Science and
Technology
Facilities Council

Technical Report
RAL-TR-2022-002

Global optimization of crystal field parameter fitting in Mantid

M O'Flynn, J Fowkes, N Gould

February 2022

Enquiries concerning this report should be addressed to:

RAL Library
STFC Rutherford Appleton Laboratory
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446677
email: libraryral@stfc.ac.uk

DOI: 10.5286/raltr.2022002

ISSN 1358-6254

Global Optimization of Crystal Field Parameter Fitting in Mantid

Megan O'Flynn[1], Jaroslav Fowkes[1], and Nick Gould[1]

[1]UKRI-STFC Rutherford Appleton Laboratory, Scientific Computing Department, Computational Mathematics Group

24th February 2022

Abstract

Currently local optimization algorithms are used by the Mantid software package to fit parameters to crystal field data, however such fits are extremely sensitive to the selection of initial parameters. Even introducing a small perturbation to the initial parameters can change the fitting significantly, hence there is a need to consider a global, rather than a local, optimization approach to fit crystal field data. In this report, we propose and test several different global optimization algorithms that could be integrated into Mantid's fitting routines to try and make crystal field fitting more robust. Our numerical results on two real-world datasets demonstrate that there is great benefit to the use of global optimization for crystal field parameter fitting in Mantid.

Contents

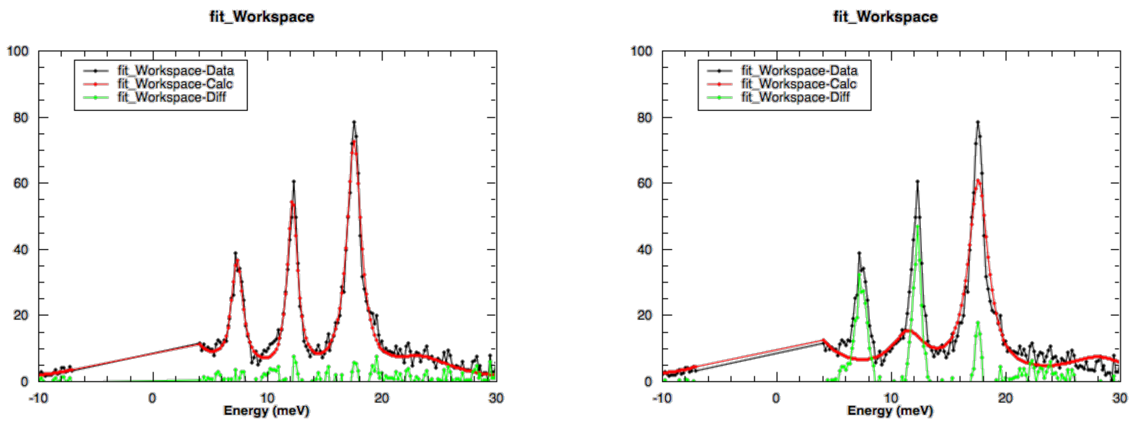
1 Introduction

Mantid is an open-source project that provides a framework for the computation and visualisation of neutron scattering and muon spectroscopy data [1, 2]. Mantid utilizes different optimization algorithms for fitting the measurements of spectra from scientific experiments, such as calculating and fitting the measurements of the transitions between Crystal Field (CF) energy levels [1, 2]. We are interested in the `CrystalFieldSpectrum` function and how it fits parameters for Inelastic Neutron Scattering (INS) spectra.

This report comprises four sections. Section 1 introduces the crystal field problem and gives a brief introduction to Mantid nonlinear least-squares problems. Sections 2 and 3 outline the global optimization algorithms that we have developed to solve the problem, and the results of benchmarking the best candidate global optimization algorithm against the current local optimization algorithms used in Mantid, respectively. Section 4 concludes with suggestions of the next steps for taking global optimization in Mantid further.

1.1 Crystal Field Parameter Fitting in Mantid

Currently local optimization algorithms are used by Mantid to fit parameters to crystal field data. However, the INS spectra are extremely sensitive to the selection of initial parameters. Figure 1 shows parameter fitting using highly-tuned parameters and parameter fitting when we perturbed these parameters slightly (namely rounded up several of them). We can see that even introducing a small perturbation to the crystal field parameters changes the fitting significantly. Local optimization methods often struggle on problems that have many local minima, and this increases the risk that these optimizers may find minima that fit crystal field data poorly.



(a) Crystal field fitting with expert-tuned initial parameters.

(b) Crystal field fitting with the expert initial parameters perturbed slightly.

Figure 1: Plots of crystal field fitting with different sets of initial parameters.

The crystal field problem can be treated generically as a nonlinear least-squares fitting problem, as we detail in the next section. Since the INS spectra are very sensitive to the parameters chosen, there is a need to consider a global, rather than a local, optimization approach to fit crystal field data. In this report, we consider a range of different global optimization algorithms that could be integrated into

Mantid's fitting routines to try to make crystal field fitting of INS spectra more robust.

1.2 Nonlinear Least-Squares Problems

Nonlinear least-squares optimization focuses on minimizing functions of the form

$$f(x) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(x) = \frac{1}{2}\|r(x)\|^2, \tag{1}$$

where r_i is the i-th residual function. For fitting problems, the residual function describes the discrepancy between the model and the observed data [3].

As an example, given a set of scalar inputs $\{a_i\}_{i=1}^m$ and outputs $\{y_i\}_{i=1}^m$, one might be interested in finding the straight line

$$y = x_1 a + x_0$$

that 'best' fits them. Our aim is then to find the parameters x_0 and x_1. If we define the residuals

$$r_i(x_0, x_1) = x_1 a_i + x_0 - y_i,$$

we may choose x_0 and x_1 to minimize

$$f(x_0, x_1) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(x_0, x_1) = \frac{1}{2}\sum_{i=1}^{m}(x_1 a_i + x_0 - y_i)^2.$$

Since we are measuring 'best' by minimizing the sum of squares of linear functions, this is a linear least-squares problem. For nonlinear problems we seek a curve of best fit.

In order to minimize (1) we make use of the gradient of $f(x)$, given by

$$g(x) = \nabla f(x) = J(x)^T r(x),$$

where the Jacobian matrix $J(x)$ is defined by its components $J_{ij}(x) = \partial r_i(x)/\partial x_j$.

For the crystal field problem, the objective function we want to minimize is[1]

$$f(x) = \sum_{i=1}^{m} r_i^2(x) = \sum_{i=1}^{m}\left(\frac{\mathrm{CF}(a_i, x) - y_i}{e_i}\right)^2, \tag{2}$$

where $\mathrm{CF}(a, x)$ is the crystal field model function, (a_i, y_i) are the set of m data points, x is the set of parameters to be optimized, and e_i are scaling weights that often represent approximations to data error standard deviations (e.g. due to measurement error).

1.3 The Crystal Field Model

The crystal field model function can be thought of as a weighted superposition of p peak functions, e.g. Gaussians, the i-th being centered at c_i with scale parameter σ_i, that is

$$\mathrm{CF}(a, (B, S, \sigma)) = S\sum_{i=0}^{p-1} I_i(B)\exp\left(-\frac{(a - c_i(B))^2}{2\sigma_i^2}\right). \tag{3}$$

[1]Note that here we drop the 1/2 in front of the objective, this merely scales various quantities in the optimization algorithms that follow but otherwise has no implications for the optimization.

where S is a scaling parameter for the weights (or intensities) I_i. For clarity we present Gaussians here, although the problems we will subsequently test on use Lorentzians with scale parameters γ_i, that is

$$\mathrm{CF}(a,(B,S,\gamma)) = S\sum_{i=0}^{p-1} I_i(B)\left(\frac{\gamma_i^2}{(a-c_i(B))^2+\gamma_i^2}\right). \tag{4}$$

The peak centers c_i and intensities I_i themselves depend on parameters B of the Stevens' crystal field Hamiltonian, and thus the parameters to be optimized in this case are $x=(B,S,\sigma)$ or $x=(B,S,\gamma)$, with a being a data point x-value as before. Note that rather than optimizing over the scale parameters directly, Mantid uses the Full Width at Half Maximum (FWHM), defined as the width of the peak measured at half of its maximum amplitude. For a Gaussian $\textsc{fwhm}_i = 2\sqrt{2\ln 2}\sigma_i$ and for a Lorentzian $\textsc{fwhm}_i = 2\gamma_i$. More specifically, the peak centres c_i and intensities I_i depend on the eigenvalues $\lambda_j(B)$ and discretised eigenfunctions $\psi_j(B)$ of a time-independent Schrödinger operator, specifically on the solutions to the eigenvalue problem

$$H(B)\psi = \lambda\psi,$$

with the Stevens' crystal field Hamiltonian $H(B)$ given by[2]

$$H(B) = \sum_{k=0,2,4,6}\sum_{q=-k}^{k} B_q^k O_q^k$$

with known Stevens' operator matrices O_q^k. Given the eigensolutions, we then have

$$\begin{aligned} c_i(B) &= \lambda_j(B) - \lambda_k(B) \\ I_i(B) &\propto |\psi_j^T(B) D \psi_k(B)|^2 \end{aligned}$$

for the appropriate transitions between energy levels j and k, and known magnetic dipole operator D. We refer the interested reader to [4] for further details of crystal field theory and its use within Mantid.

[2]The $k=0$ term $B_0^0 O_0^0$ only introduces a uniform shift in the values of all the eigenvalues, but as we are only interested in the differences between eigenvalues here, it is often ignored or omitted.

2 Developing Global Optimization Algorithms

We investigate different approaches to global optimization of nonlinear least-squares problems of the form

$$f(x) = \frac{1}{2}\sum_{i=1}^{m} r_i^2(x) = \frac{1}{2}\|r(x)\|^2. \tag{1}$$

In particular, we focus on three multi-start global optimization methods in this section — the multi-start selective search algorithm, a multi-start regularisation approach, and a multi-start linesearch approach — implement each, and compare all three on problems formulated in [5]. The algorithms are implemented using the NumPy [6], SciPy [7], and Surrogate Modelling Toolbox [8] packages in Python.

2.1 Multi-start Optimization Algorithms

The idea behind multi-start global optimization algorithms is a simple one. The main weakness of using traditional (so called 'local') optimization algorithms to find global minimisers is that they can (and often do) get stuck at local optima. In order to remedy this, multi-start algorithms simply run a local optimization algorithm from multiple starting points within the parameter space. The key to their success is therefore to ensure that the multiple starting points are well spaced-out within the parameter space. There are multiple techniques for this, but one of the simplest and most effective is to use Latin Hypercube Samples (LHS) [9]. To generate a Latin Hypercube Sample of l points in 2D, one creates an $l \times l$ equally spaced square grid in the parameter domain and arranges the l points at random such that there is one centered in every row and column of the grid (see Figure 2 below). This approach generalises to higher parameter dimensions in the obvious way.

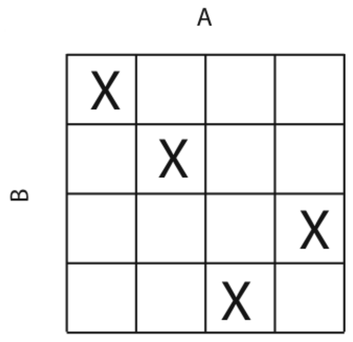


Figure 2: Four Latin Hypercube Samples for two parameters A and B.

2.2 Multi-start Selective Search (MS3)

The multi-start selective search method, outlined by Velázquez et al. [5], was the first candidate that we considered. This approach incorporates a variation of the Levenberg-Marquardt method and a multi-start strategy, with the assumption that having multiple initial points will improve the ability of the algorithm to find global minima for small or zero-residual least-squares problems. Algorithm 1 below shows the pseudocode for the MS3 algorithm.

The Levenberg-Marquardt method [3] is one of the most popular and widely-used algorithms for nonlinear least-squares problems. At every iteration k, Levenberg-Marquardt calculates a step s_k that gives us a new iterate $x_k + s_k$ by solving the perturbed normal equations

$$(J(x_k)^T J(x_k) + \mu_k I)s_k = -J(x_k)^T r(x_k)$$

for some carefully chosen perturbation $\mu_k \geq 0$. For $\mu_k = 0$, Levenberg-Marquardt becomes the Gauss-Newton method (i.e. Newton's method where the Hessian H has been approximated by its first-order term $J^T J$). Conversely, for sufficiently large μ_k Levenberg-Marquardt approaches the steepest-descent method with a suitable stepsize.

Notice that in MS3 there is no attempt to enforce uniform descent of the residuals $\|r(x)\|$, and this has the potential to allow iterates to move between basins containing different local minimizers. Note also that the choice of τ here is largely problem-dependent.

Algorithm 1 MS3 Algorithm with LHS

```
Require: $\epsilon > 0, \tau > 0$
  Set $f_{min} = \infty$ and $x_{min} = \infty$
  Generate $l_{max}$ points using LHS and set $l = 0$
  while $l < l_{max}$ and $\|r(x_k)\|^2 > \epsilon$ do
      Select an unused LHS point $x_0$ and set $k = 0$
      while $k < k_{max}$ and $\|g(x_k)\| > \epsilon$ do
          Set $\mu_k = \tau\|r(x_k)\|$
          Solve $(J(x_k)^T J(x_k) + \mu_k I)s_k = -J(x_k)^T r(x_k)$
          Set $x_{k+1} = x_k + s_k$ and increment $k$
      end while
      if $f(x_k) < f_{min}$ then
          Set $f_{min} = f(x_k)$ and $x_{min} = x_k$
      end if
      Increment $l$
  end while
  return $x_{min}$
```

Although it was possible to reproduce the results for the lower dimensional problems in Velázquez et al's paper, the MS3 algorithm struggled to consistently find global minima in higher dimensions. Fitting in higher dimensions for the test problems from [5] is challenging because the number of local minima increases rapidly as the number of dimensions increases. Therefore the higher the dimension, the more difficult it becomes to find global minima. It proved possible, after increasing the number of iterations to search for the global minima to $4,000$, to find global minima for the higher dimensional problems. Unfortunately, due to the nature of the crystal field objective function, the MS3 algorithm was not able to find global minima in this case. As a result we conclude that this algorithm is not a good candidate for such problems. However, for completeness, the results from running the algorithm for the test problems in [5] will be included.

2.3 Multi-start with Regularisation

The second approach we considered was a multi-start variant of a quadratic regularisation method with adaptive regularisation [10]. In this approach, at every iteration k, we approximate the objective function f around x_k using the quadratic model

$$m_k(s) = \frac{1}{2}\|J(x_k)s + r(x_k)\|^2 + \frac{\sigma_k}{2}\|s\|^2 \tag{5}$$

with adaptive regularisation parameter σ_k. We then minimize (5) to find a step s_k that gives us a new iterate $x_k + s_k$ with lower objective value. In order to minimize (5) we find zeros of its gradient

$$\nabla m_k(s_k) = (J(x_k)^T J(x_k) + \sigma_k I)s_k + J(x_k)^T r(x_k) = 0$$

by solving the above set of linear equations, called the perturbed normal equations. Algorithm 2 below outlines the pseudocode for this multi-start method, including how the regularisation parameter σ_k is chosen adaptively based on the level of agreement ρ_k between the objective function f and the quadratic model m_k.

Algorithm 2 Multi-start Regularisation Method

```
Require: $\epsilon > 0, \epsilon_s > 0$
  Set $f_{min} = \infty$ and $x_{min} = \infty$
  Generate $l_{max}$ points using LHS and set $l = 0$
  while $l < l_{max}$ and $\|r(x_k)\|^2 > \epsilon$ do
     Set $\sigma_0 = \|g(x_0)\|/10$ and $k = 0$
     while $k < k_{max}$ and $\|g(x_k)\| > \epsilon$ and $\|s_k\| > \epsilon_s$ do
        Solve $(J(x_k)^T J(x_k) + \sigma_k I)s_k = -J(x_k)^T r(x_k)$
        Compute $\rho_k = (f(x_k) - f(x_k + s_k))/(m_k(0) - m_k(s_k))$
        if $\rho_k \geq 0.1$ then
           Set $x_{k+1} = x_k + s_k$
        end if
        if $\rho_k < 0.1$ then
           $\sigma_k = \min\{\sqrt{2}\sigma_k, \sigma_{max}\}$
        else if $\rho_k \geq 0.75$ then
           $\sigma_k = \max\{\sqrt{0.5}\sigma_k, \sigma_{min}\}$
        end if
        Increment $k$
     end while
     if $f(x_k) < f_{min}$ then
        Set $f_{min} = f(x_k)$ and $x_{min} = x_k$
     end if
     Increment $l$
  end while
  return $x_{min}$
```

Note that here σ_{max} and σ_{min} are chosen to prevent numerical overflow or underflow when the regularisation parameter is increased or decreased, respectively. For our experiments we used $\sigma_{max} = 10^{20}$ and $\sigma_{min} = 10^{-15}$.

2.4 Multi-start with Linesearch

The third approach that we considered was a multi-start variant of a backtracking linesearch method [3]. In this approach, at every iteration k, we take a step s_k along the Gauss-Newton direction

$$s_k = -(J(x_k)^T J(x_k))^{-1} g(x_k) = -(J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k)$$

which is just the Newton direction $-H^{-1}g$ where the Hessian H has been approximated by its first-order term $J^T J$. In order to guarantee convergence, the step s_k is scaled by a step-size α_k computed using a standard backtracking Armijo linesearch. Algorithm 3 below outlines the pseudocode for this method, where $\kappa(J(x_k))$ denotes the condition number of $J(x_k)$, i.e. the ratio of largest to smallest singular value.

Algorithm 3 Multi-start Linesearch Method

```
Require: $\epsilon > 0, \tau > 0$
  Set $f_{min} = \infty$ and $x_{min} = \infty$
  Generate $l_{max}$ points using LHS and set $l = 0$
  while $l < l_{max}$ and $\|r(x_k)\|^2 > \epsilon$ do
      while $k < k_{max}$ and $\|g(x_k)\| > \epsilon$ do
          if $\kappa(J(x_k)) > 10^8$ then
              Set $\mu_k = \tau\|(r(x_k)\|$
              Solve $(J(x_k)^T J(x_k) + \mu_k I)s_k = -J(x_k)^T r(x_k)$
          else
              Solve $J(x_k)^T J(x_k) s_k = -J(x_k)^T r(x_k)$
          end if
          Set $\Delta_k = 0.5\, g(x_k)^T s_k$ and $\alpha_k = 5$
          Perform backtracking to compute $\alpha_k$:
          while $f(x_k + \alpha_k s) > f(x_k) + \alpha_k \Delta_k$ do
              $\alpha_k = \alpha_k / 2$
          end while
          Set $x_{k+1} = x_k + \alpha_k s_k$ and increment $k$
      end while
      if $f(x_k) < f_{min}$ then
          Set $f_{min} = f(x_k)$ and $x_{min} = x_k$
      end if
      Increment $l$
  end while
  return $x_{min}$
```

Notice, in particular, that the precaution introduced when $J(x_k)$ is ill-conditioned to prevent the method from stalling is vital, and resembles to a certain extent the approach adopted by explicit regularization in the previous section. Unfortunately, due to the nature of the crystal field objective function, $J(x_k)$ turned out to be particularly ill-conditioned, leading the multi-start linesearch method to compute poor search directions s_k despite our precautions, making it unsuitable for fitting crystal field data.

2.5 Sine Components Test Problems in High Dimensions

We compared the rival algorithms on two classes of related test problems taken from [5] (note that these problems can be rewritten in the form (1)). In [5], these problems are referred to as test problems 2-7. The first class involves a function containing sine components:

$$f(x) = \frac{\pi}{n}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i-1)^2(1+10\sin^2(\pi y_{i+1}))\right] + (y_n-1)^2\right) \tag{6}$$

where $y_i = 1 + 0.25(x_i - 1)$, $-10 \le x_i \le 10, i = 1, 2, ...n$

We choose $n = 2, 3, 4$ for this function, and these give us problems 2–4.

The second class of problems are variants of the first, and involve the function

$$f(x) = \frac{\pi}{n}\left(10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}\left[(x_i-1)^2(1+10\sin^2(\pi x_{i+1}))\right] + (x_n-1)^2\right); \tag{7}$$

we considered this function with $n = 5, 8, 10$, and these give us problems 5–7.

The global minima of the functions (6) and (7) are $f(x_i^*) = 0$ and occur for all $x_i = 1$. Each algorithm used 15 initial starting points selected randomly using Latin Hypercube Sampling (LHS) [9] and searched for the minimum over 4,000 iterations (rather than the 400 iterations used in [5]). We chose to use Latin Hypercube Sampling to generate random initial points in order to ensure that we had a wide range of initial points and to avoid generating clusters of initial points.

For these test problems, the algorithms were implemented for Python 3.8.2, and LHS was provided by the SciPy library [7]. For FitBenchmarking in the later sections of this report, Mantid at the time of this report was only compatible with Python 3.6 and so the Surrogate Modelling Toolbox [8] was used to provide LHS as the Scipy package compatible with Python 3.6 did not yet support LHS.

Tables 1 and 2 show the results for test problems (6) and (7) respectively.

P	**D**	**M**	**Method**	**Global min**	$f(x^*)$	**Time (s)**
2	2	25	MS3	[1. 1.00000001]	2.600807e-18	7.41
			Regularisation	[0.99999902 0.9999753]	6.915500e-11	11.58
			Linesearch	[1. 0.99999999]	5.141708e-18	25.45
3	3	125	MS3	[1. 1.00288098 0.44927581]	8.148573e-07	16.31
			Regularisation	[1. 0.6114404 6.61058064]	1.482229e-02	21.37
			Linesearch	[1.00204555 0.90349506 -8.94670272]	9.552993e-04	60.48
4	4	625	MS3	[1. 1. 1. 0.99999999]	2.548132e-20	23.76
			Regularisation	[1.11194848 0.86463193 7.500665 -1.20510409]	1.262897e-01	36.70
			Linesearch	[5.09998398 1.08516193 4.3481204 8.28728688]	1.822146e+00	80.53

Table 1: Results of Multistart algorithms on Velázquez et al's Test Problems 2-4 (P - Problem Number, D - Number of Dimensions, M - Number of Local Minima)

From Table 1, we can see that for the lower dimensional problems MS3 has been able to find the global minimizer for problems 2 and 4, however it struggles to find the global minimizer for problems 6 and 7 from Table 2. This suggests that although the MS3 algorithm works for finding the minima in lower dimensions,

P	D	M	Method	Global min	$f(x^*)$	Time (s)
5	5	10^5	MS3	[1. 1. 1. 1. 1.]	1.221638e-21	41.35
			Regularisation	[1.99005591 1.00000028 1. 1.00000009 1.00313873]	6.220209e-01	53.11
			Linesearch	[1. 1. 1. 1. 1.]	3.981310e-20	69.55
6	8	10^8	MS3	[-1.96969121 -1.99647615 -3.98709658 -4.14596438 -1.15071274 1. 0.99999999 1.23710579]	6.985101e+01	136.41
			Regularisation	[5.94896522 -3.99793217 1.00121448 1.04761504 3.32587584 1. -0.18652503 -1.9788127]	2.572665e+01	135.72
			Linesearch	[1. 1. 1. 1. 1. 1. 1. 1.]	1.968636e-20	175.68
7	10	10^{10}	MS3	[1.04102684e-02 1.95090534e-02 -3.90188515e+00 5.08711096e+00 1.26143798e-03 -3.90728964e+00 6.08185861e+00 4.10667882e+00 6.95810223e+00 -3.74890973e+00]	1.270703e+02	234.88
			Regularisation	[1.00000000e+00 1.49566561e+00 3.88833753e+00 -5.99150671e+00 5.99896373e+00 2.99914778e+00 9.97716261e+00 -9.99956728e-01 -9.94941357e-01 2.53953430e-03]	5.547867e+01	231.98
			Linesearch	[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]	1.961076e-20	282.89

Table 2: Results of Multistart algorithms on Velázquez et al's Test Problems 5-7 (P - Problem Number, D - Number of Dimensions, M - Number of Local Minima)

it cannot consistently find the global minimum when the number of dimensions increases. For the Regularisation and Linesearch algorithms we can see that they can find the global minimum for a 2D problem (problem 2). In Figure 3 we can see the path that all three algorithms took in order to reach the global minima for the 2-dimensional problem. However while the regularisation algorithm struggles to find the global minimum for higher dimensions, the Linesearch algorithm is able to find the global minimum for problems 5-7. We found, however, that for the crystal field problem the Regularisation algorithm performed consistently better than the Linesearch algorithm.

However, Table 2 indicates that as the dimensions are increased the MS3 algorithm struggles to find the global minimum. This is also the case for the regularisation algorithm. However the Linesearch algorithm was able to successfully find global minima for test problems 6 and 7.

Although the MS3 algorithm was able to perform better on these test problems, it was not able to fit data from the crystal field problem. This is possibly due to how the crystal field problem has been scaled. The Regularisation and Linesearch methods however were able to fit the data better than MS3. As a consequence, in the remaining sections of this report, the MS3 algorithm will not be considered as a candidate for crystal field parameter fitting.

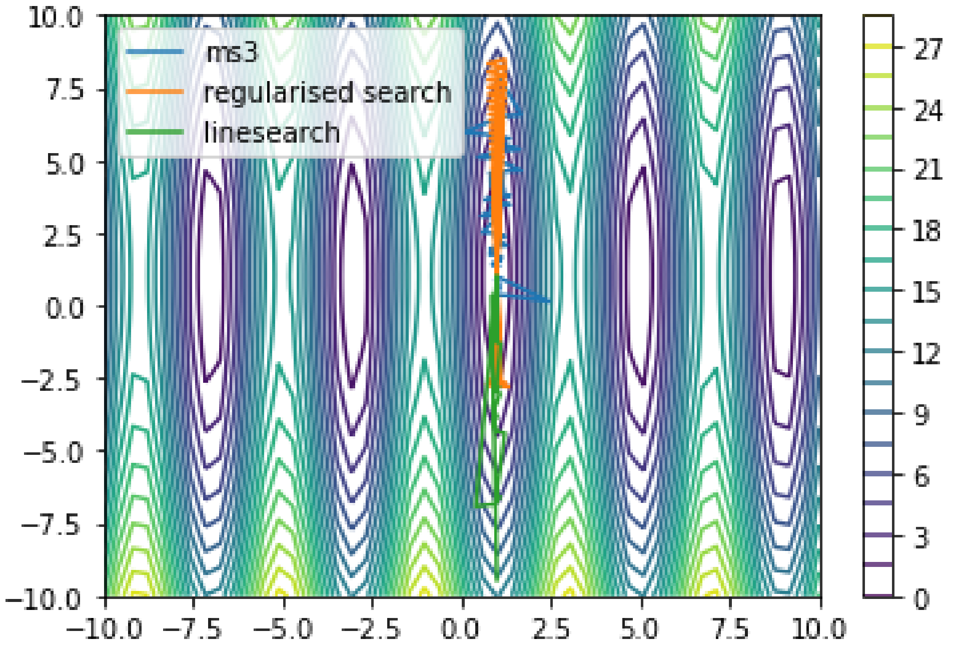


Figure 3: The search paths the three multi-start algorithms took to the global minimiser at $(1, 1)$ for test problem 2. Notice in particular how each starts with an x-coordinate close to 1, showing the importance of starting a local search in the basin of attraction of the global minimiser.

3 Fitting Crystal Field Data from Mantid

The next step is to benchmark the global optimization algorithms against the current local optimization algorithms being used in Mantid. The multi-start regularisation algorithm was identified as a suitable candidate for fitting crystal field data. In this section we will describe the crystal field datasets that we used for benchmarking, outline the algorithms that were benchmarked, and discuss the results of these algorithms for fitting crystal field parameters.

3.1 Crystal Field Test Problems

We used two different datasets for benchmarking crystal field parameter fitting.

3.1.1 Test Problem 1

The first test problem we use is taken from the Mantid crystal field examples [11] and consists of fitting the INS spectrum of $NdOs_2Al_{10}$ measured at 5K using a 35meV neutron beam at ISIS.

3.1.2 Test Problem 2

The second test problem we use is taken from the paper by Ritter et al. [12] and consists of fitting the INS spectrum of $NdOs_2Al_{10}$ measured at both 5K and 15K using a 35meV neutron beam at ISIS. Note that this is the same compound as in test problem 1 but using a different set of measurements, hence the INS spectra fits at 5K are very similar to test problem 1 but not identical.

3.2 Benchmarking Optimization Algorithms

The FitBenchmarking suite [13] was used to benchmark and compare the performance of multi-start global optimization algorithms against the current local optimization solvers in Mantid. It's important to note here that the Mantid solvers are implemented in C++, while the multi-start algorithms were implemented in Python. This difference has an influence on the runtimes of the algorithms applied to the the crystal field data, which can be seen in Figures 8 and 20. As well as the difference in implementations, the global multi-start algorithms have several more stages when searching for the global minima, and the nested Python loops implemented to do this increase runtime further. The cost function in FitBenchmarking is defined as the weighted nonlinear least-squares cost function precisely as described in (2) for the crystal field model with Lorentzian peaks (4).

3.2.1 Multi-start Global Optimization Algorithms

In addition to the multi-start regularisation algorithm, we also compare to a multi-start L-BFGS-B algorithm as a baseline. We omit results for the multi-start line-search algorithm since it struggled to fit crystal field data due to ill-conditioning of the Jacobian. As a result, two multi-start algorithms were benchmarked:

- Multi-start L-BFGS-B

- Multi-start Regularisation

The Multi-start L-BFGS-B method repeated parameter fitting with different initial points using the L-BFGS-B local optimization algorithm from SciPy and returned the minimum with the smallest cost function value across all fits. For the multi-start regularisation algorithm, in addition to a gradient norm termination condition of $\epsilon = 10^{-4}$, we included an additional termination condition, defined in Algorithm 2, that stops the algorithm when the norm of the step size is less than $\epsilon_s = 10^{-8}$. As well as this, we used an alternating optimization approach for the regularisation algorithm, described in subsubsection 3.2.2. All multi-start algorithms used 100 initial points for finding the global minima. These points were generated by Latin Hypercube Sampling, using suitable bounds for each parameter and scaled as described in subsubsection 3.2.3. All iterates were rescaled to the original parameter scales before being passed to the cost function in Mantid. The global minimum was returned scaled within the original parameter bounds. The Jacobian was calculated outside of Mantid using finite-differences as described in subsubsection 3.2.4.

3.2.2 Alternating Optimization for the Regularisation Algorithm

Alternating Optimization was performed for the Regularisation Algorithm in order to treat the crystal field problem as a lower-dimensional problem. To do this, optimization was done for the crystal field parameters in two blocks: the B parameters ($B_2^0, B_2^2, B_4^0, B_4^2, B_4^4, B_6^0, B_6^2, B_6^4, B_6^6$) and the shape parameters (S and either $\textsc{fwhm}_0, \textsc{fwhm}_1, \ldots, \textsc{fwhm}_4$ at 5K or $\textsc{fwhm}_0, \textsc{fwhm}_1, \ldots, \textsc{fwhm}_{20}$ at 10K). The stages for alternating optimization are as follows:

1. Fix the shape parameters and optimize over the B parameters for the crystal field problem. The shape parameters are fixed to the default initial parameters as defined for the crystal field data.
2. Fit the optimized B parameters and optimize over the shape parameters.
3. Fix the optimized shape parameters and optimize over the B parameters one more time.
4. Fix the optimized B parameters and optimize over the shape parameters.

3.2.3 Bounds and Scaling Parameters for Multi-start Algorithms

The Latin Hypercube sampling for the multi-start algorithms requires bounds on the parameters within which to sample the initial starting points, i.e. for the parameters x we require lower bounds x^L and upper bounds x^U such that

$$x \in [x^L, x^U].$$

For the crystal field B parameters, we used the Mantid built-in `split2range` function to determine suitable bounds. For the intensity scaling S, we used $[0, 10]$ and for the FWHM parameters we used $[0.1, 5]$ (except for $\textsc{fwhm}_4$ at 5K for both Problem 1 and 2 which used $[0.1, 7]$). Note that it does not matter if the global minimiser lies outside of these bounds, provided the basin of attraction of the global minimiser for the multi-start algorithm lies at least partially within these bounds.

For the multi-start algorithms, the initial points were also rescaled using min-max normalization to be within the range $[0, 1]$, that is

$$x^{scaled} = \frac{x - x^L}{x^U - x^L}.$$

This was done so that for each parameter the algorithm would take steps of similar magnitude, which improved convergence to the global minimum. These scaled points were rescaled to the original parameter scales once the global minimum was located.

3.2.4 Calculating the Jacobian for Multi-start Algorithms

The Jacobian was calculated using forward finite-differences outside of Mantid for better accuracy. It was observed during benchmarking that the basins around the global minimiser are often very narrow and the gradients of points close to the global minimiser are very large. This makes it harder to estimate derivatives accurately, and we therefore used SciPy forward finite-differences to calculate the Jacobian as it was more numerically stable than using the estimated derivatives from Mantid. The step-size influences the accuracy of the Jacobian and as a result the relative step-size for approximating the Jacobian was fixed at $\approx 10^{-8}$ ($\sqrt{\epsilon}$, where $\epsilon \approx 10^{-16}$ is machine precision, which is considered optimal for forward finite-differences).

3.2.5 Local Optimization Algorithms in Mantid

There are nine local optimization solvers in Mantid that were successfully benchmarked against the global optimization algorithms for the crystal field problem:

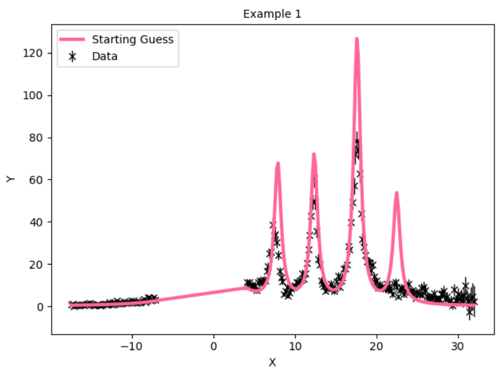
- BFGS
- Conjugate Gradient (Fletcher-Reeves imp.)
- Conjugate Gradient (Polak-Ribiere imp.)
- Damped Gauss-Newton
- Levenberg-Marquardt
- Levenberg-MarquardtMD
- Simplex
- Steepest Descent
- Trust Region

See [14] for details of the above methods. The two solvers from Mantid we are particularly interested in are the Levenberg-Marquardt and Trust Region methods.

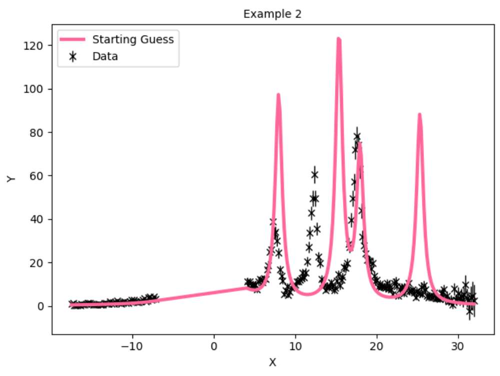
3.3 Test Problem 1 Initial Points

To compare performance of local versus global optimization algorithms on the first crystal field test problem, we used five different initial optimization starting points for the local optimization algorithms (note that these were not used by the global multi-start algorithms which used 100 Latin Hypercube Samples as starting points):

1. Ideal initial point
2. Imperfect initial point (rounded version of above)
3. Initial point selected at random using LHS
4. Another initial point selected at random using LHS
5. Perturbed ideal initial point

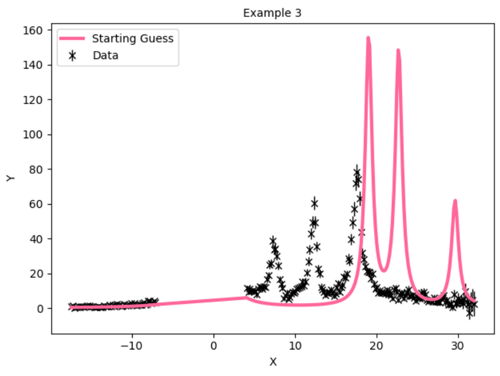


(a) Example 1

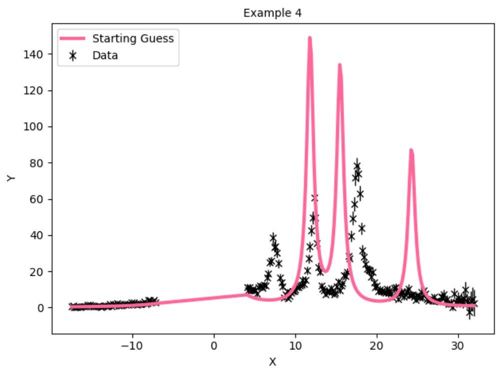


(b) Example 2

Figure 4: Plot of Initial Points for Examples 1 and 2



(a) Example 3



(b) Example 4

Figure 5: Plot of Initial Points for Examples 3 and 4

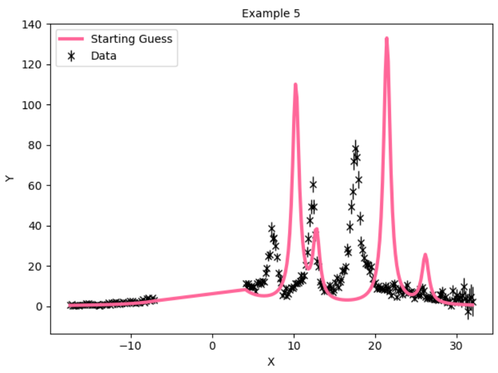
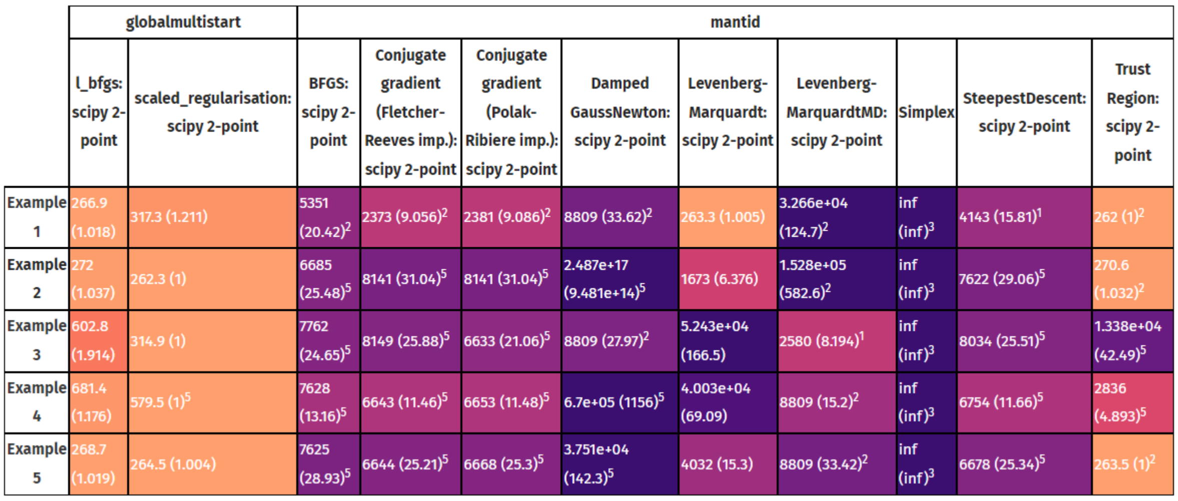


Figure 6: Plot of Initial Point for Example 5

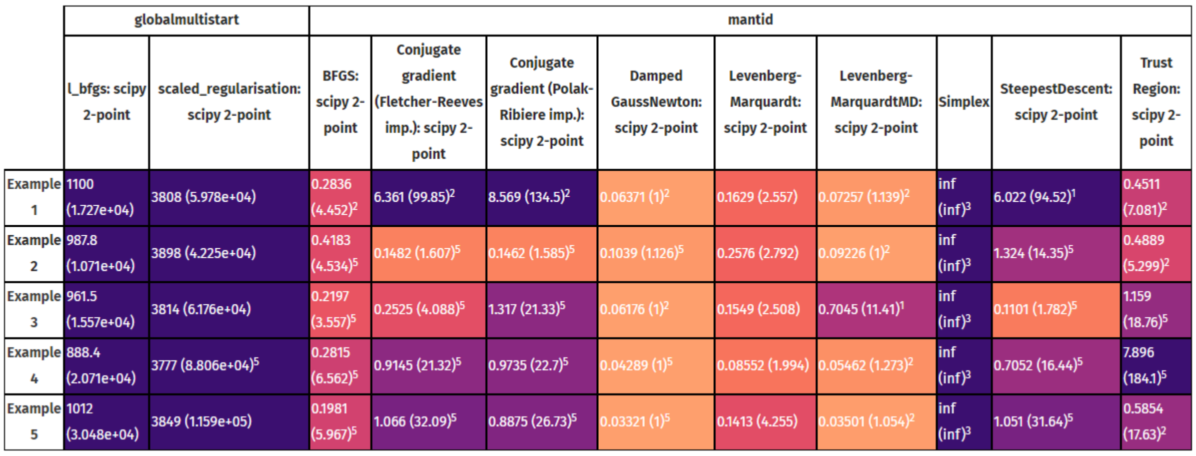
3.4 Test Problem 1 Results

Figures 7 and 8 show the performance in terms of accuracy (the value of the cost function f at the minimiser x^*) and the CPU runtime in seconds. As we can see from both figures, the Simplex method had raised an internal error (indicated by the 3 in the superscript) and did not return any parameters. From Figure 7, we can see that the Levenberg-Marquardt and the Trust Region methods from Mantid show good results for Example 1. However, for Example 2 onwards the Levenberg-Marquardt method does not perform as well as the Trust region method or the Regularisation method (here denoted as *scaled_regularisation* to indicate we are using scaled parameters). We can see that the Trust Region method shows similar performance to the Regularisation algorithm for Examples 2 and 5, but does not perform as well for Examples 3 and 4. The multi-start algorithms however perform better for these examples, with the Regularisation algorithm consistently finding global minima that fit the crystal field data well across all the examples. Figures 9–13 show the crystal field parameter fits for the multi-start global optimization, Levenberg-Marquardt, and Trust Region methods in the next subsections.



	globalmultistart		mantid								
	l_bfgs: scipy 2-point	scaled_regularisation: scipy 2-point	BFGS: scipy 2-point	Conjugate gradient (Fletcher-Reeves imp.): scipy 2-point	Conjugate gradient (Polak-Ribiere imp.): scipy 2-point	Damped GaussNewton: scipy 2-point	Levenberg-Marquardt: scipy 2-point	Levenberg-MarquardtMD: scipy 2-point	Simplex	SteepestDescent: scipy 2-point	Trust Region: scipy 2-point
Example 1	266.9 (1.018)	317.3 (1.211)	5351 $(20.42)^2$	2373 $(9.056)^2$	2381 $(9.086)^2$	8809 $(33.62)^2$	263.3 (1.005)	3.266e+04 $(124.7)^2$	inf $(\text{inf})^3$	4143 $(15.81)^1$	262 $(1)^2$
Example 2	272 (1.037)	262.3 (1)	6685 $(25.48)^5$	8141 $(31.04)^5$	8141 $(31.04)^5$	2.487e+17 $(9.481\text{e+}14)^5$	1673 (6.376)	1.528e+05 $(582.6)^2$	inf $(\text{inf})^3$	7622 $(29.06)^5$	270.6 $(1.032)^2$
Example 3	602.8 (1.914)	314.9 (1)	7762 $(24.65)^5$	8149 $(25.88)^5$	6633 $(21.06)^5$	8809 $(27.97)^2$	5.243e+04 (166.5)	2580 $(8.194)^1$	inf $(\text{inf})^3$	8034 $(25.51)^5$	1.338e+04 $(42.49)^5$
Example 4	681.4 (1.176)	579.5 $(1)^5$	7628 $(13.16)^5$	6643 $(11.46)^5$	6653 $(11.48)^5$	6.7e+05 $(1156)^5$	4.003e+04 (69.09)	8809 $(15.2)^2$	inf $(\text{inf})^3$	6754 $(11.66)^5$	2836 $(4.893)^5$
Example 5	268.7 (1.019)	264.5 (1.004)	7625 $(28.93)^5$	6644 $(25.21)^5$	6668 $(25.3)^5$	3.751e+04 $(142.3)^5$	4032 (15.3)	8809 $(33.42)^2$	inf $(\text{inf})^3$	6678 $(25.34)^5$	263.5 $(1)^2$

Figure 7: FitBenchmarking accuracy $f(x^*)$ on Crystal Field Problem 1 (values in brackets are relative to the best result). Superscript meanings: 1 - maximum number of iterations exceeded, 2 - unable to converge to solution, 3 - internal error, 5 - solution does not respect parameter bounds.

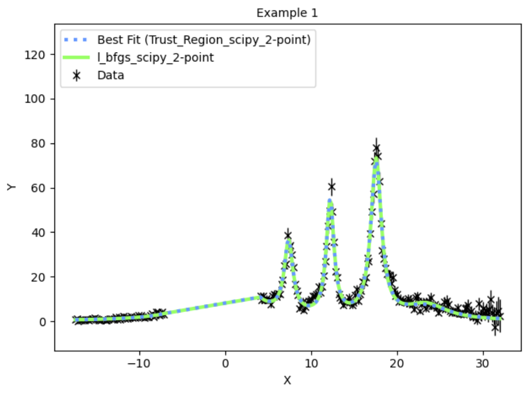


	globalmultistart		mantid									
	l_bfgs: scipy 2-point	scaled_regularisation: scipy 2-point	BFGS: scipy 2-point	Conjugate gradient (Fletcher-Reeves imp.): scipy 2-point	Conjugate gradient (Polak-Ribiere imp.): scipy 2-point	Damped GaussNewton: scipy 2-point	Levenberg-Marquardt: scipy 2-point	Levenberg-MarquardtMD: scipy 2-point	Simplex	SteepestDescent: scipy 2-point	Trust Region: scipy 2-point
Example 1	1100 (1.727e+04)	3808 (5.978e+04)	0.2836 (4.452)^2	6.361 (99.85)^2	8.569 (134.5)^2	0.06371 (1)^2	0.1629 (2.557)	0.07257 (1.139)^2	inf (inf)^3	6.022 (94.52)^1	0.4511 (7.081)^2
Example 2	987.8 (1.071e+04)	3898 (4.225e+04)	0.4183 (4.534)^5	0.1482 (1.607)^5	0.1462 (1.585)^5	0.1039 (1.126)^5	0.2576 (2.792)	0.09226 (1)^2	inf (inf)^3	1.324 (14.35)^5	0.4889 (5.299)^2
Example 3	961.5 (1.557e+04)	3814 (6.176e+04)	0.2197 (3.557)^5	0.2525 (4.088)^5	1.317 (21.33)^5	0.06176 (1)^2	0.1549 (2.508)	0.7045 (11.41)^1	inf (inf)^3	0.1101 (1.782)^5	1.159 (18.76)^5
Example 4	888.4 (2.071e+04)	3777 (8.806e+04)^5	0.2815 (6.562)^5	0.9145 (21.32)^5	0.9735 (22.7)^5	0.04289 (1)^5	0.08552 (1.994)	0.05462 (1.273)^2	inf (inf)^3	0.7052 (16.44)^5	7.896 (184.1)^5
Example 5	1012 (3.048e+04)	3849 (1.159e+05)	0.1981 (5.967)^5	1.066 (32.09)^5	0.8875 (26.73)^5	0.03321 (1)^5	0.1413 (4.255)	0.03501 (1.054)^2	inf (inf)^3	1.051 (31.64)^5	0.5854 (17.63)^2

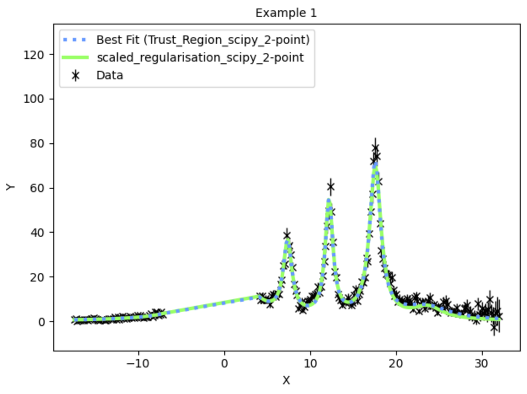
Figure 8: FitBenchmarking runtime in seconds on Crystal Field Problem 1 (values in brackets are relative to the best result). Superscript meanings: 1 - maximum number of iterations exceeded, 2 - unable to converge to solution, 3 - internal error, 5 - solution does not respect parameter bounds.

3.4.1 Results for Example 1

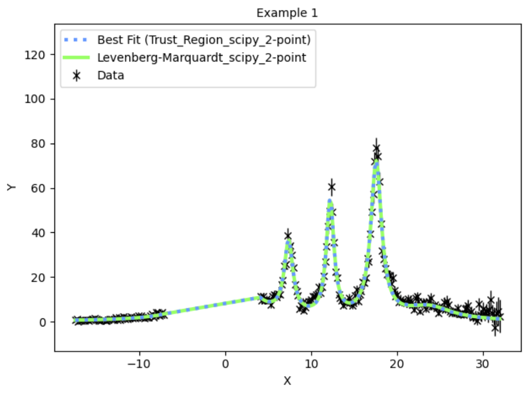
For this example, solvers requiring initial points used initial parameters that were the best guess and close to the ideal parameters.



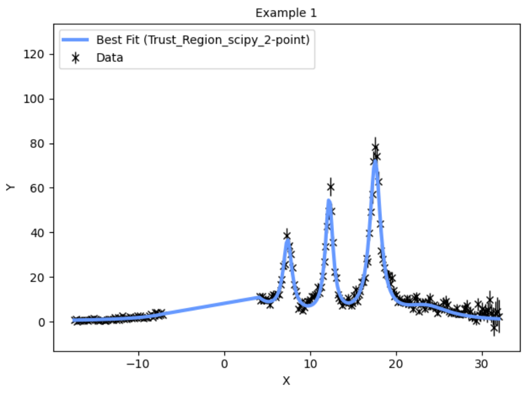
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

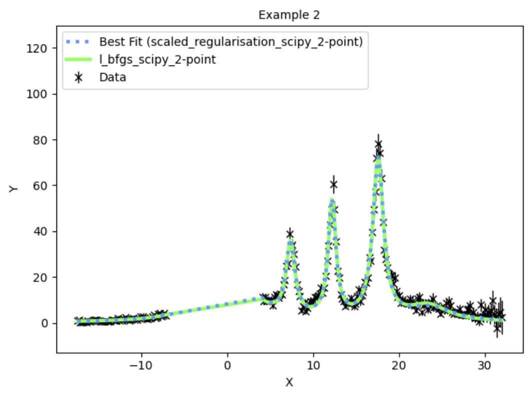


(d) Trust Region

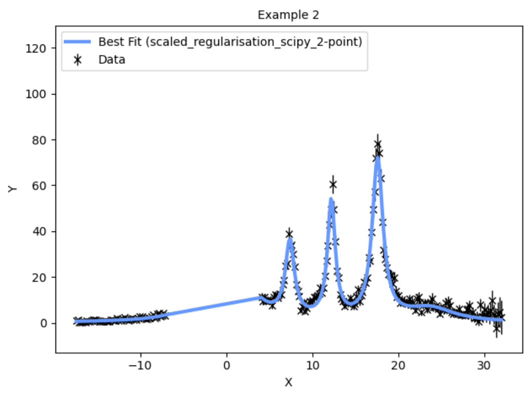
Figure 9: Multi-start and Mantid fitting of Example 1
(Best overall fit for the example is shown in blue)

3.4.2 Results for Example 2

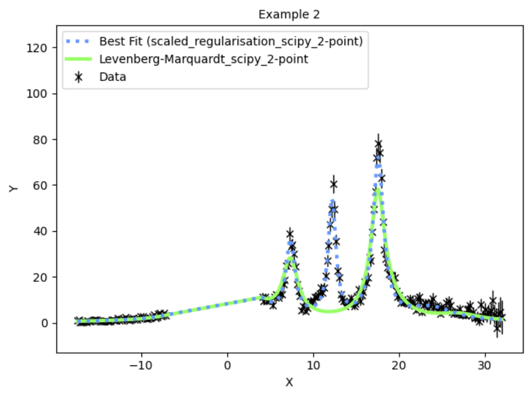
In this example, solvers requiring initial points used initial parameters which were not perfect (rounded versions of initial parameters in Example 1).



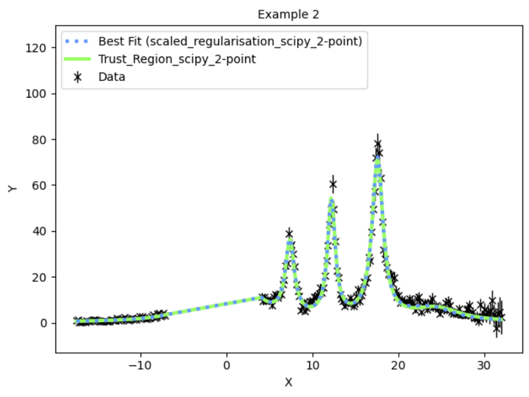
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

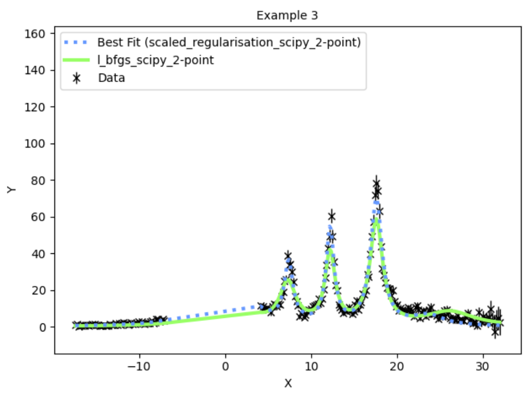


(d) Trust Region

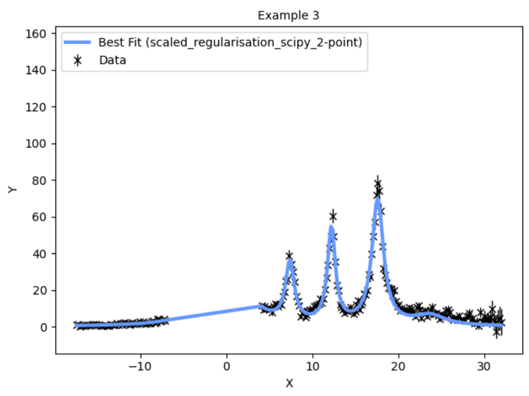
Figure 10: Multi-start and Mantid fitting of Example 2
(Best overall fit for the example is shown in blue)

3.4.3 Results for Example 3

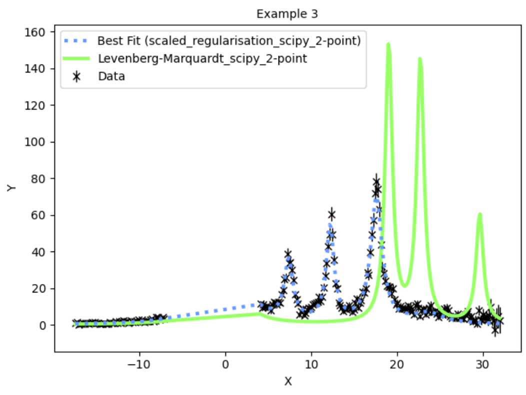
For this Example (and for Example 4), for solvers requiring initial starting points, the initial parameters were chosen from a sample of parameters generated using Latin Hypercube Sampling.



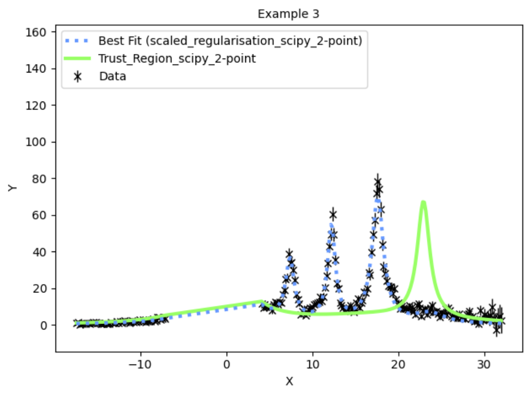
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

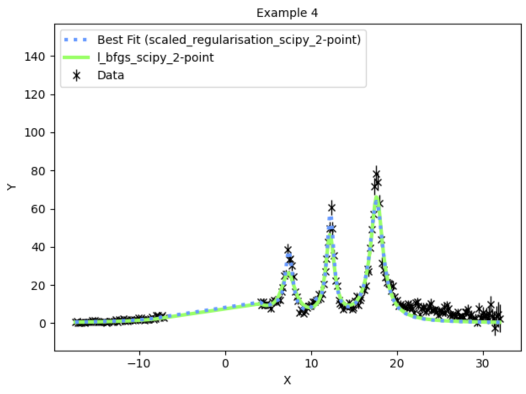


(d) Trust Region

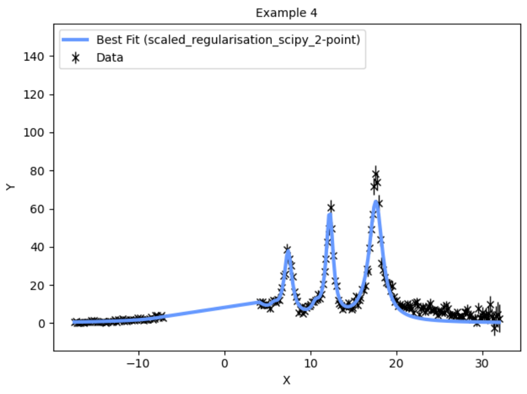
Figure 11: Multi-start and Mantid fitting of Example 3
(Best overall fit for the example is shown in blue)

3.4.4 Results for Example 4

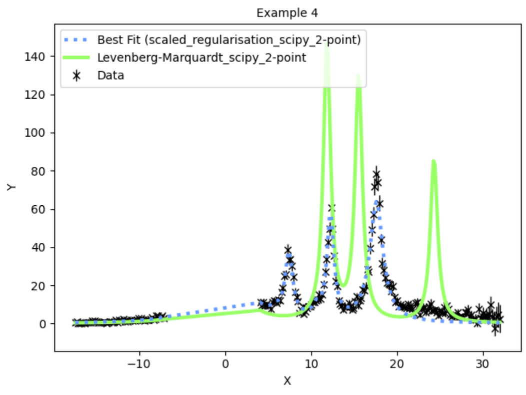
Similar to Example 3, for solvers requiring initial starting points, the initial parameters were generated randomly using Latin Hypercube Sampling.



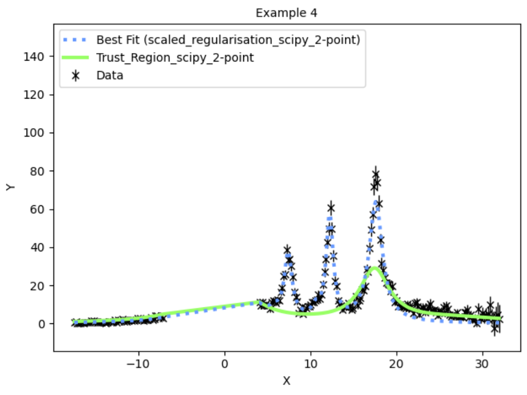
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

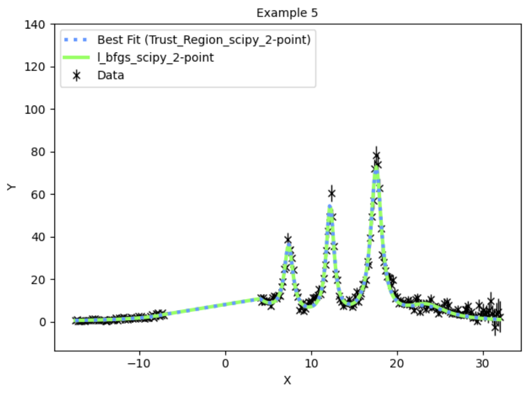


(d) Trust Region

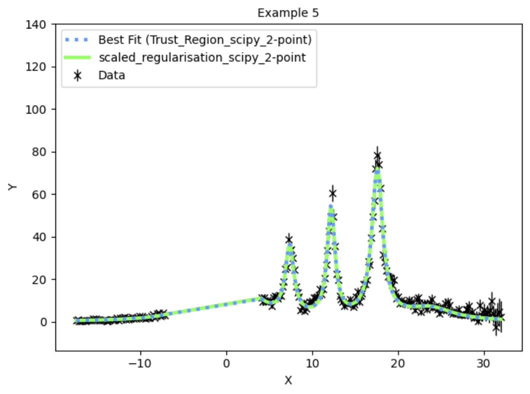
Figure 12: Multi-start and Mantid fitting of Example 4
(Best overall fit for the example is shown in blue)

3.4.5 Results for Example 5

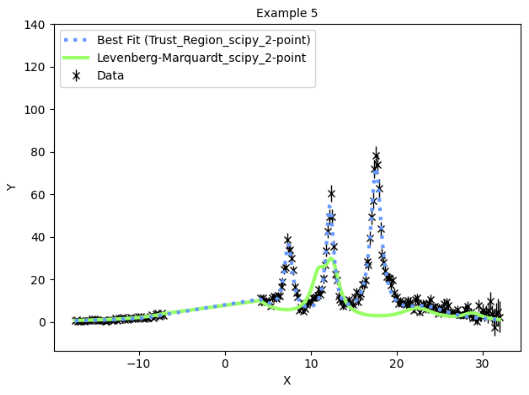
For solvers requiring initial starting points, the initial parameters have been defined as the perturbed starting parameters from Example 1.



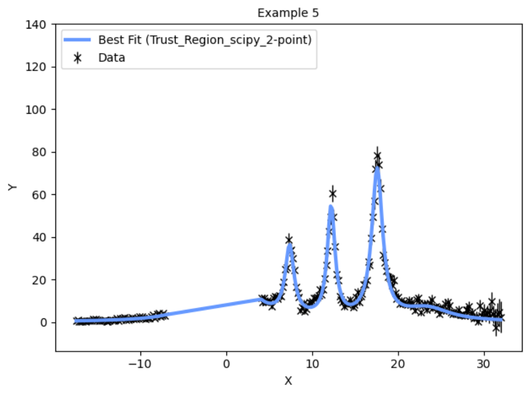
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt



(d) Trust Region

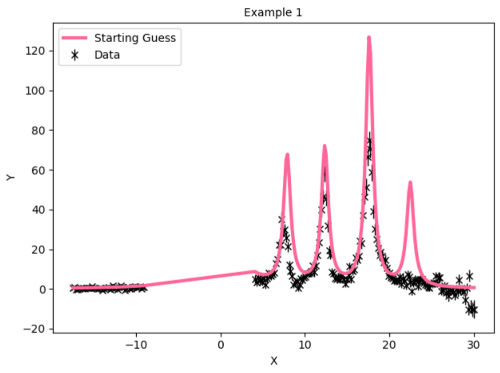
Figure 13: Multi-start and Mantid fitting of Example 5
(Best overall fit for the example is shown in blue)

3.5 Test Problem 2 Initial Points

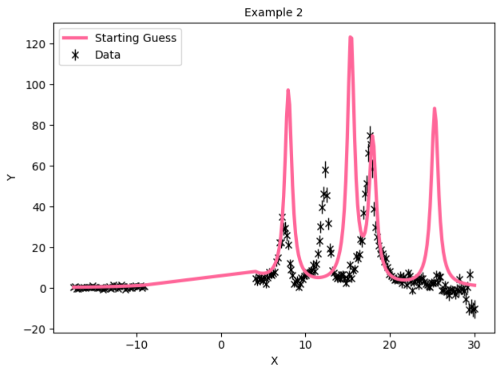
To compare performance of local versus global optimization algorithms on the second crystal field test problem, we used ten different initial optimization starting points for the local optimization algorithms (note that these were not used by the global multi-start algorithms which used 100 Latin Hypercube Samples as starting points):

1. Ideal initial point at 5K
2. Imperfect initial point at 5K (rounded version of above)
3. Initial point selected at random using LHS at 5K
4. Another initial point selected at random using LHS at 5K
5. Perturbed ideal initial point at 5K

6–10. Same as 1–5. above but at 10K

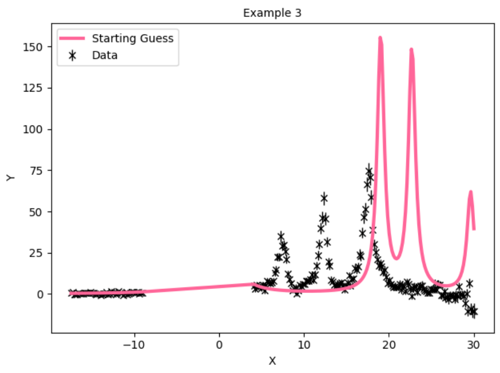


(a) Example 1

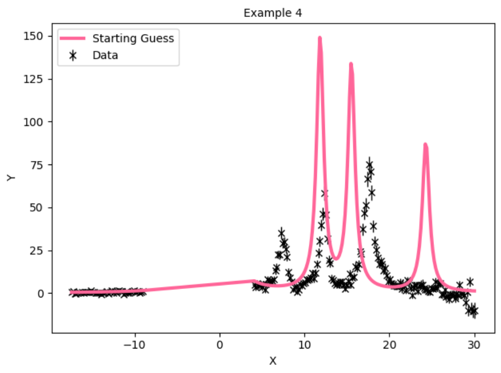


(b) Example 2

Figure 14: Plot of Initial Points for Examples 1 and 2

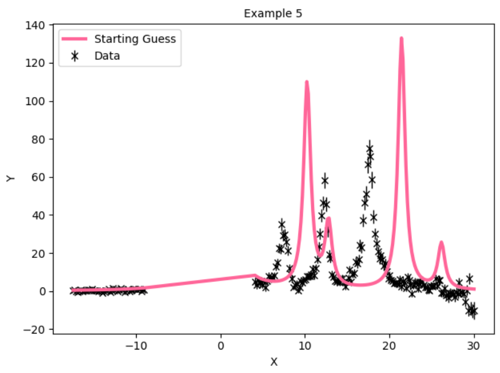


(a) Example 3

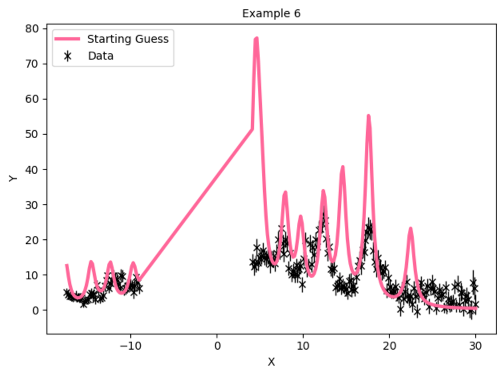


(b) Example 4

Figure 15: Plot of Initial Points for Examples 3 and 4

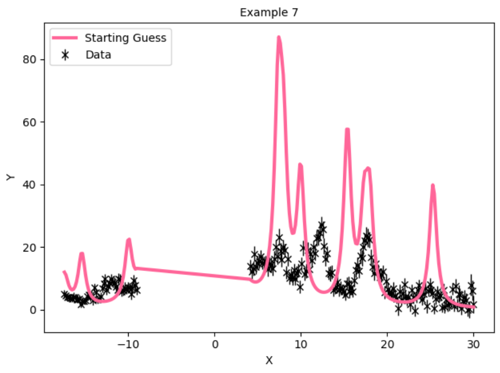


(a) Example 5

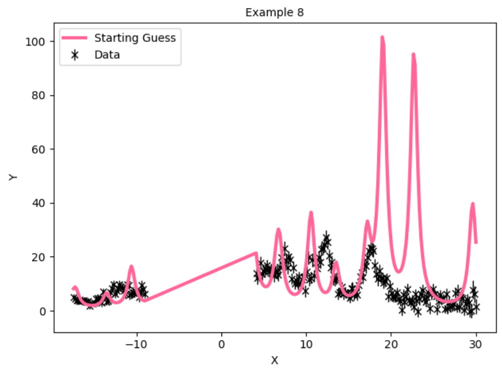


(b) Example 6

Figure 16: Plot of Initial Points for Examples 5 and 6

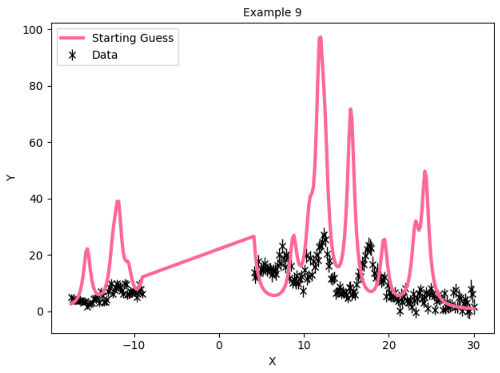


(a) Example 7

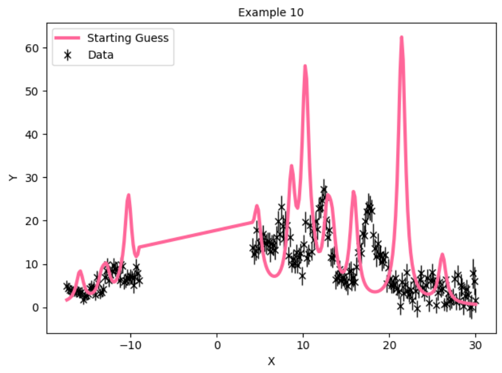


(b) Example 8

Figure 17: Plot of Initial Points for Examples 7 and 8



(a) Example 9



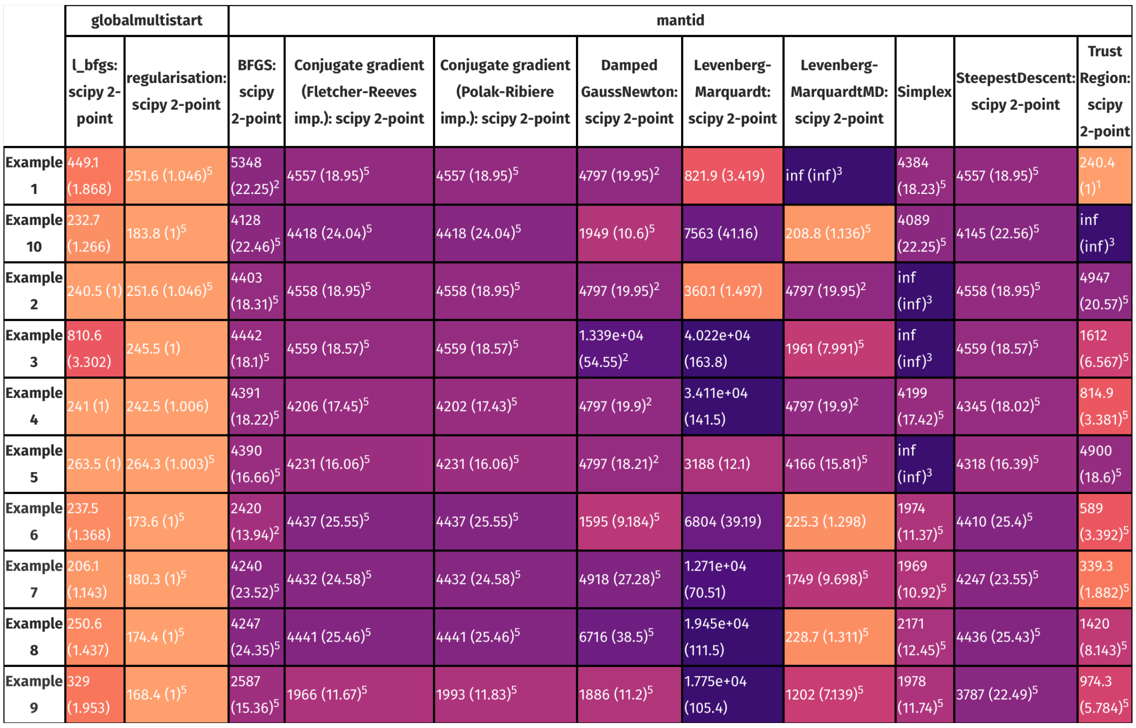
(b) Example 10

Figure 18: Plot of Initial Points for Examples 9 and 10

3.6 Test Problem 2 Results

Figures 19 and 20 show the performance in terms of accuracy (the value of the cost function f at the minimiser x^*) and the CPU runtime in seconds. As we can see from both figures, the Simplex method had raised an internal error (indicated by the 3 in the superscript) for three of the examples and did not return any parameters. Similarly, internal errors were raised by the Levenberg-MarquardtMD method for

Example 1 and the Trust Region method for Example 10. From Figure 19, we can see that the Levenberg-Marquardt method shows good results for Example 2, the Levenberg-MarquardtMD method shows good results for Examples 6, 8 and 10, and the Trust Region method shows good results for Examples 1 and 7, all being comparable to the the Regularisation method on these examples (which again uses scaled parameters). However, for all the other examples the Levenberg-Marquardt, Levenberg-MarquardtMD and Trust Region methods do not perform as well as the Regularisation method. The global multi-start algorithms perform better for these examples, with the Regularisation algorithm consistently finding global minima that fit the crystal field data well across all the examples. Figures 10–25 show the crystal field parameter fits for the multi-start global optimization, Levenberg-Marquardt, and Trust Region methods in the next subsections.



	globalmultistart		mantid								
	l_bfgs: scipy 2-point	regularisation: scipy 2-point	BFGS: scipy 2-point	Conjugate gradient (Fletcher-Reeves imp.): scipy 2-point	Conjugate gradient (Polak-Ribiere imp.): scipy 2-point	Damped GaussNewton: scipy 2-point	Levenberg-Marquardt: scipy 2-point	Levenberg-MarquardtMD: scipy 2-point	Simplex	SteepestDescent: scipy 2-point	Trust Region: scipy 2-point
Example 1	449.1 (1.868)	251.6 (1.046)^5	5348 (22.25)^2	4557 (18.95)^5	4557 (18.95)^5	4797 (19.95)^2	821.9 (3.419)	inf (inf)^3	4384 (18.23)^5	4557 (18.95)^5	240.4 (1)^1
Example 10	232.7 (1.266)	183.8 (1)^5	4128 (22.46)^5	4418 (24.04)^5	4418 (24.04)^5	1949 (10.6)^5	7563 (41.16)	208.8 (1.136)^5	4089 (22.25)^5	4145 (22.56)^5	inf (inf)^3
Example 2	240.5 (1)	251.6 (1.046)^5	4403 (18.31)^5	4558 (18.95)^5	4558 (18.95)^5	4797 (19.95)^2	360.1 (1.497)	4797 (19.95)^2	inf (inf)^3	4558 (18.95)^5	4947 (20.57)^5
Example 3	810.6 (3.302)	245.5 (1)	4442 (18.1)^5	4559 (18.57)^5	4559 (18.57)^5	1.339e+04 (54.55)^2	4.022e+04 (163.8)	1961 (7.991)^5	inf (inf)^3	4559 (18.57)^5	1612 (6.567)^5
Example 4	241 (1)	242.5 (1.006)	4391 (18.22)^5	4206 (17.45)^5	4202 (17.43)^5	4797 (19.9)^2	3.411e+04 (141.5)	4797 (19.9)^2	4199 (17.42)^5	4345 (18.02)^5	814.9 (3.381)^5
Example 5	263.5 (1)	264.3 (1.003)^5	4390 (16.66)^5	4231 (16.06)^5	4231 (16.06)^5	4797 (18.21)^2	3188 (12.1)	4166 (15.81)^5	inf (inf)^3	4318 (16.39)^5	4900 (18.6)^5
Example 6	237.5 (1.368)	173.6 (1)^5	2420 (13.94)^2	4437 (25.55)^5	4437 (25.55)^5	1595 (9.184)^5	6804 (39.19)	225.3 (1.298)	1974 (11.37)^5	4410 (25.4)^5	589 (3.392)^5
Example 7	206.1 (1.143)	180.3 (1)^5	4240 (23.52)^5	4432 (24.58)^5	4432 (24.58)^5	4918 (27.28)^5	1.271e+04 (70.51)	1749 (9.698)^5	1969 (10.92)^5	4247 (23.55)^5	339.3 (1.882)^5
Example 8	250.6 (1.437)	174.4 (1)^5	4247 (24.35)^5	4441 (25.46)^5	4441 (25.46)^5	6716 (38.5)^5	1.945e+04 (111.5)	228.7 (1.311)^5	2171 (12.45)^5	4436 (25.43)^5	1420 (8.143)^5
Example 9	329 (1.953)	168.4 (1)^5	2587 (15.36)^5	1966 (11.67)^5	1993 (11.83)^5	1886 (11.2)^5	1.775e+04 (105.4)	1202 (7.139)^5	1978 (11.74)^5	3787 (22.49)^5	974.3 (5.784)^5

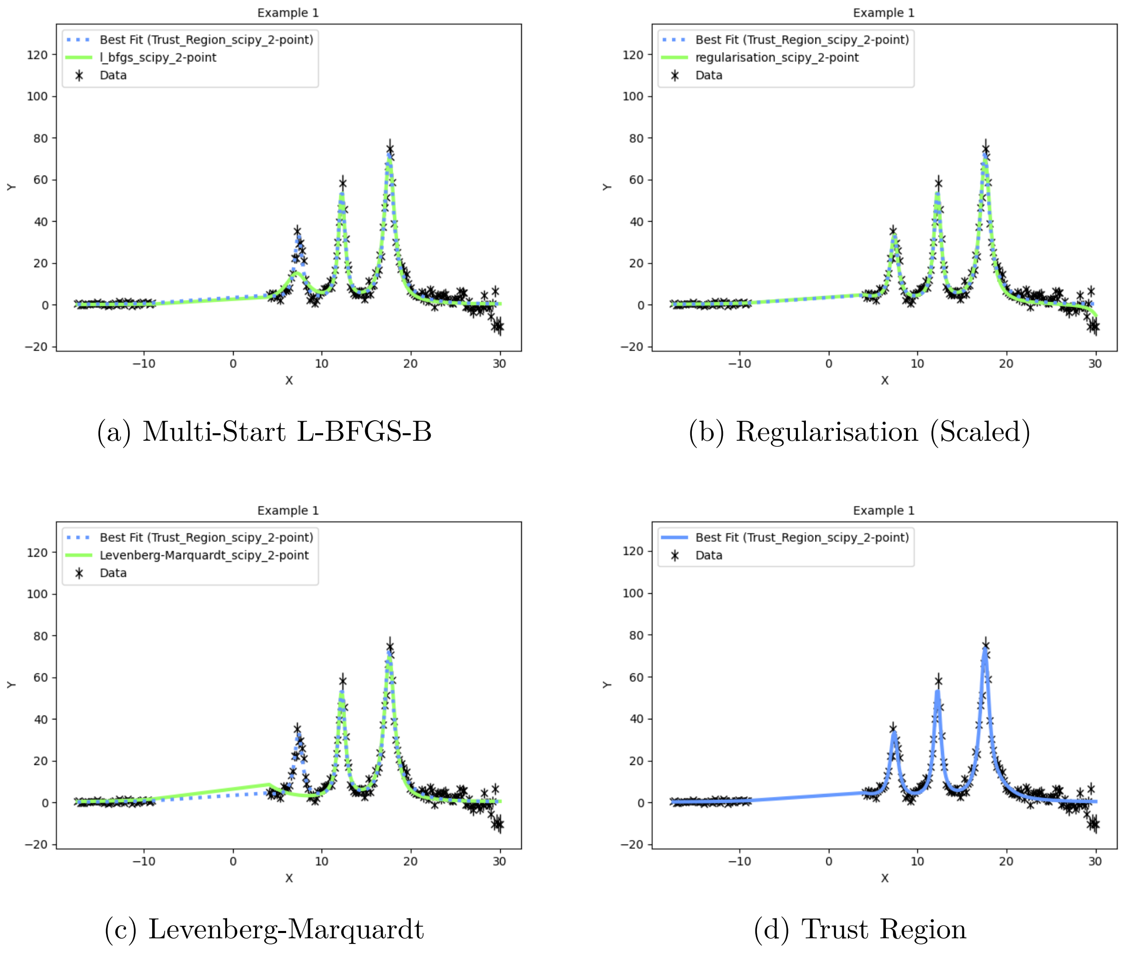
Figure 19: FitBenchmarking accuracy $f(x^*)$ on Crystal Field Problem 2 (values in brackets are relative to the best result). Superscript meanings: 1 - maximum number of iterations exceeded, 2 - unable to converge to solution, 3 - internal error, 5 - solution does not respect parameter bounds.

	globalmultistart		mantid								
	l_bfgs: scipy 2-point	regularisation: scipy 2-point	BFGS: scipy 2-point	Conjugate gradient (Fletcher-Reeves imp.): scipy 2-point	Conjugate gradient (Polak-Ribiere imp.): scipy 2-point	Damped GaussNewton: scipy 2-point	Levenberg-Marquardt: scipy 2-point	Levenberg-MarquardtMD: scipy 2-point	Simplex	SteepestDescent: scipy 2-point	Trust Region: scipy 2-point
Example 1	897.2 (9753)	4366 $(4.746e+04)^5$	0.2538 $(2.76)^2$	0.1829 $(1.988)^5$	0.1831 $(1.991)^5$	0.09199 $(1)^2$	0.2076 (2.256)	inf $(inf)^3$	1.512 $(16.44)^5$	0.09891 $(1.075)^5$	9.26 $(100.7)^1$
Example 10	1821 (2.023e+04)	8903 $(9.89e+04)^5$	0.2058 $(2.287)^5$	0.2862 $(3.179)^5$	0.2778 $(3.086)^5$	0.09002 $(1)^5$	0.2452 (2.724)	0.7047 $(7.829)^5$	2.833 $(31.47)^5$	0.6364 $(7.07)^5$	inf $(inf)^3$
Example 2	917.2 (1.005e+04)	4397 $(4.816e+04)^5$	0.3675 $(4.025)^5$	0.2256 $(2.471)^5$	0.2209 $(2.419)^5$	0.1042 $(1.142)^2$	0.2397 (2.625)	0.0913 $(1)^2$	inf $(inf)^3$	0.09884 $(1.083)^5$	7.683 $(84.15)^5$
Example 3	789 (9707)	4280 (5.265e+04)	0.1957 $(2.408)^5$	0.1824 $(2.244)^5$	0.18 $(2.215)^5$	0.08128 $(1)^2$	0.1443 (1.776)	0.411 $(5.057)^5$	inf $(inf)^3$	0.09974 $(1.227)^5$	1.584 $(19.49)^5$
Example 4	819.2 (1.496e+04)	4182 (7.639e+04)	0.2186 $(3.992)^5$	1.24 $(22.65)^5$	1.676 $(30.61)^5$	0.08267 $(1.51)^2$	0.1285 (2.347)	0.05475 $(1)^2$	2.102 $(38.4)^5$	0.7132 $(13.03)^5$	7.745 $(141.5)^5$
Example 5	882 (1.233e+04)	4324 $(6.043e+04)^5$	0.2444 $(3.415)^5$	0.8294 $(11.59)^5$	0.8451 $(11.81)^5$	0.07156 $(1)^2$	0.1299 (1.815)	0.39 $(5.451)^5$	inf $(inf)^3$	5.702 $(79.68)^5$	9.15 $(127.9)^5$
Example 6	1715 (2.295e+04)	8422 $(1.127e+05)^5$	0.3469 $(4.641)^2$	0.2874 $(3.845)^5$	0.297 $(3.974)^5$	0.07474 $(1)^5$	0.2462 (3.294)	0.6171 (8.256)	3.234 $(43.27)^5$	0.6662 $(8.913)^5$	11.47 $(153.4)^5$
Example 7	1645 (2.18e+04)	8992 $(1.192e+05)^5$	0.2871 $(3.806)^5$	0.1998 $(2.649)^5$	0.2041 $(2.705)^5$	0.07543 $(1)^5$	0.293 (3.884)	0.2165 $(2.87)^5$	3.301 $(43.76)^5$	2.589 $(34.32)^5$	0.7534 $(9.987)^5$
Example 8	1663 (2.056e+04)	8705 $(1.077e+05)^5$	0.2647 $(3.273)^5$	0.2222 $(2.748)^5$	0.2232 $(2.761)^5$	0.08085 $(1)^5$	0.2632 (3.255)	0.75 $(9.277)^5$	3.322 $(41.09)^5$	0.6565 $(8.119)^5$	3.316 $(41.01)^5$
Example 9	1758 (2.313e+04)	8816 $(1.16e+05)^5$	0.415 $(5.462)^5$	1.808 $(23.79)^5$	4.219 $(55.52)^5$	0.07599 $(1)^5$	0.2175 (2.863)	0.5466 $(7.193)^5$	3.338 $(43.93)^5$	0.6962 $(9.162)^5$	5.059 $(66.58)^5$

Figure 20: FitBenchmarking runtime in seconds on Crystal Field Problem 2 (values in brackets are relative to the best result). Superscript meanings: 1 - maximum number of iterations exceeded, 2 - unable to converge to solution, 3 - internal error, 5 - solution does not respect parameter bounds.

3.6.1 Results for Example 1

For this example, solvers requiring initial points used initial parameters that were the best guess and close to the ideal parameters.



(a) Multi-Start L-BFGS-B

(b) Regularisation (Scaled)

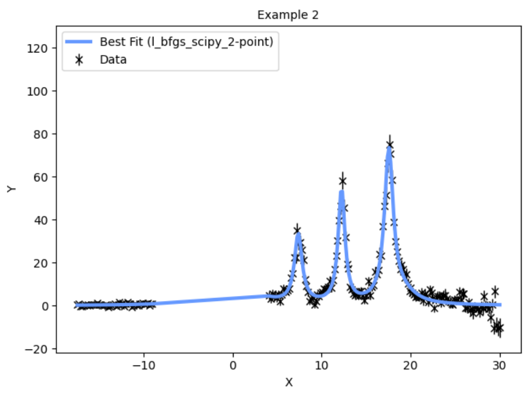
(c) Levenberg-Marquardt

(d) Trust Region

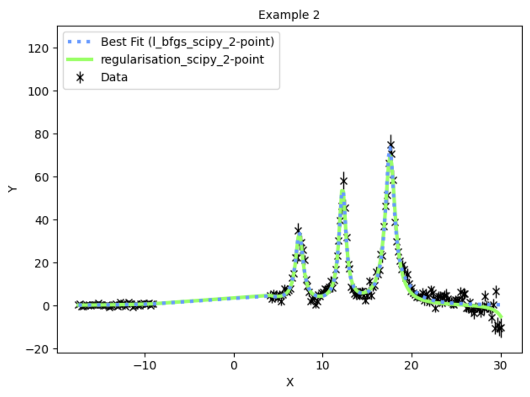
Figure 21: Multi-start and Mantid fitting of Example 1
(Best overall fit for the example is shown in blue)

3.6.2 Results for Example 2

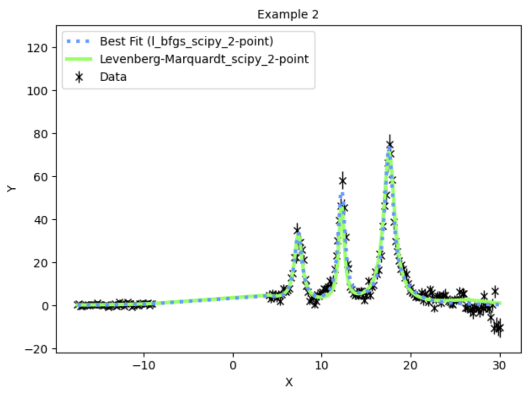
In this example, solvers requiring initial points used initial parameters which were not perfect (rounded versions of initial parameters in Example 1).



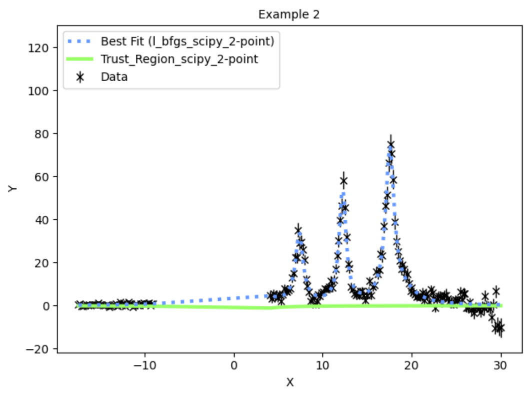
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

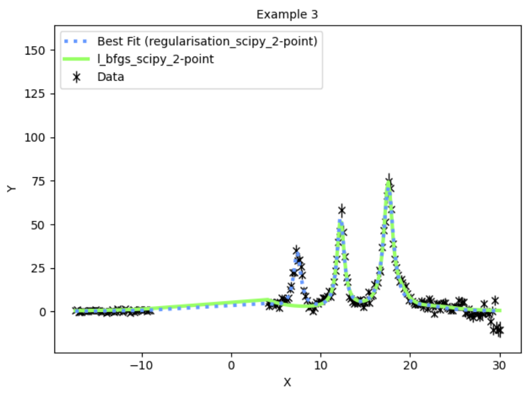


(d) Trust Region

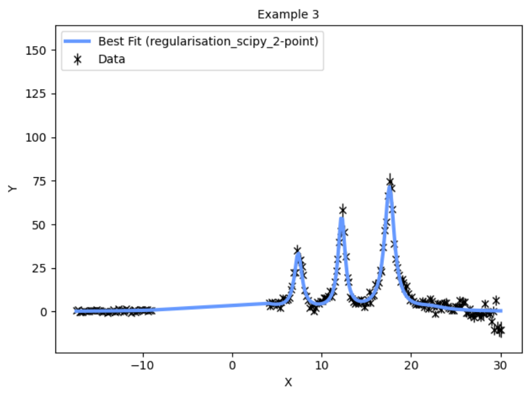
Figure 22: Multi-start and Mantid fitting of Example 2
(Best overall fit for the example is shown in blue)

3.6.3 Results for Example 3

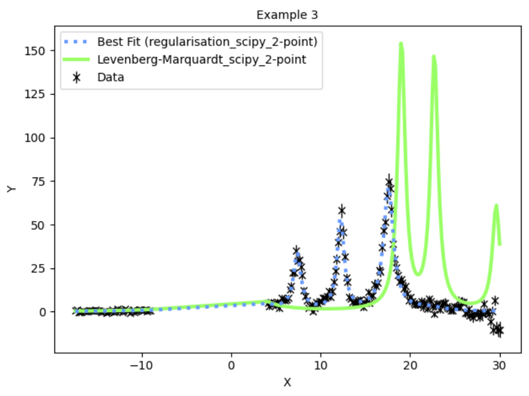
For this Example (and for Example 4), for solvers requiring initial starting points, the initial parameters were chosen from a sample of parameters generated using Latin Hypercube Sampling.



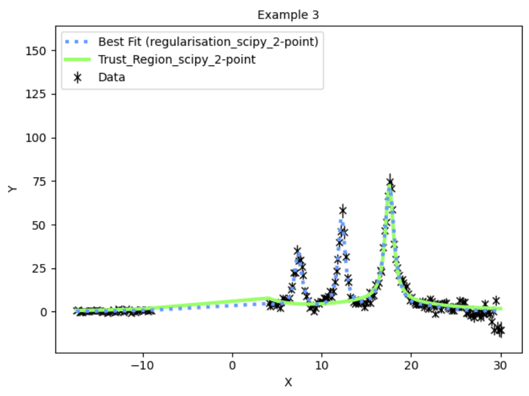
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

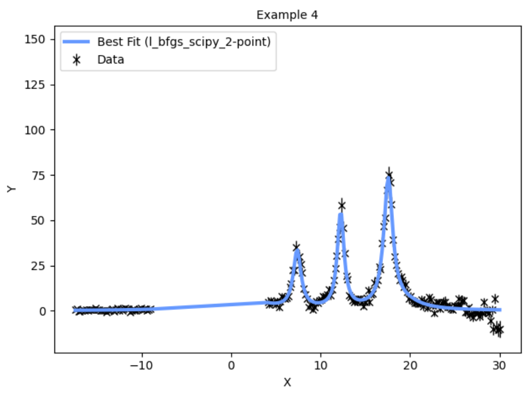


(d) Trust Region

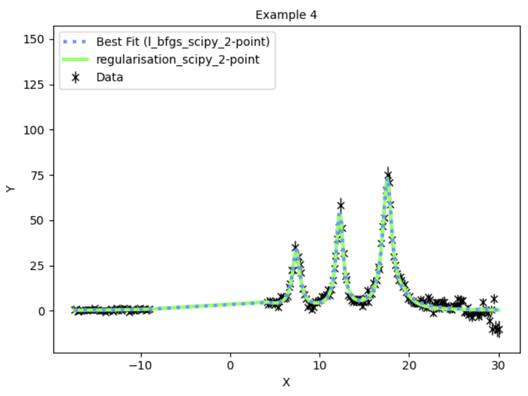
Figure 23: Multi-start and Mantid fitting of Example 3
(Best overall fit for the example is shown in blue)

3.6.4 Results for Example 4

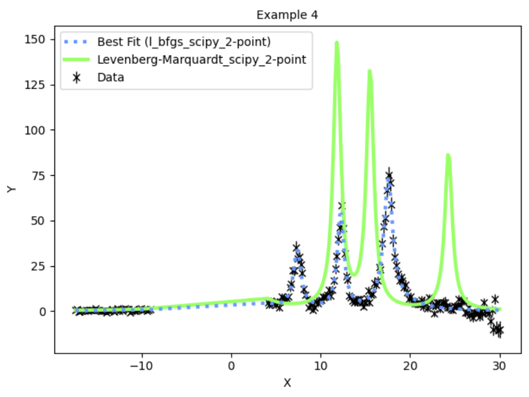
Similar to Example 3, for solvers requiring initial starting points, the initial parameters were generated randomly using Latin Hypercube Sampling.



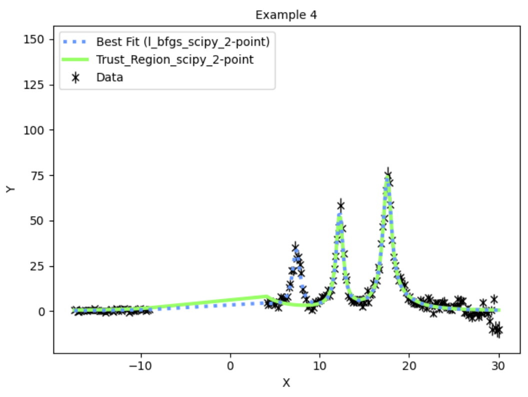
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

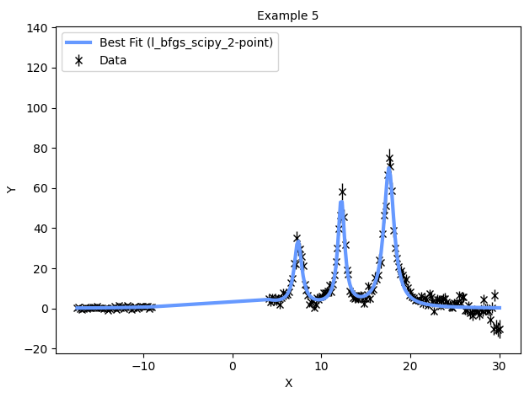


(d) Trust Region

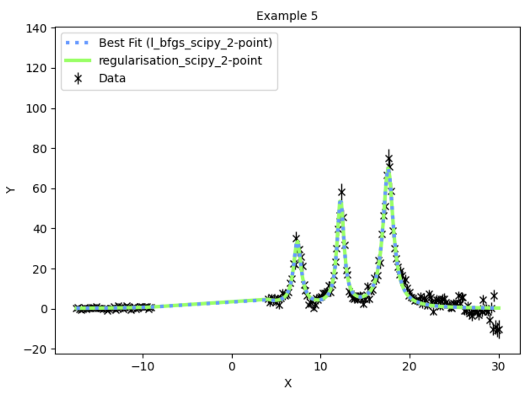
Figure 24: Multi-start and Mantid fitting of Example 4
(Best overall fit for the example is shown in blue)

3.6.5 Results for Example 5

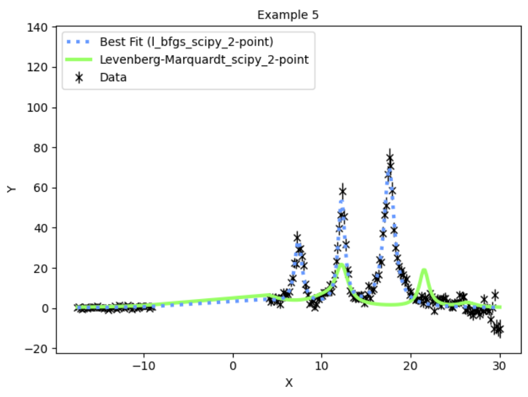
For solvers requiring initial starting points, the initial parameters have been defined as the perturbed starting parameters from Example 1.



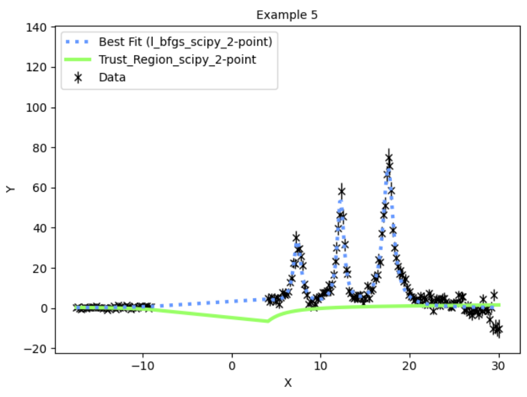
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

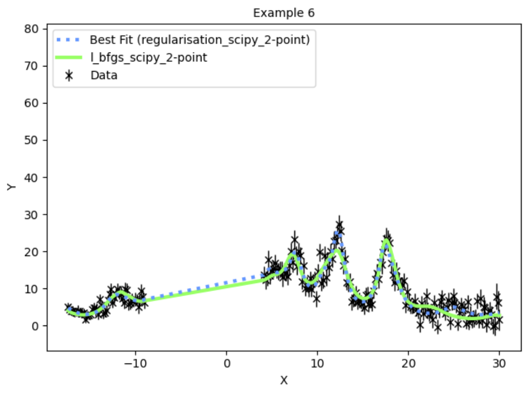


(d) Trust Region

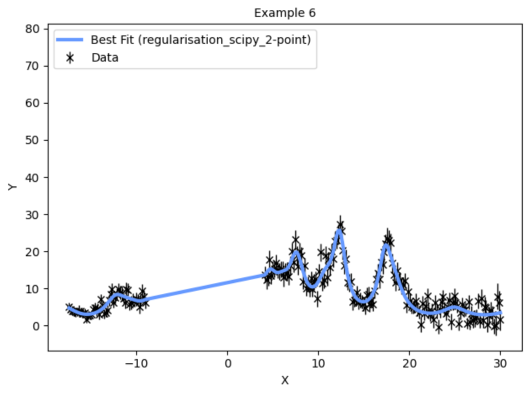
Figure 25: Multi-start and Mantid fitting of Example 5
(Best overall fit for the example is shown in blue)

3.6.6 Results for Example 6

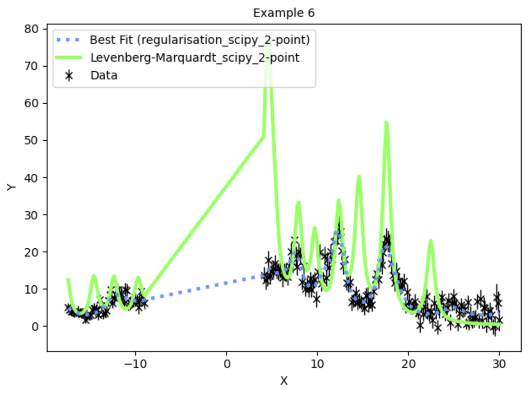
For this example, solvers requiring initial points used initial parameters that were the best guess and close to the ideal parameters.



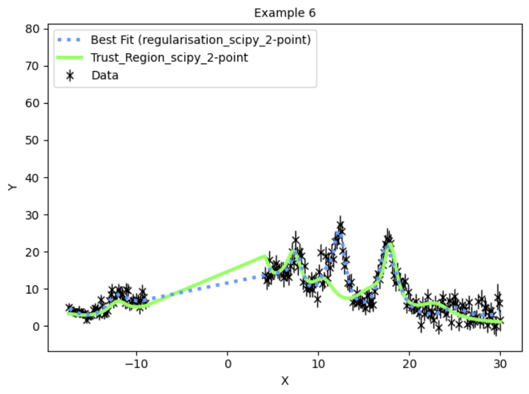
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

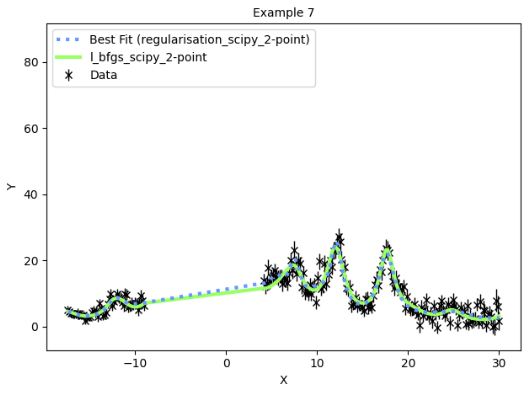


(d) Trust Region

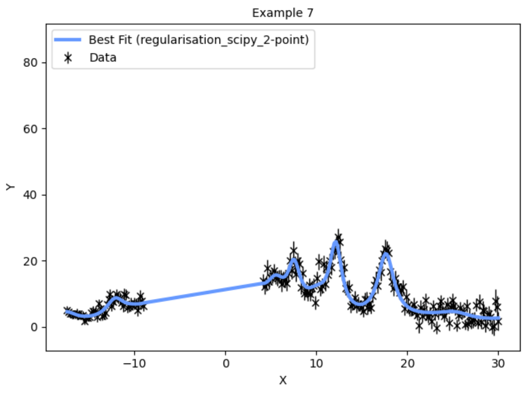
Figure 26: Multi-start and Mantid fitting of Example 6
(Best overall fit for the example is shown in blue)

3.6.7 Results for Example 7

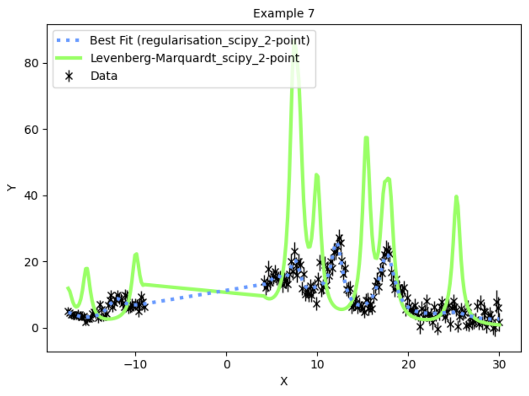
In this example, solvers requiring initial points used initial parameters which were not perfect (rounded versions of initial parameters in Example 6).



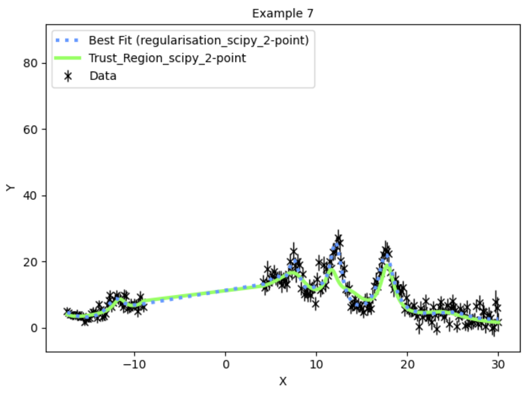
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

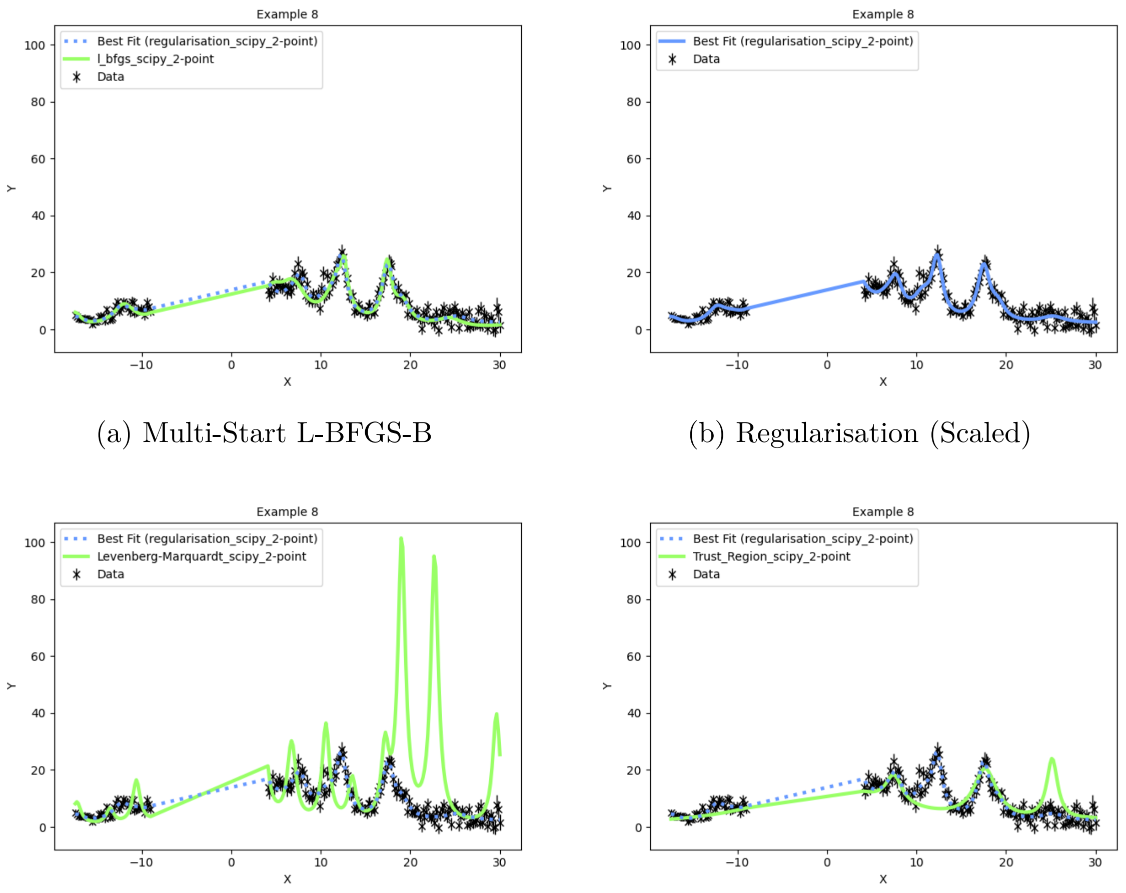


(d) Trust Region

Figure 27: Multi-start and Mantid fitting of Example 7
(Best overall fit for the example is shown in blue)

3.6.8 Results for Example 8

For this Example (and for Example 9), for solvers requiring initial starting points, the initial parameters were chosen from a sample of parameters generated using Latin Hypercube Sampling.



(a) Multi-Start L-BFGS-B

(b) Regularisation (Scaled)

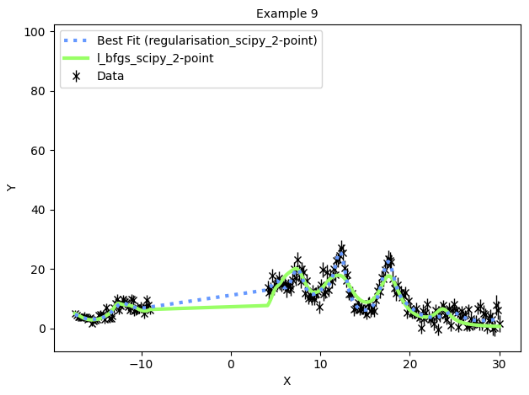
(c) Levenberg-Marquardt

(d) Trust Region

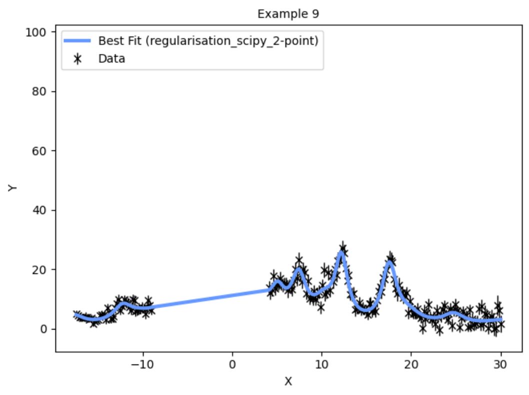
Figure 28: Multi-start and Mantid fitting of Example 8
(Best overall fit for the example is shown in blue)

3.6.9 Results for Example 9

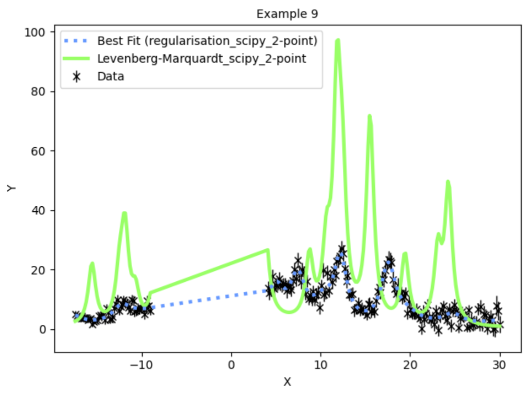
Similar to Example 8, for solvers requiring initial starting points, the initial parameters were generated randomly using Latin Hypercube Sampling.



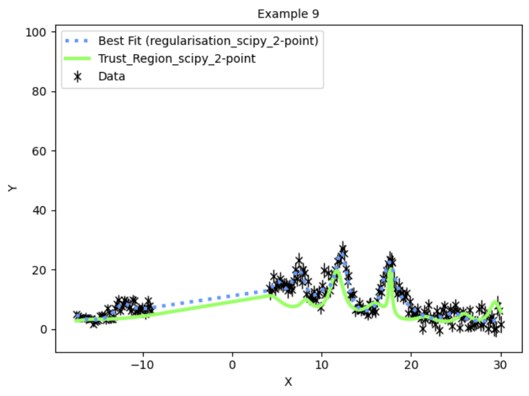
(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

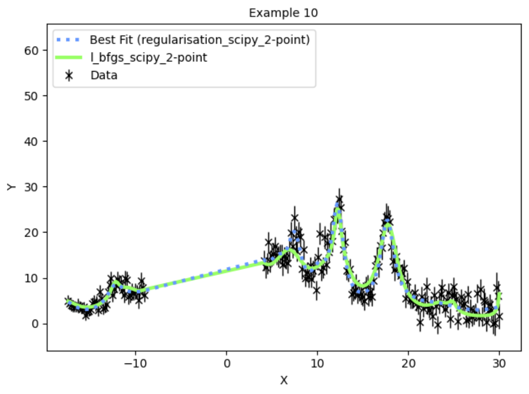


(d) Trust Region

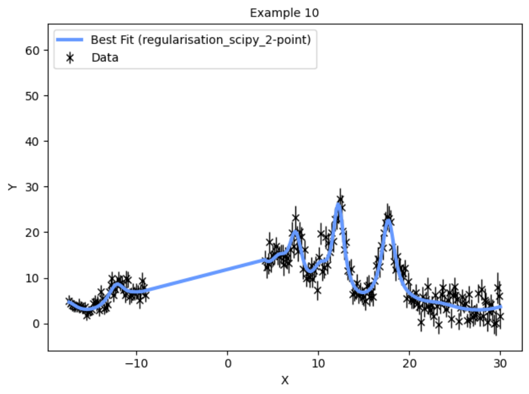
Figure 29: Multi-start and Mantid fitting of Example 9
(Best overall fit for the example is shown in blue)

3.6.10 Results for Example 10

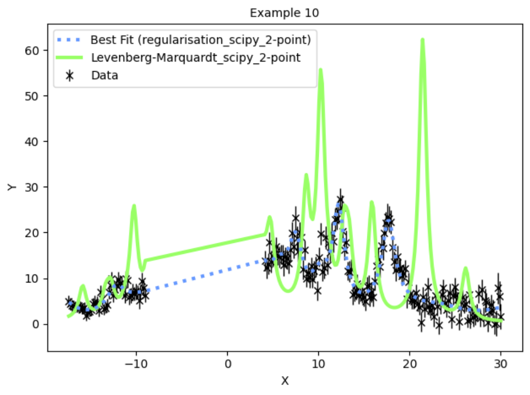
For solvers requiring initial starting points, the initial parameters have been defined as the perturbed starting parameters from Example 6.



(a) Multi-Start L-BFGS-B



(b) Regularisation (Scaled)



(c) Levenberg-Marquardt

Figure 30: Multi-start and Mantid fitting of Example 10
(Best overall fit for the example is shown in blue)

4 Conclusions and Future Work

Overall, our benchmarking results demonstrate that the global multi-start Regularisation algorithm is able to find ideal parameters for all the crystal field examples tested, while the Trust Region, Levenberg-Marquardt and Levenberg-MarquardtMD methods are the only Mantid solvers that can find ideal parameters for some of the examples. The results suggest that if the initial parameters are close to the global minimiser, then Trust Region will be better able to fit crystal field data than Levenberg-Marquardt, although Trust Region can sometimes struggle with initial parameters far away from the global minimiser.

The global optimization algorithms, however, are not dependent on the initial parameters and have been able to fit all the crystal field spectra using random initial starting points. Therefore, the multi-start Regularisation algorithm is a good candidate for improving Mantid's performance in crystal field parameter fitting because it removes the need to carefully choose initial parameters.

The trade-off, however, is the runtime as the global optimization algorithms are several orders of magnitude slower than the Mantid local optimization solvers. However, most of the runtime for the global optimization algorithms is taken up implementing two nested loops in Python and evaluating every set of initial parameters, while Mantid's solvers are already optimized in C++, and evaluate one set of initial parameters. Implementing and optimizing the Regularisation algorithm in C++ should reduce its runtime. Despite the downside of increased runtime, the Regularisation algorithm removes the need to have precise initial parameters and clearly outperforms the local solvers in Mantid in these cases.

Crystal field problems are highly sensitive to the choice of parameters, and the majority of local optimization solvers in Mantid struggle to fit crystal field data unless the initial parameters are close to the optimal parameters. By using a global optimization approach, we can consistently find the global minimiser without relying on expert insight to determine ideal initial starting points.

Now that we have a global optimization algorithm (namely the multi-start Regularisation algorithm) implemented in Python and benchmarked against the solvers in Mantid, the next step is to implement this algorithm in C++, so that it can be tested and potentially integrated into Mantid.

Acknowledgements

This work was supported by the Ada Lovelace Centre, and this work also used the STFC Cloud. We are grateful to the ISIS team for providing their expertise, supplying us with their datasets for testing, and for helpful feedback on a draft of this report. We would also like to thank the FitBenchmarking team for helping to integrate the crystal field fitting functionality into FitBenchmarking.

References

[1] Mantid, “Manipulation and Analysis Toolkit for Instrument Data.” `https://www.mantidproject.org`, 2013.

[2] O. Arnold, J. C. Bilheux, J. M. Borreguero, A. Buts, S. I. Campbell, L. Chapon, M. Doucet, N. Draper, R. Ferraz Leal, M. A. Gigg, V. E. Lynch, A. Markvardsen, D. J. Mikkelson, R. L. Mikkelson, R. Miller, K. Palmen, P. Parker, G. Passos, T. G. Perring, P. F. Peterson, S. Ren, M. A. Reuter, A. T. Savici, J. W. Taylor, R. J. Taylor, R. Tolchenov, W. Zhou, and J. Zikovsky, “Mantid—Data analysis and visualization package for neutron scattering and µSR experiments,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 764, pp. 156–166, 2014.

[3] J. Nocedal and S. J. Wright, *Numerical Optimization.* Springer series in Operations Research, New York: Springer, 1999.

[4] Mantid, “Crystal Field Theory.” `https://docs.mantidproject.org/nightly/concepts/CrystalField.html`.

[5] L. Velázquez, G. N. Phillips, R. A. Tapia, and Y. Zhang, “Selective Search for Global Optimization of Zero or Small Residual Least-Squares Problems: A Numerical Study,” *Computational Optimization and Applications*, vol. 20, no. 3, pp. 299–315, 2001.

[6] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[7] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.

[8] M. A. Bouhlel, J. T. Hwang, N. Bartoli, R. Lafage, J. Morlier, and J. R. R. A. Martins, “A python surrogate modeling framework with derivatives,” *Advances in Engineering Software*, p. 102662, 2019.

[9] M. D. McKay, R. J. Beckman, and W. J. Conover, “A comparison of three methods for selecting values of input variables in the analysis of output from a computer code,” *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[10] N. I. M. Gould, T. Rees, and J. A. Scott, "Convergence and evaluation-complexity analysis of a regularized tensor-Newton method for solving non-linear least-squares problems," *Computational Optimization and Applications*, vol. 73, no. 1, pp. 1–35, 2019.

[11] Mantid, "Crystal Field Examples." `https://www.mantidproject.org/Crystal_Field_Examples.html`.

[12] C. Ritter, D. Adroja, M. Le, Y. Muro, and T. Takabatake, "Magnetic structure and crystal field excitations of NdOs2Al10: a neutron scattering study," *Journal of Physics: Condensed Matter*, vol. 33, no. 18, p. 185802, 2021.

[13] A. Markvardsen, T. Rees, M. Wathen, A. Lister, P. Odagiu, A. Anuchitanukul, T. Farmer, A. Lim, F. Montesino, T. Snow, and A. McCluskey, "FitBenchmarking: an open source Python package comparing data fitting software," *Journal of Open Source Software*, vol. 6, no. 62, p. 3127, 2021.

[14] Mantid, "Fit Minimizers." `https://docs.mantidproject.org/nightly/fitting/fitminimizers/categories/FitMinimizers.html`.