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.

1. **Przeprowadzenie ćwiczenia**
2. 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);

}

}

1. 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;

}

1. 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;

}

1. 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

}

1. **Parametry algorytmów**

Dokładność (epsilon): 1E-4

Maksymalna ilość wywołań funkcji (Nmax): 10 000

1. **Dyskusja wyników**
2. Wyniki testowe:

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.

1. Wyniki dla problemu rzeczywistego:

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.

1. **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).