# MathMOuse: A Mathematical MOdels WarehoUSE to handle both Theoretical and Numerical Data

Cyrille Ponchateau
LIAS/ENSMA
86961 – Futuroscope Chasseneuil, France
ponchateau@ensma.fr

Ladjel Bellatreche
LIAS/ENSMA
86961 – Futuroscope Chasseneuil, France
bellatreche@ensma.fr

Carlos Ordonez
University of Houston
Houston, Texas
ordonez@cs.uh.edu

Mickael Baron
LIAS/ENSMA
86961 – Futuroscope Chasseneuil, France
mickael.baron@ensma.fr

## ABSTRACT

The evolution of the numeric technologies triggered an increase in the volume of data to process, along with the technical solutions to handle those volumes. Time series processing is not an exception, since they are widely used in many fields such as finance, medicine or physics. Our studied domain concerns the experimental science, in general, and automatic control in particular, which intensively uses time series data. In such domains, the numerical data comes from the observations of a physical system, usually captured using sensors. The data are then processed to find a mathematical model (usually a differential equation), which models the system dynamic behavior. Due to the volume of available data and technologies to process them, the number of the available mathematical models is increasing as well. Therefore, storing and organizing those models to ease their management and retrieval has become an essential challenge. As a solution, we propose *MathMOuse*, an enriched Data Warehouse structure storing *mathematical models, instead of their raw numerical data* (time series). From an automation scientists point of view, *MathMOuse* is a "query by data" system, where she/he can query the Warehouse considering the raw time series as a query parameter.

During the demonstration, we first illustrate different GUI associated to *MathMOuse*, in terms of storing models, navigating through them, visualizing their data, etc. Then, we will focus on a use case, where a user comes with his raw time series data, obtained from an experiment, and checks whether a model corresponding to these data exists in the warehouse. In the case, where the model exists, the user will save time and efforts, since she/he is not obliged to elaborate the mathematical model representing his data.

## CCS CONCEPTS

• **Information systems** → *Digital libraries and archives*;

## KEYWORDS

Mathematical Models Store, Time Series Analysis, Differential Equations Management

## 1 INTRODUCTION

The evolution of the computer technologies in terms of computing and storage capacity allows processing large sets of numerical data. Time series storage and processing took a great advantage of this trend, especially since they find diverse applications in numerous scientific fields such as: finance (weekly sales total [3]), medicine (electrocardiogram [3, 7], medical surveillance [2]) and physics (particle tracking [7]).

Our work, in particular, was initiated by the automatic control team of our lab. In their domain (as in other experimental sciences domains), the researchers conduct experiments to study physical systems. The electric motor is one of the relevant examples of these systems. When a voltage is applied at the terminals of the motor, it starts rotating and the rotating speed depends on the aforementioned voltage. Then, the aim of the automatic control researcher would be to mathematically describe the dependence between the voltage and the rotating speed. To do so, the motor would be tested with different voltages and the rotating speed evolution would be measured by a sensor. The measurements would be taken at a fixed rate, generating a series of chronological values, represented as a time series. The next step would consist in analyzing the time series, using numerical tools or software such as *Matlab*, *Octave*, R, etc. Finally, the analysis would provide a mathematical model (a differential equation), describing how the rotating speed of the engine behaves according to the voltage.

When the models are identified, the researchers have no standard structure to store them. Therefore, they may be stored in different format, sometimes hidden in Matlab or python scripts, or text files. As a result, retrieving and analyzing a particular model requires time to (i) find its folder and (ii) to read and understand it. In order to bring a solution that would provide both standardization and organization and avoid manual search and retrieval of models, we propose to store the models in a data warehouse associated with a specific ETL process dealing with time series.

In terms of models storage, some initiatives exist. For instance, in [1], the authors explained how they expected "an avalanche of new asteroid models in the next decade" and proposed the DAMIT (Database of Asteroid Models from Inversion Techniques) database. It aims at providing a repository to share those models among the astronomical community. The closest work to ours is a Mathematical Models Database, introduced in [4], which aims at offering a free web-accessible database of models. The studied models are either purely mathematical or physics-based. An important point mentioned by the authors is the analysis of the feedbacks of users of this database (through discussions and email exchanges). This analysis shows a great interest in such tools. Other studies motivate

the development of a database dedicated to store models by the spectacular explosion of models [6, 8, 12]. This situation creates a need to gather them in a unique easily accessible point, in order to help other researchers reaching and analyzing those models. All the aforementioned work mostly aimed at providing public free web-access to mathematical models.

OpenAlea [1], on the contrary, propose a whole Modeling Framework. It is an open source Framework, primarily meant for the plant research community. It aims at collaboratively develop Python libraries and tools useful for Plant Architecture Modeling. It includes modules to analyze, visualize and model functioning and growth of plant architecture, a visual programming tool, methodologies and tools to integrate existing models into Python language. The framework was build to address the complexity of model coupling in plant based modeling and provide a visual programming interface in the same time [10].

The MathMOuse prototype does not focus only on storing models, it also contains numerical support for processing new raw time series data, in order to provide a "query by data" system. The said query system aims at giving the user the possibility to ask the database to return a model or a list of models that fit some raw time series input data. Consequently, the user may not have to build the model from scratch, which would save both his time and energy. OpenAlea also proposes some numerical tools and libraries, but it aims at helping the users designing complex plant architecture models. In OpenAlea, raw data can be used as inputs to exploit a previously built architecture, but not to support the derivation of a model, from those experimental data.

The solution we will demonstrate is based on the classical Data Warehouse structure. The ETL process of this Warehouse has the particularity to handle with both mathematical models data (differential equations, to fit the needs of the automation researchers) and raw time series data. The former data type is used to populate the warehouse, whereas the latter is used for querying it. Since queries involves comparing theoretical models with raw numeric data, the querying system had to integrate some numerical tools for solving equations and statistical comparison of numerical data.

## 2 SYSTEM OVERVIEW

### 2.1 From Binary Tree to ER Model

In a previous paper [9], we explained how equations are represented as binary tries. To persist the equations in a DBMS, the tree structure has to be mapped into an ER model representing the binary tree. We propose the schema displayed on Fig. 1. Storing nested objects such as trees and lists (of variables or initial values or input functions) requires use of associations [5] with many-to-many relationships. Since a star schema requires only one-to-many relationships, it is not well adapted to represent our Models Warehouse schema. However, in the proposed schema, we use a central table, which plays the same role as the fact table of a star schema. Therefore, it will be possible to add actual dimensions to the model, in order to store more information about the equations (date of insertion, experiment, name of the experimenter, etc). For instance, each equation can be associated to the experimenter who added the

[1] http://openalea.gforge.inria.fr/dokuwiki/doku.php

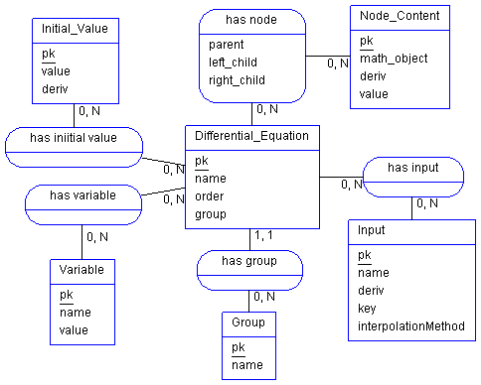


**Figure 1: Star-Like Schema**

equations. The experimenter has his personal identity (name, date of birth, etc) and may also be part of a team, which can be part of a department. A hierarchy can be easily identified: $department \longrightarrow team \longrightarrow experimenter$.

The star-like schema gives the opportunity to perform OLAP operations such as Roll-Up or Drill-Down, in order to group equations that belongs to the same experimenter, or to the same team, or to the same department. Similarly, adding the date of insertion of the equations would require an additional time dimension. The equations could then be aggregated by month or year, which could help to follow the laboratory activity.

To illustrate how a differential equation (corresponding to a given experiment) is stored in the warehouse modeled with our logical schema, let us consider the following example:

*Example 2.1.*

$$Ty' + y = Ku \tag{1}$$

The equation (1) contains two parameters (also called variables), $T$ and $K$, an unknown function $y$, called output, and a known (usually user-defined) function $u$, called input.

The proposed schema has the following tables:

- `Differential_Equations`: it plays the role of the fact table. It contains three measures, the order of the equation, its group and its name. The `order` attribute is automatically derived from the equation formula. Though, the order is a derived field, it is better to store it as a measure, since its computation in SQL is complex and requires the use of a materialized view. Moreover, it is necessary to be able to efficiently know the order of an equation, since it can be used as a condition clause parameter in a query.
- `Node_Content`: The `Node_Content` table, contains the description of the nodes of the tree representing the equation. Each node represents an entity of the equation, which can be of four different types: function, operator, real number or variable. The type is saved in the `math_object` attribute. The `value` attribute is the symbol of the operator (such as + or =), the name of the function (such as $y$ or $u$), the real number value (any real number) or the name of the variable (such as $T$ or $K$). The `deriv` attribute is the order of derivation of

a function. For other types, this field can be ignored. If no other value is specified, it is set to zero by default.
- `Initial_Value`: contains the initial values of the equations. The $y_0$ attribute contains the actual value, while the $deriv$ attribute contains the order of derivation. Therefore, a tuple such as $(y_0 = 0, deriv = 1)$ means $y'_0 = 0$.
- `Variable`: contains the variables ($T$ and $K$ in example 2.1), saved in the attribute `name`, associated to their values, stored in the attribute `value`.
- `Input`: contains the input functions ($u$ in example 2.1). The attribute `name` stores the name of the function as it appears in the equation. The attribute `key` is a unique key, generated by the ETL process. The input function is represented as a time series. The values are stored in an automatically created text file. The `key` attribute is also the name of the file. The attribute `interpolationMethod` is an enumerated field, indicating the appropriate method to use, to interpolate a value between two points of the input time series.

*Serialization of the equations.* In addition to the binary tree and ER Model representations of differential equations, an XML representation has been defined in order to serialize the equations and allow their exchange among several applications.

## 2.2 Comparing Time Series and Equations

It is possible to convert an equation into a time series, by solving the former numerically using numerical algorithms, such as the *Fourth Order Runge-Kutta Method* (see [11, p. 710-714]). This solving process is implemented in the Generator service (see Fig. 4), whose role is to generate time series from the equations. Once the values are generated, they can be compared to a raw input time series, given by the user (taken from the `TS` source Fig. 4).

When comparing an equation to a time series, three main cases may occur:

- There are one or several equations fitting the series. Those equations will be tagged `ModelAccepted`;
- Some equations do not fit the raw time series and are tagged `ModelRejected`;
- Some equations give result that are not close enough from the raw data to accept them, but not different enough either, to reject them. Those models will be tagged `Undetermined`.

The comparison of time series is not an easy task. Since the domain of the values is continuous, the comparison methods need to cope with values approximation and noise. We implemented a comparison technique, consisting in computing the errors between the data calculated from the equations and the input data. Then it calculates the mean error and standard deviation (it is also possible to add the calculation of other statistical values e.g. correlation coefficients). Afterward, discriminatory threshold values, to decide whether an equation fits the data or not are defined. Both the mean and standard deviation will come with a minimum and a maximum threshold value. Using those values, the `ModelRejected`, `ModelAccepted` and `Undetermined` cases can be defined as follows:

- `ModelRejected`: the mean error is greater than the maximum mean threshold or the standard deviation is greater than the maximum standard deviation threshold;

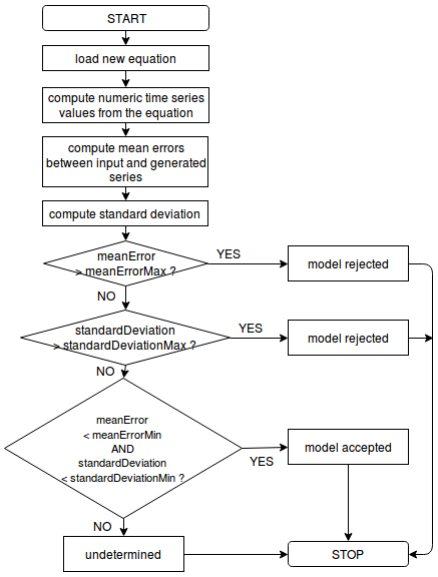


**Figure 2: Comparison algorithm diagram**

- `ModelAccepted`: both mean and standard deviation are less than the minimum mean and standard deviation thresholds;
- `Undetermined`: none of the two previous cases is met.

Fig. 2 gives the diagram of the comparison algorithm described above.

# 3 MATHMOUSE IMPLEMENTATION

## 3.1 Software Architecture

The schema of a Mathematical Models Warehouse general architecture has been published in [9]. *MathMOuse* implements the said architecture, using a microservices software architecture described on Fig. 4 (further explanation are given in subsection 3.2). The source files are available at the forge of our laboratory: **https://forge.lias-lab.fr/projects/mathmouse/files.** This Website contains an instruction manual to use our tool.

Generating and comparing time series are the actions that require the most resources in terms of the processing time. Using microservices technology allows having multiple instances of the same services, in order to *parallelize* the processing. The `Comparators` and `Generators` are represented as dashed box on Fig. 4, since their number can vary from one to five instances, depending on the processing requirements.

Fig. 3, shows how the variation of the number of service instances impacts the general execution time. The left and right graphs display the results for a fixed number of generators and comparators, respectively (on the left, the curves for 3 and 5 comparators overlap each other). The time fall is significant, when the number of generators increases. The number of comparators starts having an impact only when the number of generators is sufficient. The interest of microservices is the possibility to add only the needed services. The tests were made with 100 models in the database and a series of 2000 elements to compare to all the models. The `Populator` service is not useful in the normal use of the prototype. But it allows adding

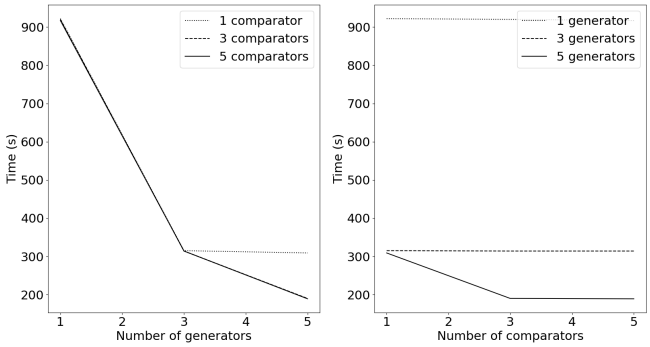


**Figure 3: Execution time**

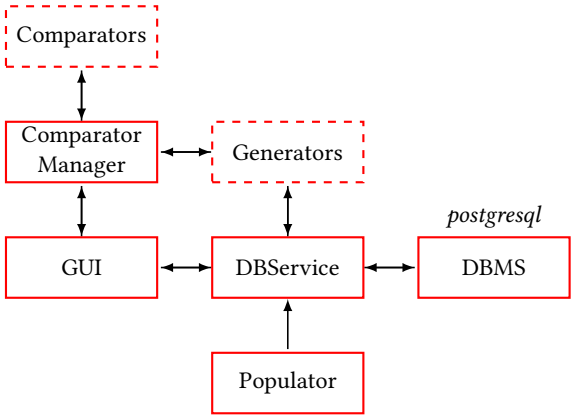


**Figure 4: Software architecture of MathMOuse**

several models more quickly than with the GUI in order to have a sufficient amount of models for testing.

All the services have been implemented in Java and the different services can interact with each other and exchange data, using a rabbitmq broker[2]. The services are managed and ran by docker[3].

The GUI service offers a graphical interface, to allow the user to exploit the Warehouse. Through the interface, it is possible to: (i) add new models, using XML files; (ii) select a time series (stored in a csv-formatted text file) to compare to the available models; (iii) visualize the progress and results of a comparison.

## 3.2 Insertion and Comparison Workflows

*3.2.1 Inserting a new model.* To add a model, the user selects an XML file containing the data about the equation to add. The file content is send to the DBService service. In the latter, the data are mapped into the target schema of the warehouse. Each element (nodes, inputs, variables and initial values) is given a unique key and inserted into the DBMS, using the JDBC library, to communicate with the DBMS.

*3.2.2 Comparison.* The user selects a text file containing the series raw data. The file content is sent to the Comparator Manager service, which will send a request (using rabbitmq) to the generators. The request will be distributed to one of the available generators (the rabbitmq broker uses a round robin algorithm to distribute the messages). The selected generator will request data to the DBService, which will send an SQL request to the DBMS to get all the equations available. The DBService will send the SQL query results back to the generator, which will be able to solve each equations, to create time series. The time series will be sent back to the comparator manager, which will distribute the comparisons work to the available comparators (round robin algorithm). The results of the comparisons are sent back to the GUI, in real time, so the user do not have to wait for all the comparisons to be over, to start consulting the results.

[2]https://www.rabbitmq.com/
[3]https://www.docker.com/

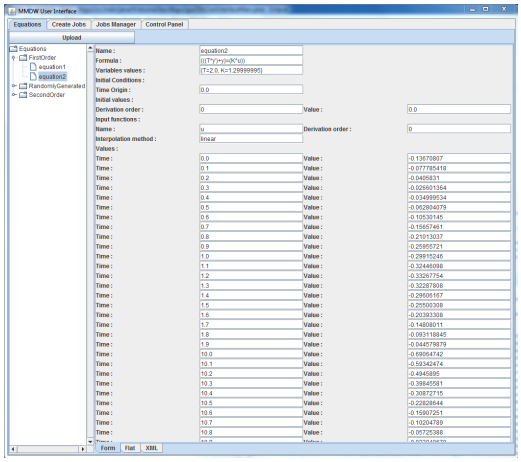

**Figure 5: Database Content visualization**

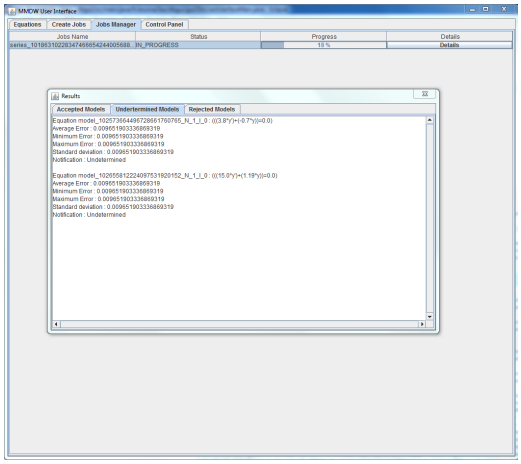

**Figure 6: Work in progress**

# 4 DEMONSTRATION AND USE CASE SCENARIO

Our demonstration scenario consists in two parts.

(1) An overview of the GUI, which is made of four tabs. The *Equations* tab (Fig. 5) is divided into a main view and a tree view on the left side. The navigation tree allows the user to navigate through the models available in the database. The models are sorted into different user-created groups. A click on an equation name in the tree triggers the printing of the details of the equation in the main view. The latter view is divided into three tabs, which gives different representations of the data. One of the tab gives the XML representation of the equation. In the *Create Jobs* tab, the user can select a time series text file, in order to prepare a comparison job request and start (or cancel) it. In the *Jobs Manager* tab (Fig. 6), the evolution of the job is represented with a progress bar and a `details` button opens a new window, in which the results are printed in real time. The *Control Panel* tab, displays the different services currently working and the number of the comparators and generators currently in use. It also displays the number of operations (series to generate or compare) waiting in each services. This has been added to allow the user to identify a potential bottleneck in the comparators or generators and add a comparator or a generator, if necessary.
(2) Then we will go through two use cases: (i) insertion of models and (ii) fitting models search.
(a) Insertion: Some ready-to-use XML files examples were meant to show the audience how to add a new model in the Warehouse, using the `Equations` tab of the GUI. Which will trigger an automatic update of the models list in the navigation tree, where the newly-added model will appear.
(b) Fitting models search (comparing time series and models): In order to demonstrate the *query by data* system, the data warehouse is initially filled with one hundred models and some time series are available in csv-formatted text files. The latter time series data are actual measurement data given by researchers of the automatic control field. Those data will be used to query the warehouse for a model fitting the data. The results of the query appear incrementally and the equations are automatically sorted into the `ModelAccepted`, `ModelRejected` and `Undetermined` tags (Fig. 6) and are displayed in a tabbed window (one tab for each tag).

## 5 CONCLUSION AND FUTURE WORK

The traditional trend was to store more and more data for analysis purposes. Usually, the analysis and understanding these data allow engineers and researchers to represent the behaviors behind the data by models. Persisting these models has become a necessity and contributes in moving from the data store to the model store. The need of development of comprehensive methods for sharing very large amount of models has been widely requested by researchers and engineers. Therefore, the general concept of Models Warehouse and the MathMOuse prototype represent a step further for this move.

Our current work consists in giving the possibility to parametrize comparison requests (select only a part of the equations, give the comparison method in parameter, give the generation method in parameter, override the initial conditions or input functions or even variables values), which could later lead to an extension of the SQL language, that implements those options. Further work will consist in validating the project with researchers from automatic control domain and a more distant perspective is the deployment of the project in a cloud architecture.

## REFERENCES


[1] J. Durech, V. Sidorin, and M. Kaasalainen. 2010. DAMIT: a database of asteroid models. *Astronomy and Astrophysics* 513 (2010), 13. Issue 8.
[2] P. Esling and C. Agón. 2012. Time-series data mining. *ACM Comput. Surv.* 45, 1, Article 12 (Dec. 2012), 34 pages. DOI: http://dx.doi.org/10.1145/2379776.2379788
[3] T.-C. Fu. 2011. A review on time series data mining. *Engineering Applications of Artificial Intelligence* 24, 1 (2011), 164–181.
[4] W. Giernacki, D. Horla, and T. Sadalla. 2016. Mathematical models database (MMD ver. 1.0). Non-commercial proposal for researchers. In *Proceedings of the 21st International Conference on Methods and Models in Automation and Robotics*. 555 – 558.
[5] B. Karwin. 2010. *SQL Antipattern : Avoiding the Pitfalls of Database Programming*. Pragmatic Bookshelf. 34 – 53 pages.
[6] A. Kasper, Z. Xue, and R. Dillmann. 2012. The KIT object database: An object model database for object recognition, localization and manipulation in service robotics. *International Journal of Robotics Research* 31 (2012), 927 – 934. Issue 8.
[7] W. Lang, M. Morse, and J.M. Patel. 2010. Dictionary-Based Compression for Long Time-Series Similarity. *IEEE Transactions on Knowledge and Data Engineering* 22, 11 (2010), 1609–1622.
[8] U. Pieper, B. Webb, D. Barkan, D. Schneidman-Duhovny, A. Schlessinger, H. Braberg, Z. Yang, E. Meng, E. Pettersen, C. Huang, R. Datta, P. Sampathkumar, M. Madhusudhan, K. Sjolander, T. Ferrin, S. Burley, and A. Sali. 2011. ModBase, a database of annotated comparative protein structure models, and associated resources. *Nucleic Acids Research* 39 (2011), 465 – 474.
[9] C. Ponchateau, L. Bellatreche, C. Ordonnez, and M. Baron. 2016. A Database Model for Time Series : From a traditional Data Warehouse to a Mathematical Models Warehouse. In *32ème Conférence sur la Gestion de Données - Principes, Technologies et Applications.*
[10] C. Pradal, C. Fournier, P. Valduriez, and S. Cohen-Boulakia. 2015. OpenAlea: Scientific Workflows Combining Data Analysis and Simulation. In *27th International Conference on Scientific and Statistical Database Management (SSDBM 2015)*. 6.
[11] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. 2002. *Numerical Recipes in C: The Art of Scientific Computing* (second ed.). Cambridge University Press.
[12] S. Wimalaratne, P. Grenon, H. Hermjakob, N. Le Novere, and C. Laibe. 2014. BioModels linked dataset. *BMC Systems Biology* 8 (2014), 9. Issue 91.