1

September 11, 2020

[1]:
```python
import numpy as np
```

[2]:
```python
def generate_edges(phi, n):
    """
    Construct the polyhedral cone with angle phi and n edges.
    :param phi: <float> Cone angle
    :param n: <int> Number of cone edges
    :return: <2-dim np.array> Cone edge matrix of size (n, 3)
    """
    # ------------------------------------------------
    # FILL WITH YOUR CODE

    v = np.zeros(3)
    # print(v)
    V = np.zeros((n,3))
    # print(type(V))
    theta = 0

    for i in range(n):
        theta = 2 * np.pi * i / n
        v = np.array([np.cos(theta)*np.cos(phi), np.sin(theta)*np.cos(phi), np.
↪sin(phi)])
        # print(v)
        V[i:] = v
        # print(V)

    cone_edges = V # TODO: Replace None with your result
    # ------------------------------------------------
    return cone_edges
# test
V_ = generate_edges(phi = 0.4, n = 5)
print(V_)
V = generate_edges(np.pi / 4,3)
print(V)
```

```
[[ 0.92106099  0.          0.38941834]
 [ 0.2846235   0.87598106  0.38941834]
```

```
 [-0.745154    0.54138607  0.38941834]
 [-0.745154   -0.54138607  0.38941834]
 [ 0.2846235  -0.87598106  0.38941834]]
[[ 0.70710678  0.          0.70710678]
 [-0.35355339  0.61237244  0.70710678]
 [-0.35355339 -0.61237244  0.70710678]]
```

[3]:
```
def compute_normals(cone_edges):
    """
    Compute the facet normals given the cone edge matrix.
    :param cone_edges: <2-dim np.array> Cone edge matrix of size (n, 3)
    :return: <2-dim np.array> Facet normals matrix of size (n, 3)
    """
    # ------------------------------------------------
    # FILL WITH YOUR CODE

    V = cone_edges
    # print(V)
    n = len(V)
    # print(n)
    s = np.zeros(3)
    S = np.zeros((n,3))
    # print(s,"\n",S)

    for i in range(n):
        if i >= n-1:
            s = np.cross(V[i],V[0])
        else:
            s = (np.cross(V[i],V[(i+1)]))
        S[i:] = s

    facet_normals = S  # TODO: Replace None with your result
    # ------------------------------------------------
    return facet_normals
# test
S = compute_normals(V)
print(S)
```

```
[[-4.33012702e-01 -7.50000000e-01  4.33012702e-01]
 [ 8.66025404e-01 -3.33066907e-16  4.33012702e-01]
 [-4.33012702e-01  7.50000000e-01  4.33012702e-01]]
```

[4]:
```
def compute_minimum_distance_from_facet_normals(a, facet_normals):
    """
    Compute the minimum distance from a point 'a' to the polyhedral
    cone parametrized by the given facet normals.
    :param a: <np.array> 3D point
```

```
    :param facet_normals: <2-dim np.array> Facet normals matrix of size (n, 3)
    :return: <float> Minimum distance from 'a' to the cone.
    """
    # ------------------------------------------------
    # FILL WITH YOUR CODE

    S = facet_normals
    d = np.zeros(len(S))
    # print(d)

    for i in range(len(S)):
        # print(S[i])
        S_abs = np.sqrt(S[i][0]**2+S[i][1]**2+S[i][2]**2)
        # d[i] = np.absolute(np.dot(S[i],a) / S_abs)
        d[i] = np.dot(S[i],a) / S_abs
        # print(np.dot(S[i],a))
        # print(d[i])
    d_star = np.amin(np.absolute(d))

    minimum_distance = d_star  # TODO: Replace None with your result
    # ------------------------------------------------
    return minimum_distance
# test
a = np.array([0,0,1])
d_star1 = compute_minimum_distance_from_facet_normals(a,S)
print(d_star1)
a = np.array([10,-10,0.2])
d_star2 = compute_minimum_distance_from_facet_normals(a,S)
print(d_star2)
```

```
0.4472135954999578
3.3632734565152456
```

[5]:
```
def compute_minimum_distance(a, n, phi):
    """
    Compute the minimum distance from a point 'a' to the polyhedral
    cone of n edges and angle phi
    :param a: <np.array> 3D point
    :param n: <int> Number of cone edges
    :param phi: <float> Cone angle
    :return: <float> Minimum distance from 'a' to the cone.
    """
    # ------------------------------------------------
    # FILL WITH YOUR CODE
    V = generate_edges(phi,n)
    S = compute_normals(V)
    d_star = compute_minimum_distance_from_facet_normals(a,S)
```

```
    minimum_distance = d_star  # TODO: Replace None with your result
    # ------------------------------------------------
    return minimum_distance
# test
d_star_1 = compute_minimum_distance(a = np.array([0.2,-0.3,0.1]), n = 7, phi =␣
 ↪0.3)
print(d_star_1)
d_star_2 = compute_minimum_distance(a = np.array([0.2,-0.3,0.1]), n = 10, phi =␣
 ↪0.01)
print(d_star_2)
```

```
0.0062024061079438446
0.09620653495102942
```

[6]:
```
def check_is_interior_point(a, n, phi):
    """
    Return whether a is an interior point of the polyhedral cone
    of n edges and angle phi
    :param a: <np.array> 3D point
    :param n: <int> Number of cone edges
    :param phi: <float> Cone angle
    :return: <bool> If a is an interior point
    """
    # ------------------------------------------------
    # FILL WITH YOUR CODE

    V = generate_edges(phi,n)
    S = compute_normals(V)
    d = np.zeros(len(S))
    # print(d)

    for i in range(len(S)):
        # print(S[i])
        S_abs = np.sqrt(S[i][0]**2+S[i][1]**2+S[i][2]**2)
        # d[i] = np.absolute(np.dot(S[i],a) / S_abs)
        d[i] = np.dot(S[i],a) / S_abs
        # print(np.dot(S[i],a))
        # print(d[i])
    d_star = np.amin(d)

    result = d_star > 0

    is_interior_point = result  # TODO: Replace None with your result
    # ------------------------------------------------
    return is_interior_point
# test
```

```python
res1 = check_is_interior_point(a = np.array([0.2,-0.3,0.1]), n = 7, phi = 0.3)
print(res1)
res2 = check_is_interior_point(a = np.array([0.2,-0.3,10]), n = 7, phi = 0.3)
print(res2)
```

```
False
True
```

[7]:
```python
if __name__ == "__main__":
    # You can use this main function to test your code with some test values

    # Test values
    phi = 30. * np.pi / 180.
    n = 4
    a = np.array([0.00, 0.01, 1.00])

    # Example for testing your functions
    cone_edges = generate_edges(phi, n)
    print("Generate Edges")
    print(cone_edges)
    print()
    facet_normals = compute_normals(cone_edges)
    print("Comptue Normals")
    print(facet_normals)
    print()
    minimum_distance = compute_minimum_distance_from_facet_normals(a,␣
 ↪facet_normals)
    print("Compute Minimum Distance from Facet Normals")
    print(minimum_distance)
    print()
    minimum_distance_ = compute_minimum_distance(a, n, phi)
    print("Compute Minimum Distance")
    print(minimum_distance_)
    print()
    Result = check_is_interior_point(a, n, phi)
    print("Check is Interior Point")
    print(Result)
    print()
```

```
Generate Edges
[[ 8.66025404e-01  0.00000000e+00  5.00000000e-01]
 [ 5.30287619e-17  8.66025404e-01  5.00000000e-01]
 [-8.66025404e-01  1.06057524e-16  5.00000000e-01]
 [-1.59086286e-16 -8.66025404e-01  5.00000000e-01]]

Comptue Normals
[[-0.4330127 -0.4330127  0.75      ]
```

```
 [ 0.4330127 -0.4330127  0.75      ]
 [ 0.4330127  0.4330127  0.75      ]
 [-0.4330127  0.4330127  0.75      ]]

Compute Minimum Distance from Facet Normals
0.7701245332864838

Compute Minimum Distance
0.7701245332864838

Check is Interior Point
True
```