Ex2/A3.cpp

```cpp
#include <stdio.h>
#include "HermiteBandMatrix.h"
#include "SCvector.h"
#include <iostream>
#include <fstream>

/// @brief Generates a .csv file for given u and x vector
/// @param filename Name of output file with extension embedded
/// @param u First output vector
/// @param x Second output vector
void write_to_csv(const std::string &filename, const SC::Vector<double> &u, const SC::Vector<double> &x)
{
    // Check dimensions of vectors
    if (u.Size() != x.Size())
    {
#ifndef NDEBUG
        throw std::out_of_range("Error: Vectors u and x must have the same size.");
#endif
        std::cerr << "Error: Vectors u and x must have the same size." << std::endl;
        return;
    }

    std::ofstream file(filename);

    if (!file.is_open())
    {
#ifndef NDEBUG
        throw std::out_of_range("Error opening file for writing: ");
#endif
        std::cerr << "Error opening file for writing: " << filename << std::endl;
        return;
    }

    // Write headers
    file << "x,u\n";

    // Add values for -1 index as stated in task for visualisation
    file << 0 << "," << 0 << "\n";

    for (size_t i = 0; i < u.Size(); ++i)
    {
        file << x(i) << "," << u(i) << "\n";
    }

    // Add values for n index as stated in task for visualisation
    file << 1 << "," << 0 << "\n";

    file.close();
    std::cout << "Data written to " << filename << std::endl;
}
```

```cpp

/// @brief Creates evenly spaced numbers in the interval [0, 1]
/// @param n Number of samples to generate
/// @return Vector of equally spaced samples in the closed interval [0, 1]
SC::Vector<double> linspace(int n)
{
    double ne = n + 1;
    SC::Vector<double> points(n);
    for (int i = 0; i < n; i++)
    {
        points(i) = (i + 1) / double(ne);
    }
    return points;
}

/// @brief Calculates the sourceterms
/// @param x Value to calculate source of
/// @return Sourceterm of given x
int f(double x)
{
    if (x < .5)
    {
        return 0;
    }
    return 1;
}

int main()
{
    int k = 10;
    int n = 100;
    int b_ = 2;
    int numIter = 20000;

    double h = 1 / double(n + 1);

    SC::Vector<double> x_i = linspace(n);
    // x_i.Print(std::cout);
    // std::cout << "\n";

    // Definition von A
    SC::HermiteBandMatrix<double> A(n, b_);
    for (int row = 0; row < n; row++)
    {
        A.Set(row, row, 2 * k / (h * h)); // diag
        if (row > 0)
        {
            A.Set(row - 1, row, -k / (h * h)); // neben
        }
    }
    // A.Print(std::cout);

    // rechte-Seite-Vektor
```

```cpp
    SC::Vector<double> b(n);
    for (int row = 0; row < n; row++)
    {
        b(row) = f(x_i(row));
    }
    // b.Print(std::cout);
    // std::cout << "\n";

    // Lösungsvektor
    SC::Vector<double> u(n);
    u.SetAll(0);

    // Residiuumsvektor
    SC::Vector<double> r(n);
    r.SetAll(1);

    // Vektor für Zwischenergebnisse
    // SC::Vector<double> tmp(n);
    // tmp.SetAll(0);

    // Dämpfungsfaktor
    double theta = h * h / (2.0 * k);

    int i = 0;
    while (i < numIter && r.Norm() > 1e-6)
    {
        // Gl. 8
        // A.Apply(u, tmp);
        // r += b;
        // r -= tmp;
        A.Apply(u, r, -1.);
        r.Add(b);

        // Gl. 9
        // tmp = r;
        // tmp.Mult(theta);
        // u += tmp;
        u.AddMultiple(theta, r);
        i++;
    }

    // Output
    std::cout << "n = " << n << "\n";
    std::cout << "Max. iterations: " << numIter << "\n";
    std::cout << "Iterations needed: " << i << "\n";
    std::cout << "|r| = " << r.Norm() << "\n\n";

    // Constructiong filename and writing to csv
    std::ostringstream filename_stream;
    filename_stream << "A3_n_" << n << ".csv";
    std::string filename = filename_stream.str();
    write_to_csv(filename, u, x_i);
}
```