TALLINNA TEHNIKAÜLIKOOL

EESTI MEREAKADEEMIA

Meremajanduse keskus

Tiit Jalasto

# Termomõjutsooni lisamine LEM programm ABAQUS mudelisse

Lõputöö

Juhendaja: DSc, Mihkel Kõrgesaar

Kuressaare 2021

Olen koostanud töö iseseisvalt.

Töö koostamisel kasutatud kõikidele teiste autorite töödele,

olulistele seisukohtadele ja andmetele on viidatud.

Tiit Jalasto

……...

(allkiri, kuupäev)

Üliõpilase kood: 178753SDSR

Üliõpilase e-posti aadress: tiit.jalasto@gmail.com

Juhendaja: DSc, Mihkel Kõrgesaar

Töö vastab lõputööle esitatud nõuetele

……..…

(allkiri, kuupäev)

Kaitsmiskomisjoni esimees: DSc, Mihkel Kõrgesaar

Lubatud kaitsmisele

……..…

(ametikoht, nimi, allkiri, kuupäev)

# SISUKORD

Sisukord ..... 3

Annotatsioon ..... 5

Sissejuhatus ..... 6

1 Laevade mittelineaarsed purunemisanalüüsid ..... 8

1.1 Taust ..... 8

1.2 LEM tarkvara ..... 9

2 Abaqus-ja Python programm ..... 11

2.1 LEM analüüsiprogramm ABAQUS ..... 11

2.2 Python programm ..... 14

3 Koodi kirjutamine ..... 15

3.1 Näidismudel ..... 15

3.2 Termomõjutsooni kood (HAZ_kood) ..... 17

3.2.1 Sisendfailist elementide ning sõlmede otsimine ..... 18

3.2.2 Naaberelementide otsimine ..... 18

3.2.3 Naaberelementide omavahelise nurga leidmine ..... 19

3.2.4 Termomõjutsooni elementide komplektid ..... 21

3.2.5 Termomõjutsooni elementide komplektide sektsioonid ..... 22

3.2.6 Vahefaili kirjutamine ..... 23

3.2.7 Termomõjutsooni materjalide omadused ..... 24

3.2.8 Lõpliku termomõjutsooni sisendfaili kirjutamine ....25
3.2.9 Valmis kood....26
4 Koodi testimine ....27
4.1 Test 1....27
4.2 Test 2....29
4.3 Test 3....30
4.4 Tulemus ....32
4.5 Edasiarendused ....32
4.5.1 Sisendfaili andmed ....32
4.5.2 Termomõjutsooni elementide leidmine....33
Kokkuvõte ....35
Võõrkeelne lühikokkuvõte ....36
Viidatud allikad ....38
Lisad ....39
Lisa 1 / HAZv1.0/....39
Lisa 2 / HAZv2.0/....43
Lisa 3 / Lihtlitsents /....48

# ANNOTATSIOON

Merekonstruktsioonide kokkupõrke analüüse teostatakse lõplike elementide meetodi (LEM) abil. Simulatsioonides kasutatud parameetrid ei vasta aga alati tegelikkusele. Üks osa simulatsioonide arendamisest on termomõjutsoonide lisamine analüüsi. LEM programmis on termomõjutsoonide lisamine liiga ajakulukas. Selles töös koostatakse „Python“ programmeerimiskeele abil kood, mis eeltöötleb LEM-programmi „Abaqus“ sisendfaili ja võimaldab tõhusalt lisada termomõjutsoone. Selleks tutvutakse LEM- mudelitega ja sisendfailidega. Kirjutatakse kood, mis leiab õiged elemendid, muudab nende materjaliomadusi ja lisab termomõjutsooni elemendid eeltöödeldud sisendfailis ettenähtud kohtadesse. Seejärel kontrollitakse koodi keerukamate mudelitega. Tulemuseks saadi kood, mis töötab ühe osa eksemplari piires ja nelja sõlmega elementide puhul.

Võtmesõnad: Python, Abaqus, lõplike elementide meetod (LEM), kokkupõrkekindlus, termomõjutsoon

# SISSEJUHATUS

Merekonstruktsioonide kokkupõrke analüüs toimub üha enam lõplike elementide meetodi (LEM) abil. Arvutusvõime kiire kasv võimaldab analüüsida suuri ja keerukaid probleeme ja mida täpsemad on tulemused, seda ökonoomsemalt saab merekonstruktsioone ehitada. Suured merekonstruktsioonid valmistatakse enamasti metallist ja metalli ühendamise kõige levinum viis on keevitamine.

Kõrgesaar jt. võrdlesid simulatsioone katsetega ja näitasid, et kokkupõrke simulatsioonidest tehtavad järeldused võivad suuresti erineda olenevalt sellest, kas termomõjutsoon (i.k. heat affected zone - HAZ) on modelleeritud või mitte (Kõrgesaar jt. 2019). Terminoloogiliselt on termomõjutsoon keevisliite põhimetalli sulamata osa, kus on toimunud materjali struktuuri muutused, mis on põhjustatud keevitusprotsessi kiiretest temperatuuri kõikumistest (Kulu jt. 2015a). Antud töös käsitletakse termomõjutsooni mõistet aga laialdasemalt, hõlmates endas nii termomõjutsooni kui ka keevismetalli.

Tuginedes Berntsson jt. uuringule, oli termomõjutsooniga mitte arvestanud simulatsioonides 20% kõrgem kokkupõrkeenergia neeldumine kui termomõjutsooni efekti arvesse võtnud simulatsioonides. See näitab, et võrreldes keevitamata materjaliga vajab keevisliide purunemiseks 20% vähem kokkupõrkeenergiat (Berntsson jt. 2019).

Termomõjutsooni mõju uurimiseks erinevates simulatsioonides on vajalik mudel, kus termomõjutsooni on arvesse võetud. LEM programmis luuakse mudelitele võrk, mis ei sisalda termomõjutsooni, sest see oleks liiga ajakulukas. Kokkupõrke analüüside optimeerimiseks tuleks luua programm mudelite sisendfailide eeltöötlemiseks, mis võimaldaks tõhusalt lisada termomõjutsoone.

Antud lõputöö eesmärgiks on kirjutada LEM programmi Abaqus sisendfaili põhjal Python tarkvara abil kood, mis automaatselt otsib etteantud mudeli konstruktsioonist liitekohti ja lisab termomõjutsoonile uued materjali omadused. Liitekohtade materjaliomadusi muudetakse vastavalt kasutaja sisendile. Selleks tuleb tutvuda Abaqus programmi mudeli loomise printsiipidega ja sisendfailiga ning lihtsa mudeli abil kirjutada kood, mida katsetada keerukamate mudelite peal.

Töö on jaotatud nelja peatükki:

Esimeses osas tutvutakse LEM-tarkvaraga.

Teises osas tutvutakse „Abaqus“ programmi mudeli ja sisendfailiga, ning „Python“ programmiga.

Kolmandas osas kirjutatakse Python programmi abil kood, mis otsib LEM-programmi Abaqus mudeli sisendfailist elemente, mis asuvad termomõjutsoonis ja muudab nende materjaliomadusi.

Neljandas osas katsetatakse erinevate mudelite abil koodi ning vaadatakse, mis piirides kood töötab. Kood peab leidma omavahel määratud nurkade all olevaid elemente. Sellega leitakse kõik keevitatud ühendused ning antakse termomõjutsoonidele uued materjaliomadused.

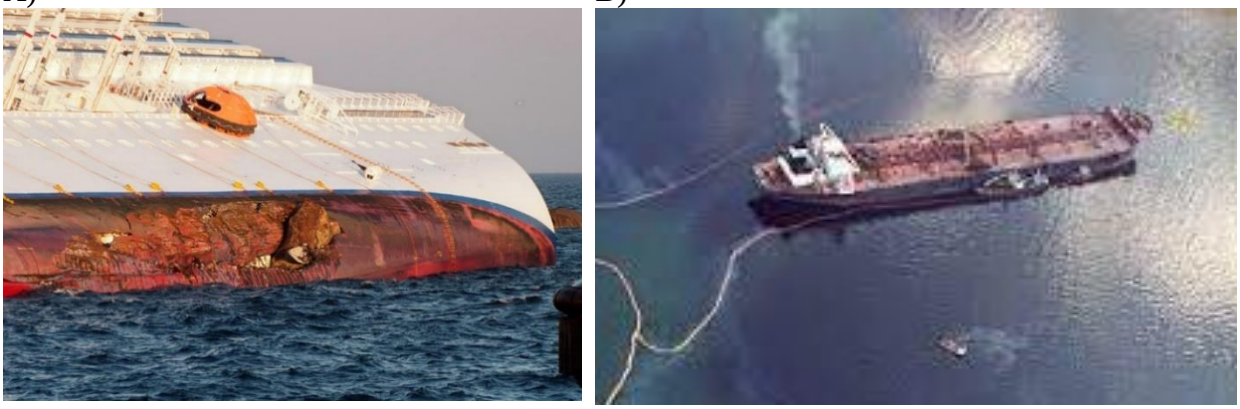
Töös tuginetakse eelnevatele kokkupõrkeuuringutele, Abaqus analüüsi kasutusjuhendile ja Python programmi juhistele.

# 1 LAEVADE MITTELINEAARSED PURUNEMISANALÜÜSID

## 1.1 Taust

Laevade kokkupõrked ja karilesõidud on küll haruldased, kuid võivad lõppeda katastroofiga. Nii juhtus Titanicuga, mis võttis 1912.a. endaga kaasa 1496 inimest. Tänapäevalgi on laevaõnnetuste tagajärjel inimesi hukkunud. Näiteks 2012.a. sõitis Costa Concordia (Joonis 1A) Giglio saare lähedal karile ja õnnetuse tagajärjel hukkus 32 inimest. Exxon Valdez (Joonis 1B) põhjustas 1989a. Alaska lahes suure keskkonnareostuse, kui kokkupõrkel kariga purunesid naftamahutid ja merre voolas 37000 tonni naftat. Seda peetakse keskkonnakahjustuste seisukohast halvimaks naftareostuseks kogu maailmas ja reostuse tagajärgi likvideeritakse tänase päevani. Lisaks inimelude kaotustele ja ökokatastroofidele põhjustavad sellised õnnetused ka suurt majanduslikku kahju.

A) B)

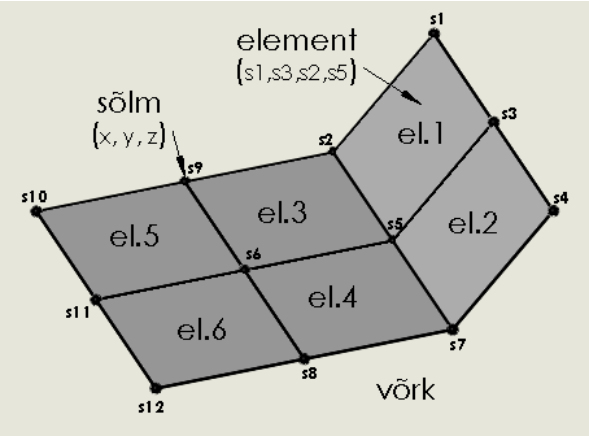


Joonis 1. A) Costa Concordia. B) Exxon Valdez
Allikad: A)https://et.wikipedia.org B) http://www.findingdulcinea.com

On oluline, et kokkupõrkeolukordades oleks laeva konstruktsioonid võimelised vastu pidama löögikoormustele. Seepärast on vaja ka neid õnnetusi simuleerida ja reaalsema tulemuse saavutamiseks peab simulatsioonid viima kooskõlla tegeliku olukorraga. Üks osa simulatsioonide arendamisest on termomõjutsoonide arvestamine analüüsis.

Simulatsioonideks kasutatakse lõplike elementide meetodi (LEM) tarkvara. LEM tarkvaras jagatakse analüüsitav konstruktsioon väiksemateks osadeks ehk lõplikeks elementideks. See tähendab, et koostatakse elementide võrk, milles elemendid on omavahel seotud sõlmedega ja sõlmed on määratletud koordinaatidega (x, y, z) (Joonis 2). Lõplike elementide meetodiga saab

arvestada erinevate materjaliomadustega ja lokaalsete efektidega ning kirjeldada täpselt keerukaid geomeetriaid.



Joonis 2. Võrk

## 1.2 LEM tarkvara

Lõplike elementide meetod ehk LEM (i.k. finite element method-FEM) on numbriline arvutusmeetod insener-tehniliste ja füüsikaliste probleemide lahendamiseks. Meetod hõlmab üldiste (osatuletistega) diferentsiaalvõrrandite ja/või nende süsteemi aproksimeerimist ehk lähendamist üle mingi (pideva) piirkonna (nt tasand), mis jaotatakse väiksemateks, lõplikeks elementideks (nn piirkonna diskreetimine). Numbriline lahend kogu piirkonna jaoks saadakse lõplike elementide summeerimisega mingi eeskirja järgi.(“Lõplike elementide meetod – Vikipeedia,” 2021)

LEM programmi kasutatakse paljudes valdkondades ning see võimaldab teha lineaarseid ja mittelineaarseid analüüse.

Merekonstruktsioonide käitumist kokkupõrkel uuritakse praegu lõplike elementide meetodiga, millega saab simuleerida elastset ja plastset käitumist ning samuti materjalide purunemist. Tehakse pingutusi, et välja töötada täpseid purunemise hindamise meetodeid suurte laevade konstruktsioonidele ja hetkel põhinevad kõige täpsemad meetodid LEM simulatsioonidel. LEM programme eelistatakse analüütilisele meetodile, mis ei suuda arvestada mittelineaarset geomeetriat ja materjali suurtel deformatsioonidel, mis esinevad konstruktsioonis enne

purunemist. Simulatsioonide keerukuse tõttu ei ole aga garanteeritud, et kasutatud simulatsioonimeetodid ja parameetrid vastavad tegelikkusele (Palokangas jt. 2016).

# 2 ABAQUS-JA PYTHON PROGRAMM

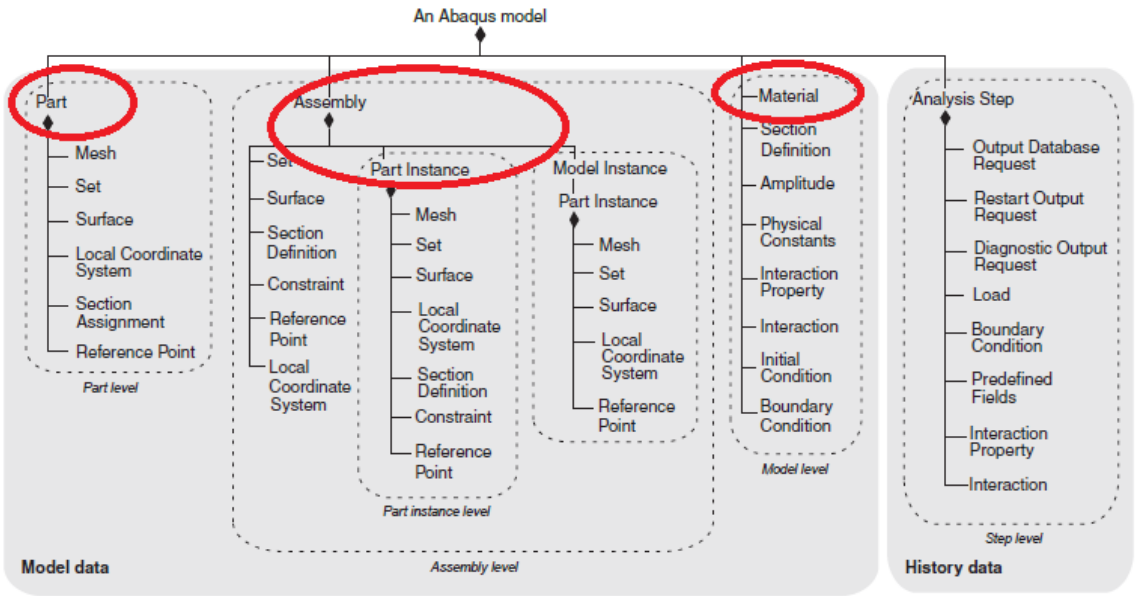
Antud töös kasutatakse Abaqus lõplike elementide meetodi programmi ja Python programmeerimiskeelt.

Abaqus on lõplike elementide analüüsi jaoks mõeldud arvutipõhine inseneritarkvara, mis loodi 1978. aastal dr David Hibbitti ja dr Bengt Karlssoni poolt. Abaqus on tuletatud ladinakeelsest nimest „abacus“ mis tähendab kivikeste või pulgakestega arvutuslauda.

Python on üldotstarbeline, objektorienteeritud, väike, võimas, lihtne ja lõbus ( nagu väidavad keele loojad ja arendajad) vabavaraline programmeerimiskeel, mis loodi 1991. aastal Guido van Rossumi poolt. Nimi on võetud inglise koomikute grupi Monty Python järgi.

## 2.1 LEM analüüsiprogramm ABAQUS

Abaqus programm suudab lugeda teistest tarkvaradest imporditud CAD (Computer aided design-arvutipõhine disain) vorminguid ning tal on ka oma modelleerimisliides. LEM mudeleid saab importida tekstifailidesse sisendfailina, mis on ASCII-fail. Sisendfail koosneb ridadest, mis sisaldavad Abaquse märksõnasid ja andmeridasid ja enamikel sisendfailidel on sama põhistruktuur (Joonis 3)(Simulia, 2014).



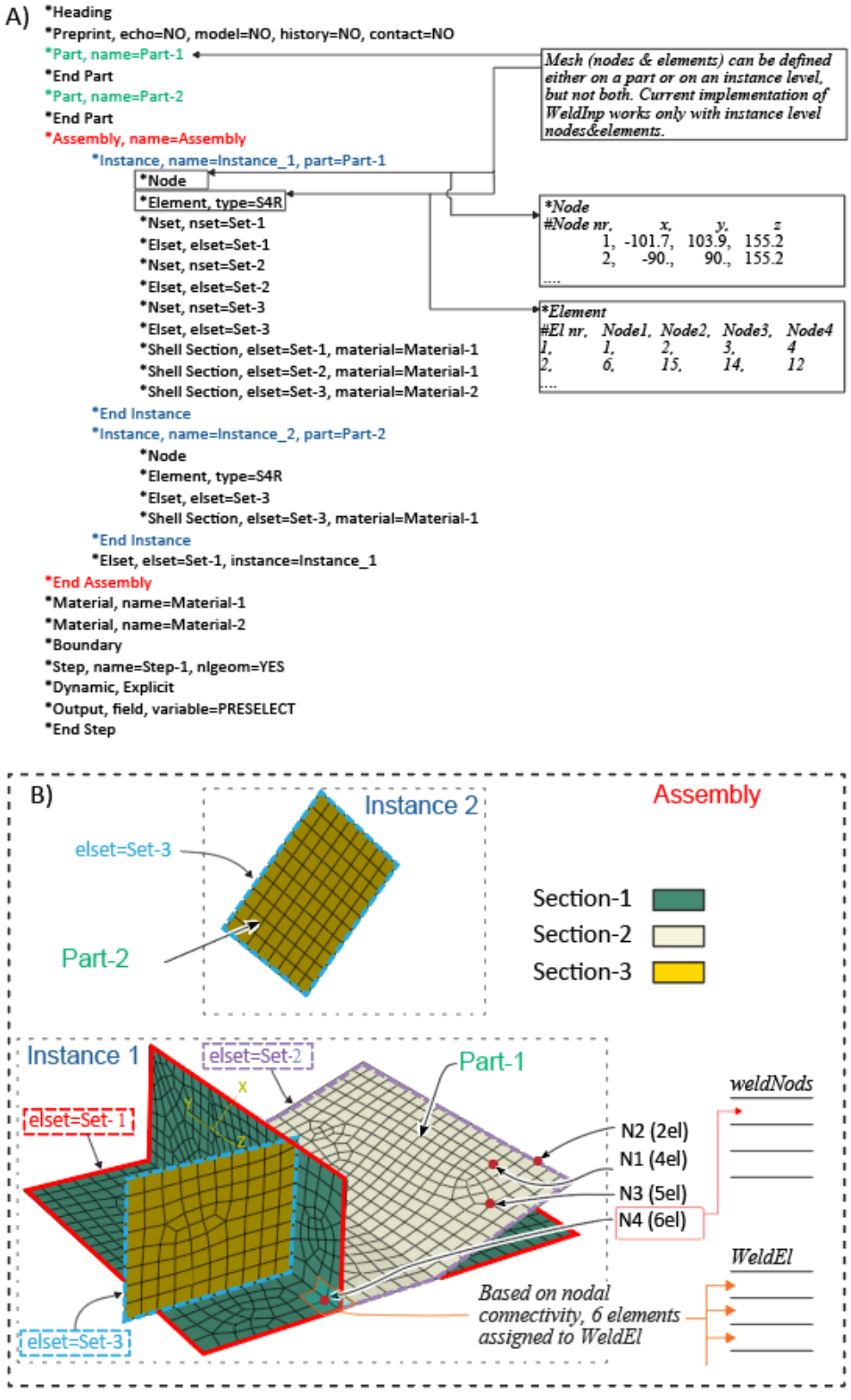

Joonis 3. Osa eksemplaride koostuna määratletud mudeli korraldus
Allikas: Simulia, 2014

Kõik insenertehnilised mudelid koosnevad erinevatest osadest (i.k. part). Osade abil saab luua keerukama koostu (i.k. assembly). Näiteks võib kirjutuslaud koosneda puidust plaadist ja metalljalgadest, mis on omavahel ühendatud poltidega. Kui koostus on vaja midagi muuta (nt. jalg), ei pea muutma iga üksikut osa eraldi, vaid saab muuta osa definitsiooni. LEM mudeli koostus ei analüüsita osasid otse, vaid nad on oma eksemplaride (i.k. part instance) plaan ja osa määratlus ilmub väljaspool koostut (Joonis 3). Seega hõlbustab osaeksemplaride komplekteerimine koostu modelleerimist.

Matrerjali omadused on määratletud mudeli tasandil, et neid saaks korduvalt kasutada ning kõigil mudeli materjalidel on ainulaadsed nimed (Joonis 3).

Võrk (i.k. mesh) määratletakse kas osas või selle osaeksemplaris, mitte mõlemas korraga. Kui on vaja määratleda ühe osa eksemplaris täpsemat võrku kui teises, peab võrgu määratlema eraldi eksemplarides. Seega eelistatakse määratleda võrk osa eksemplarides (Joonis 4A).

Osa eksemplarid (i.k. instance) koosnevad komplektidest (i.k.-set), et määrata erinevate omadustega elementide komplektid ja sektsioonidest (i.k-section), mis määratlevad vastavas elementide komplektis oleva materjali omadused (Joonis 4).



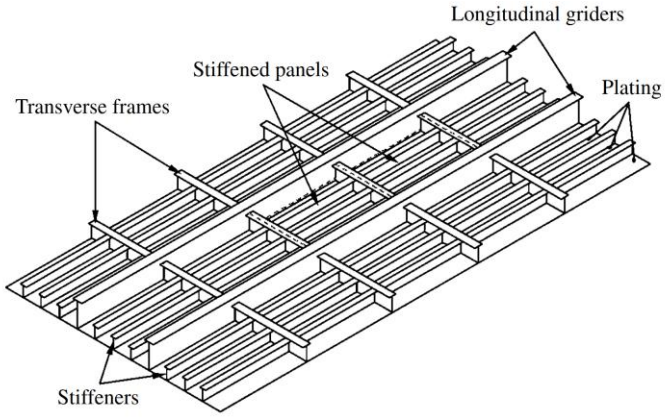

Joonis 4. A) LEM mudeli sisendfaili märksõnad B) sisendfaili LEM mudeli koost

## 2.2 Python programm

Pythoni programm valiti sellepärast, et see on vabavarana saadaval ja selle tõttu saab programmi jagada teiste kasutajatega. Pythoni rakendustes saab kasutada erinevat liiki andmeid: märkandmed (arvud, tekstid, ajaväärtused ja tõeväärtused), graafikaandmed: pildid, skeemid, joonised jms. Saab kasutada ka heliandmeid ja multimeedia vahendeid, lugeda andmeid andmebaasidest ja kirjutada neid andmebaasi. Lihtsalt saab kasutada erineva organisatsiooniga andmeid: loendeid, massiive, maatrikseid, sõnastikke, faile jms. Võimalik on luua ja töödelda erineva struktuuriga dokumente. Python on loodud funktsioonidega, mis hõlbustavad andmete analüüsi ja visualiseerimist (“Tutvumine Pythoniga,” n.d.).

# 3 KOODI KIRJUTAMINE

Koodi mõte on töödelda valmis mudeli sisendfaili, mis sisaldab mudeli informatsiooni. Laeva konstruktsiooniks on enamasti jäigastatud plaat, ehk plaat, millele on keevitatud jäikusribid (Joonis 5). Eesmärk on leida mudelist need elemendid, mis asuvad ühenduskohtades ja muuta nende materjali omadusi.




Joonis 5. Laeva konstruktsioon
Allikas: Paik JK (2018), Ultimate Limit State Analysis and Design of Plated Structures.
Teine väljaanne. John Wiley & Sons Ltd lk39

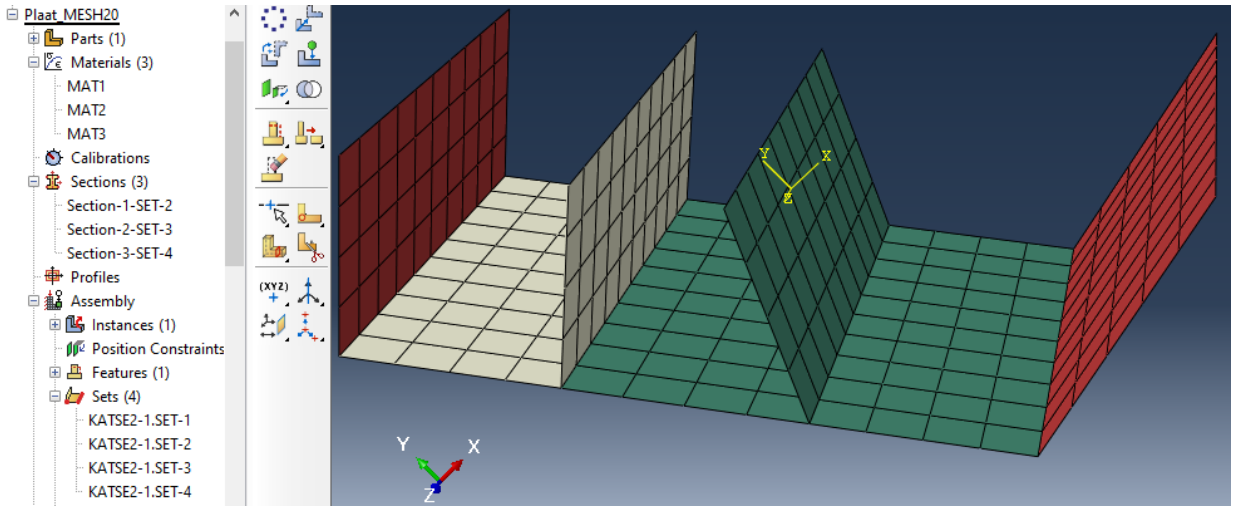
## 3.1 Näidismudel

Näidismudeli sisendfaili andmeid kasutatakse koodi väljatöötamiseks. Laeva põhiline konstruktsioonielement on jäigastatud plaat. Seega valiti näidismudeliks nelja erineva ühendusega jäigastatud plaat (Joonis 6).

Näidismudeli koostamisel võeti aluseks, et tavaliselt koostatakse elementide võrk osaeksemplarides ja kokkupõrke analüüsis on eesmärgiks võtta arvesse termomõjutsooni ainult kokkupõrke kohas, mis enamikel juhtudel jääb ühe osaeksemplari piiridesse. Seega lihtsustuse mõttes eeldati, et mudelis on üks osa ja selle osaeksemplar. See tähendab seda, et *Elemendi ja *Sõlme definitsioon esineb failis vaid ühe korra.

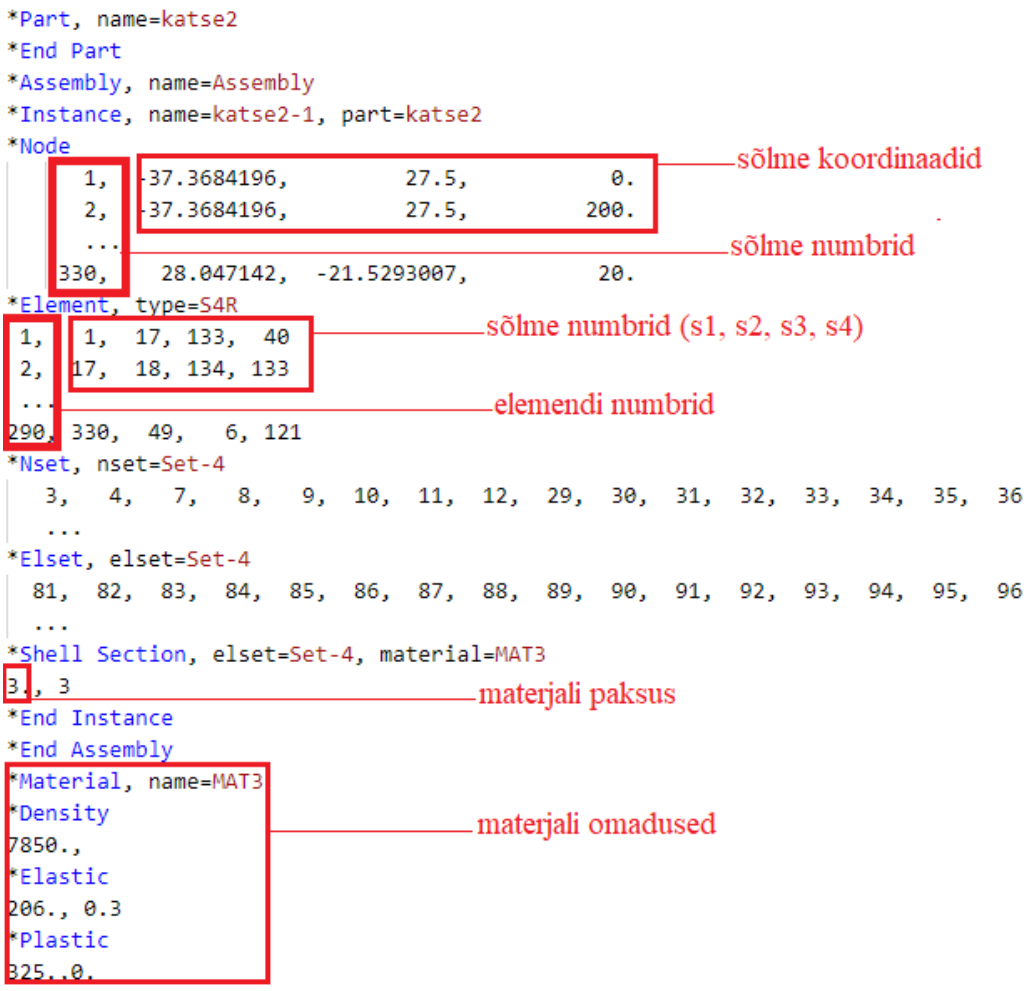
Näidismudel modelleeriti Abaqus tarkvaras. Mudel koosneb ühest osast ning tema osaeksemplarist, kolmest materjali paksusest ja neljast komplektist, mis on defineeritud kolme sektsiooniga. Joonisel tähistavad erinevad värvid erinevaid materjali paksuseid, mis on

määratletud sektsioonidega. Kasutades mudelis erinevaid materjale ja plaadi paksusi, sarnaneb mudel reaalse konstruktsiooniga, mis võimaldab loodavat koodi efektiivselt testida. Reaalses konstruktsioonis võib esineda jäigastajaid, mis ei ole risti plaadiga. Seepärast on lihtsustatud mudelis üks jäigastaja samuti modelleeritud kaldega.




Joonis 6. Kolme erineva materjali paksusega, jäigastatud plaat

Näidismudeli sisendfail koosneb ühest osast (*Part) ja selle osaeksemplarist (*Instance) ning sõlmede (*Node) ja elementide (*Element) definitsioon esineb ühe korra ja on määratletud osaeksemplaris. Elemendid on jaotatud nelja komplekti (*Elset) ja sektsioonides (*Shell Section) on määratud komplektide materjali omadused ning materjali omadused (*Material) on määratletud mudeli tasandil (Joonis 7).




Joonis 7. Abaquse lühendatud näidismudeli sisendfail

## 3.2 Termomõjutsooni kood (HAZ_kood)

Sisendfailist on vaja üles leida naaberelemendid, mis jäävad omavahel määratud nurga alla, koostada nendest elementidest uued termomõjutsooni komplektid ning anda neile elementidele uued materjaliomadused.

Selles töös on Python programmis kirjutatud kood „HAZv1.0“ jaotatud 13 plokki (Lisa 1):

1) import
2) konfiguratsioon
3) materjali omadused
4) sisendfailist elementide ning sõlmede otsimine

5) naaberelementide otsimine
6) naaberelementide normaalide omavahelise nurga leidmine
7) elementide komplekti märksõnade leidmine
8) sektsiooni märksõnade leidmine
9) termojõutsooni elementide komplekti andmeread
10) termojõutsooni sektsiooni andmeread
11)vahefaili kirjutamine
12) termomõjutsooni materjali omadused
13) termomõjutsooni materjalide lisamine lõplikku faili

### 3.2.1 Sisendfailist elementide ning sõlmede otsimine

Kuigi antud töös on arvesse võetud ainult ühe osaeksemplari andmed, võib mudelis olla rohkem osasid. Seega peab kood leidma õige sisendfaili ja failist õige osaeksemplari. Selleks sisestatakse „konfiguratsiooni“ plokki sisendfaili ja osaeksemplari nimed, millega töötama hakatakse.

Kood avab faili ja otsib sisestatud eksemplari alt üles elementide ja sõlmede andmeread ja arvutustõhususe huvides salvestatakse eraldi numpy massiivi. Et massiividega saaks töötada, tuleb „import“ plokki importida *numpy* (Numerical Python) ja *pandas* (Python Data Analysis Library). Numpy on Pythoni teek, mida kasutatakse massiividega töötamiseks (teek-infokandjate korrastatud kogu) ning pandas on tarkvarakogu, mis on kirjutatud Pythoni programmeerimiskeele jaoks andmete manipuleerimiseks ja analüüsimiseks. Eelkõige pakub see andmestruktuure ja toiminguid numbriliste tabelite ja aegridade manipuleerimiseks.

Importimine on Pythoni faili või Pythoni mooduli skriptidele juurdepääsu lubamine teisest Pythoni failist või moodulist. Pythonis saab kasutada ainult funktsioone ja atribuute, millele on programmil juurdepääs.

### 3.2.2 Naaberelementide otsimine

Plaatide külge keevitatud jäigastajad on alati plaadi suhtes mingi nurga all. Et leida nurga all olevaid ühendusi, tuleb leida omavahel määratud nurga all olevad elemendid. Elementide omavahelise nurga võrdlemiseks tuleb kõigepealt leida naaberelemendid, mille normaale saaks hakata võrdlema.

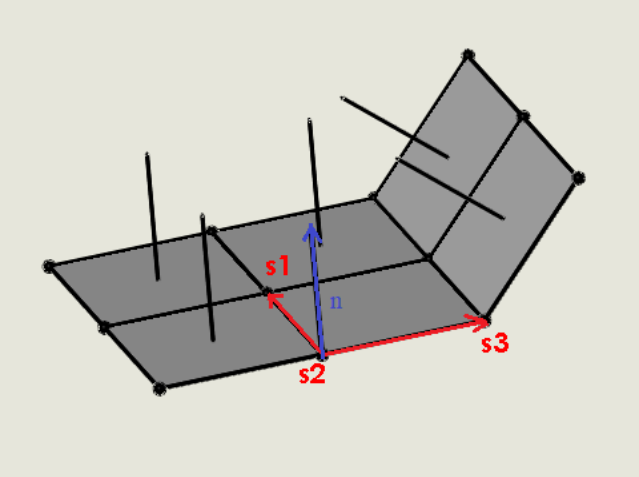
Selleks võrreldakse elementide massiivi andmeid ja leitakse kõigi elementide naabrid, millel on vähemalt üks ühine sõlm. Vähemalt ühe ühise sõlme leidmine on vajalik selleks, et leida kolmnurksete elementide naabreid. Kui on ainult nelinurksed elemendid, piisab vähemalt kahest ühisest sõlmest, mis teeb andmete töötlemise kiiremaks, sest naabreid jääb vähemaks.

Kui naabrid on leitud, koostatakse sõnastik, kuhu lisatakse elementide massiivist elemendi number, selle elemendi sõlmed, mis on eraldatud uude loendisse, ja antud elemendi naabrite loend.

### 3.2.3 Naaberelementide omavahelise nurga leidmine

Naaberelementide omavahelise nurga leidmiseks võrreldakse elementide normaale ( n ), mille leiame elementitasandi vektorite ( $\overrightarrow{S_2S_1} ja \overrightarrow{S_2S_3}$ ) kaudu (Joonis 8).

Normaalide arvutamiseks tuleb importida *math*. Matemaatika moodulit kasutatakse Pythoni matemaatiliste funktsioonide kasutamiseks.



Joonis 8. Elemendi normaal (n)

Igal elemendile on loodud sõnastikus oma sõlmed, millest kasutatakse kolme sõlme: $S_1$, $S_2$, $S_3$. Igal sõlmel on koordinaadid: $S_1(x_1, y_1, z_1)$, $S_2(x_2, y_2, z_2)$, $S_3(x_3, y_3, z_3)$, mis võetakse sõlmede massiivist.

Kui tasandil on kaks mittekollineaarset vektorit $\overrightarrow{S_2S_1} ja \overrightarrow{S_2S_3}$, mis lähtuvad ühisest punktist, siis tasandi normaalvektor $\vec{n}$ avaldub vektorkorrutisena: $\vec{n} = \overrightarrow{S_2S_1} \times \overrightarrow{S_2S_3}$

Seega kõigepealt leiame elemendi kolme sõlme abil kaks elemendi tasandi vektorit (1).

$$\overrightarrow{S_2S_1} = (x_2, y_2, z_2) - (x_1, y_1, z_1) = (x_a, y_a, z_a)$$
$$\overrightarrow{S_2S_3} = (x_2, y_2, z_2) - (x_3, y_3, z_3) = (x_b, y_b, z_b) \quad (1)$$

Seejärel annab vektorkorrutis tulemuseks elemendi normaalvektori (2).

$$\vec{n} = \overrightarrow{S_2S_1} \times \overrightarrow{S_2S_3} = \begin{pmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_a & y_a & z_a \\ x_b & y_b & z_b \end{pmatrix} = (y_a \cdot z_b - z_a \cdot y_b), (z_a \cdot x_b - x_a \cdot z_b), (x_a \cdot y_b - y_a \cdot x_b) = \quad (2)$$
$$= xn\ , yn\ , zn$$

kus $\hat{i}$, $\hat{j}$, $\hat{k}$ on ühikvektorid ja nende vektorkorrutis on null.

Et $\cos\varphi$ ja $\cos(180^\circ - \varphi)$ absoluutväärused on võrdsed, siis kahe elemendi vahelise nurga saame, kui võtame normaalvektorite vahelise nurga valemi lugejast absoluutväärtuse. Seega kahe elemendi vaheline nurk $\varphi$ on määratud valemiga (3).

$$\cos\varphi = \frac{\left|\vec{n}_1 \cdot \vec{n}_2\right|}{\left|\vec{n}_1\right| \cdot \left|\vec{n}_2\right|} = \frac{\left|xn_1 xn_2 + \mathrm{yn}_1 yn_2 + \mathrm{zn}_1 zn_2\right|}{\left|\sqrt{xn_1^{\,2} + \mathrm{yn}_1^{\,2} + \mathrm{zn}_1^{\,2}}\right| \left|\sqrt{xn_2^{\,2} + \mathrm{yn}_2^{\,2} + \mathrm{zn}_2^{\,2}}\right|} \quad (3)$$

$$\varphi = \arccos\varphi$$

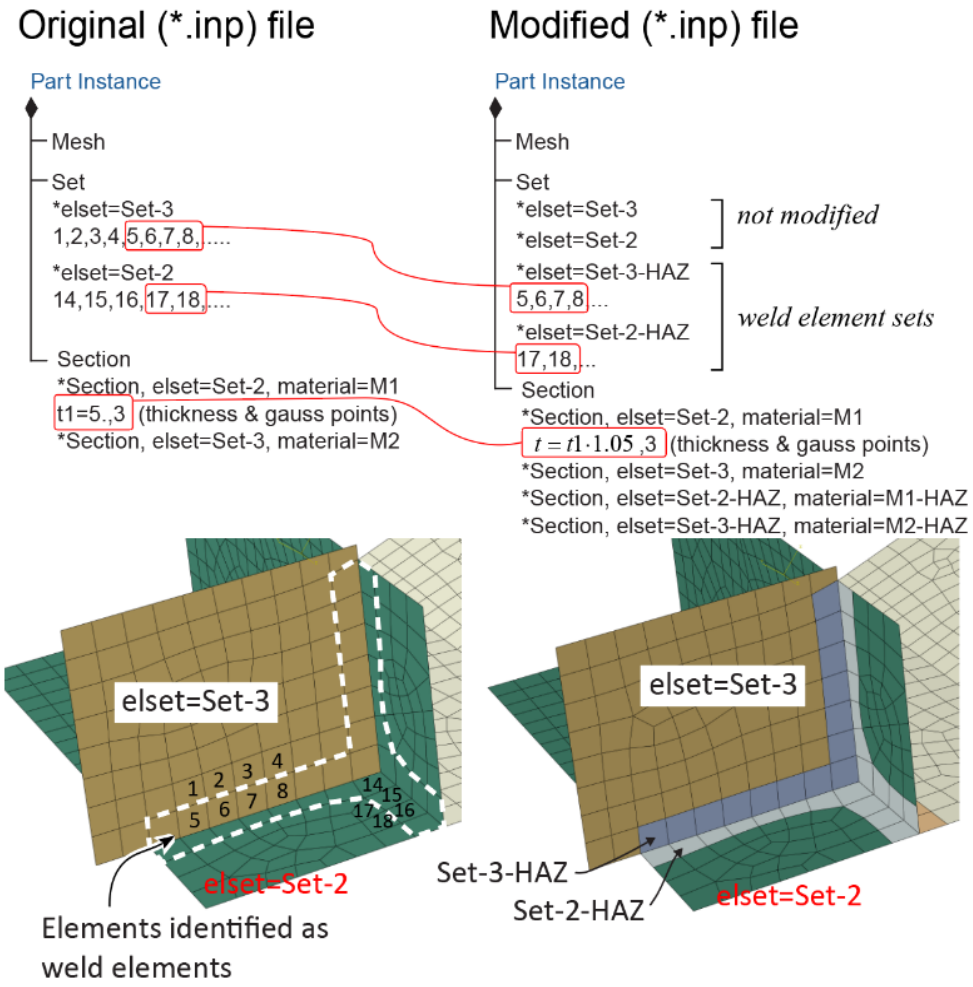
Nurk, mida otsitakse, on defineeritud „konfiguratsioon“ plokis. Otsitav nurk võib olla kas nurkade vahemik või konkreetne nurk. Kui elementide omavaheline nurk jääb defineeritud vahemikku, lisatakse element termomõjutsooni elementide loendisse.

### 3.2.4 Termomõjutsooni elementide komplektid

Termomõjutsooni elementide komplekti koostamisel otsitakse üles komplektide märksõnad (*Elset, elset=Set-4) ja andmeread, mis algavad „*Elset“ käsuga, kuni jõutakse järgmise käsureani, mis defineeritakse „*“ märgiga.

Märksõnadele lisatakse lõppu HAZ, et termomõjutsooni elemendid oleksid eristatavad (*Elset, elset=Set-4-HAZ).

Andmeridasid võrreldakse termomõjutsooni elementide loendiga, mis koostati „naaberelementide omavahelise nurga leidmine“ plokis ja kattuvad elemendid lisatakse vastava komplekti loendisse (Joonis 9).



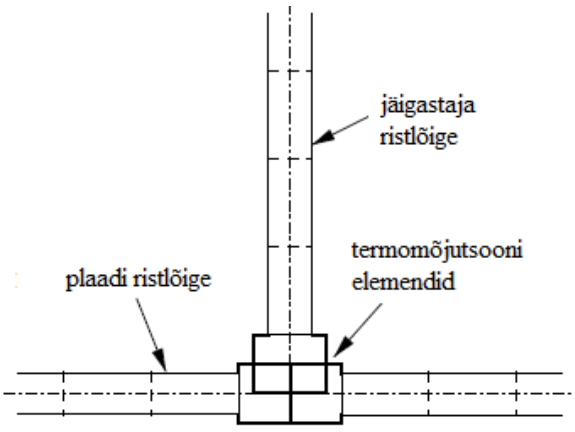

Joonis 9. Termomõjutsooni elementide komplekti genereerimine

### 3.2.5 Termomõjutsooni elementide komplektide sektsioonid

Sektsioonides defineeritakse elemendi komplektide materjali paksus ja omadused, nagu on näidatud joonisel „Abaquse lühendatud näidismudeli sisendfail“ (Joonis 7).

Termomõjutsooni sektsiooni koostamisel otsitakse üles sektsioonide märksõnad (*Shell Section, elset=Set-4, material=MAT3) ja andmeread, mis algavad „*Shell Section“ käsuga, kuni jõutakse järgmise käsureani, mis defineeritakse „*“ märgiga. Märksõnadele lisatakse lõppu HAZ, et termomõjutsooni sektsioonid oleksid eristatavad (*Shell Section, elset=Set-4-HAZ, material=MAT3-HAZ).

Materjali paksuse andmereale liidetakse vajalik keevituse paksus, mis on määratud „konfiguratsioon“ plokis. Liidetud paksusega termomõjutsooni materjali andmeread lisatakse vastava sektsiooni loendisse. Elementide paksuse muutmine on üks võimalikest viisidest, kuidas keevisõmbluse mõju mudelis arvesse võtta. Seda võimalust kasutatakse just väikeste elementidega mudelil. Keevisõmbluste lisamine annab ka sujuvama ristlõike ülemineku jäigastaja ja plaadi vahel (Joonis 10) (Alsos jt. 2009).




Joonis 10. Termomõjutsooni elemendid
Allikas: Alsos jt. 2009

Lisatud keevituse paksust sisendfailis näeme joonisel (Joonis 11).

```
*Shell Section, elset=Set-4, material=MAT3
3., 3
*Shell Section, elset=Set-4-HAZ, material=MAT3-HAZ
4.2, 3
```

Joonis 11. Lisatud keevituse paksus

### 3.2.6 Vahefaili kirjutamine

Uued andmed peab paigutama sisendfailis õigetesse kohtadesse. Termomõjutsooni elementide komplektid ja sektsioonid peavad jääma selle eksemplari sisse (*Instance ja *End instance vahemikku), millega töötati. Materjalide andmed peavad jääma aga koostust välja mudeli tasandile. Seega lisame kõigepealt termomõjutsooni komplektid ja sektsioonid eksemplari sisse õigesse kohta.

Kui termomõjutsooni elementide komplektid ja sektsioonid on määratud, kirjutatakse need uude faili. Uue faili nimi defineeritakse „konfiguratsioon“ plokis.

Kuna Abaqus loeb sisendfaili järjest ning termomõjutsooni elemendid ei ole algsest komplektist eemaldatud (termomõjutsooni elemendid esinevad mõlemas komplektis), tuleb termomõjutsoonide sektsioonid lisada pärast algseid sektsioone. See tagab selle, et failis hiljem ilmuv termomõjutsooni sektsiooni definitsioon kirjutab varem ilmunud algse sektsiooni definitsiooni üle. Seega on oluline, et termomõjutsooni sektsioonid oleksid pärast algseid sektsioone.

Vahefaili kirjutamiseks luuakse uus sisendfail, kuhu kopeeritakse vana fail ja lisatakse termomõjutsooni elementide komplektid ja sektsioonid. Uued andmed lisatakse failis eksemplari lõppu (Joonis 12).

```
*Elset, elset=Set-2-HAZ
   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  31,  32,  33,  34,  35,  36
  37,  38,  39,  40, 214, 218, 222, 226, 230, 234, 238, 242, 246, 250
*Elset, elset=Set-3-HAZ
  41,  42,  43,  44,  45,  46,  47,  48,  49,  50, 131, 132, 133, 134, 135, 136
 137, 138, 139, 140, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 254, 258
 262, 266, 270, 274, 278, 282, 286, 290
*Elset, elset=Set-4-HAZ
 171, 175, 179, 183, 187, 191, 195, 199, 203, 207
*Shell Section, elset=Set-2-HAZ, material=MAT2-HAZ
3.2, 3
*Shell Section, elset=Set-3-HAZ, material=MAT1-HAZ
6.2, 3
*Shell Section, elset=Set-4-HAZ, material=MAT3-HAZ
4.2, 3
*End Instance
```

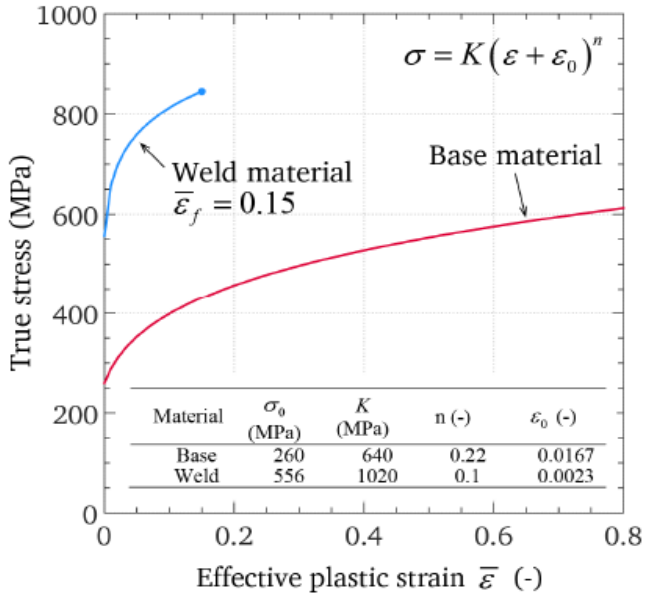
Joonis 12. Termemõjutsooni elementide komplektid ja sektsioonid sisendfailis

### 3.2.7 Termomõjutsooni materjalide omadused

Materjali vastupanu deformeerimisele ja purunemisele iseloomustavad materjalide mehaanilised omadused: tugevus, kõvadus, plastsus, elastsus, sitkus ja haprus. Plastsus on materjali võime talle rakendatud väliskoormuse mõjul muuta purunemata oma kuju ja mõõtmeid ning säilitada jäävat (plastset) deformatsiooni pärast väliskoormuse lakkamist. (Kulu jt. 2015b)

Termomõjutsoonis kasutatakse lähtematerjaliga võrreldes erinevaid materjali omadusi. Keevismaterjali pinge-deformatsiooni suhe on ligilähedane keevismetalli kõvaduse väärtusele termomõjutsoonis. Lisaks modelleeritakse termomõjutsoonides purunemine konstantse ekvivalentse plastse deformatsiooniga 0,15. Pinge-deformatsiooni kõverad nii keevisõmbluse (weld material) kui ka alusmaterjali (base material) jaoks defineeritakse vastavalt graafikule (Joonis 13) (Kõrgesaar jt. 2018).

Vastavalt „materjali omadused“ plokis sisestatud andmetele genereeritakse termomõjutsooni materjali plastsed omadused.




Joonis 13. Keevisõmbluse ja alusmaterjali pinge-deformatsiooni kõver
Allikas: Kõrgesaar jt. 2018

### 3.2.8 Lõpliku termomõjutsooni sisendfaili kirjutamine

Lõpliku termomõjutsooni sinendfaili kirjutamiseks lisatakse uude faili termomõjutsooni materjalide omaduste read. Selleks luuakse uued termomõjutsooni materjalide read, kuhu lisatakse materjali tihedus ja elastsus, mis on defineeritud „materjali omadused“ plokis ning plastsus, mis on genereeritud „termomõjutsooni materjali omadused“ plokis (Joonis 14).

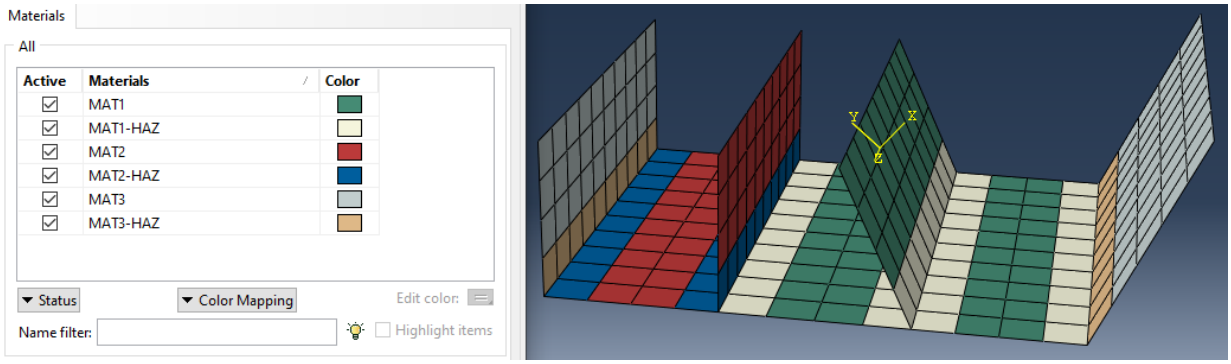
Kuna materjalid määratletakse mudeli tasandil, siis paigutatakse uued materjalid algsete materjalide ette.

```
*Material, name=MAT2-HAZ
*Density
7850.,
*Elastic
206., 0.3
*Plastic
      440.0000,    0.0000
      440.0000,    0.0025
      440.0000,    0.0068
      440.0000,    0.0100
      440.0000,    0.0150
      440.0000,    0.0200
      440.0000,    0.0270
      462.9350,    0.0359
```

Joonis 14. Termomõjutsooni materjali omadused.

### 3.2.9 Valmis kood

Tulemuseks saame termomõjutsoonidega näidismudeli (Joonis 15). Joonisel on erinevad materjaliomadused tähistatud erinevate värvidega. Termomõjutsoonidega näidismudelilt on näha, et kõigis termomõjutsoonide alades on materjaliomadused muudetud õigesti.



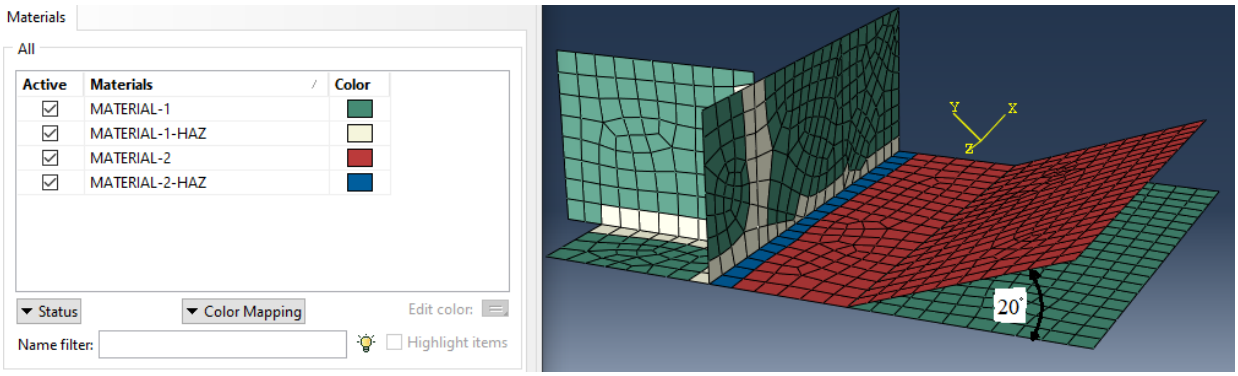

Joonis 15. Termomõjutsoonidega näidismudel

# 4 KOODI TESTIMINE

Kood kirjutati näidismudeli sisendfaili põhjal. Et vaadata, kas kood töötab ka teiste mudelitega, testitakse koodi erinevate mudelite abil.

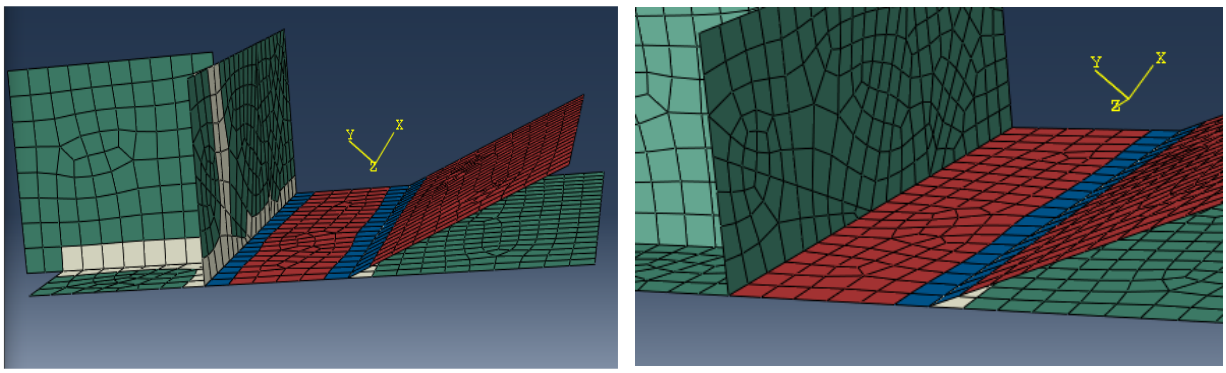
## 4.1 Test 1

Test kirjeldab olukorda, kus plaadil on erinevate nurkade all jäigastajad ja mudel koosneb 1059 elemendist. Otsitakse $85^{\circ} - 95^{\circ}$ nurkasid, mis näitab, kas kood väldib $20^{\circ}$ teravnurka (Joonis 16). Otsitakse $20^{\circ} - 160^{\circ}$ nurkasid, kus kaasatakse kõik nurgad (Joonis 17A). Otsitakse $20^{\circ}$ nurkasid, et leida spetsiifilist nurka (Joonis 17B). Kõigi otsingutega leiti õiged elemendid üles ja lisati elementidele termomõjutsooni materjali omadused.




Joonis 16. $85^{\circ} - 95^{\circ}$ nurga all jäigastajad

A) B)




Joonis 17. A) $20^{\circ}$-$160^{\circ}$ nurga all jäigastajad B) $20^{\circ}$ nurga all jäigastajad

Programmi efektiivsuse katsetamiseks mõõdeti ka operatsioonideks kulunud aega. Seda aega võrreldi ka koodiga, mis on juhendajapoolne edasiarendus antud lõputöös kirjeldatud koodist. Järgnevalt viidatakse antud töös loodud koodile kui HAZv1.0 ja edasiarendatud koodile HAZv2.0. Koodi edasiarendus seisneb korduste vähendamises koodis, asendades need maatriksarvutustega, mis tunduvalt vähendab koodi käitusaega. Maatriksarvutuste peale üleminek tähendas aga ühe lihtsustuse tegemist. Nimelt suudab HAZv2.0 kood protsessida ainult mudeleid, mis on loodud elementidest, millel on 4 sõlme. Kuna katsetes kasutatakse tavaliselt T-ühendusi, ja nende töötlemine on kiirem kui kõikide ühenduste otsimine, siis võrreldakse HAZv2.0 koodiga omavahel ka neid ühendusi.

Koodide tööks kulunud aeg on esitatud tabelis (Tabel 1).

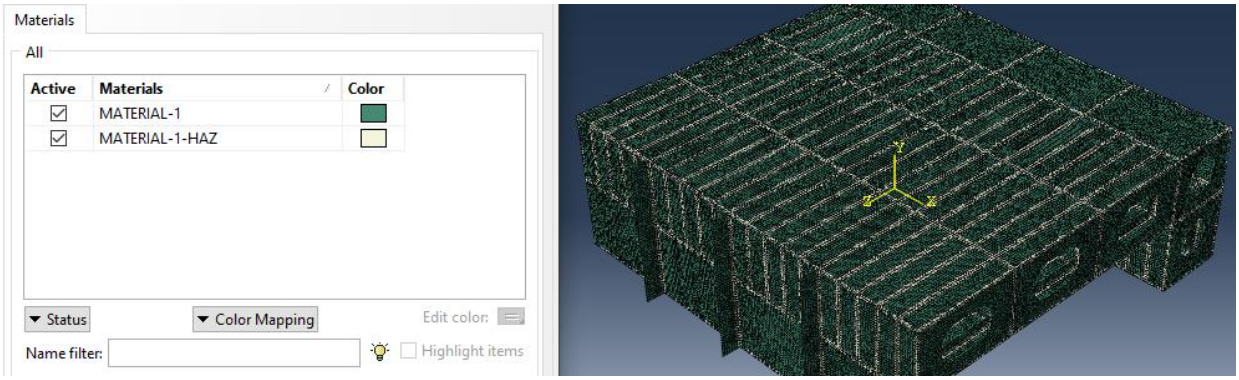
Tabel 1. Test 1- koodi töötamis ajad

| Plokid | HAZv1.0 | HAZv2.0 | |
|---|---|---|---|
| | | T-ühendus | Kõik ühendused |
| Elementide ja sõlmede otsimine | 0.022s | 0.033s | 0.024s |
| Naabrite leidmine | 10.266s | 0.003s | 0.002s |
| Elementide omavahelise nurga leidmine | 11.515s | 0.002s | 0.033s |
| Ülejäänud plokid | 0.047s | 0.148s | 0.115s |
| Kokku | 21.850s | 0.186s | 0.174s |

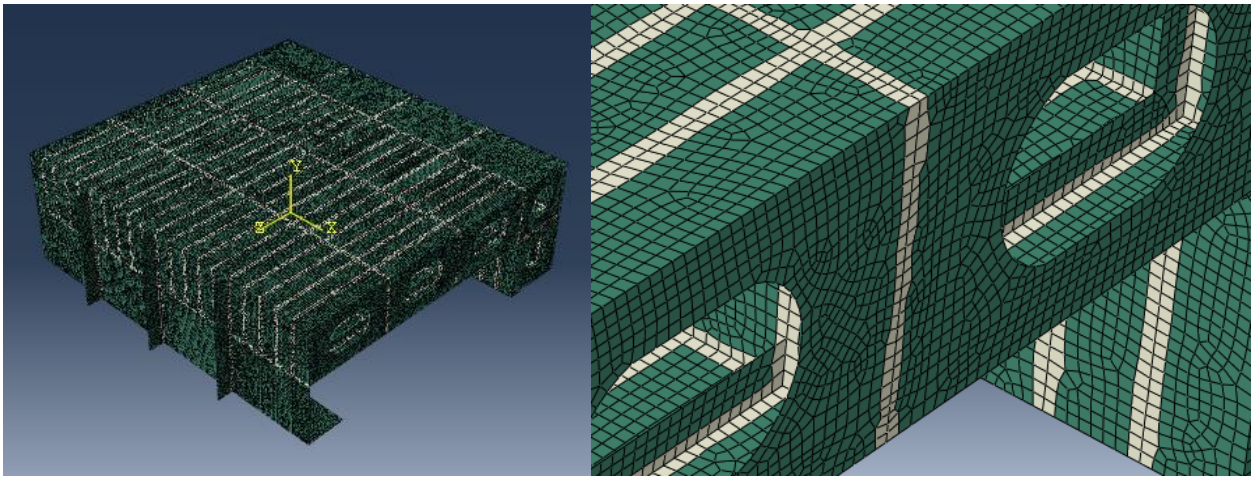
Mudelil 1059 elemendiga töötab HAZv2.0 kood üle 100 korra kiiremini kui HAZv1.0 kood.

## 4.2 Test 2

Testiks kasutatakse laevakonstruktsiooni mudelit ning test kirjeldab olukorda, kus on vaja leida kõiki ühendusi (Joonis 18) ja ainult T-ühendusi (Joonis 19). Mudeliks valiti üks sektsioon laeva küljekonstruktsioonist, mis koosneb 144477 elemendist.




Joonis 18. Laeva konstruktsiooni mudel- kõik ühendused



Joonis 19. Laeva konstruktsiooni mudel- T-ühendus

HAZv1.0 kood jõudis 20 tunniga töödelda 144477 elemendist 51495 ja leida neile elementidele naabrid.

Esimese testiga võrreldes võime oletada, et 60 tundi läheb naabrite leidmiseks ja vähemalt teine 60 tundi läheb normaalide arvutamiseks. Seega kood töötab selle mudeliga vähemalt 120 tundi, mis on 5 päeva.

Maatriksarvutustel põhinev kood HAZv2.0 leiab kõik ühendused (Joonis 18) ning ka ainult T-ühendused (Joonis 19) tunduvalt kiiremini.

Koodi tööks kulunud aeg on esitatud tabelis (Tabel 2).

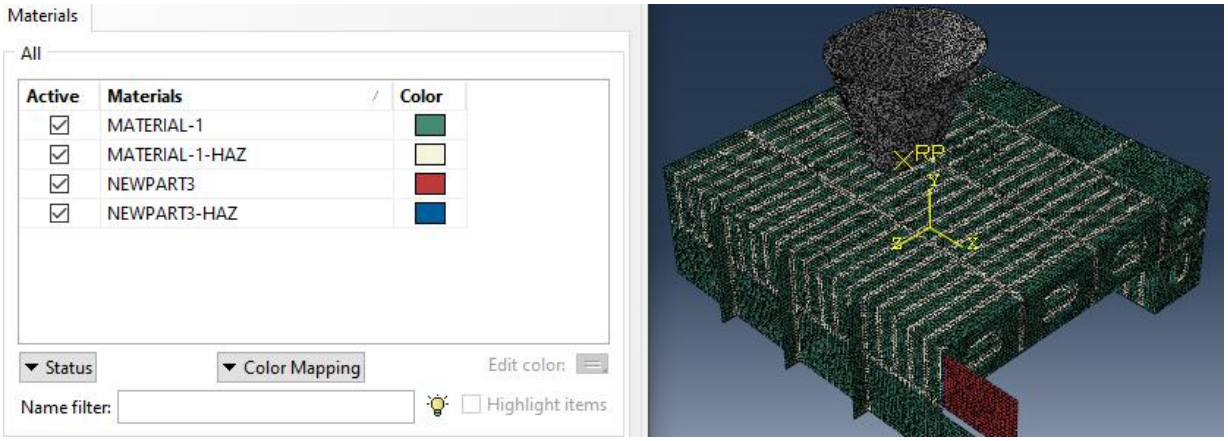
Tabel 2. Test 2 koodi töötamisajad

| Plokid | HAZv1.0 | HAZv2.0 | |
|---|---|---|---|
| | | T-ühendus | Kõik ühendused |
| Elementide ja sõlmede otsimine | 2.576s | 2.277s | 2.706s |
| Naabrite leidmine | 20 tundi<br>51495 elementi | 0.138s | 0.138s |
| Elementide omavahelise nurga leidmine | - | 0.301s | 106.109s |
| Ülejäänud plokid | - | 5.129s | 5.244s |
| Kokku | 5 päeva | 7.845s | 114.197s |

Mudelil 144477 elemendiga leiab HAZv2.0 kood T-ühendused üle 10 korra kiiremini kui kõik ühendused.

## 4.3 Test 3

Test kirjeldab olukorda, kus mudelis on kaks osa: laeva küljekonstruktsioon, mis koosneb 54558 elemendist ja vööripirn (Joonis 20). See on kokkupõrkekatsetes tavaline olukord, kus simuleeritakse laeva konstruktsiooni vastupidavust löögikoormustele.




Joonis 20. Kahe osaga mudel

HAZv1.0 koodil kulus mudeli läbitöötamiseks üle 17,5 tunni ja termomõjutsoonidega sisendfail on vigaselt kirjutatud. Failist puudusid termomõjutsooni elementide komplektid ja termomõjutsooni sektsiooni andmed lisati vale eksemplari sisse. Termomõjutsooni materjalid olid lisatud õigesti, aga kood kaotas originaalandmed, mis jäävad pärast materjali märksõna. Seega tuleks ka üle vaadata plokid „vahefaili kirjutamine“ ja „termomõjutsooni materjalide lisamine lõplikku faili“.

HAZv2.0 koodis on need vead parandatud ja kood lisab termomõjutsooni andmed õigetesse kohtadesse (Joonis 20).

Koodi tööks kulunud aeg on esitatud tabelis (Tabel 3).

Tabel 3. Test 3- koodi töötamis ajad

| Plokid | HAZv1.0 | HAZv2.0 | |
|---|---|---|---|
| | | T-ühendus | Kõik ühendused |
| Elementide ja sõlmede otsimine | 1.31s | 0.940s | 0.893s |
| Naabrite leidmine | 28359s | 0.054s | 0.062s |
| Elementide omavahelise nurga leidmine | 35244s | 0.178s | 6.857s |

| | | | |
|---|---|---|---|
| Ülejäänud plokid | 1.69s | 2.206s | 2.353s |
| Kokku | 63606s | 3.378s | 10.165s |

Mudelil, millel on 54558 elementi, töötab HAZv2.0 kood juba üle 6000 korra kiiremini kui HAZv1.0.

## 4.4 Tulemus

Antud lõputöös loodud kood on eelduseks edasiarendatud koodile ja tõi välja olulised kitsaskohad koodi kirjutamises. Kõige suurem kitsaskoht oli korduste üleliigne kasutamine, mis oluliselt pikendab programmi käitusaega. Koodis tuleb ümber kirjutada plokid „naaberelementide otsimine“ ja „naaberelementide normaalide omavahelise nurga leidmine“, et muuta kood kiireks. Üle tuleb vaadata ka plokid „vahefaili kirjutamine“ ja „termomõjutsooni materjalide lisamine lõplikku faili“, sest praegune kood kirjutab kahe osaga mudelis need valesti.

## 4.5 Edasiarendused

### 4.5.1 Sisendfaili andmed

Uue koodi kirjutamisel peab lähtuma, et vajalikud märksõnad ja andmeread on kõik ühe eksemplari sees (v.a. materjali andmed). Seega otsitakse andmeid, mis jäävad konkreetse eksemplari sisse. Vaja läheb ainult materjali nime ridasid, sest termomõjutsooni materjalidele genereeritakse uued omadused.

Tulemuseks peavad kõik muudetud andmed jääma sama eksemplari sisse (v.a. materjali andmed) ja termomõjutsooni sektsioon peab olema pärast algmaterjali sektsiooni. Kuna materjalid määratletakse mudeli tasandil, siis paigutatakse uued materjalid algsete materjalide ette:

*Instance, name= eksemplari nimi
*Node
Sõlmede andmeread
*Element

Elementide andmeread

*Elset, elset= elemendikomplekti nimi (kõik elementide komplektid)

Elemendikomplekti andmeread

*Shell Section, elset= elemendikomplekti nimi, material=materjali nimi (kõik sektsioonid)

Sektsiooni andmerida

*Elset, elset= elemendikomplekti nimi-HAZ (kõik termomõjutsooni elementide komplektid)

Termomõjutsooni elemendikomplekti andmeread

*Shell Section, elset= elemendikomplekti nimi-HAZ, material=materjali nimi-HAZ (kõik termomõjutsooni sektsioonid)

Termomõjutsooni sektsiooni andmerida

*End Instance

*Material, name= materjali nimi-HAZ (kõik termomõjutsooni materjalid)

*Density

Materjali omaduse andmeread

*Elastic

Materjali omaduse andmeread

*Plastic

Materjali omaduse andmeread

*Material, name= materjali nimi

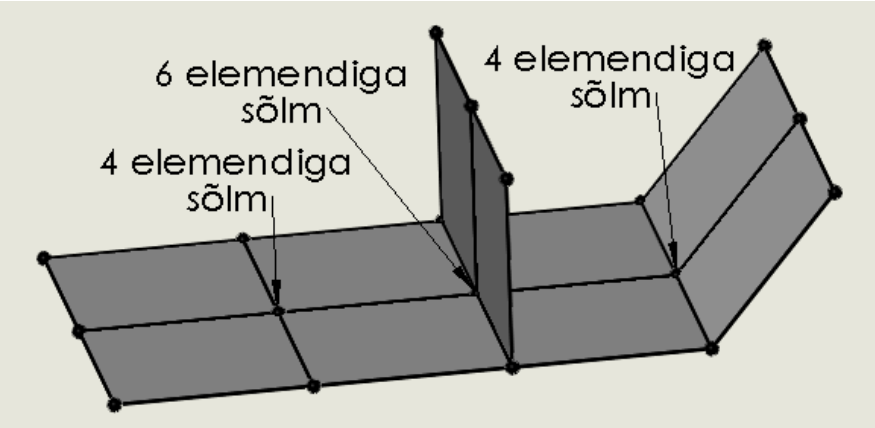
### 4.5.2 Termomõjutsooni elementide leidmine

Praegusel koodil on plokid 5. „naaberelementide otsimine“ ja 6. „naaberelementide normaalide omavahelise nurga leidmine“ väga aeglased ning tuleb leida teine lahendus.

Praegune kood otsib elemente, millel on üks või kaks sama sõlme ja lisab elemendid, sõlmed ja naabrid sõnastikku, mis võtab väga palju aega. Otstarbekas oleks otsida sõlme, millel on vähemalt 4 või 6 sama elementi ja tõhususe huvides töödelda neid, kasutades numpy maatriksarvutusi. Maatriksarvutused eeldavad seda, et kõigil elementidel on neli sõlme, mis välistab kolmnurksete elementidega töötamise, sest nelja ja kolme sõlme ei saa ühes maatriksis kasutada.

Jooniselt (Joonis 21) näeme, et T-ühenduse sõlmel on 6 ühist elementi ja L-ühenduse ja pinna sõlmel 4 ühist elementi ( pinna sõlmel võib olla ka 5 ühist elementi (Joonis 4)). Seega on mõistlik otsida sõlmi, millel on teatud arv ühiseid elemente. Kui on vaja otsida T-ühendusi, siis piisab ainult

sellest, kui otsida 5 suuremaid sama sõlmega elemente. Kui on vaja otsida kõiki ühendusi, siis saab otsida 3 suuremaid sama sõlmega elemente. See otsib pinna- elemendid ka, aga selle jaoks on koodis elementide omavahelise nurga arvutamine.




Joonis 21. 4 ja 6 ühise sõlmega elemendid

Katsetes kasutatakse tavaliselt T-ühendusi ja 6 ühise elemendiga sõlme otsimine välistab pinna elementidega arvutamise.

Elementide omavaheliste nurkade leidmine on vajalik, et leida kõiki ühendusi ja ka selleks, et vajadusel leida teatud nurga all olevaid elemente, nagu oli näidatud esimeses testis. Antud lahendus on aga liiga aeglane ja tuleb leida uus normaalidega arvutamise lahendus.

Ülejäänud kood oli kiire, aga 11. plokis „vahefaili kirjutamine“ ja 13. plokis „termomõjutsooni materjalide lisamine lõplikku faili“ olid vead sees. Seda on arvestatud uue koodi välja töötamisel.

Mihkel Kõrgesaar töötas välja maatriksarvutustega kiirema koodi „HAZv2.0“ ( Lisa 2).ja kood lisab termomõjutsooni elementide andmed õigetesse kohtadesse. Peatükkides „4.2 Test 2“ ja „4.3 Test 3“ näeme, et HAZv2.0 kood teeb kõik operatsioonid õigesti. Samas saab koodiga opereerida ainult nelja sõlmega elementidega.

# KOKKUVÕTE

Lõputöö eesmärgiks oli kirjutada LEM programmi Abaqus mudelile Python tarkvara abil kood, mis automaatselt otsib etteantud mudeli konstruktsioonist liitekohti ja lisab termomõjutsoonile uued materjaliomadused. Liitekohtade materjaliomadusi muudetakse vastavalt kasutaja sisendile.

Selleks tutvuti Abaqus mudeliga ja selle sisendfailiga. Lõplike elementide meetodi (LEM) tarkvaras jagatakse analüüsitav konstruktsioon väiksemateks osadeks ehk lõplikeks elementideks. Koostatakse elementide võrk, milles elemendid on omavahel seotud sõlmedega ja sõlmed on määratletud koordinaatidega (x, y, z). Elementide ja sõlmede kaudu leiti naaberelemendid. Elemendid on omavahel alati mingi nurga all. Naaberelementide normaalvektorite kaudu leiti elementide omavahelised nurgad. Koodis määratud termomõjutsooni elementide omavahelise nurga leidmisel lisati need termomõjutsooni elementide hulka. Termomõjutsooni elementidel muudeti materjali paksust ja omadusi vastavalt kasutaja sisendile. Mudeli sisendfailis asetsevad andmed kindlates kohtades. Termomõjutsooni elemendid ja nende materjali omadused lisati uude sisendfaili õigetesse kohtadesse.

Saadud koodi katsetati erinevate mudelite abil ning vaadeldi, mis piirides kood töötab. Kood leidis keevitatud ühendused ning andis termomõjutsoonidele uued materjaliomadused. Kood töötab mudeli ühe osa eksemplari piires ja nelja sõlmega võrguelementidega.

Töötati välja kood, mis eeltöötleb Abaqus sisendfaili ja lisab mudelisse termomõjutsoonid.

HAZv1.0 kood oli aluseks edasiarendusele HAZv2.0 koodile, mis kasutab maatriksarvutusi ja on seetõttu oluliselt kiirem. Maatriksarvutused eeldavad, et kõigil elementidel on neli sõlme. Laevade kokkupõrke simulatsioonides kasutatakse ka kolme sõlmega elemente. Antud koodi saab edasi arendada, et ta töötaks nelja ja kolme sõlmega elementide puhul.

# VÕÕRKEELNE LÜHIKOKKUVÕTE

Adding the heat-affected zone into the FEM programme ABAQUS model

Tiit Jalasto

The thesis is written in Estonian and contains 31 pages of text, 4 chapters, 21 figures, 3 tables, 10 references and 2 additions.

Keywords: Python, Abaqus, finite element method (FEM), crashworthiness, heat affected zone

Ship collisions and groundings are quite rare but can end with a disasters. It is very important that the ship constructions can resist the impact load in these accidental scenarios. For that, accidents need to be simulated and to get more realistic results, the simulations must be adapted to the real-life situations. Marine constructions are mostly made of metal and the most common way to join metals is welding. Conclusions from the collision simulation may vary greatly depending on whether the heat-affected zone (HAZ) is modelled or not. The heat-affected zone is the unmelted part of the base metal in the welded joint where changes occur in the structure of the material caused by rapid temperature fluctuations during the welding process. In this thesis, the definition of HAZ is broadened to include also the weld metal itself. The collision analysis of marine structures is performed using the finite element method (FEM). In FEM software, the construction under analysis is divided into smaller parts or finite elements. It means that a mesh of elements is compiled in which the elements are connected to each other with nodes defined by coordinates (x, y, z). Due to the complexity of the simulations, it is not guaranteed that the simulation methods and parameters correspond to reality. A part of the simulation development the heat-affected zones could be considered in the analysis. In FEM mode creation, a mesh without the heat-affected zone is created because modeling HAZ is too time-consuming.

In this thesis, a code was written to post-process Abaqus model input file, so that HAZ would be included in the model. During the research, FEM software, Abaqus program model and the input file as well as Python program were familiarized. The code for searching elements from the Abaqus input file located in the heat-affected zone, and for changing their material properties according to the user´s input was written using Python program.

The code was tested with various models to see within which limits the code works. The code could find the welded joints and new material properties could be assigned to the heat-affected zones. The code worked within one part instance of the model and with elements of a four-node meshwork.

HAZv1.0 code was the basis for the further development of HAZv2.0 code which uses matrix calculations and therefore is significantly faster. The matrix calculations assume that all the elements have four nodes. In ship collision simulations, three-node elements are also used. This code can be developed to make it work with both four-node and three-node elements.

# VIIDATUD ALLIKAD


Alsos, H.S., Amdahl, J., Hopperstad, O.S., 2009. On the resistance to penetration of stiffened plates, Part II: Numerical analysis. Int. J. Impact Eng. 36, 875–887. https://doi.org/10.1016/j.ijimpeng.2008.11.004

Berntsson, K., Kõrgesaar, M., Goncalves, B.R., Romanoff, J., 2019. The influence of modelling weld effects when optimizing thin-walled structures for crashworthiness. Proc. Int. Offshore Polar Eng. Conf. 4, 4280–4287.

Kõrgesaar, M., Romanoff, J., Remes, H., Palokangas, P., 2018. Experimental and numerical penetration response of laser-welded stiffened panels. Int. J. Impact Eng. 114, 78–92. https://doi.org/10.1016/j.ijimpeng.2017.12.014

Kõrgesaar, M., Romanoff, J., St-Pierre, L., Varsta, P., 2019. Effect of weld modelling on crashworthiness optimization. Trends Anal. Des. Mar. Struct. - Proc. 7th Int. Conf. Mar. Struct. MARSTRUCT 2019 231–237. https://doi.org/10.1201/9780429298875-27

Kulu, P., Kübarsepp, J., Laansoo, A., Veinthal, R., 2015a. Konstruktsioonimaterjalide tehnoloogia.

Kulu, P., Kübarsepp, J., Laansoo, A., Veinthal, R., 2015b. Materjalitehnika I.

Lõplike elementide meetod – Vikipeedia [WWW Document], n.d. URL https://et.wikipedia.org/wiki/Lõplike_elementide_meetod (23.02.21).

Palokangas, P., fi, pekka palokangas aalto, Palokangas, P., Romanoff, J., fi, pekka palokangas aalto, 2016. Quasi-static indentation experiments and simulations of stiffened steel panels 1–114.

Simulia, D.S., 2014. Abaqus 6.14 / Analysis User's Guide. ABAQUS 6.14 Anal. User's Guid. I, 862.

Tutvumine Pythoniga, n.d. http://rlpa.ttu.ee/python/Python.pdf.(07.03.2021)

# LISAD

## Lisa 1 / HAZv1.0/

```
import numpy as np
import math
import pandas as pd
#%%
# konfiguratsioon #
nurgad = (15, 165)
keevitusepaksus = 1.2
file= 'T04-plaat.inp'
out_file=file[:-4]+'_termo.inp'
instance='Part-3-1'           #T04-plaat
#instance='TNO_iges_struc-1'    #T05-sidestructure
#instance='Merge-1'            #T06-Mutlipart
#instance='katse2-1'           #Plaat_MESH20
#%%
# materjali omadused #
dens=7850.
Elastus=2.06e+11
poisson=0.3
sy= 440.0000
eplat= 0.0275
n_mat= 0.1965
A= 999.7916
Q= 181.5320
C1= 1.6628
alfa= 0.4297
ep=[0.0000,0.0025,0.00679,0.0100,0.015, 0.0200,0.0270,
0.0359,0.0387,0.0466,0.0576,0.0692,0.0805,0.0915,0.1026,0.1135,0.1283,0.1443,0.1607,0.1789,0.1980,0.2200,0.2453,0.2760,0.
3073,0.3482,0.3867,0.4354,0.4746,0.5185,0.5731,0.6387,0.7010,0.80,0.9,1.,1.1,1.4]
#%%
###  sisendfailist elementide ning sõlmede otsimine ###
start_el=0
start_node=0
in_instance=False
EL=[]
Node=[]
with open(file) as f:
    for row_num, line in enumerate(f, 1):
        if '*Instance, name='+instance in line:
            in_instance = True
        if '*End Instance' in line and in_instance:
            in_instance = False
            break
        if in_instance:
            if '*Element' in line:
                start_el=row_num
                continue
            if  start_el:
                try:
                    if '*' in line:
                        break
                    else:
                        elist = [int(x) for x in line.split(',') if x.strip().isdigit()]
                        EL.append(elist)
                except:
                    pass
            if '*Node' in line:
```

```
            start_node=row_num
            continue
        if  start_node:
            try:
                if '*' in line:
                    break
                else:
                    n_table= [float(x) for x in line.split(',') if x.strip(',')]
                    Node.append(n_table)
            except:
                pass
El1= np.array(EL, dtype=object)
NodeXYZ= np.array(Node, dtype=object)
#%%
### naaberelementide otsimine ###
Naabrid = []
for i in El1:
    noded = i[1:]
    naabrid = []
    for e in El1:
        if not e[0] == i[0]:
            noded2 = e[1:]
            same_nodes = 0
            for n in noded:
                if list(noded2).count(n) == 1:
                    same_nodes += 1
            if same_nodes >= 2:
                naabrid.append(e[0])
    Naabrid.append({"elem_id":i[0], "noded":list(noded), "naabrid":naabrid})
#%%
###  naaberelementide normaalide omavahelise nurga leidmine ###
muudetavad_elemendid = []
for elem in Naabrid:
    elem1_noded = []
    for node in elem["noded"]:
        otsitav = list(filter(lambda x: (x[0] == node), NodeXYZ))[0]
        elem1_noded.append(list(otsitav[1:]))
    for naaber in elem["naabrid"]:
        elem2_noded = [] # loend sees [x,y,z]
        naaber = list(filter(lambda x: (x["elem_id"] == naaber), Naabrid))[0]
        for node in naaber["noded"]:
            otsitav = list(filter(lambda x: (x[0] == node), NodeXYZ))[0]
            elem2_noded.append(list(otsitav[1:]))
        x11 = elem1_noded[0][0] - elem1_noded[1][0]
        y11 = elem1_noded[0][1] - elem1_noded[1][1]
        z11 = elem1_noded[0][2] - elem1_noded[1][2]
        x12 = elem1_noded[2][0] - elem1_noded[1][0]
        y12 = elem1_noded[2][1] - elem1_noded[1][1]
        z12 = elem1_noded[2][2] - elem1_noded[1][2]
        x21 = elem2_noded[0][0] - elem2_noded[1][0]
        y21 = elem2_noded[0][1] - elem2_noded[1][1]
        z21 = elem2_noded[0][2] - elem2_noded[1][2]
        x22 = elem2_noded[2][0] - elem2_noded[1][0]
        y22 = elem2_noded[2][1] - elem2_noded[1][1]
        z22 = elem2_noded[2][2] - elem2_noded[1][2]
        xa1,ya1,za1 = [(y11*z12-z11*y12),(z11*x12-x11*z12),(x11*y12-y11*x12)]
        xa2,ya2,za2 = [(y21*z22-z21*y22),(z21*x22-x21*z22),(x21*y22-y21*x22)]
        upper = (xa1*xa2 + ya1*ya2 + za1*za2)
        lower = ((xa1**2 + ya1**2 + za1**2)**(1/2))*((xa2**2 + ya2**2 + za2**2)**(1/2))
        final = int(round(math.acos(round(upper / lower,7))*180/math.pi))
        if type(nurgad) == int:
            if final == nurgad:
                muudetavad_elemendid.append(elem["elem_id"])
        else:
```

```
            if final in range(nurgad[0], nurgad[1]):
                muudetavad_elemendid.append(elem["elem_id"])
EL_HAZ_all = list(set(muudetavad_elemendid))
#%%
### elementide komplekti märksõnade leidmine  ###
with open(file) as f:
    lines = f.readlines()
read_rea_nr=[]
Elemset=[]
for num, line in enumerate(lines, 1):
    if '*Instance, name='+instance in line:
        in_instance = True
    if '*End Instance' in line and in_instance:
        in_instance = False
        break
    if '*Elset' in line and in_instance:
        read_rea_nr.append(num)
        Elemset.append(line.split(','))
Elemset = [[i[0]]+[i[1].replace("\n", "")+"-HAZ\n"] for i in Elemset]
#%%
###  sektsiooni märksõnade leidmine ###
Sect_rea_nr=[]
Shell_Sect=[]
for num, line in enumerate(lines, 1):
    if '*Instance, name='+instance in line:
        in_instance = True
    if '*End Instance' in line and in_instance:
        in_instance = False
        break
    if '*Shell Section' in line and in_instance:
        Sect_rea_nr.append(num)
        Shell_Sect.append(line.split(','))
Shell_Sect = [[i[0]] + [i[1]+"-HAZ", i[2].replace("\n", "")+"-HAZ\n"] for i in Shell_Sect]
#%%
# termojõutsooni elementide komplekti andmeread #
def read_between(start_line_nr,lines2read):
    dat_read=[]
    for num, line in enumerate(lines2read):
        if  num >= start_line_nr:
            if '*' in line:
                break
            else:
                line_data = [float(x) for x in line.split(',') if x.strip()]
                dat_read.append(line_data)
    dat_pd=pd.DataFrame(dat_read)
    return dat_pd
HAZ_elsets=[]
out_data=[]
for i in range(len(read_rea_nr)):
    out_data.append(read_between(read_rea_nr[i],lines))
    elset1=out_data[i]*out_data[i].isin(EL_HAZ_all)
    el1=[elset1.to_numpy().flatten()!=0]
    t=elset1.to_numpy().flatten()
    HAZ_elsets.append(t[tuple(el1)])
#%%
# termojõutsooni sektsiooni andmeread #
Sectinfo=[]
for i in range(len(Sect_rea_nr)):
    tmp=read_between(Sect_rea_nr[i],lines)
    Sectinfo.append(tmp.to_numpy()[0])
Sectinfo = [ [list(i)[0]+keevitusepaksus, int(list(i)[1])] for i in Sectinfo]
#%%
#### vahefaili kirjutamine ###
def vorminda(epi, leng=4):
```

```
    epi = int(epi)
    return " "*(leng-len(str(epi))) + str(epi)
def lisa_setid_uude_faili(elsets):
    splitter = "*End Instance"
    with open(file, "r") as f:
        vana_sisu = f.read()
    vana_sisu = vana_sisu.split(splitter)
    vahe_sisu = []
    try:
        max_vorminda_length = len(str(int(sorted(elsets[-1], reverse=True)[0]))) + 1
    except Exception as e:
        print(e)
        max_vorminda_length = 4
    for i in range(len(elsets)):
        set_content=",".join(Elemset[i])
        j=1
        nr_f_rows=16
        for k,epi in enumerate(elsets[i]):
            if math.isnan(epi):
                j+=1
                continue
            if k+1<nr_f_rows*j:
                if k+1==len(elsets[i]):
                    set_content += f'{vorminda(epi, max_vorminda_length)}\n'
                else:
                    set_content += f'{vorminda(epi, max_vorminda_length)},'
            else:
                set_content += f'{vorminda(epi, max_vorminda_length)}\n'
                j+=1
        vahe_sisu.append(set_content)
    for i in range(len(vahe_sisu)):
        if vahe_sisu[i][-1] == ",":
            vahe_sisu[i] = vahe_sisu[i][:-1]
    for i in range(len(Shell_Sect)):
        vahe_sisu.append(",".join(Shell_Sect[i])+", ".join([str(j) for j in Sectinfo[i]]))
    uus_sisu = vana_sisu[0]+"\n".join([i.strip() for i in vahe_sisu])+"\n"+splitter+vana_sisu[1]
    with open(out_file, "w") as f:
        f.write(uus_sisu)
lisa_setid_uude_faili(HAZ_elsets)
#%%
### termomõjutsooni materjali omadused ###
def s_combined(ep,sy,eplat,n,A,Q,C1,alfa):
    # global ep
    s_model=[];s_swift=[];s_voce=[]
    e0=(sy/A)**(1/n)-eplat
    for i,ei in enumerate(ep):
        if ei < eplat:
            s_model.append(sy)
            s_swift.append(sy)
            s_voce.append(sy)
        else:
            s_swift.append(A*(ei+e0)**n)
            s_voce.append(sy+Q*(1-np.exp(-C1*(ei))))
            s_model.append(alfa*s_swift[i]+(1-alfa)*s_voce[i])
    return s_model
#%%
### termomõjutsooni materjalide lisamine lõplikku faili ###
def lisa_materjalid_uude_faili(mats):
    splitter = "*Material"
    with open(out_file,"r") as f:
        vana_sisu = f.read()
    vahe_sisu = vana_sisu.split(splitter)
    vahe_sisu = [vahe_sisu[0], splitter.join(vahe_sisu[1:])]
    smodel=s_combined(ep,sy,eplat,n_mat,A,Q,C1,alfa)
```

```
    sisu = ""
    for mat in mats:
        sisu += f"*Material, name={mat}*Density\n{dens},\n*Elastic\n{Elastus},{poisson} \n*Plastic\n"
        for i in range(len(ep)):
            sisu += "{:14.4f}, {:9.4f} \n".format(smodel[i],ep[i])
    lopp_sisu = vahe_sisu[0]+sisu+splitter+vahe_sisu[1]
    with open(out_file,"w") as f:
        f.write(lopp_sisu)
Material_names=np.unique([i[2].split('=')[1] for i in Shell_Sect] )
lisa_materjalid_uude_faili(Material_names)
```

**Lisa 2 / HAZv2.0/**

```
import numpy as np
import pandas as pd
import fileinput
from shutil import copy2
#%%
# konfiguratsioon #
keevitusepaksus = 0.1
exclude_element_nrs=100000
file= 'T06-Mutlipart.inp'
out_file=file[:-4]+'_HAZ.inp'
#Instance_list=['Part-3-1']                 #T04-plaat
#Instance_list=['TNO_iges_struc-1']    #T05-sidestructure
Instance_list=['Merge-1']                  #T06-Mutlipart
#Instance_list=['katse2-1']                  #Plaat_MESH20
copy2(file,out_file)
#%%
# materjali omadused #
dens=7850.
Elastus=2.06e+11
poisson=0.3
sy= 440.0000
eplat= 0.0275
n_mat= 0.1965
A= 999.7916
Q= 181.5320
C1= 1.6628
alfa= 0.4297
ep=[0.0000,0.0025,0.00679,0.0100,0.015, 0.0200,0.0270,
0.0359,0.0387,0.0466,0.0576,0.0692,0.0805,0.0915,0.1026,0.1135,0.1283,0.1443,0.1607,0.1789,0.1980,0.2200,0.2453,0.2760,0.
3073,0.3482,0.3867,0.4354,0.4746,0.5185,0.5731,0.6387,0.7010,0.80,0.9,1.,1.1,1.4]
c1mat= 0.205
c2mat=385.
c3mat=0.972
#%%
### sisendfailist elementide ning sõlmede otsimine  ###
start_el=0
start_node=0
partrow=10000000
EL=[]
Node=[]
with open(file) as f:
    lines = f.readlines()
    for row_num, line in enumerate(lines, 1):
        if not Instance_list:
            partrow=0
        else:
            if '*Instance, name='+Instance_list[0] in line:
                partrow=row_num
        if '*Element,' in line and row_num>partrow:
```

```
            if '*Element, type=MASS,' in line:
                continue
            else:
                start_el=row_num
            continue
        if  start_el:
            try:
                if '*' in line:
                    start_el=False
                    continue
                else:
                    elist = [int(x) for x in line.split(',') if x.strip().isdigit()]
                    EL.append(elist)
            except:
                pass
with open(file) as f:
    for row_num, line in enumerate(lines, 1):
        if not Instance_list:
            partrow=0
        else:
            if '*Instance, name='+Instance_list[0] in line:
                partrow=row_num
        if '*Node' in line and row_num>partrow:
            start_node=row_num
            continue
        if  start_node:
            try:
                if '*' in line:
                    start_node=False
                    continue
                else:
                    n_table= [float(x) for x in line.split(',') if x.strip(',')]
                    Node.append(n_table)
            except:
                pass
NodeXYZ= np.array(Node)
ELP=pd.DataFrame(EL)
ELs=ELP.to_numpy()
Nods=ELs[:,1:]
shnormCrit=0.1
# sys.exit()
def sortNstrip(argn, A, B):
    argn = argn[(A[argn] > 3)]
    indx = B[argn]
    return (argn, indx)
def findSharednodes(ELs,Nods):
    iE = np.linspace(0, len(ELs) - 1, len(ELs))
    EP = np.array([iE, iE, iE, iE]).astype(int)
    fEP = np.ndarray.flatten(EP.T)
    fEPP = np.ndarray.flatten(np.array([ELs[:,0], ELs[:,0], ELs[:,0], ELs[:,0]]).astype(int).T)
    fNods = np.ndarray.flatten(np.array(Nods))
    Nodnum, ind, count = np.unique(fNods, return_inverse=True, return_counts=True)
    NodCount = count[ind] #
    argn = np.argsort(fNods)
    argn, dummy = sortNstrip(argn, NodCount, fNods)
    indx = fEP[argn]
    iweldNods = argn
    weldNods = fNods[argn]
    iweldEL = indx
    weldEL = fEPP[argn]
    return(iweldNods,weldNods,iweldEL,weldEL,fEPP)
iweldNods,weldNods,iweldEL,weldEL,fEPP=findSharednodes(ELs,Nods)
def findSharedShellnormals(NodeXYZ,iweldEL,Nods,weldNods,shnormCrit):
    Nxyz = NodeXYZ #node nr + koordinaadid
```

```
    NT = np.array(Nxyz).T
    i = int(max(NT[0]) + 1)
    NxyzN = np.zeros((i, 3))
    NxyzN[NT[0].astype(int)] = NT[1:].T
    indx = iweldEL
    NodsW = np.array(Nods[indx])
    NodsT = NodsW.T.astype(int)
    Nxyz1 = NxyzN[NodsT[0]]
    Nxyz2 = NxyzN[NodsT[1]]
    Nxyz3 = NxyzN[NodsT[2]]
    Vec1 = Nxyz1 - Nxyz2
    Vec2 = Nxyz3 - Nxyz2
    n = len(NodsT[0])
    Norm = np.zeros((n, 3))
    Norm = np.cross(Vec1, Vec2, axis=1)
    absNorm = np.sqrt(np.sum((Norm ** 2), axis=1))
    absNorm = np.array([absNorm, absNorm, absNorm])
    uNorm = Norm / absNorm.T
    dweldNods = np.diff(weldNods)
    indTmp = np.nonzero(dweldNods)
    indNS = np.append(np.array([0]), np.array(indTmp) + 1)
    ind12 = np.array([])
    for i in range(len(indNS)-1):
        i1, i2 = indNS[i], indNS[(i + 1)]
        Corr = np.sum((uNorm[i1:i2 - 1] * uNorm[i1 + 1:i2]), axis=1)
        dCorr = np.diff(abs(Corr))
        if max(abs(dCorr)) < shnormCrit:
            ind12 = np.append(ind12, np.arange(i1, i2))
        if i==len(indNS)-2:
            print('last column')
            Corr = np.sum((uNorm[i2:len(uNorm)-1] * uNorm[i2+1:len(uNorm)]), axis=1)
            dCorr = np.diff(abs(Corr))
            if max(abs(dCorr)) < shnormCrit:
                ind12 = np.append(ind12, np.arange(i2, len(uNorm)))
    indx = np.delete(indx, ind12.astype(int))
    return indx
weld_index=findSharedShellnormals(NodeXYZ,iweldEL,Nods,weldNods,shnormCrit)
EL_HAZ_all=ELs[weld_index][:,0]
#%%
# ABI funktsioonid #
with open(file) as f:
     lines = f.readlines()
def find_from_file(findthis, lines,*args):
    rea_nr=[]
    rea_tekst=[]
    startPart=[10e+9]
    if len(args)==1:
        pname=args[0]
        startPart,_=find_from_file(pname,lines)
        for num, line in enumerate(lines, 1):
            if findthis in line and num>startPart[0]:

                rea_nr.append(num)
                rea_tekst.append(line.split(','))
    else:
        for num, line in enumerate(lines, 1):
            if findthis in line:
                rea_nr.append(num)
                rea_tekst.append(line.split(','))
    return (rea_nr, rea_tekst)
def read_between(start_line_nr,lines2read):
    dat_read=[]
    for num, line in enumerate(lines2read):
        if  num >= start_line_nr:
```

```
            if '*' in line:
                break
            else:
                line_data = [float(x) for x in line.split(',') if x.strip()]
                if len(line_data)==3 and int(line_data[2])==1:
                    line_data=np.linspace(line_data[0],line_data[1],int(line_data[1]-line_data[0]+1))
                dat_read.append(line_data)
    dat_pd=pd.DataFrame(dat_read)
    return dat_pd
def s_combined(ep,sy,eplat,n,A,Q,C1,alfa):
    s_model=[];s_swift=[];s_voce=[]
    e0=(sy/A)**(1/n)-eplat
    for i,ei in enumerate(ep):
        if ei < eplat:
            s_model.append(sy)
            s_swift.append(sy)
            s_voce.append(sy)
        else:
            s_swift.append(A*(ei+e0)**n)
            s_voce.append(sy+Q*(1-np.exp(-C1*(ei))))
            s_model.append(alfa*s_swift[i]+(1-alfa)*s_voce[i])
    return s_model
#%%
# Leia muudetavad setid ja tee uued setid #
Sect_rea_nr, Shell_Sect=find_from_file('*Shell Section', lines)
Setname_2_look=[i[1].split('=')[1] for i in Shell_Sect]
ELset_line=[]
ELset_line.append(["*Elset, elset="+".join(j) for j in Setname_2_look])
read_rea_nr=[]
i=0
for elsetL in ELset_line[0]:
    for num, line in enumerate(lines, 1):
        if elsetL in line:
            read_rea_nr.append(num)
            i+=1
            if i==len(ELset_line[0]):
                break
HAZ_elsets=[]
out_data=[]
for i in range(len(read_rea_nr)):
    out_data.append(read_between(read_rea_nr[i],lines))
    elset1=out_data[i]*out_data[i].isin(EL_HAZ_all)
    el1=[elset1.to_numpy().flatten()!=0]
    t=elset1.to_numpy().flatten()
    t=np.nan_to_num(t)
    if not np.any(t):
        continue
    writeto=t[tuple(el1)]
    writeto=np.trim_zeros(writeto)
    HAZ_elsets.append(writeto)
#%%
# Uue sektsiooni koostamine #
Elsets_new=[]
for i in Shell_Sect:
    try:
        Elsets_new.append([i[1].split('_')[1]])
    except:
        Elsets_new.append([i[1].split('=')[1]])
ELsets_haz=[]
ELsets_haz.append(["*Elset, elset="+"".join(i[0])+"-HAZ" for i in Elsets_new[:]])
Shell_Sect_new=[]
for ind,shell in enumerate(Shell_Sect,start=0):
    if shell[2][-1:]=="\n":
        Shell_Sect_new.append([shell[0], 'elset='+Elsets_new[ind][0]+"-HAZ", shell[2].replace("\n", "-HAZ\n")])
```

```
    if shell[2][-1:] != "\n":
        Shell_Sect_new.append([shell[0], 'elset='+Elsets_new[ind][0]+"-HAZ", shell[2]+"-HAZ\n"])
#%%
# Uue materjali koostamine #
Material_names=np.unique([i[2].split('=')[1] for i in Shell_Sect_new] )
Mat_rea_nr,_=find_from_file('*Material', lines)
#%%
# termojõutsooni sektsiooni andmeread #
Sectinfo=[]
for i in range(len(Sect_rea_nr)):
    tmp=read_between(Sect_rea_nr[i],lines)
    Sectinfo.append(tmp.to_numpy()[0])
Sectinfo = [ [list(i)[0]+keevitusepaksus, int(list(i)[1])] for i in Sectinfo]
#%%
# UUE FAILI KIRJUTAMINE #
fnew = fileinput.input(out_file, inplace=True)
for n, line in enumerate(fnew, start=1):
    if line.startswith('*End Assembly'):
        for i in range(len(HAZ_elsets)):
            print(ELsets_haz[0][i]+',instance='+Instance_list[0])
            for j,item in enumerate(HAZ_elsets[i],start=1):
                print('{:d},'.format(int(item)),end='')
                if (j/16).is_integer():
                    print('')
                if j==len(HAZ_elsets[i]):
                    print('\n')
    print(line, end='')
    if line.startswith('*Instance, name='+Instance_list[0]):
        # print('you are stupid')
        for i in range(len(HAZ_elsets)):
            print(ELsets_haz[0][i])
            for j,item in enumerate(HAZ_elsets[i],start=1):
                print('{:d},'.format(int(item)),end='')
                if (j/16).is_integer():
                    print('')
                if j==len(HAZ_elsets[i]):
                    print('\n')
with open(out_file) as f:
    lines = f.readlines()
New_Sect_start,_=find_from_file('*End Instance', lines,'*Instance, name='+Instance_list[0])
fnew = fileinput.input(out_file, inplace=True)
for n, line in enumerate(fnew, start=1):
    # print(line)
    if n==New_Sect_start[0]:
    # if line.startswith('*End Instance'):
        # print('you are stupid')
        for i in range(len(HAZ_elsets)):
            print(",".join(Shell_Sect_new[i])+", ".join([str(j) for j in Sectinfo[i]]))
    print(line, end='')
#%%
### termomõjutsooni materjali omaduste arvutused ###
smodel=s_combined(ep,sy,eplat,n_mat,A,Q,C1,alfa)
# Add Materials
fnew = fileinput.input(out_file, inplace=True)
for n, line in enumerate(fnew, start=1):
    if line.startswith('** MATERIALS'):
        for i in range(len(Mat_rea_nr)):
            print("*Material, name={}*Density\n{},\n*Elastic\n{},{} ".format(Material_names[i],dens,Elastus,poisson))
            print('*Plastic')
            for j in range(len(ep)):
                print("{:14.4f},{:9.4f}".format(smodel[j],ep[j]))
            # print(string)
    print(line, end='')
fnew.close()
```

## Lisa 3 / Lihtlitsents /

rektori 07.04.2020 käskkirjale nr 1-8/17

**Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]**

Mina Tiit Jalasto

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose Termomõjutsooni lisamine LEM programm ABAQUS mudelisse, mille juhendaja on DSc, Mihkel Kõrgesaar,

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

---

12.05.2021

[1] *Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. jq 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.*