CRANK-NICOLSON EXAMPLE

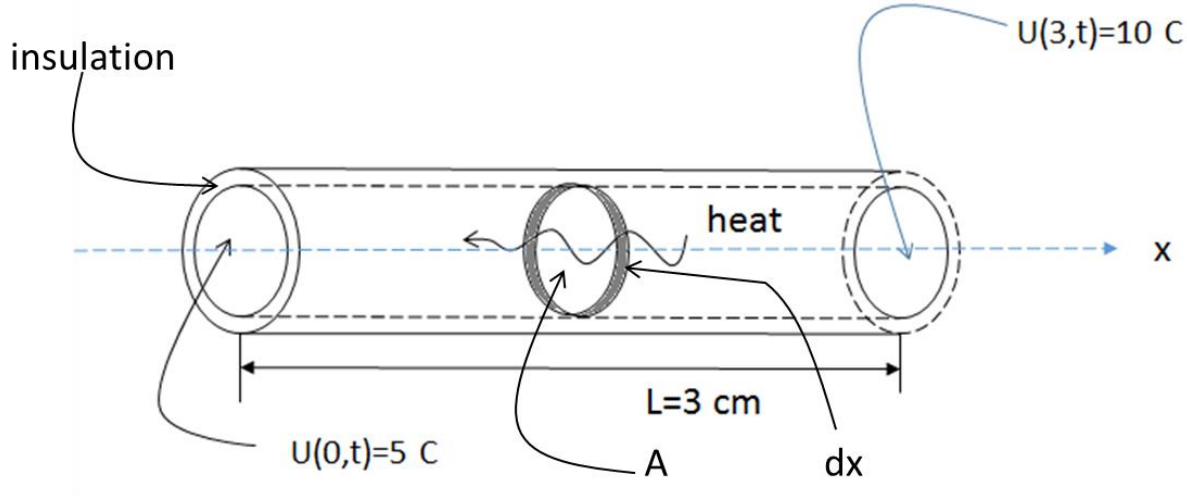
File: CRANK-Example with MATLAB code-V2 (DOC)

PDE: Heat Conduction Equation

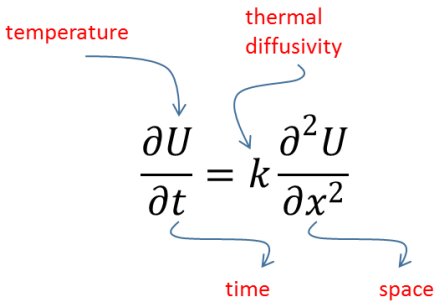
PDF report due before midnight on xx, XX 2016 to marcoantonioarochaordonez@gmail.com. Report includes: code, output and plot. Three-people teams required. Email subject: **PDE-CN**. Submit with a copy to your teammates

Problem Description:

For the problem of a thin, insulated piece of wire with no heat exchange with surroundings except at the two ends.



Solve the following governing PDE:



subject to the Initial Condition and BCs below:

At t = 0, the temperature of interior nodes is zero and the boundary conditions are fixed for all times at

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

Use the Crank-Nicolson method to solve for the temperature distribution of the thin wire insulated at all points, except at its ends with the following specifications:

L = 10 cm (rod length)

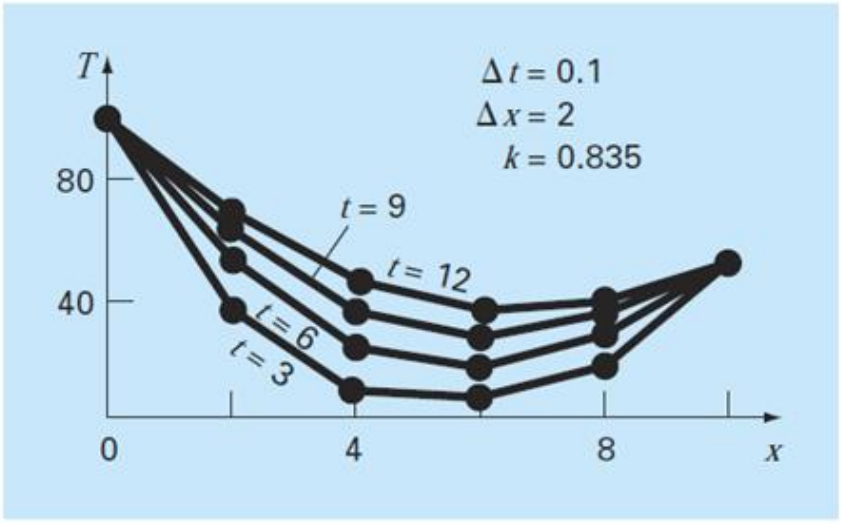
Assume: Δx = 2 cm, Δt = 0.1 s, k = 0.835 cm^2/s, and

λ = $k\Delta t/\Delta x$^2=0.835(0.1)/$(2)^2$ = 0.020875

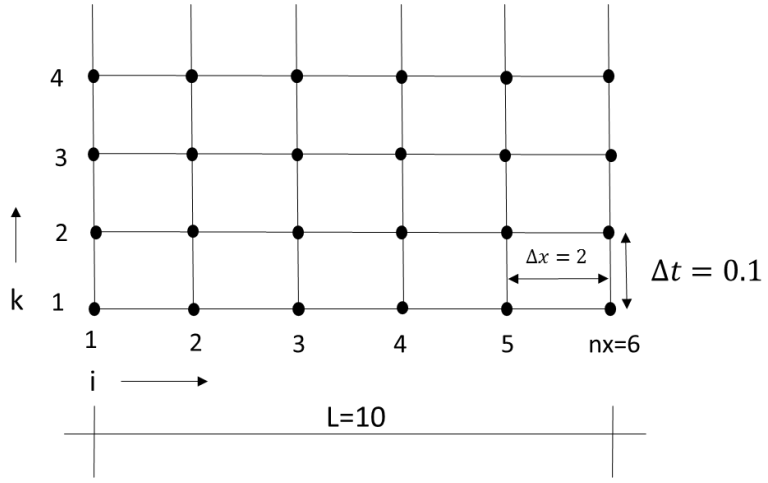
Solve it in the range of t=[0,12] and plot specific time points, t=[3, 6, 9, 12] as shown in the following plot:

Temperature distribution in a long, thin rod as computed with the explicit method

FTCS



For the toy problem:



The CN requires a matrix solution at each time step. The parameter $\theta = 1/2$

Recurrence formula:

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

Define new parameters:

$$c = b = \lambda$$

$$a = 2(1+\lambda)$$

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

Then

$$-cU_{i-1}^{k+1} + aU_i^{k+1} - bU_{i+1}^{k+1} = d_i$$

Be aware that a, b, c are constant coefficients, while d_i varies depending on the i-index. The main diagonal is a; the subdiagonal is b; and c is the superdiagonal. The RHS vector is d_i

To help programming with matlab, take into account that the running index in space dimension is $i = 1,2,3,4,5,nx \quad where\ nx = 6$, and $k = 1,2,3,4,5,nt$ where nt =11 and that the boundary conditions are located at $i = 1$ and $i = nx = 6$ for all k and the initial condition is located at $k = 1$.

At each time step, the space interior nodes have indices: i =2, 3, 4, 5. Therefore, four equations need to be developed to solve the system at each interior node.

For $i = 2, k = 1$:

$$-cU_1^2 + aU_2^2 - bU_3^2 = d_2$$

where $d_2 = cU_1^1 + 2(1-c)U_2^1 + bU_3^1$

As the term containing $U_1^2 = 100$ is known (i.e., BC), pass it to the other side of the equation, then

$$aU_2^2 - bU_3^2 = d_2 + cU_1^2 = d_2^*$$

$$d_2^* = cU_1^1 + 2(1-c)U_2^1 + bU_3^1 + cU_1^2$$

For $i = 3, k = 1$:

$$-cU_2^2 + aU_3^2 - bU_4^2 = d_3$$

$$d_3 = cU_2^1 + 2(1-c)U_3^1 + bU_4^1$$

For $i = 4, k = 1$:

$$-cU_3^2 + aU_4^2 - bU_5^2 = d_4$$

$$d_4 = cU_3^1 + 2(1-c)U_4^1 + bU_5^1$$

For $i = 5, k = 1$:

$$-cU_4^2 + aU_5^2 - bU_6^2 = d_5$$

$$d_5 = cU_4^1 + 2(1-c)U_5^1 + bU_6^1$$

As the term containing $U_6^2 = 50$ is known (i.e., a BC) pass it to the other side of the equation, then

$$-cU_4^2 + aU_5^2 = d_5 + bU_6^2 = d_5^*$$

$$d_5^* = cU_4^1 + 2(1-c)U_5^1 + bU_6^1 + bU_6^2$$

Since d_2^* and d_5^* can be renamed as d_2 and d_5 to simplify the notation:

$$\begin{bmatrix} a & -b & 0 & 0 \\ -c & a & -b & 0 \\ 0 & -c & a & -b \\ 0 & 0 & -c & a \end{bmatrix} \begin{bmatrix} U_2^2 \\ U_3^2 \\ U_4^2 \\ U_5^2 \end{bmatrix} = \begin{bmatrix} d_2 \\ d_3 \\ d_4 \\ d_5 \end{bmatrix}$$

Please keep in mind from above that d_2 and d_5 have specific expressions due to BCs.

For a MATLAB code roll over the i-indices and use the 'old' and 'new' notation for *U* (i.e., *Uo* & *Un*). Rolling indices is needed as MATLAB array indices start in 1:

$$\begin{bmatrix} a & -b & 0 & 0 \\ -c & a & -b & 0 \\ 0 & -c & a & -b \\ 0 & 0 & -c & a \end{bmatrix} \begin{bmatrix} U_2^2 \\ U_3^2 \\ U_4^2 \\ U_5^2 \end{bmatrix} = \begin{bmatrix} d_2 \\ d_3 \\ d_4 \\ d_5 \end{bmatrix} \xrightarrow{from\ math\ to\ matlab} \overbrace{\begin{bmatrix} a & -b & 0 & 0 \\ -c & a & -b & 0 \\ 0 & -c & a & -b \\ 0 & 0 & -c & a \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} U(1) \\ U(2) \\ U(3) \\ U(4) \end{bmatrix}}^{U} = \overbrace{\begin{bmatrix} d(1) \\ d(2) \\ d(3) \\ d(4) \end{bmatrix}}^{d}$$

In other words, the equivalency from Math to MATLAB:

Math	MATLAB
U_2^2	Un(1)
U_3^2	Un(2)
U_4^2	Un(3)
U_5^2	Un(4)

Math	MATLAB
d_2	d(1)
d_3	d(2)
d_4	d(3)
d_5	d(4)

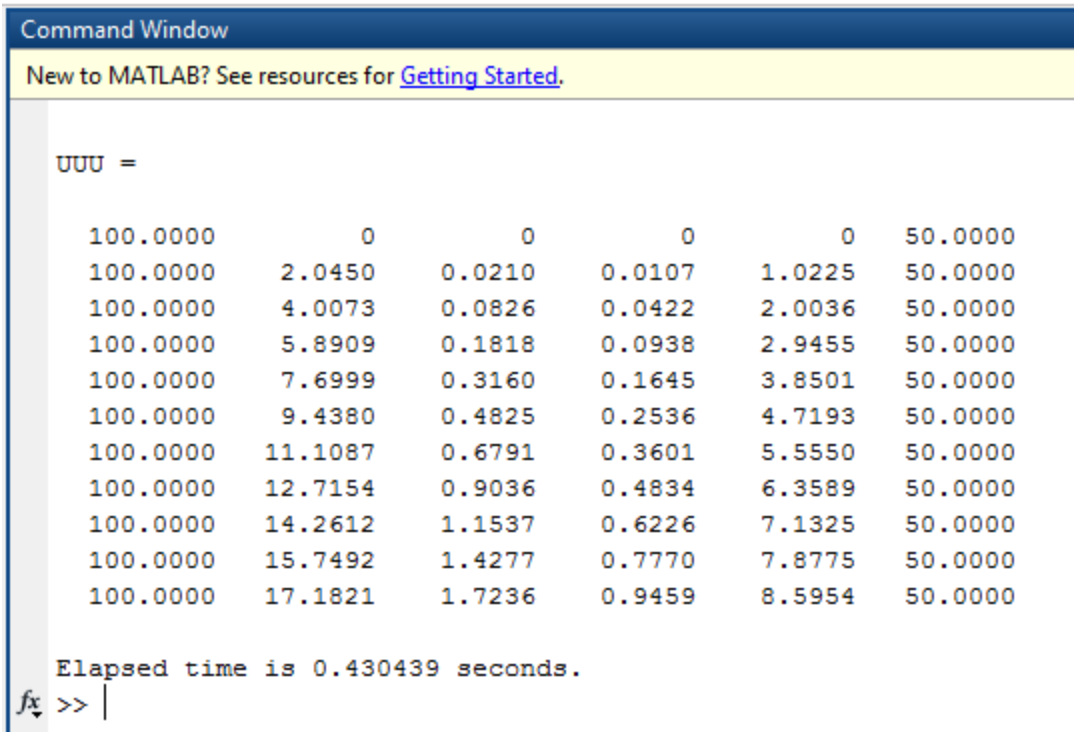
The matrix equation has a simple solution

$$U = A\backslash d$$

For the first iteration, U represents the solution at the new time step t_2 (or t(2))—recall that $t(1)$ is the initial condition. To keep solving the system along time with matlab the U_i^2 (i.e., Un(i)) with superscript $k = 2$ which represents the new U must become the $Uo(i)$ or U_i^1. The new and old U's are implemented in matlab with Un and Uo arrays

```
% The Crank Nicolson Method
% Example
% This CN solution uses theta=1/2 and the matlab backslash operator
% File: CrankNicolsonP3.m

clc, clear, close

tic

% Parameters
La=0.020875;     % lambda
dx=2;
dt=0.1;
nx=6;    % or nx=[(10-0)/dx]+1 with L=10 cm
nt=11;   % to compute 10 time steps k=[1,11]

% --- Constant Coefficients of the tridiagonal Matrix
b = La;              % Super diagonal: coefficients of u(i+1)
c = b;               % 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;

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

% Loop over time
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 diagonal matrices
    AA=diag(aa)+ diag(-bb,1)+ diag(-cc,-1);  %AA is one triadiagonal Matrix

    % 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 over
    Uo=Un;

end

UUU     % Output

toc
```

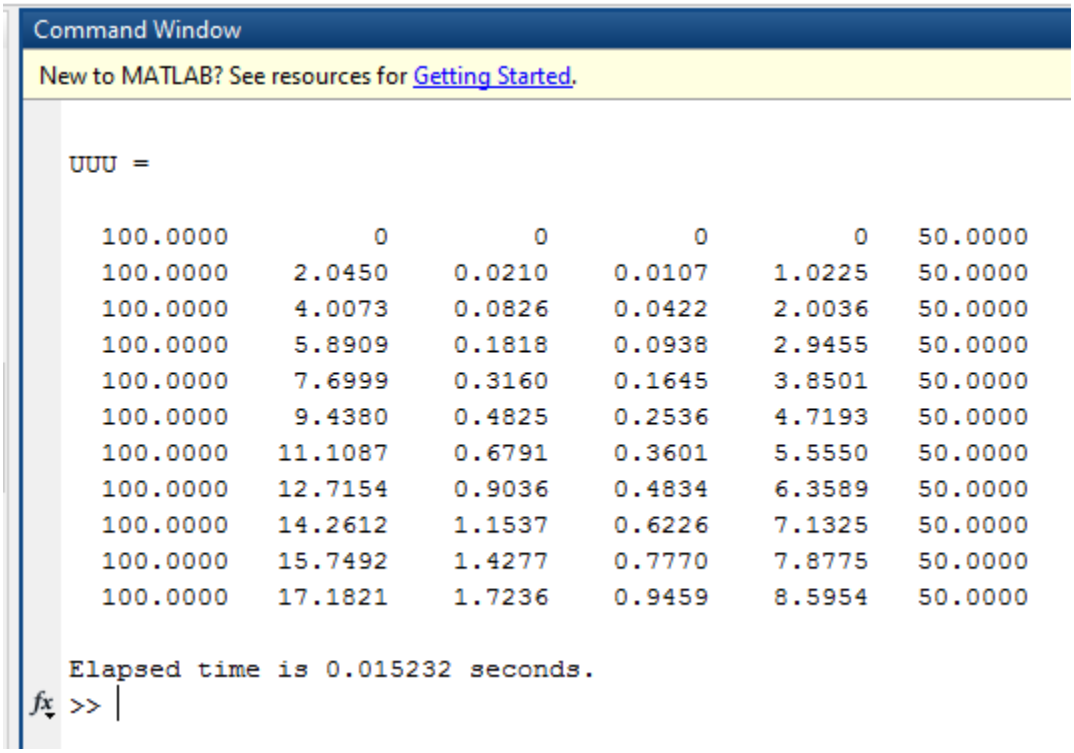


Following alternate solution uses Thomas Algorithm via a Triadiagonal Solver:

```
% The Crank Nicolson Method
% Example
% This CN solution uses theta=1/2 and the Thomas Algorithm
% File: CrankNicolsonP4.m

clc, clear, close

tic
% Parameters
La=0.020875;     % lambda
dx=2;
dt=0.1;
nx=6;    % or nx=[(10-0)/dx]+1 with L=10 cm
nt=11;   % to compute 10 time steps k=[1,11]

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

% Boundary conditions and Initial Conditions
Uo(1)=100;  Uo(2:nx-1)=0; Uo(nx)=50;
Un(1)=100; Un(nx)=50;

% Store results for future use
```

```
UUU(1,:)=Uo;

% Loop over time
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  % note that d is row vector

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

    % Call Tridiagonal Solver
    p=[0,-cc];
    q=aa;
    r=[-bb];
    s=d;
    UU=Tridiag(p,q,r,s);

    % Build the whole solution as row vector
    Un=[Un(1),UU,Un(nx)];

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

    % to start over
    Uo=Un;

end

UUU     % Output

toc


function u=Tridiag(p,q,r,s)
    % Triagonal Solver
    % input
    % p = subdiagonal vector
    % q = diagonal vector
    % r = superdiagonal vector
    % s = right hand side vector
    % output
    % u =solution vector

    n= numel(q);
```

```
    % forward elimination
    for k = 2:n
        factor = p(k)/q(k-1);
        q(k)= q(k) - factor*r(k-1);
        s(k)= s(k) - factor*s(k-1);
    end

    % back substitution
    u(n)=s(n)/q(n);
    for k=n-1:-1:1
        u(k)=(s(k)-r(k)*u(k+1))/q(k);
    end

end
```



Which is $\left(\frac{0.015232}{0.430432}\right)100 = 3.54\%$ of the time required by the previous method using the "slash, \" operator for the solution.