Abstract:

This paper compares the effectiveness of the Finite Volume method and the Finite Difference method in numerically computing solutions to the inviscous Burgers equation $q_t + qq_x = 0$. It was assumed that the boundary conditions and initial conditions were known. A fourth order Taylor approximation with respect to time in conjunction with a finite difference approximation with respect to *x* was the most successful finite difference method attempted. As higher order approximations were implemented, the calculation became more accurate but time consuming. Further, a small amount of chaos was always seen near the shock. In contrast, the finite volume method was implemented with only a first order approximation of flux, known as the Local Lax-Fredrichs method. Under proper grid settings a stable solution was always obtained and more refined grids exhibited greater accuracy. It was discovered that calculation time was far more predictable for the finite difference method than for the finite volume method.

Introduction:

Non-hyperbolic PDE's are known to be very difficult problems to solve, both analytically and numerically. One such equation is the Inviscous Burgers Equation, $q_t + qq_x = 0$, having a solution that can develop a shock in finite. At a shock, the solution's derivative is undefined and hence using finite differences, which approximate the derivatives of the solution, will inherently have problems computing the solution. The finite volume method is much more adapted to handling shocks. It works under the principle of calculating what amount of volume or area is transferred to neighboring cells over a small amount of time. The advantage here is that it doesn't require any derivative approximations of the solution.

The performances of both methods, in calculating solutions the inviscous Burgers equation, are thoroughly tested and compared. The initial/boundary conditions of the solution are assumed to be known. The solution is always calculated on the finite spatial interval [-5, 5] and we assume the boundary values are held constant over time. As a convention for this paper we let the numerical approximation of the solution at a given (x, t) coordinate be denoted as follows: $Q_i^n \equiv q(x_i, t_n)$ where $x_i \equiv x_0 + i\Delta x$ and $t_n \equiv t_0 + n\Delta t$.

There are two forms in which Burgers equation can be expressed. The form displayed above is known as the quasi-linear form. The second form is called the conservative form as shown in equation 1.

$$q_t + \left(\frac{q^2}{2}\right)_x = 0 \tag{1}$$

This is the form used with the finite volume method and hence it will also be used with the finite difference method.

Part I: Finite Difference Method

The conventional way to apply the finite differences method is to approximate derivatives of the solution given in the PDE using the appropriate finite difference formula. If one were to apply a high order finite difference formula to equation 1, the resulting stencil would require knowing many values in both the *x*- and *t*-directions, respectively. If the solution is solved stepwise through time then, given what is known about the initial and boundary conditions, it would be best if the stencil used required knowing minimal values in the t-direction and could instead rely exclusively on the known *x*-values from the previous time iterate. This is possible if the finite difference method is used on a Taylor approximation of the solution.

$$Q_i^{n+1} = Q_i^n + \Delta t\left(Q_i^n\right)_t + \frac{\Delta t^2}{2}\left(Q_i^n\right)_{tt} + \frac{\Delta t^3}{6}\left(Q_i^n\right)_{ttt} + \dots \qquad (2)$$

Now each of the *t*-derivates can be written in terms of *x*-derivatives analytically because of the simplicity of the PDE. We already know from equation 1 that $q_t = -\left(\frac{q^2}{2}\right)_x$. Continuing to differentiate both sides in terms of t we find the following simple set of relations.

$$q_{tt} = -\left(\frac{q^3}{3}\right)_{xx} \qquad q_{ttt} = -\left(\frac{q^4}{4}\right)_{xxx} \qquad q_{tttt} = -\left(\frac{q^5}{5}\right)_{xxxx} \qquad (3)$$

By plugging this set of formulas into equation 2 we can express Q_i^{n+1} completely in terms of *x*-derivatives.

$$Q_i^{n+1} = Q_i^n - \Delta t\left(\frac{(Q_i^n)^2}{2}\right)_x + \frac{\Delta t^2}{2}\left(\frac{(Q_i^n)^3}{3}\right)_{xx} - \frac{\Delta t^3}{6}\left(\frac{(Q_i^n)^4}{4}\right)_{xxx} + \dots \qquad (4)$$

Using only minimal order centered finite difference approximations[1] on the *x*-derivatives; the formula takes the form given in equation 5 where the ellipses represent linear combinations of the solution, to the appropriate power[2], at the previous time iterate and close spatial values.

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{4\Delta x}(\dots) + \frac{\Delta t^2}{6\Delta x^2}(\dots)_{xx} - \frac{\Delta t^3}{48\Delta x^3}(\dots)_{xxx} + \mathrm{O}(\Delta t^4 + \Delta x^4) \qquad (5)$$

[1] Higher order derivatives require higher order centered approximations at minimum (e.g. there would be no way to approximate a fourth order derivative with a second order approximation)
[2] The power of the solution Q corresponds to those in equation 4, within the correct set of parenthesis.

For the remainder of the paper, equation 5 will be referred to as an FD formula and its order is defined by the order of the corresponding Taylor approximation that was used in deriving the FD formula. To begin, the smooth wave depicted in Figure 1 will be used as the initial condition.

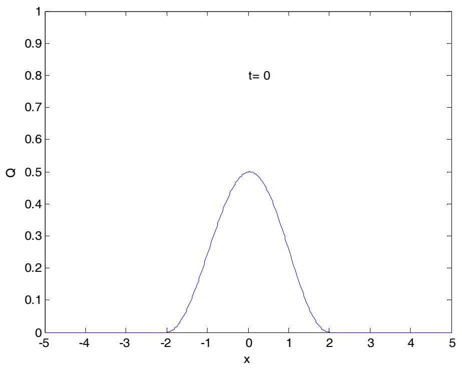


Figure 1: Initial Condition

For now we will always set $\Delta x = \Delta t = 0.01$, later on we will analyze what happens as these parameters are varied. For each trial, the solution is computed to $t = 5$. When using a second order FD formula, the solution remains smooth for small values of t. When the shock is reached, the solution becomes chaotic very rapidly. Figure 2 shows what this solution looks like at $t = 5$. We know the solution must be smooth and so clearly this is a very poor approximation.

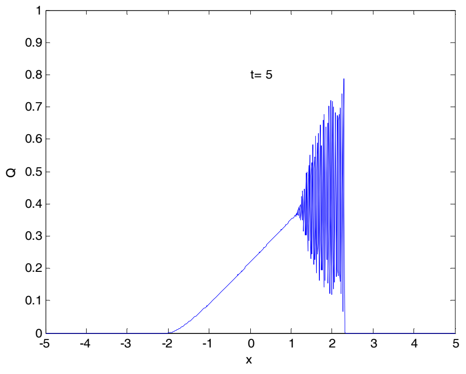


Figure 3: Using second order FD at t=5

Refining the grid does not give a better approximation of the solution. This chaos arises from the fact that the x-derivative of the solution is undefined right at the shock, so the finite difference approximation breaks down. To better approximate the solution, we use a higher order FD formula. Using fourth order FD formula with the same grid settings, we expect that the solution should look good up until the shock is reached and then some will occur. Figure 3 shows the the solution solved for by the fourth order FD at $t = 5$ and $t = 10$. The amount of chaos, after the shock, has been reduced dramatically and this chaotic nature of the solution does not grow with time. The chaos actually stays contained near the shock. This is a very promising result however one can clearly see that it is not a perfect solution as there is still some strange Gibbs phenomena that occurs near the shock. To find a close approximation of the weak solution this method would be insufficient. As the grid is refine the thickness of this spike at the top of the shock reduces however it's overshoot, or height, is not reduced. Fourth order was the highest order that was tested here.

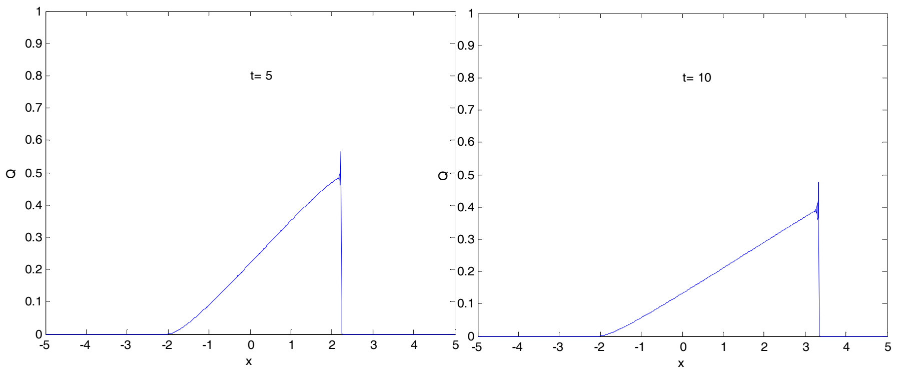
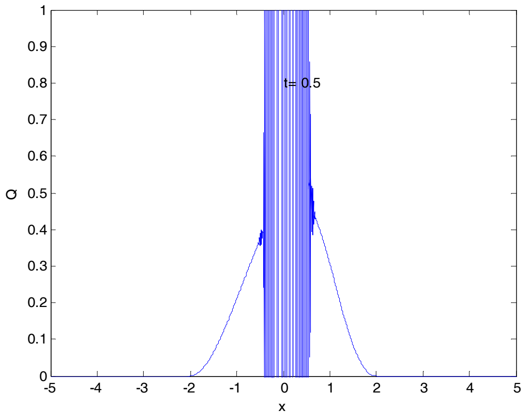
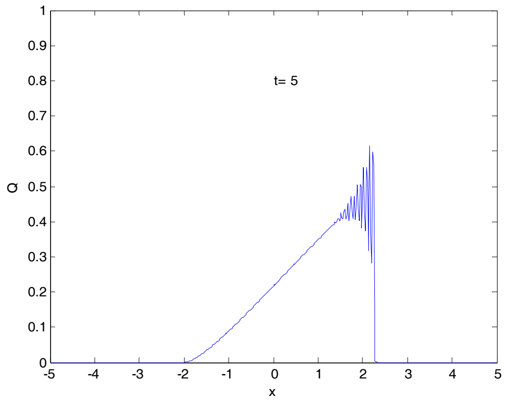
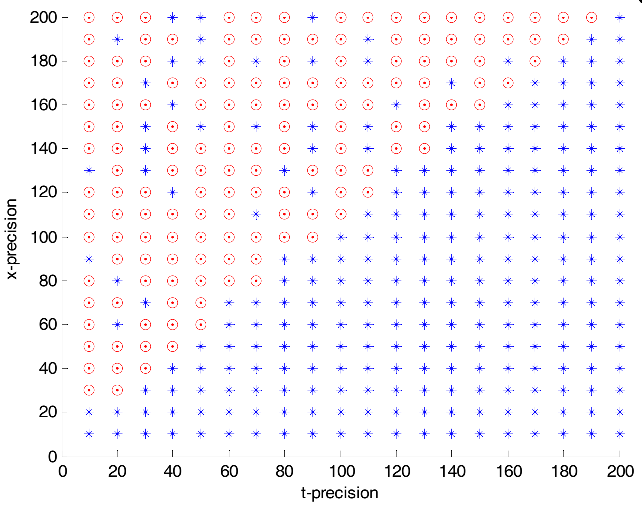


Figure 2: Using fourth order FD at t=5 and t=10





Next, the entropy conditions were checked for Δx and Δt. It was observed that when the *x*-precision was considerably greater than the *t*-precision, the solution would appear to completely diverge off to infinity very quickly as shown in Figure 4a. However, when the *t*-precision was far greater than the *x*-precision, the chaos was a bit more acceptable as shown in Figure 4b. One property of the solutions to Burger's equation is that the area beneath the curve must remain constant over time. In the first case of chaos (as in Figure 4a), which I will define as type I chaos, the area under the curve diverges off to infinity. Surprisingly however, for the second type of chaos (as in Figure 4b), which I will define as type II chaos, the area under the curve remains constant over time. Because there is always some degree of type II chaos in the solutions computed by the FD formula, only solutions exhibiting type I chaos are considered chaotic solutions. Hence the variance of the area under the curve over time suffices as a measure of chaos for these solutions.



Under this definition of chaos, it would be interesting to know for which precisions[3] of *x* and *t* will the solution be chaotic. Figure 5 shows the results from the experiment of testing for chaos over a wide variety of precisions. The *x*-axis represents the *t*-precision and the *y*-axis represents the *x*-precision used. The graph shows a red circle whenever the calculated solution was chaotic, and a blue star otherwise. There is a clear linear relationship between the two indicating that the entropy condition has the form $\Delta t < \alpha \Delta x$. In this case it appears that $\alpha = 1$. Hence this gives the restriction on the grid to avoid type II entropy. This is good because it implies that as long as one chooses an

[3] x- and t- precision in this case are defined as 1/dx and 1/dt respectively.

adequate ratio between Δt and Δx, the grid can contiuously be refined without having to worry about type I chaos. As one can tell, Figure 5 shows several anomolies in the data set. Luckily, every anamoly is a stable solution which that we would have expected to be chaotic.

The calculation time as a function of the precision is tested next. We will revert to keeping that $\Delta t = \Delta x$ and will change both by the same amount when refining the grid. For this study, calculation time was defined as the time it took to initialize the solution matrix, plus the time to fill in the initial and boundary condtions, plus the time to required to iteratively fill in the solution matrix using the finite difference formula. Figure 6 displays the results of this experiment where the x-axis represents the grid percision, which is measured as the inverse of the product $\Delta t \Delta x$, and the y-axis represents the time required to compute the solution.

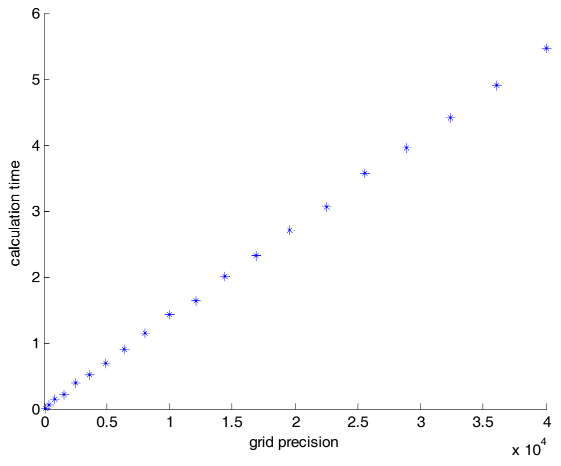


Figure 6: Grid precision versus calculation time

There is a clear linear dependence on the calculation time versus grid precision. This implies a quadratic relationship between the precision of one of the dimensions, assuming *x*- and *t*- precision are held equal to eachother.

While this finite difference method may offer very close approximations to the solution as the order is increased, there will always remain a small amount of chaos and imperfection in the solution around the shock. Further, to increase the order is very costly in terms of time. The calculation times, The table below shows the calculation times of the solution up to $t = 10$, using $\Delta x = \Delta t = 0.01$, for a second, third and fourth order FD formula. The true relationship cannot be determined from these three data point but clearly the calculation time more than doubled for each added order. This suggests possible exponential growth on calculation time as order is increased.

Order	Calculation Time
2	0.281
3	1.437
4	3.157

Part II: Finite Difference Method

The same experiments and analysis were used to test the performance of the finite volume method. First let us derive the necessary formulae that will be used. The idea behind the finite volume method is to calculate the flux across the cells of the spatial grid and change the function values over time according to these fluxes. The advantage here is that the derivative of the solution is not needed in the calculation of the solution. The formula is derived through integration techniques rather than derivative approximations. Suppose the space and time dimensions are broken up into uniformly spaced intervals of size Δx and Δt respectively. Denote the *i*th spatial interval as $X_i = (x_{i-1/2}, x_{i+1/2})$ and the *n*th time interval as $T_n = (t_{n-1}, t_n)$. Let us start with equation 1 and integrate both sides with respect to *x* on the *i*th spatial interval.

$$\int_{X_i} q_t dx + \frac{q^2(x_{i+1/2},t) - q^2(x_{i-1/2},t)}{2} = 0 \quad \text{or} \quad \frac{d}{dt}\int_{X_i} q dx = \frac{q^2(x_{i-1/2},t) - q^2(x_{i+1/2},t)}{2} \tag{6}$$

Next, integrate both sides with respect to *t* on the *n*th *t*-interval and divide both sides of by Δx.

$$\frac{1}{\Delta x}\int_{X_i} q(x,t_n)dx - \frac{1}{\Delta x}\int_{X_i} q(x,t_{n-1})dx = \frac{1}{\Delta x}\left(\int_{T_n} \frac{q^2(x_{i-1/2},t)}{2}dt - \int_{T_n} \frac{q^2(x_{i+1/2},t)}{2}dt\right) \tag{7}$$

The value of $\frac{1}{\Delta x}\int_{X_i} q(x,t_n)dx$ is the average value of the function *q* at the time t_n in the interval X_i. Thus, as the grid is refined, $\frac{1}{\Delta x}\int_{X_i} q(x,t_n)dx \approx Q_i^n$. Further, let us define the *flux* across the *i*th spatial cell as $F(x_{i-1}, x_i) = \frac{1}{\Delta t}\int_{T_n} \frac{q^2(x_{i-1/2},t)}{2}dt$. Now, plugging this into equation 7 gives the basic finite volume formula.

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}\left(F(x_i, x_{i+1}) - F(x_{i-1}, x_i)\right) \tag{8}$$

A lot of literature has been written regarding how to approximate the flux. The method that seemed most suitable to Burgers equation is the Local Lax-Fredrichs method. For this method the flux is approximated as follows.

$$F(x_{i-1}, x_i) = \frac{1}{2}\left(\frac{(Q_{i-1}^n)^2 + (Q_i^n)^2}{2} - \max(Q_{i-1}^n, Q_i^n)\cdot(Q_i^n - Q_{i-1}^n)\right) \tag{9}$$

Finally, plugging this definition of flux into equation 8 yields the numerical formula used to calculate solutions to Burgers equation with the finite volume method. It should be noted that this method is only a first order approximation of the flux.

The same initial conditions depicted in Figure 1 will be used and the boundary are assumed to remain constant over time. Fix $\Delta x = \Delta t = 0.01$, and let the solution run until $t = 5$. Figure 7 displays the result of this experiment and remarkably, there is absolutely no chaos. The solution is perfectly smooth and appears to be relatively accurate. This is impressive considering the solution is only first order accurate and its calculation wasn't very time consuming. To check how far the solution could be carried before becoming chaotic, higher values of t were solved for and the solution remained smooth through t=50.

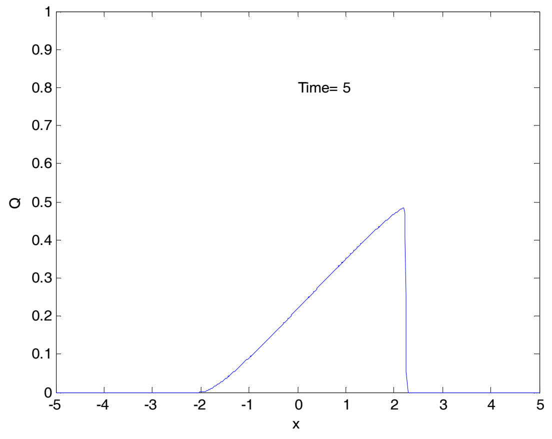


Figure 7: Finite Volume method at t=5

By the Lax and Wendroff theorem, the solution being converged to by this method as the grid is refined is the actual weak solution to Burgers equation. Figure 8 displays the computed solution for different refinements of the grid where the solution at t=5, for all refinements, are given on the same graph. Clearly there is a convergence and the slope at the shock approaches vertical, as it should.

Setting $\Delta x = \Delta t = 0.01$ again, we measure the variation of the area beneath the curve over time. The most that the area differed from the initial area at a given time had an order of magnitude of -14. This is very close to the machine accuracy so it is assumed that area was conserved.

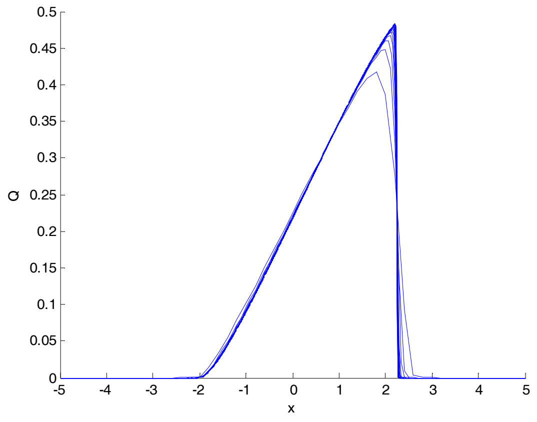


Figure 8: Finite Volume Method for various grid precisions

Because of the high accuracy of the finite volume method, it is not necessary to distinguish between type I and type II chaos. Type I chaos is still exhibited for grids with a much higher x-precision than t-precision, however type II entropy is never seen as the solutions, when not type I chaotic, are always perfectly smooth. The analogue of Figure 4 is given for the finite volume method in Figure 9. Figure 9a displays type I entropy in the case when x-precision is much greater than t-precision. It does not appear to be much different than the type I entropy observed with the finite difference method. Figure 9b shows a case where the t-precision is much greater than the x-precision. While the relative accuracy has not been computed, Figure 9b is smooth and a much better approximation of the weak solution than Figure 4b.

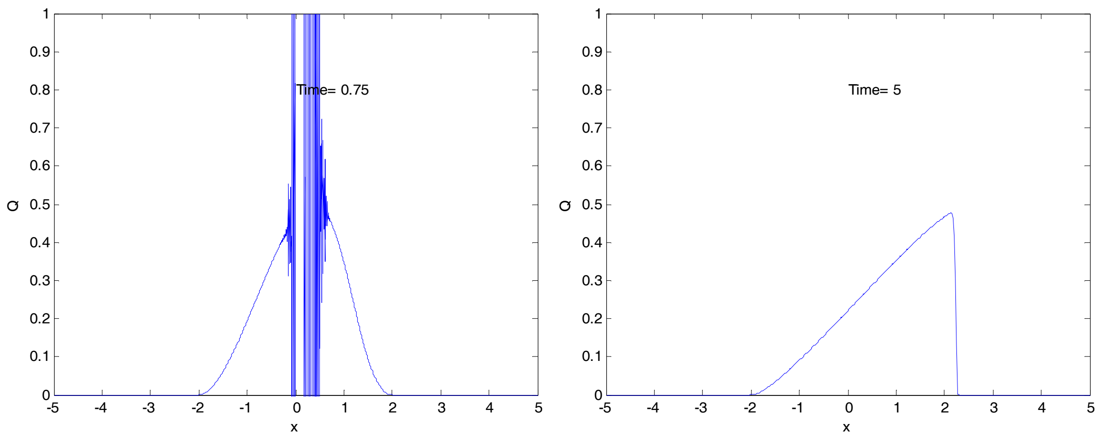


Figure 9: (a) Type I chaos. Much greater x-precision (b) No chaos despite much greater t-precision

As with the finite difference method, we can test the entropy restrictions on the grid for this finite volume method. Figure 10 shows the result of this experiment. The criterion for chaos used here is the variation of the area beneath the curve over time. There is a very obvious linear relationship dividing the chaotic and non-chaotic solutions, and what's more pleasing is that there are absolutely no anomalies. The entropy condition for this method has the form $\Delta t < \alpha \Delta x$ where in this case it appears that $\alpha = 1$. This is good because it implies that we can find a stable solution no matter how refined the grid is, so long as an appropriate ration between the *x*- and *t*-precision is chosen.

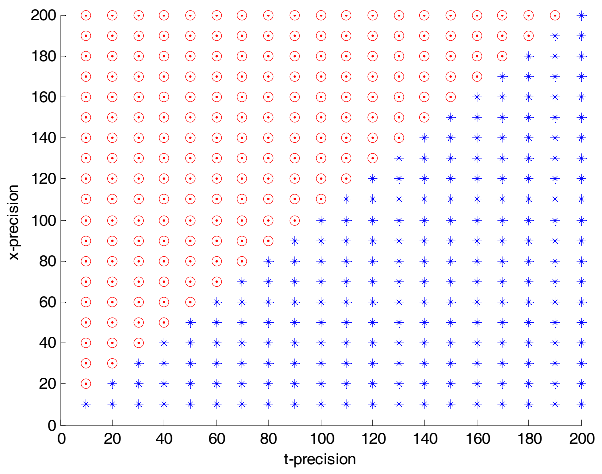


Figure 10: Entropy plot of x-precision versus t-precision

Next we test the calculation time against grid precision. The calculation time here is defined the same way as it was for the finite difference method. Figure 11a shows the results of this experiment where the *x*-axis represents the grid precision and the y-axis represents the runtime. Strangely, there is not a distinct linear relationship between the grid precision and calculation time of the solution. The solution seems almost linear for less refined grids but then becomes more chaotic for more well refined grids. This is disturbing because it doesn't give a clear upper bound for run. All of the times are faster than the corresponding times for the finite difference method. The longest calculation took just under 2 seconds; where as the longest in the finite difference method took just under 6 seconds. It cannot be said with certainty weather this trend will continue for all grid precisions. To investigate this interesting result further, more refine grids were looked at and Figure 11b displays the resulting graph. The points at higher values still appear somewhat uncertain. All calculations take less than 4 seconds, still less than some of the longer calculations of the finite difference method.

There is no clear computational explanation for this inconsistency in runtimes other than the fact that the finite volume method uses several subroutines to calculate the flux.

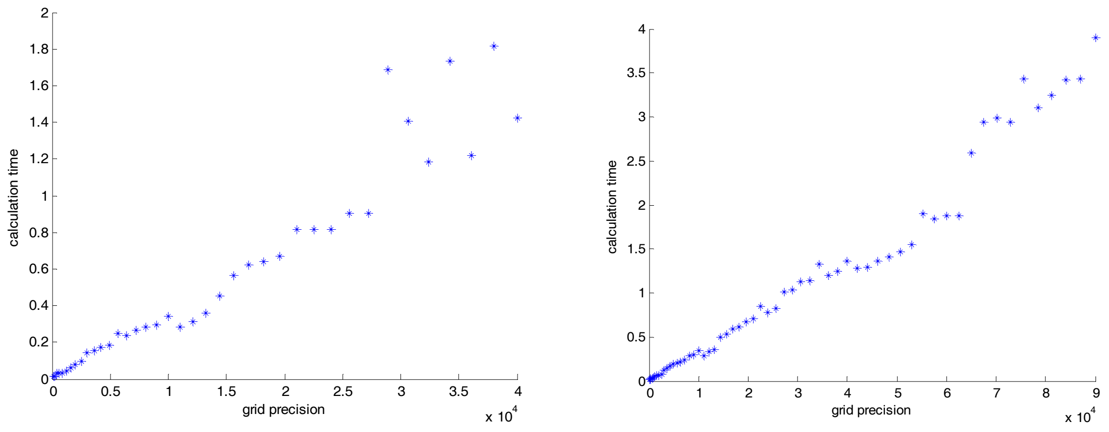


Figure 11: (a) Grid precision versus run time. (b) Run time for much more refined grids.

The problem of obtaining a higher order approximation for the finite volume method is considered to be a hard one and is not investigated in this paper. Thus we cannot compare runtime to the order of method as was done with the finite difference method.

Conclusion:

After thorough testing and experimentation on both methods, the finite volume method is clearly the preferred. Not only does it offer stable solutions at the first order approximation, it also computes the solution much more quickly than comparably accurate finite difference methods. As the grid is refined we know the finite volume method converges to the correct solution; however the only problem encountered was the runtime versus the grid precision. The runtime appeared to be fairly unpredictable, as more refinements were made to the grid, which is problematic because it will be difficult to place an upper bound on the runtime as we'd like to do when making extremely accurate calculations of the solution. None-the-less, for the grids that were tested, the finite volume method always performed better in terms of time and accuracy.

The entropy condition for both methods were precisely the same however we assumed that type II entropy was not really chaos. The fact that the finite volume method never exhibited type II entropy makes it a far more accurate solution over grids within the restriction. The finite volume method is definitely worth looking into further for equations in general with non-hyperbolic solutions.

Appendix A: (Matlab Code)

```
% Finite Differences Method on Conservative PDE
%
% Determines the solution to a PDE of the following form:
%
% q_t + (f(q))_x = 0
%
% Subject to the initial/boundary conditions:
%
% u(q,0) = a(x) ; u(q_0,t) = u(q_N,t) = 0 ;
%
% The functions f(u) and a(x) are to be determined by the user.  After the
% solution is solved for it can be graphed as a plane in 3D or be shown as
% an animation on the 2D plane.



function stable = finite_difference(xprec, tprec)

stable = true;

% Define the mesh upon which the solution will be solved

xprec = 100;
tprec = 100;

Tinterval = [0 10];
Xinterval = [-5 5];

X = (Xinterval(2) - Xinterval(1))*xprec;
T = (Tinterval(2) - Tinterval(1))*tprec;


dt = (Tinterval(2) - Tinterval(1))/T;
dx = (Xinterval(2) - Xinterval(1))/X;

t1=clock;

% Initialize the solution matrix

Q = zeros(T+1, X+1);

% Input the initial values using the initial condition a(x)

Q(1,(1:X-1)+1) = a(Xinterval(1)+(1:X-1)*dx);
```

```
% Input the boundary values using the boundary conditions b(t) and c(t).

Q((0:T)+1,1) = b(Tinterval(1)+(0:T)*dt, Xinterval(1));
Q((0:T)+1, X+1) = c(Tinterval(1)+(0:T)*dt, Xinterval(2));



%Area = dx*sum(Q(1,2:X+1));

% Fill in the rest of the matrix using the finite difference formula

% Basic Second Order Finite Difference: Quasi-Linear Form

%Q(2,(2:X))=Q(1,(2:X)) - (dt/(2*dx))*Q(1, (2:X)).*(Q(1, (2:X)+1)-Q(1,(2:X)-1));
%for n = 2:T
%    Q(n+1,(2:X))=Q(n-1,(2:X)) - (dt/dx)*Q(n, (2:X)).*(Q(n, (2:X)+1)-Q(n,(2:X)-1));
%end

%Basic Second Order Finite Difference: Conservative Form

%Q(2,(2:X))= Q(1,(2:X)) - (dt/(4*dx))*(Q(1, (2:X)+1).^2-Q(1,(2:X)-1).^2);
%for n = 2:T
%    Q(n+1,(2:X))=Q(n-1,(2:X)) - (dt/(2*dx))*(Q(n, (2:X)+1).^2-Q(n,(2:X)-1).^2);
%end

% Fourth Order Taylor Using Finite Differences

for n = 1:T
 Q(n+1,2)= Q(n,2) - (dt/(4*dx))*(Q(n,2+1).^2-Q(n,2-1).^2) +
(dt^2/(6*dx^2))*(Q(n,2+1).^3-2*Q(n,2).^3+Q(n,2-1).^3) -
(dt^3/(48*dx^3))*(-3*Q(n,2+4).^4+14*Q(n,2+3).^4 -
24*Q(n,2+2).^4+18*Q(n,2+1).^4-5*Q(n,2).^4);

Q(n+1,(3:X-1))= Q(n,(3:X-1)) –
(dt/(4*dx))*(Q(n,(3:X-1)+1).^2-Q(n,(3:X-1)-
1).^2)+(dt^2/(6*dx^2))*(Q(n,(3:X-1)+1).^3-2*Q(n,(3:X-1)).^3+Q(n,(3:X-
1)-1).^3) - (dt^3/(48*dx^3))*(Q(n,(3:X-1)+2).^4-2*Q(n,(3:X-
1)+1).^4+2*Q(n,(3:X-1)-1).^4-Q(n,(3:X-1)-2).^4);
    Q(n+1,X)= Q(n,X) - (dt/(4*dx))*(Q(n,X+1).^2-Q(n,X-
1).^2)+(dt^2/(6*dx^2))*(Q(n,X+1).^3-2*Q(n,X).^3+Q(n,X-1).^3)-
(dt^3/(48*dx^3))*(3*Q(n,X-4).^4-14*Q(n,X-3).^4+24*Q(n,X-2).^4-18*Q(n,X-
1).^4+5*Q(n,X).^4);

   diffArea = dx*sum(Q(n+1,2:X+1))-Area;
   if (diffArea > 0.01)
      stable = false;
      break;
   end

end
```

```
if (stable)
    ['xprec=',num2str(xprec),' tprec=',num2str(tprec),' STABLE']
else
    ['xprec=',num2str(xprec),' tprec=',num2str(tprec),' UNSTABLE']
end

t2 = clock;

% Display solution time

if (t1(5) == t2(5))
    runtime = t2(6)-t1(6);
else
    runtime = 60-t1(6)+t2(6);
end

['It took ',num2str(runtime),' seconds to calculate the solution.']


if (stable)

     fps = 12;
     timestep = floor(tprec/fps);
     i = 1;

     figure(1);
     for k=1:timestep:T+1
         time=floor(Tinterval(1)+(k-1)*dt);
         plot(Xinterval(1)+(1:X+1)*dx-dx,Q(k,:)');
         axis([-5 5 0 1]);
         text(0,0.8,['t= ',num2str(time)]);
         xlabel('x');
         ylabel('Q');
          mov(i) = getframe(gca);
          i = i+1;
      end

      movie(mov,1,fps*3);
end

return
end



% Initial condition q(x, 0) = a(x)

function y = a(x)
    y=0.5*(x>=-2 & x<=2).*(cos(x*pi/2)/2+0.5);
return
end
```

```
% Right boundary condition q(0, t) = b(t)

function y = b(t, k)
[m,n] = size(t);
y=zeros(1,n)+a(k);
return
end

% Left boundary condition q(X, t) = c(t)

function y = c(t, k)
[m,n]=size(t);
y=zeros(1,n)+a(k);
return
end
```

```
% Finite Volume Method on Conservative PDE
%
% Determines the solution to a PDE of the following form:
%
% q_t + (f(q))_x = 0
%
% Subject to the initial/boundary conditions:
%
% u(q,0) = a(x) ; u(x_0,t) = b(t) ; u(x_n,t) = c(t)
%
% The functions f(u), a(x), b(t), c(t) are to be determined by the user.  After the
% solution is solved for it can be graphed as a plane in 3D or be shown as
% an animation on the 2D plane.  For this program the initial/boundary
% conditions are assumed to be those described above.

 function runtime = finite_volume(xprec, tprec)

% Define the mesh upon which the solution will be solved



stable = true;

%xprec = 100;
%tprec = 100;

Tinterval = [0 5];
Xinterval = [-5 5];

X = (Xinterval(2) - Xinterval(1))*xprec;
T = (Tinterval(2) - Tinterval(1))*tprec;


dt = (Tinterval(2) - Tinterval(1))/T;
dx = (Xinterval(2) - Xinterval(1))/X;

t1 = clock;

% Initialize the solution matrix

Q = zeros(T+1, X+1);

% Input the initial values using the initial condition a(x)

Q(1,(1:X-1)+1) = a(Xinterval(1)+(1:X-1)*dx);


% Input the boundary values using the boundary conditions b(t) and c(t).

Q((0:T)+1,1) = b(Tinterval(1)+(0:T)*dt, Xinterval(1));
Q((0:T)+1, X+1) = c(Tinterval(1)+(0:T)*dt, Xinterval(2));

maximum = max([Q(1,:), Q(:,1)', Q(:,X+1)']);
```

```
% Fill in the rest of the matrix using the finite volume formula

for n = 1:T
    Q(n+1, 2:X) = Q(n, 2:X) - (dt/dx)*(flux(Q(n,2:X),Q(n,(2:X)+1)) - flux(Q(n,(2:X)-1),Q(n,2:X)));
    if (max(Q(n+1,:) > (maximum+0.1)) == 1)
        stable = false;
        Q = 0;
        break;
    end
end

%display time it took to solve pde.

t2=clock;

if (t1(5) == t2(5))
    runtime = t2(6)-t1(6)
else
    runtime = 60-t1(6)+t2(6);
end

['time=',num2str(runtime),' seconds']

if (stable)

    % Graph the solution in real time

     fps = 12;
     timestep = floor(tprec/fps);
     i=1;

     figure(1);
     plot(Xinterval(1)+(1:X+1)*dx-dx,Q(1,:)');

     axis([-5 5 0 1]);
     for k=2:timestep:T+1
         time=floor(Tinterval(1)+(k-1)*dt);
         plot(Xinterval(1)+(1:X+1)*dx-dx,Q(k,:)');
         axis([-5 5 0 1]);
         xlabel('x');
         ylabel('Q');
         text(0,0.8,['Time= 5']);
         mov(i)=getframe(gca);
         i=i+1;
     end

     movie(mov,1,fps*3)

end

 return
end
```

```
% Initial condition q(x,0) = a(x)

function y = a(x)
    y=1*(x>=-2 & x<=2).*(cos(x*pi/2)/2+0.5);
return
end

% Right boundary condition q(0, t) = b(t)

function y = b(t, k)
[m,n] = size(t);
y=zeros(1,n)+a(k);
return
end

% Left boundary condition q(X, t) = c(t)

function y = c(t, k)
[m,n]=size(t);
y=zeros(1,n)+a(k);
return
end

% The conservative flux function

function u = f(q)

    u = (q.^2)/2;  % Berger's equation

return
end

% The derrivative of the conservative flux function

function u = ff(q)

u=q;    % Berger's Equation

return
end

% The flux algorithm impliments the Lax-Friedrichs and Local Lax-Friedrichs
% Methods

function F=flux(Q1, Q2)

F = 1/2*(f(Q1) + f(Q2) - max(abs(ff(Q1)), abs(ff(Q2))).*(Q2-Q1));

return
end
```

References

Leveque, Randall. Finite Volume Methods for Hyperbolic Problems. Cambridge: Cambridge University Press, 2002.

Jacobsen, Jon. Apr. 2006. Harvey Mudd College

Yong, Darryl. Apr. 2006. Harvey Mudd College