

Periodic Boundary Conditions for RVEs

Python Code (`rve_mpc_generator.py`) to Construct Constraints for WARP3D¹
(Rectangular Prism RVEs)

Robert H. Dodds, Jr., PhD, NAE

rhodods@gmail.com

Updated: December 25, 2025 at 2:40pm

1 Purpose

The script generates a ready-to-use, WARP3D boundary-condition input file for a 3D representative volume element (RVE) using periodic displacement boundary conditions (PBCs). The 3D mesh must be a rectangular prism constructed of hex or tet elements and satisfy the node-pairing requirement. Pairs of nodes are located on opposite faces of the RVE whose in-plane coordinates are identical. They represent the same physical point in the periodic material and are coupled by the periodic displacement constraints. The codes NEPER and GMSH have capabilities to generate meshes with node pairing. The generated output file is written in WARP3D format:

- a **constraints** block (absolute displacement constraints), followed by
- a **multipoint** block (multi-point constraints, MPCs).

No manual editing of the generated constraints/MPC file should be required for a correctly defined periodic RVE mesh.

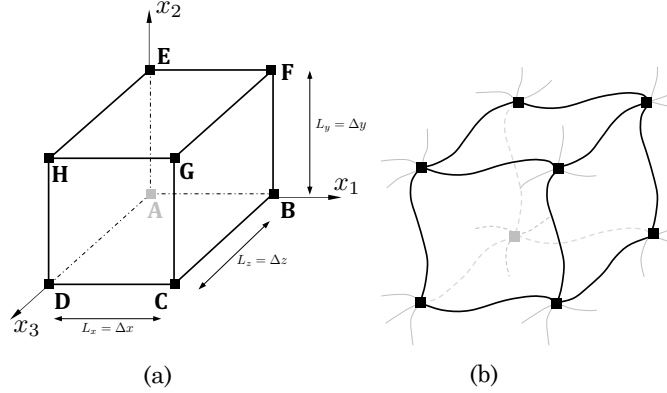


Fig. 1. (a) Labelling of vertexes, edges and faces for rectangular prism RVE. (b) Example periodic displacements.

Figure 1 illustrates a rectangular prism RVE with some notation used in this document.

The periodic loadings are specified by

$$u_1^{j+} - u_1^{j-} = \bar{\epsilon}_{11} \Delta x_1^j + \bar{\epsilon}_{12} \Delta x_2^j + \bar{\epsilon}_{13} \Delta x_3^j, \quad (1a)$$

$$u_2^{j+} - u_2^{j-} = \bar{\epsilon}_{21} \Delta x_1^j + \bar{\epsilon}_{22} \Delta x_2^j + \bar{\epsilon}_{23} \Delta x_3^j, \quad (1b)$$

$$u_3^{j+} - u_3^{j-} = \bar{\epsilon}_{31} \Delta x_1^j + \bar{\epsilon}_{32} \Delta x_2^j + \bar{\epsilon}_{33} \Delta x_3^j. \quad (1c)$$

¹The generated constraint equations, absolute and multi-point, maybe readily converted into those required for Abaqus.

where $\bar{\varepsilon}_{ik}$ denotes the known, average strain tensor at a material point to be imposed on the RVE. This general form is used to support algorithmic enforcement of periodicity and does not imply that the imposed field corresponds to a physically admissible macroscopic strain in the classical sense. In the present implementation, $\bar{\varepsilon}_{ik}$ is not required to be symmetric. Example: $\bar{\varepsilon}_{12}$ may be different than $\bar{\varepsilon}_{21}$. There exists no strict requirement that $\bar{\varepsilon}_{ik}$ actually define a real, macroscale strain tensor. Here it is an often employed framework to impose periodic loadings on the RVE.

Table 1 lists the symbolic MPC equations prior to elimination of zero terms, absolute constraints, and dummy-node substitutions.

2 Input data required by the script

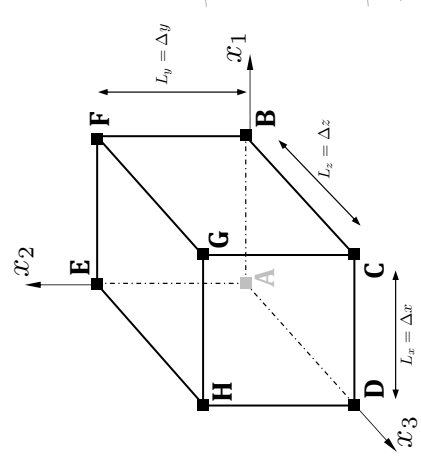
The input is a text file describing the RVE mesh and loading. Comment lines (or inline comments) may be included using the character `#`.

Required items include:

- number of nodes, excluding additional dummy-nodes required to enforce the PBCs. Element data may be present but is not used by the script,
- declared dimensions L_x , L_y , L_z of the rectangular prism (used for reporting and consistency checks),
- node numbers for the 8 vertex nodes $A-H$,
- the imposed 3×3 macroscopic strain tensor $\bar{\varepsilon}_{ij}$ (row-wise input). Does not need to be symmetric,
- an optional list of absolute constraints (DOFs fixed to zero). Usually to prevent rigid-body motion,
- a `DUMMY_EPS_MAP` block defining the dummy-node method for each strain component used, and
- the X, Y, Z coordinates of all nodes. Nodes must be numbered sequentially. Coordinate values are truncated at 7 significant figures after reading. The code expects the dummy-node pairs to be defined after all nodes of the RVE. Values for dummy nodes are ignored when included.

Face Nodes	Vertex Nodes	Edge Nodes
$u_i^{\text{FGCB}} - u_i^{\text{EHDA}} - L_x \bar{\epsilon}_{i1} = 0$	$u_i^{\text{G}} - u_i^{\text{A}} - L_x \bar{\epsilon}_{i1} - L_y \bar{\epsilon}_{i2} - L_z \bar{\epsilon}_{i3} = 0$	$\text{FG} \leftrightarrow \text{AD}:$ $u_i^{\text{FG}} - u_i^{\text{AD}} - L_x \bar{\epsilon}_{i1} - L_y \bar{\epsilon}_{i2} = 0$
$u_i^{\text{FEHG}} - u_i^{\text{BADC}} - L_y \bar{\epsilon}_{i2} = 0$	$u_i^{\text{F}} - u_i^{\text{D}} - L_x \bar{\epsilon}_{i1} - L_y \bar{\epsilon}_{i2} + L_z \bar{\epsilon}_{i3} = 0$	$\text{HG} \leftrightarrow \text{AB}:$ $u_i^{\text{HG}} - u_i^{\text{AB}} - L_y \bar{\epsilon}_{i2} - L_z \bar{\epsilon}_{i3} = 0$
$u_i^{\text{GHDC}} - u_i^{\text{FEAB}} - L_z \bar{\epsilon}_{i3} = 0$	$u_i^{\text{H}} - u_i^{\text{B}} + L_x \bar{\epsilon}_{i1} - L_y \bar{\epsilon}_{i2} - L_z \bar{\epsilon}_{i3} = 0$	$\text{GC} \leftrightarrow \text{EA}:$ $u_i^{\text{GC}} - u_i^{\text{EA}} - L_x \bar{\epsilon}_{i1} - L_z \bar{\epsilon}_{i3} = 0$
	$u_i^{\text{C}} - u_i^{\text{E}} - L_x \bar{\epsilon}_{i1} + L_y \bar{\epsilon}_{i2} - L_z \bar{\epsilon}_{i3} = 0$	$\text{BC} \leftrightarrow \text{EH}:$ $u_i^{\text{BC}} - u_i^{\text{EH}} - L_x \bar{\epsilon}_{i1} + L_y \bar{\epsilon}_{i2} = 0$
		$\text{FE} \leftrightarrow \text{CD}:$ $u_i^{\text{FE}} - u_i^{\text{CD}} - L_y \bar{\epsilon}_{i2} + L_z \bar{\epsilon}_{i3} = 0$
		$\text{FB} \leftrightarrow \text{HD}:$ $u_i^{\text{FB}} - u_i^{\text{HD}} - L_x \bar{\epsilon}_{i1} + L_z \bar{\epsilon}_{i3} = 0$

Table 1. MPC equations for faces, vertices, and edges in periodic boundary conditions. $i = 1, 2, 3$



3 Origin-independence via automatic detection of RVE bounds

The script does *not* require that the RVE occupy $x \in [0, L_x]$, $y \in [0, L_y]$, $z \in [0, L_z]$. Instead, the script detects the geometric bounds directly from the node coordinates:

$$x_{\min} = \min(x), \quad x_{\max} = \max(x), \quad (2)$$

$$y_{\min} = \min(y), \quad y_{\max} = \max(y), \quad (3)$$

$$z_{\min} = \min(z), \quad z_{\max} = \max(z), \quad (4)$$

and computes the detected dimensions:

$$L_x^{\det} = x_{\max} - x_{\min}, \quad L_y^{\det} = y_{\max} - y_{\min}, \quad L_z^{\det} = z_{\max} - z_{\min}. \quad (5)$$

The script reports both the declared dimensions (L_x, L_y, L_z) and the detected dimensions $(L_x^{\det}, L_y^{\det}, L_z^{\det})$ and issues a warning if they differ beyond tolerance.

4 Boundary node classification

Each RVE node is classified according to how many boundary planes it lies on:

- vertex node: on the intersection of three boundary planes. The 8 nodes are labelled $A-H$.
- edge node: on the intersection of two boundary planes, excluding vertex nodes
- face node: on exactly one boundary plane, excluding vertex and edge nodes
- interior node: on no boundary plane,

Boundary planes are detected using the automatically determined bounds:

- X^- plane at $x = x_{\min}$, X^+ plane at $x = x_{\max}$,
- Y^- plane at $y = y_{\min}$, Y^+ plane at $y = y_{\max}$,
- Z^- plane at $z = z_{\min}$, Z^+ plane at $z = z_{\max}$.

A coordinate tolerance (relative to the largest RVE dimension) is used when assigning nodes to planes.

5 Construction of periodic node pairings

5.1 Face-node pairings

Nodes on opposite faces are paired by matching the two transverse coordinates (within tolerance):

- $X^- \leftrightarrow X^+$: match (y, z) ,
- $Y^- \leftrightarrow Y^+$: match (x, z) ,
- $Z^- \leftrightarrow Z^+$: match (x, y) .

For each paired node $(-, +)$, the script computes the geometric offset vector

$$\Delta \mathbf{x} = \mathbf{x}^+ - \mathbf{x}^-, \quad (6)$$

which is translation-invariant and therefore independent of the global origin.

5.2 Edge-node pairings

Edge nodes are identified by the set of two boundary planes they lie on, *e.g.*, $\{X^-, Y^-\}$. Each family of opposite edges is paired using the remaining “free” coordinate as the edge parameter:

- $\{X^-, Y^-\} \leftrightarrow \{X^+, Y^+\}$, parameter axis z ,
- $\{X^-, Y^+\} \leftrightarrow \{X^+, Y^-\}$, parameter axis z ,
- $\{X^-, Z^-\} \leftrightarrow \{X^+, Z^+\}$, parameter axis y ,
- $\{X^-, Z^+\} \leftrightarrow \{X^+, Z^-\}$, parameter axis y ,
- $\{Y^-, Z^-\} \leftrightarrow \{Y^+, Z^+\}$, parameter axis x ,
- $\{Y^-, Z^+\} \leftrightarrow \{Y^+, Z^-\}$, parameter axis x .

For each paired edge-node pair, $\Delta \mathbf{x}$ is computed from coordinates as for face nodes.

5.3 Vertex-node pairings

The user supplies the node numbers for the eight vertices $A-H$. The script forms the four body-diagonal vertex pairings and computes $\Delta \mathbf{x}$ directly from the vertex coordinates; the origin location does not matter.

6 Dummy node method for WARP3D MPC implementation

WARP3D supports only multi-point equations (MPCs) that are homogeneous, *e.g.*,

$$27 \ 1.0 \ w \ - \ 20 \ 1.0 \ w \ - \ 49 \ 3.5 \ w = 0 \quad (7)$$

To make an equivalent, non-homogenous equation required to impose non-zero terms of $\bar{\varepsilon}_{ij}$, WARP3D uses the so-called dummy-node and rigid-link element method. The non-homogeneous terms are shifted to displacements applied at auxiliary (dummy) nodes, so that all MPCs become homogeneous. This greatly simplifies assembly of the equations and is compatible with the WARP3D existing constraint infrastructure. A similar procedure is often used to enforce PBCs in Abaqus.

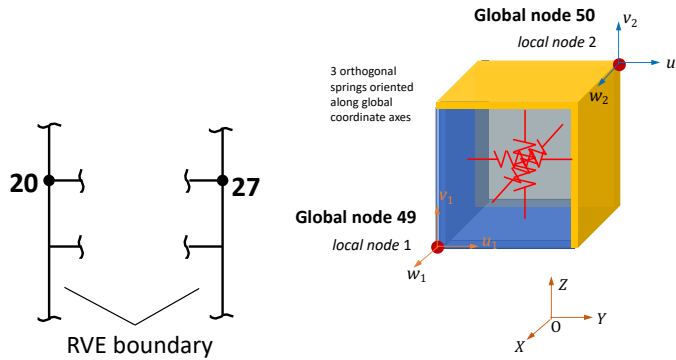


Fig. 2. Example demonstrating use of dummy nodes and rigid-link element to enable definition of homogeneous multi-point constraints. Local nodes 1, 2 of the link element are often coincident.

Figure 2 shows the setup for a simplified application of the dummy node-rigid link element method. The link2 element is connected to *dummy* nodes 49 and 50. For convenience these nodes are often defined

well outside the RVE – their actual (global) position is immaterial. Spring stiffness values for the element in global (x, y, z) directions are set to a sufficiently large value. Absolute constraints are imposed on node 50, *e.g.*,

$$50 \quad u = 0 \quad v = 0 \quad w = 0.0025 \quad (8)$$

Given the large spring stiffnesses for the link element, node 49 will have the same displacements as node 50. Node 49 thus has the same w displacement as node 50, while maintaining the required homogeneous form.

Most often, 6 link elements and corresponding dummy-node pairs are needed when all terms of $\bar{\varepsilon}_{ij}$ are non-zero.

6.1 DUMMY_EPS_MAP definition

The input file includes a `DUMMY_EPS_MAP` block with entries of the form:

```
i j dummy_node driver_node dof
```

Interpretation of a map entry (i, j) :

- (i, j) is the entry of the $\bar{\varepsilon}_{ij}$.
- `dummy_node` appears in the homogeneous MPC equation. Example: node 49 in Fig. 2.
- `driver_node` receives an absolute displacement constraint on the indicated `dof` whose value equals $\bar{\varepsilon}_{ij}$. Example: node 50 in Fig. 2.

The user connects `driver_node` and `dummy_node` using stiff link elements in the WARP3D model, so that the imposed driver displacement is transmitted to the dummy DOF used in the MPCs.

6.2 Resulting MPC structure

For a generic paired-node relation, the script generates a WARP3D MPC equation of the form

$$u_i^+ - u_i^- - \Delta x_1 d_{i1} - \Delta x_2 d_{i2} - \Delta x_3 d_{i3} = 0, \quad (9)$$

where d_{ij} denotes the dummy-node DOF representing $\bar{\varepsilon}_{ij}$. Terms are automatically omitted when:

- $\bar{\varepsilon}_{ij} = 0$,
- $\Delta x_j = 0$, or
- the referenced DOF is already fixed by an absolute constraint.

6.3 Absolute constraints and closure iteration

6.3.1 Absolute constraints emitted by the script

Absolute constraints are emitted under the WARP3D keyword `constraints`. The script writes two classes of absolute constraints:

- User-specified zero constraints from the `ABS.CONSTRAINTS` section of the input file.
- Driver constraints implied by the strain tensor and the `DUMMY_EPS_MAP`:

$$u(\text{driver_node}, \text{dof}) = \bar{\varepsilon}_{ij} \quad (10)$$

for each nonzero $\bar{\varepsilon}_{ij}$ present in the map.

A consistency check is performed: a dof cannot be both fixed to zero and assigned a nonzero driver value.

6.3.2 Implied constraints and two-pass “closure”

When some strain components in $\bar{\varepsilon}_{ij}$ are zero and/or absolute constraints are applied, some MPC equations will be simplified. If an MPC equation collapses to a single remaining *physical* term, *e.g.*, $63u = 0$, then it becomes an implied absolute constraint rather than an MPC. To detect these situations, the script performs an iterative “closure”:

- generate MPCs under the current set of absolute constraints,
- collect any implied absolute constraints,
- add them to the absolute constraint set,
- repeat until no new implied constraints are found, thereby ensuring that all constraints are output in valid WARP3D order and that no incomplete MPCs remain.

6.4 Output file structure and WARP3D ordering

The output file generated by the script begins with a comment header describing:

- input file name,
- detected bounds and dimensions,
- declared vs. detected dimension comparison,
- the imposed strain tensor,
- the DUMMY_EPS_MAP summary, and
- any warnings.

The output then contains:

- a `constraints` block (absolute constraints), followed by
- a `multipoint` block (MPC equations).

This ordering matches the WARP3D requirement that absolute constraints appear before MPC definitions.

7 Robustness and large meshes

Node-pairing is performed in the script using simple coordinate-key hashing (rounded transverse coordinates), which is efficient for large boundary node sets. The total number of MPC equations scales with the number of paired boundary nodes and three displacement components.

The script has been exercised on large RVEs (10^5 nodes) and produces tens of thousands of MPC equations with no manual editing required in typical workflows.

Execution time for the Python script is a few seconds for large models.

8 Example Problem

This example demonstrates the concepts described here and develops an input file for the Python program `rve_mpc_generator.py`. See Fig. 3 for the setup. The analysis is linear-elastic for simplicity. Extension for nonlinear behavior does not alter the constraints generated by the program.

The model has 8, `l3disop` elements (8-node isoparametrics); 4 `link2` elements; 27 nodes for the prism; 4 dummy-node pairs (8 total nodes); and a linear-elastic material.

Figure 4 lists the node coordinates and element connectivities.

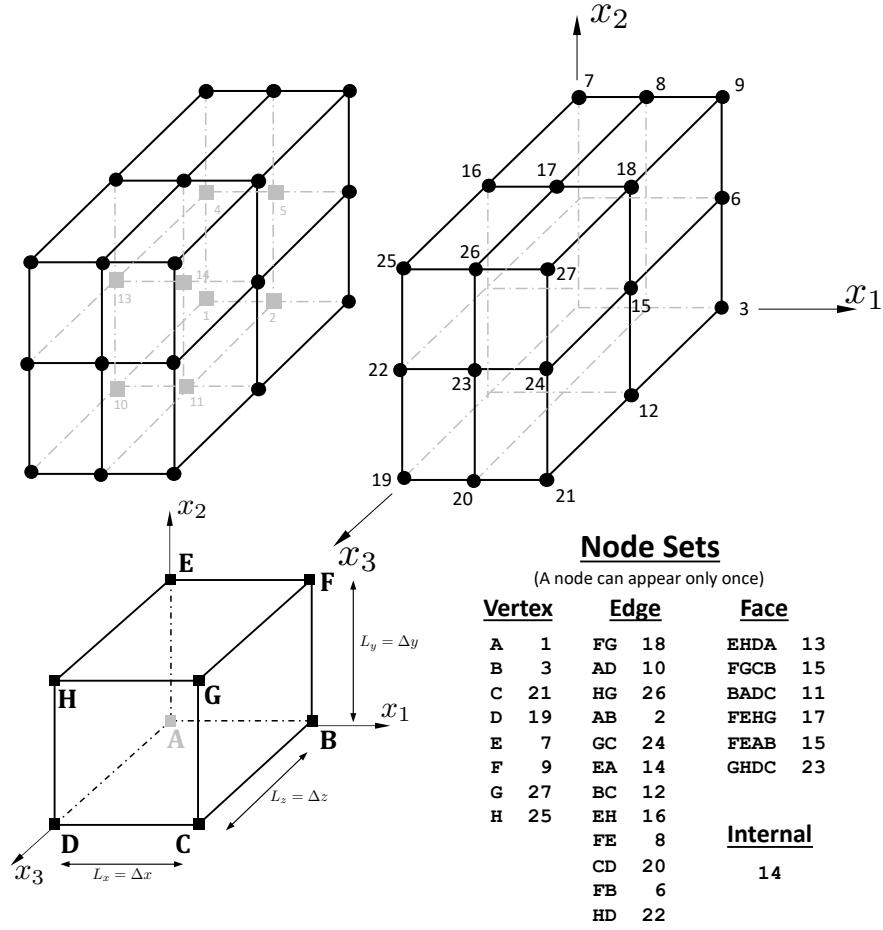


Fig. 3. Small example problem to illustrate setup and use of the `rve_mpc_generator.py` Python program.

The prism dimensions are $L_x = 1, L_y = 2, L_z = 4$ to emphasize that imposed strains are multiplied by these sizes to obtain corresponding imposed displacements. The macroscale strain tensor is

$$\bar{\epsilon} = \begin{bmatrix} \bar{\epsilon}_{11} & \bar{\epsilon}_{12} & \bar{\epsilon}_{13} \\ \bar{\epsilon}_{21} & \bar{\epsilon}_{22} & \bar{\epsilon}_{23} \\ \bar{\epsilon}_{31} & \bar{\epsilon}_{32} & \bar{\epsilon}_{33} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 & 0.5 \\ 0.2 & 0.0 & 0.3 \\ 0.5 & 0.3 & 0.0 \end{bmatrix} \quad (11)$$

where values are chosen for convenience to make obvious the results are correct.

One dummy-node pair and link2 element for each non-zero strain component are defined (here the macroscale shear strains are symmetric). The (2,2) and (3,3) strain values could be non-zero with addition of 2 more link2 elements and 2 more dummy-node pairs.

Coordinates for dummy nodes 28-35 are set at 10.0 10.0 10.0. The values are immaterial and do not affect the solution. link2 stiffnesses are set at 10^{10} – preliminary analyses showed this value is sufficient to make displacements at both nodes of link2 elements identical.

The full WARP3D input file is shown in Fig. 5. Figure 5a shows the included file of constraints data.

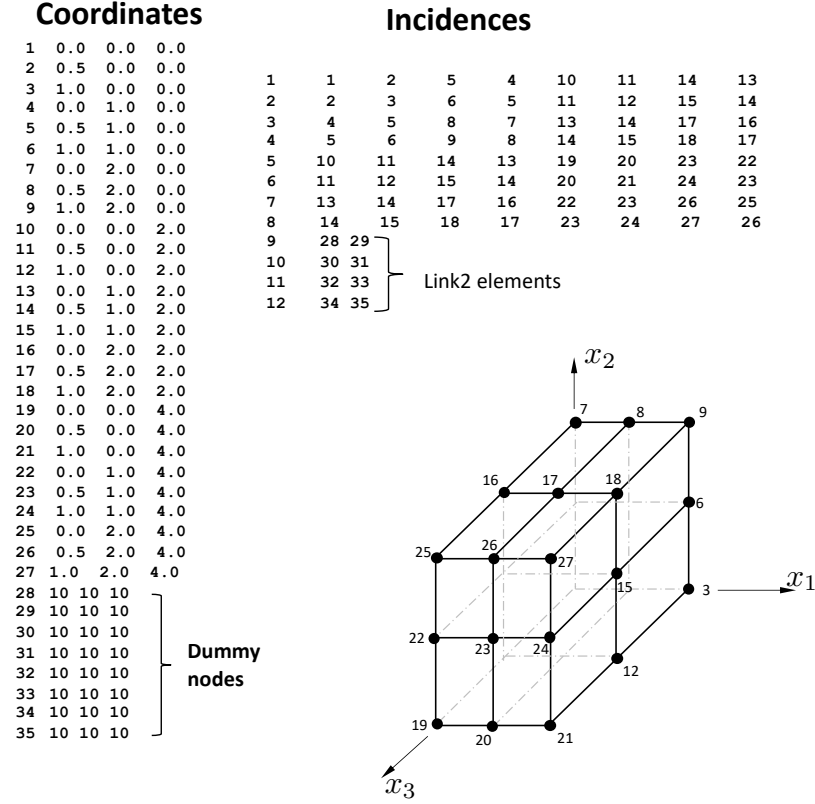


Fig. 4. Coordinates and incidences for the WARP3D input file. Put into file named `coords-incid.inp`.

8.1 User generation of constraints

For this simple problem, it is feasible to generate the absolute and MPC constraints manually. The next section uses the Python script.

Recommended steps:

- Decide upon absolute constraints to prevent 3 rigid-body translations and 3 rotations (rigid-body constraints). For this loading in Eq. (11), only node 1 needs to be constrained ($u = v = w = 0$). The non-zero imposed strains are sufficient to prevent the three rigid-rotations.
- Start with all equations in Table 1. Eliminate all the zero terms from the zero imposed $\bar{\epsilon}_{ij}$ values and constraints to remove rigid-body motions.
- Some of the MPCs to enforce the periodic boundary conditions may then have only 1 term remaining. The MPC has become an absolute constraint and must be included with them in the constraints. In WARP3D input, absolute constraints may not be intermixed with MPCs.

The imposed displacements for dummy-node loading become:

$$\begin{aligned}
\text{Node 29: } & u = 0.1, v = 0.0, w = 0.0 \leftarrow \bar{\epsilon}_{11} \\
\text{Node 31: } & u = 0.2, v = 0.2, w = 0.0 \leftarrow \bar{\epsilon}_{12} = \bar{\epsilon}_{21} \\
\text{Node 33: } & u = 0.5, v = 0.0, w = 0.5 \leftarrow \bar{\epsilon}_{13} = \bar{\epsilon}_{31} \\
\text{Node 35: } & u = 0.0, v = 0.3, w = 0.3 \leftarrow \bar{\epsilon}_{23} = \bar{\epsilon}_{32}
\end{aligned} \tag{12}$$

```

material steel
  properties mises e 30000 nu 0.3 n_power 10 yld_pt 60.e10 $
linear-elastic
!
material rve_link
  properties link stiff_link 1.0e10 mass_link 0.0
!
structure prism
!
number of nodes 35 elements 12
!
elements
  1-8 type l3disop linear material steel order 2x2x2 bbar,
                                center_output short
  9 10 11 12 type link2 material rve_link
!
*echo off
*input from file "coords_incid.inp"
*echo on
!
blocking automatic
!
*input from file "rve_constraints.inp"
!
loading test
nonlinear
  step 1 constraints 1.0
!
nonlinear analysis parameters $ only those really needed here
  solution technique sparse direct
  time step 1.0e06
  maximum iterations 5 $ global Newton iterations
  minimum iterations 1
  convergence test norm res tol 0.001
  batch messages off
  trace solution on
!
output model flat patran convention text file "model"
!
compute displacements for loading test step 1
output wide strains 1-8
output wide stresses 1-12
output wide displacements 1-35
output flat text displacements $ for ParaView visualization
output flat text element stresses $ for ParaView visualization
!
stop

```

Fig. 5. WARP3D input file for the example problem.

8.2 Constraints using rve_mpc_generator.py

Figure 6 shows the input file prepared to generate all constraints for the example problem. The required input for a very much larger model differs only in the amount of nodal coordinates data.

At execution, the script requests the input file name, the file name for generated constraints data and whether or not the symbolic form of all the MPC equations should be printed as comment lines in the constraints data file – useful for understanding the conversion of all MPCs in Table 1 for the model.

Figure 7 shows the generated constraints file in form for direct input to WARP3D.

```

constraints
1 u 0.0 v 0.0 w 0.0
29 u 0.1
31 u 0.2 v 0.2
33 u 0.5 w 0.5
35 v 0.3 w 0.3
!
multipoint
15 1.0 u - 13 1.0 u - 28 1.0 u = 0.
15 1.0 v - 13 1.0 v - 30 1.0 v = 0.
15 1.0 w - 13 1.0 w - 32 1.0 w = 0.
17 1.0 u - 11 1.0 u - 30 2.0 u = 0.
17 1.0 v - 11 1.0 v = 0.
17 1.0 w - 11 1.0 w - 34 2.0 w = 0.
23 1.0 u - 5 1.0 u - 32 4.0 u = 0.
23 1.0 v - 5 1.0 v - 34 4.0 v = 0.
23 1.0 w - 5 1.0 w = 0.
18 1.0 u - 10 1.0 u - 28 1.0 u - 30 2.0 u = 0.
18 1.0 v - 10 1.0 v - 30 1.0 v = 0.
18 1.0 w - 10 1.0 w - 32 1.0 w - 34 2.0 w = 0.
12 1.0 u - 16 1.0 u - 28 1.0 u + 30 2.0 u = 0.
12 1.0 v - 16 1.0 v - 30 1.0 v = 0.
12 1.0 w - 16 1.0 w - 32 1.0 w + 34 2.0 w = 0.
24 1.0 u - 4 1.0 u - 28 1.0 u - 32 4.0 u = 0.
24 1.0 v - 4 1.0 v - 30 1.0 v - 34 4.0 v = 0.
24 1.0 w - 4 1.0 w - 32 1.0 w = 0.
6 1.0 u - 22 1.0 u - 28 1.0 u + 32 4.0 u = 0.
6 1.0 v - 22 1.0 v - 30 1.0 v + 34 4.0 v = 0.
6 1.0 w - 22 1.0 w - 32 1.0 w = 0.
26 1.0 u - 2 1.0 u - 30 2.0 u - 32 4.0 u = 0.
26 1.0 v - 2 1.0 v - 34 4.0 v = 0.
26 1.0 w - 2 1.0 w - 34 2.0 w = 0.
8 1.0 u - 20 1.0 u - 30 2.0 u + 32 4.0 u = 0.
8 1.0 v - 20 1.0 v + 34 4.0 v = 0.
8 1.0 w - 20 1.0 w - 34 2.0 w = 0.
27 1.0 u - 28 1.0 u - 30 2.0 u - 32 4.0 u = 0.
27 1.0 v - 30 1.0 v - 34 4.0 v = 0.
27 1.0 w - 32 1.0 w - 34 2.0 w = 0.
9 1.0 u - 19 1.0 u - 28 1.0 u - 30 2.0 u + 32 4.0 u = 0.
9 1.0 v - 19 1.0 v - 30 1.0 v + 34 4.0 v = 0.
9 1.0 w - 19 1.0 w - 32 1.0 w - 34 2.0 w = 0.

```

Fig. 5a. Manually prepared constraints input file.

```

#
# comment lines begin with # in column 1
# all text beyond a # within a line is treated as comment
#
# number of nodes, elements # do not include dummy nodes/links
# number of elements not actually used.
#
27, 8
#
# Lx, Ly, Lz # dimensions of the rectangle prism RVE mesh
#
1.0, 2.0, 4.0
#
# Vertex nodes of mesh: A, B, C, D, E, F, G. See notes for ordering
#
1, 3, 21, 19, 7, 9, 27, 25
#
# 3x3 **usually** a symmetric** strain tensor. See notes for non-symmetric
# Symmetric version conforms to usual tensor definition of strains.
#
0.1 0.2 0.5 # eps_xx, eps_xy, eps_xz
0.2 0.0 0.3 # eps_xy, eps_yy, eps_yz
0.5 0.3 0.0 # eps_xz, eps_yz, eps_zz
#
# nodes with absolute constraints = 0 to suppress rigid-body motions.
# the generated constraints file will include these
#
ABS_CONSTRAINTS 1 # just to prevent rigid body motions.
1 u v w
#
# At most, 6 dummy node pairs are required to impose all
# quantities of the strain tensor as the RVE loading:
# 1 dummy node pair each for eps_11, eps_22, eps_33
# 1 dummy node pair for eps_12, eps_21
# 1 dummy node pair for eps_13, eps_31
# 1 dummy node pair for eps_23, eps_32
#
# Only those actually used need to be specified in the input here.
#
# The script outputs additional absolute constraints to impose non-zero
# strain tensor values on the 2nd dummy node attached to link2 element
#
DUMMY_EPS_MAP 7 # 7 lines < generated constraints>
1 1 28 29 u # eps_11 -> sets node 29 u = eps_11 value
#
1 2 30 31 u # eps_12 -> sets node 31 u = eps_12 value
2 1 30 31 v # eps_21 -> sets node 31 v = eps_21 value
#
1 3 32 33 u # eps_13 -> sets node 33 u = eps_13 value
3 1 32 33 w # eps_31 -> sets node 33 w = eps_31 value
#
2 3 34 35 v # eps_23 -> sets node 35 v = eps_23 value
3 2 34 35 w # eps_32 -> sets node 35 w = eps_32 value
#
# nodal x, y, z coordinates. must be sequential. script reads only
# number of nodes listed above - set that number to *not* include
# dummy nodes.
#
1 0.0 0.0 0.0
2 0.5 0.0 0.0
...
27 1.0 2.0 4.0

```

Fig. 6. Input file for the rve_mpc_generator.py Python program

```

! -----
! Enforced strain tensor eps_ij (row-wise):
!   row 1: 0.1  0.2  0.5
!   row 2: 0.2  0.0  0.3
!   row 3: 0.5  0.3  0.0
!
! Dummy strain mapping (eps_ij -> dummy_node, driver_node, dof):
!   eps_11 -> dummy 28 u, driver 29 u
!   eps_12 -> dummy 30 u, driver 31 u
!   eps_13 -> dummy 32 u, driver 33 u
!   eps_21 -> dummy 30 v, driver 31 v
!   eps_23 -> dummy 34 v, driver 35 v
!   eps_31 -> dummy 32 w, driver 33 w
!   eps_32 -> dummy 34 w, driver 35 w
!
! --- ABSOLUTE CONSTRAINTS (must appear before 'multipoint') ---
constraints
  29 u 0.1
  31 u 0.2
  31 v 0.2
  33 u 0.5
  33 w 0.5
  35 v 0.3
  35 w 0.3
  1 u 0.0
  1 v 0.0
  1 w 0.0
!
multipoint
  15 1.0 u - 13 1.0 u - 28 1.0 u = 0.
  15 1.0 v - 13 1.0 v - 30 1.0 v = 0.
  15 1.0 w - 13 1.0 w - 32 1.0 w = 0.
  17 1.0 u - 11 1.0 u - 30 2.0 u = 0.
  17 1.0 v - 11 1.0 v = 0.
  17 1.0 w - 11 1.0 w - 34 2.0 w = 0.
  23 1.0 u - 5 1.0 u - 32 4.0 u = 0.
  23 1.0 v - 5 1.0 v - 34 4.0 v = 0.
  23 1.0 w - 5 1.0 w = 0.
  18 1.0 u - 10 1.0 u - 28 1.0 u - 30 2.0 u = 0.
  ... same as Figure 5a ...
  25 1.0 w - 3 1.0 w + 32 1.0 w - 34 2.0 w = 0.
  21 1.0 u - 7 1.0 u - 28 1.0 u + 30 2.0 u - 32 4.0 u = 0.
  21 1.0 v - 7 1.0 v - 30 1.0 v - 34 4.0 v = 0.
  21 1.0 w - 7 1.0 w - 32 1.0 w + 34 2.0 w = 0.
!
! Total zero absolute constraints (final, incl. implied): 3
! Total driver constraints (from eps_ij): 7
! Total MPC equations emitted: 39
! -----

```

Fig. 7. Output file of WARP3D constraints data generated by the script.