The Marchenko-Pastur Distribution: From Understanding to Application

Elias Little

December 11, 2020

Overview

In this paper, my goal is to introduce the basics of what actually comprises random matrices, what we can learn about certain matrices, and then from there move into numerical analysis and application of the concepts that I will introduce in the first half. Additionally, for all of the graphs and numerical analysis, I will either include the code within the page if it's relevant or refer to the appendix where the remainder of the code will be. All of the code was written in Julia.[1]

[1] One of the best languages

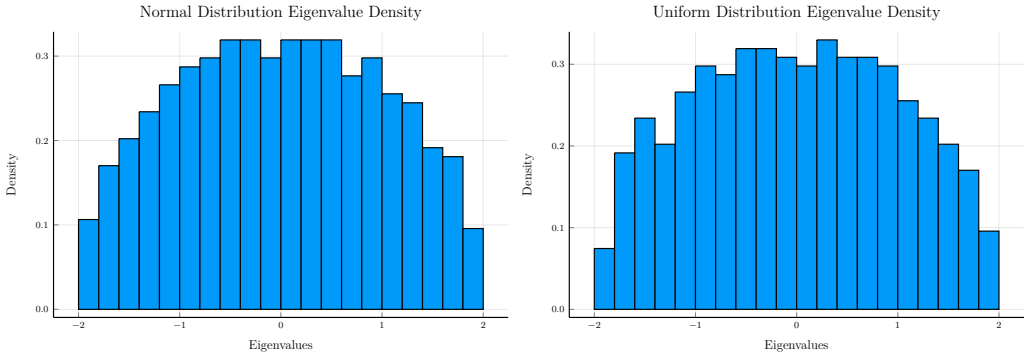
1 What Even *Is* a Random Matrix?

A random matrix isn't all that complicated, it's simply a matrix where each element is a random variable. Taken at face value, it doesn't sound all that special, but what it now allows us to do is combine ideas and tools from statistics and probability, with those from other fields that use deterministic matrices, primarily linear algebra. Statistics allows us to model and analyze non-determinant systems such as a coin toss, or usage of public transportation throughout the day. Random variables are the specific object that is the gateway to statistics. Linear algebra on the other hand gives us a framework for working with lots of information, often in many dimensions, and being able to operate and analyze that information. Vectors and matrices are the primary objects used in linear algebra and are what power its capabilities. Thus, when we can combine the two we now have a framework for dealing with lots of multi-dimensional random variables, and can do so easily with the use of random matrices, the gateway to this new framework, called Random Matrix Theory [3].

1.1 Wigner's Semi-Circle Law

One of the most important aspects of linear algebra is the use of eigenvalues and eigenvectors. These are used every day from computer graphics to signal-processing, to stress analysis, and understanding quantum mechanics. In many of these situations, the eigenvalues of a matrix are one of the most

important tools for understanding. For example, they are used in the Fourier transform which uses eigenvalues to decompose a signal into its constituent frequencies. So, a logical question to ask would be asking what the eigenvalues of different random matrices look like. We will start by taking two symmetric random matrices, one where the values are normally distributed, and the other where the values are uniformly distributed. For each of these, we calculate the eigenvalues then plot their densities. Doing so results in the following two graphs 5 6



As we can see, they both look very similar and happen to both approximate semi-circles. This is, in essence, Wigner's semi-circle law, that the distribution of eigenvalues of (suitably large) random matrices resembles that of a semi-circle [2]. More technically, the distribution has the probability density function (PDF) $\frac{2}{\pi R^2}\sqrt{R^2 - x^2}$.

As a small aside, although both of the graphs appear to be semi-circles, the scale and the factor in the PDF tell us otherwise, so really it should be named Wigner's Semi-Ellipse Law.

1.2 The Marchenko-Pastur Distribution

Now that we've seen how eigenvalues of symmetric random matrices are distributed, we are prepared to look at the Marchenko-Pastur Distribution or law. The Marchenko-Pastur law was discovered a little over a decade after Wigner's law but comes from a similar motivation. Instead of looking at the distribution of eigenvalues of a symmetric matrix, it looks at the distribution of eigenvalues of the covariance matrix of a given rectangular matrix. For example, if we have a matrix X of size $T \times N$ where each element if independent and identically distributed with mean zero and variance σ^2, then the law tells us that the covariance matrix $C = T^{-1}X'X$ has eigenvalues λ that converge as $N \to +\infty, T \to +\infty$ such that $1 < \frac{T}{N} < +\infty$ to the Marchenko-Pastur PDF:

$$f(\lambda) = \begin{cases} \frac{T}{N}\frac{\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}}{2\pi\lambda\sigma^2} & \text{if } \lambda \in [\lambda_-, \lambda_+] \\ 0 & \text{if } \lambda \notin [\lambda_-, \lambda_+] \end{cases}$$

where λ_+ and λ_- are the maximum and minimum expected eigenvalues respectively, and are defined as $\lambda_\pm = \sigma^2\left(1 \pm \sqrt{N/T}\right)^2$. This means that eigenvalues $\lambda \in [\lambda_-, \lambda_+]$ are to be expected from random behavior, but eigenvalues larger than the range are more likely attributed to the signal and not random noise.

1.3 Principal Component Analysis & Correlation

This law lends itself well to use in Principal Component Analysis (PCA) because of the relationship between eigenvalues of the covariance matrix and the singular values used for PCA [1]. Because our covariance matrix $C = T^{-1}X'X$ is symmetric, it can be diagonalized as $C = VLV^T$ where V is a matrix of the eigenvectors and L is a diagonal matrix of the eigenvalues along the diagonal. We can also perform a singular value decomposition on X, which gives us $X = USV^T$ where U is a unitary matrix, and S is a diagonal matrix of the singular values along the diagonal. It can then easily be seen that

$$C = VSU^TUSV^T * T^{-1} = VS^2V^T * T^{-1}$$

This means that the singular values are the square root of the eigenvalues of C. So by using the Marchenko-Pastur law, we can filter out random eigenvalues and thus improve our singular value matrix and our PCA overall.

2 Theoretical Application

Now that we understand what the Marchenko-Pastur distribution is and how it's useful, we can visualize how it works. First, we will define a function `mpPDF` that will generate the probability density function given the variance σ^2, the ratio of the rectangle $q = \frac{T}{N}$, and the number of points to evaluate.

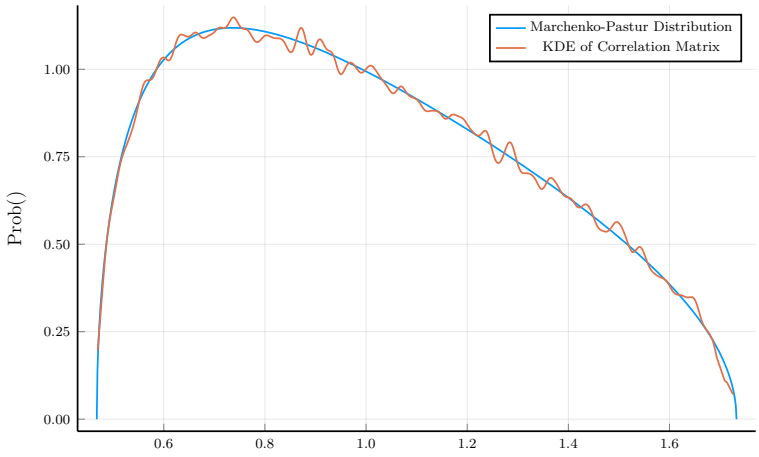
```
function mpPDF(var, q, pts)
  # Marchenko-Pastur PDF
```

```
  # q=T/N
  eMin,eMax = var*(1-sqrt(1/q))^2, var*(1+sqrt(1/q))^2
  eVal = range(eMin,stop=eMax,length=pts)
  pdf = q./(2*pi*var*eVal).*.sqrt((eMax.-eVal).*(eVal.-eMin))
  return pdf,eVal
end
```

Now we will construct our random matrix X from Normally distributed values. We then get the eigenvalues and vectors of the correlation matrix of X. From there we construct the PDF for this matrix and fit a kernel density estimate to the eigenvalues for graphing.

```
# Random Matrix
X=rand(Normal(0,1),10000,1000)
# Get the eigenvalues and vectors of the correlation matrix of X
eVal0,eVec0 = getPCA(cor(X))
# Calculates the theoretical Marchenko-Pastur distribution
pdf0,evals0=mpPDF(1.0,size(X)[1]/size(X)[2],1000)
# Fits a Kernel Density Estimate to the eigenvalues of the
↪  correlation matrix
pdf1,evals1=fitKDE(diag(eVal0),0.01)
```

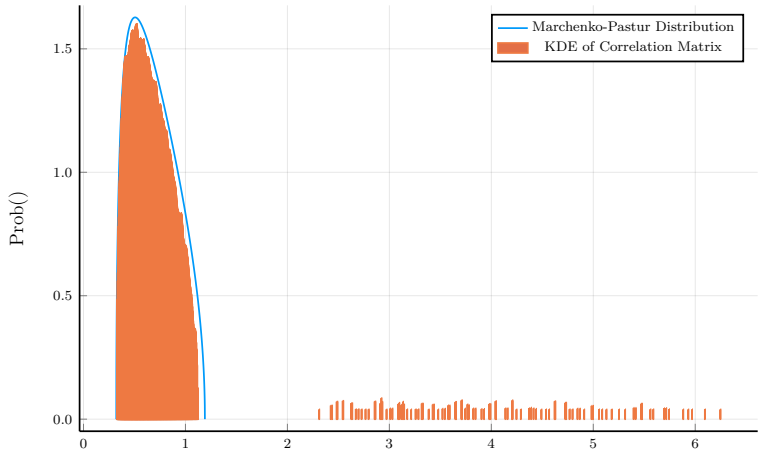
The additional functions are defined here: 1 2. We can now see that the randomly generated matrix does indeed follow the Marchenko-Pastur distribution as expected. 7



But what does it look like when we don't have a purely random matrix, but instead have a matrix with a mix of signal and noise? To do this, I created a matrix with values that were normally distributed, then added uniformly distributed noise:

```
function getRandCov(nCols, nFactors)
  w = rand(Normal(),nCols,nFactors)
  # Our signal, we're adding nFactors = 100 number of signal
  ↪ factors
  w_cov = w * w' # Covariance of w
  w_cov += Diagonal(rand(Uniform(),nCols)) # Adding noise to the
  ↪ matrix
  return w_cov
end
alpha, nCols, nFactors, q = 0.995, 1000, 100, 10
covar = cov(rand(Normal(), nCols*q,nCols))
covar = alpha*covar+(1-alpha)*getRandCov(nCols,nFactors)
corr0 = cov2corr(covar)
eVal01, eVec01=getPCA(corr0)
```

This time I also used a bar graph so we can see the individual eigenvalues compared to just their estimated PDF. 8



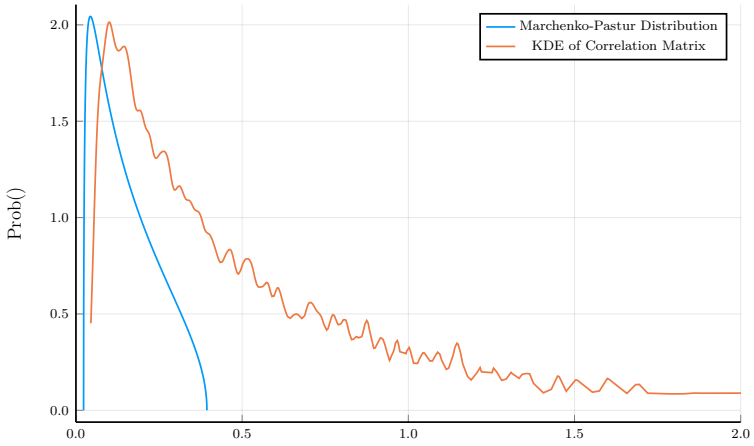
This shows clearly how the Marchenko-Pastur distribution captures all of the eigenvalues related to random noise while the eigenvalues relating to the signal are untouched. From here we can define a function that essentially filters out the eigenvalues that are captured by the Marchenko-Pastur distribution returning a denoised correlation matrix [4]. In order to do so we correct the set of eigenvalues by setting all non-important eigenvalues $\lambda_j = \frac{1}{N-i}\sum_{k=i+1}^{N}\lambda_k,\ j = i+1, ..., N$. This preserves the trace of the correlation matrix while significantly reducing the importance of these eigenvalues. The corrected set of eigenvalues Λ can then be used to create the denoised covariance matrix as we did before:

$$C = V\Lambda V^T$$

The functions to accomplish this can be found at [3] [4]

3 Application to *Real* Data

We have now seen that the Marchenko-Pastur law does indeed work with data that is generated to our specification, but how about if we use real-life data? I took stock return data from 470 companies over 5 years between 2013 and 2018 11, 10 and tried to apply the same techniques. Plotting the real distribution of eigenvalues against the best fit Marchenko-Pastur PDF resulted in this graph 9:



Here we can see that the distribution of eigenvalues of the real data is not nearly as nice and conforming as compared to the generated data. And although the Marchenko-Pastur distribution doesn't fit particularly well to the data, it does capture some noise, and it appears that with some work transforming the data correctly it could capture much more noise than it presently does. So even though it's not nearly as nice or clear-cut as the generated case, we can still see that with real-life data which is very messy, this technique still has merit in eliminating random noise.

4 Moving Forward

Moving forward with this project I want to try and get better results from real data. In addition, I want to work with other datasets, not just stock market data, but see how this technique [2] applies to other areas outside of finance.

If this technique is capable of drastically reducing noise from real data, I then hope to move to the next step of actually constructing portfolios and trading strategies from the results and backtest them to see how they would have performed. Essentially testing how strong the recovered signal is.

References

[1] amoeba (https://stats.stackexchange.com/users/28666/amoeba), *Relationship between svd and pca. how to use svd to perform pca?*, 2015. URL:https://stats.stackexchange.com/q/134283 (version: 2020-06-11).

[2] Jim Gatheral, *Random matrix theory and covariance estimation*, Merrill Lynch, 2008.

[3] Giacomo Livan, Marcel Novaes, and Pierpaolo Vivo, *Introduction to random matrices*, SpringerBriefs in Mathematical Physics (2018).

[4] Marcos M. López de Prado, *Machine learning for asset managers*, Elements in Quantitative Finance, Cambridge University Press, 2020.

[2] Which is almost always talked about for use in a financial context

All of the following is a appendix for code used, and is just for the sake of completeness.

Appendix A Functions Referenced

```
function getPCA(matrix)
  # Principal Component Analysis
  # Get eVal, eVec from a Hermitian matrix
  eVal, eVec = LinearAlgebra.eigen(matrix)
  indices = sortperm(eVal, rev=true)
  eVal, eVec = eVal[indices], eVec[:,indices]
  eVal = Diagonal(eVal)
  return eVal,eVec
end
```

Function returns eigenvalues and eigenvectors in reverse order

```
function fitKDE(obs,bWidth=0.25,kernel="gaussian", x=nothing)
  # Fit kernel to a series of observations, and derive the probability
  ↪  of the observations
  # x is the array of values on which the fit KDE will be evaluated
  if x == nothing
    x = reshape(reverse(unique(obs)),1,:)
  end
  if length(size(x)) == 1
    x = reshape(x,1,:)
  end
  k = kerneldensity(obs,xeval=x[:],h=bWidth) #Default is gaussian
  return k, x[:]
end
```

Function returns kernel density estimate for given observations

```
function cov2corr(cov)
  # Derive the correlation matrix from a covariance matrix
  std = .sqrt(diag(cov))
  corr = cov./(std.*std')
  corr[corr .< -1] .= -1
  corr[corr .> 1] .= 1
  corr
end
```

Function converts a covariance matrix into a correlation matrix

```
function denoisedCorr(eVals, W, nFactors) # W is the matrix of
↪  eigenvectors
  lam = copy(diag(eVals))
  lam[nFactors:end] .= sum(lam[nFactors:end])/(size(lam)[1]-nFactors)
  L = Diagonal(lam)
  cov = W * L * W' # Covariance matrix
  corr2 = cov2corr(cov) # Correlation matrix
end
```

Function returns a denoised correlation matrix

Appendix B Plots Referenced

```
n = 470
M = rand(Normal(0,1),470,470)
sym_M = (M+M')/sqrt(2*n) #Create a symmetric matrix
histogram(diag(getPCA(sym_M)[1]), bins = 20, normalized=true,
↪  xlabel="Eigenvalues",ylabel="Density",legend=false,title="Normal
↪  Distribution Eigenvalue Density")
md"Wigner Semi-circle for Normal Distribution"
```

Generates a Wigner semi-circle distribution from a Normal matrix

```
n = 470
M = rand(Uniform(0,1),470,470)
sym_M = sqrt(12)*(M+M'.-1)/sqrt(2*n) #Create a symmetric matrix
histogram(diag(getPCA(sym_M)[1]), bins = 20, normalized=true,
↪  xlabel="Eigenvalues",ylabel="Density",legend=false, title="Uniform
↪  Distribution Eigenvalue Density")
md"Wigner Semi-circle for Normal Distribution"
```

Generates a Wigner semi-circle distribution from a Uniform matrix

```
plot(evals0,pdf0,label="Marchenko-Pastur Distribution")
plot!(evals1,pdf1,label="KDE of Correlation Matrix")
xlabel!("")
ylabel!("Prob()")
```

Generates a mpPDF with KDE of random matrix

```
  p1 = plot(reverse(mpPDF(var0,q,1000))...,label="Marchenko-Pastur
  ↪  Distribution")
  p2 = bar!(reverse(fitKDE2(diag(eVal01),0.01,evals0))...,label="KDE of
  ↪  Correlation Matrix",linecolor="sienna2")
  xlabel!("")
  ylabel!("Prob()")
```

Generates a mpPDF with KDE of random matrix with signal

```
Plots.plot(reverse(mpPDF(var,q,1000))...,label="Marchenko-Pastur
↪  Distribution",xlims=(0,2))
plot!(reverse(fitKDE2(diag(eVals), 0.01))...,label="KDE of Correlation
↪  Matrix",linecolor="sienna2")
xlabel!("")
ylabel!("Prob()")
```

Plots real data with fitted mpPDF

Appendix C Real Data Referenced

```
M = Matrix(mainframe)[:, 2:end]
# Two small edits that won't change the data much,
# and allows us to keep a lot more good data
M[442,568] = mean(M[442, 567:2:568])
M[361,587] = mean(M[361, 586:2:588])
M = convert.(Float64, M)
md"`mainframe` code"
```

Generates and repairs a matrix M from a dataframe `mainframe`

```
  stonks = []
  for loc in readdir("individual_stocks_5yr",join=true,)[2:end]
    ticker = match(r"[A-Z]+",loc).match
    file = CSV.File(loc) |> DataFrame
    date_returns = select(file, :date, AsTable([:open,:close]) => ByRow(x
    ↪  -> 100*(x.close - x.open)/x.open) => :returns)
    date_returns[!, :company] .= ticker
    push!(stonks, unstack(date_returns, :date, :returns))
  end
  fun = x -> length(names(x)) == 1260
  filtered_stonks = filter(fun, stonks)
  mainframe = reduce(vcat, filtered_stonks)
```

Reads data from CSV files and puts it into a DataFrame `mainframe`