REPORT

최적화 알고리즘 구체화 프로젝트

과목명	수치해석
담당교수	김상철 교수님
학과	소프트웨어학부
학년	2학년
학번	20171664
이름	이범석

선택한 최적화 알고리즘의 개요 및 동작원리

모멘텀 알고리즘을 선택했다.

모멘텀은 관성이라는 뜻을 가지고 있고, 수치해석에서는 경사하강법에 관성을 더해 주는 알고리즘이다.

함수 실행 시 기울기를 매번 구하고, 이전 수정 방향을 참고하여 같은 방향으로 일정한 비율만 수정하게 하는 방향이다.

이는 진동 현상을 줄여주고, 다음 값이 이전 이동값을 고려하여 일정 비율만큼 이동하므로 관성의 효과가 있다.

선택한 최적화 알고리즘의 동작 코드와 단위 테스트

최적화 모멘텀 알고리즘을 테스트하기 위하여 로젠브록 함수와 로젠브록 함수의 도함수를 구해줄 함수를 구현했다.

코드 원형은 다음과 같다.

```
def f2(x, y):
    return (1 - x)**2 + 100.0 * (y - x**2)**2

def f2g(x, y):
    #f2(x, y)의 도함수#
    # (1 - x)**2 + 100.0 * (y - x**2)**2
    return np.array((2.0 * (x - 1) - 400.0 * x * (y - x**2), 200.0 * (y -
x**2)))
```

f2는 x, y = (1, 1) 에서 최솟값을 가지는 함수이다.

모멘텀 알고리즘을 구현하기 위해서 반복 실행 횟수, 학습률, 오차 범위, 모멘텀 계수를 정의했다.

이후 (1, 1)로 수렴하기 위해 시작할 좌표를 정해 주었다.

모멘텀 알고리즘은

```
모멘텀 계수 * 이전 이동값 - 학습률 * 편미분 값
```

의 형태로 구성되며, 이를 구현한 코드는 다음과 같다.

```
i_max = 10000 # 실행 횟수
alpha = 0.00089 # 학습률
eps = 0.000001 # 오차 범위
m = 0.7 # 모멘텀 계수

w_init = [-2, -2]# 시작 좌표
w_i = np.zeros([i_max, 2])
w_i[0, :] = w_init
x = w_init[0]
y = w_init[1]
vx = 0
vy = 0
```

```
XY = list()
XY.append([x, y])
iter = 0 # 반복 횟수
for i in range(1, i_max):
    fdx = f2g(x, y)
    vx = m * vx - alpha * fdx[0]
    vy = m * vy - alpha * fdx[1]
    x += vx
    y += vy
    iter = i
    w_i[i, 0] = x
    w_i[i, 1] = y
    XY.append([x, y])
    if f2(x, y) < eps:
        break

print(x, y)
print(iter)
print(XY)

XY = np.array(XY)
```

이처럼 (1, 1)으로 수렴하는 동안 지나게 되는 x와 y의 좌표를 XY 배열에 저장하였다.

이를 가시적으로 확인하기 위하여 matplotlib을 이용하여 그래프로 나타내고자 했다.

구현 코드는 다음과 같다.

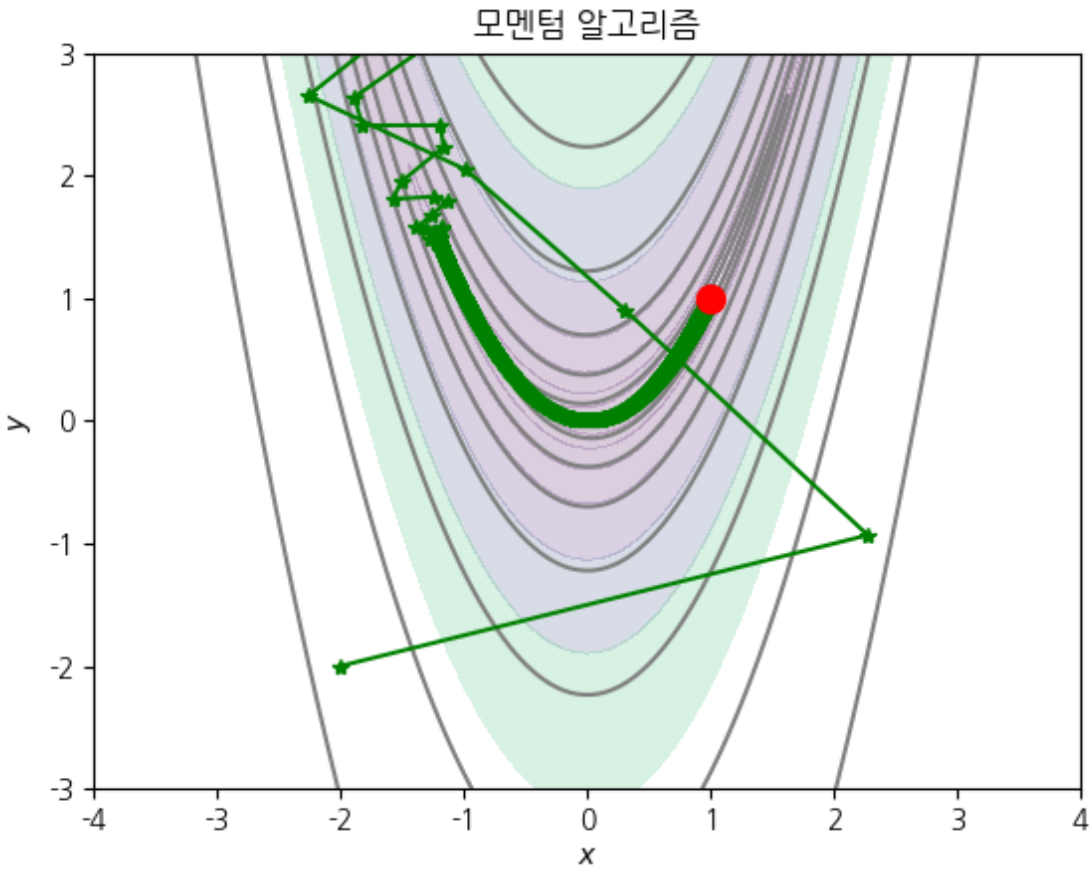
```
xx = np.linspace(-4, 4, 800)
yy = np.linspace(-3, 3, 600)
X, Y = np.meshgrid(xx, yy)
Z = f2(X, Y)

levels = np.logspace(-1, 3, 10)
plt.contourf(X, Y, Z, alpha=0.2, levels=levels)
plt.contour(X, Y, Z, colors="gray",
            levels=[0.4, 3, 15, 50, 150, 500, 1500, 5000])
plt.plot(XY[:, 0], XY[:, 1], 'g*-')
plt.plot(1, 1, 'ro', markersize=10)

plt.xlim(-4, 4)
plt.ylim(-3, 3)
plt.xticks(np.linspace(-4, 4, 9))
plt.yticks(np.linspace(-3, 3, 7))
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.title("2차원 로젠브록 함수 $f(x,y)$")
plt.show()
```

그래프보다 확실하게 로젠브록 함수를 표현하기 위하여 3d로도 표현했다.

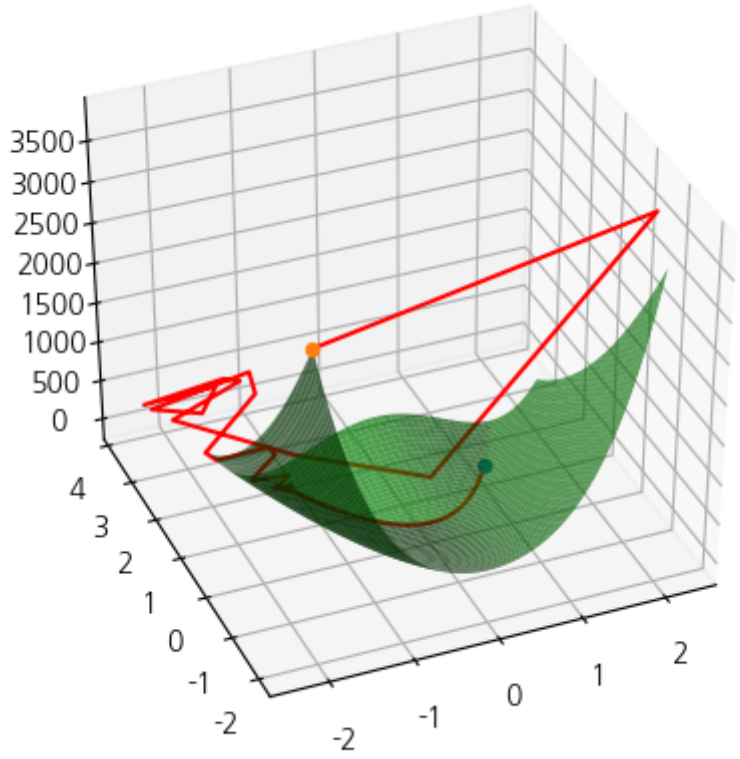
구현 코드는 다음과 같다.

```
x = np.linspace(-2, 2, 200)
y = np.linspace(-2, 2, 200)
X, Y = np.meshgrid(x, y)
Z = f2(X, Y)
fig = plt.figure(figsize=(5, 5))
ax = fig.gca(projection='3d')
ax.view_init(45, 230)
ax.plot_surface(X, Y, Z, color='g', alpha=0.7)
ax.scatter(1, 1, 0, 'ro')
ax.scatter(-2, -2, f2(-2, -2), 'ro')
ax.plot(XY[:, 0], XY[:, 1], [f2(x, y) for x, y in XY], color='r')
plt.show()
```

실행 결과



코드에 나타낸 것처럼 (-2, -2) 좌표에서 시작하여 1, 1 좌표로 진동하며 수렴하는 것을 알 수 있다.



3d로 표현한 그림에서도 동일한 결과를 나타냈지만 조금 더 쉽게 결과를 확인할 수 있었다.

단위 테스트

이러한 모멘텀 알고리즘을 함수로 묶어 단위 테스트를 실행하였다.

```
import unittest


class test(unittest.TestCase):
    def testMomentum(self):
        momentum()


if __name__ == '__main__':
    unittest.main()
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.069s

OK
```

이상 없이 실행되는 모습이다.

선택한 최적화 알고리즘의 구체화

위에서 실행하였던 코드를 클래스와 함수로 모듈화를 해서 객체지향적인 코드를 구현하였다.

```
class optimization:
    def __init__(self, start_x, start_y, i_max, alpha, eps, m):
        self.start_x = start_x
        self.start_y = start_y
        self.i_max = i_max
        self.alpha = alpha
        self.eps = eps
        self.m = m

    def f2(self, x, y):
        # 2차원 로젠브록 함수
        return (1 - x) ** 2 + 100.0 * (y - x ** 2) ** 2

    def f2g(self, x, y):
        # 2차원 로젠브록 함수의 도함수
        return np.array(
            (2.0 * (x - 1) - 400.0 * x * (y - x ** 2), 200.0 * (y - x ** 2)))

    def rosenbrock_momentum(self):
        w_init = [self.start_x, self.start_y]
        w_i = np.zeros([self.i_max, 2])
        w_i[0, :] = w_init
        x = w_init[0]
        y = w_init[1]
        vx = 0
        vy = 0
        XY = list()
        XY.append([x, y])
        cnt = 0
        for i in range(1, self.i_max):
            fdx = self.f2g(x, y)
            vx = self.m * vx - self.alpha * fdx[0]
            vy = self.m * vy - self.alpha * fdx[1]
            x += vx
            y += vy
            cnt = i
            w_i[i, 0] = x
            w_i[i, 1] = y
            XY.append([x, y])
            if self.f2(x, y) < self.eps:
                break
        print(cnt)
        print(XY)
        return np.array(XY)

    def show_contour(self):
        XY = self.rosenbrock_momentum()
        xx = np.linspace(-4, 4, 800)
        yy = np.linspace(-3, 3, 600)
        X, Y = np.meshgrid(xx, yy)
        Z = self.f2(X, Y)

        levels = np.logspace(-1, 3, 10)
        plt.contourf(X, Y, Z, alpha=0.2, levels=levels)
        plt.contour(X,
                    Y,
                    Z,
```

```
                    colors="gray",
                    levels=[0.4, 3, 15, 50, 150, 500, 1500, 5000])
        plt.plot(XY[:, 0], XY[:, 1], 'g.-')
        plt.plot(1, 1, 'ro', markersize=10)

        plt.xlim(-4, 4)
        plt.ylim(-3, 3)
        plt.xticks(np.linspace(-4, 4, 9))
        plt.yticks(np.linspace(-3, 3, 7))
        plt.xlabel("$x$")
        plt.ylabel("$y$")
        plt.title("2차원 로젠브록 함수 $f(x,y)$")
        plt.show()

    def show_3d(self):
        XY = self.rosenbrock_momentum()
        x = np.linspace(-2, 2, 200)
        y = np.linspace(-2, 2, 200)
        X, Y = np.meshgrid(x, y)
        Z = self.f2(X, Y)
        fig = plt.figure(figsize=(5, 5))
        ax = fig.gca(projection='3d')
        ax.view_init(45, 230)
        ax.plot_surface(X, Y, Z, color='g', alpha=0.7)
        ax.scatter(1, 1, 0, 'ro')
        ax.scatter(self.start_x, self.start_y,
                   self.f2(self.start_x, self.start_y), 'ro')
        ax.plot(XY[:, 0], XY[:, 1], [self.f2(x, y) for x, y in XY], color='r')
        plt.show()


start_x = float(input("시작점 x값 : "))
start_y = float(input("시작점 y값 : "))
i_max = int(input("실행 횟수 : "))
alpha = float(input("학습률 : "))
eps = float(input("오차율 : "))
m = float(input("모멘텀 계수 : "))

a = optimization(start_x, start_y, i_max, alpha, eps, m)
a.show_contour()
a.show_3d()
```

사용자는 시작점 x값, 시작점 y값, 실행 횟수, 학습률, 오차율, 모멘텀 계수를 코드 실행 시 입력하여 원하는 값을 지정하여 실행하고 그래프로 나타낼 수 있다.

각 구체화 모듈의 단위 테스트 작업

위에서 실행한 코드들을 함수별로 나누어 단위 테스트를 실행하였다.

구현 코드는 아래와 같다.

```
import unittest
from pro import optimization
```

```
class testoptimization(unittest.TestCase):
    def setUp(self):
        self.optimization = optimization(-2, -2, 10000, 0.00089, 0.000001,
0.7)

    def testF2(self):
        self.optimization.f2(-2, -2)

    def testF2g(self):
        self.optimization.f2g(-2, -2)

    def testRosenbrock(self):
        self.optimization.rosenbrock_momentum()


if __name__ == '__main__':
    unittest.main()
```

```
----------------------------------------------------------------------
Ran 3 tests in 0.068s

OK
```

위처럼 이상 없이 실행되는 모습을 볼 수 있었다.

선택한 최적화 알고리즘의 검증

모멘텀 알고리즘을 실행하면

```
0.9990012135481801 0.9979994255242441
5174
```

(0.9990012135481801, 0.9979994255242441) 의 값으로 찾아가 오차율보다 작아져 해당 값에서 반복을 멈췄다.

총 5174번만에 근사치를 찾아내었고, 아래와 같은 좌표를 지나게 되었다.

```
[[-2, -2], [2.2773399999999997, -0.9320000000000002], [0.3089146374432872,
0.9046533906567995]
...
[0.998998832235029, 0.9979946581199199], [0.9990000236015911,
0.9979970432420655], [0.9990012135481801, 0.9979994255242441]]
```

구현 소감

솔직한 심정으로는 신기했다. 이러한 식을 통해서 인간처럼 학습 능력이 없는 컴퓨터가 스스로 근사치를 찾아낸다는 사실이 놀라울 따름이었다. 이러한 방법으로 알파고와 같은 인공지능들이 학습되는 원리에 대해 알게 되었다.

이범석

국민대학교 소프트웨어학부, 20171664

Mobile 010-6401-6042

qpwoeiru6486@gmail.com

ijkoo16@kookmin.ac.kr