Heat Capacity in Markert's Group.

(Physics, UT at Austin)

Keeseong Park

Aug. 2007

This short instruction shows how to measure heat capacity by using RMC calorimeter with semi-adiabatic method. To get satisfying results,
use samples with heavier masses than around 50 mg and take data in the temperature range of 0.3 K to 40 K. See the references.

References :

Michelle Chabot's masters thesis, 1998

Calorimetery probe manual provided by RMC calorimeter company.

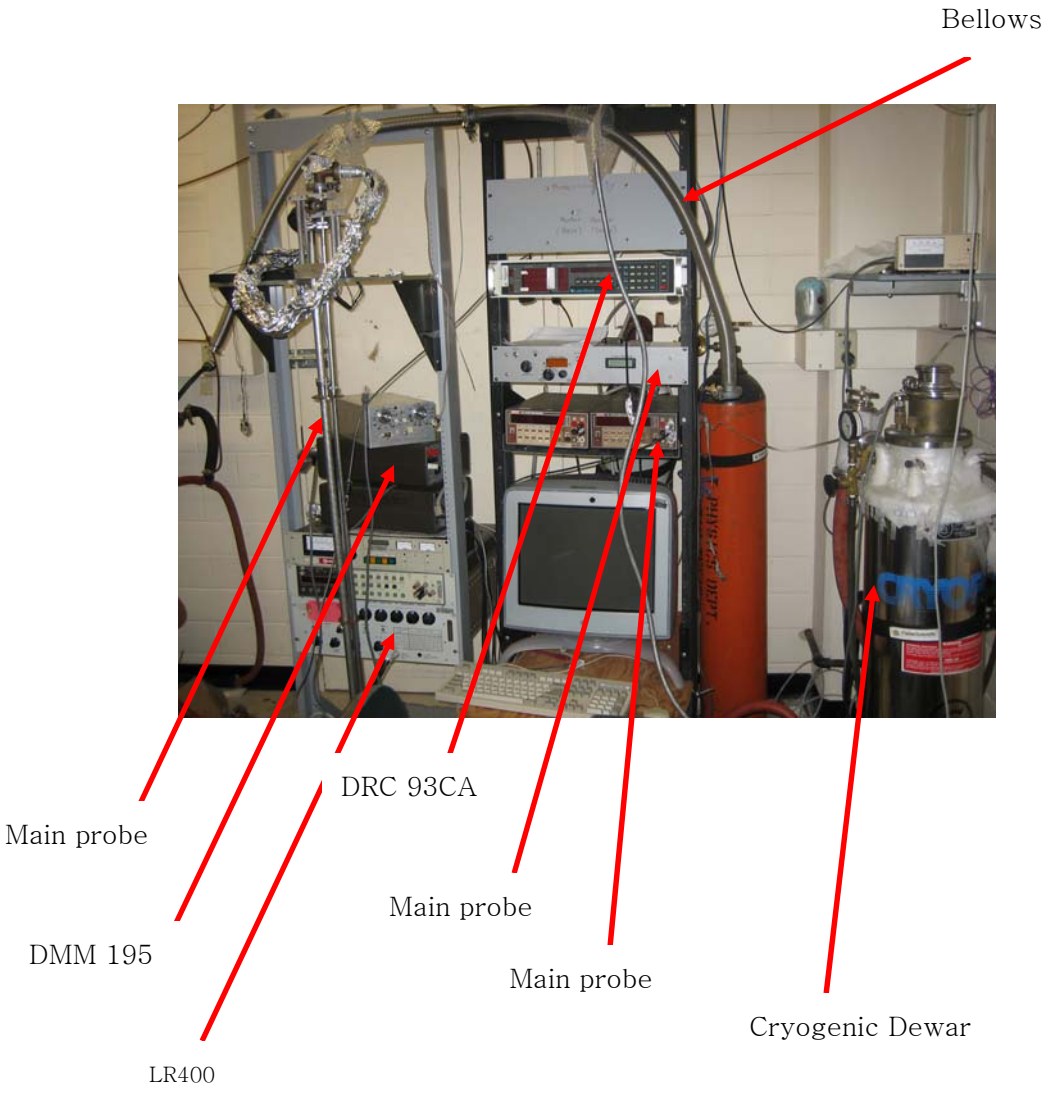
Keeseong Park's PhD dissertation, 2007

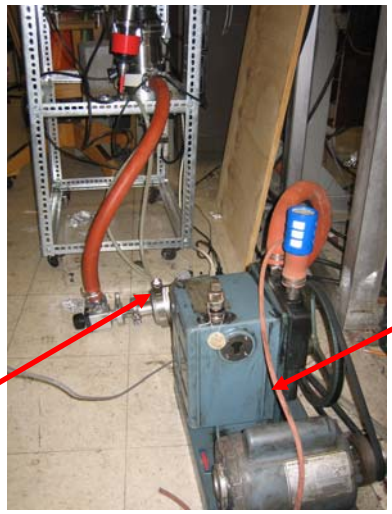
T.H.K Barron and G. K. White "Heat capacity and Thermal expansion at low temperatures", Kluwer Academic/Plenum Publishers (1999)

Outline

I. Instruments

Overall view

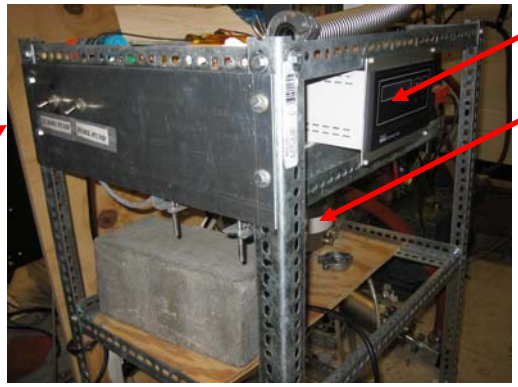




Mechanical pump

Oil trap

Vacuum guage



Turbo pump

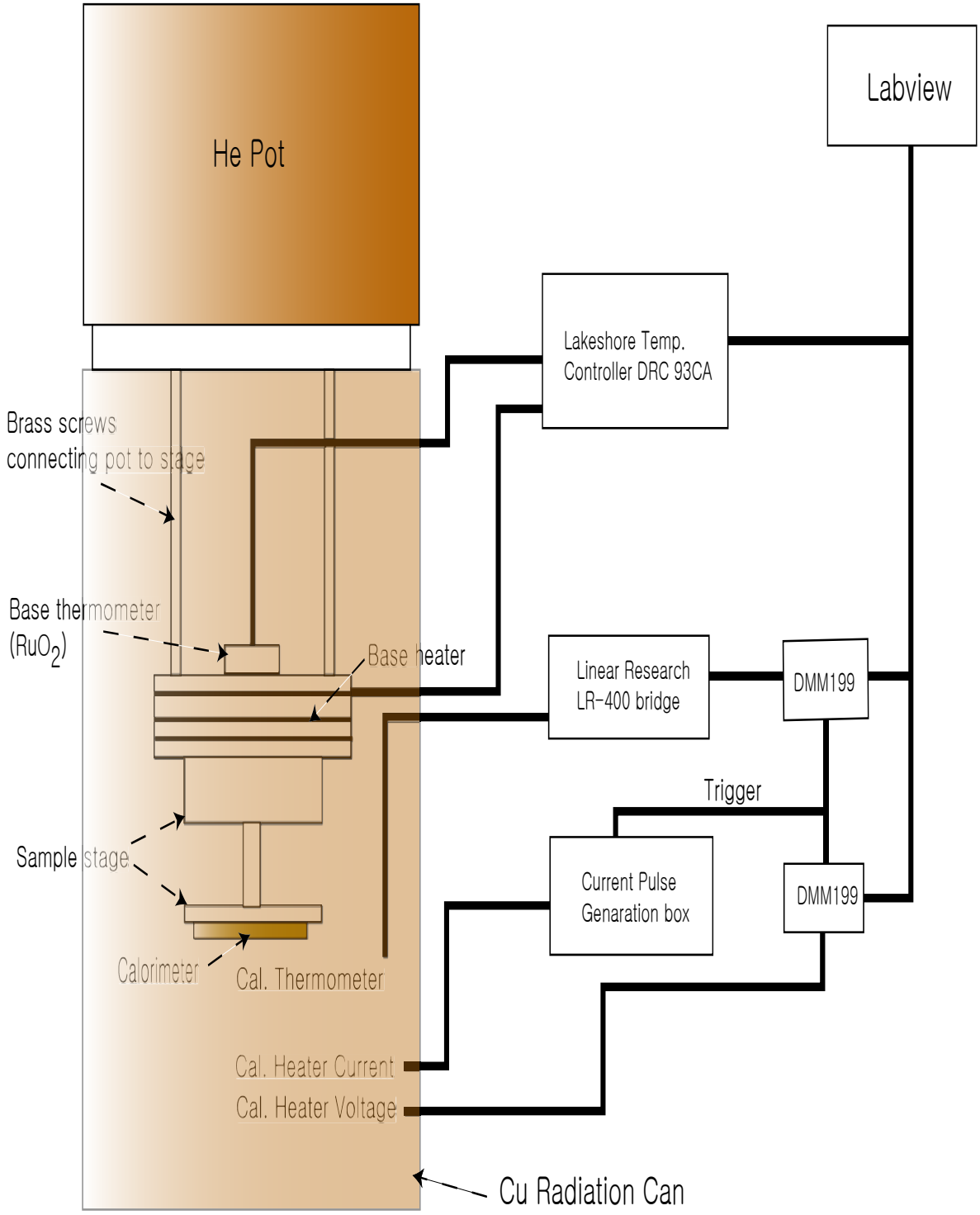
Control switch

Mechanical pump
for dewar
(purging)

Exchage gas

LHe input

LN2 input and outgas

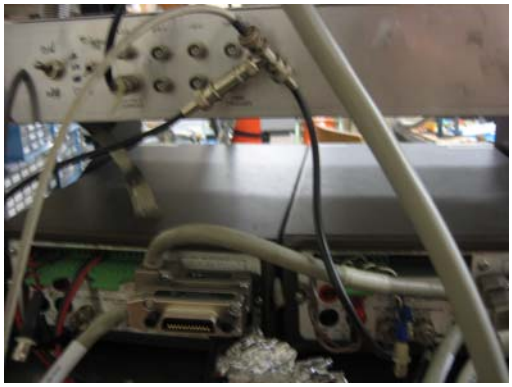

He Pot
Labview
Lakeshore Temp.
Controller DRC 93CA
Brass screws
connecting pot to stage
Base thermometer
(RuO_2)
Base heater
Linear Research
LR-400 bridge
DMM199
Trigger
Sample stage
Current Pulse
Genaration box
DMM199
Calorimeter
Cal. Thermometer
Cal. Heater Current
Cal. Heater Voltage
Cu Radiation Can

Current Pulse generation box

(front)

Time of pulse gives effective data but current reading doesn't. For current reading, a DMM 195 was used.

(back)



Trigger goes to two DMM 199's and one DMM 195.

DMM 195 for current measurement. A small box made to connect wires in series to measure current

Output from LR-400 resistance bridge is given in DC with 2V maximum.

For 200MΩ setting, 1.2V corresponds 120 MΩ

II. Set up (Connections)

24 pin connector

Line types : Copper ***Phosphor Bronze***

A	Cal th V_	**J**	Cal He I_	***S***	Pt therm I+	**Z**	Base He I-
B	Cal th V+	***K***	Pt therm I_	***T***	RuO2 th V+	**a**	extra
C	Cal th I_	***L***	RuO2 th I+	***U***	RuO2 th V-	**b**	extra
D	Cal th I+	***M***	RuO2 th I-	***V***	extra		
E	Cal He V+	***N***	extra	***W***	extra		
F	Cal He V_	***P***	extra	***X***	Pt therm V-		
H	Cal He I_	***R***	Pt therm V+	**Y**	Base He I+		

Pt thermometer (10 kΩ at room temp.) : (4 connectors)

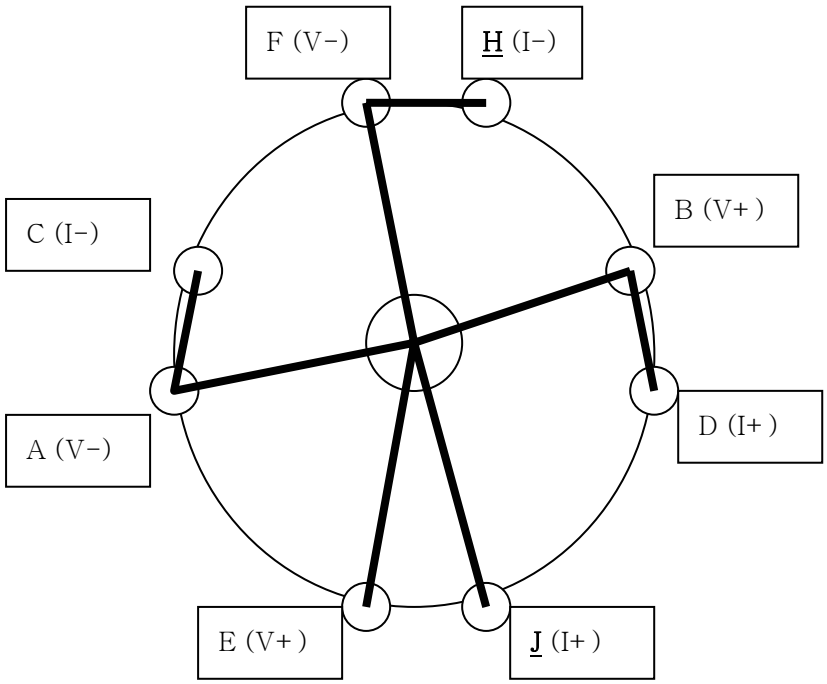
A	X (V-)	***B***	R (V+)	***D***	S (I+)	***E***	K (I-)

RuO2 thermometer (125 Ω at room temp.) : (4 connectors)

A	U (V-)	***B***	T (V+)	***D***	L (I+)	***E***	M (I-)

Base Heater (36.5 Ω at room temp.) : ***Y*** (R+), ***Z*** (R-)

RMC Calorimeter



Calorimeter thermometer : (4 connectors)

A	A (V-)	***B***	B (V+)	***D***	C (I+)	***E***	D (I-)

RuO2 thermometer : (4 connectors)

A	F (V-)	***B***	E (V+)	***D***	**J** (I+)	***E***	**H** (I-)

III. Data Taking (mostly following Michelle's note)

Before data taking

1. Cool the probe down to your lowest target temperature by using Michelle's thesis or Keeseong's dissertation. (Normally measuring with increasing temperature is easier than with decreasing because vacuum below 10^(-5) Torr is required to measure data. It means we can't use exchange gas which make longer time to cool down especially below 20K)

2. Set up electronics and cables as explained above (You can do it while cooling the probe to save your time.)

3. Open Labview file called C:\My Documents\experiments\heat capacity\vi\heat_capacity_R_T_cnverter_amp.vi which is shown on the next page.

4. Prepare the data table like below "Heat capacity data".

On data taking for **each** data point

1. Set current to desired range. 0.5 mA is a good starting point if you are not sure.

2. Fill out the blanks on the INITIAL SET UP on the Labview program.
(Changing file name is enough for the current setting. You don't need to worry about the CURRENT BOX CALIBRATION section because we measure current by using 195A multimeter.)

3. Press the "go" arrow on the Labview program. Record the "init base temp.s" on the data table.

4. Wait for the time display on the Labview program to be updated on the "Send Pulse indicator"

5. Right after the time is updated, send a current pulse. Do this by pushing the current switch up and holding it up as long as necessary. (For the 9 second trigger setting you will hold the switch up for ~9 seconds before the clock begins; after ~ 9 sec the clock will start and continue until you release the switch.) Record the time into Labview program and the data

table you made. (A time of pulse between ~ **0.4 to 0.8 sec** is good for most cases.)

6. Reset the clock with the reset button below the switch.

7. Wait until the Labview program stops. (On the OUTPUT section, you may see the calorimeter's resistance and converted temperature from resistance. Also you may see the voltage and the current of pulses you sent.)

8. Increment the file # on Labview if the output information is acceptable.

9. REPEAT

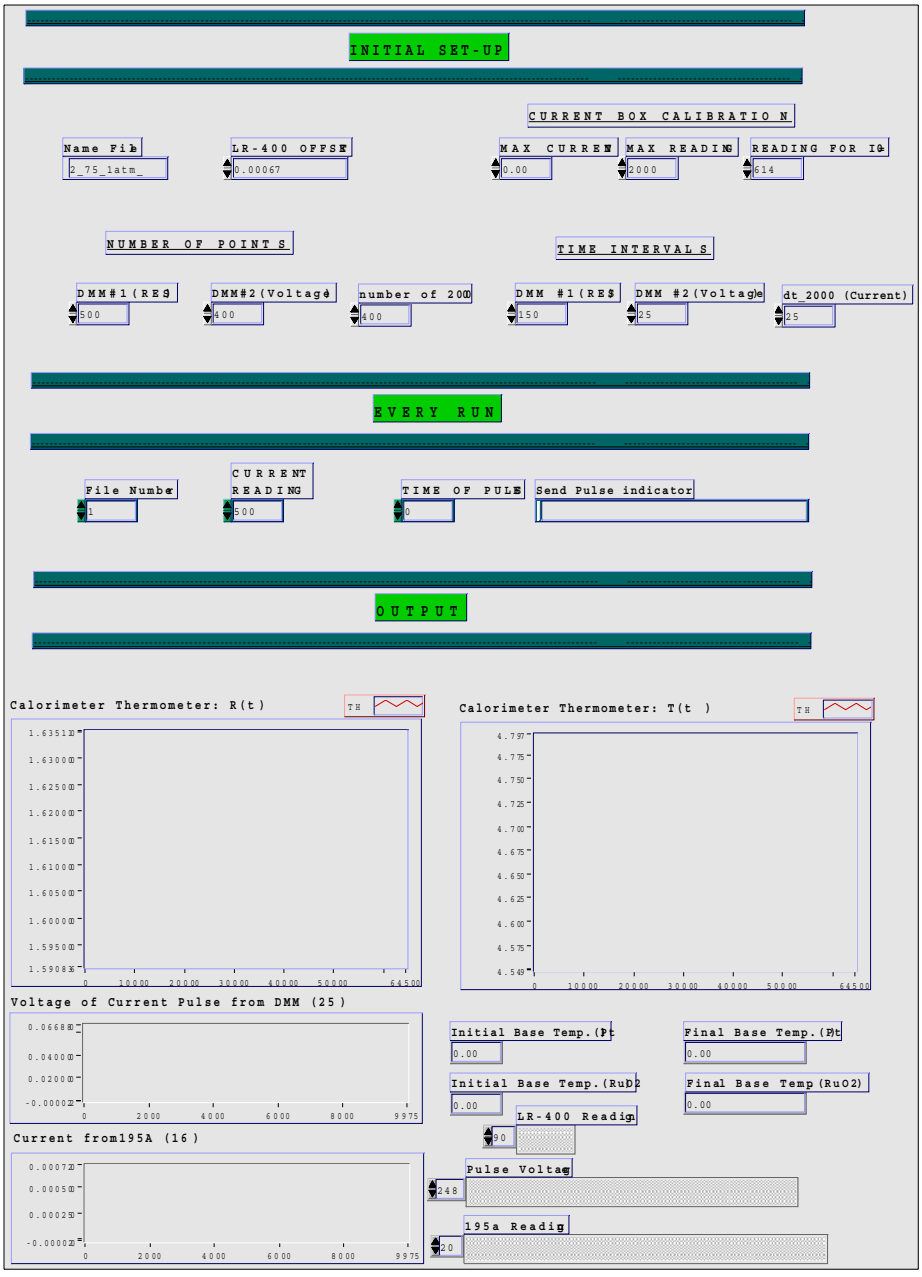
Note : You may adjust some parameters but it is not necessary for general cases.

You can change the trigger length in the circuit. If you do this, remember to adjust everything else accordingly, especially the time between measurements. This is done in Labview.

You can change the current. If your reading looks too noise, increase the current. If it looks like ΔT is too large, decrease the current.

heat capacity R T converter amp 195 new.vi
C:\My Documents\experiments\heat capacity\vi\heat capacity R T converter amp 195 new.v i
Last modified on 8/20/07 at 8:50 PM
Printed on 8/20/07 at 8:55 PM

Front Panel



Heat Capacity Data

Date: _____ . **File Name :** _____ .

File #	Temperaure			Time of Puls	Pulse Vol.	Pulse Cur	Del. Temp.	time cons.	Heat_cap_1	tempcal_1
	Target	RuO2	Cal.	(ms)	(V)	(mA)	(K)	(sec)	(J/K)	
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										
21										
22										
23										
24										
25										
26										
27										
28										
29										
30										
31										
32										
33										
34										
35										
36										
37										
38										
39										
40										

Ruthenium Oxide Film

Cryogenic Temperature Sensor

Model FS-12K-CA-PB

by TRI Research (612-645-7193)

D	A	B	C
T(K)	R(ohms)	V(logR)	
1	499.9	--	0
2	280	10005	4.00022
3	250	10007.5	4.00033
4	220	10018	4.00078
5	200	10029	4.00126
6	180	10051	4.00221
7	170	10059	4.00255
8	160	10066	4.00286
9	150	10077	4.00333
10	140	10089	4.00385
11	130	10105	4.00454
12	120	10125	4.0054
13	110	10147	4.00634
14	100	10180	4.00775
15	90	10210	4.00903
16	80	10248	4.01064
17	70	10309	4.01322
18	60	10374	4.01595
19	50	10467	4.01982
20	40	10592	4.02498
21	30	10813	4.03395
22	28	10866	4.03607
23	26	10936	4.03886
24	24	11016	4.04202
25	22	11109	4.04567

26	20	11219	4.04995
27	19	11282	4.05239
28	18	11352	4.05507
29	17	11427	4.05793
30	16	11513	4.06119
31	15	11607	4.06472
32	14	11721	4.06896
33	13	11877	4.07471
34	12	12036	4.08048
35	11	12140	4.08422
36	10	12337	4.09121
37	9.5	12452	4.09524
38	9	12584	4.09982
39	8.5	12774	4.10633
40	8	12955	4.11244
41	7.5	13172	4.11965
42	7	13381	4.12649
43	6.5	13625	4.13434
44	6	13911	4.14336
45	5.5	14246	4.15369
46	5	14642	4.1656
47	4.5	15129	4.17981
48	4	15733	4.19681
49	3.5	16505	4.21762
50	3	17528	4.24373
51	2.5	19040	4.27967
52	2	21290	4.32818
53	1.5	24899	4.39618
54	0		6.5536

IV. Data analysis with Igor

1. You may use any igor file in the data folder. Heat capacity.pxp, in the folder where this manual is located, is a good igor file to be used. (Open a file and save it as a different name before you start.)

2. Goto “Load Delimited Text” and open your data file. When screen comes up to open the file, click on the “overwrite existing waves” box. Click “Load” (Typical data from the Labview is shown below. Bold characters show the ones used in calculating heat capacity.)

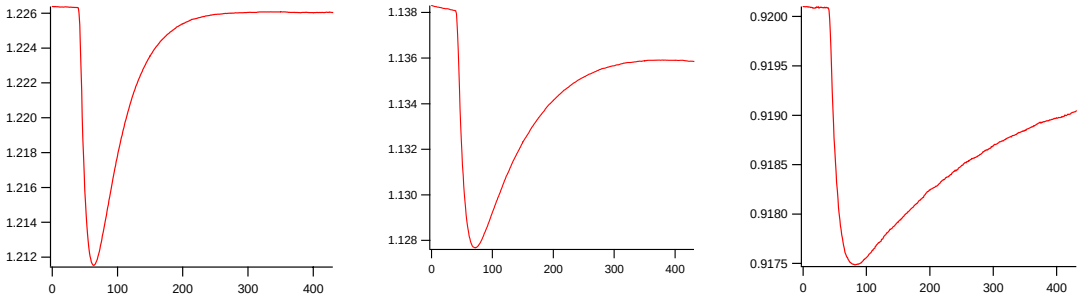
CALRES	**PLSVOLT**	**PLSCRT**	**BASETemp(RuO2)FIN**			BASETEMP(Pt)FIN		**BASETemp(RuO2)INIT**	
	BASETEMP(Pt)INIT		PLSCURRENT		**PLSTIME**	INT1	INT2	BRGOFFSET	
1.355836	-0.000016	0.000080	9.380000	9.380000	9.380000	9.380000	0.000000	720.000000	150
	25	0.000670							
1.355821	-0.000016	0.000040							
1.355821	-0.000002	0.000010							
1.355812	-0.000036	0.000020							
1.355813	-0.000022	0.000020							
1.355807	-0.000022	0.000010							
1.355805	-0.000009	0.000030							

3. Enter the # of moles in your sample, (this will be a faction), in the table in the column titled “No_moles”

4. Check the pulse time (PLSTIME) in the table with the value you recorded. Correct it if it is wrong. (It is often wrong since the computer used to save the data is slow, to say the least.)

5. You may draw the resistance graph using CALRES data. Go to the Window/newgraph. Select CALRES data as y wave and _calculated_ as x wave. (Some examples from CaYCuO data are shown below. First one is from low temperature, ~ 10 K, where specific heat is small because the temperature difference is large and the decaying time is short. The last one shows the data from high temperature ~ 35 K. Sometimes even the decay looks almost flat and linear, however don’t worry it is alright. The second graph shows a change of base temperature. It is also OK because the following command will subtract the

background. The important one is the change should be less than ΔT.)

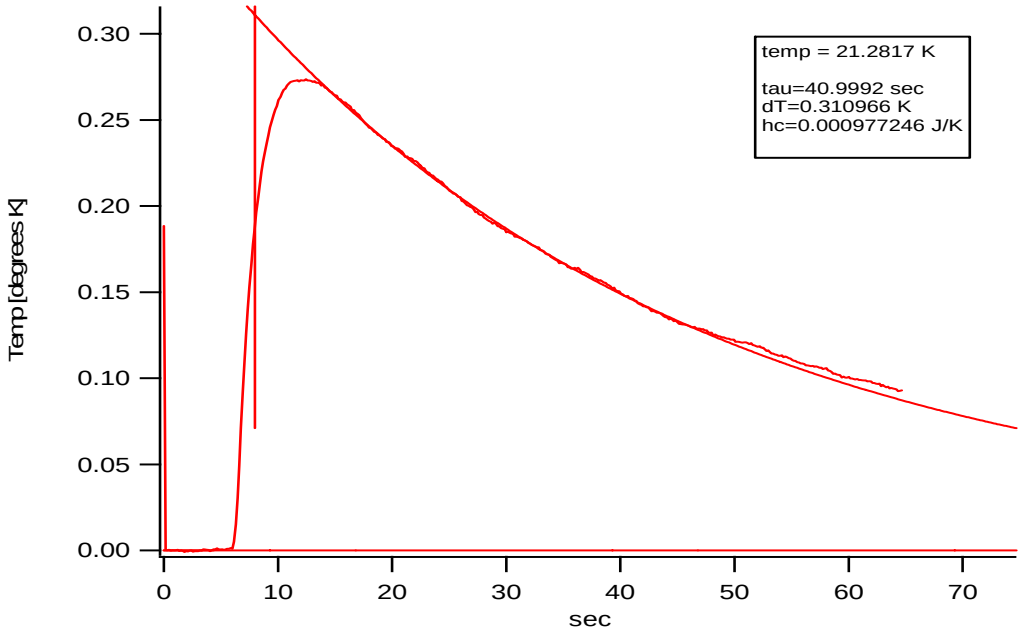


6. Click the resistance graph and type “line()” or “expon()” into the command box, depending on which one your data looks like.

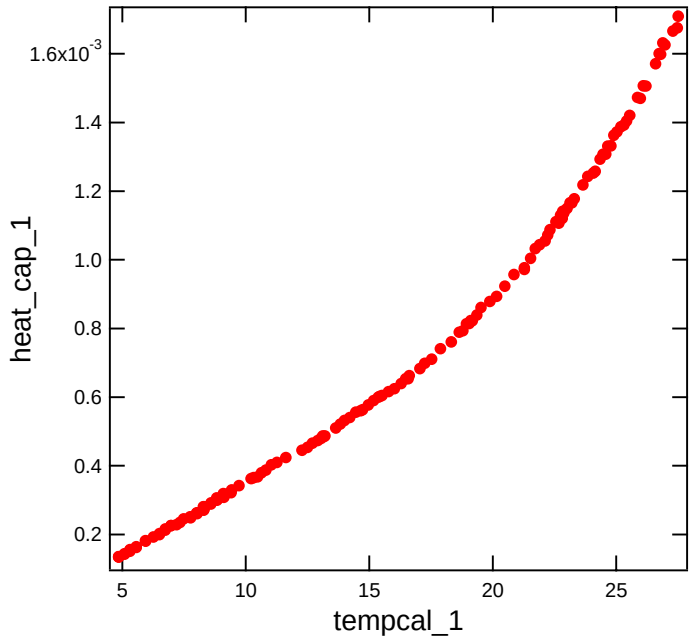
The command windows will print out some information for debugging as follows

A Pulse Voltage is: 0.213107
The input pulse current is: 0.002
The power input using P=iV is: 0.000426214
pow is: 0.000426214
beg is: 13.013
fin is: 45
Delta T is: 0.310966
Time of Pulse is: 0.713
heat capacity is: 0.000977246

Also, the graph you generated using CALRES data should change to a graph that looks like this :



7. Create a C vs T graph by using “heat_cap_1” as y wave and “tempcal_1” as x wave. After procedure 6, a point should have been added to this graph. This is the UNCALIBRATED temperature still. Do not worry if it looks off by even as much as 10 K



8. Repeat these procedure with each data file you have. Remember to click on ‘CALRES’ graph before entering the procedure name.

(If something screws up, then make table with “heat_cap_1” and “tempcal_1”, and delete the top row of the two data. Reload the data and continue)

9. Once all the data files have been processed

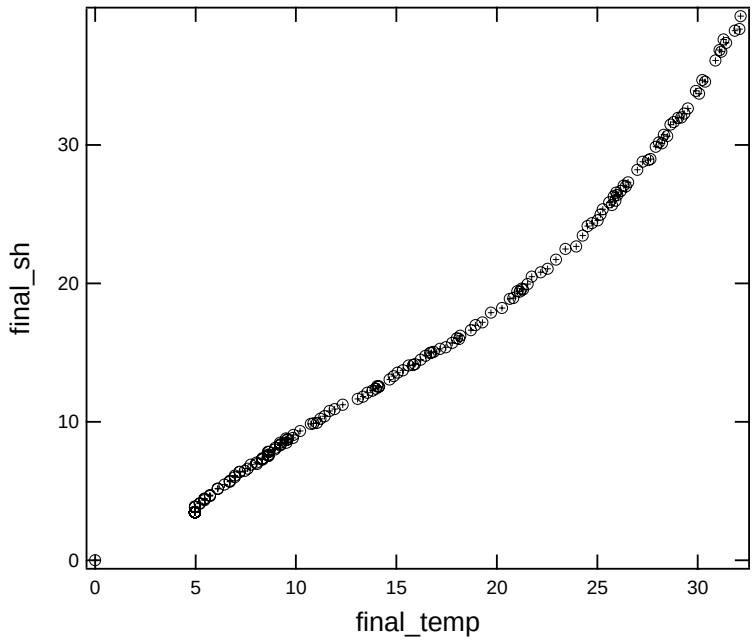
and you have a “heat_cap_1” vs “tempcal_1” curve that you like, for example, Ca3Y1 sample without oxygen anneal looks like the above curve.

then:

Type “addenda()” into the command line for subtracting addenda data from total.
Type ”tempadj()” into the command line for adjusting temperatures.

At this point, your final graph of “final_sh” vs “final_temp” for the same sample looks like the following figure. That is your final C vs T graph where C is in (J/mole K) and the temperature has been calibrated.

Some notes:
To plot C/T vs T2 : type “C_over_T()” in command line. A graph will appear.
If you want C in (J/gram K) or something else, this is easily adjusted pretending the wave named “No_moles” is the # of gram (or whatever) instead.



Procedures Windows in igor:

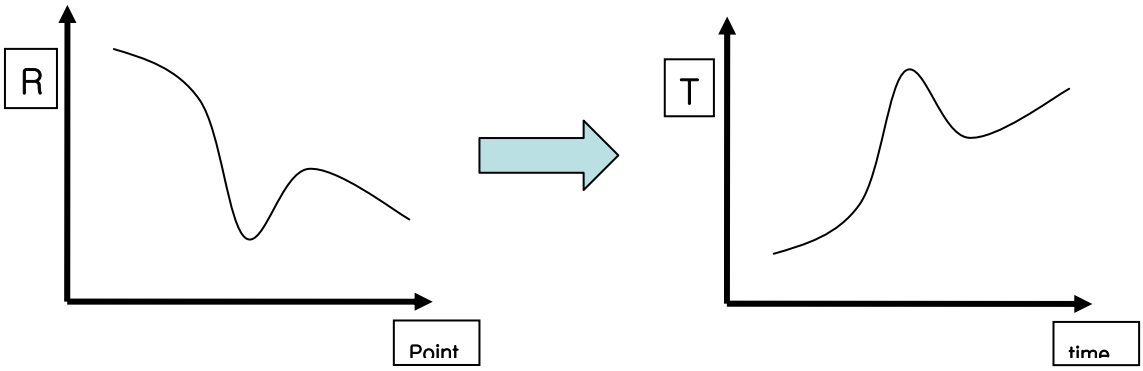
The procedure window is printed out on the following pages, with the 5 programs shown:

line()
expon()
addenda()
tempadj()
C_over_T()

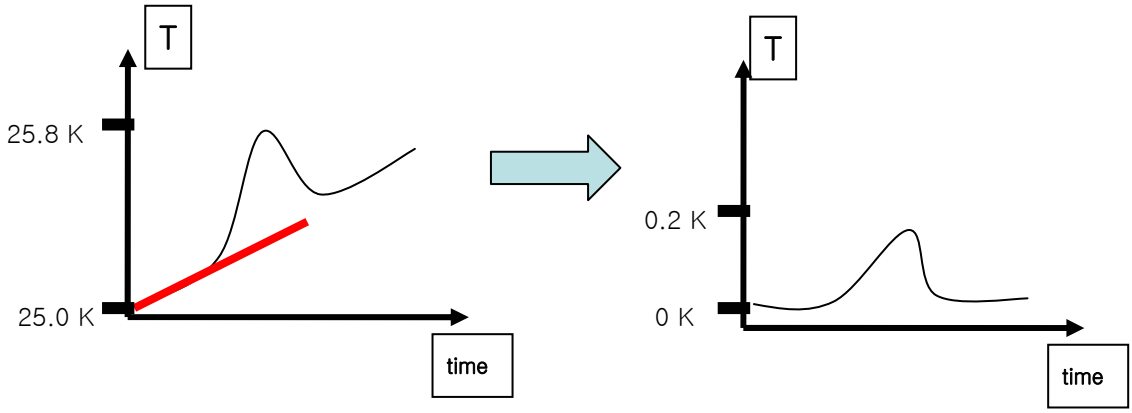
Here I'll briefly outline what they do in case changes are needed.

line() and expon()

1. First, this turns the resistant into 'temperature' (recall : this is the 'wrong' temperature)



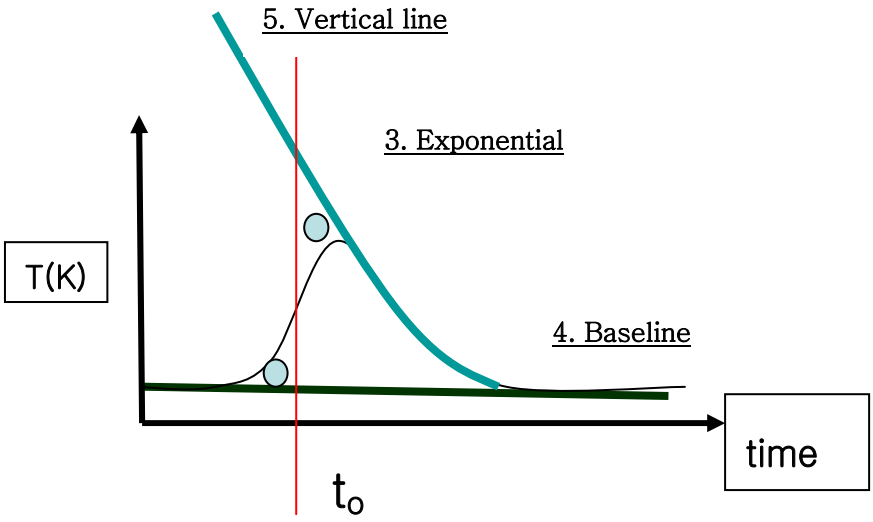
2. Then it fits the initial baseline, extrapoles this, and subtracts it off :



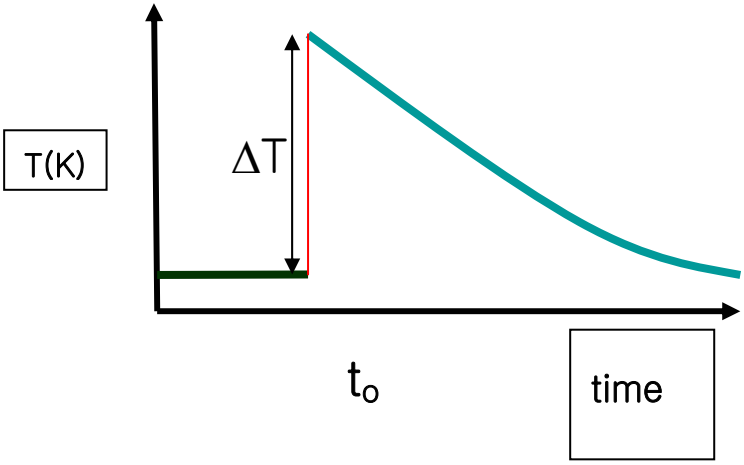
3. Then it fits either an exponential or a line to part of the T(t) curve after the pulse.

4.Then it fits a flat line to the baseline.

5.Then it finds the vertical line which makes the marked areas equal :



So that the system can accurately be estimated by an instantaneous pulse which heated it by ΔT :



Thus $$C = \frac{\Delta u}{\Delta T} = \frac{(\text{Powerof Pulse}) \cdot (\text{Timeof Pulse})}{\Delta T}$$

Then it finds the average temperature of the calorimeter by taking an average of the initial and final calorimeter temperature (Note : NOT base temperature since these are not always equal).

It adds these values for the C and T to the existing waves and puts a point on the C vs T graph.

addenda()

Subtracts addenda contribution, which has been found to be :

$$\left(\frac{T}{30}\right)^3 \cdot 0.00034 + 0.00014 \cdot \left(\frac{65}{T}\right)^2 \cdot \frac{\text{Exp}[65/T]}{(\text{Exp}[65/T]\text{-}1)^2}$$

This was a fit to experimental data taken up to 35 K (not absolute K, “wrong K”).

Then it is simply divided by # of moles and puts it in wave “final_sh”

tempadj()

Calibrate temp using :

$$T_{new} = 0.080563 + 0.97266 \cdot T_{old} + 0.006985 \cdot \left(T_{old}\right)^2$$

and puts in wave “final_temp

C_over_T()

Plots C/T vs T^2

Direct copy from igor procedure window

// NOTE: for the ~9 second pre-trigger, which is generally appropriate for all temps up to ~50K,
// and a constant interval btwn calorimeter measurements of 60ms.

//***

Function line()

```
wave wave0 = calres
variable o

o = BRGOFFSET[0]
wave0 = wave0*100000 + o*100000          // set CALRES to Ohms and correct for offset

variable a                               // define calibration constants
variable b                               // Note: this is still "incorrect" temp...
variable c                               //This ia corrected for in procedure: "tempadj"
variable d
variable e
variable f
variable g
variable h
variable j
a = 0.2700714
b =    0.07323249
c = 1.0219809
d = -1.9558035
e = 3.4889894
f = -4.4175955
g = 3.6976856
h = -1.7871035
j = 0.3821843

variable r                               // define calorimeter therm. resistance at room temp
r = 77408
```

```
        // calibration curve for calorimeter therm: changes R(t) to T(t)
wave0 = (a +b*(Sqrt(Sqrt(Ln(wave0/r)))) + c*(Ln(wave0/r)) +d*(Ln(wave0/r))^2 + e*(Ln(wave0/r))^3 +
f*(Ln(wave0/r))^4+g*(Ln(wave0/r))^5+h*(Ln(wave0/r))^6+j*(Ln(wave0/r))^7)^(-4)

variable int
int = int1[0]/1000      // defines interval bewteen CALRES data points in units of seconds

wave wave1 = plsvolt            // create variables for use
variable s
variable m
variable z
variable n
variable p
variable v
variable w
variable y
variable q
variable pow
variable dt

s = int2[0]/1000         // defines interval between PLSVOLT data points in units of seconds

m=        172       // For the ~9 sec baseline, the first measurement of the voltage
                            // is at point #176 -> #172

// The following section was made to use all the measurements taken of the pulse
// voltage and average them.    This proved to be unecesary since there is no deviation
// in the readings.    Therefore it is all commented out, and only the first measurement is
//used, in order to save time in analysis.

//-------------------------------------------------------------------
//z = m + plstime[0]/s
// calcualtes # of points from 0 to end of pulse in PLSVOLT data
// (for some reason this is ~2 over the actual # so that is corrected for)
```

```
//n = 0                                          // define iteration variable
//p = 0                        | define sum-of-voltage variable
//do                           | do loop runs for all n values between 0 and end of pulse
//          if (n>m+1)         |if loop runs for all n values above the start of pulse
//                    if (n<m+5)     |if loop calculates the data point # corresponding to
//                              v = n
//                              print(0)
//                              print(0)          | the start of the pulse & defines this as 'v'
//                              print v
//                    endif
//                    print wave1[n]
//                    p = wave1[n] + p
// |takes all data points from PLSVOLT during the pulse and
// | sums them up.    Total defined to be 'p'
//          endif
//          if (n > z-7)       | if loop calculates the data point # corresponding to
//                    w = n     | the end of the pulse & defines this as 'w'
//                    print w
//          endif
//          n = n+1            | n is incremented by +1 every cycle until end of pulse
//while(n < z-6)                                                    |end of loop
//y = (w-v) + 1                |calulates # of data points for plsvolt recorded during pulse
//q = p/y            | calculates the average voltage
//print q
//-------------------------------------------------------------------

variable v2                                    //this is just one voltage reading
v2 = wave1[250]
print v2

pow = v2*(plscrt[250])                  //calculates the power input using P=iV
print pow

SetScale/P x 0,int,"sec", wave0
//corrects x-axis to show correct time scaling in seconds
Label left "Temp [degrees K]"
```

```
SetAxis/A
//ModifyGraph rgb(plsvolt)=(1,4,52428)

variable ibt
ibt = basetemp_ruo2_init[0]
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
variable beg                    //Figures out a place to start the curvefit, and can be
beg = m*s +23+ plstime[0]/1000      // changed if desired. m=172, s=0.035
////////////////////////////// m=172, s=plstime/1000

CurveFit line wave0(0,m*s) /D                   //fits a line to the initial baseline
//ReorderTraces fit_calres,{calres,expfit}
//ModifyGraph rgb(fit_calres)=(2,39321,1)

wave wave15=linefit                             // creates a wave which is the baseline fit
SetScale/P x 0,int,"sec",wave15
wave15=W_coef[0] + W_coef[1]*x

wave0 = wave0 - linefit                         //subtracts the baseline from the T(t) curve
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
variable fin            //creates the place to stop the curvefit, and can also
fin = 350 * int                                                         // be changed.

CurveFit line wave0(beg,fin) /D
// fits the temperature decay part after the pulse to a line

variable tau
tau = -W_coef[0]/W_coef[1]

variable chisquare2
chisquare2 = V_chisq
wave wave17=expfit          //creates a wave out of the temperature decay fit.    yes, it
SetScale/P x 0,int,"sec",wave17                 //is called "expfit" even though it is a line.
wave17= W_coef[0] + W_coef[1]*(x)

CurveFit line wave0(0,m*s) /D                           //fits a line to the new baseline
```

```
//ReorderTraces fit_calres,{calres,expfit}
//ModifyGraph rgb(fit_calres)=(65535,0,0)
print m*s

wave wave16=flatlinefit                    // creates a wave out of this fit
SetScale/P x 0,int,"sec",wave16
wave16=W_coef[0] + W_coef[1]*x

//The following section matches the areas so that dT can be accurately estimated.
//---------------------------------------------------------------------
variable left
variable rt
variable ll
variable rr

wave wave20 = leftsum
wave wave21 = rtsum

Redimension/N=0 wave20
Redimension/N=0 wave21

ll = m*s
rr = beg

InsertPoints 0,1, wave20
wave20[0] = wave0(ll) - flatlinefit(ll)

InsertPoints 0,1, wave21
wave21[0] = expfit(rr) - wave0(rr)

left = leftsum[0]
rt = rtsum[0]


variable endpls
```

```
// This DO loop inreases ll by 0.01 and decreases rr by 0.01 until the position is found
// where the areas are equal on both sides.    this position is then called "endpls".

DO

        IF (left<rt)
                ll = ll+.01
                |print p
                InsertPoints 0,1, wave20
                wave0[0] = wave0(ll) - flatlinefit(ll)
                left = wave0[0] + left
        ELSE
                rr = rr-.01
                InsertPoints 0, 1, wave21
                rtsum[0] = expfit(rr) - wave0(rr)
                rt = rtsum[0] + rt
        ENDIF
        IF (ll > rr -.01)
                print rr
                print ll
                endpls = (ll+rr)/2
        ENDIF

WHILE (ll < rr    )

// This defines the delta T to be equal to the difference between the fit to the
// temp decay and the baseline at the position previaouls called "endpls"

variable deltemp
deltemp = expfit(endpls) - flatlinefit(endpls)

//print (00) //print deltemp
//printf "deltemp    is: %g\r", deltemp
//print endpls

wave wave30 = DT_1                    //puts this value of delta t into wave DT_1
```

```
InsertPoints 0,1,wave30
wave30[0]=deltemp

wave wave31 = tp_1                              // puts time of pulse into tp_1
InsertPoints 0,1,wave31
wave31[0]=plstime[0]/1000

wave wave11=vertline                    // makes it so a vertical line can be drawn
vertline = endpls               //at the point which makes areas equal.
AppendToGraph expfit vs vertline
//ModifyGraph rgb(flatlinefit)=(0,0,0)
AppendToGraph expfit
AppendToGraph flatlinefit

variable low                            //sets axis of graph
variable hi
low = W_coef[0] - .002
hi=deltemp+.005                                 // .05->.3
//SetAxis/A left
SetAxis left low, hi

v2 = wave1[250]
printf "A Pulse Voltage is: %g\r", v2

pow = v2*(PLSCRT[250])                  // calculates the power input using P=iV
printf" The input pulse current is: %g\r",PLSCRT[250]
printf "The power input using P=iV    is: %g\r", pow

variable hc1                            //Caluclulates Heat Capacity
hc1 = pow*plstime[0]/1000/deltemp
printf "pow    is: %g\r", pow


printf "beg    is: %g\r", beg
printf "fin    is: %g\r", fin
```

```
printf "Delta T    is: %g\r", deltemp
printf "Time of Pulse    is: %g\r", plstime[0]/1000
printf "heat capacity    is: %g\r", hc1

wave wave2=heat_cap_1                // puts heat cap into wave heat_cap_1
InsertPoints 0,1,wave2
wave2[0] = hc1

wave wave50=final_temp
wave wave51=C_t_final
wave wave52=T_square
wave wave53=addhc_1
wave wave54=final_sh
InsertPoints 0,1, final_temp, C_T_final, T_square, addhc_1, final_sh

wave wave4=temp_1
InsertPoints 0,1,wave4
wave4[0]=(BASETEMP_RuO2_INIT[0]+BASETEMP_RuO2_FIN[0])/2                                    //
initial and final base temp from RuO2

wave wave32=tempcal_1
InsertPoints 0,1,wave32
wave32[0]= (wave0[10] + linefit[10] + wave0[420] + linefit[420])/2
//      485 -> 420
// temperature from the sample by using LR-400

String fitText                              //puts temp,dt and hc values on graph
sprintf fitText, "\\Z08temp = %g K\r\rtau=%g sec\rdT=%g K\rhc=%g J/K\r",wave32[0],tau,deltemp, hc1
TextBox/C/N=power fitText

End
//*****************************************************************
//*****************************************************************
//*****************************************************************
//*****************************************************************
```

```
Function expon()
wave wave0 = calres

variable o

o = BRGOFFSET[0]
wave0 = wave0*100000 + o*100000            // set CALRES to Ohms and correct for offset

variable a                                                    // define calibration constants
variable b                                                  //Note: this is still "incorrect" temp...
variable c                                                 //This ia corrected for in procedure: "tempadj"
variable d
variable e
variable f
variable g
variable h
variable j
a = 0.2700714
b =    0.07323249
c = 1.0219809
d = -1.9558035
e = 3.4889894
f = -4.4175955
g = 3.6976856
h = -1.7871035
j = 0.3821843

variable r                        // define calorimeter therm. resistance at room temp
r = 77408

//calibration curve for calorimeter therm: changes R(t) to T(t)
wave0 = (a +b*(Sqrt(Sqrt(Ln(wave0/r)))) + c*(Ln(wave0/r)) +d*(Ln(wave0/r))^2 +
e*(Ln(wave0/r))^3+f*(Ln(wave0/r))^4+g*(Ln(wave0/r))^5+h*(Ln(wave0/r))^6+j*(Ln(wave0/r))^7)^(-4)

wave wave1 = plsvolt                    // create variables for use
```

```
variable s
variable m
variable z
variable n
variable p
variable v
variable w
variable y
variable q
variable pow
variable dt

variable int
int = int1[0]/1000
// defines interval bewteen CALRES data points in units of seconds

s = int2[0]/1000
// defines interval between PLSVOLT data points in units of seconds

m= 172
//For the ~9 sec baseline, the first measurement of the voltageis at point #176 -> #172
variable v2                    // this is just one voltage reading
v2 = wave1[250]
printf "A Pulse Voltage is: %g\r", v2

pow = v2*(PLSCRT[250])              // calculates the power input using P=iV
printf "The power input using P=iV    is: %g\r", pow

SetScale/P x 0,int,"sec", wave0
// corrects x-axis to show correct time scaling in seconds
Label left "Temp [degrees K]"
SetAxis/A
//ModifyGraph rgb(calres)=(1,4,52428)

variable ibt
ibt = basetemp_ruo2_init[0]
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
variable beg
beg = m*s +8+ plstime[0]/1000
// Figures out a place to start the curvefit, and can be ////////////////////////////////////////
// changed if desired.     8 ->14.78     ;1->7.78; 15->21.63
//////////////////////////////// m=172, s=plstime/1000


CurveFit line wave0(0,m*s) /D                    // fits a line to the initial baseline

wave wave15=linefit                              // creates a wave which is the baseline fit
SetScale/P x 0,int,"sec",wave15
wave15=W_coef[0] + W_coef[1]*x

wave0 = wave0 - linefit                          // subtracts the baseline from the T(t) curve
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
variable fin
fin =300 * int

// creates the place to stop the curvefit, and can also be changed.

printf "beg    is: %g\r", beg
printf "fin    is: %g\r", fin

CurveFit exp wave0(beg,fin) /D
// fits the temperature decay part after the pulse to expinential

variable tau
tau = 1/W_coef[2]

variable chisquare2
chisquare2 = V_chisq

wave wave17=expfit                //creates a wave out of the temperature decay fit.
SetScale/P x 0,int,"sec",wave17
wave17= W_coef[0] + W_coef[1]*exp(-W_coef[2]*x)
```

```
CurveFit line wave0(0,m*s) /D                    //fits a line to the new baseline

printf "m*s    is: %g\r", m*s

wave wave16=flatlinefit                    // creates a wave out of this fit
SetScale/P x 0,int,"sec",wave16
wave16=W_coef[0] + W_coef[1]*x

//The following section matches the areas so that dT can be accurately estimated.
//---------------------------------------------------------------------
variable left
variable rt
variable ll
variable rr

wave wave20 = leftsum
wave wave21 = rtsum

Redimension/N=0 wave20
Redimension/N=0 wave21

ll = m*s
rr = beg

InsertPoints 0,1, wave20
wave20[0] = wave0(ll) - flatlinefit(ll)

InsertPoints 0,1, wave21
wave21[0] = expfit(rr) - wave0(rr)

left = leftsum[0]
rt = rtsum[0]

variable endpls
```

// This DO loop inreases ll by 0.01 and decreases rr by 0.01 until the position is found //where the areas are equal on both sides. this position is then called "endpls".

```
DO

        IF (left<rt)
                ll = ll+.01
                |print p
                InsertPoints 0,1, wave20
                wave0[0] = wave0(ll) - flatlinefit(ll)
                left = wave0[0] + left
        ELSE
                rr = rr-.01
                InsertPoints 0, 1, wave21
                rtsum[0] = expfit(rr) - wave0(rr)
                rt = rtsum[0] + rt
        ENDIF


        IF (ll > rr -.01)
                print rr
                print ll
                endpls = (ll+rr)/2
        ENDIF

WHILE (ll < rr  )

variable deltemp
deltemp = expfit(endpls) - flatlinefit(endpls)
```

// This defines the delat T to be equal to the difference btwn the fit to the temp decay // and the baseline at the position previaouls called "endpls"

```
//print (00)
//print deltemp
//printf "deltemp  is: %g\r", deltemp
//print endpls
```

```
wave wave30 = DT_1                    //puts this value of delta t into wave DT_1
InsertPoints 0,1,wave30
wave30[0]=deltemp

wave wave31 = tp_1                              // puts time of pulse into tp_1
InsertPoints 0,1,wave31
wave31[0]=plstime[0]/1000

wave wave11=vertline                            // makes it so a vertical line can be drawn
vertline = endpls                     //at the point which makes areas equal.
AppendToGraph expfit vs vertline
//ModifyGraph rgb(flatlinefit)=(0,0,0)
AppendToGraph expfit
AppendToGraph flatlinefit

variable low                                            //sets axis of graph
variable hi
low = W_coef[0] - .002
hi=deltemp+.005                                                 // .05->.3
//SetAxis/A left
SetAxis left low, hi

v2 = wave1[250]
printf "A Pulse Voltage is: %g\r", v2

pow = v2*(PLSCRT[22])                 // calculates the power input using P=iV
printf" The input pulse current is: %g\r",PLSCRT[22]
printf "The power input using P=iV    is: %g\r", pow

variable hc1                                    //Caluclulates Heat Capacity
hc1 = pow*plstime[0]/1000/deltemp
printf "pow    is: %g\r", pow

printf "beg    is: %g\r", beg
printf "fin    is: %g\r", fin
```

```
printf "Delta T    is: %g\r", deltemp
printf "Time of Pulse    is: %g\r",plstime[0]/1000
printf "heat capacity    is: %g\r", hc1

wave wave2=heat_cap_1                          // puts heat cap into wave heat_cap_1
InsertPoints 0,1,wave2
wave2[0] = hc1

wave wave50=final_temp
wave wave51=C_t_final
wave wave52=T_square
wave wave53=addhc_1
wave wave54=final_sh
InsertPoints 0,1, final_temp, C_T_final, T_square, addhc_1, final_sh

wave wave4=temp_1
InsertPoints 0,1,wave4
wave4[0]=(BASETEMP_RuO2_INIT[0]+BASETEMP_RuO2_FIN[0])/2                                    //
initial and final base temp from RuO2

wave wave32=tempcal_1
InsertPoints 0,1,wave32
wave32[0]= (wave0[10]+linefit[10]+wave0[420]+linefit[420])/2       // 485 -> 420

String fitText                                 //puts temp,dt and hc values on graph
sprintf fitText, "\\Z08temp = %g K\r\rtau=%g sec\rdT=%g K\rhc=%g J/K\r",wave32[0],tau,deltemp, hc1
TextBox/C/N=power fitText

end

//*****************************************************************
//*****************************************************************
//*****************************************************************

Function addenda()                         //subtracts addenda data and divides by # moles
```

```
//Note: addenda data was taken previously for the addenda with a
// small amount of N-grease on it.  If desired, a separate addenda
// run can be done for every sample and then subtract that off, but
//it is believed that this is completely uneccessary.

wave wave0 = addhc_1
wave wave1 = tempcal_1
wave wave2 = heat_cap_1
wave wave3 = final_sh
wave wave4 = no_moles

wave0 = (wave1/30)^3
wave0 = wave0 * 0.00034
wave0 = wave0 + 0.00014*(65/wave1)^2*Exp(65/wave1)/(Exp(65/wave1) - 1)^2

wave3 = (wave2-wave0)/wave4[0]

printf "specific heat of addenda is: %g\r", wave0[0]
printf "final specific heat is: %g\r", wave3[0]

End

//*****************************************************************
//*****************************************************************
//*****************************************************************
//*****************************************************************
Function tempadj()                    //corrects temperature using calibration curve

wave wave0 = final_sh
wave wave1 = tempcal_1
wave wave2 = final_temp

wave2 = 0.080563 + 0.97266*wave1 + 0.006985*(wave1)^2

printf "Final Temp is: %g\r", wave2[0]
```

```
Display wave0 vs wave2

End

//*****************************************************************
//*****************************************************************
//*****************************************************************
//*****************************************************************
Function C_over_T()                              //Plots C/T vs T^2

wave wave0 = final_sh
wave wave1 = final_temp
wave wave2 = C_T_final
wave wave3 = T_square

wave2 = wave0/wave1
wave3 = (wave1)^2

Display wave2 vs wave3
//ModifyGraph mode=3

End
```