Optymalizacja IT gr. 2

Dariusz Homa

Wojciech Jurgielewicz

Michał Koleżyński

# Projekt 6

## Cel ćwiczenia

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

# Wykonanie

## Kody funkcji

### *Algorytm ewolucyjny*

```
solution EA(matrix(*ff)(matrix, matrix, matrix), int N, matrix lb, matrix ub, int
mi, int lambda, matrix sigma0, double epsilon, int Nmax, matrix ud1, matrix ud2)
{
    try
    {
        solution* P = new solution[mi + lambda];
        solution* Pm = new solution[mi];

        matrix IFF(mi, 1), temp(N, 2);

        double r, s, s_IFF;
        double tau = 1.0 / sqrt(2 * N), tau1 = 1.0 / sqrt(2 * sqrt(N));

        int j_min;

        for (int i = 0; i < mi; ++i)
        {
            P[i].x = matrix(N, 2);

            P[i].x(0, 0) = (lb(1) - lb(0)) * m2d(rand_mat()) + lb(0);
            P[i].x(0, 1) = sigma0(0);

            P[i].x(1, 0) = (ub(1) - ub(0)) * m2d(rand_mat()) + ub(0);
            P[i].x(1, 1) = sigma0(0);

            P[i].fit_fun(ff, ud1, ud2);

            if (P[i].y < epsilon)
            {
                P[i].flag = 1;
                return P[i];
            }
        }
        while (true)
        {
            s_IFF = 0;

            for (int i = 0; i < mi; ++i)
            {
                IFF(i) = 1 / P[i].y(0);
                s_IFF += IFF(i);
            }

            for (int i = 0; i < lambda; ++i)
            {
                r = s_IFF * m2d(rand_mat()); ;
                s = 0;
                for (int j = 0; j < mi; ++j)
                {
```

```
                s += IFF(j);
                    if (r <= s)
                    {
                        P[mi + i] = P[j];
                        break;
                    }
                }
            }

            for (int i = 0; i < lambda; ++i)
            {
                r = m2d(randn_mat());
                for (int j = 0; j < N; ++j)
                {
                    P[mi + i].x(j, 1) *= exp(tau1 * r + tau *
m2d(randn_mat()));
                    P[mi + i].x(j, 0) += P[mi + i].x(j, 1) * m2d(randn_mat());
                }
            }

            for (int i = 0; i < lambda; i += 2)
            {
                r = m2d(rand_mat());
                temp = P[mi + i].x;
                P[mi + i].x = r * P[mi + i].x + (1 - r) * P[mi + i + 1].x;
                P[mi + i + 1].x = r * P[mi + i + 1].x + (1 - r) * temp;
            }

            for (int i = 0; i < lambda; ++i)
            {
                P[mi + i].fit_fun(ff, ud1, ud2);
                if (P[mi + i].y < epsilon)
                {

                    P[mi + i].flag = 1;
                    return P[mi + i];
                }
            }

            for (int i = 0; i < mi; ++i)
            {
                j_min = 0;
                for (int j = 1; j < mi + lambda; ++j)
                    if (P[j_min].y > P[j].y)
                        j_min = j;
                Pm[i] = P[j_min];
                P[j_min].y = 1e10;
            }
```

```
            for (int i = 0; i < mi; ++i)
                P[i] = Pm[i];

            if (solution::f_calls > Nmax)
                break;
        }

        P[0].flag = 0;

        return P[0];
    }
    catch (string ex_info)
    {
        throw ("solution EA(...):\n" + ex_info);
    }
}
```

## Testowa funkcja celu

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

*Ograniczenia:*

- $x_1^0 \in [-5,5]$
- $x_2^0 \in [-5,5]$

*Cel*

- Wykonanie 100 optymalizacji dla pięciu różnych wartości początkowych współczynnika mutacji (σ=0.01, 0.1, 1, 10, 100). Porównanie skuteczności oraz złożoności czasowej przeprowadzonych dla różnych współczynników.

*Parametry*

- Liczebność populacji bazowej $\mu = 20$
- Liczebność populacji tymczasowej $\lambda = 40$
- Przedział poszukiwań $< lb_1, ub_1 > : < 5,5 >$
- Przedział poszukiwań $< lb_2, ub_2 > : < 5,5 >$
- Dokładność $\epsilon = 10^{-5}$
- Maksymalna liczba wywołań funkcji $N_{max} = 10000$

*Kod*

Funkcja celu

```
matrix lab6_fun(matrix x, matrix ud1, matrix ud2)
{
    return pow(x(0), 2) + pow(x(1), 2) - cos(2.5 * M_PI * x(0)) - cos(2.5 *
M_PI * x(1)) + 2;
}
```

100 optymalizacji

```
string msg;
    for (auto sigma : sigma_tab)
    {
        for (int i = 0; i < 100; i++)
        {
            solution result = EA(lab6_fun, N, lb, ub, mi, lambda, sigma,
epsilon, Nmax);

            msg = solution::f_calls > Nmax ? "nie" : "tak";

            SAVE_TO_FILE("Test-" + to_string(sigma) + ".txt") << result.x(0)
<< ";" << result.x(1) << ";" << result.y(0) << ";" << solution::f_calls << ";"
<< msg << "\n";

            solution::clear_calls();
        }
    }
```

*Omówienie wyników*

- Dla współczynników zakresu mutacji $\sigma \in \{0.1, 1, 10\}$ otrzymujemy wysoką skuteczność znajdowania minimów globalnych (niemalże 100%) oraz dość dobrą w porównaniu z wynikami dla innej wartości $\sigma$, liczbą wywołań funkcji.
- Dla współczynnika małego $\sigma = 0.01$ otrzymujemy wysoką skuteczność znajdowania minimów, lecz kosztem zwiększenia liczby wywołań funkcji celu.
- Dla współczynnika dużego $\sigma = 100$ otrzymujemy spadek skuteczności, oraz ogromny, prawie dwukrotny wzrost liczby wywołań funkcji celu.

## Problem rzeczywisty

*Cel*

Znalezienie takich wartości $b_1$ i $b_2$ dla których rozkład położenia ciężarków w czasie jest najbardziej optymalny

*Parametry*

- Przedział poszukiwań $< lb_1 ub_1 >: \{0.1, 3\}$
- Przedział poszukiwań $< lb_2 ub_2 >: \{0.1, 3\}$
- Współczynniki zakresu przedziału $\sigma_1 = 1, \sigma_2 = 1$
- Masy $m_1 = 5kg, m_2 = 5kg$
- Współczynniki sprężystości $k_1 = 1\frac{N}{m}, k_2 = 1\frac{N}{m}$
- Dokładność $\epsilon = 10^{-2}$

*Ograniczenia*

- $b_1 = [0.1, 3]\frac{Ns}{m}$
- $b_2 = [0.1, 3]\frac{Ns}{m}$

*Kod*

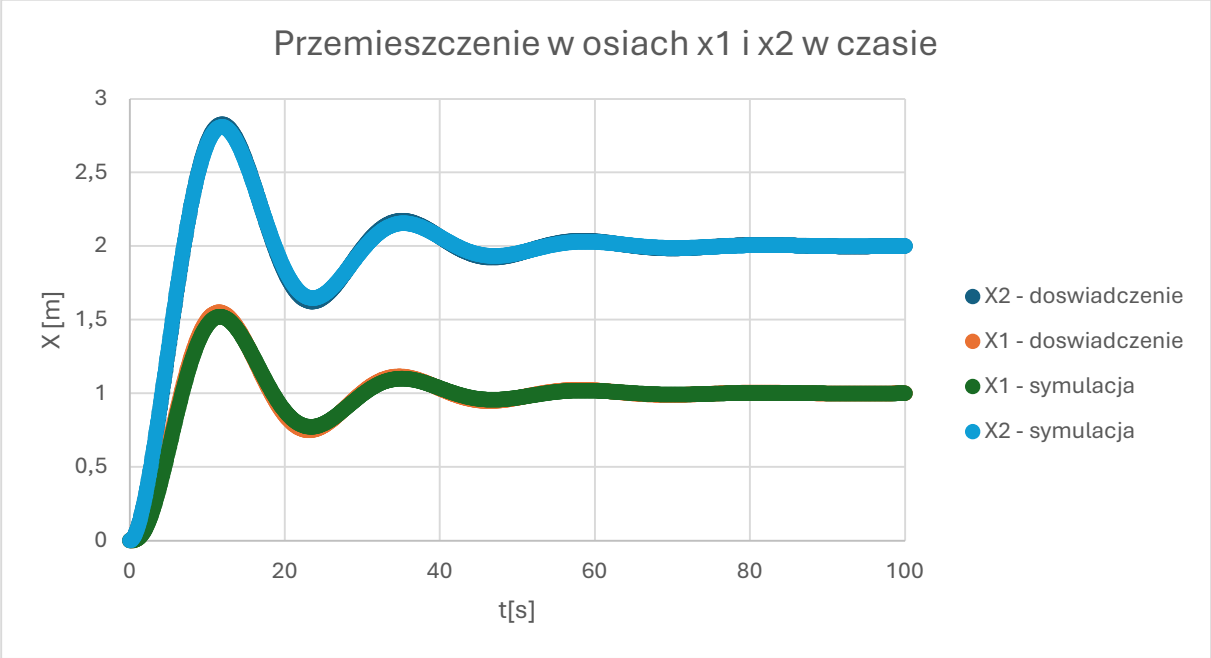
- Funkcja różniczkowa

```
matrix df6(double t, matrix Y, matrix ud1, matrix ud2)
{
    double m1 = 5;
    double m2 = 5;
    double k1 = 1;
    double k2 = 1;
    double F = 1.0;

    double b1 = ud2(0);
    double b2 = ud2(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;

}
```

- Funkcja dla problemu rzeczywistego

```
matrix fR6(matrix x, matrix ud1, matrix ud2)
{
    matrix y;
    matrix Y0(4, 1);
    matrix* Y = solve_ode(df6, 0, 0.1, 100, Y0, ud1, x[0]);
    for (int i = 0; i < ud1(0); i++)
    {
        y = y + abs(ud2(i, 0) - Y[1](i, 0)) + abs(ud2(i, 1) - Y[1](i, 2));
    }
    y(0) = y(0) / (2 * ud1(0));


    return y;
}
```

*Omówienie wyników*




Optymalizacje wykazały, że doświadczenie zostało przeprowadzone dla wartości zbliżonych do b1 = 1,2862, b2 = 2,998.

## Wnioski

- Aby uzyskać balans pomiędzy czasem obliczeń, a skutecznością znajdowania minimów globalnych przez algorytm ewoluujący, należy dobrać poprawnie współczynnik zakresu mutacji $\sigma$, ani nie za duży, ani nie za mały, najlepiej w przedziale $\sigma \in < 0.1, 10 >$
- Optymalizacja problemu rzeczywistego wykazała zbliżone współczynniki b1 i b2 do tych, które musiały być wykorzystane w doświadczeniu. Ponieważ algorytm ewolucyjny jest niedeterministyczny, otrzymane wyniki między dwoma podejściami do symulacji będą się nieznacznie różnić - jednak będą one zawsze w podanym zakresie tolerancji.