Estimation of the Length of a Rod from Thermal Data

Nathaniel Givens, Robin Haskins, and Daniel Katz; Advisor: Dr. Lester Caudill
Department of Mathematics and Computer Science
University of Richmond

Abstract

Given an initial-boundary value problem as model of a heated rod of unknown length, we consider the inverse problem of determining this length from temperature and heat flux measurements at one end of the rod. This models the situation where one end of the rod is inaccessible. We derive and test two different algorithms to numerically estimate the length of the rod, and demonstrate their performance through numerical examples.

1. Introduction.

Although we may not be aware of it, inverse problems are omnipresent in our daily lives. Blast furnaces where high-temperature reactions occur provide a great forum for understanding what is meant by the term "inverse problems". (See [3].) The walls of these furnaces are subject to extreme amounts of heat on a constant basis, which in turn causes the materials making up the wall to deteriorate. Over time, this deterioration results in a decrease in the thickness of the walls of the furnace. For safety reasons, engineers need to know the true thickness of the wall. The super high temperatures inside the blast furnace preclude any direct measurements of the extent of the deterioration, however. Thus, the only given data engineers can obtain is the *outside* temperature of the wall of the furnace. Fortunately, this minimal amount of data is enough to determine the thickness of the wall, thanks to inverse problems. It is the fact that the determination of the thickness of the wall is done indirectly, i.e. other than by physically measuring it, that makes this situation an inverse problem. So in general, inverse problems can be defined as problems in which information is obtained through indirect rather than direct measurement.

In this paper, we consider a one-dimensional version of the blast furnace problem. Our goal is to present two algorithms for numerically determining the length of a heated rod from thermal data measured at one end of the rod. This problem is described in detail in Section 2. In Sections 3 and 4, we present the two algorithms and analyze their performance. Concluding remarks comprise Section 5. Some additional material is included in the appendices.

A similar problem, with a time-dependent rod length, was considered in [5].

2. Problem Statement.

There is a rod embedded in another object, so that one end of the rod is hidden from view (kind of like a lollipop: see Figure 1). We want to obtain the length of the rod, but we can not obtain this measure directly. Thus, this problem is an inverse problem. This problem can be thought of as a simplification of the blast furnace example described above. By considering a thin rod instead of the walls of a furnace we are reducing a more complicated two- or three-dimensional problem to a simpler one-dimensional variant.

The exposed end of the rod is heated up in a specially chosen way. Then the temperature at the exposed end is measured over time. This information is used to infer the entire length (L) of the rod using a mathematical model.

Assumptions:

Several assumptions were made at the outset:

1. The rod is perfectly laterally insulated (no heat escapes along the length of the rod)
2. The rod is of uniform construction (and therefore the rod's thermal diffusivity is constant)
3. There are no thermal sources or sinks within the rod.
4. The initial temperature in the rod is uniformly 0.
5. The hidden end of the rod is perfectly insulated.

From the preceding assumptions we can create a general model for heat flow in a rod of length L. We will use the heat equation to model the thermal dynamics of this problem. For a derivation of the heat equation, and a discussion of its appropriateness in this context, see Chapter 2 of [2].

Variables:

t – time (independent variable)
x – position along rod (independent variable)
u – temperature at position x and time t(dependent variable)

Heat Equation:

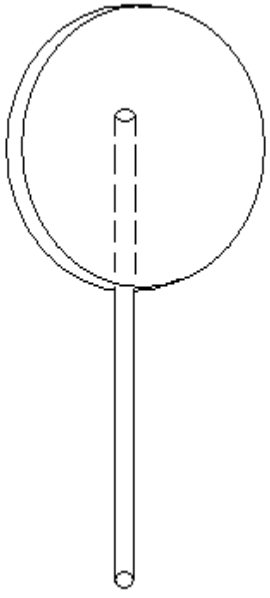


Figure 1 Illustration of an imbedded rod. The exposed end corresponds to $x=0$.

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2},\ 0 < x < L\ ,\ \ t > 0$$

Boundary/Initial Conditions:

$$u(0,x) = 0\ ,\ 0 \le x \le L$$
$$\frac{\partial u}{\partial x}(t,0) = g(t)\ ,\ t > 0$$
$$\frac{\partial u}{\partial x}(t,L) = 0\ ,\ t > 0$$

Furthermore, the temperature measurements at the exposed (*x=0*) end of the rod can be included by adding the additional boundary condition

$$u(t,0) = h(t)\ ,\ t > 0\,.$$

The goal of this inverse problem is as follows: Given *k* and *g*, use the additional information provided by *h(t)* to determine (or at least estimate) *L*. It can be shown that, under the assumption that *g(t)* is not identically zero on $0 < t < T$ for some $T > 0$, the additional data *h(t)* is sufficient to uniquely determine *L*. For details, the reader is referred to Theorem 2.1 of [1].

3. Algorithm 1.

A. Derivation.

Our first algorithm begins with an analytic solution to the IBVP. This general solution is of the form $u(t,x)$ and provides the temperature at any point *x* along the length of the rod and at any time *t* between 0 and *T*. For the model described above, solving for $u(t,x)$ gives a Fourier series representation (see [2] for derivation):

$$u(t,x) = g(t)\left(x - x^2 - \frac{L}{3}\right) - \frac{k}{L}\int_0^T g(\tau)d\tau + \sum_{n=1}^{\infty} e^{-(\frac{n\pi}{L})^2 t}\left(\int_0^T \frac{2g'(\tau)L}{n^2\pi^2} e^{(\frac{n\pi}{L})} d\tau\right)\cos(\frac{n\pi}{L})$$

We are only guaranteed to have access to the exposed end of the rod – when $x = 0$ – and thus we set $x = 0$ for all *t*. That results in the following equation:

$$u(t,0) = -\frac{L}{3}g(t) - \frac{k}{L}\int_0^T g(\tau)d\tau + \sum_{n=1}^{\infty} e^{-(\frac{n\pi}{L})^2 t}\left(\int_0^T \frac{2g'(\tau)L}{n^2\pi^2} e^{(\frac{n\pi}{L})} d\tau\right)$$

This function is set equal to *h(t)* – which is the temperature data recorded at the exposed end of the rod. Since *k* is a known constant (thermal diffusivity), the only other unknown in the equation is *L*. When the two functions, *h(t)* and *u(t,0)* are set equal to each other, the resulting equation can be solved (numerically, at least) for *L*.

It is important to note that in practice the data measured at the end of the rod would not be in the form of a function, but in the form of discrete temperature readings over time. Thus our algorithm was designed to function equally well whether given *h(t)* as a formula or merely as a list of points.

B. Implementation.

To implement the algorithm, we first constructed a Mathematica package that contained the necessary functions and definitions for solving for *L*. We then wrote a Java program that called the Mathematica kernel, loaded our custom package, and sent it the *h(t)* function or the *h(t)* data points (which it read from a .txt file). The results were saved to a .txt file for later analysis.

The program was given *u(t,0), g(t),* and *h(t)* as input, as well as a starting time, an ending time, and a time step. The two functions were evaluated and set equal. Mathematica solved for *L*, and then incremented by the time step. This process continued until the ending time was reached, producing a sequence of estimates of *L*.

Since we didn't actually have a rod embedded in some substance to heat up and test, we generated *h(t)* functions and data points ourselves. To create these functions, we simply chose an *L*-value and plugged it into our solution *u(t,x)*. Then we set *x=0* to get *u(t,0) = h(t).* In order to add some realism (by simulating measurement errors) we added a small amount of noise to *h(t),* and then used that as a more realistic *h(t)* function. Our method for generating *h(t)*-data points was somewhat more precise. In that case, we chose an *L*-value to plug into *u(t,x)*and set *x = 0*. Then we decided upon a time interval (say 5). Finally, we solved for *u* at *t = 0, t = .5, t = 1, t = 1.5...* In this way we generated a list of simulated temperature readings at the exposed end of the rod for different time steps starting with t = 0. In order to make the data more realistic, we then increased or decreased the value by a random number with a percentage range. For example, if the specified error was 10%, then a randomly generated number up to ten percent of the true value of *h(t)* at a time interval *t* would be randomly added or subtracted from the true

value. Using this method we easily created entire lists of *h(t)*-points. These points were fed into the Mathematica program from a .txt file via Java and then used to find *L* at the appropriate times.

The first difficulty encountered is that when numerically solving the equation $u(t,0) = h(t)$ for *L*, there are frequently extraneous solutions: especially when the *h(t)* input is randomized. We developed a few strategies to help our algorithm account for these extraneous answers. The simplest step was to discard all complex solutions. Usually there were still multiple real-valued solutions for at least some of the time-steps. Our second step to further narrow down the solutions was to restrict the possible *L* values. We did this in our initial program by accepting an "Lhigh" and "Llow" variable as input to the algorithm. These variables represented limits on the possible maximum and minimum lengths *L* to be considered. Real-valued solutions that fell outside of this range were discarded.

When we looked closer at the real valued solutions that we were discounting we realized that almost all of them were very close to 0. In order to improve the algorithm – by not requiring a maximum or minimum estimate for the length of the rod, later versions of the algorithm were programmed to discard the values of *L* that were close to 0. This almost always resulted in at least a few time-steps for which our algorithm returned a single real-valued solution for *L*.

We then took the average of all the values for *L* that were obtained this way. So for a problem with 10 time-steps, perhaps 3 or 4 would have only a single real-valued solution. By averaging the resulting *L*-values we created a temporary answer. The algorithm then returned to all the time-steps which still contained multiple real-valued solutions and discarded the solutions farthest from the temporary average. The temporary average was also updated every time a new *L*-value was isolated. Finally, when all of the time-steps had only one *L*-value remaining the final average was returned as the solution. This created a more robust algorithm than the original implementation.

In order to test the algorithm at the outset, we simply chose values for *L* and computed the resulting *h(t)* functions. When the exact function *h(t)* or the exact data points from the *h(t)* function were input into the algorithm, the resulting *L* was extremely accurate – as would be expected.

More realistic testing was accomplished by producing *h(t)* points that varied randomly by up to 10 – 30 percent (we did trials using up to 10, 20 and 30% error). The algorithm proved to

be resistant to these errors, however, since our averaging approach naturally caused the random errors to largely cancel out. For example with $L = 15$ and a time range from 0-15, even a 30% randomized error caused less than 1% error in the final L (over 15 distinct trials).

C. Example.

If the algorithm is workable, using $h(t)=u(0,t)$ should return precise values of L. We tested the algorithm with $h(t)=u(0,t)$ and it returned exact values of L. We also tested the algorithm with data point values of the form $(t, u(0,t))$. Since the data points were only accurate to the third decimal place, the algorithm would not necessarily return the exact L for which we were looking. An example output from a run with L=3 is shown below using *Mathematica*; with values of length L against time t.

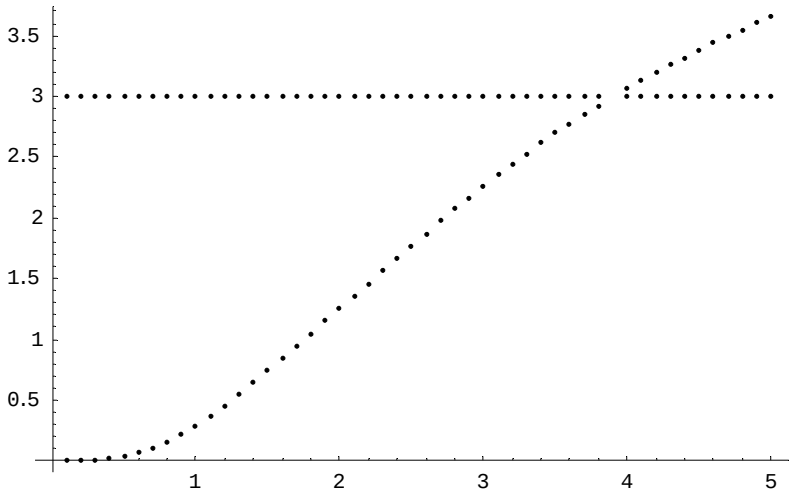


Figure 2. Values of L returned by algorithm 1, as a function of time.

It was clear from the graph that one value of L stayed constant around 3.0 while the other one increased over time. While we currently have no specific reason for this occurrence, one possible explanation could be that the inaccuracy of the data points makes the function return 2 values, one of which is affected by time. It is also likely that nonlinearity plays a role as well. This would be an interesting avenue for further investigation.

D. Discussion.

Despite the improvements in the second incarnation of this algorithm, the primary Achilles heel remains the problem of isolating the correct values for L from the additional real-valued solutions. With short time periods or large L-values the near-zero solutions tend to become slightly larger. Since the definition for "close to 0" is arbitrarily written into the code itself, the algorithm fails to isolate any lengths under those circumstances. Thus, without any isolated L-values, the algorithm can not construct a temporary average and so cannot return any solution whatsoever.

Even a fairly cursory examination of the real-valued L solutions reveals a clear and interesting pattern to these solutions, as is evidenced by the illustration in Figure 2, where the algorithm returns not only correct values of L, but also extraneous values that seem to occur in a specific pattern. It would be interesting to investigate this further. One potential way to capitalize on a greater understanding of this phenomenon would be to utilize this additional information to drastically expand the capabilities of the algorithm.

The second major limitation of the algorithm is that it works much better for constant-length rods than for rods which may be changing in length (either decaying or growing). The search method used in the latest versions of the algorithm can be very successful for finding constant-length solutions, but it is that same averaging of values that makes it so successful that also renders it entirely unsuitable for use on variable-length rods.

In the case of the variable-length rod the main obstacle remains the matter of sifting through the L-values to find the correct ones. When the algorithm was run on rods with a linearly-decaying length, the correct L-values were always present, but it would be difficult to pick them out without knowing ahead of time what should be happening to the length of the rod.

Possible solutions to this include more advanced search techniques that use a kind of Ockam's Razor-inspired approach to determine possible functions for the change in length of the rod over time, and then select the simplest functions. This would clearly be only a limited solution to the problem. A better solution would have to rely on a clearer understanding of what the excess solutions are, and how they arise.

4. Algorithm 2.

A. Derivation.

By using a test function approach, we are able to eliminate the complex process of solving for u(t,x) as well as its complicated solution inherent in Algorithm 1. With the proper $g(t)$ and $h(t)$, the algorithm works by simply inserting a value of $t=T$. At this point, Mathematica needs only to integrate and solve for L. This analytic approach is described below.

Here is the one dimensional heat equation again, with the initial and boundary conditions:

$$\frac{du}{dt} = k\frac{d^2u}{dx^2} \qquad (1)$$

$$u(0,x) = u_0(x) \qquad (2)$$

$$-\frac{du}{dx}(t,0) = g(t) \qquad (3)$$

$$u(t,L) = a(t) \qquad (4)$$

The functions u_0, g, and a, as well as the constant k, are assumed known, while L is unknown. When measuring the data at the end of the rod, we can write the temperature of the rod as a function of time. Thus:

$$u(t,0) = h(t)$$

Define ϕ, our test function, to satisfy

$$\frac{\partial\phi}{\partial t} + \frac{\partial^2\phi}{\partial x^2} = 0 \qquad 0 < x < L \quad 0 < t < T$$

$$\phi(t,L) = 0 \qquad 0 < t < T$$

Multiply both sides of the PDE by ϕ and integrate with respect to both length and time to obtain:

$$\int_0^L\int_0^T \phi u_t\,dtdx - \int_0^L\int_0^T \phi u_{xx}\,dxdt = 0$$

Using integration by parts, it is possible to move some of the derivatives from the temperature function u to the test function ϕ. After a few steps of integration by parts, our original equation now looks like:

$$\int_0^L \phi(T,x)u(T,x)dx - \int_0^L \phi(0,x)u(0,x)dx - \int_0^L\int_0^T (\phi_t + \phi_{xx})u\,dtdx -$$

$$\int_0^T \phi(t,L)u_x(t,L)dt + \int_0^T \phi(t,0)u_x(t,0)dt + \int_0^T \phi_x(t,L)u(t,L)dt - \int_0^T \phi_x(t,0)u(t,0)dt = 0$$

Because we have 'invented' our own test function ϕ, we can give properties to the function that will simplify our lengthy equation, as long as it possess the necessary derivatives.
Using the definition of ϕ and the conditions (2)-(4), the preceding equation now resembles something that is more approachable.

$$\int_0^L \phi(T,x)u(T,x)dx - \int_0^L \phi(0,x)u_0(x)dx -$$

$$\int_0^T \phi(t,0)g(t)dt + \int_0^T \phi_x(t,L)a(t)dt - \int_o^T \phi_x(t,0)h(t)dt = 0$$

We denote the left hand side of this equation by *F(L)*. Notice, that for the last four terms of the above equations, all of the variables are known except for *L*. The first term, however, contains the unknown solution *u*, which complicates the job of solving for L. We can impose an additional condition on the problem, so that *F(L)* takes a more manageable form. One way to do this is by requiring that both *a(t)* and *g(t)* approach *0* as *t* approaches infinity. Then, *u(T,x)* will be approximately *0* for large values of *T*, since the temperature of the rod will tend towards *0* under these conditions, according to (1)-(4). Thus, if we choose *T* large enough, we will have $u(T,x) \approx 0$, so we can ignore the troublesome integral term, leaving us with

$$F(L) = -\int_0^L \phi(0,x)u_0(x)dx - \int_0^T \phi(t,0)g(t)dt + \int_0^T \phi_x(t,L)a(t)dt - \int_o^T \phi_x(t,0)h(t)dt = 0 \qquad (5)$$

At this point, we see that once the ingredients of (1)-(4), and the data *h(t)* are known, we can determine an estimate *L* of the true length *LTrue* as follows:

1. Find a specific test function ϕ that satisfies the stated conditions

$$\frac{\partial \phi}{\partial t} + \frac{\partial^2 \phi}{\partial x^2} = 0 \quad \text{and} \quad \phi(t,L) = 0..$$

2. Compute *F(L)* as a function of *L* via the left-hand side of equation (5). (We used Mathematica to compute the necessary integrals.)
3. Numerically solve *F(L)=0* to get an estimate of *LTrue*

A number of important mathematical questions arise, including:

1. Does the equation *F(L)=0* have a solution? If so, how many?
2. Which solution to *F(L)=0* corresponds to the true length *L* of the rod?
3. Does the effectiveness of this algorithm depend on our choices of *g(t), a(t),* and *T* ?

We investigate these issues, within the context of a specific example, by simulating the thermal imaging data, then using this data to test our algorithm.

B. Examples.

To produce data *h(t)* to use in our algorithm, we chose a specific example. For this example, we specified the following:

- The true value *LTrue* of the length of the rod
- The ending time *T* for the data gathering period
- $g(t) = te^{-pt}$, for some $p>0$
- $a(t) = 0$

  (Note that both *g(t)* and *a(t)* satisfy the requirements that they go to 0 as $t \to \infty$)
- $u_0(x) = 0$
- $\phi(t,x) = e^{-\lambda^2(T-t)} \sin(\lambda(x-L))$ for some $\lambda > 0$

Next, after choosing a value for *p*, we solved the initial-boundary value problem (1)-(4) to obtain *u(t,0)*, which we used as our *h(t)*.

The equation *F(L)=0* could now be written as:

$$F(L) = -\int_0^T te^{-pt} e^{-\lambda^2(T-t)} \sin(\lambda L)dt + \int_0^T \lambda h(t) e^{-\lambda^2(T-t)} \cos(\lambda L)dt = 0$$

For the rest of this report, we will be investigating the accuracy and limitations of the test function algorithm for different parameter values (i.e. values of *p,λ,* and *T)*.

Figure 3 shows two graphs of the *F(L)* function for two different sets of parameter values.

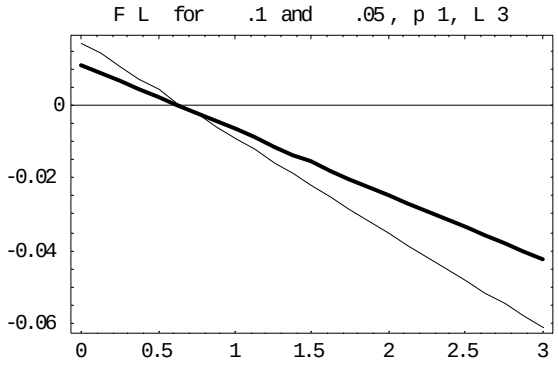


Figure 3 The function *F(L)* vs. *L*, for two different sets of parameter values. The thicker line corresponds to λ=0.05

Once all of the necessary values were defined, we could find the roots of *F(L)* using our favorite root-finding method. One of the roots obtained this way should have been equal to, or very close to, the value of *LTrue*, which was used in the computation of *h(t)*. For this research, the *FindRoot* function from *Mathematica* (see [4]) was used to solve for the values of *L*. Unfortunately, this method was not as straight forward and helpful as it might seem. Most of the time, *FindRoot* would return a value that was either very close or equal to *LTrue.* However, with some combinations of values of the parameters, *FindRoot* would return values of *L* that were not close to the supplied *LTrue* value. Thus, the accuracy of our algorithm was in jeopardy. But, were there any values of our parameters, specifically *p* and λ, which preserve the precision of the algorithm?

Our next goal was to try to find these combinations of values. The first step was to select a group of possible values that were both helpful to us, while at the same time physically reasonable. Then, we would find the best combinations of these variables, with different values of the time length *T*. We hoped to find a group of parameter values that were consistently accurate for many values of *T*. For λ, we tested the values *.01, .05, .10, .25, .50,* and *1.00*. For *p*, the values *0.1, 0.5, 1.0*, and *2.0* were tested. All values between *0.2* and *5.0*, incremented by *0.2*, were used for *LTrue*. By computing the aggregate percent error for all the *LTrue* values for a given time length, *p* and λ, we could pinpoint parameter values that are the most accurate. For example, for the *LTrue=3* case, good parameter choices included $\lambda = 0.25, p = 0.5$ and $\lambda = 0.05, p = 1.0$.

Even though we had compiled accurate pairs of constant values *p* and λ in regards to percent error, it did not mean that the test function algorithm was accurate in finding length *L*. More testing was needed. The next logical step was to increase the time length without bounds, and observe how the values of *L* changes. Our goal was to discover any limitations of the test algorithm for each pair of constant values of *p* and λ. This was accomplished by creating a computer program within *Mathematica*.

For each value of *p* and λ, *Mathematica* would test the accuracy of the given *LTrue* vs. the *L* which was solved for in the *FindRoot* function in *Mathematica*. If the absolute percent error was less than or equal to 10%, the value of *LTrue* would increase by a specific step size, generally *0.2* for most of our tests. If the percent error was greater than 10%, *Mathematica*

would return the previous *LTrue* value that was within the 10% range. Thus, we were able to plot maximum *LTrue* vs. time length, and try to find any relationships between the two.

The search algorithm was tested on six different pairs of constant values p and λ. These pairs were chosen because their percent error, which was previously calculated, was among the lowest. The length of time before the algorithm failed depended upon the specific parameter values. Figure 4 displays, for a given time interval, the maximum length of the rod for which the algorithm could recover the length accurately.

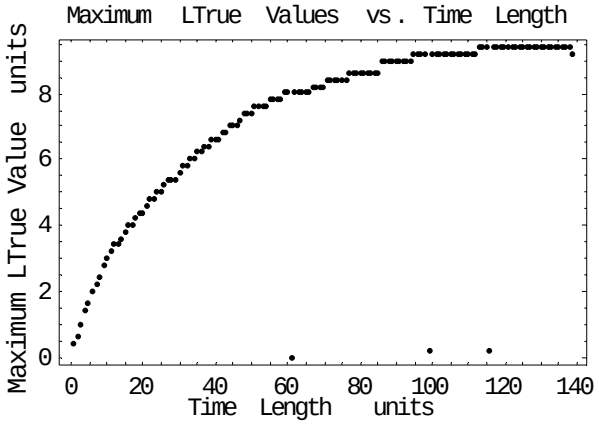


Figure 4 Maximum recoverable rod lengths as a function of the length of the time interval, for the case where λ=0.01, p=1.0, and T=140.

(For similar plots with different choices of parameter values, see Appendix 1.) As illustrated in Figure 4, the increase of maximum *LTrue* value leveled off at around *9.2*. For the other graphs, as the time length increased, the *LTrue* value approached an asymptotic value. This phenomenon suggested that for a given set of parameter values there was a corresponding maximal value for the *LTrue* which the algorithm could successfully determine. This seemed logical, because of the indirect nature and approach of finding length with this algorithm. This also explained the random points on the graph that were not a part of the curve.

Previously, we mentioned that there were pairs of constant parameter values p and λ which resulted in better absolute percent errors than other pairs. What would happen if we applied the search algorithm on a bad pair of parameter values? The graph below is a typical example of a plot in which the set of constants was not a good choice:

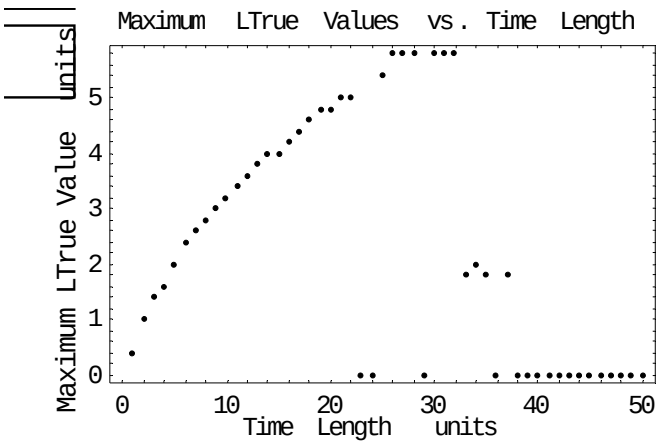


Figure 5 Maximum recoverable rod lengths as a function of the length of the time interval, for the case where λ=0.01, p=2.0, and T=50.

For about the first 20 time units, the plot looked reasonably similar to Figure 4. However, during the next 20 second interval the search algorithm started to fail. There was a sudden jump up to the maximum value of *5.8*, which was sustained for a few seconds. Then, there was another jump, but this time downwards to around *2.0*. After staying at this level for a few units of time, the algorithm finally receded to *0.0*.

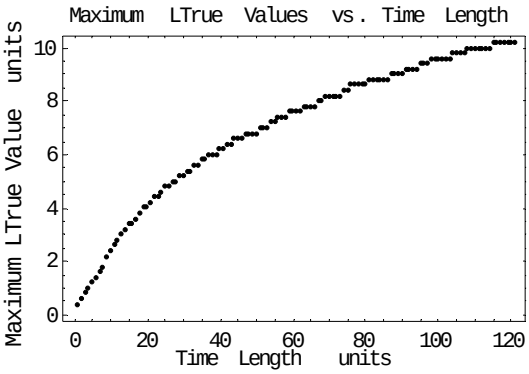
Even though the previous two plots are vastly different, only the parameter value of p has changed. By doubling the exponent value in the *g(t)* function, the heat source function, the accuracy of the test function algorithm has decreased. Appendix 1 displays plots with different λ values, while keeping the value of p constant. With a value of $p = 2.0$ in the exponent of *g(t)*, it seems that the heat source has tampered off too quickly for the test function algorithm to work. The other plots in Appendix 2, similar to Figure 5 but with different parameter values, support this claim.

5. Concluding Remarks.

In this paper, we have shown two different methods to approach a one dimensional heat rod inverse problem. In algorithm one, an analytical approach using separation of variables yielded an equation for the length of the rod. In algorithm two a differing analytic approach involving test functions was used to derive an equation as a function of length. By finding the roots of this function we recovered the length of the rod. It is important to remember the slight difference in the original equation used in both algorithms. Slight variations would occur if the boundary conditions were changed to reflect the other model. However, the purpose of this

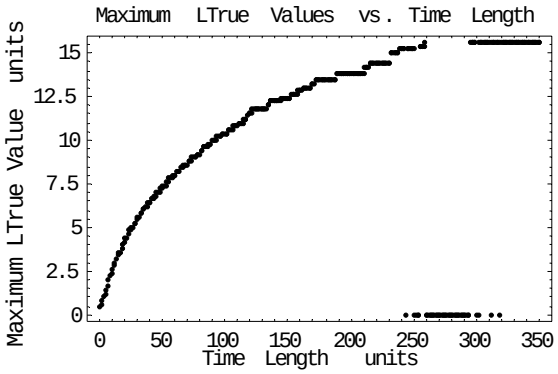
paper was not to solve specific initial value boundary problems, but rather to develop methods for more than one problem. The approaches used in this paper can be used for other IBVP problems, not just the specific examples included.

Appendix 1



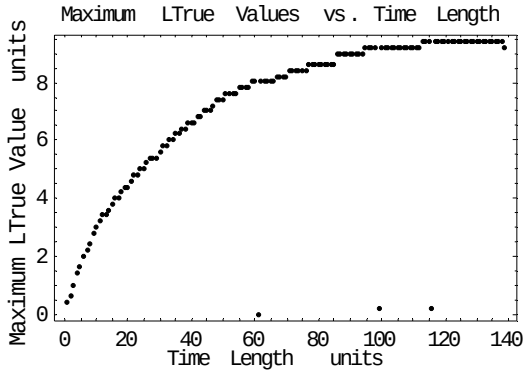
$\lambda = 0.1$

$p = 0.5$



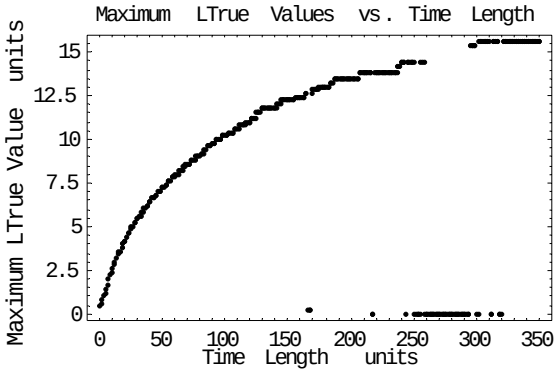
$\lambda = 0.01$

$p = 0.5$



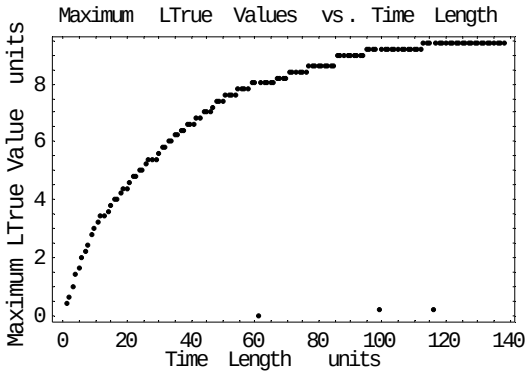
$\lambda = 0.05$

$p = 1.0$



$\lambda = 0.05$

$p = 0.5$

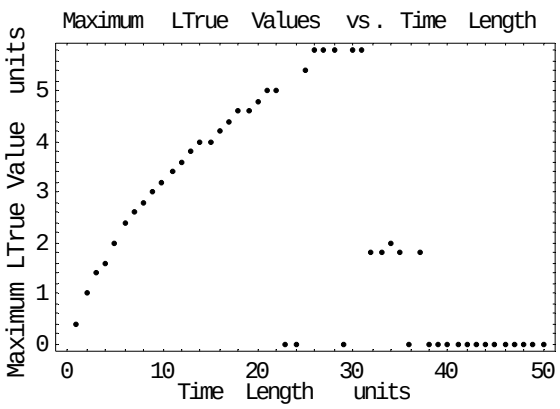


$\lambda = 0.25$

$p = 0.5$

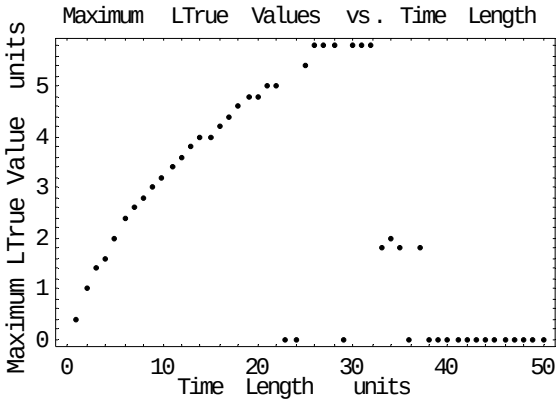
NOTE: It is unknown why the last two plots both fail in the approximate interval of [250,300]. However, the curve continues in its expected fashion after the gap.

Appendix 2



$$\lambda = 0.05$$

$$p = 2.0$$



$$\lambda = 0.1$$

$$p = 2.0$$

References

1. Bryan, Kurt, and Caudill, Lester, Uniqueness for a boundary identification problem in thermal imaging, *Elect. J. Diff. Eq.* **C-1** (1997), 23-39.
2. Keane, Michael K., *A Very Applied First Course in Partial Differential Equations*, Prentice Hall, 2002.
3. Sorli, K., and Skaar, I., Monitoring the wear-line of a melting furnace, Inverse Problems in Engineering: Theory and Practice, Proceedings of the Third International Conference on Inverse Problems in Engineering, Port Ludlow, Washington, June 13-18, 1999.
4. Wolfram, Stephen, *The Mathematica Book*, Wolfram Research, London, 1996.
5. Yarlikina, N., and Walrath, H., Determining the length of a one-dimensional bar, Rose-Hulman Math Department Technical Report 04-02.