

Problem 1

1.1

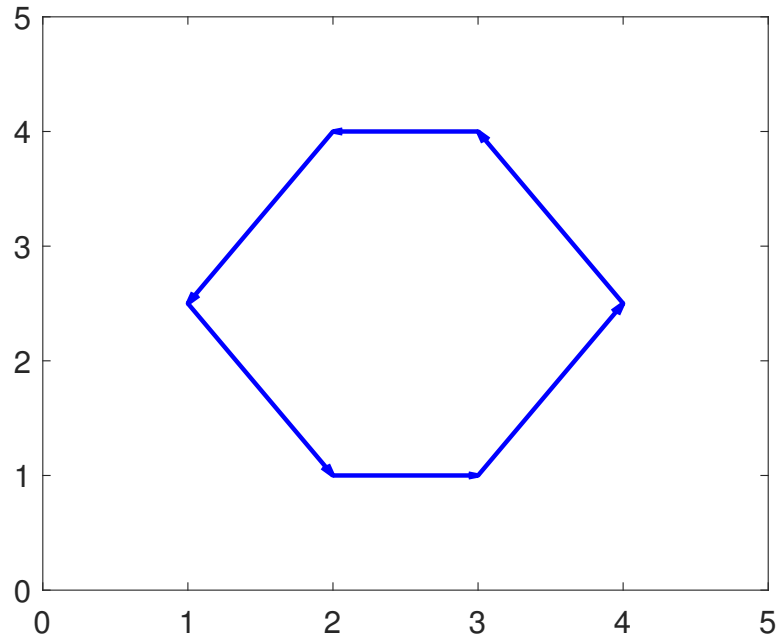


Figure 1: Polygon plotted using `polygon_plot()`

1.2

In the function `edge_angle()` `cangle` represents the cosine and `sangle` represents the sine. We began with three vertices

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

We now create vectors with v_0 as the vertex in common

$$\vec{v}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{1}$$

$$\vec{v}_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2}$$

Using matlab's `norm()` function, the magnitude of the vectors are computed

$$|v_1| = \text{norm}(\vec{v}_1) \tag{3}$$

$$|v_2| = \text{norm}(\vec{v}_2) \quad (4)$$

Equations for the dot product and cross product will now be defined as the next few steps make use of them to fully explain the meaning of *cangle* and *sangle*

$$\vec{a} \bullet \vec{b} = |a| |b| \cos(\theta) \quad (5)$$

$$\vec{a} \times \vec{b} = |a| |b| \sin(\theta) \quad (6)$$

If we rearrange the equation to give it in terms of unit vectors, it resembles the steps taken in *edge_angle()*

$$\frac{\vec{a} \bullet \vec{b}}{|a| |b|} = \vec{e}_a \bullet \vec{e}_b = \cos(\theta) \quad (7)$$

$$\frac{\vec{a} \times \vec{b}}{|a| |b|} = \vec{e}_a \times \vec{e}_b = \sin(\theta) \quad (8)$$

The unit vectors are computed in lines 26 and 27 by dividing \vec{v}_1 and \vec{v}_2 by their magnitudes. The expressions are now equal to just $\sin(\theta)$ and $\cos(\theta)$ and are computed with the following operations

$$\text{cangle} = \cos(\theta) = [e_{v1x} \quad e_{v1y}] x \begin{bmatrix} e_{v2x} \\ e_{v2y} \end{bmatrix} \quad (9)$$

$$\text{sangle} = \sin(\theta) = [0 \quad 0 \quad 1] \text{cross} \left(\begin{bmatrix} e_{v1x} \\ e_{v1y} \\ 0 \end{bmatrix}, \begin{bmatrix} e_{v2x} \\ e_{v2y} \\ 0 \end{bmatrix} \right) \quad (10)$$

The matlab function *cross()* computes the cross product of the provided vectors. The cosine and sine of theta have been computed which allows us to solve for theta using the expression

$$\tan(\theta) = \frac{\sin(\theta)}{\cos(\theta)} = \frac{\text{sangle}}{\text{cangle}} \quad (11)$$

and in the function, the last operation is $\text{edgeAngle} = \text{atan2}(\text{sAngle}, \text{cAngle})$ which equates to

$$\theta = \tan^{-1} \left(\frac{\sin(\theta)}{\cos(\theta)} \right) = \tan^{-1} \left(\frac{\text{sangle}}{\text{cangle}} \right) \quad (12)$$

The unsigned angle between the two vectors has now been computed. The last operation in the function is to convert the unsigned angle from $[0, 2\pi]$ to an angle from $[-\pi, \pi]$ if the flag is specified.

1.3

This section contains the plots outputted by *polygon_isVisible_test()*. Clearly, something is not right. My test cases for *edge_collision()* and *polygon_isSelfOccluded()* all pass and many scenarios were tested. I believe my error lies in my implementation and the way I check the points from each vertex.

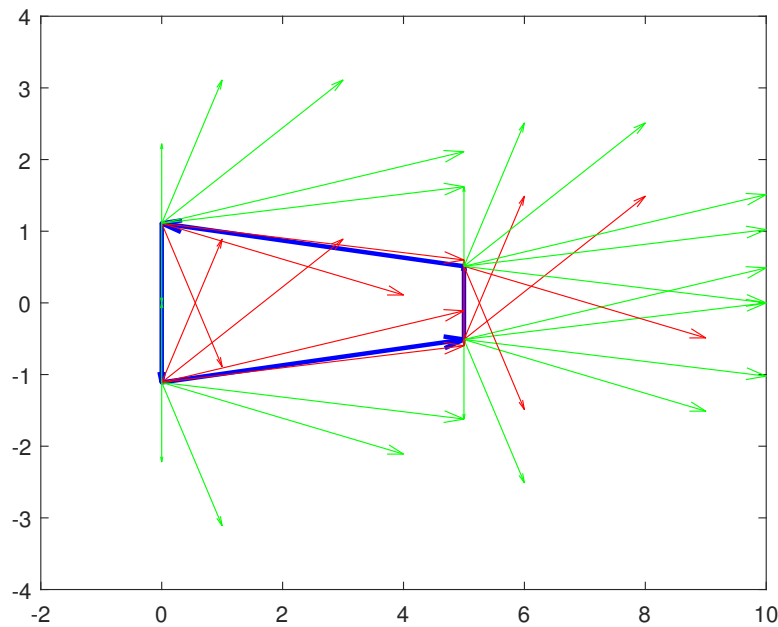


Figure 2: Results for vertices1

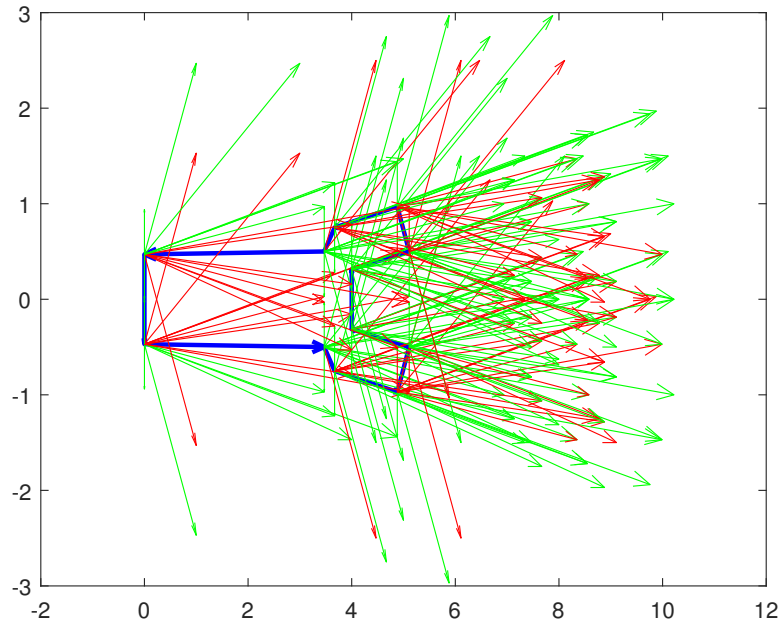


Figure 3: Results for vertices2

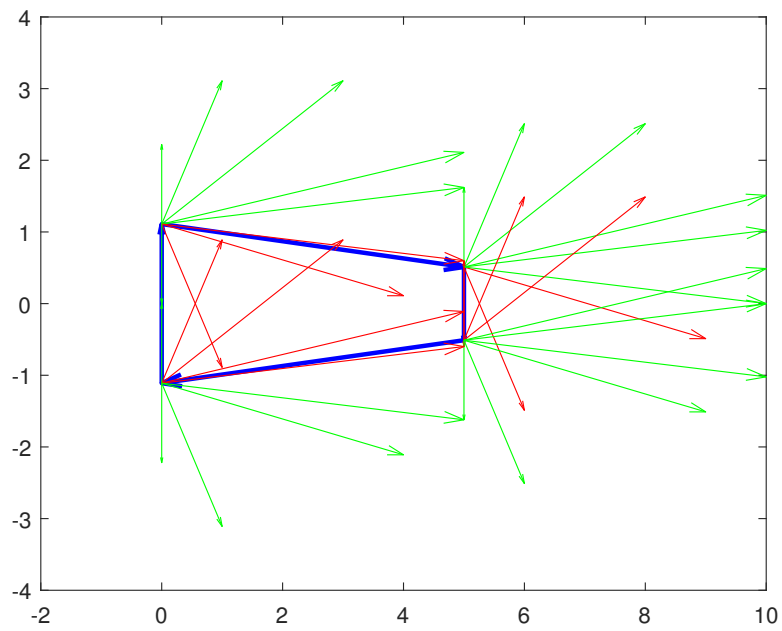


Figure 4: Results for vertices1 flipped

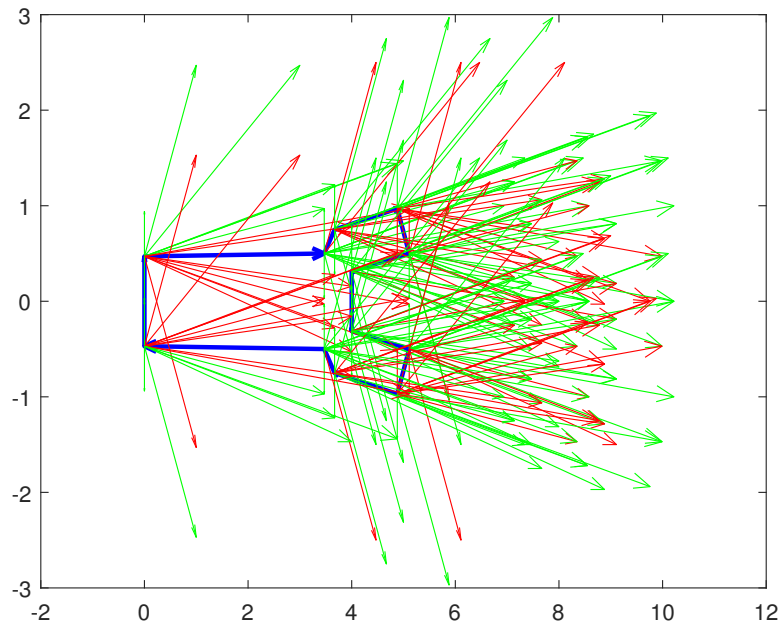


Figure 5: Results for vertices2 flipped

1.4

This section contains the plots outputted by *polygon_isCollision_test()*. Naturally, there are errors since this function is built around *polygon_isVisible*, a buggy function.

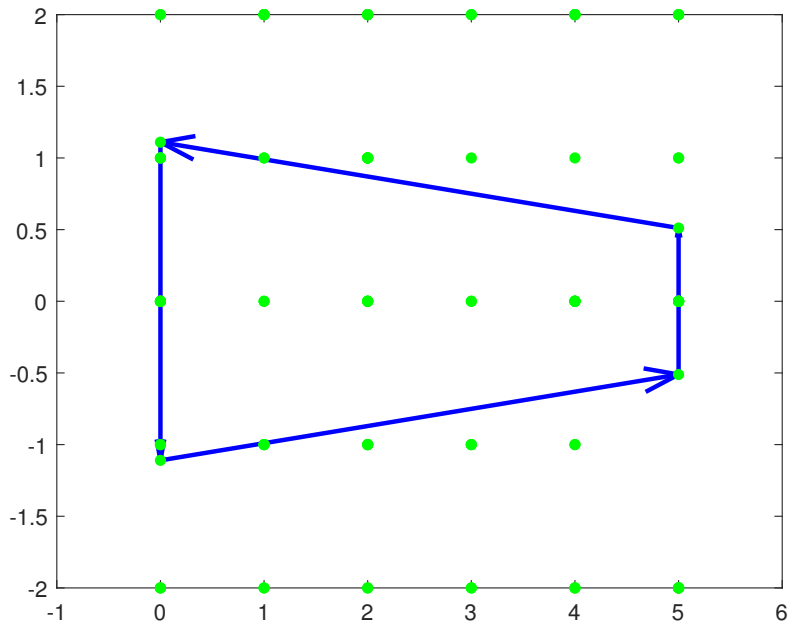


Figure 6: Results for vertices1

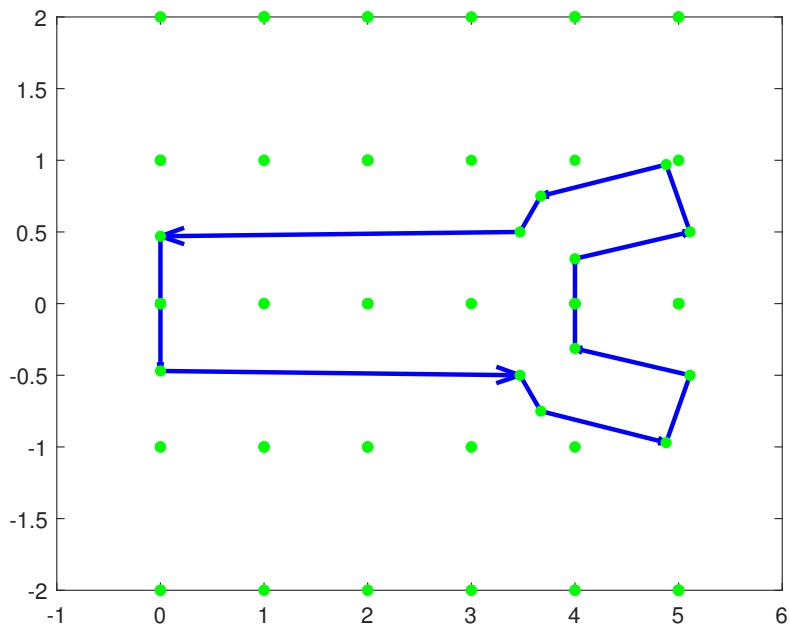


Figure 7: Results for vertices2

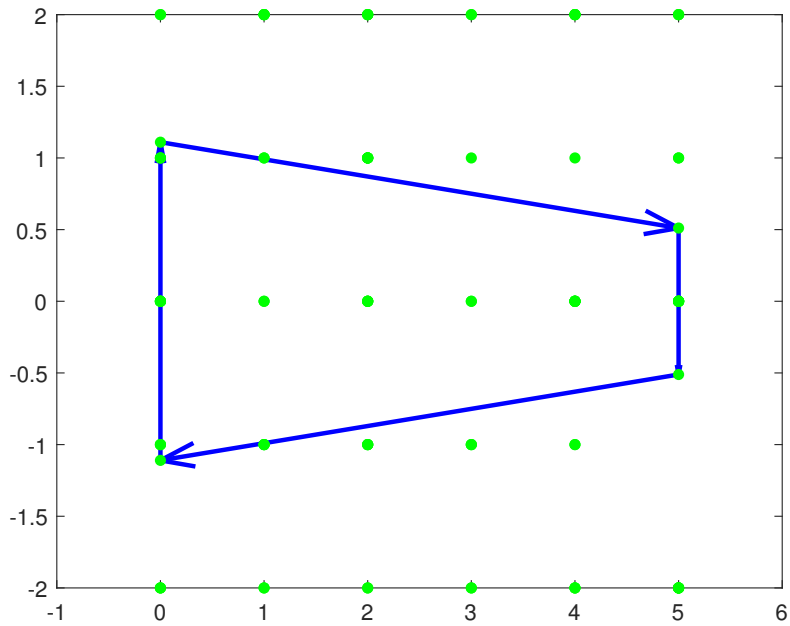


Figure 8: Results for vertices1 flipped

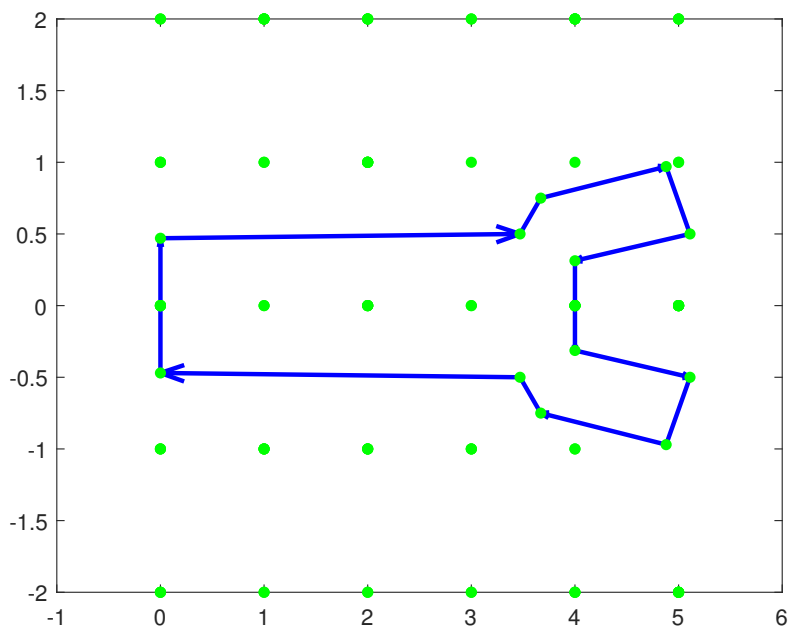


Figure 9: Results for vertices2 flipped

Problem 2

2.1

```
1 Following is the terminal output from the priority queue manipulation.
2 >> pQueue = priority_prepare()
3 pQueue =
4     0 1 empty struct array with fields:
5         key
6         cost
7 >> pQueue = priority_insert(pQueue, 'k1', 10)
8 pQueue =
9     struct with fields:
10         key: 'k1'
11         cost: 10
12 >> pQueue = priority_insert(pQueue, 'k2', 2)
13 pQueue =
14     1 2 struct array with fields:
15         key
16         cost
17 >> pQueue = priority_insert(pQueue, 'k3', 30)
18 pQueue =
19     1 3 struct array with fields:
20         key
21         cost
22 >> pQueue = priority_minExtract(pQueue)
23 pQueue =
24     1 2 struct array with fields:
25         key
26         cost
27 >> pQueue = priority_insert(pQueue, 'k4', 25);
28 >> priority_isMember(pQueue, '34')
29 ans =
30     logical
31     0
32 >> priority_isMember(pQueue, 'k4')
33 ans =
34     logical
35     1
36 >> pQueue = priority_minExtract(pQueue)
37 pQueue =
38     1 2 struct array with fields:
39         key
40         cost
41 >> pQueue = priority_minExtract(pQueue)
42 pQueue =
43     struct with fields:
44         key: 'k3'
45         cost: 30
```



```
46 >> pQueue = priority_minExtract(pQueue)
47 pQueue =
48     1  0 empty struct array with fields:
49         key
50         cost
51 >>
```

2.2

In this grid, the coordinates of each element correspond to the *key* in the pQueue. Naturally, the cost of the element in the grid corresponds to the *cost* in the pQueue.

pQueue is an array of structs, so the grip must be looped through and stored into the pQueue. Then, the minExtract function could be called until the queue is empty. Every time the function is called, the output is stored into a new queue. The new queue will be in ascending order, since it is filled by sequentially calling min extract which outputs the next lowest cost. If you want the elements back in the grid but in ascending order, you can simply loop through the length of the queue and store each element in the corresponding grid location.

3

3.1

The function *matlabStyleHinter()* was run in the directory. It checked the optional files and returned errors saying that output arguments were not used. Those return values are not shown below.

```
1 >> matlabStyleHinter()
2 File: edge_angle.m
3 * Programming style report
4     No problems found
5 * Matlab Code Analyzer report
6     No problems found
7 File: edge_angle_test.m
8 * Programming style report
9     No problems found
10 * Matlab Code Analyzer report
11     No problems found
12 File: edge_isCollision.m
13 * Programming style report
14     No problems found
15 * Matlab Code Analyzer report
16     No problems found
17 File: edge_isCollision_test.m
18 * Programming style report
```

```
19     No problems found
20 * Matlab Code Analyzer report
21     No problems found
22 File: polygon.isCollision.m
23 * Programming style report
24     No problems found
25 * Matlab Code Analyzer report
26     No problems found
27 File: polygon.isCollision.test.m
28 * Programming style report
29     No problems found
30 * Matlab Code Analyzer report
31     No problems found
32 File: polygon.isSelfOccluded.m
33 * Programming style report
34     No problems found
35 * Matlab Code Analyzer report
36     No problems found
37 File: polygon.isSelfOccluded.test.m
38 * Programming style report
39     No problems found
40 * Matlab Code Analyzer report
41     No problems found
42 File: polygon.isVisible.m
43 * Programming style report
44     No problems found
45 * Matlab Code Analyzer report
46     No problems found
47 File: polygon.isVisible.test.m
48 * Programming style report
49     No problems found
50 * Matlab Code Analyzer report
51     No problems found
52 File: polygon.plot.m
53 * Programming style report
54     No problems found
55 * Matlab Code Analyzer report
56     No problems found
57 File: polygon.plot.test.m
58 * Programming style report
59     No problems found
60 * Matlab Code Analyzer report
61     No problems found
62 File: priority.insert.m
63 * Programming style report
64     No problems found
65 * Matlab Code Analyzer report
66     No problems found
67 File: priority.isMember.m
68 * Programming style report
69     No problems found
```

```
70 * Matlab Code Analyzer report
71     No problems found
72 File: priority_minExtract.m
73 * Programming style report
74     No problems found
75 * Matlab Code Analyzer report
76     No problems found
77 File: priority_prepare.m
78 * Programming style report
79     No problems found
80 * Matlab Code Analyzer report
81     No problems found
82 File: priority_test.m
83 * Programming style report
84     No problems found
85 * Matlab Code Analyzer report
86     No problems found
87 File: twolink_polygons.m
88 * Programming style report
89     No problems found
90 * Matlab Code Analyzer report
91     No problems found
92 File: twolink_polygons_test.m
93 * Programming style report
94     No problems found
95 * Matlab Code Analyzer report
96     No problems found
97 >>
```

4

4.1

This homework took me roughly 10 hours. About the length of an instrumentation lab. I think this is a fair homework length assuming there is 2 weeks to complete it. With three other courses having a heavy workload I would be extremely pushed to get this done in a week. I spend roughly 95% of my time on problem one. It wasn't exceedingly difficult, but it was very time intensive to get it right. Even now, I know that it is not the best work possible, but I made a select few decision early on that would make going back and fixing it take just as long as doing the problem set in the first place.