Problem 1

Write an implementation of SSS matrix construction algorithm, together with a fast SSS matrix-vector multiply. Verify the correctness of the code by writing appropriate test cases.

Solution. Shown below is a sample implementation in Julia.

```
using LinearAlgebra



struct SSSmatrix{Scalar<:Number} <: AbstractMatrix{Scalar}
    N::Int64
    n::Vector{Int64}
    Gpi::Vector{Int64}   # hankel block ranks lower triangular part
    Hpi::Vector{Int64}   # hankel block rank upper triangular part
    # main diagonal
    Di::Vector{Matrix{Scalar}}
    # upper triangular part
    Ui::Vector{Matrix{Scalar}}
    Wi::Vector{Matrix{Scalar}}
    Vi::Vector{Matrix{Scalar}}
    # lower triangular part
    Pi::Vector{Matrix{Scalar}}
    Ri::Vector{Matrix{Scalar}}
    Qi::Vector{Matrix{Scalar}}

    function SSSmatrix{Scalar}(N, n, Gpi, Hpi, Di, Ui, Wi, Vi, Pi, Ri, Qi) where {Scalar<:Number}

        if N < 3
            error("N<3 not supported")
        end

        # check input dimensions
        if length(n) != N  length(Di) != N  length(Ui) != N - 1
           length(Wi) != N - 2  length(Vi) != N - 1  length(Pi) != N - 1
           length(Ri) != N - 2  length(Qi) != N - 1  length(Hpi) != N - 1
           length(Gpi) != N - 1
            error("Input dimensions do not agree")
        end

        # check diagonal matrix dimensions
        for i = 1:N
            if size(Di[i], 1) != n[i]  size(Di[i], 2) != n[i]
                error("Diagonal matrices dimensions mismatch")
            end
        end

        # check upper triangular matrix dimensions
        for i = 1:N-1
            if size(Ui[i], 1) != n[i]  size(Ui[i], 2) != Hpi[i]
                error("Ui matrices dimensions mismatch")
            end
            if size(Vi[i], 1) != n[i+1]  size(Vi[i], 2) != Hpi[i]
```

```
                error("Vi matrices dimensions mismatch")
            end
        end
        for i = 1:N-2
            if size(Wi[i], 1) != Hpi[i]  size(Wi[i], 2) != Hpi[i+1]
                error("Wi matrices dimensions mismatch")
            end
        end

        # check lower triangular matrix dimensions
        for i = 1:N-1
            if size(Pi[i], 1) != n[i+1]  size(Pi[i], 2) != Gpi[i]
                error("Pi matrices dimensions mismatch")
            end
            if size(Qi[i], 1) != n[i]  size(Qi[i], 2) != Gpi[i]
                error("Qi matrices dimensions mismatch")
            end
        end
        for i = 1:N-2
            if size(Ri[i], 1) != Gpi[i+1]  size(Ri[i], 2) != Gpi[i]
                error("Ri translation operators dimensions mismatch")
            end
        end

        new{Scalar}(N, n, Gpi, Hpi,
            map(x -> convert(Matrix{Scalar}, x), Di),
            map(x -> convert(Matrix{Scalar}, x), Ui),
            map(x -> convert(Matrix{Scalar}, x), Wi),
            map(x -> convert(Matrix{Scalar}, x), Vi),
            map(x -> convert(Matrix{Scalar}, x), Pi),
            map(x -> convert(Matrix{Scalar}, x), Ri),
            map(x -> convert(Matrix{Scalar}, x), Qi))


    end

end


Base.:size(A::SSSmatrix) = (sum(A.n), sum(A.n))

function lowrankapprox(B::AbstractArray, threshold::Float64)

    # compute SVD
    U, sigma, V = svd(B)

    # rank
    p = findlast(x -> x > threshold, sigma)
    if p == nothing
        p = 0
    end

    #truncate
```

```
    U = U[:, 1:p]
    sigma = sigma[1:p]
    V = V[:, 1:p]

    return U, sigma, V, p
end


function SSSmatrix(A::Matrix, n::Vector{Int64}, threshold::Float64 = 1E-10)
    # assertions
    @assert size(A, 1) == size(A, 2) "Matrix is not square"
    @assert sum(n) == size(A, 1) "Matrix partition is not valid"


    # some definitions
    N = length(n)
    off = [0; cumsum(n)]

    # Define diagonal matrices
    Di = Matrix{Float64}[]
    for i = 1:N
        push!(Di, A[off[i]+1:off[i]+n[i], off[i]+1:off[i]+n[i]])
    end

    ### upper hankel blocks ###

    Hpi = Int64[]
    Ui = Matrix{Float64}[]
    Wi = Matrix{Float64}[]
    Vi = Matrix{Float64}[]

    # initialize
    Z = zeros(0, sum(n[2:end]))
    Un = zeros(0, 0)
    r = 0
    Leftprev = []
    for i = 1:N-1


        Z = vcat(Z, A[off[i]+1:off[i+1], off[i+1]+1:end])
        U, sigma, V, p = lowrankapprox(Z, threshold)
        Un = [Un * U[1:r, :]
            U[r+1:end, :]]
        r = p

        Left = Un * Diagonal(sigma)

        # add Hpi
        push!(Hpi, p)
        # add Ui
        push!(Ui, Left[off[i]+1:end, :])

        # add Vi
```

```
        push!(Vi, V[1:n[i+1], :])
        if i != 1
            # add Wi
            push!(Wi, Leftprev \ Left[1:off[i], :])
        end

        Z = Diagonal(sigma) * transpose(V[n[i+1]+1:end, :])
        Leftprev = deepcopy(Left)

    end

    ### lower hankel blocks ###

    Gpi = Int64[]
    Pi = Matrix{Float64}[]
    Ri = Matrix{Float64}[]
    Qi = Matrix{Float64}[]

    # initialize
    Z = zeros(sum(n[2:end]), 0)
    Vn = zeros(0, 0)
    r = 0
    Rightprev = []
    for i = 1:N-1

        Z = hcat(Z, A[off[i+1]+1:end, off[i]+1:off[i+1]])
        U, sigma, V, p = lowrankapprox(Z, threshold)
        Vn = [Vn * V[1:r, :]
            V[r+1:end, :]]
        r = p

        Right = Vn * Diagonal(sigma)


        # add Gpi
        push!(Gpi, p)
        # add Pi
        push!(Pi, U[1:n[i+1], :])
        # add Qi
        push!(Qi, Right[off[i]+1:end, :])
        if i != 1
            # add Ri
            push!(Ri, transpose(Rightprev \ Right[1:off[i], :]))
        end

        Z = U[n[i+1]+1:end, :] * Diagonal(sigma)
        Rightprev = deepcopy(Right)
    end


    #construct SSS matrix
    A_SSS = SSSmatrix{Float64}(N, n, Gpi, Hpi, Di, Ui, Wi, Vi, Pi, Ri, Qi)

```

```
    return A_SSS
end


function Base.:*(A::SSSmatrix, x::Vector)
    # assertions
    @assert sum(A.n) == length(x) "Matrix vector dimensions not consistent"

    # break up vector
    off = [0; cumsum(n)]
    xi = Vector[x[off[i]+1:off[i]+A.n[i]] for i = 1:length(A.n)]

    N = A.N

    ### SSS multiply ###
    # Diagonal terms
    bi = [A.Di[i] * xi[i] for i = 1:N]
    # backward flow (upper triangular part)
    g = transpose(A.Vi[N-1]) * xi[N]
    bi[N-1] = bi[N-1] + A.Ui[N-1] * g
    for i = N-2:-1:1
        g = A.Wi[i] * g + transpose(A.Vi[i]) * xi[i+1]
        bi[i] = bi[i] + A.Ui[i] * g
    end
    # forward flow (lower triangular part)
    h = transpose(A.Qi[1]) * xi[1]
    bi[2] = bi[2] + A.Pi[1] * h
    for i = 2:1:N-1
        h = A.Ri[i-1] * h + transpose(A.Qi[i]) * xi[i]
        bi[i+1] = bi[i+1] + A.Pi[i] * h
    end

    #concat back into vector
    b = foldr(vcat, bi)

    return b
end



# Let's test for correctness
n = [5, 5, 5, 5, 5, 5, 5, 5]
N = sum(n)
A = rand(N, N)
A_SSS = SSSmatrix(A, n);
x = rand(N)

A * x ≈ A_SSS * x         # should evelaute to true
```

□