MATLAB Toolbox for Classical Matrix Groups

Jagger, David P.

2003

MIMS EPrint: **2007.99**

ISSN 1749-9097

MATLAB TOOLBOX FOR CLASSICAL MATRIX GROUPS

A thesis submitted to the University of Manchester
for the degree of Master of Science
in the Faculty of Science and Engineering

2003

David P. Jagger
Department of Mathematics

Contents

List of Tables

List of Figures

Abstract

We consider structured matrix groups arising in the context of nondegenerate bilinear or sesquilinear forms on $\mathbb{R}^n$ or $\mathbb{C}^n$, which remain invariant under similarities by matrices in their associated automorphism group. We develop a MATLAB toolbox for generating random matrices from these automorphism groups with, wherever possible, prescribed condition numbers. The matrix groups considered are the complex orthogonal, real, complex and conjugate symplectic, real perplectic, real and complex pseudo-orthogonal, and pseudo-unitary groups.

We outline all necessary background theory before presenting a self-contained treatment of each group, outlining some applications of the group and deriving an algorithm for their random generation. We first focus our attention on the groups for which a structured SVD or CSD is available, and show that this allows for precise control of the condition number via numerically stable algorithms. We then consider the groups which lack such a decomposition, where we construct matrices via products of generalized $\mathbb{G}$-reflectors. We perform tests which model the behaviour of the condition number in these cases, allowing for its approximate control, and finally we consider the effect of rounding errors on the resultant matrices.

The implementation in MATLAB of these algorithms is hoped to be beneficial for researchers developing structure-preserving algorithms for structured problems.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Copyright

Acknowledgements

I would like to thank my supervisor, Nick Higham, for his guidance and many helpful suggestions throughout the preparation of this thesis, Steve Mackey for his invaluable replies to my e-mails, and the EPSRC for funding my studies.

Thanks also go to my family for their love and support, my friends for appreciating the need for beer and rock 'n' roll, and to my dearly-missed grandad, Frank Jagger, for his unwavering encouragement — my memories of his words will always be an inspiration.

Chapter 1

Introduction

Structured matrices arise in various problems with intrinsic symmetries or periodicity occurring in areas of engineering, physics, and statistics. For example the eigenvalue problem for Hamiltonian matrices arises in many algorithms of control theory and in computational physics and chemistry problems. Another example is Toeplitz matrices, which arise in estimation theory and signal processing and have many further applications in physics. Many such structures remain invariant under similarities by matrices in an appropriate matrix group, and preserving the structure when transforming these problems is very desirable, for reasons including minimizing computational cost and storage demands, and maintaining inherent physical properties and matrix properties. As a result, new structure-preserving algorithms have been developed in recent years for factorizing and transforming structured problems.

We will consider structured matrices occurring in the context of nondegenerate bilinear or sesquilinear forms on $\mathbb{R}^n$ or $\mathbb{C}^n$. Each form gives rise to three classes of matrices: an automorphism group, a Lie algebra, and a Jordan algebra. The important relationship between these classes is that the Lie and Jordan algebras remain invariant under similarities by matrices in their associated automorphism group, and thus these automorphism groups in particular are vital to the further study and development of structure-preserving algorithms.

In this thesis, the automorphism groups considered are the complex orthogonals, real pseudo-orthogonals, complex pseudo-orthogonals, pseudo-unitaries, real perplectics, real symplectics, complex symplectics, and conjugate symplectics. The algebras associated with these groups include complex symmetric, pseudo-symmetric, persymmetric, pseudo-Hermitian, Hamiltonian, and J-symmetric matrices. For example the Lie algebra associated with the real symplectic group encompasses Hamiltonian matrices, and hence we have invariance of Hamiltonian matrices under real symplectic similarities.

Our aim is to develop a MATLAB toolbox for generating matrices from the automorphism groups listed above with, wherever possible, prescribed condition numbers. It is expected that this work will be beneficial towards the development of structure-preserving algorithms for structured problems. The random structured matrices generated by the toolbox will be useful for generating random test problems to ensure robustness and quality of newly-derived algorithms before they are made available to users.

The thesis is organized as follows. The following section introduces notation and the background theory required for the development of algorithms for structured matrix generation. In Chapter 2 we consider the groups for which a structured SVD or CS decomposition is available, which allows us to develop numerically stable algorithms for generating random matrices in these groups involving multiplication by orthogonal or unitary matrices, and furthermore allows for precise control of the condition number. The groups which lack a structured SVD or CSD are then considered in Chapter 3, where we form matrices via products of elementary matrix transformations known as $\mathbb{G}$-reflectors, and consider empirical results to model the behaviour of the condition numbers of the resultant matrices and the effects of rounding errors on their structures. In Chapter 4, we draw our conclusions, and all necessary MATLAB code is included in Appendix A.

1.1 Preliminaries

1.1.1 Scalar products

The following is a brief outline of the definitions and properties of scalar products. A more detailed review can be found, for example, in Jacobson [10] or Shaw [17].

Let $\mathbb{K}$ denote either the field $\mathbb{R}$ or $\mathbb{C}$. Consider a map $(x, y) \mapsto \langle x, y\rangle$ from $\mathbb{K}^n \times \mathbb{K}^n$ to $\mathbb{K}$. If the map is linear in each argument, i.e.

$$\begin{aligned}\langle \alpha_1 x_1 + \alpha_2 x_2, y\rangle &= \alpha_1\langle x_1, y\rangle + \alpha_2\langle x_2, y\rangle,\\ \langle x, \beta_1 y_1 + \beta_2 y_2\rangle &= \beta_1\langle x, y_1\rangle + \beta_2\langle x, y_2\rangle,\end{aligned}$$

then it is called a *bilinear form.* If $\mathbb{K} = \mathbb{C}$ and the map is conjugate linear in the first argument and linear in the second, i.e.

$$\begin{aligned}\langle \alpha_1 x_1 + \alpha_2 x_2, y\rangle &= \overline{\alpha}_1\langle x_1, y\rangle + \overline{\alpha}_2\langle x_2, y\rangle,\\ \langle x, \beta_1 y_1 + \beta_2 y_2\rangle &= \beta_1\langle x, y_1\rangle + \beta_2\langle x, y_2\rangle,\end{aligned}$$

then it is called a *sesquilinear form.*

Given a bilinear or sesquilinear form over $\mathbb{K}^n$, there exists a unique $M \in \mathbb{K}^{n\times n}$ such that for all $x, y \in \mathbb{K}^n$,

$$\langle x, y\rangle = \begin{cases} x^T M y & \text{if the form is bilinear,}\\ x^* M y & \text{if the form is sesquilinear.}\end{cases}$$

Here, the superscript $*$ is used for conjugate transpose. We call M the matrix associated with the form, and will denote $\langle x, y\rangle = \langle x, y\rangle_{\mathrm{M}}$ as required.

A bilinear form is called symmetric if $\langle x, y\rangle = \langle y, x\rangle$ and skew-symmetric if $\langle x, y\rangle = -\langle y, x\rangle$, and it can be shown that the matrix associated with a (skew-) symmetric form is itself (skew-)symmetric. A sesquilinear form is called Hermitian if $\langle x, y\rangle = \overline{\langle y, x\rangle}$ and skew-Hermitian if $\langle x, y\rangle = -\overline{\langle y, x\rangle}$, and the matrices associated with these forms are Hermitian and skew-Hermitian respectively.

A bilinear or sesquilinear form $\langle\cdot,\cdot\rangle_{\mathrm{M}}$ is *nondegenerate* if M is nonsingular, and from now on the term *scalar product* refers to a nondegenerate bilinear or sesquilinear form on $\mathbb{K}^n$. We will also use the associated quadratic functional $q_{\mathrm{M}}(x) = \langle x, x\rangle_{\mathrm{M}}$, the natural analogue of the quantity x^*x for vectors in $\mathbb{K}^n$ in Euclidean space. We say a nonzero vector $x \in \mathbb{K}^n$ is *isotropic* with respect to a given scalar product if $q_{\mathrm{M}}(x) = 0$, and non-isotropic otherwise.

For any $A \in \mathbb{K}^{n\times n}$ there exists a unique matrix $A^\star$, the *adjoint* of A with respect to $\langle\cdot,\cdot\rangle_{\mathrm{M}}$, defined by

$$\langle Ax, y\rangle_{\mathrm{M}} = \langle x, A^\star y\rangle_{\mathrm{M}}, \quad \forall x, y \in \mathbb{K}^n.$$

We obtain an explicit formula for $A^\star$ as follows. If $\langle\cdot,\cdot\rangle_{\mathrm{M}}$ is bilinear, then

$$\langle Ax, y\rangle_{\mathrm{M}} = x^T A^T My = x^T MM^{-1}A^T My = \langle x, M^{-1}A^T My\rangle_{\mathrm{M}},$$

and hence $A^\star = M^{-1}A^TM$. Similarly, when $\langle\cdot,\cdot\rangle_{\mathrm{M}}$ is sesquilinear, we have that $A^\star = M^{-1}A^*M$.

1.1.2 Automorphism groups, and Lie and Jordan algebras

As mentioned previously, there are three important classes of structured matrices associated with every scalar product:
the automorphism group $\mathbb{G}$, defined by

$$\begin{aligned}\mathbb{G} &= \left\{G \in \mathbb{K}^{n\times n} : \langle Gx, Gy\rangle_{\mathrm{M}} = \langle x, y\rangle_{\mathrm{M}}\right\}\\ &= \left\{G \in \mathbb{K}^{n\times n} : G^\star = G^{-1}\right\},\end{aligned}$$

the Jordan algebra $\mathbb{J}$, defined by

$$\begin{aligned}\mathbb{J} &= \left\{A \in \mathbb{K}^{n\times n} : \langle Ax, y\rangle_{\mathrm{M}} = \langle x, Ay\rangle_{\mathrm{M}}\right\}\\ &= \left\{A \in \mathbb{K}^{n\times n} : A^\star = A\right\},\end{aligned}$$

and the Lie algebra $\mathbb{L}$, defined by

$$\begin{aligned}\mathbb{L} &= \left\{A \in \mathbb{K}^{n\times n} : \langle Ax, y\rangle_{\mathrm{M}} = -\langle x, Ay\rangle_{\mathrm{M}}\right\}\\ &= \left\{A \in \mathbb{K}^{n\times n} : A^\star = -A\right\}.\end{aligned}$$

The automorphism group $\mathbb{G}$, despite not being a linear subspace, always forms a multiplicative group. $\mathbb{L}$ is closed with respect to the Lie bracket $[A_1, A_2] = A_1A_2 - A_2A_1$, and $\mathbb{J}$ is closed with respect to the Jordan product $\{B_1, B_2\} = B_1B_2 + B_2B_1$, and both groups are linear subspaces.

The key result is that for any $G \in \mathbb{G}$ we can show (see [13] for a proof) that

$$A \in \mathbb{S} \quad \Rightarrow \quad G^{-1}AG \in \mathbb{S}, \quad \text{where} \quad \mathbb{S} = \mathbb{G}, \mathbb{L}, \text{or } \mathbb{J},$$

which establishes the fundamental relationship that the automorphism groups $\mathbb{G}$ are the natural classes of structure-preserving similarities for matrices in $\mathbb{G}$, $\mathbb{L}$, and $\mathbb{J}$. Since the aim of this thesis is to develop algorithms to generate random matrices in a selection of automorphism groups, we now switch our attention solely to matrices in $\mathbb{G}$. Table 1.1 shows the structured matrices under consideration in this thesis (along with the orthogonal and unitary groups (see Section 1.1.6)), associated with their underlying scalar products. These scalar products use one of the following matrices for M: the $n \times n$ identity matrix I_n,

$$J = \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix}, \quad R = \begin{bmatrix} & & 1 \\ & ⋰ & \\ 1 & & \end{bmatrix}, \quad \Sigma_{p,q} = \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix}, \text{ with } p + q = n.$$

The automorphism groups will be defined individually later as we consider each group in turn in Chapters 2 and 3. Table 1.1 also introduces notation for each automorphism group that will be used throughout.

1.1.3 $\mathbb{G}$-reflectors

Using the terminology of Householder [9], we define an *elementary transformation* as a linear map $T : \mathbb{K}^n \to \mathbb{K}^n$ of the form $T = I + uv^*$ for some nonzero $u, v \in \mathbb{K}^n$. It is easy to see that T has an $(n-1)$-dimensional fixed point subspace $\mathcal{H} = \{x \in \mathbb{K}^n : Tx = x\}$, i.e., a hyperplane $\mathcal{H}$ on which T acts as the identity. In Chapter 3, we will be interested in elementary transformations in automorphism groups $\mathbb{G}$ associated with certain scalar products.

Table 1.1: Structured matrices to be studied, with their associated scalar products.

Space	**Bilinear Form** $\langle x,y\rangle$	**Adjoint** $A \in M_n(\mathbb{K})$	**Automorphism Group** $\mathbb{G} = \{G : G^\star = G^{-1}\}$	**Jordan Algebra** $\mathbb{J} = \{S : S^\star = S\}$	**Lie Algebra** $\mathbb{L} = \{K : K^\star = -K\}$
$\mathbb{R}^n$	$x^T y$ symmetric form	$A^\star = A^T$	Real orthogonals $O(n,\mathbb{R})$	Symmetrics	Skew-symmetrics
$\mathbb{C}^n$	$x^T y$ symmetric form	$A^\star = A^T$	Complex orthogonals $O(n,\mathbb{C})$	Complex symmetrics	Complex skew-symmetrics
$\mathbb{R}^n$	$x^T \Sigma_{p,q} y$ symmetric form	$A^\star = \Sigma_{p,q} A^T \Sigma_{p,q}$	Pseudo-orthogonals $O(p,q,\mathbb{R})$	Pseudo symmetrics	Pseudo skew-symmetrics
$\mathbb{C}^n$	$x^T \Sigma_{p,q} y$ symmetric form	$A^\star = \Sigma_{p,q} A^T \Sigma_{p,q}$	Complex pseudo-orthogonals $O(p,q,\mathbb{C})$	Complex pseudo-symmetrics	Complex pseudo-skew-symmetrics
$\mathbb{R}^n$	$x^T R y$ symmetric form	$A^\star = RA^T R$	Real perplectics $\mathcal{P}(n)$	Persymmetrics	Perskew-symmetrics
$\mathbb{R}^{2n}$	$x^T J y$ skew-symm. form	$A^\star = -JA^T J$	Real symplectics $Sp(2n,\mathbb{R})$	Skew-Hamiltonians	Hamiltonians
$\mathbb{C}^{2n}$	$x^T J y$ skew-symm. form	$A^\star = -JA^T J$	Complex symplectics $Sp(2n,\mathbb{C})$	J-skew-symmetric	J-symmetric

Space	**Sesquilinear Form** $\langle x,y\rangle$	**Adjoint** $A \in M_n(\mathbb{C})$	**Automorphism Group** $\mathbb{G} = \{G : G^\star = G^{-1}\}$	**Jordan Algebra** $\mathbb{J} = \{S : S^\star = S\}$	**Lie Algebra** $\mathbb{L} = \{K : K^\star = -K\}$
$\mathbb{C}^n$	$x^* y$ Hermitian form	$A^\star = A^*$	Unitaries $U(n)$	Hermitian	Skew-Hermitian
$\mathbb{C}^n$	$x^* \Sigma_{p,q} y$ Hermitian form	$A^\star = \Sigma_{p,q} A^* \Sigma_{p,q}$	Pseudo-unitaries $U(p,q)$	Pseudo Hermitian	Pseudo skew-Hermitian
$\mathbb{C}^{2n}$	$x^* J y$ skew-Herm. form	$A^\star = -JA^* J$	Conjugate symplectics $Sp^*(2n,\mathbb{C})$	J-skew-Hermitian	J-Hermitian

Definition 1.1 Let $\mathbb{G}$ be the automorphism group associated with a scalar product on $\mathbb{K}^n$. The elementary transformations in $\mathbb{G}$ will be referred to as *generalized* $\mathbb{G}$-*reflectors*, or $\mathbb{G}$-reflectors for short.

As $\mathbb{G}$ is a group, any $\mathbb{G}$-reflector G must be invertible. G^{-1} is also a $\mathbb{G}$-reflector, as if $\mathcal{H}$ is the hyperplane of G, we have

$$\mathcal{H} = \{x \in \mathbb{K}^n : Gx = x\} = \{x \in \mathbb{K}^n : x = G^{-1}x\},$$

hence G^{-1} fixes the same hyperplane as G. The set of $\mathbb{G}$-reflectors is thus closed under inverses. However, it is not closed under products, as although a product of $\mathbb{G}$-reflectors fixing a common hyperplane is a $\mathbb{G}$-reflector, a product of $\mathbb{G}$-reflectors fixing different hyperplanes is not.

The following result gives a complete representation of $\mathbb{G}$-reflectors belonging to a general automorphism group $\mathbb{G}$ of a scalar product on $\mathbb{K}^n$ (see [13] for a proof).

Theorem 1.1 *Any $\mathbb{G}$-reflector G is expressible in the form*

$$G = \begin{cases} I + \beta u u^T M & \text{if } \langle \cdot, \cdot \rangle_{\mathrm{M}} \text{ is bilinear,} \\ I + \beta u u^* M & \text{if } \langle \cdot, \cdot \rangle_{\mathrm{M}} \text{ is sesquilinear,} \end{cases} \tag{1.1}$$

for some $\beta \in \mathbb{K}$ and $u \in \mathbb{K}^n$. Not every G characterized by (1.1) *is in $\mathbb{G}$; the parameters β and u must satisfy the following relation:*

$$\text{bilinear forms:} \quad G \in \mathbb{G} \;\Leftrightarrow\; (M + (1 + \beta q_{\mathrm{M}}(u))M^T)u = 0. \tag{1.2}$$

$$\text{sesquilinear forms:} \quad G \in \mathbb{G} \;\Leftrightarrow\; (\beta M + (\overline{\beta} + |\beta|^2 q_{\mathrm{M}}(u))M^*)u = 0. \tag{1.3}$$

An important subset of the set of elementary transformations are *Householder reflections*. These are elementary matrices of the form

$$H(u) = I + \beta u u^*, \;\; 0 \neq u \in \mathbb{K}^n, \;\; 0 \neq \beta \in \mathbb{K}, \tag{1.4}$$

which are symmetric orthogonal if $\mathbb{K} = \mathbb{R}$ and we take $\beta = -2/(u^T u)$, and unitary if $\mathbb{K} = \mathbb{C}$ and β lies on the circle $|\beta - r| = |r|$, where $r = -1/(u^* u)$ (furthermore, $\beta \in \mathbb{R}$ on the circle yields a Hermitian $H(u)$).

Now, considering the real orthogonal group $\mathbb{G} = O(n, \mathbb{R})$, we see by solving the conditions for β and u in (1.2) that any $\mathbb{G}$-reflector can be written in the form $G = I - 2uu^T$ with $u^T u = 1$, which is precisely the symmetric Householder reflector described above. Similarly, $\mathbb{G}$-reflectors formed from an non-isotropic u in the unitary group $\mathbb{G} = U(n)$ can be written exactly in the form described above for unitary Householder reflections. These results are a consequence of the following theorem (we refer to [13] for the proof), which further characterizes $\mathbb{G}$-reflectors, assuming properties of the matrices M associated with the underlying scalar product.

Theorem 1.2 ($\mathbb{G}$-reflectors for standard scalar product classes).

(a) *Symmetric bilinear forms* ($M^T = M$, $q_{\text{M}}(u) \in \mathbb{K}$)

$G = I + \beta uu^T M \in \mathbb{G} \;\Leftrightarrow\;$ *u is non-isotropic, and* $\beta = -2/q_{\text{M}}(u)$.

(b) *Skew-symmetric bilinear forms* ($M^T = -M$, $q_{\text{M}}(u) \equiv 0$)

$G = I + \beta uu^T M \in \mathbb{G}$ *for any* $u \in \mathbb{K}^{2n}$ *and any* $\beta \in \mathbb{K}$.

(c) *Hermitian sesquilinear forms* ($M^* = M$, $q_{\text{M}}(u) \in \mathbb{R}$)

$G = I + \beta uu^* M \in \mathbb{G} \;\Leftrightarrow\;$ *u is isotropic and* $\beta \in i\mathbb{R}$, *or u is non-isotropic and* $\beta \in \mathbb{C}$ *lies on the circle* $|\beta - r| = |r|$, *where* $r = -1/q_{\text{M}}(u) \in \mathbb{R}$.

(d) *Skew-Hermitian sesquilinear forms* ($M^* = -M$, $q_{\text{M}}(u) \in i\mathbb{R}$)

$G = I + \beta uu^* M \in \mathbb{G} \;\Leftrightarrow\;$ *u is isotropic and* $\beta \in \mathbb{R}$, *or u is non-isotropic and* $\beta \in \mathbb{C}$ *lies on the circle* $|\beta - r| = |r|$, *where* $r = -1/q_{\text{M}}(u) \in i\mathbb{R}$.

Finally, for completeness, we include the following theorem (again referring to [13] for a proof) which states necessary and sufficient conditions for the existence of a $\mathbb{G}$-reflector G such that $Gx = y$.

Theorem 1.3 ($\mathbb{G}$-reflector mapping theorem).

Suppose $\mathbb{K}^n$ *has a scalar product* $\langle \cdot, \cdot \rangle_{\text{M}}$ *that is either symmetric bilinear, skew-symmetric bilinear, Hermitian sesquilinear, or skew-Hermitian sesquilinear. For distinct* $x, y \neq 0 \in \mathbb{K}^n$, *there exists a* $\mathbb{G}$*-reflector* G *such that* $Gx = y$ *if and only if* $q_M(x) = q_M(y)$ *and* $\langle y - x, x \rangle_{\text{M}} \neq 0$. *Furthermore, whenever* G *exists, it is unique and can be specified by taking* $u = y - x$ *and* $\beta = 1/\langle u, x \rangle_{\text{M}}$ *in* (1.1). *Equivalently,* G *can be specified by taking* $u = x - y$ *and* $\beta = -1/\langle u, x \rangle_{\text{M}}$ *in* (1.1).

1.1.4 Norms

A martix norm is a function $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}$ that satisfies

1. $\|A\| \geq 0$ with equality $\Leftrightarrow A = 0$.

2. $\|\alpha A\| = |\alpha|\|A\|$ for all $\alpha \in \mathbb{C}, A \in \mathbb{C}^{m\times n}$.

3. $\|A + B\| \leq \|A\| + \|B\|$ for all $A, B \in \mathbb{C}^{m\times n}$ (the triangle inequality).

In this thesis, we refer to and use two specific matrix norms: we define the Frobenius norm by

$$\|A\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2} = \left(\operatorname{trace}(A^T A)\right)^{1/2}, \tag{1.5}$$

and the 2-norm (or spectral norm) by

$$\|A\|_2 = \left(\rho(A^T A)\right)^{1/2} = \sigma_{\max}(A), \tag{1.6}$$

where the spectral radius $\rho(A) = \max\{|\lambda| : \det(A - \lambda I) = 0\}$ (i.e., the largest eigenvalue of A), and $\sigma_{\max}(A)$ denotes the largest singular value of A.

The 2-norm condition number of a matrix A is given by $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$, and satisfies $\kappa_2(A) \geq 1$ (but can be arbitrarily large).

1.1.5 Orthogonal and Unitary matrices

A matrix $Q \in \mathbb{R}^{n\times n}$ is *orthogonal* if $Q^T Q = QQ^T = I_n$, where I_n is the $n\times n$ identity matrix. Similarly, a matrix $Q \in \mathbb{C}^{n\times n}$ is *unitary* if $Q^*Q = QQ^* = I_n$. As the orthogonals are merely a subset of the unitaries, we shall often refer to both the orthogonal and unitary groups under the collective term of the unitaries, in cases where the context applies to both groups.

Unitary matrices are perfectly conditioned, i.e., $\kappa_2(Q) = 1$, for Q unitary. Also, the 2-norm and the Frobenius norm are both *unitarily invariant* norms, that is, for unitary $U, V \in \mathbb{C}^{n\times n}$, we have

$$\|UAV\|_2 = \|A\|_2, \qquad \|UAV\|_F = \|A\|_F.$$

1.1.6 The Haar distribution

The Haar distribution is a natural distribution associated with the space of orthogonal matrices $O(n, \mathbb{R})$, defined with respect to a measure called the Haar measure. On $\mathbb{R}$, the Haar measure is the same as the Lebesgue measure, which has the key property of "translation-invariance": the measure of any open set U is equal to the measure of any 'shifted' copy of U. The Lebesgue measure is thus in a sense a *uniform* measure, as it preserves the area of all mapped sets, and is associated with the statistical notion of the uniform distribution. The Haar measure replaces this notion of translation in $\mathbb{R}$ with group multiplication, resulting in the Haar distribution being in a sense a uniform distribution on groups. The following theorem illustrates what it means for an orthogonal matrix Q to belong to the Haar distribution (we refer to [20] for further background theory and a proof of the theorem).

Theorem 1.4 *If $X \in \mathbb{R}^{n\times n}$, with elements $\sim N(0,\sigma^2)$, for any variance σ^2, and $X = QR$ is the QR factorization of X, normalized so that the diagonal elements of R are positive, then Q is an orthogonal matrix from the Haar distribution.*

The computation of Q in this manner costs $2n^3$ flops. Another method of generating orthogonal matrices from the Haar distribution, which is often favoured as it can be implemented to require less flops, is based on the following theorem stated and proven by Stewart [20].

Theorem 1.5 (Stewart) *Let the independent vectors $x_i \sim N(0,1)$ over $\mathbb{R}^{n-i+1}$, for $i = 1\colon n-1$. Let $P_i = \operatorname{diag}(I_{n-i}, \overline{P}_i)$, where $\overline{P}_i$ is the Householder transformation that reduces x_i to $r_{ii}e_1$. Then the product $Q = DP_1P_2\dots P_{n-1}$ is a random orthogonal matrix from the Haar distribution, where $D = \operatorname{diag}(\operatorname{sign}(r_{ii}))$.*

This theorem allows us to form an orthogonal Q from the Haar distribution using an appropriate product of random Householder matrices, multiplied by a diagonal signature matrix. If Q is formed explicitly, the computational cost of the method is $O(n^3)$ flops. However, the explicit formation is not usually necessary – the majority of algorithms for multiplication by an orthogonal matrix simply apply the Householder matrices and signature matrix to the original matrix. As a result, this method is more efficient, saving on storage and forming the matrix at the cheaper cost of $O(n^2)$ flops.

The concept of the Haar measure and Haar distribution applies not only to the orthogonal group, but more generally to "any locally compact topological group" and "homogeneous spaces like the Grassmann and Stiefel manifolds" [11]. The concept can thus be extended to the unitary group $U(n)$, although a theorem for generating unitary matrices in the Haar distribution which is as specific as Theorem 1.5 has yet to be written, if it is indeed possible. The following proposition, however, put forward by Mackey [11], describes the method we will use in Section 2.2 to generate random unitary matrices from the Haar distribution. It is yet to be proven but carries a large weight of supporting evidence, which is discussed later in the section.

Proposition 1.6 *Let the independent vectors $x_i \in \mathbb{C}^{n-i+1}$ have elements with real and complex parts both from the $N(0,1)$ distribution for $i = 1\colon n-1$. Let $P_i = \mathrm{diag}(I_{n-i}, \overline{P}_i)$, where $\overline{P}_i$ is a Hermitian Householder transformation that reduces x_i to $s_i e_1$, where $|s_i| = \|x_i\|_2$. Then the product $Q = DP_1P_2 \dots P_{n-1}$ is a random unitary matrix from the Haar distribution, where $D = \mathrm{diag}(d_{ii})$, with each d_{ii}, $i = 1\colon n$ a complex number of unit length chosen randomly with respect to a uniform distribution on the unit complex circle.*

This result gives us an algorithm for forming a unitary Q from the Haar distribution using a sequence of complex unitary Hermitian Householder matrices, and an appropriately chosen complex diagonal matrix. To understand the reasoning behind the algorithm, we consider the parametrization of the group of unitary reflectors that align $0 \neq x \in \mathbb{C}$ with e_j (the unit vector in the j-direction) discussed in [13]: for each $\alpha \in \mathbb{C}$ with $|\alpha| = \sqrt{x^*x}$, the unitary reflector

$$G_\alpha = I + \frac{(x - \alpha e_j)(x - \alpha e_j)^*}{\overline{\alpha}(x_j - \alpha)}$$

has the property that $G_\alpha x = \alpha e_j$. Limiting ourselves to Hermitian reflectors forces $\alpha = \pm \operatorname{sign}(x_j)\sqrt{x^*x}$, where $\operatorname{sign}(x_j) = x_j/|x_j|$, and these choices are marked by

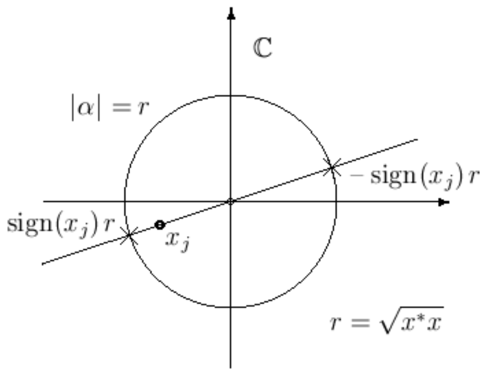

$\mathbb{C}$
$|\alpha| = r$
$-\operatorname{sign}(x_j)\, r$
$\operatorname{sign}(x_j)\, r$
x_j
$r = \sqrt{x^* x}$

Figure 1.1: Circle of α's corresponding to unitary reflector G_α with the property that $G_\alpha x = \alpha e_j$.

$\times$ in Figure 1.1 (all other α-values, $|\alpha| = r$, yield non-Hermitian reflectors). In other words, the angular direction of α depends on the angular direction of x_j, the component of x that is *not* being zeroed by the reflector. Hence, as the method outlined by Proposition 1.6 uses Hermitian reflectors that reduce a *random* vector x_i to $s_i e_1$, nothing can be specified about s_i other than its length.

Now, consider the factorization of a unitary matrix Q into a product of Hermitian reflectors, using the usual column-wise algorithm for the QR-decomposition of a general matrix. As discussed above, the best we can do at each step is to reduce the i-th column to $d_{ii}x_i$, with d_{ii} unit complex, forced by the use of Hermitian reflectors and uncontrollable apart from a $\pm$ sign. Thus the factorization into a product of Hermitians can only reduce Q to a diagonal matrix $D = \operatorname{diag}(d_{ii})$. This explains the expectancy to require the multiplication of Q by the random diagonal matrix D in Proposition 1.6 if we are to construct a random unitary matrix from Hermitian reflectors and generate from the entire group.

The above intuitive argument parallels the reasoning in [20] for the real orthogonal case, and it is clear from the above factorization argument that any unitary matrix can be constructed via this method. It needs to be proved, however, that the resultant unitaries are indeed Haar distributed. Further evidence supporting Proposition 1.6 is the fact that the routine in the LAPACK [1] test suite for generating Haar distributed random unitaries described in [4] uses this exact algorithm, with reference only to Stewart [20]. On this evidence, after consultation with Mackey, we decided that the algorithm is widely accepted enough for its use in this thesis.

1.1.7 Singular Value Decompositions (SVDs)

Each matrix $A \in \mathbb{C}^{m\times n}$ has a *singular value decomposition* (SVD), defined by

$$A = U\Sigma V^*, \quad \Sigma = \operatorname{diag}(\sigma_1, \sigma_2, \dots, \sigma_p) \in \mathbb{C}^{m\times n}, \quad p = \min(m, n),$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$, and $U \in \mathbb{C}^{m\times m}$, $V \in \mathbb{C}^{n\times n}$ are unitary. We refer to the σ_i as the *singular values* of A, and the columns of U and V as the *left* and *right singular vectors* of A respectively. For A real, U and V can be taken to be real orthogonal.

For any unitarily invariant norm, we have $\|A\| = \|\Sigma\|$, and thus, from (1.5)–(1.6), we gain

$$\|A\|_2 = \sigma_1(A), \qquad \|A\|_F = \left(\sum_{i=1}^{n} \sigma_i^2\right)^{1/2}.$$

Using the notation of $\sigma_{\min}(A)$ denoting the smallest singular value of A, $\sigma_{\max}(A)$ denoting the largest, the 2-norm condition number is given by

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_{\max}}{\sigma_{\min}} = \frac{\sigma_1}{\sigma_n}.$$

In Chapter 2, we consider structured SVDs for the real symplectic and real perplectic groups. These structures have the basic properties of the general SVD described above but with additional constraints on the singular values to limit the matrices encompassed by the decomposition to the specific group.

1.1.8 The Exchange Operator

Let $\mathbb{K}$ denote either the field $\mathbb{R}$ or $\mathbb{C}$. Consider $Q \in \mathbb{K}^{n\times n}$ and the system

$$y = \begin{array}{c c} & 1 \\ p & \left[\begin{array}{c} y_1 \\ y_2 \end{array}\right] \\ q & \end{array} = \begin{array}{c c} & \begin{array}{cc} p & q \end{array} \\ p & \left[\begin{array}{cc} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{array}\right] \\ q & \end{array} \begin{array}{c c} 1 & \\ \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] & \begin{array}{c} p \\ p \end{array} \end{array} = Qx, \tag{1.7}$$

with Q_{11} nonsingular. If we solve for x_1 in the first equation from (1.7), then substitute the solution for x_1 in the second equation, we obtain

$$\begin{bmatrix} x_1 \\ y_2 \end{bmatrix} = \operatorname{exc}(Q) \begin{bmatrix} y_1 \\ x_2 \end{bmatrix},$$

where

$$\operatorname{exc}(Q) = \begin{bmatrix} Q_{11}^{-1} & -Q_{11}^{-1}Q_{12} \\ Q_{21}Q_{11}^{-1} & Q_{22} - Q_{21}Q_{11}^{-1}Q_{12} \end{bmatrix}.$$

We refer to exc as the *exchange operator*, and use its properties in proving results in Section 2.2. It can be easily shown that

$$\operatorname{exc}(\Sigma_{p,q} Q \Sigma_{p,q}) = \Sigma_{p,q} \operatorname{exc}(Q) \Sigma_{p,q} = \operatorname{exc}(Q^*)^*. \tag{1.8}$$

A more detailed discussion of the operator can be found in [8].

1.1.9 The Least Squares Problem

In Chapter 3, we shall consider the groups from Table 1.1 which lack a structured SVD or CSD, and random matrices will be generated by taking products of $\mathbb{G}$-reflectors from the specific group. Control of the condition number will be based on empirical findings and the solution of a *least squares problem*, which is introduced here.

Definition 1.2 The *range* of $A \in \mathbb{C}^{m\times n}$ is the set of vectors that can be expressed as Ax for some x, i.e., $\operatorname{range}(A) = \{Ax : x \in \mathbb{C}^n\}$.

Consider a linear system of equations with m equations for n unknowns, with $m > n$. We wish to find a vector $x \in \mathbb{C}^n$ that satisfies $Ax = b$, with $A \in \mathbb{C}^{m\times n}$ and $b \in \mathbb{C}^m$. A suitable solution vector x exists only if b lies in range(A), and since b is an m-vector, whereas $\dim(\operatorname{range}(A)) \le n < m$, such a solution exists only for exceptional choices of b. We refer to the system of equations as being *overdetermined.*

The vector known as the *residual*,

$$r = b - Ax \in \mathbb{C}^m,$$

cannot be made to equal zero, but we can 'solve' the problem by choosing a vector norm $\|\cdot\|$ and taking our solution to be the vector x for which $\|r\|$ is as small as possible. If we choose the 2-norm, we gain the general *least squares problem*, having the following form:

$$\text{Given } A \in \mathbb{C}^{m\times n},\ m \geq n,\ b \in \mathbb{C}^m,$$
$$\text{find } x \in \mathbb{C}^n \text{ such that } \|b - Ax\|_2 \text{ is minimized.}$$

The choice of the 2-norm leads to much simpler algorithms for the solution of the least squares problem than for other norms, and its use can be justified by statistical arguments. Taking a linear problem as an example, suppose an experiment yields a large number of data points $(x_i, y_i),\ i = 1\colon m$, and there is reason to believe that the points should lie on a straight line. Supposing the data fails to lie on this straight line because of errors in the observed values y_i, with the errors are independent and distributed $\sim N(0, \sigma^2)$, then the solution of the least squares problem using the 2-norm is the maximum likelihood estimator of the true solution.

Consider the problem of fitting a data set $(x_i, y_i),\ i = 1\colon m$, with a polynomial of degree $\leq n-1$. The task is to seek a polynomial

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1}$$

such that

$$p(x_i) = y_i, \quad i = 1\colon m. \tag{1.9}$$

However, since the number of data points will typically be large and the polynomial having relatively low order, it is generally the case that $m \gg n$, and we cannot find p which satisfies (1.9) exactly. Such a polynomial is a least squares fit to the data if it minimizes the sum of the squares of the deviation from the data,

$$\sum_{i=1}^{m} |p(x_i) - y_i|^2. \tag{1.10}$$

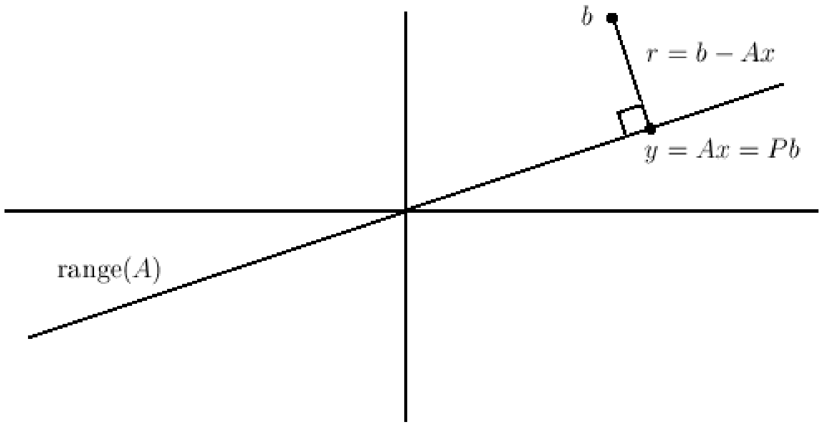


Figure 1.2: Geometric interpretation of the solution of the least squares problem, in terms of orthogonal projection.

This sum is equal to $\|r\|_2^2$ for the rectangular system $Ax = b$ given by

$$\begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ 1 & x_3 & \cdots & x_3^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_m & \cdots & x_m^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{bmatrix}. \tag{1.11}$$

The key to deriving algorithms to solve (1.11) is *orthogonal projection*, the idea of which is illustrated in Figure 1.2. We aim to find the closest point Ax in range(A) to b, thus minimizing $\|r\|_2$. Geometrically, we can see this will occur when $Ax = Pb$, where $P \in \mathbb{C}^{m \times m}$ is the orthogonal projector that maps $\mathbb{C}^m$ onto range(A). In other words, we must have r *orthogonal to* range(A), that is, $A^*r = 0$, or equivalently

$$A^*Ax = A^*b. \tag{1.12}$$

The $n \times n$ system (1.12), known as the *normal equations*, is nonsingular if and only if A has full rank, hence the solution is only *unique* if A has full rank. In this case, (1.12) is a Hermitian positive definite system of equations, and the standard method of solution is by *Cholesky factorization* (see, e.g., Golub and Van Loan [6]). This method constructs a factorization $A^*A = R^*R$, with R upper-triangular, reducing (1.12) to $R^*Rx = A^*b$. It then remains to solve the lower-triangular system $R^*w = A^*b$ for w, followed by solving the upper-triangular system $Rx = w$ for x.

Another method of solution of the least squares problem is based on reduced QR factorization. One can construct a factorization $A = \hat{Q}\hat{R}$, then the orthogonal projector P can be written $P = \hat{Q}\hat{Q}^*$ and we gain

$$y = Pb = \hat{Q}\hat{Q}^*b. \tag{1.13}$$

Now, since $y \in \text{range}(A)$, the system $Ax = y$ has an exact solution. Combining (1.13) and the QR factorization, we gain $\hat{Q}\hat{R}x = \hat{Q}\hat{Q}^*b$, which simplifies to

$$\hat{R}x = \hat{Q}^*b, \tag{1.14}$$

an upper-triangular system, nonsingular if A has full rank, which is easily solvable by back substitution.

A further method of solution of the least squares problem is based on the reduction of A to its reduced SVD $A = \hat{U}\hat{\Sigma}V^*$. Now, P can be written $P = \hat{U}\hat{U}^*$, yielding $y = Pb = \hat{U}\hat{U}^*b$. The analogue of (1.14) is then

$$\hat{\Sigma}V^*x = \hat{U}^*b, \tag{1.15}$$

which can be trivially solved from the diagonal system $\hat{\Sigma}w = \hat{U}^*b$, setting $x = Vw$.

Numerical analysts recommend the QR factorization method as the standard method for least squares solution, but each of the methods is advantageous in certain situations. The Cholesky factorization method is better when speed is the dominant consideration, but solution of the normal equations is not guaranteed to be stable in the presence of rounding errors. If A is close to rank-deficient, the QR factorization method also has stability considerations, and the SVD method is preferable, especially if $m \gg n$, in which case the cost is approximately the same as that for the QR factorization.

Chapter 2

Decompositions

We first consider the subset of the groups from Table 1.1 that have a structured SVD or CS decomposition (CSD) available, namely the pseudo-orthogonals, pseudo-unitaries, real symplectics, and real perplectics. Random matrix generation in these cases involves choosing some form of basic structured matrix, such as a diagonal matrix with constrained singular values, followed by multiplication by orthogonal or unitary matrices. We shall see that these methods yield perfectly numerically stable algorithms, and allow for precise control of the condition number.

2.1 Real Pseudo-Orthogonals: $O(p, q, \mathbb{R})$

A matrix $Q \in \mathbb{R}^{n\times n}$ is *pseudo/$\Sigma_{p,q}$-orthogonal* if $Q^T \Sigma_{p,q} Q = Q\Sigma_{p,q} Q^T = \Sigma_{p,q}$, where $\Sigma_{p,q} = \operatorname{diag}(\pm 1)$ is a signature matrix. Throughout, we take $\Sigma_{p,q}$ to have the form

$$\Sigma_{p,q} = \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix}, \quad p + q = n. \tag{2.1}$$

The group $O(1, 3, \mathbb{R})$ is known as the *Lorentz* group, and has great importance in the theory of special relativity. It is the group of time-preserving linear isometries of Minkowski space $\mathbb{R}^{(3,1)}$ of the quadratic form $dt^2 - dx_1^2 - dx_2^2 - dx_3^2$, which defines the relative space-time distance (with the convention of a system of units in which the speed of light equals one). Each element G of $O(1, 3, \mathbb{R})$ corresponds to the Lorentz

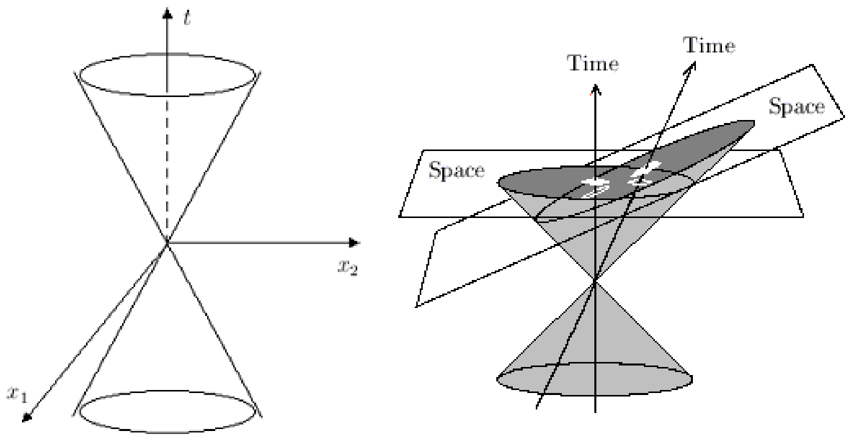


Figure 2.1: The Lorentz cone and an illustration of the Lorentz transformation.

transformation

$$\begin{pmatrix} t \\ \vec{x} \end{pmatrix} \mapsto G \begin{pmatrix} t \\ \vec{x} \end{pmatrix},$$

which maps between the space-time frames of two inertial observers (t, x_1, x_2, x_3) and (t', x_1', x_2', x_3'). G is time-preserving in the sense that the unit time vector $e_0 = (1, 0, 0, 0)$ is transformed to $Ge_0 = (t, x_1, x_2, x_3)$, with $t > 0$. The purpose of the Lorentz transformation is to relate the frame of reference of an observer (t, x_1, x_2, x_3) to the skewed perception of the observer (t', x_1', x_2', x_3'), which is moving relative to (t, x_1, x_2, x_3) at velocity ν along a different 'worldline'.

The transformation preserves the Lorentz cone of equation $t^2 - x_1^2 - x_2^2 - x_3^2 = 0$, as shown in Figure 2.1 along with a space-time diagram illustrating a Lorentz transformation between the space-time frames of two observers (marked as white dots), drawn with only 2 space dimensions for simplicity. The key point is that both observers are at the centre of the lightcone, according to the way they perceive space and time. For further description of the Lorentz group, see, for example, [16, Sec. 7.5.2].

Higham [8] describes some properties of pseudo-orthogonal matrices and further applications of the group, such as the downdating problem of computing the Cholesky factorization of a positive definite matrix $C = A^T A - B^T B$, where $A \in \mathbb{R}^{p \times n}$ $(p \geq n)$ and $B \in \mathbb{R}^{q \times n}$. He then derives an algorithm for generating random pseudo-orthogonal matrices with a user-specified condition number. His method to do so is to develop a "hyperbolic" analogue for $\Sigma_{p,q}$-orthogonal matrices of the CS decomposition for orthogonal matrices, then use this decomposition to express a $\Sigma_{p,q}$-orthogonal matrix in terms of four "half-sized" orthogonal matrices along with $2p$ scalars to determine the condition number of the matrix. The M-file `randjorth.m` (see Appendix A.1), is Higham's implementation of the algorithm in MATLAB.

The orthogonal matrices required for Higham's algorithm are generated by his MATLAB function `qmult.m` (full pathname `matlab/toolbox/matlab/elmat/private/qmult.m`). This function, when applied to matrix $A \in \mathbb{R}^{n\times n}$, generates a random orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ from the Haar distribution, and returns QA. Entering A as a scalar p, say, sets $A = I_p$, and hence returns $Q \in \mathbb{R}^{p\times p}$. The random orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ is generated by `qmult` via two possible methods:

- The default method uses Stewart's method as described in Theorem 1.5, applying $n-1$ random Householder matrices to A, followed by multiplication by the diagonal signature matrix, thus avoiding explicit formulation of Q.
- The second method forms Q explicitly via a QR factorization of a random real $n\times n$ matrix, with elements from the $N(0,1)$ distribution, taking Q as the normalized orthogonal factor, as described in Section 1.1.6.

To assess the speeds of both methods, we performed a timing experiment on `qmult`, ran on a 1400MHz AMD Athlon terminal, running MATLAB 6.5, Release 13, on Redhat Linux 6.2. For selected values of n between 10 and 1000, we used the in-built function `cputime` to time the generation of an n-by-n orthogonal matrix Q via both the Householder method and the QR method. For small values of n, we generated Q several thousand times to avoid clock resolution difficulties. We then evaluated the ratio of the Householder speed to the QR speed, the plot of which can be seen in Figure 2.2.

It was observed that the QR factorization method employed by `qmult` was faster than the Householder method for all matrix sizes examined between 10 and 1000, ranging from approximately 5 times faster for n around 10–50 to approximately 25 times faster for n around 800–1000. This result was initially surprising, as the QR

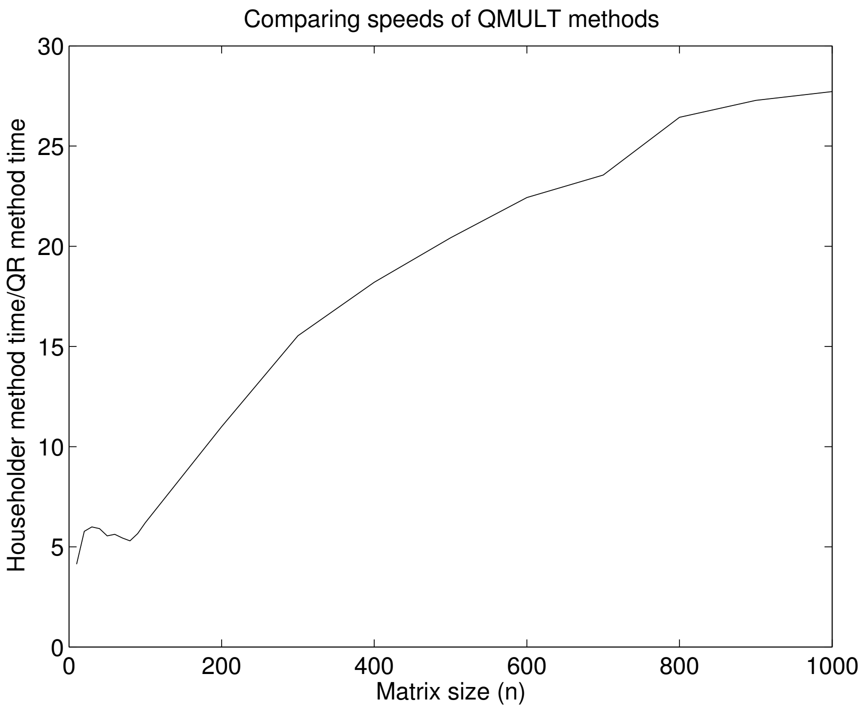


Figure 2.2: Ratio of speeds of Householder and QR method in `qmult.m`

method uses more flops (as discussed in Section 1.1.6).

The reason behind the superior performance of the QR method is the incorporation of LAPACK code into the recent MATLAB 6.5 release. The original MATLAB was built on top of LINPACK and EISPACK, the state-of-the-art Fortran subroutine libraries for matrix computation in the late 1970s. LAPACK is the modern replacement for these packages, using block algorithms which operate on several columns of a matrix at a time, offering a significant speed advantage. The speed improvement is related to the speed of the Basic Linear Algebra Subroutines, or BLAS. LINPACK used Level 1 BLAS, operating on one or two vectors of matrix columns at a time, and EISPACK didn't use any BLAS. LAPACK's algorithms make use of Level 2 and 3 BLAS, operating on larger portions of entire matrices. As a result, most floating point operations are now done in optimized BLAS and are no longer the most important factor in execution speed, with memory references and cache usage now the dominant factors.

The time taken for the QR method increases roughly linearly with n, whereas the Householder method suffers almost exponential increase for large n, say $n > 200$.

Despite the QR method performing up to 10 times faster for n between 1–200, the differences are hardly noticable at face value in this range. It is thus advisable to use the QR method for matrices of large dimension, but use whichever method is personally preferable for smaller dimensions.

The `qmult` M-file is included in Appendix A.1.

2.2 Pseudo-Unitaries: $U(p,q)$

A matrix $Q \in \mathbb{C}^{n\times n}$ is *pseudo/$\Sigma_{p,q}$-unitary* if $Q^*\Sigma_{p,q}Q = Q\Sigma_{p,q}Q^* = \Sigma_{p,q}$, where $\Sigma_{p,q} = \mathrm{diag}(\pm 1)$ is a signature matrix.

An example of an application of $\Sigma_{p,q}$-unitary matrices is the hyperbolic eigenvalue problem $Ax = \lambda Bx$, where A and B are Hermitian, A is positive definite, and B is nonsingular. Through the use of a congruence transformation, the problem can be reduced to

$$Hx = \lambda \Sigma_{p,q} x, \tag{2.2}$$

where $H \in \mathbb{C}^{n\times n}$ is a Hermitian positive definite matrix. We can always find a $\Sigma_{p,q}$-unitary matrix X such that $X^*HX = D = \mathrm{diag}(d_i)$ (for a proof of existence, see [21, Thm. VI.1.15] and [21, Cor. VI.1.19]), and hence the eigenvalues of (2.2) are given by the diagonal elements of $D\Sigma_{p,q}$.

Further applications of the pseudo-unitary group arise from the *hyperbolic singular value decomposition* (HSVD), as discussed in [18], [19]. We temporarily use the notation J for $\Sigma_{p,q}$ as in [19], to avoid confusion with the diagonal matrix Σ. Let $G \in \mathbb{C}^{m\times n}$ be of full column rank, and let $J = \mathrm{diag}(\pm 1) \in \mathbb{R}^{n\times n}$. The HSVD of the pair (G,J) is given by $G = U\Sigma X^*$, where U is unitary, $\Sigma = \mathrm{diag}(\sigma_i)$, $\sigma_i > 0$, and the *hyper-exchange matrix* X is pseudo-unitary (i.e., $X^*JX = J$). This HSVD decomposition is closely related to two eigenvalue problems: the Hermitian eigenvalue problem

$$GJG^* = U\Lambda U^*, \qquad \Lambda = \Sigma J \Sigma^*,$$

and the pseudo-Hermitian eigenvalue problem for the pair (G^*G,J), given by

$$X^*G^*GX = \Sigma^*\Sigma, \qquad X^*JX = J.$$

The HSVD is also useful in computing a numerical solution to the downdating problem: the Hermitian eigenvalue problem for a matrix $C = AA^* - BB^*$, with $A \in \mathbb{C}^{n\times p}$ $(p \geq n)$, $B \in \mathbb{C}^{n\times q}$, can be solved as the hyperbolic singular value problem for the pair (G,J), where

$$G = \begin{bmatrix} A & B \end{bmatrix}, \quad J = \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix}.$$

For the remainder of the section, we revert back to the $\Sigma_{p,q}$ notation for our signature matrix, and again take $\Sigma_{p,q}$ to have the form as defined in (2.1). The following two results are required for the main theorem of the section, which will give us the decomposition of pseudo-unitary matrices necessary for developing an algorithm for their random generation. We use the exchange operator defined in Section 1.1.8 for mapping from $\Sigma_{p,q}$-unitary matrices to unitary matrices and vice versa. The results are analogous to the results in [8] for the pseudo-orthogonal case, but are adaptated to the complex unitary case.

Lemma 2.1 *Let*

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \in \mathbb{C}^{n\times n}.$$

Then $\mathrm{exc}(Q)$ *is nonsingular if and only if* Q_{22} *is nonsingular. If* Q *is nonsingular and* $\mathrm{exc}(Q^{-1})$ *exists then* $\mathrm{exc}(Q)$ *is nonsingular and*

$$\mathrm{exc}(Q)^{-1} = \mathrm{exc}(Q^{-1}).$$

Proof. Identical to proof for real case—see Higham [8, Sec. 2]. □

We note that either of Q or $\mathrm{exc}(Q)$ can be singular with the other being nonsingular, and that a necessary and sufficient condition for both $\mathrm{exc}(Q)$ and $\mathrm{exc}(Q^{-1})$ to exist and be nonsingular is that Q, Q_{11} and Q_{22} be nonsingular.

Lemma 2.2 *Let $Q \in \mathbb{C}^{n\times n}$. If Q is $\Sigma_{p,q}$-unitary then* $\operatorname{exc}(Q)$ *is unitary. If Q is unitary and Q_{11} is nonsingular then* $\operatorname{exc}(Q)$ *is $\Sigma_{p,q}$-unitary.*

Proof. Assume first that Q is unitary with Q_{11} nonsingular. We see that $\operatorname{exc}(Q^*) = \operatorname{exc}(Q^{-1})$ exists, from the derivation of the exchange operator. By Lemma 2.1, we have that $\operatorname{exc}(Q)$ is nonsingular and further that $\operatorname{exc}(Q)^{-1} = \operatorname{exc}(Q^{-1}) = \operatorname{exc}(Q^*)$. Thus, using (1.8),

$$I = \operatorname{exc}(Q^*)\operatorname{exc}(Q) = \Sigma_{p,q}\operatorname{exc}(Q)^*\Sigma_{p,q}\cdot\operatorname{exc}(Q),$$

and hence $\operatorname{exc}(Q)$ is $\Sigma_{p,q}$-unitary.

Next, assume Q is $\Sigma_{p,q}$-unitary. This implies $Q_{11}^*Q_{11} = I + Q_{21}^*Q_{21}$, thus Q_{11} is nonsingular and $\operatorname{exc}(Q)$ exists, and we have, using (1.8),

$$\operatorname{exc}(Q)^{-1} = \operatorname{exc}(Q^{-1}) = \operatorname{exc}(\Sigma_{p,q}Q^*\Sigma_{p,q}) = \Sigma_{p,q}\operatorname{exc}(Q^*)\Sigma_{p,q} = \operatorname{exc}(Q)^*,$$

which shows that $\operatorname{exc}(Q)$ is unitary. $\square$

We now have the tools to prove the following theorem, which describes the hyperbolic analogue for $\Sigma_{p,q}$-unitary matrices of the CS decomposition for unitary matrices and is analogous to the result of Higham [8] for orthogonal matrices.

Theorem 2.3 (hyperbolic CS decomposition) *Let*

$$Q = \begin{array}{c c} & \begin{array}{cc} p & q \end{array} \\ \begin{array}{c} p \\ q \end{array} & \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \end{array} \in \mathbb{C}^{n\times n}$$

be $\Sigma_{p,q}$-unitary and assume that $q \geq p$. Then there are unitary matrices $U_1, V_1 \in \mathbb{C}^{p\times p}$ and $U_2, V_2 \in \mathbb{C}^{q\times q}$ such that

$$\begin{bmatrix} U_1^* & 0 \\ 0 & U_2^* \end{bmatrix}\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}\begin{bmatrix} V_1 & 0 \\ 0 & V_2 \end{bmatrix} = \begin{array}{cc} \begin{array}{ccc} p & p & q-p \end{array} & \\ \left[\begin{array}{c|cc} \Gamma & -\Sigma & 0 \\ \hline -\Sigma & \Gamma & 0 \\ 0 & 0 & I_{q-p} \end{array}\right] & \begin{array}{c} p \\ p \\ q-p \end{array} \end{array}, \quad (2.3)$$

where $\Gamma = \mathrm{diag}(\gamma_i) \in \mathbb{R}^{p\times p}$, $\Sigma = \mathrm{diag}(\sigma_i) \in \mathbb{R}^{p\times p}$, *and* $\Gamma^2 - \Sigma^2 = I$. *Without loss of generality we can take* $\gamma_i > \sigma_i \geq 0$ *for all* i. *Any matrix* Q *satisfying* (2.3) *is* $\Sigma_{p,q}$*-unitary.*

Proof. From Lemma 2.2 we have that $P = \mathrm{exc}(Q)$ is unitary, and further, from the proof, that its leading principle $p \times p$ submatrix is nonsingular. On partitioning P conformably with Q, the complex CS decomposition (see, e.g., Stewart and Sun [21, p. 37]) yields

$$\begin{bmatrix} V_1^* & 0 \\ 0 & U_2^* \end{bmatrix} \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \begin{bmatrix} U_1 & 0 \\ 0 & V_2 \end{bmatrix} = \begin{array}{c c} \begin{array}{ccc} p & p & q-p \end{array} & \\ \left[\begin{array}{c|cc} \widetilde{\Gamma} & \widetilde{\Sigma} & 0 \\ \hline -\widetilde{\Sigma} & \widetilde{\Gamma} & 0 \\ 0 & 0 & I_{q-p} \end{array}\right] & \begin{array}{c} p \\ p \\ q-p \end{array} \end{array}$$

for unitary $U_1, V_1 \in \mathbb{C}^{p\times p}$ and $U_2, V_2 \in \mathbb{C}^{q\times q}$, where $\widetilde{\Gamma}$ and $\widetilde{\Sigma}$ are real, nonnegative diagonal matrices, with $\widetilde{\Gamma}$ nonsingular and $\widetilde{\Gamma}^2 + \widetilde{\Sigma}^2 = I$. It is simple to show that

$$Q = \mathrm{exc}(P) = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \left[\begin{array}{c|cc} \widetilde{\Gamma}^{-1} & -\widetilde{\Gamma}^{-1}\widetilde{\Sigma} & 0 \\ \hline -\widetilde{\Sigma}\widetilde{\Gamma}^{-1} & \widetilde{\Gamma} + \widetilde{\Sigma}\widetilde{\Gamma}^{-1}\widetilde{\Sigma} & 0 \\ 0 & 0 & I_{q-p} \end{array}\right] \begin{bmatrix} V_1^* & 0 \\ 0 & V_2^* \end{bmatrix}.$$

Now, we can show by simple trigonometry that $\widetilde{\Gamma}^{-1} = \widetilde{\Gamma} + \widetilde{\Sigma}\widetilde{\Gamma}^{-1}\widetilde{\Sigma}$, as $\widetilde{\Gamma}^2 + \widetilde{\Sigma}^2 = I$ constrains that $\widetilde{\Gamma} = \mathrm{diag}(\cos\theta_i)$, $\widetilde{\Sigma} = \mathrm{diag}(\sin\theta_i)$ for $\theta_i \in [0, 2\pi)$, $i = 1{:}p$. Hence (2.3) holds with $\Gamma = \widetilde{\Gamma}^{-1} = \widetilde{\Gamma} + \widetilde{\Sigma}\widetilde{\Gamma}^{-1}\widetilde{\Sigma}$ and $\Sigma = \widetilde{\Gamma}^{-1}\widetilde{\Sigma}$, which can be easily checked to satisfy $\Gamma^2 - \Sigma^2 = I$, and $\gamma_i > \sigma_i \geq 0$ for all i. The converse is straightforward to prove, by simply solving equation (2.3) for Q, and checking $Q\Sigma_{p,q}Q^* = \Sigma_{p,q}$. □

The hyperbolic CS decomposition (2.3) defines a unitary mapping of the $\Sigma_{p,q}$-unitary matrix Q to a real symmetric matrix. Thus we see the singular values of Q are equal to the eigenvalues of the symmetric matrix on the right-hand side of (2.3), i.e.

$$\gamma_1 \pm \sigma_1, \ldots, \gamma_p \pm \sigma_p; \qquad 1, \text{ with multiplicity } q-p.$$

Moreover, the condition $\gamma_i^2 - \sigma_i^2 = 1$, for all i, dictates that the first $2p$ singular values of Q are reciprocal pairs, giving a rewriting of the singular values as

$$\gamma_i + \sigma_i \text{ and } \frac{1}{\gamma_i + \sigma_i}, \; i = 1{:}p; \qquad 1, \text{ with multiplicity } q - p. \tag{2.4}$$

It follows from the properties of singular values that the 2-norm condition number is given by

$$\kappa_2(Q) = \|Q\|_2 \|Q^{-1}\|_2 = \|Q\|_2^2 = \max_{i=1:p} \Big(\gamma_i + \sqrt{\gamma_i^2 - 1}\Big)^2. \tag{2.5}$$

We have now developed the theory required to enable us to generate a random $\Sigma_{p,q}$-unitary matrix. An algorithm for doing so is as follows:

Algorithm 1 *Let $\Sigma_{p,q}$ be defined as in* (2.1)*. The following algorithm generates a random $\Sigma_{p,q}$-unitary $Q \in \mathbb{C}^{n\times n}$ having singular values given by* (2.4)*, with the γ_i and σ_i, $i = 1{:}\min(p,q)$, chosen freely subject to the constraints $\gamma_i > \sigma_i \geq 0$ and $\gamma_i^2 - \sigma_i^2 = 1$.*

1. If $p > q$, swap p and q.

2. Generate random unitary matrices U_1, $V_1 \in \mathbb{C}^{p\times p}$ and U_2, $V_2 \in \mathbb{C}^{q\times q}$ from the Haar distribution (see Section 1.1.6).

3. Choose $\Gamma = \operatorname{diag}(\gamma_i)$ and $\Sigma = \operatorname{diag}(\sigma_i)$ according to the constraints.

4. Form $Q = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} \Gamma & -\Sigma & 0 \\ -\Sigma & \Gamma & 0 \\ 0 & 0 & I_{q-p} \end{bmatrix} \begin{bmatrix} V_1^* & 0 \\ 0 & V_2^* \end{bmatrix}$.

5. If step 1 resulted in a swap, $Q = PQP^T$, where $P = \begin{bmatrix} 0 & I_q \\ I_p & 0 \end{bmatrix}$.

After observing (2.5), we see that the condition number of Q is determined solely by its largest singular value. Thus, to generate the specified condition number, we may set γ_1 as the largest singular value by setting it as a specific value and generating all other singular values randomly, forcing them to be *less than* γ_1 i.e. $\gamma_1 > \gamma_2, \ldots, \gamma_p$. The specific value for γ_1, for $\kappa_2(Q) = c$, say, is then given by (2.5) as follows:

$$c = \Big(\gamma_1 + \sqrt{\gamma_1^2 - 1}\Big)^2 \implies \gamma_1 = \frac{1 + c}{2\sqrt{c}}.$$

The above method for selecting the singular values is the exact same method as is used by Higham in `randjorth.m`, hence the only difference between the cases is the multiplication by unitary matrices rather than orthogonals. A method of generating random unitary matrices was thus required, and my implementation was to develop `qmult_unit.m`, a unitary analogue of `qmult.m`. When applied to a matrix $A \in \mathbb{C}^{n\times n}$, `qmult_unit` generates a random unitary matrix $Q \in \mathbb{C}^{n\times n}$ from the Haar distribution, and returns QA (again, entering A as a scalar p sets $A = I_p$, and $Q \in \mathbb{C}^{p\times p}$ is returned). The random unitary matrix $Q \in \mathbb{C}^{n\times n}$ is generated by `qmult_unit` via two possible methods:

- The default method uses the extension of Stewart's method described in Proposition 1.6, applying $n-1$ complex unitary Hermitian Householder matrices to A, followed by multiplication by a diagonal matrix of uniformly-distributed complex numbers of unit length, chosen by

  $$d_{ii} = e^{2\pi i\theta}, \quad \theta \in [0, 1].$$

  The inbuilt function `rand` is used to generate θ from the uniform distribution on $[0, 1]$. The function `house` (full pathname `matlab/toolbox/matlab/elmat/private/house.m`), written by Higham, is used to generate the Householder matrices, although they are not explicitly formed: when applied to a complex vector $x \in \mathbb{C}^i$, `house` returns $v \in \mathbb{C}^i, \beta \in \mathbb{R}$ such that $H = I - \beta vv^*$ is a unitary Hermitian Householder matrix, and the β, v are applied to the relevant rows of A accordingly.

- The second method forms Q explicitly via a QR factorization of a random complex $n \times n$ matrix, with real and complex parts from the $N(0, 1)$ distribution, taking Q the unitary factor multiplied by the usual diagonal matrix of uniformly-distributed complex numbers of unit length.

We performed the exact same timing experiment as seen in Section 2.1 on `qmult_unit`, timing the generation of an n-by-n unitary matrix via both the Householder and QR methods, and evaluating the ratio of the Householder speed to the QR speed, the plot

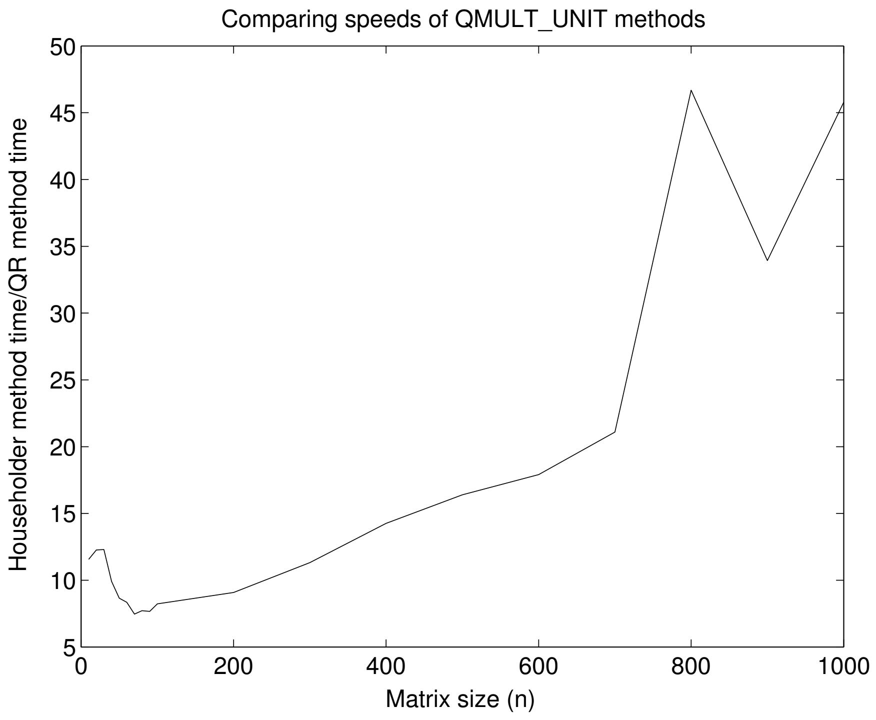


Figure 2.3: Ratio of speeds of Householder and QR method in `qmult_unit.m`

of which can be seen in Figure 2.3. Again, it was observed that the QR factorization method employed by `qmult_unit` was faster than the Householder method for all matrix sizes examined between 10 and 1000, this time ranging from approximately 5–10 times faster for n around 10–50 to approximately 40–45 times faster for n around 800–1000. The Householder method appears to be very slow for large n, possibly a result of the repeated use of the `house` function. The strange dip between $n = 10$–70 is due to the variability of the speed of the QR method, which is exceptionally fast for small n. The unexpected dip at $n = 900$ is possibly a result of "cache resonance", interactions between the memory access patterns of the algorithm and the location in cache memory of the matrices involved — if time had allowed, it would have been interesting to consider a wider range of n values between 700–1000. We again conclude that for large n, it is advisable to use the QR method, but personal preference prevails for smaller dimensions.

My implementation of Algorithm 1 in MATLAB can be seen in the M-files `rand_pseunit.m` and `qmult_unit.m` (see Appendix A.2). The function `house.m` is also included for completeness.

2.3 Real Symplectics: $Sp(2n, \mathbb{R})$

A matrix $A \in \mathbb{R}^{2n\times 2n}$ is *real symplectic* if $A^TJA = AJA^T = J$, where matrix J is the $2n\times 2n$ skew-symmetric matrix defined as

$$J = \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix}. \tag{2.6}$$

Symplectic matrices arise often in a variety of important scientific applications. For example, the eigenvalue problem for *Hamiltonian* (or *J-symmetric*) matrices (the matrices of the Lie algebra associated with $Sp(2n, \mathbb{R})$),

$$H = \begin{bmatrix} A & G \\ Q & -A^T \end{bmatrix}, \tag{2.7}$$

with $A, G, Q \in \mathbb{R}^{n\times n}$ and G, Q are symmetric, arises in many algorithms of control theory, as well as in computational physics and chemistry amongst other applications. For example, consider the *continuous-time* control problem of choosing a control function $u(t) \in \mathbb{R}^m$ to minimize

$$J(x, u) = \int_0^\infty x(t)^TQx(t) + u(t)^TRu(t)\, dt,$$

subject to

$$\dot{x} = Ax + Bu, \qquad x(0) = x_0,$$

where $Q = Q^T \in \mathbb{R}^{n\times n}$ is positive semi-definite, $R = R^T \in \mathbb{R}^{m\times m}$ is positive definite, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $x(t) \in \mathbb{R}^n$. Generally, we can find a unique optimal solution from a linear feedback law

$$u(t) = -R^{-1}B^TXx,$$

where $X \in \mathbb{R}^{n\times n}$ is the symmetric positive semi-definite solution of the *algebraic Riccati equation*

$$0 = Q + A^TX + XA - XBR^{-1}B^TX. \tag{2.8}$$

Numerical methods for solving (2.8) are often based on computing invariant subspaces of the related Hamiltonian matrices

$$H = \begin{bmatrix} A & BR^{-1}B^T \\ Q & -A^T \end{bmatrix},$$

and developing efficient numerical stable algorithms which preserve the Hamiltonian structure is still an open problem. We refer to [2], [3] and the references therein for further details.

Definition 2.1 The *real symplectic orthogonal* group, denoted $SpO(2n,\mathbb{R})$, is the intersection of the real orthogonal and real symplectic groups, that is

$$SpO(2n,\mathbb{R}) = O(2n,\mathbb{R}) \cap Sp(2n,\mathbb{R}).$$

The following theorem gives a complete description of the form of matrices in the group $SpO(2n,\mathbb{R})$. Using this result, we then describe an isomorphism to $U(n)$, the complex unitary group, which will be useful for generating symplectic orthogonals later.

Theorem 2.4

$$SpO(2n,\mathbb{R}) = \left\{ \begin{bmatrix} A & B \\ -B & A \end{bmatrix} \in O(2n,\mathbb{R}),\ A, B \in \mathbb{R}^{n\times n} \right\}.$$

Proof. Let

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \in SpO(2n,\mathbb{R}), \quad A, B, C, D \in \mathbb{R}^{n\times n}.$$

As $M \in O(2n,\mathbb{R})$, we have $M^TM = I_{2n}$, which yields equations

$$A^TA + C^TC = I_n, \quad B^TB + D^TD = I_n, \quad A^TB + C^TD = O. \tag{2.9}$$

Now, as $M \in Sp(2n,\mathbb{R})$, we have $M^TJM = J$, giving

$$D^TA - B^TC = I_n, \quad A^TC = C^TA, \quad B^TD = D^TB. \tag{2.10}$$

Further, from the fact $M^T \in Sp(2n,\mathbb{R})$, we have $MJM^T = J$ yielding

$$DA^T - CB^T = I_n, \quad AB^T = BA^T, \quad CD^T = DC^T. \tag{2.11}$$

Then, combining (2.9)–(2.11), we obtain

$$C = C(B^TB + D^TD) = (DA^T - I_n)B + DC^TD = D(A^TB + C^TD) - B = -B,$$

and similarly

$$A = A(B^TB + D^TD) = BA^TB + (I_n + BC^T)D = D + B(A^TB + C^TD) = D.$$

Hence, as required, we have

$$M = \begin{bmatrix} A & B \\ -B & A \end{bmatrix}, \quad A, B \in \mathbb{R}^{n\times n}. \qquad \square \tag{2.12}$$

Theorem 2.5 *The map $SpO(2n,\mathbb{R}) \mapsto U(n)$ between the group of $2n\times 2n$ real symplectic orthogonal matrices and the group of $n \times n$ complex unitary matrices is an isomorphism, given by the relation*

$$SpO(2n,\mathbb{R}) \ni \begin{bmatrix} A & B \\ -B & A \end{bmatrix} \Leftrightarrow A + iB \in U(n).$$

Proof. Let $M \in SpO(2n,\mathbb{R})$ as in (2.12). The conditions (2.9)–(2.11), with $D = A$, $C = -B$, boil down to the conditions

$$A^TA + B^TB = I_n, \quad A^TB = B^TA. \tag{2.13}$$

Consider the matrix $A + iB$. Using (2.13), we have

$$\begin{aligned} (A + iB)^*(A + iB) &= (A - iB)^T(A + iB) \\ &= (A^TA + B^TB) + i(A^TB - B^TA) = I_n, \end{aligned}$$

hence $A + iB \in U(n)$. Let $\theta\colon SpO(2n,\mathbb{R}) \mapsto U(n)$ be the mapping given by

$$\theta\left(\begin{bmatrix} A & B \\ -B & A \end{bmatrix}\right) = A + iB.$$

This mapping is clearly 1-1 and onto, and we have

$$\begin{aligned} \theta\left(\begin{bmatrix} A & B \\ -B & A \end{bmatrix}\begin{bmatrix} C & D \\ -D & C \end{bmatrix}\right) &= \theta\left(\begin{bmatrix} AC - BD & AD + BC \\ -BC - AD & AC - BD \end{bmatrix}\right) \\ &= AC - BD + i(AD + BC) \\ &= (A + iB)(C + iD). \\ &= \theta\left(\begin{bmatrix} A & B \\ -B & A \end{bmatrix}\right)\theta\left(\begin{bmatrix} C & D \\ -D & C \end{bmatrix}\right), \end{aligned}$$

hence θ defines an isomorphism. □

The structured SVD for real symplectic matrices is described by the following theorem.

Theorem 2.6 *For any real symplectic $A \in \mathbb{R}^{2n\times 2n}$, there exists an SVD $A = U\Sigma V^T$ such that U and V are real symplectic orthogonal, $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_{2n})$ satisfies*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq \sigma_{2n} \geq \sigma_{2n-1} \geq \cdots \geq \sigma_{n+1}, \tag{2.14}$$

and Σ is diagonal symplectic, that is,

$$\Sigma = \left[\begin{array}{c|c} D & 0 \\ \hline 0 & D^{-1} \end{array}\right], \quad D = \operatorname{diag}(\sigma_i) \in \mathbb{R}^{n\times n}.$$

Proof. See [11]. □

The above structure for Σ is achieved by taking

$$\sigma_{n+k} = \frac{1}{\sigma_k}, \quad k = 1{:}\, n. \tag{2.15}$$

The string of inequalities (2.14) is merely an ordering convention to ensure uniqueness of Σ, analogous to the convention for the general SVD of unstructured matrices to have decreasing singular values. These conventions are not essential, but can be applied without any loss of generality, as a consequence of the proof of the theorem.

We can convince ourselves of this generality by considering the effects of permutations on the diagonal entries of Σ. If we first consider the case of 2×2 diagonals, we can interchange the diagonal entries by using a similarity with the *signed permutation*

$$S_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

The crucial point is that S_2 is symplectic-orthogonal, and as a result, the transformed matrix $S_2 \Sigma S_2^T$ remains symplectic. Now, extending to the $2n \times 2n$ case, we can symplectically embed S_2 into I_{2n} (or two copies of S to work on the upper and lower halves of Σ simultaneously) to form, say, S, as described in [14, Sec. 4.5.1], enabling us to reorder the diagonal entries of Σ as desired via $S\Sigma S^T$, whilst maintaining symplecticity. Now, as $US^T, SV^T \in SpO(2n,\mathbb{R})$ (by closure under multiplication), we can permute the real symplectic matrix $A = U\Sigma V^T$ to $US^T S\Sigma S^T SV^T$ to re-order the elements of Σ, without affecting the structure of the matrix.

From Theorems 2.5 and 2.6, and (2.15), we have the necessary theory to enable us to generate a random real symplectic matrix, and an algorithm for doing so is as follows:

Algorithm 2 *The following algorithm generates a random real symplectic $A \in \mathbb{R}^{2n\times 2n}$, with singular values σ_i, $i = 1{:}n$, chosen freely according to $\sigma_1 \geq 1$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$, and σ_i, $i = n{+}1{:}2n$, fixed by* (2.15)*.*

1. Generate random unitary matrices A, $B \in \mathbb{C}^{n\times n}$ from the Haar distribution.
2. Form symplectic orthogonal matrices U, $V \in \mathbb{R}^{2n\times 2n}$ as
$$U = \begin{bmatrix} \mathrm{Re}(A) & \mathrm{Im}(A) \\ -\mathrm{Im}(A) & \mathrm{Re}(A) \end{bmatrix}, \quad V = \begin{bmatrix} \mathrm{Re}(B) & \mathrm{Im}(B) \\ -\mathrm{Im}(B) & \mathrm{Re}(B) \end{bmatrix}.$$
3. Choose $\Sigma = \mathrm{diag}(\sigma_i)$, according to the constraints.
4. Form $A = U\Sigma V^T$.

We note that the "symplectic" ordering convention given by (2.14), combined with (2.15) and the positivity of the singular values, implies that $\sigma_1, \ldots, \sigma_n$ are all ≥ 1. Thus, in our implementation of the above algorithm, we can take $\sigma_1 \geq \cdots \geq \sigma_n \geq 1$, which makes the task of controlling the condition number simple, without rendering any real symplectic matrices inaccessible to the algorithm. To generate A with the specified condition number, say $\kappa_2(A) = c$, we simply set $\sigma_1 = \sqrt{c}$, as combining the constraint that σ_1 is the largest (or equally large) singular value of A and σ_{n+1} the smallest (or equally small), and equation (2.15), we have

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_{\max}}{\sigma_{\min}} = \frac{\sigma_1}{\sigma_{n+1}} = \sigma_1^2 = c.$$

As an afterthought, due to the particular importance of the real symplectic group in applications, my initial implementation was altered to allow the option of entering an n-vector containing the first n singular values instead of a value for the desired condition number. My implementation of Algorithm 2 in MATLAB can be seen in the M-file `rand_rsymp.m` (see Appendix A.3). The function `qmult_unit.m` (see Section 2.2) is used to generate the random unitaries required for the algorithm.

2.4 Real Perplectics: $\mathcal{P}(n)$

A matrix $A \in \mathbb{R}^{n\times n}$ is *real perplectic* if $A^T R_n A = A R_n A^T = R_n$, where R_n is the $n\times n$ "reversed" identity matrix defined as

$$R_n = \begin{bmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{bmatrix} \in \mathbb{R}^{n\times n}. \tag{2.16}$$

Definition 2.2 An $n\times n$ matrix A is called *centrosymmetric* if $R_n A R_n = A$. Equivalently, $A = (a_{ij})$ is centrosymmetric if $a_{i,j} = a_{n-i+1,n-j+1}$ for $1 \le i, j \le n$.

Definition 2.3 The *real perplectic orthogonal* group, denoted $PO(n)$, is the set

$$PO(n) = \{P \in O(n,\mathbb{R}) \mid R_n P = P R_n\},$$

where $O(n,\mathbb{R})$ is the $n\times n$ real orthogonal group. It is equivalent to the set of all centrosymmetric orthogonal matrices.

Definition 2.4 The *"flip" operator* [15], which transposes a matrix across its antidiagonal, is defined by: $A^F = R_n A^T R_n$.

Real perplectic matrices play an important role in the numerical solution of doubly structured eigenproblems involving matrices that are symmetric or skew-symmetric about the anti-diagonal as well as the main diagonal. Such matrices arise in the control of mechanical and electronic vibrations, for example, involving evaluating the eigenvalues and eigenvectors of Gram matrices symmetric about both diagonals [15]. The Jordan algebra associated with $\mathcal{P}(n)$ comprises of the matrices symmetric about the anti-diagonal, known as the *persymmetric* matrices, and the associated Lie algebra consists of the matrices skew-symmetric about the anti-diagonal, known as the *perskew-symmetric* matrices. In [12], perplectic orthogonal similarity transformations are discussed as tools for the numerical solution of persymmetric and perskew-symmetric eigenproblems.

The structure of the SVD for real perplectic matrices is described in the following theorem.

Theorem 2.7 *For any real perplectic $A \in \mathbb{R}^{n\times n}$, there exists an SVD $A = U\Sigma V^T$ such that U and V are real perplectic orthogonal, Σ is perplectic diagonal*

$$\Sigma = \left[\begin{array}{c|c} D & 0 \\ \hline 0 & D^{-F} \end{array}\right] \text{ for } n \text{ even; } \Sigma = \left[\begin{array}{c|c|c} D & & \\ \hline & \pm 1 & \\ \hline & & D^{-F} \end{array}\right] \text{ for } n \text{ odd,}$$

with $D = \operatorname{diag}(\sigma_i) \in \mathbb{R}^{k\times k}$, where $k = \frac{n}{2}$ for n even, $k = \frac{n-1}{2}$ for n odd, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$, $\sigma_1 \geq 1$.

Proof. See [11]. $\square$

The above structure for Σ is achieved by taking

$$\sigma_n = \frac{1}{\sigma_1},\ \sigma_{n-1} = \frac{1}{\sigma_2},\ \ldots, \begin{cases} \sigma_{\frac{n}{2}+1} = \frac{1}{\sigma_{\frac{n}{2}}} & \text{if } n \text{ even,} \\ \sigma_{\frac{n+3}{2}} = \frac{1}{\sigma_{\frac{n-1}{2}}} & \text{if } n \text{ odd.} \end{cases} \tag{2.17}$$

It can again be shown via a permutation argument similar to that in Section 2.3 that the ordering convention given in Theorem 2.7 does not lead to loss of generality. Let P be the centrosymmetric embedding of the conventional permutation matrix $P_2 = [0\ 1; 1\ 0]$ and P_2^{-F} in I_n, as described in [14, Sec. 4.4.1]. The perplectic-orthogonality of P enables us to reorder the diagonal elements of Σ as desired via $P\Sigma P^T$, whilst maintaining perplecticity. Hence we can permute any real perplectic matrix $A = U\Sigma V^T$ to $UP^TP\Sigma P^TPV^T$ to reorder elements of Σ without affecting the overall structure.

To implement an algorithm to generate random real perplectic matrices, a method of generating random real perplectic orthogonals is thus required. The following results for real perplectic orthogonal generation split into two cases: n odd and n even.

Theorem 2.8 *Let $n = 2k+1$, and let $I_k, R_k \in \mathbb{R}^{k\times k}$ be the identity and "reversed" identity respectively. Define $X \in \mathbb{R}^{n\times n}$ blockwise by*

$$X = \frac{1}{\sqrt{2}} \begin{bmatrix} I_k & 0 & -R_k \\ 0 & \sqrt{2} & 0 \\ R_k & 0 & I_k \end{bmatrix}. \tag{2.18}$$

Let $P \in \mathbb{R}^{(k+1)\times(k+1)}$, $Q \in \mathbb{R}^{k\times k}$ be two independently generated orthogonal matrices, random with respect to Haar measure (as in Theorem 1.5), and define $B \in \mathbb{R}^{n\times n}$ by

$$B = \begin{bmatrix} P & 0 \\ 0 & Q \end{bmatrix}.$$

Then XBX^T is a random $n\times n$ perplectic orthogonal matrix, random with respect to the Haar measure on the perplectic orthogonal group $PO(n)$.

Proof. Using the fact that $R_n^2 = I$, we observe that $XX^T = I_n$, and X is thus orthogonal. The orthogonality of both X and B implies that XBX^T is orthogonal. It remains to show that XBX^T is centrosymmetric and random over the Haar distribution. Let

$$P = \begin{array}{c c} & \begin{array}{cc} k & 1 \end{array} \\ \begin{array}{c} k \\ 1 \end{array} & \begin{bmatrix} P_k & x \\ y & \alpha \end{bmatrix} \end{array},$$

and consider the explicit evaluation of XBX^T:

$$\begin{aligned} XBX^T &= \frac{1}{2} \begin{bmatrix} I_k & 0 & -R_k \\ 0 & \sqrt{2} & 0 \\ R_k & 0 & I_k \end{bmatrix} \begin{bmatrix} P & 0 \\ 0 & Q \end{bmatrix} \begin{bmatrix} I_k & 0 & R_k \\ 0 & \sqrt{2} & 0 \\ -R_k & 0 & I_k \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} P_k & x & -R_kQ \\ \sqrt{2}y & \sqrt{2}\alpha & 0 \\ R_kP_k & R_kx & Q \end{bmatrix} \begin{bmatrix} I_k & 0 & R_k \\ 0 & \sqrt{2} & 0 \\ -R_k & 0 & I_k \end{bmatrix} \end{aligned}$$

$$= \frac{1}{2}\begin{bmatrix} P_k + R_kQR_k & \sqrt{2}x & P_kR_k - R_kQ \\ \sqrt{2}y & 2\alpha & \sqrt{2}yR_k \\ R_kP_k - QR_k & \sqrt{2}\,R_kx & R_kP_kR_k + Q \end{bmatrix}. \qquad (2.19)$$

Now, for $A \in \mathbb{R}^{k\times k}$, as R_kA transposes A across the horizontal symmetry axis and AR_k transposes A vertically, we have

$$\begin{aligned} R_n(XBX^T)R_n &= \frac{1}{2}\,R_n\begin{bmatrix} P_k + R_kQR_k & \sqrt{2}x & P_kR_k - R_kQ \\ \sqrt{2}y & 2\alpha & \sqrt{2}yR_k \\ R_kP_k - QR_k & \sqrt{2}\,R_kx & R_kP_kR_k + Q \end{bmatrix}R_n \\ &= \frac{1}{2}\begin{bmatrix} R_k(R_kP_k - QR_k) & \sqrt{2}\,R_k^2x & R_k(R_kP_kR_k + Q) \\ \sqrt{2}y & 2\alpha & \sqrt{2}yR_k \\ R_k(P_k + R_kQR_k) & \sqrt{2}\,R_kx & R_k(P_kR_k - R_kQ) \end{bmatrix}R_n \\ &= \frac{1}{2}\begin{bmatrix} R_k(R_kP_kR_k + Q)R_k & \sqrt{2}\,R_k^2x & R_k(R_kP_k - QR_k)R_k \\ \sqrt{2}yR_k^2 & 2\alpha & \sqrt{2}yR_k \\ R_k(P_kR_k - R_kQ)R_k & \sqrt{2}\,R_kx & R_k(P_k + R_kQR_k)R_k \end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix} P_k + R_kQR_k & \sqrt{2}x & P_kR_k - R_kQ \\ \sqrt{2}y & 2\alpha & \sqrt{2}yR_k \\ R_kP_k - QR_k & \sqrt{2}\,R_kx & R_kP_kR_k + Q \end{bmatrix} = XBX^T, \end{aligned}$$

hence XBX^T is centrosymmetric and, by Definition 2.3, $XBX^T \in PO(n)$. Assurance that this matrix is random with respect to Haar measure can be found in [11]. □

Theorem 2.9 *Let $n = 2k$, and let $I_k, R_k \in \mathbb{R}^{k\times k}$ be the identity and "reversed" identity respectively. Let $P, Q \in \mathbb{R}^{k\times k}$ be random orthogonal matrices constructed in the usual way (via Stewart's algorithm [20]), so that P and Q are random with respect to Haar measure on the orthogonal group $O(n,\mathbb{R})$. Then the block matrix*

$$U = \frac{1}{2}\begin{bmatrix} A & B \\ R_kBR_k & R_kAR_k \end{bmatrix}, \text{ with } A = P + R_kQR_k, \quad B = PR_k - R_kQ,$$

is a real $n\times n$ perplectic orthogonal matrix, random with respect to Haar measure on the perplectic orthogonal group $PO(n)$.

Proof. Consider the matrix X in (2.18). By removing the middle row and column, we obtain a $2k \times 2k$ orthogonal matrix W, say. Defining $B = \begin{bmatrix} P & 0 \\ 0 & Q \end{bmatrix}$, we see that

$$\begin{aligned} WBW^T &= \frac{1}{2}\begin{bmatrix} I_k & -R_k \\ R_k & I_k \end{bmatrix}\begin{bmatrix} P & 0 \\ 0 & Q \end{bmatrix}\begin{bmatrix} I_k & R_k \\ -R_k & I_k \end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix} P + R_kQR_k & PR_k - R_kQ \\ R_kP - QR_k & R_kPR_k - Q \end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix} A & B \\ R_kBR_k & R_kAR_k \end{bmatrix}, \text{ with } \begin{cases} A = P + R_kQR_k \\ B = PR_k - R_kQ \end{cases}. \end{aligned}$$

By the same logic as in the proof of Theorem 2.8, we have $U = WBW^T$ orthogonal and centrosymmetric, and thus $U \in PO(n)$. Again, assurance that this matrix is random with respect to Haar measure is found in [11]. □

With Theorems 2.7–2.9, and (2.17), we have developed the necessary theory to enable us to generate a random $n\times n$ real perplectic matrix. Two algorithms, one for n even, and the other for n odd, give methods of doing so as follows:

Algorithm 3 (n even) *Let $n = 2k$. The following algorithm generates a random real perplectic $A \in \mathbb{R}^{n\times n}$, with singular values σ_i, $i = 1{:}\frac{n}{2}$, chosen freely according to $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\frac{n}{2}}$, and σ_i, $i = \frac{n}{2}+1{:}n$, chosen as in* (2.17).

1. Generate random orthogonal matrices $P_1, Q_1, P_2, Q_2 \in \mathbb{R}^{k\times k}$ from the Haar distribution.

2. With $R_k \in \mathbb{R}^{k\times k}$ as in (2.16), form matrices $X_1, Y_1, X_2, Y_2 \in \mathbb{R}^{k\times k}$ as

$$X_1 = P_1 + R_kQ_1R_k, \quad Y_1 = P_1R_k - R_kQ_1,$$
$$X_2 = P_2 + R_kQ_2R_k, \quad Y_2 = P_2R_k - R_kQ_2.$$

3. Form perplectic orthogonal matrices U, $V \in \mathbb{R}^{n\times n}$ as

$$U = \frac{1}{2}\begin{bmatrix} X_1 & Y_1 \\ R_kY_1R_k & R_kX_1R_k \end{bmatrix}, \quad V = \frac{1}{2}\begin{bmatrix} X_2 & Y_2 \\ R_kY_2R_k & R_kX_2R_k \end{bmatrix}.$$

4. Choose $\Sigma = \text{diag}(\sigma_i)$, according to the constraints.

5. Form $A = U\Sigma V^T$.

Algorithm 4 (n odd) *Let $n = 2k+1$. The following algorithm generates a random real perplectic $A \in \mathbb{R}^{n\times n}$, with singular values σ_i, $i = 1{:}\frac{n-1}{2}$, chosen freely according to $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\frac{n-1}{2}}$, σ_i, $i = \frac{n+3}{2}{:}n$, chosen as in (2.17), and $\sigma_{\frac{n+1}{2}} = \pm 1$, chosen at random.*

1. Form $R_k \in \mathbb{R}^{k\times k}$ as in (2.16), and $X \in \mathbb{R}^{n\times n}$ as in (2.18).

2. Generate random orthogonal matrices $P_1, P_2 \in \mathbb{R}^{(k+1)\times(k+1)}$, $Q_1, Q_2 \in \mathbb{R}^{k\times k}$ from the Haar distribution.

3. Form $B_1, B_2 \in \mathbb{R}^{n\times n}$ as
$$B_1 = \begin{bmatrix} P_1 & 0 \\ 0 & Q_1 \end{bmatrix}, \; B_2 = \begin{bmatrix} P_2 & 0 \\ 0 & Q_2 \end{bmatrix}.$$

4. Form $U, V \in \mathbb{R}^{n\times n}$ as $U = XB_1X^T$, $V = XB_2X^T$.

5. Choose $\Sigma = \text{diag}(\sigma_i)$, according to the constraints.

6. Form $A = U\Sigma V^T$.

In both cases, to generate to the specified condition number, we use the same idea as in the sympletic case (see Section 2.3). Setting $\sigma_1 = \sqrt{p}$, with the constraints $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ and (2.17), we gain the required value $\kappa_2(A) = p$.

For both cases, we used MATLAB matrix re-ordering commands to imitate the the effect of multiplying by R_k. For example, as we know that R_kP transposes P across horizontally and PR_k transposes P vertically, we used commands such as `P(:,k:-1:1)` rather than `P*R`, and `P(k:-1:1,k:-1:1)` rather than `R*P*R`. For the odd case, we used the structure for XBX^T derived in (2.19). As a result, we formed U in both cases with all matrix multiplications replaced by simple re-ordering of vectors and matrix rows and columns, cutting down on memory usage and flop count to produce two fast, efficient routines.

My implementation can be seen in the M-file `rand_perp.m` (see Appendix A.4). The function `qmult.m` (see Section 2.1) is used to generate the random orthogonals required for both algorithms.

2.5 Numerical Stability of Algorithms

The algorithms for random matrix generation described thus far follow a similar pattern: a basic structured matrix Σ is chosen according to certain constraints on the singular values, followed by multiplication by orthogonal or unitary matrices U, V to form the "filled in" product $A = U\Sigma V^T$. Each algorithm generates from the entire group, attaining the desired condition number through straightforward constraining of the values of the elements of Σ.

The essential property which allowed this relative ease of condition number selection is the unitary invariance of the 2-norm, as described in Section 1.1.5. This property ensures that $\kappa_2(A) = \kappa_2(U\Sigma V^*) = \kappa_2(\Sigma)$. Another important implication of this property is for error analysis, as it means that multiplication by unitary matrices does not magnify errors in the initial matrix. Consider $A \in \mathbb{C}^{n\times n}$, contaminated by errors E, and $Q \in \mathbb{C}^{n\times n}$ unitary then

$$Q(A+E)Q^* = QAQ^* + F,$$

with $\|F\|_2 = \|QEQ^*\|_2 = \|E\|_2$ (this is analogous for the Frobenius norm). In contrast, if we take a general, nonsingular similarity transformation for general $X \in \mathbb{C}^{n\times n}$, then

$$X(A+E)X^{-1} = XAX^{-1} + G,$$

and $\|G\|_2 = \|XEX^{-1}\|_2 \le \kappa_2(X)\|E\|_2$, which can be arbitrarily large.

To analyze the numerical stability of the algorithms developed in this chapter, we present a number of results from Higham [7], to which we also refer for the proofs. We make use of the notation

$$\gamma_k = \frac{ku}{1-ku}, \qquad \tilde{\gamma}_k = \frac{cku}{1-cku},$$

where c denotes a small integer constant and u is the unit roundoff. Firstly, we consider square matrix multiplication in $\mathbb{R}$: $C = AB$, where $A, B \in \mathbb{R}^{n\times n}$. We have the following normwise bound

$$\|C - \widehat{C}\|_F \leq \gamma_n \|A\|_F \|B\|_F, \tag{2.20}$$

where C is the exact result of the computation and $\widehat{C}$ is the computed matrix. We also have the following matrix norm inequality

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\|A\|_2. \tag{2.21}$$

Applying (2.20)–(2.21) to our scenario, forming $A = U\Sigma V^T$, we have

$$\begin{aligned} \|A - \widehat{A}\|_2 \ \leq \ \gamma_n \|U\|_F \|\Sigma\|_F \|V^T\|_F \ &\leq \ n^{\frac{3}{2}} \gamma_n \|U\|_2 \|\Sigma\|_2 \|V^T\|_2 \\ &\leq \ n^{\frac{3}{2}} \gamma_n \|\Sigma\|_2, \end{aligned} \tag{2.22}$$

by the unitary invariance of the 2-norm. In words, for the four cases considered in this chapter, the 2-norm of the difference between the computed $\widehat{A}$ and the exact result A is less than or equal to a small constant dependent on the matrix size n, multiplied by $n^{\frac{3}{2}}$, multiplied by the 2-norm of the matrix Σ. We assume this bound can be extended to $A \in \mathbb{C}$, in $U(p, q)$ specifically, as the matrix Σ is always real in our algorithms, making the only difference multiplication by unitaries rather than orthogonals. Of course, (2.22) assumes no rounding errors occurring in the generation of unitary matrices U and V. We will consider the Householder method of generating orthogonal matrices in greater detail later.

The appropriate measure of the numerical amount of structure for each group considered in this chapter is shown in Table 2.1.

We performed some tests, generating ten matrices for each combination of matrix size $n = 10, 100, 500$ for $O(p, q, \mathbb{R})$, $U(p, q)$ and $\mathcal{P}(n)$, $n = 5, 50, 250$ for $Sp(2n, \mathbb{R})$, with condition number $c = 10, 10^4, 10^7, 10^{10}$. We then calculated the relevant measure of structure from Table 2.1 in each case, and took the mean average of the ten matrices generated for each $[n, c]$ pairing. We expected to see a decrease in the accuracy of the approximation as n and c increase, as our error bound (2.22) involves a factor of $n^{\frac{3}{2}}$

$\mathbb{G}$	Structure Measure for $A \in \mathbb{G}$
$O(p,q,\mathbb{R})$	$\Vert A^T\Sigma_{p,q}A - \Sigma_{p,q}\Vert_2$
$U(p,q)$	$\Vert A^*\Sigma_{p,q}A - \Sigma_{p,q}\Vert_2$
$Sp(2n,\mathbb{R})$	$\Vert A^TJA - J\Vert_2$
$\mathcal{P}(n)$	$\Vert A^TRA - R\Vert_2$

Table 2.1: Measures of numerical structure for $O(p,q,\mathbb{R})$, $U(p,q)$, $Sp(2n,\mathbb{R})$ and $\mathcal{P}(n)$.

and the 2-norm of Σ, dependent on the requested condition number. We used the QR method for generating unitaries, and simplified by taking $p = q = \frac{n}{2}$ in the pseudo-orthogonal and pseudo-unitary cases. We believe making these choices differently will have only a small effect on the numerical measure of structure, perhaps multiplication by a small constant $k \approx 1$. The results of the tests are shown in Table 2.2. We see that all four cases behave similarly and as expected, with excellent numerical structure.

Finally, for completeness, we consider the stability of the process of multiplication by a random orthogonal matrix using the Householder method, using further results from [7].

Lemma 2.10 *Let $x \in \mathbb{R}^n$. Consider the "usual" construction of $\beta \in \mathbb{R}$ and $v \in \mathbb{R}^n$, used by* `qmult.m`*, such that $Px = \sigma e_1$, where $P = I - \beta vv^T$ is a Householder matrix with $\beta = 2/(v^Tv)$. In floating point arithmetic, the computed $\widehat{\beta}$ and $\widehat{v}$ satisfy $\widehat{v}(2{:}\,n) = v(2{:}\,n)$, $\widehat{\beta} = \beta(1+\tilde{\theta}_n)$, and $\widehat{v}_1 = v_1(1+\tilde{\theta}_n)$, where $|\tilde{\theta}_n| \le \tilde{\gamma}_n$.*

For convenience, we will write Householder matrices in the form $I - vv^T$, requiring $\Vert v\Vert_2 = \sqrt{2}$, amounting to redefining $v := \sqrt{\beta}v$ and $\beta := 1$ in Lemma 2.10. We can then write, from Lemma 2.10,

$$\widehat{v} = v + \Delta v, \quad |\Delta v| \le \tilde{\gamma}_n|v| \quad (v \in \mathbb{R}^n, \ \ \Vert v\Vert_2 = \sqrt{2}). \tag{2.23}$$

Next, we consider a sequence of Householder transformations applied to a matrix. Each Householder matrix P_j is arbitrary, and since the P_j are applied to the columns of A, columnwise error bounds are attained. We assume $r\tilde{\gamma}_n < \frac{1}{2}$, where r is the number of Householder transformations, and write the jth colums of A and ΔA as a_j and Δa_j, respectively.

$O(p,q,\mathbb{R})$	`c = cond(A)`			
n	10	10^4	10^7	10^{10}
10	4.2×10^{-15}	3.5×10^{-12}	2.5×10^{-9}	2.8×10^{-6}
100	8.7×10^{-15}	5.6×10^{-12}	5.3×10^{-9}	7.0×10^{-6}
500	2.0×10^{-14}	1.0×10^{-11}	1.1×10^{-8}	1.1×10^{-5}

$U(p,q)$	`c = cond(A)`			
n	10	10^4	10^7	10^{10}
10	4.4×10^{-15}	4.5×10^{-12}	3.3×10^{-9}	3.8×10^{-6}
100	1.3×10^{-14}	9.1×10^{-12}	9.4×10^{-9}	9.4×10^{-6}
500	4.5×10^{-14}	2.3×10^{-11}	2.9×10^{-8}	2.5×10^{-5}

$Sp(2n,\mathbb{R})$	`c = cond(A)`			
n	10	10^4	10^7	10^{10}
5	2.3×10^{-15}	1.7×10^{-12}	1.2×10^{-9}	1.3×10^{-6}
50	6.7×10^{-15}	5.1×10^{-12}	5.1×10^{-9}	4.4×10^{-6}
250	1.5×10^{-14}	1.1×10^{-11}	1.1×10^{-8}	1.2×10^{-5}

$\mathcal{P}(n)$	`c = cond(A)`			
n	10	10^4	10^7	10^{10}
10	3.0×10^{-15}	2.8×10^{-12}	2.4×10^{-9}	3.2×10^{-6}
100	9.1×10^{-15}	7.0×10^{-12}	7.6×10^{-9}	7.7×10^{-6}
500	1.6×10^{-14}	1.5×10^{-11}	1.5×10^{-8}	1.4×10^{-5}

Table 2.2: Experimental results for numerical structure for selected matrix size n and condition number c.

Lemma 2.11 *Consider the sequence of transformations*

$$A_{k+1} = P_k A_k, \qquad k = 1\colon r,$$

where $A_1 = A \in \mathbb{R}^{n\times n}$ and $P_k = I - v_k v_k^T \in \mathbb{R}^{n\times n}$ is a Householder matrix. Assume that the transformations are performed using computed Householder vectors $\widehat{v}_k \approx v_k$ that satisfy (2.23)*. The computed matrix $\widehat{A}_{r+1}$ satisfies*

$$\widehat{A}_{r+1} = Q^T(A + \Delta A), \tag{2.24}$$

where $Q^T = P_r P_{r-1} \dots P_1$ and

$$\|\Delta a_j\|_2 \le r\tilde{\gamma}_n \|a_j\|_2, \qquad j = 1\colon n. \tag{2.25}$$

The above lemma gives us a bound on the column-wise norms of $\widehat{A} \in \mathbb{R}^{n\times n}$ calculated by $Q^T A$ with $Q \in O(n,\mathbb{R})$ computed via the Householder method. The difference between each column of the computed $\widehat{A}$ and the same column of the exact result $Q^T A$ is less than or equal to a small constant dependent on n and u, multiplied by the number of Householder applications involved in generating Q and the 2-norm of the same column in the original matrix A. We assume the above derivation can be applied to the complex case for an analogous bound on the columns of $\widehat{A}$ formed via $Q^* A$ with $Q \in U(n)$ via the Householder method.

The combined results in this section show the perfect numerical stability of the algorithms developed in this chapter. For each case, the computed $\widehat{A}$ satisfies $\|A - \widehat{A}\|_2 \leq c_n u \|A\|_2$, where u is the unit roundoff, c_n is a constant dependent on n, the size of the matrix, and A is the exact result of the computation.

Chapter 3

Using $\mathbb{G}$-reflectors

We shall now consider the groups from Table 1.1 which lack a structured SVD or CSD, namely the complex orthogonals, complex pseudo-orthogonals, complex symplectics, and conjugate symplectics. Random matrix generation in these cases will involve taking a product of k random $\mathbb{G}$-reflectors, where k will be decided by the matrix size and the desired condition number. In this scenario, condition numbers are impossible to control precisely, due to the nature of the method of construction, namely products of matrices with highly variable individual condition numbers. However, for each group, we shall perform tests to seek patterns in the condition numbers of matrices of a range of sizes, formed via differing numbers of products of $\mathbb{G}$-reflectors, and attempt to "fit" a polynomial to the pattern of empirical data. We can then use the coefficients of this polynomial to estimate the number of $\mathbb{G}$-reflector products required for a specific matrix size to gain a condition number *approximately* close to the desired value.

We shall also consider the effects of rounding errors on the numerical structure of the generated matrices, and the numerical stability of the $\mathbb{G}$-reflector method.

All computations were done using MATLAB Version 6.5 Release 13 on a 1400MHz AMD Athlon terminal, running Redhat Linux 6.2, with IEEE double-precision arithmetic and machine precision $\epsilon = 2.2204 \times 10^{-16}$.

3.1 Complex Orthogonals: $O(n, \mathbb{C})$

A matrix $A \in \mathbb{C}^{n\times n}$ is *complex orthogonal* if $A^TA = AA^T = I_n$.

Complex orthogonal transformations can be used when solving complex symmetric eigenproblems of the form $Ax = \lambda Bx$, where A, B are complex symmetric, A is positive definite, and B is nonsingular, since similarities involving complex orthogonals preserves the complex symmetry of the problem. Eigenproblems of this form arise in quantum physics, in the solution of differential equations such as the Schrödinger equation, some explanation of which can be found in [22].

In $O(n, \mathbb{C})$, we have the symmetric bilinear form $\langle x, y\rangle = x^Ty \in \mathbb{C}$ and $q(x) = x^Tx \in \mathbb{C}$. By Theorem 1.2(a), with $M = I_n$, we characterize complex orthogonal $\mathbb{G}$-reflectors $G \in \mathbb{C}^{n\times n}$ by

$$G = I - 2\frac{uu^T}{q(u)} = I - 2\frac{uu^T}{u^Tu}, \tag{3.1}$$

with $u \in \mathbb{C}^n$ non-isotropic (as $q(x) = x^Tx$, this merely implies $u \neq 0$). Hence, to generate a random $\mathbb{G}$-reflector in $O(n, \mathbb{C})$, we can form a random $0 \neq u \in \mathbb{C}^n$ and construct G as in (3.1). In our implementation, we use the MATLAB function `randn` to generate u with elements with real and complex parts from the $N(0, 1)$ distribution.

After implementing the method for constructing a single $\mathbb{G}$-reflector G, the next step is to consider the problem of predicting the condition number of G. We will work towards modelling the condition number of a matrix $A = G_1G_2 \dots G_k$, a product of k $\mathbb{G}$-reflectors, for varying k and matrix size n. The aim is to allow the user to request $A \in O(n, \mathbb{C})$ with condition number c, with the algorithm deciding how many $\mathbb{G}$-reflectors to apply to generate A with the condition number $\approx c$.

To get a feel for the problem, we firstly investigate the behaviour of the condition number of a single complex orthogonal $\mathbb{G}$-reflector G. For selected values of n between 10 and 1000, we ran a routine `gref` in MATLAB which generated $G \in \mathbb{C}^{n\times n}$ via (3.1) from $u \in \mathbb{C}^n$ as described above. For each n-value, we constructed a number of $\mathbb{G}$-reflectors (ranging between 5–50, dependent on matrix size and the resultant time taken to compute) to try and counteract the variability of the condition number by viewing a range of values. We evaluated the condition number of each G and observed what appeared to be a general exponential increase as n increased. We plotted the natural logarithm of the condition number of each G against n, and observed a slight curve in the data points. As a result, we fit the data set with a quadratic polynomial. Suppose we observed m data points $[n_i, \log(c_i)],\ i = 1\colon m$, where n_i is the matrix size and c_i is the condition number of $G \in \mathbb{C}^{n_i \times n_i}$. We used the MATLAB function `lsqr` to find the least-squares solution of the rectangular system $Ax = b$ given by

$$\begin{bmatrix} 1 & n_1 & n_1^2 \\ 1 & n_2 & n_2^2 \\ \vdots & \vdots & \vdots \\ 1 & n_m & n_m^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \log \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}.$$

The function `lsqr` calculates the solution of the least-squares problem via the conjugate gradient method on the normal equations (1.12). The plot of the resultant polynomial $p(x) = a_0 + a_1 n + a_2 n^2$, evaluated for the observed values n_i, is shown in Figure 3.1. The fact we can fit the natural logarithm of a single G with such a simple curve is encouraging for the first step in developing the required model for an approximate condition number formula.

Next, we consider the behaviour of the condition number of $A = G_1 G_2 \dots G_k$, a product of k $\mathbb{G}$-reflectors, for increasing k. We ran a routine `grefnk` which constructed ten matrices $A = G_1 G_2 \dots G_k$ (to view a range of values), for each value of $k = 1\colon 40$, for a select fixed matrix size n. For each n tested, we observed an exponential increase in the condition number as k increased. For $n = 50$, we performed a similar process of fitting the data set $[k_i, \log(c_i)]$, where k_i is the number of products of $\mathbb{G}$-reflectors and c_i is the condition number of $A = G_1 G_2 \dots G_{k_i} \in \mathbb{C}^{50 \times 50}$, with a quadratic polynomial using `lsqr`. Before we fit the data, we removed any data points with $c_i > 10^{16}$, as we are then working outside the machine precision, rendering these results useless. This time, we gained a polynomial $p(x) = a_0 + a_1 k + a_2 k^2$, which is plotted along with the data points $[k_i, \log(c_i)]$ in Figure 3.2. The variation in the data points is well illustrated in Figure 3.2, emphasizing the difficulty of the problem of obtaining an accurate formula to approximate the condition number for a given n and k.

Finally, we can formulate a model for generating $A \in O(n, \mathbb{C})$ with condition

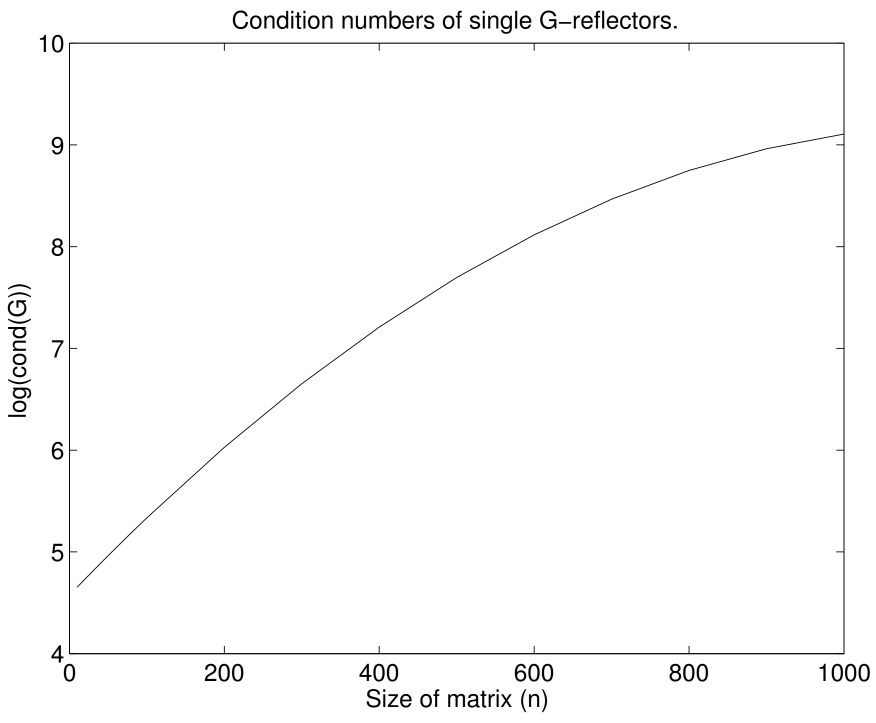


Figure 3.1: Least-squares fit of the condition number of $\mathbb{G}$-reflectors in $O(n, \mathbb{C})$, from routine `gref`.

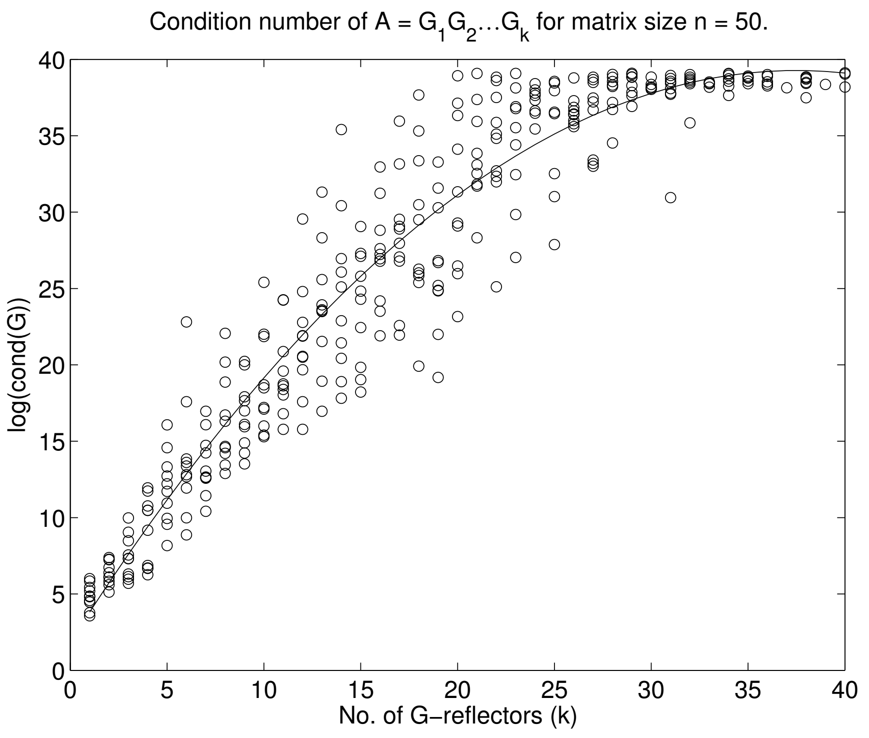


Figure 3.2: Least-squares fit of the condition number of $A = G_1G_2 \dots G_k$ in $O(50, \mathbb{C})$, from routine `grefnk`.

number approximate to the required value, combining the insight gained from the previous two experiments. We constructed $A = G_1G_2 \dots G_k \in O(n, \mathbb{C})$ for $k = 1\colon 40$, for N values of n ranging between 10 and 1000. For each n_i, k_j, we repeated this matrix generation five times and evaluated the condition number c_{ij} of each matrix to gain a large set of data. Now, suppose we observed $m = 5jN$ data points $[n_{il}, k_{jl}, c_{ijl}]$, $i = 1\colon N$, $j = 1\colon 40$, $l = 1\colon 5$. As before, we then removed any data points with $c_{ijl} > 10^{16}$ to ensure we worked with valid data. We were then left with $\widetilde{m}$ data points (with $\widetilde{m} < m$), which we will denote $[n_i, k_i, c_i]$ for ease of notation (clearly, some n_i values are the same, likewise for some k_i).

From the behaviour observed in the previous test for select values of n, it was intuitive to seek a quadratic polynomial of the form

$$p(n, k) = a_0 + a_1 n + a_2 k + a_3 k^2. \tag{3.2}$$

such that it minimizes

$$\sum_{i=1}^{\widetilde{m}} |p(n_i, k_i) - \log(c_i)|^2. \tag{3.3}$$

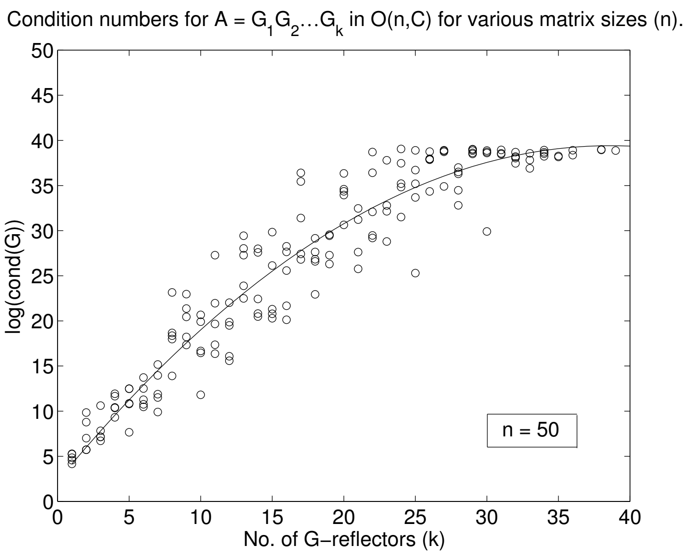
We again used `lsqr` to solve the least-squares problem (3.2)-(3.3) by minimizing the residual of the rectangular system $Mx = b$ given by

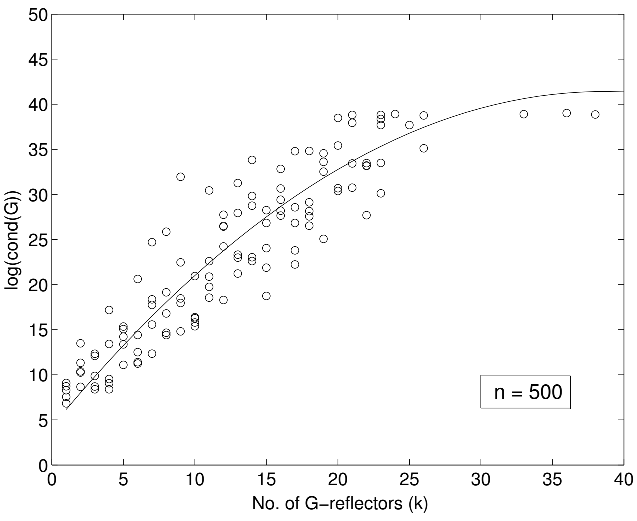
$$\begin{bmatrix} 1 & n_1 & k_1 & k_1^2 \\ 1 & n_2 & k_2 & k_2^2 \\ 1 & n_3 & k_3 & k_3^2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & n_{\widetilde{m}} & k_{\widetilde{m}} & k_{\widetilde{m}}^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \log \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{\widetilde{m}} \end{bmatrix}. \tag{3.4}$$

The solution gained by `lsqr` was

$$\begin{bmatrix} a_0, & a_1, & a_2, & a_3 \end{bmatrix} = \begin{bmatrix} 2.0344, & 0.0044, & 1.9239, & -0.0249 \end{bmatrix}. \tag{3.5}$$

Figure 3.3 shows the fit of the solution curve to the relevant data points for $n = 50$ and $n = 500$, along with an illustration of how the solution curve varies over k for a selection of n-values. The solution curve appears to be an excellent fit to the data, and can now be used to calculate the number of $\mathbb{G}$-reflectors to multiply to generate





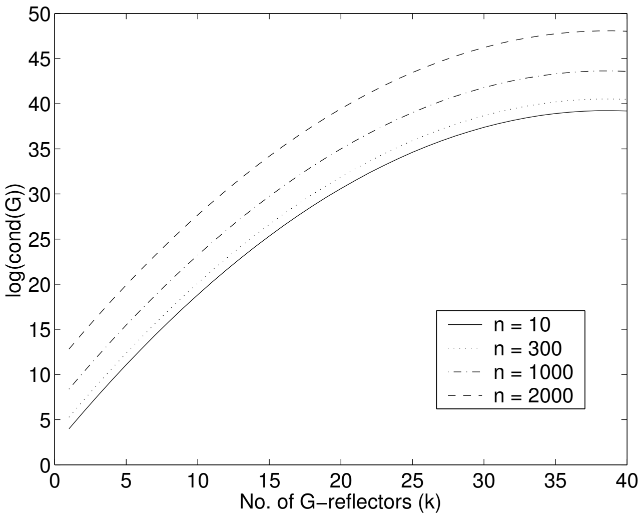


Figure 3.3: Least-squares fit of the condition number of $A = G_1 G_2 \dots G_k$ in $O(n, \mathbb{C})$, for $n = 50$, $n = 500$, and a comparison of selected n, from routine `test_corth`.

$A \in O(n, \mathbb{C})$ with condition number $\approx c$, the requested value. We have developed the empirical data required to present the following algorithm for the generation of complex orthogonals.

Algorithm 5 *The following algorithm generates a random complex orthogonal $A \in \mathbb{C}^{n\times n}$, with condition number approximately equal to c. Let $[a_0, a_1, a_2, a_3]$ be given by* (3.5)*.*

1. Solve the following quadratic for k, gaining two solutions $[k_1, k_2]$:

$$a_3k^2 + a_2k + a_1n + a_0 = \log(c).$$

2. Let r be equal to $\tilde{r}$ rounded to the nearest integer, where $\tilde{r} = \min\{k1, k2\}$.

3. Form $u_1, u_2, \ldots, u_r \in \mathbb{C}^n$ with elements with real and complex parts from the $N(0, 1)$ distribution.

4. Form $A = G_1G_2 \ldots G_r$, where

$$G_i = I - 2\frac{u_iu_i^T}{u_i^Tu_i}, \qquad i = 1{:}\,r.$$

My implementation of Algorithm 5 is case 1 in the M-file `rand_cstruct`, which can be seen in Appendix A.5.

3.2 Complex Pseudo-Orthogonals: $O(p, q, \mathbb{C})$

A matrix $A \in \mathbb{C}^{n\times n}$ is *complex pseudo/$\Sigma_{p,q}$-orthogonal* if $A^T\Sigma_{p,q}A = A\Sigma_{p,q}A^T = \Sigma_{p,q}$, where $\Sigma_{p,q} = \mathrm{diag}(\pm1)$ is a signature matrix, here taken as defined in (2.1), i.e.,

$$\Sigma_{p,q} = \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix}, \quad p + q = n.$$

In $O(p, q, \mathbb{C})$, we have the symmetric bilinear form $\langle x, y\rangle = x^T\Sigma_{p,q}y \in \mathbb{C}$ and $q(x) = x^T\Sigma_{p,q}x \in \mathbb{C}$. By Theorem 1.2(a), with $M = \Sigma_{p,q}$, we characterize complex pseudo-orthogonal $\mathbb{G}$-reflectors G by

$$G = I - 2\frac{uu^T\Sigma_{p,q}}{q(u)} = I - 2\frac{uu^T\Sigma_{p,q}}{u^T\Sigma_{p,q}u}, \tag{3.6}$$

with $u \in \mathbb{C}^n$ non-isotropic.

For $u = [u_1, u_2, \ldots, u_n]$ to be isotropic, we must have $q(u) = u^T \Sigma_{p,q} u = 0$, which is equivalent to

$$\sum_{i=1}^{p} u_i^T u_i = \sum_{i=p+1}^{n} u_i^T u_i. \tag{3.7}$$

Hence we can generate a random $\mathbb{G}$-reflector in $O(p, q, \mathbb{C})$ by forming a random $0 \neq u \in \mathbb{C}^n$ and constructing G as in (3.6), as it is virtually impossible that (3.7) would hold exactly for a random vector. Of course, if $q(u) = u^T \Sigma_{p,q} u \approx 0$, the condition number of the $\mathbb{G}$-reflector generated would be very large, but this just an example of the extreme variability involved with using random $\mathbb{G}$-reflectors. As in the complex orthogonal case, we use the MATLAB function `randn` to generate u with elements with real and complex parts from the $N(0, 1)$ distribution.

For the remaining automorphism groups, we employ almost the exact same analysis strategy as for the complex orthogonal case. We consider the problem of predicting the condition number, with the aim to allow the user to request $A \in O(p, q, \mathbb{C})$ with condition number approximately c, with the algorithm deciding how many $\mathbb{G}$-reflectors to apply.

For the ease of analysis in the complex pseudo-orthogonal case, we make the assumption that for $p^{(1)}, p^{(2)}, q^{(1)}, q^{(2)} > 0$ with $p^{(1)} + q^{(1)} = p^{(2)} + q^{(2)} = n$, the condition number for $A^{(1)} = G_1^{(1)} G_2^{(1)} \ldots G_k^{(1)} \in O(p^{(1)}, q^{(1)}, \mathbb{C})$ will behave in the same way as the condition number for $A^{(2)} = G_1^{(2)} G_2^{(2)} \ldots G_k^{(2)} \in O(p^{(2)}, q^{(2)}, \mathbb{C})$. This fact is not proven, but time requirements make it impossible to investigate the effects of varying p and q values in our analysis, and can only be mentioned as an improvement which could be made to the procedure. As a result of this assumption, we can take $q = \lfloor \frac{n}{2} \rfloor$, $p = n - q$ when formulating $\Sigma_{p,q}$ in our routine for modelling the behaviour of the condition number.

As we are now familiar with the type of problem we are modelling after the extensive analysis of $O(n, \mathbb{C})$, we skip straight to formulating a model for generating $A = G_1 G_2 \ldots G_k \in O(p, q, \mathbb{C})$. We again constructed $A = G_1 G_2 \ldots G_k$, for $k = 1{:}\,40$, this time in $O(p, q, \mathbb{C})$ using (3.6) (with $\Sigma_{p,q}$ as described above), for N values of

n ranging between 10 and 1000. We used the same procedure to the $O(n, \mathbb{C})$ case, repeating the construction of A five times and omitting data points with the condition number $> 10^{16}$. We were then left with a different $\widehat{m}$ data points ($\widehat{m} < m$), which we will again denote $[n_i, k_i, c_i]$, $i = 1{:}\,\widehat{m}$.

After inspecting some plots of k_i against $\log(c_i)$ for specific n_i values, it was evident that the condition number of A behaves almost identically to the complex orthogonal case. As a result, we chose to refit the same quadratic polynomial $p(n, k)$ from (3.2), and use `lsqr` to solve the problem by minimizing the residual of our new rectangular system $Mx = b$ of the same form as in (3.4). The solution gained by `lsqr` was

$$\begin{bmatrix} a_0, & a_1, & a_2, & a_3 \end{bmatrix} = \begin{bmatrix} 1.9510, & 0.0058, & 1.9080, & -0.0245 \end{bmatrix}. \tag{3.8}$$

Figure 3.4 shows the fit of the solution curve to the data points for $n = 100$ and $n = 500$, and an illustration of the behaviour of the solution curve as n changes. Again, the solution curve is an excellent fit to the data, allowing us to present the following algorithm for the generation of complex pseudo-orthogonals.

Algorithm 6 *The following algorithm generates a random complex pseudo-orthogonal $A \in \mathbb{C}^{n\times n}$, where $n = p + q$, with condition number approximately equal to c. Let $[a_0, a_1, a_2, a_3]$ be given by* (3.8)*.*

1. Solve the following quadratic for k, gaining two solutions $[k_1, k_2]$:

$$a_3k^2 + a_2k + a_1n + a_0 = \log(c).$$

2. Let r be equal to $\tilde{r}$ rounded to the nearest integer, where $\tilde{r} = \min\{k1, k2\}$.

3. Form $u_1, u_2, \dots, u_r \in \mathbb{C}^n$ with elements with real and complex parts from the $N(0, 1)$ distribution.

4. Form $\Sigma_{p,q} = \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix}$.

5. Form $A = G_1G_2 \dots G_r$, where

$$G_i = I - 2\frac{u_iu_i^T\Sigma_{p,q}}{u_i^T\Sigma_{p,q}u_i}, \qquad i = 1{:}\,r.$$

My implementation of Algorithm 6 is case 2 in the M-file `rand_cstruct`, which can be seen in Appendix A.5.

3.3 Complex Symplectics: $Sp(2n, \mathbb{C})$

A matrix $A \in \mathbb{C}^{2n\times 2n}$ is *complex symplectic* if $A^TJA = AJA^T = J$, where matrix J is the $2n\times 2n$ skew-symmetric matrix defined in (2.6).

Complex symplectic matrices are useful for transforming J-skew symmetric eigenproblems (that is, involving matrices from the Jordan algebra associated with $Sp(2n, \mathbb{C})$), which arise in quantum mechanical problems with time reversal symmetry (see [5] for background and further details). These problems have a simpler numerical solution than that of non-skew problems: zeros in these structures can be exploited by symplectic similarities, to fragment the problem into smaller, easier to solve problems.

Also, *near-best uniform rational approximation* (see [3] and its references) involves the computation of a symmetric SVD, which leads to a J-symmetric eigenproblem (that is, involving matrices from the Lie algebra associated with $Sp(2n, \mathbb{C})$). Other such problems arise in linear response theory and various mechanical problems.

In $Sp(2n, \mathbb{C})$, we have the skew-symmetric bilinear form $\langle x, y\rangle = x^TJy \in \mathbb{C}$ and $q(x) = x^TJx \equiv 0$. By Theorem 1.2(b), with $M = J$, we characterize complex symplectic $\mathbb{G}$-reflectors G by

$$G = I + \beta uu^TJ, \tag{3.9}$$

for any $u \in \mathbb{C}^{2n}$ and any $\beta \in \mathbb{C}$. Hence, to generate a random $\mathbb{G}$-reflector in $Sp(2n, \mathbb{C})$, we can form random $u \in \mathbb{C}^{2n}$, $\beta \in \mathbb{C}$, and construct G as in (3.9). Again, we use `randn` to generate u and β with elements with real and complex parts from the $N(0, 1)$ distribution.

It was observed from initial inspections (using routines such as `grefnk` applied to the complex symplectic case, descriptions of which are omitted for brevity) that the condition number behaves in a similar pattern as for the previous two cases, but increases much faster as k increases. It was thus only required to form $A =$

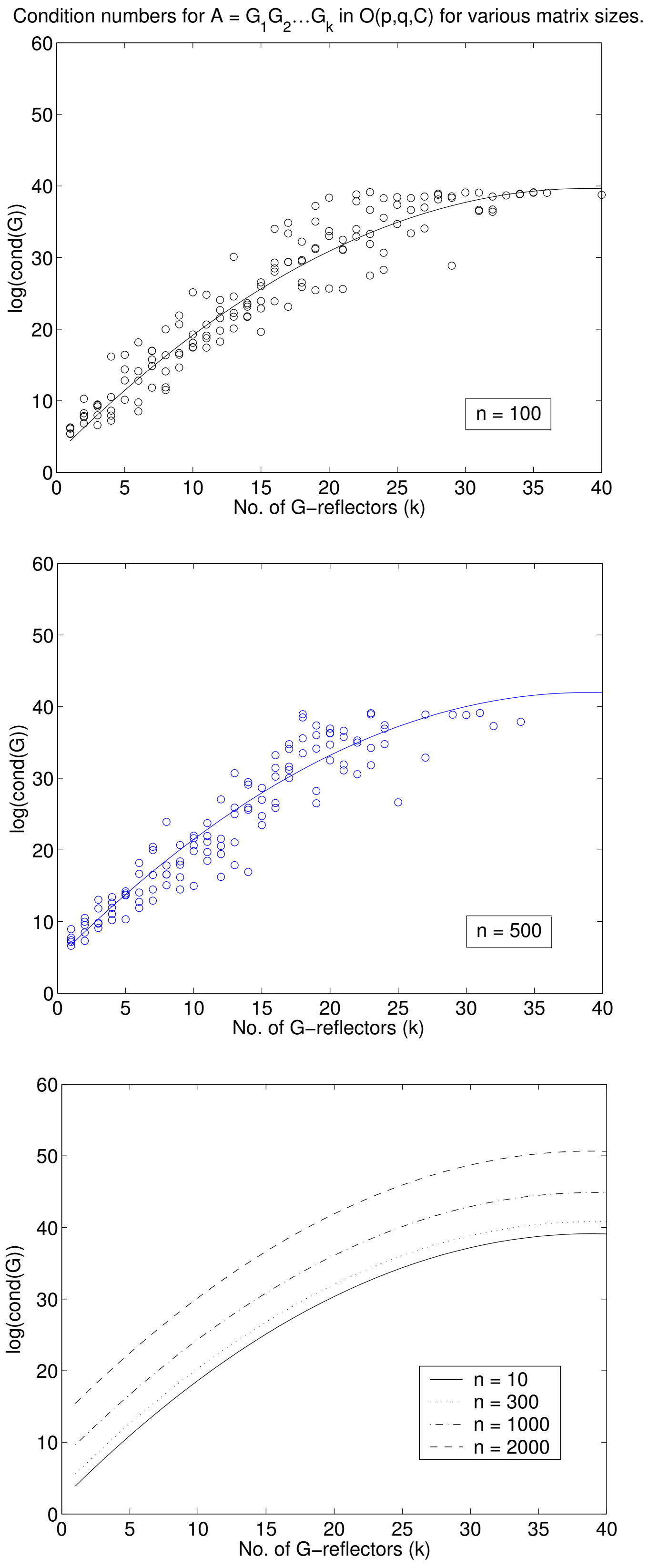


Figure 3.4: Least-squares fit of the condition number of $A = G_1G_2\ldots G_k$ in $O(p, q, \mathbb{C})$, for $n = 100$, $n = 750$, and a comparison of selected n, from routine `test_cpseorth`.

$G_1G_2\dots G_k$, for $k = 1\colon 20$, as a product of $k > 20$ generally results in the condition number being $> 10^{16}$ for any n.

We can now formulate a model for generating $A = G_1G_2\dots G_k \in Sp(2n, \mathbb{C})$. We constructed $A = G_1G_2\dots G_k$, for $k = 1\colon 20$, using (3.9), for N values of n this time ranging between 5 and 500. We observed $m = 5jN$ data points $[n_{il}, k_{jl}, c_{ijl}]$, $i = 1\colon N$, $j = 1\colon 20$, $l = 1\colon 5$, using almost the same procedure as in the last two sections. After removing data values with $c_{ijl} > 10^{16}$, we were left with a different $\widetilde{m}$ data points ($\widetilde{m} < m$), denoted $[n_i, k_i, c_i]$, $i = 1\colon \widetilde{m}$.

We chose to again refit the quadratic polynomial $p(n, k)$ from (3.2) and used `lsqr` to solve the least-squares problem by minimizing the residual of our rectangular system $Mx = b$ of the form in (3.5), with our new $\widetilde{m}$ data points. The solution gained by `lsqr` was

$$\begin{bmatrix} a_0, & a_1, & a_2, & a_3 \end{bmatrix} = \begin{bmatrix} 3.9794, & 0.0249, & 5.8397, & -0.2311 \end{bmatrix}. \tag{3.10}$$

Figure 3.5 shows the fit of the solution curve to the data points for $n = 25$ and $n = 100$, and an illustration of the behaviour of the solution curve as n changes. Again, the solution curve seems to be a good fit to the data, but perhaps further research could investigate different polynomials to seek a closer fit (see Section 4.1). The ease of applying the quadratic data fit in an algorithm for generating a complex symplectic matrix is a major advantage of using this fit rather than seeking a new fit. We present the following algorithm for the generation of complex symplectics.

Algorithm 7 *The following algorithm generates a random complex symplectic $A \in \mathbb{C}^{2n\times 2n}$, with condition number approximately equal to c. Let $[a_0, a_1, a_2, a_3]$ be given by* (3.10).

1. Solve the following quadratic for k, gaining two solutions $[k_1, k_2]$:

$$a_3k^2 + a_2k + a_1n + a_0 = \log(c).$$

2. Let r be equal to $\tilde{r}$ rounded to the nearest integer, where $\tilde{r} = \min\{k1, k2\}$.

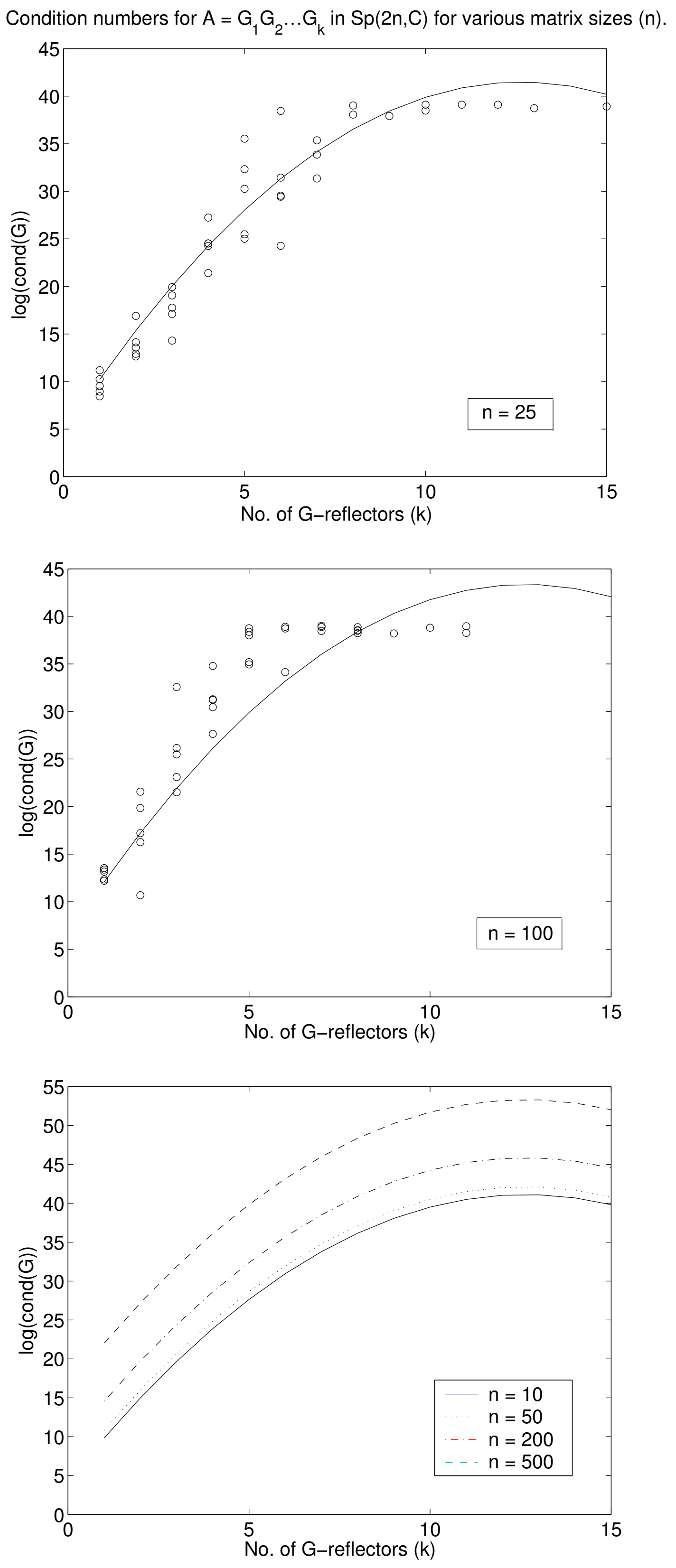


Figure 3.5: Least-squares fit of the condition number of $A = G_1 G_2 \ldots G_k$ in $Sp(2n, \mathbb{C})$, for $n = 25$, $n = 100$, and a comparison of selected n, from routine `test_csymp`.

3. Form $u_1, u_2, \ldots, u_r \in \mathbb{C}^{2n}$, $\beta \in \mathbb{C}$ with elements with real and complex parts from the $N(0,1)$ distribution.

4. Form $J = \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix}$.

5. Form $A = G_1 G_2 \ldots G_r$, where

$$G_i = I + \beta u_i u_i^T J, \qquad i = 1\colon r.$$

My implementation of Algorithm 7 is case 3 in the M-file `rand_cstruct`, which can be seen in Appendix A.5.

3.4 Conjugate Symplectics: $Sp^*(2n, \mathbb{C})$

A matrix $A \in \mathbb{C}^{2n\times 2n}$ is *conjugate symplectic* if $A^*JA = AJA^* = J$, where matrix J is the $2n\times 2n$ skew-symmetric matrix defined in (2.6).

Complex control problems arise from the solution of *discrete-time* linear-quadratic optimal control problems, analogous to those in the real continuous case described in Section 2.3. These problems lead to eigenvalue problems for J-Hermitian matrices (the matrices of the Lie algebra associated with $Sp^*(2n, \mathbb{C})$), and occur in the control of rotor systems. We refer to [3] and the references therein for further details.

In $Sp^*(2n, \mathbb{C})$, we have the skew-Hermitian sesquilinear form $\langle x, y\rangle = x^*Jy \in \mathbb{C}$ and $q(x) = x^*Jx \in i\mathbb{R}$. By Theorem 1.2(d), with $M = J$, we characterize conjugate symplectic $\mathbb{G}$-reflectors G by

$$G = I + \beta uu^*J, \tag{3.11}$$

for u isotropic and $\beta \in \mathbb{R}$, or u non-isotropic and $\beta \in \mathbb{C}$ on the circle $|\beta - r| = |r|$, where $r = -1/q(u) = -1/(u^*Ju)$. The possible β-sets are illustrated in Figure 3.6.

Now, for $u = [u_1, u_2, \ldots, u_{2n}]$ to be isotropic, we must have $q(u) = u^*Ju = 0$, which is equivalent to

$$\sum_{k=1}^{n} u_{n+k}^* u_k = \sum_{k=1}^{n} u_k^* u_{n+k},$$

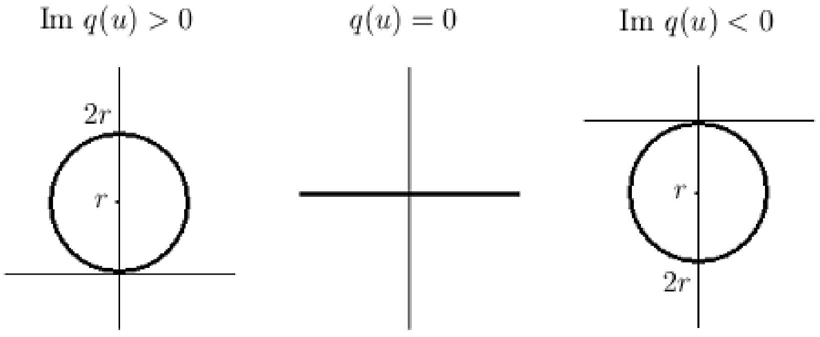


Figure 3.6: β-sets (thick lines) for a skew-Hermitian sesquilinear form, $r = \frac{-1}{q(u)}$.

which holds if and only if $u \in \mathbb{R}$ or $u \in i\mathbb{R}$. As a result, taking u isotropic produces $G \in Sp(2n, \mathbb{R})$ (clearly a subset of $Sp^*(2n, \mathbb{C})$). Taking $u \in \mathbb{C}^n$ with real and complex parts from the $N(0, 1)$ distribution will thus ensure u is non-isotropic and $G \in \mathbb{C}^{2n\times 2n}$ conjugate symplectic.

We skip straight to formulating a model for generating $A = G_1G_2 \dots G_k \in Sp^*(2n, \mathbb{C})$. As we expected the conjugate symplectic case to behave similarly to the complex symplectic case, we performed the exact same procedure as in Section 3.3, only forming $A = G_1G_2 \dots G_k$ for $k = 1\colon 20$, for N values of n ranging between 5 and 500. We amassed a different $\widehat{m}$ data points ($\widehat{m} < m$), denoted $[n_i, k_i, c_i],\ i = 1\colon \widehat{m}$.

We were surprised to see the conjugate symplectic case did not show the same behaviour as the complex symplectic case. Plots of the natural logarithm of the condition number of A for a specific n and varying k showed only a very slight curve, almost linear even. However, we decided assuming the behaviour to be linear would be an over-simplification, and chose to fit the usual quadratic polynomial $p(n, k)$ from (3.2). The least-squares solution gained by `lsqr` was

$$\begin{bmatrix} a_0, & a_1, & a_2, & a_3 \end{bmatrix} = \begin{bmatrix} 3.6339, & 0.0063, & 2.0899, & -0.0274 \end{bmatrix}. \tag{3.12}$$

Figure 3.7 shows the fit of the solution curve to the data points for $n = 25$ and $n = 100$, and an illustration of the behaviour of the solution curve as n changes. The solution curve again appears to be an excellent fit to the data, despite what appears to be an even larger variability of the condition number than in previous cases. We present the following algorithm for the generation of conjugate symplectics.

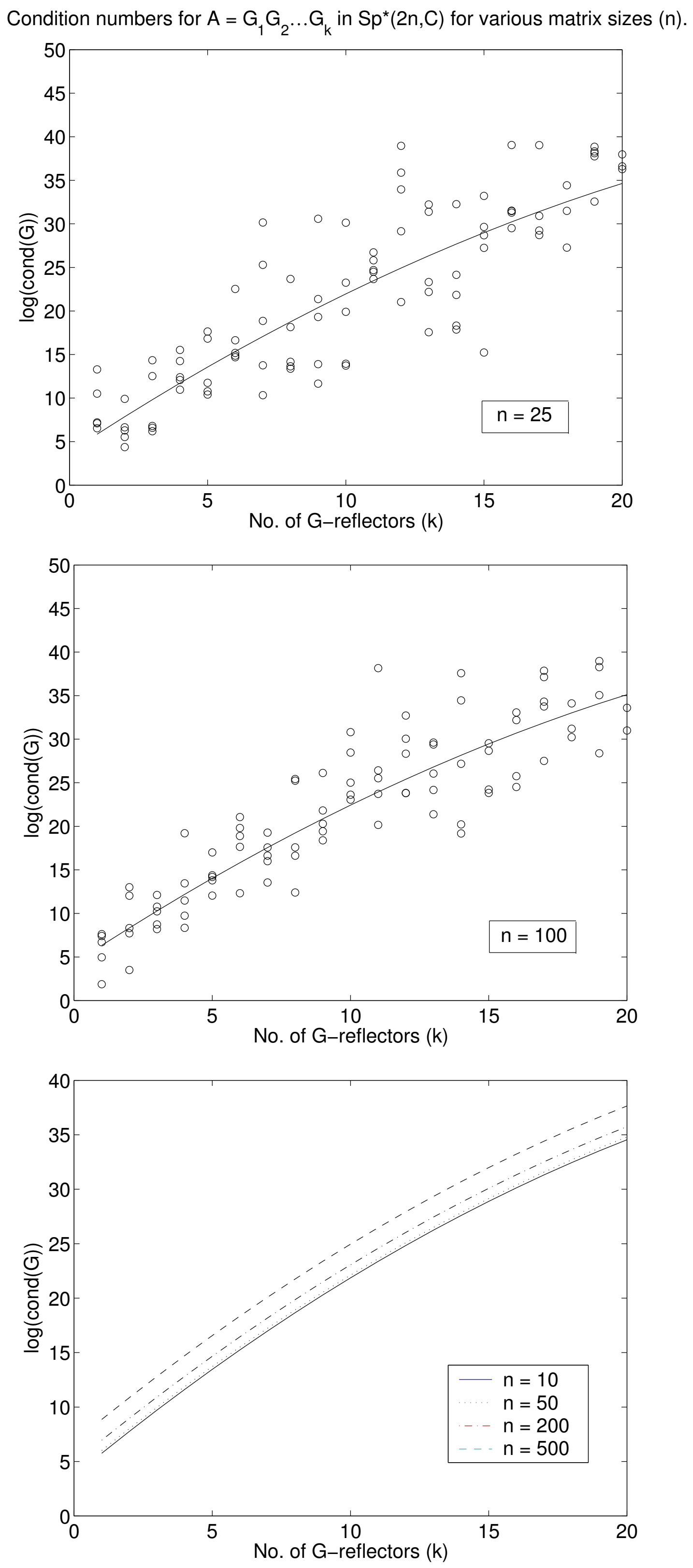


Figure 3.7: Least-squares fit of the condition number of $A = G_1G_2 \ldots G_k$ in $Sp^*(2n, \mathbb{C})$, for $n = 25$, $n = 100$, and a comparison of selected n, from routine `test_consymp`.

Algorithm 8 *The following algorithm generates a random conjugate symplectic $A \in \mathbb{C}^{2n\times 2n}$, with condition number approximately equal to c. Let $[a_0, a_1, a_2, a_3]$ be given by* (3.12).

1. Solve the following quadratic for k, gaining two solutions $[k_1, k_2]$:
$$a_3k^2 + a_2k + a_1n + a_0 = \log(c).$$
2. Let r be equal to $\tilde{r}$ rounded to the nearest integer, where $\tilde{r} = \min\{k1, k2\}$.
3. Form $u_1, u_2, \ldots, u_r \in \mathbb{C}^{2n}$, with elements with real and complex parts from the $N(0, 1)$ distribution.
4. Form $J = \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix}$.
5. Form $\beta_i \in \mathbb{C},\ i = 1{:}\,r$, on the circles $|\beta_i - r_i| = |r_i|$, where $r_i = \frac{-1}{(u_i^* J u_i)}$.
6. Form $A = G_1G_2 \ldots G_r$, where
$$G_i = I + \beta u_i u_i^* J, \qquad i = 1{:}\,r.$$

My implementation of Algorithm 8 is case 4 in the M-file `rand_cstruct`, which can be seen in Appendix A.5.

3.5 Rounding Errors

Consider the formation of $A = G_1G_2 \ldots G_k \in \mathbb{C}^{n\times n}$ in any of the groups considered in this chapter. From (2.20), we have the normwise bound

$$\|A - \widehat{A}\|_F \le \gamma_n \|G_1\|_F \|G_2\|_F \ldots \|G_k\|_F, \tag{3.13}$$

where A is the exact result of the computation and $\widehat{A}$ is the computed matrix. Applying the matrix norm inequality (2.21), we gain

$$\|A - \widehat{A}\|_2 \ \le\ \gamma_n \|G_1\|_F \|G_2\|_F \ldots \|G_k\|_F \le n^{\frac{k}{2}} \gamma_n \|G_1\|_2 \|G_2\|_2 \ldots \|G_k\|_2. \tag{3.14}$$

$\mathbb{G}$	Structure Measure for $A \in \mathbb{G}$
$O(n,\mathbb{C})$	$\|A^TA - I\|_2$
$O(p,q,\mathbb{C})$	$\|A^T\Sigma_{p,q}A - \Sigma_{p,q}\|_2$
$Sp(2n,\mathbb{C})$	$\|A^TJA - J\|_2$
$Sp^*(2n,\mathbb{C})$	$\|A^*JA - J\|_2$

Table 3.1: Measures of numerical structure for $O(n,\mathbb{C})$, $O(p,q,\mathbb{C})$, $Sp(2n,\mathbb{C})$ and $Sp^*(2n,\mathbb{C})$.

The above error bound is the best we can hope to acheive, but the methods of generation of the G_i leads to extreme variability in $\|G_i\|_2$. With the added factor of $n^{\frac{k}{2}}$, it is clear that this bound is highly unlikely to retain the structure in each case.

The appropriate measure of the numerical amount of structure for each group considered in this chapter is shown in Table 3.1.

We performed further tests, generating ten matrices for each combination of matrix size $n = 10, 100, 500$ for $O(n,\mathbb{C})$ and $O(p,q,\mathbb{C})$, $n = 5, 50, 250$ for $Sp(2n,\mathbb{C})$ and $Sp^*(2n,\mathbb{C})$, with a number of $\mathbb{G}$-reflector applications $k = 1, 5, 10, 20$. We then calculated the relevant measure of structure from Table 3.1 in each case, and took the mean average of the ten matrices generated for each $[n,k]$ pairing. We expected to see a huge decrease in the accuracy of the approximation as n and k increase, due to our error bound (3.14) involving the factor $n^{\frac{k}{2}}$, and the 2-norm of each G_i. We again simplified by taking $p = q = \frac{n}{2}$ in the $O(p,q,\mathbb{C})$ case, believing this choice will have only a small effect on the numerical measure of structure. The results of the tests are shown in Table 3.2.

The $O(n,\mathbb{C})$ and $O(p,q,\mathbb{C})$ cases behave similarly, taking until around 20 applications of $\mathbb{G}$-reflectors in the generation of A for $n = 100, 500$ before the effect of rounding errors is such that the structure of A is ruined. The $Sp(2n,\mathbb{C})$ case is much worse, with the structure ruined after 10 applications for $n = 10$, and after a mere 5 applications for $n = 100, 500$. After 20 applications, the structure is completely unrecognisable for the desired structure for all n. In the $Sp^*(2n,\mathbb{C})$ case requires 10 $\mathbb{G}$-reflector applications before the structure breaks down for all n tested.

$O(n,\mathbb{C})$	No. of $\mathbb{G}$-reflectors applied			
n	1	5	10	20
10	8.0×10^{-15}	8.4×10^{-12}	2.2×10^{-7}	3.2×10^{-3}
100	7.9×10^{-14}	1.1×10^{-10}	2.2×10^{-7}	8.0×10^{0}
500	1.1×10^{-12}	1.8×10^{-8}	2.1×10^{-6}	1.3×10^{2}

$O(p,q,\mathbb{C})$	No. of $\mathbb{G}$-reflectors applied			
n	1	5	10	20
10	3.0×10^{-15}	2.8×10^{-11}	2.5×10^{-6}	8.9×10^{-2}
100	1.8×10^{-14}	3.7×10^{-10}	6.2×10^{-5}	1.4×10^{-1}
500	4.9×10^{-13}	3.3×10^{-10}	1.4×10^{-6}	2.6×10^{5}

$Sp(2n,\mathbb{C})$	No. of $\mathbb{G}$-reflectors applied			
n	1	5	10	20
5	1.0×10^{-14}	3.8×10^{-7}	1.8×10^{1}	4.6×10^{17}
50	1.4×10^{-12}	1.2×10^{0}	5.4×10^{14}	4.8×10^{38}
250	6.5×10^{-12}	1.3×10^{4}	1.9×10^{20}	6.7×10^{55}

$Sp^*(2n,\mathbb{C})$	No. of $\mathbb{G}$-reflectors applied			
n	1	5	10	20
5	3.6×10^{-15}	4.4×10^{-7}	1.6×10^{-1}	5.8×10^{4}
50	2.3×10^{-13}	1.8×10^{-9}	1.1×10^{0}	1.2×10^{5}
250	3.5×10^{-14}	3.1×10^{-7}	1.5×10^{2}	5.6×10^{4}

Table 3.2: Experimental results for numerical structure for selected matrix size n and number of $\mathbb{G}$-reflector applications k.

While this behaviour was expected, it was disappointing to observe. All algorithms appear to behave poorly numerically from our experiments, with the effect of accumulating rounding errors making applications of a large number of $\mathbb{G}$-reflectors, required to generate with a large condition number (i.e., in the range $c = 10^{10}$–10^{15}), a possibly worthless procedure. This loss of numerical structure is the main motivation for further work, as discussed in Section 4.1.

Chapter 4

Conclusions

We have presented a MATLAB toolbox for generating random matrices in 8 automorphim groups, associated with structured nondegenerate bilinear or sesquilinear forms on $\mathbb{R}^n$ and $\mathbb{C}^n$, that we believe will be useful in practice for generating test matrix problems for testing the quality of newly-developed structure preserving algorithms.

Each automorphism group is presented in as parallel a manner as possible, by defining the group, discussing its applications where the matrix group arises and where random generations will be useful, and developing the theory and/or empirical data required to present an algorithm generation from the group.

The algorithms developed in Chapter 2 for the groups with a structured SVD or CSD available were shown to be perfectly numerical stable, and were designed to be as efficient on speed and memory usage as possible. Chapter 3 tackled the problem of the groups which lacked such a decomposition, utilising a method of forming the required matrices as a product of $\mathbb{G}$-reflectors. This approach suffered from a number of shortcomings, in particular poor numerical structure, and is an obvious area for further research.

4.1 Further work

There were a number of limitations to my methodology in Chapter 2 which could be remedied given further time to research.

Firstly, the method for testing the $O(p, q, \mathbb{C})$ case could be improved by studying the effects of varying the p, q values rather than simplifying to just take $p = \frac{n}{2}$, $q = p$.

Secondly, there were two major problems arising in the implementations of the four $\mathbb{G}$-reflector algorithms. One problem was the inability to generate large matrices with small condition numbers. For example, it would be difficult to generate $A \in O(1000, \mathbb{C})$ with condition number 100, as the algorithm has to take at least one full $\mathbb{G}$-reflector $\in \mathbb{C}^{1000\times1000}$ which has been seen from Figure 3.1 to itself have a condition number $\approx 8\times10^3$. The second problem is the breakdown in structure caused by taking a large number of $\mathbb{G}$-reflector applications in generating A, especially for large n.

In an attempt to solve these problems, we could research different ways of generating matrices in the four groups considered in Chapter 3. Possible alternatives include using products of embedded Givens-like actions, or products of non-full $\mathbb{G}$-reflectors to "fill" A without taking full matrix products. Another possibility is to research controlling the vectors $u \in \mathbb{C}^n$ and scalars $\beta \in \mathbb{C}$ used to generate each $\mathbb{G}$-reflector G, in an attempt to reduce the variability in the values of $\|G\|_2$.

Bibliography

[1] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide.* Third Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. xxvi+407 pp. ISBN 0-89871-447-8.

[2] P. Benner. Symplectic balancing of Hamiltonian matrices. *SIAM J. Sci. Comput.*, 22(5):1885-1904, 2000.

[3] A. Bunse-Gerstner, R. Byers and V. Mehrmann. A chart of numerical methods for structured eigenvalue problems. *SIAM J. Matrix Anal. Appl*, vol. 13, pp. 419–453, 1992.

[4] James W. Demmel and A. McKenney. A Test Matrix Generation Suite. Preprint MCS-P69-0389, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA, March 1989. 16 pp. LAPACK Working Note 9.

[5] J. J. Dongarra, J. R. Gabriel, D. D. Kölling, and J. H. Wilkinson. The eigenvalue problem for Hermitian matrices with time reversal symmetry. *Linear Algebra Appl.*, 60:27–42, 1984.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, USA, 3rd edition, 1996.

[7] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Second

Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. xxx+680pp. ISBN 0-89871-521-0.

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

[9] Alston S. Householder. *The Theory of Matrices in Numerical Analysis.* Blaisdell, New York, 1964. Reprinted by Dover, New York, 1975.

[10] Nathan Jacobson. *Basic Algebra I.* W. H. Freeman and Company, San Francisco, 1974.

[11] D. Steven Mackey, Private Communication, 2003.

[12] D. Steven Mackey, Niloufer Mackey, and Daniel M. Dunlavy. Structure Preserving Algorithms for Perplectic Eigenproblems. Numerical Analysis Report No. 427, Manchester Centre for Computational Mathematics, Manchester, England, May 2003. 30 pp.

[13] D. Steven Mackey, Niloufer Mackey, and Françoise Tisseur. $\mathbb{G}$-reflectors in scalar product spaces. Numerical Analysis Report No. 420, Manchester Centre for Computational Mathematics, Manchester, England, February 2003. 24 pp. To appear in Linear Algebra Appl.

[14] D. Steven Mackey, Niloufer Mackey, and Françoise Tisseur. Structured tools for structured matrices. *Electron. J. Linear Algebra*, 10(10):106–145, 2003.

[15] Russell M. Reid. Some eigenvalue properties of persymmetric matrices. *SIAM Rev.*, 39(2):313-316, 1997.

[16] Denis Serre. *Matrices: Theory and Applications.* Graduate Texts in Mathematics. Springer-Verlag, New York, 2002. xv+216pp. ISBN 0-387-95460-0.

[17] Ronald Shaw. *Linear Algebra and Group Representations. Vol. I.* Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1982.

[18] Ivan Slapničar and Ninoslav Truhar. Relative Perturbation Theory for Hyperbolic Singular Value Decomposition. *Linear Algebra Appl.*, 309:57–72, 2000.

[19] Ivan Slapničar and Krešimir Veselić. A Bound for the Condition of a Hyperbolic Eigenvector Matrix. *Linear Algebra Appl.*, 290:247–255, 1999.

[20] G. W. Stewart. The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM J. Numer. Anal.*, 17(3):403–409, 1980.

[21] G. W. Stewart and Ji-guang Sun. *Matrix Perturbation Theory.* Academic Press, London, 1990. xv+365 pp. ISBN 0-12-670230-6.

[22] Saul Youssef. A Reformulation of Quantum Mechanics. *Mod. Phys. Lett* A6, 225–236, 1991.

Appendix A

MATLAB M-Files

A.1 Random Pseudo-Orthogonals

The following function, written by Nicholas J. Higham, generates random $\Sigma_{p,q}$-orthogonal matrices with user-specified condition number. It is intended to appear in the next release of MATLAB (R14), to be called from `gallery`.

Note: $\Sigma_{p,q} = J$ here.

```
function A = randjorth(p,q,c,symm,method)
%RANDJORTH  Random J-orthogonal matrix.
%   A = GALLERY('RANDJORTH',P,Q,C) forms a random (P+Q)-by-(P+Q) J-orthogonal
%   matrix A, where J = BLKDIAG(EYE(P),-EYE(Q)) and COND(A) = C.
%   J-orthogonality means that A'*J*A = J, and such matrices are
%   sometimes called hyperbolic.
%   If omitted, C defaults to SQRT(1/EPS).
%
%   GALLERY('RANDJORTH',N) and GALLERY('RANDJORTH',N,[],C)
%   both produce an N-by-N matrix with P = CEIL(N/2), Q = FLOOR(N/2).
%
%   The full calling sequence is
%        GALLERY('RANDJORTH',P,Q,C,SYMM,METHOD).
%   If SYMM is nonzero symmetry is enforced and a symmetric positive
%   definite matrix is produced.
%   The argument METHOD specifies how the underlying orthogonal
%   transformations are carried out.   If METHOD is nonzero
%   a call to QR is used, which is much faster than the default
%   method for large dimensions, though it uses more flops.

%   Reference:
%   N. J. Higham, J-orthogonal matrices: Properties and generation,
%   Numerical Analysis Report No. 408, Manchester Centre for
%   Computational Mathematics, Manchester, England, Sept. 2002.
%
%   Nicholas J. Higham
```

```
if nargin < 2 || isempty(q), q = floor(p/2); p = p-q; end
if nargin < 3 || isempty(c), c = sqrt(1/eps); end
if nargin < 4 || isempty(symm), symm = 0; end
if nargin < 5, method = 0; end

% This function requires q >= p, so...
if p > q
   A = randjorth(q,p,c,symm,method); % diag(eye(q),-eye(p))-orthogonal matrix.
   A = A( [q+1:p+q 1:q], :); % Permute to produce J-orthogonal matrix.
   A = A(:, [q+1:p+q 1:q]);
   return
end

if c >= 1
   c(1) = (1+c)/(2*sqrt(c));
   c(2:p) = 1 + (c(1)-1)*rand(p-1,1);
elseif p ~= 1
   error('Illegal value for C.  To specify COND set C >= 1.')
end

s = sqrt(c.^2-1);

A = blkdiag([diag(c) -diag(s); -diag(s) diag(c)], eye(q-p));

if symm
   U = blkdiag(gallery('qmult',p, method),gallery('qmult',q, method));
   A = U*A*U';
   A = (A + A')/2;       % Ensure matrix is symmetric.
   return
end

A = left_mult(A,p,q,method);  % Left multiplications by orthogonal matrices.
A = left_mult(A',p,q,method); % Right multiplications by orthogonal matrices.

function A = left_mult(A,p,q,method)
%LEFT_MULT   Left multiplications by random orthogonal matrices.
A(1:p,:) = gallery('qmult',A(1:p,:), method);
A(p+1:p+q,:) = gallery('qmult',A(p+1:p+q,:), method);
```

The following M-file is called by `randjorth.m` to generate random orthogonal matrices. It is part of the MATLAB distribution, with pathname `matlab/toolbox/matlab/elmat/private/qmult.m`.

```
function B = qmult(A,method)
%QMULT Pre-multiply matrix by random orthogonal matrix.
%   QMULT(A) returns Q*A where Q is a random real orthogonal matrix
%   from the Haar distribution of dimension the number of rows in A.
%   Special case: if A is a scalar then QMULT(A) is the same as QMULT(EYE(A)).
%   QMULT(A,METHOD) specifies how the computations are carried out.
%   METHOD = 0 is the default, while METHOD = 1 uses a call to QR,
%   which is much faster for large dimensions, even though it uses more flops.

%   Called by RANDSVD.

%   Reference:
%   [1] G. W. Stewart, The efficient generation of random
%       orthogonal matrices with an application to condition estimators,
%       SIAM J. Numer. Anal., 17 (1980), 403-409.
%
%   Nicholas J. Higham
%   Copyright 1984-2002 The MathWorks, Inc.
%   $Revision: 1.4 $  $Date: 2002/01/18 15:07:53 $

[n, m] = size(A);

%  Handle scalar A.
if max(n,m) == 1
   n = A;
   A = eye(n);
end

if nargin == 2 & method == 1
   [Q,R] = qr(randn(n));
   B = Q*diag(sign(diag(R)))*A;
   return
end

d = zeros(n);

for k = n-1:-1:1

    % Generate random Householder transformation.
    x = randn(n-k+1,1);
    s = norm(x);
    sgn = mysign(x(1));
    s = sgn*s;
    d(k) = -sgn;
    x(1) = x(1) + s;
    beta = s*x(1);

    % Apply the transformation to A.
    y = x'*A(k:n,:);
    A(k:n,:) = A(k:n,:) - x*(y/beta);

end

% Tidy up signs.
for i=1:n-1
```

```
    A(i,:) = d(i)*A(i,:);
end
A(n,:) = A(n,:)*mysign(randn);
B = A;
```

The following M-file is called by `qmult.m` for the true sign function. Again, it is part of the MATLAB distribution, with pathname `matlab/toolbox/matlab/elmat/private/mysign.m`.

```
function S = mysign(A)
%MYSIGN True sign function with MYSIGN(0) = 1.

%   Called by various matrices in elmat/private.
%
%   Nicholas J. Higham, Dec 1999.
%   Copyright 1984-2002 The MathWorks, Inc.
%   $Revision: 1.4 $  $Date: 2002/04/08 20:21:15 $

S = sign(A);
S(find(S==0)) = 1;
```

A.2 Random Pseudo-Unitaries

The following function generates random $\Sigma_{p,q}$-unitary matrices with user-specified condition number. The only differences between the following routine and the routine `randjorth.m` are the comments, the use of the notation 'sigma' rather than J, and the calls to `qmult_unit.m` rather than `qmult.m`.

```
function A = rand_pseunit(p,q,c,symm,method)
%RAND_PSEUNIT  Random complex pseudo-unitary matrix.
%   A = RAND_PSEUNIT(P,Q,C) forms a random (P+Q)-by-(P+Q)
%   complex pseudo-unitary matrix A, with COND(A) = C.
%   Pseudo-unitary means that A'*SIGMA*A = SIGMA, with A complex,
%   where SIGMA = BLKDIAG(EYE(P),-EYE(Q)).
%   If omitted, C defaults to SQRT(1/EPS).
%
%   RAND_PSEUNIT(N) and RAND_PSEUNIT(N,[],C)
%   both produce an N-by-N matrix with P = CEIL(N/2), Q = FLOOR(N/2).
%
%   The full calling sequence is RAND_PSEUNIT(P,Q,C,SYMM,METHOD).
%   If SYMM is nonzero symmetry is enforced.
%   The argument METHOD specifies how the underlying unitary
%   transformations are carried out.  If METHOD is nonzero
%   a call to QR is used, which is much faster than the default
%   method for large dimensions, though it uses more flops.
```

```
if nargin < 2 || isempty(q), q = floor(p/2); p = p-q; end
if nargin < 3 || isempty(c), c = sqrt(1/eps); end
if nargin < 4 || isempty(symm), symm = 0; end
if nargin < 5, method = 0; end

% This function requires q >= p, so...
if p > q
   A = randjunit(q,p,c,symm,method); % diag(eye(q),-eye(p))-unitary matrix.
   A = A([q+1:p+q 1:q],:); % Permute to produce pseudo-unitary matrix.
   A = A(:,[q+1:p+q 1:q]);
   return
end

if c >= 1
   c(1) = (1+c)/(2*sqrt(c));
   c(2:p) = 1 + (c(1)-1)*rand(p-1,1);
elseif p ~= 1
   error('Illegal value for C.  To specify COND set C >= 1.')
end

s = sqrt(c.^2-1);

A = blkdiag([diag(c) -diag(s); -diag(s) diag(c)], eye(q-p));

if symm
   U = blkdiag(qmult_unit(p, method),qmult_unit(q, method));
   A = U*A*U';
   A = (A + A')/2; % Ensure matrix is symmetric.
   return
end

A = left_mult(A,p,q,method); % Left multiplications by unitary matrices.
A = left_mult(A',p,q,method); % Right multiplications by unitary matrices.

function A = left_mult(A,p,q,method)
%LEFT_MULT   Left multiplications by random unitary matrices.
A(1:p,:) = qmult_unit(A(1:p,:), method);
A(p+1:p+q,:) = qmult_unit(A(p+1:p+q,:), method);
```

The following routine is called by `rand_pseunit.m` to generate random unitary matrices.

```
function B = qmult_unit(A,method)
%QMULT Pre-multiply matrix by random unitary matrix.
%   QMULT_UNIT(A) returns Q*A where Q is a random unitary matrix
%   from the Haar distribution of dimension the number of rows in A.
%   Special case: if A is a scalar then QMULT(A) is the same as QMULT(EYE(A)).
%   QMULT_UNIT(A,METHOD) specifies how the computations are carried out.
%   METHOD = 0 is the default, while METHOD = 1 uses a call to QR, which
%   is much faster for large dimensions, even though it uses more flops.

[n, m] = size(A);

%  Handle scalar A.
if max(n,m) == 1
   n = A;
   A = eye(n);
end

if nargin == 2 & method == 1
   F = randn(n) + randn(n)*i;
   [Q,R] = qr(F);
   B = Q*diag(exp(rand(n,1)*2*pi*i))*A;
   return
end

for k = n-1:-1:1

    % Generate random complex unitary Hermitian Householder.
    x = randn(n-k+1,1) + randn(n-k+1,1)*i;
    [v, beta] = gallery('house',x);

    % Apply the transformation to A.
    y = v'*A(k:n,:);
    A(k:n,:) = A(k:n,:) - beta*v*y;

end

A = diag(exp(rand(n,1)*2*pi*i))*A;
B = A;
```

The following M-file is called by `qmult_unit.m` to generate random complex unitary Hermitian Householder matrices. It is part of the MATLAB distribution, with pathname `matlab/toolbox/matlab/elmat/private/house.m`.

```
function [v, beta, s] = house(x, k)
%HOUSE Householder matrix that reduces a vector to a multiple of e_1.
%   [V, BETA, S] = GALLERY('HOUSE',X, K) takes an N-by-1 vector X
%   and returns V and BETA such that H*X = S*e_1,
%   where e_1 is the first column of EYE(N), ABS(S) = NORM(X),
%   and H = EYE(N) - BETA*V*V' is a Householder matrix.
%   The parameter K determines the sign of S:
%      K = 0 (default): sign(S) = -sign(X(1)) ("usual" choice),
%      K = 1:           sign(S) = sign(X(1))  (alternative choice).
%   If X is real then a further option, for real X only, is
%      K = 2:           sign(S) = 1.
%   If X is complex, then sign(X) = exp(i*arg(X)) which equals X./abs(X)
%   when X ~= 0.
%   In two special cases V = 0, BETA = 1 and S = X(1) are returned
%   (hence H = I, which is not strictly a Householder matrix):
%      - When X = 0.
%      - When X = alpha*e_1 and either K = 1, or K = 2 and alpha >= 0.

%   References:
%   [1] G. H. Golub and C. F. Van Loan, Matrix Computations, third edition,
%       Johns Hopkins University Press, Baltimore, Maryland, 1996, Sec. 5.1.
%   [2] N. J. Higham, Accuracy and Stability of Numerical Algorithms,
%       Society for Industrial and Applied Mathematics,
%       Philadelphia, PA, 1996; Sec. 18.1.
%   [3] G. W. Stewart, Introduction to Matrix Computations, Academic Press,
%       New York, 1973, pp. 231-234, 262.
%   [4] J. H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University
%       Press, 1965, pp. 48-50.
%
%   Nicholas J. Higham
%   Copyright 1984-2002 The MathWorks, Inc.
%   $Revision: 1.11 $  $Date: 2002/01/18 15:07:53 $

[n, m] = size(x);
if m > 1, error('Argument must be a column vector.'), end
if nargin < 2, k = 0; end

v = x;
nrmx = norm(x);
if nrmx == 0, beta = 1; s = 0; return, end    % Quit if x is the zero vector.

s = nrmx * mysign(x(1));

if k == 2
   if ~any(imag(x))
      if s < 0, k = 0, else k = 1; end
   else
      k = 0;
   end
end

if k == 0
   s = -s;
   v(1) = v(1) - s;
else
   v(1) = -norm(x(2:n))^2 / (x(1)+s)';     % NB the conjugate.
   if v(1) == 0 % Special case where V = 0: need H = I.
      beta = 1;
      return
   end
end
beta = -1/(s'*v(1));                        % NB the conjugate.
```

A.3 Random Real Symplectics

The following routine generates random real symplectic matrices with user-specified condition number, and allows for entry of an n-vector specifying the first n singular values instead of the condition number argument.

```
function A = rand_rsymp(n,c,method)
%RAND_RSYMP  Random real symplectic matrix.
%   A = RAND_RSYMP(N,C,METHOD) forms a random 2N-by-2N real symplectic
%   matrix A, where C determines COND(A) via three possible cases:
%    - If C is a scalar, then COND(A) = C.
%    - If C is a real N-vector with all elements >= 1, C specifies
%      SIGMA(1) >= SIGMA(2) >= ... >= SIGMA(N), the first N singular
%      values of A. The remainding singular values are calculated
%      according to constraints: SIGMA(N+K) = 1/SIGMA(K).
%    - If omitted, C defaults to SQRT(1/EPS).
%   Real symplectic means that A'*J*A = J, where
%   J = [ZEROS(N),EYE(N);-EYE(N),ZEROS(N)].
%
%   The argument METHOD specifies how the random complex unitary
%   matrices are generated. If METHOD is nonzero, a call to QR is used,
%   which is much faster than the default method for large dimensions,
%   though it uses more flops.

if nargin < 2 || isempty(c), c = sqrt(1/eps); end
if nargin < 3, method = 0; end

s = zeros(1,2*n);
p = length(c);

switch p

    case 1
    if c >= 1
        s(1) = sqrt(c);
        s(n:-1:2) = sort(1 + (s(1)-1)*rand(n-1,1));
        s(n+1:2*n) = 1./s(1:n);
    else
        error('Illegal value for C.  To specify COND set C >= 1.')
    end

    case n
    c = sort(c);
    if c(1) < 1
        error('Illegal value for C. All values must be >=1.')
    else
        s(1:n) = c(n:-1:1);
        s(n+1:2*n) = 1./s(1:n);
    end
    otherwise
    error('Vector of singular values must be of length 1 or N.')

end

U = symp_orth(n,method); V = symp_orth(n,method);
A = U*diag(s)*V';

% subfunction
function U = symp_orth(n,method)
%SYMP_ORTH  Random symplectic orthogonal matrix
B = qmult_unit(n,method);
Re = real(B); Im = imag(B);
U = [Re,Im;-Im,Re];
```

A.4 Random Real Perplectics

The following routine generates random real perplectic matrices with user-specified condition number.

```
function A = rand_rperp(n,c,method)
%RAND_RPERP  Random real perplectic matrix.
%   A = RAND_RPERP(N,C,METHOD) forms a random N-by-N real perplectic
%   matrix A, where COND(A) = C.
%   If omitted, C defaults to SQRT(1/EPS).
%   Real perplectic means that A'*R*A = R, where R defined by
%   R = ZEROS(N); R(N:N-1:N^2-N+1) = 1 (R = 1 if N = 1)
%   The argument METHOD specifies how the random orthogonal
%   matrices are generated. If METHOD is nonzero, a call to QR is used,
%   which is much faster than the default method for large dimensions,
%   though it uses more flops.

if nargin < 2 || isempty(c), c = sqrt(1/eps); end
if nargin < 3, method = 0; end

p = floor(n/2); s = zeros(1,n);

if c < 1
   error('Illegal value for C.  To specify COND set C >= 1.')

elseif rem(n,2)

   s(1) = sqrt(c);
   s(p:-1:2) = sort(1 + (s(1)-1)*rand(p-1,1));
   s(p+1) = mysign(randn);
   s(p+2:n) = 1./s(p:-1:1);
   U = perp_orth_odd(p,method); V = perp_orth_odd(p,method);
   A = U*diag(s)*V';

else

   s(1) = sqrt(c);
   s(p:-1:2) = sort(1 + (s(1)-1)*rand(p-1,1));
   s(p+1:n) = 1./s(p:-1:1);

   U = perp_orth_even(p,method); V = perp_orth_even(p,method);
   A = U*diag(s)*V';

end

% subfunction 1
function U = perp_orth_even(k,method)
%PERP_ORTH_2N   Random perplectic orthogonal of size 2k
P = gallery('qmult',k,method); Q = gallery('qmult',k,method);
X = P + Q(k:-1:1,k:-1:1); Y = P(:,k:-1:1) - Q(k:-1:1,:);
U = (1/2)*[ X  Y ; Y(k:-1:1,k:-1:1) X(k:-1:1,k:-1:1) ];

% subfunction 2
function U = perp_orth_odd(k,method)
%PERP_ORTH_ODD2  Random perplectic orthogonal of size 2k+1
P = gallery('qmult',k+1,method); Q = gallery('qmult',k,method);
U = zeros(2*k+1);
U = (1/2)*[ P(1:k,1:k) + Q(k:-1:1,k:-1:1) sqrt(2)*P(1:k,k+1) ...
            P(1:k,k:-1:1) - Q(k:-1:1,:) ;
            sqrt(2)*P(k+1,1:k) 2*P(k+1,k+1) sqrt(2)*P(k+1,k:-1:1) ; ...
            P(k:-1:1,1:k) - Q(:,k:-1:1) sqrt(2)*P(k:-1:1,k+1) ...
            P(k:-1:1,k:-1:1) + Q ];
```

A.5 Random Complex Structured Matrices

The following routine generates random matrices in the following groups:

- Complex Orthogonals
- Complex Pseudo-Orthogonals
- Complex Symplectics
- Conjugate Symplectics

```
function A = rand_cstruct(f,n,c)
%RAND_CSTRUCT   Random complex structured matrices
%  * A = RAND_CSTRUCT(1,N,C) forms a random N-by-N complex orthogonal
%    matrix A, with COND(A) approximately equal to C.
%  * A = RAND_CSTRUCT(2,[P Q],C) forms a random (P+Q)-by-(P+Q)
%    complex pseudo-orthogonal matrix A, with COND(A) approximately
%    equal to C.
%  * A = RAND_CSTRUCT(3,N,C) forms a random N-by-N complex symplectic
%    matrix A, with COND(A) approximately equal to C.
%  * A = RAND_CSTRUCT(4,N,C) forms a random N-by-N conjugate symplectic
%    matrix A, with COND(A) approximately equal to C.
%
%    In each case, X contains the LS solution [A0,A1,A2,A3] for
%    LOG(C) = A0 + A1*K + A2*N + A3*K^2 from empirical data, where
%    K is the number of G-reflectors applied, N is the matrix size, and
%    C is the observed condition number of the matrix.
%
%    QUAD contains the co-efficients of the quadratic
%    A3*K^2 + A1*K + (A2*N + A0 - LOG(C)).
%    QUAD is solved for K, and K is rounded to the nearest integer to
%    formulate how many G-reflectors to apply.

if nargin < 3 || isempty(c), c = sqrt(1/eps); end

switch f

    case 1

    I = eye(n); A = I;
    X = [ 2.0344 0.0044 1.9239 -0.0249 ];
    quad = [ X(4), X(3), X(2)*n + X(1) - log(c) ];
    k = round(min(sort(roots(quad))));
    if k < 1, k = 1; end
    fprintf('Number of G-reflectors being applied : %g', k)

    for j = 1:k

        u = randn(n,1) + randn(n,1)*i;
        G = I - (2/(u.'*u))*u*u.';
        A = A*G;

    end

    case 2

    [a,b] = size(n);
    if a*b ~= 2
        error('Must enter vector [p q] for matrix size.')
    end

    p = n(1); q = n(2); n = p + q; I = eye(n); A = I;
    X = [ 1.9510 0.0058 1.9080 -0.0245 ];
    quad = [ X(4), X(3), X(2)*n + X(1) - log(c) ];
    k = round(min(sort(roots(quad))));
    if k < 1, k = 1; end
```

```
fprintf('Number of G-reflectors being applied : %g', k)

for j = 1:k

    u = randn(n,1) + randn(n,1)*i;
    sig = blkdiag(eye(p),-eye(q));
    G = I - (2/(u.'*sig*u))*u*u.'*sig;
    A = A*G;

end

case 3

I = eye(2*n); A = I;
X = [ 3.9794 0.0249 5.8397 -0.2311 ];
quad = [ X(4), X(3), X(2)*n + X(1) - log(c) ];
k = round(min(sort(roots(quad))));
if k < 1, k = 1; end
fprintf('Number of G-reflectors being applied : %g', k)

for j = 1:k

    u = randn(2*n,1) + randn(2*n,1)*i;
    beta = randn + randn*i;
    J = [zeros(n) eye(n) ; -eye(n) zeros(n)];
    G = I + beta*u*u.'*J;
    A = A*G;

end

case 4

I = eye(2*n); A = I;
X = [ 3.6339 0.0063 2.0899 -0.0274];
quad = [ X(4), X(3), X(2)*n + X(1) - log(c) ];
k = round(min(sort(roots(quad))));
if k < 1, k = 1; end
fprintf('Number of G-reflectors being applied : %g', k)

for j = 1:k

    u = randn(2*n,1) + randn(2*n,1)*i;
    J = [zeros(n) eye(n) ; -eye(n) zeros(n)];
    q = u'*J*u; r = -1/q; r = r - real(r);
    beta = r*exp(2*pi*i*rand) + r;
    G = I + beta*u*u'*J;
    A = A*G;

end
```

```
    otherwise
    error('Must choose matrix type 1-4.')

end
```