The Heat Equation

Mario L. Gutierrez Abed
04-04-2021

Problem 1. *Determine the order of accuracy of the method*

$$U_i^{n+2} = U_i^n + \frac{2\Delta t}{(\Delta x)^2}\left(U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}\right) \tag{1}$$

(in both space and time) for the heat equation $u_t = u_{xx}$.

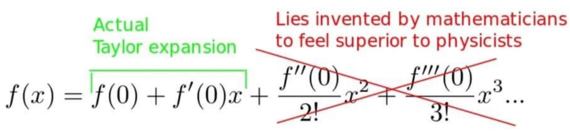
Solution. To simplify notation we set $k \equiv \Delta t$ and $h \equiv \Delta x$. We then rewrite Eq. (1) as

$$\frac{U_i^{n+2} - U_i^n}{2k} = \frac{U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}}{h^2}. \tag{2}$$

Then we look at the LTE $\tau_i^n \equiv \tau(t_n, x_i)$, which is given by inserting the true solution $u(t,x)$ into Eq. (2):

$$\tau(t,x) = \frac{u(t+2k,x) - u(t,x)}{2k} - \frac{u(t+k,x-h) - 2u(t+k,x) + u(t+k,x+h)}{h^2}. \tag{3}$$

Of course, we do not know a priori what the true solution is, but if we assume that it is smooth enough, we can then Taylor-expand the above expression.



We start with

$$
\begin{aligned}
u(t+2k,x) &= u + (2k)u_t + \frac{1}{2!}(2k)^2 u_{tt} + \frac{1}{3!}(2k)^3 u_{ttt} + \frac{1}{4!}(2k)^4 u_{tttt} + \frac{1}{5!}(2k)^5 u_{ttttt} + O(k^6) \\
&= u + 2ku_t + 2k^2 u_{tt} + \frac{4}{3}k^3 u_{ttt} + \frac{2}{3}k^4 u_{tttt} + \frac{4}{15}k^5 u_{ttttt} + O(k^6).
\end{aligned} \tag{4}
$$

Hence, the first term on the RHS of Eq. (3) becomes

$$
\frac{u(t+2k,x) - u(t,x)}{2k} = u_t + ku_{tt} + \frac{2}{3}k^2 u_{ttt} + \frac{1}{3}k^3 u_{tttt} + \frac{2}{15}k^4 u_{ttttt} + O(k^5). \tag{5}
$$

Similar to Eq. (4), the term $u(t+k,x)$ from the second expression on the RHS of Eq. (3) is given by

$$
u(t+k,x) = u + ku_t + \frac{1}{2}k^2 u_{tt} + \frac{1}{6}k^3 u_{ttt} + \frac{1}{24}k^4 u_{tttt} + O(k^5). \tag{6}
$$

The remaining terms are expanded in both space and time:

$$
\begin{aligned}
u(t+k,x\pm h) = u &+ ku_t \pm hu_x + \frac{1}{2!}\left[k^2 u_{tt} + h^2 u_{xx} \pm 2kh\, u_{tx}\right] \\
&+ \frac{1}{3!}\left[k^3 u_{ttt} \pm h^3 u_{xxx} \pm 3k^2 h\, u_{ttx} + 3kh^2\, u_{txx}\right] \\
&+ \frac{1}{4!}\left[k^4 u_{tttt} + h^4 u_{xxxx} + 4k^2h^2\, u_{ttxx} \pm 4k^3 h\, u_{tttx} \pm 4kh^3\, u_{txxx}\right] + O(k^5+h^5) \\
= u &+ ku_t \pm hu_x + \frac{1}{2}k^2 u_{tt} + \frac{1}{2}h^2 u_{xx} \pm kh\, u_{tx} \\
&+ \frac{1}{6}k^3 u_{ttt} \pm \frac{1}{6}h^3 u_{xxx} \pm \frac{1}{2}k^2 h\, u_{ttx} + \frac{1}{2}kh^2\, u_{txx} \\
&+ \frac{1}{24}k^4 u_{tttt} + \frac{1}{24}h^4 u_{xxxx} + \frac{1}{6}k^2h^2\, u_{ttxx} \pm \frac{1}{6}k^3 h\, u_{tttx} \pm \frac{1}{6}kh^3\, u_{txxx} \\
&+ O(k^5+h^5),
\end{aligned} \tag{7}
$$

where we used the commutativity of the mixed partial derivatives. From (7) we gather

$$
\begin{aligned}
u(t+k,x+h) + u(t+k,x-h) = {} & u + ku_t + hu_x + \frac{1}{2}k^2 u_{tt} + \frac{1}{2}h^2 u_{xx} + kh\, u_{tx} \\
& + \frac{1}{6}k^3 u_{ttt} + \frac{1}{6}h^3 u_{xxx} + \frac{1}{2}k^2 h\, u_{ttx} + \frac{1}{2}kh^2\, u_{txx} \\
& + \frac{1}{24}k^4 u_{tttt} + \frac{1}{24}h^4 u_{xxxx} + \frac{1}{6}k^2h^2\, u_{ttxx} + \frac{1}{6}k^3 h\, u_{tttx} + \frac{1}{6}kh^3\, u_{txxx} \\
& + u + ku_t - hu_x + \frac{1}{2}k^2 u_{tt} + \frac{1}{2}h^2 u_{xx} - kh\, u_{tx} \\
& + \frac{1}{6}k^3 u_{ttt} - \frac{1}{6}h^3 u_{xxx} - \frac{1}{2}k^2 h\, u_{ttx} + \frac{1}{2}kh^2\, u_{txx} \\
& + \frac{1}{24}k^4 u_{tttt} + \frac{1}{24}h^4 u_{xxxx} + \frac{1}{6}k^2h^2\, u_{ttxx} - \frac{1}{6}k^3 h\, u_{tttx} - \frac{1}{6}kh^3\, u_{txxx} \\
& + O(k^5+h^5) \\
= {} & 2u + 2ku_t + k^2 u_{tt} + h^2 u_{xx} + \frac{1}{3}k^3 u_{ttt} + kh^2\, u_{txx} \\
& + \frac{1}{12}k^4 u_{tttt} + \frac{1}{12}h^4 u_{xxxx} + \frac{1}{3}k^2h^2\, u_{ttxx} + O(k^5+h^5).
\end{aligned} \tag{8}
$$

Hence, combining Eqs. (6) & (8), the second term on the RHS of Eq. (3) becomes

$$
\begin{aligned}
\frac{u(t+k,x-h)-2u(t+k,x)+u(t+k,x+h)}{h^2} &= \frac{1}{h^2}\cdot\Big[2u + 2ku_t + k^2u_{tt} + h^2u_{xx} + \frac{1}{3}k^3u_{ttt} + kh^2\,u_{txx} \\
&\quad + \frac{1}{12}k^4u_{tttt} + \frac{1}{12}h^4u_{xxxx} + \frac{1}{3}k^2h^2\,u_{ttxx} \\
&\quad - 2(u + ku_t + \frac{1}{2}k^2u_{tt} + \frac{1}{6}k^3u_{ttt} + \frac{1}{24}k^4u_{tttt}) \\
&\quad + O(k^5+h^5)\Big] \\
&= \frac{1}{h^2}\cdot\Big[h^2u_{xx} + kh^2\,u_{txx} + \frac{1}{12}h^4u_{xxxx} + \frac{1}{3}k^2h^2\,u_{ttxx} \\
&\quad + O(k^5+h^5)\Big] \\
&= u_{xx} + k\,u_{txx} + \frac{1}{12}h^2u_{xxxx} + \frac{1}{3}k^2\,u_{ttxx} \qquad (9) \\
&\quad + O(k^5+h^3).
\end{aligned}
$$

Now, combining this result with Eq. (5) and plugging into Eq. (3), we find the truncation error

$$
\begin{aligned}
\tau(t,x) &= \overbrace{\frac{u(t+2k,x)-u(t,x)}{2k}}^{\textit{Eq. }(5)} - \overbrace{\frac{u(t+k,x-h)-2u(t+k,x)+u(t+k,x+h)}{h^2}}^{\textit{Eq. }(9)} \\
&= u_t + ku_{tt} + \frac{2}{3}k^2u_{ttt} + \frac{1}{3}k^3u_{tttt} + \frac{2}{15}k^4u_{ttttt} + O(k^5) \\
&\quad - \left[u_{xx} + k\,u_{txx} + \frac{1}{12}h^2u_{xxxx} + \frac{1}{3}k^2\,u_{ttxx} + O(k^5+h^3)\right] \\
&= \cancelto{0}{u_t - u_{xx}} + \cancelto{0}{ku_{tt} - ku_{txx}} + \cancelto{1/3k^2u_{ttt}}{\frac{2}{3}k^2u_{ttt} - \frac{1}{3}k^2\,u_{ttxx}} \\
&\quad - \frac{1}{12}h^2u_{xxxx} + O(k^3+h^3) \\
&= \frac{1}{3}k^2u_{ttt} - \frac{1}{12}h^2u_{xxxx} + O(k^3+h^3). \qquad (10)
\end{aligned}
$$

Here we used the heat equation $u_t = u_{xx}$ and its derivatives $u_{tt} = u_{xxt}$ and $u_{ttt} = u_{xxtt}$ to simplify the expression. We also note that we could have expanded Eq. (5) to second-order only, but we had no way of knowing this a priori, of course. The result (10) shows that the scheme is second-order accurate in both space and time, since the LTE is $O(k^2+h^2)$. □

Problem 2. *Your task is to solve the problem*

$$u_t(t,x) - \sigma u_{xx}(t,x) = f(t,x), \quad t > 0, \; a < x < b \tag{11a}$$
$$u_x(t,a) = u_A(t), \quad u_x(t,b) = u_B(t), \tag{11b}$$
$$u(0,x) = u_0(x) \tag{11c}$$

numerically. Discretize the problem using

- *second-order centered finite difference scheme with space step h for u_{xx}*
- *forward difference at point $x = a$ for u_x*
- *backward difference at point $x = b$ for u_x*
- *(a) forward Euler's (b) backward Euler's method with time step k for the time derivative u_t.* [1]

I suggest that you use grid points $x_0, x_1, \dots, x_m, x_{m+1}$ where $x_i = a + ih$, $i = 0, 1, \dots, m, m+1$ and

$$h = \frac{b-a}{m+1}$$

so that you have a solution vector with exactly m components. Write your codes for general case (ensure flexibility for all parameters and functions involved). Test your code for the case where

$$\sigma = \frac{1}{10}, \; u_A = 0, \; u_B = 4\cos(4)\sin(t), \; a = 0, \; b = 2,$$
$$f(t,x) = \cos(t)\sin(x^2) - 2\sigma\sin(t)[\cos(x^2) - 2x^2\sin(x^2)], \; u_0(x) = 0.$$

For this particular case, the exact solution is $u_{exact}(t,x) = \sin(t)\sin(x^2)$. Evaluate the error

$$e(T) = \|u_{num}(T,x) - u_{exact}(T,x)\|_\infty$$

for $T = 2$. Plotting the solution together with the exact solution at every time step is helpful. You can add commands such as `pause(0.1)` after each plot so that you can observe how the numerical solution evolves. Fill the following table with error values, as well as k values and your observations:

[1]Note that in order to keep my notation consistent, I have swapped the τ's for k's in this exercise.

Method	h	k	$e(T)$	***Observation***
explicit	0.02	0.1	2.6113×10^{35}	*unstable behavior*
explicit	0.02	0.05	3.1948×10^{62}	*unstable behavior*
explicit	0.02	0.025	5.0087×10^{104}	*unstable behavior*
explicit	0.02	$(0.02)^2/(2\sigma)$	0.0441	*stable behavior*
explicit	0.01	0.1	1.6777×10^{47}	*unstable behavior*
explicit	0.01	0.05	3.5926×10^{86}	*unstable behavior*
explicit	0.01	0.025	4.4116×10^{153}	*unstable behavior*
explicit	0.01	$(0.01)^2/(2\sigma)$	0.0221	*stable behavior*
explicit	0.005	$(0.005)^2/(2\sigma)$	0.0111	*stable behavior*
implicit	0.02	0.1	0.1414	*stable behavior*
implicit	0.02	0.05	0.0673	*stable behavior*
implicit	0.02	0.025	0.0408	*stable behavior*
implicit	0.01	0.1	0.1434	*stable behavior*
implicit	0.01	0.05	0.0691	*stable behavior*
implicit	0.01	0.025	0.0333	*stable behavior*
implicit	0.005	0.1	0.1444	*stable behavior*
implicit	0.005	0.05	0.0700	*stable behavior*
implicit	0.005	0.025	0.0341	*stable behavior*
implicit	0.0025	0.1	0.1448	*stable behavior*
implicit	0.0025	0.05	0.0705	*stable behavior*
implicit	0.0025	0.025	0.0346	*stable behavior*

Table 1: Table of errors for both the explicit and implicit Euler methods. The values I had to determine are in cyan color.

Solution. We start by discretizing (first in space only) the equation $u_t = \sigma u_{xx} + f$:

$$u_t = \frac{\sigma}{h^2}\left[U_{i-1} - 2U_i + U_{i+1}\right] + f_i, \qquad i = 1, \dots, m. \tag{12a}$$

We are also imposing the following Neumann boundary conditions:

$$u_A(t) = 0 = \frac{U_1 - U_0}{h} \quad\Longrightarrow\quad U_0 = U_1; \tag{13a}$$

$$u_B(t) = 4\cos(4)\sin(t) = \frac{U_{m+1} - U_m}{h} \quad\Longrightarrow\quad U_{m+1} = 4h\cos(4)\sin(t) + U_m. \tag{13b}$$

Next, discretizing in time Eq. (12a) we get the *Forward Time Centered Space* (FTCS) scheme if we discretize the time-derivative using Forward Euler or, alternatively, we get the *Backward Time Centered Space* (BTCS) scheme if we discretize the time-derivative using the Backward Euler method:

$$\frac{U_i^{n+1} - U_i^n}{k} = \frac{\sigma}{h^2}\left[U_{i-1}^n - 2U_i^n + U_{i+1}^n\right] + f_i^n; \tag{FTCS}$$

$$\frac{U_i^{n+1} - U_i^n}{k} = \frac{\sigma}{h^2}\left[U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}\right] + f_i^{n+1}. \tag{BTCS}$$

Now, letting $\Psi \equiv (k\sigma)/h^2$, we have

$$\boxed{\begin{aligned} U_i^{n+1} &= \Psi U_{i-1}^n + (1-2\Psi)U_i^n + \Psi U_{i+1}^n + kf_i^n \qquad (12b) \\ U_i^{n+1} &= U_i^n + \Psi\left[U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}\right] + kf_i^{n+1} \qquad (12c) \end{aligned}}$$

Eq. (12b) is explicit and can be computed directly from this expression, getting results at the next time-step in terms of results from the previous time-step. Care must be taken, however, at the boundaries; looking at the Neumann boundary conditions (13), we have

$$U_1^{n+1} = (1-\Psi)U_1^n + \Psi U_2^n + kf_1^n$$

since $U_0^n = U_1^n$. Similarly, since $U_{m+1}^n = 4h\cos(4)\sin(t_0 + kn) + U_m^n$, [2]

$$U_m^{n+1} = \Psi U_{m-1}^n + (1-\Psi)U_m^n + 4h\Psi\cos(4)\sin(t_0 + kn) + kf_m^n.$$

The following Matlab code applies the FTCS scheme (12b) to the heat equation (11):

```
%Forward Euler function
function err = ForwardEuler(a, b, h, k, t_0, T, sgm)

    m = (b-a)/h - 1;

    %function f & exact solution u_exact
    f  = @(t,x) cos(t) * sin(x.^2) - 2 * sgm * sin(t)
         * ( cos(x.^2) - 2 * x.^2 * sin(x.^2)   );
    u_true  = @(x) sin(T) .* sin(x.^2);
    u_true_vec = zeros(m,1);  %vectorize true solution...initializing

    %------------------------------------------
    %         FORWARD EULER CODE
    %------------------------------------------

    u_0 = zeros(m,1);           %Initial condition
    u = u_0;                    %initialize solution vector
    Psi = (k*sgm)/(h^2);

    %use ceiling in case t_0 + nk never equals T (it_max) exactly
    it_max = ceil( (T - t_0)/k );

    for n = 1 : it_max

        Xi = 4*h*cos(4) * sin(t_0+n*k);   %\Xi function from Neumann BC

        for i = 1 : m
            if i == 1
                u(i) = (1 - Psi) * u_0(i) + Psi * u_0(i+1)
                       + k * f(t_0 + n*k, a + i*h);

```

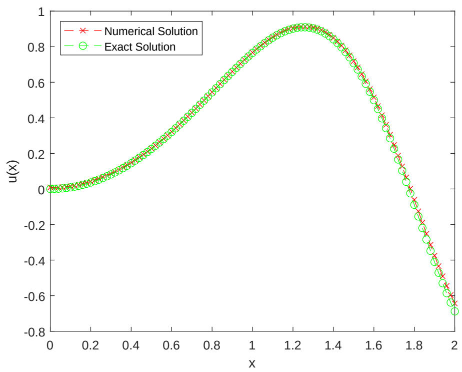
[2] t_0 denotes the initial time; it is typically taken to be $t_0 = 0$ (and we certainly do so in our code).

```
            elseif i == m
                u(i) = Psi * u_0(i-1) +  (1 - Psi) * u_0(i) + Xi*Psi
                       + k *  f(t_0 + n*k, a + i*h);
            else
                u(i) = Psi * u_0(i-1) +  (1 - 2*Psi) * u_0(i) + Psi
                       * u_0(i+1)  + k *  f(t_0 + n*k, a + i*h);
            end

            if n == it_max
                u_true_vec(i) = u_true(a + i*h);
            end
        end

        if n == it_max
            %extend solution to include boundaries
            u_true_vec_full = [u_true(a); u_true_vec; u_true(b)];
            u_full = [u(1); u; Xi + u(m)];
            %global error (output of function)
            err = norm(u_full - u_true_vec_full, Inf);
        end

        u_0 = u;  %update u_0 value for next iteration

    end

    %----------------------------------------------
    %     END OF FORWARD EULER CODE
    %----------------------------------------------

end
```

The output errors from this code are shown on Table 1. As expected, the only combinations of $k-h$ values that yield a convergent behavior are the ones that satisfy the convergence criterion

$$\frac{k\sigma}{h^2} \leq \frac{1}{2}. \tag{14}$$

In order to satisfy this condition for the slots on the table where we had to find convergent behavior, I set $k = h^2/(2\sigma)$. For instance, the following figure shows the case $h = 0.02$ with $k = (0.02)^2/(2\sigma)$:



We now tackle the BTCS scheme. Note that, unlike in the FTCS case, Eq. (12c) is implicit and cannot be evaluated in a similar manner as (12b); instead we must rewrite it as

$$-\Psi U_{i-1}^{n+1} + (1+2\Psi)U_i^{n+1} - \Psi U_{i+1}^{n+1} - kf_i^{n+1} = U_i^n, \tag{12d}$$

which is now in the form

$$A\boldsymbol{u} - k\boldsymbol{f} = \boldsymbol{b},$$

where

$$\underbrace{\begin{bmatrix} (1+2\Psi) & -\Psi & & & & \\ -\Psi & (1+2\Psi) & -\Psi & & & \\ & -\Psi & (1+2\Psi) & -\Psi & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\Psi & (1+2\Psi) & -\Psi \\ & & & & -\Psi & (1+2\Psi) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} U_1^{n+1} \\ U_2^{n+1} \\ \vdots \\ U_i^{n+1} \\ \vdots \\ U_m^{n+1} \end{bmatrix}}_{\boldsymbol{u}} - k \underbrace{\begin{bmatrix} f_1^{n+1} \\ f_2^{n+1} \\ \vdots \\ f_i^{n+1} \\ \vdots \\ f_m^{n+1} \end{bmatrix}}_{\boldsymbol{f}} = \underbrace{\begin{bmatrix} U_1^n + \Psi U_0^{n+1} \\ U_2^n \\ \vdots \\ U_i^n \\ \vdots \\ U_m^n + \Psi U_{m+1}^{n+1} \end{bmatrix}}_{\boldsymbol{b}}.$$

Admittedly, this expression is not yet in a completely satisfying form because of the U_i^{n+1} terms appearing on the first and last elements of the vector $\boldsymbol{b}$. We can remedy the situation by plugging both Neumann boundary conditions (13) into the U_i^{n+1} terms on the LHS of Eq. (12d). Since $U_0^{n+1} = U_1^{n+1}$,

$$-\Psi U_0^{n+1} + (1+2\Psi)U_1^{n+1} - \Psi U_2^{n+1} = (1+\Psi)U_1^{n+1} - \Psi U_2^{n+1},$$

and similarly, since $U_{m+1}^{n+1} = 4h\cos(4)\sin(t_0 + k(n+1)) + U_m^{n+1}$,

$$-\Psi U_{m-1}^{n+1} + (1+2\Psi)U_m^{n+1} - \Psi U_{m+1}^{n+1} = -\Psi U_{m-1}^{n+1} + (1+\Psi)U_m^{n+1} - 4h\Psi\cos(4)\sin(t_0 + k(n+1)).$$

Hence we end up with the final system

$$\begin{bmatrix} (1+\Psi) & -\Psi & & & & \\ -\Psi & (1+2\Psi) & -\Psi & & & \\ & -\Psi & (1+2\Psi) & -\Psi & & \\ & & \ddots & \ddots & \ddots & \\ & & & -\Psi & (1+2\Psi) & -\Psi \\ & & & & -\Psi & (1+\Psi) \end{bmatrix} \begin{bmatrix} U_1^{n+1} \\ U_2^{n+1} \\ \vdots \\ U_i^{n+1} \\ \vdots \\ U_m^{n+1} \end{bmatrix} = \begin{bmatrix} U_1^n + kf_1^{n+1} \\ U_2^n + kf_2^{n+1} \\ \vdots \\ U_i^n + kf_i^{n+1} \\ \vdots \\ U_m^n + \Xi + kf_m^{n+1} \end{bmatrix}, \tag{12e}$$

with $\Xi \equiv 4h\Psi\cos(4)\sin(t_0 + k(n+1))$. This implicit scheme is implemented in the following Matlab code:

```
%Backward Euler function
function err = BackwardEuler(a, b, h, k, t_0, T, sgm)

    m = (b-a)/h - 1;
    Psi = (k*sgm)/(h^2);

    %Generate the matrix A to be applied in BTCS:
    %(could also use sparse allocation)
    A = zeros(m);  %initialize mxm matrix
    A(1,1) = 1+Psi;
    A(m,m) = 1+Psi;

    for i = 1:m
        for j = 1:m
            if (i == j) &&  ( (i ~= 1) && (i ~= m) )
                A(i,i) = 1 + 2*Psi;
            elseif (j == i+1) || (i == j+1)
                A(i,j) = -Psi;
            end
        end
    end

    %function f & exact solution u_exact
    f  = @(t,x) cos(t) * sin(x.^2) - 2 * sgm * sin(t)
          * ( cos(x.^2) - 2 * x.^2 * sin(x.^2)   );
    u_true  = @(x) sin(T) .* sin(x.^2);
    u_true_vec = zeros(m,1);   %vectorize true solution...initializing
    rhs = zeros(m,1);          %initialize rhs of Eq 12e)


    %------------------------------------------
    %        BACKWARD EULER CODE
    %------------------------------------------

    u_0 = zeros(m,1);                    %Initial condition
    it_max = ceil( (T - t_0)/k );

    for n = 1 : it_max

        %\Xi function from Neumann BC
        Xi = 4*h*Psi*cos(4)*sin(t_0+(n+1)*k);

        for i = 1 : m
            if i == m
                rhs(i) = u_0(i) + Xi + k * f(t_0 + (n+1)*k, a + i*h);
            else
                rhs(i) = u_0(i) + k * f(t_0 + (n+1)*k, a + i*h);
            end

            if n == it_max
                u_true_vec(i) = u_true(a + i*h);
            end
        end

```

```
        u = A\rhs;          %solve Au = rhs

        if n == it_max
            %extend solution to include boundaries
            u_true_vec_full = [u_true(a); u_true_vec; u_true(b)];
            u_full = [u(1); u; 4*h*cos(4) * sin(t_0+n*k) + u(m)];
            %global error (output of function)
            err = norm(u_full - u_true_vec_full, Inf);
        end

        u_0 = u;  %update u_0 value for next iteration

    end

    %-----------------------------------------------
    %      END OF BACKWARD EULER CODE
    %-----------------------------------------------

end
```

The output errors from this code are also shown on Table 1. All possible combinations of $k - h$ values given on the table yield a convergent behavior for this implicit scheme, which shows a vast improvement over FTCS. □

Problem 3. *Modify* `heat_CN.m` *to solve the heat equation for* $-1 \leq x \leq 1$ *with step function initial data*

$$u(0,x) = \begin{cases} 1 & \text{if } x < 0 \\ 0 & \text{if } x \geq 0 \end{cases}. \tag{15}$$

With appropriate Dirichlet boundary conditions, the exact solution is

$$u(t,x) = \frac{1}{2}\,\mathrm{erfc}\left(x/\sqrt{4\kappa t}\right), \tag{16}$$

where erfc is the complementary error function

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-z^2}\, dz.$$

a) *Test this routine for $m = 39$ and $\Delta t = 4\Delta x$. Note that there is an initial rapid transient decay of the high wave numbers that is not captured well with this size time step. Include plots of the solution.*

b) *How small do you need to take the time step to get reasonable results? For a suitably small time step, explain why you get much better results by using $m = 38$ than $m = 39$. Include plots of the solution in your explanation (you might have to zoom in on your plots).*

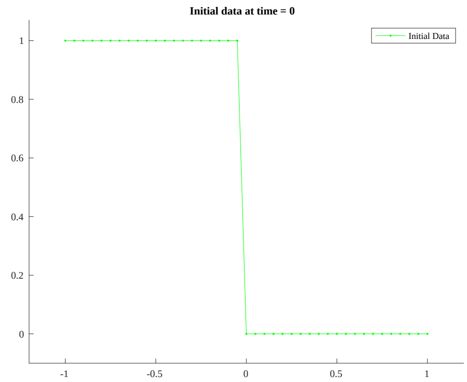
c) *What is the observed order of accuracy as $\Delta t \to 0$ when $\Delta t = \alpha \Delta x$ with α suitably small and m even? Create a table of errors in the L^∞ norm when $m = 40, 80, 160,$ and 320. You can use the function `error_table.m` to create the table.*

Solution. Here is the modified `heat_CN.m` code:

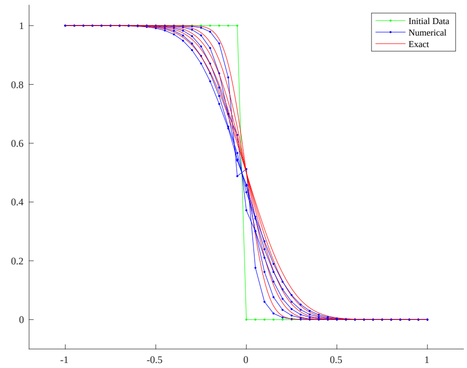
```
function [h,k,error] = CNmod(m)

clf             %clear graphics
hold on         %Put all plots on the same graph (comment out if desired)

ax = -1;
bx = 1;
kappa = .02;       %heat conduction coefficient
tfinal = 1;        %final time

h = (bx-ax)/(m+1);
x = linspace(ax,bx,m+2)';

k = 4*h;                     %time step

nsteps = round(tfinal/k);    %number of time steps
nplot = 1;                   %plot solution every nplot time steps

if abs(k*nsteps - tfinal) > 1e-5
   % The last step won't go exactly to tfinal.
   disp(' ')
   display(['WARNING *** k does not divide tfinal, k = ', num2str(k)]);
   disp(' ')
end

%True solution
utrue = @(t,x) 0.5 * erfc( x/(sqrt(4*kappa*t)) );

%Set up initial data (step function)
syms z;
y = piecewise(z<0, 1, z>=0, 0) ;
f = symfun(y,z);
u_0 = f(x);
u_0 = double(u_0);  %convert from symbolic to numerical (double precision)

% Each time step we solve MOL system U' = AU + g using the Trapezoidal method

% set up matrices:
r = (1/2) * kappa* k/(h^2);
e = ones(m,1);
A = spdiags([e -2*e e], [-1 0 1], m, m);
A1 = eye(m) - r * A;
A2 = eye(m) + r * A;

% initial data on fine grid for plotting:
xfine = linspace(ax,bx,1001);
```

```
% initialize u and plot initial data:
u = u_0;
plot(x,u,'g.-')
legend('Initial Data')
title('Initial data at time = 0')

input('Hit <return> to continue  ');

%main time-stepping loop:
tn = 0;
for n = 1:nsteps
     tnp = tn + k;   % = t_{n+1}

     %boundary values u(0,t) and u(1,t) at times tn and tnp:
     g0n = u(1);
     g1n = u(m+2);
     g0np = utrue(tnp,ax);
     g1np = utrue(tnp,bx);

     %compute right hand side for linear system:
     uint = u(2:(m+1));   % interior points (unknowns)
     rhs = A2*uint;
     % fix-up right hand side using BC's (i.e. add vector g to A2*uint)
     rhs(1) = rhs(1) + r*(g0n + g0np);
     rhs(m) = rhs(m) + r*(g1n + g1np);

     % solve linear system:
     uint = A1\rhs;

     % augment with boundary values:
     u = [g0np; uint; g1np];

     % plot results at desired times:
     if mod(n,nplot)==0 || n==nsteps
        ufine = utrue(tnp, xfine);
        plot(x,u,'b.-', xfine,ufine,'r')
        legend('Initial Data','Numerical','Exact')
        title(['t = ', num2str(tnp), ' after ', num2str(n),
               ' time steps with ', num2str(m+2), ' grid points']);
        error = max(abs(u-utrue(tnp,x)));
        display(['at time t = ', num2str(tnp), ' max error = ',
                 num2str(error)])
        if n<nsteps
            input('Hit <return> to continue  ');
        end
     end

     tn = tnp;   % for next time step

end
```

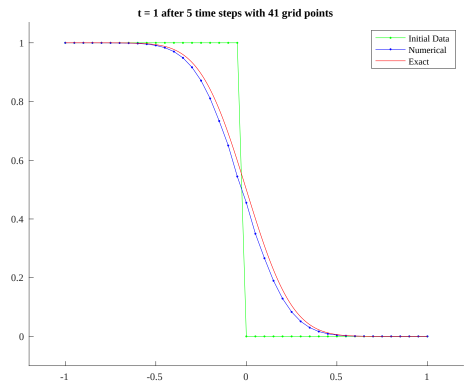
The initial data given by the step function is shown on the graph:



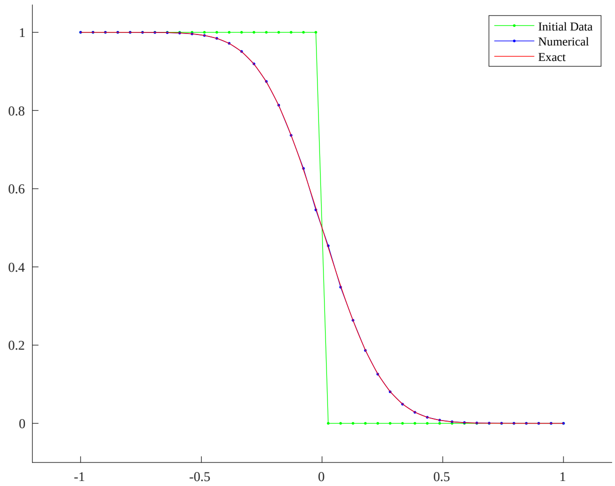
The following plot, on the other hand, shows the solutions computed at each step until we reach $t_{\text{final}} = 1$:



The figure shows that our numerical solution initially does not behave too well at the discontinuity at $x = 0$; however, if we focus on the last time step at $t_{\text{final}} = 1$ we see that the solution's behavior improves:



As we shall now see, the numerical solution improves even more if we use an even number of grid points, since in that case $x = 0$ is not part of the grid for uniform h. Here is the result from using $m = 38$ grid points:



For implicit methods (which is the case for the Crank-Nicolson scheme used in this exercise) there is no restriction on the time-stepping. However, in order to achieve comparable resolution in space and time it is reasonable to use $k \sim h/\kappa$; this is a vast improvement over explicit methods, which require $k \sim h^2/\kappa$. Notwithstanding, we should not use $k = h/\kappa$ for this particular problem, because $\kappa = 0.02$ is too small and that makes a single time step $k \approx 2.5$ go beyond our time limit $t_{\text{final}} = 1$. Instead we notice that the choice $k = 4h$ that was in the original `heat_CN.m` code worked quite well in our analysis, especially for even m grid points. Hence taking $k \sim h$ seems reasonable, and so we simply set $\alpha = 1$, so that $k = h$, and see what happens as $k \longrightarrow 0$. The following table show errors (in the L^∞ norm) for increasing values of m (and hence decreasing h and k values):

m	$\|\|e\|\|_\infty$
40	0.0018621
80	0.00047794
160	0.0001211
320	3.0558×10^{-5}

Choosing any two error values, say ${}^{m=160}\|e\|_\infty = 0.0001211$ and ${}^{m=80}\|e\|_\infty = 0.00047794$, and looking at the ratio of the corresponding k-values

$$\frac{k_{m=160}}{k_{m=80}} = \frac{h_{m=160}}{h_{m=80}} = \frac{\frac{2}{161}}{\frac{2}{81}} \approx \frac{1}{2}$$

we see that cutting k in half yields

$$\frac{0.0001211}{0.00047794} = 0.253379 \approx \frac{1}{4},$$

which shows that we have $\mathcal{O}(k^2)$ convergence. This was to be expected, since the Crank-Nicolson scheme is second-order accurate in both space and time. □