Informatyka Techniczna Grupa 5	AGH Optymalizacja metodami niedeterministycznymi	Dominika Lazarowicz Jakub Litewka Tomasz Ligęza

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z niedeterministycznymi metodami optymalizacji poprzez ich implementację oraz wykorzystanie do wyznaczenia minimum podanej funkcji celu.

Wstęp teoretyczny

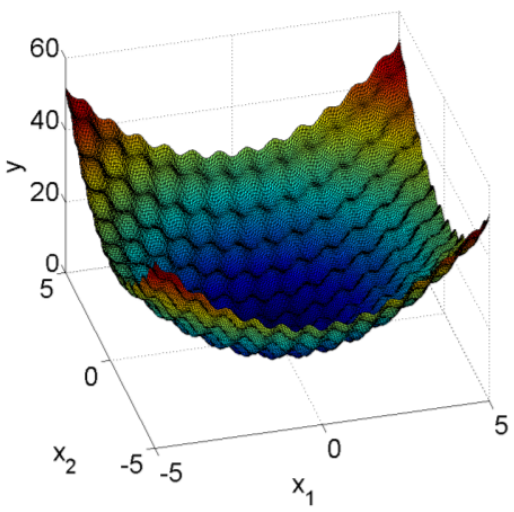
W informatyce, strategia ewolucyjna jest techniką optymalizacji opartą na idei ewolucji. Należy ona do ogólnej klasy obliczeń ewolucyjnych lub metodologii sztucznej ewolucji.

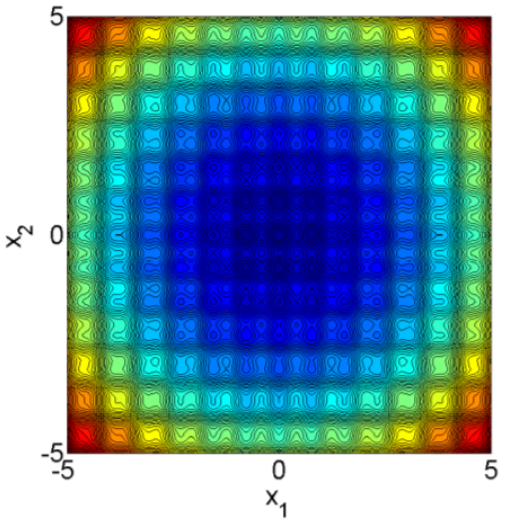
Strategie ewolucyjne wykorzystują naturalne odwzorowania zależne od problemu, a przede wszystkim mutację i selekcję, jako operatory wyszukiwania. Podobnie jak w algorytmach ewolucyjnych, operatory te są stosowane w pętli. Iteracja pętli nazywana jest generacją. Sekwencja pokoleń jest kontynuowana do momentu spełnienia kryterium zakończenia.

Samodzielnie wykonane zadania

Naszym zadaniem było znalezienie minimum globalnego dla funkcji:

$$f(x_1, x_2) = x_1^2 + x_2^2 - cos(2{,}5\pi x_1) - cos(2{,}5\pi x_2) + 2$$



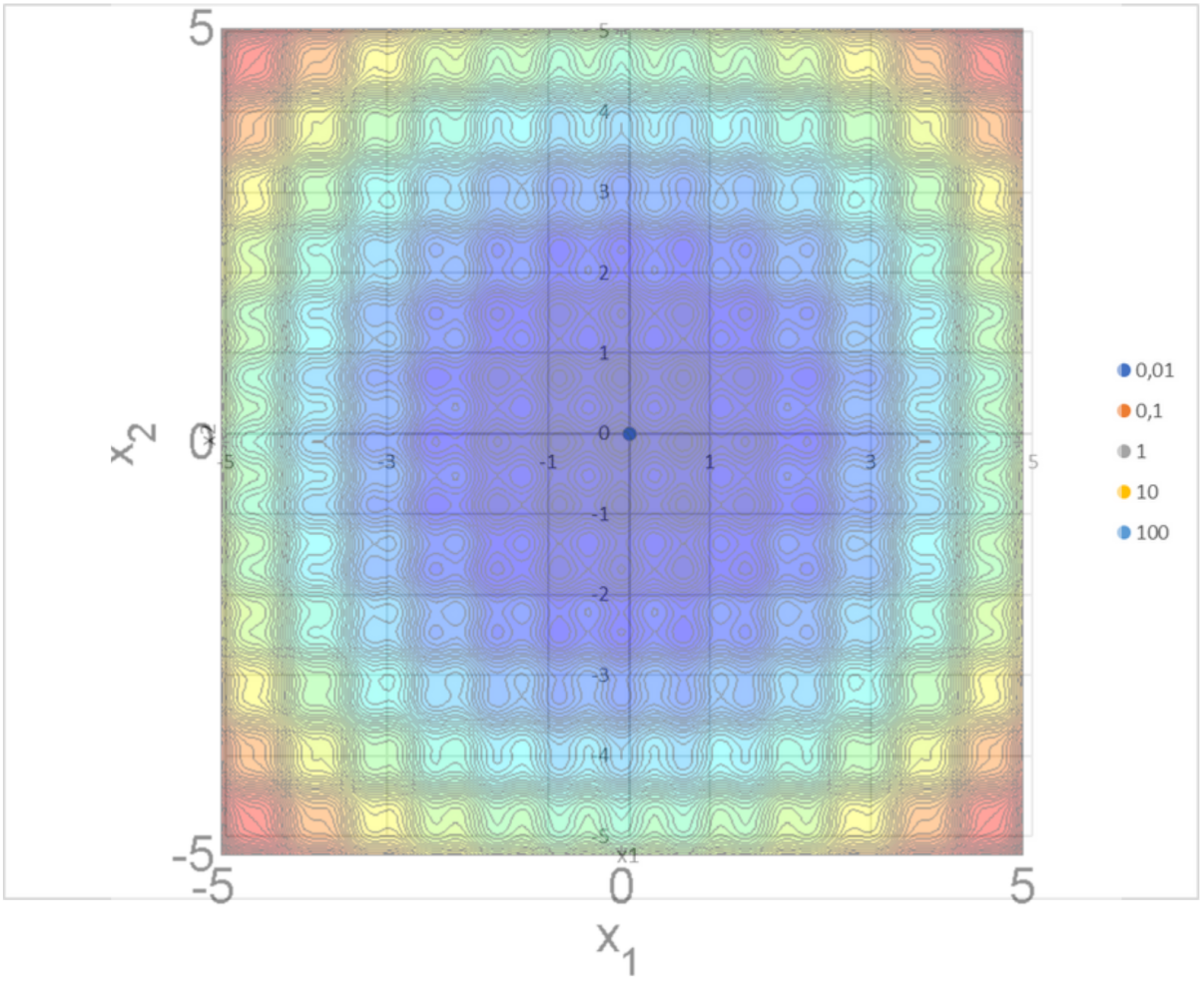


Analiza danych

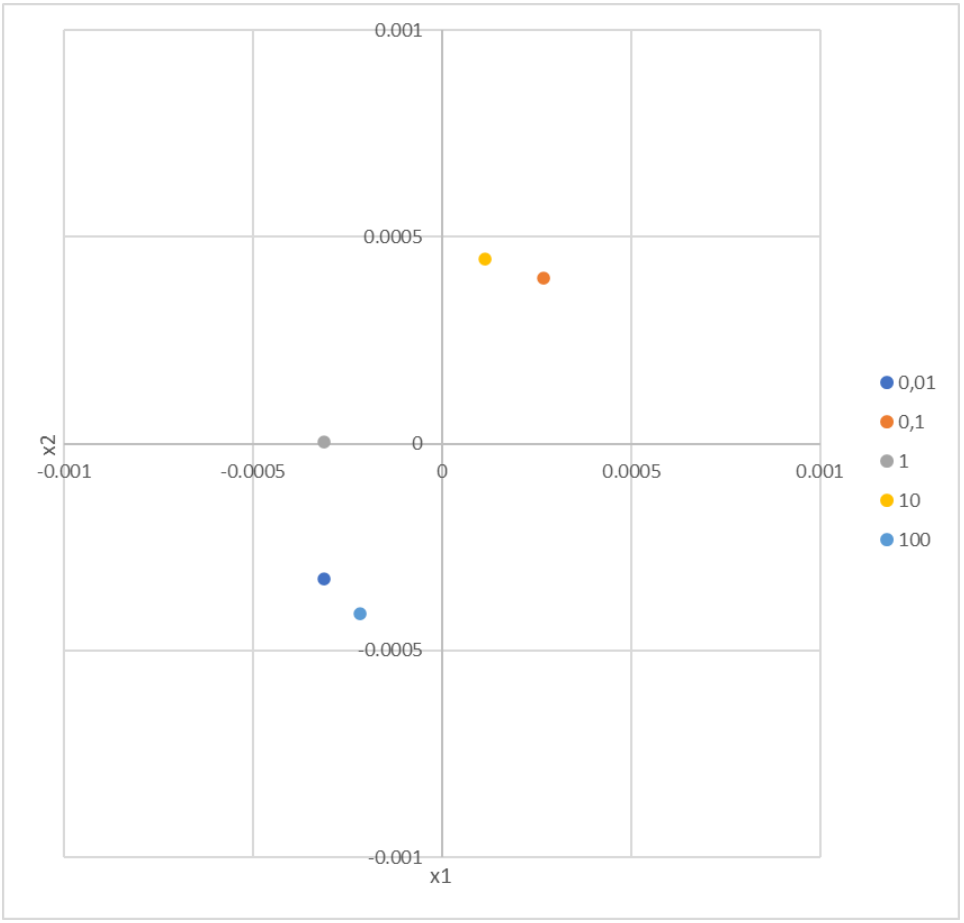
Początkowa wartość zakresu mutacji	x1*	x2*	y*	Liczba wywołań funkcji celu	Liczba minimów globalnych
0,01	-0,000311624	-0,000325989	0,000502026	316,690140845	71
0,1	0,000268738	0,000400724	0,000558714	409,163043478	92
1	-0,000312828	0,000003920	0,000511324	802,188118812	101
10	0,000112263	0,000447664	0,000545186	1428,479591837	98
100	-0,000218202	-0,000411018	0,000520854	3584,142857143	42

Jak można zauważyć na podstawie tabeli powyżej najmniej znalezionych minimów globalnych jest dla początkowego zakresu 100. Pozostałe wyniki są do siebie zbliżone. Im wyższa początkowa wartość zakresu tym większa liczba wywołań funkcji celu.

Otrzymane wyniki bardzo dokładnie pokrywają się z wykresem funkcji co można zaobserwować na wykresach poniżej.



Wszystkie z punktów są na tyle blisko siebie, że możemy jedynie dostrzec jeden wspólny punkt. Wykres w powiększeniu prezentujemy poniżej. Skok na osiach wynosi tam 0.0005 jednostki.


0.001
0.0005
0
-0.0005
-0.001
x2
-0.001
-0.0005
0
0.0005
0.001
x1
0,01
0,1
1
10
100


Liczba minimów globalnych dla różnych wartości początkowych mutacji
120
100
80
60
40
20
0
0,01
0,1
1
10
100
Liczba minimów globalnych

Problem rzeczywisty

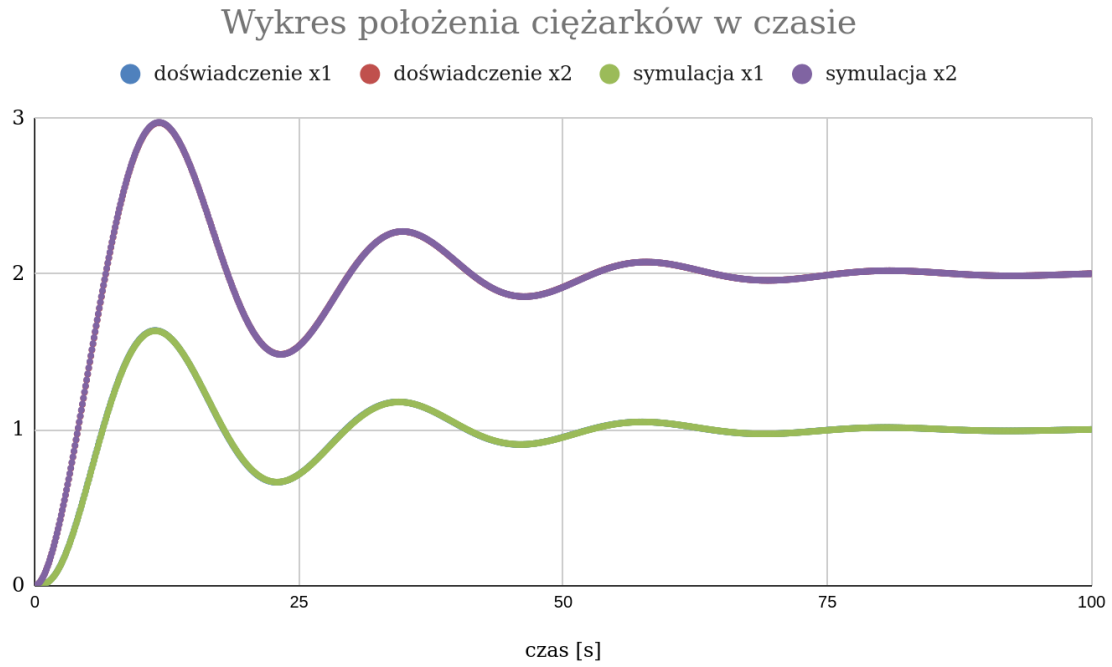
Problemem rzeczywistym było przeprowadzenie jednej optymalizacji dla problemu ciężarków.

Równania opisujące ruch ciężarków są następujące:

$$\begin{cases} m_1\ddot{x}_1 + b_1\dot{x}_1 + b_2(\dot{x}_1 - \dot{x}_2) + k_1x_1 + k_2(x_1 - x_2) = 0 \\ m_2\ddot{x}_2 - b_2(\dot{x}_1 - \dot{x}_2) - k_2(x_1 - x_2) = F \end{cases}$$

Na podstawie danych wejściowych w postaci położenia ciężarków w czasie byliśmy w stanie wyznaczyć wartości współczynników b_1 oraz b_2. Uzyskane wyniki prezentujemy w tabeli:

b1*	b2*	y*	Liczba wywołań funkcji celu
1,21829	2,04914	0,000966761	1481



Wnioski

Algorytm strategii ewolucyjnej świetnie sprawdza się w wyznaczaniu minimum funkcji. W przypadku początkowej wartości procesu mutacji równej 1 osiągnęliśmy 100% skuteczność, co jest wynikiem zdumiewającym. Im sigma bliższa wartości równej 1, tym skuteczność algorytmu jest lepsza.

Problem rzeczywisty udało nam się zrealizować poprzez najpierw uzyskanie współczynników b_1 oraz b_2 odpowiadających za opór ruchu, po czym użycie ich w symulacji dla czasu

t_0 = 0s, dt = 0,1s oraz t_{end}=100s.

Wynik symulacji przedstawiliśmy na powyższym wykresie. Jak widać położenie ciężarków wynikłe z symulacji pokrywa się idealnie z tym uzyskanym poprzez przeprowadzenie eksperymentu.

Kod

Main

```
#elif LAB_NO == 7 && LAB_PART == 1
	int N = 2, Nmax = 5000, mi = 20, lambda = 40;
	double sigmaValue = 0.01;
	ofstream sout("tabela1_k6.csv");
	for (int i = 0; i < 5; i++) {
		sout << "sigma = " << sigmaValue << "\n";
		for (int j = 0; j < 100; j++) {
			double epsilon = 1e-3;
			matrix limits(2, 2), sigma0(2, 1);
			limits(0, 0) = limits(1, 0) = -5;
			limits(0, 1) = limits(1, 1) = 5;
			sigma0(0) = sigma0(1) = sigmaValue;
			solution optEA = EA(N, limits, mi, lambda, sigma0, epsilon, Nmax);
			sout << optEA.x(0) << "\t" << optEA.x(1) << "\t" << optEA.y(0) << "\t" << solution::f_calls << "\n";
			solution::clear_calls();
		}
		sigmaValue *= 10.0;
	}
	sout.close();
#elif LAB_NO == 7 && LAB_PART == 2
	//parametry poczatkowe
	int N = 2, Nmax = 5000, mi = 20, lambda = 40;
	double epsilon = 1e-3;
	matrix limits(2, 2), sigma0(2, 1);
	limits(0, 0) = limits(1, 0) = 0.1;
	limits(0, 1) = limits(1, 1) = 3;
	sigma0(0) = sigma0(1) = 10;
	//optymalizacja
	solution optEA = EA(N, limits, mi, lambda, sigma0, epsilon, Nmax);
	ofstream sout("symulacja.csv");
	sout << optEA.x(0, 0) << ";" << optEA.x(1, 0) << ";" << optEA.y(0) << ";" << solution::f_calls << endl;
	solution::clear_calls();
	matrix Y0(4, new double[4]{0, 0, 0, 0 });
	matrix ud(2, 1);
	ud(0) = optEA.x(0, 0);
	ud(1) = optEA.x(1, 0);
	//symulacja
	matrix* R = solve_ode(0, 0.1, 100, Y0, &ud);
	matrix result = hcat(R[1][0], R[1][2]);
	sout << result << endl;
	sout.close();
```

fit_fun

```
#elif LAB_NO == 7 && LAB_PART == 1
   y = pow(x(0), 2) + pow(x(1), 2) - cos(2.5 * 3.14 * x(0)) - cos(2.5 * 3.14 * x(1)) + 2;
#elif LAB_NO == 7 && LAB_PART == 2
   int N = 1001;
   static matrix X(N, 2);
   if (solution::f_calls == 1) {
      ifstream S("polozenia.txt");
      S >> X;
      S.close();
   }
   matrix Y0(4, new double[4]{ 0,0,0,0 });
   matrix* Y = solve_ode(0, 0.1, 100, Y0, &x);
   y = 0;
   for (int i = 0; i < N; ++i) {
      y = y + abs(X(i, 0) - Y[1](i, 0)) + abs(X(i, 1) - Y[1](i, 2));
   }
   y = y / (2 * N);
#endif
```

diff

```
      #elif LAB_NO==7 && LAB_PART==2
      double m1 = 5, m2 = 5, k1 = 1, k2 = 1, F = 1;
      double b1 = (*ud)(0), b2 = (*ud)(1);
      matrix dY(4, 1);

      dY(0) = Y(1);
      dY(1) = (-b1 * Y(1) - b2 * (Y(1) - Y(3)) - k1 * Y(0) - k2 * (Y(0) - Y(2))) / m1;
      dY(2) = Y(3);
      dY(3) = (F + b2 * (Y(1) - Y(3)) + k2*(Y(0) - Y(2))) / m2;
      return dY;
```