Exercise 2

In [207…

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from math import isclose
```

In [449…

```
def generate_poly(n=20, d=3):
    '''
    This function generates a random convex polyhedron with n constraints (some
    d dimensions. Returns an nxd matrix (A) and a length n vector b such that f
    vector), Ax<=b encodes the constraints defining the polyhedron.

    :param n: the number of constraints to generate
    :param d: the number of dimensions of the polyhedron

    :return: A, the matrix of constraint coefficients, and b, the RHS of the co
    '''
    A = np.random.randn(n, d)
    A /= np.sqrt(np.sum(A**2, axis=1)).reshape(-1, 1)
    A = np.around(A, decimals=8)
    b = 1 + np.around(np.random.random(n), decimals=8)
    b = 1 + np.random.random(n)

    return A, b

def feasible(A, b, x):
    '''
    This function is a subroutine that checks whether a point x satisfies all c

    :param A: the nxd matrix of constraint coefficients
    :param b: the RHS of the constraints (length n vector)
    :param x: the d-dimensional coordinate to check (length d vector)

    :return: True if x satisfies Ax<=b, False otherwise
    '''
    #return np.all(A @ x <= b)
    return np.all(A @ x - b <= 1e-6)

def get_vertices(A, b, d=3):
    '''
    This function implements the vertex-finding algorithm described in hw6.pdf.
    vertices of the polyhedron defined by Ax<=b. Additionally, this program ret
    c_to_v, which maps constraint indices to the indices of thevertices that sa
    and v_to_c, which maps each vertex index to the indices of the constraints

    :param A: the nxd matrix of constraint coefficients
    :param b: the RHS of the constraints (length n vector)

    :return: a list of vertices of the polyhedron defined by Ax<=b, c_to_v, and
    '''

    vertices = []  # keep list of vertices
    constraint_idx = list(range(A.shape[0]))  # constraint indices
    systems = itertools.combinations(constraint_idx, 3)  # find all possible co
```

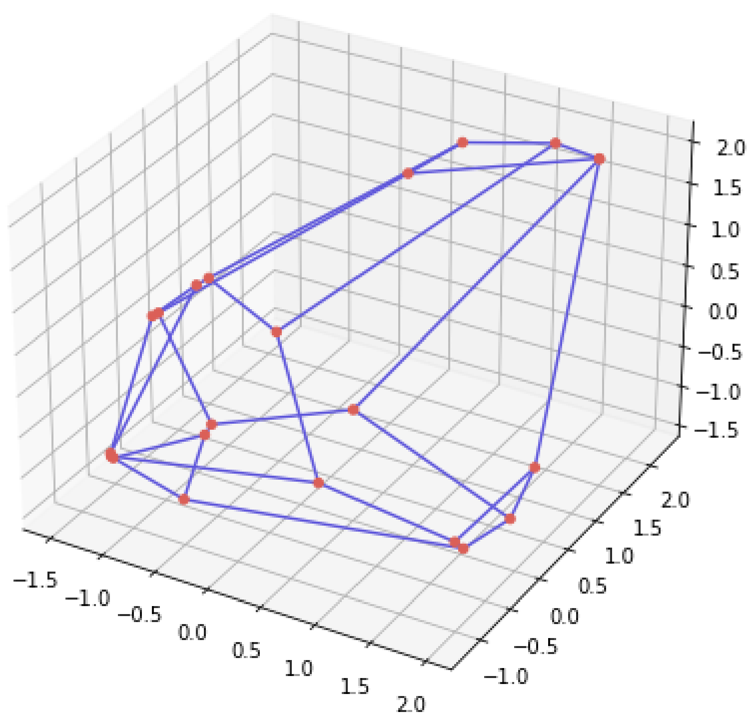
```python
    c_to_v = {c: [] for c in constraint_idx}  # map constraints to vertices
    v_to_c = {}  # map vertices to constraints

    v_idx = 0  # index of vertex

    # Solve 3x3 systems
    for s in systems:
        A_s = np.zeros((d, d))
        b_s = np.zeros(d)

        for i in range(d):
            A_s[i, :] = A[s[i], :]
            b_s[i] = b[s[i]]

        try:
            x = np.linalg.solve(A_s, b_s)

            # Check vertex
            if feasible(A, b, x):
                v_to_c[v_idx] = [c_idx for c_idx in s]

                for i in range(d):
                    c_to_v[s[i]].append(v_idx)

                vertices.append(x)
                v_idx += 1
        except np.linalg.LinAlgError:  # No unique solution
            continue

    for c in constraint_idx:
        for v in range(len(vertices)):
            vert = vertices[v]

            if np.dot(A[c, :], vert) == b[c]:
                if c not in v_to_c[v]:
                    v_to_c[v].append(c)
                if v not in c_to_v[c]:
                    c_to_v[c].append(v)

    return np.stack(vertices, axis=0), c_to_v, v_to_c

def get_edges(vertices, v_to_c):
    '''
    This function finds the edges of a polyhedron with the given vertices. Edge
    lists containing the indices of the vertices that they connect.

    :param vertices: a list of vertices of the polyhedron
    :param v_to_c: a dictionary that maps vertices to the indices of the planes

    :return: a list of edges of the polyhedron.
    '''

    edges = []

    for p in range(len(vertices)):
        for q in range(len(vertices)):
            shared = []

            for c in v_to_c[p]:
```

```
                if c in v_to_c[q]:
                    shared.append(c)

            if len(shared) == 2:
                if [p, q] not in edges and [q, p] not in edges:
                    edges.append([p, q])

    return edges

def plot_poly(vertices, edges, ax=None):
    '''
    This function plots the vertices and edges of a given polyhedron. It works

    :param vertices: a list of vertices of the polyhedron
    :param edges: a list of edges of the polyhedron
    :param ax: the axes to create the plot with
    '''

    colors = sns.color_palette('hls', 6)

    if vertices.shape[1] == 3:
        if ax is None:
            ax = plt.axes(projection='3d')
        ax.scatter3D(vertices[:, 0], vertices[:, 1], vertices[:, 2], color=colo

        for edge in edges:
            coords = np.stack([vertices[edge[0]], vertices[edge[1]]], axis=0)

            ax.plot(coords[:, 0], coords[:, 1], coords[:, 2], color=colors[4])

    else:
        if ax is None:
            ax = plt.axes()
        ax.scatter(vertices[:, 0], vertices[:, 1], zorder=3, color=colors[0], s

        for edge in edges:
            coords = np.stack([vertices[edge[0]], vertices[edge[1]]], axis=0)

            ax.plot(coords[:, 0], coords[:, 1], color=colors[4])
```

In [462…

```
A, b = generate_poly()
vertices, c_to_v, v_to_c = get_vertices(A, b)
edges = get_edges(vertices, v_to_c)

fig = plt.figure(figsize=(10, 7))
plot_poly(vertices, edges)
print(vertices)
```

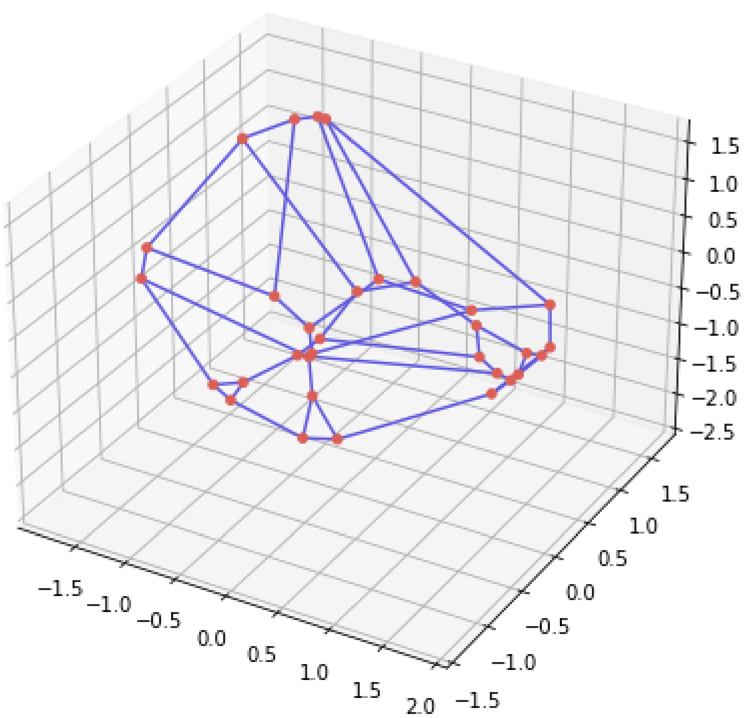
```
[[-1.05230446 -1.21688849 -0.61564131]
 [-1.04551551 -1.19476829 -0.64893658]
 [ 0.87173158 -1.18769171 -0.18092118]
 [ 1.76603697 -0.4427613  -1.13580468]
 [ 1.73429234 -0.53078564 -1.00449262]
 [-0.71332922 -0.67293702 -1.39139738]
 [-1.06260929 -1.20874729 -0.58826769]
 [-1.50480279  0.14566851  0.02827303]
 [-0.52359415 -0.70690835  1.30061662]
 [-1.07469167  0.23939943 -1.35500215]
 [-1.51277067  0.25017273 -0.00509511]
 [-1.10701808  0.40231927 -1.34541587]
 [-0.42375044 -0.68283582  1.40392183]
 [ 0.26902772 -0.79987439  1.10386805]
 [ 1.91174969  0.07523908 -1.08150477]
 [ 1.97314255  0.36737931 -0.63257957]
 [ 0.18996646  1.44470815  1.5715936 ]
 [ 1.51655607  2.29087605  1.70582516]
 [-0.09944672  1.04847108 -1.19852056]
 [ 1.52003019  2.29275945  1.70186944]
 [ 0.6065898   1.607243    1.98800413]
 [ 1.23754605  2.04042699  1.94042544]]
```



In [463…

```
A, b = generate_poly()
vertices, c_to_v, v_to_c = get_vertices(A, b)
edges = get_edges(vertices, v_to_c)

fig = plt.figure(figsize=(10, 7))
plot_poly(vertices, edges)
print(vertices)
```

```
[[ 1.83924765 -0.08867903  0.16778903]
 [ 1.83633297  0.14228258 -0.06416591]
 [ 1.8287542  -0.18933045 -0.03528196]
 [-1.32900237  1.42700789 -2.05928753]
 [-0.77250104  1.66555897 -1.32366007]
 [-1.22959879  1.32964733 -2.31222313]
 [ 0.22000873  1.61001601 -1.33240483]
 [ 1.74150567 -0.36003824  0.0939829 ]
 [ 1.65173846 -0.48088405  0.37648023]
 [ 0.51239668 -1.29012606  0.62800544]
 [ 0.67967751 -1.22967719  0.85964215]
 [ 1.60472902 -0.45956885  0.75526   ]
 [ 1.20004001 -0.70469579  1.36794544]
 [-0.77495662  0.90421635  1.50894028]
 [ 1.71097716  0.47090813 -0.28526569]
 [ 1.13834621  1.38660749 -0.70062375]
 [ 0.17278385 -0.75816201 -1.09170387]
 [ 0.39996472 -0.60384817 -1.14875956]
 [-0.55370499  0.38672257 -1.82576616]
 [ 1.80510246 -0.25798148 -0.06129246]
 [-0.12562184 -1.29511764 -0.22564372]
 [ 0.03879549 -1.3505794   0.12865051]
 [ 1.62388915 -0.25054767 -0.32469977]
 [-1.00965682  0.79641988  1.50279873]
 [-1.49189692  1.14990409 -1.42659206]
 [-0.85787018  0.90875853  1.51003475]
 [-0.29275983 -1.29763214 -0.08627886]
 [-1.70939923 -0.36682754 -0.00874449]
 [-1.22220075  1.27143373 -2.30557789]
 [ 0.86131588 -0.99598814  1.34868856]
 [-1.23372528  0.38775529  1.47352104]
 [-1.81337478 -0.15443895  0.20058005]]
```

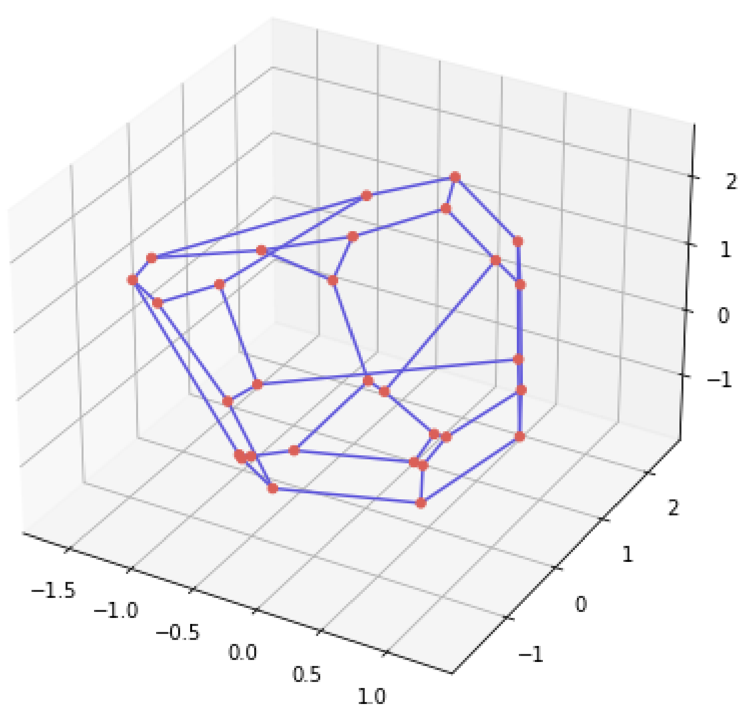


In [464…

```
A, b = generate_poly()
vertices, c_to_v, v_to_c = get_vertices(A, b)
```

```
edges = get_edges(vertices, v_to_c)

fig = plt.figure(figsize=(10, 7))
plot_poly(vertices, edges)
print(vertices)
```

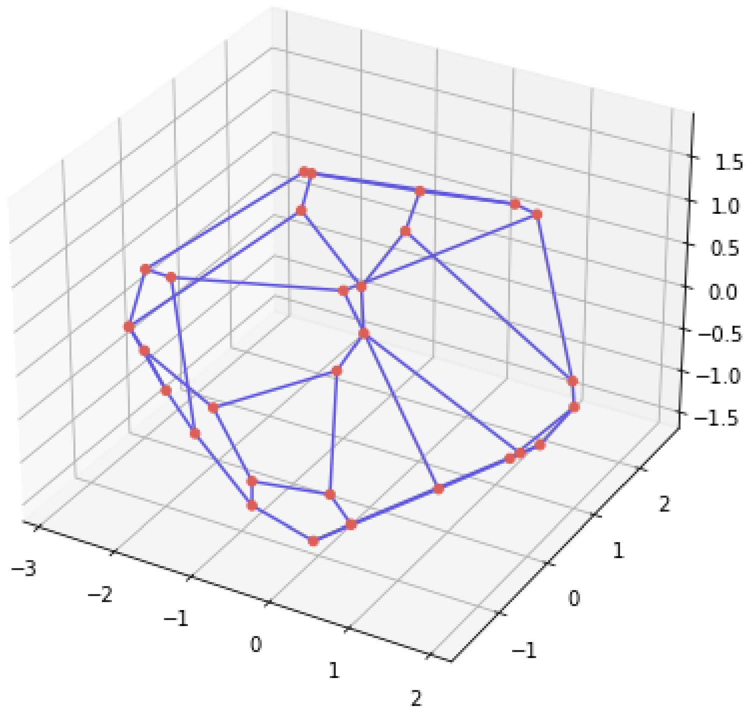
```
[[-1.63308696 -0.53974408  0.90372889]
 [-1.1521858  -1.14915737  1.21195091]
 [-0.70599844 -0.7352085  -1.08214609]
 [-0.67126786 -0.76408571 -1.09975687]
 [-0.3331108  -1.77462775  0.65025448]
 [-0.12978922 -1.45644777 -0.73642045]
 [ 0.89051538  0.71283767  1.99035859]
 [ 0.43350004  2.0426714   0.26630658]
 [ 1.27922134 -0.29169965  1.15157906]
 [ 1.26752364 -0.19421082 -0.04851731]
 [ 0.67709713  1.44340353 -0.8106873 ]
 [-0.31747775  2.42445585  0.79675568]
 [ 0.17436913  2.20011563  0.40139898]
 [ 0.29610704  0.99810022  2.44178064]
 [-1.66535657 -0.11509445  0.94229622]
 [-0.17193891  0.44051402  2.27744094]
 [-0.73516969 -0.96953397  1.58855067]
 [-0.66756701 -0.60257403 -1.17476008]
 [-1.08790115  2.02368349 -0.45995155]
 [-0.6062572   1.55629088 -1.46490636]
 [-1.40257602  1.35496809  0.26785245]
 [-0.56501501 -0.01997284 -1.43861735]
 [ 0.12168768  0.67465784 -1.71086491]
 [ 0.06286338  1.24015402 -1.70401467]
 [ 0.20628731  0.63669566 -1.68682345]
 [ 0.17021804  1.21651301 -1.67243428]
 [-1.01245037  2.24683916  0.12180056]
 [-0.45488761  1.51403643 -1.51976903]
 [-0.16126709 -1.63382068  0.89429081]
 [ 0.69242495 -0.64860863 -1.0512677 ]]
```



In [465…

```
A, b = generate_poly()
vertices, c_to_v, v_to_c = get_vertices(A, b)
edges = get_edges(vertices, v_to_c)

fig = plt.figure(figsize=(10, 7))
plot_poly(vertices, edges)
print(vertices)
```

```
[[-2.11606137 -1.14712243  0.17414959]
 [-0.92289995 -1.58220789  0.07088707]
 [-1.38325686 -1.27806863 -0.51273261]
 [-1.81780701 -1.18282297 -0.18591485]
 [-0.49102219 -1.52241708 -0.96056027]
 [-0.41011237 -1.63297511 -0.59672475]
 [ 0.41543739 -0.81232409  1.37428317]
 [ 0.65791997 -1.1363398   1.08748994]
 [-0.70068058 -0.23982699  1.65760801]
 [-2.441946   -0.96824577  0.27009642]
 [ 0.49338426 -1.3808113   0.75821495]
 [ 1.86603224  1.17970927 -0.63065217]
 [ 1.72647163  1.37227918 -0.46603631]
 [ 0.67842576  0.32353398 -1.46119529]
 [-1.42974407  1.66947561 -0.37243824]
 [ 1.36471163  0.68125417 -1.10381292]
 [ 0.52072712  2.59448754  0.56129407]
 [ 0.2955709   0.22154657  1.45175245]
 [ 1.9115448   0.3781829  -0.62996784]
 [ 1.86831279  0.03475308 -0.5493089 ]
 [-0.9486211   0.32597741  1.74176686]
 [ 0.06810364  0.86402813  1.53493166]
 [-2.44860399 -0.96280293  0.26825332]
 [-2.86101185 -0.08220876  0.38276837]
 [-1.10011634  0.41171088  1.68379215]
 [-2.71188965  0.17952566  0.1977194 ]
 [ 0.16694093 -1.34572444 -1.27658014]
 [ 0.68420918 -1.41243843 -0.89907743]
 [ 0.3917204   2.32544411  0.77859244]
 [ 0.51630931 -1.55272548 -0.52863815]]
```

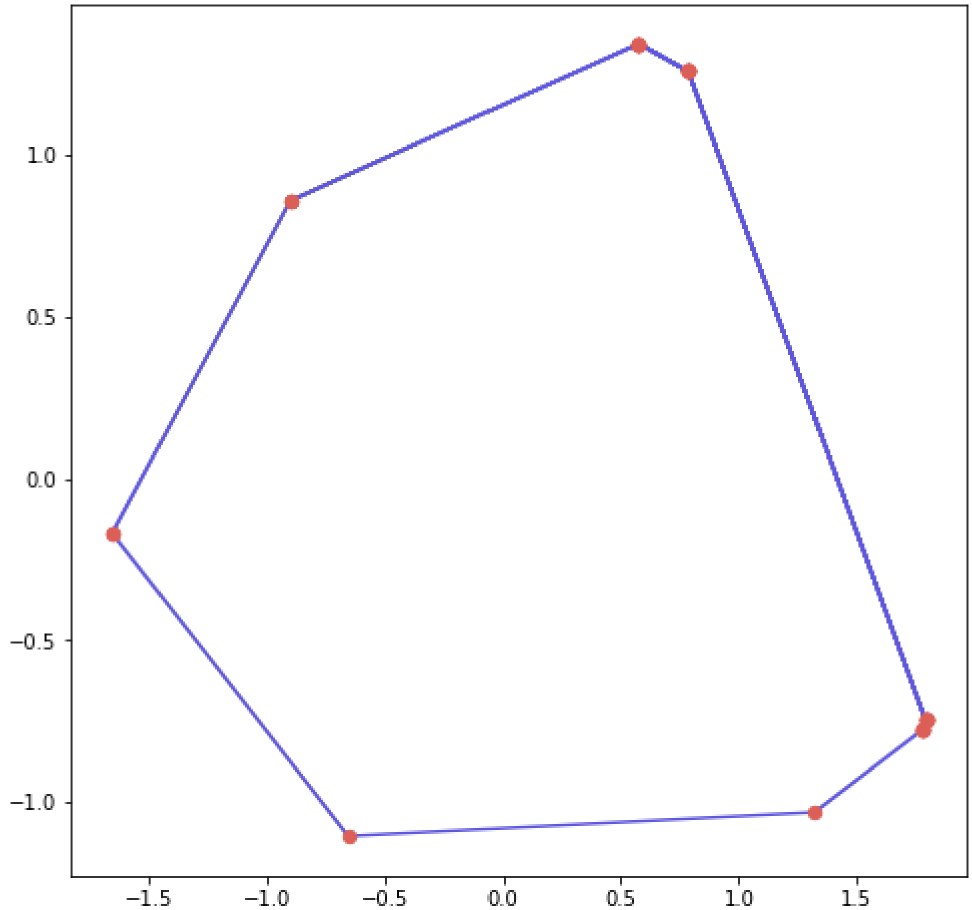


Exercise 3

In [467…

```
A2, b2 = generate_poly(d=2)
vertices, c_to_v, v_to_c = get_vertices(A2, b2, d=2)
edges = get_edges(vertices, v_to_c)

fig = plt.figure(figsize=(8,8))
plot_poly(vertices, edges)
print(np.unique(vertices, axis=0))
```

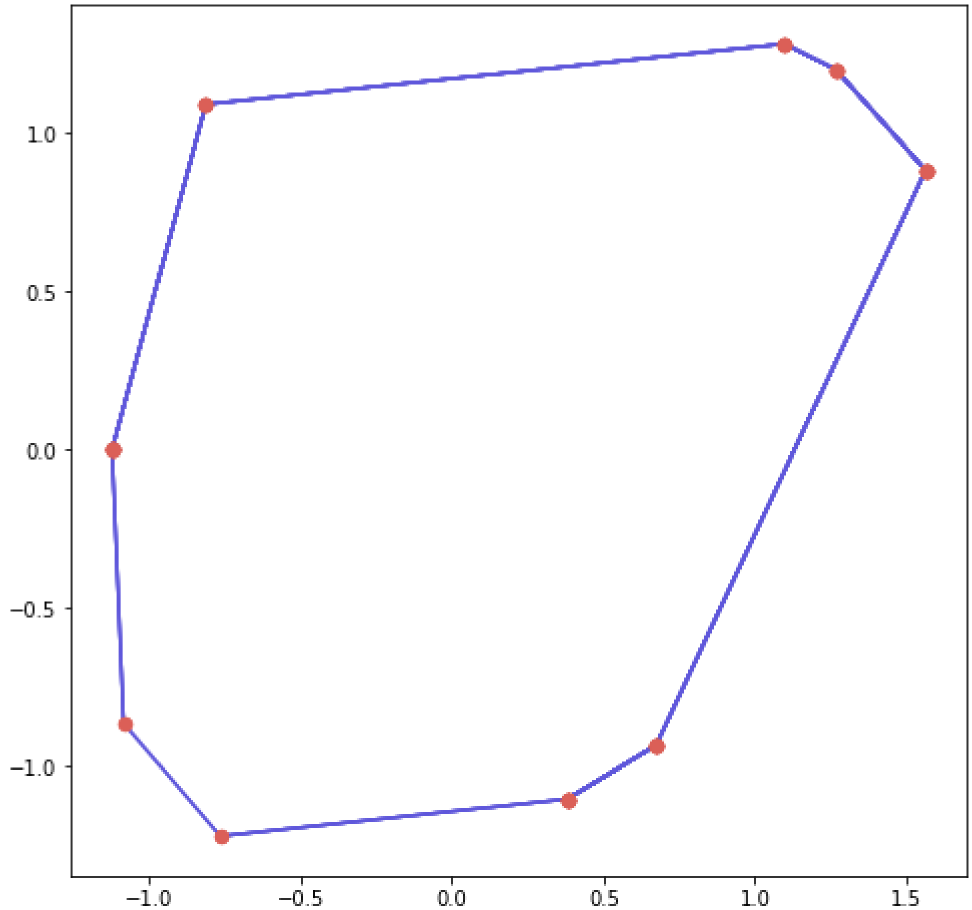
```
[[-1.65222825 -0.16907539]
 [-0.89812841  0.85874176]
 [-0.65264792 -1.10494468]
 [ 0.57627437  1.34296725]
 [ 0.78557875  1.25758885]
 [ 1.31964901 -1.03135719]
 [ 1.77746818 -0.77696236]
 [ 1.79035457 -0.74828474]]
```



In [468…

```
A2, b2 = generate_poly(d=2)
vertices, c_to_v, v_to_c = get_vertices(A2, b2, d=2)
edges = get_edges(vertices, v_to_c)

fig = plt.figure(figsize=(8,8))
plot_poly(vertices, edges)
print(np.unique(vertices, axis=0))
```

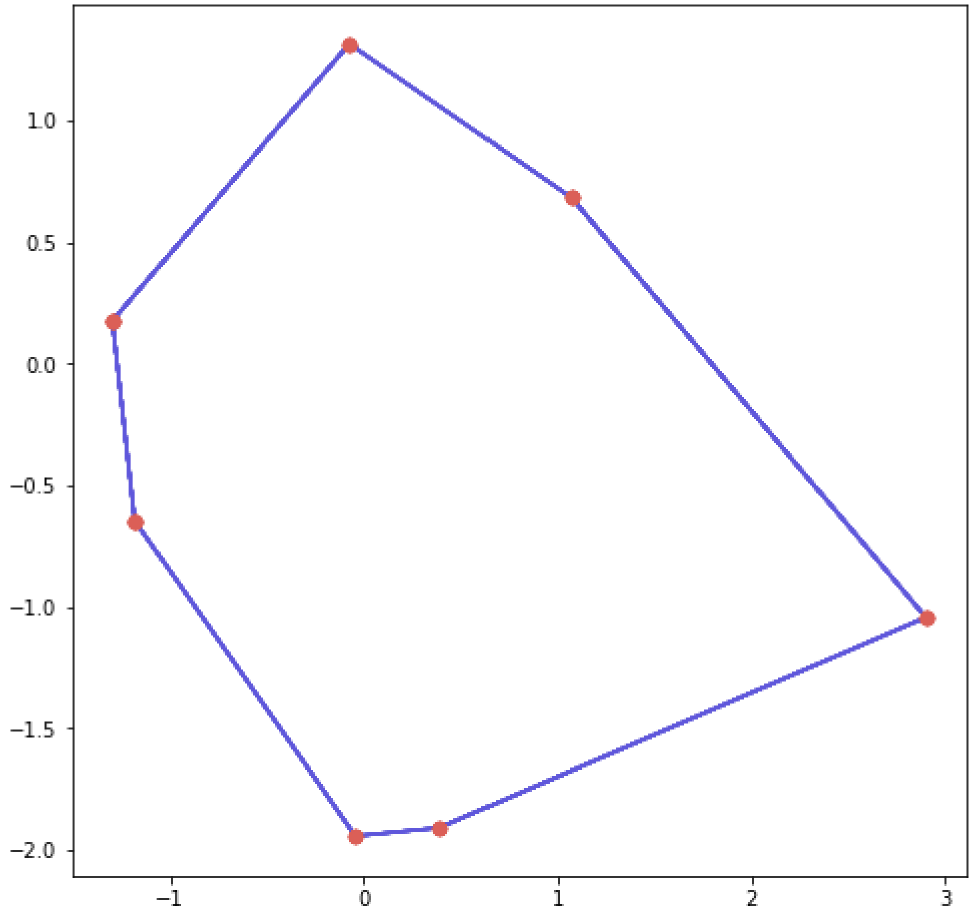
```
[[-1.12210795e+00  7.71585084e-04]
 [-1.08397404e+00 -8.64647415e-01]
 [-8.13055107e-01  1.09072520e+00]
 [-7.65407005e-01 -1.21960018e+00]
 [ 3.79691875e-01 -1.10268771e+00]
 [ 6.73837015e-01 -9.33707409e-01]
 [ 1.09827525e+00  1.28192682e+00]
 [ 1.26692812e+00  1.20072481e+00]
 [ 1.56396446e+00  8.83639544e-01]]
```



In [470…

```
A2, b2 = generate_poly(d=2)
vertices, c_to_v, v_to_c = get_vertices(A2, b2, d=2)
edges = get_edges(vertices, v_to_c)

fig = plt.figure(figsize=(8,8))
plot_poly(vertices, edges)
print(np.unique(vertices, axis=0))
```

```
[[-1.29798746  0.17729235]
 [-1.18218005 -0.64597905]
 [-0.07383374  1.3176718 ]
 [-0.04266916 -1.94380732]
 [ 0.39114767 -1.90975988]
 [ 1.07057522  0.68540941]
 [ 2.90181425 -1.04389957]]
```

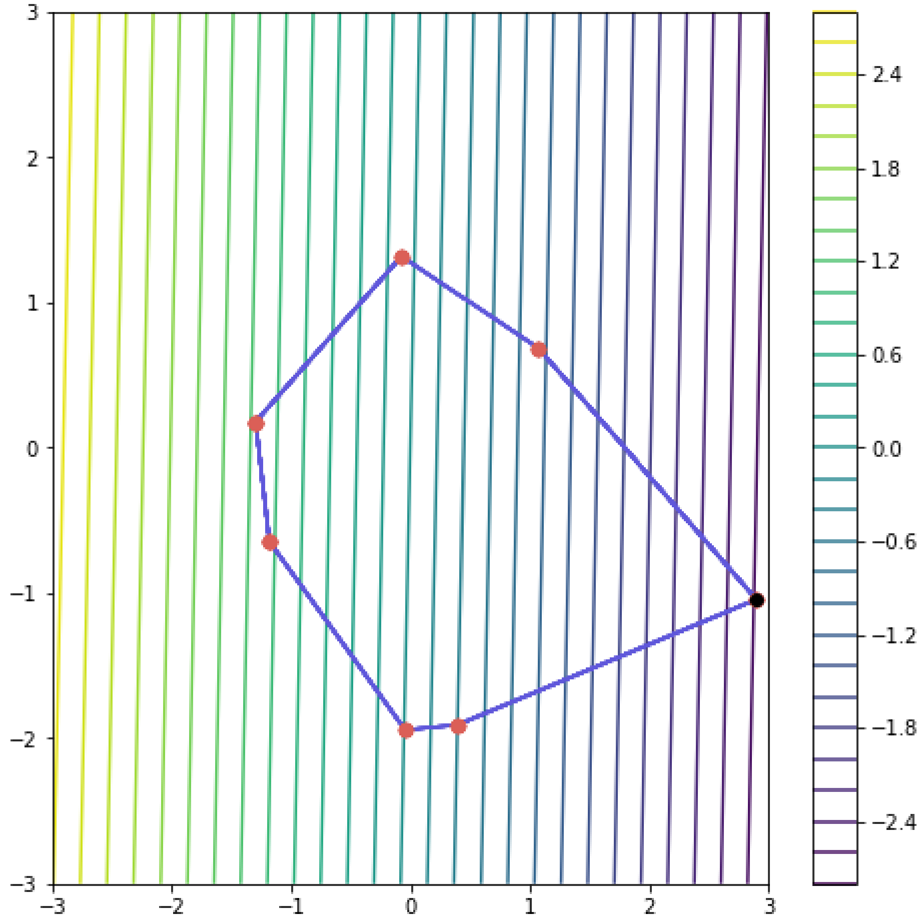


In [485…
```
c = np.random.randn(2)
```

The minimizer is plotted in black

In [492…
```
fig = plt.figure(figsize=(8,8))
plot_poly(vertices, edges)

x = np.linspace(-3, 3, 100)
X, Y = np.meshgrid(x, x)
h = c[0]*X + c[1]*Y
plt.contour(X, Y, h, levels=30, zorder=1)
plt.colorbar()

min_c = 1000000000
min_v = None
for v in vertices:
    if np.dot(v, c) < min_c:
        min_c = np.dot(v, c)
        min_v = v

plt.scatter(min_v[0], min_v[1], color='k', zorder=3)
```

Out[492]:
```
<matplotlib.collections.PathCollection at 0x13033f1c0>
```



In [ ]: