

Article

Methodology for Developing a Numerical Model of a Masonry Wall Using the NSCD Method and LMGC90 Software

Dalibor Gelo *, Časlav Dunović, Šime Serdarević and Nina Šantek

Civil Engineering Department, Zagreb University of Applied Sciences, Av. Većeslava Holjevca 15, 10000 Zagreb, Croatia; caslav.dunovic@tvz.hr (Č.D.); sime.serdarevic@tvz.hr (Š.S.); nina.santek@tvz.hr (N.Š.)

* Correspondence: dalibor.gelo@tvz.hr

Abstract

This paper presents a comprehensive methodology for developing a numerical model of a masonry wall using the Non-Smooth Contact Dynamics (NSCD) method implemented in the open-source software LMGC90 version 2025. The modeling procedure relies on Python scripting and includes defining material properties, importing geometry from CAD tools, configuring the model, and specifying contact interactions between discrete elements. Each brick is modeled as an individual rigid element, allowing realistic simulation of frictional and cohesive behavior at joints. It outlines key theoretical aspects of the NSCD framework, including the formulation of global and local variables, interaction laws, and numerical integration. Numerical examples demonstrate the discrete element approach's ability to capture complex in-plane and out-of-plane structural phenomena induced by seismic loading and differential foundation settlement. The results highlight the advantages of discrete modeling in representing discontinuities and failure processes that are difficult to simulate with a conventional continuum-based approach.

Keywords: discrete element method; in-plane behavior; LMGC90 software; masonry wall; NSCD method; out-of-plane behavior

1. Introduction

In the field of computational modeling of structural systems, especially those involving complex interactions between discrete elements, specialized numerical tools are increasingly being used to accurately simulate contact problems between discrete elements. One such tool is the LMGC90 software package, developed at the University of Montpellier in collaboration with the French National Centre for Scientific Research (CNRS) [1–3].

Software package LMGC90 is an open-source software designed for modeling discrete element problems involving dry granular materials, masonry structures, and composite systems with rigid or elastic components. Thanks to the implementation of the Non-Smooth Contact Dynamics (NSCD) method, LMGC90 enables realistic simulation of complex interactions among discrete elements, including friction and cohesion effects [1,4].

The development of the program began in the early 2000s, and significant improvements, such as the introduction of a Python interface, have further enhanced its accessibility and functionality. Today, LMGC90 is used in various research and engineering contexts where analysis of the behavior of discrete systems in two-dimensional and three-dimensional spaces is required [1,5].

Recent studies have demonstrated the effectiveness of discrete element and non-smooth contact approaches in modeling the seismic response of masonry structures, includ-



Academic Editor: Boumediene Nedjar

Received: 27 January 2026

Revised: 24 February 2026

Accepted: 25 February 2026

Published: 27 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

ing the representation of cracking, separation, and collapse mechanisms that are difficult to capture using continuum-based methods. Ref. [6] illustrates the potential of advanced numerical frameworks for simulating masonry behavior under dynamic loading, supporting the applicability of NSCD-based approaches for seismic analysis. Data-driven and machine learning techniques have recently been explored to predict seismic response and classify damage states of masonry structures [7].

This paper presents the methodology for developing a numerical model of a masonry wall made of solid bricks, including a detailed description of the modeling steps, material definition, geometry setup, and contact laws between elements. Particular emphasis is placed on the implementation of the model in Python using the LMGC90 libraries, covering tasks such as importing geometry from CAD tools, defining interactions between discrete elements, and applying contact laws such as the Coulomb and Mohr–Coulomb models.

The paper includes four numerical examples that illustrate the behavior of masonry under different loading conditions. The first two examples analyze the in-plane response of the structure under earthquake loading and differential foundation settlement, while the latter two examine the out-of-plane behavior of the wall with and without horizontal restraint. The seismic excitation is based on the El Centro NS earthquake record, which has been numerically integrated into a form suitable for discrete dynamic system analysis.

Learning the LMGC90 software is difficult due to the lack of tutorials. The software includes a useful database of examples, but without explanations, and the models are described using mathematical functions, which are not intuitive and can be hard to understand. Describing the modeling methodology in this paper will help civil engineers use software more intuitively by incorporating graphical tools into the modeling process. The focus of this paper is the development of a realistic numerical model of masonry, enabling the simulation of complex contact problems between its constituent elements—an approach particularly relevant to civil engineering, especially in the context of the seismic resistance of masonry structures [8,9]. Four examples are presented, and all necessary information is publicly available on GitHub. This paper and the presented codes will help new users accelerate their learning or deepen the knowledge of existing users. The presented examples are practical engineering problems that can be easily extended to more complex ones.

2. Basic Theory of NSCD in LMGC90 Software

The LMGC90 software was developed at the University of Montpellier (Laboratoire de Mécanique et Génie Civil) in collaboration with the French National Centre for Scientific Research (CNRS) [1]. The first version of the program was created in early 2000. The source code was originally written in Fortran, and between 2009 and 2011, a transition was made to a Python interface. This transition significantly simplified the use of the program and made it more accessible to a wider audience. The latest version of the software was released in 2024 [5]. Unlike many existing discrete modeling tools, which are often proprietary, the proposed modeling procedure is implemented using the open-source software LMGC90, thereby supporting the transparency and reproducibility of the numerical analyses.

Software is used to model discrete problems such as dry granular materials, masonry structures, and rigid granular materials embedded in rigid or flexible matrices [3,5]. Numerical problems can be solved in both two-dimensional (2D) and three-dimensional (3D) space. The software implements the Non-Smooth Contact Dynamics (NSCD) method, which, according to its developers [1], provides a more accurate description of interactions between different bodies. The discrete elements can be modeled as perfectly rigid or elastic, while friction and cohesion phenomena can be represented at the contact points between two discrete elements [4,8,9].

2.1. Contact Detection and Local Reference Frame

Bodies in potential contact are named the candidate and the antagonist [1,4]. When defining the model, the candidate refers to the body on which potential contact points are defined—typically at protruding corners or edges. In Figure 1, the potential contact point is marked C. The antagonist is the body with which contact may potentially occur. The contact point is determined by finding the shortest distance between point C on the candidate and the surface or boundary of the antagonist. In Figure 1, the minimum distance is represented by the line segment connecting points C and A. Point A is defined as the intersection of a line and the potential contact boundary. The line passes through point C and is parallel to the normal of the contact surface on the antagonist. For each pair of points C and A, a local coordinate system can be defined, consisting of the normal vector n and two tangential vectors s and t corresponding to the tangential planes.

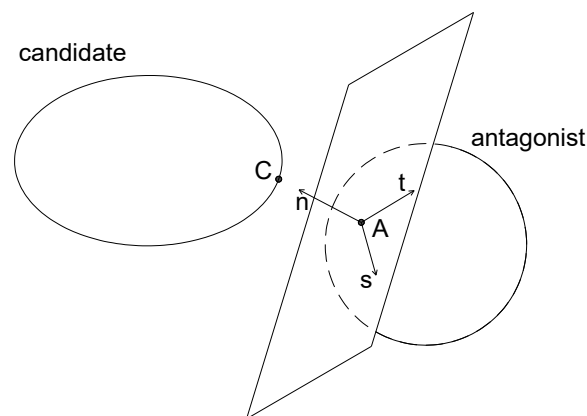


Figure 1. Illustration of the point of interaction in 3D.

To solve the dynamic problem, it is necessary to define both global and local unknowns. The global (kinematic) unknowns include displacements (q), velocities (V), and the resultant forces and moments (R) of the body's center of mass or of a mesh node. The local unknowns include the gap (g), velocity (v), and force (r) associated with each interaction or contact. This separation of global and local unknowns allows us to solve the equations of motion at the system level and the contact and friction laws at the local level.

2.2. Global Equations of Motion

If we neglect internal forces for a system of rigid bodies, global equations of motion can be written as

$$M\dot{V} = F_{ext}(t) + R, \quad (1)$$

where V is the vector of generalized velocities, M is the inertia (mass) matrix of the system, $F_{ext}(t)$ represents the external forces, and R is the resultant of the contact forces [1,2]. Integrating Equation 1 over the time interval $[t_i, t_{i+1}]$ results in the following:

$$M(V_{i+1} - V_i) = \int_{t_i}^{t_{i+1}} F_{ext}(t) dt + \int_{t_i}^{t_{i+1}} R dt. \quad (2)$$

Because contact forces between rigid bodies are instantaneous, we do not observe force over time. The contact contribution is expressed in terms of a contact impulse:

$$P_{i+1} = \int_{t_i}^{t_{i+1}} R dt. \quad (3)$$

Further, the coordinates q_{i+1} at step $i + 1$ are defined by the following expression:

$$q_{i+1} = q_i + \int_{t_i}^{t_{i+1}} V dt. \quad (4)$$

According to the LMGC90 guidelines [1], the integral problem in Equations (2) and (4) is approximated using the θ -method [5]. For problems involving rigid contacts with discontinuities, as in the NSCD method, it is recommended to choose $\theta = 1$, which ensures computational stability [1]. Then Equations (2) and (4) become

$$M(V_{i+1} - V_i) = \Delta t F_{ext}^{i+1}(t) + P_{i+1}, \quad (5)$$

$$q_{i+1} = q_i + \Delta t V_{i+1}, \quad (6)$$

where i defines a discrete time-step index and Δt is the time increment.

The local relative velocities at the contact points are related to the global velocity vector through linear kinematic mapping:

$$v_{i+1} = H V_{i+1}, \quad (7)$$

where H is an operator that maps the global translational and rotational velocities of the bodies to the normal and tangential components of the relative velocity at each contact point. The operator H is constructed from the geometry of the system and the position of the contact points with respect to the centers of mass of the interacting bodies [1].

The corresponding global contact impulse is obtained through the transpose operator

$$P_{i+1} = H^T p_{i+1}, \quad (8)$$

where p_{i+1} denotes the vector of local contact impulses. The transpose operator H^T projects the local contact impulses back into equivalent global forces and moments [2,5].

Also, the diagram in Figure 2 shows that the connection between the local velocity coordinates v and the momentum p is defined by the interaction law.

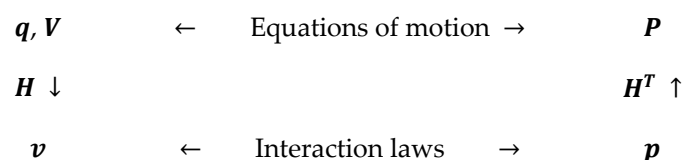


Figure 2. Diagram of the interaction between local and global unknowns.

Based on the above, the conclusion is that the NSCD method decomposes the problem into solving the global equations of motion, local interaction laws, and the mapping between local and global variables [2,3,8].

2.3. LMGC90 Workflow

The process of using the LMGC90 software can be divided into three steps: model creation, calculation, and post-processing. All these steps are displayed in Figure 3.

The model is created by defining parameters in the `gen_sample.py` file. First, it is necessary to create containers to store all the data needed to build a numerical model, including material characteristics, model type, body shapes, interaction laws, visibility, and boundary conditions. After running `gen_sample.py`, a new DATBOX folder is created that contains all the data needed to run the numerical calculation. If external loads are used, such as seismic recordings, the data file must be recorded in the DATBOX folder.

The numerical simulation starts by running the `command.py` file. The file contains commands to read data from the DATBOX folder and information about the following: space dimension, time evolution parameters, theta integrator parameter, nonlinear Gauss–Seidel parameters, write parameter, display parameters. Multiprocessing can be applied if we use the following command: `mpirun -np x python command.py`, where x defines the number of cores.

After the calculation is finished, a new folder is created that contains the file necessary for post-processing. In the DISPLAY folder are files that can be opened with ParaView, which allows users to visualize results.

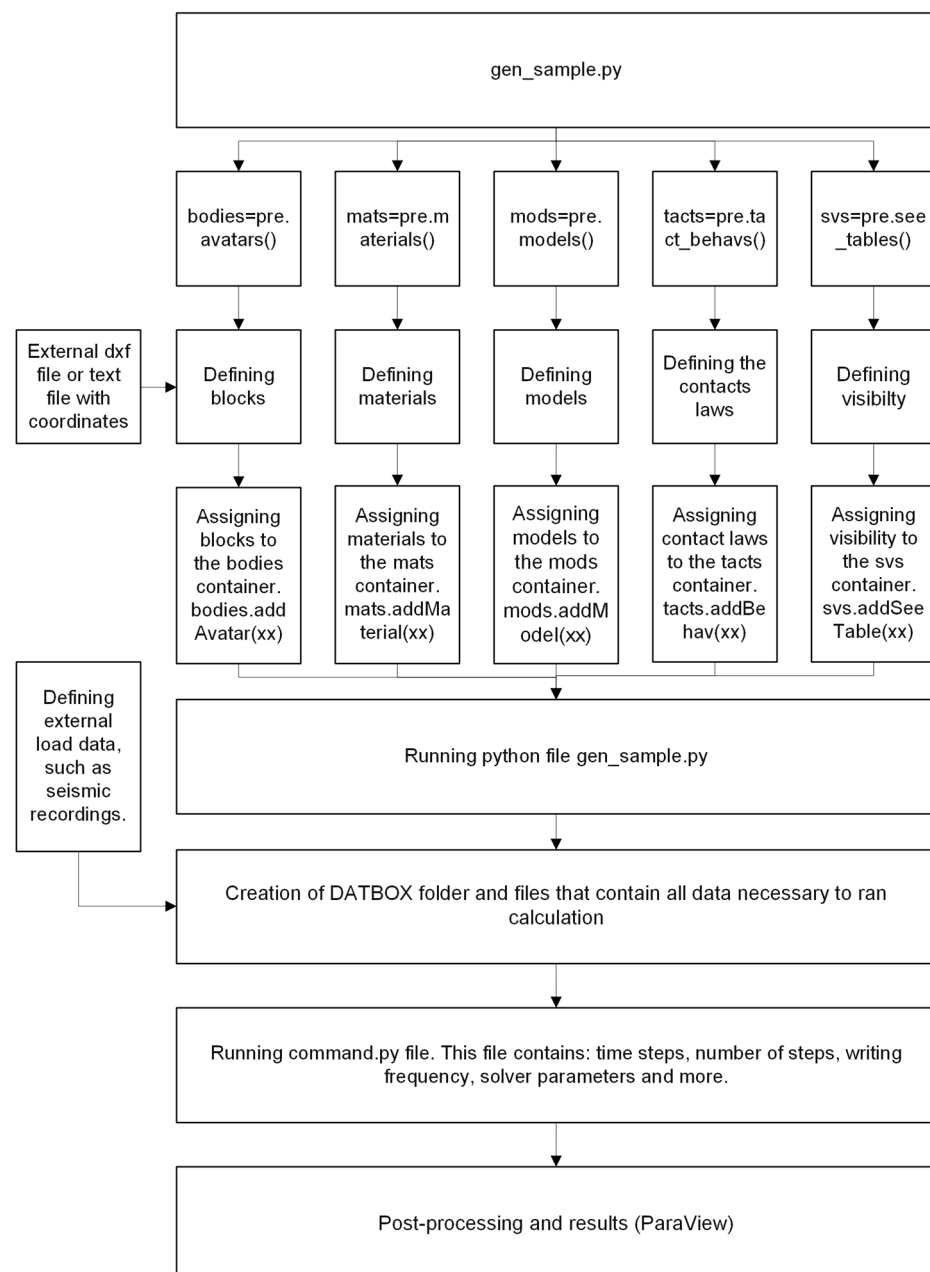


Figure 3. Flowchart of model creation.

3. Defining Numerical Model in LMGC90

Since the LMGC90 software lacks a graphical user interface, the following section presents the methodology for constructing numerical models [5]. Developing these models requires a basic understanding of the Python 3.14.1 programming language.

The objective is to model a two-dimensional exterior brick wall composed of solid bricks with standard dimensions of $25 \times 12 \times 6.5 \text{ cm}^3$. The wall includes four openings, and each lintel is constructed using an arched arrangement of solid bricks. A 1 cm joint is provided between bricks to represent the mortar layer. The overall wall dimensions are $5.97 \times 5.62 \text{ m}^2$, while each opening measures $1.05 \times 0.98 \text{ m}^2$ (Figure 4). The inner surface of the lintel follows a circular arc with a radius of 1 m.

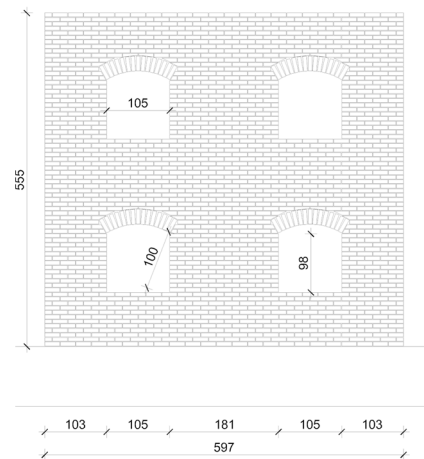


Figure 4. Wall geometry.

The interaction between solid bricks is modeled through friction. According to Equation 3.5 in [1], a friction coefficient of 0.4 is assumed, while the infill elements are considered perfectly rigid.

3.1. Initialization of the Numerical Model

To use the functions of the LMG90 program within Python [5], the pylmgc90 module must be imported using the command: `from pylmgc90 import pre`. It is also necessary to create the following containers:

1. `bodies=pre.avatars()`—list of characteristics of discrete elements;
2. `mats=pre.materials()`—list of all material properties;
3. `mods=pre.models()`—list of models (different numerical approaches can be used within the same model);
4. `svs=pre.see_tables()`—list of interaction rules between discrete elements;
5. `tacts=pre.tact_behavs()`—list of interaction laws.

3.2. Materials

Two types of materials were created: *brick* and *earth*:

1. `brick=pre.material(name='BRICK',materialType='RIGID',density=265.);`
2. `earth=pre.material(name='EARTH',materialType='RIGID',density=265.);`
3. `mats.addMaterial(brick,earth).`

By selecting the material, the behavior of each discrete element is defined. For the observed model, it was specified that the solid brick blocks would behave as rigid elements (`materialType='RIGID'`). The weight of the 12 cm thick brick wall is taken as 2.6 kN/m^2 [10], corresponding to a density of approximately 265 kg/m^3 (density = 265). All defined materials must be assigned to the container [2].

3.3. Model

Using the *model* module, the numerical problem to be solved is defined. It involves mechanics (`physics='MECAx'`) between rigid bodies (`element='Rxx2D'`) in two dimensions

(*dimension=dim*), where *dim* is defined as 2- or 3-dimensional space. After defining *model* characteristics, it is necessary to assign it to container *mods* with command *addModel(mod)*.

1. *mod=pre.model(name='rigid',physics='MECAx',element=Rxx2D',dimension=dim);*
2. *mods.addModel(mod)*.

3.4. Creation of Discrete Elements and Definition of Interaction

The geometry was created using AutoCAD 2025. Each solid brick block is defined as a separate polyline. It is important to note that using the mirror option in AutoCAD 2025 can cause problems when importing coordinates into LMGC90, and its use is therefore not recommended. By using the Python module *ezdxf*, it is possible to read coordinates from a *dxf* file and create a list of coordinates corresponding to the discrete elements. This approach significantly speeds up the modeling process, as LMGC90 does not have a graphical user interface.

Below is the code for reading the coordinates:

```
def extract_polyline_points(dxf_file):
    # Load the DXF document
    doc = ezdxf.readfile(dxf_file)
    # Get the modelspace
    msp = doc.modelspace()
    # List to store groups of polyline points
    polyline_groups = []
    # Iterate through all entities in modelspace
    for entity in msp:
        if entity.dxf.type() == 'LWPOLYLINE':
            # Extract points from the polyline
            points = entity.get_points()
            polyline_groups.append(points)
    return polyline_groups
# Example usage
dxf_file = 'wall.dxf'
polyline_groups = extract_polyline_points(dxf_file)
```

Each individual solid brick block (discrete element) is defined using the *rigidPolygon* function. The following parameters need to be specified:

1. *model=mod*—assigns the corresponding model to the discrete element;
2. *material=brick*—assigns the corresponding material to the discrete element;
3. *center=numpy.array([0.,0.])*—defines the center of mass relative to the origin of the coordinate system;
4. *color='REDxx'*—assigns a specific color to the element (the color name does not need to correspond to an actual color—it only needs to consist of five uppercase letters).

```
block=pre.rigidPolygon(model=mod,material=brick,center=numpy.array([0.,0.]),color='REDxx',generation_type='full', vertices=vertices)
```

The model must be assigned to each individual discrete element, since within the same numerical problem, discrete elements can be considered either rigid or elastic bodies. The command *generation type='full'* instructs the program to create a body with all associated parameters (geometry, mass, center of inertia, initial position, orientation, velocity) and specification indicating that it may come into contact with other bodies.

If the command *vertices* is used, it specifies that a list of coordinates is used to create polygonal discrete elements. As explained earlier, we can extract coordinates from a DXF file using the Python module *ezdxf*. The function *extract_polyline_points(dxf_file)* is created;

it extracts all coordinates for Polyline elements contained in the dxf file and records them in a list named `polyline_groups`. To create a brick's discrete element, the following code is used.

```
for i in range(0,len(polyline_groups)):
    vertices=numpy.zeros([4, 2], 'd')
    vertices[0, 0]=polyline_groups[i][0][0] # coordinates x1
    vertices[0, 1]=polyline_groups[i][0][1] # coordinates y1
    vertices[1, 0]=polyline_groups[i][1][0] # coordinates x2
    vertices[1, 1]=polyline_groups[i][1][1] # coordinates y2
    vertices[2, 0]=polyline_groups[i][2][0] # coordinates x3
    vertices[2, 1]=polyline_groups[i][2][1] # coordinates y3
    vertices[3, 0]=polyline_groups[i][3][0] # coordinates x4
    vertices[3, 1]=polyline_groups[i][3][1] # coordinates y4
    block = pre.rigidPolygon(model=mod, material=brick, center=numpy.array([0., 0.]),
    color='REDxx', generation_type='full',vertices=vertices)
    bodies.addAvatar(block)
```

The coordinates in the `polyline_groups` list are stored in the following way:

$$[((x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)), ((x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)), \dots]$$

In our case, a for loop reads the coordinates of each discrete element and adds them to a list called `vertices`, which has a structure suitable for LMGC90. Figure 5 shows one discrete element with assigned coordinates.

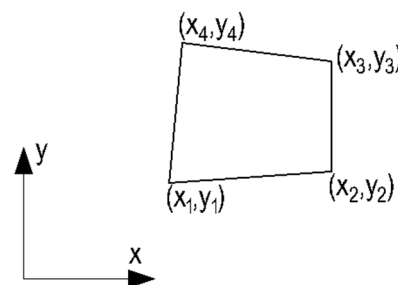


Figure 5. Description of a rigidPolygon with its associated coordinates.

Because the command `vertices` is used to create discrete elements, it is necessary to define the *center* of the discrete element with respect to the origin of the coordinate system. Then the *center* has to be set to `[0., 0.]`.

Another approach can be applied to extract coordinates. The list code (see Examples 3 and 4) written in AutoCAD can read element coordinates and record them in a txt file. Python code can be written to read data from a txt file and assign it to discrete elements in the model. This approach has been used to create discrete volume elements in examples 3 and 4.

```
i=0
while i< len(bodies_coords_1):
    vertices=numpy.zeros([8, 3], 'd')
    # - Point 1
    vertices[0, 0]=bodies_coords_1[i][0][0] # coordinates x1
    vertices[0, 1]=bodies_coords_1[i][0][1] # coordinates y1
    vertices[0, 2]=bodies_coords_1[i][0][2] # coordinates z1
    # - Point 2
```

```

vertices[1, 0]=bodies_coords_1[i][1][0] # coordinates x2
vertices[1, 1]=bodies_coords_1[i][1][1] # coordinates y2
vertices[1, 2]=bodies_coords_1[i][1][2] # coordinates z2
# - Point 3
vertices[2, 0]=bodies_coords_1[i][2][0] # coordinates x3
vertices[2, 1]=bodies_coords_1[i][2][1] # coordinates y3
vertices[2, 2]=bodies_coords_1[i][2][2] # coordinates z3
# - Point 4
vertices[3, 0]=bodies_coords_1[i][3][0] # coordinates x4
vertices[3, 1]=bodies_coords_1[i][3][1] # coordinates y4
vertices[3, 2]=bodies_coords_1[i][3][2] # coordinates z4
# - Point 5
vertices[4, 0]=bodies_coords_1[i][4][0] # coordinates x5
vertices[4, 1]=bodies_coords_1[i][4][1] # coordinates y5
vertices[4, 2]=bodies_coords_1[i][4][2] # coordinates z5
# - Point 6
vertices[5, 0]=bodies_coords_1[i][5][0] # coordinates x6
vertices[5, 1]=bodies_coords_1[i][5][1] # coordinates y6
vertices[5, 2]=bodies_coords_1[i][5][2] # coordinates z6
# - Point 7
vertices[6, 0]=bodies_coords_1[i][6][0] # coordinates x7
vertices[6, 1]=bodies_coords_1[i][6][1] # coordinates y7
vertices[6, 2]=bodies_coords_1[i][6][2] # coordinates z7
# - Point 8
vertices[7, 0]=bodies_coords_1[i][7][0] # coordinates x8
vertices[7, 1]=bodies_coords_1[i][7][1] # coordinates y8
vertices[7, 2]=bodies_coords_1[i][7][2] # coordinates z8
# * face connectivity
faces=numpy.zeros([12, 3], 'i')
faces[ 0, 0]=1; faces[ 0, 1]=2; faces[ 0, 2]=3 # face 1
faces[ 1, 0]=2; faces[ 1, 1]=5; faces[ 1, 2]=3 # face 2
faces[ 2, 0]=1; faces[ 2, 1]=2; faces[ 2, 2]=6 # face 3
faces[ 3, 0]=1; faces[ 3, 1]=6; faces[ 3, 2]=4 # face 4
faces[ 4, 0]=2; faces[ 4, 1]=5; faces[ 4, 2]=8 # face 5
faces[ 5, 0]=2; faces[ 5, 1]=8; faces[ 5, 2]=6 # face 6
faces[ 6, 0]=5; faces[ 6, 1]=3; faces[ 6, 2]=7 # face 7
faces[ 7, 0]=5; faces[ 7, 1]=7; faces[ 7, 2]=8 # face 8
faces[ 8, 0]=1; faces[ 8, 1]=3; faces[ 8, 2]=7 # face 9
faces[ 9, 0]=1; faces[ 9, 1]=7; faces[ 9, 2]=4 # face 10
faces[10, 0]=4; faces[10, 1]=6; faces[10, 2]=7 # face 11
faces[11, 0]=6; faces[11, 1]=8; faces[11, 2]=7 # face 12
# creation of a new rigid avatar for the block
block = pre.rigidPolyhedron(model=mR3D, material=stone, center=numpy.array([0., 0., 0.]),
generation_type='full', vertices=vertices, faces=faces, color='REDxx')
# adding the block to the set of bodies
bodies.addAvatar(block)
i=i+1

```

The codes suggest that the process of defining vertices is similar in 2D and 3D problems. The only differences are more nodes and a third coordinate in 3D problems. The major

difference is that faces describe 3D elements. The faces are triangular surfaces defined by three nodes, as shown in Figure 6a,b.

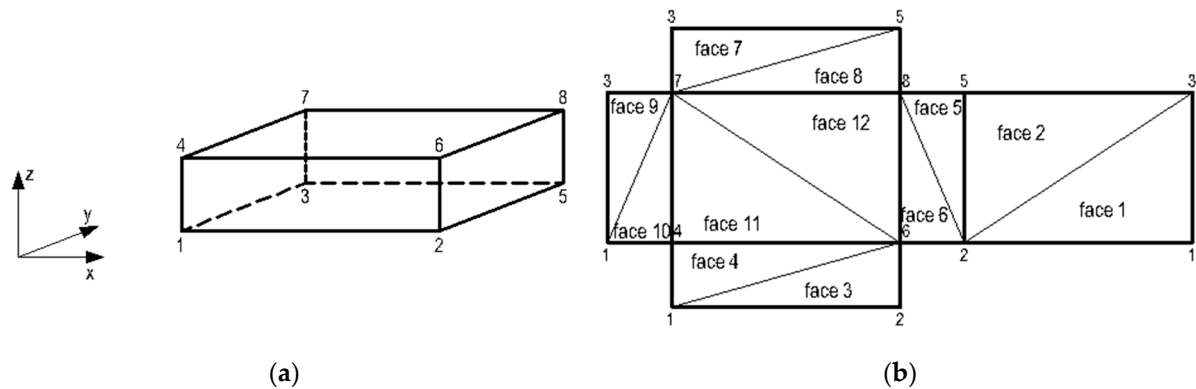


Figure 6. Discrete element in the shape of a rectangular prism: (a) associated nodes; (b) associated nodes and faces.

The LMGC90 library contains predefined discrete elements. The foundations in examples 1 and 2 are models with command `pre.rigidJonc`.

```
pre.rigidJonc(model=mod, material=earth, center=numpy.array([1., -0.25]), color='GREEN',
axe1=1.0, axe2=0.25)
```

The values `axe1` and `axe2` describe the dimensions of discrete elements, while command `center=numpy.array([1., -0.25])` defines the position as shown in Figure 7.

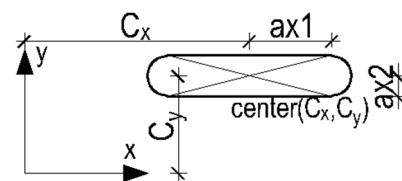


Figure 7. Shape and position of rigidJonc discrete element.

It is important to assign colors correctly to the elements, as this provides control over interactions between discrete elements, i.e., the possibility of contact between them. It is essential to add each block element to the *bodies* list using the command `bodies.addAvatar(block)`. If they are not assigned to a list, the software will not include them in the calculation.

Two records, `svplpl` and `svpljc`, which define how discrete elements interact, are presented [5,8].

```
svplpl=pre.see_table(CorpsCandidat='RBDY2', candidat='POLYG', colorCandi-
dat='REDxx', CorpsAntagonist='RBDY2', antagonist='POLYG', colorAntagonist='REDxx', be-
hav='iqsc0', alert=0.03)
```

```
svpljc=pre.see_table(CorpsCandidat='RBDY2', candidat='POLYG', colorCandi-
dat='REDxx', CorpsAntagonist='RBDY2', antagonist='JONCx', colorAntagonist='GREEN', be-
hav='iqsc0', alert=0.03)
```

The `svplpl` law defines the interaction between individual solid brick blocks, while the `svpljc` law defines the interaction between solid brick blocks and the base. The `svpljc` law specifies that the candidate, which is a 2D rigid body (RBDY2) of polygonal shape (POLYG) and color REDxx, can come into contact with an antagonist, which is also a 2D rigid body (RBDY2), polygonal (POLYG), and of color REDxx. The `behav` command specifies the interaction law, while the `alert` command issues a warning whenever initial overlap or small gap (below a defined threshold) between discrete elements is detected.

3.5. Interaction Law

The contact force between two discrete elements is determined by the interaction law. Masonry structures made of stone or solid brick represent an ideal case for the application of the discrete element method [2–4]. Such structures can be built with or without mortar (or the mortar may be of very low quality). Depending on the condition of the structure being analyzed, the appropriate interaction laws are selected. The parameters that can describe the interaction between two solid wall blocks include dynamic and static friction, as well as cohesion both along the joint and perpendicular to the joint [9]. *iqsc0=pre.tact_behav(name='iqsc0',law='IQS_CLB_g0',fric=0.4)iqsmc=pre.tact_behav(name='iqsmc',law='IQS_MOHR_DS_CLB',cohn=0,coht=30000,dyfr=0.4,stfr=0.4)*

IQS_CLB_g0 is a contact law that describes the inelastic impact law (*inelastic quasi-shock law*—hereafter referred to as IQS) and is based on Coulomb's friction law (hereafter referred to as CLB). Symbol *g0* describes the initial gap between discrete elements. If the tangential force exceeds the friction between the elements, the gap can close. Advantages of using gap include its ability to simulate mortar in masonry structures and its potential to simplify the process of defining geometry. According to the LMGC90 documentation, the IQS law is suitable for simulating problems involving large deformations of rigid bodies; however, the friction coefficient needs to be specified [1,2].

IQS_MOHR_DS_CLB describes the Mohr–Coulomb law. In addition to friction, this law also assumes the presence of cohesion. In the first step, the interaction between discrete elements is defined and cohesion between them is established. At the moment when the forces exceed the cohesion, further interaction occurs exclusively through friction, that is, the *IQS_CLB* begins to be applied [1].

3.6. Numerical Examples

In the Modeling section, the methodology for developing a numerical problem using the discrete element approach with the LMGC90 software package is presented. The following section presents four numerical examples. In the first two numerical examples, the behavior of solid brick masonry in-plane is analyzed under the effects of earthquake loading and differential foundation settlement. The remaining two numerical examples illustrate the out-of-plane behavior of solid brick masonry walls, considering cases where the wall is restrained and unrestrained. A gap of constant thickness, simulating the mortar layer, is defined between the bricks.

The El Centro NS earthquake record was used as the excitation for all numerical examples. The acceleration–time record (Figure 8) was converted into a velocity–time record (Figure 9). The conversion was carried out using trapezoidal numerical integration. In the numerical models, the excitation was defined in the form of the velocity–time record. On the foundation discrete element, boundary conditions are defined. The vertical displacement and rotation are constrained. The horizontal displacement follows the velocity–time diagram.

```
down.imposeDrivenDof(component=[ 2, 3], dofty='vlocy');
component=[ 2, 3]—defines vertical displacements and rotation equal to zero;
dofly='vlocy'—type of boundary condition (velocity);
down.imposeDrivenDof(component=[1],dofly='vlocy',description='evolution',evolution-
File='evolution.dat');
description='evolution'—instructs the software that an external data source will be
used for the boundary condition;
evolutionFile='evolution.dat'—loads the record; velocity–time history obtained by
numerical integration of the El Centro NS earthquake record.
```

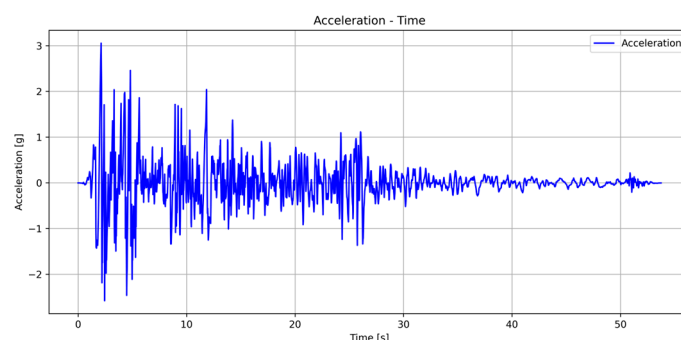


Figure 8. Excitation: Acceleration–Time for the El Centro NS Earthquake Record.

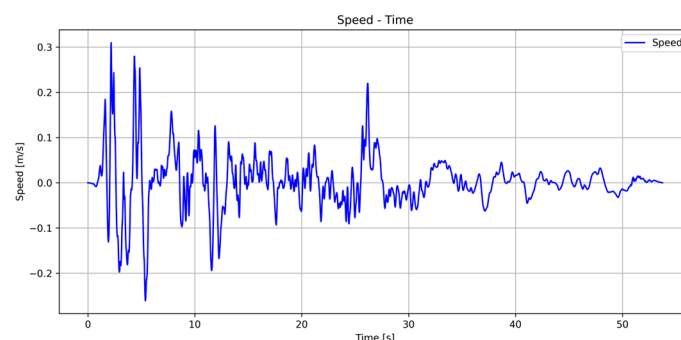


Figure 9. Excitation: Velocity–Time for the El Centro NS Earthquake Record.

The contact properties were adopted from the existing literature (Table 1).

Table 1. Table of strength values as specified in UK National Annex to Eurocode 6: Design of masonry structures [11].

Masonry Units	Strength Class of General Purpose Mortar	f_{vko} (N/mm ²)		
		General Purpose Mortar	Thin Layer Mortar (Bed Joint ≤ 0.5 mm and ≥ 3 mm)	Lightweight Mortar
Clay	M12	0.30	0.40	0.15
	M4 and M6	0.20		
	M6	0.10		
	M2			
Calcium silicate	M12	0.20	0.40	0.15
	M4 and M6	0.15		
	M6	0.10		
	M2			
Aggregate concrete, autoclaved aerated concrete, manufactured stone and dimensioned natural stone	M12	0.20	0.30	0.15
	M4 and M6	0.15		
	M6	0.10		
	M2			

4. Results

4.1. Numerical Example 1—In-Plane Wall Behavior Under Foundation Settlement [12]

The case shown in Figure 10 is considered. The assumed friction coefficient is 0.4 [13], while cohesion is neglected in this example. The simulation predicts settlement of the foundation on the right side of the structure (approximately one-third of the total building length). The settlement gradually increases until it reaches a final value of 5 cm (Figure 10e,f).

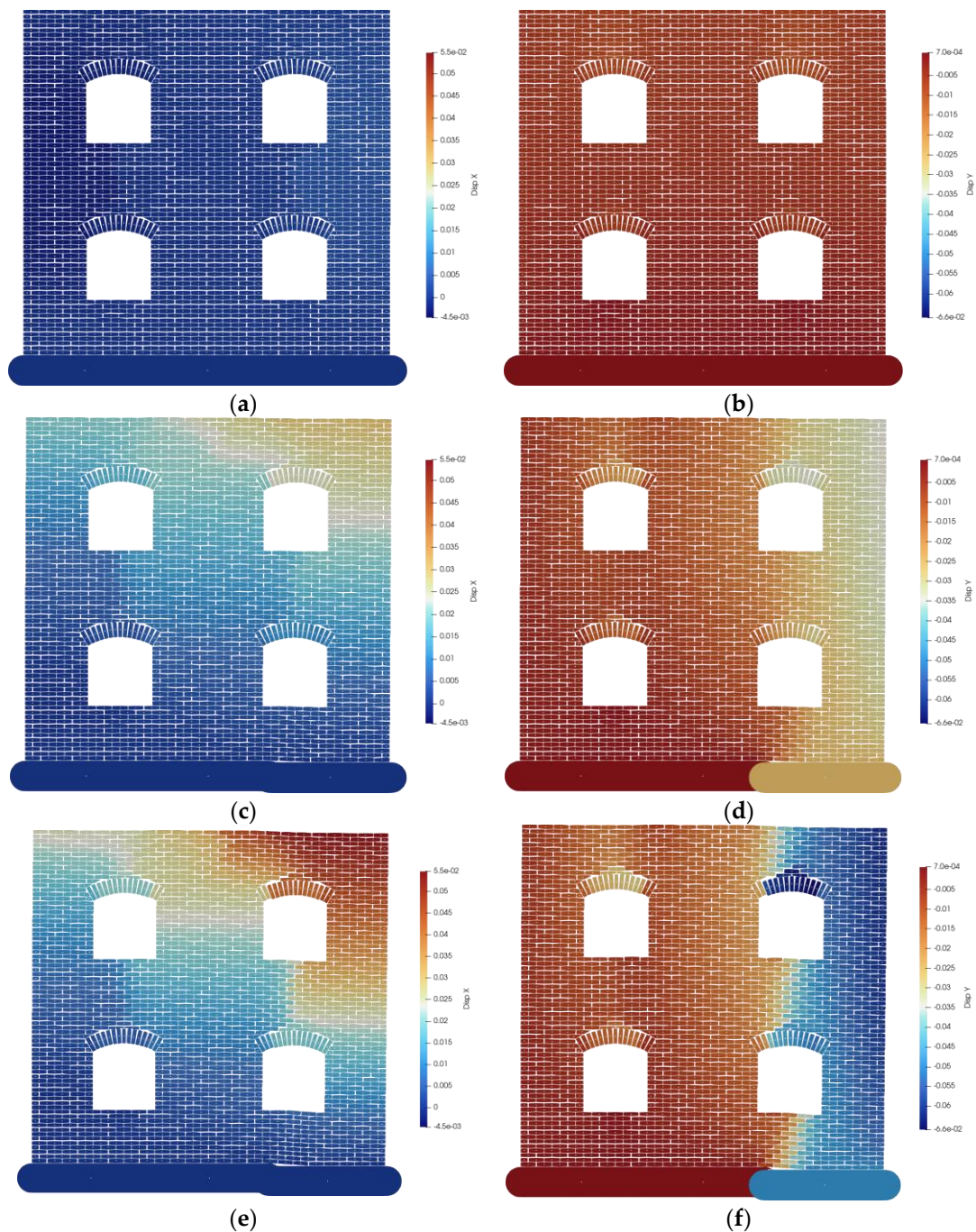


Figure 10. Horizontal and vertical displacement: (a) horizontal displacement—settlement 0 cm; (b) vertical displacement—settlement 0 cm; (c) horizontal displacement—settlement 2.5 cm; (d) vertical displacement—settlement 2.5 cm; (e) horizontal displacement—settlement 5 cm; (f) vertical displacement—settlement 5 cm.

The calculation enables tracking of crack development within the structure. It can be observed that cracks gradually increase in the parapet zones. It can be concluded that at a settlement of 5 cm, failure of the lintel occurs. Due to vertical settlements, horizontal displacements also appear in the structure. The calculation indicates a maximum horizontal displacement of 5.5 cm at the top of the second floor (Figure 10e). The maximum vertical displacements are observed in the lintel zones, and Figure 10f shows that lintel failure has occurred. The lintel has separated from the wall, and its vertical displacements progress

faster than the ground settlement. The maximum vertical displacement of the lintel is 6.6 cm, while the other vertical displacements follow the ground settlement.

Figure 11a defines the location of two control points. Point 523 (blue) is located on the top of the first floor, while point 1567 (red) is on the top of the second floor. Figure 11b shows the connection between horizontal displacement and foundation settlement. The location of foundation segments is shown in Figure 12a: from left to right—segment 1, segment 0, segment 2. Figure 12b shows the reaction/foundation settlement diagram.

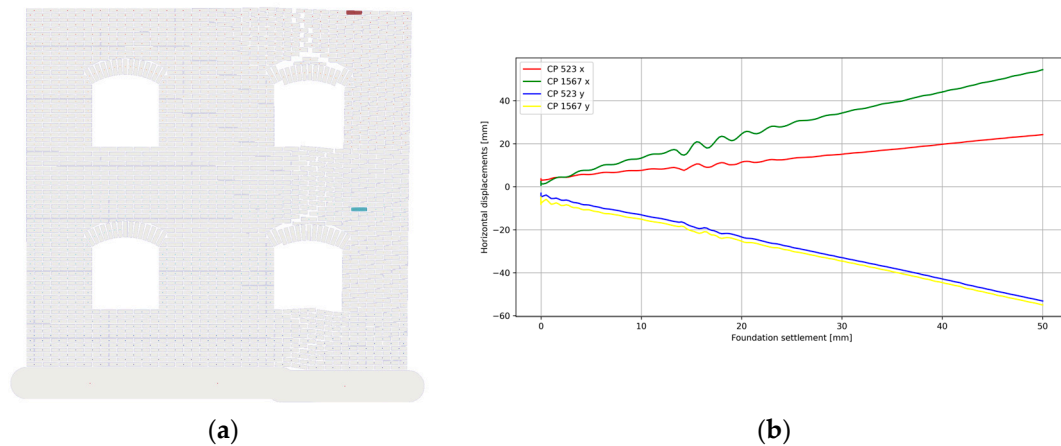


Figure 11. (a) Location of control points; (b) horizontal displacement/foundation settlement diagram.

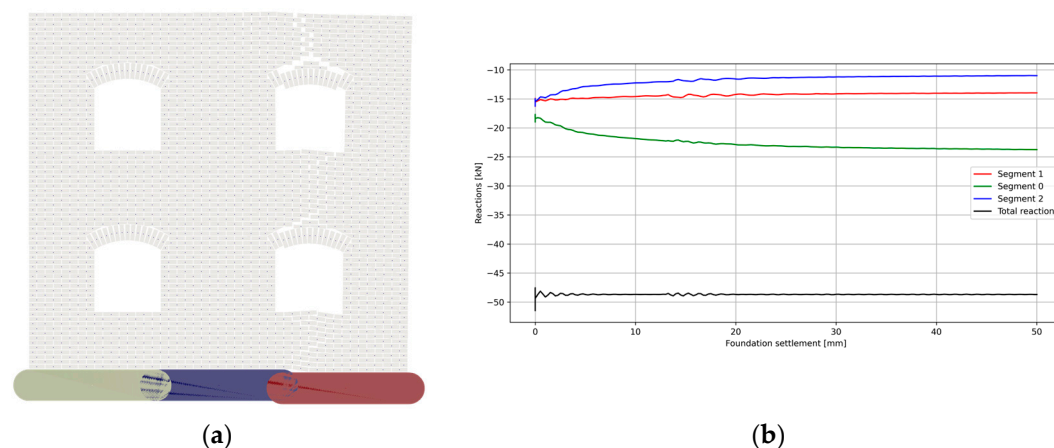


Figure 12. (a) Location of foundation segments; (b) reaction/foundation settlement diagram.

4.2. Numerical Example 2—In-Plane Wall Behavior Under Dynamic Excitation [14]

In Example 2, the same geometry and physical parameters are considered as in Example 1. The difference between the two examples lies in the applied excitation. Figure 9 shows the time–velocity record that was used as the ground excitation. In the numerical analysis, 5600 steps with time intervals of 0.02 s were analyzed, and the results are presented after every 10 steps. From this, it can be concluded that the first 20 s of the earthquake record were observed, covering the majority of the maximum accelerations in the earthquake record.

By examining Figure 13, it can be observed that at these moments, failure occurs in the lintels above the second-floor windows. For the right window, the left support fails, while for the left window, the right support fails. The moment of lintel failure occurs at 2.1 and 2.2 s. Figure 14 shows that the maximum acceleration occurs at these moments. At 4.3 and 4.5 s, the support of the lintel fails on the opposite sides as well.

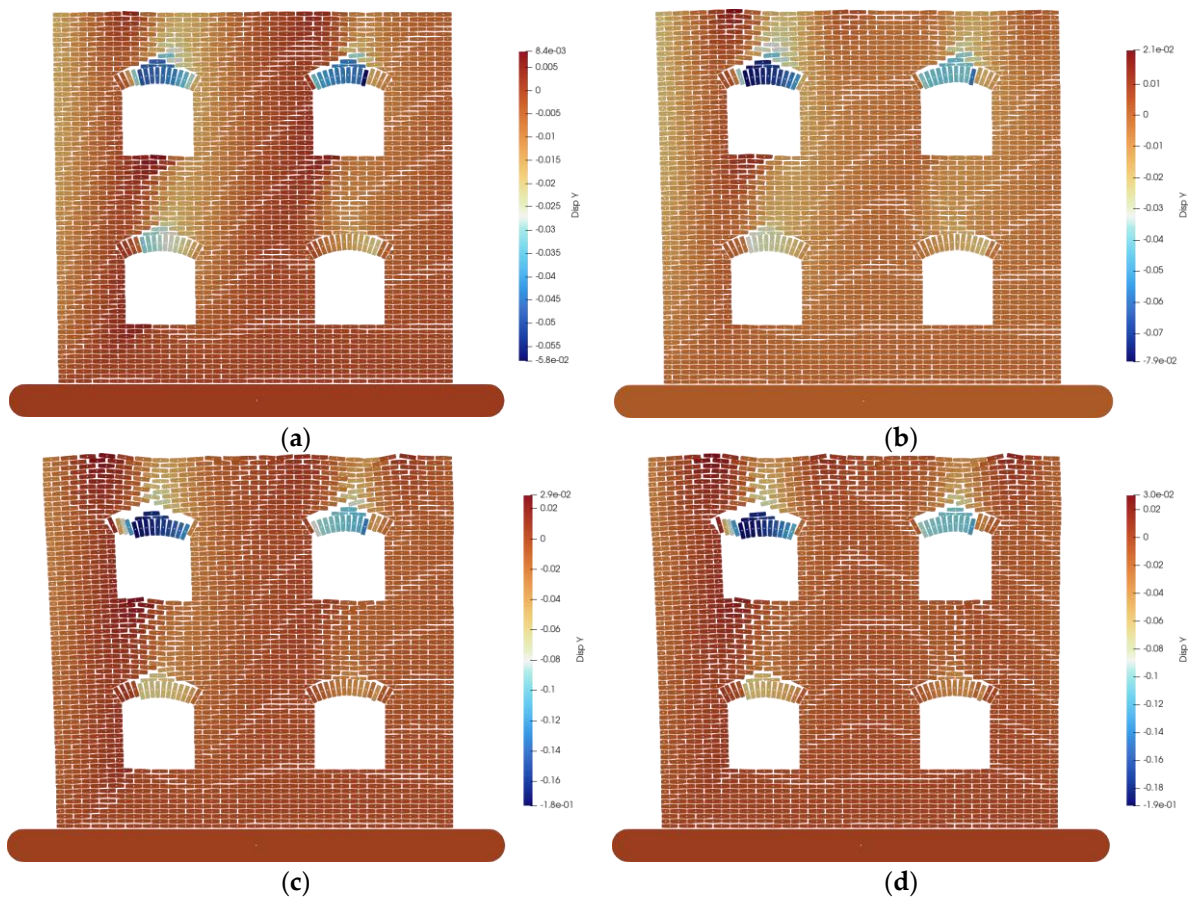


Figure 13. Vertical displacement: (a) vertical displacement—2.1 s; (b) vertical displacement—2.2 s; (c) vertical displacement—4.3 s; (d) vertical displacement—4.5 s.

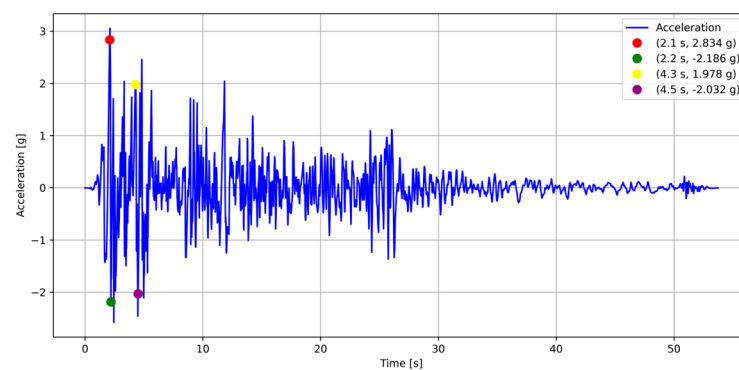


Figure 14. Earthquake record with indicated critical moments of structural failure.

Figure 15 presents the positions of the discrete elements (bricks), with horizontal displacements displayed. The figure shows that a relative displacement has occurred between the left and right sides, meaning they have moved apart, which results in an increase in the gap (crack) between the bricks.

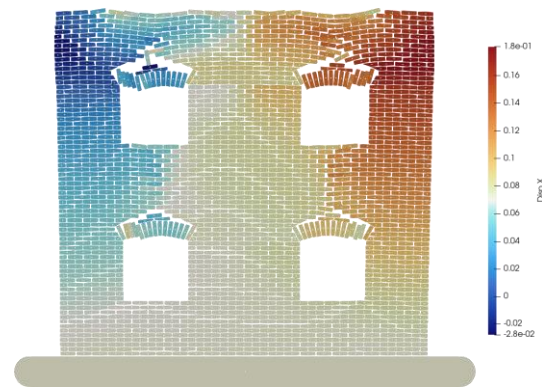


Figure 15. Horizontal displacement—20 s.

Figure 16a defines the location of two control points. Point 552 (blue) is located on the top of the first floor, while point 1557 (red) is on the top of the second floor. Figure 16b represents the connection between horizontal displacement and foundation friction force.

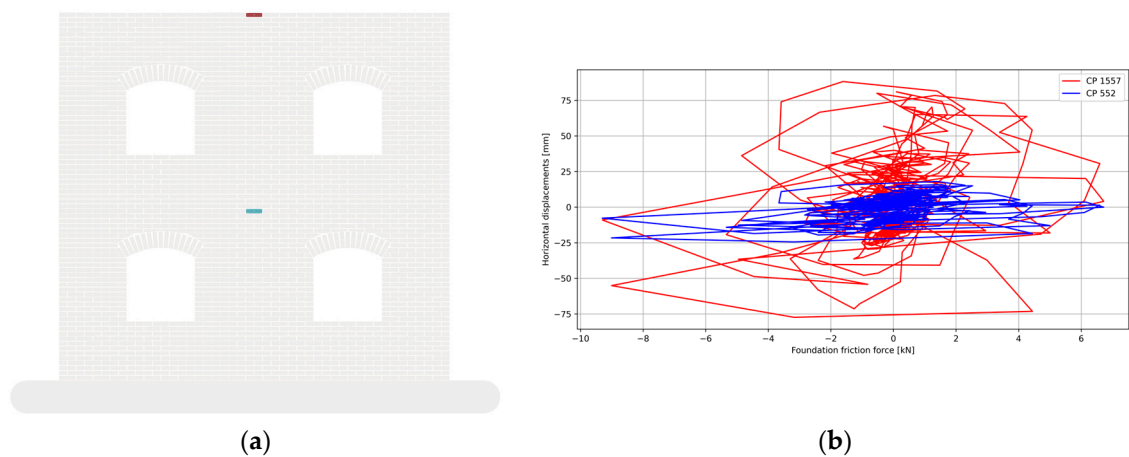


Figure 16. (a) Location of control points; (b) horizontal displacement/foundation friction force.

4.3. Numerical Example 3—Out-of-Plane Stability of the Wall (Unrestrained/Free-Standing Condition) [15]

A three-dimensional wall stability problem is considered. The dimensions of the wall are $4.375 \times 2.34 \times 0.25 \text{ m}^3$, consisting of discrete elements measuring $6.5 \times 12 \times 25 \text{ cm}^3$, corresponding to the size of a standard brick (Figure 17). Two parameters are considered on the contact surfaces: friction (static and dynamic) and cohesion. The values of static and dynamic friction are chosen as 0.4 [13], while the cohesion value is set to 0.3 N/mm^2 [11]. The tensile properties of the mortar were neglected [2–5,8,9].

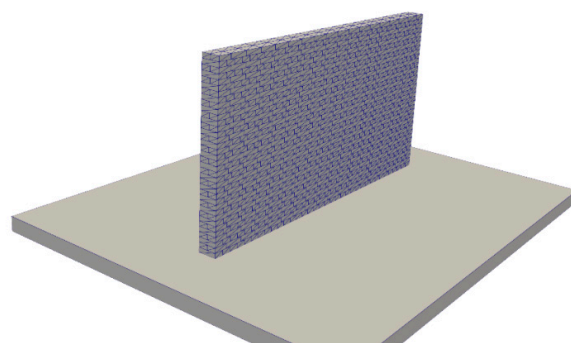


Figure 17. A three-dimensional representation of the brick wall.

The results show that the applied dynamic excitation on the wall causes its overturning. The overturning occurs at 4.5 s, which corresponds to an acceleration of 0.207 g according to the earthquake record. The overturning did not occur at 2.12 s, when the maximum acceleration of 0.305 g was recorded, but rather as a result of rocking and the second peak acceleration in the earthquake record as shown in Figures 18–20.

The calculation was performed with a time step of 0.003 s, with a total of 4000 steps. The results were recorded every 50 steps.

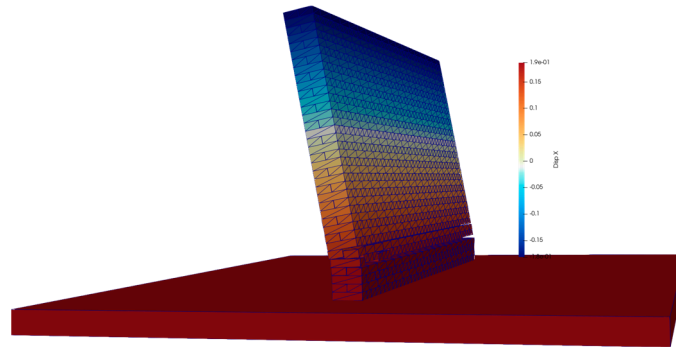


Figure 18. Wall failure by overturning—4.5 s.

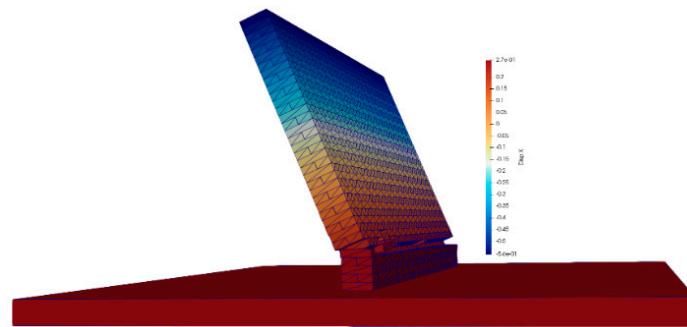


Figure 19. Wall failure by overturning—4.95 s.

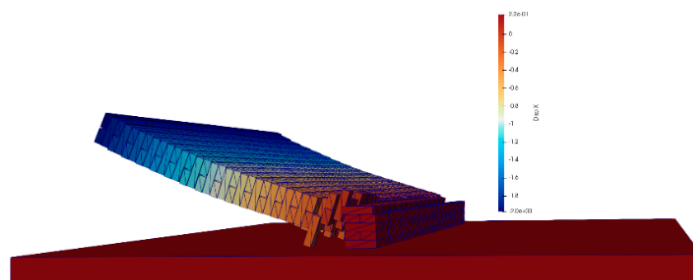


Figure 20. Wall failure by overturning—5.5 s.

4.4. Numerical Example 4—Out-of-Plane Stability of the Wall (Restrained) [16]

The analysis of the restrained condition was carried out using the same parameters as in Numerical Example 3. The geometric and material properties were also kept identical. The difference between the two models is that in Numerical Example 4, the wall is laterally restrained at its edges. It is assumed that lateral walls are present at the edges of the observed wall and move together with the base. The same kinematic parameters as for the base (ground) were applied to the two rows of bricks at both edges of the wall.

The results indicate that a complete failure of the wall, as observed in Numerical Example 3, did not occur. Due to the excitation, cohesion between the bricks was partially lost. Figure 21 shows the displacements at the base and the middle of the top of the wall. The diagram reveals that a relative deformation occurs between the two observed

points. The maximum relative displacement of 56 mm occurs at 31.67 s. Although the wall did not experience complete collapse, permanent deformations are expected—a final relative displacement of approximately 13 mm is anticipated between the observed points (Figure 22) [3,4,8].

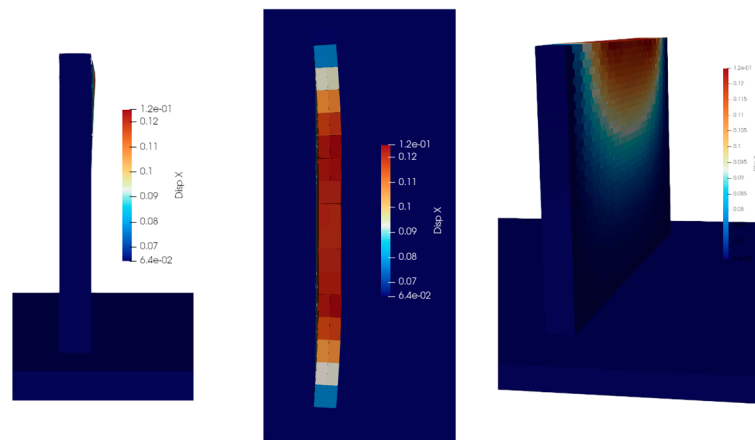


Figure 21. Wall displacement at 13.67 s.

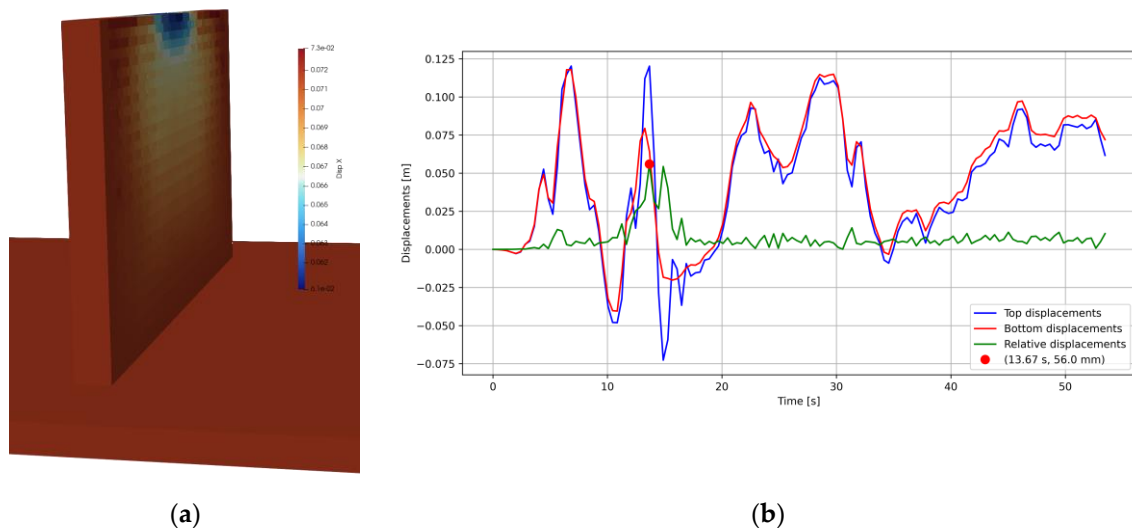


Figure 22. (a) Wall displacement at 53.4 s; (b) displacement–time curve for the top and base of the wall and their relative displacement.

5. Conclusions

The application of discrete modeling is not a standard procedure in engineering practice. The discrete approach allows insight into the behavior of structures that is difficult to describe using numerical methods that require a continuous domain. The behavior of structures built from solid bricks can be well represented using the discrete method, where each brick is modeled as a separate discrete element.

This study provides a brief overview of the theoretical basis of the Non-Smooth Contact Dynamics (NSCD) method, which underlies the LMGC90 software package. The open-source philosophy was applied in the development of LMGC90, making it widely accessible. The creation of the numerical model is carried out using Python scripts. The procedure for defining the model is presented, along with a brief explanation of the commands used and a description of the parameters that define the numerical model. The process of defining the geometry in AutoCAD 2025 is also described, significantly facilitating the geometry definition process [5].

In the four numerical examples, some of the numerical capabilities that are difficult to describe using standard structural modeling software are presented. The first and second examples illustrate lintel failure and separation of the structure due to differential foundation settlement and seismic excitation, respectively. The third and fourth examples present a three-dimensional wall stability problem out-of-plane due to seismic loading. In the third example, complete wall collapse occurs, while in the fourth example, the failure of the connections between solid brick blocks results in permanent deformations. All these phenomena highlight the importance of applying a discrete approach in structural modeling [3,4,8,9].

Future research could focus on combining the four examples into a single model, which would provide an even more realistic structural response. It would also be important to include a comparison of the model behavior with experimental results, which would further validate the obtained results and enable a better selection of contact parameters between the discrete elements.

Author Contributions: Conceptualization, D.G. and Š.S.; methodology, all authors; software, D.G.; validation, all authors; formal analysis, all authors; investigation, all authors; resources, all authors; data curation, all authors; writing—original draft preparation, D.G. and Š.S.; writing—review and editing, N.Š., Š.S. and Č.D.; visualization, all authors; supervision, all authors; project administration, all authors; funding acquisition, all authors. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available upon request from the corresponding author. The data are not publicly available due to privacy reasons.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

NSCD Non-Smooth Contact Dynamics
CNRS French National Center for Scientific Research

References

1. Dubois, F.; Mathieu, R. *Discrete Element Methods for the Simulation of Divided Media*; Laboratoire de Mécanique et Génie Civil, Université de Montpellier–CNRS: Montpellier, France, 2015. Available online: https://git-xen.lmgc.univ-montp2.fr/lmgc90/lmgc90_user/-/blob/98dd03acce321a7cf5fe2e8464d30d772aa71d90/manuals/LMGC90_Theory.pdf (accessed on 25 May 2025).
2. Sánchez, P.; Renouf, M.; Azéma, E.; Mozul, R. LMGC90: A contact dynamics open source code for the simulation of granular asteroid with realistic regolith shapes: Application to the accretion process. *EPJ Web Conf.* **2021**, *249*, 14007. [CrossRef]
3. Rafiee, A.; Vinches, M.; Bohatier, C. Modelling and analysis of the Nîmes arena and the Arles aqueduct subjected to seismic loading using the Non-Smooth Contact Dynamics method. *Eng. Struct.* **2008**, *30*, 3457–3467. [CrossRef]
4. Schiavoni, M.; Giordano, E.; Roscini, F.; Clementi, F. Advanced numerical insights for an effective seismic assessment of historical masonry aggregates. *Eng. Struct.* **2023**, *285*, 115997. [CrossRef]
5. Dubois, F.; Mozul, R. LMGC90. In Proceedings of the 11ème Colloque National en Calcul des Structures, Giens, France, 13–17 May 2013.
6. Nodargi, N.A.; Bisegna, P. A variational-based non-smooth contact dynamics approach for the seismic analysis of historical masonry structures. *Comput. Methods Appl. Mech. Eng.* **2024**, *432*, 117346. [CrossRef]
7. Zhang, Y.; Li, X.; He, R.; Tang, M.; Wu, Z. Data-Driven Machine-Learning-Based Seismic Response Prediction and Damage Classification for an Unreinforced Masonry Building. *Appl. Sci.* **2025**, *15*, 1686. [CrossRef]
8. Zhou, Z.; Andreini, M.; Sironi, L.; Lestuzzi, P.; Andò, E.; Dubois, F.; Bolognini, D.; Dacarro, F.; Andrade, J.E. Discrete structural systems modeling: Benchmarking of LS-DEM and LMGC90 with seismic experiments. *J. Eng. Mech.* **2023**, *149*, 04023097. [CrossRef]

9. Pulatsu, B.; Bretas, E.M.; Lourenço, P.B. Discrete element modelling of masonry structures: Validation and application. *Earthq. Struct.* **2016**, *11*, 563–582. [[CrossRef](#)]
10. Krapfenbauer, R.; Krapfenbauer, T. *Građevinske Tablice*; Sajam d.o.o.: Zagreb, Croatia, 2006.
11. British Standards Institution (BSI). *UK National Annex to BS EN 1996-1-1:2005: Eurocode 6: Design of Masonry Structures—Part 1-1: General Rules for Reinforced and Unreinforced Masonry Structures*; BSI: London, UK, 2007.
12. LMGC90. Example_1. GitHub Repository. Available online: https://github.com/daliborgelo/LMGC90/tree/main/Example_1 (accessed on 20 February 2026).
13. HZN. *HRN EN 1996: Eurokod 6: Projektiranje Zidanih Konstrukcija*; Hrvatski Zavod za Normiranje: Zagreb, Croatia, 2011.
14. LMGC90. Example_2. GitHub Repository. Available online: https://github.com/daliborgelo/LMGC90/tree/main/Example_2 (accessed on 20 February 2026).
15. LMGC90. Example_3. GitHub Repository. Available online: https://github.com/daliborgelo/LMGC90/tree/main/Example_3 (accessed on 20 February 2026).
16. LMGC90. Example_4. GitHub Repository. Available online: https://github.com/daliborgelo/LMGC90/tree/main/Example_4 (accessed on 20 February 2026).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.