Arkadiusz Stramko 155476, Weronika Krejner 155416 gr. 1 DSwP

Projekt nr 10: Rzut piłką do kosza.

Opis problemu

Zbadanie tolerancji kąta w chwili rzutu piłką do koszykówki za trzy punkty i zbadanie minimalnej prędkości początkowej piłki, żeby miała szansę wpaść do kosza. Wykonanie wizualizacji lotu piłki.

Dane liczbowe:

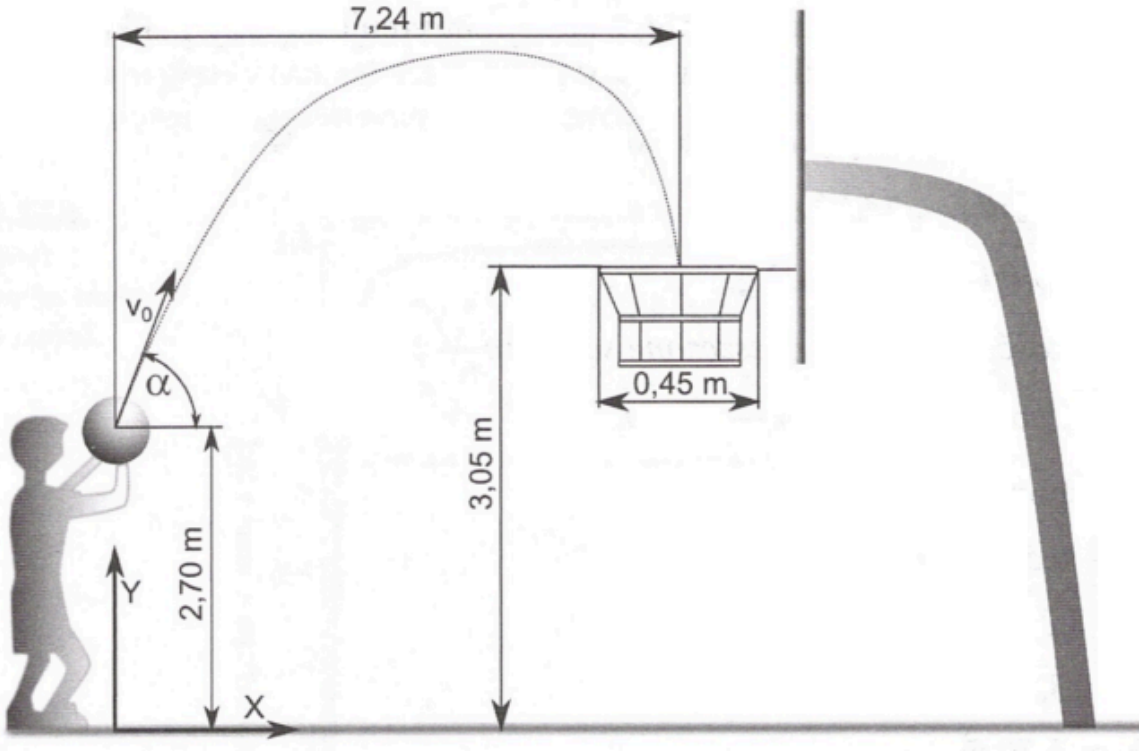
m = 0.58 kg - masa piłki,
l = 7.24 m - linia rzutów za trzy punkty (od środka obręczy),
h0 = 3.05 m - wysokość, na jakiej umieszczone są kosze,
hk = 2.70 m - wysokość z jakiej rzucana jest piłka,
cx = cy = 0.45 - współczynniki oporu,
D0 = 0.45 m - średnica obręczy kosza,
D1 = 0.45 m - średnica piłki,
ϱp = 1.2 kg/m^3 - gęstość powietrza,
g = 9.81m/s^2 - przyspieszenie ziemskie.

Założenia upraszczające:

1. Piłka wpadnie do kosza, jeżeli środek piłki znajdzie się wewnątrz obręczy na wysokości promienia piłki.
2. Gra w koszykówkę odbywa się w hali (brak wiatru).
3. Pole grawitacyjne jest jednorodne i wirowanie piłki nie ma wpływu na wpadnięcie piłki do kosza.
4. Układ współrzędnych jest układem dwuwymiarowym XY.
5. Brak tarczy kosza, piłka nie może odbić się od tarczy i wpaść do kosza.

Te założenia sprowadzają zadanie do rzutu ukośnego.

Rysunek poglądowy



Model matematyczny

Układ równań ruchu piłki w tym układzie

$$\begin{cases} m\ddot{x} &= -P_x, \\ m\ddot{y} &= -P_y - mg. \end{cases}$$

Równania modeli oporu powietrza, z lewej liniowy i z prawej kwadratowy

$$\begin{array}{rclcrcl} P_x &=& \frac{1}{2}c_x\varrho_p S\dot{x} & \text{lub} & P_x &=& \frac{1}{2}c_x\varrho_p S\dot{x}^2, \\ P_y &=& \frac{1}{2}c_y\varrho_p S\dot{y} & \text{lub} & P_y &=& \frac{1}{2}c_y\varrho_p S\dot{y}^2. \end{array}$$

Przyjmujemy warunki początkowe:

x(0) = 0,
$\dot{x}(0) = v_0 \cos\alpha$,
y(0) = hk,
$\dot{y}(0) = v_0 \sin\alpha$.

Opis metod i algorytmów

1. Rozwiązywanie równań różniczkowych (solve_ivp i calculate_motion_derivatives)

Metoda numeryczna **solve_ivp** oparta na adaptacyjnych schematach integracyjnych, np. RK45 (odwzorowanie ode45 w matlabie) z biblioteki **scipy.integrate**. Użyta do symulacji ruchu piłki w dwuwymiarowej przestrzeni pod wpływem oporu powietrza i grawitacji, poprzez rozwiązanie układu równań różniczkowych zwyczajnych (ODE).

Funkcja **calculate_motion_derivatives** oblicza pochodne prędkości i przyspieszenia przy wybranym modelu liniowym lub kwadratowym.

Parametry początkowe opisują stan piłki w chwili rzutu: [x, vx, y, vy].

2. Obliczanie trajektorii (calculate_trajectory)

Wylicza trajektorię piłki od momentu rzutu do końca podanego czasu symulacji. Wynik metody wykorzystywany jest później do animacji jak i do sprawdzania czy piłka trafiła do kosza.

3. Warunki zatrzymania (calculate_stop_frame, check_stop_condition)

Funkcje sprawdzają miejsce zatrzymania piłki, tzn. czy piłka trafiła do kosza, odbiła się od niego lub w ogóle nie trafiła w kosz.
Sprawdzane warunki:

1. Piłka nie doleciała do kosza lub odbiła się od obręczy - w tym warunku sprawdzamy czy wartość środka piłki na płaszczyźnie **x** w danym położeniu znajduje się przed przodem kosza. Obliczana jest również odległość środka piłki od przodu kosza poprzez odległość euklidesową aby sprawdzić czy piłka dotknęła kosza.
2. Piłka wpadła do kosza - sprawdzamy czy środek piłki znajduje się nad koszem nie wyżej niż wysokość kosza powiększona o promień piłki.
3. Piłka przeleciała nad koszem lub odbiła się od obręczy - w tym warunku sprawdzamy czy wartość środka piłki na płaszczyźnie **x** w danym położeniu znajduje się za końcem kosza. Jak w punkcie 1. obliczana jest odległość środka piłki do tyłu kosza aby sprawdzić czy piłka dotknęła kosza.
4. Aby wcześniej zatrzymać animację w razie gdyby żaden powyższy warunek nie został spełniony, sprawdzamy czy piłka spadła poniżej wysokości z której została rzucona lub wypadła za boisko.

4. Animacja trajektorii (animate_trajectory)

Funkcja odpowiadająca za tworzenie animacji z obliczonej trajektorii. Wykorzystuje metodę **FuncAnimation** z biblioteki **matplotlib.animation**, która przyjmuje funkcję **animate** określającą jak co klatkę ma zmieniać się wykres.
Aby skrócić czas trwania animacji redukuje ilość punktów z pierwszej fazy lotu piłki (odległe od kosza, czyli mniej istotne) ale pozostawia nietkniętą krytyczną część lotu piłki, czyli tą, która zatrzymuje piłkę. Aby lepiej móc zaobserwować relację piłki i kosza fragment pola zawierający kosz został dodatkowo wstawiony w oddzielny ekran. Ekran ten zawiera powiększony kosz wraz z dodatkowymi liniami pomocniczymi określającymi obszar, w którym jeśli znajdzie się środek piłki, to uznajemy, że piłka wpadła do kosza.

5. Szukanie kombinacji trafień (find_scoring_combinations, check_if_ball_scored)

W funkcji **find_scoring_combinations** został zastosowany algorytm typu **brute-force,** który dla podanej maksymalnej prędkości w m/s oraz podanej ilości prędkości tworzy na ich podstawie listę prędkości. Następnie zostaje wykonana iteracja po tej liście. Dla każdego elementu wykonywane jest obliczenie trajektorii lotu piłki bazujące na kombinacji kątów z zakresu od 0° do 90° z krokiem 1°. Dla każdej takiej iteracji obliczana jest klatka końcowa i sprawdzany jest rezultat rzutu przy pomocy funkcji **check_if_ball_scored**. Kombinacje prędkości i kąta, dla których piłka wpadła do kosza zapisywane są do listy i wyświetlane po zakończeniu przebiegu algorytmu.

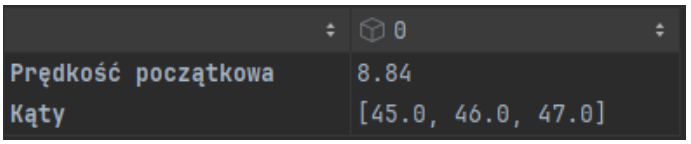
6. Wykres i tabela zależności kątów od prędkości (show_speeds_angles_chart)

Funkcja rysująca dla podanych trafiających kombinacji wykres zależności kątów od prędkości początkowej oraz zwracająca te dane w formie tabelarycznej.

Wnioski

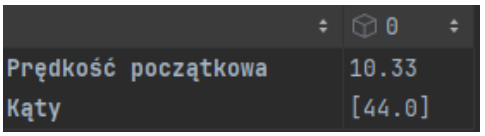
Wybranie odpowiedniego modelu oporu może okazać się kluczowe gdyż prędkość minimalna rzutu jak i zakresy kątów dla obu modeli są różne co można zaobserwować na wykresach prędkości i ich kątów zawartych w wizualizacjach poniżej. Badanie prędkości i ich kątów odbywa się dla prędkości od 0 do 100 m/s z uwzględnieniem do 2 miejsc po przecinku co otrzymujemy poprzez próbkowanie prędkości dla 10tys. wartości. Kąty wyznaczane są od 0° - 90° z przejściem o 1°.

Minimalna prędkość dla modelu liniowego wyniosła 8.84 m/s a zakres kątów, które uznane zostały za trafione wynosi 45° - 47°.



	0
Prędkość początkowa	8.84
Kąty	[45.0, 46.0, 47.0]

Dla porównania minimalna prędkość dla modelu kwadratowego jest większa i wynosi ona 10.33 m/s a zakres kątów jest dużo mniejszy ponieważ zawiera on jedynie kąt 44°.



	0
Prędkość początkowa	10.33
Kąty	[44.0]

Największy nieprzerwany zakres kątów, występujący w modelu liniowym, otrzymujemy dla prędkości początkowych 9.09m/s oraz 9.10 m/s i jego zakres to 37° - 55° (18° zakresu). Powyżej tych prędkości otrzymujemy po dwa przedziały kątów, a następnie maksymalnie dwa lub jeden kąt gwarantujący trafienie. Dla modelu kwadratowego prędkości o największym nieprzerwanym zakresie kątów to 10.73-10.77 m/s, a ich zakres to 34° - 54° (20° zakresu). Otrzymaliśmy więcej prędkości mieszczących się w szerszym zakresie co daje nam większe szanse na trafienie.

Na obu wykresach zależności możemy też zauważyć, że dla małych prędkości zakres znalezionych kątów tworzy nieprzerwaną linie ale wraz ze wzrostem prędkości trudniej jest znaleźć odpowiedni kąt, zatem w prawej części wykresów widzimy wyraźne przerwy. Dla modelu liniowego te przerwy zaczynają być widoczne dużo szybciej niż dla modelu kwadratowego, z tego wynika, że dla dużych prędkości w modelu kwadratowym mamy większą szansę na trafienie.

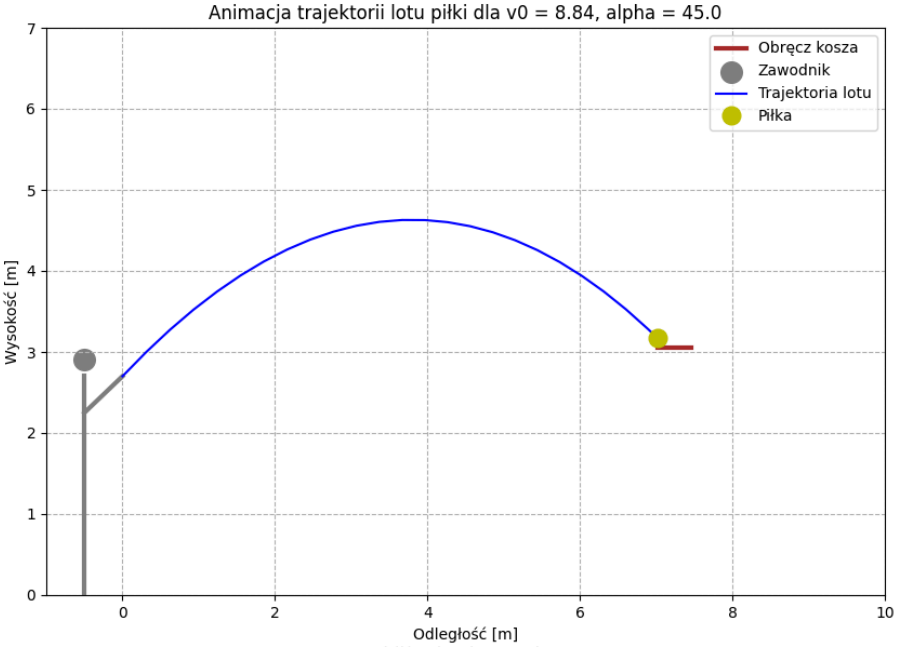
Maksymalna znaleziona prędkość dla tego modelu liniowego wynosi 25.87 m/s i jest ona znacznie mniejsza niż w modelu kwadratowym, gdzie wynosi ona 84.73 m/s.

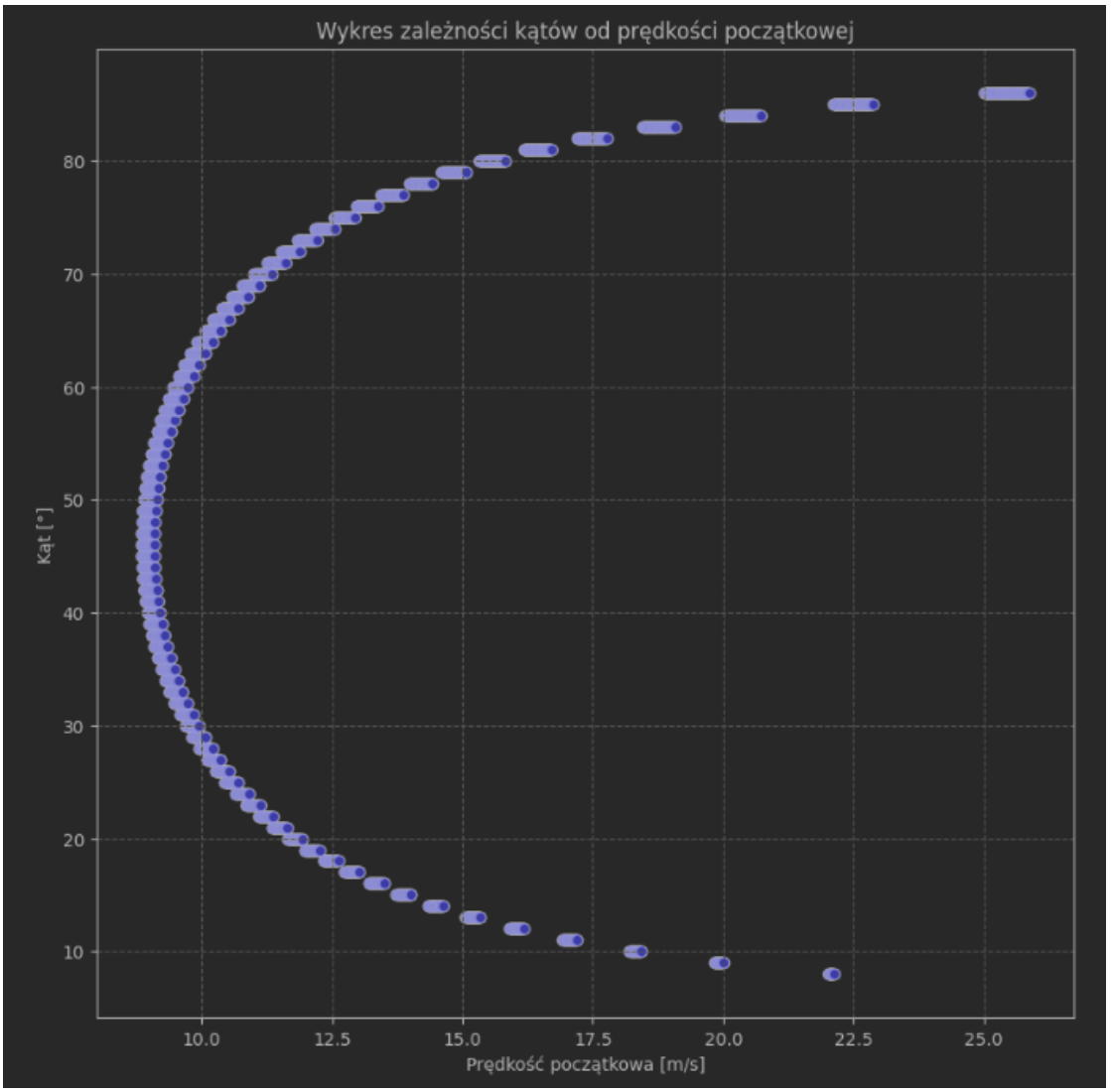
Odnosząc się teraz do trajektorii lotu piłki możemy zaobserwować, że w modelu liniowym jest ona symetrycznym łukiem, wiąże się to ze zmniejszonym oporem powietrza, natomiast w modelu kwadratowym obserwujemy, że prawa część trajektorii jest bardziej stroma.

Wizualizacja

Model oporu powietrza liniowy

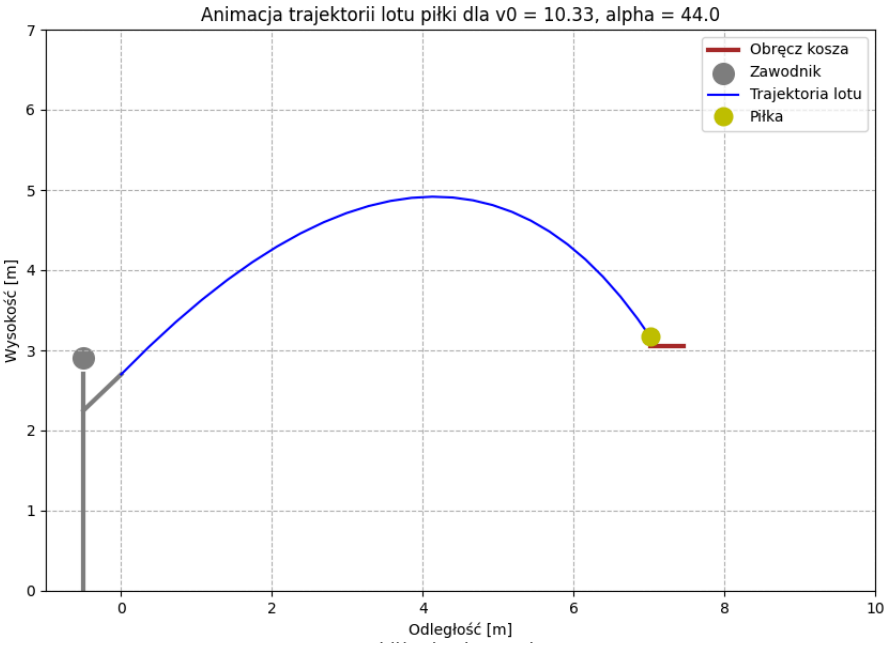
minimalna prędkość = 8.84
przedział kątów = [45° - 47°]

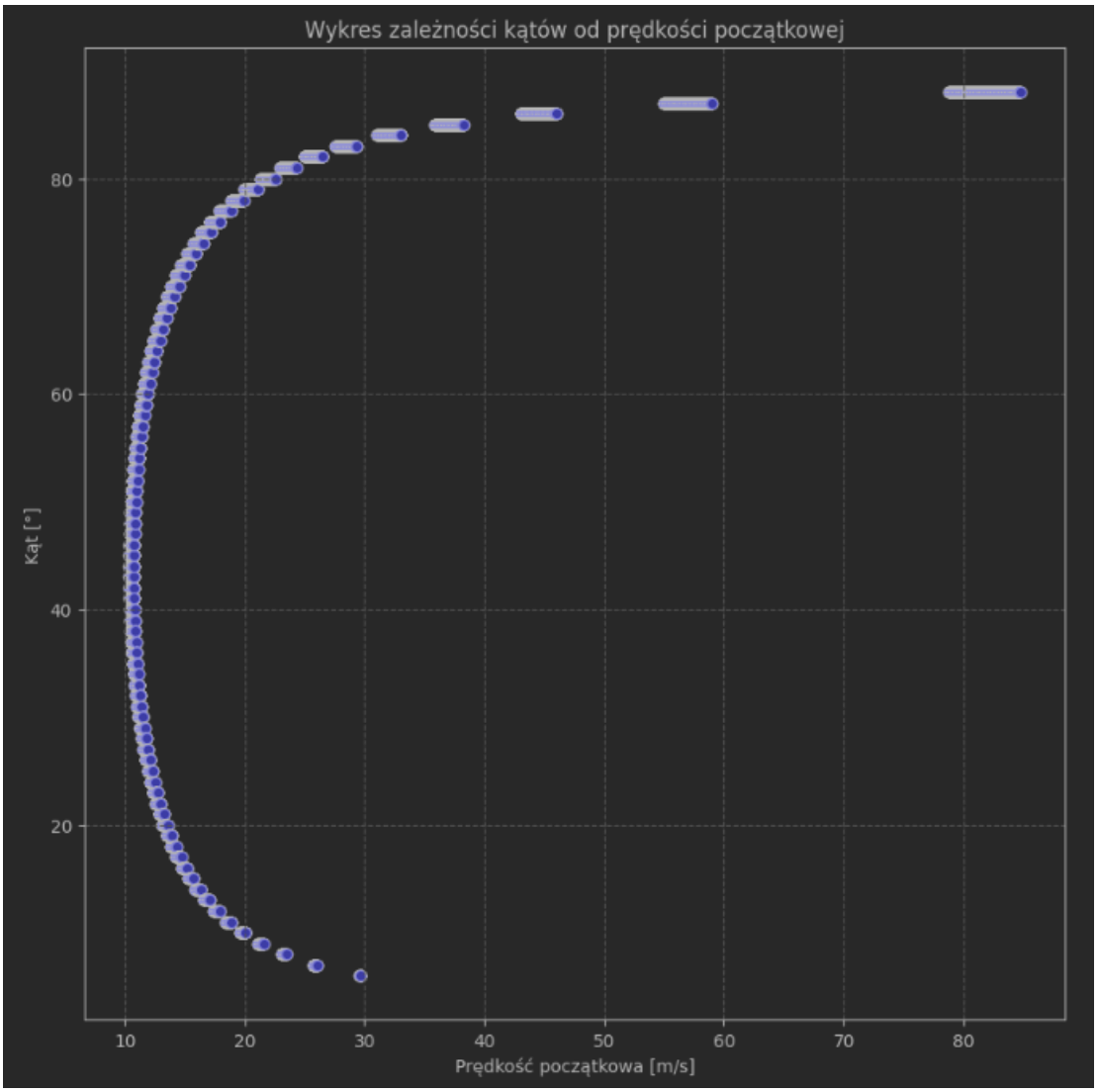




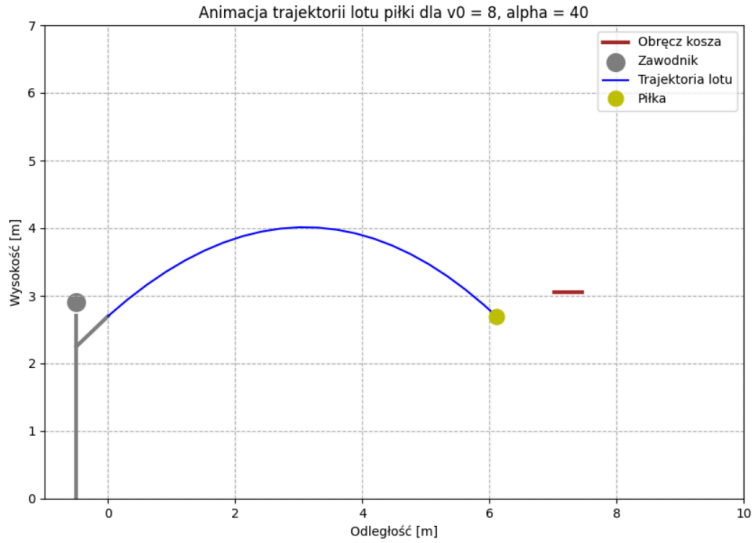
Model oporu powietrza kwadratowy

minimalna prędkość = 10.33
przedział kątów = [44°]

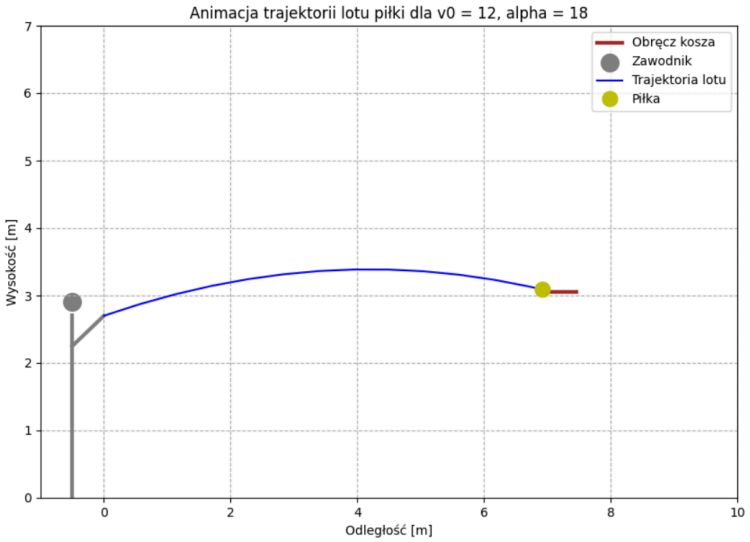




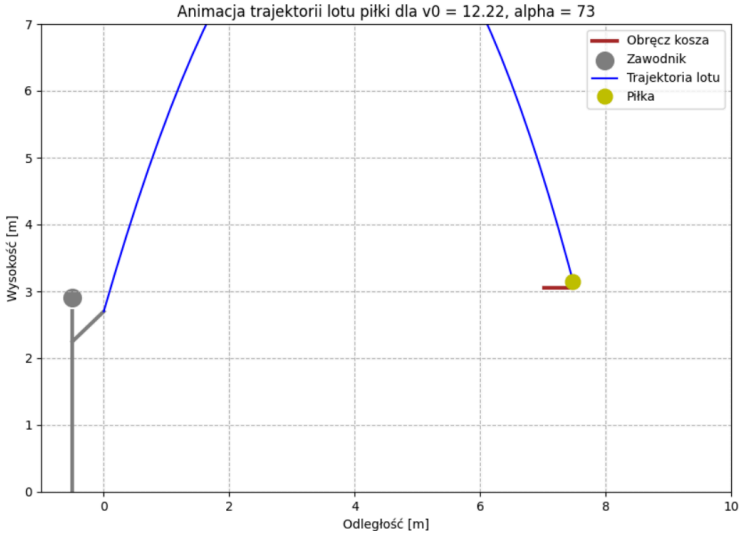
Przykłady gdy piłka nie trafia w kosz



Piłka nie trafiła do kosza.



Piłka nie doleciała do kosza lub odbiła się od obręczy.



Piłka przeleciała za kosz lub odbiła się od obręczy.

Problematyczne elementy

- stworzenie odpowiedniej animacji, kompatybilność animacji w jupyter notebook
- dobór odpowiednich parametrów
- czas obliczeń do dokładnego próbkowania
- walidacja stworzonych warunków kolizji, w szczególności brzegowych
- stworzenie przybliżenia kosza gdy zbliża się piłka w odpowiednich proporcjach
- interpretacja wzorów fizycznych, przekształcenie ich oraz znalezienie odpowiedniego solvera

Instrukcja włączenia projektu

Wymagania środowiskowe

Wersja python: **3.10** lub **3.12**.

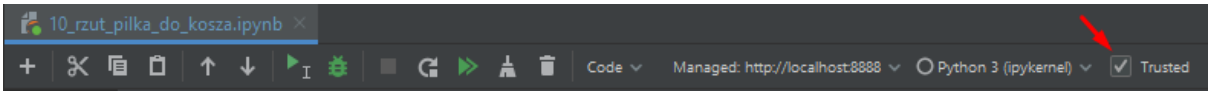
Instalacja wymaganych bibliotek z pliku **requirements.txt**:
pip install -r ./requirements.txt

Sprawdzone wersje PyCharm z wersją pythona:

- PyCharm 2023.3.3 (Professional Edition) oraz python 3.12
- PyCharm 2024.1.1 (Professional Edition) oraz python 3.10

Uruchomienie pliku (po instalacji odpowiednich bibliotek)

Aby móc odtworzyć animację w PyCharm należy zaznaczyć plik jako *Trusted*.

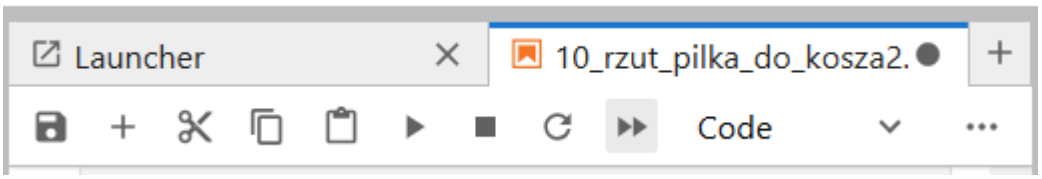


Plik jest już skompilowany dla dużego próbkowania, ale można go ponownie skompilować dla innego próbkowania zmieniając parametry (przykład na dole sprawozdania) i klikając 2 zielone strzałki *Run All*.

Jest też uniwersalny sposób bez potrzeby posiadania PyCharm - jupyter lab w przeglądarce.

W terminalu z poziomu folderu w którym znajduje się plik należy wpisać:
jupyter lab --ip 0.0.0.0 --no-browser

Następnie należy wykorzystać 1 z linków podanych w terminalu aby dostać się do jupyter lab. Plik można skompilować za pomocą podwójnej strzałki.



Dla parametrów użytych do wykonania zadania, obliczenia związane z wyszukaniem par prędkości i kątów mogą wykonywać się nawet do 3 godzin.

number_of_points = 10000
points_rarefaction_divider = 500
v_max = 100.
number_of_v = 10000

Dla celów sprawdzenia działania kodu zalecane jest zmniejszenie tych parametrów (wyniki dla zmniejszonej próbki mogą nie być prawidłowe):

number_of_points = 500
points_rarefaction_divider = 25
v_max = 30.
number_of_v = 300