NUMERICAL SOLUTIONS TO PARTIAL DIFFERENTIAL EQUATIONS 1, HW 1

NICK LEWIS

Problem 1 Consider the problem

$$u_t(x,t) = u_x(x,t) \tag{0.1a}$$

$$u(x,0) = f(x) \tag{0.1b}$$

Consider the method

$$v_j^{n+1} = v_j^n + \frac{k}{h}(v_\ell^n - v_{\ell-1}^n)$$
$$v_j^0 = f_j$$

In order for $|\hat{Q}| \leq 1$, should $\ell = j+1$ or $\ell = j$? Show the analysis. If $\ell = j+1$ this will be the upwind method. If If $\ell = j$ this will be the downwind method.

Solution To determine if $\ell = j+1$ or $\ell = j$, we will first derive the forms of $|\hat{Q}| \leq 1$ for each ℓ. For simplicity, let $\lambda = \frac{k}{h}$ and this notation will be used throughout my solution.

Recall that the forward, central and backwards difference operators are defined as:

$$D_+ v_j^n = \frac{v_{j+1}^n - v_j^n}{h}$$
$$D_0 v_j^n = \frac{v_{j+1}^n - v_{j-1}^n}{2h}$$
$$D_- v_j^n = \frac{v_j^n - v_{j-1}^n}{h}$$

Using this formulation we can re-write the equations below by simply plugging in the values for ℓ and simplifying the expressions.

$$v_j^{n+1} = (I + kD_+)v_j^n, \ell = j+1$$
$$v_j^{n+1} = (I + kD_-)v_j^n, \ell = j$$

Now we have a form for Q for both the upwind and downwind methods. Using tools from fourier analysis we can re-write $v_j^n = \frac{1}{\sqrt{2\pi}} e^{i\omega x_j} \tilde{v}^n(\omega)$. This then yields, for the upward method,

$$\tilde{v}^{n+1}(\omega)e^{i\omega x_j} = (e^{i\omega x_j} + \lambda(e^{i\omega x_{j+1}} - e^{i\omega x_j}))\tilde{v}^n(\omega)$$
$$\tilde{v}^{n+1}(\omega) = (1 + \lambda(e^{i\omega h} - 1))\tilde{v}^n(\omega)$$

Likewise for the downwind method,

$$\tilde{v}^{n+1}(\omega)e^{i\omega x_j} = (e^{i\omega x_j} + \lambda(e^{i\omega x_j} - e^{i\omega x_{j-1}}))\tilde{v}^n(\omega)$$
$$\tilde{v}^{n+1}(\omega) = (1 + \lambda(1 - e^{-i\omega h}))\tilde{v}^n(\omega)$$

In other words we can write $\hat{Q}$ more generally as

Date: October 4, 2021.

$$\hat{Q}(\omega) = \begin{cases} 1-\lambda+\lambda e^{i\omega h} & \text{for } \ell = j+1 \\ 1+\lambda-\lambda e^{-i\omega h} & \text{for } \ell = j \end{cases}$$

Now we want to identify for which method is $|\hat{Q}| \leq 1$. To do so we take the modulus squared which becomes

$$|\hat{Q}(\omega)|^2 = \begin{cases} (1-\lambda+\lambda e^{i\omega h})(1-\lambda+\lambda e^{-i\omega h}) & \text{for } \ell = j+1 \\ (1+\lambda-\lambda e^{-i\omega h})(1+\lambda-\lambda e^{i\omega h}) & \text{for } \ell = j \end{cases}$$

$$= \begin{cases} (1-2\lambda(1-\lambda)(1-cos(\omega h)) & \text{for } \ell = j+1 \\ (1+2\lambda(1-\lambda)(1-cos(\omega h)) & \text{for } \ell = j \end{cases}$$

To simplify the analysis recall the trigonmetric identity $1-cos(\omega x) = 2sin^2(\frac{\omega x}{2})$. This implies $0 \leq 1-cos(\omega x)$ which in turn implies the downwind method will not be stable for $\lambda \in [0,1]$ as the function $\lambda(1-\lambda)$ is non-negative only in the interval $[0,1]$. What about for other values of λ though? Well if λ rest outside the unit interval, this will still imply instability as λ cannot be negative, and if $\lambda > 1$, then this implies further instability.

For the upwind method, this stability is only obtained for $\lambda \in [0,1]$ since the function $\lambda(1-\lambda)$ is non-negative only in the interval $[0,1]$. This implies the upwind scheme is conditionally stable, and that the downwind scheme is unconditionally unstable.

Problem 2 Investigate the truncation error the upwind method. That is, if u solves (0.1) and is smooth find τ_j^n such that

$$u_j^{n+1} = u_j^n + \frac{k}{h}(u_{j+1}^n - u_j^n) + k\tau_j^n$$

What are the leading terms of τ_j^n? Could this method be more accurcate than the Lax-Wendroff method, in your opinion?

Solution To investigate the error of the upwind method, we need to do a taylor series expansion of u at the point (x_j, t_n). For practicality, let's denote $u_j^n = u(x_j, t_n)$ Assuming u is sufficiently smooth, and are time steps for space and time are h and k, respectively, this gives us

$$u(x_j+h, t_n) = u(x_j,t_n) + h\partial_x u(x_j,t_n) + \frac{h^2}{2}\partial_x^2 u(x_j,t_n) + \frac{h^3}{6}\partial_x^3 u(x_j,t_n)$$
$$u(x_j, t_n+k) = u(x_j,t_n) + k\partial_t u(x_j,t_n) + \frac{k^2}{2}\partial_t^2 u(x_j,t_n) + \frac{k^3}{6}\partial_t^3 u(x_j,t_n)$$

Using the forward difference method for the time steps,and likewise the forward difference for the space steps, we yield the following:

$$\frac{v_j^{n+1}-v_j^n}{k} - \frac{v_{j+1}^n - v_j^n}{h} = \partial_x u(x_j,t_n) + \frac{h}{2}\partial_x^2 u(x_j,t_n) + \frac{h^2}{6}\partial_x^3 u(x_j,t_n) - \partial_t u(x_j,t_n) + \frac{k}{2}\partial_t^2 u(x_j,t_n) + \frac{k^2}{6}\partial_t^3 u(x_j,$$

We can simplify this even further. Recall that we're dealing with the transport equation, so the first time partial and first space partial are equal, meaning they cancel out. This also implies that $\partial_t^2 u(x_j,t_n) = \partial_x^2 u(x_j,t_n)$. This will yield

$$\frac{v_j^{n+1}-v_j^n}{k} - \frac{v_{j+1}^n - v_j^n}{h} = \frac{h}{2}\partial_x^2 u(x_j,t_n) - \frac{k}{2}\partial_t^2 u(x_j,t_n) - \frac{h^2}{6}\partial_x^3 u(x_j,t_n) + \frac{k^2}{6}\partial_t^3 u(x_j,t_n) + h.o.t.$$

Given this we say that the truncation error is $\tau_j^n = O(k+h)$. No, I would say it is not possible unless we assume the condition that $h = k$ and that u is twice differentiable, then τ_j^n for the upwind method becomes $\tau_j^n = O(k^2 + h^2)$ since we can equate the second partials, and this will make the upwind method second-order accurate while still preserving its stability.

Problem 3 Consider a general explicit one step finite difference method for the transport problem of the form

$$v_j^{n+1} = Qv_j^n,$$
$$v_j^0 = f_j.$$

where $Q = \sum_{\nu=-r}^{s} A_\nu(k,h)E^\nu$. Let N be even then we can write:

$$f_j = \frac{1}{\sqrt{2\pi}} \sum_{\omega=-N/2}^{N/2} e^{i\omega x_j} \tilde{f}(\omega). \tag{0.2}$$

Show that

$$v_j^n = \frac{1}{\sqrt{2\pi}} \sum_{\omega=-N/2}^{N/2} [\hat{Q}(\omega)]^n e^{i\omega x_j} \tilde{f}(\omega).$$

What is $\hat{Q}(\omega)$?

Solution

Notice that we may write the wave representation of v_j^n as follows: $v_j^n = \frac{1}{\sqrt{2\pi}} e^{i\omega x_j} \tilde{v}^n(\omega)$. Applying Q to this term will yield us

$$v_j^{n+1} = Q\frac{1}{\sqrt{2\pi}} e^{i\omega x_j} \tilde{v}^n(\omega)$$
$$= \sum_{\nu=-r}^{s} A_\nu(k,h)E^\nu \frac{1}{\sqrt{2\pi}} e^{i\omega x_j} \tilde{v}^n(\omega)$$
$$= \sum_{\nu=-r}^{s} A_\nu(k,h)e^{i\omega\nu} \frac{1}{\sqrt{2\pi}} e^{i\omega x_j} \tilde{v}^n(\omega)$$

When using the wave representation of v_j^{n+1}, this then gives us

$$\tilde{v}^{n+1}(\omega) = \sum_{\nu=-r}^{s} A_\nu(k,h)e^{i\omega\nu} \tilde{v}^n(\omega)$$

Implying that $\hat{Q} = \sum_{\nu=-r}^{s} A_\nu(k,h)e^{i\omega\nu}$. Via induction we can then find that $\tilde{v}^n(\omega) = \hat{Q}^n\tilde{v}^0(\omega)$ so now instead of using one wave, let's apply the superposition principle to obtain

$$v_j^n = \frac{1}{\sqrt{2\pi}} \sum_{\omega=-N/2}^{N/2} e^{i\omega x_j} \tilde{v}^n(\omega).$$
$$= \frac{1}{\sqrt{2\pi}} \sum_{\omega=-N/2}^{N/2} [\hat{Q}(\omega)]^n e^{i\omega x_j} \tilde{f}(\omega).$$

Problem 4 Consider the θ scheme

$$(1-\theta kD_0)v_j^{n+1} = \Big(I + (1-\theta)kD_0\Big)v_j^n,$$
$$v_j^0 = f_j.$$

Note that when $\theta = 1$ we obtain the Backward Euler method and when $\theta = \frac{1}{2}$ you obtain the Crank-Nicholson method. Using the representation of f, (0.2), derive an explicit representation for v_j^n. In particular, what is $\hat{Q}$? Show that the method is unconditionally stable for $\frac{1}{2} \leq \theta \leq 1$.

Solution

By using the representation of f, let us assume that our solution for v_j^n occurs in the form

$$v_j^n = \frac{1}{\sqrt{2\pi}} \sum_{\omega=-N/2}^{N/2} e^{i\omega x_j} \tilde{v}^n(\omega). \tag{0.3}$$

To simplify the analysis, we will focus solely on one wave. Due to the additive nature of the PDE, we can sum for multiple waves. So to begin, assume $v_j^n = \frac{1}{\sqrt{2\pi}} e^{i\omega x_j} \tilde{v}^n(\omega)$. Then for the Numerical Method described in Problem 4, this becomes:

$$(1-\theta kD_0)e^{i\omega x_j}\tilde{v}^{n+1}(\omega) = \Big(I + (1-\theta)kD_0\Big)e^{i\omega x_j}\tilde{v}^n(\omega),$$
$$(e^{i\omega x_j} - \frac{\theta\lambda}{2}(e^{i\omega x_{j+1}} - e^{i\omega x_{j-1}}))\tilde{v}^{n+1}(\omega) = (e^{i\omega x_j} + \frac{(1-\theta)\lambda}{2}(e^{i\omega x_{j+1}} - e^{i\omega x_{j-1}}))\tilde{v}^n(\omega)$$
$$(1 - \frac{\theta\lambda}{2}(e^{i\omega h} - e^{-i\omega h}))\tilde{v}^{n+1}(\omega) = (1 + \frac{(1-\theta)\lambda}{2}(e^{i\omega h} - e^{-i\omega h}))\tilde{v}^n(\omega)$$
$$(1 - \theta\lambda isin(\omega h))\tilde{v}^{n+1}(\omega) = (1 + (1-\theta)\lambda isin(\omega h))\tilde{v}^n(\omega)$$
$$\tilde{v}^{n+1}(\omega) = \frac{(1 + (1-\theta)\lambda isin(\omega h))}{(1 - \theta\lambda isin(\omega h))}\tilde{v}^n(\omega)$$

So the end result is that

$$\hat{Q} = \frac{(1 + (1-\theta)\lambda isin(\omega h))}{(1 - \theta\lambda isin(\omega h))} \tag{0.4}$$

The logic behind going from line 3 to line 4 rest in Euler's Identity,

$$e^{i\omega x} = cos(\omega x) + isin(\omega x) \tag{0.5}$$

To get an explicit representation for the equation, recall from Problem 3 that v_j^n can be written as $\frac{1}{\sqrt{2\pi}} \sum_{\omega=-N/2}^{N/2} e^{i\omega x_j}\tilde{v}^n(\omega)$, and that $\tilde{v}^n(\omega) = \hat{Q}^n\tilde{v}^0(\omega)$. This will yield

$$v_j^n = \frac{1}{\sqrt{2\pi}} \sum_{\omega=-N/2}^{N/2} [\frac{(1 + (1-\theta)\lambda isin(\omega h))}{(1 - \theta\lambda isin(\omega h))}]^n e^{i\omega x_j}\tilde{f}(\omega).$$

Now that we have a form for $\hat{Q}$, we will now demonstrate that it is unconditionally stable for $\frac{1}{2} \leq \theta \leq 1$.

Firstly, let's simplify $\hat{Q}$ to get rid of the complex number on the bottom. This gives us

$$\hat{Q} = \frac{(1 + (1-\theta)\lambda isin(\omega h))(1 + \theta\lambda isin(\omega h))}{(1 + \theta^2\lambda^2 sin^2(\omega h))}$$

Next we take the modulus of $\hat{Q}$ which in turn gives

$$|\hat{Q}|^2 = \frac{(1+(1-\theta)^2\theta^2\lambda^2 sin^2(\omega h))(1+\theta^2\lambda^2 sin^2(\omega h))}{(1+\theta^2\lambda^2 sin^2(\omega h))^2}$$
$$= \frac{(1+(1-\theta)^2\lambda^2 sin^2(\omega h))}{(1+\theta^2\lambda^2 sin^2(\omega h))}$$

Note that this goes to 1 if $\theta = \frac{1}{2}$, and for $\frac{1}{2} \leq \theta$, $(1-\theta) \leq \theta$ implying that the denominator will be larger than the numerator, so we end up getting $|\hat{Q}| \leq 1$.

Problem 5 Derive the linear system arising from the Crank-Nicholson method.

Solution

The Crank Nicholson Method is just the numerical scheme from Problem 4 when $\theta = \frac{1}{2}$ which yields

$$(1 - \frac{1}{2}kD_0)v_j^{n+1} = \Big(I + \frac{1}{2}kD_0\Big)v_j^n,$$
$$v_j^0 = f_j.$$

By expanding the central difference method terms this can be written as

$$v_j^{n+1} - \frac{1}{2}\frac{k}{2h}(v_{j+1}^{n+1} - v_{j-1}^{n+1}) = v_j^n + \frac{1}{2}\frac{k}{2h}(v_{j+1}^n - v_{j-1}^n),$$
$$v_j^0 = f_j.$$

This looks to be intimidating at first, but let's examine the properties of system for a couple of $j \in \{1,2,3\}$ To find a pattern that might prove helpful.

$$v_1^{n+1} - \frac{\lambda}{4}(v_2^{n+1} - v_0^{n+1}) = v_1^n + \frac{\lambda}{4}(v_2^n - v_0^n),$$
$$v_2^{n+1} - \frac{\lambda}{4}(v_3^{n+1} - v_1^{n+1}) = v_2^n + \frac{\lambda}{4}(v_2^n - v_1^n),$$
$$v_3^{n+1} - \frac{\lambda}{4}(v_4^{n+1} - v_2^{n+1}) = v_3^n + \frac{\lambda}{4}(v_4^n - v_2^n)$$

With this we begin to recognize a pattern in the schema of this system of linear equations. For one, we get a matrix on both sides multiplied by some vector v^{n+1}, or v^n. To write it out explicitly,

$$Av^{n+1} = Bv^n,$$

To be more explicit, $v^{n+1} = (v_0^{n+1}, v_1^{n+1}, ..., v_N^{n+1})$, likewise for v^n. We can then explicitly write out the formulae for both A and B which are as follows:

$$A = \begin{cases} \frac{\lambda}{4} & \text{for } (i,j) = (i,i-1) \text{ and } i > 1; (i,j) = (1,N) \\ 1 & \text{for } (i,j) = (i,i) \\ -\frac{\lambda}{4} & \text{for } (i,j) = (i,i+1), (i,j) = (N,1) \end{cases}$$

$$B = \begin{cases} -\frac{\lambda}{4} & \text{for } (i,j) = (i,i-1) \text{ and } i > 1; (i,j) = (1,N) \\ 1 & \text{for } (i,j) = (i,i) \\ \frac{\lambda}{4} & \text{for } (i,j) = (i,i+1), (i,j) = (N,1) \end{cases}$$

The condition in the first row is a result of periodicity of the boundary terms, i.e. $v_0^{n+1} = v_N^{n+1}$ and $v_0^n = v_N^n$.

Problem 6 Code the following methods: Upwind Scheme, the naive method first introduced in class (which we can call the centered scheme), Lax-Wendroff scheme, Lax-Friedrichs scheme, Backward Euler scheme, and the Crank-Nicholson method.. Keeping $\lambda = k/h = 1/2$, apply these methods to the first order wave equation for $h = 1/2^j$ for $j = 5, 6, 7$. Let your initial condition be:

$$f(x) = \begin{cases} x & \text{for } 0 \leq x \leq \pi \\ 2\pi - x & \text{for } \pi < x < 2\pi \end{cases}$$

Plot your solutions at time $t_n = 1$. Also, give the error of each method measured in the $\|\cdot\|_h$ norm at time $t_n = 1$ when $h = 1/2^7$. Which methods do better?

A practical note: notice that $N + 1 = \frac{2\pi}{h}$ but with the choices I gave above N will not be an integer. Instead you should use the nearest integer of $\frac{2\pi}{h}$ to define N. Keep everything else as is.

Solution

To begin this solution let us define our parameters. Here, h is our step size for the space variable x; k is the step size for the time variable t. We begin at an initial point t_0 and x_0 and end at x_N and t_K, respectively, so there are $N + 1$ gridpoints for x, and $K + 1$ gridpoints for t. We assume $\lambda = \frac{1}{2}$, and this will imply that $k = \frac{h}{2}$ so $k \in \{\frac{1}{2^6}, \frac{1}{2^7}, \frac{1}{2^8}\}$ because $\lambda = \frac{k}{h}$. Now if we restrict $x \in [0, 2\pi]$ and $t \in [0, 1]$ this will give us that $N \in \{200, 401, 802\}$ for $N + 1 = \frac{2\pi}{h}$, and $K \in \{63, 127, 255\}$ for $K + 1 = \frac{1}{k}$. As a side note, we could easily make the domain of x something else; it is 2π periodic so the function $f(x)$ will just repeat over an interval of length 2π. Also we can change the right boundary for t; we simply set $t_K = 1$ for our own convenience.

With each of these methods you can write the numerical PDE as a linear system of equations, and fortunately the accompanying matrices for the implicit methods are invertible. I will state the matrix used to solve the system along with four graphs: One with the solution for $h = 2^{-5}$, one for $h = 2^{-6}$, one for $h = 2^{-7}$ and one with the solutions being overlapped on each other. Our initial condition is

$$f(x) = \begin{cases} x & \text{for } 0 \leq x \leq \pi \\ 2\pi - x & \text{for } \pi < x < 2\pi \end{cases}$$

which graphically can be represented as

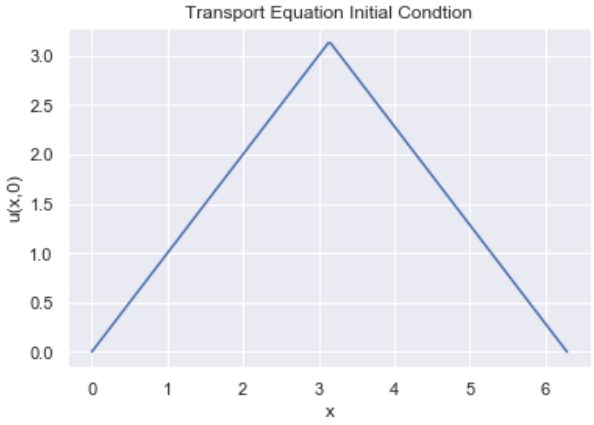
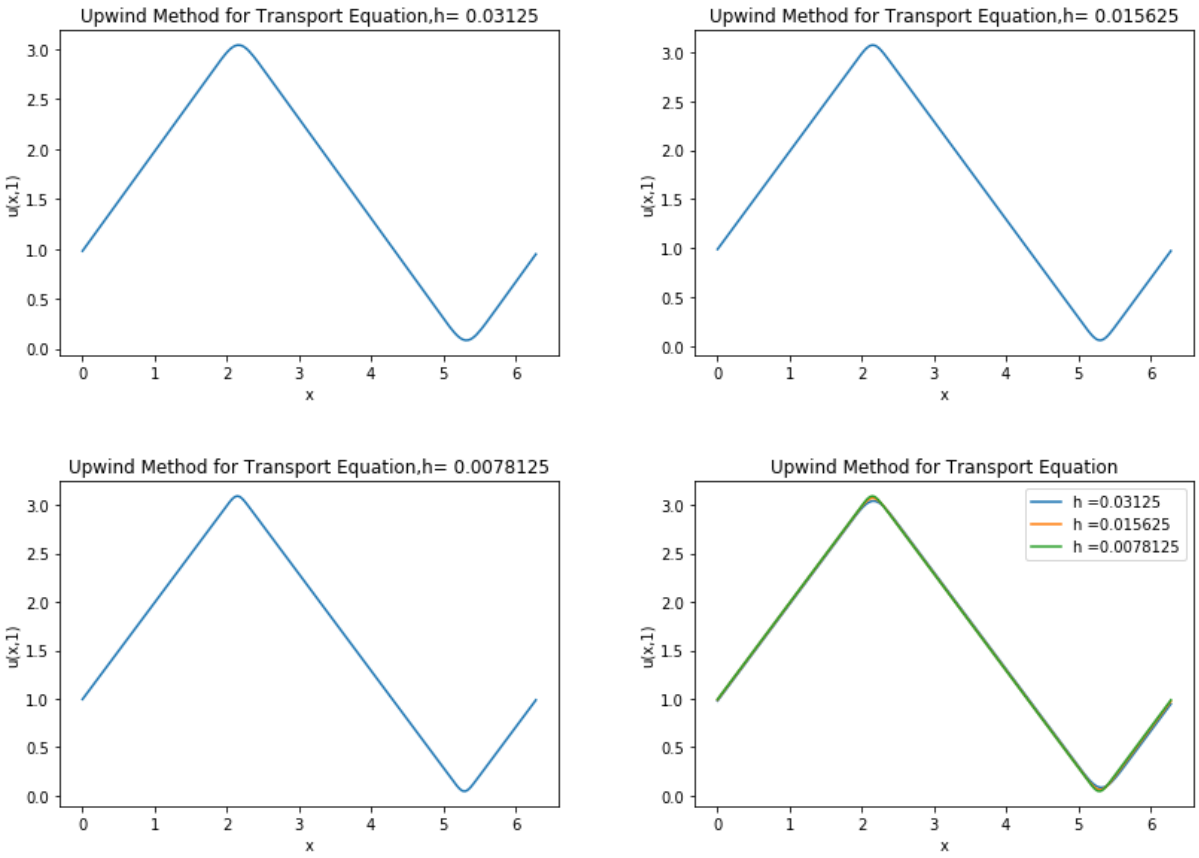


Figure 1. Initial condition for the transport equation given in Problem 6

0.1. **Upwind Method.** The matrix corresponding to the upwind method is

$$A_{Upwind} = \begin{cases} 1 - \lambda & \text{for } (i, j) = (i, i) \\ \lambda & \text{for } (i, j) = (i, i + 1) \\ \lambda & \text{for } (i, j) = (N, 1) \end{cases}$$

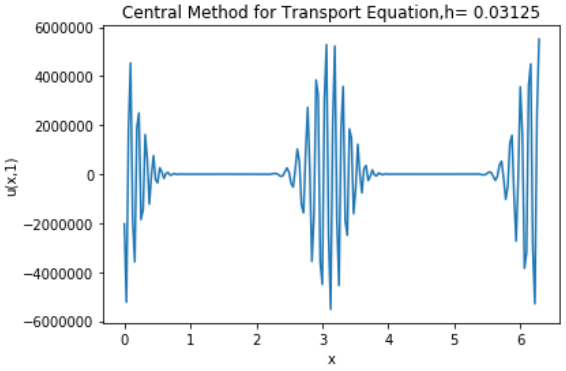
Below are the graphs for the space steps.

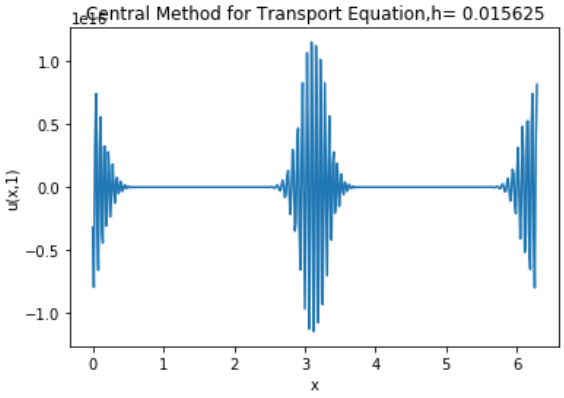


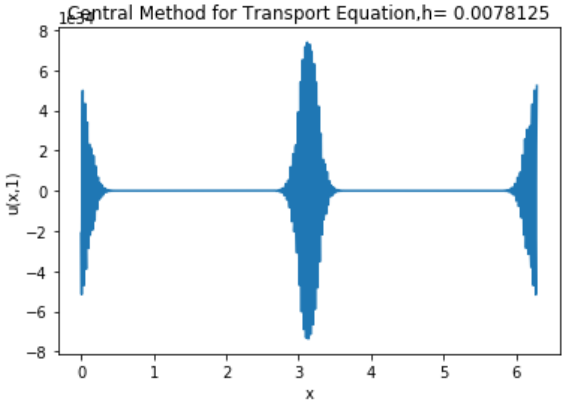
0.2. **Central Difference Method.** The matrix corresponding to the Central Difference method is

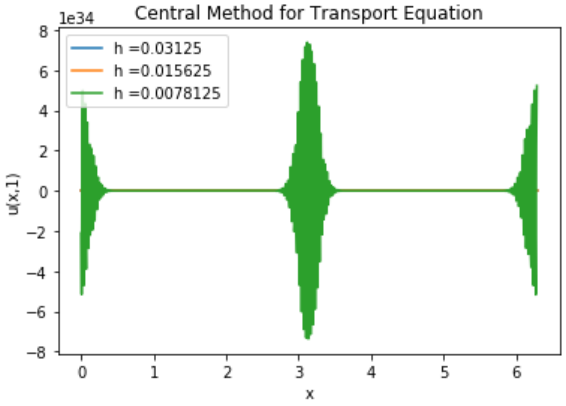
$$A_{CM} = \begin{cases} -\lambda & \text{for } (i,j) = (i, i-1) \text{ and } i > 1; \\ 1 & \text{for } (i,j) = (i,i) \\ \lambda & \text{for } (i,j) = (i, i+1) \\ -\lambda & \text{for } (i,j) = (1, N) \\ \lambda & \text{for } (i,j) = (N, 1) \end{cases}$$

Below are the graphs for the space steps.





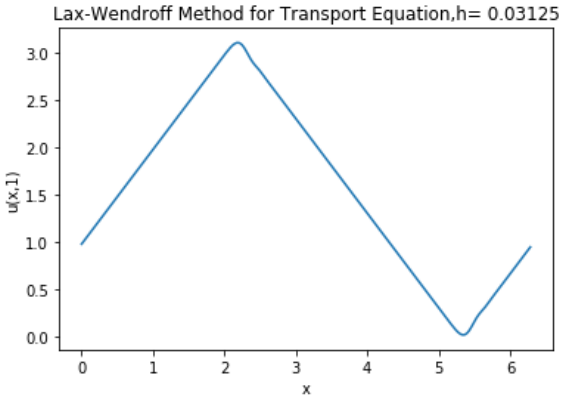


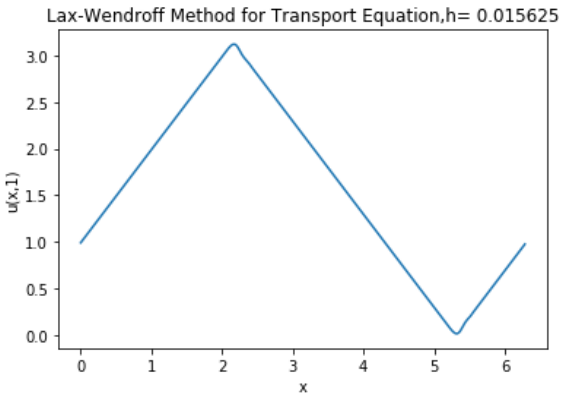


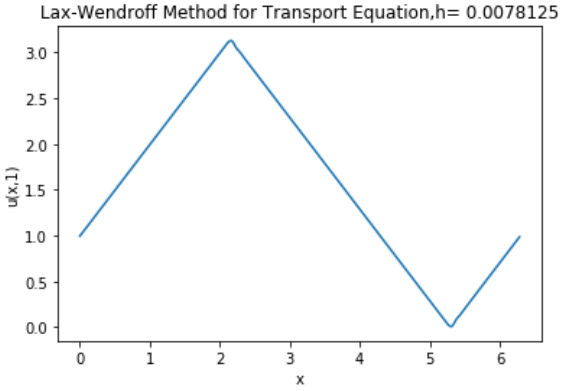
0.3. **Lax-Wendroff.** For Lax-Wendroff, $\sigma = \frac{1}{2\lambda}$. The matrix corresponding to the Lax-Wendroff method is

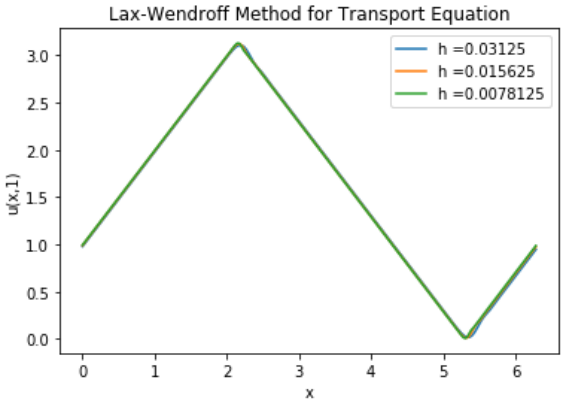
$$A_{LW} = \begin{cases} \sigma\lambda - \frac{\lambda}{2} & \text{for } (i,j) = (i, i-1) \text{ and } i > 1; \\ 1 - 2\sigma\lambda & \text{for } (i,j) = (i,i) \\ \sigma\lambda + \frac{\lambda}{2} & \text{for } (i,j) = (i, i+1) \\ \sigma\lambda - \frac{\lambda}{2} & \text{for } (i,j) = (1, N) \\ \sigma\lambda + \frac{\lambda}{2} & \text{for } (i,j) = (N, 1) \end{cases}$$

Below are the graphs for the space steps.





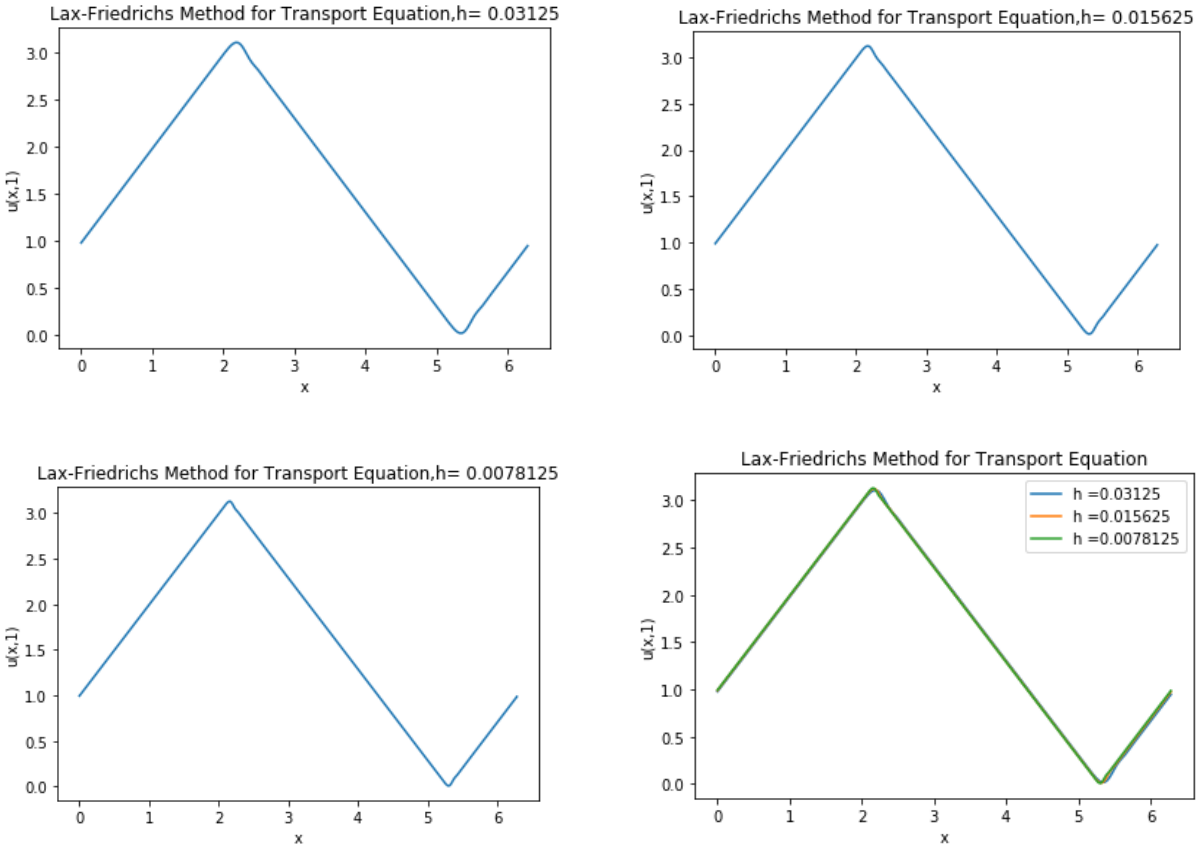




0.4. **Lax-Friedrichs.** For Lax-Friedrichs, $\sigma = \frac{\lambda}{2}$. The matrix corresponding to the Lax-Friedrichs method is

$$A_{LF} = \begin{cases} \sigma\lambda - \frac{\lambda}{2} & \text{for } (i,j) = (i, i-1) \text{ and } i > 1; \\ 1 - 2\sigma\lambda & \text{for } (i,j) = (i,i) \\ \sigma\lambda + \frac{\lambda}{2} & \text{for } (i,j) = (i, i+1) \\ \sigma\lambda - \frac{\lambda}{2} & \text{for } (i,j) = (1, N) \\ \sigma\lambda + \frac{\lambda}{2} & \text{for } (i,j) = (N, 1) \end{cases}$$

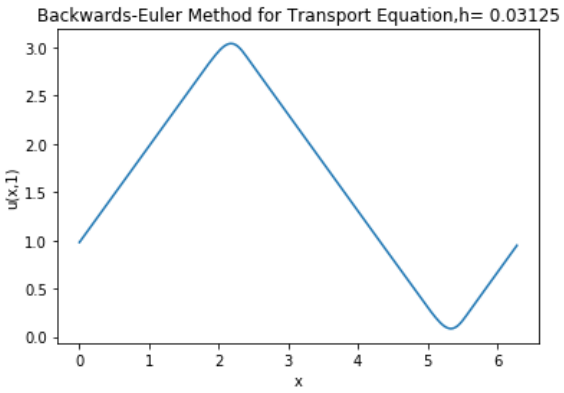
Below are the graphs for the space steps.

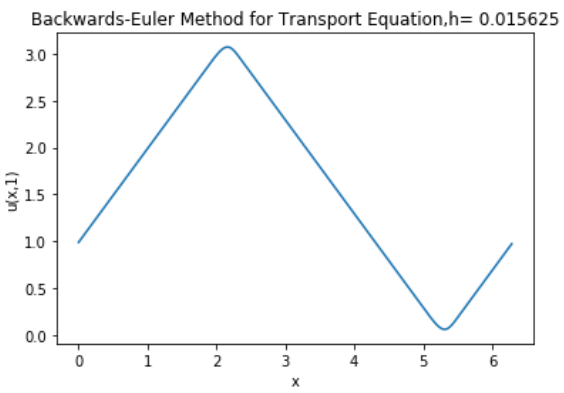


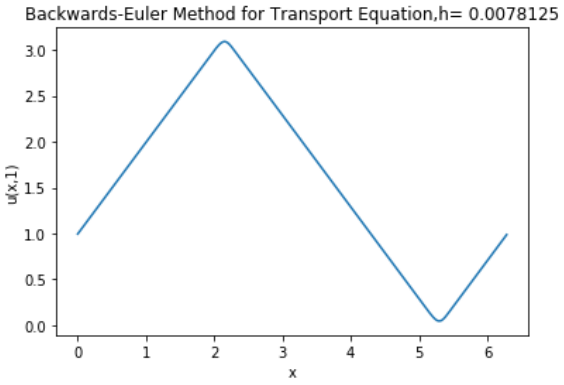
0.5. **Backwards Euler.** The matrix corresponding to the Backwars-Euler method is

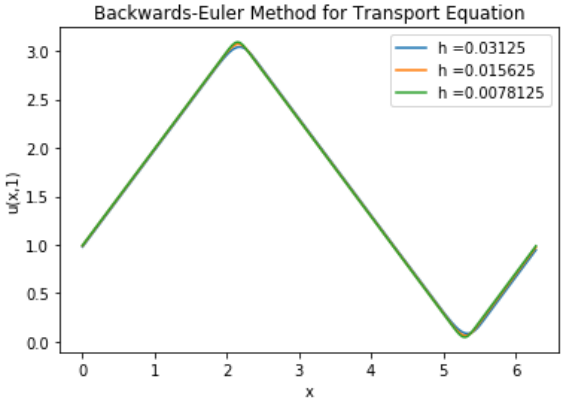
$$A_{BE} = \begin{cases} \frac{\lambda}{2} & \text{for } (i,j) = (i, i-1) \text{ and } i > 1; \\ 1 & \text{for } (i,j) = (i,i) \\ -\frac{\lambda}{2} & \text{for } (i,j) = (i, i+1) \\ -\frac{\lambda}{2} & \text{for } (i,j) = (N,1) \\ \frac{\lambda}{2} & \text{for } (i,j) = (1,N) \end{cases}$$

Below are the graphs for the space steps.

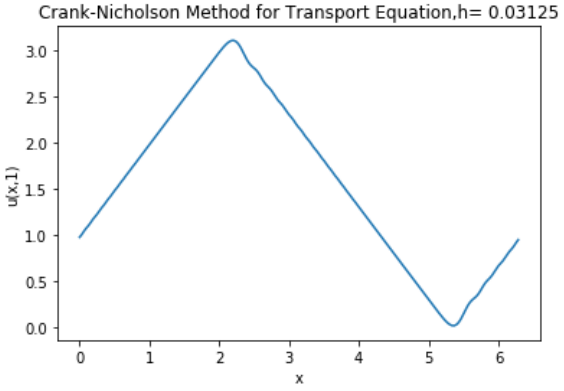


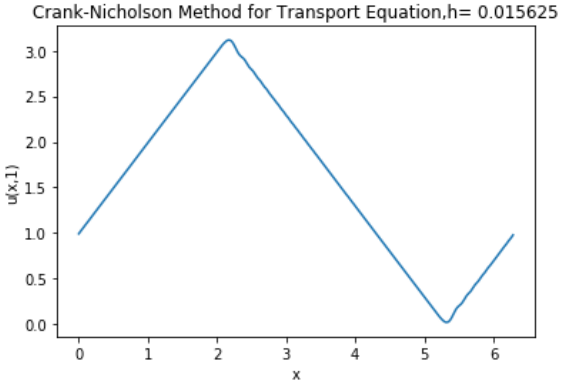


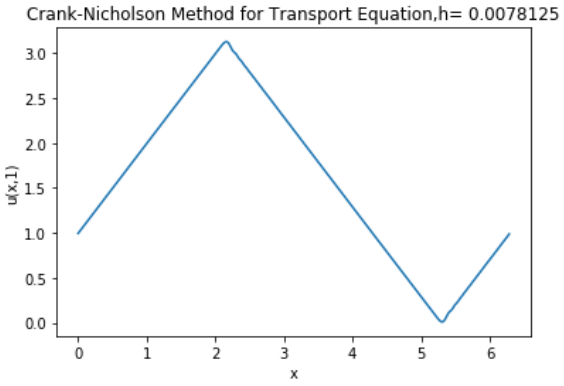


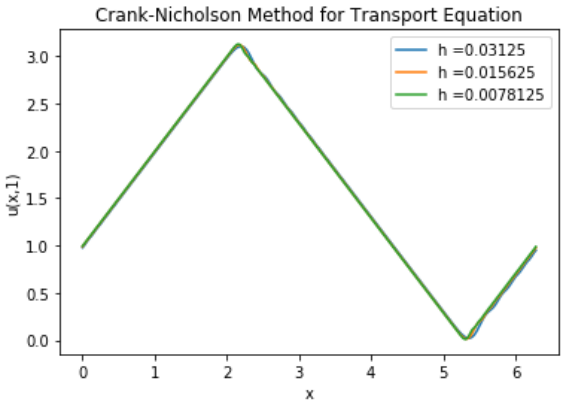


0.6. **Crank-Nicholson.** For Crank Nicholson the schema are written above in problem 5. Below are the graphs for the space steps.









0.7. **Error Terms for Different Methods.** Now that we have plotted our solutions using the various numerical methods, we would like to see how "well" they approximate the true solution. To do so, recall that the solution for the 1D transport equation is $u(x,t) = f(x+t)$. This yields the true solution to be

$$u(x,t) = \begin{cases} x+t & \text{for } 0 \leq x+t \leq \pi \\ 2\pi - (x+t) & \text{for } \pi < x+t < 2\pi \end{cases}$$

Graphically, for $t = 1$, this looks like

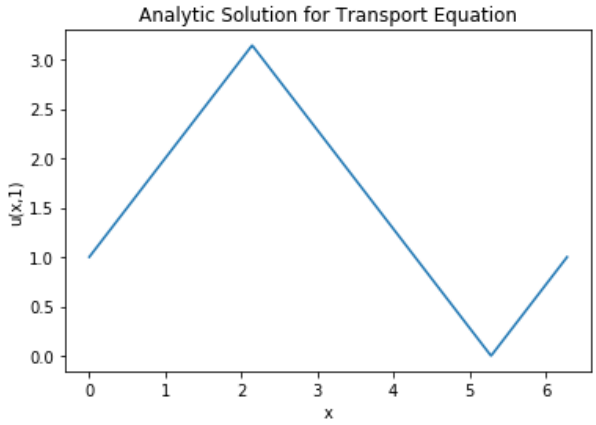


Figure 8. The graph of the analytic solution of u(x,t) at t = 1

Before moving on to the h-norm errors, we should probably say a couple words about the solution to this equation. Firstly, it makes sense intuitively. The solution to the transport equation simply translates the wave with respect to time so the solution should resemble the initial condition as it moves to the right.

We're lucky in this case to be able to produce an analytic solution to the Partial Differential Equation. It then allows us to look at the error between the solution and the numerical method through use of the h-norm, $\|u^n\|_h^2 = \sum_{j=0}^{N} \|u_j^n\|^2$. For the purpose of this problem, we will take the norm for the value of $t_n = 1$ and $h = \frac{1}{2^7}$. The results for each of the methods is shown below in the table:

Table 1. Errors in h-norm ($\|\cdot\|_h$) for Different Numerical Methods

	Numerical Methods for Transport Equation					
Time	Upwind Scheme	Central Difference	Lax Friedrichs	Lax Wendroff	Backwards Euler	Crank Nicholson
$t_n = 1$	0.245	3.680×10^{35}	0.204	0.204	0.248	0.210

In terms of the error measured by the h-norm, it would appear as though the Lax methods do the best in approximating the solution to the PDE, followed by the Crank-Nicholson method which is then followed by the Upwind method, Backwards Euler and Central Difference method, which does abysmally. Notice for the Lax methods, for the space step h, they do a good job of re-creating the spikes of crest and trough of the wave. Crank-Nicholson also does a good job of this, but soon after it produces noticeable perturbations after the sharp peaks. The Lax methods also produce these perturbations, but seem to be able to better handle them.As for the Upwind and Backward's Euler methods, they do well in approximating the shape of the actual solution, but fail to produce the sharp peaks of the actual solution. That is to say both of these methods smooth out the maximum and minimum of the solution instead of approximating the sharp edges seen. Not much needs to be said about the Central Difference method save for its rapid instability.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  1 00:02:07 2021

@author: nlewis
"""

#useful math functions and everything else
import numpy as np

#useful for setting up matrices
from scipy.sparse import spdiags
#for plotting our histograms and contours
import matplotlib.pyplot as plt

#for making cool animations


#Problem 6
# The purpose of this problem is to code up solutions for the transport equation
#different numerical methods. As such

### Step sizes for space (h) and time (k)
#we define lambda to be 1/2, but lambda = k/h so k = h/2.
lambda_ = 1/2
h = 0.5**np.array([5,6,7])
k_ = h/2

### THIS CREATES THE SEQEUNCE OF STEPS FOR EACH ANALYSIS. Recall x_j = hj, 2pi/h
###NOTE: since N will not be an integer, we can just take int of this expression
#will act as a floor function
### We use a list to save on memory.

#Boundaries for  space variable
x_0 = 0
x_N = 2*np.pi

#Boundaries for time variable
t_0 = 0
t_K = 1

##space variable gridpoints for j = 5, 6, 7
###time variable grid points for j = 5, 6, 7

x,t = [],[]
for j in range(len(h)):
    x.append( np.linspace(start = x_0, stop = x_N, num = int( ((x_N - x_0)/h[j])
    t.append( np.linspace(start = t_0, stop = t_K, num = int( ((t_K - t_0)/k_[j]
```

```
#How many gridpoints are in the space and time variables
N = np.array([len(x[0]), len(x[1]), len(x[2])])
K = np.array([len(t[0]), len(t[1]), len(t[2])])

## Define our initial condition here
def f(x):
    #let's us define a piecewise function
    tau = 2*np.pi
    #makes sure this is periodic for values not in the interval [0,2pi]
    x = x % tau
    y= np.piecewise(
        x,
        [(0 <= x)&(x <= np.pi),  (np.pi < x)&(x < 2*np.pi)      ],
        [lambda x: x, lambda x: 2*np.pi - x,0])
    return y

### Graph of f(x)
plt.plot(x[0],f(x[0]))

q = np.linspace(0,4*np.pi,1000)
plt.plot(q,f(q))
#self explanatory
plt.xlabel('x')
plt.ylabel('u(x,0)')
plt.title('Transport Equation Initial Condtion')



### UPWIND SCHEME
#v^(n+1)_j = v^n_j + lambda*(v^n_(j+1) - v^n_j)
#v^0_j = f_j
# Recall this can be written in matrix form
o1 = np.ones(N[0])
o2 = np.ones(N[1])
o3 = np.ones(N[2])
A = [spdiags( [lambda_*o1,(1-lambda_)*o1,lambda_*o1], (1-N[0],0,1), N[0], N[0] )
     spdiags( [lambda_*o2,(1-lambda_)*o2,lambda_*o2], (1-N[1],0,1), N[1], N[1] )
     , spdiags( [lambda_*o3,(1-lambda_)*o3,lambda_*o3], (1-N[2],0,1), N[2], N[2]

### Makes it so we can do this for any given j = 5,6 or 7. Simplifies analysis
solutions0 = []
for j in range(len(N)):
    solutions0.append( np.zeros((N[j],K[j])) )

for k in range(3):
    #initializes conditions to start with
    solutions0[k][:,0] = f(x[k])
    for j in range(1,K[k]):
        #using matrix corresponding to steps, we sample. We then set initial equ
        #so we can repeat this process (i.e. we use previous time to update futu
```

```
        solutions0[k][:,j] = A[k].dot(solutions0[k][:,j-1])

f1 = solutions0[0]
f2 = solutions0[1]
f3 = solutions0[2]

plt.plot(x[0], f1[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Upwind Method for Transport Equation' +',' + 'h= ' + str(1/2**5))

plt.plot(x[1], f2[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Upwind Method for Transport Equation' +',' + 'h= ' + str(1/2**6))

plt.plot(x[2], f3[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Upwind Method for Transport Equation' +',' + 'h= ' + str(1/2**7))

plt.plot(x[0], f1[:,-1])
plt.plot(x[1], f2[:,-1])
plt.plot(x[2], f3[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Upwind Method for Transport Equation')
plt.legend(['h =' + str(1/2**5), 'h =' + str(1/2**6),'h =' + str(1/2**7)])

### NAIVE METHOD (USING CENTRAL DIFFERENCE)
#v^(n+1)_j = v^n_j + lambda*(v^n_(j+1) - v^n_(j-1))
#v^0_j = f_j

A_CD = [spdiags( [lambda_*o1,-lambda_*o1, o1, lambda_*o1,-lambda_*o1], (1-N[0],-
     spdiags( [lambda_*o2,-lambda_*o2, o2, lambda_*o2,-lambda_*o2], (1-N[1],-1,0
     , spdiags([lambda_*o3,-lambda_*o3, o3, lambda_*o3,-lambda_*o3], (1-N[2],-1,

solutions1 = []
for j in range(len(N)):
    solutions1.append( np.zeros((N[j],K[j])) )

for k in range(3):
    #initializes conditions to start with
    solutions1[k][:,0] = f(x[k])
    for j in range(1,K[k]):
        #using matrix corresponding to steps, we sample. We then set initial equ
        #so we can repeat this process
        solutions1[k][:,j] = A_CD[k].dot(solutions1[k][:,j-1])

f1_CD = solutions1[0]
```

```
f2_CD = solutions1[1]
f3_CD = solutions1[2]

plt.plot(x[0], f1_CD[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Central Method for Transport Equation' +',' + 'h= ' + str(1/2**5))

plt.plot(x[1], f2_CD[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Central Method for Transport Equation' +',' + 'h= ' + str(1/2**6))

plt.plot(x[2], f3_CD[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Central Method for Transport Equation' +',' + 'h= ' + str(1/2**7))

plt.plot(x[0], f1_CD[:,-1])
plt.plot(x[1], f2_CD[:,-1])
plt.plot(x[2], f3_CD[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Central Method for Transport Equation')
plt.legend(['h =' + str(1/2**5), 'h =' + str(1/2**6),'h =' + str(1/2**7)])

### LAX-FRIEDRICHS
# choose sigma  such that 2*sigma = lambda. In this setting, sigma = 1/4
#v^(n+1)_j = (I+kD_0)v^n_j + (1/4)*kh*(D_+D_-)v^n_j
#v^0_j = f_j

#for convenience. we can manipulate matrices without having to re-enter terms an
sigma1 = lambda_/2
c = np.array([(sigma1*lambda_- lambda_/2),(1-2*sigma1*lambda_), (sigma1*lambda_+

A_LF = [spdiags( [c[2]*o1,c[0]*o1, c[1]*o1, c[2]*o1,c[0]*o1], (1-N[0],-1,0,1,N[0]
     spdiags( [c[2]*o2,c[0]*o2, c[1]*o2, c[2]*o2,c[0]*o2], (1-N[1],-1,0,1,N[1]-1
     , spdiags([c[2]*o3,c[0]*o3, c[1]*o3, c[2]*o3,c[0]*o3], (1-N[2],-1,0,1,N[2]-

solutions2 = []
for j in range(len(N)):
    solutions2.append( np.zeros((N[j],K[j])) )

for k in range(3):
    #initializes conditions to start with
    solutions2[k][:,0] = f(x[k])
    for j in range(1,K[k]):
        #using matrix corresponding to steps, we sample. We then set initial equ
        #so we can repeat this process
        solutions2[k][:,j] = A_LF[k].dot(solutions2[k][:,j-1])
```

```
f1_LF = solutions2[0]
f2_LF = solutions2[1]
f3_LF = solutions2[2]

plt.plot(x[0], f1_LF[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Lax-Friedrichs Method for Transport Equation' +',' + 'h=' + str(1/2*

plt.plot(x[1], f2_LF[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Lax-Friedrichs Method for Transport Equation' +',' + 'h=' + str(1/2*

plt.plot(x[2], f3_LF[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Lax-Friedrichs Method for Transport Equation' +',' + 'h=' + str(1/2*

plt.plot(x[0], f1_LF[:,-1])
plt.plot(x[1], f2_LF[:,-1])
plt.plot(x[2], f3_LF[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Lax-Friedrichs Method for Transport Equation')
plt.legend(['h =' + str(1/2**5), 'h =' + str(1/2**6),'h =' + str(1/2**7)])

### LAX-WINDROFF
#choose sigma such that 2*sigma = 1/lambda. In this setting, sigma = 1
#v^(n+1)_j = (I+kD_0)v^n_j + kh*(D_+D_-)v^n_j
#v^0_j = f_j

sigma2 = 1/(2*lambda_)
d = np.array([(sigma2*lambda_- lambda_/2),(1-2*sigma2*lambda_), (sigma2*lambda_+

A_LW = [spdiags( [d[2]*o1,d[0]*o1, d[1]*o1, d[2]*o1,d[0]*o1], (1-N[0],-1,0,1,N[0]
     spdiags( [d[2]*o2,d[0]*o2, d[1]*o2, d[2]*o2,d[0]*o2], (1-N[1],-1,0,1,N[1]-1
     , spdiags([d[2]*o3,d[0]*o3, d[1]*o3, d[2]*o3,d[0]*o3], (1-N[2],-1,0,1,N[2]-

solutions3 = []
for j in range(len(N)):
    solutions3.append( np.zeros((N[j],K[j])) )

for k in range(3):
    #initializes conditions to start with
    solutions3[k][:,0] = f(x[k])
    for j in range(1,K[k]):
        #using matrix corresponding to steps, we sample. We then set initial equ
        #so we can repeat this process
```

```
        solutions3[k][:,j] = A_LF[k].dot(solutions3[k][:,j-1])

f1_LW = solutions3[0]
f2_LW = solutions3[1]
f3_LW = solutions3[2]

plt.plot(x[0], f1_LW[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Lax-Wendroff Method for Transport Equation' +',' + 'h= ' + str(1/2**5

plt.plot(x[1], f2_LW[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Lax-Wendroff Method for Transport Equation' +',' + 'h= ' + str(1/2**6

plt.plot(x[2], f3_LW[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Lax-Wendroff Method for Transport Equation' +',' + 'h= ' + str(1/2**7

plt.plot(x[0], f1_LW[:,-1])
plt.plot(x[1], f2_LW[:,-1])
plt.plot(x[2], f3_LW[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Lax-Wendroff Method for Transport Equation')
plt.legend(['h =' + str(1/2**5), 'h =' + str(1/2**6),'h =' + str(1/2**7)])

### BACKWARD EULER
#(I - kD_0)v^(n+1)_j = v^n_j
#v^0_j = f_j
A_BE = [spdiags( [-0.5*lambda_*o1,0.5*lambda_*o1, o1, -0.5*lambda_*o1,0.5*lambda
     spdiags( [-0.5*lambda_*o2,0.5*lambda_*o2, o2, -0.5*lambda_*o2,0.5*lambda_*o
     , spdiags( [-0.5*lambda_*o3,0.5*lambda_*o3, o3, -0.5*lambda_*o3,0.5*lambda_

solutions4 = []
for j in range(len(N)):
    solutions4.append( np.zeros((N[j],K[j])) )


for k in range(3):
    #initializes conditions to start with
    solutions4[k][:,0] = f(x[k])
    C = np.linalg.inv(A_BE[k])
    for j in range(1,K[k]):
        #using matrix corresponding to steps, we sample. We then set initial equ
        #so we can repeat this process
        solutions4[k][:,j] = C.dot(solutions4[k][:,j-1])
```

```
f1_BE = solutions4[0]
f2_BE = solutions4[1]
f3_BE = solutions4[2]

plt.plot(x[0], f1_BE[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Backwards-Euler Method for Transport Equation' +',' + 'h= ' + str(1/2

plt.plot(x[1], f2_BE[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Backwards-Euler Method for Transport Equation' +',' + 'h= ' + str(1/2

plt.plot(x[2], f3_BE[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Backwards-Euler Method for Transport Equation' +',' + 'h= ' + str(1/2

plt.plot(x[0], f1_BE[:,-1])
plt.plot(x[1], f2_BE[:,-1])
plt.plot(x[2], f3_BE[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Backwards-Euler Method for Transport Equation')
plt.legend(['h =' + str(1/2**5), 'h =' + str(1/2**6),'h =' + str(1/2**7)])

### CRANK- NICHOLSON
# (I-0.5kD_0)v^(n+1)_j = (I+0.5kD_0)v^n_j
#v^0_j = f_j
theta = 1/2
#FOR CONVENIENCE (i.e. so I can still see what's written)
a = (1-theta)*lambda_/2
b = theta*lambda_/2

A_CN = [spdiags( [-a*o1,a*o1,o1,-a*o1,a*o1], (1-N[0],-1,0,1,N[0]-1), N[0], N[0])
        spdiags( [-a*o2,a*o2,o2,-a*o2,a*o2], (1-N[1],-1,0,1,N[1]-1), N[1], N[1])
        spdiags( [-a*o3,a*o3,o3,-a*o3,a*o3], (1-N[2],-1,0,1,N[2]-1), N[2], N[2])



B_CN = [spdiags( [b*o1,-b*o1,o1,b*o1,-b*o1],  (1-N[0],-1,0,1,N[0]-1), N[0], N[0]
        spdiags( [b*o2,-b*o2,o2,b*o2,-b*o2], (1-N[1],-1,0,1,N[1]-1), N[1], N[1])
        spdiags( [b*o3,-b*o3,o3,b*o3,-b*o3], (1-N[2],-1,0,1,N[2]-1), N[2], N[2])

solutions5 = []
for j in range(len(N)):
    solutions5.append( np.zeros((N[j],K[j])) )

for k in range(3):
```

```
    #initializes conditions to start with
    solutions5[k][:,0] = f(x[k])
    C = np.matmul(np.linalg.inv(A_CN[k]), B_CN[k])
    for j in range(1,K[k]):
        #using matrix corresponding to steps, we sample. We then set initial equ
        #so we can repeat this process
        solutions5[k][:,j] = C.dot(solutions5[k][:,j-1])

f1_CN = solutions5[0]
f2_CN = solutions5[1]
f3_CN = solutions5[2]

plt.plot(x[0], f1_CN[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Crank-Nicholson Method for Transport Equation' +',' + 'h= ' + str(1/2

plt.plot(x[1], f2_CN[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Crank-Nicholson Method for Transport Equation' +',' + 'h= ' + str(1/2

plt.plot(x[2], f3_CN[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Crank-Nicholson Method for Transport Equation' +',' + 'h= ' + str(1/2

plt.plot(x[0], f1_CN[:,-1])
plt.plot(x[1], f2_CN[:,-1])
plt.plot(x[2], f3_CN[:,-1])
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Crank-Nicholson Method for Transport Equation')
plt.legend(['h =' + str(1/2**5), 'h =' + str(1/2**6),'h =' + str(1/2**7)])


### Error calculations in h norm
def h_norm(x,y):
    norm = np.linalg.norm(x-y)
    return norm

### Theoreitcal soln
def u(x,t):
    #let's us define a piecewise function
    tau = 2*np.pi
    z = x+t
    z = z % tau
    y= np.piecewise(
        z,
        [(0 <= z)&(z <= np.pi),  (np.pi <= z)&(z <= 2*np.pi)      ],
```

```
        [lambda z: z, lambda z: 2*np.pi - (z), 0])
    return y
plt.plot(x[2],u(x[2],1))
plt.xlabel('x')
plt.ylabel('u(x,1)')
plt.title('Analytic Solution for Transport Equation')

### UPWIND, CENTRAL, LAX-WENDROFF, LAX-FRIEDRICHS, BACKWARDS EULER, CRANK-NICHOL

Upwind_error = h_norm(f3[:,-1],u(x[2],1))
Central_error = h_norm(f3_CD[:,-1],u(x[2],1))
Lax_Friedrichs_error = h_norm(f3_LF[:,-1],u(x[2],1))
Lax_Wendroff_error = h_norm(f3_LW[:,-1],u(x[2],1))
Backwards_Euler_Error = h_norm(f3_BE[:,-1],u(x[2],1))
Crank_Nicholson_Error = h_norm(f3_CN[:,-1],u(x[2],1))
```