(i[0]), mem[0], mem[1], mem[2])
            #ops.load(int(i[0]), mem[0], mem[1])
            #ops.load(int(i[0]), mem[0], 0.0)
            ops.remove('sp',int(i[0]),1)
            ops.remove('sp',int(i[0]),2)


            TCL2 = open("removes.TCL",'a')
            TCL2.write(f'remove sp {int(i[0])} 1\nremove sp
{int(i[0])} 2\n')
            TCL2.close()
```

APPENDICE 6 - py code 50 – Funzione mEleNodeTag2D

```
def meleNodeTag2D():
    m = Model()
    b = m.setGeoElement('Solid')
    m.setUniqueVector(b)
    bb = m.getElementsVectors()
    eleTag = bb[0]
    eleNodes = []
    l = len(bb[0][0])
    k = 0
    for i in range(0, l):
        mem = []
        for j in range(0, 9):
            mem.append(bb[1][0][k])
            k = k + 1
```

```
            if len(mem) == 9:
                m = [i, mem]
                eleNodes.append(m)
    return eleTag,eleNodes
```

APPENDICE 6 - py code 51 – Funzione mGidFile2D

```
def mGIDfile2D():
    m = Model()
    b = m.setGeoElement('Solid')
    m.setUniqueVector(b)
    bb = m.getElementsVectors()
    eleTag_all = bb[1][0]
    [eleTag, eleNodes] = meleNodeTag2D()

    onlyNodes = []
    for i in eleNodes:
        node = i[0]
        onlyNodes.append(node)

    LET = len(bb[0][0])
    print(LET)

    mesh = open("mesh4GID.msh", "w")
    element = open("element4GID.dat", "w")
    mesh.write("MESH dimension 2 ElemType Quadrilateral Nnode
9\n")  # camuffato come elemento a 8 nodi.
    mesh.write("Coordinates\n")
    mesh.write("#node_number   coord_x   coord_y   coord_z\n")

    m.setUniqueVector(bb[1][0])
    c = m.makeUnique()
    m.setUniqueVector(c)
    d = m.nodeNumCoordVector()

    for i in d:
        xCoord = i[1][0]
        yCoord = i[1][1]
        zCoord = i[1][2]

        mesh.write(f"{i[0]} {xCoord}    {yCoord}    {zCoord}\n")
    mesh.write("end coordinates\n")
    mesh.write("Elements\n")
    mesh.write("# element   nodo1  nodo2   nodo3   nodo4   nodo5
nodo6   nodo7   nodo8   nodo9  \n")

    for i in range(0, LET):
        elem = int(eleNodes[i][0]) + 1
```

```
        nodo2 = int(eleNodes[i][1][0])
        nodo3 = int(eleNodes[i][1][1])
        nodo4 = int(eleNodes[i][1][2])
        nodo1 = int(eleNodes[i][1][3])
        nodo6 = int(eleNodes[i][1][4])
        nodo7 = int(eleNodes[i][1][5])
        nodo8 = int(eleNodes[i][1][6])
        nodo5 = int(eleNodes[i][1][7])
        nodo9 = int(eleNodes[i][1][8])

        nodes_l = [nodo1, nodo2, nodo3, nodo4, nodo5, nodo6,
nodo7, nodo8, nodo9]
        mesh.write(
            f"{elem} {nodo1} {nodo2} {nodo3} {nodo4} {nodo5}
{nodo6} {nodo7} {nodo8} {nodo9}  \n")
        element.write(
            f"{elem} {nodo1} {nodo2} {nodo3} {nodo4} {nodo5}
{nodo6} {nodo7} {nodo8} {nodo9} \n")
    mesh.write("end elements")

    element.close()
    mesh.close()
```

APPENDICE 6 - py code 52 – Funzione OPENSEESTCLRead

```
def OPENSEESTCLRead(TCLFile, startswith, n_cols):
    fileInfo = []
    TCLfile = open(TCLFile, 'r')
    for line in TCLfile:
        if line[:len(startswith)] == startswith:
            args = line.split()
            for i in range(0, n_cols):
                fileInfo.append(args[i])

    TCLfile.close()
    return np.array(fileInfo).reshape((-1, n_cols))
```

APPENDICE 6 - py code 53 – Funzione OPENSEESOutputRead

```
def OPENSEESOutputRead(outFile):
    skip = 0
    outData = pd.read_csv(outFile, delim_whitespace=True,
header=None,
                          comment='<', skiprows=skip)
    return np.array(outData)
```

APPENDICE 6 - py code 54 – Funzione step_static

```
def step_static(TCLFile):
    integratorInfo_LoadControl = OPENSEESTCLRead(TCLFile, 'set
numSteps', 3)
```

```
    step_static = integratorInfo_LoadControl[:,2]
    return step_static
```

APPENDICE 6 - py code 55 – Funzione out_response

```
def out_response(outFile, steps,ndm,type):
    outrespFile=OPENSEESOutputRead(outFile)
    if ndm == 3:
        if type=='all':
            response = outrespFile[:, 1:].reshape(-1, 3)
    elif ndm == 2:
        if type=='all':
            response_xy = outrespFile[:, 1:].reshape(-1, 2)
            z_res = np.repeat(0, len(response_xy[:, 0]))
            response = np.column_stack([response_xy, z_res])
    return response
```

APPENDICE 6 - py code 56 – Funzione NodeCoords

```
def NodeCoords(TCLFile):
    ndm = 3
    if ndm==3:
        nodeInfo = OPENSEESTCLRead(TCLFile, 'node', 5)
        initNodeCoords = nodeInfo[:, 2:5].astype(float)
    elif ndm==2:

        nodeInfo = OPENSEESTCLRead(TCLFile, 'node', 4)
        initNodeCoord = nodeInfo[:, 2:4].astype(float)
        initNodeCoords1=[]
        z = np.repeat(0, len(initNodeCoord[:,0]))
        for i in range (len(initNodeCoord[:,0])):

            initNodeCoords1.append((initNodeCoord[i]).tolist())
        initNodeCoords=np.column_stack([initNodeCoords1, z])
    return (initNodeCoords)
```

APPENDICE 6 - py code 57 – Funzione quad_cell

```
def quad_cell(elemList, nodeList):
    nodeiRow = []
    print(f'elem_list: {elemList}')
    print(f'node_list: {nodeList}')
    for i in range(len(elemList[:, 0])):
        nodeiRow.append((9, int(np.argwhere(nodeList[:, 1] ==
elemList[i, 3])),
                         int(np.argwhere(nodeList[:, 1] == el-
emList[i, 4])),
                         int(np.argwhere(nodeList[:, 1] == el-
emList[i, 5])),
                         int(np.argwhere(nodeList[:, 1] == el-
emList[i, 6])),
```

```
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 7])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 8])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 9])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 10])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 11])))))

    print(f'quad_cell: {np.array(nodeiRow)}')
    return np.array(nodeiRow)
```

APPENDICE 6 - py code 58 – Funzione cell_type_quad cell_type_quad9

```
def cell_type_quad(elemList):
    x = np.repeat(9, (len(elemList[:, 0])))
    return x

def cell_type_quad9(elemList):
    x = np.repeat(28, (len(elemList[:, 0])))
    return x
```

APPENDICE 6 - py code 59 – Funzione hex_cell

```
def hex_cell(elemList, nodeList):
    nodeiRow = []
    print(f'elem_list: {elemList}')
    print(f'node_list: {nodeList}')
    for i in range(len(elemList[:, 0])):
        nodeiRow.append((20, int(np.argwhere(nodeList[:, 1] ==
elemList[i, 3])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 4])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 5])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 6])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 7])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 8])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 9])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 10])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 11])),
```

```
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 12])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 13])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 14])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 15])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 16])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 17])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 18])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 19])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 20])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 21])),
                            int(np.argwhere(nodeList[:, 1] == el-
emList[i, 22])))))

    print(f'hex_cell: {np.array(nodeiRow)}')
    return np.array(nodeiRow)
```

APPENDICE 6 - py code 60 – Funzione cell_type_hex20

```
def cell_type_hex20(elemList):
    x = np.repeat(25, (len(elemList[:, 0])))
    return x
```

APPENDICE 6 - py code 61 – Funzione calcola modulo spostamenti

```
def calcola_modulo_spostamenti(t):
            inizio = 0
            fine = 0
            n = int(t/0.01)
            if n == 0:
                inizio = 0
                fine = int(inizio+mesh.n_points)
            else:
                inizio = int(n*mesh.n_points)
                fine = int(inizio+mesh.n_points)
            spostamenti = def_xx[inizio:fine]
            print(spostamenti)
            return spostamenti

t_init = 0.0
```

```
modulo_spostamenti = calcola_modulo_spostamenti(t_init)
mesh['spostamenti_x'] = modulo_spostamenti[:,0]

print(f'modulo_spostamenti_x: {modulo_spostamenti[:,0]}')

p = pv.Plotter()

p.add_mesh(mesh, scalars='spostamenti_x', cmap='jet', opac-
ity=1.0, clim=[-0.01, 0.01], name='mesh', show_edges = True)
state_zero = mesh.points
p.show_axes
p.camera_position = 'iso'

p.open_gif('animation.gif')
nframe = 250
dT_nframe = 0.1
t_durata = nframe*dT_nframe
time_frame = np.arange(0.0,t_durata,dT_nframe)
for t in time_frame:
    scala = 20.0

    nuovo_modulo = calcola_modulo_spostamenti(t)
    mesh['spostamenti_x'] = nuovo_modulo[:,0]
    vec_sum = np.column_stack([nuovo_modulo[:,0]*scala,nuovo_mo-
dulo[:,1]*scala,nuovo_modulo[:,2]*scala])
    mesh.points = state_zero + vec_sum
    label_coords = mesh.points + [0, 0, 0.02]
    point_labels = [f'{i}' for i in range(mesh.n_points)]
    p.add_point_labels(label_coords, point_labels, name='label',
font_size=25, point_size=20)

    p.write_frame()
p.close()

p.clear()

sargs= dict(interactive=True)
cells = hex_cell(elemInfo_hex20, nodeInfo)
cell_type = (cell_type_hex20(elemInfo_hex20))
print(f'celltype: {cell_type}')
mesh = pv.UnstructuredGrid( cells, cell_type, nodeCoord)
n_points = mesh.n_points

p = pv.Plotter()

t_init = 0.0
modulo_spostamenti = calcola_modulo_spostamenti(t_init)
```

```
mesh['spostamenti_x'] = modulo_spostamenti[:,0]

p.add_mesh(mesh, scalar_bar_args=sargs, scalars='spostamenti_x',
clim = [-0.01, 0.01], cmap='jet', opacity=1.0, name='mesh',
show_edges = True)

label_coords = mesh.points + [0, 0, 0.02]
point_labels = [f'Point {i}' for i in range(mesh.n_points)]
p.add_point_labels(label_coords, point_labels, name='label',
                   font_size=25, point_size=20)

mesh_deformata = mesh.copy()
p.add_mesh(mesh_deformata, scalar_bar_args=sargs, sca-
lars='spostamenti_x', clim = [-0.01, 0.01], cmap='jet', opac-
ity=1.0, name='mesh', show_edges = True)
```

APPENDICE 6 - py code 62 – Funzione update

```
def update(t):

            scala = 10.0
            p.remove_actor('label')
            nuovo_modulo = calcola_modulo_spostamenti(t)
            mesh_deformata['spostamenti_x'] = nuovo_modulo[:,0]
            vec_sum = np.column_stack([nuovo_mo-
dulo[:,0]*scala,nuovo_modulo[:,1]*scala,nuovo_mo-
dulo[:,2]*scala])
            mesh.points = state_zero + vec_sum
            mesh_deformata.points = mesh.points + vec_sum*scala
            label_coords = mesh_deformata.points + [0, 0, 0.02]
            p.add_point_labels(label_coords, point_labels,
name='label', font_size=25, point_size=20)
            p.show_axes()
```

APPENDICE 6 - py code 63 – Codice per plot 2D con PyVista

```
file = "displacement.part-0.xml"
file2 = "displacement.part-1.xml"
file3 = "displacement.part-2.xml"
file4 = "displacement.part-3.xml"

import xml.etree.ElementTree as xtd
import io
import numpy as np
#import respSpectra as rsp
import matplotlib.pyplot as plt
import pandas as pd
import pyvista as pv
```

```
def OPENSEESTCLRead(TCLFile, startswith, n_cols):
    fileInfo = []
    TCLfile = open(TCLFile, 'r')
    for line in TCLfile:
        if line[:len(startswith)] == startswith:
            args = line.split()
            for i in range(0, n_cols):
                fileInfo.append(args[i])

    TCLfile.close()
    return np.array(fileInfo).reshape((-1, n_cols))

def quad_cell(elemList, nodeList):
    nodeiRow = []
    print(f'elem_list: {elemList}')
    print(f'node_list: {nodeList}')
    for i in range(len(elemList[:, 0])):
        nodeiRow.append((4, int(np.argwhere(nodeList == int(el-
emList[i, 3]))),
                            int(np.argwhere(nodeList == int(el-
emList[i, 4]))),
                            int(np.argwhere(nodeList == int(el-
emList[i, 5]))),
                            int(np.argwhere(nodeList == int(el-
emList[i, 6])))))

    return np.array(nodeiRow)

def cell_type_quad(elemList):
    x = np.repeat(9, (len(elemList[:, 0])))
    return x

def calcola_modulo_spostamenti(t,vec_def):
            '''calcola l'intervallo dei valori da prendere nel
vettore degli spostamenti'''
            # Esempio: i vettori di spostamento oscillano nel
tempo
            inizio = 0
            fine = 0
            n = int(t/0.01)
            if n == 0:
                inizio = 0
                fine = int(inizio+mesh.n_points)
            else:
                inizio = int(n*mesh.n_points)
                fine = int(inizio+mesh.n_points)
            spostamenti = vec_def[inizio:fine]
```

```
            return spostamenti

def update(t):
            sargs= dict(interactive=True)

            state_zero = mesh.points
            #point_labels = [f'{i}' for i in
range(mesh.n_points)]

            scala = 1000.0
            # Rimuove tutti gli oggetti dal plotter
            p.remove_actor('mesh')
            p.remove_actor('label')

            vect_zero = np.repeat(0,len(nuovo_modulo[:]))
            vec_sum = np.column_stack([nuovo_mo-
dulo,vect_zero,vect_zero])
            mesh_deformata.points = mesh.points + vec_sum*scala

            p.add_mesh(mesh_deformata, scalar_bar_args=sargs,
scalars='modulo_spostamenti',  cmap='jet', opacity=1.0,
name='mesh', show_edges = True)
            camera = p.camera_position
            print(f'posizione telecamera: {camera}')
            p.show_axes()

def trova_nodo(dicto, dictTag, x, y, z):
    output = 0
    jj = 0
    i = 0
    for node,coord in dicto.items():
        if x == coord['x']:
            if y == coord['y']:
                if z == coord['z']:
                    output = int(node)
                    break
    #output = nup.unique(nup.array(output, dtype=int))
    print(f'il nodo che cerchi è il numero: {output}')
    for i in dictTag:
        if i == output:
            #print(f'il numero di posizione nella lista {dict-
Tag} è: {jj}')
            return [jj,i]
        else:
            jj = jj+1

    return [jj,i]
```

```
def leggi_xml(nome_file):
    '''legge il file xml e crea una lista in Python'''
    #limite dati:
    with io.open(nome_file, 'r') as f:
        # leggi tutte le righe
        lines = f.readlines()
    f.close()
    return lines


def definizione_skip(righe,lines):
    riga_limite = 0

    for i in range(righe):
        if "<Data>"  in lines[i]:
            riga_limite = i
            break

    return riga_limite


def ritaglia_xml(lines,righe,n):
    '''ritaglia il file lasciando solo i metadati e crea un file
dei metadati
    in termine di coordinate e tag dei nodi'''
    riga_limite = 0

    for i in range(righe):
        if "<Data>"  in lines[i]:
            riga_limite = i
            break
        if riga_limite == 0:
            print("non ho trovato il limite")


    nome_file =str(f'disp_{n}_metadata.xml')
    select = open(nome_file,'w')
    select.close()

    select = open(nome_file,'a')
    for i in range(riga_limite - 1):
        select.write(lines[i])

    select.write(" </NodeOutput>")
    select.write(" </OPENSEES>")
```

```
    select.close()
    return print(nome_file)

def definizione_nodi(select):
    data_disp = xtd.parse(select)
    tag_disp = data_disp.findall(".//NodeOutput")
    nodeTag_disp = []
    node_coord_disp = {}
    for item in tag_disp:
        a = int(item.attrib['nodeTag'])
        x = float(item.attrib['coord1'])
        y = float(item.attrib['coord2'])
        z = float(item.attrib['coord3'])
        nodeTag_disp.append(a)
        node_coord_disp[a] = {'x':x, 'y':y, 'z':z}
    nodeTag_disp = np.array(nodeTag_disp)
    return nodeTag_disp,node_coord_disp

def NodeCoords(TCLFile):
    #global initNodeCoords
    #modelInfo = OPENSEESTCLRead(TCLFile, 'model', 6)
    #ndm = ndm_v(TCLFile)
    #ndm imposto = 2
    ndm = 2
    if ndm==3:
        nodeInfo = OPENSEESTCLRead(TCLFile, '   node', 5)
        initNodeCoords = nodeInfo[:, 2:5].astype(float)
    elif ndm==2:

        nodeInfo = OPENSEESTCLRead(TCLFile, '   node', 4)
        initNodeCoord = nodeInfo[:, 2:4].astype(float)
        initNodeCoords1=[]
        z = np.repeat(0, len(initNodeCoord[:,0]))
        for i in range (len(initNodeCoord[:,0])):

            initNodeCoords1.append((initNodeCoord[i]).tolist())
        initNodeCoords=np.column_stack([initNodeCoords1, z])
    return (initNodeCoords)

def output_disp(imp_disp):
    ndm = 2
    output = np.array(imp_disp)
    n_nodes = (len(output[0])-1)/2
    n_steps = len(output)
    time_frame = output[:,0]
    response_rshp = output[:, 1:].reshape(-1, ndm)
    z_res = np.repeat(0, len(response_rshp[:, 0]))
```

```
    response = np.column_stack([response_rshp, z_res])
    return [output, n_nodes, n_steps, time_frame, response_rshp,
response]


def spostamenti_x_nodo(nodo,nodeTag_vec,response,numero_nodi):
    index_node=0
    list_tag = list(nodeTag_vec)
    for i in nodeTag_vec:
        if nodo == i:
            index_node = list_tag.index(i)
            print(f'indice: {index_node}')
    #spostamenti = response[:,0][index_node-1::int(numero_nodi)]
    spostamenti = response[:,0][index_node::int(numero_nodi)]

    return spostamenti

def spostamenti_y_nodo(nodo,nodeTag_vec,response,numero_nodi):
    index_node=0
    list_tag = list(nodeTag_vec)
    for i in nodeTag_vec:
        if nodo == i:
            index_node = list_tag.index(i)
            print(f'indice: {index_node}')
    #spostamenti = response[:,1][index_node-1::int(numero_nodi)]
    spostamenti = response[:,1][index_node::int(numero_nodi)]
    return spostamenti

def extract_nodeCoord(dictCord):
    listCoord=[]
    listNodes = []
    for i in dictCord:
        x = dictCord[i]['x']
        y = dictCord[i]['y']
        z = dictCord[i]['z']
        cord = [x,y,z]
        listCoord.append(cord)
        listNodes.append(i)
    return np.array(listCoord),np.array(listNodes)

def convertFromParallel_x(time_frame,dict_global_disp):
    '''riscrive il file output come se fosse stato eseguito in
seriale il risultato è un file .out'''
    otp = open('outptu_disp.out','w')
    otp.close()
    for j in range(len(time_frame)):
        print(f'time_frame: {time_frame[j]}')
```

```
        otp = open('outptu_disp.out','a')
        #print(f'time_frame: {time_frame[j]}')
        otp.write(f'{time_frame[j]} ')
        #otp.close()
        for i in dict_global_disp:
            #if j > 0:
                if len(dict_global_disp[i]['x_disp']) == 1:
                    #extract.append([time_frame[j],i,np.nan])
                    otp = open('outptu_disp.out','a')
                    otp.write(f'0.0 ')
                    #otp.close()
                else:
                    valore = dict_global_disp[i]['x_disp'][j]
                    #extract.append([time_frame[j],i,valori])
                    otp = open('outptu_disp.out','a')
                    otp.write(f'{valore} ')
                    #otp.close()
        otp = open('outptu_disp.out','a')
        otp.write(f'\n')
        otp.close()

def analisi_len(vettore,lunghezza):
    k = 0
    for i in vettore:
        if len(i) > lunghezza:
            print(f'non omogeneo in {k}')
        if len(i) < lunghezza:
            print(f'non omogeneo in {k}')
            print('correggo con zeri....')
            vettore[k] = np.repeat(0.0,lunghezza)
        k = k + 1


riga_limite = 0

lines = leggi_xml(file)
lines2 = leggi_xml(file2)
lines3 = leggi_xml(file3)
lines4 = leggi_xml(file4)


righe = len(lines)
righe2 = len(lines2)
righe3 = len(lines3)
righe4 = len(lines4)

skip0 = definizione_skip(righe,lines)+1
```

```
skip1 = definizione_skip(righe2,lines2)+1
skip2 = definizione_skip(righe3,lines3)+1
skip3 = definizione_skip(righe4,lines4)+1

ritaglia_xml(lines,righe,0)
ritaglia_xml(lines2,righe2,1)
ritaglia_xml(lines3,righe3,2)
ritaglia_xml(lines4,righe4,3)

imp_disp =   pd.read_csv(file, delim_whitespace=True,
header=None, comment='<', skiprows=skip0)

imp_disp2 =   pd.read_csv(file2, delim_whitespace=True,
header=None, comment='<', skiprows=skip1)

imp_disp3 =   pd.read_csv(file3, delim_whitespace=True,
header=None, comment='<', skiprows=skip2)

imp_disp4 =   pd.read_csv(file4, delim_whitespace=True,
header=None, comment='<', skiprows=skip3)

select = "disp_0_metadata.xml"
nodeTag,node_coord = definizione_nodi(select)

select = "disp_1_metadata.xml"
nodeTag2,node_coord2 = definizione_nodi(select)

select = "disp_2_metadata.xml"
nodeTag3,node_coord3 = definizione_nodi(select)

select = "disp_3_metadata.xml"
nodeTag4,node_coord4 = definizione_nodi(select)

[output, n_nodes, n_steps, time_frame, response_rshp, response]
= output_disp(imp_disp)
[output2, n_nodes2, n_steps2, time_frame, response_rshp2, re-
sponse2] = output_disp(imp_disp2)
[output3, n_nodes3, n_steps3, time_frame, response_rshp3, re-
sponse3] = output_disp(imp_disp3)
[output4, n_nodes4, n_steps4, time_frame, response_rshp4, re-
sponse4] = output_disp(imp_disp4)

{'x_disp':spostamenti_x_nodo(i,nodeTag,re-
sponse_rshp,int(n_nodes)),'y_disp':sposta-
menti_y_nodo(i,nodeTag,response_rshp,int(n_nodes))}

dict_global_disp={}
```

```
for i in nodeTag:
    dict_global_disp[i] = {'x_disp':sposta-
menti_x_nodo(i,nodeTag,re-
sponse_rshp,int(n_nodes)),'y_disp':sposta-
menti_y_nodo(i,nodeTag,response_rshp,int(n_nodes))}
for i in nodeTag2:
    dict_global_disp[i] = {'x_disp':sposta-
menti_x_nodo(i,nodeTag2,re-
sponse_rshp2,int(n_nodes2)),'y_disp':sposta-
menti_y_nodo(i,nodeTag2,response_rshp2,int(n_nodes2))}
for i in nodeTag3:
    dict_global_disp[i] = {'x_disp':sposta-
menti_x_nodo(i,nodeTag3,re-
sponse_rshp3,int(n_nodes3)),'y_disp':sposta-
menti_y_nodo(i,nodeTag3,response_rshp3,int(n_nodes3))}
for i in nodeTag4:
    dict_global_disp[i] = {'x_disp':sposta-
menti_x_nodo(i,nodeTag4,re-
sponse_rshp4,int(n_nodes4)),'y_disp':sposta-
menti_y_nodo(i,nodeTag4,response_rshp4,int(n_nodes4))}

dict_coord_global={}
for i in node_coord:
    dict_coord_global[i] = node_coord[i]
for i in node_coord2:
    dict_coord_global[i] = node_coord2[i]
for i in node_coord3:
    dict_coord_global[i] = node_coord3[i]
for i in node_coord4:
    dict_coord_global[i] = node_coord4[i]


connectivity = OPENSEESTCLRead("elements.TCL", "element
SSPquadUP", 7)
elemList = connectivity[:,2]
nodeCoord, nodeList = extract_nodeCoord(dict_coord_global)

del lines
del lines2
del lines3
del lines4

del imp_disp
del imp_disp2
del imp_disp3
del imp_disp4
```

```
del node_coord
del node_coord2
del node_coord3
del node_coord4

del nodeTag
del nodeTag2
del nodeTag3
del nodeTag4

del output
del output2
del output3
del output4

del response
del response2
del response3
del response4

del response_rshp
del response_rshp2
del response_rshp3
del response_rshp4

conv = [i for i in [dict_global_disp[key]['x_disp'] for key in
dict_global_disp]]
analisi_len(conv,n_steps)
conv = np.array(conv).reshape(-1,n_steps).T

first_array = conv[0,:]
conv = conv-first_array

deform_x = np.array(conv).reshape((-1,1))

cells = quad_cell(connectivity,nodeList)
cell_type = (cell_type_quad(connectivity))
print(f'celltype: {cell_type}')
mesh = pv.UnstructuredGrid( cells, cell_type, nodeCoord)
n_points = mesh.n_points


# Aggiungere la mesh iniziale con t = 0
t_init = 0.0
modulo_spostamenti = calcola_modulo_spostamenti(t_init,deform_x)
mesh['modulo_spostamenti'] = modulo_spostamenti
spostamenti_zero = mesh.points.copy()
```

```
#####################################################################
################
############################ CREAZIONE VIDEO
###################################
#Creare un plotter
# p = pv.Plotter()

# p.add_mesh(mesh, scalars='modulo_spostamenti', cmap='jet',
opacity=1.0, name='mesh', show_edges = True)
# p.show_axes
# p.camera_position = 'xy'

# p.open_gif('animation.gif')
# nframe = 5000
# dT_nframe = 0.01
# t_durata = nframe*dT_nframe
# time_frame = np.arange(0.0,t_durata,dT_nframe)

# for t in time_frame:
    # scala = 500.0
    #label_coords = mesh.points + [0, 0, 0.02]
   # point_labels = [f'{i}' for i in range(mesh.n_points)]
   # p.add_point_labels(label_coords, point_labels, name='la-
bel', font_size=10, point_size=10)

    # nuovo_modulo = calcola_modulo_spostamenti(t,deform_x)
    # mesh['modulo_spostamenti'] = nuovo_modulo
    # vect_zero = np.repeat(0,len(nuovo_modulo[:]))
    # vec_sum = np.column_stack([nuovo_mo-
dulo,vect_zero,vect_zero])
    # mesh.points = spostamenti_zero + vec_sum*scala
    #mesh.points =  vec_sum*scala
    # p.add_mesh(mesh, scalars='modulo_spostamenti', cmap='jet',
opacity=1.0, clim = [-0.08,0.05],  name='mesh', show_edges =
True)
    #p.camera.zoom(2)

    # p.camera_position = [(1033.0888501834386,
18.893062403961615, 796.7726439626823),
     # (1036.9287631858638, 71.04047286922118, -
126.29949604983189),
     # (0.010157813813937206, 0.9983541715885984,
0.05644259818757917)]

    # p.write_frame()
# p.close()
```

```
# p.clear()
############################################################
############################################################

sargs= dict(interactive=False)
cells = quad_cell(connectivity,nodeList)
cell_type = (cell_type_quad(connectivity))
mesh = pv.UnstructuredGrid( cells, cell_type, nodeCoord)
n_points = mesh.n_points

p = pv.Plotter()

t_init = 0.0
modulo_spostamenti = calcola_modulo_spostamenti(t_init,deform_x)
mesh['modulo_spostamenti'] = modulo_spostamenti

p.add_mesh(mesh, scalar_bar_args=sargs, scalars='modulo_sposta-
menti', cmap='jet', opacity=1.0, clim=[0.0,0.03], name='mesh',
show_edges = True)
mesh_deformata = mesh.copy()

p.add_mesh(mesh_deformata, scalar_bar_args=sargs, scalars='mod-
ulo_spostamenti', cmap='jet', opacity=1.0, clim=[0.0,0.03],
name='mesh', show_edges = True)

durata = 50
p.add_slider_widget(update,rng=[0,durata],title="time (sec)")
p.show(cpos='xy')
```

RINGRAZIAMENTI

Il dottorato è stato finanziato da ICSC – Centro Nazionale di Ricerca in High Performance Computing, Big Data e Quantum Computing – Spoke 5 “Ambiente e disastri naturali” con fondi PNRR.

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

BIBLIOGRAFIA

Alpan, I. (1970). The geotechnical properties of soils. Earth-Science Reviews, 6, 5–49.

Amorosi A., Boldini D., di Lernia A., 2016. Seismic ground response at Lotung: Hysteretic elasto-plastic-based 3D analyses. Soil Dynamics and Earthquake Engineering, vol. 85, pp. 44-61.

Benz, T. (2006). Small-Strain Stiffness of Soils and its Numerical Consequences.Ph.d. thesis, Universität Stuttgart.

Benz, T., Vermeer, P.A., Schwab, R., 2009. A small-strain overlay model. International Journal for Numerical and Analytical Methods in Geomechanics, vol. 33, pp. 25-44. https://doi.org/10.1002/nag.701

Brinkgreve, R.B.J., Kumarswamy, S., Swolfs, W.M., Waterman, D., Chesaru, A., Bonnier, P.G., 2016. PLAXIS 2016. Delft: PLAXIS bv.

Cavallo, G., Elia, G., di Lernia, A., 2024. Implementazione di modelli FEM OPENSEES nell'infrastruttura HPC RECAS nell'ambito del progetto CN-Calcolo HPC. In: Incontro Annuale dei Ricercatori di Geotecnica 2024 (IARG 2024), Gaeta, 4-6 settembre 2024.

Cavallo, G., 2025. opstvis_v1.0: a node results visualizing tool (with pyvista). Zenodo. https://doi.org/10.5281/zenodo.15025232

Cavallo, G., 2024. OPENSTOOLS v2: example fem models. Zenodo. https://doi.org/10.5281/zenodo.14174619

Cavallo, G., 2024. OPENSTOOLS v2 - A python code to pre-process the finite element models in the OPENSEES, OPENSEESMP and OPENSEESSP software. Zenodo.

Chiaradonna, A., 2022. Defining the Boundary Conditions for Seismic Response Analysis - A Practical Review of Some Widely-Used Codes. Geosciences, vol. 12, no. 2, 83. https://doi.org/10.3390/geosciences12020083

Cudny, M., Truty, A., 2020. Refinement of the Hardening Soil model within the small strain range. Acta Geotechnica, vol. 15, pp. 2031-2051. https://doi.org/10.1007/s11440-020-00945-5

Cui, W., Jing, H., Potts, D.M., Dong, L., Pedone, G., Zdravkovic, L., Yao, Y., 2024. Time-step constraints in coupled hydro-mechanical finite element analysis of unsaturated soils. Computers and Geotechnics, vol. 165, 105914. https://doi.org/10.1016/j.compgeo.2023.105914

di Lernia, A., Buono, C., Elia, G., 2023. Evaluation of seismic site effects in a real slope through 2D FE numerical analyses. In: 9th ECCOMAS Thematic Conference on Computational Methods in Structural Dynamics and Earthquake Engineering (COMPDYN 2023).

Elia, G., di Lernia, A., Cavallo, G., 2025. Performance of nonlinear 2D numerical models for the seismic response of a natural slope. In: ANIDIS XX, Assisi, 7-11 settembre 2025.

Elia, G., Rouainia, M., 2022. Advanced dynamic nonlinear schemes for geotechnical earthquake engineering applications: a review of critical aspects. Geotechnical and Geological Engineering, vol. 40, pp. 3851-3887. https://doi.org/10.1007/s10706-022-02109-6

Elia, G., Rouainia, M., di Lernia, A., D'Oria, A.F., 2021. Assessment of damping predicted by kinematic hardening soil models during strong motions. Géotechnique Letters, vol. 11, no. 1, pp. 48-55. https://doi.org/10.1680/jgele.20.00078

Gu, Q., Conte, J.P., Yang, Z., Elgamal, A., 2011. Consistent tangent moduli for multi-yield-surface J2 plasticity model. Computational Mechanics, vol. 48, pp. 97-120. https://doi.org/10.1007/s00466-011-0576-7

Iwan, W.D., 1967. On a class of models for the yielding behavior of continuous and composite systems. Journal of Applied Mechanics, vol. 34, no. 3, pp. 612-617.

Laera, A., Brinkgreve, RBJ. 2015. Site response analysis and liquefaction evaluation. Plaxis BV.

Latijrelle, F.G., 1989. Finite element analysis of wave propagation in an elastic half-space under step loading. Computers & Structures, vol. 31, no. 4, pp. 553-559.

Lysmer, J., Kuhlemeyer, R.L., 1969. Finite Dynamic Model for Infinite Media. Journal of the Engineering Mechanics Division 95, 859–877. https://doi.org/10.1061/JMCEA3.0001144

Løkke, A., Chopra, A.K., 2019. Direct-finite-element method for nonlinear earthquake analysis of concrete dams including dam-water-foundation rock interaction. Earthquake Engineering and Structural Dynamics, vol. 48, no. 14, pp. 1609-1630.

Løkke, A., Chopra, A.K., 2018. Direct finite element method for nonlinear earthquake analysis of 3-dimensional semi-unbounded dam-water-foundation rock systems. Earthquake Engineering and Structural Dynamics, vol. 47, no. 5, pp. 1309-1328. https://doi.org/10.1002/eqe.3019

Løkke, A., Chopra, A.K., 2017. Direct finite element method for nonlinear analysis of semi-unbounded dam-water-foundation rock systems. Earthquake Engineering and Structural Dynamics, vol. 46, no. 8, pp. 1267-1285. https://doi.org/10.1002/eqe.2855

Løkke, A., Chopra, A.K., 2013. Response spectrum analysis of concrete gravity dams including dam-water-foundation interaction. Earthquake Engineering and Structural Dynamics, vol. 42, no. 10, pp. 1475-1494.

McGann, C.R., Arduino, P., Mackenzie-Helnwein, P., 2012. Stabilized single-point 4-node quadrilateral element for dynamic analysis of fluid saturated porous media. Acta Geotechnica, vol. 7, no. 4, pp. 297-311. https://doi.org/10.1007/s11440-012-0168-5

McKenna, F., Fenves, G.L., Scott, M.H., 2000. Open system for earthquake engineering simulation. Berkeley: University of California.

Mróz, Z., Norris, V.A., Zienkiewicz, O.C., 1981. An anisotropic, critical state model for soils subject to cyclic loading. Géotechnique, vol. 31, no. 4, pp. 451-469.

Mróz, Z., Norris, V.A., 1979. Application of an anisotropic hardening model in the analysis of elasto-plastic deformation of soils. Géotechnique, vol. 29, no. 1, pp. 1-34.

Petracca, M., Candeloro, F., Camata, G., 2017. STKO user manual. Pescara: ASDEA Software Technology.

Prevost, J.H., 1977. Mathematical modelling of monotonic and cyclic undrained clay behaviour. International Journal for Numerical and Analytical Methods in Geomechanics, vol. 1, no. 2, pp. 195-216. https://doi.org/10.1002/nag.1610010206

Qiu, Z., Lu, J., Elgamal, A., Su, L., Wang, N., Almutairi, A., 2019. OPENSEES three-dimensional computational modeling of ground structure systems and liquefaction scenarios. CMES - Computer Modeling in Engineering and Sciences, vol. 120, no. 3, pp. 629-656. https://doi.org/10.32604/cmes.2019.05759

Ricker N. H., 1940. The form and nature of seismic waves and the structure of seismograms. In: Geophysics vol. 5, n. 4, pp. 348-366

Schanz, T., Vermeer, P.A., Bonnier, P.G., 1999. The hardening soil model: Formulation and verification. In: Beyond 2000 in Computational Geotechnics, pp. 281-296. Rotterdam: Balkema.

Schmicker, D., Duczek, S., Liefold, S., Gabbert, U., 2014. Wave propagation analysis using high-order finite element methods: Spurious oscillations excited by internal element eigenfrequencies. PAMM - Proceedings in Applied Mathematics and Mechanics, vol. 14, no. 1, pp. 51-71. https://doi.org/10.24352/UB.OVGU-2017-053

Seron, F.I., Sanz, F.J., Kindelan, M., Badal, J.I., 1990. Finite-element method for elastic wave propagation. Communications in Applied Numerical Methods, vol. 6, no. 5, pp. 359-368.

Sonnessa, A., di Lernia, A., Nitti, D.O., Nutricato, R., Tarantino, E., Cotecchia, F., 2023. Integration of multi-sensor MTInSAR and ground-based geomatic data for the analysis of non-linear displacements affecting the urban area of Chieuti, Italy. International Journal of Applied Earth Observation and Geoinformation, vol. 117, 103194. https://doi.org/10.1016/j.jag.2023.103194

Sullivan, C., Kaszynski, A., 2019. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). Journal of Open Source Software, vol. 4, no. 37, 1450. https://doi.org/10.21105/joss.01450

Tagarelli, V., Santaloia, F., Elia, G., Cotecchia, F., 2025. Numerical modelling of geological processes as means for the diagnosis of ancient landslide mechanisms. Computers and Geotechnics, vol. 184, 107238. https://doi.org/10.1016/j.compgeo.2025.107238

van Rossum, G., Drake Jr., F.L., 1995. Python reference manual. Amsterdam: Centrum voor Wiskunde en Informatica.

Yang, Z., Lu, J., Elgamal, A., 2008. OPENSEES soil models and solid-fluid fully coupled elements user's manual. San Diego: University of California.

Zhang, Y., Pedroso, D.M., Li, L., Scheuermann, A., Ehlers, W., 2020. Accurate and stabilised time integration strategy for saturated porous media dynamics. Acta Geotechnica, vol. 15, pp. 1859-1879. https://doi.org/10.1007/s11440-019-00879-7

CURRICULUM

GIANLUCA CAVALLO

g.phd.pba@protonmail.com
+39 3516766346
Grottaglie (TA), Italia
LinkedIn: linkedin.com/in/gianluca-cavallo-341578142
Ord. Ing. Milano N. 32496

PROFILO PROFESSIONALE

Ingegnere Civile Geotecnico con Dottorato di Ricerca in corso e oltre 7 anni di esperienza nella progettazione di opere geotecniche complesse, gallerie, infrastrutture critiche e analisi del rischio sismico. Expertise consolidata in modellazione numerica avanzata (FEM, metodi agli elementi finiti), BIM e gestione di progetti infrastrutturali strategici. Orientato all'innovazione, alla sostenibilità e all'eccellenza tecnica.

COMPETENZE CHIAVE

Competenze Tecniche Specialistiche

Progettazione Geotecnica Avanzata: opere di sostegno, fondazioni profonde, consolidamenti, gallerie (tradizionale e meccanizzato), dighe

Modellazione Numerica FEM: analisi bidimensionali e tridimensionali, simulazioni idromeccaniche accoppiate, analisi dinamiche e sismiche

Analisi del Rischio: valutazione vulnerabilità sismica, prevenzione catastrofi naturali, mitigazione del rischio geotecnico

Software Specialistici: OPENSEES, PLAXIS 2D/3D (avanzato), FLAC 2D, SAP2000, DIANA FEA, PHASE2, SLIDE, DIPS, PARATIE PLUS, GEO5, Revit BIM

Programmazione e Automazione: Python, Visual Basic (VBA), sviluppo tool personalizzati per analisi parametriche

BIM e Digitalizzazione: Revit, AutoCAD 2D/3D, integrazione BIM nei processi di progettazione geotecnica

Competenze Trasversali

Project Management: pianificazione, coordinamento team multidisciplinari, gestione budget e scadenze
Contract Management: gestione contratti pubblici e privati, claim management, risoluzione controversie
Coordinamento Sicurezza: CSP/CSE ai sensi D.Lgs 81/08, gestione sicurezza cantieri complessi

Lingue: Italiano (madrelingua), Inglese (eccellente - C2)

FORMAZIONE ACCADEMICA E TITOLI

Dottorato di Ricerca (Ph.D.) in Prevenzione e Protezione dal Rischio di Catastrofi Naturali

Politecnico di Bari – DICATECH geotecnica | 2023-2026 (in corso)

Focus: Modellazione numerica avanzata per l'analisi del rischio geotecnico e sismico. Sviluppo di metodologie innovative per la previsione e mitigazione degli effetti delle catastrofi naturali su infrastrutture critiche con l'ausilio dell'alta capacità computazionale (sistemi HPC e cloud computing).

Pubblicazioni Scientifiche:

Cavallo G. et al. (2024) "Implementazione di modelli fem OPENSEES nell'infrastruttura hpc RECAS nell'ambito del progetto cn-calcolo hpc" - IARG, Gaeta 2024

Elia G. et al. (2025) "Performance of nonlinear 2D numerical models for the seismic response of a natural slope" - XX Convegno ANIDIS, Assisi 2024

Master di II Livello in Project and Contract Management in Construction Works

Politecnico di Milano | Ott 2020 - Feb 2021 | 110/110

Tematiche: Gestione contratti FIDIC, claim management, project financing, dispute resolution, gestione commesse internazionali

Laurea Magistrale in Ingegneria Civile (LM-23) - Indirizzo Strutture e Geotecnica

Politecnico di Bari

Tesi: "Modellazione numerica e analisi idromeccanica accoppiata del pendio Pianello nel comune di Bovino (FG) con PLAXIS 3D"

Riconoscimento: Menzione d'Onore - V Premio Nazionale di Laurea "Ing. Salvatore Fazio" (DICAR Università di Catania + CMC Ravenna) per gli aspetti innovativi e il rigore scientifico della tesi

Abilitazione alla Professione di Ingegnere Civile *Politecnico di Bari | Settembre 2016*

CERTIFICAZIONI E ALBI PROFESSIONALI

Ordine degli Ingegneri di Milano - Iscrizione N. 32496 (dal Feb 2019)
Albo Professionisti Antincendio VVF - N. MI32496I03533 (dal Dic 2021)
Coordinatore Sicurezza CSP/CSE - D.Lgs 81/08 (120h - Formedil CPT Taranto, 2017)
Certificazione Energetica Edifici - Linee Guida Nazionali (80 CFU - Unipro, 2021)

ESPERIENZA PROFESSIONALE

Ingegnere Geotecnico Senior
F&M Ingegneria SPA - Mirano (VE) | Apr 2022 - Ago 2022

Società di ingegneria specializzata in grandi opere infrastrutturali

Progettazione e calcolo di strutture geotecniche complesse con metodi standard e avanzati (FEM 3D)

Analisi di stabilità di pendii e verifiche di opere di sostegno in contesti ad elevata complessità geologica

Supporto tecnico specialistico per commesse pubbliche e private con approccio BIMoriented

Ingegnere Geotecnico - Specialista Gallerie
Pini Group srl - Roma | Ott 2021 - Mar 2022

Studio internazionale leader nella progettazione di gallerie e opere del sotterraneo

Progettazione esecutiva di gallerie in tradizionale: dimensionamento rivestimenti provvisori e definitivi, analisi di interazione terreno-struttura

Consolidamento e adeguamento normativo di gallerie esistenti con metodologie innovative

Analisi numeriche FEM per la valutazione degli effetti di scavo su pre-esistenze e reti di sottoservizi

Ingegnere Geotecnico
Studio Speri - Roma | Mag 2021 - Set 2021

Studio specializzato in analisi strutturali e geotecniche avanzate

Analisi di vulnerabilità sismica di dighe a gravità in calcestruzzo mediante software DIANA FEA

Modellazione non-lineare del comportamento dinamico di grandi opere idrauliche, valutazione indicatori di danno

Ingegnere Civile - Junior Project Engineer
RockSoil SPA - Milano | Apr 2018 - Apr 2020

Società leader in consulenza geotecnica per grandi infrastrutture nazionali

Progetti Strategici:

Alta Velocità Napoli-Bari (Tratte Napoli-Cancello e Apice-Hirpinia): progettazione esecutiva gallerie ferroviarie e opere di imbocco

Gronda di Genova (Potenziamento Autostradale): dimensionamento opere geotecniche in contesto urbano complesso

SS 106 Jonica e SS 219 Gubbio-Pian d'Assino (ANAS): progettazione gallerie stradali e opere di sostegno

Responsabilità e Competenze Acquisite:

Caratterizzazione geotecnica avanzata: interpretazione prove in situ (SPT, CPT, DMT, PMT) e di laboratorio

Dimensionamento e verifica opere di sostegno (paratie, muri, berlinesi), analisi pushover per verifiche sismiche

Progettazione gallerie: analisi convergenza-confinamento, metodo delle reazioni iperstatiche, modellazione FEM 3D

Analisi di interferenza tra scavi e pre-esistenze: stima cedimenti indotti, verifiche stabilità globale

Redazione elaborati tecnici specialistici, relazioni geotecniche secondo NTC2018

Research Assistant - Test Engineer

Plaxis BV - Delft, Paesi Bassi | Dic 2016 - Gen 2017

Software house leader mondiale in software geotecnici FEM

Testing e validazione di algoritmi avanzati: Material Point Method (MPM) per grandi deformazioni

Sviluppo e implementazione del modello costitutivo Shansep NGI-ADP per argille sovraconsolidate

Tirocinante di Ricerca

TU Delft (Delft University of Technology) - Paesi Bassi | Mag 2016 - Ago 2016

Università tecnica di eccellenza internazionale (Top 10 Engineering Universities - QS Ranking)

Sviluppo modello FEM 3D di argini in torba con simulazione idromeccanica accoppiata time-dependent per previsione cedimenti a lungo termine (tesi di laurea magistrale)

FORMAZIONE PROFESSIONALE CONTINUA (selezione)

Progettazione Antisismica con Elementi in Acciaio - Fondazione Promozione Acciaio (16 CFP, 2020)

Progettazione Ponti in Carpenteria Metallica - Fondazione Promozione Acciaio (10 CFP, 2020)

Corso Avanzato Code-Aster e Salome-Meca - Ordine Ingegneri Milano (8 CFP, 2020) - Software FEM open source

Interazione Dinamica Terreno-Struttura - Ordine Ingegneri Milano (3 CFP, 2021)

Interazione Terreno-Struttura con Software Specialistici - Ordine Ingegneri Milano (3 CFP, 2021)

Joint Summer School in Geotechnics - DICATECH Politecnico di Bari (2015)
Sperimentazione Geotecnica Avanzata (Prof. C. Viggiani) - Politecnico di Bari (2014)

COMPETENZE INFORMATICHE

Software Geotecnici e Strutturali (Livello Avanzato):

PLAXIS 2D/3D, FLAC 2D, SAP2000, DIANA FEA, PHASE2, SLIDE, DIPS, PARATIE PLUS, GEO5, VCA SLU

BIM e CAD:

Revit (BIM Coordinator level), AutoCAD 2D/3D

Programmazione e Automazione:

Python (livello intermedio), Visual Basic for Applications (VBA - avanzato), sviluppo macro Excel per calcoli parametrici

Software Open Source:

Code-Aster, Salome-Meca (FEM), QGIS (GIS), OPENSEES

Suite Microsoft Office:

Word, Excel (livello avanzato con VBA), PowerPoint, Project

Sistemi Operativi:

Windows (tutti), Linux (Ubuntu, Debian), gestione ambienti virtualizzati

INTERESSI PROFESSIONALI E ATTIVITÀ

Innovazione in Geotecnica: approcci digitali, machine learning applicato alla geotecnica, monitoraggio IoT

Sostenibilità: tecniche di consolidamento a basso impatto ambientale, geotecnica ambientale

Formazione continua: partecipazione regolare a convegni nazionali (AGI, ANIDIS) e internazionali

Attività extra-professionali: musicista (chitarra), volontariato