ME 599

Data Driven Modeling of Dynamical Systems

Comparison of different Data Driven approaches for Li-Ion Battery Modeling

Aman Tiwary (2076852), **Roshan Kaanth M K S** (2125039), **Sai Sourabh Burela** (2073436)

Advised by
Professor Krithika Manohar

University of Washington
Department of Mechanical Engineering

Contents

1. Abstract

This project is about comparing data driven approaches to model Lithium Ion batteries. Gray box modeling approaches like Genetic programming and Regression with Stochastic gradient descent were deployed and black box modeling techniques like Sparse Identification of Nonlinear Dynamics (SINDy) and Gaussian Process Regression were analyzed and compared. Due to lack of experimental test data, these models were validated using a Hammerstein Wiener System Identification model. The results showed that SINDy and Regression provide comparatively better results for the problem at hand.

2. Introduction

Data Driven modeling is the approach of analyzing and using data to derive insights about the system and make decisions. It is majorly helpful in predicting the characteristics of a system based on input-output response dataset. Data driven approaches will help us understand the system behavior & characteristics by correlating real time data with past patterns and will allow us to predict accurate insights of the dynamical system. With advances in technology and computing capabilities, the use of a variety of data driven methods can be seen in logistic applications, price prediction, understanding flow behavior , etc. Hybrid electric vehicles and electric vehicles in general are in increasing demand nowadays. Battery packs are an integral part of such systems. Like many other dynamical systems, the electrochemical reactions occurring inside batteries are governed by multiple high dimensional differential equations. This makes deriving a model of a battery a cumbersome process. The advent of data-driven approaches to modeling dynamical systems comes as a huge relief.

The goal of this project is model discovery of Li-ion battery cells using gray box and black box modeling approaches. We have used different data driven methods to predict the system characteristics of a Li-Ion battery. This approach was found to be interesting because accurate characteristic estimation and behavior modeling of batteries is crucial for electric vehicles and other applications where batteries are used extensively. A good battery model will help in satisfying multiple requirements of a battery management system like state-of-charge estimation, cell balancing, etc.

As mentioned before, there are many approaches to battery modeling such as Electrochemical modeling (purely physics based), equivalent circuit modeling (thevenin-circuit), analytical and impedance based modeling, empirical and semiempirical modeling.[9] Among these, Equivalent Circuit models are structurally simple and computationally efficient due to the use of lumped-parameter circuit elements, e.g., inductors, resistors, and capacitors, to represent the battery characteristics. So, in this project, we have decided to use equivalent circuit models to provide some intuition on the dynamics of the system and along with data, we have planned to generate a model of the battery.

In the past few decades, a lot of research has been going on battery parameter estimation using gray box modeling approaches. In [3], an estimation method is proposed which relies mostly on the relaxation portion of the battery response and involves some manual calibration. In [3], [8] and [9], system identification techniques are adopted for parameter estimation. [7] uses MathWorks”s Simulink Design Optimization toolbox which implements nonlinear least squares estimation technique and trust-region optimization method to obtain the parameters of the equivalent circuit model.

3. Approach

In this project, we are taking a data set of a Li-ion battery model which is obtained using a Pulse discharge experiment. The experiment involves applying a current discharge pulse for a certain period of time on a cell at regular intervals with a resting period in-between and measuring the cell voltage with a predefined sampling rate. The test is performed from a fully charged to completely drained battery state.

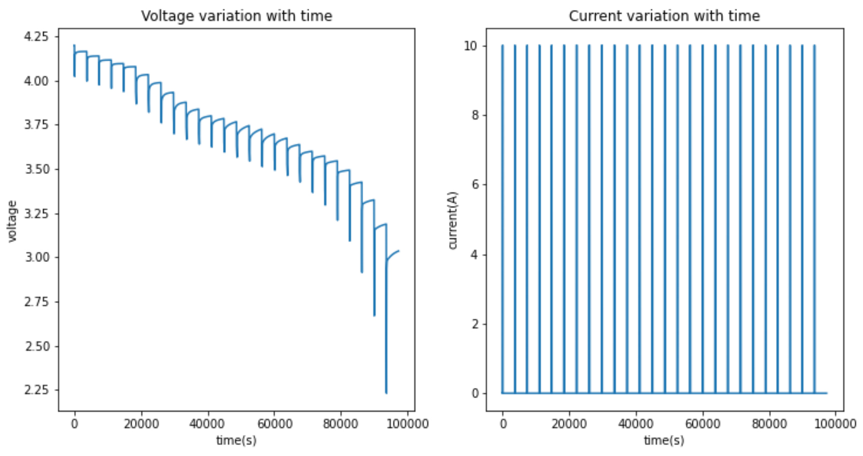


Figure 3.1: *Pulse Discharge Experiment Dataset*

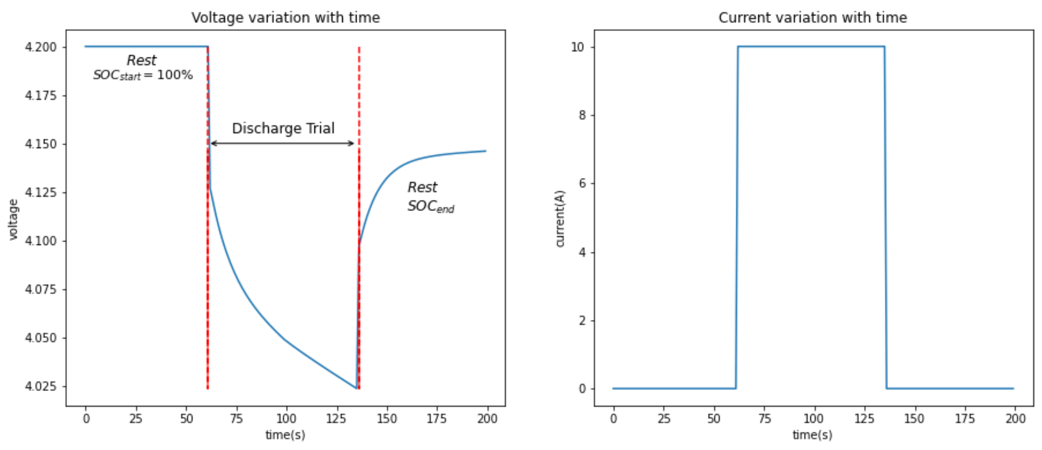


Figure 3.2: *One Discharge Cycle*

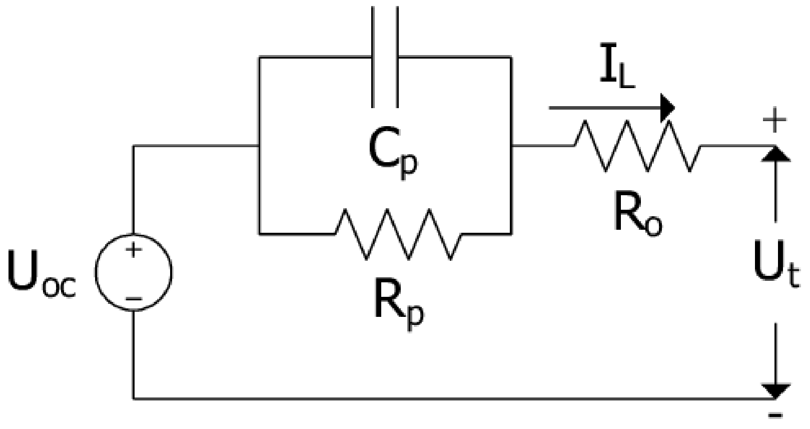


Figure 3. 3: *Common Thevenin's Equivalent circuit of Batteries*[10]

In this experimental dataset, we will apply different data driven methods like Gaussian Process Regression (GPR), Genetic Algorithm, Sparse Identification Of NonLinear Dynamics(SINDy) and Regression with Stochastic Gradient Descent to find the best fit model of system dynamics (system parameters like Resistance (R) and Capacitance(C)).

$$U_t = U_{oc} - I_L R_0 - (I_L R_p(1 - e^{(-t/(R_p C_p))}))$$

Where,

- U_{oc} = Open Circuit Voltage
- R_0, R_P = Resistance
- C_P = Capacitance

SINDy helps in finding the unknown dynamics of the system while the Genetic algorithm makes use of techniques derived from evolutionary biology to find the best possible parameters of the problem. GPR derives the perfect combination of dynamics of the system. The effectiveness of each of these methods in modeling the system will be comprehensively analyzed and presented. We plan to validate our models with the dataset generated using the System Identification toolbox in Matlab.

For the purpose of this project, we used different softwares, libraries & toolboxes as mentioned below:
Software: Python, MATLAB
Libraries & Toolboxes: Scikit-learn(Python), NumPy(Python), System Identification Toolbox (MATLAB), Pygad

4. Genetic Algorithm

4.1 Introduction :

A genetic algorithm is a search technique used in computation applications to find the true or most approximate solutions possible to optimization and search problems. Genetic algorithms are a particular part of evolutionary algorithms that make use of techniques such as mutation, selection, crossover and many more, inspired by concepts derived from evolutionary biology to find the 'fittest' solutions. Genetic algorithms are developed to understand the adaptive processes of natural occurring systems and are designed to create artificial systems to retain the robustness of natural systems. It provides an effective approach for optimization and machine learning applications when the search space is too large. Since this algorithm imitates the evolutionary dynamics, many of the Genetic algorithm's processes are random which can be controlled up to a desired level by the user. Genetic algorithms are very effective and far more capable than random and exhaustive search algorithms as no extra information is required. Using Genetic algorithms by making use of this advantage will help us find solutions to the problems that other optimization methods cannot compute. Genetic algorithms are used in various engineering and science problems ranging from design of turbine engines and antennas to design of control systems, networks, material selection etc. For example, in the present world, genetic algorithms are used by large MNCs to develop and optimize schedules and design products like aircraft components, micro-computer chips and medicines.

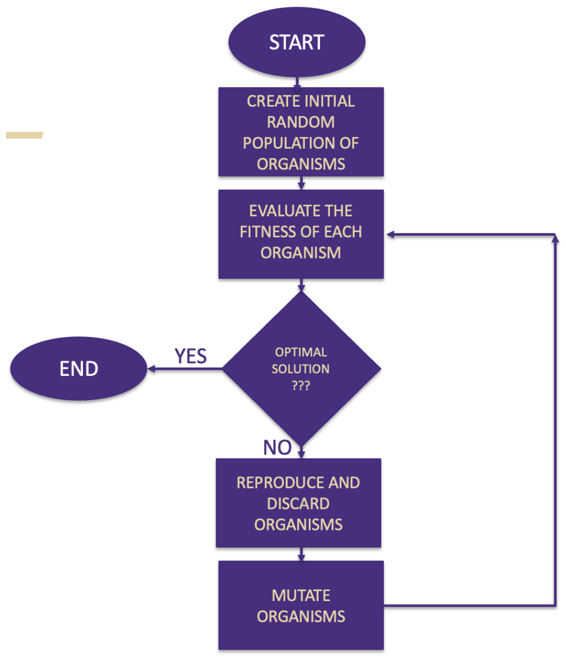

START
CREATE INITIAL RANDOM POPULATION OF ORGANISMS
EVALUATE THE FITNESS OF EACH ORGANISM
OPTIMAL SOLUTION ???
YES
END
NO
REPRODUCE AND DISCARD ORGANISMS
MUTATE ORGANISMS

Figure 4.1 *: Visual representation of the Genetic Algorithm*

4.2 Implementation:

Genetic algorithms are implemented by starting with an initial random population of possible solutions called *chromosomes* to optimize a problem in generations and make it evolve towards a better solution. The size of this population depends on the problem statement, but in general it contains several thousand combinations of possible solutions. For each generation, the fitness of each solution in the population is evaluated and multiple 'fittest' individuals are picked out from the current population ranked by their fitness score using a selection operator. Therefore, a more fit chromosome is more likely to get selected. There are certain selection methods that score the fitness of each solution and preferentially select the optimal solution.

These selected individuals are then made to recombine and probably mutate to form a new generation of population. Therefore, during each generation a suitable proportion of the existing population is selected to form a new generation. By producing this next generation of solution, often termed a "child" solution, this child solution shares many characteristics of the previous generation solutions. This process of producing new generations is beneficial in increasing the average fitness for the population because only the best population from the first generation are selected for breeding. This generation of population is done by using various operators like crossover and mutation. The crossover operator swaps the genetic information of parents that are selected randomly to reproduce an offspring. Whereas the mutation operator adds new genetic information to the child generation. Now this new population is utilized in the next iteration of the algorithm. The algorithm would terminate if the solution generated satisfies a particular fitness score that is specified by the user or if a maximum number of generations has been produced. If the algorithm terminates upon reaching a maximum number of generations, then the generation of a satisfactory solution is not guaranteed. In essence, as new generations are produced, the solutions evolve over time and the likelihood of producing optimal solutions is higher.

```
def Genetic_Algorithm ():
    initialize population
    evaluate the fitness of each individual in the population
    while optimal solution not found do:
        select fittest (best ranking) parents for reproduction
        perform mutation/crossover to breed new generation
        re-evaluate the fitness of the new population
    return (optimal solution)
```

Figure 4.2*: General Pseudocode of Genetic Algorithm*

Typically, a Genetic Algorithm requires two things to be defined, a genetic representation of the solution and a fitness function that evaluates the fitness of the solution domain. One standard way of representing the solution is an array consisting of bits. This conversion of the solution into a bit representation is preferred because genetic representation makes it convenient to facilitate crossover and mutation operations. The Fitness function is defined on the genetic representation, and it gives a measurement on the quality of the solution. The fitness function is always a problem dependent solution, which means it is always different depending on the application. In the entire algorithm, it is the only function which can determine how the chromosomes change over time, thus being a key factor in determining the optimal solution. In some cases, it is impossible to define the fitness function, and in such cases, we use interactive genetic algorithms.

4.3 Further Applications and study

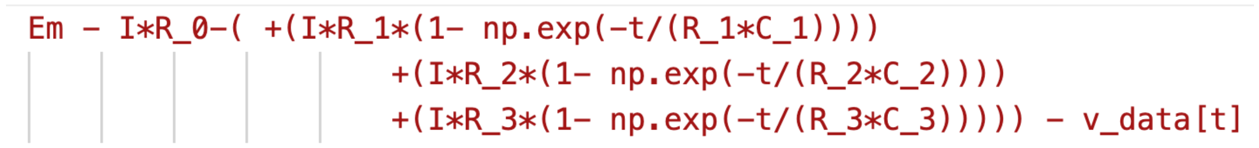
Genetic Algorithm has a huge impact in machine learning and Artificial Intelligence application because most of the existing methods are static and can usually solve one given specific problem since the purpose of their architecture being developed was for that specific problem. Thus, if a problem were to change due to some circumstance, then adapting to these changes would be difficult using these static methods. Therefore, Genetic Algorithms were developed to account for this variable change, as they are derived from natural biological evolution. The system architecture used to evaluate and employ these genetic algorithms are more likely to adapt to a wide range of problems and applications.

4.4 Limitations

Although Genetic algorithms are very efficient in optimization and search problems, the limitations of genetic algorithms lie in the vast search space that the algorithm must search through an infinite set of equations and values for finding the optimal solution. Therefore, they are not suitable for solving simple problems. Another limitation of genetic programming is the time factor. The time taken for the genetic algorithm combined with neural networks is slow and almost impractical. These difficulties can be overcome by employing parallel programming by computing and training the neural networks on GPUs. By employing parallel computation, the computation can be more time efficient for the same number of iterations thus increasing the chances of finding better solutions.

4.5 Genetic Algorithm to model the battery system

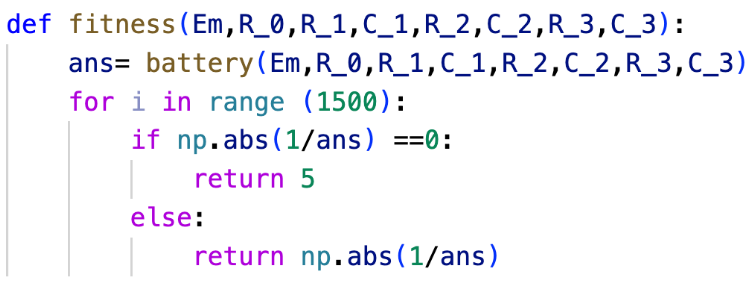
In the scope of this project, we have made use of Genetic Algorithm to estimate the best possible values of parameters like Resistance and Capacitance to model the Battery dynamics. The objective function is the model of the voltage which is varying with time. The objective function is defined as shown below in the figure.



```
Em - I*R_0-( +(I*R_1*(1- np.exp(-t/(R_1*C_1))))
                   +(I*R_2*(1- np.exp(-t/(R_2*C_2))))
                   +(I*R_3*(1- np.exp(-t/(R_3*C_3))))) - v_data[t]
```

Figure 4.3: *Objective function used in Genetic Algorithm Implementation*

Then, a set of random solutions are generated as the initial population. This set of initial populations include the possible solutions for each open circuit voltage, and the Resistance and Capacitance values for each branch. A fitness function is defined to evaluate the fitness of the random population of solutions. This function will help us determine if the solution is optimal or not. In context to battery modeling , the fitness function is defined as the reciprocal of the absolute value of the difference between the actual and predicted voltage. Using the initial population, the genetic algorithm is implemented and based on the fitness function, if the solution is optimal it will return the optimal solution, otherwise it will combine the “fittest” solutions from the initial population for producing a new generation. Now this new generation is again used in the genetic algorithm to find the optimal solution until the limit for the number of generations is reached or until the fitness function is satisfied.



```python
def fitness(Em,R_0,R_1,C_1,R_2,C_2,R_3,C_3):
    ans= battery(Em,R_0,R_1,C_1,R_2,C_2,R_3,C_3)
    for i in range (1500):
        if np.abs(1/ans) ==0:
            return 5
        else:
            return np.abs(1/ans)
```

Figure 4.4*: Fitness function used in Genetic Algorithm Implementation*

4.6 Results and Inference

Using the Genetic Algorithm, we have managed to achieve a good approximation of the true values based on the defined fitness function. As the number of iterations and producing of generations increases, the solution tends to only get better and closer to the actual value. This convergence is not guaranteed as we did not integrate the genetic algorithm with neural networks for training due to lack of computational resources. Computation of the genetic algorithm combined with neural networks is possible in this case but very slow as the algorithm has to iterate through a large search space and use the random solutions for training purposes.

Below are the results obtained from implementing the genetic algorithm to the Li-Ion battery model. It is observed that as the fitness function is tuned , and as each generation passes through, the solution of the open circuit voltage and other parameters like Resistance and Capacitance of each branch is nearing the optimal required solution. Even though this is not the best solution, it is the approximate solution for the used battery model.

Generations	Em	R_0	R_1	C_1	R_2	C_2	R_3	C_3
Gen 1	4.770678627369836	0.05955052140334083	0.6473050523210804	33406.625162218596	0.9588749944598962	49985.416406184995	0.33396407408635886	35402.53608826523
Gen 2	4.620768069624619	0.03679652682098744	0.011862573580116775	33204.213839972814	0.44712027855581926	46893.64464423981	0.04489211966800155	5766.384111062214
Gen 3	4.932742238887491	0.07373961300525733	0.6165897868183903	35084.23165627544	0.3157453937411122	29030.90667118855	0.4532605096883565	20598.379902368706
Gen 4	4.316645585994381	0.006256331398140413	0.22674582921542297	25160.122272132554	0.57134513662474	12466.244677172246	0.8437983259665447	26193.76329786244
Gen 5	4.365047815945865	0.01785430590539505	0.3577272752186711	33563.769424254744	0.28242026232550854	23399.553918226957	0.33871039628797284	29974.42257403998
Gen 6	4.274225643604449	0.012645311115497496	0.3161557300979221	33684.181670402235	0.5715385482453891	236067.0961815148	0.23816012988418533	42270.89305531984

***Table 4.1**: Obtained parameter values for each generation using Genetic Algorithm*

As new generations are created, the more approximate are the parameter values obtained. The below plot shows the convergence of the plot of the predicted values for each generation. From the below plot ,it is observed from the plot that the last produced generation has a better approximation of the voltage compared to the previous generations. This proves that as new generations are produced , the genetic algorithm provides better results and approximations and will eventually lead to the optimal solution.

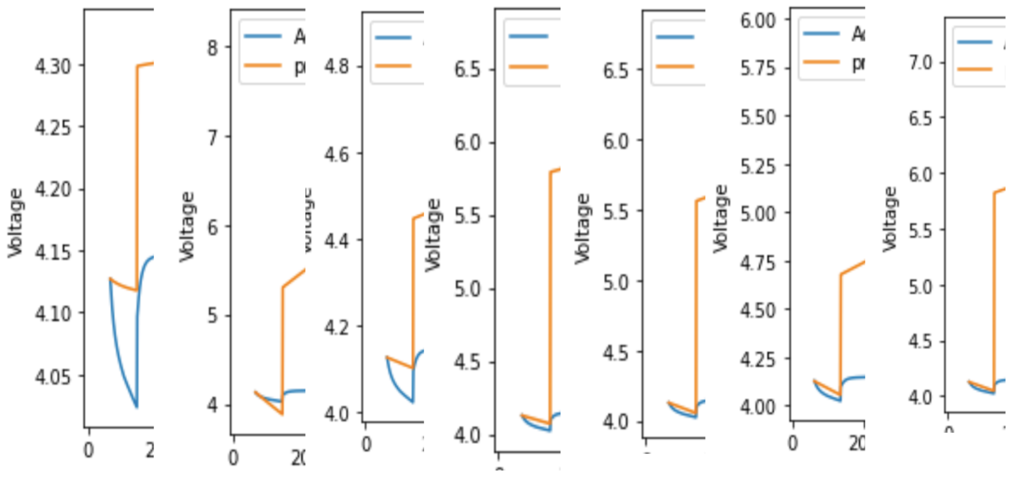
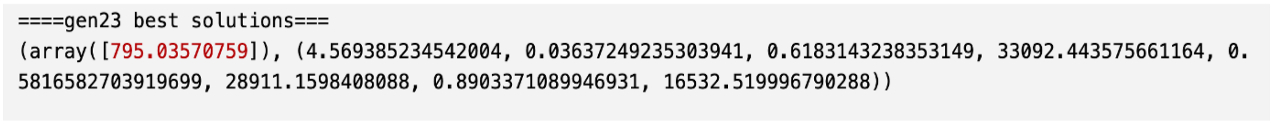


***Figure 4.5**: Obtained plots for each generation*

After multiple iterations and generations of solutions for obtaining parameters for the Li-Ion battery model , the twenty third generation (Generation 23) gives the best approximate results . The values of the parameters obtained by the 23rd generation are shown below.



```
====gen23 best solutions===
(array([795.03570759]), (4.569385234542004, 0.03637249235303941, 0.6183143238353149, 33092.443575661164, 0.
5816582703919699, 28911.1598408088, 0.8903371089946931, 16532.519996790288))
```

***Figure 4.6**: Parameter values obtained by Generation 23*

The plot of the predicted voltage using Genetic Algorithm is shown below :

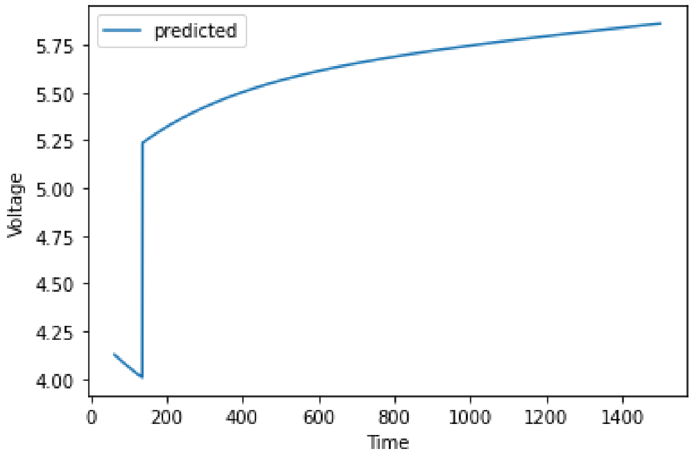


***Figure 4.7**: Characteristics of Predicted Voltage of the Li-Ion battery model using Genetic Algorithm*

Therefore , Genetic algorithm can be used to find the parameters of the Li-Ion battery model. To find a best fit optimal solution, interactive Genetic algorithm or a combination of the algorithm with Neural Networks is recommended. In the scope of this project, the algorithm has managed to produce optimal values of the parameters within the acceptable range with the characteristics of the predicted voltage vs time plot following the same behavior as the battery model data obtained from the pulse discharge method. The code for this implementation of the genetic algorithm is attached at the end of this document in the appendix.

5. SINDy

5.1 Introduction

Sparse Identification of Nonlinear Dynamics (SINDy) is one of the black box model discovery techniques for non-linear dynamical systems. Most modeling methods face great difficulty in identifying the structure and the parameters together from data as it is computationally heavy and challenging as there are many combinations of model structures to choose from.

SINDy algorithm bypasses this unmanageable combinatorial search problem, leveraging the fact that many dynamical systems

$$dx/dt = f(x) \qquad (5.1)$$

have dynamics f with only a few active terms in the space of possible right- hand side functions; it then approximates f by a generalized linear model

$$f(x) \approx \quad \Sigma^{p}_{k=1}\, \theta k(x)\xi k = \Theta(x)\xi, \qquad (5.2)$$

with the fewest non-zero terms in ξ as possible. It is then possible to solve for the relevant terms that are active in the dynamics using sparse regression that penalizes the number of terms in the dynamics and scales well to large problems.

First, time-series data is collected from and formed into a data matrix:

$$X = [\ x(t_1)\ x(t_2)\ \ldots.x(t_m)\ ]^T \qquad (5.3)$$

A similar matrix of derivatives is formed:

$$X' = [\ x'(t_1)\ x'(t_2)\ \ldots.x'(t_m)\ ]^T \qquad (5.4)$$

A library of candidate nonlinear functions Θ(X) can be constructed from the data in X:

$$\Theta(X) = [\ 1\ X\ X^2 \cdots X^d \cdots \sin(X) \cdots] \qquad (5.5)$$

Here, the matrix X^d denotes a matrix with column vectors given by all possible time-series of d-th degree polynomials in the state x. This library of candidate functions is one's own choice.

The dynamical system may now be represented in terms of the data matrices in (5.4) and (5.5) as

$$X' = \Theta(X)\ \Xi \tag{5.6}$$

5.2 Implementation

For the problem in hand, instead of equating a set of non-linear functions with its derivatives, we equate combinations of input features to our output variable. The voltage output from the battery pulse-discharge experiment is considered as the output whereas the current input is used to form our library of functions. We used Sequentially thresholded least squares for the regularization of parameters.

Here is the pseudo code of the algorithm,

- Define the current input and true voltage output
- Define the sample time interval
- Input the library of functions
- Call the function used to calculate the parameter vector
- Calculate the predicted voltage value using the parameter vector and library of functions
- Plot the true and predicted voltage vs time
- Calculate the mean squared error

5.3 Results and Inference

Here is the governing equation we obtained from the algorithm,

$$\textbf{Voltage} = w_0 + w_1 I(1-\exp^{-t/80}) + w_2 I(1-\exp^{-t/800}) + w_3 \exp^{-t/500} + w_4 \log(t)$$

Where w_0 , w_1 , w_2 , w_3 , w_4 are the parameters obtained from SINDy algorithm and t denotes sample time.

The values obtained are **$w_0 = 0.41$, $w_1 = 0.038$, $w_2 = 0.03$, $w_3 = 0.08$, $w_4 = -0.089$**

The mean squared error is 8.26e-05 sq volts

The following figure shows the model fit with true training data.

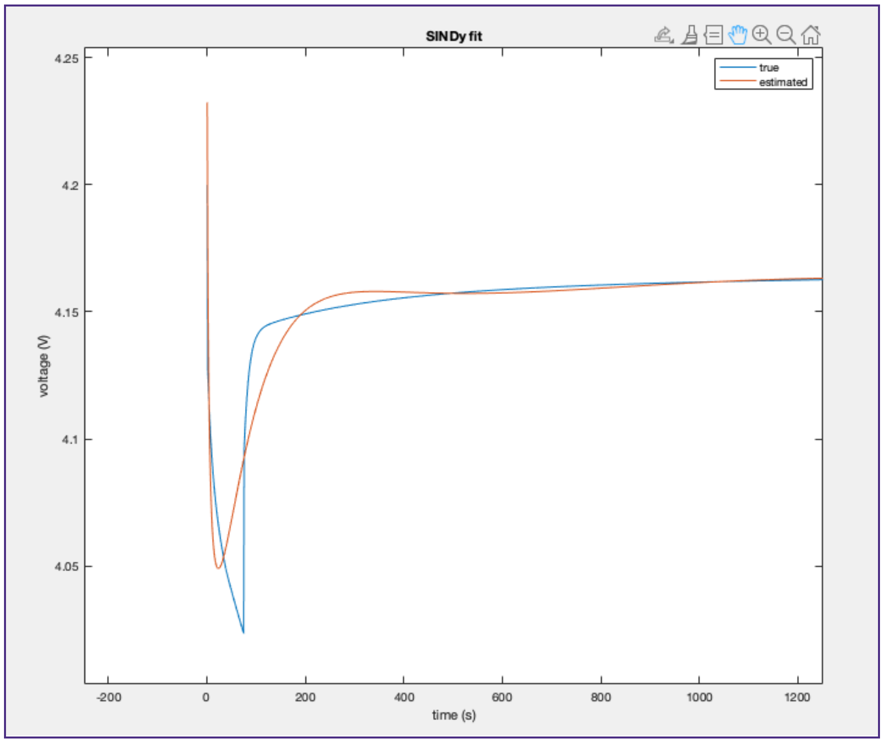


Figure 5.1: *SINDy algorithm training data fit (output voltage vs time) for pulse current input*

From the fit, we can see that the model struggles to fit the discharge part of the experiment. In our case, since the current input is a constant (being a pulse), we couldn't expand our nonlinear functions as the matrix becomes rank deficient. So, the combination of functions which produced the best fit, consists of exponential and logarithmic functions. Considering this fact, this fit seems reasonable. With a different kind of input (variable input), SINDy will have more scope to fit better.

5.4 SINDy Validation

Here is the fit of SINDy model with the test data.

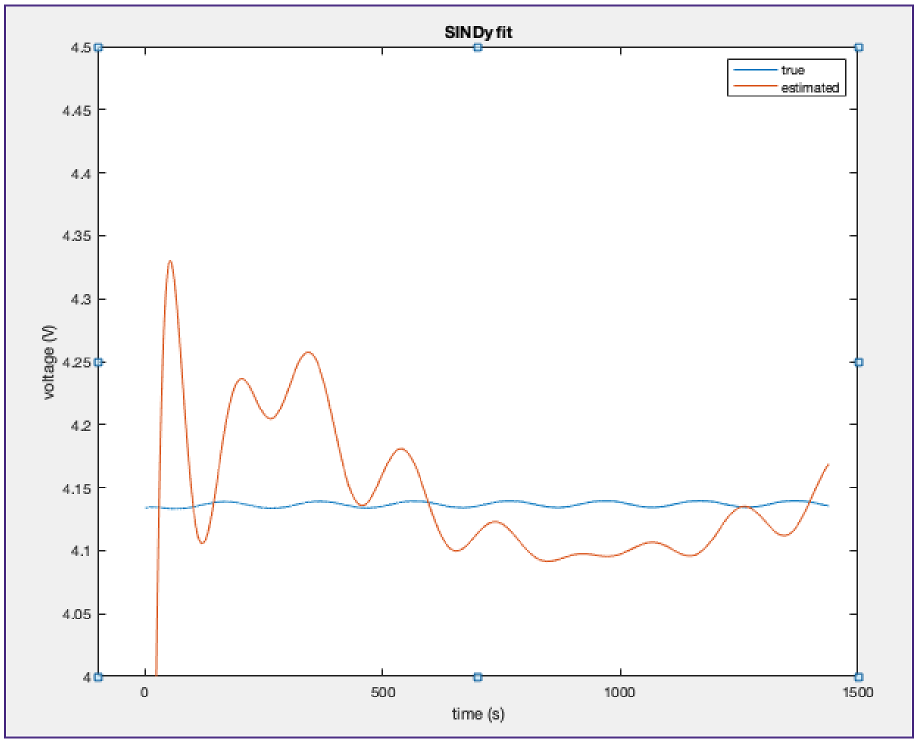


***Figure 5.2**: SINDy model validation*

The mean squared error is 0.0168 sq volts.

We can observe that the model doesn't give a good test fit, but tries to approximate the sinusoidal pattern of the true data.

6. Regression with Stochastic Gradient Descent

6.1 Introduction

Regression is one of the commonly sought out methods for model discovery. They are particularly useful in gray box modeling. We tried to use the governing equation obtained from Equivalent circuit modeling of batteries as explained in the introduction.

In this algorithm, we used Stochastic Gradient Descent which is an extension of Full gradient descent. In Full gradient descent, we compute the gradient of all data points to find the minima. However, this process is highly expensive for large data. Stochastic Gradient Descent picks random data points in the given range and we can constraint the choosing by specifying the type of probability distribution for random number generation. Also, we can use mini-batch SGD, which picks a bunch of random points.

For parameter estimation, we used the Levenberg-Marquardt algorithm which computes the hessian matrix in addition to the gradient. The Hessian (second-order derivative) improves the chances of descent to global minima in a better way than just a gradient descent.

Here is the basic equation of Levenberg Marquardt algorithm we used with SGD:

$$x_{n+1} = x_n - [\ \nabla^2 f(x_n) + \lambda * I\ ]-1 * \nabla f(x_n)$$

where

x_{n+1}	- vector of parameters at the next time step
x_n	- vector of parameters at the current time step
$\nabla^2 f(x_n)$	- hessian matrix
λ	- Learning rate
$\nabla f(x_n)$	- gradient

6.2 Implementation

Here is the pseudo code of the algorithm

- Define the governing equation, current input and true voltage output
- Define the cost function
- Define the mini-batch of random points for stochastic gradient descent algorithm
- Calculate the Hessian and gradient
- Execute the Levenberg-Marquardt algorithm
- Use the obtained parameters to calculate the predicted voltage output
- Plot the true and predicted output with time
- calculate the mean squared error

The original algorithm is coded in MATLAB.

6.3 Results and Inference

The following figure shows the model fit

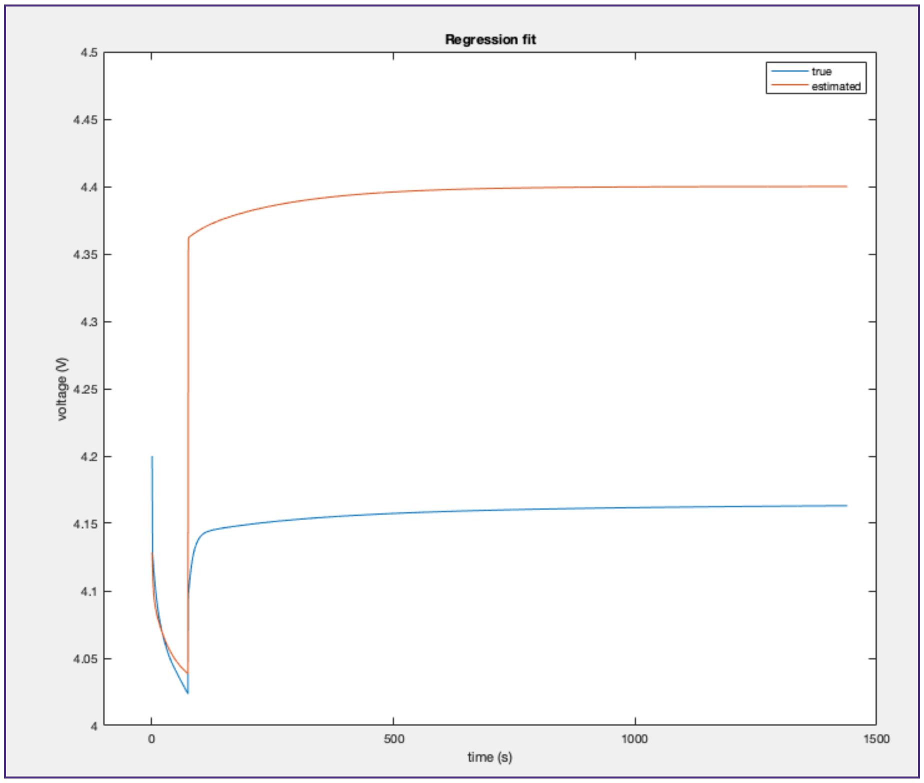


***Figure 6.1**: Regression result for voltage vs time training data*

The mean squared error is 0.0533 sq volts

We can see that we are not able to obtain a perfect fit with this model. This is because the fit largely depends on the initial guess of parameters. The wrong choice of initial guess leads to parameters getting stuck at local minima. Obtaining a global minima requires a different optimization method. But, to get close to a better minima, we made good initial assumptions of parameters using the physics of the battery system.

The initial guess of open-circuit voltage Em is obtained by taking the average value of the voltages at the beginning and end of the pulse output. The series resistance is obtained by dividing the voltage jump observed immediately after the current is removed with the current value. The time constants are obtained using a similar approach considering the voltage difference from the time when current is removed to the steady state voltage. This voltage difference is then divided by the current to obtain the total resistance. The RC resistances are then computed by subtracting the series resistance from the total resistance.

Here are the final parameter values obtained from the algorithm,

Open Circuit Voltage Em = 4.2 ; Series Resistance R_0 = 0.005 ; R_1 = 0.005 ; C_1 = 400 ; R_2 = 0.005 ; C_2 = 40000 ; R_3 = 0.005 ; C_3 = 6000

Hence the final governing equation becomes

$$\textbf{Voltage} = 4.2 - I*0.005 - I*0.005*(1-e^{-t/2}) - I*0.005*(1-e^{-t/200}) - I*0.005*(1-e^{-t/30})$$

6.4 Regression Validation

Here is the fit of the Regression model for the test data.

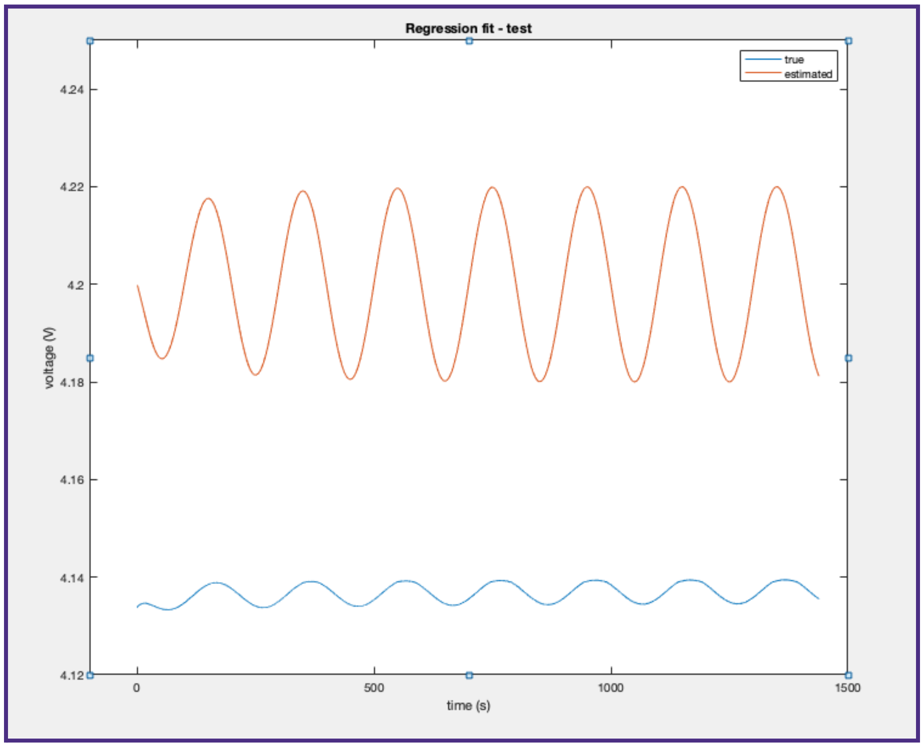


Figure 6.2: *Regression results for test data*

The mean squared error is 0.0041 sq volts.

We can see that the model closely follows the true pattern with an offset.

Validation Method

For this project, we couldn't obtain the test data. So we used MATLAB's System Identification toolbox to generate an 87% fit Hammerstein - Wiener Non-linear model to generate the test data using a sinusoidal current input of frequency 200 Hz.

7. Gaussian Process Regression

7.1 Introduction

Just like any regression algorithm, here you have some predictor variable x (input) and response variable y (observations), and the aim is to find a function that fits these points. There can be many functions that pass through those same set of points.

Gaussian Process Regression in a way is a distribution over a function. It is a powerful tool in machine learning that allows us to make predictions about our data by incorporating prior knowledge. The key assumption being that the function output with $n > 0$ inputs, $f = [f(x_1), \ldots., f(x_n)]$, is jointly gaussian with mean $\mu = m(x_1), \ldots, m(x_n)$ and covariance $\Sigma_{ij} = k(x_i, x_j)$, where μ is a mean function and $k(.\,,.)$ is a positive definite kernel. Since this holds, then for $n = m + 1$, with x_m training points and one test point x_*, we can infer information about $f(x_*)$ from the knowledge of $f(x_1), \ldots, f(x_m)$ by using the joint gaussian distribution $p(f(x_1), \ldots, f(x_m), f(x_*))$.

Let $D = \{(x_1, y_1), \ldots, (x_m, y_m)\} = (X, y)$. We presume that there is some structure in our data, that means $y_i \sim f(X_i)$. Gaussian Process model the output as a noisy function of the inputs given by $y_i = f(X_i) + \varepsilon$; where $\varepsilon \sim N(0, \sigma^2_n)$.

7.2 Implementation

We start by setting up our input data. We are considering the first discharge pulse as our dataset which starts at t = 0s and lasts till t = 1500s. But from t = 0s to t = 61s, the battery is just left ideal. Thus, this region is of no use as it won’t result in any fruitful results. So we update our region of interest from t = 62s to t = 1500s. This is where the battery is first discharged till t = 136s and then left alone while it recovers its charge.

We then sample 30 points with some bias added to sample points in the discharge region. These samples will be our observations/ training data that will be used to the function that best describes the battery behavior. Our test data is the whole dataset.

Now to find the covariance matrix, we have used the Radial basis function kernel(RBF) and Marten kernel. The RBF is parameterized by length scale parameterl and a signal variance parameter σ^2. Length scale affects the smoothness of the curves. Higher the length scale value, smoother will be the curve, more correlated are the neighboring points. Signal variance scales the functions that our GP prior can model in the output direction.

$$RBF\ Kernel = \sigma^2\, exp(-\frac{d(x, x')}{2l^2})\text{ , where}$$

$$d(x, x') = ||x - x'||^2$$

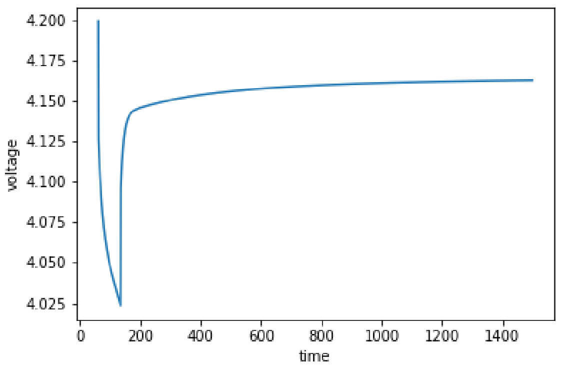


Figure 7.1*: updated data*

Matern kernel is a generalization of RBF kernel. It consists of an additional parameter v Which controls the smoothness of the resulting function. Smaller the value of v the less smooth the function. When $v \to \infty$, it becomes the RBF kernel. Here we have defined the kernel function for $v = 3/2$ and $v = 5/2$.

$$K_{v=3/2} = (1 + \frac{\sqrt{3}r}{l})\, exp(-\frac{\sqrt{3}r}{l})$$

$$K_{v=5/2} = (1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2})\, exp(-\frac{\sqrt{5}r}{l})$$

We now sample functions from prior. We have assumed the prior mean to be zero.

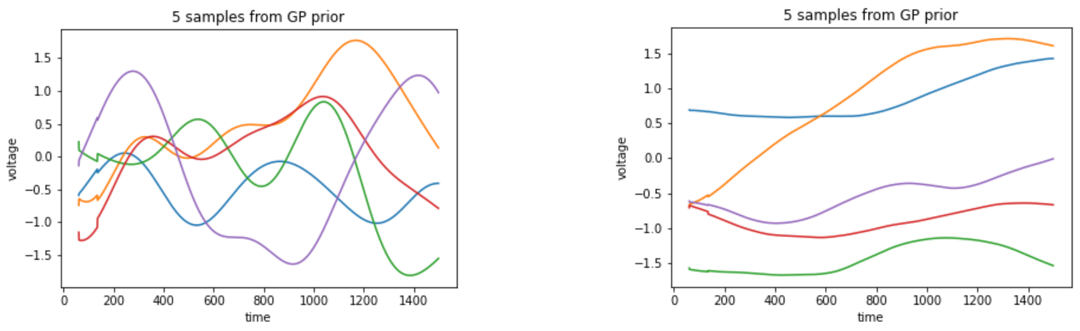


Figure 7.2*: (from l to r) Gaussian mean prior with RBF kernel, Gaussian mean prior with Matern kernel*

Algorithm:

$L = Cholesky(K + \sigma^2 I)$

$\alpha = L^T / L / y$

$\mu = K_*^T \alpha$

$b = L / K_*$

$\mathbb{V} = K_{**} - b^T b$

7.3 Results and Inference

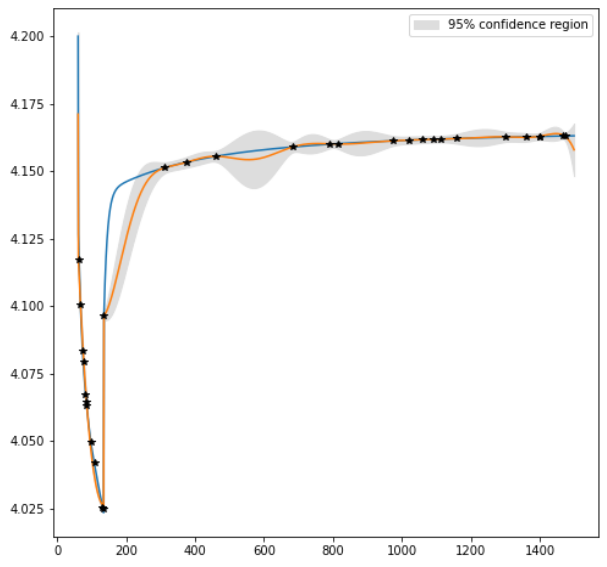
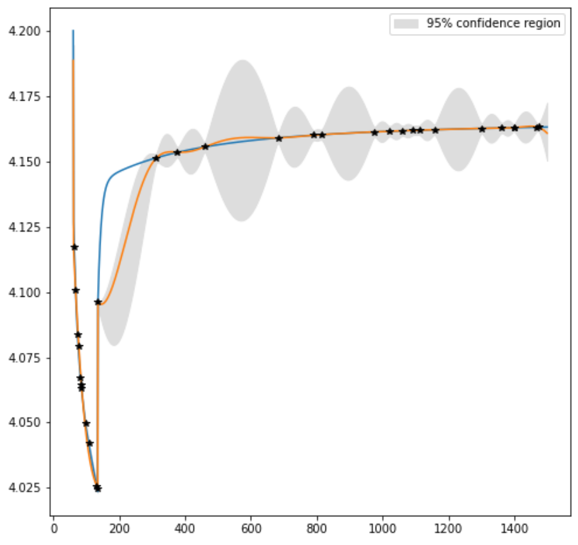


Figure 7.3: *prediction using RBF kernel*

The above plot shows the output by using the RBF kernel. For the above kernel, we get a mean squared error of **4.585e-05 sq volts** between the mean of all functions and the actual plot. As can be seen from the plot, it is able to closely follow the actual plot. Regions where the number of samples are very less have the most deviations, but the variance is still very small.
RBF Kernel Parameters: $l = 200, \sigma = 1$

Figure 7.4-Left shows the output by using Matren Kernel with $v = 1.5$. We can see from the plot that even though the mean of all the predicted functions closely follow the actual plot except from t = 200s to t = 400s. The region of 95% confidence gets wider. This indicates that there is some uncertainty at those points and wider the variance, lower is the probability that the predicted function will return the actual value. We get a mean squared error of **0.000141 sq volts** between the mean of all functions and the actual plot.
Matren Kernel Parameter: $l = 1500$

Figure 7.4-Right shows the output by using Matren Kernel with $v = 2.5$. We can see from the plot that even though the mean of all the predicted functions closely follow the actual plot. Even though the variance is better than what we got with $v = 1.5$, but the variance is still higher. We get a mean squared error of **9.3e-05 sq volts** between the mean of all functions and the actual plot. Matren Kernel Parameter: $l = 800$



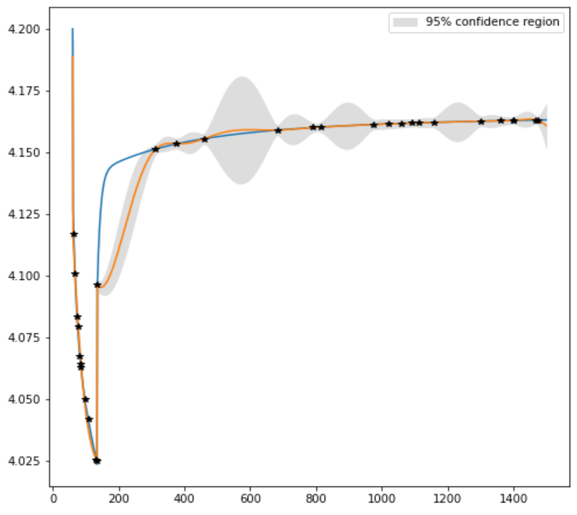


Figure 7.4:*(L to R) prediction using Matren Kernel (*$v = 1.5$*), prediction using Matren Kernel (*$v = 2.5$*)*

7.4 Validation

For validation of GPR models, we have used MATLAB's System Identification toolbox to generate an 87% fit Hammerstein - Wiener Non-linear model to generate the test data using a sinusoidal current input of frequency 200 Hz.

RBF kernel

Using the parameters of the kernel function that were used for prediction we get a mean squared error of 3.576e-06 sq volts. The trend of the mean of all functions is similar to the actual plot. Thus the RBF kernel was able to successfully predict the behavior of a different discharge trial of the battery.

Matren kernel $v = 1.5$

Using the parameters of the kernel function that were used for prediction we get a mean squared error of 3.521e-06 sq volts.

Matren kernel $v = 2.5$

Using the parameters of the kernel function that were used for prediction we get a mean squared error of 2.92e-06 sq volts.

Thus the RBF kernel is better for predicting battery voltage.

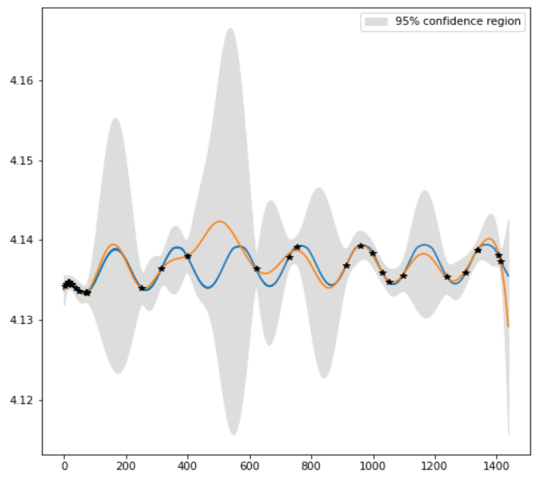


Figure 7.5: Validation with RBF kernel

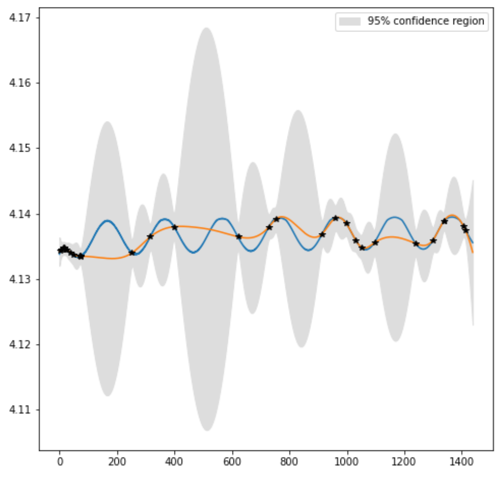


Figure 7.6: *Validation of Matren kernel* $v = 1.5$

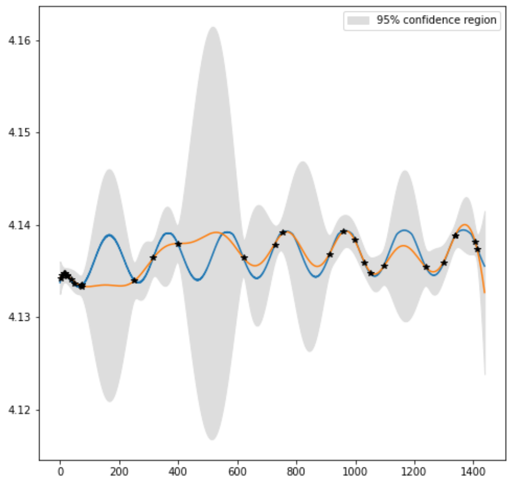


Figure 7.7: *Validation of Matren Kernel* $v = 2.5$

8. Conclusion

In the field of energy storage, the field of machine learning and data driven approaches have recently emerged as a promising modeling technique for determining important system parameters and characteristics like state of charge, state of health , and predicting the performance & useful life of the battery. With the increase in development of electric vehicles (EVs) , the need for behavior modeling of batteries is crucial to maintain a good battery management system . This project was focussed on using data driven approaches like Genetic algorithm, SINDy (Sparse Identification of nonlinear Dynamics) and Gaussian Process Regression (GPR) to find the best fit values of the system parameters like Resistance and Capacitance . In Genetic Algorithm, we made use of techniques derived from evolutionary algorithms which mimic the methods inspired by evolutionary biology such as mutation and crossover. Because of their population approach Genetic algorithms can be used to solve search and optimization problems efficiently. SINDy gives information about the system dynamics and is helpful to find the nonlinear dynamics of the system with a capability to handle large data sets. Though for the problem in hand, SINDy couldn't give best results because of the type of input, SINDy is a powerful algorithm for obtaining nonlinear dynamical system equations. Regression with Stochastic Gradient Descent is another powerful method for gray box model discovery as it is computationally effective, but it suffers from a local minima problem. Gaussian Process Regression gives a distribution over a set of functions. The prediction is probabilistic and one can compute empirical confidence intervals.

With the help of these Data driven approaches , we were able to find the best fit model of the battery system and predict the parameter values to understand the behavior of the system. As observed from the results , the best approximation of the model was given by SINDy and regression models.

9. Group Contributions

This project work is a combined effort from Aman Tiwary, Roshan Kaanth M K S and Sai Sourabh Burela. In the initial stages of the project , we did our research on battery modeling and the importance of having a robust machine learning model for battery management systems. Later as our study on the subject progressed, upon taking inputs from Professor Krithika Manohar, we decided to work on SINDy, Gaussian Process Regression and Genetic Algorithm for battery modeling. Aman Tiwary was involved in doing a thorough study on Gaussian Process regression and implementing this data driven method on the battery data to obtain the best fit model of the system. Aman also contributed to writing and formatting major parts of this report as per frameworks matching the guidelines of top research conferences. Roshan Kaanth M K S had taken up the task of going through the SINDy algorithm and implemented the same on the battery data to find the best fit of the system model. Roshan had also taken up the responsibility to include Regression analysis with Stochastic Gradient Descent to his work for getting further insights on methods for battery behavior prediction and analysis. Sai Sourabh Burela has contributed to this project by studying and understanding the Genetic Algorithm and implementing this algorithm to determine the best optimal parameters of the battery model. Sourabh has also taken responsibility to generate the validation data using a Hammerstein-Wiener System Identification model with the help of MATLAB and Simulink to validate the results from the above methods . This project thus involved equal efforts and important contributions from all the team members to obtain the anticipated results.

10. REFERENCES

1. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control- Steve L. Brunton, J. Nathan KutZ
2. Li-Ion Battery Pack Characterization and Equivalent Electrical Circuit Model Development – Ohuz H. Dagci and Ram Chadrasekaran
3. Genetic Algorithms for real parameter optimization- Alden H Wright
4. Data Driven Health Estimation and lifetime prediction of Lithium- Ion batteries- A review – Yi Li et al.
5. Multiobjective Optimization of Data-Driven Model for Lithium-Ion Battery SOH Estimation With Short-Term Feature – Lei Cai, Daniel Stroe et al
6. Multiobjective optimization using genetic algorithms- Abdullah Konak, David. W. Colt, Alice. E . Smith
7. Predicting the current and future states of battery using Data-Driven Machine learning- Man-Fai Ng, Jin Zhao, Qingyu Yan, Gareth J. Conduit, Zhi Wei Seh
8. High fidelity electrical model with thermal dependence for characterization and simulation of high power lithium battery cells- Tarun Huria, Massimo Ceraolo, Javier Gazzarri; Robyn Jackey
9. Function Optimization using Genetic Algorithms - Annie Brindle
10. System Identification and Estimation framework for pivotal automotive BMS Characteristics - Bharath Pattipati, Chaitanya Sankavaram, Krishna Pattipati.
11. Genetic Algorithm Optimization Problems - S.N Sivanandam, S. Deepa
12. System identification-based lead-acid battery online monitoring system for electric vehicles - Larry W. Juang; Phillip J. Kollmeyer; T. M. Jahns; R. D. Lorenz
13. On-line Measurement of Internal Resistance of Lithium Ion Battery for EV and Its Application Research - Hua Zhang, Rengui Lu, Chunbo Zhu and Yongping Zhao
14. Genetic Algorithms and Engineering Optimization- M. Gen, R. Cheng
15. An introduction to Genetic algorithms - Kalyanmoy Deb
16. Adaptation in natural and artificial systems - J.H Holland
17. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
18. Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Christopher K. I. Williams, The MIT Press, 2006. ISBN 0-262-18253-X.