The basic `larcc` module *

The LARCC team

January 9, 2014

Contents

*This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework [CL13]. January 9, 2014

1 Basic representations

A few basic representation of topology are used in LARCC. They include some common sparse matrix representations: CSR (Compressed Sparse Row), CSC (Compressed Sparse Column), COO (Coordinate Representation), and BRC (Binary Row Compressed).

1.1 BRC (Binary Row Compressed)

We denote as BRC (Binary Row Compressed) the standard input representation of our LARCC framework. A BRC representation is an array of arrays of integers, with no requirement of equal length for the component arrays. The BRC format is used to represent a (normally sparse) binary matrix. Each component array corresponds to a matrix row, and contains the indices of columns that store a 1 value. No storage is used for 0 values.

BRC format example Let $A = (a_{i,j} \in \{0, 1\})$ be a binary matrix. The notation $\texttt{BRC}(A)$ is used for the corresponding data structure.

$$A = \begin{pmatrix} 0,1,0,0,0,0,0,1,0,0 \\ 0,0,1,0,0,0,0,0,0,0 \\ 1,0,0,1,0,0,0,0,0,1 \\ 1,0,0,0,0,0,1,0,0,0 \\ 0,0,0,0,0,1,1,1,0,0 \\ 0,0,1,0,1,0,0,0,1,0 \\ 0,0,0,0,0,0,0,0,0,0 \\ 0,1,0,0,0,0,0,1,0,1 \\ 0,0,0,1,0,0,0,0,1,0 \\ 0,1,1,0,1,0,0,0,0,0 \end{pmatrix} \qquad \mapsto \qquad \texttt{BRC}(A) = \begin{array}{l} \texttt{[[1,7],} \\ \texttt{ [2],} \\ \texttt{ [0,3,9],} \\ \texttt{ [0,6],} \\ \texttt{ [5,6,7],} \\ \texttt{ [2,4,8],} \\ \texttt{ [],} \\ \texttt{ [1,7,9],} \\ \texttt{ [3,8],} \\ \texttt{ [1,2,4]]} \end{array}$$

1.2 Format conversions

First we give the function `format` to make the transformation from the sparse matrix as a list of triples *(row,column,value)* for each non-zero element, to the `scipy.sparse` format corresponding to the `shape` parameter, set by default to `"csr"`, that stands for *Compressed Sparse Row*, the normal matrix format of the LARCC framework.

⟨ From list of triples to scipy.sparse 3a ⟩ ≡

```
def format(triples,shape="csr"):
    n = len(triples)
    data = arange(n)
    ij = arange(2*n).reshape(2,n)
    for k,item in enumerate(triples):
        ij[0][k],ij[1][k],data[k] = item
    return scipy.sparse.coo_matrix((data, ij)).asformat(shape)
◇
```

Macro referenced in 16a.

⟨ Brc to Coo transformation 3b ⟩ ≡

```
def cooCreateFromBrc(ListOfListOfInt):
    COOm = [[k,col,1] for k,row in enumerate(ListOfListOfInt)
            for col in row ]
    return COOm
◇
```

Macro referenced in 16a.

⟨ Test example of Brc to Coo transformation 3c ⟩ ≡

```
print "\n>>> cooCreateFromBrc"
V = [[0, 0], [1, 0], [2, 0], [0, 1], [1, 1], [2, 1]]
FV = [[0, 1, 3], [1, 2, 4], [1, 3, 4], [2, 4, 5]]
EV = [[0,1],[0,3],[1,2],[1,3],[1,4],[2,4],[2,5],[3,4],[4,5]]
cooFV = cooCreateFromBrc(FV)
cooEV = cooCreateFromBrc(EV)
print "\ncooCreateFromBrc(FV) =\n", cooFV
print "\ncooCreateFromBrc(EV) =\n", cooEV
◇
```

Macro referenced in 16b.

⟨ Coo to Csr transformation 3d ⟩ ≡

```
def csrCreateFromCoo(COOm):
    CSRm = format(COOm,"csr")
    return CSRm
◇
```

Macro referenced in 16a.

⟨ Test example of Coo to Csr transformation 4a ⟩ ≡

```
print "\n>>> csrCreateFromCoo"
csrFV = csrCreateFromCoo(cooFV)
csrEV = csrCreateFromCoo(cooEV)
print "\ncsr(FV) =\n", repr(csrFV)
print "\ncsr(EV) =\n", repr(csrEV)
◇
```

Macro referenced in 16b.

⟨ Brc to Csr transformation 4b ⟩ ≡

```
def csrCreate(BRCm,shape=(0,0)):
    if shape == (0,0):
        out = csrCreateFromCoo(cooCreateFromBrc(BRCm))
        return out
    else:
        CSRm = scipy.sparse.csr_matrix(shape)
        for i,j,v in cooCreateFromBrc(BRCm):
            CSRm[i,j] = v
        return CSRm
◇
```

Macro referenced in 16a.

⟨ Test example of Brc to Csr transformation 4c ⟩ ≡

```
print "\n>>> csrCreateFromCoo"
V = [[0, 0], [1, 0], [2, 0], [0, 1], [1, 1], [2, 1]]
FV = [[0, 1, 3], [1, 2, 4], [1, 3, 4], [2, 4, 5]]
csrFV = csrCreate(FV)
print "\ncsrCreate(FV) =\n", csrFV
◇
```

Macro referenced in 16b.

2 Matrix operations

⟨ Query Matrix shape 4d ⟩ ≡

```
def csrGetNumberOfRows(CSRm):
    Int = CSRm.shape[0]
    return Int

def csrGetNumberOfColumns(CSRm):
    Int = CSRm.shape[1]
    return Int
◇
```

Macro referenced in 16a.

⟨ Test examples of Query Matrix shape 5a ⟩ ≡

```
print "\n>>> csrGetNumberOfRows"
print "\ncsrGetNumberOfRows(csrFV) =", csrGetNumberOfRows(csrFV)
print "\ncsrGetNumberOfRows(csrEV) =", csrGetNumberOfRows(csrEV)
print "\n>>> csrGetNumberOfColumns"
print "\ncsrGetNumberOfColumns(csrFV) =", csrGetNumberOfColumns(csrFV)
print "\ncsrGetNumberOfColumns(csrEV) =", csrGetNumberOfColumns(csrEV)
◇
```

Macro referenced in 16b.

⟨ Sparse to dense matrix transformation 5b ⟩ ≡

```
def csrToMatrixRepresentation(CSRm):
    nrows = csrGetNumberOfRows(CSRm)
    ncolumns = csrGetNumberOfColumns(CSRm)
    ScipyMat = zeros((nrows,ncolumns),int)
    C = CSRm.tocoo()
    for triple in zip(C.row,C.col,C.data):
        ScipyMat[triple[0],triple[1]] = triple[2]
    return ScipyMat
◇
```

Macro referenced in 16a.

⟨ Test examples of Sparse to dense matrix transformation 5c ⟩ ≡

```
print "\n>>> csrToMatrixRepresentation"
print "\nFV =\n", csrToMatrixRepresentation(csrFV)
print "\nEV =\n", csrToMatrixRepresentation(csrEV)
◇
```

Macro referenced in 16b.

⟨ Matrix product and transposition 5d ⟩ ≡

```
def matrixProduct(CSRm1,CSRm2):
    CSRm = CSRm1 * CSRm2
    return CSRm

def csrTranspose(CSRm):
    CSRm = CSRm.T
    return CSRm
◇
```

Macro referenced in 16a.

⟨ Matrix filtering to produce the boundary matrix 6a ⟩ ≡

```
def csrBoundaryFilter(CSRm, facetLengths):
    maxs = [max(CSRm[k].data) for k in range(CSRm.shape[0])]
    inputShape = CSRm.shape
    coo = CSRm.tocoo()
    for k in range(len(coo.data)):
        if coo.data[k]==maxs[coo.row[k]]: coo.data[k] = 1
        else: coo.data[k] = 0
    mtx = coo_matrix((coo.data, (coo.row, coo.col)), shape=inputShape)
    out = mtx.tocsr()
    return out
◇
```

Macro referenced in 16a.

⟨ Test example of Matrix filtering to produce the boundary matrix 6b ⟩ ≡

```
print "\n>>> csrBoundaryFilter"
csrEF = matrixProduct(csrFV, csrTranspose(csrEV)).T
facetLengths = [csrCell.getnnz() for csrCell in csrEV]
CSRm = csrBoundaryFilter(csrEF, facetLengths).T
print "\ncsrMaxFilter(csrFE) =\n", csrToMatrixRepresentation(CSRm)
◇
```

Macro referenced in 16b.

⟨ Matrix filtering via a generic predicate 6c ⟩ ≡

```
def csrPredFilter(CSRm, pred):
   # can be done in parallel (by rows)
   coo = CSRm.tocoo()
   triples = [[row,col,val] for row,col,val
            in zip(coo.row,coo.col,coo.data) if pred(val)]
   i, j, data = TRANS(triples)
   CSRm = scipy.sparse.coo_matrix((data,(i,j)),CSRm.shape).tocsr()
   return CSRm
◇
```

Macro referenced in 16a.

⟨ Test example of Matrix filtering via a generic predicate 6d ⟩ ≡

```
print "\n>>> csrPredFilter"
CSRm = csrPredFilter(matrixProduct(csrFV, csrTranspose(csrEV)).T, GE(2)).T
print "\nccsrPredFilter(csrFE) =\n", csrToMatrixRepresentation(CSRm)
◇
```

Macro referenced in 16b.

3 Topological operations

⟨ From cells and facets to boundary operator 7a ⟩ ≡

```
def boundary(cells,facets):
    csrCV = csrCreate(cells)
    csrFV = csrCreate(facets)
    csrFC = matrixProduct(csrFV, csrTranspose(csrCV))
    facetLengths = [csrCell.getnnz() for csrCell in csrCV]
    return csrBoundaryFilter(csrFC,facetLengths)

def coboundary(cells,facets):
    Boundary = boundary(cells,facets)
    return csrTranspose(Boundary)
◇
```

Macro referenced in 16a.

⟨ Test examples of From cells and facets to boundary operator 7b ⟩ ≡

```
V = [[0.0, 0.0, 0.0], [1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [1.0, 1.0, 0.0],
[0.0, 0.0, 1.0], [1.0, 0.0, 1.0], [0.0, 1.0, 1.0], [1.0, 1.0, 1.0]]

CV =[[0, 1, 2, 4], [1, 2, 4, 5], [2, 4, 5, 6], [1, 2, 3, 5], [2, 3, 5, 6],
[3, 5, 6, 7]]

FV =[[0, 1, 2], [0, 1, 4], [0, 2, 4], [1, 2, 3], [1, 2, 4], [1, 2, 5],
[1, 3, 5], [1, 4, 5], [2, 3, 5], [2, 3, 6], [2, 4, 5], [2, 4, 6], [2, 5, 6],
[3, 5, 6], [3, 5, 7], [3, 6, 7], [4, 5, 6], [5, 6, 7]]

EV =[[0, 1], [0, 2], [0, 4], [1, 2], [1, 3], [1, 4], [1, 5], [2, 3], [2, 4],
[2, 5], [2, 6], [3, 5], [3, 6], [3, 7], [4, 5], [4, 6], [5, 6], [5, 7],
[6, 7]]

print "\ncoboundary_2 =\n", csrToMatrixRepresentation(coboundary(CV,FV))
print "\ncoboundary_1 =\n", csrToMatrixRepresentation(coboundary(FV,EV))
print "\ncoboundary_0 =\n", csrToMatrixRepresentation(coboundary(EV,AA(LIST)(range(len(V)))))
◇
```

Macro referenced in 16b.

⟨ From cells and facets to boundary cells 8a ⟩ ≡

```
def zeroChain(cells):
   pass

def totalChain(cells):
   return csrCreate([[0] for cell in cells])

def boundaryCells(cells,facets):
   csrBoundaryMat = boundary(cells,facets)
   csrChain = totalChain(cells)
   csrBoundaryChain = matrixProduct(csrBoundaryMat, csrChain)
   for k,value in enumerate(csrBoundaryChain.data):
      if value % 2 == 0: csrBoundaryChain.data[k] = 0
   boundaryCells = [k for k,val in enumerate(csrBoundaryChain.data.tolist()) if val == 1]
   return boundaryCells
◇
```

Macro referenced in 16a.

⟨ Test examples of From cells and facets to boundary cells 8b ⟩ ≡

```
boundaryCells_2 = boundaryCells(CV,FV)
boundaryCells_1 = boundaryCells([FV[k] for k in boundaryCells_2],EV)

print "\nboundaryCells_2 =\n", boundaryCells_2
print "\nboundaryCells_1 =\n", boundaryCells_1

boundary = (V,[FV[k] for k in boundaryCells_2])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(boundary)))
◇
```

Macro referenced in 16b.

⟨ Signed boundary matrix for simplicial models 9a ⟩ ≡

```
def signedBoundary (V,CV,FV):
   # compute the set of pairs of indices to [boundary face,incident coface]
   coo = boundary(CV,FV).tocoo()
   pairs = [[coo.row[k],coo.col[k]] for k,val in enumerate(coo.data) if val != 0]

   # compute the [face, coface] pair as vertex lists
   vertLists = [[FV[pair[0]], CV[pair[1]]]for pair in pairs]

   # compute two n-cells to compare for sign
   cellPairs = [ [list(set(coface).difference(face))+face,coface]
               for face,coface in vertLists]

   # compute the local indices of missing boundary cofaces
   missingVertIndices = [ coface.index(list(set(coface).difference(face))[0])
                     for face,coface in vertLists]

   # compute the point matrices to compare for sign
   pointArrays = [ [[V[k]+[1.0] for k in facetCell], [V[k]+[1.0] for k in cofaceCell]]
               for facetCell,cofaceCell in cellPairs]

   # signed incidence coefficients
   cofaceMats = TRANS(pointArrays)[1]
   cofaceSigns = AA(SIGN)(AA(np.linalg.det)(cofaceMats))
   faceSigns = AA(C(POWER)(-1))(missingVertIndices)
   signPairProd = AA(PROD)(TRANS([cofaceSigns,faceSigns]))

   # signed boundary matrix
   csrSignedBoundaryMat = csr_matrix( (signPairProd,TRANS(pairs)) )
   return csrSignedBoundaryMat
```

◇

Macro referenced in 16a.

⟨ Oriented boundary cells for simplicial models 9b ⟩ ≡

```
def signedBoundaryCells(verts,cells,facets):
   csrBoundaryMat = signedBoundary(verts,cells,facets)
   csrTotalChain = totalChain(cells)
   csrBoundaryChain = matrixProduct(csrBoundaryMat, csrTotalChain)
   coo = csrBoundaryChain.tocoo()
   boundaryCells = list(coo.row * coo.data)
   return AA(int)(boundaryCells)
```

◇

Macro defined by 9b, 11.
Macro referenced in 16a.

Orienting polytopal cells

input : "cell" indices of a convex and solid polytopes and "V" vertices;

output : biggest "simplex" indices spanning the polytope.

`m` : number of cell vertices

`d` : dimension (number of coordinates) of cell vertices

`d+1` : number of simplex vertices

`vcell` : cell vertices

`vsimplex` : simplex vertices

`Id` : identity matrix

`basis` : orthonormal spanning set of vectors e_k

`vector` : position vector of a simplex vertex in translated coordinates

`unUsedIndices` : cell indices not moved to simplex

⟨ Oriented boundary cells for simplicial models 11 ⟩ ≡

```
def pivotSimplices(V,CV,d=3):
   simplices = []
   for cell in CV:
      vcell = np.array([V[v] for v in cell])
      m, simplex = len(cell), []
      # translate the cell: for each k, vcell[k] -= vcell[0], and simplex[0] := cell[0]
      for k in range(m-1,-1,-1): vcell[k] -= vcell[0]
      # simplex = [0], basis = [], tensor = Id(d+1)
      simplex += [cell[0]]
      basis = []
      tensor = np.array(IDNT(d))
      # look for most far cell vertex
      dists = [SUM([SQR(x) for x in v])**0.5 for v in vcell]
      maxDistIndex = max(enumerate(dists),key=lambda x: x[1])[0]
      vector = np.array([vcell[maxDistIndex]])
      # normalize vector
      den=(vector**2).sum(axis=-1) **0.5
      basis = [vector/den]
      simplex += [cell[maxDistIndex]]
      unUsedIndices = [h for h in cell if h not in simplex]

      # for k in {2,d+1}:
      for k in range(2,d+1):
         # update the orthonormal tensor
         e = basis[-1]
         tensor = tensor - np.dot(e.T, e)
         # compute the index h of a best vector
         # look for most far cell vertex
         dists = [SUM([SQR(x) for x in np.dot(tensor,v)])**0.5
         if h in unUsedIndices else 0.0
         for (h,v) in zip(cell,vcell)]
         # insert the best vector index h in output simplex
         maxDistIndex = max(enumerate(dists),key=lambda x: x[1])[0]
         vector = np.array([vcell[maxDistIndex]])
         # normalize vector
         den=(vector**2).sum(axis=-1) **0.5
         basis += [vector/den]
         simplex += [cell[maxDistIndex]]
         unUsedIndices = [h for h in cell if h not in simplex]
      simplices += [simplex]
   return simplices

def simplexOrientations(V,simplices):
   vcells = [[V[v]+[1.0] for v in simplex] for simplex in simplices]
   return [SIGN(np.linalg.det(vcell)) for vcell in vcells]
```
◇

Macro defined by 9b, 11.
Macro referenced in 16a.

⟨ Computation of cell adjacencies 12a ⟩ ≡

```
def larCellAdjacencies(CSRm):
    CSRm = matrixProduct(CSRm,csrTranspose(CSRm))
    return CSRm
◇
```

Macro referenced in 16a.

⟨ Test examples of Computation of cell adjacencies 12b ⟩ ≡

```
print "\n>>> larCellAdjacencies"
adj_2_cells = larCellAdjacencies(csrFV)
print "\nadj_2_cells =\n", csrToMatrixRepresentation(adj_2_cells)
adj_1_cells = larCellAdjacencies(csrEV)
print "\nadj_1_cells =\n", csrToMatrixRepresentation(adj_1_cells)
◇
```

Macro referenced in 16b.

⟨ Extraction of facets of a cell complex 13 ⟩ ≡

```
def setup(model,dim):
    V, cells = model
    csr = csrCreate(cells)
    csrAdjSquareMat = larCellAdjacencies(csr)
    csrAdjSquareMat = csrPredFilter(csrAdjSquareMat, GE(dim)) # ? HOWTODO ?
    return V,cells,csr,csrAdjSquareMat

def larFacets(model,dim=3):
    """
        Estraction of (d-1)-cellFacets from "model" := (V,d-cells)
        Return (V, (d-1)-cellFacets)
      """
    V,cells,csr,csrAdjSquareMat = setup(model,dim)
    cellFacets = []
    # for each input cell i
    for i in range(len(cells)):
        adjCells = csrAdjSquareMat[i].tocoo()
        cell1 = csr[i].tocoo().col
        pairs = zip(adjCells.col,adjCells.data)
        for j,v in pairs:
            if (i<j):
                cell2 = csr[j].tocoo().col
                cell = list(set(cell1).intersection(cell2))
                cellFacets.append(sorted(cell))
    # sort and remove duplicates
    cellFacets = sorted(AA(list)(set(AA(tuple)(cellFacets))))
    return V,cellFacets
```
◇

Macro referenced in 16a.

⟨ Test examples of Extraction of facets of a cell complex 14 ⟩ ≡

```
V = [[0.,0.],[3.,0.],[0.,3.],[3.,3.],[1.,2.],[2.,2.],[1.,1.],[2.,1.]]
FV = [[0,1,6,7],[0,2,4,6],[4,5,6,7],[1,3,5,7],[2,3,4,5],[0,1,2,3]]

_,EV = larFacets((V,FV),dim=2)
print "\nEV =",EV
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,EV))))

FV = [[0,1,3],[1,2,4],[2,4,5],[3,4,6],[4,6,7],[5,7,8], # full
   [1,3,4],[4,5,7], # empty
   [0,1,2],[6,7,8],[0,3,6],[2,5,8]] # exterior

_,EV = larFacets((V,FV),dim=2)
print "\nEV =",EV
◇
```

Macro referenced in 16b.

4 Exporting the library

4.1 MIT licence

⟨ The MIT Licence 15a ⟩ ≡

```
"""
The MIT License
===============

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
'Software'), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
"""
◇
```

Macro referenced in 16a.

4.2 Importing of modules or packages

⟨ Importing of modules or packages 15b ⟩ ≡

```
from pyplasm import *
import collections
import scipy
import numpy as np
from scipy import zeros,arange,mat,amin,amax
from scipy.sparse import vstack,hstack,csr_matrix,coo_matrix,lil_matrix,triu

from lar2psm import *
◇
```

Macro referenced in 16a.

4.3 Writing the library file

"lib/py/larcc.py" 16a ≡

```
# -*- coding: utf-8 -*-
""" Basic LARCC library """
⟨The MIT Licence 15a⟩
⟨Importing of modules or packages 15b⟩
⟨From list of triples to scipy.sparse 3a⟩
⟨Brc to Coo transformation 3b⟩
⟨Coo to Csr transformation 3d⟩
⟨Brc to Csr transformation 4b⟩
⟨Query Matrix shape 4d⟩
⟨Sparse to dense matrix transformation 5b⟩
⟨Matrix product and transposition 5d⟩
⟨Matrix filtering to produce the boundary matrix 6a⟩
⟨Matrix filtering via a generic predicate 6c⟩
⟨From cells and facets to boundary operator 7a⟩
⟨From cells and facets to boundary cells 8a⟩
⟨Signed boundary matrix for simplicial models 9a⟩
⟨Oriented boundary cells for simplicial models 9b, ... ⟩
⟨Computation of cell adjacencies 12a⟩
⟨Extraction of facets of a cell complex 13⟩

if __name__ == "__main__":
   ⟨Test examples 16b⟩
```
◇

5 Unit tests

⟨Test examples 16b⟩ ≡

```
⟨Test example of Brc to Coo transformation 3c⟩
⟨Test example of Coo to Csr transformation 4a⟩
⟨Test example of Brc to Csr transformation 4c⟩
⟨Test examples of Query Matrix shape 5a⟩
⟨Test examples of Sparse to dense matrix transformation 5c⟩
⟨Test example of Matrix filtering to produce the boundary matrix 6b⟩
⟨Test example of Matrix filtering via a generic predicate 6d⟩
⟨Test examples of From cells and facets to boundary operator 7b⟩
⟨Test examples of From cells and facets to boundary cells 8b⟩
⟨Test examples of Computation of cell adjacencies 12b⟩
⟨Test examples of Extraction of facets of a cell complex 14⟩
```
◇

Macro referenced in 16a.

A Appendix: Tutorials

A.1 Model generation, skeleton and boundary extraction

"test/py/larcc/ex1.py" 17a ≡

```
from larcc import *
from largrid import *
⟨input of 2D topology and geometry data 17b⟩
⟨characteristic matrices 17c⟩
⟨incidence matrix 17d⟩
⟨boundary and coboundary operators 18a⟩
⟨product of cell complexes 18b⟩
⟨2-skeleton extraction 18c⟩
⟨1-skeleton extraction 19a⟩
⟨0-coboundary computation 19b⟩
⟨1-coboundary computation 19c⟩
⟨2-coboundary computation 20a⟩
⟨boundary chain visualisation 20b⟩
◇
```

⟨input of 2D topology and geometry data 17b⟩ ≡

```
# input of topology and geometry
V2 = [[4,10],[8,10],[14,10],[8,7],[14,7],[4,4],[8,4],[14,4]]
EV = [[0,1],[1,2],[3,4],[5,6],[6,7],[0,5],[1,3],[2,4],[3,6],[4,7]]
FV = [[0,1,3,5,6],[1,2,3,4],[3,4,6,7]]
◇
```

Macro referenced in 17a.

⟨characteristic matrices 17c⟩ ≡

```
# characteristic matrices
csrFV = csrCreate(FV)
csrEV = csrCreate(EV)
print "\nFV =\n", csrToMatrixRepresentation(csrFV)
print "\nEV =\n", csrToMatrixRepresentation(csrEV)
◇
```

Macro referenced in 17a.

⟨incidence matrix 17d⟩ ≡

```
# product
csrEF = matrixProduct(csrEV, csrTranspose(csrFV))
print "\nEF =\n", csrToMatrixRepresentation(csrEF)
◇
```

Macro referenced in 17a.

⟨ boundary and coboundary operators 18a ⟩ ≡

```
# boundary and coboundary operators
facetLengths = [csrCell.getnnz() for csrCell in csrEV]
boundary = csrBoundaryFilter(csrEF,facetLengths)
coboundary_1 = csrTranspose(boundary)
print "\ncoboundary_1 =\n", csrToMatrixRepresentation(coboundary_1)
◇
```

Macro referenced in 17a.

⟨ product of cell complexes 18b ⟩ ≡

```
# product operator
mod_2D = (V2,FV)
V1,topol_0 = [[0.],[1.],[2.]], [[0],[1],[2]]
topol_1 = [[0,1],[1,2]]
mod_0D = (V1,topol_0)
mod_1D = (V1,topol_1)
V3,CV = larModelProduct([mod_2D,mod_1D])
mod_3D = (V3,CV)
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(mod_3D)))
print "\nk_3 =", len(CV), "\n"
◇
```

Macro referenced in 17a.

⟨ 2-skeleton extraction 18c ⟩ ≡

```
# 2-skeleton of the 3D product complex
mod_2D_1 = (V2,EV)
mod_3D_h2 = larModelProduct([mod_2D,mod_0D])
mod_3D_v2 = larModelProduct([mod_2D_1,mod_1D])
_,FV_h = mod_3D_h2
_,FV_v = mod_3D_v2
FV3 = FV_h + FV_v
SK2 = (V3,FV3)
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(SK2)))
print "\nk_2 =", len(FV3), "\n"
◇
```

Macro referenced in 17a.

⟨ 1-skeleton extraction 19a ⟩ ≡

```
# 1-skeleton of the 3D product complex
mod_2D_0 = (V2,AA(LIST)(range(len(V2))))
mod_3D_h1 = larModelProduct([mod_2D_1,mod_0D])
mod_3D_v1 = larModelProduct([mod_2D_0,mod_1D])
_,EV_h = mod_3D_h1
_,EV_v = mod_3D_v1
EV3 = EV_h + EV_v
SK1 = (V3,EV3)
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(SK1)))
print "\nk_1 =", len(EV3), "\n"
◇
```

Macro referenced in 17a.

⟨ 0-coboundary computation 19b ⟩ ≡

```
# boundary and coboundary operators
np.set_printoptions(threshold=sys.maxint)
csrFV3 = csrCreate(FV3)
csrEV3 = csrCreate(EV3)
csrVE3 = csrTranspose(csrEV3)
facetLengths = [csrCell.getnnz() for csrCell in csrEV3]
boundary = csrBoundaryFilter(csrVE3,facetLengths)
coboundary_0 = csrTranspose(boundary)
print "\ncoboundary_0 =\n", csrToMatrixRepresentation(coboundary_0)
◇
```

Macro referenced in 17a.

⟨ 1-coboundary computation 19c ⟩ ≡

```
csrEF3 = matrixProduct(csrEV3, csrTranspose(csrFV3))
facetLengths = [csrCell.getnnz() for csrCell in csrFV3]
boundary = csrBoundaryFilter(csrEF3,facetLengths)
coboundary_1 = csrTranspose(boundary)
print "\ncoboundary_1.T =\n", csrToMatrixRepresentation(coboundary_1.T)
◇
```

Macro referenced in 17a.

⟨ 2-coboundary computation 20a ⟩ ≡

```
csrCV = csrCreate(CV)
csrFC3 = matrixProduct(csrFV3, csrTranspose(csrCV))
facetLengths = [csrCell.getnnz() for csrCell in csrCV]
boundary = csrBoundaryFilter(csrFC3,facetLengths)
coboundary_2 = csrTranspose(boundary)
print "\ncoboundary_2 =\n", csrToMatrixRepresentation(coboundary_2)
◇
```

Macro referenced in 17a.

⟨ boundary chain visualisation 20b ⟩ ≡

```
# boundary chain visualisation
boundaryCells_2 = boundaryCells(CV,FV3)
boundary = (V3,[FV3[k] for k in boundaryCells_2])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(boundary)))
◇
```

Macro referenced in 17a.

A.2 Boundary of 3D simplicial grid

"test/py/larcc/ex2.py" 20c ≡

```
⟨ boundary of 3D simplicial grid 20d ⟩
◇
```

⟨ boundary of 3D simplicial grid 20d ⟩ ≡

```
from simplexn import *
from larcc import *

V,CV = larSimplexGrid([10,10,3])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,CV))))
SK2 = (V,larSimplexFacets(CV))
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(SK2)))
_,FV = SK2
SK1 = (V,larSimplexFacets(FV))
_,EV = SK1
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(SK1)))

boundaryCells_2 = boundaryCells(CV,FV)
boundary = (V,[FV[k] for k in boundaryCells_2])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(boundary)))
print "\nboundaryCells_2 =\n", boundaryCells_2
◇
```

Macro referenced in 20c.

A.3 Oriented boundary of a random simplicial complex

"test/py/larcc/ex3.py" 21a ≡

⟨Importing external modules 21b⟩
⟨Generating and viewing a random 3D simplicial complex 21c⟩
⟨Computing and viewing its non-oriented boundary 21d⟩
⟨Computing and viewing its oriented boundary 22a⟩
◇

⟨Importing external modules 21b⟩ ≡

```
from simplexn import *
from larcc import *
from scipy.spatial import Delaunay
import numpy as np
```
◇

Macro referenced in 21a.

⟨Generating and viewing a random 3D simplicial complex 21c⟩ ≡

```
verts = np.random.rand(10000, 3) # 1000 points in 3-d
verts = [AA(lambda x: 2*x)(VECTDIFF([vert,[0.5,0.5,0.5]])) for vert in verts]
verts = [vert for vert in verts if VECTNORM(vert) < 1.0]
tetra = Delaunay(verts)
cells = [cell for cell in tetra.vertices.tolist()
         if  ((verts[cell[0]][2]<0) and (verts[cell[1]][2]<0)
               and (verts[cell[2]][2]<0) and (verts[cell[3]][2]<0) ) ]
V, CV = verts, cells
VIEW(MKPOL([V,AA(AA(lambda k:k+1))(CV),[]]))
```
◇

Macro referenced in 21a.

⟨Computing and viewing its non-oriented boundary 21d⟩ ≡

```
FV = larSimplexFacets(CV)
VIEW(MKPOL([V,AA(AA(lambda k:k+1))(FV),[]]))
boundaryCells_2 = boundaryCells(CV,FV)
print "\nboundaryCells_2 =\n", boundaryCells_2
bndry = (V,[FV[k] for k in boundaryCells_2])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(bndry)))
```
◇

Macro referenced in 21a.

⟨ Computing and viewing its oriented boundary 22a ⟩ ≡

```
boundaryCells_2 = signedBoundaryCells(V,CV,FV)
print "\nboundaryCells_2 =\n", boundaryCells_2
def swap(mylist): return [mylist[1]]+[mylist[0]]+mylist[2:]
boundaryFV = [FV[-k] if k<0 else swap(FV[k]) for k in boundaryCells_2]
bndry = (V,boundaryFV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(bndry)))
◇
```

Macro referenced in 21a.

A.4 Oriented boundary of a simplicial grid

"test/py/larcc/ex4.py" 22b ≡

```
⟨ Generate and view a 3D simplicial grid 22c ⟩
⟨ Computing and viewing the 2-skeleton of simplicial grid 22d ⟩
⟨ Computing and viewing the oriented boundary of simplicial grid 22e ⟩
◇
```

⟨ Generate and view a 3D simplicial grid 22c ⟩ ≡

```
from simplexn import *
from larcc import *
V,CV = larSimplexGrid([4,4,4])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,CV))))
◇
```

Macro referenced in 22b.

⟨ Computing and viewing the 2-skeleton of simplicial grid 22d ⟩ ≡

```
FV = larSimplexFacets(CV)
EV = larSimplexFacets(FV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,FV))))
◇
```

Macro referenced in 22b.

⟨ Computing and viewing the oriented boundary of simplicial grid 22e ⟩ ≡

```
csrSignedBoundaryMat = signedBoundary (V,CV,FV)
boundaryCells_2 = signedBoundaryCells(V,CV,FV)
def swap(l): return [l[1],l[0],l[2]]
boundaryFV = [FV[-k] if k<0 else swap(FV[k]) for k in boundaryCells_2]
boundary = (V,boundaryFV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(boundary)))
◇
```

Macro referenced in 22b.

A.5 Skeletons and oriented boundary of a simplicial complex

"test/py/larcc/ex5.py" 23a ≡

⟨ Skeletons computation and vilualisation 23b ⟩
⟨ Oriented boundary matrix visualization 23c ⟩
⟨ Computation of oriented boundary cells 23d ⟩
◇

⟨ Skeletons computation and vilualisation 23b ⟩ ≡

```
from simplexn import *
from larcc import *
V,FV = larSimplexGrid([3,3])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,FV))))
EV = larSimplexFacets(FV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,EV))))
VV = larSimplexFacets(EV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,VV))))
```
◇

Macro referenced in 23a.

⟨ Oriented boundary matrix visualization 23c ⟩ ≡

```
np.set_printoptions(threshold='nan')
csrSignedBoundaryMat = signedBoundary (V,FV,EV)
Z = csrToMatrixRepresentation(csrSignedBoundaryMat)
print "\ncsrSignedBoundaryMat =\n", Z
from pylab import *
matshow(Z)
show()
```
◇

Macro referenced in 23a.

⟨ Computation of oriented boundary cells 23d ⟩ ≡

```
boundaryCells_1 = signedBoundaryCells(V,FV,EV)
print "\nboundaryCells_1 =\n", boundaryCells_1
def swap(mylist): return [mylist[1]]+[mylist[0]]+mylist[2:]
boundaryEV = [EV[-k] if k<0 else swap(EV[k]) for k in boundaryCells_1]
bndry = (V,boundaryEV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(bndry)))
```
◇

Macro referenced in 23a.

A.6 Boundary of random 2D simplicial complex

"test/py/larcc/ex6.py" 24a ≡

```
from simplexn import *
from larcc import *
from scipy.spatial import Delaunay
⟨Test for quasi-equilateral triangles 24b⟩
⟨Generation and selection of random triangles 25a⟩
⟨Boundary computation and visualisation 25b⟩
◇
```

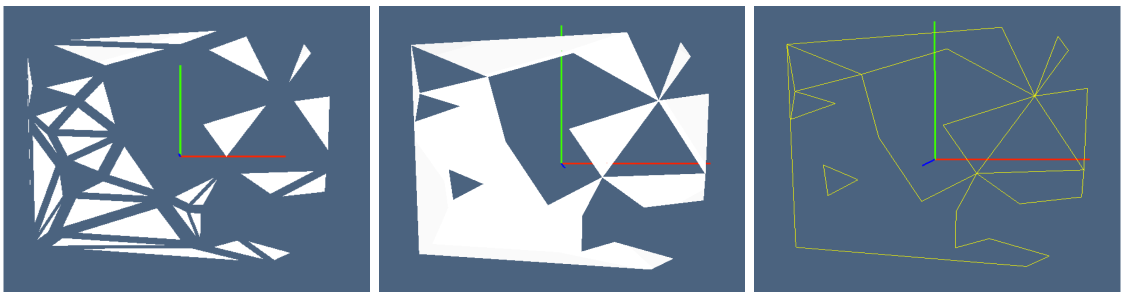

Figure 1: example caption

⟨Test for quasi-equilateral triangles 24b⟩ ≡

```
def quasiEquilateral(tria):
    a = VECTNORM(VECTDIFF(tria[0:2]))
    b = VECTNORM(VECTDIFF(tria[1:3]))
    c = VECTNORM(VECTDIFF([tria[0],tria[2]]))
    m = max(a,b,c)
    if m/a < 1.7 and m/b < 1.7 and m/c < 1.7: return True
    else: return False
◇
```

Macro referenced in 24a.

⟨ Generation and selection of random triangles 25a ⟩ ≡

```
verts = np.random.rand(20,2)
verts = (verts - [0.5,0.5]) * 2
triangles = Delaunay(verts)
cells = [ cell for cell in triangles.vertices.tolist()
          if (not quasiEquilateral([verts[k] for k in cell])) ]
V, FV = AA(list)(verts), cells
EV = larSimplexFacets(FV)
pols2D = MKPOLS((V,FV))
VIEW(EXPLODE(1.5,1.5,1.5)(pols2D))
```

◇

Macro referenced in 24a.

⟨ Boundary computation and visualisation 25b ⟩ ≡

```
boundaryCells_1 = signedBoundaryCells(V,FV,EV)
print "\nboundaryCells_1 =\n", boundaryCells_1
def swap(mylist): return [mylist[1]]+[mylist[0]]+mylist[2:]
boundaryEV = [EV[-k] if k<0 else swap(EV[k]) for k in boundaryCells_1]
bndry = (V,boundaryEV)
VIEW(STRUCT(MKPOLS(bndry) + pols2D))
VIEW(COLOR(RED)(STRUCT(MKPOLS(bndry))))
```

◇

Macro referenced in 24a.

⟨ Compute the topologically ordered chain of boundary vertices 25c ⟩ ≡

◇

Macro never referenced.

⟨Decompose a permutation into cycles 26a⟩ ≡

```
def permutationOrbits(List):
   d = dict((i,int(x)) for i,x in enumerate(List))
   out = []
   while d:
      x = list(d)[0]
      orbit = []
      while x in d:
         orbit += [x],
         x = d.pop(x)
      out += [CAT(orbit)+orbit[0]]
   return out

if __name__ == "__main__":
   print [2, 3, 4, 5, 6, 7, 0, 1]
   print permutationOrbits([2, 3, 4, 5, 6, 7, 0, 1])
   print [3,9,8,4,10,7,2,11,6,0,1,5]
   print permutationOrbits([3,9,8,4,10,7,2,11,6,0,1,5])
```

◇

Macro never referenced.

A.7 Assemblies of simplices and hypercubes

"test/py/larcc/ex7.py" 26b ≡

```
from simplexn import *
from larcc import *
from largrid import *
```

⟨Definition of 1-dimensional LAR models 27a⟩
⟨Assembly generation of squares and triangles 27b⟩
⟨Assembly generation of cubes and tetrahedra 27c⟩
◇

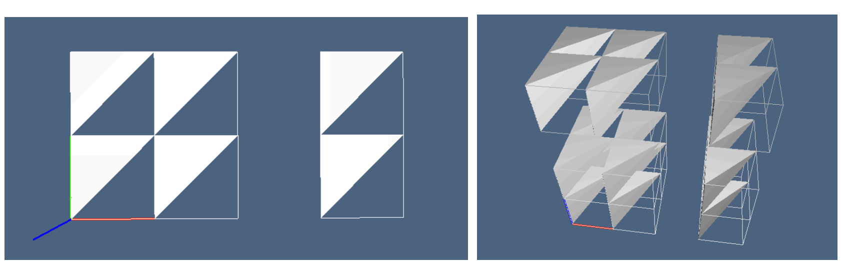

Figure 2: (a) Assemblies of squares and triangles; (b) assembly of cubes and tetrahedra.

⟨ Definition of 1-dimensional LAR models 27a ⟩ ≡

```
geom_0,topol_0 = [[0.],[1.],[2.],[3.],[4.]],[[0,1],[1,2],[3,4]]
geom_1,topol_1 = [[0.],[1.],[2.]], [[0,1],[1,2]]
mod_0 = (geom_0,topol_0)
mod_1 = (geom_1,topol_1)
```

◇

Macro referenced in 26b.

⟨ Assembly generation of squares and triangles 27b ⟩ ≡

```
squares = larModelProduct([mod_0,mod_1])
V,FV = squares
simplices = pivotSimplices(V,FV,d=2)
VIEW(STRUCT([ MKPOL([V,AA(AA(C(SUM)(1)))(simplices),[]]),
              SKEL_1(STRUCT(MKPOLS((V,FV)))) ]))
```

◇

Macro referenced in 26b.

⟨ Assembly generation of cubes and tetrahedra 27c ⟩ ≡

```
cubes = larModelProduct([squares,mod_0])
V,CV = cubes
simplices = pivotSimplices(V,CV,d=3)
VIEW(STRUCT([ MKPOL([V,AA(AA(C(SUM)(1)))(simplices),[]]),
           SKEL_1(STRUCT(MKPOLS((V,CV)))) ]))
```

◇

Macro referenced in 26b.

References

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