Homework #6

Implement Gauss-Newton's and LM

GIST AI

20231126 황산하

*코드는 Python으로 작성했고 Report와 함께 Zip으로 압축하여 제출하였습니다.

1. Implement and Result

본 과제는 최소제곱법을 이용해서 최적화하는 방법인 Gauss-Newton Method (GN)와 Levenberg-Marquardt Method (LM)을 주어진 데이터를 이용하여 구현하고, 주어진 모델에 대하여 최적의 파라미터 a, b, c, d 를 찾는 과제이다. 주어진 모델들은 다음과 같다.

$$f1 = ax + by + cz + d$$

$$f2 = \exp\left(-\frac{[(x-a)^2+(y-b)^2+(z-c)^2]}{d^2}\right)$$

위의 f1와 f2가 주어진 모델인데 x,y,z는 주어진 데이터를 이용하고, 임의의 P, 즉 a,b,c,d,를 찾으면 되는 과제이다. 주어진 데이터 Model1과 Model2에는 data1과 data2라는 데이터가 있는데 이는 해당 모델들의 각각 다른 정답 데이터로써, 알고리즘을 구현하고 data값을 다르게 하여 다른 함수를 적합시킨것처럼 실험해볼 수 있었다. Max iter 는 20으로 각각 고정시키고 각 데이터와 모델에 대해서 적합시켜 보았다. 초기 P는 각각 [1,1,1,1]로 주었고 lambda 값이 들어가는 LM에는 초기 lambda를 0.1로 주었다.

1) Gauss-Newton Method (GN)

[표 1 GN 결과표]

F1	Data	rmse	Convergence	Final P [a,b,c,d]
Model 1	Data1	0.18893	Yes	[1.036, -1.03, -0.013, 3.01]
	Data2	0.489	Yes	[1.019, -1.012, 0.009, 3.036]
Model 2	Data1	0.112895	Yes	[0.0949, -0.128, 0.0002, 0.561]
	Data2	0.152867	Yes	[0.091, -0.1428, 0.0062, 0.5681]
F2	Data	rmse	Convergence	Final P [a,b,c,d]
Model 1	Data1	0.1889	Yes	[1.036, -1.03, -0.0138, 3.0195]
	Data2*	0.768	Yes	[-3.3377, 3.443, -0.1648, 5.074]
Model 2	Data1	0.11289	Yes	[0.09, -0.1286, 0.0002, 0.56]
	Data2*	0.0.787	Yes	[-2.679, 1.669, -1.302, -5.7471]

결과를 보면 알 수 있듯이 전반적으로 다 잘 수렴했다. 표에 (*) 표시를 해놓은 부분은 수렴하는 과정이 잘 나타나있는 결과에 표시를 해놓은 것이었는데, 왜냐하면 다른 데이터세팅에서는 한번 반복만에 수렴해버려서 알고리즘이 잘 작동하는지 알기 어려웠기 때문이다. 다른 Newton 방법들과 마찬가지로 수렴속도도 굉장히 빠른 강력한 알고리즘으로 보였다.

GN은 뉴턴 방법의 작동과 거의 유사한데, Jaccobian 행렬을 구하고 이를 통해 최적의 방향을 결정하는 과정을 통해 빠른 수렴을 달성할 수 있다. Jaccobian 행렬을 구하는 이유는 다변수이기 때문이고, 원래 미분값을 통해 최적값을 찾아가는 뉴턴과 동일하다고 생각하니 구현하기 쉬웠다.

2) Levenberg-Marquardt Method (LM)

[표 2 LM결과표]

F1	Data	rmse	Convergence	Final P [a,b,c,d]
Model1	Data1	0.18893	Yes	[1.036, -1.03, -0.0138, 3.02]
	Data2*	0.4890	Yes	[1.019, -1.012, 0.009, 3.036]
Model2	Data1	0.112895	Yes	[0.095, -0.129, 0.0003, 0.06]
	Data2	0.152867	Yes	[0.091, -0.1428, 0.006, 0.568]
F2	Data	rmse	Convergence	Final P [a,b,c,d]
Model1	Data1*	0.63942	Yes	[-3.242, 3.446, -0.152, 5.03]
	Data2*	0.76813	Yes	[-3.338, 3.443, -0.165, 5.075]
Model2	Data1	1.1803	Yes	[0.303, 1.41, 0.762, 3.8]
	Data2*	1.1832	Yes	[0.3181, 1.442, 0.7565, 3.824]

LM 방법도 GN 못지않게 매우 빠르고 잘 수렴하는 모습을 보였다. 다만 잘 수렴하다가 몇몇 경우에서 발산 할 때도 있었는데 이는 종료 조건을 잘 설정하면 해결될 문제로 보인다. 잘 수렴하고 종료가 안되는 경우에 발산하는 거라서 종료 조건이 중요해 보였다. 해당 방법은 사실상 GN방법과 앞서 배웠던 gradient 방식을 결합한 방법으로서 안정적으로 해를 찾을 수 있다는 장점이 있는데 지역 해 (local minimum)에서 발산하는 경우가 생기는 건 조금 모순이라고 생각했다.

LM 방법에서는 GN과 Gradient Descent 방식을 결합하기 위해 lambda라는 파라미터를 설정했는데, 이를 찾는 것 역시 성능에 꽤 큰 영향을 미쳤다. 업데이트 이후에 구할 수 있는 P로 구한 새로운 Jaccobian을 이용해서 기존 행렬 행렬식(determinant)보다 크면 lambda에 10을 곱해 gradient 방식처럼 동작하게 하고 , 작으면 업데이트하고 1/10을 곱해 GN방식대로 동작하게끔 설정을 해주었다.

Appendix A : Code

코드는 Python으로 구현했으며 제출될 폴더 안에는 method.py, result.py, utils.py가 있다. 이전 과제와 마찬가지로 utils.py에는 알고리즘 구현을 위해 필요한 gradient를 구하는 함수와 Jaccobian, delete_nan함수가 구현되어있다. delete_nan 같은 경우는 초기 데이터를 받아올 때 pandas read_table에서 첫 행이 비어있어서 nan으로 인식되므로 전처리를 하는 코드이다. Method.py안에 각각 GN과 LM 알고리즘이 구현되어 있다. Result.py를 실행 시킴으로써 결과를 받아 볼 수 있는다. 예를 들어 python result.py로 돌릴 수 있고 result.py의 main함수 안에 하이퍼 파라미터를 조정하여 각각의 결과를 받아볼 수 있다. 마지막 파라미터에 숫자 1,2를 선택하게끔 만들었는데 이는 각각 다른 target 값의 변수라고 보면 된다.

Appendix B : Some Results

```
iter :  10 rmse :  1868.982514737352 time :  0.05700230598449707
iter :  11 rmse :  740.1696775587553 time :  0.06199979782104492
iter :  12 rmse :  296.4314326900641 time :  0.0680077075958252
iter :  13 rmse :  120.42012449643245 time :  0.07400846481323242
iter :  14 rmse :  49.74270497727892 time :  0.07951927185058594
iter :  15 rmse :  20.903171585577404 time :  0.08396625518798828
iter :  16 rmse :  8.948870824894819 time :  0.08848929405212402
iter :  17 rmse :  4.037388626151449 time :  0.0934913158416748
iter :  18 rmse :  2.19022425372901 time :  0.09748959541320801
iter :  19 rmse :  1.4847052887439045 time :  0.10449051856994629
iter :  20 rmse :  1.2099254148282526 time :  0.11249017715454102
iter :  21 rmse :  0.8241614792527581 time :  0.11849093437194824
iter :  22 rmse :  0.7682856646835646 time :  0.1254899501800537
iter :  23 rmse :  0.7681422389920962 time :  0.1324923038482666
iter :  24 rmse :  0.7681375106016499 time :  0.13749194145202637
iter :  25 rmse :  0.7681367907126908 time :  0.14349007606506348
iter :  26 rmse :  0.768136682218502 time :  0.14748930931091309
iter :  27 rmse :  0.7681366656275601 time :  0.15248942375183105
iter :  28 rmse :  0.7681366631455607 time :  0.15749239921569824
iter :  29 rmse :  0.7681366627549724 time :  0.16349434852600098
final P :  [-3.33775151  3.44308475 -0.16489297  5.07474748]

(test) C:\Users\sanha\NOPT2023\assignment6>
```

[그림 1] GN을 적용한 것 중 잘 수렴하는 것을 보인 하나의 결과

```
iter :  8 rmse :  11681.505236709081 time :  0.12451863288879395
iter :  9 rmse :  4693.442163597654 time :  0.1445178985595703
iter :  10 rmse :  1862.458957880705 time :  0.15951895713806152
iter :  11 rmse :  738.8966719979077 time :  0.17551946640014648
iter :  12 rmse :  295.9876888256693 time :  0.18752598762512207
iter :  13 rmse :  120.24541371056674 time :  0.20458769798278809
iter :  14 rmse :  49.672030107452976 time :  0.21858859062194824
iter :  15 rmse :  20.87404284893727 time :  0.23258709907531738
iter :  16 rmse :  8.936753868010076 time :  0.24858856201171875
iter :  17 rmse :  4.032581640793589 time :  0.26558804512023926
iter :  18 rmse :  2.1885143505202085 time :  0.28159618377685547
iter :  19 rmse :  1.4839476658990174 time :  0.3001127243041992
iter :  20 rmse :  1.2098395669181792 time :  0.31610846519470215
iter :  21 rmse :  0.823633979965158 time :  0.33110952377319336
iter :  22 rmse :  0.7682838463673741 time :  0.347109317779541
iter :  23 rmse :  0.768142170187911 time :  0.3671109676361084
iter :  24 rmse :  0.7681375000855345 time :  0.383115291595459
iter :  25 rmse :  0.7681367891257486 time :  0.39862680435180664
iter :  26 rmse :  0.7681366819766328 time :  0.4126267433166504
iter :  27 rmse :  0.7681366655910257 time :  0.4296271800994873
iter :  28 rmse :  0.7681366631399507 time :  0.44463086128234863
iter :  29 rmse :  0.7681366627541449 time :  0.4616274833679199
final P :  [-3.33775127  3.44308452 -0.16489299  5.07474722]
```

[그림 2] LM을 적용한 것 중 잘 수렴하는 것을 보인 하나의 결과