Physics 555 Assignment - finite volume methods for conservative equations

Martin Snajdr

Spring 2005

Contents

1 Using the template program for advection equation

The included program solves the advection equation in 1D using a high resolution shock capturing method (HRSC). These methods are based on finite volume approximation to the continuum equations. The program is written in Fortran77 but uses an "infrastructure" which makes it easier to read and modify. It also uses the `bbhutil` library for outputting `sdf` files.

After unpacking the distribution we should have the following directories

```
drwxr-xr-x    3 msnajdr  choptuik     4096 Mar 10 02:35 SCRIPTS
drwxr-xr-x   12 msnajdr  choptuik     4096 Mar 14 16:33 TEMPLATE
```

The directory `SCRIPTS` contains all the "infrastructure" scripts which are needed for the program to successfully compile. The `TEMPLATE` contains several additional directories and files

```
drwxr-xr-x    2 msnajdr  choptuik     4096 Mar 14 15:14 INCLUDE_ADV
-rw-r--r--    1 msnajdr  choptuik      962 Mar 10 03:00 Makefile
drwxr-xr-x    2 msnajdr  choptuik     4096 Mar 10 02:59 PROTOTYPES
drwxr-xr-x    2 msnajdr  choptuik     4096 Mar 14 16:05 PROTOTYPES_ADV
drwxr-xr-x    2 msnajdr  choptuik     4096 Mar 14 16:05 SRC_ADV
drwxr-xr-x    2 msnajdr  choptuik     4096 Mar 14 16:06 TEST
-rw-r--r--    1 msnajdr  choptuik        0 Mar 10 02:27 translation_adv.txt
-rw-r--r--    1 msnajdr  choptuik     1416 Mar 14 16:03 variables_adv.txt
```

The directory `INCLUDE_ADV` contains a Fortran include file `include.adv` that defines relevant Fortran constants.

The directory `PROTOTYPES` contains the source code. The directory `PROTOTYPES_ADV` contains symbolic links to the source files in `PROTOTYPES`. The reason for using symbolic links is that if we have more versions of a program (e.g., one solving the advection equation and the other the Burger's equation) we can share common source files since only few routines need to be modified. The two source codes will reside in two different directories, but will have symbolic links to the common source files. Therefore a change in any of the common files will be reflected in both codes.

The directory `SRC_ADV` contains the actual Fortran code that is generated from the "prototype" files. This is a "pure" Fortran code containing only Fortran statements. If the compiler reports error at certain line we should always look at the appropriate file in the `SRC_ADV` directory because that is the file the compiler actually compiles.

The directory `TEST` contains the script `run.sh` which will be described later.

The file `translation_adv.txt` must be present but is not important for us and is empty.

The role of the `variables_adv.txt` will be described in the next section.

2 What is the "infrastructure"?

2.1 Variable grouping

Non-trivial programs typically contain lots of variables and arrays. These variables need to be shared among subroutines and functions. One way is to introduce common blocks. However, this makes the program more difficult to understand (for other people or the same person later). Namely, it is not immediately clear on what variable the subroutine depends on and what variables it modifies. Moreover, the subroutines can not be easily used as "black boxes". Therefore we adopt the policy *never to use common blocks*. This, however, introduces another problem - many variables must be passed as arguments and, what is worse, if we want to introduce/erase arguments we have to modify all the calls to that subroutine. During development or in the process of extending functionality of existing programs tweaking variables is a very common thing. Moreover, if we are passing dozens of variables the code becomes messy.

The "infrastructure" helps to deal with this problem and others. It creates an abstract layer which groups variables with common purpose and are typically passed together. For instance we use different groups for variables describing the state of a system, input parameters, grid structure, etc.. This grouping is defined in the file `variables_adv.txt`. Then instead of passing every single variable we just include the name of the abstract group as argument. Therefore, if we want to introduce a new state variable we modify only the definition of the "state" group in the `variables_adv.txt` file.

The "infrastructure" also takes care of variable declarations. The same variable can be member of many groups - the "infrastructure" will make sure that the same variable will be passed only once and declared only once.

To illustrate this lets take a look at part of the file `variables_adv.txt`:

```
#define _params
real*8::rmin,rmax
real*8::inittime
integer::Np
integer::Ng
integer::bc(NDIM*2)
integer::nsteps
integer::bsign(VARDIM)
integer::evoltype
integer::inittype
real*8::lambda
integer::gridtype
integer::outmodulo
integer::recostype(VARDIM)
```

```
integer::fluxapprox
integer::level
real*8::r_c
real*8::delta_gauss
real*8::qleft
real*8::qcenter
real*8::qright
real*8::v

#define _creategrid
integer::Ncell
real*8::rmin,rmax,vertxL(MAXNCELLS),vertxC(MAXNCELLS),vertxR(MAXNCELLS),mindr
integer::Ng,Np,gridtype

#define _grid
integer::Ncell
integer::Ng,Np
real*8::vertxL(MAXNCELLS),vertxC(MAXNCELLS),vertxR(MAXNCELLS)

#define _state1
integer::Ncell
real*8::qL_1(MAXNCELLS)
real*8::qC_1(MAXNCELLS)
real*8::qR_1(MAXNCELLS)
```

The first group defined is the `_params` group. It defines the types of all the input parameters we want to specify.

It is customary to place an underscore in front of a group name so it is clear that it is a group name not a Fortran variable.

Note that the variable `Ncell` is a member of almost all groups.

Here is an example of passing the variables

```
call output(time,level,_stateC1,_grid)
```

To avoid potential problems *always pass the group names as last arguments.*

Here is the beginning of the subroutine output:

```
      subroutine output(time,level,_stateC1,_grid)
      implicit none
      include "include.adv"
c#declare(_stateC1,_grid)
```

The statement `c#declare(comma separated group names)` creates the variable declarations. Make sure that the `#declare(...)` statement is only *one line* long.

2.2 Taking care of parameter files

The “infrastructure” can also automatically create the body of the subroutine responsible for reading parameters from a file.

If the “infrastructure” encounters the statement

```
c#create <module> <parameters>
```

it calls the handler of the module `<module>` with the parameters `<parameters>`. It then pastes the output from the module into the source file.

One of the known modules is the module `readinput`. Here is the listing of the prototype file `read_param_file.f`.

```
      subroutine read_param_file(filename,_params)
      implicit none
      include "include.adv"
c#declare(_params)
      integer  i1,i2,i3
      character*80 filename
      open(UNIT=10,FILE=filename)

c#create readinput _params 10 variables_adv.txt

      close(10)
      return
      end
```

In the above example the module `readinput` creates a sequence of read statements that read variables from group `_params` which is defined in the file `variables_adv.txt`. The number 10 just identifies the Fortran UNIT number.

Each line of the input file must contain values for each variable defined in the group `_params` in the proper order. For arrays it expects a sequence of numbers.

In order to be able to specify more user friendly input file without the need to care about the parameter ordering we supply a script which converts the user friendly format to the one expected by the `read_param_file` subroutine. It is called `read_input.pl`.

Here is a listing of the parameter file `id` located in the directory `TEST` (without the parameter descriptions)

```
rmin:=0.0
rmax:=1.0
inittime:=0.0
Np:=100
```

```
Ng:=2
bc:=1 1
nsteps:=101
bsign:=1
evoltype:=3
inittype:=2
lambda:=0.5
gridtype:=1
outmodulo:=10
recostype:=1
fluxapprox:=1
level:=1
r_c:=0.5
delta_gauss:=0.1
v:=0.5
```

The rules for the parameter file are as follows:

- the "assign value" symbol is `:=`, i.e. for scalar variable `nsteps := 100`
- spaces and line breaks do not matter
- strings for character variables are enclosed in quotes `spacetime := "Kerr"`
- if a variable is defined at more than once the last value will be used
- arrays are read in the Fortran order, e.g., array defined as `A(2,3)` can be typed as

  ```
  A:= 11 21
      12 22
      13 23
  ```

- the order of the parameter initializations does not matter
- the comment character is `#`
- the script `read_input.pl` exits if some variable is not defined
- as of now arrays of characters are not supported

Here is an example of a script `run.sh` (located in the `TEST` directory which takes 2 command line arguments (the name of the parameter file and the value of the parameter level))

```
#!/bin/sh
rm *.sdf
echo "started at"
date
basedir=".."
scrdir="../../SCRIPTS"

cat $1 > temp
echo "level:=$2" >> temp
$scrdir/read_input.pl _params $basedir/variables_adv.txt temp > input
$basedir/advection input

echo "stopped at"
date
```

This script first copies the parameter file into `temp`. Then it appends a line `level:=$2` at the end of the file. It then calls the script `read_input.pl` with its arguments and redirects its output into the file `input`. This is the file which will be eventually read by the program `advection`.

3 Compiling and running the test program

Make sure that your environment is properly set up for your favorite compiler.

To compile the program go to the `TEMPLATE` directory and type

```
make
```

To clean up type

```
make clean
```

This produces an executable called `advection`.

To run the program go to the `TEST` directory and type

```
./run.sh id 0
```

This runs the program with `level` set to zere. See the input file `id` for parameter description.

4 Assignment

4.1 The template program

The template program solves the 1D advection equation

$$\frac{\partial q(t,x)}{\partial t} + v\frac{\partial q(t,x)}{\partial x} = 0 \tag{1}$$

using a finite volume method.

The meaning of the input parameters is explained in the input file `id` located in the `TESTS` directory. The group variables are described in the `variables_adv.txt` file.

The program is written in a form that is suitable for generalization (relativistic fluid in 1D).

The main program is located in the `main.f` file. The basic structure is as follows:

- read the input parameters
- generate the grid
- initialize variables
- loop over the `make_step` subroutine and output.

The `make_step` subroutine can use one of the following time steppings:

- 1-st order Euler (`evoltype:=1`)
- 2-nd order TVD Runge-Kutta (`evoltype:=2`)
- 3-rd order TVD Runge-Kutta (`evoltype:=3`)

See equations (3.22) and (3.23) of Ian Hawke's thesis (page 68) for desctiption of the second and third order TVD Runge-Kutta methods.

As a substep the `make_step` subroutine calls the `update` subroutine which takes one Euler step of size Δt:

$$q_i^{n+1} = q_i^n - \frac{\Delta t}{\Delta x}(f_{i+1/2} - f_{i-1/2}) \ . \tag{2}$$

4.2 The advection equation

Take a look at the code structure and make sure you understand it. Use the Gaussian initial profile (`inittype:=2`) to establish the order of convergence of the method using piecewise constant (`recostype:=0`) and piecewise linear (`recostype:=1`) reconstruction with slope limiter. Use all of the three time steppings available. Is the convergence of the same order everywhere within the domain? If not, why?

Implement a "naive" piecewise linear reconstruction (without slope limiter).

$$s = \frac{q_{i+1} - q_i}{x_{i+1} - x_i} \tag{3}$$

$$q_{L\,i+1/2} = q_i + \frac{1}{2} s \left(x_{i+1/2} - x_{i-1/2}\right) \tag{4}$$

$$q_{R\,i-1/2} = q_i - \frac{1}{2} s \left(x_{i+1/2} - x_{i-1/2}\right) \tag{5}$$

Evolve both the Gaussian and pulse initial profiles. How does the "naive" linear reconstruction perform?

Modify the flux calculation in such way that

$$f_{i\pm1/2} = \frac{1}{2}((q_L v_{i\pm1/2} + q_R v_{i\pm1/2}) - |v_{i\pm1/2}|(q_R - q_L)) \ , \tag{6}$$

i.e., allow $v = v(x)$ to be a function of x (you need to modify `calc_flux.f` and `spec_decomp.f`).

There are two equations which come to mind with this modification:

$$\frac{\partial q(t,x)}{\partial t} + v(x)\frac{\partial q(t,x)}{\partial x} = 0 \tag{7}$$

and

$$\frac{\partial q(t,x)}{\partial t} + \frac{\partial (v(x) q(t,x))}{\partial x} = 0 \ . \tag{8}$$

Which one is actually solved with this modification?

Derive an analytic solution of equation (7) for the case $v(x) = x$ for arbitrary initial data $q(t = 0, x) = q_0(x)$. (Hint: solve for the characteristics for this equation,i.e., lines in the (t, x) plane along which the function stays constant.)

Derive the analytic solution of the equation (8) using the ansatz

$$q_2(t,x) = A(t)B(x)q_1(t,x) \ , \tag{9}$$

where $q_1(t, x)$ is the solution of the equation (7) . In particular, write down the solution for $v(x) = x$.

Use numerical simulation to evolve the Gaussian profile (use $v(x) = x$). Compare the numerical solution with the analytic solutions for equations (7) and (8) . This should "experimentally" decide which equation is actually solved.

(Avoid using Maple unless you are totally lost).

4.3 The Burger's equation

Modify the code so that it can handle the Burger's equation. In what follows always use the piecewise linear reconstuction with the minmod slope limiter and the second order TVD Runge-Kutta time stepping.

Implement both the exact Riemann solver and the Roe approximation to the exact solver. At what type of Riemann problems do these two solvers yield different fluxes?

Evolve the Gaussian initial profile using both the exact and the Roe solver. Do you see any significant difference? Do you expect to see any? Observe how the initially smooth profile develops a shock front.

Monitor the volume conservation during the simulation. The l_1-norm of the function

$$||q(t,x)||_{l_1} = \int q(t,x)dx$$

should be conserved during the evolution. Check this using the `xvs` function $l1$-norm (under the *reduce* right click menu).

At later stages of the evolution the profile assumes the form of a time dependent propagating right triangle. Use this knowledge to derive the exact analytic (weak) solution for initially triangular profile (see figure 1). This would be the asymptotic solution to the evolution of the Gaussian profile.

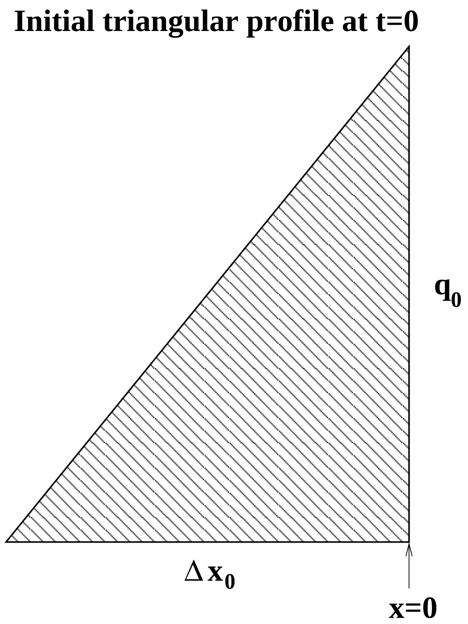

Figure 1: Profile for the right triangle initial data.

(Hint: you do not need to use the Burger's equation !!!). To check your solution evolve a triangular profile numerically and compare your solution with the exact analytic solution. User the parameters $\Delta x_0 = 0.5$, $q_0 = 1.0$.

Evolve a pulse initial profile using both the exact and Roe solver (`inittype:=1`). Compare the numerical results with the exact solution (adjust some parameters if necessary). Comment on what you see for each case. In particular, check if the Rankine-Hugoniot condition for the shock propagation speeds is satisfied:

$$v_s = \frac{f(q_R) - f(q_L)}{q_R - q_L} \ . \tag{10}$$

Use the pulse profiles defined by the following parameters.

Model A:

```
qleft:= -5.0
qcenter:= -1.0
qright:= -5.0
```

Model B:

```
qleft:= -5.0
qcenter:= 5.0
qright:= -5.0
```

Model C:

```
qleft:= 0.0
qcenter:= 5.0
qright:= 0.0
```

4.4 1D relativistic fluid in Minkowski spacetime

Modify the code so that it can evolve relativistic fluid in flat spacetime in Cartesian coordinates (using the “Roe” solver as described in class).

The equation of state for relativistic ideal fluid is

$$P = (\Gamma - 1)\rho_0 \epsilon \ , \tag{11}$$

where Γ is the adiabatic constant with a value $1 \leq \Gamma \leq 2$. The total energy-mass density ρ is related to the rest-mass density ρ_0 (the number density multiplied by the rest mass of the constituents for single component fluid) by the relation

$$\rho = \rho_0(1 + \epsilon) \ . \tag{12}$$

In order to evolve the fluid you have to maintain two sets of variables - the primitive variables ($w = (\rho_0, v, P)$) and the conservative variables ($q = (D, S, \tau)$). The relation between them is

$$D = \rho_0 W \ , \tag{13}$$

$$S = (\rho + P)W^2 v \ , \tag{14}$$

$$\tau = (\rho + P)W^2 - P - D \ , \tag{15}$$

$$W = \frac{1}{\sqrt{1 - v^2}} \ . \tag{16}$$

The conservative variables are those that are updated according the update (2) . The primitive variables are needed for the spectral decomposition (or the exact Riemann solver). The method of calculating the primitive variables from conservative is described in D.Neilsen's thesis on page 85. Note that there is a typo in the formula for $P(\omega)$ - the correct one reads

$$P(\omega) = \frac{\Gamma - 1}{\Gamma}(\omega - \rho_0) \ . \tag{17}$$

The spectral decomposition for our case can be found in Ian Hawke's thesis on page 94-95 (formulas (3.89)-(3.94)).

Since now we have two sets of variables it is not clear on which set we should actually perform the reconstruction[1]. Implement at least the reconstruction of primitive variables (to implement both is preferable).

Similar ambiguity arises when calculating the spectral decomposition. We could either use the average of the primitive variables ($w_A = (w_L + w_R)/2$) or we could average the conservative variables ($qA = (q_L + q_R)/2$) and then convert $q_A \to w_A$. Again, implement both possibilities.

In order to test the code evolve the Riemann problem for various left and right states. Table 1 contains the initial data for three qualitatively different solutions. Use $\Gamma = 1.5$ in your simulations. In case C use the average of the conservative variables ($qA = (q_L + q_R)/2$) and then convert $q_A \to w_A$ (you can still reconstruct the primitive variables). If you use $w_A = (w_L + w_R)/2$ you will get into trouble when converting primitive to conservative variables. This shows how small differences can have significant impact in practice.

Note that the solution type is controlled only by the velocity v_L (we choose $v_R = -v_L$). Estimate the "critical" values of v_L for which SS→RS and RS→RR. Notice that the variable ρ_0 also develops a special type of discontinuity called a *contact discontinuity*. A contact discontinuity is different from shock since the velocity and pressure at each side of the contact discontinuity are the same (unlike shocks).

5 References

All the resources mentioned in the text can be found at

[1]There is also a third set - the characteristic variables we obtain locally on each boundary. It would make most sense to reconstruct these but it is not in general the prefered choice. Actually, there is not an universal "best choice" (to my knowledge). It depends on the situation which variable set gives better results.

model	A	B	C
ρ_{0L}	10.0	10.0	10.0
ρ_{0R}	15.0	15.0	15.0
v_L	0.6	0.0	-0.6
v_R	-0.6	0.0	0.6
P_L	50.0	50.0	50.0
P_R	10.0	10.0	10.0

Table 1: Parameters for the Riemann problems A,B,C. Each represents a qualitatively different solution that can be characterized by the pressure profile. Model A contains two shocks (SS), model B contains one rarefaction and one shock (RS), model C consists of two rarefaction waves (RR).

`http://laplace.physics.ubc.ca/People/msnajdr/`