Hyperbolic Conservation Laws

Joe Majkut

In this project I attempt two finite-difference approximation methods for a two-dimensional hyperbolic conservation law, which arises from my senior thesis project on insect swarming, and find that the task is probably better suited for other numerical techniques. My solutions suffer from quickly-forming instabilities which limit the time and space scales along which I can calculate anything meaningful.

1 Motivation

In the natural world, the aggregation of animals is a familiar phenomena with well-understood purposes: mating, protection, and collective action are the main three. For my senior thesis, I undertook the task of building a reasonable model for the swarming of fruit flies, *drosophila melanogaster*, whose flights are characterized by straight horizontal jaunts interrupted by quick turns of ≈ 90 degrees.

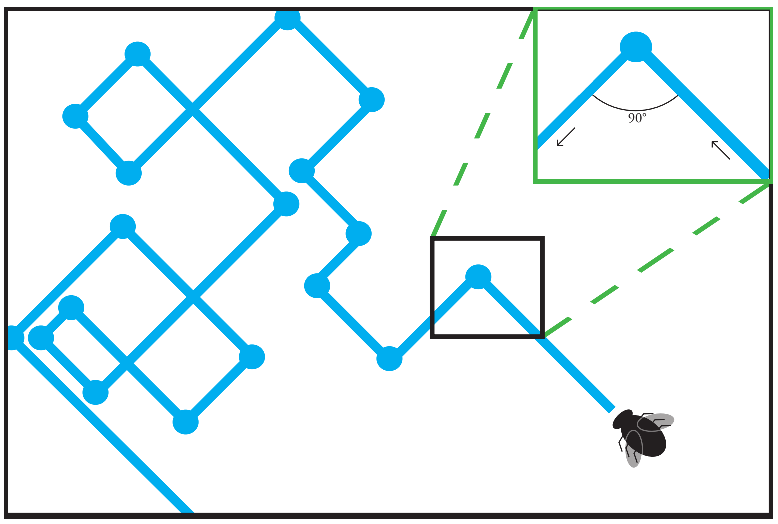

Figure 1: A saccading flight path as seen from above. The saccading behavior is limited to the horizontal plane and independent of vertical motion.

Additionally, flies interact with both their environment and each other. For instance, when a fly is in a highly desirable position, i.e. near a food source or other flies, it will take shorter flights in between turns. Hence, I decided to model the movement of each fly as a random walk, in one spatial dimension, governed by an internal variable that counts down the time until the next saccade. In the end, I would like to be able to say things about the long-term behavior of a swarm, such as its shape or searching efficiency over a domain. The model implemented in my thesis uses a individual-based, stochastic, formulation to track the positions of the N bugs in a swarm. When their individual clocks, denoted τ, reach zero, then the fly is randomly assigned a new direction and value of τ. This model is computationally fast, but each realization of it is slightly different, and in an effort to gain a deeper understanding of the dynamics of this one-dimensional swarm I seek, here, to build a continuum limit of my Lagrangian model.

2 Model Set-Up

The continuum output I seek is the probability density, ρ for the swarm as a function of space, x, and time, t. In oder to find this, I track the right- and left-moving densities, $r(x,t)$ and $l(x,t)$, where,

$$\rho(x,y) = r(x,t) + l(x,t).$$

Then, to reflect the fact that flies in the Lagrangian model each have a distinct time to next saccade, τ, I consider r and l as functions of space, time, and $\tau > 0$. The density at $\tau = 0$ is then redistributed into τ by the same rules that govern the distribution of flight lengths in the Lagrangian model. Figure 2 shows the fluxs in and out of a rectangle in (x,τ). To simplify the derivation of the governing equations for this system, I use the material derivative. Consider a parcel moving with the right-traveling group. Then by the total derivative,

$$\frac{Dr}{dt} = F(\rho, x, t, \tau),$$

where $F(\rho, x, t, \tau)$ is the redistribution function, which carries mass from the x-axis back into the domain and governs how the environment is influencing the swarm. Then, after taking some partial derivatives,

$$\begin{aligned} \frac{Dr}{dt} &= \tfrac{\partial r}{\partial t} + \tfrac{\partial r}{\partial x}\tfrac{\partial x}{\partial t} + \tfrac{\partial r}{\partial \tau}\tfrac{\partial \tau}{\partial t} \\ &= \tfrac{\partial r}{\partial t} + c\tfrac{\partial r}{\partial x} - \tfrac{\partial r}{\partial \tau}. \end{aligned}$$

Thus for $r(x,t,\tau)$ and $l(x,t,\tau)$,

$$\begin{aligned} \frac{\partial r}{\partial t} + c\frac{\partial r}{\partial x} - \frac{\partial r}{\partial \tau} &= F_r(\rho, x, t, \tau) \\ \frac{\partial l}{\partial t} - c\frac{\partial l}{\partial x} - \frac{\partial l}{\partial \tau} &= F_l(\rho, x, t, \tau). \end{aligned}$$

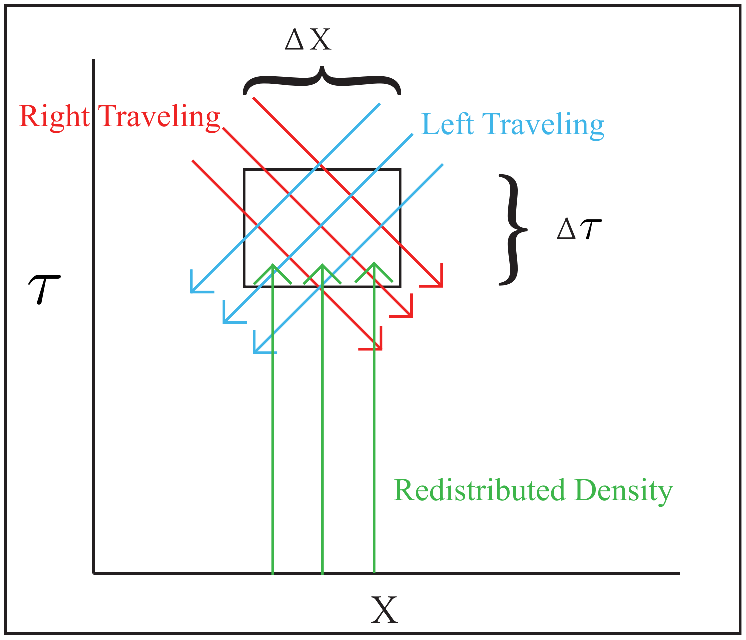

Figure 2: A pictorial view of the movement of the right-and left-moving densities in (x, τ).

The redistribution function, F, can take many forms that reflect the different ways that a swarm might behave around itself or food sources. In the simplest case, where the swarm is a group of unbiased random walkers,

$$F(x,t,\tau) = \frac{k\rho(x,t,0)}{2}\tau^{-\mu}, \text{ if } \tau > \tau_0.$$

where $1 < \mu < 3$ is a parameter approximated from biological literature and τ_0 allows the distribution to be normalizable. Attraction to food sources causes further dependence on x and interaction with other flies adds dependence on the Morse potential,

$$M(x,t) = \int_{-\infty}^{\infty}\int_{0}^{\infty} \rho(x',\tau')d\tau' dx'.$$

The inclusion of the dependence on space and density make the PDE system non-linear and suitable for numerical investigation.

3 Finite-Difference Approximations

I attempted to use finite-difference approximations to solve my system of governing equations in two ways, explicitly and semi-implicitly, which were implemented in C and MAT-

LAB, respectively. Neither of these techniques worked sufficiently well to warrant treatments of the advanced cases of the model, so for all simulations, $F(k) = c\tau_k^{-\mu}$, where $\mu = 2.32$. For the explicit version the integration used first order forward difference in time and center difference in x and τ. For example, if $r^n_{i,k}$ denotes the approximation of the right-moving density at $(x, t, \tau) = (x_i, t_n, \tau_k)$ and $F(k) = F(x, t, \tau_k)$, then

$$r^{n+1}_{i,k} = \frac{\Delta t}{2\Delta x}(r^n_{i-1,k} - r^n_{i+1,k}) + \frac{\Delta t}{2\Delta \tau}(r_{i,k+1} - r_{i,k-1}) + \frac{\Delta t}{2}F(k)(l^n_{i,0} + r^n_{i,0}) + r^n_{i,k}.$$

The boundary conditions were periodic in the x-direction and Dirichlet in τ and the total density in space and time, $\rho(x, t)$ is approximated by,

$$\rho^n_i = \sum_{k=0}^{M-1} r^n_{i,k} + l^n_{i,k}.$$

I solved for this approximation on the domain defined by,

$$-L < x < L,$$
$$0 < \tau < \tau_{max},$$

and forward from $t = 0$ to T. I found this method to be highly unstable, almost regardless of the values of $\Delta x, \Delta\tau$, and $\Delta\tau$. A sample of the output from this method is included in Figure 3. The semi-implicit solver does not do much better than the explicit one. To implement this solver, I used first order backward difference in time and again used first order central difference in (x, τ). After dicretization, I solved the system defined by,

$$r^n_{i,k} + \frac{c\Delta t}{2\Delta x}(r^n_{i+1,k} - r^n_{i-1,k}) - \frac{\Delta t}{2\Delta \tau}(r^n_{i,k+1} - r^n_{i,k-1}) = \Delta t F(k)(l^{n-1}_{i,0} + r^{n-1}_{i,0}) + r^{n-1}_{i,k},$$

using sparse matricies in MATLAB. In addition, I extend the domain in the τ direction to cover the entire range,

$$-\tau_{max} < \tau < \tau_{max},$$

because of the essentially open boundary condition at the $\tau = 0$ line. I hoped that this would clear up some of the problems with instability I saw in the explicit method, due to the large-magnitude, forced, derivatives near $\tau = 0$. The lower $\tau = -\tau_{max}$ line kept the Dirichlet boundary condition. The MATLAB method allowed for the output of the density in (x, τ) surface plots like the one in Figure 3. The visualization offered by the MATLAB software allowed me to see instability that I did not see in the explicit case, as I had full view of the output in (x, τ). The later frames in Figure 5 show the growing instabilities trailing behind the two peaks, left on the characteristics. These instabilities might have existed in the explicit case as well, but were not seen because of the way the data was extricated from the simulation. This fact does not mean much to the final evaluation of the explicit method, as it is bad enough in the x-direction, but instead speaks toward the need to consider each calculated point for instability.

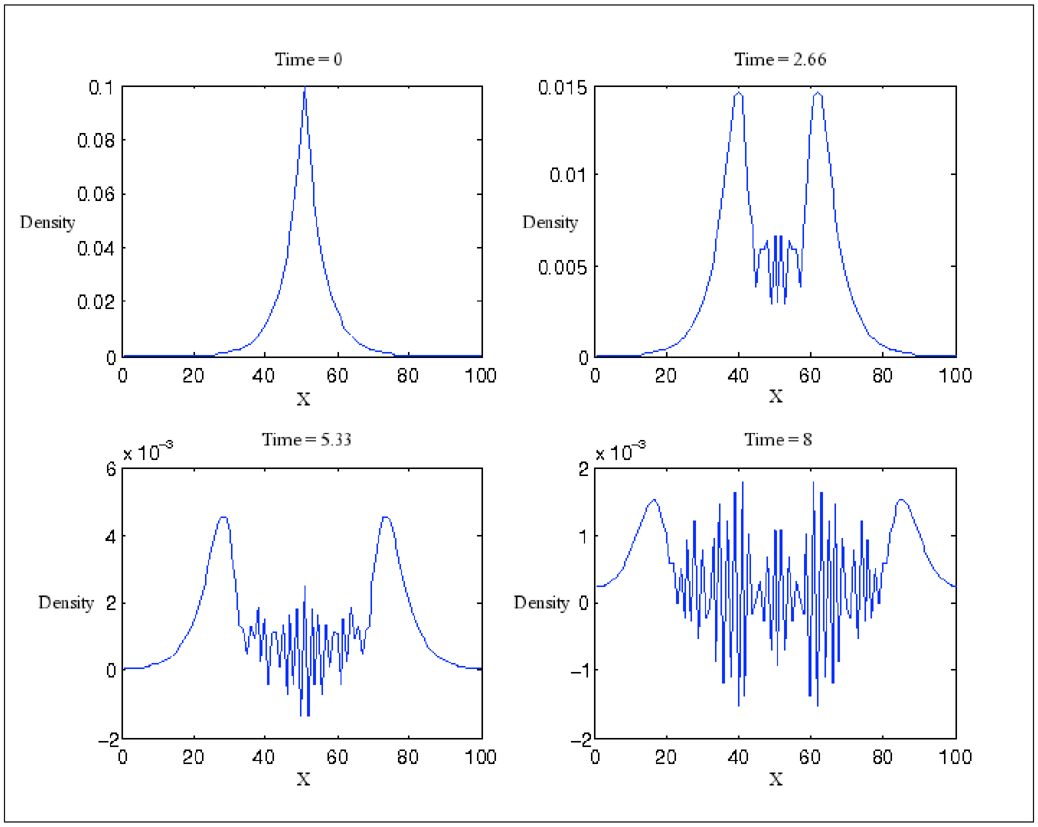

Figure 3: The time series above shows the quickly developing instabilities faced by the explicit computation. These approximations were calculated for 100 pts. in x where $-10 < x < 10$, 100 pts. for $0 < \tau < 10$, and $t < 8$ with 8000 time steps. Altering the values of $\Delta x, \Delta\tau, and \Delta t$ does little more than delay the onset of the instabilities.

4 Conclusion

I derived a two-dimensional hyperbolic conservation law as the continuum limit of a formerly stochastic model. Efforts to solve the resulting system of equations using finite-difference methods were ineffective. The approximations, in both cases, became unstable quickly in the simplest formulations of the model and the scales needed to compare those approximations to the output of the Lagrangian model are too computationally intensive for this project. This system of equations should probably be approximated using a different technique. [**?**] outlines two techniques for such a system: dimensional splitting and finite volumes. The software package, CLAWPACK (http://www.amath.washington.edu/~claw/), distributed by the University of Washington, looks like a promising way to implement finite

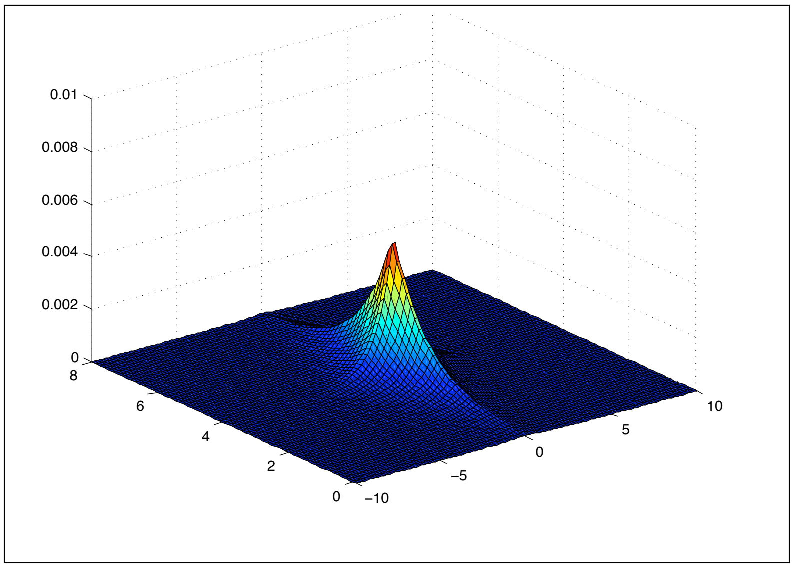

Figure 4: This figure shows the Gaussian initial condition, $\rho(x, 0, \tau)$, for $L = 10$, $\tau_{max} = 8$, and $\Delta t = .005$. Computationally, the domain extends to $\tau = -8$ to absorb the mass that is fluxed through $\tau = 0$. This area is omitted from the plots. There were 80 points in the x and τ directions for this simulation.

volume schemes on problems like this one.

5 appendix

The code that I used to run the simulations for this project.

5.1 Explicit Method

```
#include <stdio.h>
#include <math.h>

int main (int argc, const char * argv[]) {
/*Declare a bunch of variables*/
int N, M,STEPS,i,j,k;
double L,TAU,TIME,dt,dtau,dx,lam1,lam2,sum,sum2;

FILE *fp1,*fp2;

fp1 = fopen("output.dat","w+");
```

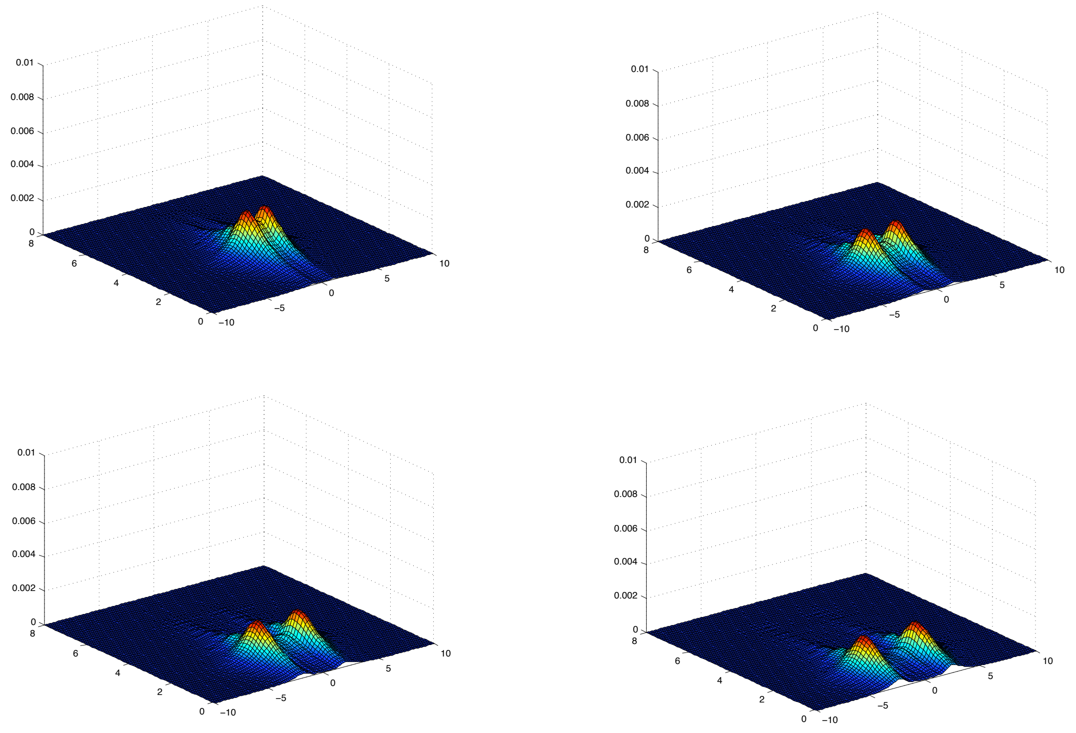

Figure 5: This figure shows the evolution of the semi-implicit approximation whose initial condtion is shown in 3. The times shown are [left to right, top to bottom] 1.0, 1. 5, 2.0, 2.5 sec. The behavior of the approximation makes intuitive sense, as the left- and right-moving groups travel away from the initial condition and begin to be redistributed. Note, however, the instability starting to trail behind (in τ) the two peak densities. Not long after the fourth frame, these instabilites grow to dominate the approximated solution.

```
fp2 = fopen("debug.dat","w+");
/*Set-up the computational domain: limits, steps, and step-size*/
L = 100;
TAU = 20;
TIME = 2;

N = 2000; //Steps in space
M = 100; //Steps in state
STEPS = 200000; //timesteps

dx = 2*L/N;
```

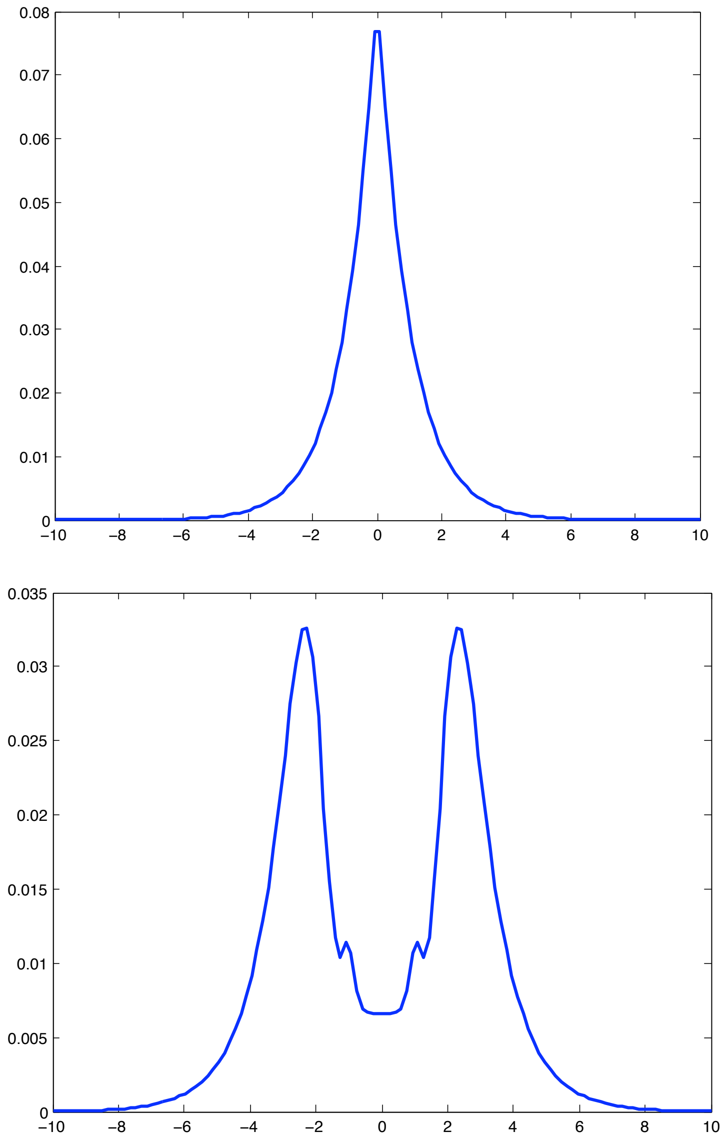

Figure 6: This figure shows the same collpased view that the explicit code would output, for the calculation shown fully in Fig. 5, at times $t = 0$ and 2.5s (top and bottom). A small trailing peak is visible behind each larger peak, but the instability which developed behind the large peaks, through values of τ is not visible.

```
dtau = TAU/M;
dt = TIME/STEPS;
```

```
lam1 = dt/(2*dx);
lam2 = dt/(2*dtau);
/* Declare the stuff we want to calculate*/
double x[N],tau[M],r[N][M],l[N][M],nr[N][M],nl[N][M],Pt[M],Px[N];

/**********************************************

Intialize r[N][M] and l[N][M] as well as the position vectors x[N] and tau[M]

**********************************************/
for(i = 0 ; i < N ; i++){
x[i] = i*dx - L;
}
for(k = 0 ; k < M ; k++){
tau[k] = k*dtau;
fprintf(fp2,"%f ",tau[k]);
}
fprintf(fp2,"\n");
//These output the correct things.
//printf("%f %f ; %f %f\n",x[0],x[99],x[100],x[199]);
//printf("%f %f ; %f %f\n",tau[0],tau[99],tau[100],tau[199]);

/*************************************

Fill the probability vector for tau transition

*************************************/
sum = 0.0;
for(k = 0 ; k < M-1 ; k++){
  if(tau[k]>1){
        Pt[k] = pow(tau[k],-2.32);
sum = sum + Pt[k];
  }
  else{
    Pt[k] = 0;
  }
}
Pt[M-1] = 0;
for(k = 0 ; k < M ; k++){
Pt[k] = Pt[k]/sum;
```

```
}
sum = 0.0;
for(k = 0 ; k < M ; k++){
sum = sum + Pt[k];
}
printf("Sum is ... %f\n",sum);

/*********************************************

Fill the probability vector for initial condition

 *********************************************/
sum2 = 0.0;
for(i = 0 ; i < N ; i++){
Px[i] = exp(-fabs(x[i])/4);
sum2 = sum2 + Px[i];
}
for(i = 0 ; i < N ; i++){
Px[i] = Px[i]/sum2;
}
sum2 = 0.0;
for(i = 0 ; i < N ; i++){
sum2 = sum2 + Px[i];
}
//printf("sum2 is ... %f\n",sum2);

/***********************************************
* Finally, we initialize
***********************************************/
for(i = 0 ; i < N ; i++){
for(k = 0 ; k < M ; k++){
l[i][k] = 0.5 * Pt[k]*Px[i];
r[i][k] = 0.5 * Pt[k]*Px[i];
}
}

for(k = 0 ; k < M ; k++){
  sum = 0;
  for(i = 0; i < N ; i++){
    sum = sum + l[i][k] + r[i][k];
  }
  fprintf(fp2,"%f ",sum);
}

/**********************************************
* Do some solving
**********************************************/
for(j = 0 ; j < STEPS ; j++){//I build a loop in time
  if(j%(STEPS/10)==0){
    printf("On time Step %d, of %d.\n",j,STEPS);
  }
for(i = 0 ; i < N ; i++){//I build a loop in space
for(k = 0 ; k < M ; k++){//I build a loop in state
if(k==0&&i==0){
//Implement Corner BC
//printf("got here\n");
nl[0][0] = l[1][1];
nr[0][0] = r[N-1][0];
//printf("left corner assigned");
}
else if(k==0&&i==(N-1)){
nl[N-1][0] = l[0][1];
nr[N-1][0] = r[N-2][1];
//printf("right bottom corner assigned\n");
}
else if(k==0){
//printf("got here (1,0)\n");
nl[i][k] = l[i+1][1];
nr[i][k] = r[i-1][1];
}
else if(k==M-1){
nl[i][k] = 0;
nr[i][k] = 0;
}
else if(i==0&&k>0&&k<M-1){
//printf("%d - got here\n",k);
nr[i][k] = lam1*(r[N-1][k]-r[i+1][k]) + lam2*(r[i][k+1]-r[i][k-1]) + dt*0.5*Pt[k]*(l[i][0]+
nl[i][k] = lam1*(l[i+1][k]-l[N-1][k]) + lam2*(l[i][k+1]-l[i][k-1]) + dt*0.5*Pt[k]*(l[i][0]+
}
else if(i==N-1&&k>0&&k<M-1){
nl[i][k] = lam1*(l[0][k]-l[i-1][k]) + lam2*(l[i][k+1]-l[i][k-1]) + dt*0.5*Pt[k]*(l[i][0]+r[
nr[i][k] = lam1*(r[i-1][k]-r[0][k]) + lam2*(r[i][k+1]-r[i][k-1]) + dt*0.5*Pt[k]*(l[i][0]+r[
//printf("%d - got here\n",k);
}

else{
//printf("big time\n");
nl[i][k] = lam1*(l[i+1][k]-l[i-1][k]) + lam2*(l[i][k+1]-l[i][k-1]) + dt*0.5*Pt[k]*(l[i][0]+
//printf("new left assigned at (%d,%d)\n",i,k);
nr[i][k] = lam1*(r[i-1][k]-r[i+1][k]) + lam2*(r[i][k+1]-r[i][k-1]) + dt*0.5*Pt[k]*(l[i][0]+
//printf("new right assigned at (%d,%d)\n",i,k);
}
}//I close a loop in state
}//I close a loop in space
//printf("got to the update stage\n");
for(i = 0 ; i < N ; i++){\
//printf("updating %d\n",i);
for(k = 0 ; k < M ; k++){
//printf("%d ",k);
l[i][k] = nl[i][k];
r[i][k] = nr[i][k];
}
}

if((j%(STEPS/10))==0){
/********************
  I Print the current total density
*********************/
for(i = 0 ; i < N ; i++){
sum = 0.0;
for(k = 0 ; k < M ; k++){
sum = sum + l[i][k] + r[i][k];
}
if(i==N-1){
fprintf(fp1,"%f\n",sum);
}
else{
fprintf(fp1,"%f ",sum);
}
}
}
}//I close a loop in time
```

```
return 0;
}
```

I used MATLAB to plot the information in output.dat and debug.dat

5.2 Implicit Method

```
%buggies.m
%
%
% Hopefully, solves the continuum case of my thesis model

% Set up the computational domain
N = 80;
M = 80;
STEPS = 200;

% Set up the domain to be simulated on
Length  = 20;
Tau     = 20;


dx      = (2*Length)/(N-1);
dtau    = (2*Tau)/(M-1);

dt      = .01;
Time    = STEPS*dt;


X       = [0:N-1]*dx - Length;
T       = [0:M-1]*dtau - Tau;

lam1    = 1;
lam2    = dt/(2*dx);
lam3    = dt/(2*dtau);

%%%%%
% Set up the probability function and initial condition
%%%%%
```

```
nr      = sparse(zeros(N*M,1));
nl      = sparse(zeros(N*M,1));
%l       = sparse(zeros(N*M,1));
%r       = sparse(zeros(N*M,1));

Px      = zeros(N,1);
Function = zeros(N*M,1);

for i = 1:M
    if T(i) > 1
        Pt(i) = T(i)^-2.32;
        %iPt(i) = exp(-abs(T(i)-(Tau/2)));
    else
        Pt(i) = 0;
        %iPt(i) = 0;
    end
end
Pt = Pt/sum(Pt(2:M-1));
iPt = exp(-abs(T-(Tau/2)));
iPt = iPt/sum(iPt);

Px = exp(-abs(X));
Px = Px/sum(Px);

%%%
% Build the Derivative matricies and the initial condition
% The derivative matrices are Ar and Al and the initial conditions
% are in r and l
%%%

%iPt = ones(1,M)*(1/M);

test = iPt'*Px;
r = sparse(.5*reshape(test(2:(M-1),:),N*(M-2),1));
l = r;

subdiag  = mod(1:N*(M-2),M-1)>0;
supdiag  = mod(0:N*(M-2)-1,M-1)>0;
subdiag2 = ones(1,N*(M-2));
supdiag2 = subdiag2;
subdiag3 = [ones(M-2,1)' zeros((N-1)*(M-2),1)'];
```

```
supdiag3 = subdiag3;

rdiagnols = [lam2*subdiag2' -lam2*subdiag2' lam3*subdiag' lam1*ones(N*(M-2),1) -lam3*supdia
Ar = spdiags(rdiagnols,[-(N-1)*(M-2) 2-M -1 0 1 M-2 (N-1)*(M-2)] ,N*(M-2),N*(M-2));

ldiagnols = [-lam2*subdiag2' lam2*subdiag2' lam3*subdiag' lam1*ones(N*(M-2),1) -lam3*supdia
Al = spdiags(ldiagnols,[-(N-1)*(M-2) 2-M -1 0 1 M-2 (N-1)*(M-2)] ,N*(M-2),N*(M-2));


%%%%
% Solve! the equations nr*Ar = F(r) + lam1*r and corresponding for left
%%%%
matrix = [zeros(1,N) ; reshape(r+l,M-2,N); zeros(1,N)];
surf(X,T((M/2):M),matrix((M/2):M,:));
pause;
output_matlab = [];
for t = 1:STEPS

    %%%%%
    % Calculate the forcing function to be used explicitly
    %%%%%

    Function = [];
     sum1 = 0;
     for i = 0:(N-1)
         sum1 = sum1 + r((i+.5)*(M-2))+l((i+.5)*(M-2));
         Function = vertcat(Function, .5*Pt(2:M-1)'*(r((i+.5)*(M-2))+l((i+.5)*(M-2))));
     end
     sum1 = sum1;
     distributed = sum(Function);
     percent_loss = abs(sum1 - 2*distributed)/sum1;
     if abs(sum1 - 2*distributed)/sum1 > 1e-8
         fprintf('Error in Redistribution!');
         break;
     end

    %%%%%
    % Solve!
    %%%%%
    nr = Ar \ r;%(dt*Function + lam1*r);
    nl = Al \ l;%(dt*Function + lam1*l);
    r = nr;
    l = nl;
    if mod(t,STEPS/10) == 0
    t
    matrix = [zeros(1,N) ; reshape(r+l,M-2,N); zeros(1,N)];
    surf(X,T((M/2):M),matrix((M/2):M,:));
    pause;
    output_matlab = vertcat(output_matlab,sum(matrix((M/2):M,:)));
    end
end % I end the time stepping
%progressbar(h,-1);
```