전산천문학 HW4

서유경, 2013-12239
제출일 : 17.05.05(또 늦었습니다..ㅠㅠ)

1.

이 문제에서 n차원의 구체의 부피를 구하는 방법은 점의 개수를 새면 된다. n차원이면 수직으로 교차하는 축이 n개 필요하므로, 이 축들을 x,y,z,v,w로 두고 각각의 축에 대한 좌표 랜덤값을 10^7개씩 생성한다. 즉, x[i]는 i번째 점의 x좌표 값이고, v[j]는 j번째 점의 v좌표 값이다. 따라서 5차원에서 i번째 점의 좌표는 (i=0~10^7-1), (x[i],y[i],z[i],v[i],w[i])가 된다. 이때 np.rand 함수를 쓰면 0~1 사이 값밖에 생성되지 않으므로, 원점을 중심으로 반지름이 1인 구를 형성하려면 2*np.rand() -1 로 조정해 줘야 한다.

그리고 x,y,z,v,w 각각의 좌표값의 제곱합이 1 이하가 되는(=원점으로부터 거리가 1 이하) 점들의 개수를 세고 전체 점의 개수 10^7으로 나누어 주면, 구에 접하는 정육면체 부피에 대한 구의 부피비가 나온다. 실제 구의 부피는 여기에, 정육면체 부피인 2^d 값(d는 차원수)을 곱하면 반지름이 1인 구의 부피가 나오게 된다.

<코드>

```
#HW4-1
from numpy import *
import numpy as np
import matplotlib.pyplot as plt

n=10**7

#d=3
d3=np.zeros(100)
for i in range(100):
    x=2*np.random.rand(n)-1
    y=2*np.random.rand(n)-1
    z=2*np.random.rand(n)-1
    count=0.
    for j in range(n):
        if (x[j]**2+y[j]**2+z[j]**2)<=1. :
            count+=1.
    d3[i]=count/n


md3=np.average(d3)*8
std3=np.std(d3*8)
```

```
#d=4
d4=np.zeros(100)
for i in range(100):
    x=2*np.random.rand(n)-1
    y=2*np.random.rand(n)-1
    z=2*np.random.rand(n)-1
    v=2*np.random.rand(n)-1
    count=0.
    for j in range(n):
        if (x[j]**2+y[j]**2+z[j]**2+v[j]**2)<=1. :
            count+=1.
    d4[i]=count/n

md4=np.average(d4*(16.))
std4=np.std(d4*(16.))

#d=5
d5=np.zeros(100)
for i in range(100):
    x=2*np.random.rand(n)-1
    y=2*np.random.rand(n)-1
    z=2*np.random.rand(n)-1
    v=2*np.random.rand(n)-1
    w=2*np.random.rand(n)-1
    count=0.
    for j in range(n):
        if (x[j]**2+y[j]**2+z[j]**2+v[j]**2+w[j]**2)<=1. :
            count+=1.
    d5[i]=count/n

md5=np.average(d5*(32.))
std5=np.std(d5*(32.))
```

<결과>
**감마함수의 값은 울프람 알파를 통해 계산하였습니다.

#d=3일 때

참값 : #gamma func(5/2)=sqrt(pi)*3/4 이므로, V3= $\frac{4\pi}{3}$ 4.1887902047863905

계산 평균값 : 4.1889158959999993
계산 표준편차 : 0.0011562630934108314

#d=4 일 때

참값 : gamma func(3)=2 이므로, V4= $\frac{\pi^2}{2}$ =4.934802200544679

계산 평균값 : 4.9346732800000002
계산 표준편차 : 0.0026489203476133504

#d=5 일 때

참값 : gamma func(7/2)=sqrt(pi)*15/8 V5= $\frac{8\pi^2}{15}$ =5.263789013914324

계산 평균값 : 5.2637839040000003
계산 표준편차 : 0.0040264750070979592

몬테-카를로 method를 통해 계산한 값의 유효숫자 세 자리수 까지는 n차원의 구의 부피의 참값과 일치한다. 그리고 d가 증가 할수록 표준편차가 증가한다. dimension이 증가할수록 구의 부피가 증가하는데, 이 증가 비율보다 표준편차의 증가비율이 더 크다.

2번.

(a) 코드

```
#HW4-2
from numpy import *
import numpy as np
import matplotlib.pyplot as plt

tmax=[0.01,0.1,1,10]
zmax=1.
#귀찮아서 아예 세로 1000개, 가로 4개의 np.array를 만들어 버렸습니다
n=1000
xn=np.zeros((n,4))
yn=np.zeros((n,4))
zn=np.zeros((n,4))



#initial position=>
for j in range(4):
    for i in range(n):
        xi1=np.random.rand()  #그리스문자를 입력할 수 없으니 영어 표기인 xi로 대체
        xi2=np.random.rand()
        u=sqrt(xi1)
#맨 처음 photon의 direction은 z+방향으로 방출되는것을 반영, 그리고 u=cos(θ)입니다.
        si=sqrt(1-u**2)
#cos^2+sin^2=1임을 이용. 어차피 θ,0~π범위에서 sinθ값은 항상 0이상
        p=2*pi*xi2
        xi3=np.random.rand()
        ta=-log(xi3)
        L=ta/tmax[j]
        #어차피 zmax=1이니, tau에 taumax를 나눠 준 값이 이동가능거리 L이 됩니다
        x=L*si*cos(p)
        y=L*si*sin(p)
        z=L*u
        while (z<=1 and z>=0):    #이제 재흡수되거나, zmax=1을 벗어날 때까지 반복
            xi4=np.random.rand()
            xi5=np.random.rand()
            p=2*pi*xi4
            u=2*xi5-1
            si=sqrt(1-u**2)
            xi3=np.random.rand()
```

```
            ta=-log(xi3)
            L=ta/tmax[j]
            x=x+L*si*cos(p)
            y=y+L*si*sin(p)
            z=z+L*u
        xn[i,j]=x
        yn[i,j]=y
        zn[i,j]=z
```

```
#4-(a) #1000개의 photon으로 통계를 낸 결과
count_tra=np.zeros(4)  #방출되는 광자 수
count_ref=np.zeros(4)  #안으로 되돌아가는 광자 수
for j in range(4):
    for i in range(n):
        if(1<=zn[i,j]):
            count_tra[j]+=1.
        else:
            count_ref[j]+=1.
```

(a) 1000개 광자의 운명을 계산한 결과

```
In [652]: count_tra
Out[652]: array([ 994.,   925.,   544.,   121.])

In [653]: count_ref
Out[653]: array([   6.,    75.,   456.,   879.])
```

τ_{max}	0.01	0.1	1	10
Transmission	0.994	0.925	0.554	0.121
reflection	0.006	0.075	0.456	0.879

->이를 10000개로 늘려 보았고 결과는 다음과 같았다.

```
In [671]: count_tra
Out[671]: array([ 9895.,  9125.,  5520.,  1194.])

In [672]: count_ref
Out[672]: array([  105.,   875.,  4480.,  8806.])
```

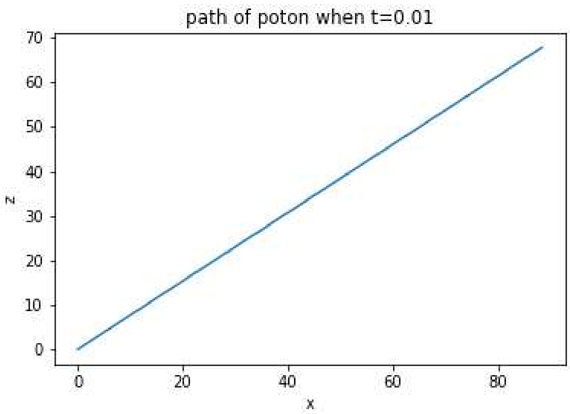
τ_{max}	0.01	0.1	1	10
Transmission	0.9895	0.9125	0.552	0.1194
reflection	0.0105	0.0875	0.448	0.8806

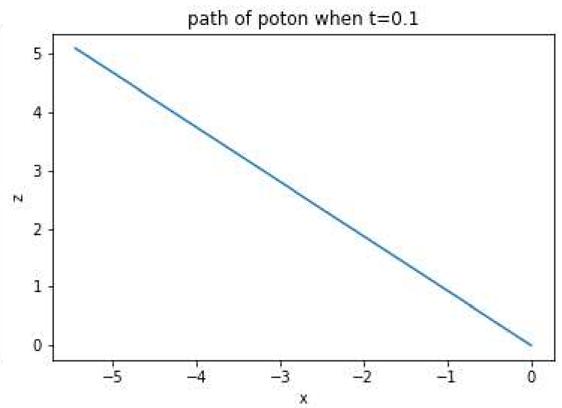
소광계수인 tau가 작을 수록 transmission이 잘 되며, 클수록 reflection이 많이 일어난다. 이 이유는, 우선 tmax가 작기 때문에, 한 번에 이동할 수 있는 이동거리 L의 크기가 커지게 되기 때문이다. 무엇보다 emit되는 photon의 첫 이동방향의 z방향값은, 무조건 양수(상향)이

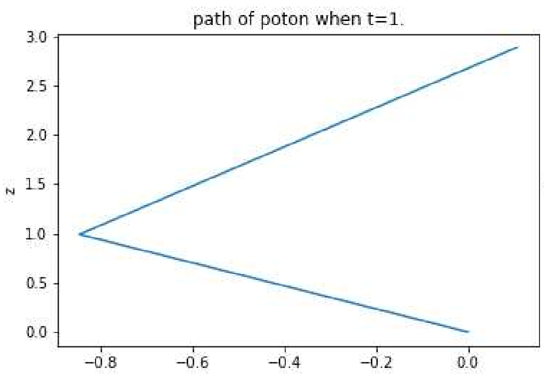
므로, tmax가 작을 수록 첫 이동때 바로 zmax=1을 넘어버리게 되는 경우가 발생한다. 때문에
반면 tmax가 클수록 매질 내에서 한번에 이동 할 수 있는 거리가 작아지기 때문에, 초기 이동방향이 +z방향이라고 하더라도 zmax=1을 넘을 수가 없다. 따라서 매질 중간 까지 갔다가 다시 되돌아와서 reflection 될 확률이 커지는 것이다.

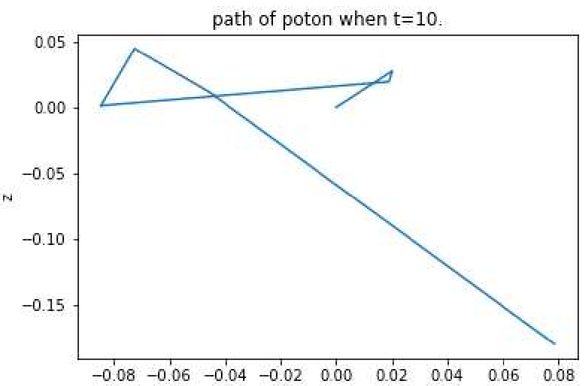
(b) 광자 이동경로 그리기

->1000개의 이동경로를 전부 그릴 수는 없으니 각각의 taumax 경우에 대해 예시를 한가지씩 들었습니다. 그리고 최종 광자의 위치 분포 그래프도 첨부합니다. (위치를 기록한 코드는 너무 길어서 문서에 삽입하지 않고, 따로 파일로 보내드리겠습니다..)

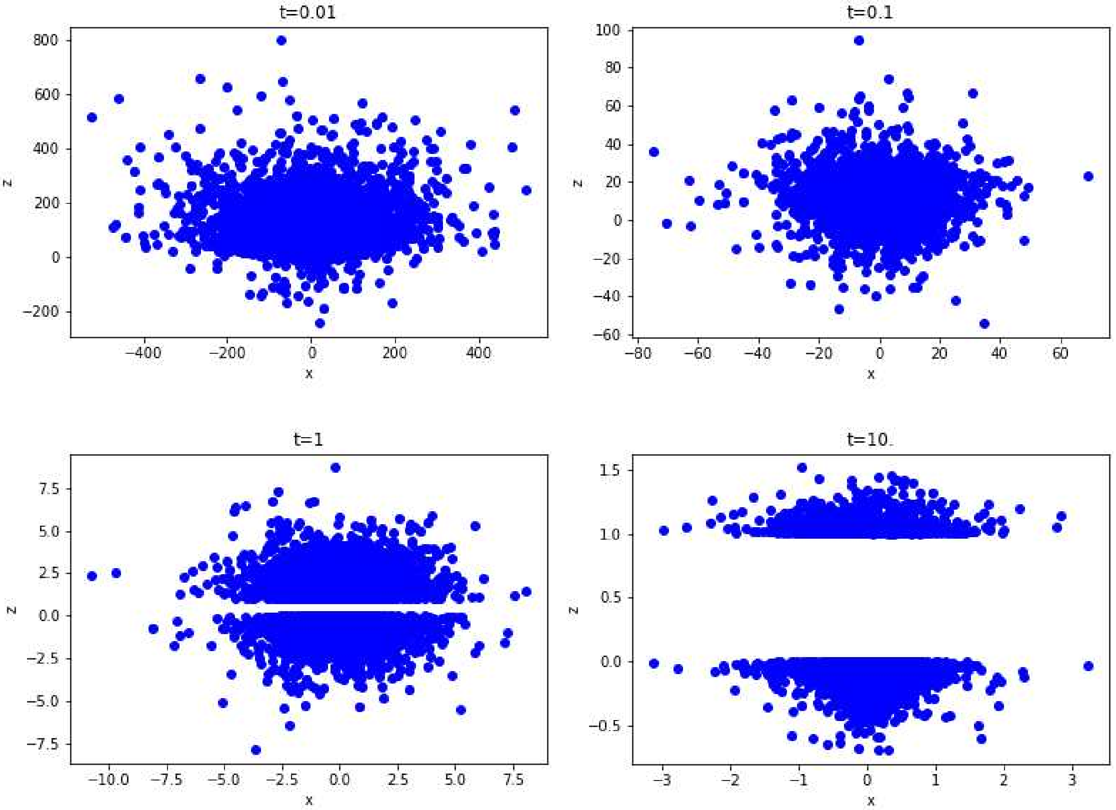








tmax=0.01,0.1의 경우 z=0에서 emit 되자마자 바로 zmax=1을 넘어 transmit되었으며, t=1의 경우 두 단계를 거쳐서 transmit되었고, t=10의 경우 수차례 꺾이고 난 후에 reflection 되었다.



<photon들의 최종 도착지 위치 분포>

3.

(a)

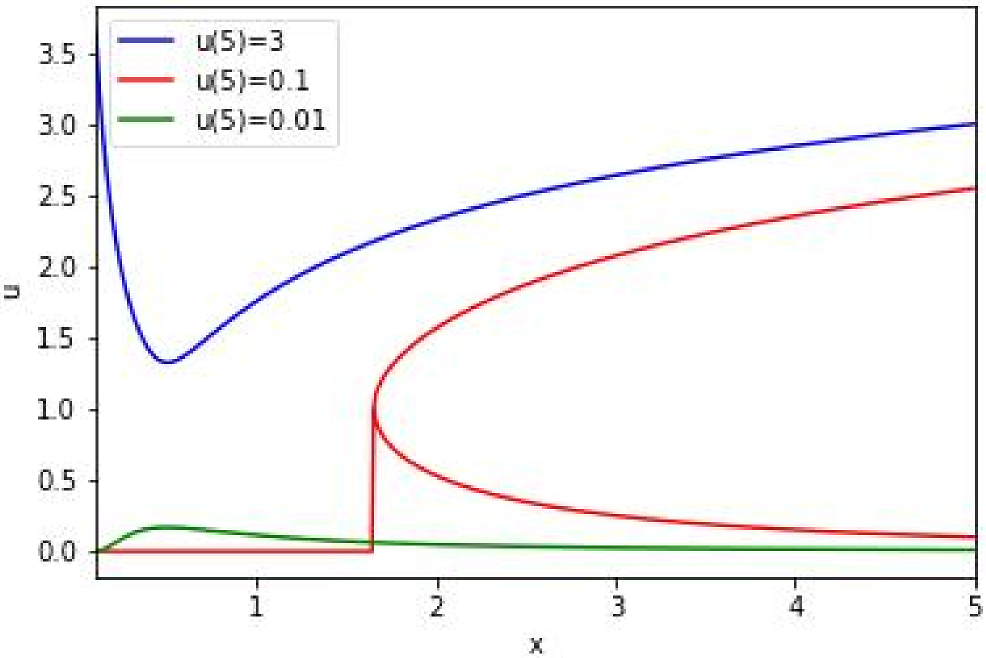
```
from numpy import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint


def dudx(u,x):
    return (2/x-1/x**2)/(u-1/u)

#x=5 to 0.1
#u=3,0.1,0.01 at x=5


x1=np.linspace(5,0.1,491)

y1=odeint(dudx,3,x1)
y2=odeint(dudx,0.1,x1)
y3=odeint(dudx,0.01,x1)


x2=np.arange(1.65,5,0.01)
y2p=odeint(dudx,1.000001,x2)

plt.plot(x1,y1[:,0],'b-',label='u(5)=3'),plt.plot(x1,y2[:,0],'r-',label='u(5)=0.1'),plt.plot(x2,y2p,'r-'),plt.plot(x1,y3[:,0],'g-',label='u(5)=0.01'),plt.xlabel('x'),plt.ylabel('u'),plt.legend(loc=2),plt.xlim(0.1,5),plt.savefig('HW4-3(a).png')
```

u(5)=0.1을 초기값으로 잡으면 x=5에서 0.1로 반경이 감소할수록, 속도가 증가하는데, u=1에 도달하고 나서 u값이 0으로 뚝 떨어진다. bondi-accreation식(3)의 변수들을 전부 우변으로 옮기면, u=1에서 (u-1/u)=0이 되기 때문에, 분모가 무한대가 되는 것이다.

$$\frac{du}{dx}=\frac{\frac{2}{x}-\frac{1}{x^2}}{u-\frac{1}{u}}$$

때문에 u=1에서 해를 구할 수가 없는거 처럼 보이는데, 컴퓨터 프로그램말고 실제로 함수 해를 구하면 u=1인 지점(대략 x=1.65인 지점)에서 접선의 기울기가 무한대(수직)이 된다. 즉, 실제 u의 개형은 가로로 눕힌 포물선 같은 형태로, 어떤 x값에 대해 u값이 2개 존재하며

즉, 함수가 u(x)가 아니라 x(u)의 형태로 존재하고, u=1미만의 지점에 대해서는 x값이 존재하지 않는 형태가 되어 버리는 것이다.



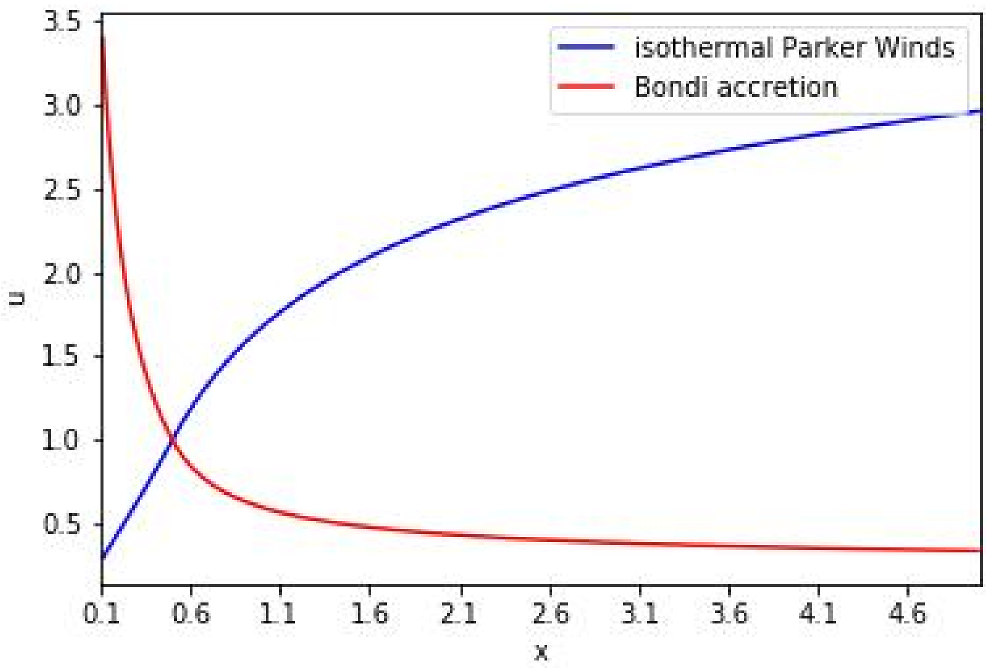
원래 u(5)=0.1을 초기값으로 주고 시작하면 저 붉은색의 포물선에서 아래쪽 부분만 나오지만, 꼭지점의 근방의 좌표라고 할 수 있는(일일이 점들의 좌표값을 확인하고 찾았습니다) u(1.65)=1.000001이라는 값을 주고 다시 odeint를 실행 시 포물선 위쪽의 그래프가 완성이 된다.

(b) - 포기..

Transonic solution이 있다는것을 보이란 얘기는, x=0.5에서 u=1을 지나가는 해 곡선이 있단 얘기이고, (0.5,1)근방에서는 $\Delta u = \pm 2\Delta x$ 즉, $\frac{du}{dx} \approx$ +-2가 됨을 보이라는 것이다. . 곡 기울기가 -2로, x가 커질수록 속도u가 감소하는 바람은 Bondi Accretion이고, 기울기가 +2fh x가 커질수록 속도u도 커지는 바람은 isothermal Parker winds가 된다는 의미이다.

(c)

```
t1=np.arange(0.5,5,0.01)
t2=np.arange(0.5,0.1,-0.01)
u1=odeint(dudx,1.000001,t1)
u2=odeint(dudx,1.000001,t2)

plt.plot(t1,u1,'b-',label='isothermal Parker Winds'),plt.plot(t2,1/u2,'b-'),plt.plot(t1,1/u1,'r-'),plt.plot(t2,u2,'r-',label='Bondi accretion'),plt.legend(loc=1),plt.xlim(0.1,5),plt.xticks(np.arange(0.1,5.1,0.5)),plt.xlabel('x'),plt.ylabel('u'),plt.savefig('HW4-3(c).png')
```



4번.

(a) - 도저히 파이썬으로 풀 수가 없어서 직접 손으로 전개해서 풀었습니다.

$$\Theta = a_0 + a_1\xi + a_2\xi^2 + a_3\xi^3 + a_4\xi^4 ....$$

우리가 구하고자 하는것은, 상수항, ξ, ξ^2, ξ^3, ξ^4 의 계수인 a0,a1,a2,a3,a4이다.
우선 초기조건에서 $\xi = 0$일때, Θ의 값이 1이라는 것과, 도함수 값이 0이라는 것을 통해 a0=1, a1=0 임을 알아내었다. 그 뒤부터는 미분 방정식을 직접 전개하였는데, 좌변과 우변을 정리해서 이게 미분방정식의 일반형으로 만들면

$$\frac{d}{d\xi}(\xi^2 \frac{d\Theta}{d\xi}) = -\xi^2\Theta^n$$

$$2\xi\Theta' + \xi^2\Theta'' = -\xi^2\Theta^n$$

세타에, 급수형태를 대입하고 양변의 같은 차수항의 계수를 비교해 나가면서 a2,a3,a4 를 구한다.

$$2a_1\xi + 6a_2\xi^2 + 12a_3\xi^3 + 20a_4\xi^4 ..... = \xi^2 * (a_0 + a_1\xi + a_2\xi^2 + a_3\xi^3 + a_4\xi^4 ...)^n$$

<u>a0=1, a1=0</u>

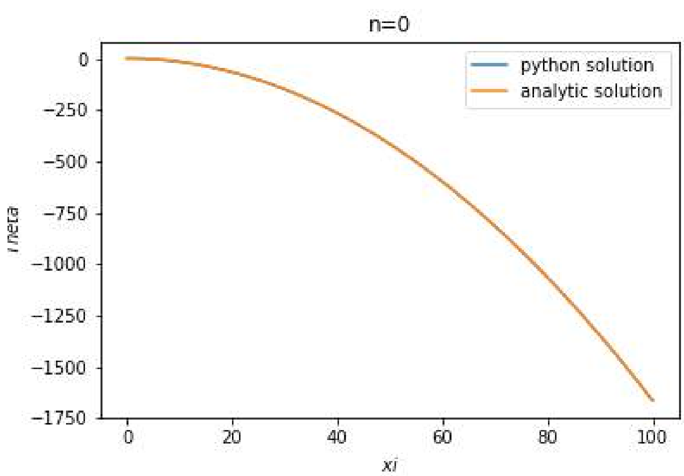
ξ^2: 우변의 계수는 a0^n=1이고, 좌변의 계수는 6이므로 <u>a2=1/6</u>

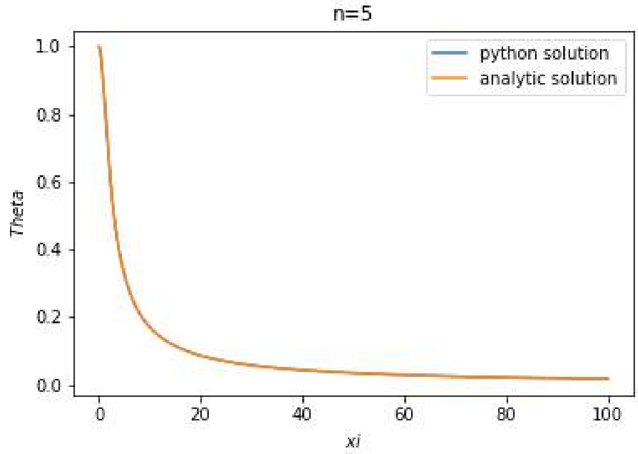
ξ^3 : 우변의 계수는 $a_1 * a_0^{n-1} * n$이고 좌변의 계수는 12a3지만, a1=0이므로, <u>a3=0</u>

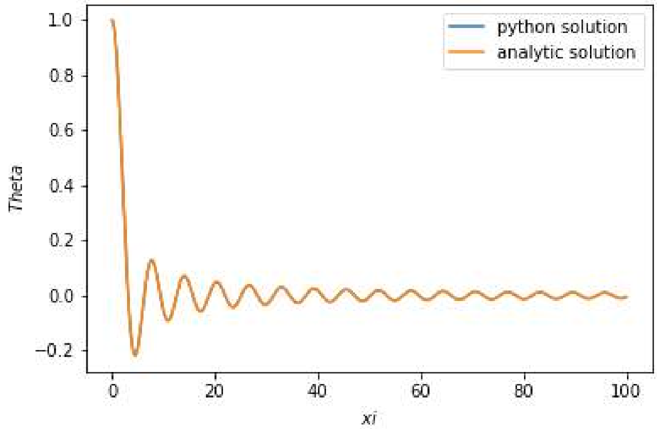
ξ^4 : a1,a3가 0이므로 우변의 계수는 $a_0 * a_2^{n-1} * n$ 이고, 좌변의 계수는 20a4. a0=1, a2=1/6이므로, <u>a4=$\frac{n}{20}(\frac{1}{6})^{n-1}$</u> 이다.

(b).

```
from numpy import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint


n=input(" ")

def deriv(y,x):
    dydt1=y[1]
    dydt2=-2/x*y[1]-(y[0])**n
    return [dydt1,dydt2]



xi0=[1.0,0.0]
t=np.arange(0.000000000001,100,0.01)
y=odeint(deriv,xi0,t)

theta1=y[:,0]

plt.plot(t,theta1,label='python solution'),plt.xlabel('$xi$'),plt.ylabel('$Theta$'),plt.legend(loc=1),plt.savefig('HW4-4(b)-n.png')
plt.plot(t,(1-t**2/6),label='analytic solution,n=0')
plt.plot(t,(np.sin(t)/t),label='analytic solution,n=1')
plt.plot(t,(1+t**2/3)**(-0.5),label='analytic solution,n=5'),plt.title('n=5')
```







python으로 ODEINT 명령어로 구한 그래프하고, 실제 미분방정식을 푼 결과의 그래프하고 잘 들어 맞습니다.

(c) ->
#n=0, 0.5, 1.0, 1.5, 2, 2.5, 3, 3.5, 4.0, 4.5, 4.9
*** n=0.5,1.5,2.5,3.5,4.5 일 때 해를 구할 수가 없고, 함수 자체의 식을 아는게 아닐때 해를 어떻게 구해야 할지 감이 안와.. 포기하였습니다...