Assignment #2 - Random Parameter Values, Random Matrices & Monte-Carlo Integration

Jonathan Rainer

February 14, 2016

1. Using the May-like random matrices developed in Week 5 Practical, and incorporating the sign structure used in the Allesina and Tang [1] interpretation of predator-prey and mixture interactions, verify numerically the key result of [1].

   [I.E. Show that under the assumptions of [1], predator-prey interactions are stabilising, and mixture interactions are destabilising.]

   In order to verify the result of Allesina and Tang we first need to generate the necessary random matrices, from which we can construct the required eigen-spectra. In order to do so we turn to the supplementary material [1] of the paper where there are instructions given as to how the matrices of each type are to be generated. The code that produces these matrices can be seen in Figure A.1. Then using these generated matrices we can re-create the plots that are found in Figure 1 of [1], shown here in Figure 1.1. The code that generated these data for these plots can be seen in Figure A.2 and A.3 and the code that constructed the complete figure can be seen in Figure A.4.

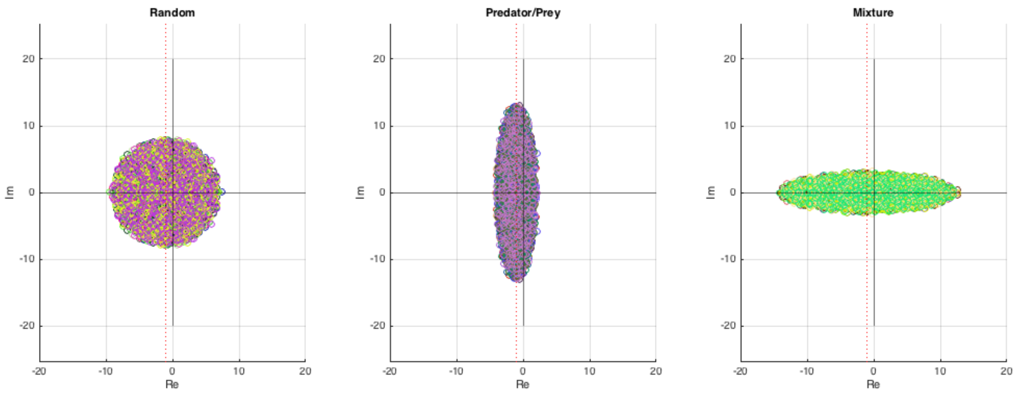

Figure 1.1.: A recreation of the figure seen in Allesina and Tang [1] showing that the tendency towards stability is more prevalent in the Predator-Prey case because the majority of the eigenvalues have negative real parts, as compared to the mixture case where there are a higher proportion of cases of eigenvalues having positive real parts. These plots were produced with the same values used in the original paper, i.e $S = 250$, $C = 0.25$, $\sigma = 1$ and $d = 1$. The red line indicates the value of $-d$ to show the centre of the generated ellipses and the colours indicate the original 10 matrices from which these eigenvalues were drawn.

From this plot we can see that in the case of Predator-Prey relationships the eigenvalues form an ellipse that this stretched along the imaginary axis and in the Mixture case we get an ellipse stretched along the real axis. Since we know from May [5] that stability arises when all eigenvalues have negative real parts we can easily see how, in the general case, predator prey relationships will tend towards stability because the ellipse, centred at $(-d)$ only protrudes into the real axis by a small amount as compared to the mixture case. Consequently the likelihood of just one eigenvalue having a positive real part is much higher in the mixture case leading it to tend towards instability, whilst the predator-prey case tends towards stability for precisely the opposite reason.

Another way of thinking about it is that in order to increase the probability of stability the centre of the ellipse $(-d)$ would only have to shifted slightly in the Predator/Prey case whereas the shift in the mixture case would have to be a lot more severe to account for the large protrusion along the positive real axis. This indicates that a higher proportion of the points lie to the right of the imaginary axis which supports the idea that more of the eigenvalues from the generated matrices have positive real parts in the mixture case.

This is further borne out by the graph seen in Allesina and Tang in Figure 2 showing the hierarchy that their stability criteria entail.

2. Suppose than an unstructured random matrix of the form considered by Allesina and Tang (i.e. a matrix leading to the circular eigenvalue spectrum in Figure 1a) is modified by having the elements of each row multiplied by a factor R_j, where $R_j \in [0, 2]$ is a uniform random variable chosen independently for each row. Evaluate the consequences of this modification for stability (in the sense of Allesina and Tang), in comparison to the roles of mixture and predator-prey interactions.

   The first thing to do in this case is to adapt the described process into the generation of our random matrices from which we illicit our eigenvalues. This is reasonably easy to do and the code that achieves this is included in Figure A.5. This then results in the plot that can be seen in Figure 1.2, which is generated by a variant of the code in Figure A.4 In the figure we see, in red, the original un-augmented eigenvalues and then in blue the augmented ones.

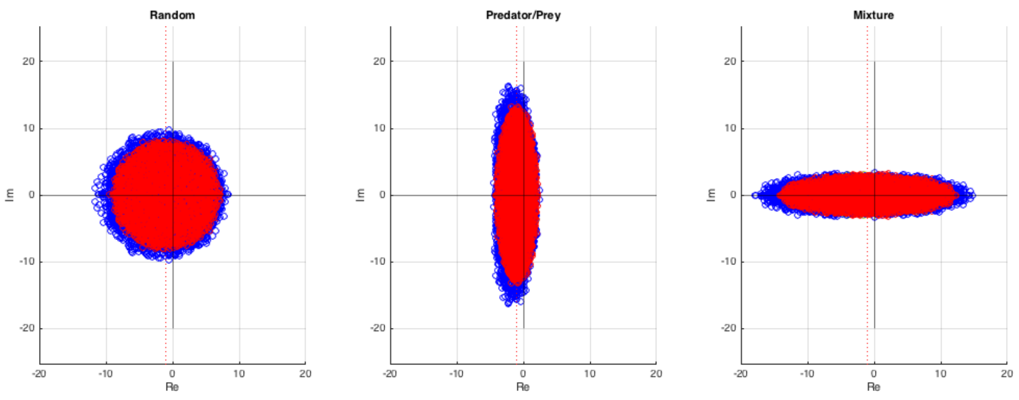

Figure 1.2.: This is similar to the previous plot in that it shows the spectra of the eigenvalues plotted for 10 different random matrices but this time they have the described change made to them prior to plotting the eigenvalues in the case of the blue points. This shows that in general the ellipses are distorted little in terms of the height but much more so in their width, which leads to higher levels of instability in the random case. The other cases are included for comparison purposes.

   As we can see in the predator prey case there is little change to the likelihood of stability, because the stretch in the ellipse is in the imaginary axis and if anything there's a higher concentration of augmented points in the section to the left of the imaginary axis. In the case of the mixture, again the effect is more pronounced in the stable zone but non-negligible in quadrants I and IV and we see this more in the random case where the effect is to change the shape of the spectra from a circle to an ellipse, more akin to the mixture case. This certainly implies that the model does not hold because it implies that the perfect circle shape of the ellipse in the random case relies on all of the interactions to be between populations of relatively the same size. This is what is being modelled by the multiplication by R_j, the case where a populous species interacts with one that is much rarer, a common phenomenon in ecology. This implies that changing the matrix in this way violates the predictions of the model which casts doubt on it's validity for general ecosystems.

3. Consider the Thébault and Fontaine model of obligate mutualism used in James et al. (2012), as detailed in the Supplementary Information. The authors claim that (SI, p12.):

"Moreover, simple simulations on systems with only a single plant species show that an isolated mutualistic pair of species, each with no other partners, has an extinction probability of over 99%"

Solve the system numerically and check whether this claim is true.

In James et al. [3] we see that the system being tested is the following:

$$\frac{1}{N_i}\frac{dN_i}{dt} = \alpha_i - \sum_{j\in P}\beta_{ij}N_j + \sum_{k\in A}\frac{A_{ik}\gamma_{ik}h_{ik}N_k}{1+h_{ik}\sum_{l\in A}N_l}$$

Where N_i is the population of plants of species i and the rest of the terms are as defined in the paper. A symmetric equation exists for the animal species. Once we specialise this equation to the case of one plant and one animal we see that it reduces to the following set of equations: (here P refers to the population of the single plant species and A refers to the population of the single animal species)

$$\begin{aligned}\frac{1}{P}\frac{dP}{dt} &= \alpha_A - \beta P + \frac{\gamma h A}{1+hA}\\ \frac{1}{A}\frac{dA}{dt} &= \alpha_P - \beta A + \frac{\gamma h P}{1+hP}\end{aligned}$$

Now the simplest way to verify the result in the question is to solve the system for lots of different random parameters, within the limits set down in the paper, and then see if they lead to extinction. Here we define extinction as having converged to the trivial fixed point $(0,0)$ which must be present in all the systems because of the $\frac{1}{A}$ or $\frac{1}{P}$ factor in each equation. The code that ran these experiments can be seen in Figure A.7 with the code for the referenced `tfderivs` visible in Figure A.6 and the outcome was a prediction of extinction at 97% as can be seen from the output contained in Figure 1.3. This is obviously below the figure that the paper quotes however looking at the small set of examples that fail this test it's obvious why we don't reach the same conclusions.

```
>> [outs, fails] = construct_probability_estimate(100, 0.0001, 1000)
>> outs

outs =
974    26
```

Figure 1.3.: The output from the construction of the estimate of probability made by counting how many times the system converges to the trivial fixed point at (0,0) against the number of times it does not. Leading to a 97.4% rate of extinction.

The first reason is to do with the choice of a strong or weak interaction. From experimentation I've observed that this parameter is among the key factors that decide survival. In the code used here, if you repeat the experiment choosing always a weak interaction then you get an extinction rate of 100% and always choosing a strong interaction the probability is much lower, nearer to 90%. If you then proceed to plot some of the strong interaction cases you can see that there emerges a situation where a non-trivial attracting fixed point appears in quadrant I and it's these situations that lead to the failures (an example of this can be seen in the plot in Figure 1.4).

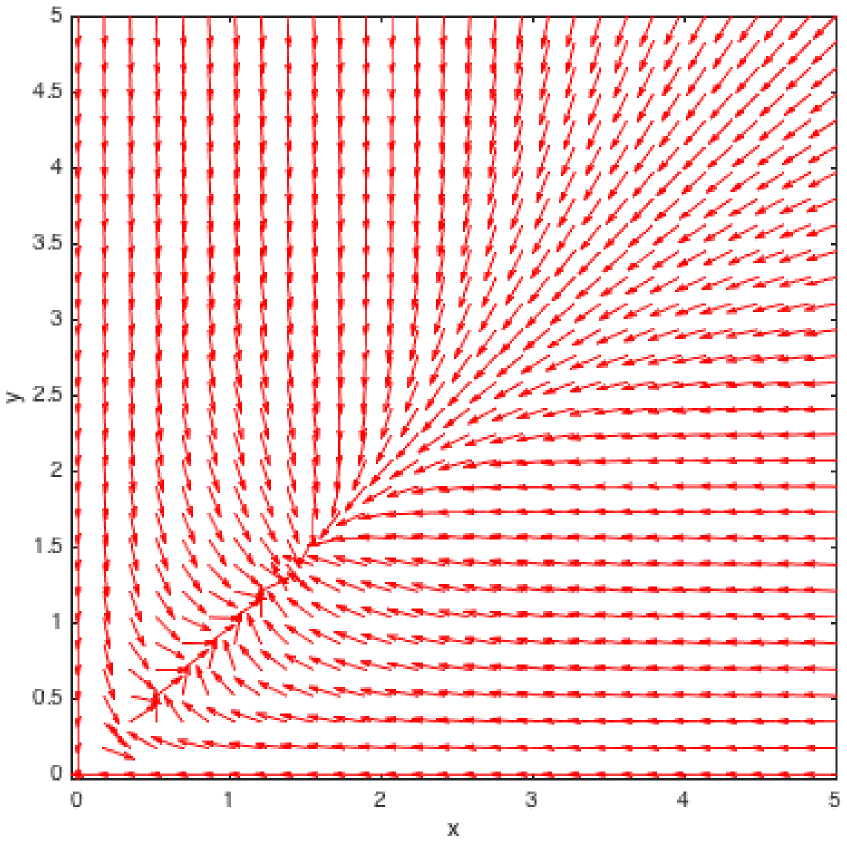

Figure 1.4.: This is the phase portrait for one of the failure cases from the random generation, as you can see, a positive fixed point has appeared in the quadrant I that is an attractor and so the convergence is towards that, and a stable coexistence rather than the extinction required.

This is borne out by evidence in the supplementary material (particularly in Figure S4) that shows that the probability of survival for a species with 1 partner under a weak interaction is very low compared to the strong interaction so all this is consistent with the model the paper considers. In addition to this, the deciding factor in the code written to generate the estimate, is a random number generator that should equally choose between both alternatives. Since the paper is unclear exactly how they chose the random parameter values that led to the 99% figure it's to be assumed that this is one of the sources of disrepancy as an alteration in the probability of choosing either interaction type would have a marked effect on the outcome.

A second point is that there is at least one case where the convergence to the zero state is simply taking too long for us to simulate under the imposed conditions. For example in the paper they estimate it over 10000 time steps but that would take too long under with the computing hardware available. Consequently there are results that appear as failures but actually are converging to the required point. This is most obvious in an example like the following where the ode45 solution to the differential equations goes from being $(0.000181759015370329, 1.30532717081323e-07)$ at timestep 900 to $(0.000137379003631396, 4.447025454157763e-08)$ at timestep 1000. This is only a decrease of 5×10^{-5} in the P direction and 9×10^{-8} in the A direction over 100 timesteps, showing the very slow rate of convergence. All this together contributes to the higher level of survival than would be expected according to James et al. [3].

4. Explain briefly why the models considered by Coyte et al. [2] (as detailed in Section Method 1a of the Supplementary Material, pages 4 and 5) are vulnerable to the criticisms about diagonal elements in the Jacobian as outlined in [4]. Do the main conclusion of Coyte et al. [2] change qualitatively when diagonal variability is added? Explain your answer with the support of appropriate numerical outputs.

   In James et al. [4] they state that making the assumption, as May does in [5], that all the diagonal elements of a matrix will be the same is a problem. They state that introducing variability tends to increase the value of the leading eigenvalue, which could have consequences for stability as we'll see later. Also they state that from an ecological standpoint this kind of assumption is simply not valid, however the problem with Coyte's paper is that she makes this very same assumption by setting the value of $s_i = s$ in all cases. Consequently it's vulnerable to the same criticism because it follows the same pattern as the systems described in James et al. [4].

   Coyte's main conclusion is two-fold in that co-operation has a destabilising influence in general and competition improves stability within the microbiome context. To introduce diagonal variability is reasonably easy because we can re-use the Allesina-Tang code and introduce the variability separately to see how this affects the outcome. This can be seen in Figure 1.5 (the code which augmented the diagonals can be see in Figure A.8, the plot itself is generated by code very similar to A.4) which seems to suggest that there actually is very little quantitatively that changes when this is introduced.

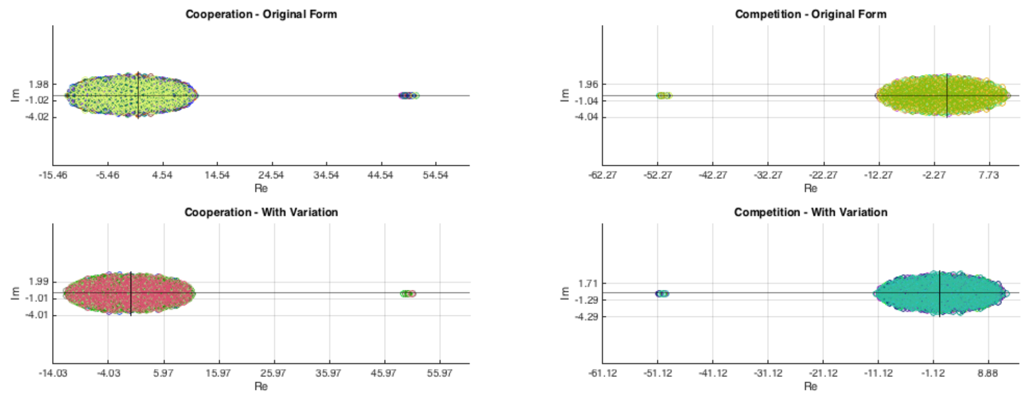

Figure 1.5.: This shows the results of applying a similar augmentation to that used in James et al. [4] and shows that there is very little qualitative change.

   Considering first the positive outliers in the co-operative case, these change very little from 50.4241 to 51.4779, extracted from Figure 1.5. So the effect of the variation is simply increase the average row sum of the matrix which is unsurprising because, considering the shape of the normal distribution, a large proportion of the numbers selected by the random function will be larger than -1 the previously selected diagonal value. However the change is not qualitatively significant, it will take slightly longer for this system to return to stability but the competitive case is much more likely to be stable. Looking at the competitive case, again we get a very small change in the value of the outliers however the important factor is the furthest "positive" eigenvalue. Again, here the increase is very modest, from 11.0154 to 12.1414, so again there will be a small change in how long it takes for systems of this form to return to stability but the main conclusions remain intact. Overall Coyte's conclusions seem relatively robust to the variance of the diagonal as the qualitative behaviour is very similar to that of the non-varying case.

Appendices

A. Programming Code

Listing A.1: This shows the code that generates the Allesina-Tang matrices that we are to investigate. The algorithm for generation is based heavily on the Supplementary Material in [1].

```
function at_matrix = generate_at_matrix(matrix_size, c, d, mode, sigma)
%GENERATE_AT_MATRIX Generate Allesina and Tang matrices of various types.

    % Generate a set of random numbers from the distribution U[0,1] as a
    % matrix rather than having to generate them all individually.
    uniform_rands = rand(matrix_size, matrix_size);
    % Switch based on the mode that the function has been called in, these
    % are listed as comments above each switch case.
    switch mode
        % mode = 0 -> Random Matrices
        case 0
            % Form a mask that consists of the positions where the random
            % numbers generated are less than or equal to C, the
            % connectance property. Then take out the diagonal elements so
            % they can be added in later
            creation_mask = (uniform_rands <= c) .* ~eye(matrix_size);
            % Using the mask created pull out random numbers from the
            % distribution N(o,sigma^2) to form the matrix.
            at_matrix = normrnd(0,sigma,matrix_size,matrix_size)  ...
                .* creation_mask;
        % mode = 1 -> Predator/Prey Matrices
        case 1
            % Form the creation mask from the upper half of the
            % uniform_rands matrix, where the entries are less than or
            % equal to C.
            creation_mask = triu(uniform_rands <= c, 1);
            % Form a mask to show the location of zeroes so that we don't
            % pick up false positives later.
            zero_mask = creation_mask == 0;
            % Generate a second set of random numbers
            uniform_rands_2 = rand(matrix_size, matrix_size);
            % Create two masks, one that shows the generated random numbers
            % greater than 0.5 and the other that shows the ones that are
            % lower, taking out those numbers below the central diagonal
            % that should be zero anyway.
            pos_mask = (uniform_rands_2 .* creation_mask) >= 0.5;
            neg_mask = ((uniform_rands_2 .* creation_mask) < 0.5) ...
                - zero_mask;
            % Turn these masks into upper triangular random matrices with
            % numbers draw from the correct distributions.
            pos_random_numbers =  ...
                abs(normrnd(0,sigma,matrix_size,matrix_size));
            neg_random_numbers = ...
                -abs(normrnd(0,sigma,matrix_size,matrix_size));
            % Glue together the matrix by matching entries in a mask with
            % their corresponding generated random numbers as well as
            % matching their transposes with the opposite generated random
            % numbers to produce the +/-, -/+ pairs that are required.
            at_matrix = ((pos_mask + neg_mask') .* pos_random_numbers) + ...
```

```
                ((pos_mask' + neg_mask) .* neg_random_numbers);
        % mode = 2 -> Mixture Case
        case 2
            % Similar process to case 1
            creation_mask = triu(uniform_rands <= c, 1);
            zero_mask = creation_mask == 0;
            uniform_rands_2 = rand(matrix_size, matrix_size);
            pos_mask = ...
                (uniform_rands_2 .* creation_mask) <= 0.5 - zero_mask;
            neg_mask = ...
                (uniform_rands_2 .* creation_mask) > 0.5;
            pos_random_numbers = ...
                abs(normrnd(0,sigma,matrix_size,matrix_size));
            neg_random_numbers = ...
                -abs(normrnd(0,sigma,matrix_size,matrix_size));
            % Difference here is that instead of matching to create
            % complementary sign pairs we create pairs of the same sign. So
            % we get +/+ and -/- pairs
            at_matrix = ((pos_mask + pos_mask') .* pos_random_numbers) ...
                 + ((neg_mask + neg_mask') .* neg_random_numbers);
        % mode = 3 -> Mutualism
        case 3
            % Create a creation mask formed from the upper triangular part
            % of the uniform_rands matrix, where the entries are less then
            % or equal to C.
            creation_mask = triu(uniform_rands <=c, 1);
            % Generate the random numbers to combine with the creation
            % mask.
            norm_rnds = abs(normrnd(0,sigma,matrix_size,matrix_size));
            % Multiply the mask and it's transpose by the normally
            % distributed random numbers to produce the +/+ pairs required
            % in the matrix.
            at_matrix = ((creation_mask + creation_mask') .* norm_rnds);
        % mode = 4 -> Competition
        case 4
            % Create a creation mask formed from the upper triangular part
            % of the uniform_rands matrix, where the entries are less then
            % or equal to C.
            creation_mask = triu(uniform_rands <=c, 1);
            % Generate the random numbers to combine with the creation
            % mask.
            norm_rnds = -abs(normrnd(0,sigma,matrix_size,matrix_size));
            % Multiply the mask and it's transpose by the normally
            % distributed random numbers to produce the -/- pairs required
            % in the matrix.
            at_matrix = ((creation_mask + creation_mask') .* norm_rnds);
    end
    % Set all the diagonals to -d within the matrix.
    at_matrix = at_matrix + (eye(matrix_size) .* -d);
end
```

Listing A.2: This shows the code that generates large sets of eigenvalues, from matrices generated by A.1 for the plotting function in A.3 to act on

```
function eigenvalues = generate_eigenvalues(iterations, matrix_size, ...
    interaction_prob, d, mode, sigma)
%GENERATE_EIGENVALUES Generate sets of eigenvalues to be plotted.

    % Create a structure to hold the eigenvalues generated.
    eigenvalues = zeros(matrix_size, iterations);
    % Generate one matrix per iteration and extract it's eigenvalues into
    % one column of the matrix defined.
    for i=1:iterations
        M = generate_at_matrix(matrix_size, interaction_prob, d, mode, ...
            sigma);
        eigenvalues(:,i) = eig(M);
    end
end
```

Listing A.3: This shows the code that generates the plot of an eigenvalue spectra for specific set of eigenvalues, usually that are generated by A.2.

```
function [] = plot_eigen_spectra(eigenvalues, colour)
%PLOT_EIGEN_SPECTRA Given a vector of eigenvales, plot their spectra.

    hold on
    % Calculate the size of the plot by looking at the minimum and maximum
    % values of what's to be plotted and then adding a fudge factor so that
    % it doesn't look squashed on the page.
    x_min = round(min(min(real(eigenvalues)))*1.2,2);
    x_max = round(max(max(real(eigenvalues)))*1.2,2);
    y_min = round(min(min(imag(eigenvalues)))*1.2,2);
    y_max = round(max(max(imag(eigenvalues)))*1.2,2);
    % Iterate over the eigenvalues
    for i=1:size(eigenvalues,2)
        % If the colour specifier is set then plot everything in one
        % colour, otherwise choose one randomly per set of eigenvalues.
        % This allows us to see how sets of eigenvalues fall or two
        % independent sets look, depednent on context.
        if ~strcmp('random', colour)
            plot(eigenvalues(:,i),'o', 'color', colour)
        else
            plot(eigenvalues(:,i),'o', 'color', rand(1,3))
        end
    end
    hold off
    % Alter the axis so the whole diagram can be seen, add a grid and make
    % sure the axis are equally spaced in the digram.
    axis([x_min x_max y_min y_max])
    grid
    axis equal
    % Try and calculate an appropriate step size for the ticks based on the
    % range of the eigenvalues.
    if (abs(x_min) + abs(x_max) > 30)
        step_size_x = 10;
    else
        step_size_x = 5;
    end
    if (abs(y_min) + abs(y_max) > 10)
        step_size_y = 5;
    else
        step_size_y = 3;
    end
    % Add in markers to the scales so it's obvious how large or small the
    % generate spectra are
    set(gca,'xtick',x_min:step_size_x:x_max, ...
        'ytick',y_min:step_size_y:y_max);
    % Label the real and imaginary axis.
    xlabel('Re');
    ylabel('Im');
    % Plot axis lines
    hold on
    plot([0 0],[y_min y_max],'k');
    plot([x_min x_max],[0 0],'k');
    hold off
end
```

Listing A.4: This code simply automates the plotting of the figure, by creating the required sets of eigenvalues, and sorting out the presentation of the subplot. The final figure can be seen in 1.1.

```
% Create a new figure and set up its position and size so the plots are the
% right shape.
f = figure(1);
set(f, 'Position', [100, 100, 1200, 400])
% Store some parameters so it's easier to run the commands to subplot in a
% loop.
critical_stabilities = [1, pi/(pi-2), pi/(pi+2)];
initial_sigma = [0.1, 0.5, 0.1];
titles = {'Random', 'Predator/Prey', 'Mixture'};
% Set up the aspect ratio on the plots so that circles look like circles
% rather than stretched ellipses.
daspect([1,1,1]);
% Iterate over the modes to generate, generate the eigenvalues and plot the
% spectra, each in their own pane of the subplot.
for i=1:3
    subplot(1,3,i)
    hold on
    plot_eigen_spectra(generate_eigenvalues(10,250, 0.25, 1,i-1,1), ...
        'random');
    % Add in a line to indicate the centre of the shapes.
    vline(-1)
    % Add titles to each of the subfigures.
    title(titles{i});
    hold off
end
```

Listing A.5: This code generates eigenvalues that come from the augmented matrices described in Q2. The augmentation is performed by creating matrices of scalars and then multiplying them element wise into the generated matrix.

```
function [eigenvalues, eigenvalues_a] = generate_eigenvalues_augmented(...
    iterations, matrix_size, interaction_prob, d, mode, sigma)
%GENERATE_EIGENVALUES_AUGMENTED Generates the augmented eigenvalues.

    % Create structures to store the eigenvalues in when generated
    eigenvalues = zeros(matrix_size, iterations);
    eigenvalues_a = zeros(matrix_size, iterations);
    for i=1:iterations
        % Generate one matrix per iteration of the required form.
        M = generate_at_matrix(matrix_size, interaction_prob, d, mode, ...
            sigma);
        % Generate a matrix for which each row is made up of repititions of
        % the same random number (drawn from the U[0,2]
        scalars = repmat((2*(rand(matrix_size, 1))), 1, matrix_size);
        % Multiply the scalars by the generated matrix, effectively the
        % same multiplying each row by a random scalar.
        M_t = scalars.*M;
        % Extract the original and augmented sets of eigenvalues.
        eigenvalues(:,i) = eig(M);
        eigenvalues_a(:,i) = eig(M_t);
    end
end
```

Listing A.6: This code shows the calculation of the derivative function in Question 3, all the parameters that are included as function parameters are dealt with by using a dummy function when it's passed into `ode45` as can be seen in A.7

```
function F = tfderivs(t, co_ords, col_flag, alpha_A, alpha_P, beta, ...
    gamma, h)

    dPdt = co_ords(1) * (alpha_P - beta  * co_ords(1) + ...
        (gamma * h * co_ords(2))/(1 + h * co_ords(2)));
    dAdt = co_ords(2) * (alpha_A - beta  * co_ords(2) + ...
        (gamma * h * co_ords(1))/(1 + h * co_ords(1)));
    if col_flag
        F = [dPdt; dAdt];
    else
        F = [dPdt, dAdt];
    end
end
```

Listing A.7: This code generates the probability estimates that are used to answer Q3. It runs 10 iterations of one set of random parameter values and the picks 100 sets of these random parameters leading to the conclusions it does.

```
function [outcomes, fails]  = construct_probability_estimate(iterations, ...
    zero_threshold, timespan)
%CONSTRUCT_PROBABILITY_ESTIMATE Estimate the probability of extinction

    % Set up the structure to track the outcomes, successes (extinction)
    % and failures (survivial) in columns 1 and 2 respectively.
    outcomes = zeros(1,2);
    % Set up a structure to track the conditions that led to failure so
    % they can be adequately explored.
    fails = zeros(iterations, 9);
    for i=1:iterations
       % Set the parameters for each of the ranges as defined in the paper.
       alpha_A = rand_range(-0.3, -0.1, 1);
       alpha_P = rand_range(-0.2, -0.001, 1);
       beta = rand_range(1, 2, 1);
       % Flip a coin to decide if the mutualistic interaction is weak or
       % strong.
       if rand() >= 0.5
           gamma = rand_range(0.2, 0.3, 1);
       else
           gamma = rand_range(2, 3, 1);
       end
       h = rand_range(0.1,1, 1);
       % For the chosen random set of parameters iterate 10 times to see
       % where ode45 converges to.
       for j = 1:10
           [~,y] = ode45(@(t,y) tfderivs(t, y, 1, alpha_A, alpha_P, beta, ...
               gamma, h), 1:timespan, rand(1,2));
           % Since we know that there must be a fixed point at (0,0) from
           % the analytical analysis we simply need to check if the system
           % is converging to it every time, so we check that the outcome
           % at the end of the timespan is close enough to zero to be
           % considered converging and then we add to our successes.
           if (y(timespan,:) <= zero_threshold)
               outcomes(1) = outcomes(1) + 1;
           % Otherwise we add to the failures and then store out the
           % details that led us to that conclusion.
           else
               outcomes(2) = outcomes(2) + 1;
               fails(i, :) = [alpha_A, alpha_P, beta, gamma, h, ...
```

```
                    y(timespan, :), y(timespan-100, :)];
            end
        end
    end
    % Eliminate any 0 rows in the failures because we've allocated enough
    % space to assume that every experiment will fail so we won't run out
    % of space.
    fails( ~any(fails,2), : ) = [];
end
```

Listing A.8: This code generates the eigenvalues from matrices that have had their diagonals varied in a method similar to that described in James et al. [4]. The variance of the non-diagonal elements is calculated then a randomly generated set of new diagonals is calculated from that variance and then added over the top of the old diagonals while retaining the rest of the elements.

```
function eigenvalues = generate_eigenvalues_diagvar(iterations, ...
    matrix_size, interaction_prob, d, mode, sigma)
%GENERATE_EIGENVALUES Generate sets of eigenvalues to be plotted.

    % Create a structure to hold the eigenvalues generated.
    eigenvalues = zeros(matrix_size, iterations);
    % Generate one matrix per iteration and extract it's eigenvalues into
    % one column of the matrix defined.
    for i=1:iterations
        M = generate_at_matrix(matrix_size, interaction_prob, d, mode, ...
            sigma);
        % Create a vector to hold all the elements that are non diagonal
        elements = zeros((matrix_size^2 - matrix_size), 1);
        % Iterate over the elements of the matrix and if a non-diagonal
        % element is found store it in elements.
        for j=1:matrix_size
            for k=1:matrix_size
                if(j ~= k)
                    elements((j-1)*matrix_size + (k-1)) = M(j,k);
                end
            end
        end
        % Calculate the variance of the non-diagonal elements
        var_offd = var(elements);
        % Generate the augmented matrix by masking out all the non-diagonal
        % elements and then adding to them the identity matrix times a
        % randomly generated matrix to give the diagonal coefficients. In
        % addition you add a small amount to the diagonals so their sum is
        % zero in line with the James paper.
        amount_to_add = (-sum(var_offd))/matrix_size;
        M_aug = (~eye(matrix_size) .* M) + (normrnd(0,sqrt(var_offd), ...
            matrix_size, matrix_size) .* eye(matrix_size)) + ...
            + (eye(matrix_size) .* amount_to_add);
        eigenvalues(:,i) = eig(M_aug);
    end
end
```

Bibliography

[1] S. Allesina and S. Tang, "Stability criteria for complex ecosystems," *Nature*, vol. 483, no. 7388, pp. 205–208, 2012. [Online]. Available: http://arxiv.org/abs/1105.2071\nhttp://www.nature.com/doifinder/10.1038/nature10832

[2] K. Z. Coyte, J. Schluter, and K. R. Foster, "The ecology of the microbiome: Networks, competition, and stability," *Science*, vol. 350, no. 6261, pp. 663–666, 2015. [Online]. Available: http://www.sciencemag.org/content/350/6261/663.abstract

[3] A. James, J. W. Pitchford, and M. J. Plank, "Disentangling nestedness from models of ecological complexity." *Nature*, vol. 487, no. 7406, pp. 227–30, 2012. [Online]. Available: http://dx.doi.org/10.1038/nature11214

[4] A. James, M. J. Plank, A. G. Rossberg, J. Beecham, M. Emmerson, and J. W. Pitchford, "Constructing Random Matrices to Represent Real Ecosystems," *The American Naturalist*, vol. 185, no. 5, pp. 680–692, 2015. [Online]. Available: http://www.jstor.org/stable/10.1086/680496

[5] R. M. May, "Will a large complex system be stable?" *Nature*, vol. 238, no. 5364, pp. 413–414, 1972.