Optymalizacja wielokryterialna	
Autorzy	Kozłowski Bartosz, Kopeć Jakub, Kobyłecki Emil
Data wysłania	12.01.2021

1. Wejściowe parametry algorytmu Powella:

```
solution Powell(matrix x0, double epsilon, int Nmax, matrix O)
```

x0 - para punktów, w której przechowywane są początkowe wartości x1 oraz x2,
epsilon - dokładność obliczeń,
Nmax - maksymalna ilość iteracji,
macierz limits - macierz ograniczająca metodę złotego podziału

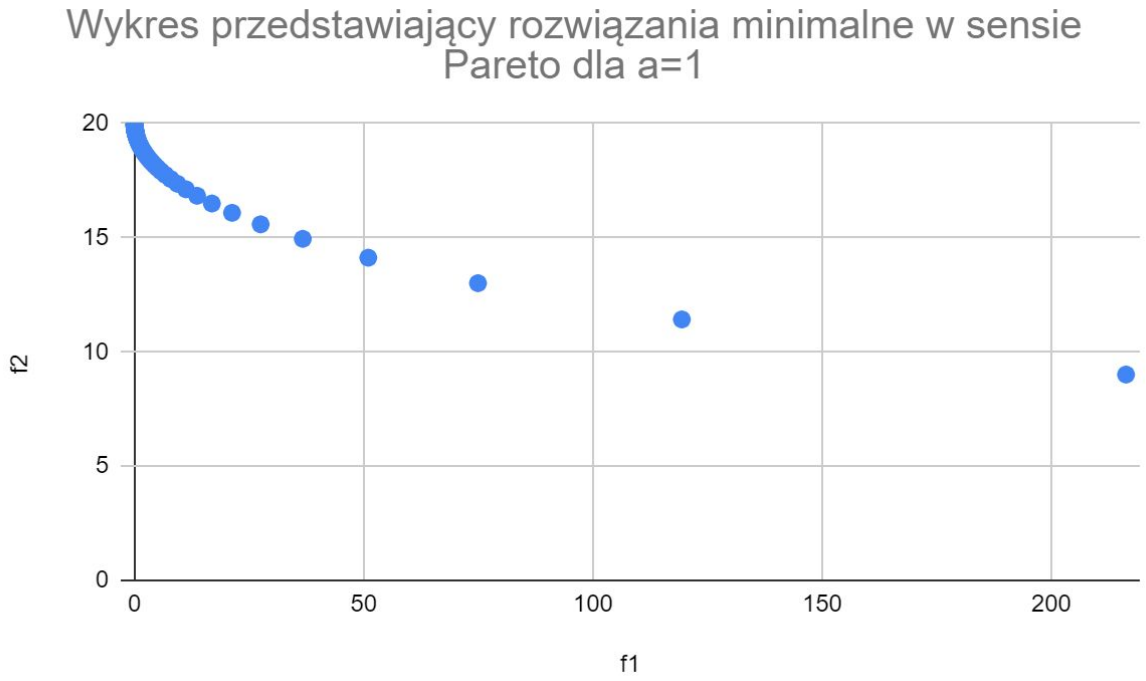
Dla testowej funkcji celu wywołanie funkcji miało postać:

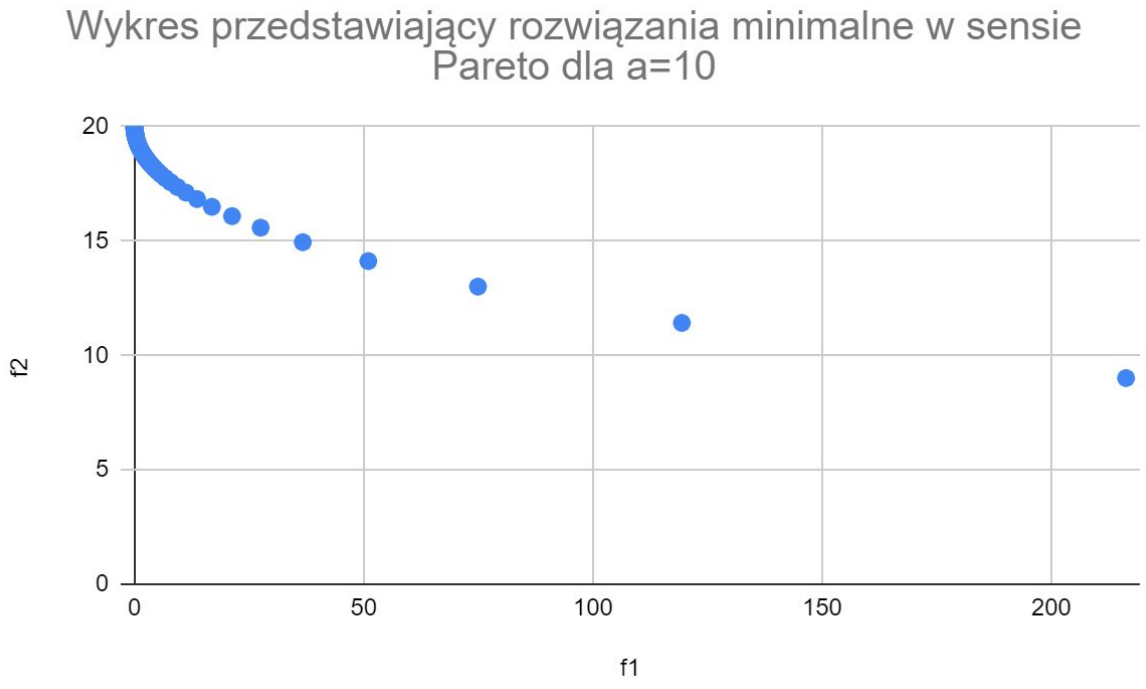
x0 - składniki macierzy są z przedziału <-10;10>,
epsilon = 0,0001,
Nmax = 1000,
O = [-10,10,w]
[-10,10,0]

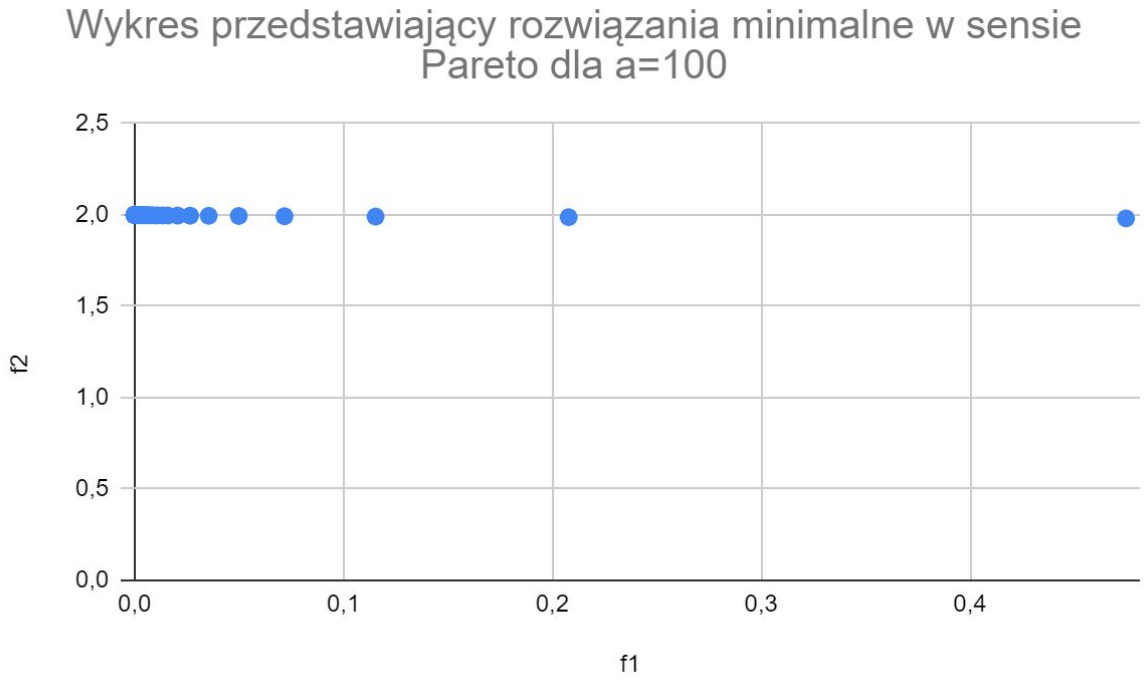
Dla problemu rzeczywistego wywołanie funkcji miało postać:

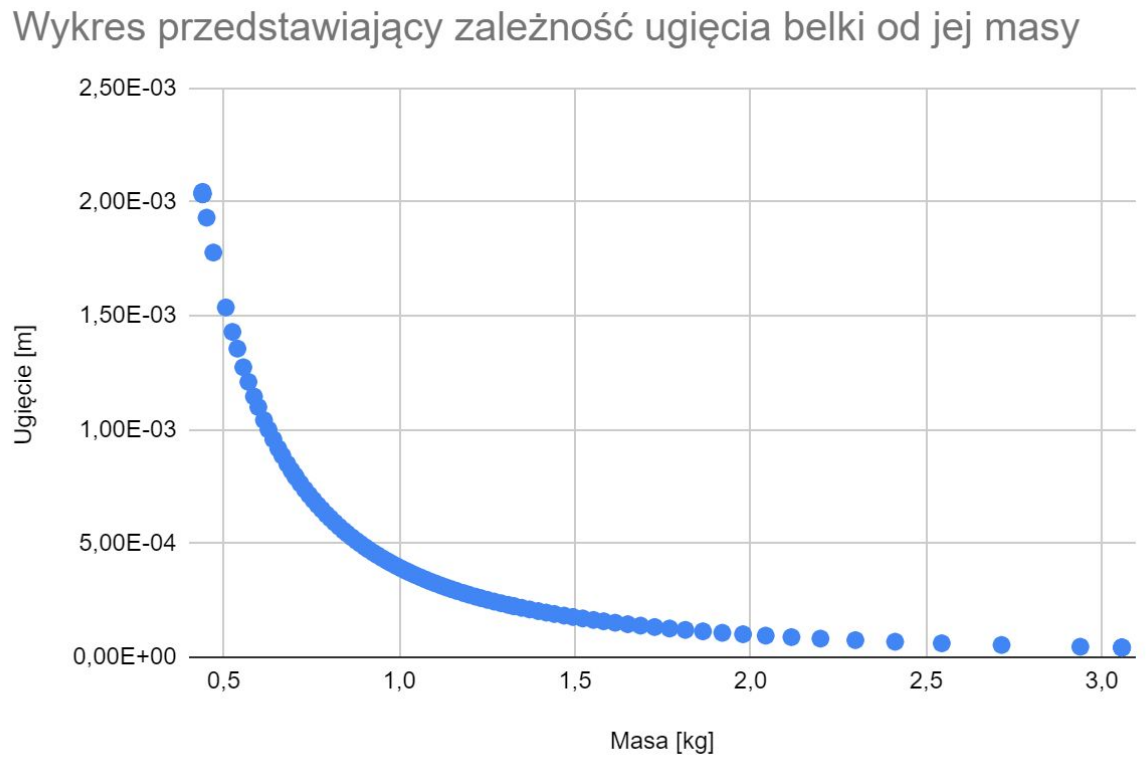
x0 - składniki macierzy są z przedziału <0.2;1>, oraz <0.01;0.05>
epsilon = 0,0001,
Nmax = 5000,
O = [0.2, 1, w]
[0.01, 0.05, 0]

2. Dyskusja wyników oraz wnioski:






Wykres przedstawiający rozwiązania minimalne w sensie Pareto dla a=100
2,5
2,0
1,5
1,0
0,5
0,0
f2
0,0
0,1
0,2
0,3
0,4
f1


Wykres przedstawiający zależność ugięcia belki od jej masy
2,50E-03
2,00E-03
1,50E-03
1,00E-03
5,00E-04
0,00E+00
Ugięcie [m]
0,5
1,0
1,5
2,0
2,5
3,0
Masa [kg]

Zadaniem optymalizacji wielokryterialnej jest znalezienie takiego zbioru wariantów rozwiązań, które możliwie dobrze spełniają kryteria, a jednocześnie spełniają wszystkie ograniczenia. Otrzymane rozwiązania nazywamy rozwiązaniami kompromisowymi, ponieważ optymalna wartość według jednej zmiennej (f1) jest ściśle powiązana z nieoptymalną zmienną (f2). Znalezienie pary rozwiązań (f1,f2), gdzie obie zmienne są optymalne jest niemożliwe.

Wszystkie wykresy przypominają kształtem fragment hiperboli. Wykres przedstawiający zależność f1 od f2 dla współczynnika a równego 100 znacznie się różni od poprzednich wykresów. Zwiększyło się w nim “zagęszczenie” punktów należących do frontu pareto (taka sama ilość punktów na znacznie mniejszym przedziale).

Wartości przyjmowane przez funkcję Powell’a podczas symulacji problemu rzeczywistego są innego rzędu wielkości niż te podane w konspekcie. Wynika to z zamiany rozpatrywanych jednostek z milimetrów na metry. Obserwując uzyskane wyniki oraz wykres łatwo zauważyć, że wraz ze wzrostem trzech jej parametrów (masy,długości,średnicy) odgięcie belki maleje o ponad rząd wielkości .

3. Kod źródłowy:
Metoda Powella:

```
#if LAB_NO>5
solution Powell(matrix x0, double epsilon, int Nmax, matrix O)
{
    int* n = get_size(x0);
    matrix A(n[0], 3), limits(n[0], 2);
    matrix D(n[0], n[0]);
    for (int i = 0; i < n[0]; i++)
    {
        D(i, i) = 1;
    }
    limits = set_col(limits, O[0], 0);
    limits = set_col(limits, O[1], 1);
    A(0, 2) = O(0, 2);
    solution X, P, h;
    X.x = x0;
    double* ab = new double[2];
    while (true)
    {
        P = X;
        for (int i = 0; i < n[0]; ++i)
        {
            A = set_col(A, P.x, 0);
            A = set_col(A, D[i], 1);
            ab = compute_ab(P.x, D[i], limits);
            h = golden(ab[0], ab[1], epsilon, Nmax, A);
            P.x = P.x + h.x * D[i];
        }
        if (norm(X.x - P.x) < epsilon || solution::f_calls > Nmax)
        {
            P.fit_fun();
            return P;
        }
        for (int i = 0; i < n[0] - 1; ++i)
            D = set_col(D, D[i+1], i);
        D = set_col(D, P.x - X.x, n[0]-1);
        A = set_col(A, P.x, 0);
        A = set_col(A, D[n[0] - 1], 1);
        ab = compute_ab(P.x, D[n[0]-1], limits);
        h = golden(ab[0], ab[1], epsilon, Nmax, A);
        X.x = P.x + h.x * D[n[0]-1];
    }
}
```

Metoda złotego podziału:

```
solution golden(double a, double b, double epsilon, int Nmax, matrix O)
{
    double alfa = (sqrt(5.0) - 1.0) / (2.0);
    solution A, B, C, D;
    A.x = a;
    B.x = b;
    C.x = B.x - alfa * (B.x - A.x);
    C.fit_fun(O);
    D.x = A.x + alfa * (B.x - A.x);
    D.fit_fun(O);
    while (true)
    {
        if (C.y < D.y)
        {
            B = D;
            D = C;
            C.x = B.x - alfa*(B.x - A.x);
            C.fit_fun(O);
        }
        else
        {
            A = C;
            C = D;
            D.x = A.x + alfa*(B.x - A.x);
            D.fit_fun(O);
        }
        if (B.x - A.x < epsilon || solution::f_calls > Nmax)
        {
            A.x = (A.x + B.x) / 2.0;
            A.fit_fun(O);
            return A;
        }
    }
}
```

Fit_fun:

```
void solution::fit_fun(matrix O)
{
    //y = x(0) * x(0) + x(1) * x(1);
    int* n = get_size(O);
    if (n[1] == 1)
    {
        y = matrix(2, 1);
        double a = 100.0;
        y(0) = a * ( pow((x(0) - 5), 2) + pow((x(1) - 5), 2) ); //f1
        y(1) = (1 / a) * (pow((x(0) + 5), 2) + pow((x(1) + 5), 2)); //f2
        ++f_calls;
    }
    else
    {
        solution temp;
        temp.x = O[0] + x*O[1];
        temp.fit_fun();
        y = O(0, 2) * temp.y(0) + (1 - O(0, 2)) * temp.y(1);
    }

    //PROBLEM RZECZYWISTY

    //f_1 masa, f_2 ugięcie
    int *n=get_size(O);
    if(n[1]==1)
    {
        y=matrix(3,1);
        double rho=7800,p=1e3,E=207e9;
        y(0)=rho*x(0)*3.14*(pow(x(1),2))/4; //masa = gęstość*wysokość*pi*(d^2)/4
        y(1)=(64*p*pow(x(0),3))/(3*E*3.14*pow(x(1),4)); //ugięcie
        y(2)=(32*p*x(0))/(3.14*pow(x(1),3)); //naprężenie
        f_calls++;
    }
    else
    {
        solution T;
        T.x=O[0]+x*O[1]; // x+alfa*d
        T.fit_fun();

        /* Wyznaczenie f1 i f2 min oraz max
        matrix yn(2,1);
        y=O(0,2)*T.y(0)+(1-O(0,2)*T.y(1));
        */

        //Normalizacja wartości
        matrix yn(2,1);
        yn(0)=(T.y(0)-0.44)/(3.061-0.44);
        yn(1)=(T.y(1)-4.201e-5)/(0.002-4.201e-5);
        y=O(0,2)*yn(0)+(1-O(0,2))*yn(1);

        //Ograniczenia
        if(T.y(1)>0.005)//przekroczenie wartości ugięcia
        {
            y=y+1e6*pow(T.y(1)-0.005,2);
        }
        if(T.y(2)>300e6)//przekroczenie wartości naprężenia
        {
            y=y+1e6*pow(T.y(2)-300e6,2);
        }
        f_calls++;
    }
}
```

Compute_ab:

```
double* compute_ab(matrix x, matrix d, matrix limits)
{
    int* n = get_size(x);
    double* ab = new double[2]{ -1e9,1e9 };
    double ai, bi;
    for (int i = 0; i < n[0]; ++i)
    {
        if (d(i) == 0)
        {
            ai = -1e9;
            bi = 1e9;
        }
        else if (d(i) > 0)
        {
            ai = (limits(i, 0) - x(i)) / d(i);;
            bi = (limits(i, 1) - x(i)) / d(i);
        }
        else
        {
            ai = (limits(i, 1) - x(i)) / d(i);;
            bi = (limits(i, 0) - x(i)) / d(i);
        }
        if (ab[0] < ai)
            ab[0] = ai;
        if (ab[1] > bi)
            ab[1] = bi;
    }
    return ab;
}
#endif
```

Funkcja main:

```cpp
int main()
{
    srand(time(NULL));
    try
    {
ofstream x01("x01.txt");    //punkty startowe
ofstream x02("x02.txt");
ofstream x1("x1.txt");
ofstream x2("x2.txt");
ofstream y1("y1.txt");
ofstream y2("y2.txt");
ofstream fcalls("fcalls.txt");



matrix x0(2, 1);
x0(0) = fRand(-10, 10);
x0(1) = fRand(-10, 10);
matrix limits(2, 3);
limits(0, 0) = limits(1, 0) = -10;
limits(0, 1) = limits(1, 1) = 10;
limits(1, 2) = 0;
double w = 0;
solution A;
for (int i = 0; i < 101; i++)
{
    cout << "W = " << w << endl;
    limits(0, 2) = w;// limits dla rzeczywistego: l(200,1000) [mm], d(10,50) [mm]
    x0(0) = fRand(-10, 10);
    x0(1) = fRand(-10, 10);
    x01 << x0(0) << endl;
    x02 << x0(1) << endl;
    A = Powell(x0, 0.0001, 1000, limits);
    x1 << A.x(0) << endl;
    x2 << A.x(1) << endl;
    y1 << A.y(0) << endl;
    y2 << A.y(1) << endl;
    fcalls << A.f_calls << endl;
    w += 0.01;
    solution::clear_calls();
}
```

```
//PROBLEM RZECZYWISTY
    matrix x0(2,1);
    matrix Q(2,3);
    int Nmax=5000;
    double epsilon=0.0001;
    double w=0;
    solution wynik;

    Q(0,0)=0.2;
    Q(0,1)=1;
    Q(1,0)=0.01;
    Q(1,1)=0.05;

    for (int i=0;i<101;i++)
    {
        double a=(double) (rand()%9);
        double b=(double) (rand()%5);
        x0(0)=(a/10)+0.2;

        //cout<<"x0(0): "<<x0(0)<<endl;
        x0(1)=(b/100)+0.01;
        //cout<<"x0(1): "<<x0(1)<<endl;


        Q(0,2)=w;
        //Q(1,2)=0;

        wynik=Powell(x0,epsilon,Nmax,Q);
        //Wypisywanie danych- początkowa długość i średnica,
        //końcowa długość i średnica,masa,ugięcie,wywołania funkcji
        cout<<x0(0)<<" "<<x0(1)<<" "<<wynik.x(0)<<" "<<wynik.x(1)
        <<" "<<wynik.y(0)<<" "<<wynik.y(1)<<" "<<solution::f_calls<<endl;
        solution::clear_calls();
        w+=0.01;
    }


        cout << "LAB NUMBER " << LAB_NO << endl;
        cout << "LAB PART " << LAB_PART << endl << endl;
    }
    catch (char const* EX_INFO)
    {
        cout << EX_INFO << endl;
    }
    //system("pause");
    return 0;
}
```