IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 4 October 2023, accepted 16 October 2023, date of publication 20 October 2023, date of current version 27 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3326360

APPLIED RESEARCH

Markov Chain Monte Carlo Methods Applied to a Synchronous Generator Model

KUSHILA JAYAMANNE[1], ZHE BAN[1], MADHUSUDHAN PANDEY[1], ALI GHADERI[2], AND BERNT LIE[1], (Member, IEEE)
[1]Department of Electrical Engineering, IT and Cybernetics, University of South-Eastern Norway, 3918 Porsgrunn, Norway
[2]Department of Mathematics and Science Education, University of South-Eastern Norway, 3918 Porsgrunn, Norway

Corresponding author: Kushila Jayamanne (Kushila.R.Jayamanne@usn.no)

This work was supported by The Research Council of Norway and Equinor ASA through the Research Council Project "Digital wells for optimal production and drainage" (DigiWell) under Grant 308817.

ABSTRACT Markov Chain Monte Carlo (MCMC) approaches are widely used for tuning model parameters to fit process measurements. While modern probabilistic programming languages (PPLs) such as Stan, PyMC, and Turing have made it easier to implement efficient MCMC samplers, configuring them for high dimensional and multi-modal parameter distributions remains a challenging task. In Pandey and Lie (2022), the No-U-Turn Sampler (NUTS) was employed via Turing to estimate parameters of an air-cooled synchronous generator model using real-world experimental data, but the produced posterior distributions were *excessively* narrow. The present study extends the findings in Pandey and Lie (2022) by producing more realistic parameter estimates using the same data. To accomplish this, the study first reviews the basics of MCMC; it offers some general advice for choosing appropriate settings for MCMC to ensure successful estimation, as well a discussion of the impact of measurement data on the computation of posteriors. The study then implements the simple classical MCMC technique, Metropolis, from scratch to estimate the generator model parameters, providing more insight into MCMC — its fundamental process and terminology. Finally, the knowledge gained is applied to select appropriate settings for NUTS — implemented via Turing — that yield more accurate parameter estimates.

INDEX TERMS Markov chain Monte Carlo, Metropolis-Hastings, model uncertainty, no-U-turn sampler, parameter estimation.

I. INTRODUCTION

The sensor-actuator configuration, or control architecture [2], is crucial to the design of a control system. Physical and budgetary constraints prevent us from monitoring and adjusting every state of a system. Thus, successful control involves judicious placement of a few sensors and actuators. Ignoring system uncertainty in this regard might compromise the reliability and performance of the system. Therefore, uncertainties must be reduced whenever possible and any remaining uncertainties must be accounted for; when process data is available, parameter estimation may be useful for this.

In this work, we use the thermal model of an air-cooled synchronous generator presented in [3] as a case study. The operation of such a generator on the power grid is strictly regulated by the transmission system operator (TSO). The TSO typically specifies hard constraints on the allowed power factor (PF). One reason for this restriction is to prevent overheating the generator. Recently, it has been suggested that through continuous monitoring of the generator's temperature, there is potential to temporarily relax the PF constraints for short periods [4]. This adjustment could facilitate handling frequency fluctuations in the grid. If this approach is adopted, it becomes essential to keep track of the generator's temperature evolution to avoid destroying it or significantly reducing its lifespan. Consequently, development of methods to assess and minimize

The associate editor coordinating the review of this manuscript and approving it for publication was Paolo Giangrande.

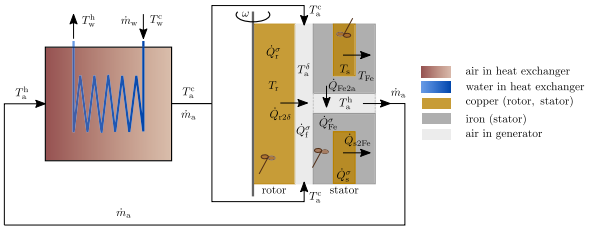


FIGURE 1. Thermal process of the synchronous generator. Source: [3].

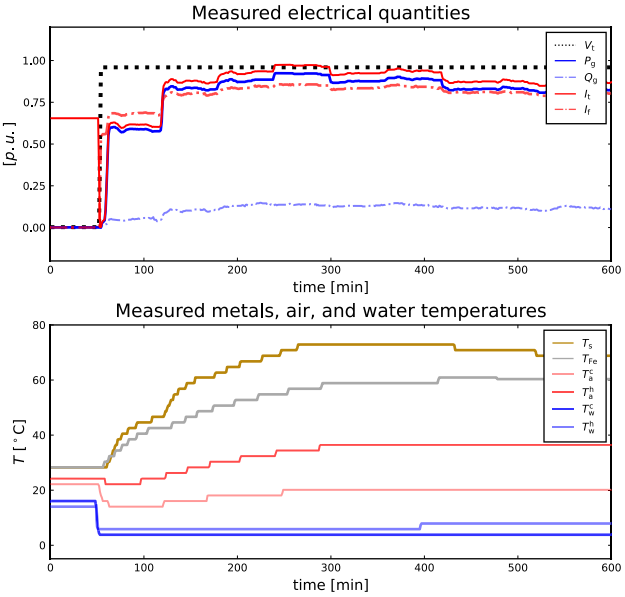


FIGURE 2. Experimental data for the generator model. Source: [3].

the uncertainties associated with temperature monitoring becomes a central objective. Previous studies have dealt with state and parameter estimation for the generator model presented in [3]: in [1] the No-U-Turn Sampler (NUTS) [5] — a popular MCMC algorithm — is used. The posterior distributions of the parameters computed in that study, however, are extremely narrow, i.e., there is very little residual uncertainty. It is of significant interest to investigate the cause for this.

The primary aim of this paper is to reduce uncertainty in the thermal generator model using MCMC methods and experimental data. Accordingly, we look at how a simple MCMC algorithm, Metropolis, can be implemented from scratch; this would give us some insight into the underlying mechanics of MCMC methods, even though not all methods are created equal. Then we apply the more sophisticated NUTS algorithm via Turing [6] in Julia [7].

The paper is organized as follows: Section II provides a description of the thermal generator model; Section III briefly introduces Bayesian inference and how it relates to MCMC techniques; Section IV gives a brief introduction to MCMC methods, discusses in detail the Metropolis algorithm, and explains how NUTS differs from Metropolis; Section V describes how to use Turing to implement MCMC; Section VI presents the results of parameter estimation; Section VII discusses some noteworthy findings; and Section VIII draws some conclusions regarding the work.

II. THERMAL GENERATOR MODEL

Fig. 1 depicts the thermal process of the air-cooled synchronous generator considered in this work. The cold air from the heat exchanger is blown into the air gap between the rotor and stator using a fan. There, the air gets heated by the heat flow from the rotor, the windage of the air gap, and the bearings' friction. The heated air is then directed into the iron cores, where it gets heated further by the heat flow from the iron cores. Finally, the heated air is collected from an outlet at the stator and returned to the heat exchanger. There, it is cooled by a steady supply of cold water. The cooled air is then delivered back into the rotor/stator air gap forming a closed loop system.

The mathematical model used here is proposed in [3]; it is an extension of the model proposed in [4]. The system is modelled by a set of Differential-Algebraic Equations (DAE), which, for the purposes of our work, we reformulate as a mass matrix ODE as follows,

$$M\frac{\mathrm{d}\,(x,z)}{\mathrm{d}t} = f(x,z,u,t;\theta)$$

where

$$\begin{aligned}
x &= (T_\mathrm{r}, T_\mathrm{s}, T_\mathrm{Fe})\,, \text{ differential variables}\\
z &= \left(T_\mathrm{a}^\mathrm{c}, T_\mathrm{a}^\delta, T_\mathrm{a}^\mathrm{h}\right), \text{ algebraic variables}\\
u &= \left(I_\mathrm{f}, I_\mathrm{t}, T_\mathrm{w}^\mathrm{c}\right), \text{ control inputs}\\
\theta &= \left(\mathcal{U}A_{\mathrm{r}2\delta}, \mathcal{U}A_{\mathrm{s}2\mathrm{Fe}}, \mathcal{U}A_{\mathrm{Fe}2\mathrm{a}}, \mathcal{U}A_\mathrm{x}, \dot{Q}_\mathrm{Fe}^\sigma, \dot{Q}_\mathrm{f}^\sigma\right),\\
&\quad \text{unknown parameters}\\
M &= \mathrm{diag}\,(1,1,1,0,0,0)\,, \text{ the mass matrix.}
\end{aligned}$$

The differential variables x are the rotor temperature T_r, the stator temperature T_s, and the iron core temperature T_Fe. The algebraic variables z are the cold air temperature T_a^c, the air gap temperature T_a^δ, and the hot air temperature T_a^h. The control inputs u are the rotor field current I_f, the terminal current I_t, and the cold water temperature T_w^c. The unknown parameters are the heat transfers from rotor to air gap $\mathcal{U}A_{\mathrm{r}2\delta}$, from stator copper to iron $\mathcal{U}A_{\mathrm{s}2\mathrm{Fe}}$, from stator iron to air $\mathcal{U}A_{\mathrm{Fe}2\mathrm{a}}$, from air to water $\mathcal{U}A_\mathrm{x}$; stator iron generated heat $\dot{Q}_\mathrm{Fe}^\sigma$; and friction heating $\dot{Q}_\mathrm{f}^\sigma$. For the complete description of the mathematical model, see [3].

The measured variables of the system are the control inputs $I_\mathrm{f}, I_\mathrm{t}, T_\mathrm{w}^\mathrm{c}$; two of the differential variables, T_s and T_Fe; and two of the algebraic variables, T_a^c and T_a^h. Experimental data for these are available in [8]; the data are plotted in Fig. 2. (Note: I_t is not measured but is calculated using a mathematical expression that relates it to generator terminal voltage V_t, active power of the generator P_g, and reactive power of the generator Q_g — which are all measured quantities.)

This study aims to estimate the following critical parameters of the generator model:

- Heat transfer coefficients: Due to their susceptibility to variable environmental conditions such as humidity and air composition, accurate measurement of the coefficients $\mathcal{U}A_{\mathrm{r}2\delta}$, $\mathcal{U}A_{\mathrm{s}2\mathrm{Fe}}$, $\mathcal{U}A_{\mathrm{Fe}2\mathrm{a}}$, and $\mathcal{U}A_{\mathrm{x}}$ is challenging.
- Heat sources: Heat generated by the stator iron $\dot{Q}^{\sigma}_{\mathrm{Fe}}$ and friction heating $\dot{Q}^{\sigma}_{\mathrm{f}}$ cannot be measured; only educated guesses of their magnitudes are available.
- Initial states: Uncertainty in initial states can propagate through the entire model and influence subsequent predictions. Hence, it is crucial to accurately estimate the initial states $T_{\mathrm{r}}(t=0)$, $T_{\mathrm{s}}(t=0)$, and $T_{\mathrm{Fe}}(t=0)$. $T_{\mathrm{r}}(t=0)$ is not directly measured due to rotor motion; although wireless sensors are an option, their practical use in synchronous generators is severely limited, and the concept is still in its early stages of exploration. $T_{\mathrm{s}}(t=0)$ and $T_{\mathrm{Fe}}(t=0)$ are measured, yet they are compromised by measurement noise and error.
- Measurement variances: Intrinsic to this work is also the estimation of measurement variances, $\mathrm{Var}(T_{\mathrm{s}})$, $\mathrm{Var}(T_{\mathrm{Fe}})$, $\mathrm{Var}(T^{\mathrm{c}}_{\mathrm{a}})$, and $\mathrm{Var}(T^{\mathrm{h}}_{\mathrm{a}})$, which encompass the inherent measurement uncertainties.

Accurately estimating these parameters and integrating them into the model will yield a more reliable representation of the generator's thermal dynamics. This knowledge will facilitate the development of optimized control strategies, ultimately contributing to improved performance and longevity of the generator.

III. BAYESIAN INFERENCE

Parameter estimation techniques based on Bayesian inference generate posterior probability density functions (PDFs) for the unknown parameters instead of point estimates. A posterior PDF serves as a model that captures the inherent uncertainty surrounding the true values of the estimated parameter, given the available evidence.

Bayesian inference involves revising our beliefs about a set of parameters θ based on observed data D. This process relies on Bayes' theorem

$$\mathrm{P}(\theta|D) = \frac{\mathrm{P}(D|\theta)\cdot\mathrm{P}(\theta)}{\mathrm{P}(D)} \tag{1}$$

where:

$\mathrm{P}(\theta|D)$ = posterior PDF of θ given D,
$\mathrm{P}(D|\theta)$ = likelihood of D given θ,
$\mathrm{P}(\theta)$ = prior PDF of θ,
$\mathrm{P}(D)$ = marginal likelihood or evidence, which is the probability of observing the data over all possible parameter values.

Computation of $\mathrm{P}(\theta|D)$ is known as the *inference* problem.

In many cases, especially those involving a high number of dimensions, deriving an analytical solution for the inference problem is intractable. The primary source of this intractability stems from the computation of the denominator term in (1):

$$\mathrm{P}(D) = \int \mathrm{P}(D|\theta)\mathrm{P}(\theta)d\theta. \tag{2}$$

The integral in (2) is usually either unavailable in closed form or requires exponential time to compute — because it has to evaluate the probability for all possible values of θ. In such scenarios, numerical techniques are frequently employed to approximate the posterior PDFs. Among these techniques, Markov Chain Monte Carlo (MCMC) stands out as the most prevalent. The basic idea of MCMC sampling is to simulate draws from the posterior PDFs. The samples generated provide an estimate of the distribution, which can be used to infer other quantities of interest with a reasonable degree of accuracy.

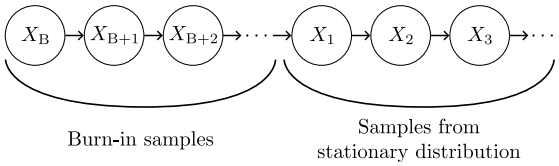


FIGURE 3. Ergodic Markov chain.

IV. MCMC METHODS

MCMC combines two concepts: *Markov Chains* and *Monte Carlo* simulations.

A Markov Chain is a mathematical process involving random transitions from one state to another in a chain. One of its defining characteristics is that the next state X_{n+1} depends solely on the current state X_n — not on those that came before it. This is referred to as the *Markov property*. It is possible to design a Markov chain to have what is known as a *stationary distribution*. This type of chain is referred to as an *ergodic* Markov chain, and it is what serves as the basis for MCMC approaches. After a time of jumping from one state to another, called the *burn-in period*, an ergodic Markov chain will *converge* to its stationary distribution no matter what state it started in. When it does, it will stay at this distribution for all subsequent samples. A simple illustration of an ergodic Markov Chain is provided in Fig. 3.

In MCMC, we engineer a Markov Chain whose stationary distribution is the posterior PDF that we want to sample from. This means that after a period of burn-in, the Markov Chain is going to simulate draws from the posterior PDF. When it does, we use the Monte Carlo method to approximate the posterior PDF. Essentially, we record many samples from the converged Markov Chain and take their distribution to be equivalent to the posterior PDF; from the law of large numbers, the more samples we record, the more accurate the approximated posterior distribution becomes.

To ensure successful approximation of the posterior PDFs it is helpful to pay attention to the following when implementing an MCMC sampler.

- *Initial state*

  Initial state is where the MCMC chain begins. Setting initial values that are likely under the main body of the posterior distribution helps accelerate convergence of the chain. Therefore, this should be done whenever possible.

- *Adaptation samples*

  MCMC algorithms have parameters which affect the efficiency of the sampling process. Most algorithms, e.g. NUTS, include an adaptation phase — distinct from the burn-in phase — in which the parameter values are changed until their optimal is found. The number of adaptation steps used affect whether or not this optimal is found.

  (Note: Similar to the burn-in phase, the samples generated during the adaptation phase are discarded. Majority of models converge to the posterior distribution during adaptation. So, the samples discarded during adaptation are often sufficient, and further burn-in is not required.)

- *Chain length*

  It is essential that the chain contains *enough* samples: enough to reach convergence as well as to achieve an acceptable level of accuracy in estimating the posterior distribution. The number of samples required for this is dependent on model complexity and the MCMC method used; some algorithms are more efficient than others at exploring the parameter space.

- *Number of chains*

  There is no assurance that the MCMC algorithm will converge to the *correct* chain of iterates. Consequently, it is generally required to compute several chains initialized with markedly different initial states, and assess whether or not they all converge to the same distribution. Typically, an evaluation of this sort requires the usage of at least three chains.

- *Measurement data*

  Unnecessarily large data sets with redundant/uninformative samples should be avoided in order to reduce computational cost. This could entail, e.g., decimating the data or down-sampling it, choosing more samples from the transient-states and fewer from the steady-states, etc.

A. METROPOLIS-HASTINGS

The main idea of the Metropolis-Hastings (MH) algorithm [9] is to evolve the Markov chain by randomly proposing a candidate $\hat{\theta}$, at each step, from a *proposal distribution*: given the current position in the parameter space θ_i, the proposal distribution $J(\hat{\theta}|\theta_i)$ defines possible locations to step to next. An accept-reject criterion is then used to decide whether to move to the proposed location or remain at the current one.

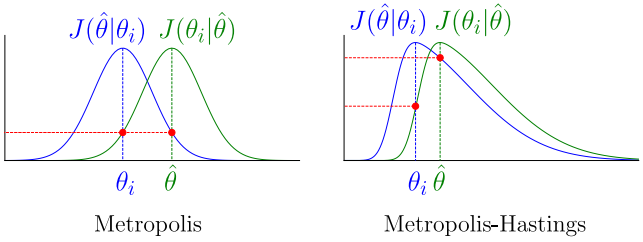


FIGURE 4. MH vs. Metropolis algorithm.

A special case of MH, known as the *Metropolis* algorithm, requires that the proposal distribution be symmetric; this implies that $J(\hat{\theta}|\theta_i) = J(\theta_i|\hat{\theta})$, i.e., the probability of stepping from the current position to the proposed one is the same as the probability of jumping from the proposed position back to the current one. Fig. 4 provides a simple illustration of the difference between the MH and the Metropolis algorithm.

In this work, we use the Metropolis algorithm whose proposal distribution is a Gaussian distribution with specified variance σ^2 — the tuning parameter of the Metropolis sampler. Below we state the parameter estimation problem and present the steps of the Metropolis algorithm.

Problem description:

Given

- a process output *model* (possibly a dynamic model)

$$y = f\,(x, z, u, t; \theta)$$

  where x is the differential variables, z is the algebraic variables, u is the control inputs, θ is the unknown parameters, and t is time,
- prior distributions of the unknown parameters $\mathrm{P}\,(\theta)$,
- measurement data y_m corresponding to the known control input u_m.

Then find

- the posterior distributions of the unknown parameters θ, $\mathrm{P}\,(\theta|y_\mathrm{m}, u_\mathrm{m})$.

The random walk Metropolis algorithm:

1) Initialization:
   - Define the prior distributions $\mathrm{P}\,(\theta)$.
   - Choose the tuning parameter: the co-variance matrix σ^2 of the multivariate *proposal* distribution $\mathcal{N}(\theta, \sigma^2)$.
   - Choose the total number of iterations N.
   - Create an N-vector θ for storing the iterations of the chain.
   - Set the iteration count to $i = 1$.
   - Draw an initial position θ_c (“current”) randomly from $\mathrm{P}\,(\theta)$, and set $\theta\,(i) = \theta_\mathrm{c}$.

The acceptance ratio $\alpha = q_\mathrm{p}/q_\mathrm{c}$ is used to decide whether to accept or reject candidates proposed by the algorithm. The denominator q_c (“current”) and the numerator q_p (“proposed”) are computed in steps 3 and 6, respectively.

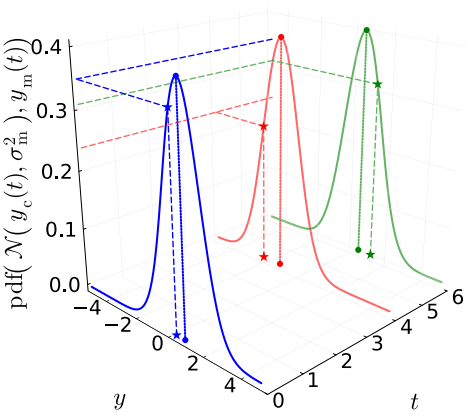


FIGURE 5. Visual illustration of the second term of (3). The circle-markers indicate the measurement data and the star-markers indicate model outputs corresponding to the proposed parameters.

2) Simulate the dynamic model of the system using θ_c to find the current system outputs $y_c(t)$,

$$y_c(t) = f(\theta_c, u_m).$$

3) Calculate $\ln(q_c)$ as follows,

$$\ln(q_c) = \sum_{\theta} \text{logpdf} \cdot (\text{P}(\theta), \theta_c) + \sum_{t} \text{logpdf}\left(\mathcal{N}(y_c(t), \sigma_m^2), y_m(t)\right). \quad (3)$$

Here,

- $\text{P}(\theta)$ is a vector of scalar distributions,
- σ_m^2 is the measurement variance that is either known or estimated as a part of θ.

A visual illustration of the second term of (3) is provided in Fig. 5

4) Propose a candidate position θ_p to move to next by randomly sampling from the proposal distribution centered at the current position $\mathcal{N}(\theta_c, \sigma^2)$.
5) Simulate the dynamic model of the system using θ_p to find out the proposed system outputs $y_p(t)$,

$$y_p(t) = f(\theta_p, u_m).$$

6) Calculate $\ln(q_p)$,

$$\ln(q_p) = \sum_{\theta} \text{logpdf} \cdot (\text{P}(\theta), \theta_p) + \sum_{t} \text{logpdf}\left(\mathcal{N}(y_p(t), \sigma_m^2), y_m(t)\right).$$

7) Assess whether to move to the proposed position θ_p or remain in the current position θ_c.
   a) Calculate $\ln(\alpha)$,

$$\alpha = q_p/q_c$$
$$\ln(\alpha) = \ln(q_p/q_c)$$
$$\ln(\alpha) = \ln(q_p) - \ln(q_c).$$

   b) Draw a random number a from the uniform distribution $\mathbb{U}_{[0,1)}$.
   c) If $\ln(\alpha) \geq \ln(a)$
      - move to the proposed position, $\theta(i+1) = \theta_p$
      - set $\ln(q_c) = \ln(q_p)$

      else
      - remain at the current position, $\theta(i+1) = \theta_c$
      - maintain current $\ln(q_c)$

      end
8) Increment i by 1 and if $i < N$, set $\theta_i = \theta(i+1)$, and go back to step 4.

B. NO-U-TURN SAMPLER (NUTS)

Traditional MCMC techniques like Metropolis can be slow to converge for high-dimensional posterior PDFs, since they use random walks to explore the parameter space. A family of MCMC algorithms known as Hamiltonian Monte Carlo (HMC) [10] improves upon Metropolis by replacing the random walks with Hamiltonian dynamics-based proposals; first-order gradient information from the likelihood guides its every step, enabling much more efficient estimation compared to simplistic random-walk methods. HMC algorithms are featured in modern PPLs like Stan [11], PyMC [12], and Turing [6], contributing to their widespread use in Bayesian inference.

The original HMC algorithm, however, necessitates manual tuning of two critical parameters — the leapfrog step size and the integration time (trajectory length) — to be efficient. [5] proposed a groundbreaking solution to this: the No-U-Turn Sampler (NUTS), an extension of HMC that automates the parameter tuning.

Details of NUTS are omitted here, since PPLs provide automated and efficient implementations. The following section describes how to use NUTS in Turing.

V. MCMC USING TURING

To perform parameter estimation using Turing, the statistical model that should be used for generating samples must be specified using the `@model` macro. The general syntax of a Turing model is as follows.

```
@model function prediction_model(ym, um)
    # Specify prior distribution
    θp ~ P(θ)

    # Simulate process output
    yp = dynamic_model(θp, um)

    # Output: dynamic model + noise
    for i in 1:length(yd)
        ym[i] ~ 𝒩(yp[i], σm²)
    end
end
```

Once we have defined the statistical model, MCMC sampling can be performed using the `sample` function, which has the form

```
sample(prediction_model, sampler,
       parallel, N, nchains).
```

This generates `nchains` number of independent chains, each contaning `N` samples, using the statistical `model` and specified MCMC `sampler`. The sampling is performed in parallel using multiple cores if a `parallel` algorithm is specified.

Turing offers several MCMC samplers, among which are two distinct implementations of the NUTS sampler: AdvancedHMC [13] and DynamicHMC [14]. In this work, we use the implementation from AdvancedHMC. The NUTS sampler function from AdvancedHMC has the form

```
NUTS(n_adapts, δ),
```

where

`n_adapts` is the number of adaptation samples,
δ is the target acceptance rate for dual averaging.

It should be noted that Turing discards the adaptation samples of NUTS by default. In other words, the resulting chains provided by Turing do not include the adaptation samples.

In [5], it was found that NUTS's optimal performance occurs around $\delta = 0.6$, but depends little on δ within the range $\delta \in [0.45, 0.65]$.

VI. PARAMETER ESTIMATION

All the simulations discussed below are implemented in Julia v1.6.5 and are executed on a 2.40 GHz laptop workstation with 32 GB memory. To solve the generator model for parameter estimation calculations we use the DifferentialEquations package [15].

We employ the same priors P (θ) used in [1]. For the initial states and unknown model parameters, truncated normal distributions are adopted, following common practice. Mean values for these distributions are selected based on the best physical knowledge of the system. (In cases lacking such insights, uniform distributions could be used.) As for the ranges/truncation points of the distributions, initially, reasonably informed approximations are available. To refine these ranges, Monte Carlo simulations of the model are conducted using the priors; the goal is to ensure that the simulation results encompass the measurement data, displaying a symmetrically balanced spread around the data (see Fig. 10). Moreover, the refinement also aims to ensure realistic representations of uncertainties in the model predictions; the uncertainties relate to the degree of spread of Monte Carlo results. This entire process involves trial-and-error. When it comes to the priors for measurement noise variances, the conventional choice is the inverse gamma function; the same is chosen in [1]. Since the measurement magnitudes are comparable, the same prior is assumed for all measurement noise variances; comparable magnitudes in general is achieved by normalizing or standardizing the data.

TABLE 1. Prior distributions of the unknown parameters $P(\theta)$ and Metropolis tuning parameter values σ^2.

θ	$P(\theta)$	Metropolis σ^2
$\mathrm{Var}(T_\mathrm{s})$	$\Gamma^{-1}(2,3)$	0.1
$\mathrm{Var}(T_\mathrm{Fe})$	$\Gamma^{-1}(2,3)$	0.1
$\mathrm{Var}(T_\mathrm{a}^\mathrm{c})$	$\Gamma^{-1}(2,3)$	0.1
$\mathrm{Var}(T_\mathrm{a}^\mathrm{h})$	$\Gamma^{-1}(2,3)$	0.1
$T_\mathrm{r}(t=0)$	$\mathcal{T}(\mathcal{N}(30,3),25,35)$	0.6
$T_\mathrm{s}(t=0)$	$\mathcal{T}(\mathcal{N}(30,3),25,35)$	0.3
$T_\mathrm{Fe}(t=0)$	$\mathcal{T}(\mathcal{N}(30,2),25,35)$	0.3
$\mathcal{U}A_{\mathrm{r2}\delta}$	$\mathcal{T}(\mathcal{N}(2.7,1),0.2,5)$	0.12
$\mathcal{U}A_{\mathrm{s2Fe}}$	$\mathcal{T}(\mathcal{N}(20,5),1,50)$	0.4
$\mathcal{U}A_{\mathrm{Fe2a}}$	$\mathcal{T}(\mathcal{N}(15,2),0.5,40)$	0.12
$\mathcal{U}A_\mathrm{x}$	$\mathcal{T}(\mathcal{N}(44,5),1,100)$	0.2
$\dot{Q}_\mathrm{Fe}^\sigma$	$\mathcal{T}(\mathcal{N}(212,40),20,400)$	4
$\dot{Q}_\mathrm{f}^\sigma$	$\mathcal{T}(\mathcal{N}(422,20),200,500)$	4.5

The second column of Table 1 lists all the priors. $\Gamma^{-1}(\alpha,\beta)$ represents an inverse Gamma function with shape parameter α and scale parameter β; $\mathcal{N}(\mu,\sigma)$ represents a normal distribution with mean μ and standard deviation σ; and $\mathcal{T}(P(\theta),\theta_\mathrm{min},\theta_\mathrm{max})$ represents a truncated distribution of a distribution $P(\theta)$ to the interval $[\theta_\mathrm{min},\theta_\mathrm{max}]$.

We write custom Julia scripts to implement the Metropolis algorithm presented in Section IV-A. Three different chains are run for 100 000 iterations each, with different, randomly chosen starting positions; the values of the Metropolis tuning parameter σ^2 that we use for this are given in the third column of Table 1. These values were determined through a process of trial-and-error: different values, several times smaller than corresponding prior variance values, were tried, and the trace plots (evolution of the chains) were examined for convergence. The final trace plots obtained are shown in the left column of Fig. 6. Based on the plots, we choose to discard the first 25 000 iterations as burn-in. The resulting posterior PDFs — after removal of the burn-in iterations — are shown in the right column of Fig. 6; although *mixing* of the chains could be better, we can nevertheless infer that the chains have converged to the target posterior distributions.

We also set up the parameter estimation problem in Turing using the default automatic differentiation backend, ForwardDiff [16]. Three chains are then sampled in parallel using the NUTS algorithm; a target acceptance rate of $\delta = 0.65$ and 1 000 adaptation samples — which are taken to be equivalent to the burn-in samples — are used. Fig. 7 shows the resulting trace plots and posterior PDFs — after removal of the burn-in iterations. Based on the trace plots, the NUTS chains also seem to have converged; mixing of the chains here is observed to be much better than with Metropolis.

It is observed that NUTS generated uni-modal posteriors for all parameters, whereas Metropolis yielded uni-modal distributions only for the measurement variances; all other posteriors are multi-modal. It should be noted here that the same tool, MCMCChains package, is used to summarize and visualize the results of both Metropolis and NUTS. The

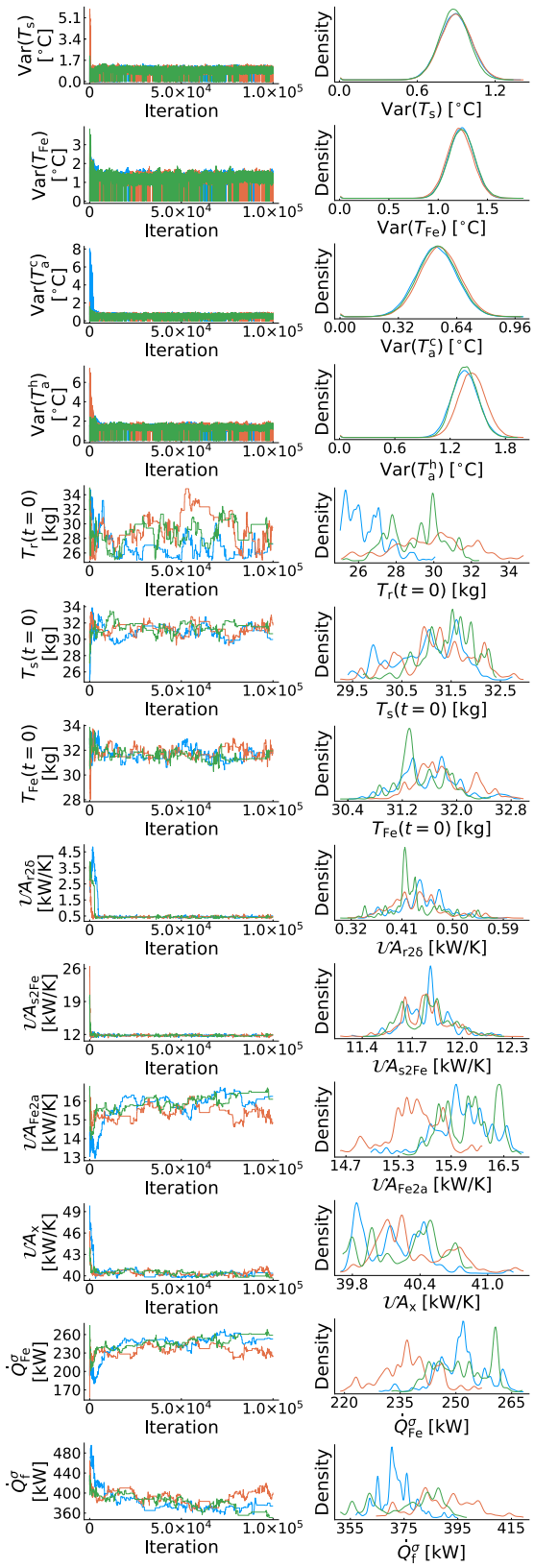


FIGURE 6. Trace plots and posterior PDFs - Metropolis (Note: trace plots include burn-in iterations, i.e., iteration 0 – 25 000; posterior PDFs do not.).

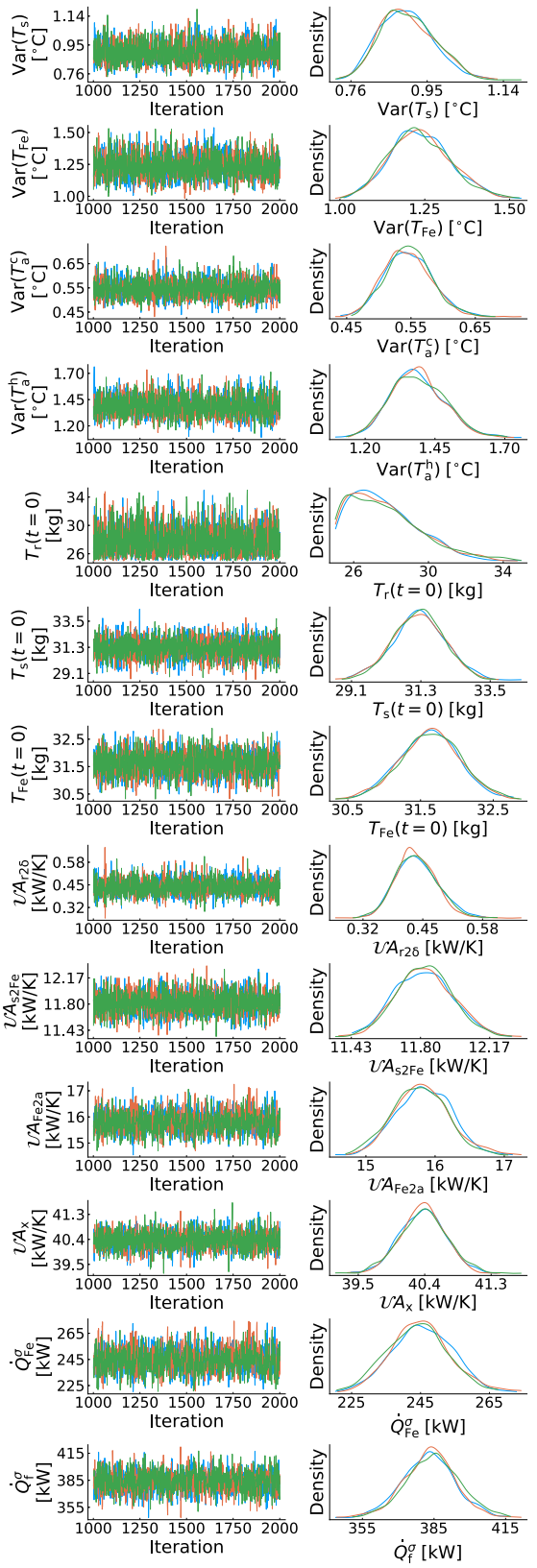


FIGURE 7. Trace plots and posterior PDFs - NUTS (Note: trace plots do not include burn-in iterations, i.e., iteration 0 – 1 000; neither do posterior PDFs.).

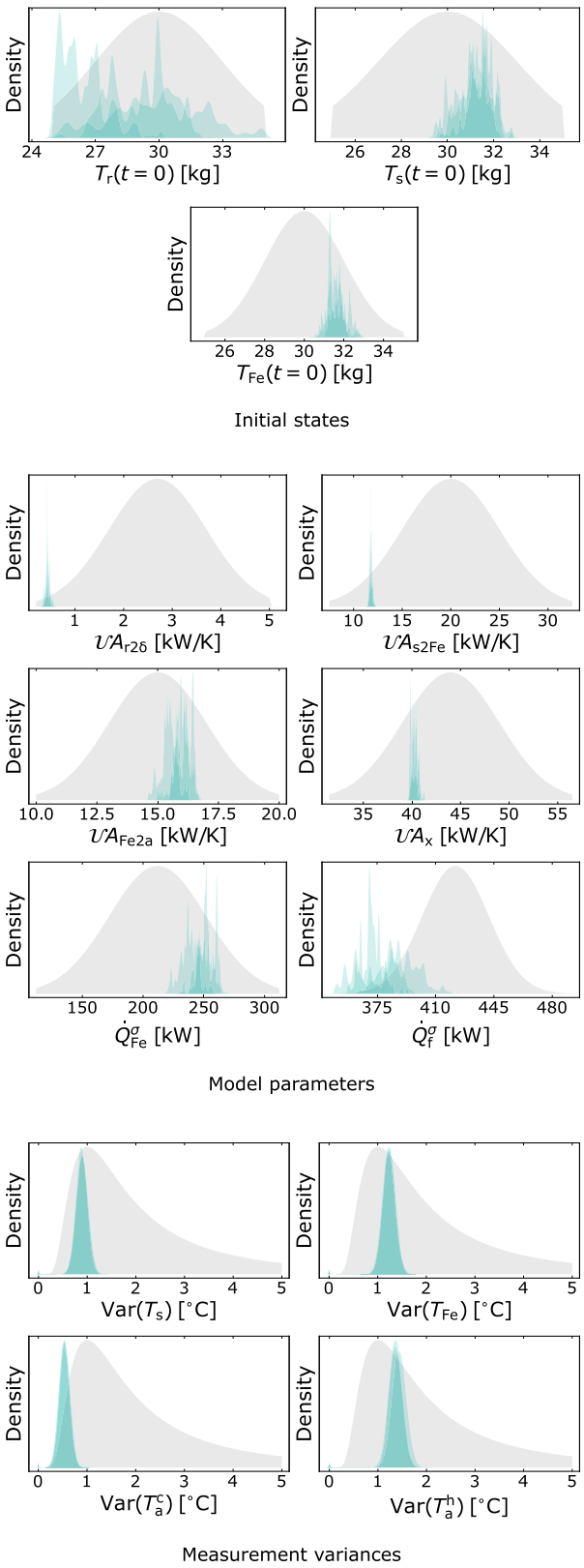


FIGURE 8. Prior PDFs (grey) vs. posterior PDFs (blue) - Metropolis.

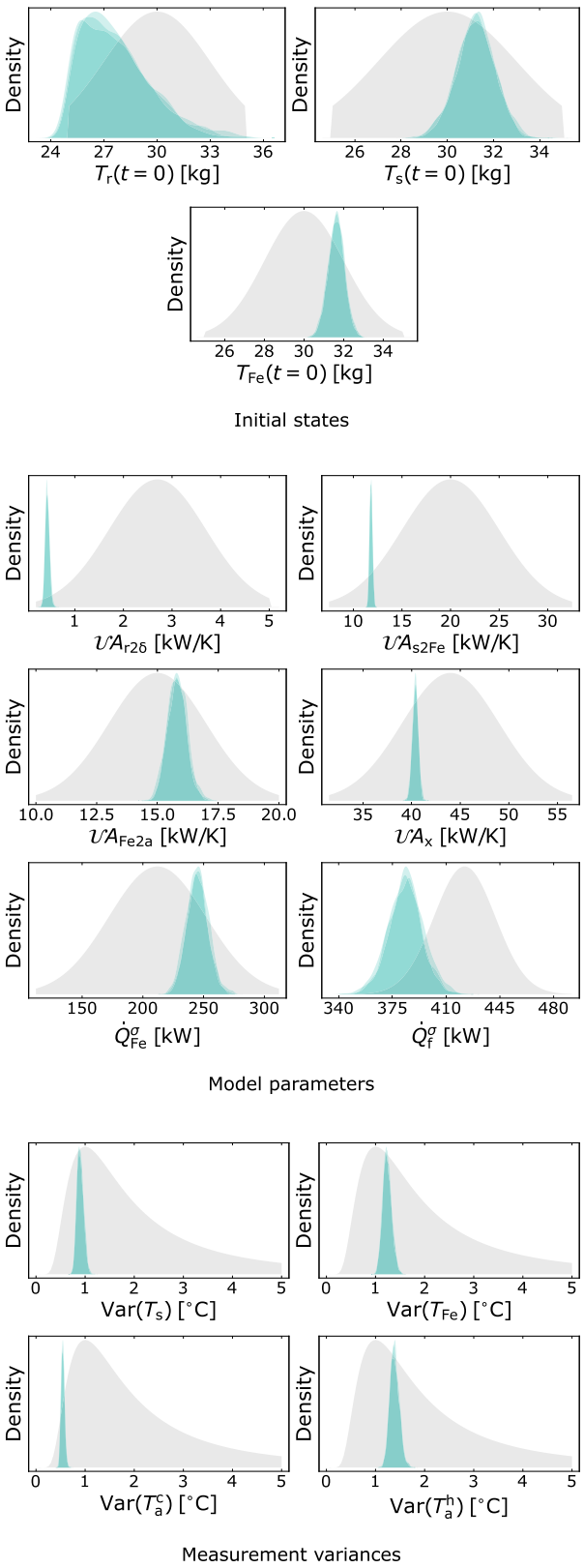


FIGURE 9. Prior PDFs (grey) vs. posterior PDFs (blue) - NUTS.

difference in posteriors is possibly best explained by the algorithms' choice of step length — i.e., distance between consecutive samples — which is reflected in the trace plots. Metropolis trace plots, given in Fig. 6, indicate consistently

small step lengths, in line with our chosen value for the tuning parameter σ^2. In contrast, NUTS trace plots, given in Fig. 7, demonstrate better traversal of the parameter space with more varied step lengths. The extent of "noise" in the chains is indicative of this difference in the manner of stepping: NUTS chains are noisier than Metropolis chains, except the Metropolis chains that correspond to the unimodal posteriors, which appear just as "noisy" as corresponding NUTS chains. Smoother unimodal posteriors may potentially be achieved by adjusting Metropolis's σ^2 parameter and/or by increasing its number of iterations. If the posteriors generated by Metropolis were smoothed out, the resultant distributions would closely resemble those produced by NUTS. Moreover, the posterior statistics — standard deviations and means — in Table 2 for Metropolis and NUTS also largely align. This agreement between Metropolis and NUTS outcomes is a good indication of the success of our implementation of both samplers.

A visual overview of how well the two algorithms estimated the parameters is provided in Fig. 8 for Metropolis and in Fig. 9 for NUTS; these figures illustrate how the calculated posterior PDFs compare to the prior PDFs. The spread of the distributions reflects the associated uncertainty. The extent of uncertainty reduction from prior to posterior is observed to vary across parameters; for certain parameters like $\mathcal{U}A_{\mathrm{r2}\delta}$, $\mathcal{U}A_{\mathrm{s2Fe}}$, $\mathcal{U}A_{\mathrm{x}}$, etc., uncertainty is significantly decreased. However, for others such as $T_{\mathrm{r}}(t=0)$, $T_{\mathrm{s}}(t=0)$, etc., the reduction is less pronounced. Nonetheless, these observations suggest successful uncertainty reduction across all parameters using both sampling methods.

For the majority of parameters, the mode of the posterior closely aligns with the mode of the prior, and the posterior limits remain within the prior limits. This alignment implies that our prior knowledge of the parameters is consistent with the measured data. However, the posterior for $T_{\mathrm{r}}(t=0)$ found by NUTS appears to cross the lower limit of the chosen prior. A similar observation holds true for Metropolis (see Fig. 8), although it is less prominent. These observations could potentially suggest that the prior information does not align well with the observed data, possibly signalling a need for revisiting and revising the priors. However, a check of the sampled values for all estimated parameters reveal that none of them actually lie outside the limits of the chosen prior. Therefore, any apparent crossing of prior limits by the posteriors is solely a result of the plotting procedure and not an actual violation of the prior boundaries.

Our objective for using the Metropolis algorithm in this work was to gain insight into MCMC. Therefore, we do not delve into investigating how it stacks up against NUTS. However, the difference in performance is obvious in many ways: Metropolis requires more burn-in iterations and more samples after convergence to estimate the posterior PDFs, mixing of its chains is poorer, etc. A standard metric for evaluating and comparing the efficiency of different MCMC samplers is the effective sample size (ESS). Due to the Markov property — which is described in Section IV — samples generated using MCMC techniques will typically be correlated. ESS is a measure of the number of generated samples that are truly independent or uncorrelated, and it is generally lower than the total number of generated samples. A higher ESS is generally considered better because it signifies more independent information. For instance, 1,000 samples with ESS 200 offer more information than 2,000 samples with ESS 100. ESS values for our implementations of Metropolis and NUTS are provided in Table 2 (columns 6 and 7); NUTS displays significantly higher ESS values across all the parameters (note: we used far fewer iterations with NUTS than with Metropolis). This finding indicates that NUTS has superior efficiency. Even though we are quite certain that our tuning of Metropolis may be optimized, it is possible that even with expert tuning, it will have a lower efficiency than NUTS; it is widely acknowledged that, at least for some problems, dynamical approaches such as NUTS may be faster [17]. This is primarily due to the fact that Metropolis explores the parameter space via inefficient random walks, while dynamical approaches avoid this random walk behaviour, as stated in Section IV-B.

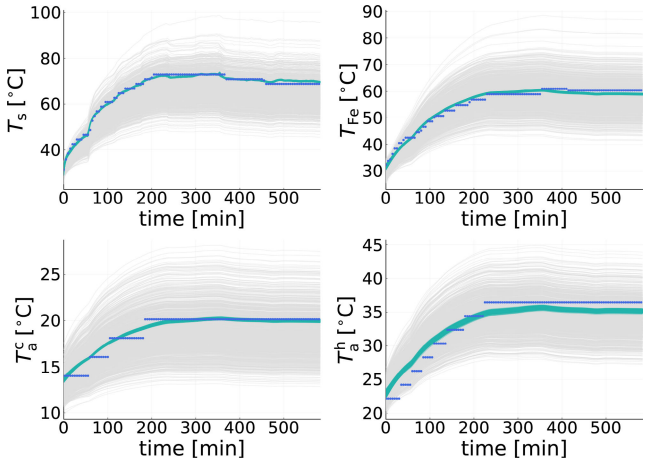


FIGURE 10. Data retrodiction of measured outputs (green) compared to the measurement data (blue) and prior model solutions (grey).

To find out how good the model fit is, we simulate the model for 10 000 randomly picked posterior samples generated using NUTS; this is referred to as retrodiction. In Fig. 10 we compare the retrodiction results with measurement data — which was used for parameter estimation — and model solutions for 10 000 randomly drawn prior samples. The figure reveals that the fitted model closely tracks the measurement data, as might be expected since the same data was used for estimation. It is also seen that the uncertainties in the model predictions have been reduced significantly, and this must be attributed to the use of improved values of the parameters. Thus, we have successfully managed to compute improved estimates of the poorly known model parameters.

The rotor temperature T_{r} — which is not measured — is the most crucial model output. Predictions of T_{r} corresponding to the 10 000 prior samples and the 10 000 posterior samples

TABLE 2. Metropolis and NUTS results statistics.

	μ		σ		ESS [a]	
Parameter	Metropolis	NUTS	Metropolis	NUTS	Metropolis	NUTS
$\mathrm{Var}(T_\mathrm{s})$	0.8899	0.9009	0.1285	0.0678	2368.9769	2863.1459
$\mathrm{Var}(T_\mathrm{Fe})$	1.2275	1.2372	0.1465	0.0899	1726.6946	2366.7368
$\mathrm{Var}(T_\mathrm{a}^\mathrm{c})$	0.5357	0.5459	0.1129	0.0367	4272.1781	3218.9983
$\mathrm{Var}(T_\mathrm{a}^\mathrm{h})$	1.3838	1.3879	0.1634	0.0956	1271.1659	2981.8289
$T_\mathrm{r}(t=0)$	28.4742	27.6622	2.1977	1.9209	457.0653	3158.5520
$T_\mathrm{s}(t=0)$	31.2221	31.2304	0.6694	0.8219	467.8964	3648.1030
$T_\mathrm{Fe}(t=0)$	31.6258	31.6314	0.4154	0.4132	488.0645	2340.7117
$\mathcal{U}A_{\mathrm{r2}\delta}$	0.4383	0.4363	0.0449	0.0455	600.4225	2664.1551
$\mathcal{U}A_{\mathrm{s2Fe}}$	11.7892	11.8194	0.1416	0.1506	644.5336	3502.4526
$\mathcal{U}A_{\mathrm{Fe2a}}$	15.8679	15.8091	0.4170	0.3948	455.9358	1620.1733
$\mathcal{U}A_\mathrm{x}$	40.2556	40.3766	0.3192	0.3119	476.3352	2855.2070
$\dot{Q}_\mathrm{Fe}^\sigma$	246.7329	244.8987	9.6073	8.7633	455.7214	1564.9992
$\dot{Q}_\mathrm{f}^\sigma$	378.8497	383.5085	12.8495	11.0508	458.0514	1667.5499

[a] When comparing ESS, it is important to note that for Metropolis, a total of 75 000 samples — excluding burn-in — were generated per chain, whereas for NUTS, only 1 000 samples were generated per chain.

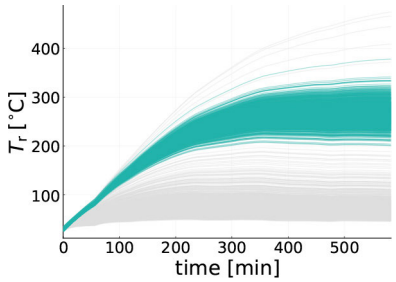


FIGURE 11. Posterior rotor temperature predictions (green) compared to prior rotor temperature predictions (grey).

are shown in Fig. 11. In both instances, the uncertainty of the prediction grows rapidly with time, but in the posterior model, this growth is greatly decreased.

VII. DISCUSSION

The DAE formulation of Section II permits priors to be proposed for both the initial differential and algebraic variables; however, when this was attempted, it seemed that the DifferentialEquations package disregarded the initial algebraic variable priors. This means that the solvers in the DifferentialEquations package compute the initial algebraic variables on their own based on the specified initial differential variables. This stands to reason since algebraic variables are essentially functions of differential variables.

In Fig. 11, the uncertainty in the rotor temperature prediction was observed to grow over time. To address this, online state estimation may be utilized; in [18], Bayesian inference techniques such as the particle filter and the ensemble Kalman filter are used to perform state estimation on the generator model. Such a solution may benefit from using the parameter estimation results produced in this study as apriori estimates for filter initialization.

It should be noted that we used the same measurement data for both estimating the model and then testing the model fit. Although it is preferable to validate the posterior model on new measurement data — that was not used for estimation — retrodiction checks are still helpful in assessing if the model makes sense to explain the observed data.

VIII. CONCLUSION

In this paper, using real-world experimental data, we illustrated how MCMC sampling approaches may be used effectively to address uncertainties in a thermal generator model. We first implemented the Metropolis algorithm from scratch to get a basic understanding of the underlying mechanics of MCMC methods. The Turing package in Julia was then used to apply the more sophisticated NUTS algorithm. Both samplers produced posterior distributions that were comparable and narrower than the predicted prior distributions — but NUTS clearly outperformed Metropolis. We further tested the model's posterior predictive capabilities using the posteriors produced by NUTS; this revealed that the predictive powers had indeed improved greatly.

To conclude, the findings of this study improve on those of [1] — obtained for the same model using the same data and methodologies. The results indicate that employing MCMC, specifically the Metropolis and NUTS algorithm, is an effective way for producing best estimates of the unknown parameters of the thermal generator model. In future work, a better choice of sensors for the generator may be considered in a control architecture study.

REFERENCES

[1] M. Pandey and B. Lie, "Bayesian inference for thermal model of synchronous generator—Part I: Parameter estimation," *IEEE Access*, vol. 10, pp. 103529–103537, 2022, doi: 10.1109/ACCESS.2022.3209232.

[2] G. C. Goodwin, S. F. Graebe, M. E. Salgado, and G. C. Goodwin, *Control System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

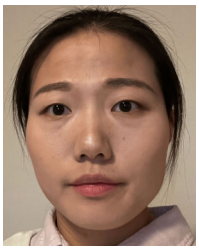
[3] M. Pandey, "Model fitting and state estimation for thermal model of synchronous generator," M.S. thesis, Dept. Elect. Eng., IT Cybern., Univ. South-Eastern Norway, Porsgrunn, Norway, 2019. [Online]. Available: https://openarchive.usn.no/usn-xmlui/handle/11250/2644748

[4] T. Øyvang, "Enhanced power capability of generator units for increased operational security," Ph.D. dissertation, Fac. Technol., Natural Sci. Maritime Stud., Univ. South-Eastern Norway, Porsgrunn, Norway, 2018.

[5] M. D. Homan and A. Gelman, "The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1593–1623, Jan. 2014. [Online]. Available: http://jmlr.org/papers/v15/hoffman14a.html
[6] H. Ge, K. Xu, and Z. Ghahramani, "Turing: A language for flexible probabilistic inference," in *Proc. 21st Int. Conf. Artif. Intell. Statist.*, Mar. 2018, pp. 1682–1690. [Online]. Available: https://proceedings.mlr.press/v84/ge18b.html
[7] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, Jan. 2017, doi: 10.1137/141000671.
[8] T. Øyvang, J. K. Nøland, G. J. Hegglid, and B. Lie, "Online model-based thermal prediction for flexible control of an air-cooled hydrogenerator," *IEEE Trans. Ind. Electron.*, vol. 66, no. 8, pp. 6311–6320, Aug. 2019, doi: 10.1109/TIE.2018.2875637.
[9] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, no. 6, pp. 1087–1092, Jun. 1953, doi: 10.1063/1.1699114.
[10] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid Monte Carlo," *Phys. Lett. B*, vol. 195, pp. 216–222, Sep. 1987, doi: 10.1016/0370-2693(87)91197-X.
[11] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *J. Stat. Softw.*, vol. 76, no. 1, pp. 1–32, 2017, doi: 10.18637/jss.v076.i01.
[12] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in Python using PyMC3," *PeerJ Comput. Sci.*, vol. 2, p. e55, Apr. 2016, doi: 10.7717/peerj-cs.55.
[13] K. Xu, H. Ge, W. Tebbutt, M. Tarek, M. Trapp, and Z. Ghahramani, "AdvancedHMC.jl: A robust, modular and efficient implementation of advanced HMC algorithms," in *Proc. 2nd Symp. Adv. Approximate Bayesian Inference*, Feb. 2020, pp. 1–10. [Online]. Available: https://proceedings.mlr.press/v118/xu20a.html
[14] T. Papp. (2021). *DynamicHMC.jl*. [Online]. Available: https://www.tamaspapp.eu/DynamicHMC.jl/dev/
[15] C. Rackauckas and Q. Nie, "DifferentialEquations.jl—A performant and feature-rich ecosystem for solving differential equations in Julia," *J. Open Res. Softw.*, vol. 5, no. 1, p. 15, May 2017, doi: 10.5334/jors.151.
[16] J. Revels, M. Lubin, and T. Papamarkou, "Forward-mode automatic differentiation in Julia," 2016, *arXiv:1607.07892*, doi: 10.48550/ARXIV.1607.07892.
[17] R. M. Neal, "Probabilistic inference using Markov chain Monte Carlo methods," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep. CRG-TR-93-1, 1993.
[18] M. Pandey and B. Lie, "Bayesian inference for thermal model of synchronous generator. Part—II: State estimation," *IEEE Access*, vol. 10, pp. 105612–105620, 2022, doi: 10.1109/ACCESS.2022.3209695.

KUSHILA JAYAMANNE received the B.S. degree in mechatronics engineering from General Sir John Kotelawala Defence University, Sri Lanka, in 2017, and the M.S. degree in industrial IT and automation from the University of South-Eastern Norway (USN), Norway, in 2021, where she is currently pursuing the Ph.D. degree in IT and cybernetics.

Her research interests include control architecture/linear systems theory, nonlinear and model predictive control, advanced control under uncertainty, and state and parameter estimation.

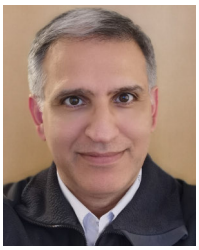


ZHE BAN received the B.S. degree in automation and the M.S. degree in advanced control engineering from The University of Manchester, Manchester, England, U.K., in 2018 and 2019, respectively. She is currently pursuing the Ph.D. degree in information technology with the University of South-Eastern Norway, Porsgrunn, Telemark, Norway.

From 2019 to 2020, she was a Teaching Assistant with The University of Manchester. Her research interests include data-driven physics-informed modeling, application of Bayesian analysis to nonlinear dynamic systems, and data reconciliation.

MADHUSUDHAN PANDEY received the M.Sc. degree in electrical power engineering from the University of South-Eastern Norway (USN), Porsgrunn, Norway, in 2019.

Since 2019, he has been a Ph.D. Researcher with the Telemark Modeling and Control Center, USN. His research interests include modeling, control, optimization of dynamic systems, and integration of dispatchable renewable energy sources with variable renewable energy sources.



ALI GHADERI received the M.Sc. degree in mathematics from the Norwegian University of Science and Technology (NTNU), in 1997, and the Ph.D. degree in chemical engineering from the University of Surrey, U.K., in 2006.

From 1997 to 2006, he was an Assistant and Senior Researcher with Tel-Tek. He then transitioned to the role of a Senior Technologist with Renewable Energy Corporation (REC), from 2007 to 2012, where he contributed to the development of high-efficiency solar cells. Since 2018, he has been an Associate Professor of mathematics with the Department of Mathematics and Science Education, University of South-Eastern Norway (USN). His research interests include Bayesian statistics, information theory, and also their applications in modeling human behavior as well as engineering systems.

BERNT LIE (Member, IEEE) received the Ph.D. degree in engineering cybernetics from the Norwegian University of Science and Technology (NTNU), Norway, in 1990.

From 1987 to 1991, he was an Assistant Professor and an Associate Professor with NTNU. In 1992, he joined the University of South-Eastern Norway, where he is currently a Professor of informatics. His research interests include control relevant modeling and advanced control, with applications mainly within the process and energy industries.