Temperature Sensors & Recursive Least Squares

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #21

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Introduction:

The next few lectures will look at reading sensors to a Pi-Pico

- Sensors allow the Pi-Pico know what's happening in the real world
- note: There is an entire course on sensors called *Instrumentation*

This lecture focuses on measuring temperature using

- Analog sensors:
  - Thermistors
  - TMP36
- Digital Sensors
  - DS18B20

With these

- Measure the temperature of a cup of how water as it cools off
- Compute the thermal time constant in real time using recursive least squares

Measuring Resistance:

The A/D on the Pi-Pico measures voltage

- 12-bit A/D
- 0V to 3.3V

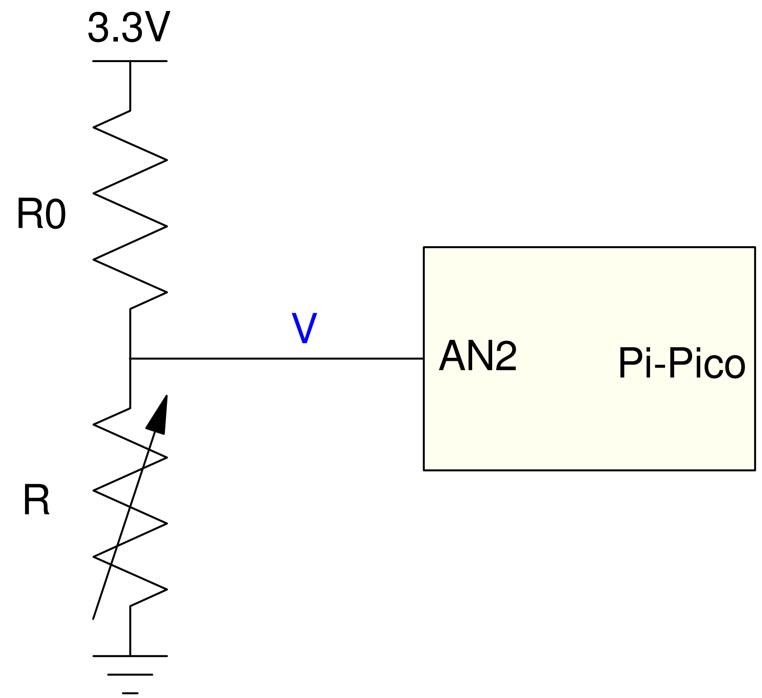
To measure a resistance

- Convert resistance to voltage
- Voltage divider works

$$V = \left(\frac{R}{R+R_0}\right)3.3V$$

By measuring V, you can compute R

$$R = \left(\frac{V}{3.3-V}\right)R_0$$



Resolution of Ohm-Meter

Depends upon the resistor values.

- Assume R0 = R = 1k
- The nominal voltage you read is 1.65V

$$V = \left(\frac{1000}{1000+1000}\right) 3.3V = 1.65V$$

The A/D has 12-bits

- The smallest change in voltage you can detect is

$$dV = \frac{3.3V}{4095} = 805.9\mu V$$

The smallest change in resistance you can detect is

- What produces a 805.9uV change in voltage

$$V = 1.65V + 805.9\mu V$$

$$R = \left(\frac{V}{3.3-V}\right) 1000 = 1000.977\Omega$$

$$dR = 0.977\Omega$$

Temperature Sensors: Thermistor

Once you can measure resistance, you can measure pretty much any sensor whose output is resistance.

A thermistor is one such sensor.

A thermistor is a piece of silicon

- Insulator at 0K
- Conductor above 0K

As temperature goes up

- More electrons escape their covalent bonds
- Each electron also creates a hole
- More charge carriers means less resistance

Thermistor Models

As temperature goes up, resistance drops

General model (K = Kelvin)

$$R = \exp\left(a + \frac{b}{K} + \frac{c}{K^2} + \frac{d}{K^3} + \ldots\right)$$

More terms gives a better model over a wider range

2-Term Model:

$$R = \exp\left(a + \frac{b}{K}\right)$$

$$R = R_{25} \cdot \exp\left(\frac{B}{T+273} - \frac{B}{298}\right)$$

where

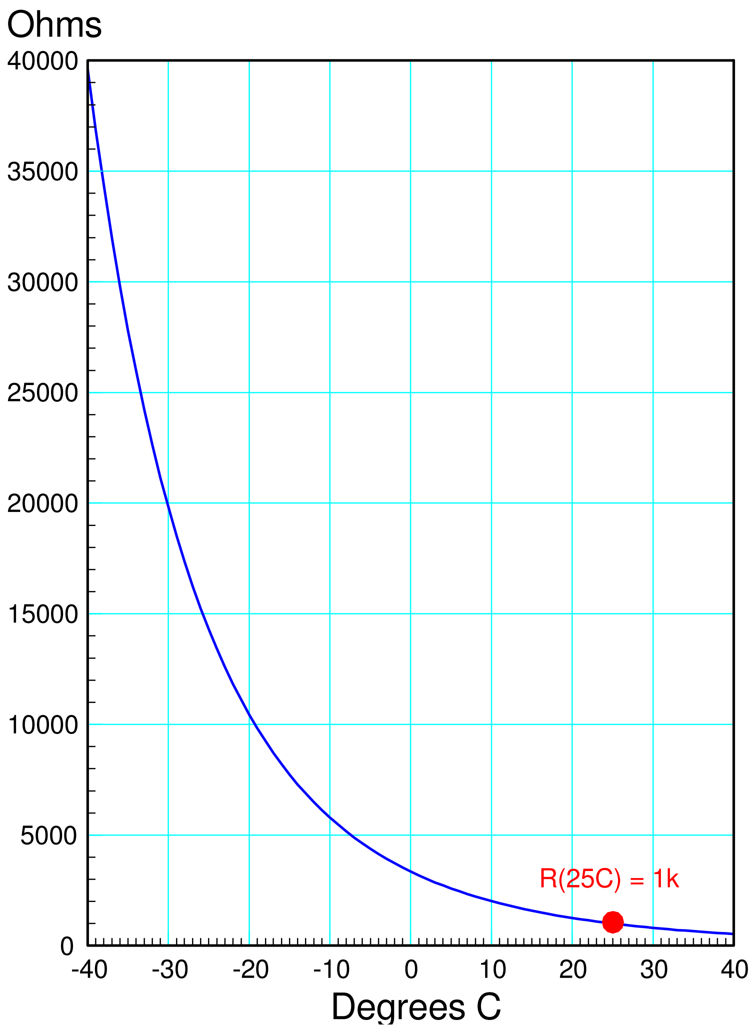
- T is the temperature in degrees C (T + 273 = K),
- R25 is the resistance at 25C, and
- B is a constant

If you look up the data sheets for a thermistor, you can find the B parameter

- Digikey Part Number: 495-2156-ND
- R25: 1k
- B25/100: 3930
- Dissipation Factor: 3.5 mW/K

This gives you the model for the thermistor:

$$R = 1000 \cdot \exp\left(\frac{3930}{T+273} - \frac{3930}{298}\right)\Omega$$



Add a voltage divider

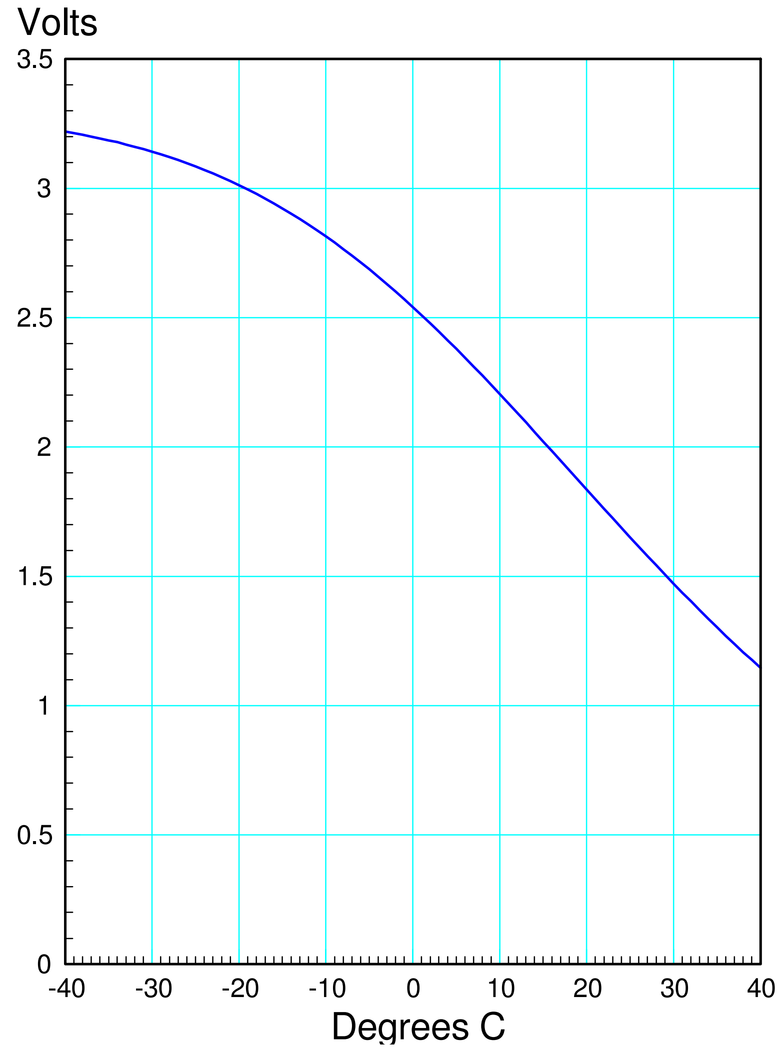
$$V = \left(\frac{R}{R+1000}\right) 3.3V$$

Solving backwards

$$R = \left(\frac{V}{3.3-V}\right) 1000\Omega$$

$$T = \left(\frac{3930}{\ln\left(\frac{R}{1000}\right)+\left(\frac{3930}{298}\right)}\right) - 273$$

Net: You can compute the temperature given a voltage measurement



Resolution: The resolution is again the smallest change in temperature you can detect with the 12-bit A/D on the Pi-Pico.

Assuming 25C

$$R = 1000\Omega$$

$$V = 1.65V$$

The smallest change in voltage you can detect is 805.9uV.

$$V = 1.65V + 805.9\mu V$$

When R = 1000 Ohms, the smallest change in resistance you can detect is 0.977 Ohms

$$R = 1000.977\Omega$$

The corresponding temperature is

$$T = \left(\frac{3930}{\ln\left(\frac{R}{1000}\right) + \left(\frac{3930}{298}\right)} \right) - 273$$

$$T = 24.9779C$$

The difference from 25C is the resolution in temperature:

$$dT = T - 25 = -0.02207C$$

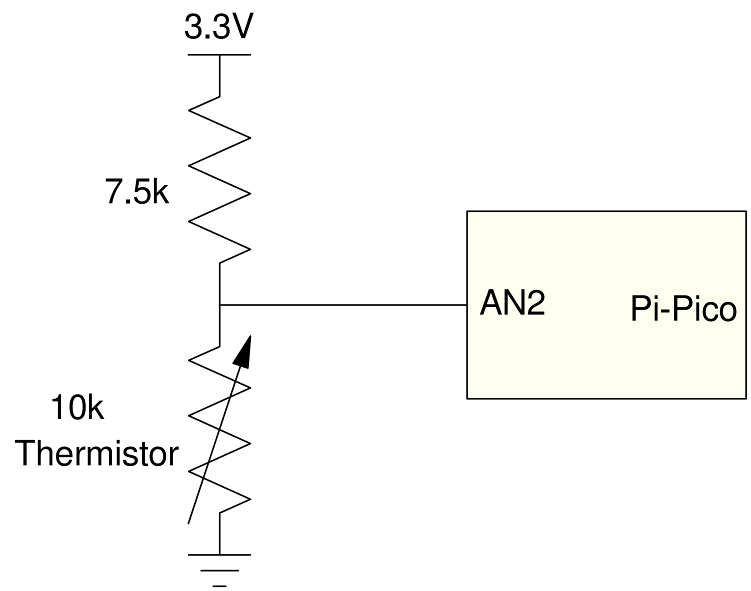
With this setup, a Pi-Pico can measure temperature with a resolution of 0.022 degrees C.

Example: Measure the temperature of an LED light bulb

- 20W LED bulb
- Measure temperature when it's turned on

How hot does the bulb get?

How long does it take to warm up?



Hardware: Use a thermistor

- Nominal = 10k @ 25C

Use a votlage divider

- 7.5k
- Goal is to get 1.65V output
- (max sensitivity)

Software:

- Uses timer interrupts to sample once per second
- Measures the voltage of AN2, and
- Computes the corresponding temperature

```
while(time < 300):
    while(flag == 0):
        pass
    flag = 0

    V = kV * a2d2.read_u16()
    R = V / (3.3-V) * 7500
    Temp = 3930 / ( log(R/10000) + (3930/298) ) - 273

    print(time, V, R, Temp)

    file1.write(str('{: 6.1f}'.format(time)) + "  ")
    file1.write(str('{: 7.4f}'.format(Temp)) + "  ")
    file1.write("\n")

    time += T
```

Result:

Temperature rises as a decaying exponential

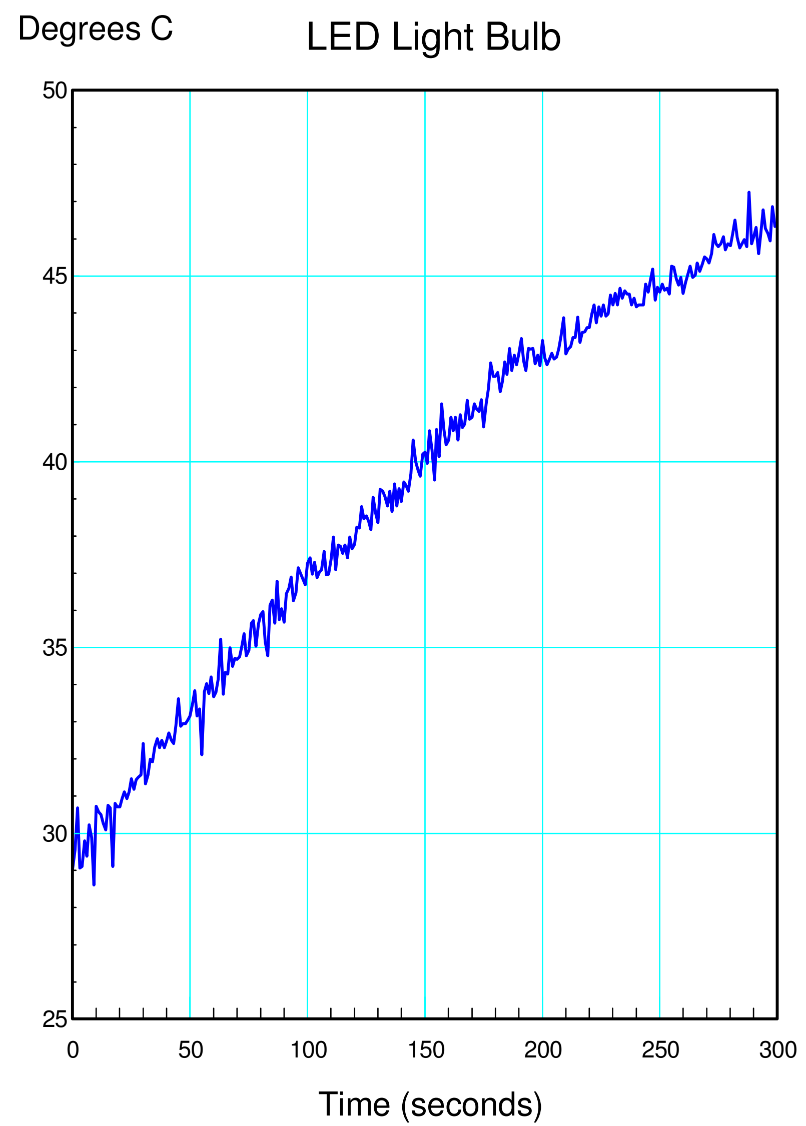
- 1st-order differential equation
- heat equation

There's considerable noise in the data

- Long wires act as antennas

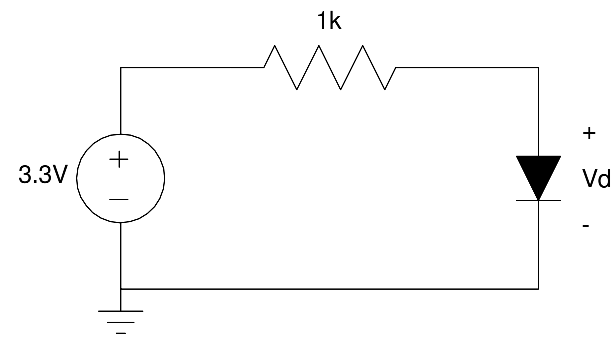
Noise could be reduced

- Use twisted pair wires
- Use shielded twisted pair wires
- Keep leads short.



Temperature Sensor: TMP36

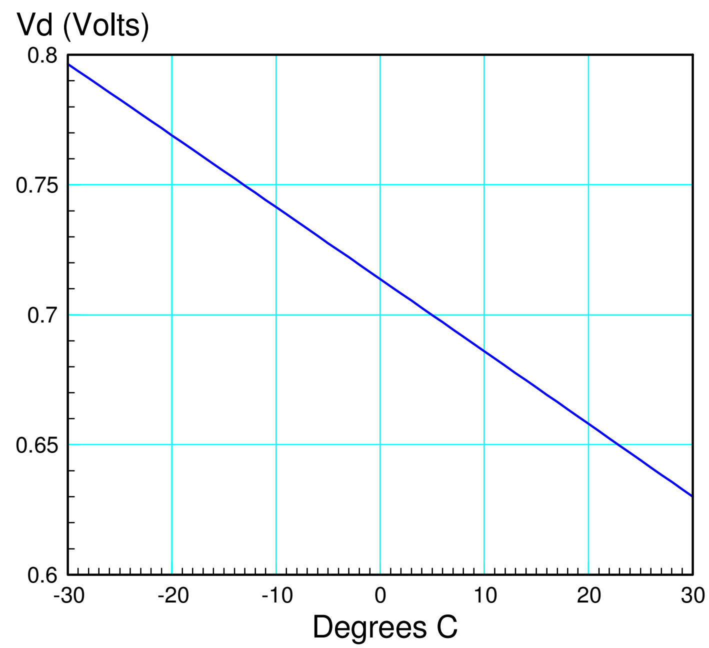
Another way to measure temperature is to measure the voltage drop across a diode.



From Electronics, the voltage drop across a diode is a function of temperature

- ECE 320 lecture #5

$$V_d = V_T \cdot \ln\left(\frac{N_A N_D}{n_i^2}\right)$$



The voltage drop vs. temperature is almost linear

Add some circuitry to get the voltage to go up +10mV / degree C

- TMP36

TMP36 (cont'd)

From the data sheets for a TMP36 (www.Digikey.com),

- Operating voltage: 2.7V to 5.5V
  - i.e. 3.3V operation works
- The output at 25C is 750mV
- The sensitivity is +10mV / degree C
- Linearity is within 0.5C over a range of -40C to +125C

TMP36GT9Z

Image shown is a representation only. Exact specifications should be obtained from the product data sheet.

DigiKey Part Number 505-TMP36GT9Z-ND

Manufacturer Analog Devices Inc.

The output voltage should be:

-40C	25C	+125C
100mV	750mV	1.750V

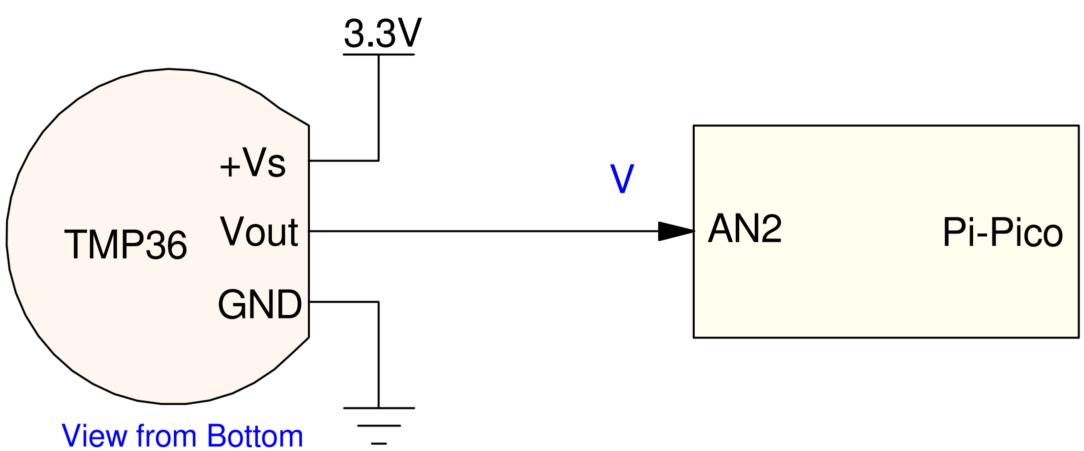
Based upon the voltage, you can then compute the temperature:

$$T = 100V - 50$$

To interface with a Pi-Pico, you can connect it directly to the A/D input. This gives a resolution of 0.08 degrees C:

A/D resolution = 805.9uV

$$\left(\frac{805.9\mu V}{10mV/C}\right) = 0.08059C \qquad \textit{resolution in degrees C}$$



Interface for a TMP36 temperature sensor to a Pi-Pico

TMP36 Code:

Recording temperature is about the same as before

- Only change is how you compute temperatgure

```
while(time < 300):
    while(flag == 0):
        pass
    flag = 0

    V = kV * a2d2.read_u16()
    Temp = 100.0*V - 50

    print(time, V, Temp)

    file1.write(str(time) + " ")
    file1.write(str(Temp) + " ")
    file1.write("\n")

    time += T
```

Temperature of an Incandescent Light Bulb

Just for fun, let's measure the temeperature of an incandescent light bulb

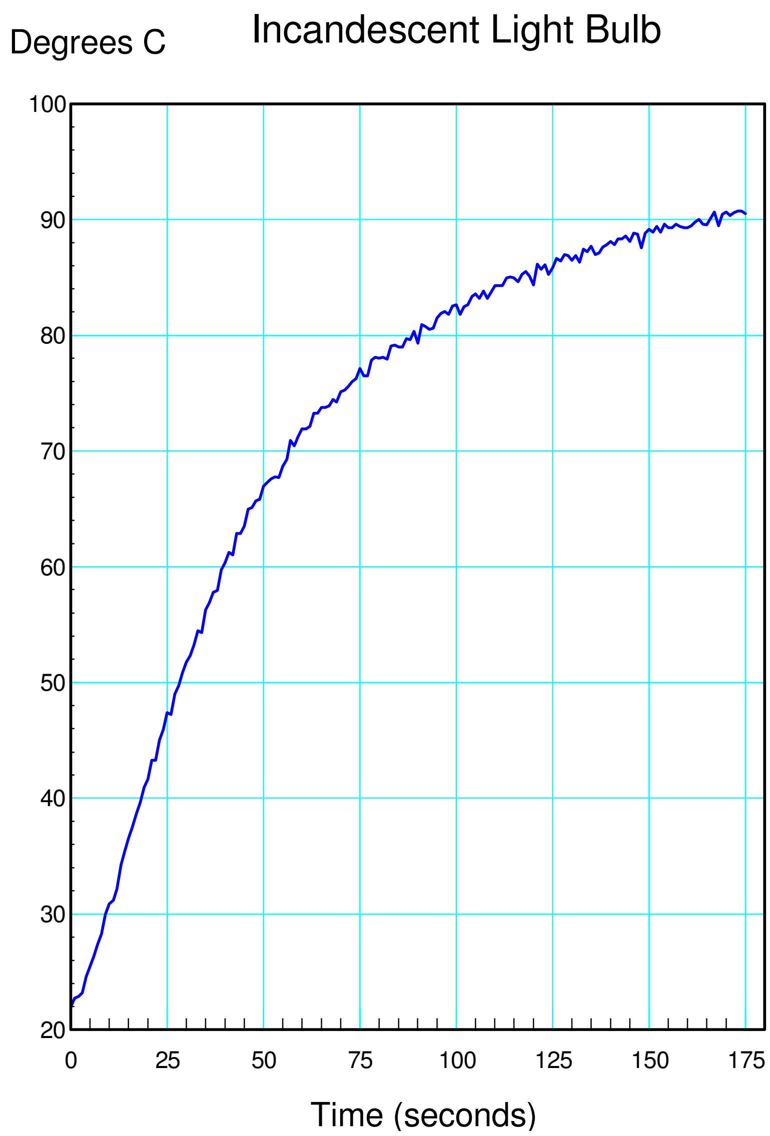
- 60W light
- 2% efficient

Again, temperature rise is a decaying exponential

- 1st-order differential equation
- heat equation

There is still considerable noise

- Long wires
- No shielding
- Not using twisted pair wires



DS18B20 Temperature Sensor

Temperature sensor with a digital interface

- Easier to get readings
- Less noise (shorter leads to uP)
- Requires drivers

Specifications:

- -55C to +125C
- 2.5V to 5.5V operation
- With a resolution of 0.0625C (12-bit)

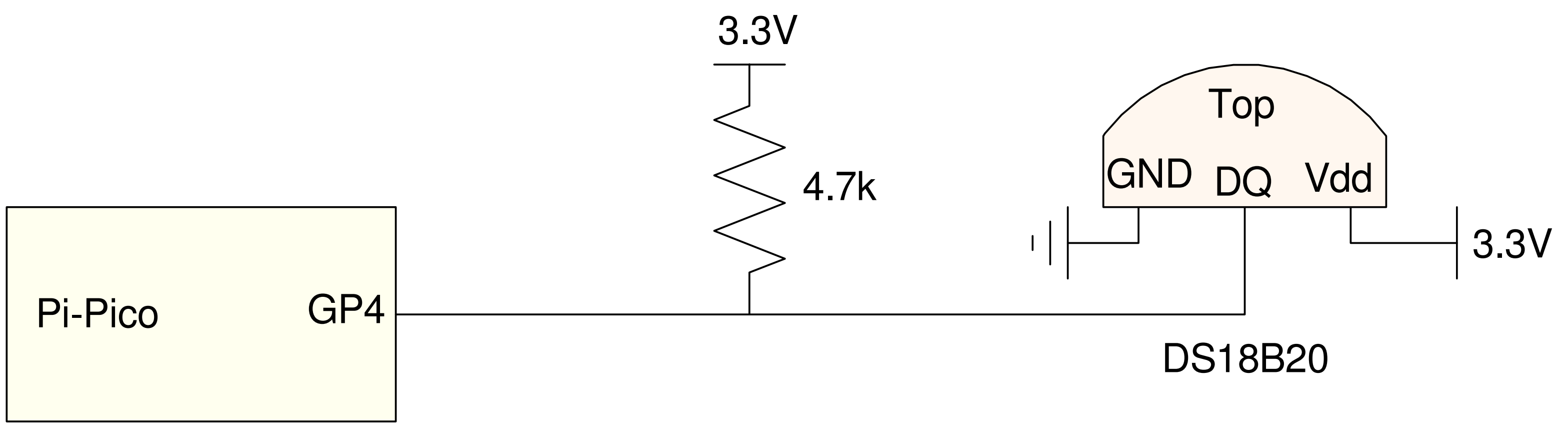
Each DS18B20 has a unique 64-bit serial code.

- Can have several on a bus
- Can read each one separately

DS18B20 Hardware

Add power, ground, and a signal from the Pi-Pico

Include a pull-up resistor on the data line



DS18B20 Software

Included in MicroPython

- onewire
- ds18x20

Connect the sensor to GP4

- any I/O pin is OK

Scan for sensors

- multiple sensors OK

Read each sensor

- 750ms delay for 12-bit read
- Reading is for previous conversion

```
import onewire, ds18x20
from machine import Pin
from time import sleep, sleep_ms

ds_pin = Pin(4)
ds_sensor = 
ds18x20.DS18X20(onewire.OneWire(ds_pin))

roms = ds_sensor.scan()
print('Found DS devices: ', roms)

while(1):
     ds_sensor.convert_temp()
     sleep_ms(750)
     Temp = ds_sensor.read_temp(roms[0])
     print(Temp)
```

Setting the sampling rate to 1.00 second

- Use a timer interrupt
- Readings are for previous conversion
  - 750ms delay

Note:

- You can remove the 750ms delay if you swap the order of read and convert
- Result is the temperature the previous sample (1 second ago)
- Frees up time to do other stuff

```
:
def tick(timer):
    global flag
    flag = 1

Time = Timer()
Time.init(freq=1/T, mode=Timer.PERIODIC,
callback=tick)

sec = 0

while(sec < 1800):
    while(flag == 0):
        pass
    flag = 0
    sec += T

    ds_sensor.convert_temp()
    sleep_ms(750)
    Temp = ds_sensor.read_temp(roms[0])

    print(sec, Temp)
```

Temperature of a Hot Cup of Water

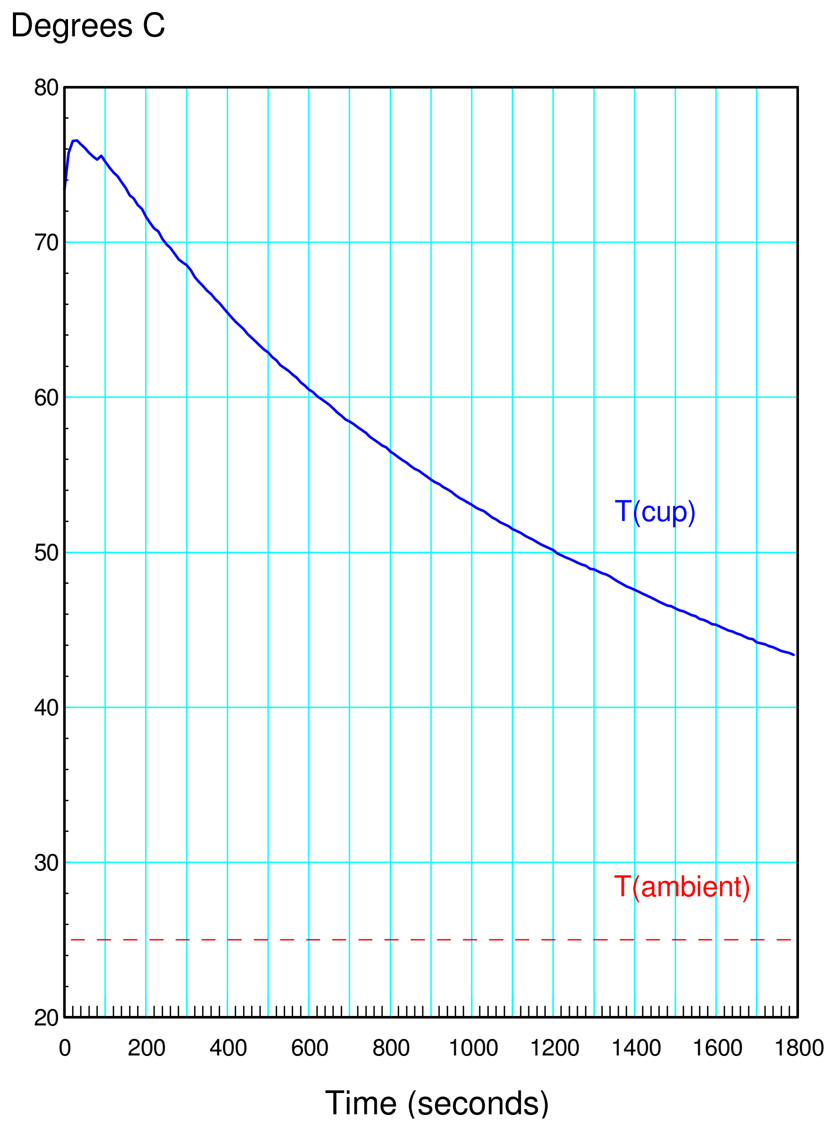
Measure the temperature of a cup of hot water

- Record temepratureevery second
- Using a DS18B20 sensor

Note:

- The recorded temperature is very clean (very little noise),
  - Short leads from sensor to uP
  - Once data is digital, noise has little impact
  - One reason sensors are going digital
- After 100 seconds, the temperature decays as

$$T = b^* \cdot \exp(at) + T_{amb}$$



Least Squares

Given the data, find the thermal time constant

- TC = -1/a seconds
- Tells you how good your coffee cup is

The thermal time constant can be found by modeling the temperature as

$$T = b^* \cdot e^{at} + T_{amb}$$

or equivalently

$$T - T_{amb} = \exp(at + b)$$

where {a, b} are constants.

Taking the log of both sides

- An equation which is linear in t:

$$\ln(T - T_{amb}) = at + b$$

This can be solved using least squares. Placing this in matrix form:

$$\begin{bmatrix} \ln(T_0 - T_{amb}) \\ \ln(T_1 - T_{amb}) \\ \ln(T_2 - T_{amb}) \\ \vdots \end{bmatrix} = \begin{bmatrix} t_0 & 1 \\ t_1 & 1 \\ t_2 & 1 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

or

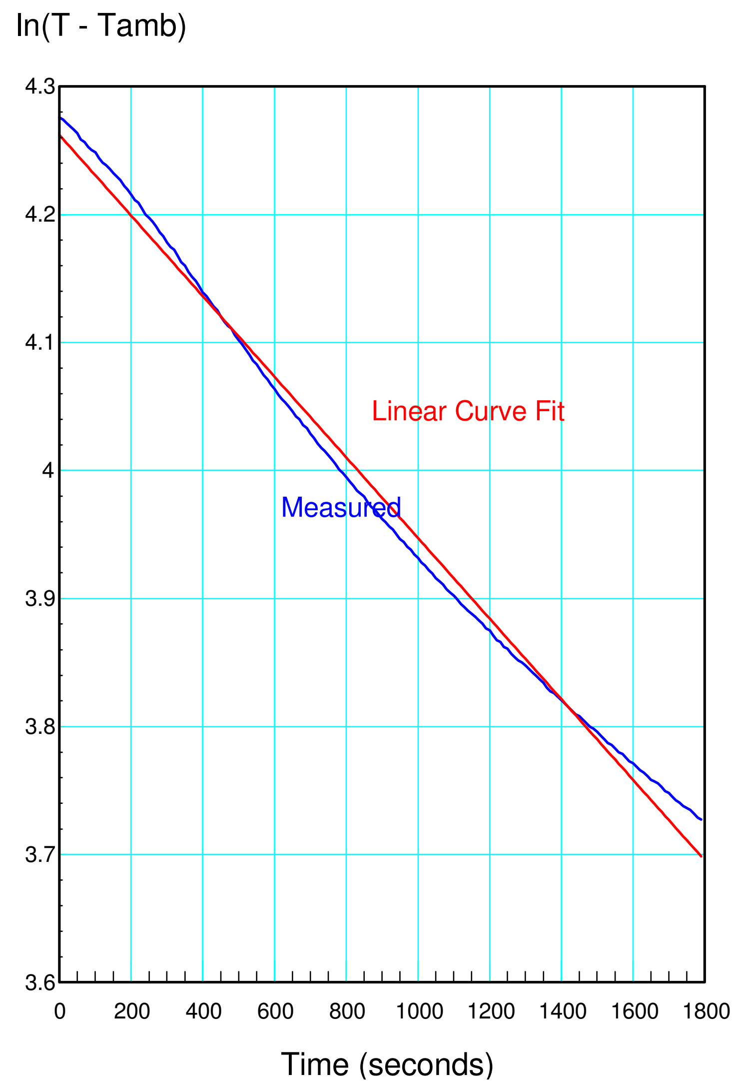
$$Y = BA$$

The least squares solution for {a, b} is

$$A = (B^T B)^{-1} B^T Y$$

In Matlab:

```
>> Data = [  <paste data from previous code >];
>> t = Data(:,1);
>> T = Data(:,2);
>> Tamb = 24.616;
>> B = [t, t.^0];
>> A = inv(B'*B)*B'*log(T - Tamb)

 -3.1563e-004
  4.2630e+000

>> a = -1/A(1)

a =  3.1682e+003

>> b = A(2)

b =  4.2630e+000

>> plot(t,T,t,exp(-t/a+b) + Tamb);
```



The thermal time constant is 3168.2 seconds for this coffee cup.

Note: You could bypass Matlab and do all of the calculations using Python - but two problems arise:

- With this method, there are a *lot* of computations to do every sample, and
- The initial measurements had errors

To solve these problems,

- Recursive least-squares will be used to simplify calculations (next section), and
- A forgetting factor will be added to weight more recent data more heavily (the following section)

Recursive Least Squares

Assume you are trying to find a linear curve fit

$$y = ax + b$$

Place your data in matrix form (assume four data points for now):

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_0 & 1 \\ x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

The least squares solution is:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \left(\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 & 1 \\ x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

or

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \Sigma x_i^2 & \Sigma x_i \\ \Sigma x_i & n \end{bmatrix}^{-1} \begin{bmatrix} \Sigma x_i y_i \\ \Sigma y_i \end{bmatrix}$$

In Python, you only need to keep track of four terms to create B and Y

$$B = \begin{bmatrix} \Sigma x_i^2 & \Sigma x_i \\ \Sigma x_i & n \end{bmatrix}$$

$$Y = \begin{bmatrix} \Sigma x_i y_i \\ \Sigma y_i \end{bmatrix}$$

or equivalently, in a recursive manner:

$$B_i = B_{i-1} + \begin{bmatrix} x_i^2 & x_i \\ x_i & 1 \end{bmatrix}$$

$$Y_i = Y_{i-1} + \begin{bmatrix} x_i y_i \\ y_i \end{bmatrix}$$

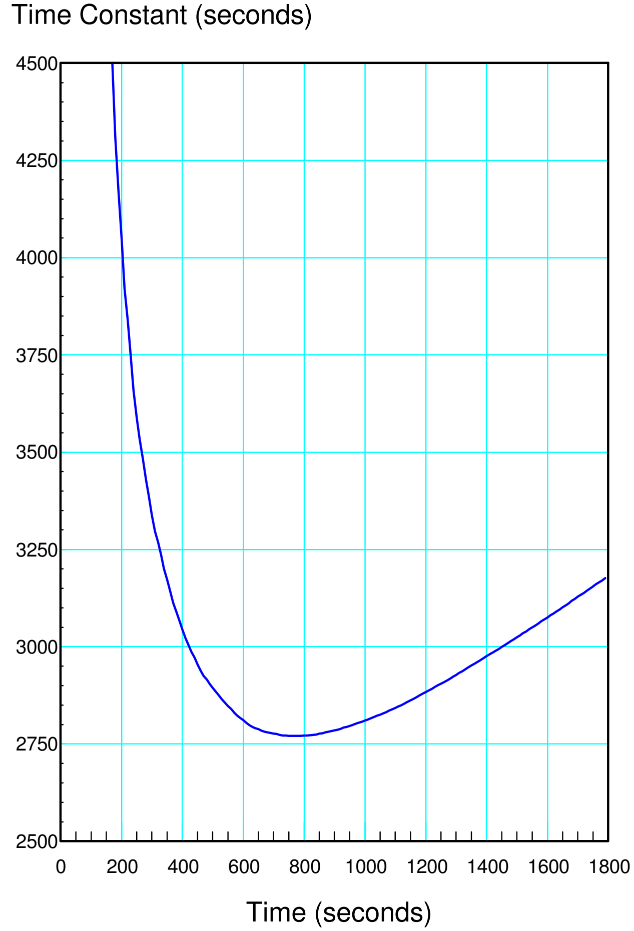
Recursive Least Squares in Python

- Keep updating B and Y
- From these, you can compute A = {a, b}

```
x = 0
Tamb = 19.38
B = [[0.01,0],[0,0.01]]
Y = [[0],[0]]
while(1):
    while(flag == 0):
        pass
    flag = 0

    ds_sensor.convert_temp()
    sleep_ms(750)
    Temp = ds_sensor.read_temp(roms[0])

    x += 1
    y = log(Temp - Tamb)

    matrix.add(B,  [[x*x, x], [x, 1]])
    matrix.add(Y, [[x*y], [y]])
    Bi = matrix.inv(B)
    A = matrix.mult(Bi, Y)
    a = A[0][0]
    b = A[1][0]
    print(x, a, b)
```

Note the following:

- The slope varies
  - The system is nonlinear
  - Evaporation adds additional heating at the start
  - Adding a lid would reduce this effect
- As time goes on, you start to ignore the most recent data
  - Treated the same as any other data point



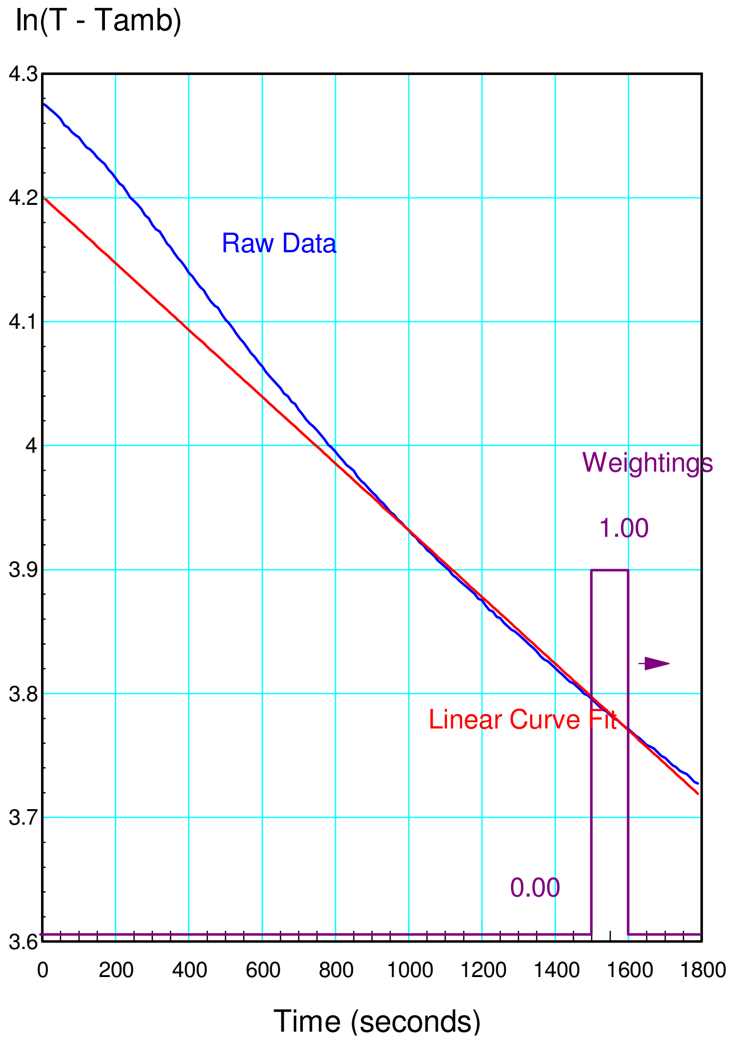
Recursive Least Squares with a Moving Window

If the system is changing, you may want to use only recent data

- Ignore really old data

Using a moving window allows this

- Do a least squares curve fit using only the last N data points
- Requires you to keep track of the last N data points
- Requires more computations



In code

- Create a buffer which saves the last 100 data points
- Recompute B and Y using these last 100 data points each sample,
- From B and Y, recompute the least-squares curve fit each sample.

A circular stack saves time (so you don't have to constantly push data onto a stack each sample). This gives a more efficient way to compute B and A

$$B_i = B_{i-1} + \begin{bmatrix} x_i^2 & x_i \\ x_i & 1 \end{bmatrix} - \begin{bmatrix} x_{i-100}^2 & x_{i-100} \\ x_{i-100} & 1 \end{bmatrix}$$

$$Y_i = Y_{i-1} + \begin{bmatrix} x_i y_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_{i-100} y_{i-100} \\ y_{i-100} \end{bmatrix}$$

Python Code

- Keep the last 100 data points in a circular stack
- Update B and Y each time point
  - Remove data point (i-100)
  - Add data point (i)

```
while(1):
    while(flag == 0):
        pass
    flag = 0
    ptr = (ptr + 1) % 100
    time += 1

    ds_sensor.convert_temp()
    sleep_ms(750)
    Temp = ds_sensor.read_temp(roms[0])

    x = X[ptr]
    Y = Y[ptr]

    matrix.subtract(B,  [[x*x, x], [x, 1]])
    matrix.subtract(Y, [[x*y], [y]])

    X[ptr] = x = time
    Y[ptr] = y = log(Temp - Tamb)

    matrix.add(B,  [[x*x, x], [x, 1]])
    matrix.add(Y, [[x*y], [y]])

    if(time >= 100):
        Bi = matrix.inv(B)
        A = matrix.mult(Bi, Y)
```

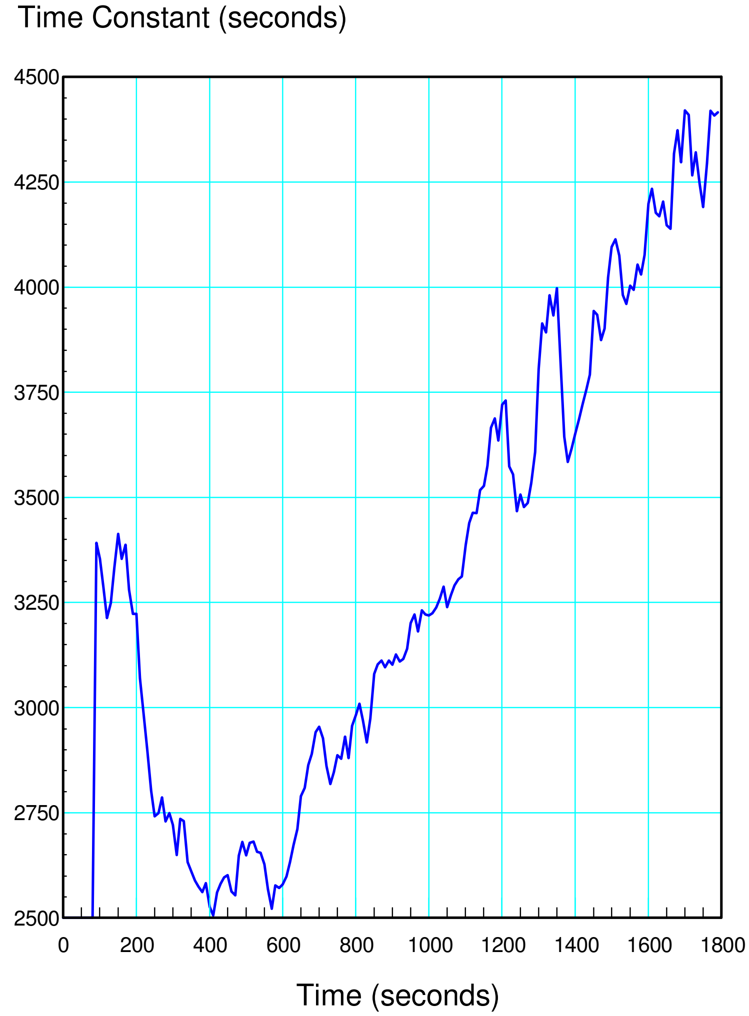
Time Contant with a Moving Window

By using a moving window

- The slope is based upon the last 100 data points
- Making the results more responsive to changes in the system,

But

- There is more noise
- (less data is being used)



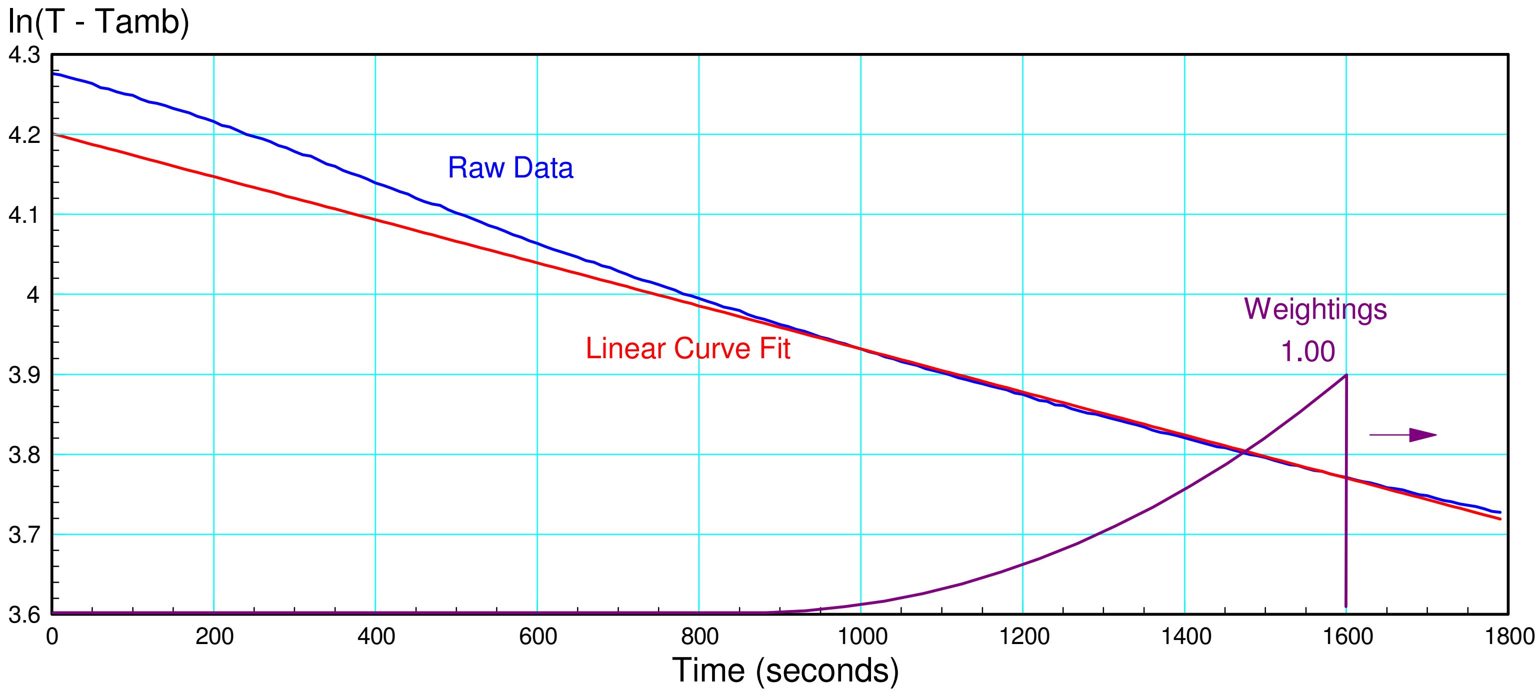
Recursive Least Squares with a Forgetting Factor

A similar scheme uses an exponential weighting factor:

$$Weight(k) = \alpha^k \qquad\qquad 0 < \alpha < 1$$

where

- k is how many samples in the past the data was collected and
- α is a forgetting factor



This actually simplifies the code

- You no longer need a buffer storing all of the old data
- All you need is the resulting B and Y matrices:

The long way to compute B and Y are:

$$B_k = \sum_{n=0}^{k} \alpha^{k-n} \begin{bmatrix} x_n^2 & x_n \\ x_n & 1 \end{bmatrix}$$

$$Y_k = \sum_{n=0}^{k} \alpha^{k-n} \begin{bmatrix} x_n y_n \\ y_n \end{bmatrix}$$

A shorter, recursive way to compute B and Y are

$$B_k = \alpha B_{k-1} + \begin{bmatrix} x_k^2 & x_k \\ x_k & 1 \end{bmatrix}$$

$$Y_k = \alpha Y_{k-1} + \begin{bmatrix} x_k y_k \\ y_k \end{bmatrix}$$

The constants are then

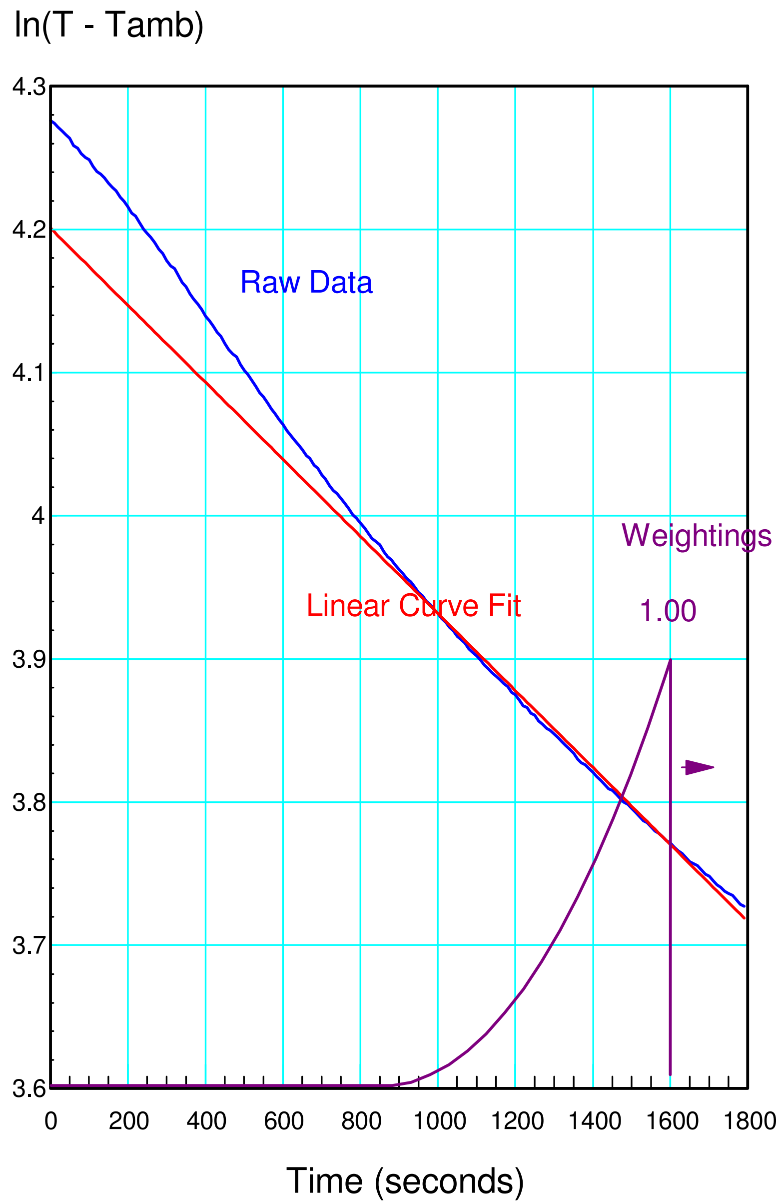
$$A = \begin{bmatrix} a \\ b \end{bmatrix} = B^{-1} Y$$

For example, let

- $\alpha = 0.995$
  - Old data is discarded by 0.5% per sample

Old datas is ignored:

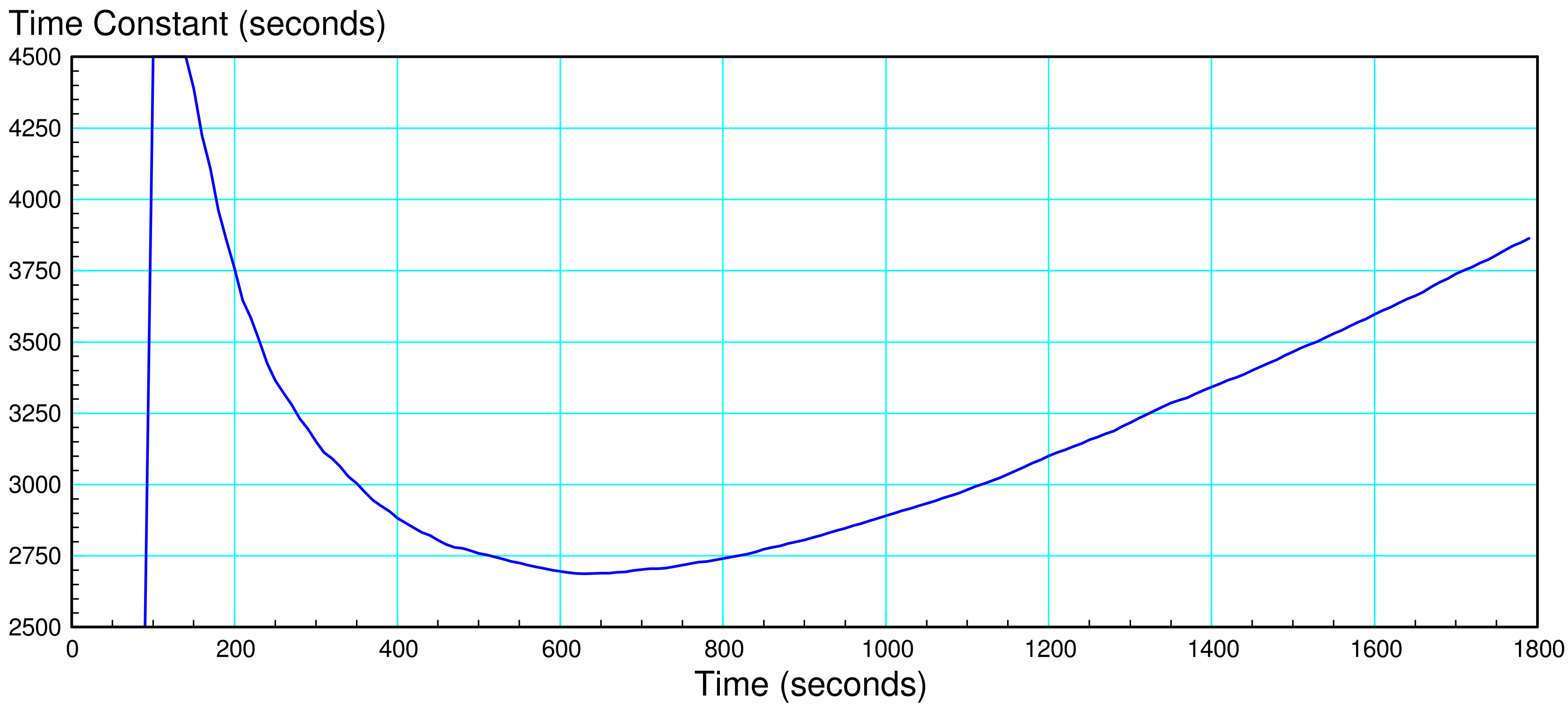
- w = 1.000 k = 0
- w = 0.61 k = 100
- w = 0.37 k = 200
- w = 0.22 k = 300
- w = 0.13 k = 400
- etc



Thermal Time Constant

By weighting more recient data more heavily, changing time constants can be captured

- With less noise due to using more data



Summary:

Temperature is pretty easy to measure.

- Thermistors and thermal diodes output an analog signal, which can be read by the 12-bit A/D on the Pi-Pico. This along with some computations allow you to determine the temperature.
- Digital temperature sensors allow you to read temperature without the need of an A/D conversion. These also have the advantage of returning temperature rather than a raw signal which needs to be converted to temperature.

Once you can measure temperature, you can do all sorts of things, like measure the thermal time constant of a coffee cup as well as other things.

References

Pi-Pico and MicroPython

- https://github.com/geeekpi/pico_breakboard_kit
- https://micropython.org/download/RPI_PICO/
- https://learn.pimoroni.com/article/getting-started-with-pico
- https://www.w3schools.com/python/default.asp
- https://docs.micropython.org/en/latest/pyboard/tutorial/index.html
- https://docs.micropython.org/en/latest/library/index.html
- https://www.fredscave.com/02-about.html

Pi-Pico Breadboard Kit

- https://wiki.52pi.com/index.php?title=EP-0172

Other

- https://docs.sunfounder.com/projects/sensorkit-v2-pi/en/latest/
- https://electrocredible.com/raspberry-pi-pico-external-interrupts-button-micropython/
- https://peppe8o.com/adding-external-modules-to-micropython-with-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-esp32-esp8266-micropython/