Homework 2

Jack Hart
Collaborators: Arishia Singh, Norman Hong, Shera Ning

Question 1: Convexity

Part 1. $f(x) = \sum_{i=1}^{\infty} ||x||_p ; p > 0$

It is known by their definition that norms which are positive-definite, scalable also subject to the triangle inequality: $||x_1 + x_2|| \leq ||x_1|| + ||x_2||$.

We can see this relationship from the triangle equality and the definition of convexity:

$$||\lambda x_1 + (1-\lambda)x_2|| \leq \lambda ||x_1|| + (1-\lambda)||x_2||$$

Let's take a case where $n = 2$ and the corresponding $x = 1$. The inequality above can be written out as follows:

$$\begin{aligned} ||1+1||_p &\leq ||1||_p + ||1||_p \\ 2^{\frac{1}{p}} &\leq 1^{\frac{1}{p}} + 1^{\frac{1}{p}} \end{aligned}$$

Since **the definition of $f(x)$ is for all $p > 0$, this inequality does not hold when $0 < p < 1$. For instance, let $p = \frac{1}{3}$:**

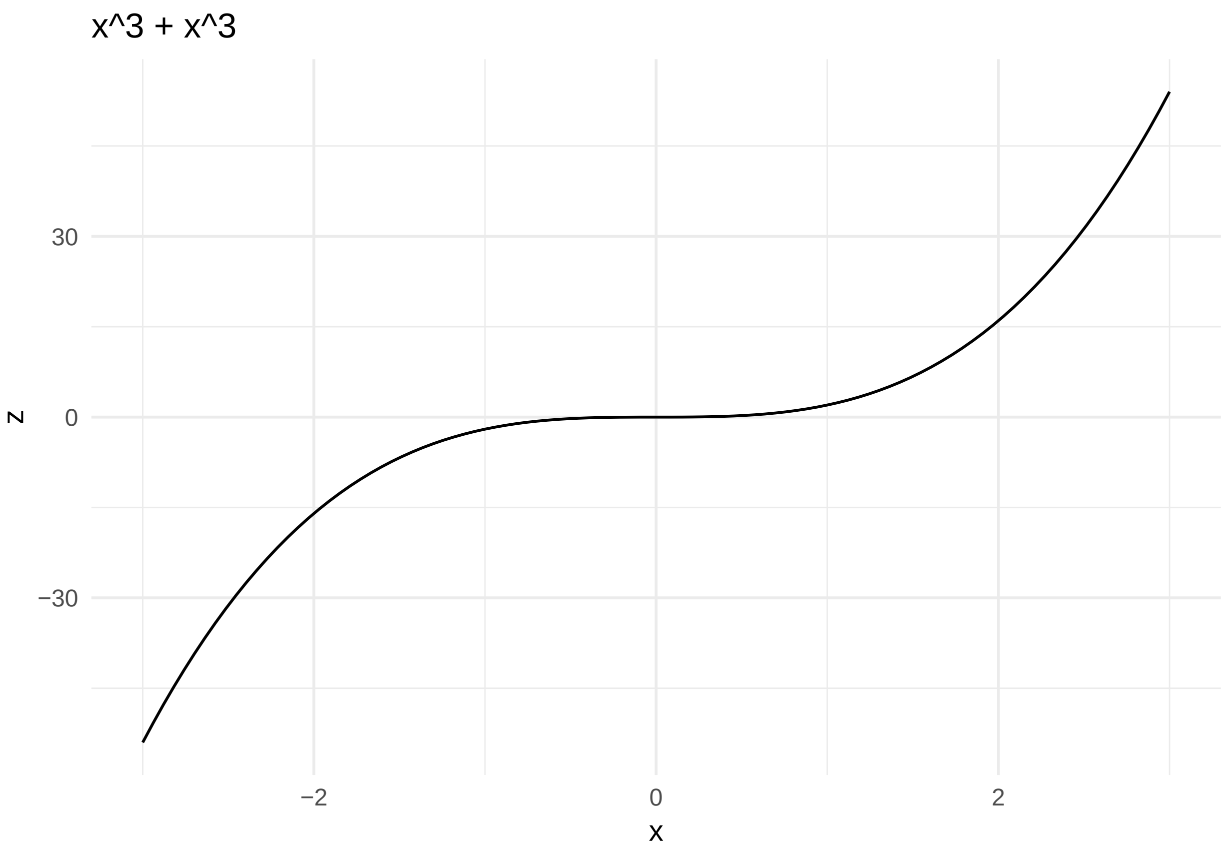
$$\begin{aligned} 2^{\frac{1}{\frac{1}{3}}} &\leq 1^{\frac{1}{\frac{1}{3}}} + 1^{\frac{1}{\frac{1}{3}}} \\ 2^3 &\leq 1^3 + 1^3 \\ 8 &\leq 2 \\ &\Rightarrow\Leftarrow \end{aligned}$$

Thus, when $0 < p < 1$, $||.||_p$ is not a norm by the definition above, and is thus not *guaranteed* to be convex. For example, for $p = \frac{1}{3}$ and $n = 2$, the equation for convexity is proven false with $x_1 = -1, x_2 = -\frac{1}{2}, \lambda = \frac{1}{2}$:

$$\begin{aligned} ||(\frac{1}{2})(-1) - (\frac{1}{2})(\frac{1}{2})||_{\frac{1}{3}} &\leq \frac{1}{2}|| - 1||_{\frac{1}{3}} + \frac{1}{2}|| - \frac{1}{2}||_{\frac{1}{3}} \\ (-\frac{3}{4})^3 &\leq -\frac{1}{2} + \frac{1}{2}(-\frac{1}{2})^3 \\ -\frac{27}{64} &\leq -\frac{9}{16} \\ -\frac{27}{64} &\leq -\frac{9}{16} \\ -0.421875 &\leq -0.5635 \\ &\Rightarrow\Leftarrow \end{aligned}$$

Therefore, we've found a function within the domain of p that is not convex.

For completeness, we can plot the function for this specific case and see that it is clearly not convex everywhere: $f(x) = ||x||_{\frac{1}{3}} + ||x||_{\frac{1}{3}}$



In conclusion , for the specific domain of p defined, $f(x)$ is not convex. However, if the domain of p was set to $p \geq 1$, then this would be a sum of convex functions, and thus would also be convex. This proof applies to all values of n up to ∞ because the triangle inequality can be applied to an infinite sum of values.

Part 2. $f(d) = k(x, x') - k'(x, x')$

Let's consider the known stationary and positive definite kernel:

$$k(x, x') = exp((x - x')^2) = exp(d^2), d = x - x'$$

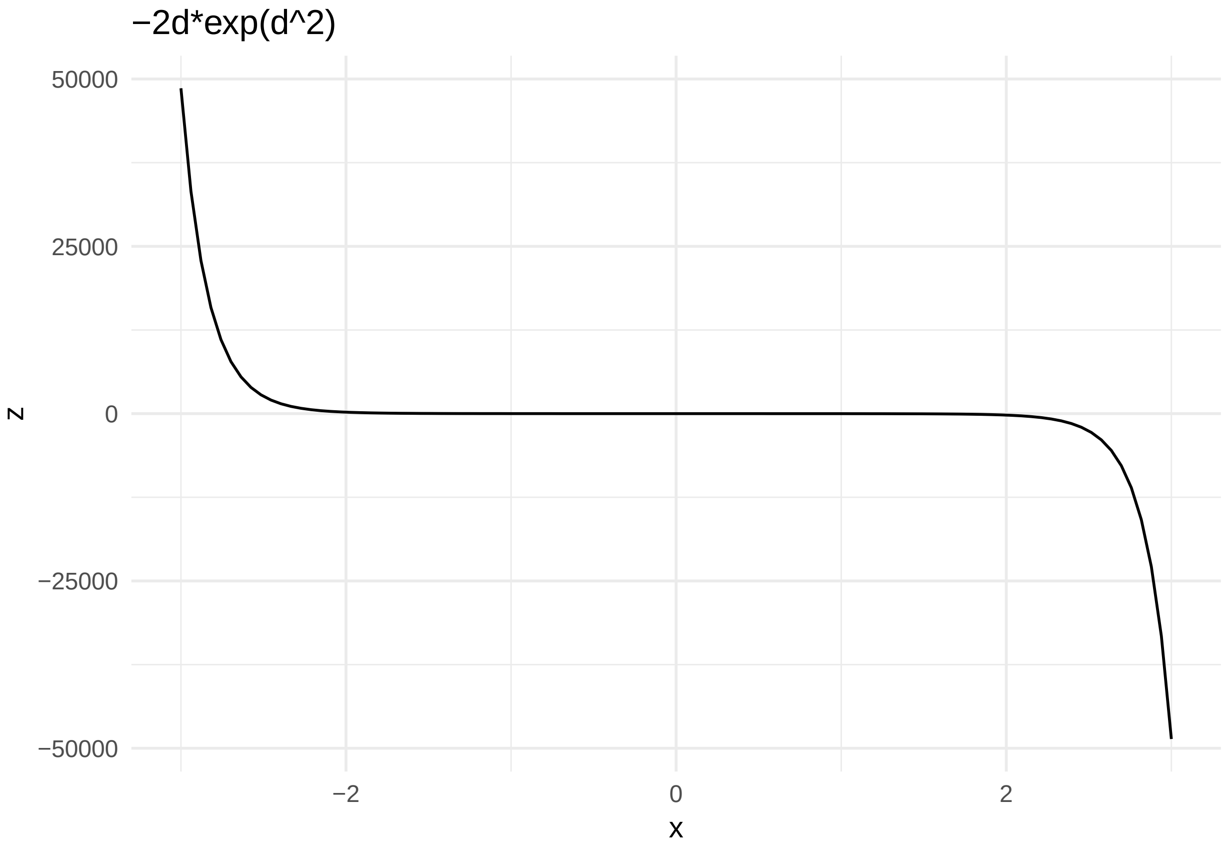
Therefore, we can derive $f(d)$ with the derivative of this kernel:

$$\begin{aligned} f(d) &= exp(d^2) - (2d)exp(d^2) \\ &= (-2d + 1)exp(d^2) \end{aligned}$$

The equation for convexity can be proven false with $d_1 = 0, d_2 = 1, \lambda = \frac{1}{2}$:

$$f(\frac{1}{2}d_1 + \frac{1}{2}d_2) \leq \frac{1}{2}f(d_1) + \frac{1}{2}f(d_2)$$
$$(-2(\frac{1}{2}) + 1)exp(\frac{1}{2}^2) \leq \frac{(-2(0) + 1)exp((0^2))}{2} + \frac{(-2(1) + 1)exp((1^2))}{2}$$
$$0 \leq \frac{1}{2} - \frac{e}{2}$$
$$0 \leq \frac{1-e}{2} \approx -0.86$$
$$\Rightarrow\Leftarrow$$

This proves that this example function is not convex, and therefore the general definition is not always convex. Again, we can also see that the function is not convex by plotting it.



Part 3. $f(d) = k(x, x') \times k'(x, x') - b$

Let's consider the known to be stationary and convex kernel:

$$k(x, x') = exp(-(x * x')) = exp(-d), d = x * x'$$

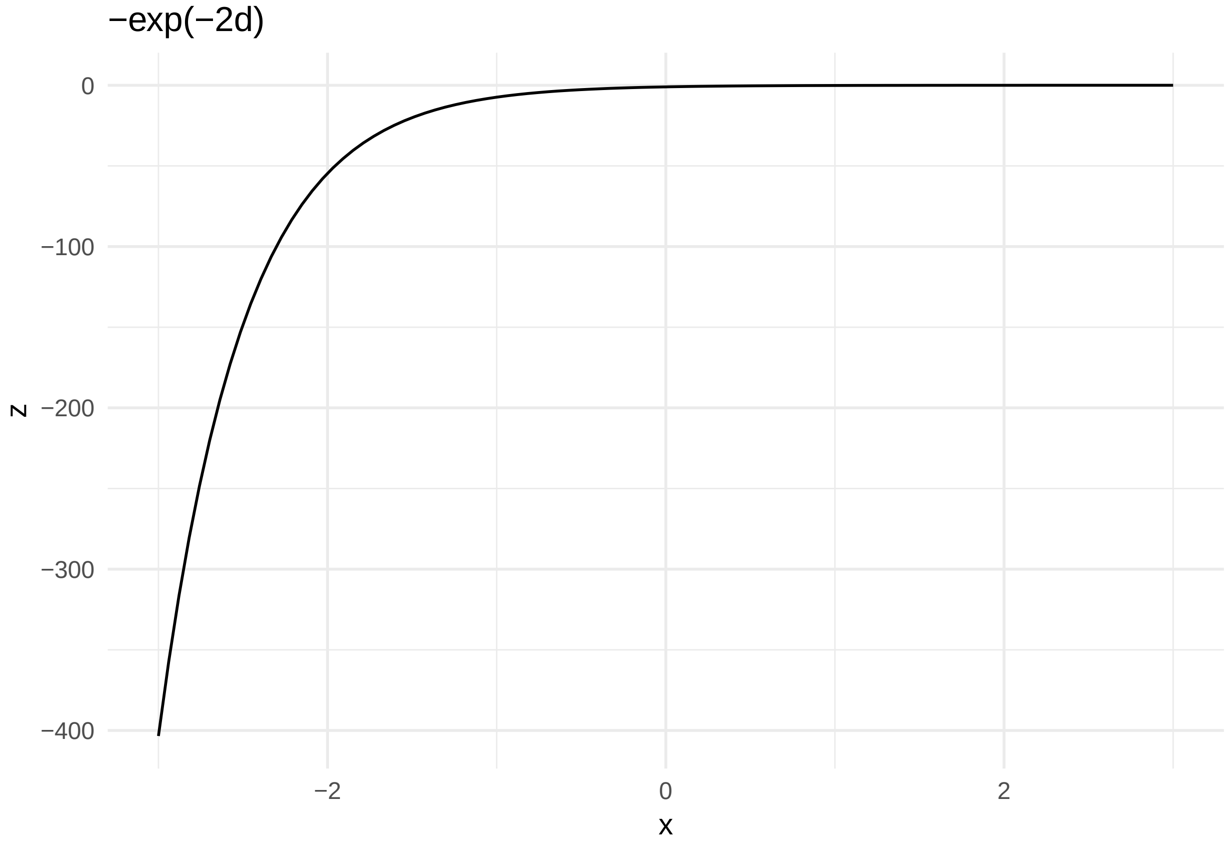
Therefore, we can derive $f(d)$ with the derivative of this kernel:

$$f(d) = exp(-d) * -exp(-d) - b = -exp(-2d) - b$$

The equation for convexity can be proven false with $b = 0, d_1 = -10, d_2 = 0, \lambda = \frac{1}{2}$:

$$f(\frac{1}{2}d_1 + \frac{1}{2}d_2) \le \frac{1}{2}f(d_1) + \frac{1}{2}f(d_2)$$
$$f(-5) \le \frac{1}{2}f(-10)$$
$$-exp(10) \approx -22026 \le \frac{-exp(20)}{2} \approx -242582597$$
$$\Rightarrow\Leftarrow$$

This proves that this example function is not convex, and therefore the general definition is not always convex. Again, we can also see that the function is not convex by plotting it.



Parts 4 & 5. $f(x) = ||x||_p - max(0, x); p > 0$

We proved in part one of this question that $||x||__p $ is not always convex when $0 < p < 1$. Therefore, the $max(0, x)$ for any x will result in a constant number, and will thus not change the curvature of the function. Therefore, **this function is also not convex**.

Question 2: Kernel Regression

Part 1:

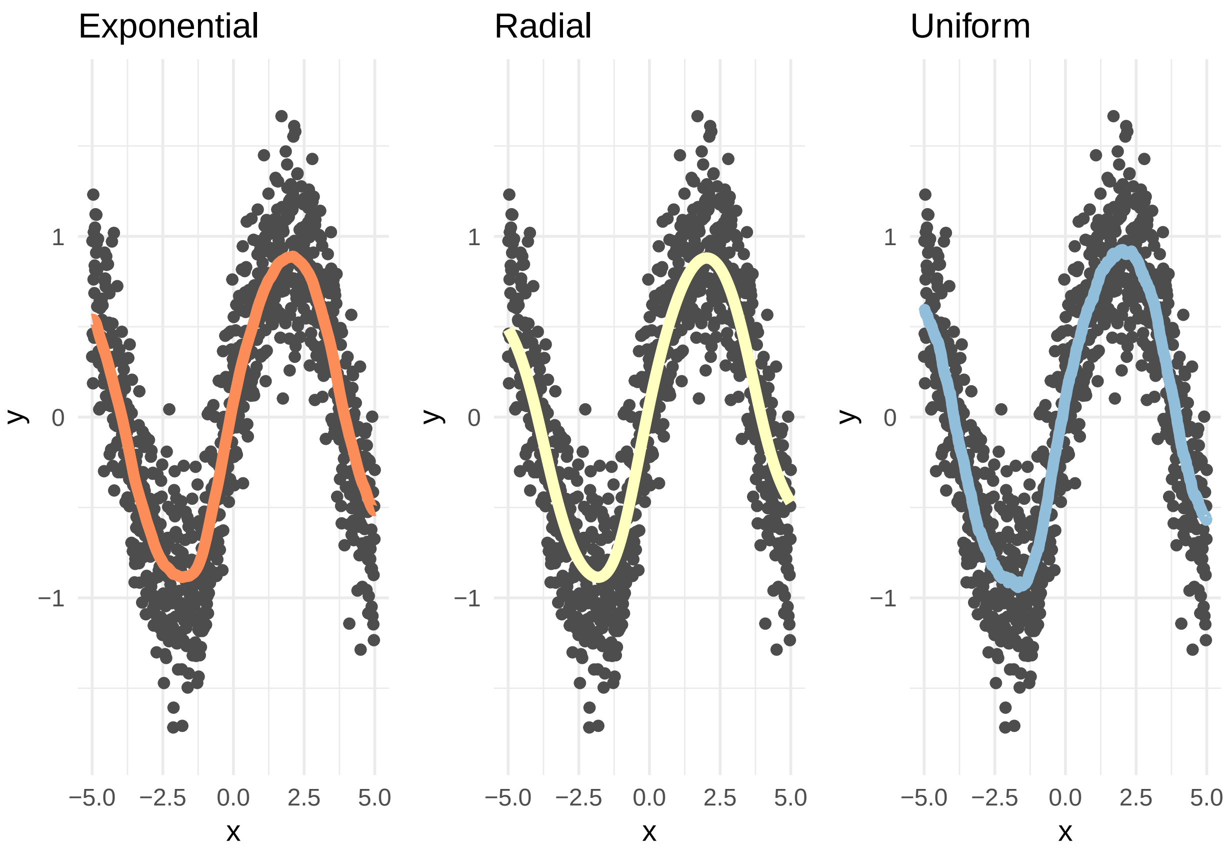
The following code is the basis for creating the kernel regressions for parts one and three. The documentation on this code explains how the regressions were implemented.

```r
create_kernel <- function(x, h = 1, kernel = "exponential"){
  #' Kernel Factory Function
  #'
  #' @param x number
  #' @param h number, the bandwidth
  #' @param kernel string, the type of kernel
  #' @return A function representing a kernel

  force(x) # force evaluation of x -- so it doesn't pass a closure to kernel function
  force(h)

  if(kernel == "exponential"){
    kernel_function <- function(x_new) return(exp(-3*abs(x_new-x)))
  } else if(kernel == "radial"){
    kernel_function <- function(x_new) return(exp((-2*sqrt((x_new-x)^2)^2) / (2*h^2 / 2)  )  )
  } else if(kernel == "uniform"){
    kernel_function <- function(x_new) return( ifelse(abs(x_new-x)<0.5,1,0) )
  } else {
    print("ERROR: kernel not found.")
  }
  return(kernel_function)
}

create_kernels <- function(x_vec, h = 1, kernel = "exponential"){
  #'  Kernels Function
  #'
  #' @param x_vec list, a list of list of features to build kernels off of
  #' @param h number, the bandwidth
  #' @param kernel string, the type of kernel
  #' @return A list of kernels for each feature

  # Create kernels from the data
  kernels = list()
  j = 1
  for(feature in x_vec){
    feature_kernels <- c()
    for(i in 1:length(feature)){
      new_kernel <- create_kernel(x=feature[[i]], h = h, kernel = kernel)
      feature_kernels <- c(feature_kernels, new_kernel)
    }
    kernels[[j]] = feature_kernels
    j = j + 1
  }
  return(kernels)
}


kernel_regression <- function(x, kernels, y_vecs=y_vec){
  #' Kernel Regression function on some x
  #'
  #' @param x list, a list of data points
  #' @param kernels list, a list of lists of kernel functions
  #' @param y_vecs list, a list of all y values from training data
  #' @return The mean estimate for a kernel regression

  # ** assert length of kernels and x are equal -- to do
  # ** assert length of a kernels and y_vecs are equal -- to do
    kernel_sums <- 0
    kernel_y_sums <- 0

    for(j in 1:length(y_vecs)){
      i_x = 1
      kernel_outputs <- c()
      for(kernels_x_i in kernels){
          kernel_outputs <- c(kernel_outputs, kernels_x_i[[j]](x_new=x[i_x]))
          i_x = i_x + 1
      }
      kernel_output <- prod(kernel_outputs) # this only applies in multivariate case
      kernel_sums <- kernel_sums + kernel_output
      kernel_y_sums <- kernel_y_sums + (kernel_output*y_vecs[j])
    }

    mean_estimate <- kernel_y_sums / kernel_sums
    return(mean_estimate)
}
```

The code below imports the data and creates an **Exponential, Radial Basis, and Uniform** kernel regression from the methods defined above.

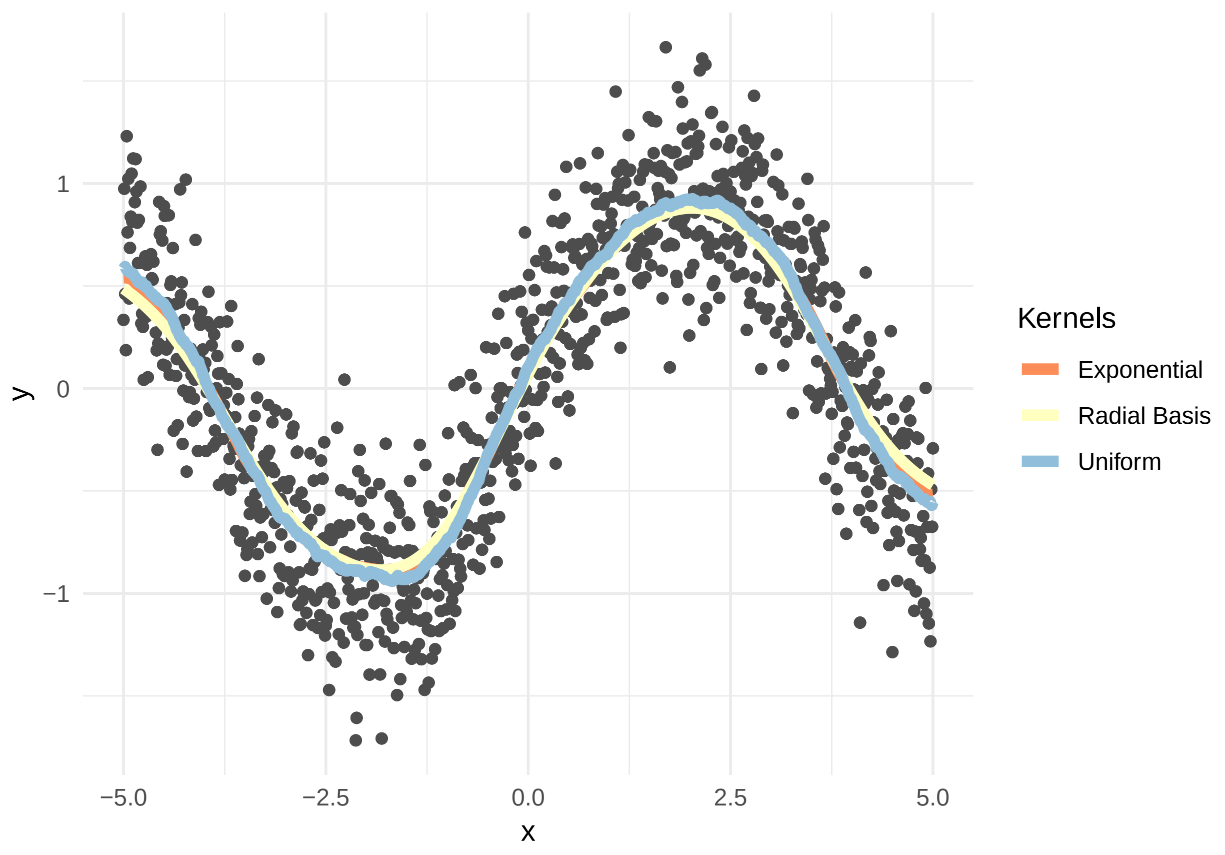
```
#here("data/kernel_regression_1.csv") -- doesn't work when knitting
kernel_regression_1 <- read.csv(paste0(DATA_DIR,"kernel_regression_1.csv"))

#create kernels
exp_kernels <- create_kernels(list(kernel_regression_1$x), kernel = "exponential")
radial_kernels <- create_kernels(list(kernel_regression_1$x), kernel = "radial")
uniform_kernels <- create_kernels(list(kernel_regression_1$x), kernel = "uniform")

#apply regression
kernel_regression_1$y_hat_exp <- sapply(kernel_regression_1$x, function(x) kernel_regression(x, exp_kern
kernel_regression_1$y_hat_radial <- sapply(kernel_regression_1$x, function(x) kernel_regression(x, radia
kernel_regression_1$y_hat_uniform <- sapply(kernel_regression_1$x, function(x) kernel_regression(x, uni
```

These are the results from the various kernel regressions, plotted on top of the data.



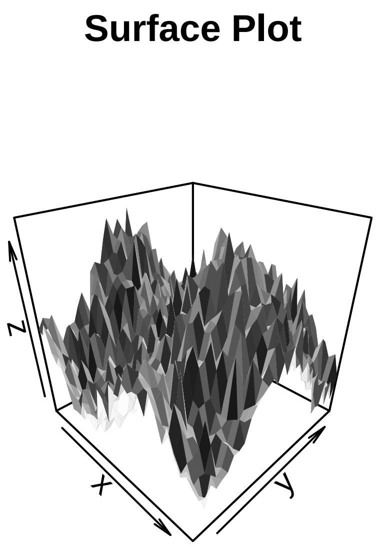
Part 2:

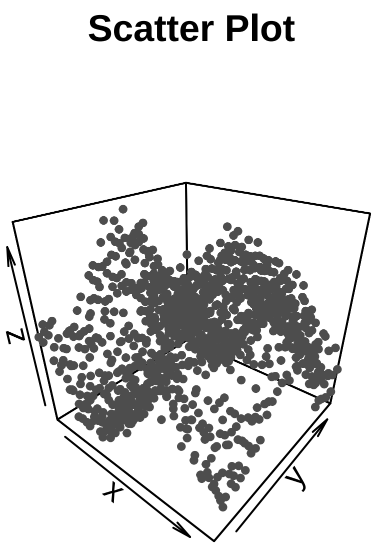
The above graphs of the kernels look very similar. The graph below plots them all together. The regression results look pretty consistent for all kernels in two dimensions, and all kernels fit the data well. As you can see, the largest differences between kernels are at the edges of the domain of x. This is consistent with the fact that *kernel regressions suffer from poor bias at the boundaries of the domain of inputs.*


Kernels
Exponential
Radial Basis
Uniform
y
x
1
0
−1
−5.0
−2.5
0.0
2.5
5.0

Part 3:

First let's look at the surface we are estimating / the data. This is the data we will be fitting.



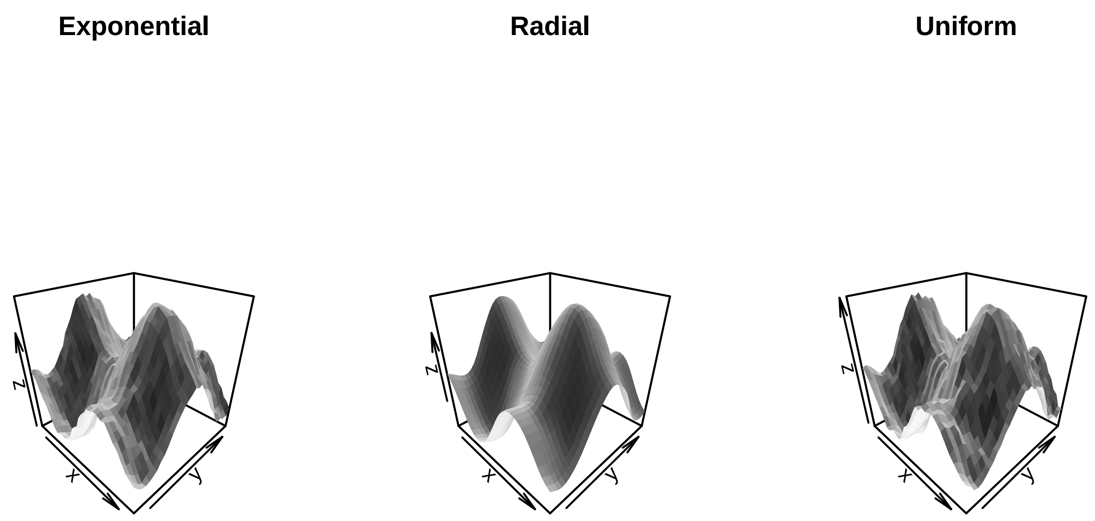


1. Use Kernels to create a smoothed surface

The following code implements the functions above for the data.

```
#create kernels
exp_kernels <- create_kernels(list(kernel_regression_2$x, kernel_regression_2$y), kernel = "exponential
radial_kernels <- create_kernels(list(kernel_regression_2$x, kernel_regression_2$y), kernel = "radial")
uniform_kernels <- create_kernels(list(kernel_regression_2$x, kernel_regression_2$y), kernel = "uniform

#apply regression
x_features <- cbind(kernel_regression_2$x, kernel_regression_2$y)
kernel_regression_2$y_hat_exp <- apply(x_features, 1, function(x) kernel_regression(x, exp_kernels, ker
kernel_regression_2$y_hat_radial <- apply(x_features, 1, function(x) kernel_regression(x, radial_kernel
kernel_regression_2$y_hat_uniform <- apply(x_features, 1, function(x) kernel_regression(x, uniform_kern
```

The following surfaces are created from the three kernels. The Radial basis kernel creates the smoothest surface, followed by the exponential, then uniform.



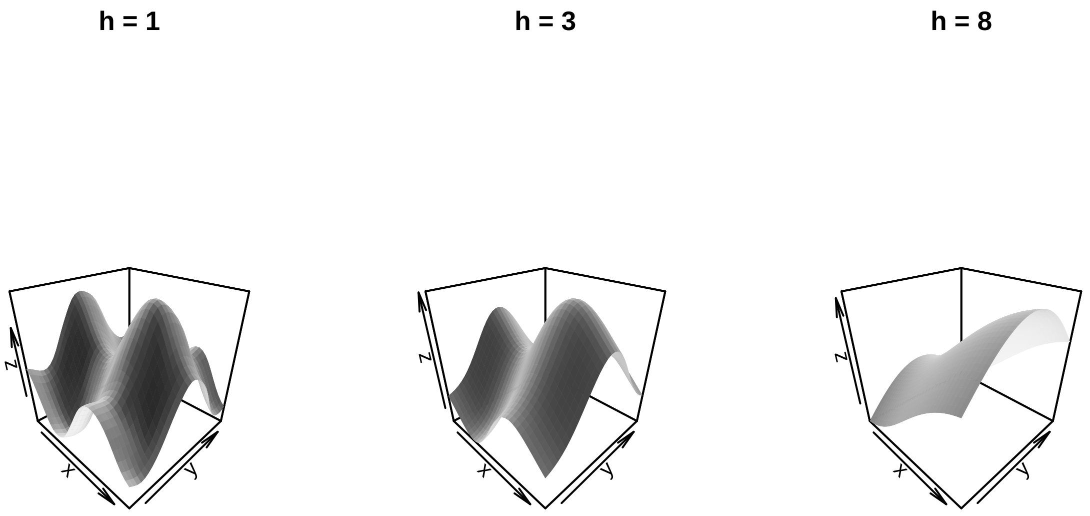
2. What happens when we change the bandwidth

For the previous sections of this question, the bandwidth (h) was set to one. Given the formula of the Radial Basis Kernel we can rewrite the formula in terms of h as:

$$k(x_i, x_j) = exp(-\frac{2||x_i - x_j||_2^2}{2h^2/2})$$

Here are plots comparing the surfaces when we set $h = 3$ and $h = 8$. First is the code that implements the functions above to apply the bandwidth. **As bandwidth increases the regression fits the data less closely**. Smaller bandwidth values will allow the kernels to span longer distances for any given data point, increasing the variance of the output. This can often cause the function to overfit. On the other hand, larger bandwidth values can over-smooth the output, and thus not reveal the true relationship. This is an example of **bias-variance trade-off**.

```
#create kernels
radial_kernels_3 <- create_kernels(list(kernel_regression_2$x, kernel_regression_2$y), h = 3, kernel =
radial_kernels_8 <- create_kernels(list(kernel_regression_2$x, kernel_regression_2$y), h = 8, kernel =

#apply regression
x_features <- cbind(kernel_regression_2$x, kernel_regression_2$y)
kernel_regression_2$y_hat_radial_3 <- apply(x_features, 1, function(x) kernel_regression(x, radial_kern
kernel_regression_2$y_hat_radial_8 <- apply(x_features, 1, function(x) kernel_regression(x, radial_kern
```



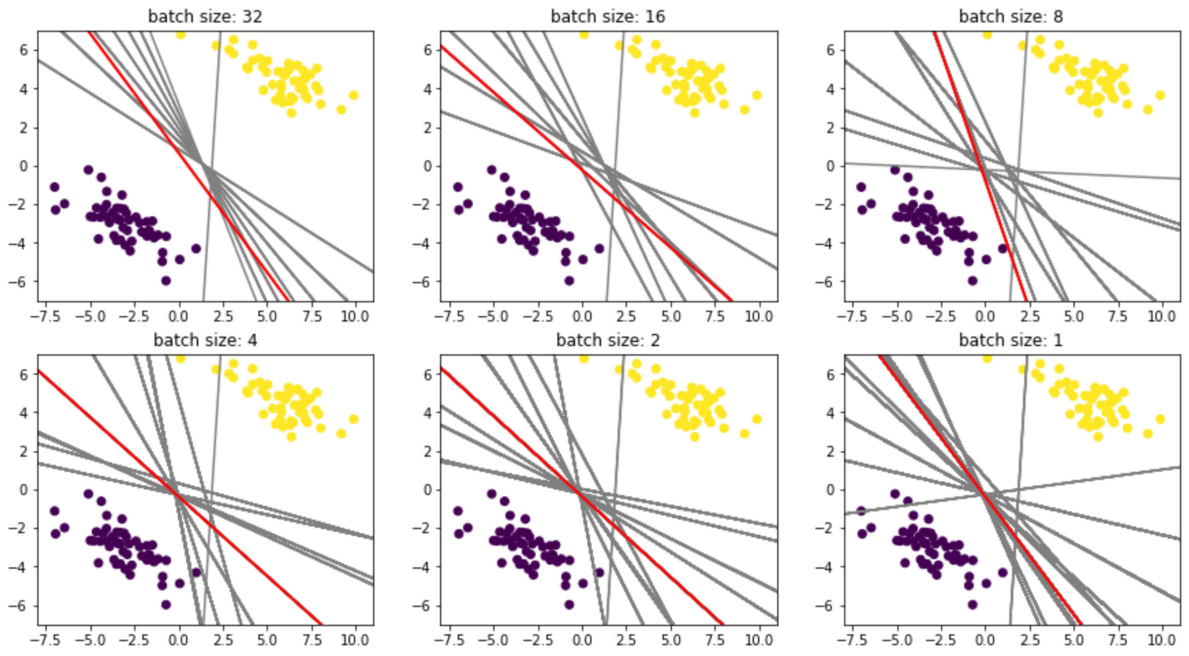
Question 3: Programming: Stochastic Subgradient Descent

As n grows large, quadratic programing used to optimize SVMs can become very computationally expensive. In the Python notebook `problem_3.ipynb`, I implement an SVM using subgradient descent as an alternative. The code is documented and the implementation of the code is explained in the notebook.

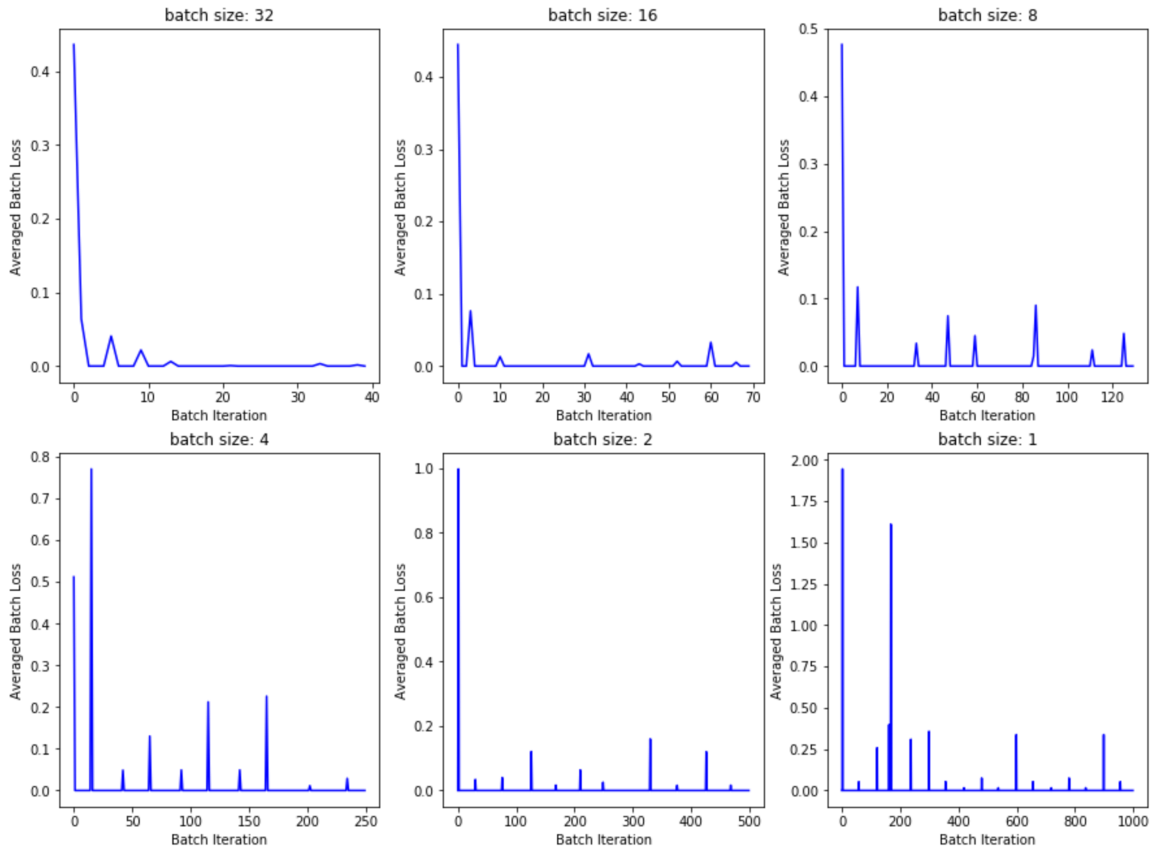
This code modifies to original algorithm to run faster by:

1. Eliminating for loops with vector multiplication with numpy when possible.
2. Extending the online algorithm for the batch setting.

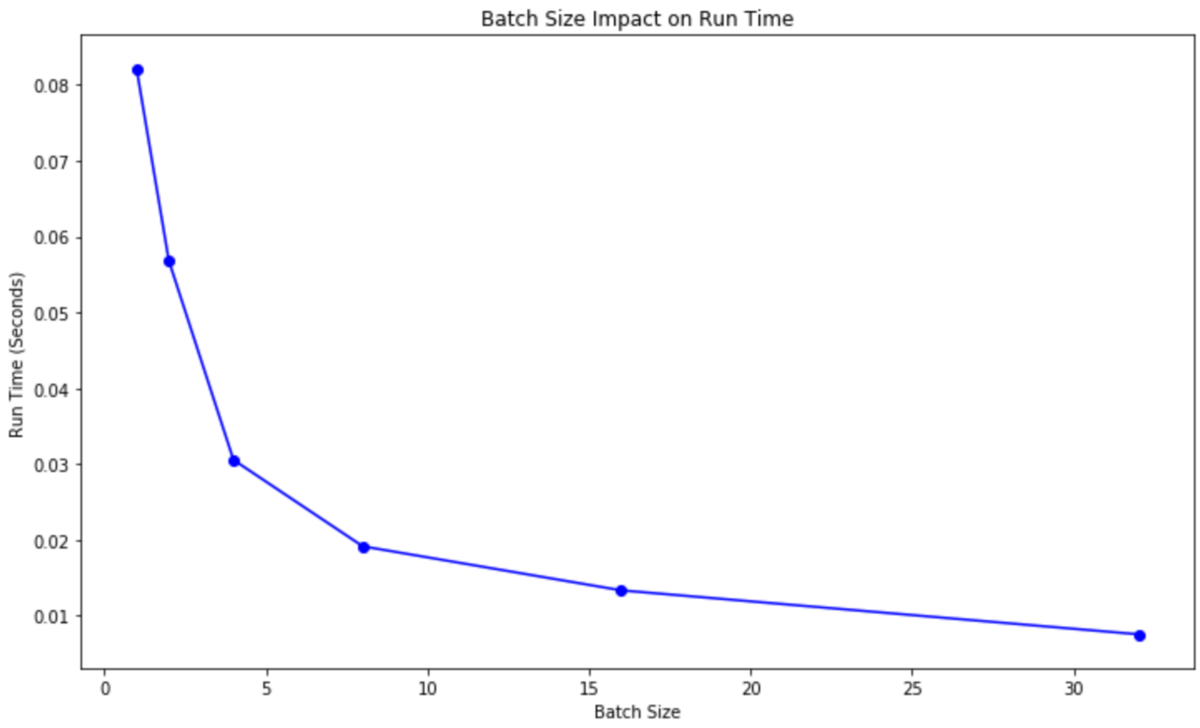
The following plot shows the final separating boundaries (plotted in red), as well as prior decision boundaries produced while still training, for various batch sizes. The plot with batch size set to one would be similar to the original implementation. The subsequent boundaries for when batch size is small have a large variance. Conversely, when batch size is larger, the updates are averaged, and the variance in subsequent decision boundaries is smaller (e.g. when batch size is 35). Most of the variance, however, comes from updates from the regularization term, since the hyperplane is usually not miss classifying data.



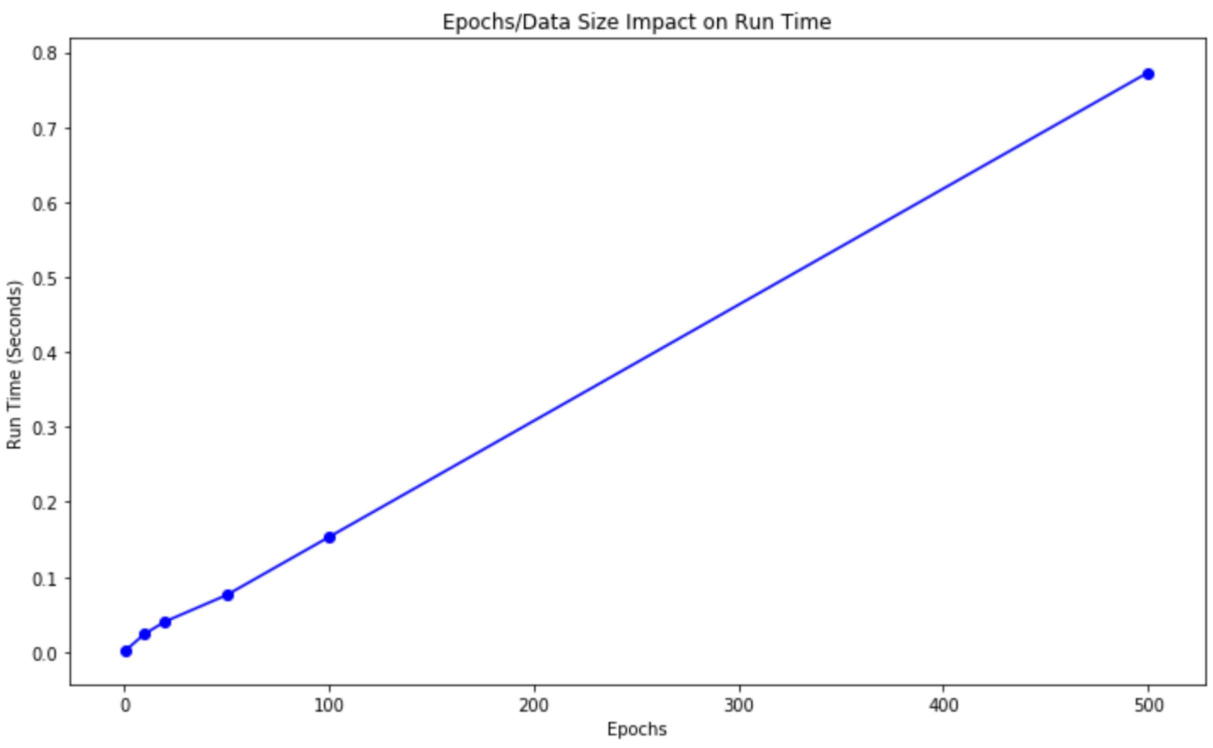
The following are loss plots for the models created above. The model is quickly trained within the first few iterations. The resulting spikes in loss occurs because the model is getting over regularized, then misclassifying an example, then correcting itself.



The next plot shows that implementing batching improves the run-time of this model (as was originally intended). As batch size increases, the total time required to train decreases.



This plot shows the benefit of subgradient descent over quadratic programming when n is large. As n increases (which was implemented here through epochs), the run time increases linearly. This is substantially better to the O(n^2) and O(n^3) time complexities of quadratic optimization.



Question 4: Calculating the Conjugate Distributions

Part 1

For clarification, the distributions we are calculating from the following distributions. *I rearranged some of the variable notation in order to utilize the properties of tau.*

$$x \sim N(\mu, \sigma^2)$$
$$\mu \sim N(v, \tau)$$
$$\sigma^2 \sim IG(\beta, \alpha)$$

Because the prior distributions are independent, the posterior can be written as:

$$f(\mu, \sigma^2|X) = f(X|\mu, \sigma^2)\pi(\mu)\pi(\sigma^2)$$

Next, we will write out this entire posterior. As a note, $\mu \sim N(v, \tau)$ and $\tau = \frac{1}{\sigma^2}$. Therefore, it is equivalent to still write the distribution in the form of σ_o^2, which will also be a constant.

$$f(\mu, \sigma^2|X) \propto \frac{1}{\sqrt{2\pi\sigma_o^2}} \frac{1}{\sqrt{2\pi\sigma^2}^n} exp(\frac{-(\sum_i^n x_i^2 - 2\mu\sum_i^n x_i + n\mu^2)}{2\sigma^2}) exp(\frac{-(\mu^2 - 2\mu v + v^2)}{2\sigma_o^2}) \frac{\beta^\alpha}{\Gamma(\alpha)(\sigma^2)^{-\alpha-1}} exp(\frac{-\beta}{\sigma^2})$$

This is a massive function. But traditional distributional forms can be found for each *conditional posterior* distribution. First, we will find the conditional posterior $\pi(\mu|X, \sigma^2)$. All parts of the equation above can be eliminated that do not contain μ, because they will be constants.

$$\begin{aligned}
\pi(\mu|X,\sigma^2) &\propto exp(\frac{-(\sum_i^n x_i^2 - 2\mu\sum_i^n x_i + n\mu^2)}{2\sigma^2})exp(\frac{-(\mu^2 - 2\mu v + v^2)}{2\sigma_o^2}) \\
&= exp(\frac{-\sum_i^n x_i^2\sigma_o^2 + 2\mu\sigma_o^2\sum_i^n x_i - n\mu^2\sigma_o^2 - \mu^2\sigma^2 + 2\mu\sigma^2 v - v^2\sigma^2}{2\sigma^2\sigma_o^2}) \\
&= exp(\frac{-\mu^2(\sigma^2 + n\sigma_o^2) + 2\mu(\sigma_o^2\sum_i^n x_i + v\sigma^2) - (\sum_i^n x_i^2\sigma_o^2 + v^2\sigma^2)}{2\sigma^2\sigma_o^2}) \\
&= exp(\frac{-\mu^2 + 2\mu\frac{(\sigma_o^2\sum_i^n x_i + v\sigma^2)}{\sigma^2 + n\sigma_o^2} - \frac{(\sum_i^n x_i^2\sigma_o^2 + v^2\sigma^2)}{\sigma^2 + n\sigma_o^2}}{\frac{2\sigma^2\sigma_o^2}{\sigma^2 + n\sigma_o^2}})
\end{aligned}$$

In order to properly square the final value in the numerator, the following transformation is applied and a constant is pulled out:

$$\begin{aligned}
\pi(\mu|X,\sigma^2) &\propto exp(\frac{-\mu^2 + 2\mu\frac{(\sigma_o^2\sum_i^n x_i + v\sigma^2)}{\sigma^2 + n\sigma_o^2} - (\frac{\sum_i^n x_i\sigma_o^2 + v\sigma^2}{\sigma^2 + n\sigma_o^2})^2}{\frac{2\sigma^2\sigma_o^2}{\sigma^2 + n\sigma_o^2}})exp(\frac{v^2\sigma^2 + \sigma_o^2\sum_i^n x_i^2}{2\sigma^2\sigma_o^2}) \\
&\propto exp(\frac{\mu - (\frac{\sum_i^n x_i\sigma_o^2 + v\sigma^2}{\sigma^2 + n\sigma_o^2})^2}{2\frac{\sigma^2\sigma_o^2}{\sigma^2 + n\sigma_o^2}}) \\
&= exp(\frac{\mu - (\frac{\bar{x}n\sigma_o^2 + v\sigma^2}{\sigma^2 + n\sigma_o^2})^2}{2\frac{\sigma^2\sigma_o^2}{\sigma^2 + n\sigma_o^2}})
\end{aligned}$$

We can rearrange the values here like so:

$$\frac{(\bar{x}n\sigma_o^2 + v\sigma^2)\frac{1}{\sigma^2\sigma_o^2}}{(\sigma^2 + n\sigma_o^2)\frac{1}{\sigma^2\sigma_o^2}} = \frac{\frac{v^2}{\sigma_o^2} + \frac{\bar{x}n}{\sigma^2}}{\frac{1}{\sigma_o^2} + \frac{n}{\sigma^2}}$$
$$\frac{\sigma^2\sigma_o^2}{\sigma^2 + n\sigma_o^2} = (\frac{1}{\sigma_o^2} + \frac{n}{\sigma^2})^{-1}$$

Thus we can say that:

$$\mu|X,\sigma^2 \sim N(\frac{\frac{v^2}{\sigma_o^2} + \frac{\bar{x}n}{\sigma^2}}{\frac{1}{\sigma_o^2} + \frac{n}{\sigma^2}}, (\frac{1}{\sigma_o^2} + \frac{n}{\sigma^2})^{-1})$$

Next, we can find the conditional posterior $\pi(\sigma^2|X,\mu)$. All parts of the equation for the joint posterior can be eliminated that do not contain σ^2, because they will be constants.

$$\begin{aligned}
\pi(\sigma^2|X,\mu) &\propto \frac{1}{\sqrt{2\pi\sigma^2}^n}exp(-\frac{\sum_i^n(x_i-\mu)^2}{2\sigma^2})\frac{\beta^\alpha}{\Gamma(\alpha)(\sigma^2)^{-\alpha-1}}exp(\frac{-\beta}{\sigma^2}) \\
&\propto \frac{1}{(\sigma^2)^{\frac{n}{2}}}\frac{1}{(\sigma^2)^{-\alpha-1}}exp(-\frac{\sum_i^n(x_i-\mu)^2}{2\sigma^2} - \frac{\beta}{\sigma^2}) \\
&= \frac{1}{(\sigma^2)^{-(\alpha+\frac{n}{2})-1}}exp(\frac{-(\beta + \frac{\sum_i^n(x_i-\mu)^2}{2}}{\sigma^2})
\end{aligned}$$

Thus, from this form we can see that:

$$\sigma^2|X,\mu \sim IG(\alpha + \frac{n}{2}, \beta + \frac{\sum_i^n(x_i-\mu)^2}{2})$$

Part 2.

We can write the posterior to be:

$$f(\mathbf{p}|X) = f(X|\mathbf{p})\pi(\mathbf{p})$$

This can be written out and shown that the posterior is the same form as the prior:

$$\begin{aligned} f(\mathbf{p}|X) &\propto \prod_{v=1}^{n}(\frac{n!}{\prod_{j=1}^{k} x_j!}\prod_{i=1}^{k} p_i^{x_{iv}})(\frac{\Gamma(\sum_{j=1}^{k}\alpha_i)}{\prod_{j=1}^{k}\Gamma(\alpha_i)}\prod_{i=1}^{k} p_i^{\alpha_i - 1}) \\ &= \frac{n!^n}{\prod_{v=1}^{n}\prod_{jv=1}^{k} x_j!}\frac{\Gamma(\sum_{j=1}^{k}\alpha_i)}{\prod_{j=1}^{k}\Gamma(\alpha_i)}\prod_{i=1}^{k} p_i^{\sum_v^n x_{iv}}\prod_{i=1}^{k} p_i^{\alpha_i - 1} \\ &\propto \prod_{i=1}^{k} p_i^{\sum_v^n x_{iv}}\prod_{i=1}^{k} p_i^{\alpha_i - 1} \\ &= \prod_{i=1}^{k} p_i^{\sum_v^n x_{iv} + \alpha_i - 1} \end{aligned}$$

Thus, with both α and x_i being vectors, we can see that:

$$\mathbf{p}|X \sim Dirichlet(\alpha + \sum_{i=1}^{n} x_i)$$

Part 3.

We can write the posterior to be:

$$f(\lambda|X) = f(X|\lambda)\pi(\lambda)$$

This can be written out and shown that the posterior is the same form as the prior:

$$\begin{aligned} f(\lambda|X) &\propto (\prod_{i=1}^{n}\frac{\lambda^{x_i}}{x_i!}e^{-\lambda})(\frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\alpha-1}exp(\beta\lambda)) \\ &= \frac{1}{\sum_{i=1}^{n} x_i}\frac{\beta^\alpha}{\Gamma(\alpha)}\lambda^{\sum_{i=1}^{n} x_i}exp(-n\lambda)\lambda^{\alpha-1}exp(-\beta\lambda) \\ &\propto \lambda^{\sum_{i=1}^{n} x_i + \alpha - 1}exp(-\lambda n - \beta\lambda) \\ &= \lambda^{(\bar{x}n+\alpha)-1}exp(-(n+\beta)\lambda) \end{aligned}$$

Thus we can see that:

$$\lambda|X \sim Gamma(\bar{x}n + \alpha, n + \beta)$$

Question 5: Priors as Regularizers

It is known that the logistic regression can be cast in terms of a **Bernoulli RV**. Since the Bernoulli distribution is part of the regular exponential family the following properties are known:

$$
\begin{aligned}
y &\sim Bernoulli(p) \\
p &= \frac{exp(\beta x)}{1 + exp(\beta x)} \\
\eta &= \beta x \\
\eta &= ln(\frac{p}{1-p})
\end{aligned}
$$

L2 Penalty

Let's assume $\beta\ N(0, \lambda^{-1})$, where $\lambda > 0$ and a constant. Therefore, for a regression with n observations:

$$
\begin{aligned}
f(\beta|Y) &= f(Y|\beta)\pi(\beta) \\
&\propto \prod_{i=1}^{n} p^{y_i}(1-p)^{(1-y_i)} \frac{1}{\sqrt{2\pi\lambda^{-1}}} exp(\frac{-(\beta)^2}{2\lambda^{-1}}) \\
&\propto p^{n\bar{y}}(1-p)^{(1-n\bar{y})} exp(\frac{-\beta^2}{2\lambda^{-1}})
\end{aligned}
$$

In a logistic regression the goal is often to minimize the negative log-likelihood in respect to p. The posterior defined above takes the form of the likelihood with an additional prior (which acts as a regularizer). We can take the negative log of this posterior and find that the term to be optimized contains an L2 loss term.

$$
\begin{aligned}
-l(p|Y) &= -(n\bar{y}ln(p) + (1-n\bar{y})ln(1-p) - \frac{\beta^2}{2\lambda^{-1}}) \\
&= -(n\bar{y}ln(\frac{exp(\beta x)}{1+exp(\beta x)}) + (1-n\bar{y})(ln(1 - \frac{exp(\beta x)}{1+exp(\beta x)}) - \frac{\lambda\beta^2}{2})
\end{aligned}
$$

From the above equation we can see that adding the normal prior created the log-likelihood equal to binary cross entropy with the additional term $\frac{\lambda\beta^2}{2}$, which is L2 loss.

L1 Penalty

A similar process can be taken to show the distribution that implements an L1 penalty when applied as a prior. Let's assume $\beta\ Laplace(0, \lambda^{-1})$, where $\lambda > 0$ and a constant. Therefore, for a regression with n observations:

$$
\begin{aligned}
f(\beta|Y) &= f(Y|\beta)\pi(\beta) \\
&\propto p^{n\bar{y}}(1-p)^{(1-n\bar{y})} \frac{1}{2\lambda^{-1}} exp(-\frac{|\beta|}{\lambda^{-1}}) \\
&\propto p^{n\bar{y}}(1-p)^{(1-n\bar{y})} exp(-\frac{|\beta|}{\lambda^{-1}})
\end{aligned}
$$

Again, in a logistic regression the goal is often to minimize the negative log-likelihood in respect to p. We can take the negative log of this posterior to find binary cross-entropy and a L1 loss term.

$$-l(p|Y) = -(n\bar{y}ln(p) + (1-n\bar{y})ln(1-p) - \frac{|\beta|}{\lambda^{-1}})$$
$$= -(n\bar{y}ln(\frac{exp(\beta x)}{1+exp(\beta x)}) + (1-n\bar{y})(ln(1-\frac{exp(\beta x)}{1+exp(\beta x)}) - \lambda|\beta|)$$

From the above equation we can see that adding the Laplace prior created the a log-likelihood equal to binary cross entropy with the additional term $\lambda|\beta|$, which is equivalent to L1 loss.

Question 6: General Questions

1. Posterior and Posterior Predictive

First, let's explain what a posterior distribution is. The posterior distribution is intended to give a probability of some θ, given our prior knowledge and new information about θ from the data. More specifically, we use a function that represents our prior knowledge of θ, and then we use another function that calculates the likelihood of θ given known data. The function representing prior knowledge is called the prior, and the other function representing the data is called the data generating function (but can also be interpreted as a likelihood function).

For completeness, the posterior distribution can be written like follows:

$$\text{Posterior} = \frac{\text{Prior * New Data}}{\text{Normalizing Constant}}$$
$$Pr(\theta|X = x_{new}) = \frac{Pr(\theta)Pr(\theta|X = x_{new})}{Pr(X = x_{new})} = \frac{\pi(\theta)L(\theta|X = x_{new})}{\int_\theta \pi(\theta)L(\theta|X = x_{new})d\theta}$$

After we've already derived a posterior distribution from our prior and data, we may be interested in predicting the probability of a new value for x, given our current knowledge. The **posterior predictive distribution is used to derive the probability of newly observed values given the posterior and a data generating function**. Intuitively, it makes sense to do this by multiplying the posterior by a data generating function given some new data point x. However, a key aspect of Bayesian statistics is that **the parameters are unknown/RV**, thus this product of the posterior and data generating function must be integrated.

For completeness, the posterior predictive distribution, the posterior predictive distribution can be written like follows:

$$\text{Posterior Predictive} = \text{Posterior * New Data, with independece from the parameter(s)}$$
$$Pr(X_{new} = x_{new}|X = x_{old}) = \int_\theta Pr(\theta|X = x_{old})Pr(X = x_{new}|\theta)d\theta$$

In summary, here are the key differences:

1. The posterior is created with new data from the prior, the posterior predictive is created when we already have a posterior and want to use it.

2.The posterior outputs the probability θ, given the observed data. The posterior predictive distribution outputs the probability of a possible value of x, averaged over all possible values of θ. **i.e. The posterior predicts the parameters, the predictive posterior predicts the data.**

Also you can note that the normalizing constant for the posterior is a very similar formula as the posterior predictive distribution. This makes sense given that this constant is supposed to represent the total likelihoods possible for all θ

2. Predicting Future Values of X

If no values of X have been observed, then we would use the prior $\pi(\theta)$ to generate values of X. Once data is observed, we want to use our new posterior distribution $Pr(\theta|X)$, which now incorporates our new knowledge from the data, to generate new values of X using Bayes theorem However, the actual value of θ is unknown, because in Bayesian statistics the parameters are random. Therefore, **we want to use the posterior predictive distribution**, which gives us an average likelihood of some data over all possible values of θ.

3. MAP and MLE

Assuming $X \sim N(\mu, \sigma^2), \sigma^2 \sim Gamma(\tau, v), \mu \sim N(\alpha, \beta)$, then we can derive the MAP estimates through their conditional posteriors

First, the posterior $\pi(\mu|\sigma^2, X)$ is known to have a normally distributed posterior distribution because both X and μ come from a normal distribution. We derived this distribution for a similar case in problem 4. Therefore, we can write the posterior as follows:

$$\mu|\sigma^2, X \sim N((\frac{1}{\frac{1}{\beta}+\frac{n}{\sigma^2}})(\frac{\alpha}{\beta}+\frac{\sum_{i=1}^n x_i}{\sigma^2}), (\frac{1}{\beta}+\frac{n}{\sigma^2})^{-1})$$

From this distribution we know that:

$$\mu_{MAP} = (\frac{1}{\frac{1}{\beta}+\frac{n}{\sigma^2}})(\frac{\alpha}{\beta}+\frac{n\bar{x}}{\sigma^2})$$

The limit for this as n approaches infinity will be equal to the MLE estimate.

$$\lim_{n\to\infty} \frac{(\frac{\alpha}{\beta}+\frac{n\bar{x}}{\sigma^2})}{(\frac{1}{\beta}+\frac{n}{\sigma^2})} = \lim_{n\to\infty} \frac{\frac{1}{n}(\frac{\alpha}{\beta}+\frac{n\bar{x}}{\sigma^2})}{\frac{1}{n}(\frac{1}{\beta}+\frac{n}{\sigma^2})}$$
$$= \lim_{n\to\infty} \frac{(\frac{\alpha}{\beta n}+\frac{\bar{x}}{\sigma^2})}{(\frac{1}{\beta n}+\frac{1}{\sigma^2})} = \frac{\frac{\bar{x}}{\sigma^2}}{\frac{1}{\sigma^2}} = \frac{\bar{x}\sigma^2}{\sigma^2} = \bar{x} = \mu_{MLE}$$

Next, the posterior $\pi(\frac{1}{\sigma^2}|\mu, X)$ is known to have a Gamma distribution because X comes from a normal distribution and σ^2 comes from a Gamma distribution. We also derived this distribution for a similar case in problem 4. Therefore, σ^2 will come from an Inverse Gamma distribution with the following form:

$$\sigma^2|\mu, X \sim IG(\tau+\frac{n}{2}, v+\frac{\sum_{i=1}^n (x_i-\mu)^2}{2})$$

Therefore, since the mean of an inverse gamma is known to be $\frac{v}{\tau-1}$, we can derive the MLE of the MAP by taking the limit as n approaches infinity.

$$\sigma^2_{MAP} = \frac{v+\frac{\sum_{i=1}^n (x_i-\mu)^2}{2}}{\tau+\frac{n}{2}-1}$$
$$\lim_{n\to\infty} \frac{(v+\frac{\sum_{i=1}^n (x_i-\mu)^2}{2})}{(\tau+\frac{n}{2}-1)} = \frac{\frac{1}{n}(v+\frac{\sum_{i=1}^n (x_i-\mu)^2}{2})}{\frac{1}{n}(\tau+\frac{n}{2}-1)}$$
$$= \lim_{n\to\infty} \frac{(\frac{v}{n}+\frac{\sum_{i=1}^n (x_i-\mu)^2}{2n})}{(\frac{\tau}{n}+\frac{n}{2n}-\frac{1}{n})}$$
$$= \frac{\frac{\sum_{i=1}^n (x_i-\mu)^2}{2n}}{\frac{1}{2}} = \frac{\sum_{i=1}^n (x_i-\mu)^2}{n} = \sigma^2_{MLE}$$

Question 7: Programming a Gibbs Sampler

In question 4 we found the posterior distribution for λ, which we can use here:

$$\lambda|X \sim Gamma(\bar{x}n + 30, n + 4)$$

Thus since this problem doesn't contain any hyper priors or other dependencies, *this is not a true Gibbs sampler*. Regardless, I go about the process/implement the algorithm of Gibbs sampling for this problem in `problem_y.ipynb`.

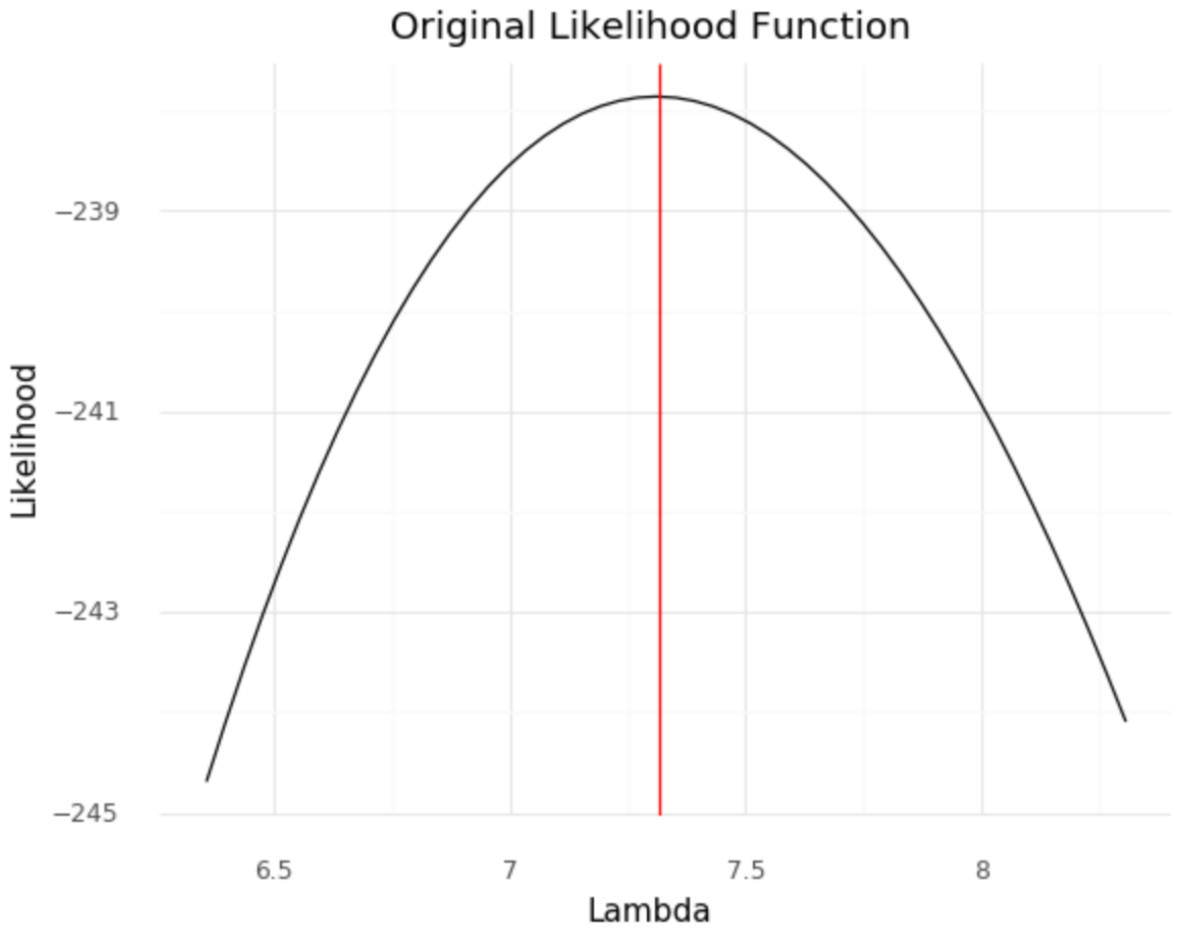
Likelihood

Since there are no dependencies, the likelihood of λ is just the Poisson distribution for our original sample:

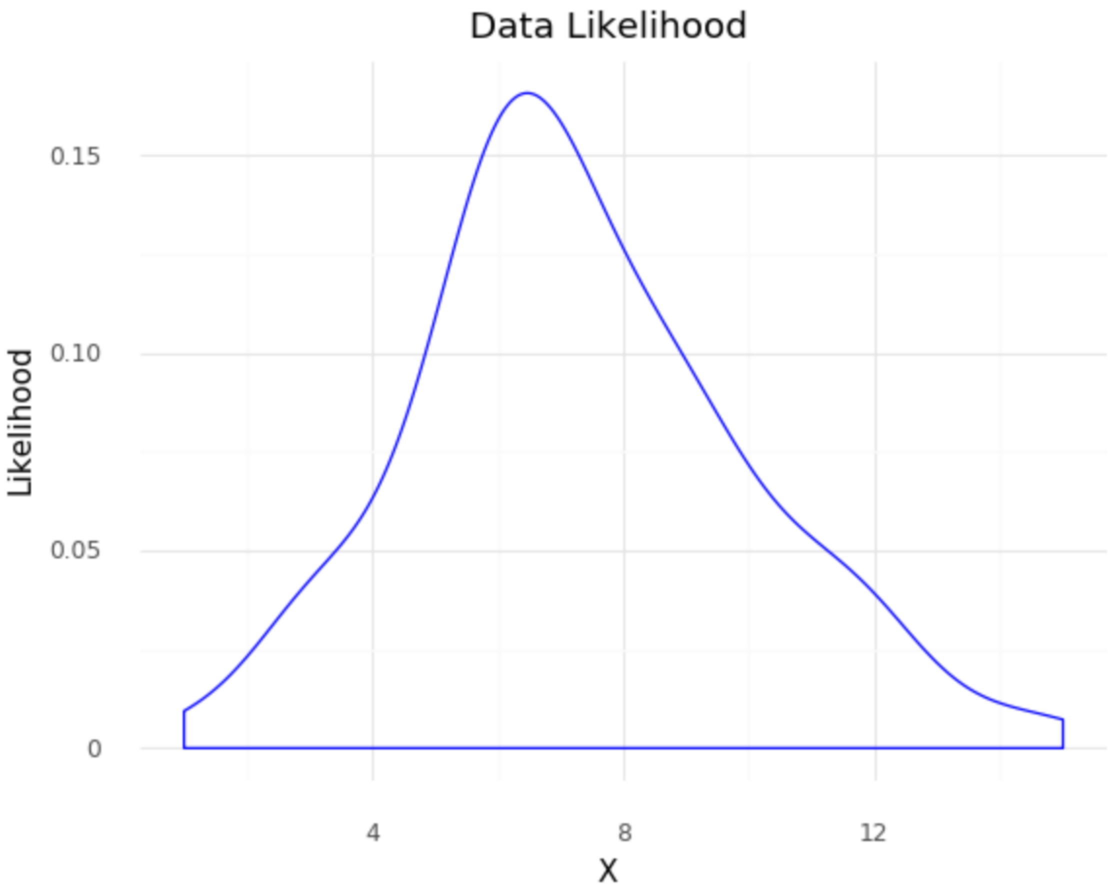
$$L(\lambda; X = x) = \prod_{i=1}^{n} \frac{\lambda^{x_i}}{x_i!} e^{-\lambda} = \frac{\lambda^{\bar{x}n}}{\sum_{i=1}^{n} x_i!} e^{-\lambda n}$$

$$l(\lambda, x_1, ..., x_n) = -\lambda n - \sum_{i=1}^{n} (\log(x_i!)) + \log(\lambda) \sum_{i=1}^{n} (x_i)]$$

The log likelihood based on our original sample can be generated using the formula above. The following is the likelihood for λ from the sample that created our prior estimate. Plotted in red is the MAP estimate of λ derived from the simulation, which is right where this likelihood is optimized.

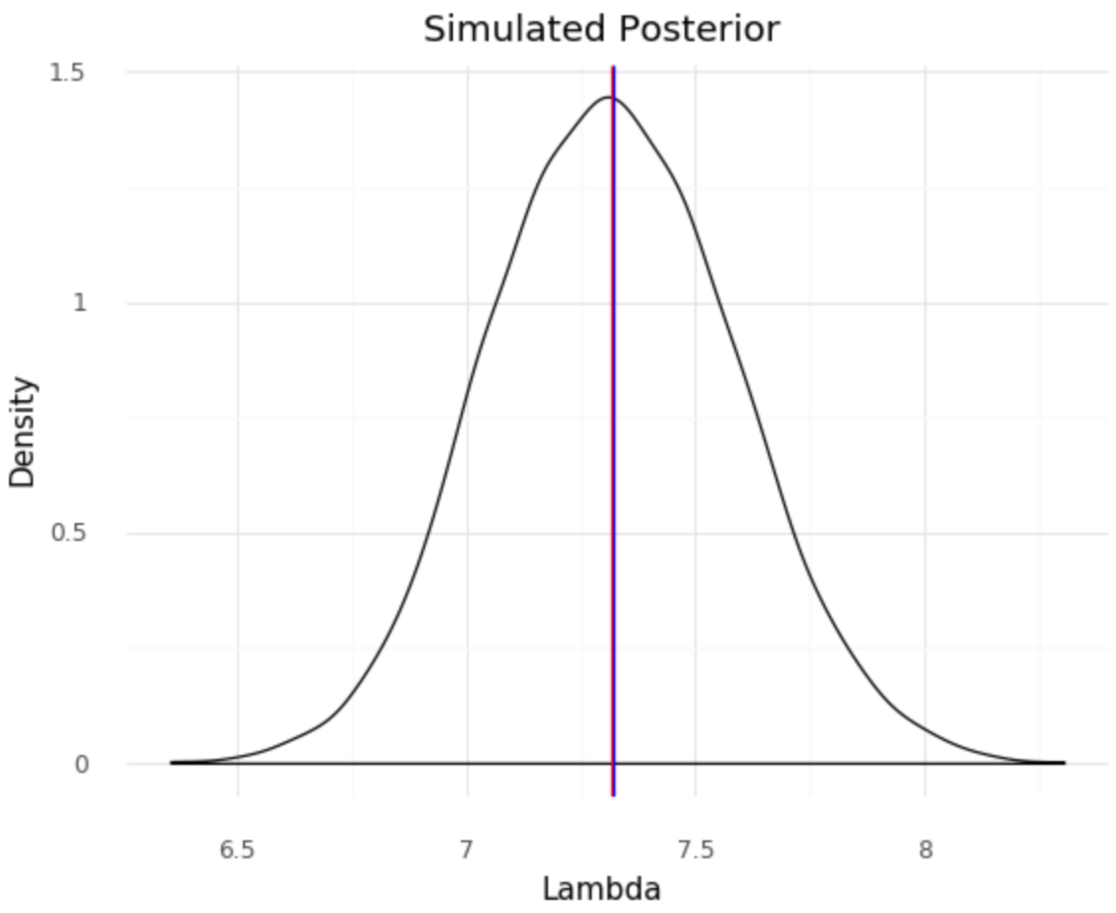


However this is not a true Bayesian Likelihood. In Bayesian statistics we are more interested in the likelihood of data, given some parameters. This can be seen here by just looking at the distribution of data in our original sample:



Posterior

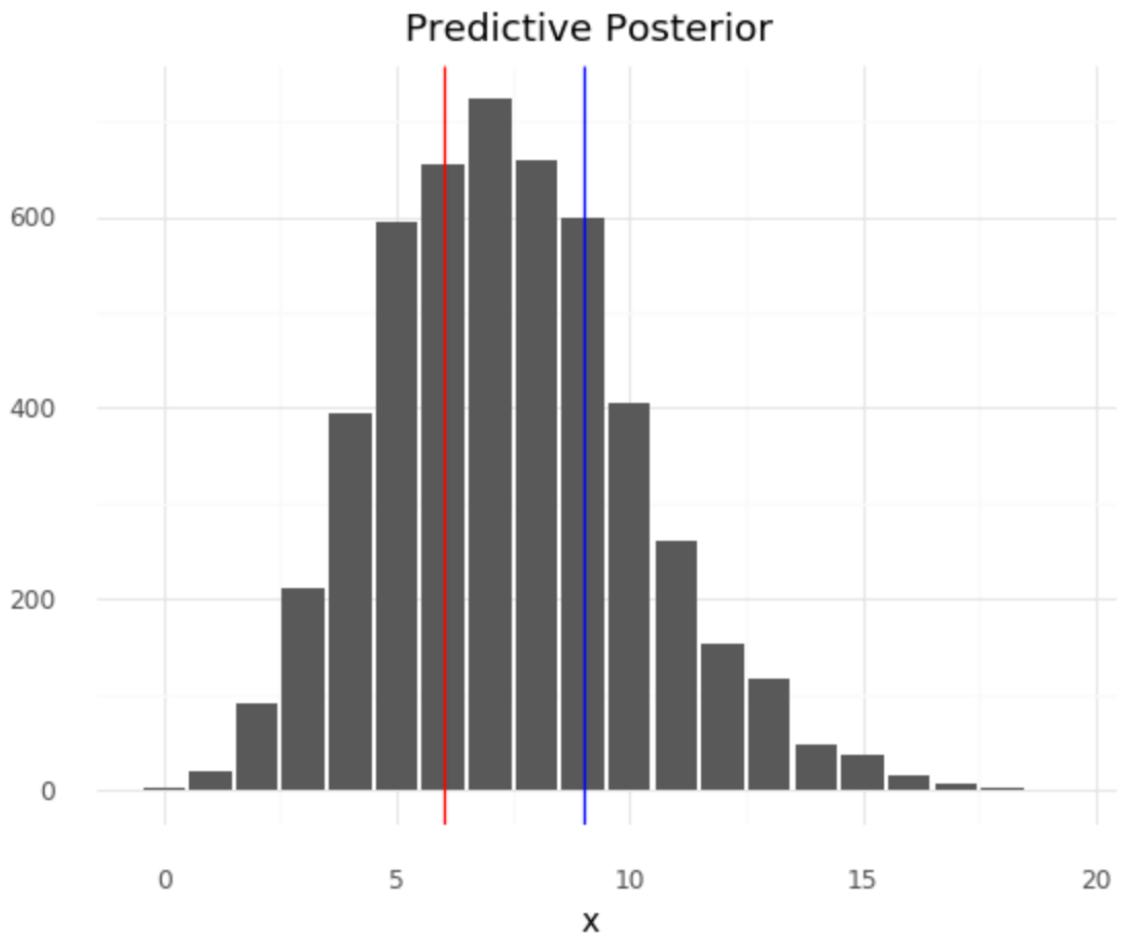
The posterior distribution was theoretically found to be: $\pi(\lambda|X) \sim Gamma(\bar{x}n + 30, n + 4)$. We can plot our simulated sample of λ to show our approximation of this distribution. One red and blue line was plotted to show the theoretical MAP and the MAP generated from our sample. They are almost identical.



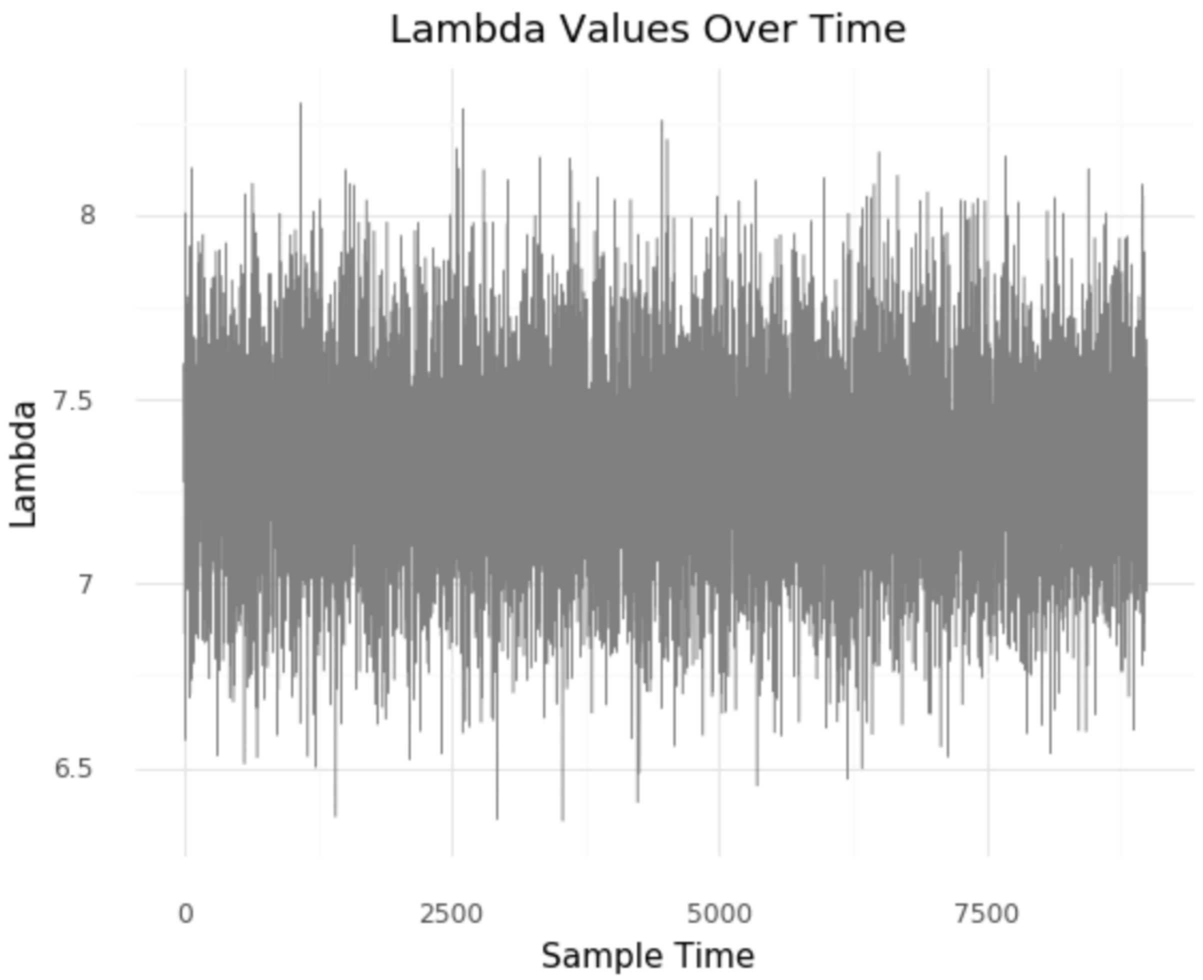
Posterior Predictive

The posterior predictive distribution can be simulated by taking many samples from the posterior distribution, and then getting a simulated x value for each λ value drawn. This will get us a good approximate of what the posterior predictive distribution looks like. With a true Gibbs sample, we would often need to sample another variable and hold it constant to determine the posterior predictive for another variable. But since this is a very simple case, that is not needed.

The red and blue lines represent two randomly sampled X values, and the corresponding place in the posterior predictive distribution. It's no surprise that the two values we randomly sampled are both near the peak of the distribution.

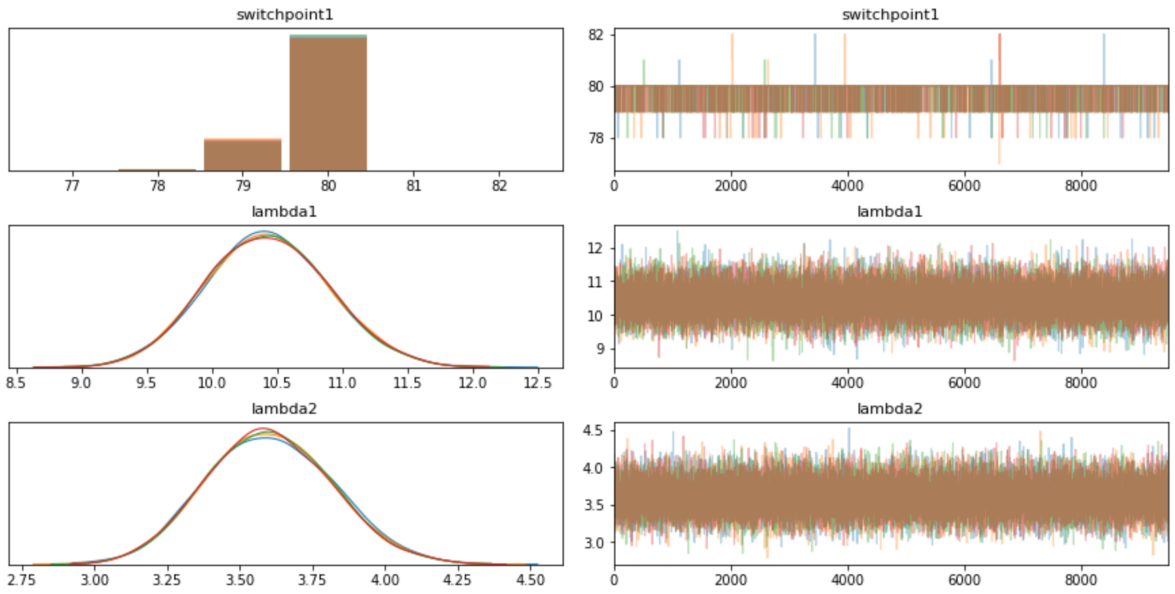


Since this is not a true Gibbs sampler, diagnostics are easier, given there are no additional parameters that could be correlated with λ. But for the sake of completeness, here's a plot of the predicted values of λ over each iteration. As expected, the values of lambda are random with respect to when they were sampled.

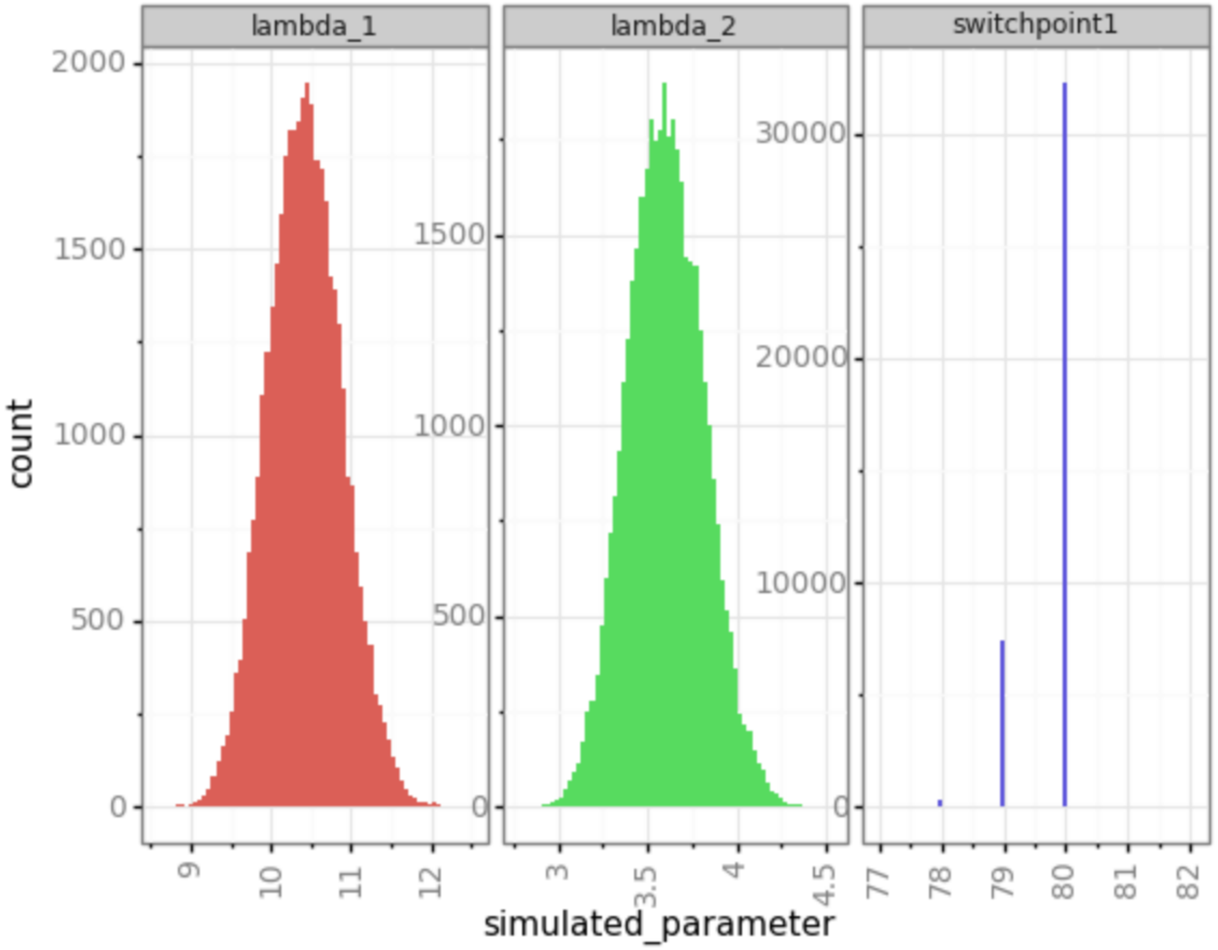


Question 8: Change Points Models

The code for creating the change point model is in `problem_8.ipynb`. From the following diagnostic plots, it seems clear that the model converged. This can be inferred because all chains found similar distributions, and a consistent, even range of values were sample during the entire sampling time.



Additionally, here are the posterior distributions for all the variables. The switchpoint appears to be 80.



Question 9: Programming a Hierarchical Model Using PYMC3

Skipping this question.

Interview Questions

1. Explain SVMs in Your Own Words

a. How are Kernels Introduced in SVMs

There is the **primal** quadratic optimization problem, that optimizes a functional form in terms of w, similar to what we worked with in problem 3:

$$min(||w||^2) + C\sum_{i=1}^{n} max(0, 1 - y_i(x_i))$$

SVMs can also be written in a different form and optimized in terms of alpha, this is called the **dual** optimization problem:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i (x_i^T x) = b$$

This dual functional form comes from that fact that w can always be written as a linear combination of the training data:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

At first it seems counterintuitive to use the dual form, because it would require you to pass over the entire dataset just to find the weights. The dual form, however, allows us to use kernel functions to tranform the feature space of the SVM. This process of applying kernels to the dual form has been coined the "kernel trick".

b. Slack Variable Penalty

Slack variables are the variables we allow to be misclassified by an SVM, since it is rarely the case that data can be linerally separable, even after using a kernel. The slack variable penalty is controlled by a constant mltipled by the equation we usually try to minimize:

$$C\sum_{i=1}^{n} z_i + \frac{1}{2}||w||^2$$

In the equation above (which we try to minimize), C controls the tradeoff between slack variables and the margin.

c. Slack Variables and the Radial Basis Kernel

The Radial Basis kernel is **invariant to translation**, or in otherwords it is a **stationary kernel**, like the kernels discussed in problem one. This property makes teh radial basis kernel unaffeted by slack variables.

d. Primal-Dual Relationship for Support Vectors

I discussed the primal-dual relationship above when talking abou how how the kernel was introduced to SVMs. When using support vectors machines, the α value will be zero for all but a few points, which are called teh **support vectors**. Thus these will be the only data points that influence the weights.

e. Computational Complexity of SVMs

SVMs are O(n^2) complexity in the best case scenario.

2. Exponential Family

The first step in this proof is taking the definition of a conjugate prior for the exponential family and applying Bayes rule to find a general form of the posterior. We know that the conjugate prior of an exponential family has the form:

$$\pi_{\boldsymbol{x_0},n_0}(\boldsymbol{\theta}) = \frac{exp(n_0\boldsymbol{x_0}^T\boldsymbol{\theta} - n_0A(\boldsymbol{\theta}))}{\int_\theta exp(n_0\boldsymbol{x_0}^T\boldsymbol{\theta} - n_0A(\boldsymbol{\theta}))}$$

Here n_0 can be thought of as the number of "examples" we say we have in our prior (i.e. how heavily we weight our priors), and x_0 is the observation that this prior represents. From this we can calculate the posterior for one new observation $\boldsymbol{x}$ using **Bayes Theorem**:

$$\begin{aligned}
p(\boldsymbol{x}|\boldsymbol{\theta})\pi_{\boldsymbol{x_0},n_0}(\boldsymbol{\theta}) &\propto exp(\boldsymbol{x}^T\boldsymbol{\theta} - A(\boldsymbol{\theta}))exp(n_0\boldsymbol{x_0}^T\boldsymbol{\theta} - n_0A(\boldsymbol{\theta})) \\
&= exp(\boldsymbol{x}^T\boldsymbol{\theta} - A(\boldsymbol{\theta}) + n_0\boldsymbol{x_0}^T\boldsymbol{\theta} - n_0A(\boldsymbol{\theta})) \\
&= exp(\boldsymbol{\theta}(\boldsymbol{x} + n_0\boldsymbol{x_0})^T - A(\boldsymbol{\theta})(1+n_0)) \\
&= exp(\boldsymbol{\theta}(1+n_0)(\frac{\boldsymbol{x}}{1+n_0} + \frac{n_0\boldsymbol{x_0}}{1+n_0})^T - A(\boldsymbol{\theta})(1+n_0))
\end{aligned}$$

We've successfully transformed the posterior into the same functional form as the prior with the new parameters $\frac{\boldsymbol{x}}{1+n_0} + \frac{n_0\boldsymbol{x_0}}{1+n_0}$ and $1+n_0$. This can be extended to a case with more than one example using the same process as above, resulting in the following posterior:

$$\begin{aligned}
p(\boldsymbol{\theta}|\boldsymbol{X_1},...,\boldsymbol{X_n}) &= \pi_{\frac{\boldsymbol{n\bar{X}}}{n+n_0} + \frac{n_0\boldsymbol{x_0}}{1+n_0},(n+n_0)}(\boldsymbol{\theta}) \\
&\propto exp(\boldsymbol{\theta}(n+n_0)(\frac{\boldsymbol{n\bar{X}}}{n+n_0} + \frac{n_0\boldsymbol{x_0}}{n+n_0})^T - A(\boldsymbol{\theta})(n+n_0))
\end{aligned}$$

Obviously the term $\frac{\boldsymbol{n\bar{X}}}{n+n_0} + \frac{n_0\boldsymbol{x_0}}{1+n_0}$ is a weighted average of the prior distribution and observations. However, we want to prove that the mean update function is this weighted average. *It is known that the mean of a distribution from the exponential family is the derivative of the cumulant.* Therefore, we will can find the expectation of the mean update function by taking the expectation of the derivative of the cumulant:

$$E[\nabla A(\boldsymbol{\theta})|X_1,..,X_n] = \int \nabla A(\boldsymbol{\theta})\pi_{\frac{\boldsymbol{n\bar{X}}}{n+n_0} + \frac{n_0\boldsymbol{x_0}}{1+n_0},(n+n_0)}(\boldsymbol{\theta})d\theta = (\frac{\boldsymbol{n\bar{X}}}{n+n_0} + \frac{n_0\boldsymbol{x_0}}{1+n_0}) - \frac{1}{n+n_0}\int \nabla\pi_{\frac{\boldsymbol{n\bar{X}}}{n+n_0} + \frac{n_0\boldsymbol{x_0}}{1+n_0},(n+n_0)}(\boldsymbol{\theta})d\theta$$

It is known that the integral of the derivative of an exponential function is zero. Therefore, this simplifies to:

$$E[\nabla A(\boldsymbol{\theta})|X_1,..,X_n] = \frac{\boldsymbol{n\bar{X}}}{n+n_0} + \frac{n_0\boldsymbol{x_0}}{1+n_0}$$

Thus we have proved that the mean update function is a weighted average of the prior distribution and observations.

3. Hierarchical Models with Sparse Data

The primary reason hierarchical models provide better model fits and regularization on sparse data is **because of the concept of pooling**. Specifically, these models use **partial pooling** to group data in a hierarchy that reflects their hierarchical structure/group relationships. When data is sparse we would like to use information from parent/related groups to inform our analysis of another group we don't have as much data for. Non-hierarchical approaches either view all groups within data separately or pool everyone together. Pooling everyone together will negate all group information and thus over regularize the trends we wish to discern. On the other hand, if data is sparse, viewing all groups separately may cause the model to overfit. Hierarchical models merge both these methods to get a good trade-off between regularization and variance.

A natural situation where hierarchies naturally arise and data is often sparse is in the field of ecology. Often Ecologists work with sparse data of specific sub-species they would like to make inferences on. Using information of related species to inform distributions about species with less data is a approach to dealing with sparse data.