# SAMPLING EIGENVALUES OF RANDOM UNITARY MATRICES

NICHOLAS WEST

**Abstract.** This project describes how all $n$ eigenvalues of a random unitary matrix may be sampled in $O(n^2)$ time using $O(n)$ random parameters. The main ingredients are (1) a sampling algorithm that samples directly from data-sparse representations of Hessenberg reduced unitary matrices, and (2) an algorithm for computing the eigenvalues of such unitary upper Hessenberg matrices in $O(n^2)$ time. We implement the main algorithms discussed in Julia. We include some numerical experiments to verify theoretical run-time guarantees and that the sampled matrices obey theoretical results about eigenvalues of random unitary matrices. We conclude with a discussion of how our implementation can be improved.

**1. Introduction.** One important ensemble of random matrices is the Gaussian unitary ensemble (GUE)—that is, the ensemble of uniformly random unitary matrices.[1] A unitary matrix $U$ multiplied by its conjugate transpose is the identity, i.e. $U^*U = I$. The eigenvalues of a unitary matrix always lie on the unit circle in the complex plane, and the eigenvalues of GUE matrices distribute themselves uniformly on the unit circle. An important application of random unitary matrices is in modeling random quantum states, and random unitary matrices also exhibit interesting combinatorial ties, for example to longest increasing subsequences (as discussed in the 18.338 course notes).

But how do we efficiently sample a random unitary matrix? How do we efficiently sample the *eigenvalues* of a random unitary matrix? Loosely speaking, one approach (described in more detail in section 3) is to form a matrix with random entries, to compute the QR-factorization of this random matrix, and then to take the $Q$ factor to be the random unitary matrix. For the eigenvalues, one simply computes the eigenvalues of $Q$ using standard eigenvalue routines. The formation of $Q$ typically requires $O(n^3)$ floating point operations (flops), and since $Q$ is a dense, non-symmetric matrix, it will require another $O(n^3)$ flops to compute the eigenvalues of $Q$.

The central goal of this report is to show how when we only want to sample *eigenvalues* of random unitary matrices, we can improve the sampling complexity to $O(n^2)$. The basic idea here is that a reduced Hessenberg form of a unitary matrix—which has the same eigenvalues as the original—can be represented in $O(n)$ parameters. This form can be sampled directly, as described in section 4, and then a modified eigenvalue routine can be run to compute all eigenvalues in $O(n^2)$ time, as described in section 5. We implement this process in Julia (see section 6 for software usage). Along the way, we will review some of the main principles involved in sampling random unitary matrices.

This report follows closely [2] for sampling random unitary matrices and [1] for the implementation of the QR algorithm for unitary matrices, although the exposition here is our own. The accompanying Julia code is available on GitHub.[2] A novel aspect of this project is its implementation in Julia, whereas the code in [1] is available in Fortran.[3]

**2. Background.** In this section, we provide some background that will be useful in the body of the paper. For the reader familiar with numerical linear algebra

[1] More precisely, when we use phrases like "uniformly random unitary matrix," "GUE matrix," and "Haar-distributed unitary matrix," we mean a random variable on the compact Lie group of unitary matrices with a probability density function given by the unique normalized left (or right) Haar distribution. We will use the same terms for a particular matrix sampled from this distribution.

[2] `https://github.com/Westnickp/FastUnitaryEigenvalues.jl`

[3] `https://github.com/eiscor/eiscor`

and random matrix theory, this may be skipped, excepting perhaps our notational conventions.

**2.1. Basic notation.** We will use $\mathcal{N}_{\mathbb{C}}(0,1)$ to denote the standard complex normal distribution of mean 0 and variance 1. Relatedly, we will denote column vectors and matrices of independent, identically distributed (i.i.d.) standard complex normals by $\mathcal{N}_{\mathbb{C}}^{n}(0,1)$ and $\mathcal{N}_{\mathbb{C}}^{n\times n}(0,1)$, where $n$ is the size of the relevant vector or matrix. We will also encounter the complex chi-squared distribution with $n$ degrees of freedom, denoted $\chi^2_{\mathbb{C}}(n)$, which importantly has the same distribution as the sum of $n$ squared i.i.d. standard complex normal random variables. The complex conjugate of a matrix $A$ is denoted by $A^*$. The standard Euclidean norm of a vector is denoted by $\|u\| = \sqrt{u^*u}$. The argument (or angle) of a complex number $z$ is denoted $\operatorname{Arg}(z)$ with range $[0, 2\pi)$. The standard basis vector with a 1 in position $j$ and zeros otherwise is denoted by $e_j$—the size of the vector will be clear from context. We sometimes adopt array-like indexing and slicing to indicate elements of vectors and matrices, e.g. $v[1]$ is the first entry of a vector $v$, and $A[k : k+1, k : k+1]$ is a 2-by-2 submatrix of $A$ corresponding to the $k$th and $(k+1)$th rows and columns.

**2.2. Householder reflectors.** Householder reflectors are special examples of unitary matrices that have a simple but profoundly useful ability to zero-out all but one entry of a vector. More precisely, let $u \in \mathbb{C}^n$ be a given non-zero vector. Then there exists a Householder reflector $H(u) \in \mathbb{C}^{n\times n}$ defined by

$$H(u) \equiv I - \frac{2ss^*}{s^*s}, \quad s \equiv u + e^{i\operatorname{Arg}(u[1])}\|u\|e_1 \tag{2.1}$$

such that

$$H(u)u = -e^{i\operatorname{Arg}(u[1])}\|u\|e_1. \tag{2.2}$$

It is easy to verify directly from the definition in (2.1) that $H(u)$ is a Hermitian, unitary matrix and that it satisfies (2.2). Notably, although $H(u)$ is an $n \times n$ matrix, it is fully determined by the vector $u$, and therefore representable with only $O(n)$ parameters, where $n$ is the number of non-zeros in $u$. Moreover, its action on a vector can be computed in $O(n)$ time. Notably, if $u$ is zero except at $u[j : k]$ for $1 \le j < k \le n$, then the Householder reflector has the block form

$$H(u) = \begin{bmatrix} I_{j-1} & & \\ & H(u[j:k]) & \\ & & I_{n-k}, \end{bmatrix} \tag{2.3}$$

and we refer to the $[j : k, j : k]$ slice of $H(u)$ as its *active part*, since this is the only relevant component for computation.

**2.3. QR decomposition.** The QR decomposition of a matrix is the decomposition of $A \in \mathbb{C}^{n\times n}$ into a unitary matrix $Q$ and an upper triangular matrix $R$ such that $A = QR$. Such a decomposition is possible for all $A$. Note that the decomposition is non-unique in general, but becomes unique if the complex phases of the diagonal entries of $R$ are fixed *a priori*. In particular, $Q$ and $R$ are unique if $R$ is forced to have positive, real diagonal entries. The decomposition can be computed via a Gram-Schmidt process; however, this can be numerically unstable. Since orthogonal matrices are numerically well-behaved, a safer approach (with the same complexity) is to use Householder reflectors. One can form a sequence of Householder reflectors

$H_1, H_2, \cdots, H_{n-1}$ such that $H_{n-1}H_{n-2}\cdots H_1 A = R$ is upper triangular, which yields the $R$ factor, and $Q^*$ is this product of reflectors. A slight modification (as shown in Algorithm 3.1) yields a unique $Q$.

**2.4. QR algorithm.** One of the most important eigenvalue algorithms is the QR algorithm, so-named because it relies so heavily on the QR decomposition. The algorithm (in theory) is very simple to state. We set $A_0 = A$, then compute for $k = 0, 1, \cdots$

$$B_k = Q_k R_k = \text{qr}(A_k), \quad A_{k+1} = R_k Q_k. \tag{2.4}$$

Swapping the $Q$ and $R$ factors is an eigenvalue-preserving similarity transformation, and it can be connected to power iterations (and implicitly to inverse-power iteration) for computing an extremal eigenvalue. A shift $\mu$ can be incorporated to accelerate convergence by modifying (2.4) to

$$B_k = Q_k R_k = \text{qr}(A - \mu I), \quad A_{k+1} = R_k Q_k + \mu I. \tag{2.5}$$

A well-selected adaptive shifting strategy means that an eigenvalue can be computed in $O(1)$ iterates. This eigenvalue can then be deflated from the problem and the process repeated on a smaller $(n-1)$-by-$(n-1)$ problem. In practice, the algorithm is seldom implemented as stated. Rather, similarity transformations are performed to reduce the starting matrix to upper Hessenberg form (a property preserved during QR iterates). Usually the shifted QR algorithm is implemented implicitly through the introduction of a *bulge* disturbing the Hessenberg structure, which is then restored through a sequence of further similarity transformations. Further details are offered in [3, 4, 5].

**2.5. Haar-distributed unitary matrices.** We formalize here the claim that random unitary matrices have eigenvalues uniformly distributed on the unit circle.

THEOREM 2.1. *Let $U_n$ be an n-by-n unitary matrix sampled from the Haar distribution. Let $x$ be a uniformly random eigenvalue of $U_n$. Then for $\theta_1 < \theta_2 \in [0, 2\pi]$, $P(\theta_1 \leq \text{Arg}(x) \leq \theta_2) = (\theta_2 - \theta_1)/(2\pi)$.*

Another important law about which we do not go into detail here is the *Wigner surmise*, which describes the distribution of the spacings of the eigenvalues on the unit circle [2]. In future work we will investigate this law as well.

**3. Sampling unitary matrices.** Our goal in this section is to describe two sampling routines for random unitary matrices. These will serve as building blocks for our main result in the next section, a third algorithm, which shows how a factored representation of the Hessenberg form of a unitary matrix can be sampled directly whose eigenvalues share the same distribution as a GUE matrix. Later on in section 5, we will detail how to compute the eigenvalues of this matrix in $O(n^2)$ time.

Perhaps the simplest approach to sampling unitary matrices is stated in Algorithm 3.1. We form a random matrix with entries independently and identically distributed according to complex standard normals, such that for each $i, j$ we have $A_{ij} \sim \mathcal{N}_{\mathbb{C}}(0, 1)$—this is sometimes called the *Ginibre ensemble*. Then we compute a QR decomposition (with the caveat that R must have positive, real entries). The resulting $Q$ factor is a unitary matrix sampled according to the Haar-distribution—for further details, see [6].

PROPOSITION 3.1. *Let $A \sim \mathcal{N}_{\mathbb{C}}^{n\times n}(0, 1)$, let $Q$ be the unitary matrix with a corresponding $R$ upper triangular with real, positive diagonal entries such that $A = QR$. Then $Q$ is a Haar-distributed unitary matrix.*

**Algorithm 3.1** Sample unitary matrix

```
Sample $A \sim \mathcal{N}_{\mathbb{C}}^{n\times n}(0,1)$
Compute $QR \leftarrow A$
Form diagonal matrix $\Lambda_{ii} = R_{ii}/|R_{ii}|$
return $Q\Lambda$
```

It is possible to sample unitary matrices in a more clever fashion. Informally, the elements of a unitary matrix are not independently distributed, so the initial sampling of $A$ from the Ginibre ensemble with $n^2$ independent complex numbers is overkill. We may seek to form our random unitary matrix by sampling fewer random numbers.

Let's derive a method for doing this. We begin by considering the intermediate steps of the Householder reduction of $A$ to compute its $QR$ factorization. Write the columns of $A \sim \mathcal{N}_{\mathbb{C}}^{n\times n}(0,1)$ via

$$A = \begin{bmatrix} g_1 & g_2 & \cdots & g_n \end{bmatrix}, \tag{3.1}$$

where each column is a vector of independent standard complex normals, $g_i \sim \mathcal{N}_{\mathbb{C}}^{n}(0,1)$. The first step of a Householder QR decomposition involves computing $H_1 \equiv H(g_1)$ such that

$$H_1 A = H(g_1)A = \begin{bmatrix} -e^{i\,\mathrm{Arg}(g_1[1])}\|g_1\|e_1 & H(g_1)g_2 & H(g_1)g_3 & \cdots & H(g_1)g_n \end{bmatrix}. \tag{3.2}$$

That is, the first column of $H_1A$ is non-zero only in the first entry. Moreover, $H_1 = H(g_1)$ is an orthogonal matrix fully determined by $g_1$ independent of $g_2, \cdots, g_n$. Then the trailing columns of $H_1A$ are of the form $Qx$ where $Q$ is a unitary matrix fixed independently of $x$ and $x$ is a vector of standard complex normals. In this context, this means that $H_1 g_i \sim \mathcal{N}_{\mathbb{C}}^{n}(0,1)$ when $i \neq 1$. We may then rewrite (3.2) in the more suggestive block form

$$H_1 A = \begin{bmatrix} -e^{i\,\mathrm{Arg}(g_1[1])}\|g_1\| & s_{n-1}^T \\ 0_{n-1} & A_{n-1} \end{bmatrix}, \tag{3.3}$$

where $s_{n-1} \sim \mathcal{N}_{\mathbb{C}}^{n-1}(0,1)$ is a vector of i.i.d. standard complex normals, $0_{n-1}$ is a column vector of $n-1$ zeros, and $A_{n-1} \sim \mathcal{N}_{\mathbb{C}}^{n-1\times n-1}(0,1)$ is a smaller Ginibre matrix. Proceeding inductively, we can construct an $H_2$ whose active part is in the lower $(n-1)$-by-$(n-1)$ block, determined by the first column of $A_{n-1}$, so that $H_2H_1A$ has its first two columns in the desired upper triangular form and the remaining entries above the diagonal or in the trailing columns i.i.d. standard normals. Proceeding in this fashion, we obtain the following result.

Proposition 3.2. *Let $v_j \sim \mathcal{N}_{\mathbb{C}}^{n-j+1}(0,1)$ for $j = 1, 2, \cdots, n$ and define a sequence of orthogonal matrices $H_1, H_2, \cdots, H_{n-1}$ and a diagonal matrix $D$ such that*

$$H_j \equiv \begin{bmatrix} I_{j-1} & 0 \\ 0 & H(v_j) \end{bmatrix}, \quad D_{jj} = e^{-i\,\mathrm{Arg}(v_j[1])}, \tag{3.4}$$

*together with the prescription $D_{nn} = e^{-i\,\mathrm{Arg}(\theta)}$ for some $\theta \sim$ Uniform$[0, 2\pi)$. Then*

$$U \equiv H_1 H_2 \cdots H_{n-1} D \tag{3.5}$$

*is a Haar-distributed random unitary matrix.*

*Proof.* This follows from the preceding discussion about the QR decomposition of a Ginibre matrix. A more detailed and formal proof is offered in [2]. □

*Remark* 3.3. Note that Proposition 3.2 implies that if $U_{n-1}$ is a Haar-distributed unitary matrix of size $(n-1)$, then

$$U_n \equiv H(v) \begin{bmatrix} 1 & \\ & U_{n-1} \end{bmatrix} \begin{bmatrix} d & \\ & I_{n-1} \end{bmatrix} \tag{3.6}$$

is Haar-distributed of size $n$ if $v \sim \mathcal{N}_{\mathbb{C}}^{n}(0,1)$ and $d \equiv e^{-i\,\mathrm{Arg}(v[1])}$. The converse holds too.

The resulting sampling algorithm is described in Algorithm 3.2. Note that rather than sampling $n^2$ random normal variables, we only sample $n+(n-1)+\cdots+3+2 \approx n^2/2$ together with a single random angle, which reduces the number of parameters by about half. Additionally, this factored form can be exploited in a QR algorithm to speed up the eigenvalue decomposition by a similar factor of 2. We omit the details; see [2] for further discussion. Importantly, to sample the eigenvalues would still be $O(n^3)$.

**Algorithm 3.2** Sample unitary matrix in factored form

```
for $j = 1 : n-1$ do
    Sample $v_j \sim \mathcal{N}_{\mathbb{C}}^{n-j+1}(0,1)$
    Form $H_j \leftarrow H(v_j)$
    Set $D_{jj} \leftarrow e^{-i\,\mathrm{Arg}(v_j[1])}$
end for
Sample $D_{nn} = e^{-i\theta}$ for $\theta \sim \mathrm{Uniform}[0, 2\pi)$
return $H_1 H_2 \cdots H_{n-1} D$
```

## 4. Sampling the Hessenberg form.

We consider in this section modifying the sampling algorithm further. An important first step in many practical QR eigenvalue algorithms is the reduction of a matrix to upper Hessenberg form. Importantly, the Hessenberg reduction of a unitary matrix is data-sparse—it can be represented with only $O(n)$ parameters, rather than $O(n^2)$ (as detailed in Proposition 4.1). In this section our goal will be to show how to sample this data-sparse reduction directly.

Consider the following result.

PROPOSITION 4.1. *Let $U$ be a unitary, upper Hessenberg matrix. Then there exist 2-by-2 unitary matrices $\tilde{C}_i$ for $i = 1, 2, \cdots, n-1$ such that defining*

$$C_i \equiv \begin{bmatrix} I_{i-1} & & \\ & \tilde{C}_i & \\ & & I_{n-i-1} \end{bmatrix} \tag{4.1}$$

*yields*

$$U = C_1 C_2 \cdots C_{n-1}. \tag{4.2}$$

*The construction of these matrices is described in the proof.*

*Proof.* One can use Givens rotators (which are closely related to Householder reflectors) to zero out the subdiagonal of the upper Hessenberg $U$ one element at a time. Since $U$ is unitary, such that the columns are orthonormal, such a rotator will also zero out the row above the diagonal corresponding to that subdiagonal entry. The result of $n-1$ such rotators is a diagonal matrix of complex numbers, call it $D$. So

$$G_{n-1}\cdots G_1 U = D, \tag{4.3}$$

which means

$$U = G_1 \cdots G_{n-1} D. \tag{4.4}$$

To eliminate the diagonal matrix, we note that the active part of each $G_j$ is in the $[j:j+1, j:j+1]$ component, so that we can multiply the $j$th column of each $G_j$ by $D_{jj}$ for $j < n$ and set $C_j$ to be the result. We additionally multiply the $n$th column of $G_{n-1}$ by $D_{nn}$ in constructing $C_{n-1}$, which yields

$$U = C_1 \cdots C_{n-1} \tag{4.5}$$

as desired. □

We have just shown that a unitary upper Hessenberg matrix can be represented as a product of $n$ essentially 2-by-2 matrices. The action of any $C_i$ on a vector can be computed in $O(1)$ time, so that with this data-sparse representation the action of a unitary upper Hessenberg matrix can be computed in $O(n)$ time. In the next section, we will show how this form can be used favorably in a QR algorithm for eigenvalue computation.

Here, we wish to show that this factored form can be sampled directly whilst maintaining the same joint distribution of the eigenvalues. In Proposition 3.2 and the following remark, we showed that a Haar-distributed $n \times n$ random unitary matrix $U_n$ has the form

$$U_n = H(v) \begin{bmatrix} 1 & \\ & U_{n-1} \end{bmatrix} \begin{bmatrix} d & \\ & I_{n-1} \end{bmatrix} \tag{4.6}$$

where $v \sim \mathcal{N}_{\mathbb{C}}^n(0,1)$, $d = e^{-i\,\mathrm{Arg}(v[1])}$, and $U_{n-1}$ is a Haar-distributed random unitary matrix of size $n-1$. Suppose we want to force the $H(v)$ to be essentially 2-by-2. We can let $\tilde{H} \equiv H(v[2:n])$ be the $(n-1)$-by-$(n-1)$ Householder reflector such that

$$\tilde{H} v[2:n] = -e^{i\,\mathrm{Arg}(v[2])} \left\| v[2:n] \right\| e_1. \tag{4.7}$$

This naturally yields an $n$-by-$n$ matrix $\hat{H}$ whose action on $v$ is known:

$$\hat{H} \equiv \begin{bmatrix} 1 & \\ & \tilde{H} \end{bmatrix}, \quad w \equiv \hat{H} v = \begin{bmatrix} v[1] \\ -e^{i\,\mathrm{Arg}(v[2])} \| v[2:n] \| \\ 0_{n-2} \end{bmatrix} \tag{4.8}$$

so that with this choice of $\hat{H}$

$$\begin{aligned} \hat{H} H(v) \hat{H}^* &= \hat{H} \left( I - \frac{2vv^*}{v^*v} \right) \hat{H}^* \\ &= I - \frac{2(\hat{H}v)(\hat{H}v)^*}{(\hat{H}v)^*(\hat{H}v)} \\ &= I - \frac{2ww^*}{w^*w} \\ &= H(w). \end{aligned} \tag{4.9}$$

Notably, $w$ has only two-nonzero entries that are characterized fully by the random (and independent!) quantities $v[1] \sim \mathcal{N}_{\mathbb{C}}(0,1)$, $\mathrm{Arg}(v[2]) \sim \mathrm{Unif}[0,2\pi)$, and $\|v[2:n]\| \sim \sqrt{\chi^2_{\mathbb{C}}(n-1)}$. Moreover, we note the following simplification of a similarity transformation on $U_n$ given by $\hat{H}$:

$$\begin{aligned} \hat{H}U_n\hat{H}^* &= \hat{H}H(v)\begin{bmatrix}1 & \\ & U_{n-1}\end{bmatrix}\begin{bmatrix}d & \\ & I_{n-1}\end{bmatrix}\hat{H}^* \\ &= (\hat{H}H(v)\hat{H}^*)\hat{H}\begin{bmatrix}1 & \\ & U_{n-1}\end{bmatrix}\begin{bmatrix}d & \\ & I_{n-1}\end{bmatrix}\hat{H}^* \\ &= H(w)\begin{bmatrix}1 & \\ & \tilde{H}U_{n-1}\tilde{H}^*\end{bmatrix}\begin{bmatrix}d & \\ & I_{n-1}\end{bmatrix}. \end{aligned} \tag{4.10}$$

But $\tilde{H}$ is a unitary matrix independent of $U_{n-1}$, so that $\tilde{H}U_{n-1}\tilde{H}^*$ is also a Haar-distributed random unitary matrix. Since the eigenvalues of $U_n$ are identical to the similar matrix $\hat{H}U_n\hat{H}^*$, in (4.6) we may consider sampling $H(w)$ rather than sampling $H(v)$, and the eigenvalues of the sampled matrix will have the same distribution as (4.6). Proceeding inductively (with a slight modification to enforce the form of Proposition 4.1) implies the following result, which is implemented in Algorithm 3.2.

Proposition 4.2. *Let $w_j \in \mathbb{C}^2$ have the form*

$$w_j \sim \begin{bmatrix}\mathcal{N}_{\mathbb{C}}(0,1) \\ \sqrt{\chi^2_{\mathbb{C}}(n-j)}\end{bmatrix} \tag{4.11}$$

*for $j = 1, 2, \cdots, n-1$ and define a sequence of orthogonal matrices $H_1, H_2, \cdots, H_{n-1}$ and a diagonal matrix $D$ such that*

$$H_j \equiv \begin{bmatrix} I_{j-1} & & \\ & H(w_j) & \\ & & I_{n-j-2}\end{bmatrix}, \quad D_{jj} = e^{-i\,\mathrm{Arg}(w_j[1])}, \tag{4.12}$$

*together with the prescription $D_{nn} = e^{-i\,\mathrm{Arg}(\theta)}$ for some $\theta \sim \mathrm{Uniform}[0,2\pi)$. Then*

$$U \equiv H_1H_2\cdots H_{n-1}D \tag{4.13}$$

*is a random unitary upper Hessenberg matrix whose eigenvalues have the same joint distribution as a Haar-distributed random unitary matrix.*

**5. Eigenvalue computation.** In this section we detail how the sampled form obtained in Algorithm 3.2 can be used in a fast eigenvalue routine.

The basic idea of the algorithm is that the practical QR algorithm outlined in subsection 2.4 when applied to the special factored form of a unitary upper Hessenberg matrix can be implemented such that one $QR$ iterate requires only $O(n)$ flops. To see this, we describe below the stages of a $QR$ iterate and how they may be implemented efficiently. Throughout, we assume that a unitary upper Hessenberg matrix $U = C_1C_2\cdots C_{n-1}$ is the input to the algorithm.

Recall that one iterate of a bulge-chasing QR algorithm has three main phases: (1) the computation of a shift, (2) the computation of a bulge, (3) the chasing of the bulge to restore Hessenberg structure. Let's detail these steps here.

*Computing a shift.* To compute a shift $\mu$, we first compute the the 2-by-2 matrix $U[n-1:n, n-1:n]$ in the bottom right corner of $U$. Note that this is computed

**Algorithm 4.1** Sample factored Hessenberg form of a unitary matrix

```
for $j = 1 : n-1$ do
    Sample $w_j \sim \begin{bmatrix} \mathcal{N}_{\mathbb{C}}(0,1) \\ \sqrt{\chi^2_{\mathbb{C}}(n-j)} \end{bmatrix}$
    Form $C_j \leftarrow H(w_j)$
    Set $D_{jj} \leftarrow e^{-i\,\mathrm{Arg}(w_j[1])}$
    Set $C_j \leftarrow C_j \begin{bmatrix} I_{j-1} & & \\ & D_{jj} & \\ & & I_{n-j} \end{bmatrix}$
end for
Sample $D_{nn} \leftarrow e^{-i\theta}$ for $\theta \sim \mathrm{Uniform}[0, 2\pi)$
Set $C_{n-1} \leftarrow C_{n-1} \begin{bmatrix} I_{n-1} & \\ & D_{nn} \end{bmatrix}$
return $U = C_1 C_2 \cdots C_{n-1}$
```

from only $C_{n-1}$ and $C_{n-2}$ since these are the only transformations acting on the last two rows of $U$. We can compute the two eigenvalues of $U[n-1:n, n-1:n]$, select the eigenvalue $\tilde{\mu}$ closer to $U[n,n]$, and since we know eigenvalues of $U$ lie on the unit circle, we project it onto the unit circle $\mu = \tilde{\mu}/|\mu|$ (called a projected Wilkinson shift [1]). All of these steps are carried out on small $2 \times 2$ matrices, requiring $O(1)$ flops and memory.

*Computing a bulge*: We wish to compute some unitary $B_1$ such that $B_1^* U B_1$ corresponds to performing one iterate of the shifted QR algorithm. It turns out[4] that this is equivalent to solving for some $B_1$ which is a unitary matrix active only in the $[1:2, 1:2]$ part such that given $x = (U - \mu I)e_1$, which has only two non-zero entries in positions 1 and 2 since $U$ is upper Hessenberg, it holds that $B_1^* x = \gamma e_1$ for some $\gamma \in \mathbb{C}$. This requires only $C_1$ and $C_2$ to form $x$, and solving for $B_1$ is equivalent to computing a $2 \times 2$ Givens rotator or Householder reflector. Consequently, this also only requires $O(1)$ flops and memory.

*Chasing the bulge*: Given $B_1$, we wish to compute the transformations

$$C_1 C_2 \cdots C_{n-1} \to B_1^* C_1 C_2 \cdots C_{n-1} B_1 \to \tilde{C}_1 \tilde{C}_2 \cdots \tilde{C}_{n-1}. \tag{5.1}$$

That is, we wish to perturb the unitary Hessenberg structure with the similarity transformation $B_1^* U B_1$ and then restore the unitary Hessenberg structure through a sequence of cheap similarity transformations to obtain a new unitary upper Hessenberg matrix $\tilde{C}_1 \tilde{C}_2 \cdots \tilde{C}_{n-1}$. We will use two fundamental operations repeatedly: *fusions* and *turnovers* (this terminology follows [1]). A *fusion* is simply the merging of two adjacent core transformations that act on the same rows. That is, if $A_j$ and $B_j$ are unitary core transformations acting on rows $j$ and $j+1$, then $A_j B_j = \tilde{A}_j$ for some new unitary $\tilde{A}_j$ acting on the same rows (in essence, the product of two $2 \times 2$ unitary matrices is a $2 \times 2$ unitary matrix). A *turnover* is an operation that converts a sequence of core transformations of the form $A_j A_{j+1} B_j$ into a sequence of the form

[4]The details showing that such a similarity transformation is equivalent to one step of the shifted-QR algorithm are somewhat involved, but they rely on the implicit $Q$ theorem and a few other results that can be found in [5, Chapter 7].

$\tilde{A}_{j+1}\tilde{A}_j\tilde{B}_{j+1}$. That is, we wish to reconfigure the $3\times 3$ active part $A$ according to

$$\begin{aligned} A_jA_{j+1}B_j &= \begin{bmatrix} * & * & \\ * & * & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & * & * \\ & * & * \end{bmatrix}\begin{bmatrix} * & * & \\ * & * & \\ & & 1 \end{bmatrix} \\ &= \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} = A \\ &= \begin{bmatrix} 1 & & \\ & * & * \\ & * & * \end{bmatrix}\begin{bmatrix} * & * & \\ * & * & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & * & * \\ & * & * \end{bmatrix} \\ &= \tilde{A}_{j+1}\tilde{A}_j\tilde{B}_{j+1}. \end{aligned} \tag{5.2}$$

In detail, we solve for $\tilde{A}_{j+1}$ such that

$$\tilde{A}_{j+1}^*A = \begin{bmatrix} * & * & * \\ * & * & * \\ 0 & * & * \end{bmatrix}, \tag{5.3}$$

so that we can compute $\tilde{A}_j$ such that

$$\tilde{A}_j^*\tilde{A}_{j+1}^*A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & * & * \\ 0 & * & * \end{bmatrix}. \tag{5.4}$$

Setting $\tilde{B}_{j+1} = (\tilde{A}_j^*\tilde{A}_{j+1}^*A)$ yields the desired turnover $A_jA_{j+1}B_j = \tilde{A}_{j+1}\tilde{A}_j\tilde{B}_{j+1}$.

Using fusions and turnovers, we may consider the following sequence of operations, where the operation performed is stated to the right of the expression. The subscripts here indicate the active components of the transformations:

$$\begin{aligned} C_1C_2\cdots C_{n-1} &\to B_1^*C_1C_2C_3\cdots C_{n-1}B_1 \quad \text{(Similarity)} \\ &\to \hat{C}_1C_2C_3C_4\cdots C_{n-1}B_1 \quad \text{(Fusion)} \\ &\to \left(\hat{C}_1C_2B_1\right)C_3C_4\cdots C_{n-1} \quad \text{(Commutativity)} \\ &\to \left(\tilde{B}_2\tilde{C}_1\hat{C}_2\right)C_3C_4\cdots C_{n-1} \quad \text{(Turnover)} \\ &\to \tilde{C}_1\left(\hat{C}_2C_3\tilde{B}_2\right)C_4\cdots C_{n-1} \quad \text{(Similarity + Commutativity)} \\ &\to \tilde{C}_1\left(B_3\tilde{C}_2\hat{C}_3\right)C_4\cdots C_{n-1} \quad \text{(Turnover)} \\ &\to \tilde{C}_1\tilde{C}_2\left(\hat{C}_3C_4B_3\right)\cdots C_{n-1} \quad \text{(Similarity + Commutativity)} \\ &\vdots \\ &\to \tilde{C}_1\tilde{C}_2\cdots\hat{C}_{n-1}B_{n-1} \quad \text{(Similarity)} \\ &\to \tilde{C}_1\tilde{C}_2\cdots\tilde{C}_{n-1} \quad \text{(Fusion).} \end{aligned} \tag{5.5}$$

Importantly, we perform 2 fusions, $n-2$ turnovers, and the similarity/commutativity transformations are symbolic manipulation that does not need to be carried out computationally (but that guarantees the eigenvalues are preserved). By overwriting

| Command | Description |
|---|---|
| `CoreTrans` | Core transformation data-type |
| `apply_ct` | Apply core transformation to matrix/vector |
| `apply_ctchain` | Apply array of core transformations |
| `fuse_ct` | Fuse core transformations |
| `turnover` | Compute turnover |
| `ct_is_diag` | Check if CT is diagonal |
| `get_unitary_naive` | Algorithm 3.1 |
| `get_uuh_naive` | Hessenberg reduction of Algorithm 3.1 |
| `factorize_uuh` | Implementation of Proposition 4.1 |
| `sample_factored_form` | Algorithm 4.1 |
| `get_eig_estimate` | Compute projected Wilkinson shift |
| `get_bulge` | Compute bulge for QR iterate |
| `qr_iterate!` | One QR iterate modifying $C_1 \cdots C_{n-1}$ |
| `uuh_qr!` | Algorithm 5.1 |

Table 1

*A selection of methods implemented in `FastUnitaryEigenvalues.jl`, the culmination of which are `sample_factored_form` and `uuh_qr!`, which implement the fast sampling algorithm Algorithm 4.1 to generate $U = C_1C_2 \cdots C_{n-1}$ with Haar-distributed eigenvalues and the fast QR algorithm Algorithm 5.1 for computing the eigenvalues of this matrix.*

the $C_k$s with $\hat{C}_k$ or $\tilde{C}_k$ and modifying the $B_k$s in place, we do not need more than $O(1)$ storage (in addition to the $O(n)$ storage of the $C_k$) to carry out this process. So one iterate of this process requires only $O(n)$ storage and $O(n)$ flops. After $O(1)$ iterates an eigenvalue will converge and we can deflate and repeat the process on a smaller problem. The resulting algorithm is stated more generally in Algorithm 5.1.

**Algorithm 5.1** Fast unitary QR algorithm

```
Initialize $U = C_1C_2 \cdots C_{n-1}$
while $C_{n-1}$ is not diagonal do
    Compute a shift $\mu$ from trailing $C_{n-2}, C_{n-1}$
    Compute a bulge $B_1^*$ from $\mu, C_1, C_2$
    Fuse $C_1 \leftarrow B_1^* C_1$
    for $j = 1, 2, n-2$ do
        Turnover $C_jC_{j+1}B_j \to B_{j+1}C_jC_{j+1}$
    end for
    Fuse $C_{n-1}B_{n-1}$
end while
Set $\lambda_n \leftarrow C_{n-1}[n, n]$
Modify $C_{n-2} \leftarrow C_{n-2} \cdot \begin{bmatrix} I_{n-2} & \\ & C_{n-1}[n-1, n-1] \end{bmatrix}$
Deflate and repeat on $C_1 \cdots C_{n-2}$
```

## 6. Julia package.

We conclude our discussion by describing the Julia package that we implemented to do this. To install the package, download and install the GitHub repository `Westnickp/FastUnitaryEigenvalues.jl`. In Table 1 we list the main functions provided by this package.

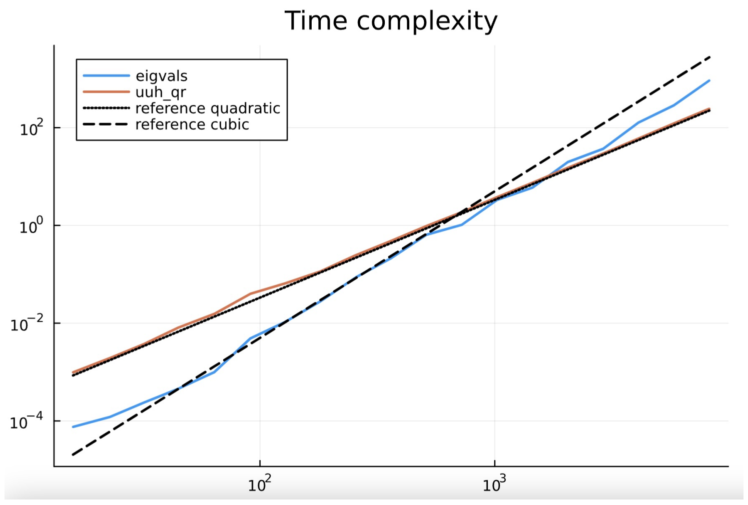

FIG. 1. *Log-log plot of the timings (y-axis) of* `uuh_qr!` *(in red) compared with Julia's built-in* `LinearAlgebra.eigvals` *(in blue) for matrices of size (x-axis) ranging from $n = 100$ to $n = 8000$. In black we plot reference lines for quadratic $O(n^2)$ and cubic $O(n^3)$ complexity. Although our algorithm has the desired quadratic time complexity, there remains code optimization to be done to increase its speed.*

**6.1. Usage.** We indicate here the main usage of the package: first, to sample a unitary matrix according to Algorithm 4.1, and second, to compute the eigenvalues of such a matrix. A minimal example sampling an $n \times n$ unitary upper Hessenberg matrix and computing its eigenvalues in Julia is as follows:

```
using FastUnitaryEigenvalues
function example(n)
  U = sample_factored_form(n)
  λ = uuh_qr!(U)
end
```

**6.2. Experiments.** We conduct some simple experiments to verify the complexity and accuracy of the algorithm. We depict in Figure 1 the timings of `uuh_qr!` compared with Julia's `LinearAlgebra.eigvals` algorithm. We obtain the desired time complexity (quadratic), but the code is highly suboptimal, as indicated by how slow it is for moderately sized matrices.

In Figure 3 we plot the error in the individual eigenvalues when compared to the eigenvalues computed by `LinearAlgebra.eigvals`. We show both the maximal error $E_{max}$ and the average error $E_{ave}$ defined by

$$E_{max} \equiv \max \left|\hat{\lambda}_i - \lambda_i\right|, \quad E_{ave} \equiv \frac{1}{n}\sum_{i=1}^{n}\left|\hat{\lambda}_i - \lambda_i\right|, \tag{6.1}$$

where $\lambda_i$ denotes an eigenvalue computed by `LinearAlgebra` and $\hat{\lambda}_i$ denotes the corresponding eigenvalue computed via `FastUnitaryEigenvalues`. It shows that eigenvalues are being computed to about 13 to 14 digits of accuracy with a moderate growth as the size of the matrix $n$ grows. In machine precision we expect 15 or 16 digits of accuracy regardless of $n$, so this will need to be investigated further.

Finally, we examine the distribution of eigenvalues computed by the function `example(n)` to verify Theorem 2.1. For various values of $n$, we plot the eigenvalues in

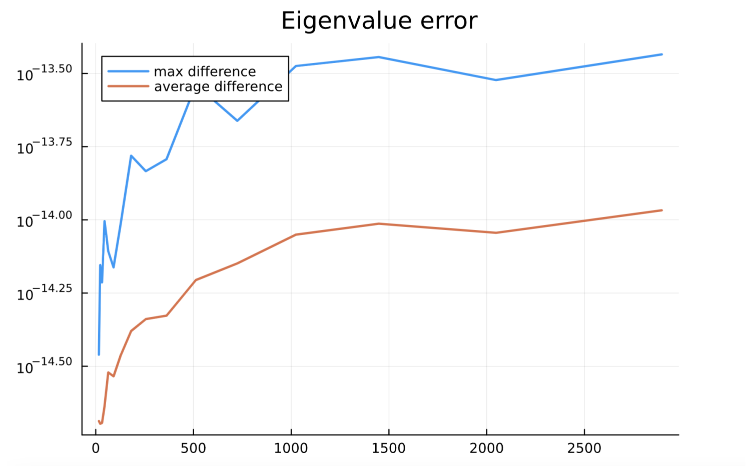

Fig. 2. *We plot on the y-axis the log of error in eigenvalue computations using* `FastUnitaryEigenvalues` *compared with results from* `LinearAlgebra` *for various sized matrices (x-axis). We plot both the maximal error and the average error.*

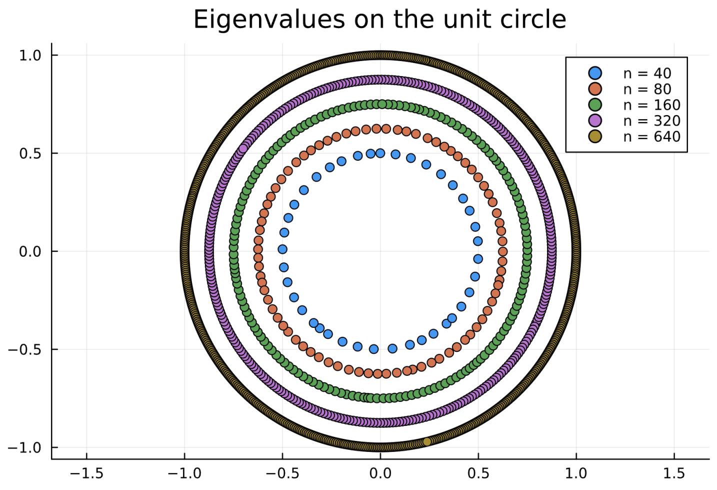

Fig. 3. *Here we show the eigenvalues of randomly sampled unitary matrices of varying size on the unit circle of varying size. The figure shows the ring of eigenvalues for random unitary matrices as sampled by Algorithm 3.2 in the complex plane, where we have varied the magnitude of the eigenvalues in order to show multiple matrix sizes ranging from $n = 40$ eigenvalues to $n = 640$. The eigenvalues distribute themselves approximately uniformly on the unit circle.*

the complex plane, which seem to distribute themselves uniformly as expected on the unit circle; however, more rigorous numerical experiments should be run to verify this result.

**7. Conclusions.** In this report, we reviewed sampling algorithms for unitary matrices. Most notable was Algorithm 4.1, which shows how to sample the Hessenberg-reduced form of unitary matrices such that the eigenvalues are distributed the same as Haar random unitary matrices but the matrix involves only $O(n)$ random variables and is stored in a data-sparse manner. We described how to use this data-sparse representation in a QR eigenvalue algorithm, as described in Algorithm 5.1, and we implemented these algorithms in Julia.

There are several next steps to be taken to improve upon this work. First, the code needs to be optimized. Figure 1 shows the desired quadratic complexity, but it is drastically slower than the built-in eigenvalue routine. In principle, this should not be the case. Fortunately, it should be possible to obtain massive speedups through a more careful implementation of the package. In particular, the code would benefit from (1) explicit typing in the function definitions, (2) further enforcement of in-place operations where possible, and (3) the reduction of high-level linear algebraic operations such as index slicing. Beyond these first steps, it is possible to implement the algorithms using only real arithmetic.

After optimizing the code, there are several functionality extensions to consider. Real orthogonal matrices, and unitary/orthogonal matrices with fixed determinants, can be sampled in an analogous way through a slight modification of Algorithm 4.1 (as is shown in [2]). Moreover, the QR algorithm can be modified from a complex single-shift variant to a real double-shift variant, which uses only real arithmetic for orthogonal matrices [1].

Finally, this work would benefit from more robust experiments examining that the eigenvalues of matrices sampled via Algorithm 4.1 obey the appropriate laws of random unitary matrices; relatedly, `FastUnitaryEigenvalues` should have a more robust set of software tests implemented to verify that functions are behaving as expected.

## REFERENCES

[1] J. L. Aurentz, T. Mach, R. Vandebril, and D. S. Watkins, *Fast and Backward Stable Computation of Roots of Polynomials*, SIAM Journal on Matrix Analysis and Applications, 36 (2015), pp. 942–973, https://doi.org/10.1137/140983434, https://epubs.siam.org/doi/abs/10.1137/140983434 (accessed 2024-10-15). Publisher: Society for Industrial and Applied Mathematics.

[2] M. Fasi and L. Robol, *Sampling the eigenvalues of random orthogonal and unitary matrices*, Linear Algebra and its Applications, 620 (2021), pp. 297–321, https://doi.org/10.1016/j.laa.2021.02.031, https://www.sciencedirect.com/science/article/pii/S0024379521000987 (accessed 2024-10-15).

[3] J. G. Francis, *The QR transformation a unitary analogue to the LR transformation—Part 1*, The Computer Journal, 4 (1961), pp. 265–271, https://academic.oup.com/comjnl/article-abstract/4/3/265/380632 (accessed 2024-10-15). Publisher: Oxford University Press.

[4] J. G. Francis, *The QR transformation—part 2*, The Computer Journal, 4 (1962), pp. 332–345, https://academic.oup.com/comjnl/article-abstract/4/4/332/432033 (accessed 2024-10-15). Publisher: Oxford University Press.

[5] G. H. Golub and C. F. Van Loan, *Matrix computations*, JHU press, 2013.

[6] F. Mezzadri, *How to Generate Random Matrices from the Classical Compact Groups*, 54 (2007).