Draft version February 6, 2025
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; we also find that the `Newton-CG` method is not effective for fitting microlensing events.

1. INTRODUCTION

Model optimization is a significant component of most quantitative analyses. Many analyses in the field of microlensing have long used on an optimization algorithm that relies on only first derivatives of the function being fit to the data. Because this feature is distinct from many of the most readily available optimization algorithms, we present the derivation of this algorithm, a Python implementation (`SFit`), and evaluate the performance of `SFit` against existing algorithms implemented in `SciPy.optimize.minimize()` for the microlensing use case.

2. 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 the 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 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.

3. IMPLEMENTATION

3.1. *General*

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()`. The user defines either the model, $F(x_k)$, or the residual, $y_k - F(x_k)$ where the y_k are the data, calculation (i.e., the method `my_func.model()` or `my_func.residuals()`). The user also defines the partial derivatives of the function to be minimized (i.e., the method `my_func.df()`), $\partial F(x_k)/\partial A_i$. 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.

3.2. *Microlensing-specific*

A point lens microlensing model (Paczynski 1986), A, is described by a minimum of three 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, $f_{{\rm S},k}$ and $f_{{\rm B},k}$, used to scale the model to each dataset, k, to obtain the model flux: $f_{{\rm mod},k} = f_{{\rm S},k} A + f_{{\rm B},k}$.

The `MulensModel` package (Poleski & Yee 2019) implements functions that calculate such models and their χ^2s and derivatives relative to data. The `mm_funcs.py` module contains microlensing-specific implementations that use `MulensModel`. The class `PointLensSFitFunction` takes a `MulensModel.Event` object as an argument and can be used with `sfit_minimizer.sfit_minimize.minimize()` to obtain the best-fitting model parameters. This usage is demonstrated in `example_01_pspl_fit.py` for fitting the three standard Paczyński parameters above. An example additionally including the finite source parameter, ρ, is given in `example_02_fspl_fit.py` (note fitting ρ requires `MulensModel` v3 or higher).

`mm_funcs.py` also includes a convenience function, `fit_mulens_event()`, that will automatically perform the fitting given a `MulensModel.Event` object.

4. 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`.

4.1. *Sample Selection*

We select our sample from microlensing events discovered in 2018 by KMTNet (Kim et al. 2018a, Kim et al. 2018b,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 [did I remove that one FS one?]** 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 also remove any points with NaN or negative errorbars.

4.2. *Fitting Procedure*

To find the initial starting value for the fit, we start by performing a series of linear fits using the EventFinder method (Kim et al. 2018a, Kim et al. 2018b). This method performs a linear fit over a grid of three parameters. From the best-fit EventFinder grid point, we take t_0, the time of the peak of the event. We remove any events for which the difference between the EventFinder t_0 and the reported KMTNet t_0 is more than 20 days (40 events). We also remove any events whose light curves appear to be flat or anomalous (8 additional events).

For the remaining events, we test a grid of values: $u_{0,i} = [0.01, 0.3, 0.7, 1.0, 1.5]$ and $t_{\rm E,}j = [1., 3., 10., 20., 40.]$. We perform a linear fit to the flux parameters $(f_{{\rm S},k}, f_{{\rm B},k})$ and choose the $(u_0, t_{\rm E})$ pair with the smallest χ^2 as the starting point for our fits. Then, we renormalize the errorbars of each dataset. The initial errorbar renormalization factor for each dataset is calculated as the factor required to make the $\chi^2/{\rm d.o.f.} = 1$.

We fit for the optimal model parameters using each algorithm. We use `MulensModel` (Poleski & Yee 2019) to calculate the point-lens microlensing light curve model, its derivatives, and the jacobians. For the three `SciPy.optimize.minimize()` algorithms, 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 $t_{\rm E}$. For `SFit`, we include the flux parameters as parameters of the fit. These distinctions mirror how we expect the algorithms to be used in practice for light curve fitting.

We remove the 34 events with t_0 for the best-fitting model outside the observing season ($8168 < {\rm HJD}' < 8413$). Our final sample has 1716 events.

To ensure a statistically robust comparison of results we renormalize the errorbars again so the $\chi^2/{\rm d.o.f.} = 1$ relative to the best-fitting model and repeat the fitting from the original starting point. This can be important in cases for which the initial starting point is relatively far from the true fit, e.g., cases with a true value of $t_{\rm E} > 40$ days. This renormalization can change the relative weighting of individual datasets, which in turn can affect the preferred model.

4.3. *Results*

For the second set of 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: algorithm reported success and found the minimum.
- 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 found the minimum.

For the purpose of these comparisons, we consider the algorithm to have found the minimum ("succeeded") if $\Delta\chi^2 < 1.0$.

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.

Table 1 shows that the `SFit` algorithm had the highest reliability for fitting microlensing events. It had very low false positive (0%) and false negative (1%) rates. The `BFGS` and `Nelder-Mead` algorithms both had low rates of false positives (0% and 1%, respectively) but most of the reported failures were false negatives ($\sim$ 100% and 91%, respectively). The false positives and false negatives for these two algorithms accounted for 32% and 13% of the fits, respectively. For the `Newton-CG` algorithm, 36% of the reported successes were false positives and 22% of the reported failures were false negatives, accounting for 46% of the total fits.

Figures 1–3 compare the performance of `SFit` to the other algorithms. All three plots have points that fall along the lines $x = 0$ or $y = 0$. This indicates that sometimes `SFit` will successfully fit events that other algorithms do not and vice versa. For Figures 2 and 3, the mix of both colors (purple=reported successes and red=reported failures) along these lines also indicates that there is no category of `SFit` fits (true positives, false positives, true negatives, false negatives) that is a subset of the same category for the other algorithm or vice versa. These qualitative impressions are quantified in the lower sections of Table 1.

In terms of number of χ^2 function evaluations, `SFit` is reasonably efficient. It scores in between the `BFGS` and `Nelder-Mead` algorithms (see Table 2).

5. SUMMARY

We presented an alternative method for generalizing Newton's method to minimize χ^2 and its implementation as Python package called `SFit`. We tested this implementation against the `BFGS`, `Nelder-Mead`, and `Newton-CG` algorithms in `SciPy.optimize.minimize()` to objectively evaluate its performance in fitting point-lens microlensing light curves from the Korea Microlensing Telescope Network survey data.

Of the three `SciPy.optimize.minimize()` algorithms, `BFGS` was able to find the best-fitting model almost 100% of the time, despite reporting failed fits for 32% of light curves. `Newton-CG` performed the worst with high rates of both false positives and false negatives. The `Nelder-Mead` algorithm performed well, successfully finding the χ^2 minimum for 98% of light curves, but with a significant number of false negatives and requiring the most number of function evaluations.

We find that `SFit` is able to successfully fit 83% of point-lens microlensing light curves. It is characterized by high reliability, with extremely low rates of both false positives and false negatives. It is also relatively efficient, requiring a median of 167 function evaluations to meet the required tolerance. An additional advantage of this algorithm and implementation is that it automatically estimates uncertainties in the fitted parameters.

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 and **NASA grant**. 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

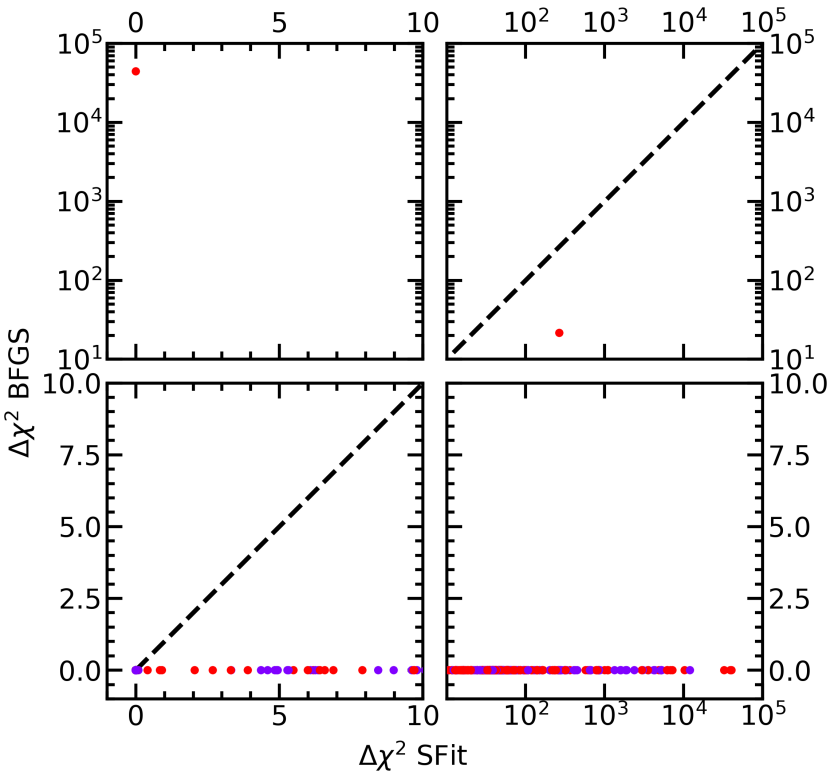

Figure 1. $\Delta\chi^2$ of the `BFGS` model relative to the best-fitting model vs. $\Delta\chi^2$ of the `SFit` model. Fits reported as successes by `BFGS`are plotted in purple, while those reported as failures are shown in red. Events that were fit successfully by both algorithms appear at (0, 0).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

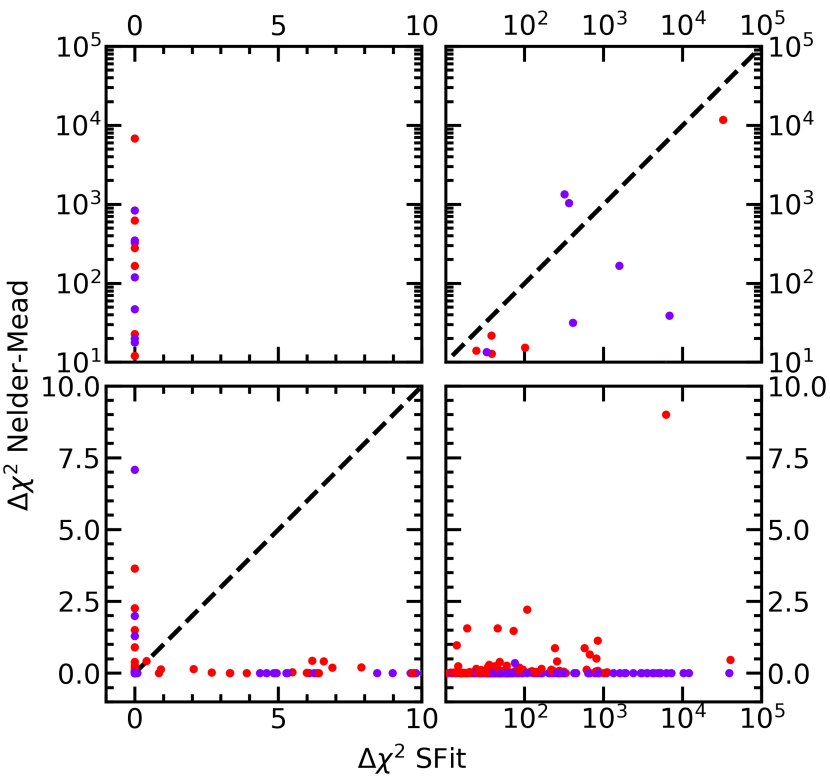

Figure 2. Same as Figure 1 but for the `Nelder-Mead` algorithm relative to `SFit`.

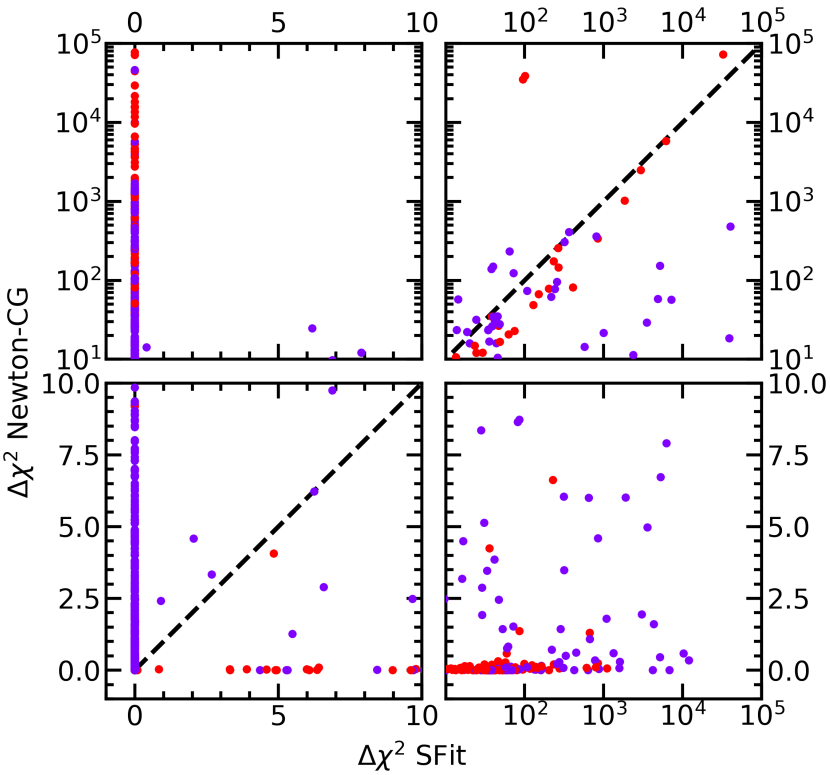

Figure 3. Same as Figure 1 but for the `Newton-CG` algorithm relative to `SFit`.

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

Algorithm	Total	$\Delta\chi^2 <$ 0.1		1.0		10.0		100.0	
		N	%	N	%	N	%	N	%
All 1716 Events:									
Algorithm Reported Success:									
`BFGS`	1172	1172	100	1172	100	1172	100	1172	100
`Nelder-Mead`	1488	1470	99	1471	99	1474	99	1481	100
`Newton-CG`	1307	590	45	843	64	1074	82	1225	94
`SFit`	1425	1425	100	1425	100	1425	100	1425	100
Algorithm Reported Failure:									
`BFGS`	544	542	100	542	100	542	100	543	100
`Nelder-Mead`	228	170	75	208	91	217	95	223	98
`Newton-CG`	409	295	72	318	78	325	79	351	86
`SFit`	291	1	0	4	1	32	11	206	71
1425 Events for which `SFit` reported success:									
Algorithm Reported Success:									
`BFGS`	1038	1038	100	1038	100	1038	100	1038	100
`Nelder-Mead`	1345	1335	99	1335	99	1338	99	1341	100
`Newton-CG`	1162	533	46	770	66	967	83	1089	94
Algorithm Reported Failure:									
`BFGS`	387	386	100	386	100	386	100	386	100
`Nelder-Mead`	80	67	84	71	89	74	92	76	95
`Newton-CG`	263	199	76	201	76	203	77	216	82
291 Events for which `SFit` reported failure:									
Algorithm Reported Success:									
`BFGS`	134	134	100	134	100	134	100	134	100
`Nelder-Mead`	143	135	94	136	95	136	95	140	98
`Newton-CG`	145	57	39	73	50	107	74	136	94
Algorithm Reported Failure:									
`BFGS`	157	156	99	156	99	156	99	157	100
`Nelder-Mead`	148	103	70	137	93	143	97	147	99
`Newton-CG`	146	96	66	117	80	122	84	135	92

Table 2. Number of χ^2 Function Evalutions

Algorithm	Mean	Median	StdDev	Max
`BFGS`	94.1	34	191.5	814
`Nelder-Mead`	430.7	419	106.4	600
`Newton-CG`	78.8	69	82.9	625
`SFit`	175.8	167	115.1	583