CPSC532W - Homework 2

1 Importance Sampling

I implemented importance sampling using algorithm 7 in the evaluation-based approach. A critical, if not the only critical, aspect of this implementation is the addition of algorithm 7 in an if-else statement in evaluate_program which determines whether or not to do importance sampling or prior sampling. The implementation of algorithm 7 looks like:

```
print('Importance Sampling')
L = prog_args
importance_out = []
for l in range(L):
    r_l, sigma_l = eval(ast[-1],{'logW': 0},{})
    importance_out.append([r_l,sigma_l['logW']])

return importance_out
```

Program 1
Importance Sampling
elapsed time: 2.9012100999999997
expectation after 10000 samples is: tensor(7.2841)
variance is: tensor(0.7555)
Program 2
Importance Sampling
elapsed time: 14.776718100000004
expectation after 10000 samples is: tensor([2.1579, -0.5967])
variance is: tensor([0.0575, 0.8448])
Program 3
Importance Sampling
elapsed time: 17.359110200000003
expectation after 10000 samples is: tensor(0.9277)
variance is: tensor(0.0670)
Program 4
Importance Sampling
elapsed time: 6.129940199999993
expectation after 10000 samples is: tensor(0.3189)
variance is: tensor(0.2172)
The histograms of the outputs are:

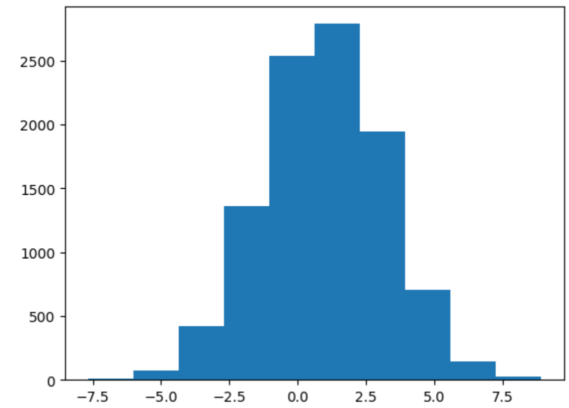

Figure 1: Program 1

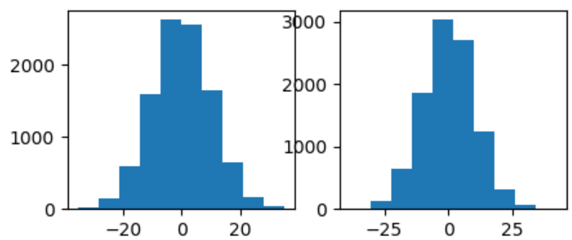

Figure 2: Program 2

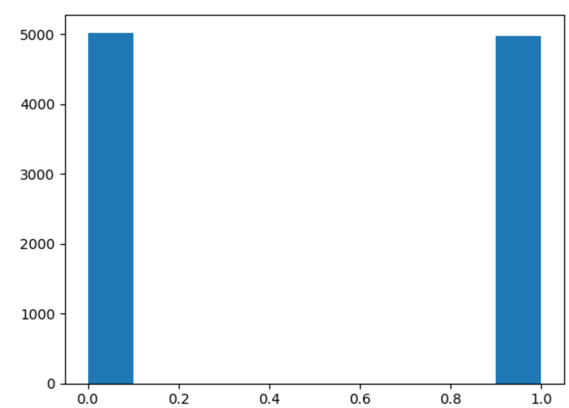

Figure 3: Program 3

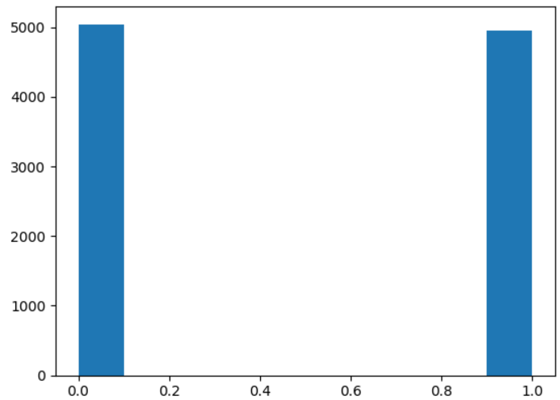

Figure 4: Program 4

2 MH in Gibbs

I implemented Metropolis Hasting within Gibbs from algorithm 1 using the graph-based evaluator. Critical aspects of this implementation were the functions `gibbs`, `gibb_step`, and `accept` below:

```
def gibbs(graph,S):
    procs, model, expr = graph[0], graph[1], graph[2]
    nodes, edges, links, obs = model['V'], model['A'], model['P'], model['Y']
    sorted_nodes = topological_sort(nodes, edges)

    full_output = sample_from_joint(graph)
    X0 = full_output[2]
    Q = {}
    Q_temp = { k : links[k] for k in set(links) - set(obs) }
    for q_key in sorted_nodes: # sort Q topologically
        if q_key in list(Q_temp.keys()):
            Q[q_key] = Q_temp[q_key]
    X = [{k : X0[k] for k in list(Q.keys())}]
    X_out = [deterministic_eval(plugin_parent_values(expr,X[0]))]
    for q_key in list(Q.keys()):
        q = Q[q_key]
        q = nested_search('sample*','sampleS',q)
        Q[q_key] = plugin_parent_values(q,obs)
    for s in range(1,S+1):
        X.append(gibbs_step({**X[s-1]},Q))
        X_out.append(deterministic_eval(plugin_parent_values(expr,X[s])))
    return X_out
```

```
def gibbs_step(X,Q):
    for x in list(Q.keys()):
        q = Q[x]
        q = plugin_parent_values(q, X)
        x_sample = deterministic_eval(q)

        alpha = accept(x,x_sample,X,Q)
        u = torch.distributions.Uniform(0,1).sample()
```

```
        if bool(u < alpha):
            X[x] = x_sample

        return X
```

```
def accept(x,x_sample,X0,Q):
    Xp = {**X0}
    Xp[x] = x_sample
    q = Q[x] # q is the same for both X and X'
    q_expr0 = ["observeS",q[1],X0[x]]
    q_expr1 = ["observeS",q[1],x_sample]
    log_alpha = deterministic_eval(q_expr0) - deterministic_eval(q_expr1)

    Vx = [x]
    for X_key in list(X0.keys()):
        if child(x,Q[X_key]):
            Vx.append(X_key)
    for v in Vx:
        v_expr1 = ["observeS",plugin_parent_values(Q[v][1],Xp),X0[v]]
        v_expr0 = ["observeS",plugin_parent_values(Q[v][1],X0),X0[v]]
        log_alpha = log_alpha + deterministic_eval(v_expr1)
        log_alpha = log_alpha - deterministic_eval(v_expr0)
    return torch.exp(log_alpha)
```

Program 1
elapsed time is: 5.9493032
expectation after 10000 samples is: -1.6739925
variance after 10000 samples is: 42.757023
updated bc of incorrectly indented return statement:
elapsed time is: 6.2780973
Expectation of return values for program 1:
expectation after 10000 samples is: 1.3670127
variance after 10000 samples is: 25.695333

Program 2
elapsed time is: 5.811423700000006
expectation after 10000 samples is: [4.5246744, 11.137395]
variance after 10000 samples is: [497.57785, 0.0]
updated bc of incorrectly indented return statement:
elapsed time is: 11.648128799999995
Expectation of return values for program 2:
expectation after 10000 samples is: [-3.583997, -10.026635]
variance after 10000 samples is: [553.0141, 575.2689]

Program 3
elapsed time is: 5.824858800000001
expectation after 10000 samples is: 1.0
variance after 10000 samples is: 0.0
updated bc of incorrectly indented return statement:
elapsed time is: 79.6610701
Expectation of return values for program 3:

expectation after 10000 samples is: 0.32966703329667035
variance after 10000 samples is: 0.2209866804540424

Program 4
elapsed time is: 7.756563999999997
expectation after 10000 samples is: 0.0
variance after 10000 samples is: 0.0
updated bc of incorrectly indented return statement:
elapsed time is: 19.36754159999998
Expectation of return values for program 4:
expectation after 10000 samples is: 0.49335065
variance after 10000 samples is: 0.24995582

The histograms of the outputs for the updated values are:

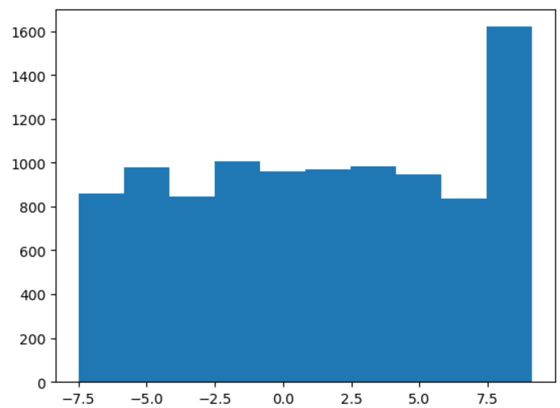

Figure 5: Program 1

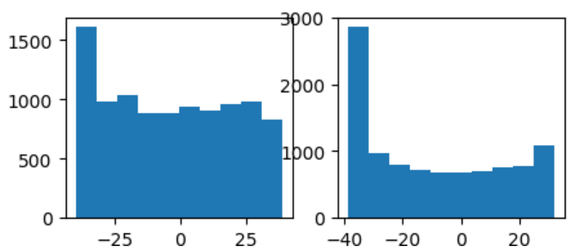

Figure 6: Program 2

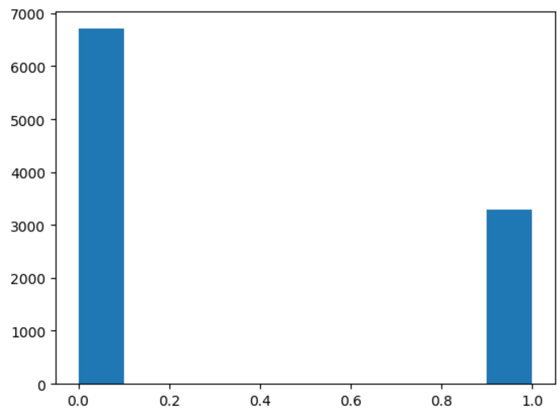

Figure 7: Program 3

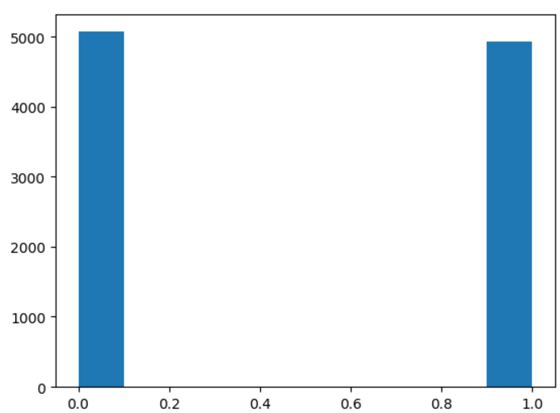

Figure 8: Program 4

And the plots of the trace for programs 1 and 2 are:

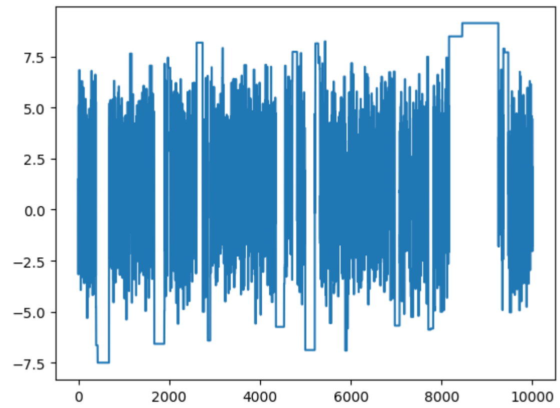

Figure 9: Program 1

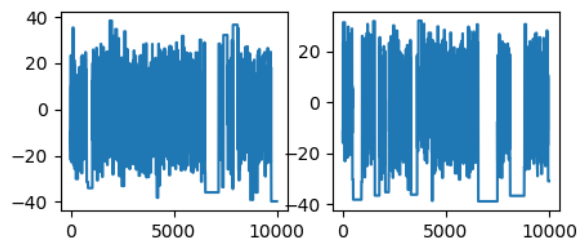

Figure 10: Program 2

Note: the plots don't look at all like the ones from importance sampling, and I'm more inclined to trust the importance sampling results.. So, there's at least one thing wrong in my MH in Gibbs code somewhere.

3 HMC and program 5

I didn't have time to get to these.

Appendix A - Code

`https://github.com/e-vic/cpsc532hw`