Optymalizacja z ograniczeniami funkcji wielu zmiennych metodami bezgradientowymi	Data wykonania: 28.11.2025 r.	Optymalizacja
Dominika Myszka Tatsiana Merzianiova	Gr. 2	ITE

1. Cel:
   Celem ćwiczenia było wykorzystanie bezgradientowych metod optymalizacji (w szczególności metody sympleksu Neldera-Meada) do wyznaczenia ekstremów funkcji celu z uwzględnieniem ograniczeń. Ćwiczenie podzielono na dwa etapy: optymalizację testowej funkcji matematycznej przy użyciu zewnętrznej i wewnętrznej funkcji kary oraz rozwiązanie rzeczywistego problemu inżynierskiego (trajektoria lotu piłki z uwzględnieniem efektu Magnusa).
2. Zadanie 5a: Testowa funkcja celu
   2.1. Opis problemu i parametry
   Zadanie polegało na znalezieniu minimum funkcji f(x1,x2) z ograniczeniami nierównościowymi, dla trzech wariantów parametrzu a (4.0, 4.4.4934, 5.0).

- Metoda Sympleksu (Neldera-Meada) : □=1.0, □=0.5, γ=2.0, δ=0. Dokładność ϵ=10^−3, Nmax =5000
  .
- Funkcje Kary:
  - Zewnętrzna: cstart=1.0, mnożnik dc=2.0
  - Wewnętrzna: cstart =1.0, mnożnik dc=0.5.

2.2. Implementacja funkjci testowej:

```
// --- Laboratorium 3 ---
matrix ff3T(matrix x, matrix ud1, matrix ud2)
{
    double x1 = x(0);
    double x2 = x(1);
    double a = (!isnan(ud1(0))) ? ud1(0) : 4.0;

    // Funkcja celu
    double arg = sqrt(pow(x1 / M_PI, 2) + pow(x2 / M_PI, 2));
    double y = (abs(arg) < 1e-9) ? 1.0 : sin(M_PI * arg) / (M_PI * arg);

    if (isnan(ud2(0)) || isnan(ud2(1))) return matrix(y);

    double c = ud2(0);
    int type = (int)ud2(1);
    double penalty = 0;

    // Ograniczenia
    double g1 = -x1 + 1.0;
    double g2 = -x2 + 1.0;
    double g3 = sqrt(x1 * x1 + x2 * x2) - a;

    if (type == 1) // Zewnetrzna
    {
        penalty += pow(max(0.0, g1), 2);
        penalty += pow(max(0.0, g2), 2);
        penalty += pow(max(0.0, g3), 2);
        return matrix(y + c * penalty);
    }
    else if (type == 2) // Wewnetrzna
    {
        if (g1 >= -1e-7 || g2 >= -1e-7 || g3 >= -1e-7) return matrix(1e10);
        penalty -= 1.0 / g1;
        penalty -= 1.0 / g2;
        penalty -= 1.0 / g3;
        return matrix(y + c * penalty);
    }
    return matrix(y);
}
```

Powyższa funkcja oblicza wartość testowej funkcji celu. W zależności od flag type, doliczana jest kara za naruszenie ograniczeń: kwadratowa dla metody zewnętrznej lub odwrotna dla metody wewnętrznej (barierowej).

Metoda Funkcji Kary:

```
// ---------------------- FUNKCJA KARY ----------------------
solution pen(matrix(*ff)(matrix, matrix, matrix), matrix x0, double c, double dc, double epsilon, int Nmax, matrix ud1, matrix ud2)
{
    try {
        double alpha = dc;
        solution X(x0);
        solution X_prev(x0);
        matrix current_ud2 = ud2;
        current_ud2(0) = c;

        while (true)
        {
            // Algorytm Neldera-Meada
            X = sym_NM(ff, X_prev.x, 0.5, 1.0, 0.5, 2.0, 0.5, epsilon, Nmax, ud1, current_ud2);

            if (solution::f_calls > Nmax)
            {
                X.flag = 0;
                return X;
            }
            if (norm(X.x - X_prev.x) < epsilon)
            {
                X.flag = 1;
                return X;
            }

            X_prev = X;
            c = alpha * c;
            current_ud2(0) = c;
        }
    }
    catch (string ex_info) { throw ("solution pen(...):\n" + ex_info); }
}
```

Funkcja realizuje algorytm metody funkcji kary. W pętli głównej wywoływana jest metoda bezgradientowa (sympleks Neldera-Meada) dla aktualnej wartości współczynnika kary c. Po każdej iteracji sprawdzany jest warunek zbieżności, a współczynnik kary jest aktualizowany.

2.3. Wyniki optymalizacji:

Tabela 1. Fragment wyników szczegułowych

Parametr a	Lp.	$x_1^{(0)}$	$x_2^{(0)}$	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
	1	2,92304	1,45801	2,96522	2,68603	4,00091	-0,18931	590	2,74504	2,90934	3,99994	-0,18919	2563
	2	2,07797	2,54547	2,95935	2,69249	4,0009	-0,18931	578	2,71652	2,93591	3,99988	-0,18919	2264
	3	1,92372	2,13237	2,94415	2,70794	4,00011	-0,18921	592	2,74395	2,90947	3,99928	-0,18912	2348
	4	2,81866	1,35243	3,01102	2,63389	4,00045	-0,18925	691	2,74828	2,90624	3,99991	-0,18919	2467
	5	2,40572	2,97906	2,74322	2,91215	4,00074	-0,18929	556	2,72762	2,92564	3,99992	-0,18919	2420
	6	1,46708	2,87361	2,67933	2,97127	4,0009	-0,18931	577	2,71324	2,93894	3,99988	-0,18919	2469
	7	1,06063	2,91356	2,87327	2,78394	4,00075	-0,18929	567	2,77137	2,88427	3,99993	-0,18919	2573
	8	1,39442	1,38866	2,56675	3,0679	4,00003	-0,1892	965	2,74143	2,91267	3,99988	-0,18919	2508
	9	1,78404	2,49622	2,83502	2,82289	4,00076	-0,18929	585	2,70693	2,94468	3,99982	-0,18918	2553
	10	1,25459	2,84196	1,34763	3,76709	4,00089	-0,1893	581	2,71956	2,93309	3,99988	-0,18919	2527
	11	2,94383	2,56956	2,83536	2,82255	4,00076	-0,18929	591	2,65658	2,99008	3,99975	-0,18917	2207
	12	2,67849	2,83456	2,89428	2,76208	4,00074	-0,18929	576	2,77024	2,8854	3,99997	-0,1892	2696
	13	1,632	3,17242	2,55052	3,08143	4,00005	-0,18921	857	2,73948	2,91417	3,99964	-0,18916	2347
	14	2,57186	2,49982	2,84536	2,81246	4,00075	-0,18929	588	2,75313	2,90078	3,99928	-0,18912	1950
	15	2,75646	2,56461	2,91419	2,74107	4,00074	-0,18929	576	2,75103	2,90277	3,99928	-0,18912	2178
	16	2,72931	2,1076	2,89346	2,76294	4,00074	-0,18929	561	2,75343	2,90049	3,99928	-0,18912	2137
	17	1,04414	1,48084	2,197	3,34266	4,00003	-0,1892	1022	2,72143	2,93143	3,99993	-0,18919	2724
	18	1,57111	2,92436	2,52999	3,09829	4,00003	-0,1892	858	2,70632	2,94524	3,99983	-0,18918	2239
	19	2,89809	2,33021	2,99644	2,65047	4,00045	-0,18925	663	2,75119	2,90262	3,99928	-0,18912	2149
	20	2,31988	2,84613	2,67185	2,978	4,00091	-0,18931	594	2,71403	2,93825	3,99991	-0,18919	2545
	21	2,23139	3,13581	2,97758	2,67231	4,00091	-0,18931	598	2,74787	2,90536	3,99899	-0,18908	1940
	22	1,47102	3,43492	2,58706	3,0509	4,00011	-0,18921	848	2,72342	2,92956	3,99992	-0,18919	2479
	23	1,47218	3,67067	2,55077	3,08123	4,00005	-0,18921	874	2,71937	2,93332	3,99992	-0,18919	2588
	24	2,00723	1,28719	3,10857	2,51731	4,00001	-0,1892	1130	2,75682	2,89815	3,99991	-0,18919	2597
	25	2,47447	1,10316	2,8174	2,84047	4,00075	-0,18929	581	2,70173	2,949	3,99949	-0,18914	2321
	26	2,88654	2,44951	2,67066	2,97906	4,00091	-0,18931	614	2,76697	2,88822	3,99975	-0,18917	2357
	27	2,17466	1,68498	2,40986	3,19259	4	-0,1892	1208	2,7149	2,93739	3,99987	-0,18919	2473
	28	2,42241	2,21581	3,19073	2,41233	4,00001	-0,1892	1040	2,7045	2,94619	3,99929	-0,18912	2096
	29	2,84428	1,88482	2,90468	2,75072	4,00045	-0,18925	671	2,7506	2,90408	3,99994	-0,18919	2548
	30	1,86587	1,074	3,05681	2,58	4,00006	-0,18921	936	2,74229	2,91132	3,99949	-0,18914	2320
	31	2,37581	2,76442	2,85354	2,80416	4,00075	-0,18929	562	2,76743	2,88471	3,99753	-0,18891	1742

(Pełne zestawienie wyników znajduje się w załączniku xlsx3.xslx Tabela 1)

Tabela 2. Wartości średnihe wyników optymalizacji

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	2,80161	2,83156	4,00048	-0,18926	715	2,74003	2,91335	3,99962	-0,18916	2374
4,4934	3,08041	2,97399	4,4934	-0,21723	208	3,04064	3,30776	4,49318	-0,21723	3464
5	2,79371	3,27852	4,49341	-0,21723	173	3,17687	3,17692	4,49281	-0,21723	928

2.3. Opis wyników:

Tabela 2.:
Analiza statystyk wykazuje, że metoda zewnętrznej funkcji kary jest znacznie wydajniejsza obliczeniowo, wymagając kilkukrotnie mniej wywołań funkcji celu niż metoda wewnętrzna. W przypadku parametru a=4, gdzie ograniczenie jest aktywne, metoda zewnętrzna minimalnie przekracza obszar dopuszczalny (r>4), podczas gdy wewnętrzna dochodzi do granicy ściśle od środka. Dla a=5ograniczenie staje się nieaktywne, co pozwoliło obu metodom zbiec do tego samego optimum lokalnego wewnątrz obszaru r≈4.49). Wyniki te potwierdzają, że stroma bariera w metodzie wewnętrznej znacząco utrudnia zbieżność algorytmu bezgradientowego, takiego jak metoda sympleksu.

3. Zadanie 5b: Problem rzeczywisty

3.1 Celem była maksymalizacja zasięgu rzutu piłki (xend) przy uwzględnieniu siły oporu powietrza oraz siły Magnusa. Zmiennymi decyzyjnymi były prędkość początkowa pozioma *v0x* oraz prędkość kątowa ω .
Ograniczenia: zmienne decyzyjne w przedziale [-10,10]

3.2. Implementacja problemu rzeczywistego:

```
matrix df3(double t, matrix Y, matrix ud1, matrix ud2)
{
    double m = 0.6, r = 0.12, C = 0.47, rho = 1.2, g = 9.81;
    double S = M_PI * r * r;
    double omega = (!isnan(ud2(0, 0))) ? ud2(0) : ((!isnan(ud1(0, 0))) ? ud1(0) : 0.0);
    double vx = Y(1), vy = Y(3);

    // Siły oporu (przeciwne do zwrotu prędkości, stąd vx * abs(vx) w mianowniku daje z
    double Dx = 0.5 * C * rho * S * vx * abs(vx);
    double Dy = 0.5 * C * rho * S * vy * abs(vy);

    // Siły Magnusa
    double FMx = rho * vy * omega * M_PI * pow(r, 3);
    double FMy = rho * vx * omega * M_PI * pow(r, 3);

    matrix dY(4, 1);
    dY(0) = vx;
    dY(1) = -(Dx + FMx) / m;
    dY(2) = vy;
    dY(3) = -(Dy + FMy) / m - g;
    return dY;
}
```

Funkcja ta definiuje układ równań różniczkowych opisujących dynamikę ruchu piłki. Uwzględnia ona siły grawitacji, oporu aerodynamicznego (zależne od kwadratu prędkości) oraz siły Magnusa wynikające z ruchu obrotowego obiektu.

```
matrix ff3R(matrix x, matrix ud1, matrix ud2)
{
    double v0x = x(0);
    double omega = x(1);
    double c = (!isnan(ud2(0))) ? ud2(0) : 0.0;

    matrix Y0(4, 1);
    Y0(0) = 0.0; Y0(1) = v0x; Y0(2) = 100.0; Y0(3) = 0.0;
    matrix omega_mat(1, 1); omega_mat(0) = omega;

    matrix* S = solve_ode(df3, 0, 0.01, 7.0, Y0, NAN, omega_mat);
    matrix Y = S[1];
    int n = get_len(S[0]);

    double x_at_50 = 0.0;
    double x_end = 0.0;
    bool found_50 = false, found_0 = false;

    for (int i = 0; i < n - 1; ++i)
    {
        double y_curr = Y(i, 2), y_next = Y(i + 1, 2);
        double x_curr = Y(i, 0), x_next = Y(i + 1, 0);

        if (!found_50 && y_curr >= 50.0 && y_next <= 50.0) {
            x_at_50 = x_curr + (x_next - x_curr) * (50.0 - y_curr) / (y_next - y_curr);
            found_50 = true;
        }
        if (!found_0 && y_curr >= 0.0 && y_next <= 0.0) {
            x_end = x_curr + (x_next - x_curr) * (0.0 - y_curr) / (y_next - y_curr);
            found_0 = true;
        }
    }
    if (!found_0) x_end = Y(n - 1, 0);
    delete[] S;
```

```
    double penalty = 0;
    // v0x [-10, 10]
    if (v0x < -10) penalty += pow(-10 - v0x, 2);
    else if (v0x > 10) penalty += pow(v0x - 10, 2);
    // omega [-10, 10]
    if (omega < -10) penalty += pow(-10 - omega, 2);
    else if (omega > 10) penalty += pow(omega - 10, 2);
    // x na 50m w odleglosci 2m od x=5
    if (abs(x_at_50 - 5.0) > 2.0) penalty += pow(abs(x_at_50 - 5.0) - 2.0, 2);

    return matrix(-x_end + c * penalty);
}
```

Funkcja celu dla problemu rzeczywistego. Przeprowadza pełną symulację lotu piłki, wyznacza punkty przecięcia z wysokością bramki (50m) oraz ziemią (0m) przy użyciu interpolacji liniowej. Następnie oblicza wartość funkcji kary za naruszenie ograniczeń i zwraca ujemny zasięg, aby algorytm minimalizacji realizował zadanie maksymalizacji

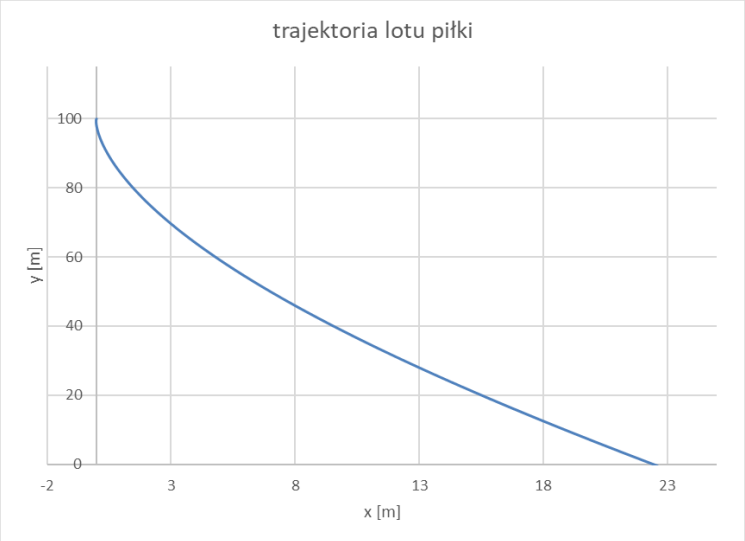
3.3. Wyniki optymalizacji:

Tabela 3. Wyniki optymalizacji problemu rzeczywistego

$v_{0x}^{(0)}$	$\omega^{(0)}$	v_{0x}^*	ω^*	x_{end}^*	x* dla y = 50m	Liczba wywołań funkcji celu
-5	6	-0,07226	10,00189	22,41902	7,00216	564

Symulacja

	A	B	C
1	t	x	y
2	0	0	100
3	0,01	-0,0007	99,9995
4	0,02	-0,0014	99,998
5	0,03	-0,0022	99,9956
6	0,04	-0,0029	99,9922
7	0,05	-0,0036	99,9877
8	0,06	-0,0043	99,9824
9	0,07	-0,005	99,976
10	0,08	-0,0057	99,9686
11	0,09	-0,0064	99,9603
12	0,1	-0,007	99,951
13	0,11	-0,0077	99,9407
14	0,12	-0,0084	99,9295
15	0,13	-0,009	99,9172
16	0,14	-0,0096	99,904
17	0,15	-0,0102	99,8898
18	0,16	-0,0108	99,8746
19	0,17	-0,0114	99,8585
20	0,18	-0,012	99,8414
21	0,19	-0,0125	99,8233
22	0,2	-0,013	99,8042
23	0,21	-0,0135	99,7842
24	0,22	-0,014	99,7632
25	0,23	-0,0145	99,7412
26	0,24	-0,0149	99,7182
27	0,25	-0,0153	99,6943



Wykres 1. Wykres trajektorii

3.4. Opis wyników problemu rzeczywistego:

Tabela 3.:

Wyniki optymalizacji pokazują, że algorytm zmaksymalizował zasięg rzutu (xend ≈22.42m). Aby zmieścić się w bramce przy maksymalnej rotacji generującej siłę nośną (ω=1), optymalizator wyznaczył nieintuicyjną, ujemną

prędkość początkową (v0x ≈−0.07m/s). Takie wysterowanie pozwoliło na pełne wykorzystanie efektu Magnusa, który zniósł piłkę w prawo dopiero w dalszej, spadkowej fazie lotu. Uzyskane rozwiązanie dowodzi skuteczności metody kary w rozwiązywaniu problemów, w których ograniczenia wymuszają kompromis między parametrami sterującymi.

Wykres 1.:

Przedstawiona trajektoria wizualizuje wynik optymalizacji, potwierdzając precyzyjne spełnienie ograniczenia przelotowego na samej granicy dopuszczalnego obszaru (x≈7m na wysokości 50 m). Charakterystyczny kształt krzywej, z początkowym lekkim odchyleniem w lewo, wynika z wyznaczonej przez algorytm ujemnej prędkości początkowej, która była konieczna do skompensowania silnego zniosu. W dalszej fazie lotu dominuje efekt Magnusa, który pod wpływem maksymalnej rotacji zakrzywia tor w prawo, znacząco wydłużając zasięg. Ostateczny punkt upadku (x≈22.4m) stanowi maksimum dla zadanych warunków, a przebieg krzywej idealnie ilustruje fizyczną interpretację znalezionego rozwiązania matematycznego.

4. Funckja lab3():

```
void lab3()
{
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis_x(-10.0, 10.0);
    uniform_real_distribution<> dis_y(-10.0, 10.0);

    // Parametry Zadania 5a
    double a_vals[] = { 4.0, 4.4934, 5.0 };
    int N_runs = 100;
    int Nmax = 5000;
    double epsilon = 1e-3;

    // Otwarcie plików dla Tabel 1 i 2
    ofstream fout_zew("wyniki_lab3_zew.csv");
    fout_zew << fixed << setprecision(5) << "a;x1_start;x2_start;x1_kon;x2_kon;y;L_wym;Flaga;r\n";
    ofstream favg_zew("wyniki_lab3_srednie_zew.csv");
    favg_zew << fixed << setprecision(5) << "a;x1_sr;x2_sr;y_sr;L_wyw_sr;r_sr\n";

    ofstream fout_wew("wyniki_lab3_wew.csv");
    fout_wew << fixed << setprecision(5) << "a;x1_start;x2_start;x1_kon;x2_kon;y;L_wyw;Flaga;r\n";
    ofstream favg_wew("wyniki_lab3_srednie_wew.csv");
    favg_wew << fixed << setprecision(5) << "a;x1_sr;x2_sr;y_sr;L_wyw_sr;r_sr\n";

    cout << "--- Lab 3: Zadanie 5a (Testowa) ---" << endl;

    for (double a : a_vals)
    {
        cout << "Przetwarzanie a = " << a << "..." << endl;
        double zew_s_x1 = 0, zew_s_x2 = 0, zew_s_y = 0, zew_s_calls = 0, zew_s_r = 0; int zew_success = 0;
        double wew_s_x1 = 0, wew_s_x2 = 0, wew_s_y = 0, wew_s_calls = 0, wew_s_r = 0; int wew_success = 0;

        for (int i = 0; i < N_runs; ++i)
        {
            solution::clear_calls();
            matrix x0(2, 1);
```

```
    do {
        x0(0) = dis_x(gen);
        x0(1) = dis_y(gen);
    } while (x0(0) <= 1.001 || x0(1) <= 1.001 || norm(x0) >= a - 0.001); // Ściśle wewnątrz

    matrix x0_base = x0;
    matrix ud1(1, 1); ud1(0) = a;
    matrix clean_ud(2, 1); clean_ud(0) = 0; clean_ud(1) = 0; // Do obliczenia czystej wartości f(x)

    // --- Metoda Zewnętrzna ---
    solution::clear_calls();
    // ud2 przechowuje: (0) współczynnik kary c, (1) typ kary (1-zew, 2-wew)
    matrix ud2_zew(2, 1); ud2_zew(0) = 1.0; ud2_zew(1) = 1.0;
    // pen: x0, c=1.0, dc=2.0 (alpha), epsilon, Nmax
    solution opt_zew = pen(f3T, x0_base, 1.0, 2.0, epsilon, Nmax, ud1, ud2_zew);

    double y_zew = f3T(opt_zew.x, ud1, clean_ud)(0);
    double r_zew = norm(opt_zew.x);
    fout_zew << a << ";" << x0_base(0) << ";" << x0_base(1) << ";" << opt_zew.x(0) << ";" << opt_zew.x(1) << ";" << y_zew << ";" << solution::f_calls << ";" << opt_zew.flag << ";" << r_zew << "\n";

    // Statystyka (tylko dla sukcesów flag=1 lub blisko optimum)
    if (opt_zew.flag == 1) { zew_s_x1 += opt_zew.x(0); zew_s_x2 += opt_zew.x(1); zew_s_y += y_zew; zew_s_calls += solution::f_calls; zew_s_r += r_zew; zew_success++; }

    // --- Metoda Wewnętrzna ---
    solution::clear_calls();
    matrix ud2_wew(2, 1); ud2_wew(0) = 1.0; ud2_wew(1) = 2.0;
    // pen: x0, c=1.0, dc=0.5 (alpha < 1 dla wew), epsilon, Nmax
    solution opt_wew = pen(f3T, x0_base, 1.0, 0.5, epsilon, Nmax, ud1, ud2_wew);

    double y_wew = f3T(opt_wew.x, ud1, clean_ud)(0);
    double r_wew = norm(opt_wew.x);
    fout_wew << a << ";" << x0_base(0) << ";" << x0_base(1) << ";" << opt_wew.x(0) << ";" << opt_wew.x(1) << ";" << y_wew << ";" << solution::f_calls << ";" << opt_wew.flag << ";" << r_wew << "\n";

    if (opt_wew.flag == 1) { wew_s_x1 += opt_wew.x(0); wew_s_x2 += opt_wew.x(1); wew_s_y += y_wew; wew_s_calls += solution::f_calls; wew_s_r += r_wew; wew_success++; }
}
if (zew_success > 0) favg_zew << a << ";" << zew_s_x1 / zew_success << ";" << zew_s_x2 / zew_success << ";" << zew_s_y / zew_success << ";" << zew_s_calls / zew_success << ";" << zew_s_r / zew_success << "\n";
if (wew_success > 0) favg_wew << a << ";" << wew_s_x1 / wew_success << ";" << wew_s_x2 / wew_success << ";" << wew_s_y / wew_success << ";" << wew_s_calls / wew_success << ";" << wew_s_r / wew_success << "\n";
```

```
    fout_zew.close(); favg_zew.close();
    fout_wew.close(); favg_wew.close();

    // ============================================================
    // ZADANIE 5B: PROBLEM RZECZYWISTY
    // ============================================================
    cout << "\n--- Zadanie 5b: Problem rzeczywisty ---" << endl;

    // 1. Weryfikacja modelu (v=5, w=10)
    {
        matrix x_ver(2, 1); x_ver(0) = 5.0; x_ver(1) = 10.0;
        matrix ud_null(1, 1); ud_null(0) = 0.0; // c=0 (bez kary)
        double ver = -ff3R(x_ver, NAN, ud_null)(0);
        cout << "[TEST] v=5, w=10 -> x_end = " << ver << " m (Oczekiwano: ~41.41)" << endl;
    }

    // 2. Optymalizacja
    // Punkt startowy
    matrix x0_real(2, 1);
    x0_real(0) = -5.0;
    x0_real(1) = 6.0;

    cout << "Start optymalizacji: v0x=" << x0_real(0) << ", omega=" << x0_real(1) << endl;

    solution::clear_calls();
    // Zewnętrzna funkcja kary: ud2(0) = c.
    matrix ud2_real(1, 1); ud2_real(0) = 1.0;

    // pen: c=1.0, dc=2.0 (alpha)
    solution opt_real = pen(ff3R, x0_real, 1.0, 2.0, 1e-3, 5000, NAN, ud2_real);

    // Wynik optymalny bez kary
    matrix final_ud(1, 1); final_ud(0) = 0.0;
    double x_end_opt = -ff3R(opt_real.x, NAN, final_ud)(0);

    // 3. Symulacja dla wykresu i tabeli
    matrix Y0(4, 1);
    Y0(0) = 0.0; Y0(1) = opt_real.x(0); Y0(2) = 100.0; Y0(3) = 0.0;
    matrix omega_opt(1, 1); omega_opt(0) = opt_real.x(1);

    matrix* S = solve_ode(df3, 0, 0.01, 7.0, Y0, NAN, omega_opt);
```

```
    double x_at_50 = 0.0;
    int n = get_len(S[0]);
    for (int i = 0; i < n - 1; ++i)
    {
        double y1 = S[1](i, 2); double y2 = S[1](i + 1, 2);
        if (y1 >= 50.0 && y2 <= 50.0) // Przejście przez y=50
        {
            double x1 = S[1](i, 0); double x2 = S[1](i + 1, 0);
            x_at_50 = x1 + (x2 - x1) * (50.0 - y1) / (y2 - y1);
            break;
        }
    }

    // Tabela 3
    ofstream f_tab3("wyniki_lab3_tabela_3.csv");
    f_tab3 << fixed << setprecision(5);
    f_tab3 << "v0x(0);omega(0);v0x_opt;omega_opt;x_end;x_dla_y_50;L_wyw\n";
    f_tab3 << x0_real(0) << ";" << x0_real(1) << ";"
        << opt_real.x(0) << ";" << opt_real.x(1) << ";"
        << x_end_opt << ";" << x_at_50 << ";"
        << solution::f_calls << "\n";
    f_tab3.close();

    // Wykres
    ofstream f_sim("lab3_symulacja.csv");
    f_sim << fixed << setprecision(4);
    f_sim << "t;x;vx;y;vy\n";
    for (int i = 0; i < n; ++i)
    {
        if (S[1](i, 2) >= -0.5) // Zapisz do momentu uderzenia w ziemię
            f_sim << S[0](i) << ";" << S[1](i, 0) << ";" << S[1](i, 1) << ";" << S[1](i, 2) << ";" << S[1](i, 3) << "\n";
    }
    f_sim.close();
    delete[] S;

    cout << "Zakonczono. Wszystkie pliki wygenerowane." << endl;
}
```

Funkcja sterująca przebiegiem laboratorium. W pierwszej części uruchamia serię testów statystycznych dla funkcji matematycznej. W drugiej części wykonuje optymalizację trajektorii lotu piłki, a następnie generuje dane do wykresu dla znalezionego rozwiązania optymalnego.

5. Wnioski:

Przeprowadzone ćwiczenie laboratoryjne pozwoliło na sformułowanie następujących wniosków końcowych:

1. Skuteczność metod bezgradientowych: Metoda sympleksu Neldera-Meada w połączeniu z funkcją kary skutecznie radzi sobie z problemami optymalizacji z ograniczeniami.
2. Dobór metody kary: Metoda zewnętrznej funkcji kary okazała się w badanych przypadkach znacznie wydajniejsza obliczeniowo (mniejsza liczba wywołań funkcji celu) i wystarczająco dokładna. Metoda wewnętrzna, choć gwarantuje spełnienie ograniczeń w trakcie obliczeń, jest numerycznie trudniejsza dla algorytmu sympleksu ze względu na strome zbocza bariery.