Eigenvalue algorithms for matrices of quaternions and reduced biquaternions and applications

by Ivan Slapničar, Nevena Jakovčević Stor, Anita Carević and Thaniporn Chaysri from the University of Split, FESB, and Sk. Safique Ahmad and Neha Bhadala from the IIT Indore, Department of Mathematics

at the 9-th European Mathematical Congress, Sevilla, July 15-19, 2024.

This work has been partially supported by the Croatian Science Foundation under the project IP-2020-02-2240 - http://manaa.fesb.unist.hr.

HRZZ Croatian Science Foundation

Aims

- To state basic NLA problems and corresponding (generic) algorithms
- To show how to implement algorithms for quaternions and reduced biquaternions
- To give some interesting examples

Basic LA problems and algorithms

- Solving systems - Gaussian elimination $PA = LU$ *(not generic!)*
- Least squares (also for systems) - QR factorization $A = QR$ or $AP = QR$
  - (Householder) `reflector!()`
  - `reflectorApply!()`
- Eigenvalues
  - reduction to Hessenberg form $X^*AX = H$ using reflectors
  - reduction to Schur form $U^*HU = T$ using pivots and `givens()` rotations
  - Then $Q = XU$ and $Q^*AQ = T$
- Singular values
  - reduction to bidiagonal form $X^*AY = B$ using reflectors
  - bidiagonal SVD $Q^*BR = \Sigma$ using pivots and `givens()` rotations
  - Then $U = XQ$, $V = YR$ and $U^*AV = \Sigma$

Compute the reflector

From Julia's package `LinearAlgebra`, file `generic.jl`

```
# Elementary reflection similar to LAPACK. The reflector is not Hermitian but
# ensures that tridiagonalization of Hermitian matrices become real. See lawn72

@inline function reflector!(x::AbstractVector{T}) where {T}
    require_one_based_indexing(x)
    n = length(x)
    n == 0 && return zero(eltype(x))
    @inbounds begin
        ξ1 = x[1]
        normu = norm(x)
        if iszero(normu)
            return zero(ξ1/normu)
        end
        ν = T(copysign(normu, real(ξ1)))
        ξ1 += ν
        x[1] = -ν
        for i = 2:n
            x[i] /= ξ1
        end
    end
    ξ1/ν
end
```

Apply the reflector - generic

From Julia's package `LinearAlgebra`, file `generic.jl`

"Generic" means that this function can be used for **ANY** number system.

```
#    reflectorApply!(x, τ, A)
#
# Multiplies `A` in-place by a Householder reflection on the left.
# It is equivalent to `A .= (I - conj(τ)*[1; x] * [1; x]')*A`.

@inline function reflectorApply!(x::AbstractVector, τ::Number, A::AbstractVecOrMat)
    require_one_based_indexing(x)
    m, n = size(A, 1), size(A, 2)
    if length(x) != m
        throw(DimensionMismatch(lazy"reflector has length $(length(x)), which must match the first dimension of matr
ix A, $m"))
    end
    m == 0 && return A
    @inbounds for j = 1:n
        Aj, xj = view(A, 2:m, j), view(x, 2:m)
        vAj = conj(τ)*(A[1, j] + dot(xj, Aj))
        A[1, j] -= vAj
        axpy!(-vAj, xj, Aj)
    end
    return A
end
```

"Plain" QR factorization - generic

From Julia's package `LinearAlgebra`, file `qr.jl`

```
function qrfactUnblocked!(A::AbstractMatrix{T}) where {T}
    require_one_based_indexing(A)
    m, n = size(A)
    τ = zeros(T, min(m,n))
    for k = 1:min(m - 1 + !(T<:Real), n)
        x = view(A, k:m, k)
        τk = reflector!(x)
        τ[k] = τk
        reflectorApply!(x, τk, view(A, k:m, k + 1:n))
    end
    QR(A, τ)
end
```

The function `qrfactPivotedUnblocked!(A::AbstractMatrix)` from the same file is generic, too.

Givens rotations

From Julia's package `LinearAlgebra`, file `qr.jl`

```
function givens(f::T, g::T, i1::Integer, i2::Integer) where T
    if i1 == i2
        throw(ArgumentError("Indices must be distinct."))
    end
    c, s, r = givensAlgorithm(f, g)
    if i1 > i2
        s = -conj(s)
        i1, i2 = i2, i1
    end
    Givens(i1, i2, c, s), r
end
```

This function is generic. The only non-generic part is `givensAlgorithm(f, g)`. It must be implemented with care (see BLAS for $\mathbb{R}$ and $\mathbb{C}$).

Quaternions - $\mathbb{Q}$

Quaternions are a non-commutative associative number system that extends complex numbers (a four-dimensional non-commutative algebra and a division ring of numbers), introduced by Hamilton (1853, 1866). Basis elements are $\mathbf{1}$, $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$, satisfying the formula

$$\mathbf{i}^2=\mathbf{j}^2=\mathbf{k}^2=\mathbf{i}\,\mathbf{j}\,\mathbf{k}=-1.$$

Quaternion $q\in\mathbb{Q}$ has the form

$$q=a+b\,\mathbf{i}+c\,\mathbf{j}+d\,\mathbf{k},\quad a,b,c,d,\in\mathbb{R}.$$

Quaternions p and q are **similar** if $p=x^{-1}qx$ for some quaternion x.

The **standard form** of the quaternion q is the unique similar quaternion $q_s=x^{-1}qx=a+\hat{b}\,\mathbf{i}$, where $\|x\|=1$ and $\hat{b}\geq 0$.

(Sudbery,1979) The value of a complex analytic function f at $q\in\mathbb{Q}$, is computed by evaluating the extension of f to the quaternions at q , for example,

$$\sqrt{q}=\pm\left(\sqrt{\frac{\|q\|+a_1}{2}}+\frac{\operatorname{imag}(q)}{\|\operatorname{imag}(q)\|}\sqrt{\frac{\|q\|-a_1}{2}}\right).$$

Basic operations and computation of functions are implemented in the package Quaternions.jl.

Reduced Bi-Quaternions - $\mathbb{Q}_\mathbb{R}$

Reduced Bi-Quaternions are a *commutative* associative number system that extends complex numbers, introduced by Segre (1892). Basis elements are 1, $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$, satisfying formulas

$$\mathbf{i}^2 = -\mathbf{j}^2 = \mathbf{k}^2 = -1, \quad \mathbf{ij} = \mathbf{ji} = \mathbf{k}, \quad \mathbf{jk} = \mathbf{kj} = \mathbf{i}, \quad \mathbf{ki} = \mathbf{ik} = -\mathbf{i}.$$

Basic non-trivial **zero divisors** are $e_1 = \dfrac{1+\mathbf{j}}{2}$ and $e_2 = \dfrac{1-\mathbf{j}}{2}$,

$$e_1 \cdot e_1 = e_1, \quad e_2 \cdot e_2 = e_2, \quad e_1 \cdot e_2 = 0.$$

Any $a = a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k} \in \mathbb{Q}_\mathbb{R}$ is a linear combination of e_1 and e_2 (the **splitting**):

$$a = a_{c1}e_1 + a_{c2}e_2 = [a_1 + a_2 + \mathbf{i}(a_1 + a_3)]e_1 + [a_1 - a_2 + \mathbf{i}(a_1 - a_3)]e_2.$$

The splittings are defined analogously for vectors and matrices.

Basic operations are implemented in the package RBiQuaternions.jl.

Conjugation and norm

For $q \in \mathbb{Q}$, the **conjugation** is defined by

$$\bar{q} = a - b\,\mathbf{i} - c\,\mathbf{j} - d\,\mathbf{k},$$

and the **norm** is defined by (quaternions are a Hilbert space),

$$\bar{q}q = q\bar{q} = |q|^2 = \|q\|^2 = a^2 + b^2 + c^2 + d^2.$$

For $a = a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k} \in \mathbb{Q}_{\mathbb{R}}$, the **conjugation** is defined by

$$\bar{a} = a_0 - a_1\,\mathbf{i} + a_2\,\mathbf{j} - a_3\,\mathbf{k},$$

and the **norm** is defined by

$$|a|^2 = \|a\|^2 = a_0^2 + a_1^2 + a_2^2 + a_3^2.$$

In all cases, the dot product of two vectors, a and y is defined as

$$x \cdot y = x^* y = \sum \overline{x_i}\, y_i.$$

Homomorphisms

Quaternions are homomorphic to $\mathbb{C}^{2\times 2}$:

$$\mathbb{Q} \ni q \to \begin{bmatrix} a + b\,\mathbf{i} & c + d\,\mathbf{i} \\ -c + d\,\mathbf{i} & a - b\,\mathbf{i} \end{bmatrix} \equiv C(q),$$

with eigenvalues q_s and $\bar{q}_s$. It holds

$$C(p+q) = C(p) + C(q), \quad C(pq) = C(p)C(q) \quad C(\bar{p}) = \overline{C(p)}.$$

Reduced bi-quaternions are homomorphic to complex symmetric matrices from $\mathbb{C}^{2\times 2}$ (zero divisors!):

$$\mathbb{Q}_{\mathbb{R}} \ni a \to \begin{bmatrix} a_0 - a_1\,\mathbf{i} & a_2 - a_3\,\mathbf{i} \\ a_2 - a_3\,\mathbf{i} & a_0 - a_1\,\mathbf{i} \end{bmatrix} \equiv C(a),$$

Again

$$C(a+b) = C(a) + C(b), \quad C(ab) = C(a)C(b) \quad C(\bar{a}) = \overline{C(a)}.$$

Eigenvalue decomposition in $\mathbb{Q}^{n\times n}$

Right eigenpairs (λ, x) satisfy

$$Ax = x\lambda, \quad x \neq 0.$$

Usually, x is chosen such that λ is the standard form.

Eigenvalues are invariant under similarity.

> Eigenvalues are **NOT** shift invariant, that is, eigenvalues of the shifted matrix are **NOT** the shifted eigenvalues. (In general, $X^{-1}qX \neq qX^{-1}X = qI$)

If λ is in the standard form, it is invariant under similarity with complex numbers.

A Quaternion QR algorithm

by Angelika Bunse-Gerstner, Ralph Byers, and Volker Mehrmann, Numer. Math 55, 83-95 (1989)

- native functions `reflector!()` and `reflectorApply!()` work for quaternions as is
- native function `hessenberg()` from the package `GenericLinearAlgebra.jl` works as is
- Schur factorization requires quaternion implementation of `givens()`:

```
function givensAlgorithm(f::T, g::T) where T<:Quaternion
    if f==zero(T)
        return zero(T), one(T), abs(g)
    else
        t=g/f
        cs=abs(f)/hypot(f,g)
        sn=t'*cs
        r=cs*f+sn*g
        return cs,sn,r
    end
end
```

The algorithm is derived for general matrices and requires $O(n^3)$ operations. The algorithm is stable.

Computing the Schur decomposition

Given the upper Hessenberg matrix $A \in \mathbb{Q}^{n \times n}$, the method applies complex shift μ to A by using Francis standard double shift on the matrix

$$M = A^2 - (\mu + \bar{\mu})A + \mu\bar{\mu}I$$

and applying it implicitly on A.

If $Ax = x\lambda$, then

$$\begin{aligned} Mx &= (A^2 - (\mu + \bar{\mu})A + \mu\bar{\mu}I)x = x\lambda^2 - x(\mu + \bar{\mu})\lambda + x\mu\bar{\mu} \\ &= x(\lambda^2 - (\mu + \bar{\mu})\lambda + \mu\bar{\mu}) \end{aligned}$$

For the perfect shift, $\mu = \lambda$, it holds

$$\lambda^2 - (\mu + \bar{\mu})\lambda + \mu\bar{\mu} = \lambda^2 - (\lambda + \bar{\lambda})\lambda + \lambda\bar{\lambda} = 0.$$

Details are given in Algorithm 4 in the Appendix of [BGBM89].

Perturbation analysis

We have the following Bauer-Fike type theorem (*Sk. Safique Ahmad, Istkhar Ali, and Ivan Slapničar, Perturbation analysis of matrices over a quaternion division algebra, ETNA, Volume 54, pp. 128-149, 2021.*):

Let $A=X\Lambda X^{-1}$, where $\Lambda=\operatorname{diag}(\lambda_1,\ldots,\lambda_n)$ and λ_i is in standard form. If μ is a standard right eigenvalue of $A+\Delta A$, then

$$\operatorname{dist}(\mu,\Lambda_s(A))=\min_{\lambda_i\in\Lambda_s(A)}\{|\lambda_i-\mu|\}\leq\kappa(X)\|\Delta A\|_2.$$

The residual bound is as follows: let $(\tilde\lambda,\tilde x)$ be the approximate eigenpair of the matrix A, where $\|\tilde x\|_2=1$, and let

$$r=A\tilde x-\tilde x\tilde\lambda,\quad \Delta A=-r\tilde x^*.$$

Then, $(\tilde\lambda,\tilde x)$ is the eigenpair of the matrix $A+\Delta A$ and $\|\Delta A\|_2\leq\|r\|_2$.

Error bounds

An error of the product of two quaternions is bounded as follows (*Joldes, M.; Muller, J. M., Algorithms for manipulating quaternions in floating-point arithmetic. In IEEE 27th Symposium on Computer Arithmetic (ARITH), Portland, OR, USA, 2020, pp. 48-55*)

$$|fl(pq) - pq| \leq (5.75\varepsilon + \varepsilon^2)|p||q|.$$

This implies bound error bound for dot product and matrix product in a usual manner.

Combining it all together, we have the following result: let $(\tilde{\mu}, \tilde{x})$ be the computed eigenpair of the matrix A, where $\tilde{\mu}$ is in the standard form and $\|\tilde{x}\|_2 = 1$. Then

$$\min_{\lambda_i \in \Lambda_s(A)} \{|\lambda_i - \tilde{\mu}|\} \leq \kappa(X)\|r\|_2.$$

Methods for matrices of reduced biquaternions

Many algorithms can be derived from splittings: let $A = A_1 e_1 + A_2 e_2$. Let

$$A_1 = Q_1 T_1 Q_1^*, \quad A_2 = Q_2 T_2 Q_2^*$$

be the respective **complex** Schur factorizations (which always exist). Then

$$A = (Q_1 e_1 + Q_2 e_2)(T_1 e_1 + T_2 e_2)(Q_1^* e_1 + Q_2^* e_2)$$

is the Schur factorization of A. (The proof is easy)

Problem and remedy

Problem

- QR factorization can only be used without pivoting (since the pivoting for A_1 and A_2 might be different)
- Gaussian elimination **cannot** be used since it can run into a non-invertible pivot element.

Remedy

Compute functions `reflector!()` and `givensAlgorithm()` using splittings.

The rest works!

```
function LinearAlgebra.givensAlgorithm(f::T, g::T) where T<:RBiQuaternion
    v=[f;g]
    s=splitc(v)
    g1,r1=LinearAlgebra.givens(s.c1[1],s.c1[2],1,2)
    g2,r2=LinearAlgebra.givens(s.c2[1],s.c2[2],1,2)
    g1.c*e₁+g2.c*e₂, g1.s*e₁+g2.s*e₂,r1*e₁+r2*e₂
end
```

Singular value decomposition

Having adequate functions `reflector!()` and `givensAlgorithm()`, the (generic) functions `bidiagonalize()` and `_svd!()` from the package `GenericLinearAlgebra.jl` readily work.

The latter function has a very nice implementation (see `__svd!(B, U, Vᴴ, tol = tol)`)

For matrices of reduced biquaternions, one can also compute the SVD using splitting!

Pseudoinverse

Having SVD, the Moore-Penrose inverse is defined as usual:

```
function LinearAlgebra.pinv(A::Matrix{T},tol::Real=1.0e-14) where T<:RBiQuaternion
    S=svd(A)
    n=length(S.S)
    Σ=pinv.(S.S)
    return S.V*Diagonal(Σ)*S.U'
end
```

Inner inverse (1-inverse, $AXA = A$)

Inner inverse (1-inverse, $AXA = A$) is defined as (*Adi Ben-Israel and Thomas N.E. Greville, Generalized Inverses - Theory and Applications, Second Edition, Springer-Verlag New York, 2003, Section 1.2*)

Assume $A \in \mathbb{F}^{n\times n}$, where $\mathbb{F} \in \{\mathbb{R}, \mathbb{C}, \mathbb{Q}, \mathbb{Q}_\mathbb{R}\}$, and $\mathrm{rank}(A) = r < n$. Let $U \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} V^*$ be the SVD of A.

Set $\Sigma_1 = \begin{bmatrix} \Sigma_r^{-1} & 0 \\ 0 & M \end{bmatrix}$ where M is non-singular. Let $E = \Sigma_1 U^*$. Then E is non-singular.

Let $V = [V_r \quad V_0]$, where V_r is $n \times r$ part of V. Set $P = [V_r \quad N]$, where N is a $n \times (n-r)$ matrix such that P is non-singular.

$$EAP = \begin{bmatrix} \Sigma_r^{-1} & 0 \\ 0 & M \end{bmatrix} U^* U \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_r^* \\ V_0^* \end{bmatrix} [V_r \quad N] = \begin{bmatrix} I_r & K \\ 0 & 0 \end{bmatrix},$$

where $K = V_r^* N$.

For a rectangular matrix X some of the blocks may be missing, depending on the rank. Usually, M and N can be chosen as random matrices of the appropriate sizes.

Generic code for 1-inverse

```
function inv₁(A::AbstractMatrix{T},tol::Real=1e-12) where T
    m,n=size(A)
    S=svd(A,full=true)
    r=rank(A)
    Σ=zeros(T,m,m)
    for i=1:r
        Σ[i,i]=S.S[i]
    end
    Σ[r+1:m,r+1:m]=randn(T,m-r,m-r)
    E=pinv(Σ)*S.U'
    P=[S.V[:,1:r] randn(T,n,n-r)]
    Ir=zeros(T,n,m)
    for i=1:r
        Ir[i,i]=one(T)
    end
    L=randn(T,n-r,m-r)
    Ir[r+1:n,r+1:m]=L
    return P*Ir*E
end
```

Outer inverse (2-inverse, $XAX = X$)

Predrag S. Stanimirović, Miroslav Ćirić, Igor Stojanović, and Dimitrios Gerontitis, Conditions for Existence, Representations, and Computation of Matrix Generalized Inverses, Complexity Volume 2017, Article ID 6429725

The paper considers real and complex matrices, but most of the results hold for matrices of quaternions and reduced biquaternions.

The codes are generic. Different inverse in obtained by changing B.

```
inv₂(A,B)=B*inv₁(A*B)
```

The inverse X also satisfies $(AX)^* = AX$.

Solving linear equation $BXAB = B$ using (nonlinear) optimization (`NLsolve.jl`)

```
function inv₂nl(A::Matrix{T},B::Matrix{T}) where T
    f!(X)=reinterpret(Float64,B*reinterpret(T,X)*A*B-B)
    X₀=reinterpret(Float64,randn(T,size(B')))
    sol=nlsolve(f!,X₀)
    U=reinterpret(T,sol.zero)
    return B*U
end
```

1-2 inverse

See also *Neha Bhadala, Sk. Safique Ahmad, and Predrag S. Stanimirović, Outer inverses of reduced biquaternion matrices, in preparation*.

Different inverse in obtained by changing C.

```
inv₁₂(A,C)=inv₁(C*A)*C
```

The inverse X also satisfies $(XA)^*=XA$.

Solving $CAXC=C$ using optimization

```
function inv₁₂nl(A::Matrix{T},C::Matrix{T}) where T
    f!(X)=reinterpret(Float64,C*A*reinterpret(T,X)*C-C)
    X₀=reinterpret(Float64,randn(T,size(C')))
    sol=nlsolve(f!,X₀)
    U=reshape(reinterpret(T,sol.zero),size(C'))
    return U*C
end
```

Codes and reference

The Julia codes will be available at https://github.com/ivanslapnicar/MANAA

Papers are being submitted.

Conclusions

- Most parts of basic LA algorithms can be implemented in a generic way.
- The only "non-generic" functions are the computation of Householder reflectors and Givens rotation parameters.
- Isomorphisms to $\mathbb{C}^{2\times 2}$ help in defining conjugation.
- Applications to (some) more difficult problems (like generalized inverses) are straightforward.
- Results can be extended to other number systems (dual numbers).

Thank you!