

Lezione 12

Bioinformatica

Mauro Ceccanti[‡] e Alberto Paoluzzi[†]

[†]Dip. Informatica e Automazione – Università “Roma Tre”

[‡]Dip. Medicina Clinica – Università “La Sapienza”



Forward computation

Example

6 1 4 2 2 6 4 5 6 6 1 4 2 2 6 4 5 6
7 6 6 4 2 6 1 3 1 7 6 6 4 2 6 1 3 1
6 1 3 1 4 2 7 5 3 6 1 3 1 4 2 7 5 3
7 3 6 5 5 4 3 4 5 7 3 6 5 5 4 3 4 5
1 6 2 3 6 7 3 2 2 1 2 6 2 3 6 7 3 2 2
4 5 4 7 4 1 1 4 4 4 1 1 3 1 7 4 1 1 4 4
4 2 1 3 5 2 3 1 2 5 1 7 8 9 1 2 5 2 3 1 5
3 5 4 5 3 1 6 1 6 3 5 6 9 1 1 3 2 1 6 5 3
6 1 4 1 3 5 4 0 2 4 0 1 4 1 3 5 4 2 4
6 1 4 2 2 6 4 1 4 5 1 1 0 6 2 4 8 1 0 1 6 4 5 6
7 6 6 4 2 6 1 1 4 1 1 7 3 8 1 2 4 2 1 2 6 1 3 1
6 1 3 1 4 2 7 5 3 6 1 4 1 3 5 4 2 1 3 5 2 3 1 5
7 3 6 5 5 4 3 4 5 7 3 6 5 5 4 3 4 5
1 6 2 3 6 7 3 2 2 1 6 2 3 6 7 3 2 2
4 5 4 7 4 1 1 4 4 4 5 4 7 4 1 1 4 4
4 2 1 3 5 2 3 1 5 4 2 1 3 5 2 3 1 5
3 5 4 5 3 1 6 5 3 3 5 4 5 3 1 6 5 3
6 1 4 1 3 5 4 2 4 6 1 4 1 3 5 4 2 4

Basic tools

aaaaaaaaaa

```
1 def colortable (n):  
2     """ To build a color lookup table with n elements  
3         Return a list of GLcolor  
4     """  
5     return [ GLcolor((array([1.0/n,0,0,0])*k).tolist()) for  
6             k in range(1,n+1) ]  
7  
8 colors = colortable(10)  
9 size = 5  
10 coords = [[i,j] for i in range(size) for j in range(size)]  
11 table = STRUCT(map(item, coords))  
11 numbers = STRUCT(map(mass, coords))
```



Basic tools

aaaaaaaaaa

```
1 def item((i,j)):  
2     """ to generate a single colored cell of the table.  
3     Return a colored unit square translated in (i,j). """  
4     transl = T([1,2])([i,j])  
5     if a[i,j] > 0: color = colors[a[i,j]]  
6     else: color = WHITE  
7     return COLOR(color)(transl(CUBOID([1,1])))  
8  
9 def mass ((i,j)):  
10    """ To write the field value in position (i,j)  
11    Return a text suitably positioned in 2D space-"""  
12    transl = T([1,2])([i+0.5,j])  
13    m = TEXTWITHATTRIBUTES('centre',0.0,0.3,0.5,0.0) (  
14        str(abs(a[i,j])))  
15    return transl(T(2)(0.25)(m))
```



aaaaaaaaaa

aaaaaaaaaa

```
1 goal = (0, (2,2))  
2 update (goal)  
3 VIEW(show(table))  
4  
5 boundary = [goal]  
6 while boundary != []:  
7     c = boundary.pop() [1]  
8     new = cellforward(c)  
9     for e in new: insert(boundary, e)  
10    VIEW(show(table))
```



Forward computation

aaaaaaaaaa

Algorithm FORWARD(A, h)

Require: a (weighted) matrix A ; a *goal* cell (h)

Ensure: A filled with minimal distances from goal.

```
1: boundary =  $h$   
2:  $A[h] = 0$   
3: while boundary  $\neq \emptyset$  do  
4:     pop( $c$ , boundary)  
5:     insert(boundary, CELLFORWARD( $c$ ))  
6: end while
```



aaaaaaaaaa

side effect: A is being updated with $\min dist(k)$ from goal

Algorithm CELLFORWARD(A, k)

Require: a matrix A ; a *current* boundary cell (k)

Ensure: a subset of boundary cells.

```
1: adjacent ( $k$ ) = { $i$  for  $i \in NEIGHBOR(k)$  if  $a[i] \geq 0.0$ }  
2: for  $i \in adjacent(k)$  do  
3:      $a[i] = -(a[i] + a[k])$   
4: end for  
5: return adjacent
```



aaaaaaaaaa

aaaaaaaaaa

aaaaaaaaaa

a d -cell γ is represented as a multi-index $k = (k_0, \dots, k_{d-1}) \in \mathbb{Z}^d$

```
1 def cellforward(multiindex):
2     adjacent = filter(lambda i: a[tuple(i)] > 0,
3                         neighbor(multiindex))
4     for i in adjacent:
5         a[tuple(i)] = a[multiindex] - a[tuple(i)]
6     return [(a[tuple(i)], tuple(i)) for i in adjacent]
```

Algorithm NEIGHBOR(k)

Require: an d -cell (k)
Ensure: the set of d -cells $\delta\partial k - \{k\}$.
1: **return** $\delta\partial k - \{k\}$



Laplace-Beltrami operator $\delta\partial$

Compute the coboundary of the boundary of a cell. Return a list of multiindices of cells

```
1 def neighbor(multiindex):
2     def pred(k): return k-1
3     def succ(k): return k+1
4     def close(f, sequence, k):
5         local = list(sequence)
6         local[k] = f(local[k])
7         return local
8     def adj(f, index):
9         ret = []
10        for k in range(len(index)):
11            ret += [close(f, index, k)]
12        return ret
13    def lowfilter(seq):
14        zeros = [0]*len(shape(a))
15        return AND(AA(ISLE)(TRANS([seq, zeros])))
16    def highfilter(seq):
17        return AND(AA(ISGT)(TRANS([seq, shape(a)])))
18    lower = filter(lowfilter, adj(pred, multiindex))
19    upper = filter(highfilter, adj(succ, multiindex))
20    return lower + upper
```



Laplace-Beltrami operator $\delta\partial$

Examples in several dimensions

```
1 >>> print neighbor([8,7])
2 [[7, 7], [8, 6], [9, 7], [8, 8]]
3 >>> print neighbor([7])
4 [[6], [8]]
5 >>> print neighbor([1,2,3])
6 [[0, 2, 3], [1, 1, 3], [1, 2, 2], [2, 2, 3], [1, 3, 3],
  [1, 2, 4]]
7 >>> print neighbor([1,2,3,4])
8 [[0, 2, 3, 4], [1, 1, 3, 4], [1, 2, 2, 4], [1, 2, 3, 3],
  [2, 2, 3, 4], [1, 3, 3, 4], [1, 2, 4, 4], [1, 2, 3, 5]]
9 >>> print neighbor([0,0])
10 [[1, 0], [0, 1]]
```

The output is bounded by the 0-values of the indices, and by the actual `shape` of the underlying array, named `a`.



aaaaaaaaaa

aaaaaaaaaa

2 2 9 4 3	2 2 9 4 3	2 2 12 4 3	2 2 12 4 3	2 2 12 8 3
9 4 3 1 5	9 4 3 1 5	9 7 3 4 5	9 7 3 4 5	9 7 3 4 9
3 3 0 7 7	3 3 0 7 7	3 3 0 7 7	6 3 0 7 7	6 3 0 7 7
8 9 8 3 7	8 9 8 3 7	8 9 8 3 7	8 12 8 3 7	8 12 8 3 7
1 6 1 6 8	1 6 1 6 8	1 6 1 6 8	1 6 1 6 8	1 6 1 6 8

2 2 12 8 3	2 2 12 8 3	2 9 12 8 3	2 9 12 8 11	2 9 12 8 11
15 7 3 4 9	15 7 3 4 9	15 7 3 4 9	15 7 3 4 9	15 7 3 4 9
6 3 0 7 7	6 3 0 7 14	6 3 0 7 14	6 3 0 7 14	6 3 0 7 14
14 12 8 3 7	14 12 8 10 7	14 12 8 10 7	14 12 8 10 7	14 12 8 10 7
1 6 1 6 8	1 6 1 6 8	1 6 1 6 8	1 6 1 6 8	1 6 1 6 8

2 9 12 8 11	11 9 12 8 11	11 9 12 8 11	11 9 12 8 11	11 9 12 8 11
15 7 3 4 9	15 7 3 4 9	15 7 3 4 9	15 7 3 4 9	15 7 3 4 9
6 3 0 7 14	6 3 0 7 14	6 3 0 7 14	6 3 0 7 14	6 3 0 7 14
14 12 8 10 7	14 12 8 10 7	14 12 8 10 17	14 12 8 10 17	14 12 8 10 17
1 15 9 15 8	1 15 9 15 8	1 15 9 15 8	15 15 9 15 8	15 15 9 15 23



aaaaaaaaaa

aaaaaaaaaa

4 6 6 9 4 3 3 3 9 5	34 33 35 29 21 22 25 27 35 38
3 2 9 3 1 8 7 1 8 2	30 27 29 20 17 19 23 24 32 33
9 5 1 5 5 6 6 8 2 6	34 25 20 21 16 11 15 16 24 25 31
8 6 3 5 7 5 5 6 7 3	32 25 19 16 12 5 10 16 23 26
5 3 5 4 7 0 9 3 8 8	24 19 16 11 7 0 9 12 20 28
9 9 2 3 3 3 1 8 9 5	29 20 11 9 6 3 4 12 21 26
4 6 7 5 2 8 3 9 9 2	28 24 18 13 8 11 7 16 25 27
6 7 7 6 8 2 8 7 7 6	34 31 25 19 16 13 15 22 29 33
3 8 5 7 4 6 2 2 6 9	37 38 30 26 20 19 17 19 25 34
8 3 3 4 4 7 7 5 3 1	42 34 31 28 24 26 24 24 27 28



aaaaaaaaaa

aaaaaaaaaa

4 6 6 9 4 3 3 3 9 5	34 33 35 29 21 22 25 27 35 38
3 2 9 3 1 8 7 1 8 2	30 27 29 20 17 19 23 24 32 33
9 5 1 5 5 6 6 8 2 6	34 25 20 21 16 11 15 16 24 25 31
8 6 3 5 7 5 5 6 7 3	32 25 19 16 12 5 10 16 23 26
5 3 5 4 7 0 9 3 8 8	24 19 16 11 7 0 9 12 20 28
9 9 2 3 3 3 1 8 9 5	29 20 11 9 6 3 4 12 21 26
4 6 7 5 2 8 3 9 9 2	28 24 18 13 8 11 7 16 25 27
6 7 7 6 8 2 8 7 7 6	34 31 25 19 16 13 15 22 29 33
3 8 5 7 4 6 2 2 6 9	37 38 30 26 20 19 17 19 25 34
8 3 3 4 4 7 7 5 3 1	42 34 31 28 24 26 24 24 27 28



aaaaaaaaaa

aaaaaaaaaa

```

1 def maxClimb(a, element):
2     coboundary = neighbor(element)
3     maxelement = coboundary.pop()
4     while coboundary != []:
5         temp = coboundary.pop()
6         if a[tuple(temp)] > a[tuple(maxelement)]:
7             maxelement = temp
8     return maxelement

```



aaaaaaaaaa
aaaaaaaaaa

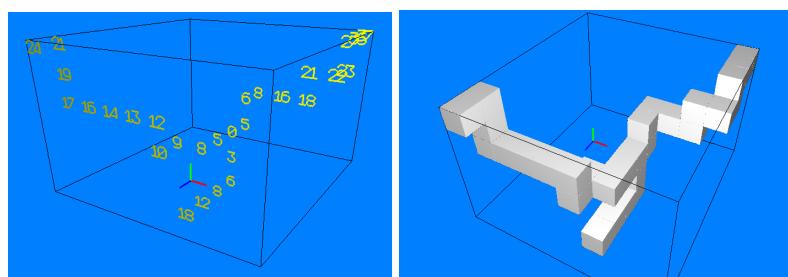
```
1 def backPath(a, List):
2     path = List
3     if a[tuple(path[-1])] != 0:
4         path += [maxClimb(a, path[-1])]
5         backPath(a, path)
6     return map(tuple, path)
```

aaaaaaaaaa
aaaaaaaaaa

```
1 def backward(a, points):
2     tree = set([])
3     for point in points:
4         tree = tree.union(backPath(a, [point]))
5     return list(tree)
```



aaaaaaaaaa
aaaaaaaaaa



Multidimensional update
aaaaaaaaaa

```
1 def item((i,j)):
2     """ to generate a single colored cell of the table.
3     Return a colored unit square translated in (i,j). """
4     transl = T([1,2])([i,j])
5     if a[i,j] > 0: color = colors[a[i,j]]
6     else: color = WHITE
7     return COLOR(color)(transl(CUBOID([1,1])))
```

```
1 def item (index):
2     """ to generate a single colored d-cell of the table
3     Return a colored cuboid translated in (i,j,...,k) """
4     coords = range(1,len(index)+1)
5     transl = T(coords)(index)
6     if a[tuple(index)] > 0:
7         color = colors[a[tuple(index)]]]
8     else: color = WHITE
9     return COLOR(color)(transl(CUBOID([1]*len(index))))
```



Multidimensional update

aaaaaaaaaa

```
1 def mass ((i,j)):  
2     """ To write the field value in position (i,j)  
3     Return a text suitably positioned in 2D space-"""  
4     transl = T([1,2])([i+0.5,j])  
5     m = TEXTWITHATTRIBUTES('centre',0.0,0.3,0.5,0.0) (   
        str(abs(a[i,j])))  
6     return transl(T(2)(0.25)(m))
```

```
1 def mass (index):  
2     """ To write the field value in position (i,j)  
3     Return a text suitably positioned in space"""  
4     coords = range(1,len(index)+1)  
5     tpar = [0]*len(index)  
6     tpar[0] = 0.5  
7     transl = T(coords)(array(index) + array(tpar))  
8     m = TEXTWITHATTRIBUTES('centre', 0.0, 0.3, 0.5, 0.0)  
    (str(abs(a[tuple(index)])))  
9     return transl(T(2)(0.25)(m))
```

