

---

# **Matrix Depot Documentation**

*Release 0.1.0*

**Weijian Zhang**

**Jan 19, 2018**



---

# Contents

---

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Installation</b>                                | <b>3</b>  |
| 1.1      | Usage . . . . .                                    | 3         |
| 1.2      | Matrices . . . . .                                 | 6         |
| 1.3      | Random Graphs . . . . .                            | 47        |
| 1.4      | Test Problems for Regularization Methods . . . . . | 48        |
| 1.5      | Groups . . . . .                                   | 51        |
| 1.6      | Interface to Test Collections . . . . .            | 53        |
| 1.7      | Adding New Matrix Generators . . . . .             | 56        |
| 1.8      | Examples . . . . .                                 | 59        |
|          | <b>Bibliography</b>                                | <b>67</b> |



Matrix Depot is an extensible test matrix collection for Julia. It provides a diverse collection of test matrices, including parametrized matrices and real-life matrices.

- [Source at Github](#)
- [Release Notes](#)



To install the release version, type:

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

## 1.1 Usage

Every matrix in the collection is represented by a string "matrix\_name", for example, the Cauchy matrix is represented by "cauchy" and the Hilbert matrix is represented by "hilb".

The matrix groups are also symbolized by strings. For example, the class of the symmetric matrices is symbolized by "symmetric".

### **matrixdepot** ()

Return a list of all the matrices in the collection:

```
julia> 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) prolate      40) randcorr
41) rando        42) randsvd      43) rohess       44) rosser
45) sampling     46) shaw         47) spikes       48) toeplitz
49) tridiag      50) triw         51) ursell        52) vand
53) wathen       54) wilkinson    55) wing

Groups:
```

|                      |                        |                     |                       |
|----------------------|------------------------|---------------------|-----------------------|
| <code>all</code>     | <code>data</code>      | <code>eigen</code>  | <code>ill-cond</code> |
| <code>inverse</code> | <code>pos-def</code>   | <code>random</code> | <code>regprob</code>  |
| <code>sparse</code>  | <code>symmetric</code> |                     |                       |

**matrixdepot** (*matrix\_name*, *p1*, *p2*, ...)

Return a matrix specified by the query string *matrix\_name*. *p1*, *p2*, ... are input parameters depending on *matrix\_name*. For example:

```
julia> matrixdepot("hilb", 5, 4)
5x4 Array{Float64,2}:
1.0      0.5      0.333333  0.25
0.5      0.333333  0.25     0.2
0.333333 0.25     0.2      0.166667
0.25     0.2      0.166667 0.142857
0.2      0.166667 0.142857 0.125
```

**matrixdepot** (*matrix\_name*)

Return the documentation of *matrix\_name*, including input options, groups and reference. For example:

```
julia> matrixdepot("moler")
Moler Matrix

The Moler matrix is a symmetric positive definite matrix. It has one small
eigenvalue.

Input options:

  • [type,] dim, alpha: dim is the dimension of the matrix, alpha is a
    scalar;

  • [type,] dim: alpha = -1.

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

References:

J.C. Nash, Compact Numerical Methods for Computers: Linear Algebra and
Function Minimisation, second edition, Adam Hilger, Bristol, 1990
(Appendix 1).
```

**matrixdepot** (*group\_name*)

Return a list of matrices which belong to group *group\_name*. For example:

```
julia> matrixdepot("pos-def")
11-element Array{ASCIIString,1}:
"hilb"
"cauchy"
"circul"
"invhilb"
"moler"
"pascal"
"pei"
"minij"
"tridiag"
"lehmer"
"poisson"
```



**matrixdepot** (*group1*, *group2*, ...)

Return a list of matrices which belong to *group1* and *group2*, etc. For example:

```
julia> matrixdepot("symmetric", "inverse", "ill-cond", "pos-def")
7-element Array{ASCIIString,1}:
"hilb"
"cauchy"
"invhilb"
"moler"
"pascal"
"pei"
"tridiag"
```

**matrixdepot** (*num*)

Access matrix by number. For example:

```
julia> matrixdepot(3)
"chebspec"
```

**matrixdepot** (*num1:num2*)

Access matrix by range. For example:

```
julia> matrixdepot(3:12)
10-element Array{ASCIIString,1}:
"chebspec"
"chow"
"circul"
"clement"
"dingdong"
"fiedler"
"forsythe"
"frank"
"grcar"
"hadamard"
```

**matrixdepot** (*num*, *num1:num2*...)

Access matrix by a mixture of numbers and ranges. For example:

```
julia> matrixdepot(1:4, 6, 10:15)
11-element Array{AbstractString,1}:
"baart"
"binomial"
"cauchy"
"chebspec"
"circul"
"fiedler"
"forsythe"
"foxgood"
"frank"
"gravity"
"grcar"
```

**matrixdepot** (*name*, *:get*)

Download a matrix from test matrix collections, where *name* is a string of collection name + / + matrix name. For example:

```
julia> matrixdepot("HB/1138_bus", :get)
```

`MatrixDepot.update()`

Update matrix collection database from the web server.

`matrixdepot (name)`

Output matrix information, where `name` is a matrix data.

`matrixdepot (name, :read)`

Generate the matrix data given by `name`.

We can define our own groups using the macro `@addgroup` and remove a defined group using `@rmgroup`.

`@addgroup group_name = ["matrix1", "matrix2", "matrix3"]`

Create a new group `"group_name"` such that it has members `"matrix1"`, `"matrix2"` and `"matrix3"`.

`@rmgroup group_name`

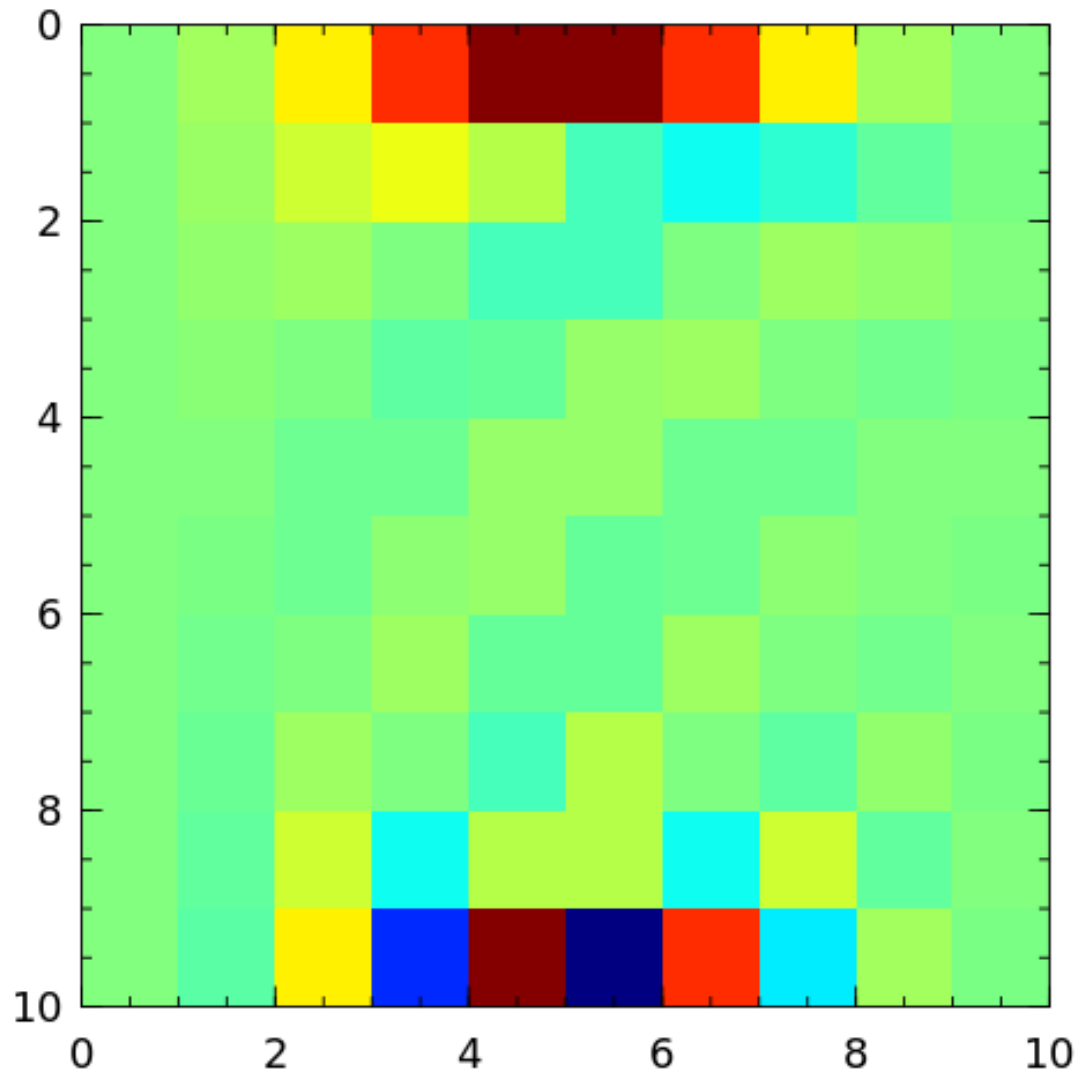
Delete a created group `group_name`.

## 1.2 Matrices

- *binomial*
- *cauchy*
- *chebspec*
- *chow*
- *circul*
- *clement*
- *companion*
- *dingdong*
- *fiedler*
- *forsythe*
- *frank*
- *golub*
- *grcar*
- *hadamard*
- *hankel*
- *hilb*
- *invhilb*
- *invol*
- *kahan*
- *kms*
- *lehmer*
- *lotkin*
- *magic*
- *minij*

- *moler*
- *neumann*
- *oscillate*
- *parter*
- *pascal*
- *pei*
- *poisson*
- *prolate*
- *randcorr*
- *rando*
- *randsvd*
- *rohess*
- *rosser*
- *sampling*
- *toeplitz*
- *tridiag*
- *triw*
- *vand*
- *wathen*
- *wilkinson*

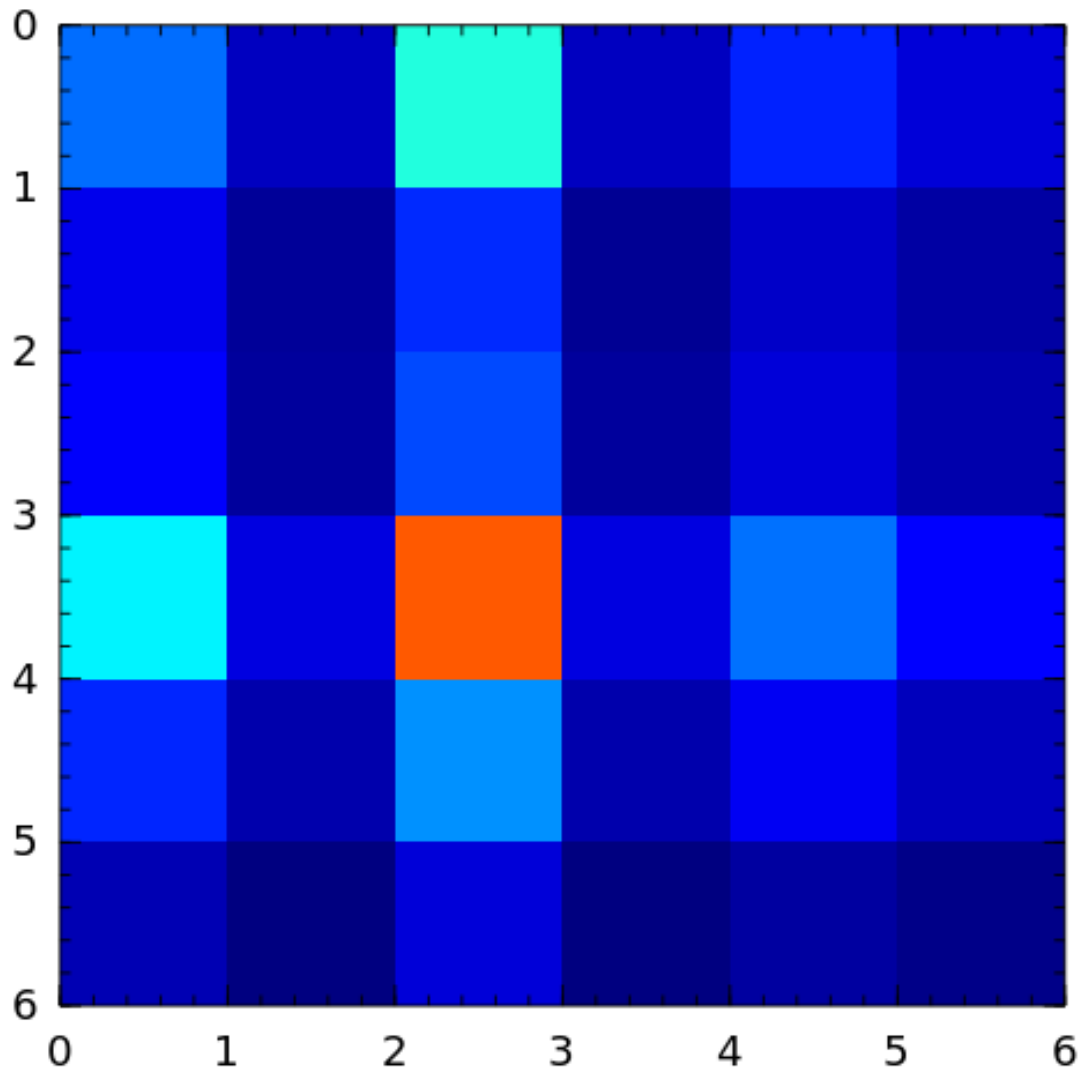
**binomial** A binomial matrix that arose from the example in [\[bmsz01\]](#). The matrix is a multiple of involutory matrix.



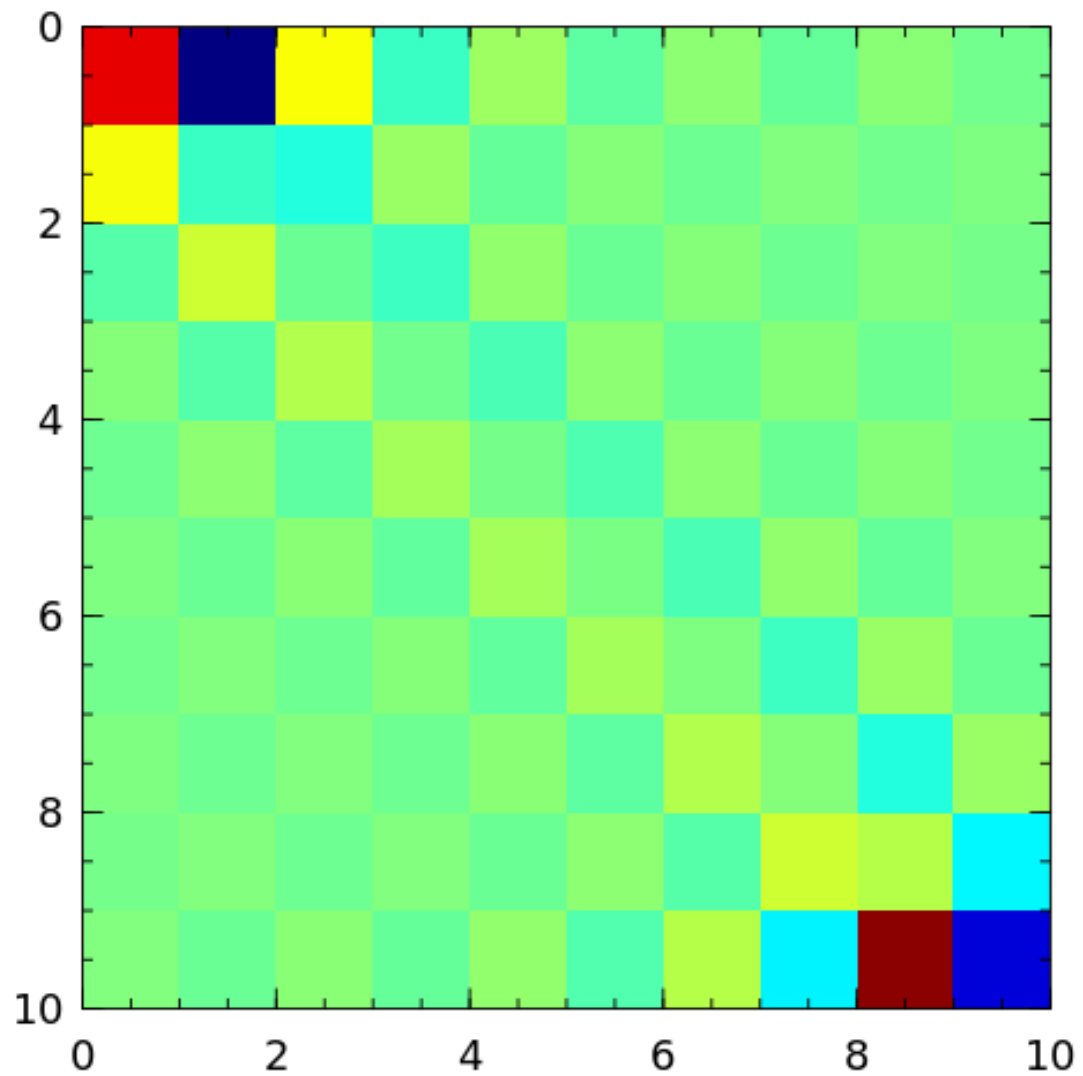
**cauchy** The Cauchy matrix is an m-by-n matrix with  $(i, j)$  element

$$\frac{1}{x_i - y_j}, \quad x_i - y_j \neq 0,$$

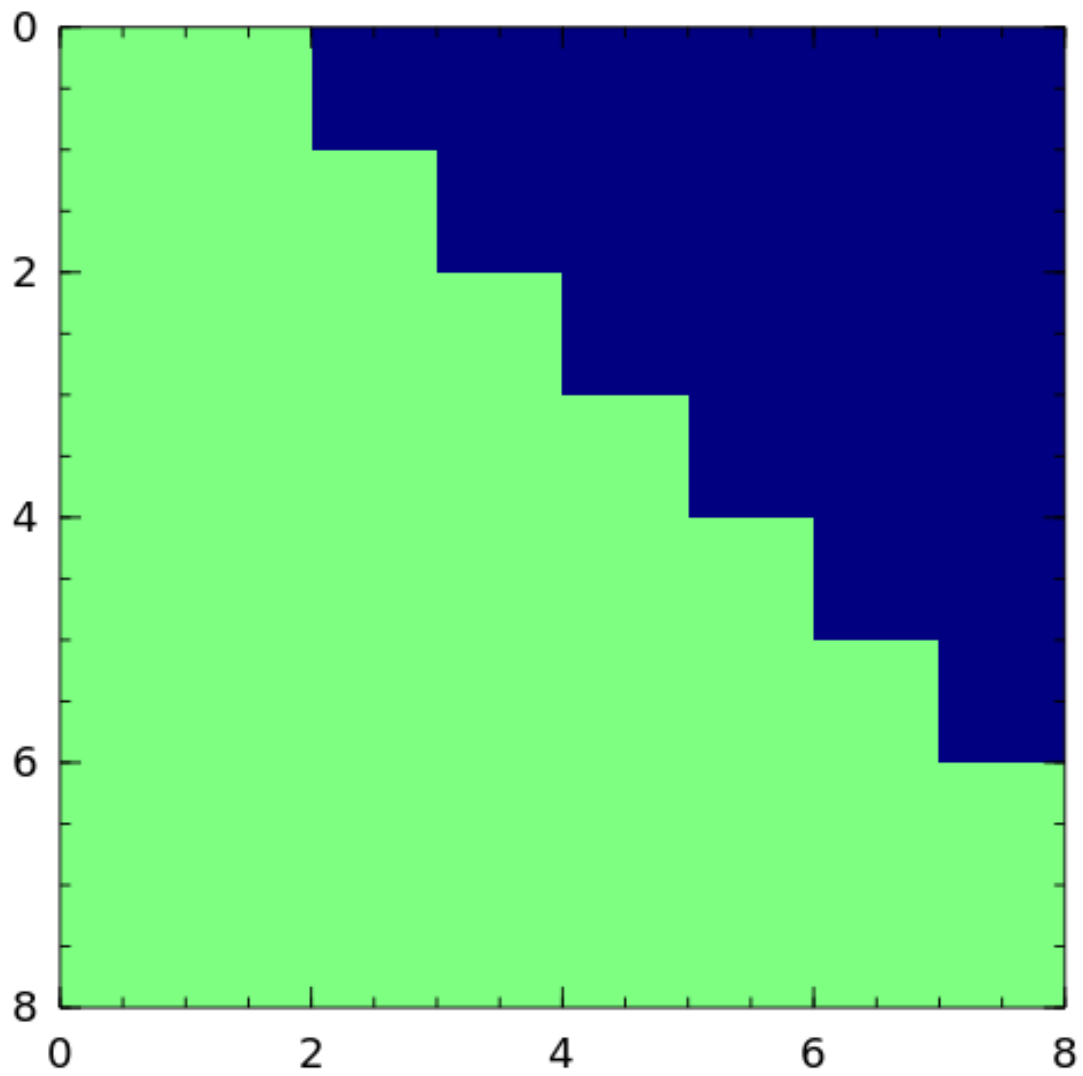
where  $x_i$  and  $y_j$  are elements of vectors  $x$  and  $y$ .



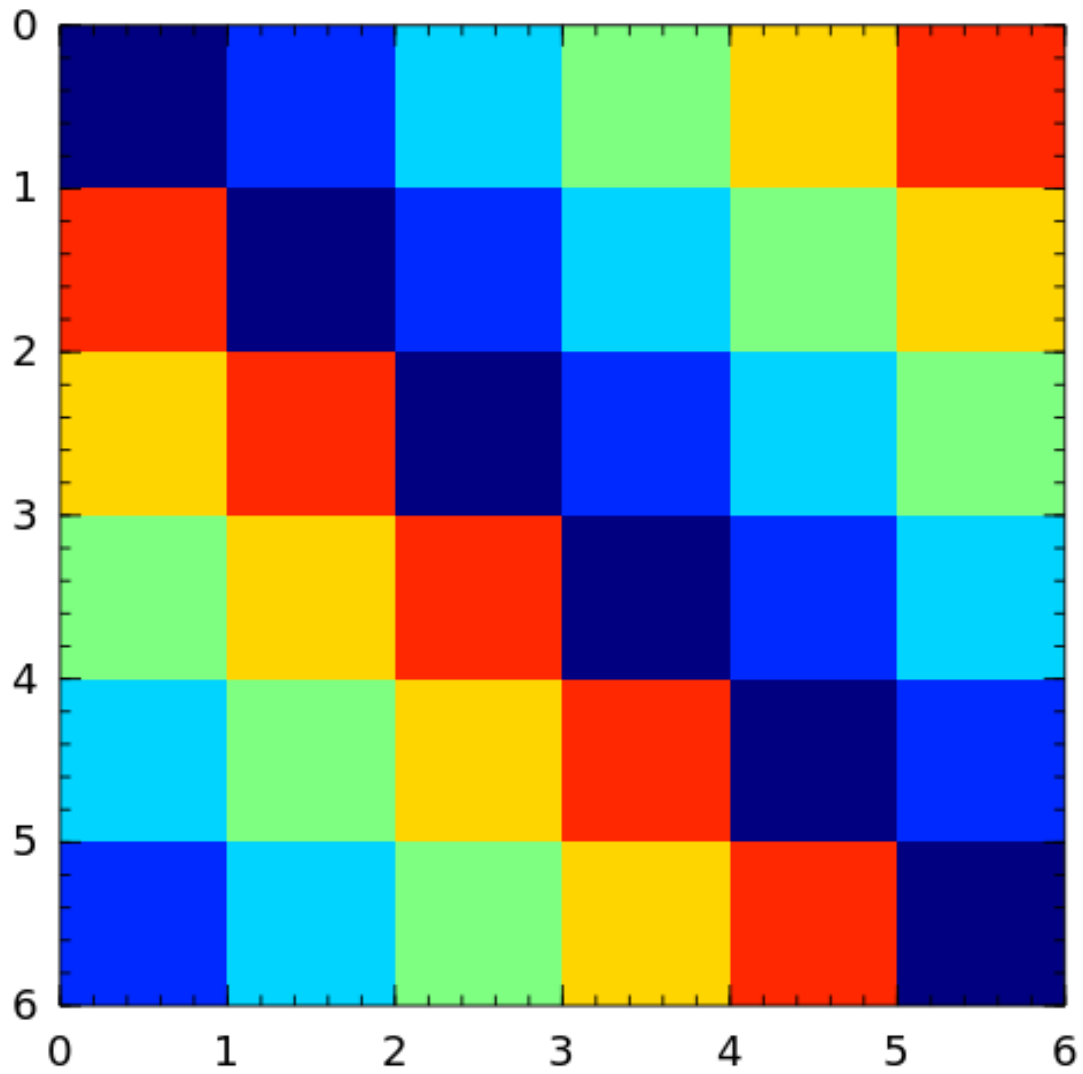
**chebspec** Chebyshev spectral differentiation matrix. If  $k = 0$ , the generated matrix is nilpotent and a vector with all one entries is a null vector. If  $k = 1$ , the generated matrix is nonsingular and well-conditioned. Its eigenvalues have negative real parts.



**chow** The Chow matrix is a singular Toeplitz lower Hessenberg matrix. The eigenvalues are known explicitly [chow69].

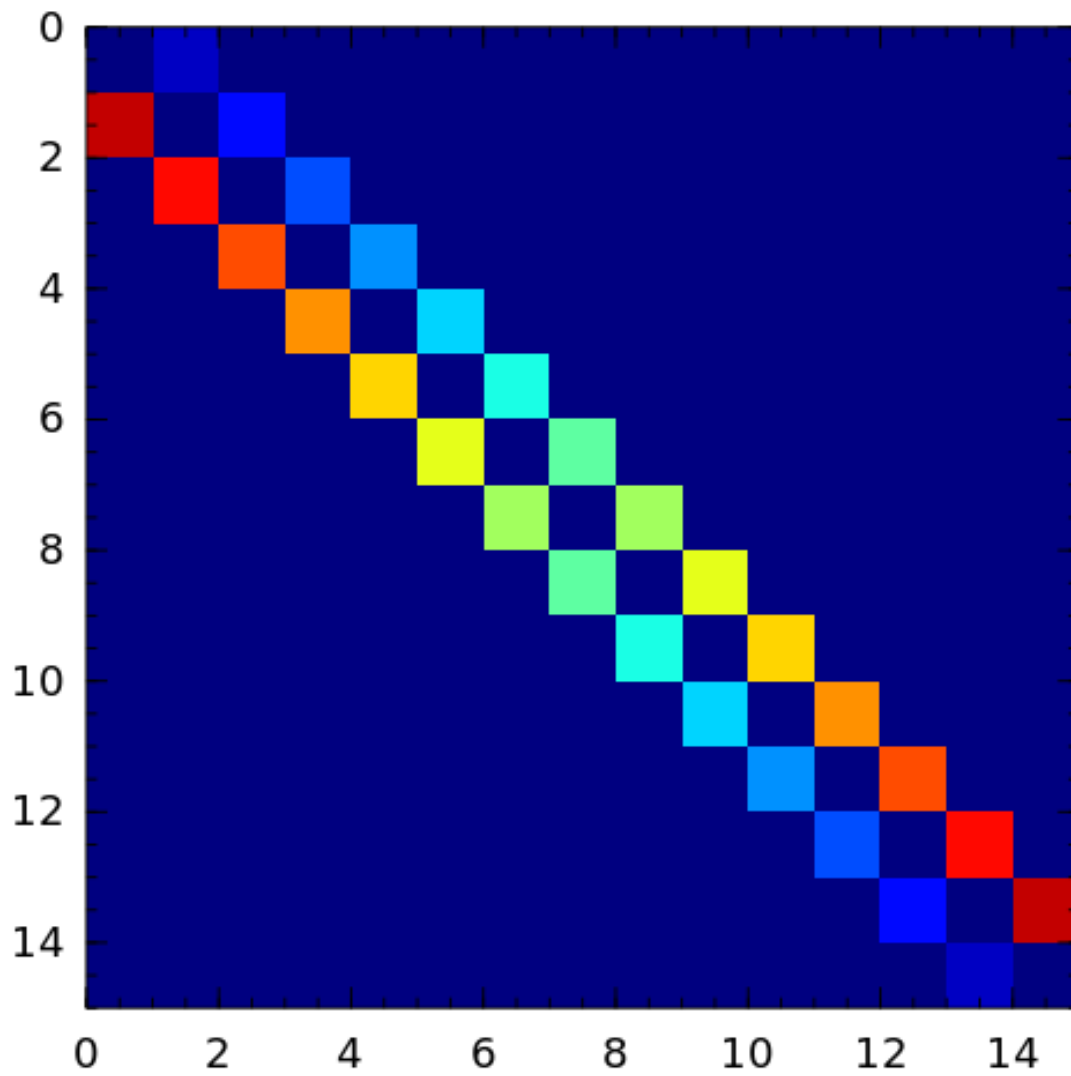


**circul** A circulant matrix has the property that each row is obtained by cyclically permuting the entries of the previous row one step forward.



**clement** The Clement matrix [*clem59*] is a Tridiagonal matrix with zero diagonal entries. If  $k = 1$ , the matrix is symmetric.

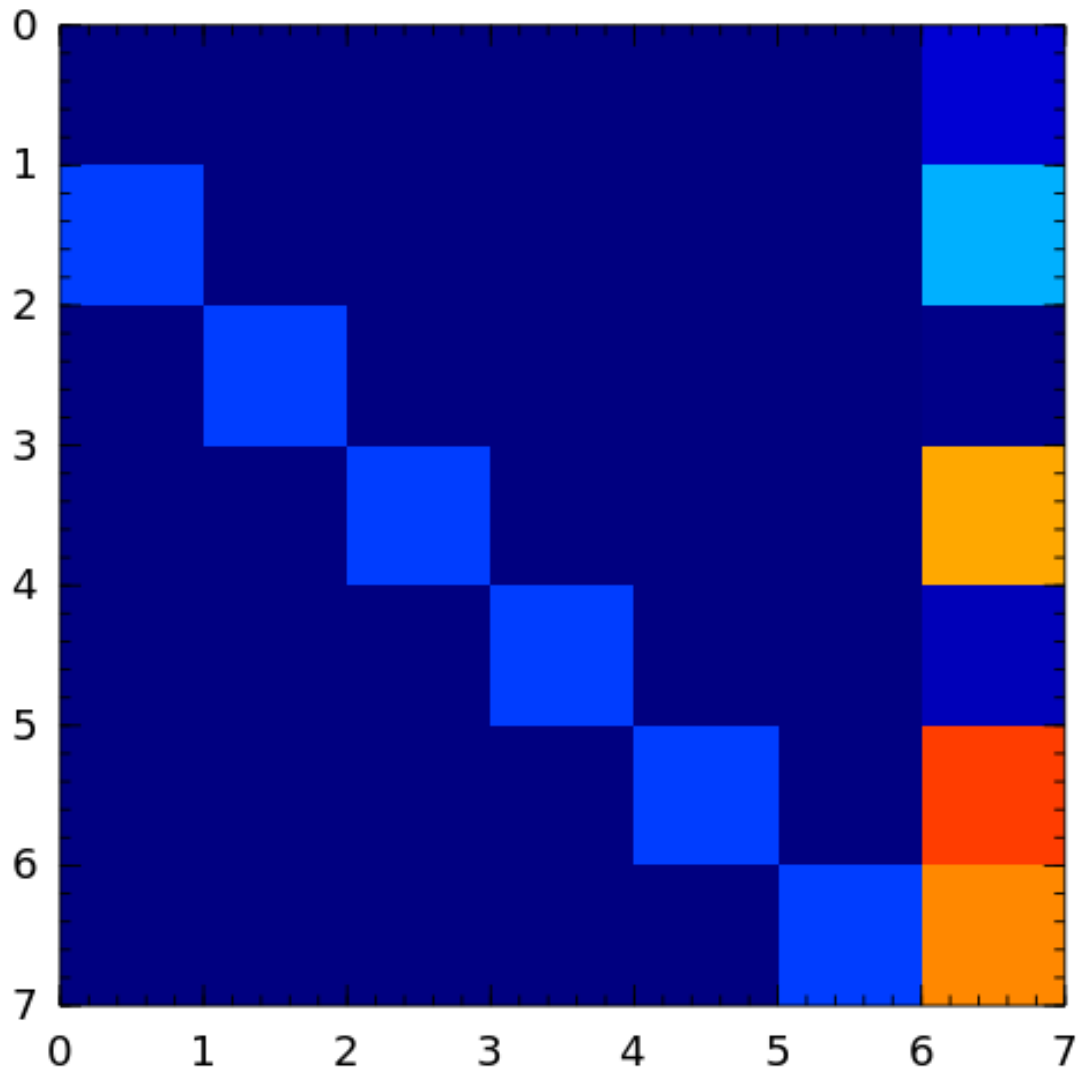




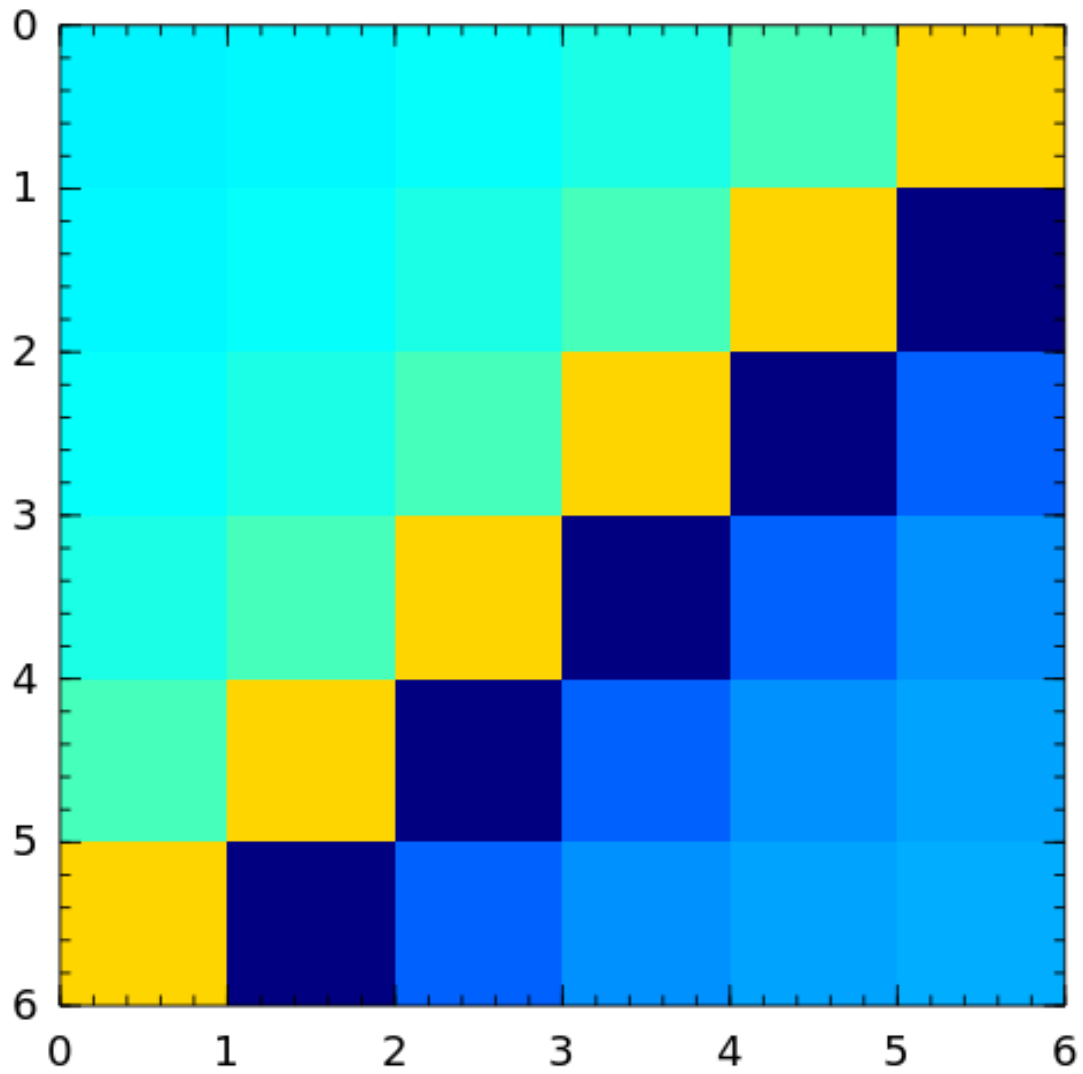
**companion** The companion matrix to a monic polynomial

$$a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} + x^n$$

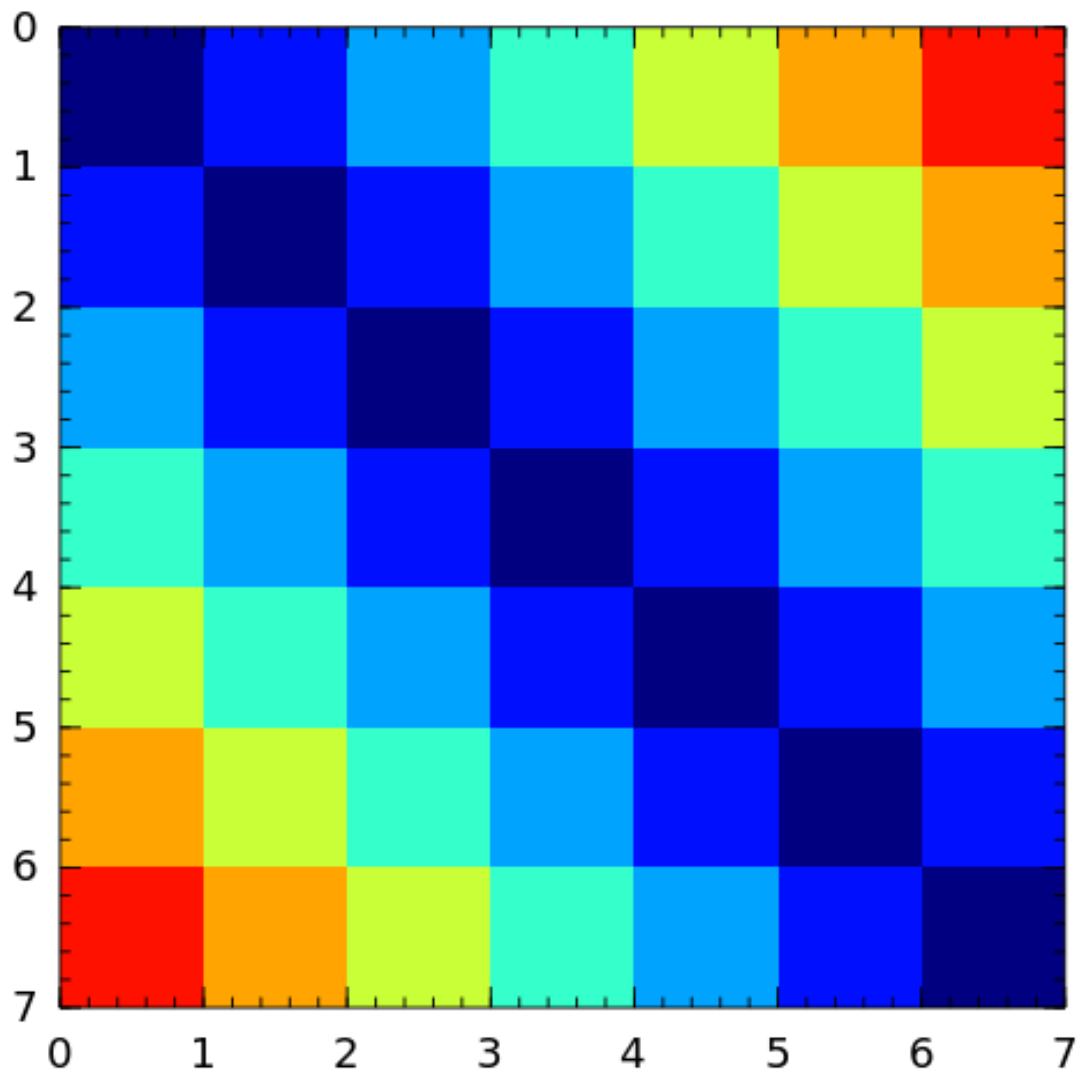
is the  $n$ -by- $n$  matrix with ones on the subdiagonal and the last column given by the coefficients of  $a(x)$ .



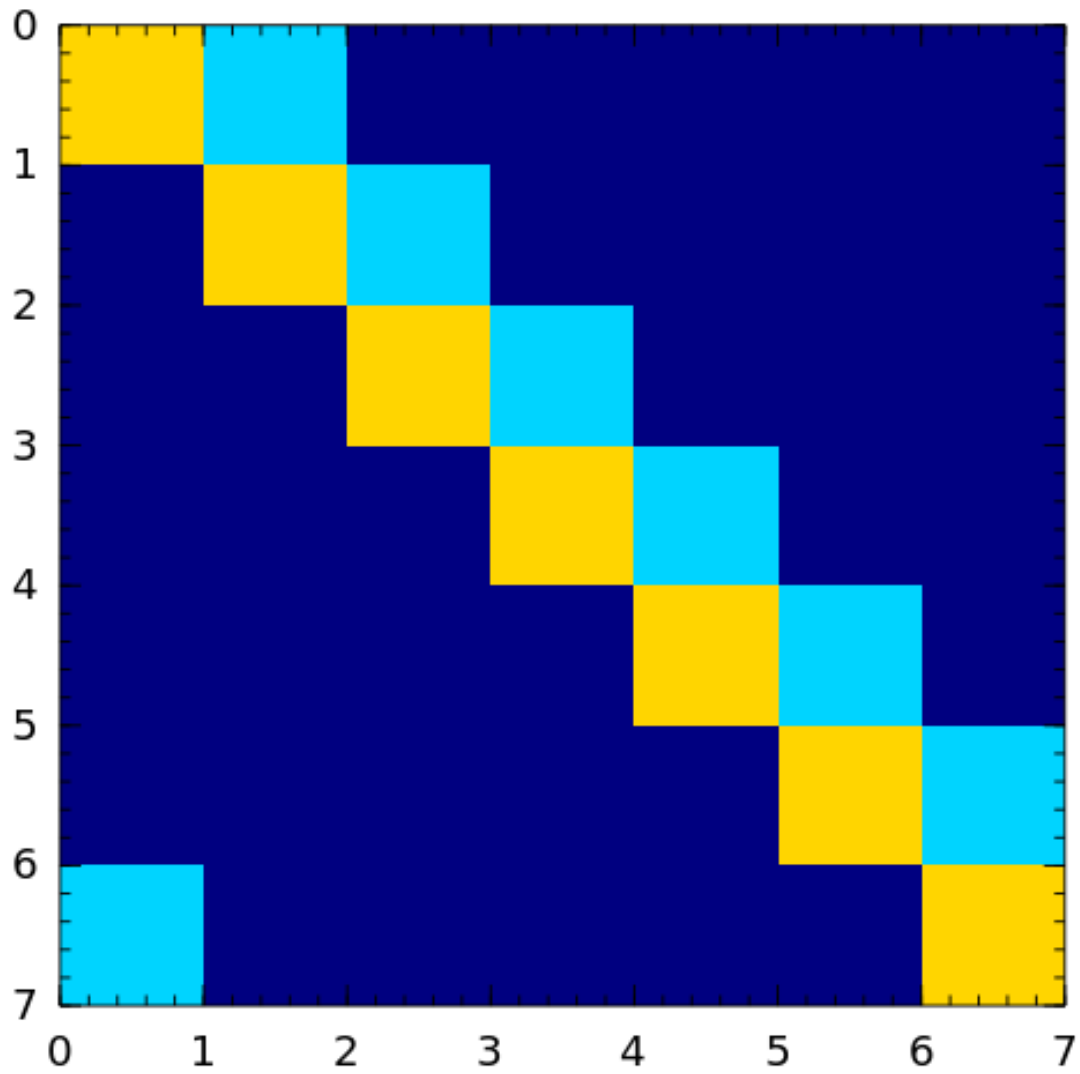
**dingdong** The Dingdong matrix is symmetric Hankel matrix invented by Dr. F. N. Ris of IBM, Thomas J Watson Research Centre. The eigenvalues cluster around  $\pi/2$  and  $-\pi/2$  [nash90].



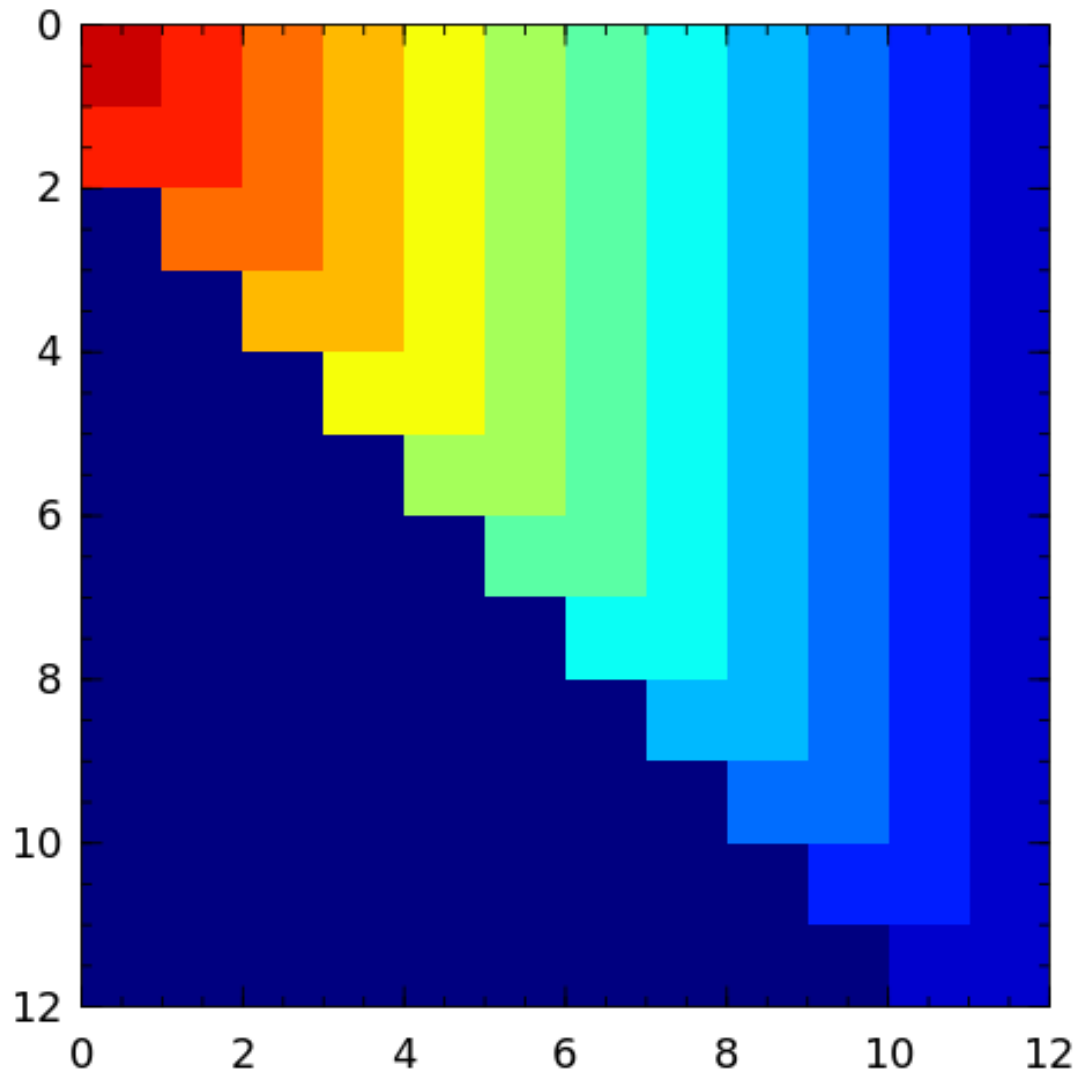
**fiedler** The Fiedler matrix is symmetric matrix with a dominant positive eigenvalue and all the other eigenvalues are negative. For explicit formulas for the inverse and determinant, see [\[todd77\]](#).



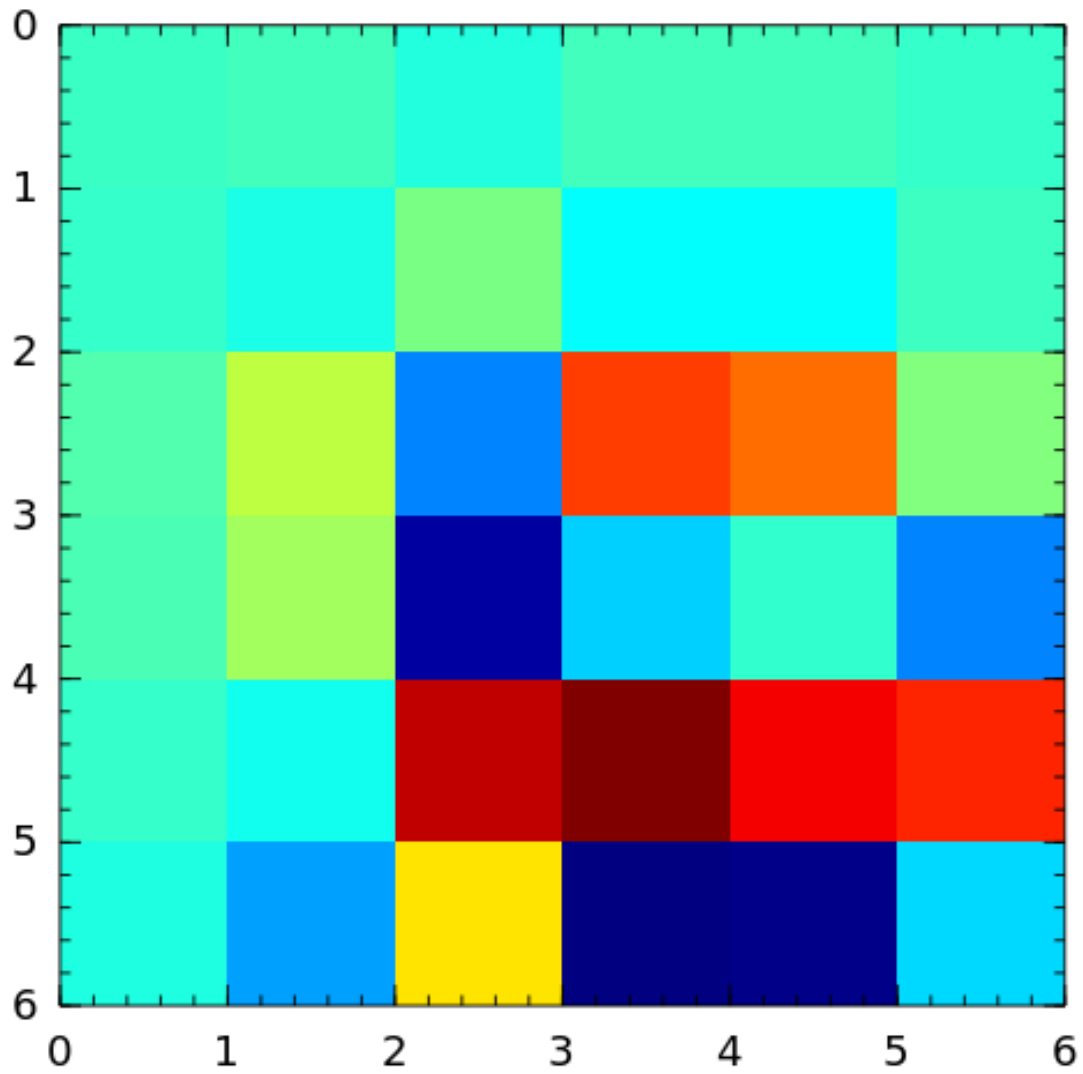
**forsythe** The Forsythe matrix is a  $n$ -by- $n$  perturbed Jordan block.



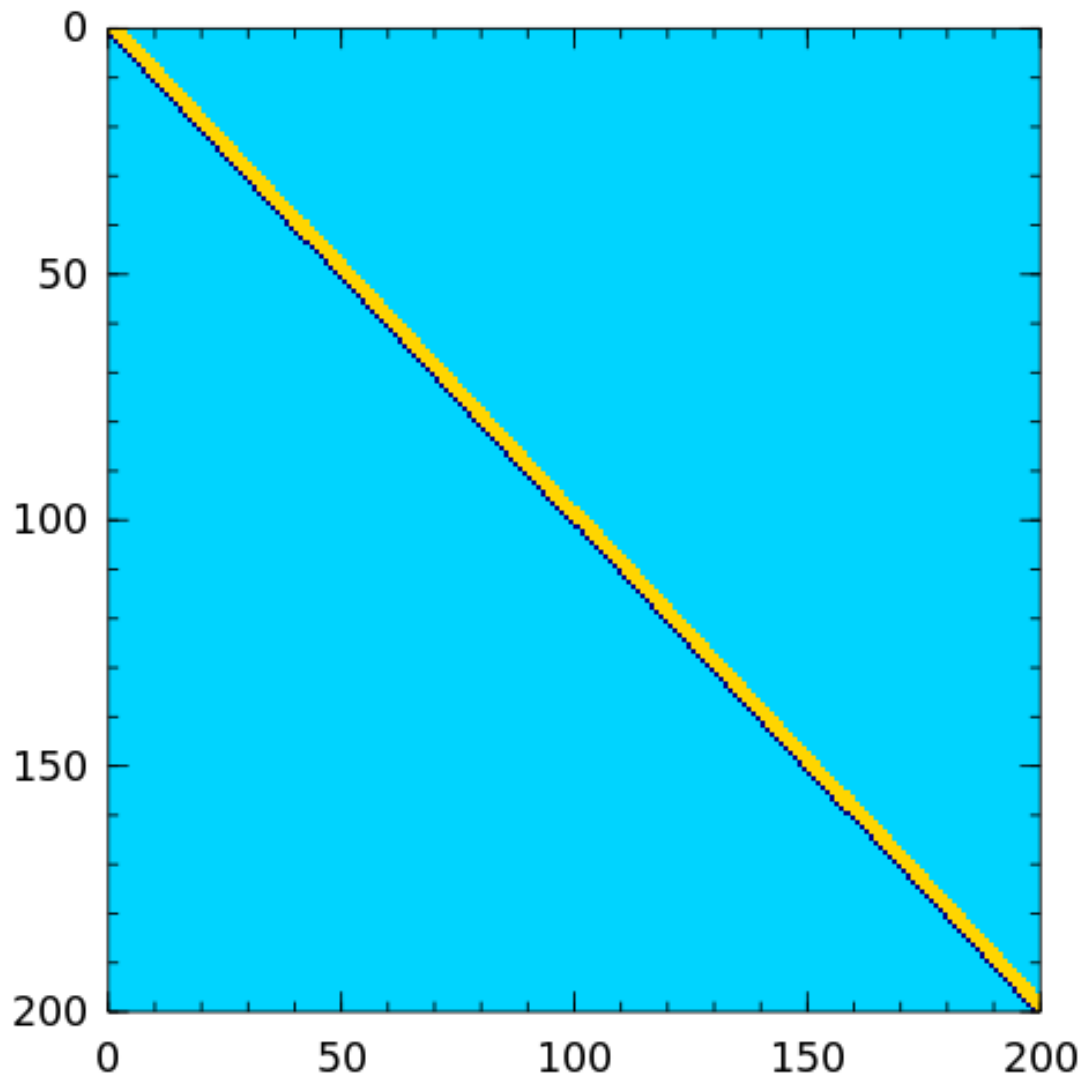
**frank** The Frank matrix is an upper Hessenberg matrix with determinant 1. The eigenvalues are real, positive and very ill conditioned [\[vara86\]](#).



**golub** Golub matrix is the product of two random unit lower and upper triangular matrices respectively. LU factorization without pivoting fails to reveal that such matrices are badly conditioned [*vistre98*].

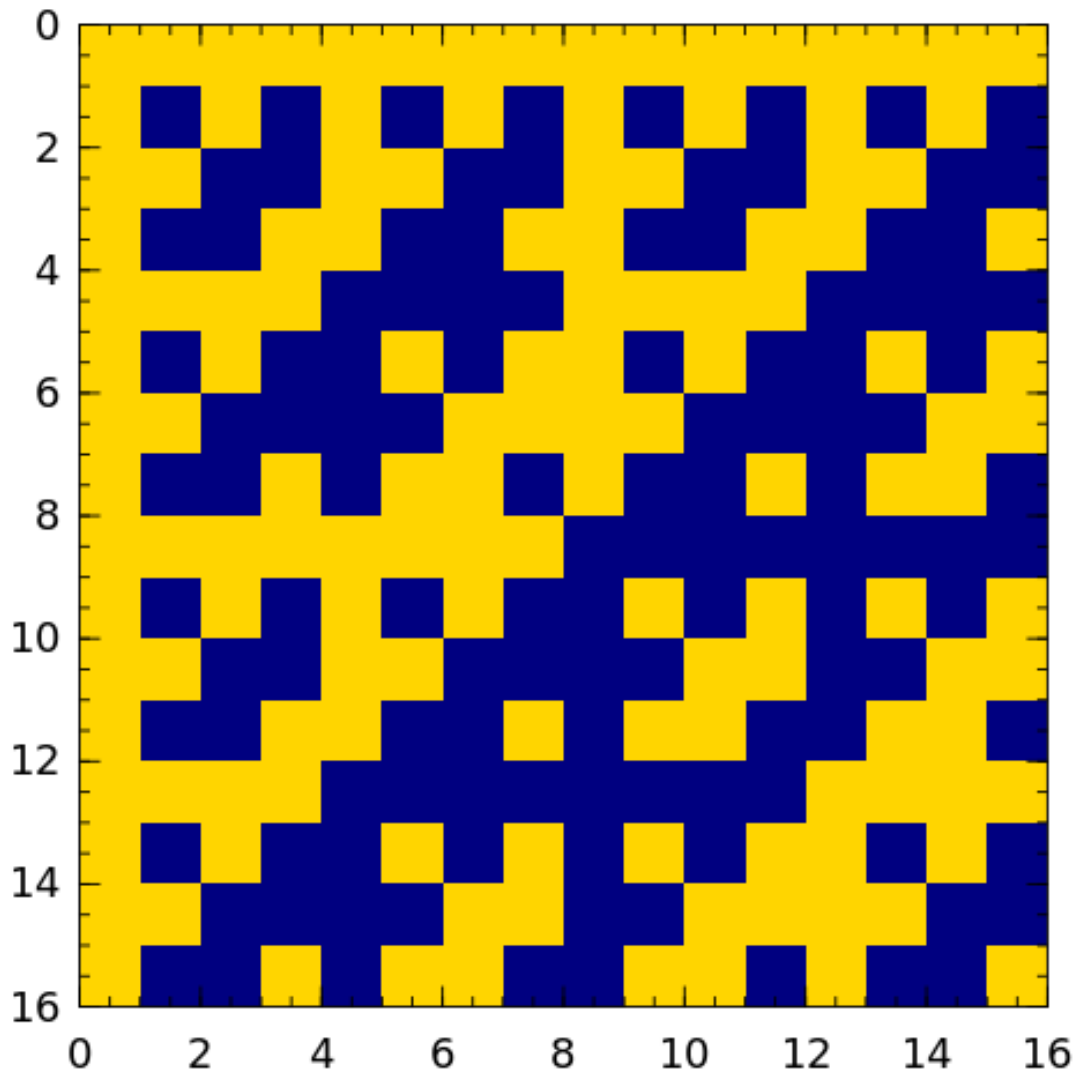


**grcar** The Grcar matrix is a Toeplitz matrix with sensitive eigenvalues. The image below is a 200-by-200 Grcar matrix used in [\[nrt92\]](#).



**hadamard** The Hadamard matrix is a square matrix whose entries are 1 or -1. It was named after Jacques Hadamard. The rows of a Hadamard matrix are orthogonal.

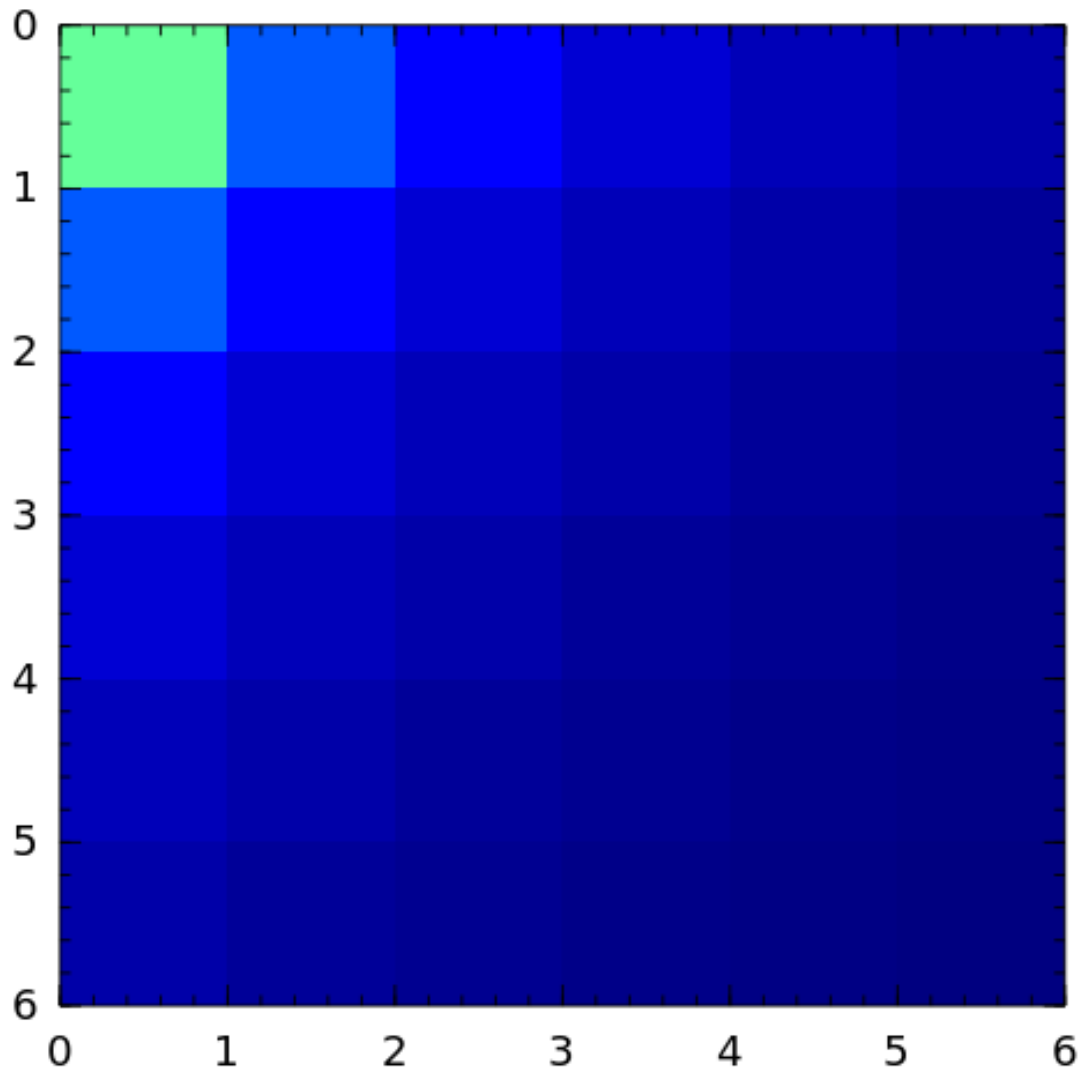




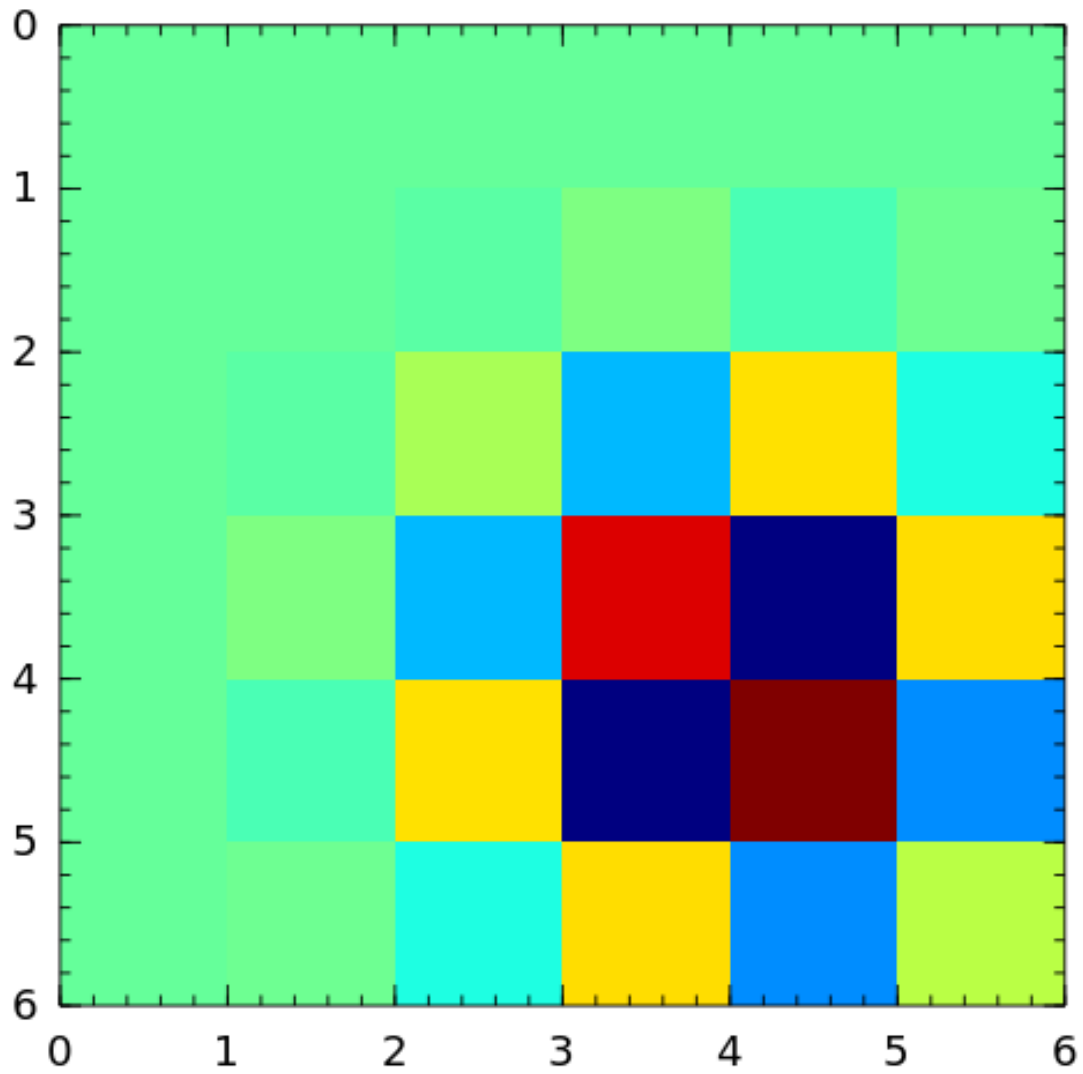
**hankel** Hankel matrix is a matrix that is symmetric and constant across the anti-diagonals. For example:

```
julia> matrixdepot("hankel", [1,2,3,4], [7,8,9,10])
4x4 Array{Float64,2}:
1.0  2.0  3.0  4.0
2.0  3.0  4.0  8.0
3.0  4.0  8.0  9.0
4.0  8.0  9.0  10.0
```

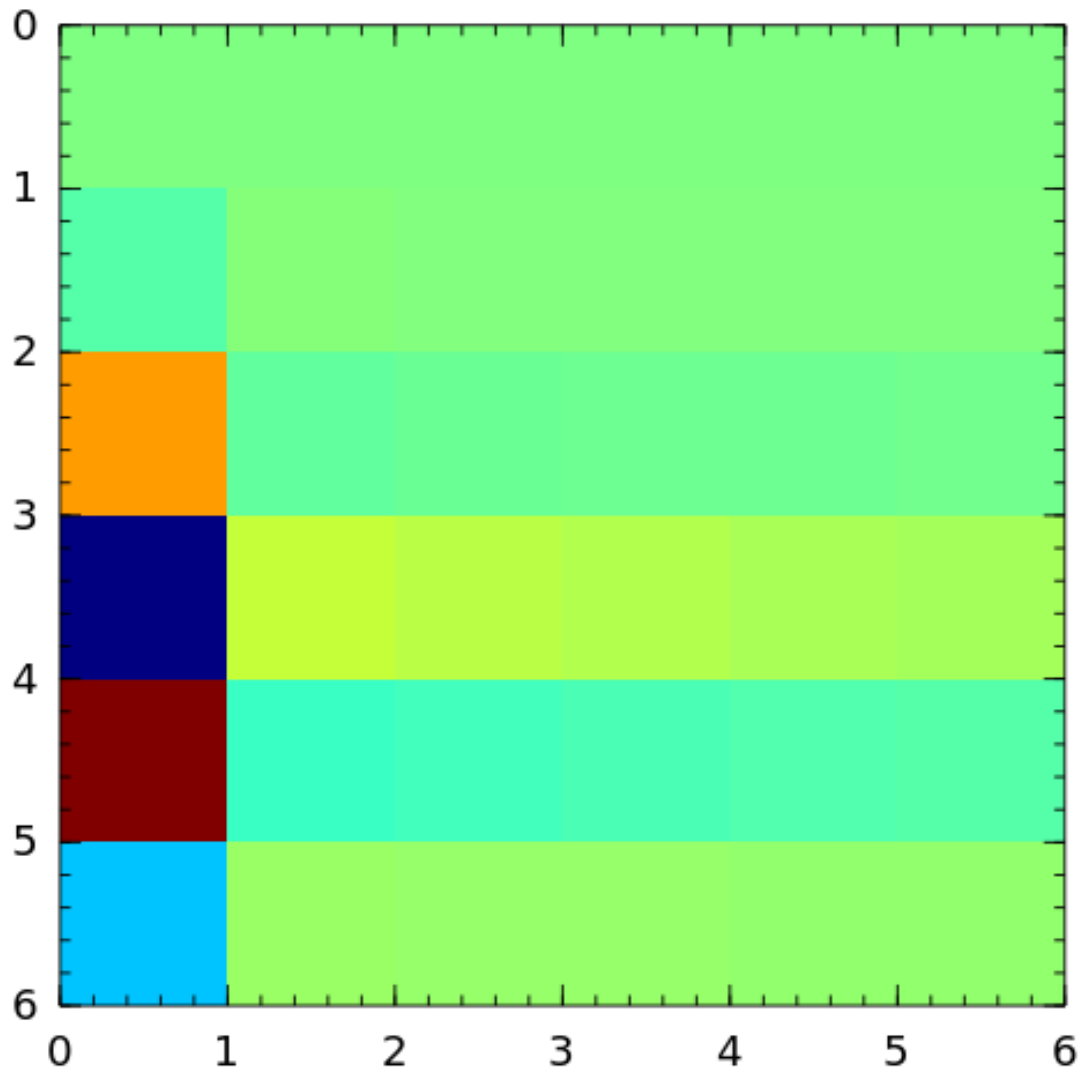
**hilb** The Hilbert matrix is a very ill conditioned matrix. But it is symmetric positive definite and totally positive so it is not a good test matrix for Gaussian elimination [\[high02\]](#) (Sec. 28.1).



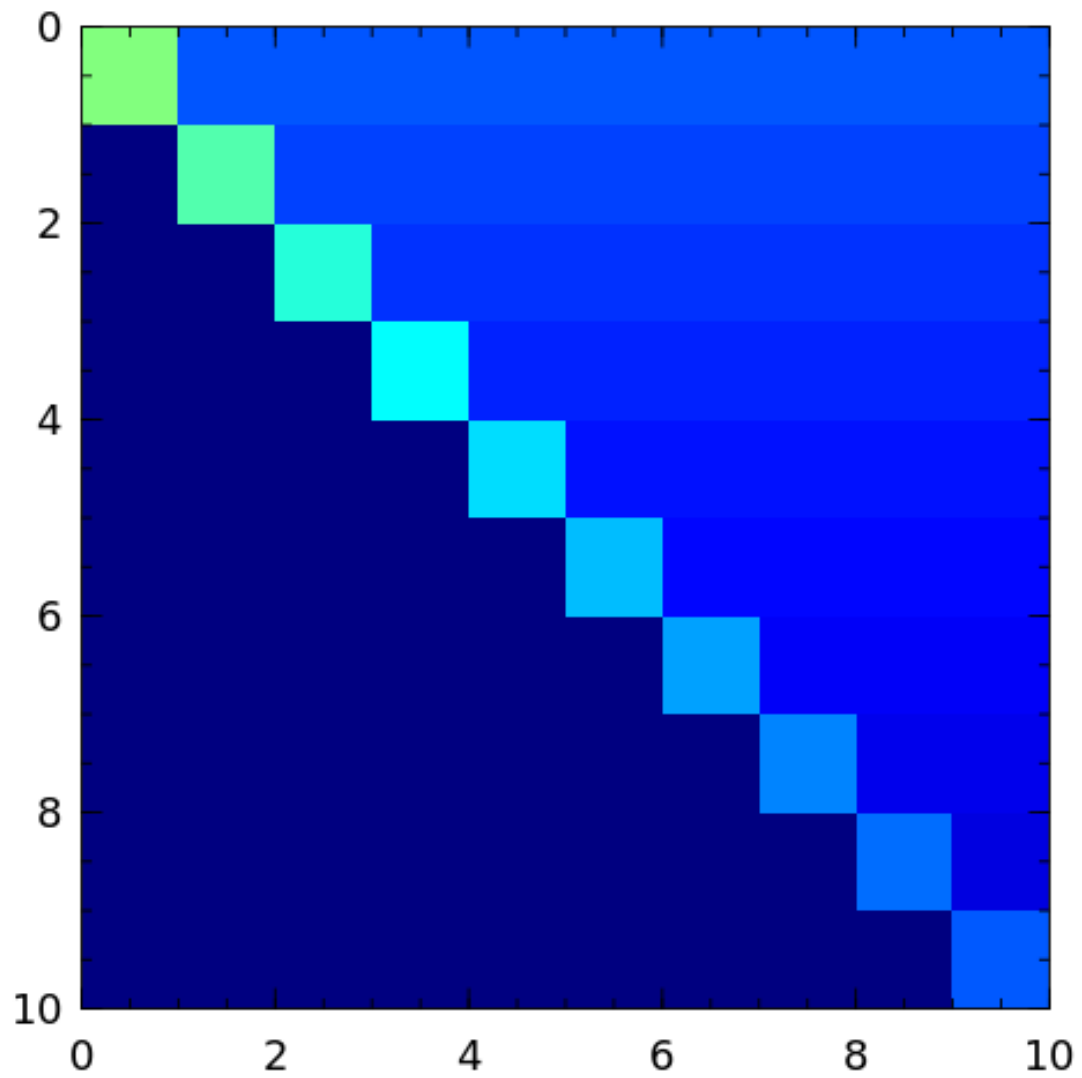
**invhilb** Inverse of the Hilbert Matrix.



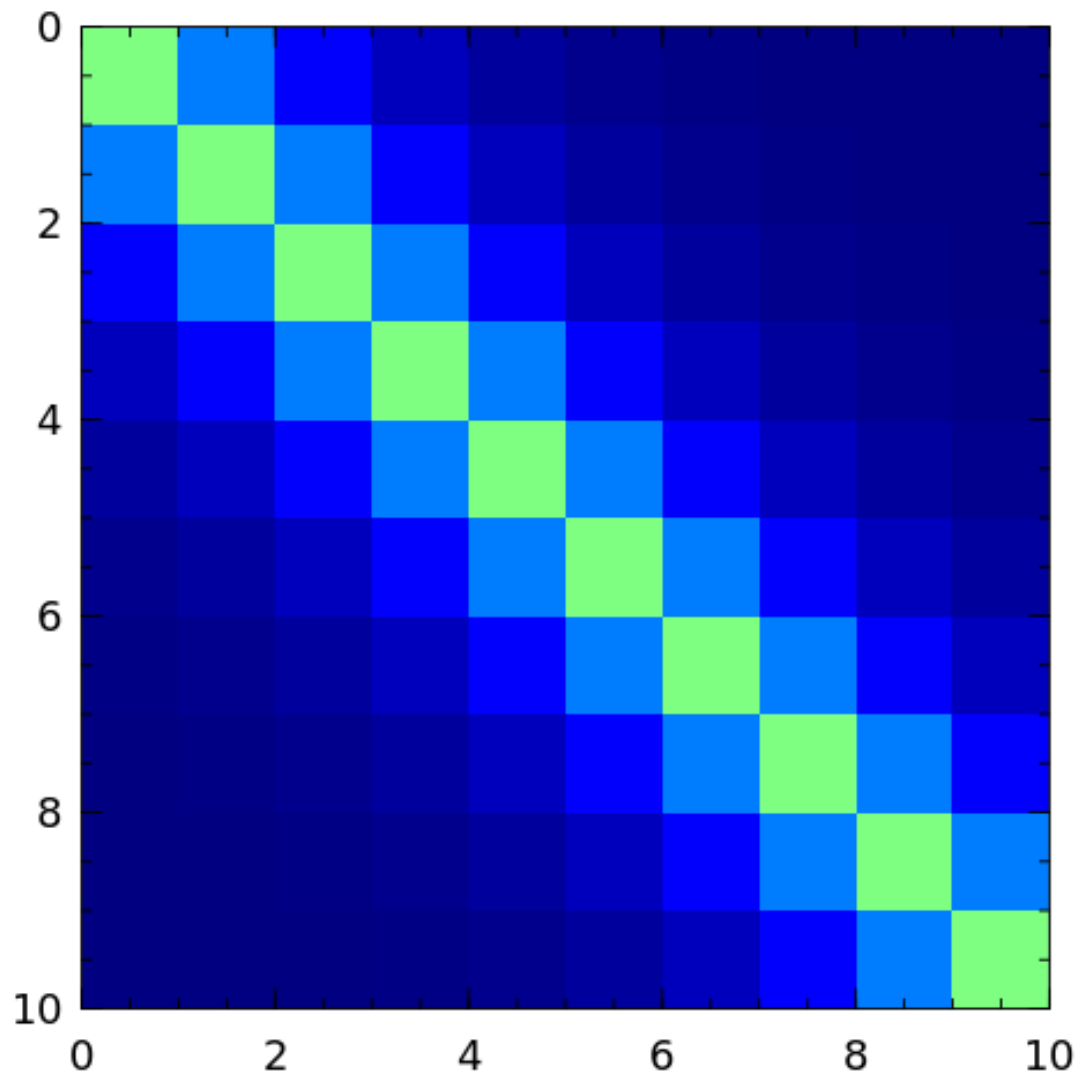
**invol** An involutory matrix, i.e., a matrix that is its own inverse. See [\[hoca63\]](#).



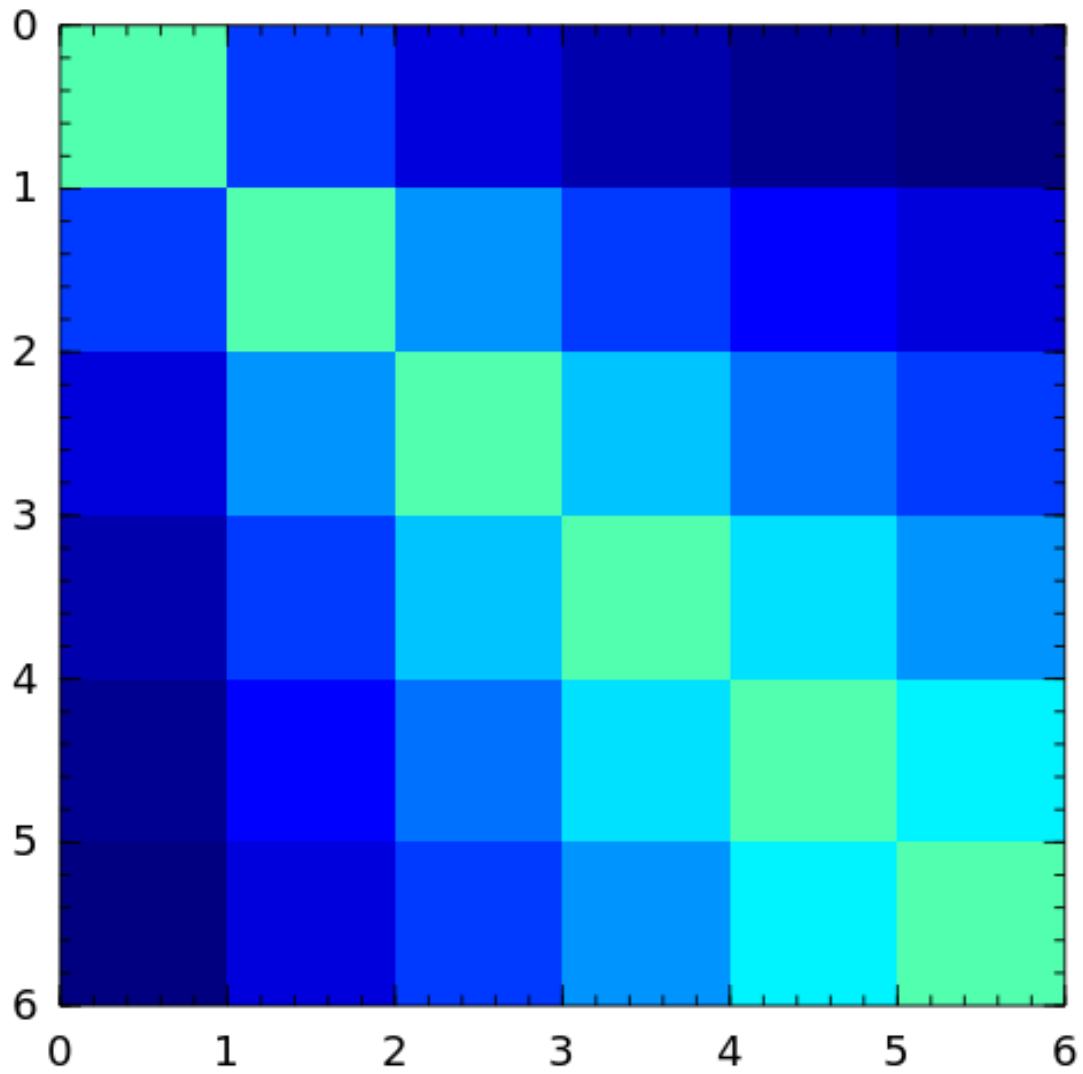
**kahan** The Kahan matrix is a upper trapezoidal matrix, i.e., the  $(i, j)$  element is equal to 0 if  $i > j$ . The useful range of  $\theta$  is  $0 < \theta < \pi$ . The diagonal is perturbed by `pert*eps()*diagm([n:-1:1])`.



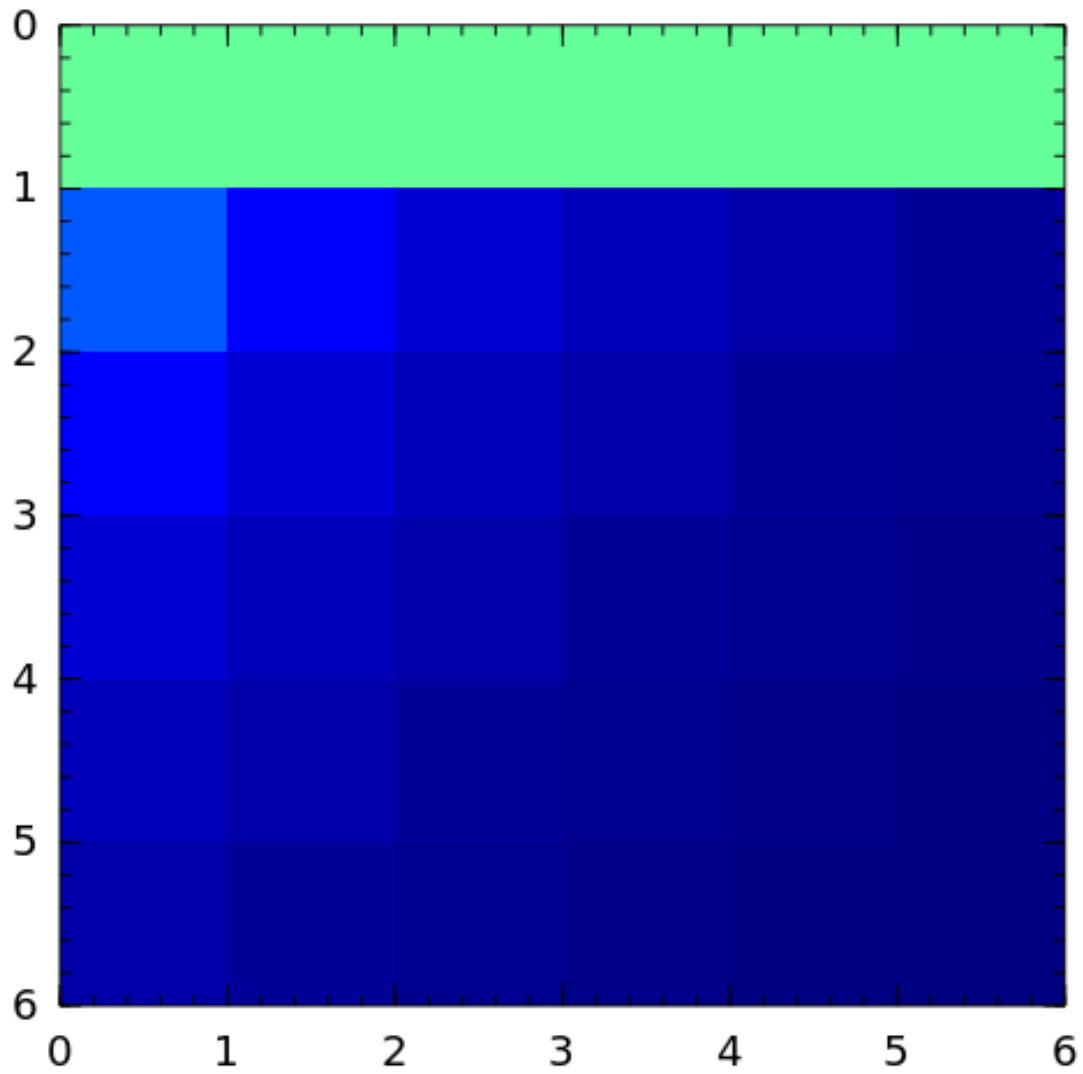
**kms** Kac-Murdock-Szego Toeplitz matrix [*tren89*].



**lehmer** The Lehmer matrix is a symmetric positive definite matrix. It is totally nonnegative. The inverse is tridiagonal and explicitly known [\[neto58\]](#).

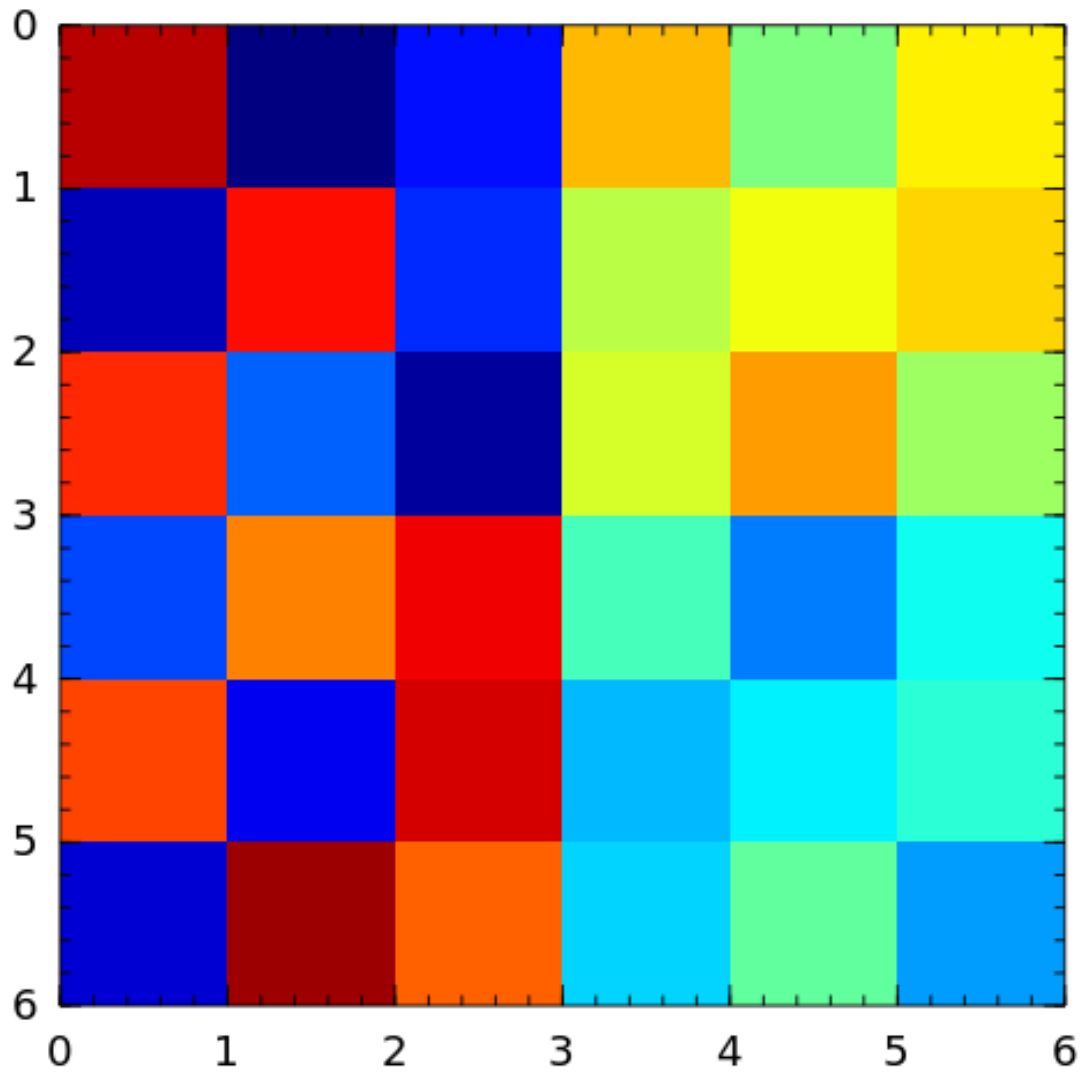


**lotkin** The Lotkin matrix is the Hilbert matrix with its first row altered to all ones. It is unsymmetric, ill-conditioned and has many negative eigenvalues of small magnitude [*lotk55*].

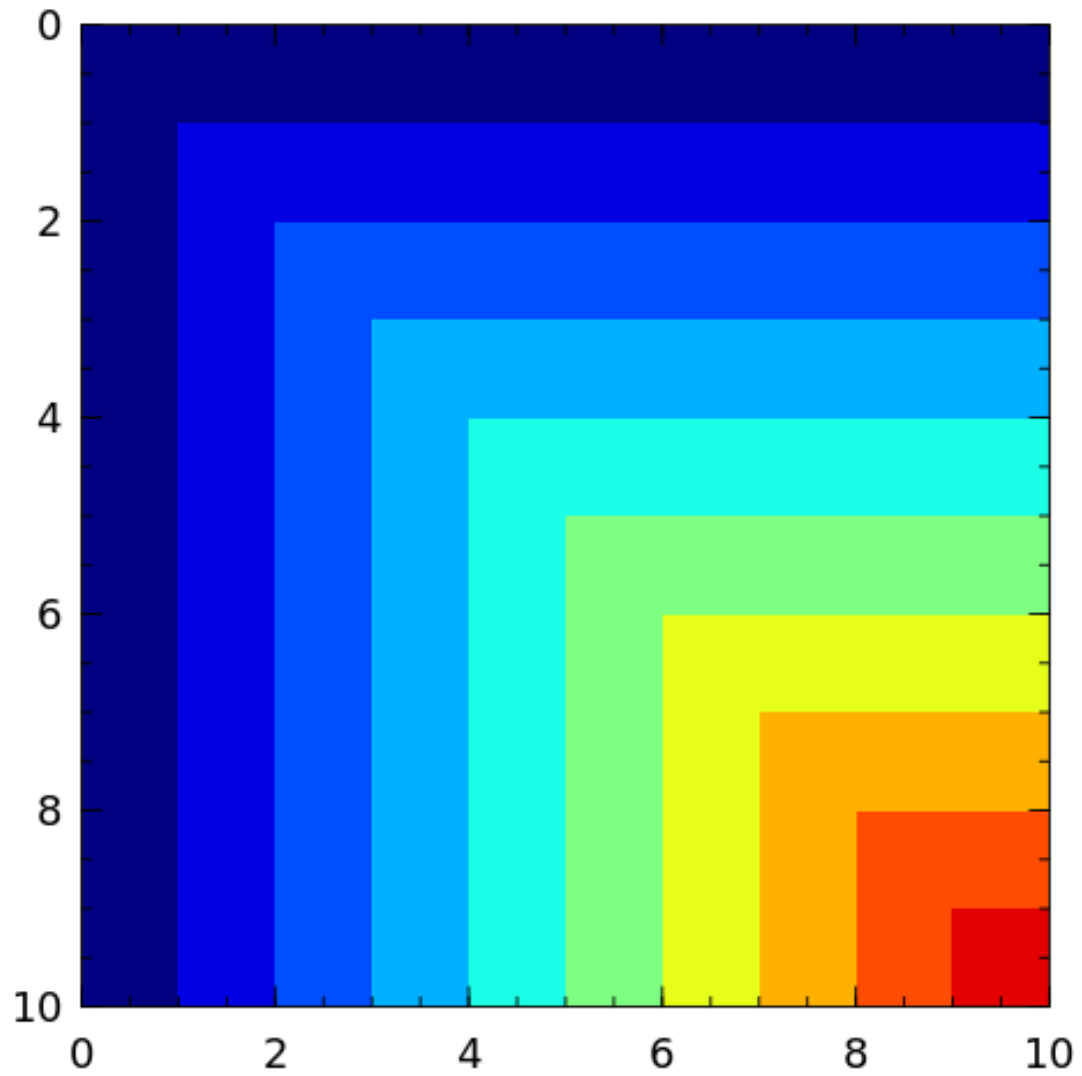


**magic** The magic matrix is a matrix with integer entries such that the row elements, column elements, diagonal elements and anti-diagonal elements all add up to the same number.

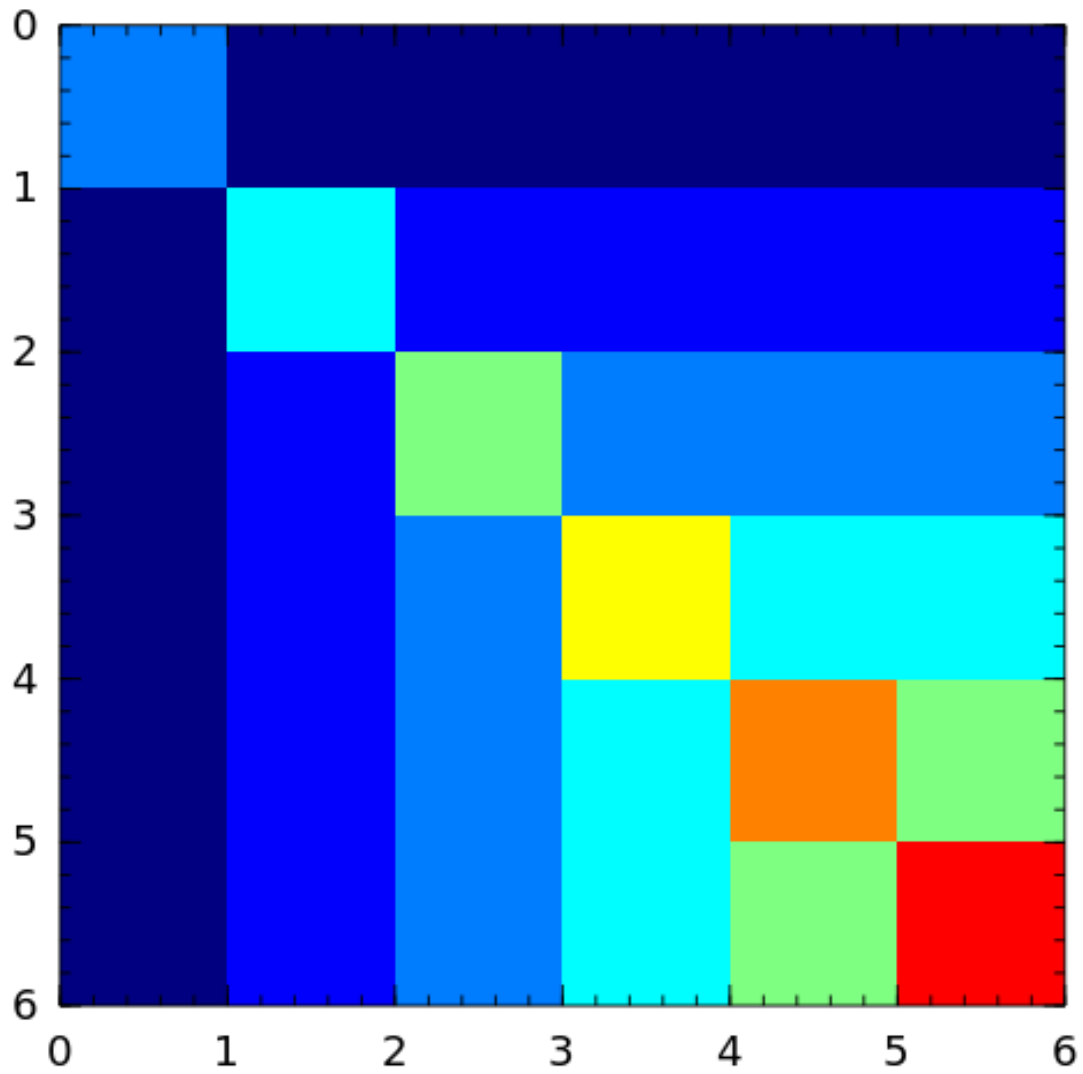




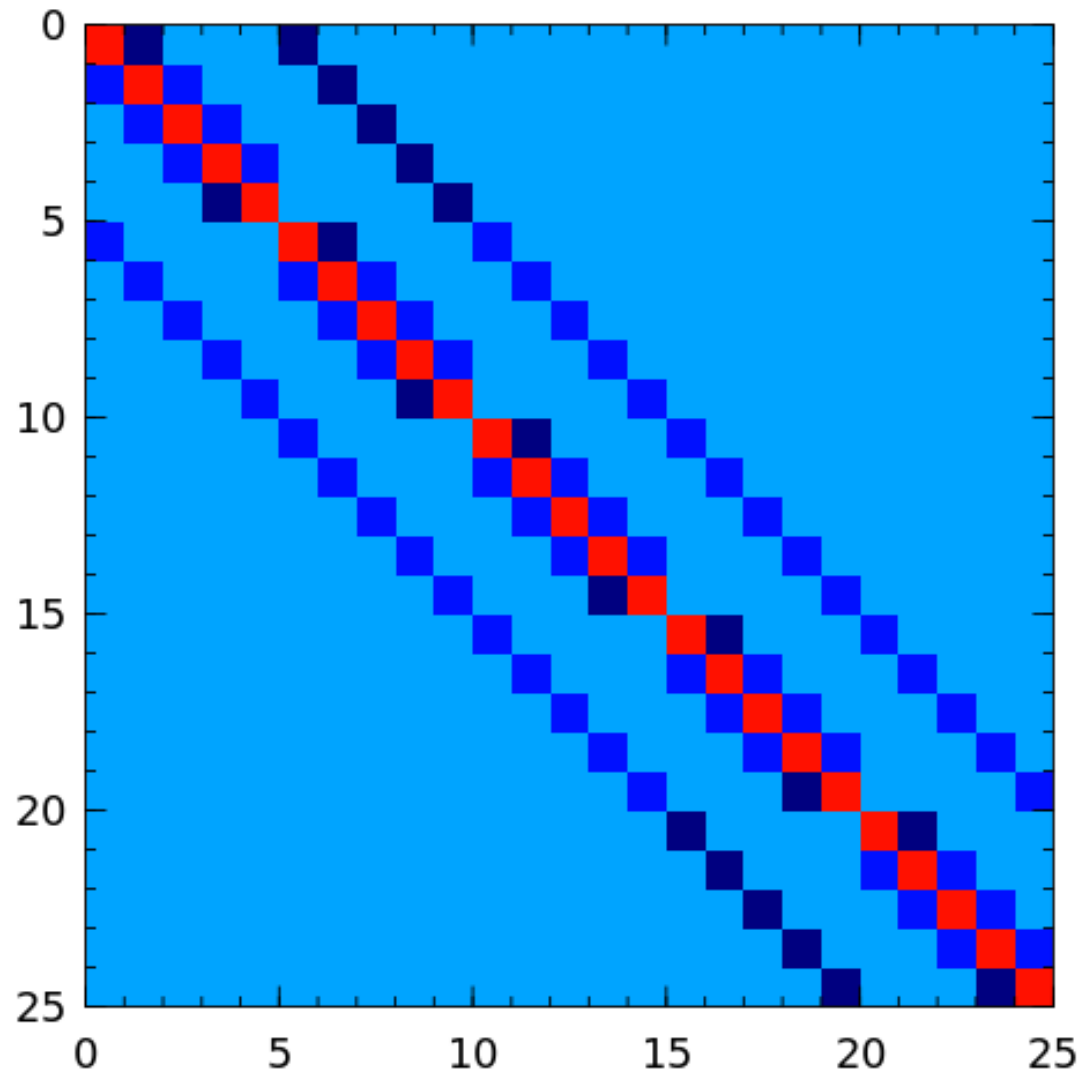
**minij** A matrix with  $(i, j)$  entry  $\min(i, j)$ . It is a symmetric positive definite matrix. The eigenvalues and eigenvectors are known explicitly. Its inverse is tridiagonal.



**moler** The Moler matrix is a symmetric positive definite matrix. It has one small eigenvalue.



**neumann** A singular matrix from the discrete Neumann problem. This matrix is sparse and the null space is formed by a vector of ones [\[plem76\]](#).



**oscillate** A matrix  $A$  is called oscillating if  $A$  is totally nonnegative and if there exists an integer  $q > 0$  such that  $A^q$  is totally positive. An  $n \times n$  oscillating matrix  $A$  satisfies:

1.  $A$  has  $n$  distinct and positive eigenvalues  $\lambda_1 > \lambda_2 > \dots > \lambda_n > 0$ .
2. The  $i$ th eigenvector, corresponding to  $\lambda_i$  in the above ordering, has exactly  $i - 1$  sign changes.

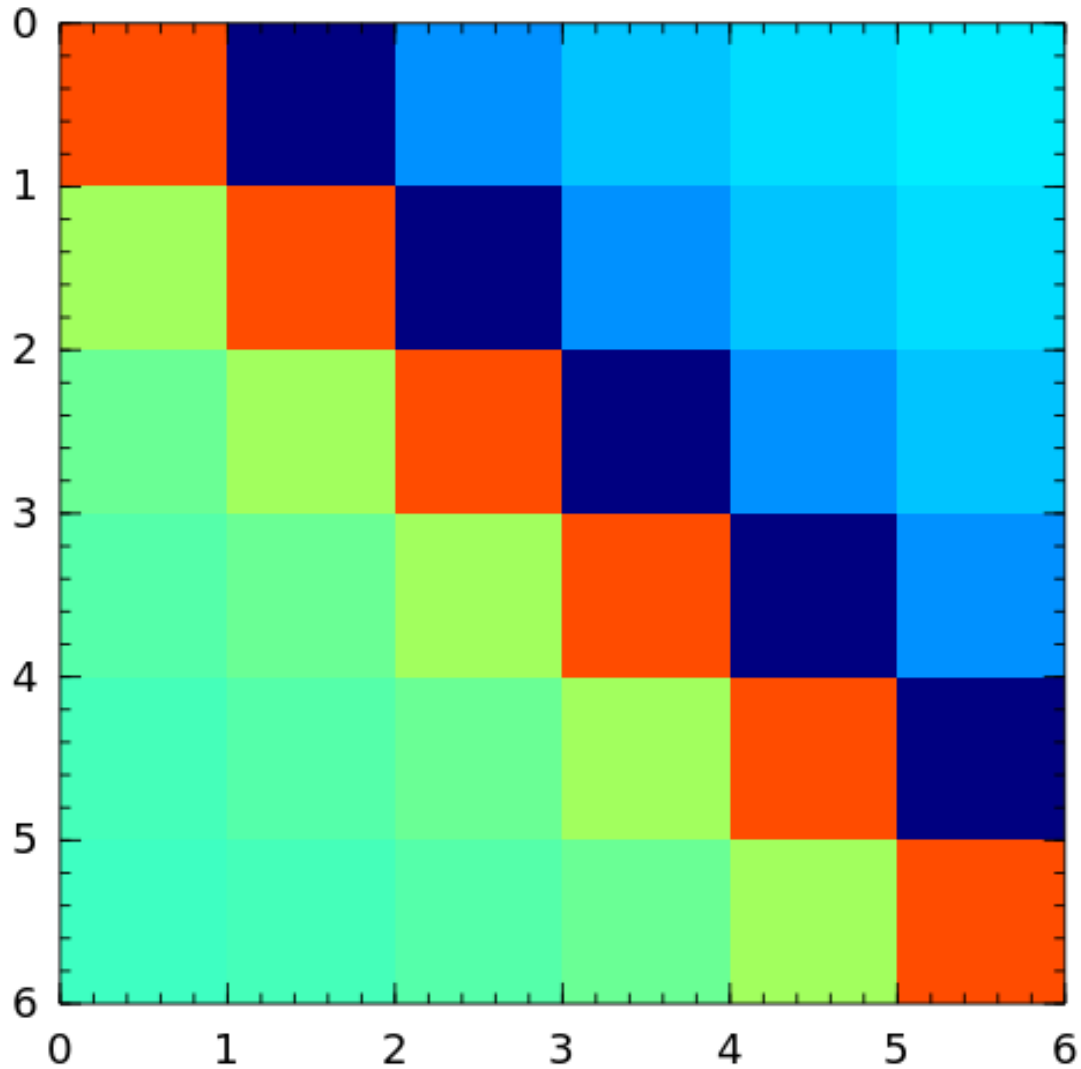
This function generates a symmetric oscillating matrix, which is useful for testing numerical regularization methods [hansen95]. For example:

```
julia> A = matrixdepot("oscillate", 3)
3x3 Array{Float64,2}:
0.98694  0.112794  0.0128399
0.112794  0.0130088  0.0014935
0.0128399  0.0014935  0.00017282

julia> eig(A)
([1.4901161192617526e-8,0.00012207031249997533,0.9999999999999983],
```

```
3x3 Array{Float64,2}:
 0.0119607  0.113658 -0.993448
-0.215799  -0.969813 -0.113552
 0.976365  -0.215743 -0.0129276
```

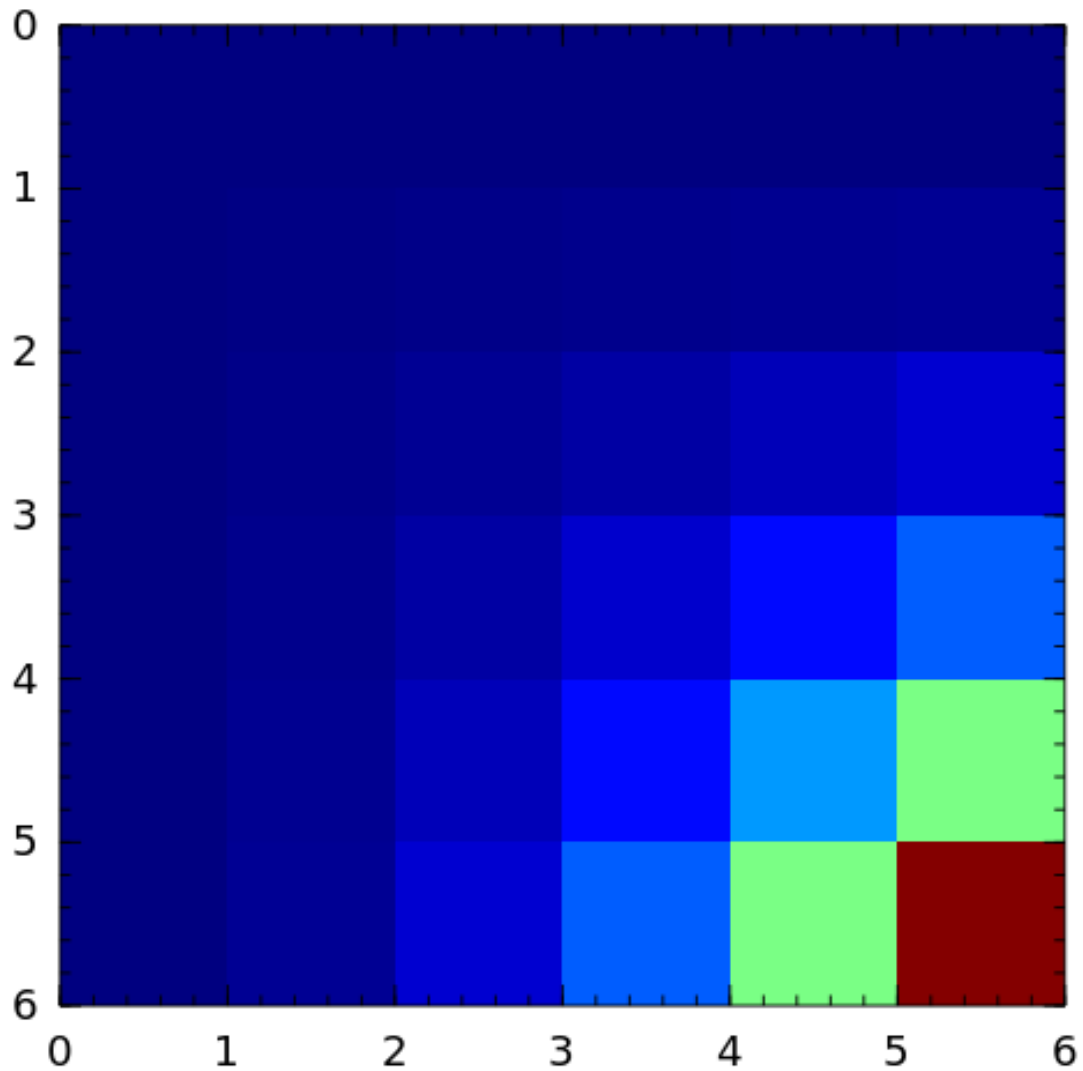
**parter** The Parter matrix is a Toeplitz and Cauchy matrix with singular values near  $\pi$  [part86].



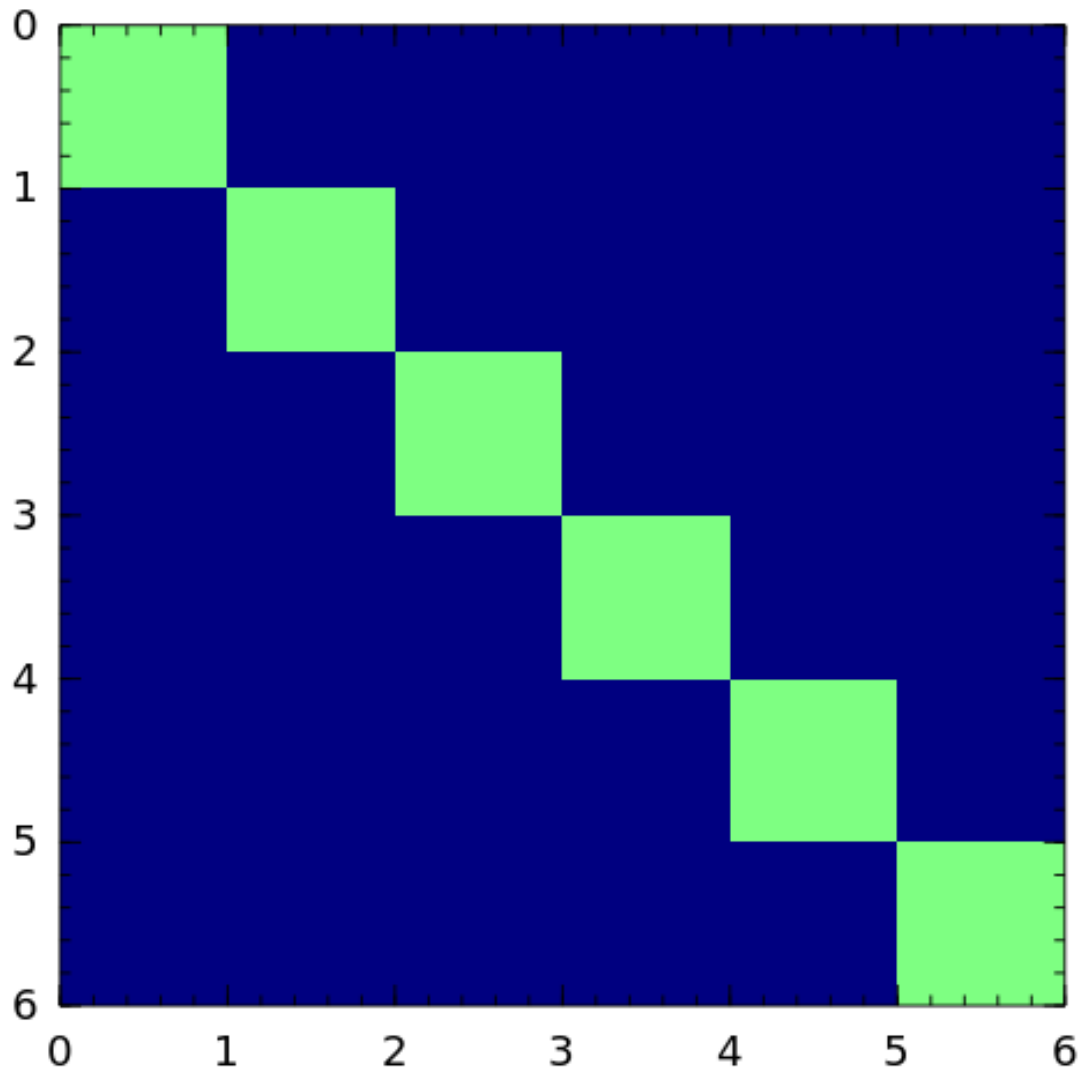
**pascal** The Pascal matrix's anti-diagonals form the Pascal's triangle:

```
julia> matrixdepot("pascal", 6)
6x6 Array{Int64,2}:
 1  1  1  1  1  1
 1  2  3  4  5  6
 1  3  6 10 15 21
 1  4 10 20 35 56
 1  5 15 35 70 126
 1  6 21 56 126 252
```

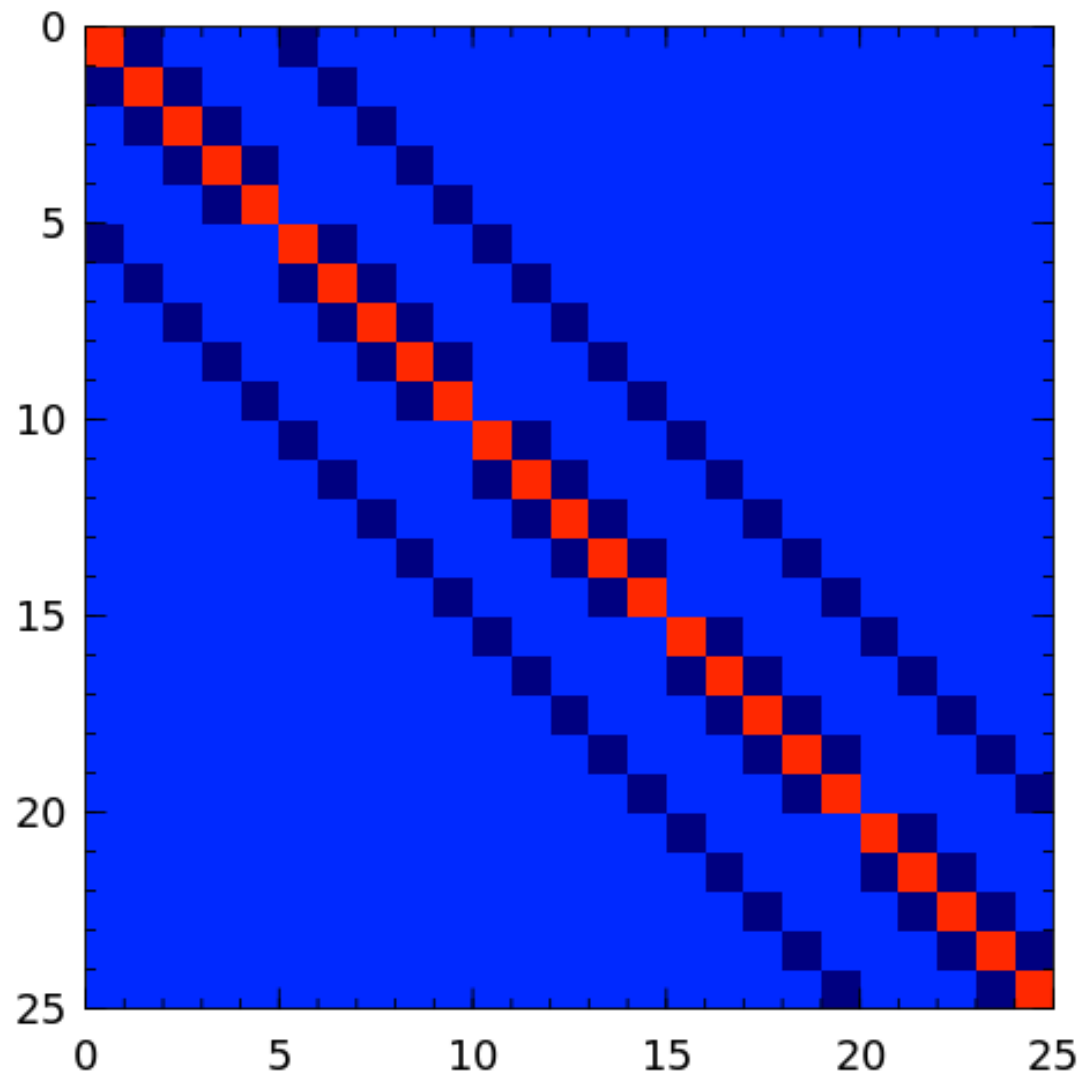
See [high02] (28.4).



**pei** The Pei matrix is a symmetric matrix with known inverse [pei62].



**poisson** A block tridiagonal matrix from Poisson's equation. This matrix is sparse, symmetric positive definite and has known eigenvalues.



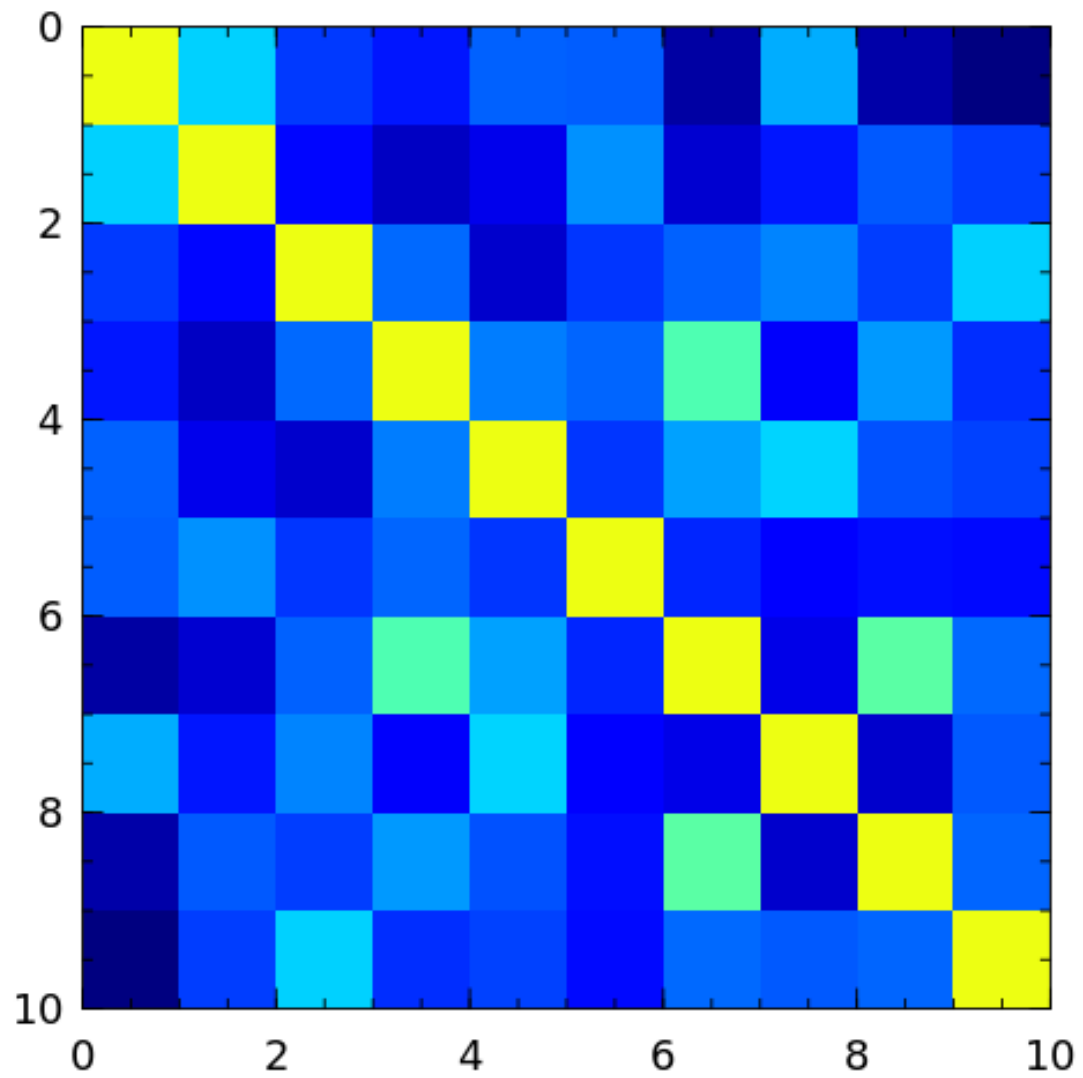
**prolate** A prolate matrix is a symmetric ill-conditioned Toeplitz matrix

$$A = \begin{bmatrix} a_0 & a_1 & \cdots \\ a_1 & a_0 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

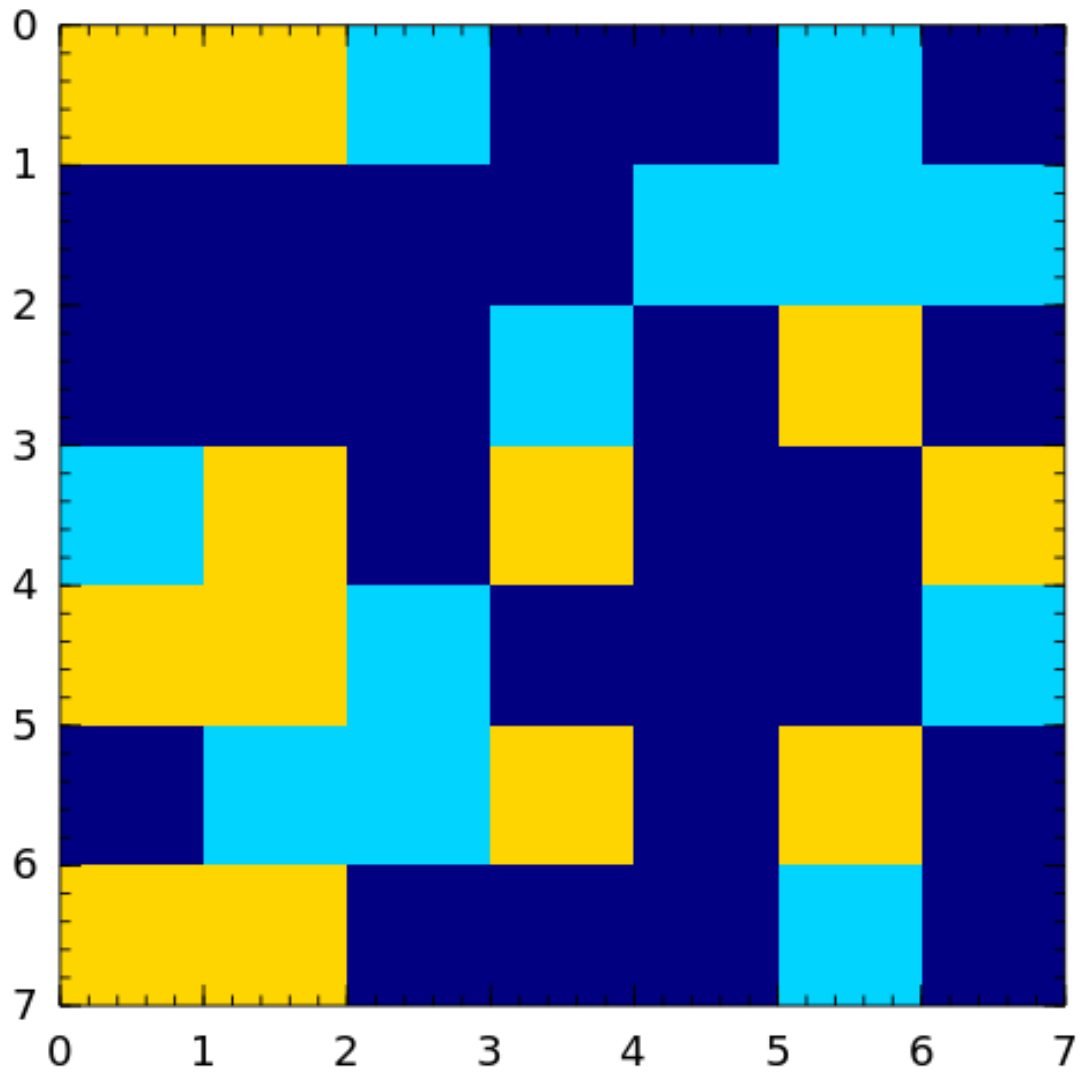
such that  $a_0 = 2w$  and  $a_k = (\sin 2\pi wk)/\pi k$  for  $k = 1, 2, \dots$  and  $0 < w < 1/2$  [varah93].

**randcorr** A random correlation matrix is a symmetric positive semidefinite matrix with 1s on the diagonal.

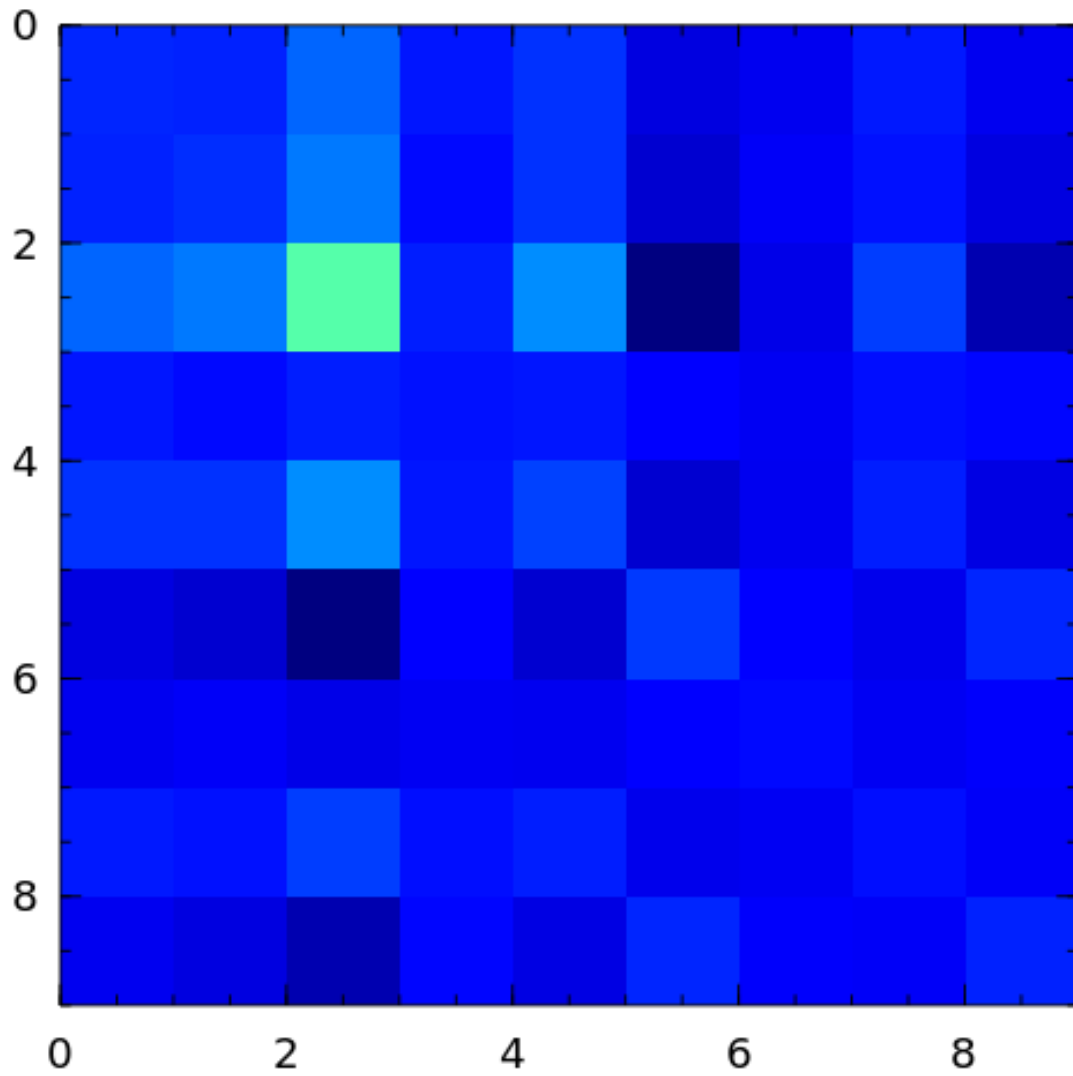




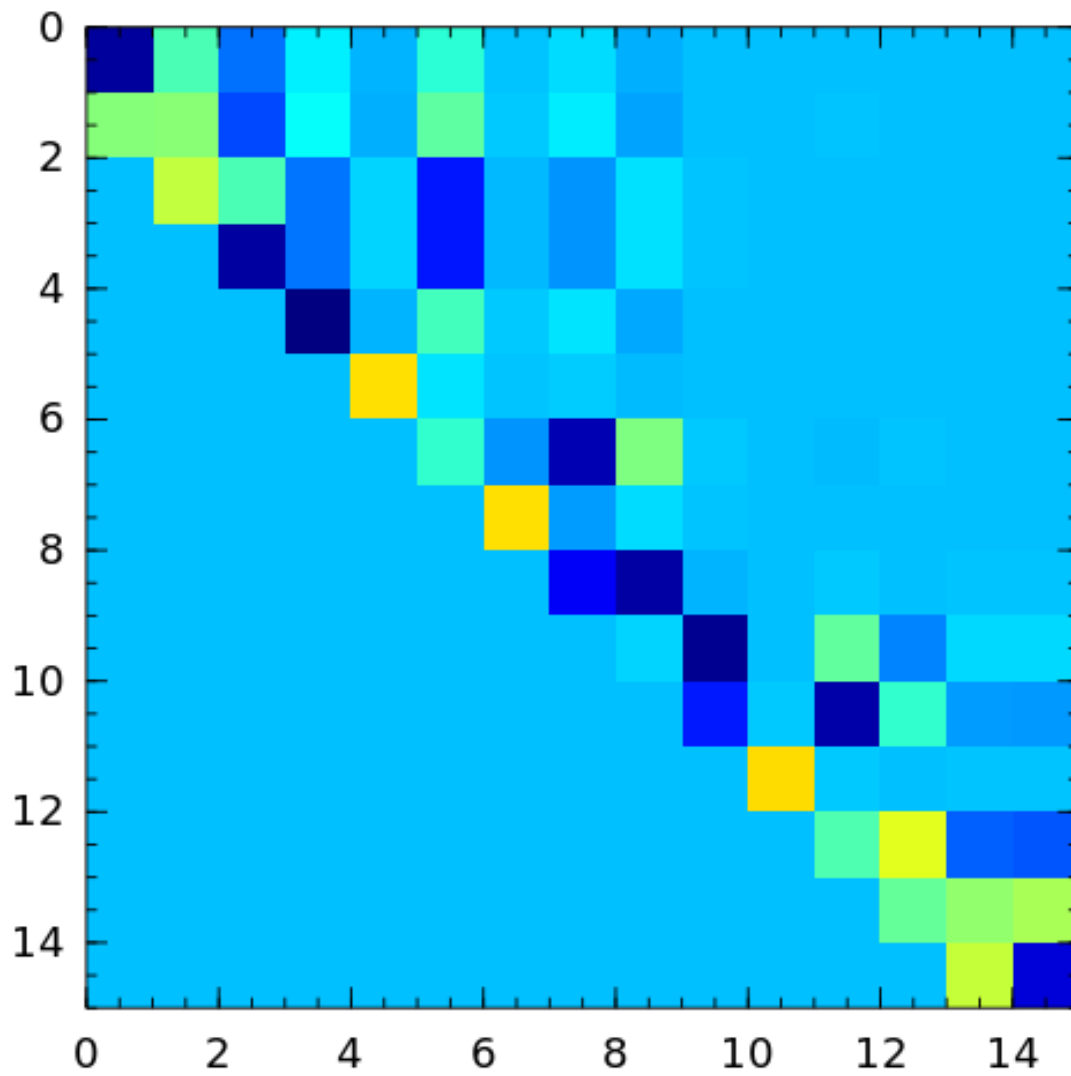
**rand0** A random matrix with entries -1, 0 or 1.



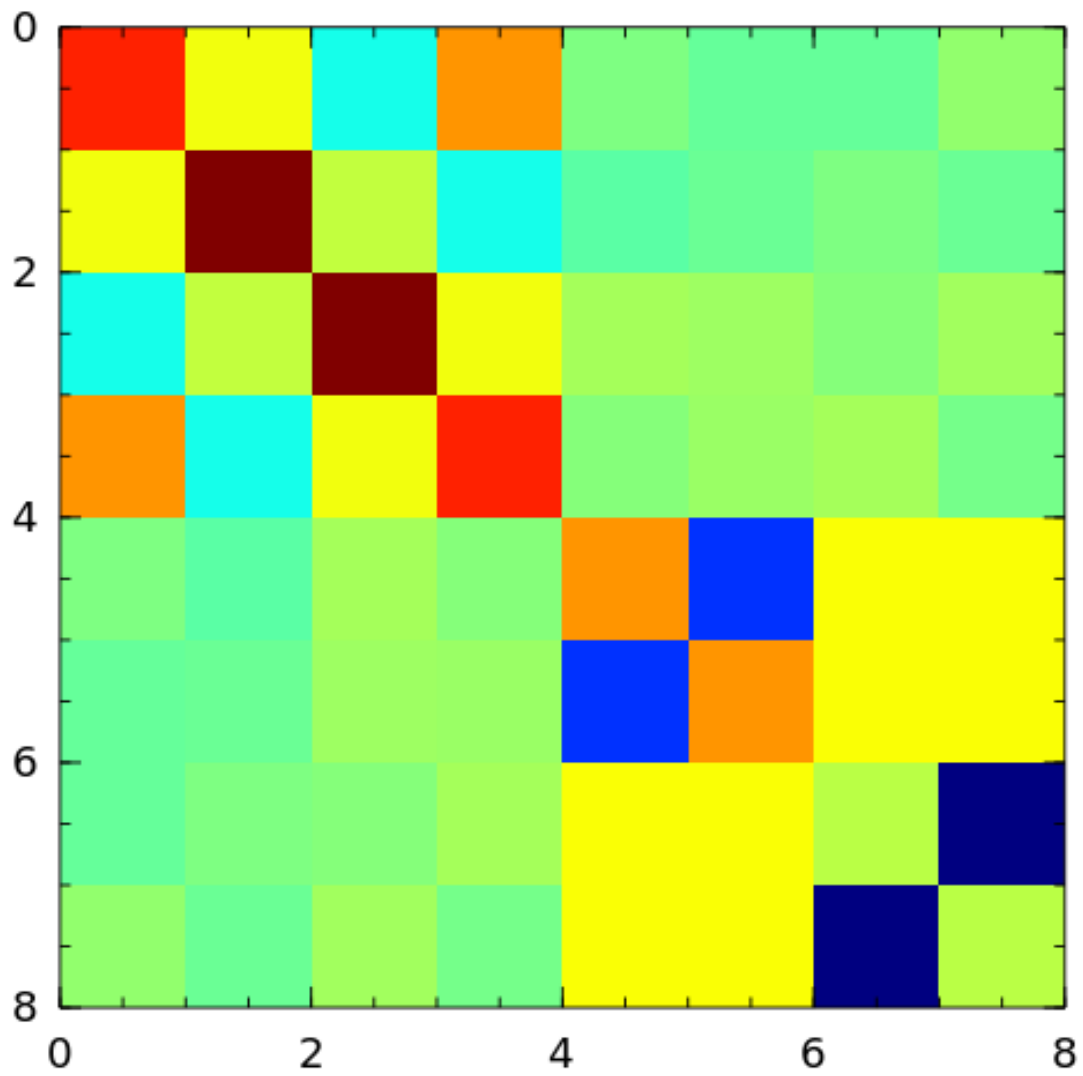
**randsvd** Random matrix with pre-assigned singular values. See [\[high02\]](#) (Sec. 28.3).



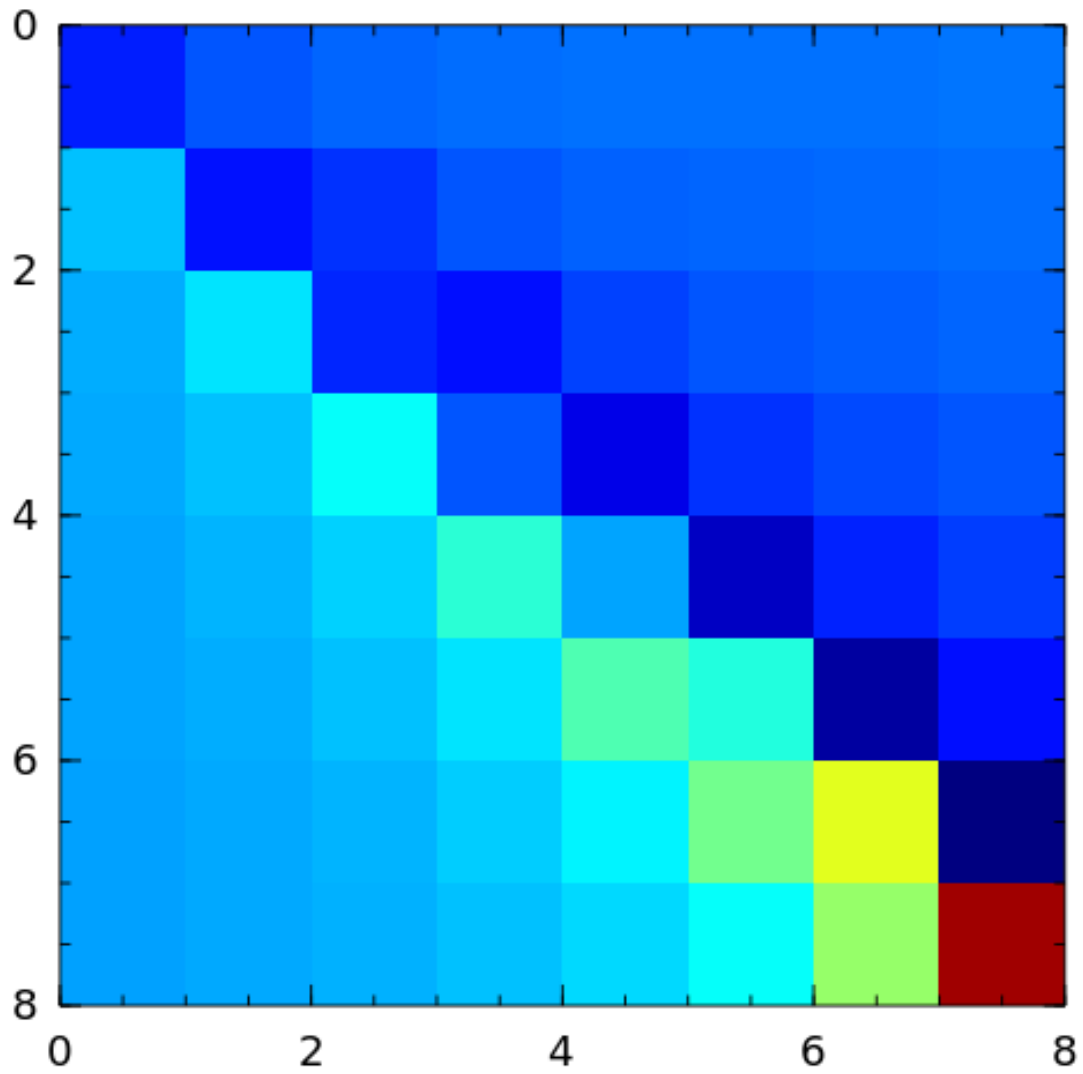
**rohess** A random orthogonal upper Hessenberg matrix. The matrix is constructed via a product of Givens rotations.



**rosser** The Rosser matrix's eigenvalues are very close together so it is a challenging matrix for many eigenvalue algorithms. `matrixdepot("rosser", 8, 2, 1)` generates the test matrix used in the paper [r1hk51]. `matrixdepot("rosser")` are more general test matrices with similar property.



**sampling** Matrices with application in sampling theory. A  $n$ -by- $n$  nonsymmetric matrix with eigenvalues  $0, 1, 2, \dots, n - 1$  [botr07].



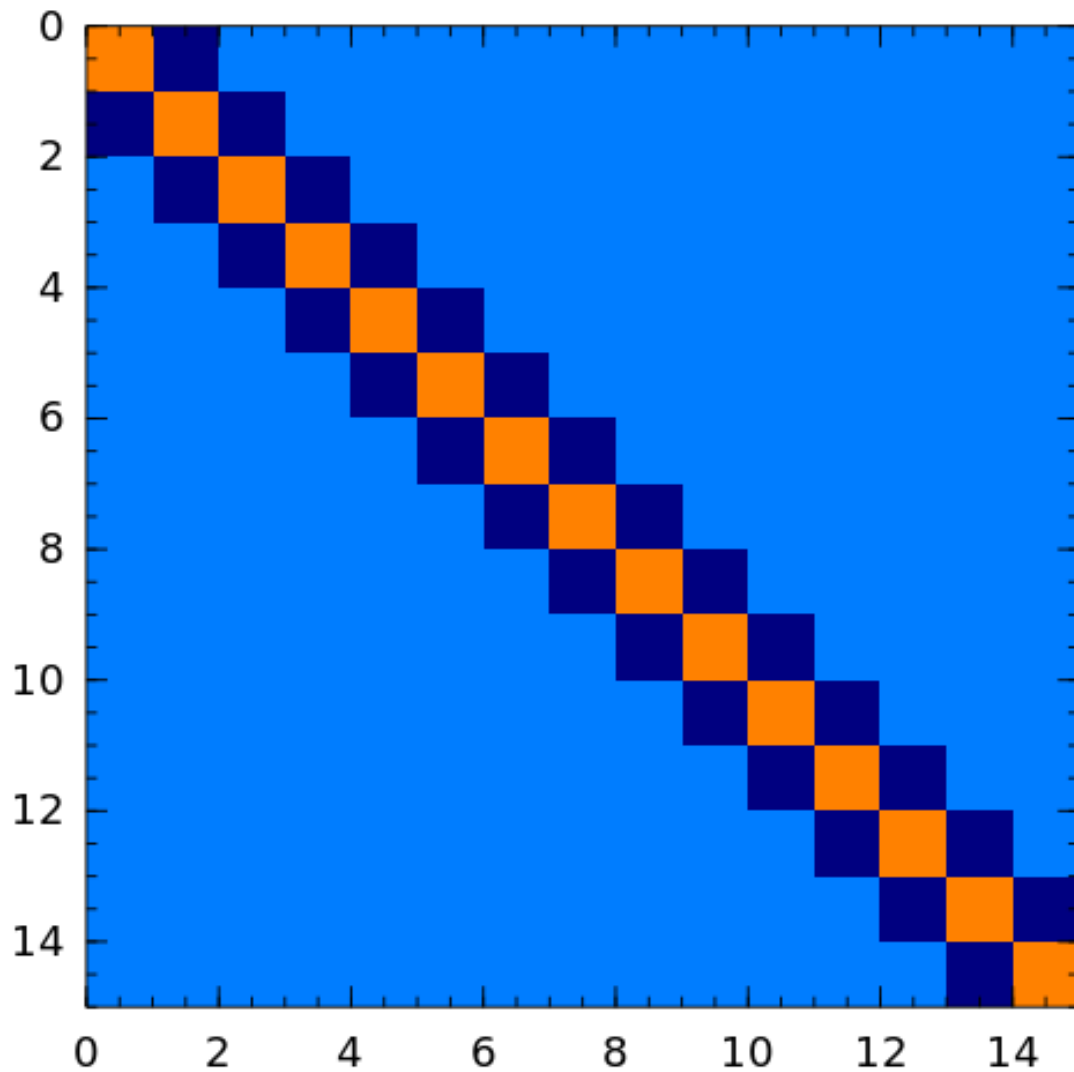
**toeplitz** Toeplitz matrix is a matrix in which each descending diagonal from left to right is constant. For example:

```
julia> matrixdepot("toeplitz", [1,2,3,4], [1,4,5,6])
4x4 Array{Int64,2}:
 1  4  5  6
 2  1  4  5
 3  2  1  4
 4  3  2  1

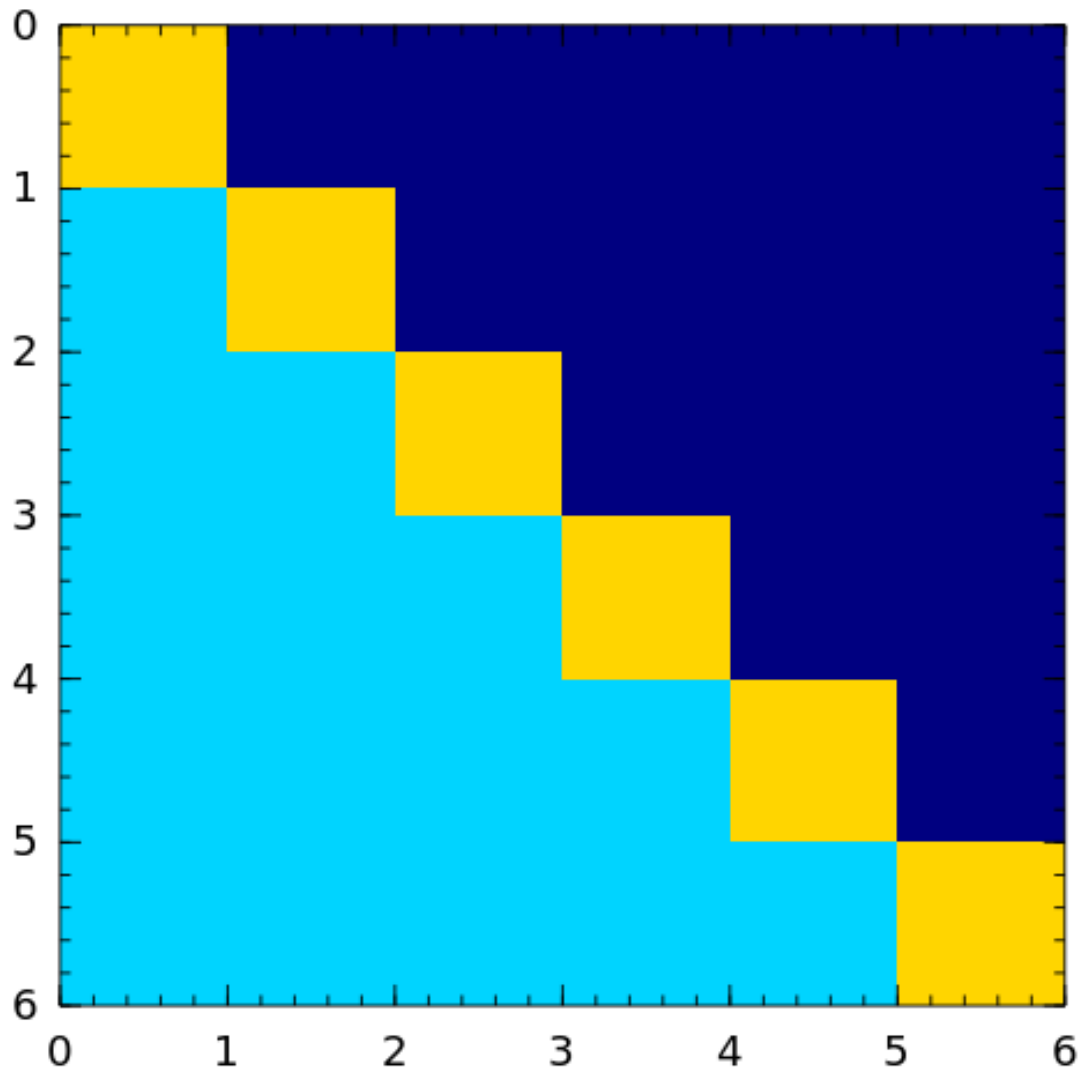
julia> matrixdepot("toeplitz", [1,2,3,4])
4x4 Array{Int64,2}:
 1  2  3  4
 2  1  2  3
 3  2  1  2
 4  3  2  1
```

**tridiag** A group of tridiagonal matrices. `matrixdepot("tridiagonal", n)` generate a tridiagonal matrix

with 1 on the diagonal and -2 on the upper- lower- diagonal, which is a symmetric positive definite M-matrix. This matrix is also known as Strang's matrix, named after Gilbert Strang.



**triw** Upper triangular matrices discussed by Wilkinson and others [gowi76].

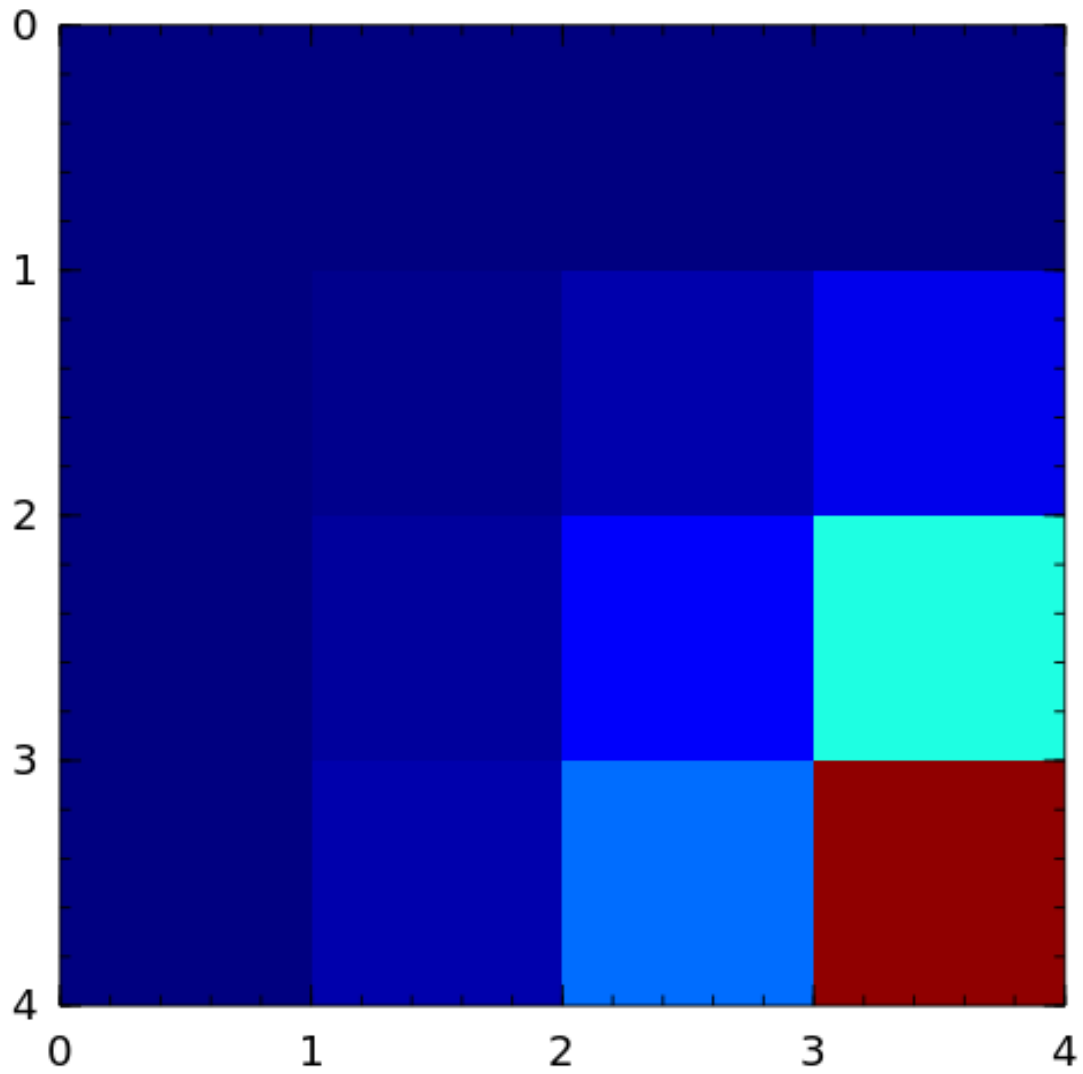


**vand** The Vandermonde matrix is defined in terms of scalars  $\alpha_0, \alpha_1, \dots, \alpha_n$  by

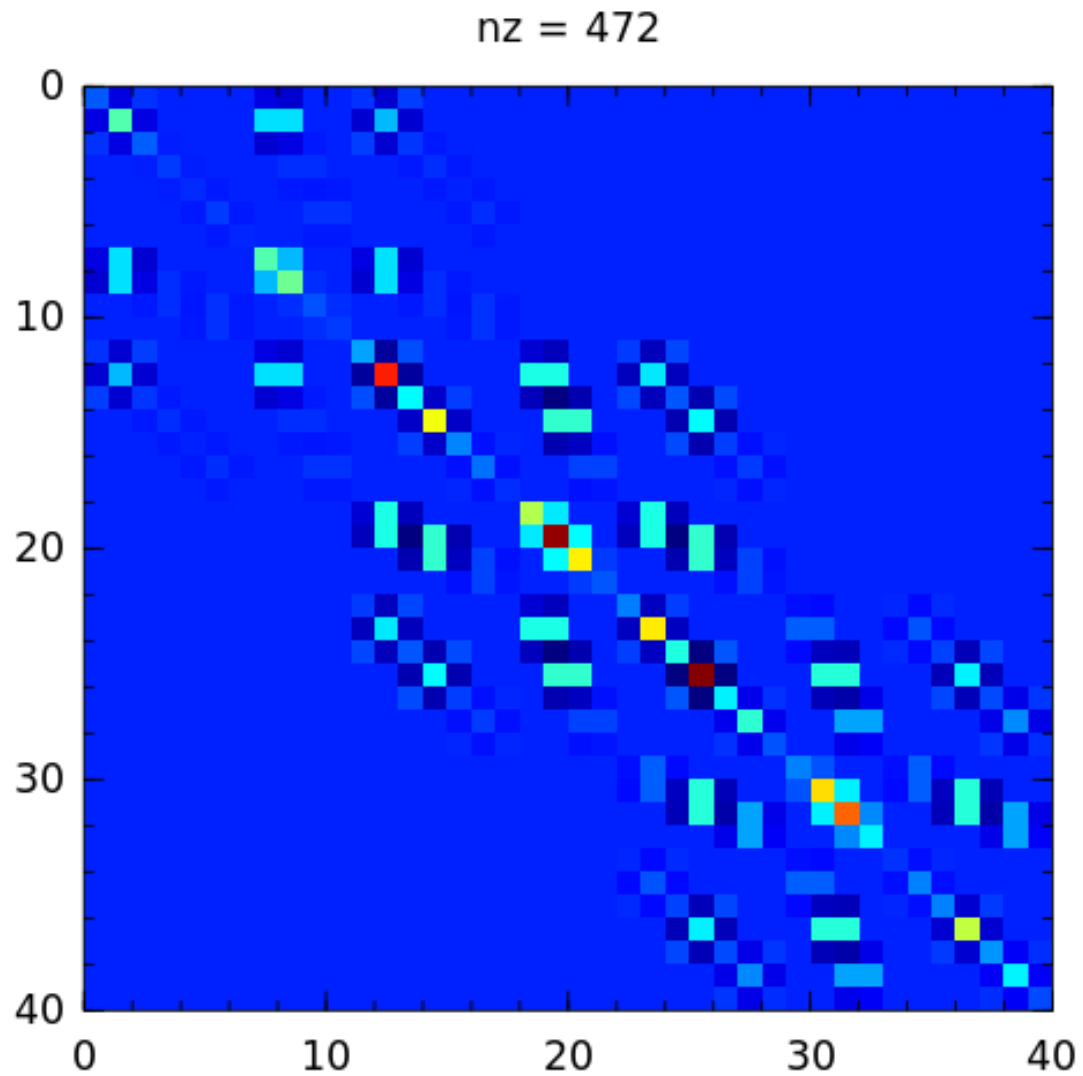
$$V(\alpha_0, \dots, \alpha_n) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \cdots & \alpha_n \\ \vdots & \vdots & & \vdots \\ \alpha_0^n & \alpha_1^n & \cdots & \alpha_n^n \end{bmatrix}.$$

The inverse and determinant are known explicitly [high02].

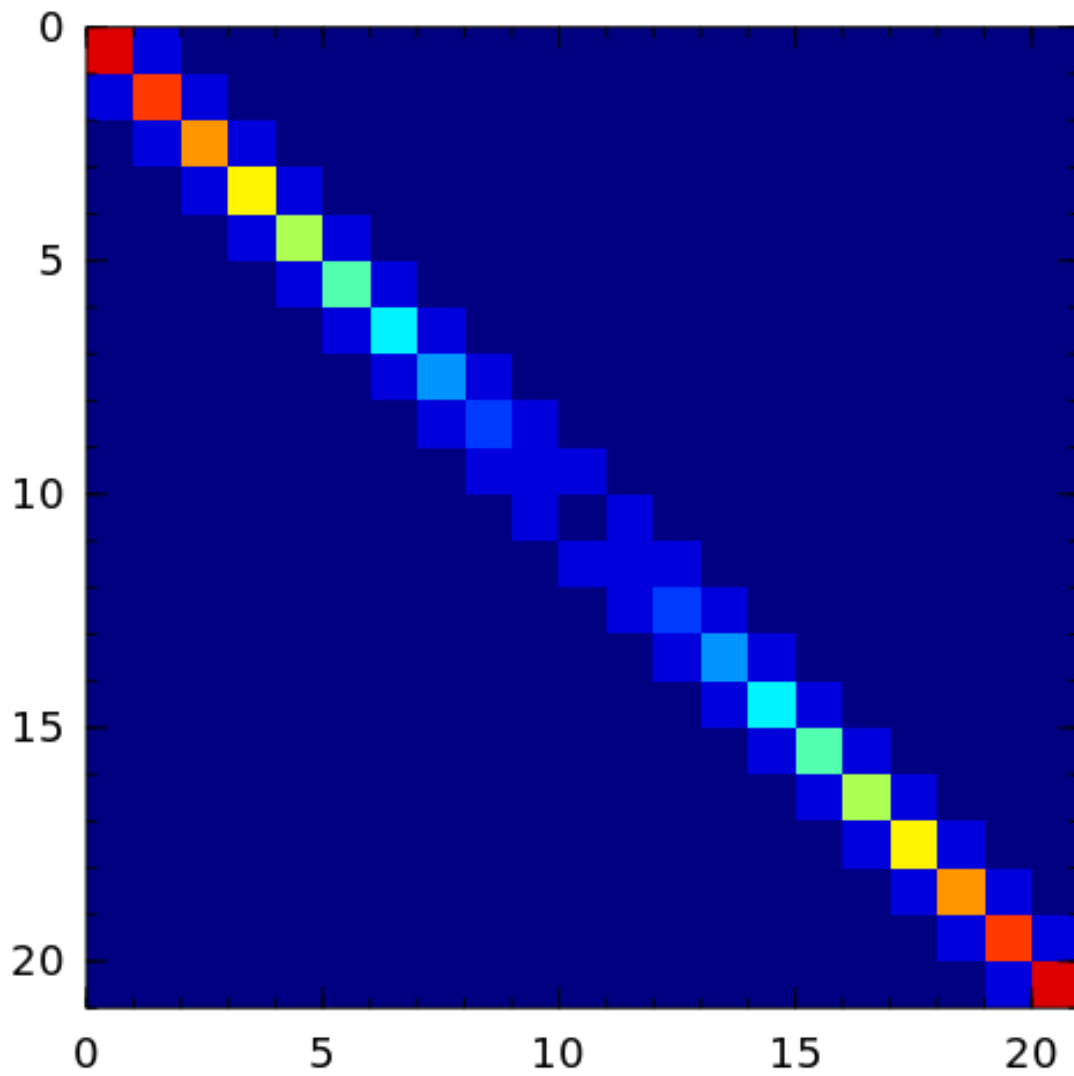




**wathen** The Wathen matrix is a sparse, symmetric positive, random matrix arising from the finite element method [wath87]. It is the consistent mass matrix for a regular  $n_x$ -by- $n_y$  grid of 8-node elements.



**wilkinson** The Wilkinson matrix is a symmetric tridiagonal matrix with pairs of nearly equal eigenvalues. The most frequently used case is `matrixdepot("wilkinson", 21)`.



---

**Note:** The images are generated using `Winston.jl`'s `imagesc` function.

---

## 1.3 Random Graphs

- *erdrey*
- *gilbert*
- *smallworld*

**erdrey** An adjacency matrix of an Erdős–Rényi random graph: an undirected graph is chosen uniformly at random from the set of all symmetric graphs with a fixed number of nodes and edges. For example:

```
julia> matrixdepot("erdrey", 5, 3) # an undirected graph with 5 nodes and 3 edges.
5x5 sparse matrix with 6 Float64 entries:
 [3, 1] = 1.0
 [3, 2] = 1.0
 [1, 3] = 1.0
 [2, 3] = 1.0
 [5, 4] = 1.0
 [4, 5] = 1.0
```

**gilbert** An adjacency matrix of a Gilbert random graph: each possible edge occurs independently with a given probability.

**smallworld** Motivated by the small world model proposed by Watts and Strogatz [wast98], we proposed a random graph model by adding shortcuts to a  $k$ th nearest neighbor ring (node  $i$  and  $j$  are connected iff  $|i - j| \leq k$  or  $|n - |i - j|| \leq k$ ).

## 1.4 Test Problems for Regularization Methods

A Fredholm integral equation of the first kind (in 1-dimensional) can be written as

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

where  $g$  and  $K$  (called kernel) are known functions and  $f$  is the unknown solution. This is a classical example of a linear ill-posed problem, i.e., an arbitrary small perturbation of the data can cause an arbitrarily large perturbation of the solution. For example, in computerized tomography,  $K$  is an X-ray source,  $f$  is the object being scanned, and  $g$  is the measured damping of the X-rays. The goal here is to reconstruct the scanned object from information about the locations of the X-ray sources and measurements of their damping.

After discretizations (by the quadrature method or the Galerkin method), we obtain a linear system of equations  $Ax = b$ . All the regularization test problems are derived from discretizations of a Fredholm integral equation of the first kind. Each generated test problem has type `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
```

Here is an example:

```
julia> matrixdepot("deriv2")
Computation of the Second Derivative:

A classical test problem for regularization algorithms.

Input options:

1. [type,] n, [matrixonly]: the dimension of the matrix is n.
   If matrixonly = false, the linear system A, b, x will be generated.
   (matrixonly = true by default.)

Reference: P.C. Hansen, Regularization tools: A MATLAB package for
analysis and solution of discrete ill-posed problems.
Numerical Algorithms, 6(1994), pp.1-35
```

```

julia> A = matrixdepot("deriv2", 4) # generate the test matrix
4x4 Array{Float64,2}:
-0.0169271  -0.0195313  -0.0117188  -0.00390625
-0.0195313  -0.0481771  -0.0351563  -0.0117188
-0.0117188  -0.0351563  -0.0481771  -0.0195313
-0.00390625 -0.0117188  -0.0195313  -0.0169271

julia> A = matrixdepot("deriv2", Float32, 4)
4x4 Array{Float32,2}:
-0.0169271  -0.0195313  -0.0117188  -0.00390625
-0.0195313  -0.0481771  -0.0351563  -0.0117188
-0.0117188  -0.0351563  -0.0481771  -0.0195313
-0.00390625 -0.0117188  -0.0195313  -0.0169271

julia> r = matrixdepot("deriv2", 3, false) # generate the test problem
Test problems for Regularization Method
A:
3x3 Array{Float64,2}:
-0.0277778  -0.0277778  -0.00925926
-0.0277778  -0.0648148  -0.0277778
-0.00925926 -0.0277778  -0.0277778
b:
[-0.01514653483985129, -0.03474793286789414, -0.022274315940957783]
x:
[0.09622504486493762, 0.28867513459481287, 0.48112522432468807]

julia> r.A # generate the matrix A
3x3 Array{Float64,2}:
-0.0277778  -0.0277778  -0.00925926
-0.0277778  -0.0648148  -0.0277778
-0.00925926 -0.0277778  -0.0277778

```

Here is a list of test problems in the collection:

- *baart*
- *blur*
- *deriv2*
- *foxgood*
- *gravity*
- *heat*
- *parallax*
- *phillips*
- *shaw*
- *spikes*
- *ursell*
- *wing*

**baart** Discretization of an artificial Fredholm integral equation of the first kind [*baart82*]. The kernel  $K$  is given by

$$K(s, t) = \exp(s \cos(t)).$$

The right-hand side  $g$  and the solution  $f$  are given by

$$g(s) = 2\frac{\sin(s)}{s}, \quad f(t) = \sin(t).$$

**blur** Image deblurring test problem. It arises in connection with the degradation of digital images by atmospheric turbulence blur, modelled by a Gaussian point-spread function

$$h(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right).$$

The matrix  $A$  is a symmetric  $n^2 \times n^2$  doubly block Toeplitz matrix, stored in sparse format.

**deriv2** Computation of the second derivative. The kernel  $K$  is Green's function for the second derivative

$$K(s, t) = \begin{cases} s(t-1), & s < t, \\ t(s-1), & s \geq t, \end{cases}$$

and both integration intervals are  $[0, 1]$ . The function  $g$  and  $f$  are given by

$$g(s) = (s^3 - s)/6, \quad f(t) = t.$$

The symmetric matrix  $A$  and vectors  $x$  and  $b$  are computed from  $K$ ,  $f$  and  $g$  using the Galerkin method.

**foxgood** A severely ill-posed problem suggested by Fox & Goodwin. This is a model problem which does not satisfy the discrete Picard condition for the small singular values [baker77].

**gravity** One-dimensional gravity surveying model problem. Discretization of a 1-D model problem in gravity surveying, in which a mass distribution  $f(t)$  is located at depth  $d$ , while the vertical component of the gravity field  $g(s)$  is measured at the surface. The resulting problem is a first-kind Fredholm integral equation with kernel

$$K(s, t) = d(d^2 + (s - t)^2)^{-3/2}.$$

**heat** Inverse heat equation [carasso82]. It is a Volterra integral equation of the first kind with integration interval  $[0, 1]$ . The kernel  $K$  is given by

$$K(s, t) = k(s - t),$$

where

$$k(t) = \frac{t^{-3/2}}{2\kappa\sqrt{\pi}} \exp\left(-\frac{1}{4\kappa^2 t}\right).$$

$\kappa$  controls the ill-conditioning of the matrix  $A$ .  $\kappa = 1$  (default) gives an ill-conditioned matrix and  $\kappa = 5$  gives a well-conditioned matrix.

**parallax** Stellar parallax problem with 26 fixed, real observations. The underlying problem is a Fredholm integral equation of the first kind with kernel

$$K(s, t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{s-t}{\sigma}\right)^2\right),$$

with  $\sigma = 0.014234$  and it is discretized by means of a Galerkin method with  $n$  orthonormal basis functions. The right-hand side  $b$  consists of a measured distribution function of stellar parallaxes, and its length is fixed at 26; i.e., the matrix  $A$  is  $26 \times n$ . The exact solution, which represents the true distribution of stellar parallaxes, is unknown.

**phillips** Phillips's "famous" problem. Discretization of the "famous" Fredholm integral equation of the first kind devised by D.L. Phillips [*phillips62*]. The kernel  $K$  and solution  $f$  are given by

$$K(s, t) = \theta(s - t), \quad f(t) = \theta(t),$$

where

$$\theta(x) = \begin{cases} 1 + \cos\left(\frac{\pi x}{3}\right), & |x| < 3, \\ 0, & |x| \geq 3. \end{cases}$$

The right-hand side  $g$  is given by

$$g(s) = (6 - |s|) \left(1 + \frac{1}{2} \cos\left(\frac{\pi s}{3}\right)\right) + \frac{9}{2\pi} \sin\left(\frac{\pi |s|}{3}\right).$$

Both integration intervals are  $[-6, 6]$ .

**shaw** One-dimensional image restoration model. This test problem uses a first-kind Fredholm integral equation to model a one-dimensional image restoration situation. The kernel  $K$  is given by

$$K(s, t) = (\cos(s) + \cos(t))^2 \left(\frac{\sin(u)}{u}\right)^2,$$

where

$$u = \pi(\sin(s) + \sin(t)).$$

Both integration intervals are  $[-\pi/2, \pi/2]$ . The solution  $f$  is given by

$$f(t) = a_1 \exp(-c_1(t - t_1)^2) + a_2 \exp(-c_2(t - t_2)^2).$$

$K$  and  $f$  are discretized by simple quadrature to produce the matrix  $A$  and the solution vector  $x$ . The right-hand  $b$  is computed by  $b = Ax$ .

**spikes** Artificially generated discrete ill-posed problem.

**ursell** Discretization of a Fredholm integral equation of the first kind with kernel  $K$  and right-hand side  $g$  given by

$$K(s, t) = \frac{1}{s + t + 1}, \quad g(s) = 1,$$

where both integration intervals are  $[0, 1]$  [*ursell*].

**wing** A problem with a discontinuous solution. The kernel  $K$  is given by

$$K(s, t) = t \exp(-st^2),$$

with both integration intervals are  $[0, 1]$ . The functions  $f$  and  $g$  are given as

$$f(t) = \begin{cases} 1, & t_1 < t < t_2, \\ 0, & \text{otherwise,} \end{cases} \quad g(s) = \frac{\exp(-st_1^2) - \exp(-st_2^2)}{2s}.$$

Here  $0 < t_1 < t_2 < 1$ . The matrix  $A$  and two vectors  $x$  and  $b$  are obtained by Galerkin discretization with orthonormal basis functions defined on a uniform mesh.

## 1.5 Groups

Groups are lists of matrix names and we use them to categorize matrices in Matrix Depot. The list below shows all the predefined groups in Matrix Depot and we can extend this list by defining new groups.

### 1.5.1 Predefined Groups

**all** All the matrices in the collection.

**data** The matrix has been downloaded from UF sparse matrix collection or the Matrix Market collection.

**eigen** Part of the eigensystem of the matrix is explicitly known.

**graph** An adjacency matrix of a graph.

**ill-cond** The matrix is ill-conditioned for some parameter values.

**inverse** The inverse of the matrix is known explicitly.

**pos-def** The matrix is positive definite for some parameter values.

**random** The matrix has random entries.

**regprob** The output is a test problem for Regularization Methods.

**sparse** The matrix is sparse.

**symmetric** The matrix is symmetric for some parameter values.

### 1.5.2 Adding New Groups

New groups can be added with the macro @addgroup:

```
@addgroup myfav = ["lehmer", "cauchy", "hilb"]
87

@addgroup test_for_paper2 = ["tridiag", "sampling", "wing"]
138

workspace()

using MatrixDepot # reload the package
matrixdepot()

Matrices:
 1) baart          2) binomial       3) cauchy         4) chebspec
 5) chow           6) circul        7) clement      8) deriv2
 9) dingdong      10) fiedler      11) forsythe     12) foxgood
13) frank         14) grcar        15) hadamard     16) heat
17) hilb          18) invhilb     19) invol        20) kahan
21) kms           22) lehmer       23) lotkin       24) magic
25) minij         26) moler        27) neumann      28) oscillate
29) parter        30) pascal       31) pei          32) phillips
33) poisson      34) prolate     35) randcorr     36) rando
37) randsvd      38) rohess      39) rosser       40) sampling
41) shaw         42) toeplitz    43) tridiag     44) triw
45) vand         46) wathen      47) wilkinson   48) wing

Groups:
data            eigen            ill-cond        inverse
pos-def       random          regprob         sparse
symmetric       myfav           test_for_paper2

matrixdepot("myfav")
3-element Array{ASCIIString,1}:
```



```
"lehmer"
"cauchy"
"hilb"
```

## 1.6 Interface to Test Collections

Before downloading test matrices, it is recommended to first update the database:

```
julia> MatrixDepot.update()
```

### 1.6.1 Interface to the UF Sparse Matrix Collection

Use `matrixdepot(NAME, :get)`, where `NAME` is `collection_name + '/' + matrix_name`, to download a test matrix from the [UF Sparse Matrix Collection](#). For example:

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

**Note:** `matrixdepot()` displays all the matrices in the collection, including the newly downloaded matrices. All the matrix data can be found by `matrixdepot("data")`.

If the matrix name is unique in the collections, we could use `matrixdepot(matrix_name, :get)` to download the data. If more than one matrix has the same name, a list of options will be returned. For example:

```
julia> matrixdepot("epb0", :get)
% Total      % Received % Xferd  Average Speed   Time    Time       Time  Current
             Dload  Upload   Total     Spent    Left  Speed
100 83244  100 83244    0     0   109k      0  --:--:--  --:--:--  --:--:--  133k
download:/home/weijian/.julia/v0.4/MatrixDepot/data/uf/Averous/epb0.tar.gz
epb0/epb0.mtx

julia> matrixdepot("1138_bus", :get)
Try MatrixDepot.get(`name`), where `name` is one of the elements in the following_
↪Array:
2-element Array{AbstractString,1}:
"HB/1138_bus"
"Harwell-Boeing/psadmit/1138_bus"
```

When download is complete, we can check matrix information using:

```
julia> matrixdepot("SNAP/web-Google")

%%MatrixMarket matrix coordinate pattern general
%-----
% UF Sparse Matrix Collection, Tim Davis
% http://www.cise.ufl.edu/research/sparse/matrices/SNAP/web-Google
% name: SNAP/web-Google
% [Web graph from Google]
% id: 2301
% date: 2002
% author: Google
% ed: J. Leskovec
```

```
% fields: name title A id date author ed kind notes
% kind: directed graph
%-----
...
```

and generate it with the Symbol `:r` or `:read`

```
julia> 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
 [43930 ,      1] = 1.0
 [46275 ,      1] = 1.0
 [48193 ,      1] = 1.0
 [50823 ,      1] = 1.0
 [56911 ,      1] = 1.0
 [62930 ,      1] = 1.0
 [68315 ,      1] = 1.0
 [71879 ,      1] = 1.0
 [72433 ,      1] = 1.0
 [73632 ,      1] = 1.0

 [532967, 916427] = 1.0
 [547586, 916427] = 1.0
 [557890, 916427] = 1.0
 [571471, 916427] = 1.0
 [580544, 916427] = 1.0
 [608625, 916427] = 1.0
 [618730, 916427] = 1.0
 [622998, 916427] = 1.0
 [673046, 916427] = 1.0
 [716616, 916427] = 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
 [843845, 916428] = 1.0
```

The metadata of a given matrix can be obtained by `matrixdepot(collection_name/matrix_name, :read, meta = true)`. For example:

```
julia> matrixdepot("TKK/t520", :get)
julia> matrixdepot("TKK/t520", :read, meta = true)
Dict{AbstractString,Any} with 3 entries:
 "t520"      => 5563x5563 Symmetric{Float64,SparseMatrixCSC{Float64,Int64}}:...
 "t520_b"    => "%MatrixMarket matrix array real general\n%-----...
 "t520_coord" => "%MatrixMarket matrix array real general\n%-----...
```

We can use `matrixdepot(collection_name/*)` to download all the matrices in a given `collection_name`. For example, we can get all the matrices contributed by The Mathworks, Inc. by

```
matrixdepot("MathWorks/*", :get):
```

```
julia> 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) prolate      40) randcorr
41) rando        42) randsvd      43) rohess       44) rosser
45) sampling     46) shaw         47) spikes       48) toeplitz
49) tridiag      50) triw         51) ursell       52) vand
53) wathen       54) wilkinson    55) wing         56) MathWorks/Harvard500
57) MathWorks/Kaufhold
58) MathWorks/Kuu  59) MathWorks/Muu  60) MathWorks/Pd  61) MathWorks/Pd_rhs
62) MathWorks/pivtol
63) MathWorks/QRpivot
64) MathWorks/Sieber
65) MathWorks/tomography
66) MathWorks/TS

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

```
julia> matrixdepot("data")
11-element Array{AbstractString,1}:
"MathWorks/Harvard500"
"MathWorks/Kaufhold"
"MathWorks/Kuu"
"MathWorks/Muu"
"MathWorks/Pd"
"MathWorks/Pd_rhs"
"MathWorks/pivtol"
"MathWorks/QRpivot"
"MathWorks/Sieber"
"MathWorks/tomography"
"MathWorks/TS"
```

## 1.6.2 Interface to NIST Matrix Market

Use `matrixdepot(NAME, :get)`, where `NAME` is collection name + `'/'` + set name + `'/'` + matrix name to download a test matrix from NIST Matrix Market: <http://math.nist.gov/MatrixMarket/>. For example:

```
julia> matrixdepot("Harwell-Boeing/lanpro/nos5", :get)
```

If the matrix name is unique, we could also use `matrixdepot(matrix name, :get)`:

```
julia> matrixdepot("bp__1400", :get)
% Total      % Received % Xferd  Average Speed   Time    Time       Time   Current
```

```

          Dload Upload Total Spent Left Speed
100 28192 100 28192 0 0 4665 0 0:00:06 0:00:06 --:--:-- 10004
download:/home/weijian/.julia/v0.4/MatrixDepot/data/mm/Harwell-Boeing/smtape/bp__1400.
↪mtx.gz

```

Checking matrix information and generating matrix data are similar to the above case:

```

julia> matrixdepot("Harwell-Boeing/smtape/bp__1400")

%%MatrixMarket matrix coordinate real general

use matrixdepot("Harwell-Boeing/smtape/bp__1400", :read) to read the data

julia> matrixdepot("Harwell-Boeing/smtape/bp__1400", :read)
822x822 sparse matrix with 4790 Float64 entries:
 [1 , 1] = 1.0
 [1 , 2] = 0.001
 [26 , 2] = -1.0
 [1 , 3] = 0.6885
 [25 , 3] = 0.9542
 [692, 3] = 1.0
 [718, 3] = 5.58

 [202, 820] = -1.0
 [776, 820] = 1.0
 [1 , 821] = 0.4622
 [25 , 821] = 0.725
 [28 , 821] = 1.0
 [202, 821] = -1.0
 [796, 821] = 1.0
 [2 , 822] = 1.0

```

## 1.7 Adding New Matrix Generators

Matrix Depot provides a diverse collection of test matrices, including parametrized matrices and real-life matrices. But occasionally, you may want to define your own matrix generators and be able to use them from Matrix Depot.

### 1.7.1 Declaring Generators

When Matrix Depot is first loaded, a new directory `myMatrixDepot` will be created. Matrix Depot automatically includes all Julia files in this directory. Hence, all we need to do is to copy the generator files to `path/to/MatrixDepot/myMatrixDepot` and use the function `include_generator` to declare them.

**include\_generator** (*Stuff\_To\_Be\_Included*, *Stuff*, *f*)

Includes a piece of information of the function `f` to Matrix Depot, where `Stuff_To_Be_Included` is one of the following:

- **FunctionName**: the function name of `f`. In this case, `Stuff` is a string representing `f`.
- **Group**: the group where `f` belongs. In this case, `Stuff` is the group name.

## 1.7.2 Examples

To get a feel of how it works, let's see an example. Suppose we have a file `myrand.jl` which contains two matrix generator `randsym` and `randorth`:

```

"""
random symmetric matrix
=====

*Input options:*

+ n: the dimension of the matrix
"""
function randsym(n)
    A = zeros(n, n)
    for j = 1:n
        for i = j:n
            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 first need to find out where Matrix Depot is installed. This can be done by:

```

julia> Pkg.dir("MatrixDepot")
"/home/weijian/.julia/v0.4/MatrixDepot"

```

For me, the package is installed at `/home/weijian/.julia/v0.4/MatrixDepot`. We can copy `myrand.jl` to `/home/weijian/.julia/v0.4/MatrixDepot/myMatrixDepot`. Now we open the file `myMatrixDepot/generator.jl` and write:

```

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

```

This is it. We can now use them from Matrix Depot:

```

julia> 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) prolate       40) randcorr
41) rando        42) randorth     43) randsvd      44) randsym
45) rohess       46) rosser       47) sampling     48) shaw
49) spikes       50) toeplitz    51) tridiag      52) triw
53) ursell       54) vand         55) wathen       56) wilkinson
57) wing
Groups:
all              data              eigen             ill-cond
inverse         pos-def          random            regprob
sparse          symmetric

julia> matrixdepot("randsym")
random symmetric matrix

Input options:

• n: the dimension of the matrix

julia> matrixdepot("randsym", 5)
5x5 Array{Float64,2}:
 1.57579   0.474591  0.0261732 -0.536217 -0.0900839
 0.474591  0.388406  0.77178   0.239696  0.302637
 0.0261732 0.77178  1.7336    1.72549   0.127008
-0.536217  0.239696 1.72549   0.304016  1.5854
-0.0900839 0.302637 0.127008  1.5854   -0.656608

julia> A = matrixdepot("randorth", 5)
5x5 Array{Float64,2}:
-0.359134  0.401435  0.491005 -0.310518  0.610218
-0.524132 -0.474053 -0.53949 -0.390514  0.238764
 0.627656  0.223519 -0.483424 -0.104706  0.558054
-0.171077  0.686038 -0.356957 -0.394757 -0.465654
 0.416039 -0.305802  0.326723 -0.764383 -0.205834

julia> A'*A
5x5 Array{Float64,2}:
 1.0          8.32667e-17  1.11022e-16  5.55112e-17 -6.93889e-17
 8.32667e-17  1.0          -1.80411e-16 -2.77556e-17 -5.55112e-17
 1.11022e-16 -1.80411e-16  1.0          1.94289e-16 -1.66533e-16
 5.55112e-17 -2.77556e-17  1.94289e-16  1.0          1.38778e-16
-6.93889e-17 -5.55112e-17 -1.66533e-16  1.38778e-16  1.0

```

We can also add group information:

```

include_generator(Group, "random", randsym)
include_generator(Group, "symmetric", randsym)

```

Now if we type:

```

julia> matrixdepot("random")
9-element Array{ASCIIString,1}:
"golub"
"oscillate"
"randcorr"
"rando"

```

```

"randsvd"
"randsym"
"rohess"
"rosser"
"wathen"

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

"pascal"
"pei"
"poisson"
"prolate"
"randcorr"
"randsym"
"tridiag"
"wathen"
"wilkinson"

```

the function `randsym` will be part of the group `symmetric` and `random`.

It is a good idea to back up your changes. For example, we could save it on GitHub by creating a new repository named `myMatrixDepot`. (See <https://help.github.com/articles/create-a-repo/> for details of creating a new repository on GitHub.) Then we go to the directory `path/to/MatrixDepot/myMatrixDepot` and type:

```

git init
git add *.jl
git commit -m "first commit"
git remote add origin https://github.com/your-user-name/myMatrixDepot.git
git push -u origin master

```

## 1.8 Examples

### 1.8.1 Demo

IJulia Notebook

### 1.8.2 Getting Started

To see all the matrices in the collection, type

```

julia> matrixdepot()

Matrices:

```

```
1) baart      2) binomial    3) cauchy      4) chebspec
5) chow       6) circul      7) clement    8) deriv2
9) dingdong  10) fiedler    11) forsythe   12) foxgood
13) frank     14) grcar      15) hadamard   16) heat
17) hilb      18) invhilb    19) invol      20) kahan
21) kms       22) lehmer     23) lotkin     24) magic
25) minij     26) moler      27) neumann    28) oscillate
29) parter    30) pascal     31) pei        32) phillips
33) poisson   34) prolate    35) randcorr   36) rando
37) randsvd   38) rohess     39) rosser     40) sampling
41) shaw      42) toeplitz   43) tridiag    44) triw
45) vand      46) wathen     47) wilkinson  48) wing
```

Groups:

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

We can generate a Hilbert matrix of size 4 by typing

```
matrixdepot("hilb", 4)

4x4 Array{Float64,2}:
 1.0      0.5      0.333333  0.25
 0.5      0.333333  0.25      0.2
 0.333333 0.25      0.2       0.166667
 0.25     0.2       0.166667  0.142857
```

and generate a circul matrix of size 5 by

```
matrixdepot("circul", 5)

5x5 Array{Float64,2}:
 1.0  2.0  3.0  4.0  5.0
 5.0  1.0  2.0  3.0  4.0
 4.0  5.0  1.0  2.0  3.0
 3.0  4.0  5.0  1.0  2.0
 2.0  3.0  4.0  5.0  1.0
```

We can type the matrix name to get help.

```
matrixdepot("hilb")
Hilbert matrix

The Hilbert matrix is a very ill conditioned matrix. 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"]

Reference: M. D. Choi, Tricks or treats with the Hilbert matrix, Amer. Math.
Monthly, 90 (1983), pp. 301-312.
```



N. J. Higham, Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002; sec. 28.1

```
matrixdepot("hadamard")
Hadamard matrix
```

The Hadamard matrix is a square matrix whose entries are 1 or -1. It was named after Jacques Hadamard. The rows of a Hadamard matrix are orthogonal.

Input options:

- [type,] n: the dimension of the matrix, n is a power of 2.

Groups: ["inverse", "orthogonal", "eigen"]

Reference: S. W. Golomb and L. D. Baumert, The search for Hadamard matrices, Amer. Math. Monthly, 70 (1963) pp. 12-17

From the information given, we can create a 4-by-6 rectangular Hilbert matrix by

```
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
```

We can also specify the data type

```
matrixdepot("hilb", Float16, 5, 3)

5x3 Array{Float16,2}:
 1.0      0.5      0.33325
 0.5      0.33325  0.25
 0.33325  0.25     0.19995
 0.25     0.19995  0.16663
 0.19995  0.16663  0.14282
```

Matrices can be accessed by groups.

```
matrixdepot("symmetric")

19-element Array{ASCIIString,1}:
 "hilb"
 "cauchy"
 "circul"
 "dingdong"
 "invhilb"
 "moler"
 "pascal"
 "pei"
 "clement"
 "fiedler"
 "minij"
 "tridiag"
```

```
"lehmer"  
"randcorr"  
"poisson"  
"wilkinson"  
"randsvd"  
"kms"  
"wathen"
```

```
matrixdepot("symmetric", "ill-cond")
```

```
7-element Array{ASCIIString,1}:  
"hilb"  
"cauchy"  
"invhilb"  
"moler"  
"pascal"  
"pei"  
"tridiag"
```

```
matrixdepot("inverse", "ill-cond", "symmetric")
```

```
7-element Array{ASCIIString,1}:  
"hilb"  
"cauchy"  
"invhilb"  
"moler"  
"pascal"  
"pei"  
"tridiag"
```

### 1.8.3 User Defined Groups

We can add new groups to MatrixDepot. Since each group in Matrix Depot is a list of strings, you can simply do, for example,

```
spd = matrixdepot("symmetric", "pos-def")
```

```
10-element Array{ASCIIString,1}:  
"hilb"  
"cauchy"  
"circul"  
"invhilb"  
"moler"  
"pascal"  
"pei"  
"minij"  
"tridiag"  
"lehmer"
```

```
myprop = ["lehmer", "cauchy", "hilb"]
```

```
3-element Array{ASCIIString,1}:  
"lehmer"
```

```
"cauchy"
"hilb"
```

Then use it in your tests like

```
for matrix in myprop
    A = matrixdepot(matrix, 6)
    L, U, p = lu(A) #LU factorization
    err = norm(A[p,:] - L*U, 1) # 1-norm error
    println("1-norm error for $matrix matrix is ", err)
end

1-norm error for lehmer matrix is 1.1102230246251565e-16
1-norm error for cauchy matrix is 5.551115123125783e-17
1-norm error for hilb matrix is 2.7755575615628914e-17
```

To add a group of matrices permanently for future use, we put the macro `@addgroup` at the beginning.

```
@addgroup myfav = ["lehmer", "cauchy", "hilb"]
87

@addgroup test_for_paper2 = ["tridiag", "sampling", "wing"]
138
```

We need to reload Julia to see the changes. Type

```
workspace()

using MatrixDepot
matrixdepot()

Matrices:
 1) baart          2) binomial      3) cauchy        4) chebspec
 5) chow          6) circul       7) clement      8) deriv2
 9) dingdong     10) fiedler     11) forsythe    12) foxgood
13) frank        14) grcar       15) hadamard    16) heat
17) hilb         18) invhilb    19) invol        20) kahan
21) kms          22) lehmer      23) lotkin       24) magic
25) minij        26) moler       27) neumann     28) oscillate
29) parter       30) pascal      31) pei          32) phillips
33) poisson     34) prolate     35) randcorr    36) rando
37) randsvd     38) rohess      39) rosser      40) sampling
41) shaw         42) toeplitz   43) tridiag     44) triw
45) vand         46) wathen      47) wilkinson   48) wing

Groups:
data          eigen          ill-cond       inverse
pos-def       random         regprob        sparse
symmetric     myfav          test_for_paper2
```

Notice new defined groups have been included. We can use them as

```
matrixdepot("myfav")
3-element Array{ASCIIString,1}:
 "lehmer"
 "cauchy"
 "hilb"
```

We can remove a group using the macro `@rmgroup`. As before, we need to reload Julia to see the changes.

```
@rmproperty myfav
```

```
153
```

```
> matrixdepot()
```

Matrices:

|             |              |               |               |
|-------------|--------------|---------------|---------------|
| 1) baart    | 2) binomial  | 3) cauchy     | 4) chebspec   |
| 5) chow     | 6) circul    | 7) clement    | 8) deriv2     |
| 9) dingdong | 10) fiedler  | 11) forsythe  | 12) foxgood   |
| 13) frank   | 14) grcar    | 15) hadamard  | 16) heat      |
| 17) hilb    | 18) invhilb  | 19) invol     | 20) kahan     |
| 21) kms     | 22) lehmer   | 23) lotkin    | 24) magic     |
| 25) minij   | 26) moler    | 27) neumann   | 28) oscillate |
| 29) parter  | 30) pascal   | 31) pei       | 32) phillips  |
| 33) poisson | 34) prolate  | 35) randcorr  | 36) rando     |
| 37) randsvd | 38) rohess   | 39) rosser    | 40) sampling  |
| 41) shaw    | 42) toeplitz | 43) tridiag   | 44) triw      |
| 45) vand    | 46) wathen   | 47) wilkinson | 48) wing      |

Groups:

|                 |                 |          |         |
|-----------------|-----------------|----------|---------|
| data            | eigen           | ill-cond | inverse |
| pos- <b>def</b> | random          | regprob  | sparse  |
| symmetric       | test_for_paper2 |          |         |

### 1.8.4 More Examples

An interesting test matrix is magic square. It can be generated as

```
M = matrixdepot("magic", 5)
```

```
5x5 Array{Int64,2}:
 17  24   1   8  15
 23   5   7  14  16
  4   6  13  20  22
 10  12  19  21   3
 11  18  25   2   9
```

```
sum(M, 1)
```

```
1x5 Array{Int64,2}:
 65  65  65  65  65
```

```
sum(M, 2)
```

```
5x1 Array{Int64,2}:
 65
 65
 65
 65
 65
```

```
sum(diag(M))
```

```
65
```

```
p = [5:-1:1]
sum(diag(M[:,p]))
```

```
65
```

Pascal Matrix can be generated as

```
P = matrixdepot("pascal", 6)
```

```
6x6 Array{Int64,2}:
```

```
1  1  1  1  1  1
1  2  3  4  5  6
1  3  6 10 15 21
1  4 10 20 35 56
1  5 15 35 70 126
1  6 21 56 126 252
```

Notice the Cholesky factor of the Pascal matrix has Pascal's triangle rows.

```
chol(P)
```

```
6x6 Array{Float64,2}:
```

```
1.0  1.0  1.0  1.0  1.0  1.0
0.0  1.0  2.0  3.0  4.0  5.0
0.0  0.0  1.0  3.0  6.0 10.0
0.0  0.0  0.0  1.0  4.0 10.0
0.0  0.0  0.0  0.0  1.0  5.0
0.0  0.0  0.0  0.0  0.0  1.0
```



---

## Bibliography

---

- [bmsz01] G. Boyd, C.A. Micchelli, G. Strang and D.X. Zhou, Binomial matrices, *Adv. in Comput. Math.*, 14 (2001), pp 379-391.
- [chow69] T.S. Chow, A class of Hessenberg matrices with known eigenvalues and inverses, *SIAM Review*, 11 (1969), pp. 391-395.
- [clem59] P.A. Clement, A class of triple-diagonal matrices for test purposes, *SIAM Review*, 1 (1959), pp. 50-52.
- [nash90] J.C. Nash, *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*, second edition, Adam Hilger, Bristol, 1990 (Appendix 1).
- [todd77] J. Todd, *Basic Numerical Mathematics, Vol. 2: Numerical Algebra*, Birkhauser, Basel, and Academic Press, New York, 1977, pp. 159.
- [vara86] J.M. Varah, A generalization of the Frank matrix, *SIAM J. Sci. Stat. Comput.*, 7 (1986), pp. 835-839.
- [vistre98] D. Viswanath and N. Trefethen. Condition Numbers of Random Triangular Matrices, *SIAM J. Matrix Anal. Appl.* 19, 564-581, 1998.
- [nrt92] N.M. Nachtigal, L. Reichel and L.N. Trefethen, A hybrid GMRES algorithm for nonsymmetric linear system, *SIAM J. Matrix Anal. Appl.*, 13 (1992), pp. 796-825.
- [high02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*, SIAM, PA, USA. 2002.
- [hoca63] A.S. Householder and J.A. Carpenter, The singular values of involutory and idempotent matrices, *Numer. Math.* 5 (1963), pp. 234-237.
- [tren89] W.F. Trench, Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices, *SIAM J. Matrix Analysis and Appl.*, 10 (1989), pp. 135-146
- [neto58] M. Newman and J. Todd, The evaluation of matrix inversion programs, *J. Soc. Indust. Appl. Math*, 6 (1958), pp. 466-476.
- [lotk55] 13. Lotkin, A set of test matrices, *MTAC*, 9, (1955), pp. 153-161.
- [plem76] R.J. Plemmons, Regular splittings and the discrete Neumann problem, *Numer. Math.*, 25 (1976), pp. 153-161.
- [hansen95] Per Christian Hansen, Test matrices for regularization methods. *SIAM J. SCI. COMPUT* Vol 16, No2, pp 506-512 (1995)
- [part86] S. V. Parter, On the distribution of the singular values of Toeplitz matrices, *Linear Algebra and Appl.*, 80 (1986), pp. 115-130.

- [pei62] M.L. Pei, A test matrix for inversion procedures, *Comm. ACM*, 5 (1962), pp. 508.
- [varah93] J.M. Varah. The Prolate Matrix. *Linear Algebra and Appl.* 187:267–278, 1993.
- [rlhk51] Rosser, Lanczos, Hestenes and Karush, *J. Res. Natl. Bur. Stand. Vol. 47* (1951), pp. 291-297. [Archive](#)
- [botr07] L. Bondesson and I. Traat, A Nonsymmetric Matrix with Integer Eigenvalues, *Linear and Multilinear Algebra*, 55(3)(2007), pp. 239-247.
- [gowi76] G.H. Golub and J.H. Wilkinson, Ill-conditioned eigensystems and the computation of the Jordan canonical form, *SIAM Review*, 18(4), (1976), pp. 578-619.
- [wath87] A.J. Wathen, Realistic eigenvalue bounds for the Galerkin mass matrix, *IMA J. Numer. Anal.*, 7 (1987), pp. 449-457.
- [wast98] D.J. Watts and S. H. Strogatz. Collective Dynamics of Small World Networks, *Nature* 393 (1998), pp. 440-442.
- [baart82] M.L. Baart, The use of auto-correlation for pseudo-rank determination in noisy ill-conditioned linear least-squares problems, *IMA, J. Numer. Anal.* 2 (1982), 241-247.
- [baker77] C.T.H Baker, *The Numerical Treatment of Integral Equations*, Clarendon Press, Oxford, 1977, p. 665.
- [carasso82] A.S. Carasso, Determining surface temperatures from interior observations, *SIAM J. Appl. Math.* 42 (1982), 558-574.
- [phillips62] D.L. Phillips, A technique for the numerical solution of certain integral equations of the first kind, *J. ACM* 9 (1962), 84-97.
- [ursell] F. Ursell, Introduction to the theory of linear integral equations, Chapter 1 in L. M. Delves and J. Walsh, *Numerical Solution of Integral Equations*, Clarendon Press, Oxford, 1974.



## A

all, [52](#)

## B

baart, [49](#)

binomial, [7](#)

blur, [50](#)

## C

cauchy, [8](#)

chebspec, [9](#)

chow, [10](#)

circul, [11](#)

clement, [12](#)

companion, [13](#)

## D

data, [52](#)

deriv2, [50](#)

dingdong, [14](#)

## E

eigen, [52](#)

erdrey, [47](#)

## F

fiedler, [15](#)

forsythe, [16](#)

foxgood, [50](#)

frank, [17](#)

## G

gilbert, [48](#)

golub, [18](#)

graph, [52](#)

gravity, [50](#)

grcar, [19](#)

## H

hadamard, [20](#)

hankel, [21](#)

heat, [50](#)

hilb, [21](#)

## I

ill-cond, [52](#)

include\_generator() (built-in function), [56](#)

inverse, [52](#)

invhilb, [22](#)

invol, [23](#)

## K

kahan, [24](#)

kms, [25](#)

## L

lehmer, [26](#)

lotkin, [27](#)

## M

magic, [28](#)

matrixdepot() (built-in function), [3–6](#)

MatrixDepot.update() (built-in function), [5](#)

minij, [29](#)

moler, [30](#)

## N

neumann, [31](#)

## O

oscillate, [32](#)

## P

parallax, [50](#)

parter, [33](#)

pascal, [33](#)

pei, [34](#)

phillips, [51](#)  
poisson, [35](#)  
pos-def, [52](#)  
prolate, [36](#)

## R

randcorr, [36](#)  
rando, [37](#)  
random, [52](#)  
randsvd, [38](#)  
regprob, [52](#)  
rohess, [39](#)  
rosser, [40](#)

## S

sampling, [41](#)  
shaw, [51](#)  
smallworld, [48](#)  
sparse, [52](#)  
spikes, [51](#)  
symmetric, [52](#)

## T

toeplitz, [42](#)  
tridiag, [42](#)  
triw, [43](#)

## U

ursell, [51](#)

## V

vand, [44](#)

## W

wathen, [45](#)  
wilkinson, [46](#)  
wing, [51](#)