1 Python correction

```python
import numpy as np
import matplotlib.pyplot as plt
```

1.1 Graham scan algorithm

For convex hull and other computational geometry algorithms, robustness must be handled with special care. Floating points operations may be really tricky and the following code is not ensured to work for all cases.

```python
# very naive precision handling
points = np.round(points, decimals=4);
```

The first step is to get the starting point.

```python
# sort first by y, then x. get first point
ind=np.lexsort(points.transpose());
P = points[ind[0],:];

# all points but first one
points = points[ind]
pp =points[1:,:]
```

Then, the points are sorted by the cosinus of the angle. This step has complexity $O(n \log n)$.

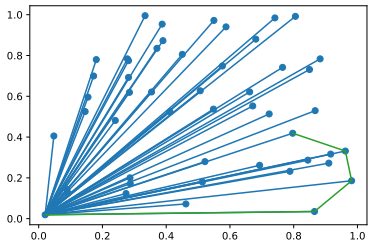
```python
# sort all points by angle
hypothenuse=np.sqrt( (pp[:,0]-P[0])**2 + (pp[:,1]-P[1])**2 );
adj_side = pp[:,0] - P[0];
# as cos is decreasing, we use minus
cosinus= -adj_side/hypothenuse;
ind=cosinus.argsort();

# construct ordered list of points
list_points = [];
list_points.append(P.tolist());
list_points= list_points + pp[ind,:].tolist();
list_points.append(P.tolist());
```

Finally, the sorted points allow to construct the convex hull by testing the orientation of the turn of the hull (see Fig.1).

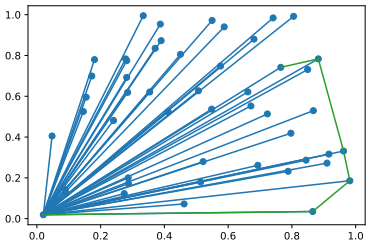
```
first = list_points.pop(0);
second= list_points.pop(0);
hull = []; # convex hull
hull.append(first);
hull.append(second);

for i, p in enumerate(list_points):
    while len(hull)>=2 and crossProduct(hull, p)<0:
        hull.pop();

    hull.append(p);

    # display result every 10 points
    if i%10 == 0:
        displayPointsAndHull(points, P, hull, 'chull_'+str(i)+'.python.
            ↪ pdf');
```

Figure 1: Graham scan illustration while constructing the convex hull.

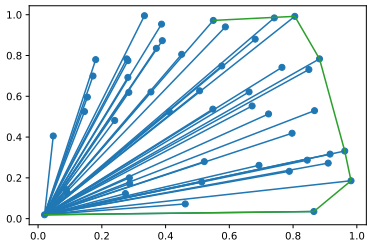
(a) After 10 tested points.



(b) After 20 tested points.



(c) After 30 tested points.



1.2 Useful functions

The cross-product function first extract the last two points of the hull, and check if there is a left-turn or a right-turn to go to the point p_3.

```python
def crossProduct(hull, p3):
    """
    Cross product
    hull : list that should contain at least 2 points
    p3   : point
    """
    p1 = hull[-2];
    p2 = hull[-1];

    c= (p2[0] - p1[0])*(p3[1] - p1[1]) - (p3[0] - p1[0])*(p2[1] - p1[1]);
    return c;
```

In order to display the results, one function is proposed.

```python
def displayPointsAndHull(points, P, hull, filename=None):
    """
    Fonction for display points and hull
    optionally save figure into pdf file
    """
    fig = plt.figure();
    if P is not(None):
        for i in np.arange(points.shape[0]):
            plt.plot([P[0],points[i,0]], [P[1],points[i,1]], 'C0');
    plt.scatter(points[:,0], points[:,1]);

    hull = np.array(hull);
    plt.plot(hull[:,0], hull[:,1], 'C2');
    plt.show()
    if filename:
        fig.savefig(filename, bbox_inches='tight');
```

1.3 Simple tests

For 5 points:

```python
Points=np.array([[ 1,  2],
       [ 1, -4],
       [ 2, -1],
       [ 3, -4],
       [ 4,  1],
       [ 3,  0]])

H = conv_hull(Points);
displayPointsAndHull(Points, H, 'sample_hull.python.pdf');
```

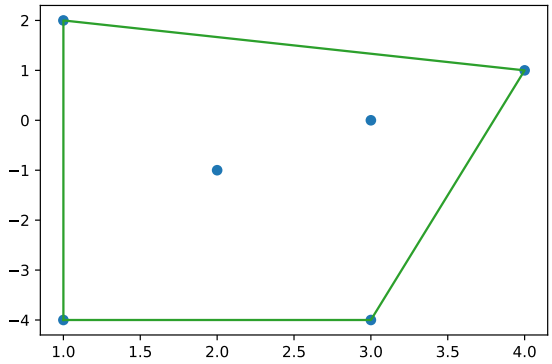
For a few random points:

```
nb=50;
Points = np.random.rand(nb, 2);
H = conv_hull(Points);
displayPointsAndHull(Points, H, 'random_hull.python.pdf');
```

The results are illustrated in Fig.2.

Figure 2: Illustration of the convex hull computation.

(a) Convex points of 5 points.



(b) Convex hull of 50 random points.