CS136

CP3 Pengcheng Xu

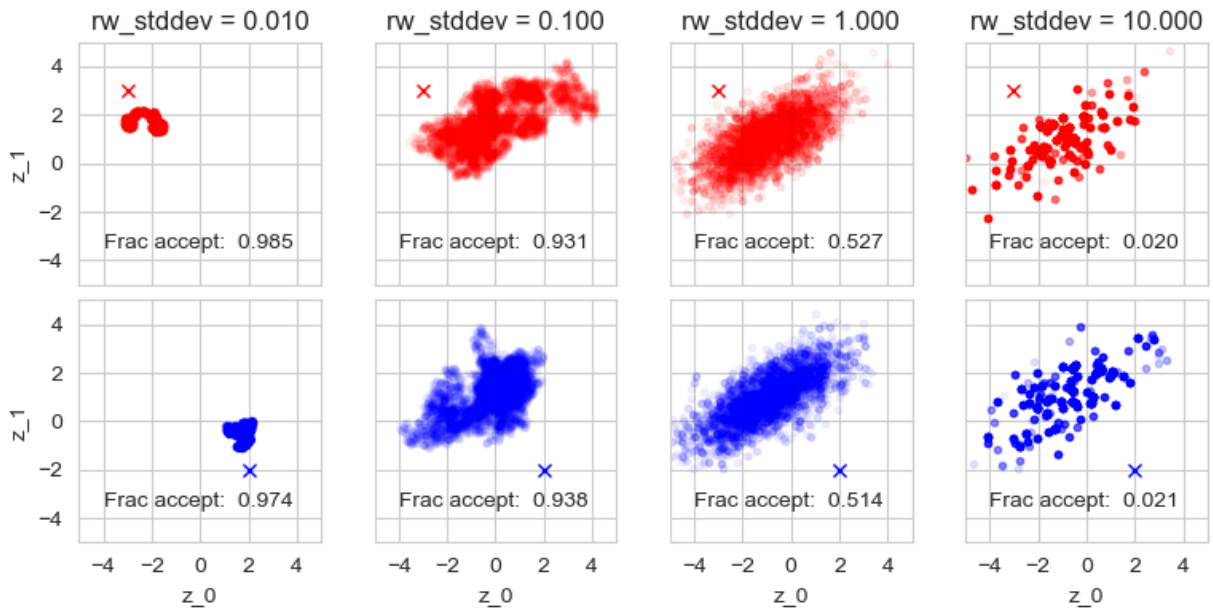
Collaboration Statement:

- Total hours spent: 8 hrs
- Consult Resources:
    - Course's website
    - Numpy website
    - Online resource
- Internet

Problem 1.

1a

After implementing `calc_tart_log_pdf` and `main` block in `run_RW_prob1.py`, we get the following figure:



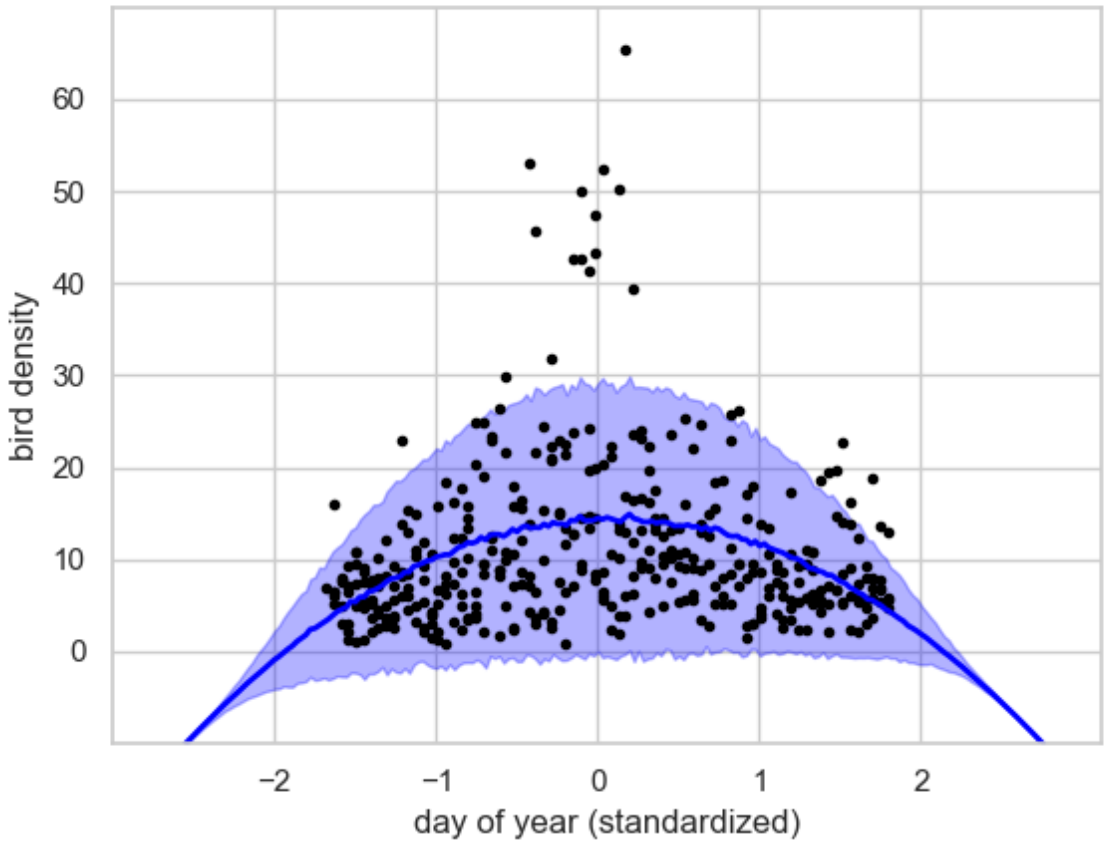
1b

Given the info, I'm not very confident that our MCMC chain has converged to the target distribution.
The reason is that the accept rate is a little too high (i.e. around 0.8 after 10,000 iterations). The situation is similiar to that in **1a** where the rw*stddev = 0.010. Even though in the current setting, the rw*stddev = 100.0, this value could be still too small compared to the target distribution's standard deviation (e.g. what if our target std is 100,000). In other words, the difference (i.e. randomwalk step) between our old Sample (i.e. z_old) and new Sample (i.e. z_new) is too small, so we're sort of stuck in the only small portion of the target distribution (and get a high accept rate), but cann't see the whole picuture just like what we did in **figure 1a** where rw_stddev = 0.010.

Problem 2.

2a

The posterior predictive visualization for order-2 polynomial model:



2b

The following is the table of per-example score of models with order 0 and order 2 on the provided test set.

order	test score
0	-4.533
2	-4.220

2c

The following is the screenshot of my implementation of `calc_score` function

```python
def calc_score(list_of_z_D, phi_RM, t_R):
    ''' Calculate per-example score averaged over provided test set of size R

    Args
    ----
    list_of_z_D : list of ndarray
        List of samples of parameters, assumed to be from target posterior
    phi_RM : 2D array, shape (R, M)
        Feature vectors for each of the examples in test set of size R
    t_R : 1D array, shape (R,)
        Output values for each of the examples in test set of size R

    Returns
    -------
    score : float
        Per-example log pdf of all t values in test set
        using Monte-Carlo approximation to marginal likelihood
    '''
    S = len(list_of_z_D)
    res = []
    for ss in range(S):
        z_ss_D = list_of_z_D[ss]
        # Compute score formula for ss-th sample (see instructions)
        # Hint: Use unpack_mean_N_and_stddev_N
        mean_R, stddev_R = unpack_mean_N_and_stddev_N(z_D = z_ss_D, phi_NM = phi_RM)
        res.append(norm.logpdf(t_R, mean_R, stddev_R))
    # TODO aggregate across all S samples
    # Hint: use scipy.special.logsumexp to be numerically stable
    res = np.vstack(res)
    res = scipy.special.logsumexp(res, axis = 0) - np.log(S)
    return np.average(res)
```