

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.

```

1 using LinearAlgebra
2
3
4
5 struct SSSmatrix{Scalar<:Number} <: AbstractMatrix{Scalar}
6     N::Int64
7     n::Vector{Int64}
8     Gpi::Vector{Int64} # hankel block ranks lower triangular part
9     Hpi::Vector{Int64} # hankel block rank upper triangular part
10    # main diagonal
11    Di::Vector{Matrix{Scalar}}
12    # upper triangular part
13    Ui::Vector{Matrix{Scalar}}
14    Wi::Vector{Matrix{Scalar}}
15    Vi::Vector{Matrix{Scalar}}
16    # lower triangular part
17    Pi::Vector{Matrix{Scalar}}
18    Ri::Vector{Matrix{Scalar}}
19    Qi::Vector{Matrix{Scalar}}
20
21    function SSSmatrix{Scalar}(N, n, Gpi, Hpi, Di, Ui, Wi, Vi, Pi, Ri, Qi) where {Scalar<:Number}
22
23        if N < 3
24            error("N<3 not supported")
25        end
26
27        # check input dimensions
28        if length(n) != N length(Di) != N length(Ui) != N - 1
29            length(Wi) != N - 2 length(Vi) != N - 1 length(Pi) != N - 1
30            length(Ri) != N - 2 length(Qi) != N - 1 length(Hpi) != N - 1
31            length(Gpi) != N - 1
32            error("Input dimensions do not agree")
33        end
34
35        # check diagonal matrix dimensions
36        for i = 1:N
37            if size(Di[i], 1) != n[i] size(Di[i], 2) != n[i]
38                error("Diagonal matrices dimensions mismatch")
39            end
40        end
41
42        # check upper triangular matrix dimensions
43        for i = 1:N-1
44            if size(Ui[i], 1) != n[i] size(Ui[i], 2) != Hpi[i]
45                error("Ui matrices dimensions mismatch")
46            end
47            if size(Vi[i], 1) != n[i+1] size(Vi[i], 2) != Hpi[i]

```

```

48         error("Vi matrices dimensions mismatch")
49     end
50 end
51 for i = 1:N-2
52     if size(Wi[i], 1) != Hpi[i]   size(Wi[i], 2) != Hpi[i+1]
53         error("Wi matrices dimensions mismatch")
54     end
55 end
56
57 # check lower triangular matrix dimensions
58 for i = 1:N-1
59     if size(Pi[i], 1) != n[i+1]   size(Pi[i], 2) != Gpi[i]
60         error("Pi matrices dimensions mismatch")
61     end
62     if size(Qi[i], 1) != n[i]   size(Qi[i], 2) != Gpi[i]
63         error("Qi matrices dimensions mismatch")
64     end
65 end
66 for i = 1:N-2
67     if size(Ri[i], 1) != Gpi[i+1]   size(Ri[i], 2) != Gpi[i]
68         error("Ri translation operators dimensions mismatch")
69     end
70 end
71
72 new{Scalar}(N, n, Gpi, Hpi,
73     map(x -> convert(Matrix{Scalar}, x), Di),
74     map(x -> convert(Matrix{Scalar}, x), Ui),
75     map(x -> convert(Matrix{Scalar}, x), Wi),
76     map(x -> convert(Matrix{Scalar}, x), Vi),
77     map(x -> convert(Matrix{Scalar}, x), Pi),
78     map(x -> convert(Matrix{Scalar}, x), Ri),
79     map(x -> convert(Matrix{Scalar}, x), Qi))
80
81
82 end
83
84 end
85
86
87 Base.size(A::SSSMatrix) = (sum(A.n), sum(A.n))
88
89 function lowrankapprox(B::AbstractArray, threshold::Float64)
90
91     # compute SVD
92     U, sigma, V = svd(B)
93
94     # rank
95     p = findlast(x -> x > threshold, sigma)
96     if p == nothing
97         p = 0
98     end
99
100     #truncate

```

```

101     U = U[:, 1:p]
102     sigma = sigma[1:p]
103     V = V[:, 1:p]
104
105     return U, sigma, V, p
106 end
107
108
109 function SSSmatrix(A::Matrix, n::Vector{Int64}, threshold::Float64 = 1E-10)
110     # assertions
111     @assert size(A, 1) == size(A, 2) "Matrix is not square"
112     @assert sum(n) == size(A, 1) "Matrix partition is not valid"
113
114
115     # some definitions
116     N = length(n)
117     off = [0; cumsum(n)]
118
119     # Define diagonal matrices
120     Di = Matrix{Float64}[]
121     for i = 1:N
122         push!(Di, A[off[i]+1:off[i]+n[i], off[i]+1:off[i]+n[i]])
123     end
124
125     ### upper hankel blocks ###
126
127     Hpi = Int64[]
128     Ui = Matrix{Float64}[]
129     Wi = Matrix{Float64}[]
130     Vi = Matrix{Float64}[]
131
132     # initialize
133     Z = zeros(0, sum(n[2:end]))
134     Un = zeros(0, 0)
135     r = 0
136     Leftprev = []
137     for i = 1:N-1
138
139
140         Z = vcat(Z, A[off[i]+1:off[i+1], off[i+1]+1:end])
141         U, sigma, V, p = lowrankapprox(Z, threshold)
142         Un = [Un * U[1:r, :];
143              U[r+1:end, :]]
144         r = p
145
146         Left = Un * Diagonal(sigma)
147
148         # add Hpi
149         push!(Hpi, p)
150         # add Ui
151         push!(Ui, Left[off[i]+1:end, :])
152
153         # add Vi

```

```

154     push!(Vi, V[1:n[i+1], :])
155     if i != 1
156         # add Wi
157         push!(Wi, Leftprev \ Left[1:off[i], :])
158     end
159
160     Z = Diagonal(sigma) * transpose(V[n[i+1]+1:end, :])
161     Leftprev = deepcopy(Left)
162
163 end
164
165 ### lower hankel blocks ###
166
167 Gpi = Int64[]
168 Pi = Matrix{Float64}[]
169 Ri = Matrix{Float64}[]
170 Qi = Matrix{Float64}[]
171
172 # initialize
173 Z = zeros(sum(n[2:end]), 0)
174 Vn = zeros(0, 0)
175 r = 0
176 Rightprev = []
177 for i = 1:N-1
178
179     Z = hcat(Z, A[off[i+1]+1:end, off[i]+1:off[i+1]])
180     U, sigma, V, p = lowrankapprox(Z, threshold)
181     Vn = [Vn * V[1:r, :];
182          V[r+1:end, :]]
183     r = p
184
185     Right = Vn * Diagonal(sigma)
186
187
188     # add Gpi
189     push!(Gpi, p)
190     # add Pi
191     push!(Pi, U[1:n[i+1], :])
192     # add Qi
193     push!(Qi, Right[off[i]+1:end, :])
194     if i != 1
195         # add Ri
196         push!(Ri, transpose(Rightprev \ Right[1:off[i], :]))
197     end
198
199     Z = U[n[i+1]+1:end, :] * Diagonal(sigma)
200     Rightprev = deepcopy(Right)
201 end
202
203
204 #construct SSS matrix
205 A_SSS = SSSmatrix{Float64}(N, n, Gpi, Hpi, Di, Ui, Wi, Vi, Pi, Ri, Qi)
206

```

```

207     return A_SSS
208 end
209
210
211 function Base.:*(A::SSSmatrix, x::Vector)
212     # assertions
213     @assert sum(A.n) == length(x) "Matrix vector dimensions not consistent"
214
215     # break up vector
216     off = [0; cumsum(n)]
217     xi = Vector{x[off[i]+1:off[i]+A.n[i]] for i = 1:length(A.n)}
218
219     N = A.N
220
221     ### SSS multiply ###
222     # Diagonal terms
223     bi = [A.Di[i] * xi[i] for i = 1:N]
224     # backward flow (upper triangular part)
225     g = transpose(A.Vi[N-1]) * xi[N]
226     bi[N-1] = bi[N-1] + A.Ui[N-1] * g
227     for i = N-2:-1:1
228         g = A.Wi[i] * g + transpose(A.Vi[i]) * xi[i+1]
229         bi[i] = bi[i] + A.Ui[i] * g
230     end
231     # forward flow (lower triangular part)
232     h = transpose(A.Qi[1]) * xi[1]
233     bi[2] = bi[2] + A.Pi[1] * h
234     for i = 2:1:N-1
235         h = A.Ri[i-1] * h + transpose(A.Qi[i]) * xi[i]
236         bi[i+1] = bi[i+1] + A.Pi[i] * h
237     end
238
239     #concat back into vector
240     b = foldr(vcat, bi)
241
242     return b
243 end
244
245
246
247 # Let's test for correctness
248 n = [5, 5, 5, 5, 5, 5, 5, 5, 5]
249 N = sum(n)
250 A = rand(N, N)
251 A_SSS = SSSmatrix(A, n);
252 x = rand(N)
253
254 A * x ≈ A_SSS * x      # should evaluate to true

```

□