Bayesian Optimal Design using Variational Inference

POCS, 7.5 ECTS, Bachelor's level.

Rasmus Hag Løvstad, pgq596@alumni.ku.dk
Advisor: Oswin Krause

Submitted June 23, 2023

Contents

1 Introduction

Scientific experiments and simulations often involve resource-consuming processes such as conducting experiments or evaluating high-cost functions. Efficiently optimizing the design of each experiment can significantly enhance the information gained, resulting in improved models while conserving resources. The *Bayesian Optimal Design Problem* offers a formal framework from a Bayesian perspective to optimize important experiment objectives, such as expected information gain from a prior model, within a given design space[1].

In this project, we will look at the Bayesian Optimal Design Problem in the context of a simple linear regression model, where the design space is the set of data points from which the model is trained.
Additionally, we explore the application of Bayesian Variational Inference for fitting the linear regression model, as this approach does not rely on as many assumptions about the nature of the model, and thus can generalize better to other models than linear models. At the end, we will explore how to integrate the variational framework into the optimal design problem by regarding it as a nested optimization problem and by using the Implicit Function Theorem to compute the gradient.
This project is written in a weekly-report style, where each section explores separate topics that will be combined to solve the general problem statement in the end.

2 Week 1

2.1 Objective

For week 1, the main objective is to explore the linear regression problem from a Bayesian perspective. First the essential model assumptions will be stated, from which the consequences of these can be examined using Bayes' rule. This will be used to see how we can incorporate previous knowledge into the model and see how observing new data will change our model. While this is apparent for any regression model, it will also be demonstrated how one can derive an exact posterior parameter distribution in the case of linear regression.
In the end, a simple linear regression model will be implemented in Python, the prior model assumptions effect on our resulting model will be explored.

2.2 Theory

2.2.1 The regression problem and Bayes' rule

The general regression problem can be stated like so **krause22**: Given a data set $\mathcal{D} = (\mathbf{d}, \mathbf{y})$, where $\mathbf{d} \in \mathbb{R}^{\ell \times d}$ and $\mathbf{y} \in \mathbb{R}^{\ell}$, find some function $\phi_\theta \in \mathbb{R}^{\ell \times d} \to \mathbb{R}^{\ell}$ such that

$$\phi_\theta(\mathbf{d}) \approx \mathbf{y} \tag{1}$$

where ϕ_θ takes some vector of parameters θ.
In the Bayesian approach, the probability distribution of $\mathbf{y}$ given data $\mathbf{d}$ can be described by $p(\mathbf{y}|\mathbf{d})$, called the *evidence*. The probability of $\mathbf{y}$ being generated from some ϕ_θ while having observed $\mathbf{d}$ can be described by $p(\mathbf{y}|\theta, \mathbf{d})$ called the *likelihood*.
One might have some prior information on how the parameters θ are distributed - perhaps from prior experiments, domain knowledge or qualified guessing. This can be encoded into the *prior* distribution $p(\theta|\mathbf{d})$, which is often denoted as $p(\theta)$ since θ is often picked to be independent from $\mathbf{d}$.
Given a likelihood, a marginal, and a prior, Bayes' rule can be used to find the *posterior* distribution of θ given the data $\mathbf{d}$ and target values $\mathbf{y}$:

$$p(\theta|\mathbf{d}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{d}, \theta)p(\theta)}{p(\mathbf{y}|\mathbf{d})} \tag{2}$$

What the posterior distribution $p(\theta|\mathbf{d}, \mathbf{y})$ models is the *change of belief* from the prior model $p(\theta)$ given the evidence $p(\mathbf{y}|\mathbf{d})$**krause22**. It represents a new best bet for the parameters θ, which can be used to produce a better ϕ_θ for (**??**).

2.2.2 The linear regression problem

In the case of linear regression, we assume that ϕ_θ is a linear function, i.e. $\phi(\mathbf{d}) = \mathbf{d}\theta$ where $\theta \in \mathbb{R}^d$. It is also common to augment the data set with an additional dimension set to 1, such that ϕ_θ also models the intercept such that $\mathbf{d} \in \mathbb{R}^{\ell \times d+1}$ and $\theta \in \mathbb{R}^{d+1}$. In this case θ_{d+1} will then be the

intercept.

We are going to assume that the each target value $\mathbf{y}^{(i)}$ can accurately be described as

$$\mathbf{y}^{(i)} = \mathbf{d}^{(i)}\theta + \epsilon \tag{3}$$

for some $\mathbf{d}$, θ, ϵ, where $\epsilon \sim \mathcal{N}(0, \sigma_{\mathbf{y}}^2)$ is some normally distributed noise term. Thus, the likelihood distribution can be described like a normal distribution.

$$p(\mathbf{y}^{(i)}|\mathbf{d}^{(i)}, \theta) = \mathcal{N}(\mathbf{y}^{(i)}; \mathbf{d}^{(i)}\theta, \sigma_{\mathbf{y}}^2) \tag{4}$$

Assuming that the target values are independent, it must follow that

$$p(\mathbf{y}|\mathbf{d}, \theta) = \prod_{i=0}^{\ell} p(\mathbf{y}^{(i)}|\mathbf{d}^{(i)}, \theta) = \mathcal{N}(\mathbf{y}; \mathbf{d}\theta, \sigma_{\mathbf{y}}^2 I_\ell) \tag{5}$$

When modeling the prior information, one could choose to model the parameter distribution as a multivariate Gaussian distribution as well

$$p(\theta) = \mathcal{N}(\theta; \mu_\theta, \Sigma_\theta) \tag{6}$$

where μ_θ, Σ_θ are chosen by the model designer.
The last term of (**??**) is the marginal distribution $p(\mathbf{y}|\mathbf{d})$, which is the probability of observing the target values $\mathbf{y}$ without any additional information. This term is difficult to make any reasonable assumptions about, and can thus hard to compute in practice. Luckily, it mostly acts as a normalization term that makes sure that the posterior distribution integrates to 1, so it can for many use cases be safely ignored. As we will see, for a linear regression problems with these assumptions, we will indeed not need it.

2.2.3 An analytical solution for the posterior distribution

Let us outline a possible derivation for the posterior distribution: Given the likelihood and prior, we can describe the joint distribution $p(\theta, \mathbf{y}|\mathbf{d})$ as

$$p(\theta, \mathbf{y}|\mathbf{d}) = p(\mathbf{y}|\mathbf{d}, \theta)p(\theta|\mathbf{d}) = p(\mathbf{y}|\mathbf{d}, \theta)p(\theta) \tag{7}$$

where the last equality is due to the model parameters being independent from the controlled data points.
From this point, we will need to condition the joint distribution on $\mathbf{y}$:

$$p(\theta|\mathbf{y}, \mathbf{d}) = \frac{p(\theta, \mathbf{y}|\mathbf{d})}{p(\mathbf{y})} \tag{8}$$

To perform these steps, we are going to use some useful lemmas about the multivariate normal distribution.

Lemma 1. *Let $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$ be a multivariate normal random variable. We can regard these variables in block notation:*

$$\mathbf{X} = \left[\begin{array}{c} \mathbf{X}_1 \\ \hline \mathbf{X}_2 \end{array}\right], \quad \mu = \left[\begin{array}{c} \mu_1 \\ \hline \mu_2 \end{array}\right], \quad \Sigma = \left[\begin{array}{c|c} \Sigma_{11} & \Sigma_{12} \\ \hline \Sigma_{21} & \Sigma_{22} \end{array}\right] \tag{9}$$

then we must have

$$\mu_{2|1} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(X_1 - \mu_1), \quad \Sigma_{2|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{21}^T \tag{10}$$

Proof. See **krause22** □

Lemma 2. *For random variables* $\mathbf{X} \sim \mathcal{N}(\mu_{\mathbf{X}}, \Sigma_{\mathbf{X}})$, $\mathbf{Y}|\mathbf{X} \sim \mathcal{N}(\mu_{\mathbf{Y}} + A\mathbf{X}, \Sigma_{\mathbf{Y}})$, *for some* A, *the joint distribution* $p(\mathbf{X}, \mathbf{Y})$ *is given by*

$$\left[\begin{array}{c}\mathbf{X}\\\hline\mathbf{Y}\end{array}\right] = \mathcal{N}\left(\left[\begin{array}{c}\mu_{\mathbf{X}}\\\hline\mu_{\mathbf{Y}} + A\mu_{\mathbf{X}}\end{array}\right], \left[\begin{array}{c|c}\Sigma_{\mathbf{X}} & \Sigma_{\mathbf{X}}A^T\\\hline A\Sigma_{\mathbf{X}} & A\Sigma_{\mathbf{X}}A^T + \Sigma_{\mathbf{Y}}\end{array}\right]\right) \tag{11}$$

Proof. Follows from Lemma **??** and the definition of the multivariate normal distribution. □

From the likelihood from (**??**), it can be seen that it is only dependent of θ by a mean-shift with $\mu_{\mathbf{y}|\theta,\mathbf{d}} = 0 + \mathbf{d}\theta$. Thus we can use Lemma **??** to get the joint distribution $p(\theta, \mathbf{y}|\mathbf{d})$:

$$\left[\begin{array}{c}\theta\\\hline\mathbf{y}\end{array}\right] = \mathcal{N}\left(\left[\begin{array}{c}\mu_{\theta}\\\hline\mathbf{d}\mu_{\theta}\end{array}\right], \left[\begin{array}{c|c}\Sigma_{\theta} & \Sigma_{\theta}\mathbf{d}^T\\\hline \mathbf{d}\Sigma_{\theta} & \mathbf{d}\Sigma_{\theta}\mathbf{d}^T + \sigma_{\mathbf{y}}^2 I_\ell\end{array}\right]\right) \tag{12}$$

To get the posterior distribution, we will need to use the conditioning Lemma **??** on a reordered term:

$$\left[\begin{array}{c}\mathbf{y}\\\hline\theta\end{array}\right] = \mathcal{N}\left(\left[\begin{array}{c}\mathbf{d}\mu_{\theta}\\\hline\mu_{\theta}\end{array}\right], \left[\begin{array}{c|c}\mathbf{d}\Sigma_{\theta}\mathbf{d}^T + \sigma_{\mathbf{y}}^2 I_\ell & \mathbf{d}\Sigma_{\theta}\\\hline \Sigma_{\theta}\mathbf{d}^T & \Sigma_{\theta}\end{array}\right]\right) \tag{13}$$

With the conditioning lemma, we get the posterior distribution $p(\theta|\mathbf{y}, \mathbf{d}) = \mathcal{N}(\theta; \mu_{\theta|\mathbf{y},\mathbf{d}}, \Sigma_{\theta|\mathbf{y},\mathbf{d}})$:

$$\mu_{\theta|\mathbf{y},\mathbf{d}} = \mu_{\theta} + \Sigma_{\theta}\mathbf{d}^T(\mathbf{d}\Sigma_{\theta}\mathbf{d}^T + \sigma_{\mathbf{y}}^2 I_\ell)^{-1}(\mathbf{y} - \mathbf{d}\mu_{\theta}) \tag{14}$$

$$\Sigma_{\theta|\mathbf{y},\mathbf{d}} = \Sigma_{\theta} - \Sigma_{\theta}\mathbf{d}^T(\mathbf{d}\Sigma_{\theta}\mathbf{d}^T + \sigma_{\mathbf{y}}^2 I_\ell)^{-1}\mathbf{d}\Sigma_{\theta} \tag{15}$$

Thus we have an analytical expression for the posterior distribution.

2.2.4 Using the posterior to optain model parameters

From this distribution, we can obtain model parameters in several ways. A common choice is to pick the θ that maximizes the posterior. This is called the *Maximum-a-posteriori* (MAP) estimate, and will for a normal distribution just be the posterior mean $\mu_{\theta|\mathbf{y},\mathbf{d}}$. For other cases than linear regression, one can also use a mean over samples of θ from the posterior distribution, but for our case, this will converge against the posterior mean $\mu_{\theta|\mathbf{y},\mathbf{d}}$. Another solution is to define a distribution called the *posterior predictive* distribution, that for some new data points $\mathbf{d}_{\text{new}}$ computes the corresponding $\mathbf{y}_{\text{new}}$:

$$p(\mathbf{y}_{\text{new}}|\mathbf{d}_{\text{new}}, \mathbf{d}, \mathbf{y}) = \int p(\theta|\mathbf{y}, \mathbf{d})p(\mathbf{y}_{\text{new}}|\theta, \mathbf{d}_{\text{new}})d\theta \tag{16}$$

Integrals like these are usually solved by sampling - examples of this will be seen later. For now, we can use Lemma **??** to get

$$p(\mathbf{y}_{\text{new}}|\mathbf{d}_{\text{new}}, \mathbf{d}, \mathbf{y}) = \mathcal{N}(\mathbf{y}_{\text{new}}; \mathbf{d}_{\text{new}}^T\mu_{\theta|\mathbf{y},\mathbf{d}}, \sigma_{\mathbf{y}}^2 + \mathbf{d}_{\text{new}}^T\Sigma_{\theta|\mathbf{y},\mathbf{d}}\mathbf{d}_{\text{new}}) \tag{17}$$

Having the posterior predictive allows us to both sample a $\mathbf{y}$ for a given $\mathbf{d}$, choose the $\mathbf{y}$ that maximizes the distribution, and to give us some measure of the inherent uncertaincy in the model **krause22**.

2.3 Implementation

Implementing a Bayesian linear regression model is fairly simple. To begin with, one needs to decide on a prior distribution for θ, as well as what the variance of the noise $\sigma_{\mathbf{y}}^2$ should be used to generate the example data. In the real world, $\sigma_{\mathbf{y}}^2$ would be measured from observed data, and the prior would be chosen based on the expected distribution of θ based on as much prior knowledge as possible , but for our toy representation, we are going to play around with many different possible priors and noise parameters. For a toy implementation, one can start by generating some data points $\mathbf{d}$ and noise samples ϵ as well as deciding on some true, underlying weight parameters θ_{true}. Then we can compute $\mathbf{y}$ as in (**??**). From this, we can simply implement the posterior distribution as in equation **??** and **??**. In Python, this can be done as follows:

```
1 def analytical_posterior_params(d, y, mu_prior=mu_prior,
  ↪  cov_prior=cov_prior): # Get parameters of pdf
2     mu = mu_prior + cov_prior @ d.T @ np.linalg.inv(d @
      ↪  cov_prior @ d.T + noise * np.eye(l)) @ (y - d @
      ↪  mu_prior)
3     cov = cov_prior - cov_prior @ d.T @ np.linalg.inv(d @
      ↪  cov_prior @ d.T + noise * np.eye(l)) @ d @ cov_prior
4     return mu, cov
5 stats.multivariate_normal.pdf(theta, mean=mu, cov=cov)
```

2.4 Results

An example of how the posterior changes from the prior can be seen in figure **??**, where 30 random prior means and covariances are plotted alongside their respective posterior. It can be seen that the quality of the posterior is very dependent on the amount of data, as well as the inherent noise in the data. It can also be seen that in a scenario with low amount of data and high noise, having a prior that is close to the true weights makes it more probable that ones posterior will be as well.

An example of a posterior predictive distribution is seen in figure **??**, where one can see how the distribution is most confident in the area that one would expect the most points to be, with only a few outliers. Thus we've seen that the Bayesian linear regression model is able to learn the true weight parameters from data, and that it is able to do so even when the prior is very different from the true weight parameters. Now, we will move on to regarding the Bayesian Optimal Design problem.

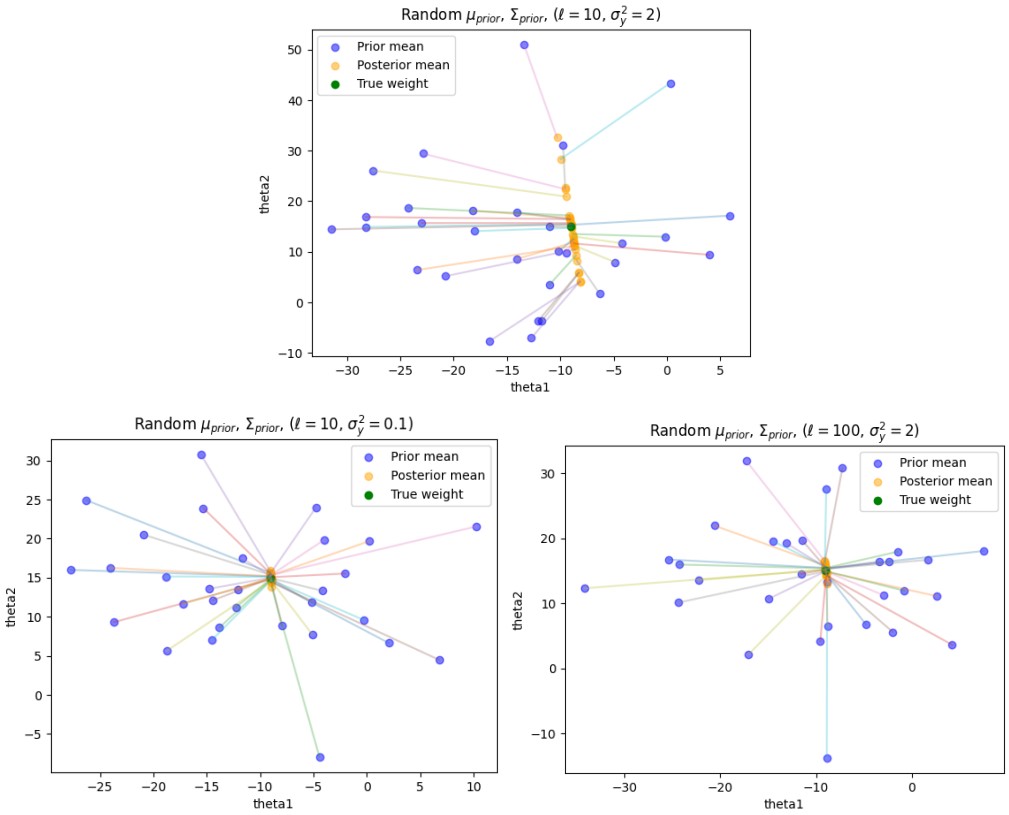

Figure 1: 30 different models trained on data sets with different sizes and noise levels. The more samples or lower noise, the closer to the true weights are the posteriors.

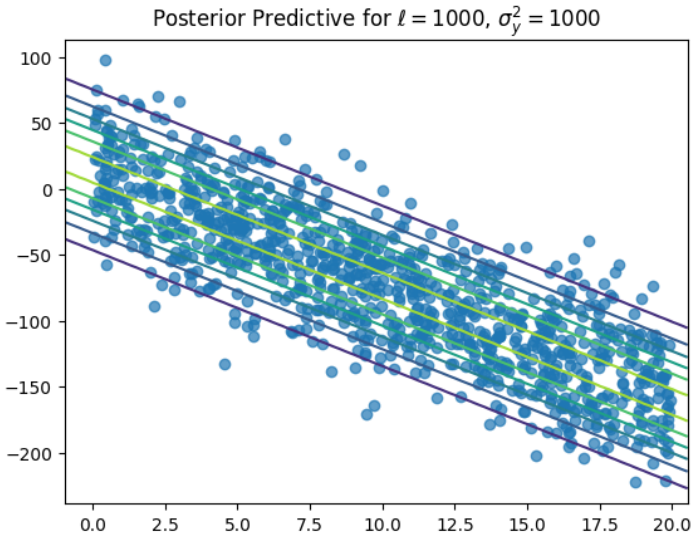

Figure 2: Contour plot of the posterior predictive distribution.

3 Week 2

3.1 Objective

For week 2, we are going to regard the *Bayesian Optimal Design* problem for linear regression, and use this to implement a reference implementation, to be used later. We can then also explore the effect of different data sizes, different priors and the difference between estimating an objective function through sampling versus calculating it analytically.

3.2 Theory

3.2.1 Bayesian Optimal Design

Often in scientific contexts as well as other cases, one might have a model that one wishes to strengthen in one way or another using experimental data. Performing the experiments needed to strengthen one's model can be expensive however, so having an efficient strategy to do such can save important resources. This is where *Bayesian Optimal Design* comes in.

The Bayesian Optimal Design problem is about finding a design $\mathbf{d}$ from a design space $\mathbf{D}$, that optimizes some kind of utility function.[1] For this project, we wish to maximize the expected information gain from the prior to the posterior. To find the optimal design, we want to find a maximizer $\mathbf{d}^*$ defined as such:

$$\mathbf{d}^* = \arg\max_{\mathbf{d}\in\mathbf{D}} I(\mathbf{d}) \tag{18}$$

Where $I(\mathbf{d})$ is the *Mutual Information* between the prior and posterior when adjusted for data observed at $\mathbf{d}$.

3.2.2 The Nature of the Mutual Information metric

The amount of information of an experiment is often defined as the negative differential entropy defined as such**lindley56**:

$$H_x = \int_X p(x)\log p(x)dx \tag{19}$$

Thus, the information known before $\mathbf{y}$ is observed from $\mathbf{d}$ is

$$H_\theta = \int_\Theta p(\theta)\log p(\theta)d\theta \tag{20}$$

and after is

$$H_{\theta|\mathbf{y},\mathbf{d}} = \int_\Theta p(\theta|\mathbf{y},\mathbf{d})\log p(\theta|\mathbf{y},\mathbf{d})d\theta \tag{21}$$

The gain of information must thus be

$$H_{\text{gain}} = H_{\theta|\mathbf{y},\mathbf{d}} - H_\theta \tag{22}$$

If we instead regard (**??**) and (**??**) as expectations we get

$$= \mathbb{E}_{\theta|\mathbf{y},\mathbf{d}}[\log p(\theta|\mathbf{y},\mathbf{d})] - \mathbb{E}_\theta[\log p(\theta)] \tag{23}$$

Before we perform the experiment, we do not know what the outcome will be. Instead, we'll just regard the expected outcome by taking the expectation over the observed evidence $\mathbf{y}|\mathbf{d}$:

$$\mathbb{E}_{\mathbf{y}|\mathbf{d}}[H_{\text{gain}}] = \mathbb{E}_{\mathbf{y}|\mathbf{d}}[\mathbb{E}_{\theta|\mathbf{y},\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d}))] - \mathbb{E}_{\theta}[\log(p(\theta))]] \tag{24}$$

This expression is called the *Mutual Information* between the prior and posterior, and will be denoted $I(\mathbf{d})$.
We can also rewrite the expectations as such using Bayes' rule and that θ is independent from $\mathbf{y}|\mathbf{d}$:

$$I(\mathbf{d}) = \mathbb{E}_{\mathbf{y}|\mathbf{d}}[\mathbb{E}_{\theta|\mathbf{y},\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d}))]] - \mathbb{E}_{\theta}[\log(p(\theta))] \tag{25}$$

$$= \mathbb{E}_{\theta}[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d}))]] - \mathbb{E}_{\theta}[\log(p(\theta))] \tag{26}$$

$$= \mathbb{E}_{\theta}[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d}))] - \log(p(\theta))] \tag{27}$$

$$= \mathbb{E}_{\theta}[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d})) - \log(p(\theta))]] \tag{28}$$

We can now use Bayes' rule to get

$$I(\mathbf{d}) = \mathbb{E}_{\theta}[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(p(\mathbf{y}|\theta,\mathbf{d})) - \log(p(\mathbf{y}|\mathbf{d}))]] \tag{29}$$

Then we can use that $\frac{p(\mathbf{y},\theta|\mathbf{d})}{p(\theta)} = p(\mathbf{y}|\theta,\mathbf{d})$:

$$I(\mathbf{d}) = \mathbb{E}_{\theta}[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log\left(\frac{p(\mathbf{y},\theta|\mathbf{d})}{p(\theta)}\right) - \log(p(\mathbf{y}|\mathbf{d}))]] \tag{30}$$

$$= \mathbb{E}_{\theta}[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log\left(\frac{p(\mathbf{y},\theta|\mathbf{d})}{p(\theta)p(\mathbf{y}|\mathbf{d})}\right)]] \tag{31}$$

$$= \text{KL}(p(\mathbf{y},\theta|\mathbf{d})||p(\theta)p(\mathbf{y}|\mathbf{d}))$$

Two random variables X and Y are said to be *independent* if the product of their distributions is the same as their joint distribution i.e.

$$p(X,Y) = p(X)p(Y) \tag{32}$$

Thus, the Mutual Information metric measures how close the prior and the evidence are to be independent. If $\mathbf{d}$ is picked such that the prior has a high probability of being able to predict $\mathbf{y}|\mathbf{d}$, then the mutual information is going to be close to 0. If, on the contrary, $\mathbf{d}$ is picked such that the prior has a low probability of being able to predict $\mathbf{y}|\mathbf{d}$, the mutual information is going to be large. Thus one could expect that an optimizer would prefer to pick a $\mathbf{d}$ within an area where the prior is not very representative of the underlying generating function. Of course, in an experimental design context we do not have access to this underlying function as we might not be able to simulate experiments accurately. Instead, the expectation expressions in (**??**) makes it such that we only regard the expected information gain for any given underlying function. For the rest of this project, we will regard the definition of mutual information as in (**??**).

3.2.3 Evaluating the Mutual Information through sampling

Without using any assumptions about the nature of our model or data, we can utilize Monte Carlo sampling to obtain an accurate estimate on the expectations in (**??**). First we'll use that the prior is independent of $\mathbf{y}|\mathbf{d}$ to get

$$I(\mathbf{d}) = \mathbb{E}_\theta[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d}))]] - \mathbb{E}_\theta[\log(p(\theta))] \tag{33}$$

$$= \mathbb{E}_\theta[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d}))]] - H_\theta \tag{34}$$

Then we can use that entropy of multivariate Gaussian distributions has a closed form solution:

$$= \mathbb{E}_\theta[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d}))]] - \frac{1}{2}\ln\det(2\pi e\Sigma_\theta) \tag{35}$$

Then we can sample the nested expectations. For N samples of $\theta \sim p(\theta)$ and M samples of $\mathbf{y}_i \sim p(\mathbf{y}_i|\theta_i,\mathbf{d})$ this looks like

$$I(\mathbf{d}) \approx \frac{1}{NM}\sum_{i=0}^{N}\sum_{j=0}^{M}(\log(p(\theta_i|\mathbf{y}_{ij},\mathbf{d}))) - \frac{1}{2}\ln\det(2\pi e\Sigma_\theta) \tag{36}$$

3.2.4 Stochastic Optimization

For the sampling approach here, and in the rest of the project, we will use a stochastic gradient descent algorithm to perform the optimization necessary to solve (**??**). From some starting point $\mathbf{d}_0$, iteratively update $\mathbf{d}_i$ by

$$\mathbf{d}_i = \mathbf{d}_{i-1} + \alpha\frac{1}{10^c + i\times 10^{-\beta}}\nabla_{\mathbf{d}} I(\mathbf{d}_{i-1}) \tag{37}$$

where $\alpha, \beta, c \in \mathbb{R}$ are hyper-parameters that ensures a slow convergence to tailor the variance that occurs when using Monte Carlo methods.

3.3 Implementation

The mutual information metric can be implemented using the sampling method like so:

```
def mutual_information(d):
    l = len(d)
    M = 2
    N = 5
    samples = []
    thetas = np.random.multivariate_normal(mu_prior,
    ↪  cov_prior, size=N)
    for theta in thetas:
        ys = ((np.tile(augment_d(d), (M, 1, 1)) @ theta).T +
        ↪  (noise * np.eye(len(d))) @
        ↪  np.random.randn(len(d), M)).T
```

```
        for y in ys:
            mu_post, cov_post =
             ↪  analytical_posterior_params(augment_d(d), y)
            val = stable_multivariate_gaussian_logpdf(theta,
             ↪  mu_post, cov_post) - 0.5 *
             ↪  np.log(np.linalg.det(2*np.pi*np.e*cov_prior))
            samples.append(val)
    return 1/(N * M) * np.sum(samples)
```

Finding the gradient can be done using `autograd.grad`. An example of this can be seen in this implementation of the gradient descent algorithm:

```
def optimize(f, d0, alpha, beta, c, iterations):
  d = d0
  g = grad(f)
  for i in range(iterations):
    d = d + alpha / (10**c + i * 10**(-beta)) * g(d)
  return d
```

3.3.1 Optimizing over 2-d matrices

A small, but important note in these implementations is that $\mathbf{d}$ is two-dimensional. To make computation of gradients and optimization much easier, whenever we are in the context of the optimization algorithm, we will regard $\mathbf{d}$ and the gradient as vectors of the shape

$$\mathbf{d}_v = \begin{bmatrix} \mathbf{d}_{1,1} \\ \vdots \\ \mathbf{d}_{1,d} \\ \mathbf{d}_{2,1} \\ \vdots \\ \mathbf{d}_{\ell,d} \end{bmatrix}$$

Whenever we are in the context of Mutual information, we will regard $\mathbf{d}$ as a matrix. The following helper functions can thus be used to flatten and reconstruct the matrices as needed:

```
def encode_d(d): # matrix to vector
    return d.flatten()
def decode_d(encoded_d, dim=2): # vector to matrix
    return encoded_d.reshape(int(len(encoded_d)/dim), dim)
```

3.3.2 Implementing the log-pdf of multivariate normal distribution

Optimizing over this MI objective function means calculating the gradient of it. We will use autograd.grad to do this, which necessitates implementing the log-pdf of the multivariate normal distribution by hand, since the one supplied by autograd.scipy.stats does not handle the case where the mean and covariance carries information about the computation graph, which also will be necessary later.
The log-pdf of the multivariate normal distribution is given by:

$$\log \mathcal{N}(x; \mu, \Sigma) = -\frac{1}{2}(n \log(2\pi) + \log(\det(\Sigma)) + (x-\mu)^T \Sigma^{-1}(x-\mu)) \quad (38)$$

It is important that this implementation is numerically stable and efficient, since it will be used a lot. The following implementation is used:

```
def stable_multivariate_gaussian_logpdf(x, mu, cov):
    n = cov.shape[-1]
    x_mu = x - mu
    _, log_det = np.linalg.slogdet(cov)
    cov_inv = np.linalg.inv(cov)
    prod = x_mu.T @ cov_inv @ x_mu
    log_prob = -0.5 * (n * np.log(2 * np.pi) + log_det +
    ↪  prod)
    return log_prob
```

3.4 Results

If we regard the problem with $\ell = 10$ data points and $d = 2$ dimensions with a zero-mean prior with identity prior covariance, we will see data closer to the optimum be spread out wider and more evenly in the parameter space. This can be seen if one regards (**??**): If $\mathbf{d}$ is picked such that $\mathbf{d}\theta_i$ has a high probability of being a small number, then $\mathbf{d}\theta_i + \sigma_{\mathbf{y}}^2 \mathbf{z}_j$ is going to get dominated by the noise term $\sigma_{\mathbf{y}}^2 \mathbf{z}_j$. Thus, the log-posterior $\log p(\theta_i | \mathbf{d}_i + \sigma_{\mathbf{y}}^2 \mathbf{z}_j, \mathbf{d})$ will be smaller. If on the other hand, $\mathbf{d}$ is picked such that $\mathbf{d}\theta_i$ has a high probability of being a large number, then $\mathbf{d}\theta_i$ is going to dominate the noise term and the posterior has a higher probability of being a large number.
In Figure **??**, we can see if the start position consists of 10 data points on top of each other at $(0, 0)$, they will swiftly move to a more spread out formation.
If we choose a zero-mean prior with an identity covariance, the points will spread out instantly from the center. If we choose a non-zero mean and non-identity covariance, the gradient seems to become more steep such that some of the points can move away quite fast. If we insert a regularization factor $\mathcal{L} = 10^{-2} \times |\mathbf{d}|^2$, the points will diverge confidently from the start

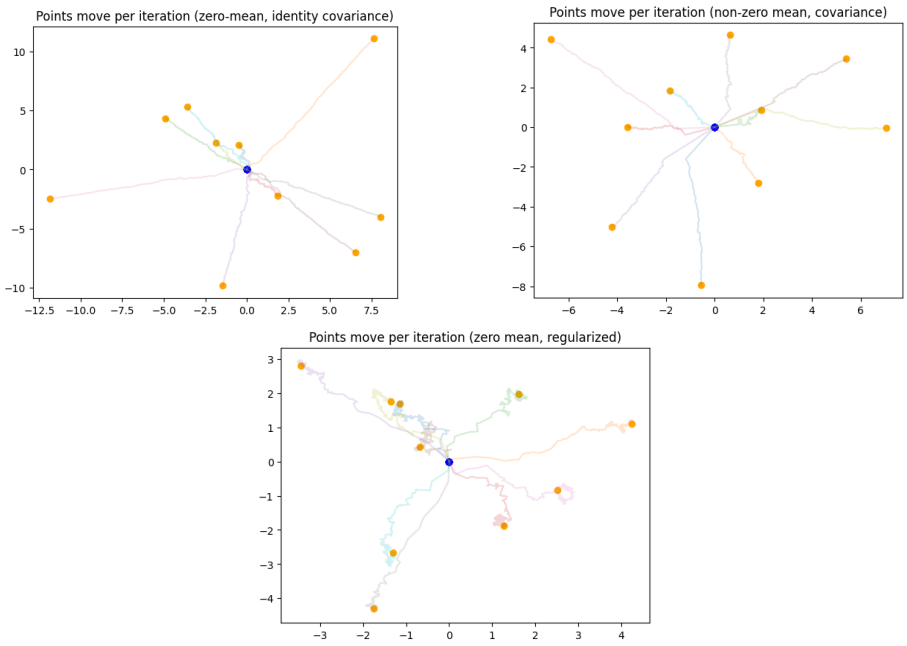

Figure 3: Points in **d**-space moving from their start position (blue) and increasingly away from it.

position until some point, where they will start to stutter and not move much more.

Thus we've seen how optimizing over the Mutual Information metric can help find solutions to the Bayesian Optimal Design problem. Now we will move on to exploring how to solve regression problems, even when analytical solutions are not available with the hopes of opening up our Mutual Information optimizer to all kinds of models.

4 Week 3-4

4.1 Objective

This week, we will study and implement the variational inference framework with the objective of estimating the posterior in a way that is indifferent to the type of regression performed. We are going to pose the problem as an optimization problem and then derive an objective function that we can optimize over. We will then try to implement it and perform some adjustments for efficiency and numerical stability.

4.2 Theory

4.2.1 Variational Inference and Variational Families

In week 1, we were able to calculate the exact posterior using the right assumption and conjugacy between the prior and likelihood. This turns out to be a rarity - in most cases, a closed form solution for the posterior is not readily available. A common approach to this problem is to approximate the posterior distribution, using a *variational distribution* q picked from

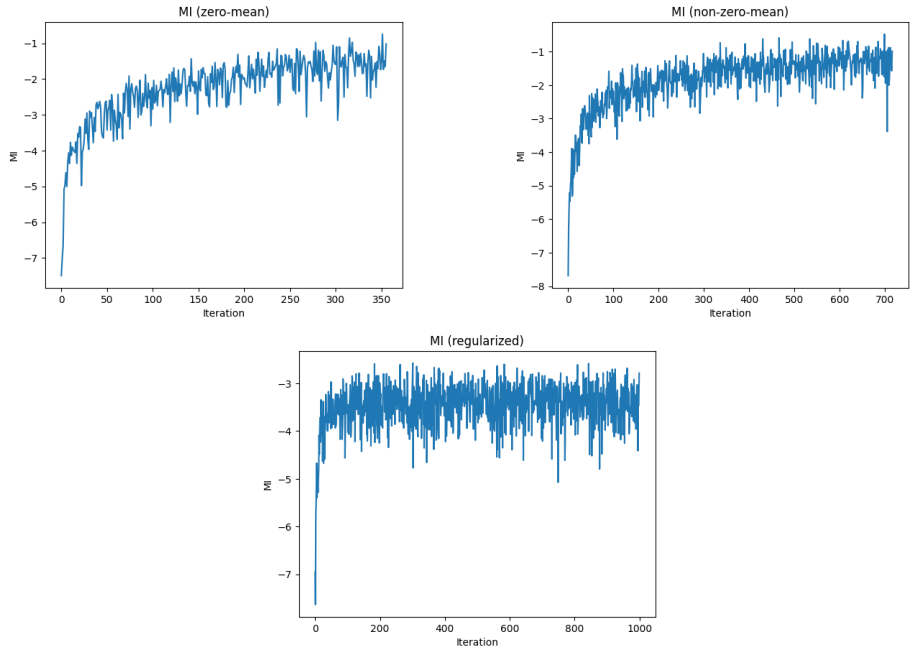

Figure 4: Convergence of each of the plots from Figure **??**

a family of distributions that we might imagine could approximate the posterior. By picking a suitable objective function, we can use optimization techniques to pick a suitable q. This is called *variational inference*. Thus, our goal is to find a q such that for any $\theta \in \Theta$, $q(\theta) \approx p(\theta|\mathbf{y}, \mathbf{d})$.

4.2.2 Deriving a suitable objective function

Without any further assumptions about our variational distribution, an immediate idea could be to use the KL-divergence between q and p as an objective function:

$$q^*(\theta) = \arg\min_q \mathrm{KL}(q(\theta)||p(\theta|\mathbf{y}, \mathbf{d})) \tag{39}$$

Calculating this requires us to calculate the posterior, which we would like to avoid since we are often not guaranteed to be able to compute the evidence term $\log(p(\mathbf{y}|\mathbf{d}))$ from (**??**). We can however rewrite the KL-divergence as follows **blei17**:

$$\mathrm{KL}(q(\theta)||p(\theta|\mathbf{y}, \mathbf{d})) = \mathbb{E}_{\theta\sim q}[\log q(\theta)] - \mathbb{E}_{\theta\sim q}[\log p(\theta|\mathbf{y}, \mathbf{d})] \tag{40}$$

and by conditioning

$$= \mathbb{E}_{\theta\sim q}[\log q(\theta)] - \mathbb{E}_{\theta\sim q}[\log p(\theta, \mathbf{y}|\mathbf{d})] - \log p(\mathbf{y}|\mathbf{d}) \tag{41}$$

One can now notice that the last term is actually constant with regards to q. Thus, when we optimize, we can ignore it. The remaining terms form the negative *evidence lower bound* (ELBO) **blei17**:

$$\mathrm{ELBO}(q) = -\mathrm{KL}(q(\theta)||p(\theta|\mathbf{y}, \mathbf{d}))+\mathrm{const} = \mathbb{E}_{\theta\sim q}[\log p(\theta, \mathbf{y}|\mathbf{d})]-\mathbb{E}_{\theta\sim q}[\log q(\theta)] \tag{42}$$

Since the KL-divergence consists of the negative ELBO and a constant, we can minimize the KL-divergence by maximizing the ELBO. Thus, we can reformulate our objective:

$$q^*(\theta) = \arg\max_q \text{ELBO}(q) \tag{43}$$

We can derive (**??**) to a more easily computable term:

$$\text{ELBO}(q) = \mathbb{E}_{\theta\sim q}[\log(p(\mathbf{y}|\theta,\mathbf{d})p(\theta|\mathbf{d}))] - \mathbb{E}_{\theta\sim q}[\log q(\theta)] \tag{44}$$

$$= \mathbb{E}_{\theta\sim q}[\log p(\mathbf{y}|\theta,\mathbf{d})] + \mathbb{E}_{\theta\sim q}[\log p(\theta|\mathbf{d})] - \mathbb{E}_{\theta\sim q}[\log q(\theta)] \tag{45}$$

Using that θ is independent of $\mathbf{d}$, we have

$$= \mathbb{E}_{\theta\sim q}[\log p(\mathbf{y}|\theta,\mathbf{d})] - (-\mathbb{E}_{\theta\sim q}[\log p(\theta)] + \mathbb{E}_{\theta\sim q}[\log q(\theta)]) \tag{46}$$

$$= \mathbb{E}_{\theta\sim q}[\log p(\mathbf{y}|\theta,\mathbf{d})] - \mathbb{E}_{\theta\sim q}\left[\log\left(\frac{q(\theta)}{p(\theta)}\right)\right] \tag{47}$$

Using the definition of the KL-divergence:

$$= \mathbb{E}_{\theta\sim q}[\log p(\mathbf{y}|\theta,\mathbf{d})] - \text{KL}(q||p) \tag{48}$$

Thus, this variational inference problem can work for any posterior, as long as one can calculate the expectation of the likelihood, and the KL-divergence between the variational distribution and the prior.

4.2.3 The Linear Regression Case

Let us now look at how one can find a good variational family for the linear regression case. An easy guess would be to pick q from a family of multivariate Gaussian distributions, since we know from conjugacy between the prior and likelihood that the posterior must also be multivariate Gaussian. Thus, one can expect q^* to be able to estimate p perfectly.
This has the nice consequence of turning the KL-divergence into a closed form expression, since both q and p follow multivariate normal distributions **kld-gaussian**.

$$\text{KL}(q||p) = \frac{1}{2}\left((\Sigma_p^{-1}\Sigma_q) + (\mu_p-\mu_q)^T\Sigma_p(\mu_p-\mu_q) - k + \log\frac{|\Sigma_p|}{|\Sigma_q|}\right) \tag{49}$$

We do have a problem, however: The first expectation of (**??**) is an integral without a closed-form solution. Thus, we wish to approximate it using Monte Carlo integration. To be able to sample θs from q, we will need to use the reparameterization trick: By property of the multivariate normal distribution, there must exist some matrix $\mathbf{A}$ such that $\theta \sim \mathcal{N}(\mu, \mathbf{A}\mathbf{A}^T)$ **krause22** and such that

$$\theta = \mu + \mathbf{A}\mathbf{z}, \quad \text{where } \mathbf{z} \sim \mathcal{N}(0, I_d) \tag{50}$$

Thus, we can simply sample N samples of $\mathbf{z}$ and calculate θ from that. This means that our ELBO estimate can be written as follows:

$$\text{ELBO}(q) \approx \frac{1}{N}\sum_{i=0}^{N}[\log p(\mathbf{y}|\mu + \mathbf{A}\mathbf{z}^{(i)}, \mathbf{d})] - \text{KL}(q||p) \tag{51}$$

μ and $\mathbf{A}$ are then the parameters of q that we will try to find through optimization.

4.3 Implementation

4.3.1 Main implementation

The main work of the algorithm is in calculating the ELBO. This can then be plugged into the same optimizer as in week 2. The ELBO can be implemented as follows in Python:

```
def ELBO(d, y, mean, A): # optimizing for mean, A
  zs = np.random.normal(size=(N, len(mean))) # N samples of
  ↪ size d
  likelihood_samples = []
  for z in zs:
    theta = mean + A @ z
    likelihood_samples.append(log_likelihood(y, theta, d))
  return 1/N * np.sum(likelihood_samples, axis=0) - KLD(mean,
  ↪ A, prior_mean, prior_A)
```

where the log-likelihood is implemented after (**??**):

```
def log_likelihood(y, theta, d):
  return log_pdf(y, theta @ d, noise * np.eye(len(theta)))
```

and the KL-divergence like so

```
def KLD(mean_q, A_q, mean_p, A_p):
    sigma_q = (A_q @ A_q.T)
    sigma_p = (A_p @ A_p.T)
    bar_sigma_q = np.linalg.det(sigma_q)
    bar_sigma_p = np.linalg.det(sigma_p)
    k = len(mean_q)
    return 0.5 * (np.trace(np.linalg.inv(sigma_p) @ sigma_q)
    ↪ + (mean_p - mean_q).T @ np.linalg.inv(sigma_p) @
    ↪ (mean_p - mean_q) - k +
    ↪ np.log(bar_sigma_p/bar_sigma_q))
```

4.3.2 Optimizing over a mean and a matrix

Like in week 2, it is not trivial to optimize over both a mean and a matrix. Thus, we will encode μ and $\mathbf{A}$ as a vector $\mathbf{v}$ of the shape

$$\mathbf{v} = \begin{bmatrix} \mu_0 \\ \vdots \\ \mu_d \\ \mathbf{A}_{1,1} \\ \vdots \\ \mathbf{A}_{1,d} \\ \mathbf{A}_{2,1} \\ \vdots \\ \mathbf{A}_{d,d} \end{bmatrix}$$

whenever we need to take the gradient or regard them in the context of the optimizer. The following help functions can help encode and decode μ and $\mathbf{A}$:

```python
def encode_q_params(q_params): # mean, A to vector
    mean, A = q_params
    return np.array(list(mean) + list(A.flatten()))
def decode_q_params(encoded_q, dim = 3): # vector to mean, A
    shape = len(encoded_q)
    A_shape = (int(np.sqrt(shape - dim)), int(np.sqrt(shape - dim)))
    mean = encoded_q[0:dim]
    A = encoded_q[dim:shape].reshape(A_shape)
    return mean, A
```

4.4 Results

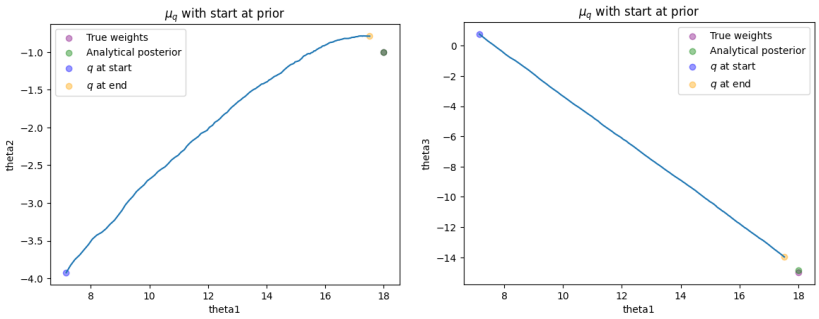

Figure 5: Change in mean of q after 200 iterations of the algorithm with $N = 10$, $\ell = 100$.

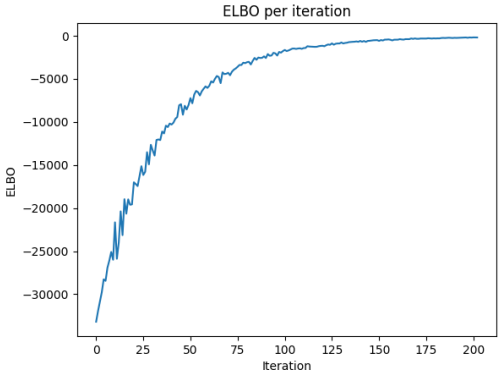

Figure 6: ELBO after each iteration of the algorithm with $N = 10$, $\ell = 100$.

As one can see in Figure **??**, the mean of q confidently moves towards the analytical posterior (and true weights). It can also been seen in Figure **??** that the ELBO converges after 200 or so iterations, indicating that we have indeed found an optimum.

4.5 Evaluation

The algorithm presented here works as expected, and functions as a possible replacement for the analytical posterior. We've been able to utilize the ELBO as a proxy for the KL-divergence and created a framework that can fitted to many different kind of regression problems, as long as one can estimate the log-likelihood and compute the KL-divergence between the variational distribution and the prior.

5 Week 5-6

5.1 Objective

For the final section of the project, we will try to see how to combine the Bayesian Optimal Design problem from week 2 with the variational inference framework from week 3-4, leading to a general framework for Bayesian Optimal Design that scales to many different kinds of model.

5.2 Theory

5.2.1 Finding the gradient of the Mutual Information

In week 2, we neatly assumed that autograd.grad was able to find the gradient of the mutual information with respect to $\mathbf{d}$. When one uses variational inference to compute the posterior, this becomes nontrivial, since one needs to differentiate through the parameters as a function of the design **d lorraine19**. We can see this from (**??**):

$$I(\mathbf{d}) = \mathbb{E}_\theta[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(p(\theta|\mathbf{y},\mathbf{d})) - \log p(\theta)]] \tag{52}$$

When using variational inference, this changes to

$$I(\mathbf{d}) \approx \mathbb{E}_\theta[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(q^*(\theta)) - \log p(\theta)]], \text{ where } q^*(\theta) = \arg\max_q \text{ELBO}_{\mathbf{d},\mathbf{y}}(q) \tag{53}$$

From now on, we will explicitly denote what $\mathbf{d}$ and $\mathbf{y}$ the ELBO is taken with regards to. Taking the gradient of (**??**) and using reparamerization results in

$$\nabla_\mathbf{d} I(\mathbf{d}) = \nabla_\mathbf{d} \mathbb{E}_\theta[\mathbb{E}_{\mathbf{y}|\theta,\mathbf{d}}[\log(q^*(\theta)) - \log p(\theta)]] \tag{54}$$

$$\approx \mathbb{E}_\mathbf{z}[\mathbb{E}_\epsilon[\nabla_\mathbf{d} \log(q^*_{\mathbf{y}',\mathbf{d}}(\theta'))]], \tag{55}$$

$$\text{where } \mathbf{y}' = \mathbf{d}\theta' + \epsilon,\ \theta' = \mu_\theta + \mathbf{A}_\theta \mathbf{z},\ \epsilon \sim \mathcal{N}(0, \sigma^2_\mathbf{y} I_N),\ \mathbf{z} \sim \mathcal{N}(0, I_M)$$

Using the chain rule gives us

$$= \mathbb{E}_\mathbf{z}[\mathbb{E}_\epsilon[\frac{1}{q^*_{\mathbf{y}',\mathbf{d}}(\theta')} \times \nabla_\mathbf{d} q^*_{\mathbf{y}',\mathbf{d}}(\theta'))]], \tag{56}$$

Thus we need to calculate $\nabla_\mathbf{d} q^*_{\mathbf{y}',\mathbf{d}}(\theta')$, or more specifically

$$\nabla_\mathbf{d} q^*_{\mathbf{y}',\mathbf{d}}(\theta') = \nabla_\mathbf{d} \arg\max_q \text{ELBO}_{\mathbf{d},\mathbf{y}'}(q)\Big|_\theta \tag{57}$$

If we let λ be the parameters of q encoded as a vector, we can regard q^* as a function of $\mathbf{d}$. Thus we can use the chain rule.

$$\nabla_\mathbf{d} q^*_{\mathbf{y}',\mathbf{d}}(\theta') = \nabla_\lambda q^\lambda_{\mathbf{y}',\mathbf{d}}(\theta') \underbrace{\nabla_\mathbf{d} \arg\max_\lambda \text{ELBO}_{\mathbf{d},\mathbf{y}'}(q^\lambda)}_{\text{Jacobian}} \tag{58}$$

The gradient, $\nabla_\lambda q^\lambda_{\mathbf{y}',\mathbf{d}}$, is simple to perform using automatic differentiation. The Jacobian of the variational parameters with regards to the design $\mathbf{d}$, needs very careful consideration. We will approach this using the Implicit Function Theorem **lorraine19**:

Theorem 1. *Let f be a continuously differentiable function from $\mathbb{R}^n \times \mathbb{R}^m$ to $\mathbb{R}^m$. Fix a point (a, b) such that $f(a, b) = \mathbf{0}$. If the Jacobian $J^f_b(a, b)$ is invertible, then there must exist an open set $U \subseteq \mathbb{R}^n$ containing a such that there exists a continuously differentiable function $g : U \to \mathbb{R}^m$ such that $g(a) = b$ and $f(x, g(x)) = \mathbf{0}$ for all $x \in U$. Furthermore,*

$$\nabla_x g(x) = -[J^f_y(x, g(x))]^{-1} J^f_x(x, g(x))$$

We will now adapt Theorem **??** to our problem. From now on, every mention of ELBO is with regards to $\mathbf{y} = \theta\mathbf{d} + \epsilon$ from (**??**) . Let $x = \mathbf{d}$, $y = \lambda$, $g(\mathbf{d}) = \arg\max_\lambda \text{ELBO}_\mathbf{d}(q^\lambda)$, $f(\mathbf{d}, \lambda) = \nabla_\lambda \text{ELBO}_\mathbf{d}(q^\lambda)$.
First, we need to fix a point $(\mathbf{d}', \lambda')$ such that $\nabla_\lambda \text{ELBO}_{\mathbf{d}'}(q^{\lambda'}) = 0$. If we take any $\mathbf{d}'$, and pick λ' to be any local optimum $\lambda' = g(\mathbf{d})$, then the gradient must be 0 at that point. If the hessian then is invertible, we must have:

$$\nabla_\mathbf{d} \arg\max_\lambda \text{ELBO}_\mathbf{d}(q^\lambda)\Big|_\theta = -[\underbrace{\nabla^2_\lambda \text{ELBO}_\mathbf{d}(q^*)}_{\text{variational hessian}}]^{-1} \underbrace{\nabla_\lambda \nabla_\mathbf{d} \text{ELBO}_\mathbf{d}(q^*)}_{\text{variational mixed partials}} \tag{59}$$

Computing both the variational hessian and the variational mixed partials should then be possible using automatic differentiation.

5.2.2 Adapting to sampling

Again, we will approach calculating the expectations using Monte Carlo sampling. From (**??**), we will instead compute

$$\nabla_{\mathbf{d}} I(\mathbf{d}) \approx \sum_{i}^{N} \sum_{j}^{M} \frac{1}{q^*_{\mathbf{d},\mathbf{y}_{ij}}(\theta_j)} \times \nabla_{\mathbf{d}} q^*_{\mathbf{d},\mathbf{y}_{ij}}(\theta_j) \tag{60}$$

where $\mathbf{y}_{ij} = \mathbf{d}\theta_j + \epsilon_i,\ \theta_j = \mu_\theta + \mathbf{A}_\theta \mathbf{z}_j,\ \epsilon_i \sim \mathcal{N}(0, \sigma^2_{\mathbf{y}} I_\ell),\ \mathbf{z} \sim \mathcal{N}(0, I_M)$

To try to keep numerically stable, we will use the log-sum-exp trick:

$$= \sum_{i}^{N} \sum_{j}^{M} \exp(\log \nabla_{\mathbf{d}} q^*_{\mathbf{d},\mathbf{y}_{ij}}(\theta_j) - \log q^*_{\mathbf{d},\mathbf{y}_{ij}}(\theta_j) I_\ell) \tag{61}$$

5.3 Implementation

5.3.1 Reducing computation time

Since both the Mutual Information gradient and variational inference algorithm utilize sampling, a naive implementation could easily end up with many nested for-loops. To reduce this, batching techniques can be used to delegate computation to `numpy`, which can perform them in a lower level language. A big bottleneck is the computation of the Hessian and Jacobian of (**??**), which can be sped up by using more modern automatic differentiation libraries than autograd.

5.3.2 Correctness of implementation

Since we have the code and theoretical knowledge for each of the components that this implementation is composed of, we can test each part of the implementation by comparing it to its analytical counterpart. For example, one can replace the variational approximation with the analytical posterior to see how the variational approximation change the convergence of the Mutual Information metric. One can furthermore compare this to the results from week 2, where we used automatic differentiation to compute the entire gradient of the Mutual Information metric.

5.4 Results

It was not possible within the scope of this project to get a fully working implementation of the combined Bayesian Optimal Design and variational inference algorithms. One of the main problems is that a lot of noise is introduced when we compute the gradient ourselves. The following code snippet uses autograd to compute the gradient inside the expectations of (**??**):

```
1 def MI_grad(d):
2     N = 5
```

```
    M = 1
    results = []
    eps = np.random.randn(N, len(d)) * np.sqrt(noise)
    for i in range(N):
        zs = np.random.randn(M, 3)
        for j in range(M):
            theta_j = mu_prior + A_prior @ zs[j]
            y_ij = augment_d(d) @ theta_j + eps[i]
            def g(d):
                mean, cov =
                ↪ analytical_posterior_params(augment_d(d),
                ↪ y_ij, mu_prior=mu_prior,
                ↪ cov_prior=cov_prior)
                return mean, np.linalg.cholesky(cov)
            def q_(theta, lam):
                mean, A = lam
                return
                ↪ stable_multivariate_gaussian_logpdf(theta,
                ↪ mean, A @ A.T)
            results.append(encode_d(grad(lambda d:
            ↪ q_(theta_j, g(d)))(d)))
    return decode_d(np.mean(np.array(results), axis=0),
    ↪ dim=2)
```

This code produces results that can be shown in Figure **??**. As it can be seen, while it seems like the Mutual Information is very slowly rising, the algorithm is much more noisy, and does not produce plots that look like the ones we saw in week 2 - even when the two implementations should be equivalent.

If this did work, we could implement the entire gradient of the Mutual Information like so.

```
def q_grad(theta, q_params, d, y_i):
    q_mean, q_A = q_params
    inverse_hessian = np.linalg.inv(hessian(lambda eq:
    ↪ ELBO(augment_d(d), y_i, decode_q_params(eq)[0],
    ↪ decode_q_params(eq)[1]))(encode_q_params(q_params)))
    mixed_partials = jacobian(lambda eq: grad(lambda d:
    ↪ ELBO(augment_d(decode_d(d, dim=2)), y_i,
    ↪ decode_q_params(eq)[0],
    ↪ decode_q_params(eq)[1]))(encode_d(d)))(encode_q_params(q_params)).T
```

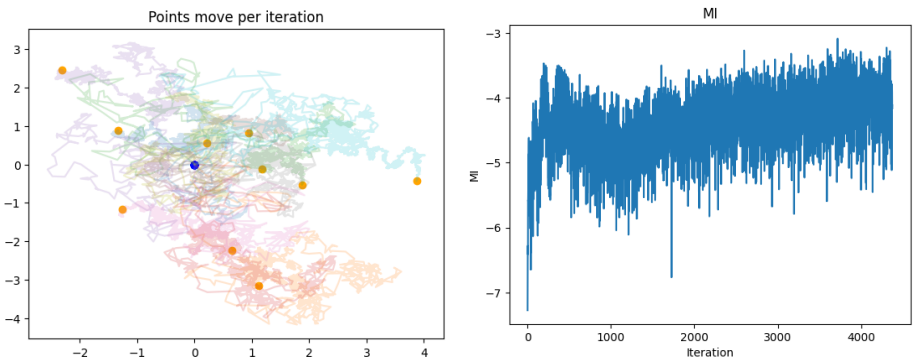

Figure 7: Movement of points using gradient from (**??**). Blue is start position, orange is end position.

```
    jac = - inverse_hessian @ mixed_partials
    return jac
def MI_grad_5(d):
    N = 5
    M = 1
    results = []
    eps = np.random.randn(N, len(d)) * np.sqrt(noise)
    for i in range(N):
        zs = np.random.randn(M, 3)
        for j in range(M):
            theta_j = mu_prior + A_prior @ zs[j]
            y_ij = augment_d(d) @ theta_j + eps[i]
            def g(d):
                mean, cov =
                ↪ analytical_posterior_params(augment_d(d),
                ↪ y_ij, mu_prior=mu_prior,
                ↪ cov_prior=cov_prior)
                return mean, np.linalg.cholesky(cov)
            def q_(theta, lam):
                mean, A = lam
                return
                ↪ stable_multivariate_gaussian_logpdf(theta,
                ↪ mean, A @ A.T)

            def inner_objective_f(encoded_q_params):
                mean, A = decode_q_params(encoded_q_params)
                elbo = ELBO(augment_d(d), y_ij, mean, A)
                return -elbo
```

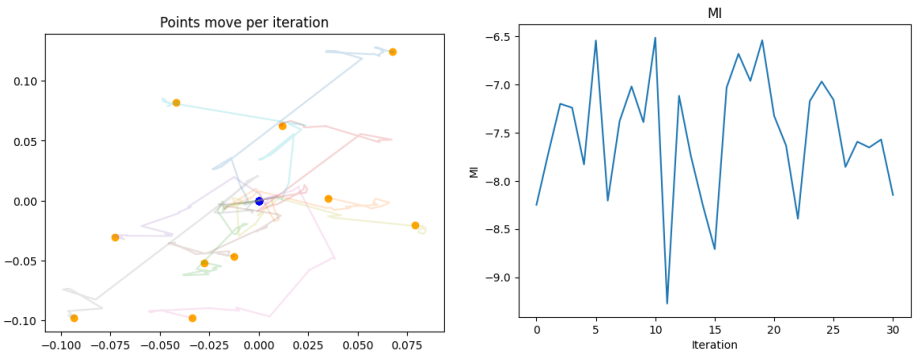

Figure 8: Movement of points from the full algorithm. Blue is start position, orange is end position.

```
            q_params =
            ↪ decode_q_params(optimize(grad(inner_objective_f),
            ↪ encode_q_params((mu_prior, A_prior)), 1e-1,
            ↪ 2, 0, 1e2))
            g1 = encode_q_params(grad(lambda lam: q_(theta_j,
            ↪ lam))(q_params))
            g2 = q_grad(theta_j, q_params, d, y_ij)
            actual = g1 @ g2
            c_part = 1/q_(theta_j, q_params)
            results.append(c_part * actual)
    return np.mean(np.array(results), axis=0)
```

Running the nested optimization is very inefficient, with one iteration of the outer optimization algorithm taking ~ 29 seconds on an average laptop. Testing shows that if one replaces the inner optimization result (`q_params`) with one obtained through the analytical posterior, the results are similar to that of Figure **??**. If one instead uses the variational posterior, the gradient becomes very large and thus the components of $\mathbf{d}$ quickly become too large to represent using floating point numbers. Reducing the learning rate by a factor of 10^{-2} can mitigate this partially but the points still behave very erratically. An example of this can be seen in Figure **??** although this example is only after 30 iterations due to the inefficiency of the algorithm. As it can be seen, we cannot conclude that the algorithm converges nor actively moves towards an optimum due to high noise and low amount of iterations.

6 Conclusion

Using the Mutual Information as an objective function, it was shown possible to optimize the locations of datapoints of a linear regression model to maximize the expected information gain from measuring at such datapoints. This was shown when using the exact posterior distribution and

shown theoritically using the variational approximation of the posterior distribution. The latter required the use of the Implicit Function Theorem to differentiate through the nested optimization problems. Implementing the gradient efficiently and correctly using the Implicit Function Theorem was however not possible within the timeframe of the project, so it is yet to show the practicality of the method.

While the theoritical derivations of the method are kept simple by using a linear regression model, the implementation is still very inefficient due to the nested optimization, even after several batching initiatives were taken. Since the optimization relied on sampling methods, the outer optimization algorithm required stochastic optimization methods, which usually requires a lot of iterations to converge. Thus, a lot of futher trial-and-error is required to make the implementation suitable for real-world applications. Specifically, the implementation was not tested on other types of model, where an optimal design is easier to recognize, which could lead to more confidence in the correctness of the implementation. Another area of futher research is within using parallelization to compute multiple samples of the gradient at once.

References

[1] E. G. Ryan, C. C. Drovandi, J. M. McGree, and A. N. Pettitt, "A review of modern computational algorithms for bayesian optimal design," *International Statistical Review*, vol. 84, no. 1, pp. 128–154, 2016. DOI: https://doi.org/10.1111/insr.12107. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/insr.12107. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/insr.12107.