Rose-Hulman Institute of Technology

Rose-Hulman Scholar

10-18-2012

Thermal Detection of Inaccessible Corrosion

Matthew Charnley
Rose-Hulman Institute of Technology

Andrew Rzeznik
Rose-Hulman Institute of Technology

Follow this and additional works at: https://scholar.rose-hulman.edu/math_mstr

Part of the Partial Differential Equations Commons

Thermal Detection of Inaccessible Corrosion

Matthew Charnley and Andrew Rzeznik

Adviser: Kurt M. Bryan

Mathematical Sciences Technical Report Series
MSTR 12-03

October 18, 2012

Thermal Detection of Inaccessible Corrosion*

Matt Charnley[†]and Andrew Rzeznik[‡]

October 18, 2012

Abstract

In this paper, we explore the mathematical inverse problem of detecting corroded material on the reverse side of a partially accessible metal plate. We will show how a linearization can be used to simplify the initial problem and explain a regularization method used to obtain acceptable results for the corrosion profile. We will also state and perform some calculations for the full three-dimensional problem for possible future work.

Contents

1 Introduction and Motivation

In many industrial and engineering applications, it can be useful to know if the reverse side of a partially accessible piece of material has been corroded. This corrosion could damage the efficiency of a system, or make it unsafe to use. By taking a mathematical approach, we hope to simplify the problem in such a way that a computer could easily perform a reconstruction of the corrosion profile from temperature measurements on the accessible surface. We also hope that this analysis will grant more insight into the problem and how exactly the corrosion affects the transfer of heat within the body.

This paper will show the process of setting up the problem, generating the necessary data, and using it to approximate the corrosion profile of the block of metal. Section 2 will discuss the forward problem, which is generally solved by a heat equation solver, while section 3 will discuss the inverse problem of finding the corrosion profile from temperature measurements, which is what we are more interested in. Section 4 will discuss a uniqueness proof, which guarantees that a single set of temperature data on the top surface defines a unique corrosion profile on the back surface. Section 5 will cover the process of generating a solution, from the linearization of the problem, use of Green's Identity, and generating a profile with the smallest L^2 norm. Section 6 will discuss different methods used to improve the resulting profiles, and section 7 will show some of our results. Sections 8 and 9 will discuss the most important part of the paper, a new linearization method that works fairly well for any choice of thermal parameters, and justifies the removal of one term from the linearization generated in section 5. Finally, the paper will close with a discussion of the three-dimensional inverse problem in section 10 and some conclusions and possibilities for future work in sections 11 and 12.

1.1 Terms and Definitions

The following terms and symbols will be used frequently through the paper

Ω	The entire region where heat transfer is taking place.
Ω_1	The section of Ω where the material has not been corroded.
Ω_2	The section of Ω where the material has been corroded.
u_0	The temperature profile in the uncorroded plate Ω made of material (1), using the same input flux used to compute u_1, u_2.
u_1	The temperature profile in Ω_1, function of (x, y, t).
u_2	The temperature profile in Ω_2, function of (x, y, t).
$C(x)$	The Corrosion Profile; The function that separates Ω_1 from Ω_2.
α	Thermal Diffusivity, a material property. Determines how fast heat can diffuse through the medium.
α_i	Thermal Diffusivity for material i, which is present in region Ω_i.
k	Thermal Conductivity, a material property. Determines how well heat can be conducted in from the environment.
k_i	Thermal Conductivity for material i, which is present in region Ω_i.

2   Forward Problem

Assume that we have a finite rectangle Ω of length L and height 1, as shown in Figure 2.1 below. The rectangle Ω is set in the Cartesian plane $\mathbb{R}^2$ so that $x = 0$ marks the left side of the rectangle, $x = L$ is at the left edge , $y = 0$ denotes the bottom of the sample and $y = 1$ indicates the top.
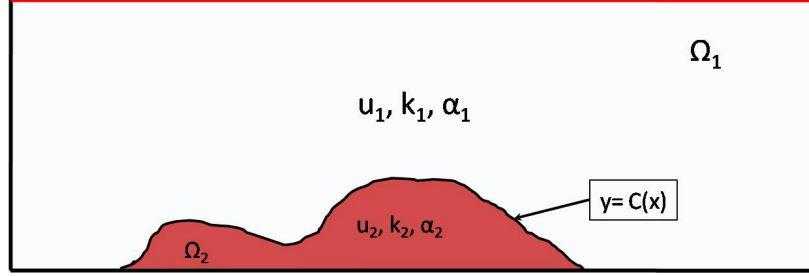

Figure 2.1: General Setup for the Problem

The forward problem is what is usually solved in differential equations classes: Given the boundary and initial conditions on the problem, solve for the temperature profile. For this specific problem, we know there are two regions, separated by the curve $C(x)$, and each region has different thermal properties, such as thermal conductivity and thermal diffusivity. We will also assume that all external boundaries are insulated except for the one at $y = 1$, where some input flux $g(x)$ is applied. We also know that the temperature and heat flux must be continuous over any interface, including the curve $C(x)$. All of these conditions can be formally stated as follows.

We know that u_1 satisfies

$$
\begin{aligned}
\frac{\partial u_1}{\partial t} - \alpha_1 \nabla^2 u_1 &= 0 \text{ on } \Omega_1 \\
\frac{\partial u_1}{\partial x} &= 0 \text{ on } x = 0, \ x = L \\
\frac{\partial u_1}{\partial y} &= g(x) \text{ on } y = 1 \\
u_1(x, y, 0) &= 0 \text{ on } \Omega_1
\end{aligned}
$$

while u_2 satisfies

$$
\begin{aligned}
\frac{\partial u_2}{\partial t} - \alpha_2 \nabla^2 u_2 &= 0 \text{ on } \Omega_2 \\
\frac{\partial u_2}{\partial x} &= 0 \text{ on } x = 0, \ x = L \\
\frac{\partial u_2}{\partial y} &= 0 \text{ on } y = 0 \\
u_2(x, y, 0) &= 0 \text{ on } \Omega_2
\end{aligned}
$$

And the continuity conditions on $C(x)$ give us

$$
\begin{aligned}
u_1 &= u_2 \text{ on } C(x) \\
k_1 \frac{\partial u_1}{\partial \vec{n}} &= k_2 \frac{\partial u_2}{\partial \vec{n}} \text{ on } C(x)
\end{aligned}
$$

The forward problem would be stated: Given the input flux $g(x)$, the thermal properties of both materials, and the curve $C(x)$ dividing Ω_1 and Ω_2, find the temperature profiles u_1 and u_2 satisfying these equations.

If the domain is simple, *i.e.* a rectangle, Fourier Series can be used to solve this problem. However, in this case, the corrosion profile makes the domain somewhat irregular, and numerical methods become the main tool for reaching a solution. COMSOL Multiphysics was our program of choice for solving the forward heat equation problem. It uses a finite element method to generate the solution to the heat equation given a geometry and any specified combination of heat fluxes, insulations and material properties. From this data, it will generate the temperature values for a mesh of points (x, y) in the plate at a set of specified timesteps. The program also contains a functionality to export different sections of the data to text files, which can be used in MATLAB or other numerical software. See Appendix B for a more detailed description of this process.

Even though the forward problem is not what we are trying to solve, this kind of software is very useful to us. Without any experimental apparatus, this forward solver is our source for getting the necessary data to test our methods for solving the inverse problem.

3 Inverse Problem

The forward problem is a deterministic differential equation that can be solved numerically. However, this is not the problem that we want to solve. In engineering applications, it is assumed that we will be able to measure temperature and an input flux, but we do not know what the inside of the material looks like. Specifically, we do not know, and would like to determine, the function $C(x)$ that divides the two regions of different materials. Assuming that the second region is corroded or missing material, knowing this function $C(x)$ could help determine the safety of that component in the structure or help to see if it needs to be replaced.

Intuition says that the change in material properties should affect the way the heat flows through the object. We hope that this will cause a significant change in the temperature profile of the plate, specifically on the top surface. Figure 3.1 shows a COMSOL image of the temperature in two identical plates after a given amount of time. The only difference between the two plates is the missing material on the bottom of the left plate.
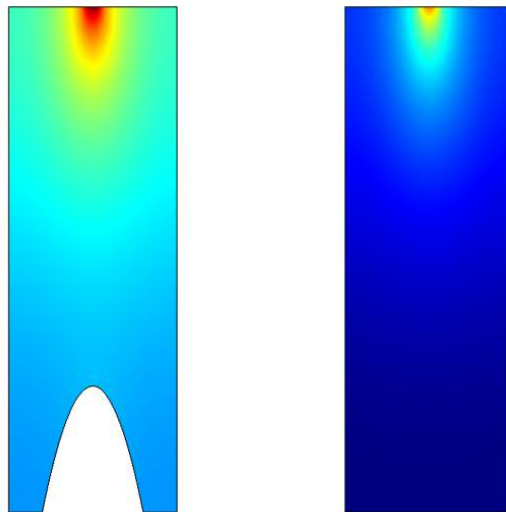

Figure 3.1: COMSOL Images Showing Changes in Temperature Profile Due to Corrosion

As can be seen in the figure above, there is clearly a difference in the temperature at the top of the plate caused by the corrosion at the bottom. This gives us hope that we should be able to recover at least some information about the corrosion profile given the accessible surface temperature.

4   Uniqueness

We desire to prove a uniqueness result for the full non-linear problem. We will proceed by using a paper by O.Poisson [3]. We want to prove that for any given temperature data on the top surface $u_1(x, 1, t)$, there is a unique corrosion profile $C(x)$ which gives that surface data for the stated boundary and initial conditions. This is stated in the following theorem:

Theorem 4.1 (Inverse Problem Uniqueness). If two corrosion profiles $C(x)$ and $C^*(x)$ both generate the same temperature solution u_1 on the top boundary of Ω_1 for all time and a given input flux function $g(x)$, then $C(x) = C^*(x)$.

To begin this proof, we first consider the two regions Ω_1 and Ω_2 that make up our plate. Assume that we know the solution u_1 on some open portion Γ_1' of the boundary Γ of our region Ω_1. This would correspond to the given temperature data on part of the accessible surface. To continue, we note one form of a unique continuation theorem:

Theorem 4.2 (Unique Continuation). Given two solutions u_1 and u_2 on some region Ω, if these solutions have the same Cauchy data, *i.e.* $u_1 = u_2$ and $\frac{\partial u_1}{\partial \vec{n}} = \frac{\partial u_2}{\partial \vec{n}}$, on an open portion of the boundary of Ω for all time then $u_1 = u_2$ on Ω for all time.

This theorem tells us that a given boundary temperature defines a unique internal temperature solution u_1 in all of Ω_1 for a given input flux $g(x)$. If we assume that the corrosion profile $C(x)$ is zero near the sides of Ω, and does not extend to the top boundary, then we can draw a curve S that starts on the bottom boundary near the left side and trace it through Ω_1 back to the bottom boundary on the right side. Since the corrosion profile $C(x)$ is zero near the sides of Ω, the curve S can be contained completely in Ω_1 and satisfy $S > C(x) \,\forall\, x$. Now we define Ω' to be the region between this curve S and the bottom boundary of Ω at $y = 0$. S can also be chosen so that the boundary of Ω' is C^2. Since S is in Ω_1, the Unique Continuation Theorem specifies the Cauchy data on S uniquely from the Cauchy data on the top boundary of Ω. We now have our corroded region entirely contained in a C^2 manifold Ω', where the Cauchy data on part of the boundary of Ω' is specified from the Cauchy data at $y = 1$.

Using the main result from Poisson's paper ([3], section 1.3), we see that for our given initial data, we need two boundary measurements on S to uniquely specify the thermal diffusivity as a function of position over Ω'. Since we are assuming the thermal diffusivity is constant within each material, this discontinuous function will give a unique determination of the corrosion profile $C(x)$ dividing the two different materials. Since the Cauchy data on S is specified uniquely from the Cauchy data on $y = 1$, we can now, with two distinct heat fluxes and temperature measurements on $y = 1$, guarantee a unique corrosion profile for a given set of this data. Since our trial set (see Section 6.2) involves three trials for each corrosion profile, we have the desired uniqueness result in this case.

5   Solving the Inverse Problem

Remark 1. This information was left in the paper as a full detail of all of our work from the summer. However, this is an incorrect method of linearization and use of Green's Identity, but this section covers the basic ideas used in the actual derivation. See Section 8 for the correct approach. Within this section, Section 5.2.1 is still valid, and is used within Section 8.

5.1   Linearization

The first step to solving this inverse problem is linearization. This is done in order to need only one test function to simplify the problem with Green's Identity; see Section 5.2 for the calculations. Specifically, we assume that the corrosion profile is a small perturbation of the uncorroded back surface $y = 0$, and then use this assumption to linearize the relationship between the function $C(x)$ and the top surface temperature data. Unlike in the electrical case (see [1]) where there is only one differential equation to solve (Laplace's Equation), there are two distinct heat equations in Ω_1 and Ω_2 because of the different thermal diffusivities, and this slightly complicates the procedure.

In order to linearize the problem, we assume that our corrosion profile is a small perturbation away from $y = 0$, and therefore the temperature in both Ω_1 and Ω_2 is very similar to that of the uncorroded plate at the same positions. We can then define

$$
\begin{aligned}
u_1 &= u_0 + \tilde{u}_1 \\
u_2 &= u_0 + \tilde{u}_2 \\
C(x) &= 0 + \epsilon C_0(x)
\end{aligned}
$$

where $\tilde{u}_1 = \epsilon \bar{u}_1$ and $\tilde{u}_2 = \epsilon \bar{u}_2$ are $O(\epsilon)$ perturbations of the uncorroded temperature profile, at least formally. Since we assume u_2 only exists on a small portion of the region, we are going to look at $\tilde{u}_1$.

By evaluating derivatives and the initial conditions, it can be seen that $\tilde{u}_1$ satisfies (in the limit $\epsilon \to 0^+$):

$$
\frac{\partial \tilde{u}_1}{\partial t} - \alpha_1 \nabla^2 \tilde{u}_1 = 0 \quad \text{on } \Omega, \text{ after extension.} \tag{5.1}
$$

$$
\frac{\partial \tilde{u}_1}{\partial \vec{n}} = 0 \quad \text{on } y = 1, \ x = 0, \text{ and } x = L. \tag{5.2}
$$

$$
\tilde{u}_1(x, y, 0) = 0 \quad \text{on } \Omega. \tag{5.3}
$$

We also know that the flux boundary condition is satisfied on $C(x)$, so

$$
\begin{aligned}
k_1 \frac{\partial u_1}{\partial \vec{n}} &= k_2 \frac{\partial u_2}{\partial \vec{n}} \\
k_1 \frac{\partial (u_0 + \tilde{u}_1)}{\partial \vec{n}} &= k_2 \frac{\partial (u_0 + \tilde{u}_2)}{\partial \vec{n}} \\
k_1 \frac{\partial \tilde{u}_1}{\partial \vec{n}} &= (k_2 - k_1) \frac{\partial u_0}{\partial \vec{n}} + k_2 \frac{\partial \tilde{u}_2}{\partial \vec{n}} \quad \text{on } C(x)
\end{aligned} \tag{5.4}
$$

However, since $\frac{\partial f}{\partial \vec{n}} := \nabla f \cdot \vec{n}$ and the normal vector to the curve $C(x)$ is defined by

$$
\vec{n} = \frac{<-C'(x), 1>}{\sqrt{C'(x)^2 + 1}}
$$

we can rewrite equation (5.4) as

$$
k_1 \left(\frac{\partial \tilde{u}_1}{\partial y} - C'(x) \frac{\partial \tilde{u}_1}{\partial x} \right) = (k_2 - k_1) \left(\frac{\partial u_0}{\partial y} - C'(x) \frac{\partial u_0}{\partial x} \right) + k_2 \left(\frac{\partial \tilde{u}_2}{\partial y} - C'(x) \frac{\partial \tilde{u}_2}{\partial x} \right) \tag{5.5}
$$

This is the equation that we want to linearize about $y = 0$ to get a formula for the y-derivative of $\tilde{u}_1$ on the bottom boundary of the plate , which will fully define our function $\tilde{u}_1$. The linearization process follows the format of a tangent line approximation. Assuming f is differentiable, and the distance between y and y_0 is order ϵ, then we have

$$f(y) \approx f(y_0) + f'(y_0)(y - y_0) \quad \text{or} \quad f(y) = f(y_0) + f'(y_0)(y - y_0) + O(\epsilon^2)$$

from Taylor's Theorem.

Using that procedure on each term in equation (5.5) gives the following, ignoring all terms of order ϵ^2.

$$\frac{\partial \tilde{u}_1}{\partial y}\bigg|_{C(x)} = \frac{\partial \tilde{u}_1}{\partial y}\bigg|_{y=0} + C(x)\frac{\partial^2 \tilde{u}_1}{\partial y^2}\bigg|_{y=0} + O(\epsilon^2) = \frac{\partial \tilde{u}_1}{\partial y}\bigg|_{y=0} + O(\epsilon^2)$$

$$C'(x)\frac{\partial \tilde{u}_1}{\partial x}\bigg|_{C(x)} = O(\epsilon^2)$$

$$\frac{\partial u_0}{\partial y}\bigg|_{C(x)} = \frac{\partial u_0}{\partial y}\bigg|_{y=0} + C(x)\frac{\partial^2 u_0}{\partial y^2}\bigg|_{y=0} + O(\epsilon^2)$$

$$C'(x)\frac{\partial u_0}{\partial x}\bigg|_{C(x)} = C'(x)\frac{\partial u_0}{\partial x}\bigg|_{y=0} + C(x)C'(x)\frac{\partial^2 u_0}{\partial x \partial y}\bigg|_{y=0} + O(\epsilon^2) = C'(x)\frac{\partial u_0}{\partial x}\bigg|_{y=0} + O(\epsilon^2)$$

$$\frac{\partial \tilde{u}_2}{\partial y}\bigg|_{C(x)} = \frac{\partial \tilde{u}_2}{\partial y}\bigg|_{y=0} + \frac{\partial \tilde{u}_2}{\partial y}\bigg|_{y=0} + O(\epsilon^2) = \frac{\partial \tilde{u}_2}{\partial y}\bigg|_{y=0} + O(\epsilon^2)$$

$$C'(x)\frac{\partial \tilde{u}_2}{\partial x}\bigg|_{C(x)} = O(\epsilon^2)$$

Replacing each term in equation (5.5) with its linearization approximation gives

$$k_1\frac{\partial \tilde{u}_1}{\partial y}\bigg|_{y=0} = (k_2 - k_1)\left(\frac{\partial u_0}{\partial y}\bigg|_{y=0} + C(x)\frac{\partial^2 u_0}{\partial y^2}\bigg|_{y=0} + C'(x)\frac{\partial u_0}{\partial x}\bigg|_{y=0}\right) + k_2\frac{\partial \tilde{u}_2}{\partial y}\bigg|_{y=0}$$

However, based on our definitions of u_0 and u_2, we know that

$$\frac{\partial u_0}{\partial y}\bigg|_{y=0} = \frac{\partial \tilde{u}_2}{\partial y}\bigg|_{y=0} = 0$$

resulting in our fully linearized result for $\tilde{u}_1$:

$$\frac{\partial \tilde{u}_1}{\partial y}\bigg|_{y=0} = \frac{k_2 - k_1}{k_1}\left(C(x)\frac{\partial^2 u_0}{\partial y^2}\bigg|_{y=0} + C'(x)\frac{\partial u_0}{\partial x}\bigg|_{y=0}\right) \tag{5.6}$$

The function $\tilde{u}_1$ is thus uniquely defined as the solution to equations (5.1)-(5.3) along with the boundary condition (5.6). Now that $\tilde{u}_1$ has been fully defined and linearized, we need to use it to approximate the curve $C(x)$.

5.2   Green's Identity and Integration

In order to use the linearized relation for reconstructing $C(x)$ from boundary data, we employed the so-called "Reciprocity Gap" approach, which is an application of Green's Identities to this specific problem. The idea is to extract information about C from boundary data by integrating u by parts against strategically chosen test functions. We begin with

Theorem 5.1 (Green's Second Identity). For any bounded region $D \subset \mathbb{R}^2$ with piecewise smooth boundary ∂D, and any two functions $u, v \in C^2(\bar{D})$, we have

$$\int_D (u\nabla^2 v - v\nabla^2 u) \, dA = \int_{\partial D} \left(u\frac{\partial v}{\partial \vec{n}} - v\frac{\partial u}{\partial \vec{n}} \right) \, ds$$

We want to use Green's Identity, combined with what we know about both the interior and the boundary of Ω in order to generate an approximation for the function $C(x)$.

In what follows we will use a collection of "test functions" ϕ_k, $1 \le k \le M$. Each test function ϕ_k will satisfy (see Section 5.2.1 for the construction of specific ϕ_k)

$$\frac{\partial \phi_k}{\partial t} + \alpha_1 \nabla^2 \phi_k = 0 \text{ on } \Omega$$

$$\frac{\partial \phi_k}{\partial \vec{n}} = 0 \text{ on } y = 0, \ x = 0, \text{ and } x = L$$

$$\phi_k(x, y, T) = 0 \text{ on } \Omega$$

We start with the equation

$$\int_0^T \int_\Omega \phi_k \left(\frac{\partial \tilde{u}_1}{\partial t} - \alpha_1 \nabla^2 \tilde{u}_1 \right) \, dA \, dt = 0$$

which is identically zero because the inside of the integral is the heat operator and $\tilde{u}_1$ satisfies the heat equation on all of Ω. Integrating the first term by parts in time gives

$$\int_\Omega \tilde{u}_1 \phi_k \big|_0^T - \int_0^T \tilde{u}_1 \frac{\partial \phi_k}{\partial t} + \phi_k \nabla^2 \tilde{u}_1 \, dt \, dA = 0$$

Since $\tilde{u}_1(x, y, 0) = \phi_k(x, y, T) = 0$, the first term goes to zero, and since ϕ_k solves the backwards heat equation

$$-\alpha_1 \Delta \phi_k = \frac{\partial \phi_k}{\partial t}$$

we get

$$\int_0^T \int_\Omega \phi_k \alpha_1 \nabla^2 \tilde{u}_1 - \tilde{u}_1 \alpha_1 \nabla^2 \phi_k \, dx \, dy \, dt = 0 \tag{5.7}$$

By Green's Second Identity, after canceling the α_1, the integral becomes

$$\int_0^T \int_{\partial \Omega} (\phi_k \frac{\partial \tilde{u}_1}{\partial \vec{n}} - \tilde{u}_1 \frac{\partial \phi_k}{\partial \vec{n}}) \, ds \, dt = 0$$

On the side boundaries, the normal derivatives are zero. Similarly, on the bottom boundary $(y = 0)$, $\frac{\partial \phi_k}{\partial \vec{n}} = 0$, and on the top boundary $(y = 1)$, $\frac{\partial \tilde{u}_1}{\partial \vec{n}} = 0$. Plugging these in reduces the above integrals to

$$\int_0^T \int_{top} \tilde{u}_1 \frac{\partial \phi_k}{\partial \vec{n}} \bigg|_{y=1} \, dt \, ds + \int_0^T \int_{bottom} \phi_k \frac{\partial \tilde{u}_1}{\partial \vec{n}} \bigg|_{y=0} \, dt \, ds = 0$$

In order to take the outward normal from Green's Identity, we have $d\vec{n} = dy$ on $y = 1$ and $d\vec{n} = -dy$ on $y = 0$, with $ds = dx$ in both cases. Using this, we can remove the arc length integrals to obtain

$$\int_0^L \int_0^T \tilde{u}_1 \frac{\partial \phi_k}{\partial y} \bigg|_{y=1} \, dt \, dx = \int_0^L \int_0^T \phi_k \frac{\partial \tilde{u}_1}{\partial y} \bigg|_{y=0} \, dt \, dx$$

We can replace the derivative on the right side of the equation with equation (5.6) to obtain

$$\int_0^L \int_0^T \tilde{u}_1 \frac{\partial \phi_k}{\partial \vec{n}} \bigg|_{y=1} dt\ dx = \int_0^L \int_0^T \phi_k \frac{k_2 - k_1}{k_1} \left[\frac{\partial}{\partial x} \left(-C(x) \frac{\partial u_0}{\partial x} \bigg|_{x,0,t} \right) + C(x) \frac{1}{\alpha_1} \frac{\partial u_0}{\partial t} \right] dt\ dx \qquad (5.8)$$

where we know everything on the right side of the equation, either from the data, or from our choice of ϕ_k. Therefore, we define

$$RG_k = \int_0^L \int_0^T \tilde{u}_1 \frac{\partial \phi_k}{\partial \vec{n}} \bigg|_{y=1} dt\ dx. \qquad (5.9)$$

Note that RG_k can be computed from the measured boundary data.

We can now write equation (5.8)

$$RG_k = \int_0^L \int_0^T \phi_k \frac{k_2 - k_1}{k_1} \left[\frac{\partial}{\partial x} \left(-C(x) \frac{\partial u_0}{\partial x} \bigg|_{x,0,t} \right) + C(x) \frac{1}{\alpha_1} \frac{\partial u_0}{\partial t} \right] dt\ dx$$

Integrating the first term by parts in x, gives

$$RG_k = \frac{k_2 - k_1}{k_1} \int_0^T \phi_k \frac{\partial \tilde{u}_1}{\partial x} C(x) \bigg|_0^L - \int_0^L \left[C(x) \frac{\partial u_0}{\partial x} \frac{\partial \phi}{\partial x} + C(x) \frac{1}{\alpha_1} \frac{\partial u_0}{\partial t} \right] dt\ dx$$

Since $\frac{\partial u_0}{\partial x}$ is zero at $x = 0$ and $x = L$, the evaluated term goes to zero and we get

$$RG_k = \int_0^L \int_0^T \frac{k_1 - k_2}{k_1} C(x) \left[\frac{\partial \phi_k}{\partial x} \frac{\partial u_0}{\partial x} + \phi_k \frac{1}{\alpha_1} \frac{\partial u_0}{\partial t} \right] dt\ dx$$

Finally, integrating the second term by parts in time, and plugging in $u(x, y, 0) = \phi_k(x, y, T) = 0$ gives a final equation as

$$RG_k = \int_0^L C(x) \left[\int_0^T \frac{k_1 - k_2}{k_1} \left(\frac{\partial \phi_k}{\partial x} \frac{\partial u_0}{\partial x} - \frac{1}{\alpha_1} u_0 \frac{\partial \phi_k}{\partial t} \right) dt \right] dx \qquad (5.10)$$

where we will define

$$w_k(x) := \int_0^T \frac{k_1 - k_2}{k_1} \left(\frac{\partial \phi_k}{\partial x} \frac{\partial u_0}{\partial x} - \frac{1}{\alpha_1} u_0 \frac{\partial \phi_k}{\partial t} \right) dt \qquad (5.11)$$

The situation can be summarized by noting that equations (5.10) and (5.11) yield

$$RG_k = \int_0^L C(x) w_k(x)\ dx \qquad (5.12)$$

where we know the numbers RG_k and functions $w_k(x)$ for $k = 1$ to $k = M$. Our goal is to use the system (5.12) to estimate $C(x)$. We approach the problem by constructing the function $C(x)$ that is consistent with the equations (5.12) and has minimal L^2 norm; see Section 5.3.

5.2.1   Generation of Test Functions

We want to numerically generate test functions ϕ_k such that

$$\frac{\partial \phi_k}{\partial t} + \alpha_1 \nabla^2 \phi_k = 0 \text{ on } \Omega$$

$$\frac{\partial \phi_k}{\partial \vec{n}} = 0 \text{ on } y = 0,\ x = 0,\ \text{and } x = L$$

$$\phi_k(x, y, T) = 0 \text{ on } \Omega$$

In order to do this, we look for a related function ψ_k, which satisfies

$$\frac{\partial \psi_k}{\partial t} - \alpha_1 \nabla^2 \psi_k = 0 \text{ on } \Omega$$

$$\frac{\partial \psi_k}{\partial \vec{n}} = 0 \text{ on } y = 0, \, x = 0, \text{ and } x = L$$

$$\psi_k(x, y, 0) = 0 \text{ on } \Omega$$

Then, letting $\phi_k(x, y, t) = \psi_k(x, y, T - t)$ will give the desired functions ϕ_k. In order to find ψ, we will utilize the Green's Function for heat, and model a situation with point heat sources.

The Green's Function for heat is

$$K_{(x_0, y_0)}(x, y, t) = \frac{1}{4\pi\alpha t} e^{-\frac{(x-x_0)^2 + (y-y_0)^2}{4\alpha t}}$$

which satisfies

$$\frac{\partial K}{\partial t} - \alpha \nabla^2 K = 0 \text{ on } \mathbb{R}^2$$

$$K(x, y, 0) = \delta(x - x_0, y - y_0)$$

where δ is the Kronecker delta function. This function models the spread of a point impulse heat source at the point (x_0, y_0).

By Duhamel's Principle, and by carrying out the calculations, it can be shown that

$$\int_0^t K_{(x_0, y_0)}(x, y, \tau) \, d\tau$$

is also a solution to the heat equation with zero initial condition. It models the spread of a constant heat source at the point (x_0, y_0). Since the initial condition is zero, any sum of these integrals will also solve the heat equation with zero initial condition.

In order to specify the zero Neumann data conditions for ψ, we need to use the method of images. The idea is if we make the array of point sources symmetric around some line, then heat has no reason to flow across the line, resulting in zero normal derivative on that line . Take an initial point (x_0, y_0), and reflect it over $x = 0$ to get $(x_0, -y_0)$. These two point sources together form a temperature profile with zero Neumann data at $x = 0$. Since we also want zero Neumann Data at $x = L$, we reflect these two points over that line, but to maintain symmetry about $x = 0$, we also reflect over $x = -L$. Repeating this process gives a set of points of the form $(x_0, 2jL - y_0)$ and $(x_0, 2jL + y_0)$ for $j \in \{-N, ..., N\}$. This is demonstrated in Figure 5.1 below
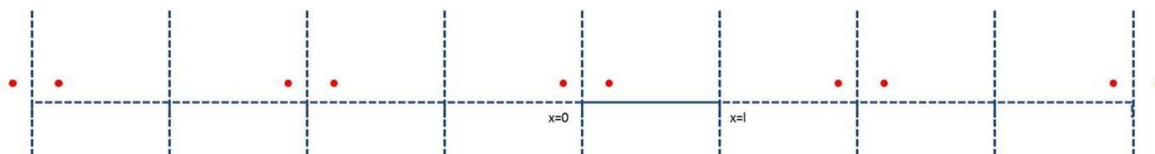

Figure 5.1: Set of Reflections in the X-Direction for the Method of Images

This set of points is not completely symmetric about the line $x = L$, but the variance from symmetry is 4 point sources more than NL units away. Since the heat equation drops off as distance squared, these points

have almost no effect on the heat flux at $x = L$, so we can take that flux to be approximately zero. The equation describing the temperature profile in this situation is

$$\sum_{j=-N}^{N} \int_0^t K_{(2jl+x_0,y_0)}(x,y,\tau) + K_{(2kl-x_0,y_0)}(x,y,\tau) \ d\tau$$

Finally, to ensure zero Neumann data at $y = 0$, we reflect all of these points over the line $y = 0$, giving a diagram similar to that in Figure 5.2 below.
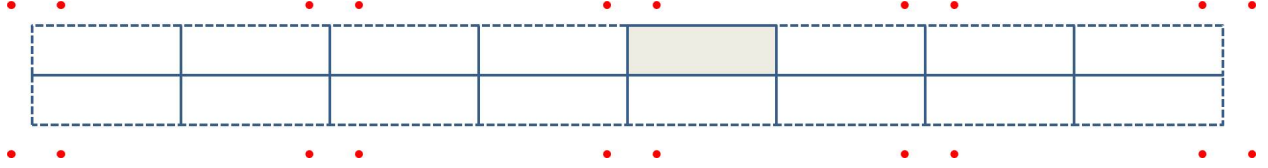

Figure 5.2: Full Set of Reflections for the Method of Images

The equation describing the temperature profile given these point sources, which satisfies the heat equation, zero initial conditions, and the desired Neumann boundary conditions is

$$\psi(x,y,t) = \sum_{j=-N}^{N} \int_0^t K_{(2jl+x_0,y_0)}(x,y,\tau) + K_{(2kl-x_0,y_0)}(x,y,\tau) + K_{(2jl+x_0,-y_0)}(x,y,\tau) + K_{(2kl-x_0,-y_0)}(x,y,\tau) \ d\tau$$

Defining $\phi(x,y,t) = \psi(x,y,T-t)$ satisfies all the desired conditions. The necessary derivatives were computed analytically and then used in the MATLAB code to create the numerical ϕ functions. In order to generate the set of functions used in the calculations, we defined the set of points for the first heat source, $(x_0,y_0)_k$ where $x_0 \in \{1,...,9\}$ and $y_0 \in \{1.1,...,1.5\}$. Working out the derivatives for ϕ gives:

$$\phi_{x_0,y_0} = \int_0^{T-t} \frac{1}{4\pi\alpha\tau} \left(e^{-\frac{(y-y_0)^2}{4\alpha\tau}} + e^{-\frac{(y+y_0)^2}{4\alpha\tau}} \right) \sum_{j=-N}^{N} \left(e^{-\frac{(x-(2jL+x_0))^2}{4\alpha\tau}} + e^{-\frac{(x-(2jL-x_0))^2}{4\alpha\tau}} \right)$$

$$\left.\frac{\partial\phi}{\partial t}\right|_{y=0} = \frac{-1}{2\pi\alpha(T-t)} e^{\frac{y_0^2}{4\alpha(T-t)}} \sum_{j=-N}^{N} \left(e^{-\frac{(x-(2jL+x_0))^2}{4\alpha(T-t)}} + e^{-\frac{(x-(2jL-x_0))^2}{4\alpha(T-t)}} \right)$$

$$\left.\frac{\partial\phi}{\partial x}\right|_{y=0} = \frac{-e^{\frac{-y_0^2}{4\alpha(T-t)}}}{\pi\alpha} \sum_{j=-N}^{N} \left(\frac{x-(2jL+x_0)}{(x-(2jL+x_0))^2+y_0^2} e^{-\frac{(x-(2jL+x_0))^2}{4\alpha(T-t)}} + \frac{x-(2jL-x_0)}{(x-(2jL-x_0))^2+y_0^2} e^{-\frac{(x-(2jL-x_0))^2}{4\alpha(T-t)}} \right)$$

$$\left.\frac{\partial\phi}{\partial y}\right|_{y=1} = \frac{-1}{2\pi\alpha} \sum_{j=-N}^{N} \left[(1-y_0)e^{-\frac{(1-y_0)^2}{4\alpha(T-t)}} \left(\frac{e^{-\frac{(x-(2jL+x_0))^2}{4\alpha(T-t)}}}{(x-(2kL+x_0))^2+(1-y_0)^2} + \frac{e^{-\frac{(x-(2jL-x_0))^2}{4\alpha(T-t)}}}{(x-(2kL-x_0))^2+(1-y_0)^2} \right) \right.$$
$$\left. + (1+y_0)e^{-\frac{(1+y_0)^2}{4\alpha(T-t)}} \left(\frac{e^{-\frac{(x-(2jL+x_0))^2}{4\alpha(T-t)}}}{(x-(2kL+x_0))^2+(1+y_0)^2} + \frac{e^{-\frac{(x-(2jL-x_0))^2}{4\alpha(T-t)}}}{(x-(2kL-x_0))^2+(1+y_0)^2} \right) \right]$$

5.3   Finding $C(x)$: Least 2-Norm

From Section 5.2, we are looking for a solution to the system of equations

$$RG_k = \int_0^L C(x)w_k(x) \ dx \tag{5.13}$$

for $1 \leq k \leq M$. In order to specify a single solution, we look for the function $C(x)$ with the smallest L^2 norm. As shown in [1], the approximation for the function $C(x)$ with this property must be a linear combination of the w_k functions, or

$$C(x) = \sum_{i=1}^{M} \lambda_i w_i(x)$$

Plugging this into equation (5.13) gives

$$
\begin{aligned}
RG_k &= \int_0^L \sum_{i=1}^{M} \lambda_i w_i(x) w_k(x) \ dx \\
&= \sum_{i=1}^{M} \lambda_i \int_0^L w_i(x) w_k(x) \ dx
\end{aligned}
$$

Defining the coefficient matrix $\mathbf{B}$ by

$$\mathbf{B}_{ik} = \int_0^L w_i(x) w_k(x) \ dx$$

gives

$$RG_k = \sum_{i=1}^{N} \mathbf{B}_{ik} \lambda_i$$

for $1 \leq k \leq M$, or

$$\vec{RG} = \mathbf{B}\vec{\lambda}$$

Then calculating λ via $\mathbf{B}^{-1}\vec{RG}$ and letting

$$C(x) = \sum_{i=1}^{M} \lambda_i w_i(x)$$

gives an approximation for the corrosion profile in the system.

6   Refining the Solution

Performing the procedure in Section 5.3 and using it to approximate the function $C(x)$ gives a graph similar to that in Figure 6.1 below.
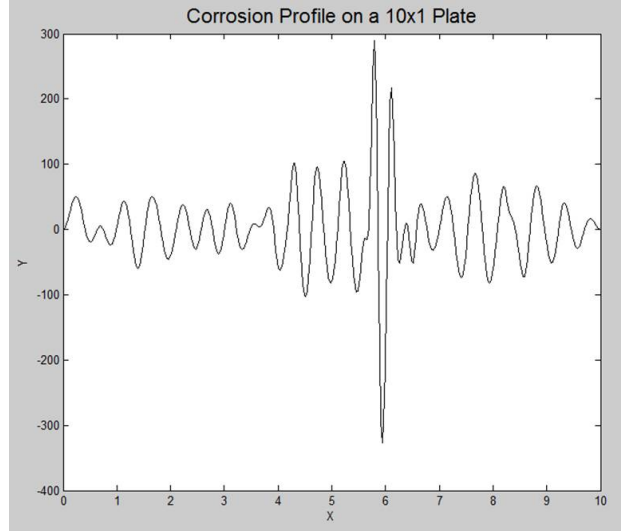

Figure 6.1: Graph of the Raw Approximation of $C(x)$

Since our plate is only supposed to be of thickness 1, this is not a feasible result. We need some way to refine our solution to get something that more closely matches the curve $C(x)$.

Our first step was to change the $w_k(x)$ functions used. For a reason which was discovered later (see Section 8 for the reason), the functions

$$w_k(x) = \frac{\partial u_0}{\partial x} \frac{\partial \phi_k}{\partial x}$$

worked much better for determining the corrosion profile $C(x)$ than the w functions defined in Section 5.2. Using those, we were able to move forward to our other two methods of improving the solution, regularization and moving the input heat flux.

6.1   Regularization

The high amplitude values and oscillatory nature of the approximation of $C(x)$ in Figure 6.1 suggest that this inverse problem is ill-posed. In our case, this presents itself in the fact that the matrix $\mathbf{B}$ is ill-conditioned: it has singular values that are very close to zero. Following the analysis in [1], we implemented a regularization process using the singular value decomposition of the matrix $\mathbf{B}$.

We define ϵ_∞ as the maximum absolute error in the system and ϵ_{thresh} as the relative noise on a 0 to 1 scale. These two parameters are variable and can be changed to adjust the amount of regularization applied to the system. We then set

$$\epsilon_d = \frac{\sqrt{M} h_w C_{max} \epsilon_\infty}{\epsilon_{thresh}}$$

where M is the number of test functions used, h_w is the average maximum value of the w_k functions, and

$$C_{max} = \max_k \int_0^L \int_0^T \left| \frac{\partial \phi_k}{\partial y} \right| \, dt \, dx$$

15

With this ϵ_d defined, we now look at the Singular Value Decomposition of $\mathbf{B}$. If any of the singular values are less than ϵ_d, we set the corresponding value in $\mathbf{B}^{-1}$ to zero, eliminating the high magnitude terms in $\mathbf{B}^{-1}$. Computing the corrosion profile with this modified $\mathbf{B}^{-1}$ matrix gives much better results for the corrosion profile. Figure 6.2 below shows how the process of regularization changes the approximation for the curve $C(x)$ for different values of ϵ_∞. In this figure, the solid line shows the reconstructed profile after the stated amount of regularization, and the dashed line is the actual corrosion profile that was used to generate the data used for RG_k.
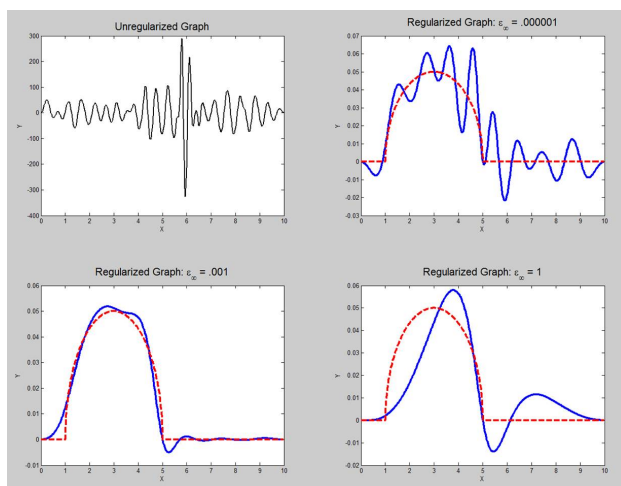

Figure 6.2: Approximations of the Curve $C(x)$ at Different Regularization Levels

As can be seen in the above figure, even a small amount of regularization helps a lot in seeing what the corrosion profile looks like. However, it is also possible to over-regularize the process, which is what has happened in the bottom right graph. With this high of a regularization coefficient, there are very few singular values left in $\mathbf{B}$, which eliminates most of the w_k functions from the final approximation of $C(x)$. With so few functions, it is impossible for the system to accurately model the curve. Since $\epsilon_\infty = 0.001$ worked well in this trial, as well as many others, that is the value used for the majority of our regularization calculations.

6.2   Variation of Input Flux

Even though our new w_k functions give a much better approximation for the curve $C(x)$, there is one major problem with them. The function will be exactly zero wherever $\frac{\partial u_0}{\partial x} = 0$. The most notable point where this is going to occur is directly under the symmetry point of the heat source. The problem with this is that no matter the choice of ϕ_k, all of the w_k functions are zero at the center of the heat source, meaning that the approximation for $C(x)$ will also go to zero there. There is no way for this method to model any corrosion at this point, or in the surrounding area because the functions are continuous. Figures 6.3 and 6.4 below show the w_k functions for a heat flux centered at $x = 5$ and the resulting $C(x)$ profile, illustrating the problem.
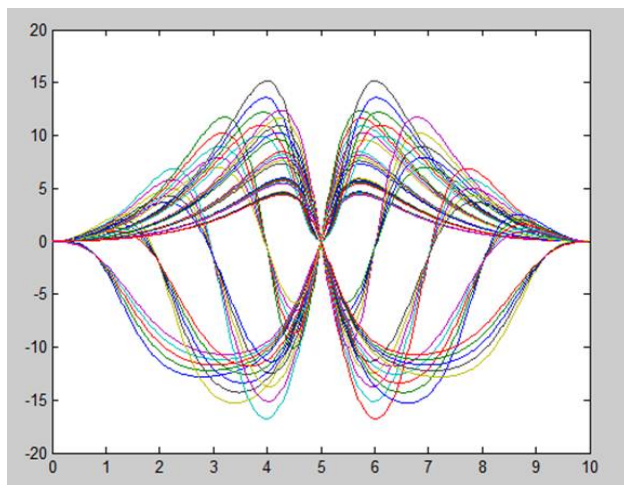
Figure 6.3: w_k Functions for a Flux Centered at 5
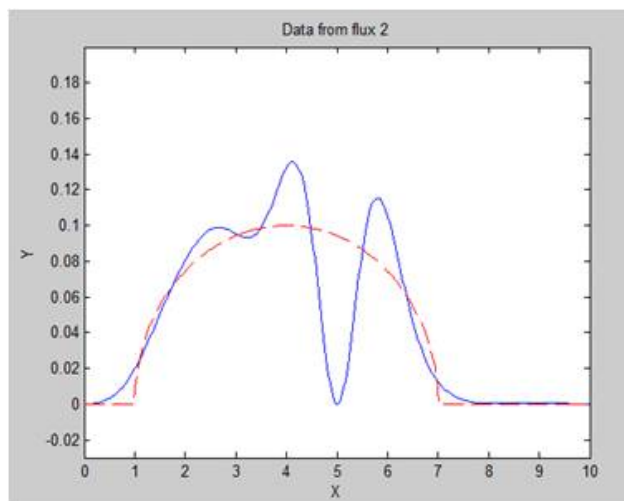
Figure 6.4: Corrosion Profile Using these w_k Functions, with an Error at 5

The way we resolved this problem was by moving the input heat flux around the plate and meshing the different solutions together. Using a time-dependent flux to move it around the plate was not met with success, so our solution was to run three separate trials for each corrosion profile. Each corrosion profile was tested with a heat flux centered at 2, 5, and 8. The three different $C(x)$ solutions were then meshed together using a set of conditional statements, ignoring each function in the interval of radius 2 around the center of the flux, with a smoothing region of .5. This set of parameters led to a smooth representation of the curve $C(x)$ that followed the actual profile fairly closely over the entire plate. Multiple images showing all three of the approximated corrosion profiles with different input fluxes, as well as the average curve, can be seen in Section 7.

7   Results

Here we show four separate numerical examples displaying the ability of this reconstruction method to match size, location, and complication of the corrosion profile originally entered. These reconstructions were generated by setting up the corrosion profile geometry in a COMSOL file, obtaining data on the top boundary for the plate with corrosion. Another COMSOL file was used to model a plate made of the same material that had not been corroded. Data was collected from both the top and bottom boundary of this model. These files were then imported into MATLAB and the approximation for $C(x)$ was made by RunFile.m, which can be seen in the appendix to the report. For all of the results, the solid line is the reconstructed profile, and the dashed line is the actual profile programmed into COMSOL. The graph in the top left is the average of the other three corrosion profiles, while the other graphs represent the individual reconstructions, with fluxes centered at 2, 5, and 8 respectively. For all of these reconstructions, the following parameters were used:

$$k_1 = 1$$
$$\alpha_1 = 1$$
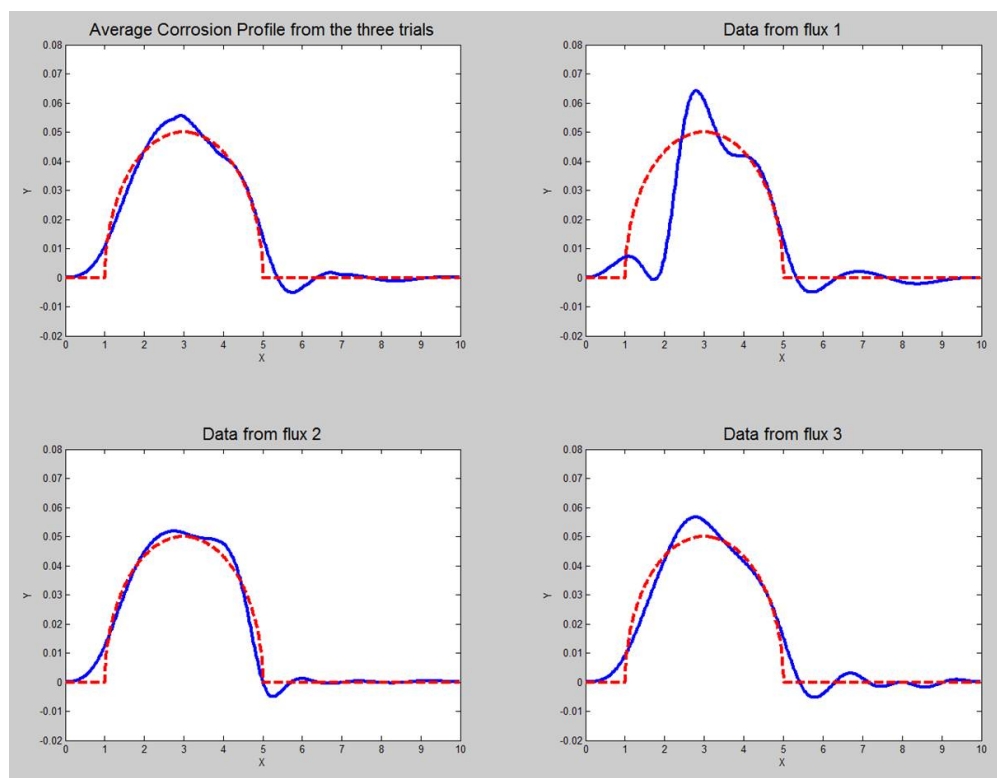$$k_2 = 0.1$$
$$\alpha_2 = 0.1$$

Figure 7.1: Reconstruction of a Small Corrosion Profile on the Left of the Plate
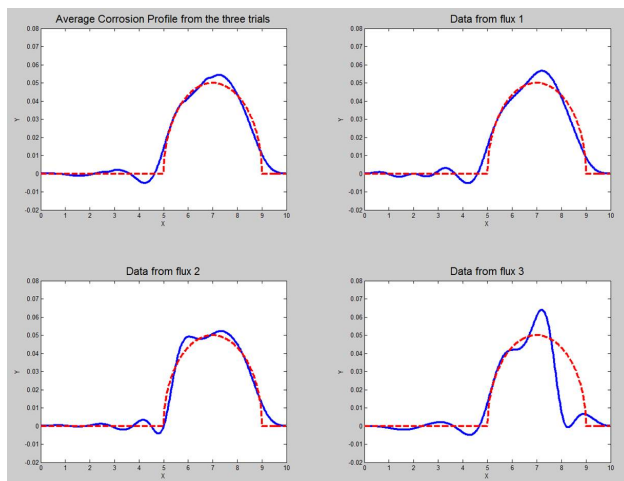
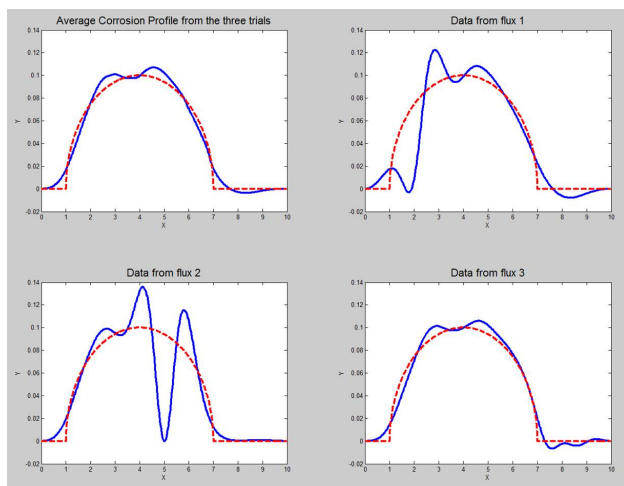Figure 7.2: Reconstruction of a Small Corrosion Profile on the Right of the Plate

Figure 7.3: Reconstruction of a Longer Corrosion Profile Centered on the Left
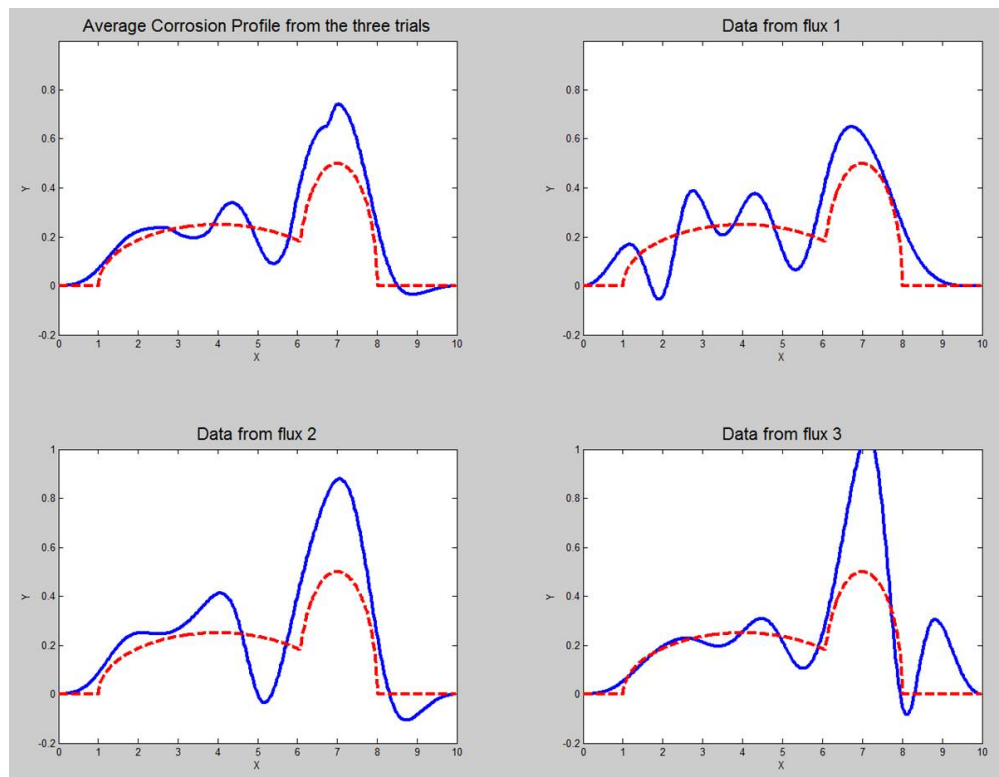
Figure 7.4: Reconstruction of a Large Corrosion Profile

8   New Linearization

A new linearization method was developed in order to attempt to justify the removal of the time derivative term from the w_k functions in the results that were reported in Section 7.
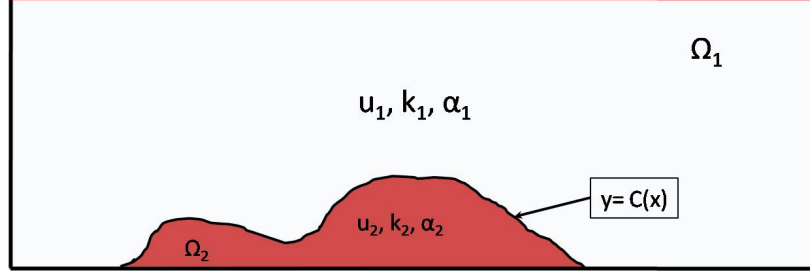
Figure 8.1: General Corrosion Profile Setup

Assume we have the situation above with insulated sides. We then have u_1 satisfying

$$\frac{\partial u_1}{\partial t} - \alpha_1 \nabla^2 u_1 \ = \ 0 \text{ on } \Omega_1$$

$$\frac{\partial u_1}{\partial x} \ = \ 0 \text{ on } x = 0, \ x = L$$

$$\frac{\partial u_1}{\partial y} \ = \ g(x) \text{ on } y = 1$$

$$u_1(x, y, 0) \ = \ 0 \text{ on } \Omega_1$$

while u_2 satisfies

$$\frac{\partial u_2}{\partial t} - \alpha_2 \nabla^2 u_2 \ = \ 0 \text{ on } \Omega_2$$

$$\frac{\partial u_2}{\partial x} \ = \ 0 \text{ on } x = 0, \ x = L$$

$$\frac{\partial u_2}{\partial y} \ = \ 0 \text{ on } y = 0$$

$$u_2(x, y, 0) \ = \ 0 \text{ on } \Omega_2.$$

We also have the continuity conditions

$$u_1 \ = \ u_2 \text{ on } C(x)$$

$$k_1 \frac{\partial u_1}{\partial \vec{n}} \ = \ k_2 \frac{\partial u_2}{\partial \vec{n}} \text{ on } C(x)$$

at the interface $y = C(x)$.

Assume that we also have a test function ϕ (see Section 5.2.1) satisfying

$$\frac{\partial \phi}{\partial t} + \alpha_1 \nabla^2 \phi \ = \ 0 \text{ on } \Omega$$

$$\frac{\partial \phi}{\partial x} \ = \ 0 \text{ on } x = 0, \ x = L$$

$$\frac{\partial \phi}{\partial y} \ = \ 0 \text{ on } y = 0$$

$$\phi(x, y, T) \ = \ 0 \text{ on } \Omega$$

21

8.1 Computations

We start with the equation

$$\int_0^T \int_{\Omega_1} u_1 \left(\frac{\partial \phi}{\partial t} + \alpha_1 \nabla^2 \phi \right) \, dA \, dt \;\; = \;\; 0$$

or

$$\int_0^T \int_{\Omega_1} u_1 \frac{\partial \phi}{\partial t} \, dA \, dt + \alpha_1 \int_0^T \int_{\Omega_1} u_1 \nabla^2 \phi \, dA \, dt \;\; = \;\; 0$$

Integrating the first term by parts in time and using Green's Identity on the second term gives

$$\int_{\Omega_1} u_1 \phi \big|_0^T - \int_0^T \phi \frac{\partial u_1}{\partial t} \, dt \, dA + \alpha_1 \int_0^T \int_{\Omega_1} \phi \nabla^2 u_1 \, dA + \int_{\partial \Omega_1} \left(u_1 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_1}{\partial \vec{n}} \right) \, ds \, dt \;\; = \;\; 0$$

or

$$\int_{\Omega_1} u_1 \phi \big|_0^T \, dA - \int_{\Omega_1} \int_0^T \phi \left(\frac{\partial u_1}{\partial t} - \alpha_1 \nabla^2 u_1 \right) \, dt \, dA + \alpha_1 \int_0^T \int_{\partial \Omega_1} \left(u_1 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_1}{\partial \vec{n}} \right) \, ds \, dt \;\; = \;\; 0$$

The first two terms in the above expression are zero because u_1 solves the heat equation and vanishes at $t = 0$, while ϕ vanishes at $t = T$. Canceling the α_1 gives:

$$\int_0^T \int_{\partial \Omega_1} \left(u_1 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_1}{\partial \vec{n}} \right) \, ds \, dt = 0$$

or, factoring in the boundary of Ω_1 and the conditions on the functions there,

$$\int_0^T \int_{top} \left(u_1 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_1}{\partial \vec{n}} \right) \, ds \, dt + \int_0^T \int_{C(x)} \left(u_1 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_1}{\partial \vec{n}} \right) \, ds \, dt = 0 \tag{8.1}$$

where $C(x)$ is defined with the downward normal.

Similarly, we can look at the region Ω_2 and start with

$$\int_0^T \int_{\Omega_2} u_2 \frac{\partial \phi}{\partial t} \, dA \, dt + \alpha_1 \int_0^T \int_{\Omega_2} u_2 \nabla^2 \phi \, dA \, dt \;\; = \;\; 0$$

Integrating the first term by parts in time and using Green's Identity on the second term gives:

$$\int_{\Omega_2} u_2 \phi \big|_0^T - \int_0^T \phi \frac{\partial u_2}{\partial t} \, dt \, dA + \alpha_1 \int_0^T \int_{\Omega_2} \phi \nabla^2 u_2 \, dA + \int_{\partial \Omega_2} \left(u_2 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_2}{\partial \vec{n}} \right) \, ds \, dt = 0$$

The first term is zero because ϕ and u_2 vanish at the endpoints in time. Since u_2 solves the heat equation in Ω_2, we know that

$$\frac{\partial u_2}{\partial t} = \alpha_2 \nabla^2 u_2$$

Plugging this in above gives

$$-\alpha_2 \int_{\Omega_2} \int_0^T \phi \nabla^2 u_2 \, dt \, dA + \alpha_1 \int_0^T \int_{\Omega_2} \phi \nabla^2 u_2 \, dA \, dt + \alpha_1 \int_0^T \int_{\partial \Omega_2} \left(u_2 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_2}{\partial \vec{n}} \right) \, ds \, dt \;\; = \;\; 0$$

$$(\alpha_1 - \alpha_2) \int_0^T \int_{\Omega_2} \phi \nabla^2 u_2 \, dA \, dt + \alpha_1 \int_0^T \int_{\partial \Omega_2} \left(u_2 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_2}{\partial \vec{n}} \right) \, ds \, dt \;\; = \;\; 0$$

$$\frac{\alpha_1 - \alpha_2}{\alpha_1} \int_0^T \int_{\Omega_2} \phi \nabla^2 u_2 \, dA \, dt + \int_0^T \int_{\partial \Omega_2} \left(u_2 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_2}{\partial \vec{n}} \right) \, ds \, dt \;\; = \;\; 0$$

Taking the boundary of Ω_2 into consideration gives

$$\frac{\alpha_1 - \alpha_2}{\alpha_1} \int_0^T \int_{\Omega_2} \phi \nabla^2 u_2 \, dA \, dt + \int_0^T \int_{C(x)} \left(u_2 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_2}{\partial \vec{n}} \right) ds \, dt + \int_0^T \int_{bottom} \left(u_2 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_2}{\partial \vec{n}} \right) ds \, dt = 0$$

where $C(x)$ has the upward normal. Flipping the normal on $C(x)$ so it can be combined with equation (8.1) and canceling the last term since both functions have zero normal derivative at $y = 0$ gives:

$$\frac{\alpha_1 - \alpha_2}{\alpha_1} \int_0^T \int_{\Omega_2} \phi \nabla^2 u_2 \, dA \, dt - \int_0^T \int_{C(x)} \left(u_2 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_2}{\partial \vec{n}} \right) ds \, dt = 0 \tag{8.2}$$

where $C(x)$ is defined with the downward normal.

We now seek to combine equations (8.1) and (8.2) to generate a relation that will allow us to approximate the function $C(x)$. We will use the temperature profile u_2 in Ω_2 and the continuity conditions on $C(x)$ to do this.

8.2   Combining Equations

To start, modify equation (8.1) using the continuity conditions on $C(x)$, <mark>to give</mark>

$$\int_0^T \int_{top} \left(u_1 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_1}{\partial \vec{n}} \right) ds \, dt + \int_0^T \int_{C(x)} \left(u_2 \frac{\partial \phi}{\partial \vec{n}} - \frac{k_2}{k_1} \phi \frac{\partial u_2}{\partial \vec{n}} \right) ds \, dt = 0 \tag{8.3}$$

Then, adding equations (8.2) and (8.3) gives:

$$\int_0^T \int_{top} \left(u_1 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_1}{\partial \vec{n}} \right) ds \, dt + \left(1 - \frac{k_2}{k_1} \right) \int_0^T \int_{C(x)} \phi \frac{\partial u_2}{\partial \vec{n}} \, ds \, dt + \frac{\alpha_1 - \alpha_2}{\alpha_1} \int_0^T \int_{\Omega_2} \phi \nabla^2 u_2 \, dA \, dt = 0$$

Defining

$$RG(\phi) = \int_0^T \int_{top} \left(u_1 \frac{\partial \phi}{\partial \vec{n}} - \phi \frac{\partial u_1}{\partial \vec{n}} \right) ds \, dt \tag{8.4}$$

and rearranging gives

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1 \right) \int_0^T \int_{C(x)} \phi \frac{\partial u_2}{\partial \vec{n}} \, ds \, dt + \frac{\alpha_2 - \alpha_1}{\alpha_1} \int_0^T \int_{\Omega_2} \phi \nabla^2 u_2 \, dA \, dt$$

Knowing that u_2 solves the heat equation on Ω_2 and that on $C(x)$

$$\vec{n} = \frac{< C'(x), -1 >}{\sqrt{C'(x)^2 + 1}} \qquad \text{and} \qquad ds = \sqrt{C'(x)^2 + 1}$$

this expression can be written as

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1 \right) \int_0^T \int_0^L C'(x) \phi \frac{\partial u_2}{\partial x} \Big|_{C(x)} - \phi \frac{\partial u_2}{\partial y} \Big|_{C(x)} dx \, dt + \frac{\alpha_2 - \alpha_1}{\alpha_1 \alpha_2} \int_0^T \int_{\Omega_2} \phi \frac{\partial u_2}{\partial t} \, dA \, dt \tag{8.5}$$

which is the fully simplified non-linear problem.

8.3 Linearization

Now, we make certain assumptions to formally linearize the problem. Firstly, we assume

$$C(x) = \epsilon C_0(x)$$

or that the corrosion profile is very small.

The last term in equation (8.5) may be written as

$$\frac{\alpha_2 - \alpha_1}{\alpha_1 \alpha_2} \int_0^T \int_0^L \int_0^{C(x)} \phi \frac{\partial u_2}{\partial t} \; dy \; dx \; dt$$

If $C(x)$ is small, we can approximate the function $\phi \frac{\partial u_2}{\partial t}$ by a constant over the innermost integral in y. This term can then be written as approximately equal to

$$\frac{\alpha_2 - \alpha_1}{\alpha_1 \alpha_2} \int_0^T \int_0^L C(x) \left. \phi \frac{\partial u_2}{\partial t} \right|_{y=0} dx \; dt$$

where we choose to evaluate the functions at $y = 0$.

We also want to linearize the parts of the first integral in equation (8.5) about the line $y = 0$, ignoring all terms of $O(\epsilon^2)$.

$$
\begin{aligned}
\left. C'(x) \, \phi \frac{\partial u_2}{\partial x} \right|_{C(x)} &= \left. C'(x) \, \phi \frac{\partial u_2}{\partial x} \right|_{y=0} + \left. C(x) C'(x) \frac{\partial \phi}{\partial y} \frac{\partial u_2}{\partial x} \right|_{y=0} + \left. C(x) C'(x) \, \phi \frac{\partial^2 u_2}{\partial x \partial y} \right|_{y=0} \\
&= \left. C'(x) \, \phi \frac{\partial u_2}{\partial x} \right|_{y=0} + O(\epsilon^2) \\
\left. \phi \frac{\partial u_2}{\partial y} \right|_{C(x)} &= \left. \phi \frac{\partial u_2}{\partial y} \right|_{y=0} + \left. C(x) \frac{\partial \phi}{\partial y} \frac{\partial u_2}{\partial y} \right|_{y=0} + \left. C(x) \, \phi \frac{\partial^2 u_2}{\partial y^2} \right|_{y=0} + O(\epsilon^2) \\
&= 0 + 0 + \left. C(x) \, \phi \frac{\partial^2 u_2}{\partial y^2} \right|_{y=0} + O(\epsilon^2)
\end{aligned}
$$

Plugging these terms into equation (8.5) gives

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1 \right) \int_0^T \int_0^L C'(x) \left. \phi \frac{\partial u_2}{\partial x} \right|_{y=0} - \left. C(x) \, \phi \frac{\partial^2 u_2}{\partial y^2} \right|_{y=0} dx \; dt + \frac{\alpha_2 - \alpha_1}{\alpha_1 \alpha_2} \int_0^T \int_0^L C(x) \left. \phi \frac{\partial u_2}{\partial t} \right|_{y=0} dx \; dt \tag{8.6}$$

Integrating the first term in equation (8.6) by parts in x (integrating $C'(x)$) gives

$$
\begin{aligned}
RG(\phi) =\ & \left(\frac{k_2}{k_1} - 1 \right) \int_0^T \left. C(x) \phi \frac{\partial u_2}{\partial x} \right|_{x=0}^{x=L} + \int_0^L -C(x) \left. \frac{\partial \phi}{\partial x} \frac{\partial u_2}{\partial x} \right|_{y=0} - \left. C(x) \, \phi \frac{\partial^2 u_2}{\partial x^2} \right|_{y=0} - \left. C(x) \, \phi \frac{\partial^2 u_2}{\partial y^2} \right|_{y=0} dx \; dt \\
& + \frac{\alpha_2 - \alpha_1}{\alpha_1 \alpha_2} \int_0^T \int_0^L C(x) \left. \phi \frac{\partial u_2}{\partial t} \right|_{y=0} dx \; dt
\end{aligned}
$$

where the first term is zero because $\frac{\partial u_2}{\partial x} = 0$ at both sides of the rectangle. Combining the two second derivatives into a Laplacian and separating the first term of the integral gives:

$$
\begin{aligned}
RG(\phi) =\ & \left(\frac{k_2}{k_1} - 1 \right) \int_0^T \int_0^L -C(x) \left. \frac{\partial \phi}{\partial x} \frac{\partial u_2}{\partial x} \right|_{y=0} dx \; dt - \left(\frac{k_2}{k_1} - 1 \right) \int_0^T \int_0^L C(x) \left. \phi \nabla^2 u_2 \right|_{y=0} dx \; dt \\
& + \frac{\alpha_2 - \alpha_1}{\alpha_1 \alpha_2} \int_0^T \int_0^L C(x) \left. \phi \frac{\partial u_2}{\partial t} \right|_{y=0} dx \; dt
\end{aligned}
$$

Converting the second term to a time derivative by the heat equation and reorganizing some terms gives:

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^L -C(x) \left.\frac{\partial\phi}{\partial x}\frac{\partial u_2}{\partial x}\right|_{y=0} dx \; dt - \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^L C(x) \left.\frac{\phi}{\alpha_2}\frac{\partial u_2}{\partial t}\right|_{y=0} dx \; dt$$
$$+ \frac{\alpha_2 - \alpha_1}{\alpha_1} \int_0^T \int_0^L C(x) \left.\frac{\phi}{\alpha_2}\frac{\partial u_2}{\partial t}\right|_{y=0} dx \; dt$$

or

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^L -C(x) \left.\frac{\partial\phi}{\partial x}\frac{\partial u_2}{\partial x}\right|_{y=0} dx \; dt - \left[\left(\frac{k_2}{k_1} - 1\right) - \frac{\alpha_2 - \alpha_1}{\alpha_1}\right] \int_0^T \int_0^L C(x) \left.\frac{\phi}{\alpha_2}\frac{\partial u_2}{\partial t}\right|_{y=0} dx \; dt$$

Simplifying the coefficient of the second term gives

$$\left(\frac{k_2}{k_1} - 1\right) - \frac{\alpha_2 - \alpha_1}{\alpha_1} = \frac{k_2}{k_1} - 1 - \frac{\alpha_2}{\alpha_1} + 1$$
$$= \frac{k_2}{k_1} - \frac{k_2}{k_1}\frac{(\rho C_p)_1}{(\rho C_p)_2}$$
$$= \frac{k_2}{k_1}\left(1 - \frac{(\rho C_p)_1}{(\rho C_p)_2}\right)$$

Thus, our expression becomes

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^L -C(x) \left.\frac{\partial\phi}{\partial x}\frac{\partial u_2}{\partial x}\right|_{y=0} dx \; dt - \left(\frac{k_2}{k_1} - \frac{\alpha_2}{\alpha_1}\right) \int_0^T \int_0^L C(x) \left.\frac{\phi}{\alpha_2}\frac{\partial u_2}{\partial t}\right|_{y=0} dx \; dt \qquad (8.7)$$

or

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^L -C(x) \left.\frac{\partial\phi}{\partial x}\frac{\partial u_2}{\partial x}\right|_{y=0} dx \; dt - \frac{k_2}{k_1}\left(1 - \frac{(\rho C_p)_1}{(\rho C_p)_2}\right) \int_0^T \int_0^L C(x) \left.\frac{\phi}{\alpha_2}\frac{\partial u_2}{\partial t}\right|_{y=0} dx \; dt$$
$$(8.8)$$

Now, making the assumption that the temperature profile u_1 is close to the uncorroded temperature profile u_0, which is reasonable in the case that $C(x)$ is small, we get

$$u_1 = u_0 + \epsilon\tilde{u_1}$$

We evaluate this expression on the curve $C(x)$ to obtain

$$u_2|_{C(x)} = u_1|_{C(x)} = u_0|_{C(x)} + \epsilon \; \tilde{u_1}|_{C(x)}$$

Linearizing the far left and far right about $y = 0$ gives

$$u_2|_{y=0} + C(x) \left.\frac{\partial u_2}{\partial y}\right|_{y=0} + O(\epsilon^2) = u_0|_{y=0} + C(x) \left.\frac{\partial u_0}{\partial y}\right|_{y=0} + O(\epsilon^2) + \epsilon \; \tilde{u_1}|_{y=0} + O(\epsilon^2)$$

Since

$$\left.\frac{\partial u_2}{\partial y}\right|_{y=0} = \left.\frac{\partial u_0}{\partial y}\right|_{y=0} 1 = 0$$

we are left with

$$u_2|_{y=0} = u_0|_{y=0} + O(\epsilon)$$

Taking this assumption with $(\rho C_p)_1 = (\rho C_p)_2$ gives

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^L -C(x) \left.\frac{\partial\phi}{\partial x}\frac{\partial u_0}{\partial x}\right|_{y=0} dx \; dt$$

as desired. This gives the exact set of w_k functions that were used to generate the results presented in Section 7.

Integrating the last term of equation (8.7) by parts in time will generate a new set of w_k functions which can be used to numerically solve for the function $C(x)$.

$$
\begin{aligned}
RG(\phi) &= \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^L -C(x) \left.\frac{\partial \phi}{\partial x} \frac{\partial u_0}{\partial x}\right|_{y=0} dx\ dt - \left(\frac{k_2}{k_1} - \frac{\alpha_2}{\alpha_1}\right) \left[u_0 \phi \big|_0^T - \int_0^T \int_0^L C(x) \left.\frac{u_0}{\alpha_2} \frac{\partial \phi}{\partial t}\right|_{y=0} dx\ dt \right] \\
&= \left(1 - \frac{k_2}{k_1}\right) \int_0^T \int_0^L C(x) \left.\frac{\partial \phi}{\partial x} \frac{\partial u_0}{\partial x}\right|_{y=0} dx\ dt + \left(\frac{k_2}{k_1} - \frac{\alpha_2}{\alpha_1}\right) \int_0^T \int_0^L C(x) \left.\frac{u_0}{\alpha_2} \frac{\partial \phi}{\partial t}\right|_{y=0} dx\ dt \\
&= \int_0^L C(x) \left[\int_0^T \left(1 - \frac{k_2}{k_1}\right) \left.\frac{\partial \phi}{\partial x} \frac{\partial u_0}{\partial x}\right|_{y=0} + \left(\frac{k_2}{k_1} - \frac{\alpha_2}{\alpha_1}\right) \left.\frac{u_0}{\alpha_2} \frac{\partial \phi}{\partial t}\right|_{y=0} dt \right] dx
\end{aligned}
$$

and, defining

$$
w_k(x) := \int_0^T \left(1 - \frac{k_2}{k_1}\right) \left.\frac{\partial \phi_k}{\partial x} \frac{\partial u_0}{\partial x}\right|_{y=0} + \left(\frac{k_2}{k_1} - \frac{\alpha_2}{\alpha_1}\right) \left.\frac{u_0}{\alpha_2} \frac{\partial \phi_k}{\partial t}\right|_{y=0} dt
$$

we are again looking for solutions to the system of integral equations,

$$
RG(\phi_k) = \int_0^L C(x) w_k(x) dx \tag{8.9}
$$

with $1 \le k \le M$.

From equation (8.9), we can see that plugging in $C(x) = 0$ gives

$$
RG_0(\phi_k) = \int_0^T \int_{top} u_0 \frac{\partial \phi_k}{\partial \vec{n}} - \phi \frac{\partial u_0}{\partial \vec{n}}\ ds\ dt = 0 \tag{8.10}
$$

Subtracting equation (8.10) from (8.4) gives

$$
\tilde{RG}(\phi_k) = \int_0^T \int_{top} \tilde{u}_1 \frac{\partial \phi}{\partial \vec{n}}\ ds\ dt = \int_0^L C(x) w_k(x)\ dx \tag{8.11}
$$

since

$$
\left.\frac{\partial u_0}{\partial \vec{n}}\right|_{y=1} = \left.\frac{\partial u_1}{\partial \vec{n}}\right|_{y=1}
$$

where $\tilde{u}_1 = u_1 - u_0$ is the disturbance from the uncorroded temperature profile. This is the expression used to compute the function $C(x)$ in the MATLAB code.

9   New Results

See Section 7 for a description of how the profiles were reconstructed. The RunFile used to generate the profiles is almost exactly the same, except for two lines in the construction of the w_k functions. RunFile_bool_modTime contains these changes. The edited section of the code can be seen in the appendices. The set of thermal parameters used in each situation will be displayed below the graph of the results. For all of these reconstructions, the thermal parameters of material 1 are set to $k_1 = 1$ and $\alpha_1 = 1$. The regularization procedure and parameters of Section 7 were also used.
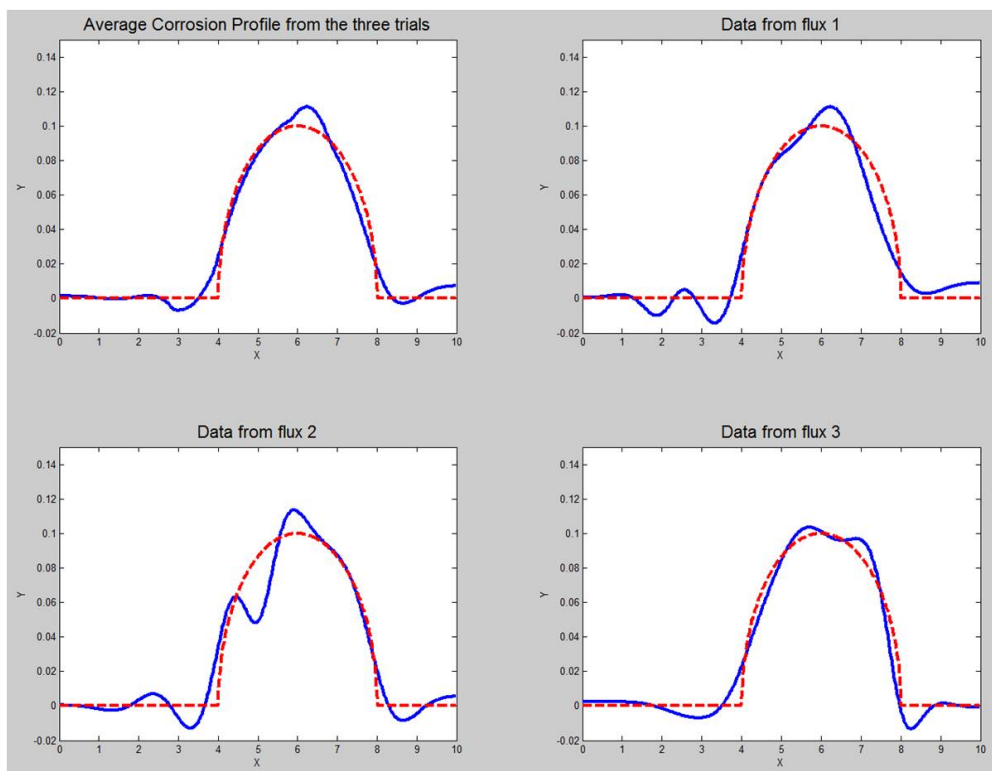
Figure 9.1: Reconstruction of Corrosion Profile with $k_2 = 0.1$, $\alpha_2 = 0.05$.
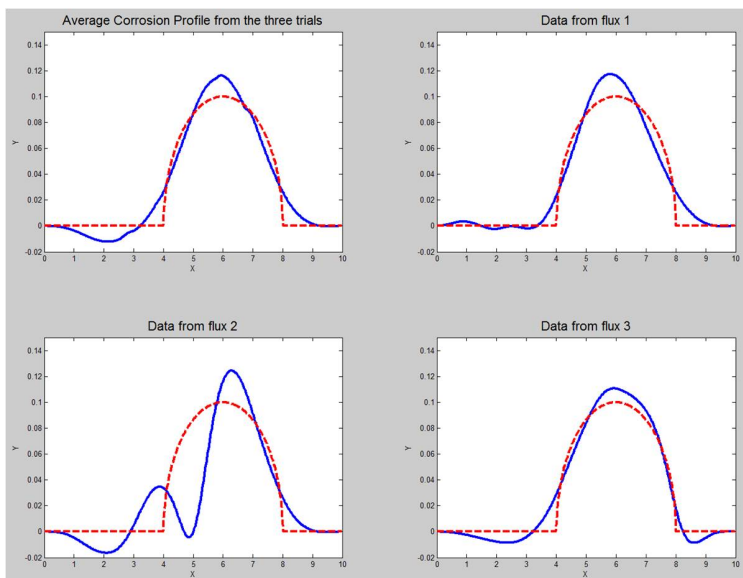
Figure 9.2: Reconstruction of Corrosion Profile with $k_2 = 0.1$, $\alpha_2 = 0.1$.

Figure 9.3: Reconstruction of Corrosion Profile with $k_2 = 0.1$, $\alpha_2 = 0.2$.

Figure 9.4: Reconstruction of Corrosion Profile with $k_2 = 1$, $\alpha_2 = 0.1$.

Figure 9.5: Reconstruction of Corrosion Profile with $k_2 = 0.1$, $\alpha_2 = 1$.

Figure 9.6: Reconstruction of Large Corrosion Profile with $k_2 = 0.1$, $\alpha_2 = 0.05$.

Figure 9.7: Reconstruction of Large Corrosion Profile with $k_2 = 0.1$, $\alpha_2 = 0.1$.

Figure 9.8: Reconstruction of Large Corrosion Profile with $k_2 = 0.1$, $\alpha_2 = 0.2$.
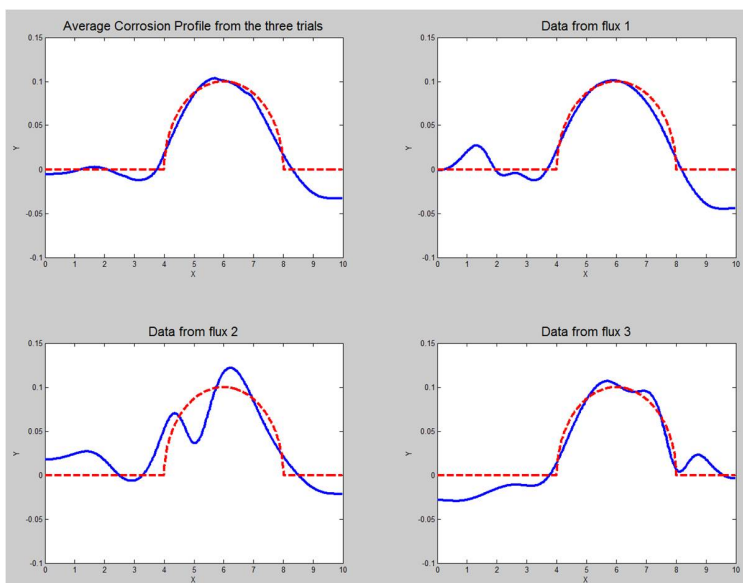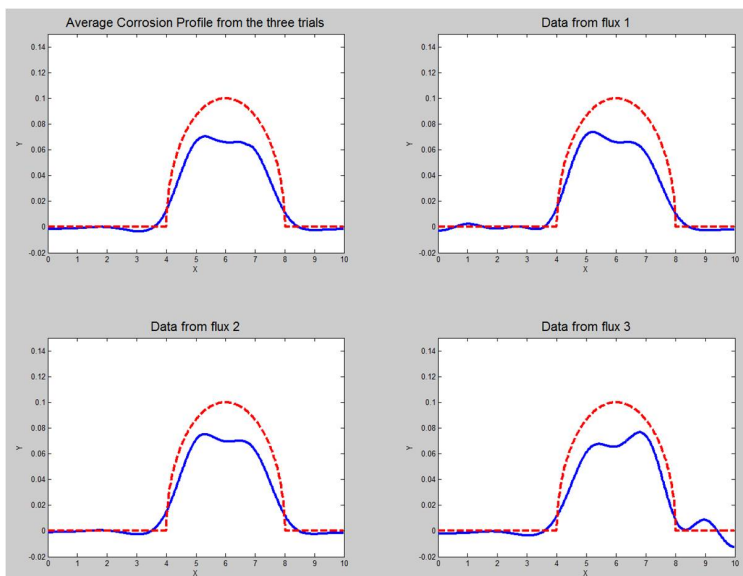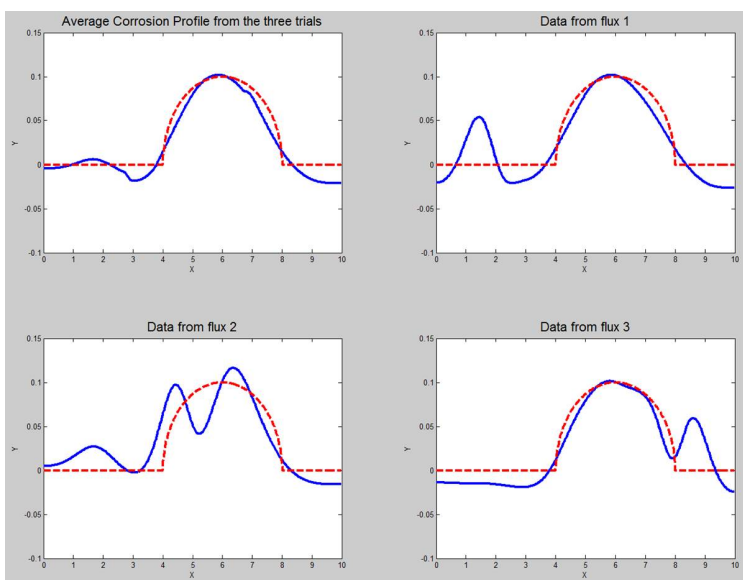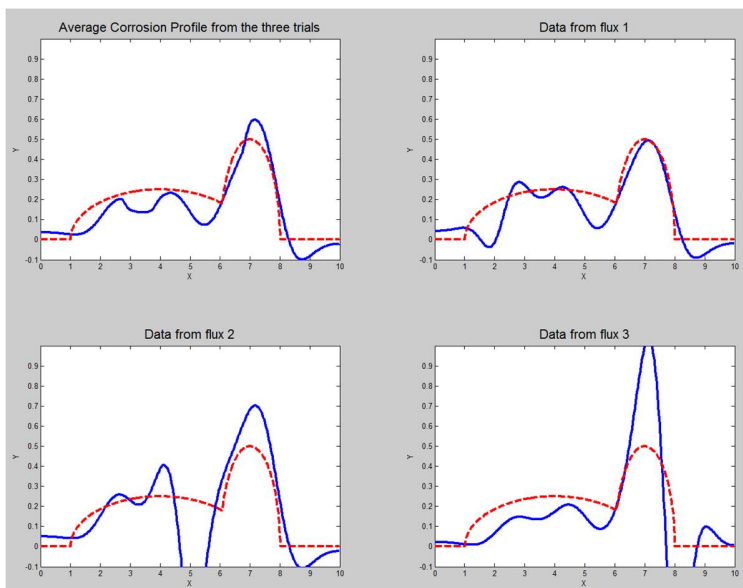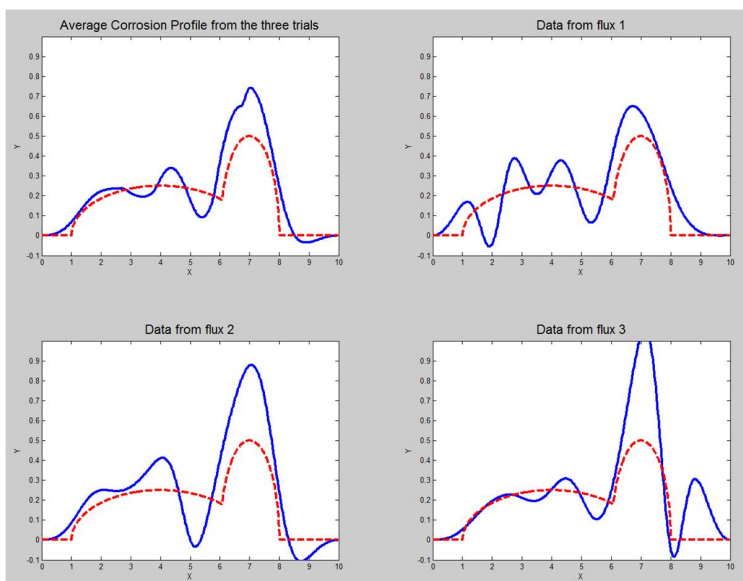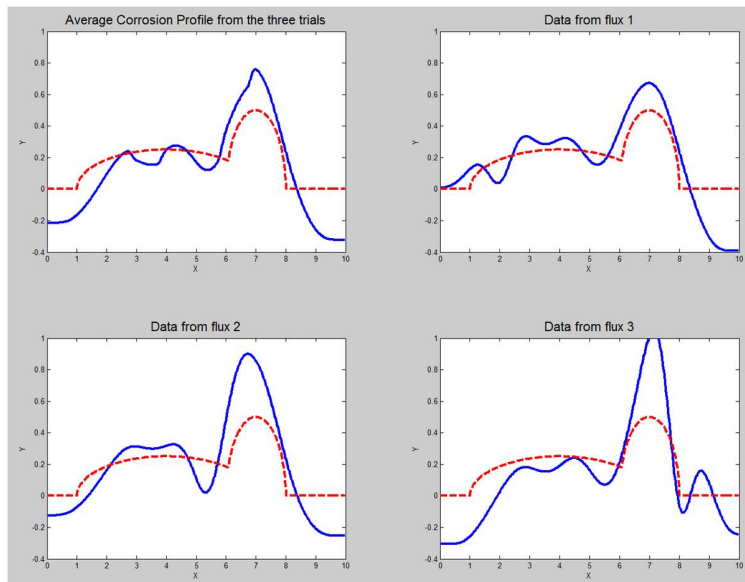
10   Three-Dimensional Calculations

After the computational successes of our two-dimensional model, we decided to look at the full three-dimensional problem and see if a similar model could be developed there. Assume that the region Ω is now a rectangular prism, with length L_x in the x-direction, L_y in the y direction, and height 1 in the z-direction. We will assume we have access to the face at $z = 1$ but nothing else. In correlation to the two-dimensional problem, it is also assumed that all of the other faces are insulated to heat transfer and the corrosion profile is some function $C(x, y)$ defined on $[0, L_x] \times [0, L_y]$. Figure 10.1 below shows a sketch of this setup.
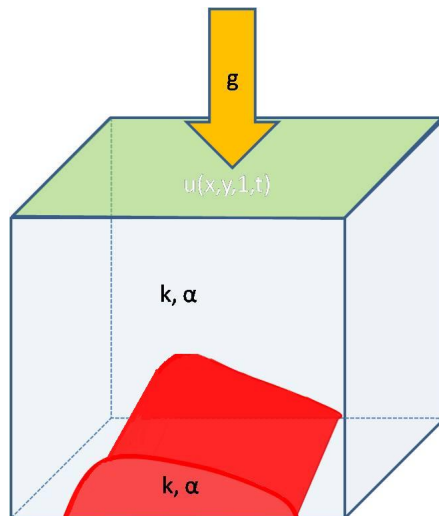
Figure 10.1: Sketch of the Full Three-Dimensional Problem

In this system, we know that the temperature profiles u_1 and u_2 satisfy

$$
\begin{aligned}
\frac{\partial u_1}{\partial t} - \alpha_1 \nabla^2 u_1 &= 0 \text{ on } \Omega_1 \\
\frac{\partial u_1}{\partial \vec{n}} &= 0 \text{ on } y = 0, \ y = L_y, \ x = 0, \ x = L_x \\
\frac{\partial u_1}{\partial \vec{n}} &= g(x, y) \text{ on } z = 1 \\
u_1(x, y, 0) &= 0 \text{ on } \Omega_1
\end{aligned}
$$

and

$$
\begin{aligned}
\frac{\partial u_2}{\partial t} - \alpha_2 \nabla^2 u_2 &= 0 \text{ on } \Omega_2 \\
\frac{\partial u_2}{\partial \vec{n}} &= 0 \text{ on } y = 0, \ y = L_y, \ x = 0, \ x = L_x, \ z = 0 \\
u_2(x, y, 0) &= 0 \text{ on } \Omega_2
\end{aligned}
$$

as well as continuity conditions

$$
\begin{aligned}
u_1 &= u_2 \text{ on } C(x, y) \\
k_1 \frac{\partial u_1}{\partial \vec{n}} &= k_2 \frac{\partial u_2}{\partial \vec{n}} \text{ on } C(x, y)
\end{aligned}
$$

where, in this case, the Laplacian is the full three-dimensional Laplacian

$$
\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}.
$$

In general, the analysis works out in exactly the same way as the two-dimensional case with a few modifications. Equations (8.1)-(8.4) are exactly the same as in the two dimensional case, with the appropriate line and area integrals changed to surface and volume integrals respectively. For this case, we now have a unit normal vector defined by

$$\vec{n} = \frac{< -\frac{\partial C}{\partial x}, -\frac{\partial C}{\partial y}, 1 >}{\sqrt{\frac{\partial C}{\partial x}^2 + \frac{\partial C}{\partial y}^2 + 1}}$$

which modifies equation (8.5) to be

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^{L_y} \int_0^{L_x} \frac{\partial C}{\partial x}\phi\frac{\partial u_2}{\partial x}\bigg|_{C(x,y)} + \frac{\partial C}{\partial y}\phi\frac{\partial u_2}{\partial y}\bigg|_{C(x,y)} - \phi\frac{\partial u_2}{\partial z}\bigg|_{C(x,y)} dx\ dy\ dt$$

$$+ \frac{\alpha_2 - \alpha_1}{\alpha_1\alpha_2} \int_0^T \int_{\Omega_2} \phi\frac{\partial u_2}{\partial t}\ dV\ dt$$

The linearization is also approached in the same way, which results in

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^{L_y} \int_0^{L_x} \frac{\partial C}{\partial x}\phi\frac{\partial u_2}{\partial x}\bigg|_{z=0} + \frac{\partial C}{\partial y}\phi\frac{\partial u_2}{\partial y}\bigg|_{z=0} - \phi\frac{\partial^2 u_2}{\partial z^2}\bigg|_{z=0} dx\ dy\ dt$$

$$+ \frac{\alpha_2 - \alpha_1}{\alpha_1\alpha_2} \int_0^T \int_0^{L_x} \int_0^{L_y} C(x,y)\phi\frac{\partial u_2}{\partial t}\ dy\ dx\ dt$$

Integrating the first term by parts in x, and the second by parts in y and simplifying gives

$$RG(\phi) = \left(\frac{k_2}{k_1} - 1\right) \int_0^T \int_0^{L_y} \int_0^{L_x} \left\{ -C\phi\frac{\partial^2 u_2}{\partial x^2}\bigg|_{z=0} - C\frac{\partial\phi}{\partial x}\frac{\partial u_2}{\partial x}\bigg|_{z=0} - C\phi\frac{\partial^2 u_2}{\partial y^2}\bigg|_{z=0} - C\frac{\partial\phi}{\partial y}\frac{\partial u_2}{\partial y}\bigg|_{z=0} \right.$$

$$\left. - \phi\frac{\partial^2 u_2}{\partial z^2}\bigg|_{z=0} \right\} dx\ dy\ dt + \frac{\alpha_2 - \alpha_1}{\alpha_1\alpha_2} \int_0^T \int_0^{L_x} \int_0^{L_y} C(x,y)\phi\frac{\partial u_2}{\partial t}\ dy\ dx\ dt$$

Combining the second derivative terms into a Laplacian, converting to a time derivative, and combining the coefficients together just as done in the two dimensional case gives an expression very similar to equation (8.7),

$$RG(\phi) = \int_0^{L_x} \int_0^{L_y} C(x,y) \left[\int_0^T \left(1 - \frac{k_2}{k_1}\right) \left(\frac{\partial\phi}{\partial x}\frac{\partial u_0}{\partial x}\bigg|_{z=0} + \frac{\partial\phi}{\partial y}\frac{\partial u_0}{\partial y}\bigg|_{z=0}\right) + \left(\frac{k_2}{k_1} - \frac{\alpha_2}{\alpha_1}\right)\frac{u_0}{\alpha_2}\frac{\partial\phi}{\partial t}\bigg|_{z=0} dt \right] dy\ dx$$

This equation gives the $w_k(x,y)$ functions that would be used to reconstruct the corrosion profile $C(x,y)$.

Generating test functions ϕ_k satisfying

$$\frac{\partial\phi_k}{\partial t} + \alpha_1\nabla^2\phi_k = 0 \text{ on } \Omega$$

$$\frac{\partial\phi_k}{\partial\vec{n}} = 0 \text{ on } y = 0,\ x = 0,\text{ and } x = L_x,\ y = L_y,\ z = 0$$

$$\phi_k(x,y,T) = 0 \text{ on } \Omega$$

is done by the same method as in Section 5.2.1 with an extra set of reflections in the y-direction gives functions of the form

$$\phi = \int_0^T \sum_{i=-N}^N \sum_{j=-N}^N K3_{(2iL_x+a,2jL_y+b,h)}(x,y,z,\tau) + K3_{(2iL_x+a,2jL_y+b,-h)}(x,y,z,\tau)$$

$$+ K3_{(2iL_x+a,2jL_y-b,h)}(x,y,z,\tau) + K3_{(2iL_x+a,2jL_y-b,-h)}(x,y,z,\tau) + K3_{(2iL_x-a,2jL_y+b,h)}(x,y,z,\tau)$$

$$+ K3_{(2iL_x-a,2jL_y+b,-h)}(x,y,z,\tau) + K3_{(2iL_x-a,2jL_y-b,h)}(x,y,z,\tau) + K3_{(2iL_x-a,2jL_y-b,-h)}(x,y,z,\tau)$$

where the function $K3$ is the three dimensional heat kernel:

$$K3_{(x_0,y_0,z_0)}(x,y,z,\tau) = \frac{1}{(4\pi\alpha\tau)^{3/2}} e^{-\frac{(x-a)^2+(y-b)^2+(z-h)^2}{4\alpha\tau}}$$

Derivatives are also computed analytically, however, the integrals can not be eliminated because of the different exponent on τ. Thus the value of the derivative at some point must be calculated via an integral in time, which greatly increases the computation time of the simulation. The derivative formulas were computed for each instance of the heat kernel, and these were added up to give the function that was integrated. So, for instance,

$$\phi_y = \int_0^{T-t} \sum_{i=-N}^{N} \sum_{j=-N}^{N} \sum_{points} K3_y(x,y,z,\tau) \, d\tau$$

where the third sum refers to the eight terms in the expression for ϕ above.

$$\phi_t = -\sum_{i=-N}^{N} \sum_{j=-N}^{N} \sum_{points} K3(x,y,z,T-t)$$

$$K3_y = \frac{b-y}{32\pi^{3/2}(\alpha\tau)^{5/2}} e^{-\frac{(x-a)^2+(y-b)^2+(z-h)^2}{4\alpha\tau}}$$

$$K3_x = \frac{a-x}{32\pi^{3/2}(\alpha\tau)^{5/2}} e^{-\frac{(x-a)^2+(y-b)^2+(z-h)^2}{4\alpha\tau}}$$

$$K3_z = \frac{h-z}{32\pi^{3/2}(\alpha\tau)^{5/2}} e^{-\frac{(x-a)^2+(y-b)^2+(z-h)^2}{4\alpha\tau}}$$

Choosing a set of points (a,b,h) specifies our set ϕ_k of test functions.

Having computed the test functions, the code for reconstructing the corrosion profile in the two dimensional case should, with a few modifications, run for the three dimensional case. However, the increase in complexity has made it impossible for any numerical trials to be run. The COMSOL file takes about a half hour to generate, as opposed to a minute or two for the two-dimensional problem, and the code to generate all of the necessary test functions will currently take around a month to run, even though only 36 test functions are being generated on a smaller range of points. In order for this system to be executed, a more powerful computer will be necessary. The extra dimension more or less squares the amount of calculations the computer has to do, which greatly increases the run time for all of the programs.

11   Conclusions/Discussion

As the results in Sections 7 and 9 have shown, this reconstruction method allows for detection and fairly accurate imaging of plate corrosion of varying sizes with varying material parameters. If the corrosion is very small (maximum height between 1 and 10% of the plate thickness), the corrosion profile can be seen very accurately. As the corrosion size increases, the linearization assumptions are not as valid, but the reconstruction still finds the corrosion in the plate. While the height of the reconstructed profile is slightly higher than that of the actual corrosion, the location is very accurate (see Figure 7.4). Even though the overshoot is around 25% of the actual corrosion profile height, this is not necessarily a bad thing for engineering applications; seeing more corrosion than is actually there would generally cause engineers to err on the side of safety as opposed to the alternative, where seeing less corrosion results in a structural failure before it can be detected.

It has also been shown that the reconstruction process is robust under small amounts of added noise. Some noise is expected in the applications of this procedure, as thermal measuring equipment is only accurate to within a known error and the environment will cause fluctuations in the data. Gaussian noise with a mean of zero and standard deviation of 0.5% of the maximum measured temperature was applied to the temperature data recorded for the corrosion case in Figure 7.3. The corrosion profile constructed from the data with this noise was not quite as good as the reconstruction from the data without noise, but the actual corrosion profile would still be able to be recovered in an engineering situation.

The results also show, in a physical sense, the best way to image corrosion. By looking at the three individual profiles that were averaged to create the final reconstruction, it can be seen that the individual profiles that do the best at modeling the corrosion profile are the ones where the heat source is off to the side of the corrosion, and the heat is allowed to diffuse across the corrosion profile. This can be explained in several ways; the first of which can be seen in all of the results of Section 7. In that case, the w_k functions only consist of one term, which is multiplied by $\frac{\partial u_0}{\partial x}$. The issue with this term is that it is always zero at the symmetry point of the heat source, independent of the choice of ϕ. Since the w_k functions are then used as a basis to reconstruct the corrosion profile, if all of them are zero at a point, it becomes impossible for this method to 'see' any corrosion directly under the heat source. This problem was addressed using the averaging method discussed in section 6.2.

Another way this can be explained is the way in which the heat diffuses. If the heat source is on the side of the corrosion, the heat is going to diffuse over the whole plate, and thus must flow towards the corrosion profile. When it hits the corrosion profile, it can not continue forward in the same way, since there is a change in thermal conductivity at the boundary. Heat flux is preserved at the boundary, but the change in thermal conductivity will induce a change in the surrounding temperature profile. Since we have assumed the corrosion profile is a function of x, the excess heat that needs to diffuse can only move one way: up the plate towards the top edge. As the heat continues to diffuse across the plate, the corrosion profile forces more of it up towards the top of the plate, which is where data is being gathered. Therefore, this type of trial will give a lot of information about the corrosion profile, and give the best overall reconstruction.

Another error that shows up in the reconstruction occurs when the corroded region is near the side of the plate. When this happens, the reconstruction often detects a small area of corrosion between the actual profile and the edge of the plate. This is most prevalent when the heat source is far away from the corrosion profile, and can also be explained in the same vein as the last paragraph. In this case, the heat is diffusing laterally across the corrosion profile, and is getting pushed up towards the top of the plate. When the corrosion profile becomes smaller near the edge of the plate, the heat must diffuse down to the corrosion profile in order to properly image it. However, if this is very close to the edge, the heat is unable to diffuse to the bottom of the plate at the same rate that it would normally. Therefore, the reconstruction mechanism assumes there is something there stopping it from diffusing downward, and adds corrosion to the section. This error may be systematic and could be corrected for, but that has not been completely determined as of yet.

12   Future Work

There are many possibilities for future work on this problem. One of the most applicable ones, as discussed in Section 10 is the process of converting these calculations to the three dimensional problem. As discussed previously, the mathematics is almost identical, but the computation time greatly increases for this problem. A more efficiently implementation may help with this issue, but more computing power would definitely be necessary for this to be done in a reasonable amount of time. An easier way to approach this problem might be to consider a cylindrical pipe or circular plate, as using the axisymmetric nature of the region may help to reduce calculations. The completion of the full three dimensional problem would allow for this method to be used in engineering applications to visualize corrosion.

The two dimensional problem could also be explored in a variety of different ways from here. One way is with the material and experimental parameters. In the applied case, the metals will not have thermal properties of 1 and 0.1, and the heat source will not have an arbitrary flux of 100. An extension of this current work would be into the domain of real world parameters. Consider an actual plate of steel 10 meters by 1 meter, and put the parameters into COMSOL. Add in a heat flux that has been calculated from devices that would be used to apply heat in the field, and see how the plate reacts. Investigations would lead to discovering what kinds of heat fluxes are actually needed to test this procedure on physical systems and figuring out if this reconstruction can be done in an engineering sense.

Some other analysis that can be done centers around the regularization process and reconstruction of the corrosion profile. There are many other regularization methods that could be implemented, as well as other ways to pick a unique corrosion profile as a solution to the system of integral equations. In the current regularization method, the parameters ϵ_∞ and ϵ_{thresh} can be varied. Analysis could be done to see what set of these parameters work best to perform the reconstruction procedure for different sets of material parameters and corrosion profiles. All of these ideas would give a more reliable reconstruction of the corrosion profile, which could then be extended to the three dimensional case for applications.

A   Matlab Scripts

A.1   Generating the ϕ Functions

```matlab
1  function M = testphi_pointsources(x,t,b,h, alpha_1, T)
2  % This function will return matrices for the values of:
3  % dphidy at 1, dphidx at 0, dphidt at 0, and phi at 0
4  % for all of the values (x,t) fed in as vectors. T is assumed to be the
5  % final time where all of the phi function vanish. b and h are parameters for
6  % the forcing function of phi, and alpha_1 is the thermal
7  % diffusivity of region 1.
8
9  % The functions phi are computed via solutions to the heat equation
10 % using a point source at point (b, 1+h) and the method of images to
11 % ensure proper boundary conditions. N determins how many reflections in
12 % the x direction are used in this computation.
13
14 N = 10;
15 l = 10;
16 l1 = length(x);
17 l2 = length(t);
18
19 %% Modifying Variables
20 % Removes the possibility that t=T, which causes the functions to be
21 % undefined, and converts the x and t vectors into matrices so that they
22 % can be added and multiplied without MATLAB getting angry.
23
24 for j=1:l2
25     if t(j) == T
26         t(j) = T-.001;
27     end
28 end
29
30 x = (ones(l2, 1) * x)';
31 t = (ones(l1, 1) * t);
32
33 %% Compute $\phi$ functions
34 % Calculates the term in the summand of each phi function. See the report
35 % for what the functions actually look like.
36
37 dphidt0_part = @(k,x,t) exp(-((x-(2.*k.*l + b)).^2)./(4.*alpha_1.*(T-t))) ...
38     + exp(-((x-(2.*k.*l - b)).^2)./(4.*alpha_1.*(T-t)));
39 dphidx0_part = @(k,x,t) ((x-(2.*k.*l + b))./(((x-(2.*k.*l + b)).^2) ...
40     +(1+h).^2)).*exp(-((x-(2.*k.*l + b)).^2)./(4.*alpha_1.*(T-t))) + ...
41     ((x-(2.*k.*l - b))./(((x-(2.*k.*l - b)).^2) +(1+h).^2))...
42     .*exp(-((x-(2.*k.*l - b)).^2)./(4.*alpha_1.*(T-t)));
43 dphidy1_part = @(k,x,t) (-h.*exp(-h.^2./(4.*alpha_1.*(T-t)))./...
44     (((x-(2.*k.*l + b)).^2) +(h).^2)).*exp(-((x-(2.*k.*l + b)).^2)...
45     ./(4.*alpha_1.*(T-t))) +(-h.*exp(-h.^2./(4.*alpha_1.*(T-t)))./...
46     (((x-(2.*k.*l - b)).^2) +(h).^2)).*exp(-((x-(2.*k.*l - b)).^2)./...
47     (4.*alpha_1.*(T-t))) +((2+h).*exp(-(2+h).^2./(4.*alpha_1.*(T-t)))./...
```

37

```matlab
48          (((x-(2.*k.*l + b)).^2) +(2+h).^2)).*exp(-((x-(2.*k.*l + b)).^2)./...
49          (4.*alpha_1.*(T-t))) +((2+h).*exp(-(2+h).^2./(4.*alpha_1.*(T-t)))./...
50          (((x-(2.*k.*l - b)).^2) +(2+h).^2)).*exp(-((x-(2.*k.*l - b)).^2)./(4.*alpha_1.*(T-t))));
51  phipart = @(k,x,t) exp(-((x-(2.*k.*l + b)).^2)./(4.*alpha_1.*(t))) ...
52          + exp(-((x-(2.*k.*l - b)).^2)./(4.*alpha_1.*(t))));
53
54  dphidt0_sum = zeros(l1, l2);
55  dphidx0_sum = zeros(l1, l2);
56  dphidy1_sum = zeros(l1, l2);
57  phi_0_int = @(x,t) 0;
58
59  %%
60  % Adds up the sums to get a computation of the actual phi functions
61  for i = 1:(2*N+1)
62      dphidt0_sum = dphidt0_sum + dphidt0_part(i-(N+1),x,t);
63      dphidx0_sum = dphidx0_sum + dphidx0_part(i-(N+1),x,t);
64      dphidy1_sum = dphidy1_sum + dphidy1_part(i-(N+1),x,t);
65      phi_0_int = @(x,t) phi_0_int(x,t) + phipart(i-(N+1),x,t);
66  end
67
68  dphidy1 = -1/(2*pi*alpha_1).*dphidy1_sum;
69  dphidx0 = -exp(-(1+h).^2./(4.*alpha_1.*(T-t)))./(pi*alpha_1) .* dphidx0_sum;
70  dphidt0 = (-exp(-((1+h).^2)./(4.*alpha_1.*(T-t))))./(2.*pi*alpha_1.*(T-t))...
71      .* dphidt0_sum;
72
73  phi_0 = zeros(l1, l2);
74
75  %%
76  % Performs an integral to compute the value of phi at the back surface,
77  % because there is no way to get rid of the integral in the calculations.
78  for i = 1:l2
79      for j = 1:l1
80          phi_0(j,i) = 1/(2*pi*alpha_1)*quadgk(@(t)((exp(-((1+h).^2)...
81              ./(4.*alpha_1.*(t))))./(t)).*phi_0_int(x(j,j),t), 0, T-t(i,i));
82      end
83  end
84
85  %% Compile results
86  % Compiles the four previously generated arrays into one matrix which is
87  % returned by the function.
88
89  M(:,:,1) = dphidy1;
90  M(:,:,2) = dphidx0;
91  M(:,:,3) = dphidt0;
92  M(:,:,4) = phi_0;
93
```

```matlab
1  function phibuild_pointSource()
2  % This function generates a set of the phi functions using the
3  % testphi_pointsources.m file. This will create a set of points sources
4  % that are evenly spaced across the top of the plate, and 5 different
5  % distances away. Using the testphi function, phiTable will be built at
6  % all of these different conditions, and then saved to a dump file, which
7  % can be loaded by RunFile.
8
9  phiTable=zeros(401,length(0:0.05:20),4,45);
10
11 for i=0:8
12     for j = 1:5
13     phiTable(:,:,:,5*i+j) = testphi_pointsources(linspace(0,10,401),...
14         0:0.05:20,i+1,j/10,1,20);
15     end
16     i
17 end
18
19 save('phidump_PS2', 'phiTable');
```

A.2   Importing the COMSOL Files

```matlab
1   function [x,f,dfdx,dfdy,dfdt]=comsol_import_1d(filename, xdiv, timestep)
2   % Matt Charnley and Andrew Rzeznik
3   % June 18th, 2012
4   % Rose-Hulman REU 2012 - Inverse Problems Group
5
6   %%
7   % This function will import data from a COMSOL files taken at a cut-line in
8   % a 2 dimensional problem. It is assumed that the file, 'filename' is a
9   % text file exported from COMSOL with the columns: x, y, and then T, Tx, Ty
10  % at each time step. It also assumes that the cutline is horizontal, so the
11  % only data that needs to be taken for position is the x coordinate. It
12  % interpolates the data to all midpoints, which simplifies the integration
13  % process. xdiv is the number of desired x divisions, all interpolated to
14  % midpoints. timestep gives the difference in time between each successive
15  % measurement, which is used in computing the time-derivative of the data.
16
17  %% Import the data
18  % Imports the data, grabs the x-coodrinate data, and then excludes the x
19  % and y coordinate columns from the matrix.
20
21  A=load(filename);
22  [~, c]=size(A);
23  xraw=A(:,1);
24  xmin=min(xraw);
25  xmax=max(xraw);
26
27  A=A(:,3:end);
28
29
30  %% Create Variables
31  % Sets up all of the arrays of zeros needed to interpolate the data
32
33  f2=zeros(xdiv,(c-2)/3);
34  dfdx2=zeros(xdiv,(c-2)/3);
35  dfdy2=zeros(xdiv,(c-2)/3);
36
37  f=zeros(xdiv,(c-2)/3-1);
38  dfdx=zeros(xdiv,(c-2)/3-1);
39  dfdy=zeros(xdiv,(c-2)/3-1);
40
41  fraw=zeros(length(xraw),(c-2)/3);
42  dfdxraw=zeros(length(xraw),(c-2)/3);
43  dfdyraw=zeros(length(xraw),(c-2)/3);
44
45  %% Get the Raw Data
46  % Pulls the raw data off of the table and finds the x steps needed to get
47  % xdiv regions, all evaluated at at the midpoints.
48
49  for i=1:(c-2)/3
```

```
50      fraw(:,i)=A(:,3*i-2);
51      dfdxraw(:,i)=A(:,3*i-1);
52      dfdyraw(:,i)=A(:,3*i);
53  end
54
55  h=(xmax-xmin)/xdiv;
56  x=(xmin+h/2):h:(xmax-h/2);
57
58  %% Interpolation
59  % Interpoloates first to the midpoints in the x-direction at each time
60  % step, and then interpolates to the midpoint in the time direction at each
61  % x points.
62
63  % X-Direction
64
65  for i=1:((c-2)/3)
66      f2(:,i)=interp1(xraw,fraw(:,i),x, 'spline');
67      dfdx2(:,i)=interp1(xraw,dfdxraw(:,i),x, 'spline');
68      dfdy2(:,i)=interp1(xraw,dfdyraw(:,i),x, 'spline');
69  end
70
71  % Time direction
72
73  for i=1:length(x)
74      f(i,:)=interp1(0:((c-2)/3-1),f2(i,:),0.5:((c-2)/3-1.5), 'spline');
75      dfdx(i,:)=interp1(0:((c-2)/3-1),dfdx2(i,:),0.5:((c-2)/3-1.5), 'spline');
76      dfdy(i,:)=interp1(0:((c-2)/3-1),dfdy2(i,:),0.5:((c-2)/3-1.5), 'spline');
77  end
78
79  %% Time Derivative
80  % Takes the time-derivative of the data using the timestep given in the
81  % function definition.
82
83  dfdt = diff(f2, 1, 2)./timestep;
```

A.3 Computing the Corrosion Profile

```matlab
%% RunFile2_bool.m
% Matt Charnley and Andrew Rzeznik
% Rose-Hulman Math REU 2012 - Inverse Problems Group

%%
% This is the main run file when there are three separate COMSOL Data files
% to be imported and used for calculation of the corrosion profile. For all
% of the data we ran, the three trials, Left, Center, and Right, refered to
% a rectangular flux source centered at 2, 5, and 8 respectively, out of a
% bar of length 10. The corrosion profile must be the same in all cases.
% The location of the heat sources is not important to the calculations,
% but we found that these three locations did a good job of showing the
% corrosion profile.

file_Corroded_Top_Left = 'Shifted_heat_sources\uleft_05.txt';
file_UC_Bottom_Left = 'Shifted_heat_sources\u0leftbottom.txt';
file_UC_Top_Left = 'Shifted_heat_sources\u0lefttop.txt';

file_Corroded_Top_Center = 'Shifted_heat_sources\umiddle_05.txt';
file_UC_Bottom_Center = 'Shifted_heat_sources\u0midbottom.txt';
file_UC_Top_Center = 'Shifted_heat_sources\u0midtop.txt';

file_Corroded_Top_Right = 'Shifted_heat_sources\uright_05.txt';
file_UC_Bottom_Right = 'Shifted_heat_sources\u0rightbottom.txt';
file_UC_Top_Right = 'Shifted_heat_sources\u0righttop.txt';

%%
% This is a boolean value to determine whether or not to vary T in the
% calculation of the w functions below. We found no improvement in the
% results by doing this, so it was left at 0. It can be switched to 1 and
% the code should still run as desired.

time_split = 0;

%% Import the data from COMSOL txt files
% See comsol_import_1d for more description of this process. The data is
% imported into matrices, which are then combined into a single three
% dimenaional array for use in the other calculations.

[~,u5,~,u5y] = comsol_import_1d(file_Corroded_Top_Center, 400, .05);
[~,u5_un,~,u5uny] = comsol_import_1d(file_UC_Top_Center, 400, .05);
[~,u0_5,du0_5dx,~,du0_5dt] = comsol_import_1d(file_UC_Bottom_Center,...
    400, .05);

[~,u2,~,u2y] = comsol_import_1d(file_Corroded_Top_Left, 400, .05);
[~,u2_un,~,u2uny] = comsol_import_1d(file_UC_Top_Left, 400, .05);
[~,u0_2,du0_2dx,~,du0_2dt] = comsol_import_1d(file_UC_Bottom_Left,...
    400, .05);

```

```
50    [~,u8,~,u8y] = comsol_import_1d(file_Corroded_Top_Right, 400, .05);
51    [~,u8_un,~,u8uny] = comsol_import_1d(file_UC_Top_Right, 400, .05);
52    [x_raw,u0_8,du0_8dx,~,du0_8dt] = comsol_import_1d(file_UC_Bottom_Right,...
53        400, .05);
54
55    [r1,c1] = size(u5);
56    u = zeros(r1, c1, 3);
57    u_un = zeros(r1, c1, 3);
58    du0dx = zeros(r1, c1, 3);
59    u0 = zeros(r1, c1, 3);
60    du0dt = zeros(r1, c1, 3);
61    du0dy1 = zeros(r1, c1, 3);
62    du0dx2 = zeros(r1, c1, 3);
63
64    u(:,:,1) = u2;
65    u_un(:,:,1) = u2_un;
66    du0dx(:,:,1) = du0_2dx;
67    u0(:,:,1) = u0_2;
68    du0dt(:,:,1) = du0_2dt;
69    du0dy1(:,:,1) = u2uny;
70
71    u(:,:,2) = u5;
72    u_un(:,:,2) = u5_un;
73    du0dx(:,:,2) = du0_5dx;
74    u0(:,:,2) = u0_5;
75    du0dt(:,:,2) = du0_5dt;
76    du0dy1(:,:,1) = u5uny;
77
78    u(:,:,3) = u8;
79    u_un(:,:,3) = u8_un;
80    du0dx(:,:,3) = du0_8dx;
81    u0(:,:,3) = u0_8;
82    du0dt(:,:,3) = du0_8dt;
83    du0dy1(:,:,1) = u8uny;
84
85    u_dist = u - u_un;
86
87    du0dx2 = (du0dx(2:end,:,:) - du0dx(1:end-1,:,:))./.025;
88    du0dx2(400,:,:) = zeros(c1,3);
89
90    %% Remainder of load variable preps
91    % See phiBuild_pointSource.m and testphi_pointsources.m for the code that
92    % goes into creating the phidump file. This file contains the desired test
93    % functions and their derivatives mentioned in section   of the report. The
94    % interpolation type is set to 'spline' which gave us some improvement in
95    % the results. This section initializes all of the vectors that will be
96    % filled later in the program, from the phi function vectors to the w
97    % functions and c = lambda values for the weights.
98
99    load('phidump_PS2.mat')
100   [~,~,~,testnum]=size(phiTable); %The number of test functions
```

```matlab
101  interpoltype = 'spline';
102
103  %%
104  % This sets the final end time of the integration. It should be the same
105  % stopping time as the COMSOL file.
106
107  T = 20;
108
109  xphi=linspace(0,10,401);
110  tphi=0:0.05:T;
111  x = (xphi(2:end) + xphi(1:end-1))/2;
112  t = (tphi(2:end) + tphi(1:end-1))/2;
113
114  %%
115  % Conductivities and thermal diffusivities of the uncorroded (1) and
116  % corroded (2) regions respectively, as well as the length of the slab.
117
118  k1=1;
119  k2=0.1;
120  alpha1=1;
121  alpha2=0.1;
122  l=10;
123
124  %%
125  % Arrays for the needed pieces of the phi functions, the w functions which
126  % will be used as a basis for the corrosion profile, and the c vectors,
127  % which weight the w functions to give the appropriate profile. RG is the
128  % reciprocity gap integral values. See section ___ of the report for
129  % calculations.
130
131  dphidy1 = zeros(length(x), length(t), testnum);
132  dphidt0 = zeros(length(x), length(t), testnum);
133  dphidx0 = zeros(length(x), length(t), testnum);
134  phi0 = zeros(length(x), length(t), testnum);
135
136  w1=zeros(length(x),testnum,3);
137  w2=zeros(length(x),testnum,3);
138  w3 = zeros(length(x), testnum, 3);
139
140  c=zeros(testnum,3);
141  Cerror=zeros(testnum,1);
142  RG=zeros(testnum,3);
143  c_new = zeros(testnum, 3);
144
145  if time_split
146      w1 = zeros(length(x),T*testnum,3);
147      w2 = zeros(length(x),T*testnum,3);
148      c=zeros(T*testnum,3);
149      c_new = zeros(T*testnum, 3);
150      Cerror=zeros(T*testnum,1);
151      RG=zeros(T*testnum,3);
```

```matlab
152  end
153
154  %% Load the test function arrays
155  % Uses the previosuly loaded in phiTable array to grab the values of the
156  % phi function and load them into the arrays defined above. It makes use of
157  % the interp2 function so that the values of the phi function are found at
158  % the same points that comsolimport1d uses. It also finds the dx and dt
159  % step sizes needed for integration.
160
161
162  for i = 1:testnum
163      dphidy1(:,:,i) = interp2(xphi, tphi', phiTable(:,:,1,i), x_raw, t',...
164          interpoltype);
165      dphidx0(:,:,i) = interp2(xphi, tphi', phiTable(:,:,2,i), x_raw, t',...
166          interpoltype);
167      dphidt0(:,:,i) = interp2(xphi, tphi', phiTable(:,:,3,i), x_raw, t',...
168          interpoltype);
169      phi0(:,:,i) = interp2(xphi, tphi', phiTable(:,:,4,i), x_raw, t',...
170          interpoltype);
171  end
172  dx = x(2) - x(1);
173  dt = t(2) - t(1);
174
175  %% Build Necessary Functions
176  % This section builds the w and RG functions for approximating the
177  % corrosion profile. Multiple attempts were made to get a good
178  % approximation, see the report for more details. The w1 function ended up
179  % working the best.
180
181  for k = 1:3
182      for i=1:testnum
183          if ~time_split
184              RG(i,k) = dt.*dx.*sum(sum(u_dist(:,:,k).*dphidy1(:,:,i) - ...
185                  0*du0dy1(:,:,k).*phi0(:,:,i)));
186              w1(:,i,k) = dt.*sum(((k1-k2)/k1).*(du0dx(:,:,k).*dphidx0(:,:,i)),2)';
187              w2(:,i,k) = dt.*sum(((k1-k2)/k1).*(u0(:,:,k).*dphidt0(:,:,i)./alpha1),2)';
188              w3(:,i,k) = dt.*sum(((k1-k2)/k1).*(phi0(:,:,i).*du0dt(:,:,k)./alpha1),2)';
189              wnew2(:,i,k) = dt.*sum((phi0(:,:,i).*du0dt(:,:,k)./alpha1),2);
190              wnew3(:,i,k) = dt.*sum((k2/k1).*phi0(:,:,i).*du0dx2(:,:,k),2);
191          else
192              for j = 1:T
193                  RG(T*(i-1)+j,k) = dt.*dx.*sum(sum(u_dist(:,1:round(j/dt),k).*...
194                      dphidy1(:,round((T-j)/dt+1):end,i)));
195                  w2(:,T*(i-1)+j,k) = dt.*sum(((k1-k2)/k1).*u0(:,1:round(j/dt),k)'...
196                      .*dphidt0(:,round((T-j)/dt+1):end,i)'./alpha1);
197                  w1(:,T*(i-1)+j,k) = dt.*sum(((k1-k2)/k1).*...
198                      (du0dx(:,1:round(j/dt),k)'.*dphidx0(:,round((T-j)/dt+1):end,i)'));
199                  w3(:,T*(i-1)+j,k) = dt.*sum(((k1-k2)/k1).*...
200                      du0dt(:,1:round(j/dt),k)'.*phi0(:,round((T-j)/dt+1):end,i)'./alpha1);
201              end
202          end
```

```matlab
203        end
204    end
205
206    %% Compute w
207    % This section computes the w functions. It adds together pieces from the
208    % last step to get the total function. This section made it easuer to
209    % change the code and recompute the w's without running all of the
210    % calculations over again.
211
212    wnew = w1-wnew2-wnew3;
213    w = w1;
214
215    %% Define B matrix and solve for the Unregularized Corrosion Profile
216    % Sets up the B matrix as defined in section 5 of the report, as the
217    % integrals of the w functions. Then, invert the B matrix, multiply by RG,
218    % and solve for the c (lambda) coefficients of the unregularized corrosion
219    % profile
220
221    if time_split
222        B = zeros(T*testnum,T*testnum);
223        for i=1:T*testnum
224            for j=1:T*testnum
225                for k = 1:3
226                    B(i,j,k) = w(:,i,k)'*w(:,j,k)*dx; % Integral w's
227                end
228            end
229        end
230    else
231        B=zeros(testnum,testnum,3);
232        for k = 1:3
233            for i=1:testnum
234                for j=1:testnum
235                    B(i,j,k) = w(:,i,k)'*w(:,j,k)*dx; % Integral w's
236                end
237            end
238        end
239    end
240
241    for k = 1:3
242        c(:,k)=B(:,:,k)^-1*RG(:,k);
243    end
244    %% Regularization
245    % Implements a regularization method based on the Singular Value
246    % Decomposition of the matrix B. If a singular value of B is less than a
247    % certain threshold as set by epsd, then the corresponding value in B
248    % inverse is set to zero. Then, that matrix is multiplied by the RG vector
249    % to generate c_new (lambda) coefficients for the regularized corrosion
250    % profile. See the report for calculation of epsd.
251
252    epsthreshold=0.2; %relative noise on a 0:1 scale
253    epsinf=1; %maximum absolute error
```

```matlab
254
255  for k = 1:3
256      [U,D,V] = svd(B(:,:,k));
257
258      if ~time_split
259          for i=1:testnum
260              Cerror(i)=dt.*dx.*sum(sum(abs(dphidy1(:,:,i))));
261          end
262      else
263          for i=1:testnum
264              for j = 1:T
265                  Cerror(T*(i-1)+j)=dt.*dx.*sum(sum(abs(dphidy1(:,:,i))));
266              end
267          end
268      end
269
270      hw=mean(max(w(:,:,k)));
271      Cmax=max(Cerror);
272      epsd=sqrt(testnum)*Cmax*epsinf*hw/epsthreshold;
273
274      D_inv = D^-1;
275      if ~time_split
276          for i = 1:testnum
277              if D(i,i) < abs(epsd)
278                  D_inv(i,i) = 0;
279              end
280          end
281      else
282          for i = 1:T*testnum
283              if D(i,i) < epsd
284                  D_inv(i,i) = 0;
285              end
286          end
287      end
288
289      c_new(:,k) = V * D_inv * U' * RG(:,k);
290  end
291  %% Compute the corrosion profiles
292  % Multiplies the c vector by the w function to generate the corrosion
293  % profile as a function of x. This section also made it easier to recompute
294  % the profile for repeated iteration without graphing the profile.
295
296  corrosion_old = zeros(length(x), 3);
297  corrosion_new = zeros(length(x), 3);
298
299  for k = 1:3
300      corrosion_old(:,k) = w(:,:,k)*c(:,k);
301      corrosion_new(:,k) = w(:,:,k)*c_new(:,k);
302  end
303
304  %% Create the final profiles
```

```matlab
305  % Uses a weighting function to combine the three generated corrosion
306  % profiles into a single profile. It ignores the area immediately around
307  % each laser source because the data does not represent the corrosion
308  % profile there. The first two commented attempts were straight boolean
309  % values of weighting the functions, where the third implemented a
310  % smoothing region between each transition to make the resulting profile
311  % look better.
312
313  nonReg = zeros(length(x),1);
314  reg = zeros(length(x),1);
315
316  smoothing = 0.3;
317  speed = 30;
318
319  for i = 1:length(x)
320      if x(i) <= (3-smoothing)
321          nonReg(i) = 1/2*(corrosion_old(i,2) + corrosion_old(i,3));
322          reg(i) = 1/2*(corrosion_new(i,2) + corrosion_new(i,3));
323      elseif x(i) <= (3+smoothing)
324          nonReg(i) = 1/2*(exp(-speed*(x(i)-(3-smoothing))^2)*corrosion_old(i,2)...
325              +(2-exp(-speed*(x(i)-(3-smoothing))^2))*corrosion_old(i,3));
326          reg(i) = 1/2*(exp(-speed*(x(i)-(3-smoothing))^2)*corrosion_new(i,2)...
327              +(2-exp(-speed*(x(i)-(3-smoothing))^2))*corrosion_new(i,3));
328      elseif x(i) <= (4-smoothing)
329          nonReg(i) = corrosion_old(i,3);
330          reg(i) = corrosion_new(i,3);
331      elseif x(i) <= (4+smoothing)
332          nonReg(i) = 1/2*((1-exp(-speed*(x(i)-(4-smoothing))^2))*corrosion_old(i,1)...
333              +(1+exp(-speed*(x(i)-(4-smoothing))^2))*corrosion_old(i,3));
334          reg(i) = 1/2*((1-exp(-speed*(x(i)-(4-smoothing))^2))*corrosion_new(i,1)...
335              +(1+exp(-speed*(x(i)-(4-smoothing))^2))*corrosion_new(i,3));
336      elseif x(i) <= (6-smoothing)
337          nonReg(i) = 1/2*(corrosion_old(i,1) + corrosion_old(i,3));
338          reg(i) = 1/2*(corrosion_new(i,1) + corrosion_new(i,3));
339      elseif x(i) <= (6+smoothing)
340          nonReg(i) = 1/2*((2-exp(-speed*(x(i)-(6-smoothing))^2))*corrosion_old(i,1)...
341              +exp(-speed*(x(i)-(6-smoothing))^2)*corrosion_old(i,3));
342          reg(i) = 1/2*((2-exp(-speed*(x(i)-(6-smoothing))^2))*corrosion_new(i,1)...
343              +exp(-speed*(x(i)-(6-smoothing))^2)*corrosion_new(i,3));
344      elseif x(i) <= (7-smoothing)
345          nonReg(i) =corrosion_old(i,1);
346          reg(i) = corrosion_new(i,1);
347      elseif x(i) <= (7+smoothing)
348          nonReg(i) = 1/2*((1-exp(-speed*(x(i)-(7-smoothing))^2))*corrosion_old(i,2)...
349              +(1+exp(-speed*(x(i)-(7-smoothing))^2))*corrosion_old(i,1));
350          reg(i) = 1/2*((1-exp(-speed*(x(i)-(7-smoothing))^2))*corrosion_new(i,2)...
351              +(1+exp(-speed*(x(i)-(7-smoothing))^2))*corrosion_new(i,1));
352      else
353          nonReg(i) = 1/2*(corrosion_old(i,2) + corrosion_old(i,1));
354          reg(i) = 1/2*(corrosion_new(i,2) + corrosion_new(i,1));
355      end
```

```matlab
356   end
357
358   %% Plotting Functions
359   % Plots all of the generated corrosion regions. The first is a single graph
360   % of the unregulated profile, and the second is a 2X2 subplot graph of the
361   % average corrosion profile from the regulated case, as well as the three
362   % corrosion profiles each generated by a single heat flux. The actual
363   % profile is recovered manually from COMSOL.
364   plot(x, nonReg, 'k', 'LineWidth', 2);
365
366   fig1 = figure;
367   subplot(2,2,1)
368   plot(x, reg, 'b', 'LineWidth', 3);
369   hold on;
370   % Get the actual corrosion profile from COMSOL - may have to do it by hand
371
372   %actualProfile = @(x) (1/30).*(sqrt(9 - (x-4).^2)).*(abs(x-4) <= 3);
373   %actualProfile = @(x) (1/2).*(sqrt(1 - (x-7).^2)).*(abs(x-7) <= 1).*...
374   %       ((1/12).*(sqrt(9 - (x-4).^2))<= (1/2).*(sqrt(1 - (x-7).^2))) + ...
375   %       (1/12).*(sqrt(9 - (x-4).^2)).*(abs(x-4) <= 3).*((1/12).*...
376   %       (sqrt(9 - (x-4).^2))> (1/2).*(sqrt(1 - (x-7).^2)));
377   actualProfile = @(x) (1/40).*(sqrt(4 - (x-3).^2)).*(abs(x-3) <= 2);
378
379   %c_star = w\actualProfile(x)';
380   %plot(x, w*c_star, 'r--');
381
382   plot(x, actualProfile(x), 'r--', 'LineWidth', 3);
383   xlabel('X');
384   ylabel('Y');
385   title('Average Corrosion Profile from the three trials', 'FontSize', 16);
386   %legend('Approximated Corrosion', 'Actual Corrosion', 'Location', 'NW');
387   axis([0 10 -.02 .08]);
388
389   vect = get(0, 'ScreenSize');
390   h = vect(4);
391   wid = vect(3);
392   set(fig1, 'Position', [50,50,wid-400,h-100]);
393
394   for k = 1:3
395       subplot(2,2,k+1);
396       plot(x, corrosion_new(:,k), 'LineWidth', 3)
397       hold on
398       plot(x, actualProfile(x), 'r--', 'LineWidth', 3);
399       xlabel('X');
400       ylabel('Y');
401       title(['Data from flux ' num2str(k)], 'FontSize', 16);
402       axis([0,10,-.02,.08]);
403   end
```

A.4   Modified Run File for New Linearization

```
1   %% ----------------------------------------------------------------------------
2   %Builds w and RG functions
3   %Assumes integrals use midpoint data in all variables
4   for k = 1:3
5       for i=1:testnum
6           if ~time_split
7               RG(i,k) = dt.*dx.*sum(sum(u_dist(:,:,k).*dphidy1(:,:,i) - ...
8                   0*du0dy1(:,:,k).*phi0(:,:,i)));
9               w1(:,i,k) = dt.*sum(((k1-k2)/k1).*(du0dx(:,:,k).*dphidx0(:,:,i)),2)';
10              w2(:,i,k) = dt.*sum((u0(:,:,k).*dphidt0(:,:,i)./alpha2),2)';
11          else
12              for j = 1:T
13                  RG(T*(i-1)+j,k) = dt.*dx.*sum(sum(u_dist(:,1:round(j/dt),k)...
14                      .*dphidy1(:,round((T-j)/dt+1):end,i)));
15                  w2(:,T*(i-1)+j,k) = dt.*sum(u0(:,1:round(j/dt),k)'...
16                      .*dphidt0(:,round((T-j)/dt+1):end,i)'./alpha2);
17                  w1(:,T*(i-1)+j,k) = dt.*sum(((k1-k2)/k1).*(du0dx(:,1:round(j/dt),k)'...
18                      .*dphidx0(:,round((T-j)/dt+1):end,i)'));
19              end
20          end
21      end
22  end
23
24  %%
25  w = w1 + ((k2/k1)-(alpha2/alpha1)).*w2;
```

B   COMSOL/MATLAB Operation

This section will outline the basic process for creating a COMSOL Model, exporting the data into a text file, and importing the data for use in MATLAB. There are many more features that can be utilized and adjusted, such as changing the mesh size or creating more complicated heat transfer functions. Look through the menus and right-click on the items in the model viewer window to edit them.

B.1   Creating a COMSOL Model

1. Open COMSOL Multiphysics - Classkit License. There should be an empty GUI interface upon loading. Either open a previously created model, or use the wizard in the bottom right to create a new model.

2. Select the dimensions of the problem, choose the heat transfer module, select Time Dependent Study, and create the model by clicking on the checkered flag.

3. Create the desired geometry.

   (a) To add a geometric figure to the model, right-click on Geometry and pick the desired shape. You can either draw in on the GUI interface, or type its size parameters into the window in the bottom left. To build the shape in the window, click the button in the top right of the bottom left window that looks like a building with a red box on it.

   (b) To create more complicated shapes, there are boolean operations like intersection and difference in the menu after right-clicking on Geometry. For each of these, you need to select two shapes that have already been built in the window to perform the operation on. Note, once you perform this operation, the original shapes will be gone, and you will just be left with the intersection. To create a rectangle with part of an ellipse on it, you will need two rectangles, one to do the intersection with, and the other to be the actual shape that stays in the model.

4. Add materials. To add materials to the model, right-click on Materials, and select '+ Material'. In this window, you can set which regions of the model the material exists in, as well as its thermal properties. Be sure to set all of the properties before trying to run the model, otherwise it will crash.

5. Define Heat Transfer. In order to define the heat transfer, initial conditions and boundary conditions need to be specified.

   (a) Click the arrow to open the Heat Transfer drop-down menu, and click on Initial Values. Here you can set the initial condition for each region in the model.

   (b) As default, all of the external boundaries are set to be insulated. That can be overridden in many ways. Right-click on Heat Transfer and pick Inflow Heat Flux, or whatever other one is what you want. In the window that appears, you can specify a heat flux for any of the external boundaries in the model. Click the boundary, click '+', and then define a function for the heat flux. If you want something more complicated than a constant, you may need to define functions for it. These can be done in the definitions and global definitions menus at the top of the list. In these functions 't[1/s]' gives you a time variable, 'x[1/m]' gives you the x-coordinate, and so on.

6. Finally, to run the model, right-click on Study and click Compute.

B.2   Exporting Data as Text

1. The first step to exporting COMSOL data is to create a Cut-Line (in the 2D case) from which COMSOL will grab the data. Right-Click on Data Sets and select Cut-Line 2D.

2. Specify the line by (x, y) coordinates for the top of bottom surface of the model, whichever side you are trying to get data from.

3. Once the line has been built, right-click on the Cut Line and click Add to Report. You can also rename the Cut Line at this point if you want.

4. Click on the arrow next to Report and click on Data 1. This will represent the data generated by the Cut Line that you just created.

5. In the window that appears, make sure that the Cut Line you just made is selected in the Data Set tab. The expressions box will let you specify what pieces of data get exported into the text file. For example, in our files, we had Temperature, Tx, and Ty as the three expressions.

6. After setting the expressions, create a save file for the text. Click the Browse button and move to a folder that you can identify. Note: It will be very helpful if this is your MATLAB directory, or somewhere where you are going to be saving MATLAB files. It will make it easier to load the file later.

7. To export the file, click the button in the top right of that window with a box and arrow. This will create the text file with all of the data in it.

8. To import into MATLAB, follow the procedure in comsol_import_1d, which can be found in Appendix A. The data will be exported in the format:

    x $\mid$ y $\mid$ E1 at $t = 0$ $\mid$ E2 at $t = 0$ $\mid$ ... $\mid$ E1 at $t = t_1$ $\mid$ E2 at $t = t_1$ $\mid$ ... $\mid$ E1 at $f = t_f$ $\mid$ E2 at $t = t_f$

    where E1, E2, etc. are the expressions that had been selected in COMSOL before the file was exported, in the order they appear in the export window.

    The MATLAB 'load' command will take this text file and turn it into a matrix without the titles. From there, the data can be converted to the desired variables and used to calculate the corrosion profile. The MATLAB files in Appendix A show an example of how to do this.

References

[1] Court Hoang and Katherine Osenbach. *Electrical Impedance Imaging of Corrosion on a Partially Accessible 2-Dimensional Region: An Inverse Problem in Non-destructive Testing.* Rose-Hulman REU 2009

[2] Kurt Bryan and Lester Caudill. *Reconstuction of an unknown boundary portion from Cauchy data in* n *dimensions.* IOP Publishing, Inverse Problems **21** (2005) 239-255

[3] Olivier Poisson. *Uniqueness and Holder stability of discontinuous diffusion coefficients in three related inverse problems for the heat equation.* IOP Publishing, Inverse Problems **24** (2008) 025012