Iterating on a Sofistik .dat file with a Python interface

https://github.com/SarvAster/Beam_Sofistik_POC/blob/master/README.md || ACTING / PINI / ARX ENGINEERING

Abstract This paper presents the protocol followed to iterate on a structure exported in a Sofistik .dat file. It discusses the choices made to come to the solution, which is a proof of concept in order to further interact with Sofistik files on bigger projects. The iteration is here made on a simple vertical cantilever beam by applying a vertical and an horizontal load to its extremity and repeating linear approximations until convergence under a chosen epsilon threshold, or until divergence is stated.

Theoretical approach

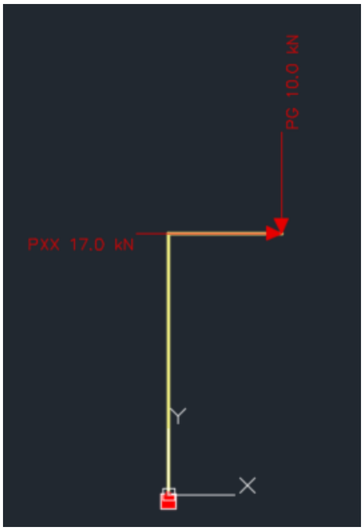
Small deflection Euler-Bernouilli beam closed form and Taylor approximation

In the frame of the approximations, and supposing that the beam parallel to the ground stays parallel with the deformations, we disinvolve the following moment curvature-curvature law (E) :

$$\begin{aligned}
(E) &\Leftrightarrow \frac{d^2u}{dy^2} = \frac{M}{EI} \\
&\Leftrightarrow \frac{d^2u}{dy^2} = \frac{(u(h) - u(y) + e)V + (h - z)H}{EI} \\
&\Leftrightarrow \frac{d^2u}{dy^2} = \frac{V}{EI}(u(h) - u(y) + e) + \frac{H}{EI}(h - y) \\
&\Leftrightarrow \frac{d^2u}{dy^2} = \alpha^2(u(h) - u(y) + e) + \beta(h - y)
\end{aligned}$$

with $\alpha^2 = \frac{V}{EI}$ and $\beta = \frac{H}{EI}$

doi:10.1017/xxxxxxx xxxx



Notes: Here are some notes.

FIGURE 1. *Sofiplus cantilever diagram with axial loads at the tip*

We declare a new displacement variable subject to the conditions :

$$\tilde{u} = u - u(h), \quad \tilde{u}(l) = 0, \quad \tilde{u}''(0) = 0$$

$$(E) \quad \Leftrightarrow \quad \frac{d^2\tilde{u}}{dy^2} + \alpha^2\tilde{u} = \alpha^2 y + \beta(h-y)$$

$$\Leftrightarrow \quad \tilde{u} = A\cos(\alpha y) + B\sin(\alpha y) + ey + \frac{H}{V}(h-y)$$

Solving for A and B given the boundary conditions, we get u(y) :

$$u(y) = \left(\frac{H}{V}\frac{\tan(\alpha h)}{\alpha} + \frac{e}{\cos(\alpha h)}\right)(1-\cos(\alpha y)) + \frac{H}{V}\left(\frac{\sin(\alpha y)}{\alpha} - y\right)$$

And at height h :

$$w = u(h) = \frac{H}{V}\left(\frac{\tan(\alpha h) - h}{\alpha}\right) + e\left(\frac{1}{\cos(\alpha h)} - 1\right)$$

$$w \underset{0}{\sim} \frac{Hh^3}{3EI} + \frac{eVh^2}{2EI}$$

which holds for $V \ll EI$ i.e for $\alpha^2 \ll 1$

This final approach then holds for small vertical loads compared to the bending stiffness, considering that we already limited the beams bending as the curvature was approximated as the second derivative of displacements which implied H to be also small.
Sofistik direclty applies the results of the second approximation when running a linear analysis, and can get to the non-approximated

Iteration Model

If Sofistik can indeed output the result that we achieve by iterating with a non linear calculation, our objective here is not to specifically improve this matter but to gain knowledge on the interactions and advantages of using python, as iterating allows to design tasks that are not natively handled by Sofistik. The iteration goes as follows :

initialization The cantilever is perpendicular to the ground, the efforts V and H are applied and displacement u is obtained.

iteration A new cantilever is now designed to fit the deformations u(n) of the previous one, but without any loads applied, which means it inherits the geometric structure after load calculations of the cantilever from the previous step, but is free of any loads at this point. We now apply the same loads V and H and compute the next deformation u(n+1)

termination condition Once the difference of successive max displacements is smaller that epsilon, the algorithm stops. An error is printed if it diverges.

It can be show that this iteration exactly converges to the closed form solution of the Euler-Bernoulli beam equation.

Extension structure and use

Code Structure

The code is structured in two python scripts : flamb.py and BeamIter.py, that respectively are the set of classes that are used to interact with the dat file, retrieve information from the CDB Sofistik database, and run the iteration ; as the second script handles the Graphic Interface with pyQT. It also constains a config.ini file that retains Sofistik's datapath. The BeamIter.py script essentially call the Iteration() class from the flamb.py

which itself makes use of the rest of the classes of flamb.py that are CDBinteract, reponsible for retrieving the data from the .dat database (node positions and displacements), FileInteraction() that overwrites the values of the beam coordinates and loads in the .dat file, and SofiFileHandler that runs the calculations when iterating. The Iteration() class then makes use of the three previous classes in that order. Using pyinstaller :

```
!pip install pyinstaller
```

We were then able to package both scripts into an app. It takes as an input : the datapath of the Sofistik file (to run the .dat), the .dat file datapath, the vertical and horizontal loads and finally the precision threshold at which the iterations stop. The app has been scripted in such a way that it is flexible regarding Sofistik's datapth

which can be subjet to change. However, due to dynamic datapathing issues, there was no straightforward way to determine the DLL's path in the code without encountering a frozen type error. It is then necessary to repackage the app for each Sofistik yearly version release. However, we made that to be easy as it is only necessary to modify the DLL file inside the DLL folder and repackage the app by running the following py intaller command :

```
pyinstaller –onefile –noconsole –add-data "config.ini;." BeamIter.py
```

Where the BeamIter.py actually is the name of the "main" script, that is to say the one responsible for the GUI.

What we proved

Besides the application that in itself is mundane, this project is the occasion to prove a few points. First, it is indeed possible to iterate on Sofistik .dat files with python through database interactions (while it is not possible to directly iterate on a Sofistik file) ; second, we are able to package an app which furthermore is flexible to sofistik datapath locations ; third, while it might be attainable to create a once and for all app that adapts to DLL path changes, it is easy to adapte to any sofistik updates through the modification of one file and the execution of one command in the terminal.

User's Guide

As the use of the app is basically self explanatory, this section is dedicated to showing how to specifically modify the DLL and repackage the app.
The files you are going to download by cloning the git's main branch can be found here :
https://github.com/SarvAster/Beam_Sofistik_POC/tree/main

To clone the git repo, if you have no git account, you first have to create one by signing up on https://github.com and then downloading git https://git-scm.com/downloads.

And then in your terminal input the following :

```
git config –global user.name "Your Name"
git config –global user.email "your.email@example.com"
```

If you already have git configured, you only have to clone the git "main" branch with the following commands :

```
cd "datapath of the chosen folder to store the files"
git clone –branch main https://github.com/SarvAster/Beam_Sofistik_POC.git
```

Finally, run the pyinstaller command :

```
pyinstaller –onefile –noconsole –add-data "config.ini;." BeamIter.py
```

You now dispose of the BeamIter.exe app in the folder that you have chosen for your scripts, inside of the dist folder. You can now copy it to your desktop or wherever you want to run it independently of any of the scripts designed to create the app, as long as you have Sofistik on your computer.

As mentionned previously, to repackage the app when a new version of sofistik releases, you only have to change the dll in the DLL folder and rerun the pyinstaller command. It might even be possible that it is not necessary to go through this process as the dll might stay the same.