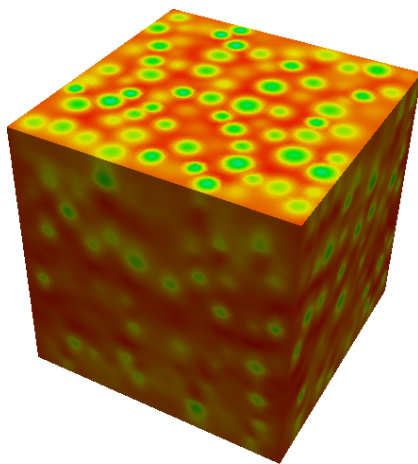


μ MECH

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

Theory manual & Program documentation



Jan Novák¹²

February 12, 2015

¹Corresponding address: novakj@cml.fsv.cvut.cz

²Czech Technical University in Prague, Faculty of Civil Engineering, Department of Mechanics

Project partners



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	Code	10
3.3	Interface functions	10
3.4	VTK file format	13
3.5	File structure of composite media description data set	13

Preface

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

This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU LGPL for more details.

You should have received a copy of the GNU LGPL along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

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.
- Oberrecht, S.P. and Novák, J. and Krysl, P., B-bar FEMs for anisotropic elasticity. *International Journal for Numerical Methods in Engineering* 98:92–104, 2014.

Chapter 1

Introduction

What's the μ MECH library about...

The library μ MECH was principally designed as a subclass of a finite element package. It provides sub-routines 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.

<i>Inclusion shape</i>	<i>Uniform eigenstrains</i>		<i>Non-uniform eigenstrains</i>	
	<i>Internal fields</i>	<i>External fields</i>	<i>Internal fields</i>	<i>External fields</i>
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, μ MECH will work less efficiently in terms of computational time.

Chapter 2

Theory manual

What's actually behind the scope of μ 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

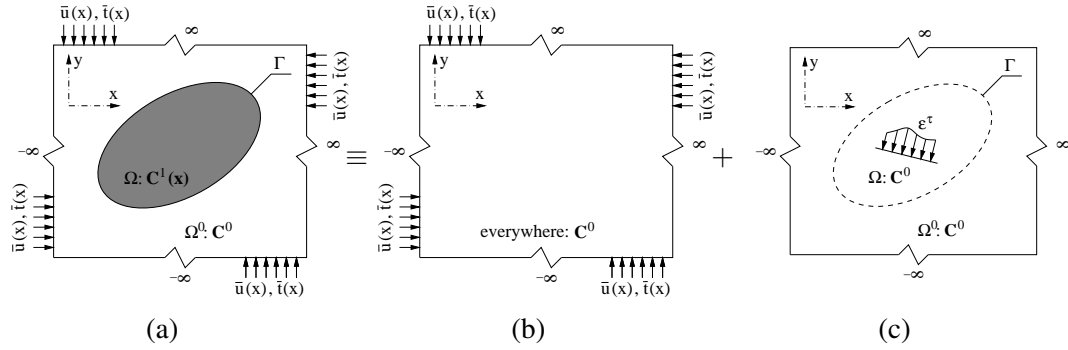


Figure 2.1: Principle of *Equivalent Inclusion Method*: a) inhomogeneity problem, b) problem of infinite homogeneous body, c) homogeneous inclusion problem

of the inhomogeneity problem consists from seeking the *equivalent transformation* 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

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

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

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

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

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

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 (disturbance) part of the strain, stress and displacement field, respectively. Moreover, the Eshelby's solution of *homogeneous inclusion problem*

$$\mathbf{u}^1(\mathbf{x}) = \boldsymbol{\mathcal{L}}(\mathbf{x}) : \boldsymbol{\varepsilon}^\tau, \quad (2.4)$$

$$\boldsymbol{\varepsilon}^1(\mathbf{x}) = \nabla_s \mathbf{u}^1(\mathbf{x}) = \nabla \boldsymbol{\mathcal{L}}(\mathbf{x}) \boldsymbol{\varepsilon}^\tau = \mathbf{S}(\mathbf{x}) : \boldsymbol{\varepsilon}^\tau \quad (2.5)$$

yields

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

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

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

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

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

which further yields

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

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

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

which can be finally recast in a compact form as

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

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

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

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 previous section, the

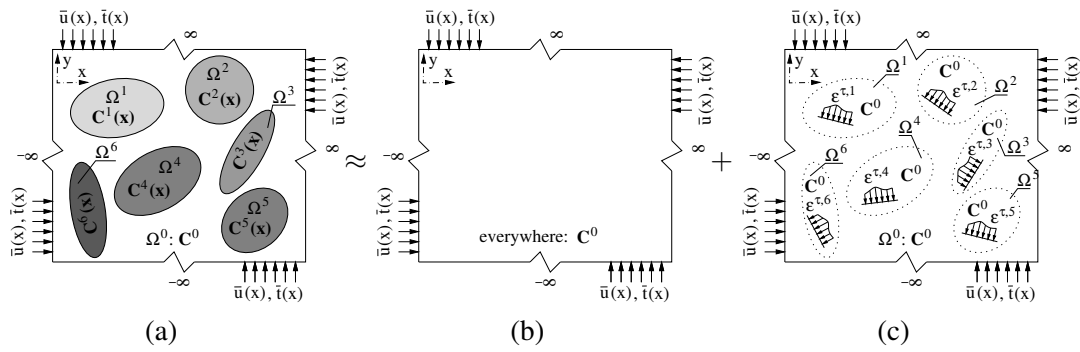


Figure 2.2: Principle of *Equivalent Inclusion Method*: a) multiple inhomogeneity problem, b) infinite homogeneous body, c) multiple homogeneous inclusion problem

stiffness of such a material is decomposed as follows [3, 2]

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

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 $\Omega = \Omega^1 \cup \dots \cup \Omega^N$, so that the characteristic function $V(\mathbf{x})$ yields

$$V(\mathbf{x}) = \begin{cases} 0 & \forall \mathbf{x} \in \Omega^0 \subset \mathbb{R}^3 \\ 1 & \forall \mathbf{x} \in \Omega \subset \mathbb{R}^3 \end{cases}. \quad (2.14)$$

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

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

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

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

which employing Eq. (2.15)³ turns into

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

where $\mathbf{S}^i(\mathbf{x})$ is the position dependent Eshelby's tensor of i^{th} inclusion and $\varepsilon^0(\mathbf{x}) = \varepsilon^0(\bar{\mathbf{u}}(\mathbf{x}), \bar{\mathbf{t}}(\mathbf{x}))$ stands for hypothetical *remote strain* field producing together with $\mathbf{C}^*(\mathbf{x})$ the required *transformation* eigenstrain $\varepsilon^{\tau,i}(\mathbf{x})$, 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}^* , ε^* and σ^* 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. ??). A FULL *self-balancing* algorithm ensures that the

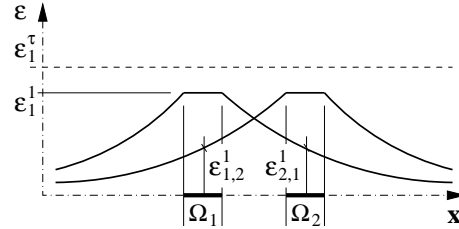


Figure 2.3: Principle of *self-balancing* algorithm for double inclusion problem in 1D, ε_1^τ denotes the initial transformation strain, ε_1^1 stands for the perturbation strain after 1st step, $\varepsilon_{1,2}^1$ represents the strain perturbation in inclusion Ω_1 caused by the presence of inclusion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

mechanical fields associated with inclusion i correctly reflect the influence of the remaining $N \setminus i$ 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 ε_i^τ is applied to each inclusion within the **step 2** (ε_1^τ in Fig. 2.3). Consequently, the perturbation strain ε_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 $\varepsilon_{j,i}^1$

at each inclusion centroid ($\epsilon_{1,2}^1$ and $\epsilon_{2,1}^1$ 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 referred to as the `_OPTIMIZED_` one.

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

Table 2.1: *Self-balancing* algorithm

Chapter 3

Tutorial

How to ...

3.1 Installation

Download and unpack `muMECH.zip` archive to a preferred directory, unzip it and compile, see file `help_install.txt` for detailed instructions. The resulting static library `libmumech.a` can be linked to your favourite software package. Do not forget include `problem.h` header file to allow calling all the addressed functions. There is also *main* file with number of examples of calling μ MECH functions. The executable file `mumechtest` with parameter `--tests` performs set of test functions.

3.2 Code

Tady napsat nejaký kecy, že se jede přes třídu `Problem`, že má prázdný konstruktor. Neco o `enumech` v `types.h`. Neco o struktuře `mumechu`...

Udelat 3 příklady, 1. vůbec nic se nenastavuje, načte se file a vytiskne 2. totéž ale direct 3. seznam toho, co je default ale dá se nastavit plus pomocné fce `visualize` atd. 4. kombinace 123

3.3 Interface functions

```
void read_input_file ( const char * filename )
```

Reads `filename` VTK input file containing inclusion geometry and topology.

`filename` – Name of the input vtk file.

§

```
void input_data_initialize_and_check_consistency ( void )
```

Initializes and checks consistency of all input data. This function has to be called after data input.

§

```
void convert_to_equivalent_problem ( void )
```

Converts the given heterogeneous problem to the equivalent problem.

§

```
void print_equivalent_problem ( const char * filename )
```

Prints the equivalent problem record into the `filename` VTK file.

`filename` – Name of the output VTK file.

§

```
long giveFieldsOfPoint ( double ** displc, double ** strain,
                        double ** stress, const double * coords,
                        char ptFlag, int rs, int nrs,
                        PFCmode pfcMode=PFCM_OPTIMIZED, long reqIncl=-3,
                        T2VreductNotation tn=TVRN_THEORETICAL_ROW ) const
```

Function gives the analytical solution of the perturbation or total fields (displacements, strains and stresses) of a point for given set of remote strains. The fields with NULL pointers are not computed. The resulting fields depend on the action region of surrounding inclusions of a given point if `pfcMode == PFCM_OPTIMIZED`. In case of one `lc`, send ukazatel, toto dodelej do vzorovych prikladu. termitovo

`coords` – Coordinates of a point.

`disp` – Set of `nrs` displacement vectors to be calculated (if `displc != NULL`). Vectors are saved in rows.

`strain` – Set of `nrs` strain tensors to be calculated (if `strain != NULL`). Tensors are saved in vector form, one tensor in one row of `strain` array.

`stress` – Set of `nrs` stress tensors to be calculated (if `stress != NULL`). Tensors are saved in vector form, one tensor in one row of `stress` array.

`ptFlag` – Flag defines the type of calculated fields, 'p' denotes 'perturbation' and 't' denotes 'total'.

`rs` – The first computed remote strain.

`nrs` – Number of computed remote strains.

`pfcMode` – Algorithm type of a point fields calculation (PFCM_FULL/PFCM_OPTIMIZED).

`reqIncl` – Number of the inclusion the point is supposed to lay inside. Allowed values: -3 - no suppose; -2 - inside of some inclusion; -1 - outside of all inclusions; 0 - inside of the inclusion.

`tn` – The notation of the strain/stress tensor to vector reduction.

§

```
long giveFieldsOfPointOneRS ( double * displc, double * strain,
                             double * stress, const double * coords,
                             char ptFlag, int rs,
                             PFCmode pfcMode=PFCM_OPTIMIZED, long reqIncl=-3,
                             T2VreductNotation tn=TVRN_THEORETICAL_ROW ) const
```

Function gives results for only one given remote strain. See function `giveFieldsOfPoint()` above.

§

```
void printFieldsOnMeshVTK ( const char * mesh_file_out, const char * mesh_file,
                           char ptFlag, int rs, int nrs,
                           PFCmode pfcMode=PFCM_OPTIMIZED ) const
```

Function computes all fields (displacements, strains and stresses) in nodes a mesh given via VTK file, see `giveFieldsOfPoint()` for details about parameters. For every required remote strain (`rs`, `nrs`) a mesh with values in nodes is printed to VTK file `mesh_file_out` with remote strain id as suffix.

`mesh_file_out` – Output file with mesh and computed fields in nodes.

`mesh_file` – Input file with mesh geometry.

`ptFlag` – Flag defines the type of calculated fields, 'p' denotes 'perturbation' and 't' denotes 'total'.

`rs` – The first computed remote strain.

`nrs` – Number of computed remote strains.

`pfcMode` – Algorithm type of a point fields calculation (PFCM_FULL/PFCM_OPTIMIZED).

§

```
void printFieldsOnMeshGrid ( const double * p1, const double * p2,
                            const long * n,
                            char ptFlag, int rs, int nrs,
                            PFCmode pfcMode=PFCM_OPTIMIZED ) const
```

Function computes all fields (displacements, strains and stresses) in nodes a regular orthogonal mesh/grid, see `giveFieldsOfPoint()` for details about parameters. Grid is given by coordinates of two diagonally opposed corners `p1` and `p2` and by count of segments `n`. For every required remote strain (`rs`, `nrs`) a mesh with values in nodes is printed to VTK file `mesh_file_out` with remote strain id as suffix.

`mesh_file_out` – Output file with mesh and computed fields in nodes.

`p1` – Coordinates of grid corner point with lower coordinates.

`p2` – Coordinates of grid corner point with upper coordinates.

`n` – Number of segments in the directions of particular axes. The grid is 2d when `n[2]==0`.

`ptFlag` – Flag defines the type of calculated fields, 'p' denotes 'perturbation' and 't' denotes 'total'.

`rs` – The first computed remote strain.

`nrs` – Number of computed remote strains.

`pfcMode` – Algorithm type of a point fields calculation (PFCM_FULL/PFCM_OPTIMIZED).

§

```
void print_visualization ( const char * filename, int n, int dim=0,
                          bool refined=false )
```

Triangulates inclusions surfaces and prints `filename` VTK file.

`filename` – Name of the output vtk file.

`n` – Number segments of a quater ellipse.

`dim` – Mesh dimension. In the case of 3d problem and `dim=2`, the 2d mesh is generated in the plane `z=0`.

`refined` – The mesh density varies according to ellipse curvature. The process of triangulation is slower.

3.4 VTK file format

At this point, the μ MECH I/O file syntax is build exclusively on freely available Visualization Toolkit - VTK ¹ file format, in particular on its both legacy and XML versions with UNSTRUCTURED_GRID dataset format. However, there is no obstacles to add support of user's favourite file format or version of VTK syntax if necessary.

The following data can be handled via files: the composite media description, ballanced internal fields, FE mesh and FE mesh with evaluated values (e.g. perturbation fields), see Fig. ???. In following sections, the file structure for particular data sets is described for legacy VTK version. The XML VTK syntax is similar a její struktura bude zřejmě z ...

3.5 File structure of composite media description data set

In brief, the 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 informations about the geometry of a calculated task. The particular meaning of each compulsory keyword mostly reflects its VTK counterpart and is as follows.

4.3. Exchange data file format The ow of data proceeds between DONKEY and MIDAS by means of les in VTK XML format, Table 1. The geometry is dened through initial pair of data blocks followed by the POINTS and CELLS keywords. Structural properties assigned to geometric elements are stored in

§

POINT_DATA and

§

CELL_DATA sections. To speed up the data ow, the ASCII is replaced with the binary format and the particular les are stored in virtual memory instead of the hard drive.

The example input file with complete description of 3d composite media with 3 inclusions is shown in Tab. ??. In souladu with VTK syntax, the second line is vyhrazen for comment with exception of first word. which determines dimension of the problem and nabyva tvaru 3D or 2D. Third and fourth lines should be same

¹<http://www.vtk.org/VTK/img/file-formats.pdf>

as in the example. The rest of file, from fifth line till end, is compound of data blocks indicated by compulsory keyword, which meaning mostly reflects its VTK counterpart.

Coordinates of inclusion centroids are given in section

§

POINTS. The keyword

§

CELLS znaci section with definition of cell connectivity (topology). In our case just simple points. Its cell type definitions is "1", see section

§

CELL_TYPES.

The unstructured section

§

FIELD contains generic information.

of the projects name, material specifications, cross-section characteristics, etc.

§

Inclusion_shape

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

symbolic constant	input file value
<code>_ELLIPSOID_</code>	1
<code>_SPHERE_</code>	2
<code>_ELLIPTIC_CYLINDER_</code>	3
<code>_CYLINDER_</code>	4
<code>_PENNY_</code>	5
<code>_CLOSED_PENNY_</code>	6
<code>_FLAT_ELLIPSOID_</code>	7
<code>_OBLATE_SPHEROID_</code>	8
<code>_PROLATE_SPHEROID_</code>	9

Table 3.1: Inclusion shape values as defined in eshelbySolTypes.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.

§

`Euler_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_`.

§

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

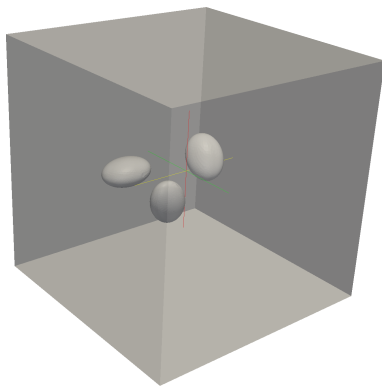


Figure 3.1: Geometry and topology of three inclusion benchmark

```

# vtk DataFile Version 3.0
3D This text is comment.
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
FIELD unstructured_data 2
Matrix_record 1 2 float
1.0 0.1
SBA_mode 1 1 int
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 Euler_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.

```

List of Figures

2.1	<i>Principle of Equivalent Inclusion Method: a) inhomogeneity problem, b) problem of infinite homogeneous body, c) homogeneous inclusion problem</i>	5
2.2	<i>Principle of Equivalent Inclusion Method: a) multiple inhomogeneity problem, b) infinite homogeneous body, c) multiple homogeneous inclusion problem</i>	7
2.3	<i>Principle of self-balancing algorithm for double inclusion problem in 1D, ε_1^T 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</i>	8
3.1	<i>Geometry and topology of three inclusion benchmark</i>	16

List of Tables

1.1	<i>Available mechanical fields with respect to applied eigenstrain and particular inclusion shape</i>	4
2.1	<i>Self-balancing algorithm</i>	9
3.1	<i>Inclusion shape values as defined in <code>eshelbySolTypes.h</code></i>	14
3.2	<i>VTK XML file generated by Donkey.</i>	17

Acknowledgements

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

CIDEAS - **C**enter of **I**ntegrated **D**esign 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.