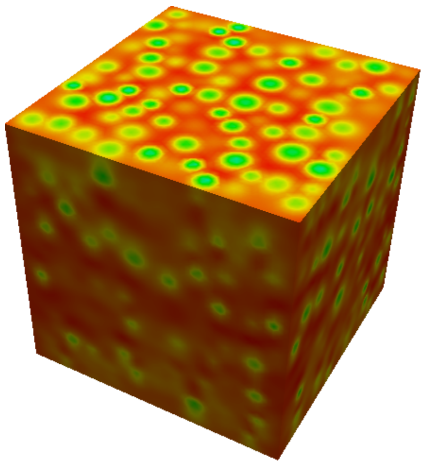
μMECH

An open source C/C++ library of analytical solutions to micromechanical problems

Theory manual & Program documentation



Jan Novák[1][2][3]

March 29, 2014

[1]Correspoding address: novakj@cml.fsv.cvut.cz

[2]**C**zech **T**echnical **U**niversity in Prague, Faculty of Civil Engineering, Department of Mechanics

[3]**U**niversity of **G**lasgow, Faculty of Engineering, Department of Civil Engineering

Project partners

University of Glasgow | Department of Civil Engineering

VIA VERITAS VITA

GRPE

GLASGOW RESEARCH PARTNERSHIP IN ENGINEERING

Centre for Integrated DEsign of Advanced Structures

GAČR

Contents

1 Introduction **4**

2 Theory manual **5**

2.1 Single inhomogeneity problem . . . 5

2.2 Multiple inhomogeneity problem . . . 7

2.2.1 Multiple inclusion problem via Self-balancing algorithm . . . 8

3 Tutorial **10**

3.1 Installation . . . 10

3.2 Input file format . . . 10

3.3 Available functions . . . 14

3.3.1 Interface - C++ . . . 14

3.3.2 Interface - C . . . 20

3.4 Implementation examples . . . 25

3.4.1 Implementation via C++ interface . . . 25

3.4.2 Implementation via C interface . . . 27

3.5 Important comments . . . 29

Preface

This tutorial is freely distributed as the complement of the μ**MECH** C/C++ library under following regulations:

We hope you enjoyed this tutorial as well as μ**MECH** *code itself and found it worth to cite our work. If this the case, please cite either one or more of the following items:*

- Novák, J. and Kaczmarczyk, Ł. and Grassl, P. and Zeman, J. and Pearce, C. J., A micromechanics-enhanced finite element formulation for modelling heterogeneous materials. Computer Methods in Applied Mechanics and Engineering 201:53–64, 2012, 1103.5633.

Chapter 1

Introduction

What's the **μMECH** *library about...*

The library **μMECH** was principally designed as a subclass of a finite element package. It provides subroutines evaluating mechanical fields (strains, stresses, displacements) inside a composite media consisting of ellipsoidal-like inclusions embedded in an infinite matrix. The implemented, purely analytical, solution of both internal and external fields (inside and outside inclusion domains, respectively) is based on [1] and is fully accomplished in three dimensions. Moreover, the implemented algorithms extend the classic *Eshelby's* solution to take into account disturbances due to the presence of adjacent inclusions so as to deal with non-dilute media.

So far, the code offers the solution of micromechanical fields within the heterogeneous media containing inclusions of various shapes as listed below.

Inclusion shape	*Uniform eigenstrains*		*Non-uniform eigenstrains*	
	Internal fields	*External fields*	*Internal fields*	*External fields*
Ellipsoid	yes	yes	no	no
Sphere	yes	yes	no	no
Elliptic cylinder	yes	no	no	no
Cylinder	yes	no	no	no
Penny	yes	no	no	no
Closed penny (crack)	yes	no	no	no
Flat ellipsoid	yes	no	no	no
Oblate spheroid	yes	no	no	no
Prolate spheroid	yes	no	no	no

Table 1.1: Available mechanical fields with respect to applied eigenstrain and particular inclusion shape

Note, that as regard the inclusion shapes yet not fully implemented, these can be treated as ellipsoidal inclusions with one or more degenerated semiaxes. In this case, **μMECH** will work less efficiently in terms of computational time.

Chapter 2

Theory manual

What's actually behind the scope of $\boldsymbol{\mu}$**MECH** *library...*

2.1 Single inhomogeneity problem

The basic principle of the solution of mechanical fields in an isotropic infinite medium containing single isotropic inhomogeneity is sketched in Fig. 2.1a. *Eshelby* discovered in his fundamental work [1] that this problem can be decomposed into exactly two tasks of known solution and then assembled back by making use of the superposition principle Fig. 2.1b, c. So that the solution of *inhomogeneity problem* is given as the sum of *homogeneous infinite body problem* and *homogeneous inclusion problem* [1, 2]. In brief, the solution of the inhomogeneity problem consists from seeking the *equivalent transformation*

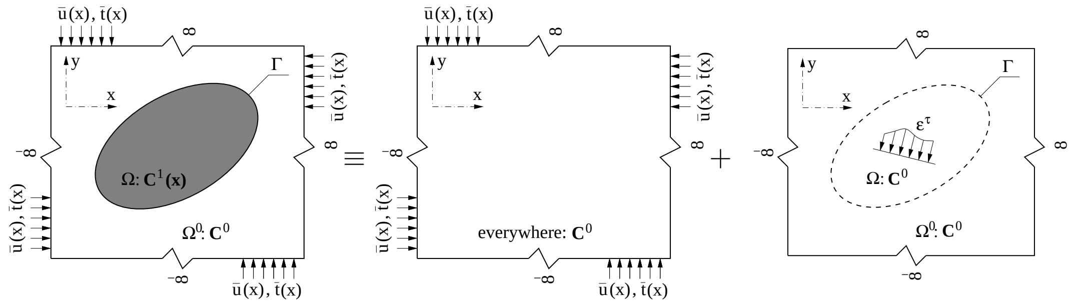


Figure 2.1: Principle of *Equivalent Inclusion Method*: a) inhomogeneity problem, b) problem of infinite homogeneous body, c) homogeneous inclusion problem

eigenstrain to be applied into homogeneous body within the inclusion domain Ω having the reference stiffness $\mathbf{C}^0$ Fig. 2.1c, so as to induce identical local mechanical response as original heterogeneous body of the total stiffness $\mathbf{C}(\mathbf{x})$. The total stiffness admits the following decomposition

$$\mathbf{C}(\mathbf{x}) = \mathbf{C}^0 + V(\mathbf{x})\mathbf{C}^1(\mathbf{x}), \tag{2.1}$$

where $\mathbf{C}^1(\mathbf{x})$ is the complementary stiffness tensor having the characteristic function

$$V(\mathbf{x}) = \begin{cases} 0 & \forall\, \mathbf{x} \in \Omega^0 \subset \mathbb{R}^3 \\ 1 & \forall\, \mathbf{x} \in \Omega \subset \mathbb{R}^3 \end{cases} . \tag{2.2}$$

Local mechanical fields are then searched according to Fig. 2.1 in the decomposed form given by

$$\boldsymbol{\varepsilon}(\mathbf{x}) = \boldsymbol{\varepsilon}^0 + \boldsymbol{\varepsilon}^1(\mathbf{x}),\ \boldsymbol{\sigma}(\mathbf{x}) = \boldsymbol{\sigma}^0 + \boldsymbol{\sigma}^1(\mathbf{x}),\ \text{and}\ \mathbf{u}(\mathbf{x}) = \mathbf{u}^0 + \mathbf{u}^1(\mathbf{x}), \tag{2.3}$$

where, the fields $\boldsymbol{\varepsilon}^0$, $\boldsymbol{\sigma}^0$, $\mathbf{u}^0$ stand for so called homogeneous part and $\boldsymbol{\varepsilon}^1(\mathbf{x})$, $\boldsymbol{\sigma}^1(\mathbf{x})$, $\mathbf{u}^1(\mathbf{x})$ for the perturbation (disturbation) part of the strain, stress and displacement field, respectively. Moreover, the *Eshelby's* solution of *homogeneous inclusion problem*

$$\begin{aligned} \mathbf{u}^1(\mathbf{x}) &= \boldsymbol{\mathcal{L}}(\mathbf{x}) : \boldsymbol{\varepsilon}^\tau, & (2.4)\\ \boldsymbol{\varepsilon}^1(\mathbf{x}) = \nabla_s \mathbf{u}^1(\mathbf{x}) &= \nabla\boldsymbol{\mathcal{L}}(\mathbf{x})\boldsymbol{\varepsilon}^\tau = \mathbf{S}(\mathbf{x}) : \boldsymbol{\varepsilon}^\tau & (2.5) \end{aligned}$$

yields

$$\boldsymbol{\sigma}^1(\mathbf{x}) = \mathbf{C}^0 :\ \left[\boldsymbol{\varepsilon}^1(\mathbf{x}) - \boldsymbol{\varepsilon}^\tau\right]. \tag{2.6}$$

Note, that $\boldsymbol{\mathcal{L}}(\mathbf{x}),\ \mathbf{S}(\mathbf{x})$ denote the *Eshelby's* tensors generally available even for the fields outside Ω. Now we enforce the equivalence of the local fields in heterogeneous and homogeneous body as

$$\boldsymbol{\sigma}\big(\mathbf{C}(\mathbf{x}), \boldsymbol{\varepsilon}(\mathbf{x})\big) \equiv \boldsymbol{\sigma}\big(\mathbf{C}^0, \boldsymbol{\varepsilon}^0, \mathbf{S}(\mathbf{x}), \boldsymbol{\varepsilon}^\tau\big), \tag{2.7}$$

and consequently seek for *equivalent* eigenstrain $\boldsymbol{\varepsilon}^\tau$ satisfying the equality between both sides of the equation. Eq. (2.7) can be expanded by using *Hook's* law as well as Eq. (2.3)^2 and Eq. (2.6) into the form

$$\mathbf{C}(\mathbf{x}) : \boldsymbol{\varepsilon}(\mathbf{x}) \equiv \mathbf{C}^0 : \boldsymbol{\varepsilon}^0 + \mathbf{C}^0 :\ \left[\boldsymbol{\varepsilon}^1(\mathbf{x}) - \boldsymbol{\varepsilon}^\tau\right], \tag{2.8}$$

which further yields

$$\mathbf{C}(\mathbf{x}) : \boldsymbol{\varepsilon}(\mathbf{x}) \equiv \mathbf{C}^0 :\ \left[\boldsymbol{\varepsilon}(\mathbf{x}) - \boldsymbol{\varepsilon}^\tau\right]. \tag{2.9}$$

Here the *stress free* transformation strain $\boldsymbol{\varepsilon}^\tau$ has identical characteristic function $V(\mathbf{x})$ as the complementary stiffness tensor $\mathbf{C}^1(\mathbf{x})$. Now, introducing Eq. (2.3)^1 and Eq. (2.1) into Eq. (2.9) we end up, after some algebra, with the relation

$$\left[\mathbf{C}(\mathbf{x}) - \mathbf{C}^0\right] : \boldsymbol{\varepsilon}^0 = \left[\mathbf{C}^0 : \mathbf{S}(\mathbf{x}) - \mathbf{C}(\mathbf{x}) : \mathbf{S}(\mathbf{x}) - \mathbf{C}^0\right] : \boldsymbol{\varepsilon}^\tau, \tag{2.10}$$

which can be finally recast in a compact form as

$$\boldsymbol{\varepsilon}^\tau = \mathbf{B}(\mathbf{x}) : \boldsymbol{\varepsilon}^0. \tag{2.11}$$

Note, that $\mathbf{B}(\mathbf{x})$ tensor in the last equation reads as

$$\mathbf{B}(\mathbf{x}) = -\left[\mathbf{C}^1(\mathbf{x}) : \mathbf{S}(\mathbf{x}) + \mathbf{C}^0\right]^{-1} : \mathbf{C}^1(\mathbf{x}). \tag{2.12}$$

2.2 Multiple inhomogeneity problem

As regard the multiple inclusion problem, the solution is based on a single inclusion problem which follows the strategy presented in previous section. In particular, a mechanical field within a body with N inclusions is obtained as the sum of N single inclusion tasks scaled by a multiplier (α_i) associated with each inclusion so as to fulfill self-equilibrium. Note, that the same strategy as in the previous section applies to derive the governing equations of multiple inclusion problem.

Let us consider a heterogeneous body consisting of clearly distinguishable inclusions in a matrix (Fig. 2.2a) subjected to a displacement and traction field $\bar{\mathbf{u}}(\mathbf{x}), \bar{\mathbf{t}}(\mathbf{x})$, respectively. Analogically to the

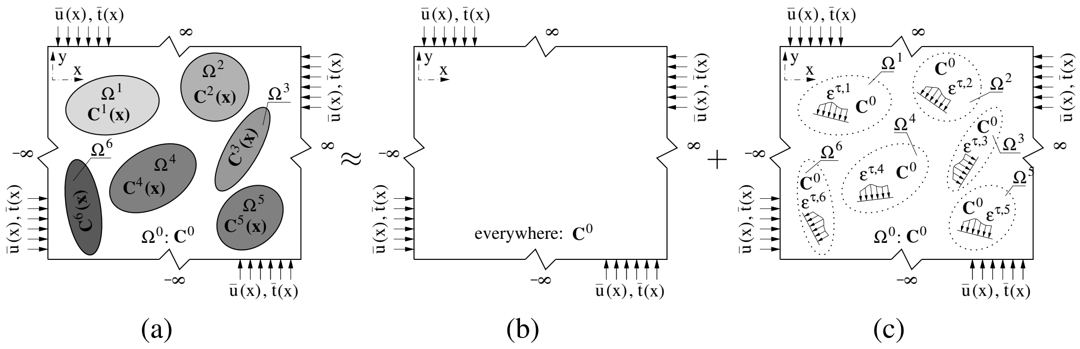


Figure 2.2: Principle of *Equivalent Inclusion Method*: a) multiple inhomogeneity problem, b) infinite homogeneous body, c) multiple homogeneous inclusion problem

previous section, the stiffness of such a material is decomposed as follows [3, 2, 5]

$$\mathbf{C}(\mathbf{x}) = \mathbf{C}^0 + V(\mathbf{x})\mathbf{C}^*(\mathbf{x}), \tag{2.13}$$

where $\mathbf{C}^0 \in \Omega^0$ is the stiffness tensor of the homogeneous infinite matrix and $\mathbf{C}^*(\mathbf{x}) = \sum_i^N [\mathbf{C}^i(\mathbf{x}) - \mathbf{C}^0]$ is its complement to $\mathbf{C}(\mathbf{x})$ caused by the presence of N inclusions. $\mathbf{C}^*(\mathbf{x})$ is nonzero only within the domain $\Omega = \Omega^1 \cup \cdots \cup \Omega^N$, so that the characteristic function $V(\mathbf{x})$ yields

$$V(\mathbf{x}) = \left\{ \begin{array}{ll} 0 & \forall\, \mathbf{x} \in \Omega^0 \subset \mathbb{R}^3 \\ 1 & \forall\, \mathbf{x} \in \Omega \subset \mathbb{R}^3 \end{array} \right. . \tag{2.14}$$

The decomposed displacement, strain and stress field, respectively, admit the form

$$\begin{aligned} \mathbf{u}(\mathbf{x}) &= \mathbf{u}^0(\mathbf{x}) + \mathbf{u}^*(\mathbf{x}), \\ \boldsymbol{\varepsilon}(\mathbf{x}) &= \boldsymbol{\varepsilon}^0(\mathbf{x}) + \boldsymbol{\varepsilon}^*(\mathbf{x}), \\ \boldsymbol{\sigma}(\mathbf{x}) &= \boldsymbol{\sigma}^0(\mathbf{x}) + \boldsymbol{\sigma}^*(\mathbf{x}). \end{aligned} \tag{2.15}$$

Here, the variables with $\cdot^0$ and $\cdot^*$ exponents stand for *homogeneous* and *perturbation* part of the fields previously defined.

As already suggested, the perturbation fields are calculated employing *equivalent inclusion method* extended for multiple inclusions by means of *Self-balancing* algorithm to satisfy their *self-equilibrium.* The equivalence of perturbation stresses inside the heterogeneous and homogeneous body (Fig. 2.2a, c) is accounted for by applying N *equivalent eigenstrain* fields $\varepsilon^{\tau,i}(\mathbf{x})$ into Ω^i. So that, written symbolically, it holds

$$\boldsymbol{\sigma}\big(\mathbf{C}(\mathbf{x}),\boldsymbol{\varepsilon}(\mathbf{x})\big) \approx \boldsymbol{\sigma}\big(\mathbf{C}^0,\boldsymbol{\varepsilon}^0(\mathbf{x}),\mathbf{S}^i(\mathbf{x}),\boldsymbol{\varepsilon}^{\tau,i}(\mathbf{x})\big), \tag{2.16}$$

which employing Eq. (2.15)[3] turns into

$$\boldsymbol{\sigma}\big(\mathbf{C}(\mathbf{x}),\boldsymbol{\varepsilon}(\mathbf{x})\big) \approx \boldsymbol{\sigma}^0\left(\mathbf{C}^0,\boldsymbol{\varepsilon}^0(\mathbf{x})\right) + \sum_i^N \alpha_i \boldsymbol{\sigma}^*\big(\mathbf{C}^0,\mathbf{S}^i(\mathbf{x}),\boldsymbol{\varepsilon}^{\tau,i}(\mathbf{x})\big), \tag{2.17}$$

where $\mathbf{S}^i(\mathbf{x})$ is the position dependent *Eshelby's* tensor of i^{th} inclusion and $\boldsymbol{\varepsilon}^0(\mathbf{x}) = \boldsymbol{\varepsilon}^0\big(\overline{\mathbf{u}}(\mathbf{x}),\overline{\mathbf{t}}(\mathbf{x})\big)$ stands for hypothetical *remote strain* field producing together with $\mathbf{C}^*(\mathbf{x})$ the required *transformation* eigenstrain $\boldsymbol{\varepsilon}^{\tau,i}(\mathbf{x})$, so as to ensure the equivalence between original, Fig. 2.2a, and equivalent, Fig. 2.2b, body. Next, the parameter α_i in Eq. (2.17) is the multiplier enforcing the self-equilibrium among all inclusions, here calculated by means of *self-balancing* algorithm. Note that for strongly *non-dilute* media (extensive mutual interactions among inclusions Ω^i) the accuracy of final solution is given by the choice of the order of equivalent eigenstrain polynomials. However, the study [4] shows that even the assumption of uniform eigenstrains exhibits unexpectedly good results as for both the quality of perturbation fields' solution as well as high computational efficiency.

2.2.1 Multiple inclusion problem via Self-balancing algorithm

The multiple inclusion perturbation fields $\mathbf{u}^*$, $\boldsymbol{\varepsilon}^*$ and $\boldsymbol{\sigma}^*$ as the counterpart of single inclusion perturbations introduced in Eq. (2.15) are determined for multiple inclusions from the separate *Eshelby's* solutions of each single inclusion Eq. (2.17). The required self-equilibrium is enforced by making use of an iterative procedure, here referred to as the *self-balancing* algorithm (Tab. 2.1). A _FULL_ *self-balancing*

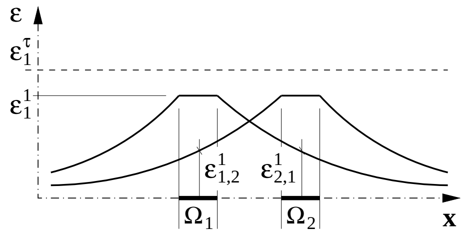


Figure 2.3: Principle of *self-balancing* algorithm for double inclusion problem in $1D$, ε_1^τ denotes the initial transformation strain, ε_1^1 stands for the perturbation strain after 1st step, $\varepsilon_{1,2}^1$ represents the strain perturbation in inclusion Ω_1 caused by the presence of inclusion Ω_2 and conversely $\varepsilon_{2,1}^1$ is the strain perturbation in inclusion Ω_2 caused by the neighboring inclusion Ω_1

algorithm ensures that the mechanical fields associated with inclusion i correctly reflect the influence of the remaining $N\backslash i$ inclusions. A modification of the *equivalent-transformation* strain inside an inclusion, so as to account for mechanical fields of adjacent inclusions, is performed iteratively. The initial transformation eigenstrain $\boldsymbol{\varepsilon}_i^\tau$ is applied to each inclusion within the **step 2** (ε_1^τ in Fig. 2.3). Consequently, the perturbation strain $\boldsymbol{\varepsilon}_i^1$ at all inclusion centroids is evaluated by means of the **step 3** (ε_1^1 in Fig. 2.3). Next, the transformation strain *correction* due to the adjacent inclusions is calculated in **step 7** by using the inverse of the *Eshelby* tensor $\mathbf{S}_i^{-1}$ and the perturbation strain $\boldsymbol{\varepsilon}_{j,i}^1$ at each inclusion centroid ($\varepsilon_{1,2}^1$ and $\varepsilon_{2,1}^1$ in Fig. 2.3). Finally, within the framework of **step 9**, the transformation strain is updated by adding the *correction* transformation strain, and the new perturbation strains are then re-calculated exclusively from this correction by means of **step 10**. The algorithm continues until a small *Euclidean* norm between last two total transformation strain fields is achieved. The computational complexity of this algorithm is $O(N^2)$, however, this can be improved by taking into account only those inclusions which have a non-negligible influence to a certain inclusion i. This version of the *self-balancing* algorithm is refereed to as the `_OPTIMIZED_` one.

```
   SelfBalancingAlgorithm($\boldsymbol{\varepsilon}_i^\tau, \mathbf{S}_i, \mathbf{S}_i^{-1}, N$)
1  For ($i \leq N$)
2      $\boldsymbol{\varepsilon}_{\text{total},i}^\tau = \boldsymbol{\varepsilon}_i^\tau$
3      $\boldsymbol{\varepsilon}_i^1 = \mathbf{S}_i : \boldsymbol{\varepsilon}_i^\tau$
4  EndFor
5  Do
6      For ($i \leq N$)
7          $\boldsymbol{\varepsilon}_i^\tau = \sum_{j \setminus i}^N \mathbf{S}_i^{-1} : \boldsymbol{\varepsilon}_{j,i}^1$
8          $\boldsymbol{\varepsilon}_{\text{total},i}^\tau = \boldsymbol{\varepsilon}_{\text{total},i}^{\tau,\text{new}}$
9          $\boldsymbol{\varepsilon}_{\text{total},i}^{\tau,\text{new}} = \boldsymbol{\varepsilon}_{\text{total},i}^{\tau,\text{new}} + \boldsymbol{\varepsilon}_i^\tau$
10         $\boldsymbol{\varepsilon}_i^1 = \mathbf{S}_i : \boldsymbol{\varepsilon}_i^\tau$
11     EndFor
12 While ($\sum_i^N \|\boldsymbol{\varepsilon}_{\text{total},i}^\tau - \boldsymbol{\varepsilon}_{\text{total},i}^{\tau,\text{new}}\| > \epsilon$)
```

Table 2.1: *Self-balancing* algorithm

Chapter 3

Tutorial

How to . . .

3.1 Installation

Download and unpack `muMECH.zip` archive to a preffered directory and simply start using it. Do not forget include `analyticalFunctions.h` and `eshelbySoluTypes.h` headers to allow calling all the addressed functions. As the code is entirely free, released under GNU regulations, you can modify/distribute it freely as well. You can even use all the functions/methods declared outside `analyticalFunctions.h` which has been not directly addressed in this tutorial, but be aware to include additional header file(s) containig appropriate declarations.

3.2 Input file format

In brief, the ***µ*MECH** input file syntax is build on freely available Visualization Toolkit - VTK [1], in particular on its `UNSTRUCTURED_GRID` version. The implemented functions described **Section 3.3** allow for evaluating mechanical fields in one or multiple points with respect to applied load cases. In particular, either one or all the six load cases must be applied. In the first case, the load case is re-called (for some particular reasons) by a keyword `TENSORS Remote_strains_11`. In the later one, six keywords `TENSORS Remote_strains_ij` have to be included in the input file. The load cases representing the actual remote strains in inclusion centroids must be specified for each single inclusion.

The input file also contains the in formations about the geometry of a calculated task. The particular meaning of each compulsory keyword mostly reflects its VTK counterpart and is as follows.

§

`POINTS`
Coordinates of inclusion centroids.

§

[1]`http://www.vtk.org/VTK/img/file-formats.pdf`

`CELLS`

Definition of cell connectivity (topology). In our case just simple points.

§

`CELL_TYPES`

Cell type definitions. In our case, only simple points (i.e. integer 1) makes sense.

§

`Inclusion_shape`

Defines the shape of each particular inclusion. The shapes are defined in `eshelbySoluTypes.h`.

symbolic constant	*input file value*
`_ELLIPSOID_`	1
`_SPHERE_`	2
`_ELLIPTIC_CYLINDER_`	3
`_CYLINDER_`	4
`_PENNY_`	5
`_CLOSED_PENNY_`	6
`_FLAT_ELLIPSOID_`	7
`_OBLATE_SPHEROID_`	8
`_PROLATE_SPHEROID_`	9

Table 3.1: Inclusion shape values as defined in `eshelbySoluTypes.h`

§

`Youngs_modulus`

Young's modulus of each individual inclusion.

§

`Poissons_ratio`

Poisson's ratio of each individual inclusion.

§

`Semiaxes_dimensions`

Semiaxes' dimensions in following order a_1, a_2, a_3. It is not required that $a_1 > a_2 > a_3$, but if this is the case, the code becomes more efficient.

§

`Euller_angles` (*Eul(l)er* is not a spelling mistake, it is really implemented with double 'l')

The rotation of each inclusion given by means of the Euler angles of its principal semiaxes. Note, that the Euler angles φ, ν and ψ correspond to successive rotation of ellipsoidal semiaxes a_1, a_2 and a_3 about global coordinate axes x_3, x_1 and x_3, respectively.

§

`Remote_strains_11`

The 1st load step. If the `lcMode` = `_SINGLE_` (load case mode), this is the only load case which must be necessarily included in the input file. On the other hand, one should not meet any troubles when other load cases included as well. In the case, the mechanical response to all the six load cases is required, instead of `_SINGLE_` set `lcMode` = `_MULTIPLE_`.

§

`Remote_strains_22`

The 2nd load step. Active, only if `lcMode` = `_MULTIPLE_`.

§

`Remote_strains_33`

The 3rd load step. Active, only if `lcMode` = `_MULTIPLE_`.

§

`Remote_strains_12`

The 4th load step. Active, only if `lcMode` = `_MULTIPLE_`.

§

`Remote_strains_23`

The 5th load step. Active, only if `lcMode` = `_MULTIPLE_`.

§

`Remote_strains_13`

The 6th load step. Active, only if `lcMode` = `_MULTIPLE_`.

§

Example

The file listed below contains three ellipsoidal inclusions of different Euler rotations loaded by exactly six (maximum number) load cases Fig. 3.1.

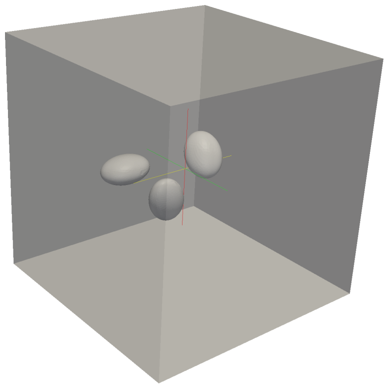


Figure 3.1: Geometry and topology of three inclusion benchmark

```
# vtk DataFile Version 3.0
created by Jan Novak, 08.12.2009
ASCII
DATASET UNSTRUCTURED_GRID
POINTS 3 double
-0.04806993  0.07826698  0.01481089
 0.01645318 -0.17864680 -0.15450740
 0.12793000 -0.06594404 -0.02731760
CELLS 3 6
1 1
1 2
1 3
CELL_TYPES 3
1
1
1
POINT_DATA 3
SCALARS Inclusion_shape int 1
LOOKUP_TABLE default
1
1
1
SCALARS Youngs_modulus double 1
LOOKUP_TABLE default
2.0
2.0
2.0
SCALARS Poissons_ratio double 1
LOOKUP_TABLE default
```

```
0.1
0.1
0.1
VECTORS Semiaxes_dimensions double
0.05 0.075 0.10
0.05 0.10  0.075
0.10 0.075 0.05
VECTORS Euller_angles double
74.2103 48.4392 -48.0699
37.2731 22.2687 -25.5056
46.7402 11.1690 -26.3025
TENSORS Remote_strains_11 double
1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0.
1. 0. 0. 0. 0. 0. 0. 0. 0.
TENSORS Remote_strains_22 double
0. 0. 0. 0. 1. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0.
0. 0. 0. 0. 1. 0. 0. 0. 0.
TENSORS Remote_strains_33 double
0. 0. 0. 0. 0. 0. 0. 0. 1.
0. 0. 0. 0. 0. 0. 0. 0. 1.
0. 0. 0. 0. 0. 0. 0. 0. 1.
TENSORS Remote_strains_12 double
0. 1. 0. 1. 0. 0. 0. 0. 0.
0. 1. 0. 1. 0. 0. 0. 0. 0.
0. 1. 0. 1. 0. 0. 0. 0. 0.
TENSORS Remote_strains_23 double
0. 0. 0. 0. 0. 1. 0. 1. 0.
0. 0. 0. 0. 0. 1. 0. 1. 0.
0. 0. 0. 0. 0. 1. 0. 1. 0.
TENSORS Remote_strains_13 double
0. 0. 1. 0. 0. 0. 1. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 0.
0. 0. 1. 0. 0. 0. 1. 0. 0.
```

3.3 Available functions

3.3.1 Interface - C++

§

Constructor

```
analyticalFunctions( char * vtkTopologyFile,
                     MatrixRecord infMedRec,
                     SBAtype SelfBalAlgorithm,
                     LCtype lcMode )
```

`vtkTopologyFile` – pointer to a VTK file containing inclusion geometry and topology
`infMedRec` – structure containing infinite medium record, i.e. one must initialize `E`, `nu`, `origin` items (see `eshelbySoluTypes.h` for more details)
`SelfBalAlgorithm` – self balance algorithm flag (`_FULL_` or `_OPTIMIZED_`)
`lcMode` – load case type flag (`_SINGLE_` or `_MULTIPLE_`)

§

Destructor

```
~analyticalFunctions()
```

§

Creating the VTK file of inclusion record

```
void createInclRecFile( char * vtkTopologyFile,
                        MatrixRecord infMedRec,
                        SBAtype SelfBalAlgorithm,
                        LCtype lcMode )
```

`vtkTopologyFile` – pointer to a VTK file containing inclusion geometry and topology
`infMedRec` – structure containing infinite medium record, i.e. one must initialize `E`, `nu`, `origin` items (see `eshelbySoluTypes.h` for more details)
`SelfBalAlgorithm` – self balance algorithm flag (`_FULL_` or `_OPTIMIZED_`)
`lcMode` – load case type flag (`_SINGLE_` or `_MULTIPLE_`)

§

Solution of the perturbation fields of a point in required notation

```
void giveEshelbyPertFieldsOfOnePoint( double *coords,
                                      double * disp,
                                      double * strain,
                                      double * stress,
                                      LoadCase LS,
                                      NotationType notationFlag )
```

`coords` – coordinates of a point
`disp` – displacement vector to be calculated

strain – strain tensor to be calculated
stress – stress tensor to be calculated
LS – given load case (_LS11_, _LS22_ , _LS33_, _LS12_, _LS23_, _LS13_, _LSALL_)
notationFlag – notation of strain/stress field (_VOIGT_ = _ENGINEERING_, _MANDEL_, _ACTUAL_ = _THEORETICAL_), default notation is _MANDEL_

§

Solution of the perturbation displacements and stresses of a point in required notation

```
void giveEshelbyPertFieldsOfOnePoint( double * coords,
                                      double * disp,
                                      double * stress,
                                      LoadCase LS,
                                      NotationType notationFlag)
```

coords – coordinates of a point
disp – displacement vector to be calculated
stress – stress tensor to be calculated
LS – given load case (_LS11_, _LS22_ , _LS33_, _LS12_, _LS23_, _LS13_, _LSALL_)
notationFlag – notation of strain/stress field (_VOIGT_ = _ENGINEERING_, _MANDEL_, _ACTUAL_ = _THEORETICAL_), default notation is _MANDEL_

§

Solution of the perturbation displacements and stresses of a point in required notation depending on chosen action region of each inclusion

```
void giveEshelbyPertFieldsOfOnePoint( double * coords,
                                      double * disp,
                                      double * stress,
                                      LoadCase LS,
                                      NotationType notationFlag,
                                      PFCmode pfcMode )
```

coords – coordinates of a point
disp – displacement vector to be calculated
stress – stress tensor to be calculated
LS – given load case (_LS11_, _LS22_ , _LS33_, _LS12_, _LS23_, _LS13_, _LSALL_)
notationFlag – notation of strain/stress field (_VOIGT_ = _ENGINEERING_, _MANDEL_, _ACTUAL_ = _THEORETICAL_), default notation is _MANDEL_
pfcMode – flag of point fields calculation type (__FULL_, __OPTIMIZED_)

§

Solution of the perturbation strain field of a point in required notation. Note: For efficiency purposes, rather use `giveEshelbyPertFieldsOfOnePoint` *especially in the case when other fields are also required.*

```
void giveEshelbyPertStrainOfOnePoint( double * coords,
                                      double * strain,
                                      LoadCase LS,
                                      NotationType notationFlag )
```

`coords` – coordinates of a point
`strain` – strain tensor to be calculated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`

§

Solution of the perturbation displacement field of a point in required notation. Note: For efficiency purposes, rather use `giveEshelbyPertFieldsOfOnePoint` *especially in the case when other fields are also required.*

```
void giveEshelbyPertDisplOfOnePoint( double * coords,
                                     double * disp,
                                     LoadCase LS )
```

`coords` – coordinates of a point
`disp` – displacement vector to be calculated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)

§

Solution of the perturbation fields of multiple points in required notation

```
void giveEshelbyPertFieldsOfMultPoint( double * coords,
                                       double * disp,
                                       double * strain,
                                       double * stress,
                                       int noPoints,
                                       LoadCase LS,
                                       NotationType notationFlag )
```

`coords` – coordinates of points in C row-by-row alignment
`disp` – displacement vectors to be calculated in C row-by-row alignment

`strain` – strain tensors to be calculated in C row-by-row alignment
`stress` – stress tensors to be calculated in C row-by-row alignment
`noPoints` – number of points in which the fields will be evaluated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`

§

Solution of the perturbation displacements and stresses of multiple points in required notation

```
void giveEshelbyPertFieldsOfMultPoint( double *coords,
                                       double *disp,
                                       double *stress,
                                       int noPoints,
                                       LoadCase LS,
                                       NotationType notationFlag )
```

`coords` – coordinates of points in C row-by-row alignment
`disp` – displacement vectors to be calculated in C row-by-row alignment
`stress` – stress tensors to be calculated in C row-by-row alignment
`noPoints` – number of points in which the fields will be evaluated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`

§

Solution of the perturbation displacements and stresses of multiple points in required notation. The resulting fields depend on the action region of surrounding inclusions of given points.

```
void giveEshelbyPertFieldsOfMultPoint( double *coords,
                                       double *disp,
                                       double *stress,
                                       int noPoints,
                                       LoadCase LS,
                                       NotationType notationFlag,
                                       PFCmode pfcMode )
```

`coords` – coordinates of points in C row-by-row alignment
`disp` – displacement vectors to be calculated in C row-by-row alignment
`strain` – strain tensors to be calculated in C row-by-row alignment
`stress` – stress tensors to be calculated in C row-by-row alignment
`noPoints` – number of points in which the fields will be evaluated

`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`
`pfcMode` – flag of point fields calculation type (`__FULL_`, `__OPTIMIZED_`)

§

Solution of the perturbation strain of multiple points in required notation. Note: For efficiency purposes, rather use `giveEshelbyPertFieldsOfMultiplePoint` *especially in the case when other fields are also required.*

```
void giveEshelbyPertStrainOfMultPoint( double * coords,
                                       double * strain,
                                       int noPoints,
                                       LoadCase LS,
                                       NotationType notationFlag )
```

`coords` – coordinates of points in C row-by-row alignment
`strain` – strain tensors to be calculated in C row-by-row alignment
`noPoints` – number of points in which the field will be evaluated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`

§

Function gives the solution of the perturbation displacement of multiple points. Note: For efficiency purposes, rather use `giveEshelbyPertFieldsOfMultiplePoint` *especially in the case when other fields are also required.*

```
void giveEshelbyPertDisplOfMultPoint( double * coords,
                                      double * disp,
                                      int noPoints,
                                      LoadCase LS )
```

`coords` – coordinates of points in C row-by-row alignment
`disp` – displacement vectors to be calculated in C row-by-row alignment
`noPoints` – number of points in which the field will be evaluated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)

§

Solution of the total displacement field of a point

```
void analDirihletNonHomogenousEshelby( double ⋆ coords,
                                       double ⋆ disp,
                                       LoadCase LS )
```

`coords` – coordinates of point
`disp` – displacement vector to be calculated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)

3.3.2 Interface - C

§

Creating the VTK file of inclusion record

```
void esh_createInclRecFile( char ⋆ vtkTopologyFile,
                            MatrixRecord infMedRec,
                            SBAtype SelfBalAlgorithm,
                            LCtype lcMode )
```

`vtkTopologyFile` – pointer to a VTK file containing inclusion geometry and topology
`infMedRec` – structure containing infinite medium record, i.e. one must initialize `E`, `nu`, `origin` items (see `eshelbySoluTypes.h` for more details)
`SelfBalAlgorithm` – self balance algorithm flag (`_FULL_` or `_OPTIMIZED_`)
`lcMode` – load case type flag (`_SINGLE_` or `_MULTIPLE_`)

§

Initializing of inclusion record (constructor-like function)

```
void esh_reloadInclRecord( char ⋆ vtkTopologyFile,
                           MatrixRecord infMedRec,
                           SBAtype SelfBalAlgorithm,
                           LCtype lcMode)
```

`vtkTopologyFile` – pointer to a VTK file containing inclusion geometry and topology
`infMedRec` – structure containing infinite medium record, i.e. one one must initialize `E`, `nu`, `origin` items (see `eshelbySoluTypes.h` for more details)
`SelfBalAlgorithm` – self balance algorithm flag (`_FULL_` or `_OPTIMIZED_`)
`lcMode` – load case type flag (`_SINGLE_` or `_MULTIPLE_`)

§

Deleting of inclusion record (destructor-like function)

```
void esh_deleteInclRecord( void )
```

§

Solution of the perturbation fields of a point in required notation

```
void esh_givePertFieldsInOnePoint( double *coords,
                                   double * disp,
                                   double * strain,
                                   double * stress,
                                   LoadCase LS,
                                   NotationType notationFlag )
```

`coords` – coordinates of a point
`disp` – displacement vector to be calculated
`strain` – strain tensor to be calculated
`stress` – stress tensor to be calculated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`

§

Solution of the perturbation displacements and stresses of a point in required notation

```
void esh_givePertFieldsInOnePoint( double *coords,
                                   double * disp,
                                   double * stress,
                                   LoadCase LS,
                                   NotationType notationFlag )
```

`coords` – coordinates of a point
`disp` – displacement vector to be calculated
`stress` – stress tensor to be calculated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`

§

Solution of the perturbation displacements and stresses of a point in required notation. The resulting fields depend upon the action region of surrounding inclusions of a given point.

```
void esh_givePertFieldsInOnePoint( double *coords,
                                   double * disp,
                                   double * stress,
                                   LoadCase LS,
                                   NotationType notationFlag,
                                   PFCmode pfcMode )
```

`coords` – coordinates of a point
`disp` – displacement vector to be calculated
`stress` – stress tensor to be calculated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`
`pfcMode` – flag of point fields calculation type (`__FULL_`, `__OPTIMIZED_`)

§

Solution of the perturbation strain field of a point in required notation. Note: For efficiency purposes, rather use `esh_givePertFieldsInOnePoint` *especially in the case when other fields are also required.*

```
void esh_givePertStrainInOnePoint( double * coords,
                                   double * strain,
                                   LoadCase LS,
                                   NotationType notationFlag )
```

`coords` – coordinates of a point
`strain` – strain tensor to be calculated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`

§

Solution of the perturbation displacement field of a point in required notation. Note: For efficiency purposes, rather use `esh_givePertFieldsInOnePoint` *especially in the case when other fields are also required.*

```
void esh_givePertDisplInOnePoint( double * coords,
                                  double * disp,
                                  LoadCase LS )
```

coords – coordinates of a point
disp – displacement vector to be calculated
LS – given load case (_LS11_, _LS22_ , _LS33_, _LS12_, _LS23_, _LS13_, _LSALL_)

§

Solution of the perturbation fields of multiple points in required notation

```
void esh_givePertFieldsInMultPoint( double ⋆ coords,
                                    double ⋆ disp,
                                    double ⋆ strain,
                                    double ⋆ stress,
                                    int noPoints,
                                    LoadCase LS,
                                    NotationType notationFlag )
```

coords – coordinates of points in C row-by-row alignment
disp – displacement vectors to be calculated in C row-by-row alignment
strain – strain tensors to be calculated in C row-by-row alignment
stress – stress tensors to be calculated in C row-by-row alignment
noPoints – number of points in which the fields will be evaluated
LS – given load case (_LS11_, _LS22_ , _LS33_, _LS12_, _LS23_, _LS13_, _LSALL_)
notationFlag – notation of strain/stress field (_VOIGT_ = _ENGINEERING_, _MANDEL_, _ACTUAL_ = _THEORETICAL_), default notation is _MANDEL_

§

Solution of the perturbation displacements and stresses of multiple points in required notation

```
void esh_givePertFieldsInMultPoint( double ⋆ coords,
                                    double ⋆ disp,
                                    double ⋆ stress,
                                    int noPoints,
                                    LoadCase LS,
                                    NotationType notationFlag )
```

coords – coordinates of points in C row-by-row alignment
disp – displacement vectors to be calculated in C row-by-row alignment
stress – stress tensors to be calculated in C row-by-row alignment
noPoints – number of points in which the fields will be evaluated
LS – given load case (_LS11_, _LS22_ , _LS33_, _LS12_, _LS23_, _LS13_, _LSALL_)
notationFlag – notation of strain/stress field (_VOIGT_ = _ENGINEERING_, _MANDEL_, _ACTUAL_ = _THEORETICAL_), default notation is _MANDEL_

§

Solution of the perturbation displacements and stresses of multiple points in required notation. The resulting fields depend on the action region of surrounding inclusions of given points.

```
void esh_givePertFieldsInMultPoint( double * coords,
                                    double * disp,
                                    double * stress,
                                    int noPoints,
                                    LoadCase LS,
                                    NotationType notationFlag,
                                    PFCmode pfcMode )
```

`coords` – coordinates of points in C row-by-row alignment
`disp` – displacement vectors to be calculated in C row-by-row alignment
`strain` – strain tensors to be calculated in C row-by-row alignment
`stress` – stress tensors to be calculated in C row-by-row alignment
`noPoints` – number of points in which the fields will be evaluated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`
`pfcMode` – flag of point fields calculation type (`__FULL_`, `__OPTIMIZED_`)

§

Solution of the perturbation strain of multiple points in required notation. Note: For efficiency purposes, rather use `esh_givePertFieldsInMultPoint` *especially in the case when other fields are also required.*

```
void esh_givePertStrainInMultPoint( double * coords,
                                    double * strain,
                                    int noPoints,
                                    LoadCase LS,
                                    NotationType notationFlag )
```

`coords` – coordinates of points in C row-by-row alignment
`strain` – strain tensors to be calculated in C row-by-row alignment
`noPoints` – number of points in which the field will be evaluated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)
`notationFlag` – notation of strain/stress field (`_VOIGT_` = `_ENGINEERING_`, `_MANDEL_`, `_ACTUAL_` = `_THEORETICAL_`), default notation is `_MANDEL_`

§

Function gives the solution of the perturbation displacement of multiple points. Note: For efficiency purposes, rather use `esh_givePertFieldsInMultPoint` *especially in the case when other fields are also required.*

```
void esh_givePertDisplInMultPoint( double * coords,
                                   double * disp,
                                   int noPoints,
                                   LoadCase LS )
```

`coords` – coordinates of points in C row-by-row alignment
`disp` – displacement vectors to be calculated in C row-by-row alignment
`noPoints` – number of points in which the field will be evaluated
`LS` – given load case (`_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)

3.4 Implementation examples

When using either C++ or C interface it is necessary to initialize infinite matrix record as either
`static MatrixRecord infMedRec = { 0.1, 1.0, { 0., 0., 0. } };`
where the values within curly breckets represent, respectively, the *Poisson's* ratio, *Young's* modulus and coordinates of the global coordinate system origin, or in verbose form
`double infMedRec.nu = 0.1;`
`double infMedRec.E = 1.0;`
`double infMedRec.origin = { 0., 0., 0. };`
Furthemore, the type of *self-balancing* algorithm must be set by e.g.
`SBAtype sbAlg = _OPTIMIZED_;` (either `_OPTIMIZED_` or `_FULL_`),
the load case mode as e.g.
`LCtype lcMode = _SINGLE_;` (optiopns: `_SINGLE_` / `_MULTIPLE_`)
`LoadCase LS = _LS11_;` (optiopns: `_LS11_`, `_LS22_` , `_LS33_`, `_LS12_`, `_LS23_`, `_LS13_`, `_LSALL_`)

The two examples for both C and C++ interface follow.

3.4.1 Implementation via C++ interface

```
//*******************************************************************
//   ## #  ###  ##  ##  ### ######  ### (c) copyright is for loosers!
//    ##  #  #  #   #  # #  #  #  # #
//    #   #  # ### ### ##   #  #  #   #
//   #     ###  #   #   ### #  #  # ###  MICROMECHANICS
//*******************************************************************
#include <stdio.h>
#include "analyticalFunctions.h"
#include "eshelbySoluTypes.h"
//*******************************************************************
```

```
// description: main function - C++_test
// last edit:   26. 11. 2010
//*****************************************************************
int main( )
{
  //declarations and initializations
  //inclusions' geometry and topology file
  char vtkFile[] = "3_icl_geom.vtk";
  //infinite medium properties
  static MatrixRecord infMedRec = { 0.1, 1.0, { 0., 0., 0. } };
  SBAtype sbAlg = _OPTIMIZED_; //type of self-balancing algorithm
  LCtype lcMode = _MULTIPLE_; //load case mode
  LoadCase LS = _LSALL_; //load case to be evaluated
  //type of algorithm of point fields to be evaluated
  PFCmode pfcMode = __OPTIMIZED_;
  //arbitrary coordinates of two points
  double coords[6] = { 0., 0., 0., .1, .3, .4 };
  //calculated fields
  double d[2*3*6]; //displs: 2 points*3 components*6 load cases
  double e[2*6*6]; //strains: 2 points*6 components*6 load cases
  double s[2*6*6]; //stresse: 2 points*6 components*6 load cases
  int i, j = 0; //increments
  //pointer to analytical functions object
  analyticalFunctions * analFunc = NULL;

  //PRE-PROCESOR
  //self-balancing algorithm and VTK file re-creating
  analFce = new analyticalFunctions(lcMode);
  analFce->createInclRecFile(vtkFile,infMedRec,sbAlg,lcMode);
  delete analFce;
  //building inclusion record
  analFunc = new analyticalFunctions(vtkFile,infMedRec,sbAlg,lcMode);
  //PROCESOR
  analFunc->giveEshelbyPertFieldsOfMultPoint(coords,d,e,s,2,LS,
                                             _ACTUAL_);
  //POST-PROCESOR
  printf("\nperturbation dlacements:\n");
  for(i = 0; i < 6; i++){
    j = i*6;
    printf("Load case %d:\tux\t\tuy\t\tuz\n",i+1);
    printf("point 1: \t%e\t%e\t%e\n",d[0+j],d[1+j],d[2+j]);
    printf("point 2: \t%e\t%e\t%e\n",d[3+j],d[4+j],d[5+j]);
  }
  printf("\nperturbation stress:\n");
  for(i = 0; i < 6; i++){
    j = i*6;
    printf("Load case %d:\texx\t\teyy\t\tezz\t\texy\t\teyz\t\texz\n",
           i+1);
    printf("point 1:\t%e\t%e\t%e\t%e\t%e\t%e\n",
           e[0+j],e[1+j],e[2+j],e[3+j],e[4+j],e[5+j]);
    printf("point 2:\t%e\t%e\t%e\t%e\t%e\t%e\n",
           e[6+j],e[7+j],e[8+j],e[9+j],e[10+j],e[11+j]);
  }
  printf("\nperturbation strain:\n");
  for(i = 0; i < 6; i++){
    j = i*6;
    printf("Load case %d:\tsxx\t\tsyy\t\tszz\t\tsxy\t\tsyz\t\tsxz\n",
           i+1);
    printf("point 1:\t%e\t%e\t%e\t%e\t%e\t%e\n",
           s[0+j],s[1+j],s[2+j],s[3+j],s[4+j],s[5+j]);
    printf("point 2:\t%e\t%e\t%e\t%e\t%e\t%e\n",
           s[6+j],s[7+j],s[8+j],s[9+j],s[10+j],s[11+j]);
  }
  //deleting inclusion record
  delete analFunc;
  printf("C++_test: done\n");
  return 0;
}//end of function: C++_test
/*end of file*/
//*******************************************************************
```

3.4.2 Implementation via C interface

```
//*******************************************************************
//   ## #  ###  ##  ##  ### ######  ### (c) copyright is for loosers!
//    ##  #  #  #   #  # #  #  #  # #
//    #   #  # ### ### ##   #  #  #   #
//   #     ###  #   #   ### #  #  # ###  MICROMECHANICS
//*******************************************************************
#include <stdio.h>
#include "analyticalFunctions.h"
#include "eshelbySoluTypes.h"
//*******************************************************************
// description: main function - C_test
// last edit:   26. 11. 2010
//*******************************************************************
int main( )
{
```

```
//declarations and initializations
//inclusions' geometry and topology file
char vtkTopologyFile[] = "3_icl_geom.vtk";
//infinite medium properties
static MatrixRecord infMedRec = { 0.1, 1.0, { 0., 0., 0. } };
SBAtype sbAlg = _OPTIMIZED_; //type of self-balancing algorithm
LCtype lcMode = _MULTIPLE_; //load case mode
LoadCase LS = _LSALL_; //load case to be evaluated
//type of algorithm of point fields to be evaluated
PFCmode pfcMode = __OPTIMIZED_;
//arbitrary coordinates of two points
double coords[6] = { 0., 0., 0., .1, .3, .4 };
//calculated fields
double d[2*3*6]; //displs: 2 points*3 components*6 load cases
double e[2*6*6]; //strains: 2 points*6 components*6 load cases
double s[2*6*6]; //stresse: 2 points*6 components*6 load cases
int i, j = 0; //increments

//PRE-PROCESOR
//self-balancing algorithm and VTK file re-creating
esh_createInclRecFile(vtkTopologyFile,infMedRec,sbAlg,lcMode);
//creating inclusion record
esh_reloadInclRecord(vtkTopologyFile,infMedRec,sbAlg,lcMode);
//PROCESOR
esh_givePertFieldsInMultPoint(coords,d,e,s,2,LS,_ACTUAL_);
//POST-PROCESOR
printf("\nperturbation dlacements:\n");
for(i = 0; i < 6; i++){
  j = i*6;
  printf("Load case %d:\tux\t\tuy\t\tuz\n",i+1);
  printf("point 1: \t%e\t%e\t%e\n",d[0+j],d[1+j],d[2+j]);
  printf("point 2: \t%e\t%e\t%e\n",d[3+j],d[4+j],d[5+j]);
}
printf("\nperturbation stress:\n");
for(i = 0; i < 6; i++){
  j = i*6;
  printf("Load case %d:\texx\t\teyy\t\tezz\t\texy\t\teyz\t\texz\n",
          i+1);
  printf("point 1:\t%e\t%e\t%e\t%e\t%e\t%e\n",
          e[0+j],e[1+j],e[2+j],e[3+j],e[4+j],e[5+j]);
  printf("point 2:\t%e\t%e\t%e\t%e\t%e\t%e\n",
          e[6+j],e[7+j],e[8+j],e[9+j],e[10+j],e[11+j]);
}
printf("\nperturbation strain:\n");
```

```
  for(i = 0; i < 6; i++){
    j = i*6;
    printf("Load case %d:\tsxx\t\tsyy\t\tszz\t\tsxy\t\tsyz\t\tsxz\n",
            i+1);
    printf("point 1:\t%e\t%e\t%e\t%e\t%e\t%e\n",
            s[0+j],s[1+j],s[2+j],s[3+j],s[4+j],s[5+j]);
    printf("point 2:\t%e\t%e\t%e\t%e\t%e\t%e\n",
            s[6+j],s[7+j],s[8+j],s[9+j],s[10+j],s[11+j]);
  }
  //deleting inclusion record
  esh_deleteInclRecord( );
  printf("C_test: done\n");
  return 0;
}//end of function: C_test
/*end of file*/
//*******************************************************************
```

3.5 Important comments

The code ***µ*****MECH**, as such, also contain a huge number of functions suitable for *pre-* and *post-processing* of calculated data, especially implemented for visualization purposes by means of VTK file(s). Further functions evaluating the standard (position independent) *Eshelby* tensors are also available. Hopefully, all of those will be documented in a near future as long as one requests so.

Any comments and questions email to: `novakj(at)cml.fsv.cvut.cz`.

List of Figures

2.1 Principle of *Equivalent Inclusion Method*: a) inhomogeneity problem, b) problem of infinite homogeneous body, c) homogeneous inclusion problem 5

2.2 Principle of *Equivalent Inclusion Method*: a) multiple inhomogeneity problem, b) infinite homogeneous body, c) multiple homogeneous inclusion problem 7

2.3 Principle of *self-balancing* algorithm for double inclusion problem in $1D$, ε_1^τ denotes the initial transformation strain, ε_1^1 stands for the perturbation strain after 1^{st} step, $\varepsilon_{1,2}^1$ represents the strain perturbation in inclusion Ω_1 caused by the presence of inclusion Ω_2 and conversely $\varepsilon_{2,1}^1$ is the strain perturbation in inclusion Ω_2 caused by the neighboring inclusion Ω_1 8

3.1 Geometry and topology of three inclusion benchmark 13

List of Tables

1.1 Available mechanical fields with respect to applied eigenstrain and particular inclusion shape . . . 4

2.1 *Self-balancing* algorithm . . . 9

3.1 Inclusion shape values as defined in `eshelbySoluTypes.h` . . . 11

Acknowledgements

Funding by following associations/organizations under specified pojects is gratefully acknowledged (chronological order applies).

CIDEAS - **C**enter of **I**ntegrated **DE**sign of **A**dvanced **S**tructures

- 1M0579

GRPE - **G**lasgow **R**esearch **P**artnership in **E**ngineering

- “Multi-scale modelling of fibre reinforced composites”

GAČR - **C**zech **F**unding **A**ssociation

- 13-22230S “A hybrid multiscale predictive modelling tool for heterogeneous solids”
- 103/09/P490 “Simulation of heterogeneous Materials Based on Integral Equations”
- 103/09/1748 “Integration of Nanoindentation with Numerical Tools for Upscaling of Nanomechanical Properties of Heterogeneous Materials”

Furthermore author(s) would like to thank Jan Zeman from CTU in Prague for careful a review of this manuscript.

Bibliography

[1] J. D. Eshelby, *The determination of the elastic field of an ellipsoidal inclusion, and related problems*, Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences **241** (1957), no. 1226, 376–396.

[2] T. Mura, *Micromechanics of Defects in Solids.*, Martinus Nijhoff Publishers, P. O. Box 163, 3300 AD Dordrecht, The Netherlands, 1987. 587 (1982).

[3] J. Novák, *Calculation of elastic stresses and strains inside a medium with multiple isolated inclusions*, Proceedings of the Sixth International Conference on Engineering Computational Technology (Stirlingshire, UK) (M. Papadrakakis and B.H.V. Topping, eds.), 2008, Paper 127, p. 16 pp.

[4] Jan Novák, Łukasz Kaczmarczyk, Peter Grassl, Jan Zeman, and Chris J Pearce, *A micromechanics-enhanced finite element formulation for modelling heterogeneous materials*, Computer Methods in Applied Mechanics and Engineering **201** (2012), 53–64.

[5] S.P. Oberrecht, J. Novk, and P. Krysl, *B-bar fems for anisotropic elasticity*, International Journal for Numerical Methods in Engineering **98** (2014), no. 2, 92–104.