~~

Random numbers and random matrices:

where quantum chaos meets number theory

Introduction

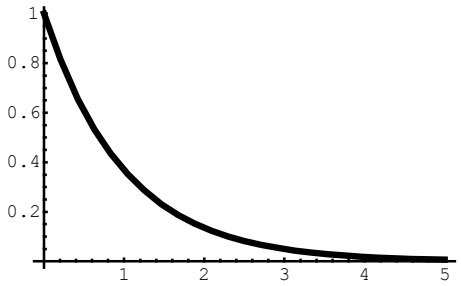
This is a *Mathematica* notebook to accompany the article "Random Numbers and Random Matrices: quantum chaos meets number theory", which has been submitted to the American Journal of Physics. All of the calculations discussed and presented in the article can be carried out using this notebook. I do not include much description of the physical relevance of the various calculations since that is covered in depth in the article (so please consult the article for that information). Instead I provide here a brief description of the calculations and the code to implement them in *Mathematica*. I have tried to make each section self-contained, although the plots of the theoretical distributions are not redone in each section. Use *Mathematica*'s Help feature for descriptions of the individual commands. If you have any questions about this material feel free to contact me at ttimberlake@berry.edu.

Plots of the Theoretical Distrubutions

In this section we create plots of the various Wigner distributions and the Poisson distrubution, as well as the Wigner semicircle law. These will be used later to compare calculated distributions to these theoretical predictions. The actual limiting distributions for random matrices are slightly different from the Wigner distributions.

Here's a plot of the Poisson distribution.

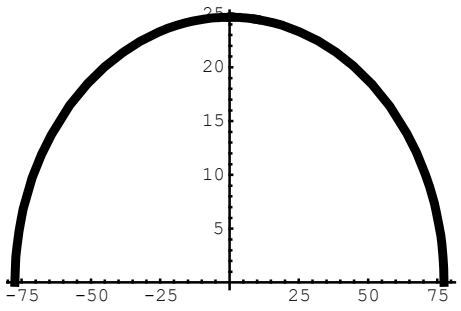
```
Poisson = Plot[Exp[-s], {s, 0, 5}, PlotStyle → {Thickness[0.015]}]
```




```
- Graphics -
```

Below we plot the theoretical density of eigenvalues known as Wigner's semicircle law (because the curve takes the shape of a semicircle centered at zero with radius $\sqrt{2N}$,where N is the dimension of the random matrix (and hence the number of eigenvalues).

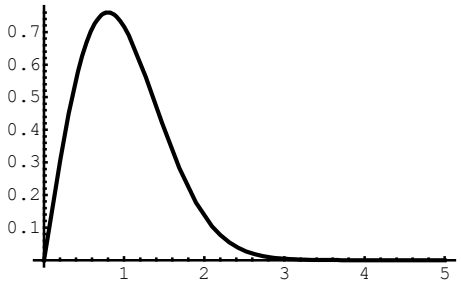
```
WSC = Plot[Sqrt[2 * n - x^2] / (Pi),
  {x, -Sqrt[2 * n], Sqrt[2 * n]}, PlotStyle → {Thickness[0.02]}]
```




```
- Graphics -
```

Here's the Wigner distribution for random real symmetric matrices (which is very close to the Gaussian Orthogonal Ensemble distribution).

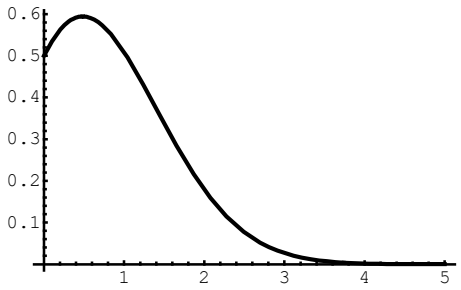
```
GOE = Plot[Pi * s * Exp[-Pi * s^2 / 4] / 2, {s, 0, 5}, PlotStyle → {Thickness[0.01]}]
```



- Graphics -

Here's the distribution for a combination of two GOE data sets.

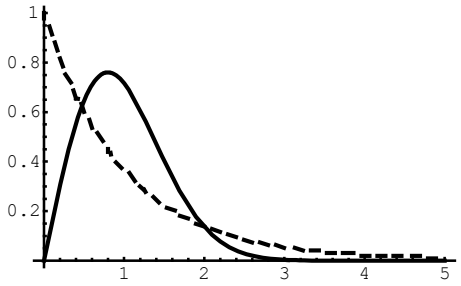
```
GOE2 = Plot[Exp[-Pi * s^2 / 8] / 2 + Pi * s * Exp[-Pi * s^2 / 16] * Erfc[Sqrt[Pi] * s / 4] / 8,
  {s, 0, 5}, PlotStyle → {Thickness[0.01]}]
```



- Graphics -

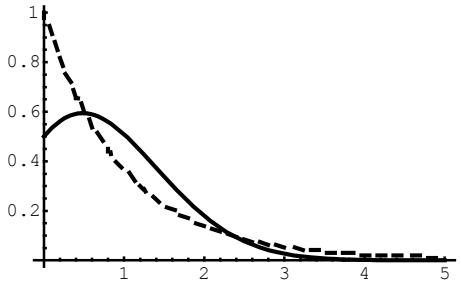
Next we plot both the Poisson distribution (dashed) and the GOE or 2 GOE distribution (solid) together. This will help to illustrate the transition from random to random-matrix statistics in our quantum map.

```
Curves1 = Plot[{Exp[-s], Pi * s * Exp[-Pi * s^2 / 4] / 2}, {s, 0, 5},
  PlotStyle → {{Thickness[0.01], Dashing[{0.02, 0.02}]}, Thickness[0.01]}]
```



- Graphics -

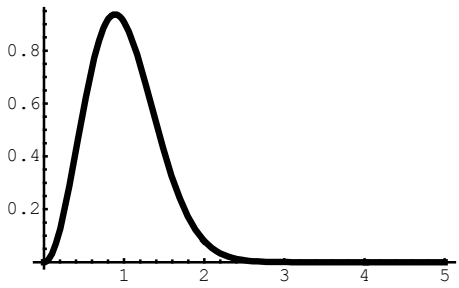
```
Curves2 = Plot[
  {Exp[-s], Exp[-Pi * s^2 / 8] / 2 + Pi * s * Exp[-Pi * s^2 / 16] * Erfc[Sqrt[Pi] * s / 4] / 8},
  {s, 0, 5}, PlotStyle → {{Thickness[0.01], Dashing[{0.02, 0.02}]}, Thickness[0.01]}]
```



- Graphics -

Finally we plot the Wigner distribution for random complex Hermitian matrices (which is very close to the Gaussian Unitary Ensemble distribution).

```
GUE = Plot[32 * s^2 * Exp[-4 * s^2 / Pi] / Pi^2, {s, 0, 5}, PlotStyle → {Thickness[0.015]}]
```




```
- Graphics -
```

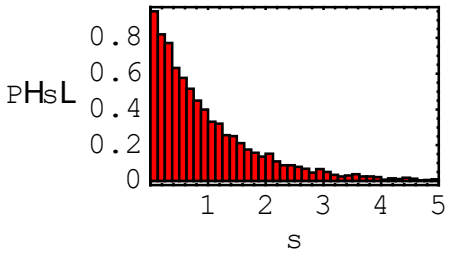
Random Numbers

In this section we look at the spacing distribution for random numbers, to see that is follows the Poisson distribution. The code below generates a list of 10,000 random numbers (all between 0 and 1), sorts them, rescales them so that the means spacing between consecutive numbers is 1, and then generates a histogram of the frequencies of different level spacings. To generate the histogram you need to load the *Mathematica* Graphics package first.

```
<< Graphics`
```

```
n = 10000;
R = Sort[Table[Random[], {i, 1, n + 1}]];
RSpacing = Table[R[[i + 1]] - R[[i]], {i, 1, n}];
RSpacingNorm = RSpacing / Mean[RSpacing];
int = Table[0 + i / 8, {i, 0, 40}];
RHist = Histogram[RSpacingNorm, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```

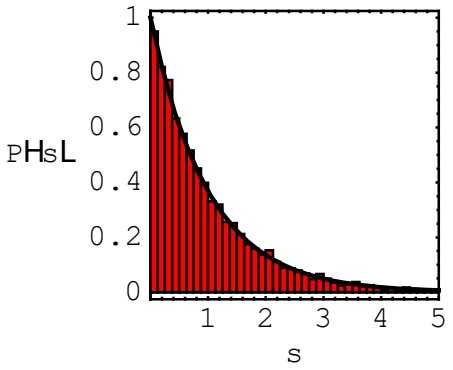
```
Histogram::rtail : Warning: 76 points from the right tail of the data, greater than or equal to 5, are
   not included in histogram.
```




```
- Graphics -
```

Next we show the level spacing histogram along with a plot of the Poisson distribution. It is clear from the results that random numbers have a Poisson level spacing distribution.

```
Show[RHist, Poisson, AspectRatio → 1]
```




```
- Graphics -
```

Random Matrix - GOE

In this section we will generate a Gaussian-random real symmetric matrix, compute its eigenvalues, calculate the level spacings, and generate a histogram of these level spacings. This histogram closely matches the GOE distribution. Be aware that these calculations take some time (roughly 900 seconds on a PowerMac G4 733 MHz).

First we must construct the matrix. To do this we need to load the *Mathematica* package entitled "Continuous Distributions" that is contained within the Statistics package.

```
<< Statistics`ContinuousDistributions`
```

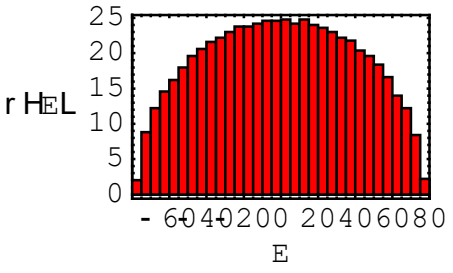
Now we construct and n by n random real symmetric matrix. Each matrix element is a random number and these numbers are Gaussian distributed with a mean of zero and a standard deviation of one (except the diagonal elements which have a standard deviation of square root of 2). Here we generate a 3000 by 3000 matrix. Once the matrix is generated we can find its eigenvalues and sort them.

```
n = 3000;
M = Table[Which[i == j, Random[NormalDistribution[0, 1]], i < j,
     Random[NormalDistribution[0, 1 / Sqrt[2]]], i > j, 0], {i, 1, n}, {j, 1, n}];
MFull = Table[Which[i ≥ j, M[[j, i]], i < j, M[[i, j]]], {i, 1, n}, {j, 1, n}];
GOEev = Sort[Eigenvalues[MFull]];
```

As mentioned in the article, the distribution of eigenvalues from a random matrix thins out toward the edges. We can illustrate this by plotting a histogram of the density of the eigenvalues, as shown below. The density is clearly much lower near the edges of the range of values. Indeed, the histogram looks very much like a semicircle centered at zero. Note that we need to load the Graphics package if it has no already been loaded.

```
<< Graphics`
```

```
GOEDensity = Histogram[GOEev, HistogramCategories → 25,
  HistogramScale → n, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"E", "ρ(E)"}, RotateLabel → False]
```

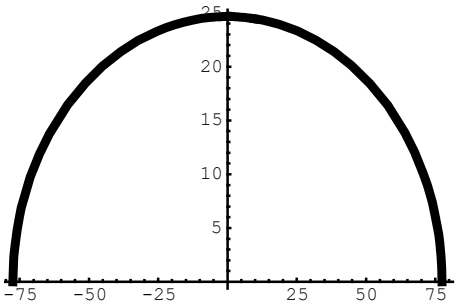



```
- Graphics -
```

Below we plot the theoretical density of eigenvalues known as Wigner's semicircle law (because the curve takes the shape of a semicircle centered at zero with radius $\sqrt{2N}$,where N is the dimension of

the random matrix (and hence the number of eigenvalues).

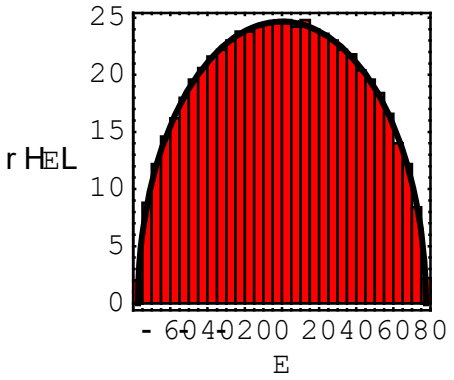
```
WSC = Plot[Sqrt[2 * n - x^2] / (Pi),
  {x, -Sqrt[2 * n], Sqrt[2 * n]}, PlotStyle → {Thickness[0.02]}]
```




```
- Graphics -
```

The plot below shows a comparison of Wigner's semicircle law to the computed eigenvalue density. The agreement is excellent.

```
Show[GOEDensity, WSC, AspectRatio → 1]
```




```
- Graphics -
```

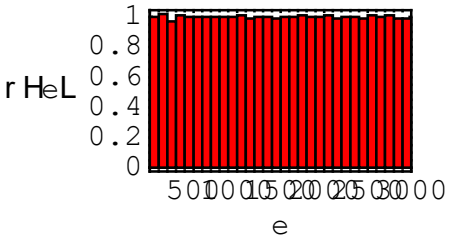
To analyze the level spacing statistics we must unfold the sequence of eigenvalues so that the unfolded eigenvalues are distributed uniformly on large scales (there will, of course, still be fluctuations about uniformity on small scales - that's what we want to focus our attention on). For a sequence that follows the Wigner semicircle law the unfolded eigenvalues are given by

$$e = \left(E\sqrt{2N - E^2} + 2N\sin^{-1}\left(E \Big/ \sqrt{2N}\right) + N\pi\right) \Big/ (2\pi)$$

The plot below shows the density of these unfolded eigenvalues. As expected, the density is essentially uniform. Note: watch out for eigenvalues with absolute value greater than $\sqrt{2N}$, which can occasiaonally occur, because these will lead to complex unfolded eigenvalues which will produce an error when

constructing the histogram.

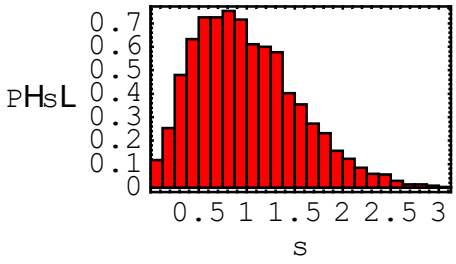
```
UGOEev = Table[
   (GOEev[[i]] * Sqrt[2 * n - GOEev[[i]]^2] + 2 * n * ArcSin[GOEev[[i]] / Sqrt[2 * n]] + n * Pi) /
    (2 * Pi), {i, 1, n}];
GOEDensity = Histogram[UGOEev, HistogramCategories → 25, HistogramScale → n, TextStyle →
    {"Times", FontSize → 16}, Frame → True, FrameLabel → {"e", "ρ(e)"}, RotateLabel → False]
```



- Graphics -

The code below calculates the spacings for the unfolded eigenvalues and generates a hisogram of the spacings.

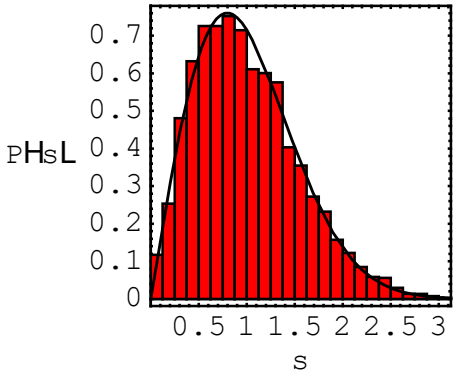
```
SpcGOEev = Table[UGOEev[[i + 1]] - UGOEev[[i]], {i, 1, n - 2}];
int = Table[0 + i / 8, {i, 0, 40}];
GOEHist = Histogram[SpcGOEev, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```



- Graphics -

Below we create a plot that compares the level spacing hisogram of our random real symmetric matrix to the GOE distribution. We see that that agreement is very good.

```
Show[GOEHist, GOE, AspectRatio → 1]
```




```
- Graphics -
```

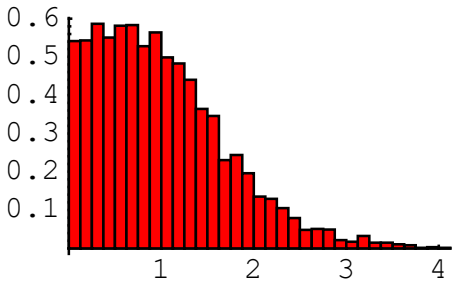
Random Matrix - 2 GOEs

In this section we generate a level spacing histogram for a combination of eigenvalues from two random real symmetric matrices. The code below generates the two random matrices, finds their eigenvalues, unfolds the eigenvalues so that they are distributed uniformly on a large scale, combines the unfolded eigenvalues into a single list, and sorts them. Note: this will take roughly twice as long as the calculations for a single GOE matrix (approximately 1800 seconds on a PowerMac G4 733 MHz). As with the GOE matrix above, watch out for eigenvalues with absolute value greater than $\sqrt{2N}$. Also, note that when the two unfolded sequences are mixed together the mean spacing of the resulting sequence will be 0.5 rather than 1, so the spacing must be rescaled to have mean spacing 1.

```
<< Statistics`ContinuousDistributions`
```

```
<< Graphics`
```

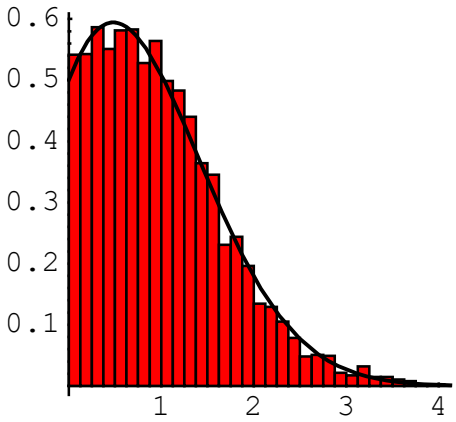
```
n = 3000;
M1 = Table[Which[i == j, Random[NormalDistribution[0, 1]], i < j,
    Random[NormalDistribution[0, 1 / Sqrt[2]]], i > j, 0], {i, 1, n}, {j, 1, n}];
M1Full = Table[Which[i ≥ j, M1[[j, i]], i < j, M1[[i, j]]], {i, 1, n}, {j, 1, n}];
M2 = Table[Which[i == j, Random[NormalDistribution[0, 1]], i < j,
    Random[NormalDistribution[0, 1 / Sqrt[2]]], i > j, 0], {i, 1, n}, {j, 1, n}];
M2Full = Table[Which[i ≥ j, M2[[j, i]], i < j, M2[[i, j]]], {i, 1, n}, {j, 1, n}];
EV1 = Eigenvalues[M1Full];
UEV1 =
  Table[(EV1[[i]] * Sqrt[2 * n - EV1[[i]]^2] + 2 * n * ArcSin[EV1[[i]] / Sqrt[2 * n]] + n * Pi) /
    (2 * Pi), {i, 1, n}];
EV2 = Eigenvalues[M2Full];
UEV2 =
  Table[(EV2[[i]] * Sqrt[2 * n - EV2[[i]]^2] + 2 * n * ArcSin[EV2[[i]] / Sqrt[2 * n]] + n * Pi) /
    (2 * Pi), {i, 1, n}];
UEV = Join[UEV1, UEV2];
SUev = Sort[UEV];

SpcUev = Table[SUev[[i + 1]] - SUev[[i]], {i, 1, 2 * n - 1}];
SpcUevNorm = SpcUev / Mean[SpcUev];
int = Table[0 + i / 8, {i, 0, 40}];
GOE2Hist = Histogram[SpcUevNorm, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16}, Frame → False]
```



- Graphics -

Below we compare the histogram to the theoretical distribution for a combination of two sets of data that have GOE spacings. The fit is very good.

```
Show[GOE2Hist, GOE2, AspectRatio → 1]
```



- Graphics -

Random Matrix - GUE

In this section we will generate a Gaussian-random complex Hermitian matrix, compute its eigenvalues, calculate the level spacings, and generate a histogram of these level spacings. This histogram closely matches the GUE distribution. Be aware that these calculations take some time (roughly 900 seconds on a PowerMac G4 733 MHz).

First we must construct the matrix. To do this we need to load the *Mathematica* package entitled "Continuous Distributions" that is contained within the Statistics package.

```
<< Statistics`ContinuousDistributions`
```

Now we construct and n by n random complex Hermitian matrix. Each matrix element is a complex number with random real and imaginary parts that are Gaussian distributed with a mean of zero and a standard deviation of one (except the diagonal elements which are real and have standard deviation of square root of 2). Here we generate a 3000 by 3000 matrix. Once the matrix is generated we can find its eigenvalues and sort them.

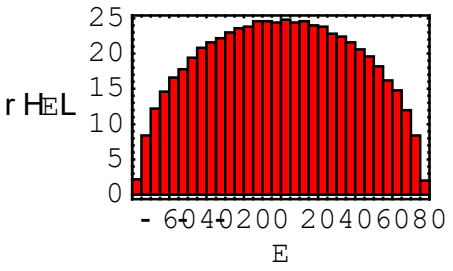
```
n = 3000;
MGUE = Table[Which[i == j, Random[NormalDistribution[0, 1]], i < j,
     Random[NormalDistribution[0, 1 / 2]] + I * Random[NormalDistribution[0, 1 / 2]],
     i > j, 0], {i, 1, n}, {j, 1, n}];
MGUEFull = Table[Which[i ≥ j, MGUE[[j, i]], i < j, Conjugate[MGUE[[i, j]]]],
    {i, 1, n}, {j, 1, n}];
GUEev = Sort[Eigenvalues[MGUEFull]];
```

For the GOE matrix above we found that the density of eigenvalues is not uniform, but thins out toward the edges. The plot below shows the density of eigenvalues of our GUE matrix. We see the same thinning of the eigenvalues toward the edges that we saw for the GOE matrix. Indeed, the eigenvalues of the GUE matrix follow the same Wigner semicircle law as did those of the GOE matrix. Note that we

need to load the Graphics package if it has not already been loaded.

```
<< Graphics`
```

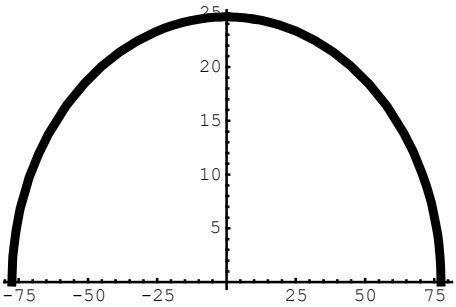
```
int = Table[-25 + 2 * i, {i, 0, 25}];
GUEHist = Histogram[GUEev, HistogramCategories → 25,
  HistogramScale → n, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"E", "ρ(E)"}, RotateLabel → False]
```




```
- Graphics -
```

Below we plot the theoretical density of eigenvalues known as Wigner's semicircle law (because the curve takes the shape of a semicircle centered at zero with radius $\sqrt{2N}$,where N is the dimension of the random matrix (and hence the number of eigenvalues).

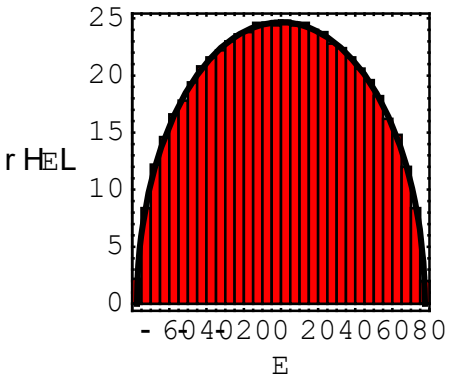
```
WSC = Plot[Sqrt[2 * n - x^2] / (Pi),
  {x, -Sqrt[2 * n], Sqrt[2 * n]}, PlotStyle → {Thickness[0.02]}]
```




```
- Graphics -
```

The plot below comapres the computed eigenvalue density to the Wigner semicircle law. The agreement is excellent.

```
Show[GUEHist, WSC, AspectRatio → 1]
```

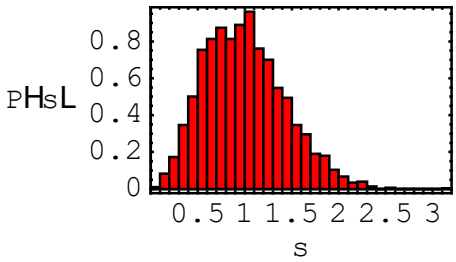


- Graphics -

The code below calculates the unfolded sequence of eigenvalues (using the same unfolding procedure as for the GOE matrix above) and then determines the spacings for these eigenvalues and generates a hisogram of the spacings. As with the GOE matrix above, watch out for eigenvalues with absolute value greater than $\sqrt{2N}$.

```
UGUEev = Table[
   (GUEev[[i]] * Sqrt[2 * n - GUEev[[i]]^2] + 2 * n * ArcSin[GUEev[[i]] / Sqrt[2 * n]] + n * Pi) /
    (2 * Pi), {i, 1, n}];
SpcGUEev = Table[UGUEev[[i + 1]] - UGUEev[[i]], {i, 1, n - 1}];
int = Table[0 + i / 10, {i, 0, 50}];
GUEHist = Histogram[SpcGUEev, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```

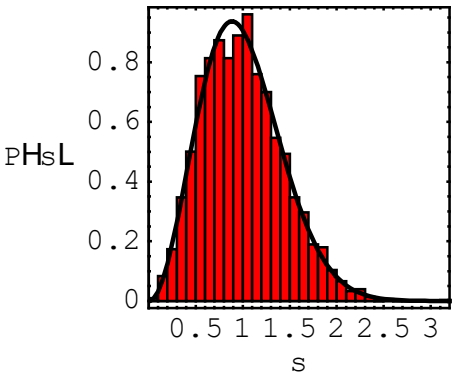
```
General::spell1 : Possible spelling error: new symbol name "UGUEev" is similar to existing symbol
   "GUEev". More...
```



- Graphics -

The plot below compares the level spacing hisogram of our random complex Hermitian matrix to the GUE distribution. We see that that agreement is very good.

```
Show[GUEHist, GUE, AspectRatio → 1]
```

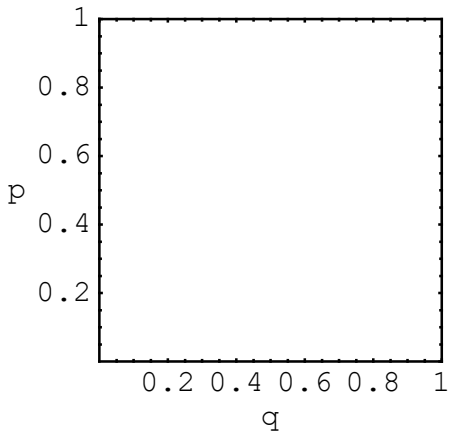



```
- Graphics -
```

Classical Standard Map

We begin with the code for plotting orbits of the standard map. This gives us an overal picture of the classical dynamics of this map. See "A computational approach to teaching conservative chaos" by T. Timberlake (Am. J. Phys. **72**, 1002-1007, August 2004) for more details. Note that the names of the variables change from (θ,r) in the aforementioned paper to (q,p) here.

```
K = 7; n = 400; q1 = 0; q2 = 0.5; np = 20;
f[{q_, p_}] :=
  N[{Mod[q + p - (K / (2 * Pi)) * Sin[2 * Pi * q], 1], Mod[p - (K / (2 * Pi)) * Sin[2 * Pi * q], 1]}];
MapData = Flatten[Table[Join[NestList[f, {q1, i / np}, n],
     NestList[f, {q2, i / np}, n]], {i, 0, np}], 1];
LP1 = ListPlot[MapData, PlotStyle → PointSize[0.005], PlotRange → {{0, 1}, {0, 1}},
  Frame → True, RotateLabel → False,
  TextStyle → {"Times", FontSize → 16}, FrameLabel → {"q", "p", "", ""},
  Axes → False, AspectRatio → 1]
```




```
- Graphics -
```

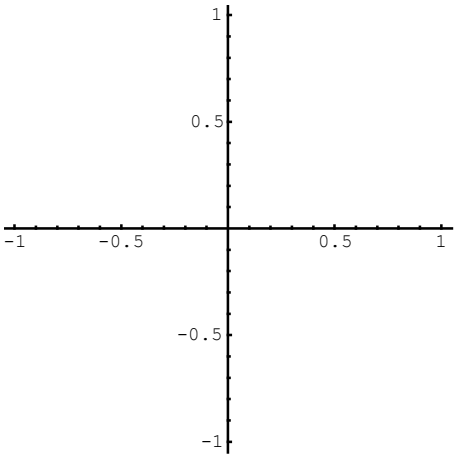
Quantum Standard Map

In this section we calculate the quasienergy eigenvalues for the quantum standard map. The code below generates the Floquet matrix (or quantum map operator) and then determines the eigenvalues and eigenvectors. We will need the eigenvectors later in order to separate the quasienergies for even and odd parity eigenstates. Note: this calculation takes a long time (roughly 1 hour on a PowerMac G4 733 MHz). You can shorten this time by reducing nq (the number of basis states / position grid pooints), but then the eigenvalues statistics will not be as good.

```
nq = 2000; k = 7;
U = Table[N[Exp[I * Pi * (i - j) ^2 / nq + I * k * nq * Cos[2 * Pi * j / nq] / (2 * Pi)] / Sqrt[nq]],
    {i, 0, nq - 1}, {j, 0, nq - 1}];
Eval = Eigenvalues[U];
Evec = Eigenvectors[U];
```

To verify that all of the eigenvalues have unit modulus (which they should, since the Floquet matrix is unitary) we can plot the eigenvalues in the complex plane as shown below. It is clear that all points lie on the unit circle in the complex plane.

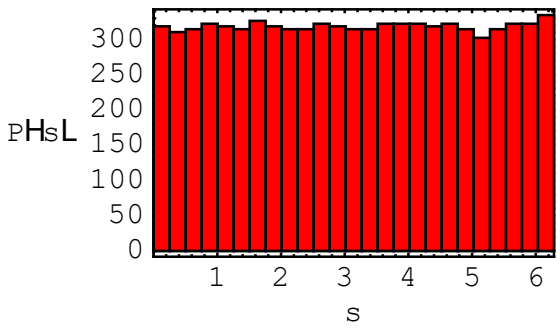
```
ListPlot[Table[{Re[Eval[[i]]], Im[Eval[[i]]]}, {i, 1, nq}], AspectRatio → 1]
```



- Graphics -

The code below calculates the quasienergies (from the eigenvalues of the Floquet matrix), sorts these quasienergies, computes the spacings between consecutive quasienergies, normalizes the spacings so that the mean spacing is 1, and then plots a hisogram of the normalized spacings. Since we are using all of the eigenvalues, what we expect to see is a mixture of 2 independent data sets (one for even parity states and one for odd parity states). If each data set follows Poisson statistics then the mixture will also be Poisson, but if each data set follows GOE statistics then the mixture will follow statistics for 2 sets of GOE data (see previous section).

```
<< Graphics`

Qnrg = Mod[Im[Log[Eval]], 2 * Pi];
SQnrg = Sort[Qnrg];
int = Table[N[i * 2 * Pi / 25], {i, 0, 25}];
QHist = Histogram[SQnrg, HistogramCategories → int,
  HistogramScale → nq, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```

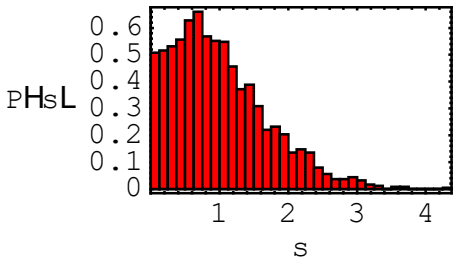


- Graphics -

```
Qnrg = Mod[-Im[Log[Eval]], 2 * Pi];
SQnrg = Sort[Qnrg];
QSpc = Join[Table[SQnrg[[i + 1]] - SQnrg[[i]], {i, 1, nq - 1}],
    {SQnrg[[1]] + 2 * Pi - SQnrg[[nq]]}];
QSpcNorm = QSpc / Mean[QSpc];
int = Table[0 + i / 8, {i, 0, 40}];
QHist2 = Histogram[QSpcNorm, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```

```
General::spell1 : Possible spelling error: new symbol name "SQnrg" is similar to existing symbol "Qnrg".
   More...
```

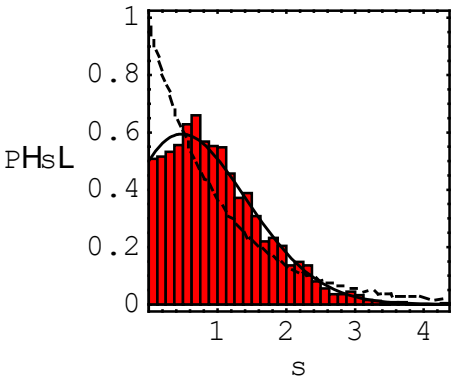
```
General::spell1 : Possible spelling error: new symbol name "QHist" is similar to existing symbol "MHist".
   More...
```




```
- Graphics -
```

The plot below compares the mixed level spacing histogram to the theoretical distributions for Poisson statistics and for 2 sets of GOE data. At high values of kick strength (large k) the histogram should match the 2 GOE curve. At low kick strengths there are degeneracies that lead to deviations from Poisson statistics.

```
Show[QHist2, Curves2, AspectRatio → 1]
```

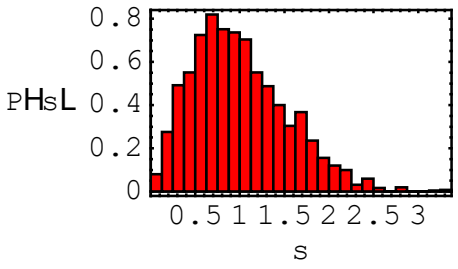


- Graphics -

The code below tests the parity of each of the eigenstates of the Floquet matrix, separates the eigenvalues of the even and odd states, computes the quasienergies for even and odd states, sorts each set of quasienergies, determines the spacings between consecutive quasienergies in each set, combines the spacing data from the two sets, normalizes the spacings so that the mean spacing is 1, and then generates a histogram of the level spacings. By calculating level spacings only between members of the same parity class we avoid the problem of mixing two independent data sets that we encountered above. Note that *Mathematica* labels the basis vectors from 1 to N instead of from 0 to N-1, as is done in the article. Also, the periodicity of the wavefunction is used in constructing the ParityTest function. Because of these two issues the sum in the ParityTest function looks a bit different from the sum for testing parity mentioned in the article, but they are equivalent.

```
ParityTest[i_] := (Sum[Abs[Evec[[i, j]] + Evec[[i, nq - j + 2]]]^2, {j, 2, nq / 2}] +
      Abs[2 * Evec[[i, 1]]]^2 + Abs[2 * Evec[[i, nq / 2 + 1]]]^2 > 1);
EvalEven = Eval[[Select[Range[1, nq], ParityTest[#] &]]];
EvalOdd = Eval[[Select[Range[1, nq], Not[ParityTest[#]] &]]];
```

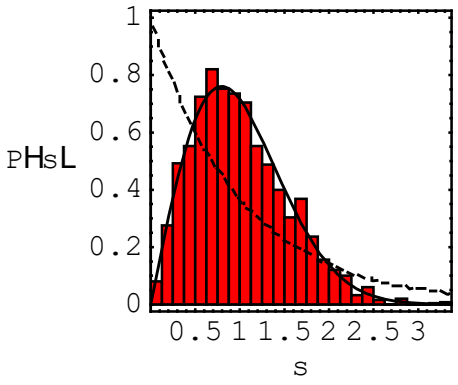
```
OQnrg = Mod[Im[Log[EvalOdd]], 2 * Pi];
SOQnrg = Sort[OQnrg];
OQSpc = Join[Table[SOQnrg[[i + 1]] - SOQnrg[[i]], {i, 1, Length[SOQnrg] - 1}],
    {SOQnrg[[1]] + 2 * Pi - SOQnrg[[Length[SOQnrg]]]}];
EQnrg = Mod[Im[Log[EvalEven]], 2 * Pi];
SEQnrg = Sort[EQnrg];
EQSpc = Join[Table[SEQnrg[[i + 1]] - SEQnrg[[i]], {i, 1, Length[SEQnrg] - 1}],
    {SEQnrg[[1]] + 2 * Pi - SEQnrg[[Length[SEQnrg]]]}];
AllQSpc = Join[EQSpc, OQSpc];
AllQSpcNorm = AllQSpc / Mean[AllQSpc];
int = Table[0 + i / 8, {i, 0, 40}];
QHist1 = Histogram[AllQSpcNorm, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```



- Graphics -

The plot below compares the separated level spacing histogram to the theoretical distributions for Poisson statistics and GOE statistics. For weak kicks (k ≈ 0) there are degeneracies that cause deviations from Poisson statistics, but the deviations are not as bad as for the mixed data and they mostly disappear as the kick strength is increased above 0 (while still remaining small). For large k (when the classical system is fully chaotic) the data match the GOE distribution.

```
Show[QHist1, Curves1, AspectRatio → 1]
```



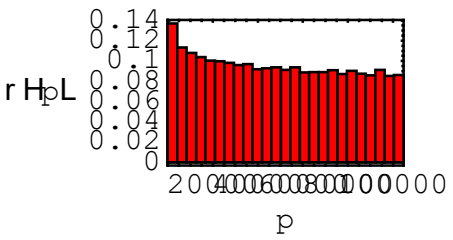

```
- Graphics -
```

Prime Numbers

The code below generates a list of the first 10,000 prime numbers and creates a histogram to show the density of the primes as a function of the size of the primes.

```
<< Graphics`

n = 10000;
P = Table[Prime[i], {i, 1, n}];
int = Table[P[[1]] + i * (P[[n]] - P[[1]]) / 25, {i, 0, 25}];
PDensity = Histogram[P, HistogramScale → n,
  HistogramCategories → int, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"p", "ρ(p)"}, RotateLabel → False]
```

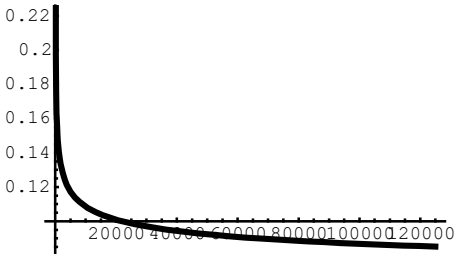
```
Histogram::rtail1 : Warning: One point from the right tail of the data, greater than or equal to 104729,
    is not included in histogram.
```




```
- Graphics -
```

Number theory tells us that the density of primes near the prime p should be 1/ln(p). Below we plot this reciprocal logarithm function.

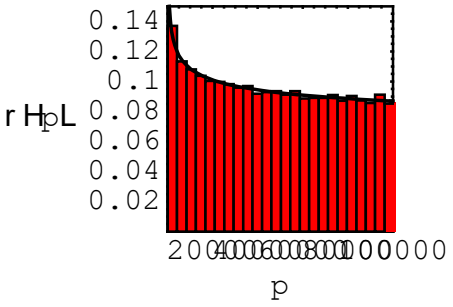
```
PD = Plot[1 / (Log[x]), {x, 2, 1.25 * 10^5}, PlotStyle → {Thickness[0.015]}]
```




```
- Graphics -
```

The plot below compares the computed density of primes with the theoretical (1/ln(p)) prediction. The agreement is very good.

```
Show[PDensity, PD, AspectRatio → 1, PlotRange → {{0, 10^5}, {0, 0.15}}]
```



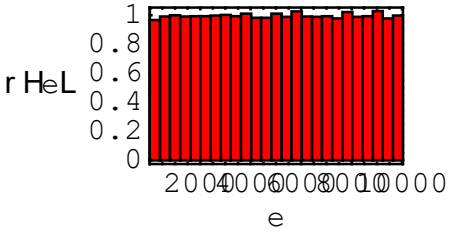

```
- Graphics -
```

To analyze the spacings between primes we must first account for their non-uniform density. To do this we must "unfold" the sequence of primes. For the primes, this means rescaling each prime p to e=Li(p), where Li(x) is the Log Integral function. The code below carries out this unfolding and then computes a histogram displaying the density of the unfolded primes. Note that the density is essentially uniform throughout.

```
n = 10000;
UP = Table[N[LogIntegral[Prime[i]]], {i, 1, n + 1}];
int = Table[UP[[1]] + i * (UP[[n]] - UP[[1]]) / 25, {i, 0, 25}];
UPHist = Histogram[UP, HistogramCategories → int,
  HistogramScale → n, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"e", "ρ(e)"}, RotateLabel → False]
```

```
General::spell1 : Possible spelling error: new symbol name "UPHist" is similar to existing symbol
   "PHist". More…
```

```
Histogram::rtail : Warning: 2 points from the right tail of the data, greater than or equal to
   10039.736337105796`, are not included in histogram.
```

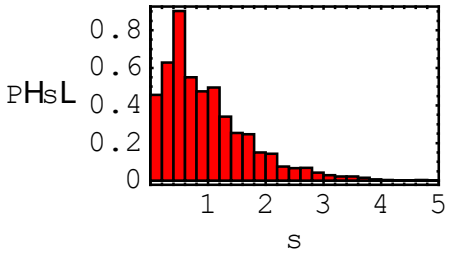



```
- Graphics -
```

The code in this section computes the spacings between the first 10,000 primes (unfolded so that the spacings are roughly uniform), rescales the spacings so that the mean is 1, and creates a histogram of the level spacings. Again, we need the Graphics package to create the histogram.

```
PSpacing = Table[UP[[i + 1]] - UP[[i]], {i, 1, n}];
PSpacingNorm = PSpacing / Mean[PSpacing];
int = Table[0 + i / 5, {i, 0, 25}];
PHist = Histogram[PSpacingNorm, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```

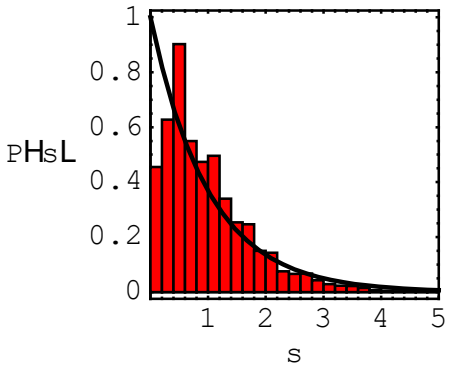
```
Histogram::rtail : Warning: 14 points from the right tail of the data, greater than or equal to 5, are
    not included in histogram.
```




```
- Graphics -
```

The plot below compares the spacing distribution for the first 10,000 unfolded primes to the Poisson distribution. While the prime spacings do seem to exhibit an exponential decay, there is a noticeable shortage of small spacings that leads to a discrepancy between the primes spacings and the Poisson distribution.

```
Show[PHist, Poisson, AspectRatio → 1]
```



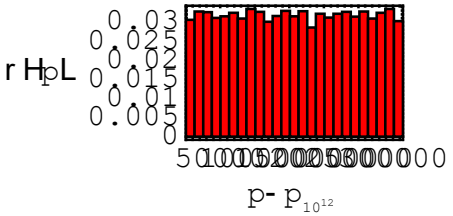

```
- Graphics -
```

The plot below shows the density of the primes between the 10^{12} + 1 st and the 10^{12} + 10^4 th primes. It is clear that the density of primes is fairly uniform over this range (which is consistent with the fact that the function 1/ln(x) does not change much over this range). Note that the drop-off in the last bar of the

histogram occurs because of the way the intervals are defined (i.e. the last interval extends well beyond the last prime number in the list).

```
n = 10000;
P2 = Table[Prime[10^12 + i] - Prime[10^12], {i, 1, n}];
int = Table[P2[[1]] + i * (P2[[n]] - P2[[1]]) / 25, {i, 0, 25}];
P2DHist = Histogram[P2, HistogramCategories → int,
  HistogramScale → n, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"p-p 10^12", "ρ(p)"}, RotateLabel → False]
```

```
Histogram::rtail1 : Warning: One point from the right tail of the data, greater than or equal to 314316,
   is not included in histogram.
```



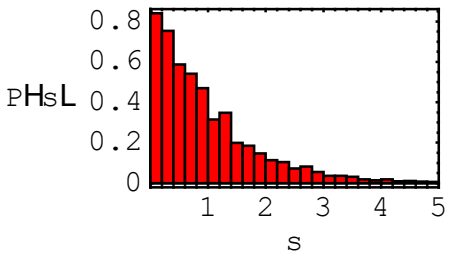

```
- Graphics -
```

Since the density of primes between the 10^{12} + 1 st and the 10^{12} + 10^4 th primesis roughly uniform, we do not need to unfold this sequence of primes in order to calculate the spacing statistics. The code below generates a histogram of the spacings between consecutive primes in this range, without unfolding (but still scaled so that the mean spacing is 1).

```
n = 10000;
P3 = Table[Prime[10^12 + i], {i, 1, n + 1}];
P3Spacing = Table[P3[[i + 1]] - P3[[i]], {i, 1, n}];
P3SpacingNorm = P3Spacing / Mean[P3Spacing];
int = Table[0 + i / 5, {i, 0, 25}];
P3Hist = Histogram[P3SpacingNorm, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```

```
General::spell1 : Possible spelling error: new symbol name "P3Hist" is similar to existing symbol
   "PHist". More…
```

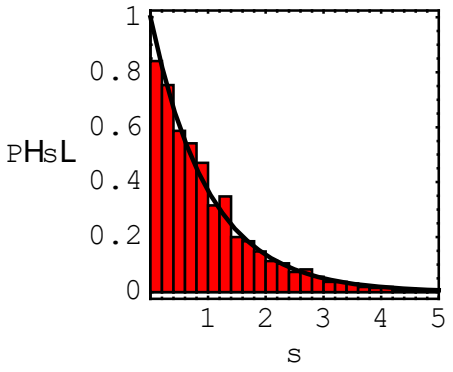
```
Histogram::rtail : Warning: 39 points from the right tail of the data, greater than or equal to 5, are
   not included in histogram.
```




```
- Graphics -
```

Finally we show a comparison of the Poisson distribution to the histogram of prime spacings near the 10^{12} th prime. We see that the agreement is very good in this case (although there is still a small shortage of small spacings).

```
Show[P3Hist, Poisson, AspectRatio → 1]
```




```
- Graphics -
```

Zeros of Riemann Zeta Function

In this section we will compute the level spacing distrubution for the imaginary parts of the nontrivial zeros of the Riemann zeta function (which I will refer to as "zeta zeros" from here on). Values of the zeta zeros have been computed by Andrew Odlyzko and generously made available on his website (http://www.dtc.umn.edu/~odlyzko/zeta_tables/). To use this data, go to Odlyzko's website and download the text files containing the zeta zeros. Name the files according to the names given on the website (zeros1, zeros3, etc.). You will need to cut out the explanations of the data given at the beginning of each file, so that what remains is just a list of numbers that can be read into *Mathematica*. Put the files into a convenient directory (such as the directory where this notebook is located) and then set the current directory to that directory using a command such as the one shown below (which will obviously need to be modified).

```
SetDirectory["/Users/timbertk/Documents/Research/RandomMatrix/"]
```

```
/Users/timbertk/Documents/Research/RandomMatrix
```

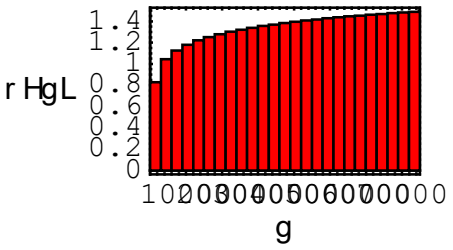
We'll need to load the Graphics package if that has not already been done.

```
<< Graphics`
```

The zeta zeros are not uniformly distributed, on average. The code below creates a histogram of the density of the first 100,000 zeta zeros.

```
ZetaZero1 = ReadList["zeros1", Real];
L1 = Length[ZetaZero1];
int = Table[ZetaZero1[[1]] + i * (ZetaZero1[[L1]] - ZetaZero1[[1]]) / 25, {i, 0, 25}];
ZHist1 = Histogram[ZetaZero1, HistogramCategories → int,
  HistogramScale → L1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"γ", "ρ(γ)"}, RotateLabel → False]
```

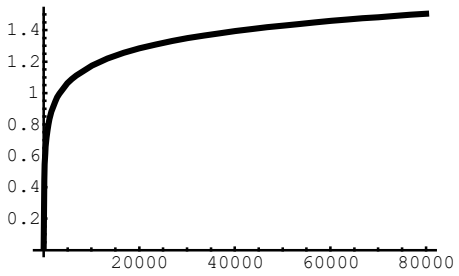
```
Histogram::rtail1 : Warning: One point from the right tail of the data, greater than or equal to
   74920.827498994`, is not included in histogram.
```




```
- Graphics -
```

Number theory tells us that if γ_n is the nth zeta zero, then the average density of zeta zeros near γ_n is $\ln(\gamma_n/(2\pi))/(2\pi)$. Below is a plot of the function $\ln(x/(2\pi))/(2\pi)$.

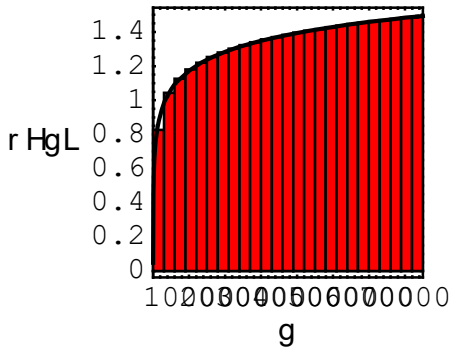
```
ZD = Plot[Log[x / (2 * Pi)] / (2 * Pi), {x, 7, 80000}, PlotStyle → {Thickness[0.015]}]
```




```
- Graphics -
```

The plot below compares the computed density of the zeta zeros to the theoretical prediction. The agreement is nearly perfect.

```
Show[ZHist1, ZD, AspectRatio → 1]
```




```
- Graphics -
```

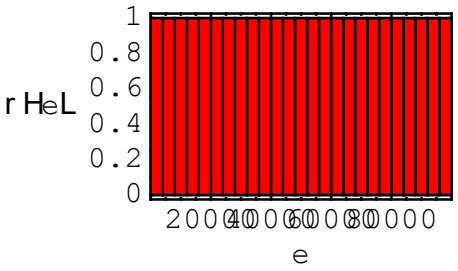
The unfolded zeta zeros are given by

```
e = γ (ln (γ / (2 π)) - 1) / (2 π).
```

The code below creates a histogram showing the density of the unfolded zeta zeros. We see that the density is uniform.

```
ZNorm =
  Table[ZetaZero1[[i]] * (Log[ZetaZero1[[i]] / (2 * Pi)] - 1) / (2 * Pi), {i, 2, 100000}];
LN = Length[ZNorm];
int = Table[ZNorm[[1]] + i * (ZNorm[[LN]] - ZNorm[[1]]) / 25, {i, 0, 25}];
ZHist1 = Histogram[ZNorm, HistogramCategories → int,
  HistogramScale → L1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"e", "ρ(e)"}, RotateLabel → False]
```

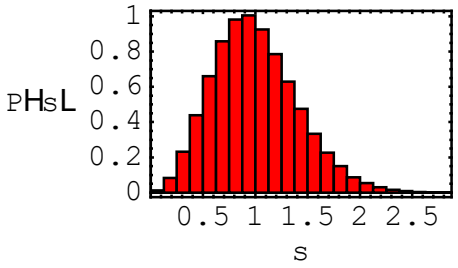
```
Histogram::rtail1 : Warning: One point from the right tail of the data, greater than or equal to
   99998.53008849114`, is not included in histogram.
```




```
- Graphics -
```

The code below creates the level spacing histogram for the first 100,000 unfolded zeta zeros.

```
ZSpacing1 = Table[ZNorm[[i + 1]] - ZNorm[[i]], {i, 1, L1 - 2}];
int = Table[0 + i / 8, {i, 0, 40}];
ZHist1 = Histogram[ZSpacing1, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```

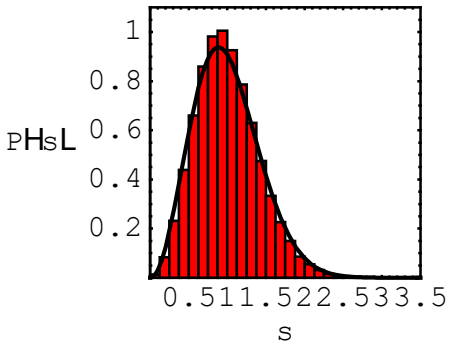



```
- Graphics -
```

The plot below compares the level spacing hisotgram for the first 100,000 zeta zeros to the theoretical

GUE distribution. The fit is not perfect, but it is good, but not perfect. In particular, there seems to be an excess of spacings near the peak of the distribution.

```
Show[ZHist1, GUE, AspectRatio → 1, PlotRange → {{0, 3.5}, {0, 1.1}}]
```

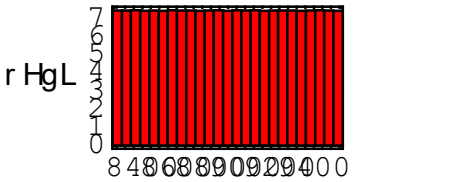



```
- Graphics -
```

Odlyzko's website also provides tables of zeta zeros that are far up the critical line. For example, the data file titled "zeros5" contains the zeta zeros $10^{22}+1$ to $10^{22}+10^4$. Because these numbers are so large the density of zeta zeros in this range is approximately uniform and we could avoid the unfolding altogether and simply rescale the average spacing to 1. The code below generates a histogram showing the density of the zeta zeros in this range. It is clear that the density is roughly uniform for this range of values. Note that the numbers listed in Odlyzko's table (file "zeros5" on his website) are actually the values of the zeta zeros minus 1370919909931995300000.

```
ZetaZero5 = ReadList["zeros5", Real];
L5 = Length[ZetaZero5];
int = Table[ZetaZero5[[1]] + i * (ZetaZero5[[L5]] - ZetaZero5[[1]]) / 25, {i, 0, 25}];
ZHist1 = Histogram[ZetaZero5, HistogramCategories → int,
  HistogramScale → L5, TextStyle → {"Times", FontSize → 16}, Frame → True,
  FrameLabel → {"γ-13709199099319953000000", "ρ(γ)"}, RotateLabel → False]
```

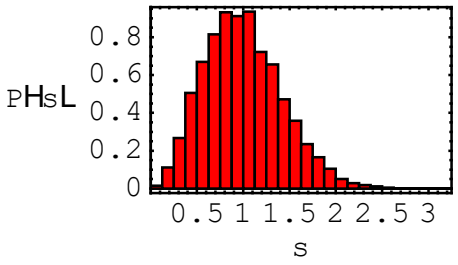
```
Histogram::rtail1 : Warning: One point from the right tail of the data, greater than or equal to
   9568.33538975`, is not included in histogram.
```




```
- Graphics -
```

The code below generates the level spacing histogram for the zeta zeros 10^{22}+1 to 10^{22}+10^4. There is no need to unfold this sequence since the zeta zeros are uniformly distributed in this range (see previous plot). All that is needed is to rescale the spacings so that the mean spacing is 1.

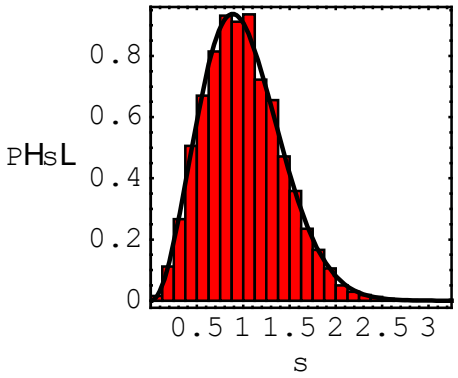
```
ZSpacing5 = Table[ZetaZero5[[i + 1]] - ZetaZero5[[i]], {i, 1, L5 - 1}];
ZNorm5 = ZSpacing5 / Mean[ZSpacing5];
int = Table[0 + i / 8, {i, 0, 40}];
ZHist5 = Histogram[ZNorm5, HistogramCategories → int,
  HistogramScale → 1, TextStyle → {"Times", FontSize → 16},
  Frame → True, FrameLabel → {"s", "P(s)"}, RotateLabel → False]
```



- Graphics -

The plot below compares the level spacing histogram for these zeta zeros that are high up the critical line to the theoretical GUE distribution. The fit is quite good, indicating that the spacings between the zeta zeros more closely follow GUE statistics as we go farther up the critical line. This implies that the limiting distribution (i.e. the distribution for the entire infinite set) for the spacings between zeta zeros is the GUE distribution.

```
Show[ZHist5, GUE, AspectRatio → 1]
```



- Graphics -