symbeam 1.0.8

```
pip install symbeam
```

Latest version

Released: Mar 7, 2021

A pedagogical package for bending diagrams

Navigation

Project description

Release history

Download files

Project links

Project description

A pedagogical package for beam bending.

Download

Tracker

Source

Statistics

GitHub statistics:

- **Stars:** 5
- **Forks:** 1
- **Open issues/PRs:** 5

View statistics for this project via Libraries.io, or by using our public dataset on Google BigQuery

Meta

License: MIT License (MIT)

Author: António Manuel Couto Carneiro @FEUP

bending-moment-diagrams, statics, sympy, python3

Industrial Engineering students learning the fundamentals of bending of beams, namely, bending diagrams and deflections.

The modular object-oriented-based design of SymBeam combined with the excellent symbolic engine SymPy, on which SymBeam relies heavily, provides a unique computational learning environment for students grasping these concepts for the first time. SymBeam can be exploited to quickly assess the solutions of exercises for a wide variety of bending loadings and supports while allowing to easily modify the parameters of the problem, fostering physical intuition and improving the students' understanding of the phenomena.

Conversely, SymBeam can also be used by teachers to create and validate new problems for classes and exams, facilitating this sometimes cumbersome task.

The following paragraphs provide a detailed description of how to use SymBeam for solving a beam equilibrium problem, together with the extent of its capabilities and also limitations. Check the comprehensive list of SymBeam application examples for a more visual overview.

Try it out

You do not need to be a *Python Pro* or even know Python at all to use SymBeam in your study. Explore this interactive notebook and enjoy from SymBeam features in your study!

As of February 2021, SymBeam is now part of Centric Engineers online platform. Here you can find a very friendly Graphical User Interface for multiple open-source projects on structural engineering. Check the their bending diagrams utility powered by SymBeam!

Installation

Maintainers

amcc

Classifiers

Development Status
- 4 - Beta

Environment
- Console

Intended Audience
- Science/Research

License
- OSI Approved :: MIT License

Natural Language
- English

Programming Language
- Python :: 3.6
- Python :: 3.7
- Python :: 3.8

Topic
- Scientific/Engineering

Clone this repository into your system

```
git clone git@github.com:amcc1996/symbeam.git
```

and install the Python package with `pip3`, running the following command inside SymBeam root directory, where the `setup.py` is located

```
pip3 install .
```

Alternatively, you can install it directly from PyPI with

```
pip3 install symbeam
```

At this point, SymBeam can be imported into your Python scripts and modules the usual Python-way

```
import symbeam
```

Theory

SymBeam is based on classical Solid Mechanics and Strenght of Materials results to solve the beam equilibirum problem. A simple outline follows in the present section, without entering in any mathematical derivations.

moment. These are computed by solving the algebraic system of linear equations arising from the equilibirum of forces and moments of the structure, accounting simultaneously for point loads and moments and distributed forces.

$$\sum F_y = 0 \qquad \sum M_O = 0$$

- **Bending diagrams** - the shear force and bending moment diagrams are computed by integrating the differential equations of equilibirum of the beam and imposing the boundary conditions in a sequential manner, starting from the initial point at `x0`. The expressions obtained at the previous segment are used to set the boundary conditions for the next one.

$$\frac{\mathrm{d}V}{\mathrm{d}x} = -q(x) \qquad \frac{\mathrm{d}M}{\mathrm{d}x} = -V(x)$$

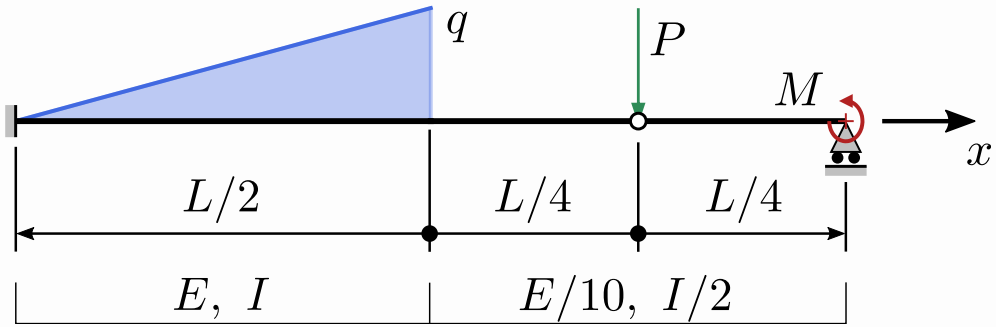
- **Deflection** - the slope and deflection of the beam are obtained by integration the elastic curve equation in each segment one and two times, respectively. The geometrical boundary conditions are used to build a system of algebraic equations for the integration constants (twice the number of segments).

$$\frac{\mathrm{d}^2 v}{\mathrm{d}x^2} = \frac{M}{EI}$$

Usage

All useful features of SymBeam can be accessed through the `beam` class. `beam` objects, this is, concrete instances of the `beam` class, are initially defined by the starting x-coordinate and the beam length (tacitly assumed to be in the positive x-direction). The beam's supports, material and section properties and loadings are set by calling a specific set of methods on the beam object.

using numerical input (e.g. `100`) or a literal expression (`100*x**2 + 100`). In any case, this input is `sympified` using `SymPy` facilities, allowing to handle input of distinct types out-of-the-box. The example to be analysed is illustrated in the following figure (distributed loads on the top side of the beam denote downward forces).



> *:warning: The `x` symbol is used by SymBeam as the independent variable for the position along the beam. This variable must be used to specify any variation along the length of the beam and for nothing else.*

Creating a beam

The fundamental tool for a bending analysis with SymBeam is a `beam` object, as emphasised above. To create a new beam, import the `beam` class from the SymBeam package. Then, simply call the `beam` constructor by passing the length of the beam and, if needed, a starting point (0 by default). For instance, a beam with a length equal to 1 and starting at 0 can be created by

```
from symbeam import beam

new_beam = beam(1, x0=0)
```

alternatives for instantiating a beam follows (the optional initial position `x0` is omitted here, for simplicity). Note that these alternatives also apply to any input data that can be given to `beam` methods, for instance, for specifying supports, loads and properties.

1. Numeric input

```
from symbeam import beam

new_beam = beam(1)
```

1. Numeric input from string

```
from symbeam import beam

new_beam = beam("1")
```

1. Symbolic input from string

```
from symbeam import beam

new_beam = beam("L")
```

1. Symbolic input from a symbolic variable created with SymPy

```
L = sympy.symbols("L")
new_beam = beam(L)
```

1. Symbolic input from a symbolic variable provided by SymPy

```
from symbeam import beam
from sympy.abc import L

new_beam = beam(L)
```

Setting beam properties: Young modulus and second moment of area

A beam must be associated with some distribution of material properties and section geometry along its length, namely, the Young modulus of the material and the second moment of area of the section. While these are not required for finding the bending diagrams, as these results simply from equilibrium considerations, they are mandatory for computing the deflections of the beam.

In SymBeam, these properties can be set in individual segments along the beam, such that the set of segments for each property must encompass all the beam span and not be overlapping at any region. For example, consider a beam of length `L`, the Young modulus and second moment of area are set by passing the starting and ending coordinate and the value to the methods `set_young()` and `set_inertia()` as follows

```
new_beam = beam(L)

# new_beam.set_young(x_start, x_end, value)
new_beam.set_young(0, L/2, E)
new_beam.set_young(L/2, L, E/10)

# new_beam.set_inertia(x_start, x_end, value)
new_beam.set_inertia(0, L/2, I)
new_beam.set_inertia(L/2, L, I/2)
```

By default, if the properties are not explicitly set by the user, SymBeam considers constant values `E` and `I` along the span of the beam, this is, the property setting methods do not need to be evoked. If any segment is explicitly set, the user must then consistently specify all segments.

> *:warning: **Our beloved symbols E and I**: Be careful when specifying symbolic Young modulus and second moment of area via strings, for instance, with "E" and "I". SymPy parses the string in the expression and will interpret "E" as the Euler's number and "I" as the imaginary unit. Prioritise using the variables directly imported from `sympy.abc` or create the variables directly with `sympy.symbols()`.*

Adding supports

The beam must be connected to the exterior via a given number of supports, which materialise the geometric boundary conditions of the problem. Currently, SymBeam can only solve statically determinate beams, therefore, redundant supports cannot be handled. Supports can be added to the beam by specifying the coordinate and the type of support. Exemplarily, this is accomplished by calling the method `add_support()`

```
new_beam.add_support(L, 'roller')
new_beam.add_support(3*L/4, 'hinge')
```

The types of support available in SymBeam are

- `roller` : a roller, fixed in the transverse direction and allows rotations in the bending plane
- `pin` : a pinned support, fixed in the axial and transverse directions and allows rotations in the bending plane
- `fixed` : a fixed/clamped support, all degrees of freedom are constrained (no displacements and no rotation)
- `hinge` : allows distinct rotations on the left and right of the point, but does not fix the beam in any direction

Adding loads

The applied external loads are the missing item for completely defining the beam bending problem. These can be point-type, namely, transverse point loads/forces and moments, and segment-type loads, this is, transverse forces distributed along the span of the beam.

Point loads and moments are incorporated by calling the `add_point_load()` and `add_point_moment()` methods, which receive the coordinate of the point and the value of the load. Distributed loads are applied by calling the `add_distributed_load()` method, which takes the starting and ending point of the distributed load and the associated expression.

```
new_beam.add_point_load(3*L/4, -P)
new_beam.add_point_moment(L, M)
```

Solving the problem

After specifying the beam properties, supports and loads, the problem can be solved by calling the method `solve()`. The program will proceed as follows

1. check if the input data is consistent
2. define the individual beam segments, such that each one is associated with a continuous function of the Young modulus, second moment of area and distributed load: in sum, this subdivision must guarantee that the shear force and bending moment diagrams are continuous in each segment and piecewise continuous along the span of the beam
3. solve for the reaction forces and moments of the supports (equilibrium equations)
4. solve for the internal loads (integrate the differential equations for beam equilibrium)
5. solve for the deflections (integrate the elastic curve equation)
6. output the results (can be suppressed if the optional argument `output=False`): identified segments, exterior reactions, shear force, bending moment, slope and deflection for each beam segment. For the current example, the output shall be as follows.

```
                                Beam points
================================================================================
     Coordinate              Type                 Load                Moment
--------------------------------------------------------------------------------
          0                 Fixed                  0                   0
        L/2           Continuity point             0                   0
      3*L/4                Hinge                  -P                  0
          L                Roller                 0                   M
================================================================================
```

Span	Young modulus	Inertia	Distributed load
[0 - L/2]	E	I	-q*x
[L/2 - 3*L/4]	E/10	I/2	0
[3*L/4 - L]	E/10	I/2	0

Exterior Reactions

Point	Type	Value
0	Force	L**2*q/8 + P + 4*M/L
0	Moment	L**3*q/24 + 3*L*P/4 + 3*M
L	Force	-4*M/L

Internal Loads

Span	Diagram	Expression
[0 - L/2]	V(x)	-L**2*q/8 - P + q*x**2/2 - 4*M/L
[0 - L/2]	M(x)	-L**3*q/24 - 3*L*P/4 - 3*M - q*x**3/6 + x*(L**3*q +
[L/2 - 3*L/4]	V(x)	-P - 4*M/L
[L/2 - 3*L/4]	M(x)	-3*L*P/4 - 3*M + x*(P + 4*M/L)
[3*L/4 - L]	V(x)	-4*M/L
[3*L/4 - L]	M(x)	-3*M + 4*M*x/L

```
================================================================================
        Span          Variable                     Expression
--------------------------------------------------------------------------------
 [   0   -  L/2  ]      v(x)    -q*x**5/(120*E*I) + x**2*(-L**3*q - 18*L*P - 72*M)/(
 [   0   -  L/2  ]    dv/dx(x)  -q*x**4/(24*E*I) + x*(-L**3*q - 18*L*P - 72*M)/(24*E
--------------------------------------------------------------------------------
 [  L/2  - 3*L/4 ]      v(x)    L**2*(L**3*q - 950*L*P - 3800*M)/(960*E*I) + L*x*(-L
 [  L/2  - 3*L/4 ]    dv/dx(x)  L*(-L**3*q + 608*L*P + 2432*M)/(128*E*I) + x*(-15*L*
--------------------------------------------------------------------------------
 [ 3*L/4 -   L   ]      v(x)    -L**2*(37*L**3*q + 1840*L*P + 16960*M)/(1920*E*I) +
 [ 3*L/4 -   L   ]    dv/dx(x)  L*(37*L**3*q + 1840*L*P + 48960*M)/(1920*E*I) - 60*M
================================================================================
```

> :warning: ***Don't be scared by the output:*** *The chosen example encompasses several features of SymBeam, therefore, the analytical expressions tend to grow in size very rapidly, especially the deflection.*

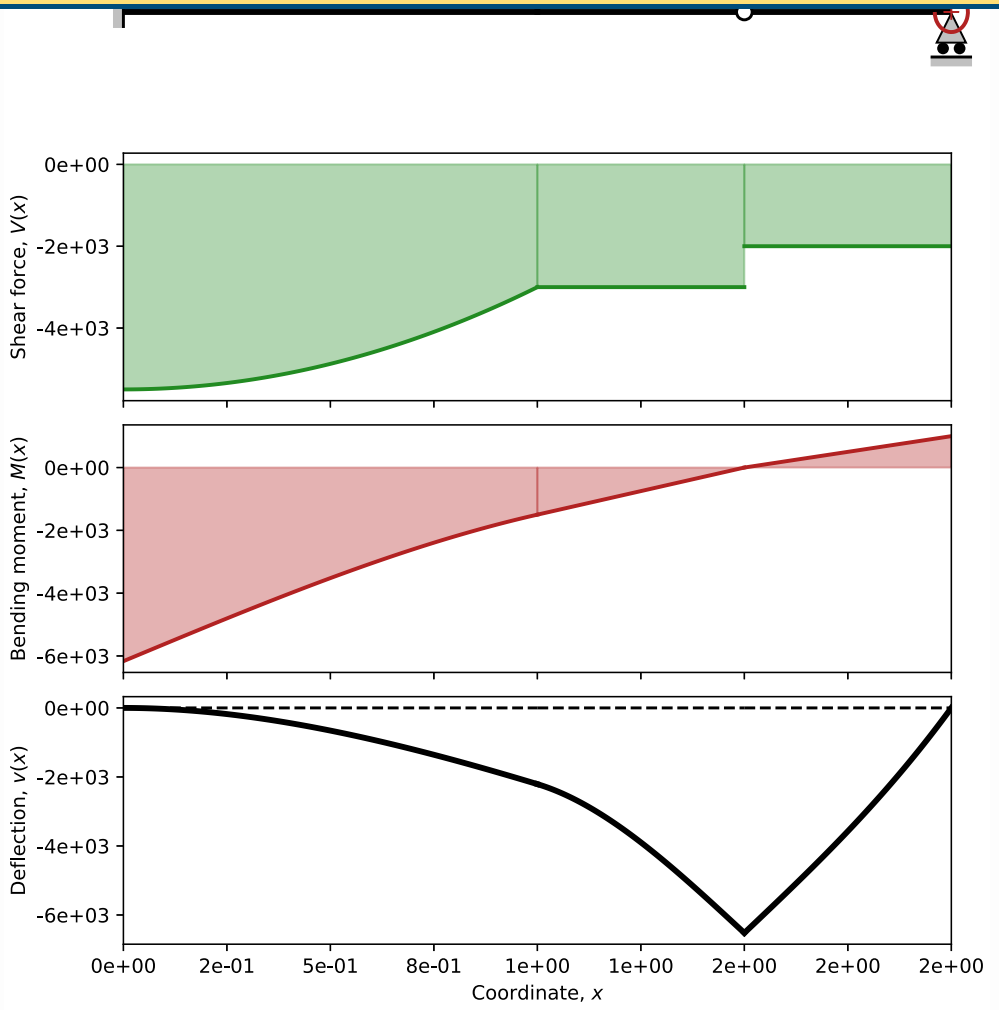
Plotting the results

The results can be plotted with matplotlib by calling the method `plot` on the beam object. The produced figure contains

1. a schematic representation of the problem
2. the shear force diagram
3. the bending moment diagram
4. the deformed shape of the beam.

is can be accomplished by passing the optional argument `subs` to the `plot` method. This must be a dictionary whose keys are the string representations of the variables and the values are the effective numerical values.

Adopting the substitutions `L=2`, `P=1000`, `q=5000` and `M=1000`, SymBeam outputs the figure below.



> *:warning: Do not forget to save the figures with* `savefig()` *method from* `matplotlib.pyplot.figure`*. In fact, you can also simply print the figure to the screen with* `show()` *from* `matplotlib.pyplot`*, but be aware that this might unformat the layout slightly, depending on the characteristics of your system.*

Here you can find the complete script discussed on the previous sections.

```
from symbeam import beam
from sympy.abc import L, E, I, P, M, q, x
import matplotlib.pyplot as plt

new_beam = beam(L)

# new_beam.set_young(x_start, x_end, value)
new_beam.set_young(0, L/2, E)
new_beam.set_young(L/2, L, E/10)

# new_beam.set_inertia(x_start, x_end, value)
new_beam.set_inertia(0, L/2, I)
new_beam.set_inertia(L/2, L, I/2)

# new_beam.add_support(x_coord, type)
new_beam.add_support(0, 'fixed')
new_beam.add_support(L, 'roller')
new_beam.add_support(3*L/4, 'hinge')

new_beam.add_point_load(3*L/4, -P)
new_beam.add_point_moment(L, M)
new_beam.add_distributed_load(0, L/2, -q * x)

new_beam.solve()

new_beam.plot(subs={'P':1000, 'q':5000, 'L':2, 'M':1000})

plt.savefig("beam.pdf")
```

SymBeam tests can by run with pytest and the image comparison plugin pytest-mpl, so start by installing the framework

```
pip3 install pytest pytest-mpl
pip3 install pytest-cov # optional, to generate coverage reports
```

and launch the testing utility from SymBeam root directory

```
make tests
```

SymBeam uses pytest-mpl for comparing the bending plots between versions. By evoking `make tests`, pytest will be called with the appropriate command-line tool and directory settings. The reference images are stored in tests/baseline. If there image comparison fails, the baseline image is written to tests/results, together with the (failing) image produced my the current version and the respective difference.

The coverage reports can be generated with

```
make coverage
```

which will run the test and create the coverage information in `htmlcov`.

License

Python Software Foundation 20th Year Anniversary Fundraiser Donate today!

Help

- Installing packages
- Uploading packages
- User guide
- FAQs

About PyPI

- PyPI on Twitter
- Infrastructure dashboard
- Package index name retention
- Our sponsors

Contributing to PyPI

- Bugs and feedback
- Contribute on GitHub
- Translate PyPI
- Development credits

Using PyPI

- Code of conduct
- Report security issue
- Privacy policy
- Terms of use

Status: All Systems Operational

Developed and maintained by the Python community, for the Python community.
Donate today!

Site map

Switch to desktop version