Homework 4

Kevin Yang - 50244152

November 4, 2021

Link to repo:

https://github.com/keviny2/CPSC532W-Assignments/tree/main/FOPPL

Program 0

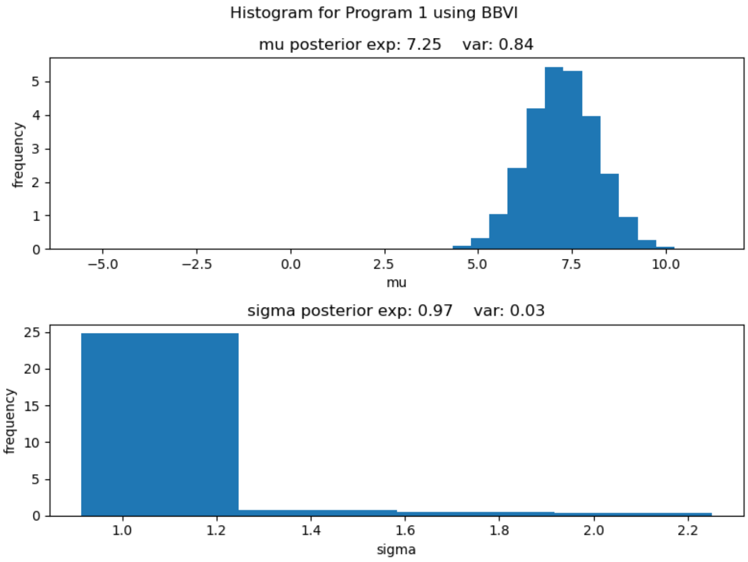
BBVI implementation:

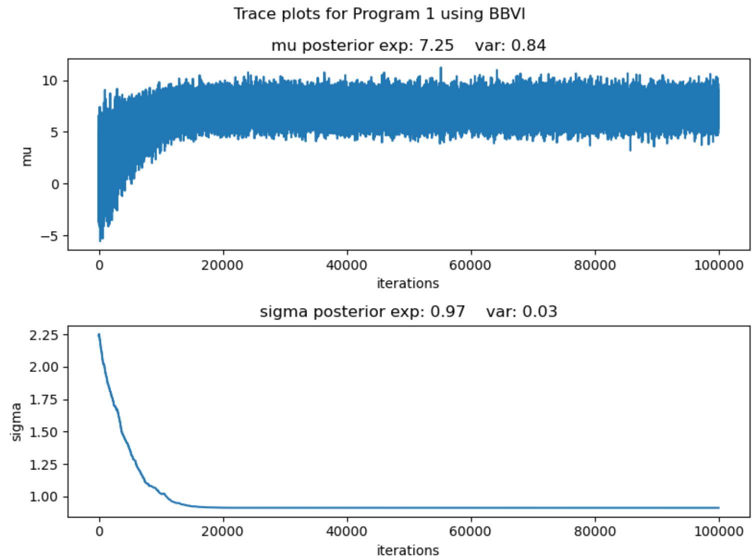
```python
        for l in range(L):
            # reset sig['logW'] at each iteration
            sig['logW'] = 0
            sig['G'] = {}

            # r_tl is the return value of the expression (won't have a value for each parameter)
            r_tl, sig_tl = evaluate_program(ast, sig, self.method)

            G_tl = clone(sig_tl['G'])

            # cause all of a sudden we want to keep track of sigma now too in hw4...
            if num == 1:
                r_tl = torch.FloatTensor([r_tl, next(iter(sig['Q'].values())).scale.clone().detach()])

            # add to return list
            samples.append([r_tl, sig_tl['logW'].clone().detach()])

            # add to list for self.elbo_gradient and self.optimizer_step functions
            G_t.append(G_tl)
            sig['logW_list'].append(sig_tl['logW'].clone().detach())

        g_hat = self.elbo_gradients(G_t, sig['logW_list'])
        bbvi_loss.append(torch.mean(torch.tensor(sig['logW_list'])))  # make a copy of the ELBO

        sig['Q'] = self.optimizer_step(sig, g_hat)

    print('Variational distribution: {}'.format(sig['Q']))

    return samples, bbvi_loss
```

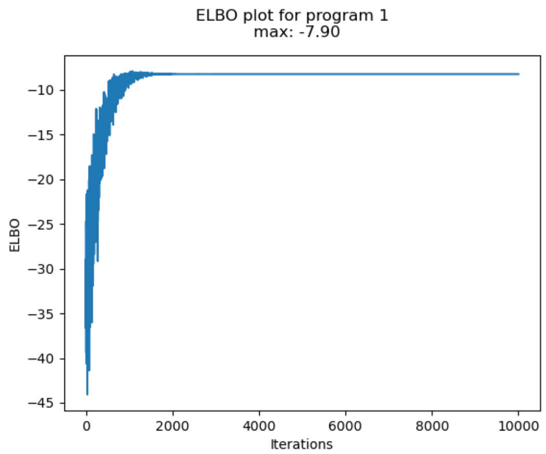
```python
def elbo_gradients(self, G, logW):
    """
    compute estimate for ELBO gradient

    :param G: list of gradients of log q -> \grad_{\lambda_{v,d}} \log{q(X_v^l ; \lambda_v}
    :param logW: list of logW of importance weight
    :return: dictionary g_hat that contains gradient components for each variable v
    """

    L = len(G)  # get number of parameters

    # obtain the union of all gradient maps
    union = []
    for G_i in G:
        union += list(G_i.keys())
    union = list(set(union))

    # dictionary containing our gradient estimates for each variable v
    g_hat = {}
    for v in union:
        num_params = len(G[0][v])
        # tensors for computing b_hat afterwards
        F_one_to_L_v = torch.empty(0)
        G_one_to_L_v = torch.empty(0)
        for l in range(L):
            if v in list(G[l].keys()):
                F_l_v = G[l][v] * logW[l]
            else:
                F_l_v, G[l][v] = torch.zeros(num_params), torch.zeros(num_params)
            # saving each F_l_v and G[l][v] for future computations
            F_one_to_L_v = torch.cat((F_one_to_L_v, F_l_v), 0)
            G_one_to_L_v = torch.cat((G_one_to_L_v, G[l][v]), 0)

        # reshape the tensors
        F_one_to_L_v = torch.reshape(F_one_to_L_v, (L, num_params))
        G_one_to_L_v = torch.reshape(G_one_to_L_v, (L, num_params))

        # line 16 & 17 in Alg. 12 seem to be incorrect; this is following equations 4.41-4.44 instead
        b_hat = torch.empty(0)
        for d in range(F_one_to_L_v.size()[1]):
            F_v_d = F_one_to_L_v[:, d]
            G_v_d = G_one_to_L_v[:, d]
            cov_F_G = np.cov(F_v_d.numpy(), G_v_d.numpy())
            b_hat = torch.nan_to_num(torch.cat((b_hat, torch.FloatTensor([cov_F_G[0, 1] / cov_F_G[1, 1]])), 0), 1)

        g_hat[v] = torch.sum(F_one_to_L_v - b_hat * G_one_to_L_v, dim=0) / L
    return g_hat
```

```python
def optimizer_step(self, sig, g_hat):
    """

    :param sig: map containing state of bbvi
    :param g_hat:
    :return:
    """
    for v in list(g_hat.keys()):
        parameters = sig['Q'][v].Parameters()
        for idx, param in enumerate(parameters):
            if len(param) > 1:
                param.grad = torch.FloatTensor(-g_hat[v])
            else:
                param.grad = torch.FloatTensor([-g_hat[v][idx]])

        sig['O'][v].step()
        sig['O'][v].zero_grad()
    return sig['Q']
```

Program 1

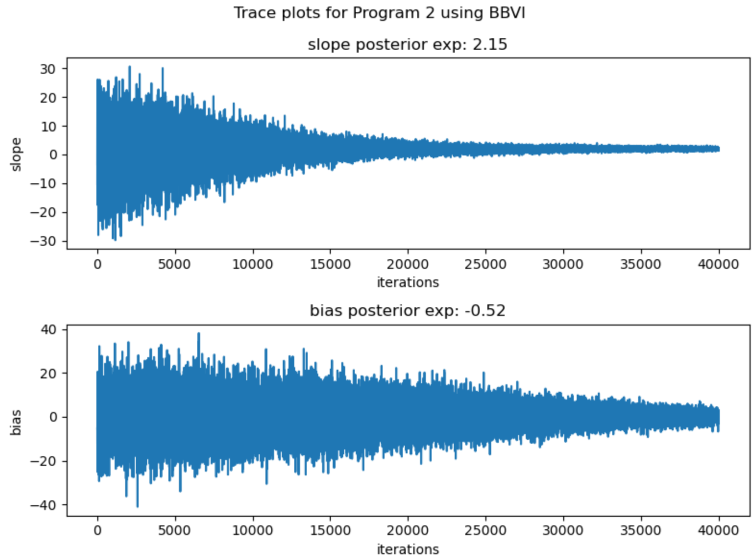


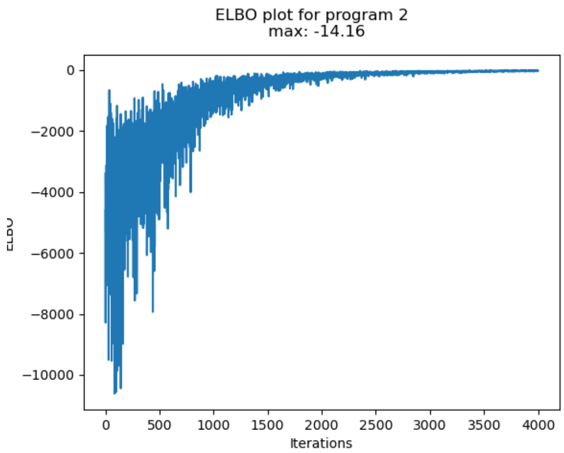





```
Took 298.65 seconds to finish Program 1
Posterior Expectation mu: tensor(7.2508)
Posterior Variance mu: tensor(0.8383)
Posterior Expectation sigma: tensor(0.9665)
Posterior Variance sigma: tensor(0.0347)
```

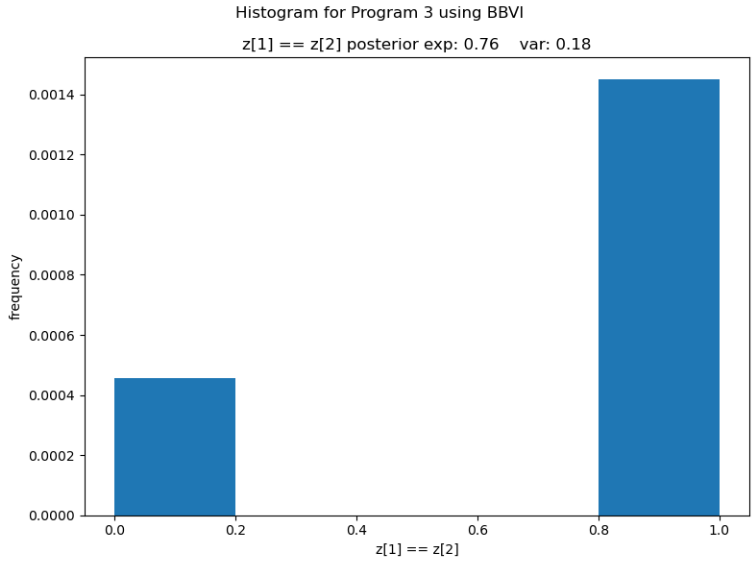
Program 2

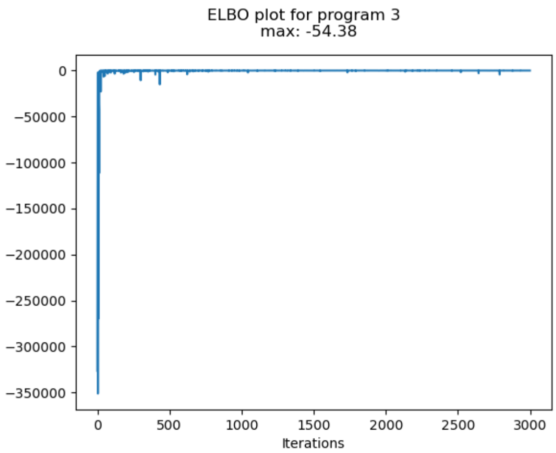





```
Took 289.66 seconds to finish Program 2
Posterior Expectation slope: tensor(2.1495)
posterior covariance of slope and bias:
 [[ 0.05515892 -0.19639921]
 [-0.19639921  0.86761098]]
Posterior Expectation bias: tensor(-0.5225)
```

Program 3

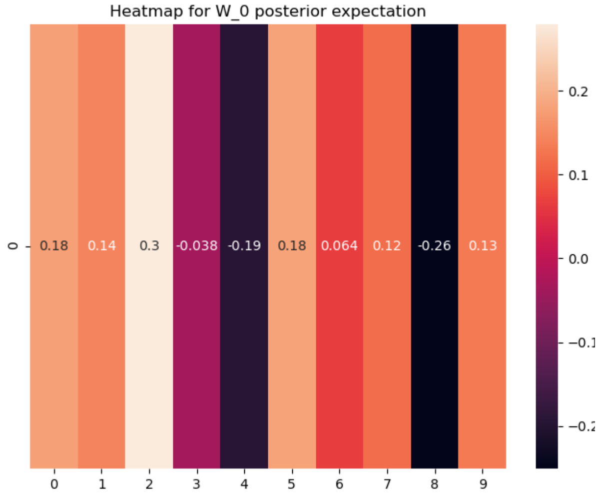


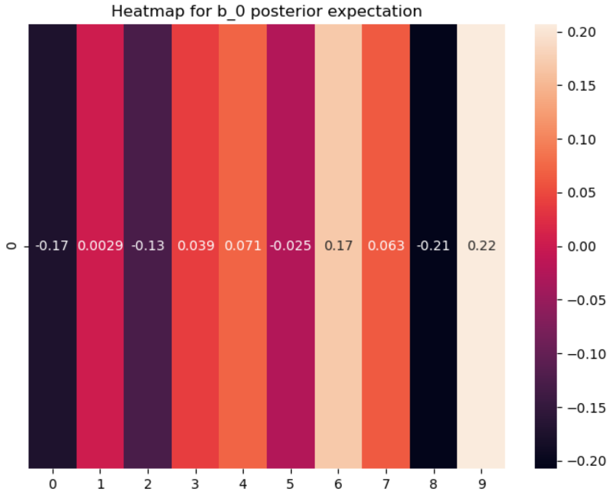


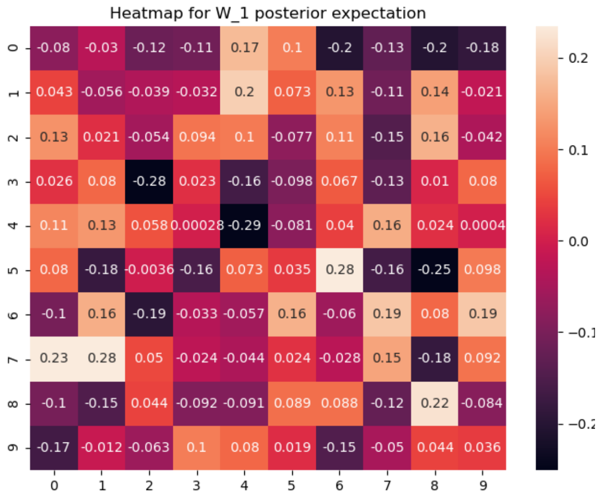

```
Took 3497.33 seconds to finish Program 3
Posterior Expectation z[1] == z[2]: tensor(0.7606)
Posterior Variance z[1] == z[2]: tensor(0.1821)
```

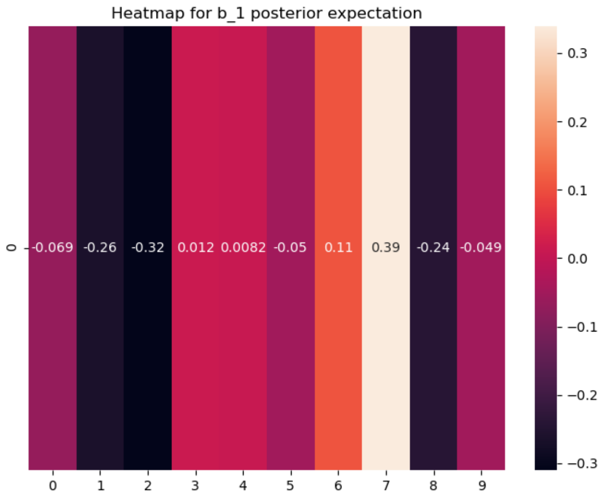
The mode-seeking behavior of VI on models with internal symmetries will find itself climbing different modes in the posterior over the course of inference. These internal symmetries render the model unidentifiable, because the model density is invariant under some classes of transformations of the latent parameter (Moore, 2016). This reduces the efficiency of inference procedures. The label switching problem is a well known problem in mixture models, as the latent variables representing class labels are unidentifiable in the sense that the model density does not change if you swap the labels of two classes - hence, *unidentifiable.*

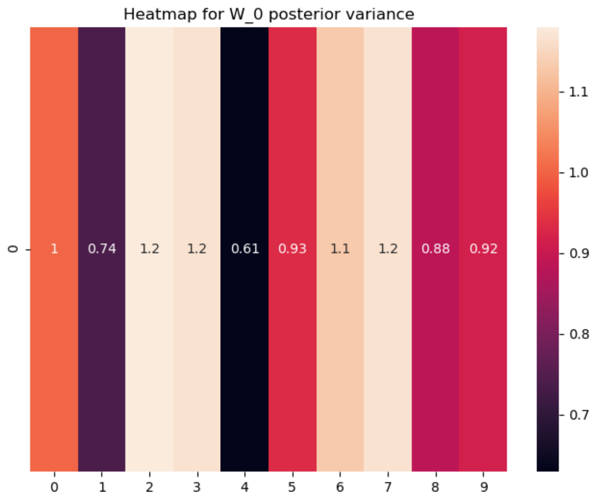
Program 4

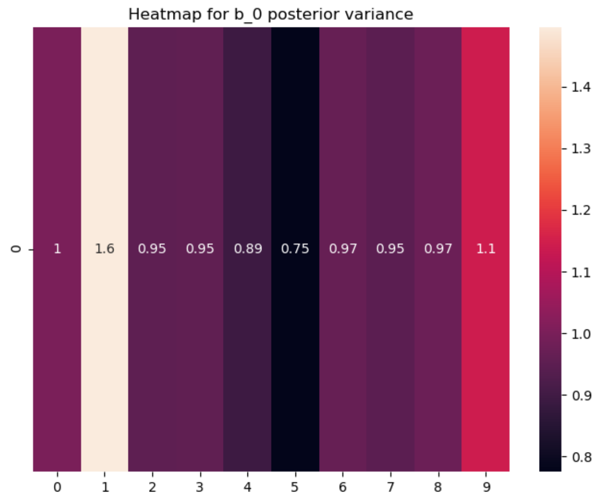

Heatmap for W_0 posterior expectation
0.18 0.14 0.3 -0.038 -0.19 0.18 0.064 0.12 -0.26 0.13

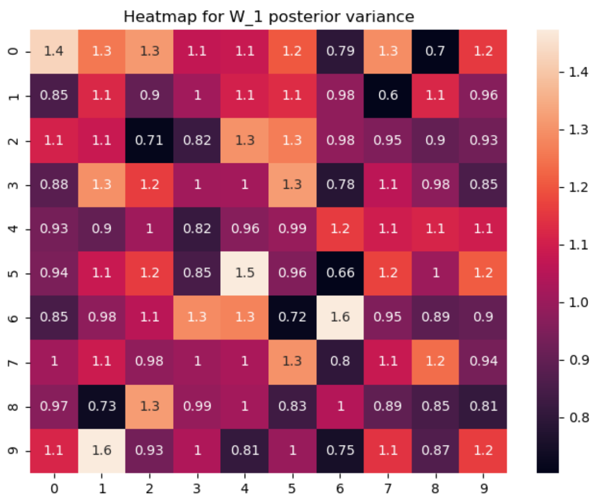

Heatmap for b_0 posterior expectation
-0.17 0.0029 -0.13 0.039 0.071 -0.025 0.17 0.063 -0.21 0.22

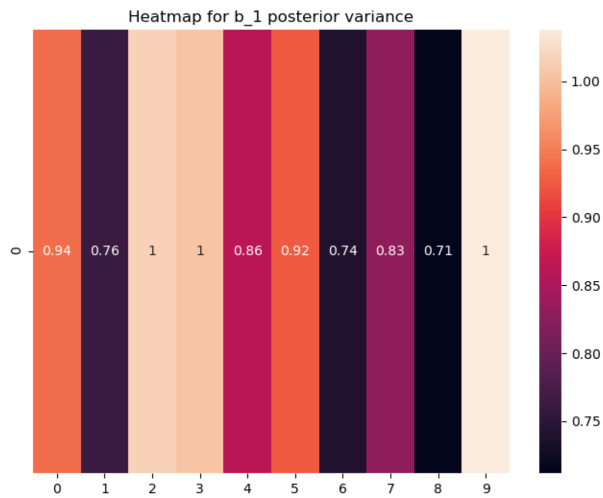

Heatmap for W_1 posterior expectation
-0.08 -0.03 -0.12 -0.11 0.17 0.1 -0.2 -0.13 -0.2 -0.18
0.043 -0.056 -0.039 -0.032 0.2 0.073 0.13 -0.11 0.14 -0.021
0.13 0.021 -0.054 0.094 0.1 -0.077 0.11 -0.15 0.16 -0.042
0.026 0.08 -0.28 0.023 -0.16 -0.098 0.067 -0.13 0.01 0.08
0.11 0.13 0.058 0.00028 -0.29 -0.081 0.04 0.16 0.024 0.0004
0.08 -0.18 -0.0036 -0.16 0.073 0.035 0.28 -0.16 -0.25 0.098
-0.1 0.16 -0.19 -0.033 -0.057 0.16 -0.06 0.19 0.08 0.19
0.23 0.28 0.05 -0.024 -0.044 0.024 -0.028 0.15 -0.18 0.092
-0.1 -0.15 0.044 -0.092 -0.091 0.089 0.088 -0.12 0.22 -0.084
-0.17 -0.012 -0.063 0.1 0.08 0.019 -0.15 -0.05 0.044 0.036

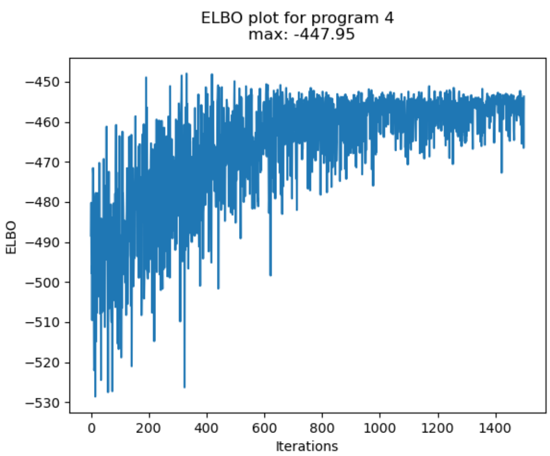

Heatmap for b_1 posterior expectation
-0.069 -0.26 -0.32 0.012 0.0082 -0.05 0.11 0.39 -0.24 -0.049









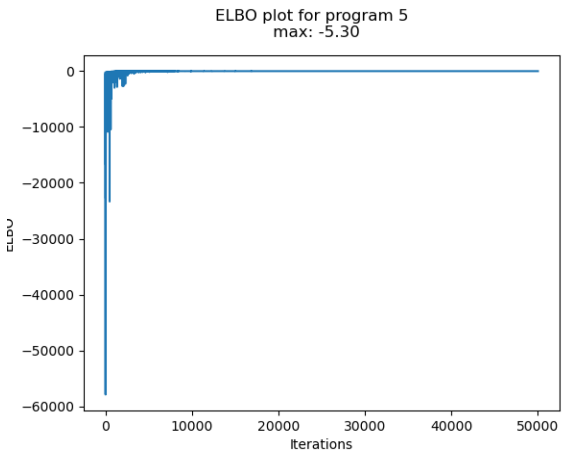


Took 2750.63 seconds to finish Program 4

Black-box variational inference (BBVI) is more generic compared to parameter estimation via gradient descent. In BBVI, we compute an estimate of the gradient of the evidence lower bound (ELBO) using a sampling procedure instead of obtaining the gradient of the ELBO analytically. This renders BBVI applicable to a range of complicated models that parameter estimation via gradient descent will not be able to handle.

Program 5

The variational distribution is a Gamma(7.79, 1.32)




```
Took 1427.37 seconds to finish Program 5
Variational distribution: Gamma(7.79, 1.32)
```

References

Moore, D. A. (2016) *Symmetrized Variational Inference.* NIPS Workshop on Advances in Approximate Inference.