

# Homework 5: Feynman problem

David Jedynak

April 8, 2018

## Abstract

We show how an angled tube can be created in an integer lattice gas method simulation. This angled tube is used with particle sources to perform an experiment to provide insight on the Feynman problem. The momentum transferred from the particles to the tube will be estimated to understand the response of the tube to different densities. The influence of density ratios inside and outside the tube on the force will be observed.

## 1 Introduction

The Feynman problem presents the question of what will occur when an angled tube attached to a pivot point has fluid pushed out or sucked into the tube. Will the tube rotate clockwise or counter clockwise? My prediction is that the tube will rotate clockwise when air is pushed out of the tube, if the experiment is set up as the figure below.

The Lattice Gas simulation where this experiment will be executed has constraints that particle momentum and the number of particles must be conserved. The collisions and resulting velocities can be modeled using random processes that can lower the computation cost from prior simulations that would calculate interactions between every particle based on distance, charge, mass, and fields. This can be done because the prior simulations exhibit this random behavior.

## 2 Walls

The Feynman experiment will require 2 Dimensional walls to create a right angled tube. Particles must be reflected if they hit a wall. Particles cannot pass through walls. To accomplish this requirement, velocities in different directions can be swapped. The lattice gas simulation is composed of many cells. Each cell has 9(3 by 3 grid) different values to describe particles motion in a cell.

Here are the indexes for each velocity. The value behind each index is the magnitude of the velocity in that direction

	0	1	2
Indexes	3	4	5
	6	7	8

Velocities corresponding to Indexes above

$(-v_x, +v_y)$	$(0, +v_y)$	$(+v_x, +v_y)$
$(-v_x, 0)$	$(0, 0)$	$(+v_x, 0)$
$(-v_x, -v_y)$	$(0, -v_y)$	$(+v_x, -v_y)$

## 2.1 Walls as Links

A wall will be composed of links. The links will have an x and y coordinates that will be the same as the lattice cell they are interacting with. Additionally these links will also store the index of the velocity position they will be swapping. Horizontal walls will be switching  $[0,1,2]$  will be switched with  $[8,7,6]$  respectively. Vertical walls will be switching  $[0,3,6]$  will be switched with  $[8,5,2]$  respectively. Only one of these sets for each Vertical and Horizontal walls will need to be stored because both will be able to derive the second set's indexes from the first set.

Code defining Horizontal Walls

test

Code Defining Vertical Walls

test

Additional code added to prevent the tube from leaking particles

test

Code Defining How particles reflect off of walls (Both horizontal and vertical)

test

## 3 Calculating Momentum

$p = \text{mass} * \text{velocity}$

## 4 Force as a Function of Density Ratios

## 5 Results

### 5.1 Wall

### 5.2 Calculating Momentum

### 5.3 Force as a Function of Density Ratios

These simulations show that particles reenter at the boundaries of the simulation and that no spurious effects are obvious at the periodic boundaries.

## 6 Conclusions

In this report we have shown how to implement periodic boundary conditions in a Molecular Dynamics simulation. We have verified that our algorithm is free from obvious errors by setting up a periodic system and observing that it shows no spurious edge effects.

## A Code

```
// 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 200
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
int x0=10,x1=20,x2=40,y2=20,yy0=10,yy1=20,yy2=40,close_tube=0;
int linkcount=0,links[LINKMAX][3];
//variables for measuring tube momentum
int tot_vx=0,tot_vy=0;
int tot_vx_list[10],tot_vy_list[10];
//particle source parameters
int src_x=12,src_y=30,src_len=5,src_den=15;
int var1=10;

//graphing
double momentum_est_x[MeasMax],momentum_est_y[MeasMax];
int MeasLen=MeasMax/2;
int range_val=5;
int val[]={0,0};
//added a running average to smooth out the graphs
void average(int range){
```



```

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++;
}
for (int x=x0; x<x2+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++;
}
//vertical walls
if(close_tube == 1){
for (int y=yy0; y<yy1+1; y++){
links[linkcount][0] = x2; //x-position
links[linkcount][1] = y;
links[linkcount][2] = 0;
linkcount++;
links[linkcount][0] = x2; //x-position
links[linkcount][1] = y;
links[linkcount][2] = 3;
linkcount++;
links[linkcount][0] = x2; //x-position
links[linkcount][1] = y;
links[linkcount][2] = 6;
linkcount++;
}
}
for (int y=yy1; y<yy2+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++;
    }
    for (int y=yy0; y<yy2+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++;
    }
}

void bounceback(){
    tot_vx =0;
    tot_vy =0;
    for (int lc=0; lc<linkcount; lc++){
        int x=links[lc][0];
        int y=links[lc][1];
        int v=links[lc][2];
        int vx=v%3-1;
        int vy=1-v/3;
        //to find the total momentum of the system
        //the tube momentum will be proportional to the
        //negative sum of all the velocities of the particles..
        //could add mass to make more accurate.. assume mass is 1 for now
        tot_vy += -2*(n[x][y][1] - n[x][y][7] + n[x][y][2] - n[x][y][6] - n[x][y][3] + n[x][y][4]);
        tot_vx += -2*(-n[x][y][3] + n[x][y][5] + n[x][y][2] - n[x][y][6] + n[x][y][1] - n[x][y][7]);

        int tmp= n[x+vx][y+vy][v];
        n[x+vx][y+vy][v]= n[x][y][8-v];
        n[x][y][8-v]=tmp;
    }
}

```

```

    Measure();
}

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;
    }
}

void init(){
    for (int x=0; x<xdim; x++){
        for (int y=0; y<ydim; y++){
            if ((abs(xdim/2-x)<25)&&(abs(ydim/2-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;
            }
            else
            {
                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]=20;
                n[x][y][1]=30;
                n[x][y][2]=20;
                n[x][y][3]=20;
                n[x][y][4]=10;
                n[x][y][5]=20;
                n[x][y][6]=0;
                n[x][y][7]=10;
                n[x][y][8]=0;
            }
            else {
                n[x][y][0]=0;
                n[x][y][1]=10;
                n[x][y][2]=00;
                n[x][y][3]=20;
                n[x][y][4]=10;
                n[x][y][5]=20;
                n[x][y][6]=20;
                n[x][y][7]=30;
                n[x][y][8]=20;
            }
        }
    }
}

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];
    }
}
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("momentum_est_x",&momentum_est_x[0],&MeasLen,NULL);
    DefineGraphN_R("momentum_est_y",&momentum_est_y[0],&MeasLen,NULL);
    StartMenu("LG",1);
    DefineFunction("init",init);
    DefineFunction("init_shear",initShear);
    StartMenu("Measure",0);
    DefineGraph(curve2d_,"Measurements");
    DefineInt("tot_vx",&tot_vx);
    DefineInt("tot_vy",&tot_vy);
    EndMenu();
    StartMenu("Wall",0);
    DefineInt("x0",&x0);
    DefineInt("x1",&x1);
    DefineInt("x2",&x2);
    DefineInt("yy0",&yy0);
    DefineInt("yy1",&yy1);
    DefineInt("yy2",&yy2);
    DefineInt("close_tube",&close_tube);
    DefineFunction("Add_Wall",FindLink);
    DefineInt("link_count",&linkcount);
    EndMenu();
    StartMenu("Particle_Source",0);
    DefineInt("src_den",&src_den);
    DefineInt("src_x",&src_x);
    DefineInt("src_y",&src_y);
    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();
            setrho();
        }
    } else sleep(1);
}
}

```