NuSA, análisis numérico estructural utilizando Python

P.J. De Los Santos

17 de septiembre de 2016

Índice general

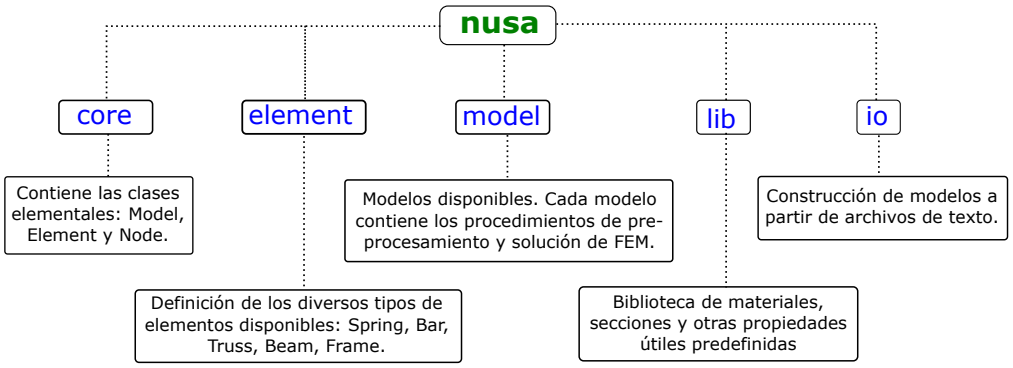
Capítulo 1

Una introducción a NuSA

1.1. ¿Qué es NuSA?

NuSA es una librería Python para resolver problemas de análisis estructural bidimensional. La idea es tener una estructura de códigos escritos utilizando la programación orientada a objetos, de modo que sea posible crear instancias de un modelo de elemento finito y operar con éste mediante métodos de clase.

La estructura de **NuSA** está basada en tres clases fundamentales que componen el *core*: `Model`, `Node` y `Element`.




```
class Model(object):
    """
    Superclass for all FEA models
    """
    def __init__(self,name,mtype):
        self.mtype = mtype # Model type
        self.name = name # Name
        self.nodes = {} # Dictionary for nodes {number: NodeObject}
        self.elements = {} # Dictionary for elements {number: ElementObject}

    def addNode(self,node):
        """
```

```
        Add element to current model

        *node* :  :class:‘~nusa.core.Node‘
        """
        current_label = self.getNumberOfNodes()
        if node.label is "":
            node.setLabel(current_label)
        self.nodes[node.label] = node

    def addElement(self,element):
        """
        Add element to current model

        *element* : :class:‘~nusa.core.Element‘
            Element instance

        ::

            m1 = Model()
            e1 = Bar(Node((0,0),0),Node((1,0),0))
            m1.addElement(e1)

        """
        if self.mtype != element.etype:
            raise ValueError("Element type must be "+self.mtype)
        current_label = self.getNumberOfElements()
        if element.label is "":
            element.setLabel(current_label)
        self.elements[element.label] = element

    def getNumberOfNodes(self):
        return len(self.nodes)

    def getNumberOfElements(self):
        return len(self.elements)

    def getNodes(self):
        return self.nodes.values()

    def getElements(self):
        return self.elements.values()

    def __str__(self):
        custom_str = ("Model: "+self.name+"\nNodes:
            "+str(self.getNumberOfNodes())+
        "\nElements: "+str(self.getNumberOfElements()))
        return custom_str
```

Capítulo 2

Elemento Spring

2.1. Fundamento teórico

El elemento *Spring* (resorte) es un elemento finito unidimensional donde las coordenadas locales y globales coinciden. Cada elemento spring tiene dos nodos como se muestra en la figura 2.1. Sea la rigidez del resorte la denotada por k, en este caso la matriz de rigidez del elemento está dada por:

$$K_{(e)} = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \tag{2.1}$$

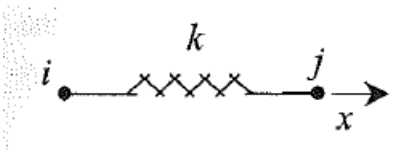


Figura 2.1 Elemento spring

Obviamente la matriz de rigidez para un elemento *spring* es de $2\,x\,2$, dado que este tiene dos grados de libertada, uno en cada nodo. Consecuentemente para un sistema de elementos *spring* con n nodos, el tamaño de la matriz global de rigidez K será de $n\,x\,n$. La matriz global de rigidez se obtiene ensamblando los matrices de rigidez por elemento $K_{(i)}$ para $i = 1, 2, ..., n$, utilizando el método directo de la rigidez.

Una vez que la matriz global de rigidez K es obtenida se tiene un sistema de ecuaciones de la forma:

$$[K]\{U\} = \{F\} \tag{2.2}$$

Donde U es el vector global de desplazamientos nodales y F es el vector global de fuerzas nodales.

El sistema de ecuaciones resultantes se puede simplificar aplicando las condiciones de frontera o restricciones de desplazamiento, quedando generalmente un sistema de menor

dimensión el cuál está determinado y puede resolverse utilizando métodos de álgebra lineal´, quedando una posible solución como:

$$\overline{U} = \overline{K}^{-1}\,\overline{F} \tag{2.3}$$

Donde $\overline{U}, \overline{K}$ *y* $\overline{F}$ corresponden a las variables descritas anteriormente, después de aplicar las condiciones de frontera correspondientes.

2.2. Un ejemplo resuelto en NuSA

Ejemplo 1. Para el ensamble mostrado en la figura 2.2, calcular a) la matriz global de rigidez b) los desplazamientos de los nodos 3 y 4 c) las fuerzas de reacción en los nodos 1 y 2, y d) las fuerzas en cada elemento. Una fuerza de 5000 lb es aplicada en el nodo 4 en la dirección x, las constantes de rigidez para cada resorte se muestran en la figura. Los nodos 1 y 2 están fijos. [1]

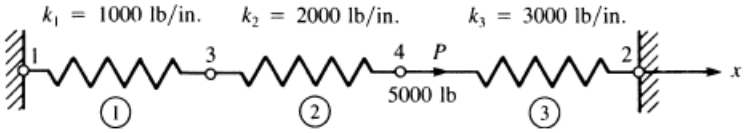


Figura 2.2 Ejemplo 1

Los pasos para solucionar el problema utilizando NuSA se resumen en la siguiente lista:

1. Importar las librerías a utilizar
2. Definir constantes o datos de entrada
3. Crear un modelo del tipo correspondiente
4. Crear nodos y elementos
5. Agregar los nodos y elementos al modelo
6. Indicar las cargas y condiciones de frontera
7. Resolver el modelo
8. Consultar los datos de salida requeridos

Siguiendo la metodología anterior, vamos a ir *desmenuzando* cada uno de los puntos expuestos.

Importar las librerías a utilizar

Se importan los módulos `core`, `model` y `element`, que contienen todas las clases necesarias para crear y solucionar el modelo de elemento finito.

```
from nusa.core import *
from nusa.model import *
from nusa.element import *
```

Definir constantes o datos de entrada

En esta paso se crean variables con datos a utilizar en el resto del procedimiento, las cuales pueden ser fuerzas nodales aplicadas, desplazamientos preescritos, o bien constantes mecánicas del material.

Para nuestro ejemplo se definen la fuerza P aplicada en el nodo 4 y las constantes de rigidez para cada resorte.

```
# Definiendo constantes
P = 5000.0
k1, k2, k3 = 1000, 2000, 3000
```

Crear un modelo de tipo correspondiente

Para este caso se crea un modelo instanciando un objeto de la clase *SpringModel*.

```
    m1 = SpringModel("2D Model")
```

Como puede notarse, el único argumento de entrada es un nombre para el modelo, mismo que no es necesario.

Crear nodos y elementos

```
# Nodos
n1 = Node((0,0))
n2 = Node((0,0))
n3 = Node((0,0))
n4 = Node((0,0))
# Elementos
e1 = Spring((n1,n3),k1)
e2 = Spring((n3,n4),k2)
e3 = Spring((n4,n2),k3)
```

Agregar los nodos y elementos al modelo

```
    for nd in (n1,n2,n3,n4):
        m1.addNode(nd)
    for el in (e1,e2,e3):
        m1.addElement(el)
```

Indicar las cargas y condiciones de frontera

```
    m1.addForce(n4,(P,))
    m1.addConstraint(n1,ux=0)
    m1.addConstraint(n2,ux=0)
```

Resolver el modelo

```
    m1.solve()
```

Consultar los datos de salida requeridos

Se puede hacer una

```
for node in m1.getNodes():
    print "UX{0}\t{1}".format(node.label, node.ux)
    print "FX{0}\t{1}\n".format(node.label, node.fx)
```

All

Ejecutando el script resultante:

```
"""
Logan, D. (2007). A first course in the finite element analysis.
Example 2.1, pp. 42.
"""
P = 5000.0
k1, k2, k3 = 1000, 2000, 3000
# Model
m1 = SpringModel("2D Model")
# Nodes
n1 = Node((0,0))
n2 = Node((0,0))
n3 = Node((0,0))
n4 = Node((0,0))
# Elements
e1 = Spring((n1,n3),k1)
e2 = Spring((n3,n4),k2)
```

```
e3 = Spring((n4,n2),k3)

# Add elements
for nd in (n1,n2,n3,n4):
    m1.addNode(nd)
for el in (e1,e2,e3):
    m1.addElement(el)

m1.addForce(n4,(P,))
m1.addConstraint(n1,ux=0)
m1.addConstraint(n2,ux=0)
m1.solve()

# a) Matriz global
print "a) Matriz global:\n {0}".format(m1.KG)
# b) Desplazamiento en los nodos 3 y 4
print "\nb) Desplazamientos de nodos 3 y 4"
print "UX3: {0}".format(n3.ux)
print "UX4: {0}".format(n4.ux)
# c) Fuerzas de reacción en los nodos 1 y 2
print "\nc) Fuerzas nodales en 1 y 2"
print "FX1: {0}".format(n1.fx)
print "FX2: {0}".format(n2.fx)
# d) Fuerzas en cada resorte
print "\nd) Fuerzas en elementos"
print "FE1:\n {0}".format(e1.fx)
print "FE2:\n {0}".format(e2.fx)
print "FE3:\n {0}".format(e3.fx)
```

Nos arroja como salida en consola los datos consutados:

```
NuSA 0.1.0
----------

a) Matriz global:
 [[ 1000.     0. -1000.     0.]
 [    0.  3000.     0. -3000.]
 [-1000.     0.  3000. -2000.]
 [    0. -3000. -2000.  5000.]]

b) Desplazamientos de nodos 3 y 4
UX3: 0.909090909091
UX4: 1.36363636364

c) Fuerzas nodales en 1 y 2
FX1: -909.090909091
FX2: -4090.90909091
```

```
d) Fuerzas en elementos
FE1:
 [[-909.09090909]
 [ 909.09090909]]
FE2:
 [[-909.09090909]
 [ 909.09090909]]
FE3:
 [[ 4090.90909091]
 [-4090.90909091]]
```

Bibliografía

[1] Logan Daryl. *A first course in the finite element method.* Thomson, United States, 2007.