Tufts University - Department of Mathematics
Math 253 - Homework 1 Solutions

1. Let $a > 0$ and consider the one-way wave equation, $u_t + au_x = 0$. For both the Forward-Central,

$$\frac{v_{k,l+1} - v_{k,l}}{h_t} + a\frac{v_{k+1,l} - v_{k-1,l}}{2h_x} = 0,$$

and Crank-Nicholson,

$$\frac{v_{k,l+1} - v_{k,l}}{h_t} + \frac{a}{2}\left(\frac{v_{k+1,l+1} - v_{k-1,l+1}}{2h_x} + \frac{v_{k+1,l} - v_{k-1,l}}{2h_x}\right) = 0,$$

finite-difference schemes, investigate their consistency and stabilty. In particular, show that the truncation error for each scheme goes to zero as h_t, $h_x \to 0$ and perform a von Neumann stability analysis for each scheme.

Answer: To compute the truncation error for Forward-Central, we plug in the true solution and expand with Taylor's Theorem:

$$\begin{aligned}
\tau_{k,l} &= \frac{u(x_k, t_{l+1}) - u(x_k, t_l)}{h_t} + a\frac{u(x_{k+1}, t_l) - u(x_{k-1}, t_l)}{2h_x} \\
&= \frac{1}{h_t}(u(x_k, t_l) + u_t(x_k, t_l)h_t + u_{tt}(x_k, \varsigma)\frac{{h_t}^2}{2} - u(x_k, t_l)) \\
&+ \frac{a}{2h_x}(u(x_k, t_l) + u_x(x_k, t_l)h_x + u_{xx}(x_k, t_l)\frac{{h_x}^2}{2} + u_{xxx}(\xi^+, t_l)\frac{{h_x}^3}{6} \\
&- u(x_k, t_l) + u_x(x_k, t_l)h_x - u_{xx}(x_k, t_l)\frac{{h_x}^2}{2} + u_{xxx}(\xi^-, t_l)\frac{{h_x}^3}{6}) \\
&= u_t(x_k, t_l) + u_{tt}(x_k, \varsigma)\frac{h_t}{2} + au_x(x_k, t_l) + \frac{a}{12}(u_{xxx}(\xi^+, t_l) + u_{xxx}(\xi^-, t_l)){h_x}^2
\end{aligned}$$

Since, $u_t(x_k, t_l) + au_x(x_k, t_l) = 0$, the expression simplifies to

$$\begin{aligned}
&= u_{tt}(x_k, \varsigma)\frac{h_t}{2} + \frac{a}{12}(u_{xxx}(\xi^+, t_l) + u_{xxx}(\xi^-, t_l)){h_x}^2 \\
&= O(h_t, {h_x}^2)
\end{aligned}$$

Thus, as $h_t, h_x \to 0$, $\tau_{k,l} \to 0$.

Similarly, we perform the same analysis for Crank-Nicholson:

$$\begin{aligned}
\tau_{k,l} &= \frac{u(x_k,t_{l+1})-u(x_k,t_l)}{h_t} + \frac{a}{4h_x}\left(u(x_{k+1},t_{l+1})-u(x_{k-1},t_{l+1})+u(x_{k+1},t_l)-u(x_{k-1},t_l)\right)\\
&= \frac{1}{h_t}(u(x_k,t_l)+u_t(x_k,t_l)h_t+u_{tt}(x_k,t_l)\frac{{h_t}^2}{2}+u_{ttt}(x_k,t_l)\frac{h_t^3}{6}+O(h_t^4)-u(x_k,t_l))\\
&+\frac{a}{4h_x}\Bigg(u(x_k,t_l)+u_x(x_k,t_l)h_x+u_t(x_k,t_l)h_t+u_{xx}(x_k,t_l)\frac{{h_x}^2}{2}+u_{tt}(x_k,t_l)\frac{{h_t}^2}{2}+u_{xt}(x_k,t_l)h_xh_t\\
&+u_{xxx}(x_k,t_l)\frac{{h_x}^3}{6}+u_{xxt}(x_k,t_l)\frac{h_x^2h_t}{2}+u_{xtt}(x_k,t_l)\frac{h_xh_t^2}{2}+u_{ttt}(x_k,t_l)\frac{{h_t}^3}{6}+O(h_x^4,h_x^3h_t,h_x^2h_t^2,h_xh_t^3,h_t^4)\\
&-u(x_k,t_l)+u_x(x_k,t_l)h_x-u_t(x_k,t_l)h_t-u_{xx}(x_k,t_l)\frac{{h_x}^2}{2}-u_{tt}(x_k,t_l)\frac{{h_t}^2}{2}+u_{xt}(x_k,t_l)h_xh_t\\
&+u_{xxx}(x_k,t_l)\frac{{h_x}^3}{6}-u_{xxt}(x_k,t_l)\frac{h_x^2h_t}{2}+u_{xtt}(x_k,t_l)\frac{h_xh_t^2}{2}-u_{ttt}(x_k,t_l)\frac{{h_t}^3}{6}+O(h_x^4,h_x^3h_t,h_x^2h_t^2,h_xh_t^3,h_t^4)\\
&+u(x_k,t_l)+u_x(x_k,t_l)h_x+u_{xx}(x_k,t_l)\frac{{h_x}^2}{2}+O(h_x^3)\\
&-u(x_k,t_l)+u_x(x_k,t_l)h_x-u_{xx}(x_k,t_l)\frac{{h_x}^2}{2}+O(h_x^3)\Bigg)\\
&= u_t(x_k,t_l)+u_{tt}(x_k,t_l)\frac{{h_t}^2}{2}+u_{ttt}(x_k,t_l)\frac{h_t^3}{6}+O(h_t^4)\\
&+au_x(x_k,t_l)+\frac{a}{2}u_{xt}(x_k,t_l)h_t+\frac{a}{12}u_{xxx}(x_k,t_l)h_x^2+\frac{a}{4}u_{xtt}(x_k,t_l)h_t^2++O(h_x^3,h_x^2h_t,h_xh_t^2,h_t^3,\frac{h_t^4}{h_x})
\end{aligned}$$

Now, again $u_t(x_k,t_l)+au_x(x_k,t_l)=0 \Rightarrow u_{tt}=-au_{xt}$:

$$\begin{aligned}
\tau_{k,l} &= u_{tt}(x_k,t_l)\frac{h_t}{2}+\frac{a}{2}u_{xt}(x_k,t_l)h_t+O(h_t^2)+O(h_x^2)\\
&= -\frac{a}{2}u_{xt}(x_k,t_l)h_t+\frac{a}{2}u_{xt}(x_k,t_l)h_t+O(h_t^2,h_x^2)\\
&= O(h_t^2,h_x^2)
\end{aligned}$$

Thus, as $h_t\to 0$ and $h_x\to 0$, $\tau_{k,l}\to 0$.
Both schemes are consistent.

Next, to test stability we perform a Von Neumann analysis. We know the error satisfies the same finite-difference scheme, and assume the form, $e_{k,l}=\hat{e}_l^{(j)}\exp(ijkh_x)$. For Forward-Central, let $\lambda=\frac{ah_t}{h_x}$, then,

$$\begin{aligned}
e_{k,l+1} &= e_{k,l}+\frac{\lambda}{2}\left(e_{k-1,l}-e_{k+1,l}\right),\\
\Rightarrow \hat{e}_{l+1}^{(j)}\exp(ijkh_x) &= \hat{e}_l^{(j)}\exp(ijkh_x)+\frac{\lambda}{2}\left(\hat{e}_l^{(j)}\exp(ijkh_x)\exp(-ijh_x)-\hat{e}_l^{(j)}\exp(ijkh_x)\exp(ijh_x)\right)\\
\Rightarrow \hat{e}_{l+1}^{(j)} &= \hat{e}_l^{(j)}+\frac{\lambda}{2}\left(\hat{e}_l^{(j)}\exp(-ijh_x)-\hat{e}_l^{(j)}\exp(ijh_x)\right)\\
&= \hat{e}_l^{(j)}\left(1-i\lambda\sin(jh_x)\right)\\
\Rightarrow s(j)=\frac{\hat{e}_{l+1}^{(j)}}{\hat{e}_l^{(j)}} &= 1-i\lambda\sin(jh_x)\\
\Rightarrow |s(j)|^2 &= \left|1+\lambda^2\right|>1
\end{aligned}$$

Thus, the scheme is *not* von Neumann stable and, therefore, not stable! Thus, we do not expect this scheme to be convergent, even though it is consistent.

For Crank-Nicholson we obtain:

$$
\begin{aligned}
e_{k,l+1} &= e_{k,l} - \frac{\lambda}{4}\left(e_{k+1,l+1} - e_{k-1,l+1} + e_{k+1,l} - e_{k-1,l}\right), \\
\Rightarrow \hat{e}_{l+1}^{(j)} &= \hat{e}_{l}^{(j)} - \frac{\lambda}{4}\left(\hat{e}_{l+1}^{(j)}\exp(ijh_x) - \hat{e}_{l+1}^{(j)}\exp(-ijh_x) + \hat{e}_{l}^{(j)}\exp(ijh_x) - \hat{e}_{l}^{(j)}\exp(-ijh_x)\right) \\
\left(1 + i\frac{\lambda}{2}\sin(jh_x)\right)\hat{e}_{l+1}^{(j)} &= \left(1 - i\frac{\lambda}{2}\sin(jh_x)\right)\hat{e}_{l}^{(j)} \\
\Rightarrow |s(j)|^2 &= \frac{|1 - i\frac{\lambda}{2}\sin(jh_x)|^2}{|1 + i\frac{\lambda}{2}\sin(jh_x)|^2} = \frac{1 + \frac{\lambda^2}{4}\sin^2(jh_x)}{1 + \frac{\lambda^2}{4}\sin^2(jh_x)} = 1 \\
\Rightarrow |s(j)| &= 1
\end{aligned}
$$

Thus, Crank-Nicholson is unconditionally von-Neumann stable and since the time discretization is not singular, we know that this implies the scheme is stable. Since it is consistent and stable, it will be convergent by the Lax principles.
Note that the fact that the symbol is exactly 1 tell us that the scheme should not have numerical dissipation...more on that later...

2. Consider the one-way wave equation, $u_t + u_x = 0$, with $a = 1$ on a finite interval in space and time, $[0,1] \times [0,1]$. For this domain, we need to specify both an initial condition, $u(x,0) = u_0(x)$ for $0 \le x \le 1$, and a boundary condition, $u(0,t) = u_1(t)$ for $0 \le t \le 1$. To avoid contradiction, note that these two conditions should match at $(0,0)$: $u_0(0) = u_1(0)$.

   Implement the Forward-Central, First-Order Upwind, Crank-Nicholson, and Backward-Central,

$$\frac{v_{k,l+1} - v_{k,l}}{h_t} + a\frac{v_{k+1,l+1} - v_{k-1,l+1}}{2h_x} = 0,$$

   finite-difference schemes for this problem. Note that the Backward-Central scheme is unconditionally von Neumann stable (or feel free to prove this). Also note that for the Crank-Nicholson and Backward-Central schemes, you will need to solve a linear system to compute the numerical solution at time $l+1$ from that at time l. You can use Matlab's '\' command to do this.

   Investigate the convergence of these schemes for various choices of initial and bounday conditions and choices of grid spacings, h_t and h_x. In particular, investigate cases that do and do not conform to the CFL condition for the First-Order Upwind scheme, $h_t \le \frac{h_x}{a}$, or for those found in Problem 1. Also, be aware that the expressions for the truncation error for these schemes are based on assumptions of the smoothness of u. Investigate how the smoothness of the exact solution (reflected in the smoothness of $u_0(x)$ and $u_1(t)$) affects the performance of the numerical schemes.

   Partial Answer:
   Forward Central: We can rewrite the Forward Central scheme as:

$$v_{k,\ell+1} = \frac{h_t}{2h_x}\left(v_{k-1,\ell} - v_{k+1,\ell}\right) + v_{k,l}.$$

   Thus, we can start with the initial condition defined for all $v_{k,0} = u_0(kh_x, 0)$ and left endpoint $v_{0,\ell} = u_1(0, \ell h_t)$, and then use the above formula to find the remaining $v_{k,\ell}$, where $k = 1, \ldots, N_k = \frac{1}{h_x}$ and $\ell = 1, \ldots, N_\ell = \frac{1}{h_t}$.
   Things get a bit tricky at the right end points. For instance, $v_{N_k,\ell}$ depends on $v_{N_k+1,\ell-1}$, for $\ell \ge 2$, but this is not defined on our domain. Two tricks can be used. First, you could just

use a lower order scheme such as upwind, which does not require this value at that point. This may degrade the order of convergence, but since it's only at 1 point, depending on the function, this may not have that much of an affect. Or, we know the characteristics are curves with slope $a = 1$, thus, all information moves along the lines $t = x$. Thus, for example if $h_x = h_t$ then, we should expect $v_{k+1,\ell} = v_{k,\ell-1}$ and we can use this just for the right-end point values. At $\ell = 1$, though, we can just use the fact that we know $u_0(x)$ everywhere. Note, that this should be altered appropriately if $h_x \neq h_t$ and in the code below if $h_x \neq h_t$, the true solution is used instead... Sample code follows:

```
function[V] = ForwardCentral(ht, hx, u0x, u1t)
% This function is implemented to numerically calculate the solution to
% u_t+u_x = 0 on a grid with uniform spacing ht in time and hx in space
% with initial condition functions u_0(x) and u_1(t). Our grid is finite
% [0,1]x[0,1] and we implement the Forward Centeral Discretization.

% NOTE u0x and u1t must be function handles!

%number of nodes in each direction
xnodes = 1/hx +1;
tnodes = 1/ht +1;
lambda = ht/(2*hx);

%zero the grid
V = zeros(tnodes, xnodes);

%initialize the grid
for i=1:xnodes,
    V(1,i) = u0x((i-1)*hx);
end

for j=1:tnodes,
    V(j,1) = u1t((j-1)*ht);
end

%Compute Forward Central approximations
for i=2:tnodes,
    for j=2:xnodes-1,
        V(i,j) = lambda*(V(i-1,j-1) - V(i-1,j+1)) + V(i-1,j);
    end
    % Need correction at the boundary since the 1 forward in space one back
    % in time node does not exist
    if(i~=2)
        if(hx==ht)
            V(i,j+1) = lambda*(V(i-1,j)- V(i-2,j+1)) + V(i-1,j+1);
        else
            V(i,j+1) = lambda*(V(i-1,j) - u0x((xnodes)*hx-(i-2)*ht)) + V(i-1,j+1);
        end
    else
        V(i,j+1) = lambda*(V(i-1,j)- u0x(xnodes*hx)) + V(i-1,j+1);
    end
end
```

First-Order Upwind:
We can rewrite this as:

$$v_{k,\ell+1} = \frac{h_t}{h_x}\left(v_{k-1,\ell} - v_{k,\ell}\right) + v_{k,\ell}.$$

Implementing this is pretty straightforward with no special modifications needed:

```
function[V] = FirstOrderUpwind(ht, hx, u0x, u1t)
% This function is implemented to numerically calculate the solution to
% u_t+u_x = 0 on a grid with uniform spacing ht in time and hx in space
% with initial condition functions u_0(x) and u_1(t). Our grid is finite
% [0,1]x[0,1] and we implement the 1st order upwind Discretization.

% NOTE u0x and u1t must be function handles!

%number of nodes in each direction
xnodes = 1/hx +1;
tnodes = 1/ht +1;
lambda = ht/hx;

%zero the grid
V = zeros(tnodes, xnodes);

%initialize the grid
for i=1:xnodes,
    V(1,i) = u0x((i-1)*hx);
end

for j=1:tnodes,
    V(j,1) = u1t((j-1)*ht);
end

%Compute FirstOrderUpwind approximations
for i=2:tnodes,
    for j=2:xnodes,
        V(i,j) = lambda*(V(i-1,j-1) - V(i-1,j)) + V(i-1,j);
    end
end
```

Crank-Nicholson: This scheme can be rewritten as:

$$-\lambda v_{k-1,\ell+1} + v_{k,\ell+1} + \lambda v_{k+1,\ell+1} = \lambda v_{k-1,\ell} + v_{k,\ell} - \lambda v_{k+1,\ell},$$

where $\lambda = \frac{h_t}{4h_x}$. Again, we can use the trick to deal with the right-end points, but we can not write an explicit scheme as we did before. Thus, we have a matrix equation for all our time steps:

$$\begin{pmatrix} 1 & \lambda & 0 & \dots & & & 0 \\ -\lambda & 1 & \lambda & 0 & \dots & & \vdots \\ 0 & -\lambda & 1 & \lambda & 0 & \dots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \lambda \\ 0 & \dots & \dots & \dots & 0 & -\lambda & 1 \end{pmatrix} \begin{pmatrix} v_{1,\ell+1} \\ v_{2,\ell+1} \\ \vdots \\ \\ \\ \\ v_{N_k,\ell+1} \end{pmatrix} = \begin{pmatrix} \lambda v_{0,\ell} + v_{1,\ell} - \lambda v_{2,\ell} + \lambda v_{0,\ell+1} \\ \lambda v_{1,\ell} + v_{2,\ell} - \lambda v_{3,\ell} \\ \lambda v_{2,\ell} + v_{3,\ell} - \lambda v_{4,\ell} \\ \vdots \\ \\ \\ \\ \vdots \\ \lambda v_{N_k-1,\ell} + v_{N_k,\ell} - \lambda v_* \end{pmatrix}$$

Note that the first row is for the 2nd spatial point as the first $k = 0$ is known by $u(0,t) = u_1(t)$. This is accounted for by the extra term in the first entry of the right-hand side vector. The last entry accounts for the trick to grab the appropriate value at the right-end point for both the current time and the previous time, $v_* = v_{N_k+1,\ell} + v_{N_k+1,\ell+1}$.
The following code builds this matrix system using several functions and uses Matlab's '\' routine to solve the system:

```
function[V] = CrankNicholson(ht, hx, u0x, u1t)
% This function is implemented to numerically calculate the solution to
% u_t+u_x = 0 on a grid with uniform spacing ht in time and hx in space
% with initial condition functions u_0(x) and u_1(t). Our grid is finite
% [0,1]x[0,1] and we implement the Crank Nicholson Discretization.

% NOTE u0x and u1t must be function handles!

%number of nodes in each direction
xnodes = 1/hx +1;
tnodes = 1/ht +1;
lambda = ht/(4*hx);

%zero the grid
V = zeros(tnodes, xnodes);

%initialize the grid
for i=1:xnodes,
    V(1,i) = u0x((i-1)*hx);
end

for j=1:tnodes,
    V(j,1) = u1t((j-1)*ht);
end

% calculate extension of inital condition t=0
extu0x = u0x(xnodes*hx);

P = constructOperatorMatrix(lambda,xnodes);

for i=2:tnodes,
    Vi = calcRightVector(V,i,xnodes,lambda, extu0x,ht,hx,u0x);
    vtemp = P\Vi;
    V(i,2:end) = vtemp;
end

end



function[Vi] = calcRightVector(V, i, xnodes, lambda, extu0x,ht,hx,u0x)
% This function computes the vector on the right side of the linear system
% we are trying to solve for a given index i and the current approximation
% V. NOTE the index i is the current row of x values we are trying to solve
% for so we will want the values at time i-1. NOTE the first entry in the
% right side must include an additional lambda*k_{0,i} by the known
% boundary conditions at the current time, which we do not include in the
% P matrix
Vi = zeros(xnodes-1,1);
for j=1:xnodes-1,
    if(j~=xnodes-1) % Not at right-boundary
        if(j~=1) % Not at left boundary
            Vi(j) = lambda*V(i-1, j) + V(i-1,j+1) - lambda*V(i-1,j+2);
        else % You are at the 2nd spot in space -> guy to left is known due to BC
            Vi(j) = V(i-1,j+1) - lambda*V(i-1,j+2) + lambda*V(i,j) +...
                lambda*V(i-1,j);
        end
    else
        %special treatment for the final xnode since can't reach forward in
        %space only works for hx=ht, so if not we'll just grab true
        %solution
        if (hx==ht)
            if(i~=2)
                Vi(j) = lambda*V(i-1,j)-lambda*V(i-2,j+1)+V(i-1,j+1)-...
                    lambda*V(i-1,j+1);
            else
                Vi(j) = lambda*V(i-1,j)-lambda*extu0x+V(i-1,j+1)-...
                    lambda*V(i-1,j+1);
            end
        else
            Vi(j) = lambda*V(i-1,j)+V(i-1,j+1)-...
                lambda*(u0x((j+1)*hx-(i-2)*ht) + u0x((j+1)*hx-(i-1)*ht));
        end
    end
end
end

function[P] = constructOperatorMatrix(lambda, xnodes)
% This function constructs the matrix to be inverted. It is the same for
% each x level so we construct it once and then keep it to compute with.
% NOTE: The first entry in the matrix must be 1 followed by lambda in the
% second, since we assume we know the left most node at all times

% Size is one less due to boundary condition
P = zeros(xnodes-1, xnodes-1);
for i=1:xnodes-1,
    if(i~=xnodes-1) % Not at right-most spatial point
        if(i~=1) % Not at left-most spatial point
            P(i,i-1) = -lambda;
            P(i,i) = 1;
            P(i,i+1) = lambda;
        else % Left-most spatial point
            P(i,i) = 1;
            P(i,i+1) = lambda;
        end
    else % Right-most spatial point
```

```
        P(i,i) = 1;
        P(i,i-1) = -lambda;
    end
end
end
```

Backward-Central Here the scheme is implicit and is written:

$$-\lambda v_{k-1,\ell+1} + v_{k,\ell+1} + \lambda v_{k+1,\ell+1} = v_{k,\ell},$$

where $\lambda = \frac{h_t}{2h_x}$. Similar to what is done for Crank-Nicholson we get the following matrix system:

$$\begin{pmatrix} 1 & \lambda & 0 & \dots & & & 0 \\ -\lambda & 1 & \lambda & 0 & \dots & & \vdots \\ 0 & -\lambda & 1 & \lambda & 0 & \dots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \lambda \\ 0 & \dots & \dots & \dots & 0 & -\lambda & 1 \end{pmatrix} \begin{pmatrix} v_{1,\ell+1} \\ v_{2,\ell+1} \\ \vdots \\ \\ \\ \\ v_{N_k,\ell+1} \end{pmatrix} = \begin{pmatrix} v_{1,\ell} + \lambda v_{0,\ell+1} \\ v_{2,\ell} \\ v_{3,\ell} \\ \vdots \\ \\ \\ \vdots \\ v_{N_k,\ell} - \lambda v_* \end{pmatrix},$$

where here $v_* = v_{N_k+1,\ell+1}$. In other words, we only need to account for that right-end point at the current time and not previous times.

```
function[V] = BackwardCentral(ht, hx, u0x, u1t)
% This function is implemented to numerically calculate the soltuion to
% u_t+u_x = 0 on a grid with uniform spacing ht in time and hx in space
% with initial condition functions u_0(x) and u_1(t). Our grid is finite
% [0,1]x[0,1] and we implement the Backward Central Discretization.

% NOTE u0x and u1t must be function handles!

%number of nodes in each direction
xnodes = 1/hx +1;
tnodes = 1/ht +1;
lambda = ht/(2*hx);

%zero the grid
V = zeros(tnodes, xnodes);

%initialize the grid
for i=1:xnodes,
    V(1,i) = u0x((i-1)*hx);
end

for j=1:tnodes,
    V(j,1) = u1t((j-1)*ht);
end

P = constructOperatorMatrix(lambda,xnodes);

for i=2:tnodes,
    Vi = calcRightVector(V,i,xnodes,lambda,hx,ht,u0x);
    vtemp = P\Vi;
    V(i,2:end) = vtemp;
end

end

function[Vi] = calcRightVector(V, i, xnodes, lambda,hx,ht,u0x)
% This function computes the vector on the right side of the linear system
% we are trying to solve for a given index i and the current approximation
% V. NOTE the index i is the current row of x values we are trying to solve
% for so we will want the values at time i-1. NOTE the first entry in the
% right side must include an additional lambda*k_{0,i} by the known
% boundary conditions
Vi = zeros(xnodes-1,1);
for j=1:xnodes-1,
    if(j~=xnodes-1)
        if(j~=1)
            Vi(j) = V(i-1,j+1);
        else
            Vi(j) = lambda*V(i,j)+V(i-1,j+1);
        end
    else
        if(hx==ht)
            Vi(j) = V(i-1,j+1)-lambda*V(i-1,j+1);
```

```
        else
            Vi(j) = V(i-1,j+1)-lambda*u0x((j+1)*hx-(i-1)*ht);
        end
    end
end
end

function[P] = constructOperatorMatrix(lambda, xnodes)
% This function constructs the matrix to be inverted. It is the same for
% each x level so we construct it once and then keep it to compute with.
% NOTE: The first entry in the matrix must be 1 followed by lambda in the
% second.
P = zeros(xnodes-1, xnodes-1);
for i=1:xnodes-1,
    if(i~=xnodes-1)
        if(i~=1)
            P(i,i-1) = -lambda;
            P(i,i) = 1;
            P(i,i+1) = lambda;
        else
            P(i,i) = 1;
            P(i,i+1) = lambda;
        end
    else
        P(i,i) = 1;
        P(i,i-1) = -lambda;
    end
end
end
```

Since, you'll be testing out various initial and boundary conditions of your own choosing, I won't show explicit results, but make a few comments. For smooth data, we'd expect non-convergent schemes for Forward-Central and thus, nonsense results. For First-order upwind, we should see convergence as long as $h_t \leq h_x$, the CFL condition for this problem. Otherwise, stability should occur. If measuring the L^2-error of the approximate solution with the true solution, $u_0(x-t)$, we should see the error converging with order, $O(h_x, h_t)$. Meaning if we compute the L^2- error on a grid of size h_x and then again on a grid of size $h_x/2$, the error should reduce by a factor of 2. Similarly, for h_t. For the Crank-Nicholson and Backward Central Schemes, which are unconditionally stable, we should see convergence for all values of h_x and h_t, with $O(h_t^2, h_x^2)$ for Crank-Nicholson and $O(h_t, h_x^2)$ for Backward Central. However, in some cases, you may not see these rates. For instance if your error due to the temporal terms is larger than the spatial ones, reducing h_x any further will not gain you anything. If we use non-smooth data, for instance an initial condition with a jump discontinuity, you should see some interesting results. Based on what we learn later about dissipation and dispersion, the stable schemes may try to "smooth" out the initial condition as it propagates the wave, or it may create numerical dispersion...