PercoNet

Release 0.2.3

Chiara Raffaelli, Wouter G. Ellenbroek

Jan 31, 2024

GENERAL USAGE

CHAPTER

ONE

INSTALLING PERCONET

The easiest way to install **perconet** is using *pip*, which downloads the code from PyPI:

```
pip install perconet
```

CHAPTER

TWO

PACKAGE CONTENTS

Being a small package, **perconet** exposes only two classes.

2.1 PeriodicNetwork

class `perconet.`**`PeriodicNetwork`**(*n: int*, *max_degree=6*, *verbose=False*, *dim=3*)

Store and analyze the topology of a periodic net.

Periodic nets are graphs embedded in a periodic topology. This class stores the topology of such a graph for the case of a d-dimensional periodic box (a d-torus). The dimensionaly defaults to 3 for use in contexts where the box represents physical space, but the *`PeriodicNetwork`* and *`LoopFinder`* classes work for arbitrary dimension.

The class stores, for every edge in the graph, a d-dimensional vector indicating the boundary-wrapping properties of that edge. See *`PeriodicNetwork.add_edge`* for details. This information is then used by *`LoopFinder`* to determine the percolation properties.

Parameters

- **`n`** (*`int`*) – The number of nodes of the graph.
- **`max_degree`** (*`int`*) – The largest number of edges coming out of any node.
- **`verbose`** (*`bool, optional`*) – Print debugging information to stdout. Defaults to False.
- **`dim`** (*`int, optional`*) – Spatial dimension. Defaults to 3.

`add_edge`(*node1: int*, *node2: int*, *boundary_vector*)

Add an edge to the periodic network

Parameters

- **`node1`** (*`int`*) – The index of the first node of the pair that defines this edge. Valid values range from 0 up to (but not including) the number of nodes of the network (node indices are 0-based).
- **`node2`** (*`int`*) – The index of the second node of the pair that defines this edge. Valid values range from 0 up to (but not including) the number of nodes of the network (node indices are 0-based).
- **`boundary_vector`** (`List` of int) – List (or numpy array) of integers denoting the number of times the edge wraps around each boundary, respectively. The length of this list must be equal to the dimensionality of the network (which defaults to 3 but can be overridden during initialization). The sign indicates the wrapping direction (e.g. (-1,0,0) indicates that the edge goes around the x-boundary in the negative x-direction when going from node1 to node2.

Returns
True if succesful. False if an error occurred.

Return type
(bool)

`crosses_boundaries()`

Check if the network contains any edges that cross a boundary.

Returns
True if the network has any edges that cross a boundary.

Return type
bool

`decompose`(*internal_only=True*)

Obtain the cluster decomposition of the network. This method is used by *LoopFinder* (using internal bonds only) to reduce the network for faster loop finding, but can also be used for generic cluster analysis.

Parameters
internal_only (*bool, optional*) – Defaults to True. If true, use only bonds that do not cross any boundary for the cluster decomposition.

Returns
A list with the cluster ID of each node and the number of clusters

Return type
Tuple[`List` of int, int]

`get_boundary_crossing`(*node*, *nb_index*)

Get the boundary crossing vector of the nb_index'th neighbor of node.

Parameters

- **node** (*int*) – node number
- **nb_index** (*int*) – index of neighbor in neighbor list of node

Returns
The list of integers denoting the number of times each boundary is crossed by this edge. Provided as a numpy array with length equal to the dimensionality of the network and dtype=int.

Return type
`numpy.ndarray`

`get_edge`(*node*, *nb_index*)

Get the edge number of the nb_index'th edge of node.

Parameters

- **node** (*int*) – node number
- **nb_index** (*int*) – index of neighbor in neighbor list of node

Returns
The edge number of that edge (to be used as an index in arrays of edge properties). A return value of -1 indicates that the edge does not exist.

Return type
int

`get_edges`(*node*, *padded=True*)

Get the list of edges linking to node.

Parameters

- **node** (*int*) – The index of the node for which to return the edge list
- **padded** (*bool, optional*) – If true (the default), the list will be padded with values -1 to the value of maximum_neighbors_per_node passed to the constructor. Otherwise the length will be the number of neighbors of node.

Returns
Numpy array (dtype=int) containing the edge numbers of all edges involving node.

Return type
`numpy.ndarray`

`get_neighbor`(*node*, *nb_index*)

Get nb_index'th neighbor of node.

Parameters

- **node** (*int*) – node number
- **nb_index** (*int*) – index of neighbor in neighbor list of node

Returns
The index of that neighbor (the value of *get_neighbors(i)[nb_index])*. A return value of -1 indicates that the edge does not exist.

Return type
int

`get_neighbors`(*node*, *padded=True*)

Get array of neighbor indices of node.

Parameters

- **node** (*int*) – node number
- **padded** (*bool, optional*) – If true (the default), the list will be padded with values -1 to the value of maximum_neighbors_per_node passed to the constructor. Otherwise the length will be the number of neighbors of i.

Returns
Numpy array (dtype=int) containing list of neighbors of node.

Return type
`numpy.ndarray`

`get_number_of_edges`()

Get total number of edges in network.

Returns
Total number of edges (bonds) in the network

Return type
int

`get_number_of_neighbors`(*node*)

Get the number of bonds of node.

Parameters
node (*int*) – node number

Returns
The number of edges (bonds) involving this node

Return type
int

`get_reduced_network()`

Generate the reduced network with identical boundary crossing properties but no internal edges.

Returns
The reduced network

Return type
`PeriodicNetwork`

`needs_reducing()`

Determine if the network could be reduced using internal connected component decomposition.

LoopFinder will perform this reduction automatically so there will not usually be a need for the user to call this function themselves.

Returns
True if the network has any edges that do not cross any boundary.

Return type
bool

2.2 LoopFinder

`class perconet.LoopFinder`(*network*, *verbose=True*)

Class implementing a depth-first search to determine the percolation directions of the network.

Parameters

- **`network`** (*`perconet.PeriodicNetwork`*) – A PeriodicNetwork object representing the graph to analyze.
- **`verbose`** (*`bool, optional`*) – Generate verbose output to stdout (to be replaced by Logging in future release)

`get_independent_loops()`

Generate a list of all linearly independent topologically nontrivial loops.

The list is returned in Hermite normal form. See *Loop independence* for details.

Returns
(list, int) A tuple containing a list of the independent loops and the length of that list. Each element of the list of loops is itself a list of the number of times each boundary is crossed by that loop.

Return type
Tuple[`List` of `List` of int, int]

`get_loops()`

Generate a raw list of boundary-crossing loops. Most use cases will require *`get_independent_loops()`* instead.

If the network contains any internal bonds, this routine performs a cluster reduction of the network before it starts, but this does not alter the *`PeriodicNetwork`* object that was used to construct this *`LoopFinder`* instance. If the reduced network is needed elsewhere, use *`PeriodicNetwork.get_reduced_network()`*.

Returns

(list, int) A tuple containing a list of the raw loops and the length of that list. Each element of the list of loops is itself a list of the number of times each boundary is crossed by that loop.

Return type

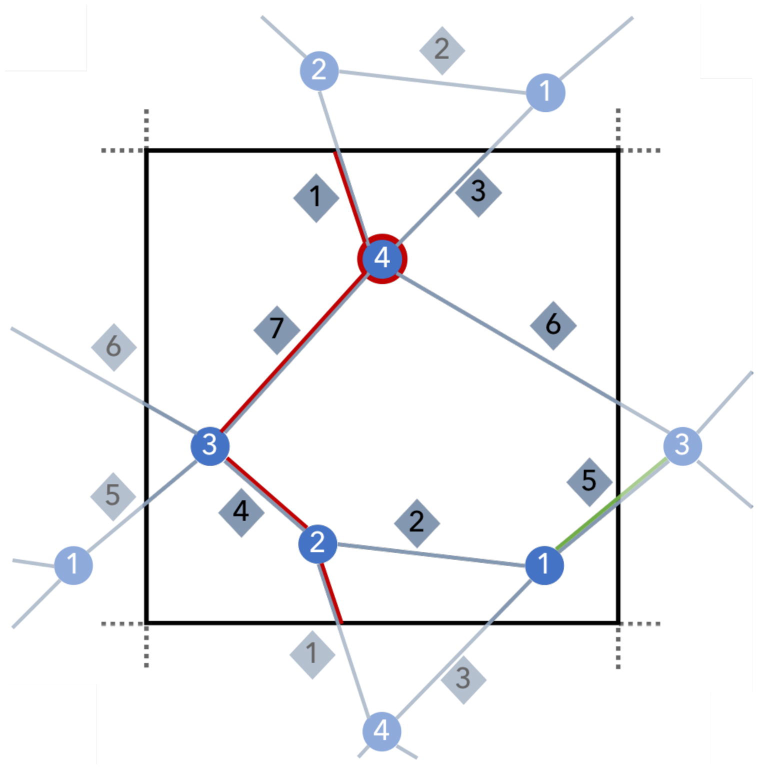
Tuple[`List` of `List` of int, int]

CHAPTER

THREE

A SIMPLE EXAMPLE

Consider the periodic net below, with only 4 nodes and 7 bonds. Each node has at one or two bonds that remain within the unit cell, and one or two bonds that cross a boundary. The example is twodimensional to facilitate visualization, but the code below includes 3 dimensions for all boundary-crossing vectors so the example can be more easily translated to a 3D setting.



```python
# Usage example for perconet package
# See https://github.com/wouterel/perconet

import perconet as pn
import numpy as np


def test_simple():
    # the example starts counting node numbers from 1,
    # but perconet counts from 0, so we define 5 nodes instead of 4.
    # The unused node "0" does not affect the percolation properties.
```

(continues on next page)

(continued from previous page)

```
    number_of_nodes = 5
    max_coordination = 6
    testnet = pn.PeriodicNetwork(number_of_nodes,
                                 max_coordination,
                                 verbose=False)

    # first add the three internal edges connecting 1-2, 2-3, and 3-4.
    testnet.add_edge(1, 2, np.array([0, 0, 0]))
    testnet.add_edge(2, 3, np.array([0, 0, 0]))
    testnet.add_edge(3, 4, np.array([0, 0, 0]))

    # we now have a small network that doesn't do anything with the boundaries yet
    loopfinder = pn.LoopFinder(testnet, verbose=False)
    loops, Nloops = loopfinder.get_independent_loops()
    print(f"Found {Nloops} loops ( = 0 because no boundary-crossing bonds are defined).")

    print("Adding the boundary-crossing bonds")
    # Note the sign of the boundary crossing for an edge between i and j
    # is determined by the direction in which you go if you follow the edge from i to j
    # add a bond between nodes 1 and 3 that crosses the x-boundary
    testnet.add_edge(1, 3, np.array([1, 0, 0]))
    # add a bond between nodes 1 and 4 that crosses the negative y-boundary
    testnet.add_edge(1, 4, np.array([0, -1, 0]))
    # add a bond between nodes 2 and 4 that crosses the negative y-boundary
    testnet.add_edge(2, 4, np.array([0, -1, 0]))
    # add a bond between nodes 3 and 4 that crosses the negative x-boundary
    testnet.add_edge(3, 4, np.array([-1, 0, 0]))

    # now the network percolates across x and y boundaries.
    loops, Nloops = loopfinder.get_independent_loops()
    print(f"Found {Nloops} independent loops.")
    for loop in loops:
        print(f"Loop: {loop}")


if __name__ == "__main__":
    test_simple()
```

CHAPTER

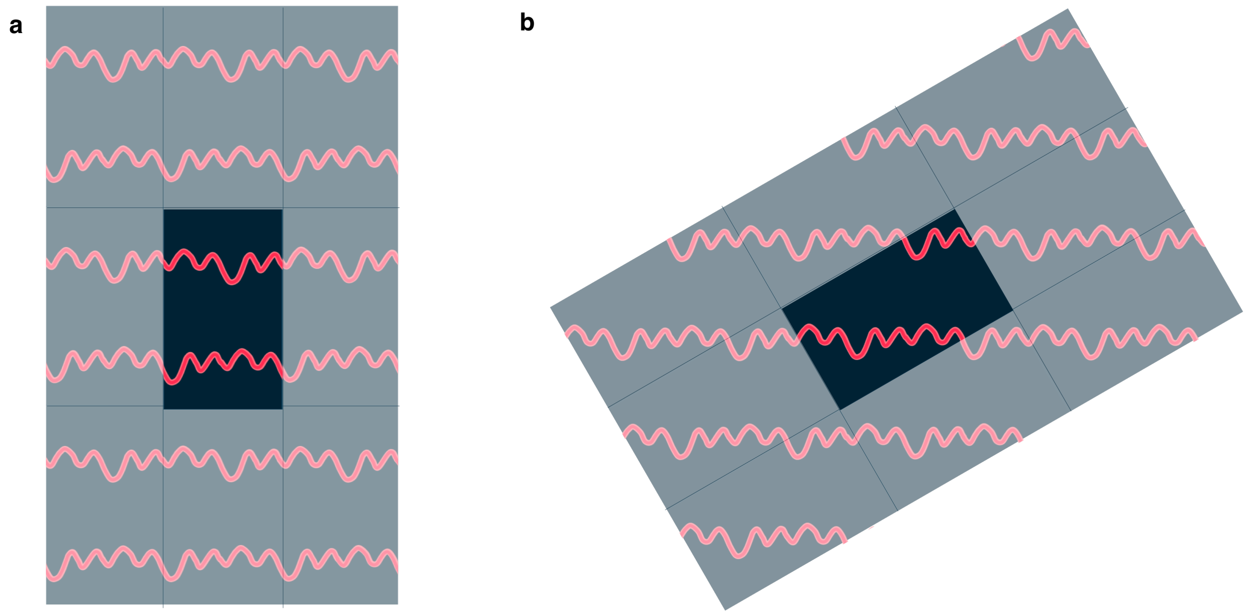
FOUR

WHEN AND WHY TO USE

The properties of structures defined using a *unit cell* or *box with periodic boundaries* depend crucially on whether those structures do or do not connect to themselves across the periodic boundaries. When the structures are somehow disordered, the task of determining whether a structure does this can become nontrivial, regardless whether the data is computer-generated or the result of an experiment.

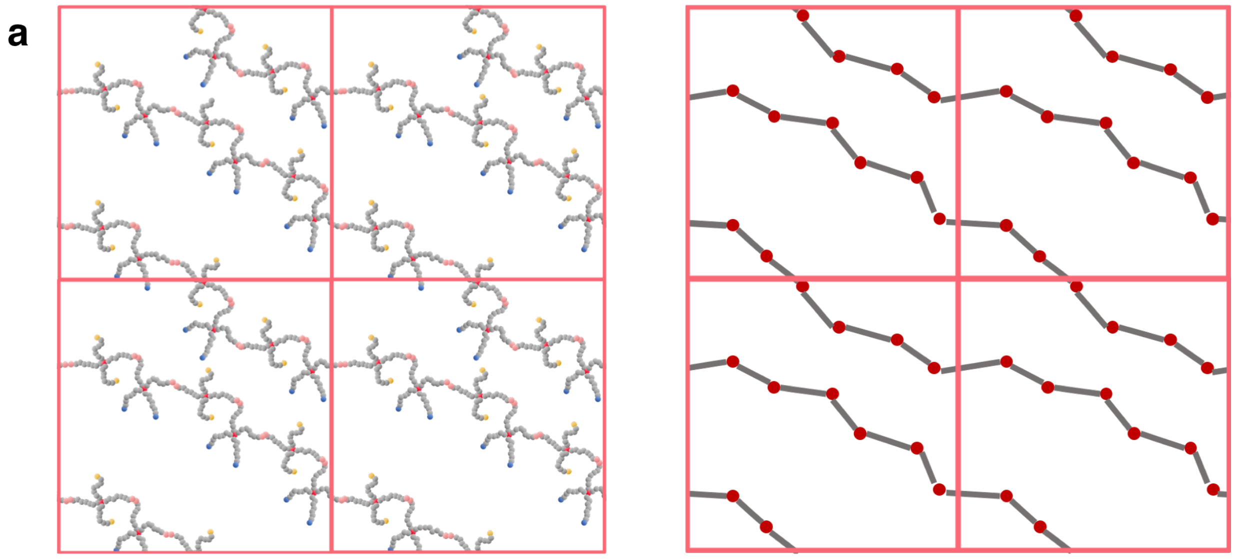
When the structures do connect to themselves accross a boundary, this is called *percolation*. Its relevance becomes clear when the unit cell is repeated many times in all directions, because percolating structures then become infinitely large structures. Often the answer is clear-cut and there is either no percolation or percolation in all directions. But the edge cases can be nontrivial to analyse. **perconet** employs a loop-finding algorithm that covers these edge cases correctly. Obviously the easier cases can also be analyzed using **perconet**.

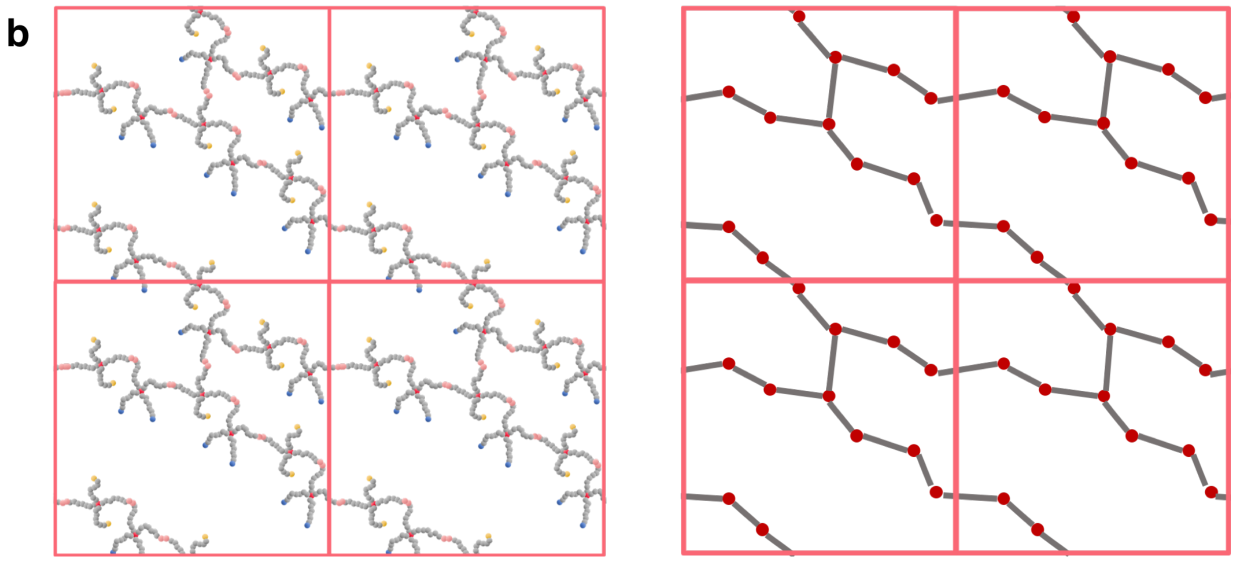
A more detailed motivation for our work will be published soon. In summary, consider the following graphic that shows the same perodic structure, but with two different choices for the unit cell. This graphic shows that a simple analysis in which the two lattice directions are considered separately cannot cover all cases correctly. From the unit cell on the left, one would conclude that this network percolates only in the x-direction, while the unit cell on the right suggests that it percolates in both x- and y-directions.



The correct answer here is that this structure only percolates in a single direction. To get the correct answer regardless of the choice of unit cell, **perconet** employs an algorithm that finds the loops in the periodic structure (called a periodic *net* or *network* or *graph*) that start from some site in the network and go around one or several of the boundaries to end up at the same site.

If your data is complex, like the samples from a polymer simulation below, it may be useful to reduce your network to its essential backbone, as shown below. Tools to do this in an automated fashion for e.g. LAMMPS simulation data are being developed.





This example shows that a single added bond in the network can make the difference between a network that percolates in only one direction vs. two.

4.1 Loop independence

If the loop finder identifies a loop that goes around both the $+x$ and $+y$ boundaries $\left[\vec{b}_1 = (1,1,0)\right]$, and another loop that only goes around the $+x$ boundary $\left[\vec{b}_2 = (1,0,0)\right]$, we can construct a loop with $\vec{b} = (0,1,0)$ by first going around the first loop and then going around the second loop in reverse: $\vec{b} = \vec{b}_1 - \vec{b}_2$. Generalizing, any linear combination of loops with integer coefficients is also a loop. Thus it makes sense to reduce the list of loops to a list of *independent* loops by constructing a basis of independent loops. Because the basis is to be used only with integer coefficients (one cannot go around a loop half a time), it is a lattice basis and the space of allowed loops is a lattice. Writing the list of loops as a matrix (each row representing a loop), the reduction is like gaussian elimination, but with the constraint that only integer multiples of loops can be added to other loops and multiplying a row by a constant is not allowed (except for -1 which is just reversing the direction of a loop).

A way of reducing the list of loops to a list of independent loops that gives a unique result, so one can compare different loop structures, is to cast it into Hermite normal form. See Wikipedia or your favorite linear algebra text for details. This is the form `perconet.LoopFinder.get_independent_loops()` returns. Note that the exact definition of Hermite normal form varies slightly between authors.

CHAPTER

FIVE

INFORMATION FOR...

5.1 Those confused about terminology

Most of the jargon used comes from the mathematics of graphs, with nodes (points) connected by edges (lines). In the context of gelation, the nodes will represent molecules or colloids, and the edges will represent chemical or physical bonds. In many cases, edges or bonds may also be called *links*.

We use the mathematical term *graph* to denote any graph, and the term *periodic nets* or *periodic networks* to denote graphs that are embedded in a periodic box.

5.2 Chemists

The most obvious use case in chemistry for **perconet** is detecting *gelation*. Models for a gelation process with periodic boundary conditions will typically lead to data that specifies positions for the building blocks (monomers) and a list of bonds that have been generated during the gelation process. There will typically be one large molecule and many small ones, and **perconet** will determine for you whether that large molecule connects to itself around the periodic boundary, signalling the presence of an infinite molecule, the *gel*.

While primarily written for periodic systems, it is also possible to use **perconet** for percolation analysis of systems with simple boundaries. To this end, denote a certain subset of the nodes to be one *side* of the system, and another subset to be the other side, and then ask **perconet** if the two sides are connected. With this approach, even the output of an experimental image analysis process could be used as input. We may features to perconet in the future to automate this.

5.3 Mathematicians

The three-dimensional periodic boxes that inspired this package are a topological space known as a 3-torus. The use of the package is, however, not limited to three dimensions and can be used to analyze graphs embedded in any cartesian power of the circle $\mathbb{T}^d = S^d$.

A loop in such an embedded graph is characterized by an element of the fundamental group of the d-torus, which is $\mathbb{Z}^d$. The element specifies, for each periodic boundary, how often the loop in question goes around that boundary.

Not all elements of the fundamental group of the d-torus are necessarily represented in every graph: Perhaps it only wraps around one of the boundaries, or some boundary can only be looped around an even number of times. The subgroup of $\mathbb{Z}^d$ that is actually realized by the periodic net is a lattice, for which the method `perconet.LoopFinder.get_independent_loops()` provides a basis. One can also say these are the generators of the subgroup. The basis is provided through a matrix of which the rows are the basis vectors, which is presented in Hermite Normal Form to make the choice of basis vectors unique. This gives a characterization of the topological strcuture of a periodic nets that can be used to define equivalence between them. For some applications it may be desirable to have a near-orthogonal basis, in which case improving it via the LLL-algorithm may prove useful.

5.4 Physicists and mechanical engineers

Related to the gelation application described above. Perconet can be used to extract the percolation properties of structures and spring networks, and thus provide information on the rigidity of network structures. Percolating directions generally indicate directions in which the network would be able to support a tensile load. Conversely, directions that are perpendicular to all percolating directions are directions in which which structure is not rigid. See *When and why to use* for more details.

CHAPTER SIX

LICENSE

```
                      EUROPEAN UNION PUBLIC LICENCE v. 1.2
                      EUPL © the European Union 2007, 2016

This European Union Public Licence (the ‘EUPL’) applies to the Work (as defined
below) which is provided under the terms of this Licence. Any use of the Work,
other than as authorised under this Licence is prohibited (to the extent such
use is covered by a right of the copyright holder of the Work).

The Work is provided under the terms of this Licence when the Licensor (as
defined below) has placed the following notice immediately following the
copyright notice for the Work:

        Licensed under the EUPL

or has expressed by any other means his willingness to license under the EUPL.

1. Definitions

In this Licence, the following terms have the following meaning:

- ‘The Licence’: this Licence.

- ‘The Original Work’: the work or software distributed or communicated by the
  Licensor under this Licence, available as Source Code and also as Executable
  Code as the case may be.

- ‘Derivative Works’: the works or software that could be created by the
  Licensee, based upon the Original Work or modifications thereof. This Licence
  does not define the extent of modification or dependence on the Original Work
  required in order to classify a work as a Derivative Work; this extent is
  determined by copyright law applicable in the country mentioned in Article 15.

- ‘The Work’: the Original Work or its Derivative Works.

- ‘The Source Code’: the human-readable form of the Work which is the most
  convenient for people to study and modify.

- ‘The Executable Code’: any code which has generally been compiled and which is
  meant to be interpreted by a computer as a program.
```

(continues on next page)

(continued from previous page)

```
- ‘The Licensor’: the natural or legal person that distributes or communicates
  the Work under the Licence.

- ‘Contributor(s)’: any natural or legal person who modifies the Work under the
  Licence, or otherwise contributes to the creation of a Derivative Work.

- ‘The Licensee’ or ‘You’: any natural or legal person who makes any usage of
  the Work under the terms of the Licence.

- ‘Distribution’ or ‘Communication’: any act of selling, giving, lending,
  renting, distributing, communicating, transmitting, or otherwise making
  available, online or offline, copies of the Work or providing access to its
  essential functionalities at the disposal of any other natural or legal
  person.

2. Scope of the rights granted by the Licence

The Licensor hereby grants You a worldwide, royalty-free, non-exclusive,
sublicensable licence to do the following, for the duration of copyright vested
in the Original Work:

- use the Work in any circumstance and for all usage,
- reproduce the Work,
- modify the Work, and make Derivative Works based upon the Work,
- communicate to the public, including the right to make available or display
  the Work or copies thereof to the public and perform publicly, as the case may
  be, the Work,
- distribute the Work or copies thereof,
- lend and rent the Work or copies thereof,
- sublicense rights in the Work or copies thereof.

Those rights can be exercised on any media, supports and formats, whether now
known or later invented, as far as the applicable law permits so.

In the countries where moral rights apply, the Licensor waives his right to
exercise his moral right to the extent allowed by law in order to make effective
the licence of the economic rights here above listed.

The Licensor grants to the Licensee royalty-free, non-exclusive usage rights to
any patents held by the Licensor, to the extent necessary to make use of the
rights granted on the Work under this Licence.

3. Communication of the Source Code

The Licensor may provide the Work either in its Source Code form, or as
Executable Code. If the Work is provided as Executable Code, the Licensor
provides in addition a machine-readable copy of the Source Code of the Work
along with each copy of the Work that the Licensor distributes or indicates, in
a notice following the copyright notice attached to the Work, a repository where
the Source Code is easily and freely accessible for as long as the Licensor
continues to distribute or communicate the Work.
```

(continues on next page)

(continued from previous page)

```
4. Limitations on copyright

Nothing in this Licence is intended to deprive the Licensee of the benefits from
any exception or limitation to the exclusive rights of the rights owners in the
Work, of the exhaustion of those rights or of other applicable limitations
thereto.

5. Obligations of the Licensee

The grant of the rights mentioned above is subject to some restrictions and
obligations imposed on the Licensee. Those obligations are the following:

Attribution right: The Licensee shall keep intact all copyright, patent or
trademarks notices and all notices that refer to the Licence and to the
disclaimer of warranties. The Licensee must include a copy of such notices and a
copy of the Licence with every copy of the Work he/she distributes or
communicates. The Licensee must cause any Derivative Work to carry prominent
notices stating that the Work has been modified and the date of modification.

Copyleft clause: If the Licensee distributes or communicates copies of the
Original Works or Derivative Works, this Distribution or Communication will be
done under the terms of this Licence or of a later version of this Licence
unless the Original Work is expressly distributed only under this version of the
Licence - for example by communicating ‘EUPL v. 1.2 only’. The Licensee
(becoming Licensor) cannot offer or impose any additional terms or conditions on
the Work or Derivative Work that alter or restrict the terms of the Licence.

Compatibility clause: If the Licensee Distributes or Communicates Derivative
Works or copies thereof based upon both the Work and another work licensed under
a Compatible Licence, this Distribution or Communication can be done under the
terms of this Compatible Licence. For the sake of this clause, ‘Compatible
Licence’ refers to the licences listed in the appendix attached to this Licence.
Should the Licensee's obligations under the Compatible Licence conflict with
his/her obligations under this Licence, the obligations of the Compatible
Licence shall prevail.

Provision of Source Code: When distributing or communicating copies of the Work,
the Licensee will provide a machine-readable copy of the Source Code or indicate
a repository where this Source will be easily and freely available for as long
as the Licensee continues to distribute or communicate the Work.

Legal Protection: This Licence does not grant permission to use the trade names,
trademarks, service marks, or names of the Licensor, except as required for
reasonable and customary use in describing the origin of the Work and
reproducing the content of the copyright notice.

6. Chain of Authorship

The original Licensor warrants that the copyright in the Original Work granted
hereunder is owned by him/her or licensed to him/her and that he/she has the
power and authority to grant the Licence.
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
Similarly, you irrevocably accept this Licence and all of its terms and
conditions by exercising any rights granted to You by Article 2 of this Licence,
such as the use of the Work, the creation by You of a Derivative Work or the
Distribution or Communication by You of the Work or copies thereof.

11. Information to the public

In case of any Distribution or Communication of the Work by means of electronic
communication by You (for example, by offering to download the Work from a
remote location) the distribution channel or media (for example, a website) must
at least provide to the public the information requested by the applicable law
regarding the Licensor, the Licence and the way it may be accessible, concluded,
stored and reproduced by the Licensee.

12. Termination of the Licence

The Licence and the rights granted hereunder will terminate automatically upon
any breach by the Licensee of the terms of the Licence.

Such a termination will not terminate the licences of any person who has
received the Work from the Licensee under the Licence, provided such persons
remain in full compliance with the Licence.

13. Miscellaneous

Without prejudice of Article 9 above, the Licence represents the complete
agreement between the Parties as to the Work.

If any provision of the Licence is invalid or unenforceable under applicable
law, this will not affect the validity or enforceability of the Licence as a
whole. Such provision will be construed or reformed so as necessary to make it
valid and enforceable.

The European Commission may publish other linguistic versions or new versions of
this Licence or updated versions of the Appendix, so far this is required and
reasonable, without reducing the scope of the rights granted by the Licence. New
versions of the Licence will be published with a unique version number.

All linguistic versions of this Licence, approved by the European Commission,
have identical value. Parties can take advantage of the linguistic version of
their choice.

14. Jurisdiction

Without prejudice to specific agreement between parties,

- any litigation resulting from the interpretation of this License, arising
  between the European Union institutions, bodies, offices or agencies, as a
  Licensor, and any Licensee, will be subject to the jurisdiction of the Court
  of Justice of the European Union, as laid down in article 272 of the Treaty on
  the Functioning of the European Union,
```

(continues on next page)

(continued from previous page)

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- search

INDEX