Solving the 1D Nonlinear Shallow Water Equations Dam-Break Problem *

Johnny Sellers

June 5, 2018

Abstract

The one-dimensional nonlinear shallow water equations modeling a dam break are solved with a first order wave propagation scheme−a Godunov method−using Roe-averaged quantities. Results are compared to solutions produced with a second-order scheme using wave limiters as well as the Riemann solvers of Clawpack [6]. The first order method proved stable for Courant numbers less that $1/2$, and it generated less oscillatory results than did the second order method with or without wave limiting. The Clawpack software produced the most accurate approximations (i.e., those closest to representing the physical reality) due to its more sophisticated methods discussed in §3.2. Issues concerning the determination of numerical stability and convergence regarding nonlinear systems of conservation laws are briefly discussed with solution validation, and references are provided for further reading on these topics.

1 Introduction

Other than providing a good numerical test set, the shallow water equations (SWEs) have come to be known for their value in modeling wave propagation. In fact, most tsunami warning centers rely on shallow water theory-based modeling.[1] These equations accurately describe open-water tsunami modeling (while near-shore tsunamis require more complex systems) as well as provide valuable insight into atmospheric modeling and dam break situations. The latter is the focus of this report.

The SWEs can be derived from conservation laws. The one-dimensional SWEs (1.1) are obtained by considering incompressible fluid flow in a channel of unit width, assuming that velocity in the vertical direction is negligible and that horizontal velocity is constant throughout any cross section of the channel. The assumptions about velocity follow from considering wave amplitudes that are shallow relative to wavelength.[2] In this way the SWEs are obtained from depth-integrating the Navier-Stokes (or Euler's) equations.

*This report was for an assignment given in the course, AMATH 586: Numerical Analysis of Initial-Value Problems, with instructor Dr. Anne Greenbaum at the University of Washington-Seattle. I decided to write the report on the SWEs because they had been an interest of mine for some time. There are various forms of the equations and physical models that I hope to investigate in the future.

The SWEs are also called the Saint-Venant equations and may be derived from Newton's second law; see Appendix A of *Modeling and Control of Hydrosystems* by Litrico et al.

In this report, the approach to solving the one-dimensional nonlinear SWEs is based on Randy LeVeque's wave propagation method. Results computed using the first-order upwind formula of Godunov's method are compared with those produced by a high-resolution method obtained by employing a second order correction term with wave limiters. The Clawpack software package is also discussed and used to compute solutions for comparison.

2 The Dam-Break Problem

The SWEs in conservative form with the aforementioned assumptions may be written as

$$\begin{bmatrix} h \\ hu \end{bmatrix}_t + \begin{bmatrix} uh \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = 0, \tag{1.1}$$

or more generally as a nonlinear PDE system conservation law,

$$q_t + f(q)_x = 0,$$

if we let

$$q(x,t) = \begin{bmatrix} h \\ hu \end{bmatrix} = \begin{bmatrix} q^1 \\ q^2 \end{bmatrix},$$

so

$$f(q) = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix} = \begin{bmatrix} q^2 \\ (q^2)^2/q^1 + \frac{1}{2}g(q^1)^2 \end{bmatrix}. \tag{1.2}$$

The vector $f(q)$ is referred to as the *convex flux*. The *flux-Jacobian* matrix is

$$f'(q) = \begin{bmatrix} 0 & 1 \\ -u^2 + gh & 2u \end{bmatrix},$$

which has eigenvalues

$$\lambda_1 = u - \sqrt{gh}, \qquad \lambda_2 = u + \sqrt{gh},$$

and corresponding eigenvectors (which represent the characteristic curves)

$$r^1 = \begin{bmatrix} 1 \\ u - \sqrt{gh} \end{bmatrix}, \qquad r^2 = \begin{bmatrix} 1 \\ u + \sqrt{gh} \end{bmatrix}. \; [2, p.\ 255]$$

The dependence of the eigenvalues and eigenvectors on q leads to the waves of the Riemann problem moving at different speeds, thus giving rise to jump discontinuities known as *shocks* or smoothly-varying regions known as *rarefactions*. This is a consequence of $f(q)$ being a nonlinear function of q.

If we impose the following initial data that is piecewise constant with a single jump discontinuity,

$$h(x,0) = \begin{cases} h_l & x < 0, \\ h_r & x > 0, \end{cases} \qquad u(x,0) = 0,$$

(where $h_l > h_r > 0$) then we have a special case of the Riemann problem−the *dam-break problem*.[3] This problem is meant to model the fluid flow after an infinitesimally thin barrier separating two sections of fluid with different heights is removed. As the name suggests, this physically models a fluid dam breaking.

At the location of the discontinuity, $x = 0$ when time $t = 0$, we have a "left" state, $q_l = (h_l, (hu)_l)^T$, and a "right" state, $q_r = (h_r, (hu)_r)^T$. The Riemann problem is characterized by determining an intermediate state q_m and how it is connected to the other two states. Figure 1 below depicts a typical situation where the wave related to the first characteristic family, r^1, is a centered rarefaction wave and the wave related to the second characteristic family, r^2, is a shock wave. It is possible to have other wave combinations, e.g., a right-going 1-shock with a left-going 2-rarefaction, but the physically correct solution to the dam-break problem consists of a left-going 1-rarefaction and a right-going 2-shock. Note that the resulting waves must satisfy the *Lax entropy condition*. See §13.7.2 of [2].

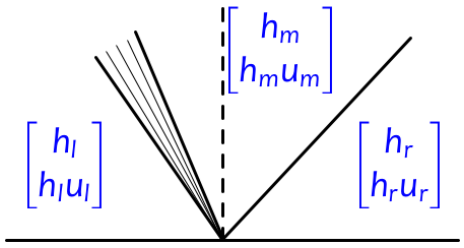


Figure 1: x-t plane showing 1-rarefaction and 2-shock and the connection of states.[3]

In order to solve the problem and obtain the intermediate state q_m, it is possible to solve this problems with an exact method or an approximate method. In practice, most use approximation methods due to their small computational workload compared to the expensive alternative of exact solvers which require solving a nonlinear equation at each cell interface for each time step.

2.1 Exact Riemann Solver

As the solution evolves, the shock wave will propagate with speed $s(t)$ that may be determined from the *Rankine-Hugoniot jump condition*,

$$s(q_r - q_l) = f(q_r) - f(q_l).$$

This condition must be satisfied across any shock wave.[2] By applying this condition to (1.1), the expressions for the Hugoniot loci are derived which can be used along with *integral curves* to determine the intermediate state q_m.

The solution varies smoothly through a rarefaction wave. That is, the variation of the solution through a rarefaction wave is proportional to the eigenvector r^p corresponding to the characteristic family of the rarefaction, and hence an expression may be derived for this variation. §13.8 of [2] discusses the derivation of integral curves based on this concept.

As noted above, the correct solution to the dam-break problem consists of a 1-rarefaction and a 2-shock. Therefore, to determine the intermediate state q_m is to find the connecting states via the intersection of an integral curve the Rankine-Hugoniot locus. I.e., to find the exact solution we must determine an intermediate state q_m that is connected to q_l by a 1-rarefaction wave and simultaneously is connected to q_r by a 2-shock wave. The state q_m must lie on an integral curve of r^1 passing through q_l , so by (13.32) of [2, p. 271] we must have

$$u_m = u_l + 2(\sqrt{gh_l} - \sqrt{gh_m}),$$

and since it must also lie on the Hugoniot locus of 2-shocks passing through q_r, by (13.19) of [2, p. 266] it must satisfy

$$u_m = u_r + (h_m - h_r)\sqrt{\frac{g}{2}\left(\frac{1}{h_m} + \frac{1}{h_r}\right)}.$$

By setting these expressions equal to each other, a nonlinear equation is obtained that may be solved at each cell interface to find q_m.

2.2 Approximate Riemann Solvers

The solution to the Riemann problem for the nonlinear SWEs may be computed by approximate solvers such as the one described in §3.1 below. These are based on the approach taken for solving the Riemann problems for linear systems.

The discontinuity $q_r - q_l$ will propagate along a characteristic direction at a speed determined by the eigenvalues of $f'(q)$. The jump discontinuity may be decomposed such that

$$q_r - q_l = \alpha^1 r^1 + \alpha^2 r^2,$$

where r^p is the pth right eigenvector of $f'(q)$, and α^p is a scalar. Then $\alpha^p r^p$ is the jump in q across the pth wave in the solution to the Riemann problem, and so we denote the waves such that

$$\mathcal{W}^p = \alpha^p r^p.$$

The exact solution in the linear case would then be

$$q(x,t) = q_l + \sum_{p:\lambda^p < x/t} \mathcal{W}^p = q_r - \sum_{p:\lambda^p \geq x/t} \mathcal{W}^p.$$

(Note that the state q_m along the $x/t = 0$ direction is the target solution.) These waves are used to compute the flux quantities at each cell interface. This relationship is described in more detail in §3.1.

In this report the case of transonic rarefactions is ignored in the numerical methods. This is where q_m will lie on the integral curve between the intersection of the Hugoniot locus and integral curve.[2, p. 314] Note that there are techniques for solving the general Riemann problem where it is unknown what the waves consist of that take this phenomenon into account. These are discussed, e.g., in §13.10 of [2].

3 Numerical Solution Method

The following finite volume method—the *flux-differencing formula*—is the basis for solutions methods used in this report:

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(F_{i+1/2}^n - F_{i-1/2}^n), \tag{3.1}$$

where Q_i^n is an approximate cell average of $q(x, t_n)$ over cell $C_i = [x_{i-1/2}, x_{i+1/2}]$, i.e.,

$$Q_i^n \approx \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} q(x, t_n)dx \equiv \frac{1}{\Delta x}\int_{C_i} q(x, t_n)dx,$$

and $F_{i-1/2}^n$ is an approximation to the average flux along $x = x_{i-1/2}$, i.e.,

$$F_{i-1/2}^n \approx \frac{1}{\Delta t}\int_{t_n}^{t_{n+1}} f(q(x_{i-1/2}, t))dt.$$

In the experiments of this report, a method based on approximate Roe solvers was used to compute the flux difference in (3.1). The approach used to solve the nonlinear system of PDEs was Godunov's method, which can be characterized by the following reconstruct-evolve-average (REA) algorithm.

REA Algorithm.

1. Reconstruct a piecewise polynomial function $\widetilde{q}(x, t_n) = Q_i^n$ for all $x \in C_i$.
2. Evolve the hyperbolic equation to obtain $\widetilde{q}(x, t_{n+1})$.
3. Average over each cell to obtain new cell averages,

$$Q_i^{n+1} = \frac{1}{\Delta x}\int_{C_i} \widetilde{q}(x, t_{n+1})dx.$$

In the wave propagation method used in this report, a simple piecewise constant function was used in step 1, leading to a first-order accurate method. The high-resolution methods implemented in the Clawpack solvers use piecewise linear functions with nonzero slopes to achieve second-order accuracy or remove oscillations in the approximation.

In order to complete the second and third steps of the algorithm, the Riemann problem must be solved at each cell interface. As mentioned in §2, an exact or approximate solver

may be employed. Once the appropriate connecting state q_m is determined, the numerical fluxes may be computed whilst avoiding solving the typically complex integral of step 3. The is done in the approach described in the following section §3.1 which falls into the category of the wave-propagation form of Godunov's method.[2] It is characterized by solving the Riemann problem through computing *fluctuations* and implementing a second order correction flux term in (3.1).

As previously mentioned, it is possible to speed up the method by using an approximate Riemann solver instead of a exact one (which would solve numerous nonlinear equations at every time step to find the intersection of the Hugonoit loci and the integral curve). According to Randy LeVeque, "on a modest 100×100 grid in two dimensions,for example, one must solve roughly 20,000 Riemann problems in every time step to implement the simplest two-dimensional generalization of Godunov's method."[2] Approximate solvers are usually more than adequate in achieving good results, so the expensive exact solvers are rarely used.

3.1 Wave Propagation Form

The following method is based on R. LeVeque's wave propagation algorithm from [4]. First, the *Roe-averaged* height and velocity are defined by

$$\hat{h} = \frac{h_{i-1} + h_i}{2},$$

and

$$\hat{u} = \frac{u_{i-1}\sqrt{h_{i-1}} + u_i\sqrt{h_i}}{\sqrt{h_{i-1}} + \sqrt{h_i}}.$$

If we let $\hat{c} = \sqrt{g\hat{h}}$, then the *Roe matrix* taken from [2] is

$$\hat{A}_{i-1/2} = \begin{bmatrix} 0 & 1 \\ -\hat{u}^2 + g\hat{h} & 2\hat{u} \end{bmatrix},$$

and its eigenvalues are

$$\lambda^1 = \hat{u} - \hat{c} \quad \text{and} \quad \lambda^2 = \hat{u} + \hat{c},$$

and its eigenvectors

$$\hat{r}^1 = \begin{bmatrix} 1 \\ \hat{u} - \hat{c} \end{bmatrix} \quad \text{and} \quad \hat{r}^2 = \begin{bmatrix} 1 \\ \hat{u} + \hat{c} \end{bmatrix}.$$

Godunov's method in wave propagation form is

$$Q_i^{n+1} = Q_i^n - \frac{\Delta t}{\Delta x}(\mathcal{A}^+ \Delta Q_{i-1/2} + \mathcal{A}^- \Delta Q_{i+1/2}),$$

where the fluctuations are defined by

$$\mathcal{A}^+\Delta Q_{i-1/2} = \sum_{p=1}^{2} (s^p_{i-1/2})^+ \mathcal{W}^p_{i-1/2},$$
$$\mathcal{A}^-\Delta Q_{i+1/2} = \sum_{p=1}^{2} (s^p_{i+1/2})^- \mathcal{W}^p_{i+1/2}.$$

Here, the wave speeds are defined by

$$(s^p_{i-1/2})^+ = \max\{\lambda^p_{i-1/2}, 0\}, \text{ and } (s^p_{i+1/2})^- = \min\{\lambda^p_{i+1/2}, 0\},$$

where λ^p are the eigenvalues defined above. The waves are defined by

$$\mathcal{W}^p_{i-1/2} = \sum_{p=1}^{2} \alpha^p_{i-1/2} r^p_{i-1/2},$$

with the α-coefficients defined by

$$\alpha^1_{i-1/2} = \frac{(\hat{u} + \hat{c})\delta^1 - \delta^2}{2\hat{c}},$$
$$\alpha^2_{i-1/2} = \frac{-(\hat{u} - \hat{c})\delta^1 + \delta^2}{2\hat{c}},$$

with $\delta \equiv Q_i - Q_{i-1}$.

Note that this definition does not guarantee that the Rankine-Higoniot condition is satisfied across each wave $\mathcal{W}^p_{i-1/2}$ because the exact Riemann solution is not used. The consequence is a nonphysical solution.[2, p. 315] However, the results in the figures of §4 exhibit behavior very close to the physical reality expected.

To extend this method to a high-resolution method, we use the fluctuations computed above in the following method,

$$Q^{n+1}_i = Q^n_i - \frac{\Delta t}{\Delta x}(\mathcal{A}^+\Delta Q_{i-1/2} + \mathcal{A}^-\Delta Q_{i+1/2}) - \frac{\Delta t}{\Delta x}(\widetilde{F}_{i+1/2} - \widetilde{F}_{i-1/2}),$$

which has the high-resolution correction terms given by

$$\widetilde{F}_{i-1/2} = \frac{1}{2}\sum_{p=1}^{2} |s^p_{i-1/2}| \left(1 - \frac{\Delta t}{\Delta x}|s^p_{i-1/2}|\right) \widetilde{\mathcal{W}}^p_{i-1/2}.$$

The quantity $\widetilde{\mathcal{W}}^p_{i-1/2}$ is the limited wave, i.e.,

$$\widetilde{\mathcal{W}}^p_{i-1/2} = \phi(\theta)\mathcal{W}^p_{i-1/2}$$

for some *TVD* limiter $\phi(\theta)$. (TVD, or *total variation diminishing*, refers to techniques for reducing oscillations in the solution by measuring the change in the solution and adjusting after each time step. (See Chapter 6 of [2].) The limiter function $\phi(\theta)$ "assesses local

variation in the solution" in order to prevent spurious oscillations near discontinuities or steep gradients.[5] There are various limiters−minmod, superbee, MC, van Leer, etc.

The plots in Figure 2(a) and Figure A.1 were generated using the MC limiter:

$$\phi(\theta) = \max(0, \min((1+\theta)/2, 2, 2\theta)),$$

where

$$\theta = \frac{Q_i^n - Q_{i-1}^n}{Q_{i+1}^n - Q_i^n}.$$

The wave limiter is implemented in lines 114 to 129 of the Python code in Appendix B.

3.2 Clawpack Solvers

The methods used in the Clawpack software [6] are more sophisticated in that they use techniques such as slope limiting, entropy fixes, and others to solve the SWEs. Numerous wave limiters are available to use, and the various physical parameters associated with the SWEs such as flow over a dry bed or wet bed, bathymetry, initial conditions, and boundary conditions can be tuned as well. The approximate Riemann solvers in Clawpack handle transonic rarefactions, shock collisions, and contact discontinuities also−facets of the Riemann problem that ensure accurate approximations to the physical reality. Chapter 5 of [2] discusses how to obtain and use the software package.

4 Results

For the experiments of this report, the solution was computed on the interval $[-5, 5]$ from time $t = 0$ to $t = 3$ seconds, taking the gravitational constant $g = 1$. (These parameters match those of Example 13.4 of §13.2 in [2].) The number of grid points used for all computations was $m = 100$, and the plots of Figures 2 and 3 were computed with the number of time steps $n = 60$.

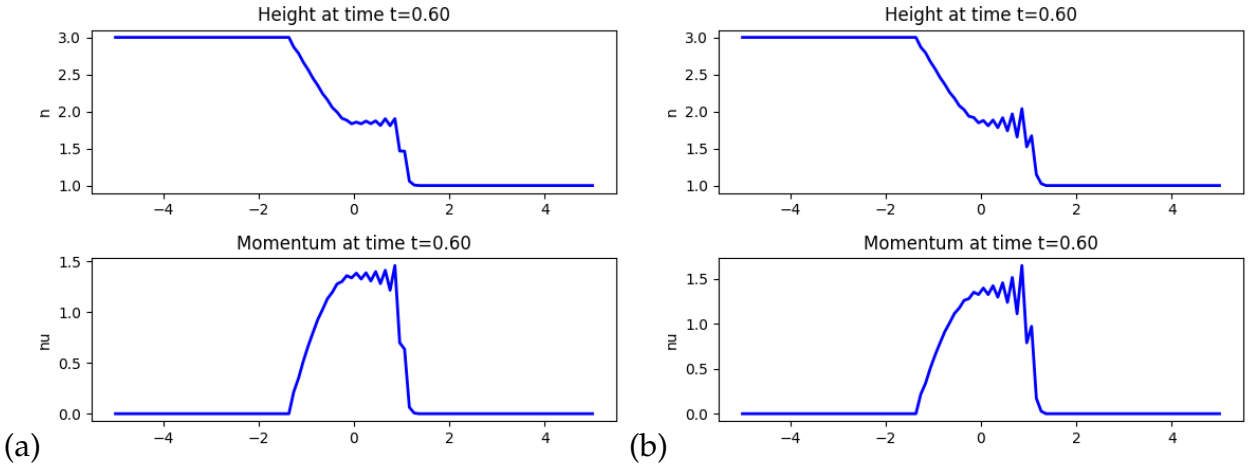
The Courant number is given by

$$c = \frac{s_{max}\Delta t}{\Delta x}.$$

In these experiments, the Courant number was taken such that $c \leq 1/2$, since for Courant numbers greater than ~ 0.505, the solution blew up after only one time step, no matter what grid spacing or time step size used. A consistent computation for the maximum wave speed was $s_{max} = 1$, so in order to maintain the desired Courant number, the time step was taken to be $dt = tfinal/n$.

Figure 2 below shows the solution computed at time $t = .6s$ using the second order correction terms (3.3) in (3.2) with the MC wave limiter and and with no wave limiter− the latter results in the second order Lax-Wendroff method. As expected, the second order method produces oscillatory approximations to the discontinuous solution. Also

observed in Figure 2(a) is how the MC wave limiter reduces the amplitude of the oscillations. §6.6 of [2] discusses the generation of these oscillations in the second order method through an interpretation of the REA algorithm.



(a) (b)

Figure 2: Solution of second order scheme (a) using the MC wave limiter and (b) without a limiter.

Figure 3 below shows the solution computed with the first order upwind scheme at times $t = 0,\ .6,\ 1.5,\ 3s$. The approximation remains relatively smooth up until the end of the time interval. The plot in Figure 4 demonstrates how these oscillations can be easily eliminated with increasing the number of time steps to $n = 100$. The Courant number is then $c = 0.303$, decreasing from $c = 0.505$ with the previous $n = 60$ time steps.

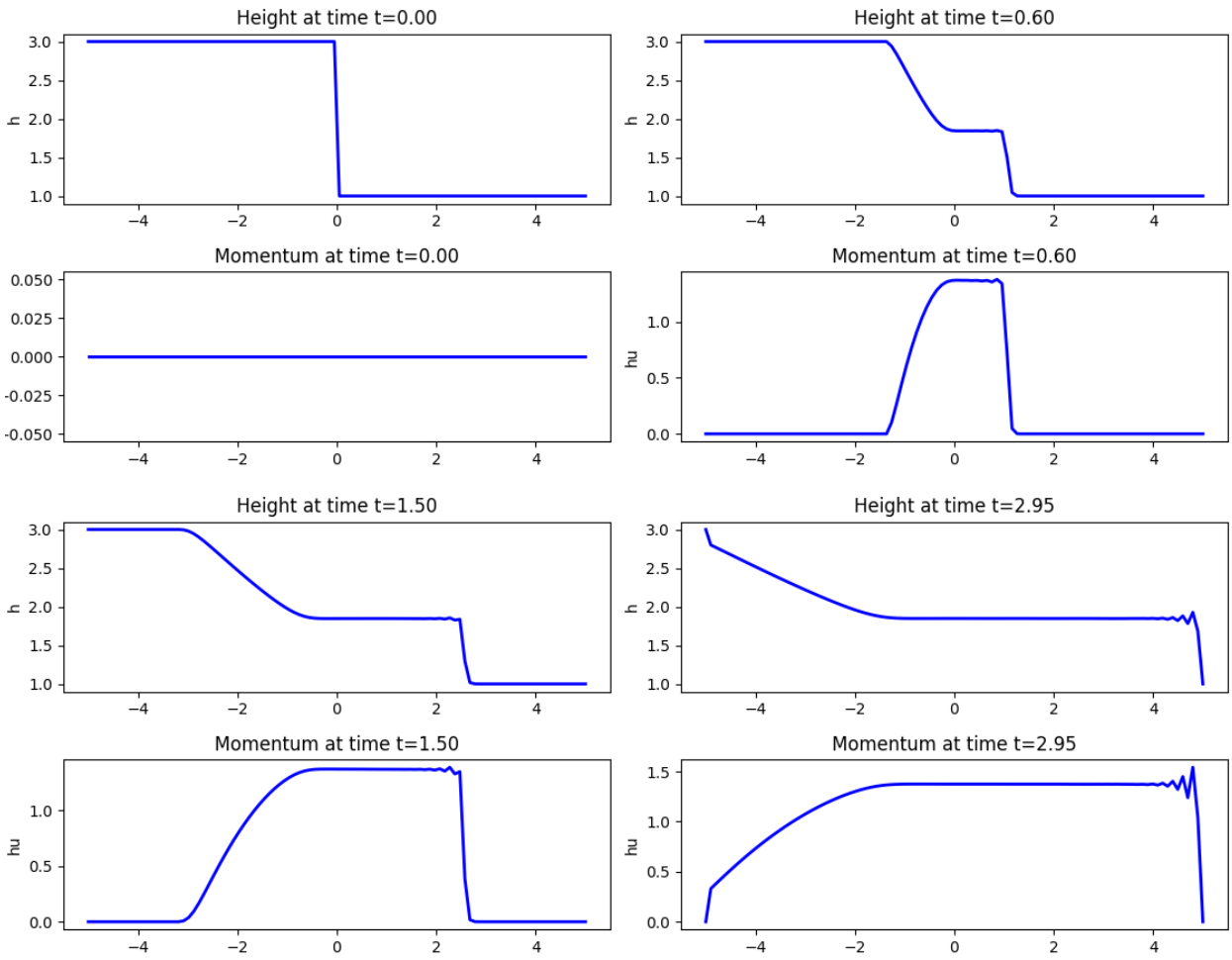


Figure 3: Solution of first order upwind scheme with $n = 60$ time steps and $c = 0.505$.

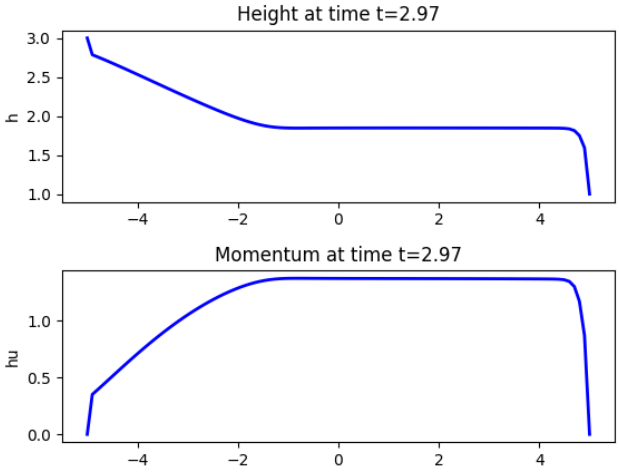


Figure 4: Solution of first order upwind scheme with $n = 100$, $c = 0.303$.

The plots in Figure 5 below and those in Figure A.1 of Appedix A were generated using the Clawpack software and the code in Appendix C which was taken from the PyClaw examples folder of [6]. This code is slightly modified to test different order solvers and limiters (lines 37 and 38, respectively). The number of grid points was also changed from $m = 500$ to $m = 100$ for these trials.

Figure 5 shows solutions computed using a second order method with the minmod wave limiter. The plots in Figure A.1 of Appendix A show those computed with Clawpack's first order solver using the MC slope limiter. These first order approximations are visually smoother that the second order ones in Figure 5.

Clawpack outperforms the wave propagation method implemented in the code of Appendix B by removing oscillations and thereby producing an approximation closer to physical reality. There is no doubt this performance is due to its implementation of entropy fixes, slope limiters, and the other facets that were discussed in §3.2.

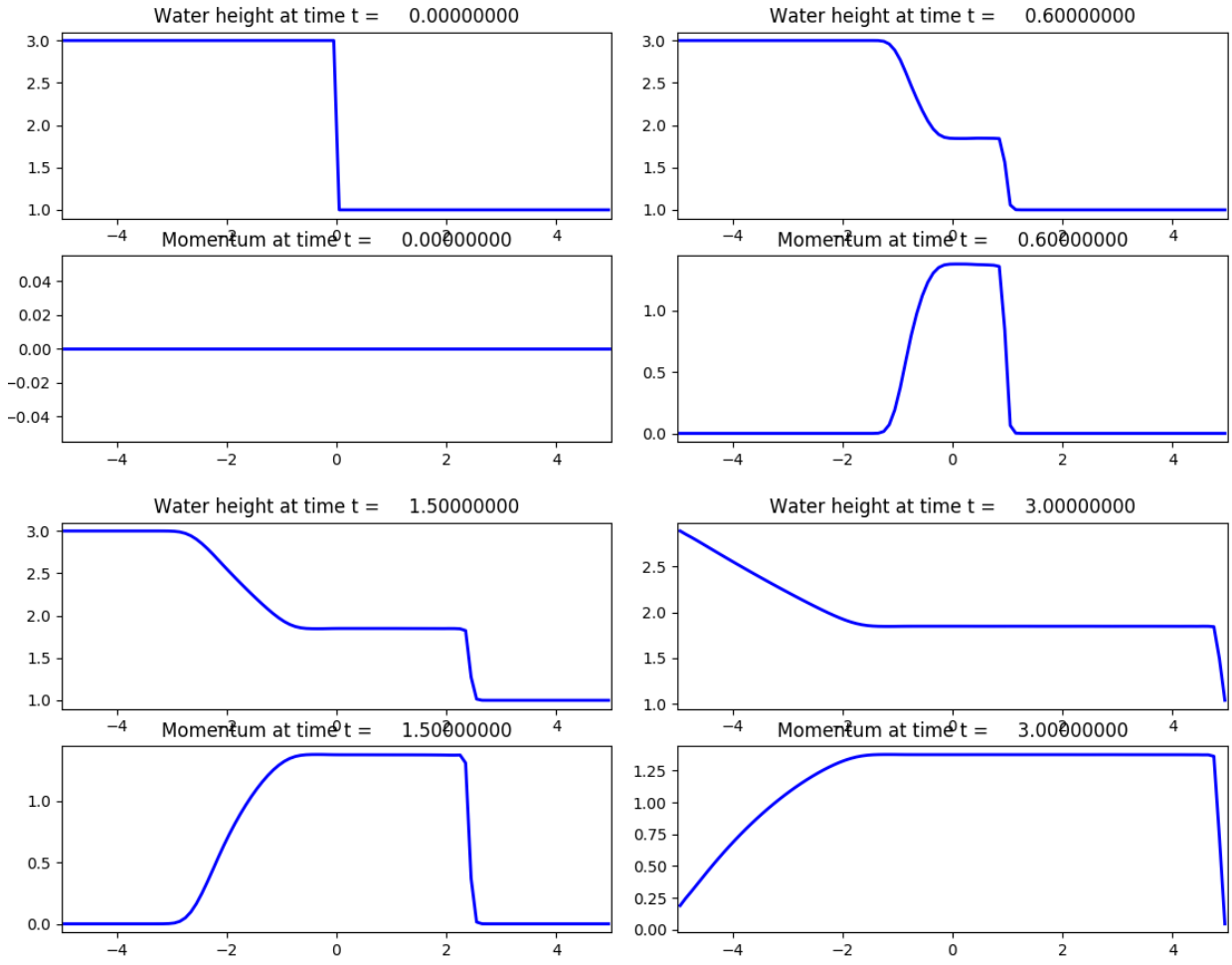


Figure 5: Solutions at times $t = 0,\ .6,\ 1.5,\ 3s$ computed with Clawpack second order solver using the minmod limiter.

4.1 Method Convergence and Solution Validation

The notions of consistency that applied to the upwind and Lax-Wendroff finite difference schemes apply here as well, but there are significant issues to address in determining convergence with nonlinear systems of conservation laws. In this regard, an investigation of convergence to a weak solution is most relevant since a differential equation of the form $q_t + f(q)_x = 0$ is "not valid in the classical sense" for solutions that are discontinuous, i.e., for those containing shocks. E.g., the weak form allows investigations into the convergence of finite volume methods as the grid is refined. See §11.11 and §12.10 of [2].

The Lax-Wendroff Theorem is one tool for determining convergence. It states that if a sequence of approximations converges, then the limit is a weak solution to the conservation law. However, in order to conclude that a method converges, some form of stability is needed.[2, p. 240] Chapter 12 of [2] discusses a form of nonlinear stability that is based on "total-variation" bounds that leads to proving convergence for some specific TVD methods. However, it is important to note the statement from [2] p. 245: "For general systems of equations with arbitrary initial data no numerical method has been proved to be stable or convergent in general."

Fortunately, intuition can play a large role in examining accuracy. For example, in the dam break problem of this report, it is obvious that neither negative fluid heights nor negative momentums would occur in reality. Also, it is safe to say that oscillations should not occur in the results. An obvious shortfall of the approximation methods of this report are their inability to model wave breaks. Nevertheless, the conclusion reached in this report is that the computed results of Figures 4, 5, and A.1 are sufficiently accurate and useful models of a physical dam break situation.

References

[1] Jorn Behrens. *Surprisingly Rich in Applications: The Shallow Water Equations and Numerical Solution Methods*. Siam News Blog, 2017.

[2] Randy LeVeque. *Finite Volume Methods for Hyperbolic Equations*. Cambridge University Press, 2002.

[3] D. Ketcheson, R. LeVeque, and M. Sarmina. *The Riemann Problem for Hyperbolic PDEs: Theory and Approximate Solvers*. http://www.clawpack.org/riemann book/html/Index.html. *In progress*.

[4] Randall LeVeque. *Wave propagation algorithms for multidimensional hyperbolic systems*. Journal of Computational Physics, 131(2):327-353, 1997.

[5] David L. George. *Augmented riemann solvers for the shallow water equations over variable topography with steady states and inundation*. Journal of Computational Physics, 227(6):3089-3113, 2008.

[6] Clawpack Development Team. Clawpack software, 2017. Version 5.4.0.

A Clawpack 1st Order Plots

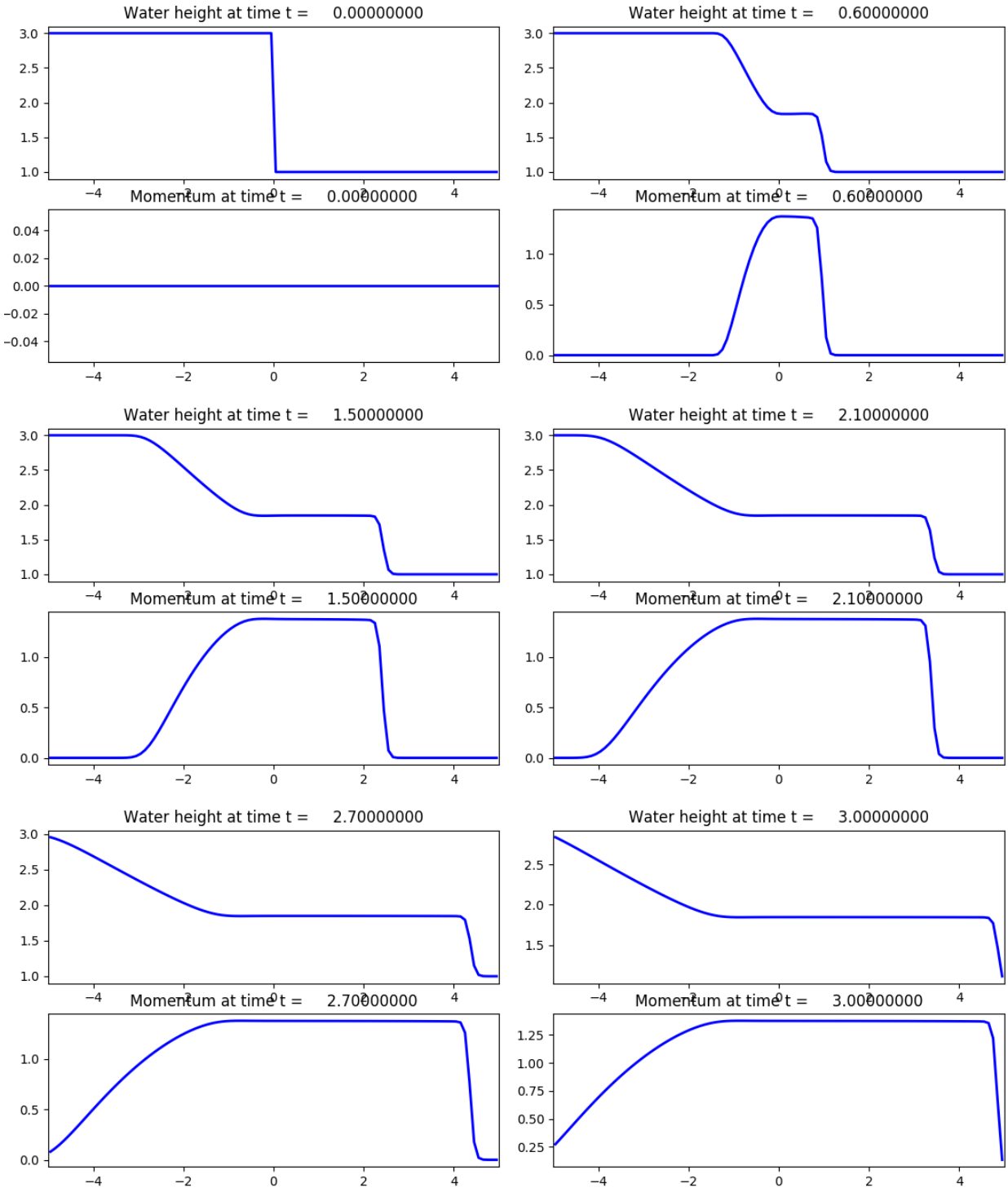


Figure A.1: Solutions at times $t = 0,\ .6,\ 1.5,\ 3s$ computed with Clawpack first order solver using the MC slope limiter.

B Wave Propagation Method Python Code

```
import numpy as np
import matplotlib.pyplot as plt
import sys


# choose first order upwind or second order method
if len(sys.argv) < 2:
print("indicate 'upwind' or 'lw'")
sys.exit()

method = str(sys.argv[1])


ax = -5
bx = 5.
tfinal = 3.
m = 100

dx = (bx-ax)/(m+1)
nsteps = 30
dt = tfinal/nsteps
cour = dt/dx


# dam-break problem
hl = 3
hr = 1
ul = 0
ur = 0
g  = 1


Q = np.zeros((2,m))
Qnp1 = np.zeros(nsteps).tolist()
smax = np.zeros(nsteps)

# set ICs
Q[0,:int(m/2)] = hl
Q[0,int(m/2):] = hr

# define flux function
def flux(U):
```

```
43    q1 = U[0]
44    q2 = U[1]
45    f = np.zeros(2)
46    f[0] = q2
47    f[1] = np.power(q2,2)/q1 + g*np.power(q1,2)/2
48    return f
49
50  # time stepping loop
51  for n in range(nsteps):
52    flux_arr = np.zeros((2,m))
53    Fcorr = np.zeros((2,m))
54
55    for i in range(1,m-1):
56      him1 = Q[0,i-1]
57      hi   = Q[0,i]
58      hip1 = Q[0,i+1]
59      uim1 = Q[1,i-1]/him1
60      ui   = Q[1,i]/hi
61      uip1 = Q[1,i+1]/hip1
62
63
64      # Roe-averaged quantities at left cell boundary
65      h_avg_m1 = (him1 + hi)/2
66      u_avg_m1 = (np.sqrt(him1)*uim1 + np.sqrt(hi)*ui)/(np.
            sqrt(him1) +
67      np.sqrt(hi))
68      c_m1 = np.sqrt(g*h_avg_m1)
69      lambda1_m1 = u_avg_m1 - c_m1
70      lambda2_m1 = u_avg_m1 + c_m1
71
72
73      # Roe-averaged quantities at right cell boundary
74      h_avg_p1 = (hip1+hi)/2
75      u_avg_p1 = (np.sqrt(hip1)*uip1 + np.sqrt(hi)*ui)/(np.
            sqrt(hip1) +
76      np.sqrt(hi))
77      c_p1 = np.sqrt(g*h_avg_p1)
78      lambda1_p1 = u_avg_p1 - c_p1
79      lambda2_p1 = u_avg_p1 + c_p1
80
81
82      # wave speeds
83      sp1_m1 = np.maximum(lambda1_m1,0.)
84      sp2_m1 = np.maximum(lambda2_m1,0.)
85      sp1_p1 = np.minimum(lambda1_p1,0.)
```

```
86      sp2_p1 = np.minimum(lambda2_p1,0.)
87
88
89      delta_m1 = np.subtract(Q[:,i],Q[:,i-1])
90      delta_p1 = np.subtract(Q[:,i+1],Q[:,i])
91
92      # alpha-coefficients of (15.39) p. 322, LeVeque[2]
93      alpha1_m1 = ((u_avg_m1+c_m1)*delta_m1[0] - delta_m1[1])
           /2/c_m1
94      alpha2_m1 = (-(u_avg_m1-c_m1)*delta_m1[0] + delta_m1[1])
           /2/c_m1
95      alpha1_p1 = ((u_avg_p1+c_p1)*delta_p1[0] - delta_p1[1])
           /2/c_p1
96      alpha2_p1 = (-(u_avg_p1-c_p1)*delta_p1[0] + delta_p1[1])
           /2/c_p1
97
98
99      # compute fluctuations
100     Ap_m1h = np.zeros(2)
101     Am_p1h = np.zeros(2)
102     Ap_m1h[0] = sp1_m1*alpha1_m1 + sp2_m1*alpha2_m1
103     Ap_m1h[1] = sp1_m1*alpha1_m1*lambda1_m1 + sp2_m1*
           alpha2_m1*lambda2_m1
104     Am_p1h[0] = sp1_p1*alpha1_p1 + sp2_p1*alpha2_p1
105     Am_p1h[1] = sp1_p1*alpha1_p1*lambda1_p1 + sp2_p1*
           alpha2_p1*lambda2_p1
106
107     Afluc = np.add(Ap_m1h,Am_p1h)
108
109
110     if method == "lw":
111       # compute MC wave limiter
112       if delta_m1[0] == 0 or delta_m1[1] == 0:
113         theta = [1., 1.]
114       else:
115         theta = np.divide(delta_p1,delta_m1)
116
117       phi_1 = [.5,.5] + np.divide(theta,2)
118       phi_2 = [2., 2.]
119       phi_3 = np.multiply(theta,2)
120       phi_i = np.minimum(phi_1, phi_2, phi_3)
121       phi = np.maximum(0,phi_i)
122       limiter = phi
123
124     elif method == "upwind":
```

```
125         limiter = [0, 0]
126
127
128       # compute second order correction
129       Fm1h = np.zeros(2)
130       Fp1h = np.zeros(2)
131       Fm1h[0] = np.absolute(lambda1_m1)*(1-cour*np.absolute(
              lambda1_m1)) \
132       *limiter[0]*(alpha1_m1 + alpha2_m1)/2
133       Fm1h[1] = np.absolute(lambda2_m1)*(1-cour*np.absolute(
              lambda2_m1)) \
134       *limiter[1]*(alpha1_m1*lambda1_m1 + alpha2_m1*lambda2_m1
              )/2
135       Fp1h[0] = np.absolute(lambda1_p1)*(1-cour*np.absolute(
              lambda1_p1)) \
136       *limiter[0]*(alpha1_p1 + alpha2_p1)/2
137       Fp1h[1] = np.absolute(lambda2_p1)*(1-cour*np.absolute(
              lambda2_p1)) \
138       *limiter[1]*(alpha1_p1*lambda1_p1 + alpha2_p1*lambda2_p1
              )/2
139
140       dFcor = np.subtract(Fm1h,Fp1h)
141       Fcorr[0][i] = cour*dFcor[0]
142       Fcorr[1][i] = cour*dFcor[1]
143
144
145       # apply Courant number to flucutations
146       flux_arr[0][i] = cour*Afluc[0]
147       flux_arr[1][i] = cour*Afluc[1]
148
149
150     # update cell average
151     Qtild = np.subtract(Q,flux_arr)
152     Qnp1[n] = np.subtract(Qtild,Fcorr)
153     Q = Qnp1[n]
154
155     # store maximum wave speed at left cell boundary
156     smax[n] = np.maximum(lambda1_m1,lambda2_m1)
157
158 # maximum wave speed
159 smaxi = np.amax(smax)
160
161
162 # choose time step to plot
163 N = 12
```

```
time = dt*N

x = np.linspace(ax,bx,m)

plt.figure(1)
plt.subplot(211)
plt.plot(x,Qnp1[N][0],'b',linewidth=2.0)
plt.title('Height at time t=%1.2f' %time)
plt.ylabel('h')

plt.subplot(212)
plt.plot(x,Qnp1[N][1],'b',linewidth=2.0)
plt.title('Momentum at time t=%1.2f' %time)
plt.ylabel('hu')

plt.subplots_adjust(hspace=.4)
plt.show()
```

C Modified Clawpack Example Code [6]

```
#!/usr/bin/env python
# encoding: utf-8

r"""
Shallow water flow
==================

Solve the one-dimensional shallow water equations:

.. math::
    h_t + (hu)_x &= 0 \\
    (hu)_t + (hu^2 + \frac{1}{2}gh^2)_x &= 0.

Here h is the depth, u is the velocity, and g is the
    gravitational constant.
The default initial condition used here models a dam break.
"""

from __future__ import absolute_import
import numpy as np
from clawpack import riemann
from clawpack.riemann.shallow_roe_with_efix_1D_constants
    import depth, momentum, num_eqn
```

```
22
23 def setup(use_petsc=False,kernel_language='Fortran',outdir='
   ./_output',solver_type='classic'):
24
25     if use_petsc:
26         import clawpack.petclaw as pyclaw
27     else:
28         from clawpack import pyclaw
29
30     if kernel_language == 'Python':
31         rs = riemann.shallow_1D_py.shallow_1D
32     elif kernel_language == 'Fortran':
33         rs = riemann.shallow_roe_with_efix_1D
34
35     if solver_type == 'classic':
36         solver = pyclaw.ClawSolver1D(rs)
37         solver.order = 1
38         solver.limiters = pyclaw.limiters.tvd.MC
39     elif solver_type == 'sharpclaw':
40         solver = pyclaw.SharpClawSolver1D(rs)
41
42     solver.kernel_language = kernel_language
43
44     solver.bc_lower[0] = pyclaw.BC.extrap
45     solver.bc_upper[0] = pyclaw.BC.extrap
46
47     xlower = -5.0
48     xupper = 5.0
49     mx = 100
50     x = pyclaw.Dimension(xlower,xupper,mx,name='x')
51     domain = pyclaw.Domain(x)
52     state = pyclaw.State(domain,num_eqn)
53
54     # Gravitational constant
55     state.problem_data['grav'] = 1.0
56     state.problem_data['dry_tolerance'] = 1e-3
57     state.problem_data['sea_level'] = 0.0
58
59     xc = state.grid.x.centers
60
61     IC='dam-break'
62     x0=0.
63
64     if IC=='dam-break':
65         hl = 3.
```

```
66          ul = 0.
67          hr = 1.
68          ur = 0.
69          state.q[depth,:] = hl * (xc <= x0) + hr * (xc > x0)
70          state.q[momentum,:] = hl*ul * (xc <= x0) + hr*ur * (
                xc > x0)
71      elif IC=='2-shock':
72          hl = 1.
73          ul = 1.
74          hr = 1.
75          ur = -1.
76          state.q[depth,:] = hl * (xc <= x0) + hr * (xc > x0)
77          state.q[momentum,:] = hl*ul * (xc <= x0) + hr*ur * (
                xc > x0)
78      elif IC=='perturbation':
79          eps=0.1
80          state.q[depth,:] = 1.0 + eps*np.exp(-(xc-x0)**2/0.5)
81          state.q[momentum,:] = 0.
82
83      claw = pyclaw.Controller()
84      claw.keep_copy = True
85      claw.tfinal = 3.0
86      claw.solution = pyclaw.Solution(state,domain)
87      claw.solver = solver
88      claw.outdir = outdir
89      claw.setplot = setplot
90
91      return claw
92
93
94  #--------------------------
95  def setplot(plotdata):
96  #--------------------------
97      """
98      Specify what is to be plotted at each frame.
99      Input:  plotdata, an instance of visclaw.data.
            ClawPlotData.
100     Output: a modified version of plotdata.
101     """
102     plotdata.clearfigures()  # clear any old figures,axes,
            items data
103
104     # Figure for depth
105     plotfigure = plotdata.new_plotfigure(name='Water height'
            , figno=0)
```

```
106
107     # Set up for axes in this figure:
108     plotaxes = plotfigure.new_plotaxes()
109     plotaxes.xlimits = [-5.0,5.0]
110     plotaxes.title = 'Water height'
111     plotaxes.axescmd = 'subplot(211)'
112     # plotaxes.axescmd = 'subplots_adjust(hspace=.5)'
113
114     # Set up for item on these axes:
115     plotitem = plotaxes.new_plotitem(plot_type='1d')
116     plotitem.plot_var = depth
117     plotitem.plotstyle = '-'
118     plotitem.color = 'b'
119     plotitem.kwargs = {'linewidth':2,'markersize':2}
120
121     # Figure for momentum[1]
122     #plotfigure = plotdata.new_plotfigure(name='Momentum',
          figno=1)
123
124     # Set up for axes in this figure:
125     plotaxes = plotfigure.new_plotaxes()
126     plotaxes.axescmd = 'subplot(212)'
127     plotaxes.xlimits = [-5.0,5.0]
128     plotaxes.title = 'Momentum'
129
130     # Set up for item on these axes:
131     plotitem = plotaxes.new_plotitem(plot_type='1d')
132     plotitem.plot_var = momentum
133     plotitem.plotstyle = '-'
134     plotitem.color = 'b'
135     plotitem.kwargs = {'linewidth':2,'markersize':2}
136
137     return plotdata
138
139
140 if __name__=="__main__":
141     # location: clawpack/pyclaw/src/pyclaw/util.py
142     from clawpack.pyclaw.util import run_app_from_main
143     output = run_app_from_main(setup,setplot)
```