Contents:
1.1 Connected springs (exs1.m)

This example is from the CALFEM manual.

**Purpose:**
Show the basic steps in a finite element calculation.

**Description:**
The general procedure in linear finite element calculations is carried out for a simple structure. The steps are:

- define the model
- generate element matrices
- assemble element matrices into the global system of equations
- solve the global system of equations
- evaluate element forces

Consider the system of three linear elastic springs, and the corresponding finite element model. The system of springs is fixed in its ends and loaded by a single load $F$. 
Import required modules:

```
In [1]: import numpy as np
In [2]: import calfem.core as cfc
```

The computation is initialized by defining the topology matrix Edof, containing element numbers and global element degrees of freedom:

```
In [3]: Edof = np.array([
    ...: [1,2],
    ...: [2,3],
    ...: [2,3],
    ...: ])
```

the global stiffness matrix $K$ (3x3) of zeros:

```
In [4]: K = np.matrix(np.zeros((3,3)))
In [5]: print(K)
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
```

and the load vector $f$ (3x1) with the load $F = 100$ in position 2:

```
In [7]: f = np.matrix(np.zeros((3,1))); f[1] = 100
In [8]: print(f)
[[ 0.]]
```
Element stiffness matrices are generated by the function `spring1e`. The element property `ep` for the springs contains the spring stiffnesses $k$ and $2k$ respectively, where $k = 1500$:

```
In [9]: k = 1500.; ep1 = k; ep2 = 2.*k
In [11]: Ke1 = cfc.spring1e(ep1)
In [12]: print(Ke1)
[[ 1500. -1500.]
 [-1500. 1500.]]
In [13]: Ke2 = cfc.spring1e(ep2)
In [14]: print(Ke2)
[[ 3000. -3000.]
 [-3000. 3000.]]
```

The element stiffness matrices are assembled into the global stiffness matrix $K$ according to the topology:

```
In []: cfc.assem(Edof[0,:], K, Ke2)
Out[16]:
matrix([[ 3000., -3000., 0.],
 [-3000., 3000., 0.],
 [ 0., 0., 0.]])
In []: cfc.assem(Edof[1,:], K, Ke1)
Out[17]:
matrix([[ 3000., -3000., 0.],
 [-3000., 4500., -1500.],
 [ 0., -1500., 1500.]]
In []: cfc.assem(Edof[2,:], K, Ke2)
Out[18]:
matrix([[ 3000., -3000., 0.],
 [-3000., 7500., -4500.],
 [ 0., -4500., 4500.]]
```

The global system of equations is solved considering the boundary conditions given in `bc`:

```
In [19]: bc = np.array([1,3])
In [20]: a, r = cfc.solveq(K, f, bc)
In [21]: print(a)
[[ 0.]
 [ 0.01333333]
 [ 0.]]
In [22]: print(r)
[[ -40.]
 [ 0.]
 [-60.]]
```

Element forces are evaluated from the element displacements. These are obtained from the global displacements $a$ using the function `extract`:

```
In [23]: ed1 = cfc.extractEldisp(Edof[0,:], a)
In [24]: print(ed1)
[ 0. 0.01333333]
```

1.1. Connected springs (exs1.m)
The spring forces are evaluated using the function spring1s:

```python
In [29]: es1 = cfc.spring1s(ep2, ed1)
In [30]: print(es1)
40.0
In [31]: es2 = cfc.spring1s(ep1, ed2)
In [32]: print(es2)
-20.0
In [33]: es3 = cfc.spring1s(ep2, ed3)
In [34]: print(es3)
-40.0
```

1.2 One-dimensional heat flow (exs2.py)

This example is from the CALFEM manual.

**Purpose:**

Analysis of one-dimensional heat flow.

**Description:**

Consider a wall built up of concrete and thermal insulation. The outdoor temperature is 17 °C and the temperature inside is 20 °C. At the inside of the thermal insulation there is a heat source yielding 10 W/m².

The wall is subdivided into five elements and the one-dimensional spring (analogy) element spring1e is used. Equivalent spring stiffnesses are $k_i = \lambda A/L$ for thermal conductivity and $k_i = A/R$ for thermal surface resistance. Corresponding spring stiffnesses per m² of the wall are:

- $k_1 = \frac{1}{0.04} = 25.0$ W/K
- $k_2 = \frac{1.7}{0.070} = 24.3$ W/K
- $k_3 = \frac{0.040}{0.100} = 0.4$ W/K
- $k_4 = \frac{1.7}{0.100} = 17.0$ W/K
- $k_5 = \frac{1}{0.13} = 7.7$ W/K

A global system matrix $K$ and a heat flow vector $f$ are defined. The heat source inside the wall is considered by setting $f_4 = 10$. The element matrices $K_e$ are computed using spring1e, and the function assem assembles the global stiffness matrix.

The system of equations is solved using solveq with considerations to the boundary conditions in bc and bcVal. The prescribed temperatures are $T_1 = 17$ °C and $T_2 = 20$°C.

Necessary modules are first imported:

```python
import numpy as np
import calfem.core as cfc
```
Next, the element topology is defined:

```python
Edof = np.array([[1,2],
                 [2,3],
                 [3,4],
                 [4,5],
                 [5,6]])
```

Create stiffness matrix and load vector:

```python
K = np.mat(np.zeros((6, 6)))
f = np.mat(np.zeros((6, 1)))
f[3] = 10.0
```

Define element properties and create element matrices for the different material layers:

```python
ep1 = 25.0
ep2 = 24.3
ep3 = 0.4
ep4 = 17.0
ep5 = 7.7
Ke1 = cfc.spring1e(ep1)
Ke2 = cfc.spring1e(ep2)
Ke3 = cfc.spring1e(ep3)
Ke4 = cfc.spring1e(ep4)
Ke5 = cfc.spring1e(ep5)
```

Assemble all element matrices into the global stiffness matrix:

```python
cfc.assem(Edof[0,:], K, Ke1)
cfc.assem(Edof[1,:], K, Ke2)
cfc.assem(Edof[2,:], K, Ke3)
cfc.assem(Edof[3,:], K, Ke4)
cfc.assem(Edof[4,:], K, Ke5)
print("Stiffness matrix K:")
print(K)
```

Define the boundary conditions and solve the system of equations:

```python
bc = np.array([1, 6])
bcVal = np.array([-17.0, 20.0])
a, r = cfc.solveq(K, f, bc, bcVal)
print("Displacements a:")
print(a)
print("Reaction forces r:")
print(r)
```

Calculate the element temperatures and calculate the heat flow:

```python
ed1 = cfc.extractEldisp(Edof[0,:], a)
ed2 = cfc.extractEldisp(Edof[1,:], a)
ed3 = cfc.extractEldisp(Edof[2,:], a)
```
ed4 = cfc.extractEldisp(Edof[3,:], a)
ed5 = cfc.extractEldisp(Edof[4,:], a)

q1 = cfc.spring1s(ep1, ed1)
q2 = cfc.spring1s(ep2, ed2)
q3 = cfc.spring1s(ep3, ed3)
q4 = cfc.spring1s(ep4, ed4)
q5 = cfc.spring1s(ep5, ed5)

print("q1 = "+str(q1))
print("q2 = "+str(q2))
print("q3 = "+str(q3))
print("q4 = "+str(q4))
print("q5 = "+str(q5))

Running this code produces the following output:

Stiffness matrix K:
[[ 25.  -25.  0.  0.  0.  0. ]
 [ -25.  49.3 -24.3  0.  0.  0. ]
 [  0. -24.3  24.7 -0.4  0.  0. ]
 [  0.  0.  -0.4  17.4 -17.  0. ]
 [  0.  0.  -17.  24.7 -7.7  7.7]
 [  0.  0.  0.  0.  -7.7  7.7]]

Displacements a:
[[-17. ]
 [-16.43842455]
 [-15.86067203]
 [ 19.23779344]
 [ 19.47540439]
 [ 20. ]]

Reaction forces r:
[[ -1.40393862e+01]
 [ -5.68434189e-14]
 [  0.00000000e+00]
 [  0.00000000e+00]
 [  0.00000000e+00]
 [  4.03938619e+00]]

q1 = 14.0393861892
q2 = 14.0393861892
q3 = 14.0393861892
q4 = 4.03938618922
q5 = 4.03938618922
CHAPTER 2

Mesh generation with CALFEM

Included in the Python version of CALFEM is a mesh generation library based on GMSH. The library encapsulates the usage of GMSH transparently for the user. It will also parse the output from GMSH and create the necessary data structures required by CALFEM for solving finite element problems.

Mesh generation in CALFEM is divided in three steps:

1. Defining the geometry of the model.
2. Creating a finite element mesh with the desired elements and properties
3. Extracting data structures that can be used by CALFEM.

The following sections describe these steps.

2.1 Required modules for geometry and mesh generation

To use the CALFEM geometry and mesh generation routines, we use the following import directives:

```python
import calfem.geometry as cfg
import calfem.mesh as cfm
import calfem.vis as cfv
```

2.2 Defining geometry

Geometry in CALFEM is described using the Geometry class. A Geometry-object will hold all points, lines and surfaces describing the geometry. This object is also what is passed to the mesh generation routines in the following sections.

A Geometry-object, g, is created with the following code:

```python
g = cfg.Geometry()
```
This creates a **Geometry** object which will be used to described our geometry. Next we define the points that will be used to define lines, splines or ellipses. In this example we define a simple triangle:

```python
g.point([0.0, 0.0]) # point 0
g.point([5.0, 0.0]) # point 1
g.point([2.5, 4.0]) # point 2
```

The points are connected together with spline-elements. These can have 2 or three nodes. In this case we only use 2 node splines (lines):

```python
g.spline([0, 1]) # line 0
g.spline([1, 2]) # line 1
g.spline([2, 0]) # line 2
```

Finally we create the surface by defining what lines make up the surface:

```python
g.surface([0, 1, 2])
```

To display our geometry, we use the calfem.vis module:

```python
cfv.drawGeometry(g)
cfv.showAndWait()
```

Running this example will show the following window with a simple rectangle:
2.3 Creating a mesh

To create a mesh we need to create a GmshMesh object and initialise this with our geometry:

```python
mesh = cfm.GmshMesh(g)
```

Next, we need to set some desired properties on our mesh:

```python
mesh.elType = 3       # Degrees of freedom per node.
mesh.dofsPerNode = 1  # Factor that changes element sizes.
mesh.elSizeFactor = 0.15 # Element size Factor
```

The `eltype` property determines the element used for mesh generation. Elements that can be generated are:

- 2 - 3 node triangle element
- 3 - 4 node quadrangle element
- 5 - 8 node hexahedron
- 16 - 8 node second order quadrangle

The `dofsPerNode` defines the number of degrees of freedom for each node. `elSizeFactor` determines the coarsness of the mesh.

To generate the mesh and at the same time get the needed data structures for use with CALFEM we call the `.create()` method of the mesh object:

```python
coords, edof, dofs, bdofs, elementmarkers = mesh.create()
```

The returned data structures are:

- `coords` - Element coordinates
- `edof` - Element topology
- `dofs` - Degrees of freedom per node
- `bdofs` - Boundary degrees of freedom. Dictionary containing the dofs for each boundary marker. More on markers in the next section.
- `elementmarkers` - List of integer markers. Row i contains the marker of element i. Can be used to determine what region an element is in.

To display the generated mesh we can use the `drawMesh()` function of the calfem.vis module:

```python
cfv.figure()
# Draw the mesh.
cfv.drawMesh(
    coords=coords,
    edof=edof,
    dofsPerNode=meshGen.dofsPerNode,
    elType=meshGen.elType,
    filled=True,
    title="Example 01"
)
```

Running the example will produce the following mesh with quad elements:
Changing the `elType` property to 2 will produce a mesh with triangle elements instead:
2.4 Specifying boundary markers

To assist in assigning boundary conditions, markers can be defined on the geometry, which can be used to identify which dofs are assigned to nodes, lines and surfaces.

In this example we add a marker, 10, to line number 2. Markers are added as a parameter to the .spline() method of the Geometry object as shown in the following code:

```python
# line 0
g.spline([0, 1])

# line 1
g.spline([1, 2])

# line 2 with marker 10
[g.spline([2, 0], marker=10) # line 2 with marker 10]
```

It is also possible to assign markers to points. The marker parameter is added to the .point() method of the Geometry object.

```python
# point 0
g.point([0.0, 0.0])

# point 1
[g.point([5.0, 0.0], marker=20) # point 1]

# point 2
[g.point([2.5, 4.0]) # point 2]
```

In the same way markers can be added to surfaces as well.
2.5 Extracting dofs defined by markers

To extract the dofs defined by the marker we use the \textit{bdofs} dictionary returned when the mesh was created by the \texttt{create()} method. If we print the \texttt{bdofs} dictionary we get the following output:

\begin{verbatim}
{(20: [2], 0: [1, 2, ..., 67], 10: [1, 3, 68, ..., 98])}
\end{verbatim}

If we examine the output we see that there is a key, 10, containing the dofs of the number 2 line. We also have the key 20 with a single dof 2 in this case. If the \textit{dofsPerNode} property in the mesh generator was set to 2 the marker 20 would have contained 2 integers.

2.6 Complete example with a solver

To illustrate the workflow of the mesh generation modules we implement a complete 2D plane stress solver.

2.6.1 Updated module imports

We need to add some additional import directives, such as the core \texttt{calfem} module as well as the \texttt{calfem.utils} module. We will also need \texttt{NumPy} as well as the standard math routines:

\begin{verbatim}
import calfem.core as cfc
import calfem.geometry as cfg
import calfem.mesh as cfm
import calfem.vis as cfv
import calfem.utils as cfu
import numpy as np
from math import *
\end{verbatim}

2.6.2 Problem variables and constants

To make it easier to update our example we define a number of variables describing our problem. First some geometric parameters describing our module, in this case a simple rectangular beam:

\begin{verbatim}
l = 5.0
h = 1.0
t = 0.2
\end{verbatim}

Next, we define our material properties we will need later in the code:

\begin{verbatim}
v = 0.35
E = 2.1e9
ptype = 1
ep = [ptype, t]
D=cfc.hooke(ptype, E, v)
\end{verbatim}

To make it easier to read our code we define 3 constants, which we will use instead of numerical markers.

\begin{verbatim}
left_support = 10
right_support = 20
top_line = 30
\end{verbatim}
2.6.3 Creating a Geometry object

We are now ready to create a Geometry object describing our geometry:

```
g = cfg.Geometry()
g.point([0.0, 0.0], marker = left_support) # point 0
g.point([l, 0.0], marker = right_support) # point 1
g.point([l, h]) # point 2
g.point([0.0, h]) # point 2

g.spline([0, 1]) # line 0
g.spline([1, 2]) # line 1
g.spline([2, 3], marker = top_line) # line 2
g.spline([3, 0]) # line 2

g.surface([0, 1, 2, 3])
```

The finished geometry is shown in below:

![Figure 1](image)

2.6.4 Creating a quad mesh

A quadrilateral mesh is now created with the following code. Please note that we use the `dofsPerNode` property to specify 2 dofs per node as this is a mechanical example.
mesh = cfm.GmshMesh(g)

mesh.elType = 3 # Degrees of freedom per node.
mesh.dofsPerNode = 2 # Factor that changes element sizes.
mesh.elSizeFactor = 0.10

coords, edof, dofs, bdofs, elementmarkers = mesh.create()

### 2.6.5 Implementing a CALFEM solver

We now have the necessary input to implement a simple CALFEM solver. First, we create some convenience variables, nDofs (total number of dofs), ex and ey (element x and y coordinates).

```
nDofs = np.size(dofs)
ex, ey = cfc.coordxtr(edof, coords, dofs)
```

The global stiffness matrix can now be allocated:

```
K = np.zeros([nDofs, nDofs])
```

For larger problems please consider using sparse matrices instead.

To make the assemblation loop less cluttered we use the `zip()` method to extract rows from `edof`, `ex` and `ey` to `eltopo`, `elx` and `ely`. The loop then becomes:

```
for eltopo, elx, ely in zip(edof, ex, ey):
    Ke = cfc.plange(elx, ely, ep, D)
    cfc.assem(eltopo, K, Ke)
```

Please note the difference from standard MATLAB CALFEM that the assem routine does not require returning the K matrix on the left side as the assemblation is done in place.

Next, we need to setup our boundary conditions. Two empty arrays are created, `bc` for storing the dof to prescribe and and a second `bcVal` for storing the prescribed values.

```
bc = np.array([], 'i')
bcVal = np.array([], 'f')
```

To prescribe a boundary condition the utility function `applybc()` is used. This function takes the boundary dictionary as input and applies the boundary condition to the correct dofs. Here we prescribe the left support as fixed and the right support fixed in y-direction.

```
bc, bcVal = cfu.applybc(bdofs, bc, bcVal, left_support, 0.0, 0)
bcb, bcVal = cfu.applybc(bdofs, bc, bcVal, right_support, 0.0, 2)
```
3.1 Core functions

CALFEM Core module
Contains all the functions implementing CALFEM standard functionality

calfem.core.assem(edof, K, Ke, f=None, fe=None)
Assemble element matrices Ke (and fe) into the global stiffness matrix K (and the global force vector f) according to the topology matrix edof.

Parameters:
- edof: dof topology array
- K: the global stiffness matrix
- Ke: element stiffness matrix
- f: the global force vector
- fe: element force vector

Output parameters:
- K: the new global stiffness matrix
- f: the new global force vector
- fe: element force vector

calfem.core.bar1e(ep)
Compute element stiffness matrix for spring element.

Parameters
- float ep: spring stiffness or analog quantity

Return mat Ke stiffness matrix, dim(Ke)= 2 x 2

calfem.core.bar1s(ep, ed)
Compute element force in spring element (spring1e).

Parameters
- ep: float spring stiffness or analog quantity
- ed: list element displacements [d0, d1]

Return float es element force

calfem.core.bar2e(ex, ey, ep)
Compute the element stiffness matrix for two dimensional bar element.
Parameters

- \textbf{ex} (list) – element x coordinates [x1, x2]
- \textbf{ey} (list) – element y coordinates [y1, y2]
- \textbf{ep} (list) – [E, A]: E - Young’s modulus, A - Cross section area

Return mat \textbf{Ke} stiffness matrix, [4 x 4]

\texttt{calfem.core.bar2g}(ex, ey, ep, N)

Compute element stiffness matrix for two dimensional geometric nonlinear bar element.

Parameters

- \textbf{ex} (list) – element x coordinates [x1, x2]
- \textbf{ey} (list) – element y coordinates [y1, y2]
- \textbf{ep} (list) – element properties [E, A], E - Young’s modulus, A - Cross section area
- \textbf{N} (float) – normal force

Return mat \textbf{Ke} stiffness matrix [4 x 4]

\texttt{calfem.core.bar2s}(ex, ey, ep, ed)

Compute normal force in two dimensional bar element.

Parameters

- \textbf{ex} (list) – element x coordinates [x1, x2]
- \textbf{ey} (list) – element y coordinates [y1, y2]
- \textbf{ep} (list) – element properties [E, A], E - Young’s modulus, A - Cross section area
- \textbf{ed} (list) – element displacements [u1, u2, u3, u4]

Return float \textbf{N} element force [N]

\texttt{calfem.core.bar3e}(ex, ey, ez, ep)

Compute element stiffness matrix for three dimensional bar element.

Parameters

- \textbf{ex} (list) – element x coordinates [x1, x2]
- \textbf{ey} (list) – element y coordinates [y1, y2]
- \textbf{ez} (list) – element z coordinates [z1, z2]
- \textbf{ep} (list) – element properties [E, A], E - Young’s modulus, A - Cross section area

Return mat \textbf{Ke} stiffness matrix, [6 x 6]

\texttt{calfem.core.bar3s}(ex, ey, ez, ep, ed)

Compute normal force in three dimensional bar element.

Parameters

- \textbf{ex} (list) – element x coordinates [x1, x2]
- \textbf{ey} (list) – element y coordinates [y1, y2]
- \textbf{ez} (list) – element z coordinates [z1, z2]
- \textbf{ep} (list) – element properties [E, A], E - Young’s modulus, A - Cross section area
- \textbf{ed} (list) – element displacements [u1, ..., u6]
Return float N  normal force
calfem.core.beam2d (ex, ey, ep)
    Calculate the stiffness matrix Ke, the mass matrix Me and the damping matrix Ce for a 2D elastic Bernoulli beam element.
    Parameters:
    ex = [x1, x2] ey = [y1, y2] element node coordinates
    ep = [E,A,I,m,(a,b)] element properties;
        E: Young’s modulus A: cross section area I: moment of inertia m: mass per unit length
        a,b: damping coefficients, Ce=aMe+bKe
    Returns:
    Ke element stiffness matrix (6 x 6) Me element mass matrix Ce element damping matrix, optional

calfem.core.beam2e (ex, ey, ep, eq=None)
    Compute the stiffness matrix for a two dimensional beam element.
    Parameters
        • ex (list) – element x coordinates [x1, x2]
        • ey (list) – element y coordinates [y1, y2]
        • ep (list) – element properties [E, A, I], E - Young’s modulus, A - Cross section area, I - Moment of inertia
        • eq (list) – distributed loads, local directions [qx, qy]
    Return mat Ke  element stiffness matrix [6 x 6]
    Return mat fe  element stiffness matrix [6 x 1] (if eq!=None)

calfem.core.beam2g (ex, ey, ep, N, eq=None)
    Compute the element stiffness matrix for a two dimensional beam element with respect to geometric nonlinearity.
    Parameters:
    ex = [x1, x2] ey = [y1, y2] element node coordinates
    ep = [E,A,I] element properties; E: Young’s modulus A: cross section area I: moment of inertia
    N axial force in the beam
    eq distributed transverse load
    Returns:
    Ke element stiffness matrix (6 x 6)
    fe element load vector (6 x 1)

calfem.core.beam2gs (ex, ey, ep, ed, N, eq=None)
    Calculate section forces in a two dimensional nonlinear beam element.
    Parameters:
    ex = [x1, x2] ey = [y1, y2] element node coordinates
    ep = [E,A,I] element properties; E: Young’s modulus A: cross section area I: moment of inertia
ed = [u1, ..., u6] element displacement vector
N axial force
eq = [qy] distributed transverse load

Returns:
es = [[N1, V1, M1], element forces, local directions] [N2, V2, M2]]
calfem.core.beam2s (ex, ey, ep, eq=None, nep=None)
Compute section forces in two dimensional beam element (beam2e).

Parameters:
ex = [x1 x2] ey = [y1 y2] element node coordinates
ep = [E A I] element properties, E: Young’s modulus A: cross section area I: moment of inertia
ed = [u1 ... u6] element displacements
eq = [qx qy] distributed loads, local directions
nep number of evaluation points (default=2)

Returns:
es = [ N1 V1 M1 section forces, local directions, in
N2 V2 M2 n points along the beam, dim(es)=
n x 3 ............]
edi = [ u1 v1 element displacements, local directions,
u2 v2 in n points along the beam, dim(es)= n x 2 .........]
eci = [ x1 local x-coordinates of the evaluation x2 points, (x1=0 and xn=L) ... ]
calfem.core.beam2t (ex, ey, ep, eq=None)
Compute the stiffness matrix for a two dimensional elastic Timoshenko beam element.

Parameters:
ex = [x1 x2] ey = [y1 y2] element node coordinates
ep = [E G A I ks] element properties
E: Young’s modulus G: shear modulus A: cross section area I: Moment of inertia
ks: Shear correction factor
eq = [qx qy] distributed loads, local directions

Returns:
Ke element stiffness matrix (6 x 6)
fe element load vector (6 x 1)
calfem.core.beam2ts (ex, ey, ep, ed=None, eq=None, nep=None)
Compute section forces in two dimensional beam element (beam2e).

Parameters:
ex = [x1, x2] ey = [y1, y2] element node coordinates
ep = [E,G,A,I,ks] element properties, E: Young’s modulus G: shear modulus A: cross section area
I: moment of inertia
ed = [u1, . . . , u6] element displacements
eq = [qx, qy] distributed loads, local directions
n number of evaluation points (default=2)

Returns:

\[
es = [[N1,V1,M1], \text{section forces, local directions, in } [N2,V2,M2], \text{ n points along the beam, dim}(es)=n \times 3 . . . . . .]
edi = [[u1,v1,teta1], \text{element displacements, local directions, } [u2,v2,teta2], \text{ and rotation of cross section at } . . . . . .] \text{ in n points along the beam, dim}(es)=n \times 2
\]

(Note! Rotation of the cross section is not equal to \(dv/dx\) for Timoshenko beam element)

eci = [[x1], local x-coordinates of the evaluation [x2], points, (x1=0 and xn=L) . . . .]

calfem.core.beam2w (ex, ey, ep, eq=None)

Compute the stiffness matrix for a two dimensional beam element on elastic foundation.

Parameters:
ex = [x1, x2] ey = [y1, y2] element node coordinates
ep = [E,A,I,ka,kt] element properties,
    E: Young’s modulus A: cross section area I: moment of inertia
    ka: axial foundation stiffness kt: transversal foundation stiffness
eq = [qx, qy] distributed loads, local directions

Returns:

Ke beam stiffness matrix (6 x 6)
fe element load vector (6 x 1)

calfem.core.beam2ws (ex, ey, ep, ed, eq=None)

Compute section forces in a two dimensional beam element on elastic foundation.

Parameters:
ex = [x1, x2] ey = [y1, y2] element node coordinates
ep = [E,A,I,ka,kt] element properties,
    E: Young’s modulus A: cross section area I: moment of inertia
    ka: axial foundation stiffness kt: transversal foundation stiffness
ed = [u1, . . . , u6] element displacement vector
eq = [qx, qy] distributed loads, local directions

Returns:

es = [[N1, V1, M1], [N2, V2, M2]] element forces, local direction

calfem.core.beam3e (ex, ey, ez, eo, ep, eq=None)

Calculate the stiffness matrix for a 3D elastic Bernoulli beam element.

Parameters:
ex = [x1 x2] ey = [y1 y2] ez = [z1 z2] element node coordinates
eo = [xz yz zz] orientation of local z axis
ep = [E G A Iy Iz Kv] element properties
CALFEM for Python Documentation, Release 3.4.1

E: Young’s modulus G: Shear modulus A: Cross section area

Iy: Moment of inertia, local y-axis Iz: Moment of inertia, local z-axis Kv: Saint-Venant’s torsion constant

eq = [qx qy qz qw] distributed loads

Returns:

Ke beam stiffness matrix (12 x 12)

fe equivalent nodal forces (12 x 1)

calfem.core.beam3s (ex, ey, ez, eo, ep, ed, eq=None, n=None)

Calculate the variation of the section forces and displacements along a three-dimensional beam element.

Parameters:

ex = [x1 x2] element node coordinates

ey = [y1 y2] ez = [z1 z2]

eo = [xz yz zz] orientation of local z axis

ep = [E G A Iy Iz Kv] element properties

ed the element displacement vector from the global coordinate system

eq = [qx qy qz qw] the distributed axial, transversal and torsional loads

n the number of point in which displacements and section forces are to be computed

Returns:

es = [[N1,Vy1,Vz1,T1,My1,Mz1], section forces in n points along the local x-axis , [N2,Vy2,Vz2,T2,My2,Mz2]],

edi = [[u1,v1,w1,fi1], displacements in n points along the local x-axis , [u2,v2,w2,fi2]],

eci = [[x1], local x-coordinates of the evaluation points , [x2], , [xn]]

calfem.core.coordxtr (edof, coords, dofs)

Create element coordinate matrices ex, ey, ez from edof coord and dofs matrices.

Parameters:

edof [nel x (nen * nnd)], nnd = number of node dofs coords [ncoords x ndims], ndims = node dimensions dofs [ncoords x nnd]

Returns:

ex if ndims = 1 ex, ey if ndims = 2 ex, ey, ez if ndims = 3

calfem.core.createdofs (nCoords, nDof)

Create dof array [nCoords x nDof]

calfem.core.effmises (es, ptype)

Calculate effective von mises stresses.

Parameters:

es

ptype= 1: plane stress 2: plane strain 3: axisymmetry 4: three dimensional
es = [[sigx, sigy, [sigz], tauxy]] element stress matrix

Returns:
eseff = [eseff_0 .. eseff_nel-1]
calfem.core.error(msg)
Write msg to error log.
calfem.core.extractEldisp(edof, a)
Extract element displacements from the global displacement vector according to the topology matrix edof.
Parameters:
a the global displacement vector edof dof topology array
Returns:
ed: element displacement array
calfem.core.flw2i4e(ex, ey, ep, D)
Compute element stiffness (conductivity) matrix for 4 node isoparametric field element
Parameters:
ex = [x1 x2 x3 x4] element coordinates
ey = [y1 y2 y3 y4]
ep = [t ir] thickness and integration rule
D = [[kxx kxy], [kyx kyy]] constitutive matrix
eq heat supply per unit volume
Returns: Ke element ‘stiffness’ matrix (4 x 4) fe element load vector (4 x 1)
calfem.core.flw2i4s(ex, ey, ep, D, ed)
Compute flows or corresponding quantities in the 4 node isoparametric element.
Parameters:
ex = [x1 x2 x3 x4] element coordinates
ey = [y1 y2 y3 y4]
ep = [t ir] thickness and integration rule
D = [[kxx kxy], [kyx kyy]] constitutive matrix
ed = [u1, u2, u3, u4] u1,u2,u3,u4: nodal values
Returns:
es = [[qx, qy], [...]] element flows
et = [[qx, qy], [...]] element gradients
calfem.core.flw2i8e(ex, ey, ep, D)
Compute element stiffness (conductivity) matrix for 8 node isoparametric field element.
Parameters:
ex = [x1, ..., x8] element coordinates
ey = [y1, ..., y8]
ep = [t, ir] thickness and integration rule

3.1. Core functions
\[ D = \begin{bmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{bmatrix} \] 
\[ \text{constitutive matrix} \]

\[ \text{eq heat supply per unit volume} \]

Returns:

Ke element ‘stiffness’ matrix (8 x 8) fe element load vector (8 x 1)

calfem.core.\texttt{flw2i8s}(ex, ey, ep, D, ed)

Compute flows or corresponding quantities in the 8 node isoparametric element.

Parameters:

\[ \text{ex = [x1, x2, x3, ..., x8]} \] element coordinates
\[ \text{ey = [y1, y2, y3, ..., y8]} \] element coordinates
\[ \text{ep = [t, ir]} \] thickness and integration rule
\[ \text{D = [k_{xx}, k_{xy}], [k_{yx}, k_{yy}]} \] constitutive matrix
\[ \text{ed = [u1, ..., u8]} \] u1, ..., u8: nodal values

Returns:

\[ \text{es = [qx, qy], [...]} \] element flows
\[ \text{et = [gx, gy], [...]} \] element gradients
\[ \text{eci} = \text{Gauss point location vector} \] 
\[ \text{nint: number of integration points} \]
\[ \text{[ix(nint), iy(nint)]} \]

calfem.core.\texttt{flw2qe}(ex, ey, ep, D, eq=None)

Compute element stiffness (conductivity) matrix for a triangular field element.

Parameters:

\[ \text{ex = [x1, x2, x3, x4]} \] element coordinates
\[ \text{ey = [y1, y2, y3, y4]} \] element coordinates
\[ \text{ep = [t]} \] element thickness
\[ \text{D = [k_{xx}, k_{xy}], [k_{yx}, k_{yy}]} \] constitutive matrix
\[ \text{eq heat supply per unit volume} \]

Returns:

Ke element ‘stiffness’ matrix (4 x 4)
fe element load vector (4 x 1)

calfem.core.\texttt{flw2qs}(ex, ey, ep, D, ed, eq=None)

Compute flows or corresponding quantities in the quadrilateral field element.

Parameters:

\[ \text{ex = [x1, x2, x3, x4]} \] element coordinates
\[ \text{ey = [y1, y2, y3, y4]} \] element coordinates
\[ \text{ep = [t]} \] element thickness
\[ \text{D = [k_{xx}, k_{xy}], [k_{yx}, k_{yy}]} \] constitutive matrix
\[ \text{ed = [u1, u2, u3, u4], [...]} \] u1, u2, u3, u4: nodal values
\[ \text{eq heat supply per unit volume} \]

Returns:

\[ \text{es = [qx, qy], [...]} \] element flows
\[ \text{et = [gx, gy], [...]} \] element gradients
calfem.core.flw2te(ex, ey, ep, D, eq=None)
Compute element stiffness (conductivity) matrix for a triangular field element.

Parameters:
- `ex` = [x1 x2 x3] element coordinates
- `ey` = [y1 y2 y3] element coordinates
- `ep` = [t] element thickness
- `D` = [kxx kxy; kyx kyy] constitutive matrix
  eq heat supply per unit volume

Returns:
- `Ke` element ‘stiffness’ matrix (3 x 3)
- `fe` element load vector (3 x 1)

calfem.core.flw2ts(ex, ey, D, ed)
Compute flows or corresponding quantities in the triangular field element.

Parameters:
- `ex` = [x1 x2 x3] element coordinates
- `ey` = [y1 y2 y3] element coordinates
- `D` = [kxx kxy; kyx kyy] constitutive matrix
- `ed` = [u1 u2 u3] u1, u2, u3: nodal values

Returns:
- `es` = [qx qy] ... ... element flows
- `et` = [gx gy] ... ... element gradients

calfem.core.flw3i8e(ex, ey, ez, ep, D, eq=None)
Compute element stiffness (conductivity) matrix for 8 node isoparametric field element.

Parameters:
- `ex` = [x1,x2,x3,...,x8] element coordinates
- `ey` = [y1,y2,y3,...,y8] element coordinates
- `ez` = [z1,z2,z3,...,z8] element coordinates
- `ep` = [ir] Ir: Integration rule
- `D` = [[kxx,kxy,kxz], [kyx,kyy,kyz], [kzx,kzy,kzz]] constitutive matrix
- `eq` heat supply per unit volume

Output:
- `Ke` element ‘stiffness’ matrix (8 x 8)
- `fe` element load vector (8 x 1)

calfem.core.flw3i8s(ex, ey, ez, ep, D, ed)
Compute flows or corresponding quantities in the 8 node (3-dim) isoparametric field element.

Parameters:
- `ex` = [x1,x2,x3,...,x8] element coordinates
- `ey` = [y1,y2,y3,...,y8] element coordinates
- `ez` = [z1,z2,z3,...,z8] element coordinates
- `ep` = [ir] Ir: Integration rule
- `D` = [[kxx,kxy,kxz], [kyx,kyy,kyz], [kzx,kzy,kzz]] constitutive matrix
- `ed` = [[u1, ..., u8], element nodal values [........]]

Output:
- `es` = [[qx,qy,qz], [........]] element flows(s)
et = [[qx,qy,qz], element gradients(s) [......]]
eci = [[ix1,ix1,iz1], location vector [..........], nint: number of integration points
       [ix(nint),iy(nint),iz(nint)]]

\[et \in \mathbb{R}^{3	imes nint}, \quad eci \in \mathbb{R}^{3	imes nint} \]
calfem.core.hooke (ptype, E, v)
Calculate the material matrix for a linear elastic and isotropic material.
Parameters:
\[ptype= 1: \text{plane stress} \quad 2: \text{plane strain} \quad 3: \text{axisymmetry} \quad 4: \text{three dimensional}\]
E Young's modulus v Poisson's const.
Returns:
D material matrix
calfem.core.info (msg)
Write msg to info log.
calfem.core.plani4e (ex, ey, ep, D, eq=None)
Calculate the stiffness matrix for a 4 node isoparametric element in plane strain or plane stress.
Parameters:
ex = [x1 . . . x4] element coordinates. Row array
ey = [y1 . . . y4]
ep = [ptype, t, ir] ptype: analysis type t: thickness ir: integration rule
D constitutive matrix
eq = [bx; by] bx: body force in x direction by: body force in y direction
Any array with 2 elements acceptable
Returns: Ke : element stiffness matrix (8 x 8) fe : equivalent nodal forces (8 x 1)
calfem.core.planqe (ex, ey, ep, D, eq=None)
Calculate the stiffness matrix for a quadrilateral plane stress or plane strain element.
Parameters:
ex=[x1 x2 x3 x4] element coordinates ey=[y1 y2 y3 y4]
ep = [ptype, t] ptype: analysis type t: element thickness
D constitutive matrix
eq = [bx; by] bx: body force in x direction by: body force in y direction
OUTPUT: Ke [element stiffness matrix (8 x 8)] fe : equivalent nodal forces (row array)
calfem.core.plane (ex, ey, ep, D, ed, eq=None)
Calculate element normal and shear stress for a quadrilateral plane stress or plane strain element.
Parameters:
ex = [x1 x2 x3 x4] element coordinates ey = [y1 y2 y3 y4]
ep = [ptype, t] ptype: analysis type t: thickness
D constitutive matrix
ed = [u1 u2 ..u8] element displacement vector
eq = [[bx] bx: body force in x direction [by]] by: body force in y direction
OUTPUT: es = [ sigx sigy (sigz) tauxy] element stress array et = [ epsx epsy (epsz) gamxy] element strain
array
calfem.core.plante (ex, ey, ep, D, eq=None)
Calculate the stiffness matrix for a triangular plane stress or plane strain element.
Parameters:
**CALFEM for Python Documentation, Release 3.4.1**

ex = [x1, x2, x3] element coordinates  
ey = [y1, y2, y3]

ep = [ptype, t]  
ptype: analysis type  
t: thickness
D constitutive matrix

eq = [bx, by]  
bx: body force x-dir  
by: body force y-dir

Returns:

Ke element stiffness matrix (6 x 6)  
fe equivalent nodal forces (6 x 1) (if eq is given)

```python
calfem.core.plantf(ex, ey, ep, es)
```

Compute internal element force vector in a triangular element in plane stress or plane strain.

Parameters:

ex = [x1, x2, x3] node coordinates  
ey = [y1, y2, y3]
ep = [ptype, t]  
ptype: analysis type  
t: thickness
es = [[sigx, sigy, sigz], tauxy] element stress matrix  
[ .... ] one row for each element

OUTPUT:

fe = [[f1], [f2], ..., [f8]] internal force vector

```python
calfem.core.plants(ex, ey, ep, D, ed)
```

Calculate element normal and shear stress for a triangular plane stress or plane strain element.

INPUT: ex = [x1 x2 x3] element coordinates  
ey = [y1 y2 y3]
ep = [ptype t]  
ptype: analysis type  
t: thickness
D constitutive matrix
ed = [u1 u2 ... u6 element displacement vector]  
[ .... ] one row for each element

OUTPUT:

es = [ sigx sigy sigz tauxy element stress matrix]  
[ .... ] one row for each element

et = [ epsx epsy sigz gamxy element strain matrix]  
[ .... ] one row for each element

```python
calfem.core.platre(ex, ey, ep, D, eq=None)
```

Calculate the stiffness matrix for a rectangular plate element. NOTE! Element sides must be parallel to the coordinate axis.

Parameters:

ex = [x1, x2, x3, x4] element coordinates  
ey = [y1, y2, y3, y4]
ep = [t] thickness
D constitutive matrix for plane stress

eq = [qz] load/unit area

Returns:

Ke element stiffness matrix (12 x 12)  
fe equivalent nodal forces (12 x 1)

```python
calfem.core.solveq(K, f, bcPrescr, bcVal=None)
```

Solve static FE-equations considering boundary conditions.

Parameters:
K global stiffness matrix, dim(K)= nd x nd f global load vector, dim(f)= nd x 1
bePrescr 1-dim integer array containing prescribed dofs. beVal 1-dim float array containing prescribed values.
If not given all prescribed dofs are assumed 0.

Returns:
a solution including boundary values Q reaction force vector
dim(a)=dim(Q)= nd x 1, nd : number of dof’s

```python
calfem.core.springle(ep)
```
Compute element stiffness matrix for spring element.

**Parameters**
- **ep** (float) – spring stiffness or analog quantity (ep = k).

**Return**
- **mat Ke** stiffness matrix, dim(Ke)= 2 x 2

```python
calfem.core.spring1s(ep, ed)
```
Compute element force in spring element (spring1e).

**Parameters**
- **ep** (float) – spring stiffness or analog quantity
- **ed** (list) – element displacements [d0, d1]

**Return**
- **float es** element force [N]

```python
calfem.core.spsolveq(K, f, bcPrescr, bcVal=None)
```
Solve static FE-equations considering boundary conditions.

**Parameters**
- K global stiffness matrix, dim(K)= nd x nd f global load vector, dim(f)= nd x 1
- bePrescr 1-dim integer array containing prescribed dofs. beVal 1-dim float array containing prescribed values.
If not given all prescribed dofs are assumed 0.

**Returns**
a solution including boundary values Q reaction force vector
dim(a)=dim(Q)= nd x 1, nd : number of dof’s

```python
calfem.core.statcon(K, f, cd)
```
Condensation of static FE-equations according to the vector cd.

**Parameters**
- K global stiffness matrix, dim(K) = nd x nd f global load vector, dim(f)= nd x 1
- cd vector containing dof’s to be eliminated dim(cd)= nc x 1, nc: number of condensed dof’s

**Returns**
- **K1 condensed stiffness matrix**, dim(K1)= (nd-nc) x (nd-nc)
- **f1 condensed load vector**, dim(f1)= (nd-nc) x 1

```python
calfem.core.stress2nodal(eseff, edof)
```
Convert element effective stresses to nodal effective stresses.

**Parameters**
eseff = [eseff_0 .. eseff_nel-1] edof = [dof topology array]

Returns:

ev: element value array [[ev_0_0 ev_0_1 ev_0_nen-1 ] ev_nel-1_0 ev_nel-1_1 ev_nel-1_nen-1]

3.2 Geometry functions

class calfem.geometry.Geometry

Instances of GeoData can hold geometric data and be passed to GmshMesher in pycalfem_Mesh to mesh the geometry.

addBSpline(points, ID=None, marker=0, elOnCurve=None, elDistribType=None, elDistribVal=None)

Adds a B-Spline curve

points - List of indices of control points that make a B-spline [p1, p2, ..., pn]

ID - Positive integer ID of this curve. If left unspecified the curve will be assigned the smallest unused curve-ID. It is recommended to specify all curve-IDs or none.

marker - Integer. Marker applied to this curve. Default 0.

elOnCurv - Positive integer. Elements on curve. The number of element edges that will be distributed along this curve. Only works for structured meshes.

eDistribType - String. Either “bump” or “progression”. Determines how the density of elements vary along the curve for structured meshes. Only works for structured meshes. elOnCurv and elDistribVal must be be defined if this param is used.

elDistribVal - Float. Determines how severe the element distribution is. Only works for structured meshes. elOnCurv and elDistribType must be be defined if this param is used.

bump:

Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the middle.

progression:

The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. elDistribVal = 2 meaning for example that each line element in the series will be twice as long as the preceding one. elDistribVal < 1 makes each element smaller than the preceding one.

addCircle(points, ID=None, marker=0, elOnCurve=None, elDistribType=None, elDistribVal=None)

Adds a Circle arc curve.

points - List of 3 indices of point that make a circle arc smaller than Pi. [startpoint, centerpoint, endpoint]

ID - Positive integer ID of this curve. If left unspecified the curve will be assigned the smallest unused curve-ID. It is recommended to specify all curve-IDs or none.

marker - Marker applied to this curve. Default 0.

eOnCurv - Elements on curve. The number of element edges that will be distributed along this curve. Only works for structured meshes.

eDistribType - String. Either “bump” or “progression”. Determines how the density of elements vary along the curve for structured meshes. Only works for structured meshes. elOnCurv and elDistribVal must be be defined if this param is used.
elDistribVal - Float. Determines how severe the element distribution is. Only works for structured meshes. elOnCurv and elDistribType must be be defined if this param is used.

bump:
Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the middle.

progression:
The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. elDistribVal = 2 meaning for example that each line element in the series will be twice as long as the preceding one. elDistribVal < 1 makes each element smaller than the preceeding one.

addEllipse(points, ID=None, elOnCurve=None, elDistribType=None, elDistribVal=None)
Adds a Ellipse arc curve.

points - List of 4 indices of point that make a ellipse arc smaller than Pi. [startpoint, centerpoint, mAxisPoint, endpoint] Startpoint is the starting point of the arc. Centerpoint is the point at the center of the ellipse. MAXAxisPoint is any point on the major axis of the ellipse. Endpoint is the end point of the arc.

ID - Positive integer ID of this curve. If left unspecified the curve will be assigned the smallest unused curve-ID. It is recommended to specify all curve-IDs or none.

marker - Integer. Marker applied to this curve. Default 0.

eOnCurv - Positive integer. Elements on curve. The number of element edges that will be distributed along this curve. Only works for structured meshes.

eDistribType - String. Either “bump” or “progression”. Determines how the density of elements vary along the curve for structured meshes. Only works for structured meshes. elOnCurv and elDistribVal must be be defined if this param is used.

eDistribVal - Float. Determines how severe the element distribution is. Only works for structured meshes. elOnCurv and elDistribType must be be defined if this param is used.

bump:
Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the middle.

progression:
The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. elDistribVal = 2 meaning for example that each line element in the series will be twice as long as the preceding one. elDistribVal < 1 makes each element smaller than the preceeding one.

addPoint(coord, ID=None, marker=0, elSize=1)
Adds a point.

Parameters: coord - [x, y] or [x, y, z]. List, not array.

ID - Positive integer ID of this point. If left unspecified the point will be assigned the smallest unused point-ID. It is recommended to specify all point-IDs or none.

marker - Marker applied to this point. Default 0. It is not a good idea to apply non-zero markers to points that are control points on B-splines or center points on circles/ellipses, since this can lead to “loose” nodes that are not part of any elements.
elSize - The size of elements at this point. Default 1. Use to make a mesh denser or sparser here. Only affects unstructured meshes

addPoints (points, markers=None, ids=None, elSizes=None)
Add points from a numpy-array

addRuledSurface (outerLoop, ID=None, marker=0)
Adds a Ruled Surface (bent surface). Parameters: outerLoop - List of 3 or 4 curve IDs that make up the boundary of
the surface.

ID - Positive integer ID of this surface. If left unspecified the surface will be assigned the smallest unused surface-ID. It is recommended to specify all surface-IDs or none.

marker - Integer. Marker applied to this surface. Default 0.

addSpline (points, ID=None, marker=0, elOnCurve=None, elDistribType=None, elDistribVal=None)
Adds a Spline curve
points - List of indices of control points that make a Spline [p1, p2, . . . , pn]

ID - Positive integer ID of this curve. If left unspecified the curve will be assigned the smallest unused curve-ID. It is recommended to specify all curve-IDs or none.

marker - Integer. Marker applied to this curve. Default 0.

elOnCurv - Positive integer. Elements on curve. The number of element edges that will be distributed along this curve. Only works for structured meshes.

elDistribType - String. Either “bump” or “progression”. Determines how the density of elements vary along the curve for structured meshes. Only works for structured meshes. elOnCurv and elDistribVal must be be defined if this param is used.

elDistribVal - Float. Determines how severe the element distribution is. Only works for structured meshes. elOnCurv and elDistribType must be be defined if this param is used.

bump:
Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the middle.

progression:
The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. elDistribVal = 2 meaning for example that each line element in the series will be twice as long as the preceding one. elDistribVal < 1 makes each element smaller than the preceeding one.

addSplines (points)
Add splines from numpy array

addStructuredSurface (outerLoop, ID=None, marker=0)
Adds a Structured Surface. Parameters: outerLoop - List of 4 curve IDs that make up the boundary of
the surface. The curves must be structured, i.e. their parameter ‘elOnCurv’ must be defined.

ID - Positive integer ID of this surface. If left unspecified the surface will be assigned the smallest unused surface-ID. It is recommended to specify all surface-IDs or none.

marker - Integer. Marker applied to this surface. Default 0.
addStructuredVolume (outerSurfaces, ID=None, marker=0)
    Adds a Structured Volume Parameters: outerSurfaces - List of surface IDs that make up the outer boundary of
    the volume. The surfaces must be Structured Surfaces.

    **ID - Positive integer ID of this volume. If left unspecified** the volume will be assigned the smallest un-
    used volume-ID. It is recommended to specify all volume-IDs or none.

    marker - Integer. Marker applied to this volume. Default 0.

addSurface (outerLoop, holes=[], ID=None, marker=0)
    Adds a plane surface (flat). Parameters: outerLoop - List of curve IDs that make up the outer boundary of
    the surface. The curves must lie in the same plane.

    **holes - List of lists of curve IDs that make up the inner** boundaries of the surface. The curves must lie
    in the same plane.

    **ID - Positive integer ID of this surface. If left unspecified** the surface will be assigned the smallest un-
    used surface-ID. It is recommended to specify all surface-IDs or none.

    marker - Integer. Marker applied to this surface. Default 0.

addVolume (outerSurfaces, holes=[], ID=None, marker=0)
    Adds a Volume Parameters: outerSurfaces - List of surface IDs that make up the outer boundary of
    the volume.

    **holes - List of lists of surface IDs that make up the inner** boundaries of the volume.

    **ID - Positive integer ID of this volume. If left unspecified** the volume will be assigned the smallest un-
    used volume-ID. It is recommended to specify all volume-IDs or none.

    marker - Integer. Marker applied to this volume. Default 0.

bspline (points, ID=None, marker=0, elOnCurve=None, elDistribType=None, elDistribVal=None)
    Adds a B-Spline curve

    **points - List of indices of control points that make a B-spline** [p1, p2, . . . , pn]

    **ID - Positive integer ID of this curve. If left unspecified the** curve will be assigned the smallest unused
    curve-ID. It is recommended to specify all curve-IDs or none.

    marker - Integer. Marker applied to this curve. Default 0.

    elOnCurv - Positive integer. Elements on curve. The number of element edges that will be distributed
    along this curve. Only works for structured meshes.

    elDistribType - String. Either “bump” or “progression”. Determines how the density of elements vary
    along the curve for structured meshes. Only works for structured meshes. elOnCurv and elDistribVal
    must be be defined if this param is used.

    elDistribVal - Float. Determines how severe the element distribution is. Only works for structured
    meshes. elOnCurv and elDistribType must be be defined if this param is used.

    bump:
    Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the
    middle.

    progression:
The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. \( \text{elDistribVal} = 2 \) meaning for example that each line element in the series will be twice as long as the preceding one. \( \text{elDistribVal} < 1 \) makes each element smaller than the preceding one.

\textbf{circle} \( (\text{points, ID=None, marker=0, elOnCurve=None, elDistribType=None, elDistribVal=None}) \)

Adds a Circle arc curve.

- **points** - list of 3 indices of point that make a circle arc smaller than Pi. [startpoint, centerpoint, endpoint]

- **ID** - Positive integer ID of this curve. If left unspecified the curve will be assigned the smallest unused curve-ID. It is recommended to specify all curve-IDs or none.

- **marker** - Marker applied to this curve. Default 0.

- **elOnCurv** - Elements on curve. The number of element edges that will be distributed along this curve. Only works for structured meshes.

- **elDistribType** - String. Either “bump” or “progression”. Determines how the density of elements vary along the curve for structured meshes. Only works for structured meshes. \( \text{elOnCurv} \) and \( \text{elDistribVal} \) must be be defined if this param is used.

- **elDistribVal** - Float. Determines how severe the element distribution is. Only works for structured meshes. \( \text{elOnCurv} \) and \( \text{elDistribType} \) must be be defined if this param is used.

  - **bump**: Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the middle.
  
  - **progression**: The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. \( \text{elDistribVal} = 2 \) meaning for example that each line element in the series will be twice as long as the preceding one. \( \text{elDistribVal} < 1 \) makes each element smaller than the preceding one.

\textbf{ellipse} \( (\text{points, ID=None, marker=0, elOnCurve=None, elDistribType=None, elDistribVal=None}) \)

Adds a Ellipse arc curve.

- **points** - List of 4 indices of point that make a ellipse arc smaller than Pi. [startpoint, centerpoint, mAxisPoint, endpoint] Startpoint is the starting point of the arc. Centerpoint is the point at the center of the ellipse. MAxisPoint is any point on the major axis of the ellipse. Endpoint is the end point of the arc.

- **ID** - Positive integer ID of this curve. If left unspecified the curve will be assigned the smallest unused curve-ID. It is recommended to specify all curve-IDs or none.

- **marker** - Integer. Marker applied to this curve. Default 0.

- **elOnCurv** - Positive integer. Elements on curve. The number of element edges that will be distributed along this curve. Only works for structured meshes.

- **elDistribType** - String. Either “bump” or “progression”. Determines how the density of elements vary along the curve for structured meshes. Only works for structured meshes. \( \text{elOnCurv} \) and \( \text{elDistribVal} \) must be be defined if this param is used.

- **elDistribVal** - Float. Determines how severe the element distribution is. Only works for structured meshes. \( \text{elOnCurv} \) and \( \text{elDistribType} \) must be be defined if this param is used.

  - **bump**:
Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the middle.

progression:
The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. elDistribVal = 2 meaning for example that each line element in the series will be twice as long as the preceding one. elDistribVal < 1 makes each element smaller than the preceding one.

**getPointCoords** *(IDs=None)*
Returns an N-by-3 list of point coordinates if the parameter is a list of IDs. If the parameter is just a single integer then a single coordinate (simple 3-element list) is returned. If the parameter is undefined (or None) all point coords will be returned

**line** *(points, ID=None, marker=0, elOnCurve=None, elDistribType=None, elDistribVal=None)*
Adds a Spline curve

points - List of indices of control points that make a Spline [p1, p2, ..., pn]

ID - Positive integer ID of this curve. If left unspecified the curve will be assigned the smallest unused curve-ID. It is recommended to specify all curve-IDs or none.

marker - Integer. Marker applied to this curve. Default 0.

elOnCurv - Positive integer. Elements on curve. The number of element edges that will be distributed along this curve. Only works for structured meshes.

elDistribType - String. Either “bump” or “progression”. Determines how the density of elements vary along the curve for structured meshes. Only works for structured meshes. elOnCurv and elDistribVal must be be defined if this param is used.

elDistribVal - Float. Determines how severe the element distribution is. Only works for structured meshes. elOnCurv and elDistribType must be be defined if this param is used.

bump:
Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the middle.

progression:
The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. elDistribVal = 2 meaning for example that each line element in the series will be twice as long as the preceding one. elDistribVal < 1 makes each element smaller than the preceding one.

**point** *(coord, ID=None, marker=0, elSize=1)*
Adds a point.

Parameters: coord - [x, y] or [x, y, z].

List, not array.

ID - Positive integer ID of this point. If left unspecified the point will be assigned the smallest unused point-ID. It is recommended to specify all point-IDs or none.

marker - Marker applied to this point. Default 0. It is not a good idea to apply non-zero markers to points that are control points on B-splines or center points on circles/ellipses, since this can lead to “loose” nodes that are not part of any elements.

elSize - The size of elements at this point. Default 1. Use to make a mesh denser or sparser here. Only affects unstructured meshes
pointsOnCurves (IDs)
Returns a list of all geometric points (not nodes) on the curves specified in IDs. IDs may be an integer or a list of integers.

removeCurve (ID)
Removes the curve with this ID

removePoint (ID)
Removes the point with this ID

removeSurface (ID)
Removes the surface with this ID

removeVolume (ID)
Removes the volume with this ID

ruledSurface (outerLoop, ID=None, marker=0)
Adds a Ruled Surface (bent surface). Parameters: outerLoop - List of 3 or 4 curve IDs that make up the boundary of
the surface.

ID - Positive integer ID of this surface. If left unspecified the surface will be assigned the smallest unused surface-ID. It is recommended to specify all surface-IDs or none.

marker - Integer. Marker applied to this surface. Default 0.

ruled_surface (outerLoop, ID=None, marker=0)
Adds a Ruled Surface (bent surface). Parameters: outerLoop - List of 3 or 4 curve IDs that make up the boundary of
the surface.

ID - Positive integer ID of this surface. If left unspecified the surface will be assigned the smallest unused surface-ID. It is recommended to specify all surface-IDs or none.

marker - Integer. Marker applied to this surface. Default 0.

setCurveMarker (ID, marker)
Sets the marker of the curve with the ID

setPointMarker (ID, marker)
Sets the marker of the point with the ID

setSurfaceMarker (ID, marker)
Sets the marker of the surface with the ID

setVolumeMarker (ID, marker)
Sets the marker of the volume with the ID

spline (points, ID=None, marker=0, elOnCurve=None, elDistribType=None, elDistribVal=None)
Adds a Spline curve

points - List of indices of control points that make a Spline [p1, p2, \ldots, pn]

ID - Positive integer ID of this curve. If left unspecified the curve will be assigned the smallest unused curve-ID. It is recommended to specify all curve-IDs or none.

marker - Integer. Marker applied to this curve. Default 0.

elOnCurv - Positive integer. Elements on curve. The number of element edges that will be distributed along this curve. Only works for structured meshes.
elDistribType - String. Either “bump” or “progression”. Determines how the density of elements vary along the curve for structured meshes. Only works for structured meshes. elOnCurv and elDistribVal must be be defined if this param is used.

elDistribVal - Float. Determines how severe the element distribution is. Only works for structured meshes. elOnCurv and elDistribType must be be defined if this param is used.

bump:
Smaller value means elements are bunched up at the edges of the curve, larger means bunched in the middle.

progression:
The edge of each element along this curve (from starting point to end) will be larger than the preceding one by this factor. elDistribVal = 2 meaning for example that each line element in the series will be twice as long as the preceding one. elDistribVal < 1 makes each element smaller than the preceding one.

structuredSurface (outerLoop, ID=None, marker=0)
Adds a Structured Surface. Parameters: outerLoop - List of 4 curve IDs that make up the boundary of the surface. The curves must be structured, i.e. their parameter ‘elOnCurv’ must be defined.

ID - Positive integer ID of this surface. If left unspecified the surface will be assigned the smallest unused surface-ID. It is recommended to specify all surface-IDs or none.

marker - Integer. Marker applied to this surface. Default 0.

structuredVolume (outerSurfaces, ID=None, marker=0)
Adds a Structured Volume Parameters: outerSurfaces - List of surface IDs that make up the outer boundary of the volume. The surfaces must be Structured Surfaces.

ID - Positive integer ID of this volume. If left unspecified the volume will be assigned the smallest unused volume-ID. It is recommended to specify all volume-IDs or none.

marker - Integer. Marker applied to this volume. Default 0.
**stuffOnSurfaces** *(IDs)*  
Returns lists of all geometric points and curves on the surfaces specified in IDs. IDs may be an integer or a list of integers

**stuffOnVolumes** *(IDs)*  
Returns lists of all geometric points, curves, and surfaces on the volumes specified in IDs. IDs may be an integer or a list of integers

**surface** *(outerLoop, holes=[], ID=None, marker=0)*  
Adds a plane surface (flat). Parameters: outerLoop - List of curve IDs that make up the outer boundary of the surface. The curves must lie in the same plane.

holes - **List of lists of curve IDs that make up the inner boundaries** of the surface. The curves must lie in the same plane.

ID - **Positive integer ID of this surface. If left unspecified** the surface will be assigned the smallest unused surface-ID. It is recommended to specify all surface-IDs or none.

marker - Integer. Marker applied to this surface. Default 0.

**volume** *(outerSurfaces, holes=[], ID=None, marker=0)*  
Adds a Volume Parameters: outerSurfaces - List of surface IDs that make up the outer boundary of the volume.

holes - **List of lists of surface IDs that make up the inner boundaries** of the volume.

ID - **Positive integer ID of this volume. If left unspecified** the volume will be assigned the smallest unused volume-ID. It is recommended to specify all volume-IDs or none.

marker - Integer. Marker applied to this volume. Default 0.

### 3.3 Mesh functions

calfem.mesh.GmshMesh  
alias of GmshMeshGenerator

**class** calfem.mesh.GmshMeshGenerator** *(geometry, el_type=2, el_size_factor=1, dofs_per_node=1, gmsh_exec_path=None, clcurv=False, min_size=None, max_size=None, meshing_algorithm=None, additional_options='', mesh_dir='', return_boundary_elements=False)*

Meshes geometry in GeoData objects or geo-files by calling the Gmsh executable. This is done when the function create() is called.

**create** *(is3D=False)*  
Meshes a surface or volume defined by the geometry in geoData. Parameters: is3D - Optional parameter that only needs to be set if geometry is loaded from a geo-file, i.e. if geoData is a path string. Default False.

Returns:

coords - Node coordinates

[[n0_x, n0_y, n0_z], [ ... ], [nn_x, nn_y, nn_z]]

edof - Element topology
dofs Node dofs
[[n0_dof1, ..., n0_dofn], [ ... ], [nn_dof1, ..., nn_dofn]]

bdofs Boundary dofs. Dictionary containing lists of dofs for each boundary marker.
Dictionary key = marker id.

elementmarkers List of integer markers. Row i contains the marker of element i.
Markers are similar to boundary markers and can be used to identify in which region an element lies.

boundaryElements (optional) returned if self.return_boundary_elements is true.
Contains dictionary with boundary elements. The keys are markers and the values are lists of elements for that marker.

Running this function also creates object variables:

nodesOnCurve Dictionary containing lists of node-indices. Key is a curve-ID and the value is a list of indices of all nodes on that curve, including its end points.

nodesOnSurface Dictionary containing lists of node-indices. Key is a surface-ID and the value is a list of indices of the nodes on that surface, including its boundary.

nodesOnVolume Dictionary containing lists of node-indices. Key is a volume-ID and the value is a list of indices of the nodes in that volume, including its surface.

calfem.mesh.trimesh2d(vertiers, segments=None, holes=None, maxArea=None, quality=True, dofs_per_node=1, logFilename='tri.log', triangleExecutablePath=None)

Triangulates an area described by a number vertices (vertices) and a set of segments that describes a closed polygon.

Parameters:
- vertices array [nVertices x 2] with vertices describing the geometry.
  \[ [v0_x, v0_y], [ ... ], [vn_x, vn_y] \]
- segments array [nSegments x 3] with segments describing the geometry.
  \[ [s0_v0, s0_v1,marker], [ ... ], [sn_v0, sn_v1,marker] \]
- holes [Not currently used]
- maxArea Maximum area for triangle. (None)
- quality If true, triangles are prevented having angles < 30 degrees. (True)
- dofs_per_node Number of degrees of freedom per node.
- logFilename Filename for triangle output ("tri.log")

Returns:
- coords Node coordinates
  \[ [n0_x, n0_y], [ ... ], [nn_x, nn_y] \]
- edof Element topology
  \[ [e0_dof1, ..., e0_dofn], [ ... ], [eln_dof1, ..., eln_dofn] \]
- dofs Node dofs
  \[ [n0_dof1, ..., n0_dofn], [ ... ], [nn_dof1, ..., nn_dofn] \]
bdofs Boundary dofs. Dictionary containing lists of dofs for each boundary marker. Dictionary key = marker id.

### 3.4 User interface functions

Created on Mon Apr 11 09:44:29 2016

@ author: lindemann

calfem.ui.appInstance(useVisVis=True)

Create a suitable application instance

calfem.ui.loadUiWidget(uifilename, parent=None)

Load user interface file and return object model

### 3.5 Utility functions

calfem.utils.applyTractionLinearElement(boundaryElements, coords, dofs, F, marker, q)

Apply traction on part of boundary with marker. q is added to all boundaryDofs defined by marker. Applicable to 2D problems with 2 dofs per node. The function works with linear line elements. (elm-type 1 in GMSH).

Parameters:
- boundaryElements Dictionary with boundary elements, the key is a marker and the values are lists of elements.
- coords Coordinates matrix
- dofs Dofs matrix
- F force matrix
- marker Boundary marker to assign boundary condition.
- q Value to assign boundary condition.

shape = [qx qy] in global coordinates

calfem.utils.applybc(boundaryDofs, bcPrescr, bcVal, marker, value=0.0, dimension=0)

Apply boundary condition to bcPresc and bcVal matrices. For 2D problems with 2 dofs per node.

Parameters:
- boundaryDofs Dictionary with boundary dofs.
- bcPresc 1-dim integer array containing prescribed dofs.
- bcVal 1-dim float array containing prescribed values.
- marker Boundary marker to assign boundary condition.
- value Value to assign boundary condition.

If not given 0.0 is assigned.

dimension dimension to apply bc. 0 - all, 1 - x, 2 - y

Returns:
- bcPresc Updated 1-dim integer array containing prescribed dofs.
- bcVal Updated 1-dim float array containing prescribed values.

calfem.utils.applybc3D(boundaryDofs, bcPrescr, bcVal, marker, value=0.0, dimension=0)

Apply boundary condition to bcPresc and bcVal matrices. For 3D problems with 3 dofs per node.

Parameters:
- boundaryDofs Dictionary with boundary dofs.
- bcPresc 1-dim integer array containing prescribed dofs.
- bcVal 1-dim float array containing prescribed values.
- marker Boundary marker to assign boundary condition.
- value Value to assign boundary condition.

If not given 0.0 is assigned.

dimension dimension to apply bc. 0 - all, 1 - x, 2 - y, 3 - z
Returns:

- `bcPresc`: Updated 1-dim integer array containing prescribed dofs.
- `bcVal`: Updated 1-dim float array containing prescribed values.

calfem.utils.**applyforce** *(boundaryDofs, f, marker, value=0.0, dimension=0)*

Apply boundary force to f matrix. The value is added to all boundaryDofs defined by marker. Applicable to 2D problems with 2 dofs per node.

Parameters:

- `boundaryDofs`: Dictionary with boundary dofs.
- `f`: Force matrix.
- `marker`: Boundary marker to assign boundary condition.
- `value`: Value to assign boundary condition.

If not given 0.0 is assigned.

- `dimension`: Dimension to apply force. 0 - all, 1 - x, 2 - y

calfem.utils.**applyforce3D** *(boundaryDofs, f, marker, value=0.0, dimension=0)*

Apply boundary force to f matrix. The value is added to all boundaryDofs defined by marker. Applicable to 3D problems with 3 dofs per node.

Parameters:

- `boundaryDofs`: Dictionary with boundary dofs.
- `f`: Force matrix.
- `marker`: Boundary marker to assign boundary condition.
- `value`: Value to assign boundary condition.

If not given 0.0 is assigned.

- `dimension`: Dimension to apply force. 0 - all, 1 - x, 2 - y, 3 - z

calfem.utils.**applyforcetotal** *(boundaryDofs, f, marker, value=0.0, dimension=0)*

Apply boundary force to f matrix. Total force, value, is distributed over all boundaryDofs defined by marker. Applicable to 2D problems with 2 dofs per node.

Parameters:

- `boundaryDofs`: Dictionary with boundary dofs.
- `f`: Force matrix.
- `marker`: Boundary marker to assign boundary condition.
- `value`: Total force value to assign boundary condition.

If not given 0.0 is assigned.

- `dimension`: Dimension to apply force. 0 - all, 1 - x, 2 - y

calfem.utils.**applyforcetotal3D** *(boundaryDofs, f, marker, value=0.0, dimension=0)*

Apply boundary force to f matrix. Total force, value, is distributed over all boundaryDofs defined by marker. Applicable to 3D problems with 3 dofs per node.

Parameters:

- `boundaryDofs`: Dictionary with boundary dofs.
- `f`: Force matrix.
- `marker`: Boundary marker to assign boundary condition.
- `value`: Total force value to assign boundary condition.

If not given 0.0 is assigned.

- `dimension`: Dimension to apply force. 0 - all, 1 - x, 2 - y, 3 - z

calfem.utils.**readFloat** *(f)*

Read a row from file, f, and return a list of floats.

calfem.utils.**readInt** *(f)*

Read a row from file, f, and return a list of integers.
**calfem.utils.readSingleFloat** *(f)*
Read a single float from a row in file f. All other values on row are discarded.

**calfem.utils.readSingleInt** *(f)*
Read a single integer from a row in file f. All other values on row are discarded.

**calfem.utils.scalfact2** *(ex, ey, ed, rat=0.2)*
Determine scale factor for drawing computational results, such as displacements, section forces or flux.

Parameters:
- ex, ey element node coordinates
- ed element displacement matrix or section force matrix
- rat relation between illustrated quantity and element size. If not specified, 0.2 is used.

**calfem.utils.which** *(filename)*
Return complete path to executable given by filename.

### 3.6 Visualisation functions

**calfem.vis.addLabel** *(text, pos, angle=0, fontName=None, fontSize=9, color='k', bgcolor=None, axes=None)*
Adds a label inside the axes. Returns the Label object. Parameters: text - String. The text of the label pos - Tuple with two numbers. The (x,y) position of the label with origin at the upper left corner.


Default ‘k’ (black).

bgcolor - Background color. See color. Default None. axes - Axes wherein the label is placed. If None then the current axes is chosen.

**calfem.vis.addText** *(text, pos, angle=0, fontName=None, fontSize=9, color='k', bgcolor=None, axes=None)*
Adds a text in the world space. Returns the Text object. Parameters: text - String. The text of the label pos - Tuple with two or three numbers. The (x,y,z) position of the text in world space.


Default ‘k’ (black).

bgcolor - Background color. See color. Default None. axes - Axes wherein the label is placed. If None then the current axes is chosen.

**calfem.vis.add_label** *(text, pos, angle=0, fontName=None, fontSize=9, color='k', bgcolor=None, axes=None)*
Adds a label inside the axes. Returns the Label object. Parameters: text - String. The text of the label pos - Tuple with two numbers. The (x,y) position of the label with origin.
angle - Float or int. The rotation of the label in degrees. fontname- String. Either 'mono', 'sans' or 'serif'.
fontSize- Int. Size of the text. Default 9. color - A 3-tuple or a character in 'rgbycmkw', etc that defines text
color.

Default 'k' (black).

bgcolor - Background color. See color. Default None. axes - Axes wherein the label is placed. If None then the
current axes is

chosen.
calfem.vis.add_text (text, pos, angle=0, fontName=None, fontSize=9, color='k',
bgcolor=None, axes=None)

Adds a text in the world space. Returns the Text object. Parameters: text - String. The text of the label pos -
Tuple with two or three numbers. The (x,y,z) position of the text in

world space.

angle - Float or int. The rotation of the label in degrees. fontname- String. Either 'mono', 'sans' or 'serif'.
fontSize- Int. Size of the text. Default 9. color - A 3-tuple or a character in 'rgbycmkw', etc that defines text
color.

Default 'k' (black).

bgcolor - Background color. See color. Default None. axes - Axes wherein the label is placed. If None then the
current axes is

chosen.
calfem.vis.camera3d ()
Get visvis 3D camera.

calfem.vis.clf ()
Clear visvis figure

calfem.vis.closeAll ()
Close all visvis windows.

calfem.vis.close_all ()
Close all visvis windows.

calfem.vis.colorBar (axes=None)
Short form of getColorbar

calfem.vis.color_bar (axes=None)
Short form of getColorbar

calfem.vis.drawDisplacements (displacements, coords, edof, dofsPerNode, elType, node_vals=None,
clim=None, axes=None, axesAdjust=True, doDrawUndisplacedMesh=True, magnfac=1.0, title=None)

Draws mesh with displacements in 2D or 3D. Scalar nodal values can also be drawn on the mesh. Returns the

displaced Mesh object. Parameters: displacements-An N-by-1 array (or matrix). Row i contains the displace-
ment of
dof i. N-by-2 or N-by-3 arrays are also accepted, in which case row i contains the x,y,z displacements
of node i.

coords - An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates of node i.
edof - An E-by-L array. Element topology. (E is the number of elements and L is the number of dofs per
element)
dofsPerNode - Integer. Dofs per node. eType - Integer. Element Type. See Gmsh manual for details. Usually 2 for triangles or 3 for quadrangles.

nodeVals - An N-by-1 array or a list of scalars. The Scalar values at the nodes. nodeVals[i] should be the value of node i.

clim - 2-tuple. Colorbar limits (min, max). Defines the value range of the colorbar. Defaults to None, in which case min/max are set to min/max of nodeVals.

axes - Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.

axesAdjust - Boolean. True if the view should be changed to show the whole model. Default True.

doDrawMesh - Boolean. True if mesh wire should be drawn. Default True. magnfac - Float. Magnification factor. Displacements are multiplied by this value. Use this to make small displacements more visible.

title - String. Changes title of the figure. Default None (in which case title depends on other parameters).

calfem.vis.drawElementValues(ev, coords, edof, dofsPerNode, eType, displacements=None, clim=None, axes=None, axesAdjust=True, doDrawMesh=True, doDrawUndisplacedMesh=False, magnfac=1.0, title=None)

Drews scalar element values in 2D or 3D. Returns the world object elementsWobject that represents the mesh.

Parameters:
ev - An N-by-1 array or a list of scalars. The Scalar values of the elements. ev[i] should be the value of element i.

coords - An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates of node i.

edof - An E-by-L array. Element topology. (E is the number of elements and L is the number of dofs per element)

dofsPerNode - Integer. Dofs per node. eType - Integer. Element Type. See Gmsh manual for details. Usually 2 for triangles or 3 for quadrangles.

displacements - An N-by-2 or N-by-3 array. Row i contains the x,y,z displacements of node i.

clim - 2-tuple. Colorbar limits (min, max). Defines the value range of the colorbar. Defaults to None, in which case min/max are set to min/max of nodeVals.

axes - Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.

axesAdjust - Boolean. True if the view should be changed to show the whole model. Default True.

doDrawMesh - Boolean. True if mesh wire should be drawn. Default True. doDrawUndisplacedMesh - Boolean. True if the wire of the undisplaced mesh should be drawn on top of the displaced mesh. Default False. Use only if displacements != None.

magnfac - Float. Magnification factor. Displacements are multiplied by this value. Use this to make small displacements more visible.

title - String. Changes title of the figure. Default “Element Values”.

3.6. Visualisation functions
calfem.vis.drawGeometry(geoData, axes=None, axesAdjust=True, drawPoints=True, labelPoints=True, labelCurves=True, title=None, fontSize=11, N=20)

Draws the geometry (points and curves) in geoData. Parameters:
- geoData: GeoData object. Geodata contains geometric information of the model.
- axes: Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.
- axesAdjust: Boolean. If True the view will be changed to show the whole model. Default True.
- drawPoints: Boolean. If True points will be drawn. labelPoints: Boolean. If True Points will be labeled. The format is: ID[marker]. If a point has marker==0 only the ID is written.
- labelCurves: Boolean. If True Curves will be labeled. The format is: ID(elementsOnCurve)[marker].

calfem.vis.drawMesh(coords, edof, dofsPerNode, elType, axes=None, axesAdjust=True, title=None, color=(0, 0, 0), faceColor=(1, 1, 1), filled=False)

Draws wire mesh of model in 2D or 3D. Returns the Mesh object that represents the mesh. Parameters:
- coords: An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates of node i.
- edof: An E-by-L array. Element topology. (E is the number of elements and L is the number of dofs per element)
- dofsPerNode: Integer. Dofs per node. elType: Integer. Element Type. See Gmsh manual for details. Usually 2 for triangles or 3 for quadrangles.
- axes: Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.
- axesAdjust: Boolean. True if the view should be changed to show the whole model. Default True.
- title: String. Changes title of the figure. Default “Mesh”.
- color: 3-tuple or char. Color of the wire. Defaults to black (0,0,0).
  Can also be given as a character in ‘rgbycmkw’.
- faceColor: 3-tuple or char. Color of the faces. Defaults to white (1,1,1). Parameter filled must be True or faces will not be drawn at all.
- filled: Boolean. Faces will be drawn if True. Otherwise only the wire is drawn. Default False.

calfem.vis.drawNodalValues(nodeVals, coords, edof, dofsPerNode, elType, clim=None, axes=None, axesAdjust=True, doDrawMesh=True, title=None)

Draws scalar nodal values in 2D or 3D. Returns the Mesh object that represents the mesh. Parameters:
- nodeVals: An N-by-1 array or a list of scalars. The Scalar values at the nodes. nodeVals[i] should be the value of node i
coords - An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates of node i.

edof - An E-by-L array. Element topology. (E is the number of elements and L is the number of dofs per element)

dofsPerNode - Integer. Dofs per node. elType - Integer. Element Type. See Gmsh manual for details. Usually 2 for triangles or 3 for quadrangles.

clim - 2-tuple. Colorbar limits (min, max). Defines the value range of the colorbar. Defaults to None, in which case min/max are set to min/max of nodeVals.

axes - Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.

axesAdjust - Boolean. True if the view should be changed to show the whole model. Default True.

doDrawMesh - Boolean. True if mesh wire should be drawn. Default True. title - String. Changes title of the figure. Default "Node Values".

calfem.vis.draw_displacements (displacements, coords, edof, dofsPerNode, elType, nodeVals=None, clim=None, axes=None, axesAdjust=True, doDrawUndisplacedMesh=True, magnfac=1.0, title=None)

Draws mesh with displacements in 2D or 3D. Scalar nodal values can also be drawn on the mesh. Returns the displaced Mesh object. Parameters: displacements-An N-by-1 array (or matrix). Row i contains the displacement of dof i. N-by-2 or N-by-3 arrays are also accepted, in which case row i contains the x,y,z displacements of node i.

coords - An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates of node i.

edof - An E-by-L array. Element topology. (E is the number of elements and L is the number of dofs per element)

dofsPerNode - Integer. Dofs per node. elType - Integer. Element Type. See Gmsh manual for details. Usually 2 for triangles or 3 for quadrangles.

nodeVals - An N-by-1 array or a list of scalars. The Scalar values at the nodes. nodeVals[i] should be the value of node i.

clim - 2-tuple. Colorbar limits (min, max). Defines the value range of the colorbar. Defaults to None, in which case min/max are set to min/max of nodeVals.

axes - Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.

axesAdjust - Boolean. True if the view should be changed to show the whole model. Default True.

doDrawMesh - Boolean. True if mesh wire should be drawn. Default True. magnfac - Float. Magnification factor. Displacements are multiplied by this value. Use this to make small displacements more visible.

title - String. Changes title of the figure. Default None (in which case title depends on other parameters).
calfem.vis.draw_element_values(ev, coords, edof, dofsPerNode, elType, displacements=None, clim=None, axes=None, axesAdjust=True, doDrawMesh=True, doDrawUndisplacedMesh=False, magnfac=1.0, title=None)

Draws scalar element values in 2D or 3D. Returns the world object elementsWobject that represents the mesh.

Parameters:
- **ev** - An N-by-1 array or a list of scalars. The Scalar values of the elements. ev[i] should be the value of element i.
- **coords** - An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates of node i.
- **edof** - An E-by-L array. Element topology. (E is the number of elements and L is the number of dofs per element)
- **dofsPerNode** - Integer. Dofs per node. elType - Integer. Element Type. See Gmsh manual for details. Usually 2 for triangles or 3 for quadrangles.
- **displacements** - An N-by-2 or N-by-3 array. Row i contains the x,y,z displacements of node i.
- **clim** - 2-tuple. Colorbar limits (min, max). Defines the value range of the colorbar. Defaults to None, in which case min/max are set to min/max of nodeVals.
- **axes** - Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.
- **axesAdjust** - Boolean. True if the view should be changed to show the whole model. Default True.
- **doDrawMesh** - Boolean. True if mesh wire should be drawn. Default True. doDrawUndisplacedMesh - Boolean. True if the wire of the undisplaced mesh should be drawn on top of the displaced mesh. Default False. Use only if displacements != None.
- **magnfac** - Float. Magnification factor. Displacements are multiplied by this value. Use this to make small displacements more visible.
- **title** - String. Changes title of the figure. Default “Element Values”.

calfem.vis.draw_geometry(geoData, axes=None, axesAdjust=True, drawPoints=True, labelPoints=True, labelCurves=True, title=None, fontSize=11, N=20)

Draws the geometry (points and curves) in geoData Parameters: geoData - GeoData object. Geodata contains geometric information of the model.

- **axes** - Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.
- **axesAdjust** - Boolean. If True the view will be changed to show the whole model. Default True.
- **drawPoints** - Boolean. If True points will be drawn. labelPoints- Boolean. If True Points will be labeled. The format is:
  - ID[marker]. If a point has marker==0 only the ID is written.
- **labelCurves**- Boolean. If True Curves will be labeled. The format is: ID(elementsOnCurve)[marker].
**calfem.vis.draw_mesh**

```python
calfem.vis.draw_mesh(coords, edof, dofsPerNode, elType, axes=None, axesAdjust=True, title=None,
                     color=(0, 0, 0), faceColor=(1, 1, 1), filled=False)
```

Draws wire mesh of model in 2D or 3D. Returns the Mesh object that represents the mesh. Parameters:

- **coords** - An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates of node i.
- **edof** - An E-by-L array. Element topology. \((E \text{ is the number of elements and } L \text{ is the number of dofs per element})\)
- **dofsPerNode** - Integer. Dofs per node. \(elType\) - Integer. Element Type. See Gmsh manual for details. Usually 2 for triangles or 3 for quadrangles.
- **axes** - Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.
- **axesAdjust** - Boolean. True if the view should be changed to show the whole model. Default True.
- **title** - String. Changes title of the figure. Default “Mesh”. \(color\) - 3-tuple or char. Color of the wire. Defaults to black (0,0,0).
  - Can also be given as a character in ‘rgbycmkw’.
- **faceColor** - 3-tuple or char. Color of the faces. Defaults to white (1,1,1). Parameter filled must be True or faces will not be drawn at all.
- **filled** - Boolean. Faces will be drawn if True. Otherwise only the wire is drawn. Default False.

**calfem.vis.draw_nodal_values**

```python
calfem.vis.draw_nodal_values(nodeVals, coords, edof, dofsPerNode, elType, clim=None,
                            axes=None, axesAdjust=True, doDrawMesh=True, title=None)
```

Draws scalar nodal values in 2D or 3D. Returns the Mesh object that represents the mesh. Parameters:

- **nodeVals** - An N-by-1 array or a list of scalars. The Scalar values at the nodes. \(nodeVals[i]\) should be the value of node i
- **coords** - An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates of node i.
- **edof** - An E-by-L array. Element topology. \((E \text{ is the number of elements and } L \text{ is the number of dofs per element})\)
- **dofsPerNode** - Integer. Dofs per node. \(elType\) - Integer. Element Type. See Gmsh manual for details. Usually 2 for triangles or 3 for quadrangles.
- **clim** - 2-tuple. Colorbar limits (min, max). Defines the value range of the colorbar. Defaults to None, in which case min/max are set to min/max of nodeVals.
- **axes** - Visvis Axes. The Axes where the model will be drawn. If unspecified the current Axes will be used, or a new Axes will be created if none exist.
- **axesAdjust** - Boolean. True if the view should be changed to show the whole model. Default True.
- **doDrawMesh** - Boolean. True if mesh wire should be drawn. Default True. **title** - String. Changes title of the figure. Default “Node Values”.

**calfem.vis.eldraw2**

```python
calfem.vis.eldraw2(ex, ey, plotpar, elnum)
calfem.vis.eldraw2(ex, ey, plotpar)
calfem.vis.eldraw2(ex, ey)
```

3.6. Visualisation functions
**PURPOSE** Draw the undeformed 2D mesh for a number of elements of the same type. Supported elements are:

1) -> bar element 2) -> beam el. 3) -> triangular 3 node el. 4) -> quadrilateral 4 node el. 5) -> 8-node isopar. elemen

**INPUT**

- `ex, ey`……. `nen`: number of element nodes
- `nel`: number of elements
- `plotpar`=[ linetype, linecolor, nodemark]
  - linetype=1 -> solid linecolor=1 -> black 2 -> dashed 2 -> blue 3 -> dotted 3 -> magenta
  - 4 -> red
  - nodemark=1 -> circle 2 -> star 0 -> no mark

`elnum=edof(:,1) ;` i.e. the first column in the topology matrix

Rem. Default is solid white lines with circles at nodes.

calfem.vis.eldraw2_mpl(ex, ey, plotpar=[1, 2, 1], elnum=[]) eldraw2(ex,ey,plotpar,elnum) eldraw2(ex,ey)
ex, ey Element coordinates ev Element values (scalar) plotpar (not implemented yet)

calfem.vis.error(msg)
    Log error message

calfem.vis.figure(figure=None, show=True)
    Create a visvis figure with extras.

calfem.vis.figureClass()
    Return visvis Figure class.

calfem.vis.figure_class()
    Return visvis Figure class.

calfem.vis.gca()
    Get current axis of the current visvis figure.

calfem.vis.getColorbar(axes=None)
    Returns the Colorbar. If axes is None the colorbar in the current axes will be found. If several colorbars exists in the axes the first found will be returned If no colorbar is found None is returned.

calfem.vis.get_color_bar(axes=None)
    Returns the Colorbar. If axes is None the colorbar in the current axes will be found. If several colorbars exists in the axes the first found will be returned If no colorbar is found None is returned.

calfem.vis.info(msg)
    Log information message

calfem.vis.showAndWait()
    Show visvis windows and enter application loop.

calfem.vis.showGrid(flag=True)
    Show grid.

calfem.vis.show_and_wait()
    Show visvis windows and enter application loop.

calfem.vis.show_grid(flag=True)
    Show grid.

calfem.vis.subplot(*args)
    Create a visvis subplot.
CHAPTER 4

Indices and tables

- genindex
- modindex
- search
Python Module Index

C

calfem.core, 17

calfem.geometry, 29

calfem.mesh, 37

calfem.ui, 39

calfem.utils, 39

calfem.vis, 41
A
add_label() (in module calfem.vis), 41
add_text() (in module calfem.vis), 42
addBSpline() (calfem.geometry.Geometry method), 29
addCircle() (calfem.geometry.Geometry method), 29
addEllipse() (calfem.geometry.Geometry method), 30
addLabel() (in module calfem.vis), 41
addPoint() (calfem.geometry.Geometry method), 30
addPoints() (calfem.geometry.Geometry method), 31
addRuledSurface() (calfem.geometry.Geometry method),
addSpline() (calfem.geometry.Geometry method), 31
addSplines() (calfem.geometry.Geometry method), 31
addStructuredSurface() (calfem.geometry.Geometry method), 31
addStructuredVolume() (calfem.geometry.Geometry method), 31
addSurface() (calfem.geometry.Geometry method), 32
addText() (in module calfem.vis), 41
addVolume() (calfem.geometry.Geometry method), 32
class() (calfem.geometry.Geometry method), 32
appInstance() (in module calfem.ui), 39
applybc() (in module calfem.utils), 39
applybc3D() (in module calfem.utils), 39
applyforce() (in module calfem.utils), 40
applyforce3D() (in module calfem.utils), 40
applyforcetotal() (in module calfem.utils), 40
applyforcetotal3D() (in module calfem.utils), 40
applyTractionLinearElement() (in module calfem.utils),
appInstance() (in module calfem.ui), 39
applybc() (in module calfem.utils), 39
applybc3D() (in module calfem.utils), 39
applyforce() (in module calfem.utils), 40
applyforce3D() (in module calfem.utils), 40
applyforcetotal() (in module calfem.utils), 40
applyforcetotal3D() (in module calfem.utils), 40
applyTractionLinearElement() (in module calfem.utils),
asem() (in module calfem.core), 17
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
calfem.core (module), 17
calfem.geometry (module), 29
calfem.mesh (module), 37
calfem.ui (module), 39
callem.utils (module), 39
callem.vis (module), 41
camera3d() (in module calfem.vis), 42
circle() (calfem.geometry.Geometry method), 33
clf() (in module calfem.vis), 42
close_all() (in module calfem.vis), 42
closeAll() (in module calfem.vis), 42
color_bar() (in module calfem.vis), 42
colorBar() (in module calfem.vis), 42
coordxtr() (in module calfem.core), 22
create() (calfem.mesh.GmshMeshGenerator method), 37
createdofs() (in module calfem.core), 22
circle() (calfem.geometry.Geometry method), 33
clf() (in module calfem.vis), 42
close_all() (in module calfem.vis), 42
closeAll() (in module calfem.vis), 42
color_bar() (in module calfem.vis), 42
colorBar() (in module calfem.vis), 42
coordxtr() (in module calfem.core), 22
class() (calfem.geometry.Geometry method), 32
appInstance() (in module calfem.ui), 39
applybc() (in module calfem.utils), 39
applybc3D() (in module calfem.utils), 39
applyforce() (in module calfem.utils), 40
applyforce3D() (in module calfem.utils), 40
applyforcetotal() (in module calfem.utils), 40
applyforcetotal3D() (in module calfem.utils), 40
applyTractionLinearElement() (in module calfem.utils),
asem() (in module calfem.core), 17
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
class() (calfem.geometry.Geometry method), 32
appInstance() (in module calfem.ui), 39
applybc() (in module calfem.utils), 39
applybc3D() (in module calfem.utils), 39
applyforce() (in module calfem.utils), 40
applyforce3D() (in module calfem.utils), 40
applyforcetotal() (in module calfem.utils), 40
applyforcetotal3D() (in module calfem.utils), 40
applyTractionLinearElement() (in module calfem.utils),
asem() (in module calfem.core), 17
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
beam2d() (in module calfem.core), 19
beam2e() (in module calfem.core), 19
beam2g() (in module calfem.core), 19
beam2gs() (in module calfem.core), 19
beam2s() (in module calfem.core), 20
beam2t() (in module calfem.core), 20
beam2ts() (in module calfem.core), 20
beam2w() (in module calfem.core), 21
beam2ws() (in module calfem.core), 21
drawGeometry() (in module calfem.vis), 43
drawMesh() (in module calfem.vis), 44
drawNodalValues() (in module calfem.vis), 44

e
  effmises() (in module calfem.core), 22
eldraw2() (in module calfem.vis), 47
eldraw2_mpl() (in module calfem.vis), 48
eldraw2_old() (in module calfem.vis), 48
eliso2_old() (in module calfem.vis), 48
ellipse() (calfem.geometry.Geometry method), 33
elval2() (in module calfem.vis), 48
error() (in module calfem.core), 23
error() (in module calfem.vis), 49
extractEldisp() (in module calfem.core), 23

F

figure() (in module calfem.vis), 49
figure_class() (in module calfem.vis), 49
figureClass() (in module calfem.vis), 49
flw2i4e() (in module calfem.core), 23
flw2i4s() (in module calfem.core), 23
flw2i8e() (in module calfem.core), 23
flw2i8s() (in module calfem.core), 24
flw2qte() (in module calfem.core), 24
flw2qts() (in module calfem.core), 24
flw2te() (in module calfem.core), 24
flw2ts() (in module calfem.core), 25
flw3i8e() (in module calfem.core), 25
flw3i8s() (in module calfem.core), 25

G

gca() (in module calfem.vis), 49
Geometry (class in calfem.geometry), 29
get_color_bar() (in module calfem.vis), 49
ggetColorbar() (in module calfem.vis), 49
getPointCoords() (calfem.geometry.Geometry method), 34
GmshMesh (in module calfem.mesh), 37
GmshMeshGenerator (class in calfem.mesh), 37

H

hooke() (in module calfem.core), 26

I

info() (in module calfem.core), 26
info() (in module calfem.vis), 49

L

line() (calfem.geometry.Geometry method), 34
loadUiWidget() (in module calfem.ui), 39

P

plani4e() (in module calfem.core), 26
planq() (in module calfem.core), 26
plante() (in module calfem.core), 26
plantf() (in module calfem.core), 27
plants() (in module calfem.core), 27
platre() (in module calfem.core), 27
point() (calfem.geometry.Geometry method), 34
pointsOnCurves() (calfem.geometry.Geometry method), 34

R

readFloat() (in module calfem.utils), 40
readInt() (in module calfem.utils), 40
readSingleFloat() (in module calfem.utils), 40
readSingleInt() (in module calfem.utils), 41
removeCurve() (calfem.geometry.Geometry method), 35
removePoint() (calfem.geometry.Geometry method), 35
removeSurface() (calfem.geometry.Geometry method), 35
removeVolume() (calfem.geometry.Geometry method), 35
ruled_surface() (calfem.geometry.Geometry method), 35
ruledSurface() (calfem.geometry.Geometry method), 35

S

scalfact2() (in module calfem.utils), 41
setCurveMarker() (calfem.geometry.Geometry method), 35
setPointMarker() (calfem.geometry.Geometry method), 35
setSurfaceMarker() (calfem.geometry.Geometry method), 35
setVolumeMarker() (calfem.geometry.Geometry method), 35
show_and_wait() (in module calfem.vis), 49
show_grid() (in module calfem.vis), 49
showAndWait() (in module calfem.vis), 49
showGrid() (in module calfem.vis), 49
solveq() (in module calfem.core), 27
spline() (calfem.geometry.Geometry method), 35
spring1e() (in module calfem.core), 28
spring1s() (in module calfem.core), 28
spsolveq() (in module calfem.core), 28
statcon() (in module calfem.core), 28
stress2nodal() (in module calfem.core), 28
structured_surface() (calfem.geometry.Geometry method), 36
structured_volume() (calfem.geometry.Geometry method), 36
structuredSurface() (calfem.geometry.Geometry method), 36
structuredVolume() (calfem.geometry.Geometry method), 36
stuffOnSurfaces() (calfem.geometry.Geometry method), 36
stuffOnVolumes() (calfem.geometry.Geometry method), 37
subplot() (in module calfem.vis), 49
surface() (calfem.geometry.Geometry method), 37
T
trimesh2d() (in module calfem.mesh), 38
V
volume() (calfem.geometry.Geometry method), 37
W
which() (in module calfem.utils), 41