Laboratorium 9

Artem Buhera
GĆ01 135678

16.05.2021

W danym laboratorium musieliśmy zaimplementować program do rozwiązywania równania różniczkowego zwyczajnego drugiego rzędu

$$\frac{d^2U(x)}{dx^2} - 4U(x) = 0 \quad \text{dla } 0 \le x \le 1$$

z warunkami brzegowymi $U(0) = 1$, $U(1) = 0$, używając trzypunktowej dyskretyzacji konwencjonalnej oraz dyskretyzacji Numerowa.

Z ogólnej postaci odczytujemy:

$$p(x)y''(x) + q(x)y'(x) + r(x)y(x) + s(x) = 0$$
$$p(x) = 1,\ q(x) = 0,\ r(x) = -4,\ s(x) = -x$$

Z warunków brzegowych też dobieramy odpowiednie współczynniki:

$$\begin{cases} \alpha\, y'(0) + \beta\, y(0) + \gamma = 0, \\ \phi\, y'(1) + \psi\, y(1) + \theta = 0 \end{cases} \Leftrightarrow \begin{cases} \alpha\cdot 0 + \beta\cdot 1 + \gamma = 0, \\ \phi\cdot 0 + \psi\cdot 0 + \theta = 0 \end{cases} \Rightarrow \begin{cases} \alpha = 0,\ \beta = 1,\ \gamma = -1, \\ \phi = 0,\ \psi = 1,\ \theta = 0 \end{cases}$$

Rozwiązania przy użyciu dyskretyzacji konwencjonalnej znajdziemy po rozwiązaniu układu równań, korzystając się z algorytmu Thomasa, gdzie h - krok siatki:

$$\begin{pmatrix} \beta - \frac{\alpha}{h} & \frac{\alpha}{h} & & \\ \ddots & \ddots & \ddots & \\ \frac{p(x_i)}{h^2} - \frac{q(x_i)}{2h} & r(x_i) - \frac{q(x_i)}{h^2} & \frac{p(x_i)}{h^2} + \frac{q(x_i)}{2h} & \\ & \ddots & \ddots & \ddots \\ & & -\frac{\phi}{h} & \frac{\phi}{h} + \psi \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} -\gamma \\ -s(x_1) \\ \vdots \\ -s(x_{N-1}) \\ -\theta \end{pmatrix}$$

Natomiast przy użyciu dyskretyzacji Numerowa:

$$\begin{pmatrix} \beta - \frac{\alpha}{h} & \frac{\alpha}{h} & & \\ \ddots & \ddots & \ddots & \\ \frac{p(x_i)}{h^2} - \frac{q(x_i)}{2h} + \frac{1}{12}r(x_i) & \frac{10}{12}r(x_i) - \frac{q(x_i)}{h^2} & \frac{p(x_i)}{h^2} + \frac{q(x_i)}{2h} + \frac{1}{12}r(x_i) & \\ & \ddots & \ddots & \ddots \\ & & -\frac{\phi}{h} & \frac{\phi}{h} + \psi \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} -\gamma \\ -s(x_1) \\ \vdots \\ -s(x_{N-1}) \\ -\theta \end{pmatrix}$$

Kod programu:

```
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector.h>

using namespace std;


double p(double x) { return  1.0; }
double q(double x) { return  0.0; }
double r(double x) { return -4.0; }
double s(double x) { return -x;   }

double Alpha = 0.0, Beta = 1.0, Gamma = -1.0, Phi = 0.0, Psi = 1.0, Theta = 0.0;
double left_bound = 0.0, right_bound = 1.0;


double f_exact(double x) {
    return (exp(2.0 - 2.0*x) - 4.0*exp(4.0 - 2.0*x) + 4.0*exp(2.0*x) - exp(2.0 + 2.0*x)
             - x + x*exp(4.0)) / (4.0 - 4.0*exp(4.0));
}


void transformThomas(Vector& u, Vector& d, Vector& l, Vector &b) {
    for (int i = 1; i < u.N; i++) {
        d[i] -= l[i-1] / d[i-1] * u[i-1];
        b[i] -= l[i-1] / d[i-1] * b[i-1];
    }
}

Vector solveThomas(Vector &u, Vector &d, Vector &b) {
    int n = u.N;
    Vector x(n, 0);

    x[n-1] = b[n-1] / d[n-1];
    for (int i = n-2; i ⩾ 0; i--) {
        x[i] = (b[i] - u[i] * x[i+1]) / d[i];
    }

    return x;
}
```

```cpp
double conventional(double h, bool save=false) {
    int n = (int) ceil((right_bound - left_bound) / h) + 1;
    Vector u(n, 0), d(n, 0), l(n, 0), b(n, 0), y;

    double x_0 = left_bound, x_i, y_i, h_square = h * h;
    double maxError = 0.0, currentError;

    d[0] = Beta - Alpha / h;
    u[0] = Alpha / h;
    b[0] = -Gamma;

    for (int i = 1; i < n - 1; i++) {
        x_i = x_0 + h * i;
        h_square = h * h;

        l[i-1] = p(x_i) / h_square - q(x_i) / (2.0 * h);
        d[i]   = r(x_i) - 2.0 * p(x_i) / h_square;
        u[i]   = p(x_i) / h_square + q(x_i) / (2.0 * h);
        b[i]   = -s(x_i);
    }
    l[n-2] = -Phi / h;
    d[n-1] =  Phi / h + Psi;
    b[n-1] = -Theta;

    transformThomas(u, d, l, b);
    y = solveThomas(u, d, b);

    ofstream out;
    if (save) {
        out.open("../lab9/conventional.csv");
        out << "h,conventional" << endl;
    }

    for (int i = 0; i < n; i++) {
        x_i = x_0 + h * i;
        y_i = y[i];

        currentError = abs(y_i - f_exact(x_i));
        if (maxError < currentError)
            maxError = currentError;

        if (save) {
            out << x_i << "," << y_i << endl;
        }
    }

    out.close();
    return maxError;
}
```

```
double numerov(double h, bool save=false) {
    int n = (int) ceil((right_bound - left_bound) / h) + 1;
    Vector u(n, 0), d(n, 0), l(n, 0), b(n, 0), y;

    double x_0 = left_bound, x_i, y_i, h_square = h * h;
    double maxError = 0.0, currentError;

    d[0] = Beta - Alpha / h;
    u[0] = Alpha / h;
    b[0] = -Gamma;

    for (int i = 1; i < n - 1; i++) {
        x_i = x_0 + h * i;

        l[i-1] = p(x_i) / h_square - q(x_i) / (2.0 * h) + r(x_i) / 12.0;
        d[i]   = 10.0 / 12.0 * r(x_i) - 2.0 * p(x_i) / h_square;
        u[i]   = p(x_i) / h_square - q(x_i) / (2.0 * h) + r(x_i) / 12.0;
        b[i]   = -s(x_i);
    }

    l[n-2] = -Phi / h;
    d[n-1] =  Phi / h + Psi;
    b[n-1] = -Theta;

    transformThomas(u, d, l, b);
    y   = solveThomas(u, d, b);

    ofstream out;
    if (save) {
        out.open("../lab9/numerov.csv");
        out << "h,Numerov" << endl;
    }

    for (int i = 0; i < n; i++) {
        x_i = x_0 + h * i;
        y_i = y[i];

        currentError = abs(y_i - f_exact(x_i));
        if (maxError < currentError)
            maxError = currentError;

        if (save) {
            out << x_i << "," << y_i << endl;
        }
    }

    out.close();
    return maxError;
}
```

```cpp
double order(double h_i, double h_j, double y_i, double y_j) {
    return (y_i - y_j) / (h_i - h_j);
}

int main() {
    double h = 0.01;
    conventional(h, true);
    numerov(h, true);

    double conventionalError, numerovError;
    double h_0, conventional_0, numerov_0,
           h_5, conventional_5, numerov_5;

    ofstream errors;
    errors.open("../lab9/errors.csv");
    errors << "h,Numerov,conventional" << endl;

    int count = 0;
    h = 0.125;
    while (h > 1e-6) {
        conventionalError = conventional(h);
        numerovError = numerov(h);
        errors << log10(h) << ","
               << log10(conventionalError) << ","
               << log10(numerovError) << endl;

        if (count == 0) {
            h_0 = log10(h);
            conventional_0 = log10(conventionalError);
            numerov_0 = log10(numerovError);
        } else if (count == 5) {
            h_5 = log10(h);
            conventional_5 = log10(conventionalError);
            numerov_5 = log10(numerovError);
        }

        count++;

        h /= 2.0;
    }

    cout << "rząd metody z dyskretyzacją trzypunktową konwencjonalną: "
         << order(h_0, h_5, conventional_0, conventional_5) << endl;
    cout << "rząd metody z dyskretyzacją Numerowa: "
         << order(h_0, h_5, numerov_0, numerov_5) << endl;

    errors.close();
}
```

Kod vector.h:

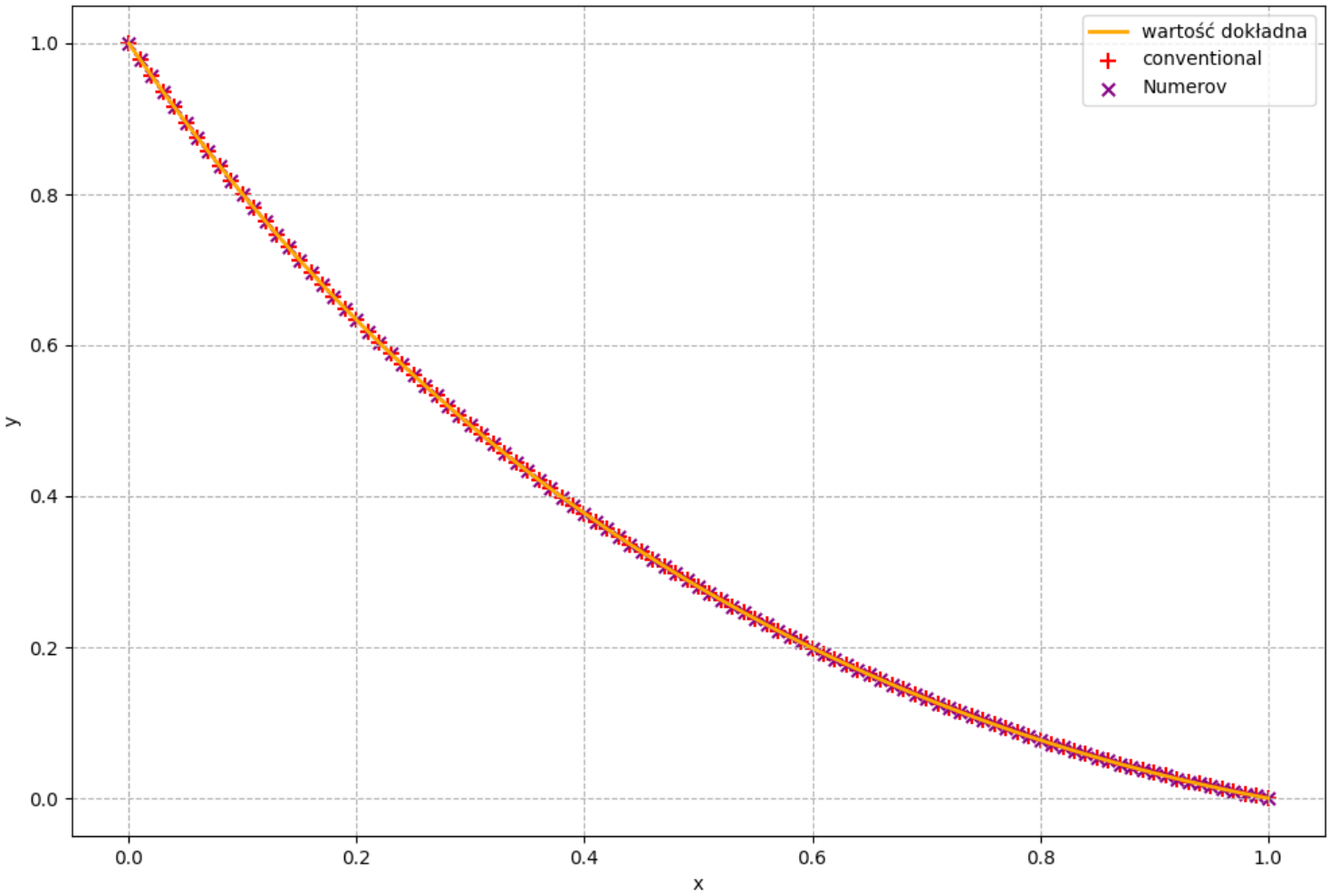
```
#include <vector>
#include <algorithm>
#include <ostream>
#include <iostream>
#include <iomanip>

#ifndef VECTOR_H
#define VECTOR_H

class Vector: public std::vector<double> {
public:
    int N;
    using vector<double>::vector;

    Vector(std::initializer_list<double> input) :
        N(input.size()), vector<double>::vector(input) {};
    Vector(int size, double value) : N(size), Vector::vector(size, value) {}
    Vector(int size) : N(size) {
        this→reserve(N);
    }

    friend Vector operator-(const Vector &left, const Vector &right);
    friend Vector operator+(Vector &left, const Vector &right);

    Vector operator*(double scalar) {
        Vector result(N);
        for(int i = 0; i < N; ++i)
            result[i] = (*this)[i] * scalar;
        return result;
    }

    void print(int width=8) {
        for (auto value : (*this))
            std::cout << std::setw(width) << value << " ";
        std::cout << std::endl;
    }

    void print(const std::string &text, int width=10) {
        std::cout << text << std::endl;
        print(width);
    }

    friend std::ostream& operator<<(std::ostream &os, const Vector &vect);

    double normMax() {
        double maximum = std::abs((*this)[0]);
        for (int i = 1; i < N; i++) {
            double current;
            if ((current = std::abs((*this)[i])) > maximum) {
                maximum = current;
            }
        }
        return maximum;
    }
};
```

```
std::ostream &operator<<(std::ostream &os, const Vector &vect) {
    os << '[';
    for (double value : vect)
        os << std::setw(11) << std::setprecision(8) << value << ", ";
    os << "\b\b]";

    return os;
}

Vector operator-(const Vector &left, const Vector &right) {
    int n = left.N;
    Vector result(n);
    for(int i = 0; i < n; i++)
        result[i] = left[i] - right[i];
    return result;
}

Vector operator+(Vector &left, const Vector &right) {
    int n = left.N;
    Vector result(n);
    for(int i = 0; i < n; i++)
        result[i] = left[i] + right[i];
    return result;
}

#endif
```

W wyniku działania programu otrzymaliśmy następujące wykresy:

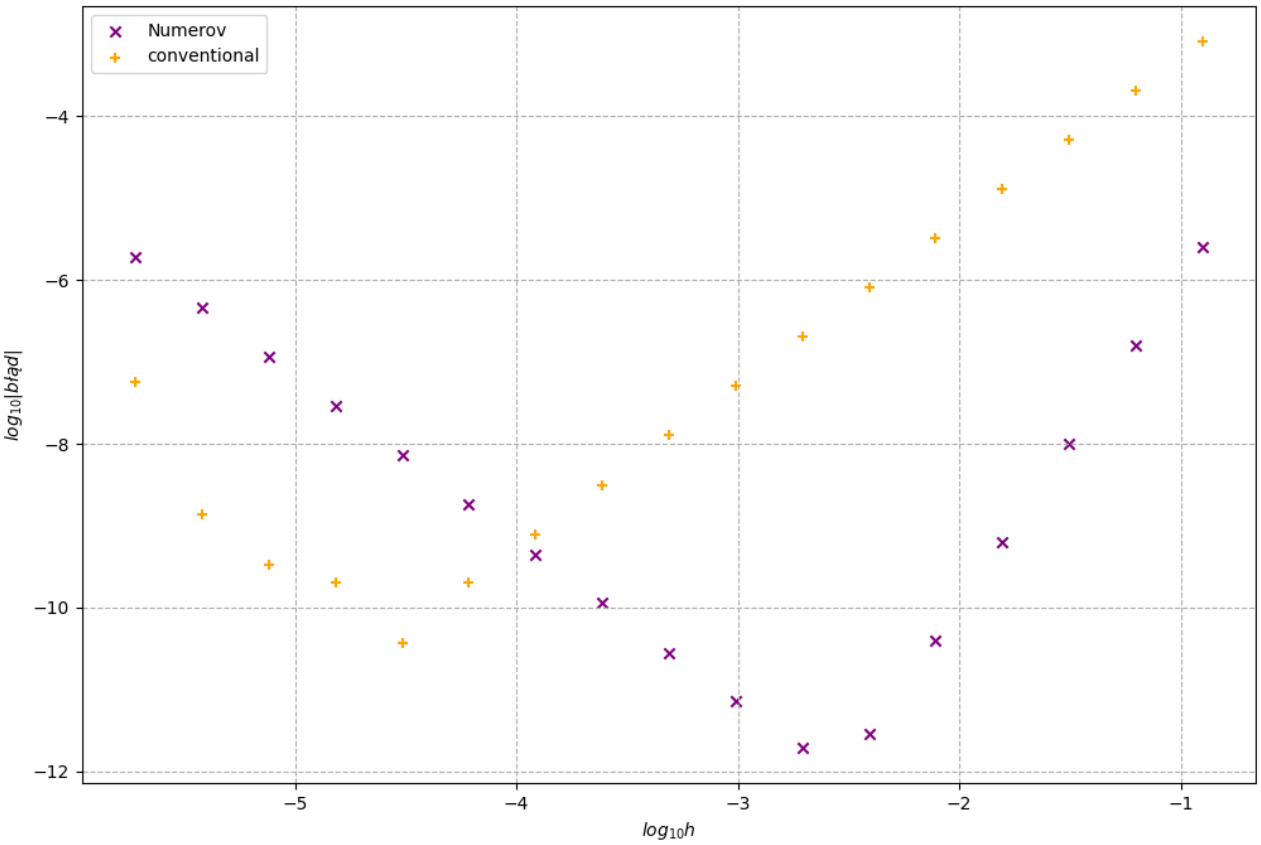
Wykres metod dyskretyzacji trzypunktowej konwencjonalnej, Numerowa oraz funkcji analitycznej

$$U(x) = \frac{e^{2-2x} - 4e^{4-2x} + 4e^{2x} - e^{2+2x} - x + xe^4}{4 - 4e^4}$$



Dla kroku h = 0.01 obie metody dają dobre wyniki zbliżone do dokładnych.

Zlogarytmowany wykres zależności błędów maksymalnych dla obu metod od kroku siatki *h*:



Z wykresu oraz z obliczeń na podstawie otrzymanych wartości możemy zaobserwować zgodność z teoretycznymi rzędami dokładności:

- 2 dla dyskretyzacji konwencjonalnej
- 4 dla dyskretyzacji Numerowa

Wynik działania programu:

```
rząd dla dyskretyzacji konwencjonalnej: 1.99658
rząd dla dyskretyzacji Numerowa: 3.95029
```

Także możemy zauważyć, że poniżej $h \approx 10^{-2.5}$ dla dyskretyzacji Numerowa oraz $h \approx 10^{-4.5}$ dokładność obliczeń zaczyna rosnąć - jest to spowodowane błędami maszynowymi.