Partial Differential Equations (PDEs) and the Finite Difference Method (FDM). An introduction

FILE:Chap 13 Partial Differential Equations-V6. Original: May 7, 2015
Revised: Dec 19, 2016, Feb 20, 2017, Mar 1, 2017, Apr 11,2018, Oct 21, 2018

A partial differential equation (PDE) is an equation stating a relationship between a function of two or more independent variables and the partial derivatives of this function with respect to these independent variables.

Typical Example	Simplified Notation	Common Name	Type
$\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2}=0$	$f_{xx}+f_{yy}=0$	Laplace Equation	Elliptic
$\frac{\partial f}{\partial t}=\propto \frac{\partial^2 f}{\partial x^2}$	$f_t=\propto f_{xx}$	Diffusion Equation	Parabolic
$\frac{\partial^2 f}{\partial t^2}=c^2\frac{\partial^2 f}{\partial x^2}$	$f_{tt}=c^2 f_{xx}$	Wave Equation	Hyperbolic

Characterization of Second Order PDEs

Most PDEs in engineering and science are second order. Second order PDEs take the general form

$$\boldsymbol{A\,u_{xx}+2B\,u_{xy}+C\,u_{yy}+\cdots=0}$$

where A, B and C are coefficients that may depend on x and y. These PDEs fall into one of the following categories:

$B^2-AC<0$: Elliptic PDE

$B^2-AC=0$: Parabolic PDE

$B^2-AC>0$: Hyperbolic PDE

There are some specific strategies for some categories. There are specialized solvers for some types of PDEs, hence knowing its category can be useful for solving a PDE.

The FDM

The main feature of the finite difference method (FDM) is to obtain discrete equations by replacing derivatives and other elements within the equation with appropriate finite divided differences and discrete approximations. We derive and solve a finite difference system of equations for a PDE in five steps.

1. Discretization of the domain of the problem
2. Discretization of the differential equation at the interior nodes
3. Discretization of the boundary conditions.

Revised:April18, 2018

4. Construction of the system of equations and/or the matrix equation.
5. Solving the equations

Many times the matrix equation is tridiagonal and the solution of tridiagonal linear systems is a very well-studied problem. Other times we do not need to solve a matrix as the solution can be obtained explicitly marching one step at a time on the independent variable.

Parabolic Equations. The Heat Conduction Equation in One-Dimension

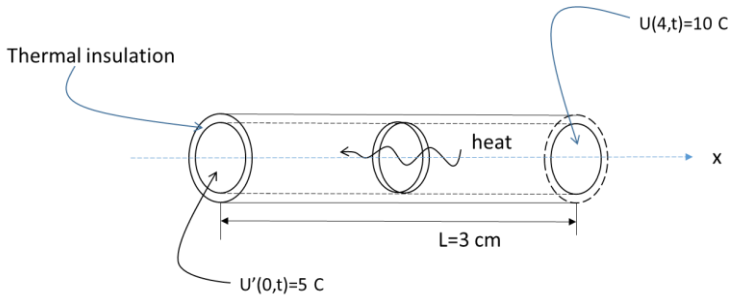
One of the simplest PDEs to learn the numerical solution process of FDM is a parabolic equation, e.g., the heat conduction equation in one dimension:

$$\frac{\partial U}{\partial t} = k\frac{\partial^2 U}{\partial x^2} \quad [Eq\ 1]$$

where U [temperature], t [time], x [space], and k [thermal diffusivity]

The 'Toy' Problem—A Quick-Example

Consider the parabolic PDE known as the Heat Equation above as the equation governing the temperature distribution in a thin rod insulated at all points, except at its ends (this means heat exchange only happens at the rod ends



with the following specifications

- L = 3 cm (rod length)
- Δx = 1 cm, Δt = 0.1 s, k = 0.835 cm^2/s (thermal diffusivity)
- λ = $k\Delta t/\Delta x^2$=0.835(0.1)/(1)^2 = 0.0835 (this parameter shows later alongside the discretization of the ODE)

At t = 0, the temperature of interior nodes of the rod is zero and the boundary conditions are fixed for all times. These can be expressed as

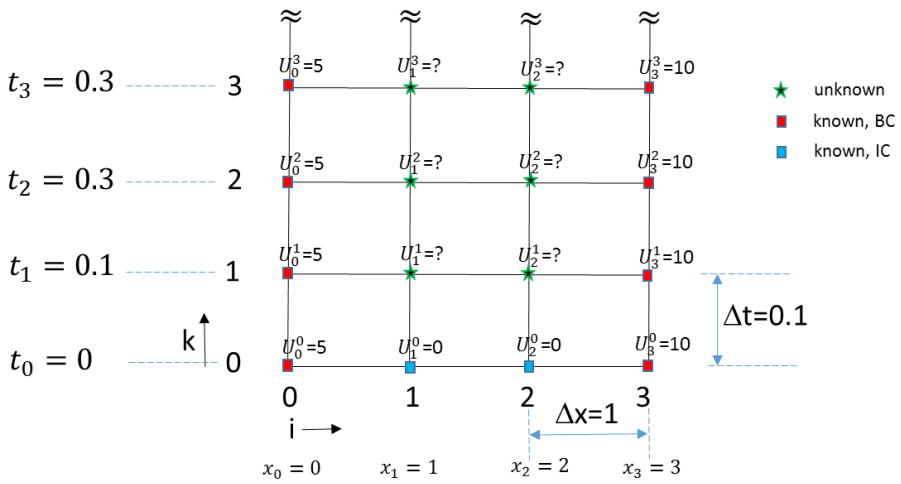
- U(x,0) = 0 °C for 0<x<3
- U(0,t) = 5 °C for t>0
- U(3,t) = 10 °C for t>0

What makes this problem a 'toy' is the small number of nodes to be solved.

<u>All three of the following methods below refer to the same problem description above</u>

Discretization is the process of transferring continuous functions, models, variables, and equations into discrete counterparts. This process is usually carried out as a first step toward making them suitable for numerical evaluation and implementation on digital computers. Whenever continuous data is discretized, there is always some amount of discretization error. The goal is to reduce the amount to a level considered negligible for the modeling purposes at hand.

Discretize the domain of the problem. For two independent variables use a mesh (also called a grid)—a mesh is a net that is formed by connecting nodes in a predefined manner. Grid points are typically arranged in a rectangular array of nodes. Each axis represents one of the independent variables. Nodes are labeled by indices, **i** for space and **k** for time, e.g., i=0,1,2,… ix, and k=0,1,2,…,kx. A detailed discretization is shown below:



Discuss: A good discretization should include a cartoon representation of the grid (nodes), the step size, and indexed nodes.

- At a given time level (a row in your grid), how many times the computational molecule fits into the grid?
- How many unknowns must be solved at each given time level? How many equations do you need?
- Regarding the discrete equations is there any difference if the running index starts at 0 or at 1? What is the range of the running index for your discretization in both cases?
- What specific indexed variables correspond to boundary conditions (BCs)? How the indices change if they start at 0 or 1?
- How do you deal with the 'corners' of the grid?

Next is the description of the three main methods used for numerical solution of the heat equation

Forward in Time, Centered in Space (FTCS) Solution. This method is also called forward Euler method, this time applied to PDEs. Using the domain discretization described above, the steps toward the solution are:

(1) Discretize the differential equation

a. Approximate the 1st order time derivative with forward finite difference approximation of $\vartheta(\Delta t)$ at the grid point *(i,k)*:

$$\left.\frac{\partial U}{\partial t}\right|_{i,k} = \frac{U_i^{K+1} - U_i^K}{\Delta t} \quad [Eq\ 2]$$

The indices (*i,k*) at the LHS derivative operator means the discretization is at the node *(i,k).* As this is a time derivative, the space *i* index remain constant. The derivative approximation can be obtained directly from the 'lazy solver' tables.

b. Approximate the 2nd order space derivative with central difference approximation of $\vartheta(\Delta x^2)$ at the grid point *(i,k)* using the 'lazy solver' tables:

$$\left.\frac{\partial^2 U}{\partial x^2}\right|_{i,k} = \frac{U_{i+1}^K - 2U_i^K + U_{i-1}^K}{\Delta x^2} \quad [Eq\ 3]$$

The time *k* index remains constant as this is a space derivative.

c. Plug [Eq 2] and [Eq 3] in the original equation [Eq 1] to get the recurrence formula:

$$\frac{U_i^{K+1} - U_i^K}{\Delta t} = k\frac{U_{i+1}^K - 2U_i^K + U_{i-1}^K}{\Delta x^2} \quad [Eq\ 4]$$

which after simplification yields the recurrence formula:

$$U_i^{K+1} = U_i^K + \lambda\left(U_{i+1}^K - 2U_i^K + U_{i-1}^K\right) \quad [Eq\ 4.a]$$

where $\lambda = k\Delta t/\Delta x^2$ Clustering variables with same indices in ascending order offers another way to express the recurrence formula which is more suitable for a computer program:

$$U_i^{K+1} = \lambda U_{i-1}^K + (1-2\lambda)U_i^K + \lambda U_{i+1}^K \quad [Eq\ 4.b]$$

The recurrence formula is expanded for all the interior nodes on the rod (the 'unknown' nodes) by properly setting values of *i* and *K*. It then provides an explicit means to compute values at each node for a future time (*K+1*) based on the present values (*K*) and its neighbors. Notice that this approach is actually an extension of Euler's method for solving ODEs at several nodes in space. That is, if we know the temperature distribution as a function of position at an initial time, we can compute the distribution at a future time based on Eq 4.b.

(2) Discretize the initial and boundary conditions (i.e., express the discrete variables involved with right indices):

Original Expression	Discretized
$U(0,t) = 5$ for t>0	$U_0^k = 5$ for $k \geq 0$ $(i.e.,\ k = 0,1,2,3,\dots)$
$U(3,t) = 10$ for t>0	$U_3^k = 10$ for $k \geq 0$ $(i.e.,\ k = 0,1,2,3,\dots)$
$U(x,0) = 0$ for 0<x<3	$U_i^0 = 0$ for $0 < i < 3$ $(i.e,\ i = 1,2)$

(3) Show the FTCS computational molecule within the grid of your discretization (see discretization above)

(4) **Develop the equations to be solved** According to the problem discretization and the number of unknowns, and using the recurrence formulation (Eq 4.b), find the appropriate number of equations for the solution at the first and second time steps (two rows of nodes). Please use algebraic expressions with indexed variables, NO numbers yet:

For the first time step [t_1=0.1], the equations are (i.e., Eq 4a yields):

i=1, k=0:

$$U_1^1 = U_1^0 + \lambda(U_2^0 - 2U_1^0 + U_0^0),$$

i=2, k=0:

$$U_2^1 = U_2^0 + \lambda(U_3^0 - 2U_2^0 + U_1^0),$$

For the second time step [t_2=0.2], the equations are (i.e., Eq 4a yields):

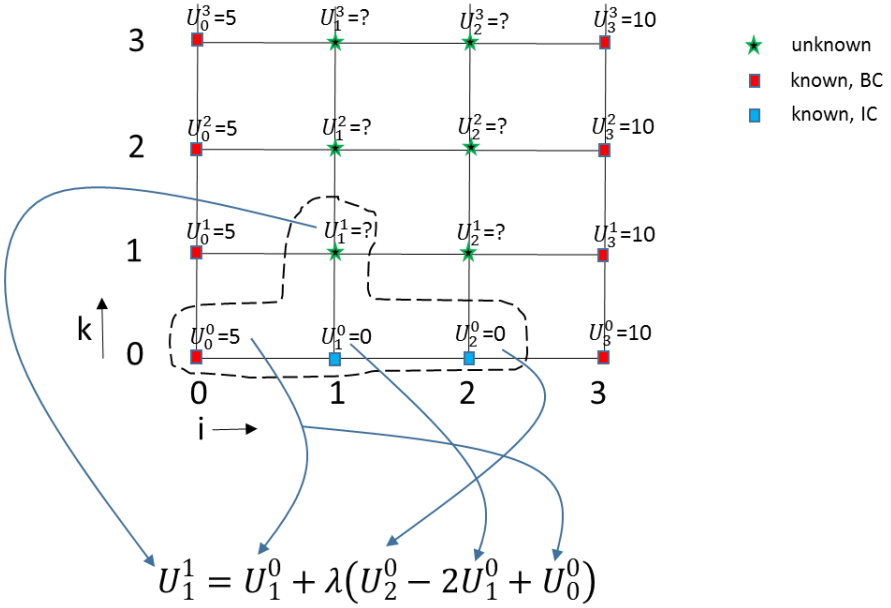
i=1, k=1:

$$U_1^2 = U_1^1 + \lambda(U_2^1 - 2U_1^1 + U_0^1)$$

i=2, k=1:

$$U_2^2 = U_2^1 + \lambda(U_3^1 - 2U_2^1 + U_1^1)$$

Where these equations are applied in the grid?



Discuss:

- Locate each U_i^k in the above equations within its computational molecule and the grid.
- How do you give specificic values to *i*- and *K*-indices in the recurrence formula to arrive to the correct equations (e.g., above). Realize, that after given the correct values for *i* and *K*, each

U_i^K in the equation is represented in the computational molecule (CM) , this is each element is bounded, i.e., no U_i^K is outside the CM. For instance, study the figure above for the first equation above (i.e., U_1^1):

(5) Plugging numbers into the equations developed in (4), and solve for two time steps (i.e.,[t_1 ,U_1], and [t_2 ,U_2]:
For the first time step [t_1=0.1]:

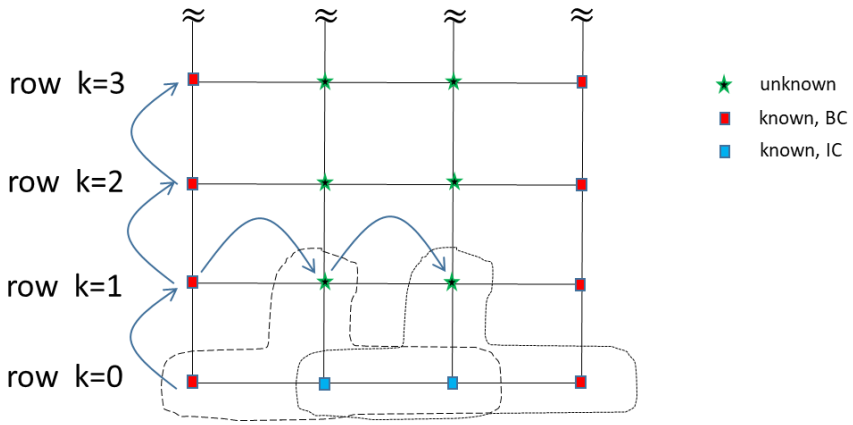
$$U_1^1 = U_1^0 + \lambda(U_2^0 - 2U_1^0 + U_0^0) = 0 + 0.0835(0 - 2(0) + 5) = 0.4175$$

$$U_2^1 = U_2^0 + \lambda(U_3^0 - 2U_2^0 + U_1^0) = 0 + 0.0835(10 - 2(0) + 0) = 0.835$$

For the second time step [t_2=0.2]:

$$U_1^2 = U_1^1 + \lambda(U_2^1 - 2U_1^1 + U_0^1) = 0.4175 + 0.0835(0.835 - 2(0.4175) + 5) = 0.835$$

$$U_2^2 = U_2^1 + \lambda(U_3^1 - 2U_2^1 + U_1^1) = 0.835 + 0.0835(10 - 2(0.835) + 0.4175) = 1.565416$$



Computation starts in the second row (i.e. row index k=1, for each interior nodes label i=1,2) which is equivalent to all nodes at t_1; left to right in this time level. Next, computation marches to the next time level up (i.e. row index k=2, interior nodes i=1,2) which is equivalent to all nodes at t_2; left to right in this time level, and so on each time step, row-by-row, until the end of the domain.

(6) Express the results on a table (critical step). For example for the FTCS computation we can use Excel (see the file: FDM with Excel):

		x(0)=0	x(1)=1	x(2)=2	x(3)=3
	K\i ==>	0	1	2	3
t(0)=0	0	5	0	0	10
t(1)=0.1	1	5	0.4175	0.835	10
t(2)=0.2	2	5	0.835	1.565416	10
t(3)=0.3	3	5	1.243767	2.208714	10
t(4)=0.4	4	5	1.637986	2.778714	10

Because computations in Excel are usually top to down, the computational molecule is 'up-side-down' when compared to our original mesh.

(7) **Life Saver Matlab code**. A better tool is a matlab code (see Appendix one).

(8) **Convergence & Stability**: *Convergence* means that as Δx and Δt approach zero, the results of the finite-difference technique approach the true solution. *Stability* means that errors at any stage of the computation are not amplified but are attenuated as the computation progresses.

It can be shown that the explicit method is both convergent and stable if: $\lambda = \kappa\Delta t/\Delta x^2 \leq 1/2$ or $\Delta t \leq \frac{1}{2}\frac{\Delta x^2}{k}$ (the later means you can't march the solution at times steps greater than $\frac{1}{2}\frac{\Delta x^2}{k}$)

Crank Nicolson (CN) Method. Now we deal with the CN method as the more accurate and popular method used to solve this type of equations. Use the same problem domain discretization as above.

(1) Discretization of the Differential equation. This very clever approach discretizes the PDE at $\left(i, k+\frac{1}{2}\right)$.

a. Approximate the 1st order time derivative with central finite difference approximation of $\vartheta(\Delta t^2)$ at the grid point $\left(i, k+\frac{1}{2}\right)$:

$$\left.\frac{\partial U}{\partial t}\right|_{i,k+1/2} = \frac{U_i^{K+1} - U_i^K}{\Delta t}$$

Appendix two shows the 'tricky' derivation of this expression, which deserves the main contribution of this method.

b. To evaluate $\left.\frac{\partial^2 U}{\partial x^2}\right|_{i,k+1/2}$ we use the average of the centered difference approximations of the 2nd order derivative at both time steps t_{k+1} and t_k

$$\left.\frac{\partial^2 U}{\partial x^2}\right|_{i,k+\frac{1}{2}} = \frac{1}{2}\left\{\left[\frac{U_{i-1}^{k+1} - 2U_i^{k+1} + U_{i+1}^{k+1}}{\Delta x^2}\right] + \left[\frac{U_{i-1}^{k} - 2U_i^{k} + U_{i+1}^{k}}{\Delta x^2}\right]\right\}$$

c. Alternatively we can do a weighted average with the θ factor which yields the following:

$$\frac{U_i^{k+1} - U_i^k}{\Delta t} = \theta\left[\frac{U_{i-1}^{k+1} - 2U_i^{k+1} + U_{i+1}^{k+1}}{\Delta x^2}\right] + (1-\theta)\left[\frac{U_{i-1}^{k} - 2U_i^{k} + U_{i+1}^{k}}{\Delta x^2}\right]$$

d. The recurrence formula with the usual $\theta = 1/2$ and $\lambda = \frac{\kappa\Delta t}{\Delta x^2}$ is:

$$-\left(\frac{1}{2}\right)\lambda U_{i-1}^{k+1} + (1+\lambda)U_i^{k+1} - \left(\frac{1}{2}\right)\lambda U_{i+1}^{k+1} = \left(\frac{1}{2}\right)\lambda U_{i-1}^{k} + (1-\lambda)U_i^{k} + \left(\frac{1}{2}\right)\lambda U_{i+1}^{k}$$

the above equation times 2, simplifies to:

$$-\lambda U_{i-1}^{k+1} + 2(1+\lambda)U_i^{k+1} - \lambda U_{i+1}^{k+1} = \lambda U_{i-1}^{k} + 2(1-\lambda)U_i^{k} + \lambda U_{i+1}^{k}$$

NOTE: You don't need to rename coefficients as they are simple enough.

Where the coefficients for the above equation are (these will be used later):

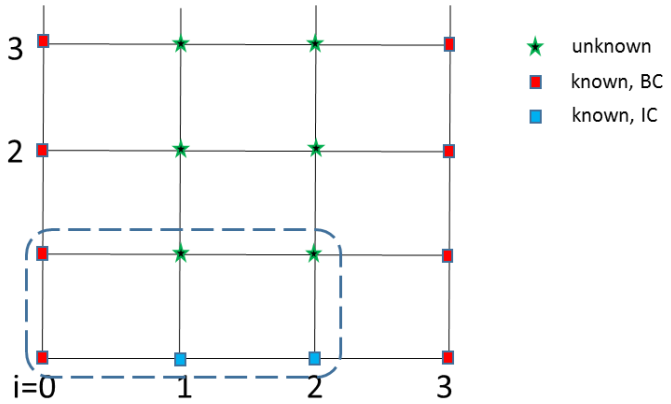
$$2(1+\lambda) = 2(1+0.0835) = 2.167$$

$$2(1-\lambda) = 2(1-0.0835) = 1.833$$

$$\lambda = 0.0835$$

(2) Discretize the initial and boundary conditions (This is the same as FTCS method).

(3) Show the CN computational molecule within the grid of your discretization.



(4) According to your problem discretization and the number of unknowns find the appropriate number of equations to solve for the first time step. We use indexed-variable algebraic equations—Not numbers yet:

For the time t_1, there are two unknowns, u_1^1 and u_2^1, therefore, we need two equations to solve them.

Set *i=1* and *k=0* in the recurrence formula:

$$-\lambda U_0^1 + 2(1+\lambda)U_1^1 - \lambda U_2^1 = \lambda U_0^0 + 2(1-\lambda)U_1^0 + \lambda U_2^0$$

Set i=2 and k=0 in the recurrence formula:

$$-\lambda U_1^1 + 2(1+\lambda)U_2^1 - \lambda U_3^1 = \lambda U_1^0 + 2(1-\lambda)U_2^0 + \lambda U_3^0$$

Notice again that with these indices we produced equations with the two unknowns U_1^1 and U_2^1at each time step, while the rest of $U_i^K$are all known.

As the two terms containing $U_0^1 = 5$ and $U_3^1 = 10$ in the first and second equations above are knowns and constant along the BCs, you transfer them to the RHS of the equation:

$$2(1+\lambda)U_1^1 - \lambda U_2^1 = \lambda U_0^0 + 2(1-\lambda)U_1^0 + \lambda U_2^0 + \lambda U_0^1$$

$$-\lambda U_1^1 + 2(1+\lambda)U_2^1 = \lambda U_1^0 + 2(1-\lambda)U_2^0 + \lambda U_3^0 + \lambda U_3^1$$

(5) Express the problem with numbers for a clean system of linear equations and also in matrix form:

$$2.167U_1^1 - 0.0835U_2^1 = 0.0835(5) + 1.833(0) + 0.0835(0) + 0.0835(5) = 0.835$$

$$-0.0835U_1^1 + 2.167U_2^1 = 0.0835(0) + 1.833(0) + 0.0835(10) + 0.0835(10) = 1.67$$

The system of equations:

$$2.167U_1^1 - 0.0835U_2^1 = 0.835$$

$$-0.0835U_1^1 + 2.167U_2^1 = 1.67$$

The matrix equation becomes:

$$\begin{bmatrix} 2.167 & -0.0835 \\ -0.0835 & 2.167 \end{bmatrix} \begin{bmatrix} U_1^1 \\ U_2^1 \end{bmatrix} = \begin{bmatrix} 0.835 \\ 1.67 \end{bmatrix}$$

(6) Solve the system of linear equations.

The current 2x2 system of equations, can be solved by <u>many</u> methods. For instance, by substitution, we solve by U_1^1 from the first equation and replaced it into the second equation, next we solve by U_2^1. However, this is a 'toy' problem and in real life problems, matrices are many times larger. We are doing this example to learn the procedure and be able to apply it to any matrix size, therefore be smart and run a simpler MATLAB script:

```
% CN-solution with matlab
% File

clc, clear
A=[2.167,-0.0835;-0.0835,2.167];
b=[0.835;1.67];
u=A\b
```

OUTPUT

```
u =

    0.4156
    0.7867
```

IMPORTANT: The solution above represents only the interior nodes $U_1^1$and U_2^1 By adding the constant BCs, the complete solution vector is then obtained, u = [5, 0.4156, 0.7867, 10]. This last step is a critical one. Many times students forget to express the whole solution.

(9) As engineers likes to organize their results, then expressing results in a table is a good practice. So far the Excel worksheet table should look like

k\ii	0	1	2	3
0	5	0	0	10
1	5	**0.4156**	**0.7867**	10
2	5			10
3	5			10
...	...			...

We have just solved the PDE for one time step. This process should be repeated for more time steps until satisfying the requested time domain.

(10) Life Saver Matlab code. A better tool is a matlab code (see Appendix Three).
(11) Please review another example in the document: CRANK-Example with MATLAB code (DOC) with a solution that uses the efficient Thomas Algorithm via a Triadiagonal Solver.

Backward in Time, Centered in Space (BTCS) Method

This method deals with the same problem described above.

(1) Discretization of the Differential equation.
   a. Approximate the 1st order time derivative with backward difference approximation of $\vartheta(\Delta t)$ at the grid point *(i,k+1):*

$$\frac{u_i^{k+1} - u_i^k}{\Delta t}$$

   b. Approximate the space derivative with central difference approximation of $\vartheta(\Delta x^2)$ at the grid point *(i,k+1):*

$$k\left[\frac{u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}}{\Delta x^2}\right]$$

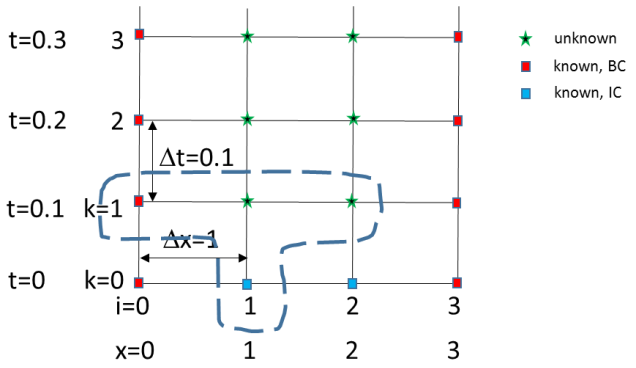
   c. Plug results in (a) and (b) in the original heat equation (Eq-1)

$$\frac{u_i^{k+1} - u_i^k}{\Delta t} = k\left[\frac{u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}}{\Delta x^2}\right]$$

   d. The recurrence formula

$$-\lambda u_{i-1}^{k+1} + (1 + 2\lambda)u_i^{k+1} - \lambda u_{i+1}^{k+1} = u_i^k$$

(2) Discretize the initial and boundary conditions (same as FTCS method)
(3) Show the BTCS computational molecule within the grid of your discretization



(4) According to the problem discretization and the number of unknowns we should find the appropriate number of equations using the recurrence formula. Equations with Indexed-variable are used—NO numbers yet:

Set i=1, and k=0 in the recurrence formula

$$-\lambda u_0^1 + (1+2\lambda)u_1^1 - \lambda u_2^1 = u_1^0$$

Set i=2, and k=0 in the recurrence formula

$$-\lambda u_1^1 + (1+2\lambda)u_2^1 - \lambda u_3^1 = u_2^0$$

As u_0^1 and u_3^1 are known values the above equations simplify to:

$$(1+2\lambda)u_1^1 - \lambda u_2^1 = u_1^0 + \lambda u_0^1$$
$$-\lambda u_1^1 + (1+2\lambda)u_2^1 = u_2^0 + \lambda u_3^1$$

(5) Plugging numbers in equations in (4) to express a 'clean' system of linear equations

$$(1+2(0.0835))u_1^1 - 0.0835u_2^1 = 0 + (0.0835)5$$
$$-0.0835u_1^1 + (1+2(0.0835))u_2^1 = 0 + (0.0835)10$$

Simplifying, yields the clean system:

$$1.167u_1^1 - 0.0835u_2^1 = 0.4175$$
$$-0.0835u_1^1 + 1.167u_2^1 = 0.835$$

or the matrix equation:

$$\begin{bmatrix} 1.167 & -0.0835 \\ -0.0835 & 1.167 \end{bmatrix} \begin{bmatrix} U_1^1 \\ U_2^1 \end{bmatrix} = \begin{bmatrix} 0.4175 \\ 0.835 \end{bmatrix}$$

Although the toy problem (2x2 matrix) does not display tridiagonal characteristics due to its small size be aware that in real life problems these matrices are quite larger and tridiagonal.

(6) Solve the matrix by similar methods as the previous CN method.

For diffusion equations (and many other equations), it can be shown that the Crank–Nicolson method is unconditionally stable. However, the approximate solutions can still contain (decaying) spurious oscillations if the ratio of time step Δt times the thermal diffusivity to the square of space step, Δx^2, is large (typically larger than 1/2 per Von Neumann stability analysis) [i.e., if $\lambda = \frac{\kappa \Delta t}{\Delta x^2} > 1/2$]. For this reason, whenever large time steps or high spatial resolution is necessary, the less accurate BTCS (also called backward Euler method) is often used, which is both **stable** and **immune to oscillations**.

EXERCISES

1- Using FTCS method and matlab rework the toy problem with Δx = 0.5 cm, Δt = 0.1 s, λ = $k\Delta t/\Delta x^2 = 0.835(0.1)/(0.5)^2 = 0.334$ in the range of $0 \leq t \leq 1$
2- Discuss the error order of each method: FTSC, CN and BTCS
3- Classify the FTCS, CN, BTCS into implicit or explicit methods

REFERENCES AND EXTERNAL LINKS (in progress)

1. Numerical PDF Techniques for Scientists and Engineers
http://www3.nd.edu/~dbalsara/Numerical-PDE-Course/

APPENDIX ONE

Next is a matlab code, results and graph.

```
% FTCS Finite Difference Method and the Heat Equation
% file: heatConductionPDE3.m (Feb 20, 2017)
% Use the explicit FTCS method to solve for the temperature distribution
% of a thin  rod insulated at all points, except at its ends with a
% length L (or a SEMIinfinite slab-"losa, plancha"- with a width L)
% dU/dt=d^2U/dx^2;
% IC & BCs:
% U(x,O)=0 C    for 0<x<10
% U(0,t) = 100 C    for t>0
% U(10,t) = 50 C    for t>0
% Specifications
% L = 10 cm     (rod length)
% Assume: dx = 2 cm, dt = 0.1 s, k = 0.835 cm2/s, and
% lambda = kdt/dx^2=0.835(0.1)/(2)^2 = 0.020875


clc, clear, close

% Define constants
L = 10;      % length of domain in x direction
tmax = 12;    % end time
dx = 2;
dt = 0.1;
ix = L/dx + 1;  % ix = 6;number of nodes in x direction for current problem
kt = tmax/dt + 1;   % if t(1)=0, kt = 121;
```

```
                    %number of total time steps for current problem
k=0.835; % thermal diffusivity

r = k*dt/dx^2; r2 = 1 - 2*r;     % r = lambda

% Nodes coordinates
ii=[1:ix];
kk=[1:kt];

% Table title (this section can be totally eliminated to erase headers)
x=([1:ix]-1)*dx;         % except by this one, need for the graph
fprintf(' x ==>      ');
fprintf('%10d ',x);  fprintf('\n');
fprintf(' t, below');

% Initial condition & BCs
t(1)= 0;
uold(1)=100; uold(2:1:ix-1)=0; uold(ix)=50;

uu=zeros(kt,ix);
uu(1,ii)=uold;     % uu stores results at each time selected
fprintf('\n');

% --- Loop over time steps
for m=2:kt
    % uold = u;   % prepare for next step
    t(m) = t(m-1) + dt;
        for i=2:ix-1     % only interior nodes
            uold(i) = r*uold(i-1) + r2*uold(i) + r*uold(i+1);
        end

        uu(m,ii)=uold;
end

% Output
t=t';   % from row to column

uuu=[t,uu]; % uuu stores whole solution: t plus u

    for jj=1:30:kt           % Output selected results
        for ii=1:ix+1
            fprintf('%10.3f ', uuu(jj,ii));
        end
        fprintf('\n');
    end
    fprintf('\n');


% For the graph

tPlot= 0:3:tmax;

plot(x,uuu(31,2:7), '.b','markers',20)
hold on  % allows to print other solutions in same graph
plot(x,uuu(61,2:7), '.r','markers',20)
plot(x,uuu(91,2:7), '.g','markers',20)
plot(x,uuu(121,2:7), '.m','markers',20)
```

```
line(x,uuu(31,2:7),'Color','b')
line(x,uuu(61,2:7),'Color','r')
line(x,uuu(91,2:7),'Color','g')
line(x,uuu(121,2:7),'Color','m')

title('U, Temperature Distribution');
ylabel('U'); xlabel('x');
legend('t = 3 s','t = 6 s','t = 9 s','t = 12 s');
grid on
hold off
```

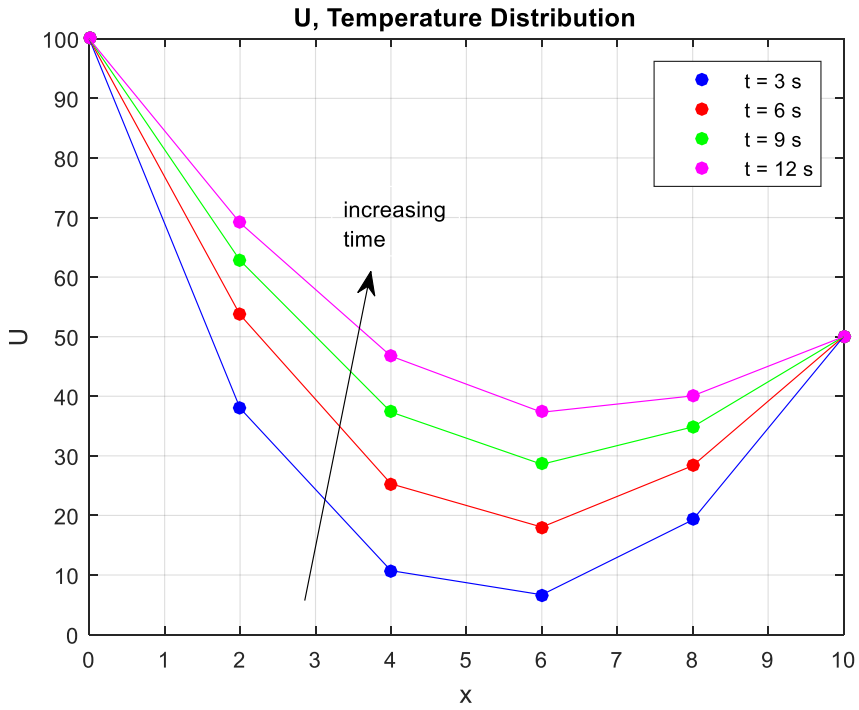
OUTPUT

Selected Results are shown: First column is time, next six columns are temperatures; each row representing the Temperatures distribution at that time step.

Command Window

New to MATLAB? See resources for Getting Started.

x ==>	0	2	4	6	8	10
t, below						
0.000	100.000	0.000	0.000	0.000	0.000	50.000
3.000	100.000	37.992	10.734	6.685	19.248	50.000
6.000	100.000	53.671	25.304	18.040	28.342	50.000
9.000	100.000	62.793	37.355	28.585	34.849	50.000
12.000	100.000	69.101	46.746	37.319	40.065	50.000



Revised: (March 31, 2017)

APPENDIX TWO

Using Taylor series expansion to pproximate U_{k+1} with FDA around $U_{k+\frac{1}{2}}$; and U_k with BDA around $U_{k+\frac{1}{2}} = U(t+\frac{\Delta t}{2})$

$$U_{k+1} = U_{k+\frac{1}{2}} + \frac{\left(\frac{\Delta t}{2}\right)}{1!}U'_{k+\frac{1}{2}} + \frac{\left(\frac{\Delta t}{2}\right)^2}{2!}U''_{k+\frac{1}{2}} + \frac{\left(\frac{\Delta t}{2}\right)^3}{3!}U'''_{k+\frac{1}{2}} + \cdots \quad [Eq1]$$

$$U_k = U_{k+\frac{1}{2}} - \frac{\left(\frac{\Delta t}{2}\right)}{1!}U'_{k+\frac{1}{2}} + \frac{\left(\frac{\Delta t}{2}\right)^2}{2!}U''_{k+\frac{1}{2}} - \frac{\left(\frac{\Delta t}{2}\right)^3}{3!}U'''_{k+\frac{1}{2}} - \cdots [Eq2]$$

Substract Eq2 from Eq1:

$$U_{k+1} - U_k = U_{k+\frac{1}{2}} + 2\left\{\frac{\left(\frac{\Delta t}{2}\right)}{1!}U'_{k+\frac{1}{2}} + \frac{\left(\frac{\Delta t}{2}\right)}{1!}U'_{k+\frac{1}{2}}\right\} + 2\left\{\frac{\left(\frac{\Delta t}{2}\right)^3}{3!}U'''_{k+\frac{1}{2}} + \frac{\left(\frac{\Delta t}{2}\right)^3}{3!}U'''_{k+\frac{1}{2}}\right\} + \cdots$$

$$U_{k+1} - U_k = \frac{\Delta t}{1!}U'_{k+\frac{1}{2}} + \frac{\cancel{\Delta t^3}}{\cancel{3!}}\cancel{U'''_{k+\frac{1}{2}}} + \cdots$$

$$\vartheta(\Delta t^2) + \frac{(U_{k+1} - U_k)}{\Delta t} = U'_{k+\frac{1}{2}}$$

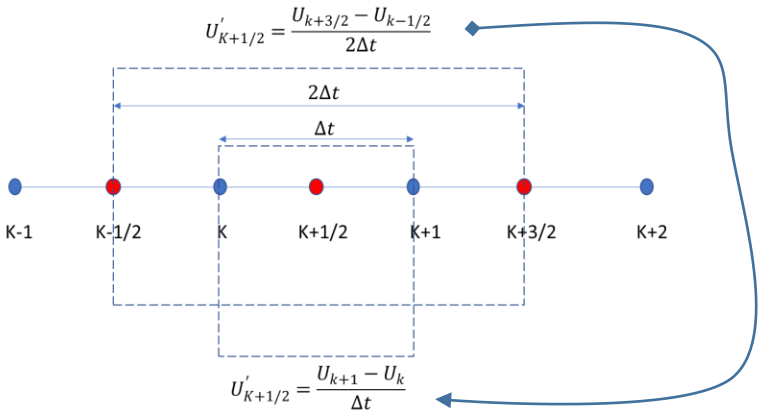
<u>Another alternative using the CDA from the magic tables</u>:

$$U'_K = \frac{U_{k+1} - U_{k-1}}{2\Delta t}$$

Shift forward half the step size $\left(\frac{1}{2}\right)h$, which is equivalent to $k+1+\frac{1}{2} = k+3/2$ and $k-1+\frac{1}{2} = k-1/2$ which yields:

$$U'_{K+1/2} = \frac{U_{k+3/2} - U_{k-1/2}}{2\Delta t} = \frac{U_{k+1} - U_k}{\Delta t}$$

The idea is illustrated below:



APPENDIX THREE.

```
% The Crank Nicolson Method
% This CN solution uses theta=1/2 and the matlab backslash operator
% File: CrankNicolsonP3.m (april18,2018)
% dU/dt=k(d^2U/dt^2)
% U(x,O)=0 for 0<x<10
% U(0,t) = 100  for t>0
% U(10,t) = 50  for t>0

clc, clear, close

tic

% Parameters
k=0.835;          % thermal diffusivity
dx=2;             % delta x
dt=0.1;           % delta t
La=k*dt/dx^2;     % lambda
L=10;             % rod length

t(1)=0; tmax=12;          % 0<=t<=tmax
t=[t(1):dt:tmax];         % time range vector
x=[0:dx:L];               % space domain vector

nx=[(L-0)/dx]+1;      % number of nodes, if L=10, dx=2, nx=6,
nt = tmax/dt + 1;     % if t(1)=0, nt = 121; k=[1,nt]
                      % number of total time steps for current problem

% --- Constant Coefficients of the tridiagonal Matrix
b = La;               % Super diagonal: coefficients of u(i+1)
c = La;               % Subdiagonal: coefficients of u(i-1)
a = 2*(1+La);         % Main Diagonal: coefficients of u(i)

% Boundary conditions and Initial Condition
Uo(1)=100;  Uo(2:nx-1)=0; Uo(nx)=50;
Un(1)=100;  Un(nx)=50;  % BCs are constant along the borders
```

```
% Store results for future use
UUU(1,:)=Uo;

% Loop over time, tango starts here
for k=2:nt

    for ii=1:nx-2
        if ii==1
            d(ii)=c*Uo(ii)+2*(1-c)*Uo(ii+1)+b*Uo(ii+2)+c*Un(1);
        elseif ii==nx-2
            d(ii)=c*Uo(ii)+2*(1-c)*Uo(ii+1)+b*Uo(ii+2)+b*Un(nx);
        else
            d(ii)=c*Uo(ii)+2*(1-c)*Uo(ii+1)+b*Uo(ii+2);
        end
    end  % d is a row vector

    % Transform a, b, c scalar constants in column vectors:
    bb=b*ones(nx-3,1);
    cc=bb;
    aa=a*ones(nx-2,1);

    % Use column vectors to construct tridiagonal matrix
    AA=diag(aa)+ diag(-bb,1)+ diag(-cc,-1);

    % Find the solution for interior nodes i=2,3,4,5
    UU=AA\d';   % UU is temp at interior nodes only

    % Build the whole solution by including BCs
    Un=[Un(1),UU',Un(nx)];  % row vector

    % Store results for future use
    UUU(k,:)=Un;

    % to start a new iteration
    Uo=Un;

end

% Output
t=t';               % from row to column
UUU=[t,UUU];        % UUU stores whole solution: t plus u

    for jj=1:30:nt          % Select results every 30 time steps
        for ii=1:nx+1
            fprintf('%10.3f ', UUU(jj,ii));
        end
        fprintf('\n');
    end
    fprintf('\n');

% For the graph
tPlot= 0:3:tmax;

plot(x,UUU(31,2:7), '.b','markers',20)
hold on  % allows to print other solutions in same graph
plot(x,UUU(61,2:7), '.r','markers',20)
plot(x,UUU(91,2:7), '.g','markers',20)
```

```
plot(x,UUU(121,2:7), '.m','markers',20)

line(x,UUU(31,2:7),'Color','b')
line(x,UUU(61,2:7),'Color','r')
line(x,UUU(91,2:7),'Color','g')
line(x,UUU(121,2:7),'Color','m')

title('U, Temperature Distribution');
ylabel('U'); xlabel('x');
legend('t = 3 s','t = 6 s','t = 9 s','t = 12 s');
grid on
hold off

toc
```

Command Window

New to MATLAB? See resources for Getting Started.

```
    0.000   100.000     0.000     0.000     0.000     0.000    50.000
    3.000   100.000    37.547    10.271     6.440    18.961    50.000
    6.000   100.000    53.246    24.660    17.459    27.924    50.000
    9.000   100.000    62.392    36.650    27.849    34.347    50.000
   12.000   100.000    68.714    46.036    36.542    39.536    50.000

Elapsed time is 0.169291 seconds.
```

First column is time, next 6 columns are U at each spatial node. The 2nd and 7th columns are BCs.