07. 실습 with Sympy

- Symbolic differenciation

: 매번 도함수나 gradient 함수를 통해서 지정 번거로움

파이썬에서 자동적으로 수행해주는 모듈

→ Symbolic differenciation 이용해

Backtracking line search, Strong backtracking 구현

Local Descent (다변량)은 backtracking / Strong backtracking

이용해 step-size 결정한 다음, 최적화

◦ Univariate

ex) $f(x) = \exp(x-2) - x$ → 함수값, 도함수 값 계산

```
# Example 3.1 : univariate

x=sp.Symbol('x') #심볼로 사용하게 될 변수 선언. x는 변수가 아니라 기호로 인식

f=sp.exp(x-2)-x #sp.exp는 계산 수행함수 아니라 심볼로 처리

df=f.diff(x) #f라는 sympy 함수를 미분. 심볼 x를 통해서~

print('df: ',df)

# 함수 값 계산
derv=sp.lambdify(x,df,'numpy') #x,df: symbol
print('derv(1): ',derv(1)) #계산된 도함수 값 나옴
print('derv(2): ',derv(2))
print('derv(3)',derv(3)) #지수 값이 양수되면 계산된 도함수 값 급격한 증가
```

```
df:  exp(x - 2) - 1
derv(1):  -0.6321205588285577
derv(2):  0.0
derv(3) 1.718281828459045
```

◦ multivariate

ex) $f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2$

```
# Example 4.2 : p.58

def f(x):
    y=x[0]**2+x[0]*x[1]+x[1]**2
    return y

x=sp.IndexedBase('x')
print('x0에 대한 f(x) 편미분: ',sp.diff(f(x),x[0])) # x[0]에 대한 f(x)편미분

print('x1에 대한 f(x) 편미분',sp.diff(f(x),x[1])) # gradient 얻기 위해 2개의 편미분 값 필요

print('두 편미분 합치기: ',[sp.diff(f(x),x[i]) for i in range(2)])

gradients=np.array([sp.diff(f(x),x[i]) for i in range(2)])

grads=sp.lambdify(x,gradients,'numpy')

x_=[1,2]
print(grads(x_))
```

```
x0에 대한 f(x) 편미분:  2*x[0] + x[1]
x1에 대한 f(x) 편미분 x[0] + 2*x[1]
두 편미분 합치기:  [2*x[0] + x[1], x[0] + 2*x[1]]
[4, 5]
```

Local Descent Algorithm 확인

- Local Descent Algorithm

① $x^{(k)}$ 종료 조건 확인 : k번째 반복

② descent direction, $d^{(k)}$ 결정

③ $alpha^{(k)}$ 결정 : 결정된 direction 이용해 α 계산

④ $x^{(k+1)} = x^{(k)} + \alpha^{(k)} \cdot d^{(k)}$: 계획점 이동

→ 이 과정 반복해 backtracking line search로 step-size

구해 최소값 구할 수 있음.

cf. Step 1 에서의 종료 조건 : Termination Condition

1. abstol : $|y - y\_prev| < 1E-8$
2. reltol : $|y - y\_prev| < 1E-8 \times (|y\_prev| + 1E-8)$
3. maximum iteration : over flow 위험으로 반복수 지정

```
# 종료 조건 : abstol

x_=np.array([1,2]) #1번 스텝
d_=-1*np.array(grads(x_)) #2번 스텝
alpha=backtracking_line_search(f,grads,x_,d_) #3번 스텝


y_prev=f(x_) #초기값으로 계산된 값

#종료조건을 만족하게끔 반복 수행
flag=True #bool 변수 선언
# 이 값이 false일때 종료

i=1
while flag: # 첫번째 계산 수행했으니까 무조건 한 번 더 계산 수행

    x_=x_+alpha*d_ #4번 단계 :local model 이동

    d_=-1*np.array(grads(x_))

    alpha=backtracking_line_search(f,grads,x_,d_)

    y_=f(x_)

    # 종료 할지말지 판단 근거: abstol 이용
    diff=np.abs(y_-y_prev)
    print(i,alpha,x_,f(x_),diff)

    if diff<1E-8:
        flag=False

    y_prev=y_ #조건 만족 안하면 비교를 위해 y_prev 지정
    i+=1
```

```
# reltol 종료조건 사용하기!

x_=np.array([1,2]) #1번 스텝
d_=-1*np.array(grads(x_)) # 2번 스텝
alpha=backtracking_line_search(f,grads,x_,d_)# 3번 스텝

y_prev=f(x_) #초기값으로 계산된 값

#종료조건을 만족하게끔 반복 수행
flag=True #bool 변수 선언
# 이 값이 false일때 종료

i=1
while flag: # 첫번째 계산 수행했으니까 무조건 한 번 더 계산 수행

    x_=x_+alpha*d_ #4번 단계 :local model 이동

    d_=-1*np.array(grads(x_))

    alpha=backtracking_line_search(f,grads,x_,d_)

    y_=f(x_)

    # 종료 할지말지 판단 근거: abstol 이용
    diff=np.abs(y_-y_prev)
    print(i,alpha,x_,f(x_),diff)

    if diff<1E-8*(np.abs(y_prev)+1E-8):
        flag=False

    y_prev=y_ #조건 만족 안하면 비교를 위해 y_prev 지정
    i+=1
```

```
# Maximum iteration 사용

x_=np.array([1,2]) #1번 스텝
d_=-1*np.array(grads(x_)) # 2번 스텝
alpha=backtracking_line_search(f,grads,x_,d_)# 3번 스텝

y_prev=f(x_) #초기값으로 계산된 값
max_iter=100

flag=True #bool 변수 선언
# 이 값이 false일때 종료

i=1
while flag: # 첫번째 계산 수행했으니까 무조건 한 번 더 계산 수행

    x_=x_+alpha*d_ #4번 단계 :local model 이동

    d_=-1*np.array(grads(x_))

    alpha=backtracking_line_search(f,grads,x_,d_)

    y_=f(x_)

    # 종료 할지말지 판단 근거: abstol 이용
    diff=np.abs(y_-y_prev)
    print(i,alpha,x_,f(x_),diff)

    if i>max_iter:
        flag=False

    y_prev=y_ #조건 만족 안하면 비교를 위해 y_prev 지정
    i+=1
```

⇒ backtracking line search 알고리즘 이용 위해

α, β 최적값 결정은 종료조건에 따라 계산결과 달라짐.

↓

목적 함수 값이 어느정도 크기를 갖게끔 계산 될?

gradient 값의 크기가 어느정도?

α값이 대체적으로 어떤 값?

에 따라서 종료조건 선택 하기

```
# 함수화 : local descent with backracking line search

def local_descent_backtracking(f,grads,x_,alpha=10,TOL=1E-8):

    d_=-1*np.array(grads(x_))

    alpha=backtracking_line_search(f,grads,x_,d_,alpha)
    print(alpha,x_,f(x_))

    y_prev=f(x_)

    i=1
    flag=True
    while flag:

        x_=x_+alpha*d_
        d_=-1*np.array(grads(x_))

        alpha=backtracking_line_search(f,grads,x_,d_,alpha)

        y_=f(x_)

        diff=np.abs(y_-y_prev)
        print(i,alpha,x_,f(x_),diff)

        if diff<TOL*(abs(y_prev)+TOL): #종료조건 reltol사용
            flag=False

        y_prev=y_
        i+=1

    return i,x_
```

```
x_=np.array([1,2])
local_descent_backtracking(f,grads,x_) #메소드 만들어졌음 -> 쉽게 구현
```

Strong_backtracking Algorithm

```
# Algorithm 4.3 : p.62

def strong_backtracking(f,grads,x,d,alpha=1.0,beta=1E-4,sigma=1E-1):

    y0,g0,y_prev,alpha_prev=f(x),np.dot(grads(x),d),np.nan,0.0
    alpha_lo,alpha_hi=np.nan, np.nan

    # bracket phase
    while True:
        y=f(x+alpha*d)

        if y>y0+beta*alpha*g0 or (not(np.isnan(y_prev)) and y>=y_prev):
            alpha_lo,alpha_hi =alpha_prev,alpha
            break
        g=np.dot(grads(x+alpha*d),d)

        if np.abs(g)<=-sigma*g0:
            return alpha
        elif g>=0:
            alpha_lo,alpha_hi=alpha,alpha_prev
            break
        y_prev,alpha_prev,alpha=y,alpha,2*alpha

    # zoom phase
    y_lo=f(x+alpha_lo*d)

    while True:

        alpha=0.5*(alpha_lo+alpha_hi)
        y=f(x+alpha*d)

        if (y>y0+beta*alpha*g0) or (y>=y_lo):
            alpha_hi=alpha

        else:
            g=np.dot(grads(x+alpha*d),d)

            if abs(g)<=-sigma*g0:
                return alpha
            elif g*(alpha_hi-alpha_lo)>=0.0:
                alpha_hi=alpha_lo
            alpha_lo=alpha
```

```
def local_descent_strong_backtracking(f,grads,x_,alpha=10,TOL=1E-8):

    d_=-1*np.array(grads(x_))
    alpha=strong_backtracking(f,grads,x_,d_,alpha) #달라지는 부분!
    print(alpha,x_,f(x_))

    y_prev=f(x_)

    i=1
    flag=True

    while flag:

        x_=x_+alpha*d_
        d_=-1*np.array(grads(x_))

        alpha=strong_backtracking(f,grads,x_,d_,alpha) # 달라지는 부분! alpha(단계값)계산

        y_=f(x_)

        diff=np.abs(y_-y_prev)
        print(i,alpha,x_,f(x_),diff)

        if diff<TOL*(abs(y_prev)+TOL): #세가지 종료 조건 이용해 구할 수 있음
            flag=False

        y_prev=y_
        i+=1

    return i,x_
```

```
x_=np.array([1,2])
local_descent_strong_backtracking(f,grads,x_)
```