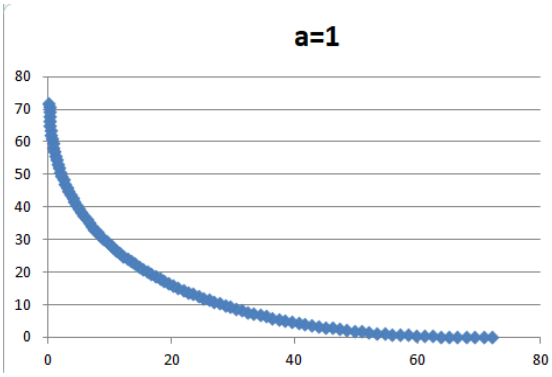
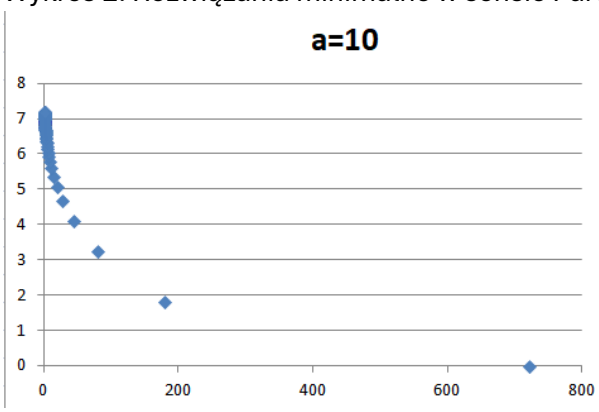
Optymalizacja wielokryterialna	Data wykonania: 15.01.2026 r.	Optymalizacja
Dominika Myszka Tatsiana Merzianiova	Gr. 2	ITE

1. Cel: . Celem laboratorium było rozwiązanie problemów optymalizacji wielokryterialnej, w tym analizy funkcji testowej i projektowania belk oraz określenie rozwiązań minimalnych w ujęciu Pareto
2. Parametry algorytmu
   Do optymalizacji wykorzystano metodę bezgradientową Powella.
   Parametry ogólne:
   - Metoda: Powell
   - Dokładność (ϵ): 1·10−5 (dla zadania testowego) oraz 1·10−6 (dla belki)
   - Ograniczenia: Zrealizowane za pomocą zewnętrznej funkcji kary (metoda barierowa).
3. Zadanie 5a: Funkcja Testowa
   Badano wpływ parametru a na kształt frontu Pareto dla dwóch funkcji kwadratowych. Przeprowadzono symulacje dla a∈{1,10,100}.
   Dla a=1: Funkcje f1 i f2 są symetryczne i mają taką samą wagę w procesie optymalizacji. Front Pareto jest symetrycznym łukiem. Dla a=10 i a=100: Zwiększenie parametru a powoduje "spłaszczenie" jednej z funkcji i zmianę krzywizny frontu Pareto. Wymusza to na algorytmie mocniejsze przesuwanie rozwiązania w stronę jednej ze składowych.

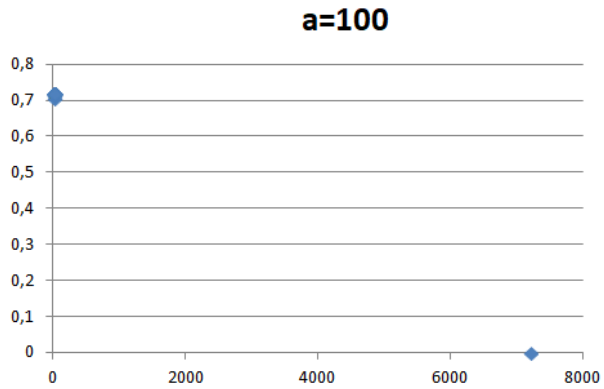
   Wykres 1. Rozwiązania minimalne w sensie Pareto dla a=1



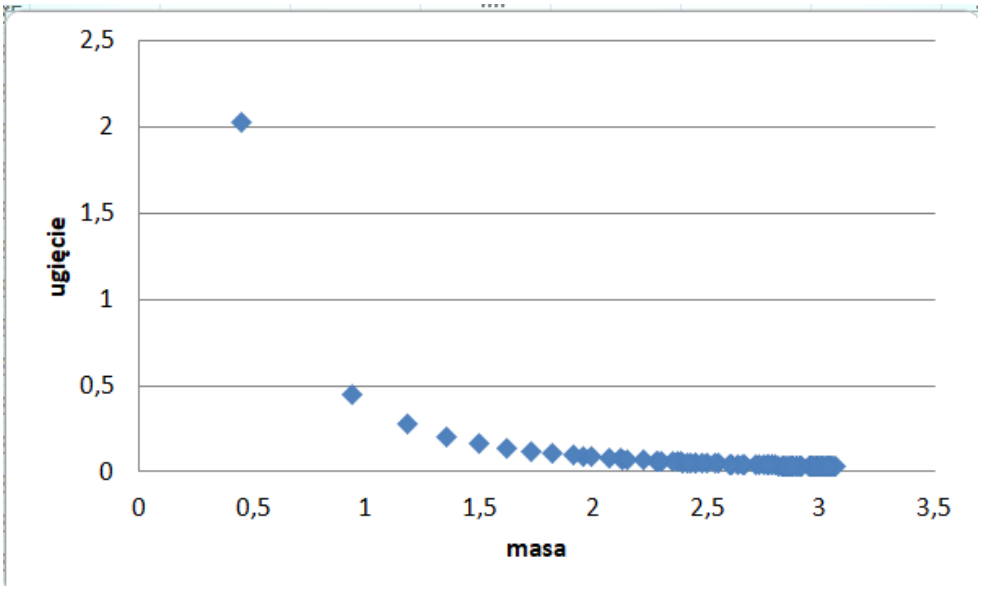
   Wykres 2. Rozwiązania minimalne w sensie Pareto dla a=10



Wykres 3. Rozwiązania minimalne w sensie Pareto dla a=100



4. Zadanie 5b: Problem rzeczywisty
   W ramach problemu rzeczywistego przeprowadzono optymalizację wielokryterialną belki wspornikowej, dążąc do jednoczesnej minimalizacji jej masy i ugięcia przy zachowaniu ograniczeń wytrzymałościowych.
   Wykres 4. Rozwiązanie minimalne w sensie Pareto dla analizowanej belki



5. **Wnioski**
   - Metoda sumy ważonej pozwala efektywnie wyznaczyć przybliżenie frontu Pareto, pod warunkiem wykonania odpowiedniej liczby symulacji dla różnych wag.
   - W problemach inżynierskich kluczowe jest **skalowanie kryteriów**. Bez sprowadzenia Masy i Ugięcia do wspólnego rzędu wielkości, jedno kryterium (masa) zdominowałoby proces optymalizacji.
   - Zastosowanie funkcji kary może powodować powstawanie minimów lokalnych. Zastosowanie podejścia **Multistart (wielokrotnego startu)** jest skuteczną metodą radzenia sobie z tym problemem i pozwala znaleźć globalne optimum.
   - Metoda Powella poprawnie radzi sobie z problemami z ograniczeniami, o ile zostaną one zaimplementowane poprzez funkcje kary.
6. Kod programu:
   Funkcja testowa:

```matrix ff5T(matrix x, matrix ud1, matrix ud2)
{
    double x1 = x(0);
    double x2 = x(1);

    double a = 1.0; if (get_len(ud1) > 0) a = ud1(0);
    double w = 0.5; if (get_len(ud2) > 0) w = ud2(0);

    double f1 = a * (pow(x1 - 3.0, 2) + pow(x2 - 3.0, 2));
    double f2 = (1.0 / a) * (pow(x1 + 3.0, 2) + pow(x2 + 3.0, 2));

    return matrix(w * f1 + (1.0 - w) * f2);
}
```

Funkcja belki:

```matrix ff5R(matrix x, matrix ud1, matrix ud2)
{
    double l = x(0);
    double d = x(1);

    double P = 1000.0;
    if (get_len(ud1) > 0) P = ud1(0);

    double w = 0.5;
    if (get_len(ud2) > 0) w = ud2(0);

    double ro = 7800.0;   // [kg/m3]
    double E = 207e9;     // [Pa]
    double Sy = 300e6;    // [Pa]
    double penalty = 0.0;
    if (l < 0.2)  penalty += pow(0.2 - l, 2);
    if (l > 1.0)  penalty += pow(l - 1.0, 2);
    if (d < 0.01) penalty += pow(0.01 - d, 2);
    if (d > 0.05) penalty += pow(d - 0.05, 2);

    if (penalty > 0) return matrix(1e12 + 1e12 * penalty);

    double mass = ro * l * M_PI * pow(d, 2) / 4.0;
    double u = (64.0 * P * pow(l, 3)) / (3.0 * E * M_PI * pow(d, 4));
    double sigma = (32.0 * P * l) / (M_PI * pow(d, 3));

    if (sigma > Sy)
    {
        penalty += pow((sigma - Sy) / Sy, 2) * 1e9;
    }
    double scaling = 1e5;

    double F = w * mass + (1.0 - w) * u * scaling + penalty;

    return matrix(F);
}
```

Funckja lab5:

```void lab5()
{
    srand((unsigned int)time(NULL));
    cout << "Generowanie Tabeli 1..." << endl;
    cout << "\nGenerowanie Tabeli 2 (Belka - Multistart)..." << endl;

    double P = 1000.0;
    int Nmax_beam = 2000;
    int RETRIES = 20;

    ofstream f2("wyniki_lab5_tabela_22.csv");
    f2 << "w;l(0) [mm];d(0) [mm];l* [mm];d* [mm];masa* [kg];ugiecie* [mm];naprezenie* [MPa];L.wyw\n";

    for (int i = 0; i <= 100; ++i)
    {
        double w = i / 100.0;
        cout << "\rPostep: " << i << "%" << flush;

        solution best_opt;
        best_opt.y = matrix(1e20);
        matrix best_x0(2, 1);
        for (int k = 0; k < RETRIES; ++k)
        {
            matrix x0_temp(2, 1);
            x0_temp(0) = random_double(0.2, 1.0);
            x0_temp(1) = random_double(0.01, 0.05);

            matrix ud1(1, 1); ud1(0) = P;
            matrix ud2(1, 1); ud2(0) = w;
            solution::clear_calls();
```

```
        solution current_opt = Powell(ff5R, x0_temp, 1e-5, Nmax_beam, ud1, ud2);
        if (current_opt.y(0) < best_opt.y(0))
        {
            best_opt = current_opt;
            best_x0 = x0_temp;
        }
    }

    double l = best_opt.x(0);
    double d = best_opt.x(1);

    double ro = 7800.0; double E = 207e9;
    double mass = ro * l * M_PI * pow(d, 2) / 4.0;
    double u = (64.0 * P * pow(l, 3)) / (3.0 * E * M_PI * pow(d, 4));
    double sigma = (32.0 * P * l) / (M_PI * pow(d, 3));

    f2 << d2str(w) << ";"
        << d2str(best_x0(0) * 1000.0) << ";"
        << d2str(best_x0(1) * 1000.0) << ";"
        << d2str(l * 1000.0) << ";"
        << d2str(d * 1000.0) << ";"
        << d2str(mass) << ";"
        << d2str(u * 1000.0) << ";"
        << d2str(sigma / 1e6) << ";"
        << solution::f_calls << "\n";
}
f2.close();
cout << "\nGotowe! Sprawdz 'wyniki_lab5_tabela_2.csv'." << endl;
```

Funkcja powell:

```
solution Powell(matrix(*ff)(matrix, matrix, matrix), matrix x0, double epsilon, int Nmax, matrix ud1, matrix ud2)
{
    int n = get_len(x0);
    matrix D = ident_mat(n);

    solution Xopt;
    Xopt.x = x0;
    Xopt.y = ff(x0, ud1, ud2);

    double h = 0.1;
    auto min_direction_local = [&](matrix x_start, matrix d) -> matrix
        {
            double step = h;
            double a = 0.0, b = step;
            double fa = ff(x_start + d * a, ud1, ud2)(0);
            double fb = ff(x_start + d * b, ud1, ud2)(0);
            solution::f_calls += 2;

            if (fb > fa) {
                step = -step;
                b = step;
                fb = ff(x_start + d * b, ud1, ud2)(0);
                solution::f_calls++;
            }
            while (true) {
                if (solution::f_calls >= Nmax) return x_start + d * b;
                double c = b + step;
                double fc = ff(x_start + d * c, ud1, ud2)(0);
                solution::f_calls++;

                if (fc < fb) {
                    a = b; fa = fb;
                    b = c; fb = fc;
                    step *= 2.0;
                }
                else {
                    if (step > 0) b = c; else { double temp = a; a = c; b = temp; }
                    break;
```

\

```
        double k = (sqrt(5.0) - 1.0) / 2.0;
        double xl = b - k * (b - a);
        double xr = a + k * (b - a);

        double fxl = ff(x_start + d * xl, ud1, ud2)(0);
        double fxr = ff(x_start + d * xr, ud1, ud2)(0);
        solution::f_calls += 2;

        while ((b - a) > epsilon) {
            if (solution::f_calls >= Nmax) break;
            if (fxl < fxr) {
                b = xr; xr = xl; fxr = fxl;
                xl = b - k * (b - a);
                fxl = ff(x_start + d * xl, ud1, ud2)(0);
                solution::f_calls++;
            }
            else {
                a = xl; xl = xr; fxl = fxr;
                xr = a + k * (b - a);
                fxr = ff(x_start + d * xr, ud1, ud2)(0);
                solution::f_calls++;
            }
        }
        return x_start + d * ((a + b) / 2.0);
    };
```

```
    while (solution::f_calls < Nmax)
    {
        matrix P0 = Xopt.x;

        for (int i = 0; i < n; ++i)
        {
            matrix di(n, 1);
            for (int j = 0; j < n; ++j) di(j) = D(j, i);
            Xopt.x = min_direction_local(Xopt.x, di);
        }

        if (norm(Xopt.x - P0) < epsilon) {
            Xopt.flag = 0;
            break;
        }

        matrix d_new = Xopt.x - P0;
        double d_len = norm(d_new);
        if (d_len < 1e-9) break;

        Xopt.x = min_direction_local(Xopt.x, d_new);

        for (int j = 0; j < n - 1; ++j) {
            for (int row = 0; row < n; ++row) D(row, j) = D(row, j + 1);
        }
        for (int row = 0; row < n; ++row) D(row, n - 1) = d_new(row);
    }

    Xopt.y = ff(Xopt.x, ud1, ud2);
    if (solution::f_calls >= Nmax) Xopt.flag = 1;

    return Xopt;
}
```