Kierunek Inżynieria Obliczeniowa	**Temat laboratoriów** Ćwiczenie 2 - Optymalizacja funkcji wielu zmiennych metodami bezgradientowymi	**Data ćwiczenia** 6.11.2024, 20.11.2024
Przedmiot Optymalizacja	Karolina Kurowska Szymon Majdak Amelia Nalborczyk	**Grupa** 2

Cel

Celem ćwiczenia jest wykorzystanie bezgradientowych metod optymalizacji do wyznaczenia minimum funkcji celu uwzględniając ograniczenia oraz wykorzystanie do rozwiązania problemu optymalizacji.

Przebieg ćwiczenia

Zadanie 1 funkcja testowa celu

Na ćwiczeniach zaimplementowano dwa algorytmy:

1. Metoda sympleksu Neldera-Meada:

```cpp
solution sym_NM (matrix (*ff)(matrix, matrix, matrix), matrix x0, double s, double alpha, double beta, double gamma, double delta, double epsilon, int Nmax, matrix ud1, matrix ud2)
{
    try
    {
        int n = get_len (x0);
        matrix e = ident_mat (n);
        int N = n + 1;
        solution* p = new solution[N];
        solution p_odb, p_z, p_e;
        matrix p_podloga (n, 1);
        int max;
        int min;

        p[0].x = x0;
        p[0].fit_fun (ff, ud1, ud2);

        for (int i = 1; i < N; ++i) {
            p[i].x = p[0].x + s * e[i - 1];
            p[i].fit_fun (ff, ud1, ud2);
        }

        while (true) {
            max = 0;
            min = 0;
            p_podloga = matrix (n, 1);

            for (int i = 1; i < N; ++i) {
                if (p[i].y < p[min].y)
                    min = i;
                if (p[i].y > p[max].y)
                    max = i;
            }


            for (int i = 0; i < N; ++i) {
                if (i != max)
                    p_podloga = p_podloga + p[i].x;
            }

            p_podloga = p_podloga / n;
            p_odb = p_podloga + alpha * (p_podloga - p[max].x);
            p_odb.fit_fun (ff, ud1, ud2);

            if (p_odb.y < p[min].y) {
                p_e = p_podloga + gamma * (p_odb.x - p_podloga);
                p_e.fit_fun (ff, ud1, ud2);
                if (p_e.y < p_odb.y) {
                    p[max] = p_e;
                }
                else {
                    p[max] = p_odb;
                }
            }
            else {
                if (p[min].y <= p_odb.y && p_odb.y < p[max].y) {
                    p[max] = p_odb;
                }
                else {
                    p_z = p_podloga + beta * (p[max].x - p_podloga); // Punkt kontrakcji
                    p_z.fit_fun (ff, ud1, ud2);
                    if (p_z.y >= p[max].y) {
                        for (int i = 0; i < N; ++i) {
                            if (i != min) {
                                p[i] = p[min].x + delta * (p[i].x - p[min].x);
                                p[i].fit_fun (ff, ud1, ud2);
                            }
                        }
                    }
                    else {
                        p[max] = p_z;
                    }
                }
            }

            if (solution::f_calls > Nmax) {
                p[min].flag = -1;
                cout << "Error: f_calls > Nmax" << endl;
                break;

            }
            double max_s = 0.0;
            for (int i = 1; i < N; ++i) {
                if (max_s < norm (p[min].x - p[i].x))
                    max_s = norm (p[min].x - p[i].x);
            }
            if (max_s < epsilon) {
                return p[min];
            }
        }
    }
    catch (string ex_info)
    {
        throw ("solution sym_NM(...):\n" + ex_info);
    }
}
```

2. Funkcję kary:

```cpp
solution pen (matrix (*ff)(matrix, matrix, matrix), matrix x0, double c, double a, double epsilon, int Nmax, matrix ud1, matrix ud2)
{
    try {
        solution::clear_calls ();
        double alpha = 1, beta = 0.5, gamma = 2, delta = 0.5, s = 0.5;
        solution X (x0), X_i;
        matrix c0 (2, new double[2] { c, a });
        while (true) {
            X_i = sym_NM (ff, X.x, s, alpha, beta, gamma, delta, epsilon, Nmax, ud1, c0);
            if (norm (X_i.x - X.x) < epsilon) {
                return X_i;
            }
            c0 (0) = c0 (0) * a;
            if (solution::f_calls > Nmax) {
                X_i.flag = -1;
                cout << "Error..." << endl;
                break;
            }
            X = X_i;

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

Wyznaczone według wzorów:

$S(x_1, x_2) \;=\; \sum_{i=1}^{n} \left(\max\left(0,\; g_i(x_1, x_2)\right)\right)^2,$

$$S(x_1, x_2) \;=\; -\sum_{i=1}^{n} \frac{1}{g_i(x_1, x_2)}$$

Kod testowej funkcji celu wraz z zewnętrzną i wewnętrzną funkcją kary:

```
matrix ff3T (matrix x, matrix ud1, matrix ud2)
{
    matrix Y;

    double x_1 = m2d(x(0));
    double x_2 = m2d(x(1));
    double numerator = sin (M_PI * sqrt (pow ((x_1 / M_PI), 2) + pow ((x_2 / M_PI), 2)));
    double denominator = M_PI * sqrt (pow ((x_1 / M_PI), 2) + pow ((x_2 / M_PI), 2));
    Y = numerator / denominator;

    double a = ud1(0);
    double c = ud2(0); //c>0

    //ograniczenia funckja kary wewnętrzna
    if (1 > x_1) {
        Y = 1e05;
    }
    else {
        Y = Y - c /(1 - x(0));
    }

    if (1 > x_2) {
        Y = 1e05;
    }
    else {
        Y = Y - c / (1 - x_2);
    }

    if (norm(x) > a) {
        Y = 1e05;
    }
    else {
        Y = Y - c / (norm(x) - a);
    }

    return Y;

    //ograniczenia fucnkja kary zewnętrzna
    if (ud2(1) > 1) {
        //g1
        if (1 > x_1) {
            Y = Y + c * pow ((-x_1) + 1, 2);
        }

        //g2
        if (1 > x_2) {
            Y = Y + c * pow ((-x_2) + 1, 2);
        }

        //g3
        if (norm(x) > a) {
            Y = Y + c * pow(norm(x) - a, 2);
        }

        return Y;
    }
}
```

Gdzie funkcja f jest zdefiniowana następująco:

$$f(x_1, x_2) = \frac{\sin\left(\pi\sqrt{\left(\frac{x_1}{\pi}\right)^2 + \left(\frac{x_2}{\pi}\right)^2}\right)}{\pi\sqrt{\left(\frac{x_1}{\pi}\right)^2 + \left(\frac{x_2}{\pi}\right)^2}},$$

Ograniczenia funkcji:

$$g_1(x_1) = \; -x_1 + 1 \; \leq \; 0,$$

$$g_2(x_2) = \; -x_2 + 1 \; \leq \; 0,$$

$$g_2(x_{1,} x_2) = \sqrt{x_1^2 + x_2^2} - \; a \; \leq \; 0,$$

Kod źródłowy funkcji lab3:

```cpp
void lab3 ()
{
    // Sekcja Tostowa

    ofstream file ("lab_03_tostowe.csv");
    if (!file.is_open ())throw string ("PLIK CSV NIE OTWARTY");
    file << "x0_1, x0_2, xz_1, xz_2, norm_z, y_z, f_calls_z"
        << "xw_1, xw_2, norm_w, y_w, f_calls_w\n";

    matrix x0 = matrix (2, 1, 1.0);

    double c0 = 2;
    matrix a[3] = { 4, 4.4934, 5 };

    const double epsilon = 1e-3;
    const int Nmax = 10000;

    double x0_1[100] = { 0 };
    double x0_2[100] = { 0 };

    //a[0]

    for (int i = 0; i < 100; i++) {
        do
            x0 = 5 * rand_mat (2, 1) + 1;
        while (norm (x0) > a[0]);

        x0_1[i] = x0 (0);
        x0_2[i] = x0 (1);
    }

    for (int i = 0; i < 100; i++) {
        x0 (0) = x0_1[i];
        x0 (1) = x0_2[i];

        cout << x0 (0) << ";" << x0 (1) << ";";

        solution curry_zewnetrzne1 = curry (ff3T, x0, c0, 2, epsilon, Nmax, a[0]);
        cout << curry_zewnetrzne1.x (0) << ";" << curry_zewnetrzne1.x (1) << ";" << norm (curry_zewnetrzne1.x) << ";" << curry_zewnetrzne1.y[0] << solution::f_calls << ";";

        file
            << x0 (0) << ","
            << x0 (1) << ","
            << curry_zewnetrzne1.x (0) << ","
            << curry_zewnetrzne1.x (1) << ","
            << norm (curry_zewnetrzne1.x) << ","
            << curry_zewnetrzne1.y (0) << ","
            << solution::f_calls << ",";

        solution curry_wewnetrzne1 = curry (ff3T, x0, c0, 0.5, epsilon, Nmax, a[0]);
        cout << curry_wewnetrzne1.x (0) << ";" << curry_wewnetrzne1.x (1) << ";" << norm (curry_wewnetrzne1.x) << ";" << curry_wewnetrzne1.y[0] << solution::f_calls << endl;
        file
            << curry_wewnetrzne1.x (0) << ","
            << curry_wewnetrzne1.x (1) << ","
            << norm (curry_wewnetrzne1.x) << ","
            << curry_wewnetrzne1.y (0) << ","
            << solution::f_calls << ","
            << "\n";

    }
```

Wykonujemy zadanie polegające na 100-krotnej optymalizacji dla każdej wartości parametru α = 4, α=4.4934 oraz α=5.Wyniki zamieszczamy w arkuszu “Tabela 1” w arkuszu Excel. Wartości średnie znajdują się w Tabeli 2 oraz poniżej:

Parametr a	Zewnętrzna funkcja kary					Wewnętrzna funkcja kary				
	x_1*	x_2*	r*	y*	Liczba wywołań funkcji celu	x_1*	x_2*	r*	y*	Liczba wywołań funkcji celu
4	2,8743655	2,7373328	4,000524255	-0,18920548	591,38	2,7004741	2,9496912	3,9997483	-0,18914736	2063,2
4,4934	2,7988517	3,2326132	4,4933997	-0,217234	184,65	2,9776211	3,3643069	4,4930532	-0,21723386	2747,8
5	3,0893371	2,9794857	4,49341	-0,217234	200,63	3,1769831	3,1769805	4,4929311	-0,21715045	963,46

Na podstawie powyższej tabeli - porównania zewnętrznej funkcji kary i wewnętrznej funkcji kary w optymalizacji może dostarczyć cennych informacji o efektywności obu podejść. Analizując wartość współczynnika α nie da się stwierdzić żadnej zależności co do efektywności, można jedynie stwierdzić, że dla wartości 5 liczba wywołań funkcji celu jest względnie niska.

Wyniki są bardzo dokładne w obu rodzajach funkcji kary. Zewnętrzna funkcja kary zdaje się prowadzić do nieco większych odchyleń od wartości rzeczywistych oraz miewa mniejszą dokładność wyników w przypadku x1*, x2* i r*. Wyniki dla y* są bardzo zbliżone w obu podejściach

Zadanie 2 problem rzeczywisty

Następnym zadaniem jest przeprowadzenie symulacji. Symulacja opisuje ruch piłki, która spada z wysokości 100 m, jednocześnie posiadając początkową prędkość poziomą i rotację. Ruch tej piłki jest modyfikowany przez siłę oporu powietrza oraz siłę Magnusa, która wynika z jej obrotu. Celem symulacji jest znalezienie takich wartości początkowej prędkości poziomej oraz rotacji, które spowodują jak największą odległość poziomą, jaką piłka osiągnie w trakcie spadania, przy jednoczesnym spełnieniu warunku, że piłka minie określony punkt (5,50) w odpowiedniej odległości.

W celu rozwiązania tego problemu stworzyliśmy funkcje odwzorowujące dane równania różniczkowe, które zostały zapisane w pliku user_funs.cpp. Funkcje te zamieszczamy poniżej:

```
matrix df3 (double t, matrix Y, matrix ud1, matrix ud2) {
    double m = 0.6;  // masa [kg]
    double r = 0.12; // promień piłki [m]

    double g = 9.81; // siła grawitacji [m/s^2]
    double C = 0.47; // wspolczynnik oporu
    double ro = 1.2; // gestosc powietrza [kg/m^3]

    double S = (M_PI * pow(r, 2)); // największy przekrój bryły (pole koła) [m^2]
    double V_x = m2d(Y(1));        // prędkość względem osi x [m/s]
    double V_y = m2d(Y(3));        // prędkość wględem osi y [m/s]
    double omega = m2d(ud2);       // predkosc kątowa [rad/s]

    //siły oporu powietrza
    double D_x = (0.5 * C * ro * S * V_x * abs(V_x));
    double D_y = (0.5 * C * ro * S * V_y * abs(V_y));

    //siły Magnusa
    double F_Mag_x = (ro * V_y * omega * M_PI * pow(r, 3));
    double F_Mag_y = (ro * V_x * omega * M_PI * pow(r, 3));

    //równania ruchu piłki (pochodne cząstkowe drugiego stopnia po czasie)
    matrix dY(4, 1);
    //dx
    dY(0) = V_x;
    dY(1) = (((-D_x) - F_Mag_x) / m);
    //dy
    dY(2) = V_y;
    dY(3) = (((-m) * g - D_y - F_Mag_y) / m);

    return dY;
}
```

Równania różniczkowe które są opisane w powyższej funkcji można wyrazić wzorami:

Równania ruchu piłki:

$$m\frac{d^2x}{dt^2} + D_x + F_{mx} = 0,$$

$$m\frac{d^2y}{dt^2} + D_y + F_{my} = -mg$$

Gdzie siły oporu powietrza to:

$$D_x = \frac{1}{2}C\rho S v_x |v_x|, \quad D_y = \frac{1}{2}C\rho S v_y |v_y|$$

Za to siły magnusa wyrażone są przez wzory:

$$F_{mx} = \rho v_y \omega \pi r^3,$$

Do rozwiązania tej części implementujemy funkcje celu, zamieszczoną poniżej:

```
void print_csv(matrix*& Y, int N) {
    ofstream file("symulacja.csv");
    if (!file.is_open())throw string("Nie da sie otworzyc pliku csv");
    file << "t,x,y" << endl;

    for (int i = 0; i < N; ++i)
        file << Y[0](i) << "," << Y[1](i, 0) << "," << Y[1](i, 2) << endl;

    file.close();
}
```

```
matrix ff3R(matrix x, matrix ud1, matrix ud2) {
    matrix y;
    matrix Y0(4, new double[4] { 0, x(0), 100, 0 });
    matrix* Y = solve_ode(df3, 0, 0.01, 7, Y0, ud1, x(1));
    int N = get_len(Y[0]);

    int minHeight50Index = 1e10, minHeight0Index = 1e10;
    double minHeight50 = 1e10;
    double minHeight0 = 1e10;

    for (int i = 0; i < N - 1; ++i) {
        double y_i = Y[1](i, 2);
        if (abs(y_i - 50) < minHeight50) { //Szukanie x, takiego, ze y = 0
            minHeight50 = abs(y_i - 50);
            minHeight50Index = i;
        }
        if (abs(y_i) < minHeight0) { //Szukanie x, takeigo, ze y = 50
            minHeight0 = abs(y_i);
            minHeight0Index = i;
        }
    }

    if (minHeight50Index == 1e10 || minHeight0Index == 1e10)
        return matrix(1e10);

    y = -Y[1](minHeight0Index, 0); //Szukanie minimum

    if (abs(x(0)) - 10 > 0) { //Kary dla y = 0, omega i y =0
        y = y + ud2() * pow(abs(x(0)) - 10, 2);
    }
    if (abs(x(1)) - 15 > 0) {
        y = y + ud2() * pow(abs(x(1)) - 15, 2);
    }
    if (abs(Y[1](minHeight50Index, 0) - 5) - 0.5 > 0) {
        y = y + ud2() * pow(abs(Y[1](minHeight50Index, 0) - 5) - 0.5, 2);
    }

    print_csv(Y, N);

    return y;
}
```

Funkcja celu ff3R oblicza wartość celu, która zależy od rozwiązania układu równań różniczkowych oraz kar za przekroczenie określonych progów zmiennych wejściowych oraz wartości w rozwiązaniu. Celem jest optymalizacja zmiennych tak, aby wartości w rozwiązaniu układu równań były bliskie 50, a zmienne wejściowe muszą mieścić się w zadanych zakresach.

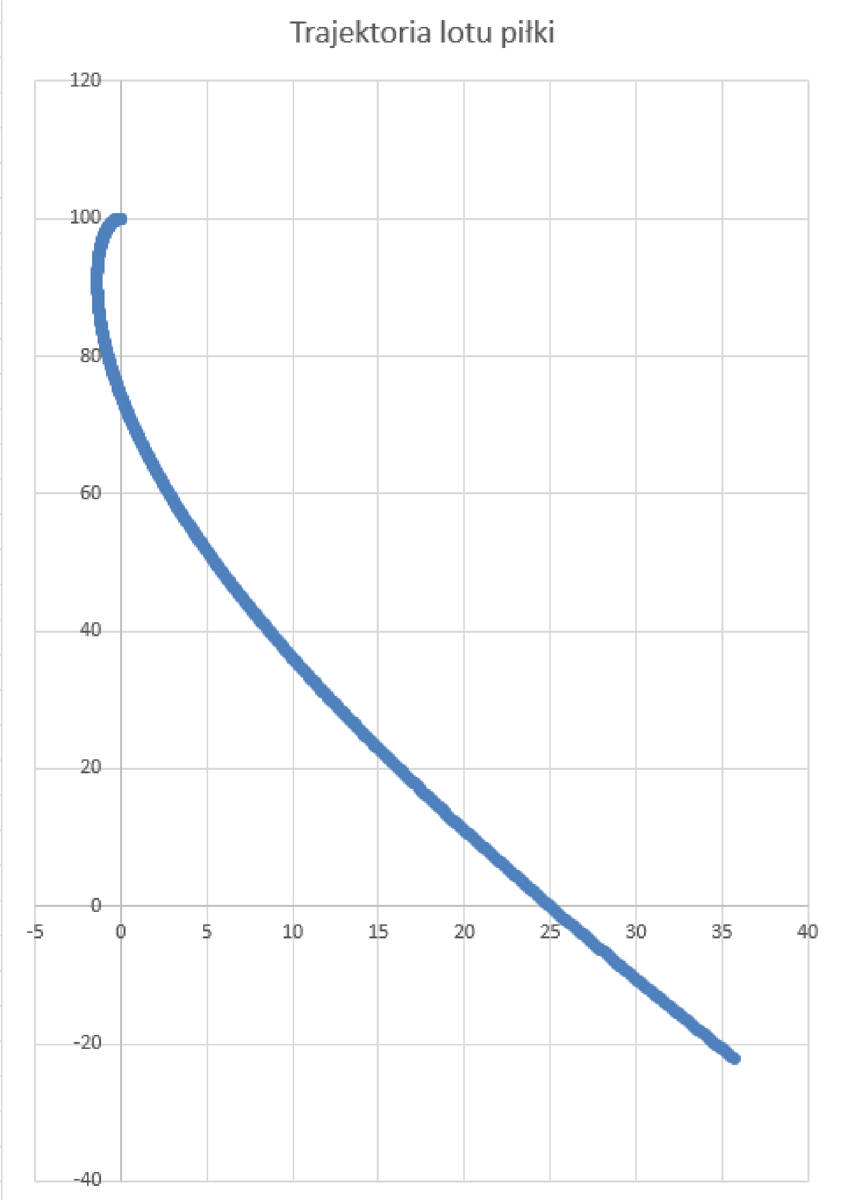
Parametry dla funkcji rzeczywistej zdefiniowane są następująco w funkcji main, co zostaje zamieszczone poniżej. Dla problemu rzeczywistego wybraliśmy punkty z przedziału –15 do 15 wartości 7 i 7.

```
//Problem rzeczywisty
matrix x0 = matrix(2, 1);
double c0 = 2;
matrix a[3] = { 4, 4.4934, 5 };

const double epsilon = 1e-5;
const int Nmax = 10000;

x0(0) = 7;         //V0x
x0(1) = 7;     //Omega [rad/s]
solution wynik_real = pen(ff3R, x0, c0, 2, epsilon, Nmax);
cout << wynik_real;
```

Wykonujemy optymalizacje. Wyniki znalezionych wartości prędkości oraz rotacji zamieszczamy w Tabeli 3. W arkuszu symulacja zamieszczam wykres na podstawie tych danych. Wykres prezentuje się następująco:



Wykres pokazuje typowy paraboliczny tor ruchu, który jest charakterystyczny dla rzutu ukośnego pod wpływem siły grawitacji. Analizując wykres możemy zobaczyć, że piłka idealnie przecina oś x (dotyka ziemi) w około 25 sekundzie. W danych wyszukujemy tą wartość.

6,03	24,875	0,196742
6,04	24,9808	-0,0314903

Piłka dotyka Ziemi dokładnie w odległości 25.6422 m od miejsca startu, dzieje się to w 6.03 s od wyrzutu piłki powietrze.

Wnioski

W ramach ćwiczenia wykorzystano metodę Neldera-Meada i funkcje kary do optymalizacji funkcji celu z ograniczeniami. W zadaniu drugim przeprowadzono symulację ruchu piłki z uwzględnieniem sił oporu powietrza i siły Magnusa. Celem było wyznaczenie optymalnej początkowej prędkości i rotacji, aby piłka osiągnęła maksymalną odległość i minęła zadany punkt. Oba zadania pokazały skuteczność bezgradientowych metod optymalizacji w zastosowaniach teoretycznych i rzeczywistych.