# Lab 2 – Temperature Measurement System

## GOALS

1) Build and test a thermistor circuit.
2) Write an Arduino program to acquire and send voltage data to a computer using the serial port.
3) Write a MATLAB program to:
    a. Read data from the serial port.
    b. Process the raw data into temperature.
    c. Display a plot of temperature vs time.
4) Perform some fun experiments with your temperature measurement system!

## GENERAL GUIDELINES

1) Each student must build his/her own circuit.
2) Due to the limited number of test stations, some students need to pair up and share the test equipment.
3) Students are encouraged to help each other. Of course, Buma will be around to provide assistance as well.
4) Ask questions! The more questions you ask, the more you learn! ☺

## REQUIRED MATERIALS

- Lab kit + power supply + oscilloscope + test probes
- Arduino Uno + USB cable + computer with Arduino IDE and MATLAB
- AD620 instrumentation amplifier (one)
- 1.0 kohm (brown/black/red) resistors (two)
- 10 kohm thermistor (comes with a long cable)
- 10 kohm resistors (brown/black/orange) (three)
- 100 kohm resistor (brown/black/yellow) (one)

**Fig. 1: Thermistor (Vishay BC Components).**

## LAB ACTIVITIES (4 PARTS)

1) Set up temperature measurement hardware.
2) Develop the Arduino data acquisition program.
3) Develop the MATLAB data processing and display program.
4) Perform some experiments with your *awesome* temperature measurement system.

## INTRODUCTION

In this lab, you will develop a temperature measurement system. A system block diagram is shown in Fig. 2. Here's how it works:

1) The thermistor (in a quarter bridge) converts temperature to a voltage signal.
2) The instrumentation amplifier increases the signal amplitude.
3) The amplifier output is recorded by the Arduino and voltage data is sent to the computer.
4) The computer processes the data (using MATLAB) to produce a plot of temperature versus time.

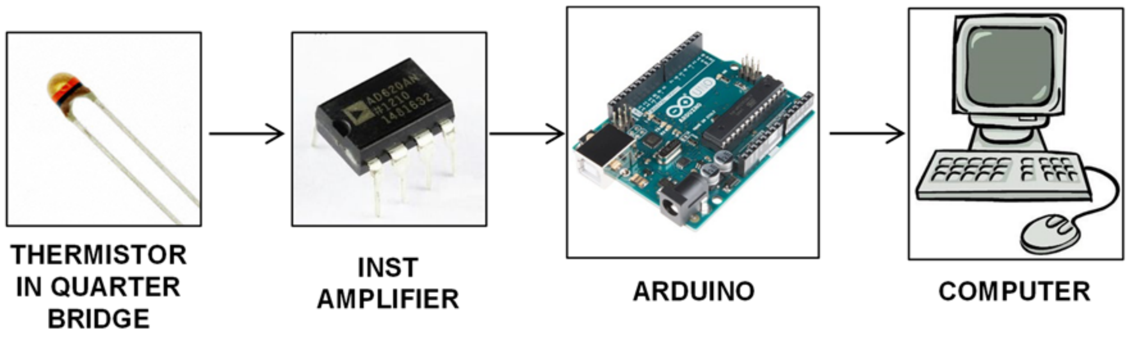


**Fig. 2: Overall temperature measurement system.**

## PART 1: THERMISTOR CIRCUIT

The thermistor circuit is shown in Fig. 3.

- **Task 1a**: Build the quarter bridge circuit and instrumentation amplifier (Fig. 3).
  - **Keep your wiring neat!** You can consult Buma's (or a neighbor's) breadboard for guidance.
    - You can re-use the AD620 chip and other components from Lab 1.
  - **Properly color code your wiring**!

| | | |
|---|---|---|
| ➢ RED | = | +5V |
| ➢ BLACK | = | GND |
| ➢ YELLOW | = | other |

  - The thermistor cable has two wires that plug into the board – it doesn't matter which wire goes to power.

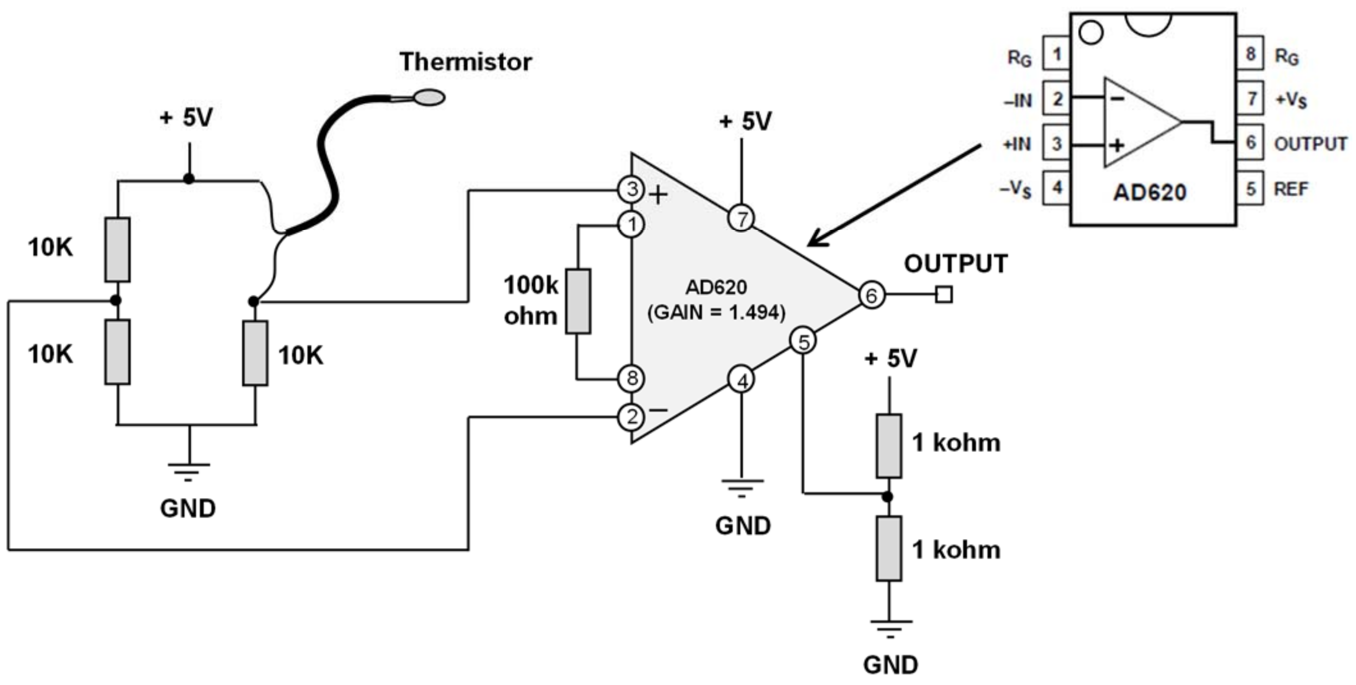


**Fig. 3: Schematic for thermistor circuit. Remember to color-code your wiring!**

- **Task 1b**: Test the circuit.
  - Use the oscilloscope to measure the output voltage at Pin 6 of the instrumentation amplifier.
    - Turn on the scope, then press "Force Trigger" to enable local (e.g. Front Panel) control
    - Press "Default Setup" to reset the scope
    - Make sure both the scope and probe are set to "1X"
    - Adjust the scope's vertical setting to 1V/div – the trace should be near 2V.
  - Zoom in on the trace.
    - First, center the trace in the middle of your screen (use the small knob to scroll down).
    - Next, zoom in by changing the vertical setting to 500 mV/div.
    - Finally, squeeze the thermistor between your thumb and index finger – the voltage should increase by roughly 200 mV or so. When you let go the voltage should sloooowly return to its original value.
      - If you have cold fingers and cannot see a voltage change, you can either:
        - Do about 386 jumping jacks to warm up your body.
        - Ask a classmate with hotter fingers to test your circuit.

(End of Part 1)

## PART 2: ARDUINO

For this lab, the Arduino will record Vmeas and send it to the PC. Unlike Lab 1, this lab will involve more communication between the Arduino board and the PC. This communication is needed to minimize any lost data during transmission. An outline of the Arduino code is shown below:

(a) **Setup( )** routine
- Initialize the serial port
- Send message to PC

(b) **Loop( )** routine
- Wait for message from PC
- If PC requests data,
  - Record ADC output
  - Compute Vmeas
  - Send Vmeas to computer
  - Wait 500 ms

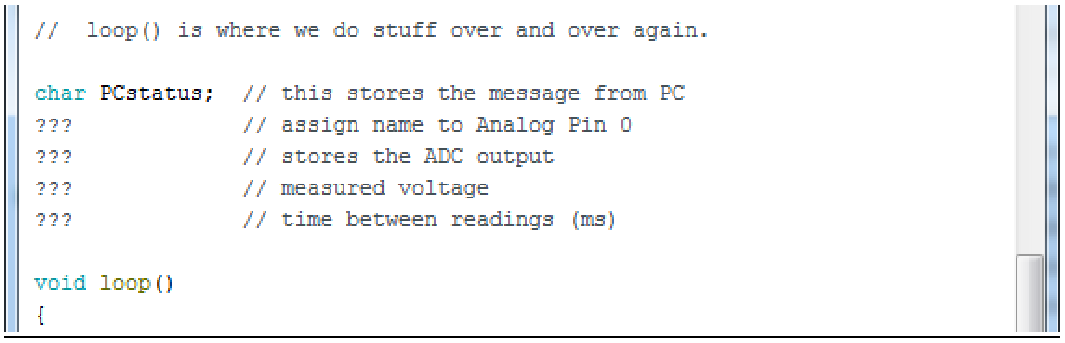
- <u>**STEP 2a**: Download and unzip the Arduino template</u>.
  - Go to the course website and download "Lab2_files.zip" to your desktop.
  - Double-click on the zipped folder.
  - **Select "Extract all files" (near top of the window) to unzip the contents of the folder**.
  - Double-click the extracted "Lab2_Arduino" folder and open the template.
  - You can consult the Lab1 handout or Google for help and/or examples using Arduino functions.

- <u>**STEP 2b**: Write the **setup( )** routine.</u>
  - Initialize the serial port using the **Serial.begin** command
    - Replace the "**???**" with appropriate code.
    - Use the **Serial.begin** command and set the data rate to 9600 bits per second.
    - Don't forget the semicolon at the end of each line!
  - Send message to PC
    - Use the **Serial.println** command to send the single letter "**a**". Double quotes are important!

- **STEP 2c**: Declare variables for the **loop( )** routine.
  - The loop routine uses five variables. Declare these before the loop( ) routine. As shown in Fig. 4, Buma wrote the first one for you (he is so kind ….). Replace any other **???** with appropriate code.

| Data Type | Name | Value | Purpose |
|---|---|---|---|
| **char** | **PCstatus** | | Stores one byte of text data from the PC |
| **int** | **A0_PIN** | **0** | Name of Analog Pin 0 |
| **int** | **ADCoutput** | | Stores the ADC output |
| **float** | **Vmeas** | | Measured voltage |
| **int** | **T** | **500** | Time between readings (ms) |

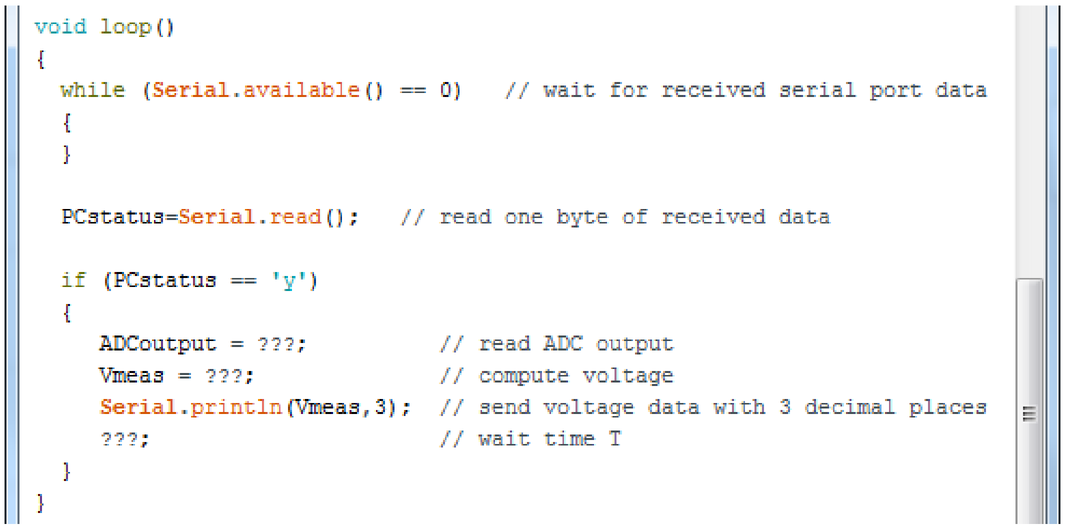
    - Don't forget semicolons at the end of each line!



```
//  loop() is where we do stuff over and over again.

char PCstatus;  // this stores the message from PC
???             // assign name to Analog Pin 0
???             // stores the ADC output
???             // measured voltage
???             // time between readings (ms)

void loop()
{
```

**Fig. 4: Declare some variables before the loop( ) routine.**

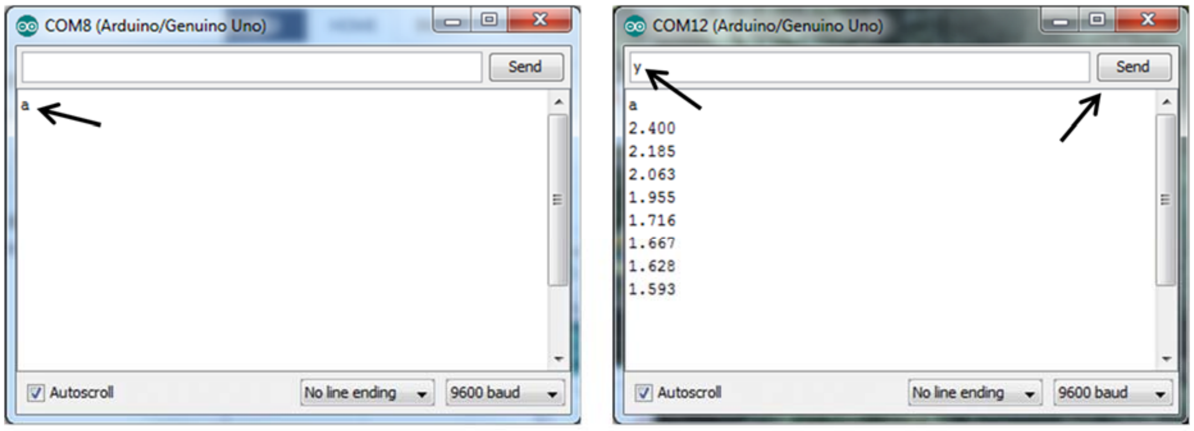
- **STEP 2d**: Write the **loop( )** routine.
  - Replace all the ??? so that your code looks like Fig. 5. Comments about the code.
    - The **Serial.available** command checks if the serial port has received any data.
      - ➢ The **while** loop repeats this process and stops when data has been received.
      - ➢ The **Serial.read** command reads one byte of received data from the serial port.
    - The **Serial.read** command reads one byte of received data and stores the value in **PCstatus**.
    - If **PCstatus** is the character "**y**", then we proceed to make a voltage measurement:
      - ➢ Use the **analogRead** command to read the data from A0_PIN.
      - ➢ Convert **ADCoutput** to **Vmeas** (see Lab1 handout if you forgot how to do this).
      - ➢ **Serial.println(Vmeas, 3)** sends the **Vmeas** value (3 decimal places)
      - ➢ Use the **delay** command to wait **T** seconds before the next sample acquisition.



```
void loop()
{
  while (Serial.available() == 0)   // wait for received serial port data
  {
  }

  PCstatus=Serial.read();   // read one byte of received data

  if (PCstatus == 'y')
  {
     ADCoutput = ???;          // read ADC output
     Vmeas = ???;              // compute voltage
     Serial.println(Vmeas,3);  // send voltage data with 3 decimal places
     ???;                      // wait time T
  }
}
```

**Fig. 5: The loop( ) routine checks for a message from the PC and then makes a voltage measurement.**

- <u>**STEP 2e**: Upload your code and observe the Arduino output using the **Serial Monitor** on the computer.</u>
  - You should notice the letter "**a**" on the first line (Fig. 6a).
    - Nothing else should be happening, since the Arduino has not received a "**y**" from the computer.
  - Type in the letter "**y**" in the command line and press the **Send** button (see Fig. 6b).
    - You should see a voltage value appear below the letter "**a**".
    - Your code works if a new voltage value appears every time you send "**y**" to the Arduino. Nice!



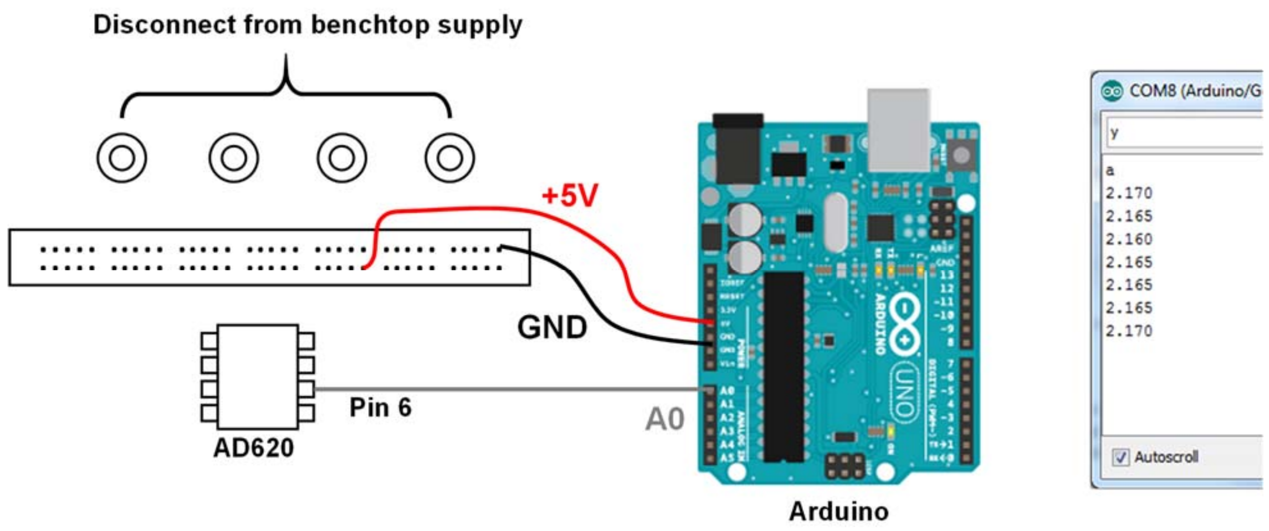
(a)
```
a
```

(b)
```
y
```
```
a
2.400
2.185
2.063
1.955
1.716
1.667
1.628
1.593
```

**Fig. 6: (a) The Serial Monitor should only show the letter "a" when the Arduino code starts. (b) Typing in "y" and pressing "Send" should cause the Arduino to record and send a voltage signal. Repeating this process results in new voltage values.**

- <u>**STEP 2f**: Connect your thermistor circuit to the Arduino board.</u>

- We want the thermistor circuit to be directly powered by the Arduino board.
  - First, **TURN OFF and DISCONNECT the benchtop power supply** from your breadboard.
    - ➢ You can put away the banana cables since they won't be needed anymore.
  - Second, **DISCONNECT the USB cable** from your Arduino board.
  - Third, use a **RED wire** to power your breadboard from the Arduino's +5V output (see Fig. 7).
    - ➢ It is OK to use a long wire (e.g. few inches long).
  - Then use a **BLACK wire** to connect the breadboard ground to the "GND" pin of the Arduino.
  - Finally, use a **YELLOW wire** to connect the amplifier output (Pin 6) to the "A0" pin of the Arduino.
- Re-connect the USB cable to the Arduino and re-run your Arduino code.
  - View the **Serial Monitor** to confirm that you get a fairly steady voltage (e.g. something like the right side of Fig. 7).
- Close the **Serial Monitor** when you have confirmed that everything works OK.



```
y
a
2.170
2.165
2.160
2.165
2.165
2.165
2.170
```

**Fig. 7: Disconnect the benchtop supply, then power your breadboard from the Arduino's +5V output and GND. Pin 6 of the AD620 chip is connected to A0 of the Arduino. It is perfectly OK to use long wires (e.g. few inches) for all three connections. When you re-run your Arduino code, the Serial Monitor should show a fairly constant voltage.**

(End of Part 2)

## PART 3: MATLAB WARM-UP EXERCISES

Now that we've acquired **Vmeas** using the Arduino, we need to process **Vmeas** and display a plot of temperature on the computer. We'll do this using MATLAB! Buma will assume that you know some MATLAB, but have no experience using MATLAB to communicate with an external device.

- In the "Lab2_files" folder you recently downloaded from the course website, double-click on the "Lab2_MATLAB.m" file.

## EXERCISE 3.1

Our first MATLAB exercise is to receive a single voltage measurement from the Arduino. How to do this? Fig. 8 is the flow chart for the Exercise 3.1 MATLAB program.

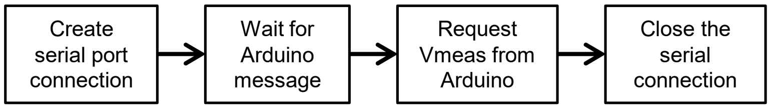
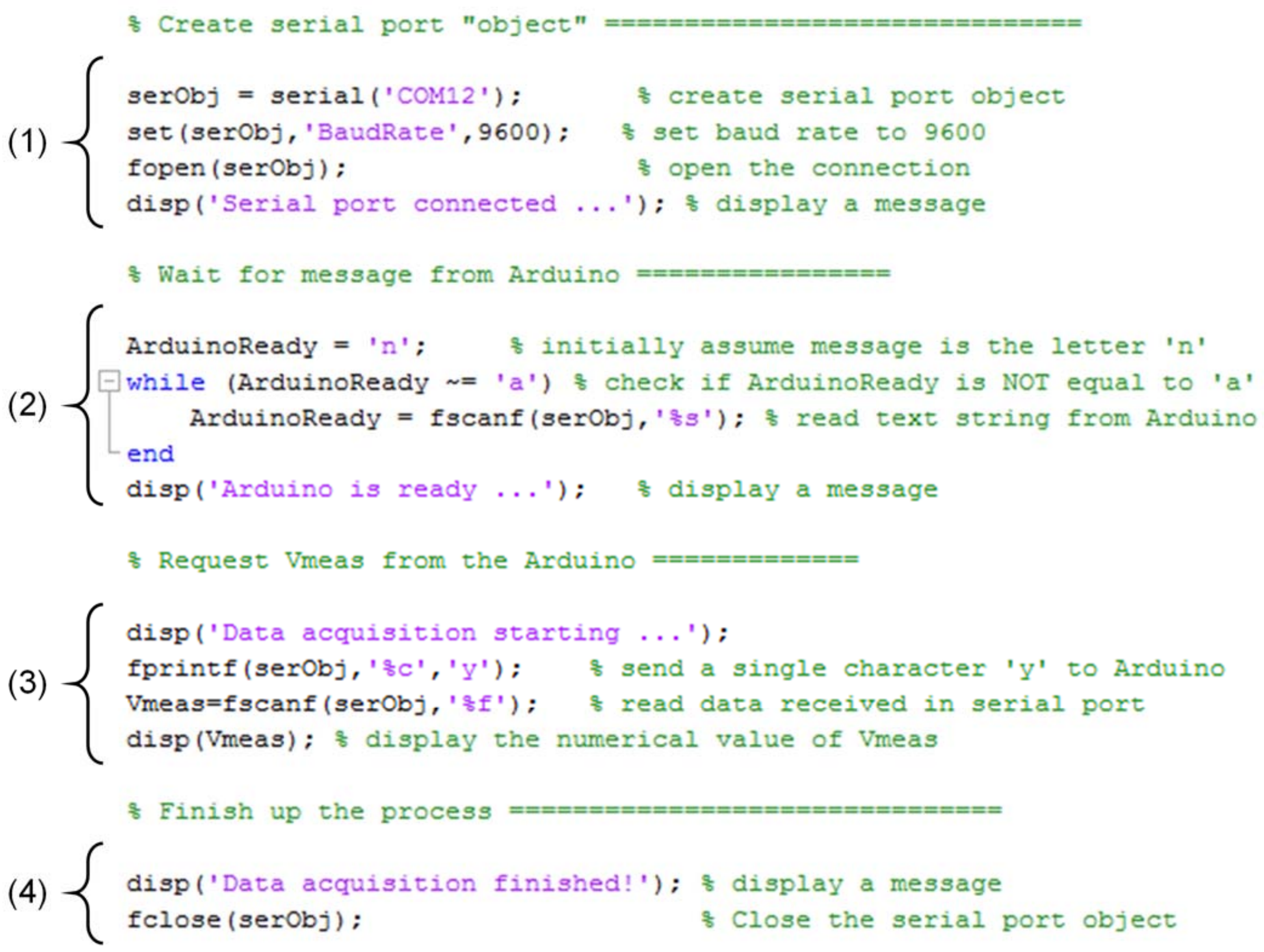

Create serial port connection
Wait for Arduino message
Request Vmeas from Arduino
Close the serial connection

**Fig. 8: Flow chart of the MATLAB program for Exercise 3.1.**

- **STEP 3.1a**: Replace any "???" in the template so that your code looks like Fig. 9. Comments about the code are shown below:
  - Creating the serial port connection:
    - The **serial** command creates a "serial port object" that we will name **serObj**.
      - **Make sure the COM number is the same as in your Arduino code!**
    - We are using the **set** command to configure the data rate for 9600 bits per second.
    - The serial connection is started by "opening" **serObj** using the **fopen** command.
    - The **disp** command displays a message in the MATLAB **Command Window**.
  - Waiting for the Arduino message:
    - The variable **ArduinoReady** stores the message from the Arduino.
      - We will initially assume **ArduinoReady** is the letter '**n**'.
    - We are using a **while** loop to repeatedly check for a message from the Arduino.
      - The loop keeps going as long as **ArduinoReady** is **NOT equal** to '**a**'.
    - The **fscanf** command reads the received data contained in **serObj**.
      - The **%s** option means that the message is a text string.



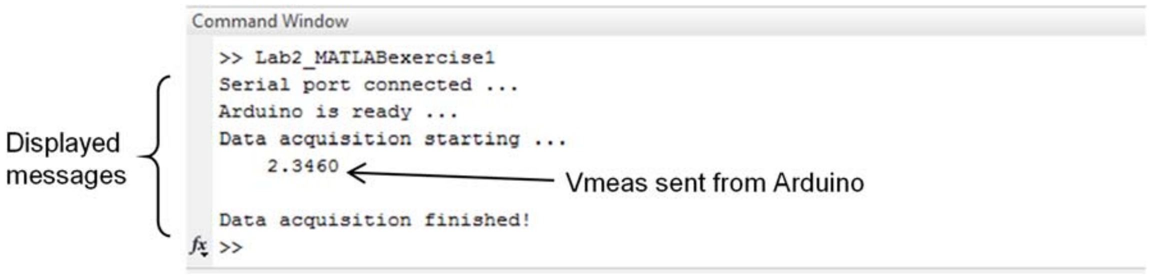
```
% Create serial port "object" ===============================

serObj = serial('COM12');        % create serial port object
set(serObj,'BaudRate',9600);    % set baud rate to 9600
fopen(serObj);                   % open the connection
disp('Serial port connected ...'); % display a message

% Wait for message from Arduino ================

ArduinoReady = 'n';       % initially assume message is the letter 'n'
while (ArduinoReady ~= 'a') % check if ArduinoReady is NOT equal to 'a'
    ArduinoReady = fscanf(serObj,'%s'); % read text string from Arduino
end
disp('Arduino is ready ...');   % display a message

% Request Vmeas from the Arduino =============

disp('Data acquisition starting ...');
fprintf(serObj,'%c','y');     % send a single character 'y' to Arduino
Vmeas=fscanf(serObj,'%f');    % read data received in serial port
disp(Vmeas); % display the numerical value of Vmeas

% Finish up the process ===============================

disp('Data acquisition finished!'); % display a message
fclose(serObj);                     % Close the serial port object
```

**Fig. 9: Your MATLABcode should look like this. Do NOT forget the semicolons!**

- Requesting Vmeas from the Arduino:
  - The **fprintf** command sends the letter '**y**' to the Arduino via **serObj**.
    - The **%c** option means that the message is a text character.
  - The **fscanf** command reads the received data contained in **serObj**.
    - The **%f** option means the data is a floating point number.
- Finishing up the process:
  - The **fclose** command closes **serObj**, which terminates the serial port connection.

- **STEP 3.1b**: Save your program, then run it by clicking on the "Run" button (with green triangle) at the top of the MATLAB window.
  - The command window (bottom of MATLAB window) should look something like Fig. 10.
    - If you get a "COM not available" error, it most likely means you did not use the correct COM port number when using the **serial** command.

- If you get errors:
  - First, manually close the serial object via the command line by doing the following:
    - >> **fclose(serObj)**
    - Why do this? When your program exits prematurely due to an error, it leaves the serObj open. MATLAB does not like this and will prevent further communication through the serial port. Highly annoying!
  - In your program, make sure:
    - You properly use semicolons.
    - You properly use SINGLE quotes instead of double quotes. For example, **‘ %s ’** is correct while **“ %s ”** is wrong.
- Once your program works, look at the Arduino while you click the MATLAB Run button:
  - The Arduino’s “L” LED should quickly blink a few times, followed by a quick flash of the “Tx” and “Rx” LEDs.



```
>> Lab2_MATLABexercise1
Serial port connected ...
Arduino is ready ...
Data acquisition starting ...
    2.3460

Data acquisition finished!
>>
```

**Fig. 10: The command window (bottom of MATLAB window) should look like this. You’ve just read data from the Arduino Uno!**

## EXERCISE 3.2

Our next MATLAB exercise is to make a plot of Vmeas streamed from the Arduino. Fig. 11 shows the flow chart.

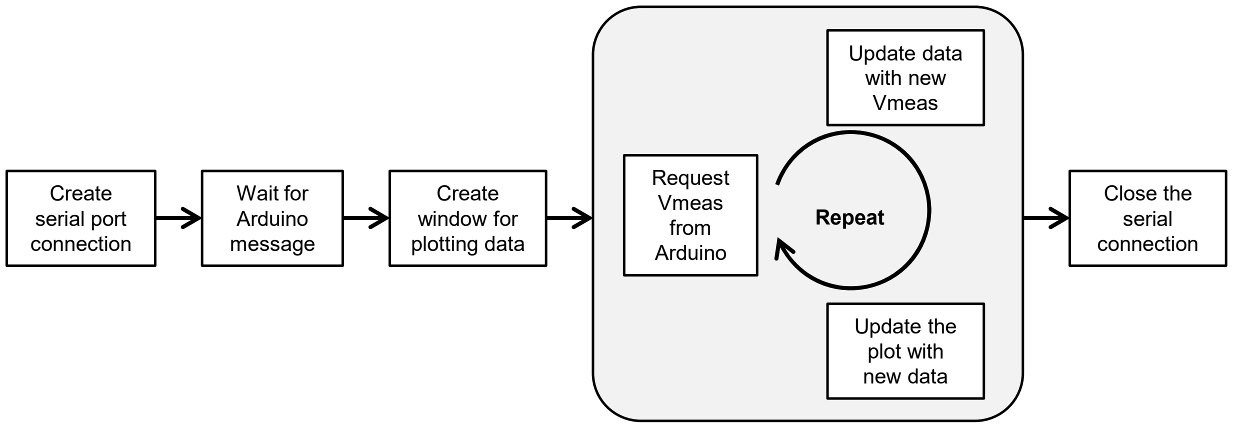
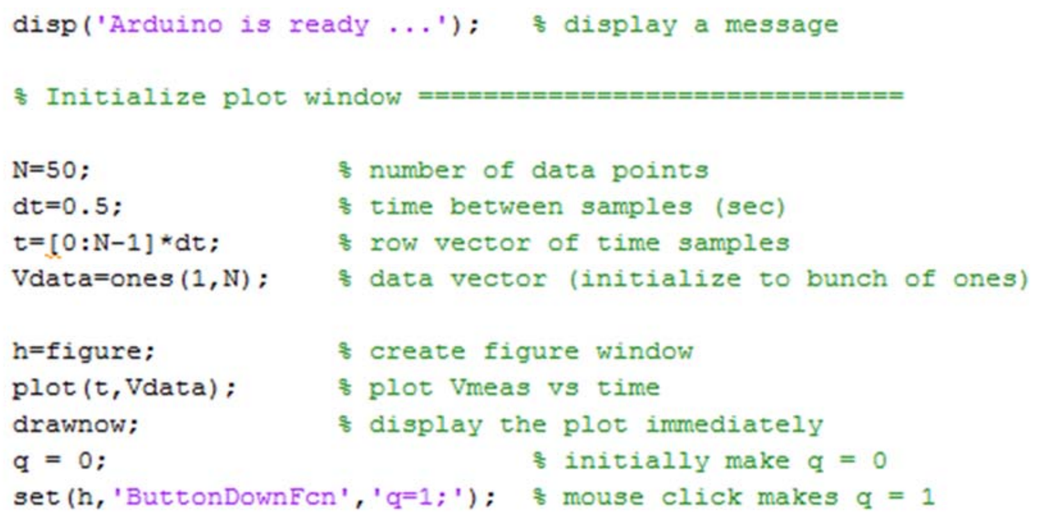

Create serial port connection
Wait for Arduino message
Create window for plotting data
Request Vmeas from Arduino
Update data with new Vmeas
Repeat
Update the plot with new data
Close the serial connection

**Fig. 11: Flow chart for the MATLAB program to plot Vmeas that is streamed from the Arduino.**

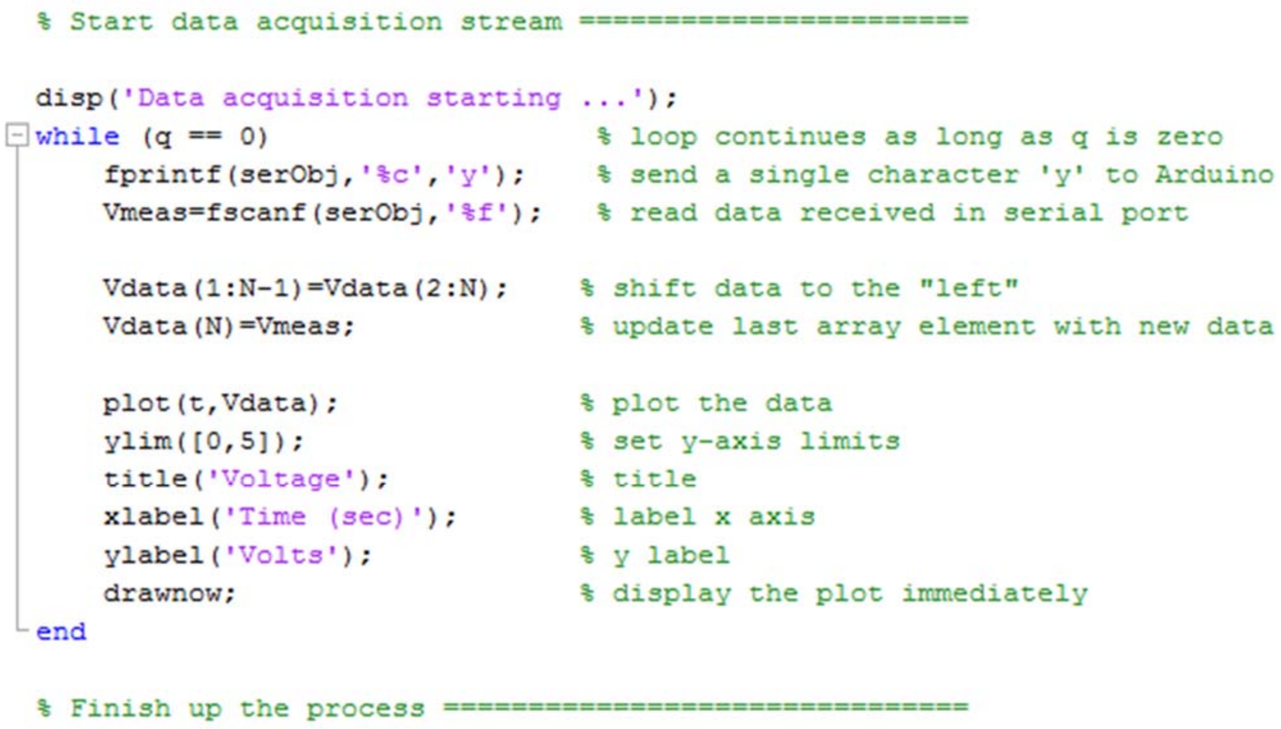
- **STEP 3.2a**: The first two sections of the MATLAB program are identical to your Exercise 3.1 code. Insert a new section of code, as shown in Fig. 12, just after the " **disp('Arduino is ready …');** " line of code. This new code will create the window for plotting data. Comments about the code are shown below:
  - We first need to create a 1-D array of time values for plotting.
    - The time vector **t** has a length of **N = 50** samples.
    - The time increment is **dt = 0.5** seconds.
    - The time vector is therefore [0, dt, 2dt, 3dt, . . . , 49dt] = [0, 1, 2, 3, . . . , N-1]dt
  - We next create an array of voltage values.
    - The voltage vector **Vdata** will initially be defined as a 1-D array of ones.
  - The **figure** command creates a window, where we assign **h** as the figure "name".
  - The **plot** command plots **t** on the x-axis and **Vdata** on the y-axis.
  - **drawnow** displays the plot immediately (this is necessary while the program is running).
  - We want the program to STOP when the user clicks on the figure window. We can do this using the **ButtonDownFcn** in MATLAB.
    - The **ButtonDownFcn** is a function that does something when the user clicks on the figure.
    - We use the **set** command to configure the **ButtonDownFcn** to do whatever we want.
      - We define a variable **q** that is initially equal to **0**.
      - When the use clicks on figure **h**, we want **ButtonDownFcn** to make **q = 1**.
      - Later in the code we'll see how q = 1 causes a while loop to exit.



```
disp('Arduino is ready ...');   % display a message

% Initialize plot window ==============================

N=50;                  % number of data points
dt=0.5;                % time between samples (sec)
t=[0:N-1]*dt;          % row vector of time samples
Vdata=ones(1,N);       % data vector (initialize to bunch of ones)

h=figure;              % create figure window
plot(t,Vdata);         % plot Vmeas vs time
drawnow;               % display the plot immediately
q = 0;                           % initially make q = 0
set(h,'ButtonDownFcn','q=1;');   % mouse click makes q = 1
```

**Fig. 12: This part of the code creates 1-D arrays for time and voltage, displays a plot, and configures the "ButtonDownFcn" feature.**

- **STEP 3.2b**: Now we need to implement a live feed of data from the Arduino! Replace all "???" with appropriate code shown in Fig. 13. Comments about the code are shown below:
  - The while loop keeps running as long as q is equal to 0.
  - Requesting **Vmeas** from the Arduino:
    - The **fprintf** command sends the letter '**y**' to the Arduino via **serObj**.
      - The **%c** option means that the message is a text character.
    - The **fscanf** command reads the received data contained in **serObj**.
      - The **%f** option means the data is a floating point number.
  - We want the most recent **Vmeas** value to be the LAST element in the Vdata array.
    - We shift the existing **Vdata** elements to the "left".
    - We then assign the last element in **Vdata** to be equal to **Vmeas**.
  - Plotting the updated data:
    - The **plot** command plots the updated **Vdata** as a function of **t**.
    - The **ylim** command sets the y-axis limits of the plot.
    - The **drawnow** command is necessary to display the updated plot immediately.
    - We then assign the last element in **Vdata** to be equal to **Vmeas**.
  - The last section of the program ("Finish up the process") is the same as before.



```
% Start data acquisition stream =======================

disp('Data acquisition starting ...');
while (q == 0)                        % loop continues as long as q is zero
    fprintf(serObj,'%c','y');         % send a single character 'y' to Arduino
    Vmeas=fscanf(serObj,'%f');        % read data received in serial port

    Vdata(1:N-1)=Vdata(2:N);         % shift data to the "left"
    Vdata(N)=Vmeas;                  % update last array element with new data

    plot(t,Vdata);                   % plot the data
    ylim([0,5]);                     % set y-axis limits
    title('Voltage');                % title
    xlabel('Time (sec)');            % label x axis
    ylabel('Volts');                 % y label
    drawnow;                         % display the plot immediately
end

% Finish up the process ===============================
```

**Fig. 13: The data acquisition loop does two things: (1) request Vmeas from the Arduino (2) display an updated plot.**

**STEP 3.2c**: Save your program, then run it by clicking on the "Run" button (with green triangle) at the top of the MATLAB window.

- If you get any errors, remember to manually close the serial object via the command line:
  - >> **fclose(serObj)**
- A plot should appear with a waveform that is scrolling to the left. An example is shown in Fig. 14.
  - When the program starts, the voltage will quickly jump to roughly 2.4V or so.
    - Pinching the thermistor causes the voltage to increase.
  - Releasing the thermistor causes the voltage to drop.
- The live data stream keeps going until you click the figure just outside the white plot area.

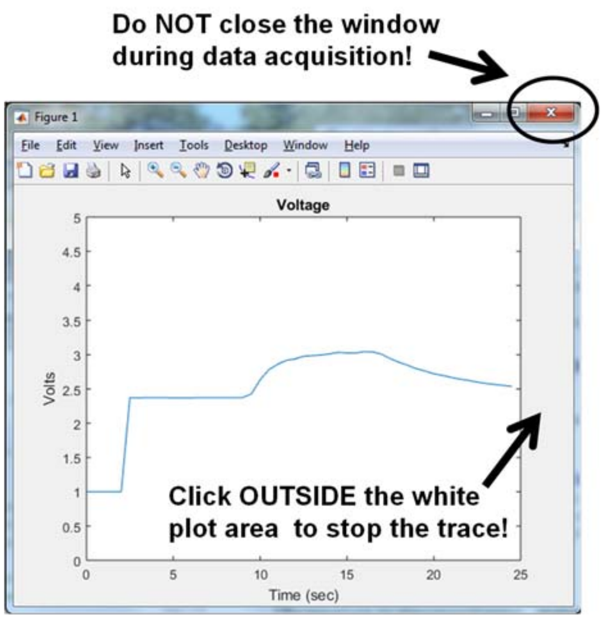

Do NOT close the window during data acquisition!
Figure 1
File Edit View Insert Tools Desktop Window Help
Voltage
Volts
Time (sec)
Click OUTSIDE the white plot area to stop the trace!

**Fig. 14: Example of streamed data from the Arduino. Click OUTSIDE the white plot area to stop the trace!**

(End of Part 3)

## PART 4: TEMPERATURE MEASUREMENT

OK, now it's time to implement the temperature measurement system! Our MATLAB code should receive one sample of **Vmeas** from the Arduino, convert **Vmeas** to **RT**, then use the Steinhart-Hart equation to compute temperature **Temp** in degrees Celsius, and update a scrolling plot of **Temp vs t**.

- From PreLab 2, **Vmeas** is converted to **RT** using the following two equations:

$$beta = (V_{MEAS} - V_{REF})/(A_d V_S); \qquad R_T = (10\ kohm)(0.5 - beta)/(0.5 + beta);$$

- The extended Steinhart-Hart equation is given below (T is in Kelvin):

$$T = 1/\left(A + B \cdot log\left(\frac{R_T}{R_{25}}\right) + C\left(\log\left(\frac{R_T}{R_{25}}\right)\right)^2 + D\left(\log\left(\frac{R_T}{R_{25}}\right)\right)^3\right)$$

  - In MATLAB, the natural logarithm function is **log** (it is not **ln**).
  - Both RT and R25 must have the same units (i.e. both are kohm)
  - You want temperature in degrees Celsius, so you need to slightly modify the above equation!

- **Task 4a**: Using your Exercise 3.2 code as a starting point, modify your code to produce a live plot of temperature. Comments about the code are shown below:
  - You should replace **Vdata** (voltage vector) with **Temp** (temperature vector).
    - In Exercise 3.2, we initially defined **Vdata** as an array of **1**s (e.g. [1, 1, 1, …, 1]).
    - When creating **Temp**, it should be initialized as an array of **25**s (e.g. [25, 25, 25, …, 25]).
  - It is a good idea to define any constants (e.g. Ad) before the data acquisition loop.
    - It is fine to assume **Ad** = 1.494 and **Vref** = 2.5.
    - Be careful with units! Since the equation for **RT** involves kohms, you should use **R25 = 10**.
    - The Steinhart-Hart coefficients for the thermistor are:
      - **A = 3.354016e-3; B = 2.569850e-4; C = 2.620131e-6; D = 6.383091e-8;**
    - The y-axis limits of the temperature plot should be between 10 and 35 degrees Celsius.
  - Due to the lack of a calibrated temperature source, we will not calibrate our system. Please do not be too disappointed … ☹

- **Task 4b**: When you run your code, a plot should appear with a waveform that is scrolling to the left.
  - If you get any errors, remember to manually close the serial object via the command line:
    - >> **fclose(serObj)**
  - A successful example plot is shown in Fig. 15.
    - Remember that the temperature is initially assumed to be 25 °C.
    - The actual temperature in Buma's office was around 22 °C, which explains the sudden drop in the waveform.
      - The temperature increased when Buma pinched the thermistor.
      - The temperature dropped when the thermistor was released.
      - The acquisition was stopped by clicking OUTSIDE the white plot area of the figure.

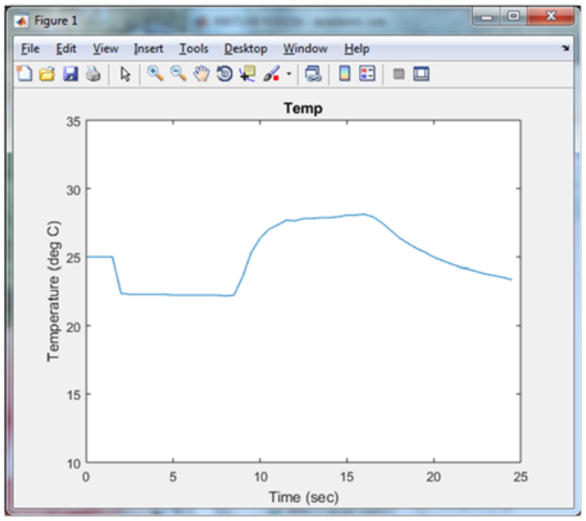


**Fig. 15: Example of streaming plot showing temperature rise and fall versus time. Click OUTSIDE the white plot area to stop the trace!**

- <u>**Task 4c**: Record and save a "warm" plot with your thermometer</u>.
  - You can pinch the thermistor with your fingers or dip the thermistor tip into a cup of warm water.
- <u>**Task 4d**: Record and save a "cold" plot (dip the thermistor tip into ice water) with your thermometer</u>.
- <u>**Task 4e: Demo your awesome system to Buma**</u>!

(End of Lab 2)