Physics 2130: Lab 2B

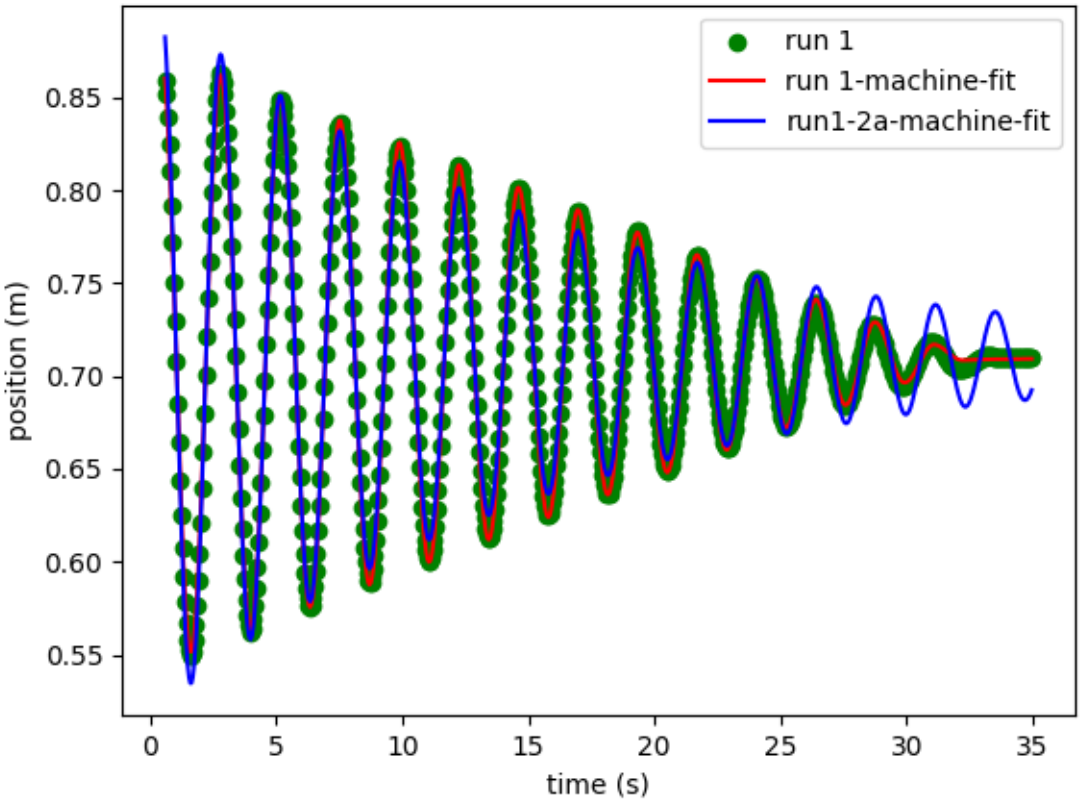
Jeremy Favro

November 15, 2024

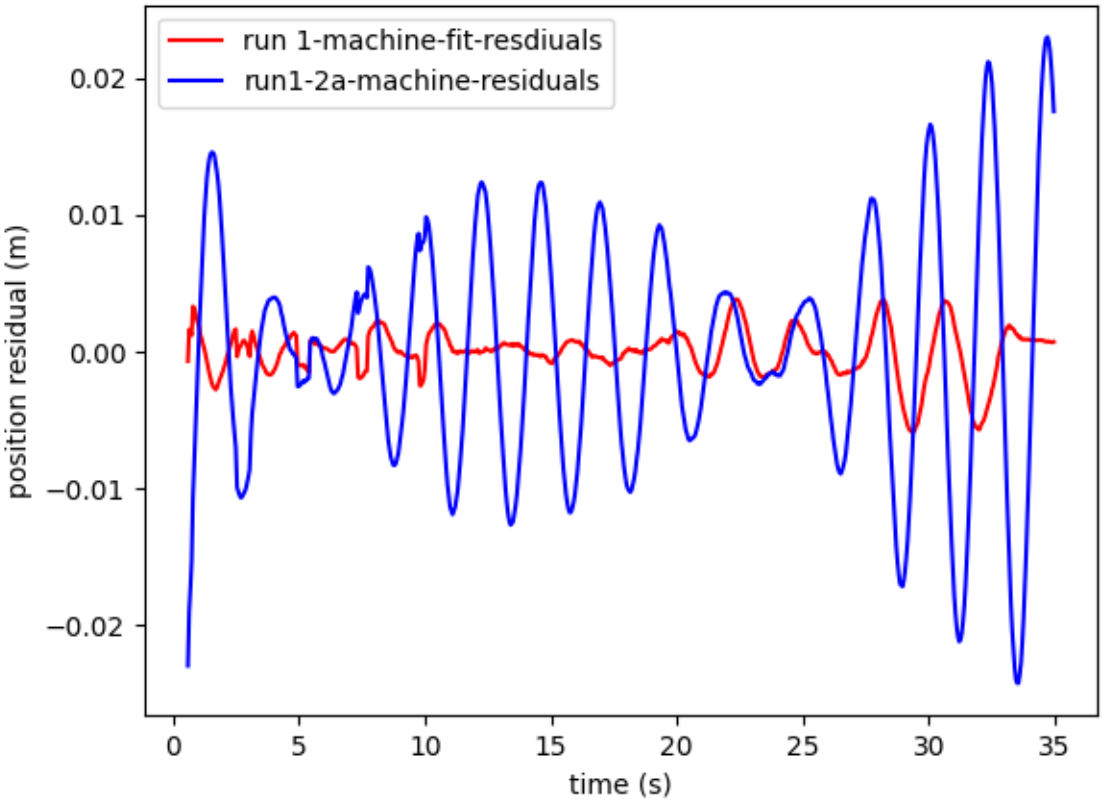
```
import matplotlib.pyplot as pyplot
import numpy as np
from scipy.optimize import curve_fit

def plot_fitted_and_raw(data_path: str,
                        mintime: float,
                        maxtime: float,
                        guessed_values: tuple[float, float, float, float, float],
                        fig: int,
                        label: str = "",
                        display_hand_fit: bool = False):
    data = np.loadtxt(data_path, delimiter=",", skiprows=1)
    data = data[(data[:, 0] > mintime) & (data[:, 0] < maxtime)]
    global t_vals
    t_vals = data[:, 0]
    global positions
    positions = data[:, 1]

    pyplot.figure(fig)
    pyplot.scatter(t_vals, positions, color = "green", label=label)

    x_0 = guessed_values[0]
    x_eq = guessed_values[1]
    v_0 = guessed_values[2]
    w_0 = guessed_values[3]
    B = guessed_values[4]

    def euler_chroma(t_vals: np.ndarray,
                     x_0_f: float,
                     x_eq_f: float,
                     v_0_f: float,
                     w_0_f: float,
                     B_f: float
                     ) -> tuple[np.ndarray, np.ndarray]:
        x_a = np.zeros_like(t_vals)
        v_a = np.zeros_like(t_vals)

        x_a[0] = x_0_f
        v_a[0] = v_0_f

        dt = (t_vals[1]-t_vals[0])
        for t_i in range(len(t_vals) - 1):
            a = -(w_0_f**2)*(x_a[t_i]-x_eq_f)-B_f*np.sign(v_a[t_i])

            v_a[t_i+1] = v_a[t_i] + a*dt
            x_a[t_i+1] = x_a[t_i] + v_a[t_i+1]*dt
```

```
        return x_a

    if (display_hand_fit):
        p = euler_chroma(t_vals, x_0, x_eq, v_0, w_0, B)
        pyplot.plot(t_vals, p, label=f"{label}-hand-fit", color="red")
    popt, covm = curve_fit(euler_chroma, t_vals, positions, p0=guessed_values)
    for i in range(0, 2):
        popt, covm = curve_fit(euler_chroma, t_vals, positions, p0=popt)

    ec_result = euler_chroma(t_vals, *popt)

    pyplot.plot(t_vals, ec_result, label=f"{label}-machine-fit", color="red")
    pyplot.figure(fig+99)
    pyplot.plot(t_vals, positions - ec_result, label=f"{label}-machine-fit-resdiuals", color="red")
    pyplot.legend()
    pyplot.xlabel("time (s)")
    pyplot.ylabel("position residual (m)")

    pyplot.figure(fig)
    pyplot.legend()
    pyplot.xlabel("time (s)")
    pyplot.ylabel("position (m)")

def plot_fit(t, x, amp, beta, omega, delta):
    pos_vals = []
    for t_val in t:
        pos_vals.append(x + amp*(np.e**(-beta*t_val))*np.cos(omega*t_val-delta))
    return pos_vals

x_0 = .87
x_eq = 0.71
v_0 = -0.2
w_0 = 2.63
B = 0.021

guessed_fits = (x_0, x_eq, v_0, w_0, B)
plot_fitted_and_raw("./data/run1.csv", 0.55, 35, guessed_fits, 0, "run 1")
a_result = plot_fit(t_vals, 0.7099221, -0.19411723, 0.06157102, -2.6606827, -4.34149397)
pyplot.plot(t_vals, a_result, label=f"run1-2a-machine-fit", color="blue")
pyplot.legend()
pyplot.figure(99)
pyplot.plot(t_vals, positions - a_result, label=f"run1-2a-machine-residuals", color="blue")
pyplot.legend()

x_0 = .87
x_eq = 0.66
v_0 = -0.2
w_0 = 2.4
B = 0.085

guessed_fits = (x_0, x_eq, v_0, w_0, B)
plot_fitted_and_raw("./data/run2.csv", 0.35, 14, guessed_fits, 1, "run 2")
a_result = plot_fit(t_vals, 0.66118243, 0.28286486, 0.25038451, 2.48782223, 0.9432566)
pyplot.plot(t_vals, a_result, label=f"run2-2a-machine-fit", color="blue")
pyplot.legend()
pyplot.figure(100)
```

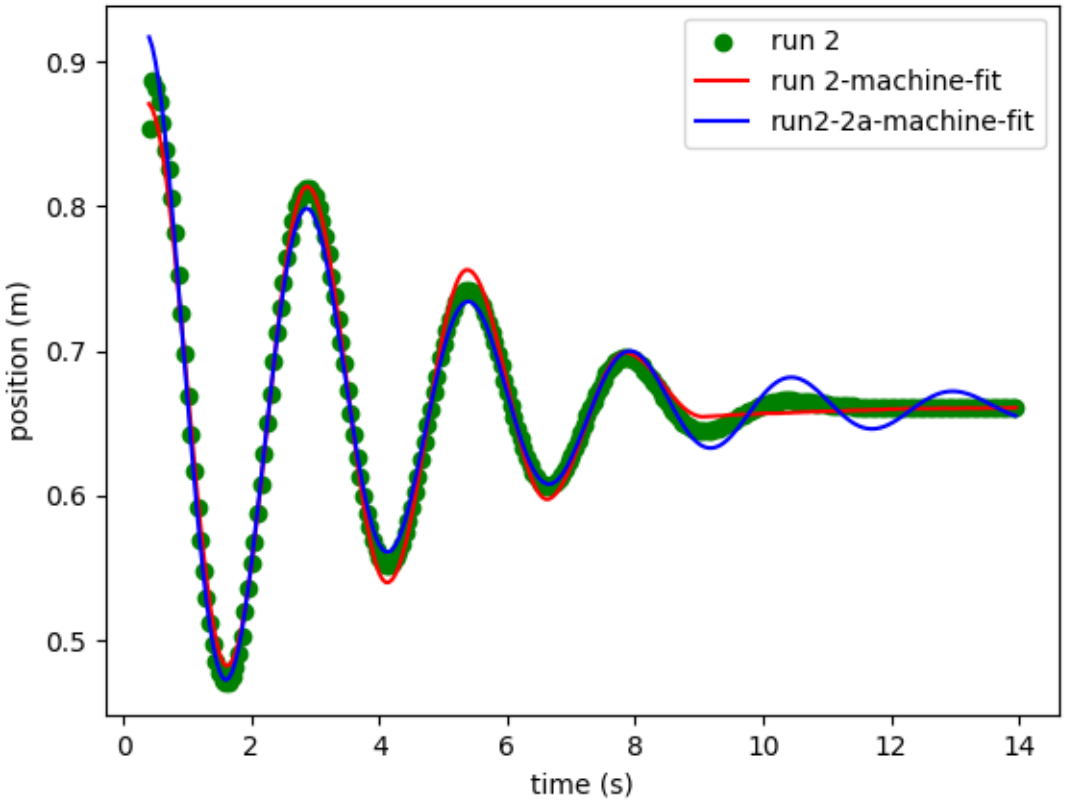
```
pyplot.plot(t_vals, positions - a_result, label=f"run2-2a-machine-residuals", color="blue")
pyplot.legend()
pyplot.show()
```



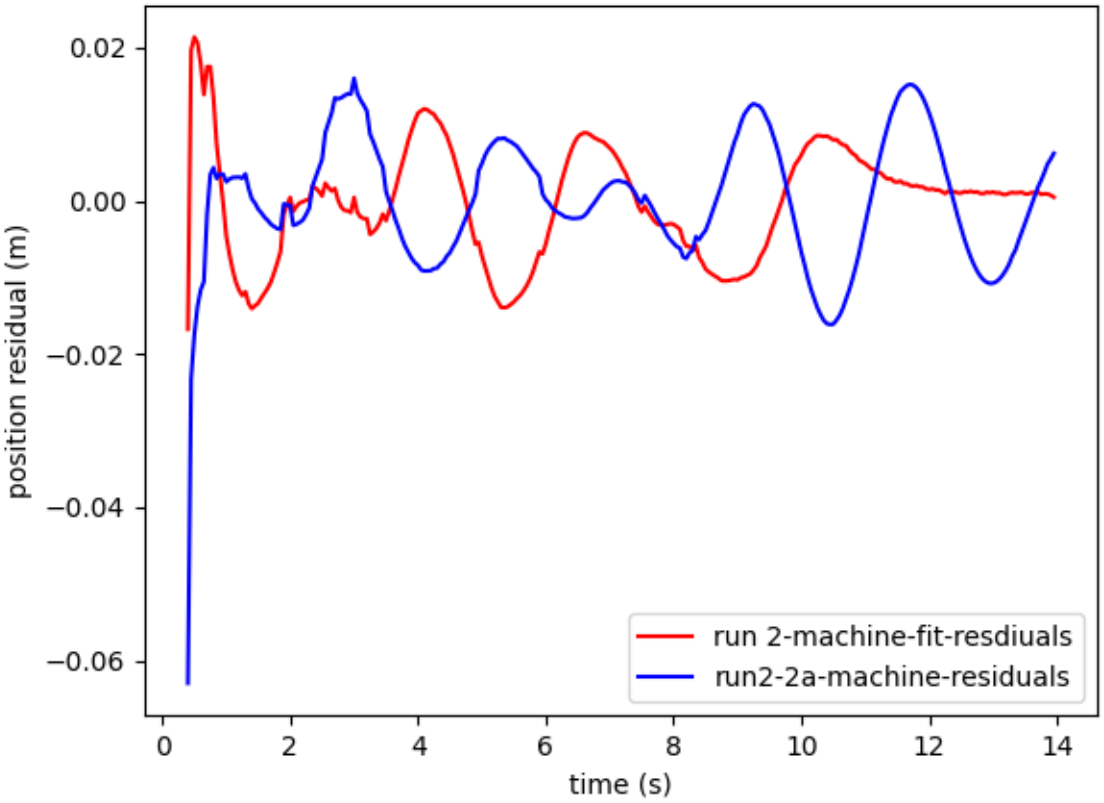
The model incorporating friction appears to (and does per the residuals) fit this data better than last week's model.



As seen on the graph this model is a much better fit compared to last week's for this data. In this case this is true effectively everywhere.



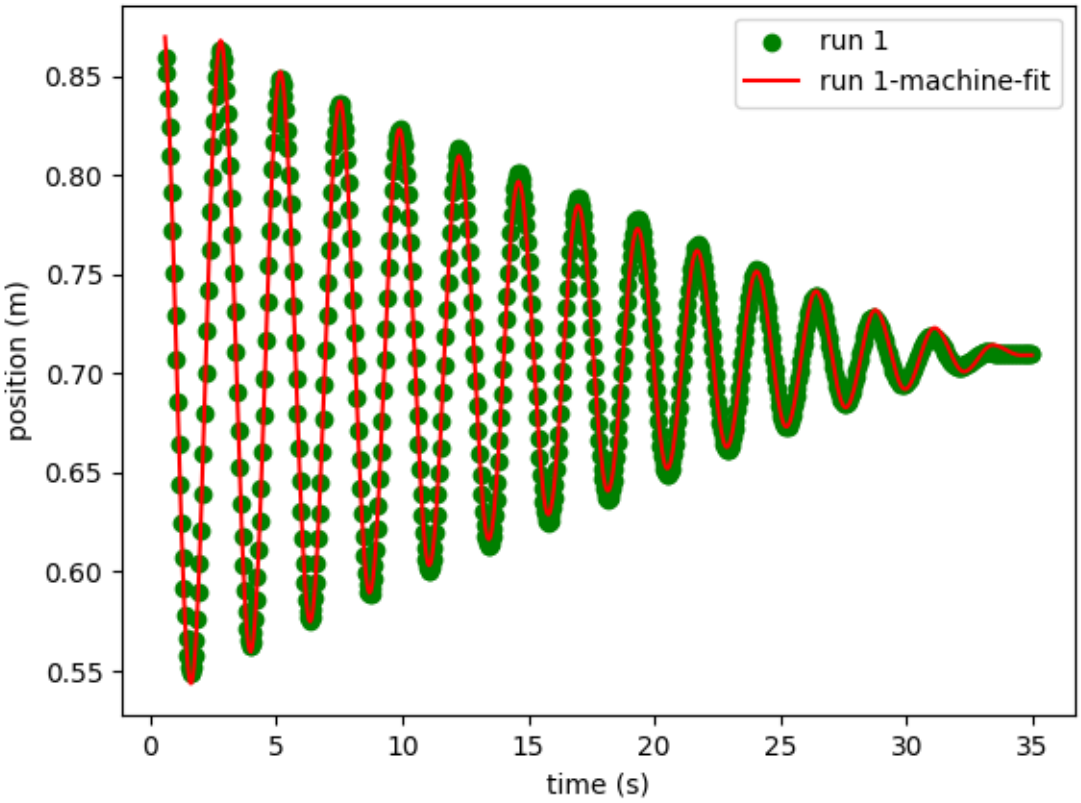
In this case both models seem to fail in different ways. This week's model overestimates the actual value, whereas last week's underestimated it. Near the end ($t \to t_f$) this week's model is a definite better fit as it actually reaches $\approx x_{eq}$.

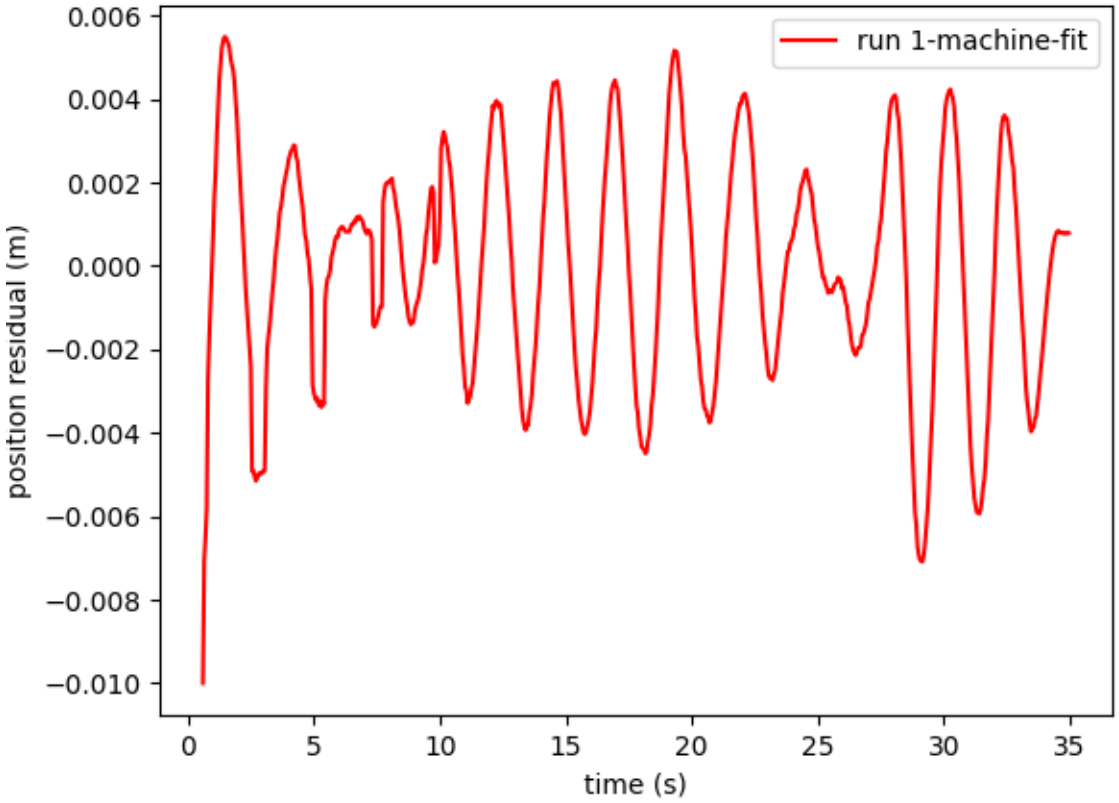


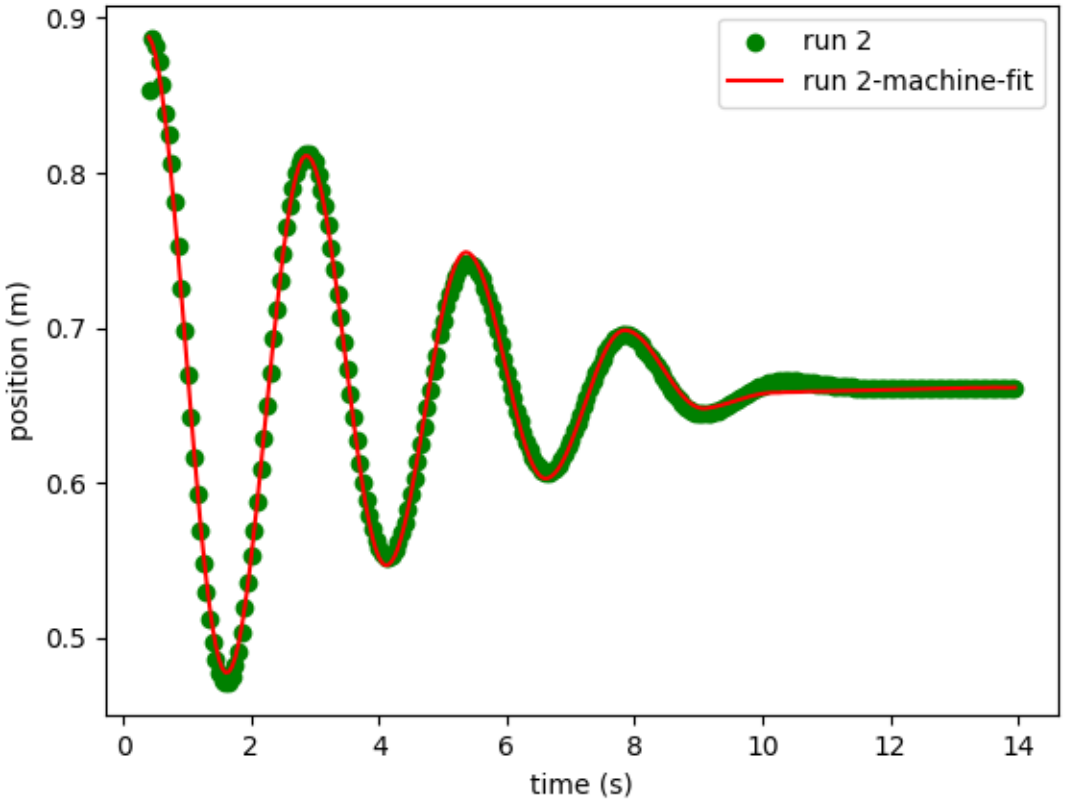
As seen on the graph last week's model underestimates whereas last week's overestimates. Also as seen on the graph this week's model does a much better job at fitting the data near the end where the real data has decayed.

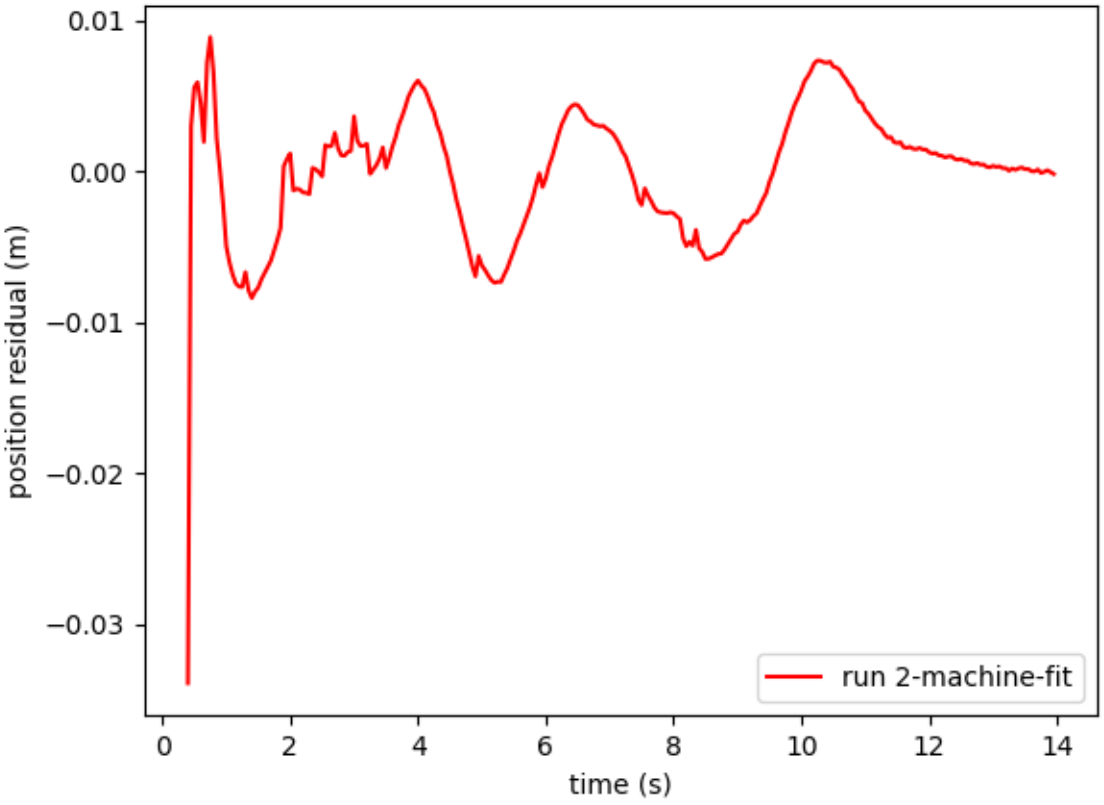
```
import matplotlib.pyplot as pyplot
import numpy as np
from scipy.optimize import curve_fit

def plot_fitted_and_raw(data_path: str,
                        mintime: float,
                        maxtime: float,
                        guessed_values: tuple[float, float, float, float, float],
                        fig: int,
                        label: str = "",
                        display_hand_fit: bool = False):
    data = np.loadtxt(data_path, delimiter=",", skiprows=1)
    data = data[(data[:, 0] > mintime) & (data[:, 0] < maxtime)]
    t_vals = data[:, 0]
    positions = data[:, 1]

    pyplot.figure(fig)
    pyplot.scatter(t_vals, positions, color = "green", label=label)

    x_0 = guessed_values[0]
    x_eq = guessed_values[1]
    v_0 = guessed_values[2]
    w_0 = guessed_values[3]
    B = guessed_values[4]
    A = guessed_values[5]
    beta = guessed_values[4]

    def euler_chroma(t_vals: np.ndarray,
                     x_0_f: float,
                     x_eq_f: float,
                     v_0_f: float,
                     w_0_f: float,
                     B_f: float,
                     A_f: float,
                     beta_f: float
                     ) -> tuple[np.ndarray, np.ndarray]:
        x_a = np.zeros_like(t_vals)
        v_a = np.zeros_like(t_vals)

        x_a[0] = x_0_f
        v_a[0] = v_0_f

        dt = (t_vals[1]-t_vals[0])
        for t_i in range(len(t_vals) - 1):
            a = -(w_0_f**2)*(x_a[t_i]-x_eq_f)-B_f*np.sign(v_a[t_i])-2*beta*v_a[t_i]

            v_a[t_i+1] = v_a[t_i] + a*dt
            x_a[t_i+1] = x_a[t_i] + v_a[t_i+1]*dt

        return x_a


    if (display_hand_fit):
        p = euler_chroma(t_vals, x_0, x_eq, v_0, w_0, B, A, beta)
        pyplot.plot(t_vals, p, label=f"{label}-hand-fit", color="red")
    popt, covm = curve_fit(euler_chroma, t_vals, positions, p0=guessed_values)
    ec_result = euler_chroma(t_vals, *popt)
```

```
    pyplot.figure(fig+10)
    pyplot.plot(t_vals, positions-ec_result, label=f"{label}-machine-fit", color="red")
    pyplot.legend()
    pyplot.xlabel("time (s)")
    pyplot.ylabel("position residual (m)")

    pyplot.figure(fig)
    pyplot.plot(t_vals, ec_result, label=f"{label}-machine-fit", color="red")
    pyplot.legend()
    pyplot.xlabel("time (s)")
    pyplot.ylabel("position (m)")



x_0 = .87
x_eq = 0.71
v_0 = -0.2
w_0 = 2.63
B = 0.021
A = -0.19411723
beta = 0.06157102

guessed_fits = (x_0, x_eq, v_0, w_0, B, A, beta)
plot_fitted_and_raw("./data/run1.csv", 0.55, 35, guessed_fits, 0, "run 1")
pyplot.legend()

x_0 = .87
x_eq = 0.66
v_0 = -0.2
w_0 = 2.4
B = 0.085
A = 0.28286486
beta = 0.25038451

guessed_fits = (x_0, x_eq, v_0, w_0, B, A, beta)
plot_fitted_and_raw("./data/run2.csv", 0.35, 14, guessed_fits, 1, "run 2")
pyplot.show()
```


run 1
run 1-machine-fit
position (m)
time (s)
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
5
10
15
20
25
30
35


run 1-machine-fit
position residual (m)
time (s)
0.006
0.004
0.002
0.000
−0.002
−0.004
−0.006
−0.008
−0.010
0
5
10
15
20
25
30
35


run 2
run 2-machine-fit
position (m)
time (s)
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8
10
12
14



Given the significant decrease in the residuals the first run and the slight improvement in run 2 I would say that the effort to change the model is definitely worth it.