

---

# NASA battery degradation data

## Table of Contents

Load the data .....	1
Create a table with the data for easier access .....	1
Populate the table with the contents of the structure. ....	2
General housekeeping and clean up. ....	3
Timetable conversion .....	4
Looking for impedance signatures of capacity fade .....	4
Visualization .....	4
R0 and R1 .....	5
Relative change in R1 and R0 .....	5
Input data preparation .....	6
Define training and testing .....	6
Visualize predicted and predictors .....	6
Classification .....	7
Regression .....	7
Parametric modeling - Linear model .....	8
Nonparametric modeling - Decision tree .....	8
Nonparametric modeling - Neural Network .....	9
X - Y plot for comparison between fit methods and measurement .....	10
Regression Performance Comparison .....	11
Impedance Plot .....	12
Looking for impedance signature of capacity fade .....	12

This script loads impedance data from the NASA open repository and performs manipulations for visualization and analysis. B. Saha and K. Goebel (2007). "Battery Data Set", NASA Ames Prognostics Data Repository (<http://ti.arc.nasa.gov/project/prognostic-data-repository>), NASA Ames Research Center, Moffett Field, CA <http://ti.arc.nasa.gov/tech/dash/pcoc/prognostic-data-repository/>

## Load the data

```
clear
dir = 'C:\Work\Projects\Aging\dataAnalytics\Sundar';
myData = load([dir '\B0006.mat']);
fieldName = char(fieldnames(myData));
% Create a structure with the data of interest
myData = myData.(fieldName);
rng('default');
```

## Create a table with the data for easier access

Determine how many sets of data the structure contains.

```
n = size(myData.cycle,2);
% Pre allocate the table.
t = cell2table(cell(n,8));
% Define the variable names (column names)
```

```
t.Properties.VariableNames = {...  
    'Cycle', 'Type', 'AmbientTemp', 'Capacity', 'R0', 'R1', 'Time', 'TestData'};
```

## Populate the table with the contents of the structure.

```
for i = 1:n  
    % Note the cycle  
    t.Cycle{i} = i;  
  
    % Add the type.  
    t.Type{i} = myData.cycle(i).type;  
  
    % Add Ambient Temp.  
    t.AmbientTemp{i} = myData.cycle(i).ambient_temperature;  
  
    % Add Time.  
    t.Time{i} = myData.cycle(i).time;  
  
    % Deal with different test data based on the cycle type.  
    switch(t.Type{i})  
  
        case('charge')  
            % Collect the Test Data into a table within the main  
            table.  
            t.TestData{i} = array2table([...  
                myData.cycle(i).data.Time', ...  
                myData.cycle(i).data.Voltage_measured', ...  
                myData.cycle(i).data.Current_measured', ...  
                myData.cycle(i).data.Temperature_measured', ...  
                myData.cycle(i).data.Current_charge', ...  
                myData.cycle(i).data.Voltage_charge'], ...  
                'VariableNames', ...  
                {'Time', 'VoltageMeas', 'CurrentMeas', ...  
                 'TempMeas', 'CurrentCharge', 'VoltageCharge'});  
            t.Capacity{i} = NaN;  
            t.R0{i} = NaN;  
            t.R1{i} = NaN;  
  
        case('discharge')  
            % Collect the Test Data into a table within the main  
            table.  
            t.TestData{i} = array2table([...  
                myData.cycle(i).data.Time', ...  
                myData.cycle(i).data.Voltage_measured', ...  
                myData.cycle(i).data.Current_measured', ...  
                myData.cycle(i).data.Temperature_measured', ...  
                myData.cycle(i).data.Current_load', ...  
                myData.cycle(i).data.Voltage_load'], ...  
                'VariableNames', ...
```

```
        {'Time','Voltage_measured','Current_measured',...
'Temperature_measured','Current_load','Voltage_load'}));

% Test to see if "Capacity" is included in this data.
if isfield(myData.cycle(i).data,'Capacity')
    t.Capacity{i} = myData.cycle(i).data.Capacity;
else
    t.Capacity{i} = NaN;
end

t.R0{i} = NaN;
t.R1{i} = NaN;

case('impedance')
    % Make sure the length of "Rectified_Impedance" is the
same as
    % the rest of the test data. It seems like it is always
    % shorter than the rest of the data. If it is shorter,
pad it
    % with NaN's.
    k = length(myData.cycle(i).data.Rectified_Impedance);
    j = length(myData.cycle(i).data.Battery_impedance);

    RectImp_pad = [...
        myData.cycle(i).data.Rectified_Impedance;...
        NaN(j-k,1)];

    % Collect the Test Data into a table within the main
table.
    t.TestData{i} = array2table([...
        myData.cycle(i).data.Sense_current',...
        myData.cycle(i).data.Battery_current',...
        myData.cycle(i).data.Current_ratio',...
        myData.cycle(i).data.Battery_impedance,...
        RectImp_pad],...
        'VariableNames',...
        {'Sense_current','Battery_current','Current_ratio',...
        'Battery_impedance','Rectified_Impedance'}));

    t.Capacity{i} = NaN;
    t.R0{i} = myData.cycle(i).data.Re;
    t.R1{i} = myData.cycle(i).data.Rct;
end
end
```

## General housekeeping and clean up.

```
t.Cycle = cell2mat(t.Cycle);
t.Type = categorical(t.Type);
t.AmbientTemp = cell2mat(t.AmbientTemp);
t.Capacity = cell2mat(t.Capacity);
```

```
t.R0 = cell2mat(t.R0);  
t.R1 = cell2mat(t.R1);  
t.Time = cell2mat(t.Time);
```

## Timetable conversion

```
t.newTime = datetime(zeros(length(t.Time),6));  
for i=1:length(t.Time)  
    t.newTime(i) = datetime(t.Time(i,:));  
end  
t.Properties.VariableNames{7} = 'oldTime';  
tt = table2timetable(t, 'RowTimes', 'newTime');  
tt.oldTime = [];
```

## Looking for impedance signatures of capacity fade

Find location of Z measurements followed by discharge

```
idxZ_0 = tt.Type == 'impedance';  
% build ordinal vector to find cycle # of each Z experiment  
idx1 = (1:length(tt.TestData)).*idxZ_0;  
% keep the non-zeros  
idx2 = nonzeros(idx1);  
% Tests begin with charge and discharge cycles. At cycle 41 impedance  
% is  
% recorded for the first time. Plotting capacity (only recorded at  
% discharge) vs. R0 or R1 requires some index manipulation.  
% Let's try to identify which impedance measurements are immediately  
% followed by discharge measurements (i.e. by capacity measurements)  
% idxZ corresponds to impedance experiments followed by capacity  
% measurement.  
idx3 = idx2 .* (tt.Type(idx2 + 1) == 'discharge');  
idxZ = nonzeros(idx3);
```

## Visualization

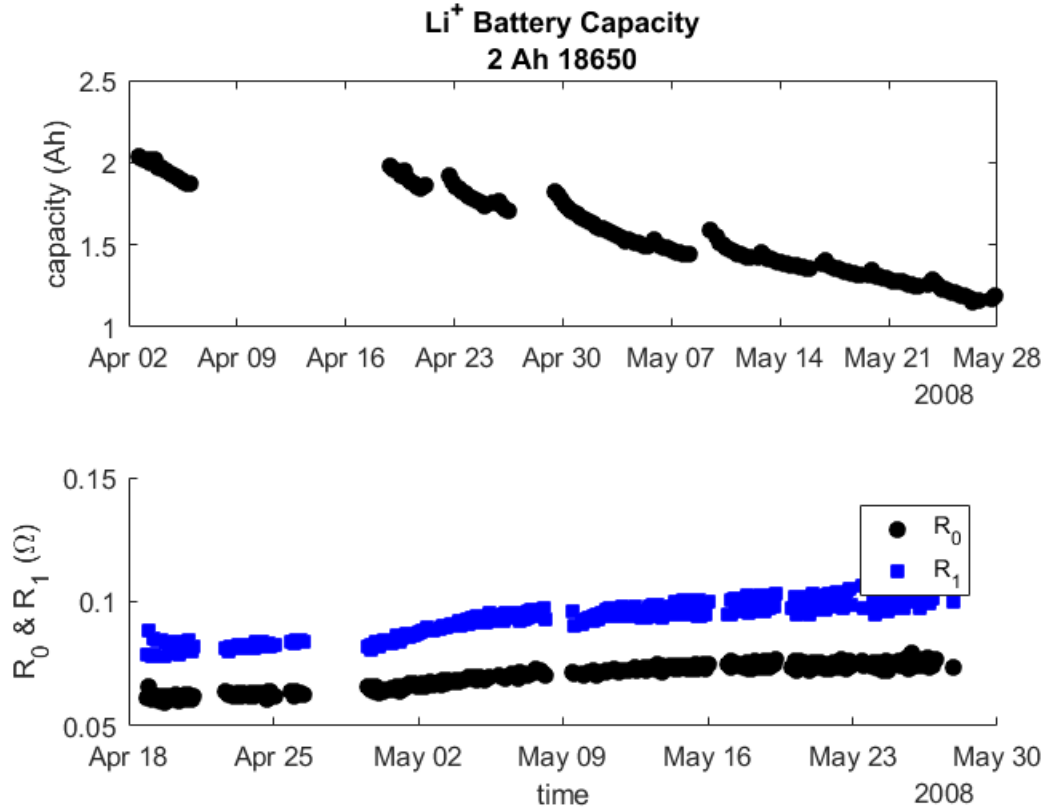
Capacity. Look for nan's in capacity data

```
idxCap = isnan(tt.Capacity);  
% Plot capacity when available  
f1 = figure(1); clf(f1);  
subplot(2,1,1)  
plot(tt.newTime(~idxCap), tt.Capacity(~idxCap), 'ok', 'MarkerFaceColor', 'k')  
ylabel('capacity (Ah)')  
% xlabel('time')  
title({'Li+ Battery Capacity' ; '2 Ah 18650'})  
  
% Resistances  
subplot(2,1,2)  
hold on
```

```

plot(tt.newTime(idx2),tt.R0(idx2),'ok','MarkerFaceColor','k')
plot(tt.newTime(idx2),tt.R1(idx2),'sb','MarkerFaceColor','b')
legend({'R_0' 'R_1'})
ylabel('R_0 & R_1 (\Omega)')
xlabel('time')

```



## R0 and R1

```

f3 = figure(3); clf(f3) hold on scatter(tt.Cycle(idx2),tt.R0(idx2),'.') scatter(tt.Cycle(idx2),tt.R1(idx2),'.','r')
legend({'R_0' 'R_1'}) ylabel('R_0 & R_1 (\Omega)') xlabel('cycle')

```

```

% Capacity vs. (R1 and R0)
% f4 = figure(4); clf(f4); hold on
% scatter(tt.Capacity(idxZ(1)+1) - tt.Capacity(idxZ
+1),tt.R1(idxZ),'.')
% scatter(tt.Capacity(idxZ(1)+1) - tt.Capacity(idxZ
+1),tt.R0(idxZ),'.','r')
% xlabel('Capacity fade (initial - final) (Ah)')
% ylabel('R_1(\Omega blue)   R_0(\Omega red)')

```

## Relative change in R1 and R0

```

R1_norm = tt.R1(idxZ(1))./tt.R1(idxZ); R0_norm = tt.R0(idxZ(1))./tt.R0(idxZ); f5 = figure(5);
clf(f5); hold on scatter(R0_norm,R1_norm,'.') xlabel('R_0^i^n^i^t/R_0') ylabel('R_1^i^n^i^t/R_1')
xlim([min(R1_norm) 1]) ylim([min(R1_norm) 1]) axis square line([0 1],[0 1])

```

## Input data preparation

Define predictor and objective. Must be an array.

```
x = [tt.R0(idxZ) tt.R1(idxZ)]; % R0 and R1
y = tt.Capacity(idxZ+1); % Capacity
```

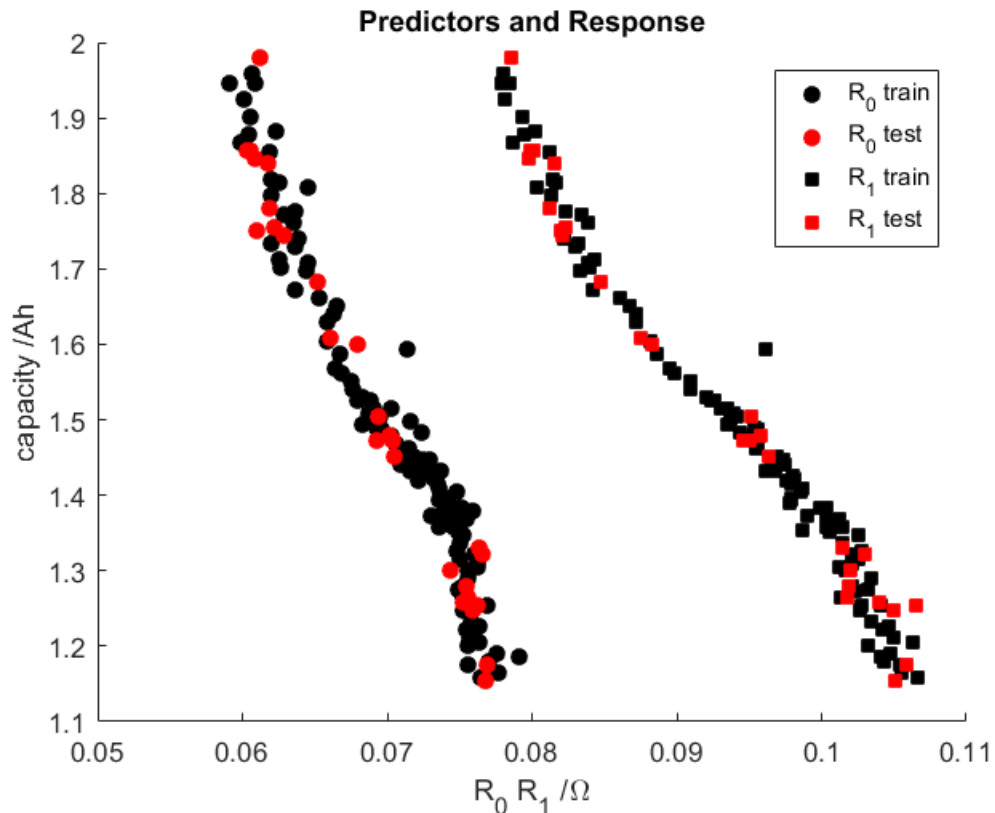
## Define training and testing

Split between training and testing using cvpartition 80% for training and 20% for testing/validation

```
testFraction = 0.2;
dataPartition = cvpartition(length(x), 'HoldOut', testFraction);
Xtrain = x(dataPartition.training, :); % R1 and R0
Ytrain = y(dataPartition.training, :); % Capacity
Xtest = x(dataPartition.test, :); % R1 and R0
Ytest = y(dataPartition.test, :); % Capacity
```

## Visualize predicted and predictors

```
f10 = figure(10); clf(f10); hold on
f10 = plot(Xtrain(:,1), Ytrain, 'ok', 'MarkerFaceColor', 'k'); hold on
      plot(Xtest(:,1), Ytest, 'or', 'MarkerFaceColor', 'r')
      plot(Xtrain(:,2), Ytrain, 'sk', 'MarkerFaceColor', 'k'); hold on
      plot(Xtest(:,2), Ytest, 'sr', 'MarkerFaceColor', 'r')
xlabel('R_0 R_1 /\Omega')
ylabel('capacity /Ah')
legend({'R_0 train' 'R_0 test' 'R_1 train' 'R_1 test'})
title('Predictors and Response')
```



## Classification

First, we classify the capacity in 3 categories: 1 = excellent 2 = good 3 = poor This requirement lends itself to classification techniques that can be explored using the Classification Learner App. Developer needs to decide whether to use the nominal 1C capacity or the first measured (pristine state) value

```
nominalCapacity = 2; %tt.Capacity(idxZ(1)+1);
tt.Group = zeros(height(tt),1);
tt.Group(tt.Capacity/nominalCapacity > 0.9) = 1;
tt.Group(tt.Capacity/nominalCapacity < 0.9 & ...
    tt.Capacity/nominalCapacity > 0.7) = 2;
tt.Group(tt.Capacity/nominalCapacity < 0.7) = 3;
z = tt.Group(idxZ+1); % Categorical Capacity
% classificationTable = table(x(:,1),x(:,2),z,'VariableNames',{'R0'
    'R1' 'catCapacity'});
classificationTable = table(tt.R0(idxZ),tt.R1(idxZ),tt.Group(idxZ
+1),...
    'VariableNames',{'R0' 'R1' 'catCapacity'});
```

## Regression

Now we try a few regression functions, both parametric and non-parametric

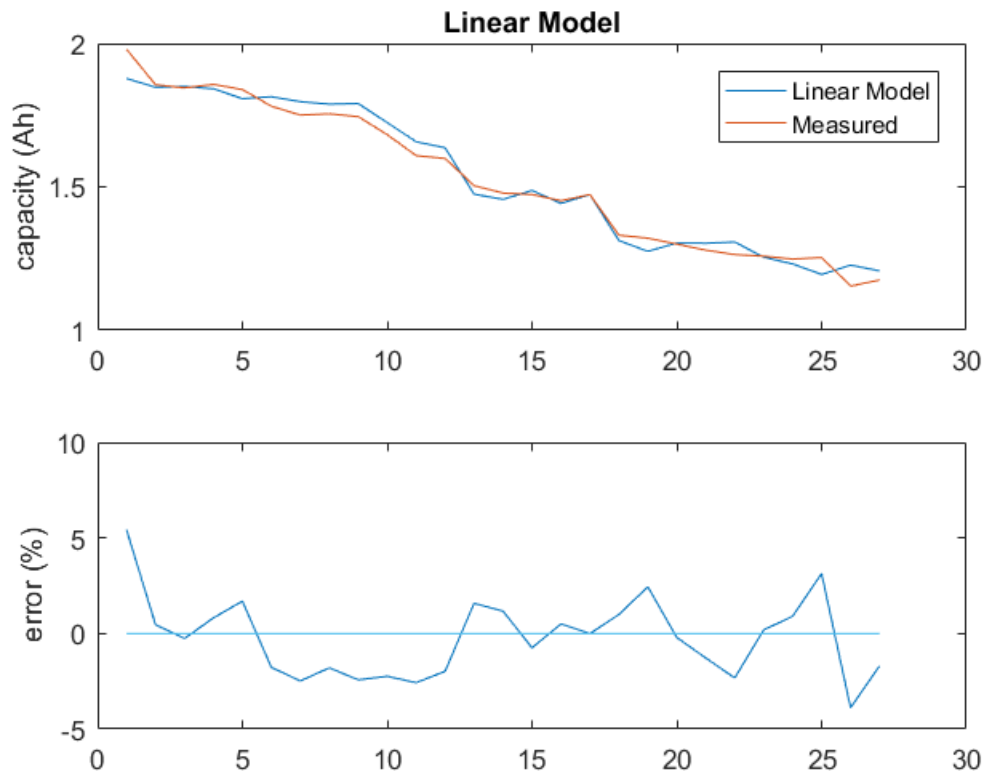
## Parametric modeling - Linear model

```
mdl_lm = fitlm(Xtrain,Ytrain);
pred_lm = predict(mdl_lm,Xtest);

fprintf('Linear Model Training set MAPE: %0.2f%%\n', ...
    mape(Ytest, predict(mdl_lm, Xtest))*100);

f9 = plotPrediction(pred_lm,Ytest,9,'Linear Model');

Linear Model Training set MAPE: 2.08%
```



## Nonparametric modeling - Decision tree

```
mdl_dt = fitrtree(Xtrain,Ytrain);%,'OptimizeHyperparameters','auto');
pred_dt = predict(mdl_dt,Xtest);

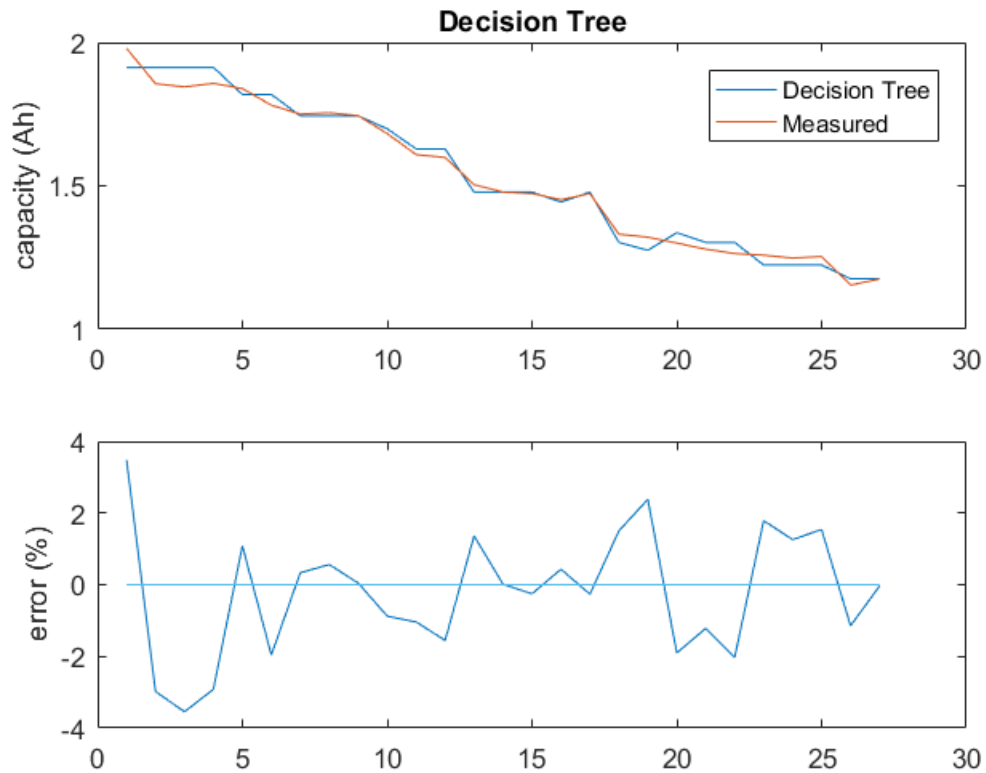
f8 = plotPrediction(pred_dt,Ytest,8,'Decision Tree');

fprintf('Tree Training set MAPE: %0.2f%%\n', ...
    mape(Ytest, predict(mdl_dt, Xtest))*100);

% view(mdl_dt,'Mode','graph')

Tree Training set MAPE: 1.73%
```





## Nonparametric modeling - Neural Network

```

trainFcn = 'trainlm'; % Levenberg-Marquardt training algorithm
hiddenLayerSize = 20;
net = fitnet(hiddenLayerSize,trainFcn);

% Train the Network
mdl_net = train(net,Xtrain',Ytrain','UseParallel','no');

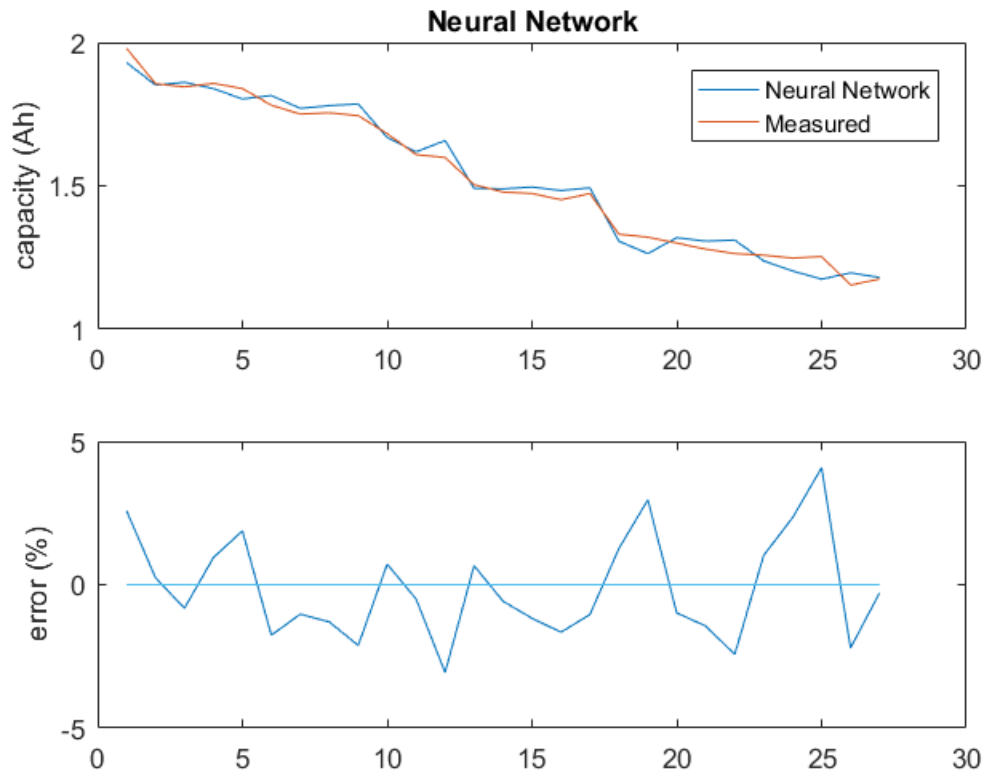
fprintf('Neural Net Training set MAPE: %0.2f%%\n', ...
        mape(Ytest, mdl_net(Xtest'))*100);

% make predictions on the test data set and plot results
Y_nn = mdl_net(Xtest')';

f6 = plotPrediction(Y_nn,Ytest,6,'Neural Network');

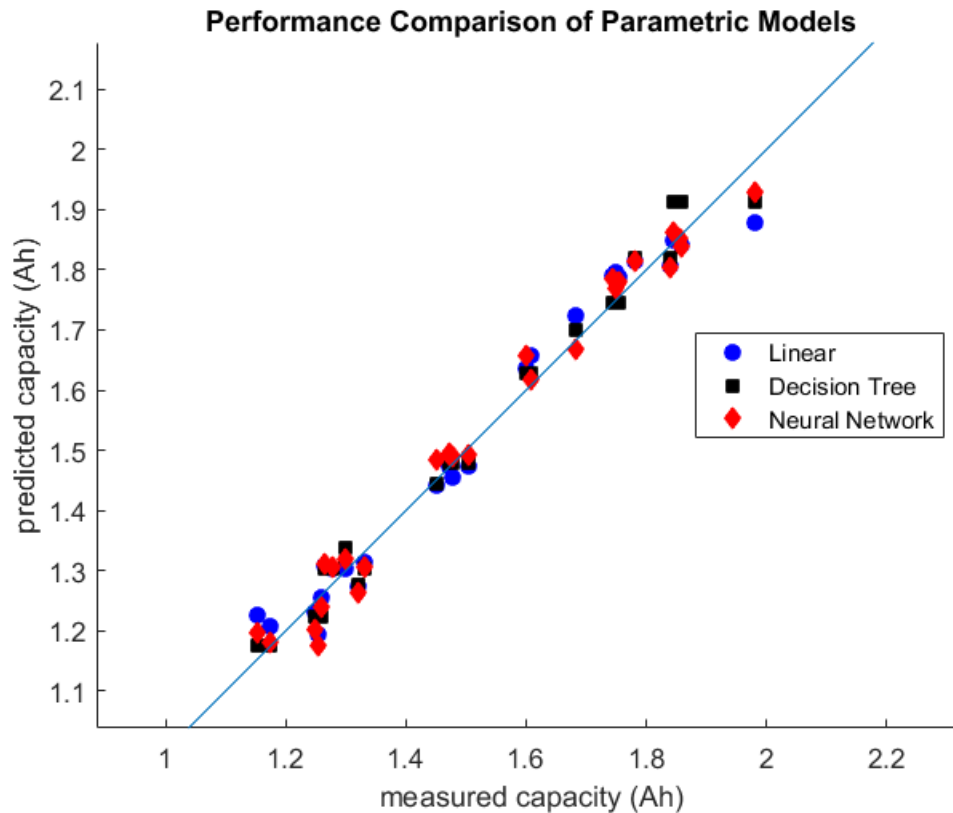
Neural Net Training set MAPE: 2.01%

```



## X - Y plot for comparison between fit methods and measurement

```
f7 = figure(7); clf(f7)
scatter(Ytest,pred_lm, 'ob', 'MarkerFaceColor', 'b'); hold on;
scatter(Ytest,pred_dt, 'sk', 'MarkerFaceColor', 'k');
scatter(Ytest,Y_nn, 'dr', 'MarkerFaceColor', 'r');
line([0.9*min(Ytest) 1.1*max(Ytest)],[0.9*min(Ytest) 1.1*max(Ytest)])
xlabel('measured capacity (Ah)')
ylabel('predicted capacity (Ah)')
legend({'Linear' 'Decision Tree' 'Neural Network'},'location','best')
xlim([0.9*min(Ytest) 1.1*max(Ytest)])
ylim([0.9*min(Ytest) 1.1*max(Ytest)])
axis equal
title('Performance Comparison of Parametric Models')
```

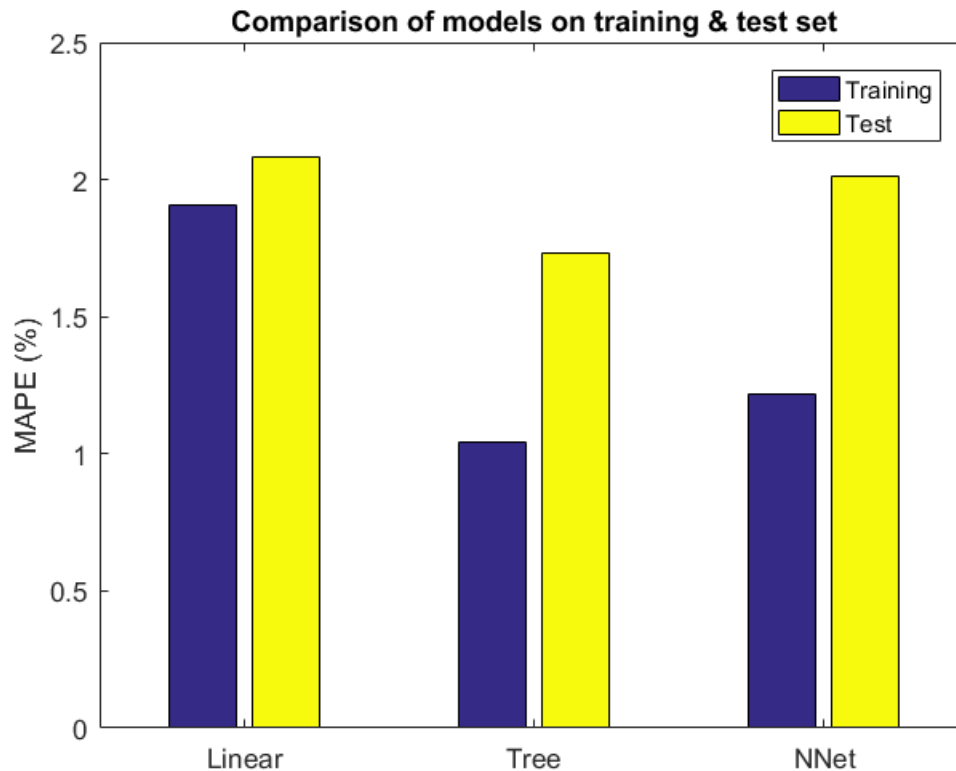


## Regression Performance Comparison

```
f11 = figure(11); clf(f11)
predTest = [pred_lm, pred_dt, Y_nn];
predTrain = [predict mdl_lm, Xtrain) predict mdl_dt, Xtrain)
            mdl_net(Xtrain')'];

perfTest = mape(Ytest, predTest);
perfTrain = mape(Ytrain, predTrain);

bar([perfTrain;perfTest]*100); set(gca, 'XTickLabel',
    {'Linear', 'Tree', 'NNet'});
legend('Training', 'Test')
ylabel('MAPE (%)'); title('Comparison of models on training & test
    set');
```



## Impedance Plot

```
find location of Z measurement idxZ_0 = tt.Type == 'impedance'; build ordinal vector to find cycle #
of each Z experiment idx1 = (1:length(tt.TestData)).*idxZ_0; keep the non-zeros idx2 = nonzeros(idx1);
aux = zeros(length(t.TestData),length(idx2)); for k=1:length(idx2) aux(:,k) = tt.TestData{idx2(k),1}.Rec-
tified_Impedance; aux(:,k) = tt.TestData{idx2(k),1}.Battery_impedance; end f2 = figure(2); clf(f2); hold
on for k=1:40:length(idx2) plot(real(aux(:,k)), -imag(aux(:,k)), 'l', 'LineStyle', 'none'); end axis equal
```

## Looking for impedance signature of capacity fade

```
Find location of Z measurements followed by discharge idxZ_0 = tt.Type == 'impedance'; % build ordinal
vector to find cycle # of each Z experiment idx1 = (1:length(tt.TestData)).*idxZ_0; % keep the non-
zeros idx2 = nonzeros(idx1); % Tests begin with charge and discharge cycles. At cycle 41 impedance
is % recorded for the first time. Plotting capacity (only recorded at % discharge) vs. R0 or R1 requires
some index manipulation. % Let's try to identify which impedance measurements are immediately %
followed by discharge measurements (i.e. by capacity measurements) % idxZ corresponds to impedance
experiments followed by capacity % measurement. idx3 = idx2 .* (tt.Type(idx2 + 1) == 'discharge'); idxZ
= nonzeros(idx3);
```

*Published with MATLAB® R2016b*