Final Project

MAE598: Advanced CFD with Interfaces

11/30/17

Adil Ansari

Contents

Methodology

In this project a 2 phase liquid was analyzed. The methods used here for interface affection was volume of fluid. The volume of fluid method guarantees volume conservation unlike the ode method which doesn't conserve volume.

The following approach was chosen:

For every cell that has non-zero and non-one psi value within the 9 neigbour stencil, following is done

1. 12 Flux triangles are created based on corner velocities and face velocities.
    1. Triangles 0 - 7 are just based on corner velocities
    2. Triangles 8-11 are additional volume to satisfy continuity
2. All cells with non-zero psi values are converted into another polygon as per their PLIC normal*. psi = 1 remains a square.
3. Cell polygons are shifted appropriately as per their [i+m][j+n] designation
4. All triangles and cells intersected** and the absolute value of intersection volume is multiplied by the sign of volume of flux triangles and summed***
5. This sum is change in psi. For psi=1 cells that are surrounded by psi=1 cells, they remain unchanged assuming 0 divergence.
6. Another approach requires to change volume of cells where liquid is being borrowed from. This ensures volume conservation while in the previous case volume conservation is entirely based on level of zero divergence of velocity.

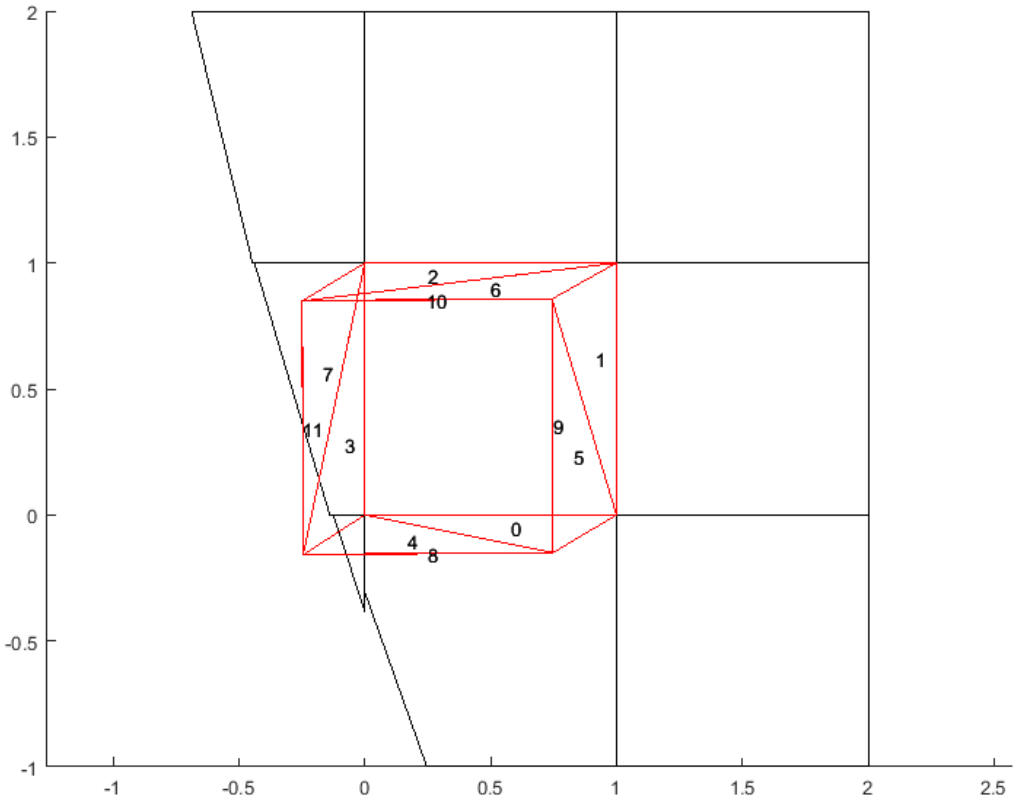
*This was done by snipping a square by the PLIC normal. The accepted points of polygon were organized clockwise by using insertion sort of angles relative to center of gravity of polygon.

**All polygons are concave.

***The library used for polygon intersection can be found at: https://rosettacode.org/wiki/Sutherland-Hodgman_polygon_clipping#C . Majority of intersections produce zero area and hence many protocols and if-statements were used to eliminate useless intersections as much as possible.

In the first case, the VOF computation was computer only on or one cell near the interface. The advantage of this method is fast computation. However this method doesn’t work well when interfaces collide or atomize. Smaller regions tend to numerically dissolve/evaporate and as a result we have some loss of volume depending upon level of atomization.

The other approach is looping over every liquid volume cell which is slower and only works well with divergence free velocity. The advantage of this is total volume conversation even in severe atomization.



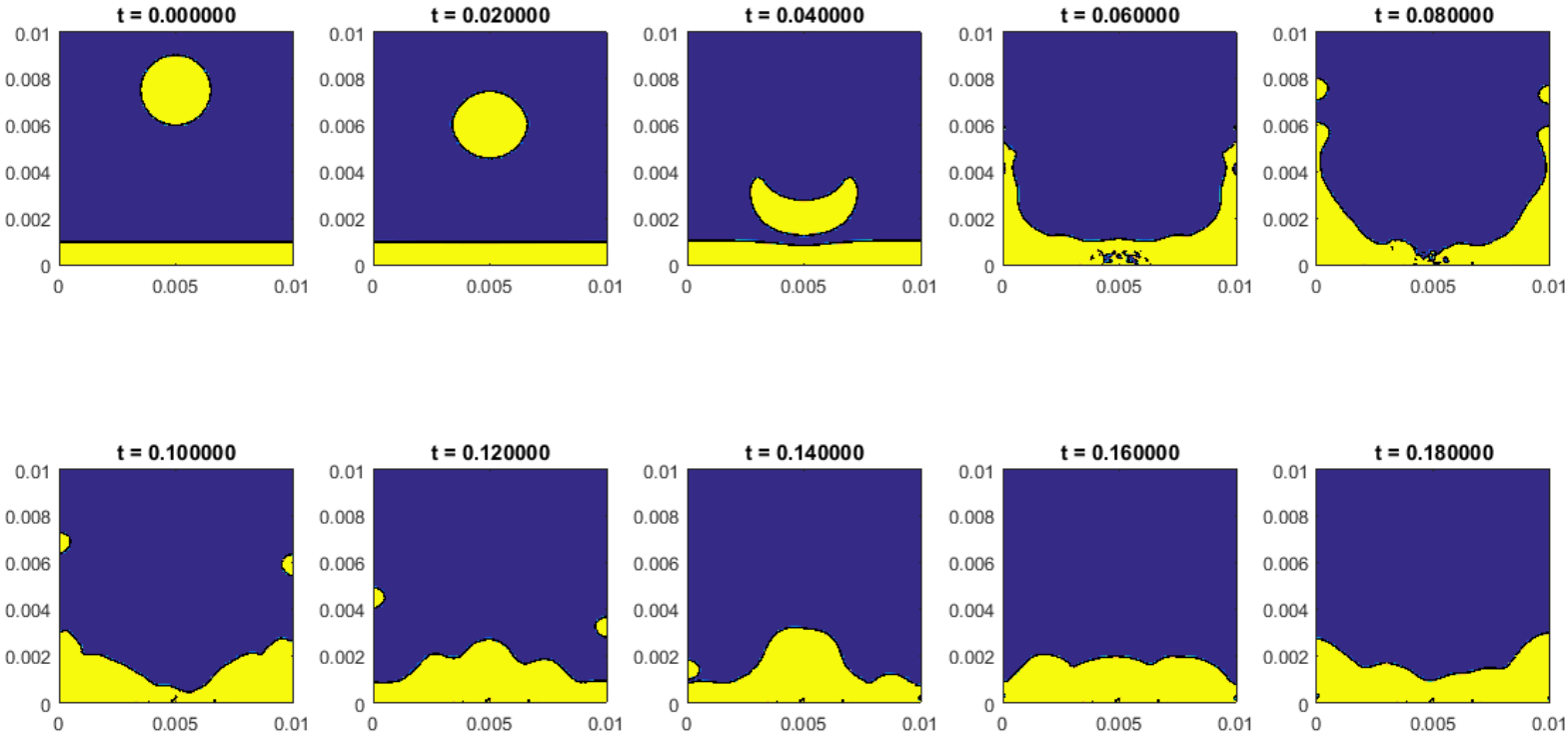
The pressure poison system was solves using GMRES and ILU preconditioners. A multigrid was coded. However, it didn't achieve any further convergence after 1e-6 and a convergence of 1e-10 is required to run VOF and so it wasn't used. GMRES with multigrid preconditioner would theoretically work much faster.

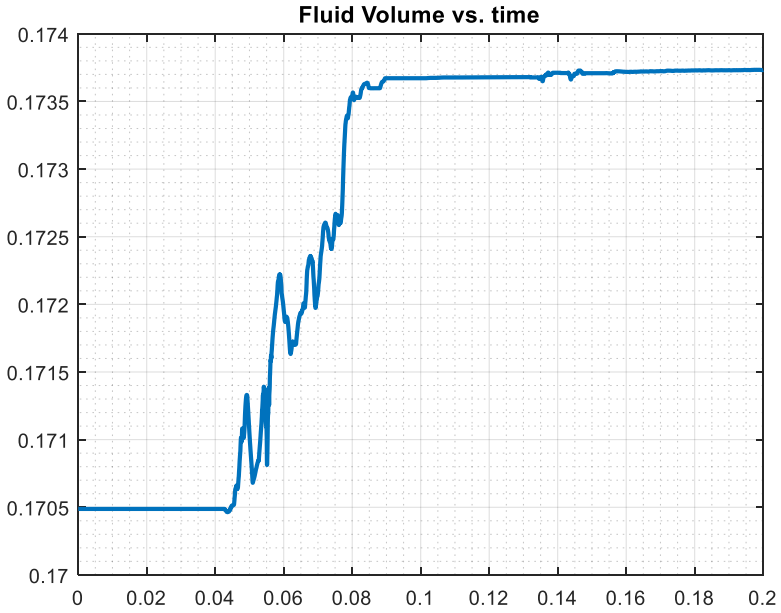
Programming Language:

The language used here is primarily MATLAB. The VOF method was coded in C and hence it was compiled as mexw64 object which can be run as MATLAB subroutine on machine level.

Results

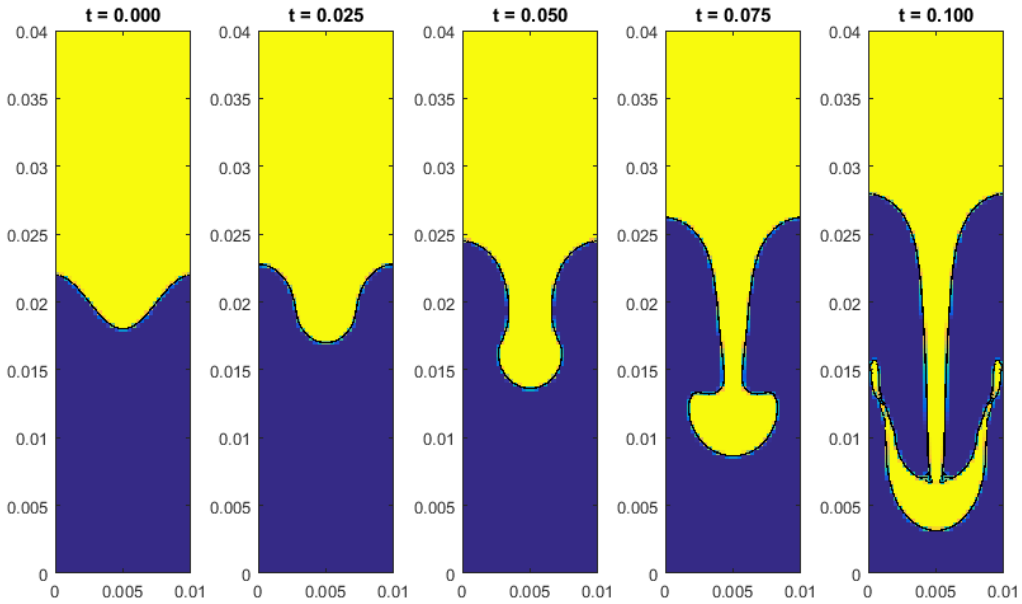
Task 2 Results: Drop Fall (100x100)

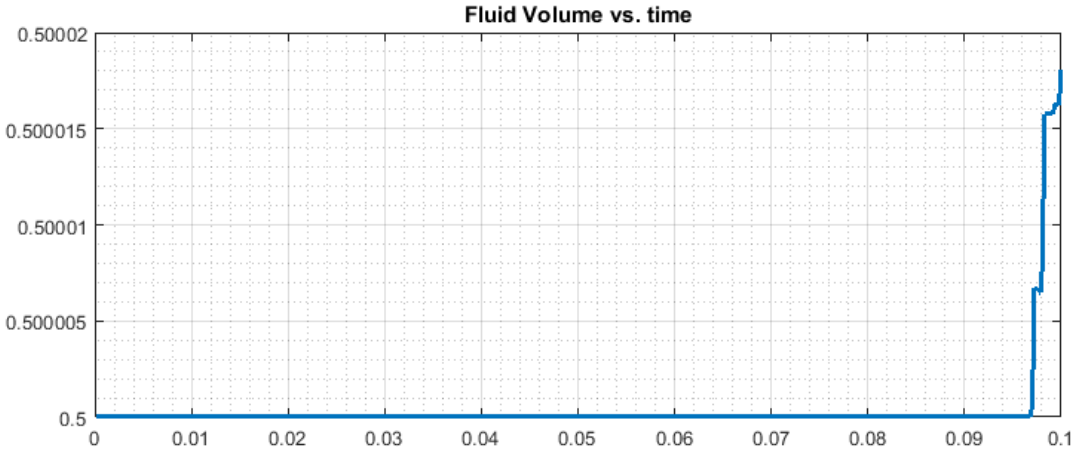




The Volume of liquid is constant until collision. Most of the liquid volume is generated by gas being surrounded liquid erroneously converted to liquid. This leads to a 1.7% Error gain in liquid volume.

Task 3: Raleigh-Taylor: (32x128)





The Volume of liquid is constant until mushroom cloud pinches out adjacent drop. A net gain of 0.004% in volume is seen after drop separation.

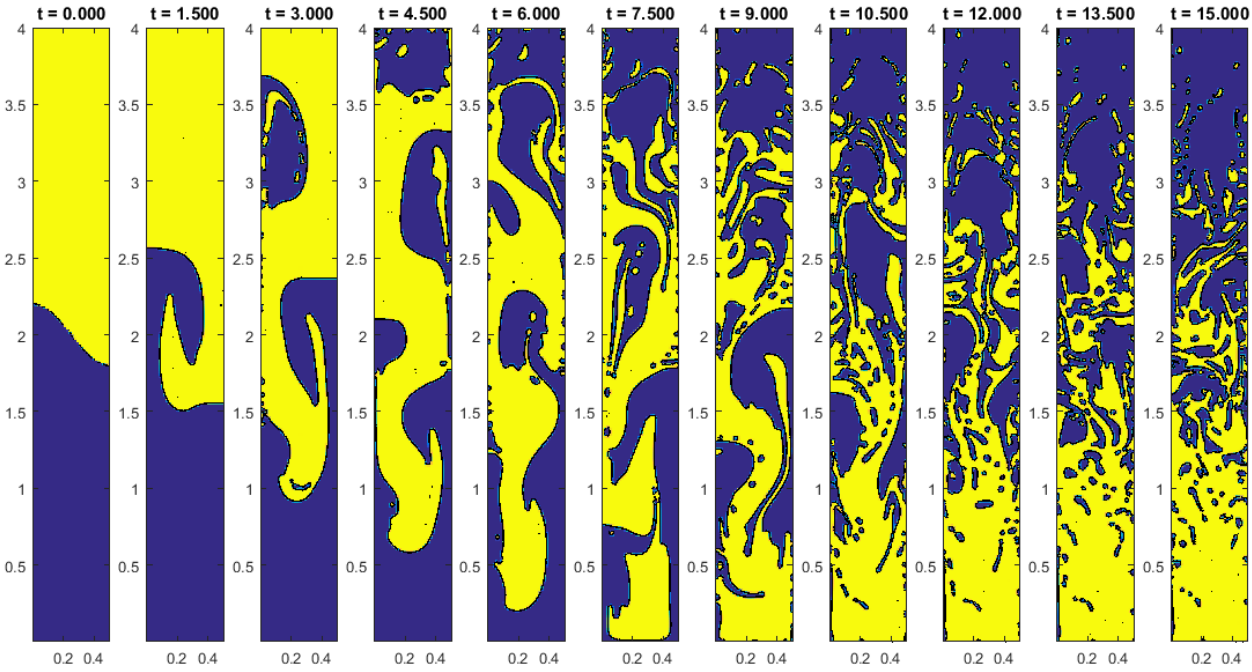
Convergence Analysis:
The following is the maximum v-velocity after 1e-4 seconds as a function of resolution:

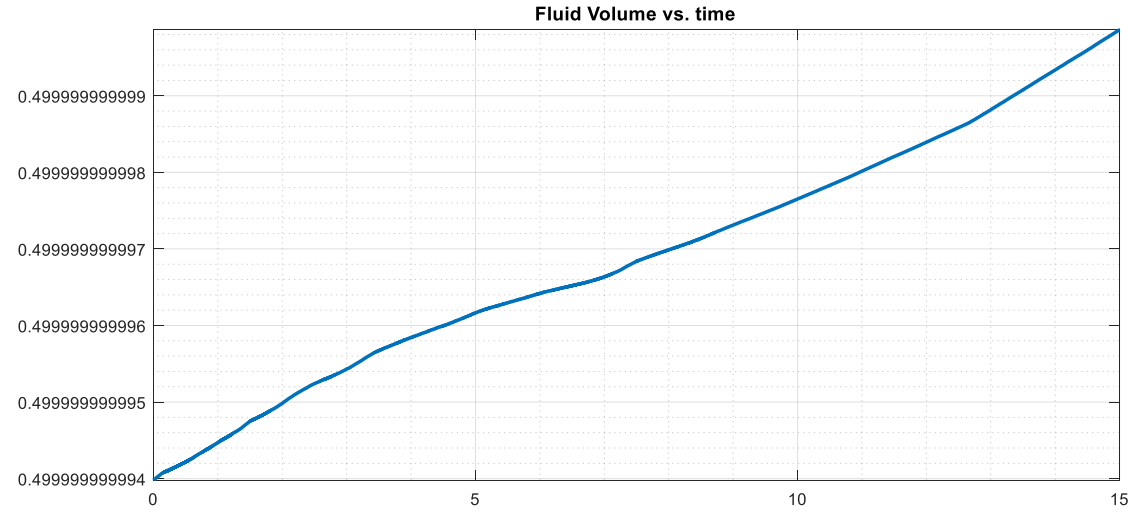
Resolution	Volume	Order of Convergence	Richardson Exploration	Error	Error %
64x256	9.59E-04				
128x512	1.34E-03	**1.32939**	0.00159	0.10128	**8.368%**
256x1024	1.49E-03				

Order of convergence of **1.33** achieved with extrapolated value of **0.00159±8.4%**

2D Atomization Test: (32x256)

The following test case employs looping over every liquid volume cell. The advection was forced by a simple Boussinesq approximation (buoyancy) approximation. Single coefficient pressure solver using fft was used to speed up the process and give divergence free velocity.





Since the every borrowed volume is subtracted and added from cell to cell, this method doesn't cause any noticeable gain in fluid volume.

Code

Cellvof.c (Unsplit-VOF Method)

```
#include <mex.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdint.h>
#include <stdlib.h>
#define max(x, y) (((x) > (y)) ? (x) : (y))
#define min(x, y) (((x) < (y)) ? (x) : (y))

double sign(double x) {return ((x > 0) ? 1 : ((x < 0) ? -1 : 0));}
int  isign(double x) {return ((x > 0) ? 1 : ((x < 0) ? -1 : 0));}
int heavy(double x) {return (x > 0);}

double randn(){
    return 1e-50;
}

void insertion_sort(double a[], int n, double b[2][7]) {
    int i,j,k;
    for(i = 1; i < n; ++i) {
        double tmp = a[i];
        double tmpx = b[0][i];
        double tmpy = b[1][i];
        int j = i;
        while(j > 0 && tmp < a[j - 1]) {
            a[j] = a[j - 1];
            b[0][j] = b[0][j-1];
            b[1][j] = b[1][j-1];
            --j;
        }
        a[j] = tmp;
        b[0][j] = tmpx;
        b[1][j] = tmpy;
    }
}


struct polygon
{
    int count;
    double x[7], y[7];
    double area;
    double bBox[4];
};

struct polygon foo(double A[2][7]){//generate poylgon
    int i,j,k;
    struct polygon B = {0};
    for (i = 0; i < 7; i++){
        B.x[i] = A[0][i];
```

```
        B.y[i] = A[1][i];
        if ((B.x[i]==0) && (B.y[i] == 0)){
            //Polygon 0 pts need to be non-zero ~1e-14)
            B.count = i;
            break;
        }
    }
    //compute area
    for (i = 0; i < B.count-1; i++){
        B.area += B.x[i]*B.y[i+1]-B.x[i+1]*B.y[i];
    }
    B.area*=0.5; //half of determinant
    return B;
}

struct polygon foo2(double mx, double my, double a){
    int i, k;
    double dat[] = {0, a/my, 1, (a - mx)/my, a/mx, 0, (a - my)/mx, 1};
    int ignore[8] = {0};
    for(i = 0; i < 8; i++){
        if ((dat[i]>1)||(dat[i]<0)){
            ignore[i] = 1;
            if (i < 4) ignore[i-1] = 1;
            else  ignore[i+1] = 1;
        }
    }
    i = 0; double cell[4] = {0};
    for(k = 0; k < 8; k++){
        if (!ignore[k]){
            cell[i] = dat[k]; i++;
        }
    }
    double XC[2][6] = {{0,0,1e-50,1,0,1},{0,0,0,0,1,1}};
    XC[0][0] = cell[0];
    XC[1][0] = cell[1];
    XC[0][1] = cell[2];
    XC[1][1] = cell[3];

    for (i = 2; i < 6; i++){
        if (!heavy(mx*XC[0][i]+my*XC[1][i]-a)){
            XC[0][i] = -1;
            XC[1][i] = -1;
        }
    }
    double XD[2][7] = {0};
    k = 0;
    for (i = 0; i < 6; i++){
        if (XC[0][i] > -.5){
            XD[0][k] = XC[0][i]+randn();
            XD[1][k] = XC[1][i]+randn();
            k++;
        }
    }
    double meanx = 0, meany = 0;
    for (i = 0; i < k; i++){
        meanx = meanx + XD[0][i];
```

```
        meany = meany + XD[1][i];
    }
    meanx = meanx/k;
    meany = meany/k;
    double theta[7] = {0};
    for(i = 0; i < 7; i++){
        theta[i] = atan2(XD[1][i]-meanx,XD[0][i]-meany);
    }
    insertion_sort(theta, k, XD);
    XD[0][k]=XD[0][0]; XD[1][k]=XD[1][0];

    struct polygon B = foo(XD);
    return B;
}

typedef struct { double x, y; } vec_t, *vec;

double dot(vec a, vec b)
{
    return a->x * b->x + a->y * b->y;
}

double cross(vec a, vec b)
{
    return a->x * b->y - a->y * b->x;
}

vec vsub(vec a, vec b, vec res)
{
    res->x = a->x - b->x;
    res->y = a->y - b->y;
    return res;
}

/* tells if vec c lies on the left side of directed edge a->b
 * 1 if left, -1 if right, 0 if colinear
 */
int left_of(vec a, vec b, vec c)
{
    vec_t tmp1, tmp2;
    double x;
    vsub(b, a, &tmp1);
    vsub(c, b, &tmp2);
    x = cross(&tmp1, &tmp2);
    return x < 0 ? -1 : x > 0;
}

int line_sect(vec x0, vec x1, vec y0, vec y1, vec res)
{
    vec_t dx, dy, d;
    vsub(x1, x0, &dx);
    vsub(y1, y0, &dy);
    vsub(x0, y0, &d);
    /* x0 + a dx = y0 + b dy ->
     * x0 X dx = y0 X dx + b dy X dx ->
```

```
     * b = (x0 - y0) X dx / (dy X dx) */
    double dyx = cross(&dy, &dx);
    if (!dyx) return 0;
    dyx = cross(&d, &dx) / dyx;
    if (dyx <= 0 || dyx >= 1) return 0;

    res->x = y0->x + dyx * dy.x;
    res->y = y0->y + dyx * dy.y;
    return 1;
}

/* === polygon stuff === */
typedef struct { int len, alloc; vec v; } poly_t, *poly;

poly poly_new()
{
    return (poly)calloc(1, sizeof(poly_t));
}

void poly_free(poly p)
{
    free(p->v);
    free(p);
}

void poly_append(poly p, vec v)
{
    if (p->len >= p->alloc) {
        p->alloc *= 2;
        if (!p->alloc) p->alloc = 4;
        p->v = (vec)realloc(p->v, sizeof(vec_t) * p->alloc);
    }
    p->v[p->len++] = *v;
}

/* this works only if all of the following are true:
 *   1. poly has no colinear edges;
 *   2. poly has no duplicate vertices;
 *   3. poly has at least three vertices;
 *   4. poly is convex (implying 3).
 */
int poly_winding(poly p)
{
    return left_of(p->v, p->v + 1, p->v + 2);
}

void poly_edge_clip(poly sub, vec x0, vec x1, int left, poly res)
{
    int i, side0, side1;
    vec_t tmp;
    vec v0 = sub->v + sub->len - 1, v1;
    res->len = 0;

    side0 = left_of(x0, x1, v0);
    if (side0 != -left) poly_append(res, v0);
```

```
    for (i = 0; i < sub->len; i++) {
        v1 = sub->v + i;
        side1 = left_of(x0, x1, v1);
        if (side0 + side1 == 0 && side0)
            /* last point and current straddle the edge */
            if (line_sect(x0, x1, v0, v1, &tmp))
                poly_append(res, &tmp);
        if (i == sub->len - 1) break;
        if (side1 != -left) poly_append(res, v1);
        v0 = v1;
        side0 = side1;
    }
}

poly poly_clip(poly sub, poly clip)
{
    int i;
    poly p1 = poly_new(), p2 = poly_new(), tmp;

    int dir = poly_winding(clip);
    poly_edge_clip(sub, clip->v + clip->len - 1, clip->v, dir, p2);
    for (i = 0; i < clip->len - 1; i++) {
        tmp = p2; p2 = p1; p1 = tmp;
        if(p1->len == 0) {
            p2->len = 0;
            break;
        }
        poly_edge_clip(p1, clip->v + i, clip->v + i + 1, dir, p2);
    }

    poly_free(p1);
    return p2;
}

double intersect(struct polygon C, struct polygon S, int test){
    vec_t c[] =
{{C.x[0],C.y[0]},{C.x[1],C.y[1]},{C.x[2],C.y[2]},{C.x[3],C.y[3]},{C.x[4],C.y[
4]},{C.x[5],C.y[5]},{C.x[6],C.y[6]}};
    vec_t s[] =
{{S.x[0],S.y[0]},{S.x[1],S.y[1]},{S.x[2],S.y[2]},{S.x[3],S.y[3]},{S.x[4],S.y[
4]},{S.x[5],S.y[5]},{S.x[6],S.y[6]}};
    poly_t clipper = {C.count, 0, c};
    poly_t subject = {S.count, 0, s};

    poly res = poly_clip(&subject, &clipper);
    int i;
    double areaI = 0,al = 0, ar = 0;

    /*if (0){
        mexPrintf("inter = [");
        for (i = 0; i < res->len+test; i++){
            mexPrintf("%g %g\n", res->v[i].x, res->v[i].y); }
    }
    mexPrintf("res->len = %i\n",res->len);*/
```

```
    if (res->len){
        for (i = 0; i < res->len-1+test; i++){
            al += (res->v[i].x)*(res->v[i+1].y);
            ar += (res->v[i].y)*(res->v[i+1].x);
            //if (test){ mexPrintf("%g %g\n", al,ar); }
        }
        al += (res->v[i].x)*(res->v[0].y);  ar += (res->v[i].y)*(res-
>v[0].x);
        //if (0){ mexPrintf("%g %g];\n\n", res->v[0].x, res->v[0].y,(res-
>v[i].x)*(res->v[0].y)); }
    }
    //mexPrintf("Area: %g;\n", al-ar);
    areaI = (al-ar)*.5;
    poly_free(res);
    return fabs(areaI);

}

void printPoly(struct polygon B){
    int i,j,k;
    //mexPrintf("\n")
    for(i = 0; i < B.count; i++) {
        mexPrintf("%g\t%g\n",B.x[i],B.y[i]);
    }
    //mexPrintf("Total:%i\n",B.count);
}

struct polygon shiftPoly(struct polygon B, double shiftx, double shifty){
    int i,j,k;
    for( i = 0; i < B.count; i++) {
        B.x[i]+=shiftx;
        B.y[i]+=shifty;
    }
    B.bBox[0] = 1e5; B.bBox[1] = -1e5; B.bBox[2] = 1e5; B.bBox[3] = -1e5;
    for (i = 0; i < B.count; i++){
        B.bBox[0] = min(B.bBox[0], B.x[i]);
        B.bBox[1] = max(B.bBox[1], B.x[i]);
        B.bBox[2] = min(B.bBox[2], B.y[i]);
        B.bBox[3] = max(B.bBox[3], B.y[i]);
    }
    return B;
}

int DoBoxesIntersect(struct polygon r1, struct polygon r2) {
     return !(r2.bBox[0] > r1.bBox[1]
        || r2.bBox[1] < r1.bBox[0]
        || r2.bBox[3] > r1.bBox[2]
        || r2.bBox[2] < r1.bBox[3]);
}

void fluxHorizontal(int M, int N, double u[M+1][N+2], double v[M+2][N+1],
double *psi ,double *nx,
        double *ny, double *alpha, double dx, double dt, double *psi2){
    int i,j,k,m,n; double Lx, Ly, Lm, L;
    for(i = 0; i < (M+4)*(N+4); i++)
        psi2[i] = psi[i];
```

```
    const int TRI[12][4] =
{{0,5,1,0},{1,6,2,1},{2,7,3,2},{3,4,0,3},{0,4,5,0},{1,5,6,1},{2,6,7,2},{3,7,4
,3},{4,8,5,4},{5,9,6,5},{6,10,7,6},{7,11,4,7}};
    const int comp[12][6] = {{4,5,1,2,-1,-1},{4,7,5,8,-1,-1},{4,3,7,6,-1,-
1},{4,1,3,0,-1,-
1},{3,4,5,0,1,2},{1,4,7,2,5,8},{5,4,3,8,7,6},{7,4,1,6,3,0},{3,4,5,0,1,2},{1,4
,7,2,5,8},{5,4,3,8,7,6},{7,4,1,6,3,0}};
    for(i = 2; i < M+2; i++){//for(i = M + 1; i > 1; i--){
        for(j = 2; j < N + 2; j++){
            int ind[9] = {i-1+(M+4)*(j-1),i+(M+4)*(j-1),i+1+(M+4)*(j-1),
            i-1+(M+4)*(j),i+(M+4)*(j),i+1+(M+4)*(j),
            i-1+(M+4)*(j+1),i+(M+4)*(j+1),i+1+(M+4)*(j+1)};

            int check0 = 1, check1 = 0, check2 = 0;
            for (k = 0; k < 9; k++){
                check0 = check0 && ((psi[ind[k]] < 0.5) && (alpha[ind[k]] >
1e19));
                check1 += ((psi[ind[k]] > 0.5) && (alpha[ind[k]] > 1e19));
                check2 += (alpha[ind[k]] > 1e19);
            }

            if ((check2!=0)&&(check2!=9)){ //add condition for skipping here!

                double F[2][12] = {{1e-50,1,1,0,-(u[i-2][j-1]+u[i-2][j-
2])*dt/(2.0*dx),1-(u[i-1][j-1]+u[i-1][j-2])*dt/(2.0*dx),1-(u[i-1][j]+u[i-
1][j-1])*dt/(2.0*dx),-(u[i-2][j]+u[i-2][j-1])*dt/(2.0*dx),0,0,0,0},
                {1e-50,0,1,1,-(v[i-1][j-2]+v[i-2][j-2])*dt/(2.0*dx),-(v[i][j-
2]+v[i-1][j-2])*dt/(2.0*dx),1-(v[i][j-1]+v[i-1][j-1])*dt/(2.0*dx),1-(v[i-
1][j-1]+v[i-2][j-1])*dt/(2.0*dx),0,0,0,0}};
                double TRIArea[12] = {0};
                for (n = 0; n < 8; n++){
                    for (k = 0; k < 3; k++)
                        TRIArea[n] += 0.5*(F[0][TRI[n][k]]*F[1][TRI[n][k+1]]-
F[0][TRI[n][k+1]]*F[1][TRI[n][k]]);}
                double faceVel[4] = {v[i-1][j-2],-u[i-1][j-1]/*u2*/,-v[i-
1][j-1],u[i-2][j-1]};
                double err2 = (v[i-1][j-1]-v[i-1][j-2]+u[i-1][j-1]-u[i-2][j-
1])/4.0;
                for  (n = 0; n < 4; n++) faceVel[n]-=err2;
                double TRIArea2[12] = {0};
                for  (n = 0; n < 4; n++){
                    TRIArea[n+8] = faceVel[n]*dt/dx - (TRIArea[n] +
TRIArea[n+4]);
                    int n5 = 0;
                    if (n == 3) n5 = 4; else n5 = n + 5;
                    Lx =  -(F[1][n+4]-F[1][n5])*sign(TRIArea[n+8]);
                    Ly = (F[0][n+4]-F[0][n5])*sign(TRIArea[n+8]);
                    Lm = sqrt(Lx*Lx+Ly*Ly);
                    L = 2*TRIArea[n+8]/Lm; if isnan(L) L = 0;
                    F[0][n+8] = (F[0][n+4] + F[0][n5])/2.0 + Lx/Lm*L;
                    F[1][n+8] = (F[1][n+4] + F[1][n5])/2.0 + Ly/Lm*L;
                    for (k = 0; k < 3; k++)
                        TRIArea2[n+8] +=
0.5*(F[0][TRI[n+8][k]]*F[1][TRI[n+8][k+1]]-
F[0][TRI[n+8][k+1]]*F[1][TRI[n+8][k]]);
                    if (sign(TRIArea2[n+8]*TRIArea[n+8])==-1.0) {
```

```
                        F[0][n+8] -= 2*Lx/Lm*L;
                        F[1][n+8] -= 2*Ly/Lm*L;
                    }
                    TRIArea2[n+8] = 0;
                    for (k = 0; k < 3; k++)
                        TRIArea2[n+8] +=
0.5*(F[0][TRI[n+8][k]]*F[1][TRI[n+8][k+1]]-
F[0][TRI[n+8][k+1]]*F[1][TRI[n+8][k]]);
                }

                for (n = 0; n < 8; n++){
                    for (k = 0; k < 3; k++)
                        TRIArea2[n] +=
0.5*(F[0][TRI[n][k]]*F[1][TRI[n][k+1]]-F[0][TRI[n][k+1]]*F[1][TRI[n][k]]);}



                ////DEFINE SQUARE CELL SHAPE
                double shiftx[9] = {-1.5,-0.5,0.5,-1.5,-0.5,0.5,-1.5,-
0.5,0.5}, shifty[9] = {-1.5,-1.5,-1.5,-0.5,-0.5,-0.5,.5,.5,.5};
                struct polygon Cell[9] = {0}; //all concerned cells
                for (k = 0; k < 9; k++){
                    if ((psi[ind[k]] > 0.5) && (alpha[ind[k]] > 1e19)){
                        double square[2][7] =
                        {{1e-100,1,1,0,0,0,0},{0,0,1,1,0,0,0}};
                        square[0][4] = square[0][0];
                        square[1][4] = square[1][0];
                        Cell[k] = foo(square);
                    }
                    else if ((psi[ind[k]] < 0.5) && (alpha[ind[k]] > 1e19)){
                        double sm = 1e-10;
                        double square2[2][7] =

{{.5,.5+sm,.5+sm,.5,0,0,0},{.5,.5,.5+sm,.5+sm,0,0,0}};
                        //Flux triangles never come close to cell center due
to prescribed BC
                        square2[0][4] = square2[0][0];
                        square2[1][4] = square2[1][0];
                        Cell[k] = foo(square2);
                    }
                    else{
                        Cell[k] = foo2(nx[ind[k]],ny[ind[k]],alpha[ind[k]]);
                    }
                    Cell[k] = shiftPoly(Cell[k],shiftx[k],shifty[k]);
                }
                int comp2[12][6];
                for(m = 0; m < 12; m++){
                    for(n = 0; n < 6; n++)
                        comp2[m][n] = comp[m][n];
                }
                for(m = 0; m < 4; m++){
                    if (TRIArea[m]>0){
                        comp2[m][0] = -1;
                        comp2[m][1] = -1;
                    }
                    else{
```

```
                    comp2[m][2] = -1;
                    comp2[m][3] = -1;
                }
            }
            /*for(m = 4; m < 8; m++){
                if ((TRIArea[m]>0) && (TRIArea[m-4]>0)){
                    comp2[m][0] = -1;
                    comp2[m][1] = -1;
                    comp2[m][2] = -1;
                    comp2[m+4][0] = -1;
                    comp2[m+4][1] = -1;
                    comp2[m+4][2] = -1;
                }

            else if ((TRIArea[m]<0) && (TRIArea[m-4]<0)){
                    comp2[m][3] = -1;
                    comp2[m][4] = -1;
                    comp2[m][5] = -1;
                    comp2[m+4][3] = -1;
                    comp2[m+4][4] = -1;
                    comp2[m+4][5] = -1;
                }
            }*/

            //DEFINE FLUX TRIANGLES
            struct polygon tria[12] = {0}; //all concerned flux triangles
            for (k = 0; k < 12; k++){
                double F2[2][7] = {0};
                for(m = 0; m < 4; m++){
                    F2[0][m] = F[0][TRI[k][m]];
                    F2[1][m] = F[1][TRI[k][m]];
                }
                tria[k] = foo(F2);
                tria[k] = shiftPoly(tria[k],-0.5,-0.5);
            }

            double iA = 0;
            //Intersect Cells
            for (m = 0; m < 12; m++){
                for (n = 0; n < 6; n++){
                    k = comp2[m][n];
                    if (k >= 0){
                        if (psi[ind[k]]>1e-14){
                                iA =
max(fabs(intersect(tria[m],Cell[k],0)),   0.0

/*fabs(intersect(Cell[k],tria[m],0))*/)*sign(tria[m].area);
                                psi2[ind[4]]+=iA;
                        }
                    }
                }
            }
            /*if (0*(check1==7)){
                double cellflux = 0, cellflux2 = 0, cellflux3;
                //for (m = 8; m < 12; m++)
mexPrintf("%i\t%g\t%g\n",m,TRIArea[m],TRIArea2[m]);
```

```
                    for (m = 0; m < 12; m++){
                        cellflux2+=TRIArea2[m];cellflux3 = 0;
                        for (k = 0; k < 9; k++){
                            iA = max(fabs(intersect(tria[m],Cell[k],0)),
fabs(intersect(Cell[k],tria[m],0)))*sign(tria[m].area);
                            cellflux+=iA;
                            cellflux3+=iA;
                            if (fabs(iA)>0)
mexPrintf("%i\t%i\t%g\t\t\t\t\n",m,k,iA);
                        }mexPrintf("Expected Total: %g\t Acheived total:
%g\n\n",TRIArea[m],cellflux3);
                    }mexPrintf("TotalCell Flux:\t%g\tsum of
triangles:\t%g\n",cellflux,cellflux2);
                    for(n = 0; n < 9; n++)
{mexPrintf("\nCPoly%i=[",n);printPoly(Cell[n]);mexPrintf("];\n");}
                    for(n = 0; n < 12; n++)
{mexPrintf("\nTPoly%i=[",n);printPoly(tria[n]);mexPrintf("];\n");}
                    goto breakout;
                }*/
            }
        }
    }//breakout: 0;//mexPrintf("Escaped\n");
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    srand(554);
    /* Macros for the ouput and input arguments */
    double *B, *U, *V, *mxin,  *myin, *a1in, *psin, dx, dy, dt;
    int M, N, i, j, k, num;
    M = mxGetScalar(prhs[0]); /* Get the dimensions of A */
    N = mxGetScalar(prhs[1]);
    U    = mxGetPr(prhs[2]);
    V    = mxGetPr(prhs[3]);
    mxin = mxGetPr(prhs[4]);
    myin = mxGetPr(prhs[5]);
    a1in = mxGetPr(prhs[6]);
    psin = mxGetPr(prhs[7]);
    dx = mxGetScalar(prhs[8]); dy = dx;
    dt = mxGetScalar(prhs[9]);

    plhs[0] = mxCreateDoubleMatrix(M+4, N+4, mxREAL); /* Create the output
matrix */

    B = mxGetPr(plhs[0]);

    double (*u)[N+2];   u = malloc((M+1) * sizeof *u);
    double (*v)[N+1];   v = malloc((M+2) * sizeof *v);
    for(j = 0; j < N+2; j++){
        for(i = 0; i < M+1; i++)
            u[i][j] = U[i + (M+1)*j];}
    for(j = 0; j < N+1; j++){
        for(i = 0; i < M+2; i++)
            v[i][j] = V[i + (M+2)*j];}
```

```
    /*************************************************************
     *Begin Computation
     *************************************************************/
    fluxHorizontal(M,N,u,v,psin,mxin,myin,a1in,dx,dt,B);

    /*nx and ny are not normal but sum vectors*/
    free(u); free(v);
    return;
}
```

Curvature.c (Computing Curvature)

```
#include "mex.h"
#include "math.h"
#include "stdint.h"

double sc = 1e-8;
int sign(double x) {return ((x > 0) ? 1 : ((x < 0) ? -1 : 0));}
double slant(double x) {return ((x > 1) ? 1 : ((x < 0) ? 0 : x));}
double step(double x) {return ((x > 0) ? 1.0 : ((x < 0) ?  0.0 : 0.5));}
double min(double a, double b) {return ((a > b) ? b : a);}
double max(double a, double b) {return ((a > b) ? a : b);}

void fix(double cell[3][7]){
}

void curvature(int M, int N, double *psi, double *nx, double *ny, double
*alpha, double dx, double *cur){
    int i,j,m,n;
    for(j = 2; j < N+2; j++){
        for(i = 2; i < M+2; i++){
            int  ind = i + (M+4)*(j);
            double mx = nx[ind]; double my = ny[ind];

            if (alpha[ind]<1e19){
                double cell[3][7]; int dir, dir2, in;
                if (fabs(mx)>=fabs(my)){
                    dir = sign(-mx); dir2 = sign(-my);
                    for (m = -1; m < 2; m++){
                        for (n = -3; n < 4; n++){
                            in = i+n + (M+4)*(j+m*dir2);
                            cell[m+1][n+3] = psi[in];
                        }
                    }
                }
                else{
                    dir = sign(-my); dir2 = sign(-mx);
                    for (m = -1; m < 2; m++){
                        for (n = -3; n < 4; n++){
                            in = i+m + (M+4)*(j+n*dir);
                            cell[m+1][n+3] = psi[in];
                        }
                    }
                }

                double H[3] = {0};
```

```
                for (m = 0; m < 3; m++){
                    for (n = 0; n < 7; n++){
                        H[m] += cell[m][n]*dx;
                    }
                }
                //H[1] =
dx*max(cell[1][0]+cell[1][1]+cell[1][2]+cell[1][3],cell[1][3]+cell[1][4]+cell
[1][5]+cell[1][6]);
                double h1x2 = (H[1]-H[0])/(dx)*(H[1]-H[0])/(dx);
                double den = pow(1.0+h1x2,1.5);
                double numer = -(H[0]-2*H[1]+H[2])/(dx*dx);
                double numer2 = -2.0*(H[0]-2*H[1]+H[2])*dx;
                double den2 = sqrt((dx*dx+(H[1]-H[0])*(H[1]-
H[0]))*(dx*dx+(H[2]-H[1])*(H[2]-H[1]))*(4.0*dx*dx+(H[2]-H[0])*(H[2]-H[0])));
                cur[ind] = min(1/dx,fabs(numer2/den2))*sign(numer2/den2);


                if (0*(fabs(mx)>fabs(my))){
                    mexPrintf("%g %g\n",mx,my);
                    int dir3 = sign(-my), dir4 = sign(-mx);
                    for (m = -1; m < 2; m++){
                        for (n = -3; n < 4; n++){

mexPrintf("[%.2f,%.2f]\t",cell[m+1][n+3],psi[i+n*dir3 + (M+4)*(j+m*dir4)]);
                        }
                        mexPrintf("\n");
                    }
                    mexPrintf("Curve: %f\t%f\t%f\n",H[0],H[1],H[2]);
                    mexPrintf("%f\n",cur[ind]);
                    //goto breakout;
                }




            }
        }
    }breakout: 0;//mexPrintf("Exited");
}


void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    /* Macros for the ouput and input arguments */
    double *B, *U, *mxin,  *myin, *alpha, *psin, dx, dy, dt;
    int M, N, i, j, k, num;
    M = mxGetScalar(prhs[0]); /* Get the dimensions of A */
    N = mxGetScalar(prhs[1]);
    mxin = mxGetPr(prhs[2]);
    myin = mxGetPr(prhs[3]);
    psin = mxGetPr(prhs[4]);
    alpha = mxGetPr(prhs[5]);
    dx = mxGetScalar(prhs[6]);
```

```
    plhs[0] = mxCreateDoubleMatrix(M+4, N+4, mxREAL); /* Create the output
matrix */

    B = mxGetPr(plhs[0]);

    /*************************************************************
     *Begin Computation
     *************************************************************/

    curvature(M,N,psin,mxin,myin,alpha,dx,B);

    /*nx and ny are not normal but sum vectors*/

    return;
}
```

PLIC.c (PLIC Reconstruction)

```
#include "mex.h"
#include "math.h"
#include "stdint.h"

double sign(double x) {return ((x > 0) ? 1.0 : ((x < 0) ? -1.0 : 0.0));}
double step(double x) {return ((x > 0) ? 1.0 : ((x < 0) ?  0.0 : 0.5));}
double min(double a, double b) {return ((a > b) ? b : a);}
double max(double a, double b) {return ((a > b) ? a : b);}

double alphacomp(double psi, double mx1, double my1){
    double a = 0, m1, m2, psi1, chi, snrm, mx, my; //computing alpha for mx
    snrm = fabs(mx1)+fabs(my1);
    mx = mx1/snrm;
    my = my1/snrm;
    if ((psi<1.0e-12) || (psi>(1.0-1.0e-12))){
        a = 1.0e20;
    }
    else{
        m1 = min(fabs(mx),fabs(my));
        m2 = max(fabs(mx),fabs(my));
        psi1 = m1/(2.0*(1.0-m1));
        chi = min(psi,1.0-psi);
        if (chi<psi1)
            a = sqrt(2.0*m1*(1-m1)*chi);
        if ((chi>=psi1) && (chi<=.5))
            a = (1-m1)*chi+m1/2.0;
        if (psi>.5)
            a = 1.0-a;

    }
    return a;
}

void normalvec(int M, int N,  double dx, double dy, double p[M+4][N+4],
        double nx[M+4][N+4], double ny[M+4][N+4], double alpha[M+4][N+4]){
    int16_t i,j;
    double (*mx )[N+4]; mx  = malloc((M+4) * sizeof *mx );
    double (*my )[N+4]; my  = malloc((M+4) * sizeof *my );
```

```
    for (i = 1; i < M+3; i++){
        for(j = 1; j < N+3; j++){
            mx[i][j] = (p[i+1][j-1]-p[i-1][j-1]) +
                    2.0*(p[i+1][j]-p[i-1][j]) +
                    (p[i+1][j+1]-p[i-1][j+1]);
            my[i][j] = (p[i-1][j+1]-p[i-1][j-1]) +
                    2.0*(p[i][j+1]-p[i][j-1]) +
                    (p[i+1][j+1]-p[i+1][j-1]);
        }
    }

    for (i = 0; i < M+4; i++){
        for(j = 0; j < N+4; j++){
            int BC = (i==0) || (i==M+3) || (j==0) || (j==N+3);
            nx[i][j] = mx[i][j]/(fabs(mx[i][j])+fabs(my[i][j]));
            ny[i][j] = my[i][j]/(fabs(mx[i][j])+fabs(my[i][j]));
            alpha[i][j] = alphacomp(1-p[i][j],nx[i][j],ny[i][j]);

            if (((mx[i][j]==0) && (my[i][j]==0)) || BC){
                nx[i][j] = 0.0;
                ny[i][j] = 0.0;
                alpha[i][j] = 1e20;
            }

            if ((p[i][j]<1e-6) || (p[i][j]>(1-1e-6))){
                nx[i][j] = 0.0;
                ny[i][j] = 0.0;
                alpha[i][j] = 1e20;
            }
            alpha[i][j] = alpha[i][j] + min(0.0,nx[i][j]) +
min(0.0,ny[i][j]);
        }
    }


    free(mx); free(my);
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    /* Macros for the ouput and input arguments */
    double *B,  *C, *D, *psin, dx, dy, dt;
    int M, N, i, j, k, num;
    M = mxGetScalar(prhs[0]); /* Get the dimensions of A */
    N = mxGetScalar(prhs[1]);
    dx =  mxGetScalar(prhs[2]);
    dy =  dx;

    psin = mxGetPr(prhs[3]);

    plhs[0] = mxCreateDoubleMatrix(M+4, N+4, mxREAL); /* Create the output
matrix */
```

```
    plhs[1] = mxCreateDoubleMatrix(M+4, N+4, mxREAL); /* Create the output
matrix */
    plhs[2] = mxCreateDoubleMatrix(M+4, N+4, mxREAL); /* Create the output
matrix */

    B = mxGetPr(plhs[0]);
    C =  mxGetPr(plhs[1]);
    D =  mxGetPr(plhs[2]);

    double (*psi)[N+4]; psi= malloc((M+4) * sizeof *psi );
    double (*nx )[N+4]; nx = malloc((M+4) * sizeof *nx );
    double (*ny )[N+4]; ny = malloc((M+4) * sizeof *ny );
    double (*alpha )[N+4]; alpha = malloc((M+4) * sizeof *alpha );


    for(j = 0; j < N+4; j++){
        for(i = 0; i < M+4; i++){
            psi[i][j] = psin[i + (M+4)*j];
        }
    }
    /*************************************************************
     *Begin Computation
     *************************************************************/

    normalvec(M, N, dx, dy, psi, nx, ny, alpha);
    /*nx and ny are not normal but sum vectors*/

    /*************************************************************
     *End Computation
     *************************************************************/
    for(i = 0; i < M+4; i++){
        for(j = 0; j < N+4; j++){
            B[i + (M+4)*j] = nx[i][j];
            C[i + (M+4)*j] = ny[i][j];
            D[i + (M+4)*j] = alpha[i][j];
        }
    }
    free(psi); free(nx); free(ny); free(alpha);
    return;
}
```

Phi2psi.c (Converting phi field of psi)

```
#include "mex.h"
#include "math.h"
#include "stdint.h"

double sign(double x) {return ((x > 0) ? 1.0 : ((x < 0) ? -1.0 : 0.0));}
double step(double x) {return ((x > 0) ? 1.0 : ((x < 0) ?  0.0 : 0.5));}
double stepsmooth(double x, double e) {return ((x > e) ? 1.0 : ((x < -e) ?
0.0 : (.5+.5*sin(3.1415926535*x/e))));}

double min(double a, double b) {return ((a > b) ? b : a);}
double max(double a, double b) {return ((a > b) ? a : b);}
```

```
double topsi(double lb, double rb,double lt, double rt){
    if ((lb>0.0) &&(rb>0.0)&& (lt>0.0) &&(rt>0.0)){
        return 1.0;
    }
    else if ((lb<0.0) &&(rb<0.0)&& (lt<0.0) &&(rt<0.0)){
        return 0.0;
    }
    else{
        double a,b;
        if ((lb*lt)<0 && (lb*rb)<0){
            a = fabs(lb)/(fabs(lb)+fabs(lt));
            b = fabs(lb)/(fabs(lb)+fabs(rb));
            if (lb>0.0) return 0.5*a*b;
            else return 1.0-0.5*a*b;
        }
        else if ((rb*lb)<0 && (rb*rt)<0){
            a = fabs(rb)/(fabs(rb)+fabs(lb));
            b = fabs(rb)/(fabs(rb)+fabs(rt));
            if (rb>0.0) return 0.5*a*b;
            else return 1.0-0.5*a*b;
        }
        else if ((rt*rb)<0 && (rt*lt)<0){
            a = fabs(rt)/(fabs(rt)+fabs(rb));
            b = fabs(rt)/(fabs(rt)+fabs(lt));
            if (rt>0.0) return 0.5*a*b;
            else return 1.0-0.5*a*b;
        }
        else if ((lt*rt)<0 && (lt*lb)<0){
            a = fabs(lt)/(fabs(lt)+fabs(rt));
            b = fabs(lt)/(fabs(lt)+fabs(lb));
            if (lt>0.0) return 0.5*a*b;
            else return 1.0-0.5*a*b;
        }
        else if ((lt*lb>0) && (rt*rb>0) && (lb*rb<0) && (lt*rt<0)){
            a = fabs(lt)/(fabs(lt)+fabs(rt));
            b = fabs(lb)/(fabs(lb)+fabs(rb));
            if (lt>0.0) return 0.5*(a+b);
            else return 1.0-0.5*(a+b);
        }
        else if (!(lt*lb>0) && !(rt*rb>0) && !(lb*rb<0) && !(lt*rt<0)){
            a = fabs(rb)/(fabs(rb)+fabs(rt));
            b = fabs(lb)/(fabs(lb)+fabs(lt));
            if (rb>0.0) return 0.5*(a+b);
            else return 1.0-0.5*(a+b);
        }
        else{
            printf("This should never happen but just in case\n");
            double s,e,x,y;
            int i,j, N = 100;
            s = 0;
            e = max(fabs(lt-lb),fabs(rt-rb))/N;
            for(i = 1; i < N+1; i++){
                for(j = 1; j < N+1; j++){
                    x = ((double)i) * 1.0/N-1.0/(2.0*N);
                    y = ((double)j) * 1.0/N-1.0/(2.0*N);
```

```
                    s += stepsmooth(y*(rt*x - lt*(x - 1.0)) - (rb*x - lb*(x -
1.0))*(y - 1.0),e);
                }
            }
            return s/(N*N);
        }
    }
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    /* Macros for the ouput and input arguments */
    double *psi, *phi, dx, dy, dt;
    int M, N, i, j, k, num;
    M = mxGetScalar(prhs[0]); /* Get the dimensions of A */
    N = mxGetScalar(prhs[1]);
    phi = mxGetPr(prhs[2]);

    plhs[0] = mxCreateDoubleMatrix(M+4, N+4, mxREAL); /* Create the output
matrix */

    psi = mxGetPr(plhs[0]);

    for(j = 0; j < N+4; j++){
        for(i = 0; i < M+4; i++){
            psi[i + (M+4)*j] = 0;
        }
    }
    /*************************************************************
     *Begin Computation
     *************************************************************/
    double phic, lb, rb, lt, rt;
    for(j = 1; j < N+3; j++){
        for(i = 1; i < M+3; i++){
            phic = phi[i + (M+4)*j];
            lb = 0.5*phic + 0.5*phi[(i-1) + (M+4)*(j-1)];
            rb = 0.5*phic + 0.5*phi[(i+1) + (M+4)*(j-1)];
            lt = 0.5*phic + 0.5*phi[(i-1) + (M+4)*(j+1)];
            rt = 0.5*phic + 0.5*phi[(i+1) + (M+4)*(j+1)];
            psi[i + (M+4)*j] = topsi(lb,rb,lt,rt);
        }
    }

    /*nx and ny are not normal but sum vectors*/
    /*************************************************************
     *End Computation
     *************************************************************/

    return;
}
```

PressureSolver.c (Pressure Matrix Value Generator)

```
#include "mex.h"
#include "math.h"
#include "stdint.h"
#include "omp.h"
```

```
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{

    /* Macros for the ouput and input arguments */
    double *pin, *phi, *d2pin, *rhs, *rhoin, *I, *J, *X, dx,divi;
    int32_t *num;
    int M, N, i, j, k;
    pin = mxGetPr(prhs[0]);
    d2pin = mxGetPr(prhs[1]);
    rhoin =  mxGetPr(prhs[2]);
    num = mxGetData(prhs[3]);
    dx =  mxGetScalar(prhs[4]);
    M = mxGetM(prhs[0])-2; /* Get the dimensions of A */
    N = mxGetN(prhs[0])-2;
    int Mc = 8*M*N - 4*N - 4*M;
    plhs[0] = mxCreateDoubleMatrix(1, Mc, mxREAL); /* Create the output
matrix */
    plhs[1] = mxCreateDoubleMatrix(1, Mc, mxREAL);
    plhs[2] = mxCreateDoubleMatrix(1, Mc, mxREAL);
    plhs[4] = mxCreateDoubleMatrix(M*N, 1, mxREAL);
    plhs[3] = mxCreateDoubleMatrix(M*N, 1, mxREAL);
    I = mxGetPr(plhs[0]);
    J = mxGetPr(plhs[1]);
    X = mxGetPr(plhs[2]);
    phi = mxGetPr(plhs[4]);
    rhs = mxGetPr(plhs[3]);
    k = 0;
    for(j = 1; j < N+1; j++){
        for(i = 1; i < M+1; i++){
            phi[k] = pin[i + (M+2)*j];
            rhs[k] = d2pin[i + (M+2)*j];
            k++;
        }
    }
    /*************************************************
     *Begin Computation
     *************************************************/
    k = 0;
    for(j = 1; j < N+1; j++){
        for(i = 1; i < M+1; i++){
            if (num[i+1 + (M+2)*j]){
                divi = 1/(dx*dx*(rhoin[i + (M+2)*j]+rhoin[i+1 + (M+2)*j]));
                I[k]=num[i + (M+2)*j];
                J[k]=num[i + (M+2)*j];
                X[k]=-1.0*divi;
                k++;
                I[k]=num[i + (M+2)*j];
                J[k]=num[i+1 + (M+2)*j];
                X[k]=1.0*divi;
                k++;

            }
            if (num[i-1 + (M+2)*j]){
                divi = 1/(dx*dx*(rhoin[i + (M+2)*j]+rhoin[i-1 + (M+2)*j]));
```

```
                I[k]=num[i + (M+2)*j];
                J[k]=num[i + (M+2)*j];
                X[k]=-1.0*divi;
                k++;
                I[k]=num[i + (M+2)*j];
                J[k]=num[i-1 + (M+2)*j];
                X[k]=1.0*divi;
                k++;
            }
            if (num[i + (M+2)*(j+1)]){
                divi = 1/(dx*dx*(rhoin[i + (M+2)*j]+rhoin[i + (M+2)*(j+1)]));
                I[k]=num[i + (M+2)*j];
                J[k]=num[i + (M+2)*j];
                X[k]=-1.0*divi;
                k++;
                I[k]=num[i + (M+2)*j];
                J[k]=num[i + (M+2)*(j+1)];
                X[k]=1.0*divi;
                k++;
            }
            if (num[i + (M+2)*(j-1)]){
                divi = 1/(dx*dx*(rhoin[i + (M+2)*j]+rhoin[i + (M+2)*(j-1)]));
                I[k]=num[i + (M+2)*j];
                J[k]=num[i + (M+2)*j];
                X[k]=-1.0*divi;
                k++;
                I[k]=num[i + (M+2)*j];
                J[k]=num[i + (M+2)*(j-1)];
                X[k]=1.0*divi;
                k++;
            }
        }
    }
    /*********************************************************
     *End Computation
     *********************************************************/
    return;
}
```

Diffmat2.m (Appending Diffusion Matrix)

```
function D = diffmat2(M,N,dt,dx,mu,rho)
%inputs
if nargin == 0
M = 128/8; N = 64/8; dt = 1e100; dx = 0.01;mu = 1; rho = ones(M+2,N+2);
rho(3:5,3:5) = 10;
end

uind = zeros(M+1,N+2, 'int32');
uind(2:end-1,2:end-1) = int32(reshape(1:((M-1)*N),[M-1 N]));
uind(:,1) = uind(:,2);  uind(:,N+2) = uind(:,N+1);
vind = zeros(M+2,N+1, 'int32');
vind(2:end-1,2:end-1) = int32(reshape(1:(M*(N-1)),[M N-1])) + max(max(uind));
vind(1,:) = vind(2,:);  vind(M+2,:) = vind(M+1,:);

i = 2:M;
j = 2:N+1; sizeu = numel(i)*numel(j);
Iu = zeros(8,sizeu); Ju = Iu; Valu = Iu; e = 1;
```

```
rhoh = 0.5*(rho(i+1,j)+rho(i,j));

[e, Iu, Ju, Valu] = appendSparse(i,j,uind,uind, 1,0,Iu,Ju,Valu,e, mu./rhoh);
[e, Iu, Ju, Valu] = appendSparse(i,j,uind,uind, 0,0,Iu,Ju,Valu,e,-
mu./rhoh+1/dt);
[e, Iu, Ju, Valu] = appendSparse(i,j,uind,uind, 0,0,Iu,Ju,Valu,e,-mu./rhoh);
[e, Iu, Ju, Valu] = appendSparse(i,j,uind,uind,-1,0,Iu,Ju,Valu,e, mu./rhoh);

[e, Iu, Ju, Valu] = appendSparse(i,j,uind,uind, 0,1,Iu,Ju,Valu,e, mu./rhoh);
[e, Iu, Ju, Valu] = appendSparse(i,j,uind,uind, 0,0,Iu,Ju,Valu,e,-
mu./rhoh+1/dt);
[e, Iu, Ju, Valu] = appendSparse(i,j,uind,uind, 0,0,Iu,Ju,Valu,e,-mu./rhoh);
[~, Iu, Ju, Valu] = appendSparse(i,j,uind,uind,0,-1,Iu,Ju,Valu,e, mu./rhoh);

i = 2:M+1;
j = 2:N; sizev = numel(i)*numel(j);
Iv = zeros(8,sizev); Jv = Iv; Valv = Iv; e = 1;
rhoh = 0.5*(rho(i,j+1)+rho(i,j));

[e, Iv, Jv, Valv] = appendSparse(i,j,vind,vind,0, 1,Iv,Jv,Valv,e, mu./rhoh);
[e, Iv, Jv, Valv] = appendSparse(i,j,vind,vind,0, 0,Iv,Jv,Valv,e,-
mu./rhoh+1/dt);
[e, Iv, Jv, Valv] = appendSparse(i,j,vind,vind,0, 0,Iv,Jv,Valv,e,-mu./rhoh);
[e, Iv, Jv, Valv] = appendSparse(i,j,vind,vind,0,-1,Iv,Jv,Valv,e, mu./rhoh);

[e, Iv, Jv, Valv] = appendSparse(i,j,vind,vind,1, 0,Iv,Jv,Valv,e, mu./rhoh);
[e, Iv, Jv, Valv] = appendSparse(i,j,vind,vind,0, 0,Iv,Jv,Valv,e,-
mu./rhoh+1/dt);
[e, Iv, Jv, Valv] = appendSparse(i,j,vind,vind,0, 0,Iv,Jv,Valv,e,-mu./rhoh);
[~, Iv, Jv, Valv] = appendSparse(i,j,vind,vind,-1,0,Iv,Jv,Valv,e, mu./rhoh);

D = sparse(Iv(Iv~=0&Jv~=0),Jv(Iv~=0&Jv~=0),Valv(Iv~=0&Jv~=0));
D(1:sizeu,1:sizeu) =
sparse(Iu(Iu~=0&Ju~=0),Ju(Iu~=0&Ju~=0),Valu(Iu~=0&Ju~=0));

D = D/dx^2;
```

AppendSparse.m (Append individual Diffusion Components in a Vectorized Fashion)

```
function [e, Ic, Jc, Valc] =
appendSparse(i,j,uind,vind,nx,ny,Ic,Jc,Valc,e,Value)
Ic(e,uind(i,j)) = reshape(uind(i,j), [1 numel(i)*numel(j)]);
Jc(e,uind(i,j)) = reshape(vind(i+nx,j+ny), [1 numel(i)*numel(j)]);
if length(Value)==1
    Valc(e,uind(i,j)) =  Value;
else
    Valc(e,uind(i,j)) =  reshape(Value, [1 numel(i)*numel(j)]);
end
e = e + 1;
```

Nuemannfft.m (Solving single coefficient poisson equation directly using fft)

```
function [sol,v] = nuemannfft(M,N,f,h)
if nargin==0
    M = 200;
    N = 100;
    h = 1/100;
```

```
    f = zeros(M+2,N+2);
    f(4,4) = 1;
    f(197,97) = -1.0;
end

kx = repmat((0:2*M-1)',1,2*N);
ky = repmat((0:2*N-1) ,2*M,1);
w = -(2/h)^2*( (sin(.5*kx*pi/M)).^2 + (sin(.5*ky*pi/N)).^2);
w(1,1) = 1;
siz = 2*[M N];
% rhs


i = 2:M+1; j = 2:N+1; fi = flip(i); fj = flip(j);
g = [f(i,j) f(i,fj);
    f(fi,j) f(fi,fj)]; %mirror in X and Y

v = ifftn(   fftn(g,siz)./w    ,siz);
sol = f*0;
sol(i,j) = real(v(i-1,j-1));
sol(1,:) = sol(2,:); sol(M+2,:) = sol(M+1,:);
sol(:,1) = sol(:,2); sol(:,N+2) = sol(:,N+1);
```

PlotInterface.m (Code to Plot Interface)

```
function [] = plotInterface(psi,alpha,mx,my,x,y,X,Y,dx,dy,str)
[I, J] = find(alpha<1e19);
x1 = X(alpha<1e19);
y1 = Y(alpha<1e19);
m1 = mx(alpha<1e19);
m2 = my(alpha<1e19);
m1(m1==0)=1e-7;
m2(m2==0)=1e-7;
alpha = alpha(alpha<1e19);

dat = [zeros(size(alpha)),alpha./m2,    ones(size(alpha)),(alpha-m1)./m2,
alpha./m1,zeros(size(alpha)),   (alpha-m2)./m1,ones(size(alpha))];
dat(dat<0 | dat>1)= -1e20;
dat1 = zeros(size(x1,1),4);
for i = 1:length(alpha)
    j = find(dat(i,:)<-1e-19);
    j2 = j;
    j2(j<5) = j(j<5)-1; j2(j>4) = j(j>4)+1;
    tdat = dat(i,:);
    tdat([j j2])=[];
    dat1(i,:) = tdat;

end

imagesc(x,y,psi');    caxis([0 1]);
hold on
for i = 1:length(I)
%     plot(x(I(i)), y(J(i)), 'kx')
    plot(x(I(i))-dx/2+dx*dat1(i,[1 3]),y(J(i))-dy/2+dy*dat1(i,[2
4]),'k','linewidth',1);
```

```
end
% xlabel('x-axis'); ylabel('y-axis');
set(gca,'Ydir','Normal');
axis equal;
% colormap(parula);
% colorbar;
% for j = 2:N+3
%     plot([x(1),x(end)],[y(j) y(j)]-dy/2,'-','color',[0.9 0.9 0.9 .2])
% end
% for i = 2:M+3
%     plot([x(i) x(i)]-dx/2,[y(1),y(end)],'-','color',[0.9 0.9 0.9 .2])
% end

% plot(x(I),y(J),'ro')
hold off
```

Project_main.m (Main Project Code)

```
M = 100;
N = M;
dim = [0 1 0 1]/100;
rho_liquid = 1000; %kg/m^3
rho_gas    = 100; %kg/m^3
mu_liquid = 1e-4; %N·s/m^2
mu_gas    = mu_liquid;%N·s/m^2
surface_tension_coefficient = 0.07; %N/m
dx = (dim(2)-dim(1))/M;
dy = (dim(4)-dim(3))/N;
h = dx;
t = 0;
k = 0; %iteration count
dist = 1e-10*pi;
x = linspace(dim(1)-3*dx/2,dim(2)+3*dx/2,M+4)+dist; %cell center locations
y = linspace(dim(3)-3*dy/2,dim(4)+3*dx/2,N+4)+dist;
xf = linspace(dim(1),dim(2),M+1)+dist; %wall locations
yf = linspace(dim(3),dim(4),N+1)+dist;
[X, Y] = meshgrid(x,y); X = X'; Y = Y'; %mesh Grid
x = linspace(dim(1)-dx/2,dim(2)+dx/2,M+2)+dist; %cell center locations
y = linspace(dim(3)-dy/2,dim(4)+dx/2,N+2)+dist;
[XU, YU] = meshgrid(xf,y); XU = XU'; YU = YU';
[XV, YV] = meshgrid(x,yf); XV = XV'; YV = YV';
x = linspace(dim(1)-3*dx/2,dim(2)+3*dx/2,M+4)+dist; %cell center locations
y = linspace(dim(3)-3*dy/2,dim(4)+3*dx/2,N+4)+dist; %bring it back

R = ((X-.005).^2+(Y-.0075).^2).^.5;
phi = .0015-R;

for i = 1:length(x)
    for j = 1:length(y)
        phi(i,j) = max(11.1025-sqrt((.005-x(i))^2+(-11.1015-y(j))^2),.0015-
sqrt((.005-x(i))^2+(.0075-y(j))^2));
    end
end
BC3 = @(p,M,N) p([4 3 3:M+2 M+2 M+1],[4 3 3:N+2 N+2 N+1]);
BC2 = @(p,M,N) p([2 2:M+1 M+1],[2 2:N+1 N+1]);
MB1 = @(p)  p(2:end-1,2:end-1);
```

```
BCu = @(u,M,N) u(1:M+1,[2 2:N+1 N+1]); BCv = @(v,M,N) v([2 2:M+1 M+1],1:N+1);

psi = phi2psi(M,N,phi); %compute psi from phi
psi = BC3(psi,M,N);
area = sum(sum(psi))*dx*dx; % compute Area
[mx, my, alpha] = PLIC(M,N,dx,psi); %compute mx my and alpha

phi = zeros(M+2,N+2); div = zeros(M+2,N+2); %phi becomes pressure
u =  -sin(pi*XU).^2.*sin(2*pi*YU)*dist*dist;
v =  sin(2*pi*XV).*sin(pi*YV).^2*dist*dist;
un = u*0;
vn = v*0;
time = 0:0.02:.2;
ti = [];
vol = [];
for count = 1:size(time,2)
    while t < time(count)
        maxu = max(max(abs(u)));
        maxv = max(max(abs(v)));
        dtu = h/(2*maxu+maxv);
        dtv = h/(maxu+2*maxv);
        dtc =
sqrt((rho_gas+rho_liquid)*dx^3/(4*pi*surface_tension_coefficient));
        dtp = 1/4*dx^2/(mu_liquid/rho_liquid);

        dt = min([0.01*pi 0.5*dtu 0.5*dtv dtc dtp time(count)-t]);

        %%%%%%%%%%  VOF advection
        rho = psi*rho_liquid + (1-psi)*rho_gas;
        mu = psi*mu_liquid + (1-psi)*mu_gas;

        psi = cellvof(M,N,u,v,mx,my,alpha,psi,dy,dt);
        psi = BC3(psi,M,N);
        [mx, my, alpha] = PLIC(M,N,dx,psi); %PLIC Recon
        mx = BC3(mx,M,N); my = BC3(my,M,N);
        mx([1 2 M+3 M+4],:) = -mx([1 2 M+3 M+4],:);
        my(:,[1 2 N+3 N+4]) = -my(:,[1 2 N+3 N+4]);


        rhon = psi*rho_liquid + (1-psi)*rho_gas;

        cur = curvature(M,N,mx,my,psi,alpha,dx);
        Tx = surface_tension_coefficient*mx./sqrt(mx.^2+my.^2).*cur;
Tx(isnan(Tx)) = 0;
        Ty = surface_tension_coefficient*my./sqrt(mx.^2+my.^2).*cur;
Ty(isnan(Ty)) = 0;
        %Precompute Advection
        %%%%%%%%%%   Momentum Advection
        i = 2:M;
        j = 2:N+1;
        gammax = 1+0*max(abs(u(i,j))*dt/dx,abs(v(i,j))*dt/dx);
        duudx = 0.25*((u(i+1,j)+u(i,j)).^2-(u(i,j)+u(i-1,j)).^2)/dx + ...
            0.25*gammax.*(abs(u(i+1,j)+u(i,j)).*(u(i,j)-u(i+1,j)) - ...
            abs(u(i,j)+u(i-1,j)).*(u(i-1,j)-u(i,j)))/dx;
        duvdy = 0.25*((u(i,j+1)+u(i,j)).*(v(i+1,j)+v(i,j))-...
```

```
            (u(i,j)+u(i,j-1)).*(v(i+1,j-1)+v(i,j-1)))/dy+...
            0.25*gammax.*((u(i,j)-u(i,j+1)).*abs(v(i+1,j)+v(i,j))-...
            (u(i,j-1)-u(i,j)).*abs(v(i+1,j-1)+v(i,j-1)))/dy;
        laplu = (u(i+1,j)-2*u(i,j)+u(i-1,j))/dx^2+(u(i,j+1)-2*u(i,j)+u(i,j-
1))/dy^2;
        tx = (0.5*(Tx(i+1,j+1)+Tx(i+2,j+1)));
        un(i,j) = u(i,j)+dt*( -duudx -duvdy + tx + ...
            mu_gas./(0.5*(rho(i+1,j+1) + rho(i+2,j+1))).*laplu);

        i = 2:M+1;
        j = 2:N;
        gammay = 1+0*max(abs(u(i,j))*dt/dx,abs(v(i,j))*dt/dx);
        dvudx = 0.25*((u(i,j+1)+u(i,j)).*(v(i+1,j)+v(i,j))-...
            (u(i-1,j+1)+u(i-1,j)).*(v(i,j)+v(i-1,j)))/dx + ...
            0.25*gammay.*(abs(u(i,j+1)+u(i,j)).*(v(i,j)-v(i+1,j))-...
            abs(u(i-1,j+1)+u(i-1,j)).*(v(i-1,j)-v(i,j)))/dx;
        dvvdy = 0.25*((v(i,j+1)+v(i,j)).^2-(v(i,j)+v(i,j-1)).^2)/dy + ...
            0.25*gammay.*(abs(v(i,j+1)+v(i,j)).*(v(i,j)-v(i,j+1))-...
            abs(v(i,j)+v(i,j-1)).*(v(i,j-1)-v(i,j)))/dy;
        laplv = (v(i+1,j)-2*v(i,j)+v(i-1,j))/dx^2+(v(i,j+1)-2*v(i,j)+v(i,j-
1))/dy^2;
        ty = (0.5*(Ty(i+1,j+1)+Ty(i+1,j+2)));
        vn(i,j)=v(i,j)+dt*(-dvudx - dvvdy + ty +...
            mu_gas./(0.5*(rho(i+1,j+1)+rho(i+1,j+2))).*laplv-9.8);
        un(:,end) =  un(:,end-1);%top
        un(:,1) = un(:,2);%bottom
        vn(1,:) = vn(2,:);%left
        vn(end,:) = vn(end-1,:);%right

        Qu = -duudx-duvdy+tx;
        Qv = -dvudx-dvvdy+ty-9.8; % Add Gravity to vertical velocity

        uv = [reshape(u(2:end-1,2:end-1),[numel(Qu) 1]);
            reshape(v(2:end-1,2:end-1),[numel(Qv) 1])];
        Quv = [reshape(Qu,[numel(Qu) 1]);
            reshape(Qv,[numel(Qv) 1])];


        %%%% Viscous Diffusion Implicit
        drhs = diffmat2(M,N,dt,dx,1e-4,MB1(rho))*uv+dt*Quv;
        D = diffmat2(M,N,dt,dx,1e-4,MB1(rhon));
        c = symrcm(D); D = D(c,c);
        [L,U] = ilu(D,struct('type','ilutp','droptol',1e-2));
        uv = gmres(D,drhs(c),[],1e-5,20,L,U,pressure);
        uv(c) = uv;
        un(2:end-1,2:end-1) = reshape(uv(1:numel(Qu)),[M-1 N]);
        vn(2:end-1,2:end-1) = reshape(uv(numel(Qu)+1:end),[M N-1]);

        un(:,end) =  un(:,end-1);%top
        un(:,1) = un(:,2);%bottom
        vn(1,:) = vn(2,:);%left
        vn(end,:) = vn(end-1,:);%right

        %%%% Divergence Correction
        i = 2:M+1;
```

```
        j = 2:N+1;
        div(i,j) = 1/dt*((un(i,j)-un(i-1,j))/dx+(vn(i,j)-vn(i,j-1))/dy);
        num = int32(zeros(M+2,N+2)); num(2:end-1,2:end-1) =
int32(reshape(1:M*N,[M N]));
        [I,J,X,divergence,pressure] =
pressuresolver(phi,div,MB1(rhon),num,dx);
        K = sparse(I,J,X);
        [L,U] = ilu(K,struct('type','ilutp','droptol',1e-4));
        [sol,~,~,~,res] = gmres(K,divergence,[],1e-10,300,L,U,pressure);
        phi(i,j) = reshape(sol,[M N]);
        phi = BC2(phi,M,N);

        %Velocity Correction
        i = 2:M; j = 2:N+1; % correct u-velocity
        u(i,j) = un(i,j) - dt/dx*(phi(i+1,j)-
phi(i,j))./(rhon(i+1+1,j+1)+rhon(i+1,j+1));
        i = 2:M+1; j = 2:N; % correct v-velocity
        v(i,j) = vn(i,j) - dt/dy*(phi(i,j+1)-
phi(i,j))./(rhon(i+1,j+1+1)+rhon(i+1,j+1));
        %u = BCu(u,M,N); v = BCv(v,M,N);
        u(:,end) =  u(:,end-1);%top
        u(:,1) = u(:,2);%bottom
        v(1,:) = v(2,:);%left
        v(end,:) = v(end-1,:);%right

        i = 50;
        j = 30;
        (u(i,j)-u(i-1,j))/dx+(v(i,j)-v(i,j-1))/dy; % div check

        t = t + dt;
        k = k + 1;
        ti(k) = t;
        vol(k) = sum(sum(psi(3:end-2,3:end-2)))*dx*dy/(dim(2)*dim(4));
        fprintf('t = %g\tA = %g\n',t,vol(k));
        %%%%%%%%%  End Loop
    end
    subplot(2,5,count)
    plotInterface(psi,alpha,mx,my,x,y,X,Y,dx,dy,'k');
    axis(dim);
    title(num2str(t,'t = %3f'));drawnow;
    %writeVideo(vid,getframe(gcf));
end
```

MultigridTest.m (Multigrid Code – Not Used)

```
function [phi, converge] = multigridTest(M,N,ph,div,rho,dx,niter)
level = log2(min(M,N));
K = cell(1,level);
L = cell(1,level);
U = cell(1,level);
Res = cell(1,level);
R = cell(1,level);
rhs = cell(1,level);
eps = cell(1,level);
diver = reshape(div(2:end-1,2:end-1),[M*N 1]);
m = M; n = N;
% ind1 = repmat((0:N-1)*M+repmat([0 1],1,N/2),M/2,1)+repmat((1:2:M)',1,N);
% ind2 = repmat((0:N-1)*M+repmat([0 -1],1,N/2),M/2,1)+repmat((2:2:M)',1,N);
```

```
%initialize Constants
for k = 1:level
    R{k} = zeros(numel(rho(2:end-1,2:end-1)),1);
    rhs{k} = R{k};
    eps{k} = R{k};

    num = zeros(m+2,n+2,'int32'); num(2:end-1,2:end-1) = (reshape(1:m*n,[m
n]));
    [I,J,X] = pressuresolver(zeros(m+2,n+2),zeros(m+2,n+2),rho,num,dx*2^(k-
1));
    K{k} = sparse(I,J,X);
    if k == 1
        [L{k}, U{k}] = ilu(K{k},struct('type','ilutp','droptol',1e-1));
    else
        [L{k}, U{k}] = ilu(K{k},struct('type','nofill','droptol',1));
    end
    I = repmat((1:m/2)',1,4); %I ondices of restriction matrix
    J = [(1:2:m)' (2:2:m)' m+(1:2:m)' m+(2:2:m)']; %J indices of restriction
matrix
    Rs = sparse(I,J,.25); %restriction matrix [.25 .25 0 0 0... 0.25 0.25
......]
    Rc = repmat({Rs}, 1, n/2);
    Res{k} = blkdiag(Rc{:});

    m = m/2; n = n/2;
    j = 2:n+1;  i = 2:m+1;
    rho2 = zeros(m+2,n+2);
    rho2(i,j) = .25*(...
        rho(2*i-1,2*j-1) + rho(2*i-2,2*j-1) + ...
        rho(2*i-1,2*j-2) + rho(2*i-2,2*j-2));%calculate inner points
    rho2(rho2<=3.5)=1; rho2(rho2>3.5)=6;
    %update ghost points
    rho = rho2;
    rho(1,:) =  rho(2,:);
    rho(:,1) =  rho(:,2);
    rho(end,:) =  rho(end-1,:);
    rho(:,end) =  rho(:,end-1);
end
ph = reshape(ph(2:end-1,2:end-1),[M*N 1]);
converge = zeros(1,niter);
for k = 1:niter
    %begin Multigrid Iteration
    i = 1;
    [ph,~] = gmres(K{1},diver,[],[],[], [],[], ph);%smoothen
    R{1} = diver - K{1}*ph; %compute residual
    for i = 2:level
        rhs{i} = Res{i-1}*R{i-1}; %restrict residual
        [eps{i},~] = gmres(K{i},rhs{i},[],[],6, L{i}, U{i}, eps{i});
%smoothen
        R{i} = rhs{i} - K{i}*eps{i}; %compute residual
    end
    for i = (level-1):-1:2
        R{i} = 4*Res{i}'*eps{i+1}; %prolong
        eps{i} = eps{i}+R{i}; %correct
        [eps{i},~] = gmres(K{i},rhs{i},[],[],6, [],[], eps{i}); %smoothen
    end
```

```
    R{1} = 4*Res{1}'*eps{2};%prolong
    ph = ph + R{1};%correct
    converge(k) = norm(diver - K{1}*ph);
    if converge(k)/converge(1) < 1e-4
        converge(k+1:end) = converge(k);
        break;
    end
end
phi = zeros(M+2,N+2);
phi(2:end-1,2:end-1) = reshape(ph,[M N]);
%semilogy(converge/norm(diver),'.-')
phi = phi([2 2:M+1 M+1],[2 2:N+1 N+1]);
```