

Thermal Model Software

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

March 24, 2011

Contents

1	Introduction	1
2	Basic Application	1
3	Spreadsheet Application	3
3.1	General Application	3
3.2	Additional Features	3
3.2.1	Time Dependant Snow Properties	3
3.2.2	Snow Micro-Structure	3
3.2.3	Input Multipliers	5
3.2.4	Confidence Intervals	6
4	Graphical User Interface	6
4.1	Performing Model Runs	7
4.2	Graphing Results	8
4.2.1	Model Inputs	8
4.2.2	Model Outputs	8
5	Closing Remarks	10
6	References	10
7	Source Code	11
7.1	runmodel.m	11
7.2	xls_input.m	12
7.3	xls_prep.m	14
7.4	thermal.m	15
7.5	rad_calc.m	16
7.6	confint.m	18

List of Tables

List of Figures

1	Flow chart demonstrating how the various functions discussed in Section 2 and 3 interact.	1
2	Example of the (a) “SnowProperties” and (b) “AtmosphericSettings” worksheets for Excel file read by <code>xls_input.m</code>	4
3	Example of the “Constants” worksheets for Excel file read by <code>xls_input.m</code>	5
4	MATLAB implementation of <code>runmodel.m</code> and the resulting data structure.	6
5	Required data structure of <code>albedo.mat</code>	6
6	Graphical user interface for implementing the snow thermal model.	7
7	Example graphs of snowpack temperatures demonstrated the two graphing options available: (a) profiles and (b) contours.	9
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.	9

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.
- MATLAB code is also presented in-line with the text as:

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

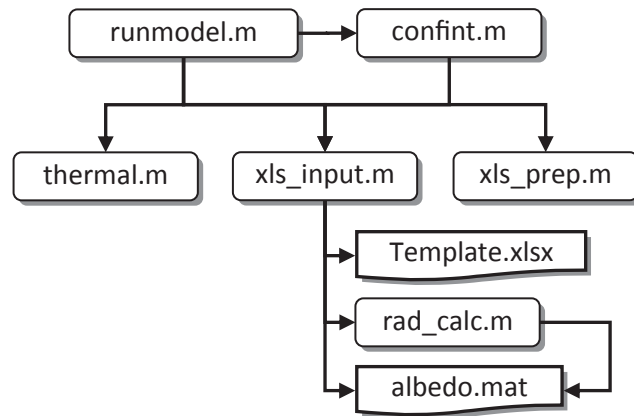


Figure 1: Flow chart demonstrating how the various functions discussed in Section 2 and 3 interact.

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 7). 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 ($\text{W}/(\text{m}\cdot\text{K})$), specific heat capacity ($\text{J}/(\text{gm}\cdot\text{K})$), snow temperature ($^{\circ}\text{C}$), and extinction coefficient ($1/\text{m}$). 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 ($^{\circ}\text{C}$), relative humidity (%), the lower boundary condition ($^{\circ}\text{C}$), and air pressure (kPa). Two additional columns may also be defined that specify 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 ($\text{kJ}/(\text{kg}\cdot\text{K})$), reference temperature ($^{\circ}\text{C}$), 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];
>> constants = [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 a finite-difference solutio that 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 Time Dependant Snow Properties

...

3.2.2 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 Armstrong and Brun (2008, Eq. 2.25, p. 56), where 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 Armstrong and Brun (2008, Tab. 2.6), where `X` is a value one to six (e.g., “class2”). The `type` may be one of three

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

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

Microsoft Excel - template.xlsx

template.xlsx - Microsoft Excel

Home Insert Page Layout Formulas Data Review View Developer Add-Ins Acrobat

Clipboard Font Alignment Number Conditional Formatting Styles Cell Styles Cells Sort & Find & Filter Select Editing

F7

1 Snow Property Settings:						
2 Depth	Density (ρ)	Thermal Conductivity (k)	Specific Heat (C_p)	Snow Temperature ($T_{initial}$)	Extinction Coefficient (κ)	NIR Extinction Coefficient (κ)
3 (cm)	(kg/m^3)	($W/m \cdot K$)	($J/g \cdot K$)	($^{\circ}C$)	($1/m$)	($1/m$)
4 0	130	0.06325	2036	-10	class2	
5 20	130	0.06325	2036	-10	class2	
6 40	150	0.05	2036	-9	class2	
7						
8						
9						

SnowProperties AtmosphericSettings Constants

(a) Snow Properties

Microsoft Excel - template.xlsx

template.xlsx - Microsoft Excel

Home Insert Page Layout Formulas Data Review View Developer Add-Ins Acrobat

Clipboard Font Alignment Number Conditional Formatting Styles Cell Styles Cells Sort & Find & Filter Select Editing

F8

1 Atmospheric Settings:										
2 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)
3 (hours)	(W/m^2)	(W/m^2)		(cm)	($^{\circ}C$)	(%)	($^{\circ}C$)	(kPa)	(W/m^2)	
4 10	185	500	d0.5	1.3	-10	20	-10	100		
5										
6										
7										
8										

SnowProperties AtmosphericSettings Constants

(b) Atmospheric Settings

Figure 2: Example of the (a) “SnowProperties” and (b) “AtmosphericSettings” worksheets for Excel file read by `xls.input.m`.

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 [Baldridge et al. \(2009\)](#). The albedo and extinction coefficient calculations are preformed in the `albedo` and `extinction` sub-functions of `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 [ASTM G-173 \(2003\)](#) the appropriated values are computed based on this spectrum via `rad_calc.m` (see Section 7.5).

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 first column of which provides the wavelength in nanometers. The second column of `x.atism` contains solar irradiance as defined by [ASTM G-173 \(2003\)](#). For the other items (e.g., `x.fine`), the second column contains albedo values as defined in [Baldrige et al. \(2009\)](#).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 [Sturm et al. \(1997\)](#).

3.2.3 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

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

MATLAB Code	Description	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
Snow Property Multipliers & Error	NIR Incoming Shortwave (SWin)	1	5
	NIR Albedo (alpha)	1	5
	Density (p)	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`.


```

1 >> filename = 'template.xlsx';
2 >> data = runmodel(filename)
3 data =
4         xls: 'template.xlsx'
5         bootsettings: []
6         name: ''
7         desc: ''
8         time: '22-Mar-2010_09:58:06'
9         T: [82x601 double]
10        Q: [81x601x5 double]
11        snw: [81x7 double]
12        atm: [601x11 double]
13        const: [1x10 double]
14        Tboot: []
15        Qboot: []
16        Sboot: []
17        Aboot: []
18        Cboot: []
19 >> data(2) = runmodel(filename); % multiple runs may be stored
20 >>

```

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

```

1 >> x = load('albedo.mat')
2 x =
3         astm: [2002x2 double]
4         fine: [179x2 double]
5         medium: [179x2 double]
6         coarse: [179x2 double]
7 >>

```

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

multiplied by the values listed, allowing the user to quickly modify the various inputs.

3.2.4 Confidence Intervals

The `runmodel.m` function includes the ability to compute confidence intervals via `confint.m`, which uses the percentile bootstrap method presented by Efron and Tibshirani (1993). 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.

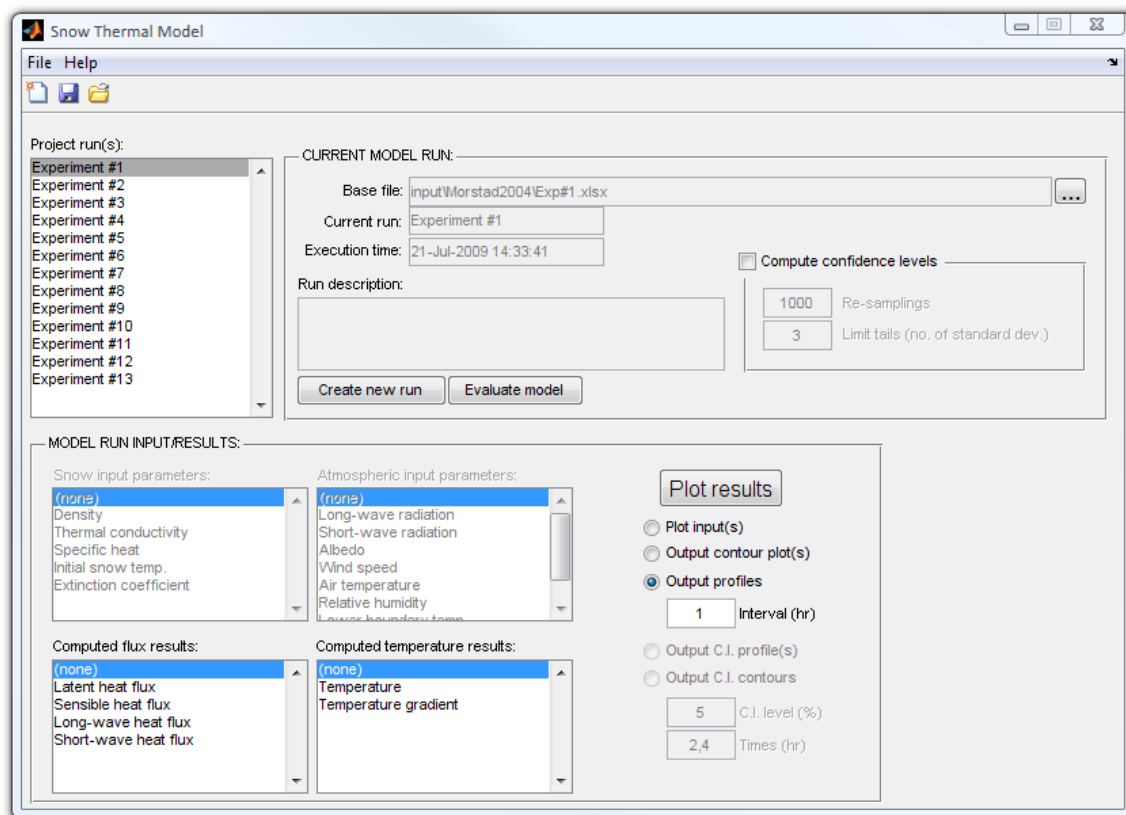


Figure 6: Graphical user interface for implementing the snow thermal model.

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 3.2.4) 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.

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 contours. Figure 7 provides examples of the snowpack temperature graphed with each of the different methods. When plotting 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 by 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

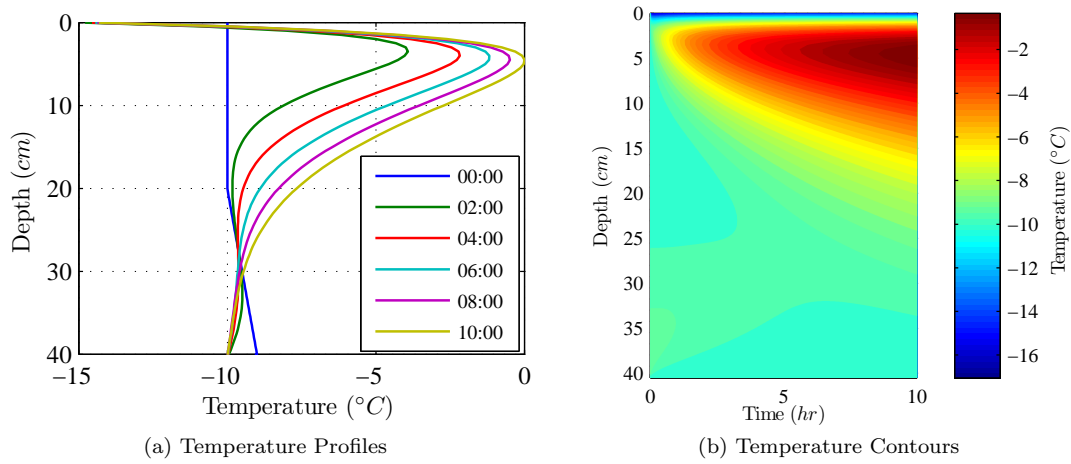


Figure 7: Example graphs of snowpack temperatures demonstrated the two graphing options available: (a) profiles and (b) contours.

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

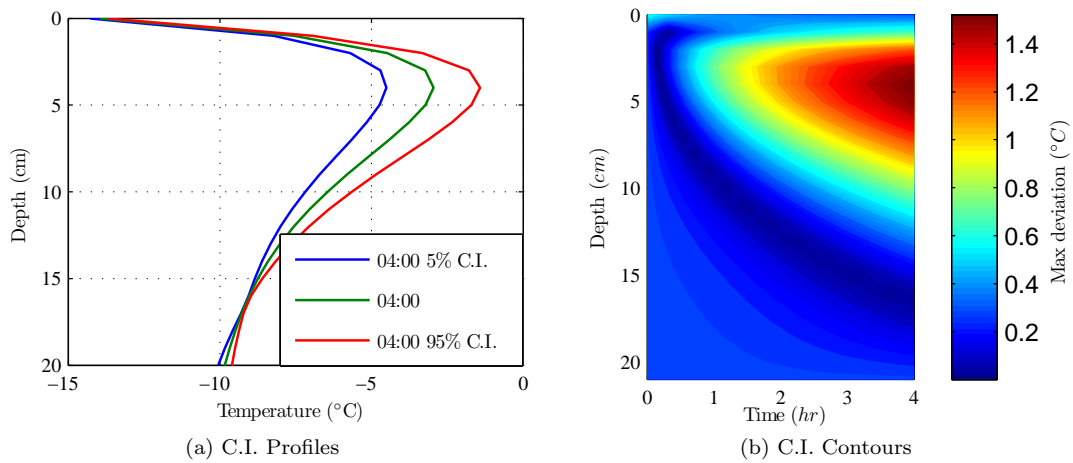


Figure 8: Example graphs of snowpack temperature demonstrated the two graphing options available for displaying confidence level intervals: (a) C.I. profiles and (b) C.I. contours.

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 References

- Armstrong, R. and E. Brun, 2008: *Snow and Climate: Physical Processes, Surface Energy Exchange, and Modeling*. Cambridge University Press.
- ASTM G-173, 2003: 14.04. Standard tables for reference solar spectral irradiances: Direct normal and hemispherical on 37° tilted surface. ASTM International. West Conshohocken, PA.
- Baldrige, A., S. Hook, C. Grove, and G. Rivera, 2009: The ASTER spectral library version 2.0. *Remote Sensing of Environment*, **113** (4), 711–715.
- Efron, B. and R. J. Tibshirani, 1993: *An introduction to the Bootstrap*. Chapman and Hall.
- Sturm, M., J. Holmgren, and M. Knig, 1997: The thermal conductivity of seasonal snow. *Journal of Glaciology*, **43** (143), 26–41.

7 Source Code

7.1 runmodel.m

```

1 function data = runmodel(varargin)
2 % RUNMODEL program to execute thermal model using Excel input file.
3 % -----
4 % SYNTAX:
5 %   data = runmodel;
6 %   data = runmodel(filename);
7 %   data = runmodel(filename,name);
8 %   data = runmodel(filename,name,desc);
9 %   data = runmodel(...,[B,N]);
10 %
11 % DESCRIPTION:
12 %   data = runmodel executes the thermal model via Excel input, prompting
13 %   the user for a filename.
14 %   data = runmodel(filename) executes the thermal model for supplied name.
15 %   data = runmodel(filename,name) same as above, but allows the user to
16 %   name the run (e.g. name = 'Model Run #3');
17 %   data = runmodel(filename,name,desc) same as above, but allows user to
18 %   also add a description to the run (e.g. desc = 'This run mimics
19 %   Feb-14-2008 at the South station of the YC');
20 %   data = runmodel(filename,name,desc,[B,N]) runs the model and
21 %   computes the bootstrap confidence intervals, where B = number
22 %   of resamplings, N = number of standard deviations to assume for
23 %   the tails
24 %
25 % OUTPUT:
26 % The data structure has the following fieldnames
27 %   xls: Input Excel filename
28 %   bootsettings: Bootstrap settings
29 %   name: Name of current run
30 %   desc: Description of the current run
31 %   time: Start time of model execution
32 %   T: Array of snowpack temperatures
33 %   Q: Array of snowpack heat fluxes
34 %   snw: Input array of snow properties
35 %   atm: Input array of atmospheric conditions
36 %   const: Array of model constants
37 %   Tboot: Bootstrap replicates of temperature
38 %   Qboot: Bootstrap replicates of heat fluxes
39 %   Sboot: Bootstrap replicates of snw inputs
40 %   Aboot: Bootstrap replicates of atm inputs
41 %   Cboot: Bootstrap replicates of const inputs
42 % -----
43 % 1 - GATHER OPTIONS
44 data = getoptions(varargin{:});
45
46 % 2 - EXECUTE MODEL
47 [S,A,data.const] = xls_input(data.xls);
48 [data.snw,data.atm] = xls_prep(S,A,data.const);
49 [data.T,data.Q] = thermal(data.snw,data.atm,data.const);
50
51 % 3 - RUN THE BOOTSTRAP
52 if ~isempty(data.bootsettings);
53     B = data.bootsettings;
54     bootdata = confint(data.xls,B(1),B(2));
55     fn = fieldnames(bootdata);
56     for i = 1:length(fn);
57         data.(fn{i}) = bootdata.(fn{i});
58     end
59 end
60
61 % -----
62 function [data,B] = getoptions(varargin)
63 % GETOPTIONS determines/sets the input options
64
65 % 1 - SET THE DEFAULTS
66 filename = '';
67 name = '';
68 desc = '';
69 B = [];
70
71 % 2 - GATHER BOOTSTRAPPING DATA
72 idx = [];
73 for i = 1:nargin; idx(i) = isnumeric(varargin{i}); end
74 ix = find(idx,1,'first');
75 if ~isempty(ix); B = varargin{ix}; end
76
77 % 3 - GATHER FILENAME, NAME, AND DESCRIPTION..
78 if isempty(B); rem = varargin; else rem = varargin(1:nargin-1); end
79 if length(rem) >= 1; filename = varargin{1}; end
80 if length(rem) >= 2; name = varargin{2}; end
81 if length(rem) == 3; desc = varargin{3}; end
82
83 % 4 - PROMPT FOR FILENAME

```

```

85 % 4.1 - Gather/define the "lastdir" preference
86 if ispref('ThermalModel_v5','lastdir');
87     defdir = getpref('ThermalModel_v5','lastdir');
88 else
89     addpref('ThermalModel_v5','lastdir',cd);
90     defdir = cd;
91 end
92
93 % 4.2 - Prompt the user for a filename
94 if isempty(filename) ;
95     FilterSpec = {'*.xlsx','Excel_Workbook_(*.xlsx)';...
96                 '*.xls','Excel_97-2003_Workbook_(*.xls)';...
97                 '*.*', 'All_files_(*.*)'};
98     [fn,pth] = uigetfile(FilterSpec,'Select_file...',defdir);
99     if isnumeric(fn); return; end
100    filename = [pth,fn];
101    setpref('ThermalModel_v5','lastdir',fileparts(filename));
102 end
103
104 % 5 - BUILD DATA STRUCTURE
105 % 5.1 - File information
106 data.xls = filename;
107 data.bootsettings = B;
108 data.name = name;
109 data.desc = desc;
110 data.time = datestr(now);
111
112 % 5.2 - Model evaluation
113 data.T = []; data.Q = [];
114 data.snw = []; data.atm = []; data.const = [];
115
116 % 5.3 - Bootstrap results
117 data.Tboot = []; data.Qboot = [];
118 data.Sboot = []; data.Aboot = []; data.Cboot = [];

```

7.2 xls_input.m

```

1 function [S,A,C,E] = xls_input(filename)
2 % XLS_INPUT builds input matrices for usage with the thermal model (v5).
3 %-----
4 % SYNTAX:
5 %   [S,A,C,E] = xls_input(filename)
6 %-----
7
8 % 1 - CHECK FILE
9 if nargin == 0; filename = 'template.xlsx'; end
10 if ~exist(filename,'file');
11     error('File_does_not_exist!'); return;
12 end;
13
14 % 2 - EXTRACT DATA FROM FILE
15 % 2.1 - Read files
16 [S,snwTXT] = xlsread(filename,'SnowProperties');
17 [A,atmTXT] = xlsread(filename,'AtmosphericSettings');
18 [const] = xlsread(filename,'Constants');
19
20 % 2.2 - Seperate constants and multipliers
21 C = const(1:10);
22 M = const(11:length(const),1); M(isnan(M)) = 0;
23 aM = M(1:10); % atmospheric multipliers
24 sM = M(11:length(M)); % snow multipliers
25
26 % 2.3 - Seperate percent error values
27 Nc = size(const,2);
28 if Nc == 1;
29     E.atm = zeros(length(aM),1); E.atm(:,1) = 0.05;
30     E.snow = zeros(length(sM),1); E.snow(:,1) = 0.05;
31     E.const = zeros(10,1);
32 else
33     E.atm = const(11:length(aM)+10,2)/100;
34     E.snow = const(length(aM)+11:length(const),2)/100;
35     E.const = const(1:10,2)/100;
36 end
37
38 % 3 - APPLY SPECIAL VALUES
39 % 3.1 - Compute the albedo based on snow type
40 A = albedo(A,atmTXT,S);
41
42 % 3.2 -Adjust snow properties
43 S = extinction(S,snwTXT);
44 S = density(S,snwTXT);
45 S = definefunctions(S,snwTXT,C(10),A(end,1));
46
47 % 4 - APPLY MULTIPLIERS
48 % 4.1 - Re-size multipliers arrays to necessary size
49 aM = [1;aM(1:size(A,2)-1)]; % 1 adds a column for time

```

```

50     sM = [1;sM(1:size(S,2)-1)]; % 1 adds a column for the depth
51
52 % 4.2 - Apply multipliers
53 for i = 1:length(sM); S(:,i) = S(:,i) * sM(i); end
54 for i = 1:length(aM); A(:,i) = A(:,i) * aM(i); end
55
56 %-----
57 function A = albedo(A,atmTXT,S)
58 % ALBEDO applies special input into albedo column: dXX, classX, <type>
59 % Special values given in the albedo column (#4) assume that the shortwave
60 % column (3) is an all-wave value, so it is divided into a VIS/NIR
61 % components as is the albedo for all "special" cases
62
63 % 1 - Determine "special" locations
64 idx = find(isnan(A(:,4)));
65
66 % 2 - Cycle through each special value and compute desired albedos
67 for i = 1:length(idx);
68     val = atmTXT{idx(i)+3,4}; % Current special case
69
70     % Optical depth case: dXX
71     if strcmpi('d',val(1)); % Optical depth caer
72         dopt = str2double(val(2:length(val)));
73         if isnan(dopt);
74             error('xls_input:albedo','optical_depth_ill_define.');
```



```

142     dopt = str2double(val(2:length(val)));
143     if isnan(dopt);
144         error('xls_input:extinction','optical_depth_ill_define. ');
145     end
146     [~,S(idx(i),6),~,S(idx(i),7)] = rad_calc(dopt,S(1,2));
147
148 % Class case: classX
149 elseif length(val) > 5 && strcmpi('class',val(1:5));
150     cls = str2double(val(6:length(val)));
151     if isnan(cls);
152         error('xls_input:extinction','class_ill_define. ');
153     end
154     [~,S(idx(i),6),~,S(idx(i),7)] = rad_calc('class',cls);
155
156 % Record an error
157 else
158     error('xls_input:albedo','error_with_albedo_input,column 4!');
159 end
160 end
161
162 %-----
163 function S = density(S,snwTXT)
164 % DENSITY applies special input for density and/or thermal conductivity
165 % columns, either can be 'auto', just not both. The auto values are
166 % replaced it the appropriate value from Sturm, 1997. The quadratic is used
167 % for solving for k and the exponential when solving for density
168
169 for i = 1:size(S,1);
170     rho = S(i,2)/1000; k = S(i,3);
171
172 % Case when both rho and k are defined with numbers
173 if isnumeric(rho) && isnumeric(k) && ~isnan(rho) && ~isnan(k);
174     S(i,2) = rho*1000; S(i,3) = k;
175
176 % Case when the density is computed
177 elseif isnan(rho) && isnumeric(k) && strcmpi(snwTXT{i+3,2},'auto');
178     S(i,2) = (log10(k) + 1.652) / 2.65 * 1000;
179
180 % Case when thermal conductivity is computed
181 elseif isnumeric(rho) && isnan(k) && strcmpi(snwTXT{i+3,3},'auto');
182     if rho < 0.156;
183         S(i,3) = 0.023 + 0.234 * rho;
184     else
185         S(i,3) = 0.138 - 1.01*rho + 3.233 * rho^2;
186     end
187
188 % Failure
189 else
190     error('xls_input:density',...
191         'error with density/conductivity input, column 2 and/or 3!');
192 end
193 end

```

7.3 xls_prep.m

```

1 function [s,a] = xls_prep(snow,atm,constants)
2 % XLS-PREP builds arrays for inputing into thermal model
3 %-----
4 % SYNTAX:
5 %     [snow,atm] = prep_input(snow,atm,constants);
6 %
7 % INPUT:
8 %     snow      = matrix containing snow data
9 %     atm       = matrix containing atmospheric data
10 %     constants = matrix containing model constants
11 %
12 % EXAMPLE INPUT:
13 %     snow      = [50,130,0.06,2030,-10,70];
14 %     atm       = [6,240,500,0.82,1.7,-10,.2,-10,101];
15 %     constants = [2833,0.0023,0.0023,0.622,0.462,-5,0.402,0.95,1,60,1];
16 %-----
17
18 % 1 - Fill in atmospheric data
19 atm(:,1) = atm(:,1) .* 3600; % Convert time to seconds
20 dt = constants(10);          % Time step in seconds
21 a = fill_array(atm,dt);
22
23 % 2 - Fill and snow properties data
24 dz = constants(9);
25 s = fill_array(snow,dz);
26
27 %-----
28 % SUBFUNCTION: fill_array
29 function out = fill_array(in,int)
30 % FILL_ARRAY builds an array from "in" using the interval in "int" based on
31 % the first column of data

```

```

32
33 % 1 - Build array for case when data is only a single row (constant data)
34 len = size(in,1);
35 if len == 1;
36     in(2,:) = in(1,:);
37     in(1,1,:) = 0;
38 end
39
40 % 2 - Build new array with spacing based on "int"
41 % 2.1 - Build the first column of the new array (e.g. time steps)
42 n = size(in,1);
43 xi = (in(1,1):int:in(n,1))';
44
45 % 2.2 - Interpolate the remaining data based on the first column
46 for i = 1:size(in,3);
47     x = in(:,1,i);
48     Y = in(:,2:size(in,2),i);
49     yi = interp1(x,Y,xi,'linear');
50     out(:,i) = [xi,yi];
51 end

```

7.4 thermal.m

```

1 function [T,Q] = thermal(snow,atm,C)
2 % THERMAL executes 1-D heat equation based thermal model
3
4 % SYNTAX:
5 [T,Q] = thermal(snow,atm,C)
6
7 % DESCRIPTION:
8 [T,Q] = thermal(snow,atm,C) based on the information provided in the
9 numeric arrays containing snow properties (snow), atmospheric
10 conditions (atm), and model constants (C) a 1-D thermal analysis is
11 performed resulting in the snowpack temperatures (T) and associated
12 heat fluxes (Q)
13
14 % 1 - PREPARE VARIABLES FOR CALCULATION
15 % 1.1 - Pre-define arrays
16 if size(atm,2) == 11 && size(snow,2) == 7;
17     ndim = 2;
18 else
19     ndim = 1;
20 end
21
22 nt = size(atm,1); % Number of time steps
23 ns = size(snow,1); % Number of snow elements
24
25 T = zeros(ns,nt); % Temperature array
26 q = zeros(ns,nt,ndim); % Short-wave flux absorbed array
27 qs = zeros(nt,3); % Surface flux array
28
29 A = zeros(ns+1,ns+1); % A-matrix for temperature solution
30 b = zeros(ns+1,1); % b-vector for temperature solution
31
32 % 1.2 - Establish user specified constants
33 Ls = C(1);
34 Ke = C(2);
35 Kh = C(3);
36 MvMa = C(4);
37 Rv = C(5);
38 T0 = C(6) + 273.15;
39 e0 = C(7);
40 emis = C(8);
41 dz = C(9)/100;
42 dt = C(10);
43
44 % 1.3 - Define additional constants needed
45 sb = 5.6696*10^(-8); % Stefan Boltzmann constant (W/m^2/K^4)
46 R = 0.287; % Gas constant for air (kJ/kg/K)
47
48 % 1.4 - Compute the properties of air
49 Cp_air = 1003; % Specific heat @-5C (J/kg/K)
50 rho_air = atm(:,9)/(R*(atm(:,6) + 273.15)); % Density (kg/m^2)
51
52 % 2 - INITILIZE ARRAYS FOR COMPUTATION
53 % 2.1 - Initilize temperature array
54 T(:,1) = snow(:,5); % Initial snow temperature
55 T(ns+1,:) = atm(:,8); % Base
56
57 % 2.2 - General Matrix coefficients
58 Ca = squeeze(snow(:,3,:)) ./ dz^2; % a
59 Cb = squeeze((snow(:,2,:)) .* snow(:,4,:)) ./ dt; % b
60 Cc = Cb + Ca; % c
61 Cd = Cb - Ca; % d
62
63

```

```

64 % 3 - BEGIN COMPUTING FOR EACH TIME STEP (time step = index "j")
65 for j = 2:nt
66     % 3.1 - Establish air/snow surface temperatures
67     Ta = atm(j,6) + 273.15;
68     Ts = T(1,j-1) + 273.15;
69
70     % 3.2 - Compute longwave heat flux
71     qs(j,1) = atm(j,2) - emis*sb*Ts^4;
72
73     % 3.3 - Compute the latent heat flux
74     ea = e0*exp(Ls/Rv *(1/T0 - 1/Ta))*atm(j,7)/100;
75     es = e0*exp(Ls/Rv *(1/T0 - 1/Ts));
76     qs(j,2) = 1000*MvMa*rho_air(j)*Ls*Ke*atm(j,5)*(ea-es)/atm(j,9);
77
78     % 3.4 - Compute the sensible heat flux
79     qs(j,3) = Kh*rho_air(j)*Cp_air*atm(j,5)*(Ta - Ts);
80
81     % 3.5 - Compute the absorbed shortwave and build solution matrix
82     % for each layer of snow
83     % 3.5.1 - Compute shortwave absorbed in the top layer
84     q(1,j,1) = atm(j,3)*(1-atm(j,4))*(1-exp(-snow(1,6)*dz));
85
86     % 3.5.2 - Compute shortwave in NIR if present
87     if ndim == 2;
88         q(1,j,2) = atm(j,10)*(1-atm(j,11))*(1-exp(-snow(1,7)*dz));
89     end
90
91     % 3.5.2 - Compute shortwave absorbed for lower layers and build
92     % solution matrices
93     for i = 2:ns
94         % Short-wave radiation absorbed
95         q(i,j,1) = q(i-1,j,1)*exp(-snow(i,6)*dz); % all-wave or VIS
96         if ndim == 2;
97             q(i,j,2) = q(i-1,j,2)*exp(-snow(i,7)*dz); % NIR
98         end
99
100         % Solution matrices
101         A(i,i-1) = -Ca(i,j)/2;
102         A(i,i) = Cc(i,j);
103         A(i,i+1) = -Ca(i,j)/2;
104         b(i,1) = Ca(i,j)/2*T(i-1,j-1) + Cd(i,j)*T(i,j-1) + ...
105             Ca(i,j)/2*T(i+1,j-1) + sum(q(i,j,:))/dz;
106     end
107
108     % 3.6 - Compute the surface flux
109     sur_flux = sum(qs(j,1:3));
110
111     % 3.7 - Insert matrix values for surface node (i = 1)
112     A(1,1) = Cc(1,j);
113     A(1,2) = -Ca(1,j);
114     b(1) = Cd(1,j)*T(1,j-1) + Ca(1,j)*T(2,j-1) + 2*sur_flux/dz + ...
115         sum(q(1,j,:))/dz;
116
117     % 3.8 - Insert matrix values for bottom boundary condition
118     A(ns+1,ns+1) = 1;
119     b(ns+1) = atm(j,8);
120
121     % 3.9 - Calculate the new temperature profile
122     Tnew = A\b;
123     Tnew(Tnew>0) = 0;
124     T(:,j) = Tnew;
125 end;
126
127 Q = zeros(ns,nt,ndim+3);
128 Q(1,:,1:3) = qs;
129 Q(:, :, 4:end) = q;

```

7.5 rad_calc.m

```

1 function varargout = rad_calc(varargin)
2 % RAD.CALC spectral calculations of radiation, albedo, and extinction.
3 %-----
4 % SYNTAX:
5 % [SWvis,SWnir,SWswir] = rad_calc(SWall);
6 % [Avis,Anir,Aswir] = rad_calc(curve);
7 % [Avis,Bvis,Anir,Bvis,Aswir,Bswir] = rad_calc(dopt,rho);
8 % [Avis,Bvis,Anir,Bvis,Aswir,Bswir] = rad_calc('class',num);
9 %
10 % DESCRIPTION:
11 % [SWvis,SWnir,SWswir] = rad_calc(SWall) computes spectral components of
12 % all-wave shortwave radiation based on ASTM standard.
13 % [Avis,Anir,Aswir] = rad_calc(curve) computes spectral albedo components
14 % based on curves: 'fine','medium','coarse'
15 % [Avis,Bvis,Anir,Bvis,Aswir,Bswir] = rad_calc(dopt,rho) computes
16 % spectral components of albedo and extinction based on Snow & Climate
17 % equations given on p.56.

```

```

18 % [Avis,Bvis,Anir,Bvis,Aswir,Bswir] = rad_calc('class',num) computes
19 % spectral components of albedo and extinction based on Snow & Climate
20 % table given on p.57, where num must be an integer between 1 and 6.
21 % -----
22
23 % 1 - Compute desired values, execute as order in SYNTAX/DESCRIPTION above
24 if nargin == 1 && isnumeric(varargin{1});
25     output = shortwave(varargin{1});
26 elseif nargin == 1 && ischar(varargin{1});
27     output = albedo_curve(varargin{1});
28 elseif nargin == 2 && isnumeric(varargin{1});
29     output = albedo_eqn(varargin{:});
30 elseif nargin == 2 && ischar(varargin{1});
31     output = albedo_table(varargin{2});
32 end
33
34 % 2 - Produce output
35 varargout = num2cell(output);
36
37 % -----
38 function out = albedo_table(N)
39 % ALBEDO.TABLE computes albedo and extinction base on Snow&Climate(p.57)
40
41 % Error handling
42 if N < 1 || N > 6;
43     error('Class must be an integer 1 through 6!'); out = NaN; return;
44 end
45
46 % Build Table 2.6 from Snow & Climate (2008), p.57
47 C(:,1) = [94,94,93,93,92,91]/100;
48 C(1:6,2) = 40;
49 C(:,3) = [80,73,68,64,57,42]/100;
50 C(:,4) = [110,136,190,110,112,127];
51 C(:,5) = [59,49,42,37,30,18]/100;
52 C(1:6,6) = inf;
53
54 % Produce output
55 out = C(N,:);
56
57 % -----
58 function out = albedo_eqn(dopt,rho)
59 % ALBEDO.EQN computes albedo and extinction base on Snow&Climate(p.56)
60
61 % Convert units (dopt mm->m; rho kg/m^3->gm/cm^3)
62 dopt = dopt/1000; rho = rho/1000;
63
64 % VIS
65 out(1) = min(0.94,0.96 - 1.58*sqrt(dopt));
66 out(2) = max(0.04, 0.0192*rho/sqrt(dopt))*100;
67
68 % NIR
69 out(3) = 0.95 - 15.4 * sqrt(dopt);
70 out(4) = max(1, 0.1098*rho/sqrt(dopt))*100;
71
72 % SWIR
73 out(5) = 0.88 + 346.6*dopt - 32.31*sqrt(dopt);
74 out(6) = inf;
75
76 % -----
77 function Aout = albedo_curve(curve)
78 % ALBEDO.CURVE computes VIS,NIR,& SWIR albedos based on input curve
79
80 % 1 - Load the desired curve
81 X = load('albedo.mat');
82 A = X.(curve);
83
84 % 2 - Parse out the albedo for each wavelength group
85 L = [285,800; 800,1500; 1500,3500];
86 for i = 1:size(L,1);
87     idx(1) = find(A(:,1)>=L(i,1),1,'first');
88     idx(2) = find(A(:,1)<=L(i,2),1,'last');
89     Aout(i) = mean(A(idx(1):idx(2),2))/100;
90 end
91
92 % -----
93 function SWout = shortwave(SWall)
94 % SHORTWAVE computes spectral components of all-wave based on ASTM standard
95
96 % 1 - Load the solar spectrum desired
97 X = load('albedo.mat');
98 S = X.astm;
99
100 % 2 - Normalize solar spectrum to inputted SW data
101 I = insolation(S,[285,3500]);
102 S(:,2) = (S(:,2)/I)*SWall;
103
104 % 3 - Parse out wavelength groups
105 L = [285,800; 800,1500; 1500,3500];
106 for i = 1:size(L,1); SWout(i) = insolation(S,L(i,:)); end
107
108 % -----
109 function I = insolation(S,L)

```

```

110 % POWER computes the insolation between the a and b wavelenghts
111
112 % 1 - Locate indicies of wavelenghts
113 x = S(:,1); y = S(:,2);
114 i(1) = find(x>L(1),1,'first');
115 i(2) = find(x<L(2),1,'last');
116 idx = i(1):i(2);
117
118 % 2 - Compute the insolation
119 I = sum((y(i(1):i(2)-1)+diff(y(idx)).*diff(x(idx))));

```

7.6 confint.m

```

1 function data = confint(filename,B,n)
2 % CONFINT computes the confidence intervals for temp profiles
3
4 % Read file input
5 [S,A,C,E] = xls_input(filename);
6
7 % Compute the actual temperature profile
8 [Sa,Aa] = xls_prep(S,A,C);
9 T = thermal(Sa,Aa,C);
10
11 % Compute the standard deviation values
12 s = getstd(S,E.snow,n,1); % standard deviation for snow properties
13 a = getstd(A,E.atm,n,1); % standard deviation for atmospheric terms
14 c = getstd(C,E.const,n,0); % standard deviation for constants
15
16 % Compute the Monte Carlo replicates
17 data.Tboot = zeros([size(T),B]); % Initilize storage array
18 h = waitbar(0,'Please wait...');
19 for i = 1:B;
20     r = rand(1);
21     S_b = norminv(r,S,abs(s)); % Re-sample snow
22     S_b(:,1) = S(:,1); % Snow depth does not change
23     S_b(isnan(S_b)) = S(isnan(S_b));
24     A_b = norminv(r,A,abs(a)); % Re-sample atmosphere
25     A_b(:,1) = A(:,1); % Duration does not change
26     A_b(isnan(S_b)) = A(isnan(A_b));
27     C_b = norminv(r,C,abs(c)); % Constant resampling
28     C_b(9) = C(9); % dz constant
29     C_b(10) = C(10); % dt constant
30     C_b(isnan(C_b)) = C(isnan(C_b));
31
32     [SS,AA] = xls_prep(S_b,A_b,C_b); % Build input for evaluation
33     [data.Tboot(:, :, i), data.Qboot(:, :, i)] = thermal(SS,AA,C_b);
34     data.Sboot(:, :, i) = SS;
35     data.Aboot(:, :, i) = AA;
36     data.Cboot(:, :, i) = C_b;
37     waitbar(i/B,h);
38 end
39 close(h);
40
41 %-----
42 function s = getstd(S,E,n,offset)
43 % GETSTD returns the standard deviation of the input items
44 if offset == 1;
45     s(:,1) = S(:,1);
46     for i = 2:size(S,2);
47         s(:,i) = S(:,i).*E(i-offset)/n;
48     end
49 elseif offset == 0;
50     for i = 1:size(S,2);
51         s(:,i) = S(:,i).*E(i)/n;
52     end
53 end

```