Република Србија
МИНИСТАРСТВО НАУКЕ,
ТЕХНОЛОШКОГ РАЗВОЈА И ИНОВАЦИЈА
Матични научни одбор за машинство и
индустријски софтвер
Датум: 22.04.2025.
Београд,
Немањина 22-26

Универзитет у Крагујевцу, Факултет инжењерских наука

34000 Крагујевац, Србија

Сестре Јањић 6

Поштовани,

Факултет инжењерских наука у Крагујевцу, односно Наставно научно веће Факултета, поднело је Матичном научном одбору за машинство и индустријски софтвер захтев број: 01-1/930-18 од 20.03.2025. године, за категоризацију техничког решења под називом **PAKtest: Софтвер за аутоматско тестирање и верификацију промена у коду МКЕ солвера,** чији су аутори: др Марко Топаловић, научни сарадник, др Мирослав Живковић, редовни професор, др Снежана Вуловић, виши научни сарадник, др Александар Николић, научни сарадник и Драгољуб Стевановић, дипл. инж.

Матични научни одбор за машинство и индустријски софтвер, на седници одржаној 22.04.2025. године, разматрао је предметни захтев и једногласно утврдио предлог да техничко решење под називом **PAKtest: Софтвер за аутоматско тестирање и верификацију промена у коду МКЕ солвера,** чији су аутори: др Марко Топаловић, научни сарадник, др Мирослав Живковић, редовни професор, др Снежана Вуловић, виши научни сарадник, др Александар Николић, научни сарадник и Драгољуб Стевановић, дипл. инж., **ИСПУЊАВА** све услове предвиђене *Правилником о стицању истраживачких и научних звања* („Службени гласник РС”, бр. 159/20) за доделу категорије **М85 – Ново техничко решење у фази реализације**.

Председник МНО
за машинство и индустријски софтвер
Проф. др Зоран Миљковић

УНИВЕРЗИТЕТ У КРАГУЈЕВЦУ
Факултет инжењерских наука
Број: 01-1/930-18
20.03.2025. године
Крагујевац

На предлог Катедре за примењену механику и аутоматско управљање (број 01-1/841 од 06.03.2025. године) а на основу чланова 1 и 3. став 5. Правилника о стицању истраживачких и научних звања (Сл. гл. РС бр. 159/2020 и 14/2023) и члана 173 Статута Факултета инжењерских наука у Крагујевцу (бр. 01-1/2700 од 17.08.2023. год. – пречишћен текст), Наставно-научно веће Факултета инжењерских наука у Крагујевцу, на седници одржаној 20.03.2025. године, донело је

# О Д Л У К У

**I** Усваја се пријава техничког решења под насловом: „**PAKtest: Софтвер за аутоматско тестирање и верификацију промена у коду МКЕ солвера**“, чију су аутори: др Марко Топаловић, научни сарадник, др Мирослав Живковић, редовни професор, др Снежана Вуловић, виши научни сарадник, др Александар Николић, научни сарадник и Драгољуб Стевановић, дипл. инж.

**II** Техничко решење се упућује Матичном одбору за машинство и индустријски софтвер.

Одлуку доставити:
- Матичном одбору Министарства
- Ауторима
- Архиви

ДЕКАН ФАКУЛТЕТА ИНЖЕЊЕРСКИХ НАУКА

Republika Srbija - Univerzitet u Kragujevcu - Fakultet inženjerskih nauka 200064465 2025.03.20 14:35:37 +01'00'

Slobodan Savić 200045797 2025.03.20 14:35:05 +01'00'

Др Слободан Савић, редовни професор

ФАКУЛТЕТ ИНЖЕЊЕРСКИХ НАУКА
УНИВЕРЗИТЕТ У КРАГУЈЕВЦУ


# ТЕХНИЧКО РЕШЕЊЕ

M85 Ново техничко решење у фази реализације

# PAKtest: Софтвер за аутоматско тестирање и верификацију промена у коду МКЕ солвера

АУТОРИ

Марко Топаловић
Мирослав Живковић
Снежана Вуловић
Александар Николић
Драгољуб Стевановић

# Подаци о техничком решењу

Врста техничког решења:

M85 – Ново техничко решење у фази реализације

## 1 Аутори техничког решења


- Др Марко Топаловић[1], научни сарадник (topalovic@kg.ac.rs)
- Др Мирослав Живковић[2], редовни професор (zile@kg.ac.rs)
- Др Снежана Вуловић[1], виши научни сарадник (vsneza@kg.ac.rs)
- Др Александар Николић[1], научни сарадник (dziga@kg.ac.rs)
- Драгољуб Стевановић (stevanovic.dragoljub.kg@gmail.com)

[1] Институт за информационе технологије, Универзитет у Крагујевцу
[2] Факултет инжењерских наука, Универзитет у Крагујевцу


## 2 Назив техничког решења

- PAKtest: Софтвер за аутоматско тестирање и верификацију промена у коду МКЕ солвера

## 3 Кључне речи

- Метода коначних елемената, Развој софтвера, Алгоритми и структура података, Тестирање и верификација, FORTRAN, Python, Wndows, GNU-Linux

## 4 Наручилац и корисник техничког решења

- Факултет инжењерских наука, Универзитет у Крагујевцу

## 5 Година када је техничко решење комплетирано

- Развој техничког решења реализован је 2024. године у оквиру пројекта Фонда за науку Републике Србије, Број 7475, “Prediction of damage evolution in engineering structures” – PROMINENT којим руководи Проф. Др. Мирослав Живковић.

## 6 Година када је техничко решење почело да се примењује

- Техничко решење се примењује од 2024. године на Факултету инжењерских наука, Универзитета у Крагујевцу

## 7 Област и научна дисциплина на коју се техничко решење односи

- Рачунска механика и информационе техологије, за које је надлежан *Матични научни одбор за машинство и индустријски софтвер*.

## 8 Опис проблема који се решава техничким решењем

Софтвер за аутоматско тестирање и верификацију промена у коду МКЕ солвера, припада области научно-техничких услуга, пројектовање и развој компјутерског софтвера. Описани софтвер представља иницијални одговор на проблем који је током година постајао све израженији, а који потиче од континуиране примене застарелих технологија (програмског језика) и немогућности

лаког преласка на савременија решења. Развој Програма за Анализу Конструкција-ПАК и чије тестирање је предмет овог техничког решења [1] започет је пре 40 година коришћењем тада популарног програмског језика FORTRAN (FORmula TRANslation). И данас, највећи део кода је испрограмиран по стандарду FORTRAN77. Иако је FORTRAN у погледу решавања система једначина и даље конкурентан, или чак и супериорнији од новијих програмских језика, највећи недостатак који континуирана примена овог застарелелог стандарда носи са собом је чување и приступ подацима и процедурална организација потпрограма. Најновији стандард је FORTRAN 2023 [2], док је Објектно Орјентисано Програмирање (ООП) било доступно већ од верзије 2003 [3]. Основне препреке имплементацији новијих стандарда су веома мали број истраживача/програмера који је ангажован на развоју софтвера, и огромна количина застарелог програмског кода који није могуће постепено преправљати. У програму PAKS највећи проблем је то што се скоро сви подаци чувају у тзв. "радном вектору", меморијском низу у који се ради уштеде меморије подаци пакују јадан за другим, док се адресе "репери" чувају у структурама "COMMON". Ови репери су по функцији јако слични показивачима у језику C++. При променама у коду солвера PAK лако може доћи до тога да се репери, односно променљиве у радном вектору добију нежељену вредност, уколико се ажурирају у различитим потпрограмима (сабрутинама), под одређеним условима за специфичне типове анализа. Такође, у самој структури солвера PAK постоји огромна количина поновљеног кода, на пример за различите типове елемената, за различите материјалне моделе или поступке решавања. Уколико се промена изврши за један случај, на пример ако се изврши нека измена у формату улазног фајла за одређени податак, треба променити све потпрограме у којима се тај податак користи, да не би дошло до грешке.
Примена основних принципа ООП ће знатно осавременити МКЕ солвер PAK, убрзаће и олакшаће његов даљи развој:

Енкапсулација: представља спајање података (атрибута) и метода (функција) које раде на подацима у једну јединицу која се зове објекат. Енкапсулација омогућава скривање података и штити стање објекта од директног модификовања. Ово осигурава да се подацима приступа и да се мењају само преко добро дефинисаних интерфејса, смањујући ризик од непланираних нежељених ефеката и чинећи код робуснијим.

• Апстракција: је процес скривања сложених детаља имплементације и приказивања само битних карактеристика објекта што поједностављује интерфејс, тако да корисник ступа у интеракцију са објектом без потребе да зна како интерно функционише. Примена овог принципа олакшаће уградњу нових материјалних модела.

• Наслеђивање: омогућава једној класи (која се зове поткласа или изведена класа) да наследи својства и методе друге класе (која се назива суперкласа или основна класа). Наслеђивање промовише поновну употребу кода и успоставља природни хијерархијски однос између класа. Наслеђивање ће посебно бити корисно приликом дефинисања типова елемената, и каснијом уградњом нових, изведених елемената на основу постојећих.

• Полиморфизам: Ово омогућава да се методе различитих поткласа третирају као методе заједничке суперкласе, представља могућност редефинисања метода у изведеним класама, омогућавајући једној методи да ради на различите начине на основу објекта на који делује. На пример, метода за рачунање запремине елемента ће имати различите имплементације за тетра и хекса елементе, али ће имати идентичан позив на заједничку виртуелну методу од које је изведена.

Свака промена кода солвера PAK, било да је у складу са FORTRAN77 стандардом, било да представља унапређење стандарда који солвер PAK задовољава, носи са собом ризик да одређене структуре програма не раде на очекиван начин, да доводе до пуцања прорачуна или још горе, до нетачних решења која могу проћи неопажено. Ово техничко решење "Софтвер за аутоматско тестирање и верификацију промена у коду МКЕ солвера", пружа могућност учестале контроле и валидације учињених измена, и бржи развој солвера PAK.

# 9 Стање решености проблема, приказ и анализа постојећих решења

## 9.1 Стање решености проблема код нас

Ауторима није познато да у Србији постоји нека група која се бави развојем и применом сопственог МКЕ солвера и која има развијен софтвер за аутоматско тестирање свог МКЕ солвера.

## 9.2 Стање решености проблема у свету

Верификација нове верзије програма подразумева да софтвер исправно имплементира математичке моделе, док валидација имплицира проверу да ли модели довољно тачно представљају феномене из стварног света. Најпознатије светске софтверске компаније које развијају МКЕ солвере (као што су Altair, Comsol, ADINA, Nastran, ABAQUS, ANSYS и друге) користе ригорозне методологије тестирања како би осигурали тачност и поузданост својих програма, које укључују пре свега стандардне "*benchmark"* примере. Резултати се пореде са познатим аналитичким решењима, решењима из литературе (која су често проистекла из теоријског и експерименталног истраживања, или индустријских стандарда), али се врши и поређење перформанси софтвера са другим комерцијалним софтвером што представља унакрсну валидацију. Основни појмови који карактеришу овакву проверу су:

• Регресивоно тестирање: које се врши да би се осигурало да нова ажурирања или промене не уносе грешке или смањују перформансе. Ово укључује покретање скупа тестова на свакој новој верзији софтвера да би се обезбедила конзистентност, односно компатибилност уназад.

• Аутоматско тестирање: Аутоматизовани оквири за тестирање се користе за ефикасно покретање бројних тест примера. Ово помаже у идентификацији проблема у раној фази развоја.

• Анализа осетљивости: Ово укључује тестирање понашања солвера при малим варијацијама улазних параметара како би се осигурала стабилност и робусност.

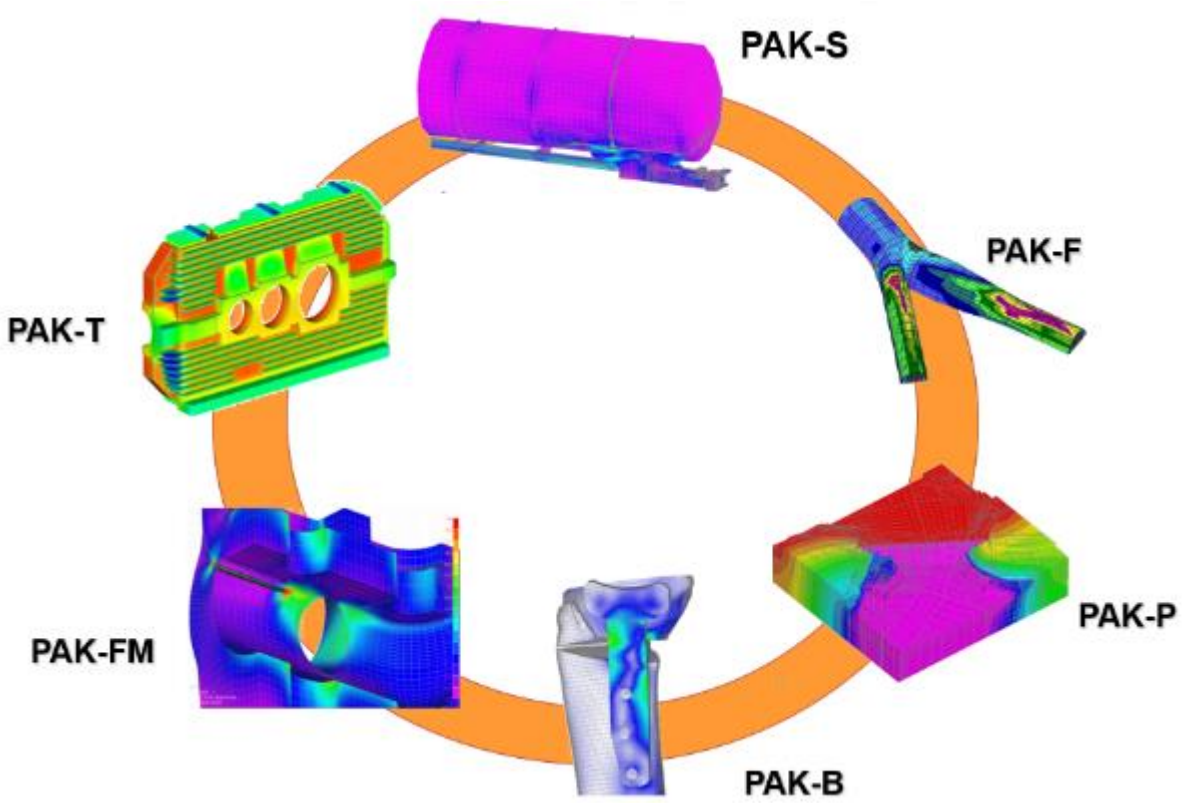
Да би њихов производ остао конкурентан на тржишту, најпознатије светске софтверске компаније које развијају МКЕ солвере примењују принцип континуиране интеграције и континуираног развоја/испоруке најновије верзије софтвера крајњим корисницима [4]. Ове компаније такође користе реалне примере коришћења свог софвера и повратне информације комерцијалних корисника које им помажу да идентификују и реше новонастале проблеме.

Ипак, чак и најпознатије светске компаније понекад објаве нову верзију свог софтвера са изменама које утичу на функционалност програма. Пример са којим се суочавамо при комерцијалној примени програма за моделирање FEMAP, са уграђеним солвером NX NASTRAN, је разлика у решењима при анализи конструкција у којима су дефинисане контактне површине са трењем. Верзија FEMAP-а 11.4 даје стабилнију конвергенцију у односу на новије верзије. Новије верзије, нпр. 2401, имају унапређене алате за генерисање мреже, па због тога новије верзије FEMAP-а користимо за моделирање, а онда модел извозимо у верзију FEMAP-а 11.4 и тамо проверавамо понашање конструкције.

При развоју софтвера за аутоматско тестирање и верификацију PAK МКЕ солвера, примењене су раније дефинисане парадигме које користе и водеће светске софтверске компаније. Иако постоје одређени недостаци, попут броја тест примера који су тренутно убачени у базу, наш софтвер за тестирање такође нуди значајне предности. Једна од њих је FORTRAN имплементација, која ће омогућити каснију интеграцију са солвером, као и Python верзија програма, која не захтева компилацију и инсталацију, а уз то је једноставна за едитовање.

# 10 Детаљан опис техничког решења (укључујући и пратеће илустрације и теоретску суштину)

Као што је у опису проблема истакнуто, континуирани развој Програма за Анализу Конструкција-ПАК траје више од 40 година. Постоји већи број засебних солвера за решавање специфичних проблема као што су структурна анализа PAK-S, провођење топлоте PAK-T, ламинарно струјање флуида PAK-F, итд. Неки од репрезентативних примера примене дати су на Слици 1. Најновији солвер који се тренутно развија PAK-DAM заснивaће се на уградњи теорије фазног поља у оквиру пројекта Фонда за науку Републике Србије, Број 7475, "Prediction of damage evolution in engineering structures" – PROMINENT.

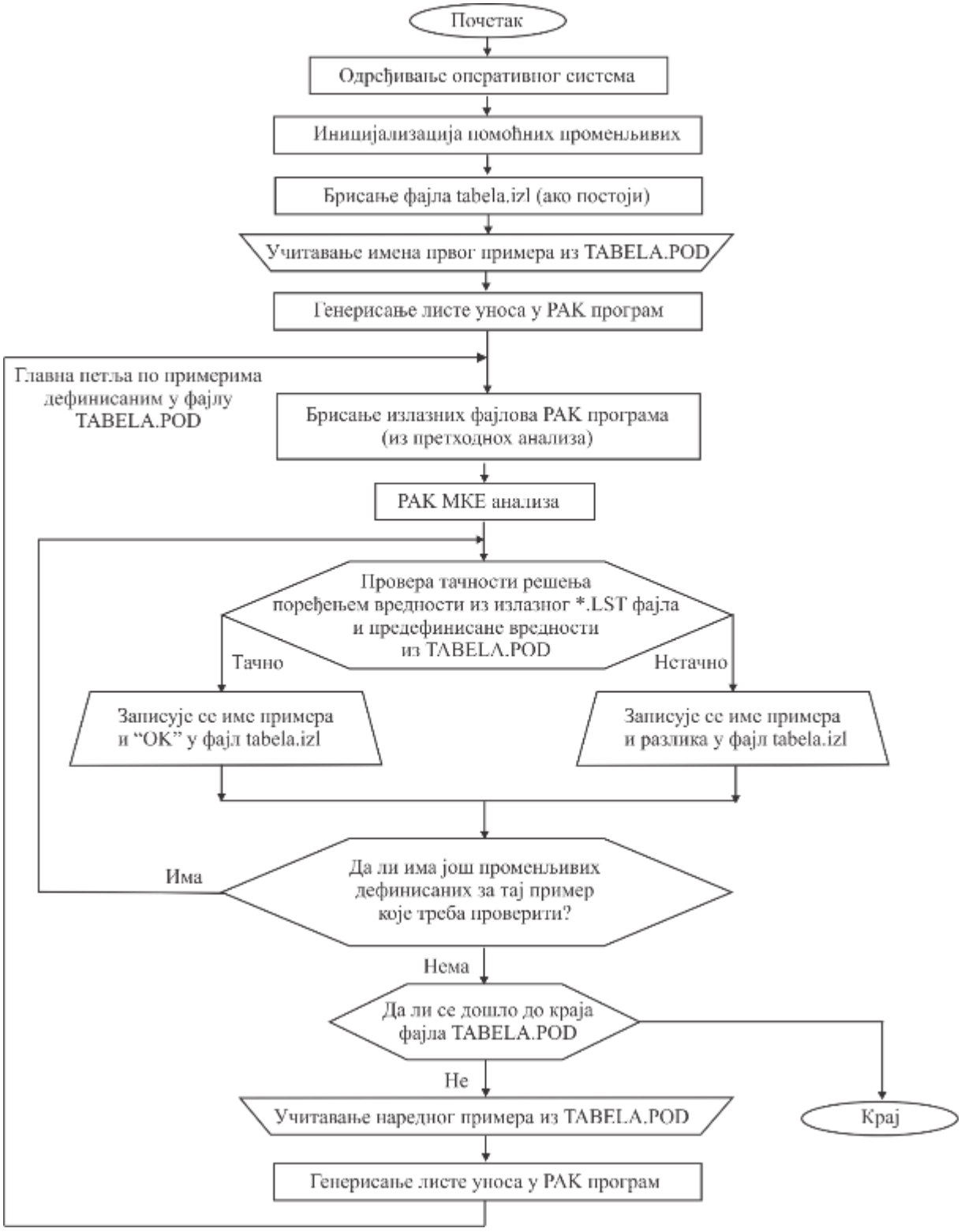



**Слика 1.** МКЕ солвери PAK и њихова различита примена

Да би се процес верификације МКЕ солвера PAK убрзао, развијен је софтвер за аутоматизацију тестирања PAKtest представљен у овом техничком решењу.

## 10.1 Преглед структуре програма за тестирање

Програм за тестирање развијен је паралелно у прграмским језицима FORTRAN и Python да аутоматизује тестирање примера за верфикацију РАК МКЕ солвера. Укључује различите потпрограме за одређивање оперативног система, брисање датотека (ако постоје) и покретање тестова. Програм осигурава да стари резултати теста не ометају извршавање нових тестова тако што брише старе излазне фајлове на крају прорачуна. Имена свих тест примера дефинисана су у датотеци “TABELA.POD“, као и тачне вредности које програм за тестирање пореди са излазним резултатима РАК прорачуна. Улазни фајлови за МКЕ анализу се налазе у директријуму “test” у коме се налази и “TABELA.POD“, док је сам програм за тестирање у директоријуму изнад. Овим се обезбеђује јасна структура која омогућава једноставно аутоматско чишћење директоријума пре и након тестирања. Поред извршне датотеке (“paktest.exe” или “paktest.py”) у основном директоријуму се, након извршења теста, налази датотека “tabela.izl” у којој је сачуван извештај са детаљима о успеху или неуспеху сваког теста. Основна логика се налази унутар подпрограма “testiran”, који се позива за сваки тест пример дефинисан у “TABELA.POD“, док остали потпрограми врше специфичне функције груписане у логичке целине. Оваква орзанизација програма омогућава лако одржавање, надоградњу и идентификацију грешака (уколико се јаве у току извршавања).
Алгоритам програма за тестирање дат је на Слици 2.




**Слика 2.** Алгоритам програма за тестирање МКЕ солвера

## 10.2 Логика извршавања програма. Преглед структуре програма за тестирање

• Одређивање оперативног система:
Потпрограм “detect_os” се позива да утврди да ли програм ради на Windows или GNU-Linux систему како би се прилагодиле путање датотека и команде у складу са тим. У Windows-у, путања садржи обрнуте косе црте а наредба “del” се користи за брисање. На GNU-Linux-у, путање имају косе црте, а за брисање се користи команда “rm”. Ово утиче на потпрограм “execute_command_line” који се користи за извршавање PAK програма и у коме се мењају обрнуте косе црте са косим цртама у путањама датотека (само за GNU-Linux системе). За овај задатак користимо потпрограм “replace_backslash”.

• Иницијализација програма
Следећи корак је декларисање променљивих и иницијализација почетних вредности. Ово укључује целобројне променљиве као што су “status”, “file_unit” и “current_test” за праћење тренутног стања процеса тестирања. Логичке варијабле се користе за проверу да ли датотеке постоје и да ли је тестирање завршено. Низ карактера “command” се користи за чување команде за извршавање ПАК програма.

• Брисање резултата претходног испитивања:
Пре покретања испитивања нове верзије солвера PAK и након сваког теста у тренутном испитивању, привремене датотеке морају бити избрисане. Ово укључује различите датотеке са префиксима “Z”, “fort.” и суфиксима “.UNV”, “.CSV”. Потпрограм “delete_if_exists” се позива више пута да обрише одређене датотеке (ако постоје). Потпрограми “delete_files_prefix” и “delete_files_sufix” се позивају да избришу опште излазне датотеке. Ово осигурава да преостали резултати теста из претходних испитивања не ометају нове тестове. Датотека “tabela.izl” остаје на крају испитивања са свим резултатима, али се аутоматски брише приликом почетка новог испитивања.

• Извршење теста, петља док се тестирање не заврши:
Програм улази у петљу која извршава све тестове користећи потпрограм “testiran”. Унутар петље, “testiran” потпрограм чита датотеку “TABELA.POD” да би одредио које тестове треба покренути. За сваки тест, он уписује име теста у датотеку под називом “aa” и позива PAK солвер користећи одговарајућу команду за одређени оперативни систем. Након што PAK заврши анализу, резултати се проверавају и пореде са очекиваним вредностима дефинисаним у датотеци “TABELA.POD”. У зависности од тога да ли се резултати поклапају или не, програм бележи или “OK”, или разлику у датотеку “tabela.izl”. Након ове провере, одређене датотеке се бришу, бројач теста “current_test” се повећава и чита се име следећег примера.

• Чишћење: Након завршетка испитивања, програм брише привремене датотеке као што су “pak.lst”, “pak.unv” и “pak.neu”, као и све друге помоћне датотеке креиране током процеса тестирања. Ово осигурава да је директоријум чист за будуће испитивање.

• Управљање грешкама: У програм је имплементирана свеобухватна провера грешака током целог процеса испитивања. Програм пружа тачне информације кориснику када се појаве грешке, а од корисника се очекује да унесе неки број на тастарури уколико жели да настави тестирање следећег примера. Излазни кодови указују на различите врсте грешака као што су:

- Проблеми са приступом датотекама
- Неуспело извршење солвера PAK
- Проблеми са верификацијом теста

## 10.3 Детаљан опис потпрограма

### *10.3.1 Потпрограм: testiran*

Потпрограм “testiran” садржи логику за читање “TABELA.POD” датотеке, одређивање који је следећи пример који се анализира, и проверу резултата. Излазна датотека након PAK прорачуна “pak.lst” садржи све битне резултате МКЕ анализе, али се проверавају само они који су дефинисани у “TABELA.POD”. У табели 1 дат је комплетан FORTRAN код потпрограма “testiran”, док је у табели 2 дата његов Python еквивалент.

Табела 1: FORTRAN потпрограм testiran

```
    subroutine testiran(status, testing_complete, current_test,is_windows)

    IMPLICIT NONE

    !Variable declarations
    CHARACTER(LEN=256) :: line, valueOfLine, search_str, compare_str, lst_str, lst_result, s2
    CHARACTER(LEN=1) :: char, emptySpace
    character(len=100) :: command, test_example,test_example_dot
    INTEGER :: i, numberOfLInes, current_hash_count, ios, test, red, startChar, endChar
    INTEGER, intent(in) :: current_test  ! Added current_test as input parameter
    logical, intent(in) :: is_windows
    LOGICAL :: file_exists
    LOGICAL, intent(inout) :: testing_complete

    ! File units
    INTEGER, PARAMETER :: UNIT_AA = 10
    INTEGER, PARAMETER :: UNIT_IZL = 11
    INTEGER, PARAMETER :: UNIT_POD = 12
    INTEGER, PARAMETER :: UNIT_POM = 13
    INTEGER, PARAMETER :: UNIT_DAT = 14
    INTEGER, PARAMETER :: UNIT_LST = 15
    integer, intent(out) :: status

    status = 0

    lst_str = ''

    INQUIRE(FILE='test/TABELA.POD', EXIST=file_exists)
    IF (.NOT. file_exists) THEN
       status = 1
       RETURN
    END IF

    OPEN(UNIT=UNIT_POD, FILE='test/TABELA.POD', STATUS='OLD', IOSTAT=ios)
    IF (ios /= 0) THEN
       status = 2
       RETURN
    END IF

    red = 0
    current_hash_count = 0
    DO
       numberOfLInes = 0
       READ(UNIT_POD, '(A)', IOSTAT=ios) line
       IF (ios /= 0) THEN
          testing_complete = .true.
          CLOSE(UNIT_POD)
          EXIT
       END IF

       IF (line(1:1) == '#') THEN
          current_hash_count = current_hash_count + 1

          IF (current_hash_count == current_test) THEN
             PRINT *, 'proverava fajl ', TRIM(line(2:))

             OPEN(UNIT=UNIT_IZL, FILE='tabela.izl', POSITION='APPEND')
             WRITE(UNIT_IZL, '(A)') TRIM(line(2:))
             CLOSE(UNIT_IZL)
```

```
      OPEN(UNIT=UNIT_AA, FILE='aa', STATUS='REPLACE')
         test_example = TRIM(line(2:)) ! Windows version
      if (is_windows.EQ.(.false.)) then
         call replace_backslash(test_example) ! Linux/Unix version
         test_example_dot = trim(test_example)
         test_example = test_example_dot(:len(trim(test_example_dot)) -1)
      endif
      WRITE(UNIT_AA, '(A)') test_example
      WRITE(UNIT_AA, '(A)') 'pak.lst'
      WRITE(UNIT_AA, '(A)') 'pak.unv'
      WRITE(UNIT_AA, '(A)') 'pak.neu'
      CLOSE(UNIT_AA)

      red = 1
      CLOSE(UNIT_POD)
      EXIT
    END IF
  ELSE
    IF (current_hash_count == current_test-1) THEN
      INQUIRE(FILE='pak.lst', EXIST=file_exists)
      IF (.NOT. file_exists) THEN
         CLOSE(UNIT_POD)
         status = 3
         RETURN
      END IF

      OPEN(UNIT=UNIT_LST, FILE='pak.lst', STATUS='OLD', IOSTAT=ios)
      IF (ios /= 0) THEN
         CLOSE(UNIT_POD)
         status = 4
         RETURN
      END IF

      IF ((line(2:2) == 't').OR.(line(2:2) == 'T')) THEN
         READ(UNIT_POD, '(A1)', ADVANCE='NO') line
         READ(UNIT_POD, '(A1)', ADVANCE='NO') line
         REWIND(UNIT_LST)
         red = 1
      END IF

      IF ((line(2:2) == 'd').OR.(line(2:2) == 'D')) THEN
         OPEN(UNIT=UNIT_IZL, FILE='tabela.izl', POSITION='APPEND')
         valueOfLine = TRIM(line(6:))
         READ(valueOfLine, '(I)', IOSTAT=ios) numberOfLInes
         IF (ios == 0) THEN
            DO i = 1, numberOfLInes
               READ(UNIT_LST, *, IOSTAT=ios)
               IF (ios /= 0) EXIT
            END DO
            red = red + numberOfLInes
         END IF
         CLOSE(UNIT_IZL)
      END IF

      IF ((line(2:2) == 'c').OR.(line(2:2) == 'C')) THEN
         OPEN(UNIT=UNIT_IZL, FILE='tabela.izl', POSITION='APPEND')
         valueOfLine = TRIM(line(6:))
         compare_str = valueOfLine(3:LEN_TRIM(valueOfLine)-1)
         IF (lst_result == compare_str) THEN
            WRITE(UNIT_IZL, '(A)') 'OK'
         ELSE
            WRITE(UNIT_IZL, '(A)') TRIM(lst_result) // '->' // TRIM(compare_str)
         END IF
         CLOSE(UNIT_IZL)
      END IF

      IF ((line(2:2) == 'r').OR.(line(2:2) == 'R')) THEN
         OPEN(UNIT=UNIT_IZL, FILE='tabela.izl', POSITION='APPEND')
         valueOfLine = TRIM(line(6:))
         READ(valueOfLine, *, IOSTAT=ios) startChar, endChar
         IF (ios == 0) THEN
            READ(UNIT_LST, '(A)', IOSTAT=ios) lst_str
            IF (ios == 0) THEN
               lst_result = lst_str(startChar:endChar)
            END IF
```

```
            END IF
            REWIND(UNIT_LST)
            DO i = 1, red
               READ(UNIT_LST, *, IOSTAT=ios)
               IF (ios /= 0) EXIT
            END DO
            CLOSE(UNIT_IZL)
         END IF

         IF ((line(2:2) == 'f').OR.(line(2:2) == 'F')) THEN
            OPEN(UNIT=UNIT_IZL, FILE='tabela.izl', POSITION='APPEND')
            valueOfLine = TRIM(line(6:))
            search_str = valueOfLine(3:LEN_TRIM(valueOfLine)-1)
            red = red - 1
            DO
               READ(UNIT_LST, '(A)', IOSTAT=ios) s2
               IF (ios /= 0) EXIT
               red = red + 1
               IF (INDEX(s2, TRIM(search_str)) > 0) EXIT
            END DO

            REWIND(UNIT_LST)
            DO i = 1, red
               READ(UNIT_LST, *, IOSTAT=ios)
               IF (ios /= 0) EXIT
            END DO
            CLOSE(UNIT_IZL)
         END IF
      END IF
   END IF
END DO

end subroutine testiran
```

Табела 2: Python потпрограм testiran

```
def testiran(self):
    """Main testing function that processes TABELA.POD and compares results"""
    pod_file_path = self.base_dir / 'test' / 'TABELA.POD'

    try:
        with open(pod_file_path, 'r') as pod_file, \
             open(self.base_dir / 'tabela.izl', 'a') as izl_file:
            current_hash_count = 0
            red = 0
            lst_lines = []

            for line in pod_file:
                line = line.strip()
                if not line:
                    continue

                if line.startswith('#'):
                    current_hash_count += 1

                    if current_hash_count == self.current_test:
                        print(f'Checking file {line[1:].strip()}')

                        # Append to tabela.izl
                        izl_file.write(f'{line[1:].strip()}\n')

                        # Create aa file
                        test_example = line[1:].strip()
                        if not self.is_windows:
                            test_example = test_example.replace('\\', '/')
                            if test_example.endswith('.'):
                                test_example = test_example[:-1]

                        with open(self.base_dir / 'aa', 'w') as aa_file:
                            aa_file.write(f'{test_example}\n')
                            aa_file.write('pak.lst\n')
                            aa_file.write('pak.unv\n')
                            aa_file.write('pak.neu\n')

                        return 0, False  # Continue testing
```

```
            else:
                if current_hash_count == self.current_test - 1:
                    # Check if pak.lst exists
                    pak_lst_path = self.base_dir / 'pak.lst'

                    # Wait for pak.lst to be created and fully written
                    if not self.wait_for_file(pak_lst_path):
                        print("Error: pak.lst was not created or is incomplete")
                        return 1
                    #if not pak_lst_path.exists():
                    #    print("pak.lst not found")
                    #    return 3, False

                    # Read lst file lines
                    with open(pak_lst_path, 'r') as lst_file:
                        lst_lines = lst_file.readlines()
                    # Process different command types
                    command_type = line[0:1].lower()
                    if command_type == 't':
                        # Text or initial processing
                        red = 1
                    elif command_type == 'f':
                        # Find line containing a specific string
                        try:
                            search_str = line[5:].strip()[1:-1]
                            red -= 1

                            # Search for line containing the string
                            for i in range(red, len(lst_lines)):
                                red += 1
                                if search_str in lst_lines[i]:
                                    break
                        except IndexError:
                            print(f"Error processing 'f' command: {line}")

                    elif command_type == 'd':
                        # Delete or skip specified number of lines
                        try:
                            num_lines = int(line[5:].strip())
                            # Skip lines and update red
                            red += num_lines
                        except (ValueError, IndexError):
                            print(f"Error processing 'd' command: {line}")

                    elif command_type == 'r':
                        # Read specific characters from a line
                        try:
                            # Parse start and end characters
                            start_char, end_char = map(int, line[5:].strip().split())
                            # Check if there are enough lines
                            if red < len(lst_lines):
                                # Extract substring from specified line and range
                                self.lst_result = lst_lines[red-1][start_char-1:end_char]
                        except (ValueError, IndexError):
                            print(f"Error processing 'r' command: {line}")

                    elif command_type == 'c':
                        # Comparison of last result with expected value
                        try:
                            # Remove quotes and strip whitespace
                            compare_str = line[5:].strip()[1:-1].strip()
                            current_result = self.lst_result.strip()

                            # Try to parse as floating-point numbers for scientific notation comparison
                            try:
                                # Normalize scientific notation
                                compare_float = float(compare_str.replace('D', 'E'))
                                result_float = float(current_result.replace('D', 'E'))

                                # Use approximate comparison with a small tolerance
                                if abs(compare_float - result_float) < 1e-10:
                                    izl_file.write('OK\n')
                                else:
                                    izl_file.write(f'{current_result}->{compare_str}\n')

                            except ValueError:
```

```
                        # Fallback to string comparison if not numeric
                        if current_result == compare_str:
                            izl_file.write('OK\n')
                        else:
                            izl_file.write(f'{current_result}->{compare_str}\n')

                except IndexError:
                    print(f"Error processing 'c' command: {line}")


    except FileNotFoundError:
        print(f"Error: TABELA.POD not found at {pod_file_path}")
        return 2, True
    except Exception as e:
        print(f"Error in testiran: {e}")
        return 2, True

    return 0, True
```

Функционалност потпрограма биће објашњена на основу FORTRAN верзије јер је програмски језик FORTRAN лакши за разумевање.

Декларација променљивих: Потпрограм почиње декларисањем променљивих укључујући знаковне низове, целобројене и логичке променљиве. Такође се декларишу улазни параметри за тренутни број теста и тип оперативног система.

Провера постојања датотеке: Потпрограм проверава да ли "TABELA.POD" постоји коришћењем INQUIRE наредбе. Ако датотека не постоји, потпрограм се враћа у главни програм са кодом грешке и прелази на следећи пример.

Читање инструкција за тестирање: Потпрограм отвара "TABELA.POD" датотеку и чита упутства ред по ред. Прати број линија и тренутни ID број тест примера. Сваки тест пример је означен симболом „#", након чега следи име примера. Садржај датотеке "TABELA.POD" ће бити касније детаљније обрађен.

Припрема извршавања тестова: Када наиђе на симбол "#", потпрограм уписује име тест примера у помоћну „aa" датотеку која садржи неопходне команде и имена датотека за PAK програм, а потом се враћа у главни програм. Главни програм за тестирање покреће PAK МКЕ анализу користећи потпрограм "execute_command_line". Програм чека да се PAK МКЕ анализа заврши и проверава да ли је дошло до било каквих грешака током извршавања. Након успешног завршетка PAK прорачуна, главни програм позива поново потпрограм "testiran" који на основу променљиве "current_hash_count" проналази тренутни пример до кога је стао у "TABELA.POD" и упоређује излаз (сачуван у "pak.lst") са унапред дефинисаним вредностима из "TABELA.POD". Упоређивање се врши на основу инструкција наведених у "TABELA.POD", које обухватају

- Директно поређење стрингова
- Издвајање опсега карактера
- Бројање и прескакање линија
- Претрага стрингова унутар датотеке.

У зависности од резултата поређења, резултати се бележе у "tabela.izl" као:

- "OK" за успешна поклапања
- У формату "Actual->Expected" за непоклапања.

### *10.3.2 Потпрограм: delete_if_exists*

Као што се може закључити из самог имена потпрограма, "delete_if_exists" брише датотеку чије се име прослеђује као параметар потпрограму. Да би се избегле могуће грешке, потпрограм прво проверава да ли датотека постоји. Ова се провера врши позивањем функције "inquire" у FORTRAN-у односно ".exists()" у Python-у. Садржај верзије FORTRAN потпрограма дат је у табели 3, док је Python еквивалент дат у табели 4.

Табела 3: FORTRAN потпрограм delete_if_exists

```
subroutine delete_if_exists(filename, UNIT_DEL)
   character(len=*), intent(in) :: filename
   INTEGER, intent(in) :: UNIT_DEL
   logical :: exists
   integer :: ios
   inquire(file=filename, exist=exists)
   if (exists) then
      open(UNIT=UNIT_DEL, file=filename, status='old')!, iostat=ios)
      !if (ios == 0) then
         close(UNIT_DEL, status='delete', iostat=ios)
      !end if
   end if
end subroutine delete_if_exists
```

Табела 4: Python потпрограм delete_if_exists

```
def delete_if_exists(self, filename):
   """Delete file if it exists, with full path handling"""
   file_path = self.base_dir / filename
   try:
      if file_path.exists():
         file_path.unlink()
   except Exception as e:
      print(f"Error deleting {filename}: {e}")
```

### *10.3.3 Потпрограм: delete_files_prefix*

Потпрограм "delete_files_prefix" брише све датотеке чије име садржи дати префикс на почетку. За ово се користе системске функције "del" за Windows и "rm" за GNU-Linux. Садржај верзије FORTRAN потпрограма дат је у табели 5, док је Python еквивалент дат у табели 6.

Табела 5: FORTRAN потпрограм delete_files_prefix

```
subroutine delete_files_prefix(prefix,is_windows)
character(len=*), intent(in) :: prefix
logical, intent(in) :: is_windows
integer :: status
character(len=100) :: command
call detect_os(is_windows)
if (is_windows) then      ! Windows version
   command = 'del ' // trim(prefix) // '* /Q'
else       ! Linux/Unix version
   command = 'rm -f ' // trim(prefix) // '*'
endif
call execute_command_line(command, wait=.true., exitstat=status)
end subroutine delete_files_prefix
```

Табела 6: Python потпрограм delete_files_prefix

```
def delete_files_prefix(self, prefix):

   is_windows = platform.system().lower() == 'windows'

   try:
      # Construct delete command based on OS
      if is_windows:
         command = f'del {prefix}* /Q'
      else:
         command = f'rm -f {prefix}*'

      # Execute the command
      result = subprocess.run(
         command,
         shell=True,
         stdout=subprocess.PIPE,
         stderr=subprocess.PIPE,
         text=True
      )

      # Return exit status (0 for success)
      return result.returncode
```

```
    except Exception as e:
        print(f"Error deleting files with prefix {prefix}: {e}")
        return 1
```

### *10.3.4 Потпрограм: delete_files_sufix*

Слично претходном, "delete_files_sufix" потпрограм брише све фајлове који имају дату екстензију. Садржај верзије FORTRAN потпрограма дат је у табели 7, док је Python еквивалент дат у табели 8.

Табела 7: FORTRAN потпрограм delete_files_sufix

```
subroutine delete_files_sufix(sufix,is_windows)
character(len=*), intent(in) :: sufix
logical, intent(in) :: is_windows
integer :: status
character(len=100) :: command
call detect_os(is_windows)
if (is_windows) then      ! Windows version
   command = 'del *' // trim(sufix) // ' /Q'
else       ! Linux/Unix version
   command = 'rm -f *' // trim(sufix)
endif
call execute_command_line(command, wait=.true., exitstat=status)
end subroutine delete_files_sufix
```

Табела 8: Python потпрограм delete_files_sufix

```
def delete_files_suffix(self, suffix):

    is_windows = platform.system().lower() == 'windows'

    try:
        # Construct the delete command based on OS
        if is_windows:
            command = f'del /F /Q *{suffix}'
        else:
            # Use `shlex` to properly handle wildcard expansion
            command = f'rm -f -- *{suffix}'

        # Execute the command in a way that expands wildcards on Linux
        result = subprocess.run(
            command,
            shell=True,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True,
            executable="/bin/bash" if not is_windows else None  # Use bash for wildcard expansion
        )

        if result.returncode != 0:
            print(f"Error: {result.stderr}")

        return result.returncode

    except Exception as e:
        print(f"Error deleting files with suffix {suffix}: {e}")
        return 1
```

### *10.3.5 Потпрограм: detect_os*

Потпрограм "detect_os" се користи да утврди да ли се извршава на Windows или GNU-Linux систему. Ово је кључно за правилно руковање путањама до датотека и извршавање системских наредби. Постоји само FORTRAN верзија овог потпрограма дата у табели 9, док се у Python-у оперативни систем одређује помоћу једне линије кода дате у табели 10.

Табела 9: FORTRAN потпрограм detect_os

```
subroutine detect_os(is_windows)
logical, intent(out) :: is_windows
character(256) :: os_name, windir
integer :: status
! Try Windows-specific environment variable
call get_environment_variable('WINDIR', windir, status=status)
if (status == 0) then
   is_windows = .true.
   write(*,*) 'Detected Windows'
   return
end if
! If we get here, assume it's Unix/Linux
is_windows = .false.
end subroutine detect_os
```

Табела 10: Python одређивање оперативног система

```
is_windows = platform.system().lower() == 'windows'
```

### *10.3.6 Потпрограм: replace_backslash*

На Windows-у, у путањама до датотека користе се обрнуте косе црте (“\”), док се на GNU-Linux системима користе косе црте (“/”). Постоји само FORTRAN верзија овог потпрограма дата у табели 11, док се у Python-у замена врши помоћу једне линије кода дате у табели 12.

Табела 11: FORTRAN потпрограм replace_backslash

```
subroutine replace_backslash(str)
implicit none
character(len=*), intent(inout) :: str
integer :: i

do i = 1, len(trim(str))
   if (str(i:i) == '\') then
      str(i:i) = '/'
   endif
end do
end subroutine replace_backslash
```

Табела 12: Python замена косих црта

```
if not self.is_windows:
                    test_example = test_example.replace('\\', '/')
```

## 10.4 Детаљан опис fajla TABELA.POD

Датотека “TABELA.POD” служи као контролна датотека за процес тестирања PAK-a. Омогућава лаке измене без промене изворног кода и садржи уносе за сваки тест случај, структуиране на следећи начин:
Линије које почињу са “#” означавају почетак новог тест случаја и садрже назив датотеке тест примера.
Наредне линије након имена тест примера дефинишу специјалне команде за читање резултата из *.lst датотеке:

- “top”: Поновно постављање позиције читања на врх излазне датотеке
- “down”: Прескакање одређеног броја линија
- “read”: Читање одређеног опсега карактера
- “find”: Претрага одређене ниске карактера
- “comp”: Поређење излаза са очекиваном предефинисаном вредношћу

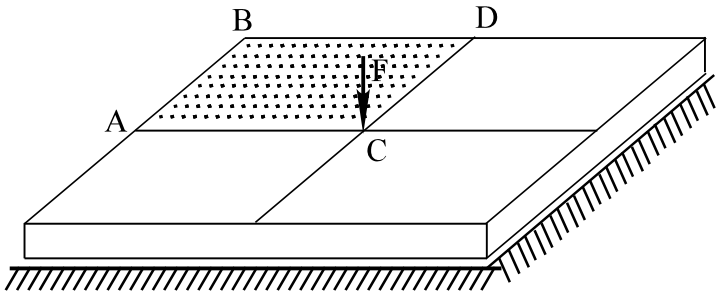
Мора се водити рачуна да су горе побројане команде увучене за једно празно “tab” место, додавање нових команди у слободном формату би изазвало грешку приликом читања. Садржај датотеке “TABELA.POD” за почетна два примера дат је у табели 13.

Табела 13: Почетни садржај датотеке “TABELA.POD”

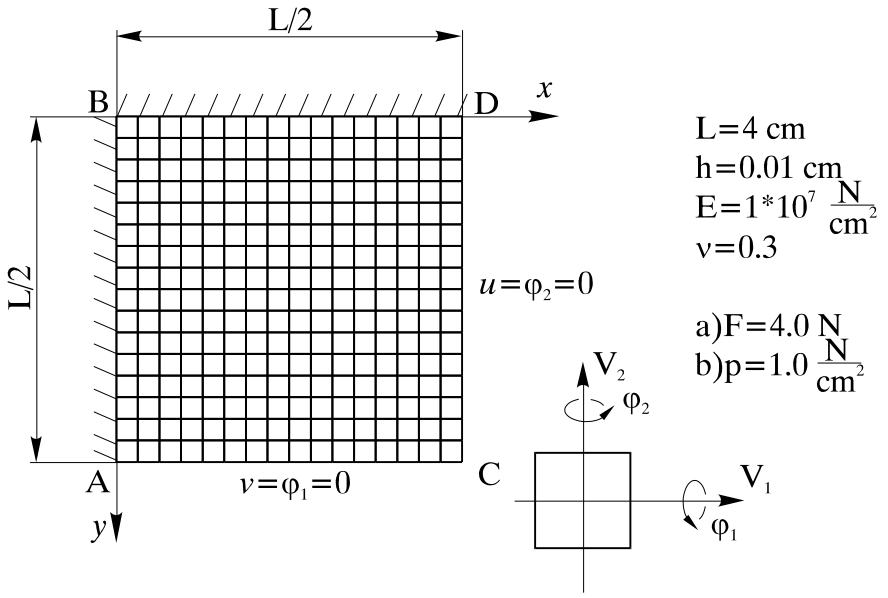
```
#test\SE8_1A2.
        find "S T E"
        down  1
        find "S T E"
        down 10
        read 38 47
        comp "3.3866E-01"
#test\SE8_1B2.
        find "S T E"
        down  1
        find "S T E"
        down 10
        read 37 47
        comp "-3.3864E-01"
```

## 10.5 Валидација методологије на примерима једноставне квадратне плоче оптерећене концентрисаном силом (пример SE8_1A2) и равномерно распоређеним притиском (пример SE8_1B2)

Ова два примера су први примери који се анализирају на основу листе у датотеци "TABELA.POD". У примерима је анализирана једноставна квадратна плоча оптерећена концентрисаном силом (пример SE8_1A2) и равномерно распоређеним притиском (пример SE8_1B2). Услед услова симетрије моделирана је ¼ плоче. Шема оптерећења, гранични услови, димензије и материјалене карактеристике плоче дати су на Слици 3.



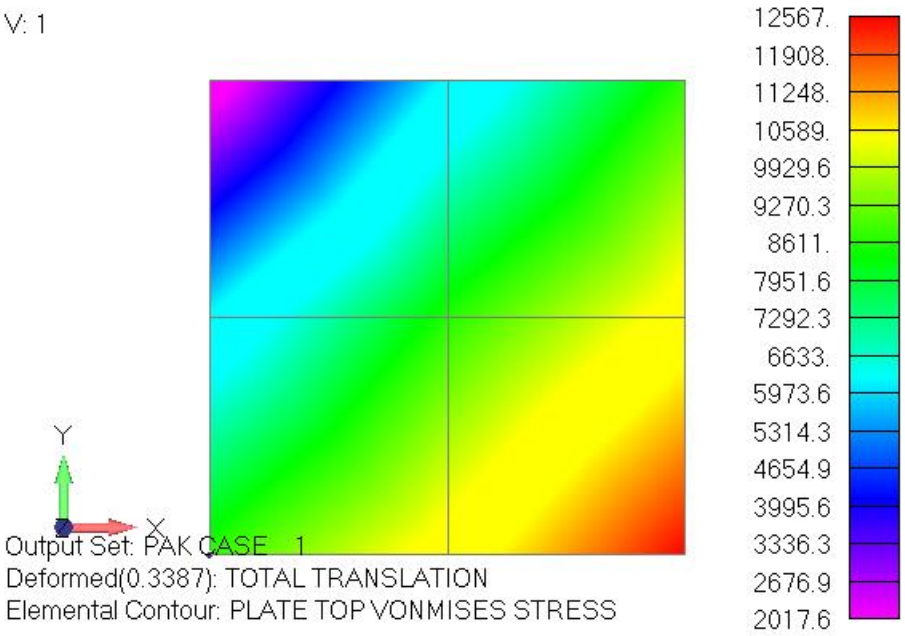

а)



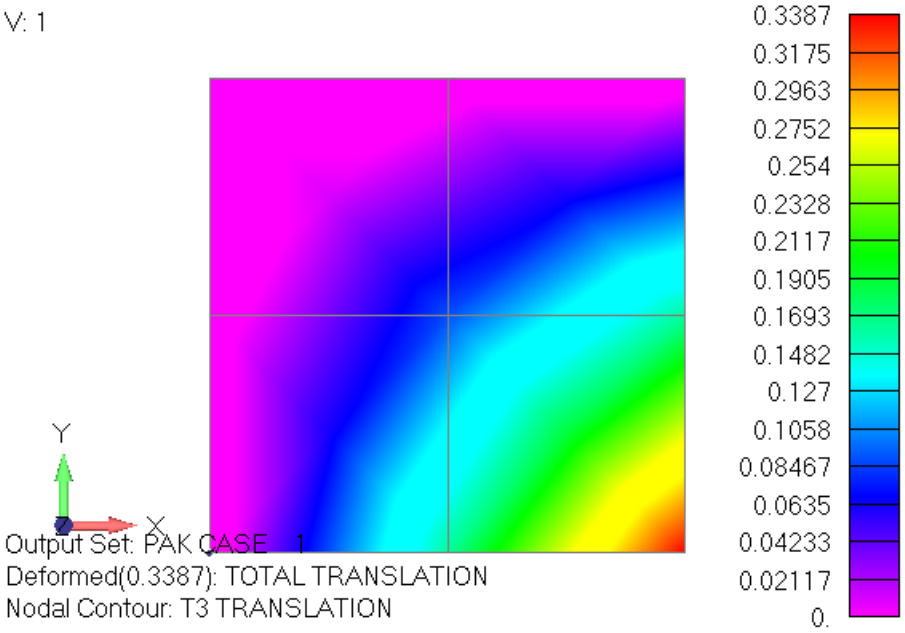

б)

**Слика 3.** Верификациони пример квадратне плоче: а) Моделирани део структуре са оптерећењима б) Шематски приказ МКЕ модела са елементима љуске, гранични услови, димензије, материјалне карактеристике

Примери SE8_1A2 и SE8_1B2 садрже 4 елемента у моделу, док су неки од каснијих примера дефинисани са гушћом мрежом, различитим оптерећењима, ограничењима, типовима елемената, материјалима и другим параметрима који могу утицати на тачност резултата.
Резултати прорачуна за пример SE8_1A2, приказани у програму за пре- и постпроцесирање FEMAP, налазе се на Сликама 4 и 5. Слика 4 приказује поље Von Mises-ових напона, док је на Слици 5 представљено померање у вертикалном правцу (Z оса).

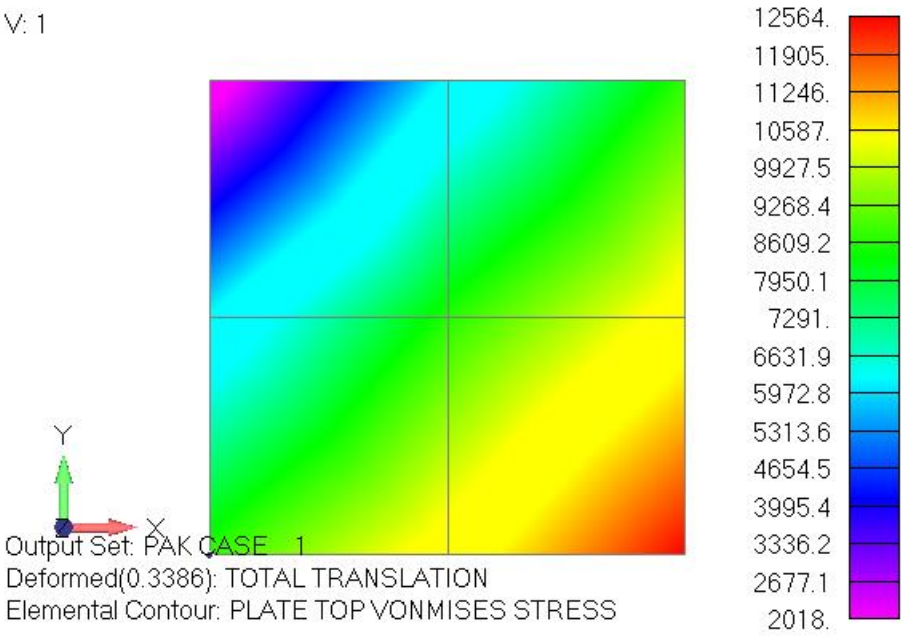



**Слика 4.** Пример SE8_1A2: поље Von Mises-ових напона

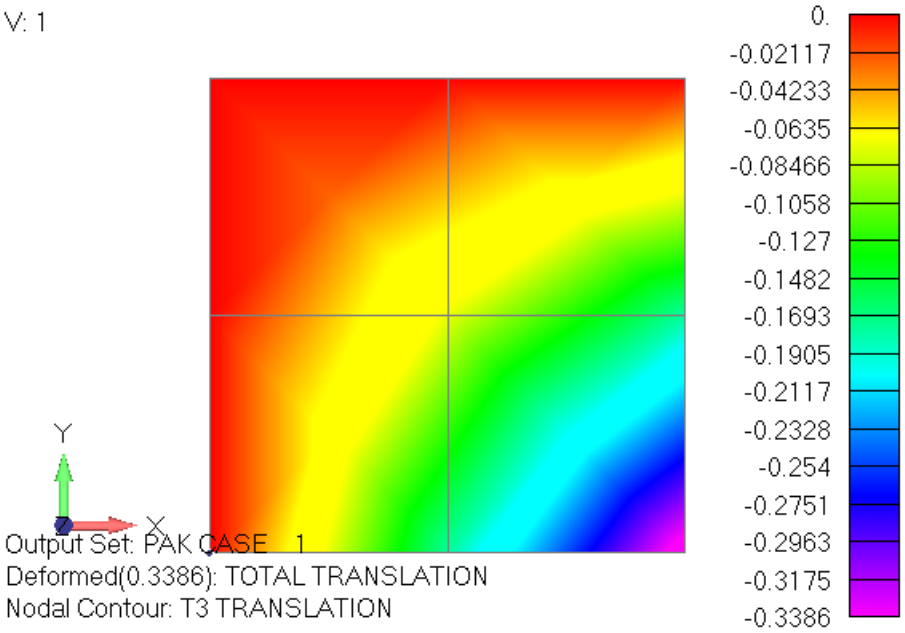



**Слика 5.** Пример SE8_1A2: поље померања у правцу Z осе

Резултати прорачуна за пример SE8_1B2 приказани у програму за пре и пост-процесирање FEMAP, дати су на Сликама 6 и 7. На Слици 6 приказано је поље Von Mises-ових напона, док је на Слици 7 дато померање у вертикалном правцу (Z оса).




**Слика 6.** Пример SE8_1B2: поље Von Mises-ових напона




**Слика 7.** Пример SE8_1B2: поље померања у правцу Z осе

Резултати прорачуна у програму PAK чувају се у више различитих формата, *.neu фајлови служе за учитавање у FEMAP док је *.lst основни листинг фајл за PAK који садржи све информације о анализираном моделу као и прегледно и концизно дате резултате анализе. На сликама 8 и 9 дати су делови *.lst фајла у којима су записана померања. Ове вредности проналази програм за тестирање на основу података датих у фајлу “TABELA.POD” и проверава да ли су тачни.

SE8_1A2.LST

```
1084    COMPONENTS  OF  DISPLACEMENTS  AND  ROTATIONS
1085    -------------------------------------------
1086
1087
1088    S T E P  N U M B E R =    1                    T I M E = 1.00000E+00
1089
1090
1091     NODE     X           Y           Z           XT          YT          ZT
1092           1  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1093           2  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1094           3  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1095           4  0.0000E+00  0.0000E+00  1.6932E-01  0.0000E+00 -3.3861E-01  0.0000E+00
1096           5  0.0000E+00  0.0000E+00  8.4662E-02 -1.6930E-01 -1.6930E-01  0.0000E+00
1097           6  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1098           7  0.0000E+00  0.0000E+00  3.3866E-01  0.0000E+00  0.0000E+00  0.0000E+00
1099           8  0.0000E+00  0.0000E+00  1.6932E-01 -3.3861E-01  0.0000E+00  0.0000E+00
1100           9  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1101
1102
1103    MAXIMUM DISPLACEMENT:
1104    NODE =     7   MAX DISPL  =  3.3866E-01
```

Слика 8. Резултат прорачуна примера SE8_1A2 у *.lst формату, померање чвора које се контролише

SE8_1B2.LST

```
1136    COMPONENTS  OF  DISPLACEMENTS  AND  ROTATIONS
1137    -------------------------------------------
1138
1139
1140    S T E P  N U M B E R =    1                    T I M E = 1.00000E+00
1141
1142
1143     NODE     X           Y           Z           XT          YT          ZT
1144           1  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1145           2  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1146           3  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1147           4  0.0000E+00  0.0000E+00 -1.6934E-01  0.0000E+00  3.3859E-01  0.0000E+00
1148           5  0.0000E+00  0.0000E+00 -8.4677E-02  1.6932E-01  1.6932E-01  0.0000E+00
1149           6  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1150           7  0.0000E+00  0.0000E+00 -3.3864E-01  0.0000E+00  0.0000E+00  0.0000E+00
1151           8  0.0000E+00  0.0000E+00 -1.6934E-01  3.3859E-01  0.0000E+00  0.0000E+00
1152           9  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
1153
1154
1155    MAXIMUM DISPLACEMENT:
1156    NODE =     7   MAX DISPL  =  3.3864E-01
```

Слика 9. Резултат прорачуна примера SE8_1B2 у *.lst формату, померање чвора које се контролише

За пример SE8_1A2, на основу низа наредби дефинисаних у фајлу “TABELA.POD” датим у Табели 13, програм за тестирање проналази прво појављивање низа карактера "S T E" у *.lst фајлу, прескаче га (јер је након њега дефинисан број корака МКЕ анализе који не контролишемо), затим проналази друго појављивање низа карактера "S T E" у *.lst фајлу, и долази до десетог реда који се налази испод низа карактера "S T E":

7 0.0000E+00 0.0000E+00 3.3866E-01 0.0000E+00 0.0000E+00 0.0000E+00

Програм за тестирање онда чита вредност између позиција 38 и 47 у овом реду (такође дефиниисано у “TABELA.POD”) и пореди је са "3.3866E-01", што је тачно и очекивано решење. Слична процедура је и за следећи пример SE8_1B2, где се у пронађеном реду:

7 0.0000E+00 0.0000E+00 -3.3864E-01 0.0000E+00 0.0000E+00 0.0000E+00

проверава позиција: 37-47 која треба да буде "-3.3864E-01", што и јесте случај, тако да и овај пример ради тачно.

# 11 Закључак

Софтвер за аутоматско тестирање и верификацију промена у коду МКЕ солвера представљен у овом техничком решењу пружа свеобухватно и ефикасно решење за аутоматизовану валидацију нових верзија РАК софтвера. Дизајниран да олакша обимно извршавање тестова, он обезбеђује тачну процену резултата за сваки тест, и води детаљан запис/дневник (log) о исходима тестова, омогућавајући темељну анализу и документацију резултата. Основне карактеристике софтвера су:

• Аутоматизација
- Захтева минималну интервенцију корисника током нормалног рада
- Обрађује више тест случајева узастопно
- Омогућава аутоматско чишћење између тестова

• Робустност
- Ефикасно управља грешкама
- Решава проблеме компатибилности на више платформи
- Пружа опцију за наставак тестирања чак и када дође до грешке

• Извештавање
- Креира детаљну листу резултата тестова у фајлу “tabela.izl”
- Читко форматиран излаз
- Пружа јасне индикаторе успеха/неуспеха

• Одржавање
- Добро структурисана организација потпрограма
- Доследни обрасци руковања грешкама
- Модуларни дизајн за лако ажурирање.

Представљени софтвер интегрише сукцесивно извршавање тестова, аутоматско откривање грешака, процедуре чишћења и компатибилан је са Windows и са GNU-Linux оперативним системима, што осигурава да РАК софтвер исправно ради у различитим рачунарским окружењима.

Основни циљ програма је да аутоматизацијом извршења прорачуна тест примера и евалуације резултата помогне развојном тиму РАК софтвера да брзо идентификују и исправе новонастале грешке, олакшавајући континуирани развој софтвера, чинећи га робуснијим и ефикаснијим. Софтвер представља структурни оквир за управљање тестовима, осигуравајући доследност у процесима верификације, смањујући ризик од неоткривених проблема и повећавајући укупни интегритет софтвера. Добро организована структура програма олакшава његово одржавање, ажурирање и проширење, осигуравајући да може да се прилагоди све већим захтевима тестирања РАК софтвера са новим примерима, чинећи га виталним делом животног циклуса развоја софтвера.

Још једна кључна предност представљеног програма је његова модуларна архитектура и постојање FORTRAN и Python верзије, које омогућавају флексибилност и скалабилност. Софтвер за тестирање је могуће ископирати у развојни директоријум, како за Windows у директоријум где се налази VisualStudio пројекат, тако и у GNU-Linux директоријум који садржи makefile и неопходне библиотеке за компилацију РАК софтвера.
Поред тога, софтвер садржи и ефикасно управљање датотекама, тако да вишеструко узастопно покретање тестирања не доводи до нагомилавања излазних фајлова. Ефикасно управљање грешкама даје неопходне информације кориснику да ли је грешка настала у софтверу за тестирање, или у самом РАК солверу.

***Верификација резултата техничког решења*** извршена је према прописаном протоколу о тестирању наручиоца и корисника техничког решења. Поступак тестирања који тренутно обухвата 37 карактеристична тест примера демонстриран је на прва два тест примера у којима је квадратна плоча оптерећена концентрисаном силом или притиском.

# 12 Захвалница

Ово техничко решење је сачињено уз финансијску подршку Фонда за науку Републике Србије. За садржину ове публикације искључиво је одговоран Факултет инжењерских наука, Универзитет у Крагујевцу и та садржина не изражава ставове Фонда за науку Републике Србије". Истраживање спроведено уз подршку Фонда за науку Републике Србије, БРОЈ 7475, Prediction of damage evolution in engineering structures – PROMINENT.

# 13 Референце

[1] M. Kojić, R. Slavković, M. Živković и N. Grujović, Metod konačnih elemenata I, Linearna analiza, Kragujevac: Mašinski fakultet, Univerzitet u Kragujevcu, 1998.

[2] J. Reid, „The new features of Fortran 2023," 2023.

[3] D. J. Worth, „State of the Art in Object Oriented Programming with Fortran," Science and Technology Facilities Council, 2008.

[4] S. Rossel, Continuous Integration, Delivery, and Deployment, O'Reilly, 2017.