1. **Pre-processing for Bayesian linear regression and Gaussian processes:**

We have a dataset of inputs and outputs \( \{x^{(n)}, y^{(n)}\}_{n=1}^N \), describing \( N \) preparations of cells from some lab experiments. The output of interest, \( y^{(n)} \), is the fraction of cells that are alive in preparation \( n \). The first input feature of each preparation indicates whether the cells were created in lab A, B, or C. That is, \( x_1^{(n)} \in \{A, B, C\} \). The other features are real numbers describing experimental conditions such as temperature and concentrations of chemicals and nutrients.

a) Describe how you might represent the first input feature and the output when learning a regression model to predict the fraction of alive cells in future preparations from these labs. Explain your reasoning.

b) Compare using the lab identity as an input to your regression (as you’ve discussed above), with two baseline approaches: i) Ignore the lab feature, treat the data from all labs as if they came from one lab; ii) Split the dataset into three parts one for lab A, one for B, and one for C. Then train three separate regression models.

Discuss both simple linear regression and Gaussian process regression. Is it possible for these models, when given the lab identity as in a), to learn to emulate either or both of the two baselines?

c) There’s a debate in the lab about how to represent the other input features: log-temperature or temperature, and temperature in Fahrenheit, Celsius or Kelvin? Also whether to use log concentration or concentration as inputs to the regression. Discuss ways in which these issues could be resolved.

Harder: there is a debate between two different representations of the output. Describe how this debate could be resolved.

---

**Answer:**

a) The outputs, being fractions of something, are in the range \([0, 1]\). However, the predictions from a Gaussian process regression model, \( p(y \mid x, D) \), are Gaussian distributions, and so predict numbers in \([-\infty, \infty]\). We could consider instead training on and predicting the logit of the fraction of alive cells. The logit is the inverse of the sigmoid function, \( \text{logit}(y) = \log(y/(1-y)) \). If all of the fractions are tiny (the vast majority of cells are dead) then the logit is nearly the same as taking the log of the output.

What if sometimes all of the cells are alive, and/or sometimes all of the cells are dead? Then the logit will sometimes return \( \infty \) or \( -\infty \), which we won’t be able to fit. We could consider fitting another function of the outputs, such as \( \text{logit}(0.01 + 0.98y) \). I suspect that, like me, you’ll find this suggestion unsatisfying. It’s become pretty clear that the rescaling is arbitrary.

If many of preparations have all the cells dead (or all alive), I would consider fitting a classifier to predict whether the preparation is either all alive, all dead, or in between. Then I could fit a regression model on the logit of the fraction, for the cases where the fraction isn’t zero or one.

I’d be happier if I could build a more principled model of the data: if we knew the number of cells, as well as the fraction dead, we could model the probability of a cell being alive as the sigmoid of a function. However, we’d then need approximate inference methods (such as the Laplace approximation) to compute the posterior over functions or model parameters. Sometimes fitting a Gaussian
model to some scaled data, even if we know it isn’t quite right, is a pragmatic first step.

The lab identity is a categorical variable. A standard way to encode such variables (seen early on in the course) is a 1-of-$M$, or ‘one-hot’ encoding (also called 1-of-$K$ and I’m sure other names). The feature is replaced with $M = 3$ binary features. All but one of the features are set to zero, and one of them is set to one, indicating whether the lab was A, B, or C. As discussed previously in class, we might use only $M = 2$ binary features, one for A and one for B, to avoid the inputs being “colinear”.

Replacing the lab identity with a single feature in \{1, 2, 3\} would not be a good answer in this situation. For linear regression models, this representation gives a model that assumes that, for a given experimental condition, the mean output from lab B is always between the mean outputs from lab A and lab C. We haven’t been given any information that would suggest this assumption is sensible.

For more advanced (non-examinable) ways of dealing with multiple related settings you could search for ‘multi-task learning’.

b) For linear regression: Ignoring the lab feature gives a model with fewer parameters, which is less prone to overfitting if there’s little data. Training three separate models triples the number of parameters; if there’s lots of data it could potentially learn a situation where the three labs lead to completely different linear relationships, although we normally hope that labs will give similar results! Fitting one linear regression model with a one-hot-encoding of the lab adds only three extra parameters. It can emulate the first baseline by setting all of the weights for the lab features to zero. It can’t emulate the second baseline: it can only model a different constant offset to the target function for each lab. Putting the lab into the model as input features could underestimate the data if the influence of the labs on the fraction of alive cells depends on the experimental conditions.

For Gaussian process (GP) regression: ignoring the lab feature could still avoid overfitting when there’s little data. If we learn kernel function lengthscales associated with the lab identity features, we can potentially emulate either baseline. Very long lengthscales mean that the lab feature has negligible effect on kernel values, and the lab identity is ignored. Zero lengthscales would mean that the model thinks the functions for the different labs are independent (zero covariance in the Gaussian distributions we construct), so we could train three separate GPs instead. Intermediate lengthscales say that the labs could behave slightly differently, but that the functions for each lab are related, which we’d hope if the experiment is reasonably reproducible!

c) In some situations, doubling a positive quantity may have a similar-sized effect on the outcome, regardless of the starting value. That is, it’s the log of the quantity that is approximately linearly-related to the output. In these situations taking the log of input features can be necessary to get a good fit with simple linear regression. While GPs can eventually get a close fit to any function, they can require less data to get a good fit if we take the log of an input. Similarly we might need fewer basis functions and/or less data to get a good fit with linear regression with basis functions.

It’s sometimes unclear whether to take logs of positive features however. I would normally take the log of a concentration, but I don’t think I’d usually take the log of a temperature (presumably in Kelvin, so it’s always positive). However, I could be wrong in a particular case, and there will be other features I don’t have good experience with. Therefore, we have multiple possible models, one for each set of feature transformations we want to consider. These could be compared by performance on a validation set, or by marginal likelihood. Alternatively we
could create a model with all versions of the features, and hope (perhaps with regularization) we’ll learn some small weights or large lengthscales so that we’ll ignore some of them.

(Warning: models with redundant features can be non-identifiable, and hard to interpret. Without careful regularization they’re more prone to numerical problems and overfitting.)

If we don’t take logs of the temperatures, the units (F, C or K) just set an offset and scaling of the numbers. That will have no effect if we standardize our inputs to have zero mean and unit variance before fitting. Even if we don’t standardize our features, the regression models can learn the same mapping from features to outputs for any offset and scale of the inputs. However, the regularization constants will have different effects for the different scalings of the inputs, which is why we normally standardize our inputs.

Why is comparing transformations of the outputs harder? If we squash the outputs into a narrower range, then the square error or negative log probability we get will go down. That doesn’t mean we are doing better! If different models use different transformations of the output, they need to be converted back to a common scale for comparison. If evaluating probabilistic models of real-valued outputs, densities need to be transformed correctly (using the Jacobian of the transformation). A model is likely to do well on the scale that it was trained to do well on: domain-specific thought is required to decide on which scale we should evaluate a model.

2. Gaussian processes with non-zero mean:

In the lectures we assumed that the prior over any vector of function values was zero mean: \( f \sim \mathcal{N}(0, K) \). We focussed on the covariance or kernel function \( k(x^{(i)}, x^{(j)}) \), which evaluates the \( K_{ij} \) elements of the covariance matrix (also called the ‘Gram matrix’).

If we know in advance that the distribution of outputs should be centered around some other mean \( \mathbf{m} \), we could put that into the model. Instead, we usually subtract the known mean \( \mathbf{m} \) from the \( y \) data, and just use the zero mean model.

Sometimes we don’t really know the mean \( \mathbf{m} \), but look at the data to estimate it. A fully Bayesian treatment puts a prior on \( \mathbf{m} \) and, because it’s an unknown, considers all possible values when making predictions. A flexible prior on the mean vector could be another Gaussian process(!). Our model for our noisy observations is now:

\[
\begin{align*}
\mathbf{m} &\sim \mathcal{N}(0, K_m), & K_m \text{ from kernel function } k_m, \\
\mathbf{f} &\sim \mathcal{N}(\mathbf{m}, K_f), & K_f \text{ from kernel function } k_f, \\
\mathbf{y} &\sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I), & \text{noisy observations.}
\end{align*}
\]

Show that — despite our efforts — the function values \( \mathbf{f} \) still come from a function drawn from a zero-mean Gaussian process (if we marginalize out \( \mathbf{m} \)). Identify the covariance function of the zero-mean process for \( \mathbf{f} \).

Identify the mean’s kernel function \( k_m \) for two restricted types of mean: 1) An unknown constant \( m_i = b \), with \( b \sim \mathcal{N}(0, \sigma^2_b) \). 2) An unknown linear trend: \( m_i = \mathbf{w}^\top x^{(i)} + b \), with Gaussian priors \( \mathbf{w} \sim \mathcal{N}(0, \sigma^2_w I) \), and \( b \sim \mathcal{N}(0, \sigma^2_b) \).

Sketch three typical draws from a GP prior with kernel:

\[
k(x^{(i)}, x^{(j)}) = 0.12 \exp \left(-\frac{(x^{(i)} - x^{(j)})^2}{2}\right) + 1.
\]

Hints in footnote1.

1. The covariances of two Gaussians add, so think about the two Gaussian processes that are being added to give
The function $f$ is the sum of two Gaussian processes. Any vector of values $f$ is the sum of two independent Gaussian vectors: $m$ and a zero-mean draw from $\mathcal{N}(0, K_f)$. Thus any set of values are marginally drawn from $f \sim \mathcal{N}(0, K_m+K_f)$, corresponding to a function drawn from a zero mean Gaussian process with covariance function $k(x^{(i)}, x^{(j)}) = k_m(x^{(i)}, x^{(j)}) + k_f(x^{(i)}, x^{(j)})$.

What are the statistics of a vector $m$, with identical elements $m_i = b$, with $b \sim \mathcal{N}(0, \sigma^2_b)$? The vector is Gaussian, and each element has mean zero and marginal variance $\sigma^2_b$.

The other elements of the covariance matrix are given by:

$$
cov[m_i, m_j] = E[m_i m_j] - E[m_i] E[m_j] = E[b^2] = \sigma^2_b.
$$

So $k_m(x^{(i)}, x^{(j)}) = \sigma^2_b$, and the resulting matrix $K_m$ has every element set to $\sigma^2_b$. What we’ve learned: to add a random offset to a process, we can just add a constant to every element of the covariance matrix. (Not just the diagonal: what would that do?)

The covariance function for a linear model was derived in the lecture notes, $k(x^{(i)}, x^{(j)}) = \sigma^2_w x^{(i)\top} x^{(j)} + \sigma^2_b$. If in addition to a random constant offset we want a random linear trend, we further add $\sigma^2_w x^{(i)\top} x^{(j)}$ to each element $K_{ij}$.

A little knowledge is a dangerous thing: Keen students, who actually want to implement Gaussian processes with a linear trend or constant offset, are advised to read Section 2.7, pp27–28, of Rasmussen and Williams, *Gaussian Processes for Machine Learning*. There can be numerical problems with an obvious implementation using the covariance matrices above.

A final comment: a linear trend is usually physically unrealistic. The model confidently predicts an extreme output if you go to an extreme input location. Under the model, the best way to measure the linear trend is to evaluate the function at extreme input locations. These properties could cause problems if using the model for planning or optimization. It may be better to model a large-scale trend by adding a stationary kernel (such as the squared exponential) with a long lengthscale, rather than the linear kernel.


The functions you were asked to sketch are from a Gaussian kernel with lengthscale 1 and marginal variance 0.1, plus a random offset, which has variance 1.
You could plot actual samples on a computer by adjusting the kernel function in the demonstration code provided with the notes. Only a small change to one line is required. If you didn’t get the answer above, you should have checked it, and you should explore the demonstration to help build some understanding.

3. **Laplace approximation:**

The Laplace approximation fits a Gaussian distribution to a distribution by matching the mode of the log density function and the second derivatives at that mode. See the w8b lecture notes for more pointers.

Exercise 27.1 in MacKay’s textbook (p342) is about inferring the parameter \( \lambda \) of a Poisson distribution based on an observed count \( r \). The likelihood function and prior distribution for the parameter are:

\[
P(r \mid \lambda) = \exp(-\lambda) \frac{\lambda^r}{r!}, \quad p(\lambda) \propto \frac{1}{\lambda}.
\]

Find the Laplace approximation to the posterior over \( \lambda \) given an observed count \( r \).

Now reparameterize the model in terms of \( \ell = \log \lambda \). After performing the change of variables\(^2\), the improper prior on log \( \lambda \) becomes uniform, that is \( p(\ell) \) is constant. Find the Laplace approximation to the posterior over \( \ell = \log \lambda \).

Which version of the Laplace approximation is better? It may help to plot the true and approximate posteriors of \( \lambda \) and \( \ell \) for different values of the integer count \( r \).

---

**Answer:**

The posterior is proportional to prior times likelihood:

\[
p(\lambda \mid r) \propto e^{-\lambda} \frac{\lambda^{r-1}}{r!}.
\]

Define an ‘energy’, equal to the negative log posterior up to a constant:

\[
E = \lambda - (r-1) \log \lambda.
\]

First find the MAP estimate, by minimizing the energy:

\[
\frac{\partial E}{\partial \lambda} = 1 - \frac{r-1}{\lambda}, \quad \Rightarrow \lambda_{\text{MAP}} = r - 1.
\]

\(^2\) Some review of how probability densities work: Conservation of probability mass means that: \( p(\ell) \, d\ell = p(\lambda) \, d\lambda \) for small corresponding elements \( d\ell \) and \( d\lambda \). Dividing and taking limits: \( p(\ell) = p(\lambda) \, d\lambda / d\ell \), evaluated at \( \lambda = \exp(\ell) \). The size of the derivative \( |d\lambda / d\ell| \) is referred to as a Jacobian term.
Then measure the curvature at the minimum:

\[ \frac{\partial^2 E}{\partial \lambda^2} \bigg|_{\lambda = \lambda_{\text{MAP}}} = \frac{r - 1}{\lambda_{\text{MAP}}^2} = \frac{1}{r - 1}. \]

Unless \( r > 1 \) we don’t have a mode at \( \lambda > 0 \) to expand around sensibly.

\[ \Rightarrow \quad p(\lambda \mid r) \approx N(\lambda; r-1, r-1), \quad r > 1. \]

The posterior over \( \ell = \log \lambda \) is proportional to prior times likelihood:

\[ p(\log \lambda \mid r) \propto e^{-\lambda \frac{\lambda'}{r!}}. \]

Define an ‘energy’, equal to the negative log posterior up to a constant:

\[ E = \lambda - r \log \lambda = e^{\ell} - r \ell. \]

First find the MAP estimate of \( \ell \), by minimizing the energy:

\[ \frac{\partial E}{\partial \ell} = e^{\ell} - r, \quad \Rightarrow \quad \ell_{\text{MAP}} = \log r. \]

We now only need \( r > 0 \) for the approximation to make sense. Then measure the curvature at the minimum:

\[ \frac{\partial^2 E}{\partial \ell^2} \bigg|_{\ell = \ell_{\text{MAP}}} = e^{\ell} \bigg|_{\ell = \ell_{\text{MAP}}} = r. \]

Writing \( \ell \) as \( \log \lambda \) again, we get:

\[ \Rightarrow \quad p(\log \lambda \mid r) \approx N(\log \lambda; \log r, \frac{1}{r}), \quad r > 0. \]

Which approximation is better? Putting the rate on a log scale means that the distribution is less skewed and unbounded. Both of these properties are a better match for a Gaussian approximation. To plot the distribution of \( \log \lambda \), I put \( \lambda \) on the \( x \)-axis, but with a log scale. The numbers are then directly comparable between the plots.

\( r = 2 \): for low counts, the Laplace approximation on \( \lambda \) (dashed, red) puts lots of mass on negative (impossible) values. The approximation based on \( \log \lambda \) is better, although neither can fit the true distribution (solid, blue) perfectly:

\( r = 20 \): for higher counts, the Laplace approximation on \( \lambda \) (dashed, red) works better.
than before. However the approximation based on log $\lambda$ is still slightly better as the true distribution (solid, blue) is less skewed in this parameterization:

\begin{align*}
\text{The code that produced these plots is available online.}
\end{align*}