Module Lar2psm *

Alberto Paoluzzi

January 7, 2014

Abstract

This software module contains all the functions needed to interface the LAR data structure and/or the geometric objects defined by it with the Plasm environment. In particular, it will include the interfaces towards the visualization primitives provided by the language.

Contents

---

*This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework [CL13]. January 7, 2014

1

1 Introduction

The standard definition of vectors and matrices in plasm is the list of vector coordinates and the list of matrix rows, respectively.

2 Implementation

Since the present `lar2psm` module is an interface between the `larcc` library and the PLaSM language, and its various incarnations, it should allow to import the language itself (in Python, the `pyplasm` module).

⟨ Import the pyplasm module 2a ⟩ ≡
```
from pyplasm import *
```
    ◇

Macro never referenced.

An useful utility will allow for the creation of a subdirectory from a `dirpath` *string*.

⟨ Create directory from path 2b ⟩ ≡
```
import os
def createDir(dirpath):
    if not os.path.exists(dirpath):
        os.makedirs(dirpath)
```
    ◇

Macro referenced in 2c, 6a.

It may be useful to define the repository(ies) for the unit tests associated to the module:

"test/py/lar2psm-tests.py" 2c ≡
```
⟨ Create directory from path 2b ⟩
createDir('test/py/lar2psm/')
```
    ◇

2.1 Convex combination

Next we define the `CCOMB` function that accepts as input a `vectors` list (i.e., a matrix) and returns *the* point their convex combination.

⟨ Compute the convex combination of a list of vectors 2d ⟩ ≡
```
import scipy as sp
from pyplasm import *
def CCOMB(vectors):
    return (sp.array(VECTSUM(vectors)) / float(len(vectors))).tolist()
```
    ◇

Macro referenced in 5d.

Unit tests   First we test `CCOMB` with some special data, then with some random vectors.

`"test/py/lar2psm/test-ccomb.py"` 3a ≡
⟨ Import the module (3b `lar2psm`) 5a ⟩
`from lar2psm import *`
⟨ CCOMB unit tests 7 ⟩
◇

2.2   LAR model of a cell complex

A very important concept introduced by the LAR package is the definition of the *model* of a cell complex, as a pair made by a list of vertices, given as lists of coordinates, and a topological relation.

Definition 1 (LAR model). *A LAR model is a pair, e.g. a Python tuple* (V, FV), *where:*

1. `V` *is the list of vertices, given as lists of coordinates;*

2. `FV` *is a* cell-vertex *relation, in this case the face-vertex relation, given as a list of cells, where each cell is given as a list of vertex indices.*

Examples   Some very simple examples of 0D, 1D, and 2D models follows. They are displayed in Figure 1.

⟨ 2D model examples 3c ⟩ ≡
```
V = [[0.,0.],[1.,0.],[0.,1.],[1.,1.],[0.5,0.5]]
VV = [[0],[1],[2],[3],[4]]
EV = [[0,1],[0,2],[0,4],[1,3],[1,4],[2,3],[2,4],[3,4]]
FV = [[0,1,4],[1,3,4],[2,3,4],[0,2,4]]

model0d, model1d, model2d = (V,VV), (V,EV), (V,FV)
```
◇

Macro referenced in 6d.

2.3   Function `MKPOLS`

The function `MKPOLS` returns a list of HPC objects, i.e. the geometric type of the PLaSM language. This list is generated to be displayed, possibly exploded, by the `pyplasm` viewer.

Each cell `f` in the model (i.e. each vertex list in the `FV` array of the previous example) is mapped into a polyhedral cell by the `pyplasm` operator `MKPOL`. The vertex indices are mapped from base 0 (the Python and C standard) to base 1 (the Plasm, Matlab, and FORTRAN standard).

⟨ MaKe a list of HPC objects from a LAR model 4a ⟩ ≡

```
def MKPOLS (model):
    V, FV = model
    pols = [MKPOL([[V[v] for v in f],[range(1,len(f)+1)], None]) for f in FV]
    return pols
```
◇

Macro referenced in 5d.

Unit tests   Some simple 3D, 2D, 1D and 0D models are generated and visualised exploded by the file

"test/py/lar2psm/test-models.py" 4b ≡

```
⟨ Import the module (4c lar2psm ) 5a ⟩
⟨ View model examples 6d ⟩
```
◇

2.4   "Explosion" of the scene

A function EXPLODE used to "explode" an HPC scene defined as a *list* of HPC values, given three real scaling parameters, sx,sy,sz, that are used to transform the position of the centroid of each HPC cell. HPC stands for *HierarchicaL Polyhedral Complex*, the type of plasm geometric values. Of course the assertion

$$sx, sy, sz \geq 1.0$$

must be true, otherways the function would induce some compenetration of the cells of the scene.

⟨ Explode the scene using sx,sy,sz scaling parameters 4d ⟩ ≡

```
def EXPLODE (sx,sy,sz):
    def explode0 (scene):
        centers = [CCOMB(S1(UKPOL(obj))) for obj in scene]
        scalings = len(centers) * [S([1,2,3])([sx,sy,sz])]
        scaledCenters = [UK(APPLY(pair)) for pair in
                         zip(scalings, [MK(p) for p in centers])]
        translVectors = [ VECTDIFF((p,q)) for (p,q) in zip(scaledCenters, centers) ]
        translations = [ T([1,2,3])(v) for v in translVectors ]
        return STRUCT([ t(obj) for (t,obj) in zip(translations,scene) ])
    return explode0
```
◇

Macro referenced in 5d.

The EXPLODE function is second order: it first application (to the scaling parameters) returns a partial function to be applied to the scene, given as a *list* of HPC (Hierarchical Polyhedral Complex) objects. EXPLODE is dimension-independent, since it can be applied

4

to points, edges, faces, 3D cells, and even to geometric values of mixed dimensionality (see Figure 1).

It works by computing the centroid of each object, and by applying to each of them a translation equal to the difference betwwen the scaled and the initial positions of its centroid. `EXPLODE` returns a single HPC object (the assembly of input objects, properly translated)

3   Source Output: `lar2psm` module

3.1   Importing a generic module

First we define a parametric macro to allow the importing of `larcc` modules from the project repository `lib/py/`. When the user needs to import some project's module, she may call this macro as done in Section 3.2.

⟨Import the module 5a⟩ ≡
```
    import sys
    sys.path.insert(0, 'lib/py/')
    import @1
```
    ◇

Macro referenced in 3a, 4b, 5bd.

Importing a module   A function used to import a generic `lacccc` module within the current environment is also useful.

⟨Function to import a generic module 5b⟩ ≡
```
    def importModule(moduleName):
```
        ⟨Import the module (5c `moduleName`) 5a⟩
    ◇

Macro referenced in 5d.

3.2   Lar2psm exporting

Here we assemble top-down the `lar2psm` module, by orderly listing the functional parts it is composed of. Of course, this one is the module version corresponding to the current state of the system, i.e. to a very initial state. Other functions will be added when needed.

"lib/py/lar2psm.py" 5d ≡
```
    """Module with functions needed to interface LAR with pyplasm"""
```
    ⟨Import the module (5e `simplexn`) 5a⟩
    ⟨Function to import a generic module 5b⟩
    ⟨Compute the convex combination of a list of vectors 2d⟩
    ⟨MaKe a list of HPC objects from a LAR model 4a⟩
    ⟨Explode the scene using `sx,sy,sz` scaling parameters 4d⟩
    ◇

4 Unit tests

4.1 Creation of repository of unit tests

A possible unit test strategy is to create a directory for unit tests associated to each source file in `nuweb`. Therefore we create here a directory in `test/py/` with the same name of the present document. Of course other

⟨ create directory and echo of creation 6a ⟩ ≡
```
    ⟨ Create directory from path 2b ⟩
    createDir('@1')
    print "'@1' repository created"
    ◇
```
Macro never referenced.

`"test/py/lar2psm/test01.py"` 6b ≡
```
    ⟨ create directory and echo of creation: (6c test/py/lar2psm/ ) ? ⟩
    ◇
```

4.2 Viewing some simplicial complexes

Let we start producing some images, displayed in Figure 1, os a small simplicial complex and of its skeletons. Notice that the `+` character operates the join of lists (of HPC values).

⟨ View model examples 6d ⟩ ≡
```
    from lar2psm import *
    ⟨ 2D model examples 3c ⟩
    explode = EXPLODE(1.5,1.5,1.5)
    VIEW(explode(MKPOLS(model0d)))
    VIEW(explode(MKPOLS(model1d)))
    VIEW(explode(MKPOLS(model2d)))
    VIEW(explode(MKPOLS(model2d) + MKPOLS(model1d) + MKPOLS(model0d)))
    ◇
```
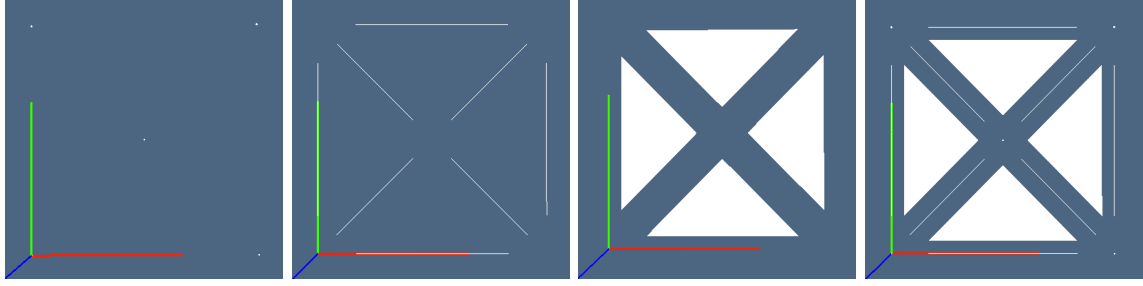Macro referenced in 4b.

Figure 1: Images of the skeletons of a small simplicial complex.

4.3   Testing convex combination of vectors

⟨ CCOMB unit tests 7 ⟩ ≡

```
assert( CCOMB([]) == [] )
assert( CCOMB([[0,1]]) == [0.0, 1.0] )
assert( CCOMB([[0,1],[1,0]]) == [0.5, 0.5] )
assert( CCOMB([[1,0,0],[0,1,0],[0,0,1]]) == [1./3,1./3,1./3])

import random
vects = [[random.random() for i in range(3)] for k in range(4)]
assert( CCOMB([VECTSUM(vects)]) == \
        (sp.array(CCOMB(vects)) * len(vects)).tolist() )
◇
```

Macro referenced in 3a.

References

[CL13] CVD-Lab, *Linear algebraic representation*, Tech. Report 13-00, Roma Tre Univer-
        sity, October 2013.