University of Central Florida

Numerical Analysis of the Sine-Gordon Equation

Kevin R. Fechtel & Mark S. Melendez

Numerical Methods for Differential Equations

Dr. Constance Schober

March 16, 2016

1. Introduction

The sine-Gordon equation (sG equation) was first discovered in 1862 by Jacques Edmond Émile Bour, a French engineer who spent most of his academic life studying the deformation of surfaces in differential geometry. In particular, he studied surfaces with a constant negative curvature. The equation was later rediscovered in the 20th century to have applications in the motion of dislocated crystals and supercurrents in condensed matter physics. Even later in the 1970's, the sG equation was found to have soliton solutions, i.e., solutions in the form of a wave that have a constant velocity and a shape that is unaltered.

2. Derivation of the Sine-Gordon & Verification of the Constants of Motion

The derivation of the sine-Gordon equation starts with the principle of least action in classical mechanics. If we minimize the functional,

$$S[q(s)] = \int_{s_1}^{s_2} L(q, \frac{\partial q}{\partial s}, s)\, ds,$$

where S is the functional and q(s) is a function on the sq-plane; then we see that:

$$\delta S = S[\overline{q} + \delta q] - S[\overline{q}] = 0,$$

where $\overline{q} = \overline{q}(s)$ is the curve that minimizes S and must hold. After calculating $\delta S = 0$ to order δq, we see that:

$$\frac{d}{ds}(\frac{\partial L}{\partial q_s}) - \frac{\partial L}{\partial q} = 0,$$

where $q_s = \frac{\partial q}{\partial s}$ and the equation must hold in order to minimize $S[q(s)]$.

The equation,

$$\frac{d}{ds}(\frac{\partial L}{\partial q_s}) - \frac{\partial L}{\partial q} = 0,$$

is in fact what is called the Euler-Lagrange equation. Now, to derive the sG-equation, we will use a form of the Euler-Lagrange equation written with four-gradients, written as:

$$\frac{\partial L}{\partial u} - \partial_{\mu}(\frac{\partial L}{\partial(\partial_{\mu} u)}) = 0.$$

Where u is a function. For clarity, we are using the four-gradients from special relativity, where

$$\partial_{\mu} = (\frac{1}{c}\frac{\partial}{\partial t}, \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}),$$
$$\partial^{\mu} = (\frac{1}{c}\frac{\partial}{\partial t}, \frac{-\partial}{\partial x}, \frac{-\partial}{\partial y}, \frac{-\partial}{\partial z}),$$
and $c = 1$

for our purposes. If we let our Lagrangian density be equal to:

$$L = \partial_{\mu} u \partial^{\mu} u - f(u)$$

and substitute it into the Euler-Lagrange equation, we get:

$$\frac{\partial}{\partial u}(\partial_{\mu} u \partial^{\mu} u - f(u)) - \partial_{\mu}[\frac{\partial}{\partial(\partial_{\mu} u)}(\partial_{\mu} u \partial^{\mu} u - f(u))$$
$$\Rightarrow \frac{-\partial f(u)}{\partial u} - \partial_{\mu}\partial^{\mu} u = 0$$
$$\Rightarrow \frac{\partial f(u)}{\partial u} + \partial_{\mu}\partial^{\mu} u = 0.$$

Finally, if we let $f(u) =- \cos u$ and exclude the y and z dimensions in the four-gradient we arrive at the (1+1) sine-Gordon equation,

$$u_{tt} - u_{xx} + \sin u = 0.$$

For the sine-Gordon equation, Alwyn Scott says that through Noether's Theorem, there are comparison of our approximations and exact solutions are the energy conservation equation,

$$\frac{\partial}{\partial t}\left(\frac{1}{2}u_t^2 + \frac{1}{2}u_x^2 - \cos u\right) - \frac{\partial}{\partial x}(u_t u_x),$$

and the Hamiltonian density of the kink soliton,

$$H = \frac{1}{2}[u_x^2 + u_t^2 + 2(1 - \cos u)]$$

where $u(x, t)$will be the kink solution.

We will verify that the Hamiltonian is a constant of motion. By using the facts that:

$$\frac{\partial}{\partial x}[\tan^{-1}(f)] = \frac{1}{1+f^2}\frac{\partial f}{\partial x}$$

where $f(x)$is a function, and

$$sech(x) = \frac{2}{e^x+e^{-x}},$$

we find the partial derivatives of the kink solution (1) with respect to x and t to be:

$$u_x = 2v[sech(\frac{x-vt}{\sqrt{1-v^2}} + a)]$$

and

$$u_t = \frac{2v}{\sqrt{1-v^2}}[sech(\frac{x-vt}{\sqrt{1-v^2}} + a)].$$

and by the extensions of the sine and inverse tangent functions to their complex arguments z, that is:

$$\sin z = \frac{e^{iz}-e^{-iz}}{2i}$$

and

$$\tan^{-1} z = \frac{1}{2}i[\ln(1 - iz) - \ln(1 + iz)],$$

we find the versine of the kink solution to be:

countably infinite constants of motion (Scott 903). Two of the constants of motion that we will use for

Now to substitute our results into the improper integral of the Hamiltonian we have:

$$\int_{-\infty}^{\infty}\frac{1}{2}[u_x^2 + u_t^2 + 2(1 - \cos u)]\,dx$$

$$= \int_{-\infty}^{\infty} 2[(\frac{1}{1-v^2} + \frac{v^2}{1-v^2} + 1)sech^2(\frac{x-vt}{\sqrt{1-v^2}} + a)]\,dx$$

$$= \int_{-\infty}^{\infty} 4[\frac{1}{1-v^2}sech^2(\frac{x-vt}{\sqrt{1-v^2}} + a)]\,dx$$

$$= \frac{4}{1-v^2}\int_{-\infty}^{\infty} sech^2(\frac{x-vt}{\sqrt{1-v^2}} + a)\,dx$$

Now, let:

$$u = \frac{x-vt}{\sqrt{1-v^2}} + a$$

and

$$\frac{\partial u}{\partial x} = \frac{1}{\sqrt{1-v^2}} \Rightarrow dx = \sqrt{1 - v^2}du.$$

This gives us:

$$\frac{4}{\sqrt{1-v^2}}\int_{u=-\infty}^{u=\infty} sech^2 u\,du$$

$$= \lim_{u \to \infty}\frac{4}{\sqrt{1-v^2}}[\tan^{-1}(u) - \tan^{-1}(-u)]$$

$$= \frac{8}{\sqrt{1-v^2}},$$

which is a constant. Therefore the Hamiltonian for the kink solution specifically is a constant of motion.

We can also prove constants of motion for general functions. We will now proceed with verifying that the energy conservation equation for

$$1 - \cos u = 2\operatorname{sech}^2\left(\frac{x-vt}{\sqrt{1-v^2}} + a\right).$$

$$\int_{-\infty}^{\infty} \frac{\partial}{\partial t}\left(\frac{1}{2}u_t^2 + \frac{1}{2}u_x^2 - \cos u\right) - \frac{\partial}{\partial x}(u_x u_t)\, dx$$

$$= \int_{-\infty}^{\infty} (u_t u_{tt} + u_x u_{xt} + u_t \sin u) - (u_x u_{tx} + u_t u_{xx})\, dx$$
$$= \int_{-\infty}^{\infty} u_t(u_{tt} - u_{xx} + \sin u)\, dx$$
$$= \int_{-\infty}^{\infty} 0\, dx$$
$$= 0,$$

which is a constant. Therefore the energy conservation equation is also a equation of motion.

3. Special Solutions and Numerical Methods Used

Some interesting solutions that come from the sine-Gordon equation are solitons, solutions in the form of a wave that have a constant velocity and a shape that is unaltered. One soliton solution is the "kink" solution, which can be found by first converting the sG equation into light cone coordinates and then computing a Bäcklund transformation [2]. The special solution is of the form:

$$u(x,t) = 4\tan^{-1}\left(\exp\left(\frac{x-vt}{\sqrt{1-v^2}}\right)\right) v \in [0,1)$$

(1)

The kink solution can be thought of as a traveling wave at a positive velocity with it's travel corresponding to the heteroclinic orbit of a simple pendulum.

Another related solution we will use is the kink-kink collision (also from [2]). We can visually the sine-Gordon PDE is also a equation of motion. By making substitutions with the sG equation, we get:

applying the Bäcklund transform to the 1-soliton solutions.

The last special solution we will use is the double-pole solution (from [3]). What is unique for this soliton is that it does not depend on the variable v. The double-pole soliton is written in the form:

$$u(x,t) = 4\tan^{-1}(t\operatorname{sech}(x)).$$

In our numerical analysis, we will be solving the (1+1) nonlinear sine-Gordon equation with Dirichlet boundary conditions. The numerical methods we will use to solve the sG equation are modified versions of the box scheme, the Crank-Nicolson scheme, and the Lax-Wendroff scheme. The box scheme, also called the Preissman box scheme, is constructed by taking the central difference approximation to the 2nd order partial derivative of $u(x,t)$with respect to both x and$y$. Substituting these approximations into the sG equation yields:

$$\frac{U_j^{n+1} - 2U_j^n + U_j^{n-1}}{k^2} = \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{h^2} - \sin U_j^n,$$

implying that;

$$U_j^{n+1} = 2U_j^n - U_j^{n-1} + \frac{k^2}{h^2}(U_{j+1}^n - 2U_j^n + U_{j-1}^n) - \sin U$$

which is our modified box scheme.

We have also decided to use a modified Crank-Nicolson scheme. The regular Crank-Nicolson scheme can be written in this form:

think about this as two kink solitons at opposite velocities passing through each other elastically. The solution is written as:

$$u(x,t) = 4\,tan^{-1}\left(\frac{v\,sinh(\,x/\sqrt{1-v^2}\,)}{cosh(\,vt/\sqrt{1-v^2}\,)}\right)\ \ where\ v \in [0, 1)$$

.

The kink-kink collision solution as well as all of the other multi-soliton solutions can be derived by

$$(2\varsigma + 1)U_j^{n+1} - \varsigma U_{j+1}^{n+1} - \varsigma U_{j-1}^{n+1} = 2(1 - \varsigma)U_j^n + \varsigma U_{j+1}^n + \varsigma U_{j-1}^n - U_j^{n-1} - k$$

where $\varsigma = \frac{k^2}{2h^2}$. Now, because the scheme is implicit, we can solve $U_j^n\ \forall j$ with the matrix equation [1]:

$$AU^{n+1} = B,$$

where

$$A = \begin{bmatrix} 2\varsigma+1 & -\varsigma & 0 & \cdots & 0 \\ -\varsigma & 2\varsigma+1 & -\varsigma & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -\varsigma & 2\varsigma+1 & -\varsigma \\ 0 & \cdots & 0 & -\varsigma & 2\varsigma+1 \end{bmatrix},$$

$$U^{n+1} = \begin{bmatrix} U_1^{n+1} \\ \\ \vdots \\ \\ U_m^{n+1} \end{bmatrix},$$

and

$$B = \begin{bmatrix} \varsigma(g_0(t_n) + g_0(t_{n+1})) + 2(1-\varsigma)U_1^n + \varsigma U_2^n - U_1^{n-1} - sin\,U_1^n \\ \\ \vdots \\ \\ \varsigma(g_1(t_n) + g_1(t_{n+1})) + 2(1-\varsigma)U_1^n + \varsigma U_{m-1}^n - U_m^{n-1} - sin\,U_m^n \end{bmatrix}.$$

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{1}{2h^2}(U_{j+1}^n - 2U_j^n + U_{j-1}^n + U_{j+1}^{n+1} - 2U_j^{n+1} + U$$

Our modification has a central difference approximation to the 2nd order partial derivative of $u(x,t)$with respect to t on the LHS and a negative sine term added on the RHS, i.e.:

$$\frac{U_j^{n+1} - 2U_j^n + U_j^{n-1}}{k^2} = \frac{1}{2h^2}(U_{j+1}^n - 2U_j^n + U_{j-1}^n + U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}) - sin\,U_j^n.$$

The method can be rewritten as:

For the derivation of the scheme, we use the relation $\frac{\partial^n}{\partial t^n}u = (-\,a)^n\frac{\partial^n}{\partial x^n}u$ to make substitutions for the partial derivatives (see [5]) with respect to time in the Taylor polynomial truncated at degree 2. This gives us:

$$u(x, t + \Delta t) \approx u - \Delta t a u_x + \frac{(\Delta t)^2 a^2}{2}u_{xx},$$

which we will now discretize with 1st and 2nd order central differences. After simple algebraic manipulations, this yields us:

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_{j+1}^n - U_{j-1}^n) + \frac{a^2k^2}{2h^2}(U_{j+1}^n - 2U_j^n + U$$

,

the Lax-Wendroff scheme. We will make a simple but powerful modification to this method. We will add a negative sine term on the RHS of the scheme, i.e.,

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_{j+1}^n - U_{j-1}^n) + \frac{a^2k^2}{2h^2}(U_{j+1}^n - 2U_j^n + U_{j-1}^n) - sin\,U$$

We will use all of the aforementioned methods and special solutions to approximate the sG equation with Dirichlet boundary conditions in the proceeding section.

4. Numerical Analysis

Because both the modified box method and the modified Crank-Nicolson method are multi-level schemes, we must use a single level scheme for the initial time step. We have chosen to use a modified form of the Lax-Wendroff method for the sine-Gordon equation. Let us derive this by starting with the Taylor series expansion of $u(x, t + \Delta t)$, that is:

$$u(x, t + \Delta t) = u + \Delta t u_t + \frac{(\Delta t)^2}{2} u_{tt} + O((\Delta t)^3).$$

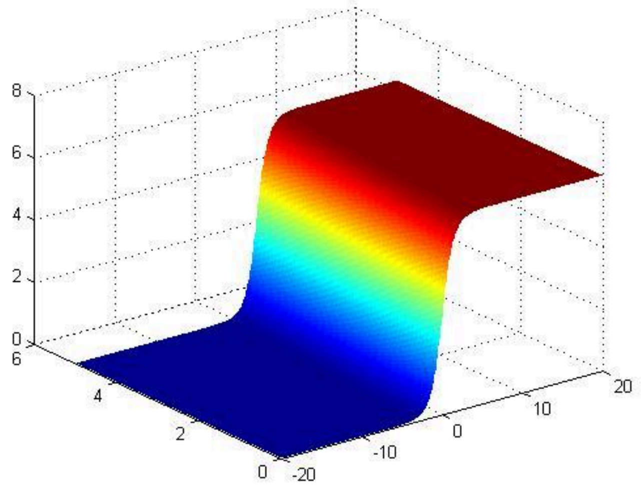


Figure 1.1: Exact Kink Soliton

We will proceed with ten different experiments to approximate solutions to the sine-Gordon equation. As mentioned earlier, because the Crank-Nicolson method and the Box method are both multi-level schemes, we will need a single level scheme to compute the first time step. We have chosen to use the Lax-Wendroff method as our single level scheme. But, having another single level scheme would be useful for comparison with the accuracy of the Lax-Wendroff method. Rather than choosing another scheme per se, we will instead generate the first time step by the exact solution itself.

Our hope is to have graphs accurate to all three special solutions. For reference, the the exact graphs to the three special solutions are as follows:

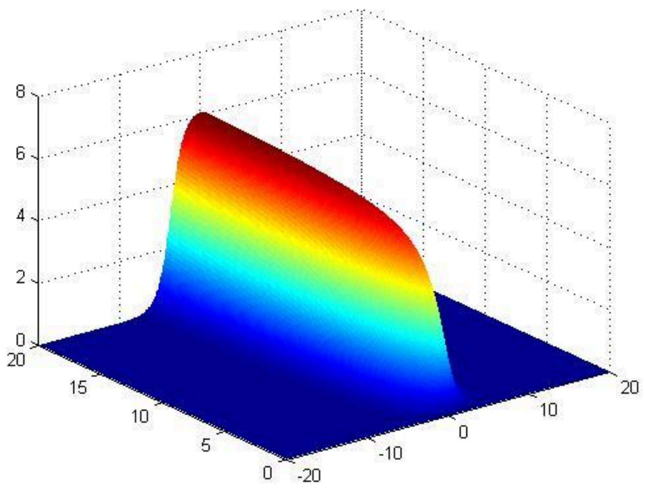


Figure 1.3: Exact Double Pole Soliton

So all in all we have two single level schemes, two multi-level schemes, and three special solutions, which we will compute ten out of the twelve combinations in total to experiment and record. We exclude the Crank-Nicolson and Box scheme with the Lax-Wendroff method on the double pole solution for reasons mentioned in section 5.

For our tables describing each of our results, m is defined to be the number of points that make up

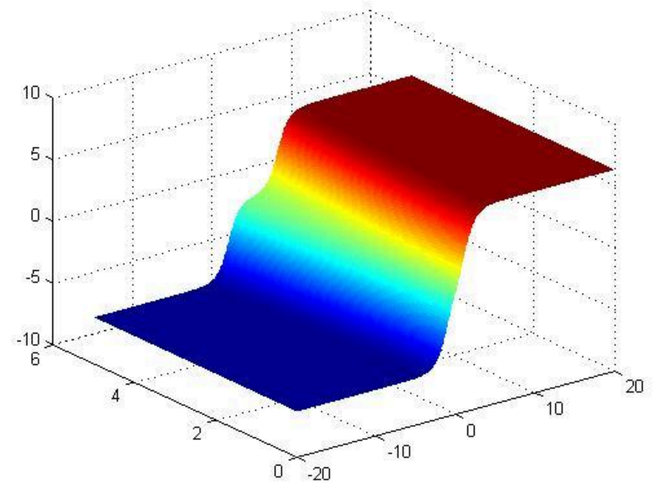


Figure 1.2: Exact Kink Kink Collision

We will first look at the kink soliton generated by the Crank-Nicolson method (C-N method) with the first time step coming from the exact solution.

Kink soliton w/C-N method and exact solution

m	Δx	Δt	E_∞	E_2
100	0.3690	0.0158	0.0939	0.2048
200	0.1990	0.0080	0.0395	0.1072
300	0.1329	0.0053	0.0577	0.1661
400	0.0998	0.0040	0.0759	0.2414

With the log-log plot of E_∞ versus Δx,

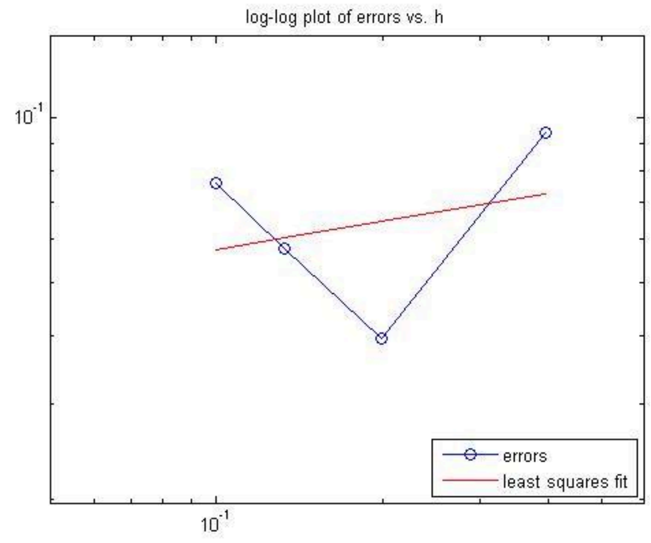
the graph, Δx is the space step, and Δt is the time step. We will measure the error of the solutions by:

$$E_\infty = \max\left(\|\vec{e}_1\|_\infty, \cdots, \|\vec{e}_n\|_\infty\right),$$

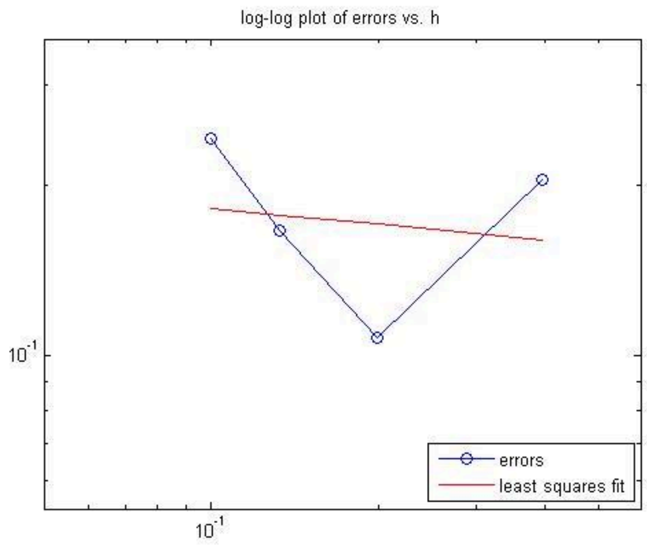
which is the maximum of the max norms of the error vectors, and also:

$$E_2 = \max\left(\|\vec{e}_1\|_2, \cdots, \|\vec{e}_n\|_2\right),$$

which is the maximum of the l^2- norms of the error vectors. The intervals over time and space that we will graph our solutions in are $[0, 5]$ and $[-20, 20]$, respectively.
the log-log plot of E_2 versus Δx,



and the graph of the solution when **m**$= 400$,



We will now look at the kink-kink collision generated by the Crank-Nicolson method with the first time step coming from the exact solution.

Kink-kink collision w/C-N method and exact solution

m	Δx	Δt	E_{∞}	E_{2}
100	0.3960	0.0158	0.0784	0.2013
200	0.1990	0.0080	0.0395	0.1073
300	0.1329	0.0053	0.0577	0.1658
400	0.0998	0.0040	0.0754	0.2413

With the log-log plot of E_{∞} versus Δx,

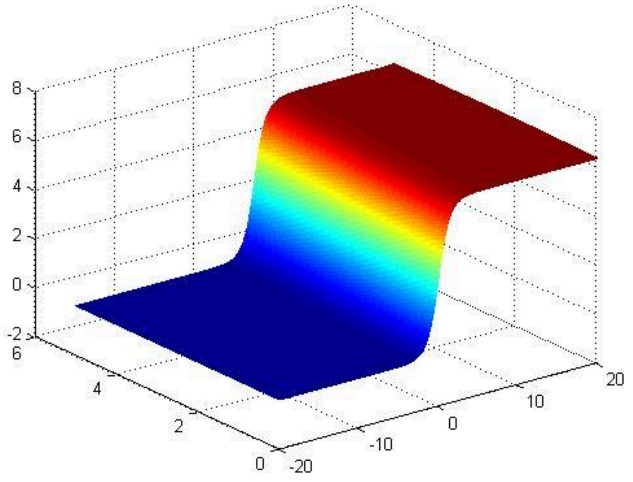
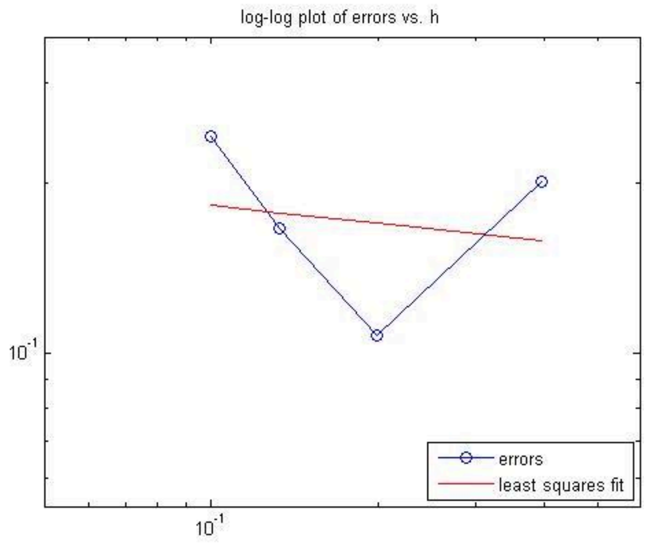
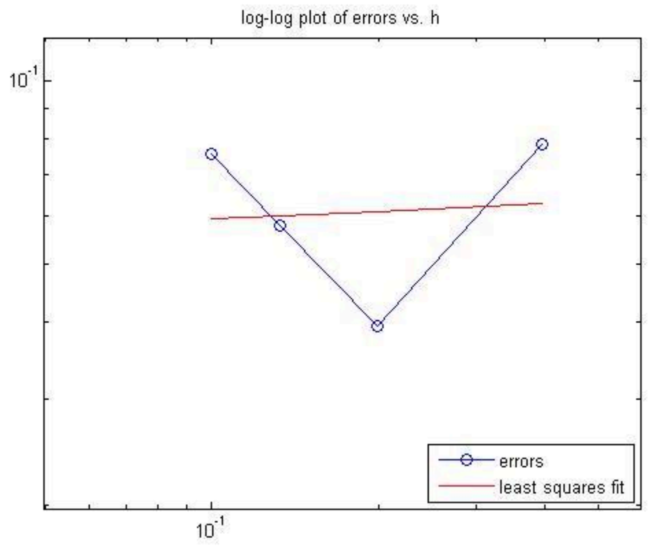


Figure 2.1: Kink soliton generated by the Crank-Nicolson method and the exact solution

the log-log plot of E_{2} versus Δx,



and the graph of the solution when **m**$=\ 400$,



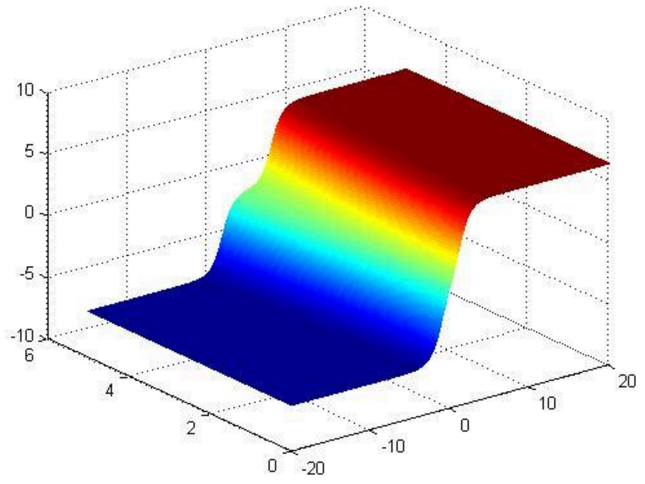


Figure 2.2: Kink-kink collision generated by the Crank-Nicolson method and the exact solution

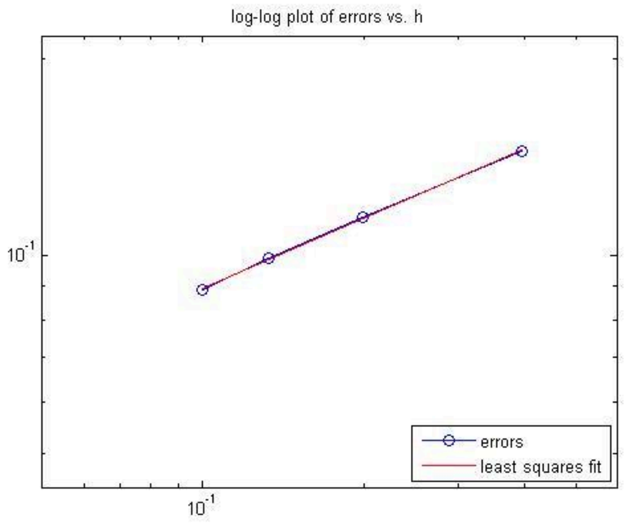
We now look at the double pole soliton generated by the Crank-Nicolson method with the first time step coming from the exact solution.

Double pole soliton w/C-N method and exact solution

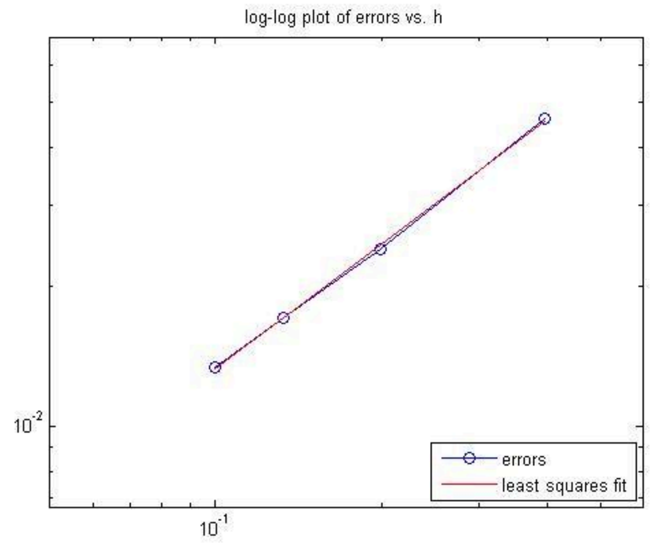
m	Δx	Δt	E_∞	E_2
100	0.3690	0.0158	0.0461	0.1445
200	0.1990	0.0080	0.0240	0.1147
300	0.1329	0.0053	0.0172	0.0993
400	0.0998	0.0040	0.0134	0.0888

With the log-log plot of E_∞ versus Δx,

The log-log plot of E_n versus Δx,



and the graph of the solution when **m**= 400,



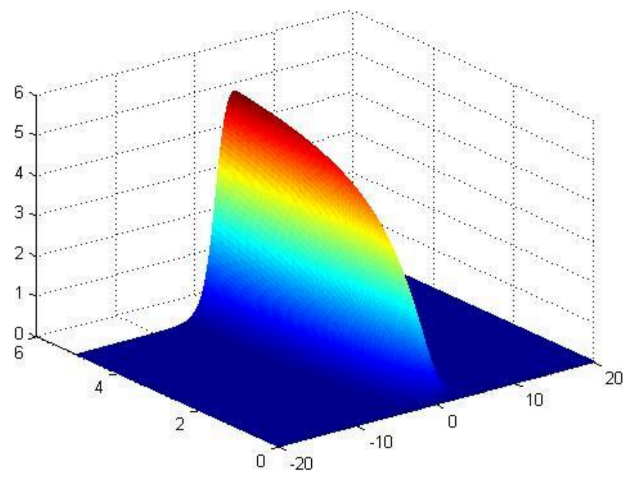


Figure 2.3: Double pole soliton generated by the Crank-Nicolson method and the exact solution

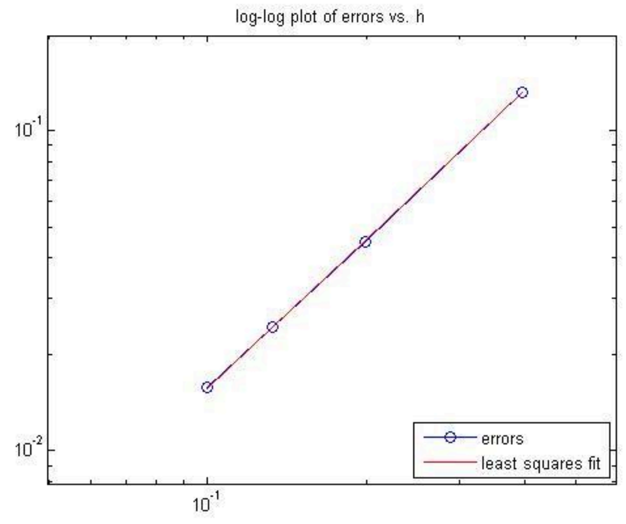
Here is the kink soliton generated by the Box method with the first time step coming from the exact solution.

Kink soliton w/Box method and exact solution

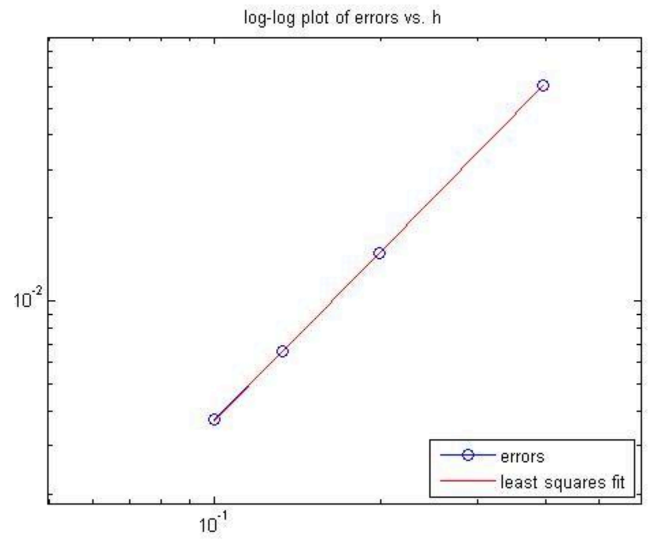
m	Δx	Δt	E_∞	E_2
100	0.3690	0.0158	0.0602	0.1320
200	0.1990	0.0080	0.0149	0.0450
300	0.1329	0.0053	0.0066	0.0243
400	0.0998	0.0040	0.0037	0.0158

With the log-log plot of E_∞ versus Δx,

the log-log plot of E_2 versus Δx,



and the graph of the solution when **m**$=400$,



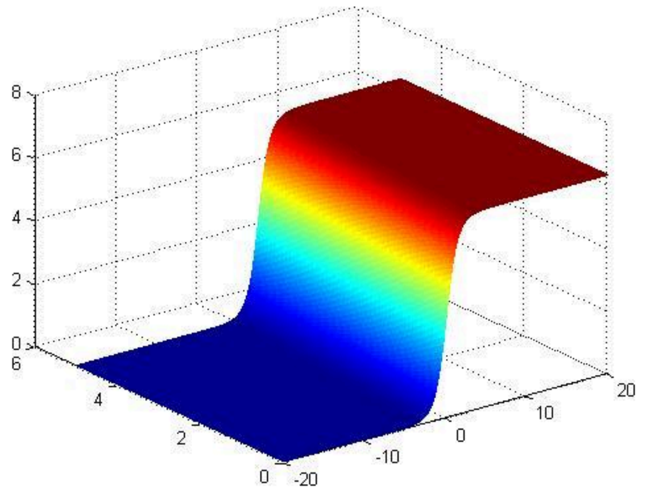


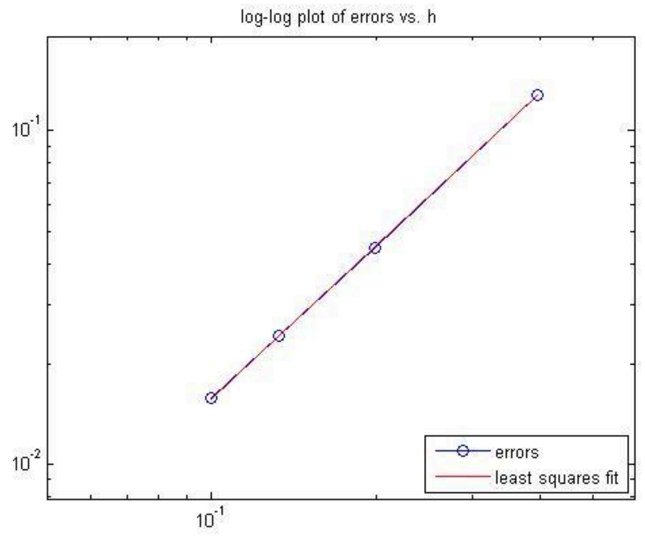
Figure 2.4: Kink soliton generated by the Box method and the exact solution

We now will look at the kink-kink collision generated by the Box method with the first time step coming from the exact solution.

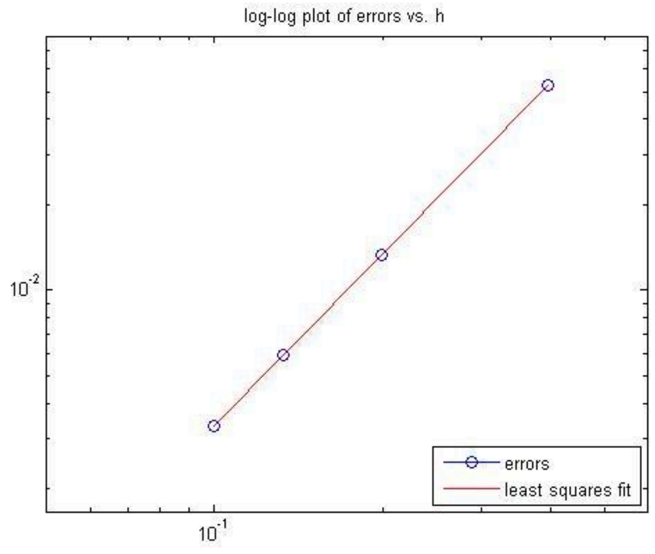
Kink-kink collision w/Box method and exact solution

m	Δx	Δt	E_{∞}	E_2
100	0.3690	0.0158	0.0523	0.1281
200	0.1990	0.0080	0.0133	0.0447
300	0.1329	0.0053	0.0059	0.0243
400	0.0998	0.0040	0.0033	0.0158

the log-log plot of E_2versus Δx,



and the graph of the solution when **m**$= 400$, With the log-log plot of E_{∞}versus Δx,



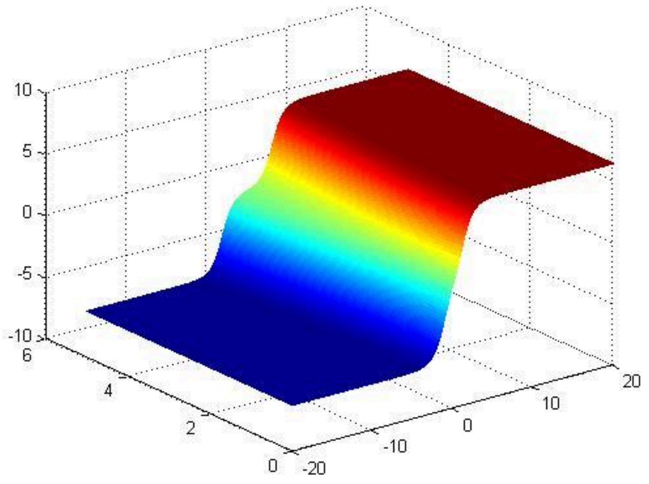
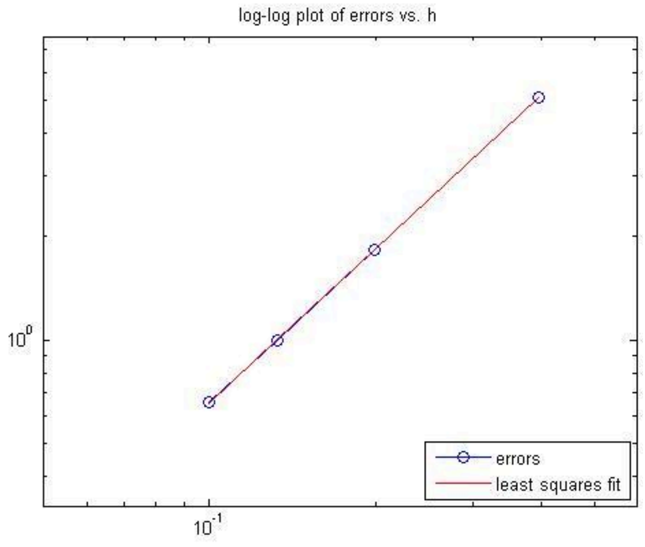


Figure 2.5: Kink-kink collision generated by the Box method and the exact solution

the log-log plot of E_{2}versus Δx,

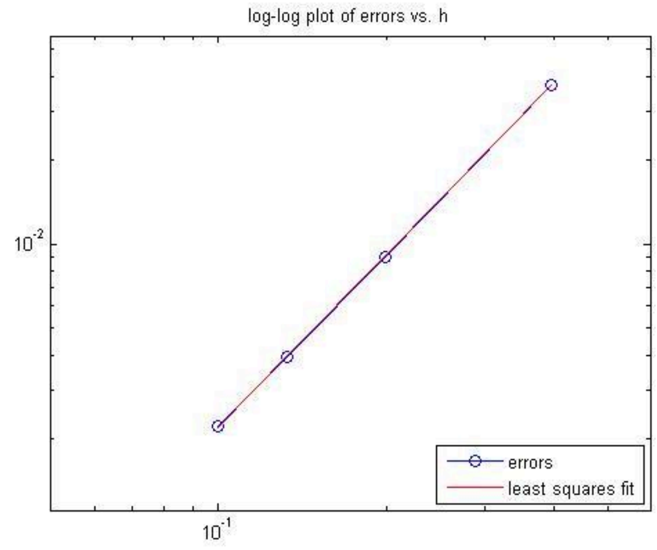


Now, here is the double pole soliton generated by the Box method with the first time step coming from the exact solution.

Double pole soliton w/Box method and exact solution

m	Δx	Δt	E_{∞}	E_{2}
100	0.3690	0.0158	0.0373	0.1175
200	0.1990	0.0080	0.0090	0.0414
300	0.1329	0.0053	0.0040	0.0225
400	0.0998	0.0040	0.0022	0.0146

and the graph of the solution when **m**$=\ 400$, With the log-log plot of E_∞ versus Δx,



Here is the kink soliton generated by the Crank-Nicolson method with the first time step coming from the Lax-Wendroff method. All of our experiments using the Lax-Wendroff method contain graphs of the first step in time.

Kink soliton w/C-N and Lax-Wendroff method

m	Δx	Δt	E_∞	E_2
100	0.3690	0.0158	3.1822	7.3414
200	0.1990	0.0080	4.0684	14.086
300	0.1329	0.0053	4.2165	18.078
400	0.0998	0.0040	4.2686	21.221

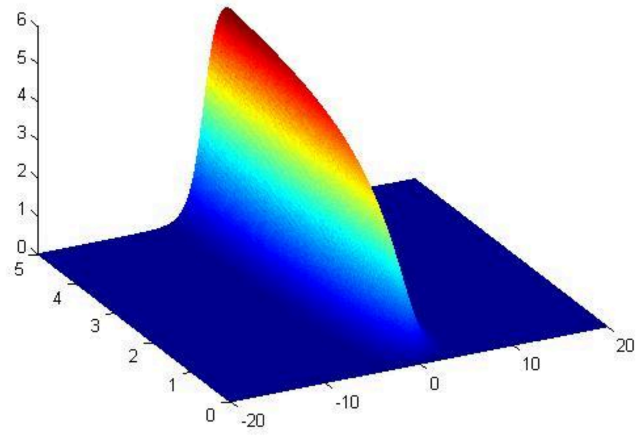
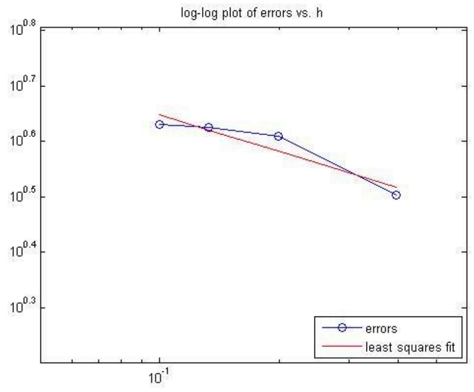


Figure 2.6: Double pole soliton generated by the Box method and the exact solution

With the log-log plot of E_∞ versus Δx,



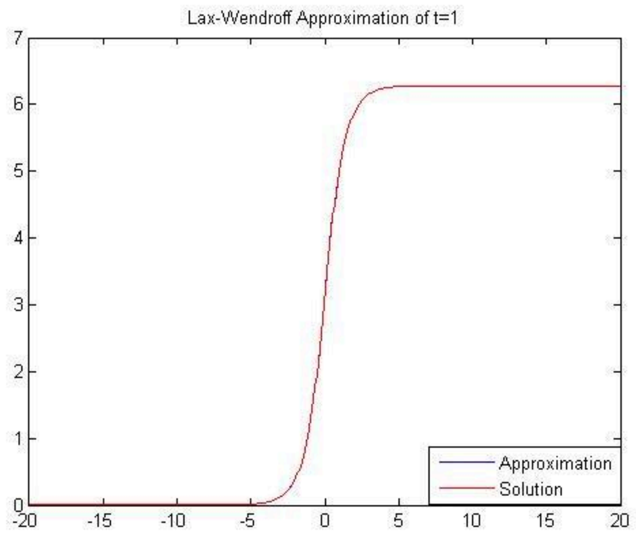
the log-log plot of E_2 versus Δx,

The graph for the Lax-Wendroff approximation of the first time step is shown below. Results show that:

$$\|\vec{e}_1\|_\infty = 0.0125,$$

and

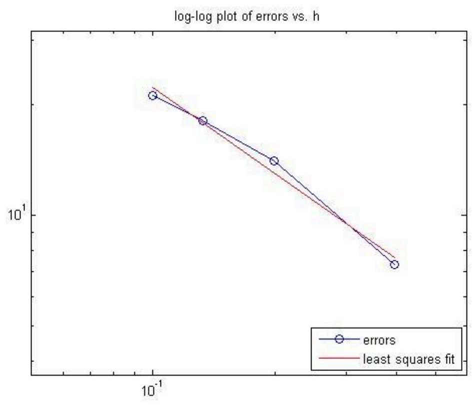
$$\|\vec{e}_1\|_2 = 0.0264.$$



We now look at the kink-kink collision generated by the Crank-Nicolson method with the first time step coming from the Lax-Wendroff method.

Kink-kink collision w/C-N and Lax-Wendroff method

m	Δx	Δt	E_∞	E_2
100	0.3690	0.0158	1.9079	5.1054
200	0.1990	0.0080	0.4792	1.8227
300	0.1329	0.0053	0.2167	0.9993
400	0.0998	0.0040	0.1247	0.6582



and the graph of the solution when **m**= 400,

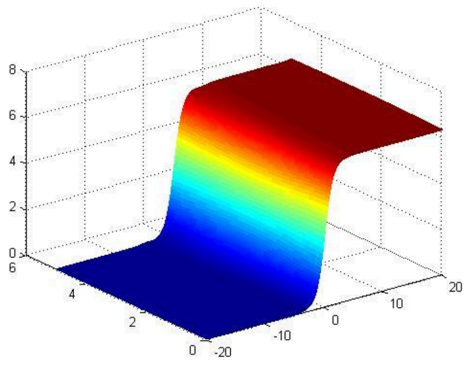
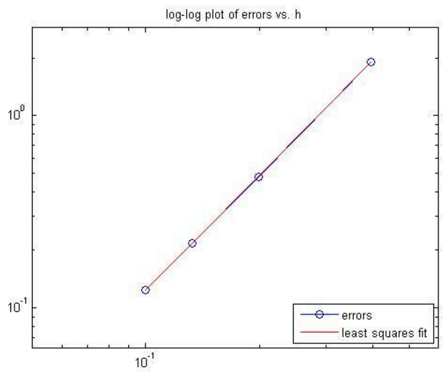


Figure 3.1: Kink soliton generated by the Crank-Nicolson method and the Lax-Wendroff method

With the log-log plot of E_∞ versus Δx,



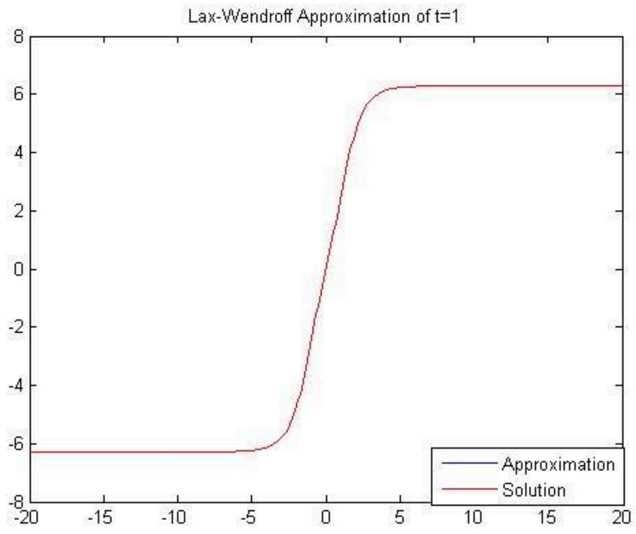
the log-log plot of E_2 versus Δx,

The graph for the Lax-Wendroff approximation of the first time step is shown below. Results show that:

$$||\vec{e_1}||_\infty = 1.1466 \times 10^{-4},$$

and

$$||\vec{e_1}||_2 = 0.6429.$$

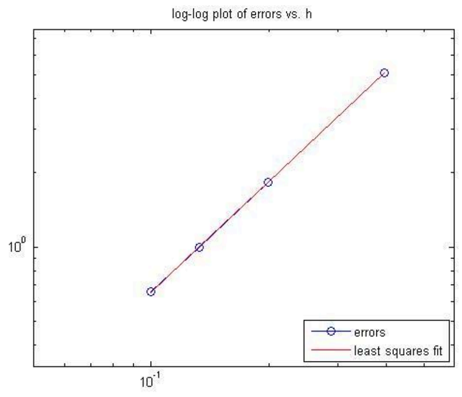


Now we will look at the kink soliton generated by the Box method with the first time step coming from the Lax-Wendroff method.

Kink soliton w/Box and Lax-Wendroff method

m	Δx	Δt	E_∞	E_2
100	0.3690	0.0158	3.1755	7.3171
200	0.1990	0.0080	4.0688	14.084
300	0.1329	0.0053	4.2168	18.078
400	0.0998	0.0040	4.2690	21.221

The graph for the Lax-Wendroff approximation of the first time step is shown below. Results show that:



and the graph of the solution when **m**= 400,

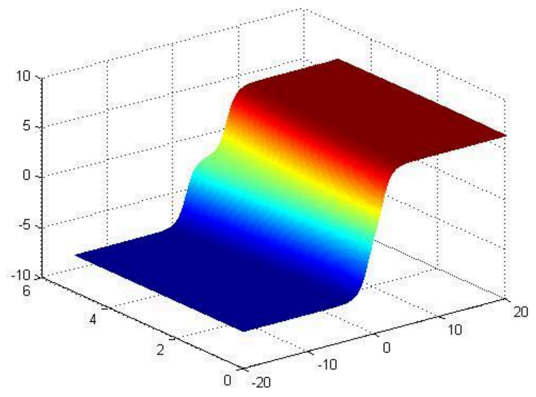
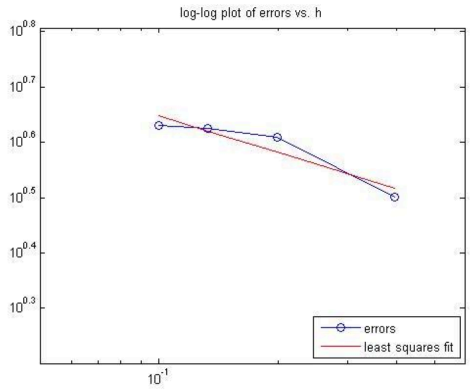


Figure 3.2: Kink-kink collision generated by the Crank-Nicolson method and the Lax-Wendroff method

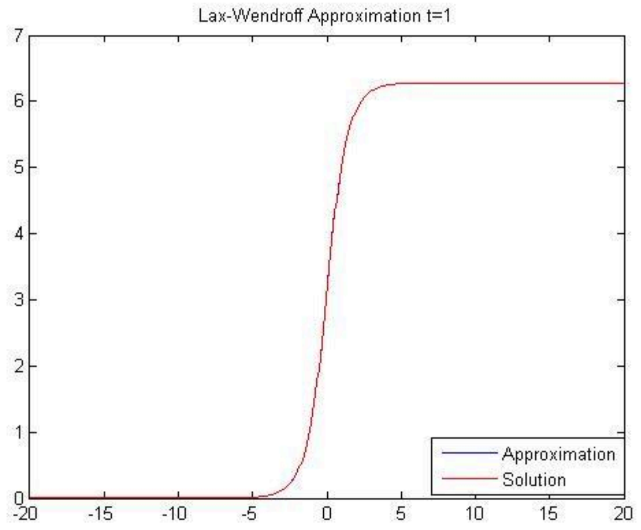
With the log-log plot of E_∞ versus Δx,



$$\|\vec{e_1}\|_\infty = 0.0045,$$

and

$$\|\vec{e_1}\|_2 = 0.0188.$$

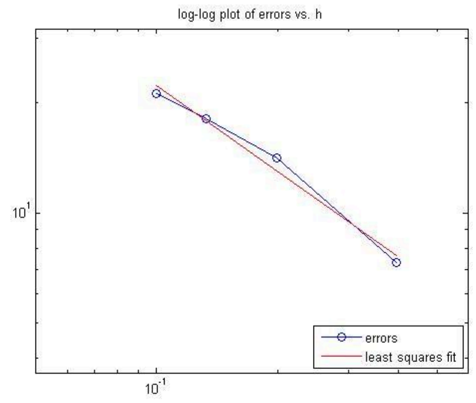


We will finally look at the kink-kink collision generated by the Box method with the first time step coming from the Lax-Wendroff method.

Kink-kink collision w/Box and Lax-Wendroff method

m	Δx	Δt	E_∞	E_2
100	0.3690	0.0158	1.8971	5.1130
200	0.1990	0.0080	0.4652	1.8204
300	0.1329	0.0053	0.2069	0.9939
400	0.0998	0.0040	0.1172	0.6472

the log-log plot of E_2 versus Δx,



and the graph of the solution when **m**= 400,

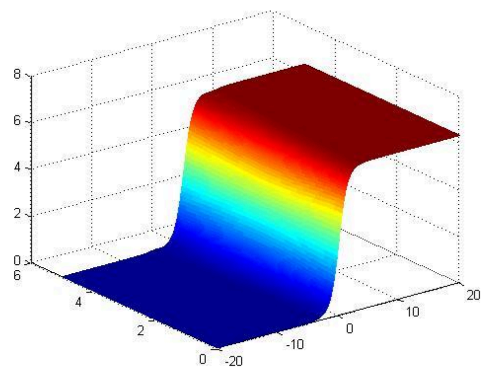
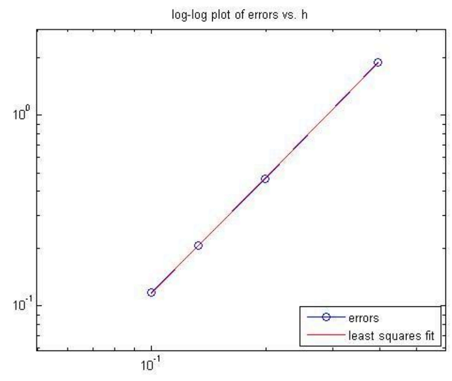


Figure 3.3: Kink soliton generated by the Box method and the Lax-Wendroff method

With the log-log plot of E_∞ versus Δx,

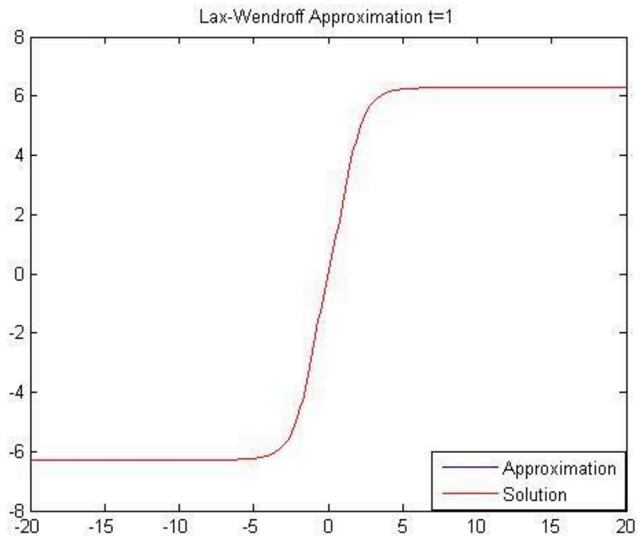


The graph for the Lax-Wendroff approximation of the first time step is shown below. Results show that:

$$||\vec{e_1}||_\infty = 1.1466 \times 10^{-4},$$

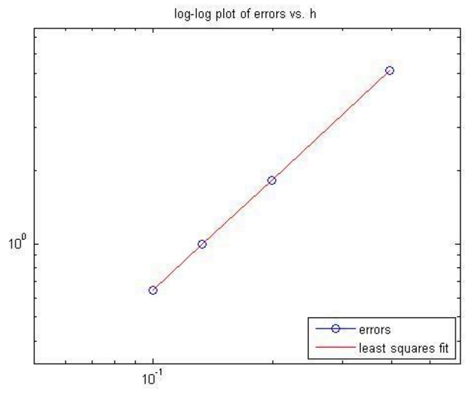
and

$$||\vec{e_1}||_2 = 6.4299 \times 10^{-4}.$$



For additional analysis, we compute the evolution of error on the Hamiltonian density and conservation of energy constants of motion. We compute the error by taking a slice in time (i.e., a vector) of the exact and approximate solutions to the kink, kink-kink collision, and the double pole solitons. With those vectors, we substitute them into a discretization of the Hamiltonian density and the conservation of energy. We obtain another vector, that which we numerically integrate by implementing the trapezoidal rule. After integrating each pair of exact solution and approximate solution vectors, we obtain two scalars. We take the difference of both scalars and take the absolute value of their difference.

Because we compute the aforementioned process for each column vector from the matrix (which itself comes from the mesh), we obtain a vector of errors. the log-log plot of E_2 versus Δx,



and the graph of the solution when **m**= 400,

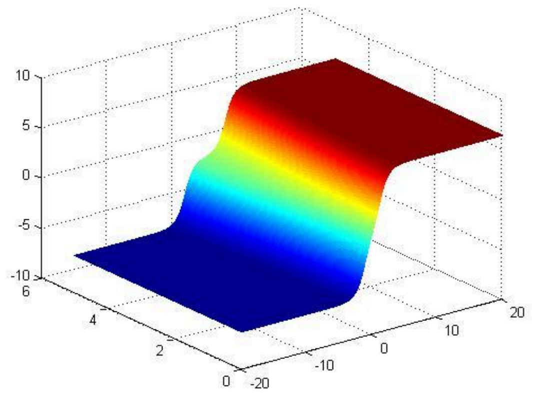


Figure 3.4: Kink-kink collision generated by the Box method and the Lax-Wendroff method

And to have a scalar describing our error rather than a vector, we take the max norm of the error vector. In the graphs below, you can see the evolution of the error for all of the experiments above excluding the experiments that use the exact solution for the first time step.

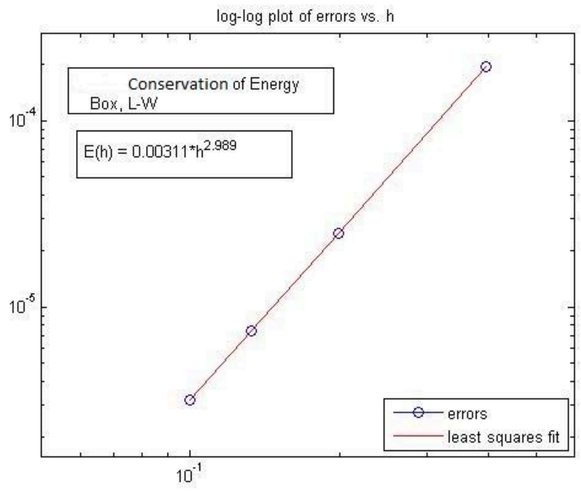


Figure 4.1: Evolution of error on the conservation of energy with the Box scheme, the Lax-Wendroff scheme, and the kink soliton

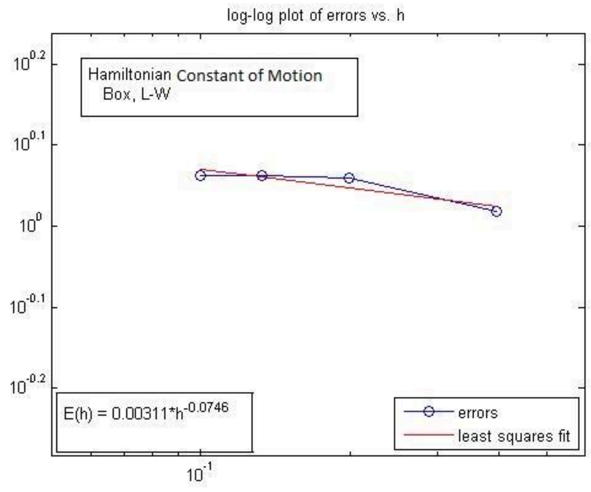


Figure 4.2: Evolution of error on the Hamiltonian with the Box scheme, the Lax-Wendroff scheme, and the kink soliton

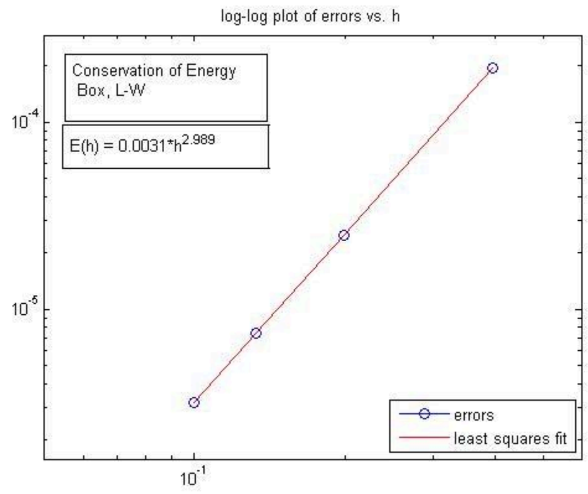


Figure 4.3: Evolution of error on the conservation of energy with the Box scheme, the Lax-Wendroff scheme, and the kink-kink collision

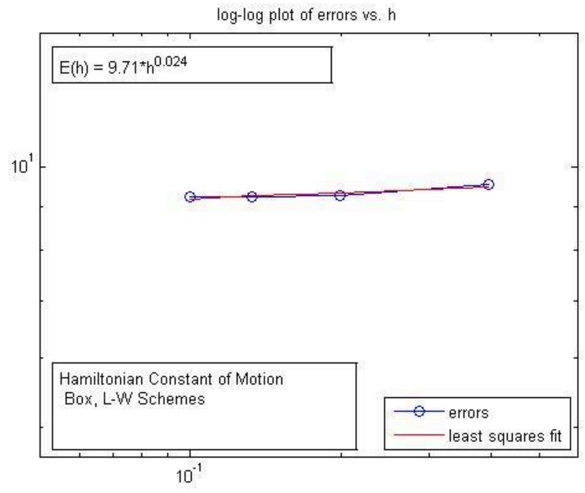


Figure 4.4: Evolution of error on the Hamiltonian with the Box scheme, the Lax-Wendroff scheme, and the kink-kink collision

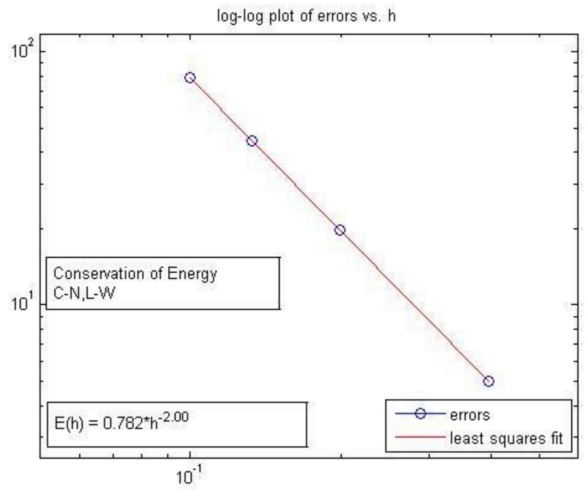


Figure 4.5: Evolution of error on the conservation of energy with the Crank-Nicolson scheme, the Lax-Wendroff scheme, and the kink soliton

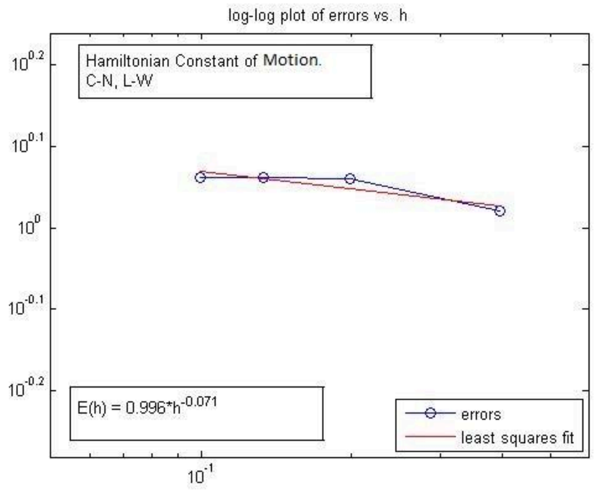


Figure 4.6: Evolution of error on the Hamiltonian with the Crank-Nicolson scheme, the Lax-Wendroff scheme, and the kink soliton

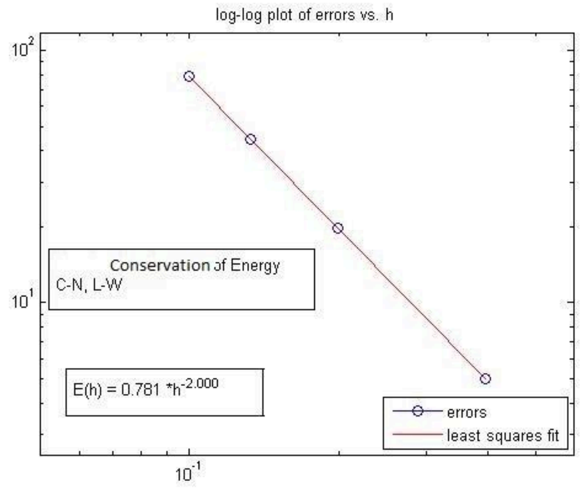


Figure 4.7: Evolution of error on the conservation of energy with the Crank-Nicolson scheme, the Lax-Wendroff scheme, and the kink-kink collision

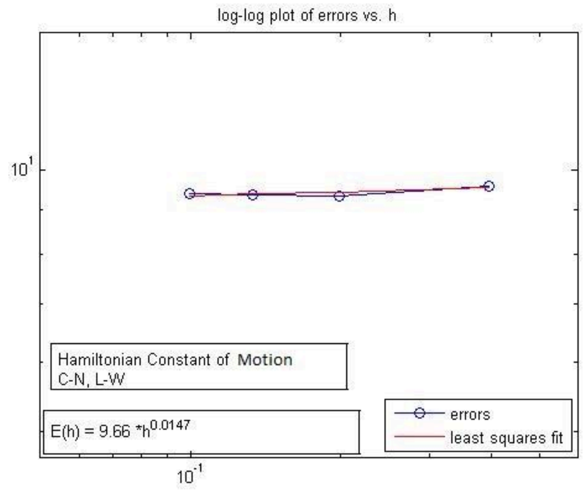


Figure 4.8: Evolution of error on the Hamiltonian with the Crank-Nicolson scheme, the Lax-Wendroff scheme, and the kink-kink collision

5. Conclusion

We will first focus on the C-N method used in with the exact solution for the kink soliton. As shown in the E_∞vs.Δx and the E_2vs.Δx log-log plots for the approximation, the error decreased and then increased as we refined the mesh. The same pattern holds with an increase of time steps, which we have verified this up to $t = 15$. In terms of order, the scheme does not accurately approximate the kink solution.

We will now look a similar scheme: the C-N method used in conjunction with the Lax-Wendroff method for the kink soliton. Although the error in the first time step is approximated well by the Lax-Wendroff scheme, the C-N method greatly exponentiates it. This problem is also seen as we refine our mesh. The increase in error is most noticeable with the l^2- norm.

The kink solution that uses the Box method with the exact solution for the first time step gives us great results. This scheme gives us a nearly identical geometry and order 2 is attained. But as before, if we

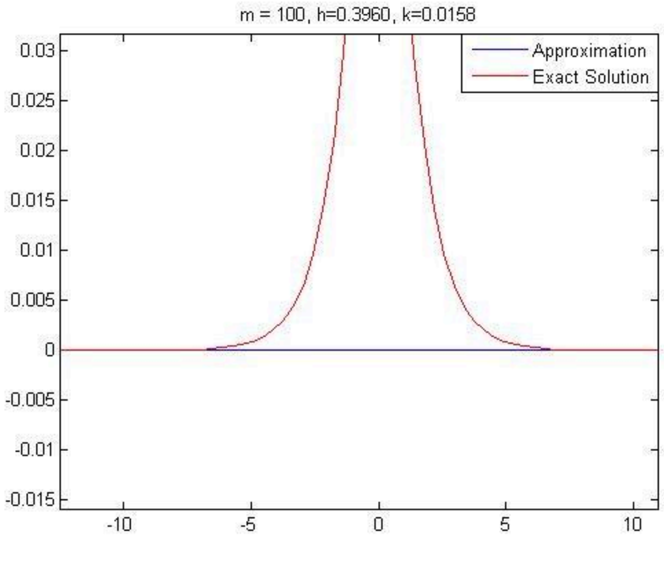


Figure 5.1: Approximation for the first time step of the double pole soliton using the Lax-Wendroff method

substitute using the exact solution with the Lax-Wendroff method, there is a small error created by the method that increases quickly as the mesh is refined.

So it seems that introducing the Lax-Wendroff method yields inaccurate approximations overall. If we approximate the kink-kink collision with the C-N and the first time step of the exact solution, we get the same increasing and decreasing error as with the kink solution approximated by the same scheme. But if we substitute the exact solution with the Lax-Wendroff method, we see that refining the mesh in fact gives us a more precise solution. This result is not what we expected. We hypothesized that using both a multi-level scheme and a single level scheme for the first time step in general would give us a worse approximation in comparison to using only one scheme.

Our best results so far are with the kink-kink collisions approximated by the Box scheme. With the first step in time being computed by the exact solution and the Lax-Wendroff method, we attain order 2 and a decrease of error in general as the mesh is refined.

Our most interesting results are from the double pole soliton. When approximated with the C-N method and the Box method with the exact solution we achieve order 1 for E_∞ and order 2 for E_∞, respectively. However, we are unable to approximate the solution through the use of the Lax-Wendroff method. That is because the initial condition for the double soliton solution, $u(x, 0) = 0$, inadvertently makes the entire Lax-Wendroff approximation zero. The graph of the issue is shown below in figure 5.1.

A possible way around the issue is to generate the first time step of the through the use of the initial condition, $\frac{\partial u}{\partial t}(x, 0) = 4\gamma sech(\gamma x)$, where γ is a non-negative real number. This idea avoids the usage of $u(x, 0) = 0$ and may yield better results.

6. Matlab Code

Before our citations, we have pasted all of the Matlab code that has been used to calculate the approximations of the solutions to the sine-Gordon equation, their corresponding errors, and the evolution of the error on the Hamiltonian and conservation of energy. The code begins on the next page.

```
%Kink Soliton Solution

m=400;

ax = -20;

bx = 20;

  c = 0.5; % velocity of the solitary wave

 Tfinal = 5;

h = (bx-ax)/(m+1);

k=(1/25)*h;

[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);

Z = 4*atan(exp(((X-c.*Y)/sqrt(1-(c^2)) )));

figure

h = surfc(X,Y,Z);

set(h,'edgecolor','none');

%KINK KINK COLLISION SOLUTION GRAPHED


m=400;

ax = -20;

bx = 20;

c = 0.5;

Tfinal = 5;

h = (bx-ax)/(m+1);

x = linspace(ax,bx,m+2);
```

```
t = linspace(ax,bx,m+2);

k=(1/25)*h;

Tsteps = (Tfinal / k);

[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);

Z = 4*atan( (c.*sinh(X./sqrt( 1-(c^2) ))) ./(cosh( c.*Y./(sqrt(1-(c^2))) )) );

figure

h = surfc(X,Y,Z)


set(h,'edgecolor','none');


% DOUBLE POLE SOLITON SOLUTION GRAPHED

m=400;

ax = -20;

bx = 20;

c = 0.5;

 Tfinal = 20;

h = (bx-ax)/(m+1);

x = linspace(ax,bx,m+2);

t = linspace(ax,bx,m+2);

k=(1/25)*h;

Tsteps = (Tfinal / k);

[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);
```

```
Z = 4*atan(Y.*sech(X));
figure
hx = surfc(X,Y,Z);


set(hx,'edgecolor','none');
% L-2 NORM
function [ error ] = l_two_norm( uApprox, uExact )


   e = [];
   for i = 1:length(uApprox)
       e(end+1) = (uApprox(i) - uExact(i));
   end
   error = norm(e);


end




%MAX NORM
function [ error ] = maxNorm(uApprox,uExact)
   e = [];
   for i = 1:length(uApprox)
       e(end+1) = abs(uApprox(i) - uExact(i));
```

```
    end

    error = max(e);

end

% ERROR LOG-LOG

function  error_loglog(h,E)

figure

%

% Produce log-log plot of E vs. h.

% Estimate order of accuracy by doing a linear least squares fit.

%

% From  http://www.amath.washington.edu/~rjl/fdmbook/  (2007)


h = h(:);            % make sure it's a column vector

E = E(:);            % make sure it's a column vector

ntest = length(h);

clf

loglog(h,E,'o-')

axis([.5*min(h) 1.5*max(h)  .5*min(E) 1.5*max(E)])

title('log-log plot of errors vs. h')


% Estimate order of accuracy from least squares fit:

Ap = ones(ntest,2);
```

```
Ap(:,2) = log(h);

bp = log(E);

Kp = Ap\bp;

K = Kp(1);

p = Kp(2);

disp(' ')

disp(sprintf('Least squares fit gives E(h) = %g * h^%g',exp(K),p))

disp(' ')


% add graph of this line to loglog plot:

hold on

err1 = exp(K)*h.^p;

loglog(h,err1,'r')

legend('errors', 'least squares fit','Location','SouthEast')

hold off


end

% C-N & L-W DOUBLE POLE SOLITON

function [hVal,maxErr,normErr] = crank_nicolson_unstable_double_pole_soliton(m)

ax = -20;

bx = 20;
```

```
        c = 0.5;


h = (bx-ax)/(m+1)

x = linspace(ax,bx,m+2);

t = linspace(ax,bx,m+2);

Tfinal = 5;

k=(1/25)*h


Tsteps = (Tfinal / k);


utrue = @(y,t)  4*atan(t.*sech(y));


unot = @(y) y*0;


d_dt_unot = @(y)4*sech(y);


initTwoLevel = false;

Tgrunt = 0;

prevApprox = [];

uApprox = [];

bigM = [];

dimCount = 1;
```

```
            collectiveErrMax = [];
            collectiveErrNorm = [];
            for i=1:Tsteps
               tgrunt = Tgrunt + k;
               if(~initTwoLevel)



                  for j=2:length(x)-1
                    if(j==2)
                      uAppror(end+1) = utrue(ax,tgrunt); %using boundry condition to complete next
time level, it being n = 1
                    end
                      uApprox(end+1) =  unot(x(j)) - (k/2*h)*( unot(x(j+1)) - unot(x(j-1)) )  +
(k^2/2*h^2)*( unot(x(j+1)) - 2*unot(x(j)) + unot(x(j-1)) ) - (k^2)*sin( unot(x(j)) ); %using initial
condition to gain iner points of next time level it being n = 1
                       %uApprox(end+1) = (k/2)*(d_dt_unot(x(j-1)) +d_dt_unot(x(j+1)));
                     if(j==length(x)-1)
                      uApprox(end+1) = utrue(bx,tgrunt); %using boundry condition to complete next
time level, it being n = 1
                      end
                  end
```

```
 %{

       %CHEATING
      tCons(1:1,1:length(x)) = tgrunt;
      uApprox = utrue(x,tCons);
 %}

    tCons(1:1,1:length(x)) = tgrunt;
    figure
    plot(x,uApprox,x,utrue(x,tCons),'r')
    max_error_init_step = maxNorm(uApprox,utrue(x,tCons))
    lTwo_error_init_step = l_two_norm( uApprox, utrue(x,tCons))
     prevApprox = unot(x);
     bigM(dimCount,:) =  unot(x);
     dimCount = dimCount+1;
     initTwoLevel = true;
else

   if(initTwoLevel)
```

```
        approxGrunt = [];

        r = (k^2)/(2*(h^2));

        A = full(gallery('tridiag',m+2, -r, 2*r+1, -r));

        rhs = [];

        for l=2:length(x)-1

          if(l == 2)


              rhs(end+1) = r*(utrue(ax,tgrunt) + utrue(ax,tgrunt+k)) -  prevApprox(l) +
2*(1-r)* uApprox(l) + r*uApprox(l+1) - (k^2)*sin(uApprox(l));

          end

              rhs(end+1) = r*uApprox(l-1) -  prevApprox(l) + 2*(1-r)* uApprox(l) +
r*uApprox(l+1) - (k^2)*sin(uApprox(l));

          if(l == m-1)


              rhs(end+1) = r*(utrue(bx,tgrunt) + utrue(bx,tgrunt+k)) -  prevApprox(l) +
2*(1-r)* uApprox(l) + r*uApprox(l-1) - (k^2)*sin(uApprox(l));

          end

        end




         approxGrunt = A\rhs';
```

```
            prevApprox = uApprox;

            uApprox = approxGrunt;

        end

    end

        tCons(1:1,1:length(x)) = tgrunt;

        %disp('***********');

        t = tgrunt;

        e_max = maxNorm(uApprox,utrue(x, tCons));

        e_norm = l_two_norm(uApprox,utrue(x, tCons));

        collectiveErrMax(end + 1) = e_max;

        collectiveErrNorm(end+1) = e_norm;

        %disp('***********');


        bigM(dimCount,:) = uApprox';

        dimCount = dimCount+1;


        Tgrunt = tgrunt;


end
```

```
disp('max error on approx: ');

maxERROR_max = max(collectiveErrMax)

maxERROR_norm = max(collectiveErrNorm)

disp('*******');


hVal = h;

maxErr = maxERROR_max;

normErr = maxERROR_norm;




[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);

figure

mesh(X,Y, bigM);


end








% C-N & L-W KINK SOLITON
```

```
        function [hVal,maxErr,normErr,M] = crank_Nicolson_kink_soliton( m )

        ax = -20;

        bx = 20;


                c = 0.5;


        h = (bx-ax)/(m+1)

        x = linspace(ax,bx,m+2);

        t = linspace(ax,bx,m+2);

        Tfinal = 5;

        k=(1/25)*h

        Tsteps = (Tfinal / k);


        utrue = @(y,t) 4*atan(exp(((y-c*t)/sqrt(1-(c^2)))));


        unot = @(y)4*atan(exp(y/sqrt(1-(c^2))));

        d_dx_unot = @(y)(4*c*exp(y/sqrt( 1-(c^2) )))/(( sqrt( 1-( c^2 )
))*(1+exp(2*(y/sqrt(1-(c^2)))))); % d/dx(u(x,0))


        initTwoLevel = false;

        Tgrunt = 0;%% => time steps that the function has gone through

        prevApprox = [];
        uApprox = [];
        bigM = [];
        dimCount = 1;
        collectiveErrMax = [];
        collectiveErrNorm = [];
        for i=1:Tsteps
          tgrunt = Tgrunt + k;
          if(~initTwoLevel)


            for j=2:length(x)-1


              if(j==2)
               uApprox(end+1) = utrue(ax,tgrunt); %using boundry condition to complete next
time level, it being n = 1
              end
               uApprox(end+1) =  unot(x(j)) - (k/2*h)*( unot(x(j+1)) - unot(x(j-1)) )  +
(k^2/2*h^2)*( unot(x(j+1)) - 2*unot(x(j)) + unot(x(j-1)) ) - (k^2)*sin( unot(x(j)) ); %using initial
condition to gain iner points of next time level it being n = 1
              if(j==length(x)-1)
               uApprox(end+1) = utrue(bx,tgrunt); %using boundry condition to complete next
time level, it being n = 1
```

```
        end
      end


            tCons(1:1,1:length(x)) = tgrunt;
      figure
      plot(x,uApprox,x,utrue(x,tCons),'r')
      max_error_init_step = maxNorm(uApprox,utrue(x,tCons))
      lTwo_error_init_step = l_two_norm( uApprox, utrue(x,tCons))



   %{
            %CHEATING
       tCons(1:1,1:length(x)) = tgrunt;
       uApprox = utrue(x,tCons);
    %}
      prevApprox = unot(x);
      bigM(dimCount,:) =  unot(x);
      dimCount = dimCount+1;
      initTwoLevel = true;
   else
     %%% => TWO LEVEL SCHEME AFTER T0 HAS BEEN DEFINED
     if(initTwoLevel)
```

```
        approxGrunt = []; %currently of dim = m
        r = (k^2)/(2*(h^2));
        A = full(gallery('tridiag',m+2, -r, 2*r+1, -r));
        rhs = [];
        for l=2:length(x)-1
           if(l == 2)
              %accounts for boundey conditions
               rhs(end+1) = r*(utrue(ax,tgrunt) + utrue(ax,tgrunt+k)) -  prevApprox(l) +
2*(1-r)* uApprox(l) + r*uApprox(l+1) - (k^2)*sin(uApprox(l));
           end
               rhs(end+1) = r*uApprox(l-1) -  prevApprox(l) + 2*(1-r)* uApprox(l) +
r*uApprox(l+1) - (k^2)*sin(uApprox(l));
           if(l == m-1)
              %accounts for boundey conditions
               rhs(end+1) = r*(utrue(bx,tgrunt) + utrue(bx,tgrunt+k)) -  prevApprox(l) +
2*(1-r)* uApprox(l) + r*uApprox(l-1) - (k^2)*sin(uApprox(l));
           end
        end

        approxGrunt = A\rhs';
        prevApprox = uApprox;

        uApprox = approxGrunt;

    end

end




    tCons(1:1,1:length(x)) = tgrunt;

    %disp('***********');

    t = tgrunt;

    e_max = maxNorm(uApprox,utrue(x, tCons));

    e_norm = l_two_norm(uApprox,utrue(x, tCons));

    collectiveErrMax(end + 1) = e_max;

    collectiveErrNorm(end+1) = e_norm;

    %disp('***********');



    bigM(dimCount,:) = uApprox;

    dimCount = dimCount+1;
    Tgrunt = tgrunt;

end



disp('max error on approx: ');
maxERROR_max = max(collectiveErrMax)
maxERROR_norm = max(collectiveErrNorm)
disp('*******');

hVal = h;
maxErr = maxERROR_max;
normErr = maxERROR_norm;

[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);
figure
mesh(X,Y, bigM);



hold on

Z = 4*atan(exp(((X-c.*Y)/sqrt(1-(c^2)) )));
```

```
h = surfc(X,Y,Z);

M = bigM;

end


%C-N & L-W KINK KINK COLLISION

function [hVal,maxErr,normErr,M] = crank_Nicolson_kink_kink_collision( m )


ax = -20;

bx = 20;


        c = 0.5; % velocity of the solitary wave


h = (bx-ax)/(m+1)          % h = delta x

x = linspace(ax,bx,m+2);

t = linspace(ax,bx,m+2);

Tfinal = 5;

k=(1/25)*h

Tsteps = (Tfinal / k);

utrue = @(y,t)  4*atan( (c.*sinh(y./sqrt( 1-(c^2) ))) ./(cosh( c.*t./(sqrt(1-(c^2))) )) );

%INITIAL CONDITIONS WHICH PERTAIN TO THE EXACT SOLITON SOLUTION
ABOVE

unot = @(y) 4*atan(c.*sinh(y./(sqrt(1-(c^2)))));
```

```
        initTwoLevel = false;
        Tgrunt = 0;%% => time steps that the function has gone through
        prevApprox = [];
        uApprox = [];
        bigM = [];
        dimCount = 1;
        collectiveErrMax = [];
        collectiveErrNorm = [];
        for i=1:Tsteps
           tgrunt = Tgrunt + k;
           if(~initTwoLevel)



              for j=2:length(x)-1
                 if(j==2)
                  uApprox(end+1) = utrue(ax,tgrunt); %using boundry condition to complete next
time level, it being n = 1
                 end
                  uApprox(end+1) =  unot(x(j)) - (k/2*h)*( unot(x(j+1)) - unot(x(j-1)) )  +
(k^2/2*h^2)*( unot(x(j+1)) - 2*unot(x(j)) + unot(x(j-1)) ) - (k^2)*sin( unot(x(j)) ); %using initial
condition to gain iner points of next time level it being n = 1
                 if(j==length(x)-1)
```

```
                uApprox(end+1) = utrue(bx,tgrunt); %using boundry condition to complete next
time level, it being n = 1
               end
             end



         %{
                %CHEATING
               tCons(1:1,1:length(x)) = tgrunt;
               uApprox = utrue(x,tCons);
        %}
             tCons(1:1,1:length(x)) = tgrunt;
             figure
             plot(x,uApprox,x,utrue(x,tCons),'r')
             max_error_init_step = maxNorm(uApprox,utrue(x,tCons))
             lTwo_error_init_step = l_two_norm( uApprox, utrue(x,tCons))
              prevApprox = unot(x);
              bigM(dimCount,:) =  unot(x);
              dimCount = dimCount+1;
              initTwoLevel = true;
           else
             %%% => TWO LEVEL SCHEME AFTER T0 HAS BEEN DEFINED
             if(initTwoLevel)
               approxGrunt = []; %currently of dim = m
               r = (k^2)/(2*(h^2));
               A = full(gallery('tridiag',m+2, -r, 2*r+1, -r));
               rhs = [];
               for l=2:length(x)-1
                   if(l == 2)
                       %accounts for boundey conditions
                        rhs(end+1) = r*(utrue(ax,tgrunt) + utrue(ax,tgrunt+k)) -  prevApprox(l) +
2*(1-r)* uApprox(l) + r*uApprox(l+1) - (k^2)*sin(uApprox(l));
                   end
                        rhs(end+1) = r*uApprox(l-1) -  prevApprox(l) + 2*(1-r)* uApprox(l) +
r*uApprox(l+1) - (k^2)*sin(uApprox(l));
                   if(l == m-1)
                       %accounts for boundey conditions
                        rhs(end+1) = r*(utrue(bx,tgrunt) + utrue(bx,tgrunt+k)) -  prevApprox(l) +
2*(1-r)* uApprox(l) + r*uApprox(l-1) - (k^2)*sin(uApprox(l));
                   end
               end

               approxGrunt = A\rhs';
               prevApprox = uApprox;
               uAppox = approxGrunt;
             end
         end



             tCons(1:1,1:length(x)) = tgrunt;


             t = tgrunt;
             e_max = maxNorm(uApprox,utrue(x, tCons));
             e_norm = l_two_norm(uApprox,utrue(x, tCons));
             collectiveErrMax(end + 1) = e_max;
             collectiveErrNorm(end+1) = e_norm;


             bigM(dimCount,:) = uApprox';
             dimCount = dimCount+1;


             Tgrunt = tgrunt;


end


disp('max error on approx: ');
maxERROR_max = max(collectiveErrMax)
```

```
maxERROR_norm = max(collectiveErrNorm)

disp('*******');


hVal = h;

maxErr = maxERROR_max;

normErr = maxERROR_norm;



[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);

figure

mesh(X,Y, bigM);


M = bigM;

end




%C-N ERROR LOGING SCRIPT


hVals = [];

errValsMax = [];

errValsNorm = [];
```

```
%{
%KINK SOLITON BOX_SCHEME
for i=1:4
    m = i*100

    [h,maxErr_MAX,normErr_MAX] =  crank_Nicolson_kink_soliton(i*100);
    hVals(end+1) = h;
    errValsMax(end+1) = maxErr_MAX;
    errValsNorm(end+1) = normErr_MAX;

  pause
end
%GRAPH OF ERROR LOG  => KINK SOLITON
error_loglog(hVals,errValsMax)
pause
error_loglog(hVals,errValsNorm)

disp('exiting ...')
pause

hVals = [];
errValsMax = [];

errValsNorm = [];

%}


%{


%KINK SOLITON BOX_SCHEME

for i=1:4

   m = i*100

   [h,maxErr_MAX,normErr_MAX] = crank_Nicolson_kink_kink_collision(i*100);

   hVals(end+1) = h;

   errValsMax(end+1) = maxErr_MAX;

   errValsNorm(end+1) = normErr_MAX;

   pause


end

%GRAPH OF ERROR LOG  => KINK SOLITON

error_loglog(hVals,errValsMax)

pause

error_loglog(hVals,errValsNorm)

%}
```

```
    for i=1:4
        m = i*100
        [h,maxErr_MAX,normErr_MAX] =
crank_nicolson_unstable_double_pole_soliton(m);
        hVals(end+1) = h;
        errValsMax(end+1) = maxErr_MAX;
        errValsNorm(end+1) = normErr_MAX;

        pause
    end
    %GRAPH OF ERROR LOG  => KINK SOLITON
    error_loglog(hVals,errValsMax)
    pause
    error_loglog(hVals,errValsNorm)

    %BOX & L-W KINK SOLITON
    function [hVal,maxErr,normErr,M] = box_scheme_kink_soliton(m)

    ax = -20;
    bx = 20;
            c = 0.5;
    h = (bx-ax)/(m+1)

    x = linspace(ax,bx,m+2);

    t = linspace(ax,bx,m+2);

    Tfinal = 5;

    k=(1/25)*h

    Tsteps = (Tfinal / k);



    utrue = @(y,t) 4*atan(exp(((y-c*t)/sqrt(1-(c^2)))));

    unot = @(y)4*atan(exp(y/sqrt(1-(c^2))));

    d_dt_unot = @(y)(4.*c.*exp(y./sqrt( 1-(c^2) )))./(( sqrt( 1-( c^2 )
)).*(1+exp(2.*(y./sqrt(1-(c^2))))));

    initTwoLevel = false;

    Tgrunt = 0;

    prevApprox = [];

    uApprox = [];

    bigM = [];

    dimCount = 1;

    collectiveErrMax = [];

    collectiveErrNorm = [];

    for i=1:Tsteps

        tgrunt = Tgrunt + k;

        if(~initTwoLevel)
        for j=2:length(x)-1

          if(j==2)
           uAppox(end+1) = utrue(ax,tgrunt);
          end
           uApprox(end+1) =  unot(x(j)) - (k/2*h)*( unot(x(j+1)) - unot(x(j-1)) )  +
(k^2/2*h^2)*( unot(x(j+1)) - 2*unot(x(j)) + unot(x(j-1)) ) - (k^2)*sin( unot(x(j)) );
          if(j==length(x)-1)
           uApprox(end+1) = utrue(bx,tgrunt);
          end
        end


      %{

         %CHEATING

          tCons(1:1,1:length(x)) = tgrunt;
          uApprox = utrue(x,tCons);

        %}
```

```
         tCons(1:1,1:length(x)) = tgrunt;
        figure
        plot(x,uApprox,x,utrue(x,tCons),'r')
        max_error_init_step = maxNorm(uApprox,utrue(x,tCons))
        lTwo_error_init_step = l_two_norm( uApprox, utrue(x,tCons))

       prevApprox = unot(x);
       bigM(dimCount,:) =  unot(x);
       dimCount = dimCount+1;
       initTwoLevel = true;
     else

       if(initTwoLevel)

           approxGrunt = [];
           for j=2:length(x)-1
            if(j==2)
                approxGrunt(end+1) = utrue(ax,tgrunt);
            end
                approxGrunt(end+1) = 2*uApprox(j) - prevApprox(j) +
((k^2)/(h^2))*(uApprox(j-1) - 2*uApprox(j) + uApprox(j+1)) - (k^2)*sin(uApprox(j));
```

```
         if(j==length(x)-1)

              approxGrunt(end+1) = utrue(bx,tgrunt);

         end


       end


       prevApprox = uApprox;

       uApprox = approxGrunt;

   end

end




tCons(1:1,1:length(x)) = tgrunt;

    tCons(1:1,1:length(x)) = tgrunt;

   % disp('***********');

    t = tgrunt;

    e_max = maxNorm(uApprox,utrue(x, tCons));

    e_norm = l_two_norm(uApprox,utrue(x, tCons));

    collectiveErrMax(end + 1) = e_max;

    collectiveErrNorm(end+1) = e_norm;

    %disp('***********');
```

```
        bigM(dimCount,:) = uApprox;

        dimCount = dimCount+1;



    Tgrunt = tgrunt;


end


disp('max error on approx: ');

maxERROR_max = max(collectiveErrMax)

maxERROR_norm = max(collectiveErrNorm)

disp('*******');


hVal = h;

maxErr = maxERROR_max;

normErr = maxERROR_norm;


[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);

figure

mesh(X,Y, bigM);
```

```
M = bigM;

end


%BOX & L-W KINK KINK COLLISION

function [hVal,maxErr,normErr,M] = box_Scheme_kink_kink_collision(m)




ax = -20;

bx = 20;

          c = 0.5;


h = (bx-ax)/(m+1)

x = linspace(ax,bx,m+2);

t = linspace(ax,bx,m+2);

Tfinal = 5;

k=(1/25)*h

Tsteps = (Tfinal / k);


utrue = @(y,t)  4*atan( (c.*sinh(y./sqrt( 1-(c^2) ))) ./(cosh( c.*t./(sqrt(1-(c^2))) )) );


unot = @(y) 4*atan(c.*sinh(y./(sqrt(1-(c^2)))));
```

```
initTwoLevel = false;
Tgrunt = 0;
prevApprox = [];
uApprox = [];
bigM = [];
dimCount = 1;
collectiveErrMax = [];
collectiveErrNorm = [];
for i=1:Tsteps
   tgrunt = Tgrunt + k;
   if(~initTwoLevel)


      for j=2:length(x)-1


         if(j==2)
          uApprox(end+1) = utrue(ax,tgrunt);
         end
          uApprox(end+1) =  unot(x(j)) - (k/2*h)*( unot(x(j+1)) - unot(x(j-1)) )  +
(k^2/2*h^2)*( unot(x(j+1)) - 2*unot(x(j)) + unot(x(j-1)) ) - (k^2)*sin( unot(x(j)) ); %using initial
condition to gain iner points of next time level it being n = 1
         if(j==length(x)-1)
```

```
     uApprox(end+1) = utrue(bx,tgrunt);
    end
 end




%{


     %CHEATING
   tCons(1:1,1:length(x)) = tgrunt;
   uApprox = utrue(x,tCons);


%}


 tCons(1:1,1:length(x)) = tgrunt;
 figure
 plot(x,uApprox,x,utrue(x,tCons),'r')
 max_error_init_step = maxNorm(uApprox,utrue(x,tCons))
 lTwo_error_init_step = l_two_norm( uApprox, utrue(x,tCons))
 prevApprox = unot(x);
bigM(dimCount,:) =  unot(x);
dimCount = dimCount+1;
```

```
            initTwoLevel = true;
          else
            if(initTwoLevel)


              approxGrunt = [];
              for j=2:length(x)-1
               if(j==2)
                  approxGrunt(end+1) = utrue(ax,tgrunt);
               end
                  approxGrunt(end+1) = 2*uApprox(j) - prevApprox(j) +
((k^2)/(h^2))*(uApprox(j-1) - 2*uApprox(j) + uApprox(j+1)) - (k^2)*sin(uApprox(j));


                if(j==length(x)-1)
                    approxGrunt(end+1) = utrue(bx,tgrunt);
                end


              end


              prevApprox = uApprox;
              uApprox = approxGrunt;
            end
          end
```

```
    tCons(1:1,1:length(x)) = tgrunt;

    hold on;


        tCons(1:1,1:length(x)) = tgrunt;

        %disp('***********');

        t = tgrunt;

        e_max = maxNorm(uApprox,utrue(x, tCons));

        e_norm = l_two_norm(uApprox,utrue(x, tCons));

        collectiveErrMax(end + 1) = e_max;

        collectiveErrNorm(end+1) = e_norm;

       % disp('***********');


        bigM(dimCount,:) = uApprox;

        dimCount = dimCount+1;




    Tgrunt = tgrunt;


end


disp('max error on approx: ');
```

```
maxERROR_max = max(collectiveErrMax)

maxERROR_norm = max(collectiveErrNorm)

disp('*******');


hVal = h;

maxErr = maxERROR_max;

normErr = maxERROR_norm;


[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);

figure


mesh(X,Y, bigM);




M = bigM;

end


%BOX & L-W DOUBLE POLE SOLITON

function [hVal,maxErr,normErr] = box_scheme_double_pole_soliton( m )


ax = -20;

bx = 20;
```

```
        c = 0.5;


h = (bx-ax)/(m+1)

x = linspace(ax,bx,m+2);

t = linspace(ax,bx,m+2);

Tfinal = 5;

k=(1/25)*h


Tsteps = (Tfinal / k);


utrue = @(y,t)  4*atan(t.*sech(y));

unot = @(y) y*0;

d_dx_unot = @(y) 4*sech(y);


initTwoLevel = false;

Tgrunt = 0;

prevApprox = [];

uApprox = [];

bigM = [];

dimCount = 1;

collectiveErrMax = [];
```

```
collectiveErrNorm = [];
for i=1:Tsteps
    tgrunt = Tgrunt + k;
    if(~initTwoLevel)


        for j=2:length(x)-1


           if(j==2)
            uApprox(end+1) = utrue(ax,tgrunt);
           end
            uApprox(end+1) =  unot(x(j)) - (k/2*h)*( unot(x(j+1)) - unot(x(j-1)) )  +
(k^2/2*h^2)*( unot(x(j+1)) - 2*unot(x(j)) + unot(x(j-1)) ) - (k^2)*sin( unot(x(j)) ); %using initial
condition to gain iner points of next time level it being n = 1
           if(j==length(x)-1)
            uApprox(end+1) = utrue(bx,tgrunt);
           end
        end



         %{
```

```
       %CHEATING
      tCons(1:1,1:length(x)) = tgrunt;
      uApprox = utrue(x,tCons);
      %}

     tCons(1:1,1:length(x)) = tgrunt;
    figure
    plot(x,uApprox,x,utrue(x,tCons),'r')
    pause
    max_error_init_step = maxNorm(uApprox,utrue(x,tCons))
    lTwo_error_init_step = l_two_norm( uApprox, utrue(x,tCons))
    prevApprox = unot(x);
   bigM(dimCount,:) =  unot(x);
   dimCount = dimCount+1;
    initTwoLevel = true;
else
   %%% => TWO LEVEL SCHEME AFTER T0 HAS BEEN DEFINED
   if(initTwoLevel)

      approxGrunt = [];
      for j=2:length(x)-1
       if(j==2)
```

```
                approxGrunt(end+1) = utrue(ax,tgrunt); %using boundry condition to complete
next time level, it being n = 1
            end
                approxGrunt(end+1) = 2*uApprox(j) - prevApprox(j) +
((k^2)/(h^2))*(uApprox(j-1) - 2*uApprox(j) + uApprox(j+1)) - (k^2)*sin(uApprox(j));

            if(j==length(x)-1)
                approxGrunt(end+1) = utrue(bx,tgrunt);
            end

          end

          prevApprox = uApprox;
          uApprox = approxGrunt;
      end
  end

  tCons(1:1,1:length(x)) = tgrunt;
  hold on;

     tCons(1:1,1:length(x)) = tgrunt;
     %disp('***********');
```

```
        t = tgrunt;

        e_max = maxNorm(uApprox,utrue(x, tCons));

        e_norm = l_two_norm(uApprox,utrue(x, tCons));

        collectiveErrMax(end + 1) = e_max;

        collectiveErrNorm(end+1) = e_norm;

        %disp('***********');


        bigM(dimCount,:) = uApprox;

        dimCount = dimCount+1;


    Tgrunt = tgrunt;


end


disp('max error on approx: ');

maxERROR_max = max(collectiveErrMax)

maxERROR_norm = max(collectiveErrNorm)

disp('*******');



hVal = h;

maxErr = maxERROR_max;
```

```
normErr = maxERROR_norm;



[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);

mesh(X,Y, bigM);


end

%CONSTANT OF MOTION ERROR ANALYSIS CODE

% H = (1/2)( ( u_{x}^{2} ) + ( u_{t}^{2} ) + 2*(1 - cos(u)) )

% E = d/dt((1/2)*( u_{t}^{2} ) + (1/2)*( u_{t}^{2} ) - cos(u)) - d/dx(

% (u_{t})(u_{x}))

ax = -20;

bx = 20;            % heat conduction coefficient:

            % final time

c = 0.5;

Tfinal = 5;

hVal = [];

errH_vals = [];

errE_vals = [];

for i=1:4

    m = i*100

    h = (bx-ax)/(m+1);
```

```
hVal(end+1) = h;


x = linspace(ax,bx,m+2);  % note x(1)=0 and x(m+2)=1

t = linspace(ax,bx,m+2);

k=(1/25)*h;

[X,Y] = meshgrid(ax:h:bx,0:k:Tfinal);

U = 4*atan(exp(((X-c.*Y)/sqrt(1-(c^2)) )));

%FUNCTION CAN BE SWITCHED OUT FOR ANY OF THE SCHEME TO
SOLUTION

%FUNCTIONS FOR EXAMPLE: crank_Nicolson_kink_kink_collision( m )

[hb mE lE M] = box_scheme_kink_soliton(m);


[r y] = size(M)




H_ERR = [];

E_ERR = [];

for j = 2:r-1

  % ***** TIME STEP SOLUTION LOOP *******

  % ***** gets solution corresponding to the jth time step

  % per each time step t solution and approximation

  % we run them through
  % wor done on U and M
  timeSliceU = U(j,1:y);
  timeSliceM = M(j,1:y);


  discHamM = [];
  discEnM = [];


  discHamU = [];
  discEnU = [];
  for l = 2:length(timeSliceU)-1
      %approx solution
      discHamM(end+1) = (1/2)*( (((1/h)*(timeSliceM(l+1) - timeSliceM(l)))^2) + ((
(1/k)*(M(j+1,l) - timeSliceM(l)) )^2) + 2*(1-cos(timeSliceM(l))) );
      discEnM(end+1) = (1/k)*(M(j+1,l) - timeSliceM(l))*( (1/(k^2))*(M(j+1,l) -
2*timeSliceM(l) + M(j-1,l)) + sin(timeSliceM(l)) - (1/(h^2))*( timeSliceM(l-1) -
2*(timeSliceM(l)) + timeSliceM(l+1) ));
      %exact olution
      discHamU(end+1) = (1/2)*( (((1/h)*(timeSliceU(l+1) - timeSliceU(l)))^2) + ((
(1/k)*(U(j+1,l) - timeSliceU(l)) )^2) + 2*(1-cos(timeSliceU(l))) );
      discEnU(end+1) = (1/k)*(U(j+1,l) - timeSliceU(l))*( (1/(k^2))*(U(j+1,l) -
2*timeSliceU(l) + U(j-1,l)) + sin(timeSliceU(l)) - (1/(h^2))*( timeSliceU(l-1) - 2*(timeSliceU(l))
+ timeSliceU(l+1) ));
```

```
        end



        hamM_C = trapz(x(2:length(x)-1), discHamM);

        enM_C = trapz(x(2:length(x)-1), discEnM);

         hamU_C = trapz(x(2:length(x)-1), discHamU);

         enU_C = trapz(x(2:length(x)-1),  discEnU);

         ham_Err = abs(hamM_C - hamU_C);
         en_Err = abs(enM_C  - enU_C);

         H_ERR(end+1) = ham_Err;
         E_ERR(end+1) = en_Err;
    end
    errH_vals(end+1) = max(H_ERR);
    errE_vals(end+1) = max(E_ERR);

end
```

```
size(hVal)
size(errH_vals)
size(errE_vals)


figure
error_loglog(hVal,errH_vals)
pause
figure
error_loglog(hVal,errE_vals)
```

Works Cited

[1] LeVeque, Randall J. "Finite Difference Methods for Partial Differential Equations." (2006): n. pag. 4 July 2007. Web. 15 Apr. 2016. <http://simulation.mosharsystem.com/Downloads/Finite%20Difference%20Methods.pdf>.

[2] Mohebbi, Akbar, and Mehdi Dehghan. "High-order Solution of One-dimensional

Sine–Gordon Equation Using Compact Finite Difference and DIRKN Methods." *Science Direct*. Elsevier, Mar. 2010. Web. 23 Apr. 2016.

[3] Schober, C. M., and T. H. Wlodarczyk. "Dispersive properties of multisymplectic integrators." *Journal of Computational Physics* 10.227 (2008): 5090-5104.

[4] Scott, Alwyn. *Encyclopedia of Nonlinear Science*. New York: Routledge, 2005. Print.

[5] Winter, Hugo. "Numerical Advection Schemes in Two Dimensions." (n.d.): n. pag. *Lancaster University*. 26 Apr. 2011. Web. 20 Apr. 2016.