How to access Abaqus odb field output efficiently via Abaqus python script

Article abstract

This article will introduce the efficient way to use Abaqus python script to access Abaqus odb field outputs. The considerations for Abaqus python scripting are given.
A comparison of several unpacking styles to access odb outputs is made. The demonstration code is provided in Github: https://github.com/tao364744553/Abaqus-DNN.git.

Considerations for Abaqus python performance

1. Make use of appropriate object sequence unpacking. The style of sequence unpacking can significantly influence the efficiency of your Python scripts.
2. Make use of python built-in functions. They are usually optimized for performance.
3. Make use of 3rd-party modules. They are usually optimized for performance.
4. In general, working with high-level objects is more efficient than working with low-level objects.
5. Avoid accessing attributes that require the use of computationally intensive methods. For example: minimize the number of calls to getSubset() and addData() methods.

Comparison of unpacking sytels

The sequence unpacking style with the Abaqus object model can have a significant influence on performance. Consider the below three methods of access the same stress information:

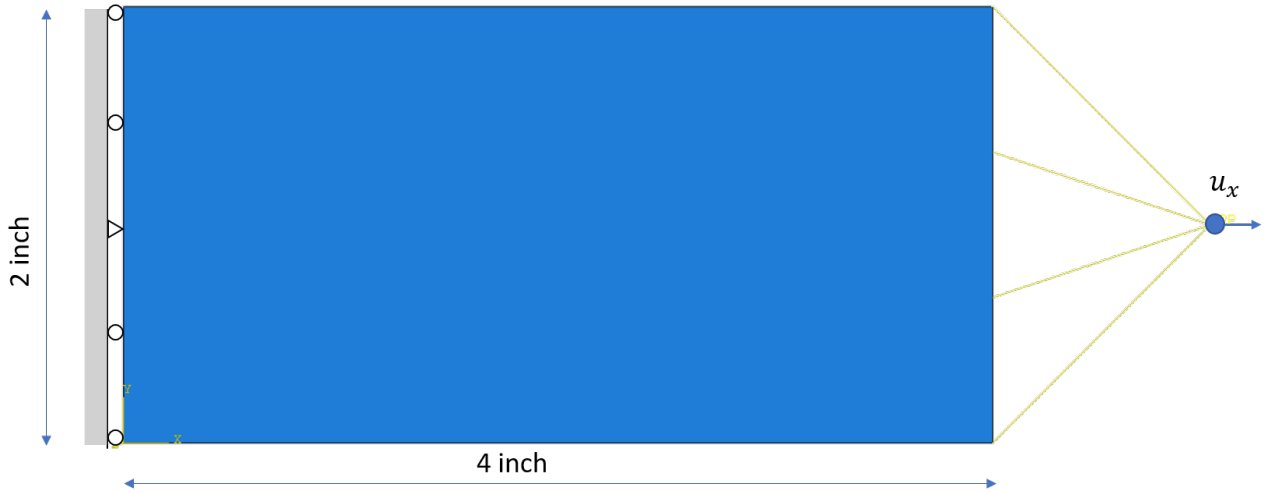
```
2   # Least efficient
3   for i in range(len(myStress.values))
4       myStress.values[i].data
5
6   # Moderate efficient
7   tmpStressValues = myStress.values
8   for i in range(len(tmpStressValues)):
9       tmpStressValues[i].data
10
11  # Most efficient
12  for value in (myStress.values):
13      value.data
14
```

For the first piece of code, it has to reopen the subdirectories in the odb dictionary every time it

accesses stress. Thus, the cost will be significantly expensive for a large model.
The second piece of code still needs to open the Value subdirectory. But it is more efficient than the first case.
The third piece of code shows the most efficient style. It is recommended to access odb outputs in this style.

Example to access multiple fields outputs

A plane stress model is fixed at one end and under a displacement $$u_x = 0.1$$ inch. The FEM model is described below



	Material properties
Elastic properties:	
E_1, psi	7.25×10^6
E_2, psi	1.16×10^6
G_{12}, psi	7.20×10^5
ν_{12}	0.3
Tensile strengths:	
X, ksi	246.00
Y, ksi	5.10
S, ksi	11.60
Compressive strengths:	
X', ksi	145.00
Y', ksi	17.40
S', ksi	6.70

HOW TO ACCESS ABAQUS ODB FIELD OUTPUT EFFICIENTLY VIA ABAQUS PYTHON SCRIPT

```python
#import necessary modules
from abaqus import *
from abaqusConstants import *
import visualization
import numpy as np
import timeit

#model information
number_integration_points = 98
number_U_direction = 2
observation_points =  list(range(0,231))
del_indices = [0, 21, 42, 63, 84, 105, 126, 147, 168, 188, 210]
observation_points = [i for j, i in enumerate(observation_points) if j not in del_indices]
observation_points = observation_points[0:-1:40]
observation_size = np.size(observation_points)
number_variables = 4
index_variables = [11,12,22,44]

#define the container for the fields outputs
disp=[]
stress = []
strain = []
d_U_D=[]

a = odb[0].steps['Step-1'].frames[-1]

#least efficient style to access displacement
start = timeit.default_timer()
for obs_opoint in observation_points:
    disp_tmp = a.fieldOutputs['U'].getSubset(position=NODAL).values[obs_opoint].data
    disp.append(disp_tmp)

stop = timeit.default_timer()
time_count = stop - start
print(time_count)

```

```
#intemediate efficient style to access displacement
start = timeit.default_timer()
disp_tmp =  a.fieldOutputs['U'].getSubset(position=NODAL).values
for obs_opoint in observation_points:
    disp_tmp1 = disp_tmp[obs_opoint].data
    disp.append(disp_tmp1)

stop = timeit.default_timer()
time_count = stop - start
print(time_count)

#Most efficient style to access displacement
start = timeit.default_timer()
disp_tmp = a.fieldOutputs['U'].getSubset(position=NODAL).values
disp = [disp_tmp[obs_opoint].data for obs_opoint in observation_points]
stop = timeit.default_timer()
time_count = stop - start
print(time_count)
```

The cost to access displacement for this model is shown below

```
>>> #least efficient style to access displacement
>>> start = timeit.default_timer()
>>> for obs_opoint in observation_points:
...     disp_tmp = a.fieldOutputs['U'].getSubset(position=NODAL).values[obs_opoint].data
...     disp.append(disp_tmp)
...
>>> stop = timeit.default_timer()
>>> time_count = stop - start
>>> print(time_count)
0.399884503469
>>>
>>> #intemediate efficient style to access displacement
>>> start = timeit.default_timer()
>>> disp_tmp =  a.fieldOutputs['U'].getSubset(position=NODAL).values
>>> for obs_opoint in observation_points:
...     disp_tmp1 = disp_tmp[obs_opoint].data
...     disp.append(disp_tmp1)
...
>>> stop = timeit.default_timer()
>>> time_count = stop - start
>>> print(time_count)
0.0600312045935
>>>
>>> #Most efficient style to access displacement
>>> start = timeit.default_timer()
>>> disp_tmp = a.fieldOutputs['U'].getSubset(position=NODAL).values
>>> disp = [disp_tmp[obs_opoint].data for obs_opoint in observation_points]
>>> stop = timeit.default_timer()
>>> time_count = stop - start
>>> print(time_count)
0.0557082583473
...
```