Vectorized Markov Chain Monte Carlo parameter estimation for the multispecies coalescent model for two species

An Xuelong

Disclaimer: the online-knowledge retrieval large language model perplexity https://www.perplexity.ai/ has aided me in writing code and summarizing relevant literature for writing the report.

Motivation

In comparative genomics, the multispecies coalescent (MSC) model is a powerful statistical framework proposed to address conflicting genealogical histories by accounting for incomplete lineage sorting (ILS) owed to polymorphism in ancestral species [1,2]. Its most important parameters are τ and θ, which represent the species divergence time and population size, respectively. Both are assumed to be constant across the genome. We can resort to exact likelihood methods such as Bayesian inference to estimate them and accommodating their uncertainties accordingly. However, such approaches are often intractable owed to the high dimensionality of the available genomic data. Thus, in this work, we resort to an approximate method by a Monte-Carlo Markov-Chain (MCMC) algorithm for tractably estimating the parameters of the MSC model [1, 3, 4] to help us understand the intermingled evolutionary history of humans and chimpanzees.

Methods

We collect genomic, multi-locus data consisting of sequence alignments of humans and chimpanzees $X = \{(x_1, n_1), \dots (x_L, n_L)\}$ for $L = 1000$ loci where for locus i we observe x_i differences at n_i sites [see 5]. These loci are loosely linked short genomic segments, such that we ignore recombination and treat each locus as independent. The posterior distribution f of the parameters of interest is, according to Bayes Theorem:$f(\tau, \theta, \{t_i\}|\, X) = \frac{1}{Z} f(X|\tau, \theta, \{t_i\}) f(\tau) f(\theta)$, where $f(X|\tau, \theta, \{t_i\}) = \prod_{\{i=1\}}^{1000} \binom{n_i}{x_i} p_i^{x_i} (1 - p_i)^{n_i - x_i}$ is the binomial likelihood of observing all differences at all sites, where $p_i = \frac{3}{4} - \frac{3}{4} e^{-\frac{8}{3}(\tau + t_i)}$ is the probability of observing a difference at any site at locus i and t_i is an exponential variable measuring the coalescent time at locus i with density $f(t_i|\tau, \theta) = \frac{2}{\theta} e^{-\frac{2}{\theta} t_i}$. The terms $f(\tau) = \frac{1}{\mu_\tau} e^{-\frac{1}{\mu_\tau}\tau}, f(\theta) = \frac{1}{\mu_\theta} e^{-\frac{1}{\mu_\theta}\theta}$ are exponential priors with $\mu_\tau = 0.005$, $\mu_\theta = 0.001$. $Z = \int f(X|\tau, \theta, \{t_i\}) f(\tau) f(\theta) d\tau, \theta$ is known as the *normalization constant*, and it is the main computational bottleneck for obtaining the posterior. Because it's a multidimensional integral, can't be derived analytically and expensive to tractably compute numerically, we resort to the classical Metropolis-Hasting algorithm [see 6] to draw probabilistic samples of $f(\tau, \theta, \{t_i\}|X)$[1] whilst bypassing the normalization constant by cancelling it out

[1] We are updating 1000 coalescent times for each locus in addition to τ and θ, however through

through the log ratio of the unnormalized posterior:

$$log\, f(\tau,\theta,\{t_i\}|X) = C - \frac{1}{\mu_t}\tau - \frac{1}{\mu_\theta}\theta + \sum_{i=1}^{1000}[log\frac{2}{\theta} - \frac{2}{\theta}t_i + x_i\, log\, p_i + (n_i - x_i)\, log(1-p_i)]$$

Equation 1: log unnormalized posterior (later simplified as $\pi(\phi)$), where C is a constant absorbing terms we ignore in the MCMC algorithm. Taking the log helps avoid numerical over/underflow during MCMC.

This MCMC algorithm, adapted for our purposes, is as follows:

1. Initialize parameters $\Phi = \{\tau = 0.01, \theta = 0.001, t_{1:1000} = 0.001\}$ and respective window sizes w_ϕ[2] (see Table 1 for candidate window sizes)
2. For each MCMC iteration, and for each parameter ϕ in Φ:
   a) Sample ϕ^* from a uniform proposal distribution $U\left(\phi - \frac{w_\phi}{2}, \phi + \frac{w_\phi}{2}\right)$, where if $\phi < 0, \phi = -\phi$
   b) Compute $\alpha = \min\left(1, \frac{\pi(\phi^*)}{\pi(\phi)} \times \frac{U(\phi|\phi^*)}{U(\phi^*|\phi)}\right)$, where $\pi(\phi^*)$ is the unnormalized posterior (Equation 1), thus the ratio effectively cancels out the normalization constant. The log ratio $\frac{\pi(\phi^*)}{\pi(\phi)}$ is computed by keeping the remaining parameters in Φ fixed to the previous state, thus treating this multidimensional MCMC into separate unidimensional updates. As such, we adjust window sizes for an acceptance rate of 43% for each ϕ.
   c) Accept the proposed sample if $u < \alpha, u \sim U(0,1)$, otherwise reject.

We optimize runtime of the algorithm as follows: 1) we vectorize the computation of $\pi(\phi)$ for summing out the coalescent times, 2) we also vectorize the proposal and subsequent acceptance/rejection of the 1000 coalescent times, noting that their logratios are directly calculated as $-\frac{2}{\theta}(\boldsymbol{t}^* - \boldsymbol{t}) + \boldsymbol{x} * \log\left(\frac{\boldsymbol{p}^*}{\boldsymbol{p}}\right) + (\boldsymbol{n} - \boldsymbol{x})\log\frac{1-\boldsymbol{p}^*}{1-\boldsymbol{p}}$ where in bold we highlight the vectorization and 3) we cache the current $\pi(\phi^*)$ to avoid duplicate computations for α. Altogether, our MCMC implementation finishes under ~ 30 seconds for $L = 1000$ loci and 20000 iterations. For implementation details see Appendix 0.

Step 2b) is done by exploring over a grid of window sizes for τ and θ, with τ: `[5.3e-06, 0.001, 0.053]` and θ`: [9.e-06, 0.001, 0.090]`, to study how it influences acceptance rate, efficiency and final posterior sample values.

Results

The results of our experiments are recorded in Table 1, where we achieve an acceptance rate of

marginalisation we ignore $t_{1:1000}$

[2] We explore different window sizes in the **Results** section to achieve an acceptance rate of $\sim 43\%$ for each ϕ

43% for both τ and θ through the same window size of 0.001. For τ, we obtain a posterior mean of 0.004012 with 2.5%, 97.5% credibility interval of $[0.003728, 0.004294]$. For θ, we obtain a posterior mean of 0.003984 and credibility interval of $[0.003416, 0.004609]$.

τ window	**θ window**	**Acceptance rate (τ,θ)**	**Efficiency (τ,θ)**	**Posterior mean with credibility interval**
5.3e-06	0.001	0.89, 0.021	0.059, 0.059	τ:0.0070, [0.006164, 0.008422] θ:0.0039,[0.003416, 0.004609]
0.001	9.e-06	0.44, 0.97	0.066, 0.059	τ:0.0049, [0.004494, 0.005270] θ:0.002, [0.001415, 0.002608]
0.001	**0.001**	**0.43, 0.43**	**0.08, 0.07**	**τ:0.0040, [0.003728, 0.004294]** **θ:0.0039,[0.003416, 0.004609]**
0.053	0.001	0.010, 0.43	0.059, 0.070	τ:0.0040, [0.003684, 0.004370] θ:0.0040,[0.003287, 0.004624]

Table 1: Effect of varying window sizes on MCMC posterior samples, acceptance rates and efficiencies. Highlighted are the window sizes for which we manage to obtain the desired acceptance rate for both parameters.

In Figure 1, the trace plots of the posterior samples after a burn-in of 5000 also show a stable convergence for both parameters. Both parameters have an efficiency measured via a ratio of the variance based on the independent sample to the variance based on the MCMC sample of ~ 0.075, yielding a reasonable[3] effective sample size (ESS) of around $20000 * .075 \cong 1500$. This means a sample of size 20000 from the MCMC is as good (in terms of variance) as an independent sample of size 1500.

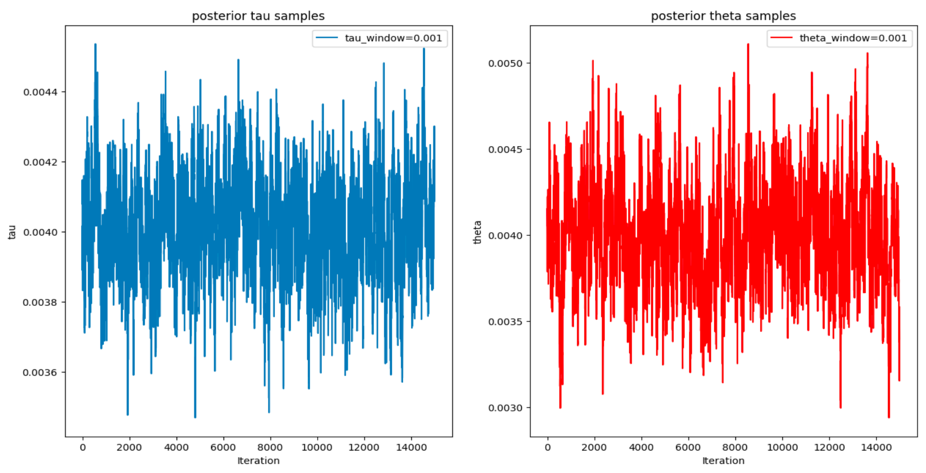

Figure 1: Trace plot for MCMC posterior τ and θ samples after burn-in.

Our experiments also shed insight how varying window sizes (at the log-scale) deeply influences the quality of the samples in the MCMC (see Appendix-Figure 2 for an example of an inefficient MCMC). Both very high and low window sizes lead to strongly autocorrelated, inefficient MCMCs. Although we treat parameters as separate unidimensional candidates, a bad window size for one

[3] An ESS greater than 1000 as per [7]

parameter may sometimes negatively impact the sampling quality of the other parameter (see window sizes for τ: `5.3e-06` and θ: `0.001`). This behavior can be explained by the computation of $\pi(\phi)$ which jointly depends on the current states of both parameters. In general, the choices of window sizes are performed through trial and error, as they depend on the characteristics of the specific problem, as well as the desired trade-off between acceptance rate, convergence time, and efficiency.

We also note a limitation of our work, which is that estimates are first affected by the assumptions of the MSC model, as well as the available data. We only worked with 1000 loci, while it's well established that more data yield more reliable estimates [3]. A preliminary run of our MCMC algorithm (using the same hyperparameters above: window size 0.001, 20000 iterations and 5000 burn-in) on all 14663 loci yield for τ a posterior mean of 0.004174 with credibility interval of $[0.004090, 0.004259]$. For θ, we obtain a posterior mean of 0.004370 and credibility interval of $[0.004203, 0.004551]$.

Conclusion

In this work, we use Bayesian MCMC to tractably estimate the parameters of a MSC model analyzing genomic data of humans and chimpanzees.

References

1. Xu, B., & Yang, Z. (2016). Challenges in Species Tree Estimation Under the Multispecies Coalescent Model. *Genetics*, *204*(4), 1353–1368. https://doi.org/10.1534/genetics.116.190173
2. Degnan, J. H., & Rosenberg, N. A. (2009). Gene tree discordance, phylogenetic inference and the multispecies coalescent. *Trends in Ecology & Evolution*, *24*(6), 332–340. https://doi.org/10.1016/j.tree.2009.01.009
3. Rannala, B., & Yang, Z. (2003). Bayes Estimation of Species Divergence Times and Ancestral Population Sizes Using DNA Sequences From Multiple Loci. *Genetics*, *164*(4), 1645–1656. https://doi.org/10.1093/genetics/164.4.1645
4. Flouri, T., Jiao, X., Huang, J., Rannala, B., & Yang, Z. (2023). Efficient Bayesian inference under the multispecies coalescent with migration. *Proceedings of the National Academy of Sciences of the United States of America*, *120*(44). https://doi.org/10.1073/pnas.2310708120
5. Burgess, R., & Yang, Z. (2008). Estimation of Hominoid Ancestral Population Sizes under Bayesian Coalescent Models Incorporating Mutation Rate Variation and Sequencing Errors. *Molecular Biology and Evolution*, *25*(9), 1979–1994. https://doi.org/10.1093/molbev/msn148
6. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. https://doi.org/10.1063/1.1699114
7. Yang, Z., & Flouri, T. (2024). *Bayesian Markov chain Monte Carlo (MCMC) in population genetics*. Moodle's Lecture Slides for the Course Advanced Computational Biology.

Appendix

Code for running MCMC

```
SLICE = 1000
df = pd.read_csv('data/HC.SitesDiffs.txt', delimiter='\t')
df = df[:SLICE]

def logpriorlikelihood(tau, theta, coalescent_times, df = df, tau_prior =
5e-3, theta_prior = 1e-3):
    '''
    main function to give feedback to random walk (vectorized implementation
to speed up computation)
    Args:
      df: pd.DataFrame - dataset containing the number of differences (xi)
at ni sites at locus i
      tau: float - Speciation time parameter
      theta: float - Population size parameter
      coalescent_times: np.array - coalescent time of locus i, for L=1000
loci
      tau_prior: float - prior parameter of tau, by default it's 0.005,
    theta_prior: float - prior parameter of theta, by default it's  1e-3

    Returns:
      unnormalized posterior given input parameters
    '''
    # Assuming df contains 1000 rows with two columns n and x
    n: ndarray = df.iloc[:, 0].values
    x: ndarray = df.iloc[:, 1].values

    # Calculate p using vectorized operations
     # t: ndarray = coalescent_times.reshape(-1, 1) # perplexity's Pro's
answer
    p: ndarray = 3/4 - 3/4 * np.exp(-8/3 * (tau + coalescent_times))

    # # Sum up the values along the first axis to get the total sum
     sum_coalescent_times: float = np.sum(np.log((2/(theta))) - 2/theta *
coalescent_times + \
                                         x * np.log(p) + (n - x) * np.log((1
- p)))

     unnormalized_posterior: float = -1/tau_prior * tau - 1/theta_prior *
theta + sum_coalescent_times
```

```
    return unnormalized_posterior

def lnratio_tj_vectorized(tau, theta, proposed_tjs, tjs, df = df):
    '''
    Vectorized helper function during MCMC to simplify log ratio computation
for changing coalescent times.
    Args:
      tau: Current value of tau.
      theta: Current value of theta.
      proposed_tjs: Array of proposed coalescent times.
      tjs: Array of current coalescent times.
      df: DataFrame containing the number of differences (xi) at ni sites at
locus i.
    Returns:
      Array of log ratios for each proposed coalescent time change.
    '''

    p_stars = 3/4 - 3/4 * np.exp(-8/3 * (tau + proposed_tjs))
    ps = 3/4 - 3/4 * np.exp(-8/3 * (tau + tjs))

    njs, xjs = df.iloc[:, 0].values, df.iloc[:, 1].values

    log_ratios = -2/theta * (proposed_tjs - tjs) + xjs * np.log((p_stars) /
(ps)) + \
                 (njs - xjs) * np.log((1 - p_stars) / (1 - ps))

    return log_ratios


### INIT PARAMETERS
w_tau = 0.00060
w_theta =  0.0009
w_t = 0.019

tau = 0.01
theta = 0.001
coalescent_times  =  np.array([0.001  for  _  in  range(df.shape[0])])  #  1000
coalescent times

init_windows = (w_tau, w_theta, w_t)
init_params = (tau, theta, coalescent_times)

# long run
BURNIN = 5000
```

```
SAMPLES = 20000
###
def  mcmc(steps  =  SAMPLES,  init_windows  =  init_windows,  init_params  =
init_params):
  '''
  window sizes (w_tau = 0.01, w_theta = 0.002, w_t = 0.002)
  initial parameters tau = 0.001, theta = 0.001, and ti = 0.001, for i = 1, …,
L
  '''

  w_tau, w_theta, w_t = init_windows

  tau, theta, coalescent_times = init_params

  sample_tau = []
  sample_theta = []
  sample_coalescent_times = np.zeros(shape=(len(coalescent_times), steps))

  accepted_tau = accepted_theta = 0
  accepted_times = np.zeros_like(coalescent_times) # log acceptance rate per
each coalescent time of a locus

   lnp  =  logpriorlikelihood(tau  =  tau,  theta  =  theta,  coalescent_times  =
coalescent_times)

  for mcmc_iteration in range(steps):

    ### CHANGE TAU with theta and coalescent times fixed
    proposed_tau = tau + (np.random.uniform(0, 1) - 0.5) * w_tau
    if proposed_tau < 0:
      proposed_tau = -proposed_tau

     #  calculate  unnormalized  posterior  for  proposed  theta,  and  posterior
ratio
      lnpnew   =   logpriorlikelihood(tau   =   proposed_tau,   theta   =   theta,
coalescent_times=coalescent_times)
    logratio = lnpnew - lnp  # reuse lnp instead of computing it again

    if logratio >= 0 or np.random.uniform() < math.exp(logratio):
      # we accept the proposal
      tau = proposed_tau
      lnp = lnpnew
      accepted_tau += 1
    # we log a sample regardless whether we accepted the proposal or not
    sample_tau.append(tau)

    ### CHANGE THETA with tau and coalescent times fixed
    # set a different window size to change acceptance rate
    proposed_theta = np.random.uniform(theta - w_theta/2, theta + w_theta/2)

    if proposed_theta < 0:
      proposed_theta = -proposed_theta

    # calculate unnormalized posterior for proposed theta, and posterior
ratio
    lnpnew = logpriorlikelihood(tau = tau, theta = proposed_theta,
coalescent_times = coalescent_times)
    logratio = lnpnew - lnp # reuse lnp instead of computing it again

    if logratio >= 0 or np.random.uniform() < math.exp(logratio):
      # we accept the proposal
      theta = proposed_theta
      lnp = lnpnew
      accepted_theta += 1

    sample_theta.append(theta)

    # CHANGE COALESCENT times with tau and theta fixed
    proposed_coalescent_times = coalescent_times + (np.random.uniform(0, 1,
size=len(coalescent_times)) - 0.5) * w_t
    proposed_coalescent_times = np.where(proposed_coalescent_times < 0, -
proposed_coalescent_times, proposed_coalescent_times)
    # Vectorized implementation treats coalescent times as independent
    logratios = lnratio_tj_vectorized(tau = tau, theta = theta, proposed_tjs
= proposed_coalescent_times,\
                                      tjs = coalescent_times)

    # Calculate acceptance probabilities and determine which proposals to
accept
    acceptance_probs = np.exp(logratios)
            accepts = (logratios >= 0) |
(np.random.uniform(size=len(coalescent_times)) < acceptance_probs)

    # Update accepted coalescent times and log posterior
    accepted_indices = np.where(accepts)[0]
    for idx in accepted_indices:
        coalescent_times[idx] = proposed_coalescent_times[idx]
        lnp += logratios[idx]
        accepted_times[idx] += 1

    # Log the samples for this iteration
    sample_coalescent_times[:, mcmc_iteration] = coalescent_times

  # return the sample and the number of accepted proposals
  return sample_tau, sample_theta, sample_coalescent_times, accepted_tau,
accepted_theta, accepted_times


# Calling the MCMC
sample = SAMPLES
sample_tau, sample_theta, sample_coalescent_times, \
    accepted_tau, accepted_theta, accepted_times \
        = mcmc(steps=sample)
```

Summary statistics of the MCMC

```
# show summary statistics for the runs discarding the first BURNIN
samples
print("Summary for tau samples")
print(pd.DataFrame(sample_tau_burned).describe()[0])
eff = 1/(1+2*sum(acf(sample_tau_burned)))
print("Acceptance: {}".format(accepted_tau/SAMPLES))
print("Efficiency: {}\n".format(eff))

# show summary statistics for the runs discarding the first BURNIN
samples
print("Summary for theta samples")
print(pd.DataFrame(sample_theta_burned).describe()[0])
eff = 1/(1+2*sum(acf(sample_theta_burned)))
print("Acceptance: {}".format(accepted_theta/SAMPLES))
print("Efficiency: {}\n".format(eff))
```

Code for generating trace plots

```
# Plotting trace plots
fig, ax = plt.subplots(1, 2, figsize=(15, 8))

ax[0].plot(list(range(len(sample_tau_burned))),sample_tau_burned,lab
el=f"tau_window={w_tau}")
```

```
ax[0].set_title('posterior tau samples')
ax[0].set_xlabel('Iteration')
ax[0].set_ylabel('tau')
ax[0].legend()

ax[1].plot(list(range(len(sample_theta_burned))),sample_theta_burned
,label=f"theta_window={w_theta}", color='r')
ax[1].set_title('posterior theta samples')
ax[1].set_xlabel('Iteration')
ax[1].set_ylabel('theta')
ax[1].legend()
fig.show()
```

Experiments varying window size

```
# Experiments to exploring varying windows sizes
w_tau = 0.00053
w_theta =  0.0009
w_t = 0.019
tau_windows = np.logspace(np.log10(w_tau)-2, np.log10(w_tau)+2, 3)
theta_windows = np.logspace(np.log10(w_theta)-2,
np.log10(w_theta)+2, 3)

def run_mcmc(tau, theta, tau_window, theta_window):
   tau = tau
   theta = theta
   coalescent_times = np.array([0.001 for _ in range(df.shape[0])])
# 1000 coalescent times
   init_windows = (tau_window, theta_window, w_t)
   init_params = (tau, theta, coalescent_times)
   results = mcmc(steps=SAMPLES, init_windows=init_windows,
init_params=init_params)
   return results

results_list = []
for tau_window in tau_windows:
   for theta_window in theta_windows:
      results = run_mcmc(tau, theta, tau_window, theta_window)
      results_list.append(results) # record results for later use
      sample_tau, sample_theta, _, accepted_tau, accepted_theta, _ =
results
      sample_tau_burned = sample_tau[BURNIN:]
      sample_theta_burned = sample_theta[BURNIN:]
```

```
        print(f"tau window: {tau_window:.3f}, theta window:
{theta_window:.3f}")
        print(f"Summary for tau samples")
        print(pd.DataFrame(sample_tau_burned).describe().loc[['mean',
'std']])
        eff = 1/(1+2*sum(acf(sample_tau_burned)))
        print("Acceptance: {}".format(accepted_tau/SAMPLES))
        print("Efficiency: {}\n".format(eff))

        # show summary statistics for the runs discarding the first
BURNIN samples
        print(f"Summary for theta samples")

print(pd.DataFrame(sample_theta_burned).describe().loc[['mean',
'std']])
        eff = 1/(1+2*sum(acf(sample_theta_burned)))
        print("Acceptance: {}".format(accepted_theta/SAMPLES))
        print("Efficiency: {}\n".format(eff))
        # Plotting trace plots
        fig, ax = plt.subplots(1, 2, figsize=(15, 8))


ax[0].plot(list(range(len(sample_tau_burned))),sample_tau_burned,lab
el=f"tau_window={tau_window:.3f}")
        ax[0].set_title('posterior tau samples')
        ax[0].set_xlabel('Iteration')
        ax[0].set_ylabel('tau')
        ax[0].legend()


ax[1].plot(list(range(len(sample_theta_burned))),sample_theta_burned
,label=f"theta_window={theta_window:.3f}", color='r')
        ax[1].set_title('posterior theta samples')
        ax[1].set_xlabel('Iteration')
        ax[1].set_ylabel('theta')
        ax[1].legend()

        plt.show()
        print(f'For tau, we obtain a posterior mean of
{np.mean(sample_tau_burned):.6f} \nwith 2.5% and 97.5% cred.
interval [{np.quantile(sample_tau_burned, 0.025):.6f},
{np.quantile(sample_tau_burned, 0.975):.6f}]')
        print(f'For theta, we obtain a posterior mean of
{np.mean(sample_theta_burned):.6f} \nwith 2.5% and 97.5% cred.
interval [{np.quantile(sample_theta_burned, 0.025):.6f},
{np.quantile(sample_theta_burned, 0.975):.6f}]')


        print('\n')

# You can then analyze the results and plot the running time,
acceptance rate, and efficiency for each combination of tau and
theta window sizes.
```

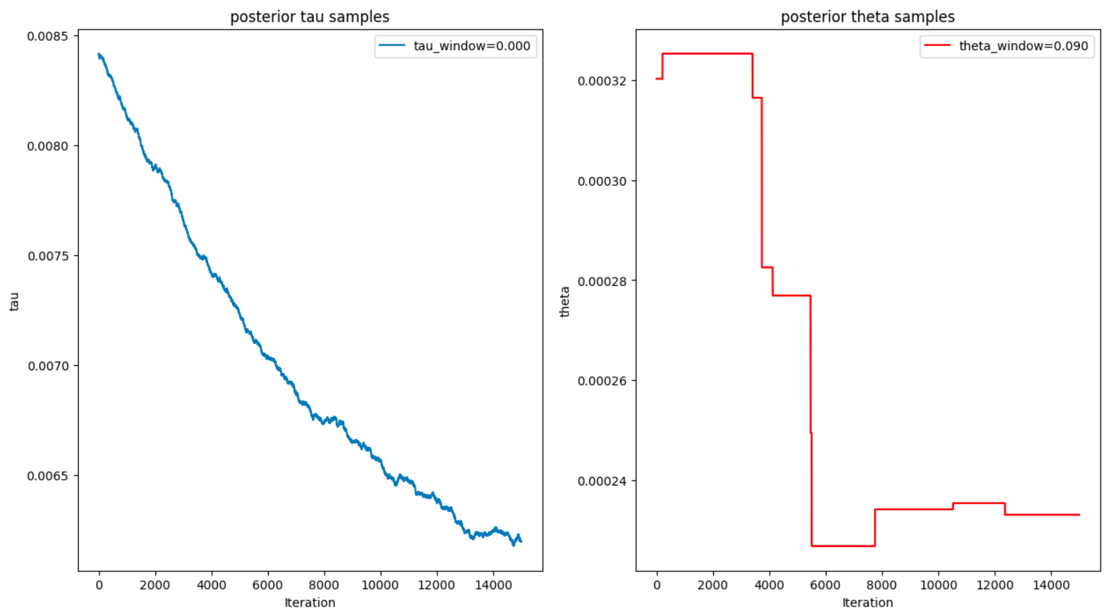

Figure 2: Example of an inefficient MCMC caused by ill-selected window sizes, where we observe no convergence after several iterations.

τ window	θ window	Acceptance rate (τ, θ)	Efficiency (τ, θ)	Posterior mean with credibility interval
5.3e-06	9.e-06	0.89, 0.73	(0.059, 0.059)	τ: 0.0070, [0.006164, 0.008422] θ: 0.000094, [0.000061, 0.000140]
5.3e-06	0.001	0.89, 0.021	0.059, 0.059	τ: 0.0070, [0.006164, 0.008422] θ: 0.0039, [0.003416,

				0.004609]
5.3e-06	0.090	0.89, 0.01	0.059, 0.058	τ: 0.0070, [0.006215, 0.008317] θ: 0.00026, [0.000227, 0.000325]
0.001	9.e-06	0.44, 0.97	0.066, 0.059	τ: 0.0049, [0.004494, 0.005270] θ: 0.002, [0.001415, 0.002608]
0.001	**0.001**	**0.43, 0.43**	**0.08, 0.07**	**τ: 0.0040, [0.003728, 0.004294] θ: 0.0039, [0.003416, 0.004609]**
0.001	0.090	0.43, 0.001	0.077, 0.059	τ:0.004, [0.003726, 0.004322] θ: 0.0039, [0.003394, 0.004633]
0.053	9.e-06	0.009, 0.96	0.0597, 0.059	τ: 0.0049, [0.004573, 0.005115] θ: 0.002, [0.001625, 0.002453]
0.053	0.001	0.010, 0.43	0.059, 0.070	τ:0.004, [0.003684, 0.004370] θ: 0.0040, [0.003287, 0.004624]
0.053	0.090	0.0096, 0.0092	0.0603, 0.059	τ: 0.00402, [0.003793, 0.004243] θ: 0.0040, [0.003314, 0.004662]

Table 2: Full table of the effect of varying window sizes on MCMC posterior samples, acceptance rates and efficiencies. Highlighted are the window sizes for which we manage to obtain the desired acceptance rate for both parameters.