In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from pdffuns import *
```

Lab 2

Daniel Fylling

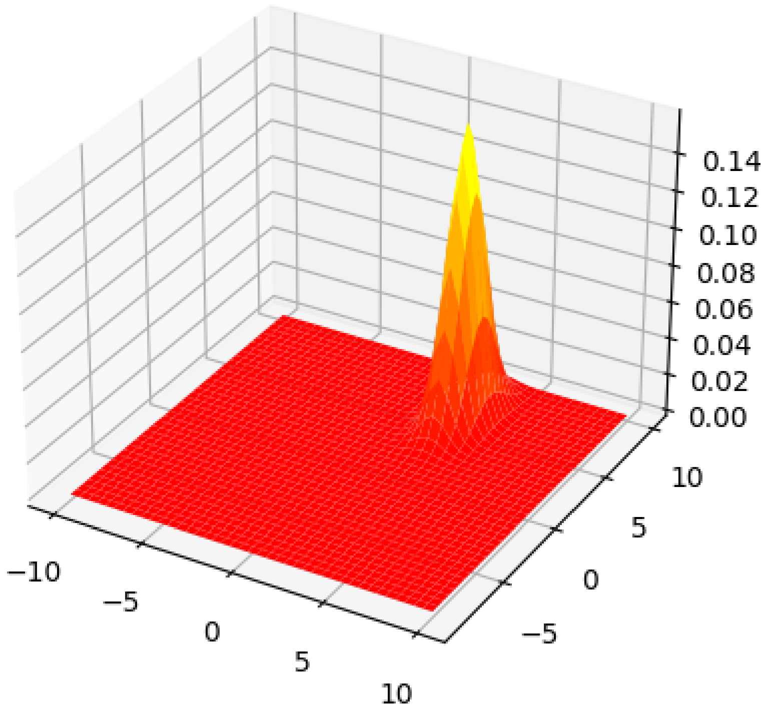
Problem 1

a) Stored values for my and Sgm in arrays for easier iteration. Reused norm2D from first lab for calculating grid points.

2D solution

In [ ]:
```
x1 = np.arange(-10,10.5,0.5).reshape(-1,1)
x2 = np.arange(-9,10.5,0.5).reshape(-1,1)

my = []
Sgm = []
P = []

my.append(np.array([3,6]))
Sgm.append(np.array([[0.5,0],[0,2]]))
my.append(np.array([3,-2]))
Sgm.append(np.array([[2,0],[0,2]]))

for i in range(len(my)):
    p, x1v, x2v = norm2D(my[i], Sgm[i], x1, x2)
    P.append(p)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x1v, x2v, P[0], cmap='autumn')
```

Out[ ]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1aa315a7f90>
```

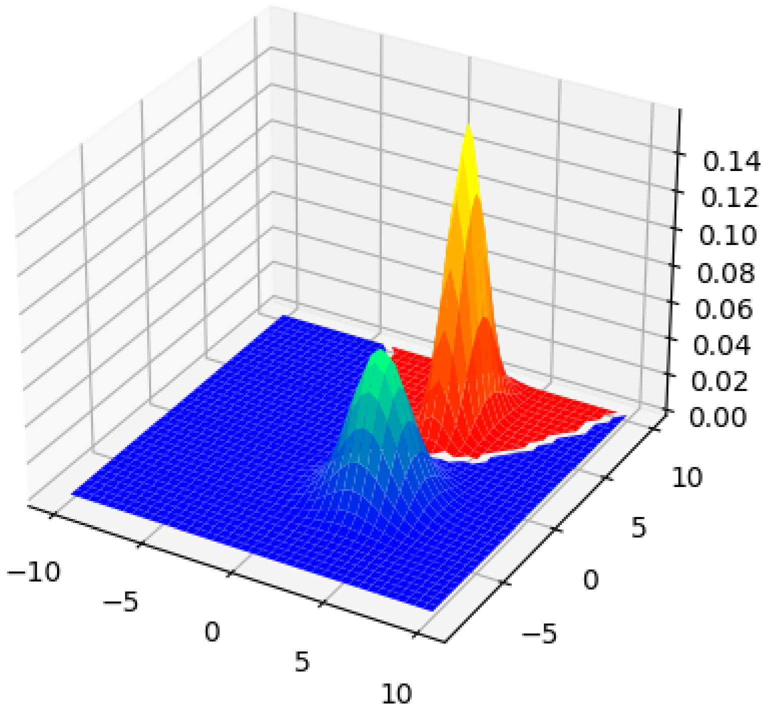


b) As was mentioned in Lab description in Canvas, the 3D plotting doesnt work well while plotting different z-values for each x1, x2 grid point.

Using boolean masking to set the lowest value of Pw1 and Pw2 to Nan for each grid point.

In [ ]:
```
a = P[0]
b = P[1]
mask = b<a
w1 = a.copy()*np.nan
w2 = a.copy()*np.nan

w1[mask] = a[mask]
w2[~mask] = b[~mask]

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x1v, x2v, w1, cmap='autumn')
ax.plot_surface(x1v, x2v, w2, cmap='winter')
```

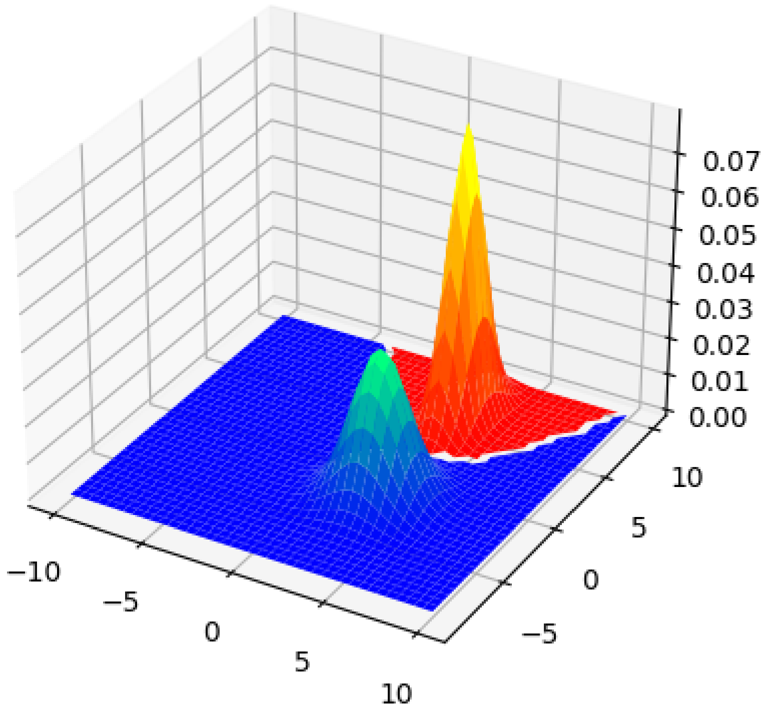
Out[ ]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1aa33128f10>
```



c)
From here on I became a bit lazy and put in the prior probabilities manually to each plot. Ideally I would like to store these in arrays and iterate to make the functions reusable, but I will not prioritize time for that right now.

In [ ]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x1v, x2v, w1*0.5, cmap='autumn')
ax.plot_surface(x1v, x2v, w2*0.5, cmap='winter')
```

Out[ ]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1aa31b22cd0>
```



d)

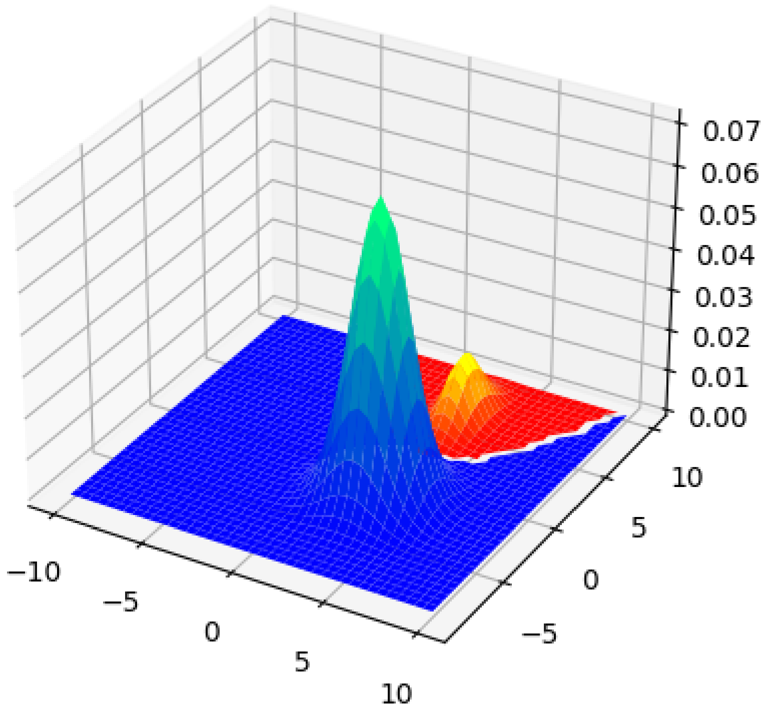
The decision boundary can be seen as the white line between the coloured graphs. We can see that the decision boundary has a non-linear form of ax^2 + bx + c. The decision regions for w1 and w2 will be the red and blue areas, respectively.

e)

Again, manually changed priors to adjust graphs.

In [ ]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x1v, x2v, w1*0.1, cmap='autumn')
ax.plot_surface(x1v, x2v, w2*0.9, cmap='winter')
```

Out[ ]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1aa319f2710>

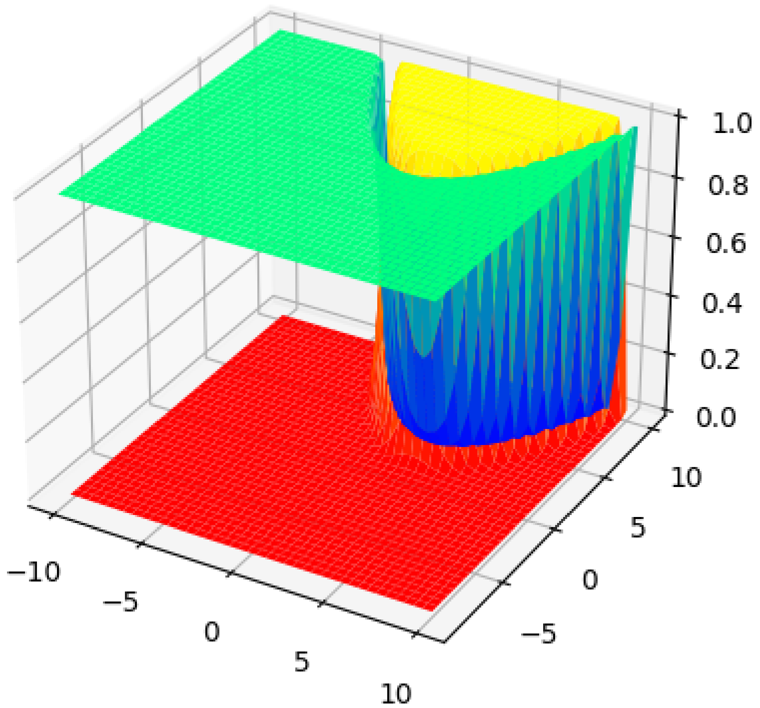


The effect is not big, but we can notice that the decision boundary has shifted towards the peak of P(w1). This means that the decision area for w2 has increased as the prior probability for w2 has also increased.

f)
Computing P(wi|x) from Bayes fomula and plotting. Again this could be done more clever with some iterations, but I'm leaving that for another day.

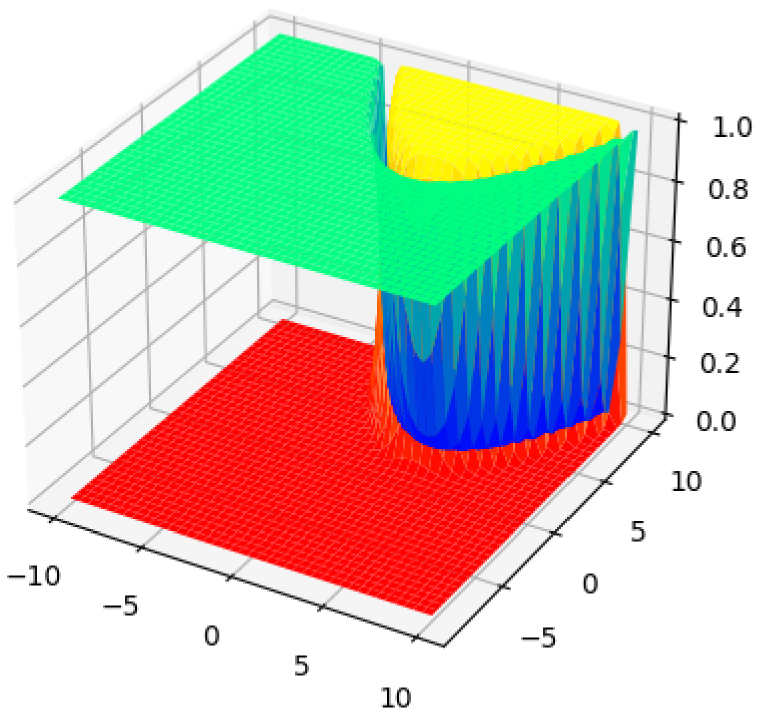
In [ ]:
```
Pw1x = P[0] * 0.5 / (P[0] * 0.5 + P[1] * 0.5)
Pw2x = P[1] * 0.5 / (P[0] * 0.5 + P[1] * 0.5)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x1v, x2v, Pw1x, cmap='autumn')
ax.plot_surface(x1v, x2v, Pw2x, cmap='winter')
```

Out[ ]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1aa3537e150>
```



In [ ]:
```
w1 = P[0] * 0.1 / (P[0] * 0.1 + P[1] * 0.9)
w2 = P[1] * 0.9 / (P[0] * 0.1 + P[1] * 0.9)

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(x1v, x2v, a, cmap='autumn')
ax.plot_surface(x1v, x2v, b, cmap='winter')
```

Out[ ]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1aa33877fd0>
```



We know from the mathematical foundation of this theory that the decision boundary and -regions are the same when using intersections of P(wi)P(x|wi) or P(wi|x). The graphs we have produced supports this as we can observe that the intersections shown here at the end corresponds to the white line from before.

In [ ]:
```
# Here I tried to do some even more clever
# boolean masking for eliminating lower values in the plots.
# Could not get this quite to work for now.

# G=np.asarray(P)

# G[np.where(G==np.max(G, axis=0))] = np.nan

# fig = plt.figure()
# ax = fig.add_subplot(projection='3d')
# ax.plot_surface(x1v, x2v, G[0], cmap='autumn')
# ax.plot_surface(x1v, x2v, G[1], cmap='winter')
```

Out[ ]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1aa33c576d0>