



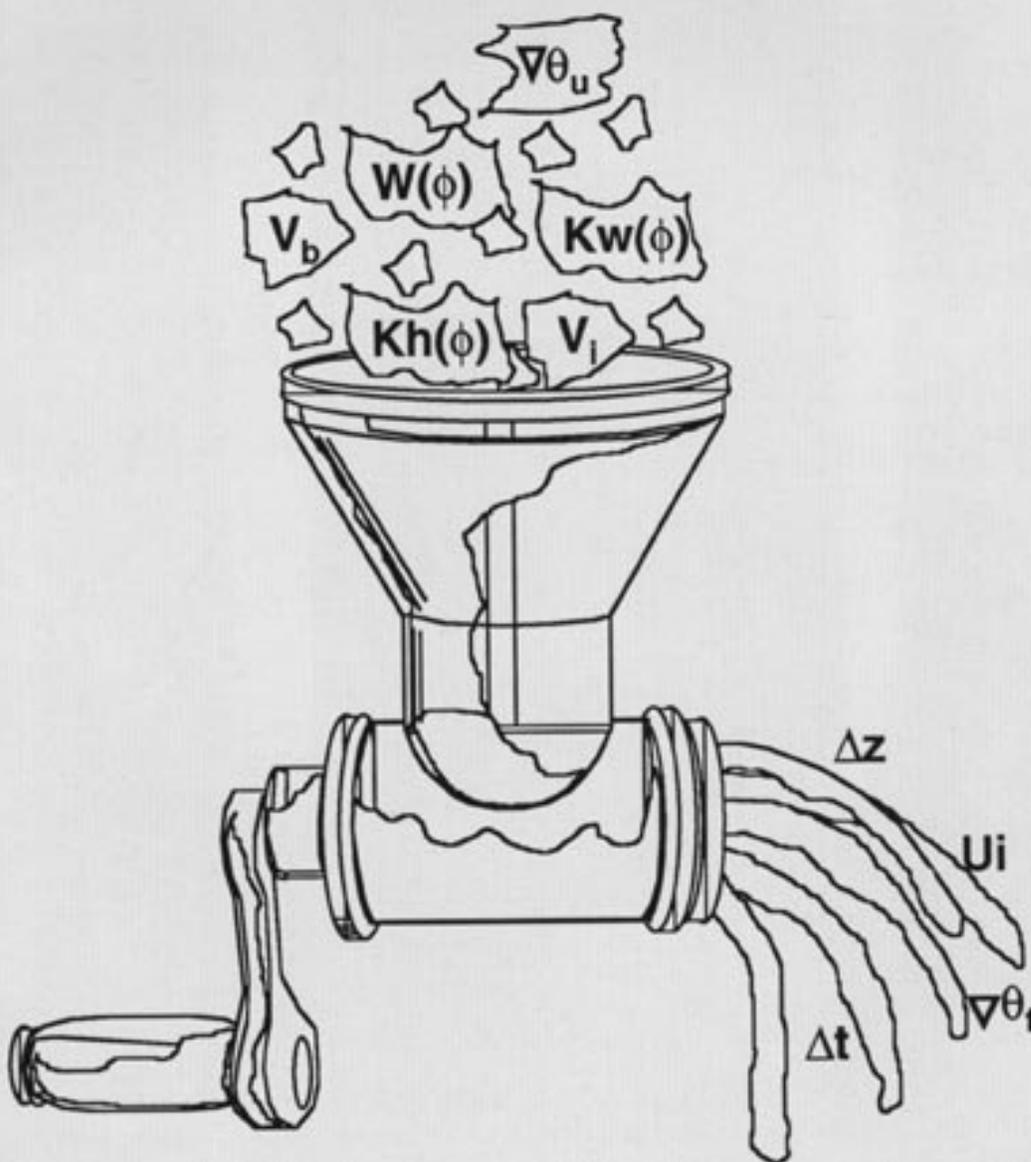
95-12

CRREL REPORT

RIGIDICE Model of Secondary Frost Heave

Patrick B. Black

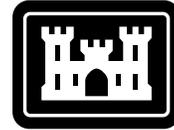
May 1995



Abstract

A revised version of an earlier attempt to numerically solve Miller's equations for the RIGIDICE model of frost heave is presented that corrects earlier mistakes and incorporates recent improvements in the scaling factors of ground freezing. The new version of the computer code also follows the concepts of Object Oriented Numerics (OON), which allow for easy modification and enhancements. Analysis of the program is accomplished with the symbolic math program MathCad. A brief sensitivity analysis of the input variables indicates that those parameters that calculate the hydraulic conductivity have the greatest influence on the variability of predicted heaving pressure.

CRREL Report 95-12



**US Army Corps
of Engineers**

Cold Regions Research &
Engineering Laboratory

RIGIDICE Model of Secondary Frost Heave

Patrick B. Black

May 1995

Prepared for
OFFICE OF THE CHIEF OF ENGINEERS

Approved for public release; distribution is unlimited.

PREFACE

This report was prepared by Dr. Patrick B. Black, Soil Physicist, Applied Research Branch, Experimental Engineering Division, U.S. Army Cold Regions Research and Engineering Laboratory. Funding was provided by U.S. Army Project 4A762784AT42, *Pavements in Cold Regions* (231), Task, BS, Work Unit AT42-CP-C01, *Freezing and Thawing of Soil Water Systems*.

Technical review was provided by Dr. S. Colbeck and Dr. R. Berg, both of CRREL.

The contents of this report are not to be used for advertising or promotional purposes. Citation of brand names does not constitute an official endorsement or approval of the use of such commercial products.

CONTENTS

	Page
Preface	ii
Nomenclature	iv
Introduction	1
RIGIDICE	1
First physical approximation	1
Second physical approximation	3
Darcy's law	3
Fourier's law	3
Clapeyron equation	3
Criterion for lens initiation	4
Algorithm	4
RIGIDICE with object-oriented numerics	5
Discussion	7
Conclusions	11
Literature cited	11
Appendix A: RIGIDICE model	13
Abstract	33

ILLUSTRATIONS

Figure

1. Lensing cycle	2
2. Flow chart of RIGIDICE	4
3. MathCad program to run RIGIDICE	6
4. Changes in heave pressure vs. heave rate for changes in chosen input parameters	7

TABLES

Table

1. Sensitivity results of calculated heave pressure to input parameter changes in RIGIDICE	10
--	----

NOMENCLATURE

<i>Symbol</i>	<i>Units</i>	<i>Definition</i>
G, W, I	$\text{m}^3 \text{m}^{-3}$	volume fractions of grains, water and ice
$W_{n'}, W_{n+1}$	$\text{m}^3 \text{m}^{-3}$	volumetric water content within layers n and $n+1$ in the fringe
e		porosity
$W_{\text{sat}}, W_{\text{d}}$	$\text{m}^3 \text{m}^{-3}$	volumetric water contents at saturation and at the lower limit of freezing
$W_{\text{f}}, I_{\text{f}}$	$\text{m}^3 \text{m}^{-3}$	volumetric water content and ice content of the frozen soil existing between two mature ice lenses
v_{i}	m s^{-1}	heave rate
v_{b}	m s^{-1}	penetration rate of freezing front
q_{w}	$\text{m}^3 \text{m}^{-2} \text{s}^{-1}$	flux of water
$(q_{\text{w}})_{\text{b}}$	$\text{m}^3 \text{m}^{-2} \text{s}^{-1}$	flux of water into the fringe
$(q_{\text{w}})_{\text{n}'}, (q_{\text{w}})_{\text{n}+1}$	$\text{m}^3 \text{m}^{-2} \text{s}^{-1}$	flux of water into a layer in the fringe
k_{w}	m s^{-1}	hydraulic conductivity
$(k_{\text{w}})_{\text{sat}}$	m s^{-1}	saturated hydraulic conductivity
α, β	—	Brooks and Corey coefficients
$u_{\text{w}}, u_{\text{i}}$	Pa	water and ice gauge pressures
ϕ	Pa	capillary pressure ($u_{\text{i}} - u_{\text{w}}$)
ϕ_{b}	Pa	capillary pressure at base of fringe (i.e., ice entry pressure)
$\phi_{\text{n}'}, \phi_{\text{j}}$	Pa	capillary pressures within layers n and j in the fringe
$\sigma_{\text{T}}, \sigma_{\text{e}}, \sigma_{\text{n}}$	Pa	total, effective and neutral stresses as expressed by the Terzaghi equation
χ	—	Snyder-Bishop stress partition factor
f_{w}	N m^{-3}	body force on liquid water
q_{h}	W m^{-2}	flux of heat
$(q_{\text{h}})_{\text{n}'}, (q_{\text{h}})_{\text{n}+1}$	W m^{-2}	flux of heat within layers n and $n+1$ in the fringe
$(q_{\text{h}})_{\text{b}'}, (q_{\text{h}})_{\text{f}}$	W m^{-2}	flux of heat through bottom of fringe and upper boundary zone
$(k_{\text{h}})_{\text{g}'}, (k_{\text{h}})_{\text{w}'}, (k_{\text{h}})_{\text{i}}$	$\text{W K}^{-1} \text{m}^{-1}$	thermal conductivities of grains, water and ice
θ	$^{\circ}\text{C}$	temperature
θ_0	K	absolute temperature of a flat ice/water interface in equilibrium at standard conditions
h	J m^{-3}	volumetric latent heat of fusion
Y	—	specific gravity of ice
z	m	position positive up from the bottom of the fringe
$(d\theta/dz)_{\text{b}'}, (d\theta/dz)_{\text{f}}$	$^{\circ}\text{C m}^{-1}$	temperature gradient at base of fringe and in upper boundary zone

RIGIDICE Model of Secondary Frost Heave

PATRICK B. BLACK

INTRODUCTION

In an earlier paper, Black and Miller (1985) reported on progress to numerically solve the differential equations of secondary frost heave (Miller 1978) using a simplified approach. They presented a series of scenarios that predicted anticipated results. The program was experimentally confirmed by comparing observed behavior of independently conducted frost heave tests to model predictions using hydraulic and thermal properties for the test soil. They found that the model indicated that the experimental data were flawed in their stress measurements, but agreed within acceptable tolerance with the thermal measurements. No additional effort was made at that time to extend the analysis or increase the utility of the program.

This report is an extension of the earlier work. It presents improvements in two areas. First, a minor mistake in the calculation of the water flux is corrected; also, the current formulations for the scaled equations of ground freezing are used (Miller 1990). Second, the computer code is presented in a form readily available for enhancements. This is accomplished by employing the concepts of Object Oriented Numerics (OON), which allow the code to be easily added to without directly changing the original source code (Wong et al. 1993).

The concepts and equations of the RIGIDICE (Black and Miller 1985) model are first reviewed. The new C++ formulation of the code is then discussed and presented in its entirety in Appendix A. The code is linked as a Dynamically Linked Library (DLL) that is attached to MathCad 5.0+ (MathSoft 1994). A preliminary sensitivity study is conducted to examine the influence of uncertainty in input parameters on the variability of the calculated output parameters.

RIGIDICE

Black and Miller (1985) simplified the solution of the differential equations for secondary frost heave (Miller 1978) by employing two physical assumptions and one numerical trick. Their model, as well as this model, is valid for one-dimensional, incompressible, air-free and solute-free soil. The solute-free restriction assumes that no additional osmotic gradient is imposed over the ever present osmotic gradient in the double layer of the unfrozen water surrounding the grains. The incompressible restriction excludes the process of consolidation at this time. Finally, the air-free condition states that the region undergoing heave must be water saturated. This makes perfect sense, since air does not freeze, which means that the soil must have been saturated, or within 10%, because of the volume expansion necessary for a lens to appear. These restrictions are minor since they result in the model predicting the behavior of the highly frost-susceptible fine-grain silts.

First physical approximation (lensing cycle)

First, the progression of frost heaving through soil is approximated physically by a series of independent lensing cycles. A lensing cycle is defined to be the time step that begins with the formation of a lens and ceases with the initiation of a new lens. During each individual lensing cycle, the frozen soil above the fringe is composed of a series of identical layers of ice and frozen soil as depicted in Figure 1. The thickness of these lenses and the frozen soil between them is equal to the thickness of the mature lens and interspaced frozen soil at the end of the lensing cycle.

This approximation bounds the fringe between the lower freezing front and the upper boundary

zone. The freezing front is the plane that separates soil containing ice from the unfrozen soil. The upper boundary zone is not defined by a plane, but is rather a zone that always contains a thickness of pure ice equal to a mature ice lens and a thickness of frozen soil equal to the thickness of the mature interspace frozen soil. Figure 1 also shows the location of these regions at various stages throughout the lensing cycle.

Under steady-state conditions, the inputs across the freezing front are given by

grains	Gv_b
water	$W_{\text{sat}}v_b + (q_w)_b$
ice	0
sensible heat	$(q_h)_b$
latent heat	$h [W_{\text{sat}}v_b + (q_w)_h]$

and the outputs across the boundary zone are

grains	Gv_b
water	$W_d v_b$
ice	$I_f v_b + v_i$
sensible heat	$(q_h)_f$
latent heat	$h [W_f v_h]$.

The global mass balance is therefore

$$(q_w)_b = Yv_i + [(Y - 1)(W_{\text{sat}} - W_f)]v_b \quad (1)$$

and heat balance is

$$(q_h)_f = (q_h)_b + h [(W_{\text{sat}} - W_f)v_b + (q_w)_b]. \quad (2)$$

In its purest sense, this results in six unknowns and two equations. In practice, though, several of the variables are known from other relationships or observations. The numerical trick is to specify at least three of the variables and guess another to start the numerical solution process.

Temperature profiles during heaving are the most common piece of experimental information collected because it is easy to measure. Freezing front penetration rate is readily determined from these data, so v_b is one logical variable to specify. Heat fluxes are also readily determined from temperature profile data if the thermal conductivities are known. The thermal conductivity of the unfrozen soil is a constant in this one-dimensional case because of the air-free restriction. The sensible heat flux across the frozen fringe $(q_h)_b$ therefore makes another reasonable choice. Unfortunately, there is no a priori justification for choosing from the remaining variables. The thermal conductivity in the frozen soil could be computed if the lens thickness and spacing and the residual unfrozen water content W_f were known. The flux of water into the fringe could also be computed if the amount of heave were known. For lack of any further justification, the rate of heave v_b is fixed.

If the residual unfrozen water W_f in the frozen soil between ice lenses is known, then the remain-

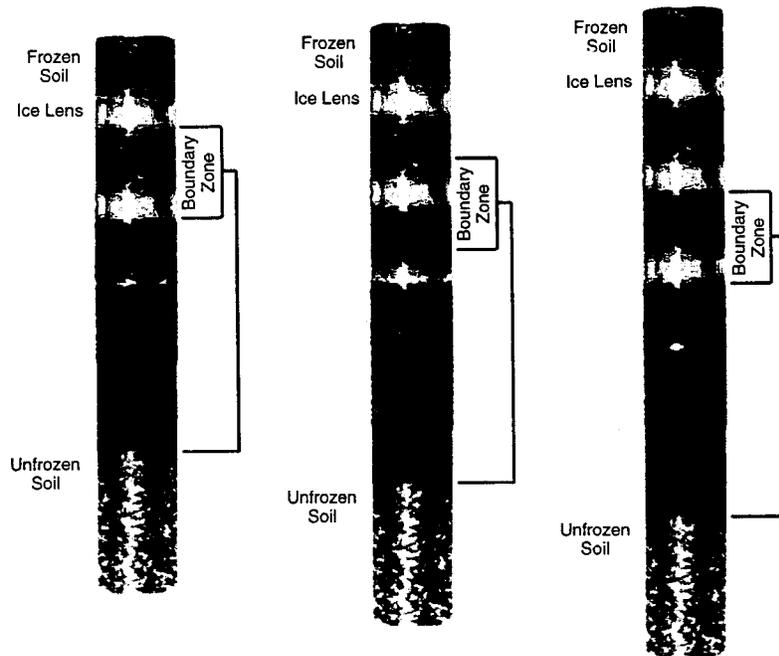


Figure 1. Lensing cycle.

ing variables are known for a given soil. As a first approximation, the amount of residual unfrozen water is assumed to be equal to the lower limit of freezing for the soil. In some situations, this might represent all the required information for the modeler. More often, the heaving pressure, and lens thickness and spacing are also required.

Second physical approximation (equivalence of instantaneous and averaged fluxes)

To calculate the heaving pressure and lens location within the fringe, profiles of temperature and pressures must be calculated throughout the fringe. To perform such calculations, a method of executing mass and energy balances within the fringe must be obtained. This is accomplished with the second physical approximation. It states that the instantaneous fluxes of matter and energy at the beginning and the end of the lensing cycle are equal to the averaged values of the fluxes during the lensing cycle.

Another way of stating the second approximation is that any instantaneous fluctuations in magnitude of the mass and energy fluxes during the lensing cycle are negligible. This means that the magnitude of the penetration rate, heave rate and temperature gradient within the unfrozen soil are invariant with time and space. In finite difference form, the local mass balance within the fringe is

$$(q_w)_{n+1} = (q_w)_n + v_b - Y(v_b + v_i)[W_n - W_{n+1}], \quad (3)$$

and thermal balance is

$$(q_h)_{n+1} = (q_h)_n + h[(W_n - W_{n+1})v_b + (q_w)_n - (q_w)_{n+1}]. \quad (4)$$

The remaining information required to complete all calculations are statements for water and ice pressures, temperature and a criterion for lens initiation.

Darcy's law

The flux of water through the fringe is assumed to obey Darcy's law

$$\frac{du_w}{dz} = f_w - \frac{q_w}{k_w}. \quad (5)$$

This relationship introduces the soil function k_w , the hydraulic conductivity. Each soil will have its unique hydraulic conductivity function. It is not a specific property of heaving soils, but is rather a general material property of all frozen soils, heaving or non-heaving. It must be known in order to calculate the heaving process.

To date, most efforts to calculate this material property are by inference. It is inferred from thermal analysis (van Loon et al. 1988), back calculations (Ratkje et al. 1982) and unsubstantiated inference to non-frozen soil (Guymon and Luthin 1974). Black and Miller (1990) were able to directly measure the change in hydraulic conductivity in air-free, lens-free and solute-free frozen soil as a function of unfrozen water content. Their analysis found that if the measured hydraulic conductivity was expressed a function of the difference between the ice and water pressures

$$\phi = u_i - u_w \quad (6)$$

then the analogous expression given by Brooks and Corey (1964) for partially saturated and ice-free soil could be transformed to

$$W(\phi) = (W_{\text{sat}} - W_d) \left(\frac{\phi_b}{\phi} \right)^\alpha + W_d \quad (7)$$

and

$$k_w(\phi) = (k_w)_{\text{sat}} \left(\frac{\phi_b}{\phi} \right)^\beta. \quad (8)$$

Fourier's law

The flux of thermal energy through the fringe is assumed to follow the Fourier's law

$$\frac{d\theta}{dz} = -\frac{q_h}{k_h}. \quad (9)$$

This relationship introduces another soil function, k_h , the thermal conductivity for the frozen soil. Just as the hydraulic conductivity, this material property is also a function of the pressure difference between the ice and water pressures. One standard expression is the geometric mean formulation of Farouki (1981)

$$k_h(\phi) = (k_h)_G^e (k_h)_W^{W(\phi)} (k_h)_I^{I(\phi)}. \quad (10)$$

Clapeyron equation

The equilibrium condition between the ice and water pressures and temperature is given by the Clapeyron equation

$$\frac{d\phi}{dz} = (\gamma - 1) \frac{du_w}{dz} - \frac{\gamma h}{\theta_o} \frac{d\theta}{dz}. \quad (11)$$

Criterion for lens initiation

To calculate when a new lens forms and the lensing cycle is complete, a criterion for lens initiation is required. Simply stated, a new lens will form when ice can penetrate between the grains, causing the grains to separate. Other factors must be included if the porous material was not granular. For example, cements and rocks that have chemical bonds between the grains must have the bonds broken before the ice can penetrate.

Geotechnical engineers express the state of stress within a granular material by the Terzaghi equation

$$\sigma_T = \sigma_e + \sigma_n \quad (12)$$

and the condition when the grains separate by

$$\sigma_e = 0. \quad (13)$$

At this instance

$$\sigma_T = \sigma_n. \quad (14)$$

Now if σ_T is assumed to be equal to the maximum heaving pressure, then the location where a new ice lens forms is where σ_n also equals the maximum heaving pressure. Snyder and Miller (1985) found that they could correctly model their empirically measured neutral stress data by

$$\sigma_n = \chi u_w + (1 - \chi) u_i \quad (15)$$

in which the new soil function was determined to be

$$\chi(\phi) = \frac{1}{2} \left[\frac{W(\phi) - W_d}{W_{sat} - W_d} - \frac{0.3}{\phi_n} \sum_{j=1}^n \frac{W(\phi_j) - W_d}{W_{sat} - W_d} \right]. \quad (16)$$

Algorithm

The solution strategy begins by stating values for v_i , v_b and $(d\theta/dz)_b$ and all necessary param-

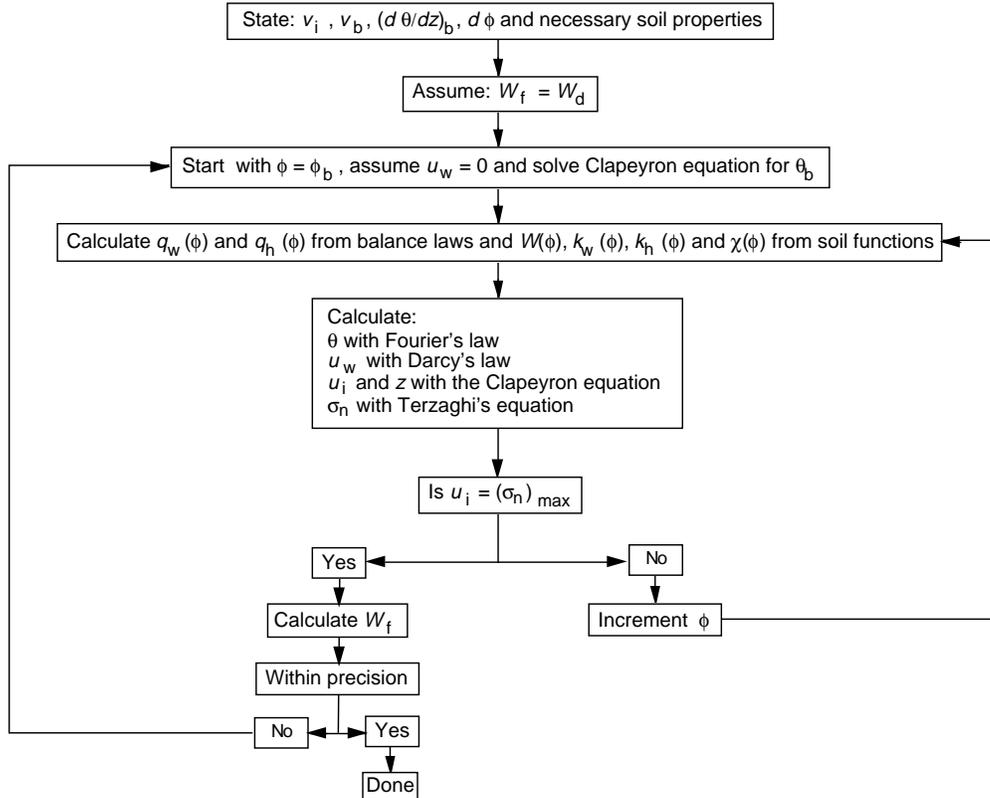


Figure 2. Flow chart of RIGIDICE.

eters for the soil functions. Next, the residual unfrozen water content in the mature frozen soil is assumed to be equal to the lower limit of freezing. Equations 1 and 2 are then solved for the water flux into the fringe and the flux of heat out of the upper boundary.

Profiles of pressures, stresses and temperature are then calculated within the fringe starting at the freezing front. The value of ϕ_b at the freezing front is obtained by setting the local water pressure to zero. This is equivalent to assuming that the water table is at the freezing front. If the water pressure is not zero, then the resulting heaving pressure obtained by assuming zero must be adjusted using eq 11.

Calculations are done in terms of ϕ . In finite difference form, balances are conducted across layers of constant $d\phi$. This has the benefit of generating thin spatial layers where ϕ is changing rapidly and large spatial layers where it changes least.

Darcy's law gives the value of the water pressure across the layer and Fourier's law gives the temperature. The Clapeyron equation then gives the ice pressure as well as the thickness of the layer. Neutral stress is calculated for each layer and a running account of its magnitude recorded to determine its maximum value. A running sum of the unfrozen water content is also made, starting from the location of maximum neutral stress to determine the residual water content. Calculations continue until the ice pressure equals the maximum neutral stress. This is the location of the base of the lens and the new lens will form where the neutral stress was maximum. The distance between these two locations gives the lens spacing. The calculated residual water content is then compared to the original guess. If the difference is unacceptable, the layer by layer calculations are performed again with the new guess until the resulting change in residual water content is acceptable.

A flow chart representation of this algorithm is presented in Figure 2.

RIGIDICE WITH OBJECT-ORIENTED NUMERICS (OON)

Code was written to solve eq 1 through 16 using the algorithm outlined above. To allow for the greatest flexibility of use, as well as ease of future enhancements, C++ was used. The entire code is presented in Appendix A.

The benefit of using OON is that it allows the writing of code in separate layers that are easily

merged together. This might be interpreted as the standard approach of writing a series of subroutines, but it is more. The traditional approach is to write a series of procedures to numerically solve the problem. These procedures are sequentially solved and in large programs result in many pages of critically linked lines of code. When, for example, the routine to calculate hydraulic conductivity must be changed, that section of code must be found and modified. Care must be taken not to remove variables that are used by the rest of the program as well as not to add variables that are already being used in other parts of the program.

The OON approach is to break the problem down into separate self-contained units called classes, a class being a collection of data and operations. Data in one class can be made inaccessible to other classes to prevent inadvertent changes by later modifications to the program. Additional classes can be made that inherit the properties of existing classes. Operations used by a class can be expanded by a process called polymorphism. Again, if a new function to calculate hydraulic conductivity is needed, it is not necessary to write a completely new class, but just merely to add to the current. In other words, the new function would access all the data used by the old as well as any new data that it requires, and the new data are prevented from interfering with any existing data in the program. The traditional approach leads to mistakes and debugging problems. The OON approach rests on the belief that if the original code worked, then don't change it, just add improvements.

RIGIDICE is setup as a series of C++ classes. At the lowest level, functions and data associated with a particular soil are grouped together in a class called *Tsoil*. Functions and data associated with the boundary conditions are grouped together as a class *Tbnds* and numerical precision in a class *Ttol*. Since all variables are reduced to dimensionless form to help in numerical calculations, a class that does scaling is called *REDUCE*. Finally, the calculations that perform the necessary algorithm are made in the class *Trigidice*. This class is declared so that it inherits all the properties of the other classes. If a different algorithm is necessary, then a new one can be written that also inherits the other classes while never touching the original source code for the other primitive classes.

In total, the program requires the initiation of 18 variables. It then returns the values of eight calculated variables. Figure 3 shows the input screen of the MathCad program that runs RIGIDICE.

This program calculates the rate of heave for a given set of initial and boundary conditions. The strategy is the same as presented by Black and Miller (1985). It is assumed that the heaving process can be approximated with a time step called the lensing cycle and that any fluctuations during this lensing cycle average out to values that represent the true behavior during the lensing cycle.

Scaling factors

$\lambda: = 1.0 \cdot 10^{-6}$

$\zeta: = 1.0 \cdot 10^{-2}$

$fw: = 1$

Thermal conductivities

$K_{hw}: = 0.52$

$K_{hi}: = 2.32$

$K_{hg}: = 3.42$

Soil water properties

$W_{sat}: = 0.42$

$W_d: = 0.02$

$K_{wsat}: = 1.0 \cdot 10^{-8}$

Brooks and Corey functions

$\phi_b: = 11.196$

$\alpha: = 0.36$

$\beta: = 2.6$

Boundary conditions

$heave: = 10$

$penetration: = 100$

$grad\theta_{unfrozen}: = -10$

Numeric settings

$prec: = 0.1$

$resol: = 0.01$

$MaxLayers: = 100000$

settings (heave, penetration): =

- λ
- ζ
- fw
- K_{hw}
- K_{hi}
- K_{hg}
- W_{sat}
- W_d
- K_{wsat}
- α
- β
- ϕ_b
- heave
- penetration
- grad $\theta_{unfrozen}$
- prec
- resol
- MaxLayers

The function RIGIDICE takes the contents of settings and returns a 1-dimensional array containing (heave pressure, temperature gradient in frozen soil, heat fluxes into the fringe and out of the upper boundary zone, the water flux into the fringe and the lens spacing and thickness and iterations)

$(ui \ grad\theta_f \ q_{hin} \ q_{hout} \ q_{win} \ spacing \ thickness \ iters): = RIGIDICE (settings [heave, penetration])$

$ui = 76.06 \quad spacing = 0.118 \quad thickness = 4.251 \quad q_{hin} = 15.504 \quad q_{hout} = 199.533 \quad iters = 2 \quad grad\theta_f = -70.255$

Figure 3. MathCad program to run RIGIDICE.

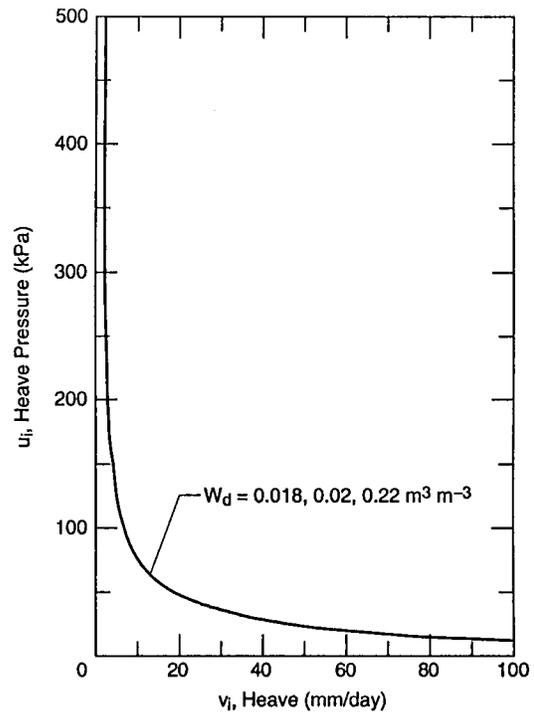
The user is free to modify the contents of all 18 input parameters that are passed to the array *settings*. The function *RIGIDICE(settings)* returns an array that contains the eight calculated variables.

DISCUSSION

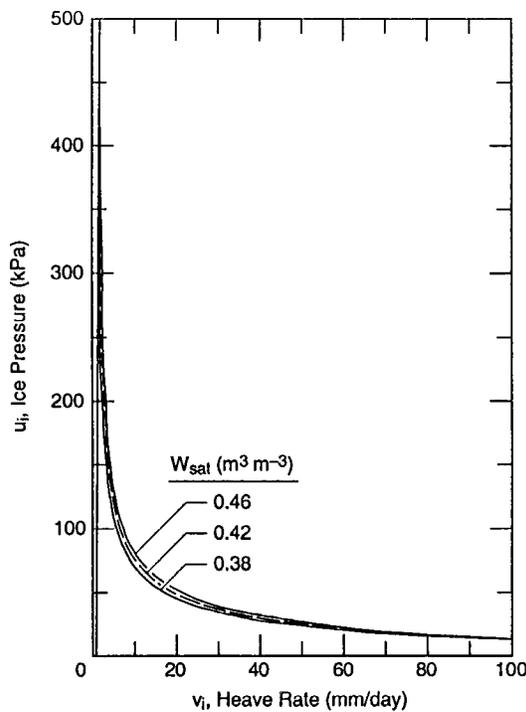
Owing to the number of input parameters, a thorough sensitivity study of the program is very time consuming. This study employed a quicker approach in which the relative significance of the various input parameters is obtained through a simple scheme of sequentially varying one parameter at a time and noting the resulting effect on the important calculated parameters. Each input parameter is modified by 10% and the calculated results noted. The initial reference value for each parameter is obtained from past measurements for one particular silty soil (Black and Miller 1985, 1990).

It is evident that the heaving pressure is of primary concern to the end user. The heaving pressure is therefore the output parameter to be examined. In addition to examining how the heave pressure alone changes with each parameter, the heave rate and heave pressure behavior will be examined as the other parameters are modified one at a time. This approach is similar to how an

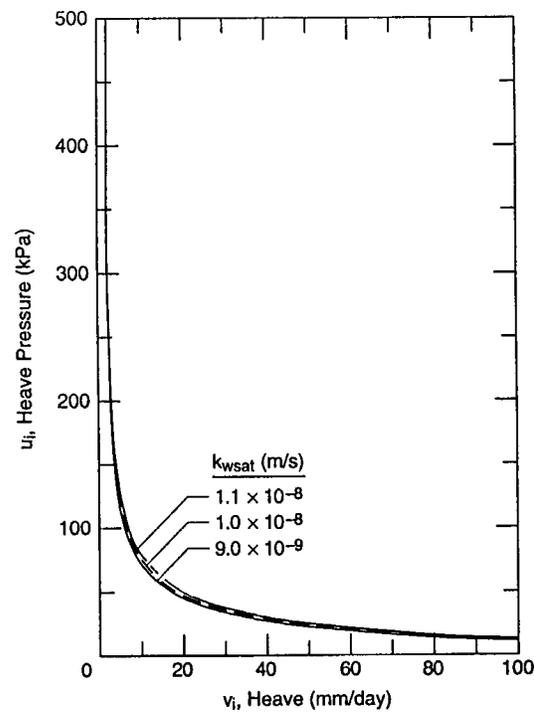
end user might use the program. By stating all the other parameters, the end user is then able to predict heave rate as a function of heave pressure (i.e., overburden pressure).



b. Lower limit of freezing, W_d .

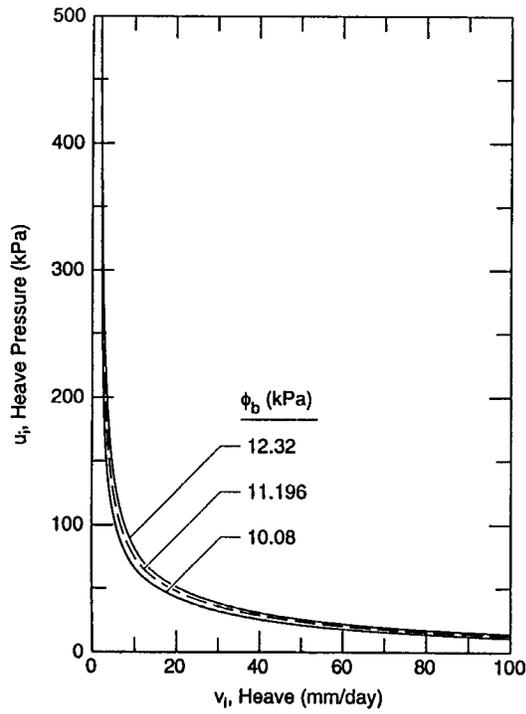


a. Saturated water content, W_{sat} .

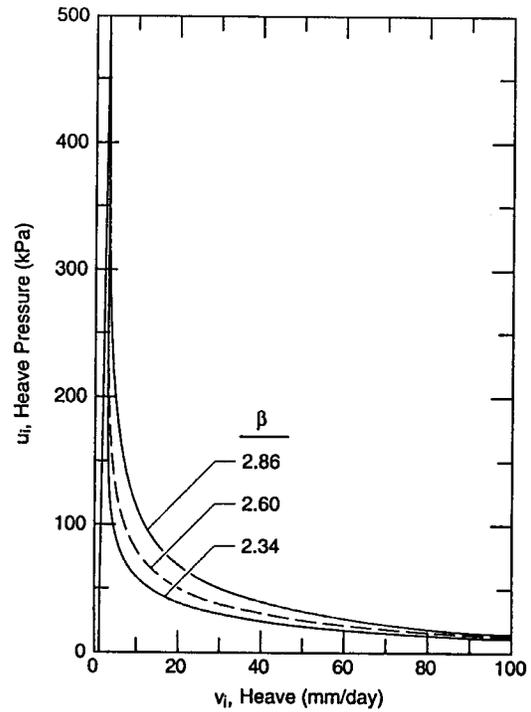


c. Saturated hydraulic conductivity, $(k_w)_{sat}$.

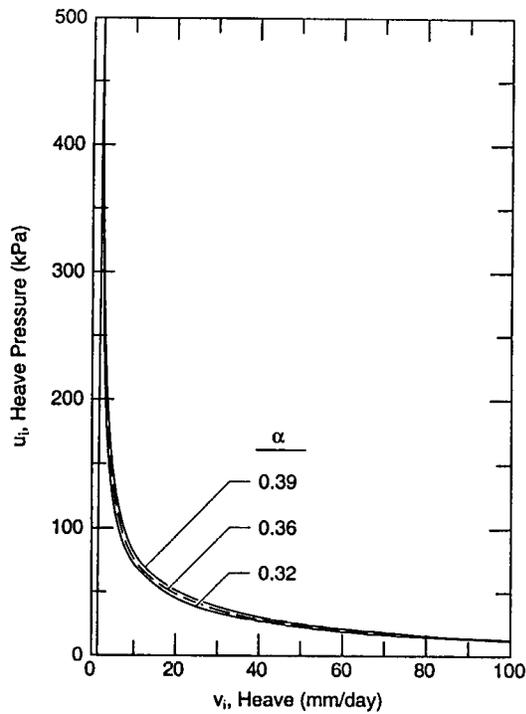
Figure 4. Changes in heave pressure vs. heave rate for changes in chosen input parameters.



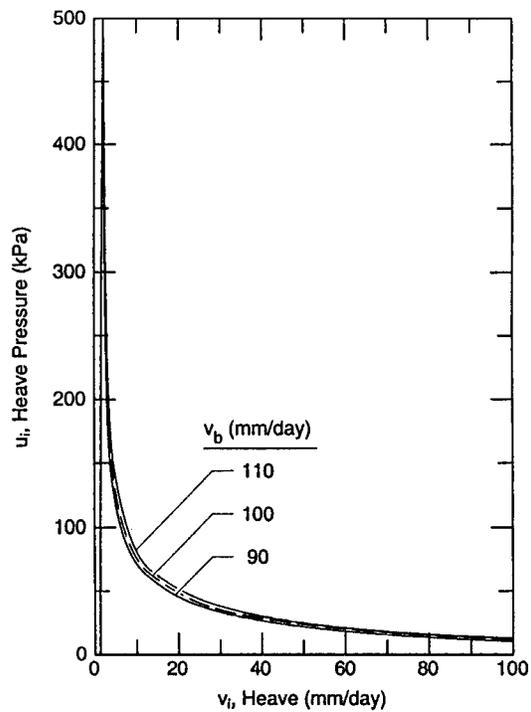
d. Ice entry pressure, ϕ_b .



f. β in Brooks and Corey equation.



e. α in Brooks and Corey equations.



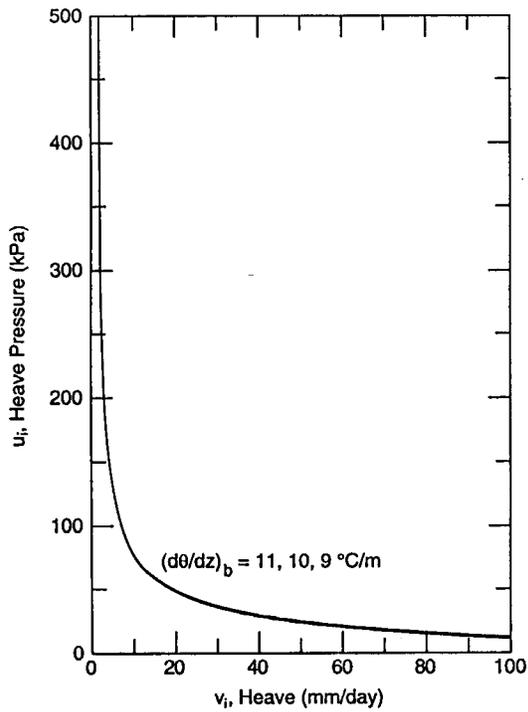
g. Penetration rate, v_b .

Figure 4 (cont'd). Changes in heave pressure vs. heave rate for changes in chosen input parameters.

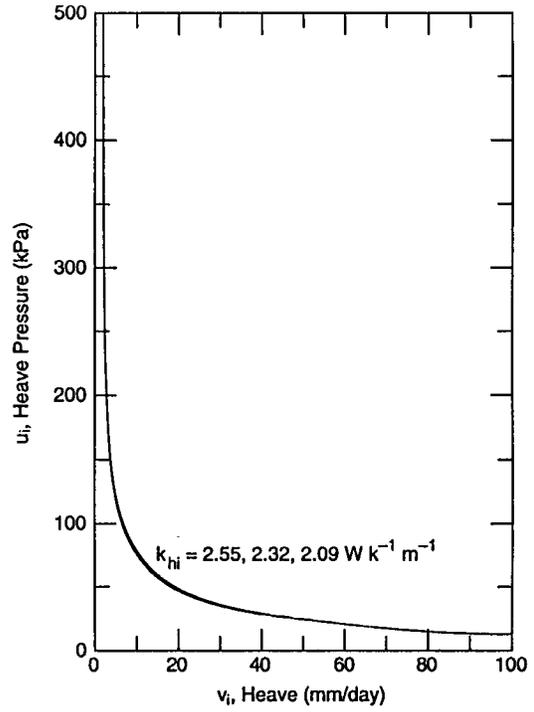
The strategy is to examine the influence that changes in the 12 chosen input parameters have on the magnitude of the predicted heaving pressure. This is accomplished by running simulations that vary the value of these parameters by 10% from the initial reference values in Table 1. The resulting pre-

dicted heaving pressures are then computed and displayed in Table 1 for direct comparison along with a series of graphs that show the overall effect on the heave pressure as a function of heave rate.

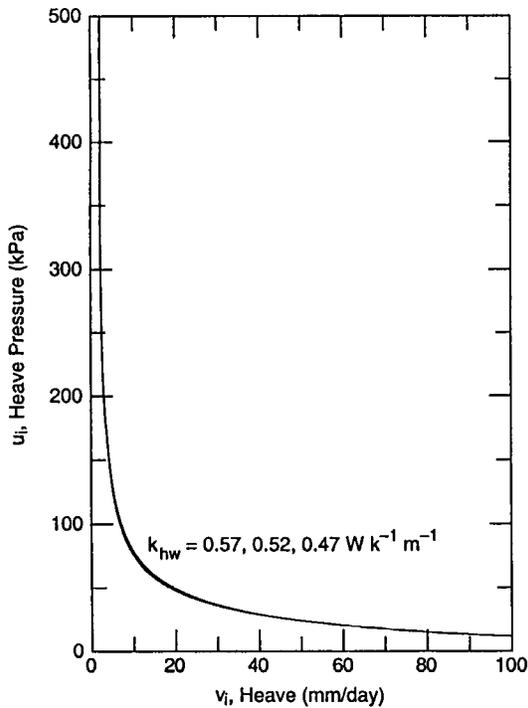
This approach is important in designing experimental tests to aid in the development of engi-



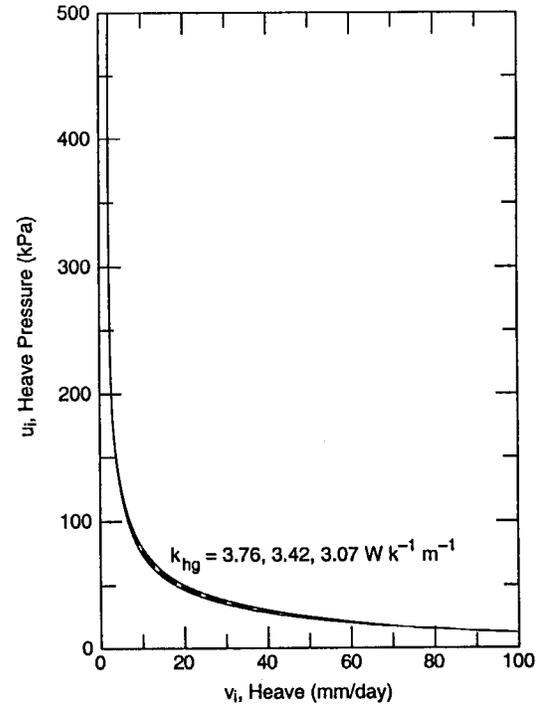
h. Temperature gradient in the unfrozen soil, $\nabla\theta$.



j. Thermal conductivity of ice, $(k_h)_i$.



i. Thermal conductivity of water, $(k_h)_w$.



k. Thermal conductivity of soil grains, $(k_h)_g$.

Figure 4 (cont'd).

neering design criteria. Variables that cause the greatest change in predicted heaving pressure are assumed to be important. The end user should therefore concentrate on obtaining correct values for these important variables if the predicted results are to be useful.

Figures 4a, b and c show how a 10% change in either direction from the reference value for the three soil water parameters influences the calculated heave pressure for a given heave rate. In either case, the resulting deviation is less than the 10% change in the control parameter. A 10% change

Table 1. Sensitivity results of calculated heave pressure to input parameter changes in RIGIDICE.

<i>Program parameter</i>	<i>Reference value</i>	<i>Calculated heave pressure (kPa)</i>		
		<i>0.9 Reference relative error*</i>	<i>Reference</i>	<i>1.1 Reference relative error*</i>
<i>Soil water properties</i>				
W_{sat}	0.42 (m ³ m ⁻³)	70.22 -7.7	76.06	82.05 7.9
W_{d}	0.02 (m ³ m ⁻³)	76.08 2.6	76.06	75.72 0.4
$(k_{\text{w}})_{\text{sat}}$	1×10 ⁻⁸ (m s ⁻¹)	72.33 -4.9	76.06	79.58 4.6
<i>Brooks and Corey constants</i>				
ϕ_{b}	11.196 (kPa)	67.94 -10.7	76.06	81.82 7.6
α	0.36	71.55 -5.9	76.06	79.86 5.0
β	2.6	109.03 43.3	76.06	56.86 -25.2
<i>Boundary conditions</i>				
v_{i}	10 (mm day ⁻¹)	81.38 7.0	76.06	71.39 -6.1
v_{b}	100 (mm day ⁻¹)	71.59 -5.9	76.06	80.24 5.5
$\nabla\theta$	-10 (°C m ⁻¹)	75.47 -0.8	76.06	76.38 0.4
<i>Thermal conductivities</i>				
$(k_{\text{h}})_{\text{w}}$	0.52 (W/K ⁻¹ m ⁻¹)	76.38 0.4	76.06	75.38 -0.9
$(k_{\text{h}})_{\text{i}}$	2.32 (W/K ⁻¹ m ⁻¹)	76.74 0.9	76.06	75.12 -1.2
$(k_{\text{h}})_{\text{g}}$	3.42 (W/K ⁻¹ m ⁻¹)	77.86 2.4	76.06	74.33 -2.3
<i>Scaling factors</i>				
λ	1.0×10 ⁻⁶ (m)			
ζ	1.0×10 ⁻² (m)			
f_{w}	1 (m s ⁻²)			
<i>Numeric settings</i>				
Precision	0.1			
Resolution	0.01			
Maximum layers	1.0×10 ⁵			

$$* \text{ relative error (\%)} = \frac{\text{value} - \text{reference}}{\text{reference}} 100$$

in the saturated water content W_{sat} resulted in the largest variation of the three, as listed in Table 1, with a relative error range of -8 to 8% . These results suggest that a 10% uncertainty in the magnitude of W_{sat} , W_{d} and $(k_w)_{\text{sat}}$ should not have a severe adverse effect on the predicted heave pressure.

The largest relative errors result from a 10% change in the Brooks and Corey constants ϕ_b and β for the soil as shown in Figures 4d and 4f. Table 1 lists the largest range of 43 to -25% relative error in the calculated heave pressure for a 10% change in β and a -10 to 8% error for ϕ_b . Clearly, this indicates that the hydrologic properties of the soil must be known if any accurate heave behavior is to be calculated. This is unfortunate since this information is rarely even collected. Figure 4e and Table 1 shows a smaller response, similar to the soil water properties, for a change in α .

Surprisingly, a 10% change in the boundary conditions also gives a similar response in relative error to heave pressure as the soil water properties. Figures 4g and h show the response for a change in penetration v_b and the temperature gradient in the unfrozen soil $\nabla\theta$. The heave rate v_i response to calculated heave pressure u_i is obtained from any of the graphs as it is always the center reference line. Again, this is unfortunate since the penetration rate and temperature gradient are easily obtained from temperature profiles. This means that more accurate temperature measurements will not necessarily result in better predictive capability.

The last group of graphs displays the response to an uncertainty in thermal conductivities. Inspection of Figures 4i, j and k along with Table 1 reveals that an uncertainty in this group of parameters has the least influence of all parameters on the calculated heave pressure.

CONCLUSIONS

The past problem of all frost heave models was the trade-off between physical correctness and ease of use. Those models that are easy to calculate tend to be based upon curve fitting heave experiments (Blanchard and Fremond 1985) or incorrectly applying phase equilibrium (Harlan 1973). The models that are physically based are difficult to implement and require time-consuming computation (O'Neill and Miller 1985). Past efforts to overcome this dilemma relied on modifying the original equations of Miller by simplifying the physics to

make the mathematics trivial (Gilpin 1980, Holden 1983) or attempting to maintain the physics while leading to a reasonable solution strategy (Black and Miller 1985). This later approach was successfully employed in this paper.

The differential equations of secondary frost heave (Miller 1978) are numerically solved in finite difference form with the program RIGIDICE. By choosing the language C++ and its object oriented nature, the core program is easily attached to the mathematical analysis program MathCad 5.0+. The ease of analysis in this new format allows simulations to be conducted in any physically correct model that the end user requires.

A brief sensitivity analysis of several important parameters shows that current practices of ground freezing monitoring are flawed. Calculated behavior for heave rate and pressure was found to be largely insensitive to penetration rate and temperature gradients. Likewise, unfrozen water content uncertainty did not strongly influence the heave rate and pressure behavior. Hydraulic conductivity, though, was found to have a dramatic influence.

These results indicate that the effort expended in making accurate temperature profiles in freezing ground is not necessary. It also indicates that the great efforts required to monitor unfrozen water content changes in the field might also be unnecessary. The sensitivity study did demonstrate the importance of the hydraulic conductivity. A wise use of research and development efforts should therefore be to develop techniques to measure the hydraulic conductivity in the laboratory and monitor it in the field.

The current model relies upon assumed standard expressions for thermal, hydraulic and stress behavior in soil. Other expressions need to be explored (i.e., van Genuchten's expression for hydraulic properties) in order to generalize the applicability of this model. The OON approach employed will make this a reasonable task. Empirical testing of this model is currently under way with a refrigerated centrifuge. This approach will test the scaling laws for freezing (Miller 1990) as well as the predictions of this model.

LITERATURE CITED

Black, P.B. and R.D. Miller (1985) A continuum approach to modeling frost heaving. *Freezing and Thawing of Soil-Water Systems, Technical Council on Cold Regions Engineering Monograph* (D.M. Ander-

- son and P.J. Williams, Ed.). American Society of Civil Engineers, p. 36–45.
- Black, P.B. and R.D. Miller** (1990) Hydraulic conductivity and unfrozen water content of airfree frozen silt. *Water Resources Research*, **26**(2): 323–329.
- Blanchard, D. and M. Fremond** (1985) Soil frost heaving and thaw settlement. *Fourth International Symposium on Ground Freezing, 5–7 August, Sapporo, Japan*.
- Brooks, R.H. and A.T. Corey** (1964) Hydraulic properties of porous media. Hydrology Paper no. 3, Civil Engineering Department, Colorado State University, Fort Collins.
- Farouki, O.T.** (1981) Thermal properties of soil. USA Cold Regions Research and Engineering Laboratory, Monograph 81-1.
- Gilpin, R.R.** (1980) A model for prediction of ice lensing and frost heave in soils. *Water Resources Research*, **16**: 918–930.
- Guymon, G.L. and J.N. Luthin** (1974) A coupled heat and moisture transport model for arctic soils. *Water Resources Research*, **10**(5): 995–1001.
- Harlan, R.L.** (1973) Analysis of coupled heat-fluid transport in partially frozen soil. *Water Resources Research*, **9**: 1314–1322.
- Holden, J.T.** (1983) Approximate solutions for Miller's theory of secondary heave. In *Proceedings, 4th International Conference on Permafrost, July 17–22, Fairbanks, Alaska*. Washington, D.C.: National Academy Press, p. 373.
- MathSoft** (1994) MathCad Plus 5.0. Cambridge, Massachusetts.
- Miller, R.D.** (1978) Frost heaving in non-colloidal soils. In *Proceedings, 3rd International Conference on Permafrost, 10–13 July, Edmonton, Alberta*. Vol. 1. Ottawa: National Research Council of Canada, p. 708–713.
- Miller, R.D.** (1990) Scaling in soil physics: principles and applications. Soil Science Society of America Special Publication 25.
- O'Neill, K. and R.D. Miller** (1985) Exploration of a rigid ice model of frost heave. *Water Resources Research*, **21**(3): 281–296.
- Ratkje, S.K., H. Yamamoto, T. Takashi, T. Ohrai and J. Okamoto** (1982) The hydraulic conductivity of soils during frost heave. *Frost I Jord*, **24**: 22–26.
- Snyder, V.A. and R.D. Miller** (1985) Tensile strength of unsaturated soils. *Soil Science Society of America Journal*, **49**: 58–65.
- van Loon, W.K.P., I.A. van Haneghem and H.P.A. Boshoven** (1988) Thermal and hydraulic conductivity of unsaturated frozen sands. In *Proceedings of the 5th International Symposium on Ground Freezing, Balkema, Rotterdam*, p. 81–90.
- Wong, M.K.W., K.G. Budge, J.S. Peery and A.C. Robinson** (1993) Object-oriented numerics: a paradigm for numerical object-oriented programming. *Computers in Physics*, **7**(6): 655–663.

APPENDIX A: RIGIDICE MODEL

RIGIDICE.H

```
// acceleration due to gravity
#define grav ((const double) 9.800) // m/sec**2
// density of liquid water
#define rhoW ((const double) 1.000E03) // kg/m**3
// density of ice
#define rhoI ((const double) 0.917E03) // kg/m**3
// viscosity of water
#define nu ((const double) 1.787E-3) // kg/m*s
// specific surface free energy of a ice-air interface
0 0 #define gammaIA ((const double) 0.100) // n/m
1
2 // Latent heat of fusion per unit volume of melt
3 #define hIW ((const double) 3.335E05) // j/m**3
4 // standard melting point of bulk ice, or its analogue, when exposed
5 // to air at standard atmospheric pressure
6 #define theta0 ((const double) 273.15) // kelvins
7 // conversions
8 // mm/day / (m/s)
9 #define mmpdToms ((const double) 8.64e7)
10 // kPa to Pa
11 #define kP2P ((const double) 1.00e3)
12
13
14 class REDUCE{
15 protected:
16     double    orglamda,
17             orgeta,
18             orgFW;
19 public:
20 // Constructor
21     REDUCE();
22     void InitScales();
23     void FirstScales(double micro, double macro, double body);
24     double    lamda;
25     double    eta;
26 //     double    body;
27     double FW;
28     double Y;
29     double L(double rSI);
30     double P(double pSI) ;
31 // reduced latent heat of fusion, per unit volume of melt
32     double H;
33     double RTHETA(double thetaSI);
34     double SIGMA(double sigmaSI);
35     double GRAD(double gradSI);
36     double F(double fSI);
37     double KW(double kwSI);
38     double V(double vSI);
```

```

39         double DIV(double divSI);
40         double T(double tSI);
41         double C(double cSI);
42         double KH(double khSI);
43         double QH(double qhSI);
44     };
45
46
47     class Tsoil{
48     public:
49
50
51
52
53
54
55
56
57
58
59         REDUCE R;
60     // saturated volumetric water content
61         double WSAT;
62     // lower limit of drying
63         double WD;
64     // reduced saturated hydraulic conductivity
65         double KWSAT;
66
67     public:
68     // Constructor
69         Tsoil();
70         void InitWATER(double wsat, double wd, double ksats);
71     // calculates water content from degree of saturation
72         double W (double DSAT);
73
74     private:
75     // volume fraction of ice
76         double I;
77     // volume fraction of soil
78         double G;
79     // thermal conductivity of water
80         double KHW;
81     // thermal conductivity of ice
82         double KHI;
83     // thermal conductivity of soil grains
84         double KHG;
85     public:
86         void InitTherm(double khw, double khi, double khg);
87     // calculates thermal conductivity for given composition
88         double RKH (double W);
89     // returns the protected reduced ice conductivity
90         double RKHI();
91
92     protected:

```

```

93 // unfrozen water content exponent
94     double ALPHA;
95 private:
96 // frozen hydraulic conductivity exponent
97     double BETA;
98 protected:
99 // reduced ice intrusion pressure
100     double PHIB;
101 public:
102     void InitBC(double alpha, double beta, double phib);
103     double DSAT(double PHI);
104     double RKW(double PHI);
105
106 // running sum
107 private:
108     double PROD;
109     int FlagSny;
110 public:
111 // initialize starting parameters
112     void InitStress();
113 // calculates Snyder's factor
114     double SNYDER (double PHI, double DSAT, double deltaDSAT);
115 };
116
117
118 // boundary conditions
119 class Tbnds
120     {
121     public: REDUCE R;
122     protected:
123         // heave rate
124     double VI, VIorg,
125         // penetration rate
126         VB, VBorg,
127         // unfrozen temperature gradient
128         GRADTB, GRADTBorg,
129         // sought after heave pressure
130         UIorg;
131
132     public:
133     void InitQuasi(double press, double pene, double gradtb);
134     void InitContinuum(double heave, double pene, double gradtb);
135     Tbnds();
136     };
137
138
139 // variables used for numeric precision
140 class Ttol
141     {
142     protected:
143     double     orgresolution,
144         orgprecision;
145
146     public:

```

```

147         double precision,      // check on Wd
148         resolution,            // layer thickness
149         LCMAX;                 // cutoff number of layers
150
151         double DPHI;
152         void InitTol();
153         void FirstTol(double prec, double resol, double lmax);
154         Ttol();
155     };
156
157
158 // variables used in calculations
159 class Trigidice:
160     public Tbnds,
161     public Ttol,
162     public REDUCE,
163     public Tsoil
164     {
165     public:
166     double    QW1, QW2, // reduced mass flux
167              DW, // reduced change in water content across layer
168              GRADUW1, GRADUW2, // reduced water pressure gradient
169              QH1, QH2, // reduced thermal flux
170              GRADTHETA1, GRADTHETA2, // reduced thermal gradient
171              PHI, // reduced phi
172              GRADPHI1, GRADPHI2, // reduced gradient of PHI from Clapeyron
173              Z, // reduced thickness
174              ZN, // reduce location of new lens
175              ZI, // reduce location of old lens
176              DZ, // reduced layer thickness
177              UW, // reduced water pressure
178              DUW, // reduced change in water pressure across layer
179              UI, // reduced ice pressure
180              CHI, // reduced stress partition factor
181              UN, // reduced neutral stress
182              UNold, // reduced previous neutral stress
183              UNmax, // reduced maximum neutral stress
184              THETA, // reduced temperature
185              DTHETA, // reduced temperature change across layer
186              WF, // unfrozen water in mature frozen zone
187              FSI;
188
189     Trigidice();
190     void LAYER(double rphi, double rdphi);
191     void PROFILE();
192     void GLOBAL();
193
194     double oldPHI;
195     double sum;
196     int iteration,
197     // error signal for GRADPHI < 0
198         GRADPROB,
199     // not enough layers

```

```
200             LAYERPROB;
201     double tic,
202     // number of layers
203     counter;
204     double maybe;
205     int check();
206     int prec();
207     void errors();
208     void once();
209     int pressdiff();
210     int slope();
211     void quasi();
212     void InitTCALC();
213     double heave();
214     double iterout();
215     double pressure();
216     double tgradout();
217     double heatin();
218     double heatout();
219     double waterin();
220     double spacing();
221     double thickness();
222     void look();
223 };
```

RIGIDICE.CPP

```
#include <iostream.h>
#include <math.h>
#include "rigidice.h"

//          REDUCED VARIABLES
// From:
// R. D. Miller (1990) Scaling of freezing phenomena,
// in Scaling in soil physics:principles and applications
// SSSA special Publication Number 25
0
0
1          REDUCE::REDUCE() {
2 // specific gravity of ice?
3          Y=rhoI/rhoW;
4          orglamda=1.0e-6;
5          orgeta=0.01;
6          FW = 1;
7          orgFW = F(FW*(-rhoW*grav));
8 // reduced latent heat of fusion, per unit volume of melt
9          H =rhoW*(lamda/gammaIA)*hIW;
10         InitScales();
11         }
12
13 void REDUCE::FirstScales(double micro, double macro, double body){
14         orglamda = micro;
15         orgeta = macro;
16         FW = body;
17         orgFW = F(FW*(-rhoW*grav));
18 // reduced latent heat of fusion, per unit volume of melt
19         H =rhoW*(lamda/gammaIA)*hIW;
20         InitScales();
21 }
22
23 void REDUCE::InitScales(){
24         lamda = orglamda;
25         eta = orgeta;
26         FW = orgFW;
27         }
28
29 // reduced length
30         double REDUCE::L(double rSI)
31                 { return ((1/lamda)*rSI);}
32 // reduced pressure
33         double REDUCE::P(double pSI)
34                 { return ((lamda/gammaIA)*pSI);}
35 // reduced temperature
36         double REDUCE::RTHETA(double thetaSI)
37                 { return ((1/theta0)*thetaSI);}
38 // reduced stress
39         double REDUCE::SIGMA(double sigmaSI)
40                 { return ((lamda/gammaIA)*sigmaSI);}
41 // reduced gradient
```

```

42         double REDUCE::GRAD(double gradSI)
43             { return (eta*gradSI);}
44 // reduced body force
45         double REDUCE::F(double fSI)
46             { return ((lamda*eta/gammaIA)*fSI);}
47 // reduced capillary conductivity
48         double REDUCE::KW(double kwSI)
49
50
51
52
53
54
55
56
57
58
59             { return ((nu/(lamda*lamda))*kwSI);}
60 // reduced velocity
61         double REDUCE::V(double vSI)
62             { return ((eta*nu/(lamda*gammaIA))*vSI);}
63 // reduced divergence
64         double REDUCE::DIV(double divSI)
65             { return (eta*divSI);}
66 // reduced time
67         double REDUCE::T(double tSI)
68             { return ((lamda*gammaIA/(eta*eta*nu))*tSI);}
69 // reduced volumetric heat capacity
70         double REDUCE::C(double cSI)
71             { return ((lamda*theta0/gammaIA)*cSI);}
72 // reduce thermal conductivity
73         double REDUCE::KH(double khSI)
74             { return ((nu*theta0/(gammaIA*gammaIA))*khSI);}
75 // reduced heat flux
76         double REDUCE::QH(double qhSI)
77             { return (eta*nu/(gammaIA*gammaIA)*qhSI);}
78
79 //
80 //
81 //
82 //             SOIL FUNCTIONS
83 //
84
85 Tsoil::Tsoil(){
86     InitWATER(0.42,0.02,1.0e-8);
87     InitBC(0.36,2.6,11.196);
88     InitTherm(0.52,2.32,3.42);
89     InitStress();
90 }
91
92 void Tsoil::InitWATER(double wsat, double wd, double ksat){
93     // saturated volumetric water content
94 //     WSAT = 0.42;
95     WSAT = wsat;

```

```

96          // lower limit of drying
97 //      WD = 0.02;
98      WD = wd;
99          // saturated hydraulic conductivity
100 //      KWSAT=1.0e-8;          // m/s
101      KWSAT = ksat;
102      KWSAT=KWSAT/(rhoW*grav);
103      KWSAT = R.KW(KWSAT);
104      }
105
106 // given degree of saturation
107 double Tsoil::W(double DSAT){
108     return( DSAT*(WSAT-WD)+WD);
109     }
110 //
111 //
112 //      Thermal conductivity
113
114 void Tsoil::InitTherm(double khw, double khi, double khg){
115 // volume fraction of soil
116     G=1-WSAT;
117 // volume fraction of ice
118     I=1-G-WD;
119 // thermal conductivity of water
120 //     KHW =0.52; // W/m
121     KHW = khw;
122     KHW = R.KH(KHW);
123 // thermal conductivity of ice
124 //     KHI =2.32; // W/m
125     KHI = khi;
126     KHI = R.KH(KHI);
127 // thermal conductivity of soil grains
128 //     KHG=3.42; // W/m
129     KHG = khg;
130     KHG = R.KH(KHG);
131 }
132
133 // calculates thermal conductivity for given composition
134 double Tsoil::RKH (double W){
135     I = 1 - G - W;
136     return ( pow(KHG,G)*pow(KHW,W)*pow(KHI,I));
137     }
138
139 // to get KHI
140 double Tsoil::RKHI(){
141     return (KHI);
142     }
143 //
144 //
145 //
146 //      Brooks & Corey equations
147 //
148 void Tsoil::InitBC(double alpha, double beta, double phib) {
149 //      ALPHA = 0.36;

```

```

150 //          BETA = 2.6;
151          ALPHA = alpha;
152          BETA = beta;
153
154          // ice entry pressure
155 //          PHIB = 11.196*kP2P; // kPa
156          PHIB = phib;
157          PHIB = R.P(PHIB*kP2P);
158          }
159
160 // returns degree of saturation for a given phi
161 double Tsoil::DSAT(double PHI) {
162         return (pow((PHIB/PHI),ALPHA));
163     }
164
165 // hydraulic conductivity for a give phi
166 double Tsoil::RKW(double PHI) {
167         // return (bc.KWSAT*pow((PHIB/PHI),BETA));
168         return (KWSAT*pow((PHIB/PHI),BETA));}
169
170
171 //
172 //          Stress partition factor
173 //
174 //
175
176         void Tsoil::InitStress(){
177 // set defaults to obtain a Snyder factor of 1
178         PROD=0.0;
179         FlagSny = 0;
180     }
181 // calculates Snyder's factor
182         double Tsoil::SNYDER (double PHI,
183                                 double DSAT,
184                                 double deltaDSAT){
185             if (FlagSny == 0) {
186                 FlagSny = 1;
187                 PROD = 0;
188                 return (1);
189             }
190                                 else{
191                 PROD = PROD + PHI*deltaDSAT;
192                 return( 0.5* (DSAT-(0.3/PHI)*PROD));
193             }
194         }
195
196
197 Tbnds::Tbnds(){
198         InitQuasi(100,10,-10);
199         InitContinuum(10,100,-10);
200     }
201
202
203 void Tbnds::InitContinuum(double heave, double pene, double gradtb){

```

```

204         VI = heave;
205         VIorg = R.V(VI/mmpdToms);
206         VI = VIorg;
207         VB = pene;
208         VBorg = R.V(VB/mmpdToms);
209         VB = VBorg;
210         GRADTB = gradtb;
211         GRADTBorg = R.GRAD(R.RTHETA(gradtb));
212         GRADTB = GRADTBorg;
213     }
214
215 void TbnDs::InitQuasi(double press, double pene, double gradtb){
216
217         UIorg = R.P(press*kP2P);
218         VBorg = R.V(pene/mmpdToms);
219         GRADTBorg = R.GRAD(R.RTHETA(gradtb));
220     }
221
222 Ttol::Ttol(){
223     FirstTol(0.1,0.1,100);
224 }
225
226 void Ttol::FirstTol(double prec, double resol, double lCmax){
227     orgprecision = prec;
228     orgresolution = resol;
229     LCMAx= lCmax;
230     InitTol();
231 }
232
233 void Ttol::InitTol(){
234     precision = orgprecision;
235     resolution = orgresolution;
236 }
237
238
239 Trigidice::Trigidice(){
240     // specific gravity of ice?
241     // Y=rhoI/rhoW;
242     // lamda=1.0e-6;
243     // eta=0.01;
244     // FW = 1.0;
245     // FW = F(FW*(-rhoW*grav));
246     // reduced latent heat of fusion, per unit volume of melt
247     // H =rhoW*(lamda/gammaIA)*hIW;
248     InitTCALC();
249 }
250
251 void Trigidice::InitTCALC(){
252     QW1=0;
253     QW2=0;// reduced mass flux
254     DW=0; // change in water content across layer
255     GRADUW1=0;
256     GRADUW2=0;// reduced water pressure gradient
257     QH1=0;

```

```

258     QH2=0;// reduced thermal flux
259     GRADTHETA1=0;
260     GRADTHETA2=0;// reduced thermal gradient
261     GRADPHI1=0;
262     GRADPHI2=0;// reduced gradient of PHI from Clapeyron
263     Z=0;// reduced thickness
264     ZN=0;//reduced location of new lens
265     ZI=0;//reduced location of old lens
266     DZ=0;// reduced layer thickness
267     UW=0;//reduced water pressure
268     DUW=0;// reduced change in water pressure
269             // across layer
270     UI=PHI;// reduced ice pressure
271     CHI=0;//reduced value of the stress partition factor
272     UN=0;// reduced neutral stress
273     UNold=0;// previous value of reduced neutral stress
274     UNmax=-1000;// maximum value of neutral stress
275     THETA=0;
276     DTHETA=0;
277     GRADPROB=0;
278     LAYERPROB=0;
279     InitStress();
280     InitTol();
281     }
282
283
284 void Trigidice::LAYER(double rphi, double rdphi){
285     DW = W(DSAT(rphi))-W(DSAT(rphi+rdphi));
286     QW2 = QW1 + (VB-Y*(VB+VI))*DW;
287     GRADUW2 = FW - QW2/RKW(rphi+rdphi);
288     QH2 = QH1 + H*(DW*VB + (QW2 - QW1));
289     GRADTHETA2 = -QH2/RKH(W(DSAT(rphi+rdphi)));
290     GRADPHI2 = (Y-1)* GRADUW2 - Y*H* GRADTHETA2;
291     if ((GRADPHI1<0)|| (GRADPHI2< 0)) {
292         GRADPROB=1;
293         DZ=1;
294     } else
295     DZ = rdphi/(sqrt(GRADPHI1*GRADPHI2));
296     DUW = FW*DZ - DZ*(QW1/RKW(rphi) +
297         QW2/(RKW(rphi+rdphi)))/2;
298     DTHETA = ((Y-1)*DUW-rdphi)/(Y*H);
299 // reset variables for beginning of next layer
300     QW1=QW2;
301     GRADUW1=GRADUW2;
302     QH1=QH2;
303     GRADTHETA1=GRADTHETA2;
304     GRADPHI1=GRADPHI2;
305     }
306
307 void Trigidice::PROFILE(){
308     LAYER(PHI,DPHI);
309     PHI=PHI+DPHI;
310     DPHI = resolution/DZ;
311     THETA=THETA+DTHETA;

```

```

312         Z=Z+DZ;
313         UW=UW+DUW;
314         UI=PHI+UW;
315         CHI=SNYDER(PHI,DSAT(PHI),DSAT(PHI+DPHI)-DSAT(PHI));
316         UN=CHI*UW+(1-CHI)*UI;
317     }
318
319 void Trigidice::GLOBAL(){
320     InitTCALC();
321     QW1 = Y*VI + ((Y-1)*(WSAT-WF))*VB;
322     GRADUW1 = FW - QW1/KWSAT;
323     QH1 = -RKH(WSAT)*GRADTB ;
324     FSI = VI/(VI+VB);
325     GRADTHETA1 = GRADTB;
326
327     GRADPHI1 = (Y-1)* GRADUW1 - Y*H* GRADTHETA1;
328     DPHI = resolution*PHIB*sqrt(GRADPHI1);
329     DZ = DPHI/GRADPHI1;
330     resolution = DZ*DPHI;
331     UW=0;
332     UI=PHI;
333     CHI=SNYDER(PHI,DSAT(PHI),DSAT(PHI+DPHI)-DSAT(PHI));
334     UN=CHI*UW+(1-CHI)*UI;
335     THETA=-PHI/(Y*H);
336 }
337
338
339
340 int Trigidice::prec(){
341
342     if (( fabs((sum/(ZI-ZN)-WF)/WD) <= precision)
343         || (LAYERPROB) || (GRADPROB)){
344         return 0;
345     }
346     else{
347         WF=sum/(ZI-ZN);
348
349         return 1;
350     }
351 }
352
353
354 int Trigidice::check(){
355
356     if (counter > LCMAX){
357         LAYERPROB=1;
358         errors();
359         iteration = -999;
360         return 0;
361     }
362     if (GRADPROB){
363         errors();
364         iteration = -888;
365         return 0;

```

```

366     }
367     if (UN > UNmax){
368         UNmax=UN;
369         ZN=Z;
370         return 1;
371     }
372     if ((UN <= UNmax)&& (UI>UNmax)){
373         sum=sum+DW*DZ;
374         ZI=Z;
375         return 1;
376     }
377
378     if ((UI<=UNmax) && (counter>=2)){
379         return 0;
380     }
381     return 1;
382 }
383
384
385 void Trigidice::errors(){
386     WF=0;
387     UI=0;
388     ZN=0;
389     ZI=1;
390     VI=0;
391 }
392
393
394 void Trigidice::once(){
395     WF=WD;
396     PHI = PHIB;
397     iteration = 1;
398     do{
399         counter = 0;
400         PHI = PHIB;
401         sum = 0;
402         GLOBAL();
403         do {
404             counter++;
405             PROFILE();
406         } while ( check());
407     } while ( prec());
408 }
409
410
411
412 int Trigidice::pressdiff(){
413
414
415     if ( fabs(UIorg-UI) <= P(precision))
416     {
417         return 0;
418     }
419     else{

```

```

420             VI=VI+VIorg;
421             }
422
423         if (/*(LAYERPROB)||*/(UI<UIorg))
424             {
425                 VI = VI - 2.0*VIorg;
426                 VIorg=VIorg/2.0;
427                 VI = VI + VIorg;
428 //                 LAYERPROB = 0;
429             }
430         if (VI<0.0)
431             {
432                 VI = V(0.5/mmpdToms);
433             }
434
435         return 1;
436
437     }
438
439 int Trigidice::slope(){
440     if(GRADPROB){
441         orgresolution=orgresolution/2;
442         LCMAX=LCMAX*2;
443         return 1;
444     }
445     else{
446         return 0;
447     }
448 }
449
450
451 void Trigidice::quasi(){
452
453     VB = VBorg;
454     GRADTB = GRADTBorg;
455     VI = V(0.5/mmpdToms);
456     VIorg = V(10/mmpdToms);
457     resolution=orgresolution;
458     do{
459         do{
460             once();
461         }while(slope());
462     } while (pressdiff());
463 }
464
465 double Trigidice::heave(){
466
467     quasi();
468     return (VI * gammaIA*lamda/(eta*nu)* mmpdToms);
469 }
470
471 double Trigidice::pressure(){
472     once();
473     return ((UI*gammaIA/lamda)/kP2P);

```

```

474     }
475
476 double Trigidice::iterout(){
477     return(iteration);
478 }
479
480 double Trigidice::tgradout(){
481     QH2 = -RKH(WSAT)*GRADTB+H*(VB*(WSAT-WF)+ Y*VI + ((Y-1)*(WSAT-WF))*VB);
482     return(-1*theta0/eta*QH2*(FSI/RKHI()+ (1-FSI)/RKH(WF)));
483 }
484 double Trigidice::heatin(){
485     return(-RKH(WSAT)*GRADTB/QH(1));
486 }
487 double Trigidice::heatout(){
488     QH2 = -RKH(WSAT)*GRADTB+H*(VB*(WSAT-WF)+ Y*VI + ((Y-1)*(WSAT-WF))*VB);
489     return(QH2/QH(1));
490 }
491
492 double Trigidice::waterin(){
493     QW1 = Y*VI + ((Y-1)*(WSAT-WF))*VB;
494     return(QW1*gammaIA*lamda/(eta*nu));
495 }
496
497 double Trigidice::spacing(){
498     return((ZI-ZN)*VI/VB * eta *1000);
499 }
500
501 double Trigidice::thickness(){
502     return(ZI * eta *1000);
503 }
504
505 void Trigidice::look(){
506     cout <<
507     iteration << ", "
508     << counter << ", "
509     << FSI << ", "
510     << (UI*gammaIA/lamda)/kP2P<< ", "
511     << VI * gammaIA*lamda/(eta*nu)* mmpdToms << ", "
512     << VB * gammaIA*lamda/(eta*nu)* mmpdToms << ", "
513     << GRADTB*theta0/eta << ", ";
514     QW1 = Y*VI + ((Y-1)*(WSAT-WF))*VB;
515     QH1 = -RKH(WSAT)*GRADTB ;
516     QH2 = QH1+H*(VB*(WSAT-WF)+QW1);
517 // fout << -1.0*theta0/eta*QH2*(VI*RKH(WF)+VB*RKH(0))/((VI+VB)*(RKH(0)*RKH(WF)))
518     cout <<-1.0*theta0/eta*QH2*(FSI/RKHI()+ (1-FSI)/RKH(WF))
519     << ", "
520     << QW1*gammaIA*lamda/(eta*nu) << ", "
521     << (ZI-ZN)*VI/VB * eta *1000 << ", "
522     << ZI * eta *1000
523     << "\n";
524 }
525

```

HEAVE.CPP

```
1 #include "mcadincl.h"
2 #include "rigidice.h"
3
4
5     LRESULT RIGIDICEFunction(     LPCOMPLEXARRAY     calculated,
6                                     LPCOMPLEXARRAY settings);
7
8
9     FUNCTIONINFO     RIGIDICE =
10    {
11        "RIGIDICE",           // Name by which mathcad will recognize the function
12        "settings",         // heaverate will be called as heavepressure(settings)
13        "pressure, heat and mass fluxes and lens sizes for given settings", //
14        description of heavepressure(settings)
15        (LPCFUNCTION)RIGIDICEFunction, // pointer to the executable code
16        COMPLEX_ARRAY, // the return type is a complex scalar
17        1, // the function takes on 1 argument
18        { COMPLEX_ARRAY} // that is an array
19    };
20
21    LRESULT RIGIDICEFunction(     LPCOMPLEXARRAY     calculated,
22                                    LPCOMPLEXARRAY settings)
23    {
24
25        Trigidice     foo;
26        double transient;
27        double     micro, macro, body;
28        double     khw, khi, khg;
29        double     wsat, wd, ksat;
30        double     alpha, beta, phib;
31        double     prec, resol, lcmax;
32        double     heave, pene, gradtb;
33
34        micro = settings->hReal[0][0];
35        macro = settings->hReal[0][1];
36        body = settings->hReal[0][2];
37        foo.FirstScales( micro, macro, body);
38
39        khw = settings->hReal[0][3];
40        khi = settings->hReal[0][4];
41        khg = settings->hReal[0][5];
42        foo.InitTherm( khw, khi, khg);
43
44        wsat = settings->hReal[0][6];
45        wd = settings->hReal[0][7];
46        ksat = settings->hReal[0][8];
47        foo.InitWATER( wsat, wd, ksat);
48
49        alpha = settings->hReal[0][9];
50        beta = settings->hReal[0][10];
```

```

51     phib = settings->hReal[0][11];
52     foo.InitBC( alpha, beta, phib);
53
54     heave = settings->hReal[0][12];
55     pene = settings->hReal[0][13];
56     gradtb = settings->hReal[0][14];
57     foo.InitContinuum( heave, pene, gradtb);
58
59     prec = settings->hReal[0][15];
60     resol = settings->hReal[0][16];
61     lcmax = settings->hReal[0][17];
62     foo.FirstTol( prec, resol, lcmax);
63
64     MathcadArrayAllocate( calculated, 1, 8, TRUE, FALSE);
65
66     transient = foo.pressure();
67     calculated->hReal [0] [0] = transient;
68     transient = foo.tgradout();
69     calculated->hReal [0] [1] = transient;
70     transient = foo.heatin();
71     calculated->hReal [0] [2] = transient;
72     transient = foo.heatout();
73     calculated->hReal [0] [3] = transient;
74     transient = foo.waterin();
75     calculated->hReal [0] [4] = transient;
76     transient = foo.spacing();
77     calculated->hReal [0] [5] = transient;
78     transient = foo.thickness();
79     calculated->hReal [0] [6] = transient;
80     transient = foo.iterout();
81     calculated->hReal [0] [7] = transient;
82
83
84
85     return 0;                // return 0 to indicate there was no error
86
87 }
88
89
90 LRESULT heavepressureFunction(      LPCOMPLEXSCALAR    pressure,
91                                  LPCOMPLEXARRAY settings);
92
93
94 FUNCTIONINFO    heavepressure =
95 {
96     "heavepressure",                // Name by which mathcad will recognize the function
97     "settings",                      // heaverate will be called as heavepressure(settings)
98     "heave pressure for given settings", // description of heavepressure(settings)
99     (LPCFUNCTION)heavepressureFunction, // pointer to the executable code
100    COMPLEX_SCALAR,                  // the return type is a complex scalar
101    1,                                // the function takes on 1 argument
102    { COMPLEX_ARRAY}                 // that is an array
103 };
104

```

```

105
106     LRESULT heavepressureFunction(          LPCOMPLEXSCALAR    pressure,
107                                         LPCOMPLEXARRAY settings)
108     {
109
110         Trigidice    foo;
111         double HeavePressure;
112         double    micro, macro, body;
113         double    khw, khi, khg;
114         double    wsat, wd, ksat;
115         double    alpha, beta, phib;
116         double    prec, resol, lcmax;
117         double    heave, pene, gradtb;
118
119         micro = settings->hReal[0][0];
120         macro = settings->hReal[0][1];
121         body = settings->hReal[0][2];
122         foo.FirstScales( micro, macro, body);
123
124         khw = settings->hReal[0][3];
125         khi = settings->hReal[0][4];
126         khg = settings->hReal[0][5];
127         foo.InitTherm( khw, khi, khg);
128
129         wsat = settings->hReal[0][6];
130         wd = settings->hReal[0][7];
131         ksat = settings->hReal[0][8];
132         foo.InitWATER( wsat, wd, ksat);
133
134         alpha = settings->hReal[0][9];
135         beta = settings->hReal[0][10];
136         phib = settings->hReal[0][11];
137         foo.InitBC( alpha, beta, phib);
138
139         heave = settings->hReal[0][12];
140         pene = settings->hReal[0][13];
141         gradtb = settings->hReal[0][14];
142         foo.InitContinuum( heave, pene, gradtb);
143
144         prec = settings->hReal[0][15];
145         resol = settings->hReal[0][16];
146         lcmax = settings->hReal[0][17];
147         foo.FirstTol( prec, resol, lcmax);
148
149         HeavePressure = foo.pressure();
150         pressure->real = HeavePressure;
151
152
153         return 0;                // return 0 to indicate there was no error
154
155     }
156
157     LRESULT heaverateFunction(          LPCOMPLEXSCALAR    heaverate,
158                                         LPCOMPLEXARRAY settings);

```

```

159
160
161     FUNCTIONINFO     heaverate =
162     {
163     "heaverate",           // Name by which mathcad will recognize the function
164     "settings",          // heaverate will be called as heavepressure(settings)
165     "heave rate for given settings", // description of heavepressure(settings)
166     (LPCFUNCTION)heavepressureFunction, // pointer to the executable code
167     COMPLEX_SCALAR,      // the return type is a complex scalar
168     1,                   // the function takes on 1 argument
169     { COMPLEX_ARRAY}     // that is an array
170     };
171
172
173     LRESULT heaverateFunction(          LPCOMPLEXSCALAR     heaverate,
174                                     LPCCOMPLEXARRAY settings)
175     {
176
177         Trigidice     foo;
178         double HeaveRate;
179         double  micro, macro, body;
180         double  khw, khi, khg;
181         double  wsat, wd, ksat;
182         double  alpha, beta, phib;
183         double  prec, resol, lcmax;
184         double  press, pene, gradtb;
185
186         micro = settings->hReal[0][0];
187         macro = settings->hReal[0][1];
188         body = settings->hReal[0][2];
189         foo.FirstScales( micro, macro, body);
190
191         khw = settings->hReal[0][3];
192         khi = settings->hReal[0][4];
193         khg = settings->hReal[0][5];
194         foo.InitTherm( khw, khi, khg);
195
196         wsat = settings->hReal[0][6];
197         wd = settings->hReal[0][7];
198         ksat = settings->hReal[0][8];
199         foo.InitWATER( wsat, wd, ksat);
200
201         alpha = settings->hReal[0][9];
202         beta = settings->hReal[0][10];
203         phib = settings->hReal[0][11];
204         foo.InitBC( alpha, beta, phib);
205
206         press = settings->hReal[0][12];
207         pene = settings->hReal[0][13];
208         gradtb = settings->hReal[0][14];
209         foo.InitQuasi( press, pene, gradtb);
210
211         prec = settings->hReal[0][15];
212         resol = settings->hReal[0][16];

```

```

213         lcmax = settings->hReal[0][17];
214         foo.FirstTol( prec, resol, lcmax);
215
216         HeaveRate = foo.heave();
217         heaverate->real = HeaveRate;
218
219
220         return 0;           // return 0 to indicate there was no error
221     }
222
223
224     BOOL WINAPI DllEntryPoint (HANDLE hDLL, DWORD dwReason, LPVOID lpReserved)
225     {
226         switch (dwReason)
227         {
228             case DLL_PROCESS_ATTACH:
229             {
230
231                 // DLL is attaching to the address space of the current process.
232                 //
233
234                 if ( CreateUserFunction( hDLL, &RIGIDICE ) == NULL )
235                     break;
236
237                 if ( CreateUserFunction( hDLL, &heaverate ) == NULL )
238                     break;
239                 if ( CreateUserFunction( hDLL, &heavepressure ) == NULL )
240                     break;
241             }
242             //         CreateUserFunction( hDLL, &heavepressure );
243
244             case DLL_THREAD_ATTACH:      // A new thread is being created in the current process.
245             case DLL_THREAD_DETACH:     // A thread is exiting cleanly.
246             case DLL_PROCESS_DETACH:    // The calling process is detaching the DLL from its address
space.
247
248                 break;
249         }
250         return TRUE;
251     }
252
253

```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestion for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE May 1995		3. REPORT TYPE AND DATES COVERED	
4. TITLE AND SUBTITLE RIGIDICE Model of Secondary Frost Heave				5. FUNDING NUMBERS PE: 6.27.84A PR: 4A762784AT42 TA: BS WU: CP-C01	
6. AUTHORS Patrick B. Black					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Cold Regions Research and Engineering Laboratory 72 Lyme Road Hanover, New Hampshire 03755-1290				8. PERFORMING ORGANIZATION REPORT NUMBER CRREL Report 95-12	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of the Chief of Engineers Washington, D.C. 20314-1000				10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited. Available from NTIS, Springfield, Virginia 22161				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) A revised version of an earlier attempt to numerically solve Miller's equations for the RIGIDICE model of frost heave is presented that corrects earlier mistakes and incorporates recent improvements in the scaling factors of ground freezing. The new version of the computer code also follows the concepts of Object Oriented Numerics (OON), which allow for easy modification and enhancements. Analysis of the program is accomplished with the symbolic math program MathCad. A brief sensitivity analysis of the input variables indicates that those parameters that calculate the hydraulic conductivity have the greatest influence on the variability of predicted heaving pressure.					
14. SUBJECT TERMS Frost heave Frozen soil				15. NUMBER OF PAGES 41	
Hydraulic conductivity Stress				16. PRICE CODE	
Thermal conductivity					
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL		