1 Python correction

```
import matplotlib.pyplot as plt
import numpy as np
from skimage.morphology import binary_erosion,disk,rectangle
from skimage.measure import perimeter
```

1.1 Shape contours

To generate a simple object, here is an example:

```
A = np.zeros((20,20)).astype('bool');
A[4:14, 9:17] = True;
A[1:18,11:16]=True;
```

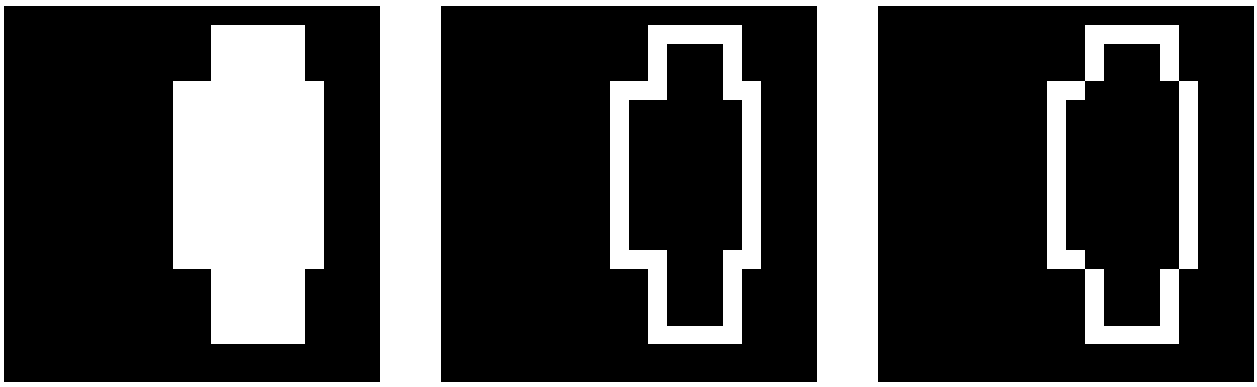
The contours are computed in 4- or 8-connectivity, see Fig.1. This function uses the mathematical morphology in order to get the contour.

```
def bwperim(I, connectivity=8):
    """
    Morphological inner contour, in 4 or 8 connectivity
    I: binary image
    return: binary image representing the contour
    """
    if connectivity==8:
        SE = disk(1);
    else:
        SE = rectangle(3,3);

    E = binary_erosion(I, selem=SE);

    return I^E;

% compute the contours
contours8 = bwperim(A, 4);
contours4 = bwperim(A, 8);
```



(a) Sample object. (b) Contour in 4-connectivity. (c) Contour in 8-connectivity.

Figure 1: Simple object and its contours in 4- or 8-connectivity.

1.2 Freeman chain code

1.2.1 First point of the shape

The important thing is to find one point in the contour. The Freeman chain code is sensitive to this choice, but several methods can transform this code so that this first choice does not have any importance.

```
def firstPoint(C):
    """
    find first point of contour
    returns point (as array)
    """
    p = np.argwhere(C);
    return p[0];
```

```
>>[r0,c0]=firstPoint(A)

r0 =
     5
c0 =
    10
```

1.2.2 Freeman chain code

The principle is to follow the contour, delete each pixel at each step, and find the direction of the next pixel.

```python
def freeman(C, p, connectivity=8):
    def getIndex(contour, point, connectivity):
        """ subfunction for getting the local direction
        """
        if connectivity==8:
            lut= np.array([[1, 2, 3], [8, 0, 4], [7, 6, 5]]);
        else:
            lut= np.array([[0, 2, 0], [8, 0, 4], [0, 6, 0]]);

        window = contour[point[0]-1:point[0]+2, point[1]-1:point[1]+2];
        window = window * lut;
        index  = np.max(window);
        return index-1;


    # Be careful that these LUTs consider coordinates from left to rith,
        ↪ top to bottom
    lutx = np.array([-1, -1, -1, 0, 1, 1, 1, 0]);
    luty = np.array([-1, 0, 1, 1, 1, 0, -1, -1]);
    lutcode = np.array([3, 2, 1, 0, 7, 6, 5, 4]);

    nbrpoints = np.sum(C);
    code=[];
    point = p.copy();
    C2 = C.copy();

    for i in np.arange(nbrpoints):
        C2[point[0], point[1]] = 0;

        index = getIndex(C2, point, connectivity);

        if (index==0):
            C2[p[0], p[1]] = 1;
            index = getIndex(C2, point, connectivity);

        # new point
        point[0] = point[0] + lutx[index];
        point[1] = point[1] + luty[index];

        # add code
        code.append(lutcode[index]);

    return code;
```

```python
code = freeman(C8, p);
```

```
code = array([6, 6, 5, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0, 7, 6, 6, 6, 0, 0,
    ↪  0, 0, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 4, 4, 4, 4])
```

1.3 Normalization

1.3.1 Differential code

This is the first step towards independence from the first point.

```
def codediff(fcc, connectivity=8):
    sr = np.roll(fcc, 1);
    d = fcc - sr;
    return d%connectivity;
```

1.3.2 Normalization

The differential code is then normalized, in order to get a rotation invariant code. All the circular shifts are evaluated, and a criterion (the minimum value) is established to be able to always find the same result, for every position of the first point.

```
def minmag(code):
    # high value for min computing
    codemin = np.max(code)* np.ones(code.shape);
    nb = len(code);
    for i in np.arange(len(code)):
        C = np.roll(code, i);

        for j in np.arange(nb):
            if C[j] > codemin[j]:
                break;
            elif C[j] < codemin[j]:
                codemin = C;
                break;
            if j == nb:
                codemin = C;

    return codemin;
```

The differential code is evaluated in d8, the normalization gives shapenumber8:

```
c = codediff(code, 8);
shapenumber8 = minmag(c);
```

```
c= array([2, 0, 7, 7, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 7, 7, 0, 0, 2, 0, 0,
    ↪ 0, 2, 0, 0, 7, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 7, 0, 2, 0, 0, 0])

shapenumber8= array([0, 0, 0, 0, 0, 0, 0, 0, 1, 7, 0, 2, 0, 0, 0, 2, 0,
    ↪ 7, 7, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 7, 7, 0, 0, 2, 0, 0, 0, 2, 0,
    ↪ 0, 7, 1])
```

1.3.3 Validation

This validation shows the effect on a different starting point.

```
p = np.array([4, 9]);
```

Another test is to verify the result after a rotation. To prevent discretization problems, we use 90 degrees and take the transpose of the matrix.

```
% check for rotation by 90 deg
contours8rot=contours8';
```

The same code should be found in both cases.

1.4 Geometrical characterization

1.4.1 Perimeter for 8-connectivity

We first need to extract the codes in the diagonal directions and apply a $\sqrt{2}$ factor, then add the number of codes in vertical and horizontal directions.

```
def Perimeter(fcode):
    """
    fcode: Freeman code
    """
    nb_diag = np.sum(np.array(fcode)%2);
    perim = nb_diag*np.sqrt(2) + len(fcode)-nb_diag;
    return perim;
```

The perimeter is evaluated in the same way in skimage.

```
Perimeter:  43.65685424949238
skimage.measure.perimeter:  43.65685424949238
```

1.4.2 Area for 8-connectivity

```
def Area(fcode):
    """
    """
    area = 0;
    B = 0;
    lutB = np.array([0, 1, 1, 1, 0, -1, -1, -1]);
    for i in np.arange(len(fcode)):
        lutArea = np.array([-B, -(B+0.5), 0, (B+0.5), B, (B-0.5), 0, -(B
            ↪ -0.5)]);
        area = area + lutArea[fcode[i]];
        B = B + lutB[fcode[i]];
    return area;
```

Notice that the area evaluated by this way is different from the number of pixels.

```
Area:  93.0
Number of pixels (area):  115
```