Draft version December 11, 2024
Typeset using LaTeX **preprint** style in AASTeX631

An Alternate Method for Minimizing χ^2

Jennifer C. Yee and Andrew P. Gould

ABSTRACT

In this paper, we describe an algorithm and associated software package (`sfit_minimize`) for maximizing the likelihood function of a set of parameters by minimizing χ^2. The key element of this method is that the algorithm estimates the second derivative of the χ^2 function using first derivatives of the function to be fitted. These same derivatives can also be used to calculate the uncertainties in each parameter. We test this algorithm against several standard minimization algorithms in `SciPy.optimize.minimize()` by fitting point lens models to light curves from the 2018 Korea Microlensing Telescope Network event database. We show that for fitting microlensing events, `SFit` works faster than the `Nelder-Mead` simplex method and is more reliable than the `BFGS` gradient method.

1. THE DERIVATION

Our method builds upon the discussion in "χ^2 and Linear Fits" (Gould 2003), which noted that the approach to non-linear models could be expanded beyond the scope of that work. Suppose that the function we want to minimize is a function $F(x)$ that is described by n parameters A_i (where we use A_i (instead of a_i) as a reminder that in the general case, they are non-linear). Considering the general (nonlinear) case, we can Taylor expand χ^2, in terms of the n parameters:

$$\chi^2 = \chi^2_0 + \sum_i \frac{\partial \chi^2}{\partial A_i} A_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 \chi^2}{\partial A_i \partial A_j} A_i A_j \tag{1}$$

$$= \chi^2_0 + \sum_i D_i * A_i + \sum_{i,j} B_{ij} * A_i A_j + ... \quad , \tag{2}$$

where

$$D_i \equiv \frac{\partial \chi^2}{\partial A_i} \tag{3}$$

$$B_{ij} \equiv (1/2) \frac{\partial^2 \chi^2}{\partial A_i \partial A_j} \quad . \tag{4}$$

Then,

$$\frac{\partial \chi^2}{\partial A_i} = -2 \sum_k \frac{(y_k - F(x_k))}{\sigma_k^2} \frac{\partial F(x_k)}{\partial A_i} \tag{5}$$

and

$$\frac{\partial^2 \chi^2}{\partial A_i \partial A_j} = -2 \sum_k \left[\frac{1}{\sigma_k^2} \frac{\partial F(x_k)}{\partial A_i} \frac{\partial F(x_k)}{\partial A_j} + \frac{(y_k - F(x_k))}{\sigma_k^2} \frac{\partial^2 F(x_k)}{\partial A_i \partial A_j} \right] \quad . \tag{6}$$

In the special case of a linear function, $F(x) = \sum_i a_i f_i(x)$ then

$$\frac{\partial F(x)}{\partial a_i} = f_i(x) \quad \text{and} \quad \frac{\partial^2 F(x)}{\partial a_i \partial a_j} = 0, \tag{7}$$

so the second term disappears, and we find that the solution (derived from second derivative of χ^2) can be expressed in terms of products of the FIRST derivatives of the general functional form. For the general case, we simply make the approximation that the second derivative term can be neglected; i.e.,

$$\frac{\partial^2 \chi^2}{\partial A_i \partial A_j} \approx -2 \sum_k \frac{1}{\sigma_k^2} \frac{\partial F(x_k)}{\partial A_i} \frac{\partial F(x_k)}{\partial A_j} \quad . \tag{8}$$

Hence, there are three ways to generalize Newton's method (actually discovered by Simpson) to multiple dimensions:

1. Use only first derivatives of the χ^2 function (which is what Simpson did in 1-D), the so-called gradient method.
2. Taylor expand χ^2 and truncate at second term, then solve this (very inexact equation) exactly by inversion of the matrix of second derivatives (Hessian).
3. First generalize Simpson's idea that a 1-D function is well described by its first derivative (which can easily be solved exactly) to several dimensions (i.e., assume the function is well described by a tangent plane) and solve this exactly, as is done here.

Because first derivatives are more stable than second derivatives, this algorithm could potentially be significantly more stable for situations in which the derivatives are derived numerically, which we investigate in the next section.

2. IMPLEMENTATION

We have implemented the above algorithm in the `sfit_minimizer` package. The goal was to make the calling sequence similar to that of `SciPy.optimize.minimize()`:

```
result = sfit_minimizer.minimize(my_func, x0=initial_guess)        (9)
```

where `my_func` is an object of the type `sfit_minimizer.SFitFunction()` in which the user defines either the model, $F(x_k)$ or the residual $y_k - F(x_k)$ calculation (i.e., the method `my_func.model()` or `my_func.residuals()`) and the partial derivatives of the function to be minimized, $\partial F(x_k)/\partial A_i$ (i.e., the method `my_func.df()`). The package includes a simple example (`example_00_linear_fit.py`) for fitting a linear model to demonstrate this usage.

The `sfit_minimizer.SFitFunction()` class contains methods that use the partial derivative function to calculate the next step from the D_i and B_{ij} following the method in Gould (2003) for linear functions. That is, D_i and B_{ij} are calculated from Equations 3 and 4, respectively. Then, the step size for each parameter, Δ_i, is

$$\Delta_i = \sum_j C_{ij} D_j \quad \text{where} \quad C \equiv B^{-1} \quad , \tag{10}$$

which is returned by `sfit_minimizer.SFitFunction.get_step()`. The new value of A_i is calculated by `sfit_minimizer.minimize()` to be

$$A_i = A_{i,0} + \epsilon\Delta_i \quad . \tag{11}$$

In `sfit_minimizer.minimize()`, the user has the option to specify the value of ϵ or to make use of an adaptive step size, which starts at $\epsilon = 0.001$ and becomes larger as the minimum is approached.

Ultimately, `sfit_minimizer.minimize()` returns an `sfit_minimizer.SFitResults()` object that contains attributes similar to the object returned by `SciPy.optimize.minimize()`. These include the best-fit values of the parameters, `x`, and their uncertainties `sigma` (i.e., $\sigma_i = \sqrt{C_{ii}}$). For the rest of this paper, we will refer to our algorithm as `SFit` for brevity.

3. PERFORMANCE TEST

To test the performance of `SFit`, we use the package to fit point-source–point-lens models (Paczynski 1986) to a sample of microlensing events from the Korea Microlensing Telescope Network (KMTNet; Kim et al. 2016). For comparison, we also perform the fitting using the `Nelder-Mead` (Gao & Han 2012), `Newton-CG` (Nocedal & Wright 2006), and `BFGS` (Nocedal & Wright 2006) algorithms in `SciPy.optimize.minimize()`. The `Nelder-Mead` algorithm is a simplex algorithm, so it only relies on evaluating the χ^2. In contrast to our algorithm, the `Newton-CG` and `BFGS` algorithms use the jacobian of the likelihood function for the minimization. In all cases, we set `tol = 1e-5`.

We select our sample from microlensing events discovered in 2018 by KMTNet (Kim et al. 2018a,b,c). We use only "clear" microlensing events with reported fit parameters. We eliminate any events that were flagged as anomalous in the 2018 AnomalyFinder search (although possible finite source or buried host events were left in the sample; Gould et al. 2022; Jung et al. 2022). These cuts left 1822 events in the sample.

For this sample, we use the online, I-band, pySIS (Albrow et al. 2009) data from the KMTNet website (https://kmtnet.kasi.re.kr/ulens/). KMTNet takes data from three different sites and has multiple, sometimes overlapping, fields of observations. We treat data from different sites and different fields as separate datasets. For each dataset, we calculate the mean sky background and standard deviation as well as the mean and standard deviation of the full-width-at-half-max (FWHM) for each observation. We eliminate points with sky background more than 1 standard deviation above the mean or FWHM more than 3 standard deviations above the mean. This removes a large fraction of the outliers from the data.

We fit each event with a standard, point-lens model A, which is parameterized by the three Paczyński parameters: t_0, u_0, and $t_{\rm E}$ (for the definitions of these parameters see, e.g., Gaudi 2012). In addition, there are two flux parameters used to scale each dataset, k, to the model, A: $f_{{\rm mod},k} = f_{{\rm S},k}A + f_{{\rm B},k}$.

We use `MulensModel` (Poleski & Yee 2019) to calculate the model, its derivatives, and the jacobians. We perform a linear fit to the flux parameters at each iteration using built in functions from `MulensModel` and use the fitting algorithm to optimize t_0, u_0, and $t_{\rm E}$.

Before doing the fitting, we determined the starting values for t_0, u_0, and $t_{\rm E}$ using the following procedure. The KMTNet website reports an estimate for the values of these parameters. We use the value of t_0 as reported by KMTNet. To ensure an optimum starting point for the fits, we tested a series of values of $u_{0,i} = [0.01, 0.3, 0.7, 1.0, 1.5]$ and calculated $t_{{\rm E},i} \equiv t_{\rm E}u_0/u_{0,i}$, where u_0 and $t_{\rm E}$ are the values reported in the KMTNet table. We performed a linear fit of the flux parameters $(f_{\mathrm{S},k}, f_{\mathrm{B},k})$ for each set of parameters i and chose the values of $u_{0,i}$ and $t_{\mathrm{E},i}$ that produced the smallest χ^2 to initialize the fits.

We calculated several metrics to evaluate the performance of each algorithm. First, for a given event, we compared the χ^2 of the best-fit reported by each algorithm to the best (minimum) value reported out of the four fits. The results are given in Table 1 for several values of $\Delta\chi^2$ classified by whether or not the algorithm reported that the fit was successful ("reported success").

Each fit may be classified in one of four ways:

- True positives: both true and reported success,
- False positives: algorithm reported success, but did not find the minimum,
- True negatives: algorithm reported failure and did not find the minimum,
- False negatives: algorithm reported failure, but did find the minimum.

For the purpose of these comparisons, we define $\Delta\chi^2 < 0.1$ as a "true success." **REWRITTEN FROM HERE:** The results are visualized in Figure 1.

We also calculated the number of χ^2 function evaluations required by each algorithm for fitting each event. The maximum number allowed was 999; if the number of evaluations exceeded this, the fit was marked as a failure. Table 2 provides statistical summaries of this metric.

4. TEST RESULTS

Table 1 shows that `BFGS`, `Nelder-Mead`, and `SFit` all had very low false positive rates; i.e., when the algorithm reported success, it also tended to find the best-fit solution. `Nelder-Mead` produced the most true successes **85%** out of all the algorithms, although it also required the most function evaluations. `SFit` was the most reliable of the algorithms; in addition to the low false positive rate, it also had a low false negative rate. Of all the algorithms, `BFGS` was arguably the most successful, getting within $\Delta\chi^2 = 0.1$ of the minimum **98.5%** of the time, but it only reported success $\sim 60\%$ of the time (it had a high false negative rate).

The `Newton-CG` algorithm performed the least well in our tests. It had both an extremely high false positive rate (**65%**) and false negative rate (**66%**).

Figure 1 and the middle and bottom sections of Table 1 compare the performance of the other algorithms to that of `SFit`. The table shows that there is not complete overlap in performance between `SFit` and any other algorithm. Nearly all `BFGS` fits reached the minimum, but some were reported as successes and some as failures. The reported success of `BFGS` was uncorrelated with the performance of `SFit`; none of the four classifications for `BFGS` completely overlapped with an `SFit` classification. There were also a handful of `BFGS` fits that failed for which `SFit` succeeded. This was also true for the `Newton-CG` fits.

The vast majority of the `SFit` successes were also true successes for `Nelder-Mead`. In addition, `Nelder-Mead` successfully fit $\sim 40\%$ of events that `SFit` failed to fit. Nevertheless, there were a handful of `Nelder-Mead` failures that were successfully fit by `SFit`.

END REVISED TEXT

5. CONCLUSIONS

Given these tests, the best method for fitting microlensing events depends on the desired outcome. The `Nelder-Mead` algorithm produces the highest success rate **(85%)** with no false positives, but requires more function evaluations than the other algorithms. The most efficient method would be to simply use `BFGS` and accept whatever results it produced as the best-fit. Our tests show that this fit will be correct **99%** of the time. However, there is no way to identify the **1%** of failures because they will be mixed up with the **40%** of fits that are reported as failures but actually succeeded (false negatives).

To achieve a better balance of efficiency, accuracy, and reliability, we combine the `SFit` and `BFGS` algorithms, both of which have extremely low false positive rates with minimal function evaluations. Most (73%) events can be fit successfully with `SFit`. The remaining **27%** of events can be fit using `BFGS`, which will increase the number of true positives to 85%. Finally, the `BFGS` fit for last **15%** of events is probably correct, but this can be verified by re-fitting them using `SFit`, but initializing the fit with the best-fit parameters from the ("failed") `BFGS` fit.

In testing this procedure, we find that **94%** of the time it successfully finds the best-fit solution and reports success (true positives) with only **five** false positives. Of these false positives, **four** had identifiable light curve characteristics that could lead to a fit failure, and none of the algorithms performed well on the fifth. Of the remaining **6%** of events, for which this procedure reports a failed fit, **more than half** had identifiable light curve characteristics that could lead to a fit failure. Thus, the false negative rate is only **3%**.

ADD a concluding sentence?

The Python implementation of this algorithm, including its specific application to microlensing events, can be found on GitHub.

ACKNOWLEDGMENTS

We thank Radek Poleski and Keto Zhang for helpful discussions in the development of the code. J.C.Y. acknowledges support from U.S. NSF Grant No. AST-2108414. This research has made use of publicly available data (https://kmtnet.kasi.re.kr/ulens/) from the KMTNet system operated by the Korea Astronomy and Space Science Institute (KASI) at three host sites of CTIO in Chile, SAAO in South Africa, and SSO in Australia. Data transfer from the host site to KASI was supported by the Korea Research Environment Open NETwork (KREONET).

REFERENCES

Albrow, M. D., Horne, K., Bramich, D. M., et al. 2009, MNRAS, 397, 2099, doi: 10.1111/j.1365-2966.2009.15098.x

Gao, F., & Han, L. 2012, Computational Optimization and Applications, 51, 259, doi: 10.1007/s10589-010-9329-3

Gaudi, B. S. 2012, ARA&A, 50, 411, doi: 10.1146/annurev-astro-081811-125518

Gould, A. 2003, arXiv:astro-ph/0310577

Gould, A., Han, C., Zang, W., et al. 2022, A&A, 664, A13, doi: 10.1051/0004-6361/202243744

Jung, Y. K., Zang, W., Han, C., et al. 2022, AJ, 164, 262, doi: 10.3847/1538-3881/ac9c5c

Kim, D. J., Kim, H. W., Hwang, K. H., et al. 2018a, AJ, 155, 76, doi: 10.3847/1538-3881/aaa47b

Kim, H.-W., Hwang, K.-H., Kim, D.-J., et al. 2018b, ArXiv e-prints. https://arxiv.org/abs/1804.03352

Kim, H.-W., Hwang, K.-H., Shvartzvald, Y., et al. 2018c, arXiv e-prints, arXiv:1806.07545. https://arxiv.org/abs/1806.07545

Kim, S.-L., Lee, C.-U., Park, B.-G., et al. 2016, Journal of Korean Astronomical Society, 49, 37, doi: 10.5303/JKAS.2016.49.1.037

Nocedal, J., & Wright, S. J. 2006, Numerical Optimization (New York, NY: Springer New York), 135–163, doi: 10.1007/978-0-387-40065-5_6

Paczynski, B. 1986, ApJ, 304, 1, doi: 10.1086/164140

Poleski, R., & Yee, J. C. 2019, Astronomy and Computing, 26, 35, doi: 10.1016/j.ascom.2018.11.001

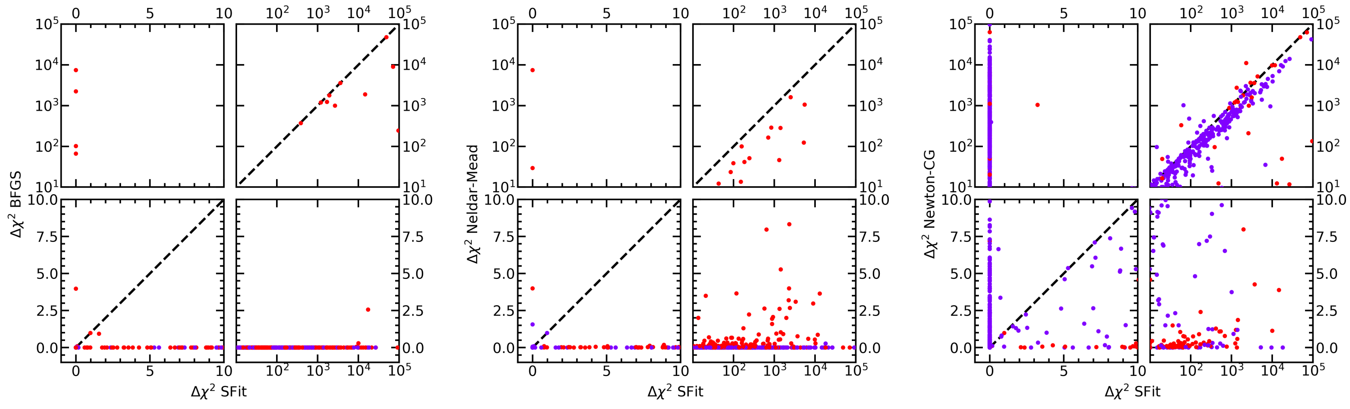

Figure 1. $\Delta\chi^2$ relative to the best-fit for `BFGS`, `Nelder-Mead`, and `Newton-CG` (the comparison algorithms) vs. `SFit`. Fits reported as successes by the comparison algorithm are plotted in purple, while those reported as failures are shown in red. Events that were fit successfully by both algorithms appear at (0, 0). In each set of four panels, the axes are split so that [0, 10] is on a linear scale, which [10, 10^5] is on a log scale. The vertical bands of points at $x = 0$ are fits that were successfully fit by `SFit` but failed to be fit by the comparison algorithm; purple points in those bands are false positives for the comparison algorithm. The horizontal bands of points at $y = 0$ are points for which the comparison algorithm successfully found the minimum but `SFit` did not; red points in those bands are false negatives.

Table 1. Number of Fits with $\Delta\chi^2 \geq X$ of the Best-Fit

Algorithm	Total	$\Delta\chi^2 <$ 0.1 N	0.1 %	1.0 N	1.0 %	10.0 N	10.0 %	100.0 N	100.0 %
All 1822 Events:									
Algorithm Reported Success:									
`BFGS`	1074	1074	100	1074	100	1074	100	1074	100
`Nelder-Mead`	1556	1551	100	1553	100	1554	100	1554	100
`Newton-CG`	1630	572	35	727	45	888	54	1101	68
`SFit`	1325	1324	100	1325	100	1325	100	1325	100
Algorithm Reported Failure:									
`BFGS`	748	722	97	725	97	727	97	728	97
`Nelder-Mead`	266	155	58	225	85	247	93	256	96
`Newton-CG`	192	65	34	111	58	125	65	137	71
`SFit`	497	1	0	7	1	54	11	255	51
1325 Events for which `SFit` reported success:									
Algorithm Reported Success:									
`BFGS`	864	864	100	864	100	864	100	864	100
`Nelder-Mead`	1321	1319	100	1320	100	1321	100	1321	100
`Newton-CG`	1265	536	42	675	53	781	62	880	70
Algorithm Reported Failure:									
`BFGS`	461	455	99	456	99	457	99	458	99
`Nelder-Mead`	4	1	25	1	25	2	50	3	75
`Newton-CG`	60	27	45	28	47	29	48	34	57
497 Events for which `SFit` reported failure:									
Algorithm Reported Success:									
`BFGS`	210	210	100	210	100	210	100	210	100
`Nelder-Mead`	235	232	99	233	99	233	99	233	99
`Newton-CG`	365	36	10	52	14	107	29	221	61
Algorithm Reported Failure:									
`BFGS`	287	267	93	269	94	270	94	270	94
`Nelder-Mead`	262	154	59	224	85	245	94	253	97
`Newton-CG`	132	38	29	83	63	96	73	103	78

Table 2. Number of χ^2 Function Evalutions

Algorithm	Mean	Median	StdDev	Max[a]
`BFGS`	122.1	44	217.2	830
`Nelder-Mead`	348.8	303	136.6	603
`Newton-CG`	44.5	21	80.2	633
`SFit`	142.5	125	121.1	999

[a]Fitting terminated after 999 function evaluations.