Part VI

Applications

Chapter 32

Biology

32.1 Random Matrix Theory in the Future of Quantitative and Systems Biology?

[This section was originally written by Bree Aldridge]

32.1.1 Introduction

In the post-genomic era, cell biologists are becoming increasing interested in studying cell functions from a systems viewpoint, where the quantitative properties of signaling, gene, and metabolic networks are evaluated. Advances in techniques to gather large volumes of quantitative biological data and the completion of the Human Genome Project are facilitating the recent fury of activity in systems biology. Biological engineers (bioinformaticists, computational biologists, systems biologicists, etc.) use tools and concepts developed in other engineering disciplines, physics, chemistry, linguistics, and mathematics to help develop their field. Despite the increasingly quantitative nature of biological research, a literature search on Random Matrix Theory and biology will turn up empty! Therefore, the focus here will be to discuss how random matrix theory might become useful in the "new" biology as the field matures in next few years.

32.1.2 Motivation

One of the primary motivations in studying biology from a systems perspective is to study whether or not biological complexity arises from signal processing in addition to signal transduction in networks. The nematode (worm) *C. elegans* is a model organism because it is orderly (the fate of each cell is predictable) and molecular and genetic studies are easily performed on them. Genomic analysis suggests that *C. elegans* have around 20,000 genes while humans have around 30,000 genes. Then the question arises: how can humans have only 50% more genes than the 1 mm long soil nematode? Certainly we have more than 50% of their complexity! While increased complexity is attributed to differential gene expression and post-translational modification of proteins, the bulk of our added complexity and flexibility is attributed to biological networks of increasing complexity. For this reason, the growth of systems biology reflects this change of ideas by studying biological systems as a whole instead of by parts: instead of asking how the cell will behave if we disable a gene, ask how the network and its parts are organized and how they result in the desired biological behavior.

32.1.3 Dynamics

By studying the dynamics of both linear and non-linear biological systems, we can begin to answer three basic questions:

1. How do these networks work?
2. How did they evolve?
3. How can engineer them?
4. How can we manipulate them?

Signaling networks have been modeled as systems of ordinary differential equations, partial differential equations, finite state machines, Boolean networks, Bayesian networks, Markov processes, and electrical circuits. Data is collected by gene expression arrays (analyzed by SVD to cluster experiments or genes into independent molecular modules), fluorescence (to measure the abundance or localization of a particular protein), BIACore (to measure the mass action kinetics of binding), and Western blots (to qualitatively measure how much of a particular form of protein is present.) For the purposes of signaling networks, western blots are the most common form to gather data; however, the measurements are not normalized and are difficult to quantitate. Major strides at "lab-on-a-chip" using microfluidics to gather reliable and reproducible quantitative data are being made and this method will probably replace Western blots.

For simplicity, this discussion will concentrate on signaling networks, although metabolic network modeling is analogous. Further, consider the case of modeling signaling networks with mass-action kinetics (by Michaelis-Menten kinetics) so that the network is represented by a system of ordinary differential equations where the state variables are the concentration of species (different proteins or molecules in the network), which are related to each other by rate constants. In general, for a linear system of the form

$$\dot{x} = Ax$$

and linearized versions of nonlinear systems at fixed points of the form

$$\dot{u} = Au + O(u^2) \qquad \text{where for a fixed point } x^*,\ u = x - x^* \text{ so } O(u^2) \text{ is very small and is neglected.}$$

the trace and determinant of the matrix A (the Jacobian matrix) can be used to draw the nullclines and phase portraits of the systems [434]. A represents the rate and stoichiometric relationships between the state variables. A can also be used to draw bifurcation diagrams. The nullclines, bifurcations, and phase portrait of a biological network paints a qualitative picture on the behavior of the system. See Figure 1 for a classification of fixed points according to $\mathrm{Tr}(A)$ and $\det(A)$.

Let us first consider a simple genetic control system example first proposed by Gilbert (1971) and analyzed in [434]. Let x be the concentration of protein, y the concentration of mRNA corresponding to that protein, a the degradation of protein, and b the degradation of mRNA. Then, the system is represented as

$$\begin{aligned} \dot{x} &= -ax + y \\ \dot{y} &= \frac{x^2}{1+x^2} - by \end{aligned}$$

In order to analyze the dynamics of the system, the nullclines are plotted for different values of a. For a above some critical value ($a < c$ where c is the bifurcation point), there are three fixed points, for $a = c$, there are 2 fixed points and for $a > c$ there is one fixed point. Figure 2 shows how the fixed points change as a is changed. The nullclines intersect at

$$x^* = \frac{1 \pm \sqrt{1 - 4a^2b^2}}{2ab}$$

which gives us the fixed points in addition to $(0, 0)$. Figure 3 shows the phase portrait with 2 fixed stable points (black circles) and a saddle node (empty circle). Therefore, the system is switch-like (bistable) when $a < 1/2b$. Biologically, the switch can be interpreted as operating so that above a threshold of protein concentration, mRNA will be synthesized but below the threshold, mRNA will not be synthesized (there is positive feedback somehow.) The phase portrait is useful because given any set of initial conditions, one can accurately guess how the system will behave.

32.1.4 Example

A recent paper, [48], uses network descriptions borrowed from dynamics, but does not give a detailed analysis. Instead, it serves as an example as a great forerunner to the dynamics studies of biological networks, which will surely follow as data quality improves. Bhalla et al. present the analysis of their model of the mitogen-activated kinase cascade (MAPK cascade) with experimental verification of their findings. Their model consists of a system of 98 ODEs based on mass-action kinetics with parameters chosen from experiments and literature. A simplified schematic of the pathway is given in Figure 4 (reproduced from [48]). The author's initial guess was the levels of MAPK oscillated because of its involvement in a positive and negative feedback loop. Simulations evaluating the amounts of activated MAPK with a stimulus to PDGF-R (a growth factor receptor) reveals that the network could behave in two manners: in the presence of low MKP concentrations, the network was bistable while in the presence of high MKP, the network was monostable (Figure 5 & 6, reproduced from [48]). Because of the crude nature of the western blots (for experimental verification), only qualitative verification of their analysis could be performed; however, as instrumentation development advances, DNA-array-like instruments could be used to produce high-throughput quantitative data. Figure 7 (reproduced from [48]) is a bifurcation diagram in disguise, mapping out the conditions in which the system in bistable or monostable. Additionally, the authors used simulations to show the memory-capabilities of their switch-like network.

The purpose in describing the recent of Bhalla et al. is to show the beginnings of nonlinear dynamics as an analysis tools for biological signaling networks. While nonlinear systems can be modeled and solved numerically, one would have to rerun the simulations at numerous combinations of initial conditions to build a vague picture of the behavior of the network. Phase portraits give a detailed qualitative description of the network behavior at all initial conditions. These descriptions also give clues as to how the topology of the network influences its behavior. I believe its use will increase in rigor and frequency as technology for probing such systems improves in the next few years. Bhalla et al.'s limited analysis reveals a possible reason why the network would have a positive and negative feedback loop: switch like behavior, memory, and flexibility. Perhaps more could be discovered about the organization and behavior of this network under more rigorous analysis. For example, presence of complex eigenvalues in the Jacobian matrix (A) would signify oscillatory behavior. We might ask under what conditions would we see such behavior? What nodes are sensitive and what is their role in the behavior of the matrix?

In addition to using dynamics to analyze real biological networks, nonlinear dynamics is already being applied to engineer networks and network components. Synthetic oscillations and switches

have already been assembled, with the help of nonlinear dynamics for design [132], [69], [84], [467]. Future work in synthetic biology will include joining these small networks to make larger networks that perform more than one function. A combination of learning how these networks work in cells and how we can engineer them could lead to drug therapy improvements as developers learn how to manipulate existing networks by treatment with drug to stimulate, attenuate, or change the parameters of the signaling network.

32.1.5 Noise in biological networks

Many molecular events in biology are stochastic. Small numbers of different species of molecules must collide in the right orientation to bind and cause reactions that drive life. Recently, biologists and physicists have been trying to answer the question of how biological networks handle noise [133], [201], [134]. In some cases, noise would have to be attenuated (signal transduction.) In other cases, noise could be exploited to achieve population heterogeneity ([16]). An attempt to model noise in small networks has been successful since perturbations due to noise can easily be calculated at each step in the reaction [133], [201], [134]. According to a recent review article by Rao, Wolf, and Arkin, noise has been incorporated into network models three ways [16]:

- Langevin: $\frac{dC(t)}{dt} = vr(c) + x(t)$ where $C(t)$ is the concentration of a molecular species, vr are the rate and stoichiometric matrix, $x(t)$ is white noise
- Fokker-Planck: $\frac{\partial p(C,t)}{\partial t} = -\nabla^* [cr(C)p(C,t)] + \frac{1}{2}\sum_{i,j} \frac{\partial^2 \sigma_{ij} p(C,t)}{\partial C_i \partial C_j}$ where $p(C,t)$ is the probability density function and σ_{ij} is the covariance matrix of the noise
- Markov chains (Master equation) which describes the probability of a particular number of molecules being in different states as a function of time.

Unfortunately, all three representations of biochemical networks with noise are difficult or impossible to compute, or are too complicated to practically enumerate for each species for complex networks [16]. To fully understand the behavior of biochemical networks, we need to develop a practical and computationally feasible measure of multivariate noise.

To demonstrate how severely noise can affect signaling in a genetic network, consider the recent design and construction of a genetic toggle switch [69]. Gardner et al. designed and built a genetic toggle switch using non-linear dynamics to analyze the design and characterize the network's behavior with changing inputs and parameters. The toggle switch consists of two promoter and repressor pairs that mutually inhibit each other (where the repressors repress the promoter of the other pair) in *E. coli* (Figure 8.) Two inducers impair the ability of the repressors to bind to the promoter it represses. Therefore, the small genetic network is designed to behave in a switch-like manner with 2 inputs (the two inducers) and Green Florescence Protein (reporter) concentration as the output. The network is named a toggle switch because with addition of an inducer, the network will "switch" states between low and high or vice versa. After using non-linear dynamics to determine the regions of bistability (where the network behaves as a switch) and robust switching, the authors built the network (in the bistable region) on two different plasmids. The toggle model is based on a simple model of gene expression to describe the mutual inhibition of the double repressor-promoter pairs (first terms) and dilution and degradation (second terms.) Since the system is analyzed using nonlinear dynamics, the equations are written as differential equations on

the concentrations of the two repressors:

$$\begin{aligned} \frac{du}{dt} &= \frac{a1}{1+v^b} - u \\ \frac{dv}{dt} &= \frac{a2}{1+u^g} - v \end{aligned}$$

where u is the concentration of repressor 1, v is the concentration of repressor 2, $a1$ is the effective rate of synthesis of repressor 1, $a2$ is the effective rate of synthesis of repressor 2, b is the cooperativity (from multimerization of the repressors to themselves and operators) of repression of promoter 2, and g is the cooperativity of repression of promoter 1. As mentioned before, the model for the network is relatively simple and assumes that there is no noise in the system and one parameter $(a1, a2)$ can represent the sequential actions involved in transcription, translation, and regulation.

One of the assumptions made in the mathematical model of the toggle switch is that there is no noise in the system. However, we know that genetic systems, like all other biological systems can be very noisy due to inherent stochasticity. Therefore, biological networks must either exploit or attenuate noisy signals. While Gardner et al. state that the toggle provides robust switching, no data was shown. In order to investigate how robust the toggle switch is to noise, a noise term $(c * \text{randn}(t))$ was added to the differential equations of u and v:

$$\begin{aligned} \frac{du}{dt} &= \frac{a1}{1+v^b} - u + c * \text{randn}(t) \\ \frac{dv}{dt} &= \frac{a2}{1+u^g} - v + c * \text{randn}(t) \end{aligned}$$

where c is a scaling factor of the amount of noise in the system and $\text{randn}(t)$ represents a randomly generated value at each time point chosen by the normal distribution with a mean of 0 and a standard deviation of 1. Figure 9 shows v with respect to time under conditions to the left (high state) and right (low state) of the separatix for 3 different magnitudes of initial u and v with 5 different c values (marked in Figure 9.) The first row shows the system behavior with no noise. We see that with no noise, the system behaves as expected where after a switching time has passed, v is high when the system is under high state conditions and v is low when the system is under low state conditions. As the magnitude of the initial u and v increases, we see that the switching time decreases. If the system is robust to noise, all other graphs should roughly match the first in the column (with no noise.) All of the cases where this was not true are labeled "WRONG." We see different responses to noise. In many cases, the system ended in the wrong state (plots of u were generated as well, but as the information is consistent with Figure 9, they are not shown.) In other cases, the system was monostable (last row) or ended in the same state (middle graph.) We can also see that the larger the magnitude of initial u and v, the more robust the system is to noise. With c ranging from 10^{-12} to 10^{-6}, we see that the system is not robust to noise and behaves badly. This raises interesting questions as to whether a model based on Langevin equations should be used to evaluate the dynamics and behavior of the system instead. We may also ask how do naturally occurring genetic toggle switches attenuate the noisy signal? Could the noise be controlled through additional feedback or cascading?

32.1.6 Random matrix theory in biology

As biology becomes more quantitative and systems oriented, biologists are adding new methodologies to their toolboxes. Use of nonlinear dynamics in analyzing biochemical networks is important to understand how the topology of the network influences its function and how it might behavior

as a modular network in the crosstalk pathways of neighboring networks. While at this time the role of random matrices in biological research can only be speculated, the discussion above points directly to two categories of potential application. First, there is a potential to use a form of random matrices to ease the computation of models incorporating stochasticity. At very least, it might be plausible to use random matrices to estimate noise and evaluate the stability of the network under noisy conditions. A second and perhaps more direct application of the work completed in random matrix theory is to compare the distribution of eigenvalues of a random matrix to the distribution of eigenvalues of the Jacobian matrix (A) of a biological network. More specifically, one can compare how many real eigenvalues are expected of a randomly distributed matrix to A. This would be interesting to ask biologically because imaginary eigenvalues would be representative of oscillations and cycles in the biological network (such as cell cycle regulators or circadian rhythms), which are expected to be tightly controlled by nature. The distribution of eigenvalues of A might also be interesting when compared to what is expected of a random matrix (Wigner's semi-circle). This type of comparison might show how the network's topology evolved from a random state to its currently, highly controlled and fine-tuned state. Finally, as biologists reach out into physics, computer science, and engineering (especially signal processing) for tools, any application of random matrices in these areas is fair game for use in a biological application.

32.2 DNA gene expression levels

[This section was originally written by Kevin Chu]

32.2.1 Introduction

Advances in DNA microarray and gene chip technologies have made it possible to simultaneously measure the gene expression level for thousands of genes. Because microarrays and gene chips produce huge amounts of data compared with traditional experimental genetic methods, new analysis techniques need to be developed to extract meaningful biological information.

Early techniques for analyzing this large amount of genetic data focused on clustering genes as a means to investigate multiple gene effects or gain insight into the functionality of poorly characterized genes [62, 56]. These techniques were primarily based on statistical clustering algorithms and often used correlations in gene expression (or some variation) across multiple cell samples as the metric for gene similarity.

More recently, there has been an effort to examine gene expression data from a global perspective [37, 3, 4, 107]. Rather than seeking a data reduction by reducing the number of genes to focus on, data reduction is achieved by identifying dominant global genetic expression states and using these to group the experimental samples. Principal component analysis (PCA) is the main tool that has been used to carry out this data reduction. Unfortunately, the criteria used to distinguish between biologically significant groupings and experimental noise do not seem to have a solid statistical foundation.

One particularly popular application of PCA bas been to analyze time series microarray data. This paper reviews the current PCA-based methods for analyzing time series data, discusses theoretical and experimental problem areas, and proposes statistical tests that may potentially be able to address some of the statistical significance issues.

32.2.2 Source of experimental data

DNA microarray technology is based on the the unique matching between complementary DNA (cDNA) and messenger RNA (mRNA) strands. First, the DNA microarray is constructed by placing known cDNA strands in individual wells on a slide. Next, the cell sample to be studied is prepared by tagging the mRNA within the cell with a fluorescent marker (typically red). Concurrently, the mRNA of reference cells are tagged with a different fluorescent marker (typically green). The mRNA from the sample and the reference cells are are then mixed and allowed to simultaneously hybridize with the cDNA fragments on the slide. After some experimental work up, the fluorescence levels of the sample and reference tags are measured for each well on the slide. Finally, the gene expression level for each gene is computed as

$$\log(I_{\text{sample}}/I_{\text{ref}}),$$

where I is the fluorescence level.

For analysis, the gene expression data is conventionally organized in an $n \times m$ matrix where n is the number of genes whose expression is measured and m is the number of experimental samples.

32.2.3 PCA-Based analysis of DNA microarray time series data

Before examining the application of principal component analysis to DNA microarray data, it is useful to review conventional PCA and to consider what kinds of results it produces for simple time series data.

Conventional PCA

The standard use of principal component analysis in data analysis attempts to find a low dimensional subspace of the space of centered experimental variables (*i.e.* mean removed) that accounts for a majority of the variance observed in the data. Another way to think of this is that the procedure tries to find "best fit" hyper-plane of low dimensionality for the data. It is important to note that this analysis takes place in *dependent* variable-space. The associated fit that occurs in "experiment"-space does not have have the same meaning as the fit in variable-space [293]. Intuitively, the analysis is not symmetric because dependent and independent variables are not interchangeable.

However, useful information can be extracted from the fit in experiment-space. The components of the k-th singular vector in experiment-space are the relative magnitudes of measurements of the k-th principal component across the different experiments. In other words, if the *created* dependent variable associated with the k-th principal component had been measured in each experiment, the relative magnitudes in the experiments would match the relative magnitudes of the components of the k-th singular vector in experiment-space. In traditional PCA, this observation is not very interesting because the goal is to find a descriptive relationship between the dependent variables. Furthermore, since there are usually very many more data points than variables, the fit in experiment-space is probably significantly affected by any noise in the data.

Application of PCA to time series data

For time series data, the data fit generated in experiment-space takes on greater meaning because the order of the components in the k-th experiment-singular vector is significant. In this context, the

components of the k-th experiment-singular vector can be interpreted as the time series that would arise by making measurements on the k-th principal component. That is, the k-th experiment-singular vector gives the time evolution of the k-th principal component. Furthermore, the time series for different principal components are orthogonal. The orthogonality can be useful in defining normal "modes" for the dynamics of the data. For the remainder of this paper, the k-th experiment-singular vector will be referred to as the k-th *principal time series* to emphasize its dynamical interpretation.

Sinusoidal data Because current applications of PCA to microarray time series data often lead to a decomposition of the dynamics into sinusoidal modes, it is useful to consider what kinds of results PCA gives for data made up of only a few frequencies.

Single frequency data

Suppose that there are n variables which are all varying at the same frequency, f, but at arbitrarily different amplitudes and phase shifts: $A_i(t) = \alpha_i \sin(2\pi f t + \phi_i)$. Let A_{it} represent the value of the i-th variable sampled at the t-th time point. Organizing the data into a matrix, we have

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_n \end{bmatrix}$$

where A_i is the time series for the i-th variable. Then, PCA gives as its first two principal time series, a sine-cosine pair with frequency f and same phase shift (Figure **??**). The variance in the remaining principal components is at the level of machine error, so they can all be disregarded. This result makes sense because the data vector $\mathbf{x}(t) = (A_1(t), \ldots, A_n(t))$ traces out an ellipse in $\mathbf{R}^n$. Thus, the data points can be collapsed onto a two dimensional hyper-plane,

$$\mathbf{x}(t) = \sigma_1 U_1 V_1' + \sigma_2 U_2 V_2' \tag{32.1}$$

where U_i and V_i are the i-th singular vectors of the data matrix, A. Because the ellipse is traced out in time, the principal time series V_1 and V_2 must be a sine-cosine pair of with the same frequency and phase shift. Furthermore, the entire ellipse is traced out in one period the n variables, so V_1 and V_2 must have frequency f.

Multiple frequency data

For data which is a linear combination of multiple sines waves, PCA produces results are not as clean as in the single frequency case. In general, the principal time series are still linear combinations of sine and cosine waves (Figure **??**). However, if the amplitudes of the different frequency sine waves are sufficiently different, the principal time series come in sine-cosine pairs ordered by the magnitude of the Fourier coefficient in the continuous signals (Figure **??**). Intuitively, this result makes sense because when the true signal is a linear combination of multiple sine waves where different frequencies have widely differing magnitudes, the data vector $\mathbf{x}(t) = (A_1(t), \ldots, A_n(t))$ roughly traces out a perturbed ellipse in $\mathbf{R}^n$.

Effects of data quality

As with the analysis of any time series data, the frequency and duration of data sampling places severe limitations on the amount of information that can be extracted from the data. Since extracting principal time series is related to spectral analysis, it is not unreasonable to expect

the limitations that arise in spectral analysis to arise in PCA. Two important data properties are sampling frequency and sampling window [354]. As in Fourier analysis, the sampling frequency limits the range of frequencies that can be observed and the sampling window limits the achievable frequency resolution. Furthermore, a small sampling window could make it difficult to distinguish between very slow varying signals and a "monotonic" signal (such as those that were observed by Holter *et al.* [37]). For current microarray data, the sampling frequency is so low that it would be difficult to detect any dynamics that occur faster than a few times the frequency of the cell cycle. In addition, the data extends only over a few cell cycles, so it would be difficult to resolve closely spaced frequencies.

Analysis of DNA microarray time series data

Many recent analyses of DNA microarray time series data based on a PCA approach seem to be trying to extract principal time series information from the data. Part of the reason for this may be nature of microarray data – there are far fewer data points than variables. This situation is the reverse of the data which is commonly analyzed using PCA. Because there is are so few data points in microarray data, analysis in the gene expression-space is inherently noisy. The subspace determined by *all* of the data points is already of very low dimension compared to the dimensionality of the entire variable-space. As a result, it seems that researchers may have looked past the asymmetry between the variable and experiment spaces in applying PCA.

Currently, there is no consensus in how PCA should be applied to the data or how the results should be interpreted. Two main points of contention are:

1. the type of "normalization" that should be done on the data,
2. the criterion for discarding singular vectors, and

Data normalization

The goal of data normalization seems to be the removal of the steady state gene expression from the data set. The competing methods are:

1. for each gene, compute the average expression of the gene across experiments and subtract this value from all the expression values for that gene [37], and
2. after computing the singular value decomposition (SVD) for the data set, discard the principal component associated with the largest singular value [3, 4].

Some sources also suggest that the average gene expression for each experiment be removed so that all columns have zero mean [37]. However, such a procedure seems inconsistent with removing the mean for each gene (*i.e.* rows having zero mean) and does not have a clear "biological" meaning. From a traditional, PCA perspective, the procedure of removing the mean on a gene by gene basis is the one that makes most sense. While the principal component with the largest singular gives an approximation of the mean gene expression for each gene, there does not seem to any gain to using the SVD to compute the mean when direct computation is more accurate. Furthermore, procedure (2) has the "unphysical" side effect of forcing all variations around the mean genetic state to be orthogonal to the mean.

Criterion for discarding principal components

Once PCA has been applied to the gene expression data, principal components and principal time series associated with "small" singular values are discarded. However, there is no consistent

way that this is done; the cut-off value used to determine significance is quite arbitrary [4]. Raychaudhuri *et al.* chose to use a $(70/n)\%$ rule, discarding any component that accounted for less than $(70/n)\%$ of the total variance [4]. Holter *et al.* take an alternative approach of rejecting all principal components with singular values that are not "significantly" larger than the singular values of a appropriately chosen random matrix [37]. In all of these approaches, it is rare that any principal components other than the top two are retained. However, in some cases it is ambiguous whether more principal components should be kept [37, 3].

While the current approaches may seem to lack adequate rigor, they seem to be in the right spirit. The remainder of this paper will discuss some statistical tests that may be able help determine which results are significant.

32.2.4 Statistical significance tests

The singular values of the data matrix, A, are critical in PCA analysis of microarray data, so it is natural to consider statistics based on the Wishart distribution. However, because microarray data is such that there are more variables than experiments, organization of the data matrix so that it is "tall and thin" requires that rows represent genes and columns represent time points. So, normalizing the data so that each gene has zero mean expression does not produce columns that have zero mean. Thus, any statistical tests must be based on the noncentral Wishart distribution. For convenience of notation in the following discussion, we adopt Muirhead's notation $\text{etr}(A)$ to denote $\exp(\text{tr}(A))$ [326].

The noncentral Wishart distribution

Definition 32.1. *Let Z be a $n \times m$ matrix distributed as $N(M, I_n \otimes \Sigma)$ with M a $n \times m$ and Σ a $m \times m$ positive definite, symmetric matrix. Then the $m \times m$ matrix, $A = Z'Z$, has noncentral Wishart distribution with n degrees of freedom, covariance matrix Σ, and matrix of noncentrality parameters $\Omega = \Sigma^{-1}M'M$. The distribution of A is denoted by $W_m(n, \Sigma, \Omega)$. If $n \geq m$, then the density function of A is*

$$\frac{1}{2^{mn/2}\Gamma_m(\frac{1}{2}n)(\det \Sigma)^{n/2}} \cdot \text{etr}(-\tfrac{1}{2}\Sigma^{-1}A)(\det A)^{(n-m-1)/2}\text{etr}(-\tfrac{1}{2}\Omega){}_0F_1(\tfrac{1}{2}n; \tfrac{1}{4}\Omega\Sigma^{-1}A). \tag{32.2}$$

In the case where $M = \mathbf{1}\mu'$ where $\mu \in R^m$ is the expected value of each row, Ω takes on the special form $\Omega = n\Sigma^{-1}\mu\mu'$.

The joint density for eigenvalues of A may be obtained by using the following

Theorem 32.1. *Let a $m \times m$ positive definite matrix have density $f(A)$. Then the joint density of the eigenvalues, $l_1, l_2, \ldots, l_m$ of A is*

$$\frac{\pi^{m^2/2}}{\Gamma_m(\frac{1}{2}m)} \prod_{i<j}^{m} (l_i - l_j) \int_{O(m)} f(HLH')(dH) \tag{32.3}$$

where $l_1 > l_2 > \ldots > l_m$ and $L = \text{diag}(l_1, l_2, \ldots, l_m)$ [326].

Thus, for a matrix A from a noncentral Wishart distribution, we find that

$$\frac{\pi^{m^2/2}}{2^{mn/2}\Gamma_m(\frac{1}{2}m)\Gamma_m(\frac{1}{2}n)(\det\Sigma)^{n/2}}\prod_{i=1}^{m} l_i^{(n-m-1)/2}\prod_{i<j}^{m}(l_i-l_j)\cdot$$
$$\operatorname{etr}(-\tfrac{1}{2}\Omega)\int_{O(m)}\operatorname{etr}\left(-\tfrac{1}{2}\Sigma^{-1}HLH'\right){}_0F_1(\tfrac{1}{2}n;\tfrac{1}{4}\Omega\Sigma^{-1}HLH')(dH). \tag{32.4}$$

When analyzing a sample covariance matrix, it is nS that has distribution $W_m(n,\Sigma,\Omega)$. If $l_1, l_2, \ldots, l_m$ are the eigenvalues of S then the joint density of the eigenvalues, is given by equation (4), with the change of variables $l_i \to nl_i$:

$$\left(\frac{n}{2}\right)^{nm/2}\frac{\pi^{m^2/2}}{\Gamma_m(\frac{1}{2}m)\Gamma_m(\frac{1}{2}n)(\det\Sigma)^{n/2}}\prod_{i=1}^{m} l_i^{(n-m-1)/2}\prod_{i<j}^{m}(l_i-l_j)\cdot$$
$$\operatorname{etr}(-\tfrac{1}{2}\Omega)\int_{O(m)}\operatorname{etr}\left(-\tfrac{1}{2}n\Sigma^{-1}HLH'\right){}_0F_1(\tfrac{1}{2}n;\tfrac{1}{4}n\Omega\Sigma^{-1}HLH')(dH). \tag{32.5}$$

As for the central Wishart distribution, it should be possible to obtain an asymptotic approximation to this density in the limit of large n using the multivariate generalization of Laplace's method described in Muirhead [326]. Unfortunately, the hypergeometric function ${}_0F_1$ does not have as simple a form as ${}_0F_0$. Furthermore, finding an asymptotic approximation for ${}_0F_1(\frac{1}{2}n;\frac{1}{4}n\Omega\Sigma^{-1}A)$ based on its integral representation is complicated by the fact that the orthogonal group that the integral is taken over has dimension equal to the limiting variable, n.

Sphericity test

For conventional PCA, the *Sphericity test* is commonly applied to determine if only the first k eigenvalues of the true covariance matrix for the variables are different. If this is found to be the case and if the estimate of the value of the last $m-k$ eigenvalues is small compared to the k-th eigenvalue, the last $m-k$ principal components are discarded.

The test proceeds in stages. At stage k, the hypothesis, H_k, that the last $m-k$ are equal is tested. The value of k is incremented until H_k is not rejected or $k=m$. If H_k passes at some stage with $k<m$, the conclusion is that the first k eigenvalues differ but the last $m-k$ are equal. If testing terminates with $k=m$, the conclusion is that all m eigenvalues of the true covariance matrix are different, so no data reduction is possible.

Suppose that A is a $N\times m$ data matrix with rows representing data samples and columns representing variables:

$$A=\begin{bmatrix} A_1 \\ \vdots \\ A_N \end{bmatrix}.$$

Let

$$S=\frac{1}{n}\sum_{i=1}^{N}(A_i-\overline{A})'(A_i-\overline{A})\quad (n=N-1)$$

be the sample covariance matrix for A, and let $l_1, l_2, \ldots, l_m$ be the eigenvalues of S. Then the likelihood ratio statistic for testing the k-th null hypothesis,

$$H_k:\lambda_{k+1}=\cdots=\lambda_m\quad(=\lambda,\text{unknown})$$

is

$$V_k = \frac{\prod_{i=k+1}^{m} l_i}{(\frac{1}{m-k}\sum_{i=k+1}^{m} l_i)^{m-k}}$$
$$= \left[\frac{(\prod_{i=k+1}^{m} l_i)^{\frac{1}{m-k}}}{\frac{1}{m-k}\sum_{i=k+1}^{m} l_i}\right]^{m-k}. \quad (32.6)$$

Note that this is the ratio of the geometric mean and arithmetic mean of the eigenvalues. Alternatively, this statistic could have been written in terms of the singular values of A:

$$V_k = \left[\frac{(\prod_{i=k+1}^{m} \sigma_i^2)^{\frac{1}{m-k}}}{\frac{1}{m-k}\sum_{i=k+1}^{m} \sigma_i^2}\right]^{m-k} \quad (32.7)$$

where $\sigma_1, \ldots, \sigma_m$ are the singular values of A. It can be shown that as $n \to \infty$, the asymptotic distribution of $-n \log V_k$ is $\chi^2_{(m-k+2)(m-k-1)/2}$ [326].

For DNA microarray data analysis, the sphericity test must be modified because the columns of the data matrix no longer have zero mean and removing column means would destroy the property that rows have zero mean. Thus, we must construct a sphericity test based on the noncentral Wishart distribution.

In the sphericity test, the hypothesis

$$H_k : \lambda_{k+1} = \cdots = \lambda_m \quad (= \lambda, \text{unknown})$$

is being tested against the alternative that there is no relation between $\lambda_{k+1}, \cdots, \lambda_m$. Thus, the likelihood ratio statistic is

$$\Lambda = \frac{\sup L(M, \hat{\Sigma})}{\sup L(M, \Sigma)} \quad (32.8)$$

where $\hat{\Sigma}$ is allowed to vary over all positive definite, symmetric matrices that have $\lambda_{k+1} = \cdots = \lambda_m = \lambda$ (λ unspecified) and Σ is allowed to vary over all positive definite, symmetric matrices. Since the likelihood function is just the probability density function under a given set of assumptions on the parameters of the distribution, the likelihood function, $L(M, \Sigma)$ for a matrix with a noncentral Wishart distribution is given by the expression in equation (2).

Theoretically, it should be possible to compute the probability distribution for the likelihood ratio statistic, Λ. With this distribution (or an asymptotic approximation), it should be straightforward to compute a rejection cut-off, Λ_0, for Λ with significance level

$$\alpha = \Pr(\Lambda \leq \Lambda_0 \mid H_k). \quad (32.9)$$

Crude singular value test

To determine whether the k largest singular values are significant, the k-th singular value could be used as a crude statistic. For a data matrix A that is distributed $W_m(n, \Sigma, \Omega)$, it should be possible to compute an asymptotic distribution for the largest singular value, σ_1. To test the hypothesis that the k largest singular values are significant, the k-th largest singular value could be compared to a critical value, c, which is determined by the criterion:

$$\alpha = \Pr(\sigma_1 > c \mid W_m(n, \Sigma, \Omega)\ ) \quad (32.10)$$

where α is the "significance level" of the test. If $\sigma_k > c$, then $\sigma_1, \ldots, \sigma_k$ are taken to be significant because the probability that $\sigma_k > c$ is less than the probability $\sigma_1 > c$.

Open issues

In both the sphericity test and crude singular value test, it is likely that there will be some unspecified parameters in the probability distribution of the statistics. In particular, the mean matrix, M, and the covariance matrix, Σ, need to be specified. For the case where $M = \mathbf{1}\mu$, μ will need to be specified instead. For microarray data, a naive estimate for these parameters might be the column means and covariance matrix. However, further investigation would be needed to determine the appropriateness of these estimates.

32.2.5 Conclusions and future directions

Principal component analysis holds promise as a method for analyzing DNA microarray data. For producing principal time series, it appears to be useful in extracting orthogonal temporal "modes" of genetic variation. However, further testing on higher quality data sets will need to be done before a solid conclusion can be reached. In addition, further research (possibly just journal research) of the noncentral Wishart distribution is necessary in order develop the statistical tools necessary to analyze the significance of PCA results.

While current data only allows genetic dynamics at the frequency of the cell cycle to be studied, it is likely that much of the interesting dynamics occurs at higher frequency and much lower amplitudes. From the resilience of living organisms, it makes sense that there should be only a few very dominant modes in genetic dynamics. However, nontrivial cellular response to environmental stimuli are probably hidden in small perturbations (of both higher and lower frequency) to the stable genetic dynamics of the cell cycle.

For both the development of data analysis techniques and deeper understanding of gene dynamics, it will be necessary to obtain higher quality data than is currently available. In particular, data taken at higher frequency and over more cell cycles will be needed to improve the frequency resolution and range that can be studied. However, experimental limitations may make collecting this data difficult.

Chapter 33

Longest Increasing Subsequence

[This section was originally written by Anand Sarwate]

33.1 Introduction

In this paper we will investigate the connection between random matrices and finding the longest increasing subsequence of a permutation. We will introduce a model for the problem using a simple card game. Then we will talk about Young tableaux and their relation to the symmetric group. Representation theory and power-sum symmetric functions serve as the bridge between this combinatorial construction and random matrices. The presentation in this paper is largely modeled on that of Aldous and Diaconis [2], but is shorter and designed to be read by a wider audience. A reader with two semesters of abstract algebra and some probability should be able to follow the ideas presented here with little difficulty.

The problem in which we are interested in is as follows. Let π be a random permutation of the integers $1, 2, \ldots, n$. $\pi(i)$ is the i-th element of the permutation. Then an *increasing subsequence* $(i_1, i_2, \ldots, i_k)$ of π is a subsequence satisfying:

$$\begin{aligned} i_1 < i_2 < &\quad \ldots \quad < i_k \\ \pi(i_1) < \pi(i_2) < &\quad \ldots \quad < \pi(i_k) \end{aligned}$$

We define $l(\pi)$ as the length of the *longest increasing subsequence*. We denote by L_n the integer valued random variable which takes on the value of $l(\pi)$. The purpose of this paper is to investigate the distribution of L_n and to express this distribution in terms of random matrices.

In the second section we discuss *patience sorting*, a simple card game model for computing $l(\pi)$ and provide some Monte Carlo simulations to show the empirical distribution of L_n. The third section deals with Young tableaux, a combinatorial construction with applications in representation theory and geometry. We describe these structures and prove the Schensted correspondence, which makes explicit the link between the symmetric group and the set of Young tableaux. The Schensted correspondence can also be used to compute the distribution of L_n. The fourth section provides a quick summary of some facts from representation theory and expresses the distribution of L_n in terms of characters of the irreducible representations of the symmetric group S_n. The fifth section discusses power-sum symmetric functions and their role in linking the symmetric and unitary groups. This allows us to express the distribution of L_n in terms of the matrix integral of the trace

of a power of a unitary matrix. The last section covers some extensions of this work to other groups of matrices and other types of increasing subsequence problems.

33.2 Raking Leaves and Longest Increasing Subsequences

Suppose we're given a permutation $p = (p_1, \ldots, p_n)$ of the integers 1 through n. An *increasing subsequence* is a sequence of elements of p (not necessarily consecutive), $p_{i_1}, p_{i_2}, \ldots, p_{i_k}$, with $i_1 < i_2 < \cdots < i_k$ and $p_{i_1} < p_{i_2} < \cdots < p_{i_k}$. Our problem is to find an increasing subsequence that is as long as possible. There may be more than one; for example, $p = (3, 1, 4, 5, 9, 2, 6, 8, 7)$ has four longest increasing subsequences, including $(1, 4, 5, 6, 7)$ and $(3, 4, 5, 6, 8)$.

We can look at this as a problem on a directed acyclic graph, or dag. The vertices of the dag are the integers 1 through n, and there is an edge directed from i to j if $i < j$ and i occurs to the left of j in the permutation. An increasing subsequence is a directed path. The length of the longest increasing subsequence is the *height* of the dag.

Define a *leaf* as a vertex with no edges directed into it. The leaves in our example are 3 and 1. Appropriately for the season, we can find the height of the dag by an algorithm called *leaf raking*:

- repeat
- identify all the leaves
- delete all the leaves
- until no vertices remain

The number of iterations this loop needs to delete all the dag's vertices is its height. Consider the example. The first iteration deletes the leaves 3 and 1. The leaves of the remaining dag are 4 and 2. (Notice that leaves are the same as left-to-right minima.) After deleting 4 and 2, the only leaf is 5, and so on. Call the vertices deleted in a single iteration a *level*. We can write down a tableau showing the vertices by level, in the order they are deleted:

$$(31 \| 42 \| 5 \| 96 \| 87).$$

This tells us that longest increasing subsequences have length five, but how do we find one? Easy lemma: every vertex in the last level of the tableau (8 and 7 in the example) is the final vertex of some longest path. (Not every vertex in the first level need be the start of a longest path. Exercise: give an example.) Thus we can construct a longest path by starting at any vertex of the last level and working backwards: start with 8, say; the lemma promises us a vertex on the previous level with an edge to 8 (it's 6); the lemma promises an edge from 5 to 6; and an edge from either 4 or 2 to 5 (it's 4); and from either 3 or 1 to 4 (either will do, say 3). Reversing the order in which we chose the vertices gives the longest increasing subsequence $(3, 4, 5, 6, 8)$.

Leaf raking has an application to parallel solution of triangular linear systems. Suppose we wish to solve $Lx = b$ for x, given an n-by-n lower triangular matrix L and a right-hand side vector b. The directed graph of L^T has n vertices, with an edge directed from i to j if $i < j$ and $l_{ji} \neq 0$. (Note the transpose.) If L is nonsingular then its diagonal elements are nonzero, but we don't include loop edges for them. The graph is a dag because L is triangular. A leaf corresponds to a row of L (or column of L^T) whose only nonzero entry is on the diagonal. Thus all the entries of x that correspond to leaves can be computed independently and simultaneously. We can then rake up those leaves and proceed to the the next level of the dag. On an ideal parallel machine with arbitrarily many processors, we solve the whole system in a step per level of leaf raking.

33.3 Patience sorting: a model

Suppose we have a deck of cards numbered $1, 2, \ldots, n$ which we shuffle into a random order[1]. We turn up cards one at a time and place them into piles according to the following rule: *a lower number may be placed on top of a higher number or be placed into its own pile.* The goal is to end with as few piles as possible.

While this may not seem to be the most interesting of games at first glance, a simple strategy for playing yields an effective way of computing $l(\pi)$. The greedy strategy involves placing the current card on the leftmost pile possible. This game is best illustrated with an example. Say we have a deck of 10 cards in the following order:

8 3 7 9 2 5 4 1 10 6

The greedy strategy will play the game in the following manner:

										2				2					
		3		3			3			3				3	5				
8		8		8	7		8	7	9	8	7	9		8	7	9			
					1			1					1						
2	4			2	4			2	4				2	4					
3	5			3	5			3	5				3	5	6				
8	7	9		8	7	9		8	7	9	10		8	7	9	10			

It turns out that patience sorting is an easy way for us to not only calculate $l(\pi)$, but also to find an increasing subsequence in π of length $l(\pi)$.

Theorem 33.1. *Let π be the ordering of a deck of cards numbered $\{1, 2, \ldots, n\}$. Using the greedy strategy to play patience sorting using π will result in exactly $l(\pi)$ piles.*

Proof. We will first prove that the number of piles is at least $l(\pi)$ and then provide a construction to show that it is at most $l(\pi)$.

Let $i_1 < i_2 < \ldots < i_k$ be an increasing subsequence in π. Then $\pi(i_j)$ must always lie in a pile to the right of $\pi(i_{j+1})$ since we are only allowed to place cards on top of cards of smaller value. Therefore any valid strategy, and particularly the greedy strategy, will result in at least $l(\pi)$ piles.

We can use patience sorting to find an instance of an increasing subsequence, thereby showing that we have at most $l(\pi)$ piles. Every time we place a card c in a pile i that is not the first pile, draw an arrow from c to the top card d of the preceding pile $i - 1$. We know $d < c$ because in our greedy strategy, c would be placed on top of d if $d > c$. Note that each arrow goes from a later card to an earlier card. This is illustrated in Figure **??**.

If we have k piles, call the the top card in the rightmost pile a_k. Then follow the arrow from a_k to a_{k-1} and so on. In this way we construct an increasing subsequence in π. Therefore we have at most $l(\pi)$ piles. □

The reason the game is called patience *sorting* is because at the end, we can sort the cards into order. Card 1 will be at the top of some pile. Removing it, we are left with a patience-sorted deck

[1] We recommend the reader to make such a deck themselves to better follow along with the examples used in this section.

of $\{2, 3, \ldots, n\}$, so card 2 must now be at the top of some pile. Proceeding as before we can sort the deck. As it turns out, however, this is also a convenient and easy way to compute $l(\pi)$.

In Appendix 33.8 we provide a MATLAB script to compute the distribution of L_n using this method. A histograms for $n = 10$ and $n = 20$ using 10^7 samples are shown in Figure **??**. In the case of patience sorting, winning is not well defined, and instead we can arbitrarily choose a number of piles as a threshold for winning or losing. Readers more interested in patience sorting and its history should read the survey by Aldous and Diaconis [2], which also covers much of the other material in this paper.

33.4 Young Tableaux

We now turn to another way of looking at permutations using diagrams called *Young tableaux*. These are special structures which are used in invariant theory and group representations of the symmetric group S_n. They are also important in combinatorics and algebraic geometry, and in the theory of symmetric functions. What is most important for our problem is an important construction called the Schensted correspondence, which relates multiset permutations to ordered pairs of Young tableaux.

Let $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ be a set of integers such that $\sum \lambda_i = n$ and $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. Then we say λ is a *partition of* n and write $\lambda \vdash n$. We denote the number of entries in λ by $|\lambda|$, which in this case is equal to k. In the special case where $\lambda = (1, 1, \ldots, 1)$ we write $\lambda = 1^n$.

A *Ferrers diagram* with shape λ is a set of cells as shown in Figure **??**, where row i has λ_i cells. A *standard Young tableau* is a Ferrers diagram with the numbers $1, 2, \ldots, n$ in the cells such each number is used once, and the entries increase along each row and down each column.

The *hook length* h_c of a cell c is the number of cells to the right of c in its row plus the number of cells below c in its column plus the cell c itself. Thus in the tableau in Figure **??** the hook length of the cell containing 5 is $h_c = 4$, and the hook length of the cell containing 8 is $h_c = 2$.

An interesting question is this: how many standard Young tableaux are there of shape λ? The answer is surprisingly simple, and given by the following theorem.

Proposition 33.2. *(Hook Formula) Let d_λ be the number of standard Young tableaux of shape λ. Then:*

$$d_\lambda = \frac{n!}{\prod_c h_c} \tag{33.1}$$

If we look at the tableau in Figure **??** we can calculate the number of tableau of that shape:

$$d_{(4,3,2,2)} = \frac{11!}{7 \cdot 6 \cdot 3 \cdot 1 \cdot 5 \cdot 4 \cdot 1 \cdot 3 \cdot 2 \cdot 2 \cdot 1} = 1320$$

This is far too many to explicitly verify, so we can look at an easier example. Consider $\lambda = (3, 2) \vdash 5$. The hook length formula tells us we can only construct $120/(4 \cdot 3 \cdot 1 \cdot 2 \cdot 1) = 5$ standard Young tableaux. These are shown in Figure **??**. It is left as an exercise to the reader to prove there exist no more standard Young tableaux of that shape.

Unfortunately, no simple combinatorial proof of the hook formula exists. A number of outlines of existing proofs are given by Sagan ([387], p. 266).

Young tableaux are related to the permutation group by a construction called the Schensted correspondence, or perhaps more appropriately, the Robinson-Schensted-Knuth correspondence. For

a more in-depth discussion of the nomenclature, see Fulton ([156], p38). The Schensted correspondence is a bijection between the set of permutations and ordered pairs of Young tableaux. Interested readers should consult Sagan [387] or Stanton and White [430] for more general treatments of the correspondence.

Theorem 33.3. (Schensted correspondence).

There exists a bijection between permutations $\pi \in S_n$ and all pairs of standard Young tableaux (P, Q) of the same shape $\lambda \vdash n$.

Proof. We will construct the pair (P, Q) from a permutation π to show that any permutation can be mapped onto an ordered pair of Young tableaux. We will then provide a means of recovering π from the pair (P, Q) to prove every ordered pair corresponds to a unique permutation. In the spirit of the previous section, we will say that the m-th *card* in our permutation is $\pi(m)$.

Assume π is a permutation of the numbers $1, 2, \ldots, n$. It will help to think of π in a two-line form where the first line are the integers from 1 to n and the second line is the permutation.

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 8 & 3 & 7 & 9 & 2 & 5 & 4 & 1 & 10 & 6 \end{pmatrix} \tag{33.2}$$

We will construct the Q-tableau from the first line and the P-tableau from the second by inserting one card at a time moving from left to right. Denote by P_m the tableau created after inserting m cards, and Q_m analogously. To insert the m-th card, we use the following rules.

1. If $\pi(m)$ is larger than all cards in the current column of P_{m-1}, append $\pi(m)$ to the end of the current column.

2. If $\pi(m)$ is not larger than all cards in the current column of the P_{m-1}, replace the smallest card b such that $b > \pi(m)$ with $\pi(m)$. Now use rules 1 and 2 to insert b into the next column to the right.

3. After $\pi(m)$ has been inserted into the P_{m-1}, we obtain P_m, which has the same shape as P_{m-1} except for a single added cell. Create Q_m by adding that same cell to Q_{m-1} with the number m in that cell.

As we can see, this algorithm produces two standard Young tableaux of the same shape from the permutation π. To give an example, we construct the two tableau for the permutation in (33.2) above in Figure **??**.

Now we must describe how to recover a permutation π from a pair of standard Young tableaux. We essentially perform the reverse operation to the column insertion described in the first half of the proof. Again, we let P_m be the P-tableau with n cells, and Q_m the Q-tableau with m cells. The procedure is outlined below:

1. Remove the largest entry m in Q_m to form Q_{m-1}. Find the corresponding entry b in P_m and remove it from the tableau.

2. If b is in the first column of P, then set $\pi(m) = b$.

3. If b is not in the first column, we insert b into the column to the left. Find the largest entry c of this column such that $b > c$. Put b in the position of c and then place c according to rules 2 and 3.

It is clear that this procedure simply reverses the previous construction. Figure **??** provides an example of recovering π from a pair of Young tableaux. The reader can verify that the given permutation recovered does generate the Young tableaux.

We have exhibited an map which takes each permutation to an ordered pair of Young tableaux. The inverse takes any pair of tableaux and sends it to a permutation in S_n. □

The Schensted correspondence has a number of interesting and beautiful properties, two of which are quickly described here to give the reader the feel for how useful it is.

Proposition 33.4. *If π corresponds to (P, Q) under the Schensted correspondence, then π^{-1} corresponds to (Q, P).*

This is not too difficult to prove, and a good solution is provided in [430]. An *involution* of $\{1, 2, \ldots, n\}$ is an element π of S_n such that $\pi = \pi^{-1}$. Thus Proposition 33.4 gives us the following corollary.

Corollary 33.5. *The number of involutions of $\{1, 2, \ldots, n\}$ is $\sum_\lambda d_\lambda$.*

Schensted's original motivation in constructing this correspondence was to investigate ways of computing $l(\pi)$ for a given permutation. While this method is more tedious to perform than patience sorting, This result is more useful to us because it will allow us, via representation theory, to connect the distribution of L_n to random matrices.

Proposition 33.6. *Given a permutation $\pi \in S_n$, the number of rows in the corresponding P-tableau given by the Schensted correspondence is equal to $l(\pi)$. The length of the longest decreasing subsequence is given by the number of columns of the P-tableau.*

We can now give an explicit formula for the distribution of L_n by counting the number of Young tableaux with a certain number of rows.

$$P(L_n = l) = \frac{1}{n!} \sum_{\lambda \vdash n, |\lambda| = l} (d_\lambda)^2 \tag{33.3}$$

33.5 Representations of S_n

In this section we review some basics of representation theory and describe the link between the representations of the symmetric group S_n and standard Young tableaux. For a more basic treatment of group representations the reader is referred to the algebra textbook by Artin [20]. For more about representations of the symmetric group, the books Diaconis [109] and Sagan [388] are good overviews.

Let G be a group and V be a finite-dimensional vector space over the complex numbers $\mathbb{C}$. Then a *representation of G* is a homomorphism $\rho : G \to GL(V)$, where $GL(V)$ is the general linear group of all invertible linear transformations from V to itself. The space V is called the G-module associated with ρ because G acts on the space V. The *degree* of ρ is defined to be $\dim(V)$.

The representation $\rho : G \to GL(V)$ that sends all elements of G to the identity is called the *trivial representation* and has degree 1. If $G = S_n$, then we can let ρ map G to the set of $n \times n$ permutation matrices. This is called the *defining representation* of S_n and has degree n.

A G-module V is called *irreducible* if there is no proper subspace W of V that is invariant under the action of G. Mashke's theorem states that every G-module can be written as the direct

sum of irreducible G-modules. The number of irreducible representations is equal to the number of conjugacy classes of G.

Consider the group S_n. The *cycle-type* of a permutation is a list $(\lambda_1, \lambda_2, \ldots, \lambda_k)$ of the cycle lengths in the permutation. The conjugacy classes of S_n consist of permutations having the same cycle type. Note that a cycle-type is the same as a partition of n. We can in fact identify with each partition $\lambda \vdash n$ an irreducible representation S^λ of S_n called a *Specht module*.

The *character of a representation* ρ is a function $\chi : G \to \mathbb{C}$ defined by $\chi(g) = tr(\rho(g))$. The character is constant on the conjugacy classes of G. Let $e \in G$ be the identity element. Then every representation must map e to the identity matrix in $GL(V)$. Thus $\chi(e) = tr(\rho(e)) = \dim(V)$. In the case where $G = S_n$ we denote the characters of S^λ by χ^λ. The most important fact for us we shall state without proof. Readers who are interested should consult [156] or [388].

Lemma 33.7. *Let λ be a partition of n, S^λ be the corresponding Specht module, and e be the identity element of S_n. Then:*

$$d_\lambda = \dim(S^\lambda) \tag{33.4}$$

This is a powerful result which relates the combinatorial enumeration of standard Young tableaux to the representations of the symmetric group. It follows from a construction which turns the standard Young tableaux of shape λ into a basis for the Specht module S^λ. Combining Lemma 33.7 and equation 33.3 we obtain the following expression for the distribution of L_n.

$$P(L_n \leq l) = \frac{1}{n!} \sum_{\lambda \vdash n, |\lambda| \leq l} (\chi^\lambda(e))^2 \tag{33.5}$$

33.6 Random matrices and L_n

We are now ready to connect the results from the previous sections to what we know about random matrices. We have expressed the density of L_n in terms of the characters of the symmetric group. Frobenius showed an explicit relation via power-sum symmetric functions between the characters of the symmetric group and the characters of the unitary group. By exploiting this relation, we can express the quantity in equation (33.5) in terms of an inner product of two power-sum symmetric functions, which is also a way of calculating a matrix integral over the unitary group. In this section we will let $k = |\lambda|$ to make the notation clearer.

The power-sum symmetric functions play an important role in the following analysis, since they are the link between random matrices and random permutations. Let U be a $k \times k$ matrix with eigenvalues $\mathbf{x} = (x_1, x_2, \ldots, x_k)$. Then the power sum symmetric function P_j is given by:

$$P_j(U) = \sum_{i=1}^{n} x_i^j$$

If $\lambda \vdash n$, then we have:

$$P_\lambda = \prod_{j=1}^{k} P_{\lambda j}$$

The most useful fact about these function is their relationship to the Schur functions. In fact, Frobenius showed that the Schur functions form a basis for the power-sum symmetric polynomials:

$$P_\lambda = \sum_{\mu \vdash k} \chi^\lambda(\mu) s_\mu$$

where $\chi^\lambda(\mu)$ is the character of the Specht module S^λ of the symmetric group S_n, and the Schur functions $\{s_\mu | \mu \vdash k\}$ are the characters of the unitary group $U(k)$ of $k \times k$ matrices.

Diaconis and Shahshahani [115] give an expression for the inner product of two power-sum symmetric functions, where the expectation is taken over the unitary group with normalized Haar measure.

$$E_{U \in U(l)}[P_\lambda(U)\overline{P_\mu(U)}] = \frac{1}{n!} \sum_{\nu,\kappa \vdash n} \chi^\nu(\lambda)\chi^\kappa(\mu) E_{U \in U(l)}(s_\mu(U) s_\kappa(U)) \tag{33.6}$$

We are now ready to prove our the main result of this paper.

Theorem 33.8. *The cumulative distribution function of the random variable L_n is given by the following equation.*

$$P(L_n \leq l) = \frac{1}{n!} \int_{U(l)} |Tr(U)^n|^2 \mathrm{d}U \tag{33.7}$$

where dU *is the normalized Haar measure on the group $l \times l$ unitary matrices.*

Proof. The power-sum symmetric function $P_1(U)$ is simply the trace of U. If $\lambda = 1^n$, then $P_\lambda(U) = Tr(U)^n$. So we can see then that the integral in the right-hand side of the result is the inner product of $P_{1^n}(U)$ with itself. Thus we can set $\lambda = \mu = 1^n$ in (33.6) to obtain a a much simpler expression.

$$\int_{U(l)} |Tr(U)^n|^2 \mathrm{d}U = \sum_{\nu \vdash n, k \leq l} (\chi^\nu(1^n))^2 (s_\nu(U) s_\nu(U)$$

Because the Schur functions s_μ are orthogonal, those terms vanish, and we are left with:

$$\int_{U(l)} |Tr(U)^n|^2 \mathrm{d}U = \sum_{\nu \vdash n, k \leq l} (\chi^\nu(1^n))^2$$

But this is the same expression as in (33.5), since the partition 1^n corresponds to the identity permutation e. The result follows immediately. □

33.7 Conclusion

There are a number of questions we can ask at this point. What is the asymptotic behavior of L_n as $n \to \infty$? Are there similar interpretations for matrix integrals over the orthogonal and symplectic group? Can we obtain similar results for colored permutations and multiset permutations?

As it turns out, the asymptotic behavior of L_n has been studied by many mathematicians, and a good summary of the research from many different perspectives can be found in the paper by Aldous and Diaconis [2]. For extensions to other classical groups, the paper by Rains [369] relates results on those groups to problems in counting longest increasing subsequences with some restrictions, as well as some results on colored permutations. Odlyzko and Rains [336] have explicit results and more in-depth Monte Carlo simulation for the behavior of L_n.

33.8 Computing $l(\pi)$ with patience sorting

Below is code in MATLAB used to compute a histogram of $l(\pi)$.

```
%% patienceSort.m
%%
%% Performs greedy patience sorting on decks of length n
%% for given number of samples.  The output histogram is
%% saved in v.

n = 10;
samples = 1e7;

v = zeros(n,1);

for loop1 = 1:samples
   ord = randperm(n);
   piles = n;

   for ind1=1:length(ord)
     curr = ord(ind1);

     [val, pos] = find(piles > curr);

     if (val ~= [])
       piles(min(pos)) = curr;
     else
       piles = [piles curr];
     end
   end

   v(length(piles)) = v(length(piles)) + 1;
end
```

Code 33.1

33.9 Constructing the Schensted correspondence

33.10 Computing $l(\pi)$ with matrix integrals

Chapter 34

Numerical Analysis

34.1 Testing Numerical Software

34.2 Trefethen's Work

34.3 Tony Chan's Work

34.4 Generating Random Variables

34.4.1 Random Orthogonal Matrices

34.4.2 The Ziggurat Method

[This section was originally written by Jason Burns]

Overview of the Ziggurat Method

Marsaglia and Tsang's ziggurat method (*Journal of Statistical Software*, October 2000, pp. 1–7) is a type of rejection method. All rejection methods have the same general method of operation: we take the graph of the p.d.f. of a distribution, then choose a random point under it with uniform probability, whose x-coordinate is our desired random number. The random point is actually chosen from a set containing the area enclosed by the p.d.f.; we test each point generated and discard those that do not fall inside the p.d.f. curve (hence the name, "rejection method").

In the case of Marsaglia and Tsang's ziggurat method (which can be found at `http://www.jstatsoft.org/v05/i08/ziggurat.pdf`), the containing set consists of a sequence of disjoint horizontal strips, each of equal area. This makes it easy to choose a point in this set (choose a strip, then choose an x-coordinate uniformly from this strip), and we may reduce the excess area as much as we like by increasing the number of strips. Moreover, if we set up our strips reasonably intelligently, most of the area of each strip will be guaranteed to lie under the curve, and we need not even test such points, further increasing the efficiency of this method in typical cases.

More specifically, let $f(x)$ be a monotone decreasing p.d.f. on the interval $[0, A]$; then we pick the n x-values $x_0 \geq \cdots \geq x_{n-1} = 0$, and the corresponding y-values $f(x_0) \leq \cdots \leq f(x_{n-1})$, such that the rectangles with upper-left corner $x = 0, y = f(x_i)$ and lower-right corner $x = x_{i-1}, y = f(x_{i-1})$ all have equal area v. (We can accomplish this inductively; given x_0 through x_{i-1}, pick x_i so that

$f(x_i) - f(x_{i-1}) = v/x_{i-1}$.) Then to pick a random element of this set, we first pick a random rectangle ($i = 0..(n-1)$, with equal probability since all areas are equal), then pick a random x-coordinate within this rectangle (which is thus uniform on $[0, x_{i-1}]$), and then if we need it, a random y-coordinate (uniform on $[f(x_{i-1}), f(x_i)]$). Having generated x and y, we need only check that $y \leq f(x)$ to know that our point is under the probability curve and may be accepted.

Notice also that if $x \leq x_i$, we need not even check y (so we don't have to generate it), because all possible y-values are at most $f(x_i)$, hence (since f is decreasing) at most $f(x)$. In general, this interior rectangle takes up most of the area of the larger rectangle, so most of the time we need only one uniform random number to generate a number in a ziggurat-compatible distribution. (We also need to determine which rectangle the number is drawn from; in practice, we just steal the low-order bits of the uniform variate generated to choose a box.)

Most random numbers begin as random integers, and are only afterward converted into floating-point numbers. Thus Marsaglia makes the suggestion that the numbers x_i be stored in the form $k_i := 2^{32} x_i / x_{i-1}$ as an integer, so that we compare the random number (from 1 to 2^{32} in this case) with k_i to determine whether it falls in the interior rectangle or not. This is simply a rescaling of the box from length x_{i-1} to 2^{32}; we store the inverse $k_i := 2^{32} x_i$ for easier conversion to the corresponding x-coordinate as well. These could be computed separately for each instance, but this saves time.

Enhancements to the Ziggurat Method

One basic enhancement is to extend the method to semi-infinite intervals $[0, \infty)$, that is, to allow an infinite tail. To do this, we replace the bottom ($i = 0$) rectangle by a strip including the tail. The area of this strip is still v, but now that area is composed of two parts: one is the interior rectangle $x \leq x_0$, and the other is the infinite tail $x > x_0$. In order to apply the same methods as before, we "rectify" this strip; if it were a rectangle with the same height and area, but the infinite tail squared off, it would have width q, where $q = v/f(x_0)$. Thus we may pick a random x-value between 0 and q, bearing in mind that the values above x_0 correspond to points in the tail; if we get one of these we move to a special algorithm for generating random points in the tail. (More on this later.)

Another basic enhancement is to allow negative values as well. If we restrict our attention to distributions symmetric about $x = 0$, such as the normal distribution, this is trivial: just use a random bit for the sign (plus or minus) of the number generated.

For more general unimodal distributions, like the chi-square distribution, we must keep track of both ends of the rectangle, since we are no longer guaranteed that $f(0) \geq f(x_i)$. We will have to keep track of values x_i' for the left endpoint of each box, as well as the right endpoints x_i. This is twice the work, but otherwise straightforward.

Example: The Normal Distribution

Here is code, taken and simplified from Marsaglia's paper, to calculate normal variates using this method. The inline expression `SHR3` calculates a uniform random variate, and `RNOR` takes this number and returns a normal variate. Notice that `RNOR` may need to call the function `nfix()` if the point falls outside the interior rectangles, which in turn may need the special `ntail()` to calculate a point in the infinite tail, but both of these are expected to happen infrequently (less than three percent of the time, with 128 boxes). I've also included a short `main()` function I used for timing.

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

static unsigned long _jz, _jsr=123456789;

#define SEED(jsrseed) (_jsr ^= (jsrseed))
#define SHR3 (_jz=_jsr, _jsr^=(_jsr<<13), _jsr^=(_jsr>>17), _jsr^=(_jsr<<5),
              _jz+_jsr)
#define UNI (.5 + (signed) SHR3 * .2328306e-9)

static long _h;
static unsigned long _i, _k[128];
static float _w[128], _f[128];

#define RNOR (_h=SHR3, _i=_h&127, (abs(_h)<_k[_i])? _h*_w[_i] : nfix())

float ntail(void) {
        const float r = 3.442620f;      /* r = start of right tail */
        static float x, y;
        do {
                x = -log(UNI)*0.2904764;        /* .2904764 is 1/r */
                y = -log(UNI);
        } while (y+y < x*x);
        return r+x;
}

float nfix(void) {
        static float x;
        for(;;){
                x = _h*_w[_i];
                if (_i==0) return (_h>0)? ntail() : -ntail();
                if( _f[_i-1] + UNI*(_f[_i]-_f[_i-1]) < exp(-.5*x*x) ) return x;
                /* else reject and start all over */
                _h = SHR3;
                _i = _h&127;
                if(abs(_h) < _k[_i]) return (_h*_w[_i]);
        }
}

void ntable(void) {
        const double max_h=2147483648.0;
        double x_curr=3.442619855899, x_prev=x_curr, v=9.91256303526217e-3, q;
        int i;

        q = v/exp(-.5*x_curr*x_curr);
        _k[0] = (x_curr/q)*max_h;
```

```
        _w[0] = q/max_h;
        _f[0] = exp(-.5*x_curr*x_curr);
        _k[127] = 0;
        _w[1] = x_curr/max_h;
        _f[127] = 1.;
        for(i=1; i<127; i++) {
                x_curr = sqrt(-2.*log(v/x_curr+exp(-.5*x_curr*x_curr)));
                _k[i] = (x_curr/x_prev) * max_h;
                x_prev = x_curr;
                _f[i] = exp(-.5*x_curr*x_curr);
                _w[i+1] = x_curr/max_h;
                printf("Variables for run %5d:\n", i);
                printf("_w=%.14f; x_curr=%f; _k=%ld\n", _w[i], x_curr,
                        _k[i]);
        }
}


int main(int argc, char* argv[])
{
        int howmany = 100, count;
        clock_t initial, final;
        double x = 42;
        double total = 0;
        double sum_of_squares = 0;
        double mean, variance;
        initial = clock();
        SEED(86947731);
        ntable();
        final = clock();
        printf("Initialization time: %lu seconds\n",
                ( (final-initial)/CLOCKS_PER_SEC) );
        printf("or: %lu cycles at %lu cycles/second \n",
                final-initial,
                CLOCKS_PER_SEC);
        initial = clock();
        for(count=0; count<howmany; count++) {
                x = RNOR;
                total += x;
                sum_of_squares += x*x;
        }
        final = clock();
        printf("Time for %d random normals: %lu seconds\n",
                howmany,
                ( (final-initial)/CLOCKS_PER_SEC) );
        printf("Last random normal: %f\n", x);
        printf("Mean: %f\n", mean=(total/howmany) );
        printf("Sample variance: %f\n",
```

```
                variance=(sum_of_squares - total*mean)/(howmany-1) );
        printf("Sample deviation: %f\n", sqrt(variance) );
        return 0;
}
```

Another Example: The Chi Distribution

Let's take another case to see what issues arise in adapting this method to other distributions. For an example, we'll take the chi distribution, which has p.d.f.

$$f(x) = \frac{x^{r-1}\exp(-x^2/2)}{2^{\frac{r}{2}-1}\Gamma(\frac{r}{2})}$$

where r is the number of degrees of freedom, and the domain is $x \geq 0$. As you might expect from the name, the chi distribution is defined so that, if X has a chi distribution, then X^2 has a chi-square distribution. So one way we could generate random chi's is to take r normal variates, sum their squares, and take the square root. This is inefficient, especially for large r. Another way is that the cumulative distribution for χ is just an incomplete gamma function; we can choose a uniform random variable $U = u$ and let χ be the value of x satisfying $F(x) = u$. But inverting the incomplete gamma function is also unappealing.

Which brings us to one of the two main caveats to this method. (The other is picking a variate from the tail, but we'll get to that.) In the case of the normal distribution, the p.d.f. was invertible, and we had a nice recurrence relation between x_{i-1} and x_i. In general, though, we have to invert the function numerically, and we have to do that for every rectangle. We only have to do it once for a given distribution, precision, and number of rectangles; but in the case of the chi distribution, we still have to do it once for any given number of degrees of freedom. For this example, I arbitrarily chose the case of $r = 2$ degrees of freedom and $n = 8$ boxes. (The case $r = 1$ is uninteresting; it's just a normal distribution, except for the loss of negative values.)

We can ignore the proportionality constant, so we need only consider $f(x) = xe^{-x^2/2}$, which has a maximum of about 0.606 at $x = 1$. If we guess the height of the first box to be about 0.05 (slightly less than a tenth of the total height, since the boxes get narrower and taller as we go up), that forces all the other numbers. Using Maple, we can thus calculate one set of boxes. (We hope to get eight boxes.)

```
delta0:=0.05;
```
$\delta 0 = .05$
```
solve(x*exp(-x^2/2)=delta0,x);
```
$.05006269611375264, 2.842705995089633$
```
v:=exp(-.5*2.842705995089633^2)+.05*2.842705995089633;
```
$v = .1597241746104702$
```
q:=v/0.05;
```
$q = 3.194483492209404$
```
delta1:=v/(2.842705995089633-0.05006269611375264);
```
$\delta 1 = .05719462083433439$
```
solve(x*exp(-x^2/2)=delta0+delta1,x);
```
$.1078195063521113, 2.5411581986993854$

$\vdots$

The other problem with this method in general is that we need a method for generating tail variants (say, for $x \geq r$). One method, if we know the cumulative distribution function, is to pick a random number u uniformly on $[F(x), 1]$, and to solve for $F^{-1}(u)$. For instance, the exponential distribution inverts nicely and gives us $x = r - \ln(u)$. Unfortunately, the incomplete gamma function is not so neatly invertible, and it would defeat the purpose of this method if we had to do a long computation every time we wanted a tail variant. Fortunately, another method is the basic rejection method: replace the chi-distribution tail by a larger but more tractable tail, and throw out what we don't want. Even a rough calculation shows $\exp(-x) > x \times \exp(-x^2/2)$ for $x > 2$, say, so we can use the easy-to-compute exponential tail. Pick an x-coordinate as above; then, since we don't know what its y-coordinate was, pick one uniformly between 0 and e^{-x}. Then test and see if (x, y) falls under the chi-distribution graph.

The disadvantage of this is that the exponential curve isn't a particularly good approximation for the very rapid decay of the chi curve, so we toss out a lot of points. We could make a better approximation, though, if we so desired. In any case, the large boxes we're using in the $n = 8$ case also include a large proportion of points not in the distribution. (Remember, our starting height of 0.05 was just a guess, so the fit of the boxes isn't very good either.)

Just for comparison's sake, when I ran this to produce 1000 chi deviates, the program required 2600 uniform deviates. That's a ratio of 2.6, which isn't very good; we could have generated two normal deviates, summed their squares, and taken the square root for a ratio of only slightly more than 2. On the other hand, it would be easy to optimize this routine, and it's not clear how to improve on the two-normal-deviates method.

Code for Generating $\chi(2)$ Random Variables

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

static unsigned long _jz, _jsr=123456789;

#define SEED(jsrseed) (_jsr ^= (jsrseed))
#define SHR3 (_jz=_jsr, _jsr^=(_jsr<<13), _jsr^=(_jsr>>17),
_jsr^=(_jsr<<5), _jz+_jsr)
#define UNI (.5 + (signed) SHR3 * .2328306e-9)

static unsigned long _h;
static unsigned long _j[8], _k[8];
static unsigned long _i;
static float _w[8], _f[8];

static float _l[8], _r[8];      /* Left and right ends of boxes */

#define RCHI (_h=SHR3,_i=_h&7,(_h<_k[_i]&&_h>_j[_i])? _h*_w[_i]+_l[_i]: cfix())

/* cfix() generates variates from the residue when rejection in RCHI occurs. */
```

```
float ctail(void) {
        const float r = 2.842705995089633;      /* r = start of right tail */
        static float x, y;
        do {
                x = r-log(UNI);
                y = UNI*exp(-x);
        } while (y > x*exp(-x*x/2));
        return x;
}

float cfix(void) {
        static float x;
        for(;;){
                x = _h*_w[_i]+_l[_i];
                if ((_i==0) && (_h>_k[0])) return (_h>0)? ctail() : -ctail();
                else if ((_i==0) && (UNI*_f[0] < x*exp(-x*x/2)) ) return x;
                else if ((_i>0) &&
                         (_f[_i-1]+UNI*(_f[_i]-_f[_i-1]) < x*exp(-x*x/2))
                        ) return x;
                /* else reject and start all over */
                _h = SHR3;
                _i = _h&7;
                if(_h<_k[_i] && _h>_j[_i]) return (_h*_w[_i]+_l[_i]);
        }
}

void ctable(void) {
        const double max_h=2*2147483648.0;
        int i;

        _f[0]=0.05;
        _f[1]=0.10719462083433439;
        _f[2]=0.1736421903828149;
        _f[3]=0.2500505649652229;
        _f[4]=0.3391243953209790;
        _f[5]=0.4472926161945060;
        _f[6]=0.5944802858700420;
        _f[7]=1.158865828976586;

        _l[0]=0.0;
        _l[1]=0.05006269611375264;
        _l[2]=0.1078195063521113;
        _l[3]=0.1763637986783827;
        _l[4]=0.2585494762461569;
        _l[5]=0.3621019904276302;
        _l[6]=0.5092099379297638;
        _l[7]=0.8618467310037498;
```

```
        _r[0]=3.194483492209404;
        _r[1]=2.842705995089633;
        _r[2]=2.511581986993854;
        _r[3]=2.266765217153845;
        _r[4]=2.051715594426328;
        _r[5]=1.838729537045261;
        _r[6]=1.594383546307218;
        _r[7]=1.144852163303276;

        for(i=0; i<8; i++) {
                _w[i]=(_r[i]-_l[i])/max_h;
        }

        _j[7]=0; _k[7]=0;
        for(i=0; i<7; i++) {
                _j[i]=(_l[i+1]-_l[i])/_w[i];
                _k[i]=(_k[i+1]-_l[i])/_w[i];
        }
}


int main(int argc, char* argv[])
{
        int howmany = 10000, count;
        clock_t initial, final;
        double x = 42;
        initial = clock();
        SEED(86947731);
        ctable();
        final = clock();
        printf("Initialization time: %lu seconds\n",
                ( (final-initial)/CLOCKS_PER_SEC) );
        printf("or: %lu cycles at %lu cycles/second \n",
                final-initial,
                CLOCKS_PER_SEC);
        initial = clock();
        for(count=0; count<howmany; count++) {
                x = RCHI;
        }
        final = clock();
        printf("Time for %d random normals: %lu seconds\n",
                howmany,
                ( (final-initial)/CLOCKS_PER_SEC) );
        printf("Last random chi(2): %f\n", x);
        return 0;
}
```

34.5 Growth Factors in Gaussian Elimination

The standard algorithm for solving linear systems of equations is Gaussian Elimination with partial pivoting. (See any numerical analysis text for further discussion.)

Trefethen and Schreiber write that "average growth vectors in Gaussian elimination exhibit a mild $n^{2/3}$ dependence on n, at lease for $n \leq 1024$". The data on the next page say otherwise. Taking $\frac{g}{\sqrt{n}}$ for many samples and then computing the empirical cdf indicates a sequence of curves that appear to converge while $\frac{g}{n^{2/3}}$ indicates no such behavior. The Trefethen, Schreiber conclusion was based upon the taking of averages rather than any play of the distribution.

34.6 Numerical Algorithms Performed Statistically

It has been rediscovered many times that many of the standard numerical linear algebra algorithms may be performed mentally on random matrices with normally distributed coefficients with particularly simple results.

As our first illustration, consider a random matrix $A \in \mathbb{R}^{n,m}$ with iid standard normal entries. Remember that if $x \in \mathbb{R}^n$ is a vector of iid standard normals, then its length $\|x\|$ has the χ_n distribution with density

$$\chi_n \text{ density: } \frac{2^{1-n/2}}{\Gamma(n/2)} e^{-r^2/2} r^{n-1} dr.$$

Remember the standard QR procedure. First we multiply A on the left by Householder matrix defined by the first column of A. It may be psychologically helpful (but mathematically unnecessary) to pretend that we first generate only the first column of A, then build the Householder matrix H_1 from this first column, and apply it to the column. The result must be the length of the column times e_1 that is a χ_n variable times e_1. We can now generate the remaining columns of A. Premultiplying these columns by H will not change the distribution at all. Continuing the procedure, we will compute an upper triangular matrix R with r_{ii} distributed as χ_{n+1-i} and strictly upper triangular elements standard normals. All elements are independent. In summary

$$r_{ij} \text{ is } \begin{cases} N(0,1) \text{ if } i < j \\ \chi_{n+1-i} \text{ if } i = j \end{cases}.$$

This result can already be read straight off the Jacobian computations, but it is somehow more satisfying to see it directly in this manner.

If $A \in \mathbb{R}^{n,n}$, the first step of an eigenvalue computation is the conversion to so-called upper Hessenberg form, H. This means that $H_{ij} = 0$ if $i > j + 1$, i.e. the matrix is upper-triangular except for one subdiagonal. It is always possible through Householder transformations to find a Q such that $Q^T AQ$ is upper Hessenberg. If A is a square matrix of iid standard normals it is easy to see that

$$h_{ij} \text{ is } \begin{cases} N(0,1) \text{ if } i \leq j \\ \chi_{n-i} \text{ if } i = j + 1 \end{cases}.$$

If $A \in \mathbb{R}^{n,m}$ again it may be bidiagonalized by multiplications on the left and right by Householder matrices. Upon doing so we obtain $A = UBV^T$, where B is bidiagonal with

$$b_{ij} \text{ is } \begin{cases} \chi_{n+1-i} \text{ if } i = j \\ \chi_{m-i} \text{ if } i = j - 1 \end{cases}.$$

If $S \in R^{n,n}$ is symmetric, then there are Householder similarity transformations to symmetric tridiagonal form. Let $S = (A + A^T)/2$, where $A \in \mathbb{R}^{n,n}$ had iid standard normal elements, i.e. S is from the Gaussian Orthogonal Ensemble. Tridiagonalization does not alter the diagonal, but does give us variables with chi distributions on the super and subdiagonal:

$$t_{ij} \text{ is } \begin{cases} N(0,1) \text{ if } i = j \\ \chi_{n-i}/\sqrt{2} \text{ if } i = j-1 \\ t_{ji} \text{ if } j = i-1 \end{cases} .$$

The quantities in all of the formulas are readily seen to be independent. Random matrix experiments may be performed with sometimes a considerable savings of effort by generating random matrices directly in this reduced form. Similar results are readily computed for matrices of complex normals.

34.7 Condition Number Distributions

34.8 Numerical Simulations of beta-ensembles and Painlevé

[This section was originally written by Per-Olof Persson (persson@mit.edu)]

34.8.1 Largest Eigenvalue Distributions

In this section, the distributions of the largest eigenvalue of matrices in the β-ensembles are studied. Histograms are created first by simulation, then by solving the Painlevé II nonlinear differential equation.

Simulation

The *Gaussian Unitary Ensemble* (GUE) is defined as the Hermitian $n \times n$ matrices A, where the diagonal elements x_{jj} and the upper triangular elements $x_{jk} = u_{jk} + iv_{jk}$ are independent Gaussians with zero-mean, and

$$\begin{cases} \text{Var}(x_{jj}) = 1, & 1 \le j \le n, \\ \text{Var}(u_{jk}) = \text{Var}(v_{jk}) = \frac{1}{2}, & 1 \le j < k \le n. \end{cases} \tag{34.1}$$

Since a sum of Gaussians is a new Gaussian, an instance of these matrices can be created conveniently in MATLAB:

```
A=randn(n)+i*randn(n);
A=(A+A')/2;
```

The largest eigenvalue of this matrix is about $2\sqrt{n}$. To get a distribution that converges as $n \to \infty$, the shifted and scaled largest eigenvalue $\lambda'_{\max}$ is calculated as

$$\lambda'_{\max} = n^{\frac{1}{6}} \left(\lambda_{\max} - 2\sqrt{n} \right) . \tag{34.2}$$

It is now straight-forward to compute the distribution for $\lambda'_{\max}$ by simulation:

```
for ii=1:trials
  A=randn(n)+i*randn(n);
  A=(A+A')/2;
  lmax=max(eig(A));
  lmaxscaled=n^(1/6)*(lmax-2*sqrt(n));
  % Store lmax
end

% Create and plot histogram
```

The problem with this technique is that the computational requirements and the memory requirements grow fast with n, which should be as large as possible in order to be a good approximation of infinity. Just storing the matrix A requires n^2 double-precision numbers, so on most computers today n has to be less than 10^4. Furthermore, computing all the eigenvalues of a full Hermitian matrix requires a computing time proportional to n^3. This means that it will take many days to create a smooth histogram by simulation, even for relatively small values of n.

To improve upon this situation, another matrix can be studied that has the same eigenvalue distribution as A above. In [**?**], it was shown that this is true for the following *symmetric* matrix when $\beta = 2$:

$$H_\beta \sim \frac{1}{\sqrt{2}} \begin{pmatrix} N(0,2) & \chi_{(n-1)\beta} & & & \\ \chi_{(n-1)\beta} & N(0,2) & \chi_{(n-2)\beta} & & \\ & \ddots & \ddots & \ddots & \\ & & \chi_{2\beta} & N(0,2) & \chi_\beta \\ & & & \chi_\beta & N(0,2) \end{pmatrix}. \tag{34.3}$$

Here, $N(0,2)$ is a zero-mean Gaussian with variance 2, and χ_d is the square-root of a χ^2 distributed number with d degrees of freedom. Note that the matrix is symmetric, so the subdiagonal and the superdiagonal are always equal.

This matrix has a tridiagonal sparsity structure, and only $2n$ double-precision numbers are required to store an instance of it. The time for computing the largest eigenvalue is proportional to n, either using Krylov subspace based methods or the method of bisection [43].

In MATLAB, the built-in function `eigs` can be used, although that requires dealing with the sparse matrix structure. There is also a large amount of overhead in this function, which results in a relatively poor performance. Instead, the function `maxeig` is used below to compute the eigenvalues. This is not a built-in function in MATLAB, but it can be downloaded from `http://www.mit.edu/~persson/mltrid`. It is based on the method of bisection, and requires just two ordinary MATLAB vectors as input, corresponding to the diagonal and the subdiagonal.

It also turns out that only the first $10n^{\frac{1}{3}}$ components of the eigenvector corresponding to the largest eigenvalue are significantly greater than zero. Therefore, the upper-left n_{cutoff} by n_{cutoff} submatrix has the same largest eigenvalue (or at least very close), where

$$n_{\text{cutoff}} \approx 10n^{\frac{1}{3}}. \tag{34.4}$$

Matrices of size $n = 10^{12}$ can then easily be used since the computations can be done on a matrix of size only $10n^{\frac{1}{3}} = 10^5$. Also, for these large values of n the approximation $\chi_n^2 \approx n$ is accurate.

A histogram of the distribution for $n = 10^9$ can now be created using the code below.

```
n=1e9;
nrep=1e4;
beta=2;

cutoff=round(10*n^(1/3));
d1=sqrt(n-1:-1:n+1-cutoff)'/2/sqrt(n);

ls=zeros(1,nrep);
for ii=1:nrep
  d0=randn(cutoff,1)/sqrt(n*beta);
  ls(ii)=maxeig(d0,d1);
end

ls=(ls-1)*n^(2/3)*2;

histdistr(ls,-7:0.2:3)
```

Code 34.1

where the function `histdistr` below is used to histogram the data. It assumes that the histogram boxes are equidistant.

```
function [xmid,H]=histdistr(ls,x)

dx=x(2)-x(1);
H=histc(ls,x);
H=H(1:end-1);
H=H/sum(H)/dx;
xmid=(x(1:end-1)+x(2:end))/2;

bar(xmid,H)
grid on
```

Code 34.2

The resulting distribution is shown in Figure 34.1, together with distributions for $\beta = 1$ and $\beta = 4$. The plots also contain solid curves representing the "true solutions" (see next section).

Painlevé II

Instead of using simulation to plot the distributions of the largest eigenvalues, it can be computed from the solution of the Painlevé II nonlinear differential equation [452]:

$$q'' = sq + 2q^3 \tag{34.5}$$

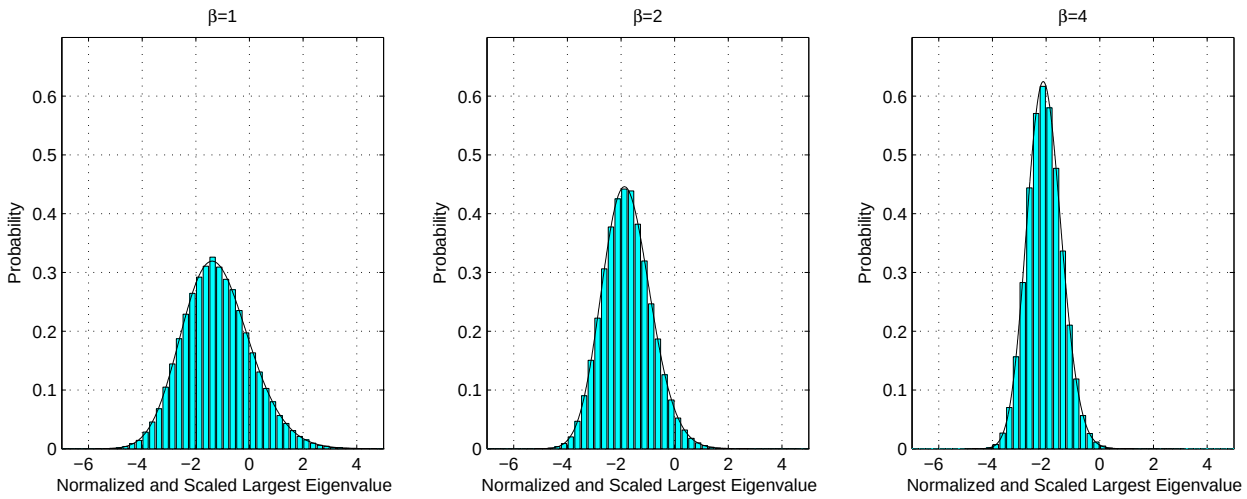


Figure 34.1: Probability distribution of scaled largest eigenvalue (10^5 repetitions, $n = 10^9$)

with the boundary condition

$$q(s) \sim \mathrm{Ai}(s), \qquad \text{as } s \to \infty. \tag{34.6}$$

The probability distribution $f_2(s)$, corresponding to $\beta = 2$, is then given by

$$f_2(s) = \frac{d}{ds} F_2(s), \tag{34.7}$$

where

$$F_2(s) = \exp\left(- \int_s^\infty (x - s) q(x)^2 \, dx \right). \tag{34.8}$$

The distributions for $\beta = 1$ and $\beta = 4$ are the derivatives of

$$F_1(s)^2 = F_2(s) e^{-\int_s^\infty q(x)\, dx} \tag{34.9}$$

and

$$F_4\left(\frac{s}{2^{\frac{2}{3}}} \right)^2 = F_2(s) \left(\frac{e^{\frac{1}{2}\int_s^\infty q(x)\, dx} + e^{-\frac{1}{2}\int_s^\infty q(x)\, dx}}{2} \right)^2. \tag{34.10}$$

To solve this numerically using MATLAB, first rewrite (34.5) as a first-order system:

$$\frac{d}{ds} \begin{pmatrix} q \\ q' \end{pmatrix} = \begin{pmatrix} q' \\ sq + 2q^3 \end{pmatrix}. \tag{34.11}$$

This can be solved as an initial-value problem starting at $s = s_0 =$ sufficiently large positive number, and integrating along the negative s-axis. The boundary condition (34.6) then becomes the initial values

$$\begin{cases} q(s_0) &= \mathrm{Ai}(s_0) \\ q'(s_0) &= \mathrm{Ai}'(s_0). \end{cases} \tag{34.12}$$

Although the distributions can be computed from $q(s)$ as a post-processing step, it is most convenient to add a few variables and equations to the ODE system. When computing $F_2(s)$, the quantity $I(s) = \int_s^\infty (x-s)q(x)^2\,dx$ is required. Differentiate this twice to get

$$I'(s) = -\int_s^\infty q(x)^2\,dx \tag{34.13}$$

and

$$I''(s) = q(s)^2. \tag{34.14}$$

Add these equations and the variables $I(s), I'(s)$ to the ODE system, and the solver will do the integration. This is not only easier and gives less code, it will also give a much more accurate solution since the same tolerance requirements are imposed on $I(s)$ as on the solution $q(s)$.

In a similar way, the quantity $J(s) = \int_s^\infty q(x)\,dx$ is needed when computing $F_1(s)$ and $F_4(s)$. This is handled by adding the variable $J(s)$ and the equation $J'(s) = -q(s)$.

The final system now has the form

$$\frac{d}{ds}\begin{pmatrix} q \\ q' \\ I \\ I' \\ J \end{pmatrix} = \begin{pmatrix} q' \\ sq + 2q^3 \\ I' \\ q^2 \\ -q \end{pmatrix} \tag{34.15}$$

with the initial condition

$$\begin{pmatrix} q(s_0) \\ q'(s_0) \\ I(s_0) \\ I'(s_0) \\ J(s_0) \end{pmatrix} = \begin{pmatrix} \mathrm{Ai}(s_0) \\ \mathrm{Ai}'(s_0) \\ \int_{s_0}^\infty (x-s_0)\mathrm{Ai}(x)^2\,dx \\ \mathrm{Ai}(s_0)^2 \\ \int_{s_0}^\infty \mathrm{Ai}(x)\,dx \end{pmatrix}. \tag{34.16}$$

This problem can be solved in just a few lines of MATLAB code using the built-in Runge-Kutta based ODE solver `ode45`. First define the system of equations as an inline function

```
deq=inline('[y(2); s*y(1)+2*y(1)^3; y(4); y(1)^2; -y(1)]','s','y');
```

Next specify the integration interval and the desired output times.

```
s0=5;
sn=-8;
sspan=linspace(s0,sn,1000);
```

The initial values can be computed as

```
y0=[airy(s0); airy(1,s0); ...
    quadl(inline('(x-s0).*airy(x).^2','x','s0'),s0,20,1e-25,0,s0); ...
    airy(s0)^2; quadl(inline('airy(x)'),s0,20,1e-18)];
```

where the `quadl` function is used to numerically approximate the integrals in (34.16). Now, the integration tolerances can be set and the system integrated:

```
opts=odeset('reltol',1e-13,'abstol',1e-15);
[s,y]=ode45(deq,sspan,y0,opts);
```

The five dependent variables are now in the columns of the MATLAB variable `y`. Using these, $F_2(s), F_1(s)$, and $F_4(s)$ become

```
F2=exp(-y(:,3));
F1=sqrt(F2.*exp(-y(:,5)));
F4=sqrt(F2).*(exp(y(:,5)/2)+exp(-y(:,5)/2))/2;
s4=s/2^(2/3);
```

The probability distributions $f_2(s), f_1(s)$, and $f_4(s)$ could be computed by numerical differentiation:

```
f2=gradient(F2,s);
f1=gradient(F1,s);
f4=gradient(F4,s4);
```

but it is more accurate to first do the differentiation symbolically:

$$f_2(s) = -I'(s)F_2(s) \tag{34.17}$$

$$f_1(s) = \frac{1}{2F_1(s)}\left(f_2(s) + q(s)F_2(s)\right)e^{-J(s)} \tag{34.18}$$

$$f_4(s) = \frac{1}{2^{\frac{1}{3}}4F_4(s)}\left(f_2(s)\left(2 + e^{J(s)} + e^{-J(s)}\right) + F_2(s)q(s)\left(e^{-J(s)} - e^{J(s)}\right)\right) \tag{34.19}$$

and evaluate these expressions:

```
f2=-y(:,4).*F2;
f1=1/2./F1.*(f2+y(:,1).*F2).*exp(-y(:,5));
f4=1/2^(1/3)/4./F4.*(f2.*(2+exp(y(:,5))+exp(-y(:,5)))+ ...
                     F2.*y(:,1).*(exp(-y(:,5))-exp(y(:,5))));
```

Finally, plot the curves:

```
plot(s,f1,s,f2,s4,f4)
legend('\beta=1','\beta=2','\beta=4')
xlabel('s')
ylabel('f_\beta(s)','rotation',0)
grid on
```

The result can be seen in Figure 34.2.

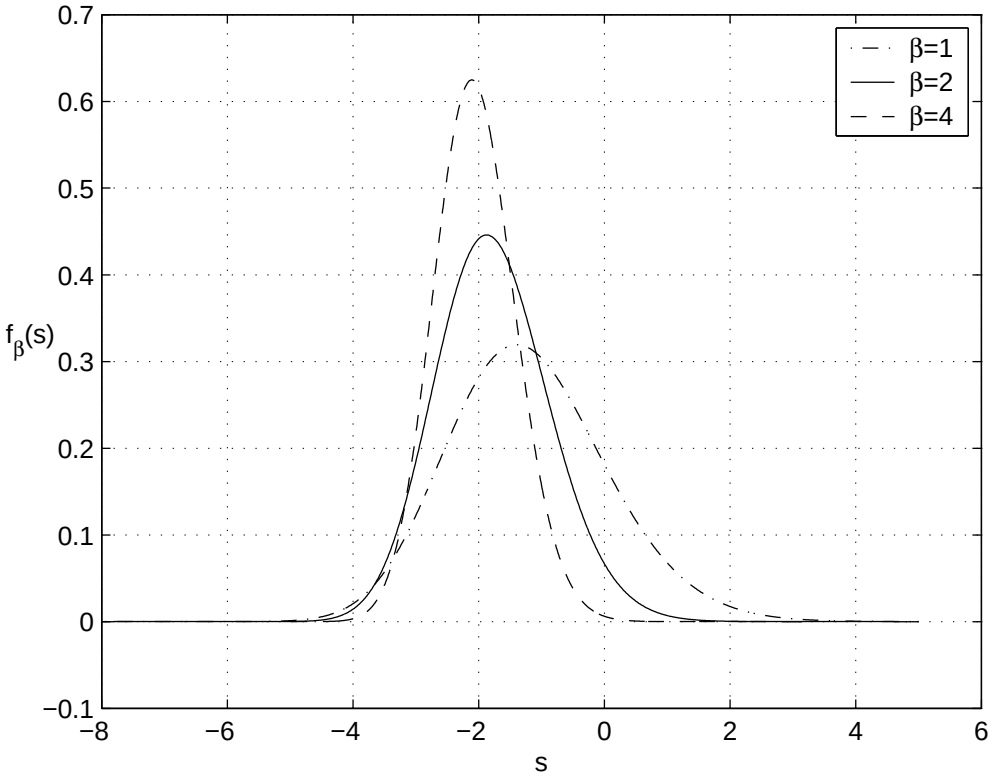


Figure 34.2: The probability distributions $f_1(s)$, $f_2(s)$, and $f_4(s)$, computed using the Painlevé II solution.

34.8.2 Eigenvalue Spacings Distributions

Another quantity with an interesting probability distribution is the spacings of the eigenvalues of random matrices. It turns out that the eigenvalues are almost uniformly distributed, which means that every random matrix gives a large number of spacings. The distributions can then be efficiently computed by simulation.

Two other methods are used to compute the spacings distribution—the solution of the Painlevé V nonlinear differential equation and the eigenvalues of the Prolate matrix. Finally, the results are compared with the spacings of the zeros along the critical line of the Riemann zeta function.

Simulation

As before, the simulations are made with matrices from the Gaussian Unitary Ensemble. The normalized spacings of the eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$ are computed according to

$$\delta'_k = \frac{\lambda_{k+1} - \lambda_k}{\pi\beta}\sqrt{2\beta n - \lambda_k^2}, \qquad k \approx n/2, \tag{34.20}$$

where $\beta = 2$ for the GUE. The distribution of the eigenvalues is almost uniform, with a slight deviation at the two ends of the spectrum. Therefore, only half of the eigenvalues are used, and one quarter of the eigenvalues at each end is discarded.

Again, to allow for a more efficient simulation, the tridiagonal matrix (34.3) is used instead of the full Hermitian matrix. In this case, all the eigenvalues are computed, which can be done in a time proportional to n^2. While this could in principle be done using the MATLAB sparse matrix structure and the `eigs` function, the more efficient `trideig` function is used below to compute all the eigenvalues of a symmetric tridiagonal matrix. It can be downloaded from `http://www.mit.edu/~persson/mltrid`

The histogram can now be computed by simulation with the following lines of code. Note that the function `chi2rnd` from the Statistics Toolbox is required.

```
n=1000;
nrep=1000;
beta=2;

ds=zeros(1,nrep*n/2);
for ii=1:nrep
  l=trideig(randn(n,1),sqrt(chi2rnd((n-1:-1:1)'*beta)/2));
  d=diff(l(n/4:3*n/4))/beta/pi.*sqrt(2*beta*n-l(n/4:3*n/4-1).^2);
  ds((ii-1)*n/2+1:ii*n/2)=d;
end

histdistr(ds,0:0.05:5);
```

Code 34.3: T

he resulting histogram can be found in Figure 34.3. The figure also shows the expected curve as a solid line.

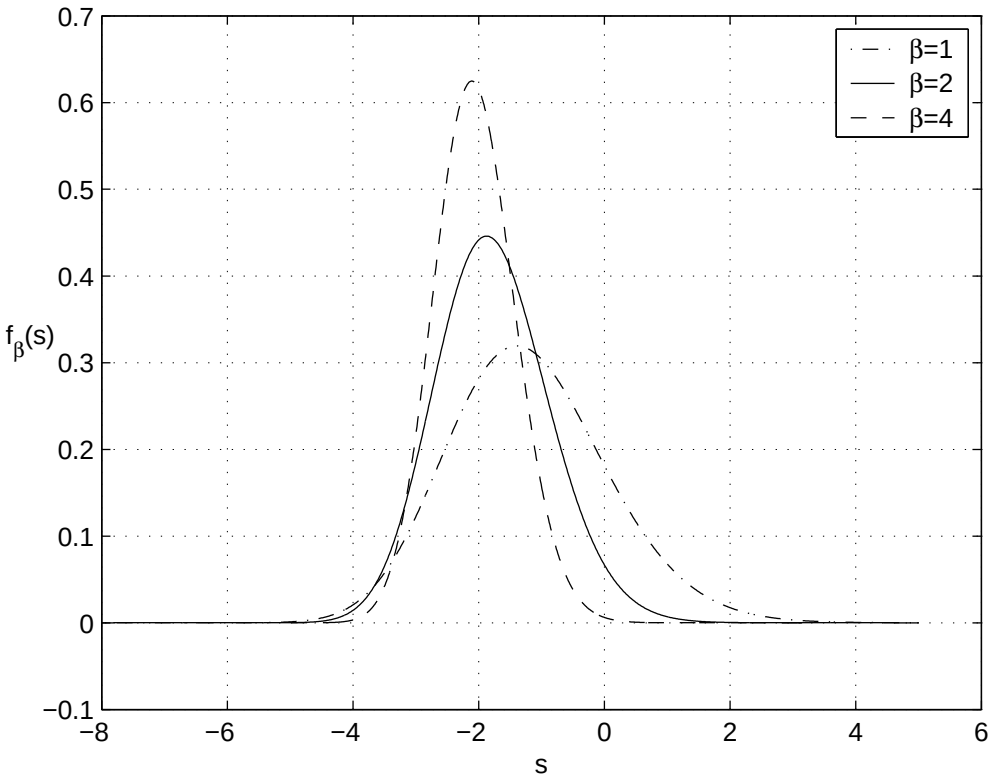


Figure 34.3: Probability distribution of consecutive spacings of random matrix eigenvalues (1000 repetitions, $n = 1000$)

Painlevé V

The probability distribution $p(s)$ for the eigenvalue spacings when $\beta = 2$ can be computed with the solution to the Painlevé V nonlinear differential equation (see [442] for details):

$$(t\sigma'')^2 + 4(t\sigma' - \sigma)\left(t\sigma' - \sigma + (\sigma')^2\right) = 0 \tag{34.21}$$

with the boundary condition

$$\sigma(t) \approx -\frac{t}{\pi} - \left(\frac{t}{\pi}\right)^2, \qquad \text{as } t \to 0^+. \tag{34.22}$$

Then $p(s)$ is given by

$$p(s) = \frac{d^2}{ds^2} E(s) \tag{34.23}$$

where

$$E(s) = \exp\left(\int_0^{\pi s} \frac{\sigma(t)}{t} dt\right). \tag{34.24}$$

Explicit differentiation gives

$$p(s) = \frac{1}{s^2}\left(\pi s \sigma'(\pi s) - \sigma(\pi s) + \sigma(\pi s)^2\right) E(s). \tag{34.25}$$

The second-order differential equation (34.21) can be written as a first-order system of differential equations:

$$\frac{d}{dt}\begin{pmatrix}\sigma \\ \sigma'\end{pmatrix} = \begin{pmatrix}\sigma' \\ -\frac{2}{t}\sqrt{(\sigma - t\sigma')\left(t\sigma' - \sigma + (\sigma')^2\right)}\end{pmatrix}. \tag{34.26}$$

This is solved as an initial-value problem starting at $t = t_0 =$ very small positive number. The value $t = 0$ has to be avoided because of the division by t in the system of equations. This is not a problem, since the boundary condition (34.22) provides an accurate value for $\sigma(t_0)$ (as well as $E(t_0/\pi)$). The boundary conditions for the system (34.26) then become

$$\begin{cases}\sigma(t_0) & = -\frac{t_0}{\pi} - \left(\frac{t_0}{\pi}\right)^2 \\ \sigma'(t_0) & = -\frac{1}{\pi} - \frac{2t_0}{\pi}.\end{cases} \tag{34.27}$$

To be able to compute $E(s)$ using (34.24), the variable

$$I(t) = \int_0^t \frac{\sigma(t')}{t'}\, dt' \tag{34.28}$$

is added to the system, as well as the equation $\frac{d}{dt} I = \frac{\sigma}{t}$. The corresponding initial value is

$$I(t_0) \approx \int_0^{t_0} \left(-\frac{1}{\pi} - \frac{t}{\pi^2}\right) dt = -\frac{t_0}{\pi} - \frac{t_0^2}{2\pi^2}. \tag{34.29}$$

Putting it all together, the final system is

$$\frac{d}{dt}\begin{pmatrix}\sigma \\ \sigma' \\ I\end{pmatrix} = \begin{pmatrix}\sigma' \\ -\frac{2}{t}\sqrt{(\sigma - t\sigma')\left(t\sigma' - \sigma + (\sigma')^2\right)} \\ \frac{\sigma}{t}\end{pmatrix} \tag{34.30}$$

with boundary condition

$$\begin{pmatrix} \sigma(t_0) \\ \sigma'(t_0) \\ I(t_0) \end{pmatrix} = \begin{pmatrix} -\frac{t_0}{\pi} - \left(\frac{t_0}{\pi}\right)^2 \\ -\frac{1}{\pi} - \frac{2t_0}{\pi} \\ -\frac{t_0}{\pi} - \frac{t_0^2}{2\pi^2} \end{pmatrix}. \tag{34.31}$$

This system is defined as an inline function in MATLAB:

```
desig=inline(['[y(2); -2/t*sqrt((y(1)-t*y(2))*' ...
              '(t*y(2)-y(1)+y(2)^2)); y(1)/t]'],'t','y');
```

Specify the integration interval and the desired output times:

```
t0=1e-12;
tn=16;
tspan=linspace(t0,tn,1000);
```

Set the initial condition:

```
y0=[-t0/pi-(t0/pi)^2; -1/pi-2*t0/pi; -t0/pi-t0^2/2/pi^2];
```

Finally, set the integration tolerances and call the solver:

```
opts=odeset('reltol',1e-13,'abstol',1e-14);
[t,y]=ode45(desig,tspan,y0,opts);
```

The solution components are now in the columns of `y`. Use these to evaluate $E(s)$ and $p(s)$:

```
s=t/pi;
E=exp(y(:,3));
p=1./s.^2.*E.*(t.*y(:,2)-y(:,1)+y(:,1).^2);
p(1)=2*s(1); % Fix due to cancellation
```

A plot of $p(s)$ can be made with the command

```
plot(s,p)
grid on
```

and it can be seen in Figure 34.4. Plots are also shown of $E(s)$ and $\sigma(t)$.

The Prolate Matrix

Another method to calculate the distribution of the eigenvalue spacings is to compute the eigenvalues λ_i of the operator

$$f(y) \to \int_{-1}^{1} Q(x,y) f(y)\, dy, \qquad Q(x,y) = \frac{\sin\left((x-y)\pi t\right)}{(x-y)\pi}. \tag{34.32}$$

Then $E(2t) = \prod_i (1-\lambda_i)$, and $p(s)$ can be computed as before. To do this, first define the infinite symmetric Prolate matrix:

$$A_\infty = \begin{pmatrix} a_0 & a_1 & \dots \\ a_1 & a_0 & \dots \\ \vdots & \vdots & \ddots \end{pmatrix} \tag{34.33}$$

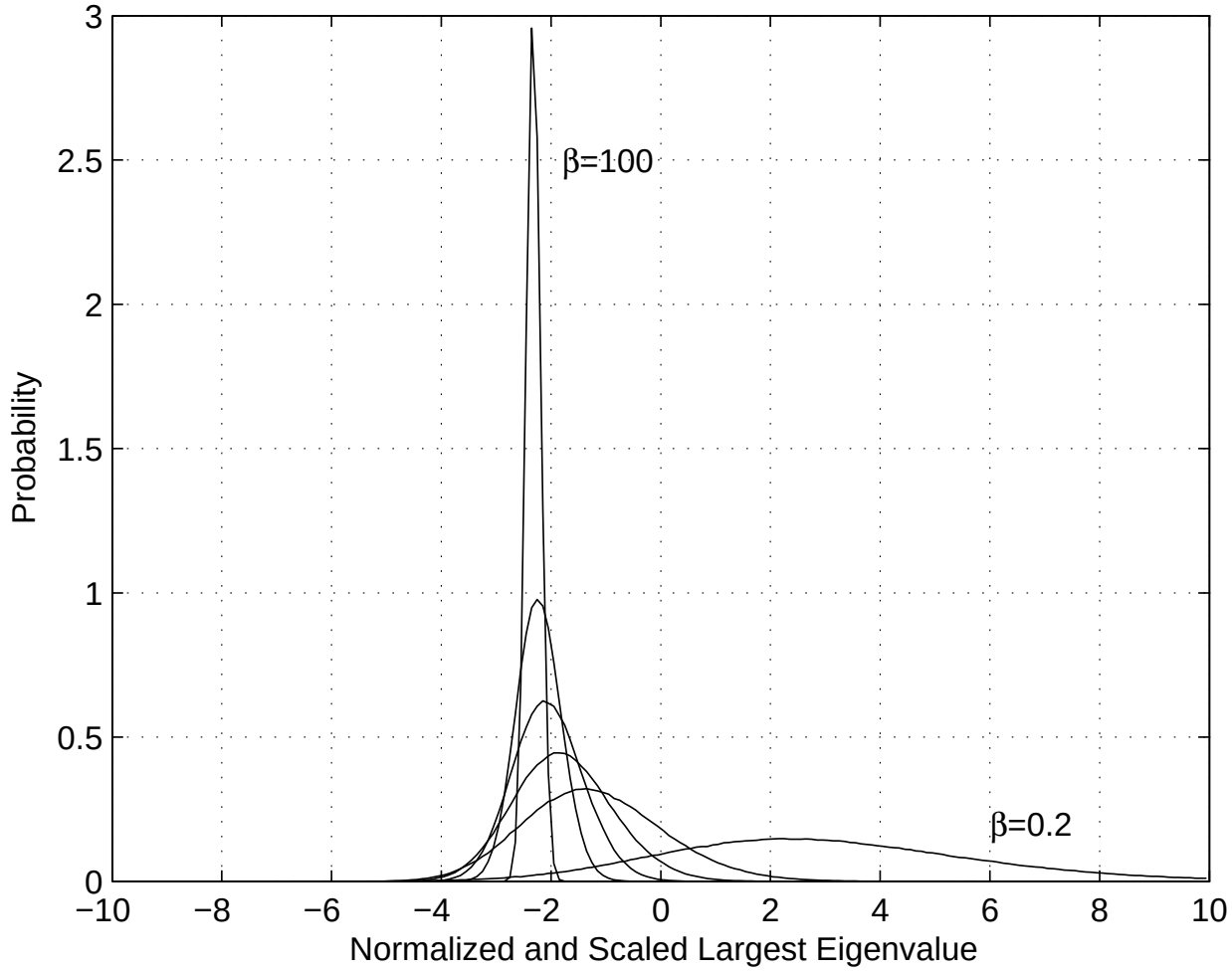


Figure 34.4: Painlevé V (left), $E(s)$ and $p(s)$ (right).

with $a_0 = 2w$, $a_k = (\sin 2\pi wk)/\pi k$ for $k = 1, 2, \ldots$, and $0 < w < \frac{1}{2}$. A discretization of $Q(x, y)$ is achieved by setting $w = t/n$ and extracting the upper-left $n \times n$ submatrix A_n of A_∞.

Below, the full matrix A_n is used, and all the eigenvalues are computed in n^3 time using the `eig` function. However, A_n commutes with the following symmetric tridiagonal matrix [413], and therefore has the same eigenvectors:

$$T_n = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{pmatrix} \tag{34.34}$$

where

$$\begin{cases} \alpha_k &= \left(\frac{n+1}{2} - k\right)^2 \cos 2\pi w \\ \beta_k &= \frac{1}{2}k(n-k). \end{cases} \tag{34.35}$$

It is then in principle possible to use the new techniques described in [108] to compute all the eigenvalues and eigenvectors of T_n in n^2 time, and then get the eigenvalues of A_n by dot products. This is not done in this example.

The code for computing $E(s)$ is shown below. This time, $p(s)$ is evaluated by numerical differentiation since no information about the derivative of $E(s)$ is available.

```
s=0:0.01:5;
n=100;
E0=zeros(size(s));
for ii=1:length(s)
  Q=gallery('prolate',n,s(ii)/2/n);
  E0(ii)=prod(1-eig(Q));
end
p0=gradient(gradient(E0,s),s);
```

Code 34.4

To improve the accuracy in $E(s)$, Richardson extrapolation can be used. This is done as follows, where the values are assumed to converge as $1/n^2$:

N	Error 0	Error 1	Error 2	Error 3
20	0.2244			
40	0.0561	0.7701		
80	0.0140	0.0483	0.5486	
160	0.0035	0.0032	0.0323	2.2673
	$\cdot 10^{-3}$	$\cdot 10^{-7}$	$\cdot 10^{-8}$	$\cdot 10^{-11}$

Table 34.1: Difference between Prolate solution $E_1(s)$ and Painlevé V solution $E(s)$ after 0, 1, 2, and 3 Richardson extrapolations.

```
% ... Compute s and E using Painleve V in previous section

Es=zeros(length(t),0);
E1=zeros(size(s));
for n=20*2.^(0:3)
  for ii=1:length(s)
    Q=gallery('prolate',n,s(ii)/2/n);
    E1(ii)=prod(1-eig(Q));
  end
  Es=[Es,E1];
end

for ii=1:3
  max(abs(Es-E(:,ones(1,size(Es,2)))))
  Es=Es(:,2:end)+diff(Es,1,2)/(2^(ii+1)-1);
end
max(abs(Es-E))
```

Code 34.5

The errors $\max_{0\le s\le 5}|E_1(s)-E(s)|$ are shown in Table 1, for $n = 20, 40, 80,$ and 160. The error after all extrapolations is of the same order as the "exact solution" using Painlevé V.

Riemann Zeta Zeros

It has been observed that the zeros of the Riemann zeta function along the critical line $\mathrm{Re}(z) = \frac{1}{2}$ behave similar to the eigenvalues of random matrices in the GUE. Here, the distribution of the scaled spacings of the zeros is compared to the corresponding distribution for eigenvalues computed using the Painlevé V equation from the previous chapters.

Define the nth zero $\gamma_n = n^{\text{th}}$ as

$$\zeta\left(\frac{1}{2} + i\gamma_n\right) = 0, \qquad 0 < \gamma_1 < \gamma_2 < \dots \tag{34.36}$$

Compute a normalized spacing:

$$\tilde{\gamma}_n = \frac{\gamma_n}{\text{av spacing near } \gamma_n} = \gamma_n \cdot \left[\frac{\log \gamma_n/2\pi}{2\pi}\right]. \tag{34.37}$$

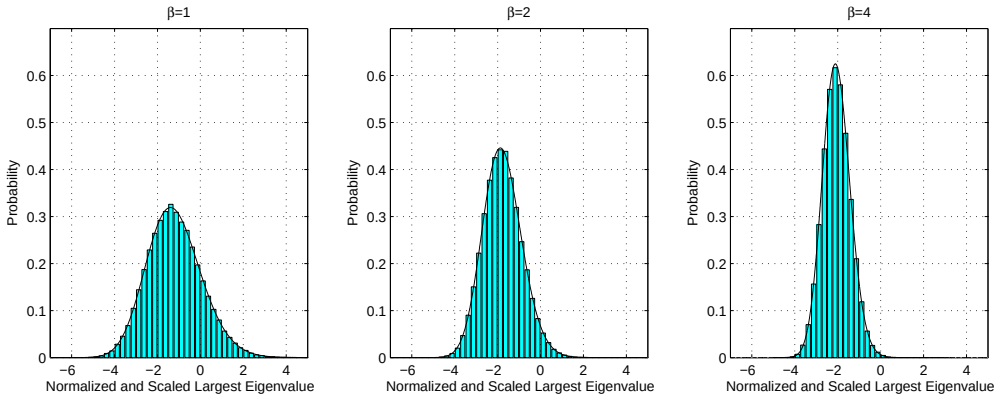


Figure 34.5: Probability distribution of consecutive spacings of Riemann zeta zeros (30,000 zeros, $n \approx 10^{12}, 10^{21}, 10^{22}$)

Zeros of the Riemann zeta function can be downloaded from [338]. Assuming that the MATLAB variable `gamma` contains the zeros, and the variable `offset` the offset, these two lines compute the consecutive spacings $\tilde{\gamma}_{n+1} - \tilde{\gamma}_n$ and plot the histogram:

```
delta=diff(gamma)/2/pi.*log((gamma(1:end-1)+offset)/2/pi);
histdistr(delta,0:0.05:5.0);
```

The result can be found in Figure 34.5, along with the Painlevé V distribution. The curves are indeed in good agreement, although the number of samples here is a little to low to get a perfect match.

Chapter 35

Wireless Communication

The following papers discuss the applications of random matrices to wireless communication.
[465] [80] [466] [211] [79] [5] [463] [135] [271] [464] [210]

35.1 The CDMA Problem [462]

Here is a specific problem in communications that has been solved by other means, but then revisited with the framework of free probability in mind. Very neat actually. This entire section summarizes the work in [462].

Description of the System

We look at the performance of a system of mobile users trying to communicate with a basestation over shared wireless medium, for example a cellular phone network. An example of such a system is shown in Figure **??**, with four mobile phones trying to communicate simultaneously with a basestation. Since the wireless bandwidth is completely shared by all users all the time, a technique call Code-Division Multiple Access (CDMA) is used to allow the basestation to separate out the signals from the individual users. Each user k is assigned a signature sequence $\mathbf{s}_k$, which it uses to multiply its data (strictly speaking, this is direct-sequence CDMA, but that is not terribly important here). The signature sequences can be thought of as vectors of length N, where each element could be as simple as ± 1. For simplicity, we can assume that the $\mathbf{s}_i$'s are orthogonal, but that is not a requirement. At the basestation, the signal seen is

$$y = \sum_{k=1}^{K} x_k \mathbf{s}_k + w, \tag{35.1}$$

where x_k is the zero-mean signal sent by the kth user, K is the total number of users, and $w \sim N(0, \sigma^2 \mathbf{I})$ is the additive Gaussian noise seen by the receiver. Each user has an averaged received power $p_k = \mathbf{E}\left[x_k^2\right]$. Since there are K vectors of length N, we are effectively trying to communicate using a K-dimensional subspace of an N-dimensional space. Alternately, N is the degrees of freedom in the system. The receiver must then try to estimate the signal transmitted by each user from their overlapping received vector y, and we assume that it knows the signatures and received power of each user.

The Receiver Structure

One class of receivers are the linear receivers, which have many nice properties, most notably its simplicity to implement. The estimate of the signal from user i is $\hat{x}_i = \mathbf{c}_i^T y$, where $\mathbf{c}_i$ is the linear receiver for user i. We could choose $\mathbf{c}_i = \mathbf{s}_i$ (called the matched filter receiver), which projects the received signal along the dimension of the signature sequence of user i. This works perfectly if the signatures are orthogonal (which is not necessarily true), but we can expect to do better if $\mathbf{c}_i$ exploited the fact that all of the signature sequences and received powers are known by the receiver.

The minimum mean-squared error receiver (MMSE) which as the name implies, minimizes $\mathbf{E}\left[(\hat{x}_i - x_i)^2\right]$ maximizes the signal-to-interference ratio (SIR), which is defined as

$$\text{SIR}_i = \frac{(\mathbf{c}_i^T \mathbf{s}_i)^2 p_i}{(\mathbf{c}_i^T \mathbf{c}_i)\sigma^2 + \sum_{j \neq i} (\mathbf{c}_i^T \mathbf{s}_j)^2 p_j}. \tag{35.2}$$

The SIR is just the ratio of the received power of the ith user after being projected onto its signature sequence divided by the sum of the noise power and the interference from all of the other user's signals. For the ith user, if we define

$$\mathbf{S}_i = [\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_{i-1}, \mathbf{s}_{i+1}, \cdots, \mathbf{s}_K] \tag{35.3}$$

as the signature matrix and

$$\mathbf{D}_i = \text{diag}(p_1, p_2, \cdots, p_{i-1}, p_{i+1}, \cdots, p_K) \tag{35.4}$$

as a diagonal matrix of the received powers, then the MMSE receiver is

$$\mathbf{c}_i = (\mathbf{S}_i \mathbf{D}_i \mathbf{S}_i^T + \sigma^2 \mathbf{I})^{-1} \mathbf{S}_i. \tag{35.5}$$

The SIR performance of this receiver is

$$\text{SIR}_i = p_i \mathbf{s}_i^T (\mathbf{S}_i \mathbf{D}_i \mathbf{S}_i + \sigma^2 \mathbf{I})^{-1} \mathbf{s}_i. \tag{35.6}$$

System Performance

From the equations above, we can then ask what the performance of a system will be in terms of the achievable SIR for a given user given the power and SIR of the other users in the system. The region of simultaneously achievable SIR's for N users given a certain power profile is the capacity region of the system.

In order to simplify the analysis (and to try to account for the fact that the capacity region depends nonlinearly on the signature sequences used, we assume that the users each have signature sequences that are composed of N i.i.d. random variables which are zero-mean and have variances normalized to $1/N$. This model also allows the users in the network to generate their signature sequences independently without having to have a centralized key server with a preselected set of signatures to distribute. The receiver still has to know all of the signatures of the users, however, so the basestation will still have to gain knowledge of all the users' signatures (how exactly this is accomplished is not terribly important here).

Since all the users are treated equally, we can just focus on the performance of user 1, and we can set $p_1 = 1$ for simplicity and without loss of generality. We also assume the ratio of users per degree of freedom $\alpha = K/N$ remains constant as $K, N \to \infty$, and that the distribution of the

user's powers converges to some limiting distribution F. Then SIR_1 has been shown to converge in probability to β^* from the fixed-point equation

$$\beta^* = \frac{1}{\sigma^2 + \alpha \int_0^\infty I(p, \beta^*) dF(p)} \tag{35.7}$$

where

$$I(p, x) = \frac{p}{1 + px}. \tag{35.8}$$

For a large system,

$$\text{SIR}_1 \approx \frac{1}{\sigma^2 + \frac{1}{N} \sum_{k=2}^{K} I(p_k, \text{SIR}_1)}. \tag{35.9}$$

We can also look at the capacity of a power-controlled system, in which there are J classes of users which have a target SIR of β_j, subject to an average power constraint of $\bar{p}_j$ for users in class j. Then $(\alpha_1, \ldots, \alpha_J)$ users per degree of freedom in class j can be supported when

$$\sum_{j=1}^{J} \alpha_j \frac{\beta_j}{1 + \beta_j} \leq \min_{1 \leq j \leq J} \left[1 - \frac{\beta_j \sigma^2}{\bar{p}_j} \right]. \tag{35.10}$$

If there are no power constraints, then the equation becomes

$$\sum_{j=1}^{J} \alpha_j \frac{\beta_j}{1 + \beta_j} \leq 1. \tag{35.11}$$

Looking at (35.9), we can define $I(p, \beta)$ as the effective interference of a interfering user with power p and target SIR β. Similarly, from (35.11), we can define

$$e(\beta) = \frac{\beta}{1 + \beta} \tag{35.12}$$

as the effective bandwidth of a user with a SIR requirement of β. The effective bandwidth is the fraction of a degree of freedom used by a given user, and the total effective bandwidths of a group of users must be less than or equal to 1 to be supportable by the system. Two non-obvious properties of the effective bandwidth and effective interference are

- the SIR converges to a value independent of the signature sequences
- the interfering effects of the users decouple simply as additive bandwidths

Fortunately, random matrix theory and free probability can give some insight here.

Using Marcenko-Pasteur

Starting from the class notes, we know that if we have random matrices of the form

$$B_{n \times m} = A_0 + A_1^T D A_1 \in \mathbb{R}^{m \times m} \tag{35.13}$$

where $A_0 \in \mathbb{R}^{m \times m}$ is not random, $A_1 \in \mathbb{R}^{n \times m}$ is random with i.i.d. columns, and $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix with i.i.d. elements which are independent from A_1, then given that $m/n \to y$ and the spectral density of A_0 converges at points of continuity,

$$m(z) = m_0 \left(z - y \int_{\mathbb{R}} \frac{x f(x)\, dx}{1 + x m(z)} \right). \tag{35.14}$$

Here, $f(x)$ is the probability density for the elements of D, and $m_0(z)$ and $m(z)$ are the Stieltjes Transform of the spectral densities of A_0 and B, respectively. This looks very similar to (35.6), the SIR of the MMSE receiver, with

$$\begin{aligned} A_0 &\rightarrow \sigma^2 I \\ A_1 &\rightarrow S_i \\ D &\rightarrow D_i \\ x &\rightarrow p \\ f(x)\,dx &\rightarrow dF(p) \\ y &\rightarrow \alpha. \end{aligned} \tag{35.15}$$

Then using the definition of the Stieltjes transform in (18.14), we can see that

$$m_0(z) = \frac{1}{\sigma^2 - z}, \tag{35.16}$$

and so

$$\begin{aligned} m(z) &= m_0\left(z - y\int_{\mathbb{R}} \frac{x f(x)\,dx}{1 + x m(z)}\right) \\ &= \frac{1}{\sigma^2 - \left(z - y\int_{\mathbb{R}} \frac{x f(x)\,dx}{1+x m(z)}\right)} \\ &= \frac{1}{-z + \sigma^2 + y\int_{\mathbb{R}} \frac{x f(x)\,dx}{1+x m(z)}} \\ &\downarrow \\ \beta^* &= \frac{1}{\sigma^2 + \alpha\int_{\mathbb{R}} \frac{p\,dF(p)}{1+p\beta^*}} = \frac{1}{\sigma^2 + \alpha\int_0^\infty I(p,\beta^*)\,dF(p)}. \end{aligned} \tag{35.17}$$

We evaluate $m(z)$ at $z = 0$, and we note that $p \geq 0$ so the limits of integration over $[0, \infty)$ is equivalent to the entire real line. In [462] they arrive at this result in a slightly different way, but the results are the same, except that they evaluate at $z = -\sigma^2$ due to a small equation difference. This gives an explanation of why the effective interference interpretation does not depend on the specific signature sequences.

Using Free Probability

Now we can consider two groups of interferers $\mathcal{C}_1$ with K_1 users with common power p_1 and $\mathcal{C}_2$ with K_2 users with common power p_2. We can define $\alpha_1 = K_1/N$ and $\alpha_2 = K_1/N$, then

$$SDS^T = p_1 \sum_{i\in\mathcal{C}_1} \mathbf{s}_i \mathbf{s}_i^T + p_2 \sum_{i\in\mathcal{C}_2} \mathbf{s}_i \mathbf{s}_i^T = U_1 + U_2. \tag{35.18}$$

If only one set of users is present, then the spectrum (eigenvalues density) only depends on U_i, but if both sets are present, then we can ask how the spectrum of $U_1 + U_2$ depends on U_1 and U_2. If U_1 and U_2 were free (and it turns out that they are, as most random matrices we are interested in become free as $N \to \infty$), then we know the answer through the R-transform. The R-transforms of these matrices can be explicitly computed (though it is just stated in [462] and not verified by hand) to be

$$R_{U_i}(\beta) = \alpha_i \frac{p_i}{1 + p_i\beta} = \alpha_i I(p_i, \beta), \tag{35.19}$$

and since they are free,

$$R_U(\beta) = R_{U_1}(\beta) + R_{U_2}(\beta). \tag{35.20}$$

If we let each user be its own class, then $\alpha_i = 1/N$ and

$$R_U(\beta) = \sum_i R_{U_i}(\beta) = \frac{1}{N} \sum_i I(p_i, \beta). \tag{35.21}$$

Finally, we can combine with (18.19) and evaluate at $z = -\sigma^2$ to recover (35.9). Thus the simple additivity of the effective bandwidths of the users is just a result of the additivity properties of the R-transform of free random matrices.

Chapter 36

Theory of Computation

Notes: The following paper are applications of random matrix theory to theoretical computer science:

[58] [312] [154] [153] [381] [73] [95] [238] [313]

There has long been a terrific interplay between important properties that a graph might enjoy, and properties of the eigenvalues of an associated matrix. Probably the most well studied property of a graph is the property of being an expander.

The main interest in RMT on the part of the ThCS community lies in the fact that a certain graph property (expander; connected to mixing time and sampling) of a graph depend on the second largest eigenvalue of a certain associated (Laplacian) matrix. A topic of interest in ThCS is how to produce a "good" graph (an expander) with high probability, starting from a random graph in a certain class. Random graph, random Laplacian matrix $\longrightarrow$ random eigenvalues.

There have been studies in this direction since the '80s, the most prominent of which are the papers by Ravi Boppana ([58]) and Brendan McKay ([312]). The first paper deals with an average-class analysis of eigenvalues and the graph bisection method, while the second one computes the expected eigenvalue distribution of a large regular graph.

Random regular graphs have quickly become a strong candidate for expanders. Friedman ([154]) has proved that, with positive probability, d-regular graphs for d large are expanders, by studying the probability that the second eigenvalue of a large d-regular graph is small. Friedman has recently been working on extending these results to noisy boolean networks (see [153]).

Some numerical studies ([381], [73]) done by two undergraduate students at Princeton seem to suggest that low-degree regular graphs (with d as low as 3 or 7) have even more interesting properties. These numerical studies seem to show that for large n, the second eigenvalue of d-regular graphs with $d = 3$ or $d = 7$ seems to converge to $2\sqrt{q-1}$, and thus, that with probability approaching 1, a random (large) 3-regular or 7-regular graph is Ramanujan (for a reference on Ramanujan graphs, see [95]).

The eigenvalues of regular graphs have been intensively studied; another numerical study by Jakobson et al ([238]) strongly suggests that the eigenvalue distribution of a generic k-regular graph approaches the GOE spacing distribution, as the number of vertices increases.

Finally, the spectral partitioning of a generic (non-regular) random graph has been studied by McSherry ([313]).

Chapter 37

Applications in Physics

37.1 Hatano-Nelson Models

See [90, 184, 158, 160, 183, 159, 269, 268, 270]

37.2 Growth Processes

Chapter 38

The Riemann Zeta Function

We define the famous Riemann zeta function either by a series

$$\zeta(s) = 1 + \frac{1}{2^s} + \frac{1}{3^s} + \frac{1}{4^s} + \cdots \qquad (\text{Re } s > 1) \tag{38.1}$$

or by an integral

$$\zeta(s) = \frac{1}{(1-2^{1-s})\,\Gamma(s)} \int_0^\infty \frac{t^{s-1}}{e^t+1}\,\mathrm{d}t \qquad (\text{Re } s > 0). \tag{38.2}$$

The formula

$$\zeta(1-s) = 2\cdot(2\pi)^{-s}\cos\left(\frac{\pi s}{2}\right)\Gamma(s)\,\zeta(s) \tag{38.3}$$

allows us to extend zeta to the entire complex plane. From (38.3) and (38.2) respectively we see the

$$\begin{array}{rrcl} \text{trivial roots:} & \zeta(-2m) & = & 0 \qquad m = 1, 2, \ldots \\ \text{and only pole:} & \zeta(1) & = & \infty\,. \end{array}$$

At the positive even integers zeta is related to a Bernoulli integer and a power of π. For the odd integers, little is known other than that $\zeta(3)$ is irrational.

It is known that all non-trivial zeros s have real part in the unit interval $[0, 1]$. The famous Riemann hypothesis, considered one of the most important unsolved problems in mathematics, is that $\zeta(s) = 0$ only if Re $s = 1/2$ or $s = -2m$.

Many zeros of Riemann zeta along the critical line have explicitly been computed by Odlyzko. The tables are available to the public at the website:

```
http://www.dtc.umn.edu/~odlyzko/zeta_tables/index.html
```

It is of interest to graph zeta in many places including along the critical line:

```
y=0:0.05:60
plot(y,abs(zeta(.5+y*i)))
```

It might be hoped that somehow the zeros correspond to a transformed Hermitian operator. To be precise if S is a Hermitian matrix, certainly $i * S + 1/2$ has all of its eigenvalues on the critical line. If the zeros of ζ were somehow associated with the eigenvalues of $i * S + 1/2$, even in some limiting sense, the Riemann hypothesis would be proved!

Chapter 39

Random Growth and Random Matrices

[This section was originally written by Alex Skorokhod]

In this paper we will consider define a class of local random growth model and consider an instance of the model for which exact results were obtained.

Think of a growing shape Ω_t which is a union of unit squares centered at points of $\mathbf{Z}^2$. At every step we increase t by 1 and for every square that neighbors our shape we add it with probability $p = 1 - q$. We $t = 0$ we start with a single square centered at $(0, 0)$. It turns out that Ω_t grows linearly with time; hence we can define an asymptotic shape $A = \lim_{t \to \infty} \Omega_t / t$.

There is another way to look at this problem: we are interested in $T(M, N)$—time when a cell centered at (M, N) is added to the growing shape. Then there is a connection between Ω_t and $T(M, N)$:

$$\Omega_t = \left\{ (M, N) \in \mathbf{Z}^2 \;\middle|\; T(M, N) \leq t \right\}$$

To get a formula for $T(M, N)$ imagine that a square centered at (i, j) has a random geometrically distributed variable $w(i, j)$ such that $P[w(i, j) = s] = (1 - q)q^s$ and all $w(i, j)$ are independent. Random Variable $w(i, j)$ is a time after which a square is added once it neighbors the growing shape. Then the $T(M, N)$ can be written as

$$T(M, N) = \min_{\pi} \sum_{(i,j) \in \pi} 1 + w(i, j)$$

where is a path from $(0, 0)$ to (M, N).

Consider a slight modification of the above problem. Allow a shape to grow only in up/right direction (hence restricting it to the 1st quadrant) and require a square to have it's down/left neighbors to be in the shape in order to be consider for adding to the shape. So the growth appears only in "corners" of Ω_t. (A reader might think of some form of a randomized Convey's Game of Life).

If we define $G(M, N)$ as a time at which square centered at (M, N) gets added in this model minus the smallest time that it takes to reach (M, N), we get an easy recurrence relation:

$$G(M, N) = w(M, N) + \max(G(M - 1, N), G(M, N - 1))$$

since (M, N) will get added $1 + w(M, N)$ time passed the moment both $(M-1, N)$ and $(M, N-1)$ are added. Hence we get the following alternative definition for G:

$$G(M, N) = \max_{\pi} \sum_{(i,j)\in\pi} w(i, j)$$

We are interested in distribution for $G(M, N)$. Think of G as a generalize sum of i.i.d random variables and a central limit theorem accompanying them: don't restrict w to be geometrically distributed and set $N = 1$ to get that $G(M, 1)$ for $M \to \infty$ approaches normal distribution.

The exact distribution for a general random variable w was obtained for any fixed N and $M \to \infty$.

However for both $M \to \infty$ and $N \to \infty$, only the special case of geometrically distributed w yielded results. This is the case that will be considered in the rest of the paper—geometrically distributed times, model restricted to the first quadrant and up/right growth.

Just as before we can define a shape

$$\Omega_t = \left\{ (M, N) \in (\mathbf{Z}^+)^2 \;\middle|\; G(M, N) + M + N - 1 \leq t \right\}$$

where $M + N - 1$ is the length of any up/right path from $(1, 1)$ to (M, N) and this term takes care of the fact that we wait for 1 unit of time before considering particle to be added.

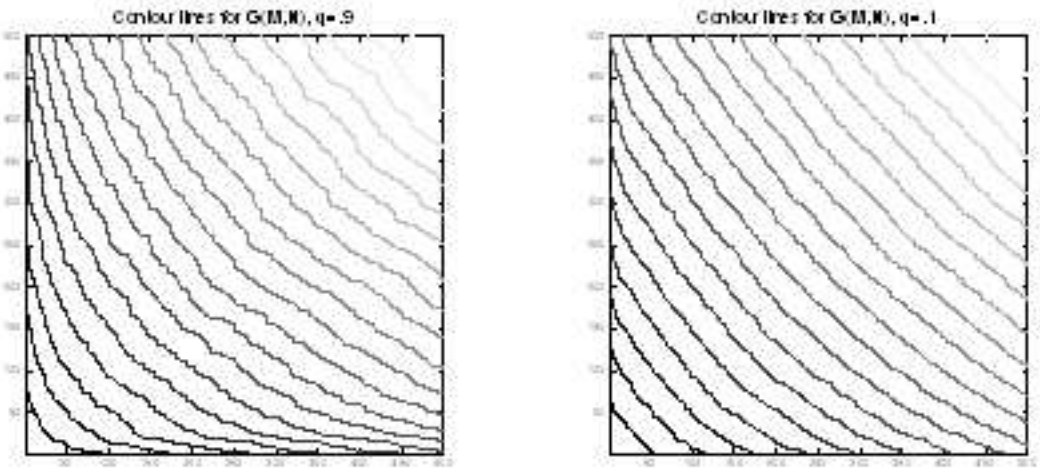


Figure 39.1: The above diagrams contain contour lines (fronts at different times) of Ω_t for different values of q (.1 and .9).

The main result about growth and fluctuation of $G(M, N)$ is the following theorem due to Johansson [247, 248]:

Theorem 1

For each $q \in (0, 1)$, $\gamma \geq 1$ and $s \in R$,

$$\lim_{N\to\infty} P\left[\frac{G\left([\gamma N]\right) - N\omega(\gamma, q)}{\sigma\left(\gamma, q\right) N^{1/3}} \leq s \right] = F_2(s)$$

where

$$\omega(\gamma, q) = \frac{\left(1+\sqrt{q\gamma}\right)^2}{1-q} - 1$$

and

$$\sigma(\gamma, q) = \frac{q^{1/6}\gamma^{1/6}}{1-q}\left(\sqrt{\gamma}+\sqrt{q}\right)^{2/3}\left(1+\sqrt{q\gamma}\right)^{2/3}$$

View γ as a parameter which give the slope of the radial line of growth that we are considering. All the numerical results presented later in this paper will consider for simplicity case $\gamma = 1$.

The variable q describes the geometrical random variable that defines growth. Functions ω and σ simply describe mean and standard deviation of G.

Distribution F2 is defined by a Fredholm determinant

$$F_2(t) = \det(I - A)_{L^2(t,\infty)}$$

where $A(x)$ is Airy-kernel defined in terms of Airy function $Ai(x)$ as follows:

$$A(x, y) = \int_0^\infty Ai(x+t)Ai(y+t)dt$$

Using ω we can easily determine asymptotic shape for a corner growth model: A is a region in the first quadrant bounded by a curve $x + 2\sqrt{qxy} + y = 1 - q$. In the next figure you can observe bounds for different q's (compare to the contours for $q = .1$ and $q - .9$ given above).

In the next figure a histogram for properly scaled $G(200, 200)$ for 10000 repetitions and $q = .9$ is drawn against $f_2 = F_2'$ (MATLAB code is given in the appendix):

39.1 Analysis of corner growth model

Consider a bit more general model then defined above: let $w(i, j)$ be a geometrically distributed variable with parameter $x_i y_j$ where x_i, y_i are numbers in $[0, 1)$. This generalization allows us to restrict our attention to $G(N, N)$ since in order to find a distribution for $G(M, N)$ $(M < N)$ for identical $w(i, j)$, we just need to set $y_i = 0, M < i \leq N, y_i = \sqrt{q}, 1 \leq i \leq M$ and $x_i = \sqrt{q}, 1 \leq i \leq N$.

The main combinatorial fact used in the analysis of corner growth model is Robinson-Schensted-Knuth (RSK) correspondence between non-negative integer matrices $M \times N$ and pairs (T, S) of semi-standard Young Tableaux of the same shape λ, where T has elements from $[N]$ and S has elements from $[M]$. Semi-standard Young tableaux is defined just a standard Young Tableaux with one exception: it can be filled in with repeated numbers and this numbers have to increase left-to-right and strictly increase up-down.

The important fact about this bijection is that $G(M, N)$ is equal to λ_1 and $\sum_j w(i, j) = m_i(S), \sum_i w(i, j) = m_j(T)$ where $m_i(T)$ is a number of times number i is present in T.

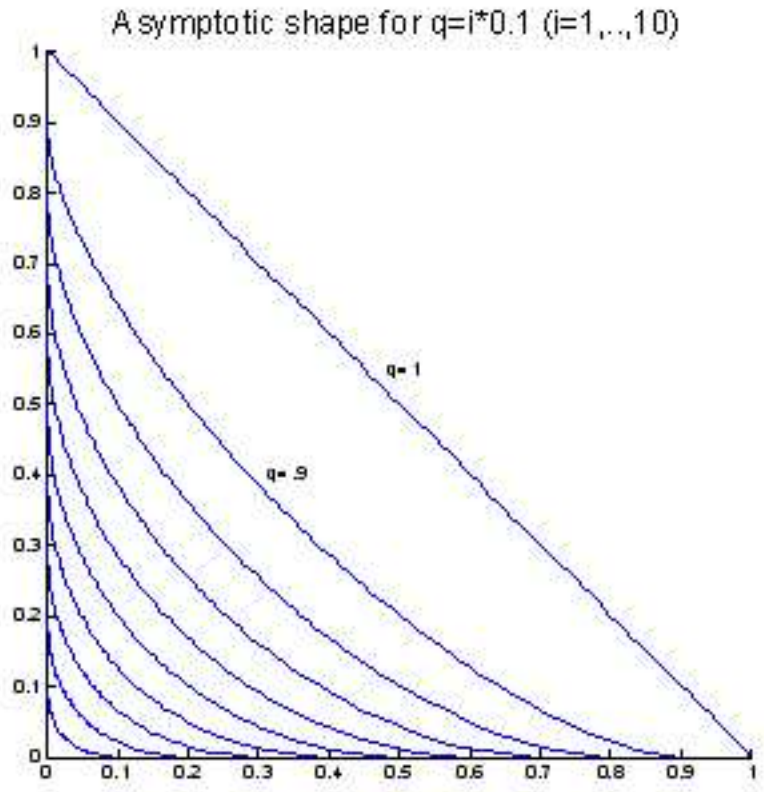


Figure 39.2:

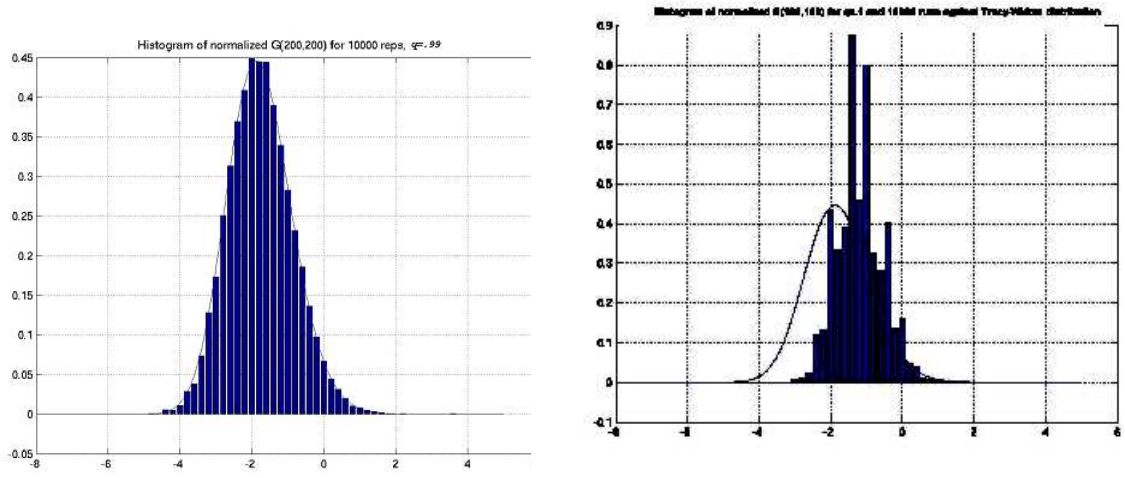


Figure 39.3: On the side note, for simulations it is better to use higher values of q. For small q convergence of $G(N,N)$ to f_2 cannot be seen for N of the order 100

With a generalized model and RSK correspondence in mind we can rewrite $G(N,N)$ as follows:

$$P\left[G(N,N)\leq n\right]=\sum_{W;G(N,N)\leq n}P\left[W\right]=$$
$$\prod_{i,j=1}^{N}(1-x_iy_j)\sum_{W;G(N,N)\leq n}\prod_{i=1}^{N}x_i^{\sum_j w(i,j)}\prod_{j=1}^{N}y_j^{\sum_i w(i,j)}=$$
$$\prod_{i,j=1}^{N}(1-x_iy_j)\sum_{\lambda:\lambda_1\leq n}\left(\sum_{S;sh(S)=\lambda}\prod_{i=1}^{N}x_i^{m_i(S)}\right)\left(\sum_{T;sh(T)=\lambda}\prod_{j=1}^{N}y_j^{m_j(T)}\right)$$

where the first equality follows from the fact that $G(N,N)$ is uniquely defined by a $N\times N$ matrix $W=(w(i,j))$; the second from the fact that $P\left[w(i,j)=s\right]=(1-x_iy_j)(x_iy_j)^s$ and the third follows from RSK correspondence and it's properties given above.

Using the following representation for Schur polynomial:

$$s_\lambda\left(x_1,\ldots,x_N\right)=\sum_{S;sh(S)=\lambda}\prod_{i=1}^{N}x_i^{m_i(S)}$$

and the definition of a Schur's measure on all partitions with at most N non-zero parts (introduced in [341]):

$$P_s(\lambda)=\prod_{i,j=1}^{N}(1-x_iy_j)s_\lambda(x)s_\lambda(y)$$

and the derivation above we get:

$$P\left[G(N,N)\leq n\right]=\sum_{\lambda;\lambda_1\leq n}P_s(\lambda)$$

The rest of the solution given in [247] is analysis of the Schur's measure.

There is another connection between random growth model introduced here and already known results that were linked to random matrix theory. If we set $q=\alpha/N^2$ then

$$G(N,N)\xrightarrow[N\to\infty]{Distr}L(\alpha)$$

where $L(\alpha)$ has the same distribution as the length of the longest increasing subsequence in a random permutation from S_N with uniform distribution [247]. It was already shown that when properly scaled, $L(\alpha)$ is distributed as F_2.

Not only F_2 Tracy-Widom distribution appears in random growth models but F_1 pops up in random growth model [244]. Define

$$G_{pl}(N)=\max_{|k|<N}G(N+k,N-k).$$

One can interpret it as a point-to-line walk—from $(1,1)$ to $x+y=2N$. It turns out that $G_{pl}(N)$ is distributed as F_1 as $N\to\infty$:

$$\lim_{N\to\infty}P\left[\frac{G_{pl}(N)-N\mu}{\sigma N^{1/3}}\leq s\right]=F_1(s)$$

39.2 Appendix

This script plots a front of asymptotic shape of corner-growth model for 10 q's.

```
figure(1)
hold
for ii = 1:11
    q = (ii-1)/10;
    x =0:p/100:1-q;
    y =(sqrt((1-q)*(1-x)) - sqrt(q*x)).^2;
    plot(y,x)
end
axis xy image
title('Asymptotic shape for q=i*0.1 (i=1,..,10)','FontSize',16);


function G = G(N,M,q)
%// Computes matrix $G(N,M)$ for a given $q$.
G = floor(log(rand(N,M))/log(q));
G(1:N,1) = cumsum(G(1:N,1)) + (1:N)';
G(1,1:M) = cumsum(G(1,1:M)) + (1:M);
for ii = 2:N
    for jj = 2:M
        G(ii,jj) = G(ii,jj) + max(G(ii,jj-1), G(ii-1,jj));
    end
end
```

Code 39.1

The following script draws a histogram of $G(N,N)$ against f_2.

```
num_trials = 10000;
g = 1;
q = .9;
N = 100;
M = round(N*g);

B = zeros(1,num_trials);
for ii = 1:num_trials
    G = G(M,N,q);
    B(ii) = G(M,N);
end
C = (B - N*om(g,q))/(sigma(g,q)*N^(1/3));
d = .2;
[n,x] = hist(C, -6:d:4);
bar(x,n/(d*num_trials));
hold
```

Code 39.2

This part was kindly provided by Per-Olof Persson

```
deq=inline('[y(2); s*y(1)+2*y(1)^3; y(4); y(1)^2; y(1)]','s','y');
opts=odeset('reltol',1e-13,'abstol',1e-15);

t0=5;
tn=-8;
tspan=linspace(t0,tn,1000);
y0=[airy(t0);airy(1,t0);0;airy(t0)^2;0];

[s,y]=ode45(deq,tspan,y0,opts);

F2=exp(-y(:,3));
f2=gradient(F2,s);
plot(s,f2)
grid on


%// helper functions sigma.m and om,m which describe mean and
%// standard deviation of G(M,N)
function om = om(g,q)
%//Computes G(qN,N)/N for N->inf
om = ((1+sqrt(g*q))^2)/(1-q) - 1;
```

Code 39.3

Chapter 40

Student Papers

40.1 On the Multiplicity of an Eigenvalue of a Random Tree

[This section was originally written by Brian D. Sutton]

Abstract

Take an undirected labeled tree on n vertices uniformly at random. We consider $\mathbf{E}[m_T(\lambda)]$, the expected value of the multiplicity of λ in the adjacency matrix of T. We give a lower bound on $\mathbf{E}[m_T(\lambda)]$, expressed as a generating function.

Take an undirected labeled tree on n vertices uniformly at random. Typically, the adjacency matrix of such a tree has several high multiplicity eigenvalues. See Figure 1 for a histogram. As we will see, the high multiplicity eigenvalues are related to the presence of certain subtrees. For example, the eigenvalue 0 arises with high multiplicity because it is the eigenvalue of (the adjacency matrix of) the tree on one vertex, and the eigenvalue 1 arises with high multiplicity because it is an eigenvalue of the tree on two vertices.

[44] compute the expected multiplicity of the eigenvalue 0. (As in this paper, the authors consider adjacency matrices of undirected labeled trees from the uniform distribution.) The expected rank deficiencies for $n = 1, 2, 3, 4, 5$ are $1, 0, 1, 1/2, 135/125$, respectively. Asymptotically, the expected rank deficiency is

$$\mathbf{E}[m_T(0)] = (2x_* - 1)n + \frac{x_*^2(x_* + 2)}{(x_* + 1)^3} + O(1/n),$$

for $x_* = 0.56714\ldots$. ($m_T(\lambda)$ denotes the multiplicity of λ as an eigenvalue of the adjacency matrix of T.)

The present paper extends the result of [44] to multiplicities of other eigenvalues. Suppose that a tree T' having λ in its spectrum is explicitly known. By counting certain occurences of T' as an induced subtree, we derive a lower bound on $\mathbf{E}[m_T(\lambda)]$, expressed as a generating function.

40.1.1 Overview

The relationship between eigenvalues and induced subtrees hinges on the eigenvalue *interlacing inequalities* for a symmetric matrix. Suppose A is real symmetric with eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$, and let $A[i]$ denote the principal submatrix of A obtained by deleting row and column i. Then the

eigenvalues of $A[i]$ lie "in between" the eigenvalues of A on the real line. To be more precise, if $\mu_1 \leq \cdots \leq \mu_{n-1}$ are the eigenvalues of $A[i]$, then $\lambda_1 \leq \mu_1 \leq \lambda_2 \leq \cdots \leq \mu_{n-1} \leq \lambda_n$.

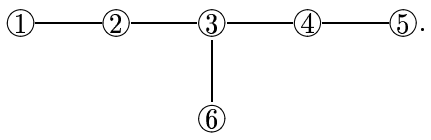
Hence, if we take a principal submatrix by deleting a single row and column, the multiplicity of any particular eigenvalue can change by at most 1. (In fact, all three possibilities—up, down, or the same—are possible.) We will denote the spectrum of the adjacency matrix of a graph G by $\sigma(G)$, and the multiplicity of the eigenvalue λ in this spectrum by $m_G(\lambda)$. (This "multiplicity" may be 0.) Then we have, for any vertex i of a tree T, $m_T(\lambda) \geq m_{(T \setminus i)}(\lambda) - 1$, or for any collection of vertices $\{i_1, \ldots, i_k\}$,

$$m_T(\lambda) \geq m_{(T \setminus \{i_1, \ldots, i_k\})}(\lambda) - k. \tag{40.1}$$

Of course, the trick is to make a clever choice for $\{i_1, \ldots, i_k\}$.

The intuition behind high eigenvalue multiplicities should begin to take shape. Suppose that we delete several vertices $i_1, \ldots, i_k$ from T. If a particular tree T' appears in the forest $T \setminus \{i_1, \ldots, i_k\}$ sufficiently often, then (40.1) will imply that the eigenvalues of T' are also eigenvalues of T, possibly with high multiplicity.

As an example, consider the tree



Removing vertex 3 leaves

which has spectrum -1, -1, 0, 1, 1. Then (40.1) implies $m_T(1) \geq 1$.

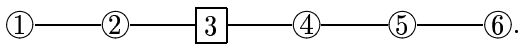
When T' is a tree of the forest $T \setminus i$, we say that T' is a *branch* of T. In this case, we call i the *bud* of T'. Our program for estimating $m_T(\lambda)$ rests on the following strategy: First, identify branches having a given eigenvalue. Second, bound $m_T(\lambda)$ by (40.1). Using this method for fixed trees, we will move on to random trees, estimating $\mathbf{E}[m_T(\lambda)]$.

40.1.2 Estimating $m_T(\lambda)$ for a non-random tree T

We begin with an extended example, estimating the multiplicity of the eigenvalue 1. Observe that the 2-path

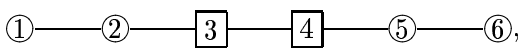
$$P_2 = \text{①——②}$$

has spectrum -1, 1. Given a tree T, we would like to remove relatively few vertices, leaving a forest which contains several copies of P_2. One way to identify vertices to delete is to start at the "outside" of the tree (at the pendant vertices) and look for places where P_2 occurs as a branch. For example, for the path on 6 vertices, just start at the left,

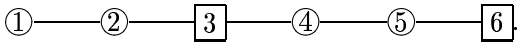


Then repeat the process recursively on each remaining subtree.

Unfortunately, this procedure does not produce a unique "marking." For example, at the next stage, we could either mark vertex 4 for deletion,

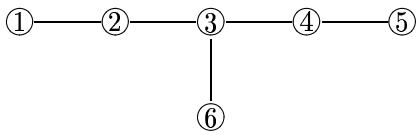


or vertex 6,

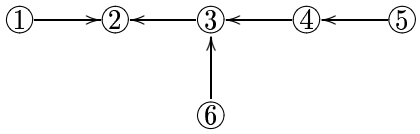


(The presence of multiple markings is not a problem for a single tree, but when counting trees, we would like to have a unique decomposition.)

In order to define an algorithm for (uniquely) marking trees, we introduce rooted trees. Let R be a rooted tree, i.e. a labeled tree with an identified "root" vertex. To identify the root, we draw all edges of R "toward the root." For example, if we choose 2 to be the root of

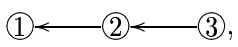


then we will draw

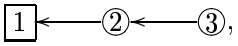


We still think of the edges of the rooted tree as undirected. (We need a symmetric adjacency matrix!) The arrows are just a convenient way to pinpoint the root.

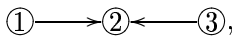
We mark vertices for deletion as before, with the restriction that the edge from the branch P_2 to its neighbor must point toward the root. For example, in the rooted tree



the root may be marked for deletion,



but in the tree



no vertex may be marked for deletion, because both of the pendant vertices have in-degree 0.

Here is the final marking algorithm.

Algorithm (MARK).
Input:

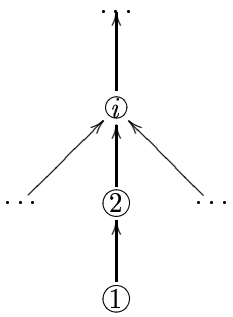
- A rooted tree $T = (V, E)$.
- A tree T' such that $\lambda \in \sigma(T')$.

Output: A "marking" $V \to \{\texttt{o}, *\}$.

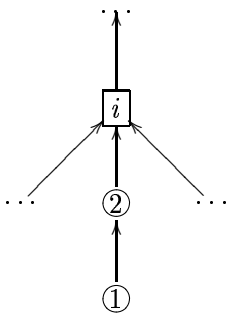
Steps:

- Choose a branch of T with bud i that is isomorphic to T', subject to the condition that the arrows incident to the branch point toward i. For example, if T' is the 2-path, then the

branch will look vaguely like



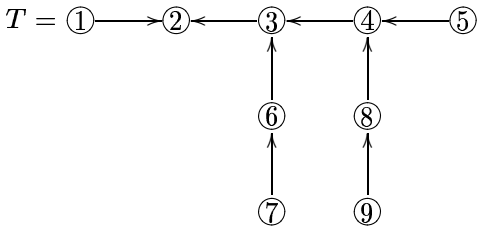
- Mark the neighbor i (if such a branch exists), e.g.



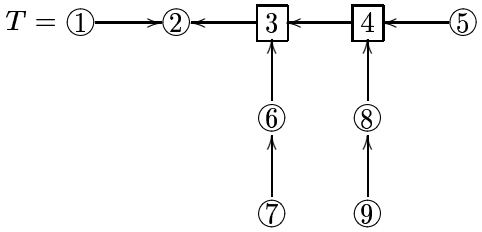
- Recurse on each component of $T \setminus i$.

Throughout the rest of the paper, T' denotes a fixed tree with λ in its spectrum. Our examples will deal with $\lambda = 1$, $T' = P_2$.

Example 40.1.1. The first stage in estimating $m_T(1)$ for



is to apply MARK to input (T, P_2). The result is



Lemma 40.1.2. *Algorithm 40.1.2 produces a unique marking, regardless of which nondeterministic choices are made.*

Proof. A □t any stage of the algorithm, if i and j are both candidates for marking and i is marked, then j may be marked at the next iteration.

By the interlacing inequalities (40.1),

$$\begin{aligned} m_T(\lambda) &\geq m_{(T \smallsetminus \{i_1,\ldots,i_k\})}(\lambda) - k \\ &\geq \#\{\text{components of } T \smallsetminus \{i_1,\ldots,i_k\} \text{ isomorphic to } T'\} - k, \end{aligned}$$

where $i_1, \ldots, i_k$ are the vertices marked by MARK. (In fact, this inequality holds for any set of vertices.) It is considerably less obvious that this estimate is the best possible (based only on looking at branches isomorphic to T').

Branch removal is the process of removing a branch from T that is isomorphic to T', along with its bud i.

Theorem 40.1.3 ([249]). *$m_T(\lambda)$ is invariant under branch removal.*

Corollary 40.1.4.

$$m_T(\lambda) = m_{(T \smallsetminus \{i_1,\ldots,i_k\})}(\lambda) - k,$$

where $i_1, \ldots, i_k$ are the vertices marked by $\mathrm{MARK}(T, T')$ *for some T' such that $\lambda \in \sigma(T')$.*

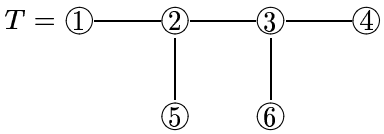
Proof. A $\square$s MARK is executed, each marked vertex i_j is incident to some branch B_j that is isomorphic to T'. Each of $B_1, \ldots, B_k$ is a connected component of $T \smallsetminus \{i_1, \ldots, i_k\}$. Let F denote the forest obtained by deleting $i_1, \ldots, i_k$ and $B_1, \ldots, B_k$ from T. By the theorem, $m_T(\lambda) = m_F(\lambda) = m_{(T \smallsetminus \{i_1,\ldots,i_k\})}(\lambda) - k$.

Define $m_T(T')$ to be the quantity $\#\{\text{components of } T \smallsetminus \{i_1, \ldots, i_k\} \text{ isomorphic to } T'\} - k$, in which $i_1, \ldots, i_k$ are the vertices marked by MARK. The corollary implies that the only source of error in the approximation

$$m_T(\lambda) = m_{(T \smallsetminus \{i_1,\ldots,i_k\})}(\lambda) - k \geq m_T(T') \tag{40.2}$$

is the fact that we approximate the multiplicity of λ in $\sigma(T \smallsetminus \{i_1, \ldots, i_k\})$ by the number of connected components that are isomorphic to T'.

Example 40.1.5. The tree



has $m_T(1) = 1$, but our lower bound only provides $m_T(1) \geq 0$ if T' is taken to be the 2-path. (No vertices are marked, regardless of the root.)

We can improve the lower bound by modifying MARK to look for more "irreducible" subtrees with λ in their spectra, e.g. the tree in the previous example.

There is perhaps one more troubling aspect of the marking procedure. Namely, in $T \smallsetminus \{i_1, \ldots, i_k\}$, the connected component containing the root may have a branch isomorphic to T' (which necessarily contains the root). This is because of the restriction on the direction of arrows. It may seem that our marking procedure is incomplete since the vertex i adjacent to this branch is not marked. It is easy to check that the estimate given by (40.2) is the same whether i is marked or not.

40.1.3 Tree Decomposition

The following construction generates every rooted tree in a unique way.

1. Choose a root r from $\{1, \ldots, n\}$.
2. Take a forest of rooted trees on $\{1, \ldots, n\} \setminus \{r\}$.
3. Draw an edge between r and every root in the forest.

We will derive a recurrence for $m_T(T')$ based on this decomposition. We treat three types of trees.

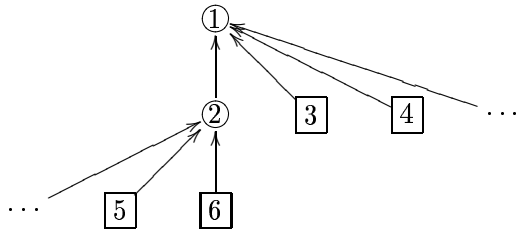
Type 1 Suppose MARK(T, T') marks the root.

Then by the equality in (40.2), $m_T(T') = m_{(T \setminus r)}(T') - 1$.

Type 2 Run MARK(T, T') and delete the marked vertices. Suppose that the connected component containing the root is isomorphic to T'.

Then $m_T(T') = m_{(T \setminus r)}(T') + 1$.

For example, in the case $T' = P_2$, either T is a 2-path, or T looks like



(The labels are unimportant. The point is that every neighbor of the 2-path is marked.)

Type 3 Suppose that T is neither Type 1 nor Type 2.

Then $m_T(T') = m_{(T \setminus r)}(T')$.

40.1.4 Random Trees

We want to calculate a lower bound on the expected multiplicity of λ over the uniform distribution on labeled trees,

$$\mathbf{E}[m_T(\lambda)] = \frac{\sum_{T \in T_n} m_T(\lambda)}{\#T_n},$$

in which T_n is the set of labeled trees on n vertices. Notice that the expected multiplicity is unchanged if we instead work with the uniform distribution on the set RT_n of rooted labeled trees,

$$\mathbf{E}[m_T(\lambda)] = \frac{\sum_{T \in RT_n} m_T(\lambda)}{\#RT_n}.$$

The equality holds because distinguishing a root vertex does not affect the spectrum, and because exactly n rooted trees correspond to any unrooted tree. (Note that using labeled trees as opposed to unlabeled trees is important here.)

Fix a tree T' with $\lambda \in \sigma(T')$. Let

$$a_{n,k} = \#\{T \in RT_n : m_T(T') = k\},$$
$$f_{n,k} = \#\{F \in RF_n : m_F(T') = k\},$$

in which RF_n denotes the set of forests of labeled trees on n vertices, and $m_F(T')$ is defined to be the sum of multiplicities $\sum_{T \in F} m_T(T')$ over all trees T in the forest F. (We define $a_{0,0} = 0$, $f_{0,0} = 1$.)

Further, let

$$b_{n,k} = \#\{T \in RT_n : m_T(T') = k, T \text{ is Type 1}\},$$
$$c_{n,k} = \#\{T \in RT_n : m_T(T') = k, T \text{ is Type 2}\},$$
$$d_{n,k} = \#\{T \in RT_n : m_T(T') = k, T \text{ is Type 3}\},$$

By the preceding section, we see that

$$\begin{aligned} nf_{n,k} &= \#\{T \in RT_n : m_{(T \setminus r)}(T') = k, r = \text{root of } T\} \\ &= b_{n,k-1} + c_{n,k+1} + d_{n,k} \end{aligned}$$

and

$$\begin{aligned} a_{n,k} &= b_{n,k} + c_{n,k} + d_{n,k} \\ &= nf_{n,k} - b_{n,k-1} - c_{n,k+1} + b_{n,k} + c_{n,k} \end{aligned}$$

At this point, exponential generating functions become useful. Let

$$A(x,y) = \sum_{n \geq 0} \sum_{k \geq 0} a_{n,k} \frac{x^n y^k}{n!},$$

and define $B(x,y)$, $C(x,y)$, $D(x,y)$, and $F(x,y)$ similarly for the other sequences.

These sorts of generating functions can be used to count structures on n elements of "weight" k. Notice the effect of the operations $+$, $*$, and $\exp(\cdot)$.

$G + H$ counts the following construction: For a given n, place either a structure of type G or a structure of type H on $\{1, \ldots, n\}$. Use the G-weighting or the H-weighting as appropriate.

$G * H$ counts the following construction: For a given n, choose an m, and partition the elements $\{1, \ldots, n\}$ into (distinguishable) sets of size $m, n-m$. Then place a structure of type G on the set of size m, and a structure of type H on the set of size $n-m$. The weight of the combined structure is the sum of the weights of the component structures.

$\exp(G) = \sum_{p \geq 0} G^p/p!$ counts the following construction: For a given n, partition $\{1, \ldots, n\}$ into one or more nonempty sets. Then place a structure of type G on each set. The weight of the combined structure is the sum of the weights of the component structures. Proof: Use the product and sum rules. First choose the number of parts p in the partition. (This is the sum rule.) Then partition $\{1, \ldots, n\}$ into an ordered partition and place structures on each partition. (This is the product rule iterated p times.) Then divide by $p!$, so that the order of the parts is ignored.

Thus,

$$A(x,y) = xF(x,y) - yB(x,y) - y^{-1}C(x,y) + B(x,y) + C(x,y), \tag{40.3}$$
$$F(x,y) = \exp(A(x,y)). \tag{40.4}$$

It remains to understand $B(x,y)$ and $C(x,y)$.

Observe that any tree of Type 1 must have one or more subtrees of Type 2 adjacent to the root, and that these subtrees are naturally distinguished from subtrees of the other types. Also, the "weight" $m_T(T')$ in this case is one less than the weight of the forest of subtrees $m_{(T \setminus r)}(T')$. Therefore,

$$B(x,y) = xy^{-1}(\exp(C(x,y)) - 1)(\exp(B(x,y) + D(x,y))) \tag{40.5}$$
$$= xy^{-1}(\exp(C(x,y)) - 1)(\exp(A(x,y) - C(x,y))). \tag{40.6}$$

Next, notice that any tree of Type 2 may be constructed by replacing each vertex of T' by a tree in which every neighbor of the root is marked. Also, the "weight" $m_T(T')$ in this case is one more than the weight of the forest of subtrees $m_{T \setminus r}(T')$. Therefore, if $G(x)$ is the exponential generating function for T', e.g. $G(x) = 2\frac{x^2}{2!}$ if T' is the 2-path, then

$$C(x,y) = yG(x\exp(B(x,y))). \tag{40.7}$$

By Stanley Volume I, $A(x,y)$, $B(x,y)$, $C(x,y)$, and $F(x,y)$ are defined by equations (40.3), (40.6), (40.7), and (40.4) under a limiting procedure. For example, if T' is the 2-path, then start with the "base cases"

$$\begin{aligned} A_0(x,y) &= x, \\ F_0(x,y) &= \exp(x), \\ B_0(x,y) &= 6\frac{x^3}{3!}, \\ C_0(x,y) &= 2\frac{x^2 y}{2!}, \end{aligned}$$

and inductively define

$$\begin{aligned} A_{m+1}(x,y) &= xF_{m+1}(x,y) - yB_{m+1}(x,y) - y^{-1}C_{m+1}(x,y) + B_{m+1}(x,y) + C_{m+1}(x,y), \\ F_{m+1}(x,y) &= \exp(A_m(x,y)), \\ B_{m+1}(x,y) &= xy^{-1}(\exp(C_m(x,y)) - 1)(\exp(A_m(x,y) - C_m(x,y))), \\ C_{m+1}(x,y) &= yG(x\exp(B_m(x,y))). \end{aligned}$$

Then $A_m(x,y)$ converges to $A(x,y)$ in the sense that for sufficiently large M, $[x^n y^k/n!]A_m(x,y) = [x^n y^k/n!]A(x,y)$ for all $m \geq M$. (Using induction, check that the coefficients of $[x^n y^k/n!]$ in $A_m(x,y)$ are correct for all $n \leq m$ and all k, in $F_m(x,y)$ for all $n \leq m-1$ and all k, in $B_m(x,y)$ for all $n \leq m$ and all k, and in $C_m(x,y)$ for all $n \leq m$ and all k.)

Theorem 40.1.6. *Let T be a rooted labeled tree on n vertices from the uniform distribution, and let $\lambda \in \sigma(T')$. Then*

$$\mathbf{E}[m_T(\lambda)] \geq \mathbf{E}[m_T(T')] = \frac{1}{n^{n-1}} \left[\frac{x^n}{n!}\right] \frac{\partial}{\partial y} A(x,y)\bigg|_{y=1},$$

in which $[x^n/n!]$ denotes the coefficient of $x^n/n!$ in the formal power series, and $A(x,y)$ is defined

by

$$
\begin{aligned}
A(x,y) &= xF(x,y) - yB(x,y) - y^{-1}C(x,y) + B(x,y) + C(x,y),\\
F(x,y) &= \exp(A(x,y)),\\
B(x,y) &= xy^{-1}(\exp(C(x,y)) - 1)(\exp(A(x,y) - C(x,y))),\\
C(x,y) &= yG(x\exp(B(x,y))),
\end{aligned}
$$

where $G(x) = \#\{$*automorphisms of* $T'\}(x^n/n!)$*, and appropriate "base cases."*

40.2 Power Law Graphs

[This section was originally written by Vijay Divi]

From biological systems to the internet topology, many evolutionary networks can be modeled by power law graphs. These graphs have a power law distribution on the degree of their vertices and represent systems where there are few central hubs yand many smaller degree nodes. The following sections will overview the definiton of power law graphs, give a model for creating them, and present an analysis of the eigenvectors of these graphs. Although the distribution of eigenvalues is unknown, we will present an analysis of the central part of the distribution and show that the highest eigenvalues also have a power law distribution.

40.2.1 Definiton

A power law graph is a graph in which the number of vertices of degree d is proportional to $d^{-\gamma}$ for some positive constant γ. Thus, there are very few nodes of high degree and many nodes of low degree. From the graph, we are able to obtain an 0-1 adjacency matrix, A, which is able to fully represent the links in our system. Because we are dealing with undirected graph, the matrix is symmetric.

40.2.2 Generation

In order for us to study power law graphs, it is important to have a good method for generating graphs whose degrees follow the desired distribution. The most common method is described below [38]:

- Begin with a small number, m_0, of disjoint vertices
- For $i = 1, ..., t$
  - Add a new vertex
  - Connect the new vertex to m distinct vertices, where the probability that the new vertex is connected to existing vertex v is $d_v / \sum_j d_j$ (d_j is the degree of the vertex j)

The graph generated has $m_0 + t$ vertices and mt edges. This model of generation captures the evolutionary characteristic of power law networks. Each new vertex randomly connects to old vertices with preferential treatment towards those vertices that have higher degrees (the hubs). Its distribution matches the power law more closely than a method where connections are chosen with uniform probability.

As Barabási and Albert show, by treating this as a continuous time problem we can see the rate that which vertex v grows as $\partial d_v / \partial t = d_v / 2t$. This yields that $d_v(t) = m(t/t_v)^{0.5}$ where $d_v(t)$ is the degree of vertex v at time t and t_v is the time that v is added to the graph. From this, they are able to obtain that the probability density of degrees for this method of graph creation is: $P(d) = 2m^2/d^3$. Thus, the degree distribution follows a power law.

The following code implements this power law graph generation method within MATLAB:

```
% powerlaw.m
% This function outputs the adjacency matrix for
% powerlaw graph with the following parameters:
%
% m0 = starting number of vertices
% m = number of vertices each new vertex attaches to
% t = number of new vertexes added

function [Amat,degreesA,eA,momA] = powerlaw(m0,m,t)

Amat = zeros(m0+t,m0+t);
for i=1:t
    sizeA = m0+i-1;
    connA = sum(Amat,1);
    connA = connA(1:sizeA)+1;
    allconn = sum(connA);
    for j=1:sizeA
        sumConn(j) = sum(connA(1:j));
    end
    for k = 1:m
        neednew = 1;
        while neednew == 1
            oldvertpos = floor(rand(1)*allconn)+1;

            posvert = find(oldvertpos <= sumConn);
            oldvert = min(posvert);

            if (Amat(m0+i,oldvert) == 0)
                Amat(m0+i,oldvert) = 1;
                Amat(oldvert,m0+i) = 1;
                neednew = 0;
            end
        end
    end
end

degreesA = sum(Amat,1);
eA = eig(Amat);
for j=1:4
   momTrace(j) = 1/length(Amat) * trace(Amat^j);
end
```

Code 40.1

Within all of the simulations, the input paramenters to the graph generator are $m_0 = 5$, $m = 5$, $t = 1000$. To verify these are power law, Figure **??** show the histogram of degrees and a log-log

plot of degree distribution of a single matrix. As one expects, the first figure matches very clearly to a power law of $\gamma = 2.9$, and the second figure is close to being linear.

Code to generate degree plots:

```
% plotdegrees.m
% First plot histograms the degrees of graph
% Second plot does this on a log-log scale

subplot(1,2,1);
b=2.9;
vals = 0:1:max(degreesA);
[mh,x]=hist(degreesA,vals); bar(x,mh/sum(mh));
hold on; scal1=1/sum(vals(m+1:end).^-b);
semilogx(vals(m+1:end),scal1*vals(m+1:end).^-b,'r');
xlabel('d'); ylabel('P(d)');
title('Histogram of degreesA');

subplot(1,2,2);
vals = 0:1:max(degreesA);
[mh,x]=hist(degreesA,vals); bar(x,mh);
loglog(x,mh/sum(mh),'x');
hold on; loglog(x,10^-3*-x)
xlabel('d'); ylabel('P(d)');
title('Log-Log Plot of Degree Distribution');
```

Code 40.2

40.2.3 Histogram of Eigenvalues

In order to get a better understanding of the properties of power law graphs, it is important to study the histogram of eigenvalues of the adjancency matrix. Currently, there is no formula for the exact distribution of these graphs; however, extensive analysis shows some properties of the extreme and central eigenvalues.

The following code is used to histogram the eigenvalues of the power law graphs:

```
% histeig.m
% Histograms eigenvalues of vector eAmat (eigenvalues from many matrices)
% Requires paraments m0,m,t

p = 2*m*t/(t+m0)^2;
n = t+m0;
step = 0.1;
vals = -5:step:5;
scal2 = sqrt(2*n*p*(1-p));
hold off;
plot(vals,real(sqrt(4-vals.^2))/sum(real(sqrt(4-vals.^2)))*scal2);
hold on
eAscale = eAmat/scal2;
[mh,x]=hist(eAscale,vals); bar(x,mh/sum(mh)*scal2);
xlabel('eigenval');
title('Eigenvalue Distribution for Power Law Graphs');
```

Code 40.3

Figure **??** shows the normalized histogram of eigenvalues from ten power law graphs. The figure also shows the semi-circle (histogram of eigenvalues for a random matrix). The outlying high eigenvalues can be ignored since they occur because the sum of the rows of the matrix is non-zero. The important characteristics of the histogram is the tails of distribution that go outside the semicircle and the triangular peak near zero. Within the following section, we will examine why these occur.

Calculation of Moments

There are two ways of calculating the moments of the distribution of eigenvalues.

$$\begin{aligned} \mu_i &= \int x^i p_X(x) dx \\ &= \frac{1}{N} Trace(A^i) \end{aligned}$$

where $p_X(x)$ is the non-normalized distribution of eigenvalues and N is the number of vertices. Using both methods on the ten matrices, the first four moments ($\mu_1 ... \mu_4$) were computed. As expected, they are equal:

```
momHist =
         0    9.9502    4.8710  434.4159

momTrace =
   -0.0000    9.9496    4.8531  433.7473
```

The moments of a graph are related to the number of loops of a certain size. In particular $N \cdot \mu_i$ is equal to the number of loops of size i. Since the number of loops of size two is equal to the sum of degrees of all the vertices, the second moment, μ_2, is the average degree of a vertex in the graph. Within the results above, $N = m_0 + t = 1005$ and the total degree is $2mt = 10000$. Thus, as expected the second moment, is equal to $2mt/N = 9.9502$.

Another interesting thing to note is that the distribution appears to be even since the first moment is zero and the graph looks symmetric around zero (ignoring the high eigenvalues). However, the third moment is non-zero, thus the distribution cannot be even. Remember that the third moment also relates to the number of loops of size in three in the graph (number of triangles). In power law graphs, there is no reason that the number of triangles is zero, thus the distribution is not even as seen in the calculations.

Highest Eigenvalues

The eigenvalue histogram of the power law graphs has long tails that stretch out beyond the values in the semi-circle. A later section will show that these tails are actually power law distributed and related to the square root of the highest degrees in the graph.

Center of Distribution

There are two conjectures about the central part of the distribution. [139] suggest that the center is triangle-like. [181] suggest that the density of eigenvalues in the center is related to $exp(-|\lambda|/a)$ for some constant a. In Figure **??**, each theory is fitted to the histogram. It appears that the triangle captures the behavior more accurately, thus we will study this in further detail.

The formula for a triangle distribution whose base is $[-\alpha, \alpha]$ and height is $1/\alpha$ is:

$$f_X(x) = \begin{pmatrix} \frac{1}{\alpha^2}x + \frac{1}{\alpha}, & \text{if } -\alpha \leq x \leq 0 \\ -\frac{1}{\alpha^2}x + \frac{1}{\alpha}, & \text{if } 0 \leq x \leq -\alpha \\ 0, & \text{otherwise} \end{pmatrix}$$

Calculating the moments of this, we get:

$$\mu_i = \begin{pmatrix} \frac{2}{(i+1)(i+2)}\alpha^i, & \text{even n} \\ 0, & \text{odd n} \end{pmatrix}$$

Using the Plemelj equation, we get the moment generating function

$$m(z) = -\frac{2}{z}\sum_{i=0}^{\infty} \frac{1}{(2i+1)(2i+2)} \left(\frac{\alpha}{z}\right)^{2i}$$

When we compute the moments of the triangle fitted to our eigenvalue histogram ($\alpha = 6.57$), we get:

```
triMom =
         0    7.1911         0  124.1101
```

The values of these moments are not close to the moments of the distribution. Therefore it is inaccurate to simplify the eigenvalue distribution of power law graphs to a triangle.

Normally, we would expect a high concentration of eigenvalues around zero if there are lots of independent clusters in the graph. This is because the eigenvalues are small in graphs with small degrees and the eigenvalues of a graph consisting of two disjoint graphs is the multiset of the eigenvalues of each individual graph. However, in the power law graphs created, the graph is fully connected. [139] conjectures that this high triangle concentration around zero is caused from the eigenvectors for the small eigenvalues being concentrated on a small subset of the graph. We tested this theory in MATLAB, no clear results were shown.

40.2.4 Power Law Distribution of Large Eigenvalues

The largest eigenvalue

It is simple to show that the largest eigenvalue is approximately the square rrot of the largest degree. The proof is outlined below [181]:

To start, we start with two Nx1 vectors x and y such that $y(n) = A^n x$. Thus, if we set $x_j = 1$ and all other $x_i = 0$, then $y_k(n)$ equals the number of ways to get from vertex j to k within n time steps. By breaking up this formula into its eigenvalue/vector decomposition, we get:

$$y_i(n) = \sum_l \sum_j \lambda_l^n v_{i,l} v_{j,l} x_j$$

where $v_{i,l}$ is the i-th component of eigenvector l. Let us call $y_{j,i}(n)$ the result of setting x_j to 1 and all other components to zero. Thus we know that:

$$y_{j,i}(n) = \sum_l \lambda_l^n v_{i,l} v_{j,l}$$

Now, we assume that for large n, this sum can be approximated by the highest eigenvalue. Within the power law graph, we also assume there is one hub in the graph, h, with the highest degree, d_h. As we will see, the degree of the hub governs the size of the largest eigenvalue.

$$y_{j,i}(n) = \lambda_1^n v_{i,1} v_{j,1}$$

By looking at loops of size n from the hub, we see that

$$y_{h,h}(n) = \lambda_1^n v_{h,1}^2$$

We also know that the number of ways to go from the hub to itself in $n + 2$ steps is:

$$y_{h,h}(n+2) = y_{h,h}(n) y_{h,h}(2) + \sum_{j \neq h} y_{h,j}(n) y_{h,j}(2)$$

By divding though by $y_{h,h}(n)$ on both sides, we get:

$$\lambda_1^2 = y_{h,h}(2) + \frac{\sum_{j \neq h} y_{h,j}(n) y_{h,j}(2)}{y_{h,h}(n)}$$

We know that $y_{h,h}(2)$ is the degree of h. The second term is negligble in size compared to the first. Thus, we get that $\lambda_1^2 = d_h$. The largest eigenvalue λ_1 is approximately equal to the $\sqrt{d_h}$, the square root of the highest degree.

Power law tails

When the degrees of a graph are power law distributedwith paramenter γ, the highest eigenvalues are equal to $\sqrt{d_1}...\sqrt{d_k}$, where d_1 is the highest degree of the graph [322]. Thus, the eigenvalues are power law distributed with a paramenter $\gamma/2$.

The proof given by Mihail et. al. uses some simple facts about matrix eigenvalues to show that if we have big stars in the graph, the largest eigenvalues are approximately the square root of the degrees of these stars.subsubsection*

The following facts are necessary:

- For a star graph (one center vertex, s, and multiple neighbors), the only non-zero eigenvalues of the graph are $\sqrt{d_s}$ and $-\sqrt{d_s}$.
- The eigenvalues of a graph are always less than both the max degree and the square root of the number of edges.
- $\lambda_i(A) + \lambda_n(B) \leq \lambda_i(A+B) \leq \lambda_i(A) + \lambda_i(B)$, where λ_n is the smallest eigenvalue.

These facts imply that if we can split a power law graph into the following subgraphs:

- Vertex disjoint stars S_i, with degrees d_i for $i = 1...k$
- Vertex disjoint G_j (also disjoint from stars) such that $min(d(G_j), \sqrt{e(G_j)} = o(d_k)$, where $d(G_j)$ is the max degree of G_j and $e(G_j)$ is the number of edges
- Graph H with $min(d(H), \sqrt{e(H)}) = o(d_k)$

then, the largest eigenvalues of the power law graph are bounded as follows:

$$\sqrt{d_i}(1 - o(1)) \leq \lambda_i \leq \sqrt{d_i}(1 + o(1)) \text{ for } i = 1...k$$

We need to prove that a power law graph can be split up into the subgraphs described above. We will use a different technique of generating power law graphs. Here, each vertex has an expected degree d_i, and two vertices i and j connect with probability $d_i d_j / D$ where $D = \sum_k d_k /$. Thus we can pick an expected power law distribution for the degrees.

Through a series of bounds, [322] show that with high probability, we will get disjoint stars whose degrees are approximately $d_1...d_k$ (the highest degrees in the distribution). Using Chernoff bounds, they show that the size of subgraphs G_j and H will stay small.

Thus, using the theorem above, we get that the largest eigenvalues are approximate the square root of largest degrees. This proof holds for large values of the power law exponent, $\gamma > 3$. [81] recently worked on a proof which hold for smaller values of γ.

40.3 The Sample Covariance Matrix of Mixed Data Under Noisy and Limited Data

[This section was originally written by Petros T. Boufounos]

40.3.1 Introduction

A common scenario in science and engineering is the analysis of mixed data. This problem appears in communications, auditory scene analysis, array signal processing, DNA data analysis, econometrics and various other scientific fields. The study of this scenario has generated a whole range of data processing techniques, such as principal component analysis, independent component analysis, clustering, and others, depending on assumptions such as linearity, mixing models, availability of data, and noise levels. In this paper we examine an often overlooked aspect of the problem, namely how to determine the number of sources present in the measured data.

The problem in its generic form assumes there are N data sources of interest, usually uncorrelated, $\{s_i | i = 1, \ldots, N\}$ measured ("sensed") by M variables ("sensors") $\{x_i | i = 1, \ldots, N\}$ at T sampling instances (usually time or space points). A linear mixing model with additive uncorrelated Gaussian noise results to the following matrix equation:

$$X = AS + \sigma U \tag{40.8}$$

where X is the $M \times T$ sensor samples, A is the $M \times N$ mixing matrix, S is the $N \times T$ source data, U the $M \times T$ unit variance i.i.d. noise process, and σ^2 the noise variance. We will assume we only have access to the samples X.

Usually one is interested in estimating S–also known as the source separation problem–but this is not the problem we will try to explore. Several algorithms dealing with this scenario, with various results, depending on the application and the problem assumptions. Most of these algorithms, however, assume that the value of N is known in advance. If this value is not known, useful data might often be rejected or extra noise might be considered significant data.

Instead, we will focus on the problem of estimating N, using as few data as possible. We will show that as $T \to \infty$ this estimation is trivial. However, it is often desirable to estimate N with very few data, in order to initiate a source separation algorithm. This is of particular importance if the source separation needs to be on-line, for example on streaming data. The rest of the paper explores the model further, presents some theoretical work on the random matrices involved, and examines the results of some simulation studies. The paper concludes by evaluating the results and suggesting directions for further research.

40.3.2 The model in detail

Having presented the model, one needs to make some further assumptions and definitions that will aid our development. To do so we revisit the basic model equation:

$$X = AS + \sigma U \tag{40.9}$$

We consider A to be random with i.i.d., unit-variance elements. Furthermore, we assume that the N sources are orthogonal and have constant power per unit of time, i.e. $SS^T = TI_N$, where I_N is the $N \times N$ identity matrix. Also, the sources are orthogonal to the noise process: $SU^T = O$. Finally, the number of sources N is less than the sensors M. This is the same model as presented in [137].

One quantity we will repeatedly use is the sample covariance matrix C_X and its eigenvalues λ_X. This is calculated using

$$\begin{aligned} C_X &= \frac{1}{T}XX^T && (40.10)\\ &= \frac{1}{T}(AS+\sigma U)(AS+\sigma U)^T && (40.11)\\ &= \frac{1}{T}(ASS^TA^T+\sigma ASU^T+\sigma US^TA^T+\sigma^2UU^T) && (40.12)\\ \Rightarrow C_X &= AA^T+\frac{\sigma^2}{T}UU^T && (40.13) \end{aligned}$$

If we let time T go to infinity, keeping N and M constant this matrix converges to the true covariance matrix of X, i.e. $AA^T+\sigma^2 I_M$. Therefore, the noise will add a $\sigma^2>0$ floor to all λ_A, the eigenvalues of AA^T. AA^T is by construction a positive semidefinite matrix. In other words, at $T\to\infty$, the variance of the noise will be the smallest eigenvalue of the sample covariance matrix: $\sigma^2=\min(\lambda_X)$. Thus, we can determine the rank of A just by counting the number of eigenvalues strictly larger than the noise floor.

Unfortunately things are not as nice if time is finite. In that case UU^T is not a multiple of the identity matrix, and the eigenvalues of a sum of matrices do not add up. Still, some theoretical results can be exploited to attack the problem. To use them in the next section we need to define two variables $y=\frac{M}{N}$ and $c=\frac{T}{N}$. These are necessary when we discuss about asymptotic behavior of random matrices and their eigenvalues. We can think of y representing the inverse of the rank of A: when $y\to 1$, A becomes full rank; when $y\to\infty$, A becomes null-rank. Similarly we can think of c as the amount of data in X, compared to the number of sensors. As $c\to 0$, then we have almost no data; as $c\to\infty$, then we have plenty of data. We are now ready to proceed to the theoretical work on the equation.

40.3.3 Theoretical evaluation of the problem

The unknown parameters in our problem are the rank N of A (or, equivalently, y) and the noise variance σ^2. In order to attack the problem, we concentrate on the analysis of the eigenvalues of the sample covariance matrix. Our goal is to get a handle of the density $f_X(\lambda_X|\sigma^2,N)$ and optimize this density for the two parameters given the sample data. Unfortunately this is a hard problem and we cannot solve it analytically. However, we develop the necessary steps to perform simulations and get a better sense on how these eigenvalues behave with varying parameters.

The basic theoretical work for our problem is the Marcenko-Pastur law, also described in several papers by Silverstein and others, such as [410]. This is an asymptotic result in the matrix dimensions. Fortunately simulations show it holds well even in very low dimensions. We use that result both for modeling the eigenvalues of AA^T and the ones of C_X, using the strategy outlined in Figure **??**.

Given the rank of A ($=N$) we compute the Stieltjes transform of its distribution using the Marcenko-Pastur law. Furthermore, given the noise variance we estimate the density of the eigenvalues of $\sigma^2 I$ (which is just the step function). We would like to use these two quantities and the Marcenko-Pastur law to evaluate the Stieltjes transform of the eigenvalue distribution of C_X: $m(z)$. We shall then invert this transform to obtain $f(\lambda_x|N,\sigma^2)$. On that function we should be able to use maximum-likelihood or Bayesian estimation techniques to estimate N given our sample data.

In the next section we will follow that strategy and show where it fails. We will suggest numerical solutions and perform some simulations to gain some further intuition about the problem.

The eigenvalues of AA^T

Since A is not full rank, $(M-N)$ eigenvalues of AA^T are zero. The remaining eigenvalues will have distribution as the ones of A^TA, known from the Marcenko-Pastur law. Thus, as $N \to \infty$ and M grows such that $\frac{M}{N} \to y > 1$, the probability density of the eigenvalues of AA^T becomes

$$f_y(x) = \frac{\sqrt{(x-y_1)(y_2-x)}}{2\pi xy} I_{[y_1,y_2]}(x) \tag{40.14}$$

$$f_A(x) = (1-\frac{1}{y})\delta(x) + \frac{f_y(x)}{y} \tag{40.15}$$

where $y_1 = (1-\sqrt(y))^2$ and $y_2 = (1+\sqrt(y))^2$, and $I_S(x) = 1$ if $x \in S$ and 0 otherwise.

Thus, we can easily compute the Stieltjes transform $m_A(z)$:

$$m_A(z) = \frac{1}{2y} + \frac{3(1-y)}{2zy} + \frac{i\pi}{y} f_y(z) \tag{40.16}$$

The Marcenko-Pastur law on the whole model

We can now use the same law on the whole model to calculate $m(z)$, the Stieltjes transform of $f_X(\lambda_x|N,\sigma^2)$. According to the law, for a model of the form $AA^T + UTU^T = AA^T + U(\sigma^2 I)U^T$ we need to solve

$$m(z) = m_A\left(z - c\int \frac{\tau h(\tau) d\tau}{1+\tau m(z)}\right), \tag{40.17}$$

where $h(\tau)$ is the distribution of the eigenvalues of the diagonal matrix T. In this case, substituting $h(\tau) = \delta(\tau - \sigma^2)$, we obtain:

$$\begin{aligned} m(z) &= m_A\left(z - \frac{c\sigma^2}{1+\sigma^2 m(z)}\right) \\ &= \frac{1}{2y} + \frac{3(1-y)}{2\left(z - \frac{c\sigma^2}{1+\sigma^2 m(z)}\right)y} + \frac{i\pi}{y} f_y\left(z - \frac{c\sigma^2}{1+\sigma^2 m(z)}\right) \\ &= \frac{1}{2y} + \frac{3(1-y)}{2\left(z - \frac{c\sigma^2}{1+\sigma^2 m(z)}\right)y} + \\ &\quad \frac{i\pi}{y} \frac{\sqrt{\left(\left(z - \frac{c\sigma^2}{1+\sigma^2 m(z)}\right) - y_1\right)\left(y_2 - \left(z - \frac{c\sigma^2}{1+\sigma^2 m(z)}\right)\right)}}{2\pi\left(z - \frac{c\sigma^2}{1+\sigma^2 m(z)}\right)y} \end{aligned}$$

to obtain $m(z)$. However, this equation is almost impossible to solve analytically (even with such tools as Mathematica). Alternatively, one could use numerical methods. Unfortunately the solution for $m(z)$ needs to be inverted to calculate the density f, which then has to be optimized for N and σ^2. Numerical solutions would make that process slow for practical use. Instead, we will focus on gaining some insight on how $m(z)$ behaves through simulations.

40.3.4 Matlab Simulations

In this section we will perform some simulations and present some results that verify some of the theory and provide some on the problem. The goal of these simulations is not to solve the problem at hand but to give some directions on how the solution might behave.

The eigenvalues of AA^T for small A

As a sanity check to our assumptions we need first to verify that the asymptotic formulas in 40.3.3 are a good approximation even for small matrices, as is often the case. To do so we will first plot the theoretical density of the eigenvalues as y varies. Figure **??** shows the theoretical probability density function f and the cumulative density function F for several y. The arrow points in the direction of y decreasing towards 0. Note that in the case of the pdf, we do not plot the impulses at the origin. However they show up as step functions in the cdf.

Figure **??** shows simulations for random Gaussian A's of size 10×20 and 20×10 elements. On the top the eigenvalues are plotted sorted by their index, while the bottom plots superimpose the empirical cumulative density function with the theoretical one. We can see that the fit is quite good. Experiments show that this is a good fit for other values of y as well. Also, we verified that the result is robust to changes in the distribution of the elements of A.

Empirical evaluation of $m(z)$

Another way we can improve our intuition on the problem is to empirically evaluate $m(z)$ for large matrices and see how it behaves with changing parameters. Indeed, we do so for $N = 500$ and numerically compute $m(z)$ from the definition of the Stieltjes transform. We plot and contrast the results for various parameter values and observe how $m(z)$ behaves. However, to get a sense of how a typical $m(z)$ might behave, the plot of the theoretical $m_A(z)$ in two different scales is shown in Figure **??**.

We can see that although there is significant action in the origin, the rest of the plane is not just roughly flat. As we zoom out, we observe a discontinuity. Figure **??** shows how this discontinuity varies with y, which is the variable we are interested to estimate.

We are now ready to produce the empirical $m(z)$ for different conditions. Figure **??** shows how changing y affects $m(z)$. The figures are generated for $c = 0.5$ and $\sigma^2 = 1$. We notice that lower y creates a bigger discontinuity along the real axis, especially at the imaginary part.

If we vary c, keeping constant $y = 2$, we produce the plots of Figure **??**. In these plots we can notice how the presence of more data (higher c softens the discontinuities in $m(z)$. This might be related to the reduction in uncertainty of the estimation.

Finally, we would like to study the effect of the noise in Figure **??**. Increasing the noise variance from 0.1 to 5 introduces more discontinuities in $m(z)$, which is probably consistent with reducing c above. However, we can see from the plots that discontinuities are of different form so, at least in principle, we should be able to distinguish among them.

We can see from the above plots that numerical computation of $m(z)$ might reveal several aspects of the problem. Indeed, one potential solution to the problem is the numerical evaluation of the sample $m(z)$. N can then be estimated by applying pattern recognition or estimation techniques on the empirical $m(z)$.

40.3.5 Conclusions

In this paper we tried to attack a frequently overlooked problem: how to infer the rank of a mixing matrix from the sample mixed data. Solution to this problem is useful under a variety of scenarios, described in the introduction. Unfortunately, we showed that the model of the problem lends itself to only partial analytic manipulation. A closed form solution still an open–if not impossible– problem. Thus, we attacked the problem numerically and performed simulations that shed some

light on the model. Although a solution was not provided, we established that the sample $m(z)$ contains significant information and might have some significance in solving the problem.