A report on solution of Inviscid Burgers Equation using Newton-Raphson Method

*Arjun Darbha*

Table of contents

# Introduction 3

# Theory of numerical techniques 4

## Newton Raphson Method 4

## Bisection Method 6

# Solution scheme

## Algorithm 7

# Results 9

# References 11

# Appendix 12

## Newton Raphson function 12

## Inviscid Burgers Equation-Solution 12

## Non-linear function (test) 13

## Derivative of non-linear function (dtest) 13

1 Introduction

The general form of Burgers equation which is a quasi linear second order partial differential equation is represented by Eq.1*.*[1]

Eq. 1

But the problem at hand is to solve an inviscid Burgers equation, which means that the viscosity term in Eq.1 tends to zero. Consequently, right hand side of equation 1 tends to zero.

Eq. 2

As it is evident that *u = u(x,t)* and also, the second term is a product of u and a derivative of u; this equation is nonlinear and its analytical solution given by *Eq.3* is also non-linear.

**Eq. 3**

This means that this equation cannot be solved analytically and one has to resort to numerical techniques for solving non-linear equations. Some of the techniques investigated for this purpose are

* Newton-Raphson Method
* Bisection Method

It is worth noting at this point that there are a handful of techniques one can employ while solving this non-linear equation, the others being Secant method, Gradient Descent method, Root Finding Algorithm etc. The analysis presented in this report is limited to the two methods mentioned as bullet points above. Moreover, only one of these, namely, Newton-Raphson method is utilized for formulating a solution to this problem.[2]

2 Theory on numerical techniques

It is important to realize that all the numerical techniques assume that any function *f(x,)* which needs to be solved, is continuous and differentiable for x Ԑ **R.** To start solving a nonlinear equation one has to formulate a function f(x) on which numerical technique is to be applied and solve for the roots of *f(x)* = 0**.** For example, consider a non-linear equation *x\*cos(x) = x+1*; to solve this equation a function *f(x) = x\*cos(x)-x-1=0* has to be solved numerically for finding its roots. A brief description of some numerical techniques is presented below.

***2.1 Newton Raphson Method***

Assuming f(x) is continuous and differentiable for x Ԑ **R**, derivative at a particular point xi on *f(x)* represents the tangent at that point. Let this tangent cross x-axis at a point xi+1. Now, a derivative is obtained at this point xi+1 on f(x) and a new point xi+2 obtained where this tangent line crosses x-axis. These set of operations are performed until the solution converges and an acceptable solution is obtained.[3]



Fig. 1 Geometrical interpretation of Newton Raphson method[3]

It is evident from the Fig. 1 that x approaches the root of *f(x)* progressively, as number of iterations increase.

From the figure it is clear that,

*fI(x) = f(x) – 0 / xi – xi+1*  **Eq. 4**

which means that

*xi+1 = xi – f(xi)/fI(xi)* **Eq. 5**

Therefore, the value of x for the next iteration depends on its current value, the value of the function and its derivative at xi.

***Drawbacks***

* Divergence at inflection points: As the solution obtained is based on the initial guess, if this guess or the root of f(x) is close to the inflection point, the solution might diverge from the root initially and then converge back to the solution. Hence the process of root finding tends to become computationally intensive at inflection points (if any)
* Division by zero: It is clear from Eqn.5 that the value of the derivative can be zero for some values of x. This is overcome by adding a very small value to the denominator but this disrupts accuracy of solution
* Root jumping: The function f(x) may have more than one root. If a particular root is desired, an initial guess closer to this guess might be used to converge at a point closer to the root. However, it is not always true and the root might converge at a completely different abscissa which is not even close to the desired root

***2.2 Bisection method***

Bisection method is an iterative root-bracketing method where one has to guess two values a and b, where a, b Ԑ R, such that *f(a)f(b)<0*. This means that *f(a)* and *f(b)* have opposite signs. Then, this method assumes that *f(x)* has a root enclosed in (a,b). Subsequently, a point *c,* where *c=(a+b)/2* is obtained and the value of function at this point *f(c)* is determined. The bracket for next iteration depends on the sign of *f(c)*. Suppose, *f(c)* and *f(a)* have a similar sign. This means that *a* is replaced by *c* for the next iteration ensuring that the bracketed interval has opposite signs and the bracket size is reduced by half. Hence, this technique assumes that the bracket interval reduces progressively so that the value of *c* can be accepted as a solution within desired tolerance limits. The same idea is graphically presented in Fig.2.

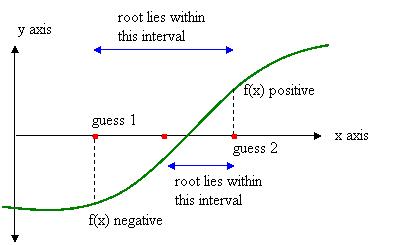


Fig. 2 Graphical representation of Bisection Method[4]

***Drawbacks***

* It can be difficult to find the appropriate bracket interval for some complex functions
* If the bracket interval contains more than two roots, the solution might not converge to the desired root
* It involves a lot of computation when compared to the Newton-Raphson method and might need a lot of computation time for finding out the root

3 Solution Scheme

It is evident from Eq.2 that inviscid Burgers equation is a hyperbolic equation which means that it is a marching type problem and a set of initial conditions are required to solve this problem. Initial conditions are given as

*u(x,0) = f(x)*

Where

*f(x) = u0 + A\*sin(x) for 0 ≤ x ≤ 2π*

*= u0 otherwise*

According to Eq.3 the analytical solution for this problem is

*u = u0 + A\*sin(x-ut)* **Eq. 6**

It is clear from this equation that at a particular instant of space(x) and time (t) this equation is non-linear in nature. Hence, one of the methods described above can be used to solve this equation. This report explicitly describes algorithm, code and results obtained by solving Eq.6 using Newton-Raphson method.

***3.1 Algorithm***

**3.1.1 Newton-Raphson function**

* Input – Function(*f(x)*), derivative of the function(*f(x)*), Initial guess for velocity(*u*), x, t
* Output – Velocity(u) at each *x* and *t*

Here it has to be realized that derivative of a function is an input parameter to the Newton Raphson function because computed derivatives involve numerical approximations which disrupt the genuineness of the solutions obtained

* Set the initial parameters like tolerance, error, iteration number etc.
* Start running the iterations in a loop, while error > desired tolerance
* Each iteration computes a new value of *xi+1* according to Eq. 4 and an error is determined by calculating the difference between newly derived *x* and old *x* until the desired tolerance
* Iteration number is also calculated at the end of each iteration and if the number of iterations is beyond 500, the control breaks out of the loop to enter a separate loop with a different initial guess value. This step ensures that there is no infinite loop situation
* When the loop stops, function returns desired velocity *u* at specific x and t

**3.1.2 Inviscid Burgers Equation**

* Set the initial parameters for Newton-Raphson function like error and tolerance
* Set the initial parameters for solving Burgers Equation like declaring values of *A(=1)* and *u0(=1)* in Eq. 6
* Declare and discretize the region of space(x) and time(t)
* The idea is to obtain the solution *u* in terms of a matrix *mxn,* where m = number of time steps and n = number of space steps. Hence, initialize a matrix umxn with zeros
* Solve for all the values of x at each time step. This requires executing two loops with the first loop running over each row and the second, running over each element of the column
* Within the loop, the first row is generated by using the set of initial conditions specified in the problem
* The second row needs to be solved using Newton-Raphson (NR) function described above. Eq.6 needs to be solved for *f(u) = 0* where *f(u) = u -u0 - A\*sin(x-ut)* and *fI(u) = 1 + cos(x-ut)*
* The most important part of defining input values to NR function is to set the value of initial guess. There are four possibilities for accomplishing this
  + A constant initial guess can be used. This value can be obtained by trial and error
  + The corresponding value of u(x) at first time step can be used
  + The corresponding value of u(x) at the last time step can be used
  + The value of previous space step of the same time step can be used
* All the four possibilities have been employed to formulate a solution and the last option seems to solve the equation by generating continuous values of u as time progresses
* A similar approach can be employed for solving Burgers equation by creating a function for Bisection method rather than Newton Raphson method

4 Results

The following plots have been generated in MATLAB by employing the solution scheme described in prior sections

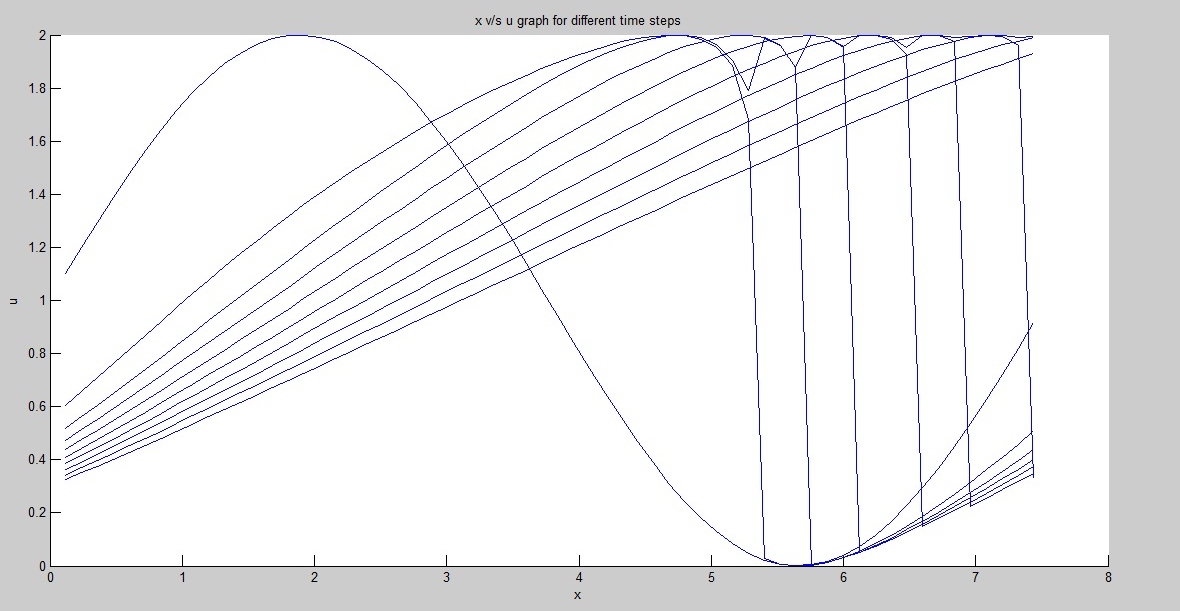


Fig.3 Plot denoting x v/s u as time progresses

It has to be noted that the problem has been solved only in the domain of interest *0≤x≤2π.* At every other location *u* has a straight forward solution.

From Eq.3

*u = f(x-ut) = u0* **Eq. 7**

for all x out of the domain of interest. This means that it has a value of *uo*(=*1m/s*)in this region

Also, it can be seen that *u* starts out as a sinusoidal wave and degenerates onto itself as a straight line as time progresses. This solution can be accepted valid as Inviscid Burgers equation is quite similar to transport equation Eq.7

**Eq. 8**

This describes a wave equation travelling with a constant velocity a. Hence the wave progresses without any degeneration in its form.

However, Eq.2 is non-linear and each point in space is travelling with a velocity u which is a function of a sinusoidal wave Eq.6. As each point encounters a different velocity with increasing time, the wave structure degenerates and the wave collapses onto itself. This fact can be appreciated by analyzing Fig.4 which is a 3-d plot; velocity (z-axis), space (x-axis), time (y-axis)

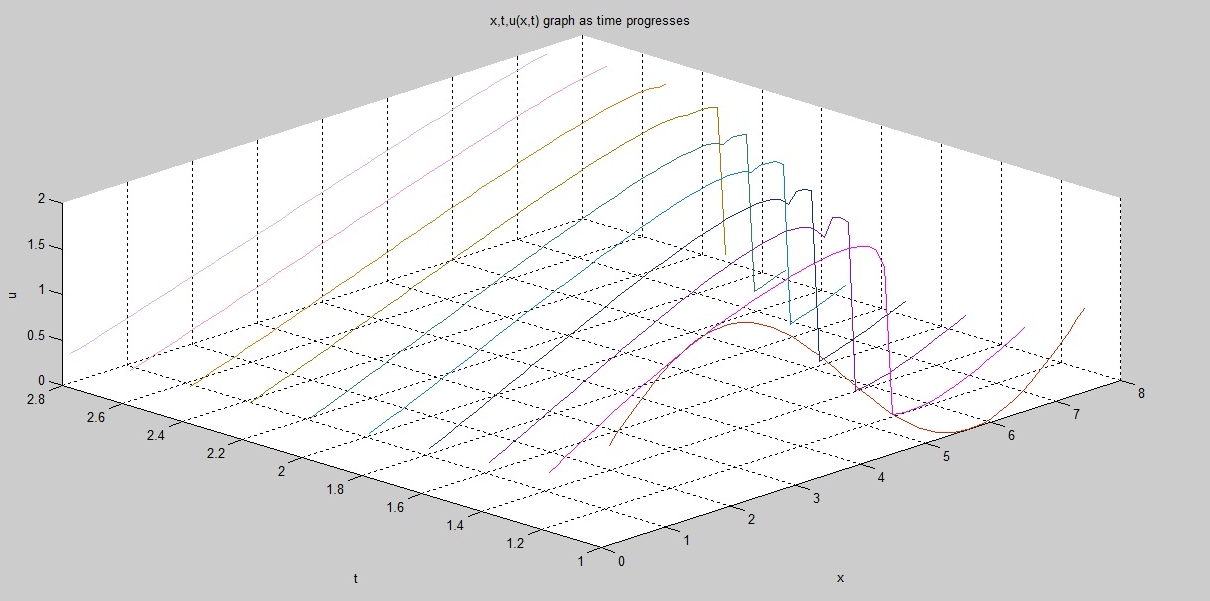


Fig.4 Graphical representation of the progression of sinusoidal wave with time

5 References

1. http://www.eng.fsu.edu/~dommelen/pdes/style\_a/burgers.html
2. http://www.cse.uiuc.edu/heath/scicomp/notes/chap05.pdf
3. http://mathforcollege.com/nm/mws/gen/03nle/mws\_gen\_nle\_txt\_newton.pdf
4. http://cse.unl.edu/~sincovec/Matlab/Lesson%2010/CS211%20Lesson%2010%20-%20Program%20Design.htm

6 Appendix

This section includes the MATLAB code utilized to solve the inviscid Burgers equation

***6.1 Newton-Raphson function***

function[res] = newton\_raphson1(func,dfunc,oldx,x,t)

tol = 1e-5;

error =1;

iter = 1;

while(error > tol && iter <500)

y = feval(func,oldx,x,t);

ydash = feval(dfunc,oldx,x,t);

newx = oldx - y/ydash;

error = abs(newx-oldx);

oldx = newx;

iter = iter+1;

end

if(iter>500)

oldx = 0.1\*randn(1);

while(error > tol)

y = feval(func,oldx,x,t);

ydash = feval(dfunc,oldx,x,t);

newx = oldx - y/ydash;

error = abs(newx-oldx);

oldx = newx;

end

end

res = oldx;

end

***6.2 Inviscid Burgers equation – solution***

%purge and clean

clc;

clear all;

%Parameters for Newton Raphson

error = 1;

tol = 1e-10;

%Parameters for the Burger's equation

A = 1;

u0 = 1;

%t = [0 0.5 1 1.5 2 3 4 6 8 10];

t = 1:0.2:10;

x = 0:0.1:2\*pi;

u = zeros(length(t),length(x));

q = 0;

for i = 1:length(t)

for j = 2:length(x)

if(i == 1)

u(i,j) = u0 + A\*sin(x(j)); %Declaration of Initial conditions

%u(i-1,j)

else % Newton Raphson goes here

q = x(j)+u(i-1,j)\*(t(i)-t(i-1));

u(i,j) = newton\_raphson1(@test,@dtest,u(i,j-1),q,t(i));

end

j = j+1;

end

i = i+1;

end

***6.3 Non-linear function (test)***

function [ f ] = test( u,x,t )

% Detailed explanation goes here

f = u-1-sin(x-(u\*t));

end

***6.4 Derivative of non-linear function (dtest)***

function [ f ] = dtest( u,x,t )

% Detailed explanation goes here

f = 1+(t\*cos(x-u\*t));

end