Consider again the one-dimensional heat equation, $u_t = u_{xx}$, $0 < x < 1, t > 0$, subject to homogeneous Dirichlet boundary conditions:

$$u(0,t) = 0, \quad u(1,t) = 0,$$

and the initial condition:

$$u(x,0) = f(x) = \begin{cases} 2x & 0 \le x \le \frac{1}{2} \\ 2-2x & \frac{1}{2} \le x \le 1 \end{cases}.$$

This time, we approximate the solution to a value of $u(x,t)$ using the *implicit* finite difference scheme consisting of a backward difference in time and centered difference in space.

Suppose we choose $N = 4$ intervals on $[0,1]$ and set $x_0 = 0$, $x_1 = \frac{1}{4}$, $x_2 = \frac{1}{2}$, $x_3 = \frac{3}{4}$ and $x_4 = 1$. Let U_j^m denote an approximation to the exact solution $u(x_j, t_m)$. If we set $t_0 = 0$ then the implicit finite difference scheme based on centered differences in space and a backward difference in time (see lecture notes) yields 3 equations for approximations to $u(x,t)$ at the interior space nodes, at each new level t_m. We have:

$$U_j^m = -\nu U_{j-1}^{m+1} + (1+2\nu) U_j^{m+1} - \nu U_{j+1}^{m+1}, \quad j = 1:3, \quad m = 1,2,\ldots$$

where $\nu = \frac{k}{h^2}$. The boundary conditions give values for the end points at each time level:

$$U_0^m = U_4^m = 0, \quad m = 1,2,\ldots$$

With $h = \frac{1}{4}$, we obtain three equations for the unknown values $U_1^{m+1}, U_2^{m+1}, U_3^{m+1}$ at each new time step:

$$\begin{array}{cccc} (1+32k)\,U_1^{m+1} & -16kU_2^{m+1} & & = U_1^m \\ -16kU_1^{m+1} & +(1+32k)\,U_2^{m+1} & -16kU_3^{m+1} & = U_2^m \\ & -16kU_2^{m+1} & +(1+32k)\,U_3^{m+1} & = U_3^m \end{array}$$

$$\Rightarrow \begin{pmatrix} 1+32k & -16k & 0 \\ -16k & 1+32k & -16k \\ 0 & -16k & 1+32k \end{pmatrix} \begin{pmatrix} U_1^{m+1} \\ U_2^{m+1} \\ U_3^{m+1} \end{pmatrix} = \begin{pmatrix} U_1^m \\ U_2^m \\ U_3^m \end{pmatrix}.$$

If we set $t_0 = 0$ and choose $k = 0.01$ and notice that the initial condition gives:

$$U_1^0 = f(x_1) = \frac{1}{2}, \quad U_2^0 = f(x_2) = 1, \quad U_3^0 = f(x_3) = \frac{1}{2},$$

we then have to solve the 3×3 linear system

$$\begin{pmatrix} 1.32 & -0.16 & 0 \\ -0.16 & 1.32 & -16k \\ 0 & -0.16 & 1.32 \end{pmatrix} \begin{pmatrix} U_1^1 \\ U_2^1 \\ U_3^1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 1 \\ \frac{1}{2} \end{pmatrix}.$$

(by hand or using MATLAB) for the solution at $t_1 = 0.01$. You can attempt to solve this system by hand or use the MATLAB code `trisolve.m` by typing:

```
u1=trisolve(3,-0.16,1.32,-0.16,[1/2;1;1/2]).
```

The solution is $U_1^1 = 0.4849$, $U_2^2 = 0.8751$, $U_3^2 = 0.4849$. To obtain the approximations at the next time step $t_2 = 0.02$, we have to solve another tri-diagonal system with the same coefficient matrix but where the right-hand side vector is the solution at the first time step. We can compute this via:

```
u2=trisolve(3,-0.16,1.32,-0.16,[0.4849;0.8751;0.4849])
```

and so on.

Example. Suppose we repeat the experiment we performed on the last handout with the **explicit** finite difference scheme. With the earlier method, we saw that choosing $k = 0.0013$ in combination with $h = \frac{1}{20}$ led to unstable results. In the figures below we plot the approximations obtained with these values of h and k using the new **implicit** scheme at time steps $t_1 = 0.0013, t_{25} = 0.0325$ and $t_{50} = 0.0650$. Notice that there are no oscillations now.

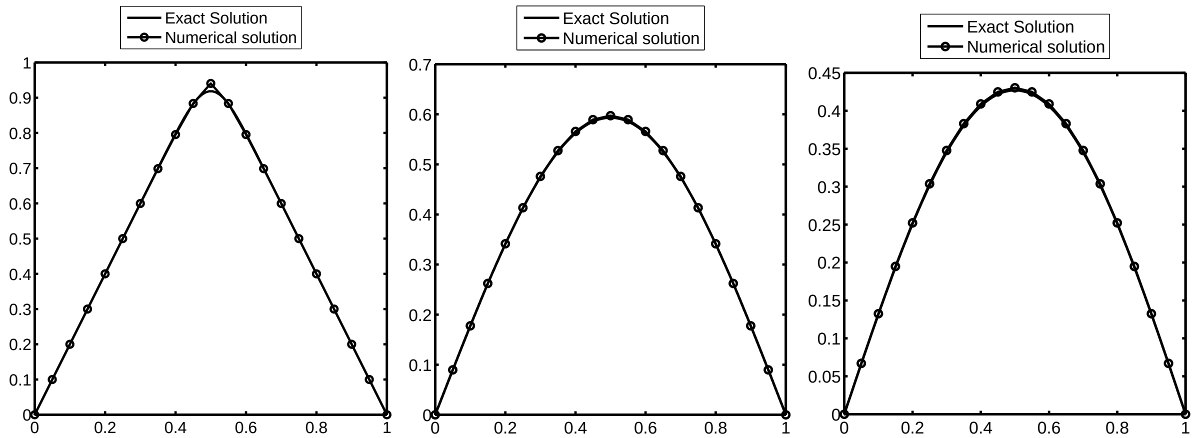

Figure 1: Exact solution and numerical approximations to the solution at time steps: 1, 25, and 50 (left to right). Implicit finite difference scheme, $h = \frac{1}{20}$, $k = 0.0013$, $\nu = 0.52$.

In fact, there are no restrictions now on the choice of k with respect to h. The method is stable for any value of $\nu = \frac{k}{h^2}$.

To reproduce the above results, you will need to recursively solve a tri-diagonal system of equations, changing the right-hand side vector in each case to the solution in the previous step. The MATLAB code `heat_eq_implicit_fd` will do this for you. Download it and perform the above experiment. Eg. to generate the approximation at t_{25} with $k = 0.0013$ and $N = 20$ type:

```
[u_approx,u_exactx]=heat_eq_implicit_fd(20,0.0013,25);
```