CP3 Report

Collaboration Statement:

Youtube and Textbook!

Total hours spent: 20

Links: [CP3 instructions] [Course collaboration policy]

Contents

1a

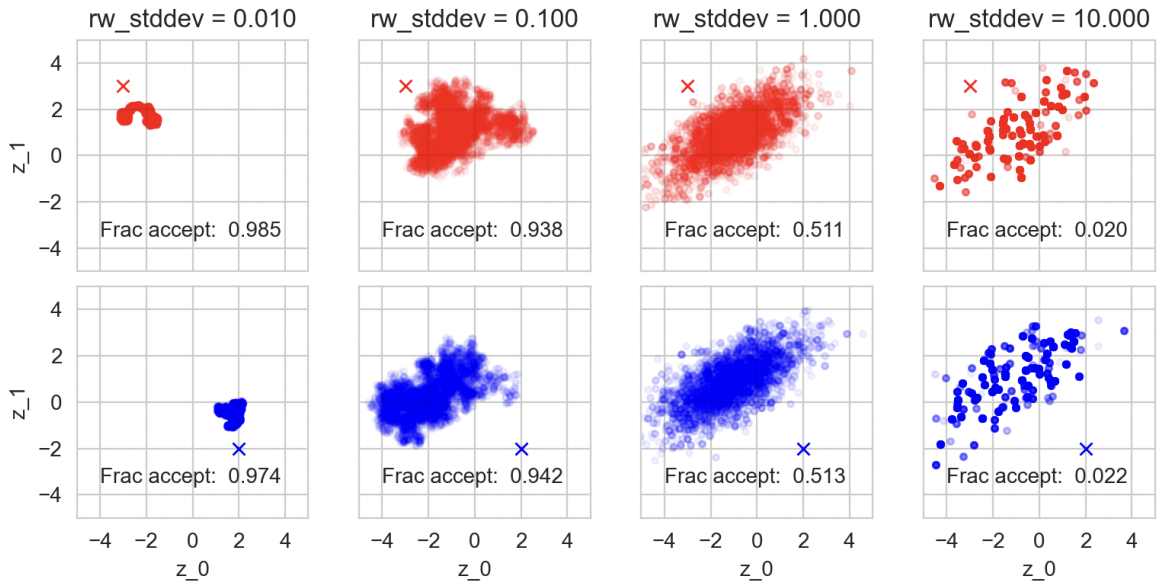

Fig. 1a: Rando Sample across a range of rw$_stddevof 0.01, 0.1, 1.0, 10.0$

1b

Suppose for a different problem (a different target distribution) you implement a RandomWalk sampler. You find that setting rw_stddev=100.0 yields an acceptance rate of around 0.8 after running 1 chain for 10,000 samples. Should you be confident your MCMC chain has converged? Justify your answer.

Even if the RandomWalk sampler that has $rw_stddev = 100.0$ and has an acceptance rate of around 0.8, we can't confirm that our MCMC will converge. First, despite 0.8 being a good acceptance rate, it might still lead to an inefficient sampling. This might happen if it is taking too small steps, which is why we might need to consider other factors like the scale of the target distribution. Second, it is possible that 10,000 sample size is insufficient for this specific problem. Although 10,000 seems like a decent amount, the convergence might not be achieved for high-dimensional spaces and complex distributions.

2a

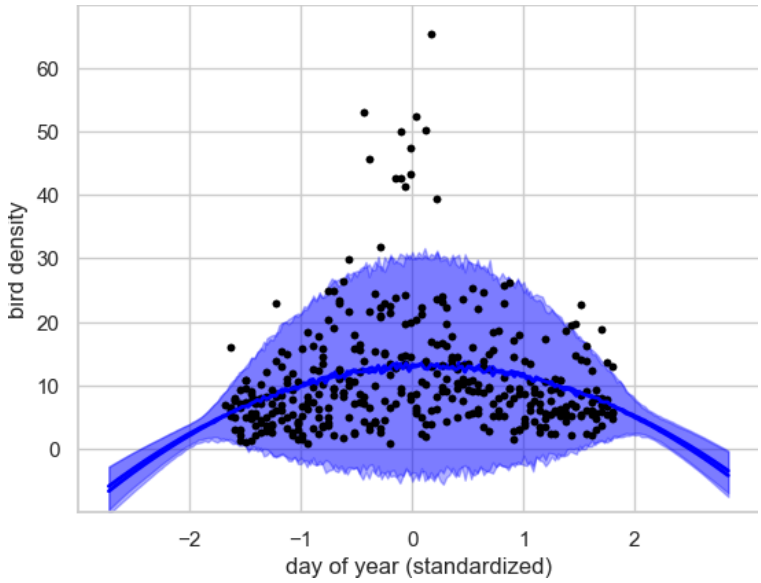

Fig. 2a: Posterior predictive visualization for order-2 polynomial model.

2b

Report the per-example score of models with order 0 and order 2 on the provided test set.

order - 0 Test Log Probability Score: -4.054159

order - 2 Test Log Probability Score: -4.062983

2c

Provide your implementation of calc_score as a Python function in your report.

```
def calc_score(list_of_z_D, phi_RM, t_R):
    score_aggr = np.zeros((len(list_of_z_D), len(t_R)))
    for i in range(S):
        # Compute score formula for ss-th sample
        # Hint: Use unpack_mean_N_and_stddev_N
        mean_R, stddev_R =
        unpack_mean_N_and_stddev_N(list_of_z_D[i]
        , phi_RM)score_formula_per_sample = scipy.stats.
        norm.logpdf(t_R, loc=mean_R, scale=stddev_R)
        score_aggr[i] = score_formula_per_sample

    # aggregate across all S samples
    # Hint: use scipy.special.logsumexp to be
    numerically stable
    final_score = np.mean(scipy.special.logsumexp( ,
        axis=0) - np.log(S))
    return final_score
```