ISSN 1360-1725

THE UNIVERSITY
of MANCHESTER

UMIST

The Matrix Computation Toolbox for MATLAB (Version 1.0)

N. J. Higham

Numerical Analysis Report No. 410

August 2002

Manchester Centre for Computational Mathematics
Numerical Analysis Reports

DEPARTMENTS OF MATHEMATICS

Reports available from:
Department of Mathematics
University of Manchester
Manchester M13 9PL
England

And over the World-Wide Web from URLs
`http://www.ma.man.ac.uk/MCCM`
`http://www.ma.man.ac.uk/~nareports`

The Matrix Computation Toolbox for MATLAB (Version 1.0)

Nicholas J. Higham[1]

August 23, 2002

[1]Department of Mathematics, University of Manchester, Manchester, M13 9PL, England (higham@ma.man.ac.uk, http://www.ma.man.ac.uk/~higham/).

Contents

1 Introduction **2**
1.1 Citing the Toolbox 2

2 Installation **3**

3 Release History **4**

4 Quick Reference Tables **5**

5 Visualization **7**

6 Direct Search Optimization **13**

7 Test Matrices **16**

Bibliography **19**

Chapter 1
Introduction

The Matrix Computation Toolbox is a collection of MATLAB M-files containing functions for constructing test matrices, computing matrix factorizations, visualizing matrices, and carrying out direct search optimization. Various other miscellaneous functions are also included.

The toolbox was developed in conjunction with the book *Accuracy and Stability of Numerical Algorithms* [8, 2002]. That book is the primary documentation for the toolbox: it describes much of the underlying mathematics and many of the algorithms and matrices (it also describes many of the matrices provided by MATLAB's `gallery` function).

The toolbox is distributed under the terms of the GNU General Public License (`http://www.gnu.org/copyleft/gpl.html`, version 2 of the License, or any later version) as published by the Free Software Foundation.

The toolbox has been tested under MATLAB 6.1 (R12.1) and MATLAB 6.5 (R13).

The M-files in the toolbox are self-documenting and so more detailed documentation than is provided here can be obtained on-line by typing `help M-file_name`. For information about MATLAB functions, and in particular the test matrices in MATLAB, see *MATLAB Guide* [2, 2000].

This document describes version 1.0 of the toolbox, dated August 23, 2002.

1.1. Citing the Toolbox

Please cite the toolbox as follows:

> N. J. Higham. The Matrix Computation Toolbox. `http://www.ma.man.ac.uk/~higham/mctoolbox`.

A BibTeX bib entry is available from the Web page shown.

Chapter 2
Installation

The Matrix Computation Toolbox is available from

`http://www.ma.man.ac.uk/~higham/mctoolbox`

It is provided as a zip file that can be uncompressed with any zip file utility. It is also available from the MATLAB File Exchange at The MathWorks' web site

`http://www.mathworks.com`

It is recommended that the toolbox be installed into a directory `matrixcomp`. To try out the toolbox from within MATLAB, change to the `matrixcomp` directory and run the demonstration script `mctdemo` (by typing `mctdemo`). For serious use it is best to put the `matrixcomp` directory on the MATLAB path, so that the M-files can be called from other directories.

This document is `mctoolbox.pdf` within the zip file.

Chapter 3
Release History

This toolbox supersedes the Test Matrix Toolbox [7, 1995]. Most of the test matrices in that toolbox have been incorporated into MATLAB in the `gallery` function. The new toolbox incorporates some of the other routines in the Test Matrix Toolbox (in some cases renamed) and adds several new ones.

The first release of the Test Matrix Toolbox (version 1.0) was described in a technical report [3, 1989]. The collection was subsequently published as ACM Algorithm 694 [4, 1991]. Ensuing releases were version 2.0 [6, 1993] and version 3.0 [7, 1995] (the final release).

Chapter 4
Quick Reference Tables

This section contains quick reference tables for the Matrix Computation Toolbox. All the M-files in the toolbox are listed by category, with a short description.

Demonstration	
`mctdemo`	Demonstration of Matrix Computation Toolbox.

Test Matrices	
`augment`	Augmented system matrix.
`gfpp`	Matrix giving maximal growth factor for Gaussian elimination with partial pivoting.
`makejcf`	A matrix with specified Jordan canonical form.
`rschur`	An upper quasi-triangular matrix.
`vand`	Vandermonde matrix.
`vecperm`	Vec-permutation matrix.

Factorizations and Decompositions	
`cholp`	Cholesky factorization with complete pivoting of a positive semidefinite matrix.
`cod`	Complete orthogonal decomposition.
`gep`	Gaussian elimination with pivoting: none, complete, partial, or rook.
`gj`	Gauss–Jordan elimination with no pivoting or partial pivoting to solve $Ax = b$.
`gqr`	Generalized QR factorization.
`gs_c`	Classical Gram–Schmidt QR factorization.
`gs_m`	Modified Gram–Schmidt QR factorization.
`ldlt_skew`	Block LDL^T factorization for a skew-symmetric matrix.
`ldlt_symm`	Block LDL^T factorization with partial pivoting or rook pivoting for a symmetric indefinite matrix.
`ldlt_sytr`	Block LDL^T factorization for a symmetric tridiagonal matrix.
`matsignt`	Matrix sign function of a triangular matrix.
`poldec`	Polar decomposition.
`signm`	Matrix sign decomposition.
`trap2tri`	Unitary reduction of trapezoidal matrix to triangular form.

Visualization	
`fv`	Field of values (or numerical range).
`gersh`	Gershgorin disks.
`ps`	Dot plot of a pseudospectrum.
`pscont`	Contours and colour pictures of pseudospectra.
`see`	Pictures of a matrix.

Direct Search Optimization	
`adsmax`	Alternating directions method.
`mdsmax`	Multidirectional search method.
`nmsmax`	Nelder–Mead simplex method.

Miscellaneous	
`chop`	Round matrix elements.
`cpltaxes`	Determine suitable `axis` for plot of complex vector.
`dual`	Dual vector with respect to Hölder p-norm.
`lse`	Solve the equality constrained least squares problem.
`matrix`	Matrix Computation Toolbox information and matrix access by number.
`pnorm`	Estimate of matrix p-norm ($1 \leq p \leq \infty$).
`rootm`	Pth root of a matrix.
`seqcheb`	Sequence of points related to Chebyshev polynomials.
`seqm`	Multiplicative sequence.
`show`	Display signs of matrix elements.
`skewpart`	Skew-symmetric (skew-Hermitian) part.
`sparsify`	Randomly set matrix elements to zero.
`strassen`	Strassen's fast matrix multiplication algorithm.
`strassenw`	Strassen's fast matrix multiplication algorithm (Winograd variant).
`sub`	Principal submatrix.
`symmpart`	Symmetric (Hermitian) part.
`treshape`	Reshape vector to or from (unit) triangular matrix.

Chapter 5
Visualization

The toolbox contains five functions for visualizing matrices. The functions can give insight into the properties of a matrix that is not easy to obtain by looking at the numerical entries. They also provide an easy way to generate pretty pictures!

The function `fv` plots the field of values of a square matrix $A \in \mathbb{C}^{n\times n}$ (also called the numerical range), which is the set of all Rayleigh quotients,

$$\left\{ \frac{x^*Ax}{x^*x} : 0 \neq x \in \mathbb{C}^n \right\};$$

the eigenvalues of A are plotted as crosses. The field of values is a convex set that contains the eigenvalues. It is the convex hull of the eigenvalues when A is a normal matrix. If A is Hermitian, the field of values is just a segment of the real line. For non-Hermitian A the field of values is usually two-dimensional and its shape and size gives some feel for the behaviour of the matrix. Trefethen and Embree [16] note that the field of values is the largest reasonable answer to the question "Where in $\mathbb{C}$ does a matrix A 'live'?" and the spectrum is the smallest reasonable answer.

Some examples of field of values plots are given in Figure 5.1. The matrix `gallery('circul',...)` is normal, hence its field of values is the convex hull of the eigenvalues. For an example of how the field of values gives insight into the problem of finding a nearest normal matrix see [12, 1987]. An excellent reference for the theory of the field of values is [9, 1985, Chapter 1].

The function `gersh` plots the Gershgorin disks of $A \in \mathbb{C}^{n\times n}$, which are the n disks

$$D_i = \{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \}$$

in the complex plane. Gershgorin's theorem tells us that the eigenvalues of A lie in the union of the disks, and an extension of the theorem states that if k disks form a connected region that is isolated from the other disks then there are precisely k eigenvalues in this region. Thus the size of the disks gives a feel for how nearly diagonal A is, and their locations give information on where the eigenvalues lie in the complex plane. Four examples of Gershgorin disk plots are given in Figure 5.2; Gershgorin's theorem provides nontrivial information only for the third matrix, `ipjfact(8,1)`.

The functions `ps` and `pscont` plot pseudospectra. The ϵ-pseudospectrum of a

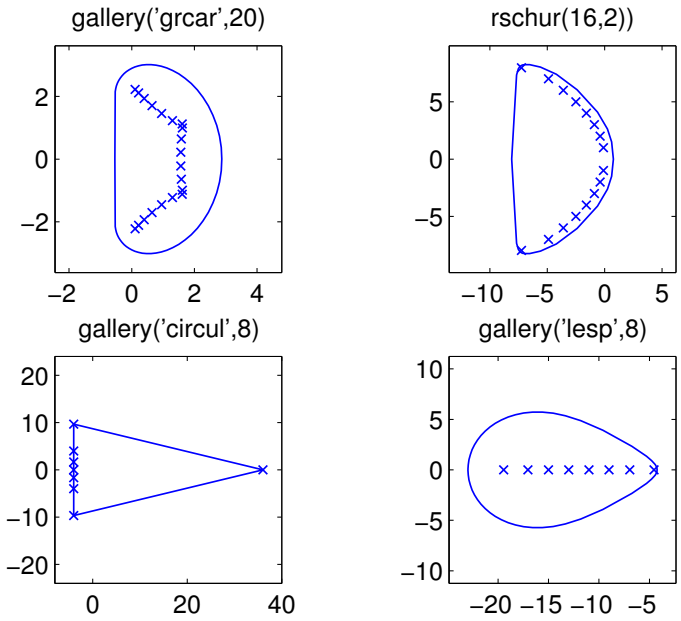

Figure 5.1. *Fields of values* (`fv`).

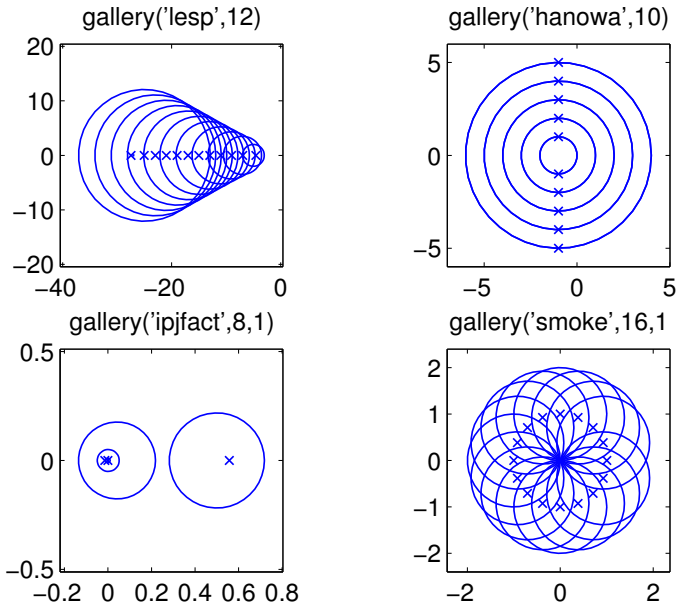

Figure 5.2. *Gershgorin disks* (`gersh`).

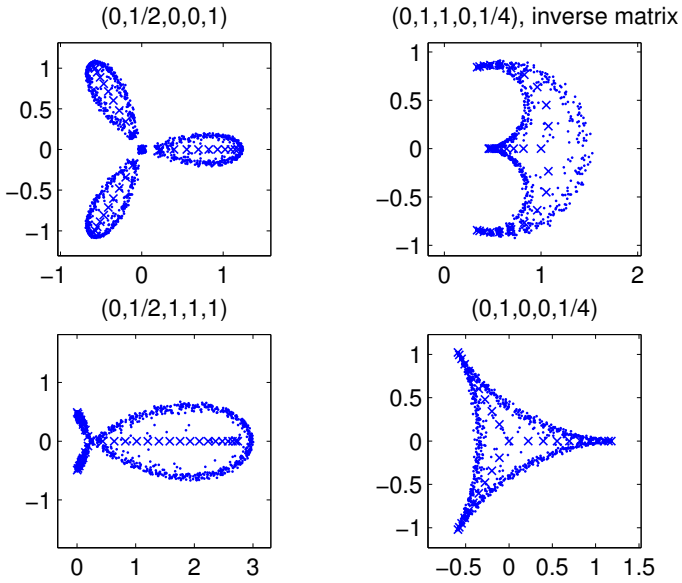

Figure 5.3. *Pseudospectra of* 32×32 *pentadiagonal Toeplitz matrices,* `gallery('toeppen',32,a,b,c,d,e)`. *Shown are the parameters* `(a,b,c,d,e)`.

matrix $A \in \mathbb{C}^{n\times n}$ is defined, for a given $\epsilon > 0$, to be the set

$$\Lambda_\epsilon(A) = \{\, z : z \text{ is an eigenvalue of } A + E \text{ for some } E \text{ with } \|E\|_2 \le \epsilon \,\}.$$

In other words, it is the set of all complex numbers that are eigenvalues of $A + E$ for some perturbation E of 2-norm at most ϵ. For a normal matrix A the ϵ-pseudospectrum is the union of the balls of radius ϵ around the eigenvalues of A. For nonnormal matrices the ϵ-pseudospectrum can take a wide variety of shapes and sizes, depending on the matrix and how nonnormal it is. Pseudospectra play an important role in many numerical problems. For full details see the work of Trefethen—in particular, [1], [14, 1999], [15, 1999], [16].

The routine `ps` plots an approximation to the ϵ-pseudospectrum $\Lambda_\epsilon(A)$, which it obtains by computing the eigenvalues of a given number of random perturbations of A. The eigenvalues are plotted as crosses and the pseudo-eigenvalues as dots. Arguments to `ps` control the number and type of perturbations. Figure 5.3, taken from [8, 2002, Fig. 28.3], gives four examples of 10^{-3}-pseudospectra, all of which involve the pentadiagonal Toeplitz matrix `gallery('pentoep'...)`.

Another characterization of $\Lambda_\epsilon(A)$, in terms of the resolvent $(zI - A)^{-1}$, is

$$\Lambda_\epsilon(A) = \{\, z : \|(zI - A)^{-1}\|_2 \ge \epsilon^{-1} \,\}.$$

An alternative way of viewing the pseudospectrum is to plot the function

$$f(z) = \|(zI - A)^{-1}\|_2^{-1} = \sigma_{\min}(zI - A)$$

over the complex plane, where $\sigma_{\min}$ denotes the smallest singular value [14, 1999]. The routine `pscont` plots $\log_{10} f(z)^{-1}$ and offers several ways to view the surface:

by its contour lines alone, or as a coloured surface plot in two or three dimensions, with or without contour lines. (The two-dimensional plot is the view from directly above the surface.) Two different `pscont` views of the pseudospectra of the triangular matrix `gallery('triw',11)` are given in Figures 5.4 and 5.5. Since all the eigenvalues of this matrix are equal to 1, there is a single point where the resolvent is unbounded in norm—this is the "bottomless pit" in the pictures. The spike in Figure 5.5 should be infinitely deep; since `pscont` evaluates $f(z)$ on a finite grid, the spike has a finite depth dependent on the grid spacing. Also because of the grid spacing chosen, the contours are a little jagged. Various aspects of the plots can be changed from the MATLAB command line upon return from `pscont`; for example, the colour map (`colormap`), the shading (`shading`), and the viewing angle (`view`). For Figure 5.4 we set `shading interp` and `colormap copper`.

Both pseudospectrum functions are computationally intensive, so the defaults for the arguments are chosen to produce a result in a reasonable time; for plots that reveal reasonable detail it is usually necessary to override the defaults.

Note that `pscont` is not written to be efficient. A much more efficient and versatile tool is the GUI `eigtool` by Tom Wright [17].

The function `see` displays a figure with four subplots in the format

`mesh(A)`	`semilogy(svd(A))`
`ps(A)`	`fv(A)`

An example for the `gallery('chebvand',...)` matrix is given in Figure 5.6. MATLAB's `mesh` command plots a three-dimensional, coloured, wire-frame surface, by regarding the entries of a matrix as specifying heights above a plane. We use `axis('ij')`, so that the coordinate system for the plot matches the (i, j) matrix element numbering. `semilogy(svd(A))` plots the singular values of A (ordered in decreasing size) on a logarithmic scale; the singular values are denoted by circles, which are joined by a solid line to emphasize the shape of the distribution.

For a sparse MATLAB matrix, `see` simply displays a `spy` plot, which shows the sparsity pattern of the matrix. The user could, of course, try `see(full(A))` for a sparse matrix, but for large dimensions the storage and time required would be prohibitive.

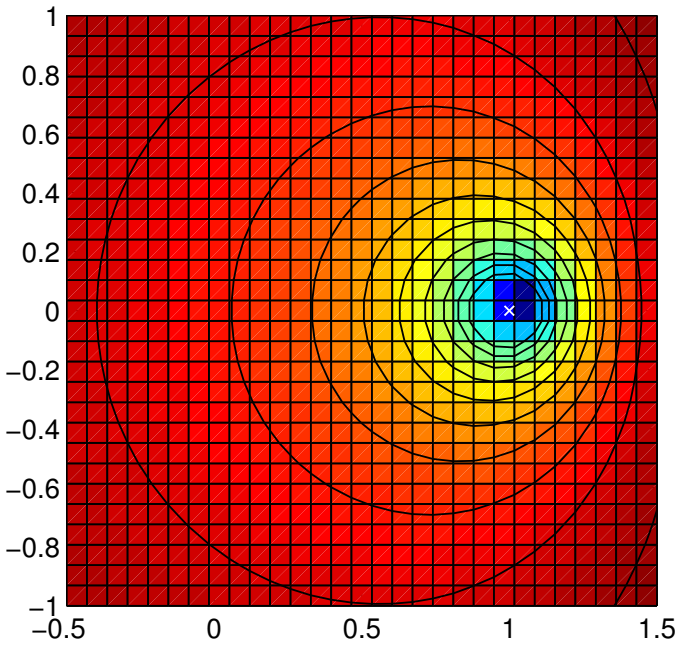

Figure 5.4. `pscont(gallery('triw',11), 0, 30, [-0.5 1.5 -1 1])`.

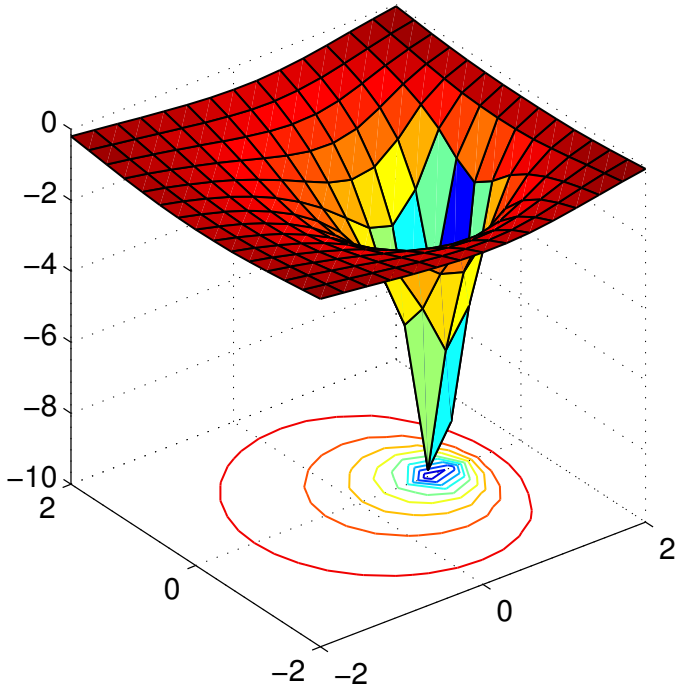

Figure 5.5. `pscont(gallery('triw',11), 2, 15, [-2 2 -2 2])`.

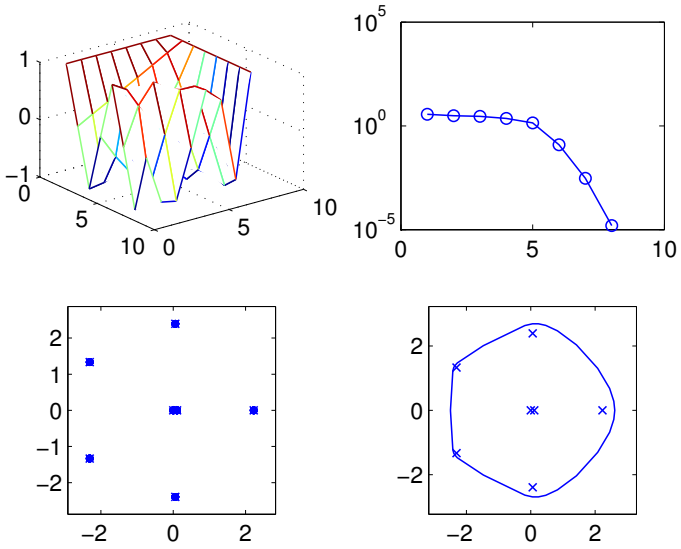

Figure 5.6. `see(gallery('chebvand',8))`.

Chapter 6
Direct Search Optimization

The toolbox contains three multivariate direct search maximization functions: `mdsmax`, `adsmax` and `nmsmax`. The functions are competitors to `fminsearch`, which is supplied with MATLAB (but `fmins` minimizes rather than maximizes).

`mdsmax`, `adsmax` and `nmsmax` are direct search methods (as is `fminsearch`), that is, they attempt to maximize a real function f of a vector argument x using function values only. `mdsmax` uses the multidirectional search method [10, 1999, §8.2], [13, 1991], `adsmax` uses the method of alternating directions, and `nmsmax` (like `fminsearch`) uses the Nelder–Mead simplex method [11, 1965], [10, 1999, §8.1]. In general, `mdsmax` and `nmsmax` can be expected to perform better than `adsmax` since they use a more sophisticated method.

These routines were developed during the work described in [5].

It is important to note that these routines, like `fminsearch`, are not competitive with more sophisticated methods such as (quasi-)Newton methods when applied to smooth problems. They are at their best when applied to non-smooth problems such as the one in the example below.

The routines are fully documented in their leading comment lines, but it is appropriate to add here a few comments about the format of the output and the use of the `savit` argument.

`mdsmax` produces output to the screen (this can be suppressed by setting the input argument `stopit(5) = 0`). The output is illustrated by

```
Iter. 10,  inner =  2,  size = -4,  nf = 401,  f = 4.7183e+001  (51.0%)
```

The means that on the tenth iteration, two inner iterations were required, and at the end of the iteration the simplex edges were 2^{-4} times the length of those of the initial simplex. Further, `nf` is the total number of function evaluations so far, `f` is the current highest function value, and the percentage increase in function value over the tenth iteration is 51%.

The output produced by `adsmax` is similar to that of `mdsmax` and is illustrated by the following extract from the start of the second outer iteration:

```
Iter  2  (nf = 146)
Comp. =  1,  steps = 12,  f = 1.5607e+000  (0.4%)
```

`Comp` denotes the component of x being varied on the current stage and steps is the number of steps in the crude line search for this stage.

The output from `nmsmax` is also similar to that from `mdsmax`, but only iterations on which an increase in the function value is achieved are reported.

In all three routines, if a non-empty fourth input argument string `savit` is present then at the end of each iteration the following "snapshot" is written to the file specified by `savit`: the largest function value found so far, `fmax`, the point at which it is achieved, `x`, and the total number of function evaluations, `nf`. This option enables the user to abort an optimization, load and examine `x`, `fmax` and `nf` using MATLAB's `load` command, and then possibly restart the optimization at `x`.

One further point worth mentioning is that `mdsmax`, `adsmax` and `nmsmax` always call the function `f` to be maximized with an argument of the same dimensions (or shape) as the starting value `x0`. Similarly, the output argument `x` and the variable `x` in `savit` saves have the same shape as `x0`. This feature is very convenient when `f` is a function of a matrix, as in the example below.

To illustrate, here is an example of how to use `mdsmax` to search for a matrix for which `rcond(A)`) overestimates the reciprocal of the exact 1-norm condition number of the matrix `A` (`rcond` is MATLAB's built-in condition estimator). The code

```
frcond = inline('cond(A,1)*rcond(A)');
A = hilb(5);  % Starting matrix.
B = mdsmax(@frcond, A)
C = nmsmax(@frcond, B)
```

uses the output from `mdsmax` as starting matrix for `nmsmax` (a useful technique), and produces the output

```
f(x0) = 1.0000e+000
Iter.  1,  inner =  0,  size =  0,  nf =  26,  f = 1.0000e+000  (0.0%)
Iter.  2,  inner =  1,  size = -1,  nf =  76,  f = 1.0000e+000  (0.0%)
Iter.  3,  inner =  2,  size = -3,  nf = 176,  f = 1.0066e+000  (0.7%)
Iter.  4,  inner =  1,  size = -3,  nf = 226,  f = 1.3521e+000  (34.3%)
Iter.  5,  inner =  1,  size = -3,  nf = 276,  f = 1.5476e+000  (14.5%)
Iter.  6,  inner =  2,  size = -5,  nf = 376,  f = 1.5697e+000  (1.4%)
Iter.  7,  inner =  1,  size = -4,  nf = 426,  f = 1.7432e+000  (11.1%)
Iter.  8,  inner =  1,  size = -3,  nf = 476,  f = 1.9645e+000  (12.7%)
Iter.  9,  inner =  1,  size = -3,  nf = 526,  f = 3.8308e+000  (95.0%)
Iter. 10,  inner =  1,  size = -3,  nf = 576,  f = 4.0586e+000  (5.9%)
Iter. 11,  inner =  1,  size = -4,  nf = 626,  f = 4.6397e+000  (14.3%)
Iter. 12,  inner =  1,  size = -5,  nf = 676,  f = 4.7817e+000  (3.1%)
Iter. 13,  inner =  2,  size = -7,  nf = 776,  f = 4.8090e+000  (0.6%)
Simplex size 8.5236e-004 <= 1.0000e-003...quitting
B =
    1.1464    0.5635    0.3913   -0.0014    0.2580
    0.5580    0.4576    0.3080    0.2580    0.2246
    0.6565    0.3080    0.2580    0.2246    0.2008
    0.3245    0.2580    0.2246    0.4218    0.1830
    0.2580    0.2246    0.2008    0.1830    0.1691
f(x0) = 4.8090e+000
Iter. 190,  how = reflect,   nf = 475,  f = 4.8357e+000  (0.6%)
Iter. 199,  how = reflect,   nf = 486,  f = 4.8361e+000  (0.0%)
```

```
Iter. 200,  how = expand,    nf = 488,  f = 4.9610e+000  (2.6%)
Iter. 223,  how = reflect,   nf = 520,  f = 4.9650e+000  (0.1%)
Iter. 224,  how = reflect,   nf = 522,  f = 4.9749e+000  (0.2%)
Iter. 225,  how = expand,    nf = 524,  f = 5.0839e+000  (2.2%)
Iter. 239,  how = expand,    nf = 542,  f = 5.1866e+000  (2.0%)
Iter. 249,  how = reflect,   nf = 554,  f = 5.1913e+000  (0.1%)
Iter. 250,  how = reflect,   nf = 556,  f = 5.2090e+000  (0.3%)
Iter. 251,  how = reflect,   nf = 558,  f = 5.2163e+000  (0.1%)
Iter. 252,  how = reflect,   nf = 560,  f = 5.2330e+000  (0.3%)
Iter. 253,  how = reflect,   nf = 562,  f = 5.2464e+000  (0.3%)
Iter. 254,  how = reflect,   nf = 564,  f = 5.2606e+000  (0.3%)
Iter. 256,  how = expand,    nf = 567,  f = 5.3933e+000  (2.5%)
Iter. 268,  how = expand,    nf = 581,  f = 5.5486e+000  (2.9%)
Iter. 286,  how = expand,    nf = 604,  f = 5.7267e+000  (3.2%)
Iter. 302,  how = expand,    nf = 624,  f = 5.8959e+000  (3.0%)
Iter. 310,  how = expand,    nf = 633,  f = 6.1229e+000  (3.9%)
... (output omitted)
Iter. 927,  how = reflect,   nf = 1430,  f = 2.6536e+002  (4.5%)
Iter. 938,  how = reflect,   nf = 1447,  f = 2.7461e+002  (3.5%)
Iter. 944,  how = reflect,   nf = 1457,  f = 2.7752e+002  (1.1%)
Iter. 962,  how = reflect,   nf = 1487,  f = 2.8312e+002  (2.0%)
Simplex size 8.5771e-004 <= 1.0000e-003...quitting
C =
    1.1571    0.5621    0.3576   -0.0102    0.2625
    0.5275    0.5064    0.3115    0.2837    0.2286
    0.6478    0.3180    0.2752    0.2533    0.2099
    0.3254    0.2455    0.2418    0.4101    0.1869
    0.2490    0.2156    0.2152    0.1755    0.1698
```

For more on the use of direct search in "automatic error analysis" see [8, 2002, Ch. 26].

Chapter 7
Test Matrices

The function `matrix` provides easy access to most of the test matrices in MATLAB and those in the Matrix Computation Toolbox. With no arguments the `matrix` function lists the available matrices by number:

```
>> matrix
 1: cauchy    12: frank     23: lesp      34: randsvd   45: rand
 2: chebspec  13: gearmat   24: lotkin    35: redheff   46: randn
 3: chebvand  14: grcar     25: minij     36: riemann   47: augment
 4: chow      15: invhess   26: moler     37: ris       48: gfpp
 5: circul    16: invol     27: orthog    38: smoke     49: magic
 6: clement   17: ipjfact   28: parter    39: toeppd    50: makejcf
 7: condex    18: jordbloc  29: pei       40: triw      51: rschur
 8: cycol     19: kahan     30: prolate   41: hilb      52: vand
 9: dramadah  20: kms       31: randcolu  42: invhilb
10: fiedler   21: krylov    32: randcorr  43: magic
11: forsythe  22: lehmer    33: rando     44: pascal
Matrices 1 to 46 are from MATLAB
```

Invoking `matrix(k,n,...)` produces the `n`-by-`n` instance of the `k`th of these matrices, all of which are full (as opposed to sparse). This provides a convenient way to run through a batch of test matrices. For example, the code

```
c = []; j = 1;

% Make experiment repeatable for the random matrices.
randn('seed',1), rand('seed',1)

fprintf('Matrix     Ratio\n-------------------\n')
for k=1:matrix(0)

    % Double on next line avoids bug in MATLAB 6.5 re. matrix(35).
    A = double(matrix(k, 16));
    c = [c max(abs(eig(A)))/norm(A)];
    name = [matrix(k) '         '];
    fprintf([name(1:8) ': %9.1e\n'], c(end))

end
```

runs through the set of 52 matrices evaluating the ratio $|\max_i \lambda_i(A)|/\|A\|_2$ for dimension 16, where $\{\lambda_i\}$ is the set of eigenvalues of A. This ratio is known theoretically to lie between 0 and 1. For only one of the matrices (`invol`) is the ratio significantly less than 1:

```
Matrix     Ratio
-------------------
cauchy  :  1.0e+000
chebspec:  1.2e-002
chebvand:  7.4e-001
chow    :  3.5e-001
circul  :  1.0e+000
clement :  9.4e-001
condex  :  1.0e+000
cycol   :  4.2e-001
dramadah:  7.5e-001
fiedler :  1.0e+000
forsythe:  3.2e-001
frank   :  5.8e-001
gearmat :  9.8e-001
grcar   :  7.0e-001
invhess :  9.7e-001
invol   :  4.9e-011
ipjfact :  1.0e+000
jordbloc:  5.0e-001
kahan   :  3.8e-001
kms     :  1.0e+000
krylov  :  3.5e-001
lehmer  :  1.0e+000
lesp    :  7.8e-001
lotkin  :  6.4e-001
minij   :  1.0e+000
moler   :  1.0e+000
orthog  :  1.0e+000
parter  :  9.5e-001
pei     :  1.0e+000
prolate :  1.0e+000
randcolu:  6.8e-001
randcorr:  1.0e+000
rando   :  9.7e-001
randsvd :  1.9e-001
redheff :  9.2e-001
riemann :  9.1e-001
ris     :  1.0e+000
smoke   :  5.2e-001
toeppd  :  1.0e+000
triw    :  1.1e-001
hilb    :  1.0e+000
```

```
invhilb :  1.0e+000
magic   :  1.0e+000
pascal  :  1.0e+000
rand    :  9.7e-001
randn   :  5.5e-001
augment :  1.0e+000
gfpp    :  5.0e-001
magic   :  1.0e+000
makejcf :  8.6e-002
rschur  :  9.2e-001
vand    :  2.6e-001
```

Bibliography

[1] Mark Embree and Lloyd N. Trefethen. Pseudospectra gateway. `http://www.comlab.ox.ac.uk/pseudospectra/`.

[2] Desmond J. Higham and Nicholas J. Higham. *MATLAB Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. xxii+283 pp. ISBN 0-89871-516-4.

[3] Nicholas J. Higham. A collection of test matrices in MATLAB. Numerical Analysis Report No. 172, University of Manchester, Manchester, England, July 1989.

[4] Nicholas J. Higham. Algorithm 694: A collection of test matrices in MATLAB. *ACM Trans. Math. Software*, 17(3):289–305, September 1991.

[5] Nicholas J. Higham. Optimization by direct search in matrix computations. *SIAM J. Matrix Anal. Appl.*, 14(2):317–333, April 1993.

[6] Nicholas J. Higham. The Test Matrix Toolbox for MATLAB. Numerical Analysis Report No. 237, Manchester Centre for Computational Mathematics, Manchester, England, December 1993. 76 pp.

[7] Nicholas J. Higham. The Test Matrix Toolbox for MATLAB (version 3.0). Numerical Analysis Report No. 276, Manchester Centre for Computational Mathematics, Manchester, England, September 1995. 70 pp.

[8] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Second edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680 pp. ISBN 0-89871-521-0.

[9] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991. viii+607 pp. ISBN 0-521-30587-X.

[10] C. T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. xv+180 pp. ISBN 0-89871-433-8.

[11] J. A. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 7:308–313, 1965.

[12] Axel Ruhe. Closest normal matrix finally found! *BIT*, 27:585–598, 1987.

[13] Virginia J. Torczon. On the convergence of the multidirectional search algorithm. *SIAM J. Optim.*, 1(1):123–145, 1991.

[14] Lloyd N. Trefethen. Computation of pseudospectra. *Acta Numerica*, 8:247–295, 1999.

[15] Lloyd N. Trefethen. Spectra and pseudospectra. In *The Graduate Student's Guide to Numerical Analysis '98*, Mark Ainsworth, Jeremy Levesley, and Marco Marletta, editors, Springer-Verlag, Berlin, 1999, pages 217–250.

[16] Lloyd N. Trefethen and Mark Embree. *Spectra and Pseudospectra: The Behavior of Non-Normal Matrices and Operators*. Book in preparation.

[17] Thomas G. Wright. Eigtool. `http://www.comlab.ox.ac.uk/pseudospectra/eigtool/`.