Homework #3

Implement Nelder-Mead Method and Powell's Method & Disscusion

GIST AI

20231126 황산하

*코드는 Python으로 작성했고 Report 이외에 스크립트도 Report와 함께 Zip으로 압축하여 제출하였습니다.

INDEX

1. Comparison and Result

이번 과제에서는 다음과 같은 목표 함수들이 주어졌다. 이번 과제의 Nelder-Mead 방법과 Powell's 방법은 주어진 다변수 함수들에 대해서 Local minimum을 찾는 알고리즘이다.

$$\mathrm{F}_1(x,y) = (x+3y-5)^2 + (3x+y-7)^2$$

$$\mathrm{F}_2(x,y) = 50(x-y^2)^2 + (1-y)^2$$

$$\mathrm{F}_3(x,y) = (1.5-x+xy)^2 + (2.25-x+xy^2)^2 + (2.625-x+xy^3)^2$$

이번 과제도 엄밀한 비교를 위해서 최대한 하이퍼파라미터는 고정을 하고 실험을 진행했다. Nelder-Mead에만 필요하고 Powell에는 안필요한 파라미터는 어쩔 수 없지만, 종료조건의 tol같은 경우는 epsilon을 모두 0.01, 0.001, 0.0001 세가지 경우로 세팅해서 비교해 봤는데 생각보다 수렴이 빨리 되는 것 같아서 0.01의 경우만 가지고 비교했다. Plot 함수로 만들어서 등고선도 그려봤는데 이는 Appendix 부분에 추가하도록 하겠다. 그리고 결과가 돌릴때마다 달라지기 때문에 random seed를 777로 고정하고 실험을 진행했다. 초기치는 둘다 -1000에서 1000에서 랜덤 샘플링해서 알고리즘을 구현했다.

[표 1 Nelder-Mead Method결과표]

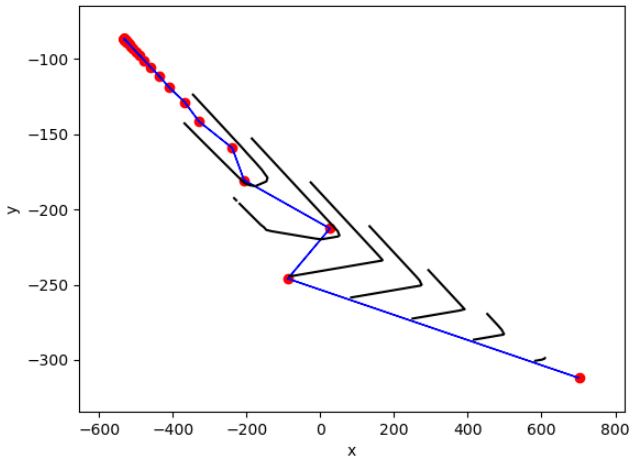
Nelder-Mead Method	(x, y)	Iter (Time)	Convergence	f(x,y)
F1	(139.376, -316.507)	48 iter (0.0081sec)	Yes	673414.209
F2	(-530.999, -87.000)	102 iter (0.0160sec)	Yes	3280507744.033
F3	(-309.5, -166.5)	33 iter (0.0060 sec)	Yes	2.040904055e+18

1) Nelder-Mead Method (NM)

Nelder-Mead Method는 polygon을 형성하여 gradient 기반으로 선택된 다음 값을 탐색한 후에 다음 값의 결과에 따라 확장과 축소를 반복하며 함수의 local minimum을 찾는 방법이다. 왜 굳이 polygon으로 찾을까 궁금했는데, 이전 값을 저장함으로써 지금 얻은 값이 좋은지 확인이 용이해지고 이를 polygon으로 표현한 거 같았다. 다변수 함수를 탐색함에도 불구하고 (변수가 하나만 늘긴 했지만) 이전 과제에서 탐색했던 단변수함수를 탐색하는 비용과 크게 차이가 나지 않았고 구현 자체도 직관적으로 느껴지는 알고리즘이었다.

다만 단점이 있다면 현재 구현에서 초기치를 랜덤으로 주도록 설정하고 과제 제출을 위해 random seed로 고정해놨는데, 초기치의 설정에 따라서 결과가 꽤 많이 달라지는 것을 확인했다. 현재의 결과에는 local minimum이 local minimum이 아닌듯 값에 매우 큰 것을 볼 수 있는데, 여러 실험의 결과 2자리 수 결과를 보기도 했고, 10만번의 iteration을 반복했음에도 수렴하지 않은 경우도 있었다. 따라서 NM 방법은 초기치 설정을 잘 해주는 것이 정말 중요하다고 느꼈다. 또한

하이퍼 파라미터가 여러가지이기 때문에 (알파, 감마, 베타) 이 파라미터들을 어떻게 설정해줄지 고민하는 과정도 유의미하다고 생각했다. 어떤 세팅이 최선의 결과를 가져올지 탐색하는 알고리즘이 있다면 NM방법만으로도 충분히 강력한 성능을 낼 수 있을것이라고 생각했다.



[그림 1] F2 함수에 대한 NM 방법의 Contour Plot,
오른쪽 하단에서 시작해서 왼쪽 상단으로 수렴하였다.

[표 2 Powell's Method 결과표]

Powell's Method	(x, y)	Iter (Time)	Convergence	f(x,y)
F1	(-13.239, -15.71)	44 iter (0.0089sec)	Yes	8173.2138
F2	(8010.18, -89.51)	32 iter (0.007sec)	Yes	8266.5183
F3	(-273.05, 1.003)	18 iter (0.003sec)	Yes	0.457635

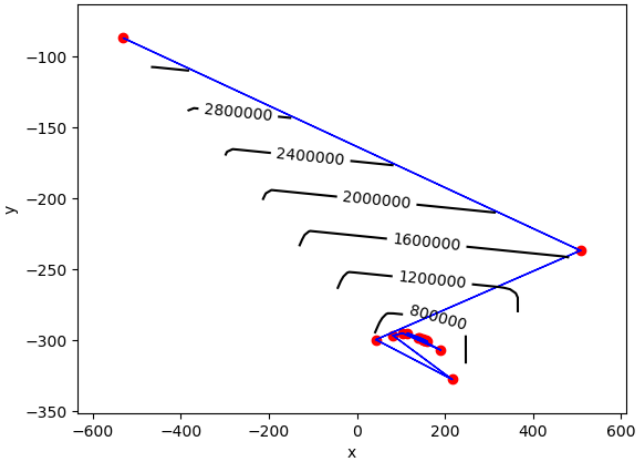
2) Powell's Method (PW)

Powell's Method는 NM방법보다도 더 직관적인 방법이라고 생각한다. 이의 이전 버전의 방법인 Univariate search 방법의 경우도 하나의 변수를 고정해 놓고, 해당 변수의 minimum value를 탐색하는 방법인데, PW는 더 나아가 방향이라는 변수를 이용해서 더 빠르게 local minimum을 향해서 수렴할 수 있다. 초기 방향은 x,y 두 가지 변수만 존재하기 때문에 표준유닛벡터인 (1,0)과 (0,1)을 이용했다. 이번에도 random 초기 변수를 이용해서 초기 시작점을 정했고 이는 추후에 랜덤시드를 통해 고정되었다.

주목할 만한 점은 아까 NM을 이야기하면서 hyper parameter에 대한 탐색이 있으면 좋겠다고 언급했는데, PW에서는 gamma를 탐색하는 알고리즘이 존재한다는 것이다. 나는 이를 처음에 과제

2에서 구현했던 golden section 방법을 이용했으나, 발산하길래 조금 더 알아보니 Univariate search를 통해 golden section의 초기 boundary를 결정함으로써 안정화를 시킬 수 있다는걸 알 수 있었다. 따라서 Golden section 방법과 Univariate search통해 gamma를 탐색하고 이를 통해 새로운 x와 y를 구함으로써 local minimum을 탐색했다.

PW방법은 NM과 비교했을 때 훨씬 빠르고, 물론 이 역시도 랜덤 초기치를 주었기 때문에 조금은 더 발전 할 수 있는 여지가 있지만 조금 더 좋은 local minimum을 찾아가는 경향을 보였다. 특히 F3같은 경우는 NM은 거의 발산했다고 봐도 좋을 정도로 높은 값에 수렴했는데, PW는 0.45라는 거의 가장 작을 수 있는 값에 수렴이 가능했다.

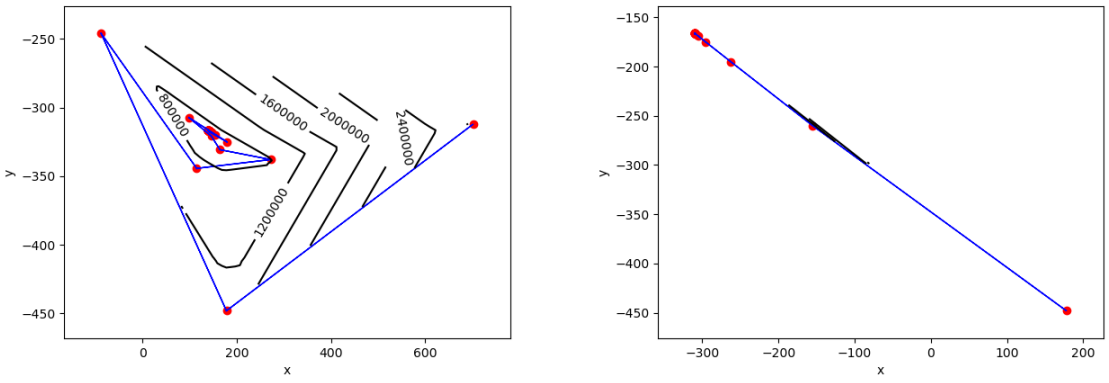


[그림 2] F1 함수에 대한 PW 방법의 Contour Plot,
왼쪽 상단에서 시작해서 오른쪽 하단으로 수렴하였다.

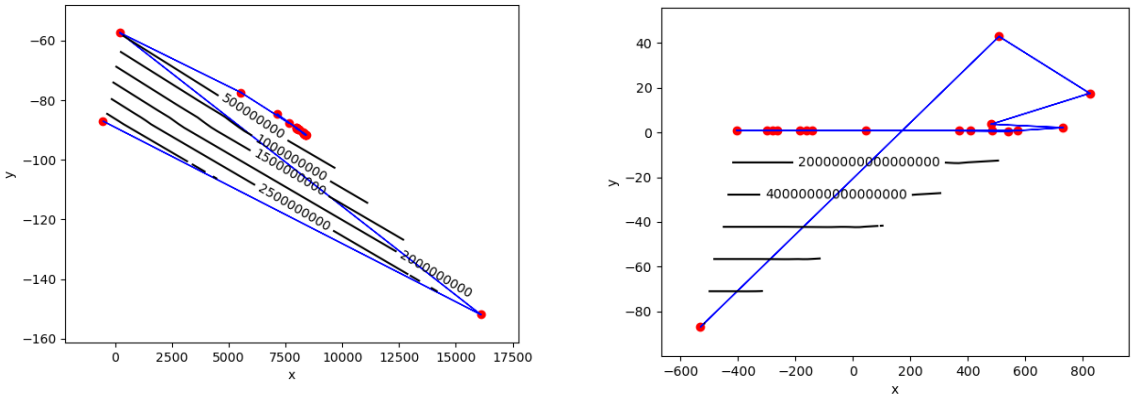
2. Appendix A : Code

Code는 Python으로 작성했으며 제출할 폴더를 열어보면, algo.py와 result.py라는 script가 작성되어있다. algo.py에는 Nelder-mead 방법과 Powell's 방법의 알고리즘이 작성되어 있고, result.py를 실행시키면 결과를 받아볼 수 있다. result.py 안에는 과제에 이용한 함수 3개와 결과를 받아보기 위한 시각화 함수가 작성되어 있다. 따라서 terminal에서 해당 폴더의 디렉토리로 이동한 후에 "python result.py --method="Nelder_Mead" --f=f1"와 같은 형식으로 실행시킬 수 있다. --Method 부분에는 "Nelder_Mead" 혹은 "powell" 을 넣어 확인할 수 있으며, --f에는 f1, f2, f3 중에 선택해서 결과를 받아 볼 수 있다. 혹시 seed를 바꿔서 결과를 보고 싶다면 --random_seed={숫자}를 넣어서 스크립트를 돌리면 된다. (만약 작동하지 않는 것 같다면 result.py의 56번 라인의 하이퍼 파라미터를 바꾸면 된다.) 이번에도 역시 가독성을 위해 코드는 보고서와 함께 따로 첨부했다.

3. Appendix B : Some Result



[그림 3,4] Nelder-Mead 방법으로 search한 그림, (좌)f1 (우)f3



[그림 5,6] Powell's 방법으로 search한 그림, (좌)f1 (우)f3

```
iter   - 15, (x, y) : (13857.726089900041, -117.7194660769162), fxy : 14095.386255399815

iter   - 16, (x, y) : (13858.932884250322, -117.72401336136444), fxy : 14095.396795862991

iter   - 17, (x, y) : (13858.059626203545, -117.72070217524622), fxy : 14095.146907495893

iter   - 18, (x, y) : (13858.16090874357, -117.72107753222316), fxy : 14095.110000250763

iter   - 19, (x, y) : (13858.386473225064, -117.72192798010484), fxy : 14095.113021621446

Converged after 19 iterations.
Final result : (x, y) : (13858.386473225064, -117.72192798010484), fxy : 14095.1130216214
, time : 0.0039997100830078125
```

[그림 7] Powell's 방법 결과 예시

```
iter  - 95, (x, y) : (-530.9999999938145, -87.00000000161397), fxy : 3280507744.222463

iter  - 96, (x, y) : (-530.9999999952896, -87.00000000122907), fxy : 3280507744.1694093

iter  - 97, (x, y) : (-530.999999996413, -87.00000000093596), fxy : 3280507744.129008

iter  - 98, (x, y) : (-530.9999999972684, -87.00000000071276), fxy : 3280507744.0982447

iter  - 99, (x, y) : (-530.9999999979198, -87.00000000054277), fxy : 3280507744.0748134

iter  - 100, (x, y) : (-530.9999999984159, -87.00000000041335), fxy : 3280507744.0569744

iter  - 101, (x, y) : (-530.9999999987937, -87.00000000031477), fxy : 3280507744.0433865

iter  - 102, (x, y) : (-530.9999999990813, -87.00000000023971), fxy : 3280507744.0330405

Converged after 103 iterations.
Final result : (x, y) : (-530.9999999993004, -87.00000000018252), fxy : 3280507744.025159
, time : 0.018001794815063477
```

[그림 8] Nelder-Mead 방법 결과 예시