Ex2/HermiteBandMatrix.h

```cpp
#include <iostream>
#include <complex>
#include <type_traits>
#include <iomanip>
#include "SCmatrix.h"

namespace SC
{
    template <typename T = double>
    class HermiteBandMatrix : public LinearOperator<T>
    {
    private:
        int n;   // nxn-Matrix
        int b;   // Bandwidth
        T *data; // Bandvalues
        // int alloc_size;

    public:
        using LinearOperator<T>::height;
        using LinearOperator<T>::width;

        /// @brief Constructor
        /// @param n Dimension of n*n-square matrix
        /// @param b Width of band
        HermiteBandMatrix(int n, int b) : n(n), b(b), data(nullptr) //, alloc_size(0)
        {
            int num_elem = n * (n + 1) / 2 - (n - b) * (n - b + 1) / 2; // Maximum allowed storage space
            data = new T[num_elem];
            // alloc_size = num_elem;
            this->height = n;
            this->width = n;
        }

        /// @brief Destructor
        ~HermiteBandMatrix()
        {
            delete[] data;
        }

        /// @brief Copy-Constructor
        /// @param other Object to copy
        HermiteBandMatrix(const HermiteBandMatrix<T> &other) : n(other.n), b(other.b), data(nullptr)
        {
            this->height = other.height;
            this->width = other.width;

            int num_elem = n * (n + 1) / 2 - (n - b) * (n - b + 1) / 2; // Maximum allowed storage space
            data = new T[num_elem];
```

```
49
50              for (int i = 0; i < num_elem; ++i)
51              {
52                  data[i] = other.data[i];
53              }
54          }
55
56          /// @brief Copy-Constructor for disabling operation
57          // HermiteBandMatrix(const HermiteBandMatrix<T>&) = delete;
58
59          /// @brief Sets the given value in matrix at (i, j)
60          /// @param i Row index
61          /// @param j Column index
62          /// @param val Value to store at position
63          void Set(int i, int j, T val)
64          {
65
66              int idx = index(i, j);
67              if (idx < 0)
68              {
69  #ifndef NDEBUG
70                  throw std::out_of_range("Index exceeds matrix dimensions");
71  #endif
72              }
73
74              if (is_swapped(i, j))
75              {
76                  data[idx] = Conjugate(val);
77              }
78              else
79              {
80                  data[idx] = val;
81              }
82          }
83
84          /// @brief Checks if given coordinate (i, j) is below the diagonal
85          /// @param i Row index
86          /// @param j Column index
87          /// @return Boolean if index is below diagonal
88          bool is_swapped(int i, int j) const
89          {
90              return j < i;
91          }
92
93          /// @brief Calculates the list index of given matrix coordinates (i, j)
94          /// @param i Row index
95          /// @param j Column index
96          /// @return Index in flat data list
97          int index(int i, int j) const
98          {
99              // Check if index is out of bound
100             if (i >= n || i < 0 || j >= n || j < 0)
101             {
```

```
102                 return -1;
103             }
104 
105             // Da obere dreiecksstruktur betrachtet
106             if (is_swapped(i, j))
107                 std::swap(i, j);
108 
109             int idx = 0;
110             // n - b = Anzahl der "ganzen" Zeilen
111             if (i <= n - b)
112             {
113                 idx = i * b + (j - i);
114             }
115             else
116             {
117                 int last_full_idx = (n - b + 1) * b;
118                 int rel_idx = 0;
119                 // k = relativer (zeilen) laufindex
120                 for (int k = n - b + 1; k < i; k++)
121                 {
122                     rel_idx += (n - k);
123                 }
124                 rel_idx += (j - i);
125                 idx = last_full_idx + rel_idx;
126             }
127             return idx;
128         }
129 
130         /// @brief Operator for reading data from matrix
131         /// @param i Row index
132         /// @param j Column index
133         /// @return Entry of matrix at given (i, j)
134         T operator()(int i, int j) const
135         {
136             // b inkludiert diagonale -> (b - 1)
137             if (j < i - b + 1 || j > i + b - 1)
138             {
139                 return T(0);
140             }
141 
142             int idx = index(i, j);
143             // Check if index is out of bounds
144             if (idx < 0)
145             {
146 #ifndef NDEBUG
147                 throw std::out_of_range("Index exceeds matrix dimensions");
148 #endif
149                 return T(0);
150             }
151             if (is_swapped(i, j))
152             {
153                 return Conjugate(data[idx]);
154             }
```

```
155
156             return data[idx];
157         }
158
159         /// @brief Calculates the matrix product
160         /// @param a Applied vector
161         /// @param r Resulting vector
162         /// @param factor Linear scaling factor; Default = 1
163         virtual void Apply(const Vector<T> &a, Vector<T> &r, T factor = 1.) const override
164         {
165             for (int row = 0; row < this->height; row++)
166             {
167                 int b_right = this->width - row;
168                 int b_left = row;
169                 if (row > b - 1)
170                 {
171                     b_left = b - 1;
172                 }
173                 if (row < this->width - b)
174                 {
175                     b_right = b;
176                 }
177
178                 r(row) = 0;
179                 for (int col = row - b_left; col < row + b_right; col++)
180                 {
181                     r(row) += operator()(row, col) * a(col);
182                 }
183                 r(row) *= factor;
184             }
185         }
186
187         /// @brief Calculates the the matrix product of A.T*x
188         /// @param a Applied vector
189         /// @param r Resulting vector
190         /// @param factor Linear scaling factor; Default = 1
191         virtual void ApplyT(const Vector<T> &a, Vector<T> &r, T factor = 1.) const
    override
192         {
193             for (int row = 0; row < this->height; row++)
194             {
195                 int b_right = this->width - row;
196                 int b_left = row;
197                 if (row > b - 1)
198                 {
199                     b_left = b - 1;
200                 }
201                 if (row < this->width - b)
202                 {
203                     b_right = b;
204                 }
205
206                 r(row) = 0;
```

```
207                 for (int col = row - b_left; col < row + b_right; col++)
208                 {
209                     r(row) += operator()(col, row) * a(col);
210                 }
211                 r(row) *= factor;
212             }
213         }
214 
215         /// @brief Calculates the matrix hermitian product with a
216         /// @param a Applied vector
217         /// @param result Resulting vector of A.H*x
218         /// @param factor Linear scaling factor; Default = 1
219         virtual void ApplyH(const Vector<T> &a, Vector<T> &result, T factor = 1.) const
    override
220         {
221             // Hermitesche Matrix ist gleich ihrer adjungierten (transponiert-
    konjugierten) Matrix
222             Apply(a, result, factor);
223         }
224 
225         /// @brief Prints the calling hermite band matrix
226         /// @param os Output stream
227         virtual void Print(std::ostream &os) const
228         {
229             os << "[HermiteBandMatrix, size " << this->height << " x " << this->width <<
    ", bandwidth " << b << "]\n";
230             for (int row = 0; row < this->height; row++)
231             {
232                 os << "|";
233                 for (int col = 0; col < this->width; col++)
234                 {
235                     os << std::setw(8) << (*this)(row, col) << std::setw(8);
236                 }
237                 os << std::setw(4) << "|\n";
238             }
239         }
240     };
241 }
242 
```