Sprawozdanie z laboratoriów z przedmiotu optymalizacja

Aleksandra Zakręcka, Łucja Wuls, Wojciech Starzyk

Laboratoria 1 (31.10-8.11.2024)

Cel:

W ramach ćwiczeń zapoznaliśmy się z wykorzystaniem i implementacją metod bezgradientowych przy rozwiązywaniu problemu optymalizacyjnego.

Wykorzystane algorytmy:

Metoda Hook'a-Jeevesa - stosuje się go do bezgradientowego znajdowania minimum funkcji. Wykorzystuje funkcję celu *(fitness function)* w kolejnych punktach, aby określić, czy osiągnięto lepszy wynik i w którym kierunku kontynuować poszukiwania. Na początku metoda ustala punkt początkowy x0 i przeprowadza ekspansję otoczenia, przesuwając się o krok poszukiwania s. O kierunku ruchu decyduje funkcja celu, jeśli wartość funkcji w nowym punkcie jest lepsza, metoda wykonuje krok kierunkowy, a jeśli nie to krok s jest zmniejszany poprzez pomnożenie przez współczynnik alfa. Metoda kończy działanie po znalezieniu punktu, w którym wartość funkcji przestaje się poprawiać, co oznacza osiągnięcie lokalnego minimum albo osiągnięcia maksimum iteracji.

```
solution HJ(matrix(*ff)(matrix, matrix, matrix), matrix x0, double
s, double alpha, double epsilon, int Nmax, matrix ud1, matrix ud2)
{
	try
	{
		solution Xopt;
		//Tu wpisz kod funkcji
		Xopt.clear_calls();
		solution XB, XB_temp, X(x0);
		do {
			XB.x = X.x;
			XB.fit_fun(ff);
			X = HJ_trial(ff, XB, s);
			if (X.y < XB.y) {
				do {
					XB_temp = XB;
					XB = X;
					X.x = 2 * XB.x - XB_temp.x;
					X.fit_fun(ff);
					X = HJ_trial(ff, X, s);
```

```
                        if (X.f_calls > Nmax) {
                              Xopt.flag = 0;
                              return Xopt;
                        }
                  } while (X.y < XB.y);
                  X = XB;
            }
            else {
                  s = alpha * s;
            }
            if (X.f_calls > Nmax) {
                  Xopt.flag = 0;
                  return Xopt;
            }
      } while (s > epsilon);

      Xopt.x = XB.x;
      Xopt.y = XB.y;
      Xopt.flag = 1;
      return Xopt;
   }
   catch (string ex_info)
   {
      throw ("solution HJ(...):\n" + ex_info);
   }
}
```

Parametry:

- matrix(*ff)(matrix, matrix, matrix): Wskaźnik do funkcji oceny (fitness function), która przyjmuje trzy argumenty typu matrix i zwraca wynik oceny.
- matrix x0: Punkt początkowy, od którego rozpoczynana jest procedura optymalizacji.
- double s: Początkowa długość kroku poszukiwania w przestrzeni rozwiązań.
- double alpha: Współczynnik skalowania kroku, który określa, jak szybko zmienia się długość kroku po nieudanej próbie znalezienia lepszego rozwiązania.
- double epsilon: Minimalna akceptowalna długość kroku, która służy jako kryterium zakończenia algorytmu (oznacza osiągnięcie odpowiedniej dokładności).
- int Nmax: Maksymalna liczba dozwolonych wywołań funkcji oceny, po której osiągnięciu algorytm przerywa działanie, zwracając aktualne rozwiązanie.
- matrix ud1: Dodatkowy parametr typu matrix, który może być przekazywany do funkcji oceny w celu dostarczenia dodatkowych danych wejściowych.

- matrix ud2: Dodatkowy parametr typu matrix, który może być przekazywany do funkcji oceny w celu dostarczenia dodatkowych danych wejściowych.

Metoda Rosenbrocka - metodę tą stosuje się do znajdowania lokalnego minimum funkcji wielowymiarowych. Algorytm rozpoczyna się od zainicjalizowania punktu początkowego x0 i macierzy kierunków d, która początkowo jest macierzą jednostkową, czyli poszukiwanie zaczyna się wzdłuż osi układu współrzędnych. W kolejnych iteracjach algorytm przeszukuje kolejne kierunki z macierzy d, przesuwając się o krok s(j). Jeśli nowy punkt poprawia wynik funkcja celu, aktualizuje punkt bazowy XB i powiększa krok poprzez pomnożenie go przez alfa. Jeśli się pogarsza to krok jest zmniejszany o minus beta. Algorytm kończy się, gdy krok s jest mniejsze od zadanego epsilonu albo przekroczy maksymalną liczbę iteracji.

```
solution Rosen(matrix(*ff)(matrix, matrix, matrix), matrix x0,
matrix s0, double alpha, double beta, double epsilon, int Nmax,
matrix ud1, matrix ud2)
{
	try
	{
		//Tu wpisz kod funkcji
		solution::clear_calls();
		//int i = 0;
		int n = get_len(x0);
		matrix d(n, n);
		for (int j = 0; j < n; j++) {
			d(j, j) = 1.0;
		}
		matrix lamda(n, 1), p(n, 1), s(s0);
		solution XB, X_temp;
		XB.x = x0;
		XB.fit_fun(ff);

		while (true) {

			//zacznijmy szukać rozwiązania zgodnie z zadaną
bazą
			for (int j = 0; j < n; j++) {
				X_temp.x = XB.x + (s(j) * d[j]);
				X_temp.fit_fun(ff);
				if (X_temp.y(0) < XB.y(0)) {
					XB = X_temp;
					lamda(j) += s(j);
```

```
                        s(j) *= alpha;
                    }
                    else {
                        p(j) = p(j) + 1;
                        s(j) *= -beta;
                    }
                }

                //czy któryś z kroków pogorszył sytuacje?
                bool change = true;
                for (int j = 0; j < n; j++) {
                    if (p(j) == 0 || lamda(j) == 0)
                    {
                        change = false;
                        break;
                    }

                }

                //zmiana bazy jeżeli jest to konieczne
                if (change)
                {
                    matrix Q(n, n), v(n, 1);
                    for (int i = 0; i < n; ++i)
                        for (int j = 0; j <= i; ++j)
                            Q(i, j) = lamda(i);

                    Q = d * Q;
                    v = Q[0] / norm(Q[0]);
                    d.set_col(v, 0);
                    for (int i = 1; i < n; ++i)
                    {
                        matrix temp(n, 1);
                        for (int j = 0; j < i; ++j)
                            temp = temp + (trans(Q[i]) *
d[j]) * d[j];
                        v = Q[i] - temp;
                        d.set_col(v, i);
                    }
                    s = s0;
                    lamda = matrix(n, 1);
                    p = matrix(n, 1);
```

```
                }

                //czy wynik jest wystarczająco dokładny?
                double max_s = abs(s(0));
                for (int i = 1; i < n; ++i) {
                      if (max_s < abs(s(i))) {
                            max_s = abs(s(i));
                      }
                }
                if (max_s < epsilon || solution::f_calls > Nmax) {
                      return XB;
                }
          }


     }
     catch (string ex_info)
     {
          throw ("solution Rosen(...):\n" + ex_info);
     }
}
```

Parametry:

- matrix(*ff)(matrix, matrix, matrix): Wskaźnik do funkcji oceny (fitness function), która przyjmuje trzy argumenty typu matrix i zwraca wynik oceny.
- matrix x0: Punkt początkowy, od którego rozpoczynana jest procedura optymalizacji.
- matrix s0: Początkowa macierz kroków, która definiuje wielkość i kierunek początkowych przemieszczeń.
- double alpha: Współczynnik skalowania kroku, który określa, jak szybko zmienia się krok, gdy ruch prowadzi do poprawy wyniku.
- double beta: Współczynnik zmiany znaku kroku, stosowany, gdy ruch nie prowadzi do poprawy wyniku (odwrócenie kierunku).
- double epsilon: Minimalna akceptowalna wartość kroku, która służy jako kryterium zakończenia algorytmu (oznacza osiągnięcie odpowiedniej dokładności).
- int Nmax: Maksymalna liczba dozwolonych wywołań funkcji oceny, po której osiągnięciu algorytm przerywa działanie, zwracając aktualne rozwiązanie.
- matrix ud1: Dodatkowe parametry typu matrix, które mogą być przekazywane do funkcji oceny w celu dostarczenia dodatkowych danych wejściowych.
- matrix ud2: Dodatkowe parametry typu matrix, które mogą być przekazywane do funkcji oceny w celu dostarczenia dodatkowych danych wejściowych.

Funkcja *Lab 2*:

```
void lab2()
{
	srand(time(NULL));
	std::ofstream Sout("symulacja_lab2.csv");

	matrix X;
	double step = 0.01, alpha = 0.8, beta = 0.1, epsilon =
0.0001;
	double a, b;
	int Nmax = 1000;

	// zadanie teoretyczne

	for (int i = 0; i < 100; i++)
	{
		a = ((rand() % 200) / 100.0) - 1;
		b = ((rand() % 200) / 100.0) - 1;
		alpha = 0.8;
		X = matrix(2, new double[2] {a, b});
		solution hooke = HJ(ff3T, X, step, alpha, epsilon,
Nmax);
		//Sout << "x" << a << ";" << "x" << b << ";" << "x" <<
hooke.x(0) << ";" << "x" << hooke.x(1) << ";" << "x" << hooke.y <<
";" << "x" << solution::f_calls << ";";
		//cout << hooke;


		alpha = 1.8;
		matrix Step = matrix(2, new double[2] { step, step});
		solution rosen = Rosen(ff3T, X, Step, alpha, beta,
epsilon, Nmax);
		//Sout << "x" << rosen.x(0) << ";" << "x" << rosen.x(1)
<< ";" << "x" << rosen.y << ";" << "x" << solution::f_calls <<
"\n";
		//cout << rosen;
	}
	//problem rzeczywisty

	//sprawdzenie poprawnosci
	matrix x(2, 1, 5); // macierz 2x1 wypelniona wartoscia 5
	cout << ff3R(x);
```

```
	X = matrix(2, new double[2] {5, 5});
	solution wynikHJ = HJ(ff3R, X, step, alpha, epsilon, Nmax);
	//cout << wynikHJ;

	alpha = 1.8;
	matrix Step = matrix(2, new double[2] { step, step});
	solution wynikR = Rosen(ff3R, X, Step, alpha, beta, epsilon,
Nmax);
	//cout << wynikR;

	//symulacja

	double t0 = 0.0;
	double tend = 100.0;
	double dt = 0.1;

	// Initial conditions for the state vector Y
	matrix Y0(2, 1);
	Y0(0) = 0.0;
	Y0(1) = 0.0;

	// Reference values for the desired angle and angular
velocity
	matrix ud1(2, 1);
	ud1(0) = 3.14;
	ud1(1) = 0.0;

	// Gain parameters, k1 and k2, within the range [0, 10]
	matrix ud2(2, 1);
	//ud2(0) = wynikHJ.x(0);
	ud2(0) = wynikR.x(0);
	//ud2(1) = wynikHJ.x(1);
	ud2(1) = wynikR.x(1);

	matrix* result = solve_ode(df3, t0, dt, tend, Y0, ud1, ud2);

	int n = get_len(result[0]);
	std::cout << "Time\tAngle\tAngular Velocity" << std::endl;
	for (int i = 0; i < n; i++) {
		std::cout << result[0](i) << "\t"
			<< result[1](i, 0) << "\t"
```

```
                << result[1](i, 1) << std::endl;
      }

      std::ofstream file("symulacja_lab2.csv");
      file << "Czas;Kat;Predkosc katowa\n";
      for (int i = 0; i < n; ++i) {
            file << result[0](i) << "x;" << result[1](i, 0) << "x;"
<< result[1](i, 1) << "x\n";
      }
      file.close();

}
```

Testowa funkcja celu:

```
matrix ff3T(matrix x, matrix ud1, matrix ud2) {
      double x1 = x(0);
      double x2 = x(1);

      double result = pow(x1, 2) + pow(x2, 2) - cos(2.5 * 3.14 *
x1) - cos(2.5 * 3.14 * x2) + 2;

      return matrix(1, 1, result);
}
```

Przyjęliśmy trzy długości kroku:

- 0,01
- 0,042
- 0,0809

Po czym dwoma metodami wykonano po sto optymalizacji startując z losowych punktów.

Dla tych samych punktów startowych oraz długości kroków, zarówno metoda Hooke’a-Jeevesa, jak i Rosenbrocka osiągają takie same rezultaty podczas znajdowania minimum globalnego. Jednak różne są wartości funkcji celu oraz liczba wywołań. Ogólnie metoda Rosenbrocka wymaga mniejszej liczby wywołań, co widać na przykład w poniższym wycinku tabeli:

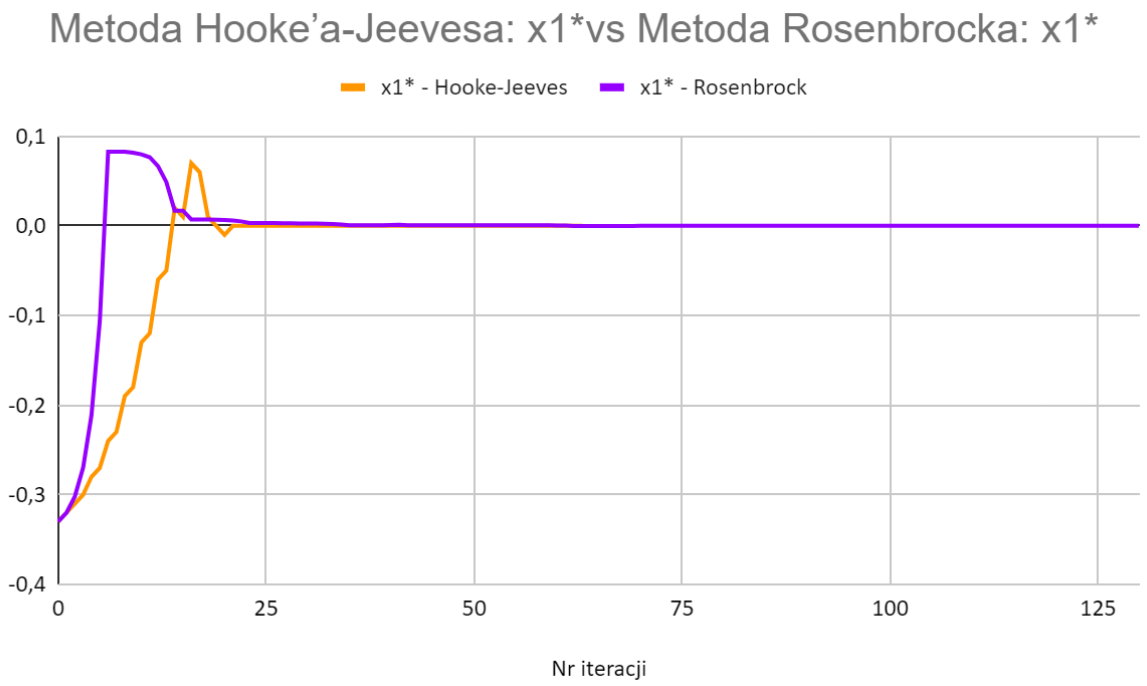
Lp.	x1(0)	x2(0)	Metoda Hooke'a-Jeevesa					Metoda Rosenbrocka				
			x1*	x2*	y*	Liczba wywołań funkcji celu	Minimum globalne [TAK/NIE]	x1*	x2*	y*	Liczba wywołań funkcji celu	Minimum globalne [TAK/NIE]
65	-0,28	0,98	0,00003950	0,77504400	0,62044600	333	NIE	-0,00000001	0,77508300	0,62044600	187	NIE
66	0,35	-0,44	0,00003950	-0,77504000	0,62044600	314	NIE	-0,00000011	-0,77508000	0,62044600	175	NIE
67	-0,15	0,56	-0,00004950	0,77504400	0,62044600	391	NIE	-0,00000001	0,77508300	0,62044600	217	NIE
68	-0,81	-0,66	-0,77503000	-0,77503000	1,24089000	304	NIE	-0,77508000	-0,77508000	1,24089000	201	NIE
69	0,78	0	0,77512500	0,00000000	0,62044600	302	NIE	0,77580000	0,00000000	0,62046200	9	NIE
70	-0,71	-0,2	-0,77512000	0,00004960	0,62044600	389	NIE	-0,77508000	-0,00000003	0,62044600	159	NIE
71	0,48	-0,29	0,77503500	0,00004950	0,62044600	308	NIE	0,77508300	0,00000006	0,62044600	185	NIE
72	0,3	0,88	0,00004050	0,77504500	0,62044600	344	NIE	0,00000000	0,77508300	0,62044600	303	NIE

Powyższa tabela to fragment wywołań dla kroku o wartości 0,042. Różnicę w liczbie wywołań funkcji celu widać na przykład w linii, której liczba porządkowa jest równa 66. Dla metody Hooke'a-Jeevesa liczba wynosi 314, zaś dla Rosenbrocka tylko 175. Otrzymane wyniki sugerują także, że dłuższy krok nie zawsze jest optymalny, ponieważ w niektórych przypadkach mniejszy krok pozwolił na dokładniejsze zbliżenie się do minimum globalnego. Trzeba jednak pamiętać, że obliczenia zostały przygotowane dla losowych punktów startu, a co za tym idzie wyniki są ciężkie do porównania pomiędzy sobą.

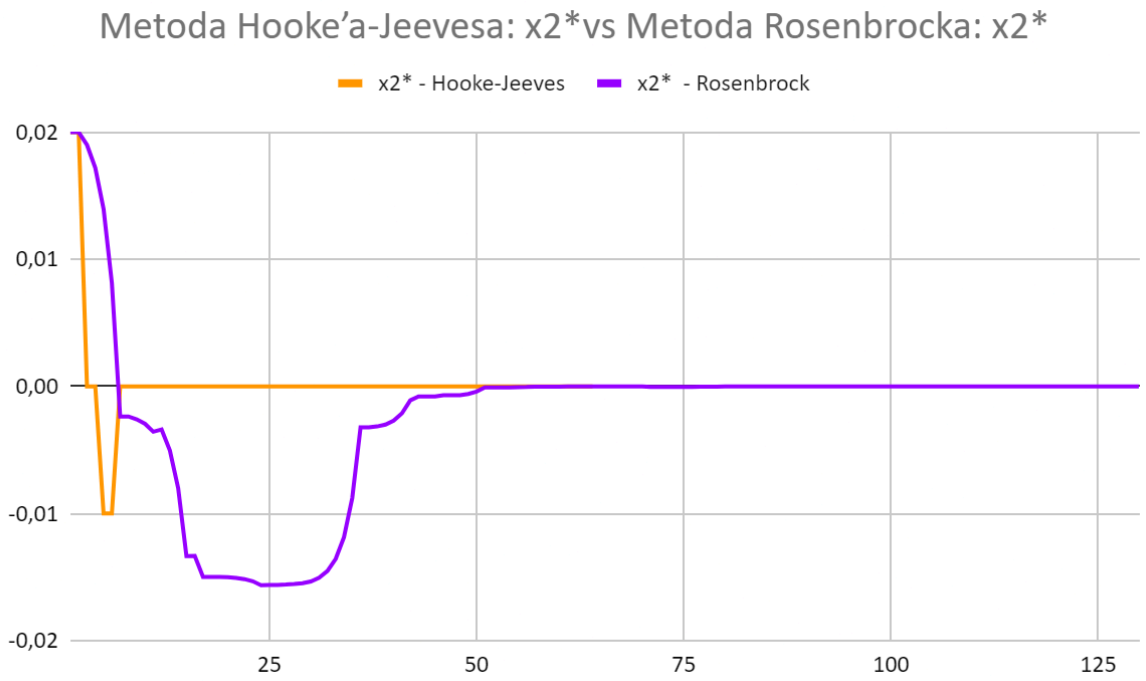
Długość kroku	Metoda Hooke'a-Jeevesa					Metoda Rosenbrocka				
	x1*	x2*	y*	Liczba wywołań funkcji celu	Liczba minimów globalnych	x1*	x2*	y*	Liczba wywołań funkcji celu	Liczba minimów globalnych
0,01	-0,04650840	0,10076820	0,66387666	228	21	-0,04650556	0,10072775	0,66388008	188	21
0,42	-0,03100362	-0,10851180	0,78176120	355	13	-0,03099571	-0,10849773	0,78176185	204	13
0,0809	0,01549300	-0,03099789	0,68249001	381	21	0,01551487	-0,03098681	0,68249166	217	21

Tabela ta pokazuje jak przedstawiają się wyniki w zależności od długości kroku, dzięki czemu można względnie ocenić skuteczność i efektywność obu metod optymalizacyjnych.

Metoda Rosenbrocka wymaga mniejszej liczby wywołań funkcji celu w porównaniu do metody Hooke'a-Jeevesa dla wszystkich trzech długości kroków. Różnice między wartościami x1*, x2* oraz y* są minimalne, co wskazuje, że obie techniki są efektywne. Obie metody znajdują identyczną liczbę minimów dla każdej długości kroków
Dla większej długości kroku liczba wywołań funkcji celu jest większa, co sugeruje, że dłuższe kroki mogą nie być optymalne dla dokładności i efektywności poszukiwania minimum. Jednak musimy pamiętać, że ciężko mówić o porównywaniu wyników między różnymi długościami kroków, ponieważ punkty startowe były losowane.



Metoda Rosenbrocka osiąga stabilną wartość x1* niemal natychmiast, z czego wynika, że charakteryzuje się ona szybką zbieżnością. Natomiast metoda Hooke'a-Jeevesa potrzebuje znacznie więcej iteracji, aby osiągnąć stabilność wartości x1*. Początkowo obserwujemy gwałtowne zmiany, co sugeruje, że metoda ta ma wolniejszą zbieżność i jest bardziej podatna na niestabilności na początku optymalizacji.
Po około 25 iteracjach wartość x1* w metodzie Hooke'a-Jeevesa stabilizuje się, a wyniki końcowe dla obu metod są bardzo podobne, co pokazuje, że obie metody osiągają podobne wyniki, jednak w różnym czasie.
Metoda Rosenbrocka pokazuje lepszą wydajność pod względem szybkości zbieżności i stabilności, podczas gdy metoda Hooke'a-Jeevesa potrzebuje więcej iteracji, aby osiągnąć podobny wynik.

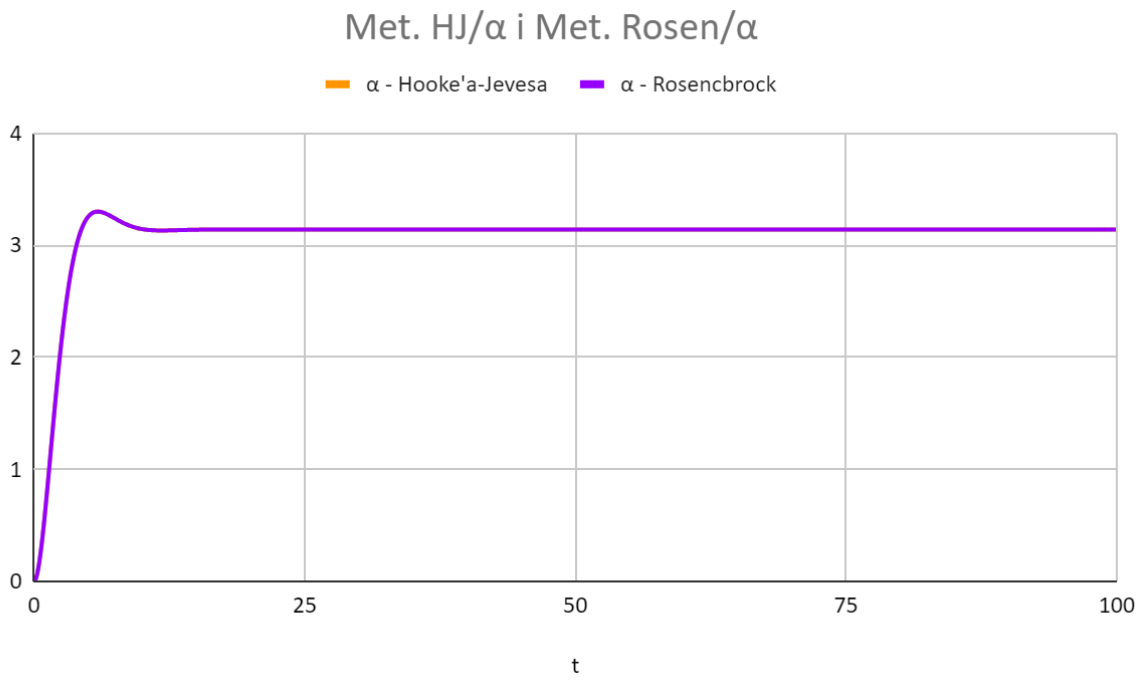


Metoda Hooke'a-Jeevesa stabilizuje się bardzo szybko, natomiast metoda Rosenbrocka dopiero po około 50 iteracjach. Druga z metod wykazuje bardzo dynamiczne zmiany w początkowej fazie, szczególnie w przedziale od 5 do 50 iteracji. Ostateczne wyniki dla obu metod są bardzo podobne i bliskie zeru, co sugeruje, że obie metody prowadzą do podobnego rozwiązania.
Podsumowując, dla zmiennej metoda Hooke'a-Jeevesa okazuje się bardziej stabilna i szybciej zbiega do wartości końcowej, podczas gdy metoda Rosenbrocka charakteryzuje się początkowymi oscylacjami, ale ostatecznie osiąga podobny wynik.

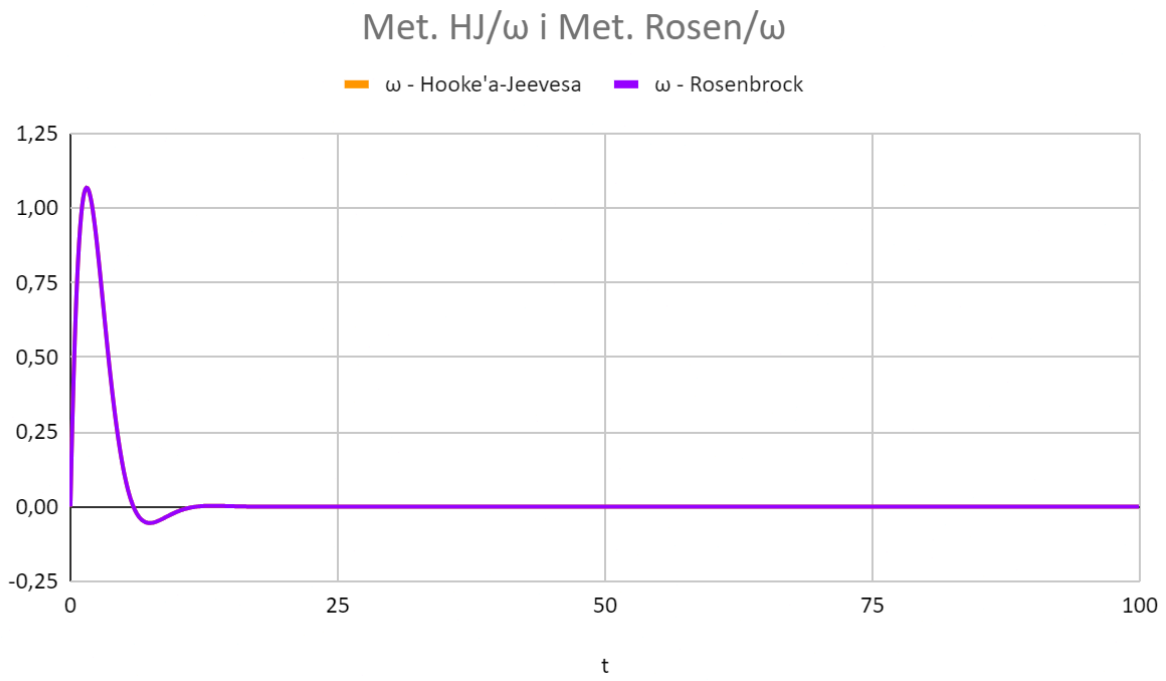
Długość kroku	Metoda Hooke'a-Jeevesa				Metoda Rosenbrocka			
	k1*	k2*	Q*	Liczba wywołań funkcji celu	k1*	k2*	Q*	Liczba wywołań funkcji celu
0.01	2,87312	4,488155	193,938	418	2,87317	4,88167	193,938	247

Tabela porównuje wyniki dwóch metod optymalizacyjnych dla długości kroku 0,01. Porównywane są końcowe wartości zmiennych k1*, k2*, wynik funkcji celu oraz liczba wywołań funkcji celu.
Obie metody uzyskują bardzo podobne wartości dla zmiennych k1* i k2*. Różnica w wynikach pojawia się dopiero na 5 miejscu po przecinku.
Wynik funkcji celu Q* jest identyczny dla obu metod i wynosi 193,938. Oznacza to, że obie metody osiągnęły to samo minimum funkcji celu, co wskazuje na ich porównywalną skuteczność.

Istotna różnica pojawia się w liczbie wywołań funkcji celu. Metoda Hooke’a-Jeevesa potrzebuje 418 wywołań funkcji, natomiast metoda Rosenbrocka tylko 247. Wskazuje to, że druga z nich jest bardziej efektywna pod względem liczby wywołań funkcji celu, co oznacza, że szybciej osiąga optymalne rozwiązanie.



Obie metody, zarówno Hooke’a-Jeevesa jak i Rosenbrocka szybko zbliżają się do wartości docelowej parametru. Początkowo wzrost wartości α jest gwałtowny, co wskazuje na szybki początkowy postęp optymalizacji. Po około 5-10 iteracjach, obie metody osiągają stabilizację.
Ostateczna wartość dla obu metod jest niemal identyczna, co sugeruje, że metody te są równie efektywne.



Metoda metody szybko zbiegają się do wartości bliskiej zeru i stabilizuje się po kilku iteracjach. Obie metody kończą z wartością ω równą zero, co sugeruje, że obie zbiegają do tej samej optymalnej wartości. Oznacza to, że obie metody są równie efektywne, w kontekście tego zadania.

Wnioski podsumowujące:

Metoda Hooke'a-Jeevesa i metoda Rosenbrocka osiągnęły zbliżone wyniki końcowe, co wskazuje, że są podobnie skuteczne.
Jednak metoda Rosenbrocka wymaga mniejszej liczby wywołań funkcji celu niezależnie od długości kroku. Jest to istotna różnica, ponieważ mniejsza liczba wywołań funkcji oznacza większą efektywność obliczeniową.
W przypadku dłuższych kroków zauważono wzrost liczby wywołań funkcji celu, co sugeruje, że dłuższe kroki mogą nie być optymalne dla dokładności i efektywności poszukiwania minimum. Jednak należy pamiętać, że długość kroku mogą różnić się w zależności od problemu i warunków początkowych.
Wyniki otrzymane przez obie metody różniły się dopiero na dalszych miejscach po przecinku, co wskazuje na porównywalną precyzję obu metod w znajdowaniu rozwiązań.