RMTool

A Random Matrix and Free Probability Calculator for MATLAB

Users Guide

Version 1.0

`http://www.mit.edu/~raj/rmtool`

N. Raj Rao
Department of EECS
Massachussetts Institute of Technology
Cambridge, MA 02139 - USA.
Email: raj@mit.edu

February 1, 2006

Contents

Chapter 1

Getting started with RMTool

RMTool is a free, third-party MATLAB toolbox for computing the limiting spectral measure of a large class of random matrices. The techniques behind it are based on the polynomial method (see arXiv: math.PR/060138) . Some applications are mentioned in the afore-mentioned paper.

In the next two sections, we will provide a quick overview of how to get started.

1.1 Systems Requirements and Installation Instructions

To install and run RMTool you need:

- MATLAB version 5 or later
- Symbolic Math Toolbox
- (optional) Extended Symbolic Math Toolbox.

RMTool can be easily run on a UNIX workstation, on a Windows PC desktop, or even a laptop. It heavily utilizes MATLAB's symbolic computation toolbox. An experienced user should be able to port the programs into MAPLE based on the mathematical principles outlined in the paper.

RMTool is available for free the GNU General Public License. The software and its users manual can be downloaded from `http://www.mit.edu/~raj/rmtool`. Once you download the zip file, you should extract its contents to the directory where you want to install RMTool. In UNIX, you max use the command:

```
unzip -U RMTool.nnn.zip -d your_dir
```

where `nnn` is the version number, and `your_dir` should be replaced by the directory of your choice. In Windows operation systems, you may use programs like Winzip to extract the files.

After this has been done, you must add the RMTool directory and its subdirectories to the MATLAB path. This is done in MATLAB by choosing the menus File → Set Path → Add Subdirectories ... and then typing the name of the RMTool main directory. This completes the RMTool installation.

1.2 Other Things You Need to Know

The directory in which you install RMTool contains subdirectories. Two of them are:

- `rmtool/docs`: containing this user manual, the original paper and license file
- `rmtool/demos`: containing several demo files.

The demo files in the second subdirectory above implement the examples in the paper and a few more.

Throughout this users manual, we use the `typewriter` typeface to denote MATLAB variables and functions, MATLAB commands that you should type, and results given by MATLAB.

You may send bug reports, comments, and suggestions to `raj@mit.edu`. Any feedback is greatly appreciated. Updates, including new features will be constantly added and posted at the URL indicated on the cover.

1.3 Quick start

After installing the package, try the following sequence of commands in MATLAB.

```
>> startRMTool
>> syms m z
>> b = wishartpol(0.5);
>> wishart_moments = Lmz2MomS(b,10)
>> [pdfinfo] = Lmz2pdf(b,[-0.05:0.01:5]);
>> figure(1); plot(pdfinfo.range,pdfinfo.density,'LineWidth',2);
```

The above sequence of commands generates the bivariate polynomial representation of the Marčenko-Pastur density and computes its first 10 moments, and the density. We can repeat this computation for the Wigner matrix by typing in the sequence of commands:

```
>> syms m z
>> b = wignerpol;
>> wigner_moments = Lmz2MomS(b,10)
>> [pdfinfo] = Lmz2pdf(b,[-4:0.01:4]);
>> figure(2); plot(pdfinfo.range,pdfinfo.density,'LineWidth',2);
```

Now try the following:

```
>> syms c
>> b = AplusB(wignerpol,wishartpol(c)); % Let c be symbolic
>> moments = Lmz2MomS(b)
```

The function `histw` has been provided to plot the normalized histogram of eigenvalues collected experimentally. Its use is illustrated below with an example where the limiting spectrum is predicted and compared to an experimental realization. (Type `help wishart` and `wigner` for documentation of these built-in functions.)

```
>> e = []; trials = 1000; n = 100;
>> for idx = 1 : trials,
   A = wigner(n); B = wishart(n,2*n);
   e = [e; real(eig(A+B))];
   end
>> [pdfinfo] = Lmz2pdf(AplusB(wignerpol,wishartpol(0.5)));
>> histw(e,40); hold on; plot(pdfinfo.range,pdfinfo.density,'red','LineWidth',2)
```

If everything works, then the red line should coincide with the histogram bars. Congratulations! You are now well on your way to using RMTool!

Chapter 2

Computing the spectral measure of random matrices

RMTool can compute the spectral measure of random matrix *models*. In other words, we do not actually generate the matrix. All the computations are done symbolically. Manipulating polynomials is at the heart of the software. Thus we have to learn how to represent probability measures as polynomial.

2.1 Polynomial Representation of Probability Measures

Polynomials in RMTool have representation as symbolic objects, using the MATLAB Symbolic Toolbox. Typically, a polynomial is created by first declaring its independent variables and then constructing it using the algebraic manipulations.

Generically we represent a probability measure by the bivariate polynomial that encodes its Stieltjes transform. The Stieltjes transform is defined as:

$$m(z) = \int \frac{1}{x - z} f(x) dx \qquad \text{for} \quad \Im z \neq 0.$$

Hence we represent probability measures by the variables m and z. In MATLAB we create a polynomial in m and z by first declaring them to be symbolic using the command

```
>> syms m z
```

Then we construct the polynomial $L(m, z)$ as follows:

```
>> L1 = numden(m-0.5/(1-z)-0.5/(2-z));
```

The above polynomial encodes the probability measure with atoms at 1 and 2 of equal weight. The command `numden` clears the denominator.

2.2 Manipulating the Polynomials

Polynomials such as the one created above can be manipulated with the usual operators: +,-,* However, since we are interested in manipulating the probability measures that they encode we do not need to worry about the usual algebraic operations involving the polynomials.

Instead, we use built-in functions to manipulate these polynomials. In doing so, we are really manipulating the probability measures encoded by these polynomials. The functions listed below can be used to this effect. The names are self-explanatory (see the paper for additional details). The polynomials can be manipulated using the function (with the arguments indicated). The arguments can themselves be symbolic.

Typing in `help invA` (for example) will provide you with some concrete examples to experiment with.

Function	Operation
`invA(L)`	Invert A
`shiftA(L,alpha)`	A + alpha * I
`scaleA(L,alpha)`	alpha * A
`mobiusA(L,p,q,r,s)`	(p*A+q*I)/(r*A+s*I)
`transposeA(L,c)`	If A = XX' then B=X'X, c = Size of A/Size of B
`squareA(L)`	A^2
`AtimesWish(L,c)`	A x Wishart with parameter c = Rows/Columns
`AgramWish(L,c,s)`	(X + s G)(X+s G)' with c = Rows/Columns
`AplusB(La,Lb)`	A + B (free addition)
`AtimesB(La,Lb)`	A x B (free multiplication)
`compressA(La,c)`	compress A by a factor of c (less than 1)

2.3 Extracting the density from the polynomial

Extracting the density from the polynomial in m and z involves determining the "right root". This is, in general, difficult so unless the right root can be easily spotted, we just return all the real and imaginary roots.

From a bivariate polynomial we can return the roots and some additional information about the density by typing the command:

```
>> xx = [xstart:xstep:xend];
>> [pdfinfo] = Lmz2pdf(L,xx);
```

The variable `pdfinfo` returned is a MATLAB data structure containing many fields. Type `help Lmz2pdf` for more details. We plot the (normalized) imaginary roots by typing the command:

```
>> plot(pdfinfo.range,pdfinfo.density,'.')
```

Somewhere among all the roots is the "right root". Often it is easy enough to spot it. The other fields will help you determine the region of support as described in the accompanying paper.

2.4 Extracting the moments from the polynomial

Even when the density cannot be extracted, the moments can often be. To do so there are two commands available `Lmz2MomS` and `Lmz2MomF`, where the suffix differentiates between the slow and fast implementations.

2.4.1 Slow implementation

As you have seen before you can type in the command:

```
>> Moments = Lmz2MomS(L,number_of_moments)
```

to enumerate the moments. Here L is the polynomial in m and z. This is a relatively slow algorithm.

2.4.2 Fast implementation

If you have the Extended Symbolic Toolbox then you can use the fast version by typing in the command:

```
>> Moments = Lmz2MomF(L,number_of_moments)
```

where L is the polynomial in m and z. When Maple finds more than one series expansion, the output returned will look non-sensical with the words `series` scattered. If that is the case, use Maple's `gfun` package as described in the accompanying paper to extract the correct moments.

To check if you have the Extended Symbolic Toolbox type in the command:

```
>> maple('help','gfun')
```

If a bunch of stuff is returned then you do have the Extended Toolbox; otherwise you do not. In general the slow implementation is competitive for simple examples; the fast implementation is invaluable for enumerating high degree moments of more complicated densities.

2.5 Some other useful commands

One of the great benefits of using symbolic software is being able to reproduce the result as often as one likes without errors. Of course, one might want to use the results generated using this package and put it in a technical publication (please acknowledge use of this software). There are a couple of built-in functions that are useful in this regard.

We can make an expression more legible using MATLAB's `pretty` command. The command `latex` produces a LaTeXcode of an expression while the command `TLmz` rewrites the bivariate polynomial as a matrix of coefficients. Try the following sequence of commands in MATLAB:

```
>> syms m z c
>> b1 = wishartpol(c)
>> b2 = wignerpol;
>> b3 = AtimesB(b1,b2)
>> pretty(b3)
>> moments = Lmz2MomS(b3,6)
>> pretty(moments)
>> latex(b3)
>> latex(moments)
>> TLmz(b3)
```


Acknowledgements

We thank Matthew Harding for valuable feedback and encouragement. The layout and organization of the content of this manual has been borrowed from the excellent documentation of SOSTOOLS.