CP3: Implementing Markov Chain Monte Carlo

Last modified: 2023-03-10 22:54

Status: **RELEASED. All problems ready.**

What to turn in:

- ZIP file of source code: https://www.gradescope.com/courses/496674/assignments/2736201/
- PDF report file: https://www.gradescope.com/courses/496674/assignments/2736216/

Your ZIP file should include

- All starter code .py files (with your edits) (in the top-level directory)
- These will be *auto-graded*. Be sure to check gradescope test results for immediate feedback.

Your PDF should include (in order):

- Your full name
- Collaboration statement
- Problem 1: Figure 1a and Answer 1b
- Problem 2 figure and short answers

Questions?: Post to the cp3 topic on the discussion forums.

Jump to: Problem 1   Problem 2   Background on Prob. 2 Data   Starter Code & Data

Overview

In this coding practical, we will implement the Metropolis MCMC algorithm with Random Walk proposals

The problems will try to address key practical questions:

- How can we use MCMC methods to sample from a desired target distribution?
- How do we use MCMC methods in practice? (e.g. select hyperparameters of proposal distribution, determine number of samples, assess burn-in, etc.)
- How can we use parameter samples to make predictions of future data?

Background on the Metropolis Algorithm with Random Walk proposals

For helpful background, you can review:

- the lecture notes from day 12
- highlights from day 12
- Bishop's PRML textbook Sec. 11.2

Data and Starter Code

You can find the starter code and our cleaned data in the course Github repository here:

https://github.com/tufts-ml-courses/cs136-23s-assignments/tree/main/unit3_CP/

Problem 1: Random Walk Algorithm Implementation

Target Distribution

You goal in Problem 1 is to draw samples from this *target distribution*:

$$p(z) = p([z_0, z_1]) = \text{MultivariateNormal}\left(\begin{bmatrix} -1.0 \\ +1.0 \end{bmatrix}, \begin{bmatrix} 2.00 & 0.95 \\ 0.95 & 1.0 \end{bmatrix} \right)$$

You want to write a Random Walk sampler to draw a Markov chain of samples $z^1, z^2, \dots z^S$.

Remember each *sample* is a 2-dim. vector: $z^s = [z_0^s \; z_1^s]$

Tasks for Code Implementation

(All code steps will be evaluated primarily by Autograder, but may also be read thru by Instructor/TA).

CODE 1(i) Implement the `calc_target_log_pdf` method in `run_RW_prob1.py`, to calculate the log PDF of our target distribution.

CODE 1(ii) Implement the `draw_samples` method of `RandomWalkSampler` class. This is a generic procedure that will implement the Metropolis algorithm correctly (draw proposal from Normal, then accept/reject using Metropolis accept threshold), using an externally defined *function* that computes the target log pdf (or its value up to an additive constant).

Tasks for Data Analysis

Now implement the `main` block of `run_RW_prob1.py` in order to solve the analysis questions below.

ANALYSIS 1(i): Using the `main` block of starter code `run_RW_prob1.py`, run your RandomWalk sampler from the first initialization (labeled "A" in the code). Set the proposal standard deviation to `rw_stddev=10.0`. Run for 5000 iterations of "burnin", then gather 5000 samples.

- Plot the 2D distribution of *kept* samples (after burnin phase). Does it qualitatively converge to the target distribution? How would you know?
- Look at the acceptance rate. Does this give you a clue about what might be going on?

ANALYSIS 1(ii): Repeat with the second initialization (labeled B) in the provided code, again set the proposal standard deviation to `rw_stddev=10.0`.

- Plot the 2D distribution of *kept* samples. Does it qualitatively converge to the intended distribution? *NB: If two chains converge to the same distribution, these plots should look indistinguishable.*

ANALYSIS 1(iii): Repeat the above across a range of `rw_stddev` hyperparameter values: [0.01, 0.1, 1.0, 10.0]. Make sure you understand (1) what acceptance rates and other metrics look like when this hyperparameter is too small, too big, and about right; and (2) how to use multiple chains from different initialization to sanity check convergence.

Tasks for Report Writing

1a FIGURE: Scatterplot of samples across hyperparameters & inits

Use the 2 row x 4 column plot arrangement provided by starter code

- In each column, show plot for one `rw_stddev` value in `[0.01, 0.1, 1.0, 10.0]`

- In row 1, show the results from initialization A. In row 2, use initialization B.

Please *keep plot styling and limits unchanged*.

1b SHORT ANSWER: 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.

Background for Prob. 2: Birds on Spring Break

Here in problem 2, you will analyze a dataset of per-day measurements of bird density observed by radar stations observing one location in the Netherlands, gathered each spring over the years 2010 to 2018.

This leads to a probabilistic regression problem: next year, what will be the peak bird density t be on day x? Accurate forecasting could be useful for ecologists as well as for air traffic control and wind farm management.

This data comes from the academic study *Ensemble predictions are essential for accurate bird migration forecasts for conservation and flight safety* by Kranstauber et al.. Original raw dataset is the "spring" data available at this link.

Your course staff cleaned this data by:

- focusing on 2010 - 2018 [2012-2013 unavailable in raw data]
- focusing only on the days in Spring (15 Feb - 30 April)
- keeping the "peak" (95th percentile) measurement observed in any 5-min interval throughout the day

Here's a human-readable sample of the training set CSV file:

```
year  days_since_0401  bird_density
2010  -44               3.09521
2010  -43               1.24416
2010  -42               4.87082
...
2017   27               3.62189
2017   28              13.59403
2017   29              13.04274
```

Column definitions:

- days_since_0401 : number of days from April 1st (e.g. so -11 means March 20)
- bird_density : peak density measured by radar (positive number, higher means more birds)

Background for Prob. 2: Probabilistic Model for Heteroskedastic Regression

Consider the regression problem of predicting bird density measurements $(t_1, \dots t_N)$ given known day-of-year values $(x_1, \dots x_N)$.

Unlike past attempts at regression like CP2, where we modeled the mean but kept the variance fixed, in this problem we treat **both mean and variance of t as learnable functions** that can vary with x. MCMC makes this possible: we don't need to know the closed-form of any posterior of interest in order to do useful estimation and prediction!

Our model consists of the following random variables

- $t_1, \dots t_N$: (observable) bird density at each day of interest in dataset of size N
- w : (hidden) weight vector that determines the mean function for predicting t from x
- v : (hidden) weight vector that determines the std. dev. for predicting t from x

We assume that each t_n r.v. comes with a known, fixed day-of-the-year feature x_n.

Likelihood

Our likelihood of the observable r.v. $t_{1:N}$ is defined as:

$$\begin{aligned} p(t_{1:N}|w, v) &= \prod_{n=1}^{N} p(t_n|w, v) \\ &= \prod_{n=1}^{N} \text{NormPDF}(t_n|w^T\phi(x_n), s(v^T\phi(x_n))^2) \end{aligned}$$

Let $s(a) = \log(1 + e^a)$ denote the softplus function, which transforms any scalar to a positive scalar. This function ensures that the standard deviation above is always positive.

Just like in CP2, $\phi(x_n)$ is a polynomial feature function. Because x variables are assumed known and fixed (not treated as random variables), we don't include x variables in the probabilistic notation $p(\cdot|\cdot)$.

We'll assume an indexing of polynomial-expanded features where:

- index 1 corresponds to order 0 (bias term)
- index 2 corresponds to order 1 (linear term x_n)
- index 3 corresponds to order 2 (quadratic term x_n^2)
- etc.

Prior on weight parameters

This model has two hidden parameters (not observable) that we'll model as random variables:

- weight vector $w \in \mathbb{R}^M$ determines the mean prediction
- weight vector $v \in \mathbb{R}^M$ determines the standard deviation

Let's assume the two weight vectors, w and v, have independent priors:

$$p(w_{1:M}) = \text{NormPDF}(w_1|10, a^2) \prod_{m=2}^{M} \text{NormPDF}(w_m|0.0, a^2)$$

$$p(v_{1:M}) = \text{NormPDF}(v_1|10, a^2) \prod_{m=2}^{M} \text{NormPDF}(v_m|0.0, a^2)$$

where we give a special larger mean to the weights for the bias term in index 1.

Let $a > 0$ is a fixed constant that controls the standard deviation of these priors. To make our priors rather weakly informative (flexible), we'll set $a = 10$.

Packing/Unpacking

When convenient, we will use notation $z \in \mathbb{R}^D$ to denote a single vector that packs all parameters together. For example, our implementation of RandomWalkSampler assumes all random variables belong to a single vector.

Throughout problem 2, we'll assume the following packing order:

$$\begin{aligned} z_1 &= w_1 \\ z_2 &= w_2 \\ &\dots \\ z_M &= w_M \\ z_{M+1} &= v_1 \\ &\dots \\ z_{2M} &= v_M \end{aligned}$$

Posterior

We'd like to get samples from the posterior of w, v (all hidden random variables) given the observed training set.

By Bayes rule, we can write the PDF of the posterior as:

$$\begin{aligned} p(w, v|t_{1:N}) &= \frac{1}{p(t_1, \dots t_N)} p(w, v) \prod_{n=1}^{N} p(t_n|w, v) \\ &= c \cdot \underbrace{p(w, v) \prod_{n=1}^{N} p(t_n|w, v)}_{\tilde{p}(z)} \end{aligned}$$

where we've gathered all terms that do not depend on w, v into a scalar constant $c > 0$.

Naturally, we can compute $\tilde{p}(z)$, also known as the *joint PDF* of all hidden and observed random variables, just by using our defined likelihood and prior PDF functions above.

From posterior to prediction

Ultimately, we'd like to make predictions for future bird traffic given our training set.

Consider trying to model a single test point t^* after seeing the train set. *Note: We'll use superscript * to indicate test rather than train data.* We can make the usual conditional i.i.d. assumption (given w, v, each t is independent of all others).

Thus, we write the posterior predictive of t_* given the train set as:

$$\begin{aligned} p(t^*|t_{1:N}) &= \int_v \int_w p(t^*, w, v|t_{1:N}) dw dv \\ &= \int_v \int_w p(t^*|w, v) p(w, v|t_{1:N}) dw dv \end{aligned}$$

Now if we have S samples from our posterior (via our MCMC routine), we can approximate this difficult integral via a Monte Carlo estimate:

$$p(t^*|t_{1:N}) \approx \frac{1}{S} \sum_{s=1}^{S} p(t^*|w^s, v^s), \qquad w^s, v^s \sim p(w, v|t_{1:N})$$

Transforming to work in log-space for numerical stability, this becomes

$$\log p(t^*|t_{1:N}) \approx \log \sum_{s=1}^{S} \exp\left[\log p(t^*|w^s, v^s)\right] - \log S$$

Therefore, we'll compute the per-example **score** for an entire test set as:

$$
\begin{aligned}
score(t^*_{1:R}) &= \frac{1}{R}\sum_{r=1}^{R} \log p(t^*_r | t_{1:N}) \\
&\approx \frac{1}{R}\sum_{r=1}^{R} \left[\log \sum_{s=1}^{S} \exp\left[\log p(t^*_r | w^s, v^s)\right] - \log S \right]
\end{aligned}
$$

We can assess this score on a test dataset of size R to understand how well our model is fitting new data drawn from the same distribution. Calculating this score requires a set of S posterior samples from MCMC.

You'll implement this evaluation strategy in a function called `calc_score`.

Problem 2: Predicting Bird Density over Time

We will now use the Random Walk sampler to do Bayesian data analysis of the bird density data.

Your goal: Build the best possible probabilistic predictions of year 2018 given the training set from 2010-2017.

Tasks for Code Implementation

CODE 2(i) Implement the `calc_joint_log_pdf` method in `run_RW_prob2.py`, to calculate the log PDF of $\log p(t_{1:N}, w, v)$. Note that this can be done by summing up the log prior and the log likelihood.

CODE 2(ii) Implement the `calc_score` method of `run_RW_prob2`. This function reports the per-example log probability density of the heldout test set, using the Monte Carlo approximation above.

Tasks for Data Analysis

Now implement the `main` block of `run_RW_prob2.py` in order to solve the analysis questions below.

ANALYSIS 2(i): Using the `main` block of starter code `run_RW_prob2.py`, fit an order=0 model to the provided training set. For each dimension of the sampled parameter vector (which concatenates w and v), find a setting of the `rw_stddev` hyperparameter such that two separate long MCMC chains (with at least 2000 kept samples) appear to reasonably converge. This means that, after burnin, both trace plots look similar and predictive distributions appear similar.

ANALYSIS 2(ii): Repeat the above for an order=2 model. This may be slightly tougher to get to converge (now you have 6 dimensions total, not 2), but the principle remains the same.

Tasks for Report

2a FIGURE: Posterior predictive visualization for order-2 polynomial model

For your best-looking chain for order-2, use the provided code that can visualize the posterior predictive $p(t_* | t_{1:N})$ given your list of samples of w, v.

The created plot will show a solid line for the mean estimate of t_*, and produce a shaded region between the 5th and 95th percentiles.

You should see a noticeably better fit to the data than the order 0 model.

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

Use a simple human-readable tabular format like this:

```
order   test score
    0       -1.111
    2       +4.444
```

2c Code Answer:

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