Assignment 2: An inverse problem for heat conduction

Hand in a written report containing a short presentation of the problem, results, discussion, source code, and a print-out of the result from your MATLAB sessions. Answer all questions and comment your results. If the report is incomplete, we will return it for completion before starting to grade it. You may work alone or in groups of two (not three). Both persons in a group should contribute to the solution and the report. Discussions between the groups are encouraged. If you receive substantial help from another group, say so in the report. You are not allowed to copy solutions or computer codes from others. We prefer answering questions in person or by phone (instead of email). We do not wish to debug your code.

The absolute deadline for submitting this year’s assignments is 2008-11-05. You will obtain a 1/2-point bonus point on the final exam if a correct solution for this assignment is handed in at the latest 2008-10-02.

The MATLAB m-files temperatures.m and param.m, available for download on the home page, are necessary for this assignment.

The problem

A long circular cylinder with a narrow hole at the core is composed of two different materials: an outer and an inner material that have different heat conduction properties (Figure 1). The hole at the core is filled with a heating element and the whole cylinder is submerged in ice water. The temperature can be measured at two points inside the cylinder by embedded sensors, one located at the boundary between the core and the inner material, and one at the boundary between the inner and outer material. These temperature measurements should be used to determine the heat conduction parameters in the inner and the outer materials.

We assume a steady situation (thermodynamic equilibrium) and that the cylinder is long. Then the temperature will vary only in the radial direction. Denote by \( u = u(r) \) the temperature as a function of \( r \), the distance from the center axis of the cylinder. Under these assumptions, the heat equation for the present case reduces to the two-point boundary value problem

\[
-(aru')' = 0 \quad \text{in } r_c < r < R, \\
-a(r_c)ru'(r_c) = g, \\
u(R) = 0,
\]

where \( r_c \) and \( R \) are the radii of the core and the cylinder, respectively. The function \( a \) is

\[
a = \begin{cases} 
  a_1 & \text{for } r_c < r < r_{12}, \\
  a_2 & \text{for } r_{12} < r < R,
\end{cases}
\]

where \( r_{12} \) is the radius to the boundary between the two materials, and \( a_1, a_2 \) are the constant heat conduction parameters for the inner and outer materials, respectively. (We have thus assumed that the two materials are homogeneous and isotropic, that is,
that the heat conduction properties for each of the materials are the same at each point and in each direction.)

The temperature at the boundary between the core and the inner material is thus \( u(r_c) \), and the temperature at the interface between the inner and outer material is \( u(r_{12}) \). The measurements of these quantities are denoted \( u_c \) and \( u_{12} \). Hence, to determine the unknown heat conduction parameters, we solve the least-squares problem

\[
\min_{a_1,a_2} \frac{1}{2}(f_1^2 + f_2^2),
\]

(2)

where \( f_1 = u(r_c) - u_c \), \( f_2 = u(r_{12}) - u_{12} \), where \( u \) is computed from \( a_1, a_2 \) by solving equation (1), and where the measured temperatures are \( u_c = 72 \) and \( u_{12} = 60 \).

The MATLAB function \texttt{temperatures}, contained in a file \texttt{temperatures.m} that can be downloaded from the course home page, takes two arguments: a two-vector \( a \) containing the heat conduction parameters, and \( r \), the distance to the axis of the cylinder. The function solves equation (1) and returns in its first output argument the temperature at position \( r \). The second and third output argument of \texttt{temperatures} returns the derivative of the temperature with respect to \( a_1 \) and \( a_2 \), respectively.

The function \texttt{param} contains the numerical values of parameters \( r_c, r_{12}, R \), and \( g \). The file \texttt{param.m} can also be downloaded from the course home page.

\textbf{Tasks}

1. Problem (2) is an unconstrained nonlinear optimization problem. Write a MATLAB program that solves the problem using the routine \texttt{fminunc} in MATLAB’s Optimization Toolbox. Use the following syntax to call \texttt{fminunc}:

\[
\text{options = optimset('GradObj','on','Hessian','off');}
\text{a = fminunc(@fun,a0,options);}
\]

The options above specify that gradients but no Hessians are available. Try both the default “large-scale” algorithm and the “medium-scale” one. The latter is called by adding the options ‘\texttt{LargeScale},’ ‘\texttt{off}’ to the parameters of
optimset. (The medium-scale algorithm is a BFGS quasi-Newton algorithm with line search.) Use \( a_0 = [1, 1] \) as initial guess for \( \text{fminunc} \).

The \$\in\$ in the first argument of \( \text{fminunc} \) indicates a function handler, and tells \( \text{fminunc} \) to use the MATLAB function \( \text{fun} \) to evaluate function values and gradients. Write this function \( \text{fun} \) using temperatures for the calculation of temperatures and temperature derivatives. Here, the objective function is

\[
    f = \frac{1}{2}(f_1^2 + f_2^2),
\]

using the previous notation. To avoid unnecessary calculations, use the build-in MATLAB function \( \text{nargout} \):

\[
\begin{align*}
    \text{function } [f,g] &= \text{fun}(a) \\
    f &= \ldots & \text{Compute the objective function value at } a \\
    \text{if } \text{nargout} &> 1 & \text{fun called with two output arguments} \\
    \ldots & = \ldots & \text{Gradient of the function evaluated at } a \\
    \text{end}
\end{align*}
\]

For each of the optimizations you run, record carefully the number of function evaluations and the number of gradient evaluations. An easy way to record these number is to use MATLAB’s build in \( \text{profile} \), an alternative is to update two counters, defined as \( \text{global variables} \) within the function \( \text{fun} \). See the Help in MATLAB (for instance by typing \( \text{help profile} \) or \( \text{help global} \) at the MATLAB prompt) for more information.

2. Since problem (2) is a least-squares problem, an alternative to the above is to use the routine \( \text{lsqnonlin} \). Use the following syntax:

\[
\begin{align*}
    \text{options} &= \text{optimset}('\text{Jacobian}','\text{on}', '\text{LargeScale}','\text{on}'); \\
    a &= \text{lsqnonlin}(@\text{fun2},a_0,[]); \\
\end{align*}
\]

Try also the option \( '\text{LargeScale}', '\text{off}' \) (then a Levenberg–Marquardt algorithm is used). Moreover, try \( '\text{LargeScale}', '\text{off}' \) in combination with \( '\text{LevenbergMarquardt}', '\text{off}' \) (then a Gauss–Newton algorithm is used). To study the sensitivity with respect to initial guesses, perform all numerical experiments with \( \text{lsqnonlin} \) using initial guess \( a_0 = [1, 1] \) as well as \( a_0 = [75, 75] \).

Note that the MATLAB function \( \text{fun2} \) to be called by \( \text{lsqnonlin} \) is completely different from the one used in part 1! The function \( \text{fun2} \) should return the vector \( (f_1, f_2)^T \) as its first argument (and not the objective function!) Moreover, the second argument should be the Jacobian of \( (f_1, f_2)^T \), that is, the matrix

\[
    J = \left( \begin{array}{cc}
    \frac{\partial f_1}{\partial a_1} & \frac{\partial f_1}{\partial a_2} \\
    \frac{\partial f_2}{\partial a_1} & \frac{\partial f_2}{\partial a_2}
    \end{array} \right).
\]

Discuss the performance for the different options and compare with part 1.