EmuCast: A python package for generating time series forecasts with tunable error in model predictive control design and testing

Rémy Rigo-Mariani, Zhewei Zhang

SoftwareX 34 (2026) 102574

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original Software Publication

EmuCast: A python package for generating time series forecasts with tunable error in model predictive control design and testing

Rémy Rigo-Mariani [a,*], Zhewei Zhang [a,b,c]

[a] Univ. Grenoble Alpes, CNRS, Grenoble INP, G2Elab, 38000 Grenoble, France
[b] CNRS@CREATE, 1 Create Way, #08-01 Create Tower, 138602, Singapore
[c] Energy Research Institute @ NTU, Interdisciplinary Graduate Programme, Nanyang Technological University, 639798 Singapore

ARTICLE INFO

Keywords:
Time Serie forecast
Forecast emulation
Markov Chain Monte Carlo
Control Engineering

ABSTRACT

EmuCast is a lightweight tool for generating synthetic time-series forecasts with tuneable error levels. It is intended for researchers and engineers to test predictive control strategies without needing forecasting expertise. It uses Markov Chain Monte Carlo (MCMC) and a reshaping process to create realistic hourly to sub-hourly forecasts. It is simple with only two main parameters, and do not require any calibration to adapt to any type of data (e.g. electricity demand and generation). It is fast and realistic with errors naturally increasing along the horizon. The package includes examples and datasets for predictive management in the energy sector.

Metadata

Nr	Code metadata description	*Metadata*
C1	Current code version	*v1.1*
C2	Permanent link to code/repository used for this code version	https://github.com/RemyRigo/EmuCast
C3	Permanent link to reproducible capsule	*N.A.*
C4	Legal code license	*MIT License*
C5	Code versioning system used	*git*
C6	Software code languages, tools and services used	*Python, Jupyter Notebook*
C7	Compilation requirements, operating environments and dependencies	*Python ≥ 3.8* *pandas≥2.2.3* *numpy≥2.2.1* *matplotlib≥3.10.0* *seaborn≥0.13.2* *setuptools≥72.1.0* *tqdm≥4.66.5* *openpyxl≥3.1.5*
C8	If available, link to developer documentation/manual	*N.A.*
C9	Support email for questions	remy.rigo-mariani@grenoble-inp.fr

1. Motivation and significance

Model Predictive Control (MPC) is an engineering research field in which systems are controlled using two-step approaches. In typical implementations, the first stage involves predicting the relevant environmental information for the application in question. In the second stage, the actual system setpoints are computed along the control horizon, either as an optimisation problem or using heuristic methods [1]. The decision-making process then considers the predictions from the first stage, along with the system's model equations and potential constraints, to compute the command with given control objective. For example, in power systems, operational planning occurs at various timescales, from several minutes to several days ahead, and requires diverse time series forecasts, such as electricity consumption profiles, energy prices and weather data, to predict renewable generation [2]. At the wholesale electricity market level, a typical MPC application is the 'Day-Ahead' market, in which power generation units and prices are optimised based on predicted consumption profiles for the following day. Clearly, the performance of any MPC strategy is significantly impacted by the accuracy of the forecasts [3].

In the energy sector, as an example, the literature on forecast techniques is obviously very vast to predict electricity consumption [4], heat/cooling demand [5], or renewable generation [6]. The techniques employed range from auto regression methods to advanced machine learning approaches and/ or specific equipment and knowledge such as real-time cloud cover estimation and fish-eye camera observations to forecast the solar radiation [7]. Such approaches generally require extensive hyperparameter tuning, are limited to specific classes of time

* Corresponding author.
E-mail addresses: remy.rigo-mariani@grenoble-inp.fr (R. Rigo-Mariani), zhewei.zhang@grenoble-inp.fr (Z. Zhang).

https://doi.org/10.1016/j.softx.2026.102574
Received 1 December 2025; Received in revised form 29 January 2026; Accepted 19 February 2026
Available online 28 February 2026

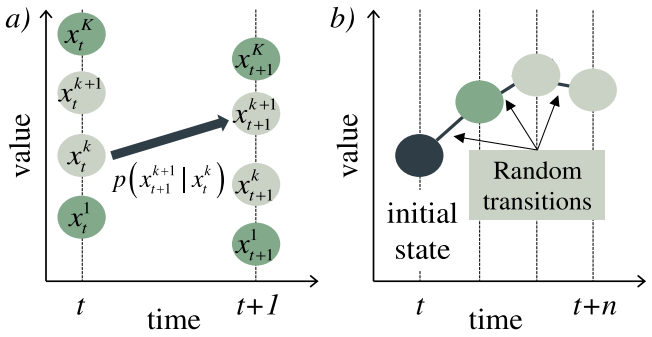

Fig. 1. MCMC sampling - a) state transitions probabilities - b) profile generation by random walk.

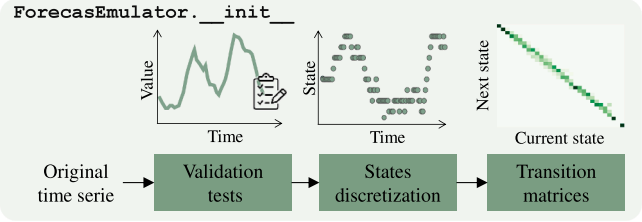

Fig. 2. ForecasEmulator._init_.

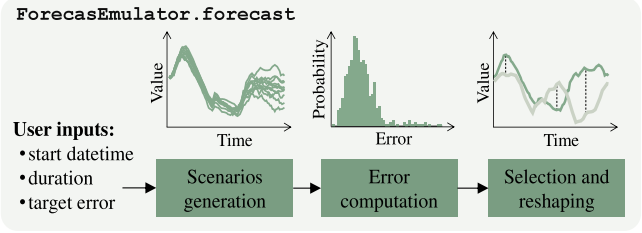

Fig. 3. ForecasEmulator.forecast.

series, and, most importantly, are rarely released as open-source implementations. Some open tools are available, such as Electric Generation Forecasting (EGF)[1] and OpenSTEF.[2] However, similar to the aforementioned studies, these tools lack versatility, as they target a narrow range of applications (e.g., energy consumption or generation only).

For control engineers, this lack of versatility would implies relying on multiple specialized tools that may demand extensive calibration, before actually starting to develop and evaluate predictive control strategies. Consequently, during the development phase of MPC, and in the absence of historical forecast data [7], researchers and engineers often implement simple prediction methods without exogeneous data to provide input profiles for the decision-making stage. The most basic approaches involve generating pseudo-forecasts by adding random noise to historical data [8,9], or employing "backcasting" techniques (e.g. using day D data as an emulated forecast for day $D + 1$) [10]. Statistical methods, such as the Auto Regressive Integrated Moving Average and Geometric Brownian Motion, are also implemented [10,11] alongside machine learning techniques specifically designed for time-series analysis, including Recurrent [12] and/or Convolutional [13] Neural Networks. However, those still lack of versatility and require hyperparameter tuning. Finally, most forecasting methods are designed to maximize accuracy. In practice, however, controller performance can be highly sensitive to forecast quality, which cannot be known a priori. EmuCast instead proposes tunable forecast emulation, enabling a more rigorous and systematic assessment of predictive control strategies under controlled uncertainty levels.

To address the aforementioned challenges, the **EmuCast** package proposed in this paper then provides a lightweight and flexible solution for emulating time-series forecasts with tunable error levels.

2. Software description

2.1. Software architecture

For simplicity purposes, the package is built around a single class ForecastEmulator plus few utilities functions. The use of the class follows the **EmuCast** package primarily relies on Markov Chain Monte Carlo (MCMC) sampling, a method commonly used for profile generation—such as wind generation in [14] or electrical consumption of vehicles in [15]. The method is actually well established in the forecast literature, including in the energy sector for load forecasting [16], generation forecasting [17], and electric vehicle charging demand forecasting [18]. However, similar to the approaches discussed in the introduction, these studies do not provide open-source implementations nor mechanisms to explicitly tune forecast accuracy.

[1] https://github.com/Helmholtz-AI-Energy/electric-generation-forecasting
[2] https://github.com/OpenSTEF/openstef

Fundamentals of Markov Chain Monte Carlo

The process can be broadly divided into two main steps. First, quantifying the transition matrix. Based on historical data, the daily values of a given time series at different time steps are classified into K discrete states, x_t^k denoting the value of the k^{th} state at time t within a day. The transition probabilities from any state k at a given time step to any other state at the next time step are then estimated (Fig. 1a). This produces a series of $K \times K$ transition matrices for each time step in the daily sequence. Second, generating profiles. M profiles are iteratively generated using a random walk [19] guided by the previously estimated transition probabilities, starting from a known initial state and progressing over a defined horizon (i.e., n successive time steps) (Fig. 1b).

Once the transition matrix is built it is possible to generate a profile by random walk from any given datetime t and known start value x_t^k with successive transitions along a time horizon T. In typical MCMC process M profiles $\tilde{y}_t^m$ can then be generated.

Profile Selection and Morphing

Remind that in the proposed procedure the objective is not to accurately predict a given time series profile y_t. Instead the idea is to generate a time series profile that displays a given target error value e^* with this actual profile y_t (which is known in simulation/development phase for control engineers). Thus, once M profiles $\tilde{y}_t^m$ are generated their error with the actual value e^m can be computed using, for instance, the Normalized Root Mean Square Error (NRMSE) or Normalized Mean Absolute Error (NMAE) in %. After the profile generation, the "best" profile is selected as the one whose error is closest to a target error value e^* (1). Finally, the selected profile is reshaped following (2) with a morphing coefficient $\alpha \in [-1, 1]$ applied along the time horizon and analytical functions so that its error precisely matches the targeted e^* value.

$$\tilde{y}_t = \left\{ \tilde{y}_t^m | \underset{m \in M}{\text{argmin}} |e^m - e^*| \right\} \tag{1}$$

$$\tilde{y}_t \leftarrow \tilde{y}_t + \alpha \times (\tilde{y}_t - y_t) \text{suchthat} |e^m - e^*| = 0 \tag{2}$$

Class ForecastEmulator

For simplicity purposes, the package is built around a single class ForecastEmulator plus few utilities functions. The overall architecture follows the two steps previously describe with the initialization/training (i.e. compute the transition matrixes) and the actual forecast generation (profile generation, selection and morphing).

Initialization

The main input for the class initialization (Fig. 2) is the time series for which the adjustable forecast will be generated. A series of validation tests is performed to ensure that the input is a valid single-column time series with a pandas.DatetimeIndex, and that it contains a sufficient amount of data (at least one month is recommended). The time resolution is then automatically inferred, after which the discrete states at each time step are defined. Finally, the transition matrices are computed and stored as the attribute ForecastEmulator.trans_matrices (similarly to Fig. 1a). At this stage, the two main parameters to be configured are the number of discrete states (K) and the number of profiles to generate in the MCMC process (M).

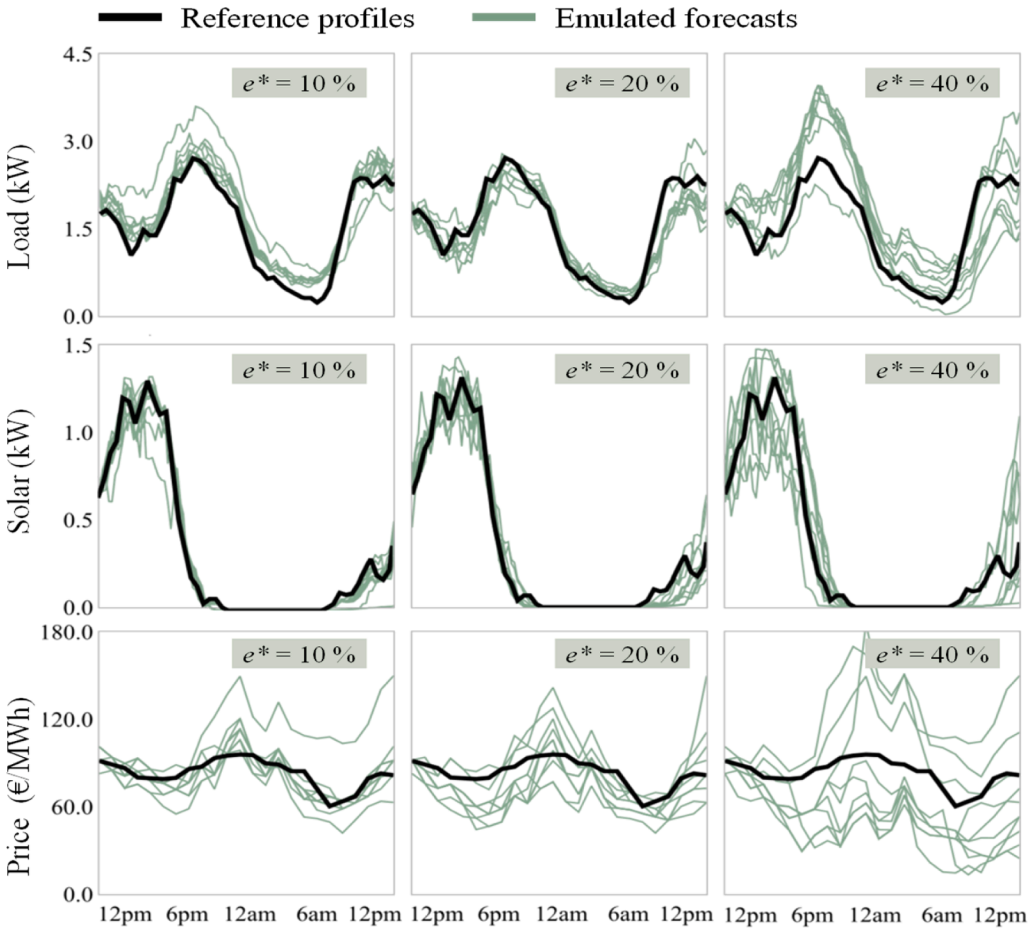

Fig. 4. Illustrative forecast profiles for different target error values.

```
ForecastEmulator.__init__(
   ts_in: pd.Series,                # input Pandas time serie
   nb_states: int = 30,             # number of states at every time steps
   nb_forecast_profiles: int = 300) # numbers of forecast scenarios
```

Pseudo Code

- **User Intputs** : x_t //reference time series - time horizon T
- Rearrange x_t as a serie of daily profiles with daily time steps t (hour, minute)
- Compute x_t^m the different states values at any daily time step.
- From historical value compute the probability to move from one state k at time step t to a state k+1 at time t+1.
- **Output** : $p\left(x_{t+1}^{k+1} \mid x_t^k\right)$

Forecast Generation

Once the emulator has been initialized, the generation, selection, and reshaping of profiles are all handled within a single method called ForecastEmulator.forecast (Fig. 3). The first input of this function is the datetime at which the forecast begins, which must fall within the range of the original time series used during initialization. The second input, duration_minutes, defines the forecast horizon, that is, the number of successive steps in the MCMC process. The final required input corresponds to the target error value (in %). Additional optional arguments allow the user to specify the error metric to be used, provide a custom reference profile y_t (the original time series is used by default), or choose the selection method for determining the "best" profile before the reshaping step.

```
ForecastEmulator.forecast(
   start_time : datetime,          # start datetime
   duration_minutes : int,         # forecast horizon
   target_error : float,           # target error value (in %)
   reference : pd.Series = None,   # reference time series, if None ts_in considered
   n_profiles : int = 300,         # if None nb_forecast_profiles considered
   metric : str = 'nrmse',         # error metric ['nrmse', 'nmae', 'eof']
   selection : str = 'closest')    # profile selection ['closest', 'median']
```

Pseudo Code

- **User Intputs** : t_0 // starting datetime for the forecast generation
  T // time horizon for the forecast
  e^* // target error value
- Identify reference profile y_t with the initial x_{t_0} and corresponding state $x_{t_0}^k$
- Generate M profiles
  For m in M
  $x_t^k = x_{t_0}^k$
  For t in [t0+1, t0+T]
  Compute $\tilde{y}_t^m = x_{t+1}^{k+1}$ from random transition matrixes $p\left(x_{t+1}^{k+1} \mid x_t^k\right)$
  Update $x_t^k = x_{t+1}^{k+1}$
  Compute e^m the error between $\tilde{y}_t^m$ and y_t
- Select the profile $\tilde{y}_t$ that displays the minimum $\left|e^m - e^*\right|$ (1)
- Reshape the selected profile such that $\left|e^m - e^*\right| = 0$ (2)
- **Output** : $\tilde{y}_t$

2.2. Software functionalities

■ **Versatility across different typed of time series:**

The **EmuCast** Consists in generating a forecast (pred) for a given

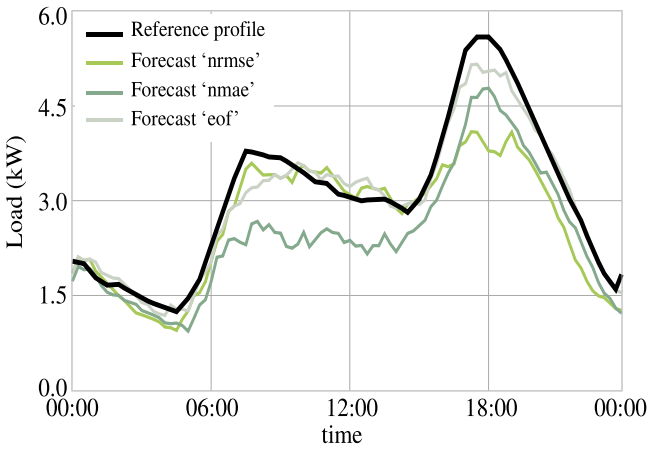

Fig. 5. Illustrative forecast profiles for different error metrics.

reference profile (ref) as illustrated below. Sample datasets for electricity load, photovoltaic generation,[3] and electricity prices[4] are embedded within the package. Fig. 4 illustrate the forecasts generated for those different data and different values for target_error parameter when running the ForecastEmulator.forecast method. The generated profiles highlight the influence of the target error e^*, showing greater deviations from the reference profile for higher values. The results also demonstrate the versatility of the proposed approach which does not require specific parameterization to handle different types of time series.

Preliminary test is also performed to assess the impacts of the two main emulator parameters – the number of discrete state (K) and the number of profiles to generate in the MCMC process (M).

At first, Fig. 6a illustrates sample profiles generated for different values of K, compared to an original sample profile. As the number of discrete states increases, the profiles produced by the MCMC process become smoother. Using electrical load data as an example, the profiles generated for $K = 5$ or10 exhibit unrealistic abrupt variations. Fig. 6b displays the distribution of the errors' values (NRMSE in %) with reference data and for different numbers of profiles generated in the MCMC process (i.e. step "error computation" in Fig. 6b). As the number of profiles M increases, the distribution of errors becomes more uniform and spans a wider range of values. This, in turn, increases the likelihood of obtaining a profile whose error value is close to the target error e^* specified by the user. Increasing both K and M then tends to improve the performances of the emulator. Their impact on computational time is marginal, which is primarily determined by the number of time steps (i.e., the forecast horizon). As illustrated in Fig. 6c, the computational time tends to scale linearly with the length of the generated profile and remains under one second for emulating a 72-hour forecast at 15-minute resolution (i.e., 288-time steps), which would be equivalent to 12 days at 1 h resolution.

A method ForecastEmulator.parameters_tuning, has then been added to the core class in order to fine-tune two parameters (discrete state (K) and the number of profiles to generate in the MCMC process (M)) through successive tests. The number of discrete states is determined when the average difference between two consecutive time steps in the MCMM profiles converges toward the value computed from the reference (training) dataset, based on the minimum mean error across

```
# Import the main class and sample data
from emucast import ForecastEmulator
from emucast.data import load_sample_data

# Initialize the emulator with the sample data
emulator = ForecastEmulator(load_sample_data)

# Generate a forecast profile
from datetime import datetime
ref, pred = emulator.forecast(start_time = datetime(2019,1,9,0,0),
                              duration_minutes = 60*24,
                              target_error = 10,
                              metric = 'nrmse')
```

■ Different error metrics:

The package also offers the possibility to consider different error metrics among the most common; namely the Normalized Root Mean Square Error ('nrmse'), the Normalized Mean Average Error ('nrmse') and the Error of Fir ('eof') that itself derives from the Goodness of Fit metric (or Chi-Square). This is done while setting the metric parameter when running the ForecastEmulator.forecast method (the default value is 'nrmse'). Fig. 5 then illustrates the results obtaining when emulating the forecasts with different metrics and while targeting a prediction error of 20 %.

■ Parameters tuning

random runs. The number of profiles is selected to maximize the spread of the error distribution obtained when generating profiles in the MCM process. The Fig. 7 presents the fitting results reported in the console during the tuning of nb_states (Fig. 7a) and nb_profiles (Fig. 7b). The selected values are determined using the elbow method, i.e., by identifying the point at which a break in performance improvement is observed.

3. Illustrative examples

Remind that the tool developed is intended for engineers and researchers in model predictive control (MPC) applications, at the design and test stage. Along with the core module and sample data, the package also includes two notebooks `da_market.ipynb` and `ems_rolling.ipynb` that illustrates practical uses of the forecast emulator in MPC applications in the power and energy sector. In both applications, the forecast emulator is executed within a temporal loop over the simulated

[3] https://www.kaggle.com/datasets/pythonafroz/electricity-demand-and-solar-generation-data-uk/data

[4] https://newtransparency.entsoe.eu

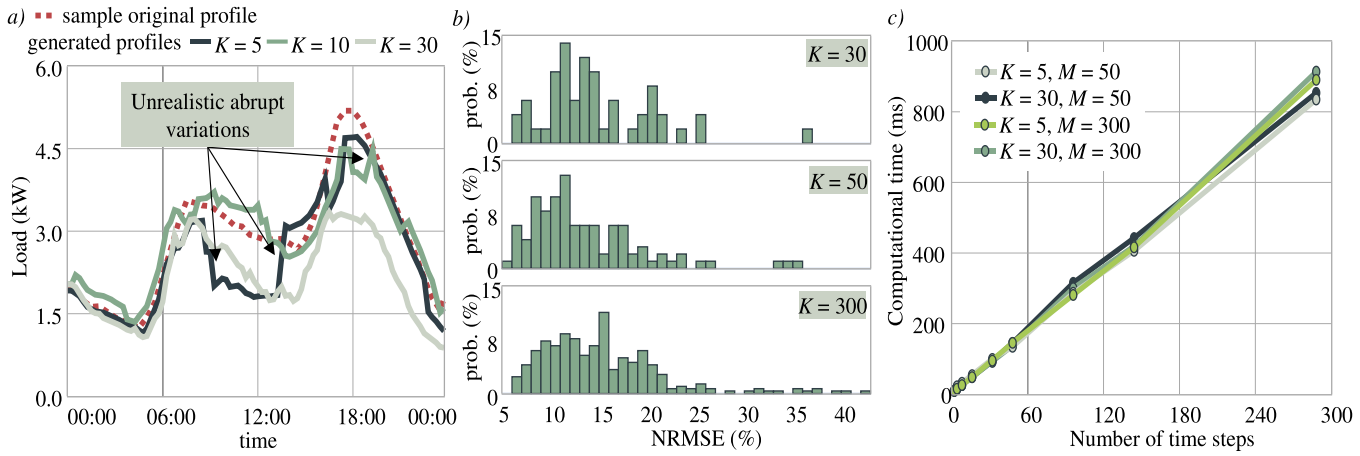

Fig. 6. Impact of emulator settings – a) Sample profiles (original and from MCMC process) for different K values, b) Distribution of error values of the profiles generated for different M values, c) Computational time Vs forecast horizon for different emulator settings.

periods. More details on problem formulation can be found in the notebooks embedded in the package.

3.1. Generic example

The following pseudo-code illustrates the generic use of the package when developing a predictive control strategy. The generic problem consists in testing a control strategy along a time horizon from t_{start}, t_{end}. The objective is to find the best controls of a system over time u_t, given time series information related to the system environment x_t and a performances metric function $f(u_t, x_t)$. The predictive control is run on a rolling window basis with an update run every hour (t_{update}) and a prediction along 24 h (T). Especially, the propose generic implementation allows to compare the control performances with forecasts (tuned with a given error value) against the best theoretical case where the controls would be computed with actual values (known at the development stage).

```
- Retrieve historical environment time series x_t
- Maximize f(u_t, x_t)  // Compute the best possible performances along [ t_start, t_end ]
- emulator = ForecastEmulator(ts in= x_t)     // Initialize the forecast emulator
- e*     // target error value
- While t_start < t_end        // loop over simulation horizon
  - x̃_t =emulator.forecast(start_time = t_start          // Generate Forecast
                           duration_minutes = T,
                           target_error = e*,

  - Maximize f(u_t, x̃_t )  // Compute the control along [ t_start, t_start+T ]based on forecast
  - f(u_t,x_t)  // Compute the performance along [ t_start, t_start+T ]based on actual values
  - t_start = t_start + t_update  // Update the rolling window for next iteration
```

3.2. Storage participating in day-ahead market

The first application example concerns an energy storage system (e.g., a battery) participating in the Day-Ahead energy markets. The objective is to maximize the system's revenue through energy price arbitrage — i.e., charging when prices are low and discharging when prices are high. In practical implementations, price forecasts are made on a daily basis (24 h long forecast at 1-hour resolution) to determine the next day's operating schedule [10].

The performance of this strategy can be evaluated in terms of accuracy (in %) by comparing the results obtained with prediction of different qualities to the ones computed in case of perfect forecasts. Results over ten independent monthly simulations for various target error levels, displays significant performances decrease when the forecast NRSME is greater than 20 % (Fig. 8).

3.3. Energy management strategy on a rolling window

The second application example refers to the operation of a small energy system consisting of a household equipped with photovoltaic (PV) panels and a battery storage unit. In such an application, a typical Energy Management Strategy (EMS) aims to optimize the battery operation to minimize energy exchanges with the grid — thereby reducing the electricity bill [20]. Practical deployment often relies on a "rolling window" EMS, in which predictions of electrical load and photovoltaic generation are successively updated over time. As in the previous example, battery operation is optimized over the entire forecast horizon (e.g., 24 h with 15-minute resolution here), but the predictions are refreshed at regular intervals (e.g., every 2 h, 4 hours, etc.). Fig. 9a illustrates the results obtained from a weekly simulation, with performance assessed in terms of precision relative to the case of perfect. Same as for the previous example, the results indicate that forecast quality

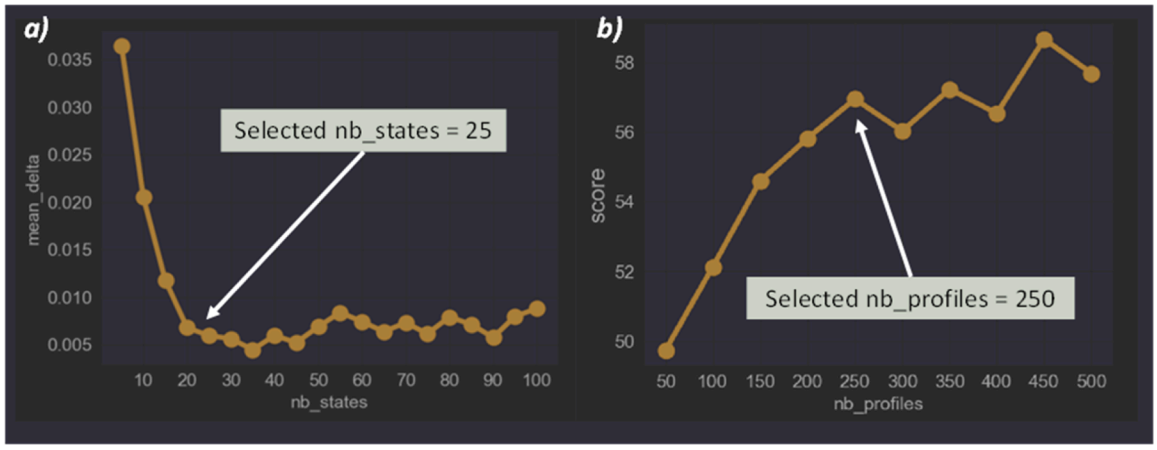

Fig. 7. Console screenshot for the ForecastEMulator.paramters_tuning() method– a) selection of nb_states value - b) selection of nb_profiles value.

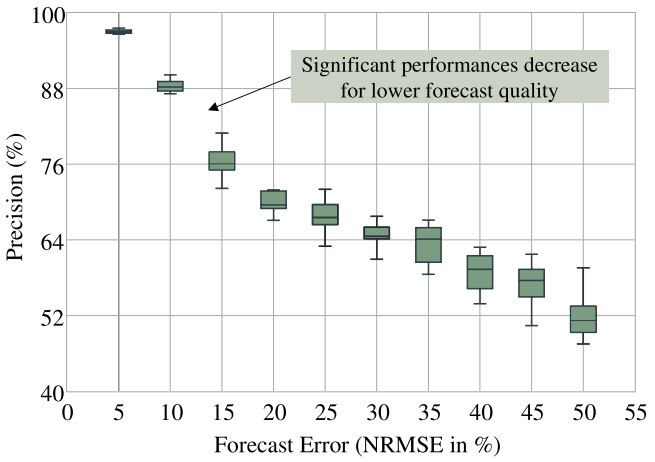

Fig. 8. MPC precision Vs forecast quality for a storage in Day-Ahead market.

(NRMSE in %) can significantly impact the precision. Also, more frequent prediction updates lead to higher precision. This is because frequent updates provide forecasts that are closer to the actual control time, improving decision accuracy. In typical rolling-window implementations, only the control actions computed between two successive forecast updates are actually applied. Each time the MPC is run at a forecast update, the previously computed controls for the remainder of the 24-hour horizon are replaced by the new optimized schedule.

Beyond MPC performance, the results show that forecast accuracy is always higher at the early time steps of the prediction window. This indicates that the emulator behaves logically, with forecast error increasing along the prediction horizon. Fig. 9b illustrates this by reporting NRMSE values over different horizons (24 h, 8 h, and 4 h) for 24-hour forecasts generated at various target error levels e^* (NRMSE Target). The error at the early time steps (e.g., 2 h) is much lower. This behaviour stems from the package implementation: each forecast uses the actual value from the previous time step as the initial state of the random walk. Therefore, when NRMSE is computed over the full horizon (24 h), it naturally matches the defined target value.

4. Impact

The need for the **EmuCast** tool emerged within our research team while studying control applications for battery management [10], smart-home energy optimization [20] or electrical vehicle charging [21] where the lack of accessible historical forecast data repeatedly slowed down development. Each researcher had to design their own ad-hoc forecasting approximation—typically simple noise models or heuristics—which were difficult to reproduce, required manual calibration, and often failed to generalize across different types of time-series data. The objective of is not to produce highly accurate forecasts, but rather to generate realistic forecast profiles suitable for look-ahead control applications. **EmuCast** was created to address this recurring bottleneck by providing a fast, consistent, and domain-agnostic way to emulate realistic forecasts without requiring forecasting expertise. By offering an easy-to-use tool, the package can save significant time for control

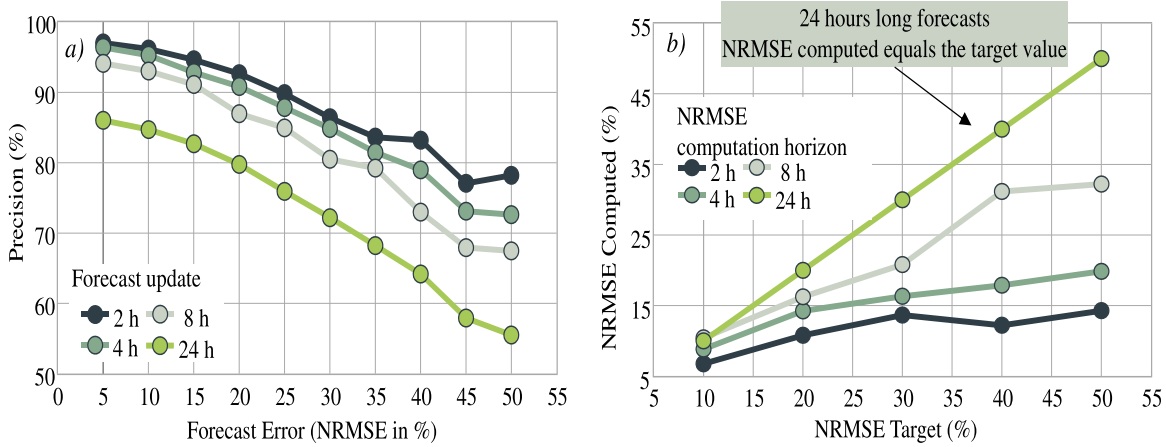

Fig. 9. Rolling window EMS results - a). MPC precision Vs forecast quality for different update periods – b) NRMSE computed Vs target value for 24 h long forecast.

engineers and researchers, enabling them to focus on algorithm design rather than inventing custom forecast generators. **EmuCast** is now shared within the laboratory and among our direct research network, responding to a broader national need within the power and energy systems community.

5. Conclusions

The package introduced in this paper is a tool for the emulation of time series forecasts. It relies on a Markov Chain Monte Carlo (MCMC) process to generate profiles corresponding to given target error values. The tool requires minimal parametrization, with only two main settings (number of discrete states K and number of generated profiles M) and recommended default values based on preliminary tests. Most importantly, it is simple, fast as – it generates a three-day forecast at 15-minute resolution ($\approx$300 steps) in under one second. It is versatile, no specific calibration is needed to handle different types of time series, such as electrical load, photovoltaic generation, or energy prices — all of which are included as examples. Designed for look-ahead and near real-time control applications, the package targets time series with daily variations at hourly to sub-hourly resolution. It is not intended for long-term forecasts at coarse resolution, e.g., yearly load growth. The tool exhibits realistic behaviour, with forecast error increasing implicitly along the prediction horizon. It is primarily intended for researchers and engineers working on Model Predictive Control (MPC) applications at the design stage. Two application examples are included in the package to illustrate its use and highlight the impact of forecast quality on control performance. From a research and development perspective, the package shall be used to assess and improve the robustness of proposed control solutions.

CRediT authorship contribution statement

Rémy Rigo-Mariani: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Funding acquisition, Conceptualization. **Zhewei Zhang:** Writing – review & editing, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Holkar KS, Waghmare LM. An overview of model predictive control. Int J Control Autom 2010;3(4):47–64. Dec.

[2] Wang H, Zhang N, Du E, Yan J, Han S, Liu Y. A comprehensive review for wind, solar, and electrical load forecasting methods. Glob Energy Interconnect 2022;5(1):9–30. https://doi.org/10.1016/j.gloei.2022.04.002. Feb.

[3] Ludolfinger U, Hamacher T, Martens M. A comprehensive evaluation of prediction techniques and their influence on model predictive control in smart energy storage systems. Smart Energy 2025;20:100202. https://doi.org/10.1016/j.segy.2025.100202. Nov.

[4] Gürses-Tran G, Oppermann F, Monti A. ProLoaF: probabilistic load forecasting for power systems. SoftwareX 2023;23:101487. https://doi.org/10.1016/j.softx.2023.101487. Jul.

[5] Niu D, Yu M, Sun L, Gao T, Wang K. Short-term multi-energy load forecasting for integrated energy systems based on CNN-BiGRU optimized by attention mechanism. Appl Energy 2022;313:118801. https://doi.org/10.1016/j.apenergy.2022.118801. May.

[6] Kishore B, et al. Advancing short-term wind power forecasting by AI-driven models for improved accuracy. Electr Eng 2025:1–16. https://doi.org/10.1007/s00202-025-03295-1. Aug.

[7] Hong T, Pinson P, Wang Y, Weron R, Yang D, Zareipour H. Energy Forecasting: a review and outlook. IEEE Open Access J Power Energy 2020;7:376–88. https://doi.org/10.1109/OAJPE.2020.3029979.

[8] Jiang Y, Preece R. Analysis of the impacts of wind power forecast error on power system operation. J Eng 2019;2019(18):4847–51. https://doi.org/10.1049/joe.2018.9363.

[9] Romero-Quete D, Cañizares CA. An affine arithmetic-based energy management system for isolated microgrids. IEEE Trans Smart Grid 2019;10(3):2989–98. https://doi.org/10.1109/TSG.2018.2816403. May.

[10] Mohamed A, Rigo-Mariani R, Debusschere V, Pin L. Operational planning strategies to mitigate price uncertainty in day-ahead market for a battery energy system. IEEE Access 2024;12:85388–99. https://doi.org/10.1109/ACCESS.2024.3415811.

[11] Mohy-ud-din G, Muttaqi KM, Sutanto D. Adaptive and predictive energy management strategy for real-time optimal power dispatch from VPPs integrated with renewable energy and energy storage. IEEE Trans Ind Appl 2021;57(3):1958–72. https://doi.org/10.1109/TIA.2021.3057356. May.

[12] Mahjoub S, Chrifi-Alaoui L, Drid S, Derbel N. Control and implementation of an energy management strategy for a PV–Wind–Battery microgrid based on an intelligent prediction algorithm of energy production. Energies 2023;16(4):1883. https://doi.org/10.3390/en16041883. Jan.

[13] Selvamurugan A, Kunnathur Ganesan P, Nayak SS, Simiyon A, Indiran T. CNN-LSTM-based nonlinear model predictive controller for temperature trajectory tracking in a batch reactor. ACS Omega 2024;9(47):47203–12. https://doi.org/10.1021/acsomega.4c07893. Nov.

[14] Calì D, Wesseling MT, Müller D. WinProGen: a Markov-Chain-based stochastic window status profile generator for the simulation of realistic energy performance in buildings. Build Env 2018;136:240–58. https://doi.org/10.1016/j.buildenv.2018.03.048. May.

[15] Wang Y, Infield D. Markov Chain Monte Carlo simulation of electric vehicle use for network integration studies. Int J Electr Power Energy Syst 2018;99:85–94. https://doi.org/10.1016/j.ijepes.2018.01.008. Jul.

[16] Sreekumar S, Khan NU, Rana AS, Sajjadi M, Kothari DP. Aggregated net-load forecasting using Markov-Chain Monte-Carlo regression and C-vine copula. Appl Energy 2022;328:120171. https://doi.org/10.1016/j.apenergy.2022.120171. Dec.

[17] Al-Duais FS, Al-Sharpi RS. A unique Markov chain Monte Carlo method for forecasting wind power utilizing time series model. Alex Eng J 2023;74:51–63. https://doi.org/10.1016/j.aej.2023.05.019. Jul.

[18] Zheng X, Zhu Y, Wang M, Lv B, Lv Y. Hierarchical Markov chain Monte Carlo framework for spatiotemporal EV charging load forecasting. Appl Sci 2025;15(20). https://doi.org/10.3390/app152011094. Oct.

[19] Bolhuis PG, Swenson DWH. Transition path sampling as Markov chain Monte Carlo of trajectories: recent algorithms, software, applications, and future outlook. Adv Theory Simul 2021;4(4):2000237. https://doi.org/10.1002/adts.202000237.

[20] Rigo-Mariani R, Ahmed A. Smart home energy management with mitigation of power profile uncertainties and model errors. Energy Build 2023;294:113223. https://doi.org/10.1016/j.enbuild.2023.113223. Sep.

[21] Zhang Z, Rigo-Mariani R, Hadjsaid N. Comparative of control strategies on electrical vehicle fleet charging management strategies under uncertainties. Energy AI 2025;21:100522. https://doi.org/10.1016/j.egyai.2025.100522. Sep.