Imię i nazwisko		Kierunek
Jakub Sputo, Jakub Stanula-Kaczka, Kacper Strzesak		Informatyka techniczna
Kurs	Grupa	Data zajęć
Optymalizacja	3	27.10, 04.11.2025 r.
Numer ćwiczenia	Temat ćwiczenia	
2	Optymalizacja funkcji wielu zmiennych metodami bezgradientowymi	

Cel ćwiczenia:

Celem ćwiczenia było zapoznanie się z metodami optymalizacji bezgradientowej funkcji wielu zmiennych poprzez implementację oraz praktyczne zastosowanie dwóch algorytmów: metody Hooke'a-Jeevesa oraz metody Rosenbrocka.

Opis zadania:

Zadanie zostało podzielone na dwie części. Pierwsza część polegała na optymalizacji testowej funkcji wyrażonej wzorem:

$$f(x_1, x_2) = x_1^2 + x_2^2 - \cos(2.5\pi x_1) - \cos(2.5\pi x_2) + 2$$

Funkcja ta charakteryzuje się obecnością wielu minimów lokalnych oraz jednego minimum globalnego w punkcie (0, 0) o wartości f(0, 0) = 0.

Przeprowadzono 100 powtórzeń optymalizacji dla trzech różnych długości kroku (0.01, 0.36, 0.876) rozpoczynając z losowych punktów startowych należących do przedziału: $x_1^{(0)} \in [-1, 1], x_2^{(0)} \in [-1, 1]$.

Na wylosowanych punktach startowych zastosowano dwie metody optymalizacji: metodę Hooke'a-Jeevesa, oraz metodę Rosenbrocka.

Druga część ćwiczenia dotyczyła rozwiązania problemu rzeczywistego związanego z ruchem ramienia robota o zadanej długości $l = 2m$ oraz masie $m_r = 1kg$, którego zadaniem jest wykonanie obrotu o kąt $\pi\ rad$ oraz zatrzymanie się, w celu umieszczenia ciężarka o masie $m_c = 5kg$ na platformie *P*. Ruch ramienia opisany był równaniem różniczkowym drugiego rzędu uwzględniającym moment bezwładności oraz współczynnik tarcia.

Zadanie optymalizacyjne polegało na znalezieniu współczynników wzmocnienia k_1 i k_2 minimalizujących funkcjonał jakości:

$$Q(k_1, k_2) = \int_0^{t_{end}} \left(10\left(\alpha_{ref} - \alpha(t)\right)^2 + \left(\omega_{ref} - \omega(t)\right)^2 + \left(M(t)\right)^2\right) dt$$

,

gdzie $\alpha_{ref} = \pi\, rad, \omega_{ref} = 0\ {}^{rad}/_{s}$. Początkowe wartości współczynników należały do przedziału $k_1^{(0)} \in [0{,}20]\ Nm, k_2^{(0)} \in [0{,}20]\ Nms$.

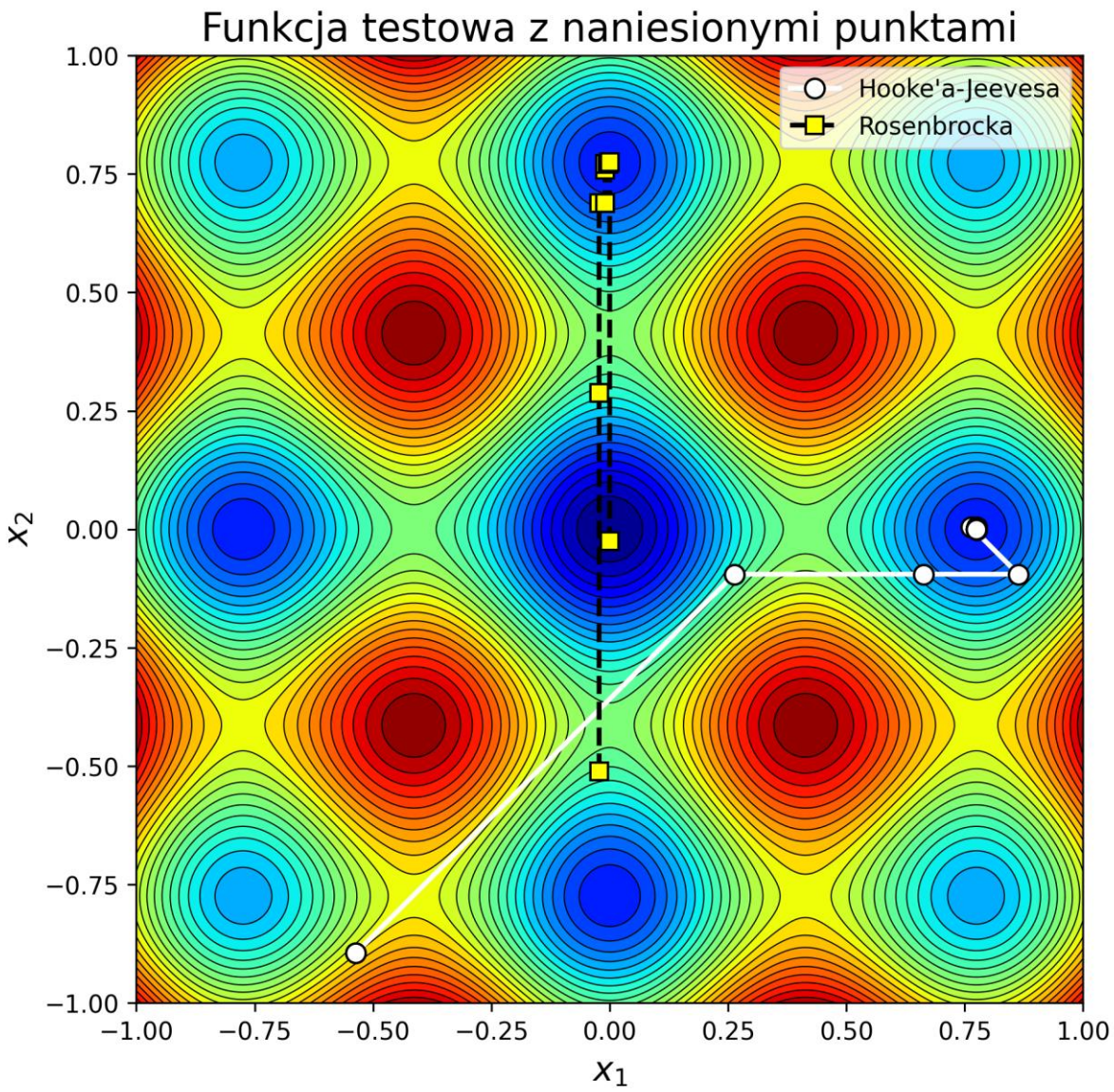
Dyskusja wyników:

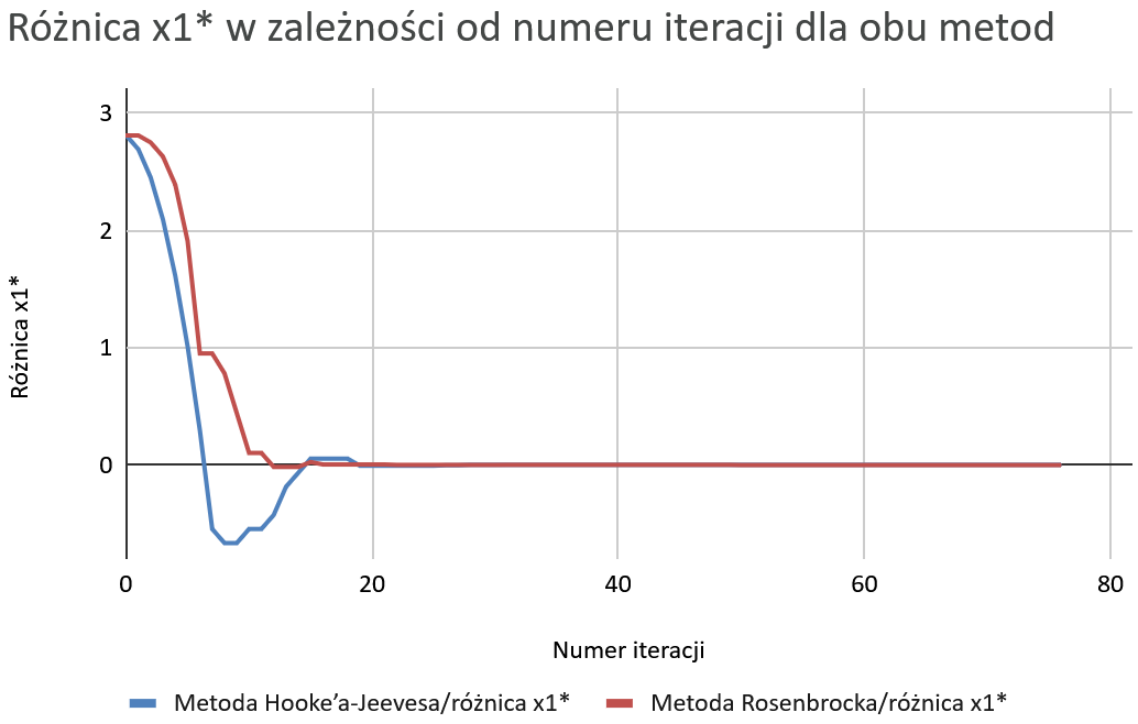
- Testowa funkcja celu

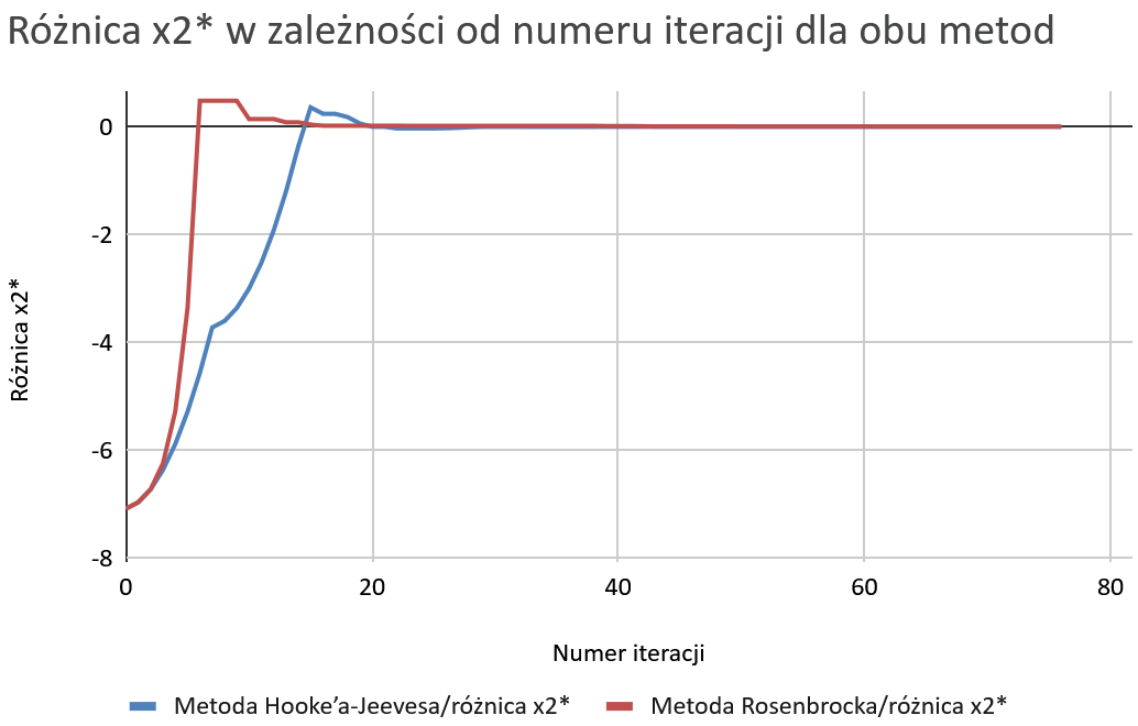
Istotnym obserwowanym zjawiskiem jest znacząca przewaga metody Rosenbrocka pod względem liczby wywołań funkcji celu. Dla najmniejszego kroku s = 0,01 metoda Rosenbrocka wymaga średnio około 86 wywołań funkcji, podczas gdy metoda Hooke'a-Jeevesa potrzebuje ich około 226, co daje niemal trzykrotną różnicę. Ta przewaga utrzymuje się również dla większych kroków - dla s = 0,036 odpowiednio 123 i 242 wywołań, a dla s = 0,876 około 193 i 282 wywołań.

Pod względem znajdowania minimum globalnego widać duży wpływ długości kroku startowego. Przy najmniejszym kroku s = 0,01 obie metody znalazły minimum globalne tylko w 14% przypadków. Zwiększenie kroku do s = 0,036 poprawiło skuteczność do 29%. Dopiero przy największym kroku s = 0,876 pojawiła się wyraźna różnica. Metoda Hooke'a-Jeevesa osiągnęła 56% skuteczności, a metoda Rosenbrocka aż 93%.

Obie metody osiągnęły bardzo wysoką dokładność w lokalizacji minimum globalnego funkcji testowej. Znalezione współrzędne x1* i x2* mieszczą się w przedziale od 10^{-6} do 10^{-5}, przy czym wartości funkcji oscylują w zakresie od $5{,}9\times10^{-8}$ do $2{,}74\times10^{-7}$. Dla porównania, wartość teoretyczna minimum globalnego wynosi dokładnie 0 w punkcie (0, 0). Warto zauważyć, że założony próg dokładności $\varepsilon = 10^{-4}$ został przekroczony o 1-2 rzędy wielkości. Długość kroku startowego wpływa nieznacznie na końcową dokładność, przy czym największy krok daje najlepsze rezultaty.

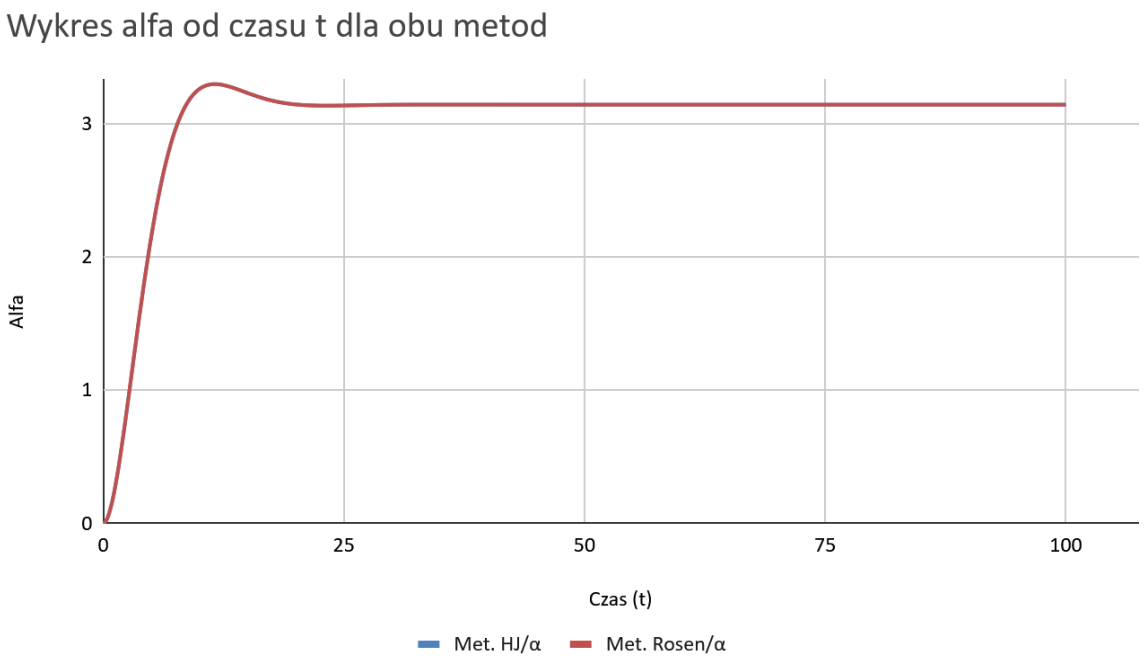

Funkcja testowa z naniesionymi punktami
Hooke'a-Jeevesa
Rosenbrocka
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x_1
x_2

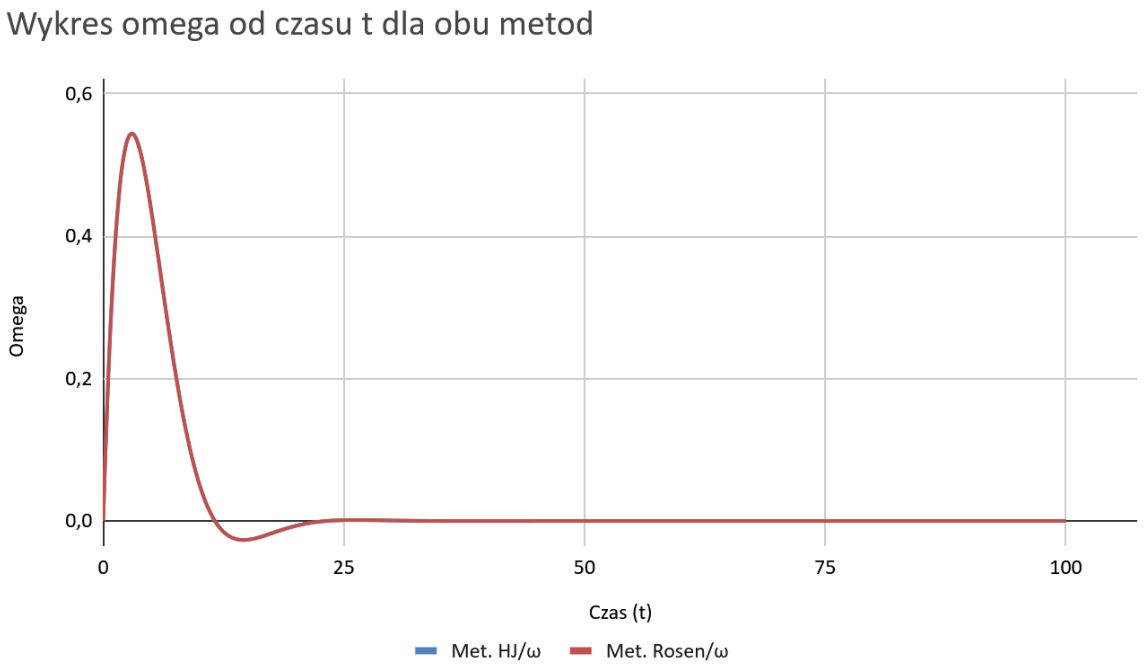




Jak można zauważyć z wykresów, Metoda Rosenbrocka szybciej stabilizuje się, natomiast dokładnie rozwiązanie szybciej znajduje metoda Hooke'a-Jeevesa. Metoda Hooke'a-Jeevesa w początkowych iteracjach charakteryzuje się odbieganiem od wartości końcowej. Obie prowadzą jednak do prawidłowego rozwiązania.

- Problem rzeczywisty





W części dotyczącej problemu rzeczywistego analizowano ruch ramienia robota o długości l = 2 m i masie m_r = 1 kg, którego zadaniem było wykonanie obrotu o kąt π rad oraz zatrzymanie się w założonym pozycji o założonym kącie. Optymalizacji podlegały współczynniki wzmocnienia k_1 i k_2, minimalizujące funkcjonał jakości opisujący błędy położenia, prędkości oraz energię sterowania.

Na podstawie uzyskanych wyników można zauważyć, że obie zastosowane metody – Hooke'a-Jeevesa i Rosenbrocka – prowadzą do zbliżonych wartości współczynników oraz porównywalnych przebiegów

czasowych zmiennych stanu. Ramię robota wykonuje ruch z niewielkim przeregulowaniem, po czym jego położenie zostaje skorygowane ruchem w przeciwnym kierunku, co skutkuje szybkim ustabilizowaniem się układu. Prędkość kątowa ω chwilowo przyjmuje wartości ujemne, co jest zgodne z oczekiwanym zachowaniem układu z tłumieniem.

Pod względem efektywności obliczeniowej lepiej wypadła metoda Rosenbrocka, która wymaga mniejszej liczby wywołań funkcji celu przy zachowaniu tej samej dokładności optymalizacji. Ostatecznie obie metody pozwoliły na uzyskanie stabilnego i dokładnego sterowania, co potwierdza ich przydatność w zadaniach optymalizacji rzeczywistych układów dynamicznych.

Wnioski:

Analiza wyników dla testowej funkcji celu pokazuje, że długość kroku startowego znacząco wpływa na skuteczność metod. Im większy krok, tym łatwiej metodom znaleźć minimum globalne. Najlepsze wyniki daje największy krok, zwłaszcza dla metody Rosenbrocka. Obie metody bardzo dokładnie wyznaczają minimum, znacznie lepiej niż wymagał założony próg, a najlepsza dokładność także pojawia się przy największym kroku.

Symulacje wykazały, że obie metody bezgradientowe prowadzą do wyznaczenia współczynników k, dla których przebiegi czasowe kąta α i prędkości kątowej ω są prawie identyczne. Oznacza to, że obie metody dają porównywalne, skuteczne rozwiązania. Ramię robota charakteryzuje się wykonaniem ruchu ponad zadany kąt, co następnie jest korygowane (prędkość kątowa schodzi poniżej zera, ruch wykonywany jest w odwrotną stronę). Metoda Rosenbrocka wygląda na metodę efektywniejszą (mniejsza liczba wywołań funkcji celu), przy osiągnięciu porównywalnych wyników.

Opis algorytmów:

Metoda Hook'a-Jeevesa - metoda poszukiwania minimum, oparta na dwóch etapach: eksploracji i ruchu wzdłuż wzorca. W fazie eksploracyjnej bada się otoczenie punktu bazowego wzdłuż osi współrzędnych. Jeśli znajdzie się lepsze rozwiązanie, wykonuje się ruch wzdłuż wzorca w tym kierunku. W przypadku braku poprawy krok jest zmniejszany aż do osiągnięcia zadanej dokładności ε.

Parametry:

- s - długość kroku (testowa funkcja celu: s = {0.01, 0.36, 0.876}; problem rzeczywisty: s = 0.1)
- alfa - współczynnik zmniejszania długości kroku (testowa funkcja celu: alfa = 0.5; problem rzeczywisty: alfa = 0.8)
- epsilon - dokładność (1e-4)
- N_{max} - maksymalna liczba wywołań funkcji celu (Nmax = 10 000)

Metoda Rosenbrocka - metoda szukania minimum, która samodzielnie dopasowuje kierunki poszukiwań. Zaczyna od prostych wektorów jednostkowych, a potem zmienia je w zależności od tego, czy próby poprawy rozwiązania się powiodły. Jeśli krok w danym kierunku daje lepszy wynik, jest zwiększany (współczynnik α), a jeśli nie, kierunek się odwraca, a krok zmniejsza (współczynnik β). Po serii prób we wszystkich kierunkach układ wektorów jest poprawiany metodą Grama-Schmidta, żeby lepiej pasować do kształtu funkcji.

Parametry:

- s - długość kroku (testowa funkcja celu: s = {0.01, 0.36, 0.876}; problem rzeczywisty: s = 0.1)
- alfa - współczynnik zmniejszania długości kroku (testowa funkcja celu: alfa = 2.0; problem rzeczywisty: alfa = 0.8)
- beta - współczynnik kontrakcji (beta = 0.5)
- epsilon - dokładność (1e-4)
- N_{max} - maksymalna liczba wywołań funkcji celu (Nmax = 10 000)

Zaimplementowana metoda Hooke'a-Jeevesa

```
solution HJ(matrix(*ff)(matrix, matrix, matrix), matrix x0, double s, double alpha,
double epsilon, int Nmax, matrix ud1, matrix ud2)
{
  try
  {
    matrix x = x0;
    matrix X_b_prev = x;
    matrix X_b = x;

    solution Xopt;

    double f_x, f_xb;

    while(s >= epsilon) {
      X_b = x;
      x = sol2mat(HJ_trial(ff, X_b, s));
      Xopt.x = x; Xopt.fit_fun(ff); f_x = m2d(Xopt.y);
      Xopt.x = X_b; Xopt.fit_fun(ff); f_xb = m2d(Xopt.y);
      if (f_x < f_xb) {
        while (f_x < f_xb) {
          X_b_prev = X_b;
          X_b = x;
          x = 2 * X_b - X_b_prev;
          x = sol2mat(HJ_trial(ff, x, s));

          Xopt.x = x; Xopt.fit_fun(ff); f_x = m2d(Xopt.y);
          Xopt.x = X_b; Xopt.fit_fun(ff); f_xb = m2d(Xopt.y);

          if (solution::f_calls >= Nmax)
          {
            throw string("Przekroczono maksymalna liczbe wywolan funkcji celu");
          }
        }
        x = X_b;
      } else {
        s = alpha * s;
      }
      if (solution::f_calls >= Nmax)
      {
        throw string("Przekroczono maksymalna liczbe wywolan funkcji celu");
      }
```

```
    }
    Xopt.x = X_b;
    Xopt.fit_fun(ff);
    return Xopt;
  }
  catch (string ex_info)
  {
    throw ("solution HJ(...):\n" + ex_info);
  }
}
```

Funkcja pomocnicza dla metody Hooke'a-Jeevesa

```
solution HJ_trial(matrix(*ff)(matrix, matrix, matrix), matrix XB, double s, matrix
ud1, matrix ud2)
{
  try
  {
    matrix ej = ident_mat(2);

    solution Xopt;
    double f_x, f_forward, f_backward;

    for (int i = 0; i < 2; i++) {
      Xopt.x = XB;
      Xopt.fit_fun(ff);
      f_x = m2d(Xopt.y);

      Xopt.x = XB + s * ej[i];
      Xopt.fit_fun(ff);
       f_forward = m2d(Xopt.y);


      Xopt.x = XB - s * ej[i]; Xopt.fit_fun(ff); f_backward = m2d(Xopt.y);

      if (f_forward < f_x) {
        XB = XB + s*ej[i];
      } else if (f_backward < f_x) {
        XB = XB - s*ej[i];
      }
    }
    Xopt.x = XB(0);
    Xopt.y = XB(1);
    return Xopt;
  }
  catch (string ex_info)
  {
    throw ("solution HJ_trial(...):\n" + ex_info);
  }
}
```

Zaimplementowana metoda Rosenbrocka

```
solution Rosen(matrix (*ff)(matrix, matrix, matrix), matrix x0, matrix s0, double
alpha, double beta, double epsilon, int Nmax, matrix ud1, matrix ud2)
{
  try
  {
    solution Xopt;

    int n = get_len(x0);
    int i = 0;

    matrix D(n, n);
    for (int j = 0; j < n; j++)
    {
      for (int k = 0; k < n; k++)
      {
        D(k, j) = (j == k) ? 1.0 : 0.0;
      }
    }

    matrix s = s0;
    matrix lambda(n, 1);
    matrix p(n, 1);

    for (int j = 0; j < n; j++)
    {
      lambda(j) = 0.0;
      p(j) = 0.0;
    }

    solution xB(x0);
    xB.fit_fun(ff, ud1, ud2);

    while (true)
    {
      for (int j = 0; j < n; j++)
      {
        matrix d_j(n, 1);
        for (int k = 0; k < n; k++)
        {
          d_j(k) = D(k, j);
        }

        matrix x_new = xB.x + s(j) * d_j;
        solution xNew(x_new);
        xNew.fit_fun(ff, ud1, ud2);
```

```
      if (xNew.y(0) < xB.y(0))
      {
        xB.x = xNew.x;
        xB.y = xNew.y;
        lambda(j) = lambda(j) + s(j);
        s(j) = alpha * s(j);
      }
      else
      {
        s(j) = -beta * s(j);
        p(j) = p(j) + 1;
      }
    }

    i++;

    bool all_lambda_nonzero = true;
    bool all_p_nonzero = true;

    for (int j = 0; j < n; j++)
    {
      if (lambda(j) == 0.0)
        all_lambda_nonzero = false;
      if (p(j) == 0.0)
        all_p_nonzero = false;
    }

    if (all_lambda_nonzero && all_p_nonzero)
    {
      matrix Q(n, n);
      for (int row = 0; row < n; row++)
      {
        for (int col = 0; col < n; col++)
        {
          if (row >= col)
          {
            Q(row, col) = lambda(col);
          }
          else
          {
            Q(row, col) = 0.0;
          }
        }
      }

      matrix Q_star = D * Q;

      for (int j = 0; j < n; j++)
      {
        matrix v_j(n, 1);
```

```
        for (int k = 0; k < n; k++)
        {
          v_j(k) = Q_star(k, j);
        }

        for (int k = 0; k < j; k++)
        {
          matrix d_k(n, 1);
          for (int l = 0; l < n; l++)
          {
            d_k(l) = D(l, k);
          }

          double dot_product = 0.0;
          for (int l = 0; l < n; l++)
          {
            dot_product += v_j(l) * d_k(l);
          }

          for (int l = 0; l < n; l++)
          {
            v_j(l) -= dot_product * d_k(l);
          }
        }

        double norm_val = 0.0;
        for (int k = 0; k < n; k++)
        {
          norm_val += v_j(k) * v_j(k);
        }
        norm_val = sqrt(norm_val);

        if (norm_val > 1e-10)
        {
          for (int k = 0; k < n; k++)
          {
            D(k, j) = v_j(k) / norm_val;
          }
        }
      }

      for (int j = 0; j < n; j++)
      {
        lambda(j) = 0.0;
        p(j) = 0.0;
        s(j) = s0(j);
      }
    }

    if (solution::f_calls > Nmax)
    {
```

```
        Xopt = xB;
        Xopt.flag = 0;
        return Xopt;
      }

      double max_s = 0.0;
      for (int j = 0; j < n; j++)
      {
        double abs_s = fabs(s(j));
        if (abs_s > max_s)
          max_s = abs_s;
      }

      if (max_s < epsilon)
      {
        break;
      }
    }

    Xopt = xB;
    Xopt.flag = 1;

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

Rzeczywista funkcja celu

```
matrix ff2R(matrix x, matrix ud1, matrix ud2)
{
  matrix y;

  double k1 = m2d(x(0));
  double k2 = m2d(x(1));

  double alpha_ref = M_PI;
  double omega_ref = 0.0;

  matrix ref(2, 1);
  ref(0) = alpha_ref;
  ref(1) = omega_ref;

  matrix Y0(2, 1);
  Y0(0) = 0.0;
  Y0(1) = 0.0;
```

```
  matrix* Y = solve_ode(df2, 0.0, 0.1, 100.0, Y0, x, ref);
  int n = get_len(Y[0]);
  double Q = 0.0;
  double dt = 0.1;

  for (int i = 0; i < n; i++)
  {
    double alpha = Y[1](i, 0);
    double omega = Y[1](i, 1);

    double M = k1 * (alpha_ref - alpha) + k2 * (omega_ref - omega);

    Q += (10.0 * pow(alpha_ref - alpha, 2) + pow(omega_ref - omega, 2) + pow(M, 2))
* dt;
  }
  y = Q;

  return y;
}
```

Funkcja opisująca równania ruchu ramienia z ciężarkiem

```
matrix df2(double t, matrix Y, matrix ud1, matrix ud2)
{
  matrix dY(2, 1);
  double m = 1.0; // masa ramienia
  double mc = 5.0; // masa ciężarka
  double l = 2.0; // długość ramienia
  double b = 0.25; // współczynnik tarcia

  double I = m * l * l / 3.0 + mc * l * l;
  double k1 = m2d(ud1(0));
  double k2 = m2d(ud1(1));
  double alpha_ref = m2d(ud2(0));
  double omega_ref = m2d(ud2(1));

  double M = k1 * (alpha_ref - Y(0)) + k2 * (omega_ref - Y(1));

  dY(0) = Y(1);
  dY(1) = (M - b * Y(1)) / I;
  return dY;
}
```

Funkcja lab2

```
void lab2()
{
  double alphaHJ = 0.5;
  double alphaRos = 2.0;
  double beta = 0.5;
  double epsilon = 1e-4;
  int Nmax = 10000;

  int n = 100;

  solution optHJ, optRos;

  ofstream Sout("tabela3_lab2.csv");

  Sout << "Lp.,x1,x2,x1*,x2*,y*,f_calls,,x1*,x2*,y*,f_calls" << endl;

  double s = 0.1;
  matrix k0 = rand_mat(2);

  for (int j = 0; j < 2; ++j)
  {
    k0(j) = 20.0 * k0(j);
  }

  matrix s0(2, 1);
  s0(0) = s;
  s0(1) = s;

  // METODA HOOKE'A JEEVESA
  cout << "HJ method:\n";
  solution::clear_calls();
  optHJ = HJ(ff2R, k0, s, alphaHJ, epsilon, Nmax);

 cout << optHJ.x(0) << "," << optHJ.x(1) << "," << optHJ.y(0) << "," << optHJ.f_calls;

  // METODA ROSENBROCKA
  cout << "Rosenbrock method:\n";
  solution::clear_calls();
  optRos = Rosen(ff2R, k0, s0, alphaRos, beta, epsilon, Nmax);
  cout << "," << optRos.x(0) << "," << optRos.x(1) << "," << optRos.y(0) << "," <<
optRos.f_calls << endl;

  Sout.close();
}
```