1. a) $H(P) = -\sum_{y \in \Omega_Y} P(y) \log P(y)$

$$= -(P(\text{class } 0)\log P(\text{class } 0) + P(\text{class } 1)\log P(\text{class } 1))$$

$$= -(0.6\log 0.6 + 0.4\log 0.4)$$

$$H(P) = \underline{0.971 \text{ bits}}$$

$$H(Q) = \underline{-(p\log p + (1-p)\log(1-p))}$$

b) $H_{min}(P,Q) = \min_{p}\left(-\sum_{i=0}^{1} P(Y=i)\log Q(Y=i)\right)$

$$= \min_{p}\left(-(0.6\log p + 0.4\log(1-p))\right)$$

$$\frac{\partial}{\partial p}\left(-(0.6\log p + 0.4\log(1-p))\right) = -\left(0.6\frac{1}{p} - 0.4\frac{1}{1-p}\right)$$

$$0 = \frac{-0.6}{p} + \frac{0.4}{1-p} \to \frac{0.6}{p} = \frac{0.4}{1-p} \to 0.6 - 0.6p = 0.4p$$

$$\underline{p = 0.6}$$

$$H_{min}(P,Q) = -(0.6\log 0.6 + 0.4\log 0.4)$$

$$= \underline{0.971}$$

c) $D_{KL}(P\|Q) = \sum_{x \in \Omega_X} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$

$$D_{KL}(P\|Q) = H(P,Q) - H(P)$$

$$\sum_{x \in \Omega_X} P(x)\log\left(\frac{P(x)}{Q(x)}\right) = H(P,Q) - H(P)$$

$$\sum_{x \in \Omega_X} P(x)\log\left(\frac{P(x)}{Q(x)}\right) = -\sum_{x \in \Omega_X} P(x)\log Q(x) + \sum_{x \in \Omega_X} P(x)\log P(x)$$

using log law $\log\left(\frac{a}{b}\right) = \log a - \log b$

$$\sum_{x \in \Omega_X} P(x)\log P(x) - \sum_{x \in \Omega_X} P(x)\log Q(x) = -\sum_{x \in \Omega_X} P(x)\log Q(x) + \sum_{x \in \Omega_X} P(x)\log P(x)$$

$$-\sum_{x \in \Omega_X} P(x)\log Q(x) + \sum_{x \in \Omega_X} P(x)\log P(x) = -\sum_{x \in \Omega_X} P(x)\log Q(x) + \sum_{x \in \Omega_X} P(x)\log P(x)$$

---

d) $\frac{d}{dp}(D_{KL}(P||Q)) = \frac{d}{dp}\left(0.6\log\frac{0.6}{p} + 0.4\log\frac{0.4}{1-p}\right)$

$= \frac{d}{dp}\left(0.6(\log 0.6 - \log p) + 0.4(\log 0.4 - \log(1-p))\right)$

$= \frac{d}{dp}\left(0.6\log 0.6 - 0.6\log p + 0.4\log 0.4 - 0.4\log(1-p)\right)$

$0 = -0.6\frac{1}{p} + 0.4\frac{1}{1-p}$

$\frac{0.6}{p} = \frac{0.4}{1-p} \rightarrow 0.6 - p = 0.4p \rightarrow \underline{p = 0.6}$

Plugging in $p = 0.6$

$0.6\log\frac{0.6}{0.6} + 0.4\log\frac{0.4}{0.4} = \underline{0}$

2. a)

```
prior_mean = 0
prior_covariance = 0
```

b) $[w_{MAP}, b_{MAP}] = \underset{\vec{w}}{\operatorname{argmin}}\left(\lambda\|\vec{w}\|_2^2 + \sum_{i=1}^{n}(y_i - \vec{w}^T\vec{x}_i)^2\right)$

We can express this as

$J(\vec{w}) = \lambda\|\vec{w}\|_2^2 + (\vec{y} - X\vec{w})^T(\vec{y} - X\vec{w})$

To minimize, we can take the derivative and set it to zero

$\frac{dJ}{d\vec{w}} = 2\lambda\vec{w} - 2X^T(\vec{y} - X\vec{w}) = 0$

Solving for $\vec{w}$ gets us

$\vec{w}_{MAP} = (X^TX + \lambda I)^{-1}X^T\vec{y}$

```
[w,b]map
[[0.00098067]
 [0.64716851]]
```

The logic here is that taking the derivative, setting it to zero, and solving for $\vec{w}$ is equivalent to finding the minimum. It is a more feasible formula for our coding implementation.

c) $V_n = \sigma^2(\sigma^2 V_0^{-1} + X^TX)^{-1}$

$w_n = V_N V_0^{-1} w_0 + \frac{1}{\sigma^2} V_N X^T y$

```
Posterior Covariance
[[6.54724280e-08 7.14454644e-07]
 [7.14454644e-07 9.90878644e-03]]
Posterior Mean
[[0.00098067]
 [0.64716851]]
```

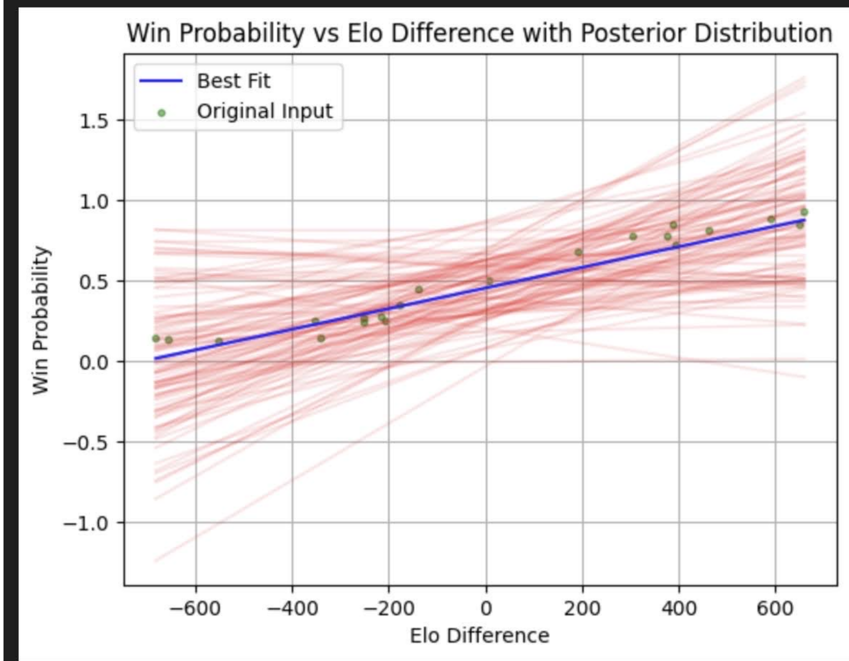
d) The posterior mean represents the expected values of w, b given the observed data and prior.

The posterior covariance matrix represents the uncertainty of the estimates of w, b. The diagnal represents variances of w, b, and off-diagonal represents the covariance of w, b.

The practical significance of w is that it captures the relationship between x (ELO difference) and y (win probability). It tells us how much we'd expect y to change with a one unit increase of x.

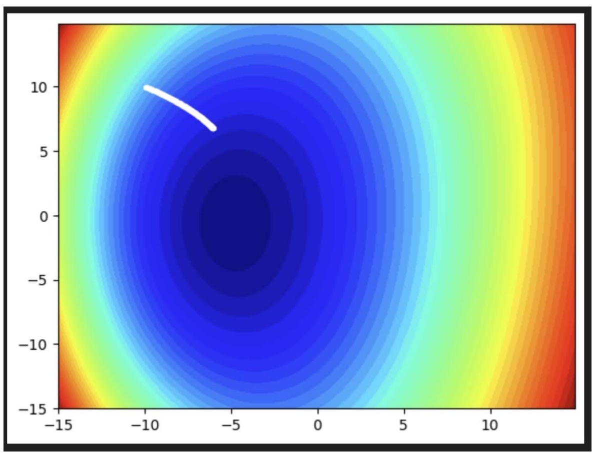
e)

```
Best Slope: 0.0006413165723068061
Best Intercept: 0.45395569509542305
```



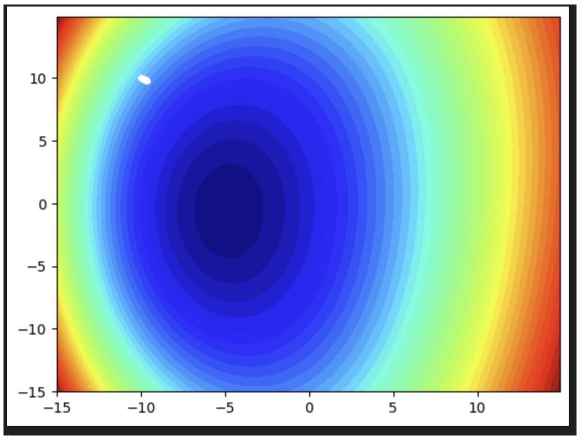
3. b)

Step size = 0.2



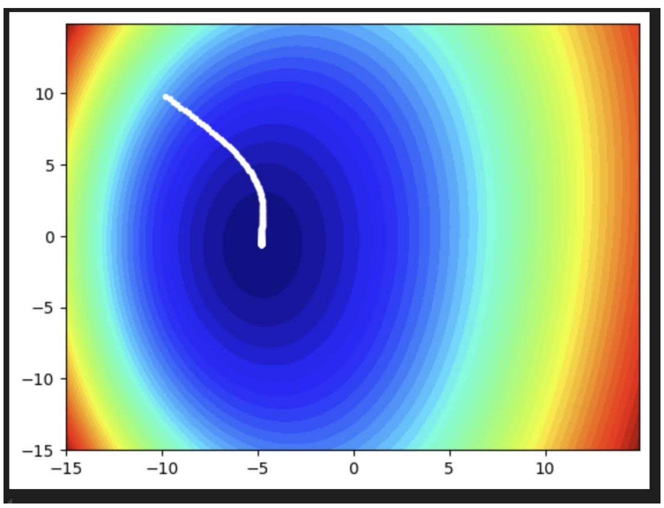

```
>0 f([-10, 10]) = 12.79326
>1 f([np.float64(-9.923455569603734), np.float64(9.95736380585442)]) = 12.75514
>2 f([np.float64(-9.848160352202887), np.float64(9.915002971942744)]) = 12.71808
>3 f([np.float64(-9.774090021309888), np.float64(9.872915094497543)]) = 12.68205
>4 f([np.float64(-9.701220775138632), np.float64(9.831097744159049)]) = 12.64700
>5 f([np.float64(-9.629529327498998), np.float64(9.789548470603341)]) = 12.61290
>6 f([np.float64(-9.558992898564064), np.float64(9.748264806831767)]) = 12.57973
>7 f([np.float64(-9.489589205540028), np.float64(9.707244273141415)]) = 12.54745
>8 f([np.float64(-9.421296453266615), np.float64(9.666484380795636)]) = 12.51604
>9 f([np.float64(-9.354093324773464), np.float64(9.62598263541271)]) = 12.48546
>10 f([np.float64(-9.287958971815902), np.float64(9.585736540089943)]) = 12.45569
>11 f([np.float64(-9.222873005411536), np.float64(9.545743598279643)]) = 12.42670
>12 f([np.float64(-9.158815486397168), np.float64(9.506001316432581)]) = 12.39848
>13 f([np.float64(-9.095766916023756), np.float64(9.466507206423778)]) = 12.37098
>14 f([np.float64(-9.033708226605473), np.float64(9.42725878777463)]) = 12.34420
>15 f([np.float64(-8.972620772237319), np.float64(9.388253589684698)]) = 12.31810
>16 f([np.float64(-8.912486319594258), np.float64(9.349489152885685)]) = 12.29267
>17 f([np.float64(-8.853287038823478), np.float64(9.310963031329466)]) = 12.26789
>18 f([np.float64(-8.795005494540066), np.float64(9.272672793721362)]) = 12.24373
>19 f([np.float64(-8.73762463693519), np.float64(9.234616024909174)]) = 12.22017
>20 f([np.float64(-8.681127793004771), np.float64(9.196790327137892)]) = 12.19720
>21 f([np.float64(-8.62549865790559), np.float64(9.159193321179409)]) = 12.17480
>22 f([np.float64(-8.570721286444792), np.float64(9.121822647345999)]) = 12.15295
>23 f([np.float64(-8.516780084707921), np.float64(9.084675966395775)]) = 12.13163
>24 f([np.float64(-8.46365980182976), np.float64(9.047750960337853)]) = 12.11084
...
>96 f([np.float64(-6.106976887536743), np.float64(6.850222246726362)]) = 11.35706
>97 f([np.float64(-6.087995250288421), np.float64(6.824783499856901)]) = 11.35204
>98 f([np.float64(-6.069264660312686), np.float64(6.799456186541181)]) = 11.34710
>99 f([np.float64(-6.050781603166108), np.float64(6.774239449270456)]) = 11.34223
```

Step size = 0.01




```
>0 f([-10, 10]) = 12.79326
>1 f([np.float64(-9.996172778480187), np.float64(9.99786819029272)]) = 12.79134
>2 f([np.float64(-9.992348685466665), np.float64(9.99573706837116)]) = 12.78943
>3 f([np.float64(-9.988527717906887), np.float64(9.993606633942111)]) = 12.78751
>4 f([np.float64(-9.984709872751576), np.float64(9.991476886712169)]) = 12.78560
>5 f([np.float64(-9.980895146954719), np.float64(9.98934782638773)]) = 12.78369
>6 f([np.float64(-9.977083537473563), np.float64(9.987219452674996)]) = 12.78179
>7 f([np.float64(-9.97327504126862), np.float64(9.985091765279972)]) = 12.77989
>8 f([np.float64(-9.969469655303655), np.float64(9.98296476390847)]) = 12.77799
>9 f([np.float64(-9.96566737654569), np.float64(9.98083844826611)]) = 12.77609
>10 f([np.float64(-9.961868201965), np.float64(9.978712818058328)]) = 12.77419
>11 f([np.float64(-9.958072128535107), np.float64(9.976587872990365)]) = 12.77230
>12 f([np.float64(-9.954279153232784), np.float64(9.974463612767277)]) = 12.77041
>13 f([np.float64(-9.950489273038043), np.float64(9.972340037093938)]) = 12.76853
>14 f([np.float64(-9.946702484934141), np.float64(9.970217145675033)]) = 12.76664
>15 f([np.float64(-9.942918785907572), np.float64(9.96809493821507)]) = 12.76476
>16 f([np.float64(-9.93913817294807), np.float64(9.965973414418375)]) = 12.76288
>17 f([np.float64(-9.935360643048597), np.float64(9.963852573989094)]) = 12.76101
>18 f([np.float64(-9.93158619320535), np.float64(9.961732416631195)]) = 12.75913
>19 f([np.float64(-9.927814820417753), np.float64(9.959612942048473)]) = 12.75726
>20 f([np.float64(-9.924046521688455), np.float64(9.957494149944546)]) = 12.75539
>21 f([np.float64(-9.920281294023328), np.float64(9.955376040022863)]) = 12.75353
>22 f([np.float64(-9.916519134431464), np.float64(9.953258611986694)]) = 12.75166
>23 f([np.float64(-9.912760039925175), np.float64(9.951141865539148)]) = 12.74980
>24 f([np.float64(-9.909004007519984), np.float64(9.949025800383161)]) = 12.74795
...
>96 f([np.float64(-9.646427267635723), np.float64(9.798440108455543)]) = 12.62071
>97 f([np.float64(-9.642886921129099), np.float64(9.796372965672331)]) = 12.61903
>98 f([np.float64(-9.639349424601505), np.float64(9.794306481829365)]) = 12.61735
>99 f([np.float64(-9.63581477530176), np.float64(9.792240656620853)]) = 12.61567
```

c) The values are changing very slightly in the last few iterations. Judging from this and the graph, the algorithm has practically converged to the minimal objective value.




```
best_beta, best_intercept
✓ 0.0s
(np.float64(0.003389351811036658 5), np.float64(-0.03856687277555937))
```

d) Adam's result better fits to the data points. The model levels off near the more extreme values of x, which better represents a probability distribution.