```
# custom wrappers for BLAS and LAPACK routines, together with some custom
  definitions
using LinearAlgebra: BlasFloat, Char, BlasInt, LAPACK, LAPACKException,
    DimensionMismatch, SingularException, PosDefException, chkstride1, checksquare,
    triu!
using LinearAlgebra.BLAS: @blasfunc, libblas, BlasReal, BlasComplex
using LinearAlgebra.LAPACK: liblapack, chklapackerror

function _one!(A::DenseMatrix)
    Threads.@threads for j = 1:size(A,2)
        @simd for i = 1:size(A,1)
            @inbounds A[i,j] = i == j
        end
    end
    return A
end

# MATRIX factorizations
#----------------------
abstract type FactorizationAlgorithm end
abstract type OrthogonalFactorizationAlgorithm <: FactorizationAlgorithm end

struct QRpos <: OrthogonalFactorizationAlgorithm
end
struct QR <: OrthogonalFactorizationAlgorithm
end
struct QL <: OrthogonalFactorizationAlgorithm
end
struct QLpos <: OrthogonalFactorizationAlgorithm
end
struct LQ <: OrthogonalFactorizationAlgorithm
end
struct LQpos <: OrthogonalFactorizationAlgorithm
end
struct RQ <: OrthogonalFactorizationAlgorithm
end
struct RQpos <: OrthogonalFactorizationAlgorithm
end
struct SVD <: OrthogonalFactorizationAlgorithm
end
struct Polar <: OrthogonalFactorizationAlgorithm
end
struct SDD <: OrthogonalFactorizationAlgorithm # lapack's default divide and
    conquer algorithm
end

Base.adjoint(::QRpos) = LQpos()
Base.adjoint(::QR) = LQ()
Base.adjoint(::LQpos) = QRpos()
Base.adjoint(::LQ) = QR()

Base.adjoint(::QLpos) = RQpos()
Base.adjoint(::QL) = RQ()
Base.adjoint(::RQpos) = QLpos()
```

```
Base.adjoint(::RQ) = QL()

Base.adjoint(alg::Union{SVD,SDD,Polar}) = alg

_safesign(s::Real) = ifelse(s<zero(s), -one(s), +one(s))
_safesign(s::Complex) = ifelse(iszero(s), one(s), s/abs(s))

function _leftorth!(A::StridedMatrix{<:BlasFloat}, alg::Union{QR,QRpos},
atol::Real)
    iszero(atol) || throw(ArgumentError("nonzero atol not supported by $alg"))
    m, n = size(A)
    k = min(m, n)
    A, T = LAPACK.geqrt!(A, min(minimum(size(A)), 36))
    Q = similar(A, m, k)
    for j = 1:k
        for i = 1:m
            Q[i,j] = i == j
        end
    end
    Q = LAPACK.gemqrt!('L', 'N', A, T, Q)
    R = triu!(A[1:k, :])

    if isa(alg, QRpos)
        @inbounds for j = 1:k
            s = _safesign(R[j,j])
            @simd for i = 1:m
                Q[i,j] *= s
            end
        end
        @inbounds for j = size(R,2):-1:1
            for i = 1:min(k,j)
                R[i,j] = R[i,j]*conj(_safesign(R[i,i]))
            end
        end
    end
    return Q, R
end
# TODO: reconsider the following implementation
# Unfortunately, geqrfp seems a bit slower than geqrt in the intermediate region
# around matrix size 100, which is the interesting region. => Investigate and
  maybe fix
# function _leftorth!(A::StridedMatrix{<:BlasFloat})
#     m, n = size(A)
#     A, τ = geqrfp!(A)
#     Q = LAPACK.ormqr!('L','N', A, τ, eye(eltype(A), m, min(m,n)))
#     R = triu!(A[1:min(m,n), :])
#     return Q, R
# end

function _leftorth!(A::StridedMatrix{<:BlasFloat}, alg::Union{QL,QLpos},
atol::Real)
    iszero(atol) || throw(ArgumentError("nonzero atol not supported by $alg"))
    m, n = size(A)
    @assert m >= n
```

```
    nhalf = div(n,2)
    #swap columns in A
    @inbounds for j = 1:nhalf, i = 1:m
        A[i,j], A[i,n+1-j] = A[i,n+1-j], A[i,j]
    end
    Q, R = _leftorth!(A, isa(alg, QL) ? QR() : QRpos() , atol)

    #swap columns in Q
    @inbounds for j = 1:nhalf, i = 1:m
        Q[i,j], Q[i,n+1-j] = Q[i,n+1-j], Q[i,j]
    end
    #swap rows and columns in R
    @inbounds for j = 1:nhalf, i = 1:n
        R[i,j], R[n+1-i,n+1-j] = R[n+1-i,n+1-j], R[i,j]
    end
    if isodd(n)
        j = nhalf+1
        @inbounds for i = 1:nhalf
            R[i,j], R[n+1-i,j] = R[n+1-i,j], R[i,j]
        end
    end
    return Q, R
end

function _leftorth!(A::StridedMatrix{<:BlasFloat}, alg::Union{SVD,SDD,Polar},
atol::Real)
    U, S, V = alg isa SVD ? LAPACK.gesvd!('S', 'S', A) : LAPACK.gesdd!('S', A)
    if isa(alg, Union{SVD, SDD})
        n = count(s-> s .> atol, S)
        if n != length(S)
            return U[:,1:n], lmul!(Diagonal(S[1:n]), V[1:n, :])
        else
            return U, lmul!(Diagonal(S), V)
        end
    else
        iszero(atol) || throw(ArgumentError("nonzero atol not supported by $alg"))
        # TODO: check Lapack to see if we can recycle memory of A
        Q = mul!(A, U, V)
        Sq = map!(sqrt,S,S)
        SqV = lmul!(Diagonal(Sq), V)
        R = SqV'*SqV
        return Q, R
    end
end

function _leftnull!(A::StridedMatrix{<:BlasFloat}, alg::Union{QR,QRpos},
atol::Real)
    iszero(atol) || throw(ArgumentError("nonzero atol not supported by $alg"))
    m, n = size(A)
    m >= n || throw(ArgumentError("no null space if less rows than columns"))

    A, T = LAPACK.geqrt!(A, min(minimum(size(A)), 36))
    N = similar(A, m, max(0, m-n));
```

```
    fill!(N, 0)
    for k = 1:m-n
        N[n+k,k] = 1
    end
    N = LAPACK.gemqrt!('L', 'N', A, T, N)
end

function _leftnull!(A::StridedMatrix{<:BlasFloat}, alg::Union{SVD,SDD}, atol::Real)
    size(A, 2) == 0 && return _one!(similar(A, (size(A,1), size(A,1))))
    U, S, V = alg isa SVD ? LAPACK.gesvd!('A', 'N', A) : LAPACK.gesdd!('A', A)
    indstart = count(>(atol), S) + 1
    return U[:, indstart:end]
end

function _rightorth!(A::StridedMatrix{<:BlasFloat}, alg::Union{LQ,LQpos, RQ,
RQpos},
                     atol::Real)
    iszero(atol) || throw(ArgumentError("nonzero atol not supported by $alg"))
    # TODO: geqrfp seems a bit slower than geqrt in the intermediate region around
    # matrix size 100, which is the interesting region. => Investigate and fix
    m, n = size(A)
    k = min(m,n)
    At = transpose!(similar(A,n,m), A)

    if isa(alg, RQ) || isa(alg, RQpos)
        @assert m <= n

        mhalf = div(m,2)
        # swap columns in At
        @inbounds for j = 1:mhalf, i = 1:n
            At[i,j], At[i,m+1-j] = At[i,m+1-j], At[i,j]
        end
        Qt, Rt = _leftorth!(At, isa(alg, RQ) ? QR() : QRpos(), atol)

        @inbounds for j = 1:mhalf, i = 1:n
            Qt[i,j], Qt[i,m+1-j] = Qt[i,m+1-j], Qt[i,j]
        end
        @inbounds for j = 1:mhalf, i = 1:m
            Rt[i,j], Rt[m+1-i,m+1-j] = Rt[m+1-i,m+1-j], Rt[i,j]
        end
        if isodd(m)
            j = mhalf+1
            @inbounds for i = 1:mhalf
                Rt[i,j], Rt[m+1-i,j] = Rt[m+1-i,j], Rt[i,j]
            end
        end
        Q = transpose!(A, Qt)
        R = transpose!(similar(A, (m,m)), Rt) # TODO: efficient in place
        return R, Q
    else
        Qt, Lt = _leftorth!(At, alg', atol)
        if m > n
            L = transpose!(A, Lt)
            Q = transpose!(similar(A, (n,n)), Qt) # TODO: efficient in place
```

```
        else
            Q = transpose!(A, Qt)
            L = transpose!(similar(A, (m,m)), Lt) # TODO: efficient in place
        end
        return L, Q
    end
end

function _rightorth!(A::StridedMatrix{<:BlasFloat}, alg::Union{SVD,SDD,Polar},
atol::Real)
    U, S, V = alg isa SVD ? LAPACK.gesvd!('S', 'S', A) : LAPACK.gesdd!('S', A)
    if isa(alg, Union{SVD, SDD})
        n = count(s-> s .> atol, S)
        if n != length(S)
            return rmul!(U[:,1:n], Diagonal(S[1:n])), V[1:n,:]
        else
            return rmul!(U, Diagonal(S)), V
        end
    else
        iszero(atol) || throw(ArgumentError("nonzero atol not supported by $alg"))
        Q = mul!(A, U, V)
        Sq = map!(sqrt, S, S)
        USq = rmul!(U, Diagonal(Sq))
        L = USq*USq'
        return L, Q
    end
end

function _rightnull!(A::StridedMatrix{<:BlasFloat}, alg::Union{LQ,LQpos},
atol::Real)
    iszero(atol) || throw(ArgumentError("nonzero atol not supported by $alg"))
    m, n = size(A)
    k = min(m,n)
    At = adjoint!(similar(A,n,m), A)
    At, T = LAPACK.geqrt!(At, min(k, 36))
    N = similar(A, max(n-m,0), n);
    fill!(N, 0)
    for k = 1:n-m
        N[k,m+k] = 1
    end
    N = LAPACK.gemqrt!('R', eltype(At) <: Real ? 'T' : 'C', At, T, N)
end

function _rightnull!(A::StridedMatrix{<:BlasFloat}, alg::Union{SVD,SDD},
atol::Real)
    size(A, 1) == 0 && return _one!(similar(A, (size(A,2), size(A,2))))
    U, S, V = alg isa SVD ? LAPACK.gesvd!('N', 'A', A) : LAPACK.gesdd!('A', A)
    indstart = count(>(atol), S) + 1
    return V[indstart:end, :]
end

function _svd!(A::StridedMatrix{<:BlasFloat}, alg::Union{SVD,SDD})
    U, S, V = alg isa SVD ? LAPACK.gesvd!('S', 'S', A) : LAPACK.gesdd!('S', A)
    return U, S, V
```

```
end

# TODO: override Julia's eig interface
# eig!(A::StridedMatrix{<:BlasFloat}) = LinAlg.LAPACK.gees!('V', A)
# function eig!(A::StridedMatrix{T}; permute::Bool=true, scale::Bool=true) where
  T<:BlasReal
#     n = size(A, 2)
#     n == 0 && return Eigen(zeros(T, 0), zeros(T, 0, 0))
#     issymmetric(A) && return eigfact!(Symmetric(A))
#     A, WR, WI, VL, VR, _ = LAPACK.geevx!(permute ? (scale ? 'B' : 'P') : (scale
    ? 'S' : 'N'), 'N', 'V', 'N', A)
#     evec = zeros(Complex{T}, n, n)
#     j = 1
#     while j <= n
#         if WI[j] == 0
#             evec[:,j] = view(VR, :, j)
#         else
#             for i = 1:n
#                 evec[i,j]   = VR[i,j] + im*VR[i,j+1]
#                 evec[i,j+1] = VR[i,j] - im*VR[i,j+1]
#             end
#             j += 1
#         end
#         j += 1
#     end
#     return Eigen(complex.(WR, WI), evec)
# end
#
# function eigfact!(A::StridedMatrix{T}; permute::Bool=true, scale::Bool=true)
  where T<:BlasComplex
#     n = size(A, 2)
#     n == 0 && return Eigen(zeros(T, 0), zeros(T, 0, 0))
#     ishermitian(A) && return eigfact!(Hermitian(A))
#     return Eigen(LAPACK.geevx!(permute ? (scale ? 'B' : 'P') : (scale ? 'S' :
    'N'), 'N', 'V', 'N', A)[[2,4]]...)
# end
#
#
# eigfact!(A::RealHermSymComplexHerm{<:BlasReal,<:StridedMatrix}) =
  Eigen(LAPACK.syevr!('V', 'A', A.uplo, A.data, 0.0, 0.0, 0, 0, -1.0)...)
#
#
#
# Modified / missing Lapack wrappers
#---------------------------------
# geqrfp!: computes qrpos factorization, missing in Base
geqrfp!(A::StridedMatrix{<:BlasFloat}) = ((m,n) = size(A); geqrfp!(A, similar(A,
min(m, n))))

for (geqrfp, elty, relty) in
    ((:dgeqrfp_,:Float64,:Float64), (:sgeqrfp_,:Float32,:Float32),
(:zgeqrfp_,:ComplexF64,:Float64), (:cgeqrfp_,:ComplexF32,:Float32))
    @eval begin
        function geqrfp!(A::StridedMatrix{$elty}, tau::StridedVector{$elty})
```

```
            chkstride1(A,tau)
            m, n  = size(A)
            if length(tau) != min(m,n)
                throw(DimensionMismatch("tau has length $(length(tau)), but needs
length $(min(m,n))"))
            end
            work  = Vector{$elty}(1)
            lwork = BlasInt(-1)
            info  = Ref{BlasInt}()
            for i = 1:2                 # first call returns lwork as work[1]
                ccall((@blasfunc($geqrfp), liblapack), Nothing,
                      (Ptr{BlasInt}, Ptr{BlasInt}, Ptr{$elty}, Ptr{BlasInt},
                       Ptr{$elty}, Ptr{$elty}, Ptr{BlasInt}, Ptr{BlasInt}),
                      Ref(m), Ref(n), A, Ref(max(1,stride(A,2))), tau, work,
Ref(lwork), info)
                chklapackerror(info[])
                if i == 1
                    lwork = BlasInt(real(work[1]))
                    resize!(work, lwork)
                end
            end
            A, tau
        end
    end
end
```