Metody numeryczne

Wojciech Chrobak

27 grudnia 2017

Zadanie 5 - obowiązkowe

Metoda Romberga

Całkowanie numeryczne za pomocą metody Romberga generuje tablicę całek:

$$\begin{array}{lllll}
R_{0,0} & & & & \\
R_{1,0} & R_{1,1} & & & \\
R_{2,0} & R_{2,1} & R_{2,2} & & \\
\vdots & \vdots & \vdots & \ddots & \\
R_{n,0} & R_{n,1} & R_{n,2} & \cdots & R_{n,n}
\end{array}$$

Gdzie $R_{n,k}$ z pierwszej kolumny określone są następująco:

$$R_{0,0} = \frac{1}{2}f(a) + \frac{1}{2}f(b)$$

oraz

$$R_{n,0} = \frac{1}{2}R_{n-1,0} + h_n \sum_{k=1}^{2^{n-1}} f(a + (2k-1)h_n)$$

z krokiem całkowania:

$$h_n = \frac{b-a}{2^n}$$

Elementy w kolejnych kolumnach liczymy korzystając ze wzoru:

$$R_{n,k} = \frac{4^k R_{n,k-1} - R_{n-1,k-1}}{4^k - 1}$$

Algorytm kończymy gdy dwa elementy na diagonali $R_{i,i}$ i $R_{i-1,i-1}$ są dostatecznie takie same (błąd 10^{-7}).

Aby znaleźć górną granice całkowania, należy iteracyjnie policzyć e^{-A} z błędem 10^{-7}. W naszym przypadku $A = 17$

Kod

```
#include <iostream>
#include <math.h>
#include <iomanip>
#include <vector>

using namespace std;
```

```
double f(double x) {
    return sin(M_PI*(1+sqrt(x))/(1+x*x))*exp(-x);
}

int findMax () {
    int B = 0;
    double res = exp(-B);
    double res_prev = res;
    B++;
    while(true) {
        res_prev = res;
        res = exp(-B);
        if(res_prev - res < 1e-7)
            break;
        B++;
    }
    return B;
}


int main() {
    cout.setf(ios::fixed, ios::floatfield);
    cout.precision(10);
    int A = 0;
    int B = findMax();

    // METODA ROMBERGA
    vector<vector<double>> R;
    double h = B-A;

    R.push_back(vector<double>());

    // liczymy R[0][0]
    R[0].push_back(h/2.0 * (f(A)+f(B)));

    double sumF = 0.5 * f(A) + 0.5 * f(B);

    int row = 1;
    while(true) {
        R.push_back(vector<double>());
        h = h/2.0;

        // liczymy pierwsza kolumne
        for (int k = 1; k <= pow(2,row-1); ++k) {
            sumF += f(A+(2*k-1)*h);
        }
        R[row].push_back(sumF * h);

        // liczymy kolejne wartosci w wierszach
        for (int col = 1; col <= row; ++col) {
            R[row].push_back(((pow(4,col)*R[row][col-1] - R[row-1][col-1]))/(pow(4,col)-1));
        }

        // sprawdzamy czy koniec algorytmu
        if(abs(R[row-1][row-1] - R[row][row]) < 1e-7) {
            break;
        }

        row++;
    }

    for (int i = 0; i < R.size(); ++i) {
```

```
        cout << R[i][R[i].size()-1] << "\\\\" << endl;
    }

    // METODA TRAPEZOW
    vector<double> T;
    sumF = 0.5 * f(A) + 0.5 * f(B);
    h = B - A;
    T.push_back(h/2.0 * (f(A)+f(B)));
    int counter = 1;
    while(true) {
        h = h/2.0;
        for (int k = 1; k <= pow(2,counter-1); ++k) {
            sumF += f(A+(2*k-1)*h);
        }
        T.push_back(sumF * h);

        if(abs(T[counter-1] - T[counter]) < 1e-7) {
            break;
        }
        counter++;
    }

    for (int j = 0; j < T.size(); ++j) {
        cout << T[j] << "\\\\" << endl;
    }

    return 0;
}
```

Wynik

$$\int\limits_0^\infty \sin\left(\pi\frac{1+\sqrt{x}}{1+x^2}\right)e^{-x}dx = \int\limits_0^{17} \sin\left(\pi\frac{1+\sqrt{x}}{1+x^2}\right)e^{-x}dx + \int\limits_{17}^\infty \sin\left(\pi\frac{1+\sqrt{x}}{1+x^2}\right)e^{-x}dx = -0.2172750475$$

Kolejne wyniki z metody Romberga (elementy na diagonali macierzy R):

$$\begin{bmatrix} 0.0000000195 \\ 0.0003854301 \\ 0.0417589014 \\ 0.3700452339 \\ 0.1903045877 \\ -0.1386427368 \\ -0.1783582379 \\ -0.2064448178 \\ -0.2135061248 \\ -0.2159571299 \\ -0.2168117564 \\ -0.2171117373 \\ -0.2172174157 \\ -0.2172547116 \\ -0.2172678860 \\ -0.2172725417 \\ -0.2172741874 \\ -0.2172747692 \\ -0.2172749748 \\ -0.2172750475 \end{bmatrix}$$

Do otrzymania wyniku potrzebujemy 20 iteracji.

Kolejne wyniki z metody trapezów:

$$\begin{bmatrix} 0.0000000195 \\ 0.0002890774 \\ 0.0294520640 \\ 0.2657806469 \\ 0.2188084713 \\ -0.0284941246 \\ -0.1370753055 \\ -0.1871271520 \\ -0.2063469586 \\ -0.2133668422 \\ -0.2158856422 \\ -0.2167825031 \\ -0.2171006966 \\ -0.2172133893 \\ -0.2172532664 \\ -0.2172673712 \\ -0.2172723590 \\ -0.2172741227 \\ -0.2172747463 \\ -0.2172749667 \\ -0.2172750447 \end{bmatrix}$$

Do otrzymania wyniku potrzebujemy 21 iteracji.