M.Tech. (Computer Science) Dissertation series

SOLITON SOLUTIONS OF BURGERS AND KdV EQUATIONS -A NUMERICAL APPROACH

A dissertation submited in partial fulfilment of the requirement for the M. Tech. (Computer Science) of the Indian Statistical Institute

by
Subir Kumar Bandyopadhyay

under the supervision of
Professor Rajkumar Roychoudhury

INDIAN STATISTICAL INSTITUTE
203, Barrackpore Trunk Road
Calcutta-700 035

July 1996

INDIAN STATISTICAL INSTITUTE
203, B.T. Road
Calcutta - 700 035, INDIA

Certificate of Approval

This is to certify that the thesis titled **SOLITON SOLUTIONS OF BURGERS AND KdV EQUATIONS-A NUMERICAL APPROACH** submitted by **Subir Kumar Bandyopadhyay**, towards partial fulfillment of the requirements for the degree of M.Tech. in Computer Science at the Indian Statistical Institute, Calcutta, embodies the work done under my supervision.

Rajkumar Roychoudhury 19.7.95

Professor Rajkumar Roychoudoury
Physics & Applied Mathematics Unit
Indian Statistical Institute
Calcutta-700 035.

Professor and Head,
PHYSICS AND APPLIED MATHEMATICS UNIT,
INDIAN STATISTICAL INSTITUTE,
Calcutta—700038

Acknowledgements

I am deeply grateful to Professor Rajkumar Roychoudhury for his guidance, advice and providing constant encouragement in this project, without which this could not have been possible. I also thank Professor S.K. Pal for helping me in preparing this report. I also thank Sri Subir Kumar Mukherjee and Sri Swapan De for various helps.

Lastly, I should take this opportunity to thank all my classmates.

S. K. Banerjee

(SUBIR KUMAR BANDYOPADHYAY)

Calcutta
July, 1996.

Contents

Chapter 1

Introduction

The Burgers equation and the Korteweg de Vries equation popularly called the KdV equation are two of the most well known equations in the study of solitary waves. The Burgers equation which may be considered as nonlinear diffusion equation reads as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

This equation was first formulated by Burgers (1948) in an attempt to model turbulent flow in a channel. The KdV equation was founded by Korteweg and de Vries (1895) in their study of two dimensional motion of weakly nonlinear long waves . This equation is widely applicable. Benney (1966) applied it to inertial waves in a rotating fluid and to internal gravity waves in a stratified fluid.

Two other important applications are in ion-acoustic waves in a plasma (Washimi and Taniuti, 1966) and to pressure waves in a mixture of gas bubbles liquid (Wijngaarden, 1968). The KdV equation is in its original form, reads as

$$\frac{\partial u}{\partial t} = \frac{3}{2}\sqrt[2]{\frac{g}{h}}(u\frac{\partial u}{\partial x} + \frac{2}{3}\alpha\frac{\partial u}{\partial x} + \frac{\sigma}{3}\frac{\partial^3 u}{\partial x^3})$$

where α is a constant and $\sigma = \frac{1}{3}h^3 - \frac{Th}{g\rho}$ where T being the surface tension, s being the density of the liquid, h the height above mean level and g accleration due to gravity.

This equation initially described the solitary wave, first observed by J.Scott Russel in 1834 on the Edinburgh to Glasgow canal. Since the discovery of the scattering method by Gardenar, Green, Kruskal and Miura (1967) tremendous interest was shown in KdV equation as it yields soliton solutions which are related to the scattering problem of Sturm-Liouville type, which in quantum theory is known as the Schrödinger equation. In this respect we shall employ numerical techniques to study the Shock wave and Solitary wave solutions for the Burgers and KdV equations.

In real life however the exact Burgers equation and exact KdV equation may not describe the actual situation. Hence perturbed form of these equations are of interest in both mathematical studies and applications (Karpman and Maslov (1978)). However here we shall limit ourselves to a perturbation of the form ϵu, ϵ being a small parameter.This will also help in understanding the stability of the equation.

Our main focus will be on KdV equation and its soliton solutions. The plan of this report is as follows.

In chapter 1, a brief history of the Burgers equation and the KdV equation is given.
In chapter 2, we discuss the basic concepts about solitary waves.
In chapter 3, we discuss some mathematical properties of the KdV equation and Burgers equations.
In chapter 4, a brief review of the numerical techniques involved in solving differential equation is given with emphasis on Crank Nicolson method, which was employed in this work.
In chapter 5, the algorithm for solving the Burgers and KdV equation are given. The results are also included in form of tables.
Finally chapter 6 is kept for discussion.
In appendix, we have included the original code of the program, which is written in High level language C.

Chapter 2

Basic Concepts

2.1 Solitary waves

Solitary waves are waves, which never crest nor dissipate, they propagate without changing shape. Solitons are solitary waves with the remarkable feature that when two or more of them collide, they do not scatter but emerge with same shape and velocity. Zabusky and Kruskal (1965) coined the term soliton to indicate this remarkable feature i.e a soliton is a localized nonlinear wave of permanent form which may interact strongly with other solitons, so that when they separate after the interaction they regain thier original shapes. The solitons are also known as Non-dissipative waves. To compare them , with the ordinary dissipative waves one has only to consider the common example of dropping a stone into a small pond. As soon as the stone is dropped a dissipated wave is formed which gradually dies away from the source. For dissipative wave , energy gradually decreases with space coordinates and the shape of the wave does not remain constant. But solitary waves do not die out from the source as the distance increases which proves non-dissipative characteristics. Their shapes remain the same.

Solitons are formed in a medium, when the effects of dispersion and nonlinearity are balanced. In the absence of nonlinearity, dispersion can destroy a solitary wave as the various components of the wave propagate at different velocities. Introducing nonlinearity

without dispersion again rules out the possibility of solitary waves because the pulse energy is continuously injected into high freqency mode. But with both dispersion and nonlinearity present formation of soliton is possible.

As mentioned before in 1845 John Scott Russel, a British naval architect, first saw a single isolated solitary wave on the water surface of a channel. By studying the nature of waves, he claimed that the propagation of isolated wave was a consequence of the property of the medium rather than the circumstances of the wave generation. Since then it took rather a long time to establish that some spatial nonlinear wave equations admit solutions consisting of isolated waves that can propagate and undergo collisions without loosing their respect identities. **Korteweg and de Vries (KdV)** explained in 1895 the phenomena for the propagation of waves in a shallow water. Later on numerical simulations on wave propagation (Perring et al. (1968), agreed with the Scott-Russel's observation that the existance of wave like excitation which rather than disperse their energy, maintain a stable shape. Zabusky and Kruskal (1965) made a computer study of KdV equation. They observed that a single solitary wave behaves like a particle in their interaction with each other. They also observed that under certain conditions any initial pulse can break up into a number of solitons which can move in a plasma with different phase velocities. Also the solitons interact with each other and after the interaction they emerge out without any change in their shape or velocity. To study soliton in a multicomponent plasma is also important. In the laboratory frame work and in ionosphere, plasma is a collection of several electron and ion species. Some important nonlinear equations which give rise to solitary waves in plasma are KdV equation, Modified KdV equation, nonlinear Schrödinger equations.

2.2 Solitary waves in Plasma

A plasma contains a wide variety of waves because of its fluid like behaviour and also because of its long range interaction between particles in it. It is well known that plasma

is a dispersive media. Again from the study of plasma oscillation it is obvious that plasma can propagate in a dispersive medium. So in a plasma medium plasma particles and wave can coexist and they can interact with each other and oscillation of plasma can occur. In a plasma two types of oscillations are theoretically possible, first oscillations of electron which are too rapid for the ions to follow and second the oscillations of ion which are so slow that the electrons continuously satisfy the Boltzmann law. It was Langmuir (1928) who first discussed the electron oscillation in plasma. In plasma, electron oscillations arise due to the property of plasma to try to remain neutral. When electrons are displaced in a uniform background of ions, the electric field will build up such as to restore the neutrality of plasma by putting the electrons back to their original position. Because of their inertia the electrons will overshoot and oscillate around their equilibrium position with a charecteristic freqency namely plasma freqency.

2.3 Mathematical Representation of Soliton

First mathematical representation of soliton was given by D.J.Korteweg and H.D.Vries in a paper published in Philosophical magazine in 1895. They showed that solitons can exist due to nonlinear wave motion in hydrodynamics. The KdV equation is of the form

$$\frac{\partial u(x,t)}{\partial t} + (1 + u(x,t))\frac{\partial u(x,t)}{\partial x} + \frac{\partial^3 u(x,t)}{\partial x^3} = 0 \tag{2.1}$$

If we linearize the above equation, we get

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0 \tag{2.2}$$

For this linearized equation any solution can be represented as a superposition of Fourier components. So the solution is of the form

$$u = c\exp(i(kx - \omega t)) \tag{2.3}$$

It follows that

$$\omega = k - k^3 \tag{2.4}$$

This is the dispersion relation which gives the freqency ω as a function of the wave number k. From it we deduce the phase velocity v_p as

$$v_p = \frac{\omega}{k} = 1 - k^2 \tag{2.5}$$

which gives the velocity of the wave fronts of the sinusoidal mode. we also deduce the group velocity,

$$v_g = \frac{d\omega}{dk} = 1 - 3k^2 \tag{2.6}$$

which gives the velocity of the wave packet i.e a group of waves with nearly the same length $2\pi/k$. From the expression of v_p and v_g we can easily conclude $v_g \leq v_p \leq 1$ and $v_g = v_p = 1$ for long waves(i.e for k=0). A short wave has a negative phase velocity c. Packets of waves of nearly the same length propagate with the group velocity, individual component moving through the packet with their phase velocity. It can in general be shown that the energy of a wave disturbance is propagated at the group velocity, not the phase velocity. Long-wave components of a general solution travel faster than the short wave components, and thereby the components disperse. Thus the linear theory predicts the dispersal of any disturbance other than a purely sinusoidal one. Looking at equation (4), we see that the effect of dispersion comes from the term k^3 in the expression of ω and from the term $\frac{\partial^3 u}{\partial x^3}$ in the equation (1).

In contrast to dispersion, nonlinearity leads to concentration of a disturbance. To observe this effect , neglect the term $\frac{\partial^3 u}{\partial x^3}$ in the KdV equn. above and retain the nonlinear term. Then we have

$$\frac{\partial u}{\partial t} + (1+u)\frac{\partial u}{\partial x} = 0 \tag{2.7}$$

The method of charecteristics may be used to show that this equation has the elementary solution

$$u = f(x - (1 + u)t) \tag{2.8}$$

for any differentiable function f. This shows that disturbances travel at the characteristic velocity $(1 + u)$. Thus the highers parts of the solution travel faster than the lower parts. This catching up tends to steepen a disturbance untill it breaks and a discontinuity or Shock wave is formed.

For a solitary wave, the dispersive effects for the term $\frac{\partial^3 u}{\partial x^3}$ and the concentrating effects of the term $u\frac{\partial u}{\partial x}$ are just in balance.

From the above discussion we can conclude that soliton is a solution of a nonlinear equation or system which

(i) represents a wave of permanent form.

(ii) is localized, decaying or becoming constant at infinity.

(iii) may interact strongly with other soiltons so that after the interaction it remains its shape form, almost as if the principle of superposition valid.

From now one we shall take KdV equation in the standard form

$$\frac{\partial u}{\partial t} - 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0 \tag{2.9}$$

Chapter 3

Properties Of Burgers and Korteweg de Vries equation

3.1 Burgers equation

The standerd form of the Burger equation is

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

This equation is almost similar to the KdV equation, where instead of u_{xx} we have u_{xxx}. However Burgers equation differs significantly from the KdV equation in its properties. Unlike the KdV equation the Burgers equation has no soliton solution. Instead it has shock wave type solution viz,

$$u(x,t) = c[1 - tanh\frac{(x-ct)}{2\nu}]$$

By the transformation $v_x = \frac{uv}{2\nu}$ (Bäcklund Transformation), Burgers equation becomes the diffusion equation

$$\frac{\partial v}{\partial t} = \nu\frac{\partial^2 v}{\partial x^2}$$

3.2 KdV equation

(i) KdV equation is invarient under Galilean transformation

The transformation

$$x' = x + u_0 t \quad t' = t \quad and \quad u' = u + u_0$$

where u_0 is a constant, reproduces the original KdV equation.

(ii) If $u(x,t)$ is a solution of the KdV equation

$$\frac{\partial u}{\partial t} - 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0$$

then u(-x, -t) is also a solution of the above equation.

(iii) **Solitary wave solution** :

The solitary wave solution of (one soliton solution) KdV equation is given as

$$u(x,t) = c_1 sech^2(c_2(x - ct))$$

where $c_1 > 0$ and $c_2 > 0$ are constants.

(iv) **Conservation laws** :

Suppose T and X are function of u, the spatial derivatives of u, x and t, so that

$$\frac{\partial T}{\partial t} + \frac{\partial X}{\partial x} = 0$$

when u(x, t) is a solution of some KdV equation. Then the above eqation is called a conservation relation with density T and flux X. If T and X are integrable for $-\infty < x < \infty$ then

$$\frac{d}{dt}\int_{-\infty}^{+\infty} T dx = 0$$

so,

$$\int_{-\infty}^{+\infty} T dx = constant$$

these idea can be applied to KdV equation as.

$$\frac{\partial u}{\partial t} - 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0$$

or,

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(-3u^2 + u_{xxx}) = 0$$

If

$$T = u$$

and

$$X = u_{xx} - 3u^2$$

then $\int_{x=-\infty}^{x=+\infty} u dx$ = constant.

Similarly it can be shown that,

$\int_{-\infty}^{+\infty} u^2\, dx$=constant

3.2.1 Initial value problem for the Korteweg de Vries equation

Now we shall discuss the remarkable role of solitons the initial-value problem for the KdV equation over the interval $(-\infty, +\infty)$. Let u(x, t) be a solution of the KdV equation over the interval $-\infty < x < +\infty$ and for $t > 0$ where

$$u(x, 0) = g(x)$$

for a given function g. Here we discuss the solution u(x, t) for $sech^2 x$ potentials i.e one seeks the solution for the initial value function

$$g(x) = -v sech^2 x$$

for various values of v.

It is found (Drazin 1983),

If $v < 0$, There is no solitary-wave component of the solution as t tends to ∞. Instead there is one set of decaying dispersive wave train.

If $v = 0$, then of course the solution is $u(x,t) = 0$ for all t and x.

If $v > 0$, then from Scattering theory [where $u(x,t)$ behaves as potential wall], it is found, if v is of the form $v = n(n+1)$ where $n = 1, 2, 3, \cdots$,then there are n of solitary waves and no dispersive wave.

For example, If $v = 2 = 1(1+1)$ then $g(x) = -2sech^2x$ there is single solitary wave and no dispersive wave train. If $v = 6 = 2(2+1)$ then there arise just two solitary waves and no dispersive wave train.

so if $v = n(n+1)$ then there exist n of solitary waves, corresponding to n bound states with eigen states $-n^2, -(n-1)^2, -(n-2)^2, \cdots, -1$

If $n(n-1) < v < n(n+1)$ then there arise n number of solitary waves with a dispersive wave in the limit as t tends to ∞.

If $2 < v < 6$ i.e $2(2-1) < v < 2(2+1)$ then there arise two solitary waves with a dispersive wave. Let $v = 4$ then $g(x) = u(x,0) = -4sech^2x$ then there arise two solitary waves and one dispersive wave.

If $6 < v < 12$ i.e $3(3-1) < v < 3(3+1)$ then there arise three solitary waves together with a dispersive wave train and so on.

In the present work, we have considered the one and two soliton solutions i.e $v = 2\&6$.

Chapter 4

Numerical Approach

4.1 NUMERICAL TECHNIQUES FOR SOLVING THE PARTIAL DIFFERENTIAL EQUATION

Partial differential equations occur in many branches of applied mathematics, for example, in hydrodyanamics, elasticity, quantum mechanics, electro-magnetic theory. The analytical treatment of these equations is a rather involved process and requires applications of advanced mathematical methods. On the other hand, it is generally easier to produce sufficiently approximate solutions by simple and efficient numerical methods. Several numerical methods have been proposed for the solutions of partial differential equation, but the finite difference methods have become popular and are more gainfully employed than others. So we will solve the nonlinear partial differential equations by finite difference methods.

Partial differential equations are classified into two types : Linear and Nonlinear. Our field of interest is in the nonlinear partial differential equations. At first we shall discuss in brief, the linear partial differential equation, then I shall discuss the nonlinear theory.

The general second order linear partial differential equation is of the form

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G \tag{4.1}$$

where A, B, C, D, E, F, G are all functions of x and y. and

$$\frac{\partial^2 u}{\partial x^2} = u_{xx}$$

$$\frac{\partial^2 u}{\partial x \partial y} = u_{xy}$$

$$\frac{\partial u}{\partial x} = u_x$$

etc.

Now consider equation number (4.1), I can be classified with respect to the sign in the discriminent $\triangle_s = B^2 - 4AC$ in the following way,

If $\triangle_s < 0$ at a point in the (x, y) plane, the equation is said to be Elliptic.

If $\triangle_s > 0$ at a point in the (x, y) plane, the equation is said to be Hyperbolic.

If $\triangle_s = 0$ at a point in the (x, y) plane, the equation is said to be Parabolic.

Parabolic and Hyperbolic problems are classified as Initial value problems whereas the elliptic problems are classified as Boundary value problem.

Initial value problem can further be classified into 2 types as Pure Initial Value problem and Initial Boundary Value problem.

In nonlinear equation, the coefficients A, B, C, D, E, F and G are not only the function of x, y but also function of u. We will now discuss the details of the special form of nonlinear parabolic partial differential equation, but before that we will discuss the finite difference methods. We will assume that a steady state solution does not exist and one of the indepnedent variable y is replaced by t, has role of time. We solve the differential equation in an arbitrary region R$\times[0, T]$ with suitable initial and boundary conditions where $R = a \leq x \leq b$ and $0 \leq t \leq T$. We superimpose on $R\times[0, T]$ a rectangular grid with grid lines parallel to co-ordinate axes with spacing h and k in space and time directions respectively. The grid points on $R\times[0, T]$ are given by

$$t_n = nk, n = 0, 1, 2, 3, \cdots, N$$

$$x_m = a + mh, m = 0, 1, 2, 3, \cdots, M$$

where $x_0 = a, x_M = b$ and $M = \frac{(b-a)}{h}$ $N = \frac{T}{k}$ The special nodes on n-th time grid constitute the n-th layer or level. Let the solution value $u(x_m, t_n)$ be denoted as u_m^n. Then we have,

$$u_x = \frac{u_{m+1}^n - u_m^n}{h} + O(h)$$

$$u_x = \frac{u_m^n - u_{m-1}^n}{h} + O(h)$$

$$u_x = \frac{u_{m+1}^n - u_{m-1}^n}{2h} + O(h^2)$$

so,

$$u_{xx} = \frac{u_{m+1}^n - 2u_m^h + u_{m-1}^n}{h^2} + O(h^2)$$

Similarly,

$$u_{xxx} = \frac{u_{m+2}^n - 3u_{m+1}^n + 3u_m^n - u_{m-1}^n}{h^3} + O(h^3)$$

etc. In similar fashion,

$$u_t = \frac{u_m^{n+1} - u_m^n}{k} + O(k)$$

There are various methods of solving Partial differential equation, most effectively used method is Crank-Nicolson method. In 1947, Crank and Nicolson proposed a method according to which $\frac{\partial^2 u}{\partial x^2}$ is replaced by the average of its Finite Difference Approximations on the n-th and $(n+1)$-th time row. So,

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{m+1}^n - 2u_m^n + u_{m-1}^n}{2h^2} + \frac{u_{m+1}^{n+1} - 2u_m^{n+1} + u_{m-1}^{n+1}}{2h^2}$$

So, in Crank-Nicolson method the Heat Conduction equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

can bewritten as

$$\frac{u_m^{n+1} - u_m^n}{k} = \frac{u_{m+1}^n - 2u_m^n + u_{m-1}^n}{2h^2} + \frac{u_{m+1}^{n+1} - 2u_m^{n+1} + u_{m-1}^{n+1}}{2h^2}$$

which gives on rearranging

$$-\lambda u_{m-1}^{n+1} + (2+2\lambda)u_m^{n+1} - \lambda u_{m+1}^{n+1} = \lambda u_{m-1}^n + (2-2\lambda)u_m^n + \lambda u_{m+1}^n$$

where $\lambda = \frac{k}{h^2}$ is called mesh ratio parameter The left side of the above equation has 3 unknowns and on the right side all the 3 quantities are known, this implicit scheme is called Crank Nicolson formula and is convergent for value of $\lambda \leq \frac{1}{2}$. If there are N internal mesh points on each row, then above formula gives N simultaneous equations for N unknowns in terms of the given boundary boundary values. Similarly internal mesh points on all rows can be calculated. Our main task in the project is to solve the Burgers equation and KdV equations (in absence and presence of perturbation) numerically by finite difference method. At first, we shall describe general form of second order nonlinear equation of the form

$$pu_{xx} = u_t + f(x, t, u, u_x) \tag{4.2}$$

where 'p' is a positive constant and $a \leq x \leq b$ and boundary conditions are $u(a,t) = g_1(t)$, $u(b,t) = g_2(t)$ A general two level finite difference approximations to equation number (11) is given by,(at (x_m, t_n)) is

$$p\frac{\Delta_x^2 u_m^{n\prime}}{h^2} = \frac{u_m^{n+1} - u_m^n}{k} + f_m^{n\prime}$$

where

$$f_m^{n\prime} = f(x_m, t_n', u_m^{n\prime}, \Delta_{2x}\frac{u_m^{n\prime}}{2h})$$

$$t_n' = \theta t_{n+1} + (1-\theta)t_n$$

for $0 \leq \theta \leq 1$

$$u_{m+q}^{n\prime} = \theta u_{m+q}^{n+1} + (1-\theta)u_{m+q}^n$$

where $q = -1, 0, +1$

$$\triangle_{2x} u_m^{n\prime} = u_{m+1}^{n\prime} - u_{m-1}^{n\prime} = [\theta u_{m+1}^{n+1} + (1-\theta)u_{m+1}^{n}] - [\theta u_{m-1}^{n+1} + (1-\theta)u_{m-1}^{n}]$$

For $\theta = 1$,

$t_n' = t_{n+1} + (1-1)t_n = t_{n+1}$

$u_{m+q}^{n\prime} = u_{m+q}^{n+1}$

So for $q = 0, u_m^{n\prime} = u_m^{n+1}$

for $q = -1, u_{m-1}^{n\prime} = u_{m-1}^{n+1}$

for $q = +1, u_{m+1}^{n\prime} = u_{m+1}^{n+1}$

$\triangle_{2x} u_m^{n\prime} = u_{m+1}^{n+1} - u_{m-1}^{n+1}$

Then equation (11) become as,

$$p\frac{\triangle_x^2 u_m^{n+1}}{h^2} = \frac{u_m^{n+1} - u_m^n}{k}$$

$$+ f(x_m, t_{n+1}, u_m^{n+1}, \frac{u_{m+1}^{n+1} - u_{m-1}^{n+1}}{2h})$$

This is analogus to *Laasonen methods.* In Laasonen scheme , solution value at (x_m, t_{n+1}) on $(n+1)$-th level is dependent on the solution values at the neighbouring points on the same level and one value on n-th level. It is stable for all values of mesh ratio parameter λ. So such scheme is unconditionally stable scheme. But we will solve our probem by Crank-Nicolson method.

θ =1/2 corresponds to the Crank-Nicolson method. Then,

$$t_n' = \frac{t_n + t_{n+1}}{2}$$

$$u_{m+q}^{n\prime} = \frac{u_{m+q}^{n+1} + u_{m+q}^{n}}{2}$$

so, for $q = 0$,

$$u_m^{n\prime} = \frac{u_m^{n+1} + u_m^n}{2}$$

so for $q = -1$,

$$u^n_{m-1}{}' = \frac{u^{n+1}_{m-1} + u^n_{m-1}}{2}$$

so for $q = +1$,

$$u^n{}'_{m+1} = \frac{u^{n+1}_{m+1} + u^n_{m+1}}{2}$$

So

$$\triangle_{2x} u^{n\prime}_m = u^{n\prime}_m - u^n_{m-1}{}'$$

For Crank-Nicolson method equation (10) become

$$p\triangle_x^2 \frac{(u^{n+1}_m + u^n_m)}{2h^2} = \frac{u^{n+1}_m - u^n_m}{k}$$

$$+f(x_m, \frac{t_n + t_{n+1}}{2}, \frac{u^{n+1}_m + u^n_m}{2}, \frac{u^{n+1}_{m+1} - u^n_{m-1} + u^n_{m+1} - u^n_{m-1}}{4h})$$

We solved the Burgers equation and KdV equation in absense and presence of perturbation by the above method.

4.2 Burgers equation in absence of Perturbation

Burgers equation is the nonlinear diffusion equation

$$u_t + uu_x = c_1 u_{xx}$$

where c_1 is the constant. The Analytical solution of the above equation is of the form $u(x,t) = c[1 - tanh((x - ct)/2c_1)]$ for all values of 'c'. We solve the above equation numerically by Crank-Nicolson method. The initial and boundary condition are given from analytical solution.

Initial condition :

$$u(x,0) = c[1 - tanh(x/2c_1)]$$

$for a < x < b$

Boundary condition :

$$u(a,t) = c[1 - tanh((a - ct)/2c_1)]$$

and

$$u(b,t) = c[1 - tanh((b - ct)/2c_1)]$$

The nodals points are given by,

$$x_m = mh, m = 0, 1, 2, \cdots, M$$

$$t_n = nk, n = 0, 1, 2, \cdots, N$$

where $h = \frac{(b-a)}{M}, k = \frac{T}{N}$ We write the given differential equation (Burgers equation) as,

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial}{\partial x}u^2 = c_1\frac{\partial^2 u}{\partial x^2}$$

The Crank-Nicolson type method gives the system of equations

$$\frac{u_m^{n+1} - u_m^n}{k} + \frac{1}{2}\frac{\Delta_x}{h}u^2 = \frac{c_1}{h^2}\Delta_x^2 u$$

For better approximation u is replaced by

$$u = \frac{u_m^{n+1} + u_m^n}{2}$$

Burgers equation becomes,

$$\frac{u_m^{n+1} - u_m^n}{k} + \frac{1}{2h}\Delta_x(\frac{u_m^{n+1} + u_m^n}{2})^2 = \frac{c_1}{h^2}\Delta_x^2\frac{u_m^{n+1} + u_m^n}{2}$$

Since it is a set of nonlinear equations it can not be solved by matrix-inversion method.

For this, we linearize the equation in the following way. We substitute,

$$u_m^{n+1} = u_m^n + v_m^n$$

where v_m^n tends to zero. So we get,

$$\frac{v_m^n}{k} + \frac{1}{8h}\Delta_x(2u_m^n + v_m^n) = \frac{c_1}{2h^2}\Delta_x^2(2u_m^n + v_m^n)$$

Let $\frac{k}{h} = c$ and $\frac{k}{h^2} = \lambda$ Finally, We get

$$-v_{m-1}^n(\frac{c_1\lambda}{2} + \frac{cu_{m-1}^n}{4}) + v_m^n(1 + c_1\lambda) + v_{m+1}^n(\frac{c}{4}u_{m+1}^n - \frac{c_1\lambda}{2}) = c_1\lambda[u_{m+1}^n - 2u_m^n + u_{m-1}^n] + \frac{c}{4}[(u_{m-1}^n)^2 - (u_{m+1}^n)^2]$$

From initial condition, we find the value of v_m^0 at the 0 label. Then we find the value of u_m^1 from the value of u_m^0 and v_m^0 After that we find the u_m^1, as $u_m^1 = u_m^0 + v_m^0$

4.3 Burgers equation in presence of perturbation

In presence of perturbation Burgers equation takes the form as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = c_1\frac{\partial^2 u}{\partial x^2} + \epsilon u$$

Where the term ϵu comes from the effect of perturbation. If we transform the above equation in numerical field, we get

$$\frac{u_m^{n+1} - u_m^n}{k} + \frac{\Delta_x}{2h}u^2 = \frac{c_1\Delta_x^2}{2h^2}\epsilon u$$

For better approximation, we substitute

$$u = \frac{u_m^{n+1} + u_m^n}{2}$$

we get,

$$\frac{u_m^{n+1} - u_m^n}{k} + \frac{\Delta_x}{2h}\frac{(u_m^{n+1} + u_m^n)^2}{4} = \frac{c_1\Delta_x^2}{h^2}\frac{u_m^{n+1} + u_m^n}{2} + \epsilon\frac{u_m^{n+1} + u_m^n}{2}$$

To linearize we put

$$u_{m+1}^n = u_m^n + v_m^n$$

So we get,

$$-v_{m-1}^n(\frac{c_1\lambda}{2} + \frac{cu_{m-1}^n}{4}) + v_m^n(1 + c_1\lambda - \frac{\epsilon}{2}) + v_{m+1}^n(\frac{cu_{m+1}^n}{4} - \frac{c_1\lambda}{2}) = c_1\lambda[u_{m+1}^n - 2u_m^n + u_{m-1}^n] +$$
$$cu_m^n + c[\frac{(u_{m-1}^n)^2 - (u_{m+1}^n)^2}{4}]$$

Where $\epsilon < 1$

4.4 KdV equation in absence of perturbation

KdV equation is almost similar to Burgers equation except the fact that it's a third order differential equation.

The general form of KdV equation in absence of perturbation

$$\frac{\partial u}{\partial t} - 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0$$

By Crank-Nicolson difference method, if we transform the above equation, we get

$$\frac{u_m^{n+1} - u_m^n}{k} - \frac{3\triangle_x}{h}u^2 + \frac{\triangle_x^3}{h^3}u = 0$$

For better approximation u is replaced by $\frac{u_m^{n+1} + u_m^n}{2}$

So we get,

$$\frac{u_m^{n+1} - u_m^n}{k} - \frac{3\triangle_x}{4h}(u_m^{n+1} + u_m^n)^2 + \frac{\triangle}{h^3}\frac{u_m^{n+1} + u_m^n}{2} = 0$$

To linearize the nonlinear equation, we substitute

$$u_m^{n+1} = u_m^n + v_m^n$$

So,

$$\frac{v_m^n}{k} - \frac{3}{4h}\triangle_x 2u_m^n + {v_m^n}^2 + \frac{\triangle_x^3}{2h^3}(2u_m^n + v_m^n) = 0$$

Let $\frac{k}{h} = c$ and $\frac{k}{h^3} = \lambda$

Finally we get,

$$v_{m-1}^n(-\frac{\lambda}{2} + 3c\frac{u_{m-1}^n}{2}) + v_m^n(1 + \frac{3\lambda}{2}) - v_{m+1}^n(\frac{3\lambda}{2} + \frac{3cu_{m+1}^n}{2}) + v_{m+2}^n\frac{\lambda}{2} =$$

$$-\lambda[u_{m+2}^n - 3u_{m+1}^n + 3u_m^n - u_{m-1}^n] + \frac{3c}{2}[{u_{m+1}^n}^2 - {u_{m-1}^n}^2]$$

We solved the KdV equation numerically in absence of perturbation for two cases.

Case(I): When number of soliton is one. The analytical solution for One soliton case is

$$u(x,t) = -2sech^2(x - 4t)$$

Case(II): When number of solitons is two. Then the analytical solution of the KdV equation is of the form

$$u(x,t) = -12\frac{3 + 4cosh(2x - 4t) + cosh(4x - 64t)}{(3cosh(x - 28t) + cosh(3x - 64t))^2}$$

The initial and boundary conditions were fixed from the above analytical solution.

4.5 KdV equation in presence of perturbation

The physical situations that give rise to standard soliton equations tend be highly idealized. Inclusin of effects of that are present in more realistic experimental situations, especially various forms of dissipation, leads to equations that differ from standard equations. If the additional terms in the equations are small in some sense , we may expect that at least for some interval of time, their effect on the various phenomena considered previously may be small. Now we consider the perturbed KdV equation which differ from the original KdV equation in having extra term.

The perturbed KdV equation is of the form

$$u_t - 6uu_x + u_xxx = \epsilon R(u)$$

where $\epsilon << 1$ and R(u) is some specified function of the solution u(x,t).

We transforme the above equation to Crank-Nicolson numerical equation as

$$\frac{u_m^{n+1} - u_m^n}{k} - 3\Delta_x \frac{(u_m^{n+1} + u_m^n)^2}{4} + \frac{\Delta_x^3}{h^3} \frac{u_m^{n+1} + u_m^n}{2} = \epsilon \frac{u_m^{n+1} + u_m^n}{2}$$

We linearize above equation by substituting,

$$u_m^{n+1} = u_m^n + v_m^n$$

Finally we get,

$$v_{m-1}^n(\frac{-\lambda}{2} + \frac{3c}{2}u_{m-1}^n) + v_m^n(1 + \frac{3\lambda}{2} - \frac{\epsilon}{2}) - v_{m+1}^n(\frac{3\lambda}{2} + \frac{3cu_{m+1}^n}{2}) + v_{m+2}^n(\frac{\lambda}{2}) =$$

$$-\lambda(u_{m+2}^n - 3u_{m+1}^n + 3u_m^n - u_{m-1}^n) + \epsilon u_m^n + \frac{3c}{2}[(u_{m+1}^n)^2 - (u_{m-1}^n)^2]$$

Chapter 5

Algorithm & Results

5.1 General algorithm for solving Burgers and KdV equation

1. Declare three 2-dimensional array u[N][M] , v[N][M] , inverse[N][M] and one 1-dimensional array c[N].

2. Initialize the matrix Inverse[N][M] by making it the diagonal matrix(i.e all elements except the diagonal elements are zero and value of each diagonal elements is made to 1). Initialize the elements of u[0][M] by using the initial condition. Enter all the boundary condition in u[N][0] and u[N][M-1].

3. Using the Gouse-Jordan method find the inverse of the matrix u[0][M] from where we find the v[0][M] by matrix multiplying the u[0][M] and c[N]. By knowing the value of u[0][M] and v[0][M] at level 0 we find u[1][M] using the equation $u_m^{n+1} = u_m^n + v_m^n$

4. Repeat the step 3 untill all N level is finished. In this way we find the u[N][M] for all value of N. (By iteration)

We tested the algorithm for :

lower-x-value = 0.0

higher-x-value = 10.0

time-level = 0.5

M = 20.0

N = 50.0

5.2 Results

The Results are given in table 1-4 and figures 1-6 In following table for result we use the symbols $u(x,t)$ for solution for $\epsilon = 0$, $u^1(x,t)$ for solution for $\epsilon = 0.001$, $u^2(x,t)$ solution for $\epsilon = 0.01$

5.3 Discussion

In this report we applied numerical techniques to obtain solitary wave solutions of the Burgers and KdV equation both with and without a perturbation term of the form cu.

The numerical solutions of the KdV equation have been attempted before by several authors (for a complete treatment see R.K. Dodd et.al (1982)).

However, as mentioned earlier for many realistic situations pure KdV and Burgers equation may not describe the physical situation. And though perturbation of KdV equation was discussed analytically through approximate solutions simultaneous treatment of the Burgers equation and the KdV equation with the cu terms were not found in the literature to the best of the author's knowledge. For weakly nonlinear care the following realistic equation viz

$$\frac{\partial}{\partial z}[\frac{\partial u}{\partial z} + u\frac{\partial u}{\partial z} - \frac{\alpha}{2c}\frac{\partial^3 u}{\partial z^3}] = -\frac{c}{2}(\frac{\partial^2 u}{\partial u^2} + \frac{\partial^2 u}{\partial u^2})$$

was used to study the stability of the solution in the form of one dimensional soliton by Kadomstev and Petviashvili (1970), as a first step towards the study of the stability of the KdV equation.

Table 1			
Numerical Solutions of Burgers Equations for $t = 0.1$			
x	u(x,t)	$u^1(x,t)$	$u^2(x,t)$
0.0	0.503125	0.503125	0.503125
0.5	0.471737	0.475934	0.515551
1.0	0.440623	0.445022	0.486649
1.5	0.409945	0.414075	0.453174
2.0	0.379953	0.383780	0.420040
2.5	0.350853	0.354300	0.387860
3.0	0.322827	0.326080	0.356881
3.5	0.296023	0.299000	0.327245
4.0	0.270560	0.273286	0.299092
4.5	0.246525	0.249008	0.272510
5.0	0.223974	0.226230	0.247586
5.5	0.202936	0.204972	0.224326
6.0	0.183410	0.185257	0.202739
6.5	0.165937	0.167042	0.182803
7.0	0.148798	0.150296	0.164474
7.5	0.133618	0.134963	0.142693
8.0	0.119782	0.120987	0.132397
8.5	0.107423	0.108503	0.118725
9.0	0.099675	0.100659	0.109967
9.5	0.129465	0.130466	0.139903

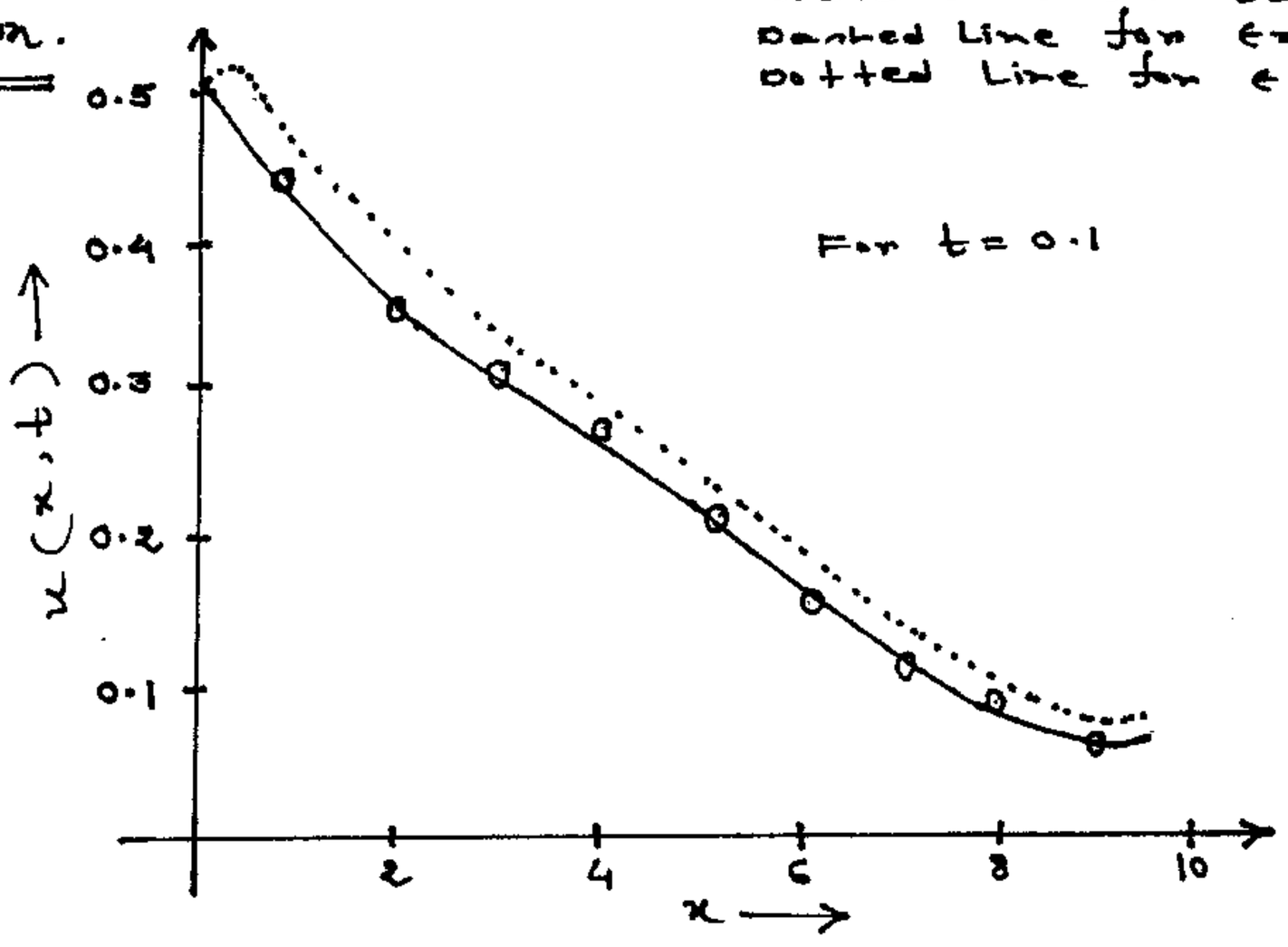
Table 2			
Numerical Solutions of Burgers Equations for $t = 0.3$			
x	u(x,t)	$u^1(x,t)$	$u^2(x,t)$
0.0	0.509374	0.509374	0.509374
0.5	0.477725	0.488239	0.595980
1.0	0.446299	0.459170	0.593350
1.5	0.415255	0.427880	0.560397
2.0	0.384863	0.396674	0.520814
2.5	0.355352	0.366269	0.481041
3.0	0.326913	0.336955	0.442510
3.5	0.299707	0.308909	0.405641
4.0	0.273858	0.282263	0.370614
4.5	0.249549	0.257111	0.337536
5.0	0.226570	0.233517	0.306519
5.5	0.205221	0.211510	0.277593
6.0	0.185414	0.191094	0.250763
6.5	0.167128	0.172246	0.225999
7.0	0.150328	0.154929	0.203951
7.5	0.134997	0.139127	0.182490
8.0	0.121418	0.125125	0.164042
8.5	0.111924	0.115295	0.150631
9.0	0.118694	0.121953	0.155910
9.5	0.182862	0.186231	0.220670

Table 3			
Numerical Solutions of KdV Equations for $t = 0.1$			
x	u(x,t)	$u^1(x,t)$	$u^2(x,t)$
0.0	-2.000572	-2.000572	-2.000572
0.5	-2.815563	-2.815715	-2.837367
1.0	-3.741462	-3.711488	-3.641392
1.5	-6.250241	-6.248875	-6.180888
2.0	-7.854316	-7.932220	-8.596443
2.5	-5.143560	-5.255533	-6.347138
3.0	-1.438573	-1.486145	-1.996959
3.5	-0.190862	-0.197163	-0.268173
4.0	-0.026590	-0.027164	-0.033326
4.5	-0.006166	-0.006247	-0.007064
5.0	-0.001978	-0.001999	-0.002202
5.5	-0.000703	-0.000710	-0.000778
6.0	-0.000256	-0.000259	-0.000283
6.5	-0.000094	-0.000095	-0.000104
7.0	-0.000035	-0.000035	-0.000038
7.5	-0.000013	-0.000013	-0.000014
8.0	-0.000005	-0.000005	-0.000005
8.5	-0.000002	-0.000002	-0.000002
9.0	-0.000000	-0.000000	-0.000000
9.5	-0.000000	-0.000000	-0.000000

Table 4			
Numerical Solutions of KdV Equations for $t = 0.3$			
x	u(x,t)	$u^1(x,t)$	$u^2(x,t)$
0.0	-1.345319	-1.345319	-1.345319
0.5	-2.381620	-2.384593	-2.403789
1.0	-2.723250	-2.740460	-2.842774
1.5	-2.791832	-2.840653	-3.198895
2.0	-2.445207	-2.522205	-3.216412
2.5	-1.395237	-1.969345	-2.788339
3.0	-1.512204	-1.553121	-2.174997
3.5	-1.522773	-1.509320	-1.729572
4.0	-2.143284	-2.041510	-1.620994
4.5	-3.761751	-3.541175	-2.023901
5.0	-5.955281	-5.883580	-3.440566
5.5	-5.634232	-6.053969	-6.564178
6.0	-2.552378	-3.033682	-8.170043
6.5	-0.478381	-0.625148	-4.859722
7.0	-0.048794	-0.065148	-1.103634
7.5	-0.004309	-0.005662	-0.103399
8.0	-0.000378	-0.000485	-0.007466
8.5	-0.000034	-0.000043	-0.000524
9.0	-0.000002	-0.000002	-0.000019
9.5	-0.000001	-0.000001	-0.000002

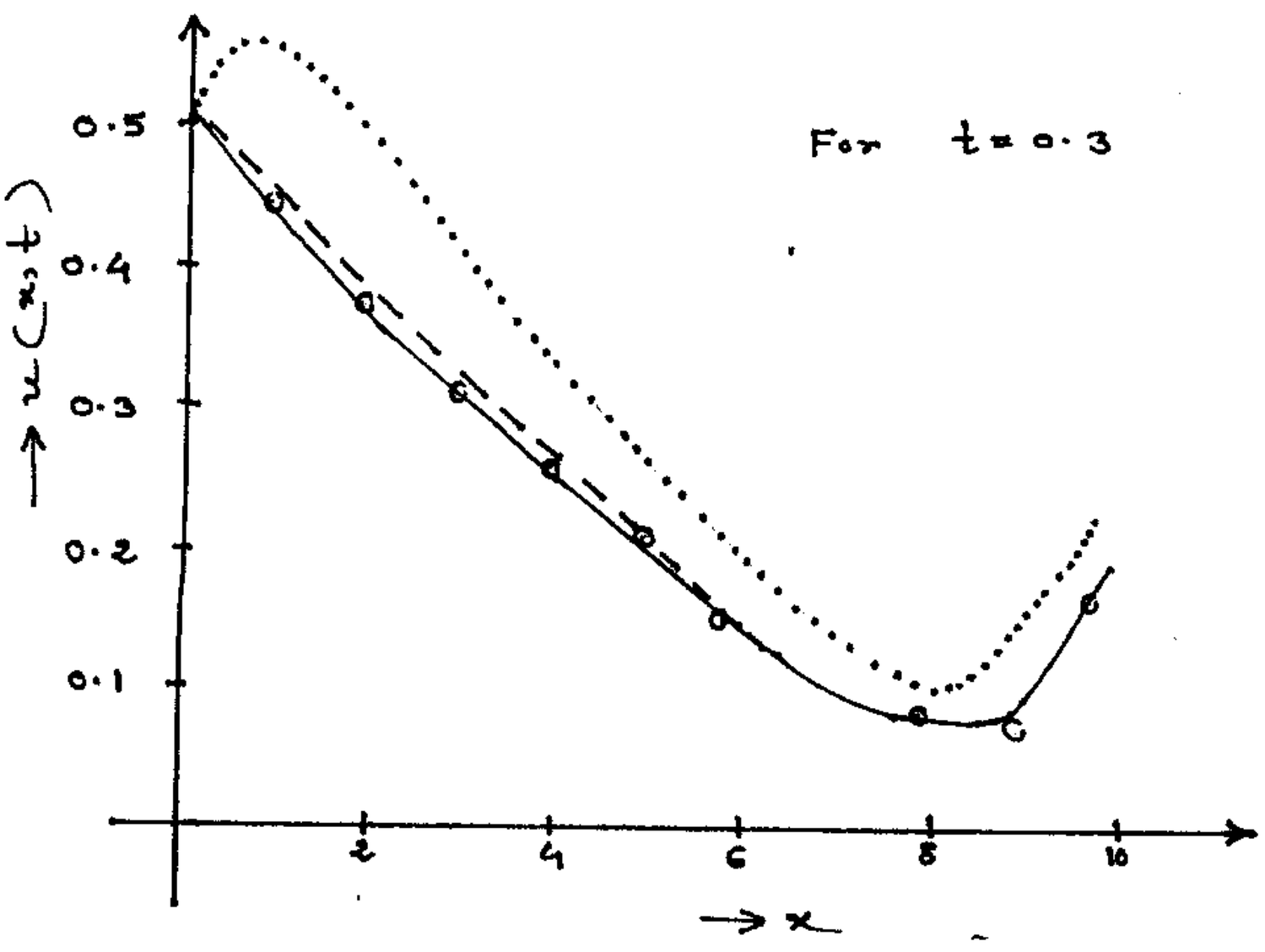
To start with one can take extra terms like ϵ R(u). The form of R(u) will depend on the physical problem. However, detailed study of the perturbation, apart form the linear form of R(u) is out side the scope of this report. We took R(u) = u, to perturb the KdV and the Burgers equation. For small ϵ the equations retained the soliton solutions. However as ϵ increased the solutions differed significantly from the solitary wave type solution as can be seen from the numerical solutions. This study is just the first step towards the perturbation of the KdV and the Burgers equation and that is why its aim was rather humble. However, one hopes these techniques can be applied to more realistic situations.

Burgers Equation.



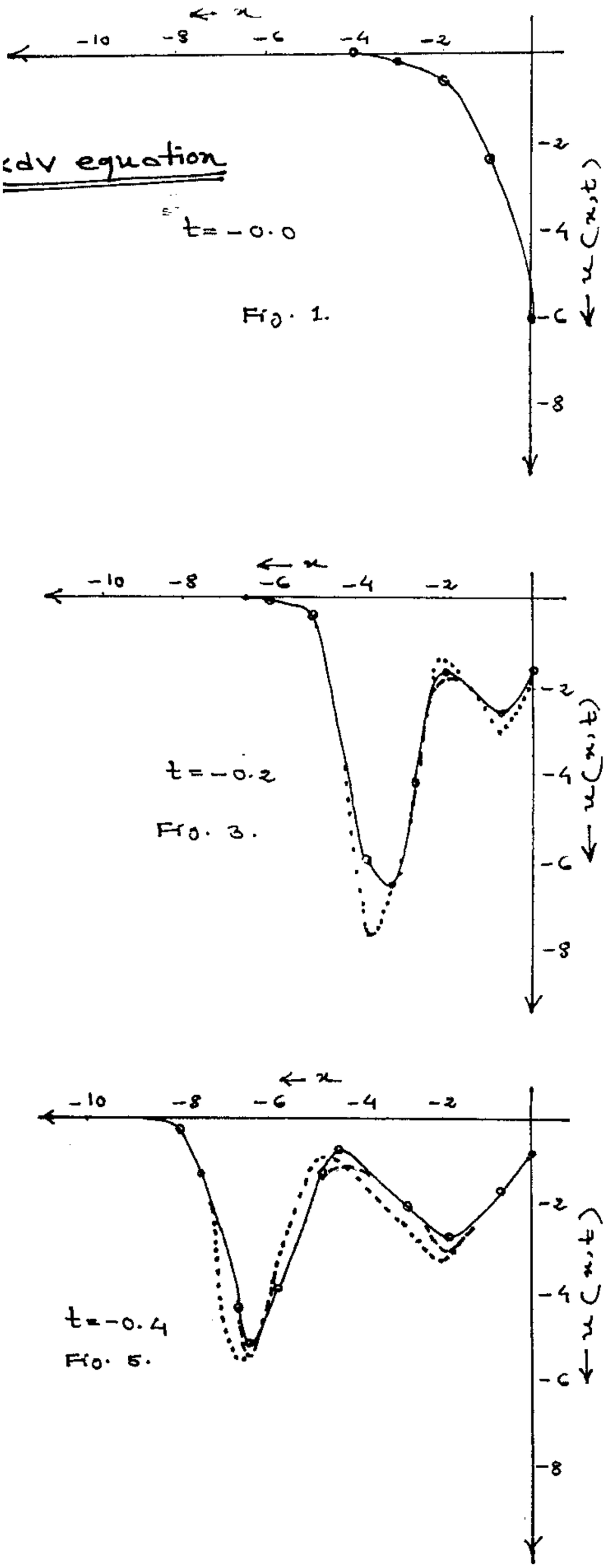
Solid Line for $\epsilon = 0$
Dashed Line for $\epsilon = 0.001$
Dotted Line for $\epsilon = 0.01$

For $t = 0.1$



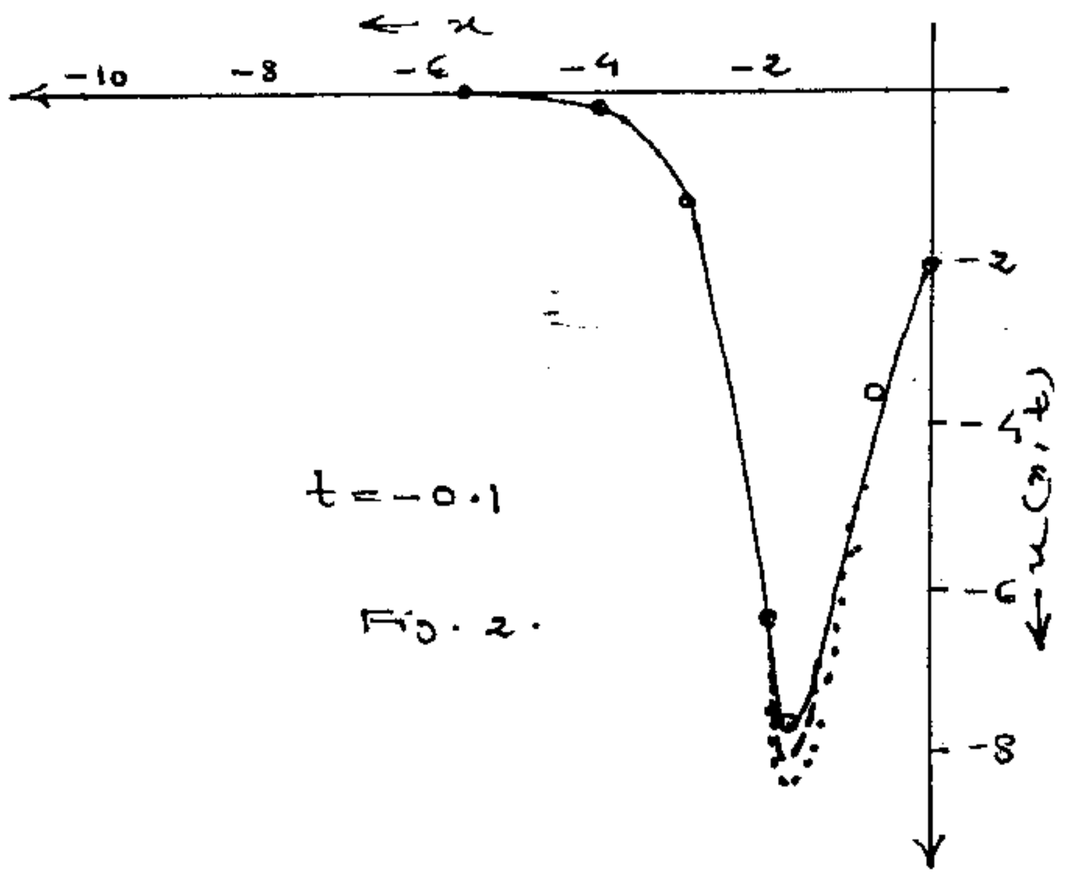
For $t = 0.3$

KdV equation



t = -0.0

Fig. 1.

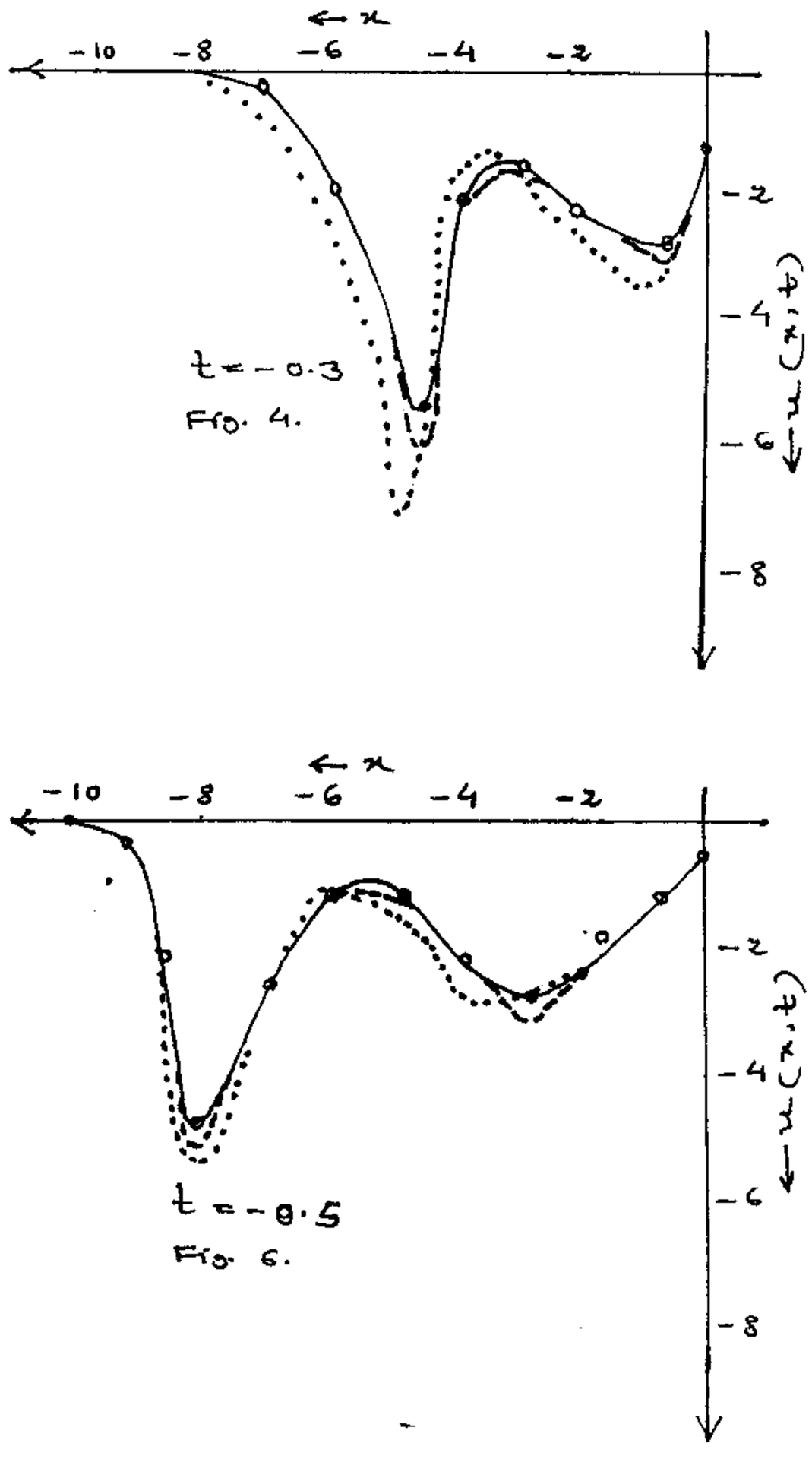


t = -0.1

Fig. 2.

t = -0.2

Fig. 3.



t = -0.3

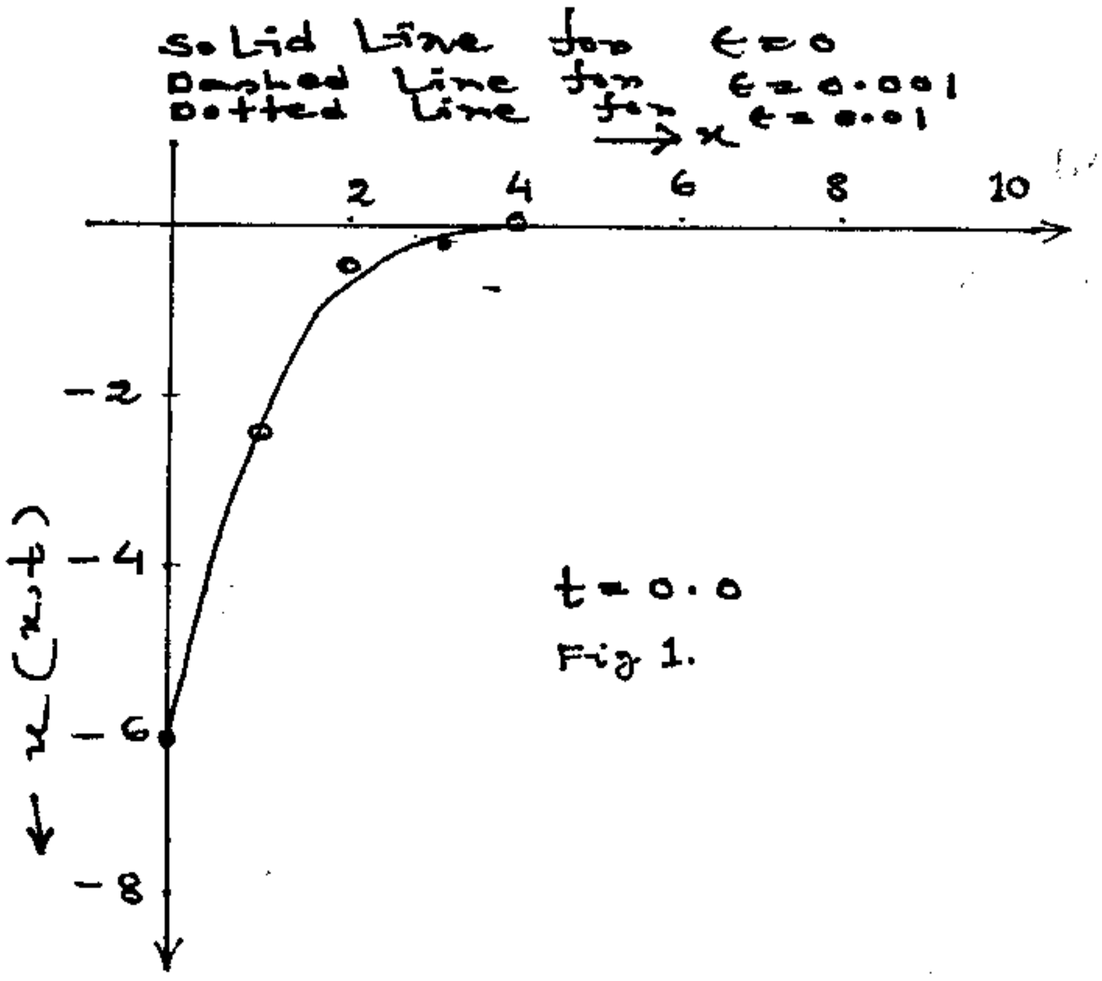
Fig. 4.

t = -0.4

Fig. 5.

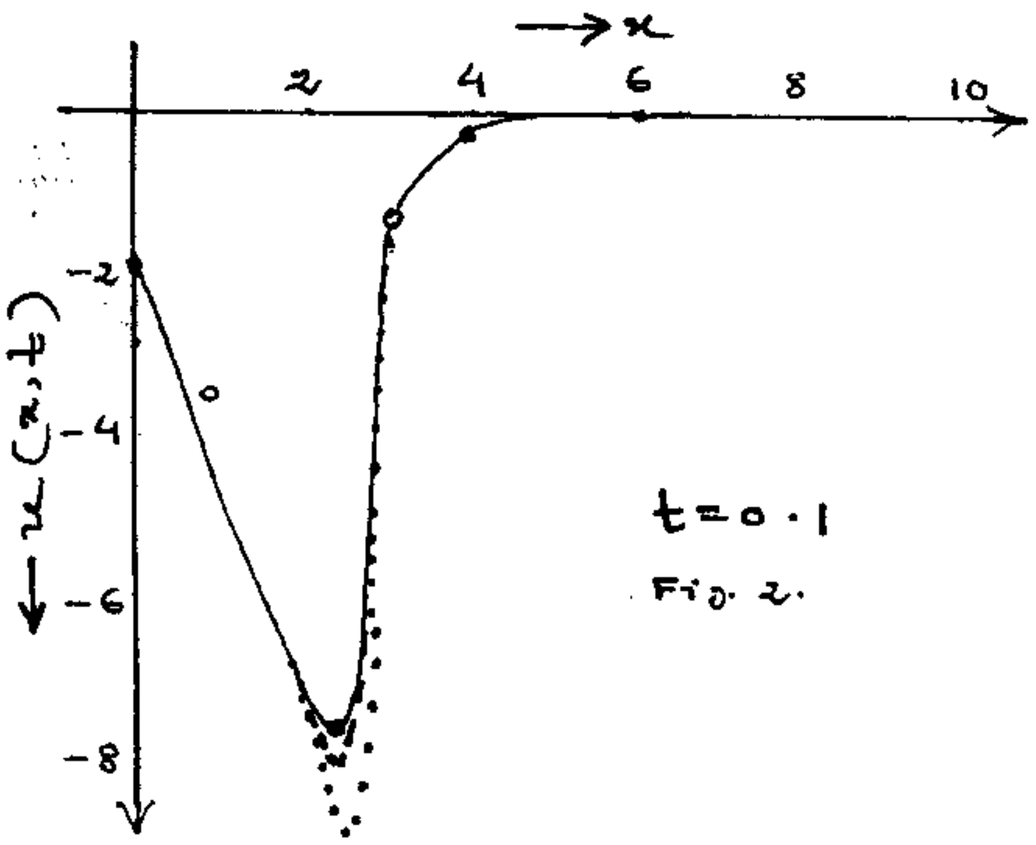
t = -0.5

Fig. 6.



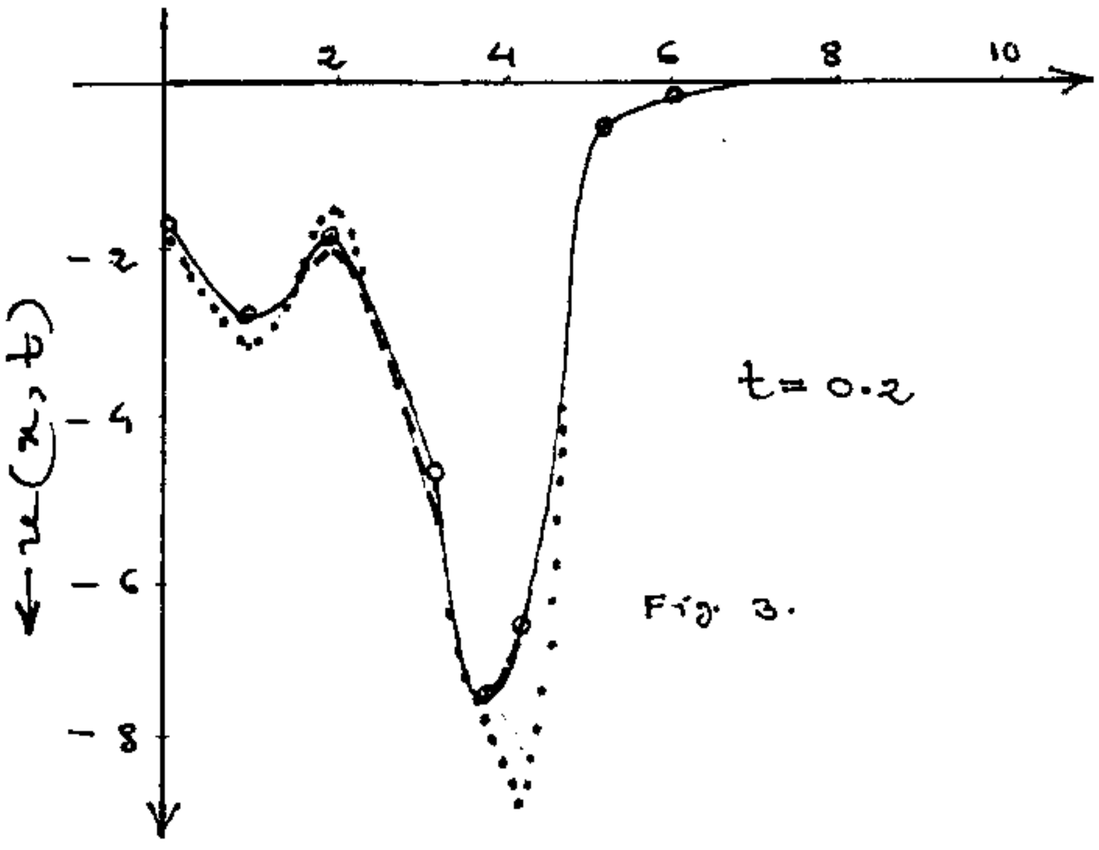
t = 0.0

Fig 1.



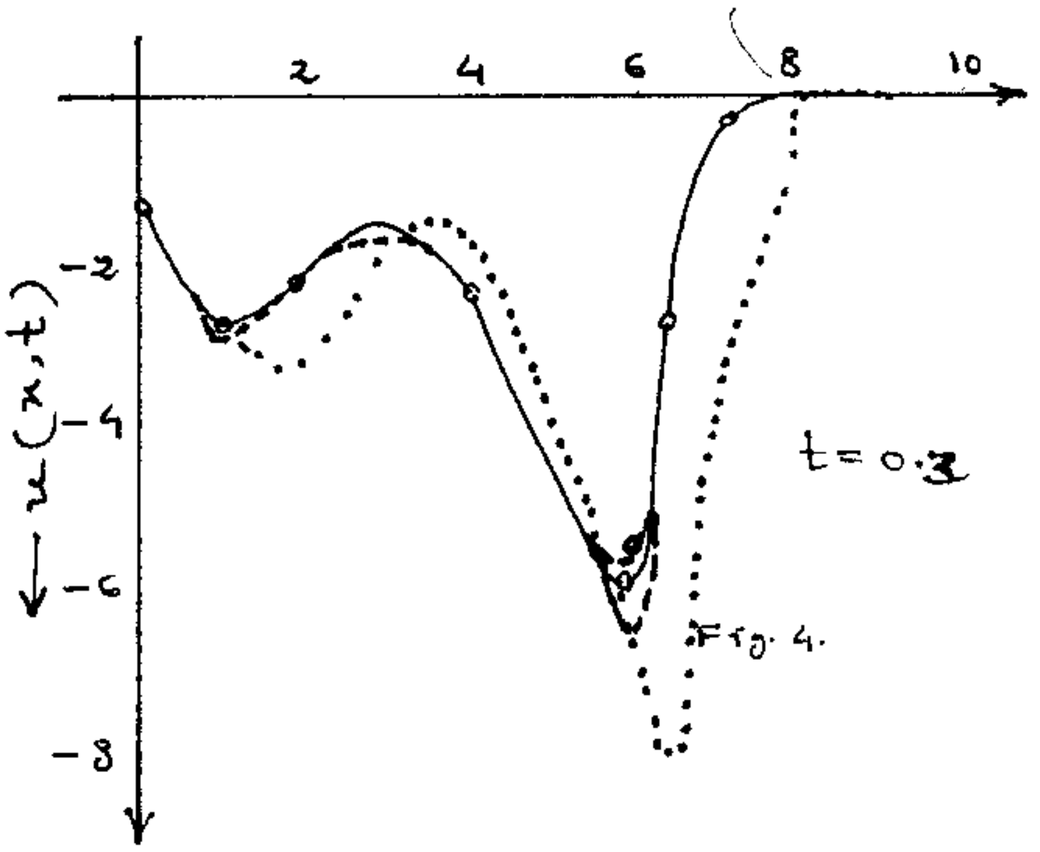
t = 0.1

Fig. 2.



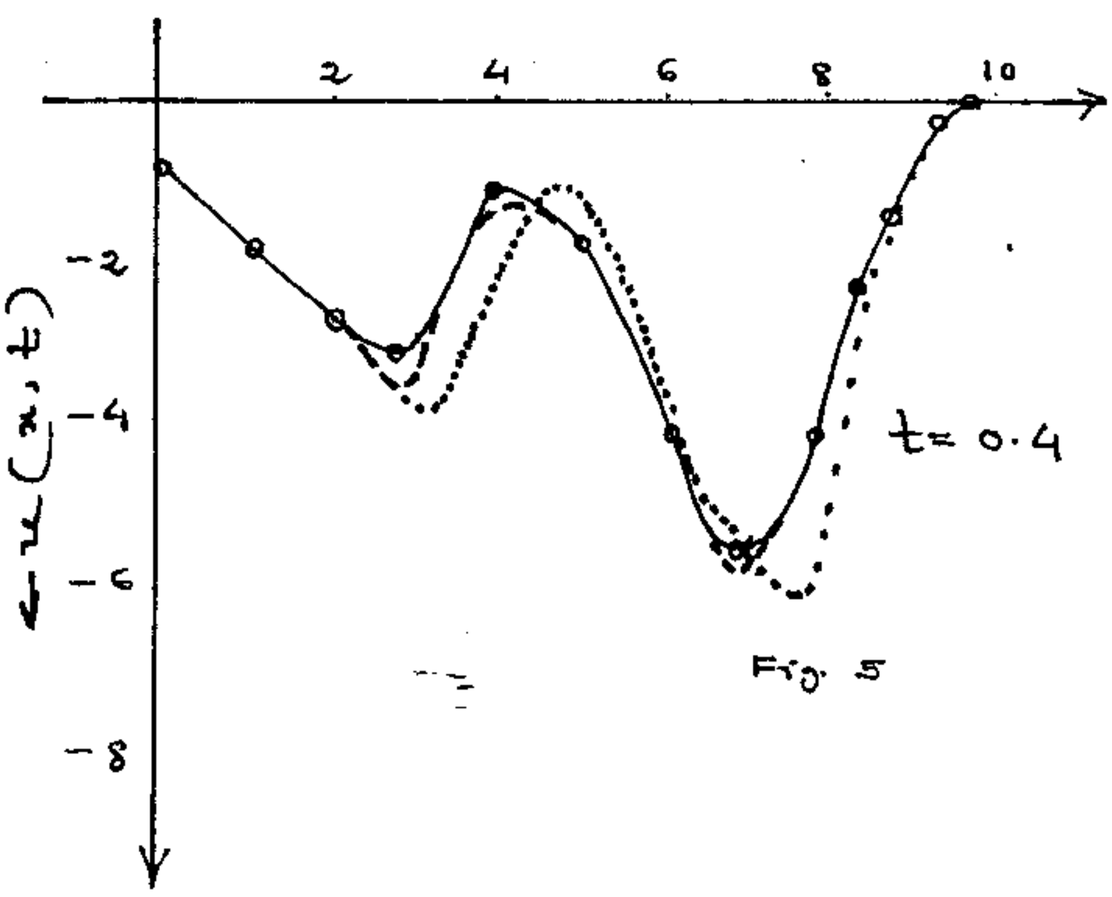
t = 0.2

Fig. 3.



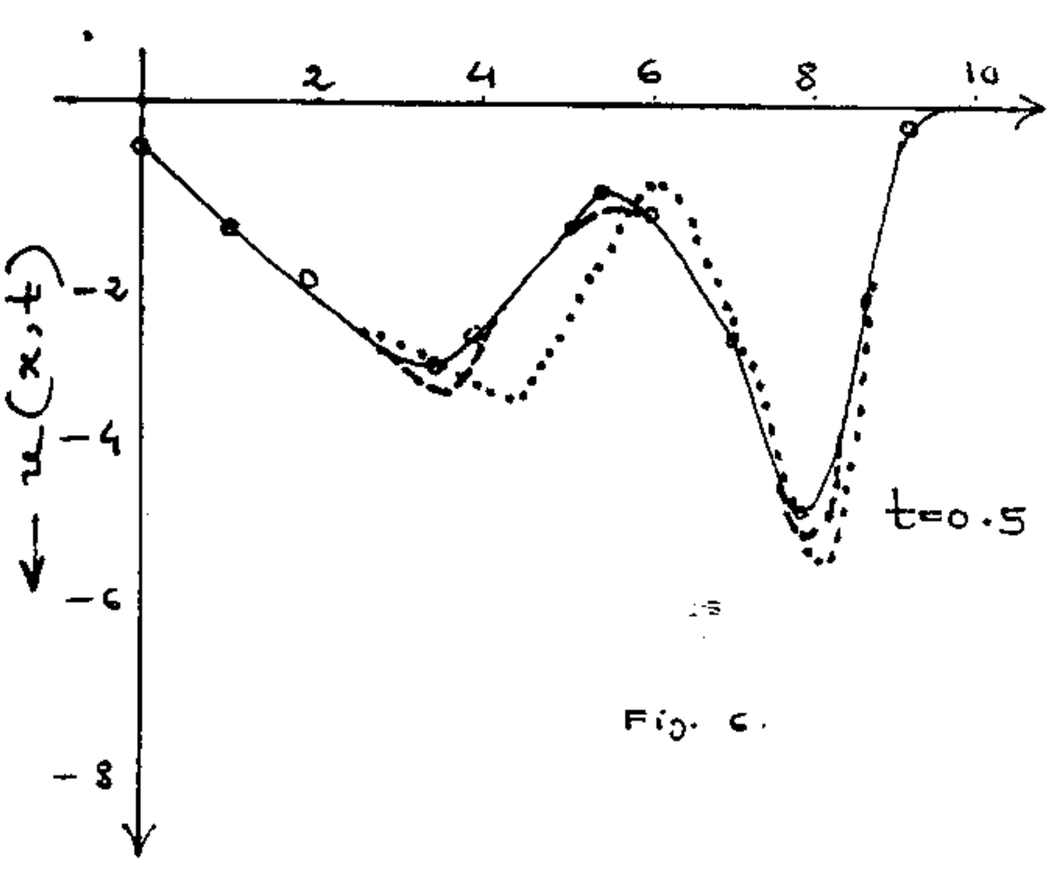
t = 0.3

Fig. 4.



t = 0.4

Fig. 5



t = 0.5

Fig. 6.

Bibliography

[1] Benny, D.J. 1966 - Long non-linear waves in fluid flows. J. Math & Phys. **45**, 52-63.

[2] Burgers, J.M. 1948 - A mathematical model illustrating the theory of turbulence. Adv. Appl. Mech. **1**, 171-199.

[3] Dodd, R.K et al. 1982 -Solitons and nonlinear wave equations-Academic press N.Y.

[4] Drazin, P.G. 1983 -: Solitons :- Cambridge University Press and References there in.

[5] Gardner, C.S. Greene, J.M. Kruskal, M.D. and Miura, R.M. 1967 - Method for solving the Korteweg-de Vries equation. Phys. Rev. Lett. **19**, 1095-2007.

[6] Kadomstev, B.B. and Petviashvili V.I. 1970 - On the stability of solitary waves in a weakly dispersing media Dokl. Akad. Nauk. SSSR **192**, 753-756.

[7] Karpman V, I and Maslov, E.M. 1978 - Perturbation theory for solitons. Sov. Phys. JETP. **46**, 281-291.

[8] Korteweg, D.J. and de Vries, G. 1895 - On the change of form of long waves advancing in a rectangular canal. Phil. Mag. (5) **39**, 422-413.

[9] Perring, J.K. and Skyrme, T.H.R 1962 - A model unifield field equation. Nucl. Phys. **31**, 550-555.

[10] Smith, G.D. Numerical solution of partial differential equations : Finite difference methods, Clarendon press, Oxford, 1978.

[10] Smith, G.D. Numerical solution of partial differential equations : Finite difference methods, Clarendon press, Oxford, 1978.

[11] Washimi, H. and Taniuti, T. 1966 - Propagation of ion-acoustic solitary waves of small amplitudes. Phys. Rev. Lett. **17**, 996-998.

[12] Wijngaarden, L. Van 1968 - On the equations of motion for mixtures of liquid and gas bubbles. J. Fluid. Mech. **33**, 465-474.

[13] Zabusky, N.J. and Kruskal. M.D. 1965 - Interactions of solitons in a collision less Plasma and the recurrence of initial state. Phys. Rev. Lett. **15**, 240-243.

APPENDIX

Source Code of the Program

```
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <process.h>
#include <alloc.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <graphics.h>
#include <ctype.h>

#define  N  55
#define ESC 27
#define ENTER 13
#define LEFT 75
#define RIGHT 77

void *p;

int ascii,scan;


void write_matrix(matrix,order)
float matrix[][N];
int order;
    {
     int row,column;
     for(row=0;row<order;++row)
        {
         for(column=0;column<order;++column)
         printf("%f       ",matrix[row][column]);
         printf("\n\n");
        }
    }

void exchange_rows(matrix,inverse,row1,row2,order)
float matrix[][N],inverse[][N];
int row1,row2,order;
     {
      int column;
      float temp;
      for(column=0;column<order;++column)
            {
             temp=matrix[row1][column];
             matrix[row1][column]=matrix[row2][column];
             matrix[row2][column]=temp;
             temp=inverse[row1][column];
             inverse[row1][column]=inverse[row2][column];
             inverse[row2][column]=temp;
            }
      }


int invert_matrix(matrix,inverse,order)
float matrix[][N],inverse[][N];
int order;
```

```
{
 int row,column,current,singular=0;
 float ratio;
 for(row=0;row<order && !singular;++row)
 if(matrix[row][row])/*if diagonal elements are not zero*/
   for(current=0;current<order;++current)
       if(current==row)
           {
            ratio=matrix[row][row];
            for(column=0;column<order;++column)
                {
                 matrix[row][column]/=ratio;
                 inverse[row][column]/=ratio;
                }
          }
       else
        {
         ratio=matrix[current][row]/matrix[row][row];
         for(column=0;column<order;++column)
             {
              matrix[current][column]-=ratio*matrix[row][column];
              inverse[current][column]-=ratio*inverse[row][column];
             }
        }
else
    {
     singular=1;
     for(current=row+1;(current<order)&& singular;++current)
        if(matrix[current][row])
            {
             singular=0;
             exchange_rows(matrix,inverse,current,row,order);
             --row;
            }
     }
     return(!singular);


    }/*end of fnction inverse*/

float c[N];

multi(a,b,c,order)
float a[][N],b[N],c[N];
int order;
   {
    int i,j,k;
    for(i=0;i<order;i++)
        {
         c[i]=0.0;
         for(k=0;k<order;k++)
         c[i]=c[i]+a[i][k]*b[k];    /***********b[k][j]*****/
        }
 }

main()
      {
       int option,dl;
       float B,C,B1,C1,B2,C2,Bm,Cm;
       int v1,i,j,l;/*dl*//*v1---used for generate the no of soliton*/
```

```
float matrix[N][N],inverse[N][N],coeff[N],u[N][N],v[N][N];
float lamda,d,k,h,b,a,neu,din,neub,dinb,result,T;
int m,n;  /*number of nodal points along X--axis and t--axis
            is m+1 and n+1 respectively */
 /*spacing along X-axis is h=(b-a)/m */
    /*spacing along t-axis is k=T/n */
    /*lamda----->mesh ratio parameter*/
        /* here d=k/h is used in place of c=k/h in book*/
int choice,gm,gd=DETECT;/*constant used for*/
int result1;              /* graphics mode*/
size_t area;
printf("_____\n");
printf("enter the values of lower(=a) and upper(=b) limits");
printf("\n");
scanf("%f %f",&a,&b);
   printf("value of a = %f\nvalue of b = %f\n",a,b);
   printf("enter the  time interval T\n");
   scanf("%f",&T);
   printf("Number of time label = %f\n",T);
   printf("enter the values of m and n\n");
   scanf("%d  %d",&m,&n);
   printf("value of m = %d    value of n = %d",m,n);
   printf("\n----------\n");
   printf("spacing along space and time axis are\n");
   h=(b-a)/(float) m;
   k=(float)T/(float)n;
   printf("spacing along X-axis h= %f  spacing along t-axis k=%f",h,k)
   printf("\n");
/*  lamda=k/(h*h*h);*/
 /*  d=k/h;*/
   printf("enter the values of lamda and d\n");
  scanf("%f  %f",&lamda,&d);
  printf("value of lamda = %f \n value of d = %f \n",lamda,d);
  printf("\n----------\n");
  printf("How many(1/2) no soliton want to generate?\n");
  scanf("%d",&option);
  switch(option)
           {
  case 1:

        for(i=0;i<=n;i++)/*store result for first boundary*/
           {
            neub=(2/(exp(-4*i*k)+exp(4*i*k)));
            neub*=neub;
            u[i][0]=(-2)*neub;
            v[i][0]=(-2)*neub;
           }
        for(i=0;i<=n;i++)/*store result for second boundary*/
           {
            neub=(2/(exp(m*h-4*i*k)+exp(4*i*k-m*h)));
            neub*=neub;
            u[i][m]=(-2)*neub;
            v[i][m]=(-2)*neub;
           }
    for(i=1;i<m;i++)/*store result for initial condition*/
           {
            neub=(2/(exp(i*h)+exp((-1)*h)));
            neub*=neub;
            u[0][i]=(-2)*neub;
           }
```

```
                    break;
                case 2:
               /*store the result in the first boundary*/
               for(i=0;i<=n;i++)
                    {
                    neub=3.0+4*cosh(-8*i*k)+cosh(-64*i*k);
     dinb=(3*cosh(-28*i*k)+cosh(-36*i*k))*(3*cosh(-28*i*k)+cosh(-36*i*k));
                    u[i][0]=(-12*neub)/dinb;
                    v[i][0]=(-12*neub)/dinb;
                   }
                  /*store the result for second boundary*/
             for(i=0;i<=n;i++)
                    {
                    neub=3.0+4*cosh(2*m*h-8*i*k)+cosh(4*m*h-64*i*k);
dinb=(3*cosh(m*h-28*i*k)+cosh(3*m*h-36*i*k))*(3*cosh(m*h-28*i*k)+cosh(3*m*h-36
                    u[i][m]=(-12*neub)/dinb;
                    v[i][m]=(-12*neub)/dinb;
                   }
               /*store the values of initial condition in u[N][N]*/
               for(i=1;i<m;i++)
                    {
                    neu=3.0+4*cosh(2*i*h)+cosh(4*i*h);
                    din=(3*cosh(i*h)+cosh(3*i*h))*(3*cosh(i*h)+cosh(3*i*h));
                    u[0][i]=(-12*neu)/(din);
                    }
                      break;
               default:
                    printf("Enter either 1 or 2,Try again\n");
                    break;
                 }/*end of swtich*/

               for(l=0;l<n;l++)
                   {
                    for(i=0;i<(m-1);i++)
                       {
                        for(j=0;j<(m-1);j++)
                           {
                            if(i==j)
                              {
                              matrix[i][j]=1.0+((3.0)*lamda)/(2.0);
                              inverse[i][j]=1.0;
                              }
                else if(j-i==1)
               matrix[i][j]=(-1.0)*(((3.0)*lamda/2.0)+(((3.0)*d)/2.0)*(u[l][j+1])
                     else if(i-j==1)
                matrix[i][j]=(((-1.0)*lamda)/2.0)+(((3.0)*d)/2.0)*(u[l][j+1]);
                           else if(j-i==2)
                matrix[i][j]=(lamda)/(2.0);
                           else
                matrix[i][j]=0.0;
                            if(i!=j)
                             inverse[i][j]=0.0;

                           }
                       }
               B1=(-1.0)*lamda*(u[l][3]-3*u[l][2]+3*u[l][1]-u[l][0]);
               C1=(3.0)*(d/2.0)*(u[l][2]*u[l][2]-u[l][0]*u[l][0]);
               coeff[0]=B1+C1-(v[l][0]*(((-1.0)*lamda)/2.0+(((3.0)*d)/2.0)*u[l][0]));
               B2=(-1)*lamda*(u[l][m]-(3.0)*u[l][m-1]+(3.0)*u[l][m-2]-u[l][m-3]);
               C2=(3.0)*(d/2.0)*(u[l][m-1]*u[l][m-1]-u[l][m-3]*u[l][m-3]);
```

```
coeff[1]=B2+C2-v[1][m]*(lamda/2.0);

Bm=(-1.0)*lamda*(u[1][m+1]-(3.0)*u[1][m]+(3.0)*u[1][m-1]-u[1][m-2]);
Cm=(3.0)*(d/2.0)*(u[1][m]*u[1][m] -u[1][m-2]*u[1][m-2]);
coeff[m-2]=Bm+Cm+v[1][m]*(((3.0)*lamda/2.0)+((3.0)/2.0)*d*u[1][m]);

   for(i=1;i<m-3;i++)
       {
         B=(-1.0)*lamda*(u[1][i+3]-3*u[1][i+2]+3*u[1][i+1]-u[1][i]);
         C=(3.0)*(d/2.0)*(u[1][i+2]*u[1][i+2]-u[1][i]*u[1][i]);
         coeff[i]=B+C;
       }/*find the coefficient matrix*/
          if(! invert_matrix(matrix,inverse,m-1))
              {
              printf("matrix is singular,Not possible to find the solution");
              }
       multi(inverse,coeff,c,m-1);
       for(i=1;i<m;i++)
       v[1][i]=c[i-1];     /*find the v[N][N] matrix*/
              /*find u at the next time lavel with the
                help of u and v at the present lavel*/
        for(i=1;i<m;i++)
          {
           u[l+1][i]=u[l][i]+v[l][i];
          }
                 }/*end of l--for loop*/
         dl=(n)/5;
         for(l=0;l<=n;)
          {
          for(j=0;j<=m;j++)
             {
printf("-----------------------------------------\n");
printf("The solution at %d-th level at %d-th point is--\n",l,j);
   printf("%f\n",u[l][j]);
                }
             l=l+dl;
           }
        }
 /*detectgraph(&gd,&gm);*/
  initgraph(&gd,&gm,"c:\\tc");
 result1=graphresult();
 if(result1!=grOk)
    {
     closegraph();
     printf("Graphics error %s \n",grapherrormsg(result1));
     printf("Graphics Device Driver files not present in\n");
     printf(" directary %s \n","c:\\tc\\bgi");
     exit(1);
    }
 area=imagesize(220,165,420,315);
 p=malloc(area);
 if(p==NULL)
   {
    outtextxy(24,104,"Insufficient memory ! press any key to exit");
    fflush(stdin);
    getch();
    closegraph();
    restorecrtmode();
    exit(0);
   }
```

```
identity();
mainscreen();
clearviewport();
rectangle(0,0,getmaxx(),20);
rectangle(0,0,getmaxx(),20);
setcolor(YELLOW);
rectangle(0,50,getmaxx(),getmaxy());
setcolor(LIGHTCYAN);
rectangle(320-100,240-75,320+100,240+75);
setcolor(GREEN);
rectangle(320-100-5,240-75-5,320+100+5,240+75+5);
while(1)
      {

       setcolor(YELLOW);
display();
choice=getresponse("LDSPX",5);
switch(choice)
        {
         case 1:
                load();
                settextstyle(TRIPLEX_FONT,HORIZ_DIR,0);
                break;
         case 2:
                draw();
                settextstyle(TRIPLEX_FONT,HORIZ_DIR,0);
                break;
         case 3:
                save();
                settextstyle(TRIPLEX_FONT,HORIZ_DIR,0);
                break;
         case 4:
                print() ;
                settextstyle(TRIPLEX_FONT,HORIZ_DIR,0);
                break;
        case 5:
                closegraph();
                restorecrtmode();
                exit(0);
        }
          }
    }

identity()
          {
           int maxx,maxy;
           maxx=getmaxx();
           maxy=getmaxy();
           setcolor(LIGHTCYAN);
           rectangle(0,0,maxx,maxy);
           settextstyle(TRIPLEX_FONT,HORIZ_DIR,0);
           settextjustify(CENTER_TEXT,TOP_TEXT);
           outtextxy(maxx/2,4,"VISION ASSIGNMENT");
           outtextxy((maxx/2)-8,40,"--PROGRAM  CREATED BY--");
           setcolor(RED);
           outtextxy(maxx/2,maxy/2,"SUBIR BANDYOPADHYAY");
           sleep(3);
           clearviewport();
           }
```

```
            choice=1;
            if(choice==0)
            choice=count;
           }
    else
           {
            if(ascii==ENTER)
            return(choice);
            if(ascii==ESC)
            return(ESC);
            len=strlen(hotkeys);
            hotkeychoice=1;
            ascii=toupper(ascii);
            while(*hotkeys)
            {
                  if(*hotkeys==ascii)
                  return(hotkeychoice);
                  else
                  {
                      hotkeys++;
                      hotkeychoice++;
                  }
                  if(choice==count)
                  hotkeys=hotkeys-len;
                  printf("\a");
            }
      }

}
}
```

```
int  getkey()
     {
      union REGS i,o;
      while(!kbhit());
      i.h.ah=0;
      int86(22,&i,&o);
      ascii=o.h.al;
      scan=o.h.ah;
      return 0 ;
     }


  load()
     {
      char fname[30];
      int area,in,retvalue;
      settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
      outtextxy(40*8,6*19,"LOAD THE FILE FROM DISK....");
      sleep(2);
      cleartext(0);
      outtextxy(40*8,6*19,"PRESS ESC KEY TO EXIT...");
      sleep(3);
      cleartext(0);
      outtextxy(30*8,6*19,"ENTER THE FILE NAME...");
      retvalue=getstring1(fname);
      if(retvalue==ESC)
```

```
mainscreen()
            {
             int maxx,maxy,in,area;
             maxx=getmaxx();
             maxy=getmaxy();
             setcolor(RED);
             rectangle(0,0,maxx,maxy);
             settextstyle(TRIPLEX_FONT,HORIZ_DIR,0);
             settextjustify(CENTER_TEXT,TOP_TEXT);
             outtextxy(maxx/2,10,"IMAGE EDITOR");
             outtextxy(maxx/2+15,maxy/2,"PRESS ANY KEY TO CONTINUE.....");
             fflush(stdin);
             getch();
             clearviewport();
            }


display()
      {
       setcolor(LIGHTRED);
       outtextxy(72,19,"LOAD");
       outtextxy(200,19,"DRAW");
       outtextxy(328,19,"SAVE");
       outtextxy(456,19,"PRINT");
       outtextxy(584,19,"EXIT");
      }


int getresponse(hotkeys,count)
char *hotkeys ;
int count;
      {
       int col,choice=1,hotkeychoice,len;
       while(1)
          {
               col=(choice-1)*(79/count)+1;
               setcolor(LIGHTMAGENTA);
               outtextxy((col+8)*8,(3*19),"=>");
               getkey();
               if(ascii==0)
                    {
                     setcolor(getbkcolor());
                     outtextxy((col+8)*8,(3*19),"=>");
                     setcolor(LIGHTMAGENTA);
switch(scan)
    {
     case RIGHT:
                    choice++;
                    break;
     case LEFT:
                    choice--;
                    break;
    }
if(choice>count)
```

```
return;
outtextxy(50*8,6*19,fname);
in=open(fname,O_BINARY|O_RDONLY,0x0000);
if(in==-1)
     {
      cleartext(2);
      outtextxy(40*8,371,"UNABLE TO OPEN THE FILE ! PRESS ANY KEY TO EXIT")
      fflush(stdin);
      getch();
      cleartext(2);
      return;

     }
area=imagesize(220,165,420,315);
read(in,p,area);
close(in);
  putimage(220,165,p,COPY_PUT);
  cleartext(2);
 }



we()
 {
  char fname[30];
  int out,area,retvalue;
  delete();
  settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
  outtextxy(40*8,114,"SAVE THE FILE IN DISK");
  outtextxy(40*8,371,"PRESS ESC KEY TO EXIT");
  sleep(2);
  cleartext(1);
outtextxy(40*8,371,"ENTER THE FILE NAME :-");
retvalue=getstring1(fname);
outtextxy(50*8,371,fname);
if(retvalue==ESC)
return;
out=open(fname,O_BINARY|O_CREAT|O_RDWR,0666);
if(out==-1)
        {
         sleep(2);
         cleartext(1);
         outtextxy(40*8,371,"UNABLE TO STORE THE FILE ! PRESS ANY KEY TO EX
         fflush(stdin);
         getch();
         sleep(2);
         cleartext(2);
         return;
        }
  getimage(220,165,420,315,p);
  area=imagesize(220,165,420,315);
  write(out,p,area);
  close(out);
  cleartext(2);
}
```

```
print()
   {
    char ch;
    union REGS i,o;
    delete();
    settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
    cleartext(2);
    setcolor(YELLOW);
    outtextxy(30*8,6*19,"PRINT THE IMAGE.....");
    outtextxy(40*8,371,"PRESS ESC KEY TO EXIT...");
    sleep(3);
    cleartext(1);
    outtextxy(30*8,371,"SET UP THE PRINTER AND PRESS ESC KEY TO EXIT");
    sleep(3);
    cleartext(1);
    fflush(stdin);
    getkey();
    ch=getch();
    cleartext(0);
    if(ch==ESC)
    return;
    int86(5,&i,&o);
  }



draw()
   {
    int i,poly[100],no,retvalue,color;
    char temp[10],ch;
    delete();
    settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
    cleartext(0);
    outtextxy(40*8,6*19,"HOW MANY POINTS IN THE POLYGON");
    while(!kbhit());
    retvalue=getstring1(temp);
    if(retvalue==ESC)
    return;
    no=atoi(temp);
    outtextxy(50*9+20,6*19,temp);
    while(!kbhit());
    cleartext(0);
    outtextxy(40*8,371,"ENTER THE SCENE COORDINATES");
    for(i=0;i< 2*no;i++)
       {
        retvalue=getstring1(temp);
        if(retvalue==ESC)
        return;
        poly[i]=atoi(temp);
       {
        if(i< 10)
        outtextxy(30*i+50,400,temp);
        else if (i>=10 && i<20)
        outtextxy(30*i-300+50,420,temp);
        else
        outtextxy(30*i-600+50,440,temp);
       }
```

```
        if((i%2)==0)
                        poly[i]=220+(int)((5*poly[i])/16);
        else
              poly[i]=165+(int)((5*poly[i])/16);

      }
cleartext(1);
cleartext(3);
drawpoly(no,poly);
cleartext(0);
outtextxy(40*8,114,"PRESS ESC KEY TO EXIT");
while(1)
        {
         ch=getch();
              if(ch==0)
              continue;
              if(ch==ENTER)
              continue;
              if(ch=='\b')
              continue;
              if(ch==ESC)
              return;
              if(ch<32)
              continue;
             }
    }


cleartext(int n)
    {
     int i,j;
int color;
switch(n)
          {
           case 0:
                color=getcolor();
                setcolor(getbkcolor());
                for(i=105;i<=125;i++)
                 for(j=10;j<80*8-10;j++)
                  outtextxy(j,i,"a");
                  setcolor(color);
                  break;
            case 1:
                color=getcolor();
                setcolor(getbkcolor());
                for(i=365;i<=385;i++)
                 for(j=10;j<80*8-10;j++)
                    {
                     outtextxy(j,i,"a");
                    }
              setcolor(color);
              break;
        case 3:
               color=getcolor();
               setcolor(getbkcolor());
               for(i=395;i<=455;i++)
                for(j=10;j<=80*7;j++)
                   outtextxy(j,i,"a");
```

```
                    setcolor(color);
                    break;
                case 2:
                      cleartext(0);
                      cleartext(1);
                      break;
                 }
      }


delete()
   {
    int color,i,j;
    color=getcolor();
    setcolor(getbkcolor());
    for(i=50;i<=60;i++)
        for(j=10;j<640-10;j++)
            {
              outtextxy(j,i,"a");
            }
        setcolor(color);
      }

int getstring1(str)
char *str;
 {
  int j=0;
  char ch;
  while(1)
     {
      fflush(stdin);
      ch=getch();
    if(ch==0)
     {
      getch();
      continue;
     }
   if(ch==ENTER)
   {
    str[j]='\0';
    break;
   }
  else if(ch==ESC)
  return(ESC);
  else if(ch=='\b' && j>0)     /*If backspace is pressed*/
   {
    j--;
    continue;
   }
   else if(ch<32)   /*If control charecter is pressed*/
   continue;
   else str[j]=ch;
   j++;
   if(j==30)
     {
      str[j]='\0';
      break;
     }
  }
  return(0);
```