Méthodes numériques pour l'approximation des lois de conservation hyperboliques

TP2: Mélanges de fluides

Bonneel Nicolas et Durand Elise

6 avril 2006

1 Présentation du problème

L'objectif de ce TP est d'étudier le comportement d'un système bi-fluide dans un domaine rectangulaire soumis à un champ de gravité. On considère un verre contenant de l'eau et de l'air qui sont au repos à l'état initial et on incline le verre d'un angle α.

On utilise l'hypothèse d'un fluide incompressible et le système vérifie les équations suivantes :

$$\begin{cases} \frac{\partial \tilde{\rho_1}}{\partial t} + div(\tilde{\rho_1}\mathbf{u}) = 0 \\ \\ \frac{\partial \tilde{\rho_2}}{\partial t} + div(\tilde{\rho_2}\mathbf{u}) = 0 \\ \\ \frac{\partial \rho \mathbf{u}}{\partial t} + div(\rho \mathbf{u} \otimes \mathbf{u}) + \nabla(P) = \rho \mathbf{g} \end{cases}$$

avec

$$P = \alpha p_1(\tfrac{\tilde{\rho_1}}{\alpha}) + (1-\alpha) p_2(\tfrac{\tilde{\rho_2}}{1-\alpha})$$

et

$$p_i(\rho_i) = c_i^2(\rho_i - \rho_{i,0}) + p_{i,0}.$$

g est le champ de pesanteur donné par le vecteur suivant :

$$\mathbf{g} = \begin{pmatrix} g_x \\ g_y \end{pmatrix}$$

et le champ de vitesse est donné par :

$$\mathbf{u} = \begin{pmatrix} u \\ v \end{pmatrix}$$

Les conditions aux limites sur le bord sont les conditions de glissement :

$$\mathbf{u}.n = 0 \quad \Leftrightarrow \quad \begin{cases} u_{x_0} = u_{x_L} = 0 \\ v_{y_H} = v_{y_H} = 0 \end{cases}$$

2 Résolution numérique

Pour résoudre ce problème on définit les vecteurs suivants :

$$U = \begin{pmatrix} \tilde{\rho_1} \\ \tilde{\rho_2} \\ \rho u \\ \rho v \\ \rho \alpha \end{pmatrix}, \quad \mathcal{F}_1(U) = \begin{pmatrix} \tilde{\rho_1} u \\ \tilde{\rho_2} u \\ \rho u^2 + P \\ \rho uv \\ \rho \alpha u \end{pmatrix}, \quad \mathcal{F}_2(\mathbf{U}) = \begin{pmatrix} \tilde{\rho_1} v \\ \tilde{\rho_2} v \\ \rho uv \\ \rho v^2 + P \\ \rho \alpha u \end{pmatrix}$$

On utilise alors le schéma de Godunov relaxé suivant :

$$\begin{cases} \tilde{U}_{i,j}^{n+1} = U_{i,j}^n + \Delta t \mathcal{F}(U^n) \\ U_{i,j}^{n+1} = \mathcal{P}(\tilde{U}_{i,j}^{n+1}) \end{cases}$$

Cette méthode de calcul se déroule en 2 étapes successives :

- la partie hyperbolique : on calcule dans un premier temps la solution du problème sans certaines contraintes : $\tilde{U}_{i,j}^{n+1}$
- la partie relaxation : on trouve ensuite la solution du problème par projection sur l'espace des solutions admissibles : $U_{i,j}^{n+1}$

2.1 Partie hyperbolique

L'opérateur hyperbolique $\mathcal{F}$ se discrétise de la manière suivante :

$$\mathcal{F}(U^n) = \frac{G_{i+\frac{1}{2},j}^n - G_{i-\frac{1}{2},j}^n}{\Delta x} + \frac{H_{i,j+\frac{1}{2}}^n - H_{i,j-\frac{1}{2}}^n}{\Delta y} + S_{i,j}^n$$

où :
- $G_{i+\frac{1}{2},j}^n$ est le flux du schéma de Godunov associé à la direction x,
- $H_{i,j+\frac{1}{2}}^n$ est le flux du schéma de Godunov associé à la direction y.

avec :

$$G_{i+\frac{1}{2},j}^n = G(U_{i,j}, U_{i+1,j}), \quad H_{i,j+\frac{1}{2}}^n = G(U_{i,j}^*, U_{i,j+1}^*) \text{ et } S_{i,j}^n = \begin{pmatrix} 0 \\ 0 \\ \rho g_x \\ \rho g_y \\ 0 \end{pmatrix}$$

où

$$U^* = \begin{pmatrix} \tilde{\rho_1} \\ \tilde{\rho_2} \\ \rho v \\ \rho u \\ \rho \alpha \end{pmatrix}$$

Pour le flux sur la direction x, on utilise le schéma de Godunov pour l'équation :

$$U_t + \mathcal{F}_1(U)_x = 0$$

dont le flux numérique est :

$$G(U_g, U_d) = \mathcal{F}_1(W_1(0, U_g, U_d))$$

$W_1(0, U_g, U_d)$ est la solution du problème de Riemann suivant :

$$\left\{ \begin{array}{l} U_t + \mathcal{F}_1(U)_x = 0 \\ U(x,0) = \left\{ \begin{array}{l} U_g \ \text{ pour } \ x < 0, \\ U_d \ \text{ pour } \ x > 0 \end{array} \right. \end{array} \right.$$

De même, pour le flux sur la direction y, on utilise le schéma de Godunov pour l'équation :

$$U_t + \mathcal{F}_2(U)_x = 0$$

de flux numérique :

$$H(U_g, U_d) = \mathcal{F}_2(W_2(0, U_g, U_d))$$

où $W_2(\frac{x}{t}, U_g, U_d)$ est la solution du problème de Riemann :

$$\left\{ \begin{array}{l} U_t + \mathcal{F}_2(U)_x = 0 \\ U(x,0) = \left\{ \begin{array}{l} U_g \ \text{ pour } \ x < 0, \\ U_d \ \text{ pour } \ x > 0 \end{array} \right. \end{array} \right.$$

2.2 Partie relaxation

On trouve U^{n+1} par projection :

$$U_{i,j}^{n+1} = \mathcal{P}(\tilde{U}_{i,j}^{n+1})$$

avec :

$$\mathcal{P}\left(\begin{pmatrix} \tilde{\rho_1} \\ \tilde{\rho_2} \\ \rho u \\ \rho v \\ \rho \alpha \end{pmatrix} \right) = \begin{pmatrix} \tilde{\rho_1} \\ \tilde{\rho_2} \\ \rho u \\ \rho v \\ \rho \alpha^* \end{pmatrix}$$

où α^* est la fraction volumique d'équilibre caractérisée par :

$$p_1(\frac{\tilde{\rho_1}}{\alpha^*}) = p_2(\frac{\tilde{\rho_2}}{1-\alpha^*})$$

Dans notre cas, on a $p_{1,0} = p_{2,0} := p_0$, on retrouve explicitement :

$$\begin{cases} \alpha^* = \frac{\gamma}{\gamma+1} \\ \\ \gamma = \frac{q-\tilde{q}+\sqrt{(q-\tilde{q})^2+4\tilde{\rho}_1 c_1^2 \tilde{\rho}_2 c_2^2}}{2\tilde{\rho}_2 c_2^2} \\ \\ q = \rho_{2,0} c_2^2 - \rho_{1,0} c_1^2 \\ \\ \tilde{q} = \tilde{\rho}_2 c_2^2 - \tilde{\rho}_1 c_1^2 \end{cases}$$

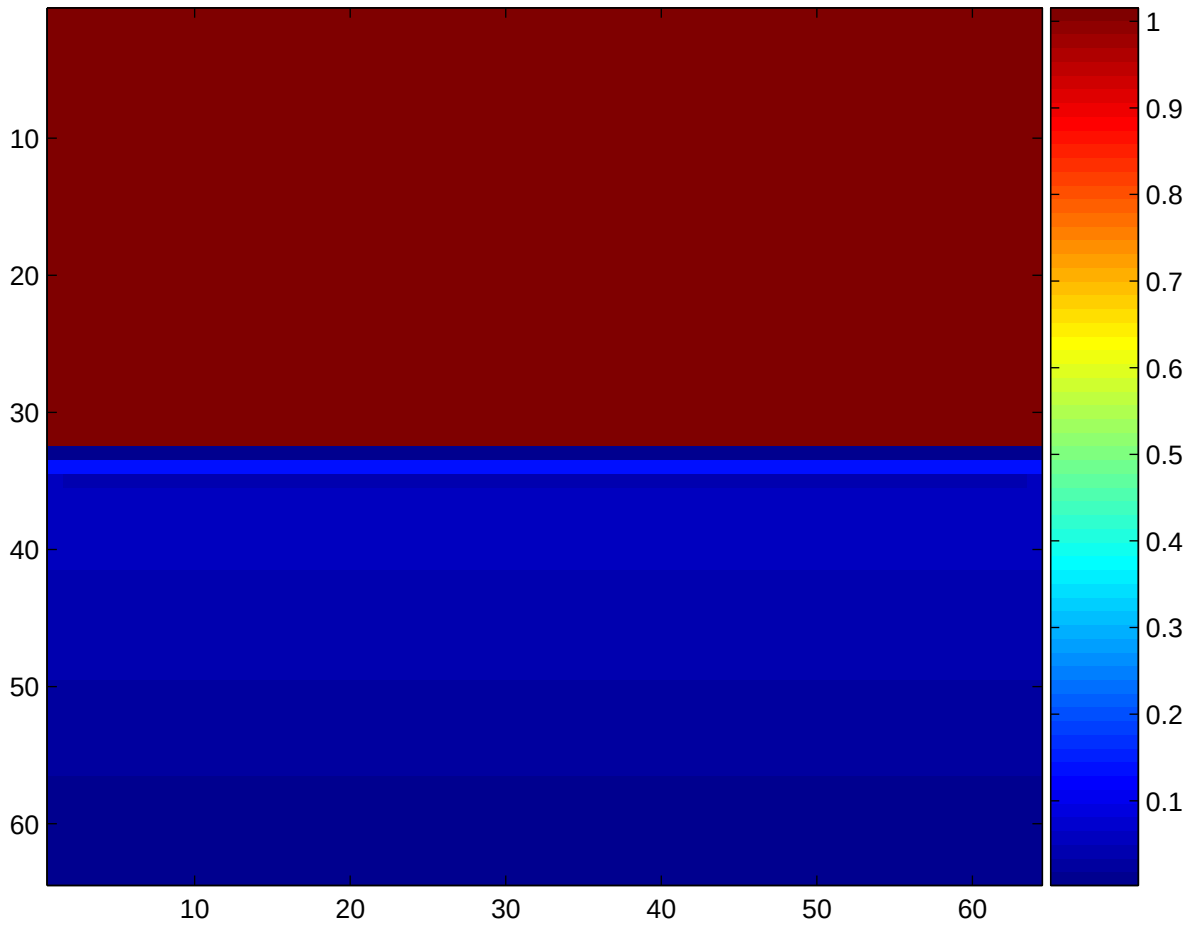

FIG. 1 – $\alpha = 1.5°$, maillage de 64x64

3 Mise en oeuvre et résultats

Bien qu'idéalement nous aurions dû recalculer un pas de temps optimal à chaque itération en fonction de la CFL, nous avons maintenu un pas de temps constant.
Nous obtenons alors des résultats qui ressemblent approximativement à un mélange de fluides mais qui ne correspondent pas en pratique à la réalité.

Programme

```
#include <math.h>
#include "preproc.h"
#include "struct.h"
#include <stdio.h>
```

```
#define dim1  64
#define dim2  64

const double c_eau     = 30.0;
const double c_air     = 20.0;
const double rhot1     = 1;
const double rhot2     = 1;
const double rho       = 2;       // rhot1 + rhot2
const double alpha_0 = (1E-6);
const double T         = 0.04;
const double deltaT    = 0.0003;
const double rho_air0  = 1.2;
const double rho_eau0  = 1000;
const double dx        = 0.2;
const double dy        = 0.2;
const double P01       = 1E4*0;
const double P02       = 1E4*0;  // = P01
const double H         = 128*0.2;   // dy*M
const double g         = 9.81;
const double theta     = M_PI*1.50/180.;

Phys  u[dim1][dim2];

coefp coefpr[3];



int main()
{
  int N = dim2;
  int M = dim1;


  FILE *sortie = fopen("sortie.txt", "w");
  FILE *params = fopen("params.txt", "w");

  fprintf(params, "%u \n", M);
  fprintf(params, "%u \n", N);



  // coeffs physiques
  coefpr[1].rhozero = rho_eau0;
  coefpr[1].c1      = c_eau;
```

```
  coefpr[1].c2      = P01;

  coefpr[2].rhozero = rho_air0;
  coefpr[2].c1      = c_air;
  coefpr[2].c2      = P02;



  // initialisation a t<0

  // ... pour la partie "air"
  int i, j;
  double P;
  for(i=0; i<M/2; i++)
    {
    for (j=0; j<N; j++)
      {
double z      = H-i*dy;
double rho2   = rho_air0*g*(H-z)/(c_air*c_air) + rho_air0;
double rho1   = (rho2-rho_air0)*c_air*c_air / (c_eau*c_eau) + rho_eau0;
u[i][j].rho   = rho1*alpha_0 + rho2*(1-alpha_0);
u[i][j].rhot1 = rho1*alpha_0;
u[i][j].rhot2 = rho2*(1-alpha_0);
u[i][j].rho2  = rho2;
u[i][j].rho1  = rho1;
u[i][j].vx    = 0;
u[i][j].vy    = 0;
u[i][j].rhovx = rhot1*0;
u[i][j].rhovy = rhot2*0;
u[i][j].alpha = alpha_0;
//     fprintf(sortie, "%f ",rho2);
      }
    // fprintf(sortie, "\n");
    }
  // ... pour la partie "eau"
  for(i=M/2; i<M; i++)
    {
    for (j=0; j<N; j++)
      {
double z      = H-i*dy;
        double rho1   = (rho_air0*g*H/2 + rho_eau0*g*(H/2-z))/(c_eau*c_e

double rho2   = (rho1-rho_eau0)*c_eau*c_eau / (c_air*c_air) + rho_air0;

u[i][j].rho1  = rho1;
u[i][j].rho2  = rho2;
```

```
u[i][j].rho   = rho1*(1-alpha_0) + rho2*alpha_0;
u[i][j].rhot1 = rho1*(1-alpha_0);
u[i][j].rhot2 = rho2*alpha_0;
u[i][j].vx    = 0;
u[i][j].vy    = 0;
u[i][j].rhovx = rhot1*0;
u[i][j].rhovy = rhot2*0;
u[i][j].alpha = 1-alpha_0;
//   fprintf(sortie, "%f ",rho2);

      }
    //  fprintf(sortie, "\n");
    }

  //exit(0);


  DPhys u_tmp[M][N];

  // resolution en temps
  double nt;
  int numT = 0;
  for (nt=0; nt<T; nt+=deltaT)
    {

      numT++;

      printf("T = %f \n", nt);
      fflush(stdout);
      fflush(stdout);


      // conditions aux limites
      for(i=0; i<M; i++)
{
  u[i][0] = u[i][1];
  u[i][N-1] = u[i][N-2];

  u[i][0].vx = -u[i][1].vx;
  u[i][N-1].vx = -u[i][N-2].vx;
  u[i][0].rhovx = -u[i][1].rhovx;
  u[i][N-1].rhovx = -u[i][N-2].rhovx;
}

      for(i=0; i<N; i++)
```

```
{
  u[0][i] = u[1][i];
  u[M-1][i] = u[M-2][i];

  u[0][i].vy = -u[1][i].vy;
  u[M-1][i].vy = -u[M-2][i].vy;
  u[0][i].rhovy = -u[1][i].rhovy;
  u[M-1][i].rhovy = -u[M-2][i].rhovy;
}

      // resolution interieure
      for(i=1; i<M-1; i++)
for (j=1; j<N-1; j++)
  {
    double Sx = u[i][j].rho*g*sin(theta);
    double Sy = -u[i][j].rho*g*cos(theta);

    DPhys G1, H1, G2, H2, F;
    Phys Ug, Ud, Ugg;

    Riemann(&G1, u[i][j], u[i][j+1]);
    Riemann(&G2, u[i][j-1], u[i][j]);

    Ugg = u[i-1][j];
    Ugg.rhovx = u[i-1][j].rhovy;
    Ugg.rhovy = u[i-1][j].rhovx;
    Ugg.vx = u[i-1][j].vy;
    Ugg.vy = u[i-1][j].vx;

    Ug = u[i][j];
    Ug.rhovx = u[i][j].rhovy;
    Ug.rhovy = u[i][j].rhovx;
    Ug.vx = u[i][j].vy;
    Ug.vy = u[i][j].vx;

    Ud = u[i+1][j];
    Ud.rhovx = u[i+1][j].rhovy;
    Ud.rhovy = u[i+1][j].rhovx;
    Ud.vx = u[i+1][j].vy;
    Ud.vy = u[i+1][j].vx;

    Riemann(&H1, Ug, Ud);
    Riemann(&H2, Ugg, Ug);
```

```
    F.rhoalpha = +(G1.rhoalpha - G2.rhoalpha)/dx + (H1.rhoalpha - H2.rho
      F.rhot1    = +(G1.rhot1 - G2.rhot1)/dx + (H1.rhot1 - H2.rhot1)/dy;
      F.rhot2    = +(G1.rhot2 - G2.rhot2)/dx + (H1.rhot2 - H2.rhot2)/dy;
    F.rhovx    = +(G1.rhovx - G2.rhovx)/dx + (H1.rhovy - H2.rhovy)/dy +
      F.rhovy    = +(G1.rhovy - G2.rhovy)/dx + (H1.rhovx - H2.rhovx)/dy

    /*      if ((F.rhot1 >0.0000000000000001)||(F.rhot2>0.0000000000000
      printf("%f %f %f %f %f \n",  F.rhoalpha, F.rhot1, F.rhot2, F.rhovx
      if (numT == 30) exit(0);
    */
    // mise a jour de U tild

    u_tmp[i][j].rhoalpha = (u[i][j].rhot1 + u[i][j].rhot2)*u[i][j].alpha
    u_tmp[i][j].rhot1    = u[i][j].rhot1 + deltaT*F.rhot1;
    u_tmp[i][j].rhot2    = u[i][j].rhot2 + deltaT*F.rhot2;
    u_tmp[i][j].rhovx    = u[i][j].rhovx + deltaT*F.rhovx;
    u_tmp[i][j].rhovy    = u[i][j].rhovy + deltaT*F.rhovy;

  }


      for(i=1; i<M-1; i++)
{
  for (j=1; j<N-1; j++)
    {
      // calcul de la relaxation
      /* double q     = rho_air0*(c_air*c_air) - rho_eau0*(c_eau*c_eau);
      double qt    = u_tmp[i][j].rhot2*(c_air*c_air) -  u_tmp[i][j].rhot
      double gamma = (q-qt+sqrt((q-qt)*(q-qt) + 4* u_tmp[i][j].rhot1*(c_

      double q     = rho_eau0*(c_eau*c_eau) - rho_air0*(c_air*c_air);
      double qt    = u_tmp[i][j].rhot1*(c_eau*c_eau) -  u_tmp[i][j].rhot
      double gamma = (q-qt+sqrt((q-qt)*(q-qt) + 4* u_tmp[i][j].rhot2*(c_

      double alpha_etoile;
      if (gamma>1E15)
alpha_etoile = 1.;
      else
alpha_etoile = gamma/(gamma+1);

      // et mise a jour de U
      u[i][j].rho   = u_tmp[i][j].rhot1+ u_tmp[i][j].rhot2 ;
      u[i][j].rhot1 = u_tmp[i][j].rhot1;
      u[i][j].rhot2 = u_tmp[i][j].rhot2;
      u[i][j].rhovx = u_tmp[i][j].rhovx;
```

```
      u[i][j].rhovy = u_tmp[i][j].rhovy;
      u[i][j].alpha = alpha_etoile;


      u[i][j].rho1  = u_tmp[i][j].rhot1 / alpha_etoile;
      u[i][j].rho2  = u_tmp[i][j].rhot2 / (1-alpha_etoile);

      u[i][j].vx = u_tmp[i][j].rhovx/ u[i][j].rho;
      u[i][j].vy = u_tmp[i][j].rhovy/ u[i][j].rho;

    }
  //if (nt==0)
  //  exit(0);
}

      //if ( numT % 45 == 0)
      for(i=0; i<M; i++)
{
  for (j=0; j<N; j++)
    fprintf(sortie, "%f ",u[i][j].alpha);

  fprintf(sortie, "\n");
}









    }



  fprintf(params, "%u \n", numT);


}
```