УДК 004.94+621.88

РОЗРОБЛЯННЯ ТА АНАЛІЗ ПАРАМЕТРИЧНИХ СКІНЧЕННО-ЕЛЕМЕНТНИХ МОДЕЛЕЙ РІЗЬБОВИХ З'ЄДНАНЬ В ABAQUS®

В.Б.Копей

ІФНТУНГ; 76019, м. Івано-Франківськ, вул. Карпатська, 15; тел. (03422) 43024;
e-mail: v k o p e y @ r a m b l e r . r u

Розглядаються принципи побудови параметричних скінченно-елементних моделей різьбових з'єднань в середовищі скінченно-елементного аналізу Abaqus мовою програмування Python. Моделі можуть бути використані для комплексного аналізу і оптимізації.

Ключові слова: скінченно-елементна модель, різьбове з'єднання, система скінченно-елементного аналізу Abaqus, мова програмування Python

Рассматриваются принципы построения параметрических конечно-элементных моделей резьбовых соединений в среде конечно-элементного анализа Abaqus на языке программирования Python. Модели могут быть использованы для комплексного анализа и оптимизации.

Ключевые слова: конечно-элементная модель, резьбовое соединение, система конечно-элементного анализа Abaqus, язык программирования Python

Principles of development of parametric finite-element models of thread connections in the finite-element analysis system Abaqus with Python programming language are examined. Models can be utilized for a complex analysis and optimization.

Keywords: finite-element model, thread connection, finite-element analysis system Abaqus, Python programming language

Abaqus® – комп'ютеризована система скінченно-елементного аналізу (Finite Element Analysis – FEA) [1]. Abaqus реалізує ефективний чисельний метод розв'язування різноманітних задач механіки деформівного твердого тіла – метод скінченних елементів (МСЕ). Abaqus дає змогу розв'язувати статичні і динамічні задачі, а також сильно нелінійні перехідні швидкоплинні динамічні задачі. Він має зручний інтерфейс користувача, підтримує велику кількість типів скінченних елементів, дає змогу розв'язувати контактні і інші нелінійні задачі, має інтерфейси для відомих CAD, модулі для розв'язування вузькоспеціалізованих задач, наприклад, механіки руйнування. Abaqus успішно конкурує з такими відомими програмами FEA, як Ansys®, Nastran®, Comsol®, Cosmos®. Основною перевагою, яка відрізняє його від інших програм, є наявність інтерфейсу прикладного програмування (API) популярною мовою Python [2, 3].

Python – розповсюджена мова програмування загального призначення, яка характеризується простотою використання, підтримкою усіх основних парадигм програмування (в тому числі об'єктно-орієнтованого програмування), динамічною типізацією, зручними високорівневими структурами даних та великою бібліотекою стандартних функцій. Python API надає Abaqus широкі можливості ефективної автоматизації роботи і створення прикладних програм для побудови скінченно-елементних моделей, їх аналізу і розв'язування оптимізаційних задач.

Розробка і аналіз моделей в системах FEA, як правило, виконується в такій послідовності: побудова геометричної моделі; задання характеристик матеріалів, кроків навантаження, граничних умов, параметрів контакту і навантажень; побудова мережі скінченних елементів; розрахунок і аналіз результатів. Розробляти і аналізувати модель в Abaqus можна вручну, заповнюючи потрібні елементи дерева побудови (табл. 1), і автоматизовано – за допомогою Python програм. Переваги другого методу очевидні, зокрема можна створювати моделі з можливістю легкої зміни будь-якого її параметра (параметричні моделі). Розглянемо принципи розробки прикладної програми для побудови і аналізу скінченно-елементних осесиметричних моделей різьбових з'єднань нафтогазового обладнання.

Відомо, що об'єктно-орієнтований підхід у програмуванні дає змогу створити абстрактні програмні моделі реальних предметів і понять, що суттєво полегшує розробляння складного програмного забезпечення, уможливлює повторне використання окремих програмних компонентів і спрощення їх модифікації. В об'єктній моделі Abaqus об'єкти програми (геометричні моделі деталей, матеріали, зборки, кроки навантаження, контактні взаємодії, зовнішні навантаження, мережа скінченних елементів, двовимірні ескізи тощо) описані у вигляді класів і їх ієрархій мовою Python.

Для підключення модулів з класами Abaqus (модулі `part`, `material`, `section`, `assembly`, `step`, `interaction`, `load`, `mesh`, `job`, `sketch`, `visualization`) інтерпретатору Python слід виконати команду **`import`** , наприклад:

```
from part import *
```

Прикладний програміст може розробити власні класи і їх ієрархії, які описують певні поняття. Так, автором розроблено клас, що описує поняття розміру деталі. Будь-який розмір деталі має властивості: номінальне значення, верхнє відхилення, нижнє відхилення і дійсне значення. Використ-

Таблиця 1 – Дерево побудови моделі муфтового різьбового з'єднання НКТ в ABAQUS 6.8

Models (1)	Моделі
Model-1	Скінченно-елементна модель різьбового з'єднання
Parts (2)	Геометричні моделі деталей з'єднання
Materials (1)	Матеріали
Sections (2)	Секції
Profiles	
Assembly	Зборка і елементи зборки
Steps (3)	Кроки навантаження
Field Output Requests (1)	
History Output Requests (1)	
Time Points	
ALE Adaptive Mesh Constraints	
Interactions (1)	Опис контактної взаємодії
Interaction Properties (1)	Властивості контакту
Contact Controls	
Constraints	
Connector Sections	
Fields	
Amplitudes	
Loads (2)	Навантаження
BCs (2)	Граничні умови
Predefined Fields	
Remeshing Rules	
Sketches (4)	Ескізи деталей з'єднання
Annotations	
Analysis	Аналіз моделі
Jobs (1)	Задачі
Model-1 (Completed)	Розрахована модель
Adaptivity Processes	

товуючи ці дані можна обчислити максимальний і мінімальний допустимі розміри. Отже клас **Dim** містить такі члени-дані: `n, ei, es, v` і члени-функції: `__init__(), min(), max()`.

```
class Dim:
    '''Клас описує поняття розміру'''
    n=0.0 #номінальний розмір
    ei=0.0 #нижнє відхилення
    es=0.0 #верхнє відхилення
    v=0.0 #дійсне значення
    def __init__(self,n,ei,es):
        '''конструктор'''
        self.n=n
        self.ei=ei
        self.es=es
    def min(self):
        '''повертає мінімальний розмір'''
        return self.n+self.ei
    def max(self):
        '''повертає максимальний розмір'''
        return self.n+self.es
```

Розроблено також клас, який описує поняття матеріалу. Клас **Material** містить члени-дані, які описують механічні характеристики пружності і пластичності, та члени-функції, які повертають ці механічні характеристики в заданому форматі.

Бібліотека матеріалів може бути створена за допомогою типу Python - словник:

```
matlib={
'40':Material(E=210000.0e+6,mu=0.28,st=314.0e+6,
    sb=559.0e+6,delta=16.0,psi=45.0),
'20H2M':Material(E=210000.0e+6,mu=0.28,st=382.0e+6,
    sb=588.0e+6,delta=21.0,psi=56.0)
}
```

Тут кожен елемент словника є парою "рядок назва матеріалу"-"об'єкт класу `Material`".

Словник розмірів різьбового з'єднання певного типорозміру описується, наприклад, так:

```
nkt114={
'D':Dim(114.3,0.0,0.0),#зовнішній діаметр труби
'd':Dim(100.3,0.0,0.0),#внутрішній діаметр труби
}
```

Тут кожен елемент словника є парою "рядок назва розміру"-"об'єкт класу **`Dim`**".

Словник стандартних топорозмірів різьбових з'єднань відповідно ГОСТ:

```
nkt={114:nkt114,102:nkt102}
```

Тоді присвоїти дійсне значення розміру з назвою `'d'` з'єднання типорозміром `114` можна так:

```
d=nkt[114]
d['d'].v=d['d'].max()/2
```

Створюємо ескізи профілів заготовок деталей з'єднання та профілів їх різьби в осьовому перетині. Наприклад, наступний код створює ескіз, лінію з прив'язкою, розмір і відповідний йому параметр:

```
s=model.ConstrainedSketch(name='Sketch-1', sheetSize=200.0)
g2=s.Line(point1=(0.0, -15.0), point2=(0.0, 15.0))
s.VerticalConstraint(entity=g2)
d1=s.VerticalDimension(vertex1=g2.getVertices()[0],
    vertex2=g2.getVertices()[1],textPoint=(0.0, 0.0))
s.Parameter(name='p_n', path='dimensions[0]')
```

Для пошуку геометричних елементів моделі (точок, кромок, площин) в Abaqus Python API використовується функція `findAt`, параметром якої є послідовність з трьох координат точок. Тому потрібно задати характерні точки кромок моделі, на яких задаватимуться навантаження, граничні умови чи параметри контакту, наприклад:

```
en1=((d['D'].v+d['d'].v)/2,d['L'].v+20,0.0) #верхній торець ніпеля
```

Надаємо значення параметрам розмірів моделі, наприклад:

```
model=mdb.models['Model-1']
s=model.sketches['Sketch-1']
s.parameters['ln'].setValues(expression=str(d['L'].v+20))
```

Створюємо осесиметричну геометричну модель (`Part`) заготовки деталі з'єднання, наприклад, ніпеля:

```
model.Part(dimensionality=AXISYMMETRIC, name=n,
    type=DEFORMABLE_BODY)
model.parts[n].BaseShell(sketch=model.sketches[s])
```

Тут `n` - ім'я деталі, `s` - ескіз заготовки ніпеля.

Аналогічно створюємо іншу деталь - муфту.

Побудова профілю різьби деталей з'єднання є найскладнішим завданням, оскільки неточна побудова може призвести до некоректної геометрії або геометрії з дуже дрібними елементами. Наступна функція (**`createCut`**) створює частину профілю різьби шляхом послідовного переміщення ескізу профілю на величину кроку і створення ним вирізу:

```
def createCut(Part,Sketch,Begin,P,Fi,Len,X,Y,dx,dy):
 i=Begin#номер витка (0-перший)
 while i*P<=Len:#довжина різьби
   s=model.ConstrainedSketch(name='__profile__',sheetSize=200.)
   model.parts[Part].projectReferencesOntoSketch(filter=
       COPLANAR_EDGES, sketch=s)
   s.ConstructionLine(point1=(0.0,0.0), point2=(0.0, 10.0))
   s.retrieveSketch(sketch=model.sketches[Sketch])
   s.delete(objectList=(s.vertices.findAt((0.0,0.0),), ))
   s.move(objectList=s.geometry.values(),
       vector=(X+dx*P*tan(Fi*pi/180)*i,Y+dy*P*i))
   model.parts[Part].Cut(sketch=s)
   del s
   i=i+1
 return i-1
```

Тут `Part` - назва деталі, `Sketch` - назва ескізу, `Begin` - початок різьби (ціле), `P` - крок різьби, `Fi` - кут конуса конічної різьби (градуси), `Len` - довжина різьби, `X,Y` - початкові координати центра профілю, `dx` - радіальний напрямок подачі під час "нарізання" різьби (+1 - вправо, -1 - вліво), `dy` - осьовий напрямок подачі (+1 - вверх, -1 - вниз).

Наприклад, основна частина профілю різьби ніпеля створюється так:

```
createCut(Part='Part-1',Sketch='Sketch-3',
    Begin=0,P=d['P'].v,Fi=d['fi'].v,Len=d['L'].v-12.7,
    X=dsr,Y=d['L'].v-12.7,dx=-1,dy=-1)
```

Тут `dsr` - середній діаметр різьби в основній площині.

Використовуючи функцію `createCut` можна створювати циліндричні і конічні різьби, збіги різьби і інші різьбові поверхні.

Створюємо матеріали деталей з'єднання і присвоюємо їм механічні характеристики:

```
m=model.Material(name=n)
m.Elastic(table=et)
m.Plastic(table=pt)
```

Тут `n` - ім'я матеріалу, `et` - пружні характеристики, `pt` - пластичні характеристики.

Створюємо і присвоюємо секції (частини з одного матеріалу) деталям:

```
model.HomogeneousSolidSection(material=m, name=n, thickness=None)
model.parts[p].SectionAssignment(region=
    Region(faces=model.parts[p].faces), sectionName=n)
```

Тут `n` - ім'я секції, `m` - матеріал, `p` - деталь.

Створюємо елементи зборки різьбового з'єднання:

```
model.rootAssembly.Instance(dependent=OFF,
    name=n,part=model.parts[p])
```

Тут `n` - ім'я елемента, `p` - деталь.

Створюємо кроки статичного навантаження:

```
model.StaticStep(name=n, previous=pr)
```

Тут n - ім'я кроку, pr - попередній крок. Можна, наприклад, створити два кроки навантаження. На першому задається згвинчування з'єднання, на другому - зовнішнє навантаження.

Створюємо множину кромок, що контактують:

```
model.rootAssembly.regenerate()
ae=model.rootAssembly.instances[i].edges
e=ae.findAt(*ep)#*ep - розпакування кортежу
p=[x.pointOn for x in ae if x not in e]
model.rootAssembly.Set(name=n,edges=ae.findAt(*p))
```

Тут `n` - ім'я множини, `i` - елемент зборки, `ep` - кортеж точок кромок не для контакту (наприклад, `ep=((em1, ), (em2, ),))`).

Створюємо властивості контакту, в яких, зокрема, вказуємо формулювання контакту і коефіцієнт тертя:

```
model.ContactProperty('IntProp-1')
model.interactionProperties['IntProp-1'].TangentialBehavior(
    dependencies=0, directionality=ISOTROPIC,
    elasticSlipStiffness=None,formulation=PENALTY,
    fraction=0.005, maximumElasticSlip=FRACTION,
    pressureDependency=OFF, shearStressLimit=None,
    slipRateDependency=OFF,table=((0.05, ), ),
    temperatureDependency=OFF)
model.interactionProperties['IntProp-1'].NormalBehavior(
    allowSeparation=ON,constraintEnforcementMethod=DEFAULT,
    pressureOverclosure=HARD)
```

Створюємо контакт:

```
sm=model.rootAssembly.sets['Master']
ss=model.rootAssembly.sets['Slave']
model.SurfaceToSurfaceContactStd(adjustMethod=NONE,
    clearanceRegion=None, createStepName='Step-1',
    datumAxis=None, enforcement=SURFACE_TO_SURFACE,
    initialClearance=OMIT, interactionProperty='IntProp-1',
    interferenceType=SHRINK_FIT,master=
    Region(side1Edges=sm.edges), name='Int-1',
    slave=Region(side1Edges=ss.edges), sliding=SMALL,
    surfaceSmoothing=NONE,thickness=ON)
```

Створюємо множини кромок для навантаження і граничних умов:

```
s=model.rootAssembly.Set(edges=
  model.rootAssembly.instances[i].edges.findAt(ep), name=n)
```

Тут n - ім'я множини, і - елемент зборки, ер - кортеж характерних точок кромок (наприклад, `ep=(em1, )`).

Створюємо навантаження і граничні умови на кожному кроці навантаження. Наприклад, гранична умова, яка робить неможливим переміщення муфти:

```
s=model.rootAssembly.sets['Encastre']
model.EncastreBC(createStepName='Step-1',
    name='BC-2', region=Region(edges=s.edges))
```

Від розміру і кількості скінченних елементів суттєво залежить точність результатів і тривалість розрахунку. Задаємо загальний розмір скінченних елементів і кількість елементів вздовж дрібних кромок (наприклад, кромок різьби). Створюємо мережу скінченних елементів:

```
model.rootAssembly.seedPartInstance(deviationFactor=0.1,
    regions=(model.rootAssembly.instances['Part-1-1'],
    model.rootAssembly.instances['Part-2-1']), size=2.6)
sm=model.rootAssembly.sets['Master']
ss=model.rootAssembly.sets['Slave']
model.rootAssembly.seedEdgeByNumber(edges=sm.edges, number=4)
model.rootAssembly.seedEdgeByNumber(edges=ss.edges, number=4)
model.rootAssembly.generateMesh(regions=(
    model.rootAssembly.instances['Part-1-1'],
    model.rootAssembly.instances['Part-2-1']))
```

Моделювання згвинчування муфтового різьбового з'єднання НКТ виконується шляхом осьового зміщення муфти відносно ніпеля на величину, яка кратна кроку різьби (3.175):

```
model.rootAssembly.translate(instanceList=('Part-2-1', ),
    vector=(0.0, 2*3.175, 0.0))
    #моделювання згвинчування(0(вручну),1,2(станок))
```

Моделювання згвинчування муфтового різьбового з'єднання насосних штанг або замкового різьбового з'єднання бурильних труб можна реалізувати за допомогою функції `BoltLoad`, якою задається видовження ніпеля або скорочення муфти під час згвинчування.

Формулюємо задачу і розв'язуємо її:

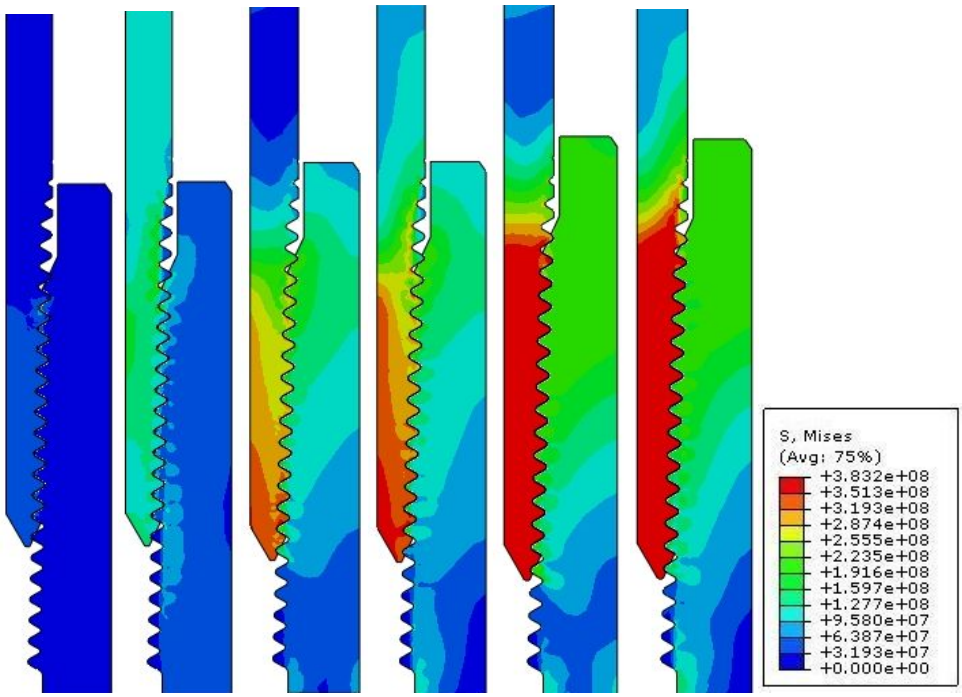
```
myJob = mdb.Job(name=model.name, model=model.name)
myJob.submit()
# Чекати поки задача не буде розв'язана
myJob.waitForCompletion()
```

Читаємо базу даних результатів. В кожному кроці навантаження знаходимо максимальне значення напруження за критерієм Мізеса в зоні з'єднання з іменем `'SET-6'`:

```
myOdb = openOdb(path=model.name + '.odb')
es=myOdb.rootAssembly.elementSets['SET-6']
#для кожного кроку
for s in myOdb.steps.values():
    fo=s.frames[-1].fieldOutputs['S'].getSubset(region=
        es,position=INTEGRATION_POINT)
    #знайти максимальне напруження за критерієм Мізеса
    max=0
    for v in fo.values:
        if v.mises>max: max=v.mises
    print max
```

Наступним етапом є порівняння знайденого максимального напруження за критерієм Мізеса в заданій зоні з'єднання з допустимим напруженням. За результатами порівняння може бути прийняте рішення про перехід до наступного наближення (зміни певного вхідного параметра, перебудови і розрахунку моделі), або про завершення ітерації. Таким чином реалізується оптимізація конструкції різьбового з'єднання.

Використовуючи ці принципи, автором розроблено програми для побудови і аналізу моделей муфтових різьбових з'єднань насосних штанг, НКТ (рис. 1) і замкових різьбових з'єднань бурильних труб. Така методика ефективна для комплексного аналізу і оптимізації фактично будь-яких конструкцій. Для зменшення трудомісткості побудови параметричних геометричних моделей слід використовувати такі САПР як SolidWorks®, CATIA®, Pro/ENGINEER® та їх інтерфейси з Abaqus.



а б в г д е

а,б – 6,5 мм; в,г – 3,325 мм; д,е – 0,15 мм

Рисунок 1 – Розподіл напружень за критерієм Мізеса (Па) в муфтовому різьбовому з'єднанні НКТ з умовним діаметром 114 мм без зовнішнього навантаження (а, в, д) і при навантаженні, яке створює напруження розтягу в тілі труби 100 МПа (б,г,е) при різних значеннях натягу

Література

1 Манилык Т. Практическое применение программного комплекса ABAQUS в инженерных задачах. Версия 6.5 / Тарас Манилык, Кирилл Ильин. — М.: МФТИ, ТЕСИС, 2006. — 67с.

2 Лутц Марк. Программирование на Python / Марк Лутц; перевод с английского. — СПб.: Символ-Плюс, 2002. — 1136 с.

3 Сузи Р. А. Python. Наиболее полное руководство / Р.А. Сузи. — СПб.: БХВ-Петербург, 2002. — 768 с.

Стаття надійшла до редакційної колегії 05.03.10
Рекомендована до друку професором ***Ю. Д. Петриною***