

# ECE 110 Lab: Measurement Data Demo for Lab 6

## 1 Starting BenchVue

Search for BenchVue from the Start menu to access the program. When initially opened you will see an interface similar to the one shown in Figure 1. At this point make certain that your benchtop DC power supply is powered on. Benchvue may not initially recognize our benchtop devices. Click the icon in the lower right corner of the program (which is outlined in Figure 1). Refresh the listed devices, and then choose the option to show the device. This should add your equipment to the bottom bar of the interface, as shown in Figure 2.

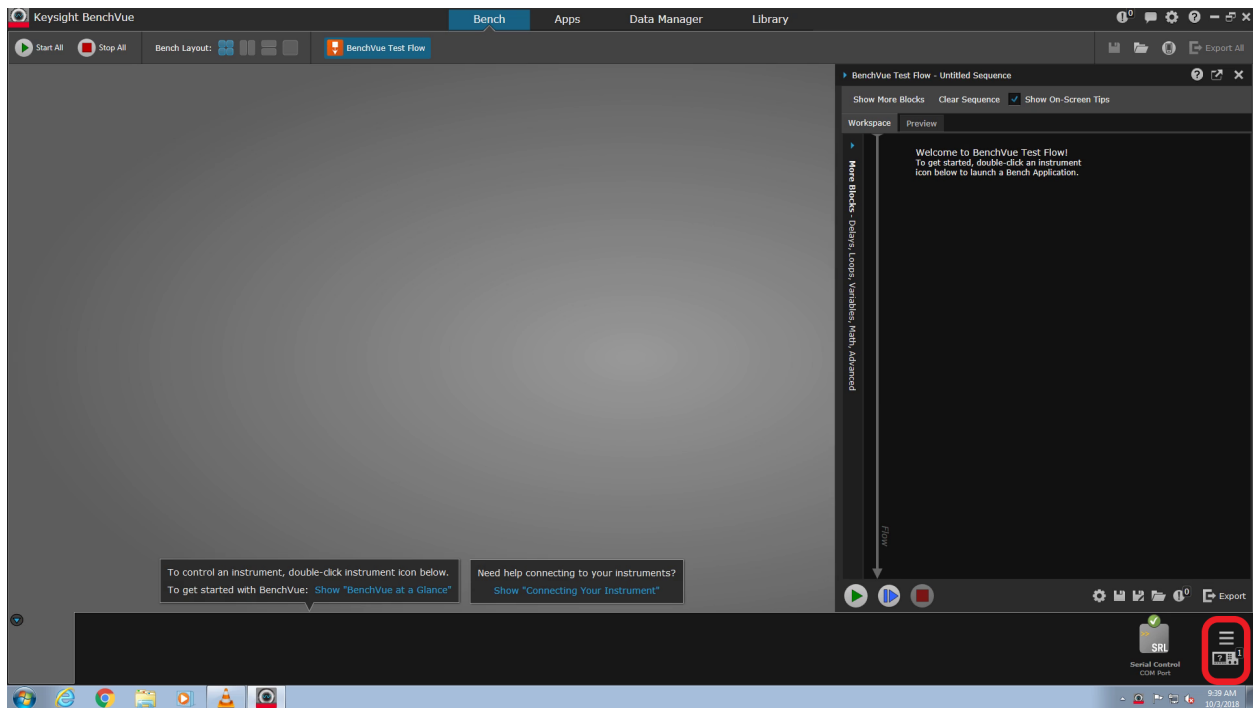


Figure 1: BenchVue when first opened. If your bench equipment does not appear on the bottom bar, press the icon on the lower right, outlined in the figure above.

## 2 Running a BenchVue Sequence

BenchVue executes instruction provided in Sequence format. One can construct their own sequence using a drag-and-drop interface, but to get started we provide a sequence file that will ramp up the voltage source from 0V to 6V, and collect voltage and current measurements in 100 mV increments. Once collected the data is put into a Matlab compatible format and saved locally. The sequence file `iv_collect.bvseq` can be downloaded from the ECE 110 Lab website. Once downloaded, the file can be opened by clicking the Load icon on the bottom of the BenchVue Test Flow window. Once loaded we will see that BenchVue interfaces

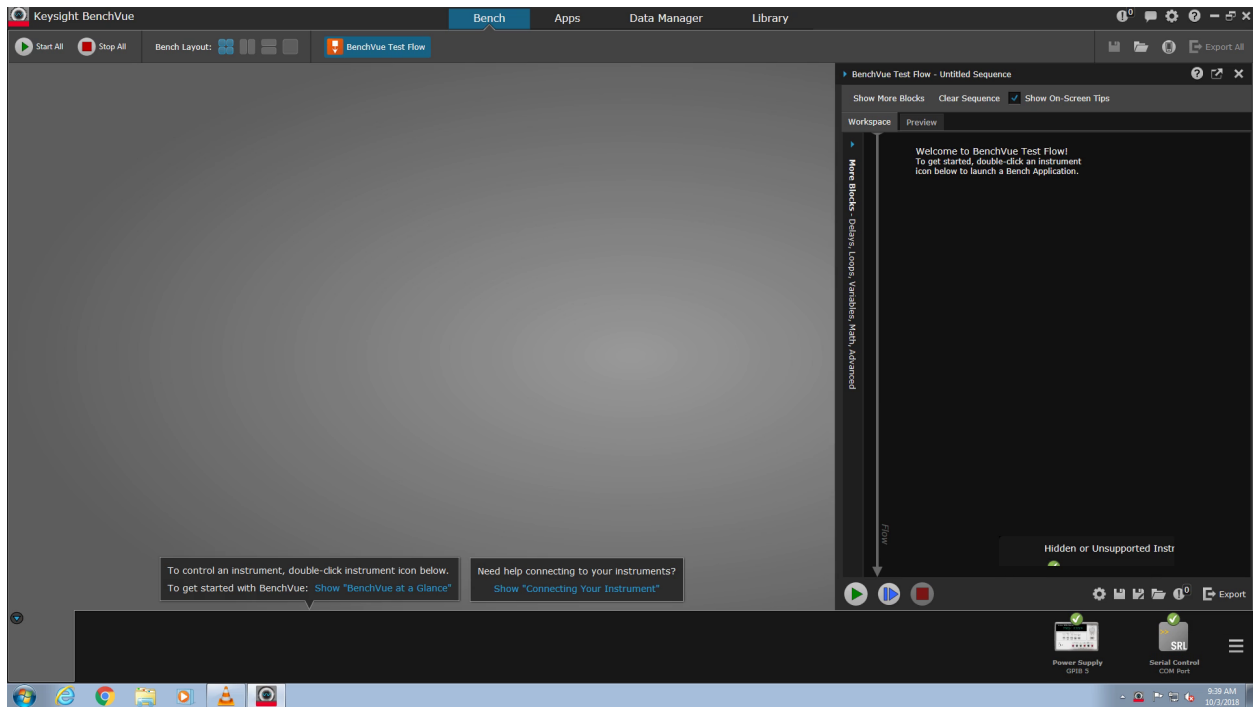


Figure 2: BenchVue with the DC power supply recognized and shown in the bottom bar.

with our DC power supply, and that the instructions of our measurement are displayed, as shown in Figure 3. At this point, make certain that the sequence is set to output the data to your EWS U: drive.

### 3 Importing BenchVue Data

BenchVue can export data in a number of standardized file formats, including an array format that works well with Matlab. Below we provide brief code snippets showing how to import these files into either Matlab or Python. In this case we measured the voltage and current provided by our bench power supply. Additionally we note that when working with data from the bench power supplies, BenchVue will append an extra data point with the value zero after the last point is measured. We can easily ignore this data point by simply not referring to its index. In the following code we assume the zero is appended to the end of the array. Additionally, sometimes erroneous data is appended to the beginning of our data. We will include sample code to remove data from beginning of the data vector as well.

#### 3.1 Matlab Code

```
%Your file location will differ than this test location
load('W:\documents\IV_output 2018-02-26')
current = Trace_1_GetCH1CurrentMeasurement;
voltage = Trace_1_GetCH1VoltageMeasurement;

%We re-define our variables and omit the last data point
current = current(1:end-1);
voltage = voltage(1:end-1);

%Example: If we want to ignore the first 3 values and last 2 values
%we can use the following commands
```

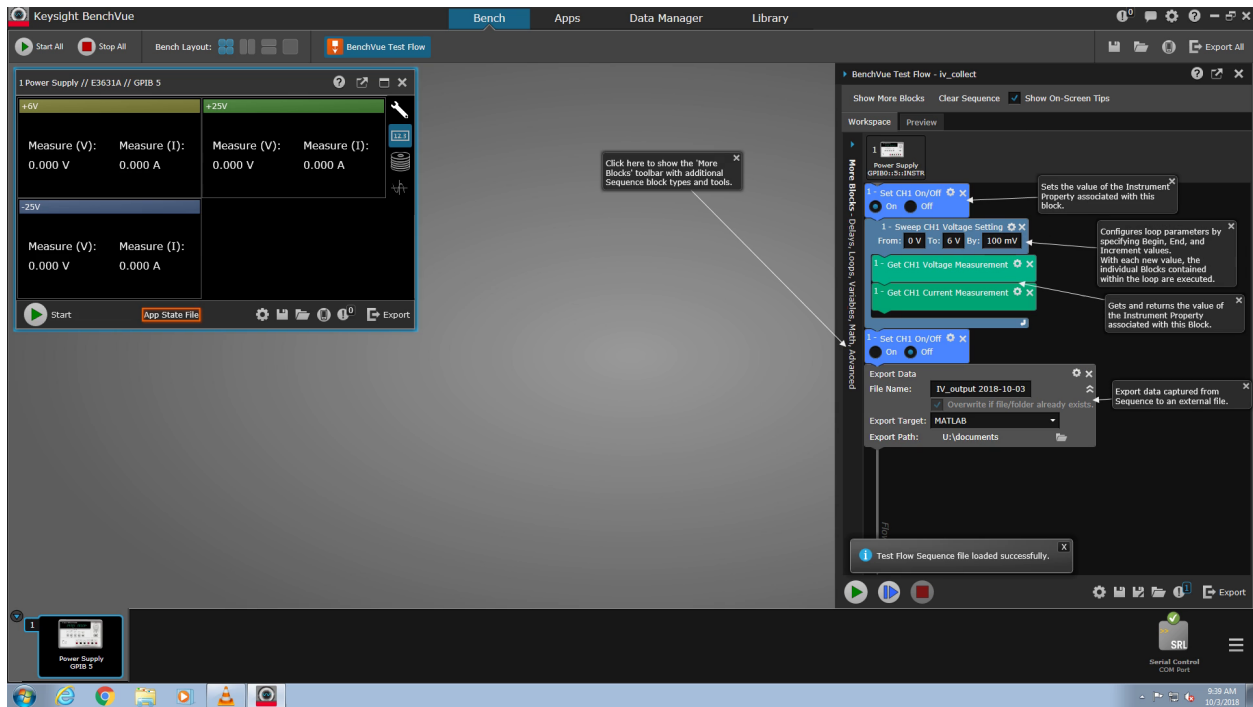


Figure 3: BenchVue with `iv_collect.bvseq` file loaded.

```
current = current(4:end-2);
voltage = voltage(4:end-2);
```

### 3.2 Python Code

```
import scipy.io
```

```
#Your file location will differ than this test location
mat = scipy.io.loadmat('W:\documents\IV_output 2018-02-26.mat')
```

```
current = mat['Trace_1_GetCH1CurrentMeasurement']
voltage = mat['Trace_1_GetCH1VoltageMeasurement']
```

```
#We re-define our variables and omit the last data point
current = current[0:-1]
voltage = voltage[0:-1]
```

```
#Example: If we want to ignore the first 3 values and last 2 values
#we can use the following commands
current = current[3:-2]
voltage = voltage[3:-2]
```

## 4 Working On Data Subsets

In lab we often collect data about our circuits, measuring multiple data points as functions of time, voltage, or current. Once we have our measurement data we can perform polynomial curve fitting in order to determine a polynomial that best replicates the properties of our data. Lower order polynomials can only represent

simple behavior, with a 1<sup>st</sup>-order polynomial simply being a linear representation in the form of  $y = mx + b$ . For such a case, for some given data points of  $y$  and  $x$ , we must determine the values  $m$  and  $b$ .

We can apply a polynomial fit to an entire dataset, but sometimes we are interested in the behavior of distinct subsets of our measurement data. For instance, if we are taking IV measurements of a motor we will see two very different trends in our data as the motor transitions from being stalled to when it starts moving. In these cases it is helpful to make new variables for these relevant subsets. Below are two code snippets that show how to split an array with one new variable holding the first 5 values, and the second variable holding the remaining values.

## 4.1 Matlab Code

```
voltage = [8.837, 8.772, 8.759, 8.707, 8.688, 8.688, 8.627, 8.627, 8.586, ...  
          8.514, 8.375, 7.99];
```

```
%Note that Matlab indexing starts at one and the end index term is inclusive  
voltage_subset1 = voltage(1:5)  
voltage_subset2 = voltage(6:end)
```

## 4.2 Python Code

```
#Create arrays of the provided data points  
voltage = np.array([8.837, 8.772, 8.759, 8.707, 8.688, 8.688, 8.627, 8.627,  
                  8.586, 8.514, 8.375, 7.99])
```

```
#Note that Python indexing starts at zero and that the end index term is exclusive  
voltage_subset1 = voltage[0:5]  
voltage_subset2 = voltage[5:]
```

# 5 Polyfit Sample Code

In the following subsections we provide sample Matlab and Python code to determine the polynomial coefficients for some test values of current and voltage using `polyfit` commands (read the documentation of the function for further information). We then plot the original data and the output of the linear model. In Lab 6 you will be performing the polyfit on the data subset that contains the running motor data.

Try increasing the order of the `polyfit` function and plotting the higher-order polynomial representation!

## 5.1 Matlab Code

```
%Polyfit demo  
%By John D  
  
%Create arrays of the provided data points  
current = [0.0006, 87.99, 93.60, 131.5, 148.9, 188.8, 215.2, 215.2, 258.4, ...  
          347.7, 534.9, 800];  
  
voltage = [8.837, 8.772, 8.759, 8.707, 8.688, 8.688, 8.627, 8.627, 8.586, ...  
          8.514, 8.375, 7.99];  
  
%Plot the original data  
figure;  
plot(voltage,current,'DisplayName','Original data points');  
  
%Perform a polyfit on the data points voltage and current  
%The corresponding coefficients for a 1st order polynomial are stored in p
```

```

p = polyfit(voltage,current,1);

%Plot the linear model, in (m*V)+b form
hold on;
plot(voltage,voltage*p(1) + p(2), 'DisplayName', '1st-order polynomial');

title('Linear curve fitting')
xlabel('Voltage (V)')
ylabel('Current (mA)')
legend('show')
grid('on')

```

## 5.2 Python Code

```

# -*- coding: utf-8 -*-
"""
Polyfit demo
@author: John O
"""

import numpy as np
import matplotlib.pyplot as plt

#Create arrays of the provided data points
current = np.array([0.0006, 87.99, 93.60, 131.5, 148.9, 188.8, 215.2, 215.2,
                    258.4, 347.7, 534.9, 800])

voltage = np.array([8.837, 8.772, 8.759, 8.707, 8.688, 8.688, 8.627, 8.627,
                    8.586, 8.514, 8.375, 7.99])

#Plot the original data
plt.plot(voltage,current, label='Original data points')

#Perform a polyfit on the data points voltage and current
#The corresponding coefficients for a 1st order polynomial are stored in p
p = np.polyfit(voltage,current,1)

#Plot the linear model, in (m*V)+b form
plt.plot(voltage, voltage*p[0] + p[1], label='$1st-order polynomial')

plt.title('Linear curve fitting')
plt.xlabel('Voltage (V)')
plt.ylabel('Current (mA)')
plt.legend()
plt.grid('True')
plt.show()

```