HW13 최적화 실습

[Git code]

Mathvision22
02221081 황지현

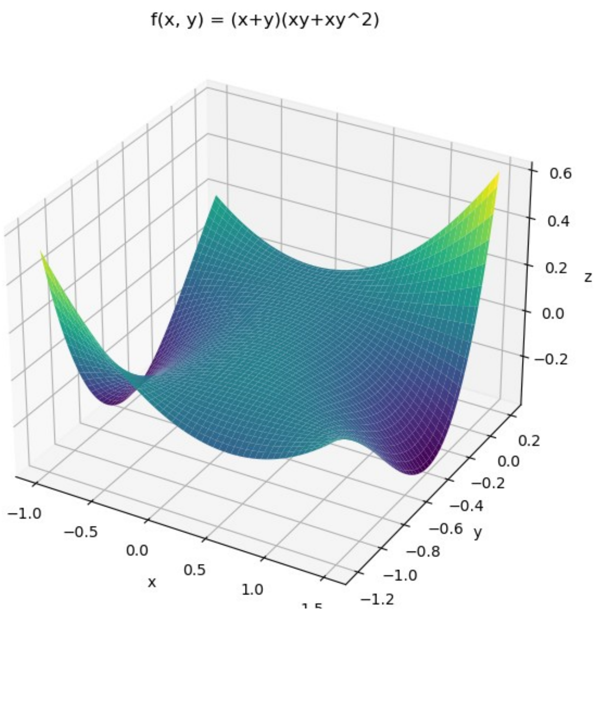
Q1. 다음과 같은 이변수 스칼라 함수 f에 대해 아래의 내용을 수행하시오.

$$z = f(x, y) = (x + y)(xy + xy^2)$$

– -1<=x<=1.5, -1.2<=y<=0.2 구간에서 이 함수의 그래프를 도시하시오.

\# 그래프 도사

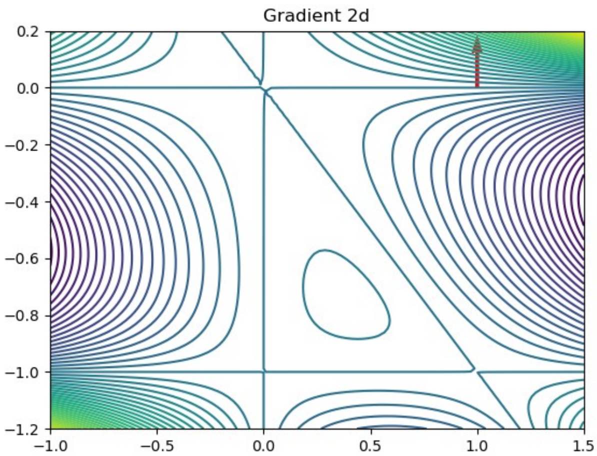
```python
# 함수
def f(x, y):
    # f(x, y) = (x+y)(xy+xy^2)
    return (x + y) * (x * y + x * y * y)

# x, y 구간 3d 그래프 도사
x = np.linspace(-1, 1.5, 100)
y = np.linspace(-1.2, 0.2, 100)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

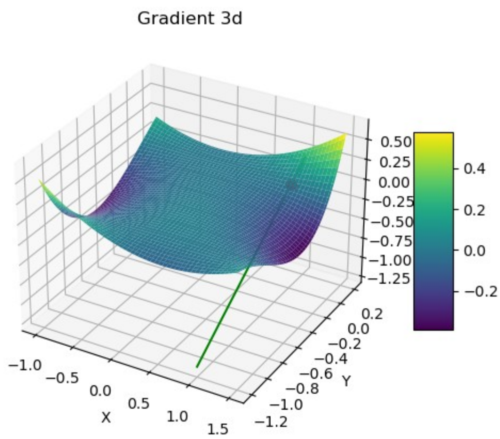


– (1, 0)에서 f의 gradient를 구한 후, 구한 gradient가 함수의 최대 증가 방향과 일치함을 그래프를 통해 확인하시오.

```
def gradiant(x,y) :
    # f(x, y) = (x+y)(xy+xy^2)
    # df/dx = y + 2xy + y^2
    # df/dy = x + xy + x^2
    return np.array([y + 2*x*y + y*y, x + x*y + x*x])

p = np.array([1, 0])
print(gradiant(p[0], p[1]))  gradient at (1, 0) :  [0 2]

grad = gradiant(p[0], p[1])
grad = grad / np.linalg.norm(grad)
```

(1, 0) 에서의 gradient 구하기

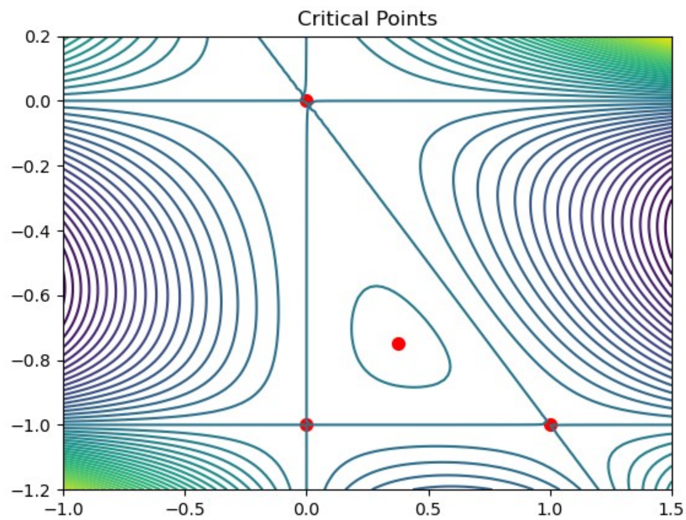




– 이 함수의 모든 critical point를 구한 후, 각 critical point에서 이 함수가 극대인지, 극소인지, 안장점(saddle point)인지 여부를 Hessian 테스트를 이용하여 판별하시오. 그리고 실제 그래프를 통해 이를 확인하시오.

critical points

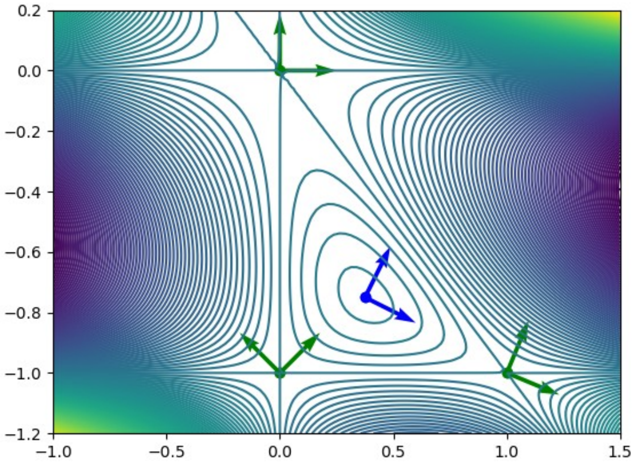
```
critical_points = get_critical_points()
critical_points = np.array(critical_points, dtype=np.float64)
print("cirtical points : ", critical_points)
plt.contour(X, Y, Z, 50, cmap='viridis')
plt.scatter(critical_points[:, 0], critical_points[:, 1], color='red', s=50)
plt.title("Critical Points")
plt.show()
```



극대, 극소, 인장점(saddle point) 찾기

```
for x, y in critical_points:
    hess = hessian(x, y)
    #print(np.linalg.eigvals(hess))

    if np.all(np.linalg.eigvals(hess) > 0):
        print("x: {}, y: {} is a local minimum".format(x, y))
    elif np.all(np.linalg.eigvals(hess) < 0):
        print("x: {}, y: {} is a local maximum".format(x, y))
    else:
        print("x: {}, y: {} is a saddle point".format(x, y))
```

```
x: 0.0, y: -1.0 is a saddle point
x: 0.0, y: 0.0 is a saddle point
x: 0.375, y: -0.75 is a local maximum
x: 1.0, y: -1.0 is a saddle point
```



4 개의 critical points 중 1 개는 maximum(청색), 3 개는 saddle point(녹색)임을 볼 수 있다.

Critical point: (0.0, -1.0)
critical point: [0. -1.], hessian: [[-0. 1.]
 [1. -0.]]
eigenvalues: [1. -1.], eigenvectors: [[0.70710678 -0.70710678]
 [0.70710678 0.70710678]]
saddle point: [0. -1.]

Critical point: (0.0, 0.0)
critical point: [0. 0.], hessian: [[0. 0.]
 [0. 0.]]
eigenvalues: [0. 0.], eigenvectors: [[1. 0.]
 [0. 1.]]
saddle point: [0. 0.]

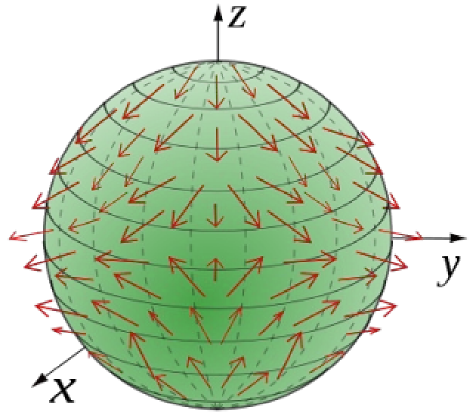
Critical point: (0.375, -0.75)
critical point: [0.375 -0.75], hessian: [[-0.375   -0.1875]
 [-0.1875 -0.65625]]
eigenvalues: [-0.28125 -0.75   ], eigenvectors: [[0.89442719 0.4472136]
 [-0.4472136   0.89442719]]
local maximum: [0.375 -0.75]

Critical point: (1.0, -1.0)
critical point: [1. -1.], hessian: [[-0. -1.]
 [-1. -2.]]
eigenvalues: [0.41421356 -2.41421356], eigenvectors: [[0.92387953 0.38268343]
 [-0.38268343 0.92387953]]
saddle point: [1. -1.]

Q2. 다변수 함수에서 함수의 gradient는 함수의 최대증가 방향이 됨을 증명 또는 설명하시오.

다변수 함수에서의 Gradient 정의

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$$



- Gradient 는 스칼라 함수의 임의의 점 a 에서 나올 수 있는 법선 벡터의 크기와 방향을 모두 고려하여 계산한 스칼라 함수의 최대 공간 증가율을 의미한다.

  (법선 벡터이므로 주어진 함수에 대해 수직임을 알 수 있다.)

- 위 식을 보면, 각 변수의 일차 편미분 값으로 구성되는 벡터(2 변수 이상일 경우)이며 각 벡터의 크기는 증가의 기울기(증가량의 크기)를 나타낸다.

  : 벡터의 크기 가장 크면 최대증가 방향이 된다.

- 반대로 생각해 보았을 때, 음수를 취하게 되면 최소점을 찾을 수 있으므로 이 특성을 활용하여 함수의 최대 및 최소값을 구할 수 있다.

Q3. 다음과 같은 이변수 스칼라 함수 f에 대해 아래의 내용을 수행하시오.

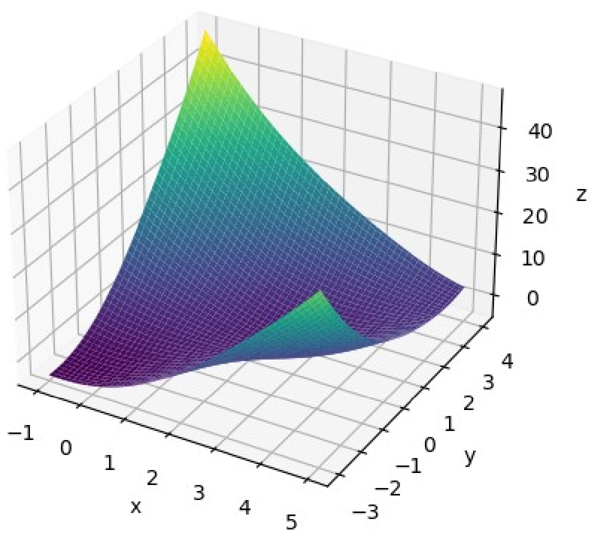
$$f(x, y) = \sin(x + y - 1) + (x - y - 1)^2 - 1.5x + 2.5y + 1$$

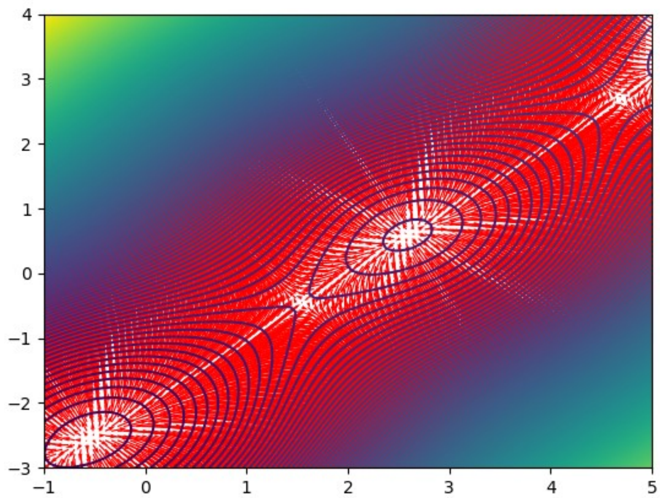
- -1<x<5, -3<y<4 구간에서 이 함수의 그래프를 도시하시오.
- gradient descent 방법을 이용하여 이 함수의 최소값 및 최소점을 구하시오 (lamda 및 초기값은 임의로 정해서 사용, 최적화 과정을 그래프에 도시하여 확인)
- Newton's 방법을 이용하여 이 함수의 최소값 및 최소점을 구하시오 (gradient descent 방법의 경우와 동일한 초기값에서 출발, 최적화 과정을 그래프에 도시하여 확인)
- gradient descent 방법과 Newtown's 방법의 결과를 비교하시오. (수렴하는데 걸리는 step의 수 등...)

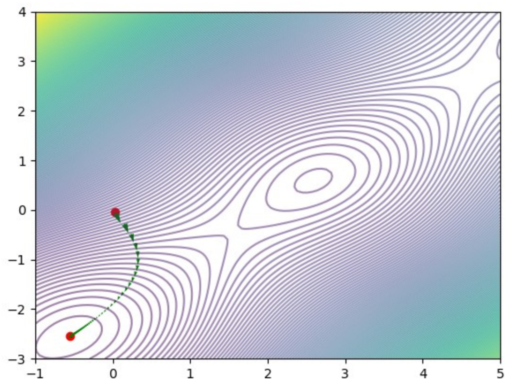
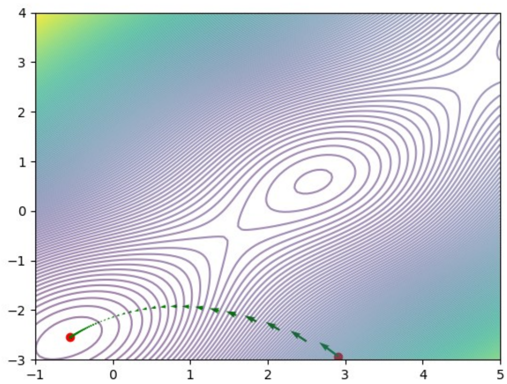
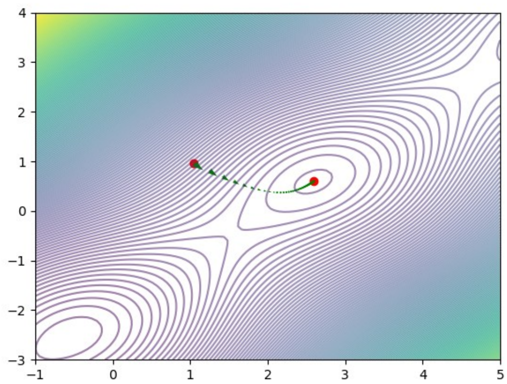
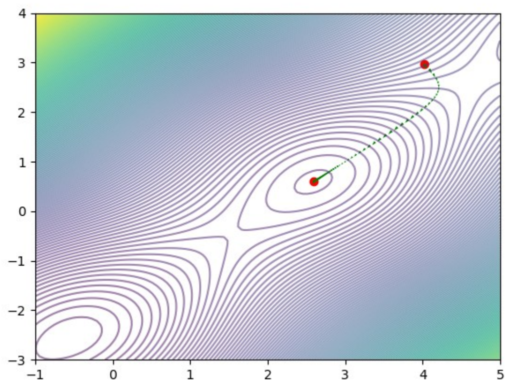
\# 함수 그래프 도사

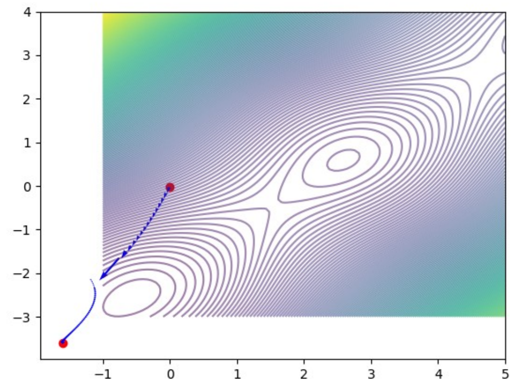
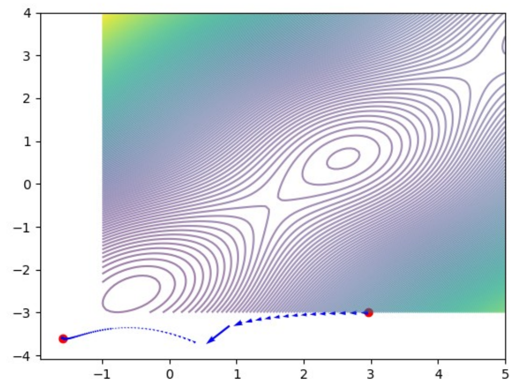
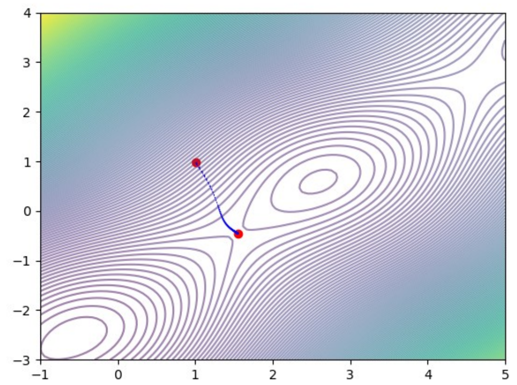
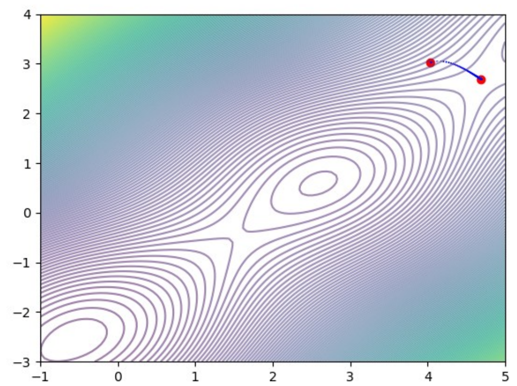
```python
def f(x, y):
    # f(x, y) = sin(x+y+1) + (x-y-1)^2 - 1.5x + 2.5y + 1
    return np.sin(x+y+1) + (x-y-1)**2 - 1.5*x + 2.5*y + 1

x = np.linspace(-1, 5, 100)
y = np.linspace(-3, 4, 100)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

f(x, y) = sin(x+y+1) + (x-y-1)^2 - 1.5x + 2.5y + 1





<table>
<tr><th colspan="2">Gradient method</th></tr>
<tr><th>graph</th><th>elements</th></tr>
<tr><td>
</td><td>start point : [0.0, 0.0]
converge at : 872 steps
converge point : [-0.54719714 -2.54719714]</td></tr>
<tr><td>
</td><td>start point : [3.0, -3.0]
converge at : 872 steps
converge point : [-0.54719714 -2.54719714]</td></tr>
<tr><td>
</td><td>start point : [1.0, 1.0]
converge at : 849 steps
converge point : [2.59439468 0.59439468]</td></tr>
<tr><td>
</td><td>start point : [4.0, 3.0]
converge at : 928 steps
converge point : [2.59439552 0.59439552]</td></tr>
</table>

Newton’s method	
graph	elements
	start point : [0.0, 0.0] converge at : 1238 steps converge point : [-1.59439573 -3.59439447]
	start point : [3.0, -3.0] converge at : 1298 steps converge point : [-1.59439447 -3.59439573]
	start point : [1.0, 1.0] converge at : 1233 steps converge point : [1.54719692 -0.45280182]
	start point : [4.0, 3.0] converge at : 1186 steps converge point : [4.68878959 2.68879082]

Gradient descent & Newton method

- 두 가지 방법을 동일한 함수와 좌에 적용하여 결과를 비교해 본 결과, 수렴하는 point 와 step, 속도 등 에서 차이가 나타난다는 것을 알 수 있었다.

- Newton method :
  Gradient method 와 비교하여 보았을 때 수렴속도가 더 빨랐지만, 해에 가까이 도달할 수록 속도가 느려지는 문제점이 있었다.
  해를 근사적으로 찾아냄, 정확한 해에 도달하는 방식이 아니기 때문에 보완을 위해 다른 방식을 추가적으로 적용할 필요가 있다고 생각된다.

- Gradient descent :
  해에 근접할 수록 기울기가 0 에 가까워져 수렴 속도가 느려졌다.
  f'(x)가 0 이 되는 점을 찾기 때문에 극대/극소/안장점에 더 정확하게 도달한다고 보여짐.

- 주어진 문제 (함수)에 어떤 방식을 적용해야 더 좋다고 판단하기에는 아직은 잘 모르겠으나, 수업시간에 배웠던 이론을 생각해 보았을 때 모든 차원과 공간에 적용이 가능한 경사하강법을 사용하는 것이 조금은 더 효율적일 것이라고 생각된다.

결론

- Gradient 를 이용하여 주어진 다변수 함수의 최대 및 최소값을 구하는 방법에 대해 이해할 수 있었다.

- Hessian Matrix 를 활용하여 극대, 극소, 안장점을 찾아볼 수 있었다.

- Gradient descent & Newton method 의 차이점을 알아볼 수 있었다.