AMATH 582 Final: Predicting Remaining Useful Life of Li-Ion Batteries from Discharge Profiles

Brady Griffith

Abstract

Li-Ion batteries are an important tool for scientific instrumentation, but their capacity decreases as they are used. In this project, A reduced dimensional basis is found though SVD to represent the discharge curves in. A linear regression is then used to predict the capacity remaining. In test data, not part of the training, the correct capacity is consistently found within 2% capacity.

Introduction and Overview

Li-ion batteries are everywhere. They're in almost every modern electronic device: smartphones, headphones, and electric cars. But my personal interest is in their application to scientific payloads. Satellites and balloons wouldn't be able to operate on the dark side of the earth without these batteries. They are often exposed to conditions worse than even the most reckless phone owner can concoct. And if you recover these batteries, as is often possible for balloon flights, I'd like to know if these batteries still have life left.

For this analysis, I'm using a dataset from the NASA Ames Research Center. [1] Four 18650 sized Li-ion were repeatedly discharged and recharged with the voltage and current being recorded. This cycle was repeated until each battery had lost 30 % of its original capacity. This dataset also includes Electrochemical Impedance Spectroscopy analysis, which I will ignore, since I've never worked in a lab with an instrument that makes that measurement and wouldn't have much use for it to make predictions.

Because of their prevalence, there is a lot of literature on indicators of battery health. I am deliberately ignoring this, because I would like this project have more of a focus on the analysis techniques that can apply to systems that have been less well characterized.

Theoretical Background

Some basic electrical theory is needed to understand how I reworked the discharge curves to be more general. In the data set, the measurements are presented as time series of voltage and current. If I used them as is, I couldn't apply the results to a system that takes a similar measurement, but with a different constant current or a constant discharge resistance (which is what my undergrad lab's battery characterizer used.) It also wouldn't generalize to larger capacity lithium ion batteries. Instead, I would like to preform the analysis on the voltage as a function of the percentage discharge.

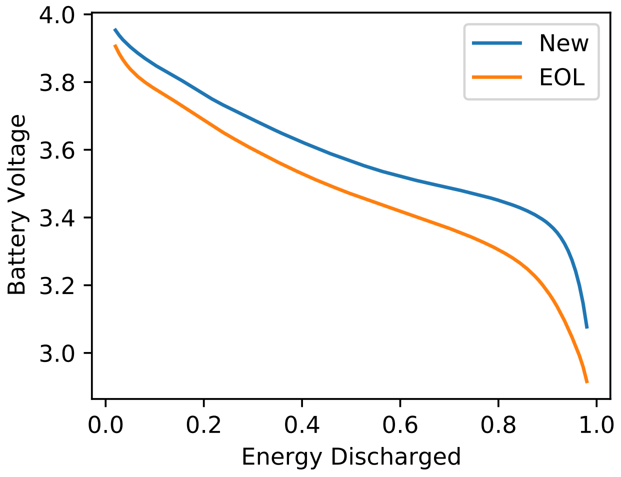

Figure 1: Two example discharge curves for Battery 5. One curve is a new battery with full capacity and the second is at the battery end of life (EOL) when it has lost 30% of its starting capacity.

The power consumed from a battery can be found from the current,$I(t)$, and voltage, $V(t)$. by

$$P(t) = I(t)V(t)$$

The power consumed from the start at t_0 up to a time t is given by

$$W(t) = \int_{t_0}^{t} P(t')\,\mathrm{d}t'$$

If the battery is fully discharged by t_f, the fraction discharged $C(t)$ is then

$$C(t) = W(t)/W(t_f)$$

I also define a remaining capacity value for curve i defined to be

$$R_i \equiv C_i(t_f)/C_{\max}(t_f) \tag{1}$$

With the curves expressed in terms of a preferred independent variable, I can now reduce the dimensionality of the system through SVD. The discharge curves all have the same general shape, with some changes occurring as they age (See figure 1). For this reason I would expect the system to represented well by a fairly low dimension system.

Looking at these changes to the discharge curve (see figure 2), the leading order terms can by expressed as a linear function of the aging. This model can be written out

$$R = \frac{1}{k}\sum_{i}^{k} a_i x_i + b_i \tag{2}$$

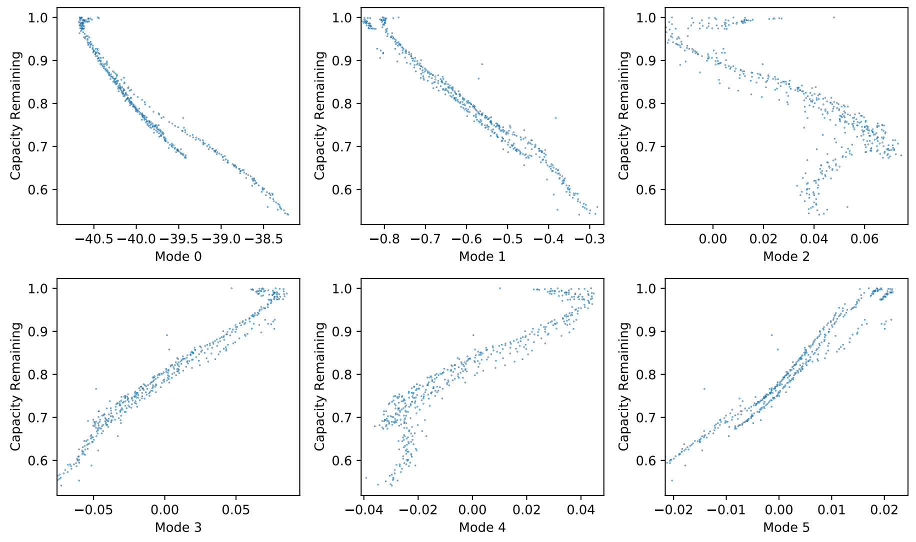

Figure 2: A scatter plot of how the battery capacity changes with the first 6 SVD modes. This has been cropped to highlight main distribution, so there are outliers not shown.

$$\mathbf{R} = \begin{bmatrix} \mathbf{x_0} & 1 \\ \mathbf{x_1} & 1 \\ \vdots & \vdots \\ \mathbf{x_m} & 1 \end{bmatrix} \begin{bmatrix} a_0/k \\ a_1/k \\ \vdots \\ a_k/k \\ \frac{1}{k}\sum_i^k b_i \end{bmatrix} \tag{3}$$

$$\mathbf{R} = \mathbf{X}\mathbf{b}$$

Which is now a regression problem and can be solved least squares or any of the different norm methods discussed in chapter 18 of the book [2].

Algorithm Implementation and Development

Generality to other batteries and discharge curve recorders is a major objective in my analysis, so I would like to put the curves into a form independent of the sampling rate. After calculating $C(t)$ for each recording, I define linearly spaced set of 128 points over $C(t) = (.02, .98)$. The first and last 2% were ignored because battery voltages bounce back after the load is removed, and that is not part of this analysis.

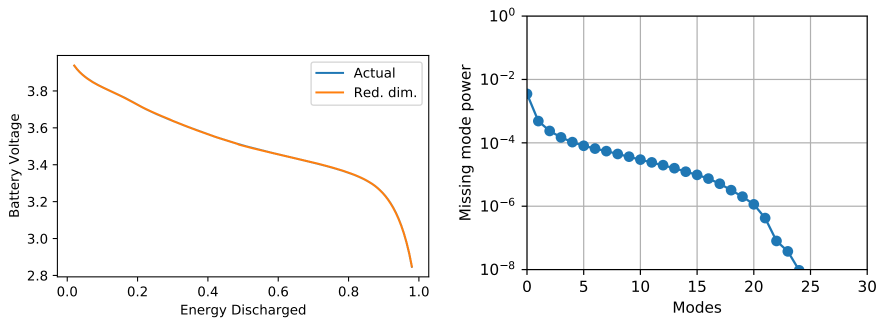

Figure 3: On the right, fraction of variance contained in n svd modes. On the left, an example discharge curve represented by 20 modes.

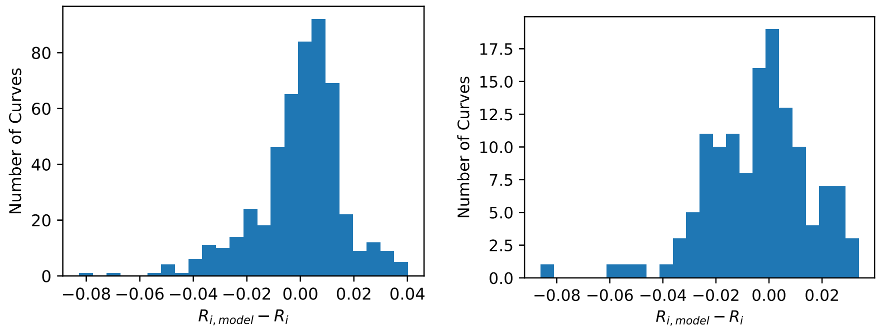

Figure 4: On the left, the distribution of errors for the model with the training data. On the right with the test data.

I preform the model fitting using the curves from batteries 5, 6, and 7. Battery 18 is reserved for testing. The regression used here was least squares. There are outlier, so it may have benefitted from use of a Lasso regression instead.

Computational Results

First, I need to determine the how many SVD modes are needed. In figure 3 I plot the fraction of total covariance in n modes. 20 modes excluded $< 10^6$ of total covariance and visually recreates the curve to a subjectively good quality.

The weights for the model is found. The predicted values for the training set consistently fall within a range of 2% capacity remaining. The performance on the battery not used in the training data is slightly worse, but still . See the distribution in figure 4.

Summary and Conclusions

In this project, SVD was first used create an ideal basis to represent Li-Ion discharge curves. Then once in this basis, a linear regression is to determine the battery capacity, and thus remaining life. For the test data, not part of the training, the regression could accurately predict the capacity an error $\pm 2\%$ capacity. This would accurate enough to make decisions about when a battery needs to be replaced.

Another test that wasn't performed for this project, but that would be an important next step is checking how well these results generalize to another lithium ion battery. I have a couple discharge curves saved from an undergrad lightning ballooning project that could have been used for this.

References

[1] D. Mcintosh, "Li-ion battery aging datasets." "https://c3.nasa.gov/dashlink/resources/133/", 2010.

[2] J. N. Kutz, *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford: Oxford University Press, first edition ed., 2013. OCLC: ocn858608087.

Here is a link to the Github repository for this project.

Python Functions

loadCurves

create_nasa_curves

`create_nasa_curves()`

Creates a matrix of all NASA battery curves, and its labels

Creates each curve as a row in in a large X matrix. Each curve is voltage interpolated to be in 128 steps as a function of the power delivered from 2% to 98%.

load_nasa_curves

`load_nasa_curves(to_load)`

Loads the curves cached to disk

Returns:

- `X` *ndarray* - Matrix of curves
- `p` *ndarray* - The discharge percentage that X rows are a function of
- `labels` *ndarray* - Whuch battery each curve is
- `capacity` *ndarray* - The capacity of each battery during each curve

main

plot_aging

`plot_aging(p, X, labels, capacity)`

Plot examples of new and old battery curves.

Arguments:

- `X` *ndarray* - Matrix of curves
- `p` *ndarray* - The discharge percentage that X rows are a function of
- `labels` *ndarray* - Whuch battery each curve is
- `capacity` *ndarray* - The capacity of each battery during each curve

svd

plot_mode_fraction

`plot_mode_fraction(s)`

Plots the fraction of power represented with n modes.

Arguments:

- `s` *array-like* - 1D arrray of the variances of the principal components.

plot_n_modes

```
plot_n_modes(p, X, V, n)
```

Shows the numbers represented with the selected number of SVD modes.

Arguments:

- `p` *array_like* - The discharge percentage that X rows are a function of.
- `X` *array_like* - Data matrix with rows of images.
- `V` *array_like* - Matrix with mode vectors as columns.
- `n` *int* - Number of modes to use in the representation.

plot_mode_vs_life

```
plot_mode_vs_life(capacity, X, V, n)
```

Plots how the capacity changes for values of each SVD mode.

Arguments:

- `capacity` *ndarray* - The capacity of each battery during each curve.
- `X` *ndarray* - Matrix of curves.
- `V` *array_like* - Matrix with mode vectors as columns.
- `n` *int* - Number of modes to plot.

predict

find_weights

```
find_weights(X, b)
```

Calculate the weights of the linear regression.

Arguments:

- `X` *array-like* - Matrix with rows of curves.
- `b` *array-like* - The capacity for each curve.

Returns:

- `ndarray` - Weights for the model.

predict

```
predict(X, w)
```

Using previously calculated weights, predict the capacity of each curve.

Arguments:

- `X` *array-like* - Matrix with rows of curves.
- `w` *array-like* - Weights for the model.

Returns:

- `ndarray` - Predicted Capacities

accuracy_dist

`accuracy_dist(b_real, b_model, path_out)`

Plots the error for two sets of capacties

Arguments:

- `b_real` *array-like* - The capacity actually measured.
- `b_model` *array-like* - Predicted capacities.

Python Code

main.py

```python
import numpy as np
import matplotlib.pyplot as plt
import loadCurves
import svd
import predict


def plot_aging(p, X, labels, capacity):
    """Plot examples of new and old battery curves.

    Args:
        X (ndarray): Matrix of curves
        p (ndarray): The discharge percentage that X rows are a function of
        labels (ndarray): Whuch battery each curve is
        capacity (ndarray): The capacity of each battery during each curve
    """
    fig, ax = plt.subplots(figsize=(4, 3))
    ax.set_xlabel('Energy Discharged')
    ax.set_ylabel('Battery Voltage')

    bat_sel = labels == 5
    curve_new = X[bat_sel][np.argmax(capacity[bat_sel])]
    curve_old = X[bat_sel][np.argmin(capacity[bat_sel])]
    ax.plot(p, curve_new, label='New')
    ax.plot(p, curve_old, label='EOL')
    ax.legend()
    fig.savefig('Final/figures/aging_curves.pdf', bbox_inches='tight')


def run_analysis():
    loadCurves.create_nasa_curves()
    X, p, labels, capacity = loadCurves.load_nasa_curves([5, 6, 7])
    plot_aging(p, X, labels, capacity)

    u, s, vh = np.linalg.svd(X)
    svd.plot_mode_fraction(s)
    k = 20
    svd.plot_n_modes(p, X, vh, k)
    svd.plot_mode_vs_life(capacity, X, vh, 6)

    Y = np.dot(vh, X.T)[:k].T
    w = predict.find_weights(Y, capacity)
    print(w)
    cap_pred = predict.predict(Y, w)
    predict.accuracy_dist(capacity, cap_pred, 'Final/figures/train-pref.pdf')

    X_tst, _, _, cap_tst = loadCurves.load_nasa_curves([18])
    Y = np.dot(vh, X_tst.T)[:k].T
    cap_pred = predict.predict(Y, w)
    predict.accuracy_dist(cap_tst, cap_pred, 'Final/figures/test-pref.pdf')


if __name__ == '__main__':
    run_analysis()
```

loadCurves.py

```
import numpy as np
import pandas as pd
from scipy import io


def create_nasa_curves():
    """Creates a matrix of all NASA battery curves, and its labels

    Creates each curve as a row in in a large X matrix. Each curve is voltage
    interpolated to be in 128 steps as a function of the power delivered from
    2% to 98%.
```

```
"""
battery_files = ['B0005.mat', 'B0006.mat', 'B0007.mat', 'B0018.mat']

X = None
p = np.linspace(.02, .98, 128)
labels = None
capacity = None

for bat_file in battery_files:
    bat_path = 'Final/data/battery/' + bat_file
    matlab_file = io.loadmat(bat_path, simplify_cells=True)
    bat_str = list(matlab_file.keys())[-1]
    label = int(bat_str[1:])

    discharge_cycles = [cycle for cycle in matlab_file[bat_str]['cycle']
                        if cycle['type'] == 'discharge']

    N = len(discharge_cycles)
    X_bat = np.zeros((N, len(p)))
    l_bat = np.zeros(N, dtype=np.int8)
    cap_bat = np.zeros(N)

    for i, cycle in enumerate(discharge_cycles):
        df = pd.DataFrame(cycle['data'])

        power = -(df['Voltage_measured'])*(df['Current_measured'])
        dt = np.roll(df['Time'], -1) - df['Time']
        dt[0] = 0
        energy = np.cumsum(power*dt)
        energy_frac = energy/np.max(energy)
        V = np.interp(p, energy_frac, df['Voltage_measured'])
        X_bat[i, :] = V
        l_bat[i] = label
        cap_bat[i] = np.max(energy)

    cap_bat /= np.max(cap_bat)

    X = X_bat if X is None else np.append(X, X_bat, axis=0)
    labels = l_bat if labels is None else np.append(labels, l_bat)
    capacity = cap_bat if capacity is None else np.append(capacity, cap_bat)

np.save('Final/output/X.npy', X)
np.save('Final/output/p.npy', p)
np.save('Final/output/labels.npy', labels)
np.save('Final/output/capacity.npy', capacity)
```

```python
def load_nasa_curves(to_load):
    """Loads the curves cached to disk

    Returns:
        X (ndarray): Matrix of curves
        p (ndarray): The discharge percentage that X rows are a function of
        labels (ndarray): Whuch battery each curve is
        capacity (ndarray): The capacity of each battery during each curve
    """
    X = np.load('Final/output/X.npy')
    p = np.load('Final/output/p.npy')
    labels = np.load('Final/output/labels.npy')
    capacity = np.load('Final/output/capacity.npy')

    mask = np.isin(labels, to_load)
    return X[mask], p, labels[mask], capacity[mask]


if __name__ == '__main__':
    create_nasa_curves()
```

svd.py

```python
import numpy as np
import matplotlib.pyplot as plt


def plot_mode_fraction(s):
    """Plots the fraction of power represented with n modes.

    Args:
        s (array-like): 1D arrray of the variances of the principal components.
    """
    fig, ax = plt.subplots(figsize=(4, 3))
    ax.set_xlim(0, 30)
    ax.set_ylim(1e-8, 1)
    ax.set_yscale('log')
    ax.grid()
    f = 1 - np.cumsum(s**2) / np.sum(s**2)

    ax.set_xlabel('Modes')
    ax.set_ylabel('Missing mode power')

    ax.plot(f, marker='o')
    fig.savefig('Final/figures/mode_frac.pdf', bbox_inches='tight')
```

```python
def plot_n_modes(p, X, V, n):
    """Shows the numbers represented with the selected number of SVD modes.

    Args:
        p (array_like): The discharge percentage that X rows are a function of.
        X (array_like): Data matrix with rows of images.
        V (array_like): Matrix with mode vectors as columns.
        n (int): Number of modes to use in the representation.
    """
    fig, ax = plt.subplots(figsize=(5, 3))
    ax.set_xlabel('Energy Discharged')
    ax.set_ylabel('Battery Voltage')

    selected = -1
    Y = np.dot(V, X[selected])
    Z = np.dot(V[:n].T, Y[:n]).T

    ax.plot(p, X[selected], label='Actual')
    ax.plot(p, Z, label='Red. dim.')
    ax.legend()
    fig.savefig('Final/figures/red_dim.pdf', bbox_inches='tight')


def plot_mode_vs_life(capacity, X, V, n):
    """Plots how the capacity changes for values of each SVD mode.

    Args:
        capacity (ndarray): The capacity of each battery during each curve.
        X (ndarray): Matrix of curves.
        V (array_like): Matrix with mode vectors as columns.
        n (int): Number of modes to plot.
    """
    Y = np.dot(V, X.T)[:n]

    fig, axs = plt.subplots(int(np.ceil(n/3)), 3, figsize=(10, 6))

    for n, (C, ax) in enumerate(zip(Y, axs.flatten())):
        ax.set_xlabel(f'Mode {n}')
        ax.set_ylabel('Capacity Remaining')
        t = .15
        l = np.quantile(C, t)
        r = np.quantile(C, 1-t)
        ax.set_xlim(l - .2*(r-l), r + .2*(r-l))
        ax.scatter(C, capacity, s=.1)
    fig.tight_layout()
    fig.savefig('Final/figures/mode_proj.pdf', bbox_inches='tight')
```

predict.py

```
import numpy as np
import matplotlib.pyplot as plt


def find_weights(X, b):
    """Calculate the weights of the linear regression.

    Args:
        X (array-like): Matrix with rows of curves.
        b (array-like): The capacity for each curve.
    Returns:
        ndarray: Weights for the model.
    """
    X = np.append(X, np.ones((X.shape[0], 1)), axis=1)
    w, _, _, _ = np.linalg.lstsq(X, b)
    return w


def predict(X, w):
    """Using previously calculated weights, predict the capacity of each curve.

    Args:
        X (array-like): Matrix with rows of curves.
        w (array-like): Weights for the model.

    Returns:
        ndarray: Predicted Capacities
    """
    X = np.append(X, np.ones((X.shape[0], 1)), axis=1)
    return np.dot(X, w)


def accuracy_dist(b_real, b_model, path_out):
    """Plots the error for two sets of capacties

    Args:
        b_real (array-like): The capacity actually measured.
        b_model (array-like): Predicted capacities.
    """
    fig, ax = plt.subplots(figsize=(4, 3))
```

```
ax.set_xlabel(r'$R_{i,model} - R_i$')
ax.set_ylabel('Number of Curves')

error = b_model - b_real
print(np.std(error))
ax.hist(error, bins=24)
fig.savefig(path_out, bbox_inches='tight')
```