

FUNDAMENTALS OF FINITE DIFFERENCES

- **Key elements of numerical transport model;**
- **Derivation of finite difference approximations;**

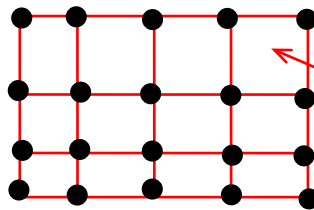
Excellent General Reference:

Smith, G.D. Numerical solution of partial differential equations: finite difference methods. 1978. Clarendon Press.

How does a numerical flow model work?

Numerical models solve the flow equation for head or pressure (or water content) by:

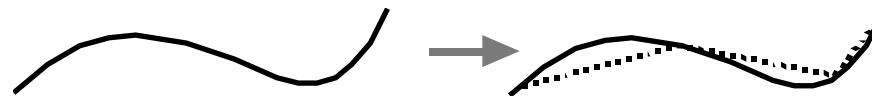
1. Subdividing the flow region into ‘n’ finite blocks in a mesh of grid points (discretization), such that different K values can be assigned to each block and the “∂’s” can be converted to finite differences (Δ’s).



finite difference mesh with nodes (grid points)

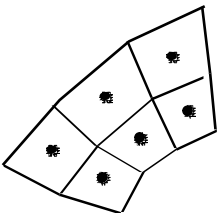
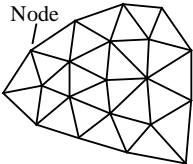
2. Writing the flow equation in algebraic form (using finite differences or finite elements) for each node or block. I.e. write continuous form into discrete form:

$$\text{e.g., } \left(K(h) \frac{\partial h}{\partial x} \right) \rightarrow \left(K(h) \frac{\Delta h}{\Delta x} \right)$$



3. Using “numerical methods” to solve the resulting n equations in n unknowns for h, subject to boundary and initial conditions.

Ways to Subdivide the Model Domain

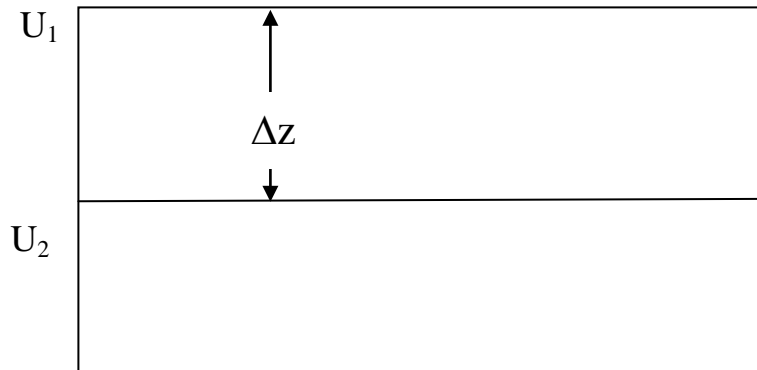
	Advantages	Disadvantages
Finite Difference Node	Simple and intuitively clear; easy to program and apply.	Complex geometries and variable orientation of anisotropy difficult to handle.
Integrated Finite Difference 	Simple; complex geometries handled readily. The method combines the advantages of an integral formulation with the simplicity of finite difference gradients and is very convenient for handling multidimensional heterogeneous systems .	Large quantity of input data required for mesh.
Finite Element Node 	Good for complex geometry with intricate geologic architecture; angle of anisotropy is arbitrary.	More difficult to understand; fairly large input data requirements; sometimes computationally slower. Uses interpolation functions between the grid points.

Solution of flow or transport equations, using numerical modeling (soil water, solute, gas, heat)

- Define grid points (one, two or three-dimensional) within vadose zone domain.

Number of grid points will vary, but number of grid points is partly controlled by need to describe soil spatial variations and time dynamics of flow; Note that model error will generally increase as number of grid points decrease. This is because the distance between the grid points will determine the accuracy of the derivative approximations.

Variable U could be soil water pressure head, solute concentration, or soil temperature.



- Compute steady state flux (J) from values of soil water potential, temperature, solute concentration (etc) at layer boundaries (nodes or grid points)

$$J = -K \frac{du}{dz} = -K \frac{\Delta u}{\Delta z} = -K \frac{u_1 - u_2}{\Delta z}$$

ODE

where $\Delta u/\Delta z$ denotes the gradient in concentration or potential and K defines a soil factor, relating gradient to flux (hydraulic conductivity, diffusion coefficient, or thermal conductivity);

- Spatial resolution and dynamics is increased by increasing the number of grid points/ layers in the spatial domain, thereby increasing the number of nodes at which u- and J- values are computed;
- U- or J- values at the boundaries of the spatial domain are defined as boundary conditions and must be known to solve flow problem;
 1. Dirichlet boundary condition – head or concentration
 2. Neumann boundary condition – flux (derivative boundary condition)
 3. Mixed boundary condition – combination of (1) and (2)

- Time dependency of transport process comes about from boundary condition changes, resulting in storage (S) changes:

$$\frac{\partial S}{\partial t} = -\frac{\partial J}{\partial z} + \text{Sinc} / \text{Source} \quad \mathbf{PDE}$$

which is approximated by:

$$\frac{\Delta S}{\Delta t} = -\frac{\Delta J}{\Delta z} + \text{Sinc} / \text{Source}$$

for each pair of grid points, while assuming steady-state flow within between grid points for each time step Δt .

- Modeling of dynamics of vadose zone transport processes, requires solution of transient transport equations, with size of Δt controlling time resolution

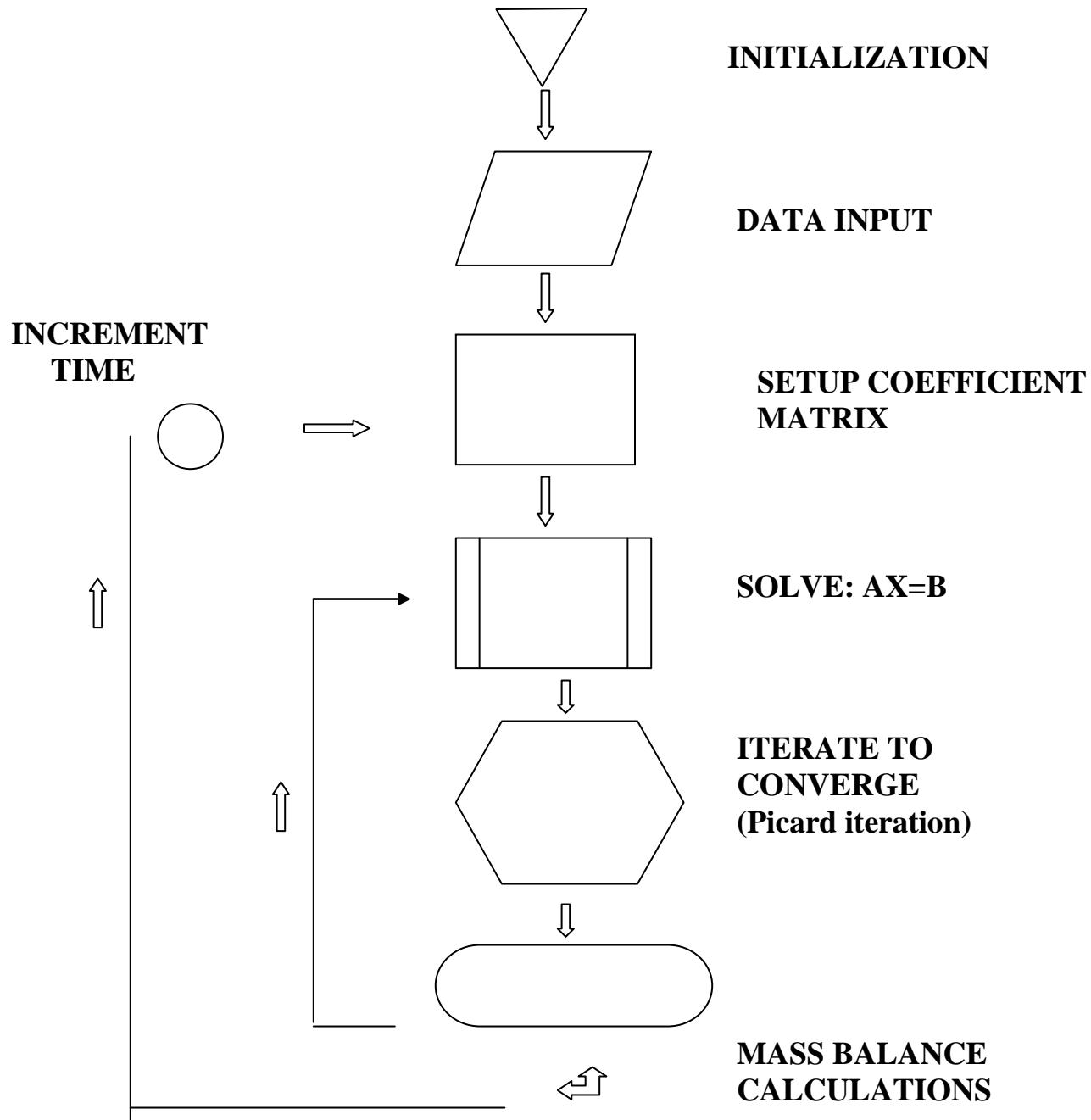
In addition to boundary conditions, solution of transient transport problems require knowledge of the initial conditions (at $t = 0$);

- Computer algorithm to solve flow/transport equation depends on:
 - Type of partial differential equation
 - Steady state or transient problem
 - Size of spatial domain
 - Required accuracy : **Mass Balance**
- In some cases, analytical solutions might be available;

The word "HYDRUS" is written in a bold, serif font and is enclosed within a rounded rectangular border.

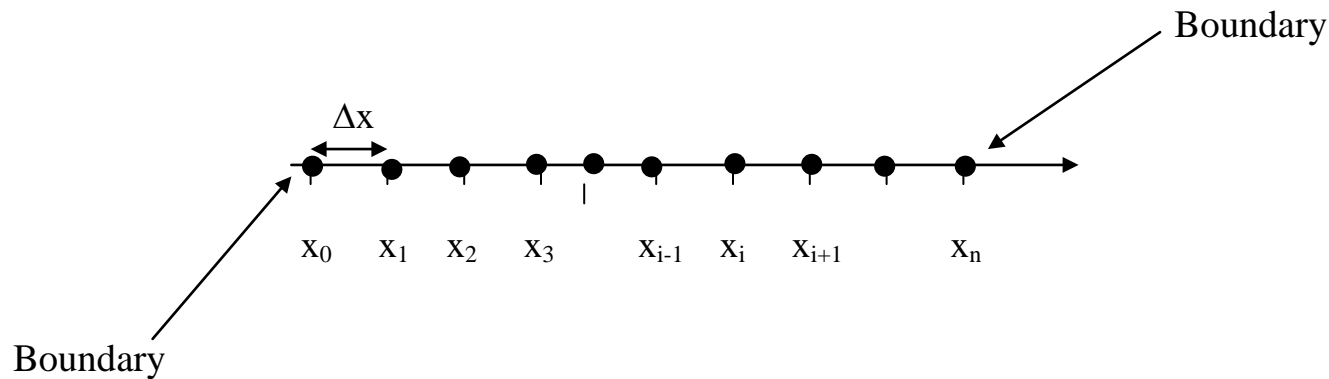
HYDRUS

Key elements of a numerical transport model:



FINITE DIFFERENCES APPROXIMATIONS

- Represent time-space domain by a set of discrete points.
For each point, one writes a separate algebraic equation as an approximation to the partial differential equation (pde), solving for the variable $u(x,t)$. The solution is found by solving a set of N equations for each of the N grid points
- Define a set of points \Rightarrow mesh, grid points or nodes, e.g. only consider space, and define the space variable, x , for each of the nodes:



- Divide x-domain into a set of equally-spaced points, so that

$x_i = i (\Delta x)$ and $u(x_i)$ is value of dependent variable at x_i , we want to solve for, for example, what is the soil temperature or soil water pressure (h) at each x_i :

i	x_i	x	u_i
0	X_0	0	u_0
1	X_1	Δx	u_1
2	X_2	$2\Delta x$	u_2
$i-1$	X_{i-1}	$(i-1)\Delta x$	u_{i-1}
i	X_i	$i\Delta x$	u_i
$i+1$	X_{i+1}	$(i+1)\Delta x$	u_{i+1}
...			
n	X_n	$(n)\Delta x$	u_n

- Approximate the space derivative of u_i (**go from continuous to discrete variables**):

Use Taylor series expansion about neighboring points, i.e. expand $u(x+\Delta x)$ about (x) , e.g.,

$$u(x + \Delta x) = u(x) + \frac{\partial u}{\partial x} \Delta x + \frac{\partial^2 u}{\partial x^2} \frac{(\Delta x)^2}{2!} + \frac{\partial^3 u}{\partial x^3} \frac{(\Delta x)^3}{3!} + \dots$$

Or:

$$u_{i+1} = u_i + \left. \frac{\partial u}{\partial x} \right|_i \Delta x + \left. \frac{\partial^2 u}{\partial x^2} \right|_i \frac{(\Delta x)^2}{2} + \left. \frac{\partial^3 u}{\partial x^3} \right|_i \frac{(\Delta x)^3}{6} + \dots \quad [1]$$

and

$$u_{i-1} = u_i - \left. \frac{\partial u}{\partial x} \right|_i \Delta x + \left. \frac{\partial^2 u}{\partial x^2} \right|_i \frac{(\Delta x)^2}{2} - \left. \frac{\partial^3 u}{\partial x^3} \right|_i \frac{(\Delta x)^3}{6} + \dots \quad [2]$$

From [1], one can get a first order approximation of the **first derivative** (**discrete**):

$$\frac{\partial u}{\partial x}|_i = \frac{u_{i+1} - u_i}{\Delta x} + O(\Delta x) \quad [3]$$

where $O(\Delta x)$ includes terms containing first and higher powers of Δx , and signifies **the truncation error**.

- Equation [3] is a first order approximation of du/dx , and as it involves a function value at the next point (u_{i+1}), it is defined as a **forward difference** approximation.

From [2], one can also derive the so-called **backward difference** approximation:

$$\frac{\partial u}{\partial x}|_i = \frac{u_i - u_{i-1}}{\Delta x} + O(\Delta x) \quad [4]$$

- Moreover, subtracting [2] from [1], one obtains:

$$u_{i+1} - u_{i-1} = 2\Delta x \frac{\partial u}{\partial x}|_i + O(\Delta x)^3$$

yielding

$$\frac{\partial u}{\partial x}|_i = \frac{u_{i+1} - u_{i-1}}{2\Delta x} + O(\Delta x)^2 \quad [5]$$

which is defined as the **central difference approximation** for the first derivative.

Note that the truncation error is of order $(\Delta x)^2$

How about second derivative ?

We use the so-called centered difference approximation as well:

Add Eqs. [1] and [2]:

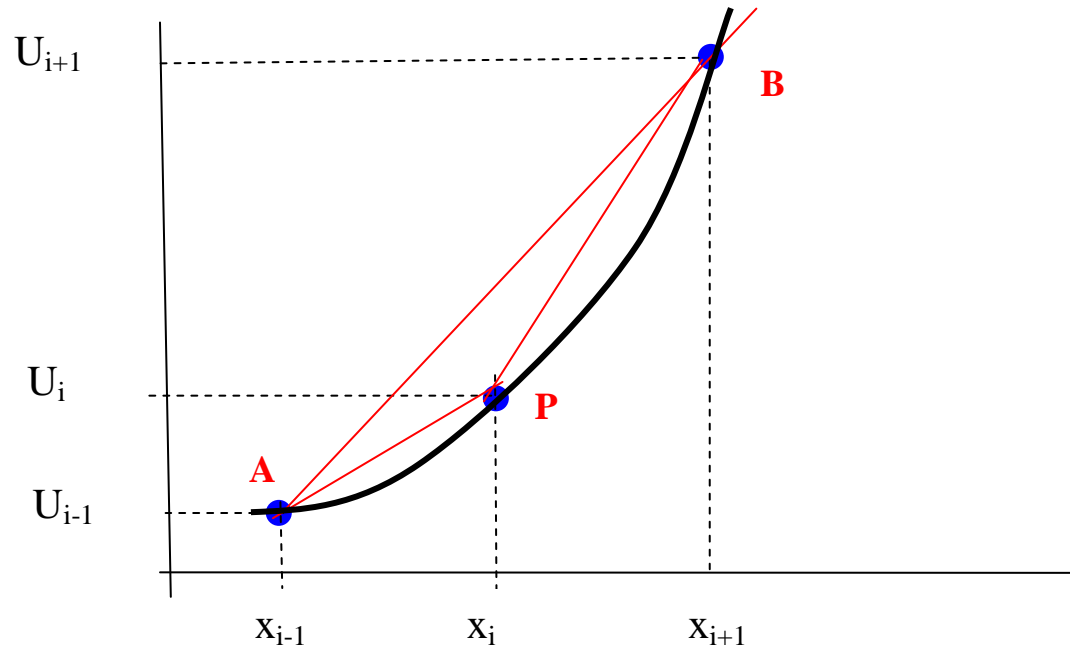
$$u_{i+1} + u_{i-1} = 2u_i + \frac{\partial^2 u}{\partial x^2} \Big|_i (\Delta x)^2 + O(\Delta x)^4$$

Yielding:

$$\frac{\partial^2 u}{\partial x^2} \Big|_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} + O(\Delta x)^2 \quad [6]$$

- Graphical representation of first derivative approximations:

Graphical representation of first derivative approximations at point P:



Forward difference: approximate slope at P by slope of chord PB
Backward difference: approximate slope at P by slope of chord AP

Central difference: approximate slope at P by slope of chord AB

Note: approximations are improved if Δx becomes smaller

- Next step: Include time as second independent variable, and approximate the first derivative, where $u_{i,j}$ denotes the value of the variable u , at position x_i and time, t_j

$$\frac{\partial u}{\partial t}|_{i,j} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t} \quad [7]$$

- Example: simple heat conduction equation along a metal rod of length L :

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} \quad [8]$$

The variable α denotes thermal diffusivity (ratio of thermal conductivity and heat capacity) of rod and u is temperature. Discretize space-time domain (i =space, j = time): or $u_{i,j}$ = temperature u at grid i and at time j

Define \mathbf{u} , for $j=0$: initial condition ($t=0$): $u(x,0) = \dots$ (bottom of domain, $i = 0$)

Define \mathbf{u} , for $i=0$ and $i= L$: boundary conditions:

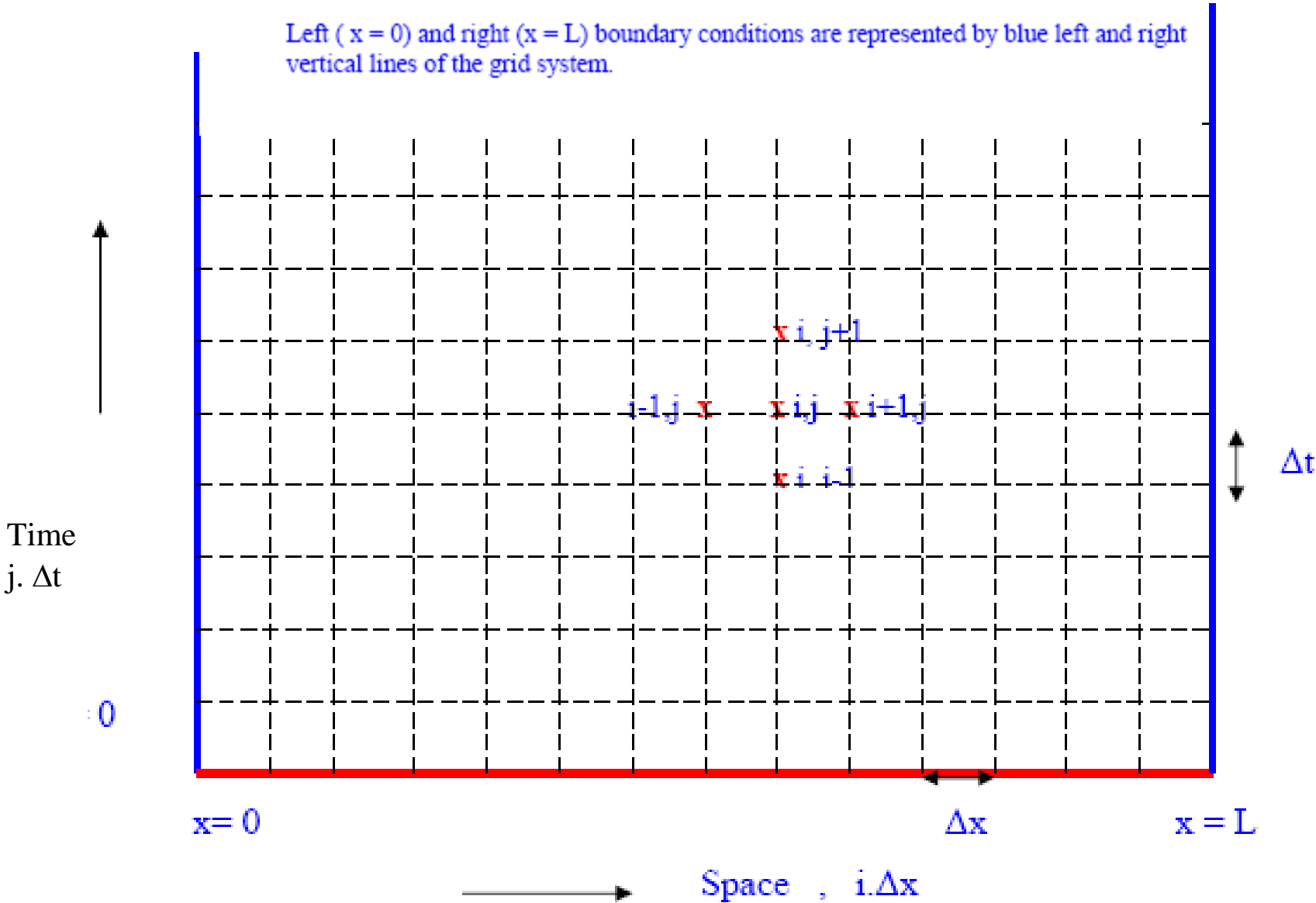
$$u(0,t) = \dots \quad (x = 0)$$

$$u(L,t) = \dots \quad (x = L)$$

DEVELOP MESH THAT REPRESENTS MODEL DOMAIN BY GRID POINTS OR NODES:

Initial Condition: $t = 0$, represented by red bottom horizontal line.

Left ($x = 0$) and right ($x = L$) boundary conditions are represented by blue left and right vertical lines of the grid system.



Substitute finite difference approximations in pde:

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \alpha \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} \quad \text{and set} \quad r = \frac{\alpha \Delta t}{(\Delta x)^2} \quad [9]$$

and solve for $u_{i,j+1}$:

$$\mathbf{u_{i,j+1} = r(u_{i+1,j}) + r(u_{i-1,j}) + (1-2r)u_{i,j} \quad [10]}$$

At $t=0$, $u_{i,j}$ is known ($i = 1, \dots, N$) **INITIAL CONDITION**

Advance to time $t=1 \cdot \Delta t$ ($j=1$) by employing [10] and solve for $u_{i,j+1}$

March forward in time by subsequently solving for $u_{i,j+1}$
($j=2, i=1, \dots, N$)



EXPLICIT SOLUTION

Solution is simple, but time-consuming.
Solution is stable and converges only if:

$$0 \leq r < 0.5$$

Size of Δx must be small to minimize truncation error of finite difference approximation,

Hence, to satisfy stability criterion: Δt must be **extremely** small.

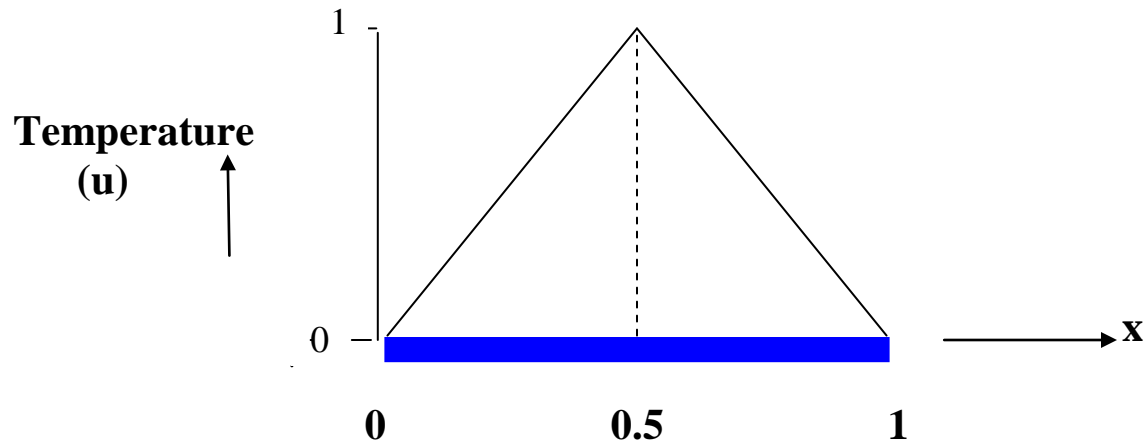
Example:

Heat conduction in a one-dimensional solid rod of length 1, with

- temperature (T) at endpoint is zero: $u(0,t)$ and $u(1,t)=0$
- initial temperature at midpoint is 1 and is linearly decreasing towards zero at endpoints:

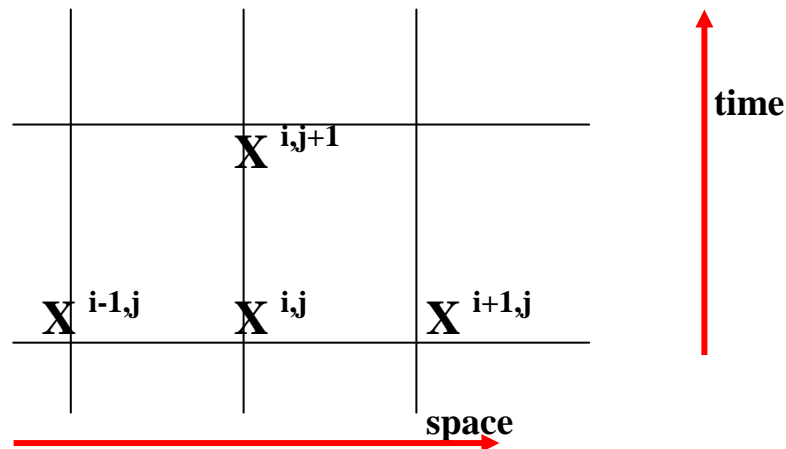
$$u(x,0) = 2x, \quad 0 < x < 0.5$$

$$u(x,0) = 2(1-x), \quad 0.5 < x < 1$$



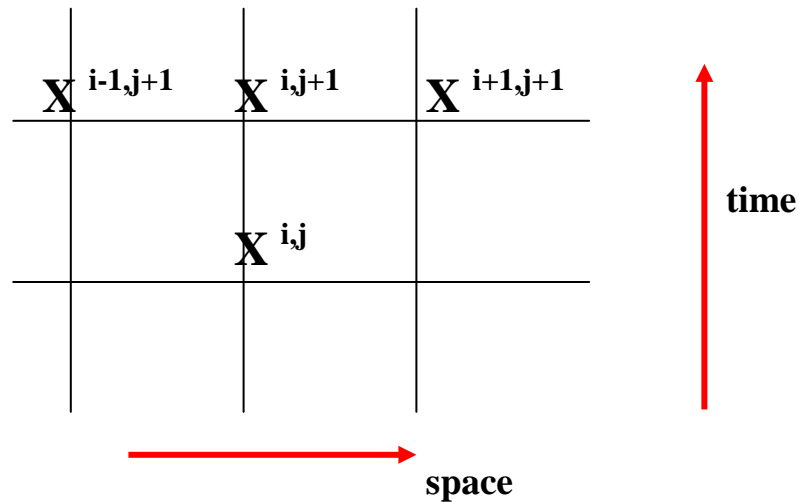
Question: How does temperature in bar change with time, if the heat source in the center of the bar is removed ?

Computational cell for explicit solution:



Alternative Solution: Use Implicit Method (Computational cell):

You do not have to worry about stability criterion



$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = K \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{(\Delta x)^2} \quad \text{and set } r = \frac{\alpha \Delta t}{(\Delta x)^2}$$

PUT ALL UNKNOWNNS (j+1 time level) TO ONE SIDE

Then, for each node i , we have the following equation:

$$r \cdot u_{i-1,j+1} - (2r+1)u_{i,j+1} + r \cdot u_{i+1,j+1} = -u_{i,j} \quad ,$$

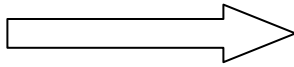
which contains three unknowns (but only one equation)

Write down for each node, and put in matrix form: $Au = b$:

BOUNDARY CONDITION

$$\begin{array}{|c|} \hline \begin{array}{ccccc} -(2r+1) & r & 0 & 0 & 0 \\ r & -(2r+1) & r & 0 & 0 \\ 0 & r & -(2r+1) & r & 0 \\ 0 & 0 & r & -(2r+1) & 0 \\ \text{etc, one row for} \\ \text{each node} \end{array} \\ \hline \end{array} \cdot \begin{array}{|c|} \hline u_1 \\ u_2 \\ u_3 \\ u_4 \\ \cdot \\ \cdot \\ (j+1) \\ \hline \end{array} = \begin{array}{|c|} \hline -u_{1,j} - ru_{0,j+1} \\ -u_{2,j} \\ -u_{3,j} \\ -u_{4,j} \\ \cdot \\ \cdot \\ \hline \end{array}$$

- Solve system of equations, and solve for all unknown u -values at $(j+1)$ time step simultaneously.
- Requires inversion of the matrix A . Time-consuming, and only works if $\det(A)$ is not close to 0.



IMPLICIT METHOD

Algorithm to be used if each row in matrix A contains a maximum of 3 non-zero entries, is the

TRIDIAGONAL (THOMAS) ALGORITHM

- So far it has been assumed that K (coefficients in matrix A) are constant and independent of U

However, in flow and transport problems in the unsaturated zone (HYDRUS), where U may denote soil water potential (h) or concentration (c), the coefficients such as K (unsaturated hydraulic conductivity) may be dependent on U , and is not constant.

- In that case, one applies **PICARD ITERATION**:

Within a time step, the system of equations ($AX=b$) is solved iteratively, while updating the coefficients with each solution, until differences between iterations within the time step is smaller than a pre-defined criterion.

- This method is used to solve

Water flow equation (Richards equation)

Solute transport (CDE) and gaseous diffusion equation

In General, one can write the second order derivative in a general finite difference form:

$$\frac{d^2u}{dx^2} = (1-\gamma) \left[\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} \right] + (\gamma) \left[\frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{(\Delta x)^2} \right]$$

γ = time weighting factor

$\gamma = 0$: explicit method

$\gamma = 1$: fully implicit method

$\gamma = 1/2$: **Cranck-Nicolson method**: Unconditionally stable for all time and space steps.

The finite difference solution is unconditionally stable and convergent if:

1. $1/2 \leq \gamma \leq 1$
2. Only if $r \leq \frac{1}{2(1-2\gamma)}$ for $0 \leq \gamma < 1/2$

OTHER SOLUTION ALGORITHM'S

- Point-iterative methods: (Jacobi, Gauss-Seidel, SOR method)
- To solve elliptic equation: steady-state ground water flow:

$$\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} = 0$$

where the piezometric head at all 4 boundaries is known, and the solution gives the head distribution within the spatial domain (soil profile)

$$\frac{\partial^2 h}{\partial x^2} \Big|_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2}$$

$$\frac{\partial^2 h}{\partial y^2} \Big|_{i,j} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2}$$

- Substitute in pde, and solve for $h_{i,j}$:

$$h_{i,j} = \frac{h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1}}{4}$$

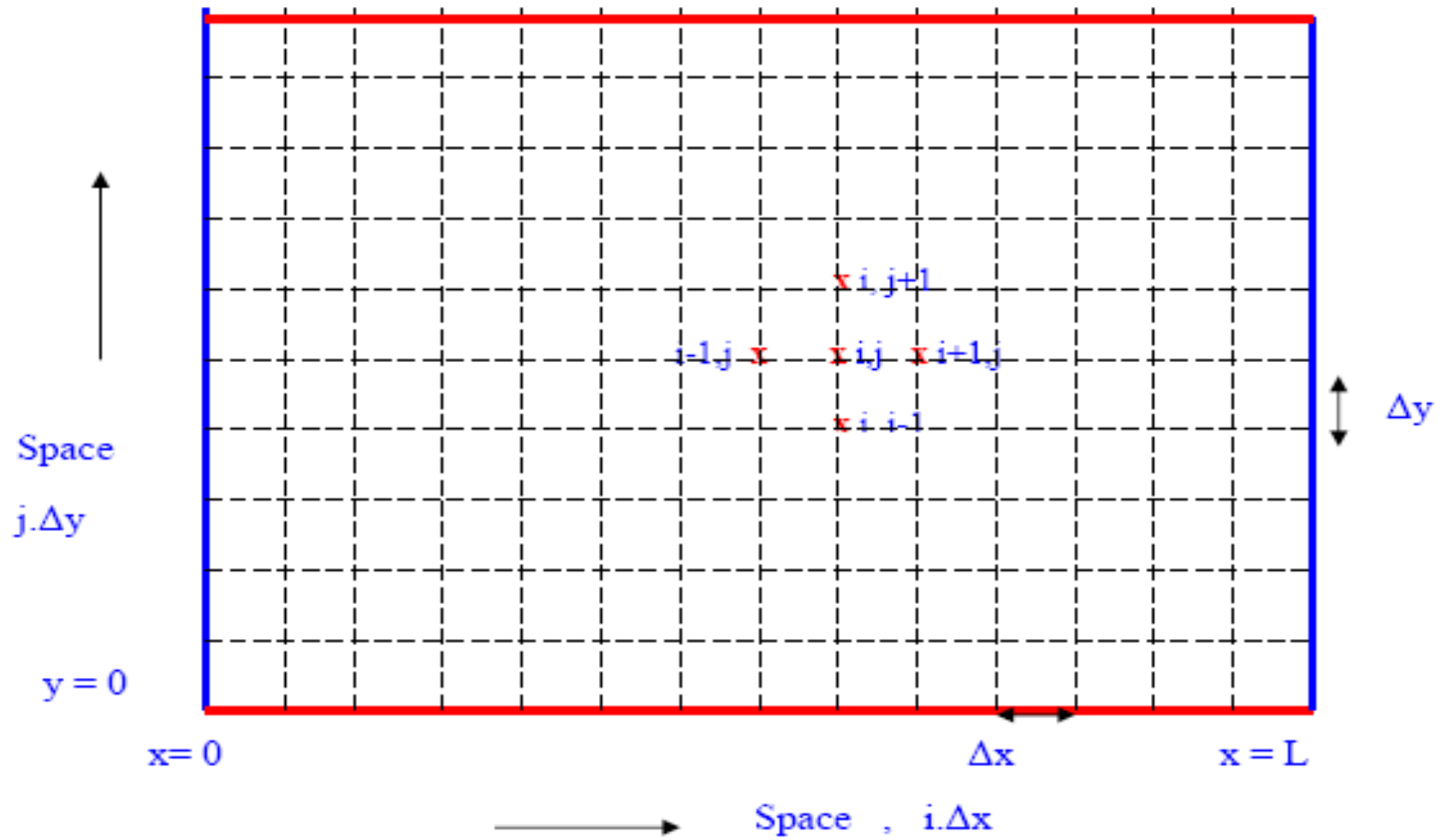
- Start in left hand, bottom corner and solve for $h_{i,j}$ for the bottom row, using some initial estimate for each nodal value;

Move one row up, and continue the same way;

In this way, solve for $h_{i,j}$ for the whole spatial domain;

Start over again at left hand, bottom corner, and repeat the whole process, resulting in many iterations. Repeat iterations until largest differences between nodal values after successive iterations become smaller than a predetermined accuracy value. If that is achieved, the solution is said to have **converged**.

Let $\Delta x = \Delta y$. Moreover, we set boundary conditions at all 4 boundaries.



Three techniques with increasing efficiency:

I. Jacobi Method:

Let (m) denote the iteration index:

$$h_{ij}^{(m+1)} = \frac{h_{i-1,j}^m + h_{i+1,j}^m + h_{i,j-1}^m + h_{i,j+1}^m}{4}$$

Start with initial set of trial values. Solution can be done in any order. Use boundary conditions to make intuitive guess, although not necessary. One could take any initial value, however, the number of iterations will be reduced if the the initial values are closer to the final result.

II. Gauss-Seidel Method (GS):

Instead, go through the grid system in an orderly way. Sweep from left to right and then up and down. Always update the new values, so that within each iteration, one consistently uses two newly computed values for each grid point:

$$h_{ij}^{(m+1)} = \frac{h_{i-1,j}^{m+1} + h_{i+1,j}^m + h_{i,j-1}^{m+1} + h_{i,j+1}^m}{4}$$

It is more efficient as Gauss Seidel will use less iterations, and converges faster (about twice).

III. Successive Over-relaxation Method (SOR)

It is generally the fastest method. It is sometimes also called an extrapolated iteration formula.

Let $\tilde{h}_{i,j}^{m+1}$ be the Gauss Seidel solution. We introduce a relaxation parameter, ω , and solve for $h_{i,j}^{m+1}$:

$$h_{ij}^{m+1} = h_{ij}^m + \omega(\tilde{h}_{ij}^{m+1} - h_{ij}^m) , \text{ which becomes the Gauss Seidel method for } \omega = 1.$$

In SOR, one always selects $1 < \omega < 2$, thus extrapolating the previous GS solution. Sometimes SOR may overshoot the true solution and would cause more iterations. The more cautious choice would be to select $0 < \omega < 1$, however, this would be an interpolation between iterations, causing more iterations than GS. Problem is how to find optimal value of ω . For fixed problem, one can use trial and error method.

In general SOR leads to:

$$h_{ij}^{(m+1)} = h_{ij}^m(1-\omega) + \omega \cdot \frac{h_{i-1,j}^{m+1} + h_{i+1,j}^m + h_{i,j-1}^{m+1} + h_{i,j+1}^m}{4}$$

SAME PRINCIPLES APPLY FOR LARGER SPATIAL DIMENSIONS (TWO OR THREE-DIMENSIONAL); IT JUST TAKES MORE COMPUTING TIME

READ CHAPTER 4 IN TEXT.

THURSDAY, WE WILL REVIEW HEAT FLOW THEORY

**APPLY HYDRUS TO SOLVE FOR UNSATURATED HEAT
TRANSPORT IN SOILS**

**ASSIGNMENT 3: APPLY FINITE DIFFERENCE
TECHNIQUE TO SOLVE FOR HEAT FLOW**