Citrine Applied Scientist Challenge Report

Xiaoting Zhong
xiaotinz@andrew.cmu.edu

1 Notes

Because I implemented two algorithms, I named them sampler_iterative and sampler_batch instead of sampler as instructed in the hand out. Hope this is not a problem.

2 Dependencies

- numpy: `https://scipy.org/install.html`
- networkx: `https://pypi.org/project/networkx/2.2/`

Note because I use anaconda, my python path is different from the default. In order to properly run my code, change the first line of the script from `!/Users/xiaotingzhong/anaconda2/bin/python` to `!/usr/local/bin/python`. The path can be found by typing `which python` in terminal.

3 Performance

I implemented two algorithms, an iterative one and a batch one. The two algorithms will be discussed in detail in section 3 and section 4. Here the performance of the code on the four given examples is presented first:

	alloy	example	formulation	mixture
Execution$_time\_itr, sec$	3.8678	0.9308	176.0281	0.3884
Execution$_time\_batch, sec$	1.0599	0.2485	0.5100	0.0958

As a reference for the distribution of the high dimensional data points, a distance $d = data[i] - < data >$ is calculated within each sampled data set. $< data >$ is the average value of data points. The distribution of d within results for the four given examples are shown in Fig 1 to Fig 4

4 Algorithm Design, Iterative Method

The classic algorithm for solving this kind of sampling problem is the Markov-chain Monte Carlo algorithm. If the feasible region is known, then one can discretize the space and perform random walk random walk on the discrete lattice [1, 2]. The difficulty of our problem comes from the non-linear constraints.

The key ideas in my algorithm design includes:

- Reorganize constraints into the order of involved variables so that constraints of a given variable can be easily checked. See `Constraint.get_var_constraints()` for details.
- Reduce the feasible region by checking linear constraints. The feasible region is recorded by a list called `bounds`, whose size is [n_dim, 2]. The first column of `bounds` records the low limits and the second column records the high limits. `bounds` is checked in two places.

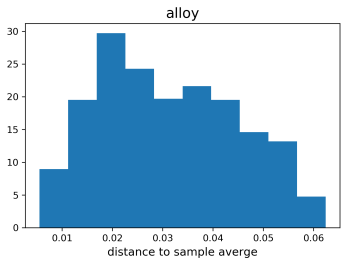

Figure 1: Distribution of d, alloy.txt

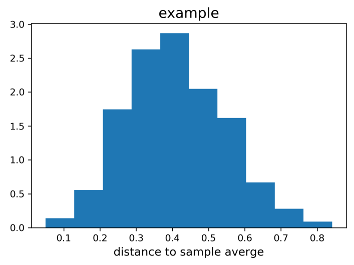

Figure 2: Distribution of d, example.txt

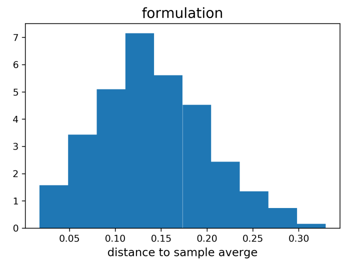

Figure 3: Distribution of d, formulation.txt

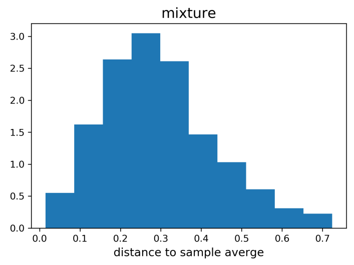

Figure 4: Distribution of d, mixture.txt

The first place is before sampling. The purpose is to take care of the single variable linear constraints. The second place is during sampling. The purpose is to check the multiple variable linear constraints if all variables in that constraint has been sampled. `bounds` is updated accordingly if linear constraint is passed.

- Group variables that share constraints and sample data from group to group. Note not all variables share constraints. In other words, variables that don't share constraints are independent of each other. We don't need to sample independent variables at the same time.
- For the schedule of sampling, a randomly picked variable is sampled from its feasible region time. In expectation, this strategy should be able to uniformly span the feasible region since the variables are independent and identically distributed. After a group of samples have all been sampled, check the constraints. If a constraint was violated, re-sample all variables appeared in that constraint. If all constraint satisfied, accept the data group and continue sample the next group until all data groups have been sampled.

Other possible improvements:

- Check of non-linear constraints. In current design, I only check non-linear constraints are the entire data tuple has been sampled. An alternative strategy is to check validity of a variables non-linear constraints after the sampling of this variable. This can be done by using `scipy.optimize.minimize` if the function value is bounded. However, I ended up not using it because of the many division operators in the expressions. Note if the first sampled variable sits in the denominator, the function value will be unbounded and `scipy.optimize.minimize` will fail. The checking of expression structure is not trivial and prone to bugs. As a result, I gave up checking non-linear constraints each time a variable is sampled.
- Schedule of re-sampling process. In current design, I started re-sampling several variables immediately after a constraint is violated. However, re-sample the last sampled data several times and check the violated constraint before start over is probably more efficient. This concept comes from the ideal of epsilon greedy algorithm [3].
- I also found a few papers on the topic of sampling data with non-linear constraints. However, wasn't able to digest them due to the time limit [4].

5 Algorithm Design, Batch

The batch algorithm was inspired from the iterative algorithm. The key ideas are:

- It's difficult check the validity of non-linear constraints before an entire data group has been sampled.
- The execution of for loops are slow.

As a result, we can just sample many data groups at one time and keep the valid ones. Moreover, there is actually no need to differentiate independent data groups and one can just sample many data tuples at one time. The simple one variable constraints were still checked before sampling data.
In the implementation, 1e6 data tuples were sampled each time. If the number of good data tuples are less than 1000, sample again until 1000 good data samples have been generated. It can be seen that this method works much better than the iterative method.
Note that the execution time of this method is almost linear with respect to the variable dimension. The performance on alloy.txt can actually be much better. I kind of arbitrarily chose to sample 1e6 data points each time. In the case of alloy.txt, more than 9e5 good data points were generated though I only kept 1000. Similar situation also applies to example.txt and mixture.txt.

References

[1] Luis Rademacher. Algorithmic convex geometry.

[2] Karel Van den Meersche, Karline Soetaert, and Dick Van Oevelen. xsample (): an r function for sampling linear inverse problems. *Journal of Statistical Software*, 30(Code Snippet 1), 2009.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[4] John W Chinneck. The constraint consensus method for finding approximately feasible points in nonlinear programs. *INFORMS Journal on Computing*, 16(3):255–265, 2004.