Estymacja parametrów modelu - układ nieliniowych równań różniczkowych zwyczajnych drugiego rzędu

W ramach kierunku Informatyka i Systemy Informacyjne

Na Wydziale Matematyki i Nauk Informacyjnych
Politechniki Warszawskiej

W ramach przedmiotu: Modelowanie Matematyczne

Nadzorowane przez: dr inż. Jakub Wagner

Autor: Kamila Wachulec

Baszkówka, 28 grudnia 2024r.

Spis treści

1 Lista symboli i oznaczeń

1.1 Akronimy

- **UNRRZ** – Układ Nieliniowych Równań Różniczkowych Zwyczajnych

1.2 Funkcje

- **proj2** – Program 1 do kompilacji rozwiązania
- **Pstrona** – Program 2 do kompilacji rozwiązania: pomocnicza funkcja wyznaczająca prawą stronę rozważanego układu równań różniczkowych
- **pom** – Funkcja wyznaczająca przybliżoną wartość lewej strony układu równań różniczkowych
- **odefun** – Funkcja pomocnicza dla **ode45**
- **crit** – Funkcja wyznaczająca sumę kwadratów błędów wszystkich parametrów
- **rcount** – Funkcja wyznaczająca $r_{jk}(t) \equiv \sqrt{[x_k(t) - x_j(t)]^2 + [y_k(t) - y_j(t)]^2}^3$
- **readtable** – Wbudowana procedura zapisująca dane z pliku do tabeli
- **linspace** – Wbudowana funkcja generująca równomiernie rozmieszczone punkty między a i b
- **zeros** – Wbudowana funkcja zapełniająca macierz o wybranym rozmiarze zerami
- **height** – Wbudowana procedura wyznaczająca wysokość tablicy
- **length** – Wbudowana procedura wyznaczająca długość największego wymiaru tablicy
- **odeset** – Wbudowana funkcja tworząca strukturę opcji, którą można podać jako argument do **ode45**
- **optimset** – Wbudowana procedura tworząca lub modyfikująca strukturę opcji optymalizacji
- **ode45** – Wbudowana funkcja numerycznego rozwiązywania równań różniczkowych
- **sum** – Wbudowana funkcja zliczająca sumę elementów wektora

- **plot** – Wbudowana procedura tworząca wykresy rozwiązań
- **fminsearch** – Wbudowana funkcja poszukująca minimum lokalnego nieograniczonej funkcji wielowymiarowej

1.3 Symbole

- *data* – tabela danych z pliku "data_57.csv"
- *time* – tabela danych z pliku "query_57.csv"
- $x_1, y_1, x_2, y_2, x_3, y_3$ – współrzędne obiektów
- m – wektor rozważanych mas w pierwszej części rozwiązania
- *masa* – początkowe przybliżenie masy
- G – stała grawitacyjna
- t – punkt w czasie
- x – wektor przechowujący pierwsze współrzędne obiektów
- y – wektor przechowujący drugie współrzędne obiektów
- p – parametry początkowe do optymalizacji
- J – najmniejszy błąd uzyskany w pierwszej części zadania

2 Wprowadzenie

2.1 Tematyka Zadania

Zadanie "Estymacja parametrów modelu - układ nieliniowych równań różniczkowych zwyczajnych drugiego rzędu" ma na celu zaimplementowanie metody pozwalającej wyestymować rozwiązania UNRRZ, dla róźych punktów w czasie t zawartych w pliku "query_57.csv". Można korzystać z podanych parametrów i wartości początkowych znajdujących się w pliku "data_57.csv". Należy korzystać z wiedzy wyciągniętej z zajęć, własnoręcznie napisanych funkcji i przekształceń oraz procedur dostępnych w aplikacji MATLAB. Wektory estymat współrzędnych $x_1, y_1, x_2, y_2, x_3, y_3$ należy następnie przedstawić na wykresie i upewnić się, że rozwiązania spełniają określoną w zadaniu dokładność. Do tego należy wykorzystać dostępny plik z funkcją: "test_solution_57.p", który przyjmuje na wejściu wektory $x_1, y_1, x_2, y_2, x_3, y_3$ i wyznacza dokładność rozwiązania. Za w pełni zadowalające można uznać rozwiązanie charakteryzujące się wartością $\Delta \leqslant 4 \cdot 10^{-3}$.

Metoda zostanie opracowana dla UNRRZ:

$$\begin{cases} \dfrac{d^2x_1(t)}{dt^2} = Gm_2\dfrac{x_2(t)-x_1(t)}{r_{12}^3(t)} + Gm_3\dfrac{x_3(t)-x_1(t)}{r_{31}^3(t)} \\ \dfrac{d^2y_1(t)}{dt^2} = Gm_2\dfrac{y_2(t)-y_1(t)}{r_{12}^3(t)} + Gm_3\dfrac{y_3(t)-y_1(t)}{r_{31}^3(t)} \\ \dfrac{d^2x_2(t)}{dt^2} = Gm_3\dfrac{x_3(t)-x_2(t)}{r_{23}^3(t)} + Gm_1\dfrac{x_1(t)-x_2(t)}{r_{12}^3(t)} \\ \dfrac{d^2y_2(t)}{dt^2} = Gm_3\dfrac{y_3(t)-y_2(t)}{r_{23}^3(t)} + Gm_1\dfrac{y_1(t)-y_2(t)}{r_{12}^3(t)} \\ \dfrac{d^2x_3(t)}{dt^2} = Gm_1\dfrac{x_1(t)-x_3(t)}{r_{31}^3(t)} + Gm_2\dfrac{x_2(t)-x_3(t)}{r_{23}^3(t)} \\ \dfrac{d^2y_3(t)}{dt^2} = Gm_1\dfrac{y_1(t)-y_3(t)}{r_{31}^3(t)} + Gm_2\dfrac{y_2(t)-y_3(t)}{r_{23}^3(t)} \end{cases} \tag{1}$$

gdzie:

- t oznacza czas,
- $x_k(t)$ i $y_k(t)$ to współrzędne położenia k-tego obiektu dla $k = 1, 2, 3$,
- m_k to masa k-tego obiektu dla $k = 1, 2, 3$,
- G to stała grawitacyjna,
- $r_{jk}(t) \equiv \sqrt{[x_k(t) - x_j(t)]^2 + [y_k(t) - y_j(t)]^2} \quad \text{dla } j, k = 1, 2, 3.$

2.2 Oczekiwania

Po utworzeniu metody o zadawalającej dokładności oczekuje się uzyskania wykresu, który w estetyczny sposób przedstawi rozwiązanie. Metoda powinna być oszczędna pod względem wykonywanych operacji i pod względem czasowym. Powinna równie spełniać oczekiwania zadania.

2.3 Metodyka

Aby uzyskać oczekiwane rozwiązanie, kluczowym wyzwaniem będzie znalezienie odpowiedniej masy, która spełni Układ Równań 1, jednocześnie minimalizując czasochłonne wielokrotne wywoływanie wbudowanych funkcji w aplikacji MATLAB. Dlatego niezwykle istotne jest przeprowadzenie wstępnej analizy przedziału mas za pomocą alternatywnych metod, co pozwoli na wyznaczenie dobrych punktów startowych do późniejszej optymalizacji.

Po wstępnym oszacowaniu masy można skorzystać z funkcji **fminsearch** dostępnej w aplikacji MATLAB, która optymalizuje wartość funkcji dla zadanych parametrów. W tym przypadku funkcja celu powinna zwracać błąd między wartościami uzyskanymi za pomocą **ode45** - narzędzia do rozwiązywania równań różniczkowych - a rzeczywistymi wartościami z pliku "data_57.csv" biorąc pod uwagę ewentualny błąd pomiaru. Dzięki temu wyznaczenie najlepszego rozwiązania stanie się możliwe.

Dodatkowym wyzwaniem jest zapewnienie odpowiedniej dokładności rozwiązań oraz dostarczenie funkcjom optymalizacyjnym wystarczająco dobrych parametrów początkowych. Układ Równań 1 cechuje się bowiem dużą wrażliwością na warunki początkowe, co może utrudniać uzyskanie stabilnych wyników.

3 Przeprowadzenie algorytmu

Aby móc pracować na danych z załączonych plików, należy zapisać je za pomocą funkcji **readtable** do tabel. Zatem z pliku "data_57.csv" dane zapisuje się do tabeli **data**, a z pliku "query_57.csv" do tabeli **time**. Tabela **data** przechowuje tym samym 50 wierszy wartości odpowiednio z kolumn: $t, x_1, y_1, x_2, y_2, x_3, y_3$. Są to wyniki pomiaru położenia trzech obiektów o identycznych masach w pierwszych 50 punktach czasowych, przyciągających się grawitacyjnie. Tabela **time** zawiera natomiast 200 punktów w czasie, dla których należy wyznaczyć współrzędne położenia tych obiektów.

By zaoszczędzić na czasie wykonywania operacji, wyznacza się początkowe wartości, które później będą optymalizowane. Korzystając z tabeli data, można wyznaczyć przybliżoną masę **m** obiektów. Przyjmijmy przybliżoną stałą grawitacyjną $\mathbf{G} = 1$ oraz zgodnie z sugestią z zadania, niech $\mathbf{m} \in \left[\frac{1}{10\mathbf{G}}, \frac{10}{\mathbf{G}}\right]$. Niech **m** będzie zatem wektorem 100 równo rozłożonych punktów między $\frac{1}{10}$ a 10. Wtedy $\mathbf{m} = linspace(1/10, 10, 100)$.

Aby z danych z tabeli **data** wyznaczyć przybliżoną masę przydaje się przybliżenie drugich pochodnych występujących po lewej stronie Równania 1. Możemy przyjąć, że:

dla pewnego $x(t)$:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \lim_{h \to 0} \frac{x(t) - x(t-h)}{h} \tag{2}$$

co oznacza, że:

$$\left.\frac{\mathrm{d}x}{\mathrm{d}t}\right|_{t=t_n} \approx \frac{x(t_n) - x(t_{n-1})}{t_n - t_{n-1}} \tag{3}$$

Wiedząc, że można przyjąć stały krok $h = t_n - t_{n-1} = t_{n+1} - t_n$, można wyznaczyć przybliżenie drugiej pochodnej:

$$\left.\frac{\mathrm{d}^2x}{\mathrm{d}t^2}\right|_{t=t_n} \approx \frac{x(t_{n+1}) - x(t_n) - (x(t_n) - x(t_{n-1}))}{h^2} = \frac{x(t_{n+1}) - 2x(t_n) + x(t_{n-1})}{h^2} \tag{4}$$

Wzór ten sprawdza się dla każdej wartości t poza pierwszą i ostatnią. Zatem dla każdego punktu w czasie zawartego w tabeli **data** z tego przedziału tworzy się wektory **x** i **y**, które przechowują wartości x_i i y_i dla tego t. Następnie wylicza się przy pomocy Równania 4 drugie pochodne w tych punktach. Do tego tworzy się funkcję pomocniczą **pom**, którą przykładowo dla x_1 wywołuje się następująco:

pom(j, data.x1(j+1), data.x1(j), data.x1(j-1));

gdzie j jest indeksem rozważanego punktu w czasie w tabeli **data**, a *data.x1(j)* oznacza wartość w wierszu o numerze j i kolumnie o etykiecie *x1*.

Kolejnym etapem, jest przyrównanie wypadkowych wartości drugich pochodnych z ich odpowiednikiem - należy porównać dwie strony Układu Równań 1. By to zrobić, dla każdego z rozważanych punktów w czasie można przejść przez wszystkie rozważane masy z wektora **m** i sprawdzić, która najlepiej przybliża oczekiwaną wartość. Stąd dla każdej masy z **m** przy pomocy funkcji **Pstrona** wyznacza się prawą stronę Układu Równań 1. Wywołanie tej funkcji wygląda następująco:

$$\mathrm{P = Pstrona(m(i),\ x,\ y);}$$

gdzie **m(i)**, **x**, **y** to odpowiednio aktualnie rozważana masa i wektory zawierające położenia obiektów w danej chwili.

Po wyznaczeniu obu stron równnia, metodą najmniejszej sumy kwadratów można wyznaczyć masę, przy której występuje najmniejsza różnica między stronami Równania 1. Używa się tej metody, by ominąć błędy pomiarowe. Dla $\mathbf{m} \in [\frac{1}{10}, 10]$ początkowo wyznaczonna w ten sposób masa wynosi **0.6**. Jest to tym samym początkowa wartość masy dla dalszych przybliżeń. Niech powstanie zmienna **masa** = 0.6.

Kolejnym krokiem jest optymalizacja parametrów za pomocą funkcji **fminsearch**. Funkcja ta przyjmuje funkcję której minimalny wynik jest szukany, jej parametry początkowe i parametry opcjonalne dostępne dla funkcji **fminsearch**. W zamian zwraca parametry początkowe w postaci wektora **p** odpowiadające najlepszemu wynikowi funkcji **fminsearch** oraz wartość tego najlepszego wyniku w postaci wartości **J**.

Niech funkcją której minimalny wynik jest szukany będzie funkcja **crit**, która korzystając z **ode45** wyznacza wartości współrzędnych dla punktów w czasie z tabeli **data**. Następnie jak wcześniej, metodą najmniejszej sumy kwadratów wyznaczy współczynnik błędu danych. Współczynnik ten jest sumą kwadratów błędów pomiarów dla każdej rozważanej współrzędnej dla każdego rozważanego punktu w czasie. To minimalna wartość tego współczynnika jest poszukiwana przez **fminsearch**.

Niech parametrami początkowymi funkcji **crit** będą współrzędne, prędkości i masa obiektów. Niech współrzędne będą wartościami z drugiego punktu w czasie z tabeli **data**, niech przybliżone prędkości będą wyznaczone Równaniem 3 dla $t_n = t_2$ i parametr **masa**. Te trzynaście wartości początkowych będzie optymalizowane przez funkcję **crit**.

Istotne są tutaj parametry opcjonalne przyjmowane przez funkcje **ode45** i **fminsearch**. W tej metodzie przyjmuje się, że:

dla **ode45**:

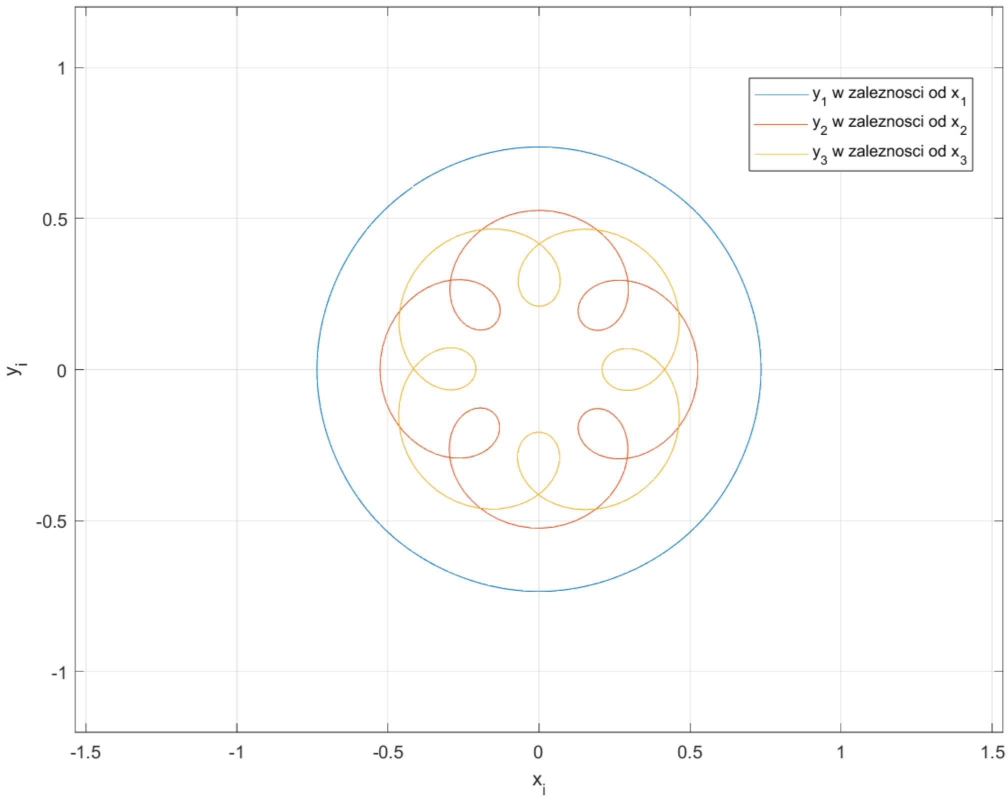
- $AbsTol = 10^{-8}$ odpowiada za to, by błąd absolutny **ode45** był mniejszy od 10^{-8},
- $RelTol = 10^{-8}$ odpowiada za to, by błąd względny **ode45** był mniejszy od 10^{-8},

dla **fminsearch**:

- $TolX = 10^{-8}$ określa minimalną różnicę między wartościami zmiennych w kolejnych iteracjach, po osiągnięciu której algorytm zatrzymuje się,
- $TolFun = 10^{-8}$ określa minimalną różnicę między wartościami funkcji celu w kolejnych iteracjach, po osiągnięciu której algorytm zatrzymuje się,
- $MaxFunEvals = 100000$ ogranicza liczbę wywołań funkcji celu, aby uniknąć nieskończonych obliczeń - po osiągnięciu tego limitu, optymalizacja się zatrzymuje,
- $MaxIter = 20000$ ogranicza liczbę kroków algorytmu optymalizacji - jeśli liczba iteracji przekroczy tę wartość, optymalizacja kończy się,

- *Display* = *none* nie będą wyświetlane żadne informacje na temat przebiegu optymalizacji

Uzyskany wektor **p** wartości początkowych można teraz wykorzystać, by zgodnie z poleceniem zadania wyznaczyć współrzędne położenia rozważanych obiektów w chwilach t zapisanych w tabeli **time**. Używa się do tego tej samej funkcji **ode45** co wyżej, przyjmując za początkowe parametry wartości **p**. Następnie za pomocą procedury **plot** należy stworzyć wykres, który przedstawi uzyskane rozwiązania. Wykres widać na Rysunku 1:



Rysunek 1: Trajektorie y_i w zależności od x_i

Teraz należy sprawdzić, czy rozwiązanie spełnia warunki zadania. Za w pełni zadowalające można uznać rozwiązanie charakteryzujące się dokładnością $\Delta \leqslant 4 \cdot 10^{-3}$. Tutaj po użyciu funkcji **test_solution_57** uzyskuje się dokładność: $\Delta = 3.63 \cdot 10^{-3}$ co spełnia warunki zadania.

4 Komplikacje algorytmu

Ta część raportu poświęcona zostaje omówieniu komplikacji z którymi spotkano się podczas tworzenia tego algorytmu. Zostaną zamieszczone opisy błędów i ich rozwiązania.

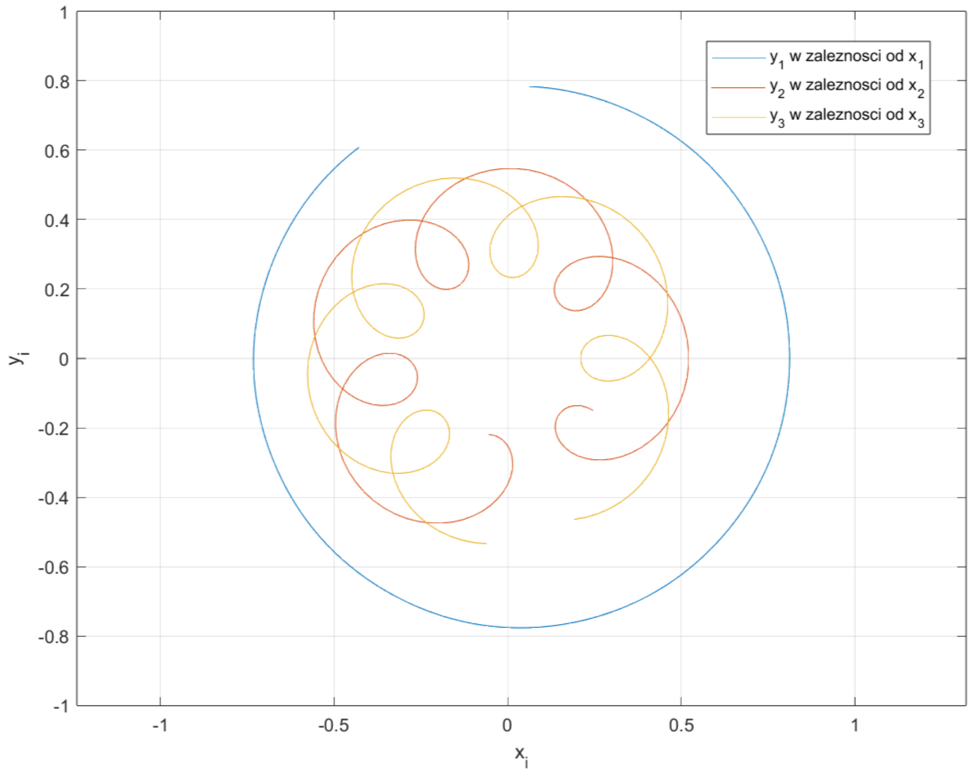
4.1 Nieefektywne wykonywanie operacji

Tworząc początkową część rozwiązania, a dokładniej wyznaczając przybliżoną masę, zauważono, że program bardzo długo się wykonuje. Nie pojawiały się żadne widoczne błędy w rozwiązaniach i wykresach, co na początku utrudniło zdiagnozowanie problemu, natomiast czas oczekiwania był nietypowy dla tej części kodu, która powinna działać najszybciej. Po wnikliwej analizie kodu odpowiedzialnego za wyznaczanie obu stron Układu Równań 1 oraz porównywanie wyników ujawniła się przyczyna. Zamiast dla każdego punktu w czasie t wyznaczać lewą stronę każdego równania jednokrotnie, a następnie wyliczać prawą dla każdej wartości z $\mathbf{m}$, kod ponownie obliczał lewą i prawą stronę Układu Równań 1 dla każdej masy oraz każdej chwili. Powtarzanie tych samych operacji dla tego samego zestawu danych znacząco obniżało wydajność algorytmu. Po optymalizacji, polegającej na zamianie tych operacji na: wybranie chwili t, wyznaczenie lewej strony równań, wybranie masy *m(i)*, wyznaczenie prawej strony równań, a następnie obliczenie błędów, algorytm zaczął działać szybciej i bardziej efektywnie.

4.2 Za niska dokładność rozwiązań

W zależności od przyjętych parametrów opcjonalnych dla **ode45** i **fminsearch** można uzyskać bardzo różne rozwiązania. Wartości tych parametrów mają istotne znaczenie dla dokładności i wydajności obliczeń, co jest szczególnie kluczowe w zadaniach wymagających precyzyjnych wyników. Z początku przyjęte wartości parametrów były nieoptymalne. Liczba wykonywanych iteracji była stanowczo za mała, co skutkowało przedwczesnym zakończeniem działania algorytmu, zanim osiągnięto zadowalające przybliżenie. Dodatkowo, dokładność do której dążył algorytm była stanowczo za duża, co również pogorszyło jakość rozwiązań. W związku z tym powstawały rozwiązania o dokładności na przykład: $\Delta = 4.90 \cdot 10^{-1}$, co nie spełniało warunków zadania. Dopiero po dostosowaniu parametrów, takich jak maksymalna liczba iteracji oraz kryterium stopu, udało się uzyskać wyniki o znacząco lepszej precyzji, spełniające wymagania problemu.

Rozwiązanie o zbyt niskiej dokładności:



Rysunek 2: Błędne trajektorie y_i w zależności od x_i

5 Spis rysunków i programów

5.1 Spis rysunków

5.2 Spis programów

plik: proj2.m

```
function proj2()

    data = readtable('data_57.csv');
    time = readtable('query_57.csv');

    G = 1;
    m = linspace(1/10, 10, 100);

    bledy = zeros(1, length(m));

    function p = pom(j, a, b, c)
        h2 = ((data.t(j+1)-data.t(j-1))/2)^2;
        p = (a - 2*b + c)/h2;
    end

    for j = 2:height(data.t)-1
        x = zeros(3);
        y = zeros(3);
        x(1) = data.x1(j);
        y(1) = data.y1(j);
        x(2) = data.x2(j);
        y(2) = data.y2(j);
        x(3) = data.x3(j);
        y(3) = data.y3(j);

        wyn = zeros(1, 6);
        wyn(1) = pom(j, data.x1(j+1), data.x1(j), data.x1(j-1));
        wyn(2) = pom(j, data.y1(j+1), data.y1(j), data.y1(j-1));
        wyn(3) = pom(j, data.x2(j+1), data.x2(j), data.x2(j-1));
        wyn(4) = pom(j, data.y2(j+1), data.y2(j), data.y2(j-1));
        wyn(5) = pom(j, data.x3(j+1), data.x3(j), data.x3(j-1));
        wyn(6) = pom(j, data.y3(j+1), data.y3(j), data.y3(j-1));

        for i = 1:length(m)
```

```
            P = Pstrona(m(i), x, y);
            L = wyn;
            for k = 1:6
                bledy(i) = bledy(i) + (P(k)-L(k))^2;
            end
            if( j == height(data.t)-1 )
                if(i == 1)
                    masa = m(i);
                    bledy_min = bledy(i);
                elseif( bledy(i) < bledy_min )
                    masa = m(i);
                    bledy_min = bledy(i);
                end
            end
        end
    end
    % mam juz wstepna mase

    ode_opts = odeset('AbsTol', 1e-12, 'RelTol', 1e-10);

    % wartosci poczatkowe
    h = data.t(2) - data.t(1);
    v1x_0 = (data.x1(2) - data.x1(1)) / h;
    v1y_0 = (data.y1(2) - data.y1(1)) / h;
    v2x_0 = (data.x2(2) - data.x2(1)) / h;
    v2y_0 = (data.y2(2) - data.y2(1)) / h;
    v3x_0 = (data.x3(2) - data.x3(1)) / h;
    v3y_0 = (data.y3(2) - data.y3(1)) / h;

    p0 = [data.x1(2), data.y1(2), data.x2(2), data.y2(2), data.x3(2), data.y3(2),
        v1x_0, v1y_0, v2x_0, v2y_0, v3x_0, v3y_0, masa];

    function dydt = odefun(t, y, masa)
        dydt = zeros(12,1);
        for i = 1:6
            dydt(i,1) = y(i+6);
        end

        r12 = (sqrt((y(3)-y(1))^2 + (y(4)-y(2))^2))^3;
        r31 = (sqrt((y(1)-y(5))^2 + (y(2)-y(6))^2))^3;
        r23 = (sqrt((y(5)-y(3))^2 + (y(6)-y(4))^2))^3;

        G_masa = 1*masa;

        dydt(7,1) = G_masa*((y(3)-y(1))/r12 + (y(5)-y(1))/r31);
        dydt(8,1) = G_masa*((y(4)-y(2))/r12 + (y(6)-y(2))/r31);
        dydt(9,1) = G_masa*((y(5)-y(3))/r23 + (y(1)-y(3))/r12);
        dydt(10,1) = G_masa*((y(6)-y(4))/r23 + (y(2)-y(4))/r12);
        dydt(11,1) = G_masa*((y(1)-y(5))/r31 + (y(3)-y(5))/r23);
        dydt(12,1) = G_masa*((y(2)-y(6))/r31 + (y(4)-y(6))/r23);
    end

    function suma = crit(p0, data)
        masa = p0(end);
        [t, y] = ode45(@(tt,yy) odefun(tt,yy,masa), data.t, p0(1:12), ode_opts);

        errors = (data.x1 - y(:, 1)).^2 + (data.y1 - y(:, 2)).^2 + ...
         (data.x2 - y(:, 3)).^2 + (data.y2 - y(:, 4)).^2 + ...
```

```
          (data.x3 - y(:, 5)).^2 + (data.y3 - y(:, 6)).^2;

        suma = sum(errors);
    end
    opts = optimset('TolX', 1e-8, 'TolFun', 1e-8, 'MaxFunEvals', 100000, ...
                    'MaxIter', 20000, 'Display', 'none');

    [p, J] = fminsearch(@(p) crit(p, data), p0, opts);

    tspan_2 = time.t;
    [t_wyn, y_wyn] = ode45(@(tt,yy) odefun(tt,yy,p(13)), tspan_2, p(1:12), ode_opts);

    labels = {'y_1 w zaleznosci od x_1', 'y_2 w zaleznosci od x_2', 'y_3 w zaleznosci od x_3', 'Interpreter','latex'};
    ll = {'x_i', 'y_i', 'Interpreter','latex'};
    figure('Name', 'Trajektorie', 'NumberTitle', 'off');
    grid on
    plot(y_wyn(:,1), y_wyn(:,2), 'DisplayName', labels{1}); hold on;
    plot(y_wyn(:,3), y_wyn(:,4), 'DisplayName', labels{2});
    plot(y_wyn(:,5), y_wyn(:,6), 'DisplayName', labels{3});
    xlabel(ll{1});
    ylabel(ll{2});
    legend show;
    legend location best;
    %title('Trajektoria x_i w zaleznosci od y_i', 'Interpreter','latex');
    grid on;
    xlim([-1.2 1.2]);
    ylim([-1.2 1.2]);
    axis equal
    set(gcf, 'Color', 'w')

    test_solution_57(y_wyn(:,1), y_wyn(:,2), y_wyn(:,3), y_wyn(:,4), y_wyn(:,5), y_wyn(:,6));

end
```

plik: Pstrona.m

```
function wyn = Pstrona(m, x, y)

   function rwyn = rcount(j, k)
      rwyn = (sqrt((x(k)-x(j))^2 + (y(k)-y(j))^2))^3;
   end

   r12 = rcount(1,2);
   r31 = rcount(3,1);
   r23 = rcount(2,3);

   wyn = zeros(1, 6);
   G = 1;

   wyn(1) = G*m*(x(2)-x(1))/r12 + G*m*(x(3)-x(1))/r31;
   wyn(2) = G*m*(y(2)-y(1))/r12 + G*m*(y(3)-y(1))/r31;
   wyn(3) = G*m*(x(3)-x(2))/r23 + G*m*(x(1)-x(2))/r12;
   wyn(4) = G*m*(y(3)-y(2))/r23 + G*m*(y(1)-y(2))/r12;
   wyn(5) = G*m*(x(1)-x(3))/r31 + G*m*(x(2)-x(3))/r23;
   wyn(6) = G*m*(y(1)-y(3))/r31 + G*m*(y(2)-y(3))/r23;

end
```

Bibliografia

[1] MATLAB Help Center, url: https://www.mathworks.com/help/matlab/

[2] dr hab. inż. Kajetana Marta Snopek, ogólnodostępne materiały wykładowe

[3] dr inż. Jakub Wagner, wsparcie w walce z głupimi błędami