

### EXAMPLE1.9-2: Cryogenic Current Leads

It is often necessary to supply a cryogenic experiment or apparatus with electrical current. Some examples include current for superconducting electronics and magnets, to energize a resistance based temperature sensor, and to energize a heater used for temperature control. In any of these cases, careful design of the wires that are used to supply and return the current to the facility is important. The heat transfer to the cryogenic device from these wires should be minimized as this energy must be removed either by a refrigeration system (i.e., a cryocooler) or by consumption of a relatively expensive cryogen (e.g., by the boil off of liquid helium or liquid nitrogen). There is an optimal wire diameter for any given application that minimizes this parasitic heat transfer to the device.

Figure 1 illustrates two current leads, each carrying  $I = 100$  ampere (one supply and the other return). These current leads extend from the room temperature wall of the vacuum vessel, where the wire material is at  $T_H = 20^\circ\text{C}$ , to the experiment, where the wire material is at  $T_C = 50$  K. The length of both current leads is  $L = 1.0$  m and their diameter,  $D$ , should be optimized. The vacuum in the vessel prevents any convection heat transfer from the surface of the wires. However, the surface of the wires radiate to their surroundings, which may be assumed to be at  $T_H = 20^\circ\text{C}$ . The external surface of the wires has emissivity,  $\varepsilon = 0.5$ .

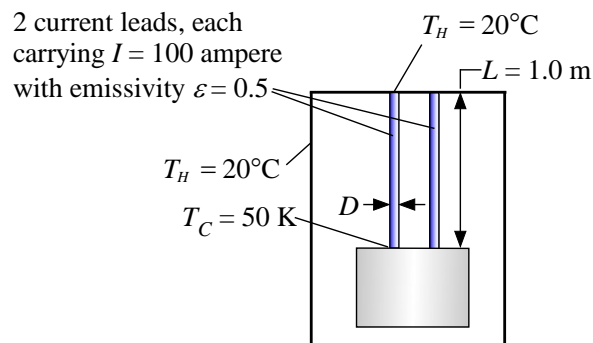


Figure 1: Cryogenic current leads.

The leads are made of oxygen free, high-conductivity copper; the thermal conductivity and resistivity of copper can vary substantially at cryogenic temperatures depending on the purity and history (e.g., whether it has been annealed or not) of the material. The purity of the metal is often expressed as the Residual Resistivity Ratio (RRR) which is defined as the ratio of a metal's electrical resistivity at 273 K to that at 4.2 K. Oxygen free, high conductivity (OFHC) has an RRR of approximately 200. The thermal conductivity and electrical resistivity of RRR 200 copper as a function of temperature is provided in Table 1 (Iwasa (1994)).

Table 1: Thermal conductivity and electrical resistivity of OFHC copper.

Temperature	Thermal Conductivity	Electrical Resistivity
500 K	4.31 W/cm-K	3.19 $\mu\text{ohm-cm}$
400 K	4.15 W/cm-K	2.49 $\mu\text{ohm-cm}$
300 K	3.99 W/cm-K	1.73 $\mu\text{ohm-cm}$
250 K	4.04 W/cm-K	1.39 $\mu\text{ohm-cm}$
200 K	4.11 W/cm-K	1.06 $\mu\text{ohm-cm}$
150 K	4.24 W/cm-K	0.72 $\mu\text{ohm-cm}$
125 K	4.34 W/cm-K	0.54 $\mu\text{ohm-cm}$
100 K	4.71 W/cm-K	0.36 $\mu\text{ohm-cm}$
90 K	4.98 W/cm-K	0.29 $\mu\text{ohm-cm}$
80 K	5.43 W/cm-K	0.22 $\mu\text{ohm-cm}$
70 K	6.25 W/cm-K	0.15 $\mu\text{ohm-cm}$
60 K	7.83 W/cm-K	0.098 $\mu\text{ohm-cm}$
55 K	9.11 W/cm-K	0.076 $\mu\text{ohm-cm}$
50 K	11.0 W/cm-K	0.057 $\mu\text{ohm-cm}$

a.) Develop a numerical model in MATLAB that can predict the rate of heat transfer to the cryogenic experiment from the pair of current leads

The input conditions are entered in a MATLAB function EXAMPLE1p9\_2.m; the two arguments to the function are diameter (the variable D) and number of nodes (the variable N) as we know that these parameters will be varied during the verification and optimization process. Any of the other parameters could be made into additional arguments in order to facilitate additional parametric studies or optimization.

```
function[]=EXAMPLE1p9_2(D,N)
```

```

I=100;           %current (amp)
T_H=20+273.2;   %hot temperature (K)
T_C=50;         %cold temperature (K)
L=1;           %length of lead (m)
eps=0.5;       %emissivity of lead surface (-)
sigma=5.67e-8; %Stefan-Boltzmann constant (W/m^2-K^4)

```

Notice that we have not, to this point, specified what parameters are returned when the function executes (i.e., there are no variables listed between the square brackets in the function header).

Functions are defined (at the bottom of the M-file) that return the conductivity and electrical resistivity of the OFHC copper; the interp1 function is used to carry out interpolation on the data provided in Table 1 using a cubic spline technique.

```
%----Property functions-----
```

```

function[k]=k_cu(T)
%returns the thermal conductivity (W/m-K) given temperature (K)
Td=[500,400,300,250,200,150,125,100,90,80,70,60,55,50]; %temperature data (K)
kd=[4.31,4.15,3.99,4.04,4.11,4.24,4.34,4.71,4.98,5.43,6.25,7.83,9.11,11.0]*100;
%conductivity data (W/m-K)
k=interp1(Td,kd,T);
end

```

```

function[rho_e]=rho_e_cu(T)
    %returns the electrical resistivity (ohm-m) given temperature (K)
    Td=[500,400,300,250,200,150,125,100,90,80,70,60,55,50]; %temperature data (K)
    rho_ed=[3.19,2.49,1.73,1.39,1.06,0.72,0.54,0.36,0.29,0.22,0.15,0.098,0.076,0.057]/(1e6*100);
    %electrical resistivity data (ohm-m)
    rho_e=interp1(Td,rho_ed,T);
end

```

The first step is to position the nodes throughout the computational domain. For this problem, the nodes will be distributed uniformly as shown in Figure 2 so that:

$$x_i = \frac{(i-1)}{(N-1)}L \quad \text{for } i = 1 \dots N$$

The distance between adjacent nodes ( $\Delta x$ ) is:

$$\Delta x = \frac{L}{(N-1)}$$

The MATLAB code that accomplishes these assignments is:

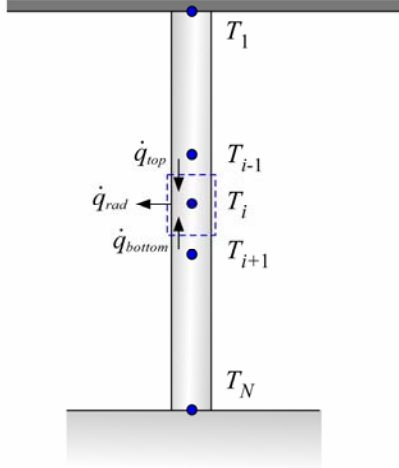
```

%Position nodes
for i=1:N
    x(i,1)=(i-1)*L/(N-1);    %position of each node (m)
end
DELTAx=L/(N-1);            %distance between adjacent nodes (m)

```

A control volume for an internal node is shown in Figure 2; the control volume experiences conduction heat transfer from the adjacent nodes above and below ( $\dot{q}_{top}$  and  $\dot{q}_{bottom}$ , respectively), as well as radiation ( $\dot{q}_{rad}$ ) and generation due to the ohmic dissipation associated with the current ( $\dot{g}$ ). An energy balance for the control volume is:

$$\dot{q}_{top} + \dot{q}_{bottom} + \dot{g} = \dot{q}_{rad} \quad (1)$$



**Figure 2: Control volume for an internal node and associated energy terms**

The conductivity used to approximate the conduction heat transfer rates must be evaluated at the temperature of the boundaries to avoid energy balance violations, as discussed in Section 1.4.3. With this understanding, these rate equations become:

$$\dot{q}_{top} = k_{T=(T_i+T_{i-1})/2} \frac{\pi D^2}{4 \Delta x} (T_{i-1} - T_i) \quad (2)$$

$$\dot{q}_{bottom} = k_{T=(T_i+T_{i+1})/2} \frac{\pi D^2}{4 \Delta x} (T_{i+1} - T_i) \quad (3)$$

The rate of thermal energy generation is evaluated using the resistivity evaluated at the temperature of each node:

$$\dot{g} = \rho_{e,T=T_i} \frac{4 \Delta x}{\pi D^2} I^2 \quad (4)$$

Radiation heat transfer is discussed briefly in Section 1.2.6 and is presented in detail in Chapter 10. The rate of radiation heat transfer is given by:

$$\dot{q}_{rad} = \varepsilon \sigma \pi D \Delta x (T_i^4 - T_h^4) \quad (5)$$

where  $\varepsilon$  is the emissivity of the surface of the leads and  $\sigma$  is the Stefan-Boltzmann constant. Substituting Eqs. (2) through (5) into Eq. (1) leads to:

$$k_{T=(T_i+T_{i-1})/2} \frac{\pi D^2}{4 \Delta x} (T_{i-1} - T_i) + k_{T=(T_i+T_{i+1})/2} \frac{\pi D^2}{4 \Delta x} (T_{i+1} - T_i) + \rho_{e,T=T_i} \frac{4 \Delta x}{\pi D^2} I^2 = \varepsilon \sigma \pi D \Delta x (T_i^4 - T_h^4) \quad (6)$$

for  $i = 2..(N-1)$

The remaining equations specify the boundary temperatures:

$$T_1 = T_H \quad (7)$$

$$T_N = T_C \quad (8)$$

Equations (6) through (8) are a set of  $N$  equations in the  $N$  unknown temperatures; however, the temperature dependence of the material properties ( $k$  and  $\rho_e$ ) as well as the non-linear rate equation associated with radiation heat transfer cause the system of equations to be non-linear. Therefore, a relaxation technique will be employed to obtain the solution; the relaxation process is discussed in Section 1.5.6. The assumed solution ( $\hat{T}$ ) will be successively substituted with the predicted solution; this process will continue until the assumed and predicted solutions agree to within an acceptable tolerance. The solution proceeds by assuming a temperature distribution that can be used to evaluate the coefficients in the linearized equations. A linear temperature distribution provides a reasonable start for the iteration:

$$\hat{T}_i = T_H + \frac{(i-1)}{(N-1)} (T_H - T_C) \quad \text{for } i = 1..N$$

```
%Start relaxation with a linear temperature distribution
```

```
for i=1:N
```

```
    Tg(i,1)=T_H-(T_H-T_C)*(i-1)/(N-1);
```

```
end
```

In order to solve this problem using MATLAB, we will need a set of linear equations; linear equations cannot contain products of the unknown temperatures with other unknown temperatures or functions of the unknown temperatures. Therefore, the temperature-dependent material properties must be evaluated at the assumed temperatures ( $\hat{T}$ ):

$$\dot{q}_{top} = k_{T=(\hat{T}_i+\hat{T}_{i-1})/2} \frac{\pi D^2}{4 \Delta x} (T_{i-1} - T_i) \quad (9)$$

$$\dot{q}_{bottom} = k_{T=(\hat{T}_i+\hat{T}_{i+1})/2} \frac{\pi D^2}{4 \Delta x} (T_{i+1} - T_i) \quad (10)$$

$$\dot{g} = \rho_{e,T=\hat{T}_i} \frac{4 \Delta x}{\pi D^2} I^2 \quad (11)$$

The fourth power temperature terms cause the radiation equation, Eq. (5), to be non-linear. Therefore, it is necessary to linearize the radiation equation so that it can be placed in matrix format, as was done for the material properties. The radiation terms can be linearized most conveniently using the same factorization that was previously used to define a radiation resistance in Section 1.2.6:

$$\dot{q}_{rad} = \sigma \varepsilon \pi D \Delta x (\hat{T}_i^2 + T_H^2) (\hat{T}_i + T_H) (T_i - T_H) \quad (12)$$

Substituting the linearized rate equations, Eqs. (9) through (12), into the energy balance for an internal node, Eq. (1), leads to:

$$\begin{aligned} k_{T=(\hat{T}_i+\hat{T}_{i-1})/2} \frac{\pi D^2}{4 \Delta x} (T_{i-1} - T_i) + k_{T=(\hat{T}_i+\hat{T}_{i+1})/2} \frac{\pi D^2}{4 \Delta x} (T_{i+1} - T_i) + \rho_{e,T=\hat{T}_i} \frac{4 \Delta x}{\pi D^2} I^2 \\ = \sigma \varepsilon \pi D \Delta x (\hat{T}_i^2 + T_H^2) (\hat{T}_i + T_H) (T_i - T_H) \end{aligned} \quad (13)$$

for  $i = 2..(N-1)$

Equations (7), (8), and (13) must be placed in matrix format:

$$\underline{\underline{A}} \underline{X} = \underline{b}$$

where  $\underline{X}$  is a vector of unknown temperatures,  $\underline{\underline{A}}$  is a matrix containing the coefficients of each equation, and  $\underline{b}$  is a vector containing the constant terms for each equation. The matrix  $\underline{\underline{A}}$  is declared as sparse in MATLAB; note that Eq. (13) indicates that there are at most three entries in each row of  $\underline{\underline{A}}$ :

```
%Setup A and b
A=spalloc(N,N,3*N);
b=zeros(N,1);
```

Equation (7) can be placed in row 1 of the matrix equation:

$$T_1 \underbrace{[1]}_{A_{1,1}} = \underbrace{T_H}_{b_1} \quad (14)$$

and Eq. (8) can be placed in row  $N$  of the matrix equation:

$$T_N \underbrace{[1]}_{A_{N,N}} = \underbrace{T_C}_{b_N} \quad (15)$$

Equation (13) must be placed in a form that makes it clear which row and column each coefficient should be entered into the matrix:

$$\begin{aligned}
& T_i \left[ \underbrace{-k_{T=(\hat{T}_i+\hat{T}_{i-1})/2} \frac{\pi D^2}{4 \Delta x} - k_{T=(\hat{T}_i+\hat{T}_{i+1})/2} \frac{\pi D^2}{4 \Delta x} - \sigma \varepsilon \pi D \Delta x (\hat{T}_i^2 + T_H^2)}_{A_{i,j}} (\hat{T}_i + T_H) \right] \\
& + T_{i-1} \left( \underbrace{k_{T=(\hat{T}_i+\hat{T}_{i-1})/2} \frac{\pi D^2}{4 \Delta x}}_{A_{i,i-1}} \right) + T_{i+1} \left( \underbrace{k_{T=(\hat{T}_i+\hat{T}_{i+1})/2} \frac{\pi D^2}{4 \Delta x}}_{A_{i,i+1}} \right) = \\
& \underbrace{-\sigma \varepsilon \pi D \Delta x (\hat{T}_i^2 + T_H^2) (\hat{T}_i + T_H) T_H - \rho_{e,T=\hat{T}_i} \frac{4 \Delta x}{\pi D^2} I^2}_{b_i} \\
& \text{for } i = 2..(N-1)
\end{aligned} \tag{16}$$

The numerical solution is placed within a while loop that checks for convergence of the relaxation scheme. The variable `err` is used to terminate the while loop and represents the average, absolute error between the assumed at predicted temperature distribution. (There are other criteria that could be used, but this is sufficient for most problems.) The while loop is terminated when the variable `err` decreases to less than the input parameter `tol`, which represents the convergence tolerance for the relaxation process. Initially, the value of `err` is set to a value greater than `tol` to ensure that the while loop executes at least one time.

```

err=999;           %error that terminates the while loop (K)
tol=0.1;          %criteria for terminating the while loop (K)
while(err>tol)

    end
end

```

Within the while loop, the matrix is filled in using the coefficients suggested by Eq. (14),

```

%specify the hot end temperature
A(1,1)=1;
b(1,1)=T_h;

```

Eq. (15),

```

%specify the cold end temperature
A(N,N)=1;
b(N,1)=T_C;

```

and Eq. (16).

```

%internal nodes
for i=2:(N-1)
    A(i,i)=-k_cu((Tg(i+1,1)+Tg(i,1))/2)*pi*D^2/(4*DELTAx)-...
            k_cu((Tg(i-1,1)+Tg(i,1))/2)*pi*D^2/(4*DELTAx)-...
            sigma*eps*pi*D*DELTAx*(Tg(i,1)^2+T_H^2)*(Tg(i,1)+T_H);
    A(i,i-1)=k_cu((Tg(i-1,1)+Tg(i,1))/2)*pi*D^2/(4*DELTAx);
    A(i,i+1)=k_cu((Tg(i+1,1)+Tg(i,1))/2)*pi*D^2/(4*DELTAx);
end

```

```

b(i,1)=-rho_e_cu(Tg(i,1))*4*DELTAx*I^2/(pi*D^2)-...
sigma*eps*pi*D*DELTAx*(Tg(i,1)^2+T_H^2)*(Tg(i,1)+T_H)*T_H;
end

```

Note that the three periods in the above code is a line break; it indicates that the code is continued on the subsequent line. Keep in mind the form of the matrix equation; the equation that governs control volume  $i$  must be placed in row  $i$  of  $\underline{\underline{A}}$  while the coefficient multiplying the unknown temperature  $T_i$  should be placed in column  $i$  of  $\underline{\underline{A}}$ . The matrix equation is solved and the error between the assumed and predicted temperature is computed.

$$err = \frac{1}{N} \sum_{i=1}^N |T_i - \hat{T}_i|$$

The final step in the while loop is to update the assumed temperature distribution with the predicted temperature distribution.

```

T=full(A\b);
err=sum(abs(T-Tg))/N %compute the error
Tg=T; %update the guess temperature array

```

Note that the full command in the above code converts the sparse matrix, T, that results from the operation on the sparse matrix A into a full matrix.

The header of the function is modified to specify the output arguments x and T:

```
function[x,T]=EXAMPLE1p9_2(D,N)
```

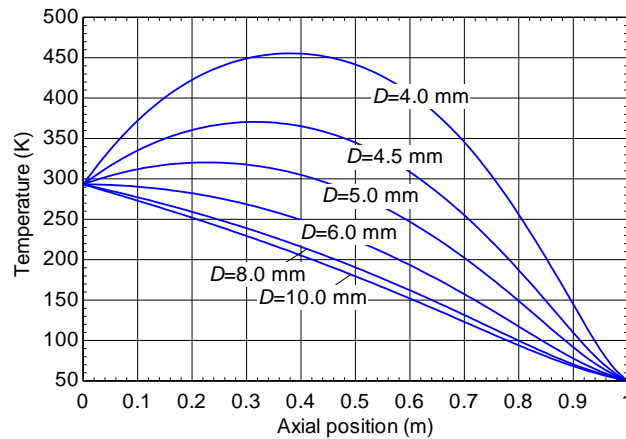
Because the statement that calculates the variable err is not terminated with a semicolon, the result of the calculation will be echoed in the workspace, allowing you to keep track of the progress. If you call this program with a diameter of 5.0 mm you should see:

```

>> [x,T]=EXAMPLE1p9_2(0.005,100);
err =
  41.2789
err =
  16.1165
err =
  6.2792
err =
  2.4864
err =
  1.0023
err =
  0.4110
err =
  0.1685
err =
  0.0693

```

The relaxation process had to iterate several times in order to converge due to the nonlinearity of the problem. Figure 3 illustrates the temperature distribution in the wire for several different values of the diameter.



**Figure 3: Temperature distribution in the current lead for various values of the diameter.**

Notice that the smaller diameter leads result in large amounts of ohmic dissipation and therefore the wire tends to become hotter and the temperature gradient at the cold end increases. For a given temperature gradient, larger diameter leads will result in a higher rate of heat transfer to the cold end. There is a balance between these effects that results in an optimal diameter.

The heat transferred to the cold end ( $\dot{q}_c$ ) is calculated using the numerical solution:

$$\dot{q}_c = k_{T=(\hat{T}_N + \hat{T}_{N-1})/2} \frac{\pi D^2}{4 \Delta x} (T_{N-1} - T_N)$$

or, in MATLAB:

```
q_dot_c=k_cu*((Tg(N)+Tg(N-1))/2)*pi*D^2*(T(N-1)-T(N))/(4*DELTAx);
%heat transfer to cold end
```

The function header is modified so that  $\dot{q}_c$  is also returned:

```
function[q_dot_c,x,T]=EXAMPLE1p9_2(D,N)
```

It is necessary to verify that the solution has a sufficient number of nodes and, if possible, verify the result against an analytical solution. The critical parameter for the solution is the rate of heat transfer to the experiment per current lead; therefore Figure 4 illustrates  $\dot{q}_c$  as a function of the number of nodes in the solution,  $N$ . The information shown in Figure 4 was generated quickly using the script varyN (below), which calls the function EXAMPLE1p9\_2 multiple times with varying values of  $N$ :

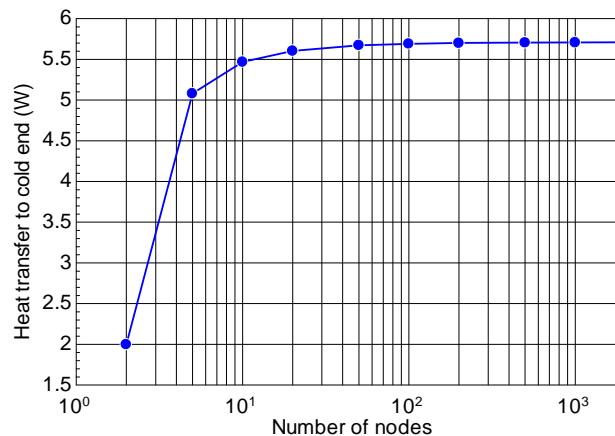
```
%Script varyN.m
clear all;
```

```

D=0.005;
N=[2,5,10,20,50,100,200,500,1000,2000]';
for i=1:10
    i
    [q_dot_c(i,1),x,T]=EXAMPLE1p9_2(D,N(i));
end

```

The clear all statement at the beginning of the script clears all variables from memory. Using of clear all statement is often a good idea as it prevents previous elements (e.g., from previous runs) of the variables q\_dot\_c or N from being retained. If you had previously run the script varyN with more than 10 runs then N and q\_dot\_c would exist in memory with more than 10 elements. Running the script varyN as shown above would overwrite the 1<sup>st</sup> 10 elements of these variables, but leave all subsequent elements which could lead to confusion.



**Figure 4: Heat transferred to the cold end per lead as a function of the number of nodes for a 5.0 mm lead.**

Figure 4 suggests that at least 100 nodes should be used for sufficient accuracy. In the absence of any radiation heat transfer ( $\varepsilon = 0$ ) and with constant resistivity and conductivity, it is possible to compare the numerical solution to the analytical solution for the temperature in a generating wall with fixed end conditions. This result was derived in Section 1.3.2 and is repeated below:

$$T = \frac{\dot{g}''' L^2}{2k} \left[ \frac{x}{L} - \left( \frac{x}{L} \right)^2 \right] - \frac{(T_H - T_C)}{L} x + T_H$$

where the volumetric generation is given by:

$$\dot{g}''' = \frac{16 I^2 \rho_e}{\pi^2 D^4}$$

```

%constant property analytical solution
g_dot_vol=16*I^2*rho_e_cu(T_H)/(pi^2*D^4);
for i=1:N
    T_an(i,1)=g_dot_vol*L^2*((x(i)/L)-(x(i)/L)^2)/(2*k_cu(T_H))-...
    (T_H-T_C)*x(i)/L+T_H;
end

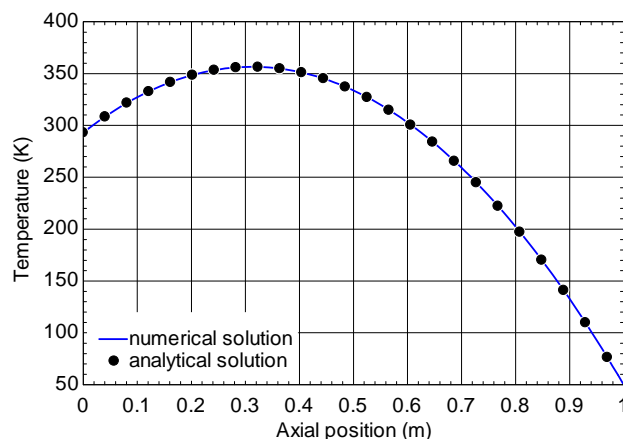
```

The property functions in MATLAB are modified to return, temporarily, constant values of  $k = 200$  W/m-K and  $\rho_e = 1 \times 10^{-8}$  ohm-m.

```
%----Property functions-----
function[k]=k_cu(T)
    %returns the thermal conductivity (W/m-K) given temperature (K)
    Td=[500,400,300,250,200,150,125,100,90,80,70,60,55,50]; %temperature data (K)
    kd=[4.31,4.15,3.99,4.04,4.11,4.24,4.34,4.71,4.98,5.43,6.25,7.83,9.11,11.0]*100;
        %conductivity data (W/m-K)
    %k=interp1(Td,kd,T);
    k=200;
end

function[rho_e]=rho_e_cu(T)
    %returns the electrical resistivity (ohm-m) given temperature (K)
    Td=[500,400,300,250,200,150,125,100,90,80,70,60,55,50]; %temperature data (K)
    rho_ed=[3.19,2.49,1.73,1.39,1.06,0.72,0.54,0.36,0.29,0.22,0.15,0.098,0.076,0.057]/(1e6*100);
        %electrical resistivity data (ohm-m)
    %rho_e=interp1(Td,rho_ed,T);
    rho_e=1e-8;
end
```

The emissivity is set to 0 and the MATLAB code is run for a 5.0 mm diameter lead. The temperature distribution predicted by the MATLAB code is compared with the analytical solution in Figure 5. Note that with these modifications (i.e., constant  $k$  and  $\rho_e$  and  $\varepsilon = 0$ ), the problem becomes linear and therefore a single iteration is required in order to reduce the relaxation error to 0. The analytical solution would provide a better set of guess values than a linear temperature distribution to start the relaxation process.



**Figure 5: Comparison of the analytical and numerical solutions in the limit that  $k = 200$  W/m-K (constant),  $\rho_e = 1e-8$  ohm-m (constant) and  $\varepsilon = 0$  for a 5.0 mm diameter wire.**

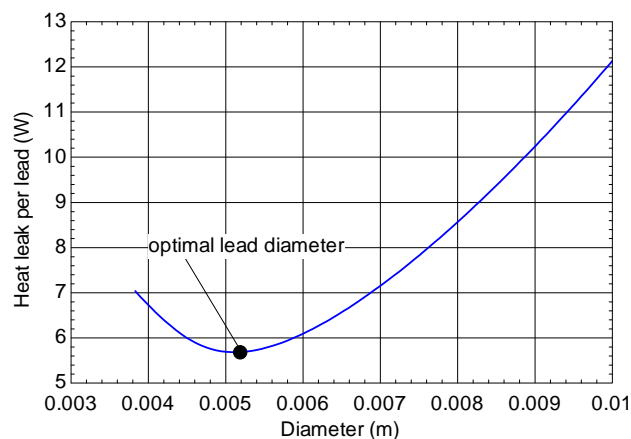
Finally, it is possible to parametrically vary the wire diameter,  $D$ , in order to minimize the heat flow to the cold end of the current lead. The code is returned to its original, non-linear form. A script (varyd) is used to call the function multiple times with various diameters in order to carry out a parametric study of this parameter:

```

%Script varyd.m
clear all;
N=100;
D=linspace(0.0038,0.01,100)'; %generate 100 values of D between 0.0038 and 0.01 [m]
for i=1:100
    [q_dot_c(i,1),x,T]=EXAMPLE1p9_2(D(i),N);
end

```

Figure 6 illustrates the heat leak to the cold end as a function of wire diameter (note that the functions for  $k$  and  $\rho_e$  were reset and the value of  $\varepsilon$  was set to 0.50) and shows that there is a clear optimal diameter around 5.1 mm for this application. Smaller values of  $D$  lead to excessive self-heating whereas larger values provide a large path for conduction heat transfer.



**Figure 6: Heat leak to cold end of each current lead as a function of diameter.**

MATLAB has powerful, built-in optimization algorithms that allow you to automate the process of determining the optimal diameter. The MATLAB function `fminbnd` is the simplest available and carries out a bounded, 1-D minimization. The function `fminbnd` is called according to:

```
x_opt = fminbnd(function,x1,x2)
```

where `function` is the name of a function that requires a single argument and provides a single output (that should be minimized) and `x1` and `x2` are the lower and upper bounds of the argument to use for the minimization. First, it is necessary to modify the function `EXAMPLE1p9_2` so that it takes a single argument (`D`) and returns a single output (`q_dot_c`):

```
function[q_dot_c]=EXAMPLE1p9_2(D)
```

```
N=100; %number of nodes (-)
```

Then the `fminbnd` function can be called directly from the workspace:

```
>> D_opt=fminbnd('EXAMPLE1p9_2',0.004,0.01);
```

and it identifies the optimal diameter as being 5.1 mm.

```
>> D_opt
D_opt =
    0.0051
```

Note that the calculation of the variable `err` in the function `EXAMPLE1p9_2` is terminated with a semicolon so that the error is not echoed to the workspace during each iteration. The function `fminbnd` will return the optimized value of the heat leak as well by adding an additional output argument to the `fminbnd` call:

```
>>[D_opt,q_dot_c_min]=fminbnd('EXAMPLE1p9_2',0.004,0.01);
```

which indicates that the optimal value of the heat leak is 5.57 W.

```
>> q_dot_c_min
q_dot_c_min =
    5.6815
```

It is possible to control the details of the optimization using an optional 4<sup>th</sup> input argument that sets the optimization parameters; the easiest way to set this last argument is using the `optimset` command. If you enter

```
>> help optimset
```

into the workspace then a complete list of the parameters that can be controlled will be returned. It is possible, for example, to display the progress of the optimization using:

```
>> [D_opt,q_dot_c_min]=fminbnd('EXAMPLE1p9_2',0.004,0.01,optimset('Display','iter'))
```

Func-count	x	f(x)	Procedure
1	0.0062918	6.35363	initial
2	0.0077082	8.12533	golden
3	0.00541641	5.73753	golden
4	0.0043798	6.14038	parabolic
5	0.00523819	5.68985	parabolic
6	0.00515231	5.6824	parabolic
7	0.00511897	5.68148	parabolic
8	0.00508564	5.68212	parabolic

Optimization terminated:  
the current x satisfies the termination criteria using `OPTIONS.ToIX` of 1.000000e-004

```
D_opt =
    0.0051
q_dot_c_min =
    5.6815
```

The first argument to `optimset` specifies the parameter to be controlled ('Display', which controls the level of display) and the second indicates its new value ('iter', which indicates that the progress should be displayed after each iteration).

It would be inconvenient to use the `fminbnd` function to carry out a parametric variation of how the optimal value of the variables `D` and `q_dot_c` are affected by current or some other parameter. In its current format, it is not possible to pass the value of the current (`I`) to the `fminbnd` function and therefore the study would have to be carried out manually by running the `fminbnd` function and then changing the value of the variable `I` within the function `EXAMPLE1p9_2`. This process would become tedious and can be avoided by parameterizing the function. For example, suppose you want to determine how the optimal value of diameter changes with current. First, include current as an additional argument to the function:

```
function[q_dot_c]=EXAMPLE1p9_2(D,I)
```

```
N=100;          %number of nodes (-)
% I=100;        %current (amp)
```

If you try to repeat the optimization using the previous protocol you will receive an error:

```
>> [D_opt,q_dot_c_min]=fminbnd('EXAMPLE1p9_2',0.004,0.01)
??? Input argument "I" is undefined.
```

```
Error in ==> EXAMPLE9_2 at 42
      b(i,1)=-rho_e_cu(Tg(i,1))*4*DELTAx*I^2/(pi*D^2)-...
```

```
Error in ==> fminbnd at 182
x= xf; fx = funfcn(x,varargin{:});
```

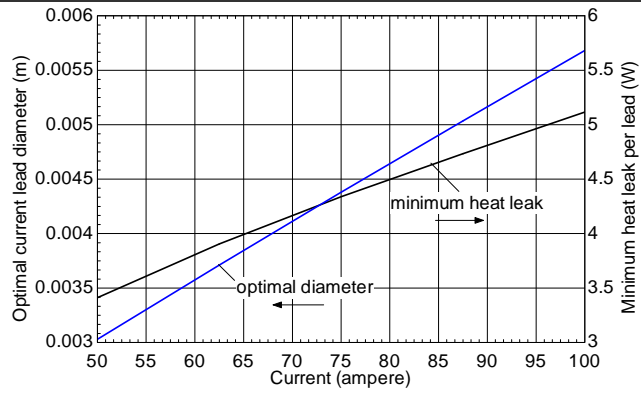
However, you can parameterize the function using a one-argument anonymous function that captures the value of `I` (set in the workspace) and calls `EXAMPLE1p9_2` with two arguments:

```
>> I=100;
>> [D_opt,q_dot_c_min]=fminbnd(@(D) EXAMPLE1p9_2(D,I),0.004,0.01)
D_opt =
    0.0051
q_dot_c_min =
    5.6815
```

Now it is possible to generate a script, `varyI`, that evaluates the optimized diameter and heat flow as a function of current.

```
%Script varyI.m
clear all;
I=linspace(1,100,10)';
for i=1:10
    [d_opt(i,1),q_dot_min(i,1)] = fminbnd(@(d) EXAMPLE1p9_2(d,I(i,1)),0.0025,0.01)
end
```

Figure 7 illustrates the optimal diameter and heat leak as a function of current for relatively high currents. Note that the optimal diameter and minimized heat leak are both approximately linear functions of the current.



**Figure 7: Optimal diameter and minimized heat leak to cold end of each current lead as a function of current.**