matrix-repository Documentation

Release 0.1

Weijian Zhang

January 22, 2015

Contents

1 Introduction **3**
- 1.1 Background . . . 3
- 1.2 License . . . 3
- 1.3 Acknowledgements . . . 4

2 Installing matrix-repository **5**
- 2.1 Dependencies . . . 5
- 2.2 Installation . . . 5

3 Matrices in the Collection **7**
- 3.1 Notation Convention . . . 7
- 3.2 Chow Matrix . . . 7
- 3.3 Cauchy Matrix . . . 7
- 3.4 Clement Matrix . . . 8
- 3.5 Chebyshev Spectral Differentiation Matrix . . . 8
- 3.6 Hilbert Matrix . . . 8

4 Using matrix-repository in Fortran **9**
- 4.1 Getting Started . . . 9
- 4.2 Using Utility Modules . . . 10
- 4.3 Matrix Subroutines . . . 10

5 Using matrix-repository in Python **13**
- 5.1 Matrix functions . . . 13
- 5.2 Access functions . . . 15

6 Using matrix-repository in Julia **19**

7 Glossary **21**

Python Module Index **23**

Warning: matrix-repository will merge with Matrix Depot project shortly.

matrix-repository is a collection of matrices with special properties. These properties include: known inverses and/or known eigenvalues; ill-conditioned and/or rank deficient matrices; symmetric, positive definite, orthogonal, defective, involutory, and/or totally positive.

Contents:

CHAPTER 1

Introduction

1.1 Background

Test matrices are born with the first computer programs for matrix computations in the 1940s. Test matrix collections have been developed ever since. The Harwell-Boeing Collection developed by Duff, Laboratory, Grimes and Lewis in 1989 is a set of test matrices for sparse matrix problems. The problems are drawn from a wide variety of scientific and engineering disciplines. In 1991, Higham present a collection of 45 parametrized test matrices in MATLAB. The matrices are mostly square, dense, nonrandom, and of arbitrary dimension. Making use of the World Wide Web, Boisvert et al. develop a repository of test matrices (Matrix Market) which incorporated the Harwell-Boeing collection. The University of Florida Sparse Matrix Collection is another large set of sparse matrices that arise in real applications.

In modern days, mixed languages programming is a common practice for developing large projects: using a high-level language for coding the big picture and a low-level language for numerically intensive elements. Similarly, Numerical analysts would first test an algorithm on MATLAB or Python and then deploy the algorithm to Fortran or C. If, however, the test matrices in different languages are coded differently, the results and error generated may be different, which is problematic to analyse. The matrix collections mentioned above either contained fixed dimension matrices or were coded for a specific language. Our aim is to provide a test matrix collection that can be used in a variety languages and contains matrices of arbitrary dimension.

1.2 License

1.3 Acknowledgements

I would like to thank my advisor Nicholas J. Higham for his recommendations, supports, comments and useful feedback on early drafts of this software. I would like to thank Edvin Deadman for checking my early drafts and providing many useful suggestions. I would also like to acknowledge suggestions and help from Sven Hammarling, Philip E. Gill and Mario Berljafa.

CHAPTER 2

Installing matrix-repository

2.1 Dependencies

- gfortran compiler
- To use the Python interface, you need Python 2.6 or above, include Python 3.x, with Numpy v1.8.
- To use the module `access`, you need python-pandas.

2.2 Installation

To install matrix-repository, type

```
$ git clone https://github.com/matrix-repository/matrix-repository.git
```

then do:

```
$ cd matrix-repository
$ make
```

To install the Python interface of matrix-repository, type

```
$ make python
```

CHAPTER 3

Matrices in the Collection

3.1 Notation Convention

We use capital letters, e.g., A, B, C, for matrices; subscripted capital letters, e.g., A_{ij} for matrix elements; lower case letters, e.g., x, y, z for vectors and lower case Greek letters, e.g., α, β, γ for scalars.

3.2 Chow Matrix

Chow matrix is a singular toeplitz lower hessenberg matrix. The pattern of a Chow matrix is defined by two parameters, say α and δ. For a $n \times n$ matrix, a Chow matrix looks like

$$\begin{bmatrix} \alpha+\delta & 1 & 0 & \cdots & \cdots & 0 \\ \alpha^2 & \alpha+\delta & 1 & 0 & \cdots & 0 \\ \alpha^3 & \alpha^2 & \alpha+\delta & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & 1 \\ \alpha^n & \alpha^{n-1} & \cdots & \cdots & \alpha^2 & \alpha+\delta \end{bmatrix} \tag{3.1}$$

3.3 Cauchy Matrix

Cauchy matrix, named after the French mathematician Augustin-Louis Cauchy, is a $n \times n$ matrix defined by two vectors x and y of length n, where $x_i - y_i \neq 0$. The (i, j) element of a Cauchy Matrix A is

$$A_{i,j} = \frac{1}{x_i - y_j}. \tag{3.2}$$

3.4 Clement Matrix

Clement matrix, named after Paul A. Clement, is a class of triple-diagonal matrices (should be called Clement matrices) of the form

$$A = \begin{bmatrix} 0 & x_1 & & & & \\ y_1 & 0 & x_2 & & & \\ & y_2 & 0 & x_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & x_n \\ & & & & y_n & 0 \end{bmatrix} \tag{3.3}$$

. When n is even, A is singular.

3.5 Chebyshev Spectral Differentiation Matrix

Chebyshev Spectral differentiation matrix A is a $n \times n$ matrix with entries

$$A_{1,1} = \frac{2(n-1)^2+1}{6}, \qquad A_{n,n} = -\frac{2(n-1)^2+1}{6}, \tag{3.4}$$

$$A_{j,j} = \frac{-x_j}{2(1-x_j^2)}, \qquad j = 2, \ldots, n-1, \tag{3.5}$$

$$A_{i,j} = \frac{c_i(-1)^{i+j}}{c_j(x_i - x_j)}, \qquad i \neq j, \tag{3.6}$$

$$\tag{3.7}$$

where

$$c_i = \begin{cases} 2 & i = 1 \text{ or } n, \\ 1 & \text{otherwise}, \end{cases} \tag{3.8}$$

and $x_j = \cos(j\pi/n)$ for $j = 1, \ldots, n$ are the Chebyshev points.

3.6 Hilbert Matrix

Hilbert matrix, named after the German mathematician David Hilbert is a special case of *Cauchy Matrix*. The (i, j) element of a Hilbert matrix A is

$$A_{i,j} = \frac{1}{i+j-1}. \tag{3.9}$$

CHAPTER 4

Using matrix-repository in Fortran

4.1 Getting Started

The command:

```
$ make
```

will create three modules `mr_type.mod`, `mr_util.mod` and `matrix_repository.mod` and an archived file `matrix_repository.a` in the directory `matrix-repository`:

```
$ ls
doc       matrix_repository.a          matrix-repository.py  README.rst
LICENSE   matrix-repository.extension  mr_type.mod           src
makefile  matrix_repository.mod        mr_util.mod           tests
```

The archive file contains object files for all matrix subroutines in the collection. To use matrix-repository, you need to write:

```
use matrix_repository, only: matrix_name
```

at the beginning of your program. Here is an example:

```
program test_fortran
  use mr_type, only: WP => DP
  use matrix_repository, only: chow
  implicit none
  real(WP), dimension(5,5) :: A
  integer :: info


  call chow(5, A, 5, 1.0_WP, 2.0_WP, info)

  write(*,*) "Welcome to Gallery!"
  if (info == 0) then
     write(*,*) "A is a double precision Chow matrix:"
     write(*,"(5 F9.4)") A
  end if
end program test_fortran
```

Note: In the example above, we also make use of the module `mr_type` which holds the KIND value for different data precision. Here `DP` stands for double precision.

To compile this example program using gfortran, you need to turn on the flag `-I` to specify the location of the `.mod` files that should be included in the program and link with `matrix_repository.a`. Suppose the directory `matrix-repository` is located at `~/matrix-repository`, then the command:

```
$ gfortran -o test -I ~/matrix-repository test_fortran.f90 \
 ~/matrix-repository/matrix_repository.a
```

will generate the executable file `test`.

4.2 Using Utility Modules

Two auxiliary modules are `mr_type` and `mr_util`.

4.2.1 Types

The file supplied as `mr_type.f90` contains a single module named `mr_type`, which holds the KIND value and mathematical constants.

Parameters	meaning
`DP`	double precision KIND value
`SP`	single precision KIND value
`PI_DP`	double precision π
`PI_SP`	single precision π

4.2.2 Utilities

The file supplied as `mr_util.f90` contains a single module named `mr_util`. The following subroutines or functions are included in this module.

mr_error(*string*[, *info*])
: Report an error message include `string`, if `info = -i`, the error message will tell the `i`-th argument has an error.

mr_outer_product(*a*, *b*)
: Perform vector outer product on two given vectors `a` and `b`.

d_negative_one_power(*i*)
: Return the `i`-th power of -1 as double precision.

s_negative_one_power(*i*)
: Return the `i`-th power of -1 as single precision.

4.3 Matrix Subroutines

cauchy(*n*, *A*, *LDA*, *x*, *y*, *info*)
: Generate a n-by-n *Cauchy Matrix*.

Parameters **n** : integer

The order of the matrix. Input.

A : (LDA, n) 2-d real array

A single precision or double precision. Output.

LDA : integer

The leading dimension of the matrix A. Input.

x : (n,) 1-d array

x single precision or double precision. Input.

y : (n,) 1-d array

y single precision or double precision. Input.

info : integer

info = 0 if successfully exits; else info = -i then i-th argument had all illegal value. Input.

clement (*n*, *A*, *LDA*, *info*)
Generate a n-by-n *Clement Matrix*.

Parameters **n** : integer

The order of the matrix A. Input.

A : (LDA, n) 2-d real array

Clement matrix, single precision or double precision. Output.

LDA : integer

The leading dimension of the matrix A. Input.

info : integer

info = 0 if successfully exits; else info = -i then i-th argument had all illegal value. Input.

chebspec (*n*, *A*, *LDA*, *info*)
Generate a n-by-n *Chebyshev Spectral Differentiation Matrix*.

Parameters **n** : integer

The order of the matrix A. Input.

A : (LDA, n) 2-d real array

Chebyshev spectral differentiation matrix, single precision or double precision. Output.

LDA : integer

The leading dimension of the matrix A. Input.

info : integer

info = 0 if successfully exits; else info = -i then i-th argument had all illegal value. Input.

hilb (*n*, *A*, *LDA*, *info*)
Generate a n-by-n *Hilbert Matrix*.

Parameters **n** : integer

The order of the matrix A. Input.

A : (LDA, n) 2-d real array

Hilbert matrix, single precision or double precision. Output.

LDA : integer

The leading dimension of the matrix A. Input.

info : integer

info = 0 if successfully exits; else info = -i then i-th argument had all illegal value. Input.

chow(*n*, *A*, *LDA*, *alpha*, *delta*, *info*)

Generate a n-by-n *Chow Matrix*.

Parameters **n** : integer

The order of the matrix A. Input.

A : (LDA, n) 2-d real array

Chow matrix, single precision or double precision. Output.

LDA : integer

The leading dimension of the matrix A. Input.

alpha : real scalar

single precision or double precision. Input.

delta : real scalar

single precision or double precision. Input.

info : integer

info = 0 if successfully exits; else info = -i then i-th argument had all illegal value. Input.

CHAPTER 5

Using matrix-repository in Python

When using matrix-repository, we recommend the following import convention:

```
import matrix_repository as mr
```

5.1 Matrix functions

matrix_repository.**chow**(*n*, *alpha=1*, *delta=0*, *dtype=None*)

Construct *Chow Matrix*.

chow(n, alpha, delta) is the Chow matrix - a singular Toeplitz lower Hessenberg matrix. A = chow(n, alpha, delta) is the matrix A = H(alpha) + delta*eye, where H is the lower Hessenberg matrix with H(i,j) = alpha^(i-j+1). Default: alpha = 1, delta = 0.

Parameters **n** : integer

The dimension of the matrix.

alpha : integer or floating point

delta : integer or floating point

dtype : dtype

Data type of the matrix.

Returns **chow** : (n, n) ndarray

Chow matrix.

Examples

```
>>> from matrix_repository import chow
>>> chow(4, dtype=int)
array([[1, 1, 0, 0],
       [1, 1, 1, 0],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

matrix_repository.**hilb**(*n*, *dtype=None*)

Construct *Hilbert Matrix*.

hilb(N) is the N-by-N matrix with elements 1/(i+j-1). It is a famous example of a badly conditioned matrix.

Parameters **n** : integer

The dimension of the matrix.

dtype : dtype

Data type of the matrix.

Returns **hilb** : (n, n) ndarray

Hilbert matrix.

Examples

```
>>> from matrix_repository import hilb
>>> hilb(5)
array([[ 1.        ,  0.5       ,  0.33333333,  0.25      ,  0.2       ],
       [ 0.5       ,  0.33333333,  0.25      ,  0.2       ,  0.16666667],
       [ 0.33333333,  0.25      ,  0.2       ,  0.16666667,  0.14285714],
       [ 0.25      ,  0.2       ,  0.16666667,  0.14285714,  0.125     ],
       [ 0.2       ,  0.16666667,  0.14285714,  0.125     ,  0.11111111]])
```

matrix_repository.**cauchy**(*x*, *y=None*, *dtype=None*)

Construct *Cauchy Matrix*

cauchy(x, y), where x, y are n-vectors, is the n-by-n matrix with c(i,j) = 1/(x(i)+y(j)). By default, y = x. Special case: if x is a scalar cauchy(x) is the same as cauchy(1:x).

Parameters **x** : (n,) array_like

1-D array or a scalar.

y : (n,) array_like

1-D array or a scalar.

dtype : dtype

Data type of the matrix.

Returns **cauchy** : (n, n) ndarray

Cauchy matrix.

Examples

```
>>> from numpy import array
>>> from matrix_repository import cauchy
>>> cauchy(array([1,2,3]))
array([[ 0.5       ,  0.33333333,  0.25      ],
       [ 0.33333333,  0.25      ,  0.2       ],
       [ 0.25      ,  0.2       ,  0.16666667]])
>>> cauchy(4)
array([[ 0.5       ,  0.33333333,  0.25      ,  0.2       ],
       [ 0.33333333,  0.25      ,  0.2       ,  0.16666667],
       [ 0.25      ,  0.2       ,  0.16666667,  0.14285714],
       [ 0.2       ,  0.16666667,  0.14285714,  0.125     ]])
```

matrix_repository.**chebspec**(*n*, *dtype=None*)

Construct *Chebyshev Spectral Differentiation Matrix*

c = chebspec(n) is a Chebyshev spectral differentiation matrix of order n. The computed eigenvector matrix x from eig is ill-conditioned (mesh(real(x)) is interesting).

Parameters **n** : integer

The dimension of the matrix.

dtype : dtype

Data type of the matrix.

Returns **chebspec** : (n, n) ndarray

Chebyshev spectral differentiation matrix.

Examples

```
>>> from matrix_repository import chebspec
>>> chebspec(4)
array([[ 3.16666667, -4.        ,  1.33333333, -0.5       ],
       [ 1.        , -0.33333333, -1.        ,  0.33333333],
       [-0.33333333,  1.        ,  0.33333333, -1.        ],
       [ 0.5       , -1.33333333,  4.        , -3.16666667]])
```

`matrix_repository.`**`clement`**(*n*, *dtype=None*)

Construct *Clement Matrix*

clement(n) is a tridiagonal matrix with zero diagonal entries and known eigenvalues. It is singular if N is odd. About 64 percent of the entries of the inverse are zero. The eigenvalues are plus and minus the numbers N-1, N-3, N-5, ..., (1 or 0).

Parameters **n** : integer

The dimension of the matrix.

dtype : dtype

Data type of the matrix.

Returns **clement** : (n, n) ndarray

Clement matrix.

Examples

```
>>> from matrix_repository import clement
>>> clement(4)
array([[ 0.,  1.,  0.,  0.],
       [ 3.,  0.,  2.,  0.],
       [ 0.,  2.,  0.,  3.],
       [ 0.,  0.,  1.,  0.]])
```

5.2 Access functions

`access.`**`matrix`**(*index*, *size=None*, *dtype=None*)

Access matrices by number.

matrix(index, size): access matrices in the collection using the integer index. The size of the matrix is defined by the integer size.

Parameters **index** : integer

The index of all the matrices in the collection. use matrix(help) to see.

size : integer

The dimension of the matrix.

dtype : dtype

Data type of the matrix.

Returns **matrix** : (n, n) ndarray

Examples

```
>>> from access import matrix
>>> matrix(help)
0 : chow 1 : hilb 2 : cauchy 3 : chebspec 4 : clement
>>> matrix(1)
hilb
>>> matrix(1,5)
array([[ 1.        ,  0.5       ,  0.33333333,  0.25      ,  0.2       ],
       [ 0.5       ,  0.33333333,  0.25      ,  0.2       ,  0.16666667],
       [ 0.33333333,  0.25      ,  0.2       ,  0.16666667,  0.14285714],
       [ 0.25      ,  0.2       ,  0.16666667,  0.14285714,  0.125     ],
       [ 0.2       ,  0.16666667,  0.14285714,  0.125     ,  0.11111111]])
```

access.**property**(*matrix_name=None*, **argv*)

Print the property of a given matrix.

property(matrix_name): if matrix_name is given, then the properties of that matrix are printed out. If no argument is given, then the properties of all matrices are printed out.

1: yes, it has that property. 0: no, it doesn't have that property.

Inverse: the inverse is known explicitly.

Ill-cond: ill-conditioned for some values of arguments.

Rank: rank-deficient for some value of arguments.

Symm: symmetric for some values of arguments.

Pos Def: symmetric positive definite for some values of the parameters.

Orth: orthogonal or a diagonal scaling of an orthognal matrix, for some values of the parameters.

Eig: something is known about the eignsystem.

Parameters **matrix_name** : str

matrix name.

Returns **property** : pandas DataFrame

list of properties

Examples

```
>>> from access import property
>>> property('chow')
Inverse     0
Ill-cond    0
Rank        1
Symm        0
Pos Def     0
Orth        0
Eig         1
>>> property()
          Inverse  Ill-cond  Rank  Symm  Pos Def  Orth  Eig
cauchy          1         1     0     1        1     0    0
chebspec        0         0     1     0        0     0    1
chow            0         0     1     0        0     0    1
clement         1         0     1     1        0     0    1
hilb            1         1     0     1        1     0    0
```

CHAPTER 6

Using matrix-repository in Julia

Coming soon.

CHAPTER 7

Glossary

Defective A defective matrix is a square matrix that does not have a complete basis of eignvectors.

Hankel A Hankel matrix has equal entries on each anti-diagonal.

Hessenberg An upper Hessenberg matrix has zero entries below the first subdiagonal and a lower Hessenberg matrix has zero entries above the first superdiagonal.

Involutory An involutory matrix is a matrix that is its own inverse, i.e., $A^2 = I$ for a matrix A.

Toeplitz A Toeplitz matrix has equal entries on each diagonal and it looks like

$$\begin{bmatrix} t_0 & t_1 & t_2 & \cdots & \cdots & t_{n-1} \\ t_{-1} & t_0 & t_1 & t_2 & \cdots & t_{n-2} \\ t_{-2} & t_{-1} & t_0 & t_1 & \cdots & t_{n-3} \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & t_1 \\ t_{-n+1} & t_{-n+2} & \cdots & \cdots & t_{-1} & t_0 \end{bmatrix}. \tag{7.1}$$

a

access, 15

m

matrix_repository, 13

A

access (module), 15

C

cauchy() (built-in function), 10
cauchy() (in module matrix_repository), 14
chebspec() (built-in function), 11
chebspec() (in module matrix_repository), 14
chow() (built-in function), 12
chow() (in module matrix_repository), 13
clement() (built-in function), 11
clement() (in module matrix_repository), 15

D

d_negative_one_power() (built-in function), 10
Defective, **21**

H

Hankel, **21**
Hessenberg, **21**
hilb() (built-in function), 11
hilb() (in module matrix_repository), 13

I

Involutory, **21**

M

matrix() (in module access), 15
matrix_repository (module), 13
mr_error() (built-in function), 10
mr_outer_product() (built-in function), 10

P

property() (in module access), 16

S

s_negative_one_power() (built-in function), 10

T

Toeplitz, **21**