초기발사속도,목표거리,대기마찰을 고려한 발사각계산

표준탄도 계산(질점 해석)

Author
201902890
이태녕

Contents

1 주제 선정 이유

이 프로젝트는 탄도학과 수치해석이 만나는 지점에서 출발한다. 탄도학과 컴퓨터의 아버지 폰 노이만은 그의 뛰어난 수학적 지식과 탄도학에 대한 탁월한 이해력으로 이 분야에 혁신을 가져왔다. 그의 노력은 수치해석의 발전에도 큰 영향을 주었다. 나는 이 부분에 영감을 받아서 이번 수치해석 수업 개인 프로젝트를 탄도계산으로 선정하였다.

2 배경지식

2.1 진공탄도

진공탄도는 다음과 같이 3가지 상황을 가정한다.
1. 공기가 없기 때문에 공기의 저항은 없다.
2. 지구의 중력 가속도는 항상 평행하다.
3. 지구는 평탄하고 전향력이 작용하지 않는다.

이 때 질량 m인 포탄이 원점에서 각도 θ, 초기속도 v_0 로 발사되었을 때 fig1 과 같이 발사된다.
이 때 x축 방향으로의 포탄의 초속은 $v_0 cos(\theta)$, y축 방향으로의 포탄의 초속은 $v_0 sin(\theta)$이다.

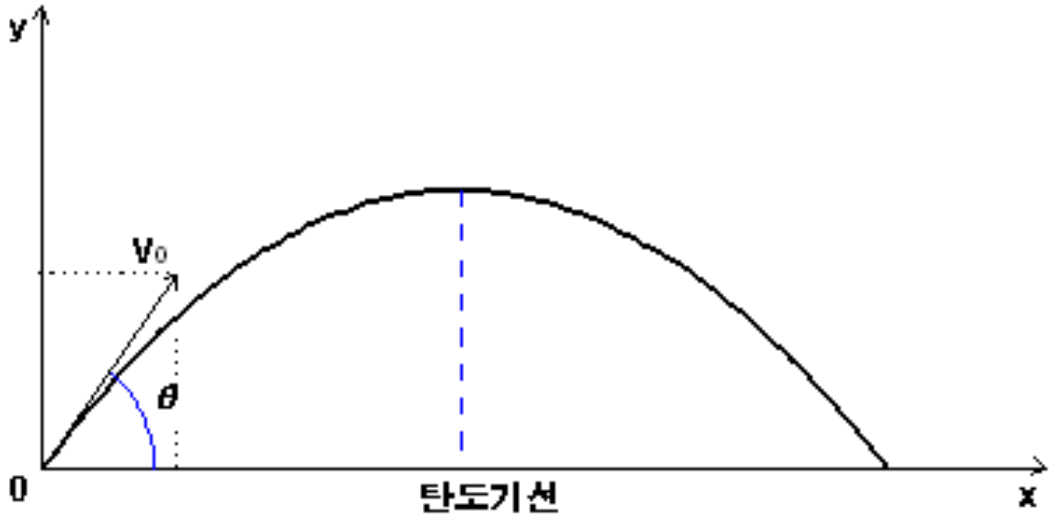

Figure 1: 진공탄도

이 때 포탄의 위치는 $x = (v_0 cos\theta)t$, $y = (v_0 sin\theta)t - \frac{1}{2}gt^2$
각각 대입하면 다음과 같이 진공탄도방정식을 구할 수 있다.

$$y = tan\theta x - \frac{g}{2v_0^2 cos^2\theta}x^2 \tag{1}$$

지면에 떨어질때의 x좌표는 y=0인 좌표이므로 $x = \frac{2v_0^2 cos\theta sin\theta}{g}$
비행시간은 $\frac{2v_0 sin\theta}{g}$

2.2 표준탄도

포탄이 대기중을 비행할 때 외부로부터 중력과 항력만을 받는 하나의 질점(mass point)이라고 가정하여 Newton의 운동방정식을 적용하여 세운 포탄의 운동방정식을 '표준탄도방정식'이라 하고, 이 방정식의 해를 '표준탄도'라고 한다. 진공탄도의 가정에서 항력만을 추가로 고려한다. 이 때 대기는 표준대기 이다.
항력계수(공기저항 등)은 다음과 같이 구할 수 있다. 공기밀도를 ρ, 포탄의 속도를 V, 운동방향에 수직한 포탄의 단면적을 S, 항력을 D라고 할 때, 항력계수 C_D

$$C_D = \frac{D}{0.5\rho V^2 S} \tag{2}$$

다만 이번 프로젝트에서는 항력을 0.5로 고정하였다.

날아가는 포탄에 작용하는 힘은 항력과 중력이 있고, 이것을 각각 x축 y축으로 구분한 것이 표준탄도 방정식이 된다.

$$a_x = -\frac{\pi C_D \rho V^2 d^2}{8m} cos\theta \tag{3}$$

$$a_y = -\frac{\pi C_D \rho V^2 d^2}{8m} sin\theta - g \tag{4}$$

$$\tag{5}$$

위와 같은 방정식을 수치적으로 풀어서 해를 구할수 있는데, 식이 매우 복잡해진다.

2.3 질량 m에 관하여

일반적으로 대기 마찰을 포함한 포물선 운동 문제에서 물체의 질량은 운동 방정식에 영향을 미치지 않는다. 질량이 나타나는 중력항에서 질량이 등장하지만, 질량이 물체의 운동에 미치는 영향은 중력 가속도에 의존하므로 운동 방정식에서 질량이 소거된다.

3 시뮬레이션

3.1 수치해석 방법

표준탄도 방정식에서 우리가 고려하는 것은 항력이다. 대기 마찰계수를 α로 표현하면 이는 속도에 비례하므로 항력 = $-\alpha$·속도 이다. 이를 이용하여 우리는 위의 표준탄도 방정식을 다음과 같이 수정할 수 있다.

$$a_x = -\alpha \cdot v_x \tag{6}$$

$$a_y = -\alpha \cdot v_x - g \tag{7}$$

$$\tag{8}$$

우리는 계산상의 편의를 위해 오일러 방법을 사용한다. 오일러 방법은 미분 방정식을 수치적으로 풀기 위한 한 가지 방법으로, 현재의 미분 값을 이용하여 다음 시간 단계에서의 값을 예측하고 업데이트한다. 시간에 따라 물체의 운동을 시뮬레이션하는 데 사용되는 방법 중 하나이다. 속도

및 위치를 업데이트 하는 오일러 방법은 다음과 같이 나타낼 수 있다.

$$v_x < -v_x - \alpha \cdot v_x \cdot dt \tag{9}$$
$$v_y < -v_y - \alpha \cdot v_x \cdot dt - g \cdot dt \tag{10}$$
$$x < -x + v_X \cdot dt \tag{11}$$
$$y < -y + v_y \cdot dt \tag{12}$$
$$\tag{13}$$

v_x, v_y는 각각 x, y 방향의 속도를 나타내며 x, y는 x와 y방향의 위치를 나타낸다. 이는 등시간 간격(dt) 모델링이다.
위의 수식을 코딩해보면 다음과 같이 쓸 수 있다.

$$vx_new = vx_old - alpha * vx_old * dtvy_new = vy_old - g * dt - alpha * vy_old * dtx_new = x_old + vx_old * dty_new = y_old \tag{14}$$

3.2 코드 및 결과

코드는 다음과 같다.

```
import numpy
import pylab

# Initial conditions
x0 = 0
y0 = 0.0001
dt = 0.005
alpha = 0.5 # (물체에 따라 달라지므로 상수값 0.5 적용)
g = 10.0

# Input the length and velocity
length = float(input("Input the overal length: "))
v0 = float(input("Input the initial velocity: "))

# Guess the initial angle based on the motion without air friction
if g*length/(v0*v0) > 1:
    print("The input is not right. Please put a proper one!")
    exit(1)
else:
    angle0 = 0.5*numpy.arcsin(g*length/(v0*v0))
print("Initial angle in degrees is: ", angle0/numpy.pi*180.0)



x_old = x0
y_old = y0
angle_old = angle0
```

```
fig = pylab.figure(1)
fig.gca().set_aspect("equal")
for iter in range(0, 10):
    vx_old = v0*numpy.cos(angle_old)
    vy_old = v0*numpy.sin(angle_old)
    traj_x = []
    traj_y = []
    theor_x = []
    theor_y = []

    t = 0.0
    x_theor = x0
    y_theor = y0

    while y_old > 0:
        traj_x.append(x_old)
        traj_y.append(y_old)
        theor_x.append(x_theor)
        theor_y.append(y_theor)
        t = t + dt

        vx_new = vx_old - alpha*vx_old*dt
        vy_new = vy_old - g*dt - alpha*vy_old*dt
        x_new = x_old + vx_old*dt
        y_new = y_old + vy_old*dt

        # Analytical solution
        vx_theor = v0*numpy.cos(angle_old) * numpy.exp(-alpha*t)
        vy_theor = (v0*numpy.sin(angle_old) + g/alpha) * numpy.exp(-alpha*t) - g/alpha
        x_theor = x_theor + vx_theor*dt
        y_theor = y_theor + vy_theor*dt

        vx_old = vx_new
        vy_old = vy_new
        x_old = x_new
        y_old = y_new

    pylab.figure(1)
    pylab.plot(traj_x, traj_y)
    pylab.plot(theor_x, theor_y, "+")

    angle_new = angle_old - (x_old-length)/(2.0*v0*v0*numpy.cos(2.0*angle_old)/g)

    print("Prediction and length = ",x_old, length)
    print("Old angle = ", angle_old*180.0/numpy.pi)
```

```
        print("Predicted angle = ", angle_new*180.0/numpy.pi)
        x_old = x0
        y_old = y0
        angle_old = angle_new

pylab.show()
```

결과는 다음과 같다. (length : 70, velocity : 40)
Input the overal length: 70
Input the initial velocity: 40
Initial angle in degrees is: 12.972239886185005
Prediction and length = 42.876752150518435 70.0
Old angle = 12.972239886185005
· · · 중략 · · ·
Old angle = 23.25438775124207
Predicted angle = 27.507451692453458
Prediction and length = 55.034216278764816 70.0
Old angle = 27.507451692453458
Predicted angle = 32.180951120190464
아래 Fig2와 위 결과값을 보면 반복하면서 최적의 발사각도를 찾아가고 있는 것을 알 수 있다.
이 때 계산된 각도는 약 32.18도 이다.

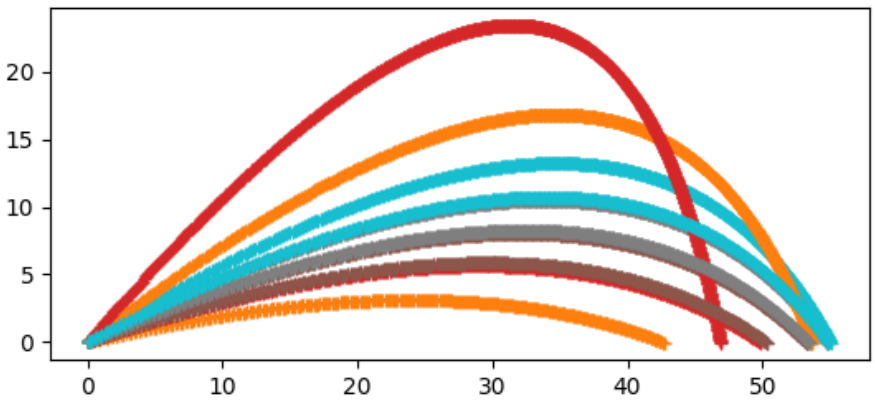

Figure 2: length : 70, velocity : 40

3.3 최적화

위 코드에서는 반복횟수를 몇번으로 줘야하는지, 가장 정확한 값이 어떤 값인지 쉽게 알기는 힘들다. 따라서 scipy의 optimization method를 활용해서 최적의 해를 구한다. scipy.optimize.broyden1 함수는 Scipy 라이브러리에서 제공되는 비선형 방정식의 수치적인 해를 찾기 위한 Broyden-Fletcher-Goldfarb-Shanno (BFGS) 방법 중 하나인 Broyden의 방법을 사용하는 함수이다. Broyden's method는 주어진 초기 추정값에 대해 비선형 방정식의 해를 찾는 반복적인 방법 중 하나이다. 이 방법은 함수의 그래디언트를 직접 계산하는 대신에 야코비안 행렬의 근사치를 이용하여 해를 찾는다. 이 방법은 BFGS 방법과 유사하지만 메모리를 덜 사용하면서도 효과적으로 수렴하는 특징이 있다. 이를 사용하여 초기 각도와 시간을 조정하여 주어진 목표 거리에 도달하는 최적의 발사 각도를 찾을 수 있다.
코드는 다음과 같다.

```
    import numpy
import pylab
import scipy.optimize

alpha = 0.25
g = 10.0

# Input the length and velocity
length = float(input("Input the overal length: "))
v0 = float(input("Input the initial velocity: "))

# Guess the initial angle based on the motion without air friction
if g*length/(v0*v0) > 1:
    print("The input is not right. Please put a proper one!")
    exit(1)

angle_init = 0.5*numpy.arcsin(g*length/(v0*v0))
time_init = 2.0*v0*numpy.sin(angle_init)/g
print("Initial angle in degrees is: ", angle_init/numpy.pi*180.0)
print("Initial time is: ", time_init)

# Theoretical equations
#vx = v0*numpy.cos(angle_old) * numpy.exp(-alpha*t)
#vy = (v0*numpy.sin(angle_old) + g/alpha) * numpy.exp(-alpha*t) - g/alpha
#x = v0*numpy.cos(angle0)/alpha * (1.0 - numpy.exp(-alpha*t))
#y = (v0*numpy.sin(angle0) + g/alpha)/alpha * (1.0-numpy.exp(-alpha*t)) - g*t/alpha

def fun(args):
    x = v0*numpy.cos(args[0])/alpha * (1.0 - numpy.exp(-alpha*args[1])) - length
    y = (v0*numpy.sin(args[0]) + g/alpha)/alpha * (1.0-numpy.exp(-alpha*args[1])) - g*args[1]/alp
    return [x,y]

angle, time=scipy.optimize.broyden1(fun, [angle_init, time_init])
```

```
print("Final angle in degrees = ", angle/numpy.pi*180.0)
print("Time = ", time)

# Final trajectory
t = numpy.linspace(0.0, time, 101)
traj_x = v0*numpy.cos(angle)/alpha * (1.0 - numpy.exp(-alpha*t))
traj_y = (v0*numpy.sin(angle) + g/alpha)/alpha * (1.0-numpy.exp(-alpha*t)) - g*t/alpha

pylab.plot(traj_x, traj_y)
pylab.show()
```

결과는 다음과 같다. (length : 70, velocity : 40) Input the overal length: 70
Input the initial velocity: 40
Initial angle in degrees is: 12.972239886185005
Initial time is: 1.7958315233127196
Final angle in degrees = 20.289510667404812
Time = 2.512746696644583

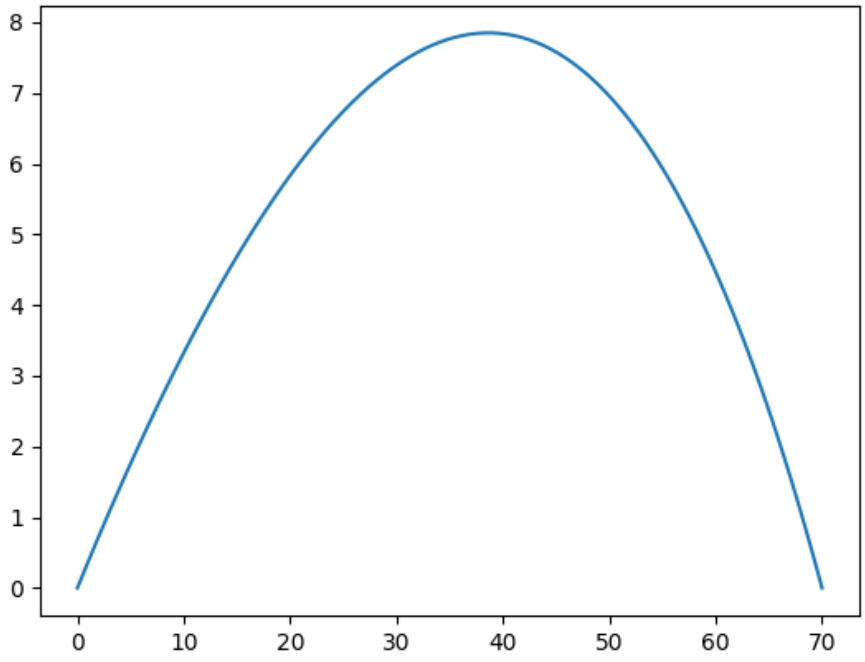

Figure 3: length : 70, velocity : 40, Optimize

따라서 위에서 구한 32.18도 보다 더 정확하게 70에 도달하는 발사각도(20.289도)를 찾을 수 있다.

4 결론

위 프로젝트를 통해 오일러 방법을 사용하여 초기 추정값에서 시작하여 대기 마찰이 있는 포물선 운동을 시뮬레이션하고, 반복적으로 최적화하여 최종적으로 목표 지점까지의 발사 각도를 찾는 것을 수행했다. 아래는 이 과정에서 얻은 결론이다.

4.1 시뮬레이션 및 오일러 방법:

초기 추정값을 사용하여 오일러 방법을 통해 대기 마찰이 있는 포물선 운동을 시뮬레이션했다.
시뮬레이션에서는 대기 마찰이 물체의 운동에 어떤 영향을 미치는지 확인했다.
반복적으로 현재 위치에서 최적의 발사 각도를 조정하여 목표 지점에 가까이 다가가려고 노력했다.

4.2 최적화:

최종적으로는 SciPy의 scipy.optimize.broyden1 함수를 사용하여 최적화를 수행했다.
초기 추정값에서 시작하여 최적의 발사 각도와 도달 시간을 찾았다.
이 최적화 과정을 통해 초기 추정값에서 목표 지점까지의 발사 각도를 더 정확하게 찾을 수 있었다.

4.3 결론 및 보완점:

코드를 통해 초기 조건에서 물체의 포물선 운동을 고려한 시뮬레이션과 최적화를 통한 발사 각도 추정이 가능했다.
대기 마찰을 고려하면서 초기 추정값보다 정확한 발사 각도를 찾을 수 있었으며, 이는 최적화 과정을 통해 높은 정확도로 수렴되었다.
이와 같은 접근 방식을 통해 물체의 운동을 모델링하고 최적화를 수행함으로써 목표 지점에 대한 발사 각도를 비교적 정확하게 찾을 수 있었다.
이번에 시행한 2D space 뿐만 아니라 3D space에서도 적용해보는것을 기대해 볼 수 있다. (스칼라함수 대신 벡터함수 사용)
포탄의 양력과 모멘트, 전향력까지 고려한 조금 더 사실적인 방정식을 세운다면 더 자세한 해석이 가능했을 것이다.

5 Github Code link

https://github.com/Taenyoung-Lee/Numerical-Analysis/tree/main

References

「Applied Numerical Methods Using MATLAB, 2nd edition」(2005), John Wiley Sons, Inc.
「총과 탄도학」(1998. 1. 10), 청문각