Simulations of Determinantal Processes

Robert Zimmerman

April 23, 2021

1 Introduction and Background

A *determinantal point process* is a (simple) point process $\mathcal{X}$ on a space Λ with an associated measurable function $\mathbb{K}(x, y) : \Lambda^2 \to \mathbb{C}$ which satisfies the striking property that for any $x_1, \ldots, x_k \in \Delta$, the k-point intensity function $\rho_k(x_1, \ldots, x_k)$ of $\mathcal{X}$ is given by the determinant of a $k \times k$ matrix whose (i, j)'th entry is given by $\mathbb{K}(x_i, x_j)$ (Hough et al. [2009]). Determinantal processes are particularly suited to modelling repulsive point processes, and are desirable for their tractability. While our focus is on applications within random matrix theory, determinantal processes arise in a diverse range of other fields including quantum physics, combinatorics (Soshnikov [2000]), and more recently, machine learning and signal processing (Tremblay et al. [2019]).

A few notable examples are relatively straightforward to simulate. The famous *Gaussian unitary ensemble* (GUE), first introduced by Wigner (Wigner [1993]), is the set of random $n \times n$ matrices with $\mathcal{N}_{\mathbb{C}}(0, \sqrt{2})$ entries that are independent up to a Hermitian constraint. Equivalently, this ensemble is characterized as the probability measure $\mathrm{d}\pi(H) \propto e^{-\frac{n}{2}\operatorname{tr}(H^2)}$ on the space of $n \times n$ Hermitian matrices (Borot [2017]). The set of eigenvalues of these random matrices forms a determinantal process on $\mathbb{R}$, with kernel given by

$$\mathbb{K}_n(x, y) = \sum_{k=0}^{n-1} H_k(x) H_k(y)$$

with respect to $\mathrm{d}\mu(x) \propto e^{-x^2/2}$ on $\mathbb{R}$, where $H_k(x) = \frac{(-1)^k}{\phi(x)} \left(\frac{\mathrm{d}^k}{\mathrm{d}x^k} \phi(x) \right)$ is the k'th (probabalist's) Hermite polynomial.

The *circular unitary ensemble* (CUE), one of the three ensembles of unitary matrices introduced by Dyson (Dyson [1962]), is characterized by the Haar measure on the set of $n \times n$ unitary matrices $\mathcal{U}(n)$. The set of eigenvalues of these random matrices forms a determinantal process on the unit circle S^1, with kernel given by

$$\mathbb{K}_n(e^{i\theta}, e^{i\phi}) = \frac{1}{2\pi} \sum_{k=0}^{n-1} e^{ik\theta - ik\phi}$$

with respect to Lebesgue measure on S^1. Surprisingly, the eigenvalues of any $m \times m$ submatrix of a CUE matrix (with $m < n$) – that is, a *truncated unitary matrix* – also form a determinantal process on the unit disk $\mathbb{D}$, but with a quite different kernel given by

$$\mathbb{K}_n(z, w) = \sum_{k=0}^{n-1} \frac{(m+1) \cdots (m+k)}{\pi k!} (z\bar{w})^k (1 - |z|^2)^{m-1} (1 - |w|^2)^{m-1}$$

with respect to Lebesgue measure on $\mathbb{C}$ (Zyczkowski and Sommers [2000]).

The *complex Ginibre ensemble*, one of three ensembles of non-Hermitian matrices introduced by Ginibre (Ginibre [1965]), is the set of random $n \times n$ matrices with independent $\mathcal{N}_{\mathbb{C}}(0, \sqrt{2})$ entries. Equivalently, it is characterized as the probability measure $\mathrm{d}\mu(H) \propto e^{-\mathrm{tr}(H^* H)}$ on the space of $n \times n$ complex-matrices (Fischmann et al. [2012]). The set of eigenvalues of these random matrices forms a determinantal process on $\mathbb{C}$ with kernel given by

$$\mathbb{K}_n(z, w) = \sum_{k=0}^{n-1} \frac{(z\bar{w})^k}{k!}$$

with respect to the measure $\mathrm{d}\mu(z) \propto e^{-|z|^2}$ on $\mathbb{C}$. Similarly, the *spherical ensemble*, introduced by Krishnapur (Krishnapur [2006]) in the context of matrix-valued Gaussian analytic functions, is the set of random $n \times n$ matrices $A^{-1}B$, where A and B are independent matrices from the complex Ginibre ensemble (or equivalently, the zeros of the random analytic function $z \mapsto \det(zA - B)$). The set of eigenvalues of these random matrices forms a determinantal process on $\mathbb{C}$ with kernel given by

$$\mathbb{K}_n(z, w) = \frac{n}{\pi} \frac{(1 + z\bar{w})^{n-1}}{(1 + |z|^2)^{\frac{n+1}{2}} (1 + |w|^2)^{\frac{n+1}{2}}}$$

with respect to Lebesgue measure on $\mathbb{C}$.

Moving away from eigenvalues, a further example of a determinantal process is that which describes the configurations at time t of independent time-homogeneous random walks on $\mathbb{Z}$ which are conditioned not to intersect and to return to their starting configurations at time $2t$. That these midway configurations of *non-intersecting random walks* are described by a determinantal process is a consequence of the famous Karlin-McGregor theorem (Karlin et al. [1959]). Specifically, if n such random walks $X_1(t), \ldots, X_n(t)$ each have t-step transition probabilities $P_t(\cdot, \cdot)$ and start at $(x_1, x_2, \ldots, x_n) \in \{z \in \mathbb{Z}^n : z_1 < z_2 < \cdots < z_n\}$, then their configuration at time t, $\{X_p(t) : 1 \le p \le n\}$ is determinantal with kernel

$$\mathbb{K}_n(u, v) = \sum_{j=1}^{n} \psi_j(u)\psi_j(v)$$

with respect to any Radon measure $\pi(\cdot)$ on $\mathbb{Z}$ that makes the random walk reversible, where

$$\psi_j(r) = \sum_{k=1}^{n} \left[A^{-\frac{1}{2}}\right]_{j,k} \frac{P_t(x_k, r)}{\pi(r)}$$

and $A_{j,k} = \frac{P_{2t}(x_j, x_k)}{\pi(x_k)}$.

In theory, these determinants should provide us with a relatively simple way to calculate joint densities. In this paper, we describe the method and results of simulating these six determinantal point processes in `R`, and visually comparing these observed distributions of their point configurations with those predicted by the underlying theory.

2 Methods

For each example of a determinantal process, we simulated a large number of bivariate realizations of the process, and then displayed a scatterplot of the realizations and compared it to a surface plot of the associated joint density $\rho_2(\cdot, \cdot)\, \mathrm{d}\mu^{\otimes 2}(\cdot, \cdot)$ over a representative (discretized) subset of Λ^2. To emphasize the "determinantal" aspect of these processes, we computed joint correlations by first calculating the relevant kernel $\mathbb{K}(\cdot, \cdot)$, populating a matrix $[K]_{i,j} = \mathbb{K}(x_i, x_j)$, and then explicitly calculating $\det(K)$. For the explicit calculation, we either directly used `R`'s `det` function for real-valued K, or computed the product of the eigenvalues of K via the `eigen` function for complex K, since `det` does not currently work for complex matrices. Where possible, we use `R`'s built-in kernel density estimation procedures to approximate the empirical density using contour lines.

We chose to use the number of points $k = 2$ as a reasonable compromise: the case $k = 1$ case is uninteresting because $\rho_1(x)\, \mathrm{d}\mu(x)$ simply returns the background measure $\mu(x)$ on the underlying space, and does nothing to exploit the underlying determinantal structure. On the other hand, there is no easy way to visualize functions whose domains are of dimension (over $\mathbb{R}$) greater than 2. However, while the choice of $k = 2$ is satisfactory for examples on spaces which are subsets of $\mathbb{R}$, it still poses problems when the spaces are subsets of $\mathbb{C}$, as is the case for the complex Ginibre ensemble, the spherical ensemble, and the truncated unitary ensemble. For each of these, we distinguished two points $w_1, w_2 \in \mathbb{C}$ – the origin $w_1 = (0, 0)$ and a randomly chosen point in the unit disk – and then plotted $\mathbb{K}_2(z, w_1)$ and $\mathbb{K}_2(z, w_2)$ over $z \in \mathbb{C}$. Rather than simulating from conditional distributions, we simulated from the full joint distribution, and then preserved those samples for which the second coordinate lies in an 0.1-neighborhood of the "distinguished" points. The `R` code used to produce these simulations can be found in Appendix: Code.

3 Results and Discussion

In the following sections, our notations and descriptions mainly follow those in Section 4.3 of Hough et al. [2009].

3.1 Gaussian Unitary Ensemble

A matrix H is easily drawn from the GUE by first sampling an $n \times n$ matrix A with independent $\mathcal{N}_{\mathbb{C}}(0, \sqrt{2})$ entries (A is thus is a member of the Ginibre ensemble, which we investigate later) and then symmetrizing it by taking $H = \frac{A+A^*}{\sqrt{2}}$.

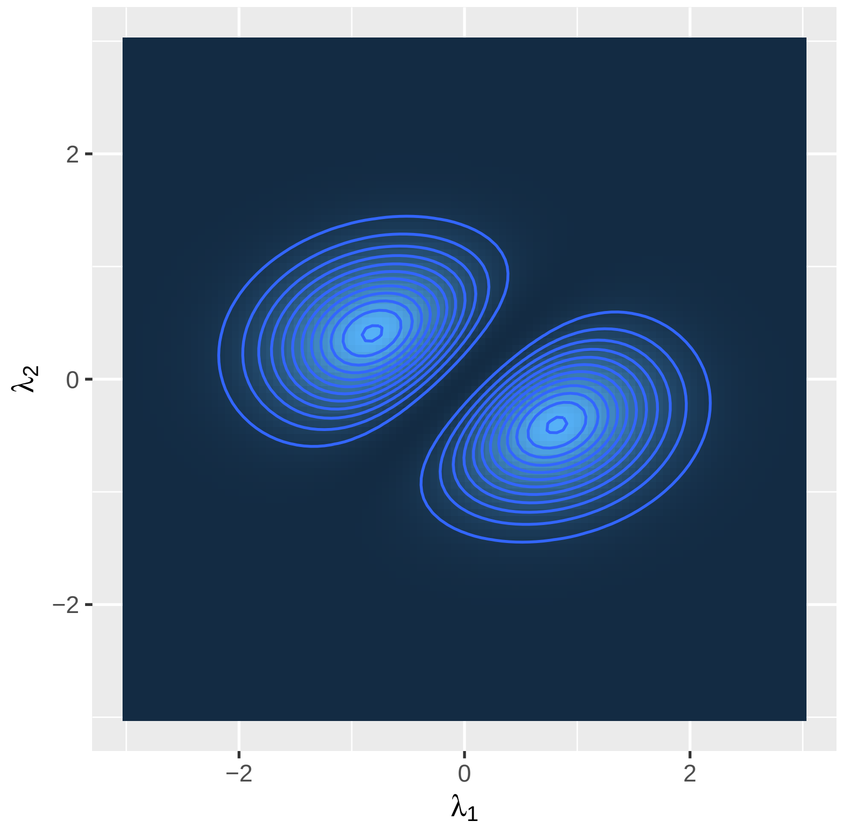
In Figure 1 (top), we plotted $\mathbb{K}_n(x, y)\, \mathrm{d}^{\otimes 2}\mu(x, y)$ for $(x, y) \in [-3, 3]^2$, discretized to a lattice of 100^2 points. Figure 1 (bottom) shows our scatterplot of 10,000 pairs of simulated eigenvalues (λ_1, λ_2). We observed a characteristic symmetric repulsion between λ_1 and λ_2 across the $y = x$ line, and mass contained within ellipses with foci at $(-1, 1)$ and $(1, -1)$ (the maxima of $\mathbb{K}_2(x, y)$).

3.2 Circular Unitary Ensemble and Truncated Unitary Ensemble

In practive, sampling a matrix from the CUE is slightly more delicate than sampling from the GUE. It is known (Eaton [1983]) that starting with an $n \times n$ matrix A with independent $\mathcal{N}_{\mathbb{C}}(0, \sqrt{2})$ entries, one can perform Gram-Schmidt orthogonalization on the columns on A, and collect the resulting vectors into a matrix to produce a sample from the CUE. While one can easily obtain an orthogonal matrix by performing a QR-decomposition on A and then extracting the Q matrix, Mezzadri [2007] and others have observed that in most software implementations, this algorithm usually fails to actually sample from the Haar measure. The reason for this is rather subtle: most numerical linear algebra packages (including the routines within the LAPACK library used by `R`) implement the QR-algorithm by producing the orthogonal matrix with Householder transformations instead of the Gram-Schmidt procedure, which can be numerically unstable;. However, this procedure does not uniquely determine the matrices Q and R of the QR-decomposition. Indeed, if (Q, R) is such a decomposition, then so is $(Q\Lambda, \Lambda^{-1}R)$, where Λ is any unitary diagonal matrix. We implemented the remedy, according to Mezzadri [2007], of obtaining the correct decomposition from *any* QR-algorithm by choosing $Q' = Q\Lambda$, where $\Lambda = \mathrm{diag}(R_{11}/|R_{11}|, \ldots, R_{nn}/|R_{nn}|)$.

Because the CUE eigenvalues are constrained to the unit circle, we can visualize the joint density of the eigenvalues by mapping each eigenvalue to its argument in the interval $[-\pi, \pi]$, rather than attempting to plot the eigenvalues directly in $\mathbb{C}$. In Figure 2 (top), we plotted $\mathbb{A}_n(\theta, \phi) := \mathbb{K}_n(e^{i\theta}, e^{i\phi})$ for $(\theta, \phi) \in [-\pi, \pi]^2$, again discretized to a lattice of 100^2 points. Figure 2 (bottom) shows the corresponding scatterplot of 10,000 pairs of simulated arguments (θ, ϕ). As with the GUE example, we observed a repulsion across the $y = x$ axis, with mass now concentrated along the bands $y = x + \pi$ and $y = x - \pi$, where the arguments are as far apart as possible. To demonstrate more vividly the repulsion between eigenvalues, we also show in Figure 3 a sample of 200 eigenvalue pairs in $\mathbb{C}$, where the second eigenvalue in each pair is (approximately) the distinguished thick point in red.

For the truncated unitary ensemble, shown in Figure 4, we again illustrated the distributions of pairs of eigenvalues with two distinguished points: $(0, 0)$ and a randomly-chosen point in the unit disk S^1, where we



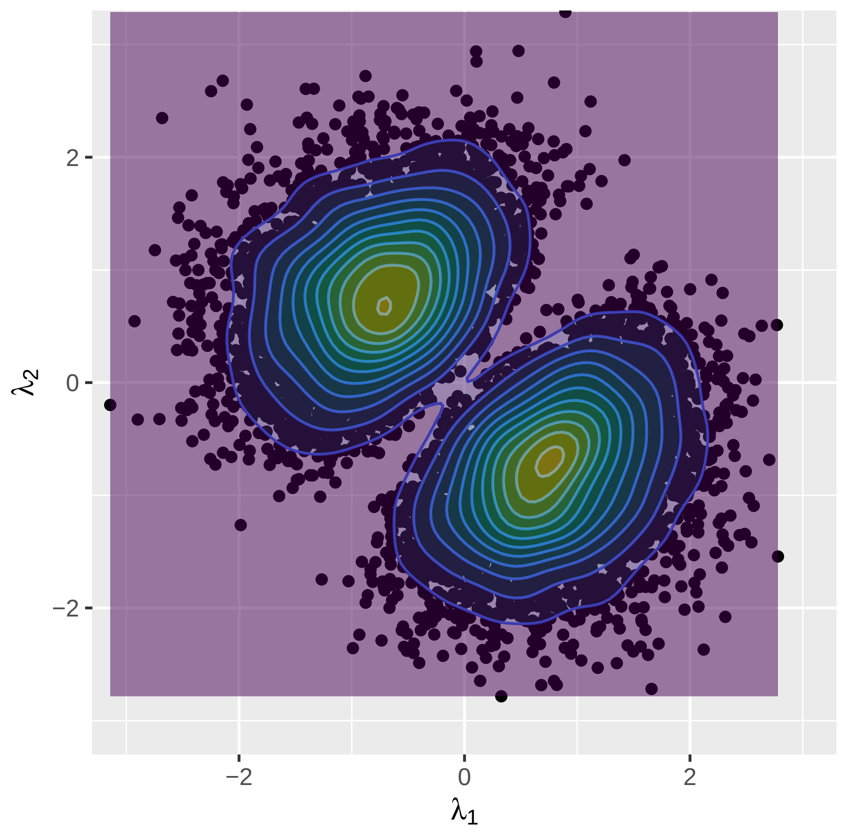
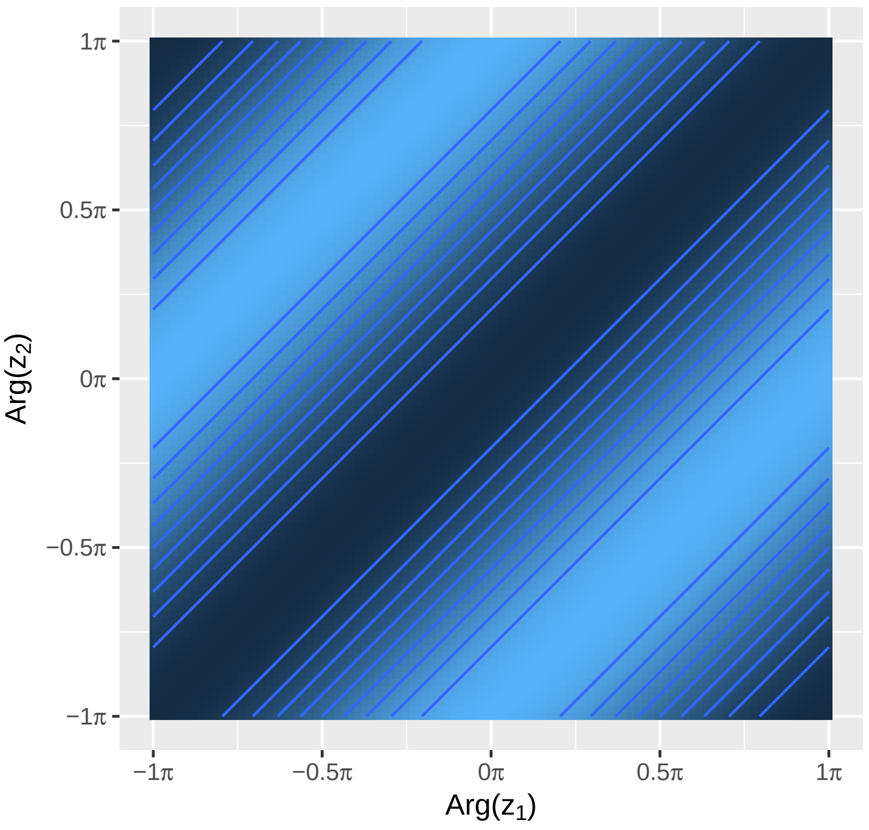


Figure 1: Top: joint density of randomly-chosen pairs of eigenvalues of GUE matrices. Bottom: 10,000 randomly-chosen pairs of eigenvalues of GUE matrices



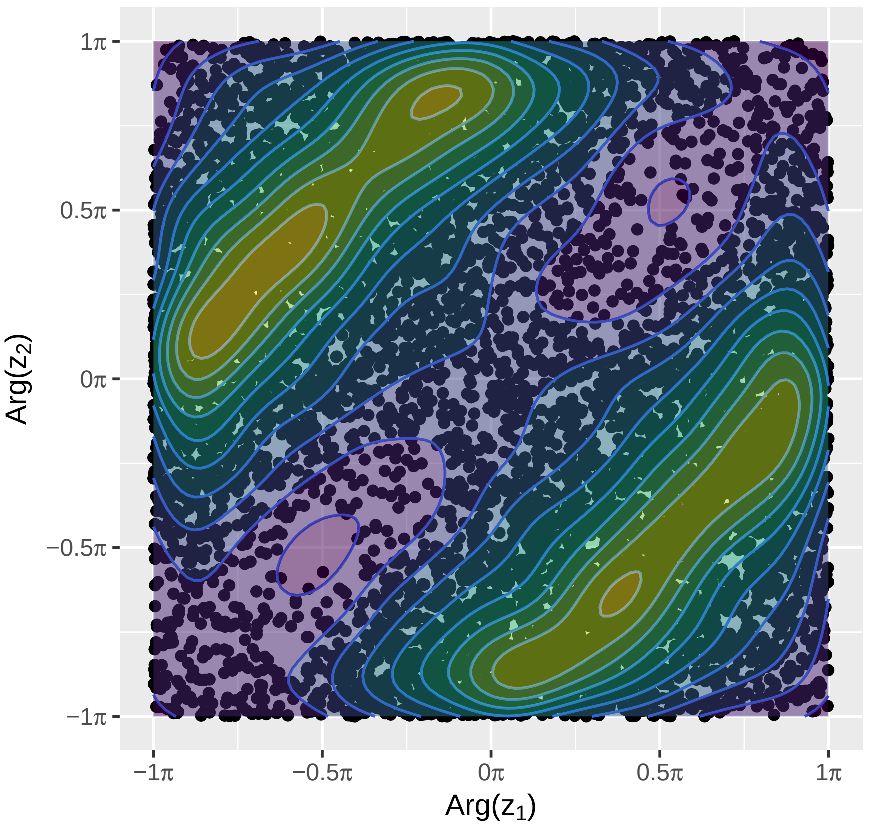


Figure 2: Top: joint density of arguments of randomly-chosen pairs of eigenvalues of CUE matrices. Bottom: arguments of 10,000 randomly-chosen pairs of eigenvalues of CUE matrices

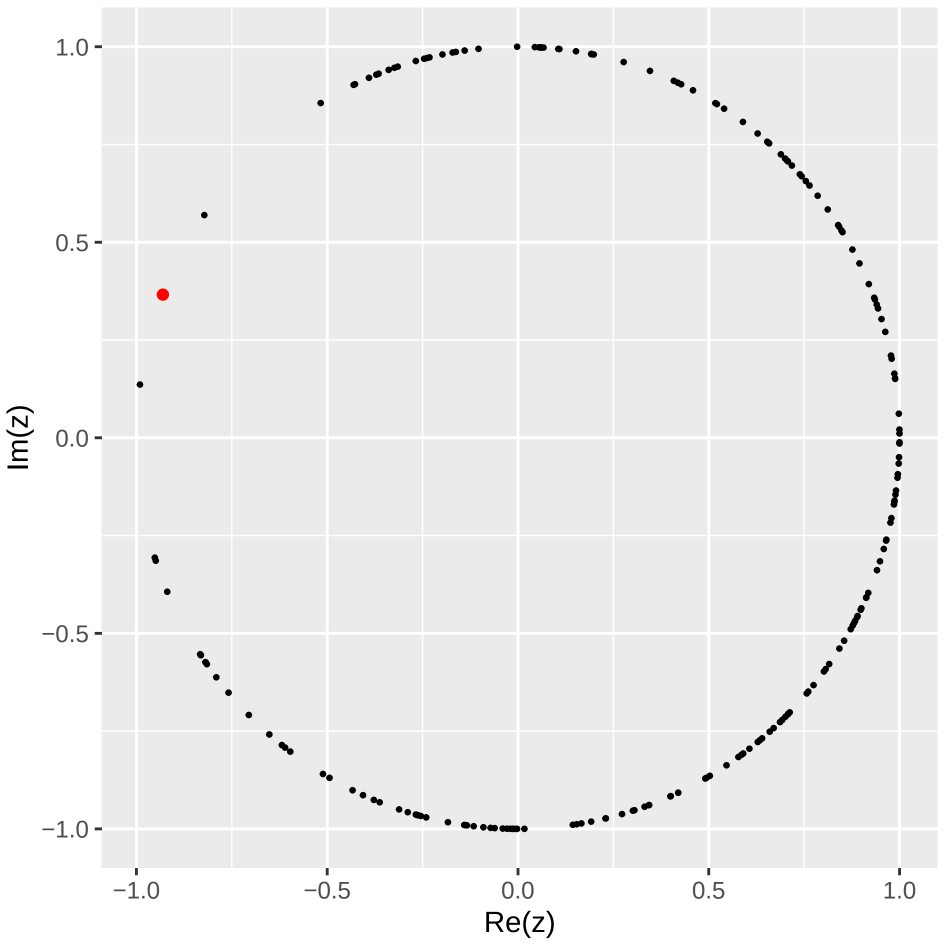


Figure 3: 200 random eigenvalue pairs of CUE matrices, for which the second eigenvalue in each pair is the red point

have chosen $n = 5$ and $m = 2$. For the distinguished point $(0, 0)$, we observed a stronger repulsion away from the origin, concentrating more eigenvalues towards S^1.

We note in passing that Zyczkowski and Sommers [2000] take an alternative path to illustrating the distribution of these eigenvalues by studying the joint distribution of the moduli of the eigenvalues, which works due to rotational symmetry of these particular matrices.

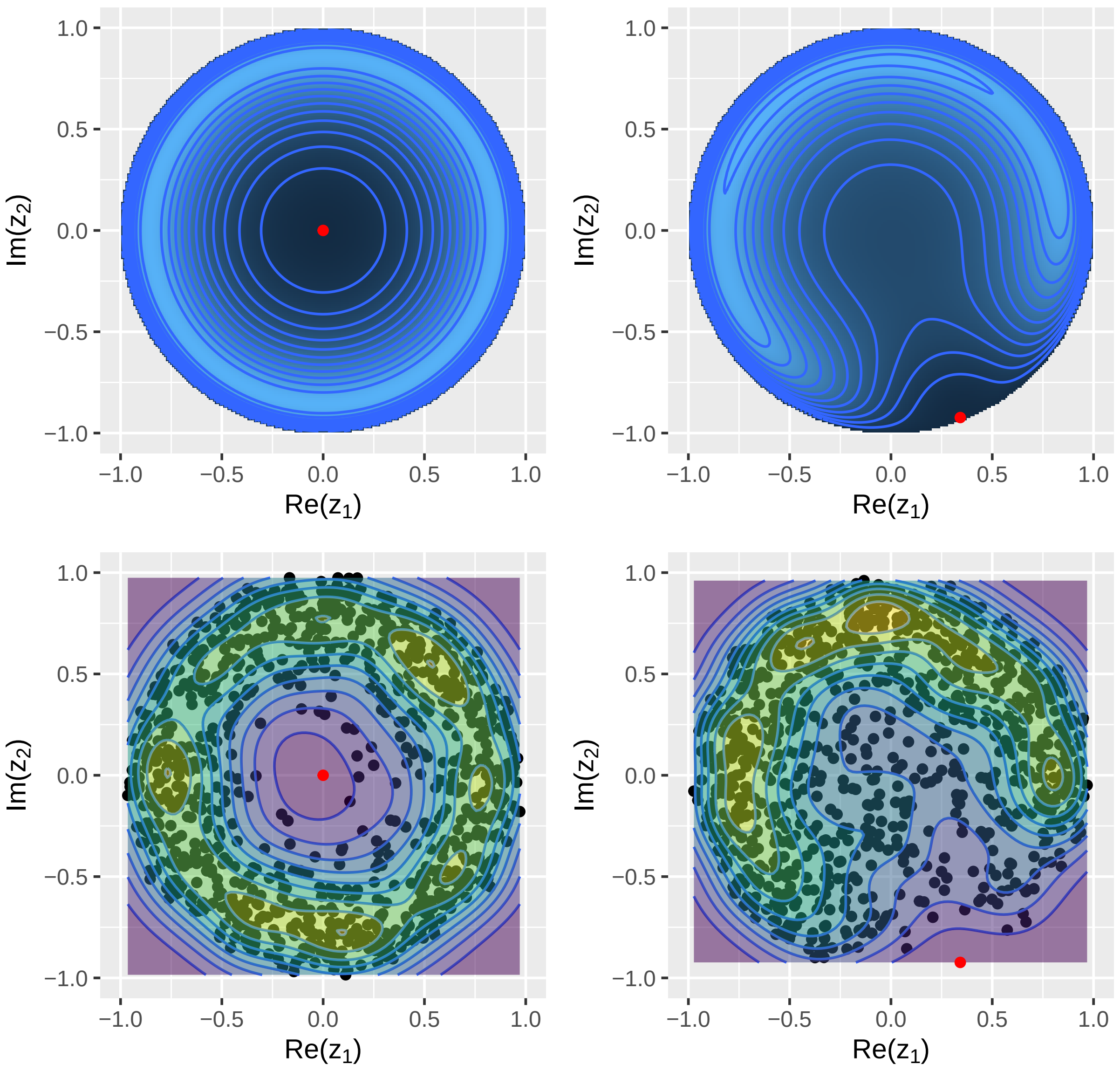


Figure 4: Left: Joint density (top) and 1,000 random eigenvalue pairs of truncated unitary matrices (bottom), for which the second eigenvalue in each pair is the origin. Right: Joint density (top) and 1,000 random eigenvalue pairs of truncated unitary matrices (bottom), for which the second eigenvalue in each pair is the red point

3.3 Complex Ginibre Ensemble and Spherical Ensemble

Sampling a matrix from the complex Ginibre ensemble is trivial. In order to observe how the rotational symmetry of the repulsion changes when the repulsion does not emanate from the origin, we plotted "slices" for two distinguished points as described above. Indeed, Figure 5 (left) shows how the first eigenvalues concentrate evenly along the unit circle S^1. By contrast, we observe in Figure 5 (right) how even a slight deviation of the distinguished point from the center results in a rather "nonisotropic" repulsion.

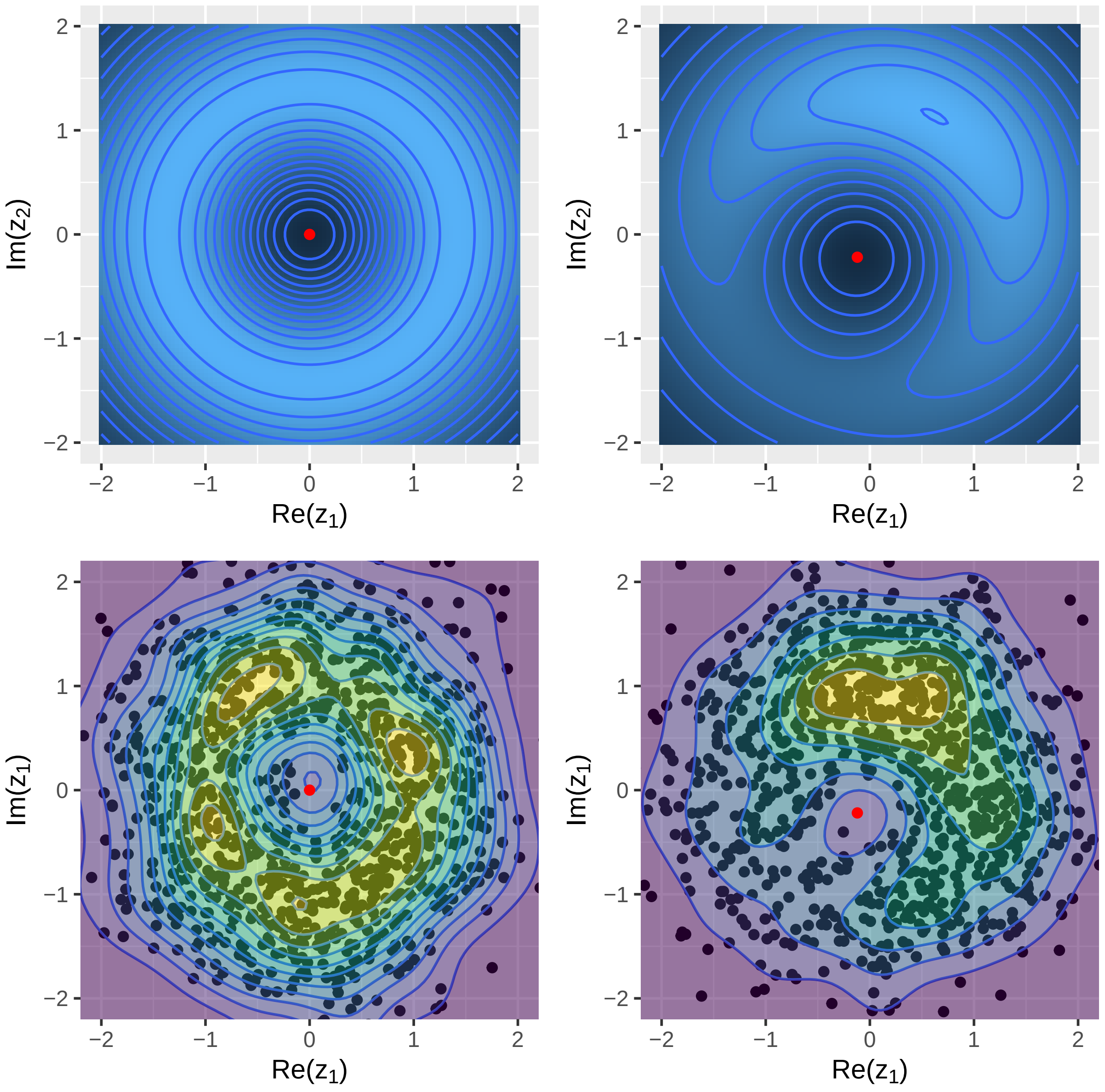


Figure 5: Left: Joint density (top) and 1,000 random eigenvalue pairs of complex Ginibre matrices (bottom), for which the second eigenvalue in each pair is the origin. Right: Joint density (top) and 1,000 random eigenvalue pairs of complex Ginibre matrices (bottom), for which the second eigenvalue in each pair is the red point

Sampling from the spherical ensemble is again trivial. Figure 6 shows that the first eigenvalues appear to concentrate more tightly in unoccupied regions of S^1 than do those from the complex Ginibre ensemble.

3.4 Non-Intersecting Random Walks

To sample midway configurations of non-intersecting random walks, we first simulated random walks themselves. We considered $n = 3$ simple symmetric random walks $X_1(t), X_2(t), X_3(t)$ started at $(-2, 0, 2)$ at time 0, and allowed them to move for 6 steps. After initializing the starting position, for each $t = 1, \ldots, 5$ we sampled a

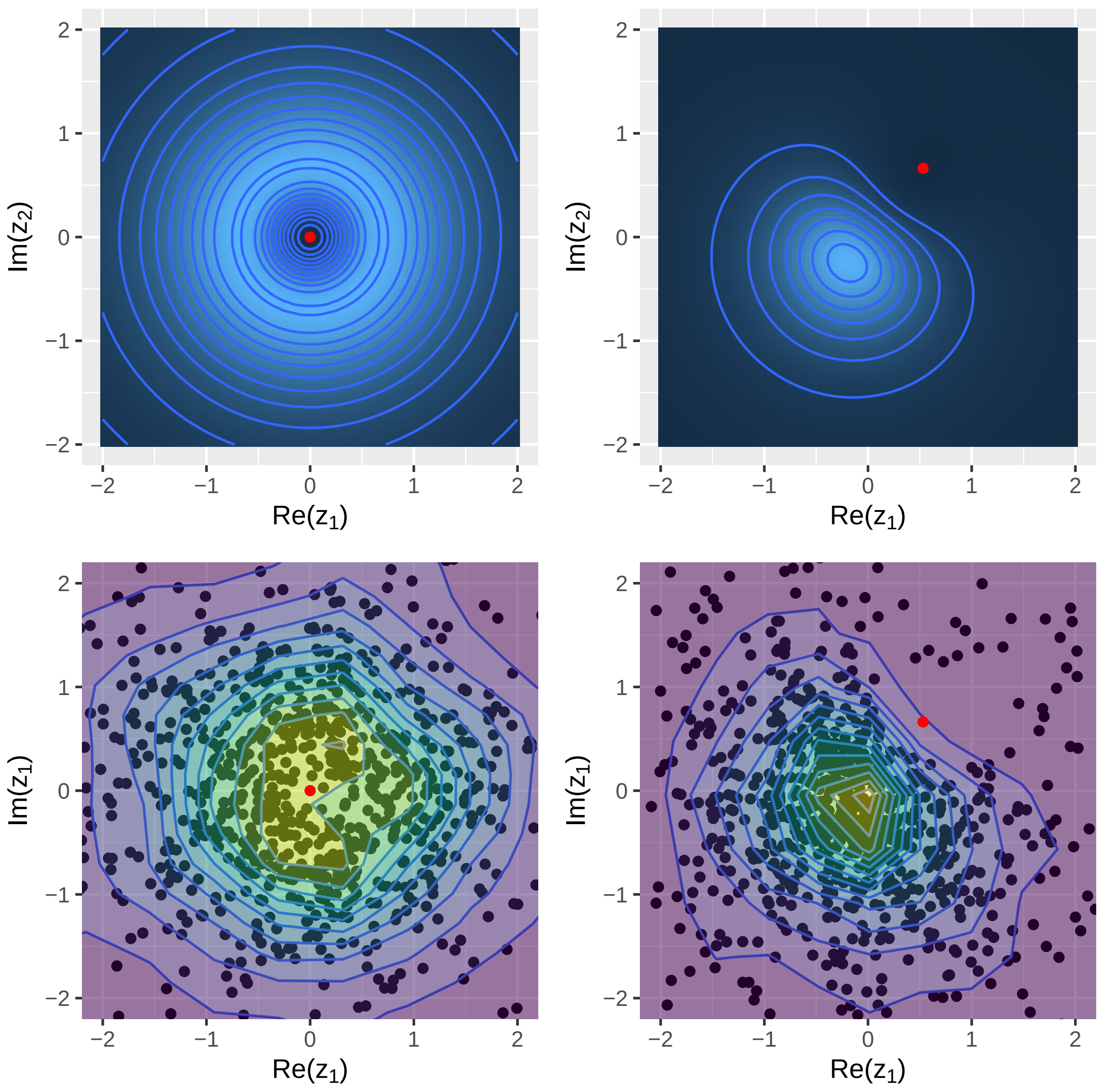


Figure 6: Left: Joint density (top) and 1,000 random eigenvalue pairs of matrices from the spherical ensemble (bottom), for which the second eigenvalue in each pair is the origin. Right: Joint density (top) and 1,000 random eigenvalue pairs of matrices from the spherical ensemble (bottom), for which the second eigenvalue in each pair is the red point

candidate $t+1$'th configuration $(X'_p(t+1) = X_p(t) + R_p)_{1 \le p \le 3}$ where the R_p's are independent Rademacher (i.e., Unif($\{-1, 1\}$)) random variables, and accepted the sample provided that $X'_1(t{+}1) < X'_2(t{+}1) < X'_3(t{+}1)$; otherwise, we sampled a fresh candidate. At $t = 6$, we checked whether $(X_1(6), X_2(6), X_3(6)) = (-2, 0, 2)$. If this condition was met, we accepted the entire random walk as a sample. Otherwise, we started a fresh random walk from the initial configuration and tried again. We repeated this procedure 5,000 times; we show four samples in Figure 7, highlighting in yellow the midway configurations of interest.

In Figure 8, we show all 20 possible midway configurations for this setup (the "direction of travel" is now up). We compared the frequencies of each of these configurations among the 5,000 simulated non-intersecting random walks to the their probabilities as predicted by the determinantal formula. Unfortunately, our observed frequencies did not completely align with the theoretical probabilities, even with a very large number of simulations; we suspect that there may be a subtle flaw in our random walk sampler.

Configuration	Predicted	Observed
C1	0.2296	0.2026
C2	0.2296	0.1968
C3	0.1020	0.1962
C4	0.1020	0.1896
C5	0.0653	0.0388
C6	0.0653	0.0318
C7	0.0367	0.0278
C8	0.0367	0.0272
C9	0.0367	0.0228
C10	0.0367	0.0200
C11	0.0092	0.0124
C12	0.0092	0.0094
C13	0.0092	0.0052
C14	0.0092	0.0048
C15	0.0092	0.0046
C16	0.0092	0.0044
C17	0.0010	0.0032
C18	0.0010	0.0020
C19	0.0010	0.0002
C20	0.0010	0.0002

4 Summary

Determinantal processes are useful to model repulsive point processes; these processes are especially tractable because determinants allow relatively simple calculations of joint densities. We simulated bivariate realizations of several determinantal processes and compared the resulting scatterplot with surface plots of the associated joint densities predicted by theory. In all but one case, we obtained an excellent correlation between the simulations and the theory, and we were able to observe clearly the repulsion that is so characteristic of these processes.

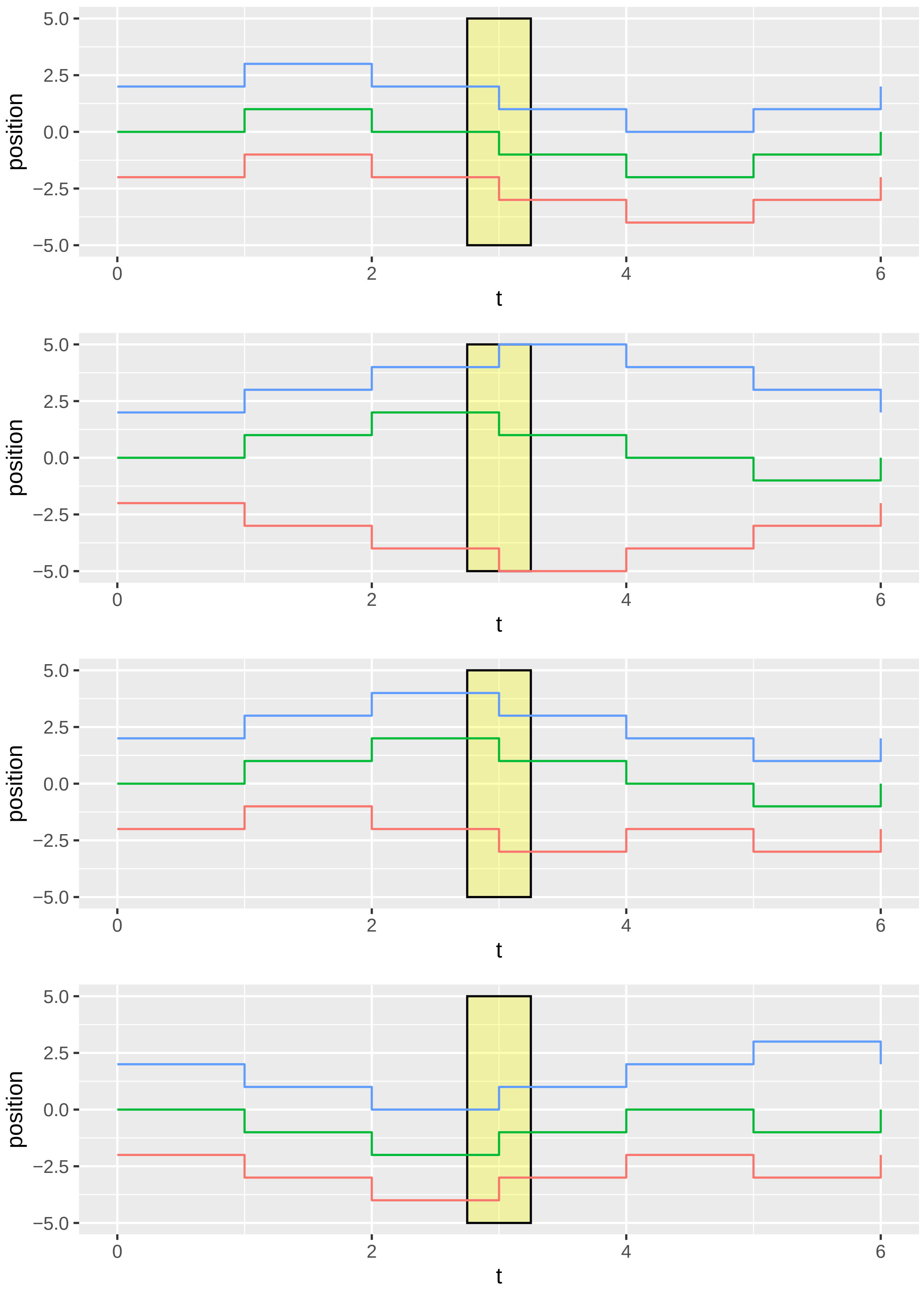


Figure 7: Four simulations of three simple symmetric random walks conditioned not to intersect and to end at their own starting configurations. We are interested in the distribution of midway configurations (highlighted in yellow).

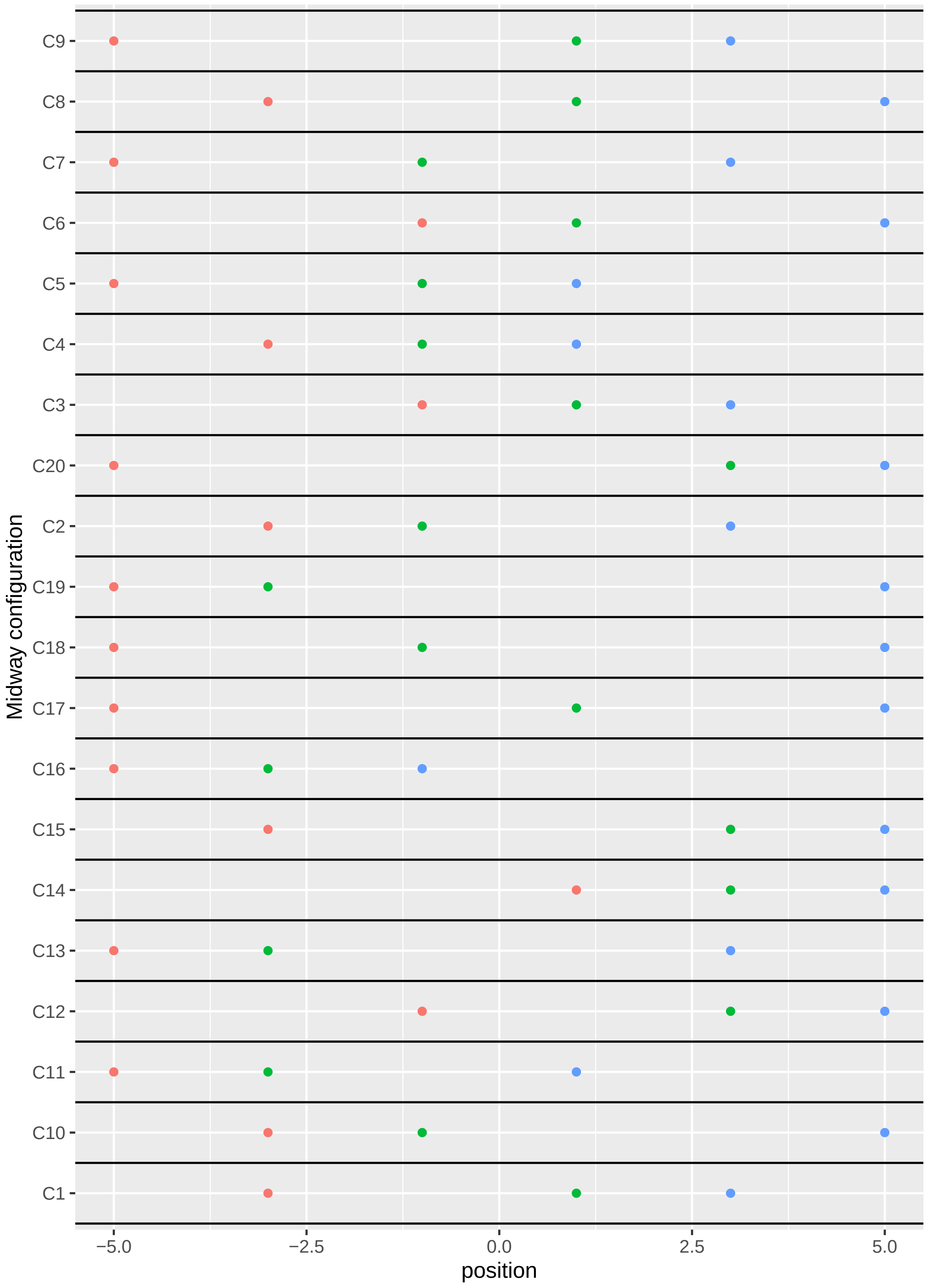


Figure 8: All midway configurations at $t = 3$ for three simple random walks started at $(-2, 0, 2)$.

5 Appendix: R Code

```r
knitr::opts_chunk$set(echo = FALSE, cache = TRUE, warning = FALSE, message = FALSE,
    fig.pos = "!H")
library(EQL)
library(tidyverse)
library(magick)
library(mapproj)
library(expm)
library(tensr)
library(patchwork)
library(scales)
library(kableExtra)

n <- 2  # number of rows/columns
xmin <- -3
xmax <- 3
pts <- 100
ax <- seq(from = xmin, to = xmax, length.out = pts)
ay <- seq(from = xmin, to = xmax, length.out = pts)

# build up kernel

Kn <- function(x, y) {
    s <- 0
    for (k in 0:(n - 1)) {
        s <- s + hermite(x = x, n = k) * hermite(x = y, n = k)
    }
    return(s * dnorm(x = x) * dnorm(x = y))
}

Kgrid <- expand.grid(x = ax, y = ay)
dens <- vector("numeric", length = pts^2)

for (i in 1:pts^2) {
    K <- matrix(0L, nrow = n, ncol = n)
    pt <- as.numeric(Kgrid[i, ])
    for (p in 1:n) {
        for (q in 1:n) {
            K[p, q] <- Kn(pt[p], pt[q])
        }
    }
    dens[i] <- det(K) * dnorm(pt[p]) * dnorm(pt[q])
}

Kgrid$density <- dens/sum(dens)  # renormalize

gplot <- ggplot(data = Kgrid, aes(x = x, y = y, z = density)) + geom_raster(aes(fill = density)) +
    geom_contour() + xlab(expression(lambda[1])) + ylab(expression(lambda[2])) +
    theme(legend.position = "none") + coord_fixed(xlim = c(xmin, xmax),
    ylim = c(xmin, xmax))

# simulate some GUE eigenvalues
```

```r
lambdas <- matrix(0L, nrow = pts^2, ncol = 2)

for (i in 1:pts^2) {
    A <- matrix(as.complex(0), nrow = n, ncol = n)
    for (p in 1:n) {
        for (q in 1:n) {
            A[p, q] <- rnorm(n = 1, mean = 0, sd = 1/sqrt(2)) + (0+1i) *
                rnorm(n = 1, mean = 0, sd = 1/sqrt(2))
        }
    }
    H <- (A + Conj(t(A)))/2
    lambdas[i, ] <- sample(eigen(H)$values, size = n, replace = F)
}
Pgrid <- data.frame(lambda1 = lambdas[, 1], lambda2 = lambdas[, 2])

gplotemp <- ggplot(data = Pgrid, aes(x = lambda1, y = lambda2)) + geom_point() +
    geom_density_2d() + geom_density_2d_filled(alpha = 0.5) + xlab(expression(lambda[1])) +
    ylab(expression(lambda[2])) + theme(legend.position = "none") + coord_fixed(xlim = c(xmin,
    xmax), ylim = c(xmin, xmax))

gplot/gplotemp


# simulate some CUE eigenvalues
lambdas.CUE <- matrix(0L, nrow = pts^2, ncol = 2)
thetas.CUE <- matrix(0L, nrow = pts^2, ncol = 2)

pi_scales <- math_format(.x * pi, format = function(x) x/pi)

for (i in 1:pts^2) {
    A.CUE <- matrix(as.complex(0), nrow = n, ncol = n)
    for (p in 1:n) {
        for (q in 1:n) {
            A.CUE[p, q] <- rnorm(n = 1, mean = 0, sd = 1/sqrt(2)) + (0+1i) *
                rnorm(n = 1, mean = 0, sd = 1/sqrt(2))
        }
    }
    qr <- qr(A.CUE)
    Q <- qr.Q(qr)  # Q-matrix from QR-decomposition
    R <- qr.R(qr)  # R-matrix ''
    D <- diag(R)  # diagonal of R-matrix
    Dnorm <- diag(D * 1/abs(D))
    H.CUE <- Q %*% Dnorm %*% Q
    lambdas.CUE[i, ] <- sample(eigen(H.CUE)$values, size = n, replace = F)  # eigenvalues
    thetas.CUE[i, ] <- Arg(lambdas.CUE[i, ])  # Args of eigenvalues
}
Pgrid.CUE <- data.frame(theta1 = thetas.CUE[, 1], theta2 = thetas.CUE[,
    2])

gplotemp.CUE <- ggplot(data = Pgrid.CUE, aes(x = theta1, y = theta2)) +
    geom_point() + geom_density_2d() + geom_density_2d_filled(alpha = 0.5) +
    xlab(expression(paste("Arg(", z[1], ")"))) + ylab(expression(paste("Arg(",
    z[2], ")"))) + theme(legend.position = "none") + scale_x_continuous(labels = pi_scales,
    breaks = seq(-pi, pi, pi/2)) + scale_y_continuous(labels = pi_scales,
    breaks = seq(-pi, pi, pi/2)) + coord_fixed(xlim = c(-pi, pi), ylim = c(-pi,
    pi))


# build up kernel

n <- 2  # number of rows/columns
xmin.CUE <- -pi
xmax.CUE <- pi
pts <- 100
seq.CUE <- seq(from = xmin.CUE, to = xmax.CUE, length.out = pts)

Kn.CUE <- function(z, w) {
    theta <- Arg(z)
    psi <- Arg(w)
    s <- 0 + 0 * (0+1i)
    for (k in 0:(n - 1)) {
        s <- s + exp((0+1i) * k * theta - (0+1i) * k * psi)
    }
    return(s/(2 * pi))
}

Kgrid.CUE <- expand.grid(Arg1 = seq.CUE, Arg2 = seq.CUE)  # ANGLES
dens.CUE <- vector("numeric", length = length(Kgrid.CUE[, 1]))

for (i in 1:length(Kgrid.CUE[, 1])) {
    K.CUE <- matrix(0L, nrow = n, ncol = n)
    pt <- as.vector(exp((0+1i) * Kgrid.CUE[i, ]), mode = "complex")
    for (p in 1:n) {
        for (q in 1:n) {
            K.CUE[p, q] <- Kn.CUE(pt[p], pt[q])
        }
    }
    dens.CUE[i] <- prod(eigen(K.CUE)$values)
}

Kgrid.CUE$density <- dens.CUE/sum(dens.CUE)

gplot.CUE <- ggplot(data = Kgrid.CUE, aes(x = Arg1, y = Arg2, z = density)) +
    geom_raster(aes(fill = density)) + geom_contour() + xlab(expression(paste("Arg(",
    z[1], ")"))) + ylab(expression(paste("Arg(", z[2], ")"))) + theme(legend.position = "none") +
    scale_x_continuous(labels = pi_scales, breaks = seq(-pi, pi, pi/2)) +
    scale_y_continuous(labels = pi_scales, breaks = seq(-pi, pi, pi/2)) +
    coord_fixed(xlim = c(-pi, pi), ylim = c(-pi, pi))
gplot.CUE/gplotemp.CUE

sims <- 200
ptct <- 0
d <- 4
eps <- 0.1

w <- exp((0+1i) * runif(n = 2, min = 0, max = 2 * pi))
```

```r
lambdas.CUE <- vector(length = sims)

while (any(ptct < sims)) {
    A.CUE <- matrix(0 + (0+1i) * 0, nrow = d, ncol = d)
    for (p in 1:d) {
        for (q in 1:d) {
            A.CUE[p, q] <- rnorm(n = 1, mean = 0, sd = 1/sqrt(2)) + (0+1i) *
                rnorm(n = 1, mean = 0, sd = 1/sqrt(2))
        }
    }
    qr <- qr(A.CUE)
    Q <- qr.Q(qr)  # Q-matrix from QR-decomposition
    R <- qr.R(qr)  # R-matrix ''
    D <- diag(R)  # diagonal of R-matrix
    Dnorm <- diag(D * 1/abs(D))
    A <- Q %*% Dnorm
    eigs <- eigen(A)$values  # eigenvalues

    # check if any of the eigenvalues are in a neighborhood of our
    # distinguished points if so, then sample any OTHER eigenvalue at
    # random, otherwise discard
    if (abs(eigs - w) < eps && ptct < sims) {
        lambdas.CUE[ptct + 1] <- sample(eigs[-which(abs(eigs - w) < eps)[1]],
            size = 1)  # index of first eigenvalue in eps-ball of w_i
        ptct <- ptct + 1
    }
}

Ptgrid.CUE <- data.frame(z = lambdas.CUE)

ptplot.CUE <- ggplot(data = Ptgrid.CUE, aes(x = Re(z), y = Im(z))) + geom_point(size = 0.5) +
    annotate("point", x = Re(w[1]), y = Im(w[1]), color = "red")

ptplot.CUE

N <- 5
m <- 2
d <- N + m

xmin.TU <- -1
xmax.TU <- 1

# simulate some truncated unitary eigenvalues
sims <- 1000
ptct <- c(0, 0)

w.TU <- c(0 + (0+1i) * 0, runif(n = 1, min = 0, max = 1) * exp((0+1i) *
    runif(n = 1, min = 0, max = 2 * pi)))

eps <- 0.1

lambdas.TU <- matrix(0L, nrow = sims, ncol = 2)
```

```r
while (any(ptct < sims)) {
    A <- matrix(0 + (0+1i) * 0, nrow = d, ncol = d)
    for (p in 1:d) {
        for (q in 1:d) {
            A[p, q] <- rnorm(n = 1, mean = 0, sd = 1/sqrt(2)) + (0+1i) *
                rnorm(n = 1, mean = 0, sd = 1/sqrt(2))
        }
    }
    qr <- qr(A)
    Q <- qr.Q(qr)  # Q-matrix from QR-decomposition
    R <- qr.R(qr)  # R-matrix from QR-decomposition
    D <- diag(R)  # diagonal of R-matrix
    D <- diag(D * 1/abs(D))
    A.TU <- Q %*% D
    U.TU <- A.TU[1:N, 1:N]  # NxN principal submatrix
    eigs <- eigen(U.TU)$values  # eigenvalues

    for (i in 1:2) {
        # check if any of the eigenvalues are in a neighborhood of our
        # distinguished points if so, then sample any OTHER eigenvalue at
        # random, otherwise discard
        if (any(abs(eigs - w.TU[i]) < eps) && ptct[i] < sims) {
            lambdas.TU[ptct[i] + 1, i] <- sample(eigs[-which(abs(eigs -
                w.TU[i]) < eps)[1]], size = 1)  # index of first eigenvalue in eps-ball of w_i
            ptct[i] <- ptct[i] + 1
        }
    }
}

colnames(lambdas.TU) <- c("p1", "p2")
Ptgrid.TU <- as.data.frame(lambdas.TU)

ptplot.TU.1 <- ggplot(data = Ptgrid.TU, aes(x = Re(p1), y = Im(p1))) +
    geom_point() + geom_density_2d() + geom_density_2d_filled(alpha = 0.5) +
    xlab(expression(paste("Re(", z[1], ")"))) + ylab(expression(paste("Im(",
    z[2], ")"))) + annotate("point", x = Re(w.TU[1]), y = Im(w.TU[1]),
    color = "red") + theme(legend.position = "none") + coord_fixed(xlim = c(xmin.TU,
    xmax.TU), ylim = c(xmin.TU, xmax.TU))
ptplot.TU.2 <- ggplot(data = Ptgrid.TU, aes(x = Re(p2), y = Im(p2))) +
    geom_point() + geom_density_2d() + geom_density_2d_filled(alpha = 0.5) +
    xlab(expression(paste("Re(", z[1], ")"))) + ylab(expression(paste("Im(",
    z[2], ")"))) + annotate("point", x = Re(w.TU[2]), y = Im(w.TU[2]),
    color = "red") + theme(legend.position = "none") + coord_fixed(xlim = c(xmin.TU,
    xmax.TU), ylim = c(xmin.TU, xmax.TU))


# build up kernel

n <- 2  # number of rows/columns
pts <- 200
ax.TU <- seq(from = xmin.TU, to = xmax.TU, length.out = pts)
ay.TU <- seq(from = xmin.TU, to = xmax.TU, length.out = pts)
```

```r
Kn.TU <- function(z, w) {
    s <- 1
    for (k in 1:(N - 1)) {
        s <- s + (prod(1:k + m)/factorial(k)) * (z * Conj(w))^k
    }
    return(s)
}

Kgrid.TU <- expand.grid(Re = ax.TU, Im = ay.TU)  # complex grid
Kgrid.TU <- Kgrid.TU[sqrt(Kgrid.TU$Re^2 + Kgrid.TU$Im^2) < 1, ]
dens.TU.1 <- vector("numeric", length = length(Kgrid.TU[, 1]))
dens.TU.2 <- vector("numeric", length = length(Kgrid.TU[, 1]))

for (i in 1:length(Kgrid.TU[, 1])) {
    # build up kernel
    K.TU.1 <- matrix(0L, nrow = n, ncol = n)
    K.TU.2 <- matrix(0L, nrow = n, ncol = n)
    pt1 <- c(Kgrid.TU$Re[i] + (0+1i) * Kgrid.TU$Im[i], w.TU[1])
    pt2 <- c(Kgrid.TU$Re[i] + (0+1i) * Kgrid.TU$Im[i], w.TU[2])
    for (p in 1:n) {
        for (q in 1:n) {
            K.TU.1[p, q] <- Kn.TU(pt1[p], pt1[q])
            K.TU.2[p, q] <- Kn.TU(pt2[p], pt2[q])
        }
    }
    dens.TU.1[i] <- prod(eigen(K.TU.1)$values) * (1 - abs(pt1[1])^2)^(m -
        1) * (1 - abs(pt1[2])^2)^(m - 1)  # determinant
    dens.TU.2[i] <- prod(eigen(K.TU.2)$values) * (1 - abs(pt2[1])^2)^(m -
        1) * (1 - abs(pt2[2])^2)^(m - 1)
}

Kgrid.TU$density.1 <- dens.TU.1/sum(dens.TU.1)  # normalize
Kgrid.TU$density.2 <- dens.TU.2/sum(dens.TU.2)

gplot.TU.1 <- ggplot(data = Kgrid.TU, aes(x = Re, y = Im, z = density.1)) +
    geom_raster(aes(fill = density.1)) + geom_contour() + xlab(expression(paste("Re(",
    z[1], ")"))) + ylab(expression(paste("Im(", z[2], ")"))) + annotate("point",
    x = Re(w.TU[1]), y = Im(w.TU[1]), color = "red") + theme(legend.position = "none") +
    coord_fixed() + xlim(xmin.TU, xmax.TU) + ylim(xmin.TU, xmax.TU)
gplot.TU.2 <- ggplot(data = Kgrid.TU, aes(x = Re, y = Im, z = density.2)) +
    geom_raster(aes(fill = density.2)) + geom_contour() + xlab(expression(paste("Re(",
    z[1], ")"))) + ylab(expression(paste("Im(", z[2], ")"))) + annotate("point",
    x = Re(w.TU[2]), y = Im(w.TU[2]), color = "red") + theme(legend.position = "none") +
    coord_fixed() + xlim(xmin.TU, xmax.TU) + ylim(xmin.TU, xmax.TU)

(gplot.TU.1/ptplot.TU.1) | (gplot.TU.2/ptplot.TU.2)

# Ginibre ensemble

sims <- 1000
ptct <- c(0, 0)

xmin.GE <- -2
```

```r
xmax.GE <- 2

w1 <- 0 + (0+1i) * 0
w.GE <- runif(n = 1, min = 0, max = 1) * exp((0+1i) * runif(n = 1, min = 0,
    max = 2 * pi))
w <- c(w1, w.GE)

eps <- 0.1

lambdas.GE <- matrix(0L, nrow = sims, ncol = 2)

while (any(ptct < sims)) {
    A.GE <- matrix(as.complex(0), nrow = n, ncol = n)
    for (p in 1:n) {
        for (q in 1:n) {
            A.GE[p, q] <- rnorm(n = 1, mean = 0, sd = 1/sqrt(2)) + (0+1i) *
                rnorm(n = 1, mean = 0, sd = 1/sqrt(2))
        }
    }
    eigs <- eigen(A.GE)$values

    for (i in 1:2) {
        # check if any of the eigenvalues are in a neighborhood of our
        # distinguished points if so, then sample any OTHER eigenvalue at
        # random, otherwise discard
        if (any(abs(eigs - w[i]) < eps) && ptct[i] < sims) {
            lambdas.GE[ptct[i] + 1, i] <- sample(eigs[-which(abs(eigs -
                w[i]) < eps)[1]], size = 1)  # index of first eigenvalue in eps-ball of w_i
            ptct[i] <- ptct[i] + 1
        }
    }
}

colnames(lambdas.GE) <- c("p1", "p2")
Ptgrid.GE <- as.data.frame(lambdas.GE)

ptplot.GE.1 <- ggplot(data = Ptgrid.GE, aes(x = Re(p1), y = Im(p1))) +
    geom_point() + geom_density_2d() + geom_density_2d_filled(alpha = 0.5) +
    xlab(expression(paste("Re(", z[1], ")"))) + ylab(expression(paste("Im(",
    z[1], ")"))) + annotate("point", x = Re(w[1]), y = Im(w[1]), color = "red") +
    theme(legend.position = "none") + coord_fixed(xlim = c(xmin.GE, xmax.GE),
    ylim = c(xmin.GE, xmax.GE))
ptplot.GE.2 <- ggplot(data = Ptgrid.GE, aes(x = Re(p2), y = Im(p2))) +
    geom_point() + geom_density_2d() + geom_density_2d_filled(alpha = 0.5) +
    xlab(expression(paste("Re(", z[1], ")"))) + ylab(expression(paste("Im(",
    z[1], ")"))) + annotate("point", x = Re(w[2]), y = Im(w[2]), color = "red") +
    theme(legend.position = "none") + coord_fixed(xlim = c(xmin.GE, xmax.GE),
    ylim = c(xmin.GE, xmax.GE))


# build up kernel

n <- 2  # number of rows/columns
```

```r
pts <- 100
Kgrid.GE <- seq(from = xmin.GE, to = xmax.GE, length.out = pts)
dens.GE.1 <- vector("numeric", length = length(pts))
dens.GE.2 <- vector("numeric", length = length(pts))

Kn.GE <- function(z, w) {
    s <- 0 + 0 * (0+1i)
    z <- as.complex(z)
    w <- as.complex(w)
    for (k in 0:(n - 1)) {
        s <- s + (z * Conj(w))^k/factorial(k)
    }
    return(s)
}

Kgrid.GE <- expand.grid(Rez = Kgrid.GE, Imz = Kgrid.GE)
dens.GE <- vector("numeric", length = pts^2)

for (i in 1:pts^2) {
    K.GE.1 <- matrix(0L, nrow = n, ncol = n)
    K.GE.2 <- matrix(0L, nrow = n, ncol = n)
    pt1 <- as.vector(c(Kgrid.GE$Rez[i] + (0+1i) * Kgrid.GE$Imz[i], w[1]),
        mode = "complex")
    pt2 <- as.vector(c(Kgrid.GE$Rez[i] + (0+1i) * Kgrid.GE$Imz[i], w[2]),
        mode = "complex")
    for (p in 1:n) {
        for (q in 1:n) {
            K.GE.1[p, q] <- Kn.GE(pt1[p], pt1[q])
            K.GE.2[p, q] <- Kn.GE(pt2[p], pt2[q])
        }
    }
    dens.GE.1[i] <- prod(eigen(K.GE.1)$values) * (1/pi) * exp(-0.5 * Mod(pt1[1])^2) *
        (1/pi) * exp(-0.5 * Mod(pt1[2])^2)
    dens.GE.2[i] <- prod(eigen(K.GE.2)$values) * (1/pi) * exp(-0.5 * Mod(pt2[1])^2) *
        (1/pi) * exp(-0.5 * Mod(pt2[2])^2)
}

Kgrid.GE$density.1 <- dens.GE.1/sum(dens.GE.1)
Kgrid.GE$density.2 <- dens.GE.2/sum(dens.GE.2)

gplot.GE.1 <- ggplot(data = Kgrid.GE, aes(x = Rez, y = Imz, z = density.1)) +
    geom_raster(aes(fill = density.1)) + geom_contour() + xlab(expression(paste("Re(",
    z[1], ")"))) + ylab(expression(paste("Im(", z[2], ")"))) + annotate("point",
    x = Re(w[1]), y = Im(w[1]), color = "red") + theme(legend.position = "none") +
    coord_fixed(xlim = c(xmin.GE, xmax.GE), ylim = c(xmin.GE, xmax.GE))
gplot.GE.2 <- ggplot(data = Kgrid.GE, aes(x = Rez, y = Imz, z = density.2)) +
    geom_raster(aes(fill = density.2)) + geom_contour() + xlab(expression(paste("Re(",
    z[1], ")"))) + ylab(expression(paste("Im(", z[2], ")"))) + annotate("point",
    x = Re(w[2]), y = Im(w[2]), color = "red") + theme(legend.position = "none") +
    coord_fixed(xlim = c(xmin.GE, xmax.GE), ylim = c(xmin.GE, xmax.GE))
(gplot.GE.1/ptplot.GE.1) | (gplot.GE.2/ptplot.GE.2)

# Spherical ensemble
```

```r
n <- 3
sims <- 1000
ptct <- c(0, 0)

xmin.SE <- -2
xmax.SE <- 2

w1 <- 0 + (0+1i) * 0
w.SE <- runif(n = 1, min = 0, max = 1) * exp((0+1i) * runif(n = 1, min = 0,
    max = 2 * pi))
w <- c(w1, w.SE)

eps <- 0.1

lambdas.SE <- matrix(0L, nrow = sims, ncol = 2)

while (any(ptct < sims)) {
    A.SE <- matrix(as.complex(0), nrow = n, ncol = n)
    B.SE <- matrix(as.complex(0), nrow = n, ncol = n)
    for (p in 1:n) {
        for (q in 1:n) {
            A.SE[p, q] <- rnorm(n = 1, mean = 0, sd = 1/sqrt(2)) + (0+1i) *
                rnorm(n = 1, mean = 0, sd = 1/sqrt(2))
            B.SE[p, q] <- rnorm(n = 1, mean = 0, sd = 1/sqrt(2)) + (0+1i) *
                rnorm(n = 1, mean = 0, sd = 1/sqrt(2))
        }
    }
    C.SE <- solve(A.SE) %*% B.SE
    eigs <- eigen(C.SE)$values

    for (i in 1:2) {
        # check if any of the eigenvalues are in a neighborhood of our
        # distinguished points if so, then sample any OTHER eigenvalue at
        # random, otherwise discard
        if (any(abs(eigs - w[i]) < eps) && ptct[i] < sims) {
            lambdas.SE[ptct[i] + 1, i] <- sample(eigs[-which(abs(eigs -
                w[i]) < eps)[1]], size = 1)  # index of first eigenvalue in eps-ball of w_i
            ptct[i] <- ptct[i] + 1
        }
    }
}

colnames(lambdas.SE) <- c("p1", "p2")
Ptgrid.SE <- as.data.frame(lambdas.SE)

ptplot.SE.1 <- ggplot(data = Ptgrid.SE, aes(x = Re(p1), y = Im(p1))) +
    geom_point() + geom_density_2d() + geom_density_2d_filled(alpha = 0.5) +
    xlab(expression(paste("Re(", z[1], ")"))) + ylab(expression(paste("Im(",
    z[1], ")"))) + annotate("point", x = Re(w[1]), y = Im(w[1]), color = "red") +
    theme(legend.position = "none") + coord_fixed(xlim = c(xmin.SE, xmax.SE),
    ylim = c(xmin.SE, xmax.SE))
ptplot.SE.2 <- ggplot(data = Ptgrid.SE, aes(x = Re(p2), y = Im(p2))) +
    geom_point() + geom_density_2d() + geom_density_2d_filled(alpha = 0.5) +
    xlab(expression(paste("Re(", z[1], ")"))) + ylab(expression(paste("Im(",
    z[1], ")"))) + annotate("point", x = Re(w[2]), y = Im(w[2]), color = "red") +
    theme(legend.position = "none") + coord_fixed(xlim = c(xmin.SE, xmax.SE),
    ylim = c(xmin.SE, xmax.SE))


# build up kernel

n <- 2  # number of rows/columns
pts <- 100
Kgrid.SE <- seq(from = xmin.SE, to = xmax.SE, length.out = pts)
dens.SE.1 <- vector("numeric", length = length(pts))
dens.SE.2 <- vector("numeric", length = length(pts))

Kn.SE <- function(z, w) {
    (1 + z * Conj(w))^(n - 1)
}

dmu.SE <- function(z) {
    n/(pi * (1 + abs(z)^2)^(n + 1))
}

Kgrid.SE <- expand.grid(Rez = Kgrid.SE, Imz = Kgrid.SE)
dens.SE <- vector("numeric", length = pts^2)

for (i in 1:pts^2) {
    K.SE.1 <- matrix(0L, nrow = n, ncol = n)
    K.SE.2 <- matrix(0L, nrow = n, ncol = n)
    pt1 <- as.vector(c(Kgrid.SE$Rez[i] + (0+1i) * Kgrid.SE$Imz[i], w[1]),
        mode = "complex")
    pt2 <- as.vector(c(Kgrid.SE$Rez[i] + (0+1i) * Kgrid.SE$Imz[i], w[2]),
        mode = "complex")
    for (p in 1:n) {
        for (q in 1:n) {
            K.SE.1[p, q] <- Kn.SE(pt1[p], pt1[q])
            K.SE.2[p, q] <- Kn.SE(pt2[p], pt2[q])
        }
    }
    dens.SE.1[i] <- prod(eigen(K.SE.1)$values) * dmu.SE(pt1[1]) * dmu.SE(pt1[2])
    dens.SE.2[i] <- prod(eigen(K.SE.2)$values) * dmu.SE(pt2[1]) * dmu.SE(pt2[2])
}

Kgrid.SE$density.1 <- dens.SE.1/sum(dens.SE.1)
Kgrid.SE$density.2 <- dens.SE.2/sum(dens.SE.2)

gplot.SE.1 <- ggplot(data = Kgrid.SE, aes(x = Rez, y = Imz, z = density.1)) +
    geom_raster(aes(fill = density.1)) + geom_contour() + xlab(expression(paste("Re(",
    z[1], ")"))) + ylab(expression(paste("Im(", z[2], ")"))) + annotate("point",
    x = Re(w[1]), y = Im(w[1]), color = "red") + theme(legend.position = "none") +
    coord_fixed(xlim = c(xmin.SE, xmax.SE), ylim = c(xmin.SE, xmax.SE))
gplot.SE.2 <- ggplot(data = Kgrid.SE, aes(x = Rez, y = Imz, z = density.2)) +
    geom_raster(aes(fill = density.2)) + geom_contour() + xlab(expression(paste("Re(",
    z[1], ")"))) + ylab(expression(paste("Im(", z[2], ")"))) + annotate("point",
```

```r
    x = Re(w[2]), y = Im(w[2]), color = "red") + theme(legend.position = "none") +
    coord_fixed(xlim = c(xmin.SE, xmax.SE), ylim = c(xmin.SE, xmax.SE))
(gplot.SE.1/ptplot.SE.1) | (gplot.SE.2/ptplot.SE.2)

ind <- function(v) {
    # takes an array of numbers and returns an array of quoted numbers (to
    # allow for negative indexing)
    sapply(X = as.numeric(v), FUN = deparse)
}

RWplots <- vector("list", length = 4)
plotct <- 1

N <- 3  # number of RWs
TT <- 2 * N  # total time after 0 (should be odd)
p <- 0.5
# step <- function() {return(sample(x=c(-1,1), size=N, replace=T,
# prob=c(p,1-p)))} # random steps up or down for each RW
step <- function() {
    B <- rbinom(n = N, size = 1, prob = p)
    B[which(B == 0)] <- -1
    return(B)
}
sims <- 5000  # number of simulations
x <- c(-2, 0, 2)  # starting/ending points
configs <- matrix(0L, nrow = (max(x) - min(x) + 2 * TT + 1), ncol = sims)  # matrix to store the midway
rownames(configs) <- (min(x) - TT):(max(x) + TT)
j <- 1

while (j <= sims) {
    RW <- matrix(0L, ncol = (TT + 1), nrow = (max(x) - min(x) + 2 * TT +
        1))
    rownames(RW) <- (min(x) - TT):(max(x) + TT)
    colnames(RW) <- 0:TT

    RW[ind(x), "0"] <- 1:N  # initialize RWs to start at x

    for (t in 1:TT) {
        current <- as.numeric(names(which(RW[, ind(t - 1)] != 0)))  # positions where the walks are cur
        new <- current + step()  # new positions
        while (anyDuplicated(new) > 0 || is.unsorted(new)) {
            # keep trying a new step until we get one without an
            # intersection/crossover
            new <- current + step()  # new positions
        }
        RW[ind(new), ind(t)] <- 1:N
    }
    if (identical(RW[, ind(0)], RW[, ind(TT)])) {
        configs[, j] <- RW[, ind(TT/2)]
        j <- j + 1

        if (plotct <= 4) {
            # save some RWs to plot
            paths <- matrix(0L, nrow = N, ncol = TT + 1)
            for (i in 1:N) {
                for (t in 1:(TT + 1)) {
                  paths[i, t] <- as.numeric(which(RW[, t] == i))
                }
            }
            rownames(paths) <- c("X1", "X2", "X3")
            pathsdat <- as.data.frame(t(paths) - 9)
            pathsdat$t <- 0:TT
            pathsdat2 <- gather(pathsdat, key = X, position, X1, X2, X3)

            RWplots[[plotct]] <- ggplot(data = pathsdat2, aes(t, position)) +
                annotate("rect", xmin = 2.75, xmax = 3.25, ymin = -5, ymax = 5,
                  color = "black", fill = "yellow", alpha = 0.3) + geom_step(aes(color = X)) +
                theme(legend.position = "none")
            plotct <- plotct + 1
        }
    }
}

uniques <- unique(configs, MARGIN = 2)
freq <- apply(uniques, MARGIN = 2, function(b) sum(apply(configs, MARGIN = 2,
    function(a) all(a == b))))
configs.ordered <- uniques[, order(freq, decreasing = T)]
prop.ordered <- freq[order(freq, decreasing = T)]/sims

configdat <- matrix(0L, nrow = N, ncol = 20)
for (i in 1:N) {
    for (c in 1:20) {
        configdat[i, c] <- as.numeric(which(configs.ordered[, c] == i)) -
            9
    }
}
colnames(configdat) <- sapply(1:ncol(configs.ordered), function(x) paste("C",
    x, sep = ""))
configdatwide <- gather(as.data.frame(configdat), key = Config, value = "position")
configdatwide$X <- c("X1", "X2", "X3")

configplot <- ggplot(data = configdatwide, aes(x = Config, y = position)) +
    geom_point(aes(color = X)) + theme(legend.position = "none") + geom_vline(xintercept = 0:20 +
    0.5) + xlab("Midway configuration")

configplot <- configplot + coord_flip()

tstep <- function(a, b, t) {
    # t-step transition probabilities
    h <- (t + b - a)/2
    if (h%%1 == 0) {
        return(choose(t, h) * (p^h) * (1 - p)^(t - h))
    } else {
        return(0)
    }
}
```

```r
Pi <- function(z) {
    # stationary measure for both symmetric and non-symmetric RWs on Z
    (p/(1 - p))^z
}

A <- matrix(0L, ncol = N, nrow = N)
for (j in 1:N) {
    for (k in 1:N) {
        A[j, k] <- tstep(x[k], x[j], TT) * 1/Pi(x[k])
    }
}
sqrtAinv <- solve(mhalf(A))

psi <- function(j, r) {
    s <- 0
    for (k in 1:N) {
        s <- s + sqrtAinv[j, k] * (1/Pi(r)) * tstep(x[k], r, TT/2)
    }
    return(s)
}

num_uniques <- length(prop.ordered)
dets <- vector("numeric", length = num_uniques)
for (l in 1:num_uniques) {
    y <- as.numeric(names(which(configs.ordered[, l] != 0)))
    K.RW <- matrix(0L, ncol = N, nrow = N)
    for (i in 1:N) {
        for (j in 1:N) {
            for (k in 1:N) {
                K.RW[i, j] <- K.RW[i, j] + psi(k, y[i]) * psi(k, y[j])
            }
        }
    }
    dets[l] <- det(K.RW) * prod(Pi(y))
}

results <- data.frame(Predicted = dets[order(dets, decreasing = T)], Observed = prop.ordered)
RWplots[[1]]/RWplots[[2]]/RWplots[[3]]/RWplots[[4]]

configplot

results$Configuration <- sapply(1:ncol(configs.ordered), function(x) paste("C",
    x, sep = ""))
kable(results[, c(3, 1, 2)], digits = 4, booktabs = T) %>%
    kable_styling(position = "center")
```

References

Gaëtan Borot. An introduction to random matrix theory. *arXiv preprint arXiv:1710.10792*, 2017.

Freeman J Dyson. Statistical theory of the energy levels of complex systems. i. *Journal of Mathematical Physics*, 3(1):140–156, 1962.

Morris L Eaton. Multivariate statistics: a vector space approach. *JOHN WILEY & SONS, INC., 605 THIRD AVE., NEW YORK, NY 10158, USA, 1983, 512*, 1983.

Jonit Fischmann, Wojciech Bruzda, Boris A Khoruzhenko, Hans-Jürgen Sommers, and Karol Życzkowski. Induced ginibre ensemble of random matrices and quantum operations. *Journal of Physics A: Mathematical and Theoretical*, 45(7):075203, 2012.

Jean Ginibre. Statistical ensembles of complex, quaternion, and real matrices. *Journal of Mathematical Physics*, 6(3):440–449, 1965.

John Ben Hough, Manjunath Krishnapur, Yuval Peres, et al. *Zeros of Gaussian analytic functions and determinantal point processes*, volume 51. American Mathematical Soc., 2009.

Samuel Karlin, James McGregor, et al. Coincidence probabilities. *Pacific Journal of Mathematics*, 9(4): 1141–1164, 1959.

Manjunath Krishnapur. Zeros of random analytic functions. *arXiv preprint math/0607504*, 2006.

Francesco Mezzadri. How to generate random matrices from the classical compact groups. *Notices of the American Mathematical Society*, 54(5):592 – 604, May 2007. ISSN 0002-9920.

Alexander Soshnikov. Determinantal random point fields. *Russian Mathematical Surveys*, 55(5):923, 2000.

Nicolas Tremblay, Simon Barthelmé, and Pierre-Olivier Amblard. Determinantal point processes for coresets. *Journal of Machine Learning Research*, 20(168):1–70, 2019.

Eugene P Wigner. Characteristic vectors of bordered matrices with infinite dimensions i. In *The Collected Works of Eugene Paul Wigner*, pages 524–540. Springer, 1993.

Karol Zyczkowski and Hans-Jürgen Sommers. Truncations of random unitary matrices. *Journal of Physics A: Mathematical and General*, 33(10):2045, 2000.