Anymatrix: An Extensible MATLAB Matrix Collection. Users' Guide

Higham, Nicholas J. and Mikaitis, Mantas

2021

MIMS EPrint: **2021.15**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: 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

Anymatrix: An Extensible MATLAB Matrix Collection. Users' Guide

https://github.com/mmikaitis/anymatrix

Nicholas J. Higham* Mantas Mikaitis*

October 11, 2021

Abstract

This guide is for users and developers of the Anymatrix MATLAB toolbox, which provides an extensible collection of matrices organized into groups and sets and the ability to search for matrices by their properties. Full details of the usage of Anymatrix are given, along with details of the implementation.

Contents

*Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK (nick.higham@manchester.ac.uk, mantas.mikaitis@manchester.ac.uk).

1 Requirements

Anymatrix [11] requires MATLAB R2020b or newer, since it makes use of the `pattern` function, which was introduced in MATLAB R2020b. Anymatrix has been tested in versions up to R2021b.

2 Installation

To install `anymatrix` download it or clone the repository https://github.com/mmikaitis/anymatrix and add the root directory `anymatrix/` to the MATLAB search path. When it is first called in a session Anymatrix will add to the MATLAB search path all the `anymatrix` group directories so that it can access the matrices.

3 Release History

Anymatrix 1,0 was released on October 11, 2021 and contained 146 matrices.

4 Tutorial

Here we show some examples of how to use Anymatrix. Throughout this guide we denote omitted output by `[...]`.

```
% List all matrix IDs in the collection.
>> m = anymatrix('all')
m =
  146×1 cell array
    {'contest/baitsample'       }
    {'contest/curvature'        }
    {'contest/erdrey'           }
    {'contest/geo'              }
    {'contest/gilbert'          }
[...]

% List available groups.
>> g = anymatrix('groups')
g =
  7×1 cell array
    {'contest' }
    {'core'    }
    {'gallery' }
    {'hadamard'}
    {'matlab'  }
    {'nessie'  }
    {'regtools'}

% List the matrices in a particular group.
>> g = anymatrix('groups','core')
g =
  22×1 cell array
    {'core/augment'       }
    {'core/beta'          }
    {'core/blockhouse'    }
    {'core/dembo9'        }
```

```
    {'core/edelman27'     }
    {'core/fourier'       }
    {'core/gfpp'          }
    {'core/hessfull01'    }
[...]

% The Hadamard group contains two matrix generators.
>> g = anymatrix('groups','hadamard')
g =
  2×1 cell array
    {'hadamard/complex_hadamard'}
    {'hadamard/hadamard'        }

% Show help of the beta matrix in the core group.
>> anymatrix('help','core/beta')
 beta   Symmetric totally positive matrix of integers.
    beta(n) is an n-by-n symmetric totally positive matrix of integers.
    It is also infinitely divisible.
    [A,R] = beta(n) returns both the matrix and its explicitly constructed
    Cholesky factor R.

% Show the properties of the beta matrix.
>> p = anymatrix('properties','core/beta')
p =
  12×1 cell array
    {'built-in'            }
    {'infinitely divisible'}
    {'integer'             }
    {'nonnegative'         }
    {'positive'            }
    {'positive definite'   }
    {'real'                }
    {'scalable'            }
    {'square'              }
    {'symmetric'           }
    {'totally nonnegative' }
    {'totally positive'    }

% Generate the 5-by-5 beta matrix.
>> A = anymatrix('core/beta',5)
A =
     1     2     3     4     5
     2     6    12    20    30
     3    12    30    60   105
     4    20    60   140   280
     5    30   105   280   630
```

```
% List all matrices that have integer entries and are positive definite.
>> anymatrix('properties','integer and positive definite')
ans =
  7×1 cell array
    {'core/beta'     }
    {'core/wilson'   }
    {'gallery/gcdmat'}
    {'gallery/minij' }
    {'gallery/moler' }
    {'gallery/pei'   }
    {'matlab/pascal' }

% Look at the source code for a matrix.
>> type private/zielke_symm % Or edit private/zielke_symm

function [A,properties] = zielke_symm(n,a)
%ZIELKE_SYMM  Symmetric matrix of Zielke.
%  A = ZIELKE_SYMM(n,a) is a nonsymmetric matrix with elements
%  a, a-1, or a+1. Its inverse also has elements a, a-1, or a+1.
%  The default is a = 0.

%  Reference:
%  G. Zielke, Testmatrizen mit maximaler Konditionszahl, Computing 13,
%  33-54, 1974, section 3.

properties = {'symmetric', 'indefinite'};
if nargin == 0, A = []; return, end
if nargin == 1, a = 0; end;

A = rot90(zielke_nonsymm(n,a));
```

5 Anymatrix Usage

In this section we summarize the usage of Anymatrix. Full explanations of all the commands used are given in later sections.

5.1 Matrices

1. `[out1,...,outK] = anymatrix(matrix_id,in1,...,inN)`: generate the matrix with the specified matrix ID and parameters (if any) `in1` to `inN`. Some matrices return multiple output arguments.
2. `IDs = anymatrix('all')`: return all matrix IDs in the collection.
3. `anymatrix('help',matrix_id)` or `anymatrix(matrix_id,'help')`: list the help for a specified matrix.
4. `p = anymatrix('properties')`: list the recognized properties.
5. `p = anymatrix('properties',matrix_id)` or `anymatrix(matrix_id,'properties')`: list the properties of a specified matrix.

6. `M = anymatrix('properties',properties)`: list matrices having the properties specified by the Boolean expression in the string or character array `properties`.
7. `IDs = anymatrix('lookfor',pattern)`: return a list of matrix IDs whose help comments contain the string or character array `pattern` (see Section 11.3).

5.2 Short Forms of Commands

It is sufficient to type enough initial letters of an Anymatrix command to specify that command uniquely. In most cases a command is uniquely determined by its first letter. For example, `'properties'` can be specified by `'p'`. On the other hand, the short forms for `'scan'` (see Section 5.6) and `'sets'` (see Section 5.5) are `'sc'` and `'se'` respectively, since `'s'` would be ambiguous.

One can also define an alias for `anymatrix` to save typing; see Section 12.5.

5.3 Parameter Passing

Most of the functionality of Anymatrix is implemented with name–value pairs of input parameters, and the name and the value can be given in any order. Combined with the command/-function duality of MATLAB, this leads to four different but equivalent ways of specifying arguments, as illustrated by

```
anymatrix('core/fourier','help')
anymatrix('help','core/fourier')
anymatrix h core/fourier
anymatrix core/fourier h
```

MATLAB supports another syntax for specifying name–value pairs (introduced in MATLAB R2021a), which uses an equals sign. The following three invocations are equivalent:

```
anymatrix('groups','core')
anymatrix(groups = 'core')
anymatrix g core
```

Inputs to Anymatrix can be either character arrays or strings, as differentiated in MATLAB by single or double quotation marks surrounding the arguments. In this manual we use single quotes for all the arguments, and if we write "`string`", we mean either a string or a character array. Note that MATLAB command style function invocation interprets double quotes as part of the argument and uses the space character to separate different input arguments. Therefore in the command style function invocation, arguments to Anymatrix having spaces must be passed with single quotes.

5.4 Groups

1. `G = anymatrix('groups')`: return the available groups.
2. `anymatrix('contents',group_name)`: display `Contents.m` of the group with the name `group_name`.
3. `M = anymatrix('groups',group_name)`: return matrix IDs that belong to the group with the name `group_name`.
4. `anymatrix('groups',group_name,repository)`: clone or update a remote group stored in the specified repository.

5. `anymatrix('test')`: run tests of all groups, where tests are available. These tests are specific to each group; see Section 9.2. For matrix property testing see Section 9.1.
6. `anymatrix('test',group_name)`: run tests of a specified group if tests are available for it.

5.5 Sets

1. `S = anymatrix('sets')`: return the available sets.
2. `[M,out1,...,outN] = anymatrix('sets',set_name)`: return matrix IDs in M and first N matrices `out1` to `outN` in a specified set with name `set_name`.

5.6 General

1. `help anymatrix`: display help information.
2. `anymatrix('scan')`: force a scan of the file system (necessary after changes have been made to any functions that are part of Anymatrix).

5.7 Source Code of the Matrix Generators

The MATLAB function generating the matrix `core/beta` can be viewed using the commands `edit private/beta` or `type private/beta`. Note that the group name is not specified here, so if two different groups have a matrix with the same name `matrix_name` then these commands will display only one of the two files `matrix_name.m` (the one that corresponds to the directory that appears first on the path). One can always directly list or edit the file `anymatrix\group_name\private\matrix_name.m`.

6 Organization

The matrices in Anymatrix are organized into groups and sets.

6.1 Groups of Matrices

A *group* contains matrices that are related. The identifier of a matrix (matrix ID) combines the group name and the matrix name in the form `group_name/matrix_name`, where `group_name` is the name of the directory that the group is stored in and `matrix_name` is the name of the function that implements the matrix. The directory structure (described in Section 12.1) ensures that group names are unique, and so each matrix ID is unique.

Table 6.1: Matrices in the Anymatrix `contest` group of adjacency matrices from random network models.

Matrix	Description
`baitsample`	Bait and prey subsampling.
`curvature`	Curvatures (clustering coefficients).
`erdrey`	Erdős–Rényi model graph.
`geo`	Geometric random graph.
`gilbert`	Gilbert model graph.
`kleinberg`	Kleinberg model graph.
`lap`	Laplacian matrix (normalized or unnormalized).
`lockandkey`	Lock and key model graph.
`mht`	Mean hitting times.
`pagerank`	PageRank matrix.
`pathlength`	Minimum path lengths.
`pref`	Scale free random graph.
`renga`	Range dependent random graph.
`rewire`	Redirect edges.
`short`	Add shortcuts.
`smallw`	Small world random graph.
`sticky`	Stickiness model random graph.
`unisample`	Uniform subsampling.

Table 6.2: Matrices in the Anymatrix `core` group.

Matrix	Description
`augment`	Augmented system matrix.
`beta`	Symmetric totally positive matrix of integers.
`blockhouse`	Block Householder matrix.
`dembo9`	Symmetric Hankel matrix of order 9 and rank 5.
`edelman27`	Matrix for which det is computed as the wrong integer.
`fourier`	Fourier matrix.
`gfpp`	Matrix giving maximal growth for Gaussian elim. with pivoting.
`hessfull01`	Totally nonnegative binary lower Hessenberg Toeplitz matrix.
`hessmaxdet`	Upper Hessenberg matrix with maximal determinant.
`kms_nonsymm`	Nonsymmetric Kac–Murdock–Szego Toeplitz matrix.
`nilpot_triang`	Nilpotent triangular matrix.
`nilpot_tridiag`	Nilpotent tridiagonal matrix (sparse).
`rschur`	Upper quasi-triangular matrix.
`soules`	Soules matrix.
`symmstoch`	Symmetric stochastic matrix with given spectrum.
`totally_nonneg`	Nonsingular totally nonnegative matrix.
`triminsval01`	Optimal upper triangular binary Toeplitz matrix.
`vand`	Vandermonde matrix.
`vecperm`	Vec-permutation matrix.
`wilson`	Wilson matrix.
`zielke_nonsymm`	Nonsymmetric matrix of Zielke.
`zielke_symm`	Symmetric matrix of Zielke.

Table 6.3: Matrices in the Anymatrix `gallery` group.

Matrix	Description
`binomial`	Binomial matrix—multiple of involutory matrix.
`cauchy`	Cauchy matrix.
`chebspec`	Chebyshev spectral differentiation matrix.
`chebvand`	Vandermonde-like matrix for the Chebyshev polynomials.
`chow`	Chow matrix—a singular Toeplitz lower Hessenberg matrix.
`circul`	Circulant matrix.
`clement`	Clement matrix—tridiagonal with zero diagonal entries.
`compar`	Comparison matrices.
`condex`	Counterexamples to matrix condition number estimators.
`cycol`	Matrix whose columns repeat cyclically.
`dorr`	Dorr matrix—diagonally dominant, ill-conditioned, tridiagonal (one or three output arguments, `sparse`).
`dramadah`	Matrix of zeros and ones with large determinant or inverse.
`fiedler`	Fiedler matrix—symmetric.
`forsythe`	Forsythe matrix—a perturbed Jordan block.
`frank`	Frank matrix—ill-conditioned eigenvalues.
`gcdmat`	GCD matrix.
`gearmat`	Gear matrix.
`grcar`	Grcar matrix—a Toeplitz matrix with sensitive eigenvalues.
`hanowa`	Matrix whose eigenvalues lie on a vertical line in the complex plane.
`house`	Householder matrix (three output arguments).
`integerdata`	Array of arbitrary data from uniform distribution on specified range of integers.
`invhess`	Inverse of an upper Hessenberg matrix.
`invol`	Involutory matrix.
`ipjfact`	Hankel matrix with factorial elements (two output arguments).
`jordbloc`	Jordan block matrix.
`kahan`	Kahan matrix—upper trapezoidal.
`kms`	Kac–Murdock–Szego Toeplitz matrix.
`krylov`	Krylov matrix.
`lauchli`	Läuchli matrix—rectangular.
`lehmer`	Lehmer matrix—symmetric positive definite.
`leslie`	Leslie matrix.
`lesp`	Tridiagonal matrix with real, sensitive eigenvalues.
`lotkin`	Lotkin matrix.
`minij`	Symmetric positive definite matrix $\min(i, j)$.
`moler`	Moler matrix—symmetric positive definite.
`neumann`	Singular matrix from the discrete Neumann problem (`sparse`).
`normaldata`	Array of arbitrary data from standard normal distribution.

Table 6.3: (continued)

Matrix	Description
`orthog`	Orthogonal and nearly orthogonal matrices.
`parter`	Parter matrix—a Toeplitz matrix with singular values near π.
`pei`	Pei matrix.
`poisson`	Block tridiagonal matrix from Poisson's equation (`sparse`).
`prolate`	Prolate matrix—symmetric, ill-conditioned Toeplitz matrix.
`qmult`	Premultiply matrix by random orthogonal matrix.
`randcolu`	Random matrix with normalized columns and specified singular values.
`randcorr`	Random correlation matrix with specified eigenvalues.
`randhess`	Random, orthogonal upper Hessenberg matrix.
`randjorth`	Random J-orthogonal matrix.
`rando`	Random matrix with elements -1, 0, or 1.
`randsvd`	Random matrix with preassigned singular values and specified bandwidth.
`redheff`	Matrix of 0s and 1s of Redheffer.
`riemann`	Matrix associated with the Riemann hypothesis.
`ris`	Ris matrix—a symmetric Hankel matrix.
`sampling`	Nonsymmetric matrix with integer, ill-conditioned eigenvalues.
`smoke`	Smoke matrix—complex, with a "smoke ring" pseudospectrum.
`toeppd`	Symmetric positive definite Toeplitz matrix.
`toeppen`	Pentadiagonal Toeplitz matrix (`sparse`).
`tridiag`	Tridiagonal matrix (`sparse`).
`triw`	Upper triangular matrix discussed by Wilkinson and others.
`uniformdata`	Array of arbitrary data from standard uniform distribution.
`wathen`	Wathen matrix—a finite-element matrix (`sparse`, random entries).
`wilk`	Various specific matrices devised/discussed by Wilkinson (two output arguments).

Table 6.4: Matrices in the Anymatrix `hadamard` group.

Matrix	Description
`complex_hadamard`	Complex Hadamard matrices.
`hadamard`	Hadamard matrices of dimension up to 428.

Table 6.5: Matrices in the Anymatrix `matlab` group.

Matrix	Description
`compan`	Companion matrix.
`hadamard`	Hadamard matrix.
`hankel`	Hankel matrix.
`hilb`	Hilbert matrix.
`invhilb`	Inverse Hilbert matrix.
`magic`	Magic square.
`pascal`	Pascal matrix.
`rosser`	Classic symmetric eigenvalue test problem.
`spiral`	Matrix with elements in a rectangular spiral pattern.
`toeplitz`	Toeplitz matrix.
`vander`	Vandermonde matrix.
`wilkinson`	Wilkinson's eigenvalue test matrix.

Table 6.6: Matrices in the Anymatrix `nessie` group from real-life networks.

Matrix	Description
`benguela`	Adjacency matrix for South Africa marine ecosystem network.
`carcorr`	Correlation matrix associated with Scottish car travel times.
`eer`	Adjacency matrix for network of European economic regions.
`gene`	Adjacency matrix for a gene network.
`guppy`	Adjacency matrix for network of guppy social interactions.
`hexgrid`	Adjacency matrix for network from nuclear reactors.
`metabolite`	Adjacency matrix for metabolite network.
`p53`	Adjacency matrix for network of genes related to the oncogene p53.
`ppi`	Adjacency matrix for yeast protein-protein interaction network.
`spl0708a`	Adjacency matrix for network of Scottish football transfers.
`spl0708b`	Adjacency matrix for network of Scottish football transfers.
`spl0809`	Adjacency matrix for network of Scottish football transfers.
`traincorr`	Correlation matrix associated with Scottish train travel times.
`usshelf`	Adjacency matrix of network for Northeast US continental shelf.
`whiskycorr`	Correlation matrix associated with malt whisky tasting.
`whiskydist`	Matrix of Euclidean distances between whisky distilleries.

The built-in groups in Anymatrix are as follows.

- `contest`: the CONTEST toolbox of random matrices from networks listed in Table 6.1 [16].
- `core`: a miscellaneous selection of matrices, listed in Table 6.2. Several of these matrices are adapted from the Matrix Computation Toolbox [9].
- `gallery`: the MATLAB `gallery` matrices, listed in Table 6.3.
- `hadamard`: 659 Hadamard matrices of dimension up to 428, mostly from the collection of Sloan [14], together with some complex Hadamard matrices from [15]; see Table 6.4. A Hadamard matrix is a real $n \times n$ matrix with elements ± 1 satisfying $H^T H = nI$. A complex Hadamard matrix has entries of modulus 1 and satisfies $H^* H = nI$.
- `matlab`: matrices that are in MATLAB but are not part of `gallery`, namely the 12 matrices in Table 6.5.
- `nessie`: matrices from the NESSIE collection coming from real-life networks [17], listed in Table 6.6.
- `regtools`: matrices from regularization problems, listed in Table 6.7. The matrices are from the Regularization Tools toolbox[1] [6], [7]. The whole toolbox, including the regularization routines, is included in Anymatrix, but only the matrices are used. The `regtools` matrices have optional output arguments that generate a right-hand side b and, in some cases, a solution x (which are discretizations of the right-hand side and exact solution of the underlying Fredholm integral equation). We note that the problems in Regularization Tools are discretizations of mainly one-dimensional problems that are regarded as easy problems today. Discretizations of more challenging two-dimensional problems can be

[1] http://www.imm.dtu.dk/~pcha/Regutools/

Table 6.7: Matrices in the Anymatrix `regtools` group.

Matrix	Description
`baart`	Fredholm integral equation of the first kind.
`blur`	Image deblurring test problem with structured matrix.
`deriv2`	Computation of the second derivative.
`foxgood`	Severely ill-posed problem.
`gravity`	One-dimensional gravity surveying problem.
`heat`	Inverse heat equation.
`i_laplace`	Inverse Laplace transformation.
`parallax`	Stellar parallax problem with 28 fixed observations.
`phillips`	Phillips' "famous" test problem.
`regutm`	Test matrix for regularization methods.
`shaw`	One-dimensional image restoration problem.
`spikes`	Test problem with a "spiky" solution.
`tomo`	Two-dimensional tomography problem with sparse matrix.
`ursell`	Integral equation with no square integrable solution.
`wing`	Test problem with a discontinuous solution.

found in the IR Tools toolbox[2] [5] and the AIR Tools II toolbox[3] [8].

Other groups can be made available and downloaded on demand, as described in Section 10. These might include groups of more specialist interest, such as test matrices for a particular problem, counterexamples to conjectures, and matrices that (nearly) optimize particular quantities of interest over certain classes of matrices.

The authors will consider proposals from users to add matrices to the built-in groups and for additional groups to be made built-in.

6.2 Sets of Matrices

A matrix *set* is a user-created subset of all the matrices in an Anymatrix collection, from one or multiple groups, gathered into one named entity. This is useful to record and later reuse a set of matrices that have something in common, such as being used in an experiment. A set is defined by creating a text file in `anymatrix/sets`.

Here is an example usage of Anymatrix sets (see Section 5.5 for a full list of *set* commands):

```
% List all sets.
>> anymatrix('sets')
ans =
  1×1 cell array
    {'my_set'}

% Show matrices in a particular set.
>> S = anymatrix('sets','my_set')
ans =
```

[2] https://github.com/jnagy1/IRtools
[3] https://github.com/jakobsj/AIRToolsII

```
  5×2 cell array
    {'core/dembo9'  }    {1×1 missing   }
    {'core/augment' }    {'[1, 1; 2, 2]'}
    {'gallery/wilk' }    {[           3]}
    {'matlab/pascal'}    {'5, 1'        }
    {'matlab/pascal'}    {'5, 2'        }

% Generate the matrices (the first output is the same as above).
>> [S, D, A, W, P1, P2] = anymatrix('sets','my_set')
S =
  5×2 cell array
    {'core/dembo9'  }    {1×1 missing   }
    {'core/augment' }    {'[1, 1; 2, 2]'}
    {'gallery/wilk' }    {[           3]}
    {'matlab/pascal'}    {'5, 1'        }
    {'matlab/pascal'}    {'5, 2'        }

D =
    -1     1     1    -1    -1     1     1    -1    -1
     1     1    -1    -1     1     1    -1    -1     1
     1    -1    -1     1     1    -1    -1     1     1
    -1    -1     1     1    -1    -1     1     1    -1
    -1     1     1    -1    -1     1     1    -1    -1
     1     1    -1    -1     1     1    -1    -1     1
     1    -1    -1     1     1    -1    -1     1    -1
    -1    -1     1     1    -1    -1     1    -1     1
    -1     1     1    -1    -1     1    -1     1     1

A =
     1     0     1     1
     0     1     2     2
     1     2     0     0
     1     2     0     0

W =
    0.0000    0.9000   -0.4000
         0    0.9000   -0.4000
         0         0    0.0000

P1 =
     1     0     0     0     0
     1    -1     0     0     0
     1    -2     1     0     0
     1    -3     3    -1     0
     1    -4     6    -4     1
```

```
P2 =
     1     1     1     1     1
    -4    -3    -2    -1     0
     6     3     1     0     0
    -4    -1     0     0     0
     1     0     0     0     0
```

In this case `my_set` is the name of the set. The file (`anymatrix/sets/my_set.txt`) that describes this set, has the following contents (it can be viewed with `type my_set.txt`).

```
% This is an example anymatrix set.
% Comments start with a '%' symbol as the first character of the line.

% Following are declarations of some matrices in this set.

% Dembo9 matrix without any input arguments.
core/dembo9:

% Augment matrix with a matrix as an input argument.
core/augment: [1, 1; 2, 2]

% Wilkinson matrix with a single input.
gallery/wilk: 3

% Two pascal matrices with different input arguments.
matlab/pascal: 5, 1
matlab/pascal: 5, 2
```

A set entry comprises a matrix ID followed by a colon and a set of input arguments, and it is terminated by a new line character. Adding a new set into Anymatrix involves creating such a file and rescanning the filestore with `anymatrix('scan')`. Note that comments can only be made by starting a line with a percentage sign.

6.3 Uniqueness of Matrix Names

It is preferable that future matrices added to Anymatrix have distinct names, though this is not essential because two matrices with the same name in two different groups have a different ID.

7 Matrix Properties

Every matrix has a set of properties associated with it and it is possible to extract the identifiers of all matrices having a specified set of properties.

For the most part a matrix is assigned a property if it has that property for the default values of the parameters. An exception is "rectangular", which is assigned if the matrix is rectangular for some input arguments, and if the default shape is square then the matrix is given both the "square" and "rectangular" properties. Examples of matrices having both properties are `gallery/krylov` and `gallery/randsvd`. Likewise a matrix can be "real", "complex", or both.

7.1 Available Properties

Anymatrix recognizes the properties given in Table 7.1 (this list is defined in the file `anymatrix/prop_list.m`). The properties are not case-sensitive. While the hyphenations shown in the table are preferred, in specifying any property a hyphen and a space are treated as equivalent, so the two calls

```
anymatrix('properties','ill conditioned')
anymatrix('properties','ill-conditioned')
```

have the same effect. For definitions of the properties not defined in the table see linear algebra or numerical linear algebra textbooks.

Table 7.1: Matrix properties.

Property	Comments
banded	
binary	Elements 0 or 1.
block Toeplitz	
built-in	Included in the base Anymatrix collection.
complex	
correlation	Symmetric positive (semi)definite with unit diagonal.
defective	
diagonally dominant	
eigenvalues	Part of the eigensystem is known explicitly.
fixed size	The matrix size is fixed at one or a finite set of values.
Hankel	Constant along the antidiagonals.
Hermitian	
Hessenberg	
idempotent	
ill conditioned	The matrix is ill conditioned for some parameter values.
indefinite	
infinitely divisible	A is symmetric positive semidefinite with nonnegative entries and `A.^r` is positive semidefinite for all $r \geq 0$ [2].
integer	All integer entries.
inverse	The inverse of the matrix is known explicitly.
involutory	$A^2 = I$
M-matrix	
nilpotent	$A^k = 0$ for some positive integer k.
nonnegative	All nonnegative entries.
normal	Normal, but not symmetric, Hermitian, orthogonal, or unitary.
orthogonal	
parameter-dependent	Matrix depends on one or more parameters other than the dimension.
permutation	

Table 7.1: (continued)

Property	Comments
positive	All positive entries.
positive definite	Applies for both Hermitian and symmetric. Includes semidefinite. Search for positive definite and rank deficient to locate singular semidefinite matrices.
pseudo-orthogonal	Also known as J-orthogonal [10].
random	Random numbers are used in the construction.
rank deficient	
real	
rectangular	
scalable	The problem dimension (or a function of it) is a parameter. Not every dimension is necessarily supported.
singular values	Part of the singular value decomposition is known explicitly.
skew-Hermitian	
skew-symmetric	
sparse	The matrix is stored in the MATLAB sparse format.
square	
stochastic	Nonnegative with unit row sums.
symmetric	
Toeplitz	Constant along the diagonals.
totally nonnegative	Every submatrix has nonnegative determinant.
totally positive	Every submatrix has positive determinant.
triangular	
tridiagonal	
unimodular	Integer entries and determinant ± 1.
unitary	

All matrices that come with the base Anymatrix collection automatically have the property `built-in` assigned.

We do not regard the properties defined for real matrices as being included in the corresponding ones for complex matrices, thus "symmetric" is not included in "Hermitian" and "orthogonal symmetric" is not included in "unitary".

In Anymatrix properties are defined within the program files defining the matrices or groups and are read in when Anymatrix initializes itself on the first call in a MATLAB session (see the following section). Anymatrix provides a warning if a matrix is detected to have a property not contained in Table 7.1.

7.2 Specifying Properties of Matrices

Properties in `anymatrix` can be specified in one of two ways.

1. The function that defines the matrix can contain an assignment such as

```
properties = {'symmetric', 'positive definite', 'positive', ...
              'integer', 'ill conditioned', 'unimodular',  ...
              'square', 'real', 'fixed size'};
```

and return `properties` in its final output argument when called with no arguments. Anymatrix will detect the number of output arguments using `nargout`, hence (as it is a requirement of `nargout`) there must not be a variable number of output arguments. See Section 8 for an example of a matrix-generating function.

2. An M-file `am_properties.m` in the `private` directory containing the group stores the properties for some or all of the matrices in the group. For example, `am_properties.m` might contain

```
properties = ...
{'wilson', 'symmetric', 'positive definite', 'positive', 'integer', ...
           'ill conditioned', 'unimodular', 'fixed size';
 'hessfull01', 'hessenberg', 'toeplitz', 'binary', ...
               'totally nonnegative', 'rank deficient', 'eigenvalues';
};
```

to define properties for the `wilson` and `hessfull01` matrices.

Anymatrix allows both possibilities: if a matrix does not have an entry in `am_properties.m` (or the file `am_properties.m` does not exist in the group) then the properties are obtained by calling the matrix-generating function. The use of `am_properties.m` makes it easier to incorporate an existing group of matrices, as just one file needs to be edited. For example, for the built-in `gallery` and `matlab` groups we did not modify the corresponding M-files (which are part of MATLAB) but instead wrote an `am_properties.m` function for each group.

Empty curly braces mark no properties, as in the examples

```
properties = {}
```

and

```
properties = {'compar', {}};
```

correponding to 1 and 2 above.

Note that the properties of a matrix must be given either in `am_properties.m` or in the function itself. If neither is provided, the function is assumed to be a support file; see Section 11.2.

7.3 Property Maps

When one property is contained in another, only the most specific property needs to be provided, as defined in Table 7.2. The structure that tells `anymatrix` about these relationships between the properties is defined in `anymatrix/prop_map.m`. Note that the "positive definite" property does not map to another property. To specify a symmetric positive definite matrix specify "symmetric" and "positive definite" and likewise to specify a Hermitian positive definite matrix specify "hermitian" and "positive definite".

An inverted map (Table 7.3) that describes properties implied by the absence of some other properties is in `anymatrix/inv_prop_map.m`.

If either of the property maps is updated, the command `anymatrix('scan')` should be issued in order for Anymatrix to update itself.

Table 7.2: Property map implemented in `prop_map.m`. Properties in the first column are added if any of the more specific properties in the same row in the right column appear in the pre-defined matrix properties.

Added property	Subproperties
banded	tridiagonal
binary	permutation
integer	binary
nonnegative	binary, positive, stochastic, totally nonnegative
orthogonal	permutation
positive	totally positive
symmetric	correlation, hankel
positive definite	correlation
totally nonnegative	totally positive

Table 7.3: Inverted property map implemented in `inv_prop_map.m`. Properties in the first column are added if the corresponding property in the same row on the right column does not appear in the pre-defined matrix properties.

Added property	Subproperties
real	complex
scalable	fixed size
square	rectangular

7.4 Searching for Matrices by their Properties

Anymatrix provides functionality to search for matrices that have specified properties. The search is performed with the command `'properties'` (or a shorthand such as `'p'` or `'prop'`) using a search string that can contain the Boolean operators `and` and `or`. For example:

```
>> anymatrix properties unimodular
ans =
  2×1 cell array
    {'core/wilson'     }
    {'gallery/dramadah'}
```

The search string can contain Boolean expressions, as in the examples

- `'random and rectangular'`,
- `'not random'`,
- `'random and not rectangular'`.

Brackets can be used to specify precedence. Example:

```
>> M = anymatrix('p','tridiagonal and (symmetric or not positive)')
M =
  7×1 cell array
    {'core/nilpot_tridiag'}
    {'gallery/clement'    }
    {'gallery/dorr'       }
    {'gallery/lesp'       }
    {'gallery/tridiag'    }
    {'gallery/wilk'       }
    {'matlab/wilkinson'   }

>> anymatrix('p','unimodular and not symmetric')
ans =
  1×1 cell array
    {'gallery/dramadah'}
```

8 Structure of the Matrix-Generating Functions

As an example of a matrix-generating function, here are the contents of the file `core/private/vecperm.m`, defining the matrix `core/vecperm`.

```
function [P,properties] = vecperm(m,n)
%VECPERM    Vec-permutation matrix.
%   P = VECPERM(M,N) is the vec-permutation matrix, an M*N-by-M*N
%   permutation matrix with the property that if A is M-by-N then
%   vec(A) = P*vec(A').  If N is omitted, it defaults to M.
%   The vec-permutation matrix is also known as the commutation matrix.

%   P is formed by taking every n'th row from EYE(M*N), starting with
%   the first and working down; see p. 277 of the reference.
```

```
%   Reference:
%   H. V. Henderson and S. R. Searle, The vec-permutation matrix,
%   the vec operator and Kronecker products: A review, Linear
%   Multilinear Algebra, 9 (1981), pp. 271-288.

properties = {'orthogonal', 'permutation'};
if nargin == 0, P = []; return, end
if nargin == 1, n = m; end

P = zeros(m*n);
I = eye(m*n);

k = 1;
for i=1:n
    for j=i:n:m*n
        P(k,:) = I(j,:);
        k = k+1;
    end
end
```

The first set of comments that follow the function definition are displayed with the command `anymatrix('core/vecperm','help')`. After all the rest of the comments, an assignment for `properties` is made before checking whether the function is being called with no arguments in order to determine the properties, and if so the function is exited. Following that is the code to generate the matrix.

9 Testing Anymatrix Matrices

9.1 Function-Based Unit Tests for Matrix Properties

We have incorporated a test suite in `anymatrix/testing` that tests the declared properties of matrices in `anymatrix`. The tests are run by the function `test_anymatrix_properties` and are intended for those who are adding matrices to Anymatrix.

The files in this directory have the following roles in the testing framework.

- The directory `private/` houses the tests for each of the properties for which a test is feasible. Each function in this directory is named `test_<property>.m`, where `<property>` is some specific property. Each function takes a matrix as an input and returns 1 if the property holds or 0 otherwise. For example, `private/test_binary.m` tests if a matrix is made up of only 1s and 0s. It contains the following code:

  ```
  function out = test_binary(M)
  out = ~any( find(M ~= 0 & M ~= 1) );
  end
  ```

  Properties that do not have tests, and therefore have no corresponding files in this directory, will be skipped in testing. Anymatrix contains tests for the following properties:
  - binary,
  - complex,

  - hankel,
  - hermitian,
  - hessenberg,
  - integer,
  - nonnegative
  - permutation
  - positive
  - square,
  - stochastic,
  - symmetric,
  - toeplitz,
  - triangular.

  The test for a stochastic matrix involves a tolerance, since we cannot expect a float-point matrix to be exactly stochastic.
- The function `anymatrix_func_based_tests` is automatically-generated and it contains functions for unit tests, defined as specified in the MATLAB function-based unit testing documentation. It is generated by the function `test_anymatrix_properties`.
- The function `test_anymatrix_properties` starts the testing. Before starting the unit tests it first checks the list of properties recognized by `anymatrix` and optionally throws warnings for those properties that do not have a test in the `private/` directory. Following that, the function goes through the whole matrix collection and automatically generates unit tests in `anymatrix_func_based_tests` if they are not already present in it (as will be the case after Anymatrix is first installed). If a new matrix has been added to the collection, a test for it is automatically added by this function. If the first input argument, `regenerate_tests`, is 1 then the test functions in `anymatrix_func_based_tests.m` are regenerated, overwriting the current contents. This is useful when some of the matrices are removed from the collection and therefore should no longer be tested.
- The function `anymatrix_check_props` takes a matrix and a matrix ID and checks that all properties that Anymatrix holds for that matrix ID are true for the given matrix by running the property tests, where available. Passing or failing of a property is indicated to the unit testing framework through the `verifyTrue` statement where properties that fail are passed back to the framework for writing the final report to the command line.

The automatic generation of unit tests passes each member of a set of small dimensions as a first argument to each matrix and leaves the rest of the inputs, if any, at their default values. Most of the matrices can be tested this way and therefore the testing is automatic—once a user adds a new matrix into `anymatrix` it will be automatically picked up by the testing framework on the next run. However, not all matrices take only a single dimension and in general we might want to test several different matrices generated by a single function. For example, we can test the matrices in the `hadamard` group by exhaustively generating all possible matrices that `hadamard/hadamard` and `hadamard/complex_hadamard` provide. In these cases the script reads in the function-based unit tests from `anymatrix/<group_name>/private/am_unit_tests.m`. If that file exists for a particular group, all the function-based unit tests for those matrices are read into the file `anymatrix/testing/anymatrix_func_based_tests.m` and not auto-generated. This allows users to share custom testing functions with the groups of matrices. For example, here is an `am_unit_tests.m` for the `hadamard` group that we have put in place to test all the matrices in that group:

```
function test_hadamard_hadamard(testcase)
    [~,dims] = anymatrix('hadamard/hadamard');
    nn = length(dims);
    for i = 1:nn
        n = dims(i,1);
        for k = 1:dims(i,2)
            A = anymatrix('hadamard/hadamard', n, k);
            anymatrix_check_props(A, 'hadamard/hadamard', testcase);
        end
    end
end

function test_hadamard_complex_hadamard(testcase)
    [~,dims] = anymatrix('hadamard/complex_hadamard');
    nn = length(dims);
    for i = 1:nn
        n = dims(i,1);
        for k = 1:dims(i,2)
            A = anymatrix('hadamard/complex_hadamard', n, k);
            anymatrix_check_props(A, 'hadamard/complex_hadamard', testcase);
        end
    end
end
```

Here is the output (truncated) from invoking the test suite in the current version of the collection.

```
>> test_anymatrix_properties
Anymatrix scanning done.
Running anymatrix_func_based_tests
.......... .......... .......... .......... ..........
.......... .......... .......... .......... ..........
.......... .......... .......... .......... ......
Done anymatrix_func_based_tests
----------


ans =

  146×6 table

                                Name                                 Passed    Failed
      Incomplete    Duration       Details

    ------------------------------------------------------------    ------    ------
      ----------    ---------    ------------

    {'anymatrix_func_based_tests/test_contest_baitsample'       }    true      false
         false        0.016046    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_curvature'        }    true      false
         false        0.011548    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_erdrey'           }    true      false
```

```
        false        0.015165    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_geo'             }    true      false
        false         0.01542    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_gilbert'         }    true      false
        false        0.015195    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_kleinberg'       }    true      false
        false        0.016267    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_lap'             }    true      false
        false        0.014667    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_lockandkey'      }    true      false
        false         0.01977    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_mht'             }    true      false
        false        0.012755    {1×1 struct}
    {'anymatrix_func_based_tests/test_contest_pagerank'        }    true      false
        false        0.012597    {1×1 struct}
[...]
```

9.2 Other Tests

The file `group_name/private/test_run.m` stores tests for group `group_name`. The commands `anymatrix('test')` and `anymatrix('test',group_name)` run the tests of all the groups or of a particular group respectively. An example is the `hadamard` group that has a test to check that all the matrices are Hadamard matrices (since "Hadamard" is not a matrix property, so is not tested by the function-based unit tests described in the previous subsection):

```
>> anymatrix test hadamard
Testing Hadamard matrices of dimension  1.
Testing Hadamard matrices of dimension  2.
Testing Hadamard matrices of dimension  4.
Testing Hadamard matrices of dimension  8.
[...]
Testing complex Hadamard matrices of dimension 10.
All complex Hadamard tests passed.
```

10 Extending the Collection

The base Anymatrix collection can be extended in several ways.

10.1 Adding a New Group

A new group of matrices, `mygroup` say, can be created using the following steps.

- Create the directories `anymatrix/mygroup/` and `anymatrix/mygroup/private/`.
- Copy `anymatrix/core/anymatrix_core.m` to `anymatrix/mygroup/`, change the name to `anymatrix_mygroup.m` and change "`core`" on line 3 to "`mygroup`".
- Create in `anymatrix/mygroup/private/` the functions that generate the matrices, and specify the matrix properties either in those functions or in the function `anymatrix/mygroup/private/am_properties.m` (see Section 7.2).
- Create `anymatrix/mygroup/private/Contents.m` (a file of that name provides a summary of the program files in the directory in which it resides).

- Optionally create a file `anymatrix/mygroup/private/am_unit_tests.m` in order to test matrix properties for matrices that will not automatically be generated by the existing testing framework; see Section 9.1.
- Optionally create the function `anymatrix/mygroup/private/test_run.m` for any other testing required; see Section 9.2.
- Run `anymatrix scan` to force a scan of the file system that will include the new group.

10.2 Extending a Group

To add new matrices into an existing group, functions that generate the matrices (see Section 8) need to be added into the appropriate group directory. We do not recommend extending the groups provided with Anymatrix; matrices should be added to a different existing group or a new group.

10.3 Integrating Remote Groups

We have incorporated Git integration to allow remote groups to be downloaded from Git repositories into Anymatrix storage space and treated in the same way as the built-in groups.

For example, we can incorporate the following four collections.

- https://github.com/higham/matrices-correlation-invalid: invalid correlation matrices collected by Higham and Strabić [12], [13]. These are matrices that are intended to be correlation matrices but for various reasons relating to their construction have a negative eigenvalue and so are not positive semidefinite.
- https://github.com/higham/hpl-ai-matrix: a parametrized $n \times n$ matrix designed for use in the HPL-AI Mixed Precision Benchmark[4] [4].
- https://github.com/Xiaobo-Liu/matrices-mp-cosm: a collection of MATLAB functions that generate the matrices used for testing multiprecision algorithms for computing the matrix cosine [1].
- https://github.com/mfasi/randsvdfast-matlab: a function `randsvdfast` that provides similar functionality to `anymatrix('gallery/randsvd')` but uses a faster algorithm [3].

To incorporate the matrices in the first GitHub repository as a group named `corrinv` we can use the `'groups'` command as follows.

```
>> anymatrix('g','corrinv','higham/matrices-correlation-invalid');
Cloning into '[...]/corrinv/private'...
[...]
Anymatrix remote group cloned.
```

First, the command creates a directory `anymatrix/corrinv` and puts a directory `private/` and a file `anymatrix_corrinv.m` inside (see Section 12.1). Then it clones the specified repository from GitHub (a full URL can be provided to clone from somewhere else) into the `private/` directory, and rescans the file structure to add the new group into `anymatrix`.

Running the same command again runs `git pull` on the matrix group to update it. In this case `git` tells us it is already up to date.

```
>> anymatrix('g','corrinv','higham/matrices-correlation-invalid');
Already up to date.
```

[4] https://icl.bitbucket.io/hpl-ai/

The name of the repository is in this case redundant since `git` records it in its own file structure. Therefore an empty third argument would also suffice to ask `anymatrix` to pull the updates.

For this functionality to work `git` has to be set up to work on the computer and we expect that most users who use this functionality will already have a working `git` installation. Anymatrix does not try to resolve `git` errors, such as failures of automatic merging on `git pull`—the repository of a group has to be fixed manually if `git pull` fails. Indeed updating remote groups can be done completely manually, as long as `anymatrix('scan')` is issued afterwards.

To prepare a new remote group that is compatible with `anymatrix` create an online repository, for example on GitHub, place the matrix M-files directly in its root directory, and add the `Contents.m` file. If the properties of matrices are not specified inside the matrix files, add an additional `am_properties.m` file to document the properties. The group can then be incorporated in Anymatrix as above.

10.4 Suggestions from Users

We welcome suggestions for matrices to add to existing groups and suggestions for new groups to be added, and we are happy to link to remote groups on the Anymatrix GitHub page.

11 Other Features

11.1 Widely Understood Names

We would like `anymatrix/<group>/<matrix_name>` to be an authoritative way of referring to a matrix, at least for the built-in groups, just as saying `gallery('orthog',4)` (say) precisely defines that matrix.

11.2 Distinguishing Matrices from Other M-Files

We need a way of distinguishing matrices from any support files that appear in a group directory. We infer this from properties—an M-file is a matrix file if it has a `properties` assignment inside or if its name appears in the `am_properties.m` (see Section 7.2). The only requirement in this case is that an empty property array is added for matrices without any properties. If an M-file does not appear in `am_properties.m` and does not contain an assignment for `properties` inside, then it will not be considered as a matrix M-file and will be ignored by `anymatrix`.

11.3 Searching by Terms in the Help Comments

A command `'lookfor'` is provided to look for terms in the matrix help comments. Here is an example usage, where we include spaces around the search term to exclude matches on "nonzero" and "zeros(".

```
% Search for matrices that mention " zero " in the comments.
>> M = anymatrix('lookfor',' zero ')
M =
  11×1 cell array
    {'contest/pathlength'}
    {'core/hessfull01'   }
    {'core/nilpot_triang'}
```

```
{'gallery/chow'      }
{'gallery/clement'   }
{'gallery/randcolu'  }
{'gallery/randcorr'  }
{'gallery/redheff'   }
{'gallery/smoke'     }
{'matlab/hankel'     }
{'matlab/rosser'     }
```

11.4 Matrix, Group, and Property Naming Rules

Since the group name is used by Anymatrix to construct an M-file name (for the bridge file discussed in Section 12.1), the group name can only contain characters that are valid in the program file names: it should begin with an alphabetical character and contain only letters, numbers, and underscores and must not contain a hyphen. The same is true for the matrix names since the function that generates a matrix has the same name as the matrix. Hyphens are supported in group names and matrix IDs when typing in commands, but `anymatrix` replaces them by underscores before searching.

Properties are also used to construct function names in `anymatrix/testing/private/`. However, we did not want to ask for underscores in the property names and so we allow white space as well as hyphens in them. The test functions of properties, however, should conform to the function naming conventions of MATLAB and we replace any hyphens or white space characters in the properties by underscores when matching them to the file names in `anymatrix/testing/private/`.

12 Implementation

12.1 Overall Directory Structure

The directory structure of Anymatrix is

```
anymatrix/
    .git/
    ...
    core/
        anymatrix_core.m
        private/
            augment.m
            beta.m
            ...
    ...
    sets/
         my_set.txt
    testing/
        private/
            test_binary.m
            ...
```

```
        anymatrix_check_props.m
        ...
    anymatrix.m
    anymatrix_alias.m
    Contents.m
    inv_prop_map.m
    license.txt
    prop_list.m
    prop_map.m
    ...
```

We can only call the function `core/private/augment.m` (for example) from the `core` directories, so this directory needs to be on the MATLAB path. The `core` directory contains a function `anymatrix_core.m` that we call from `anymatrix.m` with the desired input arguments, so that `anymatrix_core.m` acts as a bridge to the `core` matrix generators inside the `private/` directory. The directory `anymatrix` needs to be added to the MATLAB path. Anymatrix adds each of the group directories to the path so that `anymatrix.m` can call the bridge files (such as `anymatrix_core.m`) that give access to functions in the `private` directories of each group.

This approach, with its use of private directories, may seem inelegant, but it avoids the need to put each directory containing the matrices in a group on the MATLAB path, which avoids cluttering the MATLAB namespace and creating name clashes.

12.2 Directory `sets/`

The directory `sets/` contains a set of files, one per set, listing matrix IDs, parameters, and comments (see Section 6.2).

12.3 Directory `testing/`

This directory contains the code for unit testing of matrix properties (see Section 9.1).

12.4 File `anymatrix.m`

This file contains the main interface of `anymatrix`. It is comprised of three parts. The first part parses the command supplied by a user and catches some common errors in the commands, throwing warnings with some information of what went wrong or what should be fixed in the command. The second part executes the specified command if no warnings have been detected in the first part. The third part includes a set of functions for performing various operations.

12.5 File `anymatrix_alias.m`

This is a template for creating a different name for `anymatrix`, if needed. Copy this file somewhere on the MATLAB path and rename it and change its first line correpsondingly. The authors have copied it to a file `am.m` and so write `am` instead of `anymatrix`.

12.6 File `Contents.m`

Information about `anymatrix`, such as its version.

12.7 File `inv_prop_map.m`

Defines an inverted property map (see Section 7.3).

12.8 File `licence.txt`

A standard BSD 2-Clause license for `anymatrix`.

12.9 File `prop_list.m`

Defines a list of recognised properties (see Section 7.1).

12.10 File `prop_map.m`

Defines a property map (see Section 7.3).

Acknowledgements

We thank Des Higham for permission to include CONTEST and NESSIE, Per Christian Hansen for permission to include Regularization Tools, Neil Sloane for permission to include his collection of Hadamard matrices, and Cleve Moler for permission to use his function for the Hadamard matrix of Baumert, Golomb, and Hall.

We thank Bobby Cheng, Mike Croucher, Massimiliano Fasi, Xiaobo Liu, Srikara Pranesh, and Mawussi Zounon for comments and suggestions.

References

[1] Awad H. Al-Mohy, Nicholas J. Higham, and Xiaobo Liu. Arbitrary precision algorithms for computing the matrix cosine and its Fréchet derivative. MIMS EPrint 2021.13, Manchester Institute for Mathematical Sciences, The University of Manchester, August 2021. 23 pp. (Cited on p. 26.)

[2] Rajendra Bhatia. Infinitely divisible matrices. *Amer. Math. Monthly*, 113(3):221–235, 2006. (Cited on p. 17.)

[3] Massimiliano Fasi and Nicholas J. Higham. Generating extreme-scale matrices with specified singular values or condition numbers. *SIAM J. Sci. Comput.*, 43(1):A663–A684, 2021. (Cited on p. 26.)

[4] Massimiliano Fasi and Nicholas J. Higham. Matrices with tunable infinity-norm condition number and no need for pivoting in LU factorization. *SIAM J. Matrix Anal. Appl.*, 42(1): 417–435, 2021. (Cited on p. 26.)

[5] Silvia Gazzola, Per Christian Hansen, and James G. Nagy. IR tools: a MATLAB package of iterative regularization methods and large-scale test problems. *Numer. Algorithms*, 81 (3):773–811, 2018. (Cited on p. 14.)

[6] Per Christian Hansen. Regularization Tools: A Matlab package for analysis and solution of discrete ill-posed problems. *Numer. Algorithms*, 6(1):1–35, 1994. (Cited on p. 13.)

[7] Per Christian Hansen. Regularization Tools version 4.0 for Matlab 7.3. *Numer. Algorithms*, 46(2):189–194, 2007. (Cited on p. 13.)

[8] Per Christian Hansen and Jakob Sauer Jørgensen. AIR tools II: Algebraic iterative reconstruction methods, improved implementation. *Numer. Algorithms*, 79(1):107–137, 2018. (Cited on p. 14.)

[9] Nicholas J. Higham. The Matrix Computation Toolbox. `http://www.maths.manchester.ac.uk/~higham/mctoolbox`. (Cited on p. 13.)

[10] Nicholas J. Higham. *J*-orthogonal matrices: Properties and generation. *SIAM Rev.*, 45(3): 504–519, 2003. (Cited on p. 18.)

[11] Nicholas J. Higham and Mantas Mikaitis. Anymatrix: An extensible MATLAB matrix collection. MIMS EPrint 2021.16, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, October 2021. 20 pp. (Cited on p. 2.)

[12] Nicholas J. Higham and Nataša Strabić. Anderson acceleration of the alternating projections method for computing the nearest correlation matrix. *Numer. Algorithms*, 72(4): 1021–1042, 2016. (Cited on p. 26.)

[13] Nicholas J. Higham and Nataša Strabić. Bounds for the distance to the nearest correlation matrix. *SIAM J. Matrix Anal. Appl.*, 37(3):1088–1102, 2016. (Cited on p. 26.)

[14] N. J. A. Sloane. A library of Hadamard matrices. `http://neilsloane.com/hadamard/`. (Cited on p. 13.)

[15] Wojciech Tadej and Karol Życzkowski. A concise guide to complex Hadamard matrices. *Open Syst. Inf. Dyn.*, 13(02):133–177, 2006. (Cited on p. 13.)

[16] Alan Taylor and Desmond J. Higham. CONTEST: A controllable test matrix toolbox for MATLAB. *ACM Trans. Math. Software*, 35(4):26:1–26:17, 2009. (Cited on p. 13.)

[17] Alan Taylor and Desmond J. Higham. NESSIE: Network example source supporting innovative experimentation. In *Network Science: Complexity in Nature and Technology*, Ernesto Estrada, Maria Fox, Desmond J. Higham, and Gian-Luca Oppo, editors, Springer-Verlag, 2010, pages 85–106. (Cited on p. 13.)