Introducing Dynamic Walls in an Integer Lattice Gas Simulation

David Jedynak

May 10, 2018

Abstract

The purpose of this project is to integrate dynamic shapes into an Integer Lattice Gas Simulation. This paper will cover several approaches to create dynamic shapes in an Integer Lattice Gas Simulation, the methods used to achieve them, and how well each approach worked. Future development ideas are also discussed.

1 Problem Description

The lattice gas code simulation is currently unable to model systems with dynamic rigid objects. Having the feature to add movable walls would allow for simulation of interactions between solid physical objects and gasses. The project will begin with simple moving walls that will influence particle dynamics, but the particles will not influence the walls dynamics. Afterwards, implementing wall momentum dependent on particle collisions will be developed so the particles will be able to move the walls. The first and second approach are similar in how they are tested but they have a different random distribution for how to move particles to simulate a moving wall pushing on them. The third approach observes how the gas changes when a force pushes it though a tube. The mean particle velocity is compared to a theoretical value.

2 Background

In order to have complex dynamic shapes, simple dynamic walls need to be achieved. A good place to start is with a static wall. There already exists a method of creating static walls in an Integer Lattice Gas Simulation by selectively inverting particle velocities based on their position in the Lattice grid. Each lattice site has 9 different velocity states particles can exist in.

Particles can be reflected by swapping the number of particles in two of these velocity states. Horizontal walls are produced by switching indexes [0,1,2] with [8,7,6] respectively. Vertical walls are produced by switching [0,3,6] with [8,5,2] respectively.

Table 1: Different velocity states in each Lattice Cell, the maximum velocity amplitude in any one direction is 1
. x and y are the Cartesian coordinates of the Lattice in the Grid.

$n[x][y][0] = (-v_x, +v_y)$	$n[x][y][1] = (0, +v_y)$	$n[x][y][2] = (+v_x, +v_y)$
$n[x][y][3] = (-v_x, 0)$	$n[x][y][4] = (0, 0)$	$n[x][y][5] = (+v_x, 0)$
$n[x][y][6] = (-v_x, -v_y)$	$n[x][y][7] = (0, -v_y)$	$n[x][y][8] = (+v_x, -v_y)$

Here is some code defining the links to produce a horizontal wall. A vertical wall can be produced by changing the values being assigned to links[linkcount][0:2] = [0 3 6] and the dimension being iterated over in the for loop:

```
//horizontal walls
  for (int x=x0; x<x1+1; x++){
            links[linkcount][0] = x; //x-position
            links[linkcount][1] = yy2;
            links[linkcount][2] = 0;
            linkcount++;
            links[linkcount][0] = x; //x-position
            links[linkcount][1] = yy2;
            links[linkcount][2] = 1;
            linkcount++;
            links[linkcount][0] = x; //x-position
            links[linkcount][1] = yy2;
            links[linkcount][2] = 2;
            linkcount++;
  }
```

Here is the code that switches the number of particles between lattice velocity states for the static walls. Note that the momentum imparted on the wall can be calculated.

```
void bounceback(){
  dynamic_wall_momentum_x = 0;
  dynamic_wall_momentum_y = 0;
  static_wall_momentum_x = 0;
  static_wall_momentum_y = 0;

  for (int lc=0; lc<linkcount; lc++){
    //quantity of partices
    int x=links[lc][0];
    int y=links[lc][1];
    //velocity
    int v=links[lc][2];
```

```
    int vx=v%3-1;
    int vy=1-v/3;
    int tmp= n[x+vx][y+vy][v];
    //summing all momemtums
    static_wall_momentum_x += -2*vx*(n[x][y][8-v]-tmp);
    static_wall_momentum_y += -2*vy*(n[x][y][8-v]-tmp);
    //swapping the particles trying to enter and leave to have the effect of a wall
    n[x+vx][y+vy][v]= n[x][y][8-v];
    n[x][y][8-v]=tmp;
        }
}
```

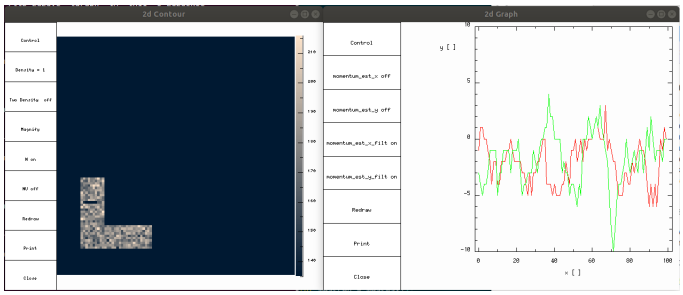

Figure 1: Here are several horizontal and vertical walls produced using the approach described above. The walls are containing the particles from leaving the shape. On the right hand side, there are graphs for the x and y momentum being imparted on the walls

3 Methods

3.1 Approach 1

The first approach taken to creating a moving wall was to move a wall very slowly with its velocity being much less than 1. The wall will move either dependently based on the momentum imparted on the wall due to particle collisions, or a set velocity will be given to wall.

$$\frac{\sum_{n=1}^{nmax} -2 * \text{net particles reflected}}{\text{wall mass}} = \text{wall velocity}$$

Code for calculating the wall momentum and resulting velocity

```
    //summing all momemtums
    dynamic_wall_momentum_x += -2*vx*(n[x][y][8-v]-tmp);
```

```
dynamic_wall_momentum_y += -2*vy*(n[x][y][8-v]-tmp);

    if(dynamic_wall_control_on == 0){
                dynamic_wall_vx = dynamic_wall_momentum_x/wall_mass;
                dynamic_wall_vy = dynamic_wall_momentum_y/wall_mass;
    }
```

As the wall moves, it would "pass over" particles due to rest particles in the $n[x][y][0]$ state. To compensate for this, a random number of particles would be added to either side of the wall depending on the wall's velocity and the density of the particles surrounding the wall. For simplicity, a flat distribution was used ranging from 0 to the smaller density of particles on either side of the wall. This is necessary to prevent negative densities. This number of particles is refereed to as "flow".

Expected value of flow

$$< flow >= \text{particle density} * \text{wall velocity}$$

$$0 < flow < \text{min particle density}$$

The code for calculating flow is defined below. First the 2 particle densities are assigned to 2 temp variables. The smaller of the two is used as the max value in the random distribution.

```
int tmp_0 = n[(((vx+x)%XDIM)+XDIM)%XDIM][((y+vy)%YDIM+YDIM)%YDIM][8-v];
  int tmp = n[(((vx+x)%XDIM)+XDIM)%XDIM][((y+vy)%YDIM+YDIM)%YDIM][v];
      int max_random = 1;
      if(tmp > tmp_0){
              max_random = tmp_0;
              }
      else{
              max_random = tmp;
              }
      if(max_random > 0){
              flow = (rand()%max_random)*dynamic_wall_vx;
              }
      else{
              flow = 0;
              }
```

The following wall momentum is then:

```
dynamic_wall_momentum_x += -2*vx*(n[x][y][8-v]-tmp+flow);
dynamic_wall_momentum_y += -2*vy*(n[x][y][8-v]-tmp);
```

Adding the flow to the lattice sites to produce a moving wall

```
//swapping the particles trying to enter and leave to have the effect of a wall
n[(((vx+x)%XDIM)+XDIM)%XDIM][((y+vy)%YDIM+YDIM)%YDIM][v] = n[(((x)%XDIM)+XDIM)%XDIM][((y
n[(((x)%XDIM)+XDIM)%XDIM][((y)%YDIM+YDIM)%YDIM][8-v] = tmp + flow;
```

3.2 Results

Initial results were promising, but this approach did not factor in the number of particles at rest position, so there was significant particle leakage.

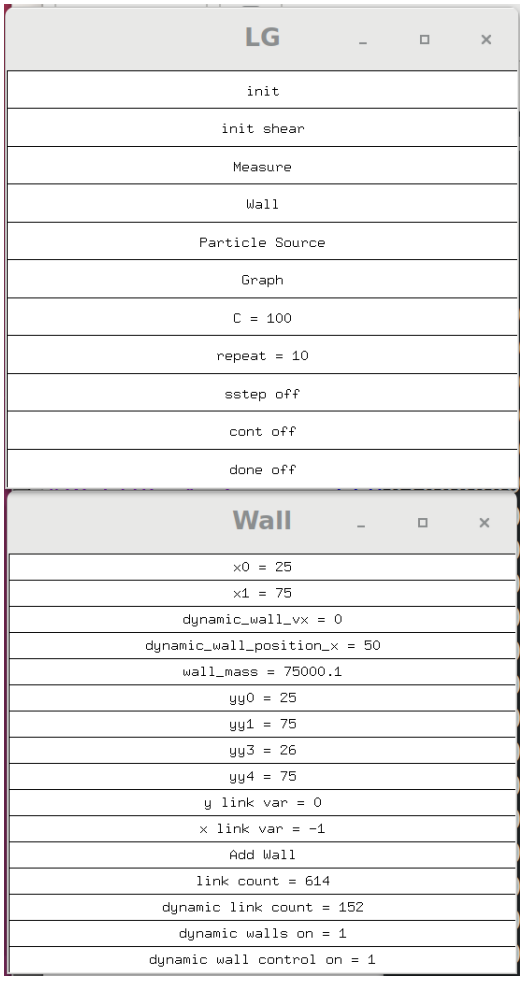

Figure 2: Settings for the simulation

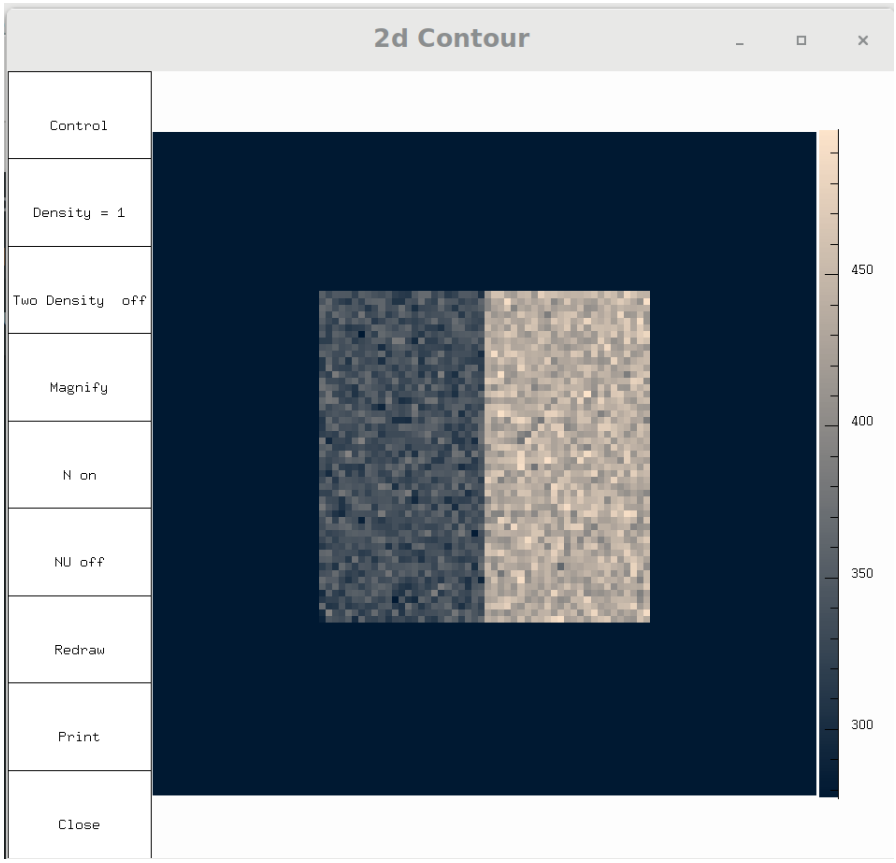

Figure 3: A box produced with static walls with one vertical dynamic wall in the center. The density on the right is 15*9 and the density on the right is 10*9. The wall is being held at position 50

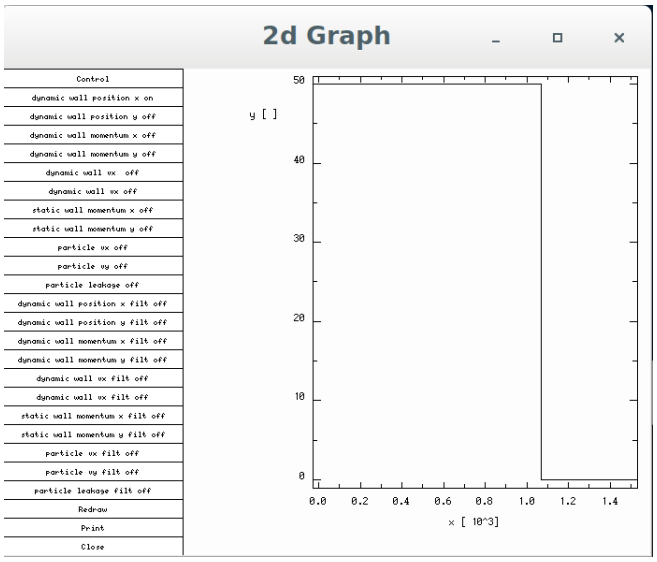

Figure 4: Graph of the dynamic wall x position

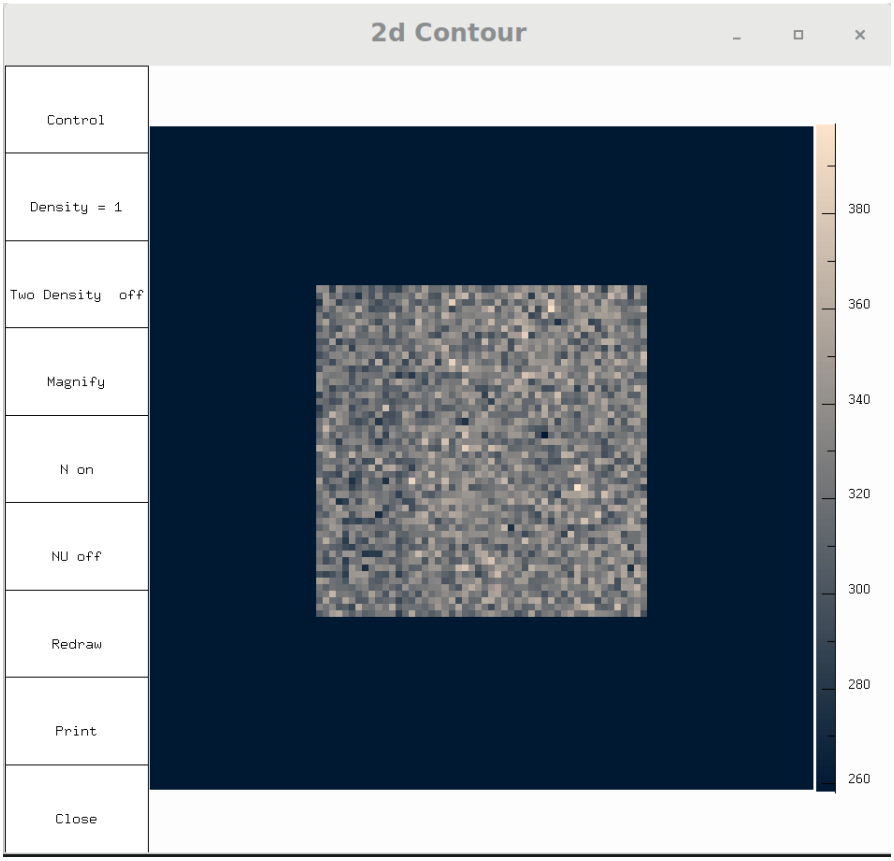

Figure 5: The force keeping the dynamic wall is removed and the wall moved towards equilibrium, but it passes the theoretical equilibrium point: 45. This error is due to leakage from rest particles

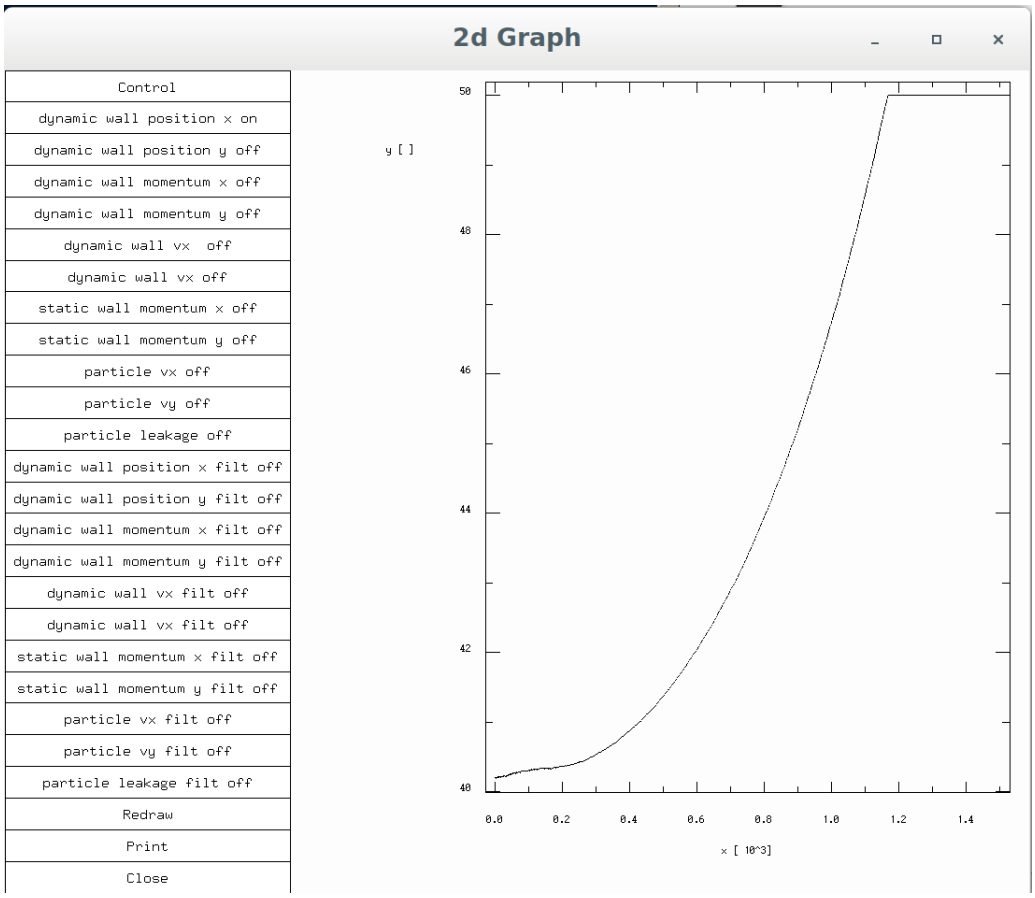

Figure 6: Graph of the walls x position as it moved towards equilibrium overt time

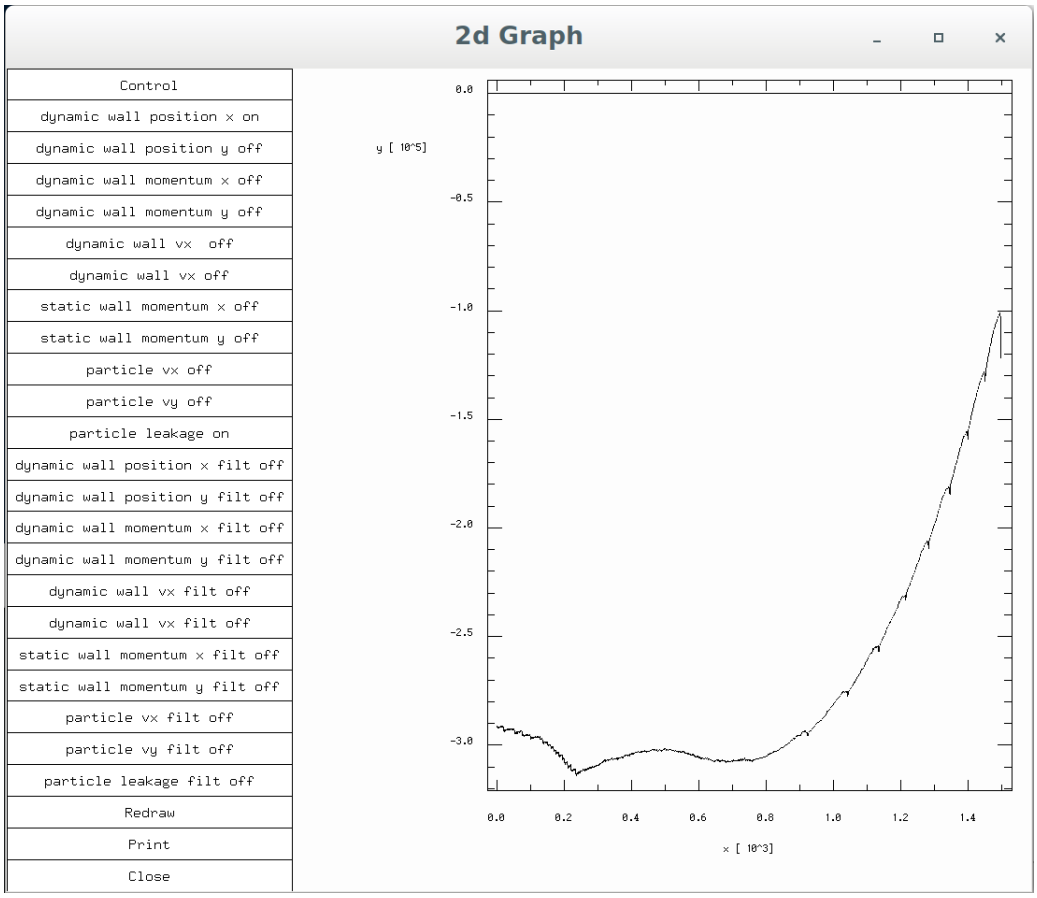

Figure 7: The particles leaking through the dynamic wall as the wall moved. Note how the leakage is proportional to the walls velocity.

3.3 Approach 2

Approach 2 is very similar to the first approach, but the primary focus was to ensure that all the particles would be removed from the lattice site by the time the wall would exit to the next lattice site. The wall has a real number for its position whereas the lattice grid is integer, so the wall will be transitioning between lattice sites. In more detail, the probability that $pr *$ particle density number of particles will be moved is defined by

$$pr = \frac{\text{Wall Vx}}{1 - (\text{real(Wall x) - int(Wall x)})}$$

. As the wall becomes closer and closer to changing lattice sites, the probability that all the particles will be moved to the next lattice site will converge to

one. This is only for a case where a wall is moving to the right. This can be reproduced for a left moving wall, but was not in this program for simplicity.

Code added for determining the flow of particles as the wall moves.

```
int tmp = n[x_v_b][y_v_b][v];
int iwp = dynamic_wall_position_x;//integer wall position
int flow = 0;//particles to be moved

double pr = dynamic_wall_vx/(1-(dynamic_wall_position_x - iwp));
//probability that p*pr particles will be moved
                if(rand()%1000 <= 1000*pr){
                        flow = pr*n[x_b][y_b][v];
                        n[x_b+1][y_b][v] +=flow;
                        n[x_b][y_b][v] -=flow;
                }
```

3.4 Results

This approach had a similar amount of leaking particles as the first approach.

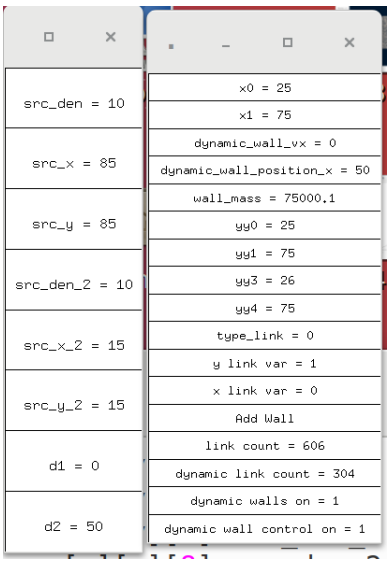

Figure 8: Settings for the simulation

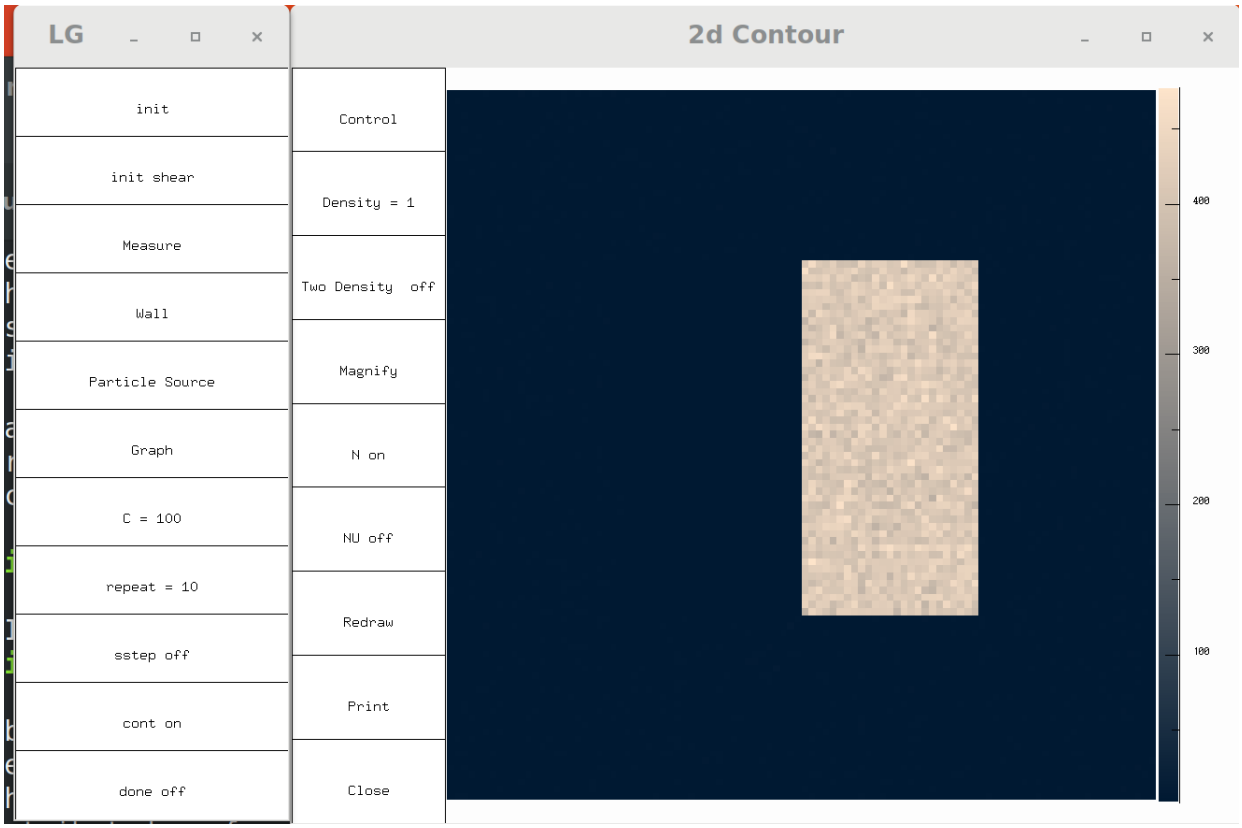

Figure 9: A box produced with static walls with one vertical dynamic wall in the center. The density on the right is 50*9 and the density on the right is 0. The wall is being held at position 50. This experiment is being done to test if the wall will leak any particles.

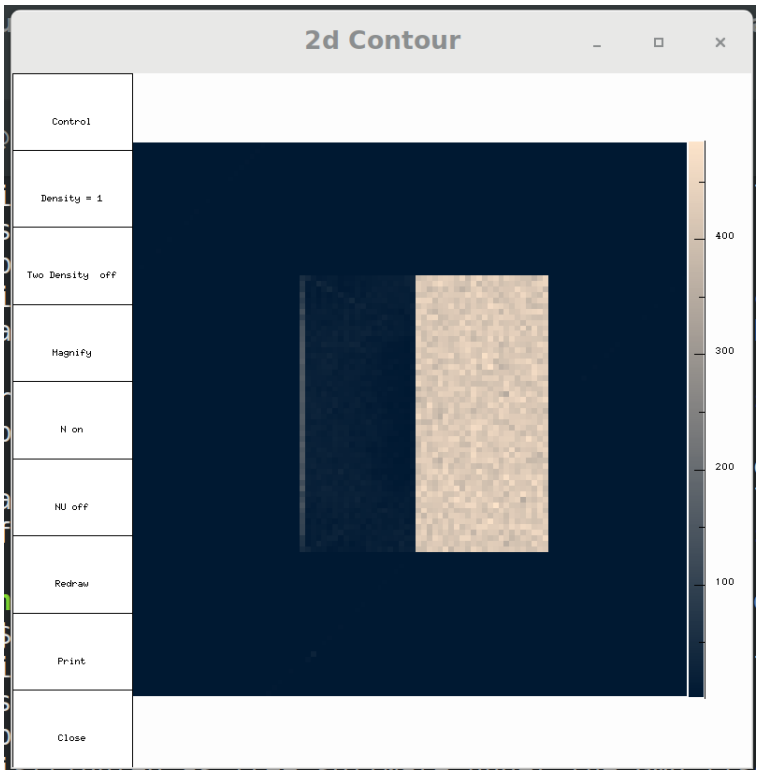

Figure 10: The wall begins to move with velocity of 0.02 in the positive x direction. Notice how a notable wave of particles are being leaked when the wall changes lattice sites.

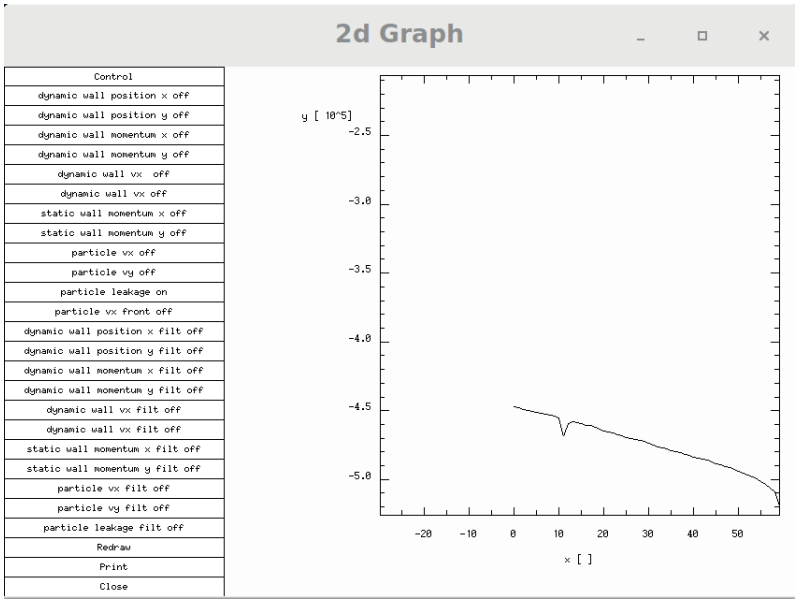

Figure 11: Graph of the particles being leaked by the wall over time

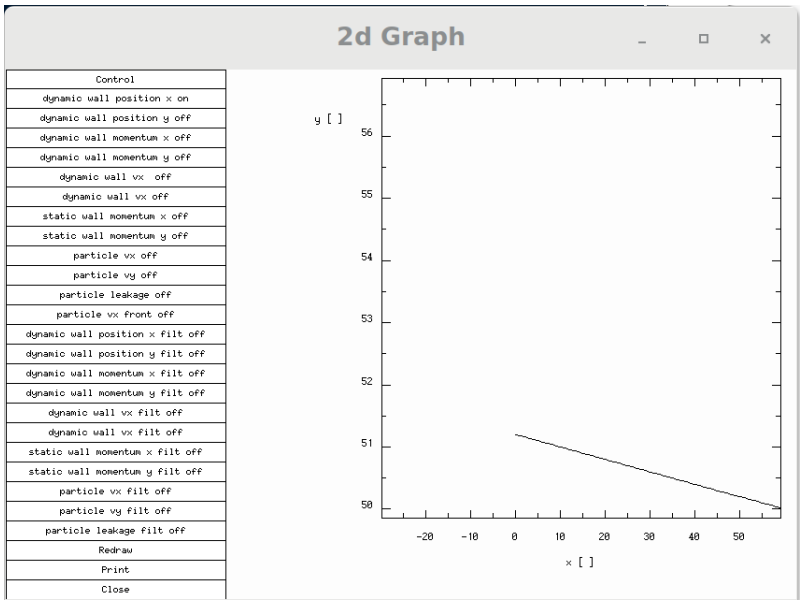

Figure 12: Graph of the walls x position as it moved towards equilibrium overt time

3.5 Approach 3

This approach tried to simplify the program by restricting the motions of the wall, then in this specific case, the walls dynamics can be tested and verified if possible. Rest particles are still known to be causing issues, so to remove them, the walls will move into a region where there are no particles. This simulation will apply a force to particles confined by 2 walls with periodic boundary conditions allowing particles to flow from one end of the tube to the other. The walls will be static to begin with and dynamic later. The force will be applied by moving a random amount of particles from the lattice velocity states of [2 5 8] to [0 3 6]. To validate the dynamics of the simulation, the mean particle velocity with respect to the cross section of the tube will be measured in the simulation and compared to a theoretical value derived from the Navier Stokes equation for fluid dynamics.

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_i)}{\partial x_i} = \nabla(\rho) + \upsilon * \nabla(\nabla(U) + (\nabla(U)T)) \tag{1}$$

The partial for ρ and ρu_i can be set to zero. This gives us:

$$0 = \nabla(\rho) + \upsilon * \nabla(\nabla(U) + (\nabla(U)T)) \tag{2}$$

$$\nabla(\rho) = F$$

$$0 = F + \upsilon * \nabla(\nabla(U_x)) \tag{3}$$

Solving the differential equation for U_x (mean velocity) above gives us:

$$U_x = \frac{F}{2 * \upsilon} * (x(x - L)) \tag{4}$$

Where L is the length of the tube in Lattice sites.

For C (Particle Collisions) the viscosity is:

$$\upsilon = \frac{1}{6} \tag{5}$$

This is the theoretical curve for the mean x velocity in the tube

$$U_x = \frac{F * 6}{2} * (x(x - L)) \tag{6}$$

The expected value of particles being moved by a set force is related to the equation below.

$$< \text{particles moved} >= force * \rho \tag{7}$$

Code for applying a force to the particles. The variable "particle flip w" is the force applied.

```
void moveParticles(){
for(int x = 0;x<xdim;x++){
        for(int y = 0;y<ydim;y++){
                int flip_parts = particle_flip_w*((double)rand()/RAND_MAX)*n[x][y][5];
                n[x][y][3] += flip_parts;
                n[x][y][5] -= flip_parts;
                flip_parts = particle_flip_w*((double)rand()/(double)RAND_MAX)*n[x][y][8];
                n[x][y][6] += flip_parts;
                n[x][y][8] -= flip_parts;
                flip_parts = particle_flip_w*((double)rand()/(double)RAND_MAX)*n[x][y][2];
                n[x][y][0] += flip_parts;
                n[x][y][2] -= flip_parts;
                }
        }
}
```

Code for calculating the theoretical curve and measuring the mean x velocity of the particles

```
void measure_function(){
particle_vx = 0;
particle_vy = 0;
int total_particles = 0;
for(int i = 0; i < YDIM -1;i++){
        total_particles = 0;
        measure_particle_velocity_front[i] = 0;
        for(int x0 = 0;x0<xdim;x0++){
                measure_particle_velocity_front[i] += n[x0][i][5]+n[x0][i][2]+n[x0][i][8] -
                n[x0][i][3]-n[x0][i][0]-n[x0][i][6];
                         for(int v = 0;v<9;v++) total_particles += n[x0][i][v];
                        }
                //find average particle velocity
                if(total_particles > 0){
                        measure_particle_velocity_front[i] =
                        (double(measure_particle_velocity_front[i]/total_particles);//+shif
                        theoretical_particle_vx_front[i] =
                        (double(particle_flip_w*6.0/9.)*(i-25)*(i-75);

                        }
                else{
                        measure_particle_velocity_front[i] = 0;
                        theoretical_particle_vx_front[i] = 0;
```

```
                }

            measure_particle_force_front[i] = (measure_particle_velocity_front[i]
             measure_particle_velocity_front_last[i]);
            measure_particle_velocity_front_last[i] = measure_particle_force_front[i];
            }
    }
```

3.6 Results

There is a very large difference in the two mean velocity in the x directions, but the measured mean x velocity does show a parabolic profile like the theory suggests. Additionally, there were no simulations with dynamic walls because the simulation needs to work first with static walls, before moving on to dynamic walls.

Figure 13: Simulation settings

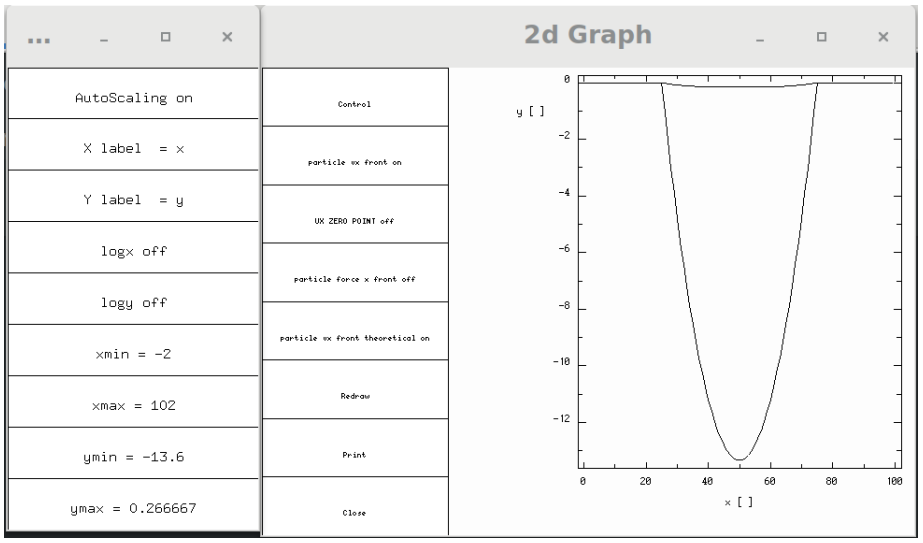

Figure 14: Theoretical and measured mean particle x velocity

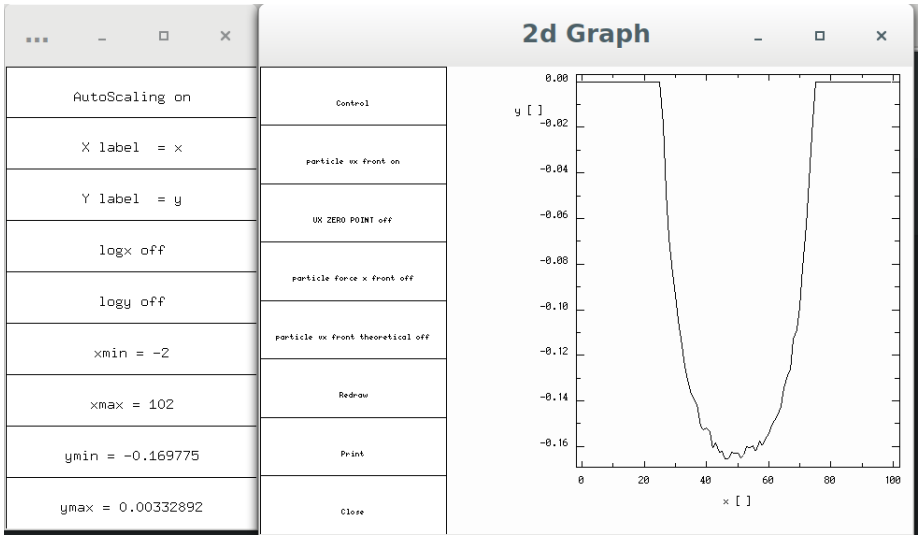

Figure 15: Measured mean particle velocity x

4 Conclusion

Although there is significant error for approaches 1 and 2 because there are on the order of a couple hundred particles leaking per iteration in a simulation that has around total particles the walls do appear to work reasonably for short

periods of time i.e. several hundred iterations. If the leaking of the particles problem was resolved, work could continue on creating complex shapes out of multiple walls and see how they interact with the simulated gas. this would work The 3rd approach was also significantly off by roughly a factor of 50, but it might work if the number of particles flipped was determined by its integer component and its probability of occurring was related to the decimal component. Future progress on this topic should try to narrow its scope to try and lower complexity so the nature of the problem of the problem can become more clear.

A Code

A.1 Code for Approach 1

```
// Lattice gas code in 1d.
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>
#include <mygraph.h>
#include <string.h>

#define xdim 100
#define ydim 100
#define V 9        //number of velocities
#define MeasMax 3000
int XDIM=xdim,YDIM=ydim;
int C=100;
double N[xdim][ydim],NU[xdim][ydim][2];
int Nreq=0,NUreq=0;
int n[xdim][ydim][V];

// Boundary variables
#define LINKMAX 10000
#define LINKMAX_DYNAMIC 10000



double dt = 1.0;

int x_link_var = -1;
int y_link_var = 0;

//variables for measuring tube momentum
```

```
//particle source parameters
int src_x = 85,src_y = 85,src_len = 5,src_den = 10;
int src_x_2 = 15,src_y_2 = 15,src_len_2 = 5,src_den_2 = 10;
int var1 =10;
//variables for measuring values
//particle velocity
int particle_vx = 0,particle_vy ,particle_leakage = 0;
double measure_particle_vx[MeasMax],measure_particle_vy[MeasMax];
double measure_particle_vx_filt[MeasMax],measure_particle_vy_filt[MeasMax];
int vx_m = 0,vy_m = 0;
int d1 = 40,d2 = 50;



//link variables
int linkcount_dynamic = 0,links_dynamic[LINKMAX_DYNAMIC][3];//link x y and v(for
double links_dynamic_velocity[LINKMAX_DYNAMIC][2];//link velocities
double wall_mass = 75000.1;
int yy3 = 26,yy4 = 75,dynamic_walls_on = 1,dynamic_wall_control_on = 0;
double flow = 0;
//dynamic wall
// position
//double x_shift = 50;
double dynamic_wall_position_x = 0,dynamic_wall_position_y = 0;
//velocity
double dynamic_wall_vx = 0,dynamic_wall_vy = 0;
//momentum
int dynamic_wall_momentum_x = 0,dynamic_wall_momentum_y = 0;
//dynamic wall position for graphing
double measure_dynamic_wall_position_x[MeasMax],measure_dynamic_wall_position_y[
//filtered
double measure_dynamic_wall_position_x[MeasMax],measure_dynamic_wall_position_y[
//
double measure_dynamic_wall_position_x_filt[MeasMax],measure_dynamic_wall_posit
//dynamic wall velocity
double measure_dynamic_wall_vx[MeasMax],measure_dynamic_wall_vy[MeasMax];
//filtered
double measure_dynamic_wall_vx_filt[MeasMax],measure_dynamic_wall_vy_filt[MeasMa
//dynamic wall momentum
double measure_dynamic_wall_momentum_x[MeasMax],measure_dynamic_wall_momentum_y[
//filtered
double measure_dynamic_wall_momentum_x_filt[MeasMax],measure_dynamic_wall_momen

//
double measure_particle_leakage[MeasMax];
```

```
double measure_particle_leakage_filt[MeasMax];

//static wall variables
int linkcount=0,links[LINKMAX][3];

//points for drawing walls
int x0=25,x1=75,yy0 = 25,yy1 = 75;

//static wall momentum
double static_wall_momentum_x = 0,static_wall_momentum_y = 0;
double measure_static_wall_momentum_x[MeasMax],measure_static_wall_momentum_y[Me
//filtered data
double measure_static_wall_momentum_x_filt[MeasMax],measure_static_wall_momentu

int MeasLen = MeasMax/2;
int range_val = 20;
int val[] = {0,0};
int filt_data[] ={0,0,0,0,0,0,0,0,0,0,0}; //dynamic walls -> x,y,px,py,vx,vy
static walls -> px py particles -> vx vy leakage


//added a running average to smooth out the graphs
void average(int range){
        //val[0] = 0;
        //val[1] = 0;
        //clear previous filtered data
        for(int i = 0; i < 11;i++){
                filt_data[i] = 0;
                }
        //update with new data
        for(int i = 0; i < range;i++){
                filt_data[0] += measure_dynamic_wall_position_x[i];
                filt_data[1] += measure_dynamic_wall_position_y[i];
                filt_data[2] += measure_dynamic_wall_momentum_x[i];
                filt_data[3] += measure_dynamic_wall_momentum_y[i];
                filt_data[4] += measure_dynamic_wall_vx[i];
                filt_data[5] += measure_dynamic_wall_vy[i];
                filt_data[6] += measure_static_wall_momentum_x[i];
                filt_data[7] += measure_static_wall_momentum_y[i];
                filt_data[8] += measure_particle_vx[i];
                filt_data[9] += measure_particle_vy[i];
                filt_data[10] += measure_particle_leakage[i];
                }
        //average values
        for(int i = 0; i < 11;i++){
                filt_data[i] = filt_data[i]/range;
```

```
                }
}

void measure_function(){
        particle_vx = 0;
        particle_vy = 0;
        int particle_wall = 0;
        int int_wall_pos = dynamic_wall_position_x;
        int particles_left = 0;
        for(int i = 0; i < YDIM -1;i++){
                particle_vx += n[50][i][2]+n[50][i][5]+n[50][i][8]-n[50][i][0]-n
                particle_vy += n[i][50][0]+n[i][50][1]+n[i][50][2]-n[i][50][6]-n
                }
        particle_vx = particle_vx/100.0;
        particle_vy = particle_vy/100.0;



        //measure the number of particles leaking

          particle_leakage = 0;

                for (int y=yy3; y<yy4; y++){

                        for(int v = 0; v < 9; v++)
                                {
                                particle_wall += n[int_wall_pos][y][v];
                                }
                }

          for (int x=x0; x<dynamic_wall_position_x; x++){

                for (int y=yy3; y<yy4; y++){

                        for(int v = 0; v < 9; v++)
                                {
                                particle_leakage += n[x][y][v];
                                }
                }
        }

        particle_leakage = particle_leakage;////(dynamic_wall_position_x-x0)*(yy

        for (int x=(1+dynamic_wall_position_x); x<x1; x++){
                for (int y=yy3; y<yy4; y++){
                        for(int v = 0; v < 9; v++)
```

```
                                {
                                particles_left += n[x][y][v];
                                }
                        }
                }

                particles_left = particles_left + particle_wall*(1 - (dynamic_wall_posit

                particle_leakage = particle_leakage + particle_wall*(dynamic_wall_positi

                particle_leakage = particle_leakage - particles_left;
}


//take measurements for plotting data
void Measure(){
  //store dynamic wall data
  //position
  measure_function();

  memmove(&measure_dynamic_wall_position_x[1],&measure_dynamic_wall_position_x[0
  measure_dynamic_wall_position_x[0]=dynamic_wall_position_x;
  memmove(&measure_dynamic_wall_position_y[1],&measure_dynamic_wall_position_y[0
  measure_dynamic_wall_position_y[0]=dynamic_wall_position_y;
  //momentum
  memmove(&measure_dynamic_wall_momentum_x[1],&measure_dynamic_wall_momentum_x[0
  measure_dynamic_wall_momentum_x[0]=dynamic_wall_momentum_x;
  memmove(&measure_dynamic_wall_momentum_y[1],&measure_dynamic_wall_momentum_y[0
  measure_dynamic_wall_momentum_y[0]=dynamic_wall_momentum_y;
  //velocity
  memmove(&measure_dynamic_wall_vx[1],&measure_dynamic_wall_vx[0],(MeasMax-1)*si
  measure_dynamic_wall_vx[0]=dynamic_wall_vx;
  memmove(&measure_dynamic_wall_vy[1],&measure_dynamic_wall_vy[0],(MeasMax-1)*si
  measure_dynamic_wall_vy[0]=dynamic_wall_vy;

  //store static wall data
  memmove(&measure_static_wall_momentum_x[1],&measure_static_wall_momentum_x[0],
  measure_static_wall_momentum_x[0]=static_wall_momentum_x;
  memmove(&measure_static_wall_momentum_y[1],&measure_static_wall_momentum_y[0],
  measure_static_wall_momentum_y[0]=static_wall_momentum_y;

  //store particle data
  memmove(&measure_particle_vx[1],&measure_particle_vx[0],(MeasMax-1)*sizeof(int
  measure_particle_vx[0]=particle_vx;
  memmove(&measure_particle_vy[1],&measure_particle_vy[0],(MeasMax-1)*sizeof(int
  measure_particle_vy[0]=particle_vy;
```

```
  memmove(&measure_particle_leakage[1],&measure_particle_leakage[0],(MeasMax-1)*
  measure_particle_leakage[0]=particle_leakage;

  //filter data by taking the average of the data points over the range_val
  average(range_val);

   //position
  memmove(&measure_dynamic_wall_position_x_filt[1],&measure_dynamic_wall_positi
  measure_dynamic_wall_position_x_filt[0]=filt_data[0];
  memmove(&measure_dynamic_wall_position_y_filt[1],&measure_dynamic_wall_positi
  measure_dynamic_wall_position_y_filt[0]=filt_data[1];
  //momentum
  memmove(&measure_dynamic_wall_momentum_x_filt[1],&measure_dynamic_wall_moment
  measure_dynamic_wall_momentum_x_filt[0]=filt_data[2];
  memmove(&measure_dynamic_wall_momentum_y_filt[1],&measure_dynamic_wall_moment
  measure_dynamic_wall_momentum_y_filt[0]=filt_data[3];
  //velocity
  memmove(&measure_dynamic_wall_vx_filt[1],&measure_dynamic_wall_vx_filt[0],(Mea
  measure_dynamic_wall_vx_filt[0]=filt_data[4];
  memmove(&measure_dynamic_wall_vy_filt[1],&measure_dynamic_wall_vy_filt[0],(Mea
  measure_dynamic_wall_vy_filt[0]=filt_data[5];

  //store static wall data
  memmove(&measure_static_wall_momentum_x_filt[1],&measure_static_wall_momentum
  measure_static_wall_momentum_x_filt[0]=filt_data[6];
  memmove(&measure_static_wall_momentum_y_filt[1],&measure_static_wall_momentum
  measure_static_wall_momentum_y_filt[0]=filt_data[7];

  memmove(&measure_particle_vx_filt[1],&measure_particle_vx_filt[0],(MeasMax-1)*
  measure_particle_vx_filt[0]=filt_data[8];
  memmove(&measure_particle_vy_filt[1],&measure_particle_vy_filt[0],(MeasMax-1)*
  measure_particle_vy_filt[0]=filt_data[9];

  memmove(&measure_particle_leakage_filt[1],&measure_particle_leakage_filt[0],(M
  measure_particle_leakage_filt[0]=particle_leakage;//filt_data[10];
}
//re draws walls
void FindLink_Dynamic(){
        int tmp_x_shift = 0;

                linkcount_dynamic = 0;
                //draw vertical walls
                for(int y = yy3; y<yy4+1;y++){

                        tmp_x_shift = dynamic_wall_position_x;
```

```
                tmp_x_shift = ((tmp_x_shift%XDIM)+XDIM)%XDIM;
                links_dynamic[linkcount_dynamic][0] = tmp_x_shift; //x-p
                links_dynamic[linkcount_dynamic][1] = y;
                links_dynamic[linkcount_dynamic][2] = 0;
                links_dynamic_velocity[linkcount_dynamic][0] = dynamic_v

                links_dynamic_velocity[linkcount_dynamic][1] = 0;//set y
                linkcount_dynamic++;
                links_dynamic[linkcount_dynamic][0] = tmp_x_shift; //x-p
                links_dynamic[linkcount_dynamic][1] = y;
                links_dynamic[linkcount_dynamic][2] = 3;
                links_dynamic_velocity[linkcount_dynamic][0] = dynamic_v
                links_dynamic_velocity[linkcount_dynamic][1] = 0;//set y
                linkcount_dynamic++;
                links_dynamic[linkcount_dynamic][0] = tmp_x_shift; //x-p
                links_dynamic[linkcount_dynamic][1] = y;
                links_dynamic[linkcount_dynamic][2] = 6;
                links_dynamic_velocity[linkcount_dynamic][0] = dynamic_v
                links_dynamic_velocity[linkcount_dynamic][1] = 0;//set y
                linkcount_dynamic++;
                }

                links_dynamic[linkcount_dynamic][0] = tmp_x_shift; //x-p
                links_dynamic[linkcount_dynamic][1] = yy1;
                links_dynamic[linkcount_dynamic][2] = 0;
                links_dynamic_velocity[linkcount_dynamic][0] = dynamic_v
                links_dynamic_velocity[linkcount_dynamic][1] = 0;//set y
                linkcount_dynamic++;

                links_dynamic[linkcount_dynamic][0] = tmp_x_shift+x_link
                links_dynamic[linkcount_dynamic][1] = yy0+y_link_var;
                links_dynamic[linkcount_dynamic][2] = 2;
                links_dynamic_velocity[linkcount_dynamic][0] = dynamic_v
                links_dynamic_velocity[linkcount_dynamic][1] = 0;//set y
                linkcount_dynamic++;

        dynamic_wall_position_x += dynamic_wall_vx*dt;
}

void FindLink(){
        //2 links added to prevent leaking on the x0,yy2 and x2,yy0 squares
        links[linkcount][0] = x0; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x1; //x-position
```

```
        links[linkcount][1] = yy0;
        links[linkcount][2] = 0;
        linkcount++;
//horizontal walls
for (int x=x0; x<x1+1; x++){
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 1;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 2;
        linkcount++;
}
  for (int x=x0; x<x1+1; x++){
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 1;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 2;
        linkcount++;
}

//vertical walls
for (int y=yy0; y<yy1+1; y++){
        links[linkcount][0] = x0; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x0; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 3;
        linkcount++;
        links[linkcount][0] = x0; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 6;
```

```
        linkcount++;
    }

    for (int y=yy0; y<yy1+1; y++){
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 3;
        linkcount++;
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 6;
        linkcount++;
    }
}

void bounceback(){
    dynamic_wall_momentum_x = 0;
    dynamic_wall_momentum_y = 0;
    static_wall_momentum_x = 0;
    static_wall_momentum_y = 0;

    for (int lc=0; lc<linkcount; lc++){
      //quantity of partices

      int x=links[lc][0];
      int y=links[lc][1];
      //velocity
      int v=links[lc][2];
      int vx=v%3-1;
      int vy=1-v/3;
      int tmp= n[x+vx][y+vy][v];
      //summing all momemtums
      static_wall_momentum_x += -2*vx*(n[x][y][8-v]-tmp);
      static_wall_momentum_y += -2*vy*(n[x][y][8-v]-tmp);
      //swapping the particles trying to enter and leave to have the effect of a u
      n[x+vx][y+vy][v]= n[x][y][8-v];
      n[x][y][8-v]=tmp;
    }
//dynamic walls
if(dynamic_walls_on == 1){
 for (int lc=0; lc<linkcount_dynamic; lc++){
      //quantity of partices
```

```
    int x=links_dynamic[lc][0];
    int y=links_dynamic[lc][1];
    //particle velocity
    int v=links_dynamic[lc][2];
    int vx=v%3-1;
    int vy=1-v/3;
    int tmp_0 = n[(((vx+x)%XDIM)+XDIM)%XDIM][((y+vy)%YDIM+YDIM)%YDIM][8-v];
    int tmp = n[(((vx+x)%XDIM)+XDIM)%XDIM][((y+vy)%YDIM+YDIM)%YDIM][v];
        //find the smaller value to be the max for the random function to avoid
    //determining values for forward and backward particle flow
        int max_random = 1;
        if(tmp > tmp_0){
                max_random = tmp_0;
                }
        else{
                max_random = tmp;
                }
        if(max_random > 0){
                flow = (rand()%max_random)*dynamic_wall_vx;
                }
        else{
                flow = 0;
                }

    //summing all momemtums
    dynamic_wall_momentum_x += -2*vx*(n[x][y][8-v]-tmp+flow);
    dynamic_wall_momentum_y += -2*vy*(n[x][y][8-v]-tmp);
    //swapping the particles trying to enter and leave to have the effect of a w
    n[(((vx+x)%XDIM)+XDIM)%XDIM][((y+vy)%YDIM+YDIM)%YDIM][v] = n[(((x)%XDIM)+XDIM
    n[(((x)%XDIM)+XDIM)%XDIM][((y)%YDIM+YDIM)%YDIM][8-v] = tmp + flow;
        printf("%f \n",flow);
  }

        if(dynamic_wall_control_on == 0){
                dynamic_wall_vx = dynamic_wall_momentum_x/wall_mass;
                dynamic_wall_vy = dynamic_wall_momentum_y/wall_mass;
        }
}

}

void setrho(){
  for (int x=src_x; x<src_x+src_len; x++){
    int y=src_y;
    n[x][y][0]=src_den;
    n[x][y][1]=src_den;
```

```
      n[x][y][2]=src_den;
      n[x][y][3]=src_den;
      n[x][y][4]=src_den;
      n[x][y][5]=src_den;
      n[x][y][6]=src_den;
      n[x][y][7]=src_den;
      n[x][y][8]=src_den;
    }
    for (int x=src_x_2; x<src_x_2+src_len_2; x++){
      int y=src_y_2;
      n[x][y][0]=src_den_2;
      n[x][y][1]=src_den_2;
      n[x][y][2]=src_den_2;
      n[x][y][3]=src_den_2;
      n[x][y][4]=src_den_2;
      n[x][y][5]=src_den_2;
      n[x][y][6]=src_den_2;
      n[x][y][7]=src_den_2;
      n[x][y][8]=src_den_2;
    }
}

void init(){
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){

      dynamic_wall_position_x = 50;
      if(x > 25 && x < 50 && y < 75 && y > 25){
        n[x][y][0]=d1;
        n[x][y][1]=d1;
        n[x][y][2]=d1;
        n[x][y][3]=d1;
        n[x][y][4]=d1;
        n[x][y][5]=d1;
        n[x][y][6]=d1;
        n[x][y][7]=d1;
        n[x][y][8]=d1;
        }
      else if(x > 50 && x < 75 && y < 75 && y > 25){
        n[x][y][0]=d2;
        n[x][y][1]=d2;
        n[x][y][2]=d2;
        n[x][y][3]=d2;
        n[x][y][4]=d2;
        n[x][y][5]=d2;
        n[x][y][6]=d2;
```

```
        n[x][y][7]=d2;
        n[x][y][8]=d2;
        }
        else if(x <= 25 || x >= 75 || y >= 75 || y <= 25){
        n[x][y][0]=0;
        n[x][y][1]=0;
        n[x][y][2]=0;
        n[x][y][3]=0;
        n[x][y][4]=0;
        n[x][y][5]=0;
        n[x][y][6]=0;
        n[x][y][7]=0;
        n[x][y][8]=0;
        }
    }
  }
}
void initShear(){
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      if (x<xdim/2){
        n[x][y][0]=1;
        n[x][y][1]=2;
        n[x][y][2]=3;
        n[x][y][3]=4;
        n[x][y][4]=5;
        n[x][y][5]=6;
        n[x][y][6]=7;
        n[x][y][7]=8;
        n[x][y][8]=9;
      }
      else {
        n[x][y][0]=8;
        n[x][y][1]=7;
        n[x][y][2]=6;
        n[x][y][3]=5;
        n[x][y][4]=4;
        n[x][y][5]=3;
        n[x][y][6]=2;
        n[x][y][7]=0;
        n[x][y][8]=0;
        }
    }
  }
}
```

```
void iterate(){
  int NI=0;
  int rate=RAND_MAX/8.;
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      NI=0;
      for (int v=0; v<V; v++) NI+=n[x][y][v];
       // first implementation: random collisions
      if (NI>0){
        for (int c=0; c<C; c++){
          //       int r1=1.0*NI*rand()/RAND_MAX;
          //       int r2=1.0*NI*rand()/RAND_MAX;
          int r1=rand()%NI;
          int r2=rand()%NI;
          if (r1!=r2){
            int v1=0;
            for (; r1>=n[x][y][v1]; v1++)
              r1-=n[x][y][v1];
            int v2=0;
            for (; r2>=n[x][y][v2]; v2++)
              r2-=n[x][y][v2];
            // now we picked two particles with velocity v1 and v2.
            int v1x=v1%3-1;
            int v1y=v1/3-1;
            int v2x=v2%3-1;
            int v2y=v2/3-1;
            // x-collision
            if ((v1x==-1)&&(v2x==1)){
              v1x=0;
              v2x=0;
            }
            else if ((v1x==1)&&(v2x==-1)){
              v1x=0;
              v2x=0;
            }
            else if ((v1x==0)&&(v2x==0)){
              int r=rand();
              if (r<rate){
                if (r<rate/2){
                  v1x=-1;
                  v2x=1;
                }
                else {
                  v1x=1;
                  v2x=-1;
                }
```

```
            }
          }
          else {
            int tmp=v1x;
            v1x=v2x;
            v2x=tmp;
          }
          // y-collision
          if ((v1y==-1)&&(v2y==1)){
            v1y=0;
            v2y=0;
          }
          else if ((v1y==1)&&(v2y==-1)){
            v1y=0;
            v2y=0;
          }
          else if ((v1y==0)&&(v2y==0)){
            int r=rand();
            if (r<rate){
              if (r<rate/2){
                v1y=-1;
                v2y=1;
              }
              else {
                v1y=1;
                v2y=-1;
              }
            }
          }
          else {
            int tmp=v1y;
            v1y=v2y;
            v2y=tmp;
          }
          n[x][y][v1]--;
          n[x][y][v2]--;
          v1=(v1y+1)*3+(v1x+1);
          v2=(v2y+1)*3+(v2x+1);
          n[x][y][v1]++;
          n[x][y][v2]++;
        }
      }
    }
  }
}
// move the particles in x-direction
```

```
  {
    int ntmp[ydim];
    for (int c=0; c<3; c++){
      for (int y=0; y<ydim; y++) ntmp[y]=n[xdim-1][y][c*3+2];
      for (int x=xdim-1; x>0; x--){
        for (int y=0; y<ydim; y++)
          n[x][y][c*3+2]=n[x-1][y][c*3+2];
      }
      for (int y=0; y<ydim; y++) n[0][y][c*3+2]=ntmp[y];
    }
    for (int c=0; c<3; c++){
      for (int y=0; y<ydim; y++) ntmp[y]=n[0][y][c*3];
      for (int x=0; x<xdim-1; x++){
        for (int y=0; y<ydim; y++)
          n[x][y][c*3]=n[x+1][y][c*3];
      }
      for (int y=0; y<ydim; y++) n[xdim-1][y][c*3]=ntmp[y];
    }
  }
    // move the particles in y-direction
  {
    int ntmp[xdim];
    for (int c=0; c<3; c++){
      for (int x=0; x<xdim; x++) ntmp[x]=n[x][ydim-1][c];
      for (int y=ydim-1; y>0; y--){
        for (int x=0; x<xdim; x++)
          n[x][y][c]=n[x][y-1][c];
      }
      for (int x=0; x<xdim; x++) n[x][0][c]=ntmp[x];
    }
    for (int c=0; c<3; c++){
      for (int x=0; x<xdim; x++) ntmp[x]=n[x][0][c+6];
      for (int y=0; y<ydim-1; y++){
        for (int x=0; x<xdim; x++)
          n[x][y][c+6]=n[x][y+1][c+6];
      }
      for (int x=0; x<xdim; x++) n[x][ydim-1][c+6]=ntmp[x];
    }
  }
  if(dynamic_walls_on == 1){
        FindLink_Dynamic();
        }
  bounceback();
}

void getdata(){
```

```
  if (NUreq||Nreq){
    NUreq=0;
    Nreq=0;
    for (int x=0; x<xdim; x++)
      for (int y=0; y<ydim; y++){
        N[x][y]=0;
        NU[x][y][0]=0;
        NU[x][y][1]=0;
        for (int v=0; v<V; v++){
          N[x][y]+=n[x][y][v];
          NU[x][y][0]+=n[x][y][v]*(v%3-1);
          NU[x][y][1]+=n[x][y][v]*(1-v/3);
        }
      }
  }
}

void debug(){
  int NUX=0,NUY=0,NN=0;
  for (int x=0; x<xdim; x++)
    for (int y=0; y<ydim; y++){
      for (int v=0; v<V; v++){
        NN+=n[x][y][v];
        NUX+=n[x][y][v]*(v%3-1);
        NUY+=n[x][y][v]*(1-v/3);
      }
    }
  printf("N=%i; NUX=%i; NUY=%i;\n",NN,NUX,NUY);
}

void main(){
  int done=0,sstep=0,cont=0,repeat=10;
  init();
  Measure();
  DefineGraphNxN_R("N",&N[0][0],&XDIM,&YDIM,&Nreq);
  DefineGraphNxN_RxR("NU",&NU[0][0][0],&XDIM,&YDIM,&NUreq);
  DefineGraphN_R("dynamic wall position x",&measure_dynamic_wall_position_x[0],&
  DefineGraphN_R("dynamic wall position y",&measure_dynamic_wall_position_y[0],&
  DefineGraphN_R("dynamic wall momentum x",&measure_dynamic_wall_momentum_x[0],&
  DefineGraphN_R("dynamic wall momentum y",&measure_dynamic_wall_momentum_y[0],&
  DefineGraphN_R("dynamic wall vx ",&measure_dynamic_wall_vx[0],&MeasLen,NULL);
  DefineGraphN_R("dynamic wall vx",&measure_dynamic_wall_vy[0],&MeasLen,NULL);
  DefineGraphN_R("static wall momentum x",&measure_static_wall_momentum_x[0],&Me
  DefineGraphN_R("static wall momentum y",&measure_static_wall_momentum_y[0],&Me
  DefineGraphN_R("particle vx",&measure_particle_vx[0],&MeasLen,NULL);
  DefineGraphN_R("particle vy",&measure_particle_vy[0],&MeasLen,NULL);
```

```
DefineGraphN_R("particle_leakage",&measure_particle_leakage[0],&MeasLen,NULL);
//filtered graphs
DefineGraphN_R("dynamic_wall_position_x_filt",&measure_dynamic_wall_position_
DefineGraphN_R("dynamic_wall_position_y_filt",&measure_dynamic_wall_position_
DefineGraphN_R("dynamic_wall_momentum_x_filt",&measure_dynamic_wall_momentum_
DefineGraphN_R("dynamic_wall_momentum_y_filt",&measure_dynamic_wall_momentum_
DefineGraphN_R("dynamic_wall_vx_filt",&measure_dynamic_wall_vx_filt[0],&MeasLe
DefineGraphN_R("dynamic_wall_vx_filt",&measure_dynamic_wall_vy_filt[0],&MeasLe
DefineGraphN_R("static_wall_momentum_x_filt",&measure_static_wall_momentum_x_
DefineGraphN_R("static_wall_momentum_y_filt",&measure_static_wall_momentum_y_
DefineGraphN_R("particle_vx_filt",&measure_particle_vx_filt[0],&MeasLen,NULL);
DefineGraphN_R("particle_vy_filt",&measure_particle_vy_filt[0],&MeasLen,NULL);
DefineGraphN_R("particle_leakage_filt",&measure_particle_leakage_filt[0],&Meas
StartMenu("LG",1);
DefineFunction("init",init);
DefineFunction("init_shear",initShear);
StartMenu("Measure",0);
DefineInt("range_val",&range_val);
DefineGraph(curve2d_,"Measurements");
EndMenu();
StartMenu("Wall",0);
DefineInt("x0", &x0);
DefineInt("x1", &x1);
DefineDouble("dynamic_wall_vx", &dynamic_wall_vx);
DefineDouble("dynamic_wall_position_x", &dynamic_wall_position_x);
DefineDouble("wall_mass", &wall_mass);
DefineInt("yy0", &yy0);
DefineInt("yy1", &yy1);
DefineInt("yy3", &yy3);
DefineInt("yy4", &yy4);
DefineInt("y_link_var", &y_link_var);
DefineInt("x_link_var", &x_link_var);
DefineFunction("Add_Wall",FindLink);
DefineInt("link_count",&linkcount);
DefineInt("dynamic_link_count",&linkcount_dynamic);
DefineInt("dynamic_walls_on",&dynamic_walls_on);
DefineInt("dynamic_wall_control_on",&dynamic_wall_control_on);
EndMenu();
StartMenu("Particle_Source",0);
DefineInt("src_den",&src_den);
DefineInt("src_x",&src_x);
DefineInt("src_y",&src_y);
DefineInt("src_den_2",&src_den_2);
DefineInt("src_x_2",&src_x_2);
DefineInt("src_y_2",&src_y_2);
DefineInt("d1",&d1);
```

```
  DefineInt("d2",&d2);
  EndMenu();
  DefineGraph(contour2d_,"Graph");
  DefineInt("C", &C);
  DefineInt("repeat",&repeat);
  DefineBool("sstep",&sstep);
  DefineBool("cont",&cont);
  DefineBool("done",&done);
  EndMenu();


  while (!done){
    Events(1);
    getdata();
    DrawGraphs();
    if (cont || sstep){
      sstep=0;

      for (int i=0; i<repeat; i++) {
        iterate();
        Measure();
        //setrho();
      }
    } else sleep(1);
  }
}
```

A.2 Code for Approach 2

```
// Lattice gas code in 1d.
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>
#include <mygraph.h>
#include <string.h>

#define xdim 100
#define ydim 100
#define V 9       //number of velocities
#define MeasMax 3000
int XDIM=xdim,YDIM=ydim;
int C=100;
double N[xdim][ydim],NU[xdim][ydim][2];
int Nreq=0,NUreq=0;
int n[xdim][ydim][V];
```

```
// Boundary variables
#define LINKMAX 10000
#define LINKMAX_DYNAMIC 10000



double dt = 1.0;

int x_link_var = 0;
int y_link_var = 1;
int type_link = 0;

//variables for measuring tube momentum



//particle source parameters
int src_x = 85,src_y = 85,src_len = 5,src_den = 10;
int src_x_2 = 15,src_y_2 = 15,src_len_2 = 5,src_den_2 = 10;
int var1 =10;
//variables for measuring values
//particle velocity
int particle_vx = 0,particle_vy ,particle_leakage = 0;
double measure_particle_vx[MeasMax],measure_particle_vy[MeasMax];
double measure_particle_vx_filt[MeasMax],measure_particle_vy_filt[MeasMax];
int vx_m = 0,vy_m = 0;
int d1 = 0,d2 = 50;



//link variables
int linkcount_dynamic = 0,links_dynamic[2][LINKMAX_DYNAMIC][3];//link x y and v(
double links_dynamic_pool[2][LINKMAX_DYNAMIC];
double links_dynamic_velocity[LINKMAX_DYNAMIC][2];//link velocities
double wall_mass = 75000.1;
int yy3 = 26,yy4 = 75,dynamic_walls_on = 1,dynamic_wall_control_on = 1;
double flow = 0;
//dynamic wall
// position
double dynamic_wall_position_x = 25,dynamic_wall_position_y = 0;
//velocity
double dynamic_wall_vx = 0,dynamic_wall_vy = 0;
//momentum
int dynamic_wall_momentum_x = 0,dynamic_wall_momentum_y = 0;
//dynamic wall position for graphing
```

```
double measure_dynamic_wall_position_x[MeasMax], measure_dynamic_wall_position_y[
//filtered
double measure_dynamic_wall_position_x[MeasMax], measure_dynamic_wall_position_y[
//
double measure_dynamic_wall_position_x_filt[MeasMax], measure_dynamic_wall_posit
//dynamic wall velocity
double measure_dynamic_wall_vx[MeasMax], measure_dynamic_wall_vy[MeasMax];
//filtered
double measure_dynamic_wall_vx_filt[MeasMax], measure_dynamic_wall_vy_filt[MeasMa
//dynamic wall momentum
double measure_dynamic_wall_momentum_x[MeasMax], measure_dynamic_wall_momentum_y[
//filtered
double measure_dynamic_wall_momentum_x_filt[MeasMax], measure_dynamic_wall_momen

//
double measure_particle_leakage[MeasMax];
double measure_particle_leakage_filt[MeasMax];
double measure_particle_velocity_front[MeasMax];


//static wall variables
int linkcount=0,links[LINKMAX][3];

//points for drawing walls
int x0=25,x1=75,yy0 = 25,yy1 = 75;

//static wall momentum
double static_wall_momentum_x = 0,static_wall_momentum_y = 0;
double measure_static_wall_momentum_x[MeasMax], measure_static_wall_momentum_y[Me
//filtered data
double measure_static_wall_momentum_x_filt[MeasMax], measure_static_wall_momentu


int MeasLen = MeasMax/2;
int range_val = 20;
int val[] = {0,0};
int filt_data[] ={0,0,0,0,0,0,0,0,0,0,0,0}; //dynamic walls -> x,y,px,py,vx,vy
static walls -> px py particles -> vx vy leakage


//added a running average to smooth out the graphs
void average(int range){
        //clear previous filtered data
        for(int i = 0; i < 11;i++){
                filt_data[i] = 0;
                }
```

```
        //update with new data
        for(int i = 0; i < range;i++){
                filt_data[0] += measure_dynamic_wall_position_x[i];
                filt_data[1] += measure_dynamic_wall_position_y[i];
                filt_data[2] += measure_dynamic_wall_momentum_x[i];
                filt_data[3] += measure_dynamic_wall_momentum_y[i];
                filt_data[4] += measure_dynamic_wall_vx[i];
                filt_data[5] += measure_dynamic_wall_vy[i];
                filt_data[6] += measure_static_wall_momentum_x[i];
                filt_data[7] += measure_static_wall_momentum_y[i];
                filt_data[8] += measure_particle_vx[i];
                filt_data[9] += measure_particle_vy[i];
                filt_data[10] += measure_particle_leakage[i];
                }
        //average values
        for(int i = 0; i < 11;i++){
                filt_data[i] = filt_data[i]/range;
                }
}

void measure_function(){
        particle_vx = 0;
        particle_vy = 0;
        int particle_wall = 0;
        int int_wall_pos = dynamic_wall_position_x;
        int particles_left = 0;
        for(int i = 0; i < YDIM -1;i++){
                particle_vx += n[50][i][2]+n[50][i][5]+n[50][i][8]-n[50][i][0]-n
                particle_vy += n[i][50][0]+n[i][50][1]+n[i][50][2]-n[i][50][6]-n
                }
        particle_vx = particle_vx/100.0;
        particle_vy = particle_vy/100.0;



        //measure the number of particles leaking

          particle_leakage = 0;

                for (int y=yy3; y<yy4; y++){

                        for(int v = 0; v < 9; v++)
                                {
                                particle_wall += n[int_wall_pos][y][v];
                                }
                }
```

```
            for (int x=x0; x<dynamic_wall_position_x; x++){

                    for (int y=yy3; y<yy4; y++){

                            for(int v = 0; v < 9; v++)
                                    {
                                    particle_leakage += n[x][y][v];
                                    }
                    }
            }

            particle_leakage = particle_leakage;////(dynamic_wall_position_x-x0)*(yy

            for (int x=(1+dynamic_wall_position_x); x<x1; x++){
                    for (int y=yy3; y<yy4; y++){
                            for(int v = 0; v < 9; v++)
                                    {
                                    particles_left += n[x][y][v];
                                    }
                    }
            }

            particles_left = particles_left + particle_wall*(1 - (dynamic_wall_posit

            particle_leakage = particle_leakage + particle_wall*(dynamic_wall_positi

            particle_leakage = particle_leakage - particles_left;
}


//take measurements for plotting data
void Measure(){
  //store dynamic wall data
  //position
  measure_function();

  memmove(&measure_dynamic_wall_position_x[1],&measure_dynamic_wall_position_x[0
  measure_dynamic_wall_position_x[0]=dynamic_wall_position_x;
  memmove(&measure_dynamic_wall_position_y[1],&measure_dynamic_wall_position_y[0
  measure_dynamic_wall_position_y[0]=dynamic_wall_position_y;
  //momentum
  memmove(&measure_dynamic_wall_momentum_x[1],&measure_dynamic_wall_momentum_x[0
  measure_dynamic_wall_momentum_x[0]=dynamic_wall_momentum_x;
  memmove(&measure_dynamic_wall_momentum_y[1],&measure_dynamic_wall_momentum_y[0
  measure_dynamic_wall_momentum_y[0]=dynamic_wall_momentum_y;
```

```
//velocity
memmove(&measure_dynamic_wall_vx[1],&measure_dynamic_wall_vx[0],(MeasMax-1)*si
measure_dynamic_wall_vx[0]=dynamic_wall_vx;
memmove(&measure_dynamic_wall_vy[1],&measure_dynamic_wall_vy[0],(MeasMax-1)*si
measure_dynamic_wall_vy[0]=dynamic_wall_vy;

//store static wall data
memmove(&measure_static_wall_momentum_x[1],&measure_static_wall_momentum_x[0],
measure_static_wall_momentum_x[0]=static_wall_momentum_x;
memmove(&measure_static_wall_momentum_y[1],&measure_static_wall_momentum_y[0],
measure_static_wall_momentum_y[0]=static_wall_momentum_y;

//store particle data
memmove(&measure_particle_vx[1],&measure_particle_vx[0],(MeasMax-1)*sizeof(int
measure_particle_vx[0]=particle_vx;
memmove(&measure_particle_vy[1],&measure_particle_vy[0],(MeasMax-1)*sizeof(int
measure_particle_vy[0]=particle_vy;

memmove(&measure_particle_leakage[1],&measure_particle_leakage[0],(MeasMax-1)*
measure_particle_leakage[0]=particle_leakage;

//filter data by taking the average of the data points over the range_val
average(range_val);

 //position
memmove(&measure_dynamic_wall_position_x_filt[1],&measure_dynamic_wall_positi
measure_dynamic_wall_position_x_filt[0]=filt_data[0];
memmove(&measure_dynamic_wall_position_y_filt[1],&measure_dynamic_wall_positi
measure_dynamic_wall_position_y_filt[0]=filt_data[1];
//momentum
memmove(&measure_dynamic_wall_momentum_x_filt[1],&measure_dynamic_wall_moment
measure_dynamic_wall_momentum_x_filt[0]=filt_data[2];
memmove(&measure_dynamic_wall_momentum_y_filt[1],&measure_dynamic_wall_moment
measure_dynamic_wall_momentum_y_filt[0]=filt_data[3];
//velocity
memmove(&measure_dynamic_wall_vx_filt[1],&measure_dynamic_wall_vx_filt[0],(Mea
measure_dynamic_wall_vx_filt[0]=filt_data[4];
memmove(&measure_dynamic_wall_vy_filt[1],&measure_dynamic_wall_vy_filt[0],(Mea
measure_dynamic_wall_vy_filt[0]=filt_data[5];

//store static wall data
memmove(&measure_static_wall_momentum_x_filt[1],&measure_static_wall_momentum
measure_static_wall_momentum_x_filt[0]=filt_data[6];
memmove(&measure_static_wall_momentum_y_filt[1],&measure_static_wall_momentum
measure_static_wall_momentum_y_filt[0]=filt_data[7];
```

```
  memmove(&measure_particle_vx_filt[1],&measure_particle_vx_filt[0],(MeasMax-1)*
  measure_particle_vx_filt[0]=filt_data[8];
  memmove(&measure_particle_vy_filt[1],&measure_particle_vy_filt[0],(MeasMax-1)*
  measure_particle_vy_filt[0]=filt_data[9];

  memmove(&measure_particle_leakage_filt[1],&measure_particle_leakage_filt[0],(M
  measure_particle_leakage_filt[0]=particle_leakage;//filt_data[10];
}
//re draws walls
void FindLink_Dynamic(){
        int tmp_x_shift = 25;

                linkcount_dynamic = 0;
                //draw vertical walls
                for(int dir = 0; dir < 2;dir++){
                for(int y = yy3; y<yy4+1;y++){


                        tmp_x_shift = dynamic_wall_position_x;
                        tmp_x_shift = ((tmp_x_shift%XDIM)+XDIM)%XDIM;
                        links_dynamic[dir][linkcount_dynamic][0] = tmp_x_shift;
                        links_dynamic[dir][linkcount_dynamic][1] = y;
                        links_dynamic[dir][linkcount_dynamic][2] = 0;
                        linkcount_dynamic++;
                        links_dynamic[dir][linkcount_dynamic][0] = tmp_x_shift;
                        links_dynamic[dir][linkcount_dynamic][1] = y;
                        links_dynamic[dir][linkcount_dynamic][2] = 3;
                        linkcount_dynamic++;
                        links_dynamic[dir][linkcount_dynamic][0] = tmp_x_shift;
                        links_dynamic[dir][linkcount_dynamic][1] = y;
                        links_dynamic[dir][linkcount_dynamic][2] = 6;
                        linkcount_dynamic++;

                        }


                        links_dynamic[dir][linkcount_dynamic][0] = tmp_x_shift;
                        links_dynamic[dir][linkcount_dynamic][1] = yy1;
                        links_dynamic[dir][linkcount_dynamic][2] = 0;
                        linkcount_dynamic++;

                        links_dynamic[dir][linkcount_dynamic][0] = tmp_x_shift+x
                        links_dynamic[dir][linkcount_dynamic][1] = yy0+y_link_va
                        links_dynamic[dir][linkcount_dynamic][2] = 2;
                        linkcount_dynamic++;
                }
```

```
                dynamic_wall_position_x += dynamic_wall_vx*dt;
}


void FindLink(){


  //horizontal walls
  for (int x=x0; x<x1; x++){
        x = ((x%XDIM)+XDIM)%XDIM;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 1;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 2;
        linkcount++;
  }
    for (int x=x0; x<x1; x++){
        x = ((x%XDIM)+XDIM)%XDIM;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 1;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 2;
        linkcount++;
  }

  //vertical walls
  for (int y=yy0; y<yy1+1; y++){
        links[linkcount][0] = x0; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 0;
        linkcount++;
```

```
        links[linkcount][0] = x0; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 3;
        linkcount++;
        links[linkcount][0] = x0; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 6;
        linkcount++;
    }

    for (int y=yy0; y<yy1+1; y++){
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 3;
        linkcount++;
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 6;
        linkcount++;
    }
}

void bounceback(){
    dynamic_wall_momentum_x = 0;
    dynamic_wall_momentum_y = 0;
    static_wall_momentum_x = 0;
    static_wall_momentum_y = 0;



    for (int lc=0; lc<linkcount; lc++){
      //quantity of partices

      int x=links[lc][0];
      int y=links[lc][1];
      int v=links[lc][2];
      int x_b = (((x)%XDIM)+XDIM)%XDIM;
      int y_b = (((y)%YDIM)+YDIM)%YDIM;
      //velocity
      int vx=v%3-1;
      int vy=1-v/3;
      int x_v_b = (((vx+x)%XDIM)+XDIM)%XDIM;
```

```
    int y_v_b = (((vy+y)%YDIM)+YDIM)%YDIM;
    int tmp = n[x_v_b][y_v_b][v];
    //summing all momemtums
    static_wall_momentum_x += -2*vx*(n[x][y][8-v]-tmp);
    static_wall_momentum_y += -2*vy*(n[x][y][8-v]-tmp);
    //swapping the particles trying to enter and leave to have the effect of a w
    n[x_v_b][y_v_b][v] = n[x_b][y_b][8-v];
    n[x_b][y_b][8-v] = tmp;
  }
//dynamic walls
if(dynamic_walls_on == 1){
 int wall_pos = dynamic_wall_position_x;
 double dx = abs(wall_pos - dynamic_wall_position_x);
 for (int lc=0; lc<linkcount_dynamic; lc++){
        for(int dir = 0; dir < 1; dir++){
                //printf("%i \n",dir);
                //quantity of partices
                int x=links_dynamic[dir][lc][0];
                int y=links_dynamic[dir][lc][1];
                //particle velocity
                int v=links_dynamic[dir][lc][2];
                int x_b = (((x)%XDIM)+XDIM)%XDIM;
                int y_b = (((y)%YDIM)+YDIM)%YDIM;
                int vx=v%3-1;
                int vy=1-v/3;
                int x_v_b = (((vx+x)%XDIM)+XDIM)%XDIM;
                int y_v_b = (((vy+y)%YDIM)+YDIM)%YDIM;

                int tmp = n[x_v_b][y_v_b][v];
                int iwp = dynamic_wall_position_x;//integer wall position
                int flow = 0;//particles to be moved
                double pr = dynamic_wall_vx/(1-(dynamic_wall_position_x - iwp));
                if(rand()%1000 <= 1000*pr){
                        flow = pr*n[x_b][y_b][v];
                        n[x_b+1][y_b][v] +=flow;
                        n[x_b][y_b][v] -=flow;
                }

                //summing all momemtums
                dynamic_wall_momentum_x += -2*vx*(n[x][y][8-v]-tmp);//+flow);
                dynamic_wall_momentum_y += -2*vy*(n[x][y][8-v]-tmp);

                //swapping the particles trying to enter and leave to have the e
                n[x_v_b][y_v_b][v] = n[x_b][y_b][8-v];
                n[x_b][y_b][8-v] = tmp;
```

```
                }
        }
        if(dynamic_wall_control_on == 0){
                dynamic_wall_vx = dynamic_wall_momentum_x/wall_mass;
                dynamic_wall_vy = dynamic_wall_momentum_y/wall_mass;}
        }

}

void setrho(){
  for (int x=src_x; x<src_x+src_len; x++){
    int y=src_y;
    n[x][y][0]=src_den;
    n[x][y][1]=src_den;
    n[x][y][2]=src_den;
    n[x][y][3]=src_den;
    n[x][y][4]=src_den;
    n[x][y][5]=src_den;
    n[x][y][6]=src_den;
    n[x][y][7]=src_den;
    n[x][y][8]=src_den;
  }
  for (int x=src_x_2; x<src_x_2+src_len_2; x++){
    int y=src_y_2;
    n[x][y][0]=src_den_2;
    n[x][y][1]=src_den_2;
    n[x][y][2]=src_den_2;
    n[x][y][3]=src_den_2;
    n[x][y][4]=src_den_2;
    n[x][y][5]=src_den_2;
    n[x][y][6]=src_den_2;
    n[x][y][7]=src_den_2;
    n[x][y][8]=src_den_2;
  }
}

void init(){



  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){

      dynamic_wall_position_x = 50;
      if(x > 25 && x <= 50 && y < 75 && y > 25){
        n[x][y][0]=d1;
```

```
            n[x][y][1]=d1;
            n[x][y][2]=d1;
            n[x][y][3]=d1;
            n[x][y][4]=d1;
            n[x][y][5]=d1;
            n[x][y][6]=d1;
            n[x][y][7]=d1;
            n[x][y][8]=d1;
            }
          else if(x > 50 && x < 75 && y < 75  && y > 25){
            n[x][y][0]=d2;
            n[x][y][1]=d2;
            n[x][y][2]=d2;
            n[x][y][3]=d2;
            n[x][y][4]=d2;
            n[x][y][5]=d2;
            n[x][y][6]=d2;
            n[x][y][7]=d2;
            n[x][y][8]=d2;
            }
            else if(x >= 25 || x <= 75 || y <= 75 || y >= 25){
            n[x][y][0]=0;
            n[x][y][1]=0;
            n[x][y][2]=0;
            n[x][y][3]=0;
            n[x][y][4]=0;
            n[x][y][5]=0;
            n[x][y][6]=0;
            n[x][y][7]=0;
            n[x][y][8]=0;
            }
        }
    }
}
void initShear(){
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      if (x<xdim/2){
        n[x][y][0]=1;
        n[x][y][1]=2;
        n[x][y][2]=3;
        n[x][y][3]=4;
        n[x][y][4]=5;
        n[x][y][5]=6;
        n[x][y][6]=7;
        n[x][y][7]=8;
```

```
        n[x][y][8]=9;
      }
      else {
        n[x][y][0]=8;
        n[x][y][1]=7;
        n[x][y][2]=6;
        n[x][y][3]=5;
        n[x][y][4]=4;
        n[x][y][5]=3;
        n[x][y][6]=2;
        n[x][y][7]=0;
        n[x][y][8]=0;
        }
    }
  }
}

void iterate(){
  int NI=0;
  int rate=RAND_MAX/8.;
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      NI=0;
      for (int v=0; v<V; v++) NI+=n[x][y][v];
       // first implementation: random collisions
      if (NI>0){
        for (int c=0; c<C; c++){
          //       int r1=1.0*NI*rand()/RAND_MAX;
          //       int r2=1.0*NI*rand()/RAND_MAX;
          int r1=rand()%NI;
          int r2=rand()%NI;
          if (r1!=r2){
            int v1=0;
            for (; r1>=n[x][y][v1]; v1++)
              r1-=n[x][y][v1];
            int v2=0;
            for (; r2>=n[x][y][v2]; v2++)
              r2-=n[x][y][v2];
            // now we picked two particles with velocity v1 and v2.
            int v1x=v1%3-1;
            int v1y=v1/3-1;
            int v2x=v2%3-1;
            int v2y=v2/3-1;
            // x-collision
            if ((v1x==-1)&&(v2x==1)){
              v1x=0;
```

```
  v2x=0;
}
else if ((v1x==1)&&(v2x==-1)){
  v1x=0;
  v2x=0;
}
else if ((v1x==0)&&(v2x==0)){
  int r=rand();
  if (r<rate){
    if (r<rate/2){
      v1x=-1;
      v2x=1;
    }
    else {
      v1x=1;
      v2x=-1;
    }
  }
}
else {
  int tmp=v1x;
  v1x=v2x;
  v2x=tmp;
}
// y-collision
if ((v1y==-1)&&(v2y==1)){
  v1y=0;
  v2y=0;
}
else if ((v1y==1)&&(v2y==-1)){
  v1y=0;
  v2y=0;
}
else if ((v1y==0)&&(v2y==0)){
  int r=rand();
  if (r<rate){
    if (r<rate/2){
      v1y=-1;
      v2y=1;
    }
    else {
      v1y=1;
      v2y=-1;
    }
  }
}
```

```
            else {
              int tmp=v1y;
              v1y=v2y;
              v2y=tmp;
            }
            n[x][y][v1]--;
            n[x][y][v2]--;
            v1=(v1y+1)*3+(v1x+1);
            v2=(v2y+1)*3+(v2x+1);
            n[x][y][v1]++;
            n[x][y][v2]++;
          }
        }
      }
    }
  }
  // move the particles in x-direction
  {
    int ntmp[ydim];
    for (int c=0; c<3; c++){
      for (int y=0; y<ydim; y++) ntmp[y]=n[xdim-1][y][c*3+2];
      for (int x=xdim-1; x>0; x--){
        measure_particle_velocity_front[x] = 0;
        for (int y=0; y<ydim; y++){
          n[x][y][c*3+2]=n[x-1][y][c*3+2];


        //additional code for flipping some horizontal particles
        //double flip_parts = ((double)rand()/(double)RAND_MAX)*n[x][y][5];

        //printf("pflip = %f \n",flip_parts);

        //n[x][y][1] += flip_parts;
        //n[x][y][7] -= flip_parts;


        //for(int x0 = 0;x0<xdim;x0++){
                //sum up velocities to get velocity front
        //measure_particle_velocity_front[y] += n[x0][y][7] - n[x0][y][1];

        //}

        }
      }

        //go here
```

```
      for (int y=0; y<ydim; y++) n[0][y][c*3+2]=ntmp[y];
    }
    for (int c=0; c<3; c++){
      for (int y=0; y<ydim; y++) ntmp[y]=n[0][y][c*3];
      for (int x=0; x<xdim-1; x++){
        for (int y=0; y<ydim; y++)
          n[x][y][c*3]=n[x+1][y][c*3];
      }
      for (int y=0; y<ydim; y++) n[xdim-1][y][c*3]=ntmp[y];
    }
  }
    // move the particles in y-direction
  {
    int ntmp[xdim];
    for (int c=0; c<3; c++){
      for (int x=0; x<xdim; x++) ntmp[x]=n[x][ydim-1][c];
      for (int y=ydim-1; y>0; y--){
        for (int x=0; x<xdim; x++)
          n[x][y][c]=n[x][y-1][c];
      }
      for (int x=0; x<xdim; x++) n[x][0][c]=ntmp[x];
    }
    for (int c=0; c<3; c++){
      for (int x=0; x<xdim; x++) ntmp[x]=n[x][0][c+6];
      for (int y=0; y<ydim-1; y++){
        for (int x=0; x<xdim; x++)
          n[x][y][c+6]=n[x][y+1][c+6];
      }
      for (int x=0; x<xdim; x++) n[x][ydim-1][c+6]=ntmp[x];
    }
  }
  if(dynamic_walls_on == 1){
        FindLink_Dynamic();
        }
  bounceback();
}

void getdata(){
  if (NUreq||Nreq){
    NUreq=0;
    Nreq=0;
    for (int x=0; x<xdim; x++)
      for (int y=0; y<ydim; y++){
        N[x][y]=0;
        NU[x][y][0]=0;
        NU[x][y][1]=0;
```

```
        for (int v=0; v<V; v++){
          N[x][y]+=n[x][y][v];
          NU[x][y][0]+=n[x][y][v]*(v%3-1);
          NU[x][y][1]+=n[x][y][v]*(1-v/3);
        }
      }
  }
}

void debug(){
  int NUX=0,NUY=0,NN=0;
  for (int x=0; x<xdim; x++)
    for (int y=0; y<ydim; y++){
      for (int v=0; v<V; v++){
        NN+=n[x][y][v];
        NUX+=n[x][y][v]*(v%3-1);
        NUY+=n[x][y][v]*(1-v/3);
      }
    }
  printf("N=%i; NUX=%i; NUY=%i;\n",NN,NUX,NUY);
}

void main(){
  int done=0,sstep=0,cont=0,repeat=10;
  init();
  Measure();
  DefineGraphNxN_R("N",&N[0][0],&XDIM,&YDIM,&Nreq);
  DefineGraphNxN_RxR("NU",&NU[0][0][0],&XDIM,&YDIM,&NUreq);
  DefineGraphN_R("dynamic wall position x",&measure_dynamic_wall_position_x[0],&
  DefineGraphN_R("dynamic wall position y",&measure_dynamic_wall_position_y[0],&
  DefineGraphN_R("dynamic wall momentum x",&measure_dynamic_wall_momentum_x[0],&
  DefineGraphN_R("dynamic wall momentum y",&measure_dynamic_wall_momentum_y[0],&
  DefineGraphN_R("dynamic wall vx ",&measure_dynamic_wall_vx[0],&MeasLen,NULL);
  DefineGraphN_R("dynamic wall vx",&measure_dynamic_wall_vy[0],&MeasLen,NULL);
  DefineGraphN_R("static wall momentum x",&measure_static_wall_momentum_x[0],&Me
  DefineGraphN_R("static wall momentum y",&measure_static_wall_momentum_y[0],&Me
  DefineGraphN_R("particle vx",&measure_particle_vx[0],&MeasLen,NULL);
  DefineGraphN_R("particle vy",&measure_particle_vy[0],&MeasLen,NULL);
  DefineGraphN_R("particle leakage",&measure_particle_leakage[0],&MeasLen,NULL);
  DefineGraphN_R("particle vx front",&measure_particle_velocity_front[0],&MeasLe
  //filtered graphs
  DefineGraphN_R("dynamic wall position x filt",&measure_dynamic_wall_position_
  DefineGraphN_R("dynamic wall position y filt",&measure_dynamic_wall_position_
  DefineGraphN_R("dynamic wall momentum x filt",&measure_dynamic_wall_momentum_
  DefineGraphN_R("dynamic wall momentum y filt",&measure_dynamic_wall_momentum_
  DefineGraphN_R("dynamic wall vx filt",&measure_dynamic_wall_vx_filt[0],&MeasLe
```

```
DefineGraphN_R("dynamic wall vx filt",&measure_dynamic_wall_vy_filt[0],&MeasLe
DefineGraphN_R("static wall momentum x filt",&measure_static_wall_momentum_x_
DefineGraphN_R("static wall momentum y filt",&measure_static_wall_momentum_y_
DefineGraphN_R("particle vx filt",&measure_particle_vx_filt[0],&MeasLen,NULL);
DefineGraphN_R("particle vy filt",&measure_particle_vy_filt[0],&MeasLen,NULL);
DefineGraphN_R("particle leakage filt",&measure_particle_leakage_filt[0],&Meas
StartMenu("LG",1);
DefineFunction("init",init);
DefineFunction("init shear",initShear);
StartMenu("Measure",0);
DefineInt("range_val",&range_val);
DefineGraph(curve2d_,"Measurements");
EndMenu();
StartMenu("Wall",1);
DefineInt("x0", &x0);
DefineInt("x1", &x1);
DefineDouble("dynamic_wall_vx", &dynamic_wall_vx);
DefineDouble("dynamic_wall_position_x", &dynamic_wall_position_x);
DefineDouble("wall_mass", &wall_mass);
DefineInt("yy0", &yy0);
DefineInt("yy1", &yy1);
DefineInt("yy3", &yy3);
DefineInt("yy4", &yy4);
DefineInt("type_link", &type_link);
DefineInt("y link var", &y_link_var);
DefineInt("x link var", &x_link_var);
DefineFunction("Add Wall",FindLink);
DefineInt("link count",&linkcount);
DefineInt("dynamic link count",&linkcount_dynamic);
DefineInt("dynamic walls on",&dynamic_walls_on);
DefineInt("dynamic wall control on",&dynamic_wall_control_on);
EndMenu();
StartMenu("Particle Source",1);
DefineInt("src_den",&src_den);
DefineInt("src_x",&src_x);
DefineInt("src_y",&src_y);
DefineInt("src_den_2",&src_den_2);
DefineInt("src_x_2",&src_x_2);
DefineInt("src_y_2",&src_y_2);
DefineInt("d1",&d1);
DefineInt("d2",&d2);
EndMenu();
DefineGraph(contour2d_,"Graph");
DefineInt("C", &C);
DefineInt("repeat",&repeat);
DefineBool("sstep",&sstep);
```

```
  DefineBool("cont",&cont);
  DefineBool("done",&done);
  EndMenu();


  while (!done){
    Events(1);
    getdata();
    DrawGraphs();
    if (cont || sstep){
      sstep=0;

      for (int i=0; i<repeat; i++) {
        iterate();
        Measure();
        //setrho();
      }
    } else sleep(1);
  }
}
```

A.3 Code for Approach 3

```
// Lattice gas code in 1d.
//change the initial zero point for the Ux ~ mean particle velocity

//what about dividing by total number of particles for the theoretical?
//zero points seem to be really small
//why cant we just change the probability of collisions to make the parabola fit
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>
#include <mygraph.h>
#include <string.h>

#define xdim 100
#define ydim 100
#define V 9       //number of velocities
#define MeasMax 200
int XDIM=xdim,YDIM=ydim;
int C=300;//chaneged to 300 for viscocity...
double N[xdim][ydim],NU[xdim][ydim][2];
int Nreq=0,NUreq=0;
int n[xdim][ydim][V];
```

```
// Boundary variables
#define LINKMAX 10000
#define LINKMAX_DYNAMIC 10000



double particle_flip_w = 0.032;

double viscocity = 0;

double dt = 1.0;

int x_link_var = 0;
int y_link_var = 1;
int type_link = 0;

//variables for measuring tube momentum
double last_1 = 0;
double last_2 = 0;


//particle source parameters
int src_x = 85,src_y = 85,src_len = 5,src_den = 10;
int src_x_2 = 15,src_y_2 = 15,src_len_2 = 5,src_den_2 = 10;
int var1 =10;
//variables for measuring values
//particle velocity
int particle_vx = 0,particle_vy ,particle_leakage = 0;
double measure_particle_vx[MeasMax],measure_particle_vy[MeasMax];
double measure_particle_vx_filt[MeasMax],measure_particle_vy_filt[MeasMax];
int vx_m = 0,vy_m = 0;
int d1 = 0,d2 = 50;
int ux_zero_flag = 0;
double ux_zero_point[ydim];
//int ux_zero_curve_1[ydim];

//link variables
int linkcount_dynamic = 0,links_dynamic[2][LINKMAX_DYNAMIC][3];//link x y and v(
double links_dynamic_pool[2][LINKMAX_DYNAMIC];
double links_dynamic_velocity[LINKMAX_DYNAMIC][2];//link velocities
double wall_mass = 75000.1;
int yy3 = 26,yy4 = 75,dynamic_walls_on = 1,dynamic_wall_control_on = 1;
double flow = 0;
//dynamic wall
// position
//double x_shift = 50;
```

```
double dynamic_wall_position_x = 0,dynamic_wall_position_y = 0;
//velocity
double dynamic_wall_vx = 0,dynamic_wall_vy = 0;
//momentum
int dynamic_wall_momentum_x = 0,dynamic_wall_momentum_y = 0;
//dynamic wall position for graphing
double measure_dynamic_wall_position_x[MeasMax],measure_dynamic_wall_position_y[
//filtered
double measure_dynamic_wall_position_x[MeasMax],measure_dynamic_wall_position_y[
//
double measure_dynamic_wall_position_x_filt[MeasMax],measure_dynamic_wall_posit
//dynamic wall velocity
double measure_dynamic_wall_vx[MeasMax],measure_dynamic_wall_vy[MeasMax];
//filtered
double measure_dynamic_wall_vx_filt[MeasMax],measure_dynamic_wall_vy_filt[MeasMa
//dynamic wall momentum
double measure_dynamic_wall_momentum_x[MeasMax],measure_dynamic_wall_momentum_y[
//filtered
double measure_dynamic_wall_momentum_x_filt[MeasMax],measure_dynamic_wall_momen


double measure_particle_velocity_front[MeasMax];
double measure_particle_velocity_front_last[MeasMax];
double measure_particle_force_front[MeasMax];
double theoretical_particle_vx_front[MeasMax];
//static wall variables
int linkcount=0,links[LINKMAX][3];

//points for drawing walls
int x0=0,x1=100,yy0 = 25,yy1 = 74;

//static wall momentum
double static_wall_momentum_x = 0,static_wall_momentum_y = 0;
double measure_static_wall_momentum_x[MeasMax],measure_static_wall_momentum_y[Me
//filtered data
double measure_static_wall_momentum_x_filt[MeasMax],measure_static_wall_momentu

int MeasLen = MeasMax/2;
int range_val = 20;
int val[] = {0,0};
int filt_data[] ={0,0,0,0,0,0,0,0,0,0,0,0}; //dynamic walls -> x,y,px,py,vx,vy
static walls -> px py particles -> vx vy leakage


//added a running average to smooth out the graphs
void average(int range){
```

```
        //val[0] = 0;
        //val[1] = 0;
        //clear previous filtered data
        for(int i = 0; i < 11;i++){
                filt_data[i] = 0;
                }
        //update with new data
        for(int i = 0; i < range;i++){
                filt_data[0] += measure_dynamic_wall_position_x[i];
                filt_data[1] += measure_dynamic_wall_position_y[i];
                filt_data[2] += measure_dynamic_wall_momentum_x[i];
                filt_data[3] += measure_dynamic_wall_momentum_y[i];
                filt_data[4] += measure_dynamic_wall_vx[i];
                filt_data[5] += measure_dynamic_wall_vy[i];
                filt_data[6] += measure_static_wall_momentum_x[i];
                filt_data[7] += measure_static_wall_momentum_y[i];
                filt_data[8] += measure_particle_vx[i];
                filt_data[9] += measure_particle_vy[i];
                //filt_data[10] += measure_particle_leakage[i];
                }
        //average values
        for(int i = 0; i < 11;i++){
                filt_data[i] = filt_data[i]/range;
                }
}

void measure_function(){
        particle_vx = 0;
        particle_vy = 0;
        //int total_particles = 0;
        int total_particles = 0;
        double shift[ydim];
        for(int i = 0; i < YDIM -1;i++){
                total_particles = 0;
                measure_particle_velocity_front[i] = 0;
                for(int x0 = 0;x0<xdim;x0++){
                        measure_particle_velocity_front[i] += n[x0][i][5]+n[x0][
                        }
                //find average particle velocity
                if(total_particles > 0){
                        measure_particle_velocity_front[i] = (double)(measure_p
                        theoretical_particle_vx_front[i] = (double)(particle_flip

                        }
                else{
                        measure_particle_velocity_front[i] = 0;
```

```
                        theoretical_particle_vx_front[i] = 0;
                        }
                //setting zero point for ux
                if(ux_zero_flag == 0)ux_zero_point[i] += measure_particle_velocity_front[i];
                if(ux_zero_flag == 1)ux_zero_point[i] = (measure_particle_velocity_front[i]);
                printf("zp = %f \n",ux_zero_point[i]);

                measure_particle_force_front[i] = (measure_particle_velocity_front[i]-theoretical_particle_vx_front[i])*(-1.0);

                }


                //set flags after ux at iteration 0 and 1 have been recorded
                printf("flag = %i \n",ux_zero_flag);
                if(ux_zero_flag == 1)
                        {ux_zero_flag = 2;
                        //partical_flip_w = f_partical_flip_w;
                        for(int i = 0;i<ydim;i++)shift[i]=ux_zero_point[i];
                        }
                if(ux_zero_flag == 0)ux_zero_flag = 1;


        particle_vx = particle_vx/100.0;
        particle_vy = particle_vy/100.0;



        //measure the number of particles leaking

          particle_leakage = 0;
                for(int x = 0; x < xdim; x++){
                        for(int y = 0;y < ydim;y++){
                                for(int v = 0; v < 9;v++){
                                        particle_leakage += n[x][y][v];
                                        }
                                }
                        }
}


//take measurements for plotting data
void Measure(){
 //measure the force on the particles in the tube.
  measure_function();
}
```

```
void moveParticles(){
for(int x = 0;x<xdim;x++){
        for(int y = 0;y<ydim;y++){
                //additional code for flipping some horizontal particles
                int flip_parts = particle_flip_w*((double)rand()/RAND_MAX)*n[x][
                n[x][y][3] += flip_parts;
                n[x][y][5] -= flip_parts;
                flip_parts = particle_flip_w*((double)rand()/(double)RAND_MAX)*n
                n[x][y][6] += flip_parts;
                n[x][y][8] -= flip_parts;
                flip_parts = particle_flip_w*((double)rand()/(double)RAND_MAX)*n
                n[x][y][0] += flip_parts;
                n[x][y][2] -= flip_parts;
                }
        }
}

void FindLink(){


  //horizontal walls
  for (int x=x0; x<x1; x++){
        x = ((x%XDIM)+XDIM)%XDIM;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 1;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 2;
        linkcount++;
  }
    for (int x=x0; x<x1; x++){
        x = ((x%XDIM)+XDIM)%XDIM;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 1;
        linkcount++;
```

```
            links[linkcount][0] = x; //x-position
            links[linkcount][1] = yy1;
            links[linkcount][2] = 2;
            linkcount++;
    }

}

void bounceback(){
    dynamic_wall_momentum_x = 0;
    dynamic_wall_momentum_y = 0;
    static_wall_momentum_x = 0;
    static_wall_momentum_y = 0;



    for (int lc=0; lc<linkcount; lc++){
        //quantity of partices

        int x=links[lc][0];
        int y=links[lc][1];
        int v=links[lc][2];
        int x_b = (((x)%XDIM)+XDIM)%XDIM;
        int y_b = (((y)%YDIM)+YDIM)%YDIM;
        //velocity
        int vx=v%3-1;
        int vy=1-v/3;
        int x_v_b = (((vx+x)%XDIM)+XDIM)%XDIM;
        int y_v_b = (((vy+y)%YDIM)+YDIM)%YDIM;
        int tmp = n[x_v_b][y_v_b][v];
        //summing all momemtums
        static_wall_momentum_x += -2*vx*(n[x][y][8-v]-tmp);
        static_wall_momentum_y += -2*vy*(n[x][y][8-v]-tmp);
        //swapping the particles trying to enter and leave to have the effect of a u
        n[x_v_b][y_v_b][v] = n[x_b][y_b][8-v];
        n[x_b][y_b][8-v] = tmp;
    }
}

void setrho(){
    for (int x=src_x; x<src_x+src_len; x++){
        int y=src_y;
        n[x][y][0]=src_den;
        n[x][y][1]=src_den;
        n[x][y][2]=src_den;
        n[x][y][3]=src_den;
```

```
    n[x][y][4]=src_den;
    n[x][y][5]=src_den;
    n[x][y][6]=src_den;
    n[x][y][7]=src_den;
    n[x][y][8]=src_den;
  }
  for (int x=src_x_2; x<src_x_2+src_len_2; x++){
    int y=src_y_2;
    n[x][y][0]=src_den_2;
    n[x][y][1]=src_den_2;
    n[x][y][2]=src_den_2;
    n[x][y][3]=src_den_2;
    n[x][y][4]=src_den_2;
    n[x][y][5]=src_den_2;
    n[x][y][6]=src_den_2;
    n[x][y][7]=src_den_2;
    n[x][y][8]=src_den_2;
  }
}

void init(){

  linkcount = 0;
  FindLink();


  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){

      dynamic_wall_position_x = 50;
     /* if(x > 25 && x <= 50 && y < 75 && y > 25){
        n[x][y][0]=d1;
        n[x][y][1]=d1;
        n[x][y][2]=d1;
        n[x][y][3]=d1;
        n[x][y][4]=d1;
        n[x][y][5]=d1;
        n[x][y][6]=d1;
        n[x][y][7]=d1;
        n[x][y][8]=d1;
        }
      else*/ if(x >= 0 && x <= 99 && y < 75 && y > 25){
        n[x][y][0]=d2;
        n[x][y][1]=d2;
        n[x][y][2]=d2;
        n[x][y][3]=d2;
```

```
            n[x][y][4]=d2;
            n[x][y][5]=d2;
            n[x][y][6]=d2;
            n[x][y][7]=d2;
            n[x][y][8]=d2;
            }
            else if(/*x <= 25 || x >= 75 ||*/ y >= 75 || y <= 25){
            n[x][y][0]=0;
            n[x][y][1]=0;
            n[x][y][2]=0;
            n[x][y][3]=0;
            n[x][y][4]=0;
            n[x][y][5]=0;
            n[x][y][6]=0;
            n[x][y][7]=0;
            n[x][y][8]=0;
            }
      }
    }
}
void initShear(){
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      if (x<xdim/2){
        n[x][y][0]=1;
        n[x][y][1]=2;
        n[x][y][2]=3;
        n[x][y][3]=4;
        n[x][y][4]=5;
        n[x][y][5]=6;
        n[x][y][6]=7;
        n[x][y][7]=8;
        n[x][y][8]=9;
      }
      else {
        n[x][y][0]=8;
        n[x][y][1]=7;
        n[x][y][2]=6;
        n[x][y][3]=5;
        n[x][y][4]=4;
        n[x][y][5]=3;
        n[x][y][6]=2;
        n[x][y][7]=0;
        n[x][y][8]=0;
        }
      }
```

```
  }
}

void iterate(){
  int NI=0;
  int rate=RAND_MAX/8.;
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      NI=0;
      for (int v=0; v<V; v++) NI+=n[x][y][v];
       // first implementation: random collisions
      if (NI>0){
        for (int c=0; c<C; c++){
          //       int r1=1.0*NI*rand()/RAND_MAX;
          //       int r2=1.0*NI*rand()/RAND_MAX;
          int r1=rand()%NI;
          int r2=rand()%NI;
          if (r1!=r2){
            int v1=0;
            for (; r1>=n[x][y][v1]; v1++)
              r1-=n[x][y][v1];
            int v2=0;
            for (; r2>=n[x][y][v2]; v2++)
              r2-=n[x][y][v2];
            // now we picked two particles with velocity v1 and v2.
            int v1x=v1%3-1;
            int v1y=v1/3-1;
            int v2x=v2%3-1;
            int v2y=v2/3-1;
            // x-collision
            if ((v1x==-1)&&(v2x==1)){
              v1x=0;
              v2x=0;
            }
            else if ((v1x==1)&&(v2x==-1)){
              v1x=0;
              v2x=0;
            }
            else if ((v1x==0)&&(v2x==0)){
              int r=rand();
              if (r<rate){
                if (r<rate/2){
                  v1x=-1;
                  v2x=1;
                }
                else {
```

```
            v1x=1;
            v2x=-1;
          }
        }
      }
      else {
        int tmp=v1x;
        v1x=v2x;
        v2x=tmp;
      }
      // y-collision
      if ((v1y==-1)&&(v2y==1)){
        v1y=0;
        v2y=0;
      }
      else if ((v1y==1)&&(v2y==-1)){
        v1y=0;
        v2y=0;
      }
      else if ((v1y==0)&&(v2y==0)){
        int r=rand();
        if (r<rate){
          if (r<rate/2){
            v1y=-1;
            v2y=1;
          }
          else {
            v1y=1;
            v2y=-1;
          }
        }
      }
      else {
        int tmp=v1y;
        v1y=v2y;
        v2y=tmp;
      }
      n[x][y][v1]--;
      n[x][y][v2]--;
      v1=(v1y+1)*3+(v1x+1);
      v2=(v2y+1)*3+(v2x+1);
      n[x][y][v1]++;
      n[x][y][v2]++;
    }
  }
}
```

```
  }
}
// move the particles in x-direction
{
  int ntmp[ydim];
  for (int c=0; c<3; c++){
    for (int y=0; y<ydim; y++) ntmp[y]=n[xdim-1][y][c*3+2];
    for (int x=xdim-1; x>0; x--){
      measure_particle_velocity_front[x] = 0;
      for (int y=0; y<ydim; y++){
        n[x][y][c*3+2]=n[x-1][y][c*3+2];


      }
    }

      //go here
    for (int y=0; y<ydim; y++) n[0][y][c*3+2]=ntmp[y];
  }
  for (int c=0; c<3; c++){
    for (int y=0; y<ydim; y++) ntmp[y]=n[0][y][c*3];
    for (int x=0; x<xdim-1; x++){
      for (int y=0; y<ydim; y++)
        n[x][y][c*3]=n[x+1][y][c*3];
    }
    for (int y=0; y<ydim; y++) n[xdim-1][y][c*3]=ntmp[y];
  }
}
  // move the particles in y-direction
{
  int ntmp[xdim];
  for (int c=0; c<3; c++){
    for (int x=0; x<xdim; x++) ntmp[x]=n[x][ydim-1][c];
    for (int y=ydim-1; y>0; y--){
      for (int x=0; x<xdim; x++)
        n[x][y][c]=n[x][y-1][c];
    }
    for (int x=0; x<xdim; x++) n[x][0][c]=ntmp[x];
  }
  for (int c=0; c<3; c++){
    for (int x=0; x<xdim; x++) ntmp[x]=n[x][0][c+6];
    for (int y=0; y<ydim-1; y++){
      for (int x=0; x<xdim; x++)
        n[x][y][c+6]=n[x][y+1][c+6];
    }
    for (int x=0; x<xdim; x++) n[x][ydim-1][c+6]=ntmp[x];
```

```
    }
  }
  moveParticles();
  bounceback();
}

void getdata(){
  if (NUreq||Nreq){
    NUreq=0;
    Nreq=0;
    for (int x=0; x<xdim; x++)
      for (int y=0; y<ydim; y++){
        N[x][y]=0;
        NU[x][y][0]=0;
        NU[x][y][1]=0;
        for (int v=0; v<V; v++){
          N[x][y]+=n[x][y][v];
          NU[x][y][0]+=n[x][y][v]*(v%3-1);
          NU[x][y][1]+=n[x][y][v]*(1-v/3);
        }
      }
  }
}

void debug(){
  int NUX=0,NUY=0,NN=0;
  for (int x=0; x<xdim; x++)
    for (int y=0; y<ydim; y++){
      for (int v=0; v<V; v++){
        NN+=n[x][y][v];
        NUX+=n[x][y][v]*(v%3-1);
        NUY+=n[x][y][v]*(1-v/3);
      }
    }
  printf("N=%i; NUX=%i; NUY=%i;\n",NN,NUX,NUY);
}

void main(){
  int done=0,sstep=0,cont=0,repeat=10;
  init();
  Measure();
  DefineGraphNxN_R("N",&N[0][0],&XDIM,&YDIM,&Nreq);
  DefineGraphNxN_RxR("NU",&NU[0][0][0],&XDIM,&YDIM,&NUreq);

  DefineGraphN_R("particle_vx_front",&measure_particle_velocity_front[0],&MeasLe
  DefineGraphN_R("UX_ZERO_POINT",&ux_zero_point[0],&MeasLen,NULL);
```

```
  DefineGraphN_R("particle_force_x_front",&measure_particle_force_front[0],&Meas
  DefineGraphN_R("particle_vx_front_theoretical",&theoretical_particle_vx_front[

  StartMenu("LG",1);
  DefineFunction("init",init);
  DefineFunction("init_shear",initShear);
  StartMenu("Measure",1);
  DefineInt("range_val",&range_val);
  DefineInt("ux_zero_point_flag",&ux_zero_flag);
  DefineGraph(curve2d_,"Measurements");
  EndMenu();
  StartMenu("Wall",1);
  DefineDouble("particle_flip_w", &particle_flip_w);
  DefineFunction("Add_Wall",FindLink);
  DefineInt("link_count",&linkcount);
  EndMenu();
  DefineGraph(contour2d_,"Graph");
  DefineInt("C", &C);
  DefineInt("repeat",&repeat);
  DefineBool("sstep",&sstep);
  DefineBool("cont",&cont);
  DefineBool("done",&done);
  EndMenu();


  while (!done){
    Events(1);
    getdata();
    DrawGraphs();
    if (cont || sstep){
      sstep=0;

      for (int i=0; i<repeat; i++) {
        iterate();
        Measure();
        //setrho();
      }
    } else sleep(1);
  }
}
```