Matrix Depot: An Extensible Test Matrix Collection for Julia

Zhang, Weijian and Higham, Nicholas J.

2015

MIMS EPrint: 2015.118

Manchester Institute for Mathematical Sciences
School of Mathematics
The University of Manchester

Reports available from: [http://eprints.maths.manchester.ac.uk/](http://eprints.maths.manchester.ac.uk/)
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097
Matrix Depot:
An Extensible Test Matrix Collection for Julia*

Weijian Zhang and Nicholas J. Higham†

December 21, 2015

Abstract

Matrix Depot is a Julia software package that provides easy access to a large and diverse collection of test matrices. Its novelty is threefold. First, it is extensible by the user, and so can be adapted to include the user’s own test problems. In doing so it facilitates experimentation and makes it easier to carry out reproducible research. Second, it amalgamates in a single framework three different types of matrix collections, comprising parametrized test matrices, regularization test problems, and real-life sparse matrix data. Third, it fully exploits the Julia language. It uses multiple dispatch to help provide a simple interface and, in particular, to allow matrices to be generated in any of the numeric data types supported by the language.

Keywords: Julia; software package; test matrices; test problems; matrix algorithm.

1 Introduction

In 1969 Gregory and Karney published a book of test matrices [14]. They stated that “In order to test the accuracy of computer programs for solving numerical problems, one needs numerical examples with known solutions. The aim of this monograph is to provide the reader with suitable examples for testing algorithms for finding the inverses, eigenvalues, and eigenvectors of matrix.” At that time it was common for journal papers to be devoted to introducing and analyzing a particular test matrix or class of matrices, examples being the papers of Clement [7] (in the first issue of SIAM Review), Pei [24] (occupying just a quarter of a page), and Gear [13].

Today, test matrices remain of great interest, but not for the same reasons as fifty years ago. Testing accuracy using problems with known solutions is less common because a reference solution correct to machine precision can usually be computed at higher precision without difficulty. The main uses of test matrices nowadays are for exploring the behavior of mathematical quantities (such as eigenvalue bounds) and for measuring the performance of one or more algorithms with respect to accuracy, stability, convergence rate, speed, or robustness.

Various collections of matrices have been made available in software. As well as giving easy access to matrices these collections have the advantage of facilitating reproducibility of experiments [10], whether by the same researcher months later or by different researchers.

---

*The work of the second author was supported by European Research Council Advanced Grant MATFUN (267526) and Engineering and Physical Sciences Research Council grant EP/I01912X/1.

†School of Mathematics, University of Manchester, Manchester, M13 9PL, England (weijian.zhang@manchester.ac.uk, nick.higham@manchester.ac.uk).
An early collection of parametrizable matrices was given by Higham [20] and made available in MATLAB form. The collection was later extended and distributed as a MATLAB toolbox [21]. Many of the matrices in the toolbox were subsequently incorporated into the MATLAB gallery function. Marques, Vömel, Demmel, and Parlett [23] present test matrices for tridiagonal eigenvalue problems (already recognized as important by Gregory and Karney, who devoted the last chapter of their book to such matrices). The Harwell–Boeing collection of sparse matrices [11] has been widely used, and is incorporated in the University of Florida Sparse Matrix Collection [8], [9], which contains over 2700 matrices from practical applications, including standard and generalized eigenvalue problems from [1]. Among other MATLAB toolboxes we mention the CONTEST toolbox [25], which produces adjacency matrices describing random networks, and the NLEVP collection of nonlinear eigenvalue problems [2].

The purpose of this work is to provide a test matrix collection for Julia [3], [4], a new dynamic programming language for technical computing. The collection, called Matrix Depot, exploits Julia’s multiple dispatch features to enable all matrices to be accessed by one simple interface. Moreover, Matrix Depot is extensible. Users can add matrices from the University of Florida Sparse Matrix Collection and Matrix Market; they can code new matrix generators and incorporate them into Matrix Depot; and they can define new groups of matrices that give easy access to subsets of matrices. The matrices can be generated in any appropriate numeric data type, such as

- floating-point types \( \text{Float16} \) (half precision: 16 bits), \( \text{Float32} \) (single precision: 32 bits), and \( \text{Float64} \) (double precision: 64 bits);
- integer types \( \text{Int32} \) (signed 32-bit integers), \( \text{UInt32} \) (unsigned 32-bit integers), \( \text{Int64} \) (signed 64-bit integers), and \( \text{UInt64} \) (unsigned 32-bit integers);
- \text{Complex}, where the real and imaginary parts are of any \text{Real} type (the same for both);
- \text{Rational} (ratio of integers); and
- arbitrary precision type \( \text{BigFloat} \) (with default precision 256 bits), which uses the GNU MPFR Library [12].

This paper is organized as follows. We start by giving a brief demonstration of Matrix Depot in Section 2. Then we explain the design and implementation of Matrix Depot in Section 3, giving details on how multiple dispatch is exploited, how the collection is stored, accessed, and documented, and how it can be extended. In Section 4 we describe the three classes of matrices in Matrix Depot: parametrized test matrices, regularization test problems, and real-life sparse matrix data. Concluding remarks are given in Section 5.

2 A Taste of Matrix Depot

To download Matrix Depot, in a Julia REPL (read-eval-print loop) run the command

```
> Pkg.add("MatrixDepot")
```

Then import Matrix Depot into the local scope.

```
> using MatrixDepot
```

Now the package is ready to be used. First, we find out what matrices are in Matrix Depot.
> matrixdepot()

Matrices:

1) baart  
5) chebspec  
9) companion  
13) forsythe  
17) gravity  
21) heat  
25) kahan  
29) magic  
33) oscillate  
37) phillips  
41) rando  
45) sampling  
49) tridiag  
53) wathen  

2) binomial  
6) chow  
10) deriv2  
14) foxgood  
18) grcar  
22) hilb  
26) kms  
30) minij  
34) parter  
38) poisson  
42) randsvd  
46) shaw  
50) triw  
54) wilkinson  

3) blur  
7) circul  
11) dingdong  
15) frank  
19) hadamard  
23) invhilb  
27) lehmer  
31) moler  
35) pascal  
39) prolate  
43) rohess  
47) spikes  
51) ursell  
55) wing  

4) cauchy  
8) clement  
12) fiedler  
16) golub  
20) hankel  
24) invol  
28) lotkin  
32) neumann  
36) pel  
40) randcorr  
44) rosser  
48) toeplitz  
52) vand  

All the matrices and groups in the collection are shown. It is also possible to obtain just the list of matrix names.

> matrixdepot("all")

55-element Array{ASCIIString,1}:

"baart"
"binomial"
"blur"
"cauchy"
"chebspec"
"chow"
"circul"
"clement"
...
"spikes"
"toeplitz"
"tridiag"
"triw"
"ursell"
"vand"
"wathen"
"wilkinson"
"wing"

Groups:

all data eigen ill-cond
inverse pos-def random regprob
sparse symmetric

Here, “...” denotes that we have omitted some of the output in order to save space. Next, we check the input options of the Hilbert matrix hilb.

> matrixdepot("hilb")

Hilbert matrix

The Hilbert matrix has (i,j) element 1/(i+j-1). It is notorious for being ill conditioned. It is symmetric positive definite and totally positive.
Input options:

* [type], dim: the dimension of the matrix.
* [type], row_dim, col_dim: the row and column dimensions.

Groups: ["inverse", "ill-cond", "symmetric", "pos-def"]

References:


Note that an optional first argument type can be given; it defaults to Float64. The string of equals signs on the third line in the output above is Markdown notation for a header. Julia interprets Markdown within documentation, though as we are using typewriter font for code examples here, we display the uninterpreted source. We generate a $4 \times 6$ Hilbert matrix with elements in the default double precision type and then in Rational type.

```julia
> matrixdepot("hilb", 4, 6)
4x6 Array{Float64,2}:
  1.0  0.5  0.333333  0.25  0.2  0.166667
  0.5  0.333333  0.25  0.2  0.166667  0.142857
  0.333333  0.25  0.2  0.166667  0.142857  0.125
  0.25  0.2  0.166667  0.142857  0.125  0.111111

> matrixdepot("hilb", Rational, 4, 6)
4x6 Array{Rational{Integer},2}:
  1//1  1//2  1//3  1//4  1//5  1//6
  1//2  1//3  1//4  1//5  1//6  1//7
  1//3  1//4  1//5  1//6  1//7  1//8
  1//4  1//5  1//6  1//7  1//8  1//9
```

A list of all the symmetric matrices in the collection is readily obtained.

```julia
> matrixdepot("symmetric")
21-element Array{ASCIIString,1}:
"cauchy"
"circul"
"clement"
"dingdong"
"fiedler"
"hankel"
"hilb"
"invhilb"
"kms"
"lehmer"
"minij"
"moler"
"oscillate"
```

4
Here, `symmetric` is one of several predefined groups, and multiple groups can be intersected. For example, the `for` loop below prints the smallest and largest eigenvalues of all the $4 \times 4$ matrices in Matrix Depot that are symmetric positive definite and (potentially) ill conditioned.

```matlab
> for name in matrixdepot("symmetric", "pos-def", "ill-cond")
    A = full(matrixdepot(name, 4))
    @printf "%9s: smallest eigval = %0.3e, largest eigval = %0.3e\n"
    name eigmin(A) eigmax(A)
end

cauchy: smallest eigval = 2.131e-05, largest eigval = 9.776e-01
hilb: smallest eigval = 9.670e-05, largest eigval = 1.500e+00
invhilb: smallest eigval = 6.666e-01, largest eigval = 1.034e+04
kms: smallest eigval = 3.750e-01, largest eigval = 2.086e+00
moler: smallest eigval = 3.336e-02, largest eigval = 5.122e+00
oscillate: smallest eigval = 1.490e-08, largest eigval = 1.000e+00
pascal: smallest eigval = 3.802e-02, largest eigval = 2.630e+01
pei: smallest eigval = 1.000e+00, largest eigval = 5.000e+00
tridiag: smallest eigval = 3.820e-01, largest eigval = 3.618e+00
```

Matrices can also be accessed by number within the alphabetical list of matrix names.

```matlab
> matrixdepot(2)
"binomial"

> matrixdepot(2:5)
4-element Array{ASCIIString,1}:
"binomial"
"cauchy"
"chebspec"
"chow"

> matrixdepot(15:20 , 5, 6, 1:3)
11-element Array{AbstractString,1}:
"frank"
"golub"
"gravity"
"grcar"
"hadamard"
"hankel"
"chebspec"
"chow"
"baart"
"binomial"
"blur"
```
Access by number provides a convenient way to run a test on subsets of matrices in the collection. However, the number assigned to a matrix may change if we include new matrices in the collection. In order to run tests in a way that is repeatable in the future it is best to group matrices into subsets using the macro @addgroup. For example, the following command will group test matrices frank, golub, gravity, grcar, hadamard, hankel, chebspec, chow, baart, binomial, and blur into test1.

```
> @addgroup test1 = matrixdepot(15:20, 5, 6, 1:3)
```

After reloading the package, we can run tests on these matrices using group test1. Since blur (an image deblurring test problem) generates a sparse matrix and the matrix 2-norm is currently not implemented for sparse matrices in Julia, we use full to convert the matrix to dense format.

```
> for name in matrixdepot("test1")
    A = full(matrixdepot(name, 4))
    @printf "%9s has 2-norm %0.3e \n" name norm(A)
end

baart has 2-norm 3.192e+00
binomial has 2-norm 4.576e+00
blur has 2-norm 8.298e-01
chebspec has 2-norm 6.474e+00
chow has 2-norm 3.414e+00
frank has 2-norm 7.624e+00
golub has 2-norm 2.050e+02
gravity has 2-norm 6.656e+00
grcar has 2-norm 2.562e+00
hadamard has 2-norm 2.000e+00
hankel has 2-norm 1.160e+01
```

To download the test matrix SNAP/web-Google from the University of Florida sparse matrix collection (see Section 4.3 for more details), we first download the data with

```
> matrixdepot("SNAP/web-Google", :get)
```

and then generate the matrix with

```
> matrixdepot("SNAP/web-Google", :r)
```

916428x916428 sparse matrix with 5105039 Float64 entries:

```
[11343 ,  1] =  1.0
[11928 ,  1] =  1.0
[15902 ,  1] =  1.0
[29547 ,  1] =  1.0
[30282 ,  1] =  1.0
[31301 ,  1] =  1.0
[38717 ,  1] =  1.0
...
[720325 , 916427] =  1.0
[772226 , 916427] =  1.0
[785097 , 916427] =  1.0
[788476 , 916427] =  1.0
[822938 , 916427] =  1.0
[833616 , 916427] =  1.0
[417498 , 916428] =  1.0
```
Note that the omission marked “…” was in this case automatically done by Julia based on the height of the terminal window. Matrices loaded in this way are inserted into the list of available matrices, and assigned a number. After downloading further matrices HB/1138_bus, HB/494_bus, and Bova/rma10 the list of matrices is as follows.

3 Package Design and Implementation

In this section we describe the design and implementation of Matrix Depot, focusing particularly on the novel aspects of exploitation of multiple dispatch, extensibility of the collection, and user-definable grouping of matrices.

3.1 Exploiting Multiple Dispatch

Matrix Depot makes use of multiple dispatch in Julia, an object-oriented paradigm in which the selection of a function implementation is based on the types of each argument of the function. The generic function matrixdepot has eight different methods, where each method itself is a function that handles a specific case. This is neater and more convenient than writing eight “case” statements, as is necessary in many other languages.
For example, the following two functions are used for accessing matrices by number and range respectively, where `matrix_name_list()` returns a list of matrix names. The second function calls the first function in the inner loop.

As a result, `matrixdepot` is a versatile function that can be used for a variety of purposes, including returning matrix information and generating matrices from various input parameters.

In the following example we see how multiple dispatch handles different numbers and types of arguments for the Cauchy matrix.

```
> matrixdepot("cauchy")

Cauchy matrix
=============

Given two vectors x and y, the (i,j) entry of the Cauchy matrix is \(1/(x[i] + y[j])\).

Input options:

* [type,] x, y: two vectors.
* [type,] x: a vector. y defaults to x.
* [type,] dim: the dimension of the matrix. x and y default to [1:dim;].

Groups: ["inverse", "ill-cond", "symmetric", "pos-def"]

References:

```
Multiple dispatch is also exploited in programming the matrices. For example, the Hilbert matrix is implemented as

```julia
function hilb{T}(::Type{T}, m::Integer, n::Integer)
    H = zeros(T, m, n)
    for j = 1:n, i = 1:m
        @inbounds H[i,j] = one(T)/ (i + j - one(T))
    end
    return H
end
hilb{T}(::Type{T}, n::Integer) = hilb(T, n, n)
hilb(args...) = hilb(Float64, args...)
```

The function `hilb` has three methods, which enable one to request, for example, `hilb(4,2)` for a 4 × 2 Hilbert matrix of type `Float64`, or simply (thanks to the final two lines) `hilb(4)` for a 4 × 4 Hilbert matrix of type `Float64`. The keyword `@inbounds` tells Julia to turn off bounds checking in the following expression, in order to speed up execution. Note that in Julia it is not necessary to vectorize code to achieve good performance [3].

All the matrices in Matrix Depot can be generated using the function call

```julia
matrixdepot("matrix_name", p1, p2, ...),
```

where `matrix_name` is the name of the test matrix, and `p1, p2, ...`, are input arguments depending on `matrix_name`. The help comments for each matrix can be viewed by calling function `matrixdepot("matrix_name")`. We can access the list of matrix names by number, range, or a mixture of numbers and range.

1. `matrixdepot(i)` returns the name of the `i`th matrix;
2. \texttt{matrixdepot(i:j)} returns the names of the \textit{i}th to \textit{j}th matrices, where \( i < j \);

3. \texttt{matrixdepot(i:j, k, m)} returns the names of the \textit{i}th, \((i + 1)\)st, \ldots, \textit{j}th, \textit{k}th, and \textit{m}th matrices.

### 3.2 Matrix Representation

Matrix names in Matrix Depot are represented by Julia strings. For example, the Cauchy matrix is represented by "cauchy". Matrix names and matrix groups are stored as hash tables (\texttt{Dict}). In particular, there is a hash table \texttt{matrixdict} that maps each matrix name to its underlying function and a hash table \texttt{matrixclass} that maps each group to its members.

The majority of parametrized matrices are dense matrices of type \texttt{Array{T,2}}, where \( T \) is the element type of the matrix. Variables of the \texttt{Array} type are stored in column-major order. A few matrices are stored as sparse matrices (see also \texttt{matrixdepot("sparse")}), in the Compressed Sparse Column (CSC) format; these include \texttt{neumann} (a singular matrix from the discrete Neumann problem) and \texttt{poisson} (a block tridiagonal matrix from Poisson’s equation). Tridiagonal matrices are stored in the built-in type \texttt{Tridiagonal} which is defined as follows.

```julia
immutable Tridiagonal{T} <: AbstractMatrix{T}
    dl :: Vector{T}  # sub-diagonal
    d :: Vector{T}   # diagonal
    du :: Vector{T}  # sup-diagonal
    du2 :: Vector{T} # supsup-diagonal for pivoting
end
```

### 3.3 Matrix Groups

A group is a subset of matrices in Matrix Depot. There are ten predefined groups, which are described in Table 1. Each group is represented by a string. For example, the group of random matrices is represented by "random". Matrices can be accessed by group names, as was illustrated in Section 1.

The macro \texttt{@addgroup} is used to add a new group of matrices to Matrix Depot and the macro \texttt{@rmgroup} removes an added group. All the predefined matrix groups are stored in the hash table \texttt{matrixclass}. The macro \texttt{@addgroup} essentially adds a new key-value combination to the hash table \texttt{usermatrixclass}. Using a separate hash table prevents the user from contaminating the predefined matrix groups.

Being able to create groups is a useful feature for reproducible research [10]. For example, if we have implemented algorithm \texttt{alg01} and we used \texttt{circul}, \texttt{minij}, and \texttt{grcar} as test matrices for \texttt{alg01}, we could type

```julia
> @addgroup alg01_group = ["circul", "minij", "grcar"]
```

This adds a new group to Matrix Depot (we need to reload the package to see the changes).

```julia
> matrixdepot()

Matrices:
   1) baart     2) binomial   3) blur     4) cauchy
   5) chebspec  6) chow      7) circul   8) clement
```
Table 1: Matrix properties.

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>All the matrices in the collection.</td>
</tr>
<tr>
<td>data</td>
<td>The matrix has been downloaded from the University of Florida Sparse Collection or the Matrix Market Collection.</td>
</tr>
<tr>
<td>eigen</td>
<td>Part of the eigensystem of the matrix is explicitly known.</td>
</tr>
<tr>
<td>ill-cond</td>
<td>The matrix is ill-conditioned for some parameter values.</td>
</tr>
<tr>
<td>inverse</td>
<td>The inverse of the matrix is known explicitly.</td>
</tr>
<tr>
<td>pos-def</td>
<td>The matrix is positive definite for some parameter values.</td>
</tr>
<tr>
<td>random</td>
<td>The matrix has random entries.</td>
</tr>
<tr>
<td>regprob</td>
<td>The output is a test problem for regularization methods.</td>
</tr>
<tr>
<td>sparse</td>
<td>The matrix is sparse.</td>
</tr>
<tr>
<td>symmetric</td>
<td>The matrix is symmetric for some parameter values.</td>
</tr>
</tbody>
</table>

| 9) deriv2 | 10) dingdong | 11) fiedler | 12) forsythe |
| 13) foxgood | 14) frank | 15) gravity | 16) grcar |
| 17) hadamard | 18) hankel | 19) heat | 20) hilb |
| 21) invhilb | 22) invol | 23) kahan | 24) kms |
| 25) lehmer | 26) lotkin | 27) magic | 28) minij |
| 29) moler | 30) neumann | 31) oscillate | 32) parter |
| 33) pascal | 34) pei | 35) phillips | 36) poisson |
| 37) prolate | 38) randcorr | 39) rando | 40) randsvd |
| 41) rohess | 42) rosser | 43) sampling | 44) shaw |
| 45) toeplitz | 46) tridiag | 47) triw | 48) vand |
| 49) wathen | 50) wilkinson | 51) wing | 52) Bova/rma10 |
| 53) HB/1138_bus | 54) HB/494_bus | 55) SNAP/web-Google |

Groups:
- all
- data
- eigen
- ill-cond
- inverse
- pos-def
- random
- regprob
- sparse
- symmetric
- alg01_group

We can then run alg01 on the test matrices by

```julia
> for name in matrixdepot(alg01_group)
    A = matrixdepot(name, n) # n is the dimension of the matrix.
    @printf "Test result for %9s is %0.3e" name alg01(A)
end
```

3.4 Adding New Matrix Generators

Generators are Julia functions that generate test matrices. When Matrix Depot is first loaded, a directory `myMatrixDepot` is created. It contains two files, `group.jl` and `generator.jl`, where `group.jl` is used for storing all the user defined groups (see Section 3.3) and `generator.jl` is used for storing generator declarations.

Julia packages are simply Git repositories\(^1\). The directory `myMatrixDepot` is untracked by Git,

\(^1\)Git is a free and open source distributed version control system.
so any local changes to files in myMatrixDepot do not make the MatrixDepot package “dirty”. In particular, all the newly defined groups or matrix generators will not be affected when we upgrade to a new version of Matrix Depot. Matrix Depot automatically loads all Julia files in myMatrixDepot. This feature allows a user to simply drop generator files into myMatrixDepot without worrying about how to link them to Matrix Depot.

A new generator is declared using the syntax `include_generator(FunctionName, "fname", f)`. This adds the new mapping "fname" → f to the hash table `matrixdict`, which we recall maps each matrix name to its underlying function. Matrix Depot will refer to function f using string "fname" so that we can call function f by `matrixdepot("fname"...)`.

For example, suppose we have the following Julia file `rand.jl`, which contains two generators `randsym` and `randorth` and we want to use them from Matrix Depot. The triple quotes in the file delimit the documentation for the functions.

```julia
""
random symmetric matrix
------------------------

Input options:
* n: the dimension of the matrix

""
function randsym(n)
    A = zeros(n, n)
    for j = 1:n
        for i = 1:j
            A[i,j] = randn()
            if i != j; A[j,i] = A[i,j] end
        end
    end
    return A
end

""
random orthogonal matrix
-------------------------

Input options:
* n: the dimension of the matrix

""
randorth(n) = qr(randn(n,n))[1]
```

We can copy the file `rand.jl` to the directory myMatrixDepot and write the following two lines to `generator.jl`.

```julia
include_generator(FunctionName, "randsym", randsym)
include_generator(FunctionName, "randorth", randorth)
```

This includes the functions `randsym` and `randorth` in Matrix Depot, as we can see by looking at the matrix list.

```plaintext
> matrixdepot()
```

```plaintext
12
```
The new generators can be used just like the built-in ones.

```latex
\begin{verbatim}
> matrixdepot("randsym")
random symmetric matrix

Input options:
* n: the dimension of the matrix

> matrixdepot("randsym", 4)
4x4 Array{Float64,2}:
  -0.00992523  0.174531  -1.73322  -0.765096
  0.174531    1.69308   0.269062  0.594058
  -1.73322   0.269062  -0.824277 -0.541458
  -0.765096  0.594058  -0.541458 -0.480428

> matrixdepot("randorth")
random orthogonal matrix

Input options:
* n: the dimension of the matrix

> A = matrixdepot("randorth", 4)
4x4 Array{Float64,2}:
  -0.233943  0.179893   0.563926  -0.771295
  -0.769649 -0.141938  -0.5807   -0.224235
  0.247165  0.832118  -0.449941 -0.20986
  -0.540204  0.594058  -0.541458 -0.480428

> A'*A - eye(4,4)
4x4 Array{Float64,2}:
  -2.22045e-16  1.66533e-16  -2.77556e-17  -1.66533e-16
\end{verbatim}
```
We can also add group information with the function `include_generator`. The following lines are put in `generator.jl`.

```julia
include_generator( Group, "random", randsym )
include_generator( Group, "random", randorth )
```

This adds the functions `randsym` and `randorth` to the group `random`, as we can see with the following query (after reloading the package).

```julia
> matrixdepot("random")
10-element Array{ASCIIString,1}:
"golub"
"oscillate"
"randcorr"
"rando"
"randorth"
"randsvd"
"randsym"
"rohess"
"rosser"
"wathen"
```

3.5 Documentation

The Matrix Depot documentation is created using the documentation generator Sphinx\(^2\) and is hosted at Read the Docs\(^3\). Its primary goals are to provide examples of usage of Matrix Depot and to give a brief summary of each matrix in the collection. Matrices are listed alphabetically with hyperlinks to the documentation for each matrix. Most parametrized matrices are presented with heat map plots, which are produced using the Winston package\(^4\), with the color range determined by the smallest and largest entries of the matrix. For example, Figure 1 shows how the Wathen matrix is documented in Matrix Depot.

4 The Matrices

We now describe the matrices that are provided with, or can be downloaded into, Matrix Depot.

4.1 Parametrized Matrices

In Matrix Depot v0.5.2, there are 55 parametrized matrices (including the regularization problems described in the next section), most of which originate from the Test Matrix Toolbox [21]. All these matrices can be generated as `matrixdepot("matrix_name", n)`, where `n` is the dimension of the matrix.

\(^2\)http://sphinx-doc.org/
\(^3\)http://matrixdepotjl.readthedocs.org
\(^4\)https://github.com/nolta/Winston.jl
The Wathen matrix is a sparse, symmetric positive, random matrix arising from the finite element method [wath87]. It is the consistent mass matrix for a regular $nx$-by-$ny$ grid of 8-node elements.


Figure 1: Documentation for the Wathen matrix
Many matrices can have more than one input parameter, and multiple dispatch provides a convenient mechanism for taking different actions for different argument types. For example, the `tridiag` function generates a tridiagonal matrix from vector arguments giving the subdiagonal, diagonal, and superdiagonal vectors, but a tridiagonal Toeplitz matrix can be obtained by supplying scalar arguments that specify the dimension of the matrix, the subdiagonal, the diagonal, and the superdiagonal. If a single, scalar argument `n` is supplied then an `n`-by-`n` tridiagonal Toeplitz matrix with subdiagonal and superdiagonal `-1` and diagonal `2` is constructed. This matrix arises in applying central differences to a second derivative operator, and the inverse and the condition number are known explicitly [22, sec. 28.5].

Here is an example of the different usages of `tridiag`.

```plaintext
> matrixdepot("tridiag")
  Tridiagonal Matrix
  -------------------

Construct a tridiagonal matrix of type Tridiagonal.

Input options:

* [type,] v1, v2, v3: v1 and v3 are vectors of subdiagonal and superdiagonal elements, respectively, and v2 is a vector of diagonal elements.

* [type,] dim, x, y, z: dim is the dimension of the matrix, x, y, z are scalars. x and z are the subdiagonal and superdiagonal elements, respectively, and y is the diagonal elements.

* [type,] dim: x = -1, y = 2, z = -1. This matrix is also known as the second difference matrix.

Groups: ["inverse", "ill-cond", "pos-def", "eigen"]

References:


> matrixdepot("tridiag", [2,5,6;], ones(4), [3,4,1;])
4x4 Tridiagonal{Float64}:
  1.0  3.0  0.0  0.0
  2.0  1.0  4.0  0.0
  0.0  5.0  1.0  1.0
  0.0  0.0  6.0  1.0

> matrixdepot("tridiag", 4, 5, 3, 1)
4x4 Tridiagonal{Float64}:
  3.0  1.0  0.0  0.0
  5.0  3.0  1.0  0.0
  0.0  5.0  3.0  1.0
  0.0  0.0  5.0  3.0

> matrixdepot("tridiag", Int, 4)
4x4 Tridiagonal{Int64}:
  2  -1  0  0
```

16
4.2 Test Problems for Regularization Methods

A mathematical problem is ill-posed if the solution is not unique or if an arbitrarily small perturbation of the data can cause an arbitrarily large change in the solution. Regularization methods are an important class of methods for dealing with such problems [16], [19]. One means of generating test problems for regularization methods is to discretize a given ill-posed problem.

Matrix Depot contains a group of regularization test problems derived from Hansen’s MATLAB Regularization Tools [15], [17], [18] that are mostly discretizations of Fredholm integral equations of the first kind:

\[
\int_0^1 K(s, t) f(t) \, dt = g(s), \quad 0 \leq s \leq 1.
\]

The regularization test problems form the group \texttt{regprob}.

```
> matrixdepot("regprob")
11-element Array{ASCIIString,1}:
  "baart"
  "blur"
  "deriv2"
  "foxgood"
  "gravity"
  "heat"
  "phillips"
  "shaw"
  "spikes"
  "ursell"
  "wing"
```

Each problem is a linear system \( Ax = b \) where the matrix \( A \) and vectors \( x \) and \( b \) are obtained by discretization (using quadrature or the Galerkin method) of \( K \), \( f \), and \( g \). By default, we generate only \( A \), which is an ill-conditioned matrix. The whole test problem will be generated if the parameter \texttt{matrixonly} is set to \texttt{false}, and in this case the output has type \texttt{RegProb}, which is defined as

```
immutable RegProb{T}
  A::AbstractMatrix{T} # matrix of interest
  b::AbstractVector{T} # right-hand side
  x::AbstractVector{T} # the solution to \( Ax = b \)
end
```

If \( r \) is a generated test problem, then \( r.A \), \( r.b \), and \( r.x \) are the matrix \( A \) and vector \( x \) and \( b \) respectively. For example, the test problem \texttt{wing} can be generated as follows.

```
> matrixdepot("wing")
    A Problem with a Discontinuous Solution
---------------------------------------------

Input options:
```
* [type,] n, t1, t2, [matrixonly]: the dimension of matrix is n. t1 and t2 are two real scalars such that 0 < t1 < t2 < 1. If matrixonly = false, the linear system A, b, x will be generated (matrixonly = true by default).

* [type,] n, [matrixonly]: t1 = 1/3 and t2 = 2/3.

Groups: ["regprob"]

References:


```plaintext
> A = matrixdepot("wing", 4)
4x4 Array{Float64,2}:
 0.031189 0.0921165 0.148804 0.198786
 0.0310674 0.0889342 0.134959 0.164156
 0.0309463 0.085862 0.122403 0.13556
 0.0308257 0.0828958 0.111014 0.111945

> r = matrixdepot("wing", 4, false)
Test problems for Regularization Methods
A:
4x4 Array{Float64,2}:
 0.031189 0.0921165 0.148804 0.198786
 0.0310674 0.0889342 0.134959 0.164156
 0.0309463 0.085862 0.122403 0.13556
 0.0308257 0.0828958 0.111014 0.111945
b:
4-element Array{Float64,1}:
 0.0804953
 0.0751385
 0.0701787
 0.0655842
x:
4-element Array{Float64,1}:
 0.0
 0.5
 0.5
 0.0

> r.x
4-element Array{Float64,1}:
 0.0
 0.5
 0.5
 0.0
```

4.3 Matrix Data from External Sources

Matrix Depot provides access to matrices from Matrix Market [5] and the University of Florida Sparse Matrix Collection [8], [9], both of which contain many matrices taken from applications.
In particular, these sources contain many large, sparse matrices.

Matrix Market and the University of Florida Sparse Matrix Collection both categorize matrices by application domain and the problem source and both provide matrices in Matrix Market Format [6]. These similarities allow us to design a generic interface for both collections. The symbol :get (or :g) is used for downloading matrices from both collections and the symbol :read (or :r) is used for reading in matrices already downloaded. Downloaded matrix data is stored on disk in the Matrix Market format and when read into Julia is stored in the type SparseMatrixCSC.

MatrixDepot.update() downloads the matrix name data files from the two web servers.

```
> MatrixDepot.update()
% Total  % Received  % Xfer  Average Speed  Time  Time  Time  Current
Download  Upload  Total  Spent  Left  Speed
100 1887k  0 1887k  0 0  97337 0 --:--:-- 0:00:19 --:--:--  472k

% Total  % Received  % Xfer  Average Speed  Time  Time  Time  Current
Download  Upload  Total  Spent  Left  Speed
100 41552  0 41552  0 0  4421 0 --:--:-- 0:00:09 --:--:--  41018
```

The UF Sparse Matrix Collection is divided into matrix groups and the group of a matrix forms part of the full name of the matrix [9]. For example, the full name of the matrix 1138_bus in the Harwell-Boeing Collection is HB/1138_bus.

```
> matrixdepot("HB/1138_bus", :get)
% Total  % Received  % Xfer  Average Speed  Time  Time  Time  Current
Download  Upload  Total  Spent  Left  Speed
100 19829  100 19829  0 0  2320 0 0:00:08 0:00:08 --:--:--  49572

> matrixdepot("HB/1138_bus", :read)
1138x1138 Symmetric{Float64, SparseMatrixCSC{Float64, Int64}}:
1474.78   0.0  0.0 ...  0.0  0.0  0.0  0.0
 0.0  9.13665  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0   69.6147  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
-9.01713  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0 ...  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0 -3.40599  0.0  0.0  0.0  0.0  0.0  0.0
  ...  ...  ...  ...  ...  ...  ...  ...
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0 ...  0.0  -24.3902  0.0  0.0
 0.0  0.0  0.0 ...  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  26.5639  0.0  0.0  0.0
 0.0  0.0  0.0 ...  0.0  46.1767  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  10000.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  117.647
```

Matrices from the University of Florida Sparse Matrix Collection are stored in MatrixDepot/data/uf and they are stored by group (to avoid duplicate names), i.e., one directory per group. Similarly, matrices from Matrix Market are stored in MatrixDepot/data/mm. Both directories are untracked by Git.
We can download a whole group of matrices from the University of Florida sparse matrix collection using the command `matrixdepot("group name/*", :get)`. The next example downloads all 67 matrices in the Gset group of matrices from random graphs (contributed by Y. Ye) then displays all the matrices in Matrix Depot, including the newly downloaded matrices.

```julia
> matrixdepot("Gset/*", :get)

Downloading all matrices in group Gset...

```

% Total   % Received  % Xferd  Average Speed   Time   Time  Time  Current
Dload  Upload   Total    Spent   Left  Speed
100 48083 100 48083  0     0  95388  0:00:01  0:00:01  0:00:01  96166
```

download: /home/weijian/.julia/v0.4/MatrixDepot/src/../data/uf/Gset/G1.tar.gz
```

G1/G1.mtx
%
Total   % Received  % Xferd  Average Speed   Time   Time  Time  Current
Dload  Upload   Total    Spent   Left  Speed
100 55180 100 55180  0     0  75318  0:00:01  0:00:01  0:00:01  75692
```

download: /home/weijian/.julia/v0.4/MatrixDepot/src/../data/uf/Gset/G10.tar.gz
```

G10/G10.mtx
%
Total   % Received  % Xferd  Average Speed   Time   Time  Time  Current
Dload  Upload   Total    Spent   Left  Speed
100 5926  100 5926  0     0  23126  0:00:00  0:00:00  0:00:00  23515
```

download: /home/weijian/.julia/v0.4/MatrixDepot/src/../data/uf/Gset/G11.tar.gz
```

G11/G11.mtx
%
Total   % Received  % Xferd  Average Speed   Time   Time  Time  Current
Dload  Upload   Total    Spent   Left  Speed
100 6349  100 6349  0     0  24223  0:00:00  0:00:00  0:00:00  24608
```

...

> matrixdepot()

Matrices:

1) baart
2) binomial
3) blur
4) cauchy
5) chebspec
6) chow
7) circul
8) clement
9) companion
10) deriv2
11) dingdong
12) fiedler
13) forsythe
14) foxgood
15) frank
16) golub
17) gravity
18) grcar
19) hadamard
20) hankel
21) heat
22) hilb
23) invhilb
24) invol
25) kahan
26) kms
27) lehmer
28) lotkin
29) magic
30) minij
31) moler
32) neumann
33) oscillate
34) parter
35) pascal
36) pei
37) phillips
38) poisson
39) prolates
40) ranvec
41) rando
42) randsvd
43) rohess
44) rosser
45) sampling
46) shaw
47) spikes
48) toeperr
49) tridiag
50) triw
51) ursell
52) vand
53) wathen
54) wilkinson
55) wing
56) Gset/G10
57) Gset/G11
58) Gset/G12
59) Gset/G13
60) Gset/G14
61) Gset/G15
62) Gset/G16
63) Gset/G17
64) Gset/G18
65) Gset/G19
66) Gset/G2
67) Gset/G20
68) Gset/G21
69) Gset/G22
70) Gset/G23
71) Gset/G24
72) Gset/G25
73) Gset/G26
74) Gset/G27
75) Gset/G28
76) Gset/G29
77) Gset/G2
78) Gset/G30
79) Gset/G31
80) Gset/G32
81) Gset/G33
82) Gset/G34
83) Gset/G35
84) Gset/G36
85) Gset/G37
86) Gset/G38
87) Gset/G39
88) Gset/G3
89) Gset/G40
90) Gset/G41
91) Gset/G42
92) Gset/G43
93) Gset/G44
94) Gset/G45
95) Gset/G46
96) Gset/G47
97) Gset/G48
98) Gset/G49
99) Gset/G4
100) Gset/G50
The full name of a matrix in Matrix Market comprises three parts: the collection name, the set name, and the matrix name. For example, the full name of the matrix BCSSTK14 in the set BCSSTRUC2 from the Harwell-Boeing Collection is Harwell-Boeing/bcsstruc2/bcsstk14. Note that both set name and matrix name are in lower case.

We recommend downloading matrices from the University of Florida sparse matrix collection when there is a choice, because almost every matrix from Matrix Market is included in it.

5 Concluding Remarks

Matrix Depot follows in the footsteps of earlier collections of matrices. Its novelty is threefold. First, it is extensible by the user, and so can be adapted to the user’s needs. In doing so it facilitates experimentation, and in particular makes it easier to do reproducible research. Second, it
combines three distinct test matrix collections, namely, parametrized test matrices, regularization test problems, and real-life sparse matrix data, in a single framework. Third, it fully exploits the Julia language. It uses multiple dispatch to help provide a simple interface and, in particular, to allow matrices to be generated in any of the numeric data types supported by the language. Matrix Depot therefore anticipates the development of intrinsic support in Julia for computations with BigFloat and other data types.

Matrix Depot has been in development since 2014. It is an open source project\textsuperscript{5} hosted on GitHub and is available under the MIT License. A first release was announced in December 2014. Matrix Depot v0.5.2 is the latest official release and consists of around 3,000 lines of source code, with test coverage of 98.91\% according to Codecov\textsuperscript{6}. From GitHub traffic analytics, we learn that Matrix Depot has 40 to 70 unique downloads (unique cloners) every month. Matrix Depot also benefits the development of other Julia packages. LightGraphs\textsuperscript{7}, an optimized graph package for Julia, for example, has embedded Matrix Depot as its database.

We built Matrix Depot to facilitate the development and testing of matrix (and other) algorithms in Julia. and we will continue to develop Matrix Depot by introducing new test matrices and integrating with other test collections.

Acknowledgements

The authors are grateful to Jiahao Chen (MIT), Stefan Güttel (The University of Manchester), and Tim Davis (Texas A&M University) for suggestions, and to Per Christian Hansen for allowing us to incorporate problems from Regularization Tools.

References


\textsuperscript{5}https://github.com/weijianzhang/MatrixDepot.jl
\textsuperscript{6}https://codecov.io/
\textsuperscript{7}https://github.com/JuliaGraphs/LightGraphs.jl


