전산천문학 Hw6

2013-12239 서유경
제출일 :17.06.07

1번. #HW6-1. find minimum

```
import numpy as np
from numpy import *
from scipy import linalg
import scipy.optimize as opt
import matplotlib.pyplot as plt
import math
from scipy.optimize import minimize


def f(t):   <=함수지정
    x,y=t[0],t[1]
    return 2.*x**2+y**2-2.*x*y+abs(x-3.)+abs(y-2.)

def gradf(t):   <=그래디언트 지정
    x,y=t[0],t[1]
    gx=4.*x-2.*y+abs(x-3.)/(x-3.)
    gy=2*y-2*x+abs(y-2.)/(y-2.)
    return np.array([gx,gy])

tol=1.e-6 <=Tolerance 지정

x=np.arange(-2,3.01,0.01)
y=np.arange(-2,2.01,0.01)
X,Y=np.meshgrid(x,y)
E=np.log10(f(np.meshgrid(x,y)))
lev1=np.linspace(np.min(E),np.max(E),20)     <= meshgrid 미리 지정

#%%
```

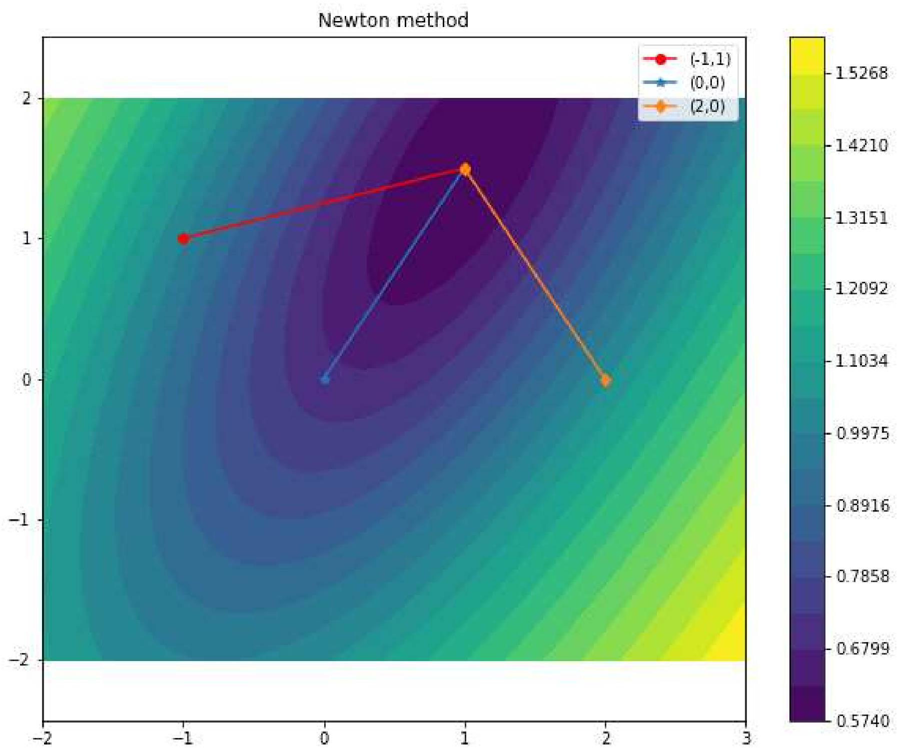
#(1) Newton's method

```
def s(t):   <=수동으로 Hessian 구해서, s(t) 구했습니다.
    s0=-2*gradf(t)[1]-2*gradf(t)[0]
    s1=-2*gradf(t)[0]-4*gradf(t)[1]
    return np.array([s0/4.,s1/4.])
x1=np.array([-1,1.])   <=시작점 (-1,1)
```

```
px1,py1=[x1[0]],[x1[1]]
xn=x1+s(x1)
px1.append(xn[0])
py1.append(xn[1])
while(sqrt(sum((xn-x1)**2))>tol):
    x1=xn
    xn=x1+s(x1)
    px1.append(xn[0])
    py1.append(xn[1])

x2=np.array([0.,0.])
px2,py2=[x2[0]],[x2[1]]
xn=x2+s(x2)
px2.append(xn[0])
py2.append(xn[1])
while(sqrt(sum((xn-x2)**2))>tol):
    x2=xn
    xn=x2+s(x2)
    px2.append(xn[0])
    py2.append(xn[1])

x3=np.array([2.0,0.0])
px3,py3=[x3[0]],[x3[1]]
xn=x3+s(x3)
px3.append(xn[0])
py3.append(xn[1])
while(sqrt(sum((xn-x3)**2))>tol):
    x3=xn
    xn=x3+s(x3)
    px3.append(xn[0])
    py3.append(xn[1])
```

<u>세 점 모두 #minimum point (x,y)=(1.0,1.5) 로 나왔습니다., xt,yt의 length를 세어본 결과 iteration은 시작점으로부터 1단계 만에 찾았습니다.</u>

```
plt.figure(figsize=(10,8)),plt.contourf(X,Y,E,lev1),plt.colorbar(),plt.axis('equal'),plt.title('Newton method'),\
            plt.plot(px1,py1,'ro-',label='(-1,1)'),\
            plt.plot(px2,py2,'*-',label='(0,0)'),\
            plt.plot(px3,py3,'d-',label='(2,0)'),plt.legend(),plt.savefig('hw6-1(a).png')
```



#(2) Steepest test

def Golden2d(x,direct,Tol): <=Simplest method를 해주는 함수지정, tolerance를 x좌표 차이보단 그냥 거리차이로 두었습니다.

```
    R=(sqrt(5.)-1.)/2
    b=x+R*direct*5
    a=x-R*direct*5
```

<=황금비의 5배만큼 앞 뒤로 준것은 eigenvalue를 모르기 때문이기도 하고, 1배로 주니 너무 시간이 길어지고, 그 이상으로 주면 경로가 이상해져버려서 이렇게 되었습니다.

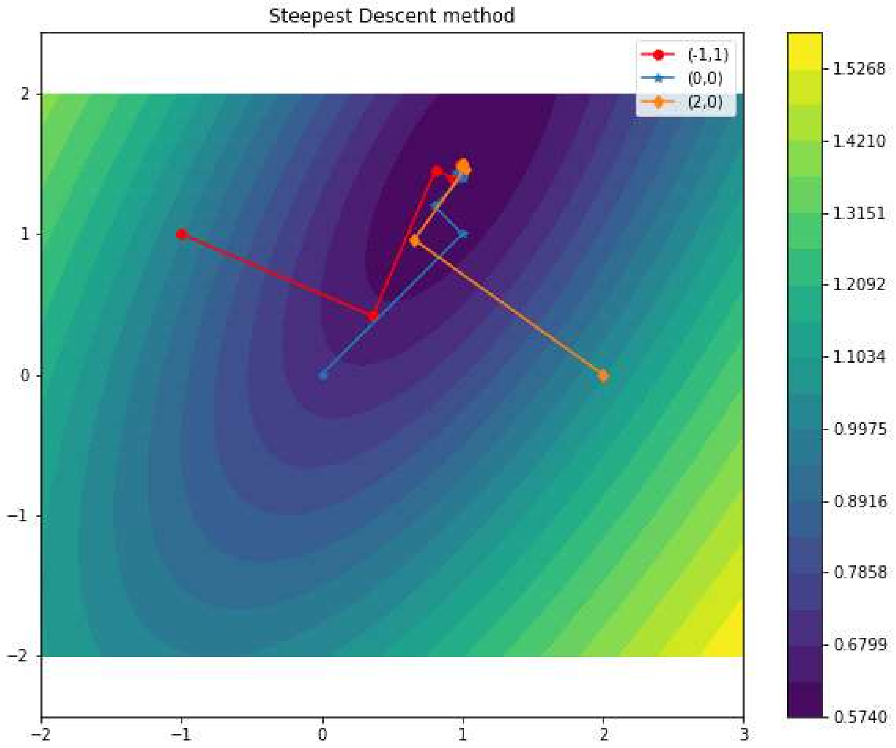
```
    while (sqrt(sum((a-b)**2))>Tol):
        x1=b-R*(b-a)
        x2=a+R*(b-a)
        f1=f(x1)
        f2=f(x2)
        if(f2>f1): b=x2
        else : a=x1
    if(f(a)>f(b)): xmin=b
    else: xmin=a
    return xmin
```

```
x1=np.array([-1,1.])
px1,py1=[x1[0]],[x1[1]]
def cbF1(x):  <=각 시작점마다 경로위치를 append해주는 함수를 일일이 지정합니다.
    global px1,py1
    px1.append(x[0])
    py1.append(x[1])

n1=gradf(x1)/sqrt(sum(gradf(x1)**2))   <=gradient 단위백터 설정
nh1=np.array([-n1[1],n1[0]]) <=gradient 단위백터에 수직인 단위백터 설정
xn=Golden2d(x1,n1,tol)
cbF1(xn)
xn_1=Golden2d(xn,nh1,tol)
cbF1(xn_1)
c1=2.
while (sqrt(sum((xn-xn_1)**2))>tol):
        if(c1%2==0 ):
            xn=Golden2d(xn_1,n1,tol)
            cbF1(xn)
            c1+=1     <=번갈아가면서 gradient방향으로 갔다가, 그에 수직인 방향으로 갔다
가 할 수 있게 짝수번째는 gradient방향으로 진행, 홀수번째는 수직방향으로 진행을 택하였습
니다.
        else:
            xn_1=Golden2d(xn,nh1,tol)
            cbF1(xn_1)
            c1+=1



x2=np.array([0.,0.])
px2,py2=[x2[0]],[x2[1]]
def cbF2(x):                <=경로저장함수 지정
    global px2,py2
    px2.append(x[0])
    py2.append(x[1])

n2=gradf(x2)/sqrt(sum(gradf(x2)**2)) <=gradient 단위백터 설정
nh2=np.array([-n2[1],n2[0]]) <=gradient 단위백터에 수직인 단위백터 설정
xn=Golden2d(x2,n2,tol)
cbF2(xn)
xn_1=Golden2d(xn,nh2,tol)
```

```
cbF2(xn_1)
c2=2.
while (sqrt(sum((xn-xn_1)**2))>tol):
        if(c2%2==0 ):
            xn=Golden2d(xn_1,n2,tol)
            cbF2(xn)
            c2+=1
        else:
            xn_1=Golden2d(xn,nh2,tol)
            cbF2(xn_1)
            c2+=1

x3=np.array([2.0,0.0])
px3,py3=[x3[0]],[x3[1]]
def cbF3(x):            <=경로저장함수 설정
    global px3,py3
    px3.append(x[0])
    py3.append(x[1])

n3=gradf(x3)/sqrt(sum(gradf(x3)**2)) <=gradient 단위백터 설정
nh3=np.array([-n3[1],n3[0]])     <=gradient 단위백터에 수직인 단위백터 설정
xn=Golden2d(x3,n3,tol)
cbF3(xn)
xn_1=Golden2d(xn,nh3,tol)
cbF3(xn_1)
c3=2.
while (sqrt(sum((xn-xn_1)**2))>tol):
        if(c3%2==0 ):
            xn=Golden2d(xn_1,n3,tol)
            cbF3(xn)
            c3+=1
        else:
            xn_1=Golden2d(xn,nh3,tol)
            cbF3(xn_1)
            c3+=1

plt.figure(figsize=(10,8)),plt.contourf(X,Y,E,lev1),plt.colorbar(),plt.axis('equal'),plt.title('S
teepest Descent method'),\
            plt.plot(px1,py1,'ro-',label='(-1,1)'),\
            plt.plot(px2,py2,'*-',label='(0,0)'),\
            plt.plot(px3,py3,'d-',label='(2,0)'),plt.legend(),plt.savefig('hw6-1(b).png')
```

(그래프는 맨 뒤에)

#iteration (-1,1)은 14번, (0,0)은 c2=18, (2,0)은 c3=9번 계산이 진행
최소점 (x,y)=(1.0,1.5)



#(3) Powell method

```
x1=np.array([-1,1.])
px1,py1=[x1[0]],[x1[1]]
def cbF1(x):    <=경로저장함수 까지 (2)와 같음
    global px1,py1
    px1.append(x[0])
    py1.append(x[1])

res1=minimize(f,x1,method='Powell',options={'xtol':tol, 'disp':True},callback=cbF1)



x2=np.array([0.,0.])
px2,py2=[x2[0]],[x2[1]]
def cbF2(x):                <=경로저장함수 까지 (2)와 같음
```

```
    global px2,py2
    px2.append(x[0])
    py2.append(x[1])

res2=minimize(f,x2,method='Powell',options={'xtol':tol, 'disp':True},callback=cbF2)


x3=np.array([2.0,0.0])
px3,py3=[x3[0]],[x3[1]]
def cbF3(x):                <=경로저장함수 까지 (2)와 같음
    global px3,py3
    px3.append(x[0])
    py3.append(x[1])
res3=minimize(f,x1,method='Powell',options={'xtol':tol, 'disp':True},callback=cbF3)
```

<Powell 결과>

#iteration : 모두 4times <-minimize 함수 기능에서 나온 res에 iteration이 count되므로 그걸 그대로 가져다 썼습니다. callbackF함수를 통한 저장된 경로의 length도 iteration+1로 (처음 위치의 점도 포함되기 때문에 개수가 +1) iteration 결과와 일치하였습니다.

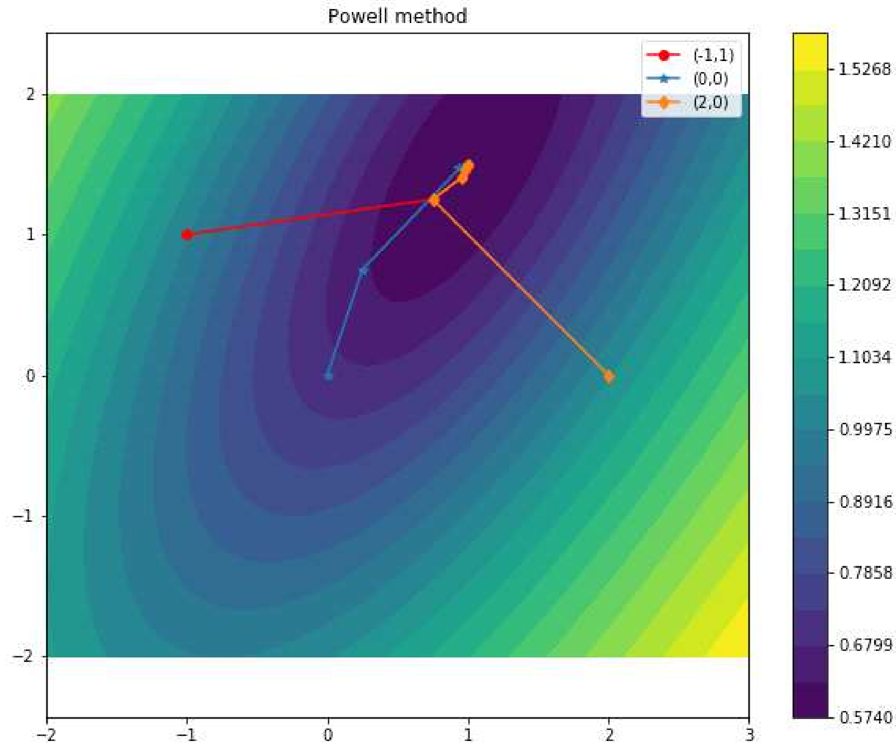
<각 시작점에 따른 찾은 최소점 결과>

#px1[4],py1[4] =(1.0000000022371405, 1.5000000024069642)

#px3[4],py2[4] = (1.0000000022371405, 1.5000000011088985)

#px3[4],py3[4] =(1.0000000022371405, 1.5000000024069642)

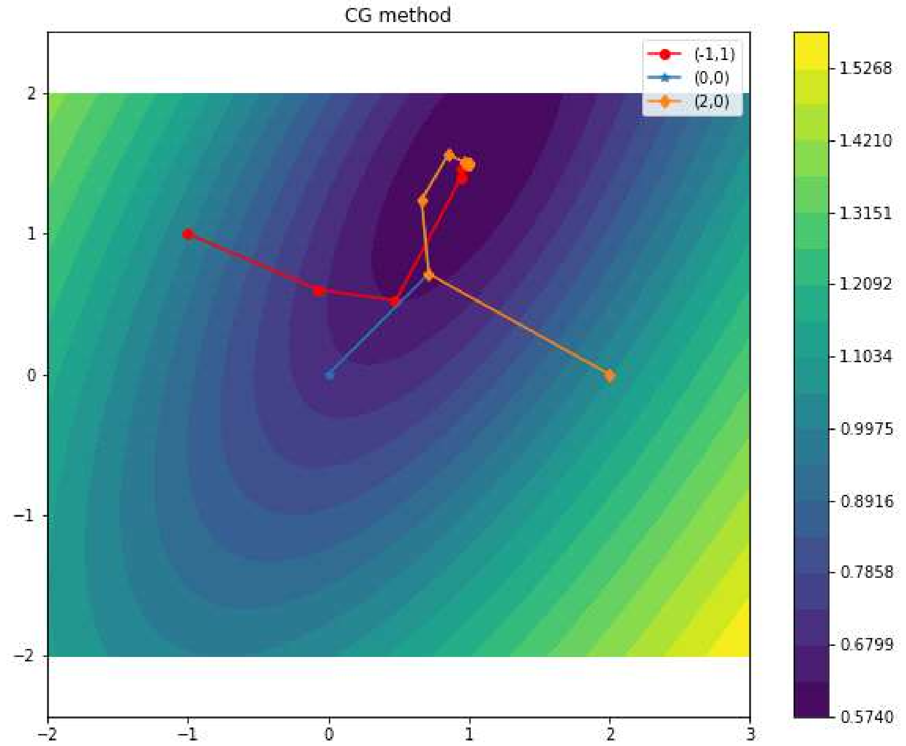
```
plt.figure(figsize=(10,8)),plt.contourf(X,Y,E,lev1),plt.colorbar(),plt.axis('equal'),plt.title('Powell method'),\
            plt.plot(px1,py1,'ro-',label='(-1,1)'),\
            plt.plot(px2,py2,'*-',label='(0,0)'),\
            plt.plot(px3,py3,'d-',label='(2,0)'),plt.legend(),plt.savefig('hw6-1(c).png')
```



#%%
#(4) conjugate method

```
x1=np.array([-1,1.])
px1,py1=[x1[0]],[x1[1]]
def cbF1(x):
    global px1,py1
    px1.append(x[0])
    py1.append(x[1])
res1=minimize(f,x1,method='CG', options={'xtol':1.e-6,'disp':True},callback=cbF1)

x2=np.array([0.,0.])
px2,py2=[x2[0]],[x2[1]]
def cbF2(x):
    global px2,py2
    px2.append(x[0])
    py2.append(x[1])
res2=minimize(f,x2,method='CG', options={'xtol':tol,'disp':True},callback=cbF2)

x3=np.array([2.0,0.0])
px3,py3=[x3[0]],[x3[1]]
```

```
def cbF3(x):
    global px3,py3
    px3.append(x[0])
    py3.append(x[1])
res3=minimize(f,x2,method='CG', options={'xtol':tol,'disp':True},callback=cbF3)

plt.figure(figsize=(10,8)),plt.contourf(X,Y,E,lev1),plt.colorbar(),plt.axis('equal'),plt.title('C
G method'),\
            plt.plot(px1,py1,'ro-',label='(-1,1)'),\
            plt.plot(px2,py2,'*-',label='(0,0)'),\
            plt.plot(px3,py3,'d-',label='(2,0)'),plt.legend(),plt.savefig('hw6-1(d).png')

#x1: iteration 5번,array([ 0.99999994,  1.49999992]),/
 x2: iteration 12번,array([ 0.99999706,  1.49999332])/
 x3: iteration 12번, array([ 0.99999706,  1.49999332])
```



2번. 3차원 optimize

(scipy의 minimize 함수도 써보고, 직접 Powell과 CGmethod 스크립트도 짜보았습니다.)

```
import numpy as np
from numpy import *
from scipy import linalg
import scipy.optimize as opt
import matplotlib.pyplot as plt
import math
from scipy.optimize import minimize



def f(t):                        <=함수지정
    x,y,z=t[0],t[1],t[2]
    return 100*(y-x**2)**2+(1-x)**2+100*(z-y**2)**2+(1-z)**2

tol=1.e-6          <=tolerance 지정
x0=np.array([0.,2.,-1.])              <=2번문제는 시작점이 하나뿐이어서 편했습니다..
```

#(a) Powell

```
res1=minimize(f,x0,method='Powell',options={'xtol':tol, 'disp':True})
```

<u>결과 : #powell- iteration:23, 최소점 위치 : (1,1,1)</u>

```
#or

def Golden2d(x,direct,Tol):                    <=2차원 Golden2d함수 지정(1번과 동일)
    R=(sqrt(5.)-1.)/2
    b=x+R*direct*2                 <=마찬가지로 eigenvalue를 몰라서 이상한 경로로 되지 않으면서도 적당한 계산범위를 할 수 있는 2*R배를 앞뒤로 범위를 주었습니다.
    a=x-R*direct*2
    while (sqrt(sum((a-b)**2))>Tol):
        x1=b-R*(b-a)
        x2=a+R*(b-a)
        f1=f(x1)
        f2=f(x2)
        if(f2>f1): b=x2
        else : a=x1
    if(f(a)>f(b)): xmin=b
    else: xmin=a
    return xmin
```

```
dir1=np.array([1.,0.,0.])  <=초기 단위백터 지정
dir2=np.array([0.,1.,0.])
dir3=np.array([0.,0.,1.])

    x1=Golden2d(x0,dir1,tol)
    x2=Golden2d(x1,dir2,tol)
    x3=Golden2d(x2,dir3,tol)
    dir4=(x3-x0)/sqrt(sum(x3-x0)**2)
    x4=Golden2d(x3,dir4,tol)
    count=4.                    <=직접 코드를 짰을때의 count방식은 방향 한번 옮길때마다 1씩 추가해서 (3차원은 세방향이고, 시작점과 끝점 이어서 나온 최소점 까지 더하면) 총 4번 계산)

while (sqrt(sum((x4-x1)**2))>tol):         <=앞에도 나오지만 tolerance를 x좌표 차이보단 그냥 거리차이로 두었습니다.
    x1=Golden2d(x4,dir1,tol)
    x2=Golden2d(x1,dir2,tol)
    x3=Golden2d(x2,dir3,tol)
    dir4=(x3-x4)/sqrt(sum(x3-x4)**2)
    x4=Golden2d(x3,dir4,tol)
    count+=4
    dir1=dir2
    dir2=dir3
    dir3=dir4
```

<결과>

#iteration : 40 (즉 10세트 실시)

#최소점 x4=array([1.   , 1.00000011, 1.00000021])

#(b) CG method

```
x0=np.array([0.,2.,-1.])
res2=minimize(f,x0,method='CG', options={'xtol':tol,'disp':True})
```

minimize 함수 결과 : #cg: iteration:68, array([0.9999989 , 0.99999779, 0.99999559])

```
#or
def gradf(r):
    x=r[0]
    y=r[1]
```

```
    z=r[2]
    grx=-400*x*(y-x**2)-2*(1-x)
    gry=200*(y-x**2) +400*y*(y**2-z)
    grz=200*(z-y**2)+2*(z-1)
    return np.array([grx,gry,grz])


g0=-gradf(x0)
count=0.

 x1=Golden2d(x0,g0,tol)
 g1=-gradf(x1)+g0*sum((gradf(x1))**2)/sum(gradf(x0)**2)
 x2=Golden2d(x1,g1,tol)
 g2=-gradf(x2)+g1*sum((gradf(x2))**2)/sum(gradf(x1)**2)
 count+=2.
```

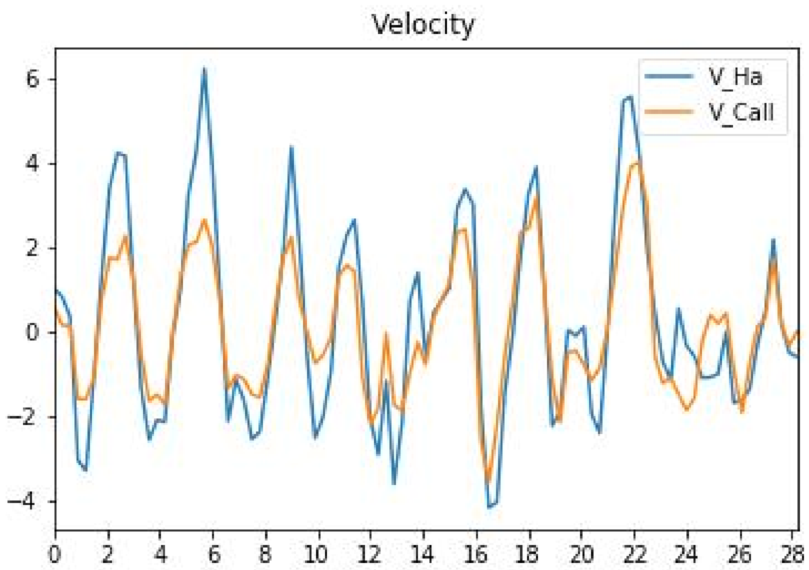
<=CGmethod count 방식도 gradient를 계속 변화시켜가면서 진행되므로 그냥 한번씩 방향바꿔서 minimize 시킬때마다 count+1씩 했습니다.

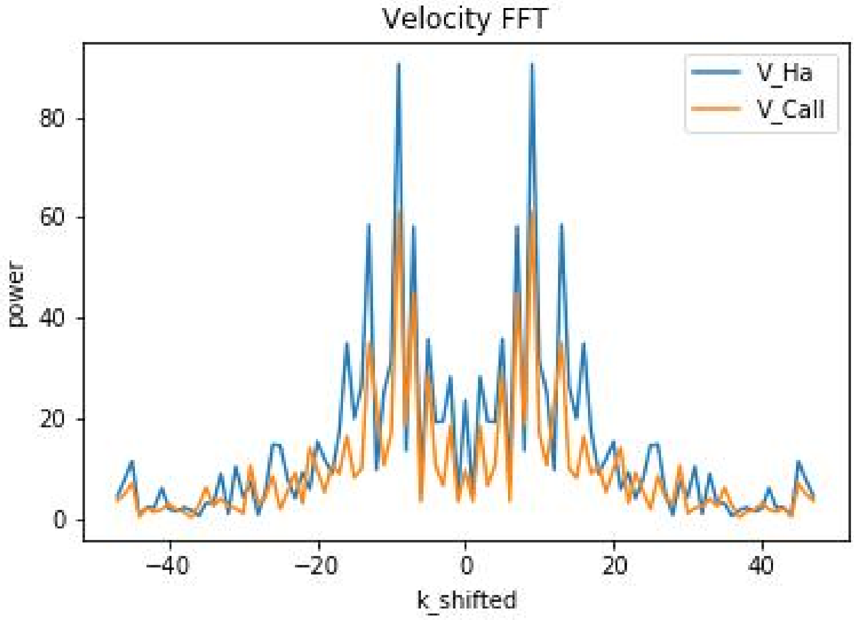
```
while(sqrt(sum((x2-x1)**2))>(tol)):
    x1=Golden2d(x2,g2,tol)
    g1=-gradf(x1)+g2*sum((gradf(x1))**2)/sum(gradf(x2)**2)
    x2=Golden2d(x1,g1,tol)
    g2=-gradf(x2)+g1*sum((gradf(x2))**2)/sum(gradf(x1)**2)
    count+=2.
```

#iteration=78(count된 숫자를 세었습니다.)
#x2:array([1.00002355, 1.00004486, 1.00008775])

3번. 시선속도 주기 찾기

```
import numpy as np
from numpy import *
import matplotlib.pyplot as plt

sol=np.loadtxt('sol_vel.dat')
t=sol[:,0]
vha=sol[:,1]
vca=sol[:,2]
n=len(vha)



v1=np.fft.fft(vha)
v2=np.fft.fft(vca)
freq=np.fft.fftfreq(n,1./n) <=freq구함
freq_s=np.fft.fftshift(freq) <=freq도 shift
v1_s=np.fft.fftshift(v1)  <=속력도 shift함
v2_s=np.fft.fftshift(v2)

plt.plot(t,vha,label='V_Ha'),plt.plot(t,vca,label='V_CaII'),plt.xticks(arange(0,28.2,2)),plt.xli
m(0,max(t)),plt.title('Velocity'),plt.legend(),plt.savefig('Hw6-3(a).png')

plt.plot(freq_s,abs(v1_s),label='V_Ha'),plt.xlabel('k_shifted'),plt.ylabel('power'),plt.title('V
e l o c i t y
FFT'),plt.plot(freq_s,abs(v2_s),label='V_CaII'),plt.legend(),plt.savefig('Hw6-3(b).png')
```





(b) 주기 찾기

max1 = np.argmax(abs(v1_s)) <=Maximum값을 가진게 몇번째 칸인지 찾는다
max2 = np.argmax(abs(v2_s))
kmax1 = abs(freq_s[max1])
kmax2 = abs(freq_s[max2])

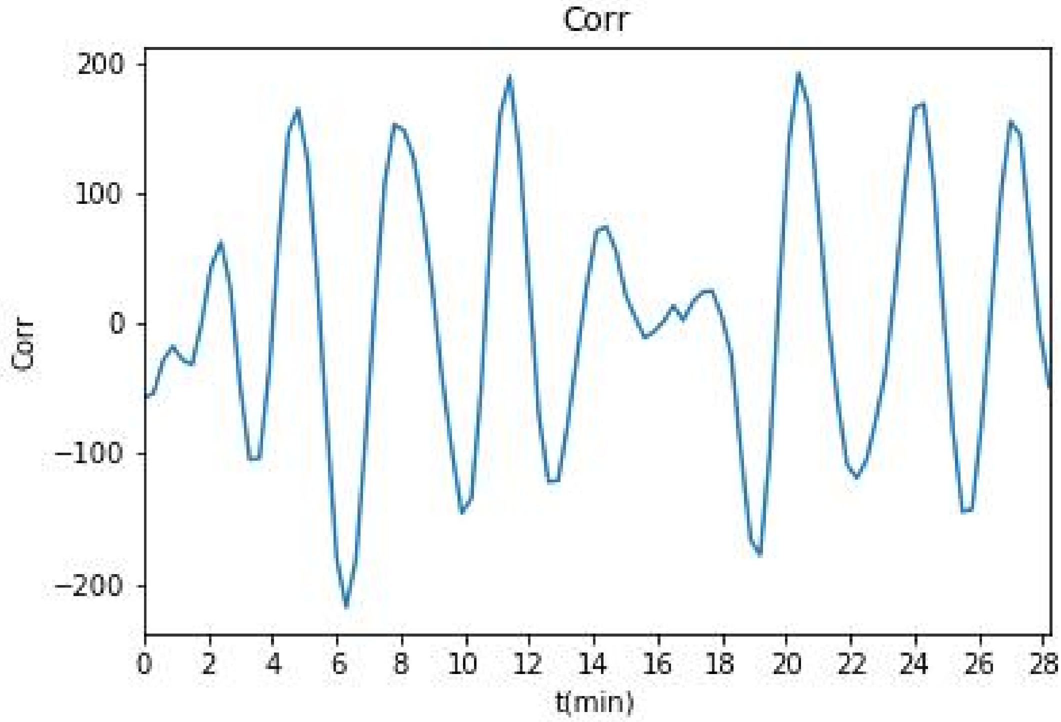
#kmax1=9.0
#kmax2=9.0

dt = t[n-1]-t[0] # time interval
P1 = dt/float(kmax1) # vHa의 주기
P2 = dt/float(kmax1) # VCall의 주기

#P1=3.1333333 min
#P2=3.1333333 min

#%%(c) Correalation

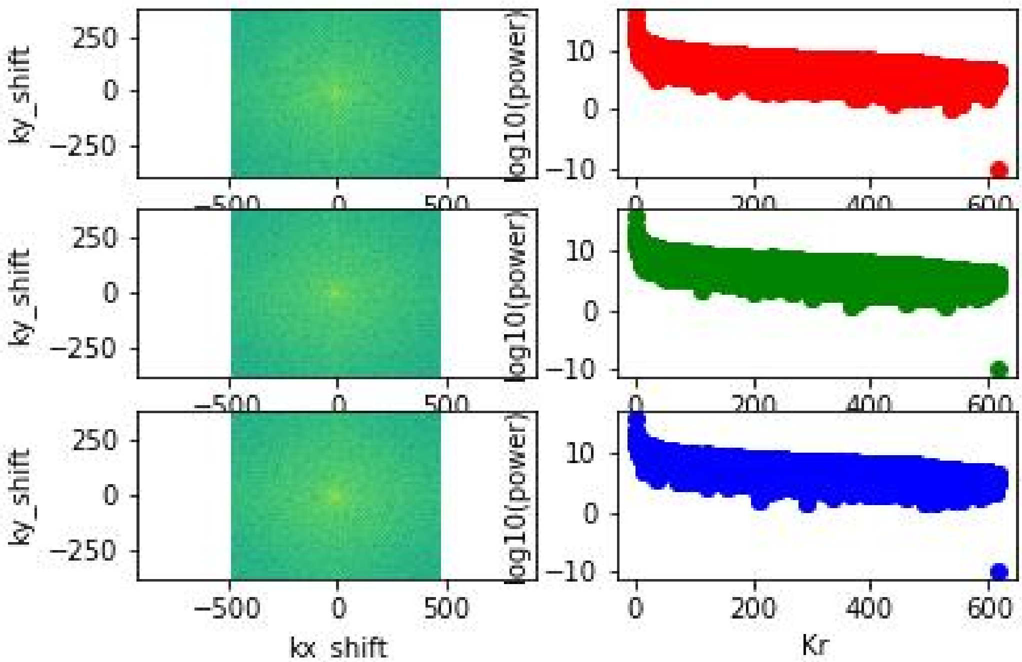
v=v1*v2 <=Hint대로 속력을 fft한 array를 곱해서
iff=np.fft.ifft(v) <= ifft 실행
plt.plot(t,iff),plt.xticks(arange(0,28.2,2)),plt.xlim(0,max(t)),plt.title('Corr'),plt.ylabel('Corr'
),plt.xlabel('t(min)'),plt.savefig('Hw6-3(c).png')


Corr
200
100
0
−100
−200
Corr
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
t(min)

4번. 그림 편집

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

#read image
img=plt.imread('M51_hw.jpg')

lx,ly,lz=img.shape

R=img[:,:,0]
G=img[:,:,1]
B=img[:,:,2]

RF=np.fft.fft2(R)
GF=np.fft.fft2(G)
BF=np.fft.fft2(B)

RF_shift=np.fft.fftshift(RF)
GF_shift=np.fft.fftshift(GF)
BF_shift=np.fft.fftshift(BF)

kx=np.fft.fftfreq(ly,1./ly)
ky=np.fft.fftfreq(lx,1./lx)
kx_shift=np.fft.fftshift(kx)
ky_shift=np.fft.fftshift(ky)

Rp=abs(RF)**2
Rp_shift=abs(RF_shift)**2
Rp[0,0],Rp_shift[0,0]=1.e-10,1.e-10
lRp=np.log10(Rp)
lRp_shift=np.log10(Rp_shift)

Gp=abs(GF)**2
Gp_shift=abs(GF_shift)**2
Gp[0,0],Gp_shift[0,0]=1.e-10,1.e-10
lGp=np.log10(Gp)
lGp_shift=np.log10(Gp_shift)

Bp=abs(BF)**2
```

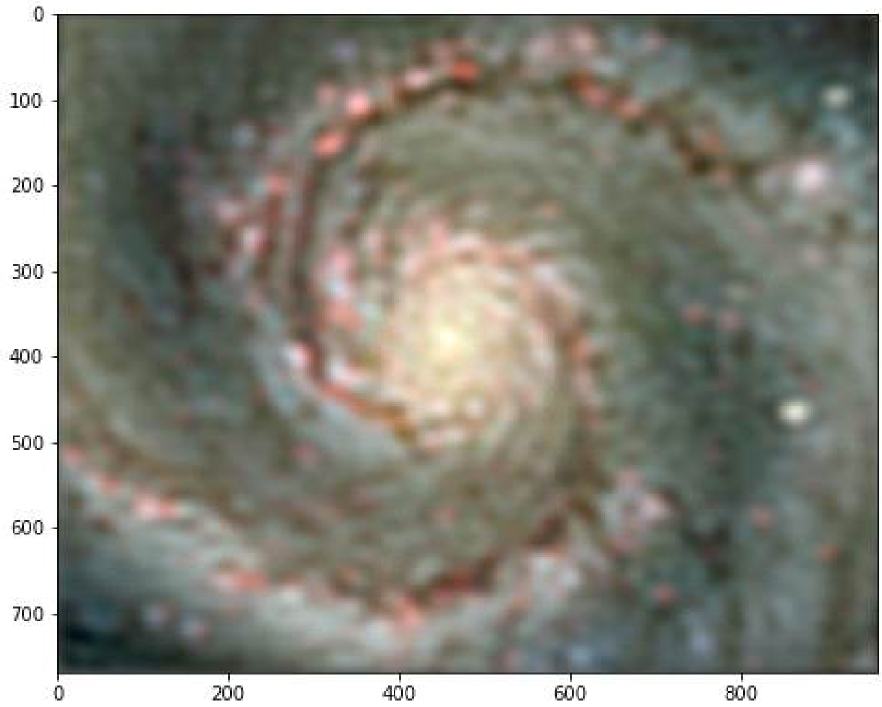
```
Bp_shift=abs(BF_shift)**2
Bp[0,0],Bp_shift[0,0]=1.e-10,1.e-10
lBp=np.log10(Bp)
lBp_shift=np.log10(Bp_shift)


Rkr=np.zeros((lx,ly))
for i in range(lx):
    for j in range(ly):
        Rkr[i,j]=sqrt(kx_shift[j]**2+ky_shift[i]**2)   <-Kr 구함

plt.subplot(321),plt.pcolormesh(kx_shift,ky_shift,lRp_shift),
plt.axis('equal'),plt.xlabel('kx_shift'),plt.ylabel('ky_shift'),\
plt.subplot(323),plt.pcolormesh(kx_shift,ky_shift,lGp_shift),plt.axis('equal'),plt.xlabel('kx
_shift'),plt.ylabel('ky_shift'),\
plt.subplot(325),plt.pcolormesh(kx_shift,ky_shift,lBp_shift),\
plt.axis('equal'),plt.xlabel('kx_shift'),plt.ylabel('ky_shift'),\
plt.subplot(322),plt.plot(Rkr,lRp_shift,'ro'),plt.xlabel('Kr'),plt.ylabel('log10(power)'),plt.su
bplot(324),plt.plot(Rkr,lGp_shift,'go'),plt.xlabel('Kr'),plt.ylabel('log10(power)'),plt.subplot(
326),plt.plot(Rkr,lBp_shift,'bo'),plt.xlabel('Kr'),plt.ylabel('log10(power)'),plt.savefig('Hw6-
4-(a)2.png')
```



선으로 나와야 할텐데 이렇게 나오는 이유를 모르겠습니다...

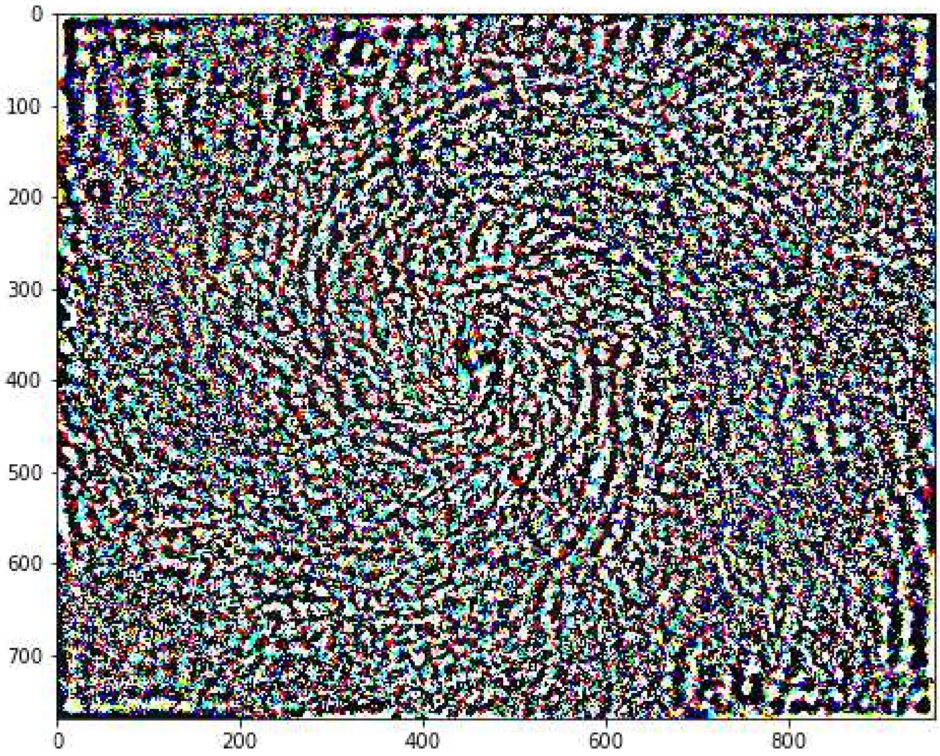
(b) Highfrequency만 남기는 경우

```
kcut=30
X,Y=np.ogrid[0:lx,0:ly]

mask1=(X-lx/2.)**2+(Y-ly/2.)**2 > kcut**2

RFmask, GFmask, BFmask = RF_shift,GF_shift, BF_shift
RFmask[mask1]= 1.e-10*complex(1,1)
GFmask[mask1]= 1.e-10*complex(1,1)
BFmask[mask1]= 1.e-10*complex(1,1)

iFRshift=np.fft.ifftshift(RFmask)
iFGshift=np.fft.ifftshift(GFmask)
iFBshift=np.fft.ifftshift(BFmask)

iFR=np.fft.ifft2(iFRshift)
iFG=np.fft.ifft2(iFGshift)
iFB=np.fft.ifft2(iFBshift)

img1=np.zeros((lx,ly,lz))
img1[:,:,0],img1[:,:,1],img1[:,:,2]=iFR.real.astype(np.int8),iFG.real.astype(np.int8),iFB.real
.astype(np.int8)
plt.figure(figsize=(10,8)),plt.imshow(-img1),plt.savefig('Hw6-4(b).png')
```



(c) 이번경우는 High frequency를 제거하고 low만 남김

```
mask2=(X-lx/2.)**2+(Y-ly/2.)**2 <= kcut**2
RFmask, GFmask, BFmask = RF_shift,GF_shift, BF_shift

RFmask[mask2]= 1.e-10*complex(1,1)
GFmask[mask2]= 1.e-10*complex(1,1)
BFmask[mask2]= 1.e-10*complex(1,1)

iFRshift=np.fft.ifftshift(RFmask)
iFGshift=np.fft.ifftshift(GFmask)
iFBshift=np.fft.ifftshift(BFmask)

iFR=np.fft.ifft2(iFRshift)
iFG=np.fft.ifft2(iFGshift)
iFB=np.fft.ifft2(iFBshift)

img2=np.zeros((lx,ly,lz))
img2[:,:,0],img2[:,:,1],img2[:,:,2]=iFR.real.astype(np.int8),iFG.real.astype(np.int8),iFB.real
.astype(np.int8)

plt.figure(figsize=(10,8)),plt.imshow(img2),plt.savefig('Hw6-4(c).png')
```


0
100
200
300
400
500
600
700
0
200
400
600
800