Formulation of the pull-out problem using index notation

Yingxiong Li, Rostislav Chudoba

Institute of Strutural Concrete, RWTH Aachen Univserstiy,
Aachen, Germany

1 Background

The Python package NumPy provides very useful tools for scientific computing, such as the multidimensional array and summation tool over spatial dimensions. Using these tools, mathematic problems formulated using index notation can be efficiently and consistently implemented. We use the pull-out problem as a demonstrative example for this kind of implementation. The pull-out test is a common procedure to determine the property of the bond interface in composites. The mathematical idealization of the direct pull-out is formulated as an initial boundary value problem reflecting equilibrium and compatibility conditions, as well as the material behavior of bond, matrix and reinforcement. In the considered case of cementitious composites with a negligible shear deformation the problem can be regarded as one-dimensional. The bond between the reinforcement and the matrix is assumed in the form of a general nonlinear function $\tau(s)$ coupling the shear stress τ with the slip s. Then the pull-out problem can be simplified as shown in Fig. 1.

1.1 Constitutive laws

The constitutive laws of the reinforcement and the matrix are given as

$$\sigma_{\mathrm{f}} = D_{\mathrm{f}}(\varepsilon_{\mathrm{f}}) \tag{1}$$

$$\sigma_{\mathrm{m}} = D_{\mathrm{m}}(\varepsilon_{\mathrm{m}}), \tag{2}$$

where σ_{f} and σ_{m} are the reinforcement stress and the matrix stress, ε_{f} and ε_{m} are the reinforcement strain and the matrix strain, respectively. And the constitutive law of the bond interface reads

$$\tau = \tau(s). \tag{3}$$

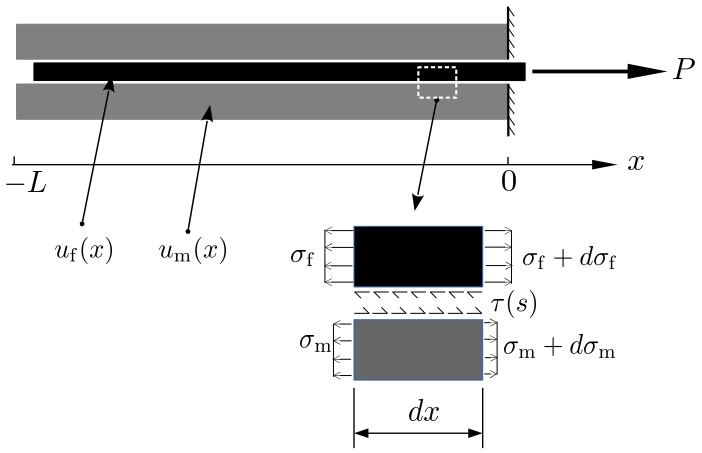

Figure 1: A simplified mechanical model of the pull-out problem

1.2 Kinematic

The strains in the matrix and reinforcement are given as

$$\varepsilon_{\mathrm{f}} = u_{\mathrm{f,x}} \tag{4}$$

$$\varepsilon_{\mathrm{m}} = u_{\mathrm{m,x}}, \tag{5}$$

where the index $(.)_{,x}$ denotes the derivative with respect to the spatial coordinate x, u_{f} and u_{m} are the reinforcement and matrix displacements, respectively. The relative slip in the bond interface can be expressed as

$$s = u_{\mathrm{f}} - u_{\mathrm{m}}. \tag{6}$$

1.3 Equilibrium

The equilibrium of an infinitesimal segment of the reinforcement leads to

$$A_{\mathrm{f}}\sigma_{\mathrm{f},x} - p\tau(s) = 0, \tag{7}$$

where A_{f} and p are the cross-sectional area and the perimeter of the reinforcement, receptively. Similarly, the equilibrium of the matrix can be expressed as

$$A_{\mathrm{m}}\sigma_{\mathrm{m},x} + p\tau(s) = 0, \tag{8}$$

in which A_{m} is the cross-sectional area of the matrix.

2 Weak formulation of the pull-out problem

In order to provide a more general model of the pull-out problem defined by the differential Eqs. (7) and (8), we reformulate the boundary value problem in a

weak form within the domain $\Omega := [-L, 0]$ shown in Fig. 1 . The essential and natural boundary conditions are specified as

$$\begin{aligned} u_{\mathrm{m}} &= \bar{u}_{\mathrm{m}}(\theta) \ \text{ on } \ \Gamma_{u_{\mathrm{m}}}, & \sigma_{\mathrm{m}} A_{\mathrm{m}} &= \bar{t}_{\mathrm{m}}(\theta) \ \text{ on } \ \Gamma_{t_{\mathrm{m}}}, \\ u_{\mathrm{f}} &= \bar{u}_{\mathrm{f}}(\theta) \ \text{ on } \ \Gamma_{u_{\mathrm{f}}}, & \sigma_{\mathrm{f}} A_{\mathrm{f}} &= \bar{t}_{\mathrm{f}}(\theta) \ \text{ on } \ \Gamma_{t_{\mathrm{f}}}, \end{aligned} \tag{9}$$

in which $\Gamma_{u_{\mathrm{m}}}, \Gamma_{t_{\mathrm{m}}}, \Gamma_{u_{\mathrm{f}}}$ and $\Gamma_{t_{\mathrm{f}}}$ denote the boundaries where the corresponding boundary conditions are applied. The essential boundary conditions $\bar{u}_{\mathrm{mf}}$ and the natural boundary conditions $\bar{t}_{\mathrm{mf}}$ are prescribed as functions of the intrinsic time variable θ which is used to control the loading history. Denoting the integration of the product of two functions u, v over the domain V as $(u, v)_V$, the weak formulation can be expressed as

$$\begin{aligned} &(\delta u_{\mathrm{m}}, A_{\mathrm{m}}\sigma_{\mathrm{m},x} + p\tau)_{\Omega} &+ \quad &(\delta u_{\mathrm{f}}, A_{\mathrm{f}}\sigma_{\mathrm{f},x} - p\tau)_{\Omega} + \\ &(\delta u_{\mathrm{m}}, -A_{\mathrm{m}}\sigma_{\mathrm{m}} + \bar{t}_{\mathrm{m}}(\theta))_{\Gamma_{t_m}} &+ \quad &(\delta u_{\mathrm{f}}, -A_{\mathrm{f}}\sigma_{\mathrm{f}} + \bar{t}_{\mathrm{f}}(\theta))_{\Gamma_{t_f}} + \\ &(\delta u_{\mathrm{m}}, u_{\mathrm{m}} - \bar{u}_{\mathrm{m}}(\theta))_{\Gamma_{u_{\mathrm{m}}}} &+ \quad &(\delta u_{\mathrm{f}}, u_{\mathrm{f}} - \bar{u}_{\mathrm{f}}(\theta))_{\Gamma_{u_f}} \\ &= 0. \end{aligned} \tag{10}$$

Here δu_{m} and δu_{f} denote the test functions that are assumed to have continuous first derivatives, are L^2 integrable and to implicitly fulfill the essential boundary conditions. Since $\delta u_{\mathrm{m}} = 0$ on $\Gamma_{u_{\mathrm{m}}}$ and $\delta u_{\mathrm{f}} = 0$ on $\Gamma_{u_{\mathrm{f}}}$, the last two terms in Eq. (16) vanish. Using integration by parts, the orders of the the stress derivatives $\sigma_{\mathrm{m},x}$ and $\sigma_{\mathrm{f},x}$ in the above integration can be reduced as follows

$$\left(\delta u_{(.)}, A_{(.)}\sigma_{(.),x}\right)_{\Omega} = \left(\delta u_{(.)}, A_{(.)}\sigma_{(.)}\right)_{\Gamma} - \left(\delta u_{(.),x}, A_{(.)}\sigma_{(.)}\right)_{\Omega} \tag{11}$$

Then, by substituting Eq. (17) for both matrix and reinforcement into Eq. (16), the following variational formulation of the pull-out problem is obtained

$$\begin{aligned} (\delta u_{\mathrm{m},x}, A_{\mathrm{m}}\sigma_{\mathrm{m}})_{\Omega} \quad &+ \quad (\delta u_{\mathrm{f}} - \delta u_{\mathrm{m}}, p\tau)_{\Omega} \quad + \\ (\delta u_{\mathrm{f},x}, A_{\mathrm{f}}\sigma_{\mathrm{f}})_{\Omega} \quad &- \quad (\delta u_{\mathrm{m}}, \bar{t}_{\mathrm{m}}(\theta))_{\Gamma_{tm}} \quad - \\ (\delta u_{\mathrm{f}}, \bar{t}_{\mathrm{f}}(\theta))_{\Gamma_{tf}} \quad &= \quad 0. \end{aligned} \tag{12}$$

3 Index notation based finite element formulation

The implementation approach utilizes two new features provided by current tools for scientific computing that have become available during the recent decade:

- multi-dimensional arrays, and
- highly efficient multi-dimensional index operators based on the Einstein summation rule.

In order to exploit these two concepts, we need to formulate the finite element model of the direct pull-out problem in index form.

3.1 Index notation

To allow for the Einstein summation rule not only for spatial dimensions, nodes, elements and integration points, but also for individual material components, we replace the subscripts $(.)_{\mathrm{m}}$ and $(.)_{\mathrm{f}}$ with the indexes 1 and 2. In the present case of the pull-out problem, index 1 represents the matrix m and index 2 the reinforcement f. The convention implies summation over repeated indexes, for example the slip between the matrix and reinforcement given in Eq. (6) can be rewritten as

$$s = -u_{\mathrm{m}} + u_{\mathrm{f}} = (-1)^1 u_1 + (-1)^2 u_2 = (-1)^c u_c \tag{13}$$

with $c = 1, 2$. To avoid ambiguity, we note that here the upper index c means both exponent and index, alternatively, $(-1)^c$ can be understood as the c-th component of the vector $\{-1, 1\}$. Using this convention, the variational formulation in Eq. (18) can be rewritten as

$$(\delta u_{c,x}, F_c)_\Omega + (\delta u_c \, (-1)^c, p\tau)_\Omega - (\delta u_c, \bar{t}_c(\theta))_{\Gamma_{t_c}} = 0, \tag{14}$$

where F_c is the component of the vector $\{A_1\sigma_1, A_2\sigma_2\}$, and the term $\delta u_c(-1)^c = \delta(u_2 - u_1) = \delta s$ renders the virtual slip between the components.

3.2 Weak formulation of the multi-layer problem

In order to provide a more general model of the pull-out problem defined by the differential Eqs. (7) and (8) we construct a weak form of the boundary value problem within the domain $\Omega := [0, L]$ shown in Fig. 1 . The corresponding essential and natural boundary conditions are specified as

$$u_c \;=\; \bar{u}_c(\theta) \;\text{ on }\; \Gamma_{u_c} \quad \text{and} \quad \sigma_c A_c \;=\; \bar{t}_c(\theta) \;\text{ on }\; \Gamma_{t_c} \tag{15}$$

Denoting the integration of the product of the terms u, v over V as $(u, v)_V$, the weak formulation can be expressed as

$$(\delta u_c, A_c\sigma_{c,x} + (-1)^c \, p_c\tau)_\Omega + (\delta u_c, u_c - \bar{u}_c(\theta))_{\Gamma_{u_c}} + (\delta u_c, -A_c\sigma_c + \bar{t}_c(\theta))_{\Gamma_{t_c}} = 0 \tag{16}$$

Since $\delta u_c = 0$ on Γ_{u_c}, the second and fifth terms in Eq. (16) vanish. Using integration by parts, the orders of the the stress derivatives $\sigma_{c,x}$ can be reduced as follows

$$(\delta u_c, A_c\sigma_{c,x})_\Omega = -\,(\delta u_{c,x}, A_c\sigma_c)_\Omega + (\delta u_c, A_c\sigma_c)_\Gamma \tag{17}$$

Finally, by substituting Eq. (17) for both matrix and fibers into Eq. (16), the following variational formulation of the pull-out problem is obtained

$$(\delta u_{c,x}, A_c\sigma_c)_\Omega + (\delta u_c \, (-1)^c, p_c\tau)_\Omega - (\delta u_c, \bar{t}_c(\theta))_{\Gamma_{t_c}} = 0. \tag{18}$$

Note that the term $\delta u_c(-1)^c = \delta(u_2 - u_1) = \delta s$ renders the virtual slip between the components.

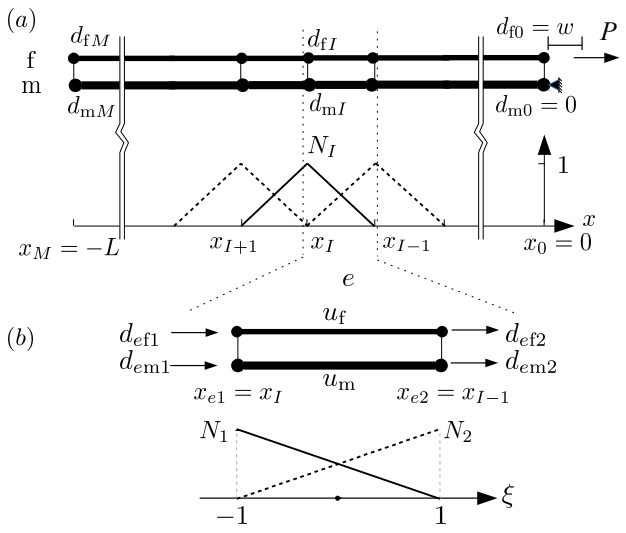

Figure 2: Index-based finite element discretization

3.3 Finite element discretization

The displacement field of the material component c is approximated using the shape functions shown in Fig. 2a,

$$u_c = N_I \; d_{cI}, \; \delta u_c = N_I \; \delta d_{cI}$$

with the index $I = 1, 2, \ldots, M$ representing the I-th node.

For brevity, we use globally defined shape functions, centered at the node I and covering two neighboring elements throughout this appendix. In the standard finite element implementation, an element-wise, parametric specification of the shape function is usual. The present mathematical expressions related to node I can be easily transformed into such kind of element-centered expressions using an index function $[e, i] \rightarrow I$ that defines the mapping of the local elemental degrees of freedom to the global nodal degrees of freedom (Fig. 2b). This mapping function was used in the implementation to obtain an efficient, "loopless" assembly procedure of system matrix and force vectors.

The strain field and virtual strain field in each material component c are then expressed as

$$u_{c,x} = B_I \; d_{cI}, \; \delta u_{c,x} = B_I \; \delta d_{cI} \tag{19}$$

where $B_I = N_{I,x}$, x is the global coordinate. The slip field is approximated as

$$s = (-1)^c u_c = (-1)^c \, N_I d_{cI}, \tag{20}$$

so that, after substitutions, Eq. (14) reads

$$\delta d_{cI} \, (B_I, F_c)_\Omega + \delta d_{cI} \, ((-1)^c \, N_I, p\tau)_\Omega - \delta d_{cI} \, (N_I, \bar{t}_c(\theta))_{\Gamma_{t_c}} = 0. \tag{21}$$

Since δd_{cI} is arbitrary, and $(N_I, \bar{t}_c(\theta))_{\Gamma_{t_c}}$ can be simplified to nodal loads as $\bar{t}_{cI}(\theta)$, Eq. (21) can be reduced to

$$R_{cI}^{(\theta)} = (B_I, F_c)_\Omega + ((-1)^c N_I, p\tau)_\Omega - \bar{t}_{cI}(\theta) = 0, \tag{22}$$

with $R_{cI}^{(\theta)}$ representing the residuum.

3.4 Iterative solution algorithm

In case of nonlinear material behavior assumed either for the matrix, reinforcement or bond, Eq. (22) needs to be linearized using the Taylor expansion:

$$R_{cI}^{(\theta)}(d_{dJ}^{(k)}) \approx R_{cI}^{(\theta)}(d_{dJ}^{(k-1)}) + \left.\frac{\partial R_{cI}^{(\theta)}}{\partial d_{dJ}}\right|_{d_{dJ}^{(k-1)}} \Delta d_{dJ}^{(k)} = 0. \tag{23}$$

Since we do not consider geometrical nonlinearity and the nodal loads $\bar{t}_{cI}$ are assumed independent of the displacements, the derivative in Eq. (23) can be written as

$$\frac{\partial R_{cI}^{(\theta)}}{\partial d_{dJ}} = \left(B_I, \frac{\partial F_c}{\partial d_{dJ}}\right)_\Omega + \left((-1)^c N_I, p\frac{\partial \tau}{\partial d_{dJ}}\right)_\Omega. \tag{24}$$

In case of nonlinear material behavior of the components c, the derivative of the cross-sectional component force F_c with respect to d_{cI} reads

$$\frac{\partial F_c}{\partial d_{dJ}} = \frac{\partial F_c}{\partial \varepsilon_d}\frac{\partial \varepsilon_d}{\partial d_{dJ}} = \frac{\partial F_c}{\partial \varepsilon_d} B_J \tag{25}$$

Similarly, for nonlinear bond-slip law, the derivative of shear flow with respect to the nodal displacement can be written as

$$\frac{\partial \tau}{\partial d_{dJ}} = \frac{\partial \tau}{\partial s}\frac{\partial s}{\partial d_{dJ}} = \frac{\partial \tau}{\partial s} N_J (-1)^d. \tag{26}$$

Thus, the tangential stiffness in Eq. (23) reads

$$\begin{aligned} \left.\frac{\partial R_{cI}^{(\theta)}}{\partial d_{dJ}}\right|_{d_{dJ}^{k-1}} &= K_{cIdJ}^{(\theta,k-1)} = \left(B_I, B_J \left.\frac{\partial F_c}{\partial \varepsilon_d}\right|_{d_{dJ}^{k-1}}\right)_\Omega \\ &+ (-1)^{c+d} p \left(N_I, N_J \left.\frac{\partial \tau}{\partial s}\right|_{d_{dJ}^{k-1}}\right)_\Omega \end{aligned}. \tag{27}$$

The integrals in Eq. (27) are evaluated using the numerical quadrature,

$$(u, v)_\Omega = w_i X_i, \tag{28}$$

in which w_i is the weight factor and $X_i = u(\xi_i)v(\xi_i)$ the value of the product uv at the integration point ξ_i. Finally, by rewriting Eq. (23) the following incremental form of equilibrium is obtained

$$K_{cIdJ}^{(\theta,k-1)} \Delta d_{dJ}^{(k)} = -R_{cI}^{(\theta)}(d_{dJ}^{(k-1)}). \tag{29}$$

3.5 Linear case

As the code is for demonstrative purpose, for simplicity we consider a simple case where all the materials are assumed to be linear elastic. Alternatively, it can be understood as a particular linearized step in the iterative solution algorithm. Now we change the finite element formulation from node-centered (Fig. 2a) to element-centered (Fig. 2b).

$$\begin{aligned} u_{ce} &= N_i d_{cI(e,i)} \\ \delta u_{ce} &= N_i \delta d_{cI(e,i)} \end{aligned} \tag{30}$$

with index $i = 1, 2$ representing a node of the element. The shape functions are given as

$$\begin{aligned} N_1 &= \frac{1}{2}(1-\xi), \\ N_2 &= \frac{1}{2}(1+\xi). \end{aligned} \tag{31}$$

The strain field in each material component is then expressed as

$$u_{ce,x} = B_i \; d_{cI}(e,i) \tag{32}$$

$$\delta u_{ce,x} = B_i \; \delta d_{cI}(e,i) \tag{33}$$

where

$$B_i = N_{i,x} = N_{i,\xi} J(\xi)^{-1} \tag{34}$$

with J being the Jacobian. Assuming a linear elastic constitutive law of the components the derivatives of σ_c with respect to ε_d read

$$\frac{\partial \sigma_c}{\partial \varepsilon_d} = \delta_{cd} E_c \tag{35}$$

and linear bond leads to

$$\frac{\partial \tau}{\partial s} = G. \tag{36}$$

Finally, substituting Eqs. (34) to (36) into Eq. (27), the following incremental form of the element stiffness matrix is obtained,

$$\begin{aligned} K_{cIdJei} = \Bigg[& A_c B_{I(e,i)M(e,m)} B_{J(e,j)M(e,m)} \left. \frac{\partial \sigma_c}{\partial \varepsilon_d} \right|_{d_{dJ}^{k-1}} \\ & + (-1)^{c+d} \, p N_{I(e,i)M(e,m)} N_{J(e,j)M(e,m)} \left. \frac{\partial \tau}{\partial s} \right|_{d_{dJ}^{k-1}} \Bigg] w_{M(e,m)} \left| J \right|_{M(e,m)} . \end{aligned} \tag{37}$$

3.6 Implementation

```
import numpy as np                                                              1
import sympy as sp                                                              2
from conjugate_gradient import cg                                               3
from constraint import constraint                                               4
```

Lines 1-2 import the required packages `numpy` and `sympy`, line 3 and line 4 import the iterative solver for linear systems using the conjugate method and the function to apply constraints, respectively.

```
# generate shape functions with sympy                                           5
xi_ = sp.symbols('xi')                                                          6
N_i_xi = sp.Matrix([(1. - xi_) / 2.0, (1 + xi_) / 2.0, ], dtype=np.float_)      7
dN_i_xi = N_i_xi.diff('xi')                                                     8
```

Lines 5-8 define the shape functions given in Eq. (31) using `sympy`. `sympy` is a Python library for symbolic mathematics, the advantage of using `sympy` is that one can generate the derivatives of the shape functions automatically, which simplifies the calculation of the displacement-strain matrix, see line 8. Although in the present simple case this feature seems to be not so advantageous, however if higher order shape functions are used, it would be much more convenient to get the derivative of the of the shape function using `sympy.diff` than to derive it analytically.

```
# numerical integration points (IP) and weights                                 9
xi_m = np.array([[-1], [1]])                                                    10
w_m = np.array([1, 1])                                                          11
# the values of the shape functions and their derivatives at the IPs            12
N_mi = np.array([N_i_xi.subs(xi_, xi) for xi in xi_m], dtype=np.float_)          13
dN_mei = np.array([[dN_i_xi.subs(xi_, xi)]for xi in xi_m], dtype=np.float_)      14
```

Lines 9-14 prepare the relevant arrays for the numerical integration scheme.

```
# element specifications                                                        15
n_dim = 1  # dimension of the element                                           16
n_c = 2  # number of the composite components                                   17
n_e_n = 2  # number of nodes per element                                        18
n_e_dof = n_e_n * n_c * n_dim  # number of DOFs per element                     19
# mesh specifications                                                           20
n_e = 5  # number of element used                                               21
n_n_tot = n_e + 1  # total number of nodes                                      22
n_dof_tot = n_c * n_n_tot * n_dim  # total number of DOFs                       23
```

Lines 15-23 specify the parameters of the element and the mesh. The total number of elements is assigned in line 21, if a finer mesh is desired, one can increase the value of `n_e`.

```
# array handling the interaction of the material components                     24
DELTA_cd = np.identity(n_c)                                                     25
c = np.arange(n_c)                                                              26
c1 = c + 1                                                                      27
SWITCH_cd = np.power(-1.0, c1[np.newaxis, :] + c1[:, np.newaxis])               28
```

Lines 24-28 generate the supporting arrays, such as the Kronecker delta used in Eq. (35). `SWITCH _cd` handles the term $(-1)^{c+d}$ in Eq. (37).

```
# geometrical and material parameters                                           29
A_c = [1., 1.]  # the cross-sectional area of each material component          30
```

```
E_c = [1., 1.]  # the modulus of each material component
G = 0.1  # the shear modulus of the bond interface
p = 1.  # the perimeter of the interface
L_x, L_y, L_z = 10., 1., 1.  # dimension of the structure
```

Lines 29-34 specify the material and geometrical parameters, we note that in the present case a one dimensional problem is considered thus only `L_x` is relevant, `L_y` and `L_z` are kept for future extension in higher dimensions.

```
# Geomerty approximation
I = np.arange(n_n_tot)
x_grid = np.array(np.mgrid[0:L_x:complex(0, n_n_tot), ])
x_Id = np.einsum('d...->...d', x_grid).reshape(-1, 1)
I_Ei = np.c_[I[:-1], I[1:]]
x_Eid = x_Id[I_Ei]
J_Emde = np.einsum('mei,Eid->Emed', dN_mei, x_Eid)
J_det_Em = np.linalg.det(J_Emde)
J_inv_Emed = np.linalg.inv(J_Emde)
# Quadratic forms
dN_Eimd = np.einsum('mei,Eide->Eimd', dN_mei, J_inv_Emed)
BB_ECidDjf = np.einsum('m, CD, C, C, Eimd,Ejmf,Em->ECidDjf',
                       w_m, DELTA_cd, A_c, E_c, dN_Eimd, dN_Eimd, J_det_Em)
NN_ECidDjf = np.einsum('m, CD,mi,mj,,,Em->ECiDj',
                       w_m, SWITCH_cd, N_mi, N_mi, p, G, J_det_Em)
BB_Eij = BB_ECidDjf.reshape(-1, n_e_dof, n_e_dof)
NN_Eij = NN_ECidDjf.reshape(-1, n_e_dof, n_e_dof)
K_Eij = BB_Eij + NN_Eij
# Multilayer expansion
C = np.arange(n_c) * n_n_tot
I_C = I[np.newaxis, :] + C[:, np.newaxis]
I_ECi = np.vstack([[I_C[:, :-1], I_C[:, 1:]]]).T.reshape(-1, n_e_dof)
```

Lines 35-56 evaluate the stiffness matrix. The stiffness matrix is stored in the 3-dimensional array `I_ECi` in such a way that `I_ECi[i]` returns the element stiffness matrix.

```
# apply constraints and solve
F_ext = np.zeros((n_dof_tot,), np.float_)
K_Eij, F_ext = constraint(n_dof_tot / 2 - 1, 0., K_Eij, I_ECi, F_ext)
K_Eij, F_ext = constraint(n_dof_tot - 1, 0.01, K_Eij, I_ECi, F_ext)
d_I = cg(K_Eij, F_ext, I_ECi)
print 'd_I', d_I
```

Line 58 specify the external force vector, in the present case no external force is applied on the structure. Line 59 fixes the DOF 5, and Line 60 applies a displacement constraint of the value of 0.01 on the DOF 11. The displacement constraints are applied through modifying the stiffness matrix and the external force vector. The boundary conditions are shown in Fig. 3. Line 61 solves the system equation. By using the conjugate gradient method, the stiffness matrix can be kept in the stacked form, the assembly of the global matrix as a sparse matrix is not required.

```
# post processing
import matplotlib.pyplot as plt
plt.plot(x_Id[:, 0], d_I.reshape(2, -1).T)
plt.xlabel('x')
plt.ylabel('displacement')
plt.show()
```

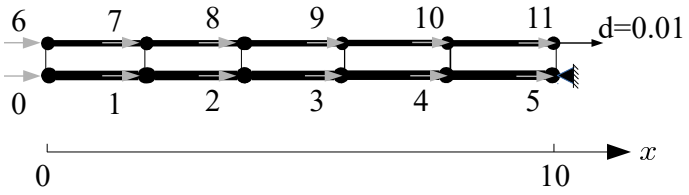

Figure 3: The boundary condition used in the code

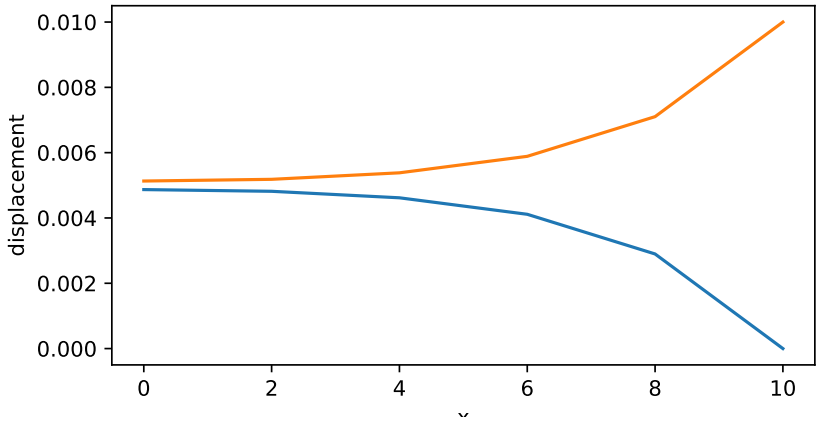

Figure 4: The displacement fields in both components

Line 63-68 visualize the displacement filed in both material components using `matplotlib`, as shown in Fig. 4.