SHOOTING METHOD IN SOLVING BOUNDARY VALUE PROBLEM

Badradeen Adam[1] **& Mohsin H. A. Hashim**[2]

[1]Department of Mathematics, Faculty of Education, University of Khartoum, Omdurman ,Sudan
[2]Department of Applied Mathematics, Faculty of Mathematical Science, University of Khartoum, Khartoum ,Sudan

Email:bdr_uofk@yahoo.com & mohsinhashim@yahoo.com

ABSTRACT

This study is conducted to test the method of shooting on finding solution to the boundary values problems .where it is supposed that he could resolve the boundary of value for differential equation of second order, with knowing tow marginal values. Due to the importance of finding and knowledge of the initial values problems with an accurate way in physical a applications . The study has solved many physical problems for finding the boundary values problems solutions with using shooting method. As a result of what has been a pplied , the study has reached that the shooting method is the best and easiest way to resolve marginal values problems ,but there are some disadvantages when using the Newton Rapson's method of counting initial values ,and then shooting's boundary values method ,we find that the error is larger comparing with Ode-RK4 method for counting the initial values and then shooting boundary values.Finally the study has presented some recommendations and proposals with which can resolve the boundary values problems in very accurate way.

Keywords: Shooting Method, Boundary Value Problem ,Ode-RK4.

1. INTRODUCTION

In mathematics, in the field of differential equations, an initial value problem (IVP) is an ordinary differential equation (ode), which frequently occurs in mathematical models that arise in many branches of science, engineering and economics, together with specified value, call the initial condition, of the unknown function at a given point in the domain of the solution.

$$y' = f(t, y) \qquad (1.1)$$

$$y(t0) = y0 \qquad (1.2)$$

There is also another case that we consider an ordinary differential equation (ode), we require the solution on an interval [a,b] , and some conditions are given at a, and the rest at b, although more complicated situations are possible, involving three or more points .we call this a boundary value problem (BVP).

$$-y'' + r(t)y = f(t), a < t < b \qquad (1.3)$$

with the boundary conditions

$$y(a) = A, \qquad y(b) = B \qquad (1.4)$$

For analytical solutions of IVPs' and BVPs', there exist many different methods in literature [1].

Numerical solutions of such kind of problems is asubjec which can be treated separately. There are also several methods derived until now. The numerical methods for the solution of IVP of ode's are classified in two major groups: the one-step methods and multi-step methods.The one-step methods are as follows: Taylor methods Euler's Method , Runge-Kutta Methods.

The linear multistep methods are implicit Euler method, Trapezium rule method, Adams – Bash forth method,Adams-Moulton method, Predictor- Corrector methods. Similarly, for the numerical study of boundary value problems there exists some methods like, Shooting method for linear and nonlinear BVP , Finite-Difference method for linear and nonlinear BVP.

In this project, our aim is to study the Shooting Method for the numerical solutions of second order BVPs both for linear and nonlinear case.

The algorithm of these methods are presented to see how the method works .Some examples are given to show the performance and advantages. The plan of this project is as follows: In the first chapter we will give a definition of shooting method and where we use it and for which kind of problems it is used. In the second chapter we give an explanation to Linear Shooting. The third chapter is about Nonlinear Shooting. Before the conclusion part we will solve some examples in the Application chapter with using our method.

Ordinary differential equations are given either with initial conditions or with boundary conditions. The shooting method uses the same methods that were used in solving initial value problems. This is done by assuming initial values that would have been given if the ordinary differential equation were an initial value problem. The boundary value obtained is then compared with the actual boundary value. Using trial and error or some scientific approach, one tries to get as close to the boundary value as possible. Mainly, the central idea of the method is to replace the boundary value problem under consideration by an initial value problem of the form

$$-y'' + r(t)y = f(t), \quad a < t < b \qquad (1.5)$$

$$y(a) = A, y'(a) = s \qquad (1.6)$$

where t is to be chosen in such a way that y(b) = B. This can be thought of as a problem of trying to determine the angle of inclination $arc\ tans$t of a loaded gun, so that, when shot from height B at the point t= a, the bullet hits the target placed at height B at the point x = b. Hence the name ,shooting method.

Once the boundary value problem has been transformed into such an 'equivalent' initial value problem, any of the methods for the numerical solution of initial value problems can be applied to find a numerical solution.

The following theorem gives general conditions that ensure that the solution to a second-order boundary value problem exists and is unique.

Theorem 1.1(Boundary Value Problem): Suppose the function f in the boundary value problem

$$y'' = f(t, y, y'),$$

$$a \leq t \geq b,$$

$$y(a) = A,\ y(b) = B$$

is continuous on the set

D = $\{(t,y,y'): a \le t \le b, -\infty < y > \infty, -\infty < y' > \infty\}$

and that $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial y'}$ are also continuous on D. If

(i) $\frac{\partial f}{\partial y}(t, y, y') > 0$ for all $(t,y,y') \in$ D, and

(ii) A constant M exists, with

$$\left|\frac{\partial f}{\partial y}(t, y, y')\right| \le M, \quad \text{for all } (t, y, y') \in D.$$

Then the boundary value problem has a unique solution [2].

proof: See the reference [2].

Corollary 1.1(linear Boundary Value Problem)

Assume that f in Theorem (1.1) has the form

$$f(t,y,y') = p(t)y' + q(t)y + r(t)$$

$$and\ that\ f\ and\ its\ partial\ derivatives\ \frac{\partial f}{\partial y} = q(t)\ and$$

$$\frac{\partial f}{\partial y'} = p(t)\ are\ continuous\ D. If\ there\ exists\ aconstant$$

M> 0 for which p(t) and q(t) satisfy

$$q(t) > 0\ \ for\ all\ t \in [a,b]$$

And

$$|p(t)| \le M = \max_{a \le t \le b}\{|p(t)|\}$$

Then **the linear boundary value problem**

$$y'' = p(t)y' + q(t)y + r(t)\ \ with\ \ y(a) = A\ and\ y(b) = B$$

has a unique solution $y = y(t)\ \ over\ \ a \le t \le b.$

2. LINEAR SHOOTING METHOD

A linear two - point boundary value problem can be solved by forming a linear combination of the solutions to two initial value problems. The form of the IVP depends on the form of the boundary conditions . We begin with the simplest case , Dirichlet boundary conditions , in which the value of the function is given at each end of the interval. We then consider some more general boundary conditions [3].

2.1 Simple Boundary Conditions

Suppose the two - point boundary value problem is linear, i.e., of the form

$$y'' = p(x)y' + q(x)y + r(x);\ a \le t \ge b\,, \qquad (2.1)$$

with boundary conditions y(a) = A, y(b) = B. The approach is to solve the two IVPs

$$u'' = p(t)u' + q(t)u + r(t) \text{ with } u(a) = A, u'(a) = 0, (2.2)$$

$$v'' = p(t)v' + q(t)v; \quad v(a) = 0;\ v(a) = 1 \qquad (2.3)$$

If $v(b) \neq 0,$ the solution of the original two - point BVP is given

by $$y(t) = u(t) + \frac{B - u(b)v(x)}{v(b)} \qquad (2.4)$$

linear ode by finding a general solution of the homogenous equation (expressed as the ode for v) and a particular solution of the nonhomogeneous equation (ex-pressed as the ODE for u). The arbitrary constant C that would appearin the solution

$y(t) = u(t) + C\,v(t)$ is found from t he requirement that y(b) = u(b) + Cv(b) = B, which yields C = $\frac{B-u(b)}{v(b)}$.

In order to approximate the solution of the linear ode – BV

$y'' = p(x)y' + q(x)y + r(x)$, with boundary conditions y(a) = A, y(b) = B, using the linear shooting method, we must convert the problem to a system of four first order ode - IVP,

which we write as $z' = f(t,z)$. The variables z_1 and z_2 are u and u', u'' respectively, where u satisfies the ode – IVP

$u'' = p(t)u' + q(t)u + r(t)$, with initial conditions u(a) = A,

$u'(a) = 0$. The variables z_3 and z_4 are v and v', respectively,

where v satisfies the ode –IVP $v'' = p(t)v' + q(t)v$, with

initial conditions $v(a) = 0\ , v'(a) = 1.$

Define the variables , z_1 , z_2, z_3 and z_4

$$z_1 = u \quad z_2 = u' \quad z_3 = v \quad z_4 = v'$$

Define the ode

$Z'_1 = Z_2$

$Z'_2 =$ p(t)Z_2 + q(t)Z_1 + r(t)

$Z'_3 = Z_4$

Z_4 = p(t)$Z\ _4$+ q(t)Z_3

Define the initial conditions

Z_1(a) = A Z_2(a) = 0 Z_3(a) = 0 Z_4(a) = 0

The original problem has been converted into the appropriate System of the first – order ode - IVP as described above. The

components of the solutions are shown as Z_1, Z_2, Z_3, and Z_4, although a 2-dimensional array could also be used for z if desired.

Example2.1 (A simple Linear Shooting Problem)

We shall use the Runge–Kutta scheme detailed to solve the problem

$$\frac{d^2y}{dx^2} + y = 0 \qquad \textbf{(2.5)}$$

subject to the boundary conditions

$$y(0) = \frac{1}{2} and\ y\left(\frac{\pi}{3}\right) = \frac{1}{2}. \qquad (2.6)$$

we can solve this problem analytically to give

$$y(x) = \frac{1}{2}\cos(x) + \frac{1}{2\sqrt{3}} \qquad (2.7)$$

let us solve the equation subject to $y(0) = 1/2$ and $y(0) = \lambda$, where we are free to choose λ (at the moment).we then integrate from 0 to $\frac{\pi}{3} : if\ y\left(\frac{\pi}{3}\right)$is not equal to 1/2 then we can adjust the value of λ. In order to do this we use the Newton–Raphson scheme. The main routine is:

$$z(1) = y \qquad (2.8)$$

$$z(2) = y'$$

(2.9)

$$z(3) = y''$$

$$z(1,1) = \frac{1}{2} \qquad (2.10)$$

$$z(1,2) = \lambda \qquad (2.11)$$

Matlap code see page(17)&(18).

Function(1) to find lambda=0.2071. and function(3)to find err [5].

2.2 General Boundary Condition at x = b

Suppose that the linear ode

$$y'' = p(t)y' + q(t)y + r(t), \qquad (2.12)$$

has boundary conditions consisting of the value of y given at

t = a ,but the condition at t = b involves a linear combination of y(b) and $y'(b)$:

$$y(a) = A, \quad y'(b) + cy(b) = B \qquad (2.13)$$

as in the previous discussion, the approach is to solve the two IVP:

$$u'' = p(x)u' + q(x)u + r(x); \; u(a) = A, u'(a) = 0, \qquad (2.14)$$

$$v'' = p(x)v' + q(x)v; \quad v(a) = 0; \; v'(a) = 1 \qquad (2.15)$$

The linear combination, $y = u + dv$, satisfies the condition at

t=a , since y(a) = A. We now need to find d (if possible) so that y satisfies

$$y'(b) + cy(b) = B \qquad (3.16)$$

If $v'(b) + cv(b) \neq 0$, there is a unique solution, given by

$$y(t) = u(t) + \frac{B - u'(b) - cu(b)}{v'(b) + cv(b)} v(t) \qquad (2.17)$$

2.3 General Seperated Boundary Conditions

Suppose that the linear ode

$$y'' = p(t)y' + q(t)y + r(t) \qquad (2.18)$$

has mixed boundary conditions at both x = a and x = b, i.e,

$$y'(a) + c_1 y(a) = A; \quad y'(b) + c_2 y(b) = B; \qquad (2.19)$$

as in the previous discussion, the approach is to solve two IVPs, however, the appropriate forms are now

$$u'' = p(t)u' + q(t)u + r(t), u(a) = 0, u'(a) = A, \qquad (2.20)$$

$$v'' = p(t)v' + q(t)v; \; v(a) = 1; \; v'(a) = c_1: \qquad (2.21)$$

The linear combination $y = u + dv$ satisfies

$y'(a) + c_1 y(a) = A$, we need to find d (if possible) such that y

satisfies $y'(b) + c_2 y(b) = B.$

If $v'(b) + c_2 v(b) \neq 0$, there is a unique solution, given by

$$y(t) = u(t) + \frac{B - u'(b) - c_2 u(b)}{v'(b) + c_2(b)} \tag{2.22}$$

2.4 Description of Program (1)

LINSHOOT approximate the solution of the linear boundary value problem

$$u'' = p(x)\, u' + q(x)\, u + r(x)$$

$$\alpha_1 u(a) + \alpha_2 u'(a) = \alpha_3$$

$$\beta_1 \; u(b) + \beta_2 \; u'(b) = \beta_3$$

Function w = linshoot (coeff, a, b, n, alpha, beta)

using the shooting method, with the solution of all initial value problems approximated using the classical

4th-order Runge-Kutta method.

calling sequences:

w = linshoot (coeff, a, b, n, alpha, beta)

linshoot (coeff, a, b, n, alpha, beta

inputs:

coeff string containing name of m-file defining the functions p(x), q(x) and r(x) on the right-hand side of the differential equation; function should take a single input and return the values of p, q and r, in that order; i.e., the m-file header should be of the form

[p, q, r] = coeff (x)

a left endpoint of problem domain

b right endpoint of problem domain

n number of uniformly-sized steps to take

in marching

from x = a to x = c

alpha three-component vector of the coefficients

which define the boundary condition at x = a beta three-component vector of the coefficients

which define the boundary condition at x = b

output:

w vector of length n+1 containing the approximate values of the solution of the boundary value problem at the locations x = linspace (a, b, n+1)

```
u(1) = zeros ( 2, n+1 );
u(2) = zeros ( 2, n+1 );
  x = linspace ( a, b, n+1 );
  h = (b-a)/n;
if ( alpha(2) == 0 )
```

```
                         ivp(3) = 0;
                        u(1)(:,1) = [alpha(3)/alpha(1); 0];
                        u(2)(:,1) = [0;1];
                  else if ( alpha(1) == 0 )
                         ivp(3 )= 0;
                        u(1)(:,1) = [0; alpha(3)/alpha(2)];
                        u(2)(:,1) = [1;0];
                  else
                          ivp(3) = 1;
                             u(3) = zeros ( 2, n+1 );
                         u(1)(:,1) = [0;0];
                         u(2)(:,1) = [1;0];
                         u(3)(:,1) = [0;1];
                 end;
                  for i = 1 : n
                  [p q r] = feval ( coeff, x(i) );
                    k11 = h * [ 0 1; q p ] * u(1)(:,i) + h * [0; r];
                    k12 = h * [ 0 1; q p ] * u(2)(:,i);
                 if ( ivp(3) ) k13 = h * [ 0 1; q p ] * u(3)(:,i); end;
                  [p q r] = feval ( coeff, x(i) + h/2 );
                 k21 = h * [ 0 1; q p ] * ( u(1)(:,i) + 0.5*k11 ) + h * [0;r];
                 k22 = h * [ 0 1; q p ] * ( u(2)(:,i) + 0.5*k12 );
               if ( ivp(3) ) k23 = h * [ 0 1; q p ] * ( u3(:,i) + 0.5*k13 );   end;
              k31 = h * [ 0 1; q p ] * ( u(1)(:,i) + 0.5*k21 ) + h *[0;r];
               k32 = h * [ 0 1; q p ] * ( u(2)(:,i) + 0.5*k22 );
if ( ivp(3) ) k33 = h * [ 0 1; q p ] * ( u(3)(:,i) +0.5*k23 ); end;
                 [p q r] = feval ( coeff, x(i) + h );
           k41 = h * [ 0 1; q p ] * ( u(1)(:,i) + k31 ) + h * [0; r];
           k42 = h * [ 0 1; q p ] * ( u(2)(:,i) + k32 );
          if ( ivp(3) ) k43 = h * [ 0 1; q p ] * ( u(3)(:,i) + k33 ); end;
           u(1)(:,i+1) = u(1)(:,i) + ( k11 + 2*k21 + 2*k31 + k41 ) / 6;
           u(2)(:,i+1) = u(2)(:,i) + ( k12 + 2*k22 + 2*k32 + k42 ) / 6;
if ( ivp(3) ) u(3)(:,i+1) = u(3)(:,i) + ( k13 + 2*k23 + 2*k33 +k4)/6;
end;
if ( ~ivp(3) )
   if ( beta(2) == 0 )
   c = ( beta(3)/beta(1) – u(1)(1,n+1) ) / u(2)(1,n+1);
   elseif ( beta(1) == 0 )
```

```
     c = ( beta(3)/beta(2) – u(1)(2,n+1) ) / u(2)(2,n+1);
 else
  c = ( beta(3) - beta(1) * u(1)(1,n+1) - beta(2) *u(1)(2,n+1) ) / (      beta(1) * u(2)(1,n+1) + beta(2) * u(2)(2,n+1) );
   end;
   w = u(1) + c * u(2);
   else
   if ( beta(2) == 0 )
    denom = alpha(1) * u(3)(1,n+1) - alpha(2) * u(2)(1,n+1);
     rhs = beta(3) / beta(1) – u(1)(1,n+1);
     c1 = ( alpha(3) * u(3)(1,n+1) - alpha(2) * rhs ) / denom;
     c2 = ( alpha(1) * rhs - alpha(3) * u(2)(1,n+1) ) / denom;
     elseif ( beta(1) == 0 )
      denom = alpha(1) * u(3)(2,n+1) - alpha(2) * u(2)(2,n+1);
       rhs = beta(3) / beta(2) – u(1)(2,n+1);
     c1 = ( alpha(3) * u(3)(2,n+1) - alpha(2) * rhs ) / denom;
 c2 = ( alpha(1) * rhs - alpha(3) * u(2)(2,n+1) ) / denom;
    else
     a1 = beta(1) * u(2)(1,n+1) + beta(2) * u(2)(2,n+1);
     a2 = beta(1) * u(3)(1,n+1) + beta(2) * u(3)(2,n+1);
    rhs = beta(3) - beta(1) * u(1)(1,n+1) - beta(2) * u(1)(2,n+1);
      denom = alpha(1) * a2 - alpha(2) * a1;
      c1 = ( alpha(3) * a2 - alpha(2) * rhs ) / denom;
      c2 = ( alpha(1) * rhs - alpha(3) * a1 ) / denom;
     end;
     w = u(1) + c1 * u(2) + c2 * u(3);
end;
```

2.5 Description of Program (2)

The boundary value problems constructed here require information at the present time (t = a) and a future time (t = b). However, the time-stepping schemes developed Previously only require information about the starting time t = a. Some effort is then needed to reconcile the time-stepping schemes with the boundary value problems presented here.

We begin by reconsidering the generic boundary value problem

$$y'' = f(t, y, y') \qquad (2.23)$$

on $t \in [a, b]$ with the boundary conditions

$$y(a) = \propto \qquad (2.24a)$$

$$y(b) = \beta \qquad (2.24b)$$

The stepping schemes considered thus far for second order differential equations involve a choice of the initial conditions $y\ (a)$ and $y'(a)$ We can still approach the boundar value problem from this framework by choosing the "initial" conditions

$$y(a) = \propto \qquad (2.25a)$$

$$\frac{dy(a)}{dt} = A \qquad (2.25b)$$

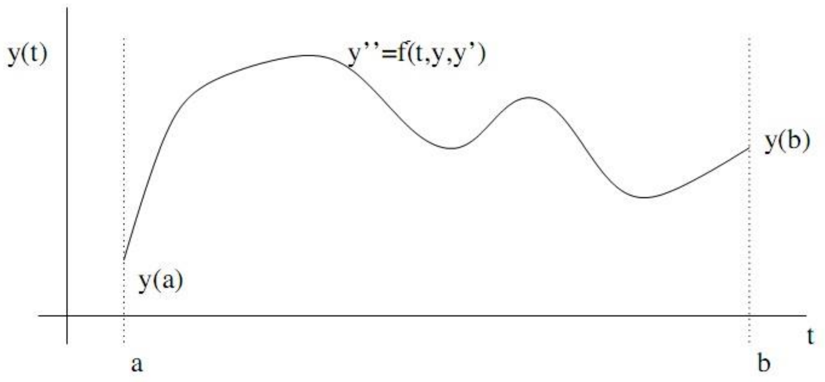


Figure 1: Graphical depiction of the structure of a typical solution to a boundary value problem with constraints at $t{=}a$ *and* $t{=}b$

where the constant A is chosen so that as we advance the solution to $t = b$ we find $y(b) = \beta$. The shooting method gives an iterative procedure with which we can determine this constant A. Figure 2 illustrates the solution of the boundary value problem given two distinct values of A. In this case, the value of A = A gives a value for the initial slope which is too low to satisfy the boundary conditions (2.25),whereas the value of A = A is too large to satisfy (2.24).

2.6 Computational Algorithm

The above example demonstrates that adjusting the value of A in (2.25b) canlead to a solution which satisfies (2.24b). We can solve this using a self consistent algorithm to search for the appropriate value of A which satisfies the original problem. The basic algorithm is as follows:

1. Solve the differential equation using a time stepping scheme with the initial conditions $y(a) = \alpha$ and $y(a) = A$.

2. Evaluate the solution $y(b)$at $t = b$ and compare this value with the target value of $y(b) = \beta$.

3. Adjust the value of A (either bigger or smaller) until a desired level of tolerance and accuracy is achieved. A bisection method for determining values of A, for instance, may be appropriate.

4.Once the specified accuracy has been achieved the numerical solution is complete and is accurate to the --level of the tolerance chosen and the discretization scheme used in the time-stepping.

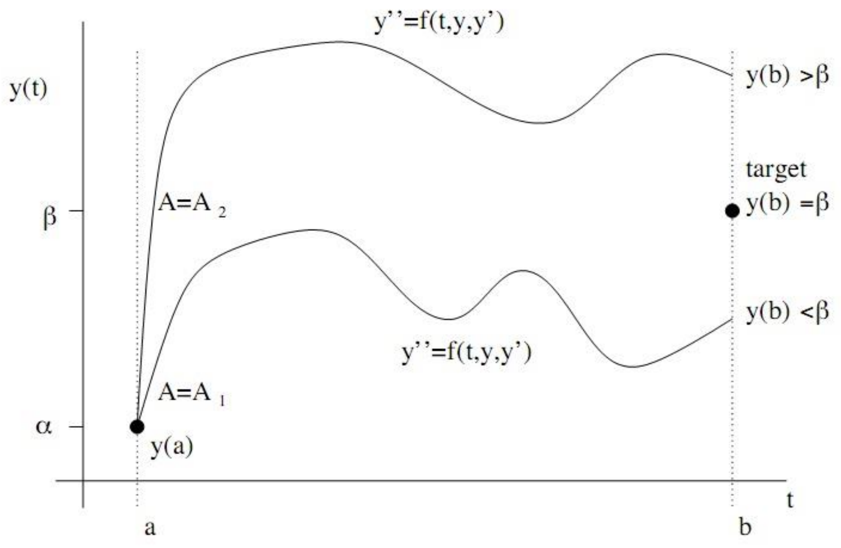


Figure 2: Solutions to the boundary value problem with $y(a) = \alpha$ *and* $y(a) = A$*. Here, two values of* A *are used to illustrate the solution behavior and its lack of matching the correct boundary value* $y(b) = \beta$*. However, the two solutions suggest that a bisection scheme could be used to find the correct solution and value of* A*.*

We illustrate graphically a bisection process in Fig. 2 and show the convergence of the method to the numerical solution which satisfies the originaloundary conditions y(a) = α and y(b) = β. This process can occur quickly so hat convergence is achieved in a relatively low amount of iterations provid differential equation is well behaved.

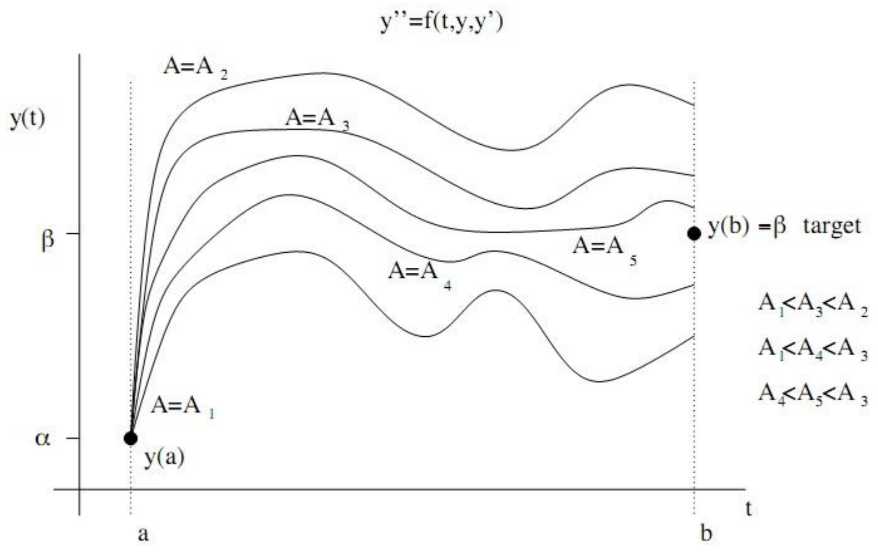


Figure 3:Graphical illustration of the shooting process which uses a bisection scheme to converge to the appropriate value of A for which y(b) = β.

3. NONLINEAR SHOOTING

We now consider the shooting method for nonlinear problems of the form $y'' = f(x, y, y')$ on the interval [a; b]. We assum that y(a) is given and that some condition on the solution is also given at x = b. The idea is the same as for linear problems,namely, to solve the appropriate initial - value problems and use the results to find a solution to the nonlinear problem.However, for a nonlinear BVP, we have an iterative procedure rather than a simple formula for combining the solutions of two IVPs. In both the linear and the nonlinear case, we need to find a zero of the function representing the error that is, the amount by which the solution to the IVP fails to satisfy the boundary condition at x = b. We assume the continuity of f, f_x and f_y on an appropriate domain, to ensure that the initial - value problems have unique solutions.

We begin by solving the initial value problem

$$u'' = f(x, u, u');\ u(a) = A,\ u'(a) = t, \qquad (4.1)$$

for some particular value of t. We then find the error associated with this solution; that is, we evaluate the boundary condition at $x = b$ using $u(b)$ and $u'(b)$. Unless it happens that $u(x)$ satisfies the boundary condition at $x = b$, we take a different initial value for u0(a) and solve the resulting IVP. Thus,the error (the amount by which our shot misses its mark) is a function of our choice for the initial slope. We denote this function as $m(t)$. There are two different approaches that we use. The first approach we consider uses the secant method to find the zero of the error function. This allows us to treat a fairly general boundary condition at $x = b$. The second approach is based on Newton's method.

3.1 Nonlinear Shooting Based on the Secant Method:

To solve a nonlinear BVP of the form

$y'' = f(x,y,y'), y(a) = A, h(y(b,y'(b)) = 0$ (4.2)

we may use an iterative process based on the secant method. We need to find a value of t, the initial slope, so that solving eq. (4.1 gives a solution that is within a specified tolerance of the boundary condition at x = b. We begin by solving the equation *with* $u'(a) = t(1) = 0$; the corresponding error is $m(1)$. Unless the absolute value of m(1) is less than the tolerance, we continue by solving eq. (4.1) *with* $u'(a) = t(2) = 1$. *If* this solution does not happen to stisfy the boundary condition (at $x = b$) either, we continue by updating our initial slopes according to the secant rule (until our stopping condition is satisfied), i.e.,

$$t(i) = t(i-1) - \frac{t(i-1) - t(i-2)}{m(i-1) - m(i-2)} \qquad (4.3)$$

3.2 Nonlinear Shooting using Newton's Method

We next illustrate how Newton's method can be used to find the value *of* $y'(a) = t$ *in the* initial value problem for nonlinear shooting.We consider the following nonlinear BVP with simple boundary conditions *at* $x = a$ *and* $x = b$:

$y'' = f(x',y,y'), \quad y(a) = A, y(b) = B, \qquad (3.4)$

We begin by solving the initial - value problem

$u'' = f(x,y,u'); u(a) = A; u'(a) = t; \qquad (3.5)$

The error in this solution is the amount by which $y(b)$ misses the
desired value , B. For different choices of t, we get different errors ,so we define

$m(t) = u(b,t) - B \qquad (3.6)$

we need to find t such that $m(t) = 0$ (or m(t) is as close to zero as we wish to continue the process). In the previous section, we found a sequence of t using linear interpolation between the two previous solution; in order to use Newton's method, we need to have the derivative of the function whose zeor is required, namely, $m(t)$. Although we do not have an explicit formula for m(t), we can construct an additional differential equation whose solution allows us to update t at each iteration [4].

$u'' = f(x,u,u'); u(a) = A; u'(a) = t(k-1) \qquad (3.7)$

$v'' = vf_u(x,u,u') + v'^{f_u}(x,u,u');$

$v(a) = 0; v'(a) = 1 \qquad (3.8)$

Example(3.1):(non linear shooting problem)

This problem comes from fluid dynamics and its solution provides a description of the flow profile within a boundary layer on a flat plate. We shall not dwell on its derivation, since it is far beyond the scope of this text, but we are required to solve

$$\frac{d^3y}{d\eta^3} + f\frac{d^2y}{d\eta^2} = 0$$

subject to the boundary conditions $f(0) = f'(0) = 0$ and $\lim_{\eta\to\infty} f'(\eta) = 1.$

In this case we have a third-order equation and as such we shall introduce

$$z_1(0) = 0, \ z_2(0) = 0 \text{ and } z_3(0) = \lambda.$$

as in the previous example we are short of a boundary condition at one end, so we solve the problem using the initial conditions

$$z_1(0) = 0, \quad z_2(0) = 0 \text{ and } z_3(0) = \lambda.$$

We then integrate towards infinity (in fact in this case a value of 10 is fine for infinity, even if we integrated further nothing would change). The discrepancy between the value of f'at this point and unity is used to iterate on the value of λ. The Matlab codes (see pages(20)&(21)). The actual value of λ is approximately 0.4689. (see page(22).

4. A PPLICATIONS

<u>Example(4.1):</u>

Solve for $y(t),$ altitude of rocket, given

- $y'' = -g$ (the differential equation)
- $g = 9.8\ [m/sec\text{^}2]$ (acceleration due to gravity)
- $y(0) = 0$ (launch from ground)
- $y(5) = 40$ (fireworks explode after 5 seconds, we want them $40\ m$ off ground)
- In particular, what should launch velocity $y'(0)\ be$?

This can be solved analytically; exact motion is quadratic in t, final answer is y'(0)=32.5 [m/sec].

Matlab code for the shooting method:

(a)using Euler'method:

(b)Function (1) Listing of rocket.m

```
Function   y = rocket (dy0)
% return altitude at t=te (y(te)) as a function of initial velocity (y'(0)=dy0)
```

```
 global   h  te
 [tv ,yv] = euler2(h,0,te,0,dy0);
 plot(tv,yv,'o-', 'LineWidth',2);
 % invariant: tv(te/h+1)==te
  y = yv(te/h+1);              % returns y at t=te
return;
function (2):euler2.m
function [tv,yv] = euler2(h,t0,tmax,y0,dy0)
    % use Euler's method to solve 2nd order ode y"=-g+a*y^2
    % return tvec and yvec sampled at t=(t0:h:tmax) as col. vectors
    % y(t0)=y0, y'(t0)=dy0
     global a;                                 % coeff of nonlinear acceleration
    g = 9.8;      % accel. of gravity, [m/sec^2]
    y = y0;       % position
    dy = dy0;                    % velocity
    tv = [t0];
    yv = [y0];
    for t = t0:h:tmax
                  y = y+dy*h; % this and following line are Euler's method
                  dy = dy+(-g+a*y^2)*h;
                  tv = [tv; t+h];
                  yv = [yv; y];
end
    return;
function (3):Listing of shoot.m
function ret = shoot(hh)
% shooting method for fireworks problem
    global a te ye h;
    a = 0;        % coeff of nonlinear acceleration
    te = 5;       % end time [sec]
    ye = 40;      % end height [m]
    h = hh;
    clf;
    hold on;
    for dy = 20:10:50
                  y = rocket(dy);
                  text(te+.2,y,sprintf('y\047(0)=%g',dy), 'FontSize',15);
    end
```

```
   % now use root finding to find correct initial velocity  dy = secant(20,30,1e-4);
   % draw last curve
   y = rocket(dy);
   text(te+.2,y,sprintf('y\047(0)=%g',dy), 'Color','r', 'FontSize',15);
   set(gca, 'FontSize', 16);                  % for tick marks
   line([te te],[-40 140], 'Color','k');
  line([te te],[ye ye], 'Color','r', 'Marker','o', 'LineWidth',3);
   xlabel('t', 'FontSize',20);
   ylabel('y', 'FontSize',20);
   title(sprintf('Shooting Method on y\047\047=-g'), 'FontSize',20);
   return;
```

<u>function (4): secant.m</u>

```
function x = secant(x1,x2,tol)
 % secant method for one-dimensional root finding
   global ye;
   y1 = rocket(x1)-ye;
   y2 = rocket(x2)-ye;
   while abs(x2-x1)>tol
             disp(sprintf('(%g,%g) (%g,%g)', x1, y1, x2, y2));
             x3 = x2-y2*(x2-x1)/(y2-y1);
             y3 = rocket(x3)-ye;
             x1 = x2;
             y1 = y2;
             x2 = x3;
             y2 = y3;
   end
   x = x2;
   return;
```

at h=0.5 (h too big) see figure(3),page().

h=0.1 smaller h gives more accurate results. But note that the y'(0) that secant method solves for, in red, is still not correct (not 32.5), because of errors of our IVP solution(see figure(4),page()).

Example (4.2):

let’s consider a BVP consisting of the

second-order differential equation

$$x''(t) = 2x^2(t) + 4tx(t)x'(t) \qquad (4.1)$$

$$\text{with } x(0) = \frac{1}{4} \quad , x(1) = \frac{1}{3}; \qquad (4.2)$$

The solution $x(t)$ and its derivative $x(t)$ are known as

$$x(t) = \frac{1}{4-t^2} \quad \text{and} \quad x'(t) = \frac{2t}{(4-t^2)^2} = 2t\ x^2(t) \ (4.3)$$

Note that this second-order differential equation can be written in the form of state equation as

Let $x_1(t) = x(t)$

$$x_2(t) = x'(t) \qquad (4.4)$$

$$x_1'(t) = x_2(t)$$

$$x'_2(t) = 2x_1{}^2(t)x_1(t)x_2(t)$$

$$\begin{bmatrix} x'_1(t) \\ x'_2(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ 2x_1{}^2(t) + 4t\ x_1(t)\ x_2(t) \end{bmatrix} \quad \text{with}$$

(4.5)

$$\begin{bmatrix} x_1(0) \\ x_2(1) \end{bmatrix} = \begin{bmatrix} x_0 = 1/4 \\ x_f = 1/3 \end{bmatrix}$$

In order to apply the shooting method, we set the initial guess of $x_2(0) = x'(0)$ to

$$dx0[1] = x_2(0) = \frac{x_f - x_0}{t_f - t_0}$$

and solve the state equation with the initial condition

[$x_1(0)\ x_2(0)$ = dx0[1]].

Then, depending on the sign of the difference e(1) between

the final value x_1 (1) of the solution and the target final value x_f , we make the next guess *dx*0[2] larger/smaller than the initial guess *dx*0[1] and solve the state equation again with the initial condition [$x_1(0)$ *dx*0[2]]. We can start up the secant method with the two initial values *dx*0[1] and *dx*0[2] and repeat the iteration until the difference (error) e(k) becomes sufficiently small. For this job, we compose the MATLAB program "do_shoot.m", which uses the routine "bvp2_shoot()" to get the numerical solution and compares it with the true analytical solution.

<u>function(1) bvp_shoot:</u>

```
function [t,x] = bvp2_shoot(f,t0,tf,x0,xf,N,tol,kmax)
 % To solve BVP2: [x1,x2]' = f(t,x1,x2) with x1(t0) = x0,
x1(tf) = xf
if nargin < 8,    kmax = 10;  end
if  nargin < 7,      tol  = 1e-8;  end
if  nargin < 6,     N = 100;     end
dx0(1) = (xf - x0)/(tf-t0);                  %  the initial guess of x’(t0)
[t,x] = ode_RK4(f,[t0 tf]  ,[x0 dx0(1)],N);      % start up with RK4
plot(t,x(:,1)),
hold on
e(1) = x(end,1) - xf;              % x(tf) - xf: the 1st mismatching (deviation)
dx0(2) = dx0(1) - 0.1*sign(e(1));
for k = 2: kmax-1
[t,x] = ode_RK4(f,[t0 tf],[x0 dx0(k)],N);
plot(t,x(:,1))
% difference between the resulting final value and the target onee(k) = x(end,1) - xf;  % x(tf)- xf
ddx = dx0(k) - dx0(k - 1);        % difference between successive derivatives
if   abs(e(k))< tol | abs(ddx)< tol, break;   end
deddx = (e(k) – e(k – 1))/ddx;      % the gradient of mismatching error
dx0(k + 1) = dx0(k) – e(k)/deddx;      %move by secant method
end
```

<u>function(2)do_shoot:</u>

```
% do_shoot to solve BVP2 by the shooting method
t0 = 0; tf = 1;  x0 = 1/4;   xf = 1/3;
 %initial/final times and positions
N = 100; tol = 1e-8; kmax = 10;
[t, x] = bvp2_shoot(’df41’ ,t0,tf,x0,xf,N,tol,kmax);
xo = 1./(4 - t.*t); err = norm(x(:,1) - xo)/(N + 1)
plot(t, x(:,1),’b’, t, xo,’r’)     %compare with true solution (4.4)
```

<u>function(3)d41:</u>

```
function  dx = df41(t,x)     %eq.(4.4)
dx(1) = x(2); dx(2) = (2*x(1) + 4*t*x(2))*x(1);
```

Note:

the solution of BVP obtained by using the shooting method,see page(23).

5. OVERALL CONCLUSIONS

In this project, shooting method is presented for the numerical solution of second - order BVPs. Both linear and nonlinear versions of shooting method are described and the procedure of these methods are presented to show they are applied on a given problem and their performance. Some representative examples are presented. In the last part of this work, a LNEAR BVP is solved by using linear second - order shooting method as an application. the difficulties that I faced in this project I could not find a solution algorithm for the general boundary values of problems. I recommend using an algorithm shooting method in solving boundary value problem in practical applications, because then the error is very small compared with other algorithms and gives the same solution.

Function (1) lambda

```
%
%
global x z
lambda = 1; % Initial guess
delta = 1e-2;
for its = 1:40
   f_lambda = int_eqn(lambda);
   if abs(f_lambda)<1e-8
      break
   end
   f_lambda_del = int_eqn(lambda+delta);
   lambda = lambda ...
       - f_lambda*delta/(f_lambda_del-f_lambda);
end
exact = 1/2*cos(x)+1/(2*sqrt(3))*sin(x);
```

Function (2) func1

```
function [out] = func1(x,in)
out(1) = in(2);
out(2) = -in(1);
```

Function (3) int_eqn.m

```
%
% int_eqn.m
function [err] = int_eqn(lambda)
global x z
delta_x = (pi/3)/100;
x = 0.0:delta_x:pi/3;
N = length(x);
z = zeros(N,2);
z(1,1) = 1/2;
z(1,2) = lambda;
for j=1:(N-1)
    k1 = delta_x*func1(x(j),z(j,:));
    k2 = delta_x*func1(x(j)+delta_x,z(j,:)+k1);
    z(j+1,:) = z(j,:) + 1/2*(k1+k2);
end
err = z(N,1) - 1/2;
```

Function(4) nonlambda:

```
%
%
global x z
lambda = 0.5; % Initial guess
delta = 1e-2;
for its = 1:40
   f_lambda = int_blas(lambda);
   if abs(f_lambda)<1e-8
      break
   end
   f_lambda_del = int_blas(lambda+delta);
   lambda = lambda ...
        - f_lambda*delta/(f_lambda_del-f_lambda);
end
plot(z(:,2),x)
xlabel('Flow velocity')
ylabel('Distance from the wall')
axis([-0.5 1.5 0 10])
```

Function (5)

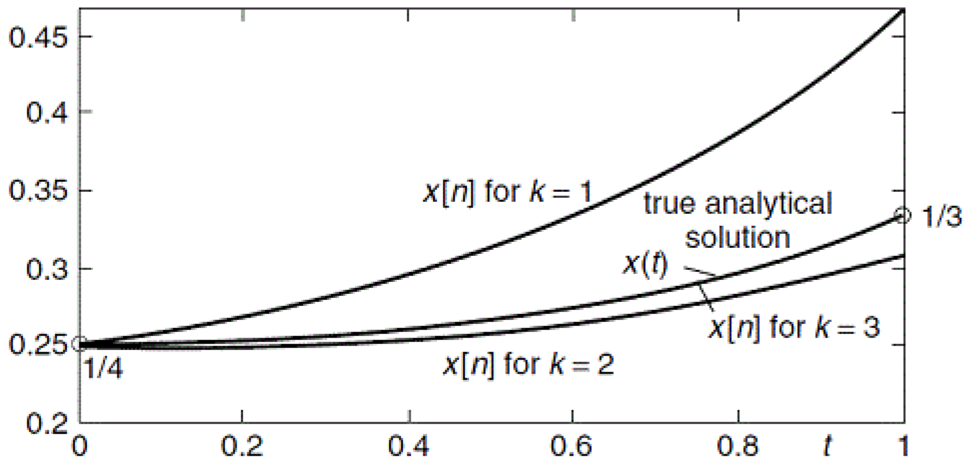
```
function [out] = funcb(x,in)
out(1) = in(2);
out(2) = in(3);
out(3) = -in(1)*in(3);
```

Function (6)

```
%
% int_blas.m

function [err] = int_blas(lambda)
global x z
delta_x = 0.1;
x = 0.0:delta_x:10;
N = length(x);
z = zeros(N,3);
z(1,1) = 0;
z(1,2) = 0;
z(1,3) = lambda;


for j=1:(N-1)
    k1 = delta_x*funcb(x(j),z(j,:));
    k2 = delta_x*funcb(x(j)+delta_x,z(j,:)+k1);
    z(j+1,:) = z(j,:) + 1/2*(k1+k2);
end
err = z(N,2) - 1;
```



The solution of a BVP obtained by using the shooting method.

Figure(4)

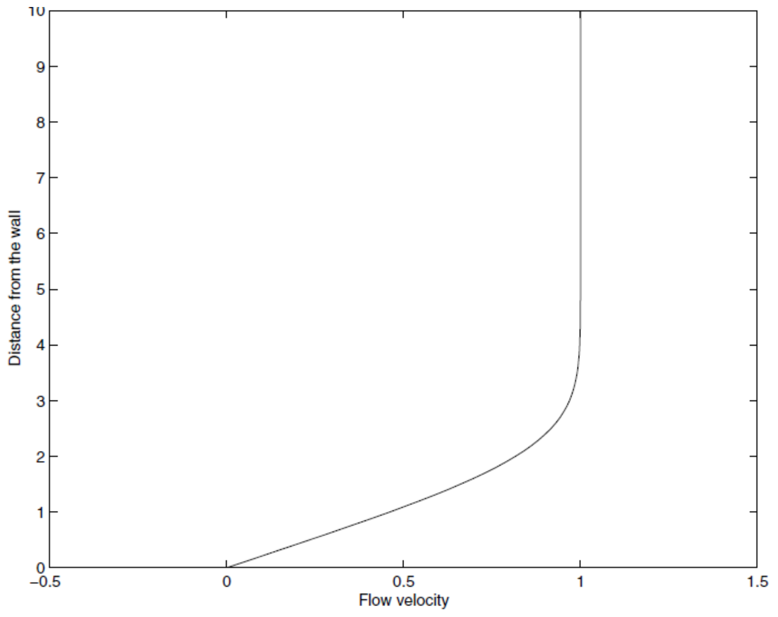
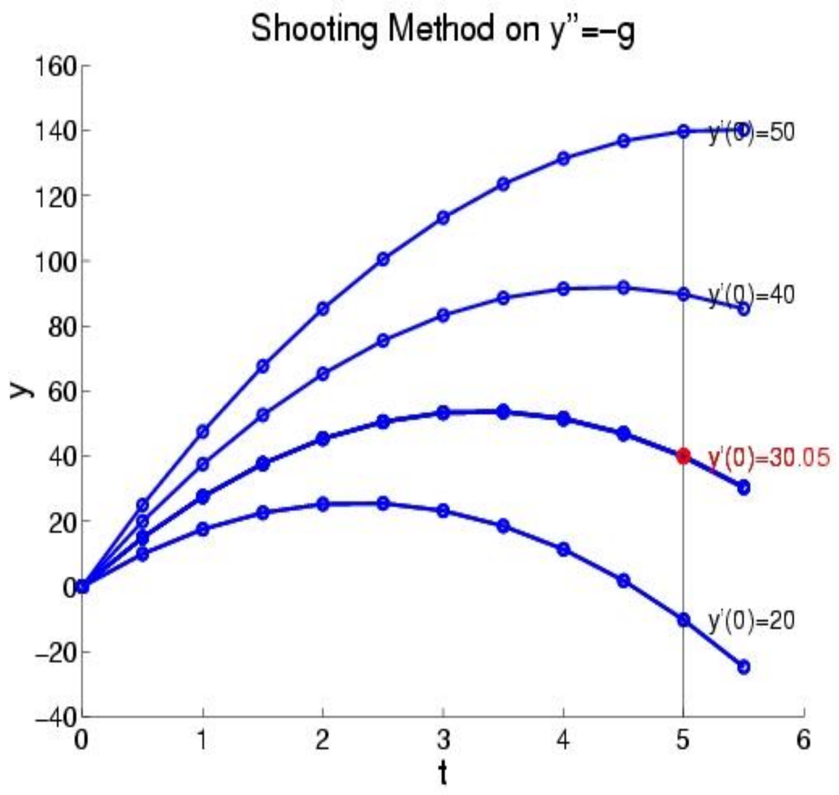
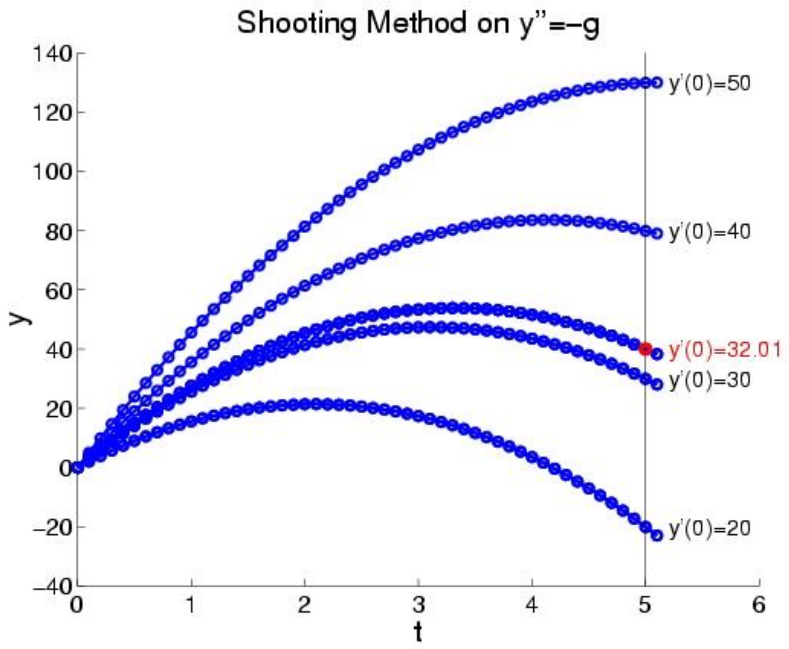


Figure (5) The actual value of λ is approximately 0.4689



Figure(6): At h=0.5 (h too big)



Figure(7): h=0.1 - smaller h gives more accurate results. But note that the y'(0) that secant method solves for, in red, is still not correct (not 32.5), because of errors of our IVP solution.

6. ACKNOWLEDGEMENTS

This paper is extracted from a Msc. Thesis by Badradeen Adam (supervised by Mohsin H. A. Hashim) at Khartoum University in 2011

7. REFERENCES

[1]. B. Rai, D. P. Choudhury, H.I. Freedman, A Course in Ordinary Differiential Equations, Alpha Service, (2002) .

[2]. Richard L. Burden, J. Douglas Faires, Numerical Analysis, Brooks , (2004) .

[3]. Endre Suli , david Meyers, An Introduction to Numerical Analysis ,Cambridg, (2003).

[4]. Lauren V. Fausett, Numerical Methods, Prentice Hall, (2003).

[5]. S.R. Otto. Verlag London, Introduction To programming and Numarical Methods in Matlap,(2005).

[6]. J.P. Denier, Verlag London ,Introduction To programming an Numarical Methods in Matlap, Adelaide,(2005).

[7]. Richard L. Burden. Cambridge University Press, Numerical Analysis,2003.