FINITE VOLUME METHODS FOR ONE-DIMENSIONAL SCALAR CONSERVATION LAWS

Luis Cueto-Felgueroso

1. BACKGROUND

1.1. Problem statement

Consider the 1D scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \qquad x \in \Omega = [0, L] \qquad t > 0 \tag{1}$$

with suitable initial and boundary conditions. The *flux* $f(u)$ is, in general, a nonlinear function of $u(x)$. We will assume that the above PDE is *hyperbolic* (loosely speaking, the flux $f(u)$ does not include spatial derivatives of u). The incorporation of diffusive-like terms can be accomplished as an extension of the the finite difference framework that we studied in the previous homework, and will be considered directly in section 2 for a practical application.

1.2. Integral form and basic finite volume formulation

For problems with a predominantly hyperbolic character, it is convenient to start with the *integral form* of the conservation law (1). Considering an arbitrary *control volume* $\mathcal{I} = [x_L, x_R]$, the solution $u(x)$ satisfies

$$\frac{\partial}{\partial t} \int_{\mathcal{I}} u(x,t) dx + f(u_R) - f(u_L) = 0 \tag{2}$$

Finite volume methods attempt to compute approximations to *weak* solutions of (1) by enforcing (2) on a series of *cells* or control volumes. Thus, consider a partition of the problem domain Ω into a set of N non-overlapping cells $\{\Omega_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}], i = 1, \ldots, N\}$, such that

$$\Omega = \bigcup_{i=1}^{N} \Omega_i \tag{3}$$

We will assume uniform grid spacing, which implies that the spatial coordinates of the cell centers are given by

$$x_i = \left(\frac{1}{2} + (i-1)\right)\Delta x \tag{4}$$

whereas the interfaces are located at

$$x_{i+1/2} = i\Delta x \tag{5}$$

The basic finite volume formulation assumes a *piecewise constant* spatial representation of the solution. In particular, first-order finite volume methods study the evolution of the average value of $u(x)$ over each control volume. We therefore define the *grid function* $\{u_i, i = 1, \ldots, N\}$, where u_i is associated to the center point of cell i, and

$$u_i(t) \approx \int_{\mathcal{I}} u(x,t)dx \tag{6}$$

Conservation on each cell thus reads

$$\frac{du_i}{dt} + f(u_{i+1/2}) - f(u_{i-1/2}) = 0 \qquad i = 1, \ldots, N \tag{7}$$

In addition to the idea of working with cell-averaged values u_i, a particular finite volume method is characterized by the specific definition (approximation) of the interface fluxes, $f(u_{i+1/2})$ and $f(u_{i-1/2})$. In general, we may define a *numerical flux function* $\mathcal{H}(u^-, u^+)$, which is *consistent* with the flux $f(u)$, in the sense that

$$\mathcal{H}(u,u) = f(u) \tag{8}$$

In terms of this numerical flux function, equation (7) reads

$$\frac{du_i}{dt} + \mathcal{H}(u^-_{i+1/2}, u^+_{i+1/2}) - \mathcal{H}(u^-_{i-1/2}, u^+_{i-1/2}) = 0 \qquad i = 1, \ldots, N \tag{9}$$

What does (9) mean? Recall that the numerical solution is piecewise constant, and hence discontinuous across interfaces. This implies that the fluxes $f(u_{i-1/2})$ and $f(u_{i+1/2})$ are not uniquely determined with the information at hand and, therefore, we need to construct suitable approximations to their values. The key feature of finite volume methods is that these approximations and defined exploiting some known properties of nonlinear conservation laws like (1). In particular, the fact that they usually describe *nonlinear advection* processes (thus with a distinctive directionality), and tend to develop discontinuities (shocks). And here is where the numerical fluxes enter: given the values $u^-_{i+1/2}$ and $u^+_{i+1/2}$ on each side of the interface $i+1/2$, the numerical flux function $\mathcal{H}(u^-, u^+)$ returns an *upstream* approximation to the actual flux $f(x_{i+1/2})$. The "canonical form" of the numerical flux may be written as

$$\mathcal{H}(u^-, u^+) = \frac{1}{2}\left(f(u^+) + f(u^-)\right) - \frac{1}{2}|\widetilde{a}|\left(u^+ - u^-\right) \tag{10}$$

Furthermore, let us define the *advection speed* at the interface, $\widetilde{a}$, as

$$\widetilde{a}(u^-, u^+) = \begin{cases} \dfrac{f(u^+) - f(u^-)}{u^+ - u^-} & u^+ \neq u^- \\ f'(u^-) & u^+ = u^- \end{cases} \tag{11}$$

where $f'(u) = \dfrac{df(u)}{du}$.

First order schemes retain the basic piecewise-constant structure, and therefore

$$\begin{aligned} u^-_{i+1/2} &= u_i \\ u^+_{i+1/2} &= u_{i+1} \\ u^-_{i-1/2} &= u_{i-1} \\ u^+_{i-1/2} &= u_i \end{aligned} \tag{12}$$

which results in a semi-discretization of the form

$$\frac{du_i}{dt} + \mathcal{H}(u_i, u_{i+1}) - \mathcal{H}(u_{i-1}, u_i) = 0 \qquad i = 1, \ldots, N \tag{13}$$

with the numerical fluxes defined as in (10)–(11). The above expression is a system of ordinary differential equations of the form

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{R} \tag{14}$$

where $\boldsymbol{u} = \boldsymbol{u}(t)$, and we have the initial condition $\boldsymbol{u}(0) = \boldsymbol{u}^0$. The system (14) can be integrated in time using standard ODE solvers, like the ones we studied during the lectures (Runge-Kutta etc.).

1.3. Higher order extensions: ENO3

Following the ideas presented in the previous section, we may develop high-order finite volume methods through the use of high-order *extrapolated* values u^- and u^+ at each interface, rather than the piecewise constant cell-averages. In the next section we will see a practical example of the implementation of the essentially non-oscillatory scheme explained in the lectures (ENO3). The basic idea is to substitute the *piecewise constant* representation of the solution by a *piecewise polynomial* reconstruction, but choosing the neighbor cells that contribute to the reconstruction (*stencil*, in such a way that the resulting reconstruction is *smooth* in an suitable sense. Once the extrapolated values u^- and u^+ at each interface have been computed, we may evaluate the numerical fluxes as in the previous section.

2. AN EXAMPLE IN MATLAB: THE BUCKLEY-LEVERETT PROBLEM

2.1. Mathematical model

Consider the following model of immiscible flow (water and oil) in 1D,

$$\frac{\partial S_w}{\partial t} + \frac{\partial}{\partial x}\left(f(S_w)u_T + \epsilon\, k_{ro}(S_w) f(S_w) J'(S_w) \frac{\partial S_w}{\partial x} \right) = 0, \qquad x \in [0, L] \tag{15}$$

In the above equation, S_w is the water saturation, u_T is the total velocity, and $f(S_w)$ is the *fractional flow* function,

$$f(S_w) = \frac{M k_{rw}}{M k_{rw} + k_{ro}} \qquad M = \frac{\mu_o}{\mu_w} \tag{16}$$

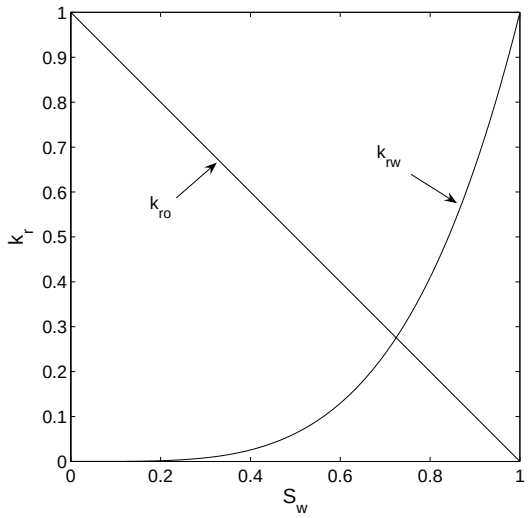

Figure 1. Relative permeability functions, corresponding to $\alpha = 4$ and $\beta = 1$ in (17).

where M is the mobility (viscosity) ratio, and k_{rw} (resp. k_{ro}) is the is the relative permeability of the wetting (resp. non-wetting) phase. The function $J(S_w)$ denotes a generic functional expression for the capillary pressure, and $J'(S_w) = \dfrac{dJ}{dS_w}$. In order to fix ideas, consider the following prototype constitutive relations

$$\begin{aligned} k_{rw}(S_w) &= S_w^{\alpha} \\ k_{ro}(S_w) &= (1 - S_w)^{\beta} \end{aligned} \tag{17}$$

and

$$J(S_w) = S_w^{-1/\gamma} \qquad \gamma > 2 \tag{18}$$

Typical values of the parameters are $\alpha = 4$, $\beta = 1$ and $\gamma = 3$. In the following sections, we will solve the problem (1) in $[0, 1]$, assuming unit, constant total velocity $u_T = 1$, and the initial condition

$$u(x, 0) = \begin{cases} 1 - \delta & x < 0.2 \\ \delta & x \geq 0.2 \end{cases} \tag{19}$$

where $\delta \ll 1$ is a small number, introduced to avoid singularities for certain capillary pressure functionals at $S_w = 0$ or $S_w = 1$. We will impose the *numerical* boundary conditions $u(0, t) = u_{LB} = 1 - \delta$ and $u(1, t) = u_{RB} = \delta$.

2.2. Low-order finite volume discretization

The code `BL11.m` computes approximate solutions to (15) using a finite volume method. The hyperbolic term is discretized using first-order upwind, whereas the capillary (diffusive) term is

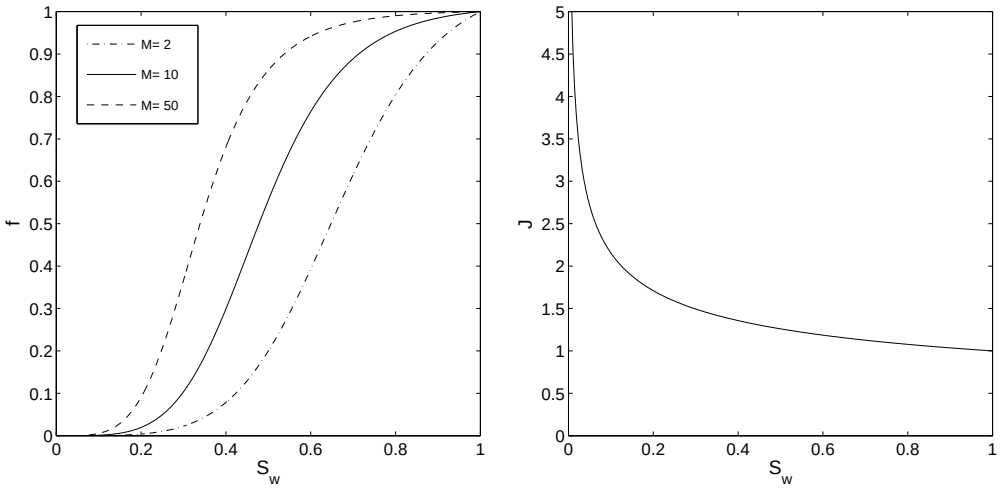

Figure 2. Fractional flow (left) and capillary pressure function (right), corresponding to $\alpha = 4$ and $\beta = 1$ in (17), and $\gamma = 3$ in (18).

aprroximated using second order centered differences. The time integration is carried out using forward Euler (FE).

Given the distinctive nature of the advective and diffusive terms, their discretization follows separate paths. Let us write the conservation law (15) as

$$\frac{\partial S_w}{\partial t} + \frac{\partial}{\partial x}\left(f_{adv}(S_w) + f_{diff}(S_w)\right) = 0 \tag{20}$$

where the hyperbolic-like (advective) flux is given by

$$f_{adv} = f(S_w)u_T \tag{21}$$

and the elliptic-like (diffusive) flux is

$$f_{diff} = -\epsilon\, k_{ro}(S_w) f(S_w)(-J'(S_w))\frac{\partial S_w}{\partial x} \tag{22}$$

or

$$f_{diff} = -\epsilon\, D(S_w)\frac{\partial S_w}{\partial x}, \qquad D(S_w) = k_{ro}(S_w) f(S_w)(-J'(S_w)) \tag{23}$$

with the above definitions, and advancing in time with FE, the finite volume scheme reads

$$\frac{\Delta x}{\Delta t}(u_i^{n+1} - u_i^n) + \left(f_{adv}^n + f_{diff}^n\right)\Big|_{x_{i+1/2}} - \left(f_{adv}^n + f_{diff}^n\right)\Big|_{x_{i-1/2}} = 0 \tag{24}$$

Rearranging the advective and diffusive contributions, we arrive at

$$\frac{\Delta x}{\Delta t}(u_i^{n+1} - u_i^n) + \left(f_{adv}^n\big|_{x_{i+1/2}} - f_{adv}^n\big|_{x_{i-1/2}}\right) + \left(f_{diff}^n\big|_{x_{i+1/2}} - f_{diff}^n\big|_{x_{i-1/2}}\right) = 0 \tag{25}$$

The advective fluxes are evaluated using the upwind numerical flux (10)–(11) and the piecewise constant representation of the saturation field. For the diffusive part, we need to approximate (22) at the interfaces, which involves 1) *interpolating* the saturations S_w at the interfaces, and 2) approximating the spatial derivatives of the saturations $\partial S_w/\partial x$ also at the interfaces. Once we have the interpolated saturations we can evaluate the "diffusion" coefficients $\epsilon\, D(S_w)$, given in (23), and multiplying by the estimated derivative we get the flux at the interface, according to (23).

Let us see how this is done in the code. We start by creating functions to evaluate the fractional flow and constitutive relations

```
syms u M alpha beta gamma
krw= u^alpha;
kro= (1-u)^beta;
f= M*krw/(M*krw+kro);
J= u^(-1/gamma);
df= diff(f,1);
dJ= diff(J,1);
D= f*kro*dJ;

Ff = inline(vectorize(simplify(f)));   %Syntax f = Ff (M,alpha,beta,u)
Fdf= inline(vectorize(simplify(df)));  %Syntax df= Fdf(M,alpha,beta,u)
FD = inline(vectorize(simplify(D)));   %Syntax D = FD (M,alpha,beta,gamma,u)
```

Note that the saturations are denoted by `u` in the code. The model parameters are defined as

```
%Model parameters
M= 20;
alpha= 4;
beta = 1;
gamma= 3;
epsilon= 0.01;
```

The initialization of the code also includes specifications about the grid (number of cells N), boundary conditions and initial solution; in particular,

```
%Grid generation
N = 100;
L= 1;
h= L/N;
x= (h/2:h:L-h/2)';

%Boundary conditions
uLB= 0.9999;
uRB= 10^-4;

%Initial solution
u= uRB + (uLB-uRB)*(x<=0.2*L);
```

Note that the grid size is denoted by h instead of Δx. In order to approximate the derivatives of the saturations at the interfaces, we may construct a differentiation matrix which, given the saturations at the cell centers and at the boundaries, provides the derivatives at the midpoints through a matrix-vector product, i.e.

$$\boldsymbol{u}' = \begin{pmatrix} u'_{1/2} \\ u'_{3/2} \\ \vdots \\ u'_{j+1/2} \\ \vdots \\ u'_{N-1/2} \\ u'_{N+1/2} \end{pmatrix} = \boldsymbol{D} \begin{pmatrix} u_{LB} \\ u_1 \\ \vdots \\ u_j \\ \vdots \\ u_N \\ u_{RB} \end{pmatrix} \tag{26}$$

where u_{LB} and u_{RB} are the prescribed saturations at the boundaries. Note that $\boldsymbol{D}$ is an $(N+1)\times(N+2)$ matrix. For interior interfaces, $x_{i+1/2}, i = 1, \ldots, N-1$, we may use second-order centered differences, as

$$u'_{i+1/2} = \frac{u_{i+1} - u_i}{\Delta x} \tag{27}$$

while at the left and right boundaries we use the first-order formulas

$$u'_{1/2} = \frac{u_1 - u_{LB}}{\Delta x/2} \qquad u'_{N+1/2} = \frac{u_{RB} - u_N}{\Delta x/2} \tag{28}$$

Using (27) and (28) we can now assemble the differentiation matrix. In the code, this is done with

```
%Differentiation matrix for the derivatives at the interfaces
%Syntax du_{interfaces}= D1*[uLB;u;uRB]
D1= spalloc(N+1,N+2,2*(N+1));
R= [-1 1 zeros(1,N-2)]/h;
C= [-1 zeros(1,N-2)]/h;
D1(2:N,2:N+1)= toeplitz(sparse(C),sparse(R));
%Boundaries...
D1(1,:)= [-1 1 zeros(1,N)]/(h/2);
D1(end,:)= [zeros(1,N) -1 1]/(h/2);
```

Now we can start advancing in time...

```
dt= 0.2*L/N;
NSTEP= 200;
for istep= 1:NSTEP;
```

Note that the choice of time step is purely heuristic. At each time step we will compute the contributions to $\boldsymbol{R}^n$ in (14) (advective+diffusive), and then just advance the saturations with a forward Euler step as

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \Delta t \boldsymbol{R}^n \tag{29}$$

Let us start with the advective terms. For each interface, we need to extrapolate the saturations from both sides of the interface, and then evaluate the upwind numerical flux. In our first-order scheme, the extrapolated saturations are just the cell center (cell-average) values

$$\begin{aligned} u^-_{i+1/2} &= u_i \\ u^+_{i+1/2} &= u_{i+1} \end{aligned}$$

$$\begin{aligned} u^-_{i-1/2} &= u_{i-1} \\ u^+_{i-1/2} &= u_i \end{aligned} \tag{30}$$

The above expression is written from the perspective of the *cells*, i.e. each cell has an $i+1/2$ interface, as well as an $i-1/2$ interface. But we can actually "recycle" the interface values between neighbor cells, as the $i+1/2$ interface (looking from cell i) is of course the same as the $(i+1)-1/2$ one (looking from cell $i+1$). Thus, it is more convenient to work from the perspective of the *interfaces*, and say that each interface $i+1/2$ has two associated saturations u^- and u^+, coming from either side of the interface. For a first-order scheme,

$$\begin{aligned} u^-_{i+1/2} &= u_i \\ u^+_{i+1/2} &= u_{i+1} \end{aligned} \tag{31}$$

bearing in mind that this interface is the *right* interface of cell i, but also the *left* interface of cell $i+1$. At the boundaries, we simply say

$$u^-_{1/2} = u_{LB} \qquad u^+_{N+1/2} = u_{RB} \tag{32}$$

In the code, the u^- and u^+ values at each interface are denoted by `um` and `up`, and they are simply

```
%Extrapolate variables to interfaces
um= [uLB;u];
up= [u;uRB];
```

Now that we have the saturations on each side of each interface, we can evaluate the advective terms using the upwind numerical flux. Given the u^- and u^+ saturations at the interfaces, the function `upwflux_BL` evaluates the upwind fluxes (10)–(11). Furthermore, the fluxes are returned by `upwflux_BL` within the *cell* perspective; i.e. for each cell i, `upwflux_BL` returns the numerical fluxes its left and right interfaces, respectively f^n_{Li} and f^n_{Ri}, such that the contribution of the advective term to R_i is, for cell i and at time level t^n

$$R_{adv} = -\frac{f^n_{Ri} - f^n_{Li}}{\Delta x} \tag{33}$$

In the code, this is done as

```
%Evaluate numerical fluxes
[fR,fL]= upwflux_BL(Ff,Fdf,um,up,M,alpha,beta);
%Inviscid term
Ri= -(1/h)*(fR-fL);
```

The next step is the computation of the diffusive part of the fluxes. We need again the saturations at the interfaces, but now a unique value for each interface. There are many possibilities, but for this first-order scheme we may just take the average of u^- and u^+, which we have already computed. With these saturations, the diffusive fluxes at the interfaces are

```
%Capillary term
ui= 0.5*(um+up);
D= FD(M,alpha,beta,gamma,ui);
fluxd=  epsilon*D.*(D1*[uLB;u;uRB]);
```

Note that the fluxes are now available within the *interface* perspective; i.e. we know the flux at each interface. It is convenient to move back again to the *cell* perspective, and collect the left and right fluxes for each cell, as

```
fluxdR= fluxd(2:N+1);
fluxdL= fluxd(1:N);
```

Finally, the contribution to $\boldsymbol{R}$ from the capillary term is

```
Rd= -(1/h)*(fluxdR-fluxdL);
```

and we can add the advective and diffusive terms as

```
%Residual
R= Ri+Rd;
```

At this point, it just remains to advance the saturations to t^{n+1}, as

```
%Forward Euler step
u= u + dt*R;
```

If you run `BL11`, you will get something like figure 3

2.3. High-order finite volume discretization

The code `BL33.m` solves the same problem using ENO3 reconstruction for the advective fluxes and a fourth-order centered discretization for the capillary term. The integration in time is carried out using a third-order Runge-Kutta method.

The structure of the code is, however, almost identical to that of the first-order one. The idea is again to discretize in space and then arrive at a system of ODE's of the form

$$\frac{d\boldsymbol{S}_w}{dt} = \boldsymbol{R}(\boldsymbol{S}_w) \tag{34}$$

The time integration proceeds then in *three stages* to arrive at the t^{n+1} saturations, starting from the t^n ones:

$$\begin{aligned} \boldsymbol{S}_w^1 &= \boldsymbol{S}_w^n + \Delta t \boldsymbol{R}(\boldsymbol{S}_w^n) \\ \boldsymbol{S}_w^2 &= \frac{3}{4}\boldsymbol{S}_w^n + \frac{1}{4}\boldsymbol{S}_w^1 + \frac{1}{4}\Delta t \boldsymbol{R}(\boldsymbol{S}_w^1) \\ \boldsymbol{S}_w^{n+1} &= \frac{1}{3}\boldsymbol{S}_w^n + \frac{2}{3}\boldsymbol{S}_w^2 + \frac{2}{3}\Delta t \boldsymbol{R}(\boldsymbol{S}_w^2) \end{aligned} \tag{35}$$

Note that the cost of each stage is roughly that of a forward Euler step (the cost of a full step with (35) is about three times higher than with FE). With this time integration in mind, we just need to be able to compute the residual $\boldsymbol{R}(\boldsymbol{S}_w)$ for given cell saturations $\boldsymbol{S}_w$, which is precisely what `BL11.m` does for low order.

The only difference in the initialization of the problem is the definition of the differentiation matrix $\boldsymbol{D}_1$. In this case, we are interested in fourth-order formulas at the midpoints, given the values of the saturations at the cell centers. For interior interfaces, this can be done using the function `fdcoefs.m` we used in homework #4, as

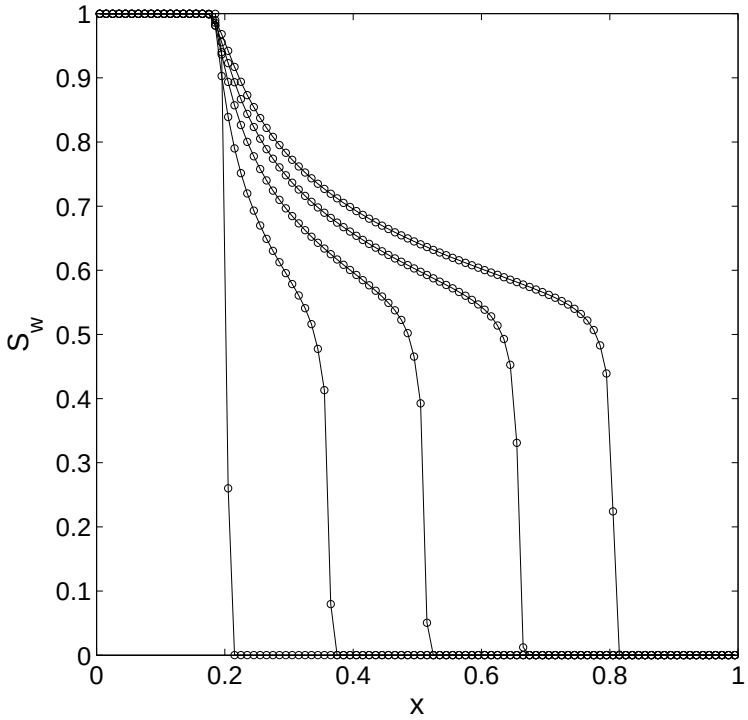

Figure 3. Solution of the Buckley-Leverett problem at various time levels using BL11.

```
>> [coefs]= fdcoefs(1,3,[0 1 2 3],1.5)

coefs =

   0.04166666666667  -1.12500000000000   1.12500000000000  -0.04166666666667
```

The above command line gives the coefficients of a formula to compute the first derivative at $i+1/2$, given the values of the function at $i-1$, i, $i+1$ and $i+2$. We may see the coefficients as fractions using the Matlab function `rat`; for example

```
>> [num,den]= rat(0.041666666667)

num =

     1


den =

    24
```

Thus, the finite difference approximation reads

$$u'_{i+1/2} = \frac{u_{i-1} - 27u_i + 27u_{i+1} - u_{i+2}}{24\Delta x} + O(\Delta x^4) \tag{36}$$

We cannot use centered formulas near the boundaries. Using `fdcoefs.m` again we may derive difference formulas for interfaces $1/2$, $3/2$, $N-1/2$ and $N+1/2$, as

$$\begin{aligned}
u'_{1/2} &= \frac{-184u_{LB} + 225u_1 - 50u_2 + 9u_3}{60\Delta x} + O(\Delta x^3) \\
u'_{3/2} &= \frac{8u_{LB} - 75u_1 + 70u_2 - 3u_3}{60\Delta x} + O(\Delta x^3) \\
u'_{N-1/2} &= \frac{-8u_{RB} + 75u_{N-1} - 70u_{N-2} + 3u_{N-3}}{60\Delta x} + O(\Delta x^3) \\
u'_{N+1/2} &= \frac{184u_{RB} - 225u_{N-1} + 50u_{N-2} - 9u_{N-3}}{60\Delta x} + O(\Delta x^3)
\end{aligned} \tag{37}$$

With these approximations in mind, the differentiation matrix can be assembled through

```
%Differentiation matrix for the derivatives at the interfaces
% Syntax du_{interfaces}= D1*[uLB;u;uRB]
D1= spalloc(N+1,N+2,4*(N+1));
R= [1 -27 27 -1 zeros(1,N-4)]/(24*h);
C= [1 zeros(1,N-4)]/(24*h);
D1(3:N-1,2:N+1)= toeplitz(sparse(C),sparse(R));
%Boundaries...
D1(2,:)= [8 -75 70 -3 zeros(1,N-2)]/(60*h);
D1(end-1,:)= [zeros(1,N-2) 3 -70 75 -8]/(60*h);

D1(1,:)= [-184 225 -50 9 zeros(1,N-2)]/(60*h);
D1(end,:)= [zeros(1,N-2) -9 50 -225 184]/(60*h);
```

Apart from the differentiation matrix, another difference between the high-order and the low-order codes is the way the saturations are extrapolated to the interfaces in order to evaluate the inviscid fluxes; i.e. the way $\boldsymbol{u}^-$ and $\boldsymbol{u}^+$ are computed. Instead of using the cell-average value, we use the ENO3 reconstruction. In he code this is specified by

```
[um,up]= ENO3rec(u,uLB,uRB);
```

We will come back later to this function. The important practical consequence is that `ENO3rec` returns the saturations at the interfaces u^- and u^+, which can be used now to evaluate the fluxes. The remaining of the residual evaluation is identical to that of `BL11`. The only difference, now due to the time integration scheme, is that these residuals are used to advance the solution within a three-stage time integration, rather than with forward Euler, and hence the lines

```
if istage==1;
    u1= u0 + dt*R;
    u= u1;
elseif istage==2;
    u2= (3/4)*u0 + (1/4)*u1 + (1/4)*dt*R;
    u= u2;
else;
    u= (1/3)*u0 + (2/3)*u2 + (2/3)*dt*R;
end;
```

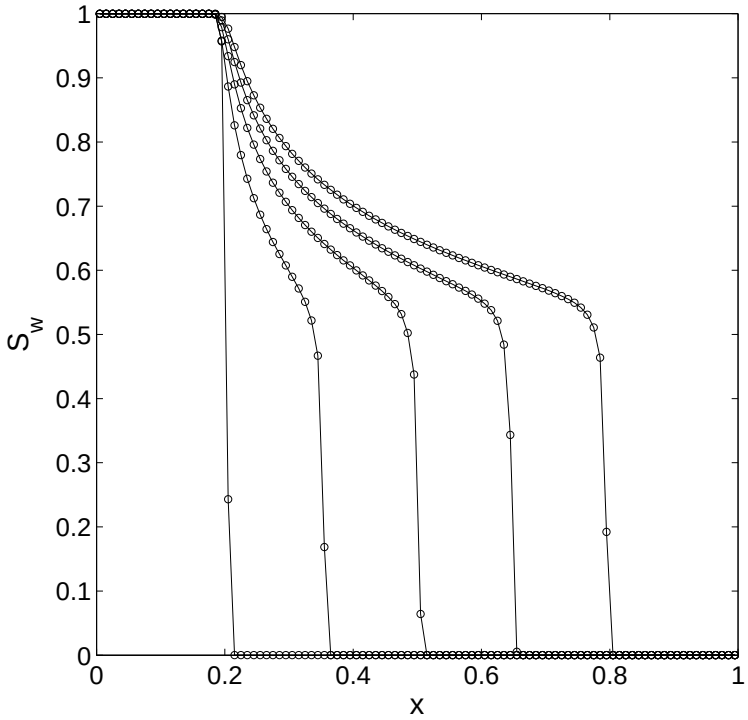

Figure 4. Solution of the Buckley-Leverett problem at various time levels using BL33.

If you run `BL33`, you will get something like figure 4. A comparison between the low and high-order solutions is plot in figure 5

2.4. The ENO reconstruction

Let us take a look at the function `ENO3rec`, which performs the ENO reconstruction of the saturations at the interfaces. The syntax is

```
>> [um,up]= ENO3rec(u,uLB,uRB,N);
```

The first step is the computation of the differences

$$
\begin{aligned}
d_{i+1/2} &= d_{i+1} - d_i \\
d_i &= \frac{d_{i+1/2} + d_{i-1/2}}{2} \\
D_i &= d_{i+1/2} - d_{i-1/2}
\end{aligned}
\tag{38}
$$

Higher-order reconstructions are based on substituting the *piecewise constant* representation of the solution by a *piecewise polynomial* reconstruction inside each cell, of the form

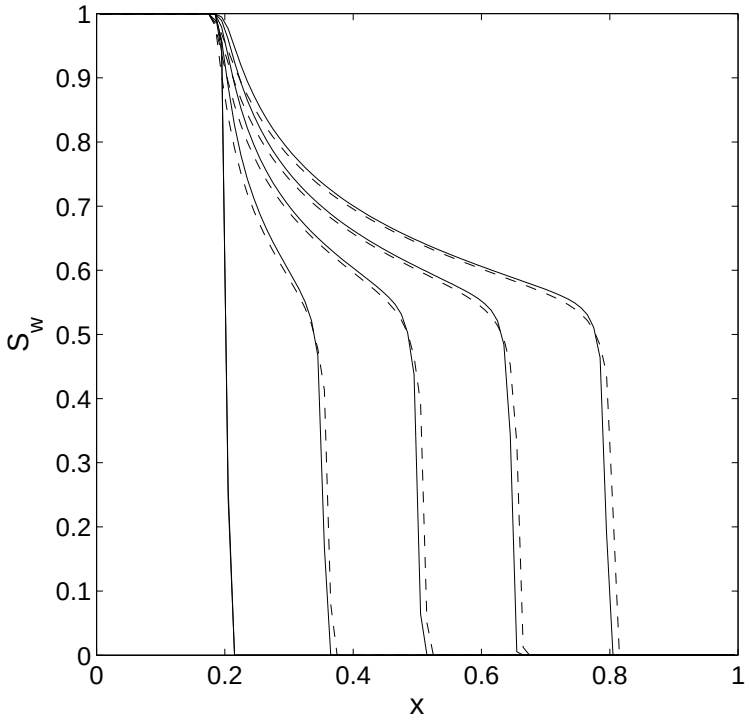

Figure 5. Comparison of ENO3 (solid) and first-order (dashed) solutions at different time levels.

$$\widehat{u}_i(x) = p_i(x) \qquad i = 1, \ldots, N \tag{39}$$

for some locally-defined polynomials $p_i(x)$. Moreover, we enforce that the reconstruction is *conservative*, in the sense that

$$\int_{x_{i-1/2}}^{x_{i+1/2}} p(x)dx = \int_{x_{i-1/2}}^{x_{i+1/2}} u_i dx = \Delta x u_i \tag{40}$$

Essentially Non-Oscillatory (ENO) methods choose the reconstructing polynomial among a certain number of candidates, based on the estimated smoothness of the resulting reconstruction. For ENO3, the reconstruction inside cell i uses three possible *stencils*: $S_1 = \{i-2, i-1, i\}$, $S_2 = \{i-1, i, i+1\}$ and $S_3 = \{i, i+1, i+2\}$. Their associated quadratic polynomials are

$$\begin{aligned}
p_1(x) &= u_i - \frac{D_{i-1}}{24} + \frac{x - x_i}{\Delta x}\left[d_{i-1/2} + \frac{D_{i-1}}{2} + \frac{D_{i-1}}{2}\left(\frac{x - x_i}{\Delta x}\right)\right] \\
p_2(x) &= u_i - \frac{D_i}{24} + \frac{x - x_i}{\Delta x}\left[d_i + \frac{D_i}{2}\left(\frac{x - x_i}{\Delta x}\right)\right] \\
p_3(x) &= u_i - \frac{D_{i+1}}{24} + \frac{x - x_i}{\Delta x}\left[d_{i+1/2} - \frac{D_{i+1}}{2} + \frac{D_{i+1}}{2}\left(\frac{x - x_i}{\Delta x}\right)\right]
\end{aligned} \tag{41}$$

Focusing on the interfaces $x_{i+1/2}$ and $x_{i-1/2}$, the extrapolated values "*from cell i*" are

$$\begin{aligned}
p_1(x_{i+1/2}) &= u_i - \frac{D_{i-1}}{24} + \frac{1}{2}\left[d_{i-1/2} + \frac{D_{i-1}}{2} + \frac{D_{i-1}}{2}\left(\frac{1}{2}\right)\right] \\
p_2(x_{i+1/2}) &= u_i - \frac{D_i}{24} + \frac{1}{2}\left[d_i + \frac{D_i}{2}\left(\frac{1}{2}\right)\right] \\
p_3(x_{i+1/2}) &= u_i - \frac{D_{i+1}}{24} + \frac{1}{2}\left[d_{i+1/2} - \frac{D_{i+1}}{2} + \frac{D_{i+1}}{2}\left(\frac{1}{2}\right)\right]
\end{aligned} \tag{42}$$

and

$$\begin{aligned}
p_1(x_{i-1/2}) &= u_i - \frac{D_{i-1}}{24} - \frac{1}{2}\left[d_{i-1/2} + \frac{D_{i-1}}{2} - \frac{D_{i-1}}{2}\left(\frac{1}{2}\right)\right] \\
p_2(x_{i-1/2}) &= u_i - \frac{D_i}{24} - \frac{1}{2}\left[d_i - \frac{D_i}{2}\left(\frac{1}{2}\right)\right] \\
p_3(x_{i-1/2}) &= u_i - \frac{D_{i+1}}{24} - \frac{1}{2}\left[d_{i+1/2} - \frac{D_{i+1}}{2} - \frac{D_{i+1}}{2}\left(\frac{1}{2}\right)\right]
\end{aligned} \tag{43}$$

Assume that we have selected one of the three candidates, $p_i(x)$. Then the above expressions yield the interface values

$$\begin{aligned}
u^-_{i+1/2} &= p_i(x_{i+1/2}) \\
u^+_{i-1/2} &= p_i(x_{i-1/2})
\end{aligned} \tag{44}$$

Following the same procedure with the all the cells we would end up with all the interface extrapolations, $u^-_{i+1/2}$ and $u^+_{i+1/2}$.

In the function `ENO3rec`, the candidate values at the interfaces are computed through

```
%Candidates for u+1/2
up1= u - Dim1/24 + 0.5*(dim12 + Dim1/2 + 0.5*Dim1/2);
up2= u - Di/24 + 0.5*(di + 0.5*Di/2);
up3= u - Dip1/24 + 0.5*(dip12 - Dip1/2 + 0.5*Dip1/2);
%Candidates for u-1/2
um1= u - Dim1/24 - 0.5*(dim12 + Dim1/2 - 0.5*Dim1/2);
um2= u - Di/24 - 0.5*(di - 0.5*Di/2);
um3= u - Dip1/24 - 0.5*(dip12 - Dip1/2 - 0.5*Dip1/2);
```

which is just an implementation of (42)–(43). The ENO selection procedure, for each cell i, goes as follows:

if $|d_{i-1/2}| \leq |d_{i+1/2}|$ **then**
  if $|D_{i-1}| \leq |D_i|$ **then**
    $\widehat{u}_i(x) = p_1(x)$
  else
    $\widehat{u}_i(x) = p_2(x)$
  end
else
  if $|D_i| \leq |D_{i+1}|$ **then**
    $\widehat{u}_i(x) = p_2(x)$
  else
    $\widehat{u}_i(x) = p_3(x)$
  end
end

In the code the above algorithm is implemented as

```
%ENO selection
a= (abs(dim12)<=abs(dip12)).*(abs(Di)>abs(Dim1));
b= (abs(dim12)<=abs(dip12)).*(abs(Di)<=abs(Dim1));
c= (abs(dim12)>abs(dip12)).*(abs(Di)<=abs(Dip1));
d= (abs(dim12)>abs(dip12)).*(abs(Di)>abs(Dip1));
um0= a.*um1 + b.*um2 + c.*um2 + d.*um3;
up0= a.*up1 + b.*up2 + c.*up2 + d.*up3;
```

What we have so far is, for each cell i, the ENO-extrapolated values $u_{i+1/2}$ and $u_{i-1/2}$. Thus, and since the reconstruction is performed locally (we do not enforce continuity of the reconstructed saturations across interfaces), we have computed two extrapolated values of the saturation at each interface: for example, at interface $x_{i+1/2}$, we have computed a saturation coming from cell i, $u^-_{i+1/2} = u_{i+1/2}$, and another one coming from cell $i+1$, $u^+_{i+1/2} = u_{(i+1)-1/2}$. These u^- and u^+ values will be introduced in the numerical flux function, so it is interesting to compute them. They are actually the arguments returned by `ENO3rec`, and are obtained as

```
%Split u^- and u^+ for each interface
um= [uLB;up0];
up= [um0(1:end);uRB];
```

REFERENCES

1. R.J. Leveque. Finite volume methods for hyperbolic problems. Cambridge University Press (2002)
2. G. Strang. Computational science and engineering. Wellesley-Cambridge Press (2007)
3. B. Fornberg. A practical guide to pseudospectral methods. Cambridge University Press (1998)
4. L.N. Trefethen. Spectral methods in MATLAB. SIAM (2001)
5. J.P. Boyd. Chebyshev and Fourier spectral methods. Dover (1999)
6. J.S. Hesthaven, S. Gottlieb and D. Gottlieb. Spectral Methods for Time-Dependent Problems. Cambridge University Press (2007)
7. J. Strikwerda. Finite Difference Schemes and Partial Differential Equations. SIAM. 2nd edition (2007)
8. T.J. Barth, H. Deconinck (eds.). High-order methods for computational physics. Lecture Notes in Computational Science and Engineering. Springer (1999)
9. E. Hairer, S.P. Nørsett and G. Wanner. Solving ordinary differential equations I: Nonstiff Problems. Springer Series in Computational Mathematics. Springer (1993)
10. E. Hairer and G. Wanner. Solving ordinary differential equations II: Stiff and Differential-Algebraic Problems. Springer Series in Computational Mathematics. Springer (1996)
11. U. Ascher and L. Petzold. Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. SIAM (1998)
12. J.D. Lambert. Numerical methods for ordinary differential systems. John Wiley & Sons (1991)