06. 경사법 (Local descent)

$\downarrow$

$\boxed{\text{다변량}}$

- 반복 계산 과정 (Algorithm)
- 선 탐색 (Line search)
- 근사 선 탐색 (approximate line search)
  - → Armijo 조건 , 곡률 (curvature) 조건
  - Wolfe 조건
- strong backtracking line search

국소 하강법 (Local Descent)

- 최적화

feasible set : 해가 있는 집합

: $\underset{x}{\text{minimize}}\ f(x)$ subject to $x \in \mathcal{X}$

$f(x)$
- 일변량 : 구간 분할법 사용
- 다변량 : Local Descent Method 사용
  - ↳ gradient = 0 이 되는 최소값 x 찾기

gradient 움직이는 방향으로 x값 (계획점) 이동시켜

목적함수 값 계산해 작아질 때 까지 반복

※ 다변량 Algorithm 정리

① 초기 계획점 결정, 목적 함수 값 계산

② 종료 조건 : 이전의 계획점의 목적함수와 새롭게 계산된 목적함수 차이 X

③ 목적함수 값 작아지도록 계획점 어떻게 이동?

: gradient = 0 인 점 최소값. gradient 이용해 목적함수 감소

방향 결정. 결정된 방향으로 계획점 이동

이 때 이동되는 크기 (단계값)은 line Search method 통해

매 반복 마다 찾아낸다.

k번째 계획점에서 경사도 하강방향으로 단계값 곱해준 만큼 이동

- 하강 방향법 : Algorithm

① 초기 계획점 $x^{(1)}$ 결정

② k번째 반복에서의 계획점 $x^{(1)}$ 종료조건 확인

③ $x^{(k)}$ 에서 경사도 (헷세 행렬) 정보 이용해 하강방향 $d^{(k)}$ 결정

cf. 경우에 따라 $\|d\| = 1$

④ 단계값 (학습률) $\alpha^{(k)}$ 결정

⑤ (k+1) 번째 계획점 갱신 : $x^{(k+1)} \leftarrow x^{(k)} + \alpha^{(k)} \cdot d^{(k)}$

— 선 탐색 (Line Search Method) : 단계값 탐색

∘ 목적 함수 : $\underset{\alpha}{\text{minimize}}\ f(\underbrace{\overset{\text{계획점}}{x} + \alpha d}_{\text{하강 방향}})$ → x와 d 주어진 값

∘ 주어진 계획점 x와 하강 방향 d에 대하여 목적함수 f를

최소화 하는 α를 찾는 문제

∘ 일변량 탐색법 : α라는 일변량 미지수 최소화

⇒ bracket_min 으로 구간 찾기

Golden Search Method 로 α를 매 반복마다 찾기

→ 계산량 많아 Approach Line Search Method 사용

- 근사 선 탐색 (Approximate line Search) : Armijo 조건

∘ 매 반복에서 선 탐색을 수행하는 것이 아니라 보다 적은 단계값

이용해 반복 수행

∘ 하강법은 항상 하강하므로 목적함수를 작게할 수 있다면

적절히 작은 α 만으로도 충분

⇒ Armijo 조건 이용해 α를 최적화 문제 아니라 계산문제로 바꿔

α 찾기

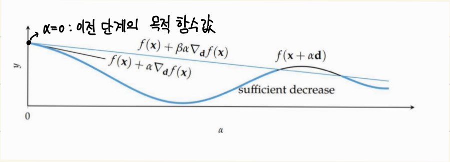
이 값 기준으로 테일러 전개 / k번째 gradient / k번째 velocity / 방향 도함수 (directional derivates)

$$\Rightarrow f(x^{(k)} + \alpha \cdot d^{(k)}) \approx f(x^{(k)}) + \alpha\, \boxed{\nabla f(x^{(k)})^T d^{(k)}}$$

① $f(x^{(k+1)}) \le f(x^{(k)}) + \beta \cdot \alpha \cdot \nabla_{d^{(k)}} f(x^{(k)})$ → α를 β만큼 완화시키기

$f(x^{(k+1)}) \le f(x^{(k)}) + \beta \cdot \alpha \cdot \nabla f(x^{(k)})^T \cdot d^{(k)}$

② $\beta \in [0, 1]$: $\beta = 1 \times 10^{-4}$ 로 많이 결정함



국소 하강법 (Local Descent)

- 근사 선 탐색 : 곡률 (curvature) 조건

• 지나치게 크거나 작은 α값이 Armijo 조건에 의해 찾아지면 최소값에 수렴 X → curvature 조건 추가 필요

• 목적함수 : $\underset{\alpha}{\text{minimize}}\ f(x^{(k)} + \alpha d)$ velocity

→ 목적함수 자체를 local descent에 의해 찾기, 구간 분할법 X

① $f(x^{(k+1)}) = f(x^{(k)} + \alpha d)$

$\nabla_{d^{(k)}} f(x^{(k+1)}) = \boxed{\nabla f(x^{(k+1)})^T d^{(k)}}$ α에 대해 미분

k번째에서의 값이 함수 내 포함

$\nabla_{d^{(k)}} f(x^{(k+1)}) \geq \sigma \nabla_{d^{(k)}} f(x^{(k)})$ 이전 계획점에서의 방향 도함수 값

1보다 작은 값

$f(x^{(k+1)})$ 기울기 $= \nabla f(x^{(k+1)})^T d^{(k)}$

② $f(x^{(k)} + \alpha d^{(k)}) \approx f(x^{(k)}) + \alpha \cdot \nabla f(x^{(k)})^T d^{(k)}$

α에 대해 미분

$\nabla f(x^{(k+1)})^T d^{(k)} \approx \nabla f(x^{(k)})^T d^{(k)}$ → (k+1)번째의 x 대한 도함수 이전 반복의 도함수에 근사

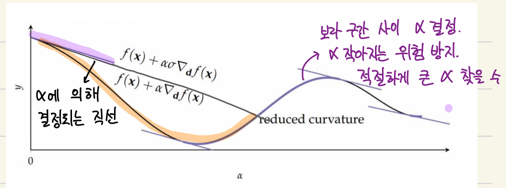
$\nabla f(x^{(k+1)})^T d^{(k)} \geq \sigma \nabla f(x^{(k)})^T d^{(k)}$ ⇒ Curvature Condition

$\sigma \in (\beta, 1)$, $\sigma \in \{0.1, 0.9\}$로 多 사용

α에 대한 기울기 크게 해줌 : $\nabla f(x^{(k+1)})^T d^{(k)}$

⊕ Strong Wolfe Condition

: $|\nabla_{d^{(k)}} f(x^{(k+1)})| \geq -\sigma \nabla_{d^{(k)}} f(x^{(k)})$



구간 내 모든 α는 Armijo 조건 만족

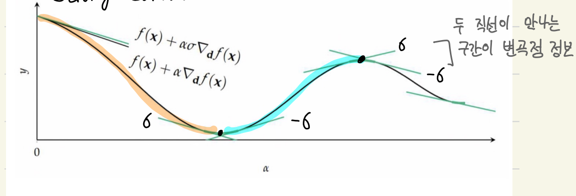
→ α 매우 작아질 위험 有 : 계산량 多, 수렴 X 문제

→ α 크게 하기 위해 σ값 곱해 Curvature information 추가

→ 보라색 함수 직선이 목적함수의 접선 (탄젠트 라인)과 같음

⇒ Curvature 조건은 α에 의해 변하는 함수에서 기울기 변하게 해주는 변곡점 구간 결정하자! (idea)

Strong Curvature Condition



: $|\nabla_{d^{(k)}} f(x^{(k+1)})| \geq -\sigma \nabla_{d^{(k)}} f(x^{(k)})$

⇒ Curvature 조건이 추가되면, Armijo 조건에 의해 찾아진 α값이 갖는 구간 내 값 삭제

구간으로 α값 제한해 α 작아지는 문제점 해결

국소 하강법 (Local Descent)

- Strong Wolfe 조건

- 충분 감소 조건 (Armijo 조건)

  : $f(x^{(k+1)}) \le f(x^{(k)}) + \beta \alpha \nabla_{d^{(k)}} f(x^{(k)})$

  → α 작게 하고 이동시키면 목적함수값 작아짐. 너무 큰 α값 제거

- 곡률 조건 (Curvature Condition)

  : $\nabla f(x^{(k+1)})^T d^{(k)} \ge \sigma \cdot \nabla f(x^{(k)})^T d^{(k)}$

  → 너무 작은 α값 제거 by. 변곡점 정보

  절댓값 취하면 strong wolfe condition

- Strong backtracking algorithm

- Step 1) back tracking phase

  : $[\alpha^{(k-1)}, \alpha^{k}]$의 충분히 큰 α를 포함하는 구간 탐색

- Step 2) Zoom phase

  : Strong wolfe 조건이 만족되도록 구간 축소

  → 구간 차 적어지면, 구간 중앙값으로 α 지정

- backtracking phase : 구간 탐색

조건 ↳ 적절히 큰 α 존재하는 구간 찾기

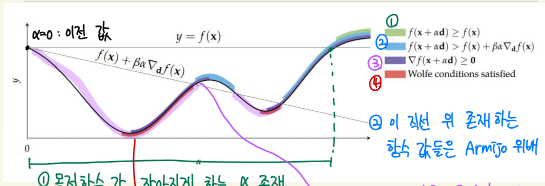
① $f(x+\alpha^{(k)} d) \ge f(x)$

② $f(x^{(k)} + \alpha^{(k)} d^{(k)}) > f(x^{(k)}) + \beta \alpha^{(k)} \nabla_{d^{(k)}} f(x^{(k)})$ Armijo 위배

③ $\nabla f(x+\alpha^{(k)} d) \ge 0$

⇒ 구간 탐색시, α가 가진 특징 위배되는 경우로 이용.

적절히 큰 α 존재하는 구간 찾기!



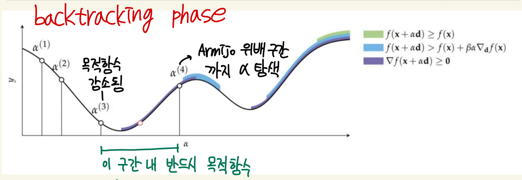
① 목적함수 값 작아지게 하는 α 존재.

④ α에 대한 도함수 기울기 =0 구간만 고려!

Curvature 조건 포함해 α 구간 결정

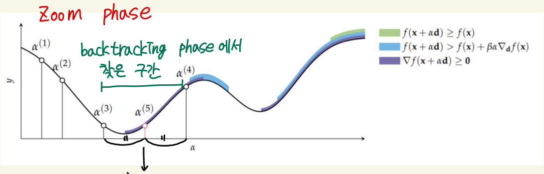
③ Armijo 이후, 목적함수 값 작게 해주는 α값 구간 내 존재

backtracking phase



이 구간 내 반드시 목적함수 작게 하는 최적 α 존재

Zoom phase



$\alpha^{(5)}$는 Bisection Method 이용해 찾은 값.

구간 감소 : 줄어든 구간 내에서 똑같이 구간 step halving.

반씩 나눠가며 α 찾자!

국소 하강법 (Local Descent)

- Strong back tracking

: Algorithm

```
function strong_backtracking(f, ∇, x, d; α=1, β=1e-4, σ=0.1)
    y0, g0, y_prev, α_prev = f(x), ∇(x)·d, NaN, 0
    αlo, αhi = NaN, NaN

    # bracket phase
    while true
        y = f(x + α*d)
        if y > y0 + β*α*g0 || (!isnan(y_prev) && y ≥ y_prev)
            αlo, αhi = α_prev, α
            break
        end
        g = ∇(x + α*d)·d
        if abs(g) ≤ -σ*g0
            return α
        elseif g ≥ 0
            αlo, αhi = α, α_prev
            break
        end
        y_prev, α_prev, α = y, α, 2α
    end
```

→ 계산에서 초기값 : 알려지지 X

앞 그림에서 초록 조건

or

and

→ Strong wolfe condition의 2번째 조건.

⌋ α 결정 구간 찾음

```
    # zoom phase
    ylo = f(x + αlo*d)
    while true
        α = (αlo + αhi)/2
        y = f(x + α*d)
        if y > y0 + β*α*g0 || y ≥ ylo
            αhi = α
        else
            g = ∇(x + α*d)·d
            if abs(g) ≤ -σ*g0
                return α
            elseif g*(αhi - αlo) ≥ 0
                αhi = αlo
            end
            αlo = α
        end
    end
end
```

→ Bysection 통해 Step harving

— 중앙값

왼쪽 값

오른쪽 값

- 인자 설명

f : 목적 함수 , ∇ : 도함수 , d : 방향 벡터

α : α초기값 , β : Armijo 조건   σ : curvature 조건

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

$x = [1, 2]$, $d = [-1, -1]$

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

def gradient(x):
    return [2*x[0]+x[1],2*x[1]+x[0]]

x=np.array([1,2])
d=np.array([-1.1,-1.2]) # 방향벡터 변경함 -> 계산 편리성

print('f([1,2])',f(x))
print('gradient([1,2])',gradient(x))
print(' ')


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

    y0,g0,y_prev,alpha_prev=f(x),np.dot(gradient(x),d),np.nan,0.0
    alpha_lo,alpha_hi=np.nan, np.nan # 값 저장안되고 메모리 주소만 있음 -> 결측

    ## bracket phase ##
    while True:
        y=f(x+alpha*d)
        # Armijo condition 위배 확인
        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(gradient(x+alpha*d),d) # 방향 도함수 값 계산
        if np.abs(g) <= -sigma*g0: # 절댓값 붙어있는 strong condition
            return alpha
        elif g>=0: # 구간 옮길 수 있음
            alpha_lo,alpha_hi , alpha,alpha_prev
            break

        y_prev,alpha_prev,alpha=y,alpha,2*alpha
    print('bracket: %.4f, %.4f' %(alpha_lo,alpha_hi))

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

    while True: # bysection -> step harving
        alpha=0.5*(alpha_lo+alpha_hi)
        y=f(x+alpha*d)# 이동

        # Arimijo condition
        if (y>y0+beta*alpha*g0) or (y>=y_lo):
            alpha_hi =alpha
        else:
            g=np.dot(gradient(x+alpha*d),d) # 두번째 wolfe 조건 확인
            if abs(g)<=-sigma*g0:
                return alpha
            elif g*(alpha_hi-alpha_lo)>=0: # 구간만큼 이동값 0 이상 -> 3번째 조건 만족되는지 확인
                alpha_hi=alpha_lo
            alpha_lo=alpha


alpha=strong_backtracking(f,gradient,x,d)
print(alpha)
print(x+alpha*d) # x 계획점은 이 구간으로 이동
print(f(x+alpha*d)) #그 계획점에서의 목적함수 값
```

(k+1)번째에서 방향 도함수 값

이미 계산되어서 y_pred 가 NAN 구분 조건 X

```
f([1,2]):  7
gradient([1,2]):  [4, 5]

bracket: 1.0000, 2.0000
1.25
[-0.375  0.5  ]
0.203125
```

첫번째 backtracking phase 에서 구간 [1,2] 정해짐

Zoom phase에서 2번 self harving : 1.25

계획점 이동

새로운 계획점에서의 목적함수 값