FConcrete Documentation

Luis Coimbra

Mar 01, 2020

General Information:

CHAPTER 1

FConcrete

Concrete beams according to NBR:6118:2014. Usage examples here.

- Free software: MIT license
- Documentation: https://fconcrete.readthedocs.io.

Warning: This is a project in a alpha version. Much testing is needed yet. **Do not use on your real life projects.**

1.1 A Quick Introduction

FConcrete is a python package to calculate the steel bars (longitudinal and transversal) with less material cost as possible and in a human friendy way (see default configs):

```
n1 = fc.Node.SimpleSupport(x=0)
n2 = fc.Node.SimpleSupport(x=400)
f1 = fc.Load.UniformDistributedLoad(-0.3, x_begin=0, x_end=400)

concrete_beam = fc.ConcreteBeam(
        loads = [f1],
        nodes = [n1, n2],
        section = fc.Rectangle(30,80),
        division = 200
)
```

It is also implemented a Analysis Class that can help you to get the best retangular section for your beam. As you can see on the documentations, by the default all units are in cm, kN or combination of both.

1.2 Features

- Define input parameters: available materials, cost, geometry definition, loads, fck, etc
- Calculation of efforts at any point
- Moment diagram decalaged
- Section balance and calculation of the required steel area
- Anchorage length
- Remove longitudinal bars
- Calculation of transversal steel bar (area per cm)
- Check limits and spacing per transversal bar span
- Check compliance with the steel area limits
- Calculation of rotation at any point
- Calculation of displacement at any point
- Implement test routines comparing Ftool
- Check dimensioning in E.L.S (except rupture)
- Associate costs with materials
- Program expense calculation function
- Create interaction functions and create table to follow the convergence of the algorithm
- Examples of tool usage (completion for optimized pre-dimensioning)
- Program expense calculation function
- Documentation
- Dinamic calculation of d (steel height) when there is change of the expected steel position

1.3 TODO

- Check rupture (ELS)
- Check minimum area on the support
- Draw the beam
- Correct displacement value when there is variation of E * I along the beam

- Plot correctly when stirrups are not vertical
- Plot longitudinal bars correctly when the height or position of the beam base changes.
- Calculate the total length of the bar correctly when the height or position of the beam base changes.
- Implement compression armor

1.4 Installation

To install FConcrete, run this command in your terminal:

```
$ pip install fconcrete
```

This is the preferred method to install FConcrete, as it will always install the most recent stable release. If you don't have pip installed, this Python installation guide can guide you through the process.

1.5 Credits

Most of vectorized calculus made with Numpy, unit conversion with Pint, all plots with Matplotlib, minor functions with Scipy, docs made with the help of Sphinx and Numpydoc, analysis table with Pandas, this package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

1.5.1 Installation

Stable release

To install FConcrete, run this command in your terminal:

```
$ pip install fconcrete
```

This is the preferred method to install FConcrete, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for FConcrete can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/luisggc/fconcrete
```

Or download the tarball:

```
$ curl -OJL https://github.com/luisggc/fconcrete/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

1.5.2 Usage

To use FConcrete in a project:

```
import fconcrete as fc
```

Beam Usage Example

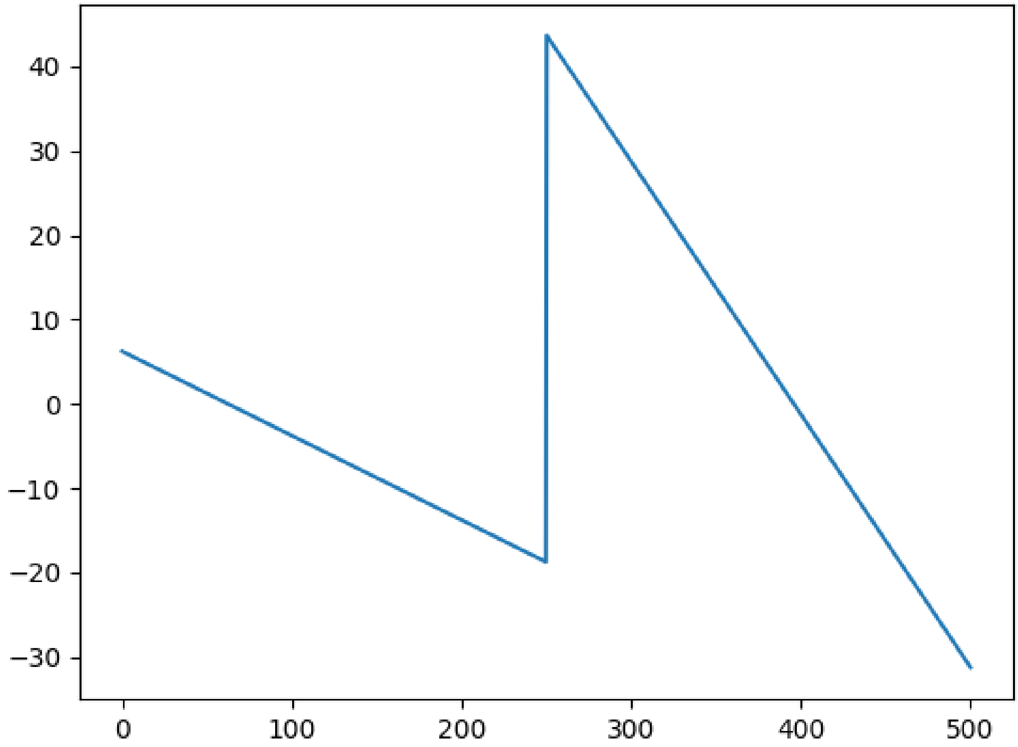
How to create a beam:

```
In [1]: import fconcrete as fc

In [2]: n1 = fc.Node.SimpleSupport(x=0)

In [3]: n2 = fc.Node.SimpleSupport(x=250)

In [4]: n3 = fc.Node.SimpleSupport(x=500)

In [5]: f1 = fc.Load.UniformDistributedLoad(-0.1, x_begin=0, x_
↪end=250)

In [6]: f2 = fc.Load.UniformDistributedLoad(-0.3, x_begin=250, x_
↪end=500)

In [7]: section = fc.Rectangle(12, 25)

In [8]: material = fc.Material(E=10**6, poisson=1, alpha=1)

In [9]: beam_element_1 = fc.BeamElement([n1, n2], section, material)

In [10]: beam_element_2 = fc.BeamElement([n2, n3], section,␣
↪material)

In [11]: beam = fc.Beam(loads=[f1, f2], beam_elements=[beam_element_
↪1, beam_element_2])
```

You can use all properties and methods of the *Beam Class* such as plot shear diagram, momentum, etc.

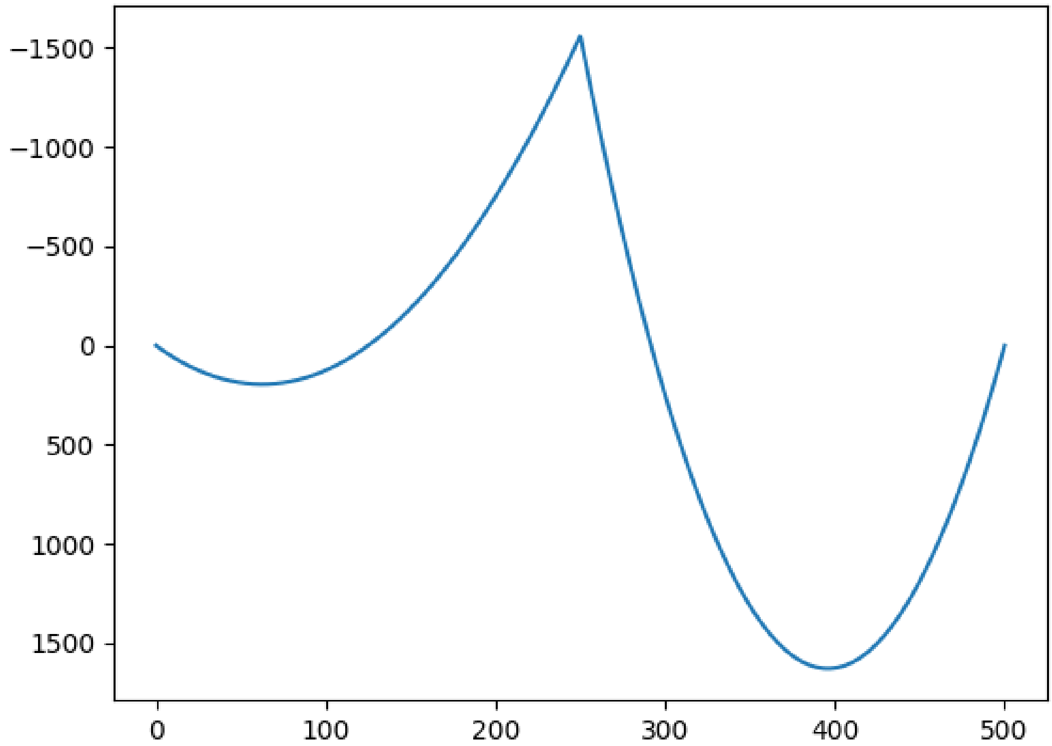
Plot Shear Diagram:

```
In [12]: beam.plotShearDiagram()
Out[12]:
(<Figure size 640x480 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fab053ef4e0>)
```



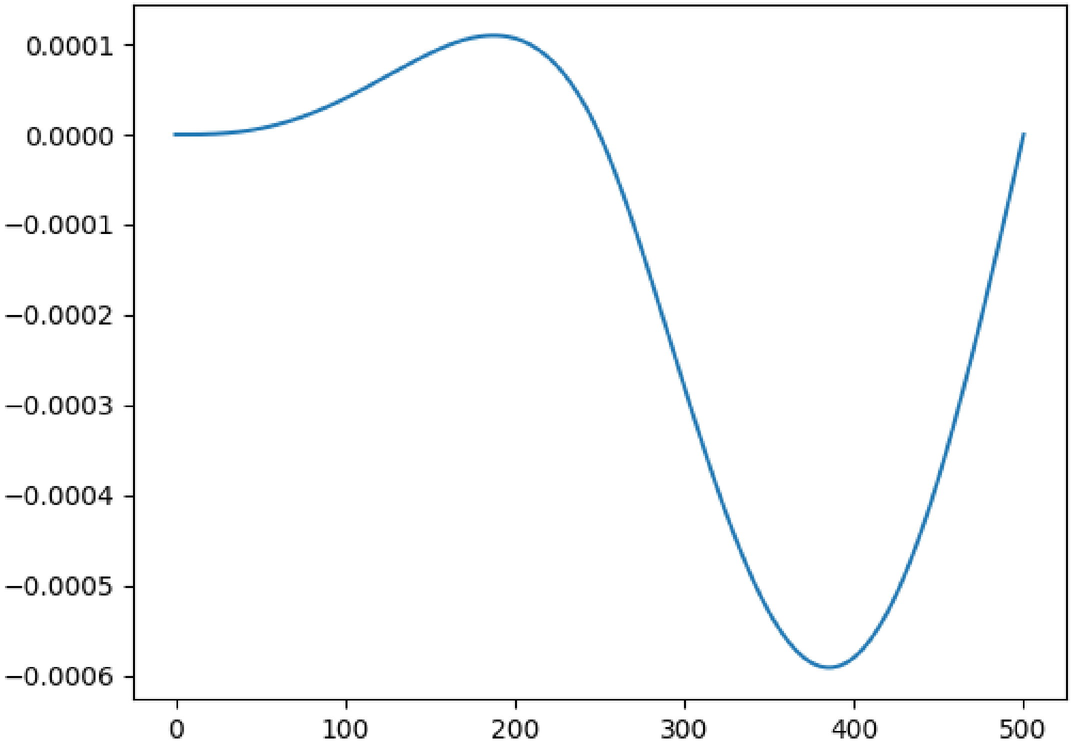
Plot Momentum Diagram:

```
In [13]: beam.plotMomentumDiagram()
Out[13]:
(<Figure size 640x480 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fab04f38390>)
```



Plot Displacement Diagram:

```
In [14]: beam.plotDisplacementDiagram()
Out[14]:
(<Figure size 640x480 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fab04f91e48>)
```



If you only want to get the values, but not to plot. You can use the "get" instead of "plot".

```
In [15]: x, displacement = beam.getDisplacementDiagram()

In [16]: print(x[0:10])
[1.00000000e-05 5.00510480e-01 1.00101096e+00 1.50151144e+00
 2.00201192e+00 2.50251240e+00 3.00301288e+00 3.50351336e+00
 4.00401384e+00 4.50451432e+00]

In [17]: print(displacement[0:10])
[1.01361134e-25 8.34214859e-12 6.66013167e-11 2.24325385e-10
 5.30660630e-10 1.03435172e-09 1.78374174e-09 2.82677213e-09
 4.21098277e-09 5.98351191e-09]
```

ConcreteBeam Usage Example

How to create a beam:

```
In [1]: import fconcrete as fc

In [2]: n1 = fc.Node.SimpleSupport(x=0, length=20)

In [3]: n2 = fc.Node.SimpleSupport(x=400, length=20)
```

(continues on next page)

(continued from previous page)

```
In [4]: f1 = fc.Load.UniformDistributedLoad(-0.6, x_begin=0, x_
↪end=400)

In [5]: concrete_beam = fc.ConcreteBeam(
   ...:     loads = [f1],
   ...:     nodes = [n1, n2],
   ...:     section = fc.Rectangle(30,80),
   ...:     division = 200
   ...: )
   ...:
```

You can use all properties and methods of the *ConcreteBeam Class* including *Beam Class* such as plot shear diagram, momentum, etc. See examples in *Beam usage example*.

See general information:

```
In [6]: print("Cost of the concrete beam, in reais: ", concrete_
↪beam.cost)
Cost of the concrete beam, in reais:  522.3346059575427

In [7]: print("Processing time of the concrete beam, in seconds: ",␣
↪concrete_beam.processing_time)
Processing time of the concrete beam, in seconds:  0.
↪22832703590393066

In [8]: print(concrete_beam.cost_table)
[['Material' 'Price' 'Quantity' 'Unit' 'Commentary' 'Is Subtotal']
 ['Concrete' '339.17' '0.96' 'm3' 'Between 0.0m and 0.0m' 'False']
 ['Concrete' '339.17' '0.96' 'm3' '' 'True']
 ['Longitudinal bar' '56.78' '459.98' 'm'
  'Diameter 8.0mm. Between -56.68m and 456.68m' 'False']
 ['Longitudinal bar' '14.81' '359.8' 'm'
  'Diameter 8.0mm. Between -6.59m and 406.59m' 'False']
 ['Longitudinal bar' '13.16' '319.92' 'm'
  'Diameter 8.0mm. Between 13.35m and 386.65m' 'False']
 ['Longitudinal bar' '11.18' '271.68' 'm'
  'Diameter 8.0mm. Between 37.47m and 362.53m' 'False']
 ['Longitudinal bar' '8.53' '207.36' 'm'
  'Diameter 8.0mm. Between 69.64m and 330.36m' 'False']
 ['Longitudinal bar' '104.47' '1618.73' 'm' '' 'True']
 ['Transversal bar' '4.63' '225.0' 'm'
  '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 0.0m ' 'False']
 ['Transversal bar' '4.63' '225.0' 'm'
  '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 25.0m ' 'False']
 ['Transversal bar' '4.63' '225.0' 'm'
  '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 50.0m ' 'False']
 ['Transversal bar' '4.63' '225.0' 'm'
  '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 75.0m ' 'False']
 ['Transversal bar' '4.63' '225.0' 'm'
```

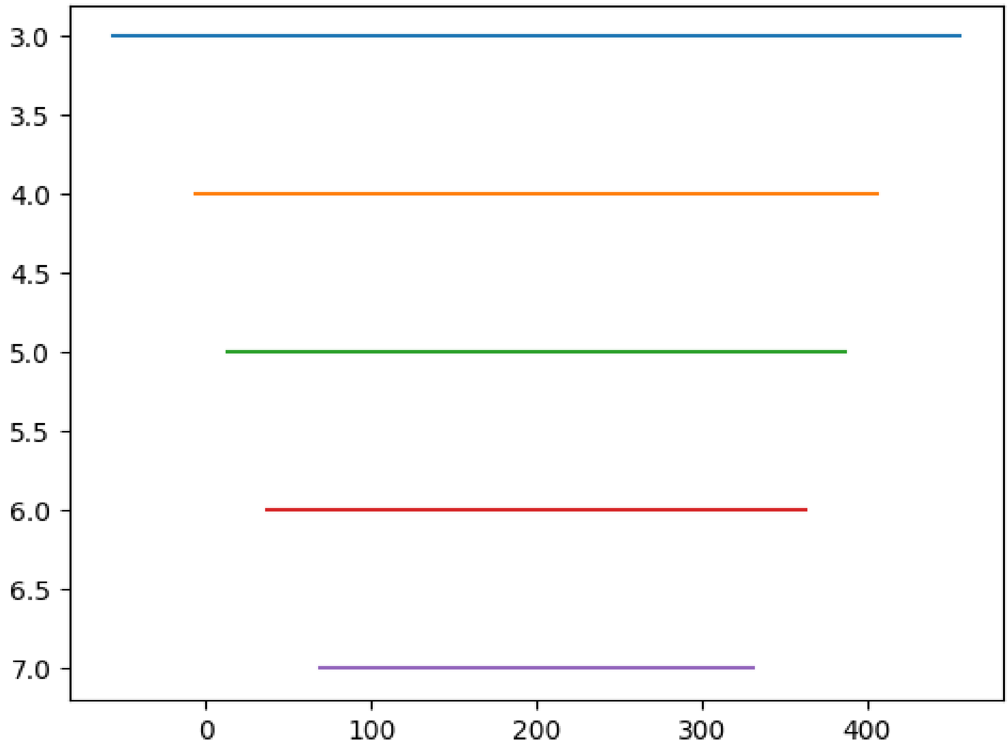
(continues on next page)

(continued from previous page)

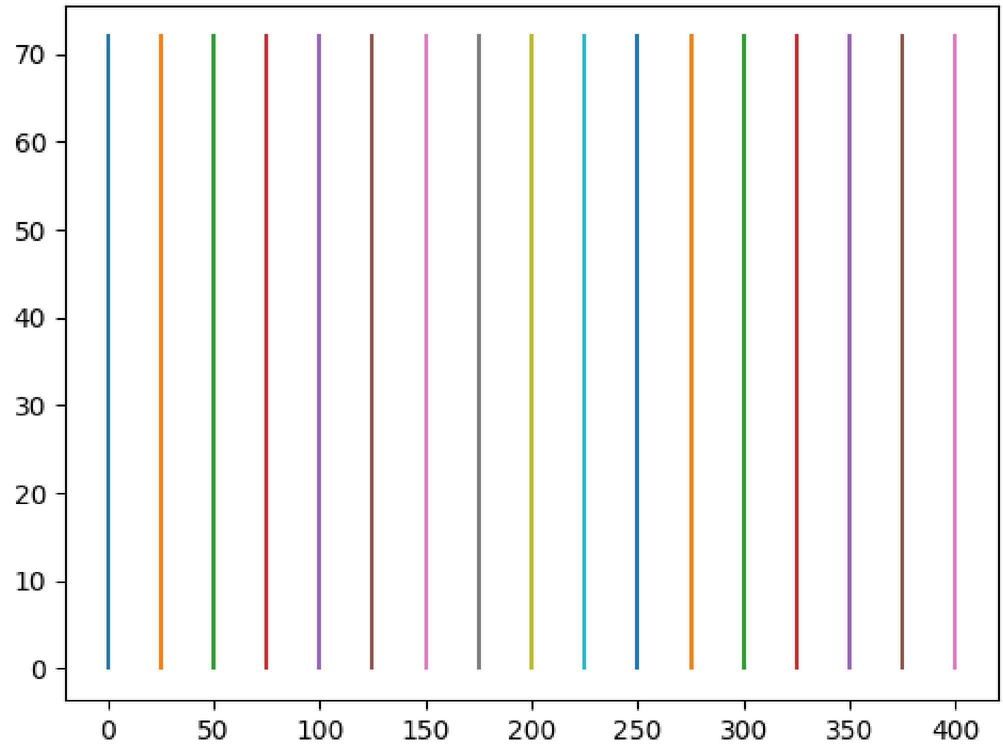
```
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 100.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 125.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 150.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 175.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 200.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 225.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 250.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 275.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 300.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 325.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 350.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 375.0m ' 'False']
['Transversal bar' '4.63' '225.0' 'm'
 '22.0cm x 72.0cm. Diameter 8.0mm. Placed in 400.0m ' 'False']
['Transversal bar' '78.7' '3825.0' 'm' '' 'True']]
```

Plot longitudinal informations:

```
# Longitudinal steel
In [9]: concrete_beam.long_steel_bars.plot(prop='area_accumulated')
```

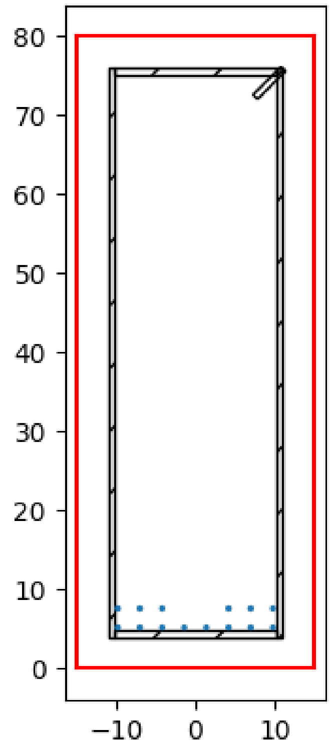



```
# Transversal steel
In [10]: concrete_beam.transv_steel_bars.plotLong()
Out[10]:
(<Figure size 640x480 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fab052e6470>)
```



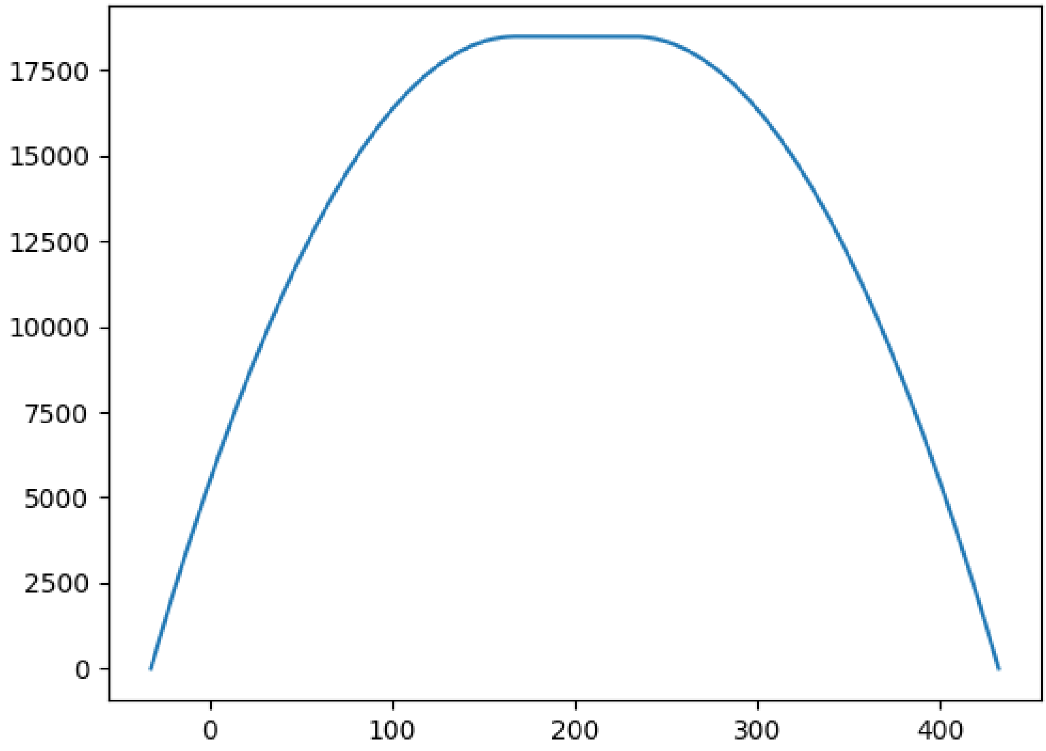
Plot transversal section:

```
In [11]: concrete_beam.plotTransversalInX(200)
Out[11]:
(<Figure size 640x480 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fab052c7e10>)
```



Also you can explore many informations related to the solution steps. Some examples:

```
In [12]: concrete_beam.long_steel_bars_solution_info.
↪plotDecalagedMomentumDesignDiagram()
Out[12]:
(<Figure size 640x480 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fab0542b2b0>)
```



Analysis Usage Example

First of all we have to create a function that we input the parameters that we want to make the analysis and return a concrete_beam.

More information about the ConcreteBeam class *click here*.

Let's see an example:

```
In [1]: def concrete_beam_function(width, height):
   ...:         n1 = fc.Node.SimpleSupport(x=0)
   ...:         n2 = fc.Node.SimpleSupport(x=200)
   ...:         pp = fc.Load.UniformDistributedLoad(-width*height*25/
↪1000000, x_begin=0, x_end=200)
   ...:         f1 = fc.Load.UniformDistributedLoad(-0.01, x_begin=0,
↪ x_end=1)
   ...:         beam = fc.ConcreteBeam(
   ...:             loads = [f1, pp],
   ...:             nodes = [n1, n2],
   ...:             section = fc.Rectangle(height,width),
   ...:             division = 200
   ...:         )
   ...:         return beam
   ...:
```

Now we can use the Analysis class to loop through the possibilities. In this example, we are going to set **avoid_estimate=True** and **show_progress=False** because it is a statical demonstration, but It is good practice to keep the default values.

The first argument is always the **function** that you created before, then you can set or disable some **optinal features** and finally you **must** provide values for the same inputs that are necessary on the concrete_beam_function **with the same name**. In this case, we have choosen width and height to change, so we can provide a list os possible values. See the example:

```
In [2]: import fconcrete as fc

In [3]: full_report, solution_report, best_solution = fc.Analysis.
↪getBestSolution(
   ...:                                          concrete_beam_function,
   ...:                                          max_steps_without_
↪decrease=15,
   ...:                                          sort_by_
↪multiplication=True,
   ...:                                          avoid_estimate=True,
   ...:                                          show_progress=False,
   ...:                                          width=[15, 17, 19],
   ...:                                          height=[30, 34, 38])
   ...:
```

Instead of providing a list such as width=[15, 17, 19], you can also provide a tuple like that: **(start, end_but_not_included, steps)**. It is going to create a list for you. Both ways have the same effect:

```
width = (15, 31, 2)
width = [15, 17, 19, 21, 23, 25, 27, 29]
```

Once the reports are created, we can see its information:

```
In [4]: full_report
Out[4]:
0 width height     cost error Concrete Longitudinal bar Transversal␣
↪bar
1     15      30   66.8154            31.8            17.43           ␣
↪17.59
2     15      34   72.5364           36.04            17.43           ␣
↪19.07
3     17      30   72.0428           36.04            17.67           ␣
↪18.33
4     15      38   82.8251           40.28            17.43           ␣
↪25.12
5     19      30   77.2703           40.28            17.92           ␣
↪19.07
6     17      34   78.3291           40.84            17.67           ␣
↪19.81
7     17      38   89.3477           45.65            17.67           ␣
↪26.03
```

(continues on next page)

(continued from previous page)

```
8     19      34   84.1218          45.65            17.92              ␣
↪20.55
9     19      38   95.8702          51.02            17.92              ␣
↪26.93

# We can see that the error column just have empty strings, so in␣
↪this case no errors of combinations were found.
# The solution (only without the errors) table is sorted by cost␣
↪ascending, so the first one is the most economic solution.
```

A alternative way to see the beast beam and its properties:

```
In [5]: best_solution
Out[5]:
{'width': 15.0,
 'height': 30.0,
 'cost': 66.8154415742161,
 'error': '',
 'Concrete': 31.8,
 'Longitudinal bar': 17.43,
 'Transversal bar': 17.59}
```

The first values are the parameters to be analysed and the last columns are 'cost' (total cost) and the cost for the 3 elements: 'Concrete', 'Longitudinal bar' and 'Transversal bar'.

1.5.3 fconcrete

fconcrete package

Subpackages

fconcrete.Structural package

Submodules

fconcrete.Structural.Beam module

class `fconcrete.Structural.Beam.`**`Beam`**(*loads*, *beam_elements*, ***options*)

Bases: `object`

Structural Beam.

Note for this documentation: "Not only the ones provided by the initial beam_Elements" means that internally the Beam automatically creates some nodes even if it was not created for the user initially. It happens in Load.x_begin and Load.x_end.

Attributes

U: list of number Displacement in the nodes.

beam_elements: BeamElements BeamElements instance of beam_elements, not only the ones provided by the initial beam_Elements.

beams_quantity: number Number of beam_elements, not only the ones provided by the initial beam_Elements.

external_loads: Loads loads argument but used as a Loads class. Same as fconcrete.Structural.Load.Loads.create(loads).

initial_beam_elements: array of BeamElement beam_elements argument used when the instance is created.

length: number Length of the beam. Can also use len(beam).

loads: Loads Loads instance with all efforts in the beam. Including the load given by the supports.

nodal_efforts: list of number The nodal efforts that happens in all nodes, not only the ones provided by the initial beam_Elements.

nodes: Nodes Nodes instance of the beam, not only the ones provided by the initial beam_Elements.

x_begin: number Where the beam starts, in cm.

x_end: number Where the beam ends, in cm.

Methods

`copy`(self)	Makes a deep copy of the instance of Beam.
`getBeamElementInX`(self, x)	Get the beam element in x (in cm).
`getDisplacement`(self, x)	Get the vertical displacement in a position x (in cm) or multiple positions.
`getDisplacementDiagram`(self, **options)	Apply beam.getDisplacement for options["division"] parts of the beam.
`getInternalMomentumStrength`(self, x)	Get the internal momentum strength in a position x (in cm) or multiple positions.
`getInternalShearStrength`(self, x)	Get the internal shear strength in a position x (in cm) or multiple positions.
`getMomentumDiagram`(self, **options)	Apply beam.getInternalMomentumStrength for options["division"] parts of the beam.
`getRotation`(self, x)	Get the rotation in a position x (in cm) or multiple positions.
`getRotationDiagram`(self, **options)	Apply beam.getRotation for options["division"] parts of the beam.

Continued on next page

Table 1 – continued from previous page

`getShearDiagram`(self, **options)	Apply beam.getInternalShearStrength for options["division"] parts of the beam.
`matrix_rigidity_global`(self)	Returns the global rigidity matrix.
`plotDisplacementDiagram`(self, **options)	Simply applies the beam.getDisplacementDiagram method results (x,y) to a plot with plt.plot(x, y).
`plotMomentumDiagram`(self, **options)	Simply applies the beam.getMomentumDiagram method results (x,y) to a plot with plt.plot(x, y).
`plotRotationDiagram`(self, **options)	Simply applies the beam.getRotationDiagram method results (x,y) to a plot with plt.plot(x, y).
`plotShearDiagram`(self, **options)	Simply applies the beam.getShearDiagram method results (x,y) to a plot with plt.plot(x, y).
`solve_displacement`(self)	Starts the process of solution for the structural beam displacement.
`solve_structural`(self)	Starts the process of solution for the structural beam.

copy (*self*)
Makes a deep copy of the instance of Beam.

getBeamElementInX (*self*, *x*)
Get the beam element in x (in cm).

Call signatures:

beam.getBeamElementInX(x)

Parameters

x [`number`] Position in the beam, in cm.

Returns

index [`python:int`] The order of the beam_element in the structure.

beam_element beam_element located in x.

getDisplacement (*self*, *x*)
Get the vertical displacement in a position x (in cm) or multiple positions.

Call signatures:

beam.getDisplacement(x)

Parameters

x [`number` or `python:list of number`] Position in the beam, in cm.

Returns

displacement [`number` or `python:list of number`] The vertical displacement of the beam in cm.

getDisplacementDiagram(*self*, ***options*)

Apply beam.getDisplacement for options["division"] parts of the beam.

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

Returns

x [`python:list of number`] The x position of the division in cm

y [`python:list of number`] The value of displacement for each x.

getInternalMomentumStrength(*self*, *x*)

Get the internal momentum strength in a position x (in cm) or multiple positions.

Call signatures:

beam.getInternalMomentumStrength(x)

Parameters

x [`number` or `python:list of number`] Position in the beam, in cm.

Returns

x [`python:list of number`] The x position of the division in cm

momentum [`number` or `python:list of number`] The internal value of the momentum strength in kNcm.

getInternalShearStrength(*self*, *x*)

Get the internal shear strength in a position x (in cm) or multiple positions.

Call signatures:

beam.getInternalShearStrength(x)

Parameters

x [`number` or `python:list of number`] Position in the beam, in cm.

Returns

shear [`number` or `python:list of number`] The internal value of the shear strength in kN.

`getMomentumDiagram`(*self*, ***options*)

Apply beam.getInternalMomentumStrength for options[“division”] parts of the beam.

Parameters

****options**

`division:` Number of divisions equally spaced (*int*).

`x_begin:` Begin of the x_axis (*number*).

`x_end:` End of the x_axis (*number*).

Returns

x [`python:list of number`] The x position of the division in cm

y [`python:list of number`] The value of momentum for each x.

`getRotation`(*self*, *x*)

Get the rotation in a position x (in cm) or multiple positions.

Call signatures:

beam.getRotation(x)

Parameters

x [`number` or `python:list of number`] Position in the beam, in cm.

Returns

rotation [`number` or `python:list of number`] The rotation value of the momentum strength in rad.

`getRotationDiagram`(*self*, ***options*)

Apply beam.getRotation for options[“division”] parts of the beam.

Parameters

****options**

`division:` Number of divisions equally spaced (*int*).

`x_begin:` Begin of the x_axis (*number*).

`x_end:` End of the x_axis (*number*).

Returns

x [`python:list of number`] The x position of the division in cm

y [`python:list of number`] The value of rotation for each x.

getShearDiagram(*self*, ***options*)

Apply beam.getInternalShearStrength for options["division"] parts of the beam.

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

Returns

x [`python:list of number`] The x position of the division in cm

y [`python:list of number`] The value of shear for each x.

matrix_rigidity_global(*self*)

Returns the global rigidity matrix. Also known by the letter "K".

plotDisplacementDiagram(*self*, ***options*)

Simply applies the beam.getDisplacementDiagram method results (x,y) to a plot with plt.plot(x, y).

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

plotMomentumDiagram(*self*, ***options*)

Simply applies the beam.getMomentumDiagram method results (x,y) to a plot with plt.plot(x, y).

Also invert y axis.

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

plotRotationDiagram(*self*, ***options*)

Simply applies the beam.getRotationDiagram method results (x,y) to a plot with plt.plot(x, y).

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

plotShearDiagram(*self*, ***options*)

Simply applies the beam.getShearDiagram method results (x,y) to a plot with plt.plot(x, y).

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

solve_displacement(*self*)

Starts the process of solution for the structural beam displacement.

solve_structural(*self*)

Starts the process of solution for the structural beam.

fconcrete.Structural.BeamElement module

class fconcrete.Structural.BeamElement.**BeamElement**(*nodes*, *section=<fconcrete.Structural.Secti... object>*, *material=<fconcrete.Structural.Mate... object>*)

Bases: `object`

Class that defines a primitive elements of a beam.

Methods

get_efforts_from_bar_element(beam_element, load)	Get the efforts coused by the load in a double crimped beam element.
get_matrix_rigidity_unitary(self)	Returns the unitary rigidity matrix.
split(self, x)	Split a beam_element in two.

classmethod get_efforts_from_bar_element(*beam_element*, *load*)

Get the efforts coused by the load in a double crimped beam element.

Parameters

distance_a [`number`] Distance, in cm, from the left node to the force.

get_matrix_rigidity_unitary(*self*)
Returns the unitary rigidity matrix.

split(*self*, *x*)
Split a beam_element in two. The node in x is considered a Middle Node.

Parameters

x [number] Distance, in cm, from the left node to the split point.

class fconcrete.Structural.BeamElement.**BeamElements**(*bar_elements*)
Bases: object

Class that defines a primitive elements of a beam list with easy to work properties and methods.

Methods

changeProperty(self, prop, function[, . . .])	Change all properties of the beam elements in a single function.
create(beam_elements)	Recommended way to create a BeamElements class.
split(self, x)	Similar to BeamElement.split, but can guess what element of the array is going to be splited.

changeProperty(*self*, *prop*, *function*, *conditional=<function BeamElements.<lambda> at 0x7fab0fe697b8>*)
Change all properties of the beam elements in a single function.

classmethod create(*beam_elements*)
Recommended way to create a BeamElements class.

split(*self*, *x*)
Similar to BeamElement.split, but can guess what element of the array is going to be splited.

fconcrete.Structural.Load module

class fconcrete.Structural.Load.**Load**(*force*, *momentum*, *x_begin*, *x_end*, *q=0*, *order=0*, *displacement=0*)
Bases: object

Class that defines a load.

Methods

PontualLoad(load, x)	Define a pontual load.
UniformDistributedLoad(q, x_begin, x_end)	Define a uniform and distributed load.

classmethod PontualLoad(*load*, *x*)

Define a pontual load.

Call signatures:

fc.PontualLoad(load, x)

```
>>> pontual_load_1 = fc.Load.PontualLoad(-10.0, 200)
>>> pontual_load_2 = fc.Load.PontualLoad('-10.0kN',
↪'2m')
>>> repr(pontual_load_1) == repr(pontual_load_2)
True
```

Parameters

load [number or python:str] Represent the load measure. If it is a number, default unit is kN, but also [force] unit can be given. Example: '20kN', '10N', etc

x [number or python:str] Where the load is going to end. If it is a number, default unit is cm, but also [length] unit can be given. Example: '20cm', '10dm', etc

classmethod UniformDistributedLoad(*q*, *x_begin*, *x_end*)

Define a uniform and distributed load.

Call signatures:

fc.UniformDistributedLoad(q, x_begin, x_end)

```
>>> uniform_load_1 = fc.Load.
↪UniformDistributedLoad(0.1, 0, 2000)
>>> uniform_load_2 = fc.Load.UniformDistributedLoad(
↪'10.0kN/m', '0m', '20m')
>>> repr(uniform_load_1) == repr(uniform_load_2)
True
```

Parameters

q [number or python:str] Represent the load by length measure. If it is a number, default unit is kN/cm, but also [force]/[length] unit can be given. Example: '20kN/m', '10N/m', etc

x_begin [number or python:str] Where the load is going to start. If it is a number, default unit is cm, but also [length] unit can be given. Example: ‘20cm’, ‘10dm’, etc

x_end [`number` or `python:str`] Where the load is going to end. If it is a number, default unit is cm, but also [length] unit can be given. Example: ‘20cm’, ‘10dm’, etc

class `fconcrete.Structural.Load.`**`Loads`**(*loads*)
Bases: `object`

Class that defines a load list with easy to work properties and methods.

Methods

`add`(self, loads)	Add a array of Load in the Loads instance.
`create`(loads)	Creates a instance of Loads with array of Load.

`add`(*self*, *loads*)
Add a array of Load in the Loads instance.

classmethod `create`(*loads*)
Creates a instance of Loads with array of Load.

fconcrete.Structural.Material module

class `fconcrete.Structural.Material.`**`Material`**(*E*, *poisson*, *alpha*)
Bases: `object`

Define the class for the material.

Attributes

E [`number`] Represent the Young Modulus (E) in kN/cm^2.

poisson [`number`] Poisson’s ratio is a measure of the Poisson effect, that describes the expansion of a material in directions perpendicular to the direction of compression.

alpha [`number`] Coefficient of thermal expansion which is the relative expansion (also called strain) divided by the change in temperature.

fconcrete.Structural.Node module

class `fconcrete.Structural.Node.`**`Node`**(*x*, *condition_boundary*, *length=0*)
Bases: `object`

Methods

`Crimp`(x[, length])	Represents a node with vertical displacement and rotation equal to zero.
`Free`(x)	Represents a node with vertical displacement and rotation.
`MiddleNode`(x)	Represents a node with vertical displacement and rotation.
`SimpleSupport`(x[, length])	Represents a node with vertical displacement equal to zero.

classmethod Crimp (*x*, *length=0*)

Represents a node with vertical displacement and rotation equal to zero.

Call signatures:

fc.Node.Crimp(x)

```
>>> crimp_node_1 = fc.Node.Crimp(100)
>>> crimp_node_2 = fc.Node.Crimp('1m')
>>> repr(crimp_node_1) == repr(crimp_node_2)
True
```

Parameters

x [`number` or `python:str`] Position of the node. If it is a number, default unit is cm, but also [length] unit can be given. Example: '20m', '10dm', etc

length [`number` or `python:str`, optional] Length of the node if applicable. If it is a number, default unit is cm, but also [length] unit can be given. Example: '20m', '10dm', etc. Default is 0.

classmethod Free (*x*)

Represents a node with vertical displacement and rotation.

Call signatures:

fc.Node.Free(x)

```
>>> free_node_1 = fc.Node.Free(100)
>>> free_node_2 = fc.Node.Free('1m')
>>> repr(free_node_1) == repr(free_node_2)
True
```

Parameters

x [`number` or `python:str`] Position of the node. If it is a number, default unit is cm, but also [length] unit can be given. Example: '20m', '10dm', etc

classmethod MiddleNode(*x*)

Represents a node with vertical displacement and rotation.

Call signatures:

fc.Node.Free(x)

```
>>> middle_node_1 = fc.Node.MiddleNode(100)
>>> middle_node_2 = fc.Node.MiddleNode('1m')
>>> repr(middle_node_1) == repr(middle_node_2)
True
```

Parameters

x [`number` or `python:str`] Position of the node. If it is a number, default unit is cm, but also [length] unit can be given. Example: '20m', '10dm', etc

classmethod SimpleSupport(*x*, *length=0*)

Represents a node with vertical displacement equal to zero. But it allows rotation.

Call signatures:

fc.Node.SimpleSupport(x, length=0)

```
>>> simple_support_1 = fc.Node.SimpleSupport(100)
>>> simple_support_2 = fc.Node.SimpleSupport('1m')
>>> repr(simple_support_1) == repr(simple_support_2)
True
```

Parameters

x [`number` or `python:str`] Position of the node. If it is a number, default unit is cm, but also [length] unit can be given. Example: '20m', '10dm', etc

length [`number` or `python:str`, optional] Length of the node if applicable. If it is a number, default unit is cm, but also [length] unit can be given. Example: '20m', '10dm', etc. Default is 0.

class `fconcrete.Structural.Node.`**Nodes**(*nodes*)

Bases: `object`

fconcrete.Structural.Section module

class `fconcrete.Structural.Section.`**Rectangle**(*width*, *height*)

Bases: *fconcrete.Structural.Section.Section*

Attributes

height [`number`] Maximum height of the section in cm.

function_width [`function`] Define the width along the y axis. The function starts with x=0 and ends in x=height.

bw [`number`] Minimum width in cm.

area [`number`] Total area of the section in cm^2.

I [`number`] Moment of inertia in cm^4.

y_cg [`number`] Gravity center in the y axis.

x0 [`number`] Initial reference in the x axis.

y0 [`number`] Initial reference in the y axis.

Methods

`getAreaBetween`(self, begin_height, end_height)	Area between 2 y values.
`plot`(self[, N, color_plot, ax, fig])	Plot the section.
`width`(self[, height])	Width value in cm.

`getAreaBetween` (*self*, *begin_height*, *end_height*)
Area between 2 y values.

`width` (*self*, *height=0*)
Width value in cm.

`class` `fconcrete.Structural.Section.`**`Section`** (*function_width*, *height*)

Bases: `object`

Class to represent simetrical section along the y axis.

Attributes

height [`number`] Maximum height of the section in cm.

function_width [`function`] Define the width along the y axis. The function starts with x=0 and ends in x=height.

area [`number`] Total area of the section in cm^2.

I [`number`] Moment of inertia in cm^4.

x0 [`number`] Initial reference in the x axis.

y0 [`number`] Initial reference in the y axis.

Methods

`getAreaBetween`(self, begin_height, end_height)	Area between 2 y values.
`plot`(self[, N, color_plot, ax, fig])	Plot the section.
`width`(self, y)	Gets the width in y.

getAreaBetween (*self*, *begin_height*, *end_height*, *interations=100*)
Area between 2 y values.

plot (*self*, *N=100*, *color_plot='red'*, *ax=None*, *fig=None*)
Plot the section.

width (*self*, *y*)
Gets the width in y.

Module contents

Code for structural calculus. Not related to any specific material. Uses FEM (Finite Element Method) to define the efforts.

fconcrete.StructuralConcrete package

Subpackages

fconcrete.StructuralConcrete.LongSteelBar package

Submodules

fconcrete.StructuralConcrete.LongSteelBar.LongSteelBar module

class `fconcrete.StructuralConcrete.LongSteelBar.LongSteelBar.`**`LongSteelBar`**

Bases: `object`

Methods

getMinimumAndMaximumSteelArea(area, fck)	Giving the fck in kN/cm^2, returns the minimum and maximum area.
getSteelArea(section, material, steel, momentum)	Giving the section, material, type of steel and momentum, this funciton calculates the necessary steel area.
plot(self[, prop])	Plot the Long Steel Bar giving the property.

static getMinimumAndMaximumSteelArea(*area*, *fck*)
Giving the fck in kN/cm^2, returns the minimum and maximum area.

static getSteelArea(*section*, *material*, *steel*, *momentum*)
Giving the section, material, type of steel and momentum, this funciton calculates the necessary steel area.

plot(*self*, *prop='area_accumulated'*)
Plot the Long Steel Bar giving the property.

class `fconcrete.StructuralConcrete.LongSteelBar.LongSteelBar.`**`LongSteelBar`**
Bases: `object`

Class that defines a LongSteelBar list with easy to work properties and methods.

Methods

add(self, new_steel_bars)	Add a LongSteelBar to the LongSteelBars instance.
changeProperty(self, prop, function[, ...])	Change all properties of the LongSteelBar in a single function.
getBarTransversalPosition(self, concrete_beam, x)	Get the bars in a x transversal position, in cm.
getPositiveandNegativeLongSteelBarsInX(self, x)	Get the bars in a x longitudinal position, in cm.
plot(self[, prop])	Plot the lonfitudinal vision of the longitudinal bars.
plotTransversal(self, concrete_beam, x[, ...])	Plot the transversal vision of the longitudinal bars.

add (*self*, *new_steel_bars*)
Add a LongSteelBar to the LongSteelBars instance.

changeProperty (*self*, *prop*, *function*, *conditional=<function LongSteelBars.<lambda> at 0x7fab0fd87488>*)
Change all properties of the LongSteelBar in a single function.

getBarTransversalPosition (*self*, *concrete_beam*, *x*)
Get the bars in a x transversal position, in cm.

Returns

transversal_positions [array] Each array array contains the x, y position in the transversal section, the radius of the bar and its area: x, y, radius, area.

getPositiveandNegativeLongSteelBarsInX (*self*, *x*)
Get the bars in a x longitudinal position, in cm.

Returns

positive_steel_bar_in_x [LongSteelBars] The positive steel bar found in x.

negative_steel_bar_in_x [LongSteelBars] The negative steel bar found in x.

plot (*self*, *prop='area_accumulated'*)
Plot the lonfitudinal vision of the longitudinal bars.

plotTransversal (*self*, *concrete_beam*, *x*, *ax=None*, *fig=None*, *color_plot='red'*)
Plot the transversal vision of the longitudinal bars.

fconcrete.StructuralConcrete.LongSteelBar.LongSteelBarSolve module

class fconcrete.StructuralConcrete.LongSteelBar.LongSteelBarSolve.**LongSte**
Bases: object

Methods

getComercialSteelArea(self, x, momentum)	Returns comercial steel area given the position and momentum.
getComercialSteelAreaDiagram(self, ...)	Returns comercial steel area diagram.
getDecalagedLength(self, beam_element)	Returns decalaged length of a beam element.
getDecalagedMomentumDesignDiagram(self, ...)	Return tuple with 3 np.array: x (axis), momentum_positive, momentum_negative.
getMinimumAndMaximumSteelArea(self, x)	Returns tuple of minimum and maximum necessary steel area given the position.
getSteelArea(self, x, momentum)	#only working with rectangle section Returns necessary steel area given the position and momentum.
getSteelAreaDiagram(self, **options_diagram)	Returns necessary steel area diagram.
plotDecalagedMomentumDesignDiagram(self[, ...])	Plot DecalagedMomentumDesignDiagram.

getComercialSteelArea (*self, x, momentum*)

Returns comercial steel area given the position and momentum. Implements: minimum steel area, check maximum steel area and do not allow a single steel bar. Does not have the removal by step implemented here. Not recommended to use in loops.

Call signatures:

concrete_beam.long_steel_bars_solution_info.getComercialSteelArea(x, momentum)

```
>>> concrete_beam.long_steel_bars_solution_info.
↪getComercialSteelArea(300, 2500)
(6.0, 0.8, 3.0)
```

Parameters

x [number] Define the position in cm.

momentum [number] Define the momentum in kNcm.

getComercialSteelAreaDiagram (*self, **options_diagram*)

Returns comercial steel area diagram. Implements: minimum steel area, check maximum steel area and do not allow a single steel bar. Does not have the removal by step implemented here.

Call signatures:

concrete_beam.long_steel_bars_solution_info.getComercialSteelAreaDiagram(division=

```
>>> x_decalaged, positive_areas_info, negative_areas_
↪info = concrete_beam.long_steel_bars_solution_info.
↪getComercialSteelAreaDiagram()
>>> x_decalaged, positive_areas_info, negative_areas_
↪info = concrete_beam.long_steel_bars_solution_info.
↪getComercialSteelAreaDiagram(division=5000)
```

getDecalagedLength (*self*, *beam_element*)

Returns decalaged length of a beam element.

Call signatures:

concrete_beam.long_steel_bars_solution_info.getDecalagedLength(beam_element)

getDecalagedMomentumDesignDiagram (*self*, ***options_diagram*)

Returns tuple with 3 np.array: x (axis), momentum_positive, momentum_negative.

Call signatures:

concrete_beam.long_steel_bars_solution_info.getDecalagedMomentumDesignDiagram(d

```
>>> x_decalaged, momentum_positive, momentum_
↪negative = concrete_beam.long_steel_bars_solution_
↪info.
↪getDecalagedMomentumDesignDiagram(division=100)
```

Parameters

division [`python:int`, optional (default 1000)] Define the step to plot the graph. A high number means a more precise graph, but also you need more processing time.

getMinimumAndMaximumSteelArea (*self*, *x*)

Returns tuple of minimum and maximum necessary steel area given the position.

Call signatures:

concrete_beam.long_steel_bars_solution_info.getMinimumAndMaximumSteelArea(x)

```
>>> concrete_beam.long_steel_bars_solution_info.
↪getMinimumAndMaximumSteelArea(300)
(2.76, 19.2)
```

Parameters

x [`number`] Define the position in cm.

getSteelArea (*self*, *x*, *momentum*)

#only working with rectangle section Returns necessary steel area given the position and momentum.

Call signatures:

concrete_beam.long_steel_bars_solution_info.getSteelArea(x, momentum)

```
>>> concrete_beam.long_steel_bars_solution_info.
↪getSteelArea(10, 2500)
0.903512040037519
```

Parameters

x [number] Define the position in cm.

momentum [number] Define the momentum in kNcm.

getSteelAreaDiagram (*self*, ***options_diagram*)

Returns necessary steel area diagram.

Call signatures:

concrete_beam.long_steel_bars_solution_info.getSteelAreaDiagram(division=1000)

```
>>> x_decalaged, positive_areas, negative_areas = ␣
↪concrete_beam.long_steel_bars_solution_info.
↪getSteelAreaDiagram()
>>> x_decalaged, positive_areas, negative_areas = ␣
↪concrete_beam.long_steel_bars_solution_info.
↪getSteelAreaDiagram(division=20)
```

plotDecalagedMomentumDesignDiagram (*self*, *ax=None*, *fig=None*, ***options_diagram*)

Plot DecalagedMomentumDesignDiagram.

Module contents

fconcrete.StructuralConcrete.TransvSteelBar package

Submodules

fconcrete.StructuralConcrete.TransvSteelBar.TransvSteelBar module

class fconcrete.StructuralConcrete.TransvSteelBar.TransvSteelBar.**TransvSt**

Bases: object

Methods

plot(self[, c, ax, fig, color_plot])	Plot the transversal vision of the transversal bar.

plot (*self*, *c=2*, *ax=None*, *fig=None*, *color_plot='blue'*)
Plot the transversal vision of the transversal bar.

class fconcrete.StructuralConcrete.TransvSteelBar.TransvSteelBar.**TransvSt**
Bases: object

Class that defines a the TransvSteelBar list with easy to work properties and methods.

Methods

add(self, new_steel_bars)	Add a array of Load in the Loads instance.
changeProperty(self, prop, function[, ...])	Change all properties of the TransvSteelBar in a single function.
getTransversalBarAfterX(self, x)	Get the next transversal bar in x or after.
plotLong(self)	Plot the longitudinal vision of the transversal bar.

add (*self*, *new_steel_bars*)
Add a array of Load in the Loads instance.

changeProperty (*self*, *prop*, *function*, *conditional=<function TransvSteelBars.<lambda> at 0x7fab0fd7b730>*)

Change all properties of the TransvSteelBar in a single function.

getTransversalBarAfterX (*self*, *x*)
Get the next transversal bar in x or after.

plotLong (*self*)
Plot the longitudinal vision of the transversal bar.

fconcrete.StructuralConcrete.TransvSteelBar.TransvSteelBarSolve module

class fconcrete.StructuralConcrete.TransvSteelBar.TransvSteelBarSolve.**Tra**

Bases: object

Methods

checkProbableCompressedConnectingRod(self)	Check probable compressed connecting rod.
getComercialInfo(self, as_per_cm)	Get comercial info giving the area per cm.
getMinimumSteelAreaPerCm(self, …)	Giving a beam element, calculates the minimum steel area (cm^2) per cm.
getShearSteelAreaPerCm(self, x, v_sd)	Calculates the shear steel area (cm^2) per cm considering the restrictions.
getShearSteelAreaPerCmDiagram(self)	Apply concrete_beam.transv_steel_bars_solution_info.getShearSteelAr for parts of the concrete_beam.
getStirrupsInfo(self)	Format all informations and return a TransvSteelBars instance.
getV_rd2(self, single_beam_element)	Giving a beam element, calculates the shear related to the ruin of compressed concrete diagonals in kN.

checkProbableCompressedConnectingRod (*self*)
Check probable compressed connecting rod. It is probable because checks only where the shear is maximum.

Call signatures:

concrete_beam.transv_steel_bars_solution_info.checkProbableCompressedConnectingRo

Returns

v_rd2 [number] Shear of calculation, related to the ruin of compressed concrete diagonals in kN.

d [`number`] Distance from longitudinal steel bars to the other extremity of the section in cm.

max_shear [`number`] Maximum shear in kN.

getComercialInfo(*self*, *as_per_cm*)
Get comercial info giving the area per cm.

Returns

diameter [`number`] Diameter in cm.

space [`number`] Longitudinal space between the transversal steel.

area [`number`] Area of the transversal steel bar in cm^2.

as_per_cm [`number`] Area of the transversal steel bar in cm^2 per cm.

getMinimumSteelAreaPerCm(*self*, *single_beam_element*)
Giving a beam element, calculates the minimum steel area (cm^2) per cm.

getShearSteelAreaPerCm(*self*, *x*, *v_sd*)
Calculates the shear steel area (cm^2) per cm considering the restrictions.

getShearSteelAreaPerCmDiagram(*self*)
Apply concrete_beam.transv_steel_bars_solution_info.getShearSteelAreaPerCm for parts of the concrete_beam.

Returns

x [`python:list of number`] The x position of the division in cm

y [`python:list of number`] The value of shear area per cm for each x.

getStirrupsInfo(*self*)
Format all informations and return a TransvSteelBars instance.

getV_rd2(*self*, *single_beam_element*)
Giving a beam element, calculates the shear related to the ruin of compressed concrete diagonals in kN.

Module contents

Submodules

fconcrete.StructuralConcrete.Analysis module

class `fconcrete.StructuralConcrete.Analysis.`**Analysis**
Bases: `object`

Methods

`getBestSolution`(concrete_beam_function[, ...])	Returns a report with all materials and cost.

static getBestSolution(*concrete_beam_function*, *max_steps_without_decrease=inf*, *avoid_estimate=False*, *show_progress=True*, *sort_by_multiplication=False*, ***kwargs*)

Returns a report with all materials and cost.

Call signatures:

*fc.Analysis.getBestSolution(concrete_beam_function, ... max_steps_without_decrease = float("inf"), ... avoid_estimate=False, ... show_progress=True, ... sort_by_multiplication=False, ... **kwargs)*

```
>>> def concrete_beam_function(width, height,␣
↪length):
...         slab_area = 5*5
...         kn_per_m2 = 5
...         distributed_load = -slab_area*kn_per_m2/
↪500
...         pp = fc.Load.UniformDistributedLoad(-
↪width*height*25/1000000, x_begin=0, x_end=length)
...         n1 = fc.Node.SimpleSupport(x=0, length=20)
...         n2 = fc.Node.SimpleSupport(x=400,␣
↪length=20)
...         f1 = fc.Load.UniformDistributedLoad(-0.01,
↪ x_begin=0, x_end=1)
...         beam = fc.ConcreteBeam(
...             loads = [f1, pp],
...             nodes = [n1, n2],
...             section = fc.Rectangle(width, height),
...             division = 200
...         )
...         return beam
>>> full_report, solution_report, best_solution = fc.
↪Analysis.getBestSolution(concrete_beam_function,
...                                         max_steps_
↪without_decrease=15,
...                                         sort_by_
↪multiplication=True,
...                                         avoid_
↪estimate=True,
...                                         show_
↪progress=False,
...                                         width=[15],
...                                         height=(30,␣
↪34, 2),
```

(continues on next page)

(continued from previous page)

```
...                                                length=[150])
>>> # Table is sorted by cost ascending, so the␣
↪first one is the most economic solution.
>>> # Alternative way to look to the best solution
>> print(best_solution)
{'width': 15.0, 'height': 30.0, 'length': 150.0,
↪'cost': 126.2650347902965, 'error': '', 'Concrete
↪': 63.59, 'Longitudinal bar': 35.31, 'Transversal␣
↪bar': 27.36}
```

Parameters

concrete_beam_function Define the function that is going to create the beam given the parameters.

max_steps_without_decrease [python:int, optional] If the cost has not decrescead after max_steps_without_decrease steps, the loop breaks. Only use it in case your parameter combination has a logical order. Default inf.

show_progress [*bool*, optional] Estimate time using the last combination. If a exception is found, 80s per loop is set and a message about the not precision is shown. Also show progress bar in percentage. Default True.

sort_by_multiplication [*bool*, optional] Sort combinations by the multiplication os all parameter. Useful to use with max_steps_without_decrease when the is a logical order. Default False.

kwargs Possible arguments for the concrete_beam_function. If a set of 3 elements is given, np.arange(*kwarg_value) will be called. The kwargs must have the same name that the concrete_beam_function expects as arguments. The combination is made with np.meshgrid.

fconcrete.StructuralConcrete.AvailableMaterials module

class fconcrete.StructuralConcrete.AvailableMaterials.**AvailableConcrete**(*fc*
c
a
g
s
n
a
g
g
b

Bases: object

Define the available concrete. You can set the available fck, cost_by_m3, aggressiveness and aggregate. See more information in fc.AvailableConcrete docstring. For example, AvailableConcrete() means:

- 30 MPa;
- R$353.30 by meter^3;
- The aggressiveness is 3;
- Aggregate is granite;
- Biggest aggregate dimension is 1.5cm.

class fconcrete.StructuralConcrete.AvailableMaterials.**AvailableLongConcre**

Bases: `object`

Define the available longitudinal steel bars. You can set the available diameters, cost_by_meter, fyw, E, etc. See more information in fc.AvailableLongConcreteSteelBar() docstring. For example, AvailableLongConcreteSteelBar([8]) means:

- 8mm diameter;
- 0.5cm^2 area;
- R$2.0575 by meter cost;
- fyw equal to 50kN/cm^2;
- Young Modulus (E) is 21000kN/cm^2;
- Max number of steel in the section is 200;
- Surface type is ribbed.

`class fconcrete.StructuralConcrete.AvailableMaterials.AvailableTransvConc`

Bases: `object`

Define the available transversal steel bars. You can set the available diameters, cost_by_meter, fyw, E, etc. See more information in fc.AvailableTransvConcreteSteelBar docstring. Default AvailableTransvConcreteSteelBar([8]) which means:

- 8mm diameter;
- 0.5cm^2 area;
- R$2.0575 by meter cost;
- The longitudinal space between transversal steel are multiple of 5;
- fyw equal to 50kN/cm^2;
- Transversal bar inclination angle of 90 degrees;
- Tilt angle of compression struts of 45 degrees.

`fconcrete.StructuralConcrete.AvailableMaterials.`**`solve_cost`**(*concrete_beam*, *decimal_numbers=2*)

fconcrete.StructuralConcrete.Concrete module

class `fconcrete.StructuralConcrete.Concrete.`**`Concrete`**(*fck*, *aggressiveness*, *aggregate='granite'*)

Bases: *`fconcrete.Structural.Material.Material`*

Define the Concrete to be used and all its properties.

Attributes

fck [`number`] Define the characteristic resistance of the concrete in kN/cm^2.

E_ci [`number`] Modulus of elasticity or initial tangent deformation module of concrete, always referring to the cordal module in kN/cm^2.

E_cs [`number`] Secant deformation module of concrete in kN/cm^2.

fctm [`number`] Average concrete tensile strength in kN/cm^2.

fctk_inf [`number`] Minimum value of direct tensile strength in kN/cm^2.

fctk_sup [`number`] Maximum value of direct tensile strength in kN/cm^2.

fcd [`number`] Minimum value of design direct tensile strength in kN/cm^2.

c [`number`] Concrete covering in cm.

wk [`number`] Characteristic crack opening in the concrete surface in cm.

fconcrete.StructuralConcrete.ConcreteBeam module

class fconcrete.StructuralConcrete.ConcreteBeam.**ConcreteBeam**(*loads, beam_elements=N nodes=None, section=None, design_factor=1.4, division=1000, maximum_displacemen Concrete-Beam.<lambda>> available_long_steel_b object>, bar_steel_remova bar_steel_max_re available_transv_steel object>, tilt_angle_of_com available_concrete=<f object>, time_begin_long_ lifetime_structure=70 verbose=False, max_relative_diff_ consider_own_weight **options*)

Bases: *fconcrete.Structural.Beam.Beam*

Beam associated with the material concrete. All attbributes from *Beam Class* can be used.

Attributes

available_concrete [`AvailableConcrete`] Same constant from input. Define the available concrete. You can set the available fck, cost_by_m3, aggressiveness and aggregate. See more information in fc.AvailableConcrete docstring or the *AvailableMaterials Class* documentation. Default AvailableConcrete() which means:

- 30 MPa;
- R$353.30 by meter^3;
- The aggressiveness is 3;
- Aggregate is granite;
- Biggest aggregate dimension is 1.5cm.

available_long_steel_bars [`AvailableLongConcreteSteelBar`] Same constant from input. Define the available longitudinal steel bars. You can set the available diameters, cost_by_meter, fyw, E, etc. See more information in fc.AvailableLongConcreteSteelBar docstring or the *AvailableMaterials Class* documentation. Default AvailableLongConcreteSteelBar([8]) which means:

- 8mm diameter;
- 0.5cm^2 area;
- R$2.0575 by meter cost;
- fyw equal to 50kN/cm^2;
- Young Modulus (E) is 21000kN/cm^2;
- Max number of steel in the section is 200;
- Surface type is ribbed.

available_transv_steel_bars [`AvailableTransvConcreteSteelBar`] Same constant from input. Define the available transversal steel bars. You can set the available diameters, cost_by_meter, fyw, E, etc. See more information in fc.AvailableTransvConcreteSteelBar docstring or the *AvailableMaterials Class* documentation. Default AvailableTransvConcreteSteelBar([8]) which means:

- 8mm diameter;
- 0.5cm^2 area;
- R$2.0575 by meter cost;
- The longitudinal space between transversal steel are multiple of 5;
- fyw equal to 50kN/cm^2;
- Transversal bar inclination angle of 90 degrees;
- Tilt angle of compression struts of 45 degrees.

bar_steel_max_removal [`python:int`] Same constant from input. Define the max times it is possible to remove the bar. Default value is 100.

bar_steel_removal_step [`python:int`] Same constant from input. Define the step during the removal of the bar. Instead of taking the steel bars one by one, the bar_steel_removal_step will make the removal less constant. I makes the building process easier. Default value is 2.

cost [`number`] Total material cost of the beam.

cost_table [`number`] Detailed table with all materials and their costs.

design_factor [`number`] Same constant from input. Define the number that is going to be multiplied to de momentum diagram and shear diagram. If your load is already a design load, you should set design_factor=1. Default value is 1.4.

division [`python:int`] Same constant from input. Define the number of division solutions for the beam. The beam will be divided in equally spaced points and all results (displacement, momentum, shear) will be calculated to these points. Default value is 1.4.

lifetime_structure [`number`] The time, in months, when the value of the deferred arrow is desired; Default value is 70.

long_steel_bars [`LongSteelBars`] Longitudinal steels used in the beam.

long_steel_bars_solution_info [`LongSteelBarSolve`] Information about the solution for longitudinal steels used in the beam. More information in the *LongSteelBarSolve Class* documentation.

maximum_displacement_allowed [`number`] Same constant from input. For each beam element, compare its maximum displacement with maximum_displacement_allowed(beam_element_length). This is used to solve the ELS shown in NBR 6118. If a beam_element length is 120cm, its maximum displacement is 1cm and maximum_displacement_allowed is 120/250=0.45cm < 1cm. Therefore, in this condition, the ELS step will raise an error. Default value is lambda beam_element_length : beam_element_length/250.

processing_time [`number`] Time for resolution of the concrete beam.

subtotal_table [`number`] Table with each type of material and their costs.

tilt_angle_of_compression_struts [`number`] Same constant from input. Tilt angle of compression struts in degrees. Default 45 degrees.

time_begin_long_duration [`number`] The time, in months, relative to the date of application of the long-term load Default value is 0.

transv_steel_bars [`TransvSteelBar`] Transversal steels used in the beam.

transv_steel_bars_solution_info [`TransvSteelBarSolve`] Information about the solution for transversal steels used in the beam. More information in the *TransvSteelBarSolve Class* documentation.

verbose [*bool*] Print the the steps and their durations. Default value is False.

Methods

`checkRecalculationOfD`(self)	Recalculate all beam with the true value of steel height (d)
`copy`(self)	Makes a deep copy of the instance of Beam.
`getBeamElementInX`(self, x)	Get the beam element in x (in cm).
`getConcreteDisplacementDiagram`(self, **options)	Returns necessary steel area given the position and momentum.
`getDisplacement`(self, x)	Get the vertical displacement in a position x (in cm) or multiple positions.
`getDisplacementDiagram`(self, **options)	Apply beam.getDisplacement for options["division"] parts of the beam.
`getInternalMomentumStrength`(self, x)	Get the internal momentum strength in a position x (in cm) or multiple positions.
`getInternalShearStrength`(self, x)	Get the internal shear strength in a position x (in cm) or multiple positions.
`getMomentumDiagram`(self, **options)	Apply beam.getInternalMomentumStrength for options["division"] parts of the beam.
`getRotation`(self, x)	Get the rotation in a position x (in cm) or multiple positions.
`getRotationDiagram`(self, **options)	Apply beam.getRotation for options["division"] parts of the beam.
`getShearDesignDiagram`(self, **options)	Apply beam.getShearDiagram for options["division"] parts of the beam and multiplies by concrete_beam.design_factor.
`getShearDiagram`(self, **options)	Apply beam.getInternalShearStrength for options["division"] parts of the beam.
`matrix_rigidity_global`(self)	Returns the global rigidity matrix.
`plotConcreteDisplacementDiagram`(self, …)	Apply concrete_beam.getConcreteDisplacementDiagram for options["division"] parts of the beam.
`plotDisplacementDiagram`(self, **options)	Simply applies the beam.getDisplacementDiagram method results (x,y) to a plot with plt.plot(x, y).

Continued on next page

Table 16 – continued from previous page

plotMomentumDiagram(self, **options)	Simply applies the beam.getMomentumDiagram method results (x,y) to a plot with plt.plot(x, y).
plotRotationDiagram(self, **options)	Simply applies the beam.getRotationDiagram method results (x,y) to a plot with plt.plot(x, y).
plotShearDesignDiagram(self, **options)	Simply applies the beam.getShearDesignDiagram method results (x,y) to a plot with plt.plot(x, y).
plotShearDiagram(self, **options)	Simply applies the beam.getShearDiagram method results (x,y) to a plot with plt.plot(x, y).
plotTransversalInX(self, x)	Plot an image of the transversal section with the longitudinal and transversal steel.
solve_ELS(self)	Starts the process of solution for ELS (Estado Limite de Serviço)
solve_cost(self)	Starts the process of solution for the cost table.
solve_displacement(self)	Starts the process of solution for the structural beam displacement.
solve_long_steel(self)	Starts the process of solution for the used longitudinal steel.
solve_structural(self)	Starts the process of solution for the structural beam.
solve_transv_steel(self)	Starts the process of solution for the used transversal steel.

checkRecalculationOfD(*self*)
Recalculate all beam with the true value of steel height (d)

getConcreteDisplacementDiagram(*self*, ***options*)
Returns necessary steel area given the position and momentum.

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

Returns

x [python:list of number] X axis in cm.

displacement [python:list of number] Vertical displacement value in cm.

getShearDesignDiagram(*self*, ***options*)

Apply beam.getShearDiagram for options["division"] parts of the beam and multiplies by concrete_beam.design_factor.

Parameters

****options**

division: Number of divisions equally spaced (*int*). Default concrete_beam.division.

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

Returns

x [`python:list of number`] The x position of the division in cm

y [`python:list of number`] The value of shear for each x.

plotConcreteDisplacementDiagram(*self*, ***options*)

Apply concrete_beam.getConcreteDisplacementDiagram for options["division"] parts of the beam.

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

Returns

x [`python:list of number`] The x position of the division in cm

y [`python:list of number`] The value of displacement for each x.

plotShearDesignDiagram(*self*, ***options*)

Simply applies the beam.getShearDesignDiagram method results (x,y) to a plot with plt.plot(x, y).

Parameters

****options**

division: Number of divisions equally spaced (*int*).

x_begin: Begin of the x_axis (*number*).

x_end: End of the x_axis (*number*).

plotTransversalInX(*self*, *x*)

Plot an image of the transversal section with the longitudinal and transversal steel.

Call signatures:

concrete_beam.plotTransversalInX.getSteelArea(x)

Returns

fig Figure generated by matplotlib.

ax Axis generated by matplotlib.

solve_ELS (*self*)
Starts the process of solution for ELS (Estado Limite de Serviço)

solve_cost (*self*)
Starts the process of solution for the cost table.

solve_long_steel (*self*)
Starts the process of solution for the used longitudinal steel.

solve_transv_steel (*self*)
Starts the process of solution for the used transversal steel.

fconcrete.StructuralConcrete.ConcreteSection module

class fconcrete.StructuralConcrete.ConcreteSection.**ConcreteSection** (*function_height*)

Bases: *fconcrete.Structural.Section.Section*

Inject ConcreteSection properties to a generic Section.

Methods

getAreaBetween(self, begin_height, end_height)	Area between 2 y values.
plot(self[, N, color_plot, ax, fig])	Plot the section.
setSteelHeight(section[, ...])	Inject steel height (d) to the section.
width(self, y)	Gets the width in y.

static setSteelHeight (*section, positive_steel_height=0, negative_steel_height=0*)
Inject steel height (d) to the section.

Module contents

Code for structural calculus using concrete.

Submodules

fconcrete.cli module

fconcrete.config module

fconcrete.fconcrete module

Main module.

fconcrete.helpers module

`fconcrete.helpers.`**`cond`**(*x*, *singular=False*, *order=0*)
    If It is singular, return 1 if x>0 else 0. If It is not singular, return x**order if x>0 else 0

`fconcrete.helpers.`**`duplicated`**(*array*)
    Check if it is duplicated.

`fconcrete.helpers.`**`getAxis`**(*xy0=(0, 0)*, *xy1=(0, 0)*)
    Create axis with equal aspect. xy0 and xy1 represent the visible area.

`fconcrete.helpers.`**`integrate`**(*f*, *a*, *b*, *N=100*)
    Integrate f from a to b in N steps

`fconcrete.helpers.`**`printProgressBar`**(*iteration*, *total*, *prefix=''*, *suffix=''*, *decimals=1*, *length=100*, *fill=''*, *printEnd='r'*)
    Call in a loop to create terminal progress bar

`fconcrete.helpers.`**`timeit`**(*do=True*, *name=''*)
    Decorator to print the time that the function has taken to execute.

`fconcrete.helpers.`**`to_unit`**(*input*, *expected_unit*, *return_unit=False*)
    Convert between unities according to expected_unit and return_unit.

Call signatures:

fc.helpers.to_unit(input, expected_unit, return_unit=False)

```
>>> unit1 = fc.helpers.to_unit("10cm", "m")
>>> unit1
0.1
```

```
>>> unit2 = fc.helpers.to_unit(20, "m", return_unit="cm
↪")
>>> unit2
2000.0
```

Parameters

input [`number` or `python:str`] Represents the input unit of the user.

expected_unit [`python:str`] The expected unit to be given. Useful when input is a number.

return_unit [*bool*, optional] The desired unit to return

Module contents

Top-level package for FConcrete.

1.5.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/luisggc/fconcrete/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

FConcrete could always use more documentation, whether as part of the official FConcrete docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/luisggc/fconcrete/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *fconcrete* for local development.

1. Fork the *fconcrete* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/fconcrete.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv fconcrete
$ cd fconcrete/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 fconcrete tests
$ python setup.py test or pytest
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.org/luisggc/fconcrete/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_fconcrete
```

Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

1.5.5 Credits

Development Lead

- Luis Gabriel Gonçalves Coimbra <luiscoimbraeng@outlook.com>

Contributors

None yet. Why not be the first?

1.5.6 History

0.1.1 (2020-02-15)

- MVP

0.1.0 (2019-12-12)

- First release on PyPI. Updload to reserve the name.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

A

B

C

D

I

L

M

T

U

W