

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:

```

1 print('Importance Sampling')
2 L = prog_args
3 importance_out = []
4 for l in range(L):
5     r_l, sigma_l = eval(ast[-1], {'logW': 0}, {})
6     importance_out.append([r_l, sigma_l['logW']])
7
8 return importance_out

```

Program 1

Importance Sampling

elapsed time: 2.901210099999997

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:

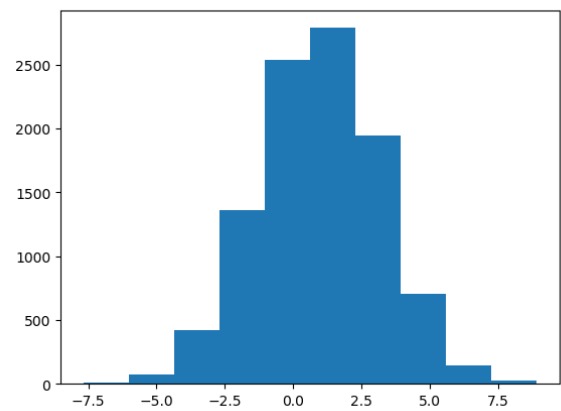


Figure 1: Program 1

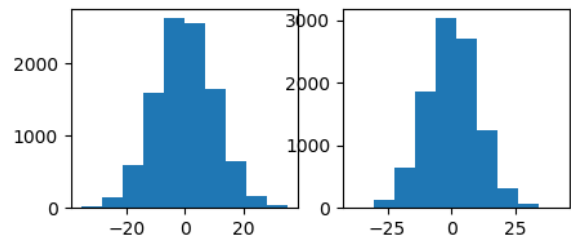


Figure 2: Program 2

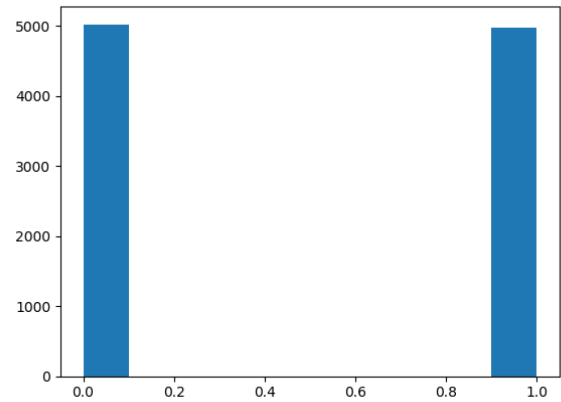


Figure 3: Program 3

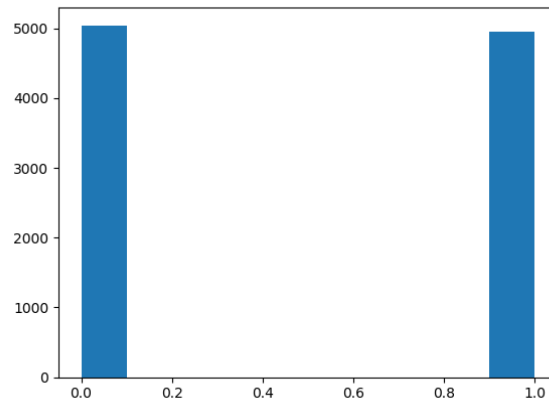


Figure 4: Program 4

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:

```

1 def gibbs(graph,S):
2     procs, model, expr = graph[0], graph[1], graph[2]
3     nodes, edges, links, obs = model['V'], model['A'], model['P'], model['Y']
4     sorted_nodes = topological_sort(nodes, edges)
5
6     full_output = sample_from_joint(graph)
7     X0 = full_output[2]
8     Q = {}
9     Q_temp = { k : links[k] for k in set(links) - set(obs) }
10    for q_key in sorted_nodes: # sort Q topologically
11        if q_key in list(Q_temp.keys()):
12            Q[q_key] = Q_temp[q_key]
13    X = [{k : X0[k] for k in list(Q.keys())}]
14    X_out = [deterministic_eval(plugin_parent_values(expr,X[0]))]
15    for q_key in list(Q.keys()):
16        q = Q[q_key]
17        q = nested_search('sample*', 'sampleS', q)
18        Q[q_key] = plugin_parent_values(q, obs)
19    for s in range(1,S+1):
20        X.append(gibbs_step({**X[s-1]},Q))
21        X_out.append(deterministic_eval(plugin_parent_values(expr,X[s])))
22    return X_out

```

```

1 def gibbs_step(X,Q):
2     for x in list(Q.keys()):
3         q = Q[x]
4         q = plugin_parent_values(q, X)
5         x_sample = deterministic_eval(q)
6
7         alpha = accept(x,x_sample,X,Q)
8         u = torch.distributions.Uniform(0,1).sample()

```

```

9         if bool(u < alpha):
10             X[x] = x_sample
11
12         return X

1 def accept(x, x_sample, X0, Q):
2     Xp = {**X0}
3     Xp[x] = x_sample
4     q = Q[x] # q is the same for both X and X'
5     q_expr0 = ["observeS", q[1], X0[x]]
6     q_expr1 = ["observeS", q[1], x_sample]
7     log_alpha = deterministic_eval(q_expr0) - deterministic_eval(q_expr1)
8
9     Vx = [x]
10    for X_key in list(X0.keys()):
11        if child(x, Q[X_key]):
12            Vx.append(X_key)
13    for v in Vx:
14        v_expr1 = ["observeS", plugin_parent_values(Q[v][1], Xp), X0[v]]
15        v_expr0 = ["observeS", plugin_parent_values(Q[v][1], X0), X0[v]]
16        log_alpha = log_alpha + deterministic_eval(v_expr1)
17        log_alpha = log_alpha - deterministic_eval(v_expr0)
18    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

Program 2

elapsed time is: 5.8114237000000006

expectation after 10000 samples is: [4.5246744, 11.137395]

variance after 10000 samples is: [497.57785, 0.0]

Program 3

elapsed time is: 5.8248588000000001

expectation after 10000 samples is: 1.0

variance after 10000 samples is: 0.0

Program 4

elapsed time is: 7.7565639999999997

expectation after 10000 samples is: 0.0

variance after 10000 samples is: 0.0

The histograms of the outputs are:

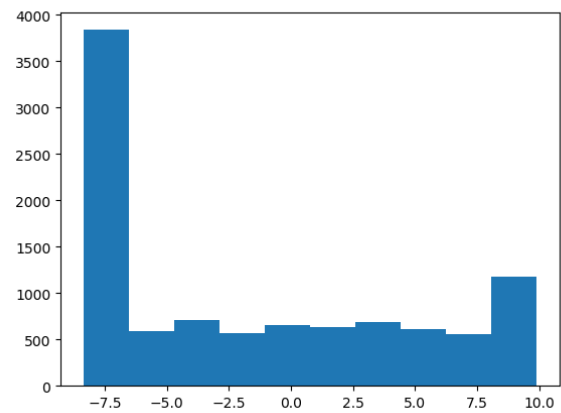


Figure 5: Program 1

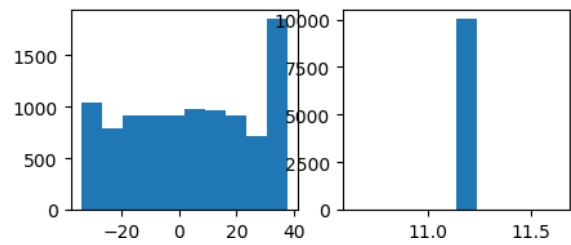


Figure 6: Program 2

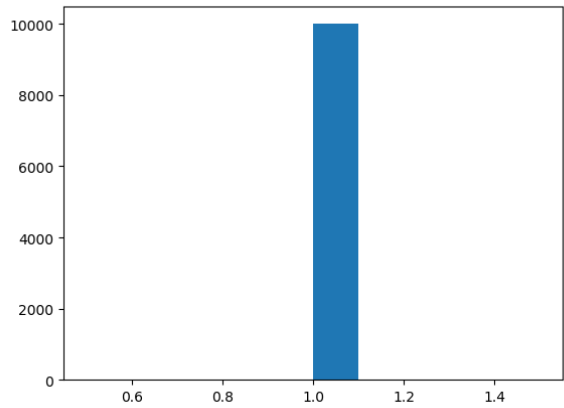


Figure 7: Program 3

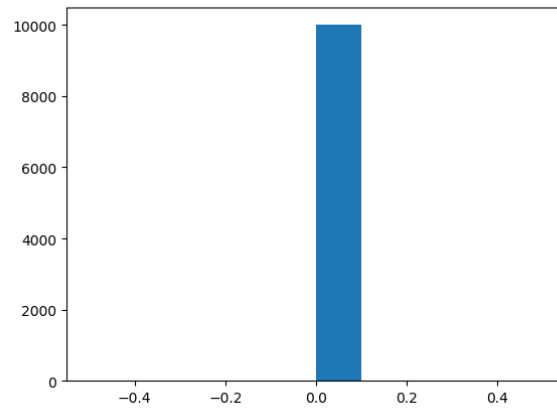


Figure 8: Program 4

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

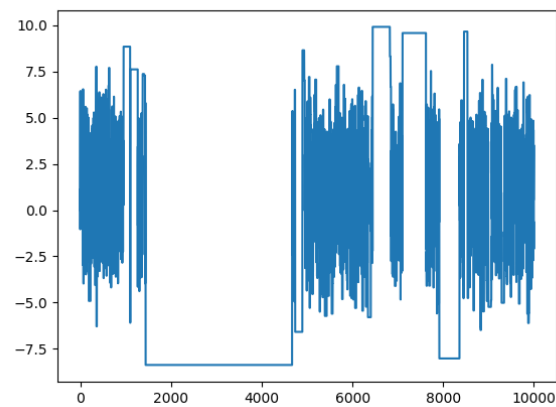


Figure 9: Program 1

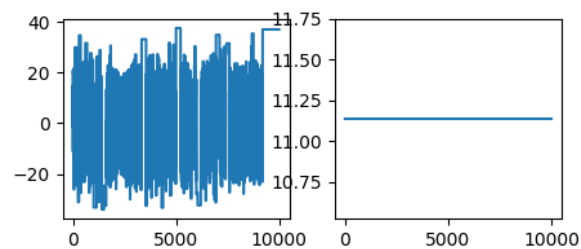


Figure 10: Program 2

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>