$Simple^n_X$ module

IN480 exam [1]

Fabio Fatelli Emanuele Loprevite

June 9, 2018

Contents

1 Introduction

The $Simple_X^n$ library, named **simplexn** within the Python version of the LARCC framework, provides combinatorial algorithms for some basic functions of geometric modelling with simplicial complexes. In particular, provides the efficient creation of simplicial complexes generated by simplicial complexes of lower dimension, the production of simplicial grids of any dimension, and the extraction of facets (i.e. of $(d-1)$-faces) of complexes of d-simplices. [2]

1.1 Requirements

There is no need to install extra packages to run the Julia code, it is sufficient loading the built-in package *Combinatorics.jl* (i.e. `using Combinatorics`); however, in order to be able to make the graphs it is necessary to install the package *Plots.jl* [5] (i.e. `Pkg.add("Plots")`).

We used four processors (`addprocs(4)` at startup), five REPL included.

1.2 API

`larExtrude1(model::Tuple{Array{Array{Int64,1},1},Array{Array{Int64,1},1}}, pattern::Array{Int64,1})`
Description: this function generates the output model vertices in a multiple extrusion of a LAR model.
Input: `model` contains a pair (V, FV), where V is the array of input vertices, and FV is the array of d-cells (given as arrays of vertex indices) providing the input representation of a LAR cellular complex. `pattern` is an array of integers, whose absolute values provide the sizes of the ordered set of 1D (in local coords) subintervals specified by the pattern itself.
Output: it is a model that contains a pair (outV, triangles) representing the triangulation of the input `model`.

`larSimplexGrid1(shape::Array{Int64,1})`
Description: this function generates the simplicial grids of any dimension and shape.
Input: `shape` is an array of integers used to specify the shape of the created array.
Output: it is a model that contains a pair (V, FV), where V is the array of input vertices, and FV is the array of d-cells (given as arrays of vertex indices) providing the input representation of a LAR cellular complex.

`larSimplexFacets(simplices::Array{Array{Int64,1},1})`
Description: this function provides the extraction of non-oriented $(d-1)$-facets of d-dimensional simplices.
Input: `simplices` is the array of d-cells (given as arrays of vertex indices) providing the input representation of a LAR cellular complex.
Output: it is an array of d-cells of integers, i.e. the input LAR representation of the topology of a cellular complex.

`quads2tria(model::Tuple{Array{Array{Int64,1},1},Array{Array{Int64,1},1}})`
Description: this function gives the conversion of a LAR boundary representation (B-Rep), i.e. a LAR model **V, FV** made of 2D faces, usually quads but also general polygons, into a LAR model **VERTS, TRIANGLES** made by triangles.
Input: `model` contains a pair (V, FV), where V is the array of input vertices, and FV is the array of *d*-cells (given as arrays of vertex indices) providing the input representation of a LAR cellular complex.
Output: it is a model that contains a pair (V, triangles) representing the triangulation of the input `model`.

2 Implementation

All the code in this section works with a simple copy and paste in Julia REPL; however, if a code block starts in a page and ends at the following one, it is required to pay attention at the numbers and headers of the pages.

Moreover, we left in the code some comments we think could be useful.

2.1 larExtrude1

This function generates the output model vertices in a multiple extrusion of a LAR model.

2.1.1 Python code

```
def larExtrude1(model,pattern):
    V, FV = model
    d, m = len(FV[0]), len(pattern)
    coords = list(cumsum([0]+(AA(ABS)(pattern))))
    offset, outcells, rangelimit = len(V), [], d*m
    for cell in FV:
        tube = [v + k*offset for k in range(m+1) for v in cell]
        cellTube = [tube[k:k+d+1] for k in range(rangelimit)]
        outcells += [reshape(cellTube, newshape=(m,d,d+1)).tolist()]

    outcells = AA(CAT)(TRANS(outcells))
    cellGroups = [group for k,group in enumerate(outcells) if pattern[k]>0]
    outVertices = [v+[z] for z in coords for v in V]
    outModel = outVertices, CAT(cellGroups)
    return outModel
```

2.1.2 Julia code - serial

```
# Generation of the output model vertices in a multiple extrusion of a LAR model
function larExtrude1(model::Tuple{Array{Array{Int64,1},1},Array{Array{Int64,1},1}},
  pattern::Array{Int64,1})
```

```
    V, FV = model
    d, m = length(FV[1]), length(pattern)
    coords = cumsum(append!([0],abs.(pattern))) # built-in function cumsum
    offset, outcells, rangelimit = length(V), Array{Int64}(m,0), d*m
    for cell in FV
        tube = [v + k*offset for k in 0:m for v in cell]
        celltube = Int64[]
        for k in 1:rangelimit
            append!(celltube,tube[k:k+d])
        end
        outcells = hcat(outcells,permutedims(reshape(celltube,d*(d+1),m),[2,1]))
    end
    cellGroups = Int64[]
    for k in 1:m
        if pattern[k]>0
            cellGroups = vcat(cellGroups,outcells[k,:])
        end
    end
    outVertices = [vcat(v,z) for z in coords for v in V]
    outCellGroups = Array{Int64,1}[]
    for k in 1:d+1:length(cellGroups)
        append!(outCellGroups,[cellGroups[k:k+d]])
    end
    return outVertices, outCellGroups
end
```

2.1.3 Parallel optimization

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.1.4 Julia code - parallel

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus.

Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2.1.5 Unit test code

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

2.1.6 Script

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

2.2 larSimplexGrid1

This function generates the simplicial grids of any dimension and shape.

2.2.1 Python code

```
def larSimplexGrid1(shape):
    model = VOID
    for item in shape:
        model = larExtrude1(model,item*[1])
    return model
```

2.2.2 Julia code - serial

```
# Generation of simplicial grids of any dimension and shape
function larSimplexGrid1(shape::Array{Int64,1})
    model = [Int64[]],[[0]] # the empty simplicial model
    for item in shape
        model = larExtrude1(model,repmat([1],item))
    end
    return model
end
```

2.2.3 Parallel optimization

It is not possible to parallelize this function because every iteration of the loop requires the `model` that is computed in the previous one. The only difference here is the addition of `@everywhere`.

2.2.4 Julia code - parallel

```
# Generation of simplicial grids of any dimension and shape
@everywhere function larSimplexGrid1(shape::Array{Int64,1})
    model = V0,CV0 = [Int64[]],[[0]] # the empty simplicial model
    for item in shape # no parallelism
        model = larExtrude1(model,repmat([1],item))
    end
    return model
end
```

2.2.5 Unit test code

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

2.2.6 Script

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

2.3 larSimplexFacets

This function provides the extraction of non-oriented $(d-1)$-facets of d-dimensional simplices.

2.3.1 Python code

```
def larSimplexFacets(simplices):
    out = []
```

```
    d = len(simplices[0])
    for simplex in simplices:
        out += AA(sorted)([simplex[0:k]+simplex[k+1:d] for k in range(d)])
    out = set(AA(tuple)(out))
    return  sorted(out)
```

2.3.2 Julia code - serial

```
# Extraction of non-oriented (d-1)-facets of d-dimensional simplices
using Combinatorics # for combinations() function

function larSimplexFacets(simplices::Array{Array{Int64,1},1})
    out = Array{Int64,1}[]
    d = length(simplices[1])
    for simplex in simplices
        append!(out,collect(combinations(simplex,d-1)))
    end
    return sort!(unique(out),lt=lexless) # array of arrays, not of tuples
end
#map(x->tuple(x...),[[0, 1],[0, 4],[1, 2]])
```

2.3.3 Parallel optimization

Here, other than the classic addition of `@everywhere`, we used `@parallel` to split the computation of the `for` among multiple processors. The `return` automatically waits the end of the computation.

2.3.4 Julia code - parallel

```
# Extraction of non-oriented (d-1)-facets of d-dimensional simplices
@everywhere using Combinatorics # for combinations() function

@everywhere function larSimplexFacets(simplices::Array{Array{Int64,1},1})
    out = Array{Int64,1}[]
    d = length(simplices[1])
    out = @parallel (append!) for simplex in simplices
            collect(combinations(simplex,d-1))
        end
    return sort!(unique(out),lt=lexless) # array of arrays, not of tuples
end
#map(x->tuple(x...),[[0, 1],[0, 4],[1, 2]])
```

2.3.5 Unit test code

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

2.3.6 Script

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

2.4 quads2tria

This function gives the conversion of a LAR boundary representation (B-Rep), i.e. a LAR model **V, FV** made of 2D faces, usually quads but also general polygons, into a LAR model **VERTS, TRIANGLES** made by triangles.

2.4.1 Python code

```
def quads2tria(model):
   V,FV = model
   out = []
   nverts = len(V)-1
   for face in FV:
      centroid = CCOMB([V[v] for v in face])
      V += [centroid]
      nverts += 1

      v1, v2 = DIFF([V[face[0]],centroid]), DIFF([V[face[1]],centroid])
      v3 = VECTPROD([v1,v2])
      if ABS(VECTNORM(v3)) < 10**3:
         v1, v2 = DIFF([V[face[0]],centroid]), DIFF([V[face[2]],centroid])
         v3 = VECTPROD([v1,v2])
      transf = mat(INV([v1,v2,v3]))
      verts = [(V[v]*transf).tolist()[0][:-1]  for v in face]

      tcentroid = CCOMB(verts)
      tverts = [DIFF([v,tcentroid]) for v in verts]
      rverts = sorted([[ATAN2(vert),v] for vert,v in zip(tverts,face)])
      ord = [pair[1] for pair in rverts]
```

```
        ord = ord + [ord[0]]
        edges = [[n,ord[k+1]] for k,n in enumerate(ord[:-1])]
        triangles = [[nverts] + edge for edge in edges]
        out += triangles
    return V,out
```

2.4.2 Julia code - serial

```
# Transformation to triangles by sorting circularly the vertices of faces
function quads2tria(model::Tuple{Array{Array{Float64,1},1},Array{Array{Int64,1},1}})
    V, FV = model
    out = Array{Int64,1}[]
    nverts = length(V)-1
    for face in FV
        arr = [V[v+1] for v in face]
        centroid = sum(arr)/length(arr)
        append!(V,[centroid])
        nverts += 1
        v1, v2 = V[face[1]+1]-centroid, V[face[2]+1]-centroid
        v3 = cross(v1,v2)
        if norm(v3) < 1/(10^3)
            v1, v2 = V[face[1]+1]-centroid, V[face[3]+1]-centroid
            v3 = cross(v1,v2)
        end
        transf = inv(hcat(v1,v2,v3)')
        verts = [(V[v+1]'*transf)'[1:end-1] for v in face]
        tcentroid = sum(verts)/length(verts)
        tverts = [v-tcentroid for v in verts]
        iterator = collect(zip(tverts, face))
        rverts = [[atan2(reverse(iterator[i][1])...),iterator[i][2]]
          for i in 1:length(iterator)]
        rvertss = sort(rverts,lt=(x,y)->isless(x[1],y[1]))
        ord = [pair[2] for pair in rvertss]
        append!(ord,ord[1])
        edges = [[i[2],ord[i[1]+1]] for i in enumerate(ord[1:end-1])]
        triangles = [prepend!(edge,nverts) for edge in edges]
        append!(out,triangles)
    end
    return V, out
end
```

2.4.3 Parallel optimization

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.4.4 Julia code - parallel

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

2.4.5 Unit test code

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

2.4.6 Script

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

3 Conclusions

The graphs have showed the parallel code is (significantly) slower than the serial one. A possible way to improve this problem could be to rewrite all the functions using different structures and procedures to handle the data, avoiding array of arrays and similar.

However, the complete lack of documentation online, official and non, for the correct use of the macros and how they specifically work makes the task quite difficult.

References

[1] IN480 course web page.

[2] Python $Simple_X^n$ module pdf.

[3] A. Paoluzzi, *Geometric Programming for Computer-Aided Design*, Wiley, 2003.

[4] Official Julia website.

[5] Documentation of the Plots package for Julia.

[6] M. Sherrington, *Mastering Julia*, Packt Publishing, 2015.