Curves, surfaces and splines with LAR *

Alberto Paoluzzi

April 24, 2014

Abstract

In this module we implement above LAR most of the parametric methods for polynomial and rational curves, surfaces and splines discussed in the book [Pao03], and implemented in the PLaSM language and in the python package pyplasm.

Contents

1 Introduction

2 Tensor product surfaces

The tensor product form of surfaces will be primarily used, in the remainder of this module, to support the LAR implementation of polynomial (rational) surfaces. For this purpose,

*This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework [CL13]. April 24, 2014 we start by defining some basic operators on function tensors. In particular, a toolbox of basic tensor operations is given in Script 12.3.1. The ConstFunTensor operator produces a tensor of constant functions starting from a tensor of numbers; the recursive FlatTensor may be used to ?flatten? a tensor with any number of indices by producing a corresponding one index tensor; the InnerProd and TensorProd are used to compute the inner product and the tensor product of conforming tensors of functions, respectively.

Toolbox of tensor operations

⟨ Multidimensional transfinite Bernstein-Bezier Basis 1 ⟩ ≡

```
""" Toolbox of tensor operations """
def larBernsteinBasis (U):
   def BERNSTEIN0 (N):
      def BERNSTEIN1 (I):
         def map_fn(point):
            t = U(point)
            out = CHOOSE([N,I])*math.pow(1-t,N-I)*math.pow(t,I)
            return out
         return map_fn
      return [BERNSTEIN1(I) for I in range(0,N+1)]
   return BERNSTEIN0
◇
```

Macro referenced in 4b.

2.1 Tensor product surface patch

⟨ Tensor product surface patch 2a ⟩ ≡

```
""" Tensor product surface patch """
def larTensorProdSurface (args):
   ubasis , vbasis = args
   def TENSORPRODSURFACE0 (controlpoints_fn):
      def map_fn(point):
         u,v=point
         U=[f([u]) for f in ubasis]
         V=[f([v]) for f in vbasis]
         controlpoints=[f(point) if callable(f) else f
            for f in controlpoints_fn]
         target_dim = len(controlpoints[0][0])
         ret=[0 for x in range(target_dim)]
         for i in range(len(ubasis)):
            for j in range(len(vbasis)):
               for M in range(len(ret)):
                  for M in range(target_dim):
                     ret[M] += U[i]*V[j] * controlpoints[i][j][M]
         return ret
```

```
        return map_fn
    return TENSORPRODSURFACE0
◇
```

Macro referenced in 4b.

Bilinear tensor product surface patch

⟨ Bilinear surface patch 2b ⟩ ≡

```
""" Bilinear tensor product surface patch """
def larBilinearSurface(controlpoints):
    basis = larBernsteinBasis(S1)(1)
    return larTensorProdSurface([basis,basis])(controlpoints)
◇
```

Macro referenced in 4b.

Biquadratic tensor product surface patch

⟨ Biquadratic surface patch 3a ⟩ ≡

```
""" Biquadratic tensor product surface patch """
def larBiquadraticSurface(controlpoints):
    basis1 = larBernsteinBasis(S1)(2)
    basis2 = larBernsteinBasis(S1)(2)
    return larTensorProdSurface([basis1,basis2])(controlpoints)
◇
```

Macro referenced in 4b.

Bicubic tensor product surface patch

⟨ Bicubic surface patch 3b ⟩ ≡

```
""" Bicubic tensor product surface patch """
def larBicubicSurface(controlpoints):
    basis1 = larBernsteinBasis(S1)(3)
    basis2 = larBernsteinBasis(S1)(3)
    return larTensorProdSurface([basis1,basis2])(controlpoints)
◇
```

Macro referenced in 4b.

3 Transfinite Bézier

⟨ Multidimensional transfinite Bézier 3c ⟩ ≡

```
""" Multidimensional transfinite Bezier """
def larBezier(U):
   def BEZIER0(controldata_fn):
      N = len(controldata_fn)-1
      def map_fn(point):
         t = U(point)
         controldata = [fun(point) if callable(fun) else fun
            for fun in controldata_fn]
         out = [0.0 for i in range(len(controldata[0]))]
         for I in range(N+1):
            weight = CHOOSE([N,I])*math.pow(1-t,N-I)*math.pow(t,I)
            for K in range(len(out)):  out[K] += weight*(controldata[I][K])
         return out
      return map_fn
   return BEZIER0

def larBezierCurve(controlpoints):
   return larBezier(S1)(controlpoints)
```
◇

Macro referenced in 4b.

4 Coons patches

⟨ Transfinite Coons patches 4a ⟩ ≡

```
""" Transfinite Coons patches """
def larCoonsPatch (args):
   su0_fn , su1_fn , s0v_fn , s1v_fn = args
   def map_fn(point):
      u,v=point
      su0 = su0_fn(point) if callable(su0_fn) else su0_fn
      su1 = su1_fn(point) if callable(su1_fn) else su1_fn
      s0v = s0v_fn(point) if callable(s0v_fn) else s0v_fn
      s1v = s1v_fn(point) if callable(s1v_fn) else s1v_fn
      ret=[0.0 for i in range(len(su0))]
      for K in range(len(ret)):
         ret[K] = ((1-u)*s0v[K] + u*s1v[K]+(1-v)*su0[K] + v*su1[K] +
         (1-u)*(1-v)*s0v[K] + (1-u)*v*s0v[K] + u*(1-v)*s1v[K] + u*v*s1v[K])
      return ret
   return map_fn
```
◇

Macro referenced in 4b.

5 Computational framework

5.1 Exporting the library

"lib/py/splines.py" 4b ≡

```
""" Mapping functions and primitive objects """
⟨Initial import of modules 8⟩
⟨Tensor product surface patch 2a⟩
⟨Bilinear surface patch 2b⟩
⟨Biquadratic surface patch 3a⟩
⟨Bicubic surface patch 3b⟩
⟨Multidimensional transfinite Bernstein-Bezier Basis 1⟩
⟨Multidimensional transfinite Bézier 3c⟩
⟨Transfinite Coons patches 4a⟩
◇
```

6 Examples

Examples of larBernsteinBasis generation

⟨Examples of larBernsteinBasis 5a⟩ ≡

```
larBernsteinBasis(S1)(3)
""" [<function __main__.map_fn>,
   <function __main__.map_fn>,
   <function __main__.map_fn>,
   <function __main__.map_fn>] """
larBernsteinBasis(S1)(3)[0]
""" <function __main__.map_fn> """
larBernsteinBasis(S1)(3)[0]([0.0])
""" 1.0 """
◇
```

Macro never referenced.

Graph of Bernstein-Bezier basis

"test/py/splines/test04.py" 5b ≡

```
""" Graph of Bernstein-Bezier basis """
import sys
""" import modules from larcc/lib """
sys.path.insert(0, 'lib/py/')
from splines import *

def larBezierBasisGraph(degree):
   basis = larBernsteinBasis(S1)(degree)
   dom = larDomain([32])
   graphs = CONS(AA(larMap)(DISTL([S1, basis])))(dom)
```

```
    return graphs

graphs = larBezierBasisGraph(4)
VIEW(STRUCT( CAT(AA(MKPOLS)( graphs )) ))
```
◇

Some examples of curves

"test/py/splines/test01.py" 5c ≡

```
""" Example of Bezier curve """
import sys
""" import modules from larcc/lib """
sys.path.insert(0, 'lib/py/')
from splines import *

controlpoints = [[-0,0],[1,0],[1,1],[2,1],[3,1]]
dom = larDomain([32])
obj = larMap(larBezierCurve(controlpoints))(dom)
VIEW(STRUCT(MKPOLS(obj)))

obj = larMap(larBezier(S1)(controlpoints))(dom)
VIEW(STRUCT(MKPOLS(obj)))
```
◇

Transfinite cubic surface

"test/py/splines/test02.py" 6a ≡

```
""" Example of transfinite surface """
import sys
""" import modules from larcc/lib """
sys.path.insert(0, 'lib/py/')
from splines import *

dom = larDomain([20],'simplex')
C0 = larBezier(S1)([[0,0,0],[10,0,0]])
C1 = larBezier(S1)([[0,2,0],[8,3,0],[9,2,0]])
C2 = larBezier(S1)([[0,4,1],[7,5,-1],[8,5,1],[12,4,0]])
C3 = larBezier(S1)([[0,6,0],[9,6,3],[10,6,-1]])
dom2D = larExtrude1(dom,20*[1./20])
obj = larMap(larBezier(S2)([C0,C1,C2,C3]))(dom2D)
VIEW(STRUCT(MKPOLS(obj)))
```
◇

Coons patch interpolating 4 boundary curves

"test/py/splines/test03.py" 6b ≡

```
""" Example of transfinite Coons surface """
import sys
""" import modules from larcc/lib """
sys.path.insert(0, 'lib/py/')
from splines import *
Su0 = larBezier(S1)([[0,0,0],[10,0,0]])
Su1 = larBezier(S1)([[0,10,0],[2.5,10,3],[5,10,-3],[7.5,10,3],[10,10,0]])
Sv0 = larBezier(S2)([[0,0,0],[0,0,3],[0,10,3],[0,10,0]])
Sv1 = larBezier(S2)([[10,0,0],[10,5,3],[10,10,0]])
dom = larDomain([20])
dom2D = larExtrude1(dom, 20*[1./20])
out = larMap(larCoonsPatch([Su0,Su1,Sv0,Sv1]))(dom2D)
VIEW(STRUCT(MKPOLS(out)))
◇
```

Bilinear tensor product patch

"test/py/splines/test05.py" 6c ≡

```
""" Example of bilinear tensor product surface patch """
import sys
""" import modules from larcc/lib """
sys.path.insert(0, 'lib/py/')
from splines import *

controlpoints = [
   [[0,0,0],[2,-4,2]],
   [[0,3,1],[4,0,0]]]
dom = larDomain([20])
dom2D = larExtrude1(dom, 20*[1./20])
mapping = larBilinearSurface(controlpoints)
patch = larMap(mapping)(dom2D)
VIEW(STRUCT(MKPOLS(patch)))
◇
```

Biquadratic tensor product patch

"test/py/splines/test06.py" 7a ≡

```
""" Example of bilinear tensor product surface patch """
import sys
""" import modules from larcc/lib """
sys.path.insert(0, 'lib/py/')
from splines import *

controlpoints=[
   [[0,0,0],[2,0,1],[3,1,1]],
   [[1,3,-1],[2,2,0],[3,2,0]],
```

```
   [[-2,4,0],[2,5,1],[1,3,2]]]
dom = larDomain([20])
dom2D = larExtrude1(dom, 20*[1./20])
mapping = larBiquadraticSurface(controlpoints)
patch = larMap(mapping)(dom2D)
VIEW(STRUCT(MKPOLS(patch)))
◇
```

Bicubic tensor product patch

"test/py/splines/test07.py" 7b ≡

```
""" Example of bilinear tensor product surface patch """
import sys
""" import modules from larcc/lib """
sys.path.insert(0, 'lib/py/')
from splines import *

controlpoints=[
   [[ 0,0,0],[0 ,3  ,4],[0,6,3],[0,10,0]],
   [[ 3,0,2],[2 ,2.5,5],[3,6,5],[4,8,2]],
   [[ 6,0,2],[8 ,3 , 5],[7,6,4.5],[6,10,2.5]],
   [[10,0,0],[11,3  ,4],[11,6,3],[10,9,0]]]
dom = larDomain([20])
dom2D = larExtrude1(dom, 20*[1./20])
mapping = larBicubicSurface(controlpoints)
patch = larMap(mapping)(dom2D)
VIEW(STRUCT(MKPOLS(patch)))
◇
```

A Utility functions

Initial import of modules

⟨Initial import of modules 8⟩ ≡

```
from pyplasm import *
from scipy import *
import os,sys
""" import modules from larcc/lib """
sys.path.insert(0, 'lib/py/')
from lar2psm import *
from simplexn import *
from larcc import *
from largrid import *
from mapper import *
◇
```

Macro referenced in 4b.

References

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

[Pao03] A. Paoluzzi, *Geometric programming for computer aided design*, John Wiley & Sons, Chichester, UK, 2003.