Solution to the Final Project

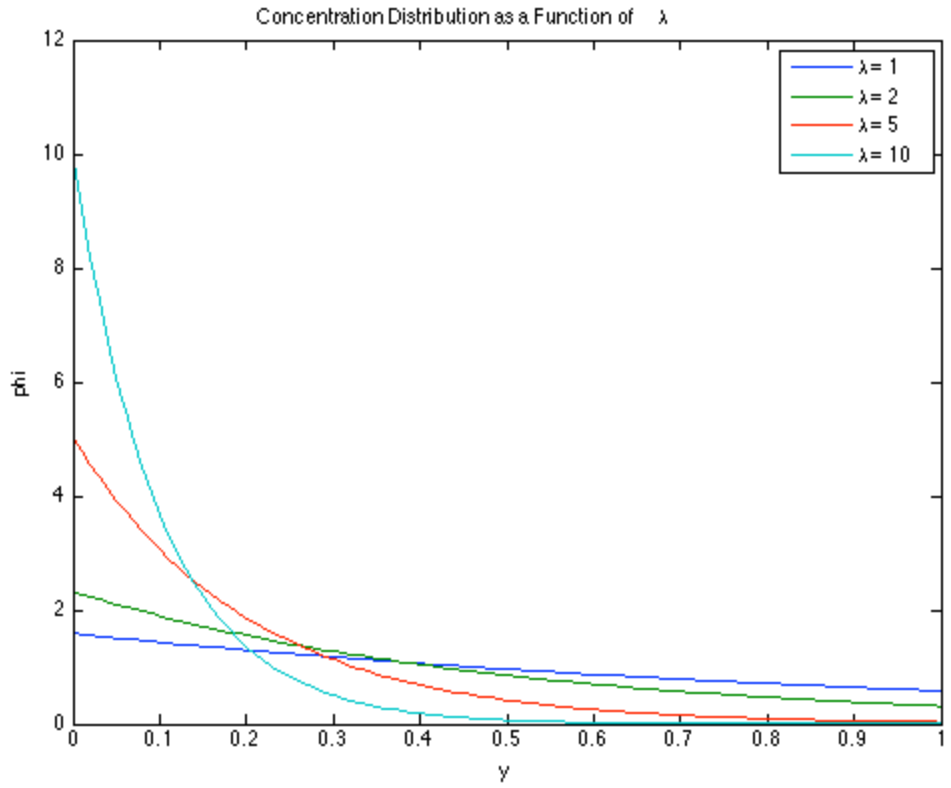
In this script we analyze the behavior of Electrical Field Flow Fractionation, a technique developed to separate large molecules and small particles based on the ratio of their electrophoretic mobility to molecular diffusivity. A transverse electric field is applied across a channel, creating a concentration polarization layer that is a function of this ratio. This segregation is coupled with a pressure-driven convective flow in the x-direction. Because the velocity is a function of position, the average solute velocity will be a function of lambda, this ratio. Different species will thus move with different average velocities, resulting in separation. In addition, however, because not all molecules of the same type have the same velocity at any instant, there will also be Taylor dispersion in the flow direction. Thus, the channel length (or separation time) will depend on the ratio of the difference of average velocities to the Taylor dispersivities. We shall examine this phenomenon by means of a Brownian Dynamics simulation of the diffusive process, and conclude with a simulation of a difficult separation.

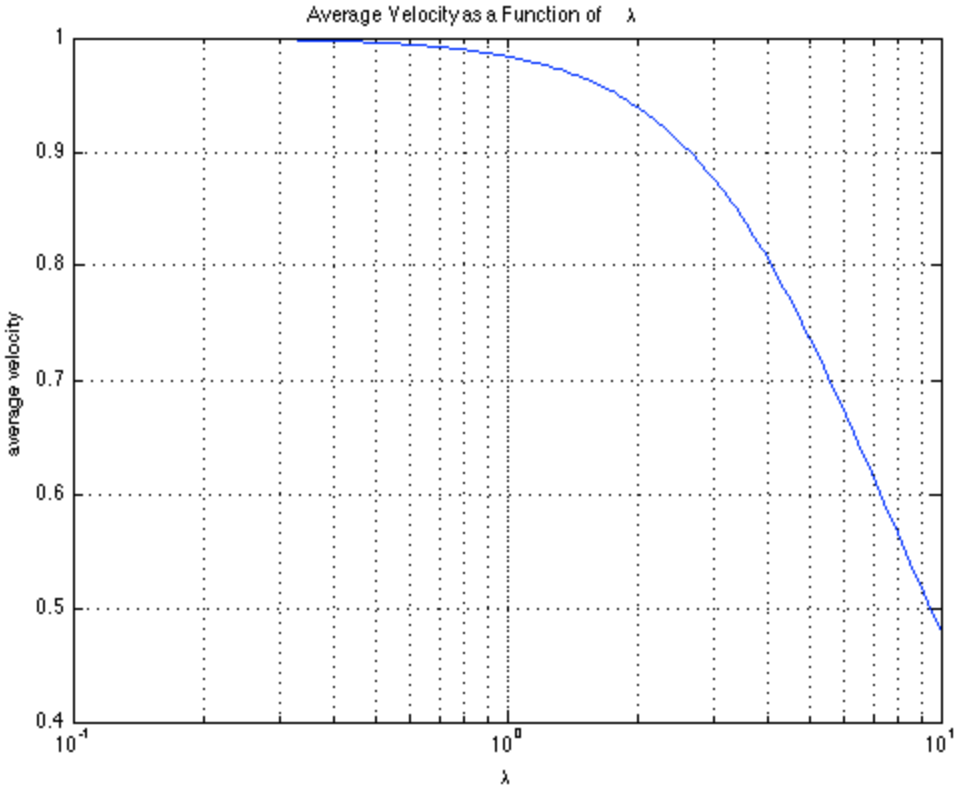
Contents

Part 1: The Steady-State Profile:

The steady-state profile can be integrated analytically, as can the average velocity, as a function of \lambda. Here we do this using both pencil and paper, as well as using wolfram alpha for the velocity. An ipod with wolfram alpha installed is a pretty cool toy... Thus:

```
phi = @(y,lambda) lambda/(1-exp(-lambda))*exp(-lambda*y);

ybar = @(lambda) 1.0./lambda + 1.0./(1-exp(lambda));

uxbar = @(lambda) 6*(exp(lambda).*(lambda-2)+lambda+2)./lambda.^2.0./(exp(lambda)-1)

u = @(y) 6*y.*(1-y);

%Now we plot these up for the requested values:

y=[0:.01:1];
figure(1)
plot(y,phi(y,1),y,phi(y,2),y,phi(y,5),y,phi(y,10))
xlabel('y')
ylabel('phi')
legend('\lambda = 1','\lambda = 2','\lambda = 5','\lambda = 10')
title('Concentration Distribution as a Function of \lambda')

lambda=10.0.^[-1:.01:1];
figure(2)
semilogx(lambda,uxbar(lambda))
grid on
xlabel('\lambda')
ylabel('average velocity')
title('Average Velocity as a Function of \lambda')
```


Concentration Distribution as a Function of λ
λ = 1
λ = 2
λ = 5
λ = 10
phi
y
0
2
4
6
8
10
12
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Average Velocity as a Function of λ
average velocity
λ
0.4
0.5
0.6
0.7
0.8
0.9
1
10^-1
10^0
10^1

Part 2: Transient Evolution in the Y Direction:

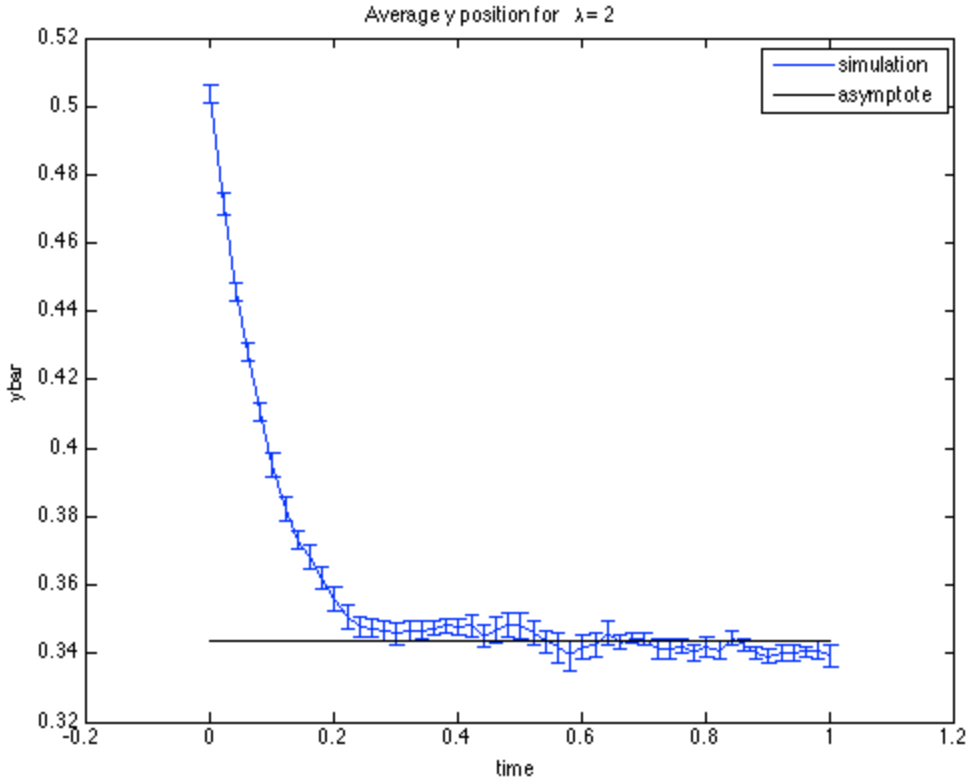
We simulate the transient evolution in the y-direction for a specific value of lambda. We use a random walk in the y-direction and reflect off of walls at y = 0 and y = 1. Steady-state is achieved pretty quickly as it is the shorter of either the convection time (1/lambda) or the diffusion time (which is order one, but numerically fairly small). Thus, going out to a t of unity is more than sufficient.

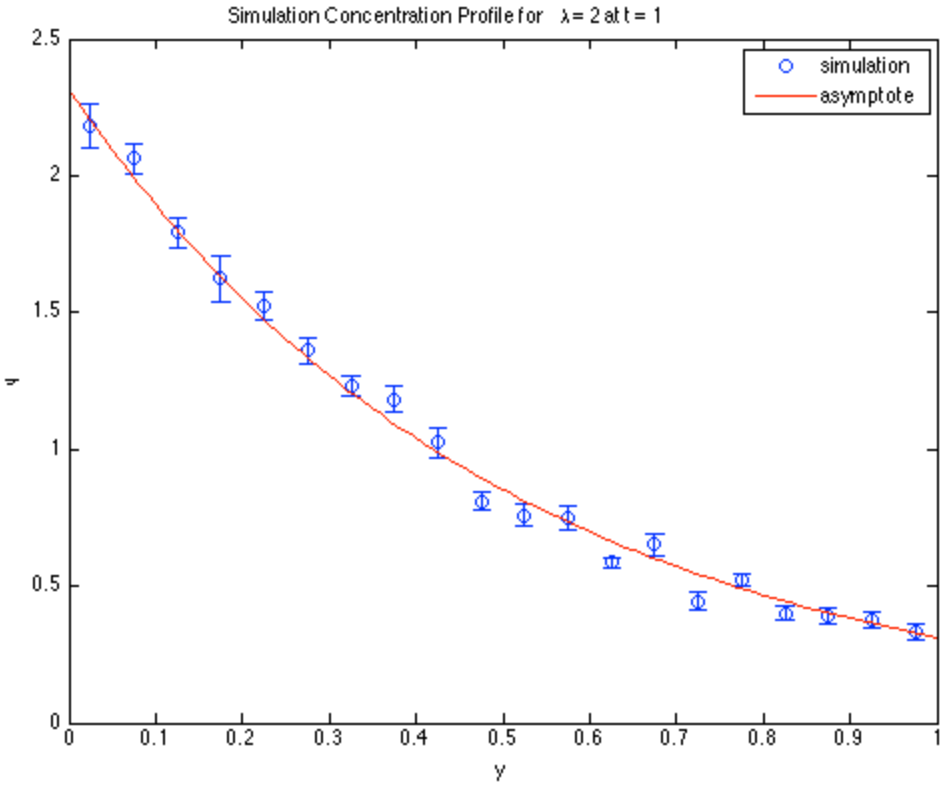
We want to keep track of the average y-position, as well as to keep track of the standard deviation. This is done using the technique of undersampling: we solve for an array of y values, producing m realizations of n particle simulations.

We shall finish this part off by plotting up the histogram of the concentration profile and comparing that to the analytical distribution from part 1. Thus:

```
n = 1000;
m = 10;
y = rand(n,m); %We have m columns of n particle positions!

dt = 10^-4; %We choose a time step so that the displacement is "fairly" small.

lambda = 2; %The value of lambda we are working on

t = [0:dt:1]; %The array of times.
ybarsim = zeros(size(t)); %We initiallize the array.
sigybarsim = zeros(size(t)); %We initiallize the array.
ubarsim=zeros(size(t)); %We initialize an array we need for part 3.
sigubarsim = zeros(size(t)); %Again, an array for part 3.

avey = mean(y); %This is a row vector containing the mean values of each realizatior
ybarsim(1) = mean(avey); %The average of all the simulations.
sigybarsim(1) = std(avey)/m^.5; %The error in the mean of the realizations.

aveu = mean(u(y)); %A row vector containing the mean values of u
ubarsim(1) = mean(aveu);
sigubarsim(1) = std(aveu)/m^.5;

% And now we do it for all the times!
for i = 2:length(t)
    dy = -lambda*dt + (2*dt)^.5*randn(size(y));
    y = y + dy;

    ib = find(y<0);
    y(ib) = abs(y(ib)); %We reflect off the bottom

    it = find(y>1);
    y(it) = 2-y(it); %We reflect off the top

    avey = mean(y); %This is a row vector containing the mean values of each realiza
    ybarsim(i) = mean(avey); %The average of all the simulations.
    sigybarsim(i) = std(avey)/m^.5; %The error in the mean of the realizations.

    aveu = mean(u(y)); %A row vector containing the mean values of u
    ubarsim(i) = mean(aveu);
    sigubarsim(i) = std(aveu)/m^.5;

end
```

```
% And we plot it up:

figure(3)
%We use every 200th point for clarity...
errorbar(t(1:200:end),ybarsim(1:200:end),sigybarsim(1:200:end))
hold on
plot([0,max(t)],ybar(2)*ones(1,2),'k')
hold off
legend('simulation','asymptote')
xlabel('time')
ylabel('ybar')
title(['Average y position for \lambda = ',num2str(lambda)])

%We also want to plot up the distribution in y.  We use the histogram
%command:
nb=20; %We use 20 bins.
edges=[0:1/nb:1];
yc=(edges(1:end-1)+edges(2:end))/2; %The centers of the bins
phis=histc(y,edges);
phis = phis(1:nb,:)*nb/n; %We convert to concentration
phisave=mean(phis')'; %We get the mean values (it works along the columns)
phisstd=std(phis')'/m^.5; %The standard deviation in the mean

%And we plot it up:
figure(4)
yp=[0:.01:1];
errorbar(yc,phisave,phisstd,'o')
hold on
plot(yp,phi(yp,2),'r')
hold off
xlabel('y')
ylabel('\phi')
legend('simulation','asymptote')
title(['Simulation Concentration Profile for \lambda = ',num2str(lambda),' at t = ',
```

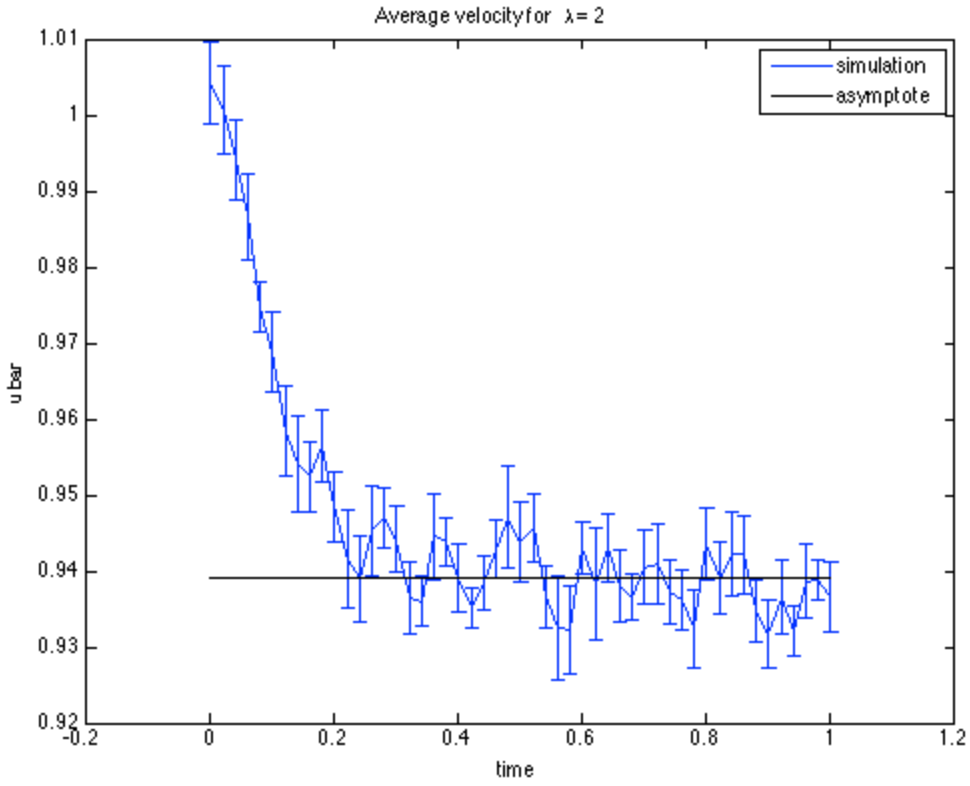




Part 3: Displacement in the X-Direction:

We now calculate the average velocity in the x-direction. To be more efficient, we calculated this in the simulation above. Thus, here we shall simply plot it up!

```
figure(5)
%We use every 200th point for clarity...
errorbar(t(1:200:end),ubarsim(1:200:end),sigubarsim(1:200:end))
hold on
plot([0,max(t)],uxbar(2)*ones(1,2),'k')
hold off
legend('simulation','asymptote')
xlabel('time')
ylabel('ubar')
title(['Average velocity for \lambda = ',num2str(lambda)])
```



Part 4: Taylor Dispersivity:

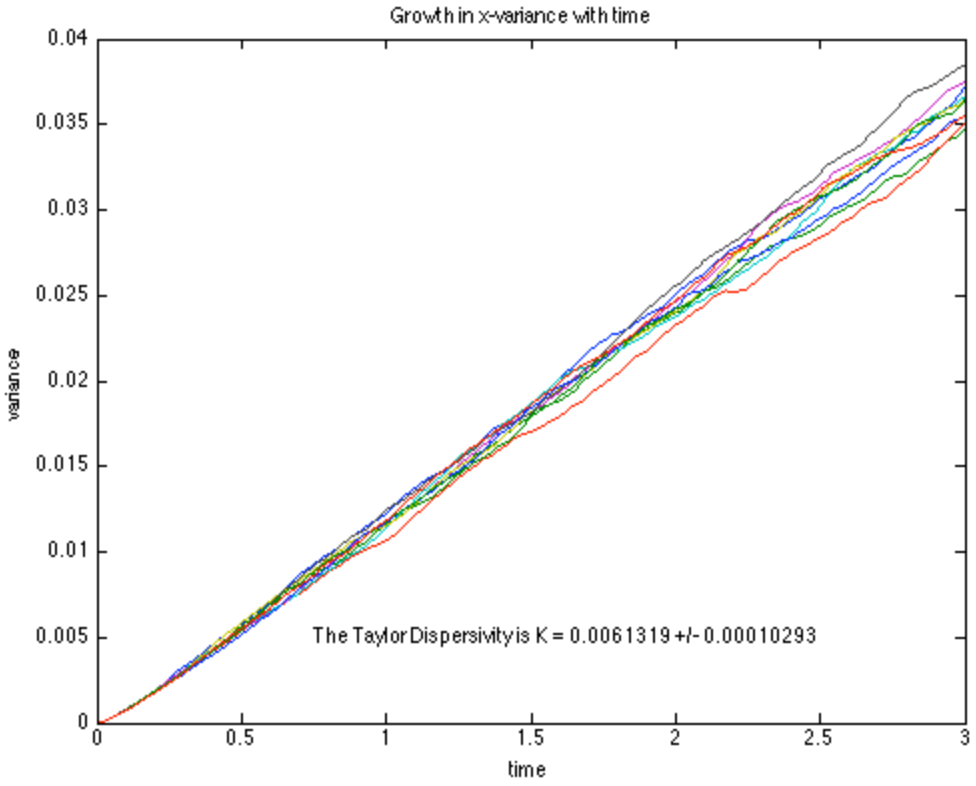
Now we actually simulate the motion in the x-direction and use that to calculate the Taylor Dispersivity. This dispersion coefficient is simply half the growth rate of the variance in the x-direction over time. Thus, we need to track both x and y locations. We will do the simulation over a longer period so that we are sure that the initial transient has passed (a value of t* of 1 or so). We calculate the variance in the x-direction for each realization of the simulation and fit a line to the data. This yields the slope, and in turn gives us K*. We can cut and paste the code from part 2:

```
n = 1000;
m = 10;
y = rand(n,m); %We have m columns of n particle positions!
x = zeros(size(y)); %The initial x-locations
```

```
dt = 10^-4; %We choose a time step so that the displacement is "fairly" small.

lambda = 2; %The value of lambda we are working on

tmax = 3;
t = [0:dt:tmax]; %The array of times.

varxsim=zeros(length(t),m); %We initialize an array of variances.

varxsim(1,:) = var(x);

% And now we do it for all the times!
for i = 2:length(t)
    dy = -lambda*dt + (2*dt)^.5*randn(size(y));
    ynew = y + dy;

    ib = find(y<0);
    ynew(ib) = abs(ynew(ib)); %We reflect off the bottom

    it = find(ynew>1);
    ynew(it) = 2-ynew(it); %We reflect off the top

    dx = (u(y)+u(ynew))/2*dt; %Trapezoidal rule
    dx(ib) = dx(ib)*2/3; %Adjustment for wall reflection
    dx(it) = dx(it)*2/3; %The same for the other wall

    x = x+dx; %We update the positions!
    y = ynew;

    varxsim(i,:) = var(x); %The variances for each set of points.
end

% OK, now we plot this up:

figure(6)
plot(t,varxsim)
xlabel('time')
ylabel('variance')
title('Growth in x-variance with time')

% This looks pretty linear for t greater than one.  Let's use these values
% to get K*.  Each column will yield a different estimate for K, letting us
% calculate the random error in the final value.

ikeep = [1/dt:length(t)]; %The times we keep for calculating K.

a = [t(ikeep)',ones(length(ikeep),1)];
xfit = a\varxsim(ikeep,:);
k = xfit(1,:)/2; %These are the realizations of k.
kave = mean(k); %The average value
kstd = std(k)/m^.5; %The error in the mean

kstring=['The Taylor Dispersivity is K = ',num2str(kave),' +/- ',num2str(kstd)];
disp(kstring)
```

```
figure(6)
text(0.75,.005,kstring)
```

```
The Taylor Dispersivity is K = 0.0061319 +/- 0.00010293
```



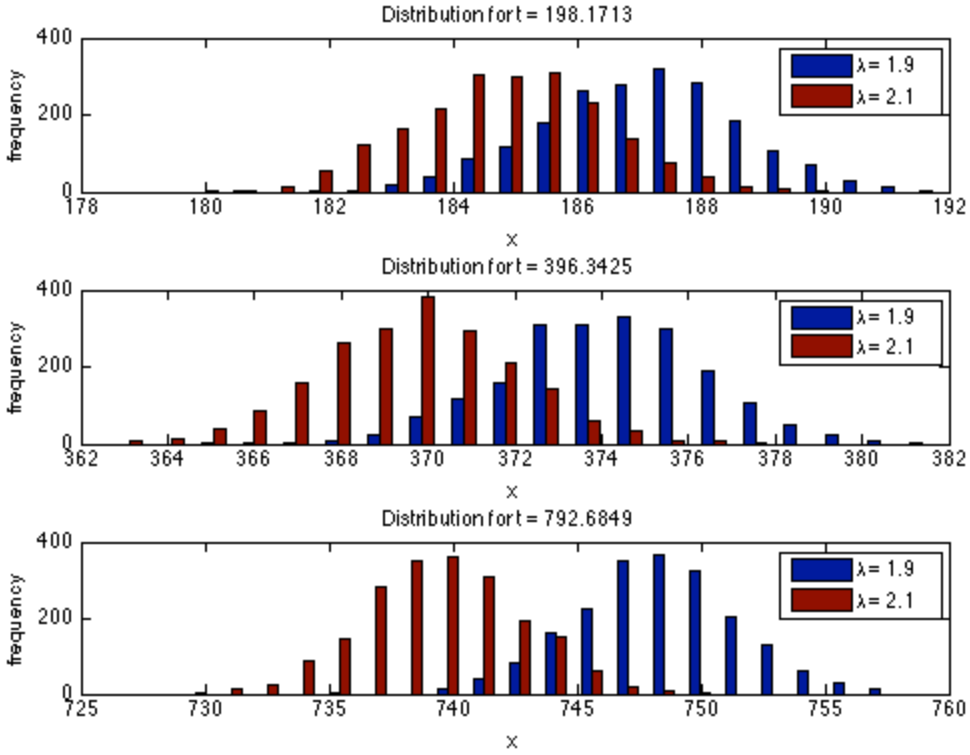
Part 5: Simulation of the Separation:

We can calculate the time necessary for the simulation from the variation in velocity with lambda, and from the magnitude of the dispersivity. The expected separation time is given by the equation in the project statement, and we shall do simulations for half this time, this value, and twice this time. The time is rather long, so we shall reduce the number of particles we track so that it goes much faster. Otherwise, we simply use the code from part 4 over again.

```
texp=8*kave/(uxbar(2.1)-uxbar(1.9))^2

n = 2000;

lambdaall=[1.9 2.1]; %The two values of lambda we want to work on.
m = length(lambdaall); %We use one array for each species.

dt = 10^-4; %We choose a time step so that the displacement is "fairly" small.

y = rand(n,m); %We have m columns of n particle positions!
x = zeros(size(y)); %The initial x-locations

t = 0;
```

```
for j = 1:3
    while t<texp/4*2^j
        dy = -ones(n,1)*lambdaall*dt + (2*dt)^.5*randn(size(y));
        ynew = y + dy;

        ib = find(y<0);
        ynew(ib) = abs(ynew(ib)); %We reflect off the bottom

        it = find(ynew>1);
        ynew(it) = 2-ynew(it); %We reflect off the top

        dx = (u(y)+u(ynew))/2*dt; %Trapezoidal rule
        dx(ib) = dx(ib)*2/3; %Adjustment for wall reflection
        dx(it) = dx(it)*2/3; %The same for the other wall

        x = x+dx; %We update the positions!
        y = ynew;

        t = t+dt;
    end
    %Now we grab the histogram of the different species at the jth time!
    figure(7)
    subplot(3,1,j),hist(x,20)
    xlabel('x')
    ylabel('frequency')
    title(['Distribution for t = ',num2str(t)])
    legend(['\lambda = ',num2str(lambdaall(1))],['\lambda = ',num2str(lambdaall(2))]
end
```

```
texp =

  396.3424
```



Published with MATLAB® 7.12