Incorporation of Neumann and Mixed Boundary Conditions into the Crank-Nicholson Method

David Keffer
Department of Materials Science & Engineering
University of Tennessee, Knoxville
date begun: February 26, 2014

I. Problem statement

Consider a linear parabolic partial differential equation in one spatial dimension:

$$d(x,t)\frac{\partial T}{\partial \mathrm{t}} = c(x,t)\frac{\partial^2 T}{\partial \mathrm{x}^2} - a(x,t)T + \left(\frac{\partial c(x,t)}{\partial \mathrm{x}} - b_x(x,t)\right)\frac{\partial T}{\partial \mathrm{x}} + f(x,t) \tag{1}$$

where the functions, $a(x,t)$, $b_x(x,t)$, $c(x,t)$, $d(x,t)$ and $f(x,t)$ are known functions of space and time.

Because the equation is first order in time, this problem requires one initial condition of the form

$$T(x,t_o) = T_i(x)$$

where the function, $T_i(x)$, provides the initial profile of the unknown.

In a previous lecture package, we showed how Dirichlet boundary conditions could be incorporated into the solution of the PDE. In this package, we extend the approach to arbitrary linear boundary conditions of the form,

$$a_{BCo}(t)T(x_o,t) + b_{BCo}(t)\frac{\partial T(x_o,t)}{\partial x} + c_{BCo}(t) = 0 \tag{2.o}$$

and

$$a_{BCf}(t)T(x_f,t) + b_{BCf}(t)\frac{\partial T(x_f,t)}{\partial x} + c_{BCf}(t) = 0 \tag{2.f}$$

These are general boundary conditions because they contain both Dirichlet and Neumann boundary conditions. Sometimes, equation (2) is called mixed boundary conditions. The mixed boundary conditions revert to Dirichlet boundary conditions when $b_{BC}(t)$ is set to zero. The mixed boundary conditions revert to Neumann boundary conditions when $a_{BC}(t)$ is set to zero.

We define two additional spatial nodes before the first boundary and after the last boundary to allow us to account for mixed boundary conditions. See the figure below.

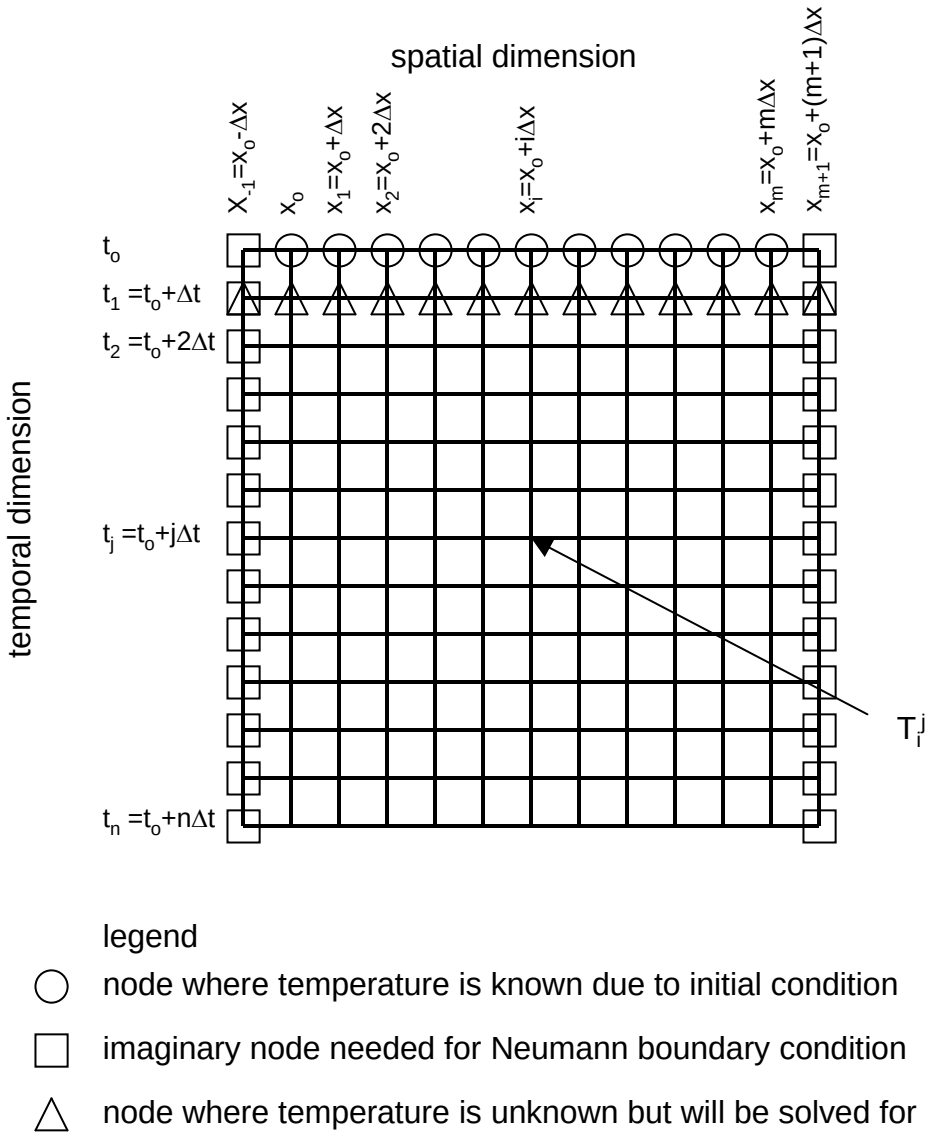

Figure One. Schematic of the spatial and temporal discretization. Case II. Mixed Boundary Conditions.

II. Discretization

The discretization of the PDE is unaffected by the change in boundary conditions. We use the same centered finite difference formula to approximate the first spatial derivatives in the boundary condition. Thus equation (2.o) becomes in discretized form

$$a_{BCo}(t)T(x_o,t)+b_{BCo}(t)\frac{T(x_o-\Delta x,t)-T(x_o+\Delta x,t)}{2\Delta x}+c_{BCo}(t)=0 \qquad (3.o)$$

which in the notation adopted earlier can be written as

$$-\frac{b_{BCo}(t)}{2\Delta x}T_{i-1}^{j}+a_{BCo}(t)T_i^{j}+\frac{b_{BCo}(t)}{2\Delta x}T_{i+1}^{j}=-c_{BCo}(t) \qquad (4.o)$$

and analogously for the RHS boundary

$$-\frac{b_{BCf}(t)}{2\Delta x}T_{i-1}^{j}+a_{BCf}(t)T_i^{j}+\frac{b_{BCf}(t)}{2\Delta x}T_{i+1}^{j}=-c_{BCf}(t) \qquad (4.o)$$

These are two linear algebraic equations. If the boundary conditions are Neumann or Mixed (in other words b_{BCo} and/or b_{BCf} is non-zero, then these equations are used to define the values of the dependent variable at the imaginary node(s). In this case the PDE is used to define the values of the dependent variable at the boundary node(s), located at x_o and x_f.

This is in contrast to the case where the boundary conditions were of the Dirichlet form, (in other words b_{BCo} and/or b_{BCf} are zero. In that case, there were no imaginary nodes. The equations given above are used to define the value of the dependent variable at the boundary node(s) located at x_o and x_f.

We have previously derived the Crank-Nicholson method, as

$$\left(\underline{\underline{I}}-\underline{\underline{K}}^{*\,j+1}\right)\underline{T}^{j+1}=\left[\left(\underline{\underline{I}}+\underline{\underline{K}}^{*\,j}\right)\underline{T}^{j}+\underline{R}^{*\,j+1}+\underline{R}^{*\,j}\right] \qquad (5)$$

where

$$K^{*j}_{\ell,i} = \begin{cases} K_d = \dfrac{\Delta t}{d_i^j}\left(-\dfrac{2c_i^j}{\Delta x^2} - a_i^j\right) & \text{for i} = \ell, 1 \le \text{i} \le \text{m - 1} \\ K_{d+} = \dfrac{\Delta t}{d_i^j}\left(\dfrac{c_i^j}{\Delta x^2} + \dfrac{1}{2\Delta x}\left(\dfrac{dc_i^j}{dx} - b_i^j\right)\right) & \text{for i} = \ell + 1, 1 \le \text{i} \le \text{m - 2} \\ K_{d-} = \dfrac{\Delta t}{d_i^j}\left(\dfrac{c_i^j}{\Delta x^2} - \dfrac{1}{2\Delta x}\left(\dfrac{dc_i^j}{dx} - b_i^j\right)\right) & \text{for i} = \ell - 1, 2 \le \text{i} \le \text{m - 1} \\ 0 \quad \text{otherwise} \end{cases}$$

The matrix, $K^{*j}_{\ell,i}$, is tridiagonal. The associated vector of constants is

$$R^{*j}_i = \begin{cases} R_d = \dfrac{\Delta t}{d_i^j} f_i^j & \text{for } 2 \le \text{i} \le \text{m - 2} \\ R_1 = \dfrac{\Delta t}{d_i^j} f_i^j + K_{d-} T_o & \text{for } i = 1 \\ R_{m-1} = \dfrac{\Delta t}{d_i^j} f_i^j + K_{d+} T_f & \text{for } i = \text{m} - 1 \end{cases}$$

The only remaining issue is integrating these sets of linear equations, describing the BCs adn the PDEs.

We have to be careful about our book-keeping. There are conceptually *m* intervals in the real space of the system as shown in Figure 1. With the imaginary nodes, the indexing of these nodes runs from -1 to *m*+1. In Matlab, the indices on matrices must begin at 1. Therefore, the matrix index runs from 1 to $m_x = m + 3$, which is the total number of nodes.

The accounting is slightly complicated by the fact that there are four choice of types of boundary conditions, depending on whether the first and second BCs are Dirichlet or Mixed (here we use Mixed BCs to include Neumann BCs as well, since their treatment is equivalent). In Table 1 below, we provide a summary for which algebraic equations are used to solve for each node, as a function of the combination of types of BCs.

If both BCs are of the Dirichlet form, then we don't need the imaginary nodes. However, these nodes exist in a generalized code that can also solve Mixed BCs. Therefore, the algebraic equation that we use to define the imaginary nodes in this case is to simply set them to zero and ignore them thereafter. The algebraic equations representing the boundary conditions are used at nodes 2 and $m_x - 1$. The algebraic equations representing the PDEs are used at all of the interior nodes.

If both BCs are of the Mixed (or Neumann) type, then we use the algebraic equations representing the boundary conditions at the imaginary nodes (different than what was done for the Dirichlet BCs). All other nodes use the algebraic equations representing the PDE.

The cases representing a combination of boundary conditions, either D on the LHS and M on the RHS or vice versa are described in Table 1 below.

x subscript in Figure 1.	matrix index	$BC(x_o) = D$ $BC(x_f) = D$	$BC(x_o) = D$ $BC(x_f) = M$	$BC(x_o) = M$ $BC(x_f) = D$	$BC(x_o) = M$ $BC(x_f) = M$
-1	1	$T_1 = 0$	$T_1 = 0$	use $BC(x_o)$	use $BC(x_o)$
0	2	use $BC(x_o)$	use $BC(x_o)$	use PDE	use PDE
$1 \le i \le m-1$	$3 \le i \le m_x - 2$ 3	use PDE	use PDE	use PDE	use PDE
m	$m_x - 1$	use $BC(x_f)$	use PDE	use $BC(x_f)$	use PDE
m+1	m_x	$T_{m_x} = 0$	use $BC(x_f)$	$T_{m_x} = 0$	use $BC(x_f)$

Table 1. This table shows which algebraic equation to use depending upon the combination of boundary conditions.

A code that implements the Crank-Nicholson method for any combination of Dirichlet and Mixed boundary conditions is given at the end of this document.

An example application of the code is provided below.

Consider the following problem

$$(d=1)\frac{\partial T}{\partial \mathrm{t}} = (c=1)\frac{\partial^2 T}{\partial \mathrm{x}^2} - (a=0)T + \left((\frac{\partial c}{\partial \mathrm{x}} = 0) - (b=0) \right)\frac{\partial T}{\partial \mathrm{x}} + (f=0)$$

$$T(x,t_o) = T_i(x) = 300 \qquad T(x_o,t) = T_o(t) = 400 \qquad \left.\frac{dT}{\partial x}\right|_{x=x_f} = 0$$

In this case, the rod is initially at 300 K. One end of the rod is then maintained at 400 K. The other end of the rod is insulated so that the heat flux (and thus the temperature gradient) is zero.

We used 10 spatial intervals and solved from to = 0 to tf = 1, with 100 temporal intervals. In the plot, we show the profile at every time.

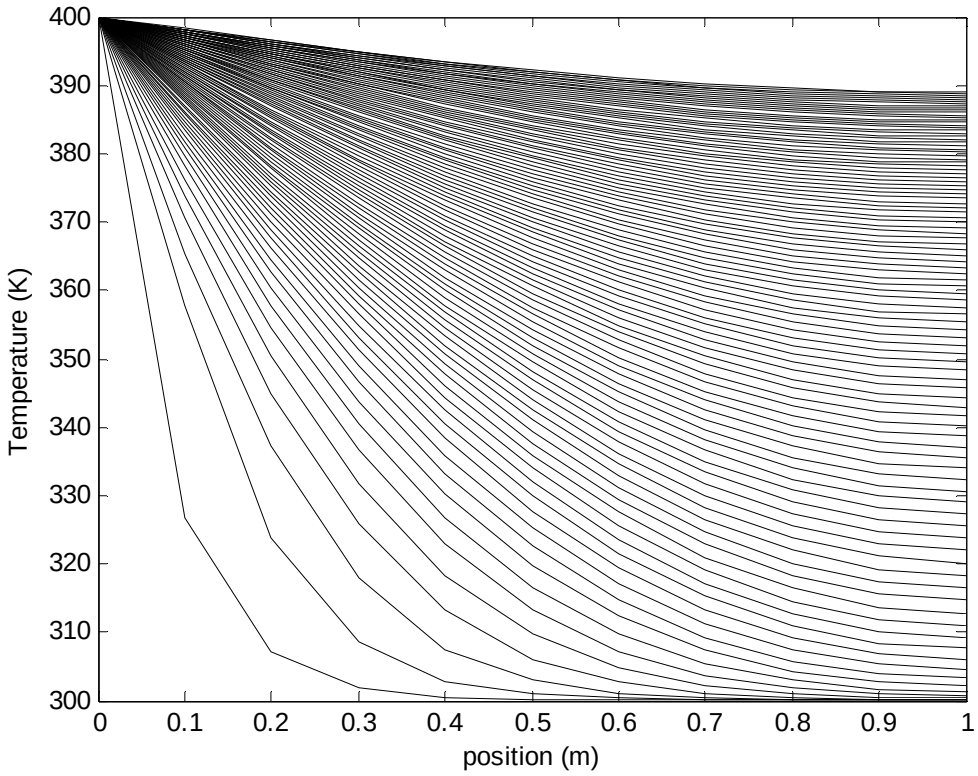

Figure 2. Transient behavior of a metal rod initially at 300 K with one boundary attached to a thermal reservoir that maintains the temperature at 400 K and the other boundary insulated. Since there is no heat-loss to the surroundings in this problem, the steady-state temperature corresponds to a uniform temperature of 400 K in the rod. In this plot, the lowest profile is the earliest time. As time progresses, the profile approaches the steady state profile. The temporal spacing between lines is constant and so the gap between lines serves as a measure of the rate of change in the temperature. The rate is fastest early on and slows down as the temperature gradients decrease and the rod approaches the steady state profile. In this plot, we have clearly not reached steady state. Theoretically, it takes an infinite time to reach steady state. Practically speaking our ability to measure deviations from steady state is limited by the accuracy of our thermocouples and the noise present due to external disturbances not accounted for in this idealized model.

Appendix. Matlab Subroutines

Code A. linparapde_crank_anyBC.m

```
function linparapde_crank_anyBC
%
%  The routine linparapde_euler will solve
%  a single linear 1-D parabolic PDE of the form
%
%  d(x,t)*dT/dt = div(c(x,t)*grad(T)) - a(x,t)*T -b(x,t)*dTdx +f(x,t)
%
%  using the Euler Method.
%
%  The initial condition must have the form
%  IC:  T(x,to) = To(x);
%
%  The boundary conditions must both be
%  Dirichlet Boundary Conditions of the form
%  single linear 1-D parabolic PDE with
%  2 Mixed Boundary Conditions
%  BC 1:  aBCo(t)*T(xo,t) + bBCo(t)*dTdx(xo,t) + cBCo(t) = 0;
%  BC 2:  aBCf(t)*T(xf,t) + bBCf(t)*dTdx(xf,t) + cBCf(t) = 0;
%
%  The necessary functions
%  a(x,t), b(x,t), c(x,t), dcdx(x,t), f(x,t), To(x),
%  aBCo(t), bBCo(t), cBCo(t), aBCf(t), bBCf(t), cBCf(t)
%  are entered at the bottom of the file
%
%  The initial value, final value and discretization in time, t,
%  to, tf, and dt must be entered at the top of the file.
%
%  The initial value, final value and discretization in space, x,
%  xo, xf, and dx must be entered at the top of the file.
%
%  The type of boundary conditions 'D', 'N' or 'M'
%  for each boundary must be entered at the top of the file.
%
%  code written by:  David J. Keffer
%  University of Tennessee, dkeffer@utk.edu
%  code written:  October 21, 2001
%  comments last updated:  February 26, 2014
%
clear all;
close all;
%
%  define type of boundary condition
%  Choices are 'D' = Dirichlet, i.e. bBC(t) = 0
%  Choices are 'N' = Neumann, i.e. aBC(t) = 0
%  Choices are 'M' = Mixed, bBC(t) ~= 0 & aBC(t) ~= 0
%
BC(1) = 'D';
BC(2) = 'N';

% discretize time
to = 0;
tf = 1.0e-0;
dt = 1.0e-2;
```

```
tvec = [to:dt:tf];
nt = length(tvec);

% discretize space
xo = 0;
xf = 1.0;
dx = 1.0e-1;
% include imaginary boundary nodes
xvec = [xo-dx:dx:xf+dx];
nx = length(xvec);

% dimension solution
Tmat = zeros(nx,nt);


% dimension temporary vectors
Told = zeros(nx,1);
Tnew = zeros(nx,1);
Amat = zeros(nx,nx);
bvec = zeros(nx,1);

% apply initial conditions to all real nodes
i = 1;
t = tvec(i);
for j = 2:1:nx-1
    x = xvec(j);
    Amat(j,j) = 1.0;
    bvec(j) = icfunk(x);
end

%  implement LHS BCs
   if (BC(1) == 'D')
      Amat(1,1) = 1.0;
      bvec(1) = 0.0d0;
%      Amat(2,2) = aBCo(t);
%      bvec(2) = -cBCo(t);
   else
      Amat(1,1) = -1.0/(2.0*dt)*bBCo(t);
      Amat(1,2) =               aBCo(t);
      Amat(1,3) =  1.0/(2.0*dt)*bBCo(t);
      bvec(1) = -cBCo(t);
   end

%  implement RHS BCs
   if (BC(2) == 'D')
      Amat(nx,nx) = 1.0;
      bvec(nx) = 0.0d0;
%      Amat(nx-1,nx-1) = aBCf(t);
%      bvec(nx-1) = -cBCf(t);
   else
      Amat(nx,nx-2) = -1.0/(2.0*dt)*bBCf(t);
      Amat(nx,nx-1) =               aBCf(t);
      Amat(nx,nx)   =  1.0/(2.0*dt)*bBCf(t);
      bvec(nx) = -cBCf(t);
   end

% store initial conditions in solution matrix
Tmat(1:nx,1) = inv(Amat)*bvec;

%
```

```
%  Determine, based on type of BC, how to define matrix
%  ivaro = index of first node to be solved by PDE
%  ivarf = index of last node to be solved by PDE
%  nvar = number of nodes to be solved by PDE
%
if (BC(1) == 'D')
   ivaro = 3;
else
   ivaro = 2;
end
if (BC(2) == 'D')
   ivarf = nx-2;
else
   ivarf = nx-1;
end
nvar = ivarf - ivaro + 1;

% loop over times
Told(1:nx) = Tmat(1:nx,i);
for i = 2:1:nt
%  reinitialize Amat and bvec
   Amat = zeros(nx,nx);
   bvec = zeros(nx,1);
%  update time
   told = tvec(i-1);
   t = tvec(i);

%  implement LHS BCs
   if (BC(1) == 'D')
      Amat(1,1) = 1.0;
      bvec(1) = 0.0d0;
      Amat(2,2) = aBCo(t);
      bvec(2) = -cBCo(t);
   else
      Amat(1,1) = -1.0/(2.0*dt)*bBCo(t);
      Amat(1,2) =               aBCo(t);
      Amat(1,3) =  1.0/(2.0*dt)*bBCo(t);
      bvec(1) = -cBCo(t);
   end

%  implement RHS BCs
   if (BC(2) == 'D')
      Amat(nx,nx) = 1.0;
      bvec(nx) = 0.0d0;
      Amat(nx-1,nx-1) = aBCf(t);
      bvec(nx-1) = -cBCf(t);
   else
      Amat(nx,nx-2) = -1.0/(2.0*dt)*bBCf(t);
      Amat(nx,nx-1) =               aBCf(t);
      Amat(nx,nx)   =  1.0/(2.0*dt)*bBCf(t);
      bvec(nx) = -cBCf(t);
   end

%  implement Crank-Nicholson for PDEs from ivaro to ivarf
   Kmatold = 0.5*getK(nx,ivaro,ivarf,told,xvec,dt,dx);
   Rvecold = 0.5*getR(nx,ivaro,ivarf,told,xvec,dt,dx);
   Kmatnew = 0.5*getK(nx,ivaro,ivarf,t,   xvec,dt,dx);
   Rvecnew = 0.5*getR(nx,ivaro,ivarf,t,   xvec,dt,dx);

%  add missing contributions to diagonal elements
```

```
   Kmatnew = -Kmatnew;
   for j = ivaro:1:ivarf
       Kmatnew(j,j) = 1.0 + Kmatnew(j,j);
       Kmatold(j,j) = 1.0 + Kmatold(j,j);
   end

%  build and invert Amat
       Amat(1:nx,1:nx) = Amat(1:nx,1:nx) + Kmatnew(1:nx,1:nx);
       Ainv = inv(Amat);

% build the bvec
       KmatTold = Kmatold*Told;
       for j = ivaro:1:ivarf
           bvec(j) = bvec(j) + KmatTold(j) + Rvecnew(j) + Rvecold(j);
       end

% solve for the new temperatures
   Tnew = Ainv*bvec;
   Tmat(1:nx,i) = Tnew;
   Told = Tnew;
end

% plot
figure(1);
nskip = 1;
for i = 1:nskip:nt
   plot(xvec(2:nx-1),Tmat(2:nx-1,i),'k-');
   %pause(1);
   hold on;
end
xlabel('position (m)')
ylabel('Temperature (K)');

function Kmat = getK(nx,ivaro,ivarf,t,xvec,dt,dx);
   Kmat = zeros(nx,nx);
   % diagonal elements of matrix
   for j = ivaro:1:ivarf
      x = xvec(j+1);
      Kmat(j,j) =  dt/dfunk(x,t)* (-2*cfunk(x,t)/dx^2 - afunk(x,t) );
   end
   % upper off-diagonal elements of matrix
   for j = ivaro:1:ivarf
      x = xvec(j+1);
      Kmat(j,j+1) =  dt/dfunk(x,t)* (cfunk(x,t)/dx^2 + (dcdxfunk(x,t) -
bfunk(x,t))/(2*dx) );
   end
   % lower off-diagonal elements of matrix
   for j = ivaro:1:ivarf
      x = xvec(j+1);
      Kmat(j,j-1) =  dt/dfunk(x,t)* (cfunk(x,t)/dx^2 - (dcdxfunk(x,t) -
bfunk(x,t))/(2*dx) );
   end

   function Rvec = getR(nx,ivaro,ivarf,t,xvec,dt,dx);
   Rvec = zeros(nx,1);
   for j = ivaro:1:ivarf
      x = xvec(j+1);
      Rvec(j) =  dt/dfunk(x,t)*ffunk(x,t);
   end
```

```
function a = afunk(x,t);
a = 0;

function b = bfunk(x,t);
b = 0;

function c = cfunk(x,t);
c = 1;

function dcdx = dcdxfunk(x,t);
dcdx = 0;

function d = dfunk(x,t);
d = 1;

function f = ffunk(x,t);
f = 000;

%
%  function defining initial condition
%

function ic = icfunk(x);
ic = 300;

%
%  functions defining LHS boundary condition
%

function f = aBCo(t);
f = 1;

function f = bBCo(t);
f = 0;

function f = cBCo(t);
f = -400;

%
%  functions defining RHS boundary condition
%

function f = aBCf(t);
f = 0;

function f = bBCf(t);
f = 1;

function f = cBCf(t);
f = 0;
```

An example of using `linparapde_crank_anyBC` is given below.

```
>> linparapde_crank_anyBC
```