Summer Internship Report

Solving Burgers Equation using Numerical Techniques and its Parametric Study

Presented by

Aratrick Mondal
Department of Mechanical Engineering
Indian Institute of Technology, Guwahati

Presented to

Prof.Alankar Alankar
Department of Mechanical Engineering
Indian Institute of Technology, Bombay

1 Burgers Equation

1.1 Equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0$$

1.2 Description

Burgers equation is a combination of nonlinear wave motion with linear diffusion term and is the simplest model for analyzing combined effect of nonlinear advection and diffusion. The advection term u$\frac{\partial u}{\partial x}$ leads to shocking up effect that causes waves to break.The presence of viscous term $\nu\frac{\partial^2 u}{\partial x^2}$ helps suppress the wave-breaking, smooth out shock discontinuities and hence we expect to obtain a well-behaved and smooth solution. As the value of ν is decreased i.e as it is moves towards the inviscid regime, the smooth viscous solutions converge non-uniformly to the appropriate discontinuous shock wave, leading to an alternative mechanism for analyzing conservative nonlinear dynamical processes.

1.3 Applications

Burgers equation is a non-linear partial differential equation that relates convection and diffusion in an unsteady flow field. Hence it has wide applications in the field of Materials Science, Gas Dynamics, Fluid Mechanics and Traffic Flow.

1.4 Solution

1.4.1 Classical Solution

Initially, I tried to solve the Burgers equation using Classical Method. As it is a non-linear PDE ,solving it classically is very tough. So I converted the non-linear partial differential equation into a linear partial differential equation using Cole-Hopf Transformation. I assumed an exponential function:

$$w = e^{\frac{-1}{2\nu}\phi}$$

$$\phi = \phi(x,t)$$

Let us assume

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

Therefore,

$$u = -2\nu\frac{w_x}{w}$$

Now, rewriting the original Burgers equation in terms of the function ϕ we get

$$\frac{\partial^2\phi}{\partial t\partial x} + (\frac{\partial\phi}{\partial x})\frac{\partial^2\phi}{\partial x^2} - \nu\frac{\partial^3\phi}{\partial x^3} = 0$$

Integrating both sides w.r.t x we get,

$$\frac{\partial\phi}{\partial t} + \frac{1}{2}(\frac{\partial\phi}{\partial x})^2 - \nu\frac{\partial^2\phi}{\partial x^2} = 0$$

Now multiplying both sides by $\frac{-1}{2\nu}e^{\frac{-1}{2\nu}}$,

$$\frac{-1}{2\nu}e^{\frac{-1}{2\nu}}\frac{\partial\phi}{\partial t} - \frac{1}{4\nu}e^{\frac{-1}{2\nu}}(\frac{\partial\phi}{\partial x})^2 + \frac{1}{2}e^{\frac{-1}{2\nu}}\frac{\partial^2\phi}{\partial x^2} = 0$$

$$\frac{\partial w}{\partial t} - \nu\frac{\partial^2 w}{\partial x^2} = 0$$

Hence I have converted a complex non-linear PDE to a simple linear PDE which is now much easier to solve.

I solved the following question using classical method.

Example

Find the solution of the Burgers Equation given the following conditions:

$$IC : u(x,0) = sin(2\pi x) \qquad 0 < x < 1$$

$$BC : u(0,t) = u(1,t) = 0 \qquad t > 0$$

Solution:

Using Cole-Hopf Transformation,

$$u(x,t) = -2\nu\frac{w_x}{w}$$

where w satisfies the following linear PDE,

$$\boxed{\frac{\partial w}{\partial t} = \nu\frac{\partial^2 w}{\partial x^2}} \qquad 0 < x < 1, t > 0 \qquad (1)$$

Using the IC,

$$u(x,0) = sin(2\pi x) = -2\nu\frac{dw(x,0)_x}{w(x,0)}$$

$$\int_0^x \frac{-sin(2\pi x)}{2\nu}dx = \int_0^x \frac{dw(x,0)}{w(x,0)}$$

$$\frac{1}{4\nu\pi}cos(2\pi x)|_0^x = ln(w(x,0))$$

$$\frac{1-cos(2\pi x)}{-4\pi\nu} = ln(w(x,0))$$

$$\boxed{w(x,0) = e^{\frac{1-cos(2\pi x)}{-4\pi\nu}}} \tag{2}$$

From the BC we get,

$$\boxed{w(0,t)_x = w(1,t)_x = 0, t > 0} \tag{3}$$

So, now I need to solve the equation (1), using the conditions (2) and (3) :-
Let us solve by using the Separation of Variables,
Let,

$$w(x,t) = X(x)T(t)$$

$$XT' = \nu X''T$$

$$\frac{T'}{\nu T} = \frac{X''}{X} = k(say)$$

Therefore we get 2 sets of differential equations,

$$\boxed{T' - k\nu T = 0}$$

$$\boxed{X'' - kX = 0}$$

For getting non-trivial solution we must take k to be any negative constant,
Let k=-λ^2,

$$X'' + \lambda^2 X = 0$$

$$T' + \lambda^2 \nu T = 0$$

So the solution of these two differential equations are

$$X(x) = Asin(\lambda x) + Bcos(\lambda x)$$

$$T(t) = Ce^{-\nu\lambda^2 t}$$

$$w(x,t) = X(x)T(t)$$

$$w(x,t) = [Asin(\lambda x) + Bcos(\lambda x)][Ce^{-\nu\lambda^2 t}]$$

Using the Neumann Boundary Conditions, we get

$$\begin{aligned} A &= 0 \\ \lambda_n &= n\pi \quad\quad n = 0, 1, 2... \end{aligned} \tag{4}$$

Solution corresponding to each n is: $w_n(x,t) = A_n cos(n\pi x)e^{-\nu n^2\pi^2 t}$
The general solution is:

$$\boxed{w(x,t) = \sum_{n=0}^{\infty} w_n(x,t) = \sum_{n=0}^{\infty} A_n cos(n\pi x) e^{-\nu n^2 \pi^2 t}}$$

Using the IC (2) we get ,

$$w(x,0) = e^{\frac{1-cos(2\pi x)}{-4\pi\nu}} = \textstyle\sum_{n=0}^{\infty} A_n cos(n\pi x)$$

Let the solution be represented in the form of Fourier Series

$$w(x,t) = \tfrac{a_0}{2} + \textstyle\sum_{n=1}^{\infty} a_n e^{-\nu n^2 \pi^2 t} cos(n\pi x)$$

$$a_0 = 2 \textstyle\int_0^1 e^{\frac{1-cos(2\pi x)}{-4\pi\nu}}\, dx$$

$$a_n = 2 \textstyle\int_0^1 e^{\frac{1-cos(2\pi x)}{-4\pi\nu}} cos(n\pi x)\, dx$$

So finally writing the original solution function u(x,t) in terms of w(x,t)

$$u(x,t) = -2\nu \tfrac{w_x(x,t)}{w(x,t)}$$

$$\boxed{u(x,t) = \frac{2\nu\pi \sum_{n=1}^{\infty} n a_n e^{-n^2\pi^2\nu t} sin(n\pi x)}{\frac{a_0}{2} + \sum_{n=1}^{\infty} a_n e^{-n^2\pi^2\nu t} cos(n\pi x)}} \qquad (5)$$

Hence we get the final solution of the Burgers equation.

Remarks

Solving the Burgers Equation in a classical way was very complicated and time-consuming.So next I solved the Burgers Equation using different ***Numerical Techniques*** which gave me an approximate solution of the Burgers Equation.

1.4.2 Approximate Solution

1.4.2.1 Finite Difference Method (FDM)

- In FDM, I learnt different Finite Difference Schemes and the Order of Accuracy of these schemes.
- I learnt how to evaluate all these schemes using the Taylor Series and what is Truncation Error.
- I saw how the space and time domain is discretized and what is Von-Neumann Stability Factor.
- I could not go deep into the Convergence Analysis as it was out of scope for me to understand without learning higher level mathematics.

Example 1

I have used the FDM technique to solve the Burgers Equation in the domain $x \in [0, 1]$ with the following Initial and Boundary Conditions :

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0$$

$$u(x, 0) = sin(2\pi x)$$

$$u(0, t) = u(0, t) = 0$$

Solution:

The Python Code for solving the above problem using FDM is as follows:-

```python
import numpy as np
import sympy as sym
from matplotlib import pyplot
from numpy import linspace

#Parameters
h=0.025
k=0.0001
nu=1
L=1
T=1
fact1=k/h
fact2=nu*k/h**2

x=np.arange(0,L+h,h)
t=np.arange(0,T+k,k)

#Boundary Condition
bc=[0,0]

#Initial Condition
ic=np.sin(2*np.pi*x)

n=len(x)
m=len(t)
u=np.zeros((n,m))
u[0,:]=bc[0]
u[-1,:]=bc[1]
u[:,0]=ic
u.round(3)

for j in range(0,m-1):
    for i in range(1,n-1):
        u[i,j+1]=u[i][j]+fact1*((u[i][j]*u[i+1][j])-((u[i][j])**2))+
        fact2*(u[i+1][j]-2*u[i][j]+u[i-1][j])

u=u.round(3)
print(m)
print(u)
pyplot.title("Burgers␣Equation")
pyplot.xlabel("Distance(x)")
pyplot.ylabel("Displacement(u)")
R=np.linspace(1,0,m)
```

```
B=np.linspace(0,1,m)
G=0
line1,=pyplot.plot(x,u[:,1],color='blue')
#for j in range(1,m-1):
    #pyplot.plot(x,u[:,j],color=[R[j],G,B[j]])
line2,=pyplot.plot(x,u[:,10],color='orange')
line3,=pyplot.plot(x,u[:,100],color='olive')
line4,=pyplot.plot(x,u[:,500],color='violet')
line5,=pyplot.plot(x,u[:,1000],color='green')
line6,=pyplot.plot(x,u[:,m-1],color='cyan')
pyplot.legend([line1,line2,line3,line4,line5,line6],
["t=0","t=0.001s","t=0.01s","t=0.1s","t=0.5s","t=0.9s"])
pyplot.show()
```

Remark: I have used the **Explicit Finite Difference Method** to solve this problem. **FTCS(Forward Time Central Space)** Scheme has been used over here i.e the first order derivative of u with respect to time has been evaluated using the Forward Difference Scheme and the second order derivative of u with respect to x has been evaluated using the Central Difference Scheme. The spatial mesh size(h) and the time step(k) is taken very small so as to generate the result as accurate as possible.

Convergence: The value of the parameters are taken in a way that they fulfill the Convergence Criteria : $|\frac{\nu k}{h^2}| \leq \frac{1}{2}$

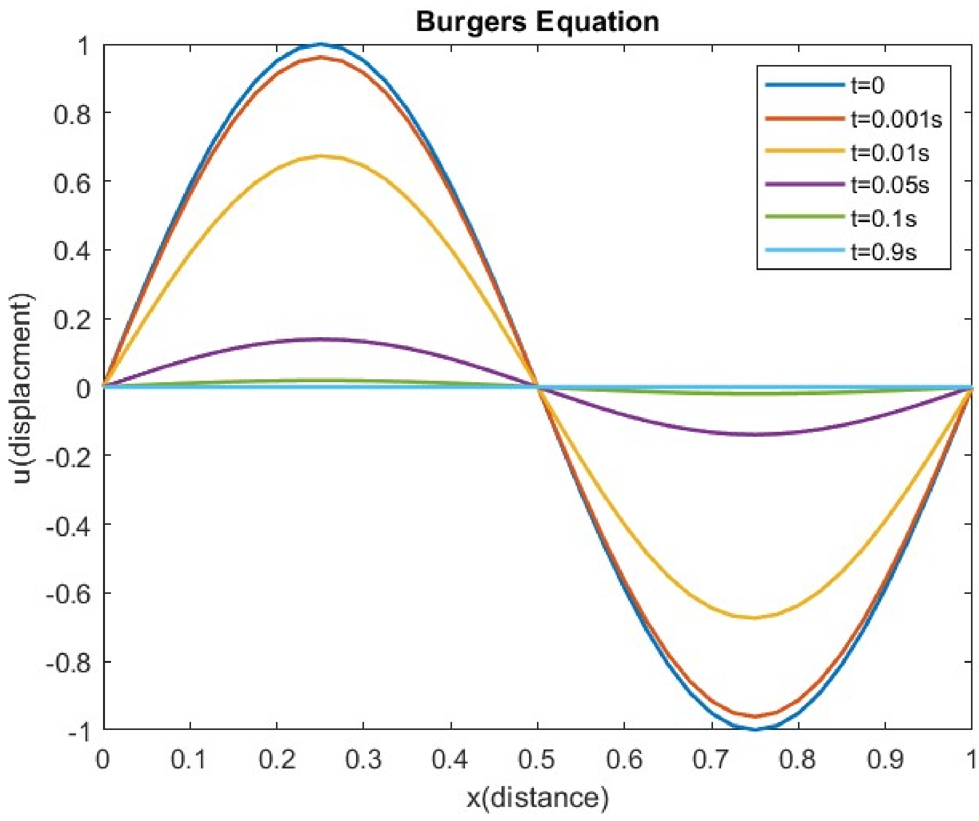


Figure 1: FDM solution of Burgers equation with $N = 40, L = 1, t = 1, \nu = 1, \Delta t = 0.0001$

1.4.2.2 Finite Element Method (FEM)

Before starting with FEM, I learned the Weighted-Integral and Variational (or Weak) Form of PDE. I understood what does the word ”weak” mean in the Weak or Variational Form. I learned different variational and weighted-residual methods for solving PDE. I recapitulated Vector Space, Norm, Basis, Inner Product, and various other topics of Linear Algebra so as to get a strong understanding of these different methods used in FEM.

I learned the *Rayleigh-Ritz method* under the category of variational methods. Under the weighted-residual methods, I learned the *Petrov-Galerkin, Galerkin, Least Square and Collocation Method.*

Example 1

I solved the following question using *Least Square Method:*

$$\frac{du}{dx} - 3x^2 = 0 : x \in (0, 4)$$

$$BC : u(0) = 0$$

Solution:

The Matlab Code for solving the above problem using *Least Square Method* is as follows:-

For simplicity I have chosen only one basis function : $\psi(x) = x^i$

```
syms psi(x);
syms R(c,x);

% Variable Declaration
i=2;
L=4;

% Basis Function
psi(x)=x^i;

% Trial Solution
uh(c,x)=c*psi(x);

% Residual and Norm
R(c,x)=diff(uh,x)-3*x^2;
Rnorm_square=int(R^2,x,[0,L])
eqn=diff(Rnorm_square,c)==0;
c1=solve(eqn,c);
c1
uh(x)=uh(c1,x);

% Domain
```

```
x=linspace(0,L,100);

% Plots
figure(1)
hold on;
plot(x,uh(x),"--r",'linewidth',1.5)
plot(x,x.^3,'-g','Linewidth',2)
title("Trial and Actual Solution")
legend("u_{trial}","u_{exact}")
xlabel('x')
ylabel('u(x)')
```

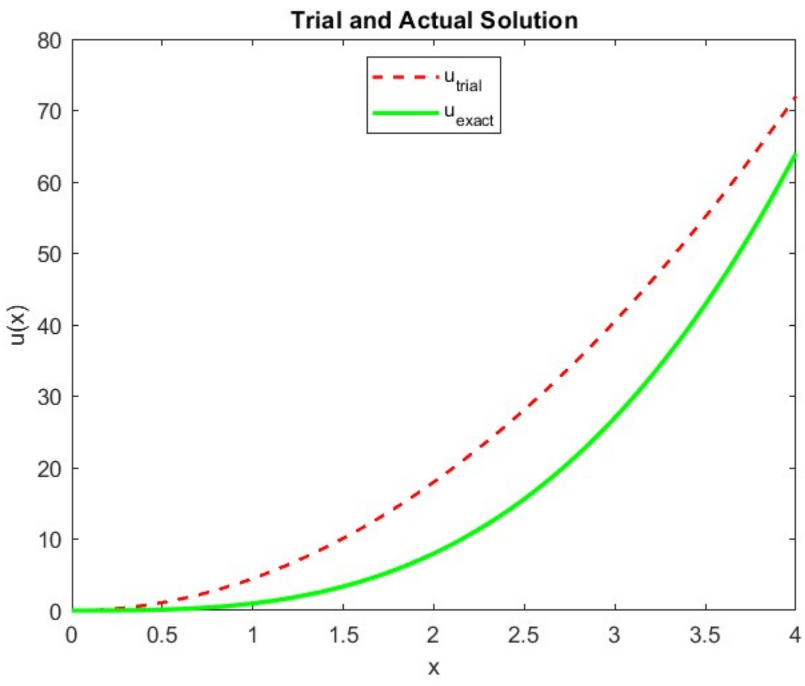


Figure 2: Graphical Interpretation of Exact and Approximate Solution

Example 2

Next, I solved another differential equation but now by using *Galerkin Method:*

$$\frac{d^2u}{dx^2} + sin(x - x^2) = 0 \quad x \in [0, 4]$$

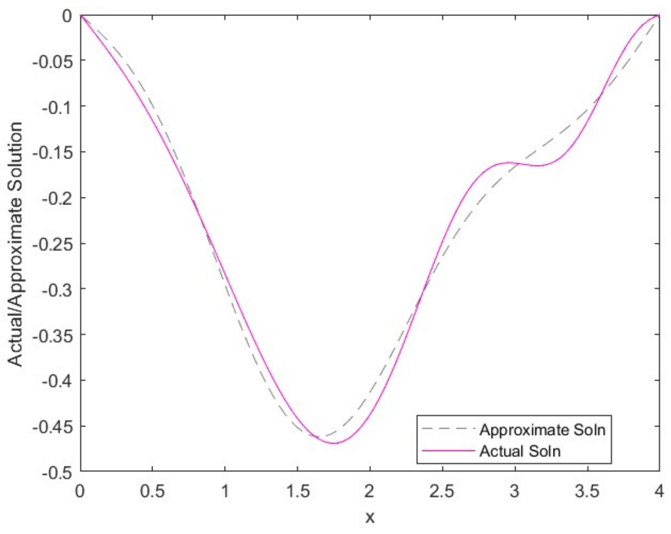
$$BC : u(0) = 0, u(4) = 0$$

Solution:

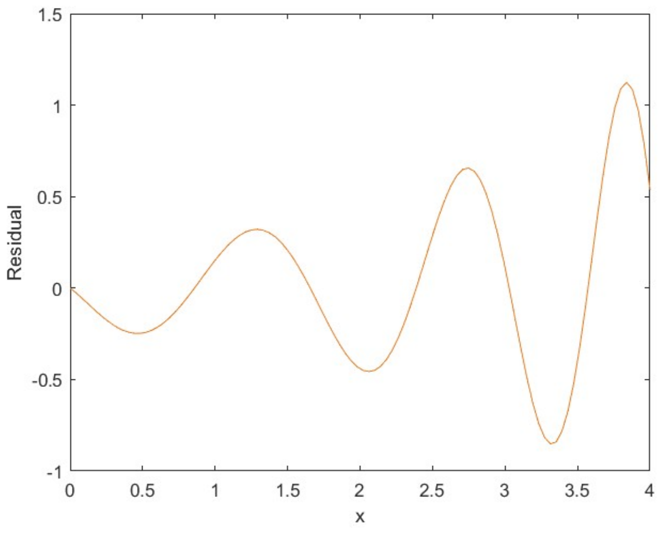
The Matlab Code for solving the above problem using *Galerkin Method* is as follows (This time I have also plotted Residual and Error associated at every point)

```matlab
syms u(x) f(x) u_trial(x);
syms A(i,j) b(i);
syms psi(x,i);
syms ddpsi(x,i);
syms x i j;
assume(i,{'real','positive'});
assume(j,{'real','positive'});

% Defining the function
f(x)=sin(x-x^2);
% Defining the ODE
ode=diff(u,x,2)+f==0;
% Parameters
L=4;                                    %%% Length of Domain
N=input('Dimension: ');                 %%% Dimension of Trial Space
% Boundary Conditions
condition=[u(0)==0,u(L)==0];
uactual(x)=dsolve(ode,condition);
%uactual(x)

% Defining the basis function
psi(x,i)=sin(i*x*pi/L);
ddpsi(x,i)=diff(psi,x,2);

b(i)=int(-f*psi(x,i),x,[0 L]);
A(i,j)=int(psi(x,i)*ddpsi(x,j),x,[0 L]);


Af=zeros(N,N);
bf=zeros(N,1);
for a=1:N
    for d=1:N
        Af(a,d)=double(A(a,d));
    end
    bf(a)=double(b(a));
end

c=inv(Af)*bf

div=input('Enter no.of divisions for the domain: ');
X=linspace(0,L,div);
psiM=zeros(div,N);
ddpsiM=zeros(div,N);

for i1=1:N
    psiM(:,i1)=double(psi(X,i1));
    ddpsiM(:,i1)=double(ddpsi(X,i1));
end

psiM;
```

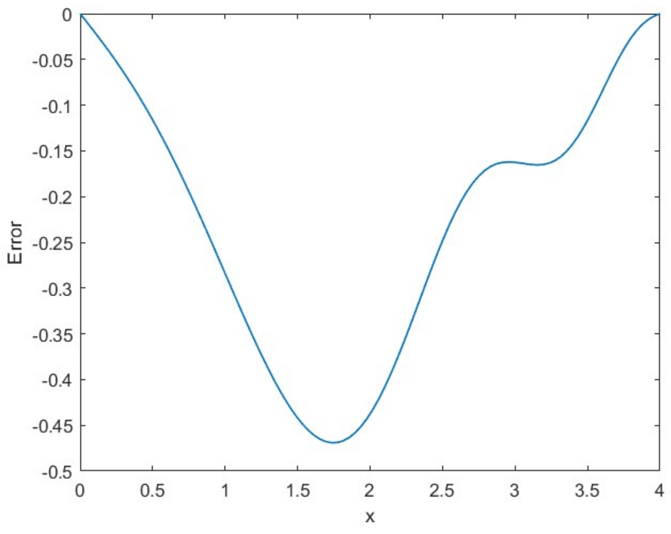
```
ddpsiM;

U=zeros(div,1);
for j1=1:N
    U=U+c(j1,1)*psiM(:,j1);
end

R=zeros(div,1);
for k1=1:N
    R=R+c(k1,1)*ddpsiM(:,k1);
end
fM=zeros(div,1);
for l1=1:div
    fM(l1,1)=fM(l1,1)+double(f((l1-1)*L/(div-1)));
end
fM
R=R+fM;

figure(1)
hold on;
plot(X,U,'--',"Color",[0.5 0.5 0.5])
plot(X,uactual(X),"Color",[1 0 0.8])
ylabel("Actual/Approximate Solution")
xlabel("x")
legend("Approximate Soln","Actual Soln")

figure(2)
hold on;
plot(X,R,'-',"Color",[0.9 0.47 0.1])
xlabel("x")
ylabel("Residual")

figure(3)
for i=1:div
  plot(X,double(uactual(X)-U(i)),'Linewidth',1)
end
xlabel("x")
ylabel("Error")
```



(a) Plot of Exact and Approximate Solution



(b) Plot of Residual



(c) Plot of Error

Figure 3: Graphical Analysis in MATLAB

Example 3 **(Finite Element Analysis of a 1-D Problem)**

$$-\frac{d^2u}{dx^2} - u + x^2 = 0 \quad x \in [0,1]$$

$$BC : u(0) = 0, u(1) = 0$$

Solution:

Here I will use a uniform mesh of three linear elements.
Using the Galerkin Finite Element Method, the weighted-integral form can be written as:

$$\int_0^1 w\left[-\frac{d^2u}{dx^2} - u + x^2\right]dx = 0$$

Decomposing the overall integral into the summation of integrals over each element, we get:

$$\sum_e \int_e w\left[-\frac{d^2u}{dx^2} - u + x^2\right]dx = 0$$

Developing the weak form:

$$\sum_e \int_{x_a}^{x_b} \left[\frac{dw}{dx}\frac{du}{dx} - wu + wx^2\right]dx - \sum_e \left[w\frac{du}{dx}\right]\Bigg|_{x_a}^{x_b} = 0 \tag{6}$$

The unknown primary variable,u is interpolated by polynomials such that

$$u(x) \approx U^e = \sum_{j=1}^{n} u_j^e \psi_j^e(x) \tag{7}$$

where n represents the no.of nodes which is 4 in this case.
The polynomials used for the interpolation are the same as the ones used to weight the integrated equation as per the Galerkin formulation.

$$w = \psi_i^e(x) \tag{8}$$

The shape functions are defined such that

$$\psi_i^e(x_j^e) = \delta_{ij} \qquad and \qquad \sum_i \psi_i^e(x) = 1$$

Substituting equations (7) and (8) into equation (6) and writing the equation in the indicial form:

$$\sum_{j=1}^{n} K_{ij}^e u_j^e - f_i^e - Q_i^e = 0 \qquad (i = 1, 2, 3, ..., n) \tag{9}$$

where

$$K_{ij}^e = \int_{x_a}^{x_b} \Big(\frac{d\psi_i^e}{dx} \frac{d\psi_j^e}{dx} - \psi_i^e \psi_j^e \Big) dx$$

$$f_i^e = - \int_{x_a}^{x_b} \psi_i^e x^2 dx$$

In matrix notation

$$[K^e]\{u^e\} = \{f^e\} + \{Q^e\} \tag{10}$$

The matrix $[K^e]$ is called the *coefficient matrix*, or *stiffness matrix.* The column vector $\{f^e\}$ is called the *source vector*, or *force vector.* The column vector $\{u^e\}$ contains the unknown information at the nodes that we want to find, it is also called *primary nodal degree of freedom.* Finally, the column vector $\{Q^e\}$ contains the information about the *secondary degrees of freedom* at the nodes.

These different matrices depend only on the shape functions and can be analytically evaluated. These local matrices are evaluated for each element and inserted in the global system of equations for all the nodes.

The interpolation inside an element bounded by the nodes i and $i+1$ is:

$$u(x) = u_i^e \psi_1^e(x) + u_{i+1}^e \psi_2^e(x)$$

and the shape functions are

$$\psi_1^e(x) = 1 - \frac{x}{h}$$

$$\psi_2^e(x) = \frac{x}{h}$$

where $h = 1/3$ i.e the length of each element.

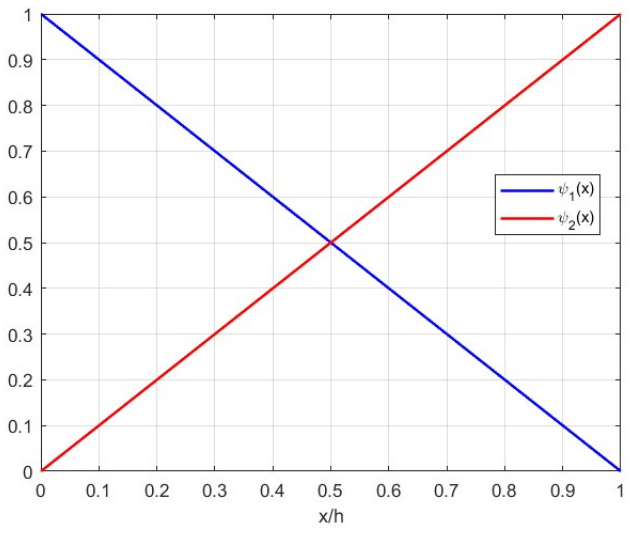


Figure 4: Shape functions for the linear Lagrange element

The local matrices for the element bounded by the nodes i and i+1 are given by

$$[K^e] = \begin{bmatrix} (1/h - h/3) & (-1/h - h/6) \\ (-1/h - h/6) & (1/h - h/3) \end{bmatrix}$$

$$= \begin{bmatrix} 2.8889 & -3.0556 \\ -3.0556 & 2.8889 \end{bmatrix}$$

Evaluating f_i^e for each element,

$$f_1^1 = -\int_0^h (1 - x/h)x^2 dx = -0.0031$$

$$f_2^1 = -\int_0^h (x/h)x^2 dx = -0.0093$$

Similarly,

$$f_1^2 = -0.0340 \quad f_2^2 = -0.0525$$
$$f_1^3 = -0.1019 \quad f_2^3 = -0.1327$$

So, after assembling we get

$$\begin{bmatrix} 2.8889 & -3.0556 & 0 & 0 \\ -3.0556 & 5.7778 & -3.0556 & 0 \\ 0 & -3.0556 & 5.7778 & -3.0556 \\ 0 & 0 & -3.0556 & 5.7778 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{Bmatrix} = \begin{Bmatrix} -0.0031 \\ -0.0432 \\ -0.1543 \\ -0.1327 \end{Bmatrix} \tag{11}$$

Now, imposing the boundary conditions $U_1 = 0$ and $U_4 = 0$, we get the condensed set of equations as

$$\begin{bmatrix} 5.7778 & -3.0556 \\ -3.0556 & 5.7778 \end{bmatrix} \begin{Bmatrix} U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} -0.0432 \\ -0.1543 \end{Bmatrix} \tag{12}$$

Solving this we get,

$$U_2 = -0.0300, \quad U_3 = -0.0426$$

MATLAB Code

```
%% Variable Declaration
syms x;
syms i;
syms j;
syms u(x);
%% Parameters
L=1;
N=3;     % no.of elements
n=N+1;   % no.of nodes
h=L/N;
%% Importing nodes and elements from text files
nodedata=readtable('Nodes.txt','HeaderLines',1)
elementdata=readtable('Elemets.txt','HeaderLines',1)

for k=1:n
 node(k)=nodedata.Var2(k);
end
%% ODE
ode=-diff(u,x,2)-u+x^2==0;
cond1=u(0)==0;
cond2=u(L)==0;
u_actual=dsolve(ode,[cond1 cond2]);
%% Defining the shape function
edit shape
%% Formulating element level stiffness(k) and force matrix(f)
Ke=zeros(2,2);
for i=1:2
    for j=1:2
        Ke(i,j)=int((diff(shape(x,1,i,h),x)*diff(shape(x,1,j,h),x
           ))-(shape(x,1,i,h)*shape(x,1,j,h)),x,[0 h]);
    end
end

f=zeros(n-1,2);
for i=1:n-1
    for j=1:2
        f(i,j)=-int(shape(x,i,j,h)*x^2,x,[(i-1)*h i*h]);
    end
end

Km=zeros(n,n);
fm=zeros(n,1);
%% Assembling to form Global Stiffness Matrix K
for i=1:n-1
        Km(i,i)=Km(i,i)+Ke(1,1);
        Km(i,i+1)=Km(i,i+1)+Ke(1,2);
        Km(i+1,i)=Km(i+1,i)+Ke(2,1);
        Km(i+1,i+1)=Km(i+1,i+1)+Ke(2,2);
end
Km;
%% Assembling to form Global Force Matrix F
for j=1:n-1
    fm(j,1)=fm(j,1)+f(j,1);
    fm(j+1,1)=fm(j+1,1)+f(j,2);
```

```
end
fm;
%% Boundary Conditions
fixed_nodes=[1 n];
free_nodes=setxor(1:n,fixed_nodes);
%% Partitioning K and F matrix
Kpart=Km(free_nodes,free_nodes);
Fpart=fm(free_nodes,1);
%% Solving the system of Eqns
Up=Kpart\Fpart;
U=zeros(n,1);
U(free_nodes)=Up;
U
```

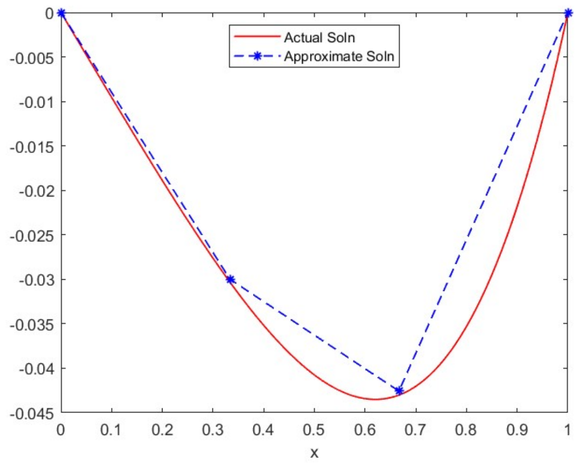


Figure 5: Graphical Interpretation of Exact and Approximate Solution

Remark: We can notice that the solution we obtained using FEM is close to the actual solution at the nodes only. At points other than the nodes the approximated solution vary greatly from the actual solution. This error can be minimised by refining the mesh i.e by taking the length of the element as small as possible.

Semidiscrete Finite Element Modeling of Burgers Equation

The time-dependent Burgers Equation is written as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0 \tag{13}$$

$$Initial\ Condition : u(x,0) = sin(2\pi x) \qquad 0 < x < 1$$

$$Boundary\ Conditions : u(0,t) = u(1,t) = 0 \qquad t > 0$$

This semidiscrete finite element formulation involves the use of finite elements in space and discretizing the solution with respect to time coordinate.

First of all, we will construct the weak form over a typical element by using Galerkin Finite element formulation. For this, we multiply the Burgers Equation (13) with a weight function w and apply integration by parts to get

$$\int_{x_a}^{x_b} \left[w\frac{\partial u}{\partial t} - \frac{1}{2}\frac{\partial w}{\partial x}u^2 + \nu \frac{\partial w}{\partial x}\frac{\partial u}{\partial x} \right] dx + \left[\frac{w}{2}u^2 \right] \Bigg|_{x_a}^{x_b} - \left[\nu w \frac{\partial u}{\partial x} \right] \Bigg|_{x_a}^{x_b} = 0$$

$$\int_{x_a}^{x_b} \left[w\frac{\partial u}{\partial t} - \frac{1}{2}\frac{\partial w}{\partial x}u^2 + \nu \frac{\partial w}{\partial x}\frac{\partial u}{\partial x} \right] dx = \overline{Q_1} + \overline{Q_2} + \overline{Q_3} + \overline{Q_4} \tag{14}$$

where

$$\overline{Q_1} = -\left(\frac{w}{2}u^2 \right) \Bigg|_{x_b}, \qquad \overline{Q_2} = \left(\frac{w}{2}u^2 \right) \Bigg|_{x_a}$$

$$\overline{Q_3} = \left(\nu w \frac{\partial u}{\partial x} \right) \Bigg|_{x_b}, \qquad \overline{Q_4} = -\left(\nu w \frac{\partial u}{\partial x} \right) \Bigg|_{x_a}$$

The solution u is approximated using the Linear Lagrange elements as

$$u(x,t) \approx U^e(x,t) = \sum_{j=1}^{n} u_j^e(t)\psi_j^e(x) \tag{15}$$

where n represents the no.of nodes in the spatial domain.
The polynomials used for the interpolation are the same as the ones used to weight the integrated equation in accordance with the Galerkin formulation.

$$w = \psi_i^e(x) \tag{16}$$

The shape functions are defined such that

$$\psi_i^e(x_j^e) = \delta_{ij} \qquad and \qquad \sum_i \psi_i^e(x) = 1 \tag{17}$$

The interpolation inside an element bounded by the nodes i and $i+1$ is:

$$u(x) = u_i^e \psi_1^e(x) + u_{i+1}^e \psi_2^e(x)$$

and the shape functions are (represented in Fig 4)

$$\psi_1^e(x) = 1 - \frac{x}{h}$$

$$\psi_2^e(x) = \frac{x}{h}$$

where h is the length of each element.
After obtaining the finite element solution in spatial domain we will use one of the time approximation schemes to find u_j at time t_{s+1} using the known values

of u_j from previous times. Hence at the end of this two-stage formulation we will have a continuous spatial solution at discrete intervals of time as

$$U^e(x,t_s) = \sum_{j=1}^{n} u_j^e(t_s)\psi_j^e(x) \qquad (s = 0,1,2,...t_f) \tag{18}$$

Substituting (15) and (16) into equation (14) and then performing summation over all the elements, we get

$$\sum_{j=1}^{n}\int_{\Omega_e}\left[\psi_i\psi_j\frac{du_j}{dt}\right]dx - \sum_{j=1}^{n}\int_{\Omega_e}\left[\frac{1}{2}\frac{d\psi_i}{dx}\psi_j^2u_j^2\right]dx + \nu\sum_{j=1}^{n}\int_{\Omega_e}\left[\frac{d\psi_i}{dx}\frac{d\psi_j}{dx}u_j\right]dx = Q_i$$

Finally, we have

$$\sum_{j=1}^{n} M_{ij}\frac{du_j}{dt} + \sum_{j=1}^{n} A_{ij}u_j^2 + \nu\sum_{j=1}^{n} K_{ij}u_j = Q_i \tag{19}$$

where

$$M_{ij} = \int_{\Omega_e}\psi_i\psi_j dx$$

$$A_{ij} = -\frac{1}{2}\int_{\Omega_e}\frac{d\psi_i}{dx}\psi_j^2 dx$$

$$K_{ij} = \int_{\Omega_e}\frac{d\psi_i}{dx}\frac{d\psi_j}{dx}dx$$

In matrix form, we have

$$[M]\{\dot{u}\} + [A]\{u\} + \nu[K]\{u\} = \{Q\} \tag{20}$$

[M] and [K] are called mass matrix and stiffness matrix respectively.
$\{Q^e\}$ contains the information about the *secondary degrees of freedom* at the nodes.
The local matrices evaluated over an element are as follows

$$[M^e] = \frac{h}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$[K^e] = \frac{1}{h}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$[A^e] = \frac{1}{6}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

We need to assemble these matrices to form the global set of matrices which will be used to obtain the final solution of (20).

<u>**Time Discretization**</u>

I used the θ-method to replace the time-derivative by a simple difference as

$$\boxed{\{\dot{u}\} = \frac{\{u\}^{s+1} - \{u\}^s}{\Delta t}} \tag{21}$$

where

$\{u\}^s = u(x, t_s)$ denotes the primary variable's value at time $t = t_s$ and
Δt is the time increment and $t_{s+1} = t_s + \Delta t$
The solution u can be written in terms of the relaxation parameter θ as

$$u = \theta \cdot u^{s+1} + (1 - \theta) \cdot u^s, \quad t_s < t < t_{s+1} \tag{22}$$

Setting $\theta = 0$ i.e using the explicit Euler forward scheme, we get

$$\boxed{u = u^s} \tag{23}$$

Substituting Eqs. (21) and (22) into (20), I obtain

$$[M]\{u\}^{s+1} - [M]\{u\}^s + \Delta t[A]\{u\} + \nu\Delta t[K]\{u\} = \Delta t\{Q\}$$

$$\boxed{\{u\}^{s+1} = [M]^{-1}\Big[[M]\{u\}^s - \Delta t[A]\{u\} - \nu\Delta t[K]\{u\} + \Delta t\{Q\}\Big]} \tag{24}$$

Before finding the solution we need to lump the mass matrix as we are using the Euler explicit forward difference scheme.
Let the lumped mass matrix be denoted by $\mathbf{M}^l$

$$\mathbf{M}^l = \left[M^l_{ij}\right] \tag{25}$$

where

$$M^l_{ij} = \begin{cases} 0 & if\ i \neq j \\ \sum_{j=1}^{n} M_{ij} & if\ i = j \end{cases} \tag{26}$$

[M_{ij} represents the elements of the mass matrix M]
The lumped mass matrix is a diagonal matrix so finally the equation becomes

$$\boxed{\{u\}^{s+1} = \Delta t[M^l]^{-1}\Big[-[A]\{u\} - \nu t[K]\{u\} + t\{Q\}\Big] + \{u\}^s} \tag{27}$$

where $s = 0, 1, 2, ..., t_s$
$\{u\}^0$ can be obtained from the initial condition $u(x, 0) = sin(2\pi x)$
We then run a loop to iterate the process and get $\{u\}$ at each time step till we reach $t = t_s$.

MATLAB Code

```
syms x;
syms u0(x);
%% Taking input from the user
N=input('Enter the number of elements: ');
nu=input('Enter the value of kinematic viscocity: ');
L=input('Enter the length of the domain: ');
time=input('Enter the total duration of time: ');
k=input('Enter the time step: ');
%% Length of each element
h=L/N;
%% Total number of time steps
n=time/k;
%% Defining and Assembling the matrix
Mij=(h/6)*[2 1;1 2];
M=zeros(N+1,N+1);
for i=1:N
       M(i,i)=M(i,i)+Mij(1,1);
       M(i,i+1)=M(i,i+1)+Mij(1,2);
       M(i+1,i)=M(i+1,i)+Mij(2,1);
       M(i+1,i+1)=M(i+1,i+1)+Mij(2,2);
end
Kij=(1/h)*[1 -1;-1 1];
K=zeros(N+1,N+1);
for i=1:N
       K(i,i)=K(i,i)+Kij(1,1);
       K(i,i+1)=K(i,i+1)+Kij(1,2);
       K(i+1,i)=K(i+1,i)+Kij(2,1);
       K(i+1,i+1)=K(i+1,i+1)+Kij(2,2);
end
Aij=(1/6)*[1 1;-1 -1];
A=zeros(N+1,N+1);
for i=1:N
       A(i,i)=A(i,i)+Aij(1,1);
       A(i,i+1)=A(i,i+1)+Aij(1,2);
       A(i+1,i)=A(i+1,i)+Aij(2,1);
       A(i+1,i+1)=A(i+1,i+1)+Aij(2,2);
end
%% Lumping Mass Matrix
m=zeros(N+1,N+1);
for i=1:N+1
    for j=1:N+1
       m(i,i)=m(i,i)+M(i,j);
    end
end
%% Creating the Nodes
X=linspace(0,L,N+1)';
%% Initial and Boundary Condition
u0(x)=sin(2*pi*x);
U0=zeros(N+1,1);
```

```
for i=1:N+1
  U0(i,:)=double(u0(X(i)));
end
U0
%% Function for squaring the elements of the matrix
edit MatrixSquare
%% Main Code
figure(1)
hold on
box on
plot(X,U0,'LineWidth',1.5)
xlim([0 1])
U=zeros(N+1,n);
for j=1:n
  U(:,j)=inv(m)*[-(k*A*MatrixSquare(U0,N+1))-(k*nu*K*U0)]+U0;
  U(1,j)=0;
  U(N+1,j)=0;
  U0=U(:,j);
end
U
plot(X,U(:,10),'LineWidth',1.5)
plot(X,U(:,100),'LineWidth',1.5)
plot(X,U(:,500),'LineWidth',1.5)
plot(X,U(:,1000),'LineWidth',1.5)
plot(X,U(:,n-1),'LineWidth',1.5)
```

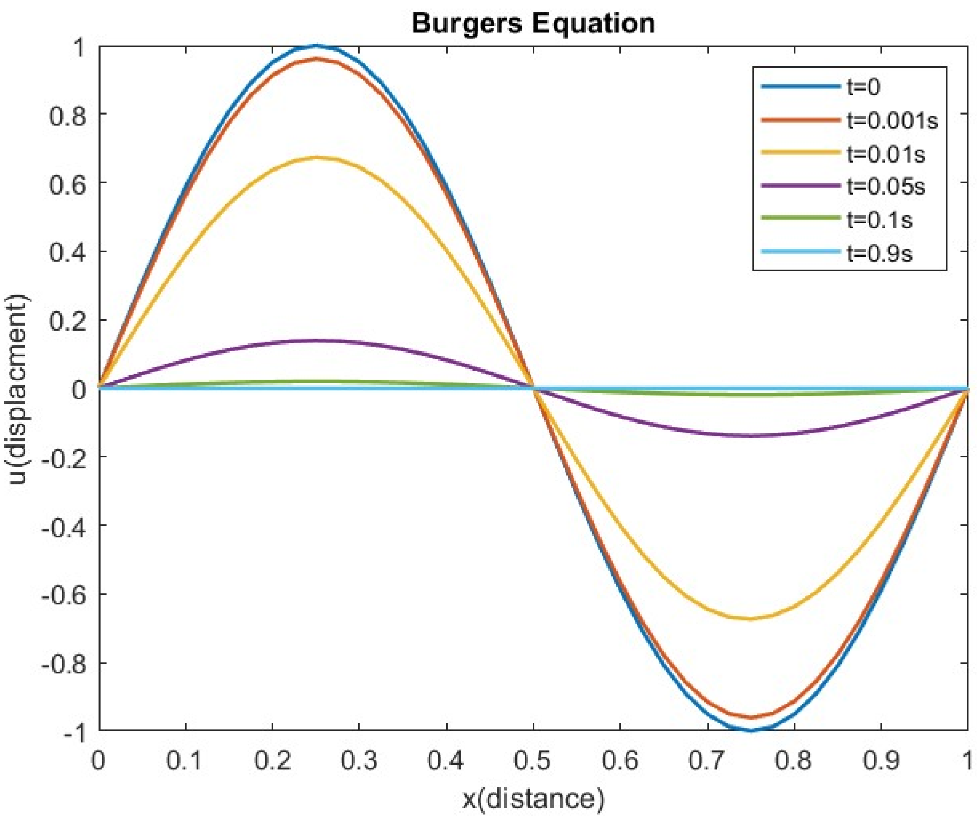


Figure 6: FEM solution of Burgers equation with $N = 40, L = 1, t = 1, \nu = 1, \Delta t = 0.0001$

1.4.2.3 Parametric Study

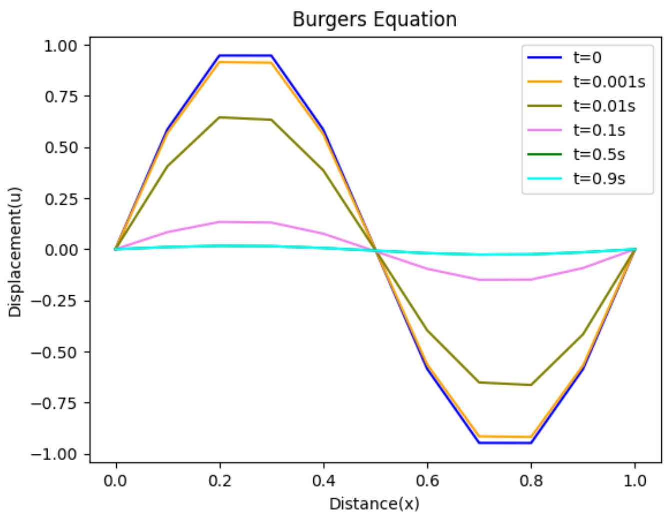
In this section, we will analyse the solution of Burgers equation by changing different parameters involved like ν, Δt, and N. We will then see how these parameters affect the solution of Burgers equation by graphical interpretation.

	Numerical Solution					
	N=10		N=20		N=40	
x	FDM	FEM	FDM	FEM	FDM	FEM
0.1	0.0106	0.0128	0.0056	0.0061	0.0084	0.0088
0.2	0.0166	0.0207	0.0144	0.0160	0.0163	0.0172
0.3	0.015	0.0207	0.0173	0.0198	0.0177	0.0191
0.4	0.0061	0.0128	0.0129	0.0160	0.0120	0.0137
0.5	-0.007	0.0000	0.0027	0.0061	0.0013	0.0030
0.6	-0.0195	-0.0128	-0.0095	-0.0061	-0.0104	-0.0088
0.7	-0.0264	-0.0207	-0.0191	-0.0160	-0.0187	-0.0172
0.8	-0.0249	-0.0207	-0.0222	-0.0198	-0.0202	-0.0191
0.9	-0.0151	-0.0128	-0.0176	-0.0160	-0.0143	-0.0136

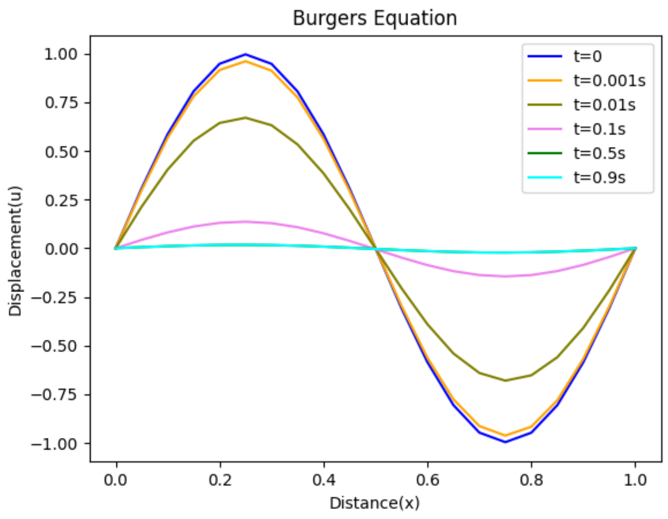
Table 1: Comparison of the numerical solutions obtained with different number of basis elements for $\nu = 1$, $\Delta t = 0.0001$ at t = 0.1

	Numerical Solution for FEM									
	$\nu = 1$		$\nu = 0.5$		$\nu = 0.1$		$\nu = 0.05$		$\nu = 0.01$	
x	FDM	FEM	FDM	FEM	FDM	FEM	FDM	FEM	FDM	FEM
0.1	0.0084	0.0088	0.0627	0.0623	0.3770	0.2580	0.5181	0.2941	0.7084	0.3220
0.2	0.0163	0.0172	0.1217	0.1231	0.6323	0.5428	0.7834	0.6311	0.9177	0.7016
0.3	0.0177	0.0191	0.1324	0.1376	0.5923	0.6719	0.6873	0.8141	0.7600	0.9394
0.4	0.0120	0.0137	0.0920	0.0993	0.3751	0.5407	0.4226	0.6969	0.4577	0.8683
0.5	0.0013	0.0030	0.0165	0.0221	0.0758	0.1280	0.0863	0.1731	0.0943	0.2352
0.6	-0.0104	-0.0088	-0.0665	-0.0639	-0.2426	-0.3627	-0.2672	-0.4815	-0.2845	-0.6307
0.7	-0.0187	-0.0172	-0.1270	-0.1247	-0.5238	-0.6441	-0.5916	-0.8036	-0.6399	-0.9581
0.8	-0.0202	-0.0191	-0.1415	-0.1370	-0.6877	-0.6336	-0.8195	-0.7497	-0.919	-0.8460
0.9	-0.0143	-0.0136	-0.1022	-0.0973	-0.5984	-0.4134	-0.7954	-0.4749	-1.0023	-0.5229

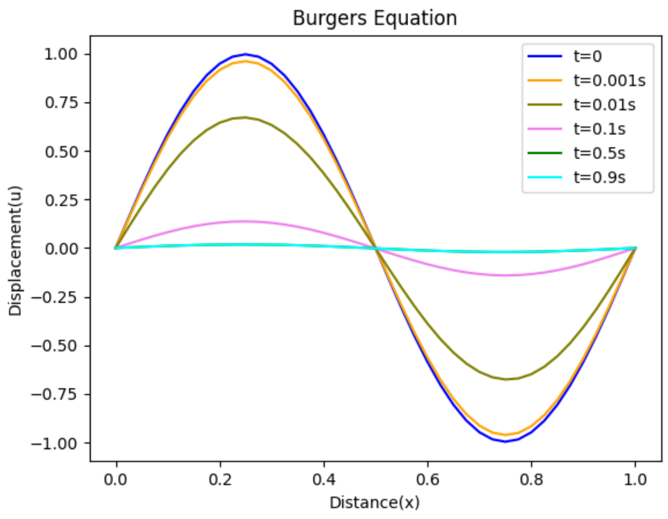
Table 2: Comparison of the numerical solutions obtained with different values of kinematic viscosity for N = 40, $\Delta t = 0.0001$ at t = 0.1



(a) N=10, $\nu = 1$, $\Delta t = 0.0001$

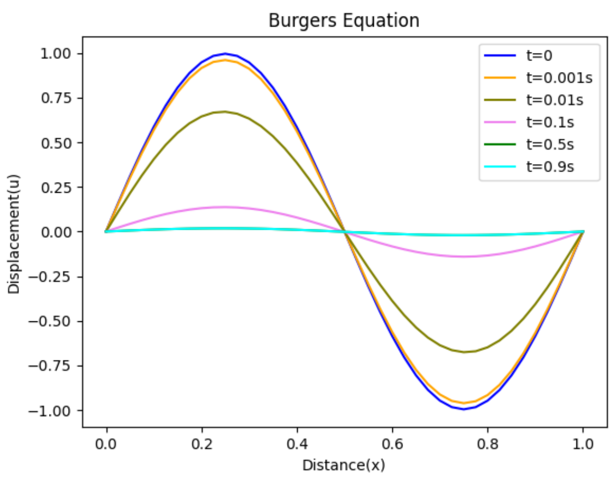


(b) N=20, $\nu = 1$, $\Delta t = 0.0001$

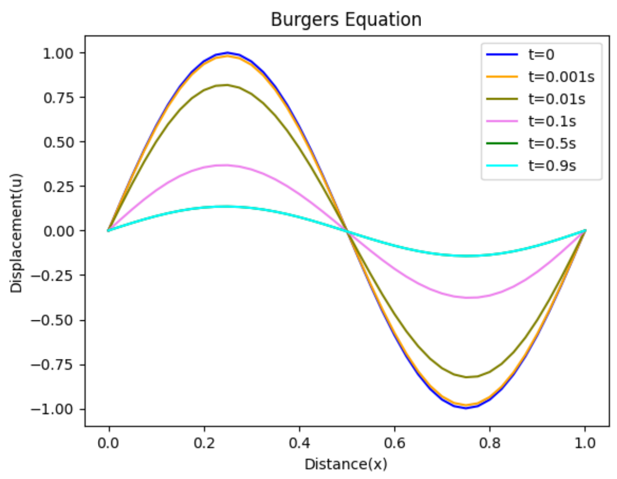


(c) N=40, $\nu = 1$, $\Delta t = 0.0001$

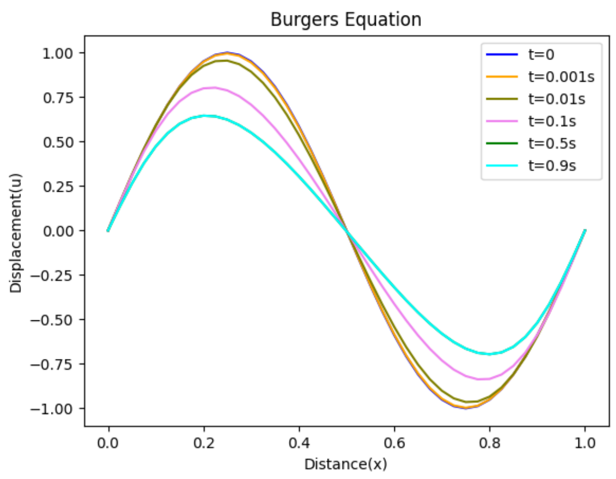
Figure 7: Numerical solutions of FDM obtained with different number of spatial elements in Python



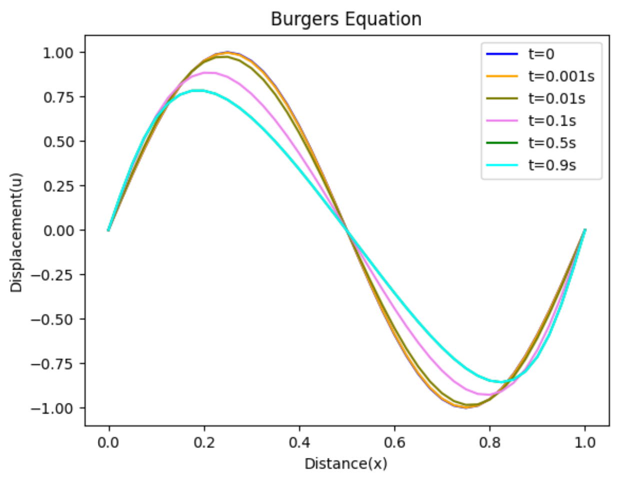
(a) N=40, $\nu = 1$, $\Delta t = 0.0001$



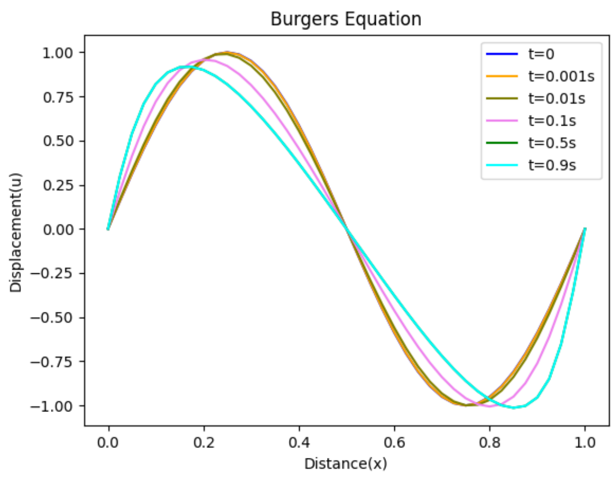
(b) N=40, $\nu = 0.5$, $\Delta t = 0.0001$



(c) N=40, $\nu = 0.1$, $\Delta t = 0.0001$

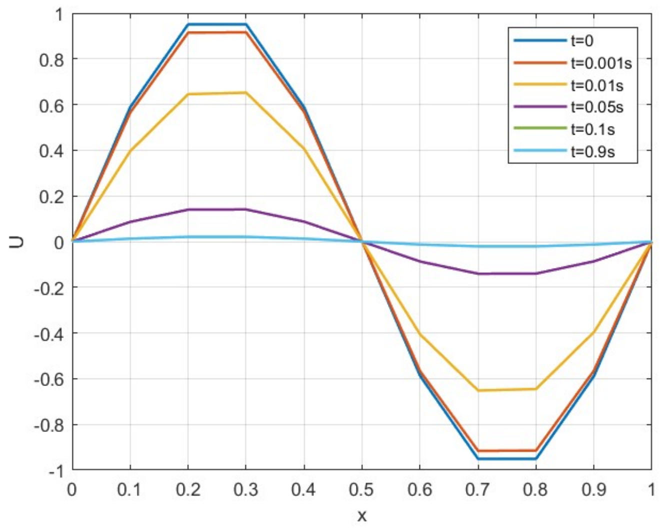


(d) N=40, $\nu = 0.05$, $\Delta t = 0.0001$

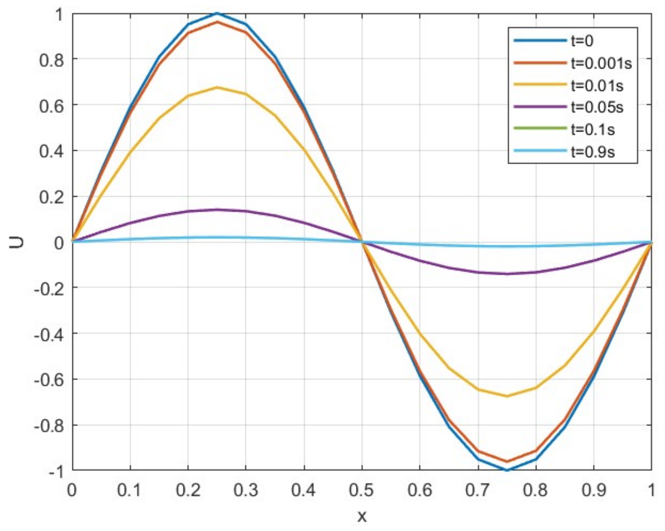


(e) N=40, $\nu = 0.01$, $\Delta t = 0.0001$

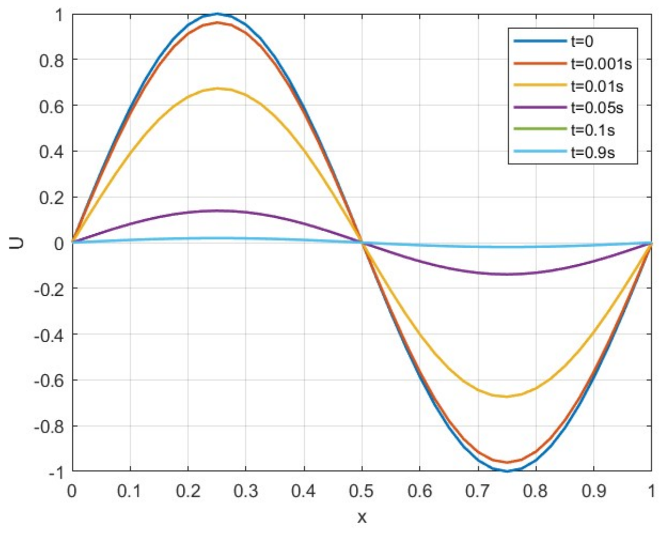
Figure 8: Numerical solutions of FDM obtained for different values of kinematic viscosity in Python



(a) $N = 10$, $\nu = 1$, $\Delta t = 0.0001$

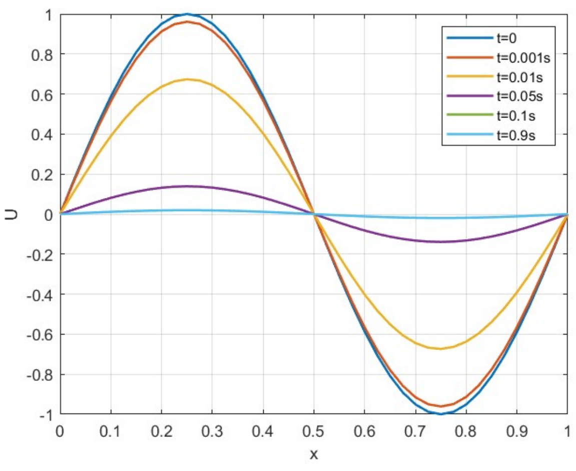


(b) $N = 20$, $\nu = 1$, $\Delta t = 0.0001$

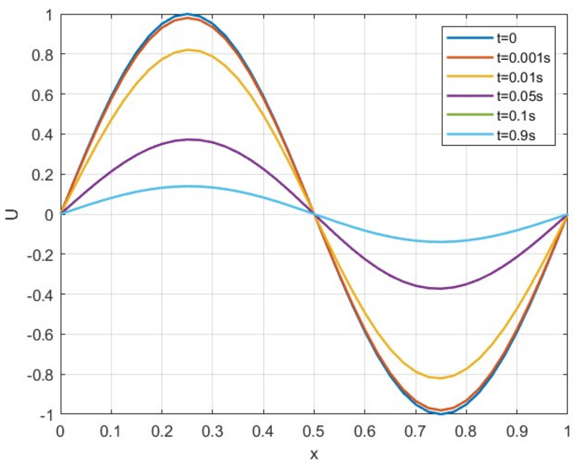


(c) N=40, $\nu = 1$, $\Delta t = 0.0001$

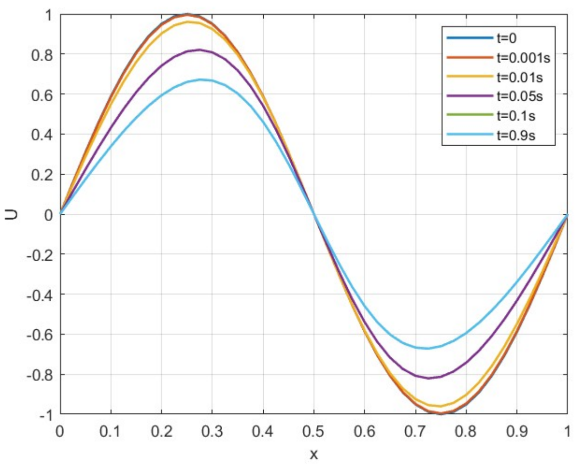
Figure 9: Numerical solutions of FEM obtained with different number of linear Lagrange elements in MATLAB



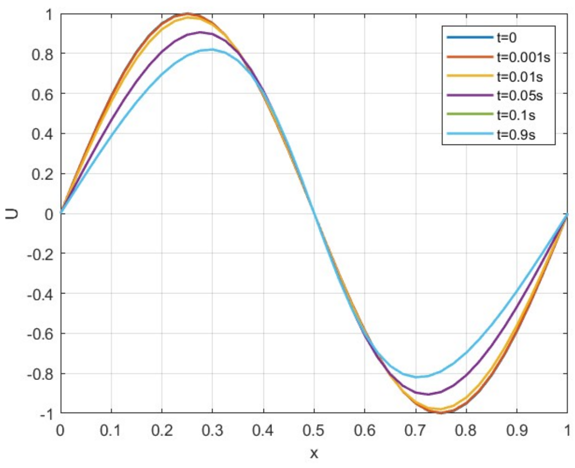
(a) N=40, $\nu = 1$, $\Delta t = 0.0001$



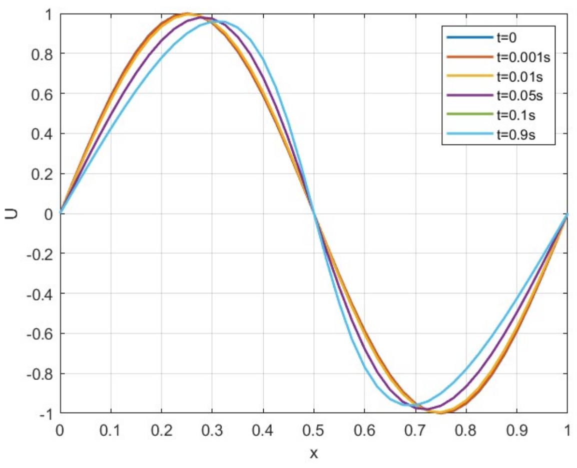
(b) N=40, $\nu = 0.5$, $\Delta t = 0.0001$



(c) N=40, $\nu = 0.1$, $\Delta t = 0.0001$



(d) N=40, $\nu = 0.05$, $\Delta t = 0.0001$



(e) N=40, $\nu = 0.01$, $\Delta t = 0.0001$

Figure 10: Numerical solutions of FEM obtained for different values of kinematic viscosity in MATLAB