Project_2

September 28, 2021

1 Project 2

Max Schrader

1.0.1 Required Imports

[1]:
```
import os
import numpy as np
import pandas as pd
from scipy.optimize import least_squares
from scipy.integrate import trapz, cumtrapz
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from datetime import timedelta
from dataclasses import dataclass, field
from collections import namedtuple
import plotly.io as pio

pio.templates.default = "ggplot2"
pio.renderers.default = "jupyterlab"
```

[2]:
```
df = pd.read_csv(os.path.join(os.getcwd(), "pulse_discharge_test_data.csv"),␣
↪encoding="iso-8859-1")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8497 entries, 0 to 8496
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TIME       8497 non-null   object
 1    "VDC"     8497 non-null   float64
 2    "AMPS"    8497 non-null   float64
 3    "°F"      8497 non-null   float64
 4    "VDC".1   8497 non-null   float64
 5    "AMPS".1  8497 non-null   float64
 6   "°F"       8497 non-null   float64
```

```
dtypes: float64(6), object(1)
memory usage: 464.8+ KB
```

1.1 Question 1: Reading in the File

1.1.1 Fix these column names

[3]:
```
r = iter([0, 1])
re_table = str.maketrans(dict.fromkeys(' "'))
df = df.rename({col: col.translate(re_table) if "°F" not in col else col.
 ↪translate(re_table) + str(next(r)) for col in df.columns}, axis=1)
```

1.1.2 Fix the Time Format

[4]:
```
start_time = pd.to_datetime(df['TIME'].iloc[0], )
```

[5]:
```
df['t_delta'] = 10
df['t_delta'] = df['t_delta'].cumsum().shift(1).fillna(0)
df['dt'] = df['t_delta'].apply(lambda x: start_time + timedelta(seconds=x))
# df['time'] = df['dt'].dt.hour * 3600 + df['dt'].dt.minute * 60 + df['dt'].dt.
 ↪second
```

[6]:
```
#df.set_index('dt', inplace=True, drop=True)
df = df.sort_values('dt', )
df = df.reset_index(drop=True)
df.head()
```

[6]:

	TIME	VDC	AMPS	°F0	VDC.1	AMPS.1	°F1	\
0	7:49:26 AM	26.3467	0.012093	83.8055	26.2096	-0.012426	79.4296	
1	7:49:36 AM	26.3484	0.013611	83.8133	26.2070	-0.011908	79.4368	
2	7:49:46 AM	26.3477	0.025167	83.7744	26.2091	0.000537	79.3995	
3	7:49:56 AM	26.3483	0.030908	83.8188	26.2078	0.011945	79.4607	
4	7:50:06 AM	26.3471	0.028649	83.7770	26.2060	0.001907	79.4264	

	t_delta	dt
0	0.0	2021-09-28 07:49:26
1	10.0	2021-09-28 07:49:36
2	20.0	2021-09-28 07:49:46
3	30.0	2021-09-28 07:49:56
4	40.0	2021-09-28 07:50:06

1.1.3 Create an Object with Battery One and Two

[7]:
```
b1 = df[["VDC", "AMPS", "t_delta", "dt"]].copy()
b2 = df[["VDC.1", "AMPS.1", "t_delta", "dt"]].copy().rename({"VDC.1": "VDC",␣
 ↪"AMPS.1": "AMPS"}, axis=1)
```

```
del df
```

[8]:
```
@dataclass
class Battery:
    df: pd.DataFrame
    rest_groups: object = None
    discharge_groups: object = None
    ls_results: list = field(default_factory=lambda: {})
```

[9]:
```
Batteries = namedtuple("Batteries", 'b1 b2')
batteries = Batteries(Battery(b1), Battery(b2))
```

1.2 Question 5: Calculating the SOC

[10]:
```
for bat in batteries:
    Q_total = trapz(bat.df['AMPS'].values, x=bat.df['t_delta'].values)
    bat.df['SOC'] = np.concatenate(([1], 1 - cumtrapz(bat.df['AMPS'].values,␣
 ↪x=bat.df['t_delta'].values) / Q_total))
```

1.3 Question 2: Grouping the Data into Rest Periods and Discharge Periods

1.3.1 Flag the Events

[11]:
```
def rising_falling(mode="falling"):
    COUNT = 0
    MODE = 0 if mode in "falling" else 1
    def _fn(val):
        nonlocal COUNT
        nonlocal MODE
        if (MODE and val['AMPS'] < 10 and val['AMPS_next'] > 10) or \
           (not MODE and val['AMPS'] > 10 and val['AMPS_next'] < 10):
            COUNT += 1
            return COUNT
        elif (MODE and val['AMPS'] > 10) or (not MODE and val['AMPS'] < 10):
            return COUNT
        else:
            return 0
    return _fn
```

[12]:
```
for bat in batteries:
    bat.df['AMPS_next'] = bat.df['AMPS'].shift(-1)
    bat.df['rest_period'] = bat.df.apply(rising_falling('falling'), axis=1)
    bat.df['discharge_period'] = bat.df.apply(rising_falling('rising'), axis=1)
    bat.rest_groups =  bat.df.groupby('rest_period')
    bat.discharge_groups =  bat.df.groupby('discharge_period')
    bat.df.drop('AMPS_next', inplace=True, axis=1)
```

1.3.2 Plotting the Groups

[13]:
```
fig = make_subplots(specs=[[{"secondary_y": True}]])

fig.add_trace(go.Scatter(
    x=batteries.b1.df['dt'],
    y=batteries.b1.df['AMPS'],
    name="AMPS"
))

fig.add_trace(go.Scatter(
    x=batteries.b1.df['dt'],
    y=batteries.b1.df['VDC'],
    name="VDC"
),secondary_y=True
              )

fig.add_trace(go.Scatter(
    x=batteries.b1.df['dt'],
    y=batteries.b1.df['rest_period'],
    name="Rest Period #"
),secondary_y=True
              )


fig.add_trace(go.Scatter(
    x=batteries.b1.df['dt'],
    y=batteries.b1.df['discharge_period'],
    name="Discharge Period #"
),
               secondary_y=True
              )

fig.update_layout(yaxis_title="Amps", yaxis2_title="Volts", title="Battery 1")

# fig.show()
```

[14]:
```
fig = make_subplots(specs=[[{"secondary_y": True}]])

fig.add_trace(go.Scatter(
    x=batteries.b2.df['dt'],
    y=batteries.b2.df['AMPS'],
    name="AMPS"
))

fig.add_trace(go.Scatter(
    x=batteries.b2.df['dt'],
```

```
    y=batteries.b2.df['VDC'],
    name="VDC"
),secondary_y=True
             )

fig.add_trace(go.Scatter(
    x=batteries.b2.df['dt'],
    y=batteries.b2.df['rest_period'],
    name="Rest Period #"
),secondary_y=True
             )


fig.add_trace(go.Scatter(
    x=batteries.b2.df['dt'],
    y=batteries.b2.df['discharge_period'],
    name="Discharge Period #"
),
              secondary_y=True
             )

fig.update_layout(yaxis_title="Amps", yaxis2_title="Volts", title="Battery 2")

# fig.show()
```

1.4 Question 3: Compute four sets of optimal battery parameters for the exponential decaying function

[15]:
```
@dataclass
class BatteryEstimator:
    i_vect: float
    t_vect: pd.Series
    v_vect: pd.Series

    def decay_func(self, x):
        return x[0] + self.i_vect[0] * x[1] + self.i_vect[0] * sum(self.
 ↪_exponent_helper(x[2:]))

    def _exponent_helper(self, x_slice):
        return (r * np.exp(-1 * self.t_vect / tau) for r, tau in␣
 ↪list(zip(*[iter(x_slice)] * 2)))

    def optimizer(self, ):
        def fn(c):
            val = self.v_vect - self.decay_func(c)
            return val
```

```
        return fn

    def compute_estimate(self, c):
        return self.decay_func(c)

    def x_0(self, n):
        return np.array([self.v_vect[-1], abs((self.v_vect[1] - self.v_vect[0])/
 ↪ self.i_vect[0])] + [abs((self.v_vect[-1] - self.v_vect[1])/ -1 * self.
 ↪i_vect[0]), self.t_vect[-1] / 3] * n )
```

1.4.1 Battery 1

[16]:
```
battery = batteries.b1
```

[17]:
```
# battery.estimator = BatteryEstimator()
for group, rest_group in battery.rest_groups:
    battery.ls_results[group] = {}
    if group >= 1:
#         print(group)
        estimator = BatteryEstimator(
            i_vect=rest_group['AMPS'].values * -1,
            t_vect=rest_group['t_delta'].diff().fillna(0).cumsum().values,
            v_vect=rest_group['VDC'].values
        )
#         print([estimator.x_0(exp_term + 1) for exp_term in range(4)])
        battery.ls_results[group] = {exp_term + 1: least_squares(estimator.
 ↪optimizer(), x0=estimator.x_0(exp_term + 1), bounds=(0, np.inf),)  #␣
 ↪loss='soft_l1',)  # f_scale=0.01,)
                                     for exp_term in range(4)}
```

Residual Analysis

[18]:
```
index = []
records = []
for group, rest_group in battery.rest_groups:
    if 0 < group < 13:
        inner_rec = {}
        index.append(f"Group {group}")
        for exp_terms in range(4):
            inner_rec[f"{exp_terms + 1} Exp Terms"] = (sum(battery.
 ↪ls_results[group][exp_terms + 1].fun ** 2) / \
                                                       (rest_group['t_delta'].
 ↪iloc[-1] - rest_group['t_delta'].iloc[0]))**(1/2) / rest_group['VDC'].mean()
        records.append(inner_rec)

print("Residual NRMSD")
residual_df = pd.DataFrame(records, index=index)
```

```
residual_df['Min Res'] = residual_df.idxmin(axis=1)
residual_df.head(16)
```

Residual NRMSD

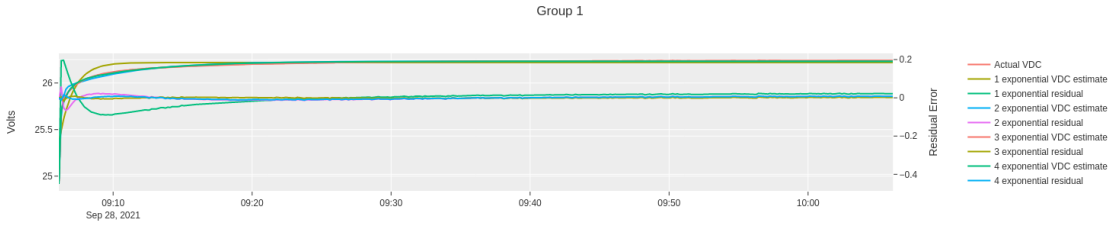
[18]:

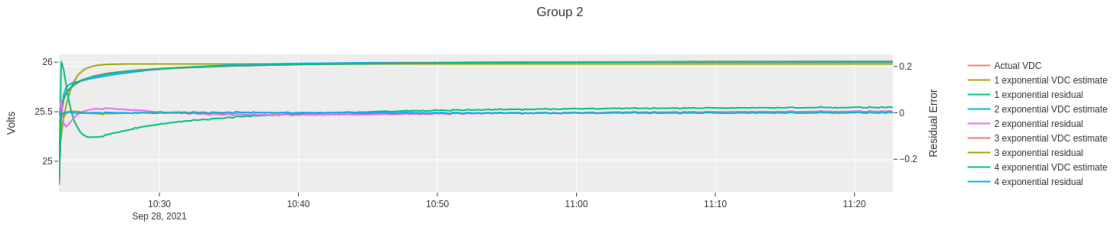
	1 Exp Terms	2 Exp Terms	3 Exp Terms	4 Exp Terms	Min Res
Group 1	0.000506	0.000126	0.000028	0.000074	3 Exp Terms
Group 2	0.000481	0.000111	0.000027	0.000016	4 Exp Terms
Group 3	0.000505	0.000283	0.000281	0.000282	3 Exp Terms
Group 4	0.000511	0.000389	0.000029	0.000059	3 Exp Terms
Group 5	0.000563	0.000101	0.000037	0.000031	4 Exp Terms
Group 6	0.000632	0.000112	0.000038	0.000031	4 Exp Terms
Group 7	0.000723	0.000145	0.000045	0.000029	4 Exp Terms
Group 8	0.000715	0.000141	0.000043	0.000065	3 Exp Terms
Group 9	0.000716	0.000151	0.000043	0.000028	4 Exp Terms
Group 10	0.000939	0.000332	0.000321	0.000321	3 Exp Terms
Group 11	0.001022	0.000374	0.000354	0.000354	3 Exp Terms
Group 12	0.001527	0.000351	0.000064	0.000030	4 Exp Terms

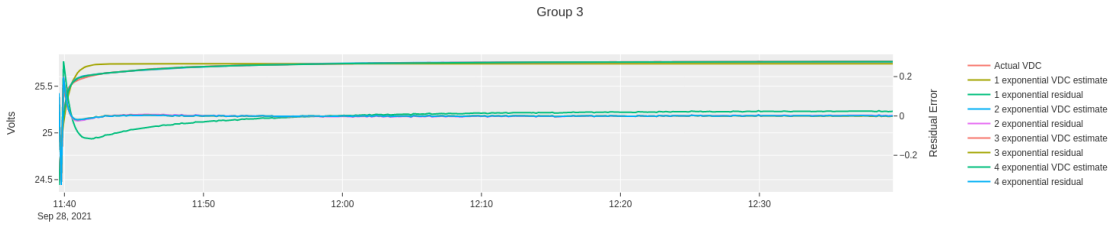
Plotting the Battery 1 Results

[19]:
```
for group, rest_group in battery.rest_groups:

    estimator = BatteryEstimator(i_vect=rest_group['AMPS'].values * -1,
                                 t_vect=rest_group['t_delta'].diff().fillna(0).
 ↪cumsum().values,
                                 v_vect=rest_group['VDC'].values
                                )

    if 0 < group < 13:

        fig = make_subplots(specs=[[{"secondary_y": True}]])

        fig.add_trace(go.Scatter(
            x=rest_group['dt'],
            y=rest_group['VDC'],
            name="Actual VDC"
        ))

        for exp_terms in range(4):

            fig.add_trace(go.Scatter(
                x=rest_group['dt'],
                y=estimator.compute_estimate(battery.
 ↪ls_results[group][exp_terms + 1].x),
                name=f"{exp_terms + 1} exponential VDC estimate"
                )
```

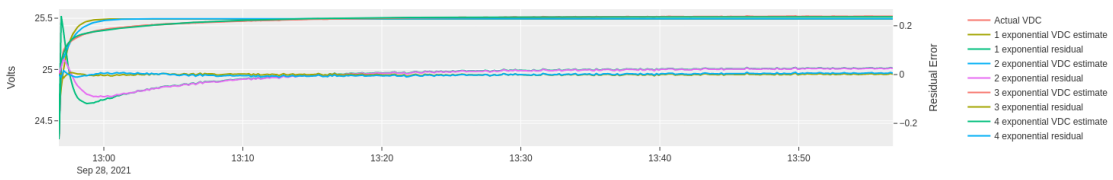
```
            )


            fig.add_trace(go.Scatter(
                x=rest_group['dt'],
                y=battery.ls_results[group][exp_terms + 1].fun,
                name=f"{exp_terms + 1} exponential residual"
            ),secondary_y=True
                          )

        fig.update_layout(title=f"Group {group}", yaxis_title="Volts",␣
↪yaxis2_title="Residual Error")

        fig.show()
```



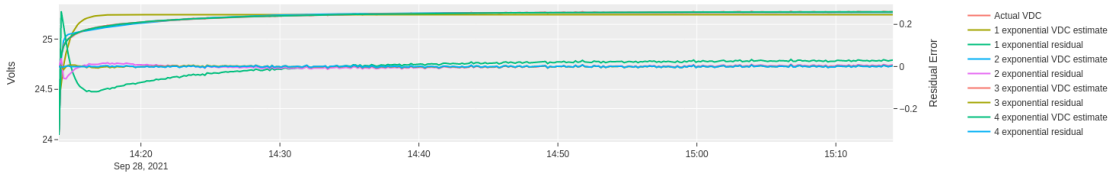




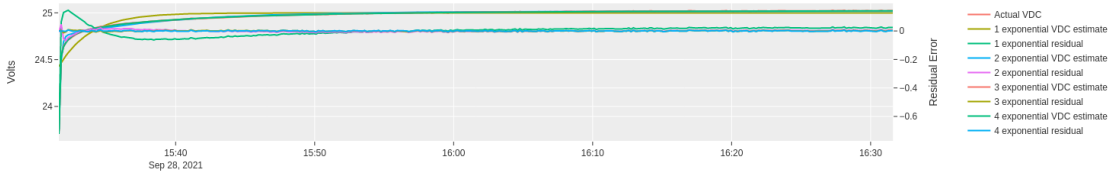
Group 4



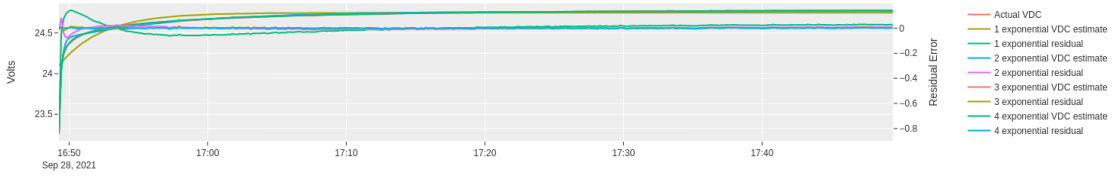
Group 5



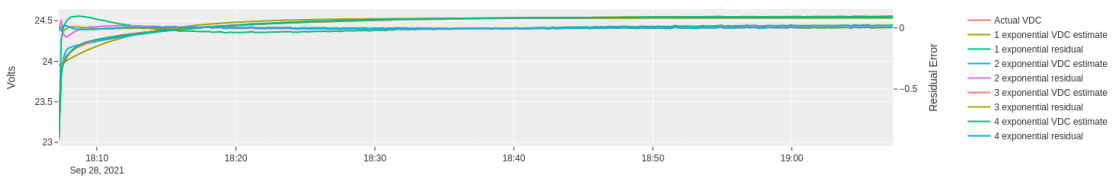
Group 6



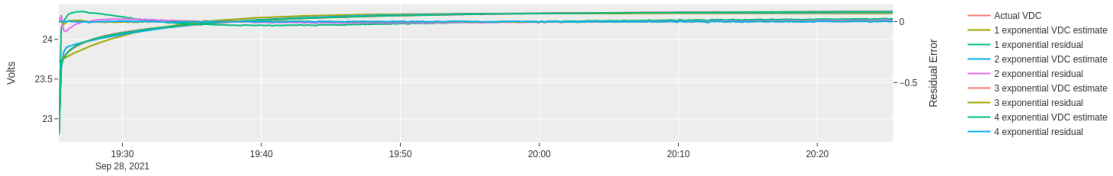
Group 7



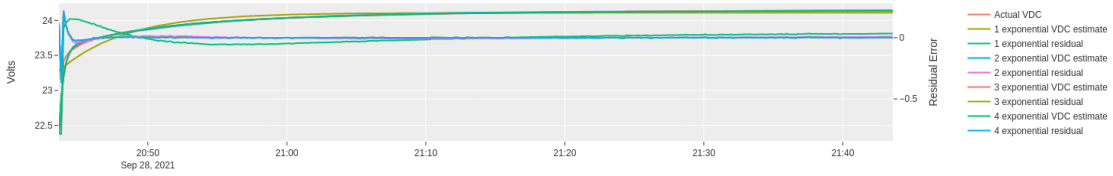
Group 8



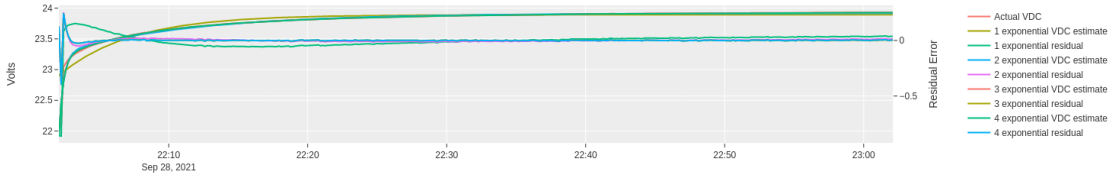
Group 9

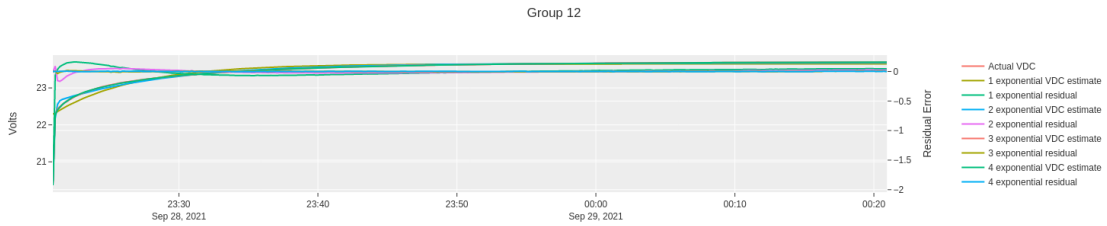


Group 10



Group 11





1.4.2 Battery 2

[20]:
```
battery = batteries.b2
```

[21]:
```
# battery.estimator = BatteryEstimator()
for group, rest_group in battery.rest_groups:
    battery.ls_results[group] = {}
    if 0 < group < 14:
        estimator = BatteryEstimator(i_vect=rest_group['AMPS'].values * -1,
                                     t_vect=rest_group['t_delta'].diff().
 ↪fillna(0).cumsum().values,
                                     v_vect=rest_group['VDC'].values
                                    )
        battery.ls_results[group] = {exp_term + 1: least_squares(estimator.
 ↪optimizer(), x0=estimator.x_0(exp_term + 1), bounds=(0, np.inf), )␣
 ↪#loss='soft_l1')  #, f_scale=0.01,)
                                     for exp_term in range(4)}
```

Residual Analysis

[22]:
```
index = []
records = []
for group, rest_group in battery.rest_groups:
    if 0 < group < 14:
        inner_rec = {}
        index.append(f"Group {group}")
        for exp_terms in range(4):
            inner_rec[f"{exp_terms + 1} Exp Terms"] = (sum(battery.
 ↪ls_results[group][exp_terms + 1].fun ** 2) / \
                                                       (rest_group['t_delta'].
 ↪iloc[-1] - rest_group['t_delta'].iloc[0]))**(1/2) / rest_group['VDC'].mean()
        records.append(inner_rec)

print("Residual Mean")
residual_df = pd.DataFrame(records, index=index)
residual_df['Min Res'] = residual_df.idxmin(axis=1)
```

```
residual_df.head(16)
```

Residual Mean

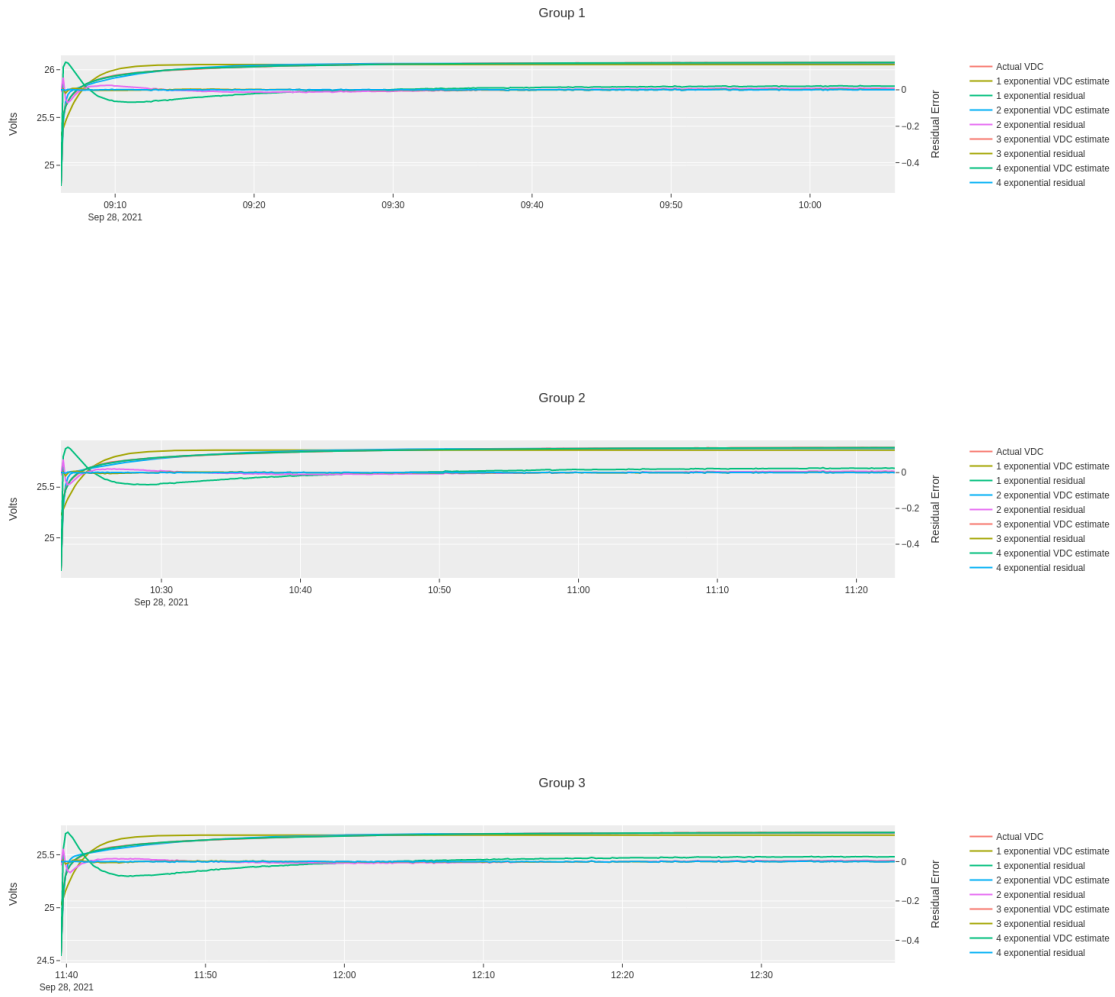
[22]:

	1 Exp Terms	2 Exp Terms	3 Exp Terms	4 Exp Terms	Min Res
Group 1	0.000505	0.000137	0.000029	0.000015	4 Exp Terms
Group 2	0.000530	0.000129	0.000027	0.000014	4 Exp Terms
Group 3	0.000517	0.000106	0.000026	0.000015	4 Exp Terms
Group 4	0.000530	0.000106	0.000028	0.000022	4 Exp Terms
Group 5	0.000537	0.000100	0.000037	0.000046	3 Exp Terms
Group 6	0.000548	0.000150	0.000034	0.000042	3 Exp Terms
Group 7	0.000561	0.000094	0.000034	0.000048	3 Exp Terms
Group 8	0.000589	0.000095	0.000030	0.000044	3 Exp Terms
Group 9	0.000651	0.000177	0.000032	0.000040	3 Exp Terms
Group 10	0.000728	0.000137	0.000035	0.000023	4 Exp Terms
Group 11	0.000760	0.000163	0.000042	0.000028	4 Exp Terms
Group 12	0.000790	0.000191	0.000044	0.000026	4 Exp Terms
Group 13	0.000839	0.000236	0.000053	0.000033	4 Exp Terms

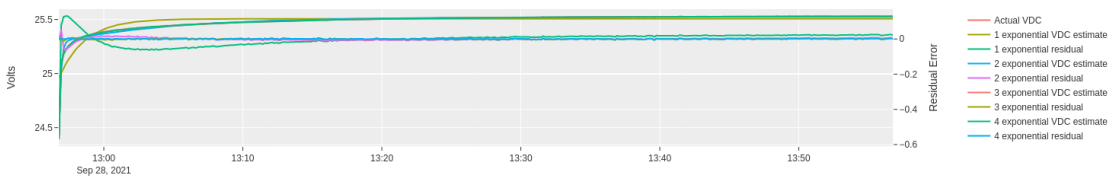
Plotting the Battery 2 Results

[23]:
```
for group, rest_group in battery.rest_groups:

    estimator = BatteryEstimator(i_vect=rest_group['AMPS'].values * -1,
                                 t_vect=rest_group['t_delta'].diff().fillna(0).
↪cumsum().values,
                                 v_vect=rest_group['VDC'].values
                                )
    if 0 < group < 14:

        fig = make_subplots(specs=[[{"secondary_y": True}]])

        fig.add_trace(go.Scatter(
            x=rest_group['dt'],
            y=rest_group['VDC'],
            name="Actual VDC"
        ))

        for exp_terms in range(4):

            fig.add_trace(go.Scatter(
                x=rest_group['dt'],
                y=estimator.compute_estimate(battery.
↪ls_results[group][exp_terms + 1].x),
                name=f"{exp_terms + 1} exponential VDC estimate"
                )
            )
```

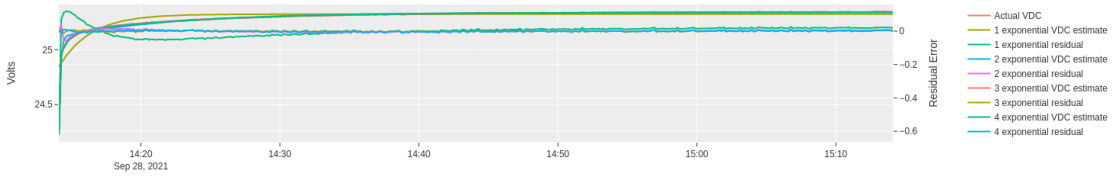
```
        fig.add_trace(go.Scatter(
            x=rest_group['dt'],
            y=battery.ls_results[group][exp_terms + 1].fun,
            name=f"{exp_terms + 1} exponential residual"
        ),secondary_y=True
                      )

    fig.update_layout(title=f"Group {group}", yaxis_title="Volts",␣
↪yaxis2_title="Residual Error")

    fig.show()
```



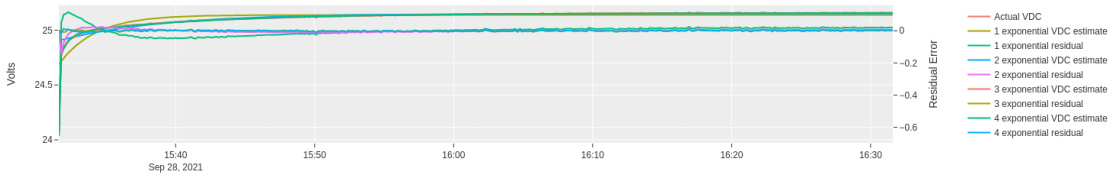
Group 4



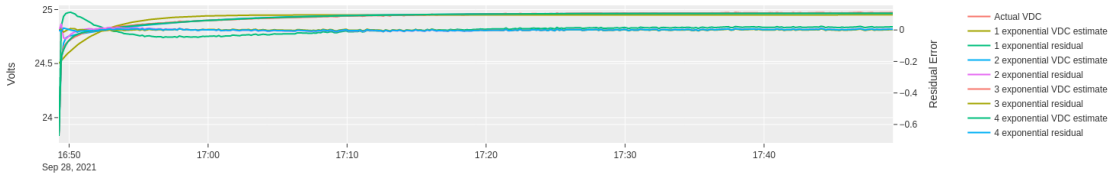
Group 5



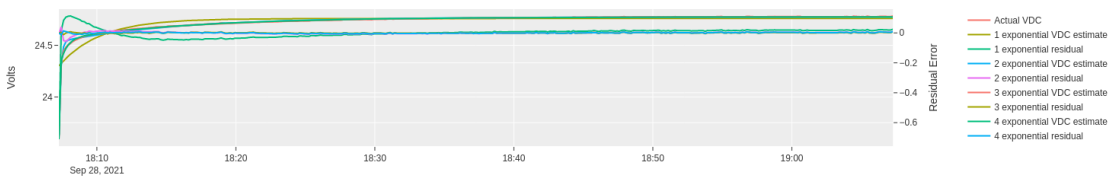
Group 6



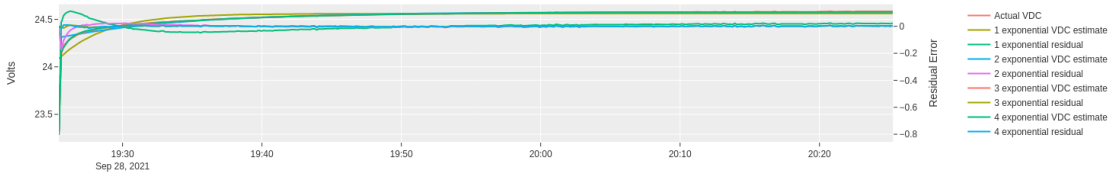
Group 7



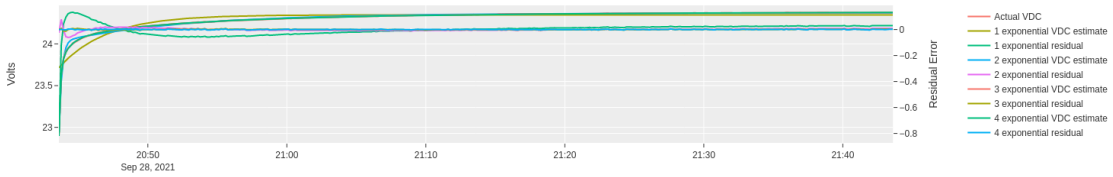
Group 8



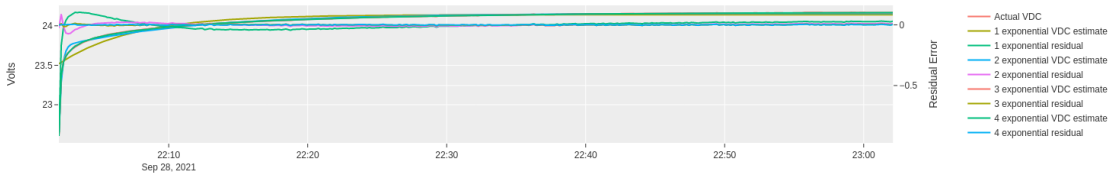
Group 9

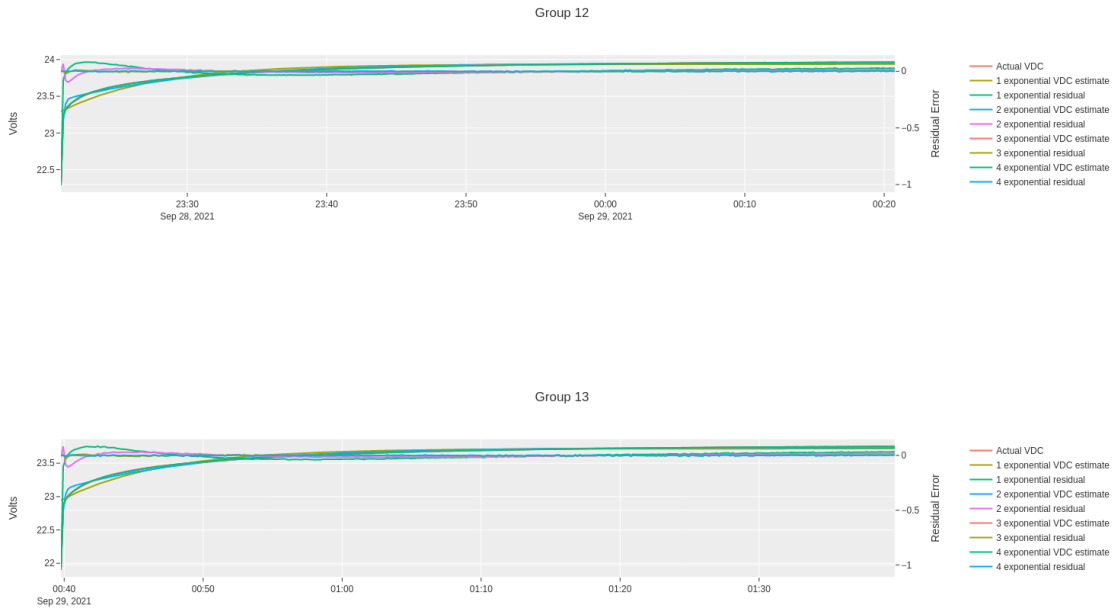


Group 10



Group 11





1.5 Question 4: Plot the estimated parameters as a function of over each pulse discharge test

As the number of number of exponential terms in the fit equation increase, the NRMSE decreases and the esitmated parameters seem to converge to a final value

1.5.1 Battery 1

[24]:
```
bat = batteries.b1
```

OCV

[25]:
```
SOC = []
OCV = []

fig = go.Figure()

for i in range(4):
    ocv = []
    for group, rest_group in bat.rest_groups:
            if 0 < group < 13:
                if i < 1:
                    SOC.append(rest_group['SOC'].iloc[-1])
                ocv.append(bat.ls_results[group][i+1].x[0])
    OCV.append(ocv)
```

```
    fig.add_trace(
        go.Scatter(
            x=SOC,
            y=ocv,
            name=f"OCV Estimate with {i + 1} exp terms"
        )
    )

fig.update_layout(
    xaxis_title="SOC",
    yaxis_title="OCV"
)

# fig.show()
```

R_0

[26]:
```
SOC = []

fig = go.Figure()

for i in range(4):
    R0 = []
    for group, rest_group in bat.rest_groups:
            if 0 < group < 13:
                if i < 1:
                    SOC.append(rest_group['SOC'].iloc[-1])
                R0.append(bat.ls_results[group][i+1].x[1])

    fig.add_trace(
        go.Scatter(
            x=SOC,
            y=R0,
            name=f"{i + 1} exp terms"
        )
    )

fig.update_layout(
    xaxis_title="SOC",
    yaxis_title="R<sub>0</sub>"
)

# fig.show()
```

R_1

[27]:
```
SOC = []

fig = go.Figure()

for i in range(4):
    R1 = []
    for group, rest_group in bat.rest_groups:
            if 0 < group < 13:
                if i < 1:
                    SOC.append(rest_group['SOC'].iloc[-1])
                R1.append(bat.ls_results[group][i+1].x[2])

    fig.add_trace(
        go.Scatter(
            x=SOC,
            y=R1,
            name=f"{i + 1} exp terms"
        )
    )

fig.update_layout(
    xaxis_title="SOC",
    yaxis_title="R<sub>1</sub>"
)

# fig.show()
```

C_1

[28]:
```
SOC = []

fig = go.Figure()

for i in range(4):
    R1 = []
    for group, rest_group in bat.rest_groups:
            if 0 < group < 13:
                if i < 1:
                    SOC.append(rest_group['SOC'].iloc[-1])
                R1.append(bat.ls_results[group][i+1].x[3] / bat.
 ↪ls_results[group][i+1].x[2])

    fig.add_trace(
        go.Scatter(
            x=SOC,
            y=R1,
            name=f"{i + 1} exp terms"
```

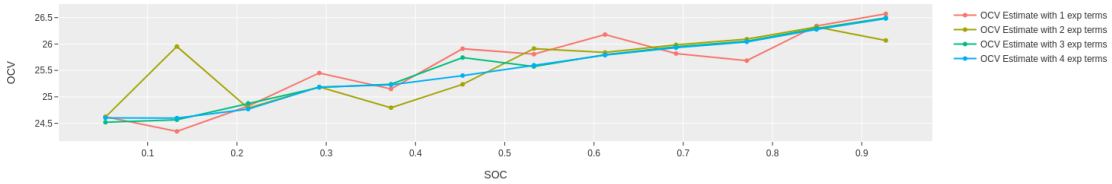
```
        )
    )

fig.update_layout(
    xaxis_title="SOC",
    yaxis_title="C<sub>1</sub>"
)

# fig.show()
```

1.5.2 Battery 2

[29]:
```
bat = batteries.b2
```

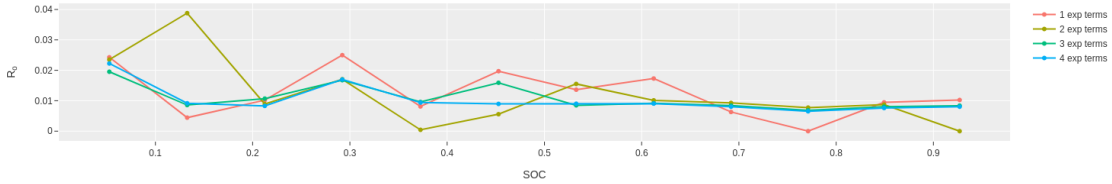
OCV

[30]:
```
SOC = []
OCV = []

fig = go.Figure()

for i in range(4):
    ocv = []
    for group, rest_group in bat.rest_groups:
            if 0 < group < 13:
                if i < 1:
                    SOC.append(rest_group['SOC'].iloc[-1])
                ocv.append(bat.ls_results[group][i+1].x[0])
    OCV.append(ocv)

    fig.add_trace(
        go.Scatter(
            x=SOC,
            y=ocv,
            name=f"OCV Estimate with {i + 1} exp terms"
        )
    )

fig.update_layout(
    xaxis_title="SOC",
    yaxis_title="OCV"
)

# fig.show()
```



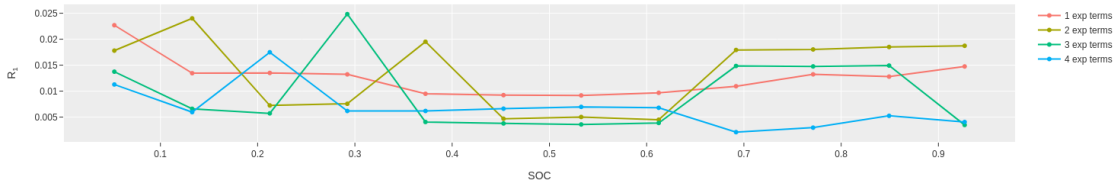
R_0

[31]:
```
SOC = []

fig = go.Figure()

for i in range(4):
    R0 = []
    for group, rest_group in bat.rest_groups:
            if 0 < group < 13:
                if i < 1:
                    SOC.append(rest_group['SOC'].iloc[-1])
                R0.append(bat.ls_results[group][i+1].x[1])

    fig.add_trace(
        go.Scatter(
            x=SOC,
            y=R0,
            name=f"{i + 1} exp terms"
        )
    )

fig.update_layout(
    xaxis_title="SOC",
    yaxis_title="R<sub>0</sub>"
)

# fig.show()
```



R_1

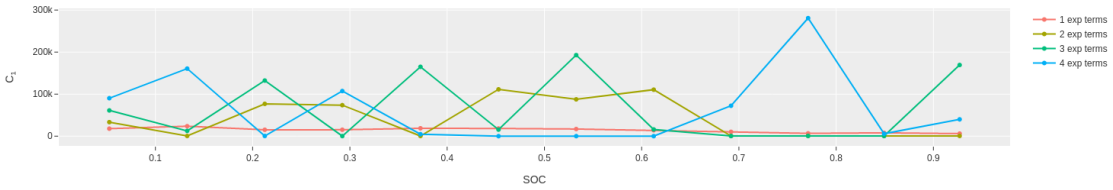
[32]:
```
SOC = []

fig = go.Figure()

for i in range(4):
    R1 = []
    for group, rest_group in bat.rest_groups:
            if 0 < group < 13:
                if i < 1:
                    SOC.append(rest_group['SOC'].iloc[-1])
                R1.append(bat.ls_results[group][i+1].x[2])

    fig.add_trace(
        go.Scatter(
            x=SOC,
            y=R1,
            name=f"{i + 1} exp terms"
        )
    )

fig.update_layout(
    xaxis_title="SOC",
    yaxis_title="R<sub>1</sub>"
)

# fig.show()
```



C_1

[33]:
```
SOC = []

fig = go.Figure()
```

```
for i in range(4):
    R1 = []
    for group, rest_group in bat.rest_groups:
            if 0 < group < 13:
                if i < 1:
                    SOC.append(rest_group['SOC'].iloc[-1])
                R1.append(bat.ls_results[group][i+1].x[3] / bat.
 ↪ls_results[group][i+1].x[2])

    fig.add_trace(
        go.Scatter(
            x=SOC,
            y=R1,
            name=f"{i + 1} exp terms"
        )
    )

fig.update_layout(
    xaxis_title="SOC",
    yaxis_title="C<sub>1</sub>"
)

# fig.show()
```

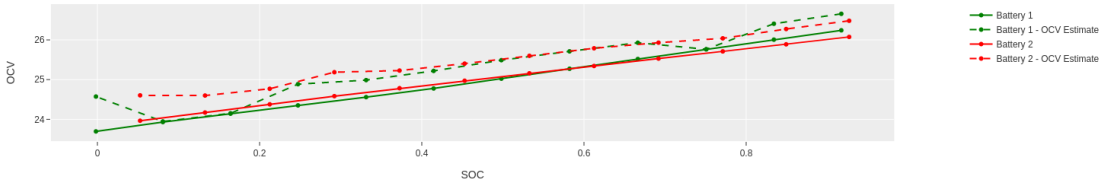


1.6 Question 6: Estimate the OCV-SOC curve for each battery and compare for both batteries

[34]:
```
socs = {}
for i, bat in enumerate(batteries):
    socs[i] = {'vdc': [], 'soc': [], 'vdc_est': []}
    for group, rest_group in bat.rest_groups:
        if 0 < group < 13:
            vdc_est = bat.ls_results[group][4].x[0] # rest_group['VDC'].iloc[-1]
            vdc = rest_group['VDC'].iloc[-1]
            soc = rest_group['SOC'].iloc[-1]
            socs[i]['vdc_est'].append(vdc_est)
```

```
            socs[i]['vdc'].append(vdc)
            socs[i]['soc'].append(soc)
```

[35]:
```
fig = make_subplots(specs=[[{"secondary_y": True}]])


fig.add_trace(go.Scatter(
    x=socs[0]['soc'],
    y=socs[0]['vdc'],
    name="Battery 1",
    line_color="green"
))

fig.add_trace(go.Scatter(
    x=socs[0]['soc'],
    y=socs[0]['vdc_est'],
    name="Battery 1 - OCV Estimate",
    line_dash="dash",
    line_color="green"
))


fig.add_trace(go.Scatter(
    x=socs[1]['soc'],
    y=socs[1]['vdc'],
    name="Battery 2",
    line_color="red"
))


fig.add_trace(go.Scatter(
    x=socs[1]['soc'],
    y=socs[1]['vdc_est'],
    name="Battery 2 - OCV Estimate",
    line_dash="dash",
    line_color="red"
))



fig.update_layout(xaxis_title="SOC", yaxis_title="OCV")

# fig.show()
```


OCV
26
25
24
0
0.2
0.4
0.6
0.8
SOC
Battery 1
Battery 1 - OCV Estimate
Battery 2
Battery 2 - OCV Estimate