# Thermal Model Software

Andrew E. Slaughter
Montana State University
Department of Civil Engineering
Subzero Science and Engineering Research Facility

March 21, 2011

# Contents

# List of Tables

# List of Figures

# 1 Introduction

This manual is divided into three main sections. Section 2 explains the basic operation of the thermal software. Section 3 presents an interface that links the thermal model with Microsoft Excel, allowing inputs to be easily modified. Figure 1 contains a flow chart demonstrating how the various functions detailed in the first two sections (2 and 3) interact. Finally, in Section 4, a complete graphical user interface is briefly presented that operates as a stand-alone Windows application.

A few notational conventions are utilized throughout this user manual:

- Monospaced typeface indicates a MATLAB m-file, function, or variable (e.g., `sobol.m`).
- MATLAB code is provided in figure windows (e.g., Figure ?? when referenced many times).
- MATLAB code is also presented in-line with the text as:

```
>> 2+2
ans = 4
>>
```

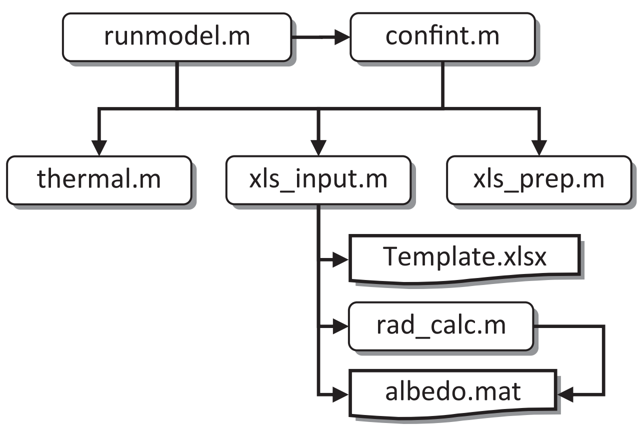



Figure 1: Flow chart demonstrating how the various functions discussed in Section 2 and 3 interact.

# 2 Basic Application

The basic operation of the thermal model is performed via the MATLAB command-line using two functions: `xls_prep.m` and `thermal.m` (the source code is included in Section 6). First, `xls_prep.m` is implemented, which requires three input arrays that contain the snow properties, atmospheric conditions, and model constants. The syntax for `xls_prep.m` is as follows:

```
function [s,a] = xls_prep(snow,atm,constants)
% XLS_PREP builds arrays for inputing into thermal model
%-------------------------------------------------------------------------
% SYNTAX:
%    [snow,atm] = prep_input(snow,atm,constants);
```

The `snow` variable may be arranged in two ways: as a uniform or a varying snowpack. If the snowpack is assumed uniform then `snow` is a 1-D array with six values (in order): depth (cm), density (kg/m$^3$), thermal conductivity (W/(m·K)), specific heat capacity (J/(gm·K)), snow temperature (°C), and extinction coefficient (1/m). If the snowpack varies then the array may be composed of any number of rows of the same parameters that dictates the different layers. The following MATLAB code provides example definitions of the `snow` variable:

```
>> snow = [50, 130, 0.06, 2030, -10, 70]
snow =
                50   130   0.06   2030   -10   70
>> snow = [0, 130, 0.06, 2030, -10, 70; 50, 180, 0.1, 2030, -5, 90;...
 100, 180, 0.1, 2030, -5, 90]
snow =
         0   130   0.06   2030   -10   70
         50   180   0.1   2030   -5   90
         100   180   0.1   2030   -5   90
>>
```

The first example defines a 50 cm thick snow pack with constant properties. The second example defines a 100 cm deep snowpack that increases in density, thermal conductivity, temperature, and extinction coefficient from 0 to 50 cm. Then from 50 to 100 cm the conditions remain constant. The `xls_prep.m` function performs linear interpolation between the rows according to the layer thickness. An additional seventh column is optional that specifies the extinction coefficient for the near-infrared wavebands, in this case the extinction coefficient previously mentioned is used for the visible waveband.

In similar fashion, the atmospheric conditions are defined in the `atm` variable, which includes nine parameters (in order): time (hours), incoming long-wave radiation (W/m$^2$), incoming short-wave radiation (W/m$^2$), albedo, wind speed (m/s), air temperature (°C), relative humidity (%), the lower boundary condition (°C), and air pressure (kPa). Two additional columns may also be defined that specifiy the incoming short-wave radiation and albedo for the near-infrared wavebands. Again, the short-wave radiation and albedo previously defined are then used as the values for the visible spectrum.

The model constants are defined in the `constants` variable, which must include the following (in order): latent heat of sublimation (kJ/kg), the latent heat transfer coefficient, the sensible heat transfer coefficient, the ratio of molecular weights of dry-air and water-vapor, the gas constant for water-vapor (kJ/(kg·K)), reference temperature (°C), reference vapor-pressure (kPa), the emissivity of snow, the layer thickness (cm), and time step (s).

Once the three input variable arrays are defined the thermal model may be executed, for example:

```
>> snow = [50, 130, 0.06, 2030, -10, 70];
>> atm = [0,240,0,0.82,1.7,-10,.2,-10,101; 10,240,500,0.82,1.7,-10,.2,-10,101];
>> contants = [2833,0.0023,0.0023,0.622,0.462,-5,0.402,0.95,1,60,1];
>> [S,A] = xls_prep(snow, atm, constants);
>> [T,Q] = thermal(S, A, constants);
```

The `thermal.m` function implements the finite-difference solution presented in Chapter **??**. This function outputs an array containing snow temperatures (`T`) as a function of model evaluation time (columns) and depth (rows). The various heat-fluxes—long-wave, sensible, latent, short-wave—are output in the `Q` variable in similar fashion.

# 3 Spreadsheet Application

## 3.1 General Application

To make the thermal model more powerful, two additional functions were developed—`xls_input.m` and `runmodel.m`—that provide an interface between MATLAB and Microsoft Excel. This allows the various input matrices previously explained to be easily developed. First, the required structure of the Excel file must be established. The Excel spreadsheet must be composed of three worksheets named "SnowProperties", "AtmosphericSettings", and "Constants". Each worksheet must be formatted in a specific fashion, as shown in Figures 2 and Figures 3.[1] Section 3.2 details some additional features available when using `xls_input.m`, particularly for the "Constants" worksheet.

Once the Excel file is setup as desired, the function `xls_input.m` is used to process the data contained in the spreadsheet. As was the case for the basic operation, the near-infrared columns are optional. For example, for the template.xlsx file available for download, the following code implements the thermal model:

```
>> filename = 'template.xlsx';
>> [s,a,c] = xls_input(filename);
>> [S,A] = xls_prep(s, a, c);
>> [T,Q] = thermal(S, A, c);
```

The `runmodel.m` function performs the above actions, groups the results into a data structure, and adds the ability to compute confidence level intervals for the snow temperatures. The confidence intervals are explained further in the following section. The code shown in Figure 4 implements the thermal model via `runmodel.m` and displays the data structure produced. The data structure and details regarding various optional inputs are explained in the help associated with the `runmodel.m`. The data structure was designed to be implemented via the graphical user interface (Section 4), as such the data structure may contain many model runs, as shown in Figure 4.

## 3.2 Additional Features

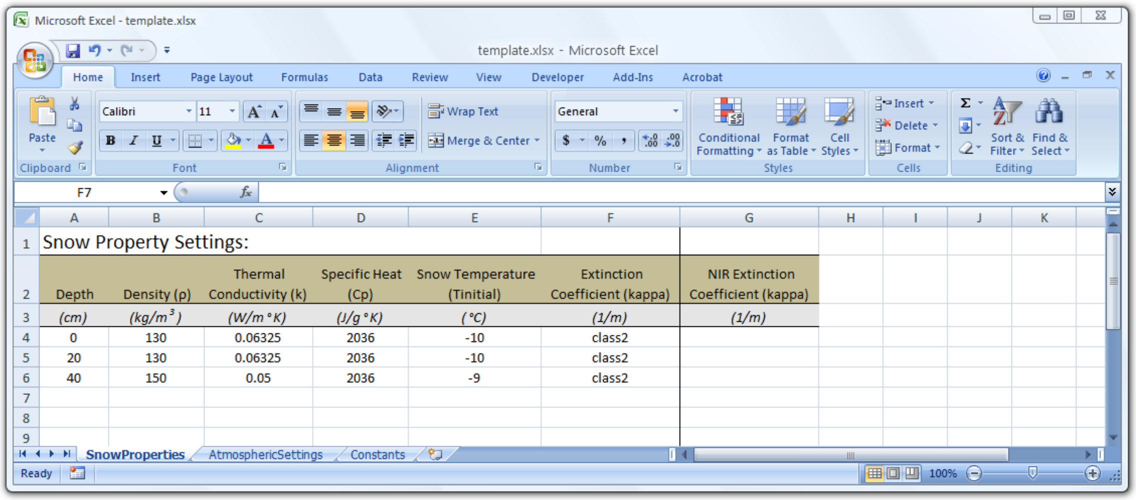
The usage of the function `xls_input.m` offers additional functionality for inputs, including the usage of tabulated snow micro-structure data, input multipliers, and confidence interval calculations.

### 3.2.1 Snow Micro-Structure

The snow albedo and extinction coefficient may be input into the Excel document using keys: *dXX*, *classX*, or *type*.[2] The *dXX* key allows the snow grain diameter to be used to compute albedo and extinction coefficient according to **?**, Eq. 2.25, p. 56, were the *XX* is a number representing the size of the grain in millimeters (e.g., "d5"). Figure 2b includes the implementation of this option. The *classX* key uses the tabulated values from **?**, Tab. 2.6, where *X* is a value one to six (e.g., "class2"). The type may be one of three strings: "fine", "medium", or "coarse", this option is only available for the computation of albedo. The usage of these keys results in the computation of the albedo from the information provided in **?**. The albedo and extinction coefficient calculations are preformed in the `albedo` and `extinction` sub-functions of `xls_input.m`.

[1] A template may be downloaded at: `www.coe.montana.edu/ce/subzero/snow/thermalmodel/template.xlsx`.

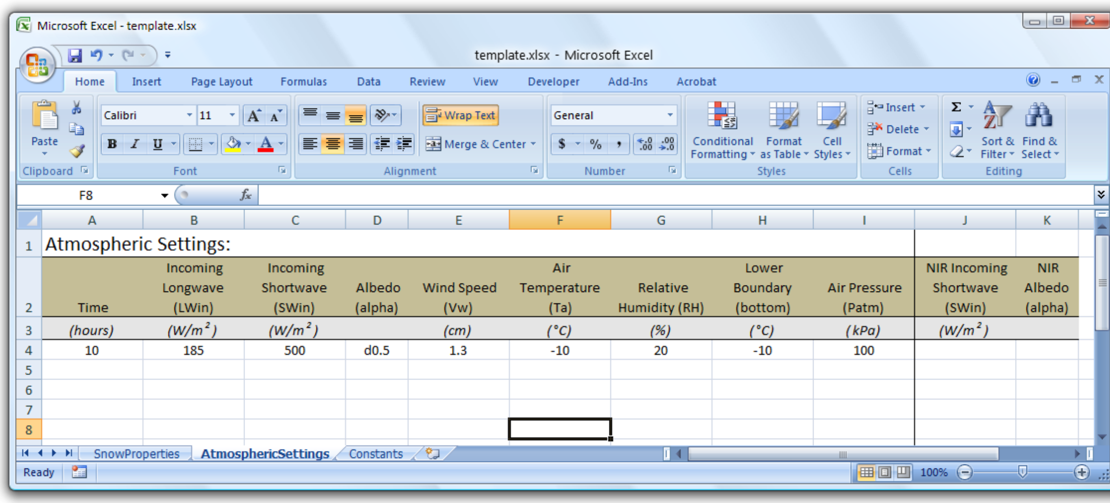
[2] The *italicized* keys are used to reference the inputs, the actual text as would be entered in the Excel worksheets is provide in quotes.



Snow Property Settings:

| Depth | Density (ρ) | Thermal Conductivity (k) | Specific Heat (Cp) | Snow Temperature (Tinitial) | Extinction Coefficient (kappa) | NIR Extinction Coefficient (kappa) |
|---|---|---|---|---|---|---|
| (cm) | (kg/m³) | (W/m °K) | (J/g °K) | (°C) | (1/m) | (1/m) |
| 0 | 130 | 0.06325 | 2036 | -10 | class2 | |
| 20 | 130 | 0.06325 | 2036 | -10 | class2 | |
| 40 | 150 | 0.05 | 2036 | -9 | class2 | |

(a) Snow Properties



Atmospheric Settings:

| Time | Incoming Longwave (LWin) | Incoming Shortwave (SWin) | Albedo (alpha) | Wind Speed (Vw) | Air Temperature (Ta) | Relative Humidity (RH) | Lower Boundary (bottom) | Air Pressure (Patm) | NIR Incoming Shortwave (SWin) | NIR Albedo (alpha) |
|---|---|---|---|---|---|---|---|---|---|---|
| (hours) | (W/m²) | (W/m²) | | (cm) | (°C) | (%) | (°C) | (kPa) | (W/m²) | |
| 10 | 185 | 500 | d0.5 | 1.3 | -10 | 20 | -10 | 100 | | |

(b) Atmospheric Settings

Figure 2: Example of the (a) "SnowProperties" and (b) "AtmosphericSettings" worksheets for Excel file read by `xls_input.m`.

In all cases, when the optional near-infrared columns are not utilized it is assumed that albedo and extinction coefficients are defined for the "all-wave" spectrum that includes both the visible and near-infrared spectrum. Using **?** the appropriated values are computed based on this spectrum via `rad_calc.m` (see Section 6.5).

Both `xls_input.m` and `rad_calc.m` require the `albedo.mat` file that contains the data structure shown in Figure 5. Each component of this structure contains a two-column numeric array, the

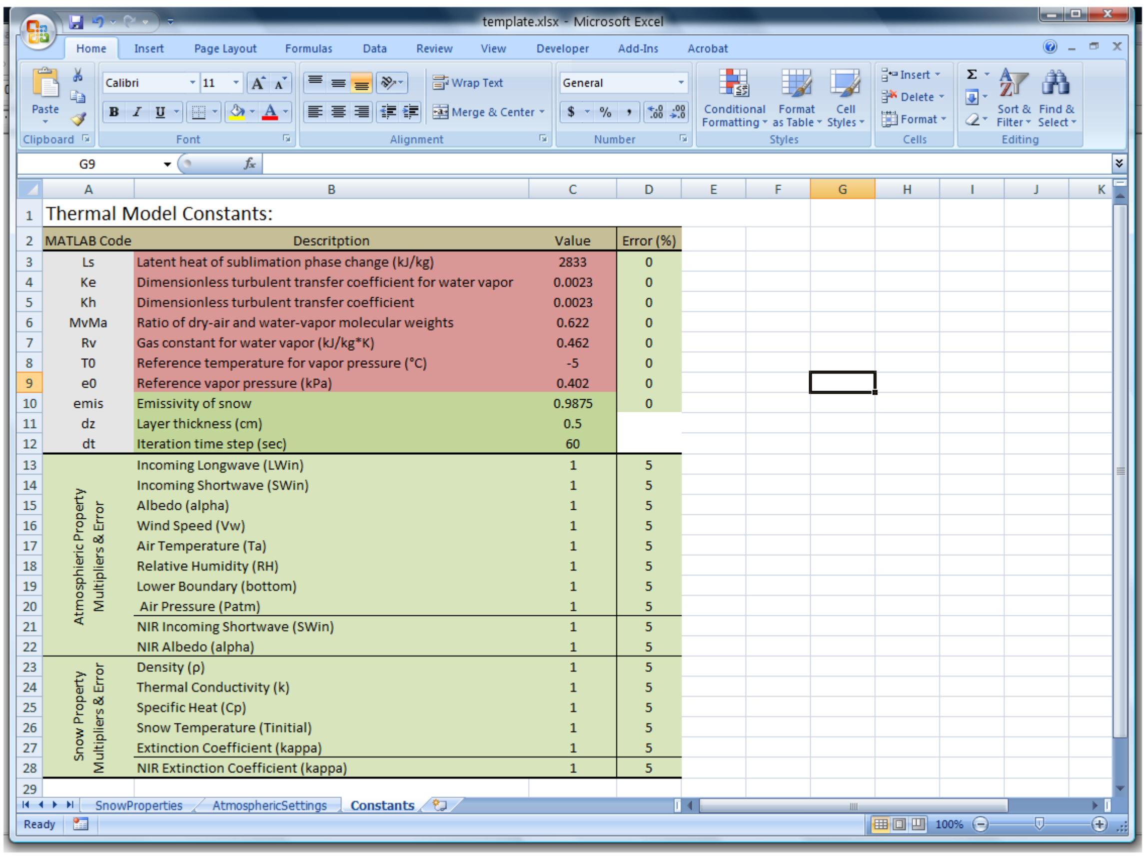
the first column of which provides the wavelength in nanometers. The second column of `x.atsm` contains solar irradiance as defined by ?. For the other items (e.g., `x.fine`), the second column contains albedo values as defined in ?.[3]

Finally, the snow density or the thermal conductivity may be automatically computed by using "auto" in either column, but not both. The desired density or thermal conductivity calculations are preformed using the relationships presented by ?.

### 3.2.2 Input Multipliers

To enable simple modification of entire columns of data, multipliers are provided on the "Constants" worksheet, as shown in Figure 3. The corresponding column from the other worksheets are simply multiplied by the values listed, allowing the user to quickly modify the various inputs.

[3]This file may downloaded at `www.coe.montana.edu/ce/subzero/snow/thermalmodel/albedo.mat`.



Thermal Model Constants:

| MATLAB Code | Descritption | Value | Error (%) |
|---|---|---|---|
| Ls | Latent heat of sublimation phase change (kJ/kg) | 2833 | 0 |
| Ke | Dimensionless turbulent transfer coefficient for water vapor | 0.0023 | 0 |
| Kh | Dimensionless turbulent transfer coefficient | 0.0023 | 0 |
| MvMa | Ratio of dry-air and water-vapor molecular weights | 0.622 | 0 |
| Rv | Gas constant for water vapor (kJ/kg*K) | 0.462 | 0 |
| T0 | Reference temperature for vapor pressure (°C) | -5 | 0 |
| e0 | Reference vapor pressure (kPa) | 0.402 | 0 |
| emis | Emissivity of snow | 0.9875 | 0 |
| dz | Layer thickness (cm) | 0.5 | |
| dt | Iteration time step (sec) | 60 | |
| Atmospheric Property Multipliers & Error | Incoming Longwave (LWin) | 1 | 5 |
| | Incoming Shortwave (SWin) | 1 | 5 |
| | Albedo (alpha) | 1 | 5 |
| | Wind Speed (Vw) | 1 | 5 |
| | Air Temperature (Ta) | 1 | 5 |
| | Relative Humidity (RH) | 1 | 5 |
| | Lower Boundary (bottom) | 1 | 5 |
| | Air Pressure (Patm) | 1 | 5 |
| | NIR Incoming Shortwave (SWin) | 1 | 5 |
| | NIR Albedo (alpha) | 1 | 5 |
| Snow Property Multipliers & Error | Density (ρ) | 1 | 5 |
| | Thermal Conductivity (k) | 1 | 5 |
| | Specific Heat (Cp) | 1 | 5 |
| | Snow Temperature (Tinitial) | 1 | 5 |
| | Extinction Coefficient (kappa) | 1 | 5 |
| | NIR Extinction Coefficient (kappa) | 1 | 5 |

Figure 3: Example of the "Constants" worksheets for Excel file read by `xls_input.m`.

```
>> filename = 'template.xlsx';
>> data = runmodel(filename)
data =
              xls: 'template.xlsx'
     bootsettings: []
             name: ''
             desc: ''
             time: '22-Mar-2010␣09:58:06'
                T: [82x601 double]
                Q: [81x601x5 double]
              snw: [81x7 double]
              atm: [601x11 double]
            const: [1x10 double]
            Tboot: []
            Qboot: []
            Sboot: []
            Aboot: []
            Cboot: []
>> data(2) = runmodel(filename); % multiple runs may be stored
>>
```

Figure 4: MATLAB implementation of `runmodel.m` and the resulting data structure.

```
>> x = load('albedo.mat')
x =
       astm: [2002x2 double]
       fine: [179x2 double]
     medium: [179x2 double]
     coarse: [179x2 double]
>>
```

Figure 5: Required data structure of `albedo.mat`.

### 3.2.3 Confidence Intervals

The `runmodel.m` function includes the ability to compute confidence intervals via `confint.m`, which uses the percentile bootstrap method presented by **?**. First, the percent error is prescribed by the values listed in the "Error" column on the "Constants" worksheet, as shown in Figure 3. These values allow the parameter to vary plus or minus this amount according to a normal distribution, such that the $n\sigma$ tails of this distribution are at these limits, where $n\sigma$ is the number of standard deviations. The graphical user interface described in the following sections provides the means for utilizing this feature.

## 4 Graphical User Interface

A graphical user interface (GUI), as shown in Figure 6, was develop to act as front-end to the software explained in the previous sections. This interface was deployed via MATLAB's `deploytool` tool and wrapped into an installable Windows-based program. The complete installer, TMsetup.exe, may be downloaded at: `www.coe.montana.edu/ce/subzero/snow/thermalmodel/TMsetup.exe`.

The stand-alone application may prompt the user to download a newer version, which is recommended. By agreeing to this prompt the website listing the associated files will automatically open in a browser. The only file that needs to be downloaded is model.exe, this file should replace the original that is located in the installation directory.

The GUI serves two functions, first it controls the operation of the `runmodel.m` function and manages the data structure produced by this function (see Section 3). This is done through the use of projects, which are nothing more than MATLAB mat-files that store the data structure produced by `runmodel.m`. However, the extension was changed to *.prj. The GUI also provides tools for the visualization of the input and output variables.

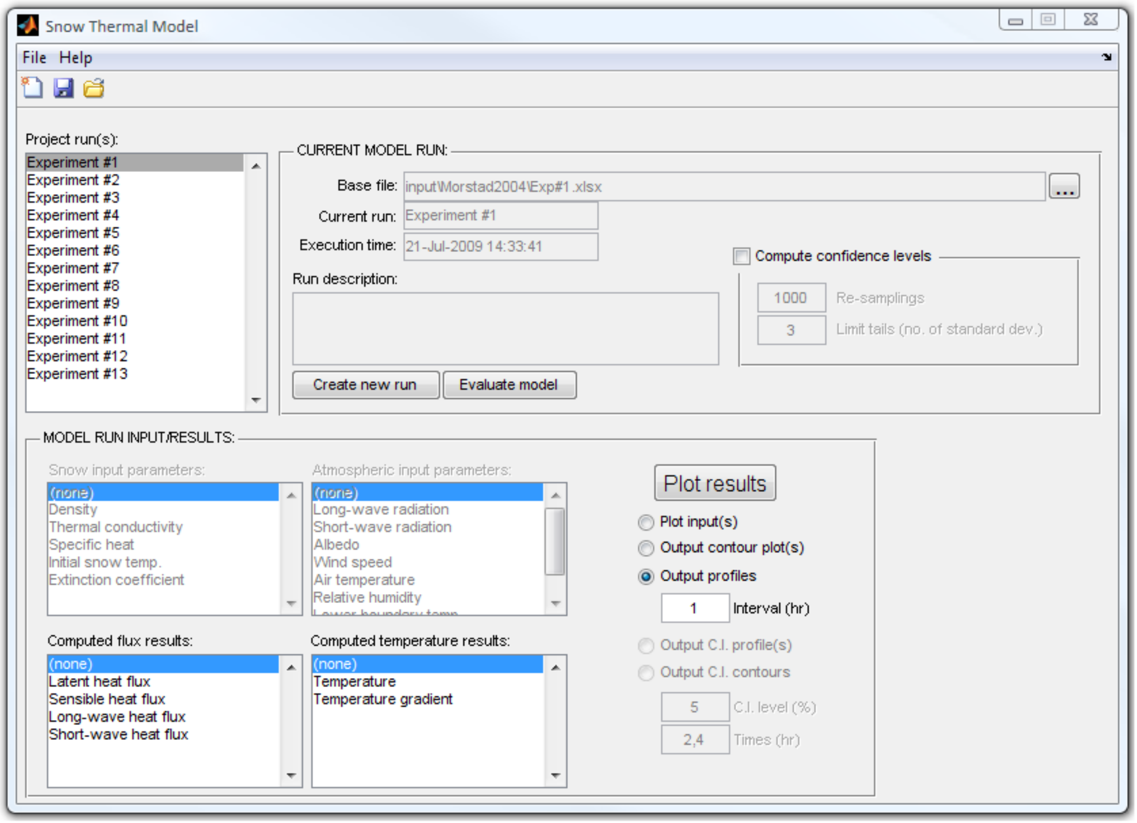



Figure 6: Graphical user interface for implementing the snow thermal model.

## 4.1 Performing Model Runs

The following briefly describes the basic steps of performing thermal model evaluations:

1. Select New Project from the file menu.

2. A prompt will appear that gives options regarding the Excel spreadsheet file to utilize. Selecting New copies the template.xlsx file previously discussed to a file selected by the user. Selecting Existing allows the user to select a previously created file. In both cases the Excel file will open when the necessary actions are complete.

3. Modify and saved the Excel file created for the desired conditions, as detailed in Section 3.
4. Return to the GUI application and type a name for the current run as well as a description. Also, if confidence levels are desired (see Section 3.2.3) the Compute Confidence Levels option should be checked at this time. The computation of the confidence levels can be extremely time consuming, so begin with a small number of re-samplings.
5. Press the Evaluate Model button on the GUI, this starts the model evaluations which may take several minutes depending on the computer and model inputs. If confidence levels are being computed a window will appear showing the progress of the calculations.
6. When the run is complete it appears in the Project Run(s) menu on the left-side of the GUI.
7. Additional runs may be computed by selecting the Create New Run button and the Excel file may be changed by selecting the "..." button at the right-end of the Base File text. This same button will also open the associated Excel file when a model run is activated. It is not necessary to create a new Excel file, but any changes made to the Excel file for additional model evaluations must be saved, these changes will not be stored and cannot not be recalled (this functionality may be available in future versions). Run names may be edited or runs may be deleted by right-clicking on the run in the Project Run(s) list.
8. After all the desired runs are completed the project should be saved by selecting Save Project from the File menu.

## 4.2 Graphing Results

It is possible to create graphs of both the model inputs and outputs, this is done using the lower pane of the GUI shown in Figure 6. First, a model run must be selected in the Project Run(s) panel. The graphs created share all of the functionallity of the graphs presented in Appendix **??** (Section **??**).

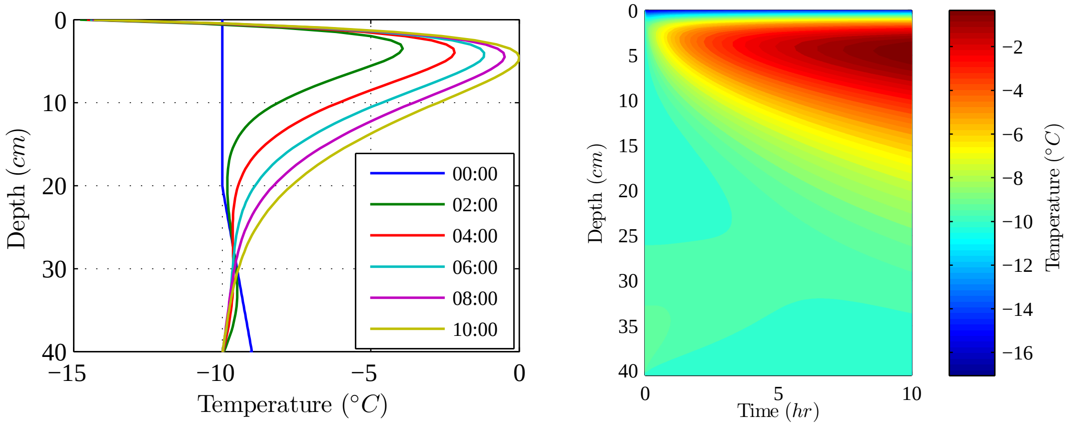
### 4.2.1 Model Inputs

The model inputs are graphed by selecting the Plot Input(s) radio button, this will cause the Snow and Atmospheric parameter lists to become activated. To create a graph simply select the desired item and press the Plot Results button. A graph will appear for each item selected.

### 4.2.2 Model Outputs

Two different graph styles of model outputs are available: profiles and coutours. Figure 7 provides examples of the snowpack temperature graphed with each of the different methods. When ploting profiles the interval, in hours, must also be set (e.g., 2 results in profiles being plotted every 2 hours). It is possible to graph a single profile directly from a contour plot, this is done be right-clicking on the contour where the profile is desired and the selecting either a vertical or horizontal profile.

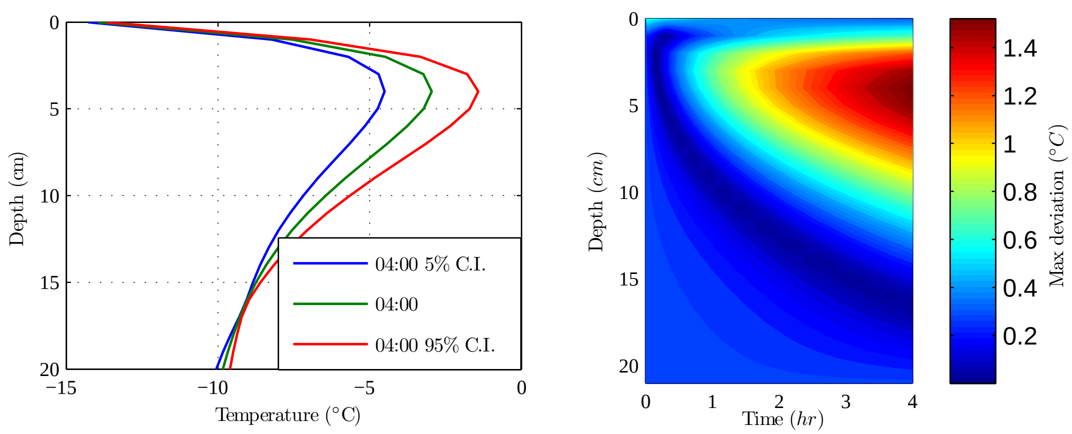
In similar fashion, if confidence intervals were computed it is possible to graph these intervals using the Output C.I. Profiles or Contours radio buttons. Figure 8 provides examples of temperature data plots with confidence level intervals. Both the profiles and the contours require the confidence level to be specified by a scalar value (in percent) entered into the C.I. Level location. When profiles are plotted the time(s) at which the profiles are desired must be specified in the Times (hr) location




(a) Temperature Profiles (b) Temperature Contours

Figure 7: Example graphs of snowpack temperatures demonstrated the two graphing options available: (a) profiles and (b) contours.

(e.g., 2 or 2, 4). The confidence level contour graphs show the absolute value of the largest deviation from the mean value.




(a) C.I. Profiles (b) C.I. Contours

Figure 8: Example graphs of snowpack temperature demonstrated the two graphing options available for displaying confidence level intervals: (a) C.I. profiles and (b) C.I. contours.

# 5 Closing Remarks

The information presented here explains the basic and advanced functionality of the thermal model developed. The details presented as well as the entire software package was developed to make the model easily accessible, thus please contact the author if more information is required.

# 6 Source Code

## 6.1 runmodel.m

```
function data = runmodel(varargin)
% RUNMODEL program to exceucte thermal model using Excel input file.
%-----------------------------------------------------------------------
% SYNTAX:
%   data = runmodel;
%   data = runmodel(filename);
%   data = runmodel(filename,name);
%   data = runmodel(filename,name,desc);
%   data = runmodel(...,[B,N]);
%
% DESCRIPTION:
%   data = runmodel executes the thermal model via Excel input, prompting
%       the user for a filename.
%   data = runmodel(filename) executes the thermal model for supplied name.
%   data = runmodel(filename,name) same as above, but allows the user to
%       name the run (e.g. name = 'Model Run #3');
%   data = runmodel(filename,name,desc) same as above, but allows user to
%       also add a description to the run (e.g. desc = 'This run mimics
%       Feb-14-2008 at the South station of the YC';)
%   data = runmodel(filename,name,desc,[B,N]) runs the model and
%       computes the bootstrap confidence intervals, where B = number
%       of resamplings, N = number of standard deviations to assume for
%       the tails
%
% OUTPUT:
% The data structure has the following fieldnames
%           xls: Input Excel filename
% bootsettings: Bootstrap settings
%          name: Name of current run
%          desc: Description of the current run
%          time: Start time of model execution
%             T: Array of snowpack temperatures
%             Q: Array of snowpack heat fluxes
%           snw: Input array of snow properties
%           atm: Input array of atmospheric conditions
%         const: Array of model constants
%         Tboot: Bootstrap replicates of temperature
%         Qboot: Bootstrap replicates of heat fluxes
%         Sboot: Bootstrap replicates of snw inputs
%         Aboot: Bootstrap replicates of atm inputs
%         Cboot: Bootstrap replicates of const inputs
%-----------------------------------------------------------------------

% 1 - GATHER OPTIONS
    data = getoptions(varargin{:});

% 2 - EXECUTE MODEL
    [S,A,data.const] = xls_input(data.xls);
    [data.snw,data.atm] = xls_prep(S,A,data.const);
    [data.T,data.Q] = thermal(data.snw,data.atm,data.const);

% 3 - RUN THE BOOSTRAP
    if ~isempty(data.bootsettings);
        B = data.bootsettings;
        bootdata = confint(data.xls,B(1),B(2));
        fn = fieldnames(bootdata);
        for i = 1:length(fn);
            data.(fn{i}) = bootdata.(fn{i});
        end
    end

%-----------------------------------------------------------------------
function [data,B] = getoptions(varargin)
% GETOPTIONS determines/sets the input options

% 1 - SET THE DEFAULTS
    filename = '';
    name = '';
    desc = '';
    B = [];

% 2 - GATHER BOOTSTRAPPING DATA
    idx = [];
    for i = 1:nargin; idx(i) = isnumeric(varargin{i}); end
    ix = find(idx,1,'first');
    if ~isempty(ix); B = varargin{ix}; end

% 3 - GATHER FILENAME, NAME, AND DESCRIPTION..
    if isempty(B); rem = varargin; else rem = varargin(1:nargin-1); end
    if length(rem) >= 1; filename = varargin{1}; end
    if length(rem) >= 2; name = varargin{2}; end
    if length(rem) == 3; desc = varargin{3}; end

% 4 - PROMPT FOR FILENAME
```

```
    % 4.1 - Gather/define the "lastdir" preference
        if ispref('ThermalModel_v5','lastdir');
            defdir = getpref('ThermalModel_v5','lastdir');
        else
            addpref('ThermalModel_v5','lastdir',cd);
            defdir = cd;
        end

    % 4.2 - Prompt the user for a filename
        if isempty(filename) ;
            FilterSpec = {'*.xlsx','Excel␣Workbook␣(*.xlsx)';...
                '*.xls','Excel␣97-2003␣Workbook␣(*.xls)';...
                '*.*','All␣files␣(*.*)'};
            [fn,pth] = uigetfile(FilterSpec,'Select␣file...',defdir);
            if isnumeric(fn); return; end
            filename = [pth,fn];
            setpref('ThermalModel_v5','lastdir',fileparts(filename));
        end

% 5 - BUILD DATA STRUCTURE
    % 5.1 - File information
        data.xls = filename;
        data.bootsettings = B;
        data.name = name;
        data.desc = desc;
        data.time = datestr(now);

    % 5.2 - Model evaluation
        data.T = []; data.Q = [];
        data.snw = []; data.atm = []; data.const = [];

    % 5.3 - Bootstrap results
        data.Tboot = []; data.Qboot = [];
        data.Sboot = []; data.Aboot = []; data.Cboot = [];
```

## 6.2 xls_input.m

```
function [S,A,C,E] = xls_input(filename)
% XLS_INPUT builds input matrics for usage with the thermal model (v5).
%-------------------------------------------------------------------------
% SYNTAX:
%   [S,A,C,E] = xls_input(filename)
%-------------------------------------------------------------------------

% 1 - CHECK FILE
    if nargin == 0; filename = 'input/WetSnow/base.xlsx'; end
    if ~exist(filename,'file');
        errordlg('File␣does␣not␣exist!'); return;
    end;

% 2 - EXTRACT DATA FROM FILE
    % 2.1 - Read files
        [S,snwTXT] = xlsread(filename,'SnowProperties');
        [A,atmTXT] = xlsread(filename,'AtmosphericSettings');
        [const] = xlsread(filename,'Constants');

    % 2.2 - Seperate constants and multipliers
        C = const(1:10);
        M = const(11:length(const),1); M(isnan(M)) = 0;
        aM = M(1:10); % atmospheric multipliers
        sM = M(11:length(M)); % snow multipliers

    % 2.3 - Seperate percent error values
        Nc = size(const,2);
        if Nc == 1;
            E.atm = zeros(length(aM),1);  E.atm(:,1) = 0.05;
            E.snow = zeros(length(sM),1); E.snw(:,1) = 0.05;
            E.const = zeros(10,1);
        else
            E.atm = const(11:length(aM)+10,2)/100;
            E.snow = const(length(aM)+11:length(const),2)/100;
            E.const = const(1:10,2)/100;
        end

 % 3 - APPLY SPECIAL VALUES TO ALBEDO AND EXTICTION COLUMNS
        A = albedo(A,atmTXT,S);
        S = extinction(S,snwTXT);
        S = density(S,snwTXT);

 % 4 - APPLY MULTIPLERS
    % 4.1 - Re-size multipliers arrays to necessary size
        aM = [1;aM(1:size(A,2)-1)]; % 1 adds a column for time
        sM = [1;sM(1:size(S,2)-1)]; % 1 adds a column for the depth

    % 4.2 - Apply multipliers
        for i = 1:length(sM); S(:,i) = S(:,i) * sM(i); end
```

```
        for i = 1:length(aM); A(:,i) = A(:,i) * aM(i); end

%--------------------------------------------------------------------------
function A = albedo(A,atmTXT,S)
% ALBEDO applies special input into albedo column: dXX, classX, <type>
% Special values given in the albedo column (#4) assume that the shortwave
% column (3) is an all-wave value, so it is divided into a VIS/NIR
% components as is the albedo for all "special" cases

% 1 - Determine "special" locations
    idx = find(isnan(A(:,4)));

% 2 - Cycle through each special value and compute desired albedos
for i = 1:length(idx);
    val = atmTXT{idx(i)+3,4}; % Current special case

    % Optical depth case: dXX
    if strcmpi('d',val(1)); % Optical depth caer
        dopt = str2double(val(2:length(val)));
        if isnan(dopt);
            error('xls_input:albedo','optical␣depth␣ill␣define.');
        end
        [A(idx(i),4),b1,A(idx(i),11)] = rad_calc(dopt,S(1,2));

    % Class case: classX
    elseif length(val) > 5 && strcmpi('class',val(1:5));
        cls = str2double(val(6:length(val)));
        if isnan(cls);
            error('xls_input:albedo','class␣ill␣define.');
        end
        [A(idx(i),4),b1,A(idx(i),11)] = rad_calc('class',cls);

    % Cuvre case: 'fine','medium','coarse'
    elseif sum(strcmpi(val,{'fine','medium','coarse'})) == 1;
        [A(idx(i),4),A(idx(i),11)] = rad_calc(val);

    % Record an error
    else
        error('xls_input:albedo','error␣with␣albedo␣input,␣colum␣4!');
    end

    % Redifine all-wave shortwave to VIS/NIR components
        [A(idx(i),3),A(idx(i),10)] = rad_calc(A(idx(i),3));
end

%--------------------------------------------------------------------------
function S = extinction(S,snwTXT)
% EXTINCTION applies special input for extinction column: dXX or classX
% Special values given in the extection column (#6) overwrite VIS/NIR
% columns with the desired numeric value

% 1 - Determine "special" locations
    if size(S,2) == 5; S(:,6) = NaN(size(S,1),1); end
    idx = find(isnan(S(:,6)));

% 2 - Cycle through each special value and compute desired albedos
for i = 1:length(idx);
    val = snwTXT{idx(i)+3,6}; % Current special case

    % Optical depth case: dXX
    if strcmpi('d',val(1)); % Optical depth caer
        dopt = str2double(val(2:length(val)));
        if isnan(dopt);
            error('xls_input:extinction','optical␣depth␣ill␣define.');
        end
        [a1,S(idx(i),6),a2,S(idx(i),7)] = rad_calc(dopt,S(1,2));

    % Class case: classX
    elseif length(val) > 5 && strcmpi('class',val(1:5));
        cls = str2double(val(6:length(val)));
        if isnan(cls);
            error('xls_input:extinction','class␣ill␣define.');
        end
        [a1,S(idx(i),6),a2,S(idx(i),7)] = rad_calc('class',cls);

    % Record an error
    else
        error('xls_input:albedo','error␣with␣albedo␣input,␣colum␣4!');
    end
end

%--------------------------------------------------------------------------
function S = density(S,snwTXT)
% DENSITY applies special input for density and/or thermal conductivity
% columns, either can be 'auto', just not both. The auto values are
% replaced it the appopriate value from Sturm, 1997. The quadratic is used
% for solving for k and the exponential when solving for density

for i = 1:size(S,1);
    rho = S(i,2)/1000; k = S(i,3);
    if isnumeric(rho) && isnumeric(k); % Both specified
        S(i,2) = rho*1000; S(i,3) = k;
```

```
    elseif isnan(rho) && isnumeric(k); % Compute density
        S(i,2) = (log10(k) + 1.652) / 2.65 * 1000;
    elseif isnumeric(rho) && isnan(k); % Compute k
        if rho < 0.156;
            S(i,3) = 0.023 + 0.234 * rho;
        else
            S(i,3) = 0.138 - 1.01*rho + 3.233 * rho^2;
        end
    else
        error('xls_input:density',...
            'error with density/conductivity input, column 2 and/or 3!');
    end
end
```

## 6.3 xls_prep.m

```
function [s,a] = xls_prep(snow,atm,constants)
% XLS_PREP builds arrays for inputing into thermal model
%-------------------------------------------------------------------------
% SYNTAX:
%   [snow,atm] = prep_input(snow,atm,constants);
%
% INPUT:
%   snow      = matrix containing snow data
%   atm       = matrix containing atmospheric data
%   constants = matrix containing model constants
%
% EXAMPLE INPUT:
%   snow     = [50,130,0.06,2030,-10,70];
%   atm      = [6,240,500,0.82,1.7,-10,.2,-10,101];
%   contants = [2833,0.0023,0.0023,0.622,0.462,-5,0.402,0.95,1,60,1];
%-------------------------------------------------------------------------

% 1 - Fill in atmospheric data
    atm(:,1) = atm(:,1) .* 3600; % Convert time to seconds
    dt = constants(10);          % Time step in seconds
    a  = fill_array(atm,dt);

% 2 - Fill and snow properties data
    dz = constants(9);
    s = fill_array(snow,dz);

%-------------------------------------------------------------------------
% SUBFUCTION: fill_array
function out = fill_array(in,int)
% FILL_ARRAY builds an array from "in" using the interval in "int" based on
% the first column of data

% 1 - Build array for case when data is only a single row (constant data)
    len = size(in,1);
    if len == 1;
        in(2,:) = in(1,:);
        in(1,1) = 0;
    end

% 2 - Build new array with spacing based on "int"
    % 2.1 - Build the first column of the new array (e.g. time steps)
        n = size(in,1);
        xi = (in(1,1):int:in(n,1))';

    % 2.2 - Interpolate the remaining data based on the first column
        x = in(:,1);
        Y = in(:,2:size(in,2));
        yi = interp1(x,Y,xi,'linear');
        out = [xi,yi];
```

## 6.4 thermal.m

```
function [T,Q] = thermal(snow,atm,C)
% THERMAL executes 1-D heat equation based thermal model
%-------------------------------------------------------------------------
% SYNTAX:
%   [T,Q] = thermal(snow,atm,C)
%
% DESCRIPTION:
%   [T,Q] = thermal(snow,atm,C) based on the information provided in the
%   numeric arrays containing snow properties (snow), atmospheric
%   conditions (atm), and model constants (C) a 1-D thermal analysis is
%   performed resulting in the snowpack temperatures (T) and associated
%   heat fluxes (Q)
%-------------------------------------------------------------------------
```

```

% 1 - PREPARE VARIABLES FOR CALCULATION
    % 1.1 - Pre-define arrays
        if size(atm,2) == 11 && size(snow,2) == 7;
            ndim = 2;
        else
            ndim = 1;
        end

        nt = size(atm,1);           % Number of time steps
        ns = size(snow,1);          % Number of snow elements

        T = zeros(ns,nt);           % Temperature array
        q = zeros(ns,nt,ndim);      % Short-wave flux absorbed array
        qs = zeros(nt,3);           % Surface flux array

        A = zeros(ns+1,ns+1);       % A-matrix for temperature solution
        b = zeros(ns+1,1);          % b-vector for temperature solution

    % 1.2 - Establish user specified constants
        Ls = C(1);
        Ke = C(2);
        Kh = C(3);
        MvMa = C(4);
        Rv = C(5);
        T0 = C(6) + 273.15;
        e0 = C(7);
        emis = C(8);
        dz = C(9)/100;
        dt = C(10);

    % 1.3 - Define additional constants needed
        sb = 5.6696*10^(-8);        % StefanBoltzmann constant (W/m^2/K^4)
        R = 0.287;                  % Gas constant for air (kJ/kg/K)

    % 1.4 - Compute the properties of air
        Cp_air = 1003;              % Specific heat @-5C (J/kg/K)
        rho_air = atm(:,9)./(R*(atm(:,6) + 273.15)); % Density (kg/m^2)

 % 2 - INITILIZE ARRAYS FOR COMPUTATION
    % 2.1 - Initilize temperature array
        T(:,1) = snow(:,5);   % Initial snow temperature
        T(ns+1,:) = atm(:,8); % Base

    % 2.2 - General Matrix coefficients
        Ca = snow(:,3) ./ dz^2;                     % a
        Cb = (snow(:,2) .* snow(:,4))./dt;          % b
        Cc = Cb + Ca;                               % c
        Cd = Cb - Ca;                               % d

% 3 - BEGIN COMPUTING FOR EACH TIME STEP (time step = index "j")
for j = 2:nt
    % 3.1 - Establish air/snow surface temperatures
        Ta = atm(j,6) + 273.15;
        Ts = T(1,j-1) + 273.15;

    % 3.2 - Compute longwave heat flux
        qs(j,1) = atm(j,2) - emis*sb*Ts^4;

    % 3.3 - Compute the latent heat flux
        ea = e0*exp(Ls/Rv *(1/T0 - 1/Ta))*atm(j,7)/100;
        es = e0*exp(Ls/Rv *(1/T0 - 1/Ts));
        qs(j,2) = 1000*MvMa*rho_air(j)*Ls*Ke*atm(j,5)*(ea-es)/atm(j,9);

    % 3.4 - Compute the sensible heat flux
        qs(j,3) = Kh*rho_air(j)*Cp_air*atm(j,5)*(Ta - Ts);

    % 3.5 - Compute the absorbed shortwave and build solution matrix
    %       for each layer of snow
        % 3.5.1 - Compute shortwave absorbed in the top layer
            q(1,j,1) = atm(j,3)*(1-atm(j,4))*(1-exp(-snow(1,6)*dz));

        % 3.5.2 - Compute shortave in NIR if present
            if ndim == 2;
                q(1,j,2) = atm(j,10)*(1-atm(j,11))*(1-exp(-snow(1,7)*dz));
            end

        % 3.5.2 - Compute shortwave absorbed for lower layers and build
        %         solution matrices
        for i = 2:ns
            % Short-wave radiation absorbed
            q(i,j,1) = q(i-1,j,1)*exp(-snow(i,6)*dz); % all-wave or VIS
            if ndim == 2;
                q(i,j,2) = q(i-1,j,2)*exp(-snow(i,7)*dz); % NIR
            end

            % Solution matrices
            A(i,i-1) = -Ca(i)/2;
            A(i,i)   = Cc(i);
            A(i,i+1) = -Ca(i)/2;
            b(i,1) = Ca(i)/2*T(i-1,j-1) + Cd(i)*T(i,j-1) + ...
                        Ca(i)/2*T(i+1,j-1) + sum(q(i,j,:))/dz;
```

```
        end

    % 3.6 - Compute the surface flux
        sur_flux = sum(qs(j,1:3));

    % 3.7 - Insert matrix values for surface node (i = 1)
        A(1,1) = Cc(1);
        A(1,2) = -Ca(1);
        b(1) = Cd(1)*T(1,j-1) + Ca(1)*T(2,j-1) + 2*sur_flux/dz + ...
            sum(q(1,j,:))/dz;

    % 3.8 - Insert matrix values for bottom boundary condition
        A(ns+1,ns+1) = 1;
        b(ns+1) = atm(j,8);

    % 3.9 - Calculate the new temperature profile
        Tnew = A\b;
        Tnew(Tnew>0) = 0;
        T(:,j) = Tnew;
end;

Q = zeros(ns,nt,ndim+3);
Q(1,:,1:3) = qs;
Q(:,:,4:end) = q;
```

## 6.5 rad_calc.m

```
function varargout = rad_calc(varargin)
% RAD_CALC spectral calculations of radiation, albedo, and extiction.
%----------------------------------------------------------------------
% SYNTAX:
%    [SWvis,SWnir,SWswir] = rad_calc(SWall);
%    [Avis,Anir,Aswir] = rad_calc(curve);
%    [Avis,Bvis,Anir,Bvis,Aswir,Bswir] = rad_calc(dopt,rho);
%    [Avis,Bvis,Anir,Bvis,Aswir,Bswir] = rad_calc('class',num);
%
% DESCRIPTION:
%    [SWvis,SWnir,SWswir] = rad_calc(SWall) computes spectral components of
%        all-wave shortwave radiation based on ASTM standard.
%    [Avis,Anir,Aswir] = rad_calc(curve) computes spectral albedo components
%        based on curves: 'fine','medium','coarse'
%    [Avis,Bvis,Anir,Bvis,Aswir,Bswir] = rad_calc(dopt,rho) computes
%        spectral components of albedo and extiction based on Snow & Climate
%        equations given on p.56.
%    [Avis,Bvis,Anir,Bvis,Aswir,Bswir] = rad_calc('class',num) computes
%        spectral components of albedo and extiction based on Snow & Climate
%        table given on p.57, where num must be an integer between 1 and 6.
%----------------------------------------------------------------------

% 1 - Compute desired values, execute as order in SYNTAX/DESCRIPTION above
    if nargin == 1 && isnumeric(varargin{1});
        output = shortwave(varargin{1});
    elseif nargin == 1 && ischar(varargin{1});
        output = albedo_curve(varargin{1});
    elseif nargin == 2 && isnumeric(varargin{1});
        output = albedo_eqn(varargin{:});
    elseif nargin == 2 && ischar(varargin{1});
        output = albedo_table(varargin{2});
    end

% 2 - Produce output
    varargout = num2cell(output);

%-----------------------------------------------------------------------
function out = albedo_table(N)
% ALBEDO_TABLE computes albedo and exciction base on Snow&Climate(p.57)

% Error handling
    if N < 1 || N > 6;
        error('Class must be an interger 1 through 6!'); out = NaN; return;
    end

% Build Table 2.6 from Snow & Climate (2008), p.57
    C(:,1) = [94,94,93,93,92,91]/100;
    C(1:6,2) = 40;
    C(:,3) = [80,73,68,64,57,42]/100;
    C(:,4) = [110,136,190,110,112,127];
    C(:,5) = [59,49,42,37,30,18]/100;
    C(1:6,6) = inf;

 % Produce output
    out = C(N,:);

%-----------------------------------------------------------------------
function out = albedo_eqn(dopt,rho)
% ALBEDO_EQN computes albedo and exciction base on Snow&Climate(p.56)
```

```
% Convert units (dopt mm->m; rho kg/m^3->gm/cm^3)
    dopt = dopt/1000; rho = rho/1000;

% VIS
    out(1) = min(0.94,0.96 - 1.58*sqrt(dopt));
    out(2) = max(0.04, 0.0192*rho/sqrt(dopt))*100;

% NIR
    out(3) = 0.95 - 15.4 * sqrt(dopt);
    out(4) = max(1, 0.1098*rho/sqrt(dopt))*100;

% SWIR
    out(5) = 0.88 + 346.6*dopt - 32.31*sqrt(dopt);
    out(6) = inf;

%--------------------------------------------------------------------------
function Aout = albedo_curve(curve)
% ALBEDO_CURVE computes VIS,NIR,& SWIR albedos based on input curve

% 1 - Load the desired curve
    X = load('albedo.mat');
    A = X.(curve);

% 2 - Parse out the albedo for each wavelength group
    L = [285,800; 800,1500; 1500,3500];
    for i = 1:size(L,1);
        idx(1) = find(A(:,1)>=L(i,1),1,'first');
        idx(2) = find(A(:,1)<=L(i,2),1,'last');
        Aout(i) = mean(A(idx(1):idx(2),2))/100;
    end

%--------------------------------------------------------------------------
function SWout = shortwave(SWall)
% SHORTWAVE computes spectral components of all-wave based on ASTM standard

% 1 - Load the solar spectrum desired
    X = load('albedo.mat');
    S = X.astm;

% 2 - Normalize solar spectrum to inputed SW data
    I = insolation(S,[285,3500]);
    S(:,2) = (S(:,2)/I)*SWall;

% 3 - Parse out wavelength groups
    L = [285,800; 800,1500; 1500,3500];
    for i = 1:size(L,1); SWout(i) = insolation(S,L(i,:)); end

%--------------------------------------------------------------------------
function I = insolation(S,L)
% POWER computers the insolation between the a and b wavelenghts

% 1 - Locate indicies of wavelenghts
    x = S(:,1); y = S(:,2);
    i(1) = find(x>L(1),1,'first');
    i(2) = find(x<L(2),1,'last');
    idx = i(1):i(2);

% 2 - Compute the insolation
    I = sum((y(i(1):i(2)-1)+diff(y(idx))).*diff(x(idx)));
```

## 6.6 confint.m

```
function data = confint(filename,B,n)
% CONFINT computes the confidence intervals for temp profiles

% Read file input
    [S,A,C,E] = xls_input(filename);

% Compute the actual temperature profile
    [Sa,Aa] = xls_prep(S,A,C);
    T = thermal(Sa,Aa,C);

% Compute the standard deviation values
    s = getstd(S,E.snow,n,1); % standard devaition for snow properties
    a = getstd(A,E.atm,n,1); % standard devaition for atmospheric terms
    c = getstd(C,E.const,n,0); % standard deviation for constants

% Compute the Monte Carlo replicates
    data.Tboot = zeros([size(T),B]); % Initilize storage array
    h = waitbar(0,'Please wait...');
    for i = 1:B;
        r = rand(1);
        S_b = norminv(r,S,abs(s)); % Re-sample snow
            S_b(:,1) = S(:,1);      % Snow depth does not change
            S_b(isnan(S_b)) = S(isnan(S_b));
```

```
        A_b = norminv(r,A,abs(a)); % Re-sample atmosphere
            A_b(:,1) = A(:,1);      % Duration does not change
            A_b(isnan(S_b)) = A(isnan(A_b));
        C_b = norminv(r,C,abs(c)); % Constant resampling
            C_b(9) = C(9);          % dz constant
            C_b(10) = C(10);        % dt constant
            C_b(isnan(C_b)) = C(isnan(C_b));

        [SS,AA] = xls_prep(S_b,A_b,C_b); % Build input for evaluation
        [data.Tboot(:,:,i), data.Qboot(:,:,:,i)] = thermal(SS,AA,C_b);
        data.Sboot(:,:,i) = SS;
        data.Aboot(:,:,i) = AA;
        data.Cboot(:,:,i) = C_b;
        waitbar(i/B,h);
    end
    close(h);

%--------------------------------------------------------------------------
function s = getstd(S,E,n,offset)
% GETSTD returns the standard deviation of the input items
    if offset == 1;
        s(:,1) = S(:,1);
        for i = 2:size(S,2);
            s(:,i) = S(:,i).*E(i-offset)/n;
        end
    elseif offset == 0;
        for i = 1:size(S,2);
            s(:,i) = S(:,i).*E(i)/n;
        end
    end
```