1-2013

Comparison of a Particle Filter and Other State Estimation Methods for Prognostics of Lithium-Ion Batteries

Eric Alan Walker
University of South Carolina - Columbia

COMPARISON OF A PARTICLE FILTER AND OTHER STATE ESTIMATION METHODS FOR PROGNOSTICS OF LITHIUM-ION BATTERIES

by

Eric A. Walker

Bachelor of Science
Georgia Institute of Technology, 2009

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science in

Chemical Engineering

College of Engineering and Computing

University of South Carolina

2013

Accepted by:

Ralph E. White, Director of Thesis

Edward P. Gatzke, Reader

Sean C. Rayman, Reader

Gabriel A. Terejanu, Reader

John W. Weidner, Reader

Lacy Ford, Vice Provost and Dean of Graduate Studies

ACKNOWLEDGEMENTS

I would like to acknowledge Dr. White for his consistent leadership. I would like to acknowledge Dr. Rayman for the countless hours he has worked to help me. Thank you to research group members: Dr. Cai, Dr. Guo, Yiling, and former group member Saeed, now Dr. Rahimian. I would like to acknowledge friends, my roommate Bryan, Elina, and my parents.

ABSTRACT

A particle filter (PF) is shown to be more accurate than non-linear least squares (NLLS) and an unscented Kalman filter (UKF) for predicting the remaining useful life (RUL) and time until end of discharge voltage (EODV) of a Lithium-ion battery. The three algorithms track four states with correct initial guesses and 5% variation on the initial guesses. The more accurate prediction performance of PF over NLLS and UKF is reported for three Lithium-ion battery models: a data-driven empirical model, an equivalent circuit model, and a physics-based single particle (SP) model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

a state in He, et al model (Ah)

b state in He, et al model $\left(\frac{1}{cycles}\right)$

c state in He, et al model (Ah)

$c_{i,max}$ electrode maximum solid phase Lithium concentration $\left(\frac{mol}{m^3}\right)$ $i = n, p$

C Cholesky decomposition of P^x/ capacitance (F)

d state in He, et al model $\left(\frac{1}{cycles}\right)$

$D_{s,i}$ solid phase diffusion coefficient $i = n, p$

$f(\cdot)$ state dynamic model

F Faraday's constant $\left(\frac{C}{mol\, e^-}\right)$ in which the moles are of moles equivalent electrons

$g(\cdot)$ He, et al model

$h(\cdot)$ measurement model

I_{app} current (A)

J_i exchange current density for each electrode $\left(\frac{A}{m^2}\right)$ $i = n, p$

k cycle index $(cycles)$

k_{EUL} cycle index $(cycles)$ at end of useful life (EUL). $*$ indicates an experimental result.

P^x covariance matrix of the states

P^{xy} cross covariance matrix

P^{yy} measurement space covariance

Q process noise in unscented Kalman filter algorithm

Q_{EUL} capacity at the end of useful life, the defined failure threshold

Q_k capacity at cycle index k (Ah)

R cell resistance Ω

R_g gas constant $\left(\frac{J}{molK}\right)$

R_i particle radius in electrodes (m)

R_{ct} interfacial charge transfer resistance Ω

S_i^0 electroactive surface area (m^2) $i = n, p$

U_i^θ electrode open circuit potential (V), $i = p, n$

V_{cell} cell potential (V)

w_i weight of particle i

W matrix containing weights in unscented Kalman filter algorithm

$\underline{x}$ state vector estimate in particle filter algorithm

$\underline{\hat{x}}$ state vector estimate in unscented Kalman filter algorithm

$x_{i,avg}$ ratio of the solid bulk concentration to the maximum solid phase concentration of Lithium in the electrodes $i = n, p$

$x_{i,surf}$ ratio of the solid surface concentration to the maximum solid phase concentration of Lithium in the electrodes $i = n, p$

SOC_i state of charge $i = n, p, cell$

t time (s)

$\underline{u}$ input variables in unscented Kalman filter algorithm

v sensor noise, also called measurement noise

z measurements in particle filter algorithm

X matrix containing samples in unscented Kalman filter algorithm

$\underline{\theta}$ He, et al model state vector containing a, b, c, d

η_i overpotentials (V) $i = n, p$

List of Abbreviations

ECM ..... Equivalent Circuit Model

EKF ..... Extended Kalman Filter

EODV ..... End of Discharge Voltage

EUL ..... End of Useful Life

EV ..... Electric Vehicle

IG ..... Initial Guess

MCMC ..... Markov Chain Monte Carlo

NLLS ..... Non-Linear Least Squares

PDF ..... Probability Distribution Function

PF ..... Particle Filter

RUL ..... Remaining Useful Life

SOC ..... State of Charge

SOL ..... State of Life

SP ..... Single Particle (Model)

UAV ..... Unmanned Aerial Vehicle

UKF ..... Unscented Kalman Filter

CHAPTER 1

INTRODUCTION

This thesis originated from an assignment to reproduce the results of a research group at a separate university. My advisor was invited to speak at this other university and my advisor later gave me the assignment to reproduce an article which is referenced in this thesis and the corresponding journal article. This thesis is a reproduction or adaptation of a separately published article, in which I am the first author, which is permitted by the University of South Carolina.

In more plain language than the article, the contribution of this thesis is at least two-fold. First, this thesis compares the work of the original work it cites to other established methods to accomplish the same task. The application for this study can put its contribution in perspective. The application is predicting the Remaining Useful Life of a battery which can be inside a satellite or an electric vehicle or an unmanned aerial vehicle. Therefore there are different methods to go about making predictions of the Remaining Useful Life. The specific assumptions and setup for making predictions is included in detail. Although the original article was the first to use the method called the particle filter, this article makes a comparison to established methods in order to know whether the Particle Filter is preferred. The results and findings of this thesis are indeed it is.

The particle filter now has more evidence to support its use. Really the Particle Filter is a departure from established methods because it is a probabilistic method based upon Bayes' Formula. In my opinion its implementation is easier than established methods, which are typically deterministic. In my thesis defense, the superior accuracy of the particle filter was attributed to its use of prior knowledge during the question and answer session. Deterministic methods are provided with prior knowledge in the form of

an initial guess, but they tend to immediately ignore it once new data is available. Again this use of prior knowledge arises from, I believe, the Particle Filter's probabilistic nature.

The second contribution of this thesis is taking the Particle Filter and applying it with physics-based models of Li-ion batteries. The Particle Filter had not been previously explored with physics-based models of Li-ion batteries and I was at the advantage of having joined a research group specializing in physics-based models of Li-ion batteries. This thesis considers two models besides the model of the original article, and each additional model receives the same treatment as the first model by a Particle Filter and comparison with established deterministic methods. This geometrically increasing set of combinations led to a substantial amount of programming since each model has its own particularities. The effort paid off because the first use of a particle filter with a physics-based Li-ion battery model is reported now. The Particle Filter is adaptable to many physics-based models, and it is based upon a simple yet powerful equation and its prevalence will no doubt increase with increasing computational power.

The work of this thesis suggests future work, some of which is listed in the conclusion. One aspect of the future work is combining a charge model with a Remaining Useful Life model. The Remaining Useful Life model is like the one suggested by the original article referenced in this thesis. A charge model is the physics-based model which is newly reported in combination with the Particle Filter. The two types of models are distinguished by what quantity they are predicting and they cannot be compared like apples to apples. The charge model predicts the time until a battery needs recharging and a Remaining Useful Life model predicts how many times the battery may

be recharged until it has reached its end of useful life. Combining the two models means making a Remaining Useful Life Prediction and a charge prediction at the same time. The information from the RUL model adjusts and helps out the charge model. Likewise, the charge model can hand over intelligence, so to speak, to the Remaining Useful Life prediction. As the Remaining Useful Life model stands, in this thesis and in literature to the best of my knowledge, it waits until the point that the battery is recharged to update the RUL model. Combining the models allows updates to the Remaining Useful Life model with each new voltage measurement which are many and close between.

One comment which arose during my defense is the selection of the quantity of deviation in the likelihood equation in the Particle Filter method. The likelihood equation quantifies how likely a possible solution is based upon a data point. If the likelihood deviation is broad, then lots of solutions are considered likely and not much knowledge is gained from a data point. If the likelihood deviation is too small then one solution, no solution, or a small number of solutions are quantified as likely. This outcome is worse because it eliminates the diversity of solutions which are vital to the performance of the Particle Filter.

My hope is that this is a step in the ongoing advancement of Lithium-ion battery modeling knowledge. The work is a first of its kind in at least two ways which are comparison of the Particle Filter to established methods for making predictions of Remaining Useful Life and expanding the Particle Filter to physics-based Li-ion battery models. Suggested future work is combining a charge model and a Remaining Useful Life model and how to select the likelihood deviation in the Particle Filter.

CHAPTER 2

COMPARISON OF A PARTICLE FILTER AND OTHER STATE ESTIMATION METHODS FOR PROGNOSTICS OF LITHIUM-ION BATTERIES[†]

[†]Eric A. Walker, Sean Rayman and Ralph E. White. To be submitted to the *Journal of Power Sources*.

INTRODUCTION

Lithium-ion batteries are utilized in spacecraft, aircraft, and electric vehicles. An accurate prognosis for the remaining useful life (RUL) of Lithium-ion batteries and time until end of discharge voltage (EODV) is desired for these applications. RUL is the number of cycles remaining until the battery's capacity falls below a predetermined threshold, an event called end of useful life (EUL). Time until EODV is the time until the battery voltage drops below a defined EODV threshold. The particle filter (PF) is emerging as the preferred method for making these predictions about Lithium-ion batteries[1-5]. In this work, a Lithium-ion battery is assumed to undergo constant, low-current, complete discharge over cycling. RUL and time until EODV are predicted with the above assumption using three models and three methods. For comparison, accuracies for each method are reported.

Previous work on Lithium-ion battery prognostics with PF found PF accurate. However, among the works considered, comparisons to other methods for Lithium-ion battery prognostics were not made or a comparison made was to a less than optimal prediction method [1-5]. Further, physics-based models of Lithium-ion batteries were not incorporated into PF for prognostics. Physics-based models provide states with physical meaning in the Lithium-ion battery, and are built from first-principles. In this work, three models including a physics-based Lithium-ion battery model are implemented with PF and tested for accuracy in predicting RUL or time until EODV.

Predicting the RUL by a data-driven model of a Lithium-ion battery undergoing constant charge and discharge cycling was investigated by He, et al[1]. He, et al did not consider predicting time until EODV for a discharge cycle during cycling. The batteries were charged completely and discharged completely, for the cycling protocol. The failure threshold for the RUL was defined as the battery's capacity falling below 80% of its original capacity. The capacity at each cycle was measured by the integral of current over time. The capacity can be considered the size of a fuel tank for the case of comparing EV's to conventional vehicles. With this analogy the RUL is the number of times the tank or battery can be refilled with fuel or recharged until it can no longer hold a useful amount of fuel or charge. Although the size of a fuel tank does not generally change with the number of times it is refilled, a Lithium-ion battery loses capacity to the extent of losing usefulness[1,2,6].

In He, et al the capacity of a Lithium-ion battery was predicted forward in cycles until it dropped below the capacity at EUL, Q_{EUL}. When comparing prediction algorithms He, et al found PF to be more accurate than an extended Kalman filter (EKF) when predicting RUL. An underlying step in the EKF algorithm is using a Taylor series expansion to linearize non-linear model equations. This linearization approximation can be less than ideal if the problem is highly non-linear[7,8].

In this work, non-linear least squares (NLLS) and an unscented Kalman filter (UKF) are used for comparison, against a PF, for predicting RUL and time until EODV. NLLS is a state estimation algorithm for non-linear problems which attempts to minimize the sum of squared errors of a model and observations. UKF is a sampling-based Kalman filter and has been shown to perform better than EKF for non-linear systems[7,8]. UKF

chooses samples of the states by the state covariance matrix. The sampling scheme in this work is $2n + 1$ symmetric where n is the number of states. In the UKF algorithm, the samples are passed through model equations then re-evaluated for mean and variance without linearizing model equations. He, et al's model used a function with two exponentials for their model equation of battery capacity as a function of cycle number. Because the model equation was highly non-linear with respect to the states, EKF was susceptible to error due to the linear approximation of the model. In this work, UKF is compared against NLLS and PF with He, et al's model.

He, et al's model was data-driven and was not developed from first principles of the physics occurring within the Lithium-ion battery. In this work, He, et al's model is tested, along with an equivalent circuit model (ECM) and physics-based single particle (SP) model. ECM represents the battery as an electrical circuit with resisters, capacitors and other elements to create an equivalent circuit to model the battery behavior. SP is derived from first-principles of physics occurring within a Lithium-ion battery. Thus, the SP parameters have a physical interpretation.

Xing, et al[2] extended He, et al's work by testing an empirical second-order polynomial model for RUL predictions as well as He, et al's model using PF. They compared the two models and their results showed that He, et al's model predicted RUL more accurately than the polynomial model. Again, the new model was data-driven, and its parameters did not have a physical interpretation. Saha, et al[3] predicted the time until EODV of a battery undergoing discharge. The application for their work was predicting the flying time of an unmanned aerial vehicle (UAV). Without prognostics, the flying time of the UAV was usually reduced to provide extra margin to prevent the UAV battery

from becoming over-discharged during flight. Saha, et al applied PF with an empirical model in order to predict the time until EODV. The time until EODV was the difference of the predicted time of EODV minus the time of prediction. Predictions were made at multiple time points and the batteries were discharged until they experimentally reached EODV. The empirical model used with PF included terms representing the battery's open circuit potential, Ohmic drop, activation polarization, and concentration polarization. This model was based upon a high level of abstraction of the physics occurring within the cell[3]. No other method was compared against PF for testing prediction accuracy by Saha, et al although PF was supported due to the predictions meeting prognostics metrics for accuracy[3].

Jin, et al[4] used a data-driven model with PF for the application of predicting the residual life of Lithium-ion batteries in spacecraft. The residual life was the number of cycles until a failure threshold was reached, essentially the same as RUL. The model was not physics-based, and PF was not established as more accurate than other RUL prediction methods. Pattipati, et al[5] used a data-driven model to predict the RUL of a battery. Their model was a modified Randles equivalent circuit. In their model they considered other states besides predicted RUL such as the state of charge (SOC) of the battery. However, their model was not physics-based. Also, they required that the battery be taken offline for taking measurements.

Ramadesigan[9], et al predicted the capacity fade of a Lithium-ion battery due to aging using a power-fade law on six states of an electrochemical model. The states' uncertainty was quantified by a Markov Chain Monte Carlo (MCMC) algorithm. Their investigation supported the accuracy of MCMC state estimates. The states followed a

power law over cycling and the electrochemical model was simulated for the individual cycles. MCMC is similar to PF. Both can make non-Gaussian, numerical estimates of the states by approximating their probability distributions. Both are built upon Bayes' rule for updating state probability distributions.

Ramadesigan, et al's states had physical significance, and some insight into the source of capacity fade was suggested, since the negative anode solid phase diffusivity decreased by a statistically significant amount. Although Ramadesigan, et al utilized a rigorous physics-based model, the states themselves followed power-law models. This work incorporates the models themselves into a PF framework. Also, PF is compared for prediction accuracy of some failure for a variety of models and methods to test whether PF is the most accurate prognostics method or not. Ramadesigan's work emphasized an investigation of modeling capacity fade.

Outside of Lithium-ion battery applications, the PF has been used for prognostics. Daigle, et al[10] used a physics-based model of a centrifugal pump with PF for predicting the failure of a pump. The PF was able to use a physics-based model to make predictions about the pump, because the PF is generalizable to prognostics. Cadini, et al[12] used PF to predict the propagation of a crack in concrete. An, et al[6] provided a tutorial in MATLAB for prognostics using a PF. Their examples were crack growth and battery degradation, using empirical models.

Although not investigated for prognostics, physics-based models of Lithium-ion batteries have been investigated in literature. A physics-based, single particle (SP) model of a Lithium-ion battery was compared against an empirical model by Rahimian, et al[11]

for fitting cell voltage. The physics-based model performed better than the empirical model in accuracy for fitting cell voltage under low constant current conditions. The models in Rahimian, et al, have not been used for Lithium-ion battery prognostics. Their SP model and ECM model are re-applied for predicting time until EODV in this work. The empirical model, considered by Rahimian, et al, was an equivalent circuit model (ECM), first reported by Verbrugge[15], different from the equivalent circuit models of Saha, et al or Pattipatti, et al.

The SP in Rahimian et al's comparison was also used with Kalman filtering approaches in a separate work[7] for estimating the SOC of a Lithium-ion battery undergoing low-earth-orbit cycling. The SP model included some extra states for capacity fade effects. The comparison made in their work was between UKF and EKF, for use with the SP model. The unscented Kalman filter (UKF) was the preferred type of Kalman filter, in their work. Both the SP and ECM from Rahimian are applied to PF for predicting time until EODV. Also, PF is compared against UKF, not EKF, for tests of prediction accuracy.

This work compares, based on accuracy of predictions, PF with NLLS and UKF, for prognostics of Lithium-ion batteries. First, RUL is predicted for a Lithium-ion battery using the model of He, et al[1] with NLLS, UKF, and PF. Next, the ECM model of Verbrugge[15] with NLLS, UKF and PF predicts the time until EODV of a Lithium-ion battery. The predictions of RUL and time until EODV assume that the battery is undergoing low constant current, complete discharge. The last model used to compare the methods is the physics-based SP model, and NLLS, UKF and PF are compared for predicting the time until EODV.

1.1 HE, ET AL MODEL

1.1.1 Data set explanation and objective

From the state estimates of He, et al[1] a synthetic data set of capacity versus cycle was made by adding zero-mean, random normal noise with a standard deviation of $0.005\ (Ah)$ to capacity. This data set is presented in Figure 1.1. The objective is to predict the remaining useful life (RUL) of the battery, as the data becomes available. When EUL is reached (the first capacity measurement which falls below the EUL failure threshold) the predictions may be tested for accuracy, against the experimental result. The EUL capacity failure threshold is specifically,

$$Q_{EUL} = 0.8Q_{k=1} \tag{1}$$

$Q_{k=1}\ (Ah)$ is the capacity when k is equal to one, the first cycle. $Q_{EUL}\ (Ah)$ is the capacity at the end of useful life (EUL). Q_{EUL} is the horizontal line in Figure 1.1. The RUL is,

$$RUL = k_{EUL} - k \tag{2}$$

k_{EUL} is the cycle when the capacity of the battery decreases below Q_{EUL}. k is the cycle when the RUL prediction is made. The RUL is calculated as the difference of the predicted cycle of EUL and the cycle of prediction. In order to quantify accuracy, k_{EUL} is subtracted from k_{EUL}^{*}, where the asterisk denotes the experimental result.

1.1.2 He, et al model

The empirical model of He, et al[1] is

$$Q_k = g(\underline{\theta}, k) = a \cdot \exp(b \cdot k) + c \cdot \exp(d \cdot k) \tag{3}$$

where

$$\underline{\theta} = [a\ b\ c\ d]^T \tag{4}$$

$g(\cdot)$ is the model, k is the cycle index. $Q_k\ (Ah)$ is capacity at cycle index k, $a(Ah), b\left(\frac{1}{cycle}\right), c(Ah), d\left(\frac{1}{cycle}\right)$ are states, and $\underline{\theta}$ is the state vector (one underbar, _, denotes a vector). Once $\underline{\theta}$ is estimated, k_{EUL} may be solved for by,

$$a \cdot \exp(b \cdot k_{EUL}) + c \cdot \exp(d \cdot k_{EUL}) = 0.8Q_{k=1} \tag{5}$$

k_{EUL} is subtracted from k_{EUL}^*, and the smaller number of cycles indicates a more accurate prediction of RUL.

1.1.3 He, et al model non-linear least squares results

The first method to make predictions of RUL with the He, et al model is non-linear least squares (NLLS). NLLS, which was used by Rahimian, et al[11] for comparing SP to ECM, estimates $\underline{\theta}$. With this estimate a prediction of k_{EUL} may be made. White and Subramanian[12] explicitly provide a non-linear least squares (NLLS) algorithm for state estimation for a general case. The first battery data set is synthesized from He, et al's states and displayed in Figure 1.1. A second battery data set from the same states with different random noise is used for NLLS state estimation. For the first test of NLLS prediction, three states are given the exact parameters, and the c state is estimated by NLLS with 50 data. The c state is given the correct initial guess (IG). The IG's from He, et al are displayed in Table 1.1. The result is NLLS makes a very accurate prediction. The second test is using NLLS to track the c and d state. The model is non-linear in the d state which makes the estimation problem non-linear unlike tracking the single state c.

The constraints are such that c is positive and d is negative. The fit for tracking two states and predicting RUL is nearly on top of the fit of the c state. If the other states are the fixed, correct values, NLLS can make an accurate RUL prediction when tracking c and d or only c.

The next test is tracking the four states in the model. None of the states are fixed. Correct IG's are supplied to NLLS. Table 1.1 includes the parameters when four states are estimated by NLLS. a and d are constrained negative, and b and c are constrained positive which correspond to the correct signs. The results are in Figure 1.2. Unlike tracking one or two states, tracking four states gives an inaccurate prediction of 88 cycles. Without the knee (downward bend) in the later data, NLLS makes an inaccurate prediction when tracking four states. The prediction does not show a knee. When NLLS is constrained to 5% of the correct four states, the prediction is more accurate than without constraints. With 5% constraint the error is 10 cycles.

1.1.4 Unscented Kalman filter

Rahimian, et al[11] applied UKF and EKF with SP for estimating SOC and state of life (SOL) for a Lithium-ion battery undergoing low-earth-orbit cycling. SOL is a measure of the aging of the Lithium-ion battery. Rahimian, et al found UKF to be more accurate than EKF based upon fitting voltage measurements. Plett[8] provides an explanation based upon the assumptions taken by EKF and UKF of why UKF is more accurate for non-linear model problems. Plett's explanation is, when calculating the mean of a random variable,

$$E[fn(x)] \approx fn(E[x]) \tag{6}$$

where $fn(\cdot)$ is a non-linear model equation. The mean of the non-linear function of the random variable, x, is only approximately equal to the function of the mean of x. This is caused because $fn(\cdot)$ is linearized by a first-order Taylor series. Likewise, covariance matrices can be inaccurate when passed through a linearized version of the model equation.

UKF does not change model equations. Instead, deterministic samples are taken, and the samples are propagated through the non-linear model equations. After being propagated through, the samples provide a mean and covariance. The sampling scheme used in this work is $2n + 1$ symmetric[13] where n is the number of states. During the update step of Kalman filtering the states are changed by the experiment measurement. The states are assumed to be Gaussian distributed. The UKF algorithm equations are presented in the Appendix.

Four states in He, et al's model are tracked by UKF. 50 data points are used to track the states. NLLS made an inaccurate 88 cycle RUL prediction with 50 data points when tracking four states. P^x, the state covariance matrix is initialized by diagonalizing the difference of the upper and lower bounds of the 95% confidence interval of He, et al's states and squaring the difference. The Cholesky decomposition takes the square roots of the diagonal elements of a matrix. The diagonal elements of the Cholesky decomposition are used to make samples. Table 1.1 lists the four state results from prediction alongside the correct states, and c is the only state to change by a noticeable amount. Figure 1.3 displays the prediction results of UKF which are less than accurate with an error of 19 cycles. The prediction results are more accurate than NLLS for tracking four states but

less accurate than when NLLS is constrained by 5% to the correct states which made an error of 10 cycles.

1.1.5 Particle filter

PF makes stochastic, meaning random, estimates of states and adjusts their weights from observable measurements. The method of particle filtering does not begin with the assumption that there is a single, best prediction. On the other hand, PF assumes that there are many possible predictions, and each prediction is associated with a weight, or probability. PF is built upon Bayesian statistics, which is a paradigm shift from deterministic methods. It parts from the notion that nature is deterministic and predictable[17].

PF makes a probability density function (PDF) approximation of capacity in the case of He, et al for a PDF of k_{EUL}. Particles approximate a probability distribution of the state vector, $\underline{x}$, which can be transferred to k_{EUL}, from the model. For He, et al's model $\underline{x}$ is $\underline{\theta}$. PF begins with a model in state-space form.

$$x_k = f_{k-1}(\underline{x}_{k-1}) \tag{7}$$

$$z_k = h_k(\underline{x}_k) + v_k \tag{8}$$

k is an integer which is a discrete index e.g. cycle index for the He, et al model. $f(\cdot)$ is the dynamic model of how the states change with k and h_k is the measurement model which predicts the measurement based on the values of states at k. In He, et al's model, Q_k is the measurement calculated from the states. v_k is the measurement noise.

The particle filter estimates the state vector, $\underline{x}$, by a probability distribution, $p(\underline{x}_k|\underline{z}_k)$. $p(\underline{x}_k|\underline{z}_k)$ gives the probabilities for a domain of possible values of the true state given all the observations at index k and prior information from training data. PF approximates the probability distribution of $\underline{x}_k$ by a series of weighted particles. $\underline{x}_k^i$ represents one estimate of the states, $\underline{x}_k$, and w_k^i is the associated probability of the estimate. Together, $\underline{x}_k^i$ and w_k^i make one particle. N is the total number of particles. In this work, particles are uniformly varied around the correct or best IG's. All of the particles together make an approximation of the probability distribution of $\underline{x}_k$. The weights are such that they sum to one.

$$\sum_i^N w_k^i = 1 \tag{9}$$

The particles approximate the distribution of $\underline{x}_k$ by

$$p(\underline{x}_k|Z_k) \approx \sum_{i=1}^N w_k^i \delta(\underline{x}_k - \underline{x}_k^i) \tag{10}$$

$\delta(\cdot)$ is the Dirac delta function. Equation (10) is a convention for particles making a probability distribution. A simpler equation to express how particles make a distribution is

$$p(\underline{x}_k^i|Z_k) = w_k^i \tag{11}$$

Equation (11) is interpreted as the probability of any particle, $\underline{x}_k^i$, is its weight.

As k advances, the posterior PDF becomes the prior PDF of the new k with the dynamic model, $f(\cdot)$. $p(\underline{x}_k|Z_k)$ is a posterior PDF $p(\underline{x}_k|Z_{k-1})$ is a prior PDF.

$$p(\underline{x}_k|Z_{k-1}) \approx \sum_{i=1}^{N} w_{k-1}^i f_{k-1}(\underline{x}_{k-1}^i) \quad (12)$$

Equation (12) generates the prior distribution of the parameters, $\underline{x}_k$. Once the prior distribution of $\underline{x}_k$ is available by Equation (12) the posterior distribution of $\underline{x}_k$ is obtained by[1,3,12,19]

$$w_k^i = w_{k-1}^i \, p(z_k|\underline{x}_k^i) \quad (13)$$

$p(z_k|\underline{x}_k^i)$ is the likelihood of $\underline{x}_k^i$. The likelihood is calculated by[6,17]

$$p(z_k|\underline{x}_k^i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\frac{\left[z_k - h_k(\underline{x}_k^i)\right]^2}{\sigma^2}\right] \quad (14)$$

σ is the standard deviation of the measurement noise, v_k. Equation (14) requires the assumption that measurement noise is Gaussian distributed with mean zero. From Equation (14), the closer $h_k(x_k^i)$ is to z_k the higher $p(z_k|\underline{x}_k^i)$ is and the higher the likelihood is for $\underline{x}_k^i$. The result of Equation (14) plugs into Equation (13), and w_k^i is obtained. In order to satisfy that the weights sum to one (Equation (11)), normalization is performed[1,12,14].

$$\overline{w}_k^i = \frac{w_k^i}{\sum_{j=1}^{N} w_k^j} \quad (15)$$

The overbar in Equation (15) indicates a normalized weight. Essentially, several possible particles of different states are generated. They can be visualized as curves with units of capacity in He, et al's model. These particles are more or less likely, from Equation (14), based upon the distance from experimental measurements. Unlike NLLS

and UKF, the PF particle state estimates do not change in this work. The weights of the particles change but not the state estimates of the particles.

Figure 1.4 displays PF results for He, et al's model. The grey lines are particle estimates of the states shaded by their weights. Figure 1.4 displays the PF results with particles of five percent uniform variation above and below the correct states. The prediction is more accurate than NLLS or UKF tracking four states. The number of particles is 100. Although one prediction PDF is shown the particle shades shift due to two later predictions. The figure reveals some effects of PF. Particles away from the data are shaded less likely, which becomes more pronounced in later predictions. The weights at the edge of the prediction PDF jump up and down. Figure 1.5 zooms in on the prediction PDF's. A spread of particles appears beginning around cycle 165, which is packed together in earlier cycles. Therefore, the algorithm didn't differentiate the likelihoods in the bundle of particles, which is why the likelihood jumps around where those particles cross the failure threshold. In this work the weighted sum of the particles is used to report prediction accuracy of PF because a choice of the most likely particle isn't as meaningful due to the jumping around of weights. The weighted sum of particles gives a prediction error of magnitude 0.82 cycles with He, et al's model. PF makes the most accurate prediction among the three algorithms considered when tracking all four model states and given correct IG's.

1.2 EQUIVALENT CIRCUIT MODEL

1.2.1 Data set explanation and objective

The data for the ECM and SP time until EODV predictions are from the NASA Ames prognostics data repository[16]. Batteries are discharged at a constant current of $2\ (A)$ from a fully charged $4.2\ (V)$ to a EODV cutoff voltage of $2.5\ (V)$. Assuming constant current throughout discharge, the objective is to predict the time until the EODV. Figure 1.6 displays the data set for time until EODV prediction.

The RUL predictions used the capacity of a discharge as one measurement. The capacity results from many discharges of a battery undergoing charging and discharge cycling were displayed in Figures 1.1-4. A single discharge is displayed, per plot, for visualization of prediction of time until EODV for the remaining figures. For a complete cycle, after EODV, the battery was charged to a fully charged $4.2\ (V)$. The discharge plots have axes with units of voltage versus time, instead of capacity versus cycle.

1.2.2 Equivalent circuit model

Verbrugge's ECM is considered for a battery discharge cycle[11,15]. The data assumes a constant current discharge, so the cell potential, $V_{cell}(V)$ can be related to the current, $I_{app}(A)$ by the following equation, according to the ECM model.

$$V_{cell}(t, I_{app}, R, Q, C, R_{ct}) = V_o + I_{app}R + \frac{Q}{C}\exp\left(-\frac{t}{R_{ct}C}\right) + I_{app}R_{ct}\left(1 - \exp\left(-\frac{t}{R_{ct}C}\right)\right) \tag{16}$$

Unlike He, et al's model, $V_{cell}\ (V)$ is the measurement. The states are $R\ [\Omega]$, the cell resistance, $Q\ (Ah)$ the capacity of the cell, $C(F)$ the capacitance, and $R_{ct}\ [\Omega]$ the interfacial charge transfer resistance. $V_o\ (V)$ is the open circuit potential of the cell, defined in Equation (17).

$$V_o = U_p^{\theta}(SOC_p) - U_n^{\theta}(SOC_n) \tag{17}$$

$U_i^{\theta}\ (V)$ are open circuit potentials of the cathode and anode, respectively[20]. SOC_i are the states of charge of the electrodes.

$$U_p^{\theta}(SOC_p) = \frac{(-4.656+88.669SOC_p^2-401.119SOC_p^4+342.909SOC_p^6-462.471SOC_p^8+433.434SOC_p^{10})}{-1+18.933SOC_p^2-79.532SOC_p^4+37.311SOC_p^6-73.083SOC_p^8+95.96SOC_p^{10}} \tag{18a}$$

$$U_n^{\theta}(SOC_n) = .7222 + .1387SOC_n + 0.29SOC_n^{0.5} - \frac{.0172}{SOC_n} + \frac{.0019}{SOC_n^{1.5}} + \cdots$$

$$+.2808e^{(0.9-15SOC_n)} - .7984e^{(0.4465SOC_n-0.4108)} \tag{18b}$$

SOC_n and SOC_p are obtained by a linear interpolation of SOC_{cell}.

$$SOC_n = 0.79SOC_{cell} + 0.01 \qquad SOC_p = 0.97 - 0.51SOC_{cell} \tag{19}$$

SOC_{cell} is governed by

$$SOC_{cell} = SOC_{0,cell} + \frac{I_{app}}{3600Q}t \tag{20}$$

$t\ (s)$ is time. Figure 1.7 displays the equivalent electric circuit representation of the battery, which is the origin of Equation (16) of the ECM model. The model terms representing battery effects are treated as resistors or capacitors in an electric circuit. ECM is a widespread type of battery model[9].

1.2.3 Equivalent circuit model results

Table 1.2 displays the training estimates from data. NLLS and ECM states from 50 data are included for comparison. All four states are tracked. Figures 1.8 and 1.9 display the prediction results of NLLS and UKF. With correct IG's NLLS makes an inaccurate

prediction unless NLLS is constrained to 5% of the correct IG's. UKF's prediction is inaccurate, but more accurate than NLLS unless NLLS is constrained to 5% of the correct IG's. In both methods, all four ECM states are tracked. NLLS 5% constrained has an error of 396 seconds. The state covariance matrix, P^x, of UKF is created by diagonalizing 5% of the best ECM states from training data and squaring the matrix. Figure 1.10 displays the PF prediction at the same time as NLLS and UKF. Although one prediction PDF is shown the particle shades shift due to two later predictions. The correct four states are uniformly varied by 5%. The weights move up and down because the state particles are bundled together over the data until after the last prediction is made. The weighted sum of particles gives an error of 25.62 seconds. The prediction of PF at 928.6 (s) is more accurate than the predictions of NLLS and UKF at that time when tracking all four model states with best IG's.

1.3 SINGLE PARTICLE MODEL

1.3.1 Single Particle model

The final model considered in this work is the single particle model (SP). SP is used for time until EODV prediction. SP makes simplifying assumptions from more rigorous physics-based models[7,11]. For the low constant-current discharging considered in this work, the assumptions of SP are met. SP considers two electrodes to contain spherically symmetric particles of solid active material. Lithium intercalates and de-intercalates at the surface of the solid active material and diffuses. By treating the concentration profile inside the solid active material spherical particles as a two-term polynomial, volume averaging techniques create an average concentration for the bulk of the particle.

Reproduced from Rahimian, et al, SP equations are as follows[11]. Capacity fade effects are removed. In the cathode $i = p$ and in the anode $i = n$.

$$J_p = \frac{I_{app}}{S_n} \tag{21}$$

$$J_n = -\frac{I_{app}}{S_n} \tag{22}$$

$$\frac{dx_{i,avg}}{dt} = -\frac{3J_i}{FR_i c_{i,max}} \tag{23}$$

$$x_{i,surf} - x_{i,avg} = -\frac{J_i R_i}{5FD_{s,i} c_{i,max}} \tag{24}$$

J_i is the exchange current density for each electrode $\left[\frac{A}{m^2}\right]$. $x_{i,avg}$ is the ratio of the solid bulk concentration to the maximum solid concentration of Lithium for each electrode. F is Faraday's constant, 96485 in $\left[\frac{C}{mol\ e^-}\right]$. $R_i[m]$ is the particle radius for each electrode. $c_{i,max}\left[\frac{mol}{m^3}\right]$ is the maximum solid phase concentration of Lithium for each electrode. $S_i^0[m^2]$ is the electroactive surface area for each electrode. $x_{i,surf}$ is the ratio of the solid surface concentration to the maximum solid concentration for each electrode, $c_{i,surf}/c_{i,max}$. $D_{s,i}\left[\frac{m^2}{s}\right]$ is the solid phase diffusion coefficient of Lithium for each electrode. Butler-Volmer kinetics are used to describe the intercalation and de-intercalation reactions of Lithium at the electrodes.

$$\frac{J_i}{F} = k_i\left(c_{i,max} - x_{i,surf}c_{i,max}\right)^{0.5}\left(x_{i,surf}c_{i,max}\right)^{0.5}c_e^{0.5}$$

$$\times\left[\exp\left(\frac{\alpha_{a,i}F}{R_gT}\eta_i\right) - \exp\left(\frac{-\alpha_{c,i}F}{R_gT}\eta_i\right)\right] \tag{25}$$

$\alpha_{a,i}$ and $\alpha_{c,i}$ are the anodic and cathodic transfer coefficients, respectively, for the intercalation and deintercalation reactions. If $\alpha_a = \alpha_c = 0.5$, then Equation (25) can be solved explicitly for η_i, as shown in Equation (26). $R_g\left[\frac{J}{molK}\right]$ is the gas contant.

$$\eta_i = \frac{R_g T}{0.5F} \ln\left(\frac{\left(J_i \pm \left(-4c_e F^2 c_{i,max}^2 k_i^2 x_{i,surf}^2 + 4c_e F^2 c_{i,max}^2 k_i^2 x_{i,surf} + J_i^2\right)^{0.5}\right)}{2F c_e^{0.5} k_i \left(c_{i,max} x_{i,surf}\right)^{0.5} \left(c_{i,max} - c_{i,max} x_{i,surf}\right)^{0.5}}\right) \tag{26}$$

Overpotentials, η_i, are also

$$\eta_p = \phi_p - U_p^\theta \tag{27}$$

$$\eta_n = \phi_n - U_n^\theta \tag{28}$$

The open circuit potentials, U_i^θ, are the same as those defined in Equations (18a) and (18b) for the ECM model with an exception. By finding an accurate fit to the experimental data, the parameter which is .7222 has been changed to .8214 and the parameter which is -4.656 has been changed to -4.8801. The difference of the electrode potentials, ϕ_i, (with Equations (27) and (28) substituted) is the cell voltage.

$$V_{cell} = \eta_p + U_p^\theta - \left(\eta_n + U_n^\theta\right) \tag{29}$$

Table 1.3 shows the best IG's from literature and training data. Figure 1.11 displays the NLLS fit of SP. The best IG's are provided for four states, S_i and $x_{i,avg,0}$. The prediction is very inaccurate unless NLLS is constrained to 5% of the best IG's. The UKF prediction, shown in Figure 1.12, is more accurate than NLLS unless NLLS is 5% constrained. The error of NLLS 5% constrained is 272 seconds. The P^x matrix of UKF was made by diagonalizing 5% of the parameters squared. Figure 1.13 shows PF with

5% variation on the correct parameters. Although one prediction PDF is shown the particle shades shift due to two later predictions. PF gives an error of 8.54 seconds for the most accurate prediction. Table 1.4 includes a matrix of the prediction error of the models and methods. The results indicate that the methods are of increasing accuracy from NLLS to UKF to NLLS 5% constrained to PF across all three models.

1.4 CONCLUSIONS

The PF was compared to other algorithms for prediction accuracy using three separate models. Fifty data points were used for predictions in all cases. Tracking the c parameter and/or the d state with correct initial guesses and correct fixed states gave good results with NLLS. Tracking four states did not unless NLLS was constrained to 5% of the correct or best initial guesses. UKF was more accurate given 5% variation on the correct initial guesses of the states unless NLLS was 5% constrained. PF was most accurate. PF used state estimates from training data creating a diversity of particles. Adjustments were made to the weights of the particles, which affected the prediction PDF, but the state estimates were not changed from data available for predictions. With four states tracked and correct initial guesses varied by 5% PF performed more accurately than UKF which performed more accurately than NLLS but less than NLLS 5% constrained for all three models. The three models for prediction testing were a data-driven model from He, et al, an equivalent circuit model, and a physics-based single particle model. The physics-based model has the advantage over the other models in that its parameters are physically meaningful and it is derived from first principles. The predictions made of Lithium-ion ion batteries were the remaining useful life with He et

al's model, and the time until end of discharge with the equivalent circuit model and the single particle model.

Possible future work is increasing the complexity of assumptions for making predictions, such as considering current loads other than constant current for predicting time until end of discharge or using incorrect initial guesses. Less than accurate initial guesses and non-constant current loading can require introducing resampling in the PF algorithm. Other possible future work includes making remaining useful life predictions by an equivalent circuit model or single particle model. States in the models may be obtained from voltage measurements during a cycle and a particle filter may use these states to make predictions about future cycles and the remaining useful life.

1.5 TABLES AND FIGURES

Table 1.1 States in He, et al model. The states returned by estimating four states with NLLS with 50 available data and correct initial guesses are listed for comparison. UKF states from 50 available data are listed for comparison.

Parameter	Value	Unit	NLLS estimates with 50 available data	UKF estimates with 50 available data
$\boldsymbol{a}$	$-9.86e-7$	Ah	$-5.08e-3$	$-9.86e-7$
$\boldsymbol{b}$	$5.752e-2$	$\frac{1}{cycle}$	$2.19e-3$	$5.752e-2$
$\boldsymbol{c}$	$8.983e-1$	Ah	$9.06e-1$	$8.898e-1$
$\boldsymbol{d}$	$-8.340e-4$	$\frac{1}{cycle}$	$-9.01e-4$	$-8.340e-4$

Table 1.2. States in ECM model. The states from NLLS and UKF estimation with 50 data points are listed for comparison.

Parameter	Value	Unit	NLLS estimates with 50 available data	UKF estimates with 50 available data
$\boldsymbol{R}$	$1.41e-1$	Ω	$1.19e-1$	$1.39e-1$
$\boldsymbol{Q}$	2.06	Ah	2.61	2.05
$\boldsymbol{C}$	282.6	F	434.6	564.7
$\boldsymbol{R_{ct}}$	$3.70e-3$	Ω	$4.13e-2$	$2.96e-3$

Table 1.3 States in SP model. Values are from training data.

Parameter	Value	Unit
$\boldsymbol{k_n}$	$3.74e-11$	$\frac{m^{2.5}}{mol^{0.5}s}$
$\boldsymbol{k_p}$	$1.75e-11$	$\frac{m^{2.5}}{mol^{0.5}s}$
$\boldsymbol{R_n}$	$2e-6$	m
$\boldsymbol{R_p}$	$2e-6$	m
$\boldsymbol{D_{s,n}}$	$2.91e-14$	$\frac{m^2}{s}$
$\boldsymbol{D_{s,p}}$	$2.79e-14$	$\frac{m^2}{s}$
$\boldsymbol{c_{n,max}}$	30555	$\frac{mol}{m^3}$
$\boldsymbol{c_{p,max}}$	51555	$\frac{mol}{m^3}$
$\boldsymbol{c_e}$	1000	$\frac{mol}{m^3}$
$\boldsymbol{S_n}$	0.2604	m^2
$\boldsymbol{S_p}$	0.2570	m^2
$\boldsymbol{x_{n,avg,0}}$	0.9401	
$\boldsymbol{x_{p,avg,0}}$	0.5169	
$\boldsymbol{\alpha_{a,c,i}}$	0.5	
$\boldsymbol{T}$	298.15	K
$\boldsymbol{R_g}$	8.3143	$\frac{J}{mol\,K}$
$\boldsymbol{F}$	96485	$\frac{C}{mol\,e^-}$

Table 1.4 Matrix of models and methods. Prediction errors reported. Methods are ordered by increasing accuracy.

	He, et al	ECM	SP
NLLS	88 $(cycles)$	874 $(seconds)$	772 $(seconds)$
UKF	19 $(cycles)$	396 $(seconds)$	272 $(seconds)$
NLLS 5% constrained	10 $(cycles)$	212 $(seconds)$	127 $(seconds)$
PF (weighted average)	0.82 $(cycles)$	25.62 $(seconds)$	8.54 $(seconds)$

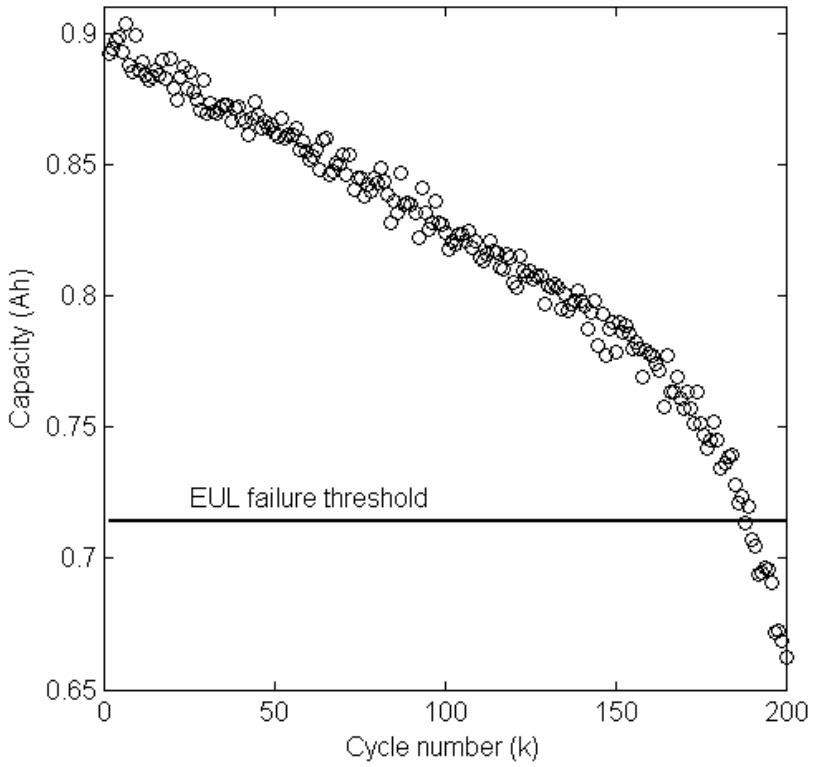

Figure 1.1 Capacity data set for RUL prediction

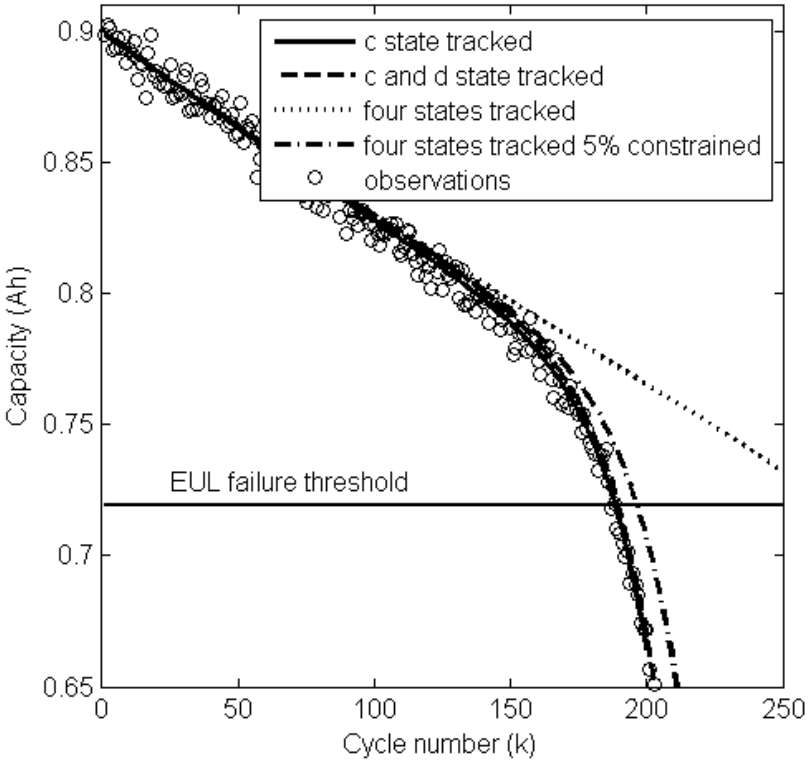

Figure 1.2 Tracking a different number of states. Given the correct fixed states, a variety of states are tracked for prediction. The first 50 data points are used for tracking. Tracking four states gives an inaccurate prediction, unless the knee is in the available data.

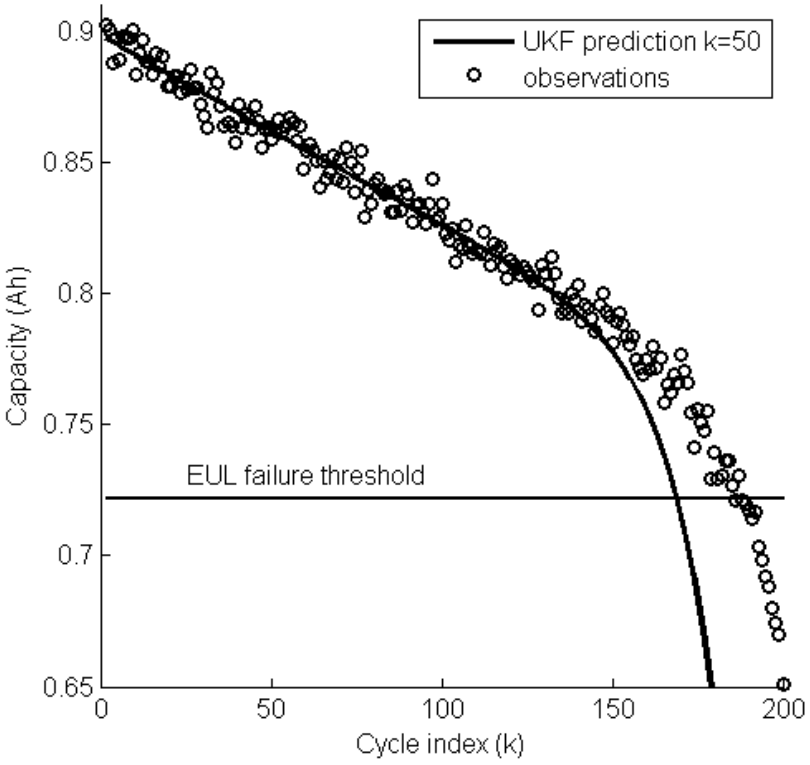

Figure 1.3 UKF RUL prediction for He, et al model

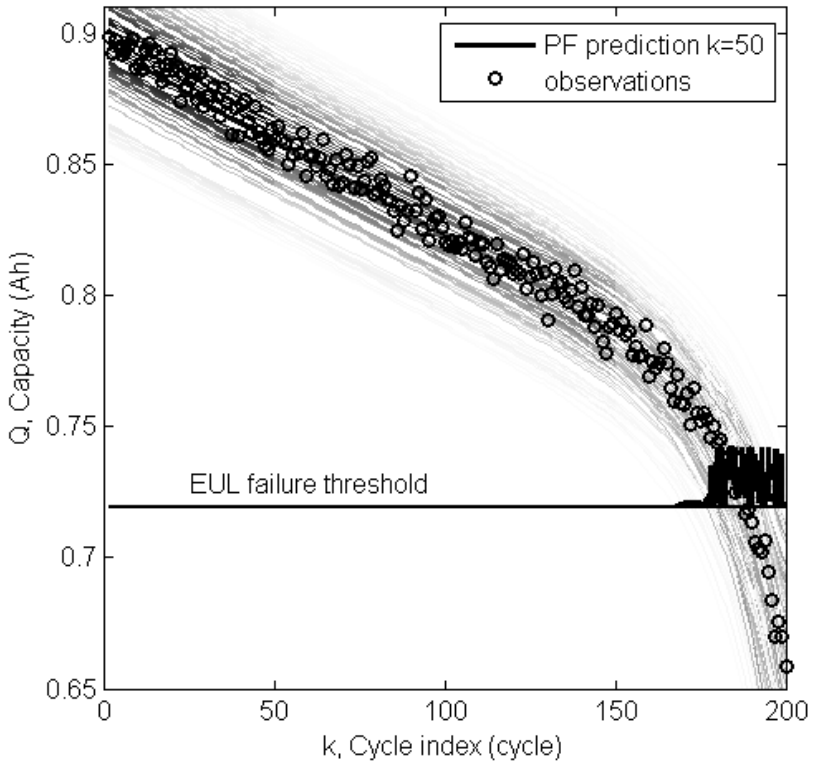

Figure 1.4 PF RUL prediction for He, et al model

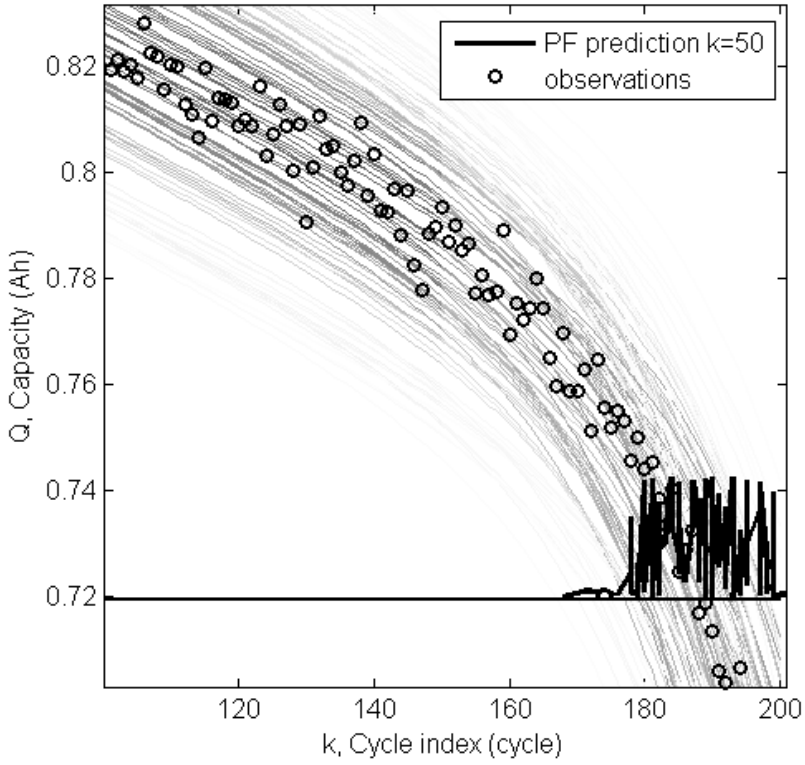

Figure 1.5 Zoom on PF RUL prediction for He, et al model

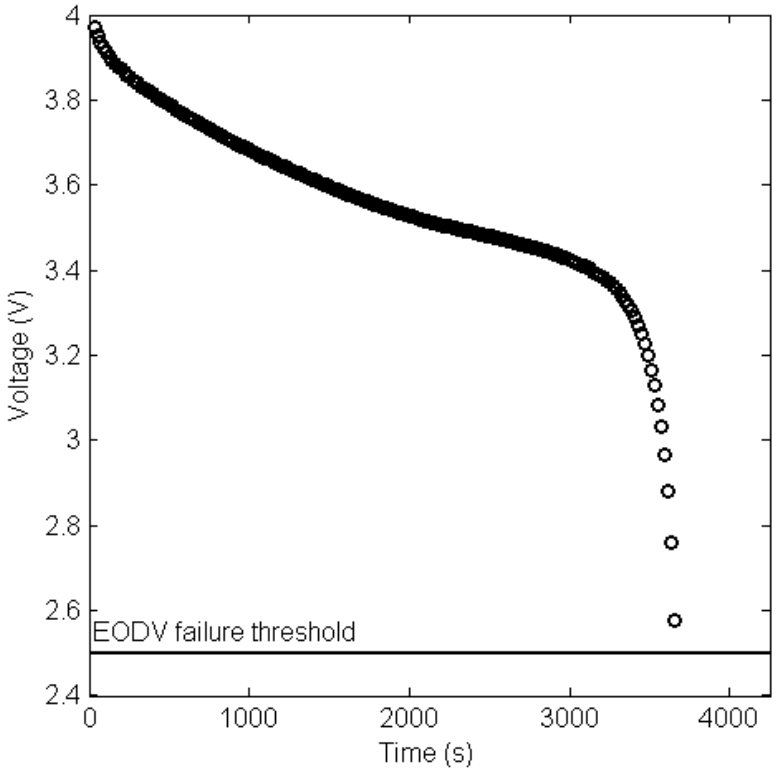

Figure 1.6 ECM data set

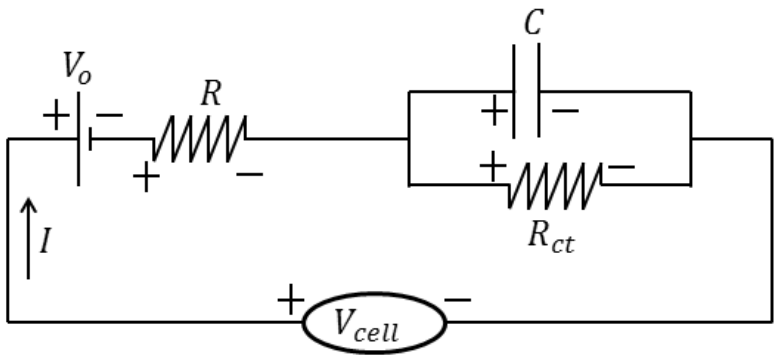

Figure 1.7 Equivalent circuit representation of a Lithium-ion battery.

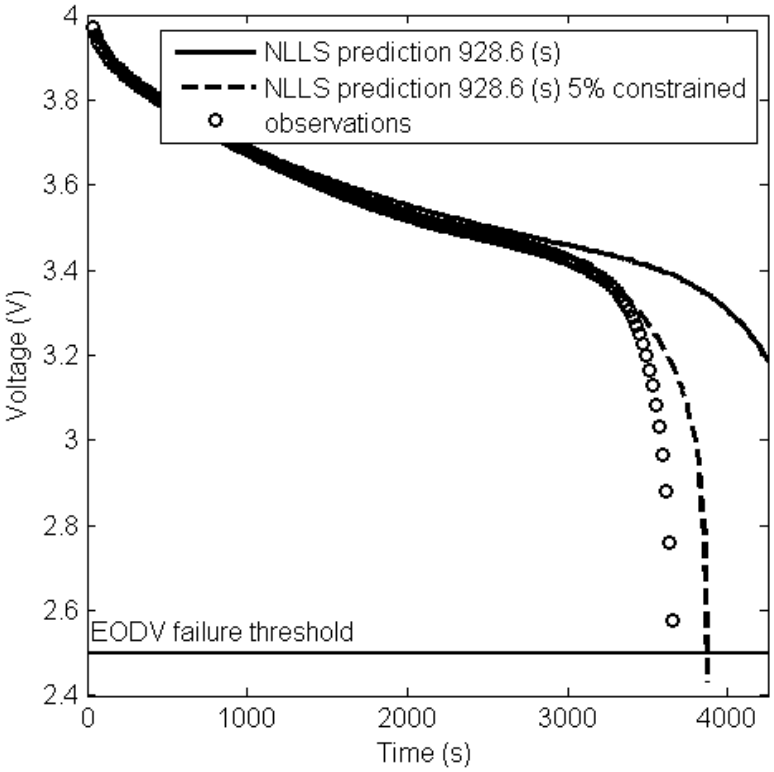

Figure 1.8 ECM NLLS prediction

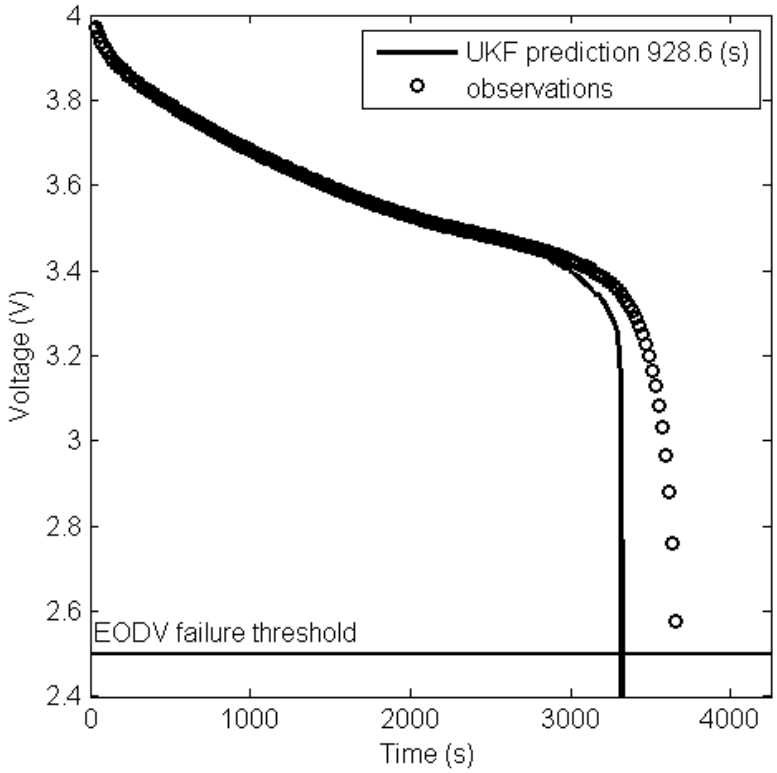

Figure 1.9 ECM UKF prediction

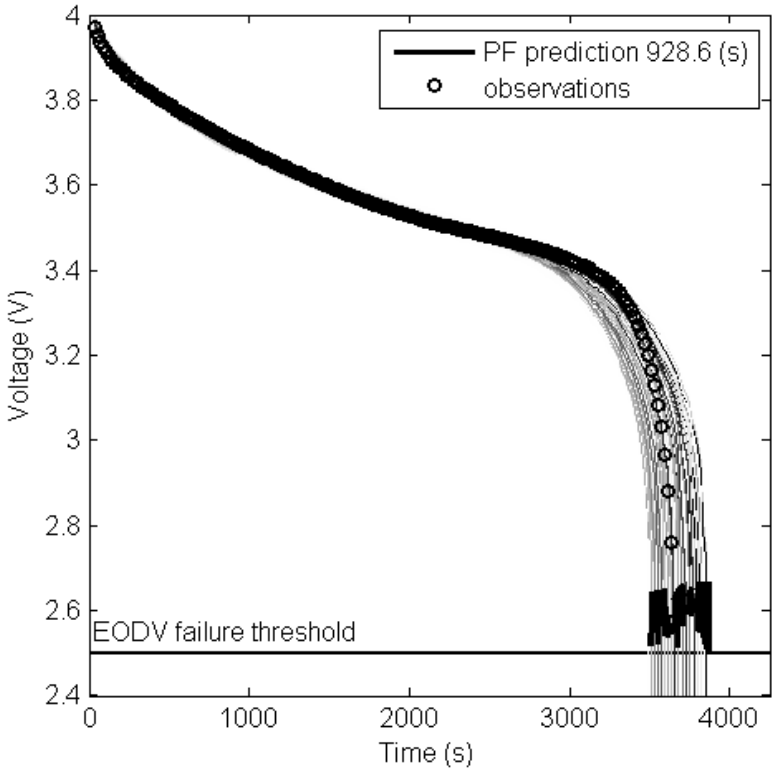

Figure 1.10 ECM PF prediction

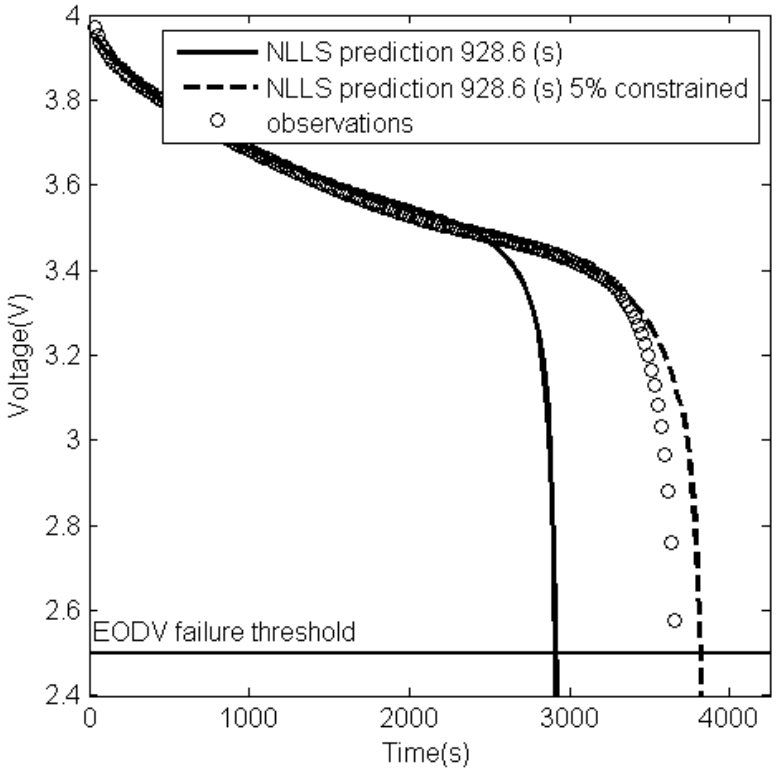

Figure 1.11 SP NLLS prediction

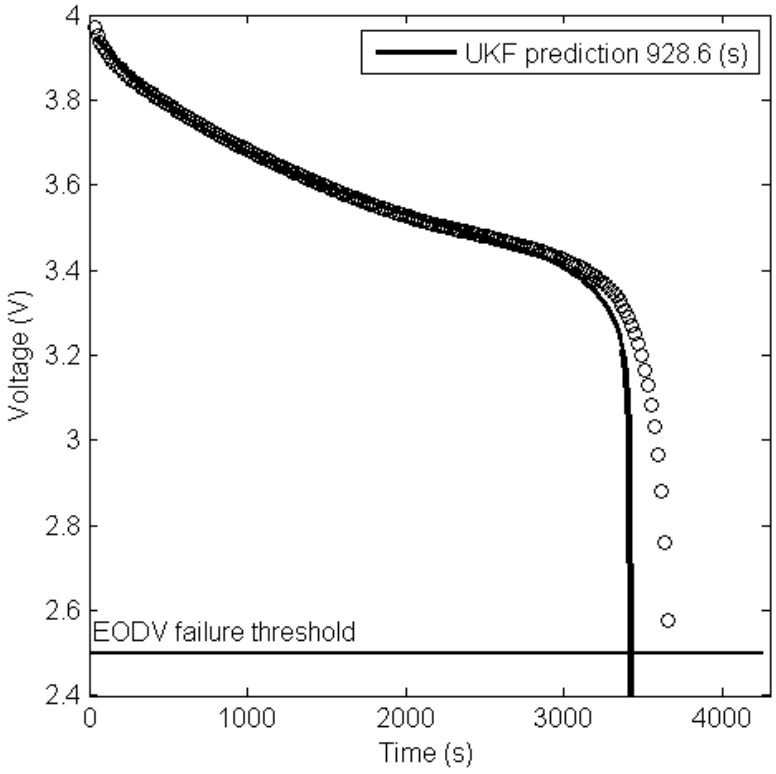

Figure 1.12 SP UKF prediction

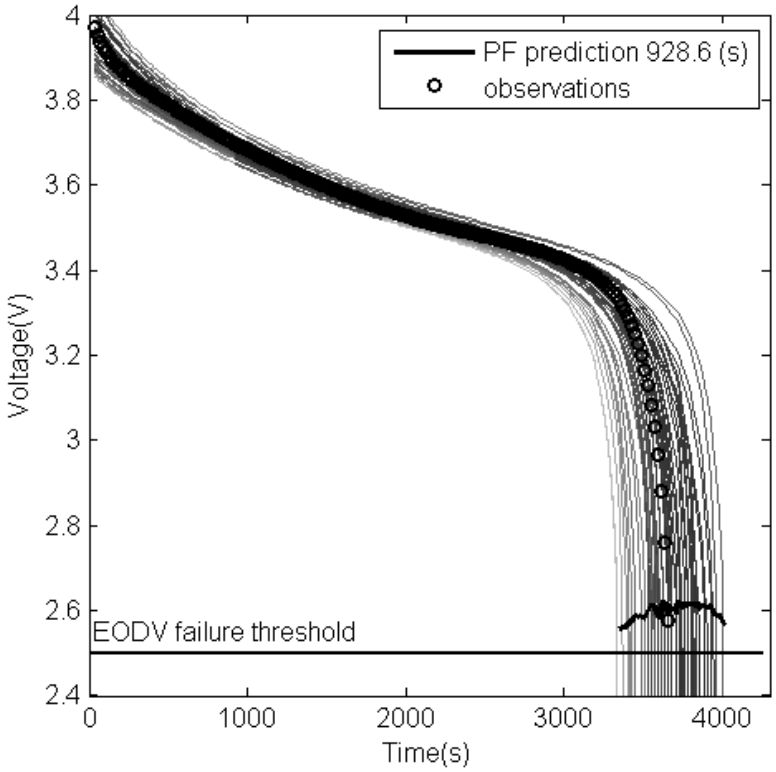

Figure 1.13 SP PF prediction

REFERENCES

[1] W. He, N. Williard, M. Osterman, M. Pecht, Journal of Power Sources 196 (2011) 10314-10321.

[2] Y. Xing, E.W.M. Ma, K.L. Tsui, M. Pecht, Prognostics & System Health Management Conference (2012) Beijing.

[3] B. Saha, P. Quach, K. Goebel, Annual Conference of the Prognostics and Health Management Society (2011).

[4] G. Jin, D.E. Matthews, Z. Zhongbao, Reliability Engineering and System Safety 23 v.196 (2011) 7-20.

[5] B. Pattipati, C. Sankavaram, K. R. Pattipati, IEEE Transactions on Systems, Man, and Cybernetics- Part C: Applications and Reviews 41 v.6 (2011) 869-884.

[6] D. An, J.H. Choi, N.H. Kim, Reliability Engineering and System Safety 115 (2013) 161-169.

[7] S.K. Rahimian, S. Rayman, R.E. White, Journal of the Electrochemical Society 159 (2012) A860-A872.

[8] G.L. Plett, Journal of Power Sources 161 (2006) 1356-1368.

[9] V. Ramadesigan, K. Chen, N. Burns, V. Boovaragavan, R. Braatz, V.R. Subramanian, Journal of the Electrochemical Society 158 (9) (2011) A1048-A1054.

[10] M. Daigle, B. Saha, K. Goebel, IEEE Aerospace Conference (2012).

[11] S.K. Rahimian, S. Rayman, R.E. White, Journal of Power Sources 20 (2011) 8450-8462.

[12] F Cadini, E Zio, D Avram, Probabilistic Engineering Mechanics 24 (2009) 367-373.

[13] S.M. Grewal, A.P. Andrews, Kalman Filtering, John Wiley and Sons, Hoboken, 2008.

[14] N.J. Gordon, D. J. Salmond, A.F.M. Smith, IEE Proceedings- F, 2 (1993) 107-113

[15] M.W. Verbrugge, Modern Aspects of Electrochemistry, Modeling and Numerical Simulations I, vol. 43, Springer, New York, 2009.

[16] B. Saha, K. Goebel, NASA Ames Prognostics Data Repository (2007) http://ti.arc.nasa.gov/project/prognostic-data-repository.

[17] K.J. Beers, Numerical Methods for Chemical Engineering, Cambridge University Press (2007).

[18] R.E. White, V. Subramanian, Computational Methods in Chemical Engineering with Maple, Springer, 2010.

[19] D. Simon, Optimal State Estimation, John Wiley and Sons, Hoboken, 2006.

[20] P. Ramadass, B. Haran, R.E. White, B.N. Popov, J. Power Sources 123 (2003) 230.

APPENDIX A – UNSCENTED KALMAN FILTER

The unscented Kalman filter (UKF) is a deterministic, sample-based filter method, used in online estimation applications. An updated state estimate is calculated at each new measurement, as opposed to NLLS, which uses all measurements for making state estimates. For non-linear models, UKF is preferred to EKF, among Kalman filters. The UKF algorithm follows[13]: First, the state vector estimate, $\underline{\hat{x}}_0$, and its covariance, P_0^x, are initialized. For example, in He, et al's model, $\underline{\hat{x}}_0$ is in fact $\underline{\theta}$, which contain a, b, c, d.

$$\underline{\hat{x}}_0 = E[x_0] \qquad P_0^x = E\left[\left(\underline{x}_0 - \underline{\hat{x}}_0\right)\left(\underline{x}_0 - \underline{\hat{x}}_0\right)^T\right] \tag{30}$$

The hat indicates an estimate. The next step in the algorithm is to create a symmetric set of samples, with the Cholesky decomposition of P_0^x. The symmetric $2n + 1$ sampling scheme is the sampling choice in this work[16].

$$C = chol(P_{k-1}^x) \tag{31}$$

$$\mathrm{X}_{0,k-1} = \underline{\hat{x}}_{k-1} \qquad \mathrm{X}_{i,k-1} = \underline{\hat{x}}_{k-1} + \sqrt{n + \kappa c_{jj}} \quad \mathrm{X}_{i+n,k-1} = \underline{\hat{x}}_{k-1} - \sqrt{n + \kappa c_{jj}} \tag{32}$$

X is the matrix that holds all the samples of the states. c_{jj} are the diagonals of the Cholesky decomposition. κ is a tuning parameter, and $n = 1$ is the number of state variables. j is an index which begins at one and continues to include n, and is as long as the number of states. The weights, W, associated with the samples in X are calculated,

$$W_0 = \frac{\kappa}{\sqrt{n+\kappa}} \qquad W_i = \frac{1}{[2(n+\kappa)]} \qquad W_{i+n} = \frac{1}{[2(n+\kappa)]} \tag{33}$$

The above unscented transform gives the prior, $-$, state estimate at index, k.

$$\left[\underline{\hat{x}}_{k,-}, P_{k,-}^{x}\right] = UT\left(\underline{\hat{x}}_{k-1,+}, C_{k-1,+}, f(\cdot), \kappa, \underline{u}_{k-1}\right) \tag{34}$$

UT, the unscented transform, is the sampling scheme of Equations (32) and (33). $\underline{u}_{k-1}$ are the input variables. $f(\cdot)$ is the state dynamic model. The He, et al model does not have a state dynamic model, but other models may. Considering the same example model, the input variable is k for He, et al's model. Next, the measurement update leads to the posterior state estimate at k.

$$Y_{i,k} = h_k\left(X_i, I_{app,k}\right) \tag{35}$$

$$\hat{y}_k^- = \sum_i^{2n} W_i Y_i \tag{36}$$

$$P_k^{yy} = \sum_{i=0}^{2n} W_i\left(Y_{i,k} - \hat{y}_k^-\right)\left(Y_{i,k} - \hat{y}_k^-\right)^T \tag{37}$$

$$P_k^{xy} = \sum_{i=0}^{2n} W_i\left(\mathrm{X}_i - \underline{\hat{x}}_{k,-}\right)\left(Y_{i,k} - \hat{y}_{k,-}\right) \tag{38}$$

P_k^{yy} is the measurement space covariance, and P_k^{xy} is the cross covariance. The update step is completed after,

$$K_k = P_k^{xy}\left(P_k^{yy}\right)^{-1} \tag{39}$$

$$\underline{\hat{x}}_k^+ = \underline{\hat{x}}_k^- + K_k\left(V_{cell,k}^* - \hat{y}_k^-\right) \tag{40}$$

$$P_k^{x+} = P_k^{x-} - K_k P_k^{yy} K_k^T + Q \tag{41}$$

Q is the process noise. The UKF algorithm is presented as a flowchart in Figure A.1 .

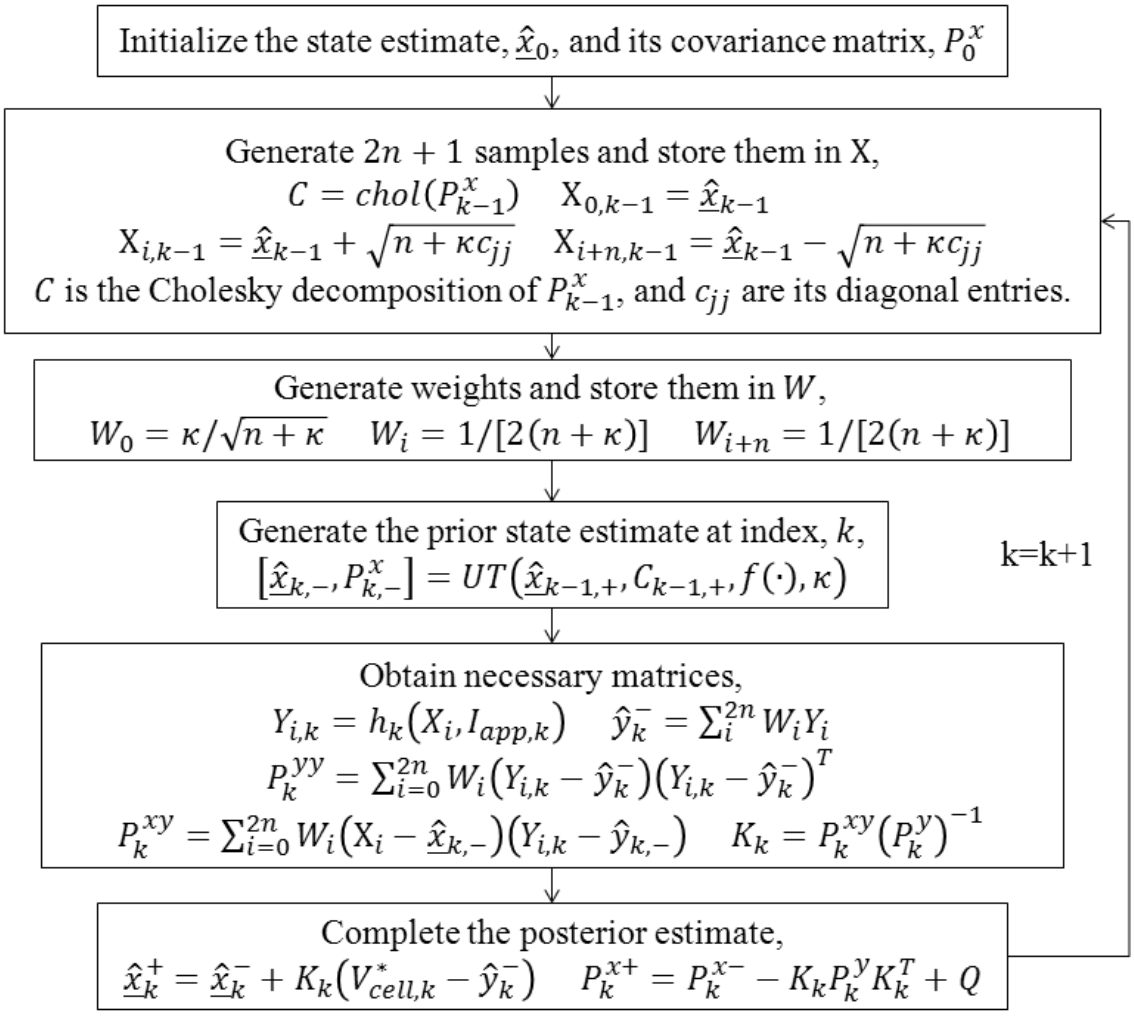

Figure A.1 UKF algorithm flowchart

APPENDIX B – MATLAB PROGRAMS

```
clc

clear all

close all

% Eric Walker

% M.S. thesis NLLS



%% He, et al



load(['D:\Documents and Settings\Eric\My Documents\spring 2013\NASA'...

    ' Ames Data\B0005.mat']);

load(['D:\Documents and Settings\Eric\My Documents\spring 2013\NASA'...

    ' Ames Data\B0007.mat']);
```

```
first_batt = (-9.86e-7)
 * exp(5.752e-2 * (1:200)) + (8.983e-1) * ...
    exp((-8.340e-4) * (1:200)) + 0.005*randn(1,200);

second_batt = (-9.86e-7) * exp(5.752e-2 * (1:300)) + (8.983e-1) *...
    exp((-8.340e-4) * (1:300)) + 0.005*randn(1,300);

% Present the data in Figure 1.
hold on % Show all plots on the same figure.
plot(1:length(first_batt), first_batt, 'ko')
plot([1,200],first_batt(1)*0.8*[1,1],'k-','linewidth',1.5)
text(25,first_batt(1)*0.81,'EUL failure threshold')
xlabel('Cycle number (k)')
ylabel('Capacity (Ah)')
ylim([0.65, 0.91])
```

```
axis square

box on



%NLLS

theta=[-9.86e-7,5.752e-2,8.983e-1,-8.34e-4];




[theta_50_one_st, resnorm] = lsqnonlin(@(t_1) (second_batt(1:50))...

    - (-9.86e-7*exp(5.752e-2*...

    (1:50)) + t_1 * exp(-8.34e-4*(1:50))),8.983e-1);



[theta_50_two_st, resnorm] = lsqnonlin(@(theta) (second_batt(1:50))...

    - (-9.86e-7*exp(5.752e-2*...

    (1:50)) + theta(1) * exp(theta(2)*(1:50))),[8.983e-1,-8.34e-4],...

    [0, -inf], [inf, 0]);



[theta_50_four_st, resnorm] = lsqnonlin(@(t_4) (second_batt(1:50)) ...
```

```
    - (t_4(1)*exp(t_4(2)*...
    (1:50)) + t_4(3) * exp(t_4(4)*(1:50))),[-9.86e-7,5.752e-2,...
    8.983e-1,-8.34e-4],[-inf, 0, 0, -inf], [0, inf, inf, 0]);


[theta_50_four_st_const, resnorm] = lsqnonlin(@(t_4_c) ...
    (second_batt(1:50)) - (t_4_c(1)*exp(t_4_c(2)*...
    (1:50)) + t_4_c(3) * exp(t_4_c(4)*(1:50))),[-9.86e-7,...
    5.752e-2,8.983e-1,-8.34e-4],[-9.86e-7*1.05, 5.752e-2*0.95,...
    8.983e-1*0.95, -8.34e-4*1.05], [-9.86e-7*0.95, 5.752e-2*1.05,...
    8.983e-1*1.05, -8.34e-4*0.95]);



figure  % See three NLLS predictions
hold on
plot(1:200, (theta(1)*exp(theta(2)*...
    (1:200)) + theta_50_one_st * exp(theta(4)*(1:200))),'k-',...
    'linewidth',2)
```

```
plot(1:length(second_batt), (theta(1)*exp(theta(2)*...
    (1:length(second_batt))) + theta_50_two_st(1) * exp(...
    theta_50_two_st(2)*(1:length(second_batt)))),'k--',...
    'linewidth',2)
plot(1:length(second_batt), (theta_50_four_st(1)*exp(...
    theta_50_four_st(2)*...
    (1:length(second_batt))) + theta_50_four_st(3) * exp(...
    theta_50_four_st(4)*(1:length(second_batt)))),'k:',...
    'linewidth',2)
plot(1:length(second_batt), (theta_50_four_st_const(1)*exp(...
    theta_50_four_st_const(2)*...
    (1:length(second_batt))) + theta_50_four_st_const(3) * exp(...
    theta_50_four_st_const(4)*(1:length(second_batt)))),'k-.',...
    'linewidth',2)
plot(1:length(second_batt), second_batt, 'ko')
plot([1,250],second_batt(1)*0.8*[1,1],'k-','linewidth',1.5)
text(25,second_batt(1)*0.81,'EUL failure threshold')
```

```
xlabel('Cycle number (k)')

ylabel('Capacity (Ah)')

ylim([0.65, 0.91])

axis square

box on

legend('c state tracked ','c and d state tracked',...

    'four states tracked','four states tracked 5% constrained',...

    'observations')



%% ECM



load(['D:\Documents and Settings\Eric\My Documents\spring 2013\'...

    'NASA Ames Data\B0006.mat']);



global first_discharge_time first_discharge_current...

    first_discharge_voltage;

first_discharge_voltage = B0006.cycle(1,2).data.Voltage_measured...
```

```
    (3:end);
first_discharge_time    = B0006.cycle(1,2).data.Time(3:end);
first_discharge_current = B0006.cycle(1,2).data.Current_measured...
    (3:end);



[ecm, resnorm, residuals] = lsqnonlin( @ecm_obj_fun,
[0.129635,2.0764,21.045,0.110328],[0 0 0 0], [1, 10, 2000, 1]);
second_discharge_voltage = B0006.cycle(1,4).data.Voltage_measured...
    (3:end);
second_discharge_time    = B0006.cycle(1,4).data.Time(3:end);
second_discharge_current = B0006.cycle(1,4).data.Current_measured...
    (3:end);
second_discharge_time    = [second_discharge_time ...
    (second_discharge_time(1:100) + second_discharge_time(end))];
second_discharge_current = [second_discharge_current ...
    second_discharge_current(1:100)];
```

```
% Take the first forty data points for the NLLS prediction

first_discharge_voltage  = second_discharge_voltage(1:50);

first_discharge_time     = second_discharge_time(1:50);

first_discharge_current  = second_discharge_current(1:50);

[ecm_2, resnorm, residuals] = lsqnonlin( @ecm_obj_fun, ecm,...

   [0 0 0 0], [1, 10, 2000, 1]);




R = ecm_2(1);

Q = ecm_2(2);

C = ecm_2(3);

R_ct = ecm_2(4);



SOC_cell = 1 + second_discharge_current ./ (Q*3600) .*...

   second_discharge_time;
```

```
SOC_n = 0.79.*SOC_cell + 0.01;

SOC_p = 0.97-0.51*SOC_cell;



x_nsurf = SOC_n;

x_psurf_set = SOC_p;



U_n     = .7222 + .1387*x_nsurf + .029*x_nsurf.^0.5 -...

   .0172./x_nsurf + ...

   .0019./x_nsurf.^1.5 + .2808 * exp(0.9-15*x_nsurf) -.7984...

   * exp (...

   0.4465*x_nsurf - 0.4108);



U_p_set     = ( -4.656 + 88.669 * x_psurf_set.^2 - 401.119 *...

   x_psurf_set.^4 + 342.909 * ...

   x_psurf_set.^6 - 462.471 * x_psurf_set.^8 + 433.434 *...

   x_psurf_set.^10 ) ./ ...

   ( -1 + 18.933*x_psurf_set.^2 - 79.532 * x_psurf_set.^4 +...
```

```
    37.311 * x_psurf_set.^6 ...
    - 73.083 * x_psurf_set.^8 + 95.96*x_psurf_set.^10 );

V_o = U_p_set - U_n ;

V_cell = V_o + second_discharge_current*R + Q./C .* exp(-...
    second_discharge_time./(R_ct * C))...
    + second_discharge_current.*R_ct.*(1-exp(-...
    second_discharge_time./(R_ct * C)));

% Now the constrained

[ecm_2_c, resnorm, residuals] = lsqnonlin( @ecm_obj_fun, ecm,...
    0.95*ecm, 1.05*ecm);

R = ecm_2_c(1);
```

```
Q = ecm_2_c(2);

C = ecm_2_c(3);

R_ct = ecm_2_c(4);



SOC_cell = 1 + second_discharge_current ./ (Q*3600) .*...

    second_discharge_time;



SOC_n = 0.79.*SOC_cell + 0.01;

SOC_p = 0.97-0.51*SOC_cell;



x_nsurf = SOC_n;

x_psurf_set = SOC_p;



U_n     = .7222 + .1387*x_nsurf + .029*x_nsurf.^0.5 -...

    .0172./x_nsurf + ...

    .0019./x_nsurf.^1.5 + .2808 * exp(0.9-15*x_nsurf) -.7984...

    * exp (...
```

```
    0.4465*x_nsurf - 0.4108);


U_p_set     = ( -4.656 + 88.669 * x_psurf_set.^2 - 401.119 *...
    x_psurf_set.^4 + 342.909 * ...
    x_psurf_set.^6 - 462.471 * x_psurf_set.^8 + 433.434 *...
    x_psurf_set.^10 ) ./ ...
    ( -1 + 18.933*x_psurf_set.^2 - 79.532 * x_psurf_set.^4 +...
    37.311 * x_psurf_set.^6 ...
    - 73.083 * x_psurf_set.^8 + 95.96*x_psurf_set.^10 );


V_o = U_p_set - U_n ;


V_cell_c = V_o + second_discharge_current*R + Q./C .* exp(-...
    second_discharge_time./(R_ct * C))...
    + second_discharge_current.*R_ct.*(1-exp(-...
    second_discharge_time./(R_ct * C)));
```

```
first_discharge_voltage = B0006.cycle(1,4).data.Voltage_measured...
   (3:end);
first_discharge_time    = B0006.cycle(1,4).data.Time(3:end);
for y = 1:length(V_cell)
   if V_cell(y) < 2.5
      break
   end
end
for z = 1:length(V_cell_c)
   if V_cell_c(z) < 2.5
      break
   end
end
V_cell = V_cell(1:y);
V_cell_c = V_cell_c(1:z);
figure
hold on
```

```
plot(second_discharge_time(1:length(V_cell)),V_cell,'k-',...
    'linewidth',2)
plot(second_discharge_time(1:length(V_cell_c)),V_cell_c,'k--',...
    'linewidth',2)
plot(second_discharge_time(1:length(second_discharge_voltage)),second_discharge_volt
age, 'ko',...
    'linewidth', 1.5)
plot([1,4250],2.5*[1,1],'k-','linewidth',1.5) % The EODV line.
text(25,2.55,'EODV failure threshold')
axis([0 4250 2.4 4])
axis square
box on
xlabel('Time (s)')
ylabel('Voltage (V)')
legend('NLLS prediction 928.6 (s)', ...
    'NLLS prediction 928.6 (s) 5% constrained', 'observations')


% Reset the following three variables after the above nlls.
```

```
first_discharge_voltage = B0006.cycle(1,2).data.Voltage_measured(3:end);

first_discharge_time    = B0006.cycle(1,2).data.Time(3:end);

first_discharge_current = B0006.cycle(1,2).data.Current_measured(3:end);

[ecm, resnorm, residuals] = lsqnonlin( @ecm_obj_fun, ...

   [1.17e-8, 2.1, 1795.6, 0.28],[0 0 0 0], [1, 2.5, 2000, 1]);



%%  NLLS Single Particle model



[S_x_avg,resnorm,res] = lsqnonlin(@SP_obj_fun, ...

   [0.2607, 0.2571, 0.9388, 0.5171]...

   ,[],[1 1 1],[],second_discharge_time(1:50),...

   second_discharge_current(1:50),second_discharge_voltage(1:50));



[S_x_avg_c,resnorm_c,res_c] = lsqnonlin(@SP_obj_fun, ...

   [0.2607, 0.2571, 0.9388, 0.5171]...

   ,[0.2607, 0.2571, 0.9388, 0.5171]*0.95,...

   [0.2607, 0.2571, 0.9388, 0.5171]*1.05,...
```

```
    [],second_discharge_time(1:50),...
    second_discharge_current(1:50),second_discharge_voltage(1:50));



[voltagePredi]        = SP(S_x_avg,second_discharge_time,...
    second_discharge_current);
s = find(voltagePredi<2.4,1);
voltagePredi=voltagePredi(1:s);
[voltagePredi_c]      = SP(S_x_avg_c,second_discharge_time,...
    second_discharge_current);


for z = 1:length(voltagePredi_c)
    if voltagePredi_c(z) < 2.4
        break
    end
end
voltagePredi_c = voltagePredi_c(1:z);
```

```
figure

axis([0 4250 2.4 4])

axis square

hold on

box on

xlabel('Time(s)');

ylabel('Voltage(V)');

plot(second_discharge_time(1:length(voltagePredi)),voltagePredi, 'k-',...

    'linewidth',2)

plot(second_discharge_time(1:length(voltagePredi_c)),voltagePredi_c, 'k--',...

    'linewidth',2)

plot(second_discharge_time(1:length(second_discharge_voltage))...

    ,second_discharge_voltage,'ko')

plot([1,4250],2.5*[1,1],'k-','linewidth',1.5) % The EODV line.

text(25,2.55,'EODV failure threshold')

legend('NLLS prediction 928.6 (s)', ...
```

```
    'NLLS prediction 928.6 (s) 5% constrained', 'observations')

clc

clear all

close all



%Eric Walker

%M.S. thesis UKF



%% UKF He, et al

second_batt = (-9.86e-7) * exp(5.752e-2 * (1:200)) + (8.983e-1) *...

    exp((-8.340e-4) * (1:200)) + 0.005*randn(1,200);



theta=[-9.86e-7; 5.752e-2; 8.983e-1; -8.34e-4];



P       = diag([(3.442e-8 - (-2.007e-6)), (6.221e-2 - 5.283e-2), ...

    (9.035e-1 - 8.931e-1), (-7.670e-4 - (-9.007e-4))])^2  ;...

    % 0.95 confidence from He, et al.
```

```
kappa   = 0.5;

Q_storage = []

for i = 1:length(second_batt)

    C       = chol(P);

    Chi(:,1) = theta;

    Chi(:,2) = theta + sqrt(4 + kappa)*[C(1,1); 0; 0; 0];

    Chi(:,3) = theta + sqrt(4 + kappa)*[0; C(2,2); 0; 0];

    Chi(:,4) = theta + sqrt(4 + kappa)*[0; 0; C(3,3); 0];

    Chi(:,5) = theta + sqrt(4 + kappa)*[0; 0; 0; C(4,4)];

    Chi(:,6) = theta - sqrt(4 + kappa)*[C(1,1); 0; 0; 0];

    Chi(:,7) = theta - sqrt(4 + kappa)*[0; C(2,2); 0; 0];

    Chi(:,8) = theta - sqrt(4 + kappa)*[0; 0; C(3,3); 0];

    Chi(:,9) = theta - sqrt(4 + kappa)*[0; 0; 0; C(4,4)];

    W(1)    = kappa / sqrt(4 + kappa);

    W(2:9)  = 1    / (2*(4+kappa));

    W       = W/sum(W);
```

```
Q_cap    = Chi(1,:) .* exp( Chi(2,:) * i ) + Chi(3,:) .* ...
   exp( Chi(4,:) * i );

Q_hat    = sum(W.*Q_cap);
Q_storage = [Q_storage; Q_hat];

if i < 51
   P_yy     = sum(W.*((Q_cap).^2));

   P_xy     = (Chi - repmat(theta,1,9)) .* [W;W;W;W] *(Q_cap...
      - Q_hat)';
   K_k      = P_xy/(P_yy+0.05);
   egg      = theta + K_k*(second_batt(i) - Q_hat);
   theta(1) = egg(1);
   theta(2) = egg(2);
   theta(3) = egg(3);
```

```
        theta(4) = egg(4);




        P        = P - K_k * P_xy';
    end




end
figure
hold on
plot(1:length(second_batt), Q_storage, 'k-','linewidth',2)
plot(1:length(second_batt), second_batt,'ko','linewidth',1.5)
legend('UKF prediction k=50', 'observations')
plot([1,200],second_batt(1)*0.8*[1,1],'k-','linewidth',1.5)
text(25,second_batt(1)*0.81,'EUL failure threshold')
ylim([0.65, 0.91])
xlabel('k, Cycle index (cycle)')
```

```
ylabel('Q, Capacity (Ah)')

axis square

box on




%% UKF ECM

clear all

load(['D:\Documents and Settings\Eric\My Documents\spring 2013\'...

    'NASA Ames Data\B0006.mat']);




global first_discharge_time first_discharge_current...

    first_discharge_voltage;

first_discharge_voltage = B0006.cycle(1,2).data.Voltage_measured...

    (3:end);

first_discharge_time    = B0006.cycle(1,2).data.Time(3:end);

first_discharge_current = B0006.cycle(1,2).data.Current_measured...
```

```
    (3:end);


second_discharge_voltage = B0006.cycle(1,4).data.Voltage_measured...
    (3:end);
second_discharge_time    = B0006.cycle(1,4).data.Time(3:end);
second_discharge_current = B0006.cycle(1,4).data.Current_measured...
    (3:end);
second_discharge_time    = [second_discharge_time ...
    (second_discharge_time(1:30) + second_discharge_time(end))];
second_discharge_current = [second_discharge_current ...
    second_discharge_current(1:30)];


[ecm, resnorm, residuals] = lsqnonlin( @ecm_obj_fun, ...
    [1.17e-8, 2.1, 1795.6, 0.28],[0 0 0 0], [1, 10, 2000, 2]);
ecm=ecm'
sigma = 0.0015;
```

```
V_cell_storage = [];

R = ecm(1) %+ ecm(1)/4 * randn(1,50)%1795;

Q = ecm(2) % + ecm(2)/20*(randn(1,50))); % %2.0593;

C = ecm(3) %+ ecm(3)/4 * randn(1,50);%1.17e-8;

R_ct = ecm(4) %+ ecm(4)/4*randn(1,50) %0.1451;



P       = (diag(ecm) / 20)^2;



ecm_orig= ecm;

kappa = 0.5;

for i = 1:length(second_discharge_voltage)


    C       = chol(P);

    Chi(:,1) = ecm;
```

```
Chi(:,2) = ecm + sqrt(4 + kappa)*[C(1,1); 0; 0; 0];

Chi(:,3) = ecm + sqrt(4 + kappa)*[0; C(2,2); 0; 0];

Chi(:,4) = ecm + sqrt(4 + kappa)*[0; 0; C(3,3); 0];

Chi(:,5) = ecm + sqrt(4 + kappa)*[0; 0; 0; C(4,4)];

Chi(:,6) = ecm - sqrt(4 + kappa)*[C(1,1); 0; 0; 0];

Chi(:,7) = ecm - sqrt(4 + kappa)*[0; C(2,2); 0; 0];

Chi(:,8) = ecm - sqrt(4 + kappa)*[0; 0; C(3,3); 0];

Chi(:,9) = ecm - sqrt(4 + kappa)*[0; 0; 0; C(4,4)];

W(1)     = kappa / sqrt(4 + kappa);

W(2:9)   = 1     / (2*(4+kappa));

W        = W/sum(W);



SOC_cell = 1 + second_discharge_current(i) ./ ...

   (Chi(2,:)*3600) .* second_discharge_time(i);



SOC_n = 0.79.*SOC_cell + 0.01;

SOC_p = 0.97-0.51*SOC_cell;
```

```
x_nsurf = SOC_n;

x_psurf_set = SOC_p;



U_n     = .7222 + .1387*x_nsurf + .029*x_nsurf.^0.5 -...
    .0172./x_nsurf + ...
    .0019./x_nsurf.^1.5 + .2808 * exp(0.9-15*x_nsurf) -...
    .7984 * exp (...
    0.4465*x_nsurf - 0.4108);


U_p_set     = ( -4.656 + 88.669 * x_psurf_set.^2 - 401.119 *...
    x_psurf_set.^4 + 342.909 * ...
    x_psurf_set.^6 - 462.471 * x_psurf_set.^8 + 433.434 *...
    x_psurf_set.^10 ) ./ ...
    ( -1 + 18.933*x_psurf_set.^2 - 79.532 * x_psurf_set.^4 +...
    37.311 * x_psurf_set.^6 ...
```

```
    - 73.083 * x_psurf_set.^8 + 95.96*x_psurf_set.^10 );


V_o = U_p_set - U_n ;


V_cell = V_o + second_discharge_current(i)*Chi(1,:) + ...
    Chi(2,:)./Chi(3,:) .* exp(-second_discharge_time(i)./...
    (Chi(4,:) .* Chi(3,:)))...
    + second_discharge_current(i).*Chi(4,:).*(1-exp(...
    -second_discharge_time(i)./(Chi(4,:) .* Chi(3,:))));



y_hat    = sum(W.*V_cell);
V_cell_storage = [V_cell_storage; y_hat];


if i < 50
    P_yy     = sum(W.*((V_cell).^2));
```

```
        P_xy     = (Chi - repmat(ecm,1,9)) .* [W; W; W; W] *...

           (V_cell - y_hat)'; %P_xz is (1x1).

        K_k      = P_xy/(P_yy+0.05);

        ecm      = ecm + K_k*(second_discharge_voltage(i) - y_hat);




        err(i)   = (second_discharge_voltage(i) - y_hat);



        P        = P - K_k * P_xy';

    end




end



% This block of code is to stop the prediction when the

%voltage drops below 2.5 volts.
```

```
k=1
while V_cell_storage(k) > 2.5 && k < 195
    k = k+1;
end
V_cell_storage = V_cell_storage(1:k);


figure
hold on
plot(second_discharge_time(1:length(V_cell_storage)),...
    V_cell_storage,'k-','linewidth',2)
plot(second_discharge_time(1:length(second_discharge_voltage)),...
    second_discharge_voltage, 'ko', 'linewidth', 1.5)
plot([1,4250],2.5*[1,1],'k-','linewidth',1.5) % The EODV line.
text(25,2.55,'EODV failure threshold')
axis([0 4250 2.4 4])
axis square
box on
```

```
xlabel('Time (s)')

ylabel('Voltage (V)')

legend('UKF prediction 928.6 (s)', 'observations')



%% UKF SP



S_n    = 0.2604;

S_p    = 0.2570;

k_n    = 37.4312e-12;

k_p    = 17.4733e-12;

R_n    = 2e-6;

R_p    = 2e-6;

D_n    = 29.0798e-15;

D_p    = 27.9034e-15;

c_nmax = 30074.5;

c_pmax = 51563.5;

c_e    = 1000;
```

```
x_navg = 0.9401;

x_pavg = 0.5169;

T      = 298.15;

R_g    = 8.3143;

F      = 96487;

alpha_a  = 0.5;

alpha_c  = 0.5;



P = (diag([S_n,S_p,x_navg,x_pavg]./20))^2;

C = P;

kappa = 0.2;



for i=1:length(second_discharge_time);



   C       = chol(P);

   Chi(:,1) = [S_n; S_p; x_navg; x_pavg];
```

```
Chi(:,2) = [S_n; S_p; x_navg; x_pavg] + ...
    sqrt(4 + kappa)*[C(1,1); 0; 0; 0];
Chi(:,3) = [S_n; S_p; x_navg; x_pavg] + ...
    sqrt(4 + kappa)*[0; C(2,2); 0; 0];
Chi(:,4) = [S_n; S_p; x_navg; x_pavg] + ...
    sqrt(4 + kappa)*[0; 0; C(3,3); 0];
Chi(:,5) = [S_n; S_p; x_navg; x_pavg] + ...
    sqrt(4 + kappa)*[0; 0; 0; C(4,4)];
Chi(:,6) = [S_n; S_p; x_navg; x_pavg] - ...
    sqrt(4 + kappa)*[C(1,1); 0; 0; 0];
Chi(:,7) = [S_n; S_p; x_navg; x_pavg] - ...
    sqrt(4 + kappa)*[0; C(2,2); 0; 0];
Chi(:,8) = [S_n; S_p; x_navg; x_pavg] - ...
    sqrt(4 + kappa)*[0; 0; C(3,3); 0];
Chi(:,9) = [S_n; S_p; x_navg; x_pavg] - ...
    sqrt(4 + kappa)*[0; 0; 0; C(4,4)];
W(1)     = kappa / sqrt(4 + kappa);
```

```
W(2:9)   = 1     / (2*(4+kappa));

W        = W/sum(W);




%%%%%Now the SP measurement model

Iapp = second_discharge_current(i);

J_n  = -Iapp./Chi(1,:);

J_p  = Iapp./Chi(2,:);

x_nsurf = Chi(3,:) - ( J_n * R_n ) / ( 5 * F * D_n * c_nmax);

x_psurf = Chi(4,:) - ( J_p * R_p ) / ( 5 * F * D_p * c_pmax);



U_n     = .8214 + .1387*x_nsurf + .029*x_nsurf.^0.5 - .0172./...

   x_nsurf + ...

   .0019./x_nsurf.^1.5 + .2808 * exp(0.9-15*x_nsurf) -.7984 ...

   * exp (...

   0.4465*x_nsurf - 0.4108);
```

```
U_p     = ( -4.8801 + 88.669 * x_psurf.^2 - 401.119 * x_psurf.^4 ...
+ 342.909 * ...
    x_psurf.^6 - 462.471 * x_psurf.^8 + 433.434 * x_psurf.^10 )...
    ./ ...
    ( -1 + 18.933*x_psurf.^2 - 79.532 * x_psurf.^4 + 37.311 *...
    x_psurf.^6 ...
    - 73.083 * x_psurf.^8 + 95.96*x_psurf.^10 );
eta_n   = R_g * T ./ (F * alpha_a) .* log( ( J_n + (-4*c_e*...
    F.^2*c_nmax.^2*k_n.^2.*x_nsurf.^2 ...
   + 4*c_e*F^2*c_nmax.^2*k_n.^2.*x_nsurf+J_n.^2).^0.5 ) ./ ...
   (2*F*c_e^0.5*k_n.*(c_nmax.*x_nsurf).^0.5 .* ...
   (c_nmax-c_nmax.*x_nsurf).^0.5) );


eta_p   = R_g * T / (F * alpha_c) .* log( ( J_p + (-4*c_e...
    *F^2*c_pmax^2*k_p^2.*x_psurf.^2 ...
   + 4*c_e*F^2*c_pmax^2*k_p^2.*x_psurf+J_p.^2).^0.5 ) ./ ...
```

```
  (2*F*c_e^0.5*k_p.*(c_pmax.*x_psurf).^0.5 .* ...
  (c_pmax-c_pmax.*x_psurf).^0.5) );


zeta = U_p + eta_p - U_n - eta_n;
y_hat    = sum(W.*zeta);
P_yy     = sum(W.*((zeta-y_hat).^2));
y_hat_storage(i) = y_hat;


if i <= 50
   P_xy     = (Chi - repmat([S_n; S_p; x_navg; x_pavg],1,9))...
      .* [W; W; W; W] *(zeta - y_hat)';
   K_k      = P_xy/(P_yy+0.2);
   egg      = [S_n; S_p; x_navg; x_pavg] + K_k*(...
      second_discharge_voltage(i) - y_hat);
   S_n      = egg(1);
   S_p      = egg(2);
   x_navg   = egg(3);
```

```
        x_pavg   = egg(4);



       Q        = 0.4e-4 * ones(4,4);

       P        = P - K_k * P_xy' + Q;

    end

       J_n  = -Iapp./S_n;

       J_p  = Iapp./S_p;

       x_navg    = x_navg - 3 * J_n / (F * R_n * c_nmax) ;

       x_pavg    = x_pavg - 3 * J_p / (F * R_p * c_pmax) ;




end



y_hat_storage = y_hat_storage(1:find(y_hat_storage<2.5,1));

figure

hold on

plot(second_discharge_time(1:length(y_hat_storage)), y_hat_storage,...
```

```
    'k-', 'linewidth', 2)

axis([0 4300 2.4 4])

xlabel('Time (s)')

ylabel('Voltage (V)')

legend('UKF prediction 928.6 (s)', 'observations')

plot([1,4250],2.5*[1,1],'k-','linewidth',1.5) % The EODV line.

text(25,2.55,'EODV failure threshold')

plot(second_discharge_time(1:length(second_discharge_voltage)),...

    second_discharge_voltage,'ko')

axis square

box on

clc

clear all

close all

% Eric Walker

% M.S. thesis PF
```

```
%% PF He, et al model


%Load the data set.


load(['D:\Documents and Settings\Eric\My Documents\spring 2013\'...
    'NASA Ames Data\B0005.mat']);
load(['D:\Documents and Settings\Eric\My Documents\spring 2013\'...
    'NASA Ames Data\B0007.mat']);


theta=[-9.86e-7,5.752e-2,8.983e-1,-8.34e-4]';


first_batt = (-9.86e-7) * exp(5.752e-2 * (1:200)) + (8.983e-1) ...
    * exp((-8.340e-4) * (1:200)) + 0.005*randn(1,200);


second_batt = (-9.86e-7) * exp(5.752e-2 * (1:200)) + (8.983e-1) ...
    * exp((-8.340e-4) * (1:200)) + 0.005*randn(1,200);
```

```
%PF

theta_set=repmat(theta,1,100);

theta_set(1,1:100) = theta(1) + theta(1)/10 * (0.5-rand(100,1));

theta_set(2,1:100) = theta(2) + theta(2)/10 * (0.5-rand(100,1));

theta_set(3,1:100) = theta(3) + theta(3)/10 * (0.5-rand(100,1));

theta_set(4,1:100) = theta(4) + theta(4)/10 * (0.5-rand(100,1));

weights = 0.01 * ones(1,100);



tic

for j = 1:100

   choose_par(j,:) = theta_set(1,j) * exp(theta_set(2,j) * ...

      (1:250)) + theta_set(3,j) * exp(theta_set(4,j)*(1:250));

   RULs(j)        = find(choose_par(j,:) <= 0.8*(second_batt(1)),1);

end

toc
```

```
tic

sigma = 0.1;

for i = 1:200

    if i ==50

        weights_50 = weights;

    end

    if i == 100

        weights_100 = weights;

    end

    if i == 150

        weights_150 = weights;

    end

    % Get the likelihood

    likelihood = 1/(sigma*sqrt(2*pi)) * exp(-1/2 * ...

        ((second_batt(i)) - (theta_set(1,:) .* exp(theta_set(2,:)...

        * i) + theta_set(3,:) .* exp(theta_set(4,:) * i))).^2 /...

        sigma^2);
```

```
    % Update the weights
    weights = weights .* likelihood;
    weights = weights / sum(weights);
end
toc
[RULs, ind] = sort(RULs);
weights_50s = weights_50(ind);
weights_100s = weights_100(ind);
weights_150s = weights_150(ind);

figure

xlabel('k, Cycle index (cycle)')
ylabel('Capacity (Ah)')
axis square
hold on
plot(RULs', weights_50s + 0.8*second_batt(1),'k-', 'linewidth', 2)
```

```
plot(1:length(second_batt), second_batt,'ko','linewidth',1.5)



% Make them range from 0 to 1, otherwise they will be light.
for j = 1:100
    plot(1:50,choose_par(j,1:50),'color',(1-weights_50(j)/max...
        (weights_50))*[1, 1, 1]); % Smaller numbers are darker.
    plot(50:100,choose_par(j,50:100),'color',(1-weights_50(j)/...
        max(weights_100))*[1, 1, 1]); % Smaller numbers are darker.
    plot(100:150,choose_par(j,100:150),'color',(1-weights_50(j)/...
        max(weights_150))*[1, 1, 1]); % Smaller numbers are darker.
    plot(150:200,choose_par(j,150:200), 'color',(1-weights_50(j)/max...
        (weights))*[1, 1, 1]);
end


plot(RULs', weights_50s + 0.8*second_batt(1),'k-', 'linewidth', 2)
plot(1:length(second_batt), second_batt,'ko','linewidth',1.5)
```

```
legend('PF prediction k=50', 'observations')

plot([1,200],second_batt(1)*0.8*[1,1],'k-','linewidth',1.5)

text(25,second_batt(1)*0.81,'EUL failure threshold')

axis([0 200 0.65, 0.91])

%title('PF tracking four states, five percent particle variation')

xlabel('k, Cycle index (cycle)')

ylabel('Q, Capacity (Ah)')

axis square

box on



err_early = sum(weights_50s.*RULs)-190

err_late  = sum(weights_100s.*RULs)-190

err_final = sum(weights_150s.*RULs)-190

sig_early = sqrt(sum(weights_50s.*(RULs - (err_early + 190)).^2) )

sig_late  = sqrt(sum(weights_100s.*(RULs - (err_late + 190)).^2) )

sig_final = sqrt(sum(weights_150s.*(RULs - (err_final + 190)).^2) )
```

```
%% PF Equivalent circuit model

clear all

load(['D:\Documents and Settings\Eric\My Documents\spring 2013\'...

    'NASA Ames Data\B0006.mat']);



global first_discharge_time first_discharge_current...

    first_discharge_voltage;

first_discharge_voltage = B0006.cycle(1,2).data.Voltage_measured(3:end);

first_discharge_time    = B0006.cycle(1,2).data.Time(3:end);

first_discharge_current = B0006.cycle(1,2).data.Current_measured(3:end);




[ecm, resnorm, residuals] = lsqnonlin( @ecm_obj_fun, ...

    [1.17e-8, 2.1, 1795.6, 0.28],[0 0 0 0], [1, 2.5, 2000, 1]);

second_discharge_voltage = B0006.cycle(1,4).data.Voltage_measured(3:end);

second_discharge_time    = B0006.cycle(1,4).data.Time(3:end);

second_discharge_current = B0006.cycle(1,4).data.Current_measured(3:end);
```

```
second_discharge_time    = [second_discharge_time ...
   (second_discharge_time(1:30) + second_discharge_time(end))];
second_discharge_current = [second_discharge_current ...
   second_discharge_current(1:30)];



% PF
sigma = 0.0015;


%In the following lines, set the IG and variation percent.
R = ecm(1) + ecm(1)/10*(0.5-rand(1,50));
Q = ecm(2) + ecm(2)/10*(0.5-rand(1,50));
C = ecm(3)  + ecm(3)/10*(0.5-rand(1,50));
R_ct = ecm(4) + ecm(4)/10*(0.5-rand(1,50));



weights = 0.02*ones(1,50);
```

```
V_cell_storage = [];

tic

for i = 1:224

    % Get the likelihood

    % Quantity inside the square, first

    SOC_cell = 1 + second_discharge_current(i) ./ (Q*3600) .* ...

        second_discharge_time(i);



    SOC_n = 0.79.*SOC_cell + 0.01;

    SOC_p = 0.97-0.51*SOC_cell;



    x_nsurf = SOC_n;

    x_psurf_set = SOC_p;



    U_n     = .7222 + .1387*x_nsurf + .029*x_nsurf.^0.5 - ...

        .0172./x_nsurf + .0019./x_nsurf.^1.5 + .2808 * exp(0.9-...

        15*x_nsurf) -.7984 * exp (0.4465*x_nsurf - 0.4108);
```

```
U_p_set     = ( -4.656 + 88.669 * x_psurf_set.^2 - 401.119 * ...
    x_psurf_set.^4 + 342.909 * ...
    x_psurf_set.^6 - 462.471 * x_psurf_set.^8 + 433.434 * ...
    x_psurf_set.^10 ) ./ ...
    ( -1 + 18.933*x_psurf_set.^2 - 79.532 * x_psurf_set.^4 + ...
    37.311 * x_psurf_set.^6 ...
    - 73.083 * x_psurf_set.^8 + 95.96*x_psurf_set.^10 );


V_o = U_p_set - U_n ;


V_cell = V_o + second_discharge_current(i)*R + Q./C .* ...
    exp(-second_discharge_time(i)./(R_ct .* C))...
    + second_discharge_current(i).*R_ct.*(1-exp(...
    -second_discharge_time(i)./(R_ct .* C)));
V_cell_storage = [V_cell_storage; V_cell];
```

```
if i<=194
quantity = (second_discharge_voltage(i) - V_cell).^2;
likelihood = 1./(sigma*sqrt(2*pi)) .* exp(-1/2 * ...
    (quantity).^2 ./ sigma^2);
% Update the weights
weights = weights .* likelihood;
weights = weights/sum(weights);
end
if i == 50
    weights_50 = weights;
end
if i == 100
    weights_100 = weights;
end
if i == 150
    weights_150 = weights;
end
```

```
end
toc


tic
for j = 1:50
    EODs(j) = find(V_cell_storage(:,j) <= 2.5,1);
    V_cell_storage(EODs(j):end,j) = 0;
end
toc


[EODs ind] = sort(EODs);
weights_50s = weights_50(ind);
weights_100s = weights_100(ind);
weights_150s = weights_150(ind);


figure
```

```
hold on
plot(second_discharge_time(EODs), weights_50s*5 + 2.5,...
    'k-','linewidth',2)
plot(second_discharge_time(1:length(second_discharge_voltage)),...
    second_discharge_voltage, 'ko', 'linewidth', 1.5)
axis([0 4250 2.4 4])
plot([1,4250],2.5*[1,1],'k-','linewidth',1.5) % The EODV line.
text(25,2.55,'EODV failure threshold')
axis square
box on
%title('5 percent variation correct IG')
xlabel('Time (s)')
ylabel('Voltage (V)')
legend('PF prediction 928.6 (s)', 'observations')
for j=1:50
    plot(second_discharge_time(1:50), V_cell_storage(1:50,j),...
        'color', (1-weights_50(j)/max(weights_50))*[1,1,1])
```

```
    plot(second_discharge_time(50:100), V_cell_storage(50:100,j),...
        'color', (1-weights_100(j)/max(weights_100))*[1,1,1])
    plot(second_discharge_time(100:150), V_cell_storage(100:150,j),...
        'color', (1-weights_150(j)/max(weights_150))*[1,1,1])
    plot(second_discharge_time(150:length(V_cell_storage)), ...
        V_cell_storage(150:end,j), 'color', (1-weights_150(j)/...
        max(weights_150))*[1,1,1])

end
plot(second_discharge_time(EODs), weights_50s*5 + 2.5,'k-',...
    'linewidth',2)
plot(second_discharge_time(1:length(second_discharge_voltage))...
    ,second_discharge_voltage, 'ko', 'linewidth', 1.5)

err_early = sum(weights_50.*second_discharge_time(EODs))-3690
err_late  = sum(weights_100.*second_discharge_time(EODs))-3690
err_final = sum(weights_150.*second_discharge_time(EODs))-3690
```

```
sig_early = sqrt(sum(weights_50.*(second_discharge_time(EODs) -...
    (err_early + 3690)).^2) )
sig_late  = sqrt(sum(weights_100.*(second_discharge_time(EODs) -...
    (err_late + 3690)).^2) )
sig_final = sqrt(sum(weights_150.*(second_discharge_time(EODs) -...
    (err_final + 3690)).^2) )
sum(weights_50.*second_discharge_time(EODs))


%%  PF Single Particle model


S_n    = 0.2604;
S_p    = 0.2570;
k_n    = 37.4312e-12;
k_p    = 17.4733e-12;
R_n    = 2e-6;
R_p    = 2e-6;
D_n    = 29.0798e-15;
```

```
D_p    = 27.9034e-15;

c_nmax = 30074.5;

c_pmax = 51563.5;

c_e    = 1000;

x_navg = 0.9401;

x_pavg = 0.5169;

T      = 298.15;

R_g    = 8.3143;

F      = 96485;

alpha_a  = 0.5;

alpha_c  = 0.5;



S_n    = S_n + S_n/10 * (0.5-rand(50,1));

S_p    = S_p + S_p/10 * (0.5-rand(50,1));

x_navg = x_navg + x_navg/10 * (0.5-rand(50,1));

x_pavg = x_pavg + x_pavg/10 * (0.5-rand(50,1));

weights    = 0.02*ones(1,50);
```

```
stop_cycle = length(second_discharge_voltage)+29;

tic

for i = 1:stop_cycle


    Iapp = second_discharge_current(i);

    J_n  = -Iapp./S_n;

    J_p  = Iapp./S_p;

    x_nsurf = x_navg - ( J_n * R_n ) / ( 5 * F * D_n * c_nmax);

    x_psurf = x_pavg - ( J_p * R_p ) / ( 5 * F * D_p * c_pmax);


    U_n     = .8214 + .1387*x_nsurf + .029*x_nsurf.^0.5 - .0172./...

        x_nsurf + ...

        .0019./x_nsurf.^1.5 + .2808 * exp(0.9-15*x_nsurf) -.7984 ...

        * exp (...

        0.4465*x_nsurf - 0.4108);
```

```
U_p     = ( -4.8801 + 88.669 * x_psurf.^2 - 401.119 * x_psurf.^4 ...
+ 342.909 * ...
    x_psurf.^6 - 462.471 * x_psurf.^8 + 433.434 * x_psurf.^10 )...
    ./ ...
    ( -1 + 18.933*x_psurf.^2 - 79.532 * x_psurf.^4 + 37.311 *...
    x_psurf.^6 ...
    - 73.083 * x_psurf.^8 + 95.96*x_psurf.^10 );
eta_n   = R_g * T ./ (F * alpha_a) .* log( ( J_n + (-4*c_e*...
    F.^2*c_nmax.^2*k_n.^2.*x_nsurf.^2 ...
   + 4*c_e*F^2*c_nmax.^2*k_n.^2.*x_nsurf+J_n.^2).^0.5 ) ./ ...
   (2*F*c_e^0.5*k_n.*(c_nmax.*x_nsurf).^0.5 .* ...
   (c_nmax-c_nmax.*x_nsurf).^0.5) );


eta_p   = R_g * T / (F * alpha_c) .* log( ( J_p + (-4*c_e...
    *F^2*c_pmax^2*k_p^2.*x_psurf.^2 ...
   + 4*c_e*F^2*c_pmax^2*k_p^2.*x_psurf+J_p.^2).^0.5 ) ./ ...
   (2*F*c_e^0.5*k_p.*(c_pmax.*x_psurf).^0.5 .* ...
```

```
      (c_pmax-c_pmax.*x_psurf).^0.5) );


   V_cell_set(i,:) = real(U_p + eta_p - U_n - eta_n);


   x_navg    = x_navg - 3 * J_n / (F * R_n * c_nmax) ;
   x_pavg    = x_pavg - 3 * J_p / (F * R_p * c_pmax) ;
end


for j = 1:50
   try
      EOD(j) = find(V_cell_set(:,j) <= 2.5,1);
   catch
      EOD(j) = EOD(j-1);
   end
   V_cell_set(EOD(j):end,j) = 0;
end
toc
```

```
tic

for i=1:(stop_cycle-29)



    weights         = 1/(0.04*sqrt(2*pi)).*exp(-(V_cell_set(i,:)...

        -second_discharge_voltage(i)).^2/(2*0.04^2));

    weights   = weights/sum(weights) ;




    if i==50

            weights_50 = weights;



    end

    if i==100

            weights_100 = weights;

    end

    if i==150

            weights_150 = weights;
```

```
    end


end

toc




figure

axis square

[EOD ind] = sort(EOD);

weights_50s = weights_50(ind);

weights_100s = weights_100(ind);

weights_150s = weights_150(ind);



hold on

box on

xlabel('Time(s)');

ylabel('Voltage(V)');
```

```
plot(second_discharge_time(EOD),weights_50s*5+2.5, 'k-',...
    'linewidth', 2)
plot(second_discharge_time(1:length(second_discharge_voltage)),...
    second_discharge_voltage, 'ko', 'linewidth', 1.5)
legend('PF prediction 928.6 (s)','observations')
for j = 1:50
    plot(second_discharge_time(1:100), V_cell_set(1:100,j), 'color',...
        (1-0.8*weights_50(j)/max(weights_50))*[1,1,1])
    plot(second_discharge_time(50:100), V_cell_set(50:100,j), ...
        'color', (1-0.8*weights_100(j)/max(weights_100))*[1,1,1])
    plot(second_discharge_time(100:150), V_cell_set(100:150,j), ...
        'color', (1-0.8*weights_150(j)/max(weights_150))*[1,1,1])
    plot(second_discharge_time(150:length(V_cell_set)), ...
        V_cell_set(150:end,j), 'color', (1-0.8*weights_150(j)/...
        max(weights_150))*[1,1,1])

end
```

```
plot(second_discharge_time(EOD),weights_50s*5+2.5, 'k-', ...
    'linewidth', 2)


plot(second_discharge_time(1:length(second_discharge_voltage)),...
    second_discharge_voltage, 'ko', 'linewidth', 1.5)
axis([0 4300 2.4 4])
plot([1,4250],2.5*[1,1],'k-','linewidth',1.5) % The EODV line.
text(25,2.55,'EODV failure threshold')
axis square
hold off



err_early = sum(weights_50.*second_discharge_time(EOD))-3690
err_late  = sum(weights_100.*second_discharge_time(EOD))-3690
err_final = sum(weights_150.*second_discharge_time(EOD))-3690
sig_early = sqrt(sum(weights_50.*(second_discharge_time(EOD) -...
```

```
    (err_early + 3690)).^2) )
sig_late  = sqrt(sum(weights_100.*(second_discharge_time(EOD) -...
    (err_late + 3690)).^2) )
sig_final = sqrt(sum(weights_150.*(second_discharge_time(EOD) -...
    (err_final + 3690)).^2) )


function obj = SP_obj_fun(pars,time,current,voltage)


S_n    = 0.2607;
S_p    = 0.2571;
k_n    = 37.4312e-12;
k_p    = 17.4733e-12;
R_n    = 2e-6;
R_p    = 2e-6;
D_n    = 29.0798e-15;
D_p    = 27.9034e-15;
c_nmax = 30074.5;
```

```
c_pmax = 51563.5;

c_e    = 1000;

x_navg = 0.9388;

x_pavg = 0.5171;

T      = 298.15;

R_g    = 8.3143;

F      = 96487;

alpha_a  = 0.5;

alpha_c  = 0.5;



if length(pars) > 1

   S_n = pars(1);

   S_p = pars(2);

end

if length(pars) > 2

   x_navg = pars(3);

   D_pavg = pars(4);
```

```
end
if length(pars) > 4
    k_n = pars(5);
    k_p = pars(6);
end
if length(pars) > 6
    x_navg = pars(7);
    x_pavg = pars(8);
end
if length(pars) > 8
    c_nmax = pars(9);
    c_pmax = pars(10);
end
if length(pars) > 10
    alpha_a = 0.5;
    alpha_c = 0.5;
end
```

```
if length(pars) > 12
    R_n = pars(13);
    R_p = pars(14);
end
if length(pars) > 14
    c_e = pars(15);
    T   = pars(16);
end
V_cell = [];  % The model returns voltage, which is diplayed
% in plots.  The vector is initialized before assigning entries.
% Likewise, initialize the vectors for states of charge.
x_navg_vec = [];
x_pavg_vec = [];


% The model is put into motion.
for i = 1:length(time)  % The model is going to calculate,
    % one time point at an iteration, forward to the end.
```

```
% Calculate the voltage at the current time point, first.

% Then, thedynamic model will reach ahead to prepare the

% changing states of charge for the next loop, the next

% time point.



Iapp    = current(i); % Assign the current at the present

% time point, so it's less bulky in the equations.

J_n     = -Iapp / S_n;

J_p     = Iapp / S_p;

x_nsurf = x_navg - ( J_n * R_n ) / ( 5 * F * D_n * c_nmax);

x_psurf = x_pavg - ( J_p * R_p ) / ( 5 * F * D_p * c_pmax);



% Now, we have enough for the open circuit potentials.

U_n     = .8214 + .1387*x_nsurf + .029*x_nsurf^0.5 - .0172/...

   x_nsurf + ...

   .0019/x_nsurf^1.5 + .2808 * exp(0.9-15*x_nsurf) -.7984 *...

   exp (...
```

```
    0.4465*x_nsurf - 0.4108);


U_p     = ( -4.8811 + 88.669 * x_psurf^2 - 401.119 * x_psurf^4 +...
    342.909 * ...
    x_psurf^6 - 462.471 * x_psurf^8 + 433.434 * x_psurf^10 ) / ...
    ( -1 + 18.933*x_psurf^2 - 79.532 * x_psurf^4 + 37.311 *...
    x_psurf^6 ...
    - 73.083 * x_psurf^8 + 95.96*x_psurf^10 );



eta_n    = R_g * T / (F * alpha_a) * log( ( J_n + (-4*c_e*F^2*...
   c_nmax^2*k_n^2*x_nsurf^2 ...
   + 4*c_e*F^2*c_nmax^2*k_n^2*x_nsurf+J_n^2)^0.5 ) / (2*F*...
   c_e^0.5*k_n*(c_nmax*x_nsurf)^0.5 * ...
   (c_nmax-c_nmax*x_nsurf)^0.5) );


eta_p    = R_g * T / (F * alpha_c) * log( ( J_p + (-4*c_e*...
```

```
    F^2*c_pmax^2*k_p^2*x_psurf^2 ...
    + 4*c_e*F^2*c_pmax^2*k_p^2*x_psurf+J_p^2)^0.5 ) / (2*...
    F*c_e^0.5*k_p*(c_pmax*x_psurf)^0.5 * ...
    (c_pmax-c_pmax*x_psurf)^0.5) );


% Now, the model returns its voltage.
V_cell(i) = U_p + eta_p - U_n - eta_n;


% Prepare the state of charge for the next iteration,
%  based upon the present current and the time step to come.


if i<length(time)  % The conditional statement is
    % necessary because at the very end, 'i + 1' is out of bounds
    % of the data vector.
    t_step    = time(i+1) - time(i);
end % t_step will be left as the last time step, when the end
% of the data vector has passed.
```

```
  % Before changing the SOC, save the current point, for plotting.
  x_navg_vec(i) = x_navg;
  x_pavg_vec(i) = x_pavg;

  x_navg    = x_navg - 3 * J_n / (F * R_n * c_nmax) ;
  x_pavg    = x_pavg - 3 * J_p / (F * R_p * c_pmax) ;

end

obj = voltage - V_cell; % Change to (1:50) for 928.6s estimate.
function [obj]=ecm_obj_fun(theta)

R = theta(1);
Q = theta(2);
C = theta(3);
```

```
R_ct = theta(4);



global first_discharge_time first_discharge_current...
    first_discharge_voltage;
SOC_cell = 1 + first_discharge_current / (3600 * Q) .*...
    first_discharge_time;



SOC_n = 0.79*SOC_cell + 0.01;
SOC_p = 0.97-0.51*SOC_cell;



x_nsurf = SOC_n;
x_psurf_set = SOC_p;



U_n     = .7222 + .1387*x_nsurf + .029*x_nsurf.^0.5 - .0172./...
    x_nsurf + ...
        .0019./x_nsurf.^1.5 + .2808 * exp(0.9-15*x_nsurf) -...
        .7984 * exp (...
```

```
        0.4465*x_nsurf - 0.4108);



U_p_set     = ( -4.656 + 88.669 * x_psurf_set.^2 - 401.119 *...
    x_psurf_set.^4 + 342.909 * ...
        x_psurf_set.^6 - 462.471 * x_psurf_set.^8 + 433.434 *...
        x_psurf_set.^10 ) ./ ...
        ( -1 + 18.933*x_psurf_set.^2 - 79.532 * x_psurf_set.^4 ...
        + 37.311 * x_psurf_set.^6 ...
        - 73.083 * x_psurf_set.^8 + 95.96*x_psurf_set.^10 );



V_o = U_p_set - U_n ;



V_cell = V_o + first_discharge_current.*R + Q./C .* exp(...
    -first_discharge_time./(R_ct * C))...
    + first_discharge_current.*R_ct.*(1-exp(...
    -first_discharge_time./(R_ct * C)));
```

```
obj = first_discharge_voltage - V_cell ;


function V_cell = SP(pars,time,current)


S_n    = 3.41;

S_p    = 3.86;

k_n    = 37.4312e-12;

k_p    = 17.4733e-12;

R_n    = 2e-6;

R_p    = 2e-6;

D_n    = 29.0798e-15;

D_p    = 27.9034e-15;

c_nmax = 30074.5;

c_pmax = 51563.5;

c_e    = 1000;

x_navg = 0.8957971;
```

```
x_pavg = 0.5075848;

T      = 298.15;

R_g    = 8.3143;

F      = 96487;

alpha_a  = 0.5;

alpha_c  = 0.5;

if length(pars) > 0

   S_n = pars(1);

   S_p = pars(2);

end

if length(pars) > 2

   x_navg = pars(3);

   x_pavg = pars(4);

end

if length(pars) > 4

   k_n = pars(5);

   k_p = pars(6);
```

```
end
if length(pars) > 6
    x_navg = pars(7);
    x_pavg = pars(8);
end
if length(pars) > 8
    c_nmax = pars(9);
    c_pmax = pars(10);
end
if length(pars) > 10
    alpha_a = 0.5;
    alpha_c = 0.5;
end
if length(pars) > 12
    R_n = pars(13);
    R_p = pars(14);
end
```

```
if length(pars) > 14
    c_e = pars(15);
    T   = pars(16);
end
V_cell = [];  % The model returns voltage, which is diplayed
% in plots.  The vector is initialized before assigning entries.
% Likewise, initialize the vectors for states of charge.
x_navg_vec = [];
x_pavg_vec = [];

% The model is put into motion.
for i = 1:length(time)  % The model is going to calculate,
    % one time point at an iteration, forward to the end.
    % Calculate the voltage at the current time point, first.
    % Then, the dynamic model will reach ahead to prepare the
    % changing states of charge for the next loop, the next time
    % point.
```

```
Iapp    = current(i); % Assign the current at the present

% time point, so it's less bulky in the equations.

J_n     = -Iapp / S_n;

J_p     = Iapp / S_p;

x_nsurf = x_navg - ( J_n * R_n ) / ( 5 * F * D_n * c_nmax);

x_psurf = x_pavg - ( J_p * R_p ) / ( 5 * F * D_p * c_pmax);



% Now, we have enough for the open circuit potentials.

U_n     = .8214 + .1387*x_nsurf + .029*x_nsurf^0.5 - .0172/...

   x_nsurf + ...

   .0019/x_nsurf^1.5 + .2808 * exp(0.9-15*x_nsurf) -.7984 *...

   exp (0.4465*x_nsurf - 0.4108);



U_p     = ( -4.8811 + 88.669 * x_psurf^2 - 401.119 * x_psurf^4 +...

   342.909 * ...

   x_psurf^6 - 462.471 * x_psurf^8 + 433.434 * x_psurf^10 ) / ...
```

```
    ( -1 + 18.933*x_psurf^2 - 79.532 * x_psurf^4 + 37.311 *...
    x_psurf^6 ...
    - 73.083 * x_psurf^8 + 95.96*x_psurf^10 );



% In order to get the overpotentials, and complete the
% voltage model, root-finding is necessary.  fzero,
% with an anonymous function inside, returns the overpotentials.

eta_n    = R_g * T / (F * alpha_a) * log( ( J_n + (-4*c_e*F^2*...
   c_nmax^2*k_n^2*x_nsurf^2 ...
   + 4*c_e*F^2*c_nmax^2*k_n^2*x_nsurf+J_n^2)^0.5 ) / (2*F*...
   c_e^0.5*k_n*(c_nmax*x_nsurf)^0.5 * ...
   (c_nmax-c_nmax*x_nsurf)^0.5) );

eta_p    = R_g * T / (F * alpha_c) * log( ( J_p + (-4*c_e*...
   F^2*c_pmax^2*k_p^2*x_psurf^2 ...
```

```
    + 4*c_e*F^2*c_pmax^2*k_p^2*x_psurf+J_p^2)^0.5 ) / (2*F*...

    c_e^0.5*k_p*(c_pmax*x_psurf)^0.5 * ...

    (c_pmax-c_pmax*x_psurf)^0.5) );



% Now, the model returns its voltage.

V_cell(i) = U_p + eta_p - U_n - eta_n;



% Prepare the state of charge for the next iteration,

% based upon the present current and the time step to come.



if i<length(time)  % The conditional statement is

    %necessary because at the very end, 'i + 1' is out of

    %bounds of the data vector.

    t_step    = time(i+1) - time(i);

end % t_step will be left as the last time step,

% when the end of the data vector has passed.
```

```
    % Before changing the SOC, save the current point, for plotting.

    x_navg_vec(i) = x_navg;

    x_pavg_vec(i) = x_pavg;



    x_navg    = x_navg - 3 * J_n / (F * R_n * c_nmax) ;

    x_pavg    = x_pavg - 3 * J_p / (F * R_p * c_pmax) ;


end


end
```