University of Nevada, Reno

Use and Uncertainty of an Inverse Heat Conduction Code for Estimating Heat Flux to a Large Pipe Calorimeter in a Jet Fuel Fire

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Mechanical Engineering

by

Timothy R. Bullard

Dr. Miles Greiner/Thesis Advisor

May, 2010

University of Nevada, Reno
Statewide • Worldwide

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

TIMOTHY BULLARD

entitled

Use And Uncertainty Of An Inverse Heat Conduction Code For Estimating Heat Flux To A Large Pipe Calorimeter In A Jet Fuel Fire

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Miles Greiner, Advisor

Chanwoo Park, Committee Member

George Fernandez, Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

May, 2010

ABSTRACT

Heat flux to a large pipe calorimeter is determined by use of an inverse heat conduction code for the purpose of benchmarking large scale fire simulators. Benchmarked results from Container Analysis Fire Environment (CAFE) simulations are used to calibrate an inverse heat conduction code by optimizing number of future times (NFT) at 11 and identifying a linear correlation and uncertainty range for the unknown heat flux. Time-varying heat flux is estimated for three fire tests at 58 locations along the calorimeter occupied by thermocouples in the original experiments. A Curie transition in the pipe calorimeter steel causes high levels of instability in the calculation, and as a result any data obtained from the inverse heat conduction code during this transition are discarded. Results for a single thermocouple location are presented and are representative of other thermocouples in the experiments. The calibration is then used to adjust the heat flux prediction for the calorimeter during the three experiments. Estimated heat fluxes are approximately an 11 second window average of the actual heat fluxes. The maximum heat flux occurred at the beginning of the first experimental fire and was 195 ± 37 kW/m^2 at a 95% confidence level.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Miles Greiner for his support and encouragement throughout this research project. I would like to thank him for providing a sound foundation for conducting research, learning, and discovering new experiences. He made a generous effort to be available to me which facilitated an excellent environment for this work. He has also acted as a role model through which I have gained many insights into the future of my career as an engineer.

I would also like to thank my committee member, Dr. Chanwoo Park, for being a helpful and encouraging colleague during my graduate studies. The numerous research projects that I have conducted under his guidance prior to and during this graduate program have greatly enhanced my experiences.

Thanks to Dr. George Fernandez, for working with me to serve on my defense committee and making the effort to help me satisfy degree requirements. His discussion on statistical methods is also highly appreciated.

Finally, I thank Narayana Chalasani for assisting me in many endeavors and teaching me certain skills which will be invaluable to me in the future.

This work was sponsored by the United States Department of Energy Global Nuclear Energy Partnership (GNEP) under contract DEFC07-06ID14782.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

a	linear correlation slope
b	linear correlation intercept [kW/m^2]
k	thermal conductivity [W/mK]
N	number of temperature samples in fire test
NFT	number of future times
N_{It}	total number of cumulative iterations
q_A	applied heat flux [kW/m^2]
q_I	SODDIT-indicated heat flux [kW/m^2]
q_W	window-averaged applied heat flux [kW/m^2]
ρc	volumetric specific heat [MJ/m^{3}K]
R_T	temperature random (noise) error [K]
S_A	68%-confidence-level uncertainty of the applied heat flux
S_T	temperature bias error
S_W	68%-confidence-level uncertainty of the window-averaged heat flux
$T_{i,C}$	inner surface temperature calculated by CAFE [C]
$T_{i,M}$	measured inner surface temperature [C]
$T_{i,S}$	inner surface temperature with errors used as input to SODDIT [C]
$T_{o,C}$	outer surface temperature calculated by CAFE [C]
$T_{o,I}$	outer surface temperature indicated by SODDIT [C]
u	random number in standard normal distribution
w	width of window average in seconds
w_x	95% uncertainty limit of x
x	material property or dimension
$\overline{x}$	nominal value of x

SUPERSCRIPTS

m	cumulative iteration number
n	fire test number
p	thermocouple number
t	time (s)

INTRODUCTION

Spent nuclear fuel transport casks must maintain their containment, shielding and criticality functions even under severe accident conditions [1]. The response of these casks under severe fire conditions can be demonstrated using either physical testing or computer simulations.

The Container Analysis Fire Environment (CAFE) computer code links a computational fluid dynamics (CFD) fire simulator to package thermal models [2]. It was developed by Sandia National Laboratories to predict the response of spent nuclear fuel transport casks under severe accident conditions. CAFE is highly versatile, and can use different models to balance simulation time against modeling accuracy. For example, the cask model can be a fully three-dimensional and geometrically-accurate finite element (FE) thermal model. It can also be a series of one-dimensional modules that model conduction and radiation in the radial direction at several locations of the cask walls (but neglects axial and azimuthal heat transfer). The fire simulator uses a number of physics-based fuel evaporation, turbulent mixing, reaction chemistry, and radiation heat transfer models, each of which employ modeling parameters. The value of these parameters must be validated from comparisons with large fire test data before CAFE can be used with confidence to predict cask fire behavior.

In 2007 three large-scale fire experiments were conducted wherein a 2.4 m-(8ft)-diameter, 4.6 m-(15ft)-long, and 2.54 cm-(1in)-thick A36-steel calorimeter was suspended 1 m above a 7.9 m-(26ft)-diameter pool of containing water [3]. In each test, 7,570L (2,000gal) of jet fuel floated on the water and was burned until it was exhausted.

The three tests, referred to as Test 1, 2, and 3, experienced average wind speeds of 1.0, 1.2, and 3 m/s, and lasted 40.5, 37.6, and 25.3 minutes, respectively.

Fifty-eight thermocouples were placed on the interior surface of the calorimeter and backed by three inches of insulation. Ten were located on the endplates, and 48 were located on fire rings of the pipe body. Figure 1 shows the calorimeter, fuel pool, and some of the inner thermocouple locations. The wind conditions were measured by twelve ultrasonic anemometers on four poles around the burn site. The calorimeter inner surface temperatures and wind conditions were measured every second during and after the three fires to acquire data to benchmark fire models and simulators. The article by Greiner et al. (2009) [3] describes the fire tests, wind conditions, and temperature measurements in detail. The data acquired in the tests are available through a website which can be reached by contacting the authors.

del Valle [4] used the data acquired through the three tests to adjust and benchmark CAFE. The measured wind conditions were applied as boundary conditions to the fire simulator. Fifty-eight one-dimensional heat transfer models were used to calculate the calorimeter response of the calorimeter wall at each thermocouple location. The CAFE-calculated interior surface temperatures were compared to the measured data for all 58 locations and all times during all three fire tests. The effect of the modeling parameters on the comparison was determined.

Figure 2 shows the simulated and experimental average calorimeter temperature rise over time for the three tests. This temperature rise is the average temperature of the 58 thermocouple locations minus the initial temperature. Test 3 experienced stronger wind conditions than Tests 1 and 2. As a result, the calorimeter was less engulfed and experienced a smaller temperature rise in Test 3 than the other two experiments. Agreement of the experimental and simulated average temperature rise shows that the simulation effectively captures the dependence of the average heat transfer on wind conditions.

CAFE can be adjusted and benchmarked using measured interior surface temperatures because it incorporates a calorimeter model that calculates those temperatures. Other fire models and simulators predict heat flux to an object's external surface as a function of the surface temperature but do not include a heat transfer model of an engulfed object [5, 6]. Since they do not calculate the interior surface temperature, they cannot be directly benchmarked using the recent data set. To benchmark these codes and models, the values and uncertainty of the outer surface heat flux and temperature must be estimated.

The objective of the current work is to determine the value and uncertainty of the heat flux to the exterior surface of the calorimeter near all 58 measurement thermocouples, at all times during the three tests. The problem of finding the temperature throughout an object from its initial temperature and boundary heat flux is relatively well posed. This involves *integration* of heat flux, which is not sensitive to small random errors. To find temperatures at a given time we use heat fluxes at *earlier* times. The inverse problem, of finding surface heat flux from temperature history at another location, is ill-posed [7]. This involves *differentiation* of the temperature, which is highly sensitive to random errors. To find heat flux at a given time, we use temperatures at *future* times.

One option available for the determination of external surface heat flux from inner surface temperature measurements is the Sandia One-Dimensional Direct and Inverse Thermal (SODDIT) code [8]. SODDIT is a one-dimensional conduction code that can accommodate planar, cylindrical, and spherical geometries and assumes no phase changes occur during the calculation. SODDIT calculates conduction in the direction normal to the wall surface and neglects it in the other two directions. In an inverse problem, SODDIT uses an iterative calculation to find a heat flux whose resulting temperature response most nearly matches the supplied measured temperature. SODDIT finds the outer surface heat flux and temperature at discrete times. The required inputs are inner surface temperatures at the same discrete times, temperature-dependent volumetric-specific heat and thermal conductivity, wall thickness, and a parameter referred to as the number of future times (NFT). NFT is the number of temperature samples-at future times-that are used to find heat flux at a given time.

In SODDIT, NFT is specified by the user within the range $1 \leq \text{NFT} \leq 25$. Increasing its value adds damping and stability to the inverse calculation [8]. This is necessary to avoid sensitivity to random errors or abrupt changes in temperature. However, excessive NFT may cause SODDIT to overly damp important time-dependent heat flux variations and decrease accuracy. In the current work we consider how to select NFT to balance damping and stability of the inverse calculation.

The pipe calorimeter A36-steel undergoes a Curie effect, or a change in magnetic properties from ferromagnetic to paramagnetic [9], during a short time while being heated by the fire. Much like a phase change, this transition is characterized by absorption of energy without an increase in temperature. SODDIT is not designed to handle this type of transition. In the current work, we consider the validity of results obtained during and outside of this transition time period.

As mentioned earlier, del Valle [4] performed CAFE simulations of the three recent fire tests. These simulations calculated the heat flux applied to the calorimeter exterior surface by the fire, and the resulting calorimeter inner and outer surface temperatures. In the current work, the CAFE-calculated inner surface temperatures are used as input to SODDIT to determine the SODDIT-indicated outer surface temperature and heat flux to the calorimeter during the fires. This SODDIT-indicated heat flux was calculated using different values of NFT and compared to the CAFE-applied heat flux. This comparison was performed using a linear regression and confidence interval, which characterized systematic and random uncertainties in the calculation. The value of NFT that minimized the difference between the SODDIT-indicated and CAFE-applied heat flux is used for all subsequent calculations.

The SODDIT calculations were then repeated with randomly-generated simulated measurement uncertainty added to its inputs. The amount this uncertainty increased the difference between the SODDIT-indicated and CAFE-applied heat flux is quantified. Finally, the CAFE-applied heat flux was window averaged with different window widths. The window width that brought the results closest to the SODDIT-indicated heat flux is used to characterize the type of damping inherent in the SODDIT calculation.

The website created for the data of Greiner et al. (2009) [3] also lists the time- and location-dependent heat flux determined in this work. It is available by contacting the authors. This paper describes how to find the heat fluxes that are provided on that website.

CALORIMETER MATERIAL PROPERTIES

Material properties required for a SODDIT calculation are the volumetric specific heat (ρc) and thermal conductivity (k). These properties were experimentally measured for the A36-steel used in the fire experiments [10], and are shown in Figure 3. Lines denote the upper and lower 95% confidence level uncertainties. A spike in the specific heat can be seen in the temperature range of 726 to 764 ºC. This range is indicated using vertical lines and is referred to as the Curie temperature range. The specific heat was measured using the differential scanning calorimetry (DSC) method [10]. DSC is a technique in which heat transfer to an object and its change in temperature with respect to an unheated, reference object, are measured. The heat capacity is then obtained by taking the ratio of these two parameters. DSC measurement of the A36 steel produces a spike in the heat capacity due to a Curie effect. This transition is endothermic and the heat absorbed is reflected in DSC by an increase in heat capacity.

The 95% confidence level uncertainties of the quantities used as input to SODDIT are collected in Table 1. The uncertainties in specific heat and thermal conductivity are 2% and 4%, respectively. The actual uncertainty in the specific heat during the Curie temperature range is unknown but is expected to be significantly greater than that shown in Figure 3. The calorimeter wall thickness is assumed to vary spatially by 4% based on manufacturing tolerances. The thermocouples used in the fire tests are specified to have approximately 0.76% span error and were experimentally determined to exhibit random noise on the order of 0.02K.

HEAT FLUX CALCULATION UNCERTAINTY

In the current work, temperature measurements made by the inner thermocouples are used as input to SODDIT to estimate the outer surface temperature and heat flux at each measurement location. The uncertainties associated with these predictions are also estimated.

The dots in Figure 4a show the CAFE-calculated heat flux that was *applied*, q_A, to one location of the calorimeter versus time during Test 1 [4]. (This location, numbered p=17 of 58, was on the outer surface of the north side of the calorimeter at the midpoint of its height and length, external to thermocouple #3000 in Greiner et al. [3].) The simulated fire is ignited at t=0 and flames quickly engulf the calorimeter, initially causing a sharp increase in heat flux. The heat flux is highly oscillatory due to fire motion but generally decreases with time as the calorimeter temperature approaches that of the fire.

The lines in Figure 4b marked $T_{i,C}$ and $T_{o,C}$ are the *inner* and *outer* surface temperatures calculated by *CAFE* [4]. The surface temperatures rise more rapidly at the beginning of the fire, when the temperature difference between fire and calorimeter are greatest, than at the end. The inner surface temperature $T_{i,C}$ is slightly less oscillatory than outer surface temperatures $T_{o,C}$ and $T_{o,I}$ due to the thermal mass of the calorimeter wall. The Curie temperature range is shown using two horizontal lines. The Curie time period is defined as any time when either the inner or outer surface temperatures of the calorimeter are within the Curie temperature range. This period is shown in Fig. 4a and 4b using vertical lines. During this period, CAFE used specific heats with a very large uncertainty.

In the current work, $T_{i,C}$ is used as input to SODDIT to calculate the *indicated outer surface temperature* $T_{o,I}$ and heat flux q_I. The values of $T_{o,I}$ (Fig. 4b) and q_I, (Fig. 4a) generated using NFT=10 and 15 are shown.

Within the Curie time period, $T_{o,I}$ exhibits oscillatory behavior and the oscillations are larger for NFT=10. Outside the Curie time period, $T_{o,I}$ is not strongly affected by choice of NFT and agrees closely with $T_{o,C}$ regardless of NFT. Generally, SODDIT-indicated outer temperatures are very well matched with those calculated by CAFE. Because of this, remaining discussion will not focus on temperature predictions, but on outer surface heat flux.

In Figure 4a, outside the Curie time period, q_I for NFT=10 and 15 agree closely with each other and appear to be window averages of the applied heat flux, q_A. Figure 4c is an expanded region of Figure 4a from t=10 to 16 min. The values of q_A and q_I (NFT=10 and 15) are the same as shown in Figure 4a. We can see that q_I for NFT=10 and 15 are similar to each other but increased damping is attained with a higher NFT. We can see that the CAFE-applied heat flux q_A is highly oscillatory in comparison with q_I. An 11 second window average of q_A, called q_W, is also plotted. We see that the window-averaged CAFE heat flux is in closer agreement with SODDIT-indicated heat flux.

In Fig. 4a, within the Curie time period, NFT=10 and 15 exhibit high amplitude oscillations, but NFT=10 is more oscillatory. The behavior of q_I within the Curie time may be attributed to the spike in the specific heat curve in the Curie temperature range. For the remainder of this work, we discard heat fluxes calculated during Curie time period due to their high contribution to overall uncertainty.

The general agreement of q_I and q_A may be partially due to SODDIT and CAFE considering radial conduction alone. Future work may employ more computationally intensive three dimensional FE models that include the effects of axial and azimuthal conduction.

We wish to quantify how well the SODDIT-indicated flux q_I follows the CAFE-applied flux q_A. Figure 5 is a plot of the SODDIT-indicated heat flux q_I versus CAFE-applied heat flux q_A. It uses the same data as Fig. 4 with NFT=10, except the heat fluxes calculated during the Curie time period are not included. The number of total points for this location (p=17) and Test 1 (n=1) is $N^{n,p}=N^{1,17}$=2184. This is the duration of the fire test in seconds (2435s) minus the number of samples that occurred within the Curie time period (251s). The value of N for different n and p varies due to differences in the duration of each respective Curie time period.

Ideally, the indicated flux would exactly match the CAFE-applied flux, that is, all the data would fall along the line $q_I = q_A$. Deviations of the data from this line are the result of random and systematic errors in the assumptions and calculation methods in SODDIT. For example, for a given value of the applied flux q_A, non-repeatable (random) errors cause SODDIT to indicate a range of different fluxes q_I. In Fig. 5, a line ($q_I = aq_A+b$) is fit to the data using a least squares method [11] with a slope $a=0.74$ and intercept b=15 kW/m^2. Deviations from a=1 and b=0 are measures of the systematic errors in the calculation. This linear fit can be inverted to yield Eq. 1.

$$q_A = \frac{q_I - b}{a} \qquad \text{(Eq. 1)}$$

This relationship can be used to determine the best estimate of the applied flux q_A for a given value of the SODDIT-indicated flux q_I. Evaluating Eq. 1 for q_A would be sufficient if all the data in Fig. 5 lay directly on the fit line. However, random errors exist in addition to systematic errors. Thus, by characterizing only the systematic uncertainty in the indicated heat flux q_I, we do not fully describe the uncertainty in a SODDIT calculation.

We wish to understand the degree of random uncertainty in a value of SODDIT-indicated heat flux. That is, we now wish to quantify the range of q_A indicated by a single value of q_I. This range can be quantified by a standard deviation of q_A about the fit line. Figure 5 shows the standard deviation of the applied flux about the fit line, S_A. For q_I (NFT=10) and q_A of Fig. 4a, S_A can be calculated by Eq. 2, in which the differences between the applied heat flux and fit line are summed over all times t (in seconds). This evaluates S_A for the full fire duration.

$$S_A = \sqrt{\frac{\sum_{t=1}^{N^{1,17}} \left[\left(\frac{q_I^t - b}{a}\right) - q_A^t\right]^2}{N^{1,17} - 1}} \qquad \text{(Eq. 2)}$$

S_A is a measure of the random uncertainty in the applied heat flux. It is an indication that 68% of the applied heat fluxes will land horizontally within $\pm S_A$ of the fit line. For the data in Fig. 5 (n=1, p=17), S_A=15 kW/m^2.

For this calculation, time steps within the Curie time period are omitted from the summation over the range of t=[1, $N^{1,17}$], and the denominator value of $N^{1,17}$ is reduced by the number of omitted time steps. This same process is used for any subsequent calculations of S_A that exclude the Curie time period data.

Now, we wish to quantify S_A including the data from all thermocouple locations and fire tests. In Eq. 3 which the differences between the applied heat flux and fit line are summed over all times t, thermocouple locations p, and fire tests n.

$$S_A = \sqrt{\frac{\sum_{n=1}^{3}\sum_{p=1}^{58}\sum_{t=1}^{N^{n,p}}\left[\left(\frac{q_I^{n,p,t}-b}{a}\right)-q_A^{n,p,t}\right]^2}{\sum_{n=1}^{3}\sum_{p=1}^{58}N^{n,p}-1}} \qquad \text{(Eq. 3)}$$

For this work we prefer to use twice the value of S_A ($2S_A$). This confidence interval quantifies the uncertainty for a larger majority of data. It is an indication that 95% of the applied fluxes will land within $\pm 2S_A$ of the fit line.

The values of *a, b,* and S_A of Eq. 3 are unique for each value of NFT. The line in Figure 6 labeled *Exclude Curie,* $N_{it}=1$ is a plot of $2S_A$ versus NFT excluding from the calculation any data within the Curie temperature range. The line labeled *Include Curie,* $N_{it}=1$ includes the data within the Curie time period. We can see that for low NFT, including data within the Curie time period significantly increases $2S_A$. We recall from Fig. 4a the erratic behavior of the SODDIT-indicated heat flux during the Curie time period. Attributing the cause of said behavior to the spike in the specific heat curve, a direct link exists between the uncertainty of the SODDIT calculation and the specific heat curve. Thus it is desirable to minimize uncertainty in the heat flux calculation by excluding the data within the Curie time period.

This being so, we cannot claim that the indicated heat flux q_I can be used to estimate the actual applied flux q_A during the Curie time period.

The locations in Fig. 6 where $2S_A$ are at a minimum are circled. Generally, $2S_A$ is high at low values of NFT, decreases over the next several NFT until a minimum is reached, and then begins to increase. Using NFT in the range of $1 \leq \text{NFT} \leq 8$ would indicate uncertainties orders of magnitude higher than those of Fig. 6 and may cause the SODDIT calculation to fail due to instability. The damping effect of NFT is easily seen by higher values of $2S_A$ at low NFT (less stable), transitioning to lower values of $2S_A$ at slightly higher NFT (more stable). As NFT is increased beyond a minimum, $2S_A$ increases as the SODDIT-indicated heat flux becomes such a damped prediction of the applied flux that accuracy is lost. The curves converge toward NFT=25 as the SODDIT-indicated heat fluxes during the Curie time period are no longer highly oscillatory, thereby contributing minimally to the magnitude of $2S_A$.

EFFECT OF MEASUREMENT UNCERTAINTY

The uncertainty S_A calculated in the last section assumed the measured quantities (volumetric specific heat ρc, thermal conductivity k, wall thickness d, and interior surface temperatures) were known exactly. However, uncertainty in these values contributes to overall uncertainty of a SODDIT calculation. We wish to quantify S_A considering the effect of the uncertainties listed in Table 1.

Let x be a SODDIT input parameter (i.e. ρc, k, or d) for which there is known uncertainty. Repeated measurements of x will yield different values. We characterize the range of different values of x by the nominal or average value $\overline{x}$ and 95% confidence level uncertainty w_x. We assume the values of x are normally distributed about the nominal value $\overline{x}$.

The following expression (Eq. 4) is used to calculate each SODDIT input parameter,

$$x^{m,n,p} = \overline{x}\left(1 + \frac{w_x}{2} u_x^{m,n,p}\right) \qquad \text{(Eq. 4)}$$

in which $x^{m,n,p}$ is determined for each thermocouple location p and test n and repeated for each cumulative iteration m. Each $x^{m,n,p}$ is calculated from the nominal value of the parameter $\overline{x}$, its 95% confidence level uncertainty w_x, and a random number $u_x^{m,n,p}$ generated within the standard normal distribution. If x is a function of temperature (i.e. ρc, k), $u_x^{m,n,p}$ is constant over the range of temperatures for which the material property is defined. In this work, the Polar Method is used for standard normal distribution number generation [12].

Similarly, the inner surface temperatures are adjusted for measurement uncertainty according to Equation 5. The inner surface temperature provided to SODDIT $T_{i,S}^{m,n,p,t}$ is calculated for each *m, n, p,* and *t* from the inner surface temperature calculated by CAFE $T_{i,C}$, span error S_T, and random noise R_T.

$$T_{i,S}^{m,n,p,t} = T_{i,C}^{n,p,t}\left(1+\frac{S_T}{2}u^{m,n,p}\right)+\frac{R_T}{2}u^{m,n,p,t} \qquad \text{(Eq. 5)}$$

The line in Figure 7 shows the SODDIT-indicated heat flux q_I versus time for the same thermocouple location as Fig. 4 but for NFT=11. The dots show q_I calculated using uncertainties in the SODDIT input parameters. The magnitudes of the uncertainties are randomly calculated within the 95% confidence level ranges of Table 1. We see that for a single iteration the SODDIT-indicated fluxes calculated with uncertainties are similar but slightly varied from those without uncertainties. There is little visible effect on the heat flux prediction for a single iteration.

It is then necessary to quantify the effect of measurement uncertainties on S_A over many iterations. S_A is now calculated by Eq. 6, where measurement uncertainties are calculated and added to the SODDIT input data for every iteration *m,* fire test *n,* and thermocouple location *p*.

$$S_A = \sqrt{\frac{\sum_{m=1}^{N_{it}}\sum_{n=1}^{3}\sum_{p=1}^{58}\sum_{t=1}^{N^{n,p}}\left[\left(\frac{q_I^{m,n,p,t}-b}{a}\right)-q_A^{m,n,p,t}\right]^2}{\sum_{m=1}^{N_{it}}\sum_{n=1}^{3}\sum_{p=1}^{58}N^{m,n,p}-1}} \qquad \text{(Eq. 6)}$$

The lines in Figure 8 show $2S_A$ over 50 cumulative iterations for NFT 10 and 11. Heat fluxes within the Curie time period are excluded from the calculation. At m=0, $2S_A$ for NFT=10 is lower than that for NFT=11. Measurement uncertainties are included in the calculation beginning at m=1. Both lines increase with iteration m and level off quickly as the normal distributions of the input data are filled out. After 50 iterations, $2S_A$ for NFT=11 is now lower than NFT=10. The cumulative effect of measurement uncertainties is relatively small compared to the magnitude of the overall uncertainty.

In Figure 6, the line labeled *Exclude Curie, Measurement Uncertainties Included* $N_{it}=50$ shows $2S_A$ versus NFT with uncertainties included in the SODDIT input data. At NFT=9 we see that measurement uncertainties cause a large increase in the value of $2S_A$ with respect to the value without uncertainties. This is a clear indication of the sensitivity of SODDIT to small random errors. A minimum is circled at NFT=11. With the consideration of measurement uncertainties, the optimal NFT has changed from 10 to 11.

With $2S_A$ optimized using NFT=11, the correlation constants a and b Eq. 6 are selected. Substituting into Eq. 1, we may estimate the unknown applied flux q_A based on the SODDIT-indicated flux q_I using Eq. 7.

$$q_A = 1.1 q_I - 6.2 \ \pm 37 \, {}^{kW}\!/_{m^2} \quad (95\%) \qquad \text{(Eq. 7)}$$

Eq. 7 corrects systematic error and uses S_A to characterize random uncertainty in the SODDIT-indicated heat flux. This equation is the result of 50 cumulative iterations of all three fire tests, 58 thermocouples, and all time steps (excluding the Curie time period) with uncertainties in the input data. The total number of data points included is approximately 16.3 million.

EFFECT OF WINDOW AVERAGING

As previously discussed, SODDIT-indicated heat fluxes appear to be a damped or window-averaged prediction of the actual applied heat flux. In Fig. 4c we saw that an 11 second window average of the CAFE-applied heat flux q_W was in fair agreement with the SODDIT-indicated flux q_I for NFT=10 and 15. In order to estimate the amount of window-averaging caused by a SODDIT calculation, we quantify the agreement of the window-averaged CAFE heat flux q_W with the SODDIT-indicated flux q_I using NFT=11.

The window-averaged heat flux, q_W, is calculated using the CAFE heat flux q_A by Eq. 8.

$$q_W^{n,p,t} = \frac{1}{w} \sum_{j=-\frac{1}{2}(w-1)}^{\frac{1}{2}(w-1)} q_A^{n,p,t+j} \qquad \text{(Eq. 8)}$$

In this equation, q_W is found at a given time step by averaging a number of surrounding q_A equal to the window width (w). Window-widths are always odd values so that the window is symmetric about q_A at a given time step.

Eq. 9 can be used to calculate S_W, or the uncertainty in the window-averaged heat flux q_W with respect to the SODDIT-indicated heat flux q_I.

$$S_W = \sqrt{\frac{\sum_{n=1}^{3}\sum_{p=1}^{58}\sum_{t=1}^{N_n}\left[\left(aq_I^{n,p,t} + b\right) - q_W^{n,p,t}\right]^2}{\sum_{n=1}^{3}\sum_{p=1}^{58} N^{n,p} - 1}} \qquad \text{(Eq. 9)}$$

Similar to Eq. 3, this equation evaluates S_W for all fire tests n, thermocouple locations p, and times t. The Curie time period data are excluded from the calculation. New values of a and b are evaluated for this calculation.

Figure 9 is a plot of S_W versus a range of window widths. Similar to S_A, S_W is large for small and large window widths, indicating under- or over-averaging of the CAFE-applied heat flux. The window width which minimizes S_W is circled at w=11; therefore, q_I is most nearly an 11 second window average of q_A. This may be a direct result of the number of temperature samples used for each calculation step in SODDIT; in other words, by the selection of NFT=11.

ESTIMATION OF HEAT FLUX USING MEASURED TEMPERATURES

In the previous sections it has been shown that a SODDIT calculation with NFT=11 gives the best estimate of the applied heat flux for the recent fire tests, assuming the given uncertainties in the material properties, dimensions, and temperature measurements. The estimate is an approximation of an 11 second window average of the actual heat flux. It is now possible to use the calibration and uncertainty range with experimental data. In this section we summarize the previous method and apply it to the measured data for the recent fire tests. Applying Eq. 7 to the SODDIT-indicated heat fluxes that use the experimental inner temperature measurements acquired by Greiner et al. (2009) [3], the external heat flux and surface temperature of the calorimeter during the experiment can be estimated.

Figure 10a shows the inner surface temperature measured during the first experimental fire test $T_{i,M}$ versus time for the same thermocouple location of Figure 4. We wish to use this as input to SODDIT along with the volumetric specific heat, thermal conductivity, and wall thickness. Randomly generated uncertainties are calculated and added to all inputs. SODDIT then calculates the indicated outer surface temperature $T_{o,I}$ and heat flux q_I using NFT=11. This process is then repeated for all 58 thermocouple locations of the three experimental fire tests. Calibration-corrected results are obtained using the indicated flux q_I and Eq. 7 to calculate the best estimate of the applied flux q_A for all thermocouple locations and fire tests. The uncertainty in the estimate is quantified by $2S_A$ (Eq. 6) with respect to the actual heat flux and by $2S_W$ with respect to an 11-second window average of the actual heat flux.

Figure 10a shows the SODDIT-indicated outer surface temperature $T_{o,I}$ calculated using the measured inner surface temperature $T_{i,M}$. The Curie time period and temperature range are shown. The instability of SODDIT during the Curie transition is once again seen by the oscillations of the outer surface temperature during the Curie time period.

We see that for the same thermocouple location and fire test, the inner and outer surface temperatures of Figs. 4b and 10a are not the same. The initial temperature rise in Fig. 4b is not as steep as that of Fig. 10a. The Curie time periods are of different duration, and the final local temperatures are not the same. These differences are expected as the CAFE simulations were benchmarked based on average calorimeter temperature rise and did not predict the local temperature behavior exactly. However, based on the agreement of the SODDIT-indicated and CAFE-calculated outer surface temperatures of Figure 4b, the temperature prediction $T_{o,I}$ of Fig. 10a is expected to very closely resemble the actual temperature.

Figure 10b shows the SODDIT-indicated heat flux q_I (calculated using the measured inner surface temperature $T_{i,M}$ of Fig. 10a) and the adjusted best estimate for the unknown applied heat flux q_A. Results during the Curie time period are shown, but are not considered valid estimations of the heat flux during that time. As indicated by the sharp initial temperature rise of Fig. 10a, the heat flux is highest at the beginning of the fire as the cool calorimeter was being heated. Thereafter, the flux decreased as the calorimeter approached thermal equilibrium with the fire.

An error bar labeled $\pm 2S_A$ shows with 95% confidence the uncertainty range for the applied heat flux outside the Curie time period. Another error bar labeled $\pm 2S_W$

shows with 95% confidence the uncertainty range for an 11 second window average of the applied heat flux.

The SODDIT-indicated flux q_I and applied flux q_A are nearly identical. This is expected as the calibration equation (Eq. 7) is very close to the ideal response (a=1, b=0). In Fig. 4a we saw that unknown fluxes are well predicted, yet their oscillations are highly damped by SODDIT. Thus, the uncertainty range in the applied flux can be viewed as an indication of the amplitude of the oscillations of the unknown flux about the estimated flux q_A. This result allows us to visualize the behavior of the unknown flux while only estimating its average behavior. The unknown flux would typically closely resemble the SODDIT-indicated flux in trend, but with high frequency oscillations of amplitude near $2S_A$ (at a 95% confidence level).

Comparing Figures 4 and 10, we see that heat fluxes and temperatures do not agree exactly between simulated and experimental results. This may be due to the method used for benchmarking CAFE based on average calorimeter temperature rise. Information about overall heat absorption is of higher priority than local temperature behavior. Local temperatures are highly sensitive to initial and boundary conditions, which are difficult to specify in an environment with varying wind conditions. Further benchmarking based on local temperatures may improve the agreement of simulated and experimental heat fluxes and temperatures.

CONCLUSIONS

A procedure is documented wherein the heat flux to a large pipe calorimeter in a jet fuel fire is estimated using calibration-corrected results from the SODDIT inverse heat conduction code. The number of future times input parameter, NFT, was optimized at NFT=11. Some data are excluded from the analysis due to the erratic behavior of SODDIT-indicated heat flux during a magnetic property transition known as a Curie transition.

Results show that SODDIT-indicated heat fluxes are in good systematic agreement with benchmarked CAFE heat fluxes. Predicted heat fluxes are within a specified confidence interval. The uncertainty in the unknown applied heat flux is not highly affected by uncertainties in material properties and dimensions. The SODDIT-indicated heat flux is approximately an 11 second window average of the applied heat flux; however, the magnitude of the uncertainty in the calculated flux is an implication toward the magnitude of the oscillations of the actual flux. This allows visualization of the actual flux while only predicting its average behavior.

The maximum heat flux experienced by the calorimeter (by use of Eq. 7) occurred during the beginning of Test 1 and measured q_A=195±37 kW/m^2 (95%). The flux decreased thereafter as the calorimeter approached thermal equilibrium with the fire.

Heat flux data for thermocouple locations and fire tests not shown in this work are available via a website by contacting the authors.

The procedure described in this work may be used as a method for estimating outer surface heat flux and temperatures based on inner surface temperature measurements of a large object engulfed in a jet-fuel fire with varying wind conditions.

Future work should focus on a similar procedure with CAFE simulations benchmarked using an FE model inclusive of axial and azimuthal conduction. This will provide a better estimation of local temperature behavior which will be reflected in the location-dependent heat fluxes calculated in this work.

The material property data, namely the specific heat, may be more accurately measured. If a specific heat can be obtained without the definitive spike as seen in this work, the effect of the Curie transition on the SODDIT calculation may be avoided. This will allow heat fluxes calculated during the Curie time period to be included in the overall uncertainty calculation and will better represent the uncertainty over the full range of fire temperatures.

It may also be beneficial to assess a probability distribution of the CAFE-calculated interior surface temperatures for all three fire tests and 58 thermocouple locations. This will allow us to see how the range of input temperatures is represented by the heat fluxes calculated using the calibration equation. We may also examine the residual error of the SODDIT-indicated heat flux about the calibration equation to find whether the indicated flux varies randomly or systematically over the range of CAFE-calculated heat flux.

Finally, future work may examine the effect of larger measurement uncertainties on the overall uncertainty of the calculation. The magnitude of these uncertainties may change the value of NFT required for the calculation thus the magnitude of the overall uncertainty.

TABLES

Table 1: Parameter 95%-confidence-level fractional and absolute uncertainties of measured properties.

Property	Uncertainty
Volumetric Specific Heat, $w_{\rho c}$	2%
Thermal Conductivity, w_k	4%
Thickness, w_d	4%
Temperature Span Error, S_T	0.76%
Temperature Random Noise, R_T	0.02 K

FIGURES

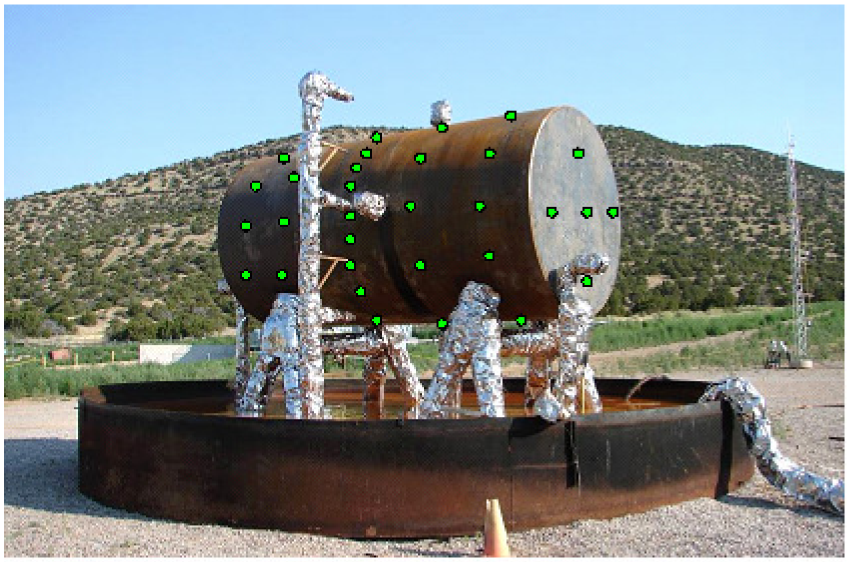


Figure 1: Photo of pipe calorimeter used for fire tests. Dots show locations of some of the 58 inner surface thermocouples located on five rings of the pipe and on both lids.

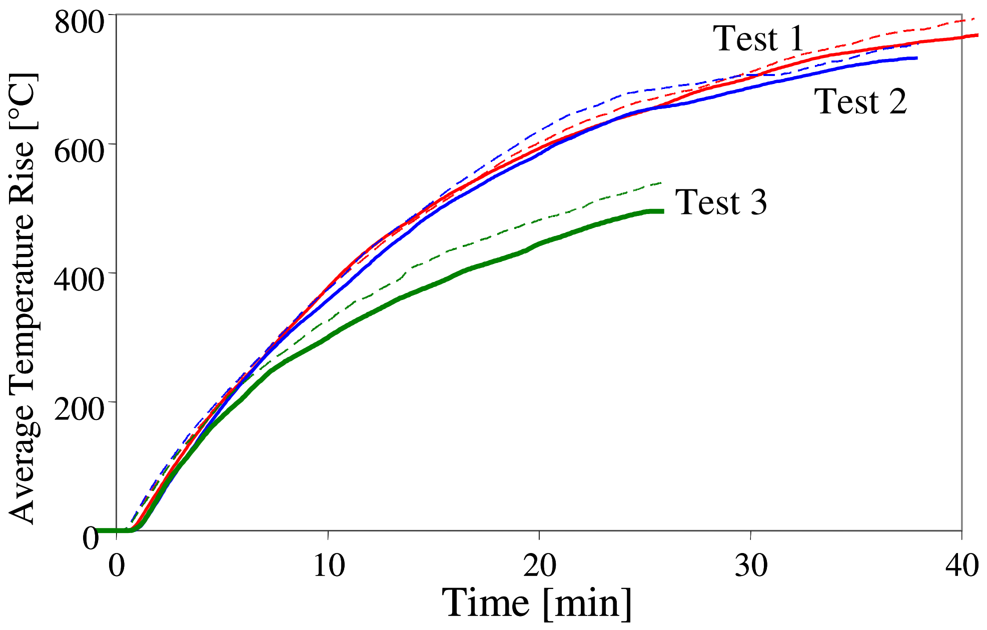


Figure 2: Measured (solid lines) and CAFE-calculated (dashed lines) average calorimeter temperature rise over time for three tests with different wind conditions [4].

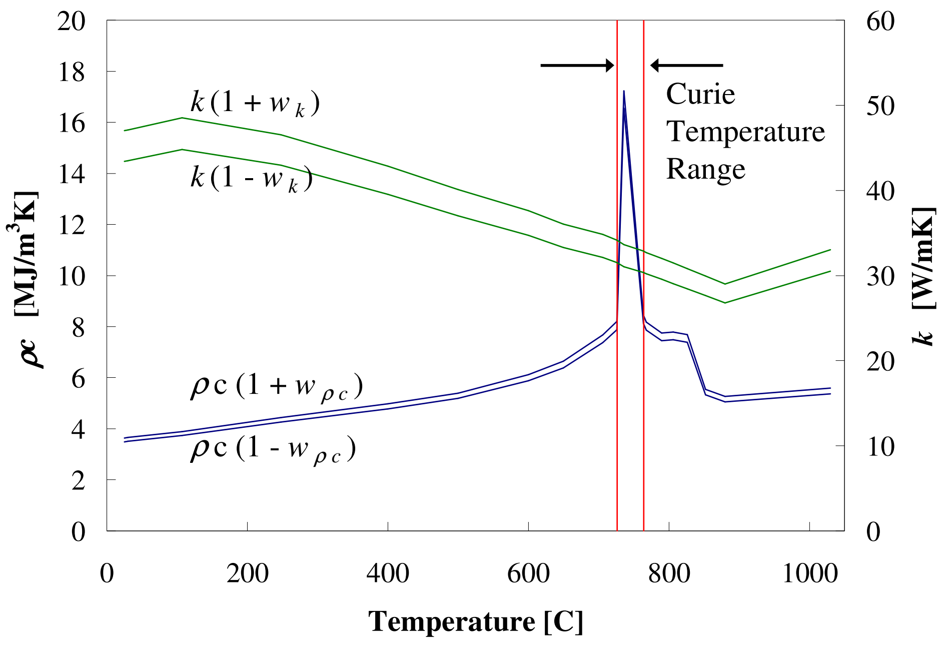
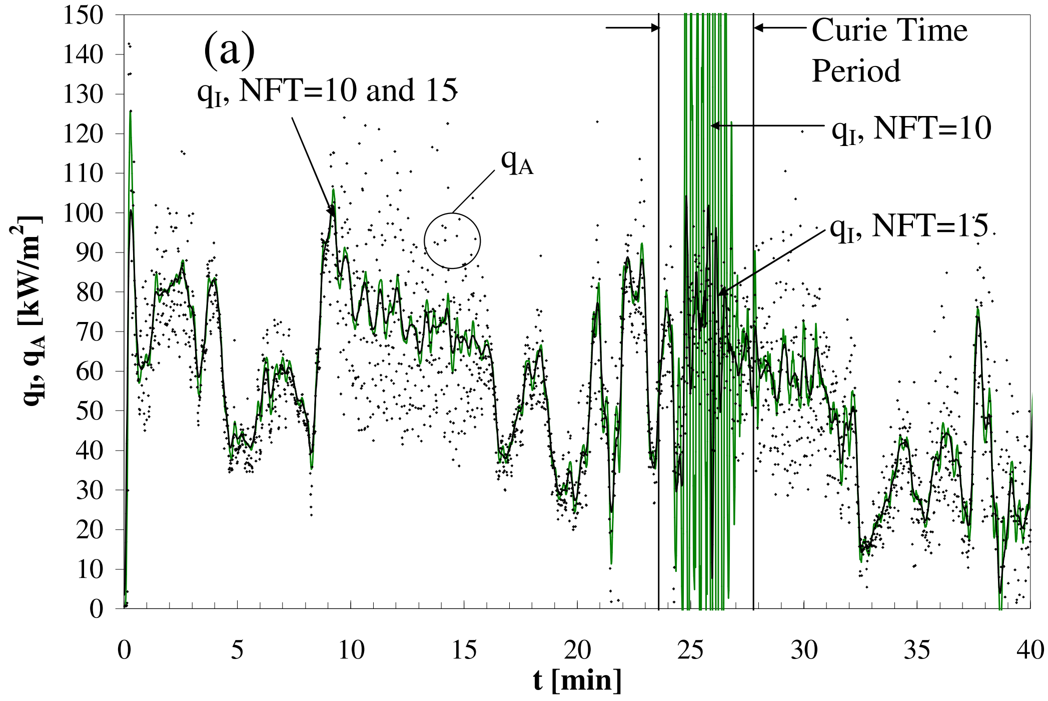
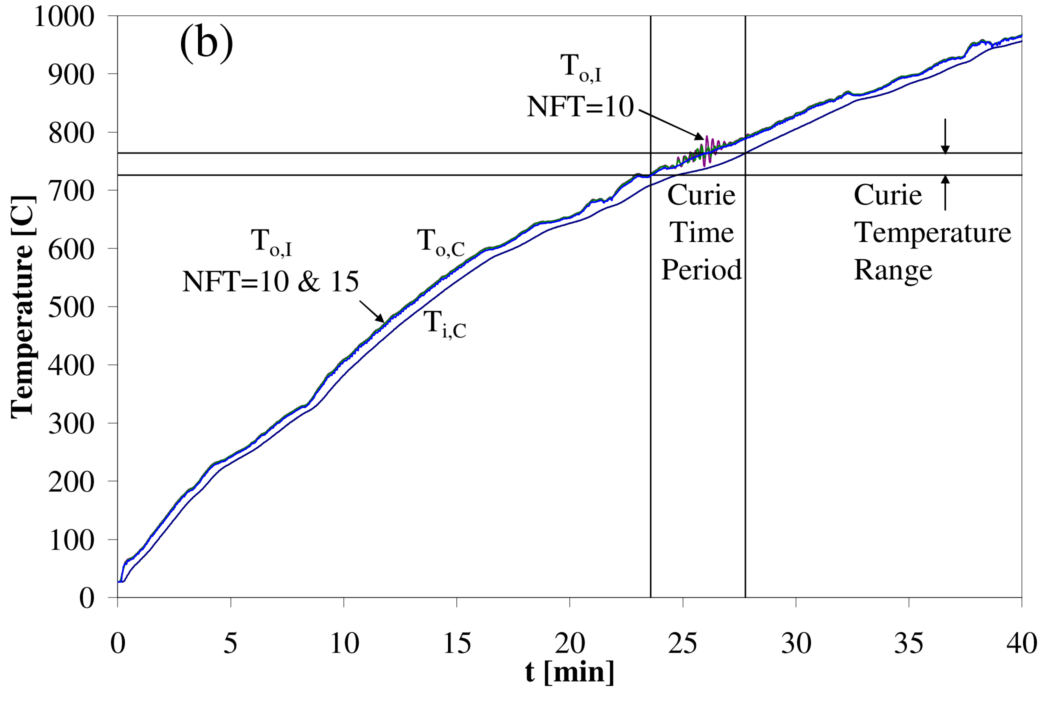


Figure 3: Pipe steel volumetric specific heat and thermal conductivity as functions of temperature. The upper and lower 95% confidence interval and Curie temperature range are shown.


(a)
q_I, NFT=10 and 15
q_A
Curie Time Period
q_I, NFT=10
q_I, NFT=15
q_I, q_A [kW/m^2]
t [min]


(b)
$T_{o,I}$ NFT=10
$T_{o,I}$ NFT=10 & 15
$T_{o,C}$
$T_{i,C}$
Curie Time Period
Curie Temperature Range
Temperature [C]
t [min]

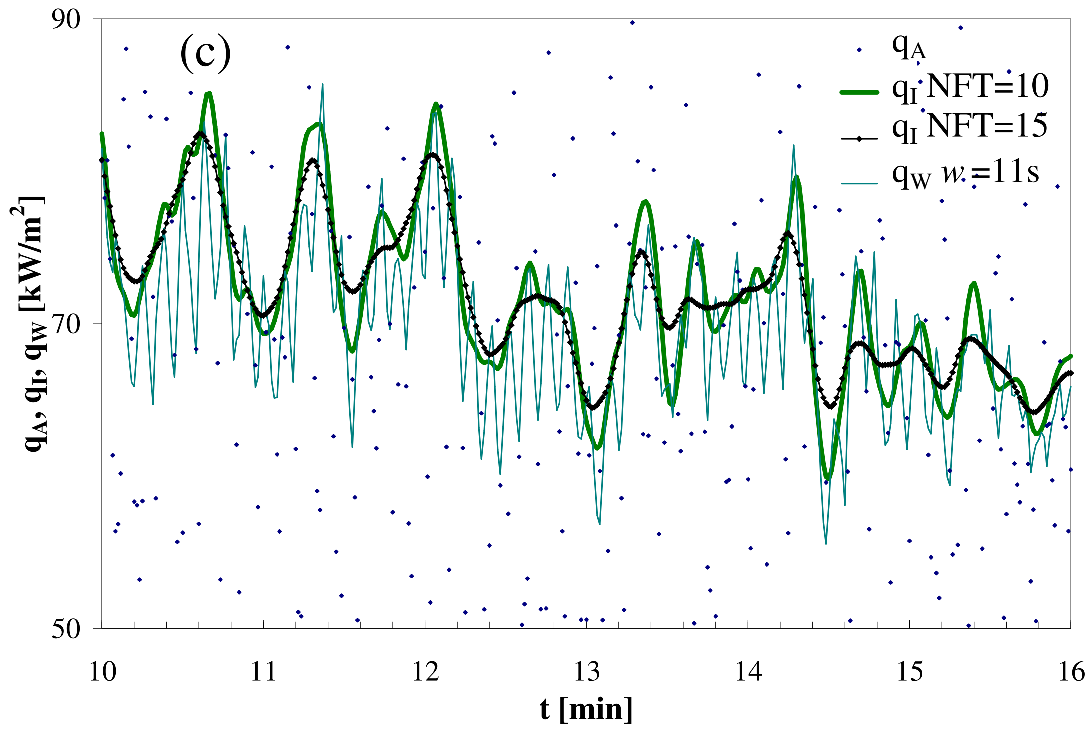


Figure 4: (a) CAFE-applied (dots) and SODDIT-indicated (based on inner surface temperature, NFT=10 and 15) outer surface heat flux vs. time. (b) CAFE inner and outer surface temperatures and SODDIT-indicated outer surface temperatures for NFT=10 and 15 vs. time. (c) Detail of Fig. 4a with window-averaged CAFE heat flux added.

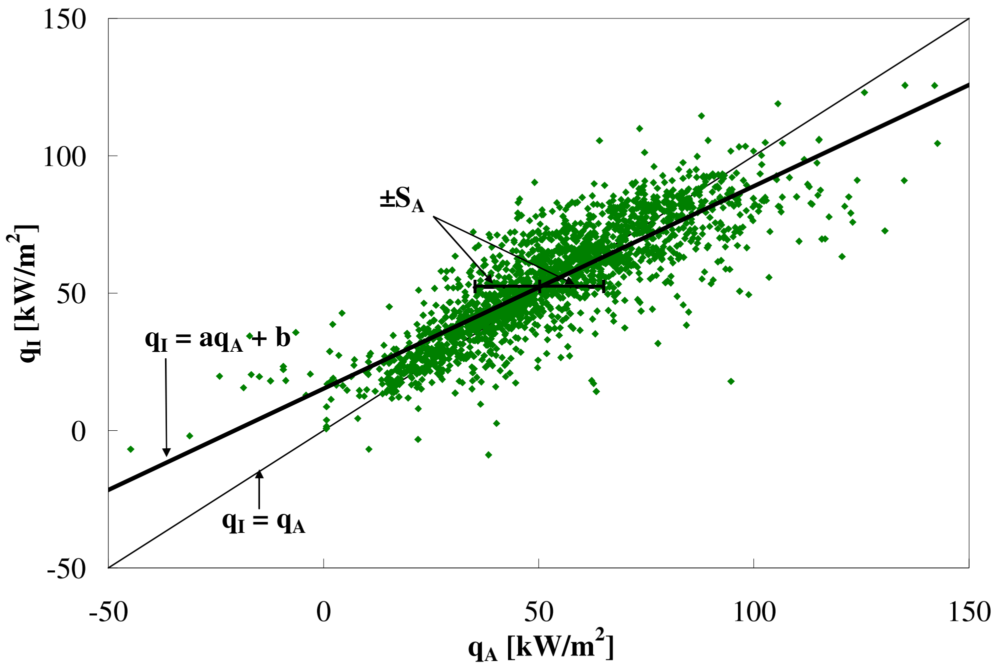


Figure 5: SODDIT-indicated versus CAFE-applied heat flux for the same thermocouple and test as Fig. 4. NFT=10. The ideal response, least squares correlation, and 68% confidence interval for the deviation of the applied flux from the correlation are shown.

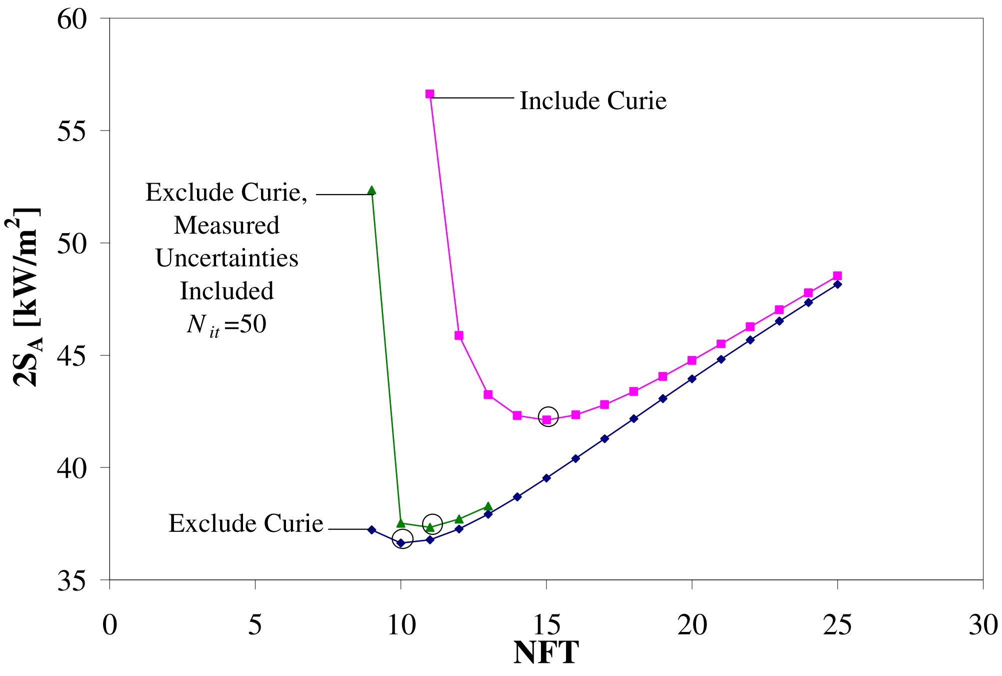


Figure 6: 95% confidence level uncertainty in the applied heat flux for all 58 thermocouples and 3 fire tests. Results including and excluding the Curie time periods, and with and without uncertainties of the measured parameters (Table 1) are shown. The points at which $2S_A$ are at a minimum are circled.

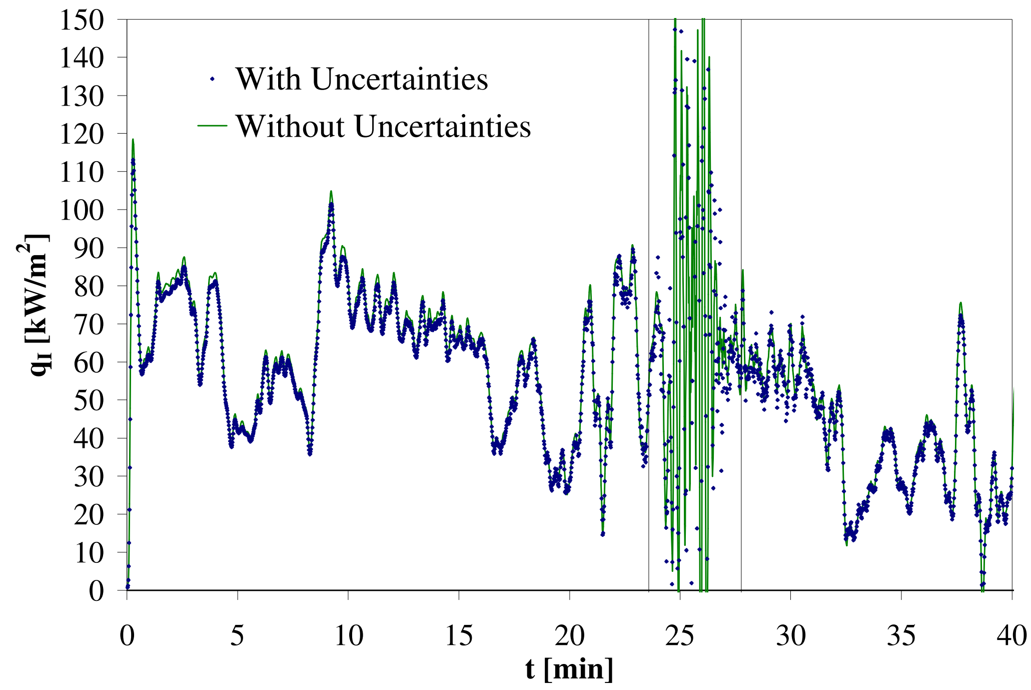


Figure 7: SODDIT-indicated heat flux calculated with and without uncertainties in the measured properties for the same thermocouple location as Fig. 4 (NFT=11).

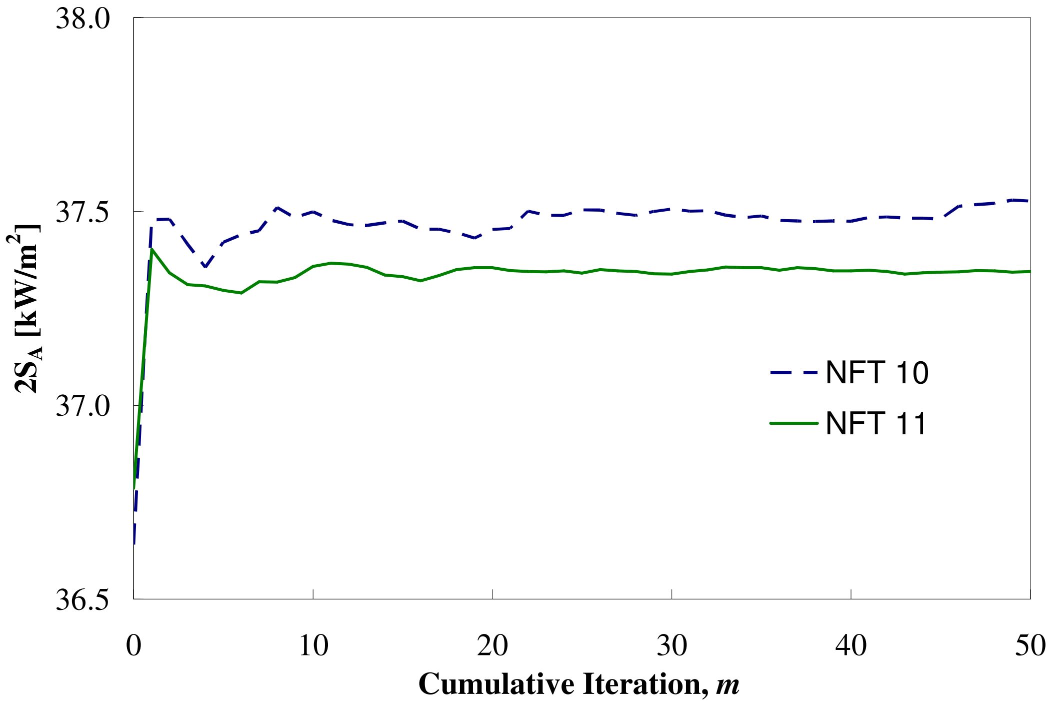


Figure 8: Cumulative 95% confidence level uncertainty in the applied heat flux versus number of cumulative iterations for NFT 10 and 11. Curie data are excluded. Measured uncertainties are included for N_{Ir}>0.

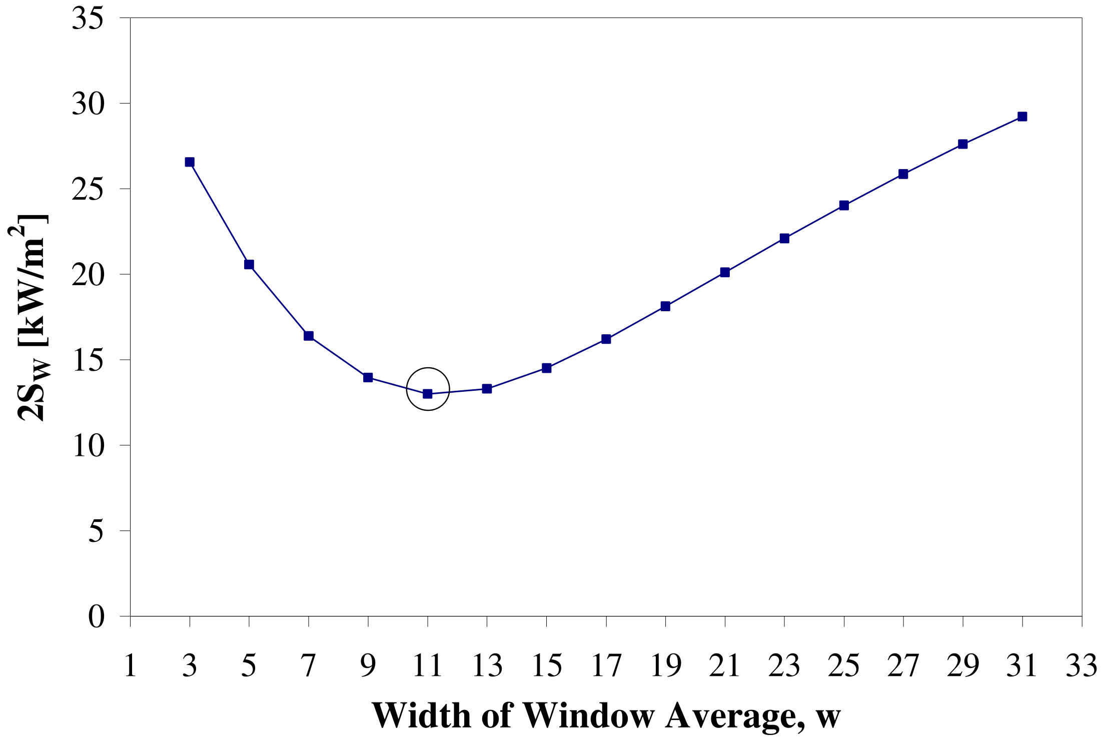


Figure 9: 95% confidence level uncertainty in the window averaged heat flux over a range of window widths (Curie data excluded).

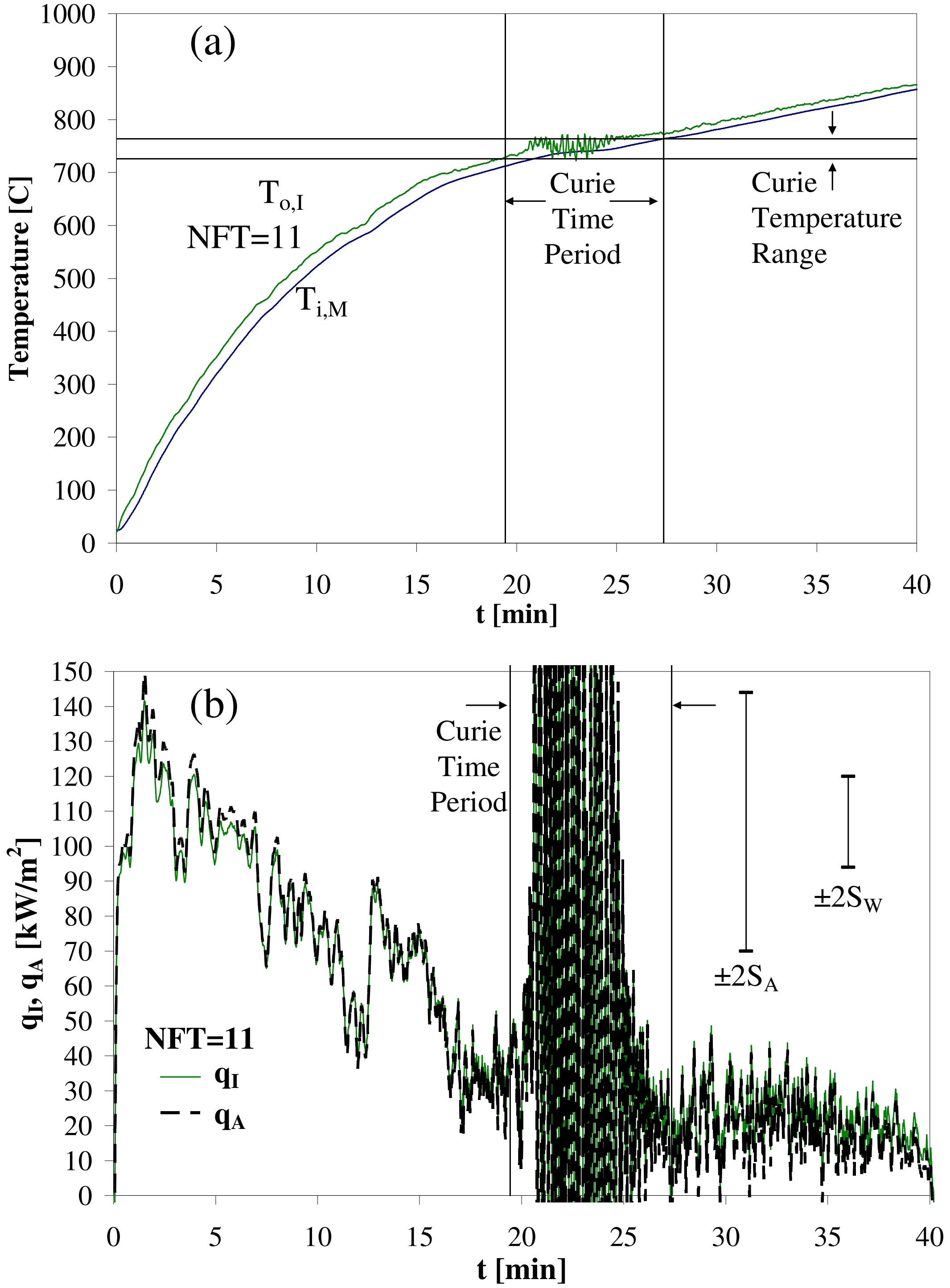


Figure 10: (a) SODDIT-indicated and applied outer heat fluxes for NFT=11, and 95% confidence interval in the applied flux estimated using the experimental inner temperature measurements made by Greiner et al. (2009) for the same thermocouple location and fire test as Fig. 4. (b) Measured inner and SODDIT-indicated outer surface temperatures over time for the same thermocouple location and fire test as Figure 4.

APPENDIX 1 FORTRAN CODE SODDIT_MAIN

```
Program Soddit_Main !This program performs all the basic functions of
this study, including processing the data, calculating the uncertainty
Sa, reading and writing SODDIT input files, and displaying/saving the
results
USE Dflib
!///////////////////////////////////////////////////////////////////////
////////////////////
!//////////////////////Variable
declaration////////////////////////////////////////////////////
!///////////////////////////////////////////////////////////////////////
////////////////////
Integer iteration, test_num, loc, nft
!test_num = the test number (1, 2, or 3)
!loc = the current thermocouple location (1 through 58) (for
thermocouple ID's see excel spreadsheet)
!nft = the current nft for SODDIT
!iteratn = number of times to run tests
Real test1_Ti(2461,59), test1_To(2461,59), test1_q(2461,59) !test 1
input data from CAFE
Real test2_Ti(2281,59), test2_To(2281,59), test2_q(2281,59) !test 2
input data from CAFE
Real test3_Ti(1561,59), test3_To(1561,59), test3_q(1561,59) !test 3
input data from CAFE
Real time_array(2461), temp_array(2461) !arrays for the time and
temperature soddit input data
Real qc_array(2436), qs_array(2436) !arrays representing the heat flux
from cafe and soddit, respectively
Real Toc_array(2436), Tos_array(2436) !arrays representing the outer
temp from cafe and soddit, respectively
Integer num_data(3), num_data_out(3) !Number of input and output data
for each test
Integer tot_rows
Character*1 incl_error !tells whether errors are to be included
Double Precision sx, sxx, sxy, sy, syy
Double Precision sx_ww(31), sxx_ww(31), sxy_ww(31), sy_ww(31),
syy_ww(31), row_count(31)
Double Precision deno_ww(31), a_ww(31), b_ww(31), So_ww(31)
!\\\\\\\\\\\\\\\\\\\\\\End of variable
declaration\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\



!///////////////////////////////////////////////////////////////////////
////////////////////
!//////////////////////Initialize
Values/////////////////////////////////////////////////////
!///////////////////////////////////////////////////////////////////////
////////////////////
num_data(1) = 2461
num_data(2) = 2281
num_data(3) = 1561
```

```
num_data_out(1) = 2435
num_data_out(2) = 2255
num_data_out(3) = 1535
!\\\\\\\\\\\\\\\\\\\\\\End value
initialization\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\



!///////////////////////////////////////////////////////////////////////
////////////////////
!//////////////////////Read input data
files////////////////////////////////////////////////
!///////////////////////////////////////////////////////////////////////
////////////////////
Write(*,*) 'Please wait, reading input data files...'
Write(*,*) ' '
!Read test 1 data
Open(1, file='test1_Ti.txt')
Open(2, file='test1_To.txt')
Open(3, file='test1_q.txt')
Write(*,*) 'test1_Ti.txt'
Do line = 1,num_data(1)
      Read(1,*) (test1_Ti(line,j), j=1,59)
Enddo
Write(*,*) 'test1_To.txt'
Do line = 1,num_data(1)
      Read(2,*) (test1_To(line,j), j=1,59)
Enddo
Write(*,*) 'test1_q.txt'
Do line = 1,num_data(1)
      Read(3,*) (test1_q(line,j), j=1,59)
Enddo
Close(1)
Close(2)
Close(3)

!Read test 2 data
Open(1, file='test2_Ti.txt')
Open(2, file='test2_To.txt')
Open(3, file='test2_q.txt')
Write(*,*) 'test2_Ti.txt'
Do line = 1,num_data(2)
      Read(1,*) (test2_Ti(line,j), j=1,59)
Enddo
Write(*,*) 'test2_To.txt'
Do line = 1,num_data(2)
      Read(2,*) (test2_To(line,j), j=1,59)
Enddo
Write(*,*) 'test2_q.txt'
Do line = 1,num_data(2)
      Read(3,*) (test2_q(line,j), j=1,59)
Enddo
Close(1)
Close(2)
Close(3)

!Read test 3 data
Open(1, file='test3_Ti.txt')
Open(2, file='test3_To.txt')
Open(3, file='test3_q.txt')
Write(*,*) 'test3_Ti.txt'
Do line = 1,num_data(3)
      Read(1,*) (test3_Ti(line,j), j=1,59)
Enddo
Write(*,*) 'test3_To.txt'
Do line = 1,num_data(3)
      Read(2,*) (test3_To(line,j), j=1,59)
Enddo
Write(*,*) 'test3_q.txt'
Do line = 1,num_data(3)
      Read(3,*) (test3_q(line,j), j=1,59)
Enddo
Close(1)
Close(2)
Close(3)
Write(*,*) "Done!"
!\\\\\\\\\\\\\\\\\\\\\\End of input data
read\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\


!///////////////////////////////////////////////////////////////////////
////////////////////
!//////////////////////Prompt for soddit run
specifications/////////////////////////////////
!///////////////////////////////////////////////////////////////////////
////////////////////
Write(*,*) ' '
Write(*,*) 'Please specify the number of iterations to run tests 1, 2,
and 3:'
Write(*,FMT = '(A)', advance = 'no') '(Input only one value) '
Read(*,*) iteration
Write(*,*) ' '
Write(*,FMT = '(A)', advance = 'no') 'Include material property and
temperature errors? (y/n)  '
Read(*,*) incl_error
!\\\\\\\\\\\\\\\\\\\\\\\End
prompt\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\


!///////////////////////////////////////////////////////////////////////
////////////////////
!///////////////////////Call
soddit/////////////////////////////////////////////////////////
!///////////////////////////////////////////////////////////////////////
////////////////////
Write(*,*) ' '
```

```
Open(1,file='iter_analysis.txt')
!Write(*,*) 'Please set initial values for sx, sxx, sxy, sy, syy, and
tot_rows.'
!Write(*,*) 'For a new data set, initialize all values to zero.'
!Write(*,*) ' '
!Read(*,*) sx, sxx, sxy, sy, syy, tot_rows
!Write(*,*) ' '
Write(*,*) 'Press ENTER to begin simulations...'
Read(*,*)
Write(1,*) 'Iteration, a, b, So, sx, sxx, sxy, sy, syy, tot_rows'
Do nft = 15,15

!     sx  = 0.
!     sxx = 0.
!     sxy = 0.
!     sy  = 0.
!     syy = 0.
!     tot_rows = 0.

      Do iteratn = 1,iteration !Calculate a cumulative universal So for
each iteration
            Do test_num = 1,1
                  Do loc = 17,17
                        !Create input time and temperature arrays
                        If (test_num .eq. 1) Then

                              Do i=1,num_data(1)
                                    time_array(i) = test1_Ti(i,1)
                                    temp_array(i) = test1_Ti(i,1+loc)
                              Enddo
                              Do i=1,num_data_out(1)
                                    qc_array(1+i) = test1_q(1+i,1+loc)
                                    Toc_array(1+i) = test1_To(1+i,1+loc)
                              Enddo

                        Else if (test_num .eq. 2) Then

                              Do i=1,num_data(2)
                                    time_array(i) = test2_Ti(i,1)
                                    temp_array(i) = test2_Ti(i,1+loc)
                              Enddo
                              Do i=1,num_data_out(2)
                                    qc_array(1+i) = test2_q(1+i,1+loc)
                                    Toc_array(1+i) = test2_To(1+i,1+loc)
                              Enddo

                        Else if (test_num .eq. 3) Then

                              Do i=1,num_data(3)
                                    time_array(i) = test3_Ti(i,1)
                                    temp_array(i) = test3_Ti(i,1+loc)
                              Enddo
                              Do i=1,num_data_out(3)
                                    qc_array(1+i) = test3_q(1+i,1+loc)
                                    Toc_array(1+i) = test3_To(1+i,1+loc)
                              Enddo

                        Endif

                        Call run_soddit(iteratn, test_num, loc, nft,
num_data, time_array, temp_array, incl_error)
                        Call Read_Output(Tos_array, qs_array, test_num,
num_data_out)

                        tot_rows = tot_rows + num_data_out(test_num)
!tally the total data rows for error analysis

                        !Calculate error (So and fit line parameters a,
b)
                        Do i=2,(num_data_out(test_num)+1)
                              IF ((Toc_array(i).GT.999 .AND.
Toc_array(i).LT.1037) .OR. (temp_array(i).GT.999 .AND.
temp_array(i).LT.1037)) Then
                                    tot_rows = tot_rows - 1 !This IF
statement must be commented out to include the curie region, otherwise
curie is excluded
                              Else
                                    x = qc_array(i)
                                    y = qs_array(i)
                                    sx  = sx  + x     !linear regression
equation*
                                    sxx = sxx + x*x   !See Wheeler and
Ganji, Intro to Engineering Experimentation
                                    sxy = sxy + x*y   !second edition
page 156
                                    sy  = sy  + y
                                    syy = syy + y*y
                              Endif

                        Enddo

                        !Calculate window width data
                        !Do ii=3,31 !must be equal to length of
sx_ww(ii) etc. matrices
                        !     IF(ii/2.0 .NE. INT(ii/2)) Then
                        !           !Write(*,*) ii
                        !           ww_start = INT(1/2.0*(ii+1))
!beginning row for ww calculations
                        !           window_size = INT(1/2.0*(ii-1))
!window size depending on current window width
                        !           ww_end = num_data_out(test_num)-
window_size !ending row for ww calculation
                        !
                        !           Do jj=ww_start,ww_end
                        !                 IF ((Toc_array(jj).GT.999
.AND. Toc_array(jj).LT.1037) .OR. (temp_array(jj).GT.999 .AND.
temp_array(jj).LT.1037)) Then !This IF statement must be commented out
to include the curie region, otherwise curie is excluded
```

```
                        !                       !Do nothing if inside 
curie region
                        !                 Else
                        !                       x_ww = qs_array(jj)
                        !                       q_ww = 0.

                        !                       Do kk = jj-
window_size,jj+window_size
                        !                             q_ww = q_ww + 
qc_array(kk)
                        !                       Enddo
                        !                       y_ww = q_ww/ii
                        !                       sx_ww(ii) = sx_ww(ii) + 
x_ww
                        !                       sxx_ww(ii) = sxx_ww(ii) 
+ x_ww*x_ww
                        !                       sxy_ww(ii) = sxy_ww(ii) 
+ x_ww*y_ww
                        !                       sy_ww(ii) = sy_ww(ii) + 
y_ww
                        !                       syy_ww(ii) = syy_ww(ii) 
+ y_ww*y_ww
                        !                       row_count(ii) = 
row_count(ii)+1
                        !                 Endif
                        !           Enddo
                        !     Endif
                        !Enddo


                  Enddo
            Enddo

            !Error analysis
            deno = tot_rows*sxx - sx*sx               !* continued, next 
4 lines may be commented out if window width data is being calculated
            a    = (tot_rows*sxy - sx*sy)/deno
            b    = (sxx*sy - sx*sxy)/deno
            So   = ((syy - b*sy - a*sxy)/(tot_rows-2.0))**(0.5)

            !Write(1,*) iteratn, a, b, So, sx, sxx, sxy, sy, syy, 
tot_rows !save data in case the program fails, must be commented out if 
window width data is being calculated
            Write(*,*) iteratn, a, b, So

            !Window width analysis
            !Do ii=3,31
            !     IF(ii/2.0 .NE. INT(ii/2)) Then
            !           deno_ww(ii) = row_count(ii)*sxx_ww(ii) - 
sx_ww(ii)*sx_ww(ii)
            !           a_ww(ii) = (row_count(ii)*sxy_ww(ii) - 
sx_ww(ii)*sy_ww(ii))/deno_ww(ii)
            !           b_ww(ii) = (sxx_ww(ii)*sy_ww(ii) - 
sx_ww(ii)*sxy_ww(ii))/deno_ww(ii)
```

```
            !           So_ww(ii) = ((syy_ww(ii) - b_ww(ii)*sy_ww(ii) -
a_ww(ii)*sxy_ww(ii))/(row_count(ii) - 2.0))**(0.5)
            !           Write(1,*) ii, a_ww(ii), b_ww(ii), So_ww(ii)
            !     Endif
            !Enddo



      Enddo
Enddo
Close(1)
!\\\\\\\\\\\\\\\\\\\\\\\\End of Soddit
Call\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

EndProgram Soddit_Main





REAL Function normrand()
      USE DFLIB
      !Generate values in the standard normal distribution, aka
      !values with a mean mu = 0 and standard deviation sigma = 1.0,
      !i.e. the range (-1,1) @ 68% confidence.
      !This is also known as the "polar method."
      Real fac, rsq, v1, v2, gset, a, b
      Logical flag
      If (flag) Then
            normrand = gset
            flag = .false.
      Else
            rsq = 2. !Initialize rsq so the following loop runs at least
once
            Do While (rsq .GT. 1.)
                  Call Random_Number(a)
                  Call Random_Number(b)
                  v1 = 2*a - 1.
                  v2 = 2*b - 1.
                  rsq = v1*v1 + v2*v2
            Enddo
            fac = sqrt(-2 * log(rsq) / rsq)
            normrand = v2 * fac
            gset = v1 * fac
            flag = .true.
      Endif
End Function




!Perform necessary tasks and runs soddit
Subroutine run_soddit(iteratn, test_num, loc, nft, num_data, time_array,
temp_array, incl_error)
      !See above for these variable descriptions
      Character*1 incl_error
      Integer iteratn, test_num, loc, nft, num_data(3), num_data_out(3)
      Real matprops(22,3), time_array(2461), temp_array(2461)
      Real normrand

      !Declare error values
      !cp = specific heat error (1 sigma percentage)
      !k = conductivity error (1 sigma percentage)
      !rho = density error (1 sigma percentage)
      !w = thickness error (1 sigma percentage)
      !rtemp_error = random Temperature error (1 sigma Kelvin)
      !stemp_error = Temperature span error (1 sigma percentage)
      !btemp_error = Temperature bias error (1 sigma Kelvin)
      Real cp_error, k_error, rho_error, w_error, rtemp_error,
stemp_error, btemp_error
      cp_error = 0.01
      k_error = 0.02
      rho_error = 0.0
      w_error = 0.02
      rtemp_error = 0.01
      stemp_error = 0.0038
      btemp_error = 0.

      !Delete the previous run output files, should be performed
      !some time before the run soddit command
      If (test_num .eq. 1) Then
            deletepreviousoutputfiles = DELFILESQQ('test1_out.txt')
            deletepreviousoutputfiles = DELFILESQQ('test1_plt.txt')
      Else if (test_num .eq. 2) Then
            deletepreviousoutputfiles = DELFILESQQ('test2_out.txt')
            deletepreviousoutputfiles = DELFILESQQ('test2_plt.txt')
      Else if (test_num .eq. 3) Then
            deletepreviousoutputfiles = DELFILESQQ('test3_out.txt')
            deletepreviousoutputfiles = DELFILESQQ('test3_plt.txt')
      End if

      !Get the material property data and add errors if necessary
      Open(4,file='matprops.txt')
      Read(4,*) matprops(1,1) !Read the density
      Do row = 2,21 !Read the temperature, cp, conductivity data
            Read(4,*) (matprops(row,j), j=1,3)
      Enddo
      Read(4,*) matprops(22,1)!Read the thickness
      Close(4)
      Call Random_Seed()
      If (incl_error .eq. 'y') Then !Add errors
            r = normrand()
            s = normrand()
            t = normrand()
            matprops(1,1) = matprops(1,1)*(1. + rho_error*r) !density
            Do i=2,21
                  matprops(i,2) = matprops(i,2)*(1. + cp_error*s) !cp
                  matprops(i,3) = matprops(i,3)*(1. + k_error*t)
!conductivity
            Enddo
            matprops(22,1) = matprops(22,1)*(1 + w_error*normrand())
!thickness
            a = normrand()
            b = normrand()
            Do i=1,num_data(test_num) !temperature
                  temp_array(i) = temp_array(i) + (temp_array(i)-
273)*(stemp_error*a) + btemp_error*b + rtemp_error*normrand()
            Enddo
      End if

      !Write the soddit input file
      If (test_num .eq. 1) Then
            Open(3,file='test1.txt', form = 'formatted')
      Else if (test_num .eq. 2) Then
            Open(3,file='test2.txt', form = 'formatted')
      Else if (test_num .eq. 3) Then
            Open(3,file='test3.txt', form = 'formatted')
      End if
      Write(3,*) '**block     1     control     flags'
      Write(3,*) 'c23456789   123456789   123456789   123456789
      123456789   123456789   123456789   123456789'
      Write(3,*) '0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     2'
      Write(3,*) 'cal48.inp   Cylindrical shell receiving   unknown
      heat  flux  to    outside     "surface,"'
      Write(3,*) 'given temperature versus      time  at    inside
      "surface,"  soddit      is    to    determine'
      Write(3,*) 'the   heat  flux  to    the   outside     surface.'
      Write(3,*) '**block     3     print times and   print "intervals,"
maximum     of    20    and   "19," respectively'
      Write(3,*) '**tprnt(1)  tprnt(2)    tprnt(3)'
      Write(3,*) '-235  5000'
      Write(3,*) '**dtprnt(1) dtprnt(2)'
      Write(3,*) '1'
      Write(3,*) '**block     4     general     problem     constants'
      Write(3,*) '**dtmin     dtmax theta dtempm      sigma radius
      expn  fract1      fract2'
      Write(3,*) '0.01  0     1     0     5.67E-08    1.2192      1'
      Write(3,*) '**block     5     ***material property    data'
      Write(3,*) '**rho dhf   treff qstar tabl'
      Write(3,*) matprops(1,1) !density
      Write(3,*) '**temp      cp    cond  emit  absrp'
      Do i = 2,21 !material property data
            If (i .eq. 21) Then
                  Write(3,10, advance = 'no') '-1' !-1 to show end of
data table
                  10 Format (A)
                  Write(3,*) (matprops(i,j), j=1,3)
            Else
                  Write(3,*) (matprops(i,j), j=1,3)
            Endif
      Enddo
      Write(3,*) '**rho dhf   treff qstar tabl'
      Write(3,*) '86    0     0                 '
      Write(3,*) '**temp      cp    cond  emit  absrp'
      Write(3,*) '300   800   0.72702                        '
      Write(3,*) '500   800   0.72702                        '
      Write(3,*) '1000  800   1.5059                         '
      Write(3,*) '1500  800   2.562                    '
      Write(3,*) '1800  800   3.272                    '
      Write(3,10, advance='no') '-1' ! -1 to show end of data table
      Write(3,*) '2000  800   3.756             '
      Write(3,*) '/end  of    block 5     material    property
      tables'
      Write(3,*) '**block     6     automatic   node  generation  data'
      Write(3,*) '**initial   temperature'
      Write(3,*) '300'
      Write(3,*) '**(mat      no)   (elements/region) (region
      thickness)(tck    1st   elem)(tck   last  elem)'
      Write(3,10, advance = 'no') ' 1     20    '
      Write(3,FMT = '(F8.7)', advance = 'no') matprops(22,1) !thickness
here
      Write(3,*) '      0     0'
      Write(3,*) '      2     5     0.0762      0     0'
      Write(3,*) '/end  of    block 6     automatic   node  generation'
      Write(3,*) '**block     7     front face  node  generation  data'
      Write(3,*) '**ibcty1    ifmt1 trad1'
      Write(3,*) '4     0'
      Write(3,*) '**nft ndttq ifst  iskip "ninv(i),i+1,ninvn"'
      Write(3,*) nft, 1, 1, 1, 21 !nft specified here
      Write(3,*) '**time      y(i)'
      Do i = 1,num_data(test_num)
            Write(3,*) time_array(i), temp_array(i) !time temperature
data here
      Enddo
      Write(3,*) '/end  of    block 7'
      Write(3,*) '**block     8     back  face  boundary    condition
      data'
      Write(3,*) '**ibctyn    (perfectly  insulated   back  face
      boundary    condition)'
      Write(3,*) '0'

      Close(3)

!Run soddit
If (test_num .eq. 1) Then
      test = runqq('soddit.exe','test1.txt')
Else if (test_num .eq. 2) Then
      test = runqq('soddit.exe','test2.txt')
Else if (test_num .eq. 3) Then
      test = runqq('soddit.exe','test3.txt')
End If

EndSubroutine



Subroutine Read_Output(Tos_array, qs_array, test_num, num_data_out)
```

```
      Use DFPORT
      Real Tos_array(2436), qs_array(2436), time_junk
      Integer test_num, num_data_out(3), junk(3)
      Character*80 out_junk
      Character*13 filename1
      Logical fileexists

      junk(1) = 2630 !number of lines to be read before reaching the
output data in soddit file
      junk(2) = 2450
      junk(3) = 1730

      If (test_num .eq. 1) Then
            filename1='test1_out.txt'
      Else if (test_num .eq. 2) Then
            filename1='test2_out.txt'
      Else if (test_num .eq. 3) Then
            filename1='test3_out.txt'
      Endif

      INQUIRE(file=filename1, exist = fileexists)

      Do While (fileexists .eq. .false.)
            Call SLEEP(1)
            INQUIRE(file=filename1, exist = fileexists)
      Enddo

      Open(10,file=filename1)

      Do k=1,junk(test_num)
            Read(10,30) out_junk
            30 Format (A)
      Enddo

      Do k=1,num_data_out(test_num)
            Read(10,*) time_junk, Tos_array(1+k), qs_array(1+k)
      Enddo
      Close(10)

EndSubroutine
```

APPENDIX 2 FORTRAN CODE SODDIT_EXPERIMENTAL_DATA

```
Program Soddit_Main !A variation of Soddit_Main is used to run the
experimental data through soddit
USE Dflib
!////////////////////////////////////////////////////////////////////////
////////////////////
!//////////////////////Variable
declaration/////////////////////////////////////////////////
!////////////////////////////////////////////////////////////////////////
////////////////////
Integer iteration, test_num, loc, nft
!test_num = the test number (1, 2, or 3)
!loc = the current thermocouple location (1 through 58) (for
thermocouple ID's see excel spreadsheet)
!nft = the current nft for SODDIT
!iteratn = number of times to run tests
Real test1_Ti(2461,59)!, test1_To(2461,59), test1_q(2461,59) !test 1
input data from CAFE
Real test2_Ti(2281,59)!, test2_To(2281,59), test2_q(2281,59) !test 2
input data from CAFE
Real test3_Ti(1561,59)!, test3_To(1561,59), test3_q(1561,59) !test 3
input data from CAFE
Real time_array(2461), temp_array(2461) !arrays for the time and
temperature soddit input data
Real qc_array(2436), qs_array(2436) !arrays representing the heat flux
from cafe and soddit, respectively
Real Toc_array(2436), Tos_array(2436) !arrays representing the outer
temp from cafe and soddit, respectively
Integer num_data(3), num_data_out(3) !Number of input and output data
for each test
Integer tot_rows
Character*1 incl_error !tells whether errors are to be included
Double Precision sx, sxx, sxy, sy, syy
Real To_array(2436,58), q_array(2436,58)
!\\\\\\\\\\\\\\\\\\\\\\End of variable
declaration\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\



!////////////////////////////////////////////////////////////////////////
////////////////////
!//////////////////////Initialize
Values///////////////////////////////////////////////////
!////////////////////////////////////////////////////////////////////////
////////////////////
num_data(1) = 2461
num_data(2) = 2281
num_data(3) = 1561
num_data_out(1) = 2435
num_data_out(2) = 2255
num_data_out(3) = 1535
```

```
!\\\\\\\\\\\\\\\\\\\\\\End value
initialization\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\



!///////////////////////////////////////////////////////////////////////
////////////////////
!//////////////////////Read input data
files////////////////////////////////////////////////
!///////////////////////////////////////////////////////////////////////
////////////////////
Write(*,*) 'Please wait, initializing input data files...'
Write(*,*) ' '
!Read test 1 data
Open(1, file='test1_Ti.txt')
!Open(2, file='test1_To.txt')
!Open(3, file='test1_q.txt')
Write(*,*) 'test1_Ti.txt'
Do line = 1,num_data(1)
      Read(1,*) (test1_Ti(line,j), j=1,59)
Enddo
!Write(*,*) 'test1_To.txt'
!Do line = 1,num_data(1)
!     Read(2,*) (test1_To(line,j), j=1,59)
!Enddo
!Write(*,*) 'test1_q.txt'
!Do line = 1,num_data(1)
!     Read(3,*) (test1_q(line,j), j=1,59)
!Enddo
Close(1)
!Close(2)
!Close(3)

!Read test 2 data
Open(1, file='test2_Ti.txt')
!Open(2, file='test2_To.txt')
!Open(3, file='test2_q.txt')
Write(*,*) 'test2_Ti.txt'
Do line = 1,num_data(2)
      Read(1,*) (test2_Ti(line,j), j=1,59)
Enddo
!Write(*,*) 'test2_To.txt'
!Do line = 1,num_data(2)
!     Read(2,*) (test2_To(line,j), j=1,59)
!Enddo
!Write(*,*) 'test2_q.txt'
!Do line = 1,num_data(2)
!     Read(3,*) (test2_q(line,j), j=1,59)
!Enddo
Close(1)
!Close(2)
!Close(3)

!Read test 3 data
```

```
Open(1, file='test3_Ti.txt')
!Open(2, file='test3_To.txt')
!Open(3, file='test3_q.txt')
Write(*,*) 'test3_Ti.txt'
Do line = 1,num_data(3)
      Read(1,*) (test3_Ti(line,j), j=1,59)
Enddo
!Write(*,*) 'test3_To.txt'
!Do line = 1,num_data(3)
!     Read(2,*) (test3_To(line,j), j=1,59)
!Enddo
!Write(*,*) 'test3_q.txt'
!Do line = 1,num_data(3)
!     Read(3,*) (test3_q(line,j), j=1,59)
!Enddo
Close(1)
!Close(2)
!Close(3)
Write(*,*) "Done!"
!\\\\\\\\\\\\\\\\\\\\\\\End of input data
read\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\



!////////////////////////////////////////////////////////////////////////
////////////////////
!///////////////////////Prompt for soddit run
specifications/////////////////////////////////
!////////////////////////////////////////////////////////////////////////
////////////////////
Write(*,*) ' '
Write(*,*) 'Please specify the number of iterations to run tests 1, 2,
and 3:'
Write(*,FMT = '(A)', advance = 'no') '(Input only one value) '
Read(*,*) iteration
Write(*,*) ' '
Write(*,FMT = '(A)', advance = 'no') 'Include material property and
temperature errors? (y/n)  '
Read(*,*) incl_error
!\\\\\\\\\\\\\\\\\\\\\\\\End
prompt\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\



!////////////////////////////////////////////////////////////////////////
////////////////////
!////////////////////////Call
soddit/////////////////////////////////////////////////////////
!////////////////////////////////////////////////////////////////////////
////////////////////
nft = 11
Write(*,*) ' '
!Open(1,file='iter_analysis.txt')
```

```
!Write(*,*) 'Please set initial values for sx, sxx, sxy, sy, syy, and
tot_rows.'
!Write(*,*) 'For a new data set, initialize all values to zero.'
!Write(*,*) ' '
!Read(*,*) sx, sxx, sxy, sy, syy, tot_rows
!Write(*,*) ' '
Write(*,*) 'Press ENTER to begin simulations...'
Read(*,*)
!Write(1,*) 'Iteration, a, b, So, sx, sxx, sxy, sy, syy, tot_rows'
Do iteratn = 1,iteration !Calculate a cumulative universal So for each
iteration
      Do test_num = 1,3

            IF (test_num .eq. 1) Then
                  Open(20, file='test1_Tos.txt')
                  Open(21, file='test1_qs.txt')
            Elseif (test_num .eq. 2) Then
                  Open(20, file='test2_Tos.txt')
                  Open(21, file='test2_qs.txt')
            Elseif (test_num .eq. 3) Then
                  Open(20, file='test3_Tos.txt')
                  Open(21, file='test3_qs.txt')
            Endif

            Do loc = 17,17
                  !Create input time and temperature arrays
                  If (test_num .eq. 1) Then

                        Do i=1,num_data(1)
                              time_array(i) = test1_Ti(i,1)
                              temp_array(i) = test1_Ti(i,1+loc)
                        Enddo
                        !Do i=1,num_data_out(1)
                        !     qc_array(1+i) = test1_q(1+i,1+loc)
                        !     Toc_array(1+i) = test1_To(1+i,1+loc)
                        !Enddo

                  Else if (test_num .eq. 2) Then

                        Do i=1,num_data(2)
                              time_array(i) = test2_Ti(i,1)
                              temp_array(i) = test2_Ti(i,1+loc)
                        Enddo
                        !Do i=1,num_data_out(2)
                        !     qc_array(1+i) = test2_q(1+i,1+loc)
                        !     Toc_array(1+i) = test2_To(1+i,1+loc)
                        !Enddo

                  Else if (test_num .eq. 3) Then

                        Do i=1,num_data(3)
                              time_array(i) = test3_Ti(i,1)
                              temp_array(i) = test3_Ti(i,1+loc)
                        Enddo
                        !Do i=1,num_data_out(3)
                        !     qc_array(1+i) = test3_q(1+i,1+loc)
                        !     Toc_array(1+i) = test3_To(1+i,1+loc)
                        !Enddo

                  Endif

                  Write(*,*) 'About to run soddit'
                  Call run_soddit(iteratn, test_num, loc, nft, num_data,
time_array, temp_array, incl_error)
                  Call Read_Output(Tos_array, qs_array, test_num,
num_data_out)

                  !tot_rows = tot_rows + num_data_out(test_num) !tally
the total data rows for error analysis

                  !Do i=2,(num_data_out(test_num)+1)
                  !     x = qc_array(i)
                  !     y = qs_array(i)
                  !     sx  = sx  + x     !linear regression equation*
                  !     sxx = sxx + x*x   !See Wheeler and Ganji, Intro
to Engineering Experimentation
                  !     sxy = sxy + x*y   !second edition page 156
                  !     sy  = sy  + y
                  !     syy = syy + y*y
                  !Enddo

                  !Put the output data into a test-wide array
                  Do i=1,num_data_out(test_num)
                        To_array(i,loc) = Tos_array(i+1)
                        q_array(i,loc) = qs_array(i+1)
                  Enddo


            Enddo

            !Write the output data to a file for plotting
            Do i=1,num_data_out(test_num)
                  Write(20,*) i, (To_array(i,j), j=17,17)
                  Write(21,*) i, (q_array(i,j), j=1,58)
            Enddo

            Close(20)
            Close(21)

      Enddo

      !deno = tot_rows*sxx - sx*sx              !*
      !a    = (tot_rows*sxy - sx*sy)/deno
      !b    = (sxx*sy - sx*sxy)/deno
      !So   = ((syy - b*sy - a*sxy)/(tot_rows-2.0))**(0.5)
      !Write(1,*) iteratn, a, b, So, sx, sxx, sxy, sy, syy, tot_rows
!save data in case the program fails
      !Write(*,*) iteratn, a, b, So
```

```
Enddo
Close(1)
Write(*,*) 'Complete.'
!\\\\\\\\\\\\\\\\\\\\\\\End of Soddit
Call\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

EndProgram Soddit_Main




REAL Function normrand()
      USE DFLIB
      !Generate values in the standard normal distribution, aka
      !values with a mean mu = 0 and standard deviation sigma = 1.0,
      !i.e. the range (-1,1) @ 68% confidence.
      !This is also known as the "polar method."
      Real fac, rsq, v1, v2, gset, a, b
      Logical flag
      If (flag) Then
            normrand = gset
            flag = .false.
      Else
            rsq = 2. !Initialize rsq so the following loop runs at least
once
            Do While (rsq .GT. 1.)
                  Call Random_Number(a)
                  Call Random_Number(b)
                  v1 = 2*a - 1.
                  v2 = 2*b - 1.
                  rsq = v1*v1 + v2*v2
            Enddo
            fac = sqrt(-2 * log(rsq) / rsq)
            normrand = v2 * fac
            gset = v1 * fac
            flag = .true.
      Endif
End Function



!Perform necessary tasks and runs soddit
Subroutine run_soddit(iteratn, test_num, loc, nft, num_data, time_array,
temp_array, incl_error)
      !See above for these variable descriptions
      Character*1 incl_error
      Integer iteratn, test_num, loc, nft, num_data(3), num_data_out(3)
      Real matprops(22,3), time_array(2461), temp_array(2461)
      Real normrand

      !Declare error values
      !cp = specific heat error (1 sigma percentage)
      !k = conductivity error (1 sigma percentage)
      !rho = density error (1 sigma percentage)
      !w = thickness error (1 sigma percentage)
      !rtemp_error = random Temperature error (1 sigma Kelvin)
      !stemp_error = Temperature span error (1 sigma percentage)
      !btemp_error = Temperature bias error (1 sigma Kelvin)
      Real cp_error, k_error, rho_error, w_error, rtemp_error,
stemp_error, btemp_error
      cp_error = 0.01
      k_error = 0.02
      rho_error = 0.0
      w_error = 0.02
      rtemp_error = 0.01
      stemp_error = 0.0038
      btemp_error = 0.

      !Delete the previous run output files, should be performed
      !some time before the run soddit command
      If (test_num .eq. 1) Then
            deletepreviousoutputfiles = DELFILESQQ('test1_out.txt')
            deletepreviousoutputfiles = DELFILESQQ('test1_plt.txt')
      Else if (test_num .eq. 2) Then
            deletepreviousoutputfiles = DELFILESQQ('test2_out.txt')
            deletepreviousoutputfiles = DELFILESQQ('test2_plt.txt')
      Else if (test_num .eq. 3) Then
            deletepreviousoutputfiles = DELFILESQQ('test3_out.txt')
            deletepreviousoutputfiles = DELFILESQQ('test3_plt.txt')
      End if

      !Get the material property data and add errors if necessary
      Write(*,*) 'Reading matprops.txt'
      Open(4,file='matprops.txt')
      Read(4,*) matprops(1,1) !Read the density
      Do row = 2,21 !Read the temperature, cp, conductivity data
            Read(4,*) (matprops(row,j), j=1,3)
      Enddo
      Read(4,*) matprops(22,1)!Read the thickness
      Close(4)
      Call Random_Seed()
      If (incl_error .eq. 'y') Then !Add errors
            Write(*,*) 'Generating random errors'
            r = normrand()
            s = normrand()
            t = normrand()
            matprops(1,1) = matprops(1,1)*(1. + rho_error*r) !density
            Do i=2,21
                  matprops(i,2) = matprops(i,2)*(1. + cp_error*s) !cp
                  matprops(i,3) = matprops(i,3)*(1. + k_error*t)
!conductivity
            Enddo
            matprops(22,1) = matprops(22,1)*(1 + w_error*normrand())
!thickness
            a = normrand()
            b = normrand()
            Do i=1,num_data(test_num) !temperature
```

```
                temp_array(i) = temp_array(i) + (temp_array(i)-
273)*(stemp_error*a) + btemp_error*b + rtemp_error*normrand()
          Enddo
      End if

      !Write the soddit input file
      Write(*,*) 'Writing Input text file'
      If (test_num .eq. 1) Then
            Open(3,file='test1.txt', form = 'formatted')
      Else if (test_num .eq. 2) Then
            Open(3,file='test2.txt', form = 'formatted')
      Else if (test_num .eq. 3) Then
            Open(3,file='test3.txt', form = 'formatted')
      End if
      Write(3,*) '**block     1     control     flags'
      Write(3,*) 'c23456789   123456789   123456789   123456789
      123456789   123456789   123456789   123456789'
      Write(3,*) '0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     2'
      Write(3,*) 'cal48.inp   Cylindrical shell receiving   unknown
      heat  flux  to    outside     "surface,"'
      Write(3,*) 'given temperature versus      time  at    inside
      "surface,"  soddit      is    to    determine'
      Write(3,*) 'the   heat  flux  to    the   outside     surface.'
      Write(3,*) '**block     3     print times and   print "intervals,"
maximum     of    20    and   "19," respectively'
      Write(3,*) '**tprnt(1)  tprnt(2)    tprnt(3)'
      Write(3,*) '-235  5000'
      Write(3,*) '**dtprnt(1) dtprnt(2)'
      Write(3,*) '1'
      Write(3,*) '**block     4     general     problem     constants'
      Write(3,*) '**dtmin     dtmax theta dtempm      sigma radius
      expn  fract1      fract2'
      Write(3,*) '0.01  0     1     0     5.67E-08    1.2192      1'
      Write(3,*) '**block     5     ***material property    data'
      Write(3,*) '**rho dhf   treff qstar tabl'
      Write(3,*) matprops(1,1) !density
      Write(3,*) '**temp      cp    cond  emit  absrp'
      Do i = 2,21 !material property data
            If (i .eq. 21) Then
                  Write(3,10, advance = 'no') '-1' !-1 to show end of
data table
                  10 Format (A)
                  Write(3,*) (matprops(i,j), j=1,3)
            Else
                  Write(3,*) (matprops(i,j), j=1,3)
            Endif
      Enddo
      Write(3,*) '**rho dhf   treff qstar tabl'
      Write(3,*) '86    0     0                 '
      Write(3,*) '**temp      cp    cond  emit  absrp'
      Write(3,*) '300   800   0.72702                       '
      Write(3,*) '500   800   0.72702                       '
      Write(3,*) '1000  800   1.5059                        '
      Write(3,*) '1500  800   2.562                   '
      Write(3,*) '1800  800   3.272                   '
      Write(3,10, advance='no') '-1' ! -1 to show end of data table
      Write(3,*) '2000  800   3.756             '
      Write(3,*) '/end  of    block 5     material    property
      tables'
      Write(3,*) '**block     6     automatic   node  generation  data'
      Write(3,*) '**initial   temperature'
      Write(3,*) '300'
      Write(3,*) '**(mat      no)   (elements/region) (region
      thickness)(tck    1st   elem)(tck   last  elem)'
      Write(3,10, advance = 'no') ' 1     20    '
      Write(3,FMT = '(F8.7)', advance = 'no') matprops(22,1) !thickness
here
      Write(3,*) '      0     0'
      Write(3,*) '      2     5     0.0762      0     0'
      Write(3,*) '/end  of    block 6     automatic   node  generation'
      Write(3,*) '**block     7     front face  node  generation  data'
      Write(3,*) '**ibcty1    ifmt1 trad1'
      Write(3,*) '4     0'
      Write(3,*) '**nft ndttq ifst  iskip "ninv(i),i+1,ninvn"'
      Write(3,*) nft, 1, 1, 1, 21 !nft specified here
      Write(3,*) '**time      y(i)'
      Do i = 1,num_data(test_num)
            Write(3,*) time_array(i), temp_array(i) !time temperature
data here
      Enddo
      Write(3,*) '/end  of    block 7'
      Write(3,*) '**block     8     back  face  boundary    condition
      data'
      Write(3,*) '**ibctyn    (perfectly  insulated   back  face
      boundary    condition)'
      Write(3,*) '0'

      Close(3)

!Run soddit
Write(*,*) 'Running Soddit'
If (test_num .eq. 1) Then
      test = runqq('soddit.exe','test1.txt')
Else if (test_num .eq. 2) Then
      test = runqq('soddit.exe','test2.txt')
Else if (test_num .eq. 3) Then
      test = runqq('soddit.exe','test3.txt')
End If

EndSubroutine



Subroutine Read_Output(Tos_array, qs_array, test_num, num_data_out)
      Use DFPORT
      Real Tos_array(2436), qs_array(2436), time_junk
      Integer test_num, num_data_out(3), junk(3)
      Character*80 out_junk
      Character*13 filename1
      Logical fileexists

      junk(1) = 2630 !number of lines to be read before reaching the
output data in soddit file
      junk(2) = 2450
      junk(3) = 1730

      If (test_num .eq. 1) Then
            filename1='test1_out.txt'
      Else if (test_num .eq. 2) Then
            filename1='test2_out.txt'
      Else if (test_num .eq. 3) Then
            filename1='test3_out.txt'
      Endif

      INQUIRE(file=filename1, exist = fileexists)

      Do While (fileexists .eq. .false.)
            Call SLEEP(1)
            INQUIRE(file=filename1, exist = fileexists)
      Enddo

      Open(10,file=filename1)

      Do k=1,junk(test_num)
            Read(10,30) out_junk
            30 Format (A)
      Enddo

      Do k=1,num_data_out(test_num)
            Read(10,*) time_junk, Tos_array(1+k), qs_array(1+k)
      Enddo
      Close(10)

EndSubroutine
```

REFERENCES

[1] U.S. Nuclear Regulatory Commission, 2010, “Packaging and Transportation of Radioactive Material,” Rules and Regulations, Title 10, Part 71, *Code of Federal Regulations.*

[2] Suo-Anttila, A., Lopez, C., and Khalil, I., 2005, “Users Manual for CAFE-3D: A Computational Fluid Dynamics Fire Code,” Sandia National Laboratories Report, SAND2005-1469.

[3] Greiner, M., del Valle, M., Lopez, C., Figueroa, V., Abu-Irshaid, E., 2009, “Thermal Measurements of a Rail-Cask-Size Pipe-Calorimeter in Jet Fuel Fires,” HT2009-88520, June 19-23, San Francisco, CA, USA.

[4] del Valle, M., 2008, “Benchmark and Sensitivity Study of the Container Analysis Fire Environment (CAFE) Computer Code Using a Rail-Cask-Size Pipe Calorimeter in Large-Scale Pool Fires,” MS Thesis, University of Nevada, Reno.

[5] Spinti, J., Thornock, J., Eddings, E., Smith, P., and Sarofim, A., 2008, “Heat Transfer to Objects in Pool Fires,” Transport Phenomena in Fires, WIT Press, Southampton, U.K.

[6] Kramer, M.A., Greiner, M., and Koski, J.A., 2001, “Radiation Heat Transfer to the Leeward Side of a Massive Object Suspended Over a Pool Fire,” International Mechanical Engineering Congress and Exposition, IMECE2001/HTD-24250, CD-ROM, New York, NY, November 11-16, 2001.

[7] Beck, J.V., Blackwell, B., St. Clair, C.R., 1985, *Inverse Heat Conduction: Ill-posed Problems*, Wiley, New York.

[8] Blackwell, B.F., Douglass, R.W. and Wolf, H., 1987, “A User’s Manual for the Sandia One-Dimensional Direct and Inverse Thermal (SODDIT) Code,” issued by Sandia National Laboratories, SAND85-2478, 136 pages.

[9] Callister**,** W. D. Jr., 1991, *Materials science and engineering, An introduction***,** 2nd Edition**.** John Wiley & Sons, New York.

[10] Kramer, M.A., Greiner, M., Koski, J.A. Lopez, C., and Suo-Anttila, A., 2003, "Measurements of Heat Transfer to a Massive Cylindrical Object Engulfed in a Circular Pool Fire," *J. Heat Transfer*, Vol. 125, pp. 110-118.

[11] Doebelin, Ernest O. *Measurement Systems, Application and Design.* 5th Edition. McGraw-Hill, 2004. p54. ISBN 0-07-243886-X.

[12] Gentle, James E. *Random Number Generation and Monte Carlo Methods.* 2nd Edition. Springer, 2003. p186. ISBN 0387001786, 9780387001784.