Jakub Latawiec

Rafał Malik

Optymalizacja

Laboratorium – optymalizacja wielokryterialna

1. **Cel ćwiczenia**
   Celem ćwiczenia jest zapoznanie się z problematyką optymalizacji wielokryterialnej i wyznaczenie rozwiązań minimalnych w sensie Pareto.

2. **Przeprowadzenie ćwiczenia**
   a) Implementacja metody Powella

```
solution Powell(matrix(*ff)(matrix, matrix, matrix), matrix x0, double epsilon,
int Nmax, matrix ud1, matrix ud2)
{
	try
	{
		solution Xopt;

		int n = get_len(x0);
		matrix d = ident_mat(n);
		matrix p;

		matrix h_sol_data(n, 2);
		solution h;

		matrix XB;
		XB = x0;

		double* range;

		while (true)
		{
			p = XB;

			for (int j = 0; j < n; ++j)
			{
				h_sol_data.set_col(p, 0);
				h_sol_data.set_col(d[j], 1);
				range = expansion(ff, 0.0, 1.0, 1.2, Nmax, ud1,
h_sol_data);
				h = golden(ff, range[0], range[1], epsilon, Nmax, ud1,
h_sol_data);
				p = p + h.x * d[j];
			}

			if (norm(p - XB) < epsilon)
			{
				Xopt.x = p;
				Xopt.fit_fun(ff, ud1, ud2);

				return Xopt;
			}

			if (solution::f_calls > Nmax)
				throw std::string("Maximum amount of f_calls reached!");

			for (int j = 0; j < n - 1; ++j)
```

```
                        d.set_col(d[j + 1], j);
                d.set_col(p - XB, n - 1);

                h_sol_data.set_col(p, 0);
                h_sol_data.set_col(d[n - 1], 1);
                range = expansion(ff, 0.0, 1.0, 1.2, Nmax, ud1, h_sol_data);
                h = golden(ff, range[0], range[1], epsilon, Nmax, ud1,
h_sol_data);

                XB = p + h.x * d[n - 1];
            }

            free(range);
            return Xopt;
        }
        catch (string ex_info)
        {
            throw ("solution Powell(...):\n" + ex_info);
        }
}
```

b) Implementacja testowej funkcji celu

```
matrix ff5T(matrix x, matrix ud1, matrix ud2)
{
    matrix y;

    double w = ud1(0);
    double a = ud1(1);

    if (isnan(ud2(0, 0)))
    {
        y = matrix(2, 1);
        y(0) = a * (pow(x(0) - 2, 2) + pow(x(1) - 2, 2));
        y(1) = (1.0 / a) * (pow(x(0) + 2, 2) + pow(x(1) + 2, 2));
    }
    else
    {
        matrix yt;
        yt = ff5T(ud2[0] + x * ud2[1], ud1);
        y = w * yt(0) + (1 - w) * yt(1);
    }
    return y;
}
```

c) Implementacja funkcji celu dla problemu rzeczywistego:

```
matrix ff5R(matrix x, matrix ud1, matrix ud2)
{
    matrix y;

    const double rho = 7800;
    const double P = 1000;
    const double E = 207e9;

    const double w = ud1(0);

    const double c = 1e10;
```

```
	if (isnan(ud2(0, 0)))
	{
		const double l = x(0);
		const double d = x(1);

		y = matrix(3, 1);
		y(0) = rho * l * M_PI * (std::pow(d, 2) / 4);
		y(1) = (64.0 * P * std::pow(l, 3)) / (3.0 * E * M_PI * std::pow(d,
4));
		y(2) = (32.0 * P * l) / (M_PI * std::pow(d, 3));
	}
	else
	{
		matrix xt = ud2[0] + x * ud2[1];
		matrix yt = ff5R(xt, ud1);

		y = w * (yt(0) - 0.12) / (15.3 - 0.12) + (1.0 - w) * (yt(1) - 4.2e-
5) / (3.28 - 4.2e-5);

		if (xt(0) < 0.2)
			y = y + c * pow(0.2 - xt(0), 2);
		if (xt(0) > 1.0)
			y = y + c * pow(xt(0) - 1.0, 2);

		if (xt(1) < 0.01)
			y = y + c * pow(0.01 - xt(1), 2);
		if (xt(1) > 0.05)
			y = y + c * pow(xt(1) - 0.05, 2);

		if (yt(1) > 0.005)
			y = y + c * pow(yt(1) - 0.005, 2);
		if (yt(2) > 300e6)
			y = y + c * pow(yt(2) - 300e6, 2);

	}
	return y;
}
```

d) Implementacja funkcji lab5:

```
void lab5()
{
#ifdef SAVE_TO_FILE
	create_environment("lab05");
#endif

	//Dane dokładnościowe
	double epsilon = 1E-4;
	int Nmax = 10000;

	//Generator liczb losowych
	std::random_device rd;
	std::mt19937 gen(rd());

#ifdef CALC_TEST
	std::uniform_real_distribution<> x0_dist(-10.0, 10.0);

	//Stringstream do zapisu danych
	std::stringstream test_ss;

	//Rozwiązanie dla wyników testowych
	solution test_sol;
```

```
	//Punkty startowe
	matrix test_x0{};

	//Długości kroków
	double a_arr[] = { 1.0, 10.0, 100.0 };

	for (double w = 0.0; w <= 1.0; w += 0.01)
	{
		test_x0 = matrix(2, new double[2] {x0_dist(gen), x0_dist(gen)});
		test_ss << test_x0(0) << ";" << test_x0(1) << ";";

		for (auto& a : a_arr)
		{
			test_sol = Powell(ff5T, test_x0, epsilon, Nmax, matrix(2, new
double[2] {w, a}));
			test_ss << test_sol.x(0) << ";" << test_sol.x(1) << ";" <<
test_sol.y(0) << ";" << test_sol.y(1) << ";" << solution::f_calls << ";";
			solution::clear_calls();
		}
		test_ss << "\n";
	}

#ifdef SAVE_TO_FILE
	save_to_file("test.csv", test_ss.str());
#endif
#endif

#ifdef CALC_SIMULATION
	std::uniform_real_distribution<> l_dist(0.2, 1.0);
	std::uniform_real_distribution<> d_dist(0.01, 0.05);

	//Stringstream do zapisu danych
	std::stringstream simulation_ss;

	//Rozwiązanie dla wyników testowych
	solution simualtion_sol;

	//Punkty startowe
	matrix x0{};

	for (double w = 0.0; w <= 1.0; w += 0.01)
	{
		x0 = matrix(2, new double[2] {l_dist(gen), d_dist(gen)});
		simulation_ss << x0(0) * 1000 << ";" << x0(1) * 1000 << ";";

		simualtion_sol = Powell(ff5R, x0, epsilon, Nmax, matrix(w));
		simulation_ss << simualtion_sol.x(0) * 1000 << ";" <<
simualtion_sol.x(1) * 1000 << ";" << simualtion_sol.y(0) << ";" <<
simualtion_sol.y(1) * 1000 << ";" << solution::f_calls << "\n";
		solution::clear_calls();
	}

#ifdef SAVE_TO_FILE
	save_to_file("simulation.csv", simulation_ss.str());
#endif

#endif
}
```

3. Parametry algorytmów

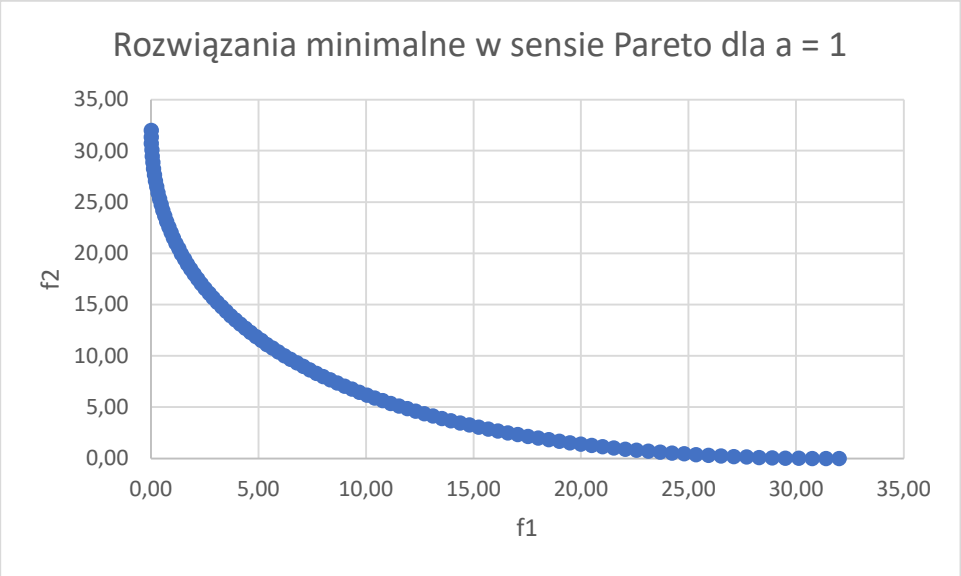
Dokładność (epsilon): 1E-4

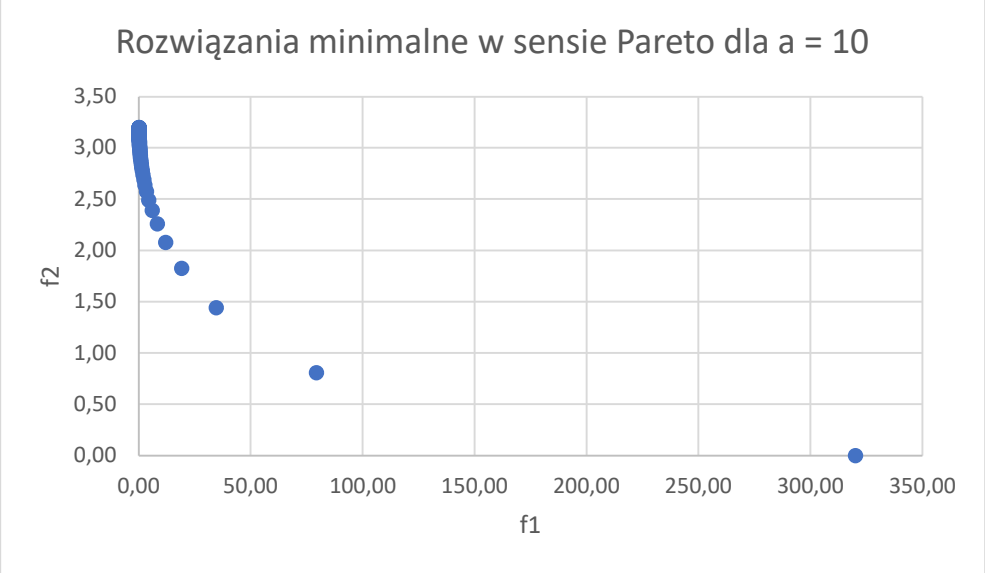
Maksymalna ilość wywołań funkcji (Nmax): 10 000

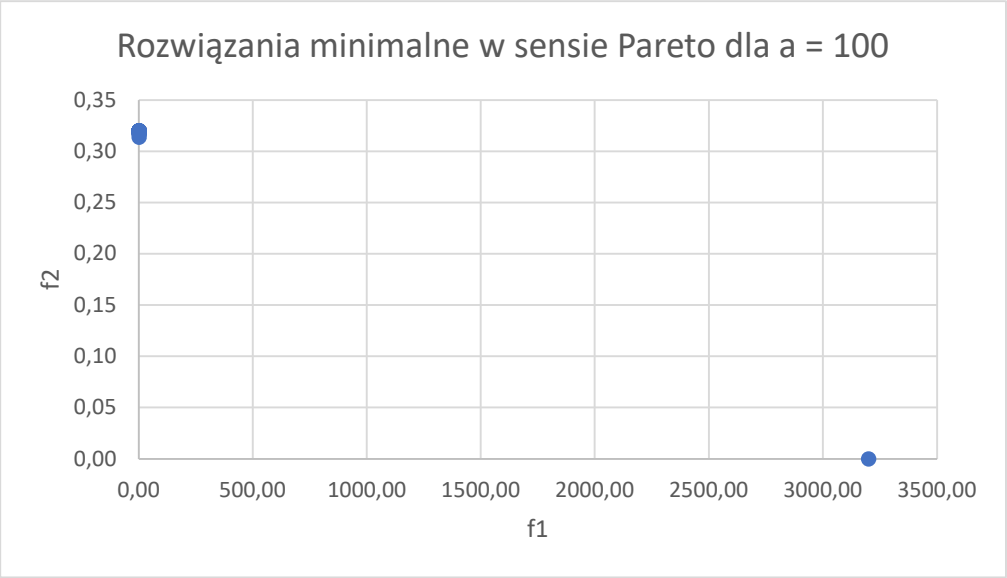
4. Dyskusja wyników

a) <u>Wyniki testowe:</u>

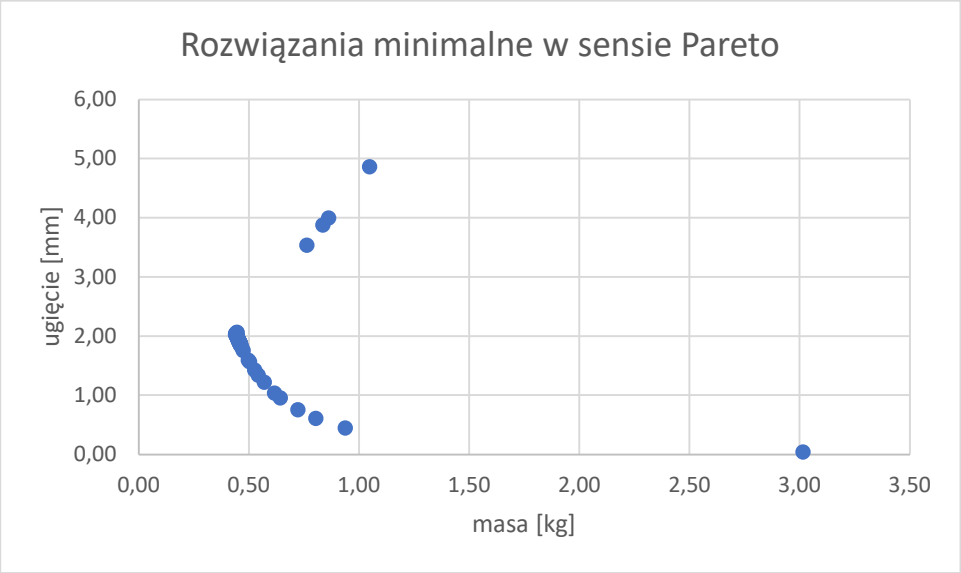
Ze wzrostem parametru „a” wyniki rozwiązań minimalnych są bardziej skupione. Dzieje się tak, ponieważ dla f1 „a” jest proporcjonalne a dla f2 jest odwrotnością, czyli wartości f1 są dużo większe od wartości f2 wraz ze zwiększaniem parametru „a”. Przy zmianie problemu wielokryterialnego na jednokryterialny, przez różnice w wartościach między f1 i f2 dla większych parametrów „a”, optymalizacja jest skierowana ku minimalizacji wartości f1. Wyjątkami są punkty odbiegające granicznie na prawą stronę wykresów dla a = 10 i a = 100. Występują one w obu przypadkach gdy waga (w) jest równa 0.







b) <u>Wyniki dla problemu rzeczywistego:</u>
Po odrzuceniu odbiegających wyników jesteśmy w stanie narysować wykres omawiający zależność ugięcie od masy. Możemy odczytać że dla większości przypadków ugięcie maleje wraz ze wzrostem masy.



5. Wnioski

Dzięki metodzie Powella i zamianie problemu jednokryterialnego na wielokryterialny jesteśmy w stanie znaleźć wiele rozwiązań minimalnych w zależności od wag kryteriów. Pozwala to na wyrysowanie wykresów Pareto i znalezienie interesującego nas rozwiązania (np. mniejsze ugięcie kosztem większej masy belki).