Symulowanie rzutu ukośnego

Zuzanna Szester
Magdalena Syrek
Marzena Bielecka

Wydział Fizyki, Matematyki i Informatyki
Politechnika Krakowska im. T.Kościuszki

20 czerwca 2018

Spis treści

1. Rzut ukośny
2. Modelowanie ruchu obiektu
3. Wyniki
4. Wykres
5. Makefile

Rzut ukośny - definicja

Rzut ukośny

Jest to ruch w jednorodnym polu grawitacyjnym z prędkością początkową vo o kierunku ukośnym do kierunku pola. Ruch ten odpowiada ruchowi ciała rzuconego pod kątem do poziomu.

Rzut ukośny

Rozważając rzut ukośny wiemy, że ruch w kierunku poziomym odbywa się bez przyśpieszenia, dlatego też składowa pozioma prędkości vx obiektu nie ulega zmianie podczas ruchu i jest równa początkowej prędkości vo. Wiemy, że w rzucie ukośnym parametryczne równania ruchu są zapisane następująco

$$x(t) = v_x t = (v_o cos\theta)t$$

$$y(t) = v_y t - \frac{1}{2}gt^2 = (v_o sin\theta)t - \frac{1}{2}gt^2$$

Rzut ukośny

Naszym celem jest wyznaczenie toru obiektu, któremu nadano prędkość początkową vo i skierowana pod kątem θ do poziomu. Obliczenia możemy prowadzić w przedziale czasu (to,t) otrzymując wartości położenia, współrzędne x i y. Zasadnicze obliczenia przeprowadza funkcja simulate():

simulate(fx,20.0,30.0,0.1,1.0)

Odpowiednio są to wartości: vo, α,dt,eps.

Modelowanie ruchu obiektu

main.cpp (~/rzut ukośny/src) -

Otwórz

```
#include <stdio.h>
#include <curses.h>
#include <math.h>
#include "simulate.h"

using namespace std;
double fx(double t);

int main()
{
simulate(fx,20.0,30.0,0.1,1.0);
getch();
return 0;
}

double fx(double t)
{
return 9.81*t;
}
```

Funkcja simulate()

Funkcja simulate() otrzymuje następujące parametry:

- prędkość początkową vo
- kąt θ
- krok czasowy dt
- warunek stopu eps

  Symulacje możemy prowadzić w dowolnym przedziale czasowym, w naszym przypadku iteracja jest zatrzymana, gdy wartość współrzędnej y staje się ujemna (obiekt przebił poziom). Dokładność obliczeń zależy od kroku czasowego dt i wartości eps.

Funkcja simulate()

Funkcja simulate() otrzymuje następujące parametry:

- prędkość początkową vo
- kąt θ
- krok czasowy dt
- warunek stopu eps

  Symulacje możemy prowadzić w dowolnym przedziale czasowym, w naszym przypadku iteracja jest zatrzymana, gdy wartość współrzędnej y staje się ujemna (obiekt przebił poziom). Dokładność obliczeń zależy od kroku czasowego dt i wartości eps.

Funkcja simulate()

Funkcja simulate() otrzymuje następujące parametry:

- prędkość początkową vo
- kąt θ
- krok czasowy dt
- warunek stopu eps

  Symulacje możemy prowadzić w dowolnym przedziale czasowym, w naszym przypadku iteracja jest zatrzymana, gdy wartość współrzędnej y staje się ujemna (obiekt przebił poziom). Dokładność obliczeń zależy od kroku czasowego dt i wartości eps.

Funkcja simulate()

Funkcja simulate() otrzymuje następujące parametry:

- prędkość początkową vo
- kąt θ
- krok czasowy dt
- warunek stopu eps
  Symulacje możemy prowadzić w dowolnym przedziale czasowym, w naszym przypadku iteracja jest zatrzymana, gdy wartość współrzędnej y staje się ujemna (obiekt przebił poziom). Dokładność obliczeń zależy od kroku czasowego dt i wartości eps.

Funkcja simulate()

simulate.h (~/rzut ukośny/include) - gedit

Otwórz

```
#ifndef simulate_h
#define simulate_h
#define R 3.1415926/180.0
double simulate(double (*f)(double),double vo,double alfa,double dt,double eps)
{
double v=0.0, vx, vx2, vy;
double x,y;
double t = 0.0;
double co=cos(R*alfa);
double si=sin(R*alfa);
vx= vo*co;
vy= vo*si;
vx2= vx*vx;
v=sqrt(vx2+vy*vy);
printf("\n czas\t x\t y\t vy\t v",t,x,y,vy);
while(y > -eps)
{
printf("\n %5.2f\t %5.2f\t %5.2f\t %6.2f\t %6.2f\t ",t,x,y,vy,v);
t=t+dt;
x=vo*co*t;
y=vo*si*t - 0.5*t*f(t);
vy=vo*si-f(t);
v=sqrt(vx2+vy*vy);
}
printf("\n vx=%6.2f", vx);
return t;
}

#endif
```

Tabela z wynikami

czas	x	y	vy	v
0.00	0.00	0.00	10.00	20.00
0.10	1.73	0.95	9.02	19.53
0.20	3.46	1.80	8.04	19.09
0.30	5.20	2.56	7.06	18.70
0.40	6.93	3.22	6.08	18.36
0.50	8.66	3.77	5.09	18.05
0.60	10.39	4.23	4.11	17.80
0.70	12.12	4.60	3.13	17.60
0.80	13.86	4.86	2.15	17.45
0.90	15.59	5.03	1.17	17.36
1.00	17.32	5.09	0.19	17.32
1.10	19.05	5.06	-0.79	17.34
1.20	20.78	4.94	-1.77	17.41
1.30	22.52	4.71	-2.75	17.54
1.40	24.25	4.39	-3.73	17.72
1.50	25.98	3.96	-4.72	17.95
1.60	27.71	3.44	-5.70	18.23
1.70	29.44	2.82	-6.68	18.56
1.80	31.18	2.11	-7.66	18.94
1.90	32.91	1.29	-8.64	19.36
2.00	34.64	0.38	-9.62	19.81
2.10	36.37	-0.63	-10.60	20.31
vx =	17.32			

Podsumowanie wyników

Z pokazanych symulacji wynika, że z naszymi danymi czas trwania rzutu wynosi około 2 sekundy, zasięg rzutu wynosi 35 metrów, maksymalną wysokość równą 5 metrów obiekt osiąga po jednej sekundzie, a prędkość całkowita w tym momencie jest najmniejsza i wynosi około 17.3 m/s.
W programie obliczamy także składową prędkości vy oraz prędkość całkowitą v. Należy mieć na uwadze, że w tych symulacjach nie uwzględniono oporów powietra, który zdecydowanie potrafi zmienić tor rzuconego obiektu.

Podsumowanie wyników

Ze wzorów parametrycznych możemy wyeliminować czas, a otrzymamy równanie toru ciała.

$$y = (tg\theta)x - \frac{gx^2}{2(v_o cos\theta)^2}$$

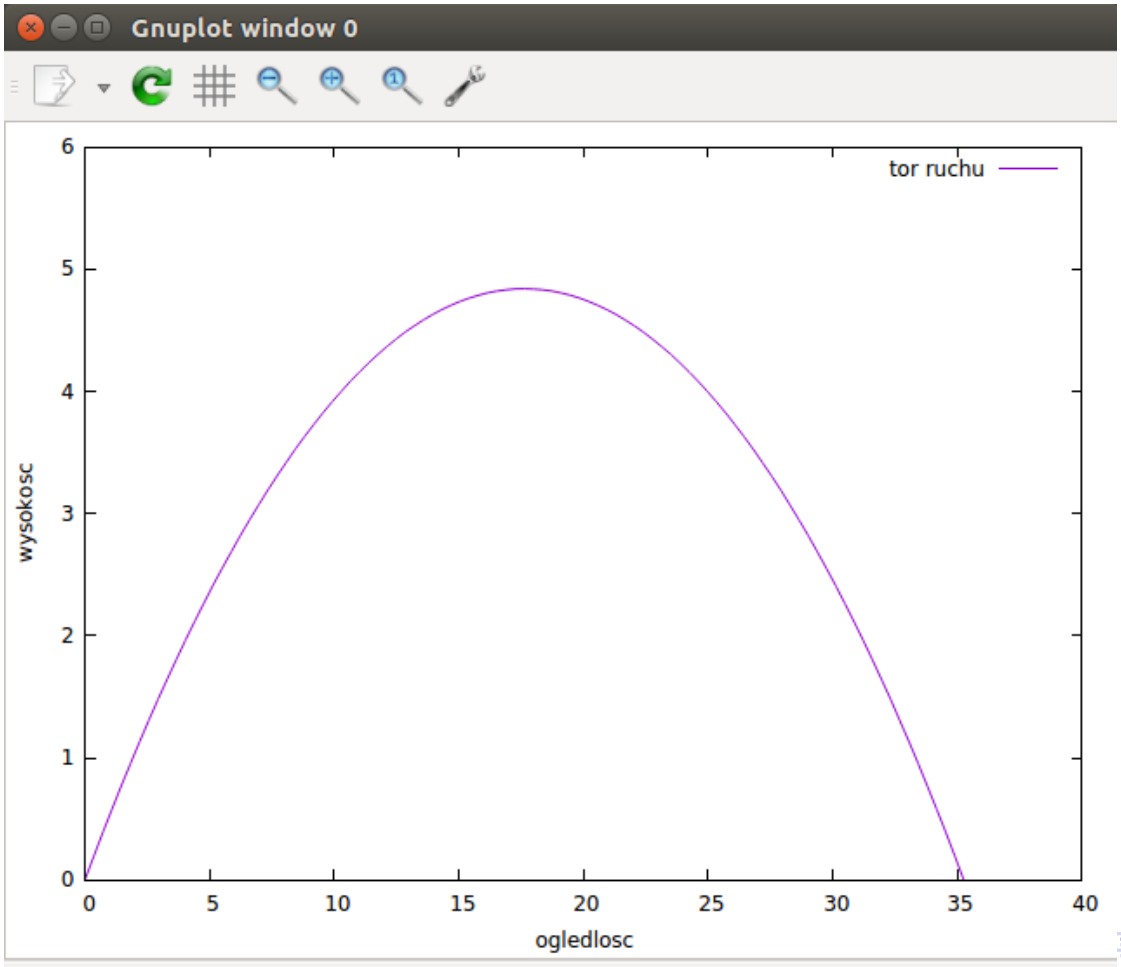
Jak widać jest to równanie typu:

$$y = ax + bx^2$$

Tor ciała w rzucie ukośnym jest paraboliczny.

Wykres

Do wykonania wykresu użyłyśmy środowiska gnuplot.



Makefile

Do kompilacji programu stworzyłyśmy plik Makefile.

Makefile (~/rzut ukośny) - gedit

Otwórz Zapisz

```
CXX := g++ # glowny kompilator
# CXX := ustalamy zmienna CXX
SRCDIR := src
BUILDDIR := build
TARGET := bin/wyniki.x

SRCEXT := cpp
LDFLAGS_LINUX = -lpthread -lncurses
LDFLAGS       = $(LDFLAGS_LINUX)
SOURCES := $(shell find $(SRCDIR) -type f -name *.$(SRCEXT)) #jako zrodlo ustalamy wszystkie pliki
z katalodu src z rozszerzeniem .cpp
OBJECTS := $(patsubst $(SRCDIR)/%,$(BUILDDIR)/%,$(SOURCES:.$(SRCEXT)=.o)) #definiujemy obiekty w
rozszerzeniem .o do zapisu w katalogu build
CFLAGS := -g
INC := -I include
EXEC=$(shell find $(TARGET) -type f -name *.x)

$(TARGET): $(OBJECTS)
        @echo " Laczenie..."
        @echo " $(CXX) $^ $(LDFLAGS_LINUX) -o $(TARGET) $(LIB)"; $(CXX) $^ $(LDFLAGS_LINUX) -o
$(TARGET)

$(BUILDDIR)/%.o: $(SRCDIR)/%.$(SRCEXT)
        @mkdir -p $(BUILDDIR)
        @echo " $(CXX) $(CFLAGS) $(INC) -c -o $@ $<"; $(CXX) $(CFLAGS) $(INC) -c -o $@ $<

clean:
        @echo " Czyszczenie...";
        @echo " $(RM) -r $(BUILDDIR) $(TARGET)"; $(RM) -r $(BUILDDIR) $(TARGET) #czyszczenie
folderu build oraz bin

run: $(TARGGET)
        $(EXEC)

.PHONY: run
.PHONY: clean
```