MXB261 Problem Solving Task

Kassia Lembryk-Walsh (n11090677)

PART 1 INDEX

PART 2 INDEX

Part 1: A biased random walk

Case 1: Equal probabilities $(south = {}^1/_3, west = {}^1/_3, east = {}^1/_3)$

Since there is no northward movement, particles in Case 1 have a greater opportunity ($p = {}^2/_3$) to move along the x-axis (greater variance). However, with an equal weighting to move each direction, the further from the starting position, the less likely it is for a particle to travel there.

Case 2: Faster fall, equal x-wise probabilities $(south = {}^2/_3, west = {}^1/_6, east = {}^1/_6)$

Particles have a much higher probability of falling south than moving x-wise, meaning that particles on average will tend to land much closer to their starting position than Case 1 (lesser variance). X-wise movement is still equal, although their distance from starting position would tend to be smaller.

Case 3: Weighted westward $(south = {}^3/_5, west = {}^3/_{10}, east = {}^1/_{10})$

The probability of moving west is 3x greater than that of moving east, and therefore particles will drift westward as they fall (westward expected value).

Case 4: Weighted eastward $(south = {}^3/_5, west = {}^1/_{10}, east = {}^3/_{10})$

The probability of moving east is 3x greater than that of moving west, and therefore particles will drift eastward as they fall (eastward expected value).

Figure 1: Cases for P = 1 and N = 100

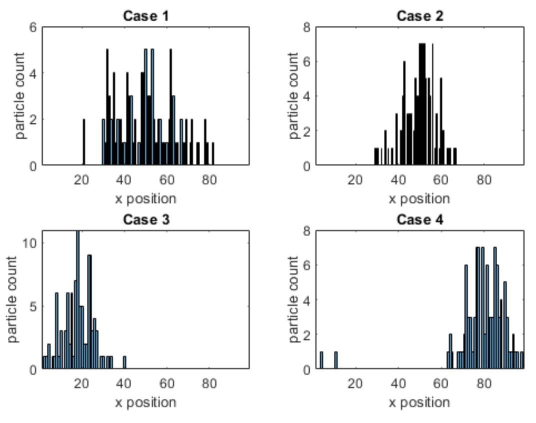
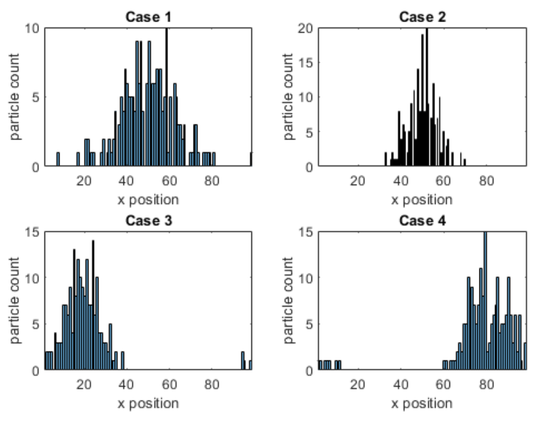


Figure 2: Cases for P = 1 and N = 200



For a **fixed starting position** of $x = 50$; Figure 1 (*above, left*) and Figure 2 (*above, right*) show normal-like distributions for all their cases. Note that the boundaries on the left and right are looping, the seemingly discontinuous heights in Cases 3 & 4 are in fact an expected part of their distribution. The least normal-like curve is Figure 1 Case 1; however, with a smaller number of histogram bins, it becomes visibly much closer to a normal distribution (a smaller sample size is more significantly affected by outliers, but the average is still normally distributed). Since the difference between them is only in the number of particles, the distributions in Figure 1 have a smaller column height and generally more outliers; but maintain the same underlying mean and variance.

- **Case 1** approximates a normal curve with $\mu = 50$ (most likely to land at starting x) and some standard deviation $\sigma = \sigma_1$ that would correspond to its south-ward probability.
- **Case 2** approximates a normal curve with $\mu = 50$ (still equal x-wise probabilities), and with a far greater chance of falling south instead of x-wise, particles will on average travel shorter distances, resulting in a smaller standard deviation, $\sigma_2 < \sigma_1$.
- **Case 3** approximates a normal curve, however with the westward weighting, the expected value is different. On average, 10 total moves will place a particle 3 units west. Expanding this over the total domain from $y = 99$ to $y = 1$, particles on average move 29.4 units west, putting the expected value at $\mu = 20.6$. This is reflected in the figures. Since the variance is directly associated with the chance of downwards movement, we can see that since $south_1 < south_3 < south_2$, therefore $\sigma_1 < \sigma_3 < \sigma_2$. From discussions with classmates, we determined this probably isn't the exact expected value since the particles can collide with each other, but with a maximum column height of 20, this is unlikely to be significant.
- **Case 4** has the same probabilities as Case 3, except weighted eastwards. As such, we can expect the average particle to move 29.4 units east from the starting position and the variance to remain equal, $\mu = 79.4$ and $\sigma_3 = \sigma_4$, which is reflected in the figures.

Figure 3: Cases for P = 'rand' and N = 100

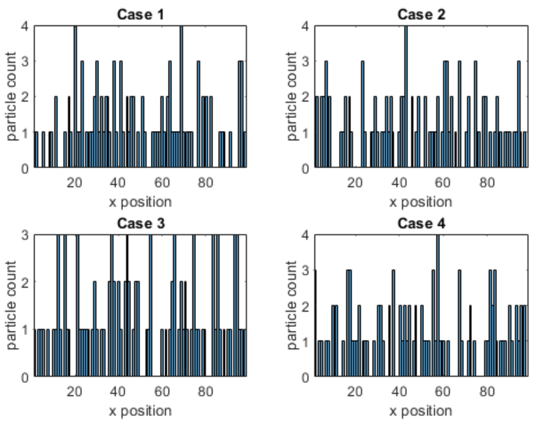
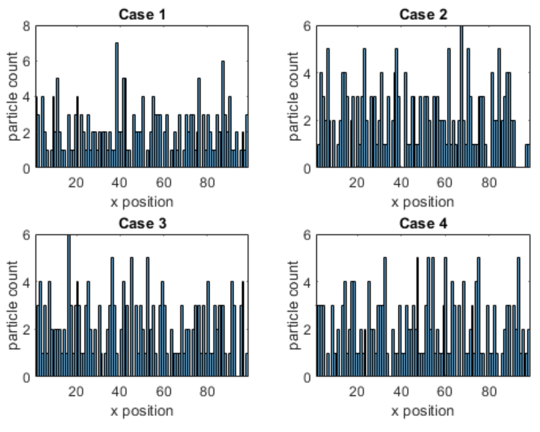


Figure 4: Cases for P = 'rand' and N = 200



Comparatively, for a **random starting position**, the figures are much less interesting. Figure 3 (*beneath paragraph, left*) and Figure 4 (*beneath paragraph, right*) appear as uniform distributions. This is expected. Although the final position of an *individual* particle might have a normal probability distribution, since the initial starting position is determined by a uniform probability distribution between 1 and 99, the chance of landing in any given end position is equally likely. Figures 3 and 4 again differ only in number of particles, reflected in higher column heights.

Part 2: Sampling from Experimental Data

As would be expected, samples generated from the probability distribution function of the given experimental data match closely to the experimental data. Since the samples were generated from a linear interpolation of the data, there might be slight inaccuracies – however, the element of randomness obscures this. As expected, the more bins in the histogram, the more Data0 and DataNew stray from similarity. This correlates well, the underlying distribution is theoretically the same, but with finer detail, randomness emphasises the outliers. This is most clearly communicated in the Kullback-Leibler measure; for 10 bins (Figure 5, *below*) the two distributions are very similar ($DKL_{Data0} = 0.0081$, and $DKL_{DataNew} = 0.0077$), becoming increasingly different as the number of bins increases. For 40 bins (Figure 7, *bottom of page*), DKL_{Data0} is 2.83x greater, and $DKL_{DataNew}$ is 2.73x greater.

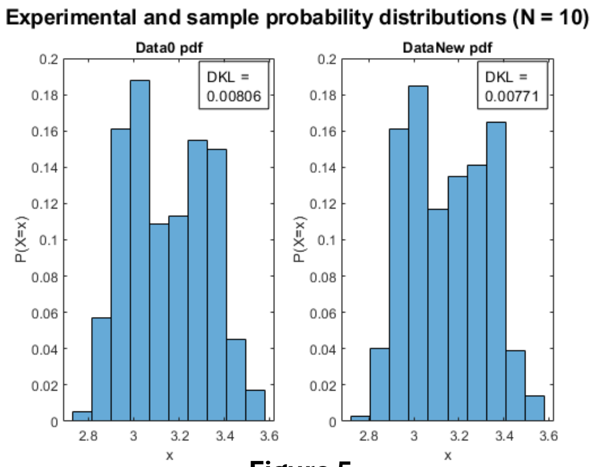


Figure 5

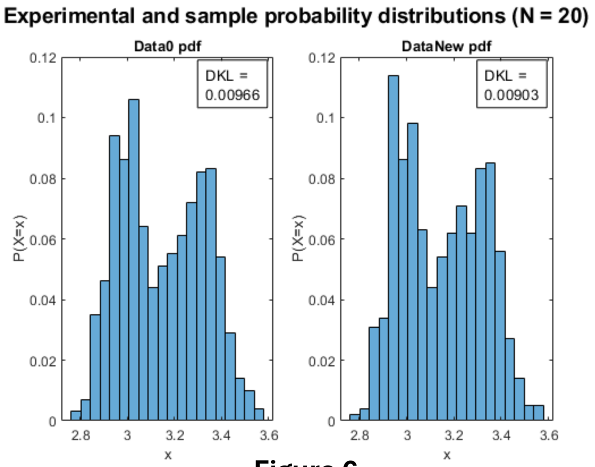


Figure 6

Interestingly, $DKL_{DataNew} < DKL_{Data0}$. I tested a few other seeds and found that this does not *always* hold true, but on average seems to be true. However, this might be confirmation bias. It would be best to run the simulation for a large range of random seeds to determine if this holds true.

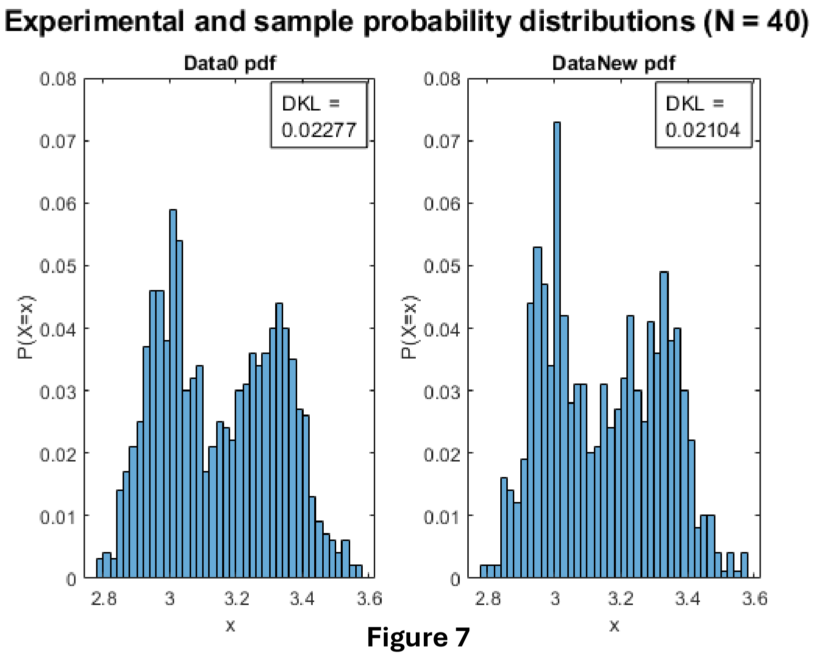


Figure 7

```matlab
%% ----------- MXB261 Problem Solving Task: Part 1 -----------------------]
% Kassia Lembryk-Walsh (n11090677)
% Referencing falling and accreting particles, rather than a crowd of
% people, because that doesn't make sense :3

%% Initialising...

% Two arrays, one for all cases N = 100, one for all cases N = 200
p_N100 = zeros(2, 100, 8);
p_N200 = zeros(2, 200, 8);

for i = 1:4 % Loops over probability cases
    % SETTING PROBABILITY
    if i == 1       % case i)------------]
        s = 1/3; w = s;     e = s; %    -]
    elseif i == 2   % case ii)          -]
        s = 2/3; w = 1/6;   e = 1/6; %  -]
    elseif i == 3   % case iii)         -]
        s = 3/5; w = 3/10;  e = 1/10; % -]
    else            % case iv)          -]
        s = 3/5; w = 1/10;  e = 3/10; % -]
    end %--------------------------------]
    for j = 1:2 % Loops over starting position cases
        % SETTING STARTING POSITION
        if j == 1 % -------]
            P = 1;      % -]
        else            % -]
            P = 'rand'; % -]
        end % -------------]

        % GENERATING RESULTS
        [p_N100(:,:, 2*i - 2 + j), ~] = accretion(P, 100, s,w,e);
        [p_N200(:,:, 2*i - 2 + j), ~] = accretion(P, 200, s,w,e);
        % accretion.m was made by me
    end
end

%% Visualising histograms

bins = 99; % 20 looks way cooler

% Fixed starting position, N = 100
P1_N100_figure = figure;
P1_N100_title = sgtitle(P1_N100_figure,'Figure 1: Cases for P = 1 and N = 100');
P1_N100_title.FontSize = 18; P1_N100_title.FontWeight = "bold";
for i = 1:4 % Plotting case i, ii, iii and iv for P = 1 and N = 100
    subplot(2,2, i);
    histogram(p_N100(1,:, i*2 - 1), bins); % P = 1 on odd numbers
    title(sprintf('Case %d', i));
    xlabel('x position');
    ylabel('particle count');
    xlim([1 99])
end

% Fixed starting position, N = 200
P1_N200_figure = figure;
P1_N200_title = sgtitle('Figure 2: Cases for P = 1 and N = 200');
P1_N200_title.FontSize = 18; P1_N200_title.FontWeight = "bold";
for i = 1:4 % Plotting case i, ii, iii and iv for P = 1 and N = 200
    subplot(2,2, i);
    histogram(p_N200(1,:, i*2 - 1), bins); % P = 1 on odd numbers
    title(sprintf('Case %d', i));
    xlabel('x position');
    ylabel('particle count');
    xlim([1 99])
end

% Random starting position, N = 100
PR_N100_figure = figure;
PR_N100_title = sgtitle("Figure 3: Cases for P = 'rand' and N = 100");
PR_N100_title.FontSize = 18; PR_N100_title.FontWeight = "bold";
for i = 1:4 % Plotting case i, ii, iii and iv for P = 'rand' and N = 100
    subplot(2,2, i);
    histogram(p_N100(1,:, i*2), bins); % P = 'rand' on even numbers
    title(sprintf('Case %d', i));
    xlabel('x position');
    ylabel('particle count');
    xlim([1 99])
end

% Random starting position, N = 200
PR_N200_figure = figure;
PR_N200_title = sgtitle("Figure 4: Cases for P = 'rand' and N = 200");
PR_N200_title.FontSize = 18; PR_N200_title.FontWeight = "bold";
for i = 1:4 % Plotting case i, ii, iii and iv for P = 'rand' and N = 200
    subplot(2,2, i);
    histogram(p_N200(1,:, i*2), bins); % P = 'rand' on even numbers
    title(sprintf('Case %d', i));
    xlabel('x position');
    ylabel('particle count');
    xlim([1 99])
end

% Saving figures as images
% saveas(P1_N100_figure, 'Figure_1.png')
% saveas(P1_N200_figure, 'Figure_2.png')
% saveas(PR_N100_figure, 'Figure_3.png')
% saveas(PR_N200_figure, 'Figure_4.png')
% I have so far spent about 4 hours trying to make the above code work
% correctly -- for some reason, sgtitle overlaps with the figure rather
% than existing within its own white space. Even though the saveas function
% supposedly works the same as "Save as" from figure window, doing it
% manually that way *does* save the figures correctly. So I'm doing it that
% way, because I refuse to spend another 4 hours on this.
```

```
function particles = accretion(P, N, s, w, e)
% ACCRETION - Biased random walk of falling and accreting particles
% particles = accretion(P,N,s,w,e) takes N many particles beginning at a
% starting point y = 99, x = [1,99] determined by P, and 'drops' them
% until they hit the bottom, or accrete onto another fallen particle. This
% fall is a biased random walk. A particle will move; south (-y) at
% probability s, west (-x) at probability w, and east (+x) at
% probability e. The west and east boundaries loop, x = 0 equivalent to
% x = 99, x = 100 equivalent to x = 1.
% The function outputs particles, a 2xN array of final positions of all
% particles.
% Assumes P == 1 or 'rand'. Assumes s + w + e == 1.

%% Initialising

domain = 99; % Set by the question, but it's useful to have here
p = cumsum([s,w,e]); % Cumulative sums of the probabilities
boundary = false([domain,domain+1]); % Records the final position
%^% NOTE!! For boundary; dim=1 is the x-coordinate, dim=2 is y + 1
boundary(:,1) = 1; % Adding a collision boundary along y = 0 (y+1=1)
% boundary is the collision mask, particles will check against this to see
% if a position they intend to move into is already occupied. There is
% collision set along the bottom of the domain (so particles do not fall
% endlessly), and once a particle reaches its final position, it will be
% added to the collision mask
particles = zeros([2,N], 'int8'); % x=row1, y=row2; dim=2 is particle no.
particles(2,:) = domain; % Setting y = 99 for all particles
% particles records the position of each N particles, in terms of x and y
% coordinates. As such, by the end of the function, particles contains the
% final positions.

if P == 1 % Applying start conditions for x
    particles(1,:) = 50; % All particles begin at x = 50
elseif isequal(P,'rand')
    particles(1,:) = randi([1,domain], 1,N); % x = N random int from 1-99
else
    error("Input error. P can only be 1 or 'rand'.")
end

%% Running simulation

for i = 1:N % Looping over each particle
    % Initialising while loop
    rest = false; % There are valid moves for particle i
    south = false; west = false; east = false; % Particle hasn't moved yet!

    safety_switch = 0;
    while rest == false % Looping until particle i can no longer move.
        %-------SAFETY SWITCH----------------]
        safety_switch = safety_switch + 1; % ]
        if safety_switch > 10^6            % ]
```

```
            safety_switch = 0, break       % ]
        end %---SAFETY SWITCH----------------]
        % Because while loops can get out of hand; this will break the loop
        % if it iterates too many times (i.e., if it's not actually
        % progressing towards the end of the loop), and will output
        % safety_switch = 0 since it ends in "," not ";"

        u = rand; % Determining a direction
        if u <= p(1) || (west && east)
        % MOVING SOUTH
        % If there are collisions to the east and west, ignore the
        % randomiser and check south.
            target = particles(:,i) + int8([0; -1]);
            south = true;
        elseif (u <= p(2) && ~west) || east
        % MOVING WEST
        % If there is a collision west, ignore the randomiser and skip. If
        % there is a collision east (and you already skipped south), ignore
        % the randomiser and check west.
            target = particles(:,i) + int8([-1; 0]);
            west = true;
        elseif u <= p(3) || west
        % MOVING EAST
        % If there is a collision to the west, ignore the randomiser and
        % check east.
            target = particles(:,i) + int8([1; 0]);
            east = true;
        end

        wrap_left2right = domain * int8(target(1) < 1);
        wrap_right2left = domain * int8(target(1) > domain);
        target(1) = target(1) + wrap_left2right - wrap_right2left;
        % If x <= 0, +domain (to loop back around to x <= domain)
        % If x > domain, -domain (to loop back around to x >= 1)
        % We're only applying this to x because;
        %   1. There is no +y movement
        %   2. We want y = 0 to exist so it triggers the boundary there
        collision = boundary(target(1), target(2)+1);

        if collision && south % Collides with something south
            boundary(particles(1,i), particles(2,i)+1) = true;
            rest = true; % There are no valid moves for particle i
        elseif ~collision % No collisions
            % If there is a collision, particle position is not updated and
            % that direction will be set =true at the start of the loop.
            south = false;
            west = false;
            east = false;
            particles(:,i) = target; % Free to move!
        end
    end
```

```
end
end
```

```
%% ----------- MXB261 Problem Solving Task: Part 2 -----------------------]
% Kassia Lembryk-Walsh (n11090677)

%% Initialising

load('sampledata2024.mat')
bins = [10, 20, 40];
seed = int8(83);

%% Visualising histograms
% v) ------------------------------------------------------------------]

N10 = sample_from_dist(Data0, bins(1), seed);
N20 = sample_from_dist(Data0, bins(2), seed);
N40 = sample_from_dist(Data0, bins(3), seed);
N40.Children(3).YLim = N40.Children(2).YLim; % Matching the y-limits for a better display
% saveas(N10, 'Figure_5.png')
% saveas(N20, 'Figure_6.png')
% saveas(N40, 'Figure_7.png')
% Once again, saveas does not seem to work well with sgtitle. :/
```

```
function f = sample_from_dist(Data0, bins, seed)
% Generates binned data for the probability distribution function of Data0,
% creates DataNew by sampling from the pdf, and generates binned data for
% the pdf of DataNew.

%% Determining pdf and generating samples

% i) -----------------------------------------------------------------]
n = length(Data0);
[N0, edges0] = histcounts(Data0, bins);
%^% Generates bins, and the frequency of data within each
Data0_pdf = N0 / n; % Such that the total count is equivalent to P = 1

% ii) ----------------------------------------------------------------]
Data0_cdf = zeros(1, bins+1); % Initialises a 1x20 array
for i = 1:bins
    Data0_cdf(i+1) = sum(Data0_pdf(1:i));
    %^% Sums elements from 1 to i of Data0_pdf, such that they add up to 1.
    % Changed to (i+1), such that there is an element zero
end

rng(seed); % Sets random number generation to a specific seed
DataNew = zeros(1,n); % Initialising generated sample
for sample = 1:n
    u = rand; % Generate random number between 0 and 1
    for i = 2:bins+1 % Starting at i=2, since Data0_cdf(1) = 0
        if u <= Data0_cdf(i) % Determining which bin this sample is from
            x1 = edges0(i-1); x2 = edges0(i); % Segment of data this bin contains
            y1 = Data0_cdf(i-1); y2 = Data0_cdf(i); % Segment of cdf
            % y1 < u <= y2; to convert this into a data sample, we find the
            % x coordinate associated with u.
            DataNew(sample) = (u - y1) * ((x2 - x1) / (y2 - y1)) + x1;
            break
        end
    end
end

% iii) ---------------------------------------------------------------------]
[N1, edges1] = histcounts(DataNew, bins);
DataNew_pdf = N1 / n; % such that the total count is equivalent to P = 1

%% Kullback-Leibler measure

% iv) ----------------------------------------------------------------------]
DKL_step = @(p, q) p * log(p/q);
DKL_0 = 0;
for i = 1:bins
    step = DKL_step(Data0_pdf(i), DataNew_pdf(i));
    if isfinite(step) && ~isnan(step)
        DKL_0 = DKL_0 + step;
    else
```

```
        % if p or q are zero, output step but do not add it
        step
    end
end
DKL_New = 0;
for i = 1:bins
    step = DKL_step(DataNew_pdf(i), Data0_pdf(i));
    if isfinite(step) && ~isnan(step)
        DKL_New = DKL_New + step;
    else
        % if p or q are zero, output step but do not add it
        step
    end
end

%% Visualising histograms

f = figure;
subplot(1,2, 1), histogram('BinCounts', Data0_pdf, 'BinEdges', edges0)
ylabel('P(X=x)'), xlabel('x'), title('Data0 pdf')
subplot(1,2, 2), histogram('BinCounts', DataNew_pdf, 'BinEdges', edges1)
ylabel('P(X=x)'), xlabel('x'), title('DataNew pdf')
complete_title = sgtitle(sprintf('Experimental and sample probability distributions (N =↙
%d)',bins));
complete_title.FontSize = 14; complete_title.FontWeight = "bold";
annotation('textbox',[.345 .76 .11 .1],'String',...
    sprintf("DKL = %0.5f", DKL_0))
annotation('textbox',[.785 .76 .11 .1],'String',...
    sprintf("DKL = %0.5f", DKL_New))
end
```