The sprawozdanie of Optymalizacja

Tom pierwszy: Piekło

Autory: Jan Waszut, Szymon Talar, Jakub Prejznar

El Temat: Lanie wody (Z dwóch zbiorników, znalezienie maksymalnego pola przekroju)

Limbo

Dostaliśmy straszne kody I naszym zadaniem było wymyślić jak to działa, żeby zrobić działający kod.

Działanie kodu otrzymanego polegało m.in. na operacjach macierzowych i magicznym robieniu równania różniczkowego.
Mieliśmy tam trochę miejsca na swoje kody, które zostało przez nas wyzyskane do cna. Takie wywłaszczenie procesu, tylko, że z kodu.

Krąg 1 - Funkcja testowa

Funkcja testowa dana była następującym kodem:

```
matrix ff1T(matrix x, matrix ud1, matrix ud2)
{
    matrix y;
    y = -cos(0.1 * x()) * exp(-pow(0.1 * x() - 2 * 3.14, 2)) + 0.002 * pow(0.1 * x(), 2);
    return y;
}
```

Odzwierciedlała funkcję podaną w konspekcie:

$$f(x) = -cos(0{,}1x) \cdot e^{-(0{,}1x-2\pi)^2} + 0{,}002 \cdot (0{,}1x)^2$$

Była ona dla nas niczym latarnia na brzegu dla zbłądzonych marynarzy wędrujących wśród złowrogich fal na wskroś suchego oceanu. Dała nam promyczek nadziei na pomyślne wykonanie czekającego na nas zadania. Funkcja jaka była – każdy widzi. Nic specjalnego. Pomogła natomiast w testowaniu wymaganych algorytmów.

Krąg 2 – Użyte środki przymusu do wykonania zadania

Było ich trzech. A w każdym z nich inna krew (i inny kod). Tak więc zaczynając od początku:

1. Metoda ekspansji:
   jedyna litościwa względem nas. Ograniczała się w gruncie rzeczy do wyszukania pożądanego przez nas zakresu operacji specjalnych aka przedziału wyszukiwania wartości extremalnych (minimalnych), z których to użytek będą mieć metody następujące:
   1.1. Metoda Fibonacciego:
   Poza wieloma podróżami w takie miejsca jak: Egipt, Syria, Grecja czy też Sycylia, udało mu się osiągnąć w życiu coś więcej. Dał on ziarno do utworzenia metody nomen omen Fibonacciego, wykorzystywanej przez nas do optymalizacji rozwiązania problemu. Nazwa ta wzięła swoją nazwę poprzez wzgląd na wykorzystanie złotego podziału, który to stopniowo zawęża przedział przeszukiwań. W tym celu wybierane są dwa punkty znajdujące się wewnątrz przedziału, równie oddalone od jego środka, a następnie na drodze eliminacji

usuwana zostawała część przedziału co do której mieliśmy pewność, że nie zawiera szukanej wartości.

1.1.1. Metoda Lagrange'a:

W każdej iteracji algorytmu również wybieramy dodatkowy punkt, tym razem praktycznie zawsze środek przedziału. W ten sposób uzyskujemy trzy punkty, pomiędzy którymi możemy przeprowadzić parabolę, zwaną w środowiskach naukowych wielomianem interpolacyjnym Lagrange'a. Jej ekstremum minimalne określa nam optymalną wartość dla tejże iteracji. Jego imieniem nazwano również krater, leżący na północny zachód od krateru Piazzi (na współrzędnych 32° 18' S; 72° 46'). Taka ciekawostka.

Krąg 3 – Opis maszkar argumentami zwanych

Dla Ekspansyji: (komentarze w kodzie autorskie)

```
//x0 punkt starowy
//d odległość poszukiwania kolejnego kroku (zaczynamy w prawo)
//alpha współczynnik ekspansji, alpha > 1
//algorytm szuka kolejnych punktów w prawo od x0
//jeśli wartość w kolejnym punkcie jest za duża lub równa tej z poprzedniego punktu: poszukiwanie
 przedziału się kończy, zwrócone są ostatnie dwa z trzech badanych punktów (xi-1, xi, xi+1). output
 zależy od wzajemnych relacji wartości funkcji w tych punktach
//jeżeli w pierwszej iteracji krok w prawo jest większy niż f(x0) algorytm poszukuje punktu na lewo od x0
double* expansion(matrix(*ff)(matrix, matrix, matrix), double x0, double d, double alpha, int Nmax, matrix
```

Dla Fibonacciego:

```
//metoda oparta o złoty podział
//a i b to punkty początkowe przedziału
//epsilon jest największą akceptowalną długością przedziału
//cel funkcji: znaleźć przedział poszukiwań, którego długość jest mniejsza niż epsilon
//wartość zwracana: x optymalny znajdujący się w granicach znalezionego przedziału.
solution fib(matrix(*ff)(matrix, matrix, matrix), double a, double b, double epsilon, matrix ud1 = NAN,
```

Dla Lagrange'a:

```
//a,b początkowy przedział poszukiwań
//epsilon minimalny przedział poszukiwań
//gamma minimalna różnica wartości punktu interpolowanego i zakładanego minimum
//działanie: tworzony jest punkt c będący w połowie odległośći ab
//przez punkty a c b przeprowadzamy parabolę
//punkt d jest minimum paraboli przeprowadzonej przez punkty  a c b ( a < c < b )
//liczona jest warość funkcji w punktach c, d, nastepnie wartości są porównywane
//jeżeli f(d)<f(c) zamieniamy koniec przedziału (b) na punkt c i punkt c na punkt d  //b->c, c->d
//ponownie tworzymy parabolę etc
//przypisywanie różni się w zależności od porównania
solution lag(matrix(*ff)(matrix, matrix, matrix), double a, double b, double epsilon, double gamma, int
```

Krąg 4 – problem rzeczywisty

Problem rzeczywisty ku zdziwieniu wszystkich towarzyszy tej podróży – okazał się być problematyczny. Do wykorzystania naszych funkcji Fibonacciego oraz Lagrange'a, potrzebowaliśmy funkcji rzeczywistej. W tym celu napisaliśmy dodatkowe funkcje, przekazywane jako jeden z argumentów funkcji docelowych. Funkcja testowa, zwana przez nas ff1T, widnieje powyżej, natomiast ta rzeczywista, dla odmiany, w ramach wyjątku, jest poniżej.

```
matrix ff1R(matrix x, matrix ud1, matrix ud2)
{
    matrix y;
    matrix Y0 = matrix(3, new double[3] { 5, 1, 10 });
    matrix* Y = solve_ode(df1, 0, 1, 1000, Y0, ud1, x);
    int n = get_len(Y[0]);
    double max = Y[1](0, 2);
    for (int i = 1; i < n; ++i)
        if (max < Y[1](i, 2))
            max = Y[1](i, 2);
    y = abs(max - 40);
    return y;
}
```

Bystre oko uważnego obserwatora spostrzegłoby użycie funkcji dodatkowej – takiej trochę skrytej – która zowie się df1. Jest wykorzystywana do obliczania równania różniczkowego naszego problemu, które odbywa się za pośrednictwem metody solve_ode . Wnętrze df1 z kolei jest znacznie nudniejsze, aczkolwiek w ramach studenckiej rzetelności zobligowani jesteśmy je również zamieścić w niniejszym sprawozdaniu.

Krąg 5 – funkcja df1

```
matrix df1(double t, matrix Y, matrix ud1, matrix ud2)
{
    double Pa = 0.75, Va = 5, Ta = 90;
    double Pb = 1, Vb = 1, Tb = 10, DB = 0.00365665;
    double Fin = 0.01, Tin = 10;
    double a = 0.98, b = 0.63, g = 9.81;

    matrix dY(3, 1);
    double FAout = Y(0) > 0 ? a * b * m2d(ud2) * sqrt(2 * g * Y(0) / Pa) : 0;
    double FBout = Y(1) > 0 ? a * b * DB * sqrt(2 * g * Y(1) / Pb) : 0;
    dY(0) = -FAout;
    dY(1) = FAout + Fin - FBout;
    dY(2) = Fin / Y(1) * (Tin - Y(2)) + FAout / Y(1) * (Ta - Y(2));
    return dY;
}
```

Krąg 6 – implementacja algorytmów

Pewien mądry człowiek kiedyś powiedział: „Obraz jest wart więcej niż tysiąc słów". Możliwe też że była to reklama Rafaello. Nie pamiętam. Ale uznamy że jest to prawda, i zamiast rozpisywać się nad implementacją po prostu ją zaprezentujemy.

```
double* expansion(matrix(*ff)(matrix, matrix, matrix), double x0, double d, double alpha, int
Nmax, matrix ud1, matrix ud2)
{
        try
        {
                double* p = new double[2] { 0, 0 };
                int i = 0;
                solution X0(x0), X1(x0 + d);
                X0.fit_fun(ff, ud1, ud2);
                X1.fit_fun(ff, ud1, ud2);
                if (X0.y == X1.y)
                {
```

```
                        p[0] = m2d(X0.x);
                        p[1] = m2d(X1.x);
                        return p;
                }
                if (X0.y < X1.y)
                {
                        d *= -1;
                        X1.x = X0.x + d;
                        X1.fit_fun(ff, ud1, ud2);
                        if (X1.y >= X0.y)
                        {
                                p[0] = m2d(X1.x);
                                p[1] = m2d(X0.x) - d;
                                return p;
                        }
                }
                solution X2;
                while (true)
                {
                        ++i;
                        X2.x = x0 + pow(alpha, i) * d;
                        X2.fit_fun(ff, ud1, ud2);
                        if (X2.y >= X1.y || solution::f_calls > Nmax)
                                break;
                        X0 = X1;
                        X1 = X2;
                }
                d > 0 ? p[0] = m2d(X0.x), p[1] = m2d(X2.x) : (p[0] = m2d(X2.x), p[1] =
m2d(X0.x));
                return p;
        }
        catch (string ex_info)
        {
                throw ("double* expansion(...):\n" + ex_info);
        }
}



solution fib(matrix(*ff)(matrix, matrix, matrix), double a, double b, double epsilon, matrix
ud1, matrix ud2)
{
        try
        {
                solution Xopt;
                Xopt.ud = b - a;
                int n = static_cast<int>(ceil(log2(sqrt(5) * (b - a) / epsilon) / log2((1 +
sqrt(5)) / 2)));
                int* F = new int[n] {1, 1};
                for (int i = 2; i < n; ++i)
                        F[i] = F[i - 2] + F[i - 1];
                solution A(a), B(b), C, D;
                C.x = B.x - 1.0 * F[n - 2] / F[n - 1] * (B.x - A.x);
                D.x = A.x + B.x - C.x;
                C.fit_fun(ff, ud1, ud2);
                D.fit_fun(ff, ud1, ud2);
                for (int i = 0; i <= n - 3; ++i)
                {
                        //cout << abs(m2d(B.x) - m2d(A.x)) << "\n";
                        if (C.y < D.y)
                                B = D;
                        else
                                A = C;
                        C.x = B.x - 1.0 * F[n - i - 2] / F[n - i - 1] * (B.x - A.x);
                        D.x = A.x + B.x - C.x;
                        C.fit_fun(ff, ud1, ud2);
                        D.fit_fun(ff, ud1, ud2);

                        Xopt.ud.add_row((B.x - A.x)());
                }
                Xopt = C;
                Xopt.flag = 0;
                return Xopt;
        }
        catch (string ex_info)
```

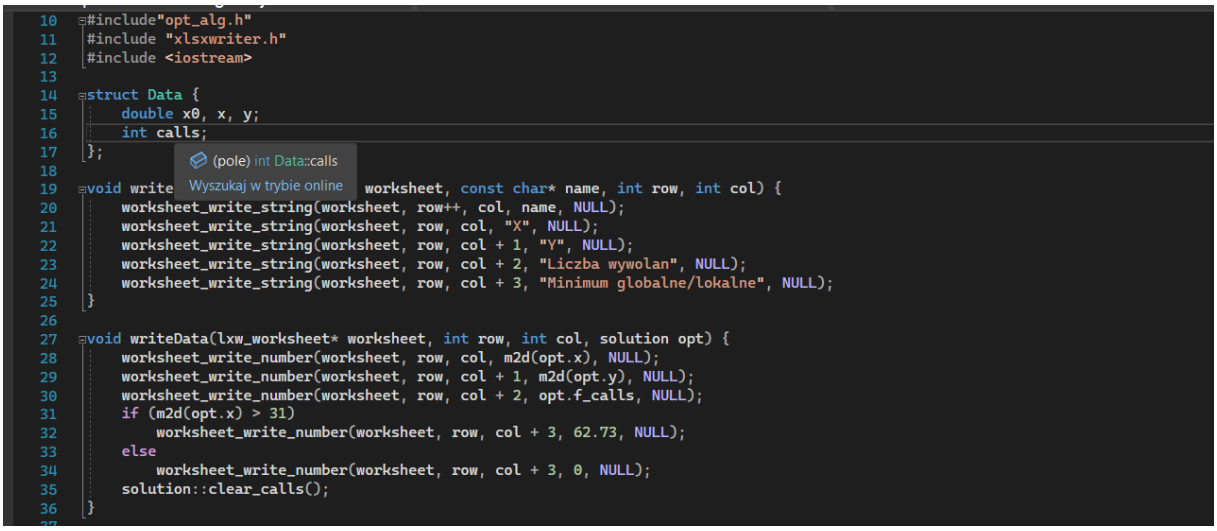
```
	{
		throw ("solution fib(...):\n" + ex_info);
	}

}



solution lag(matrix(*ff)(matrix, matrix, matrix), double a, double b, double epsilon, double
gamma, int Nmax, matrix ud1, matrix ud2)
{
	try
	{
		solution Xopt;
		Xopt.ud = b - a;

		solution A(a), B(b), C, D, D_old(a);
		C.x = (a + b) / 2;

		A.fit_fun(ff1T, ud1, ud2);
		B.fit_fun(ff1T, ud1, ud2);
		C.fit_fun(ff1T, ud1, ud2);

		double l, m;

		while (true)
		{
			l = m2d(A.y * (pow(B.x) - pow(C.x)) + B.y * (pow(C.x) - pow(A.x)) + C.y *
(pow(A.x) - pow(B.x)));
			m = m2d(A.y * (B.x - C.x) + B.y * (C.x - A.x) + C.y * (A.x - B.x));

			//cout << abs(m2d(B.x) - m2d(A.x)) << "\n";

			if (m <= 0)
			{
				Xopt = D_old;
				Xopt.flag = 2;
				return Xopt;
			}

			D.x = 0.5 * l / m;
			D.fit_fun(ff1T, ud1, ud2);

			if (A.x <= D.x && D.x <= C.x)
			{
				if (D.y < C.y)
				{
					A.x = A.x;
					C.x = D.x;
					B.x = C.x;

					A.fit_fun(ff1T, ud1, ud2);
					B.fit_fun(ff1T, ud1, ud2);
					C.fit_fun(ff1T, ud1, ud2);
				}
				else
				{
					A.x = D.x;
					C.x = C.x;
					B.x = B.x;

					A.fit_fun(ff1T, ud1, ud2);
					B.fit_fun(ff1T, ud1, ud2);
					C.fit_fun(ff1T, ud1, ud2);
				}
			}
			else if (C.x <= D.x && D.x <= B.x)
			{
				if (D.y < C.y)
				{
					A.x = C.x;
					B.x = B.x;
					C.x = D.x;
```

```
                                A.fit_fun(ff1T, ud1, ud2);
                                B.fit_fun(ff1T, ud1, ud2);
                                C.fit_fun(ff1T, ud1, ud2);

                        }
                        else
                        {
                                B.x = D.x;
                                C.x = C.x;
                                A.x = A.x;

                                A.fit_fun(ff1T, ud1, ud2);
                                B.fit_fun(ff1T, ud1, ud2);
                                C.fit_fun(ff1T, ud1, ud2);
                        }
                }
                else
                {
                        A.fit_fun(ff1T, ud1, ud2);
                        B.fit_fun(ff1T, ud1, ud2);
                        C.fit_fun(ff1T, ud1, ud2);

                        Xopt = D_old;
                        Xopt.flag = 2;

                        return Xopt;
                }

                Xopt.ud.add_row((B.x - A.x)());

                if (B.x - A.x < epsilon || abs(D.x() - D_old.x()) < gamma)
                {
                        A.fit_fun(ff1T, ud1, ud2);
                        B.fit_fun(ff1T, ud1, ud2);
                        C.fit_fun(ff1T, ud1, ud2);

                        Xopt = D;
                        Xopt.flag = 0;
                        break;
                }
                if (solution::f_calls > Nmax)
                {

                        A.fit_fun(ff1T, ud1, ud2);
                        B.fit_fun(ff1T, ud1, ud2);
                        C.fit_fun(ff1T, ud1, ud2);

                        Xopt = D;
                        Xopt.flag = 1;
                        break;
                }
                A.fit_fun(ff1T, ud1, ud2);
                B.fit_fun(ff1T, ud1, ud2);
                C.fit_fun(ff1T, ud1, ud2);

                D_old = D;
        }
        A.fit_fun(ff1T, ud1, ud2);
        B.fit_fun(ff1T, ud1, ud2);
        C.fit_fun(ff1T, ud1, ud2);

        return Xopt;

    }
    catch (string ex_info)
    {
        throw ("solution lag(...):\n" + ex_info);
    }
}
```

Krąg 7 – funkcja główna programu

Funkcja gówna miała za zadanie najpierw przechować wyniki obliczeń, a potem grzecznie pod strzałami pejcza wyrzucić je do pliku z rozszerzeniem .xlsx. Dzięki naszym wykwalifikowanym inżynierom pobierającym nauki u północnych mędrców kodowania w Oxenfurcie, udało się doprowadzić nasz program do stanu ujarzmionego i grzecznego niczym Theon Greyjoy po akcji z Boltonami w „Grze o tron”.

Kod ów miał lico +- takie, czyli średnie:



```
    worksheet_write_string(worksheet, 2, 2, "X", NULL);
    worksheet_write_string(worksheet, 2, 3, "Y", NULL);
    worksheet_write_string(worksheet, 2, 4, "Liczba wywolan", NULL);
    writeHeader(worksheet, "Fibo", 1, 6);
    writeHeader(worksheet, "Lag", 1, 11);

    for (int q = 0; q < 3; q++) {
        if (q == 0)
            alpha = 2;
        if (q == 1)
            alpha = 5;

        if (q == 2)
            alpha = 1.2;

        for (int i = 0; i < 100; i++)
        {
            ab = expansion(ff1T, data[i].x0, d, alpha, Nmax);
            data[i].x = ab[0];
            data[i].y = ab[1];
            data[i].calls = solution::f_calls;
            solution::clear_calls();
        }

        int row = 3 + 102 * q, col = 0;

        worksheet_write_number(worksheet, row, col, alpha, NULL);
```

```
        worksheet_write_number(worksheet, row, col, alpha, NULL);

        for (row; row < 103 + 102 * q; row++)
        {
            worksheet_write_number(worksheet, row, col + 1, data[row - (3 + 102 * q)].x0, NULL);
            worksheet_write_number(worksheet, row, col + 2, data[row - (3 + 102 * q)].x, NULL);
            worksheet_write_number(worksheet, row, col + 3, data[row - (3 + 102 * q)].y, NULL);
            worksheet_write_number(worksheet, row, col + 4, data[row - (3 + 102 * q)].calls, NULL);
        }

        col += 6;

        for (int row = 3 + 102 * q; row < 103 + 102 * q; row++)
            writeData(worksheet, row, col, fib(ff1T, data[row - (3 + 102 * q)].x, data[row - (3 + 102 * q)].y, epsilon));

        col += 5;


        for (int row = 3 + 102 * q; row < 103 + 102 * q; row++)
            writeData(worksheet, row, col, lag(ff1T, data[row - (3 + 102 * q)].x, data[row - (3 + 102 * q)].y, 0.0001, 1e-7,
              Nmax));
    }


    lxw_worksheet* worksheet_2 = workbook_add_worksheet(workbook, "Tabela 2");
    workbook_close(workbook);

}
```

*kod może nie zawierać najaktualniejszej wersji siebie z racji tego iż morfologia tego straszydła potrafiła się zmieniać co ułamek sekundy (w zależności od tego jak szybko ktoś klikał w klawiaturę)

Oksenfurcki Inżynier o podejrzanie znajomo brzmiącym nazwisku (nasuwa się nazwisko pewnego temerskiego szewca) piszący ten kod stwierdził, że wszelkie komentarze dot. kodu wrzuci na GitHuba w swoim czasie. A póki co wiemy co mamy robić.

Krąg 8 – Delikatnie nadgnite owoce naszej pracy (bądź co bądź 4 dni po terminie)

Wyniki są w Excelu. Rezultaty zawarte w tabeli o za małym numerze „1” dokładnie na oko wydają się być sensowne, natomiast w innej tabeli, tabeli o numerku „2” znajdują się uśrednione outputy dla każdego ze 100 wywołań funkcji dla każdego z 3 współczynników ekspansji użytego do obliczeń. Screena nie ma sensu wstawiać, skoro Excel istnieje. Ostatni 303 wiersz zawiera przykład bez inputu w postaci przedziału z metody ekspansji.

W tabeli 2 można zauważyć, że użyte metody częściej znajdywały minimum lokalne. A to nie jest najlepsza wiadomość.

W arkuszu wykres są wstawione 3 wykresy postaci: wielkość przedziału(nr iteracji). (jeden złączony z dwóch pojedynczych).

W tabeli Symulacja znajdują się wartości wymaganych parametrów w ciągu kolejnych iteracji obu funkcji. Wyniki tam zawarte są o tyle dobre, że temperatura w zbiorniku b w większości przypadków nie przekroczyła Tmax=50 stopni C, zatem można stwierdzić, że symulacja jest legitna.

Krąg 9 - Wnioski

Wniosek numer 1: Metoda Lagrange'a, w przeciwieństwie do metody Fibonacciego potrafi zaowocować niespodziewanymi rezultatami w postaci wartości ekstremum dążącym do nieskończoności. Może to mieć miejsce w przypadku, gdy minimum wielomianu interpolowanego przez wykorzystywany w algorytmie wielomian korzysta ze zbyt wysokiego współczynnika ekspansji; Dla wynoszącego 1.2 nie mieliśmy żadnych przypadków takiego zachowania, natomiast wraz ze wzrostem jego wartości takich przypadków bywało więcej.

Wniosek numer 2: Metoda Lagrange'a jest przeważnie szybsza, jednakowoż bywa również dalece niestabilna. O ile średnia liczba wywołań funkcji jest znacznie niższa w przypadku metody Lagrange'a, w Tabeli 1 są odnotowane przypadki gdzie ich liczba przekracza 100, podczas gdy maksimum dla metody Fibonacciego wynosiło 74. Co ciekawe, najwyższe wyniki dla metody Lagrange'a zdobywane są dla współczynnika „pośredniego" tzn. równego 2, natomiast zarówno dla wyższego jak i niższego wartości nie dorównywały nawet tym z metody Fibonacciego. Oznaczałoby to, że dla pewnej wartości współczynnika alpha możliwe jest, aby algorytm ten (Lagrange'a) co prawda znajdował za każdym razem poprawne minimum, ale przesuwał się jednocześnie w jego kierunku z na tyle małą szybkością że zajmuje to zdecydowanie więcej iteracji niż powinno w przypadku poprawnie zdefiniowanego współczynnika.

Wniosek numer 3: Wyniki uzyskiwane przez metodę Fibonacciego były średnio również dokładniejsze od tych uzyskiwanych przez metodę Lagrange'a. Jednakże poszczególne wywołania metody Lagrange'a potrafiły dawać dokładniejsze wyniki, przy czym przypadki występowania tego zjawiska pokrywały się z tymi gdy liczba wywołań funkcji nagle wzrastała. Oznacza to, że dla większej liczby wywołań metoda Lagrange'a jest w stanie wyświetlać dokładniejsze wyniki, przy czym jest to dość rzadkie zjawisko.

Wniosek numer 4: Zaimplementowanie poprawnej metody ekspansji znacznie faworyzuje metodę Lagrange'a, bazując na wykresach. Wynika z nich, iż dla sztywno ustawionego przedziału metoda Fibonacciego jest w stanie z łatwością przybliżyć poszukiwaną przez nas wartość, gdy tym czasem metoda Lagrange'a jest jeszcze bardziej nieprzewidywalna, ponieważ uzyskane przez nią minimum lokalne dla danej iteracji może znajdować się poza poszukiwanym przedziałem i to z większym prawdopodobieństwem, niż w przypadku poprzednich badań.

Wniosek numer 5: Dla symulacji rzeczywistego problemu różnice w działaniu obu funkcji były marginalne, co da się zinterpretować jako ich podobną wydajność w przypadku poprawnie zdefiniowanej metody ekspansji. Co prawda obie funkcje minimalnie wykroczyły poza wskazaną przez nas granicę 50 °C, jednakże różnice nie przekraczały drugiego miejsca po przecinku. W związku z tym możemy uznać je za wystarczająco dokładne dla naszych badań.