Internal Resistance Calculation for Small Vertical Axis Wind Turbine

Author: Ian Eykamp, **Date:** 6/16/2023

Purpose

All generators have some amount of internal resistance, which allows there to be a variable voltage drop across the terminals, even if the generator always outputs the same emf. By connecting various resistors across the terminals (using no resistance as a baseline), I can figure out what the internal resistance is.

Import libraries

`plecs_helper.py` is a file I created which defines some helper functions for nice plots and getting data off the oscilloscope and out of PLECS.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from UliEngineering.EngineerIO import format_value
from si_prefix import si_format
import plecs_helper as helper
%matplotlib
%matplotlib inline

# Imports and setup
from pint import UnitRegistry
from scipy.signal import find_peaks
from scipy.optimize import fsolve

# pandas display using scientific notation
# pd.set_option('display.float_format', lambda x: f'{x:.3e}')

# use pint
units = UnitRegistry()
units.default_format = "~P.2f"

def to_decibels(arr, dc_gain = 0):
    return np.log10(np.abs(arr)) * 20 + dc_gain
```

```
Using matplotlib backend: TkAgg
```

Import Data

I connected the oscilloscope leads between two terminals of the generator (the third terminal was left dangling). I set the oscilloscope up with 2V/division (y axis) and 5s/division (time axis). That way, I got a trace which lasted more than 30 seconds, enough for it to slow down considerably. I used the data density setting of 12k data points, and this was easily able to capture all the peaks. I spun the turbine by hand by giving it a moderate shove. It started spinning with a frequency around 6Hz, and as it slowed down, I was able to capture its frequency and amplitude on the oscilloscope.

I captured one dataset with no load, and three further datasets at 100Ohm, 6.8Ohm, and (6.8 || (6.8 + 6.8)) Ohm loads between the terminals I was measuring.

In [ ]:
```
# Import data
(df_1, tspan, tstep) = helper.read_rigol_csv("oscilloscope_data/unloaded.csv", ch1
(df_2, tspan, tstep) = helper.read_rigol_csv("oscilloscope_data/100Ohm.csv", ch1 =
(df_3, tspan, tstep) = helper.read_rigol_csv("oscilloscope_data/6_8Ohm.csv", ch1 =
(df_4, tspan, tstep) = helper.read_rigol_csv("oscilloscope_data/4_5Ohm.csv", ch1 =

# Combine all variables into one for convenience
df = df_1.set_index("t").join([df_2.set_index("t"), df_3.set_index("t"), df_4.set_i
# print(df.head(20))
```

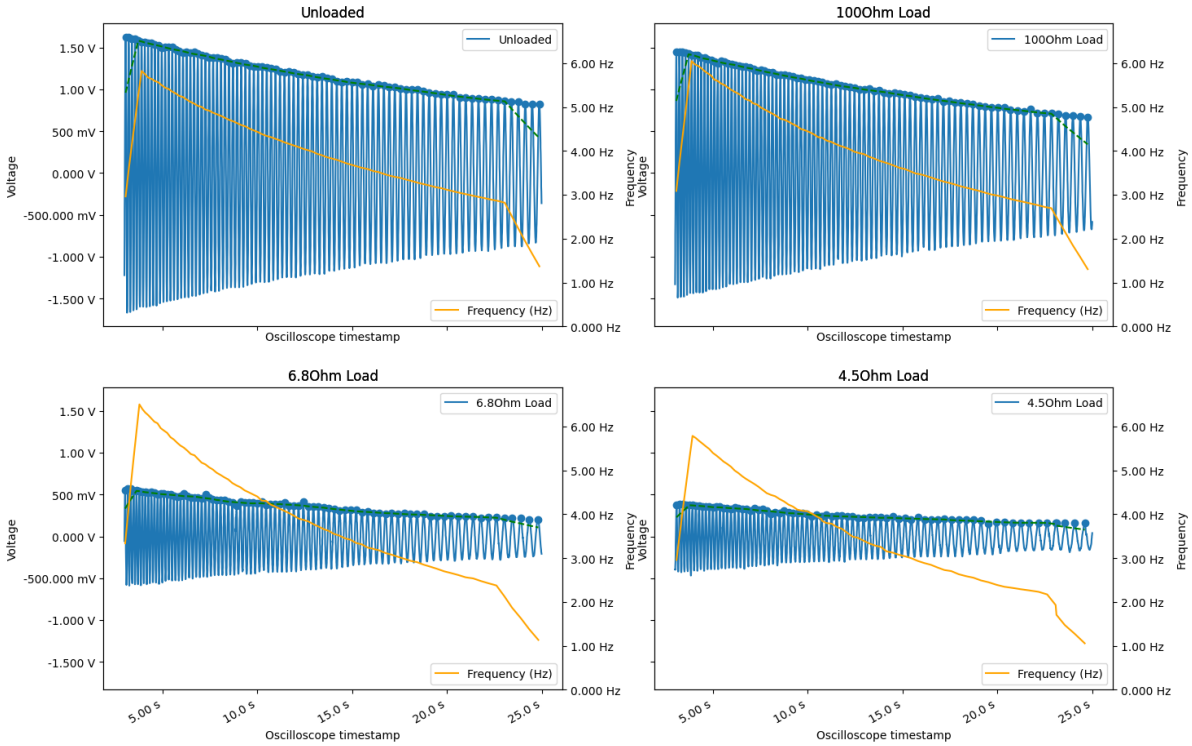
Plot Data

I calculated the frequency by looking at the spacing of the peaks. After smoothing out the frequency and the peak values, I found the location where each trace crossed 4Hz and took the amplitude at that location. Since the amplitude depends on frequency, this is to normalize across datasets, since the turbine is not necsessarily spinning at the same frequency at the same timestamp.

In [ ]:
```
def moving_average(x, w):
    w = int(w / 2) * 2
    convolution = np.convolve(x, np.ones(w), 'full') / w
    return convolution[int(w/2):-int(w/2)+1]

def find_y_crossing_monotonic_decrease(y_vals, y_target):
    left_side_idx = list(np.arange(len(y_vals))[y_vals > y_target])
    return left_side_idx[-1]

def interpolate_outliers(y_vals, outlier_bool):
    outlier_idx = np.nonzero(outlier_bool)[0]
    if len(outlier_idx) == 0:
        return y_vals
    assert(all(np.diff(outlier_idx) > 1))
    assert(all(outlier_idx > 0) and all(outlier_idx < len(y_vals - 1))) # not the f
    for idx in outlier_idx:
        y_vals[idx] = (y_vals[idx - 1] + y_vals[idx + 1]) / 2
    return y_vals

df_zoom = df[(df["t"] > 3) & (df["t"] < 25)]
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows = 2, ncols = 2, sharex = True, s
```

```python
fig.autofmt_xdate()

ax_list = [ax1, ax2, ax3, ax4]
channel_list = ["unloaded", "100Ohm", "6.8Ohm", "4.5Ohm"]
label_list = ["Unloaded", "100Ohm Load", "6.8Ohm Load", "4.5Ohm Load"]


amplitudes = []
for (ax, channel, label) in zip(ax_list, channel_list, label_list):
    helper.axes_labels("Oscilloscope timestamp", "s", "Voltage", "V", title = label

    ax.plot(df_zoom["t"], df_zoom[channel], label = label)

    # find peaks for frequency and envelope amplitude analysis
    peak_idx = find_peaks(df_zoom[channel], distance = 25, width = 25)[0]
    df_peak_trace = df_zoom.iloc[peak_idx]
    smooth_peaks = moving_average(df_peak_trace[channel], 10)

    # find frequency using time difference between peaks
    frequency = np.hstack((0, 1 / (np.diff(df_peak_trace["t"]))))
    smooth_frequency = moving_average(interpolate_outliers(frequency, frequency > 2

    # find the envelope amplitude at a certain frequency for normalized comparison
    target_freq = 4 # Hz
    target_freq_idx = find_y_crossing_monotonic_decrease(y_vals = smooth_frequency,
    amplitude_at_target_freq = smooth_peaks[target_freq_idx]
    amplitudes.append(amplitude_at_target_freq)

    # plot peaks
    ax.scatter(df_peak_trace["t"], df_peak_trace[channel])
    ax.plot(df_peak_trace["t"], smooth_peaks, linestyle = "dashed", color = "green"
    ax.legend(loc = "upper right")

    # plot frequency
    rax = ax.twinx()
    helper.axes_labels("Oscilloscope timestamp", "s", "Frequency", "Hz", title = la
    rax.plot(df_peak_trace["t"], smooth_frequency, color = "orange", label = "Frequ
    rax.set_ylim(0, 6.9)
    rax.legend(loc = "lower right")
```

```
C:\Users\ieykamp\AppData\Local\Temp\ipykernel_25180\1153253338.py:36: PeakProperty
Warning: some peaks have a width of 0
  peak_idx = find_peaks(df_zoom[channel], distance = 25, width = 25)[0]
C:\Users\ieykamp\AppData\Local\Temp\ipykernel_25180\1153253338.py:36: PeakProperty
Warning: some peaks have a width of 0
  peak_idx = find_peaks(df_zoom[channel], distance = 25, width = 25)[0]
C:\Users\ieykamp\AppData\Local\Temp\ipykernel_25180\1153253338.py:36: PeakProperty
Warning: some peaks have a width of 0
  peak_idx = find_peaks(df_zoom[channel], distance = 25, width = 25)[0]
```



The abrupt drop off in frequency near the end is a figment of the smoothing algorithm and can be ignored.

The amplitudes at 4Hz are as follows. I calcualted the internal resistance R_1 using the following equation, based on the load resistor I was using: $R1 = R2 \cdot (\frac{V_{in}}{V_{out}} - 1)$, where V_{in} is the unloaded amplitude and V_{out} is the amplitude given a load resistor of R_2. The internal resistance of the generator is approximately $14.5\Omega \pm 1\Omega$.

In [ ]:
```
def calculate_R1(Vin, Vout, R2):
    R1 = R2 * (Vin / Vout - 1)
    return R1

print("Amplitude with no load at 4Hz =", np.round(amplitudes[0], 3) * units.volt)
print("Amplitude with 100Ω load at 4Hz =", np.round(amplitudes[1], 3) * units.volt)
print("Amplitude with 6.8Ω load at 4Hz =", np.round(amplitudes[2], 3) * units.volt)
print("Amplitude with 4.5Ω load at 4Hz =", np.round(amplitudes[3], 3) * units.volt)
print("")

print("Three different calculations for internal resistance:")
print("R1 =", calculate_R1(amplitudes[0] * units.volt, amplitudes[1] * units.volt,
print("R1 =", calculate_R1(amplitudes[0] * units.volt, amplitudes[2] * units.volt,
print("R1 =", calculate_R1(amplitudes[0] * units.volt, amplitudes[3] * units.volt,
```

Amplitude with no load at 4Hz = 1.17 V
Amplitude with 100Ω load at 4Hz = 1.02 V
Amplitude with 6.8Ω load at 4Hz = 0.38 V
Amplitude with 4.5Ω load at 4Hz = 0.26 V

Three different calculations for internal resistance:
R1 = 13.97 Ω
R1 = 14.17 Ω
R1 = 15.93 Ω