BNL-224587-2023-TECH

C-A/AP/697

Higher-Order Corrections to Optimisers based on Newton's Method

S. Brooks

July 2023

Collider Accelerator Department

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC), Nuclear Physics (NP) (SC-26)

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or any third party's use or the results of such use of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof or its contractors or subcontractors. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Higher-Order Corrections to Optimisers based on Newton's Method

Stephen Brooks[a]

[a] *Collider–Accelerator Department, Brookhaven National Laboratory, Upton, NY, 11973, USA*

Abstract

The Newton, Gauss–Newton and Levenberg–Marquardt methods all use the first derivative of a vector function (the Jacobian) to minimise its sum of squares. When the Jacobian matrix is ill-conditioned, the function varies much faster in some directions than others and the space of possible improvement in sum of squares becomes a long narrow ellipsoid in the linear model. This means that even a small amount of nonlinearity in the problem parameters can cause a proposed point far down the long axis of the ellipsoid to fall outside of the actual curved valley of improved values, even though it is quite nearby. This paper presents a differential equation that 'follows' these valleys, based on the technique of geodesic acceleration, which itself provides a 2^{nd} order improvement to the Levenberg–Marquardt iteration step. Higher derivatives of this equation are computed that allow n^{th} order improvements to the optimisation methods to be derived. These higher-order accelerated methods up to 4^{th} order are tested numerically and shown to provide substantial reduction of both number of steps and computation time.

1. Definitions and Introduction

Consider finding the value of a vector $\mathbf{x}$ such that the vector-valued function $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, noting the input and output of $\mathbf{f}$ might have different dimensions.

Newton's method solves $J(\mathbf{x})(\mathbf{x}_{\text{new}} - \mathbf{x}) = -\mathbf{f}(\mathbf{x})$ where J is the Jacobian matrix. The Gauss–Newton algorithm generalises this to rectangular J using pseudo-inverses that may be calculated using Singular Value Decomposition (SVD). The Levenberg–Marquardt algorithm [1, 2] introduces a damping factor into this pseudo-inverse, which allows progress along 'easier' directions without having to go far in 'difficult' directions that may exhibit nonlinearity.

The remainder of this paper will be written for the simpler Newton method, where J^{-1} is the inverse of the square Jacobian matrix. However, the algorithms derived also work for the Gauss-Newton pseudo-inverse $[J^{-1}]_{GN} = (J^TJ)^{-1}J^T$ and the damped Levenberg–Marquardt version $[J^{-1}]_{LM(\lambda)} = (J^TJ + \lambda I)^{-1}J^T$. The latter is used in numerical tests.

One common source of slow convergence is that J is ill-conditioned, so the optimisation valley is much narrower in some directions than others, while the problem contains some nonlinearity, which may seem small but is amplified once you change into coordinates where J is well-conditioned. This is because even a small amount of nonlinearity can make the long narrow valley stop overlapping with its approximation in the linear model. This reduced range of validity of the linear model means many small steps have to be taken.

Email address: sbrooks@bnl.gov (Stephen Brooks)

2. Natural Optimisation Pathway

The goal of the Newton step is to reduce the error vector $\mathbf{f}$, ideally to zero. For a nonlinear function, the optimisation follows a curved pathway [3, 4] and one natural such pathway is $\mathbf{x}(t)$ defined implicitly by

$$\mathbf{f}(\mathbf{x}(t)) = (1-t)\mathbf{f}(\mathbf{x}(0))$$

for $t \in [0, 1]$. This scales down all components of the error equally and at $t = 1$ it reaches the true solution.

Taking the first derivative of this equation gives

$$\sum_i \partial_i \mathbf{f}(\mathbf{x}(t))\dot{x}_i(t) = J(\mathbf{x}(t))\dot{\mathbf{x}}(t) = -\mathbf{f}(\mathbf{x}(0)) = -\frac{\mathbf{f}(\mathbf{x}(t))}{1-t},$$

which at $t = 0$ makes $\dot{\mathbf{x}}$ equal to the Newton step and to a scaling of it for all $0 < t < 1$. So this pathway is always tangent to the Newton step direction and corresponds to the limit of a Newton algorithm run with steps scaled down to be infinitesimally small.

3. Higher-Order Derivatives

If the pathway curves, one may wonder if longer steps can be taken if the curvature is taken into account. The second and higher derivatives of the equation defining the natural pathway have the form

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}\mathbf{f}(\mathbf{x}(t)) = \mathbf{0}$$

for $n \geq 2$. Multiple derivatives of a function composition ($\mathbf{f} \circ \mathbf{x}$ here) are given by Faà di Bruno's formula [7, 8, 9]

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}\mathbf{f}(\mathbf{x}(t)) = \sum_{\pi \in \Pi_n} \mathbf{f}^{(|\pi|)}(\mathbf{x}(t)) \bigotimes_{p \in \pi} \mathbf{x}^{(|p|)}(t),$$

where Π_n is the set of all partitions of $\{1, 2, ..., n\}$. The d^{th} derivative of the vector function $\mathbf{f}$ is a tensor that takes d vectors as input and outputs a vector, with elements defined by

$$f^{(d)}(\mathbf{x})^i_{j_1 j_2 ... j_d} = \frac{\partial^d f_i(\mathbf{x})}{\partial x_{j_1} \partial x_{j_2} ... \partial x_{j_d}}.$$

Note that $\mathbf{f}^{(1)} = J$. This paper will adopt compact notation where tensor products of vectors $\mathbf{u} \otimes \mathbf{v} \otimes \mathbf{w}$ will be written $\mathbf{uvw}$ so that $(\mathbf{uvw})_{ijk} = u_i v_j w_k$. These may be contracted with the derivative tensor to give a vector written in the form $\mathbf{f}^{(3)}\mathbf{uvw}$, where $(\mathbf{f}^{(3)}\mathbf{uvw})_n = \sum_{i,j,k} f^{(3)n}_{ijk} u_i v_j w_k$.

3.1. Second Order

For $n = 2$, $\Pi_2 = \{\{\{1\}, \{2\}\}, \{\{1, 2\}\}\}$ and

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\mathbf{f}(\mathbf{x}(t)) = \mathbf{f}^{(2)}(\mathbf{x}(t))\dot{\mathbf{x}}(t)\dot{\mathbf{x}}(t) + \mathbf{f}^{(1)}(\mathbf{x}(t))\ddot{\mathbf{x}}(t) = \mathbf{0}$$

$$\Rightarrow \qquad \ddot{\mathbf{x}}(t) = -J^{-1}(\mathbf{x}(t))\mathbf{f}^{(2)}(\mathbf{x}(t))\dot{\mathbf{x}}(t)\dot{\mathbf{x}}(t).$$

This agrees with the well-known [3, 4, 5, 6] quadratic acceleration term for Levenberg–Marquardt if the J^{-1} is replaced by a damped pseudo-inverse $[J^{-1}]_{LM(\lambda)}$.

3.2. Third Order

For conciseness, $\mathbf{x}$ and its derivatives will be evaluated at $t = 0$ unless otherwise stated and $\mathbf{f}$ and its derivatives at $\mathbf{x}$. For $n = 3$,

$$\Pi_3 = \{\{\{1\},\{2\},\{3\}\},\{\{1\},\{2,3\}\},\{\{2\},\{1,3\}\},\{\{3\},\{1,2\}\},\{\{1,2,3\}\}\}$$

and

$$\frac{\mathrm{d}^3}{\mathrm{d}t^3}\mathbf{f} = \mathbf{f}^{(3)}\dot{\mathbf{x}}\dot{\mathbf{x}}\dot{\mathbf{x}} + 3\mathbf{f}^{(2)}\dot{\mathbf{x}}\ddot{\mathbf{x}} + \mathbf{f}^{(1)}\mathbf{x}^{(3)} = \mathbf{0}.$$

This gives the third derivative of $\mathbf{x}$ as

$$\mathbf{x}^{(3)} = -J^{-1}(\mathbf{f}^{(3)}\dot{\mathbf{x}}\dot{\mathbf{x}}\dot{\mathbf{x}} + 3\mathbf{f}^{(2)}\dot{\mathbf{x}}\ddot{\mathbf{x}}).$$

3.3. Fourth Order, Recurrence and General Case

Higher-order expressions can be obtained either from the set partitions Π_n or the equivalent differentiation chain and product rules that obtain $\frac{\mathrm{d}^{n+1}}{\mathrm{d}t^{n+1}}\mathbf{f}$ from $\frac{\mathrm{d}^n}{\mathrm{d}t^n}\mathbf{f}$. The formulae

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{f}^{(n)} = \mathbf{f}^{(n+1)}\dot{\mathbf{x}} \qquad \text{and} \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}^{(n)} = \mathbf{x}^{(n+1)}$$

together with the product rule are enough to generate the full sequence. Starting from $n = 2$,

$$\begin{aligned} \mathbf{f}^{(2)}\dot{\mathbf{x}}\dot{\mathbf{x}} + \mathbf{f}^{(1)}\ddot{\mathbf{x}} &= \mathbf{0} \\ \mathbf{f}^{(3)}\dot{\mathbf{x}}\dot{\mathbf{x}}\dot{\mathbf{x}} + 3\mathbf{f}^{(2)}\dot{\mathbf{x}}\ddot{\mathbf{x}} + \mathbf{f}^{(1)}\mathbf{x}^{(3)} &= \mathbf{0} \\ \mathbf{f}^{(4)}\dot{\mathbf{x}}\dot{\mathbf{x}}\dot{\mathbf{x}}\dot{\mathbf{x}} + 6\mathbf{f}^{(3)}\dot{\mathbf{x}}\dot{\mathbf{x}}\ddot{\mathbf{x}} + 4\mathbf{f}^{(2)}\dot{\mathbf{x}}\mathbf{x}^{(3)} + 3\mathbf{f}^{(2)}\ddot{\mathbf{x}}\ddot{\mathbf{x}} + \mathbf{f}^{(1)}\mathbf{x}^{(4)} &= \mathbf{0} \end{aligned}$$

and so on. A computer algebra system can generate these terms based on a rule like

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{f}^{(n)}\mathbf{x}^{(a)}\mathbf{x}^{(b)}\mathbf{x}^{(c)} =$$

$$\mathbf{f}^{(n+1)}\mathbf{x}^{(1)}\mathbf{x}^{(a)}\mathbf{x}^{(b)}\mathbf{x}^{(c)} + \mathbf{f}^{(n)}\mathbf{x}^{(a+1)}\mathbf{x}^{(b)}\mathbf{x}^{(c)} + \mathbf{f}^{(n)}\mathbf{x}^{(a)}\mathbf{x}^{(b+1)}\mathbf{x}^{(c)} + \mathbf{f}^{(n)}\mathbf{x}^{(a)}\mathbf{x}^{(b)}\mathbf{x}^{(c+1)}$$

and collecting like terms, for example by sorting the $\mathbf{x}$ derivatives in increasing order.

The highest derivative $\mathbf{x}^{(n)}$ may be moved to the other side to get a formula like

$$\mathbf{x}^{(4)} = -J^{-1}(\mathbf{f}^{(4)}\dot{\mathbf{x}}\dot{\mathbf{x}}\dot{\mathbf{x}}\dot{\mathbf{x}} + 6\mathbf{f}^{(3)}\dot{\mathbf{x}}\dot{\mathbf{x}}\ddot{\mathbf{x}} + 4\mathbf{f}^{(2)}\dot{\mathbf{x}}\mathbf{x}^{(3)} + 3\mathbf{f}^{(2)}\ddot{\mathbf{x}}\ddot{\mathbf{x}}),$$

shown for the $n = 4$ case, which expresses it in terms of lower derivatives of $\mathbf{x}$.

4. Taking Finite Steps

The derivatives $\mathbf{x}^{(n)}$ calculated above can produce a corrected higher-order step using the Taylor series of $\mathbf{x}$ around $t = 0$

$$\mathbf{x}(\epsilon) = \sum_{n=0}^{\infty} \frac{1}{n!}\epsilon^n \mathbf{x}^{(n)},$$

where the step is thought of as stopping at a time $t = \epsilon$ in the parameterisation of the natural pathway. This unknown ϵ may seem like a problem but it can be made to cancel. Define the correction at order n to be the n^{th} term of the Taylor series:

$$\mathbf{c}_n = \frac{1}{n!}\epsilon^n \mathbf{x}^{(n)}.$$

The step begins at $\mathbf{c}_0 = \mathbf{x}$ and the first order uncorrected step ends at $\mathbf{c}_0 + \mathbf{c}_1$, so has length $\mathbf{c}_1$. Now recall that for $n \geq 2$, the derivatives of $\mathbf{f} \circ \mathbf{x}$ are zero and use Faà di Bruno's formula as before:

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}\mathbf{f}(\mathbf{x}(t)) = \sum_{\pi \in \Pi_n} \mathbf{f}^{(|\pi|)}(\mathbf{x}(t)) \bigotimes_{p \in \pi} \mathbf{x}^{(|p|)}(t) = \mathbf{0}.$$

Multiplying both sides by ϵ^n gives

$$\sum_{\pi \in \Pi_n} \mathbf{f}^{(|\pi|)}(\mathbf{x}(t)) \bigotimes_{p \in \pi} \epsilon^{|p|}\mathbf{x}^{(|p|)}(t) = \mathbf{0},$$

using the fact that π is a partition of $\{1, 2, ..., n\}$, so the sum of sizes $|p|$ of all its elements is n. Noting that $\epsilon^n \mathbf{x}^{(n)} = n!\mathbf{c}_n$ and evaluating at $t = 0$ gives

$$\sum_{\pi \in \Pi_n} \mathbf{f}^{(|\pi|)} \bigotimes_{p \in \pi} |p|!\mathbf{c}_{|p|} = \mathbf{0}.$$

This formula is the basis for calculating corrections $\mathbf{c}_n$ for finite steps in the following sections.

4.1. The Meaning of ϵ

Observant readers might have noticed that $\mathbf{c}_1 = \epsilon\dot{\mathbf{x}}$ and in an earlier section, $\dot{\mathbf{x}} = -J^{-1}\mathbf{f}$, so taking a full Newton step would imply $\epsilon = 1$. This paper treats ϵ as a small value because when experiencing slow convergence from the 'narrow curving valleys' problem, the area of validity for the local linear model (the trust region) is much smaller than what is required to go all the way to the model minimum. This means the steps taken that succeed in reducing the function sum of squares would only be a fraction of the Newton step, for example a Levenberg–Marquardt step with λ chosen large enough to damp away the longest-range movement axes of the exact Newton scheme.

5. Finite Difference Schemes

The higher-order corrections $\mathbf{c}_n$ are expressible in terms of multiple directional derivatives of $\mathbf{f}$. For a numerical method, these derivatives must be calculated from function values, or at most, the Jacobian used by the algorithm. In this paper finite difference schemes are used, some of which have their 'stencils' of sampled points spread in multiple axes to give mixed derivatives.

5.1. Second Order

For $n = 2$, the general formula gives

$$\begin{aligned} \mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1 + \mathbf{f}^{(1)}2\mathbf{c}_2 &= \mathbf{0} \\ \Rightarrow \qquad \mathbf{c}_2 &= -\tfrac{1}{2}J^{-1}\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1. \end{aligned}$$

Taylor expansion of $\mathbf{f}$ in the direction $\mathbf{c}_1$ of the original uncorrected step gives

$$\begin{aligned} \mathbf{f}(\mathbf{x} + \mathbf{c}_1) &= \mathbf{f} + J\mathbf{c}_1 + \tfrac{1}{2}\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1 + O(\epsilon^3) \\ \Rightarrow \qquad \tfrac{1}{2}\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1 &= \mathbf{f}(\mathbf{x} + \mathbf{c}_1) - (\mathbf{f} + J\mathbf{c}_1) + O(\epsilon^3). \end{aligned}$$

In other words, the difference between $\mathbf{f}(\mathbf{x} + \mathbf{c}_1)$ and a linear estimate using the $\mathbf{f}$ and J already calculated at $\mathbf{x}(0)$, is to leading order a second derivative term similar to the one required for calculating $\mathbf{c}_2$. Thus,

$$\mathbf{c}_2 = -J^{-1}(\mathbf{f}(\mathbf{x} + \mathbf{c}_1) - (\mathbf{f} + J\mathbf{c}_1)) + O(\epsilon^3).$$

The evaluations required for this calculation are shown in Figure 1. In this case, only one other point besides the evaluations of $\mathbf{f}$ and J at $\mathbf{x}$ is needed.

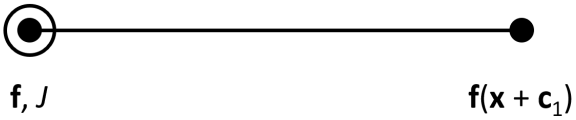

Figure 1: Finite difference stencil for calculating the second order correction $\mathbf{c}_2$. Points represent evaluations of the function $\mathbf{f}$ and rings represent evaluations of its Jacobian.

5.2. Third Order

For $n = 3$, the general formula gives

$$\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + 3\mathbf{f}^{(2)}\mathbf{c}_1 2\mathbf{c}_2 + \mathbf{f}^{(1)} 6\mathbf{c}_3 = \mathbf{0}.$$

$$\Rightarrow \qquad \mathbf{c}_3 = -\tfrac{1}{6}J^{-1}(\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + 6\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_2).$$

There are a few differences from the second order case:

- There is a third order derivative $\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1$, which will require an additional stencil point in the direction of $\mathbf{c}_1$.
- Errors will now have to be $O(\epsilon^4)$ as the main terms have size $O(\epsilon^3)$.
- There is a mixed derivative $\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_2$, requiring a two dimensional stencil pattern.

The mixed derivative requires knowledge of the direction $\mathbf{c}_2$, which must be evaluated first. The second order stencil for $\mathbf{c}_2$ had error $O(\epsilon^3)$ and now $O(\epsilon^4)$ is needed, so even the lower-order derivative $\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1$ will have to be evaluated using a third order stencil. Fortunately, this stencil is also needed for evaluating $\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1$, so all coefficients of a cubic approximation to $\mathbf{f}$ in this direction can be known.

5.2.1. Phase One: Calculating $\mathbf{c}_2$

The additional stencil point in the $\mathbf{c}_1$ direction is chosen to be $\mathbf{x} + \frac{1}{2}\mathbf{c}_1$ here, although other choices are possible. To third order,

$$\mathbf{f}(\mathbf{x} + \tfrac{1}{2}\mathbf{c}_1) = \mathbf{f} + \tfrac{1}{2}J\mathbf{c}_1 + \tfrac{1}{8}\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1 + \tfrac{1}{48}\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + O(\epsilon^4)$$

$$\mathbf{f}(\mathbf{x} + \mathbf{c}_1) = \mathbf{f} + J\mathbf{c}_1 + \tfrac{1}{2}\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1 + \tfrac{1}{6}\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + O(\epsilon^4).$$

Writing the nonlinear part of $\mathbf{f}$ as $\mathbf{f}_{nl}(\mathbf{x} + \mathbf{a}) = \mathbf{f}(\mathbf{x} + \mathbf{a}) - (\mathbf{f} + J\mathbf{a})$, the derivatives in the $\mathbf{c}_1$ direction can be expressed

$$\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1 = 16\mathbf{f}_{nl}(\mathbf{x} + \tfrac{1}{2}\mathbf{c}_1) - 2\mathbf{f}_{nl}(\mathbf{x} + \mathbf{c}_1) + O(\epsilon^4)$$

$$\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 = 12\mathbf{f}_{nl}(\mathbf{x} + \mathbf{c}_1) - 48\mathbf{f}_{nl}(\mathbf{x} + \tfrac{1}{2}\mathbf{c}_1) + O(\epsilon^4)$$

and $\mathbf{c}_2$ calculated from the formula $\mathbf{c}_2 = -\frac{1}{2}J^{-1}\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1$ with $O(\epsilon^4)$ error.

5.2.2. Phase Two: Calculating $\mathbf{c}_3$

This step requires the mixed derivative $\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_2$. Expressions of the form $\mathbf{f}^{(3)}\mathbf{uvw} = (\mathbf{u}\cdot\nabla)(\mathbf{v}\cdot\nabla)(\mathbf{w}\cdot\nabla)\mathbf{f}$ are iterated directional derivatives. Each directional derivative can be approximated to leading order as

$$\begin{aligned} (\mathbf{u}\cdot\nabla)\mathbf{f}(\mathbf{x}) &= \frac{\mathbf{f}(\mathbf{x}+\epsilon\mathbf{u})-\mathbf{f}(\mathbf{x})}{\epsilon} + O(\epsilon) \\ \Rightarrow \qquad (\epsilon\mathbf{u}\cdot\nabla)\mathbf{f}(\mathbf{x}) &= \mathbf{f}(\mathbf{x}+\epsilon\mathbf{u})-\mathbf{f}(\mathbf{x}) + O(\epsilon^2) \\ \Rightarrow \qquad (\epsilon^n\mathbf{u}\cdot\nabla)\mathbf{f}(\mathbf{x}) &= \mathbf{f}(\mathbf{x}+\epsilon^n\mathbf{u})-\mathbf{f}(\mathbf{x}) + O(\epsilon^{2n}). \end{aligned}$$

In the last formula above, $\epsilon^n\mathbf{u}$ represents an $O(\epsilon^n)$ sized term such as $\mathbf{c}_n$. Using this multiple times allows mixed derivatives to be expressed to leading order as combinations of function evaluations at different points (i.e. finite difference stencils). For example,

$$\begin{aligned} \mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_2 &= (\mathbf{c}_1\cdot\nabla)(\mathbf{c}_2\cdot\nabla)\mathbf{f} \\ &\simeq (\mathbf{c}_1\cdot\nabla)(\mathbf{f}(\mathbf{x}+\mathbf{c}_2)-\mathbf{f}(\mathbf{x})) \\ &\simeq \mathbf{f}(\mathbf{x}+\mathbf{c}_2+\mathbf{c}_1)-\mathbf{f}(\mathbf{x}+\mathbf{c}_1)-(\mathbf{f}(\mathbf{x}+\mathbf{c}_2)-\mathbf{f}(\mathbf{x})). \end{aligned}$$

Here, all terms have size $O(\epsilon^3)$ and all approximations are leading order accurate meaning the error is no worse than $O(\epsilon^4)$, as required.

In general, a d^{th} derivative of different directions would require 2^d evaluations. An n times repeated derivative in the same direction only requires $n+1$ as some of the evaluation points are coincident. A mixture like $\mathbf{f}^{(a+b+c)}\mathbf{u}^{\otimes a}\mathbf{v}^{\otimes b}\mathbf{w}^{\otimes c}$ would require evaluation at $(a+1)(b+1)(c+1)$ points.

Some efficiencies may be gained from coincident evaluation points and the fact that the full first derivative J is usually evaluated at $\mathbf{x}$ already. This was used in the previous 'second order' section, which only required one additional evaluation point for $\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1$ rather than three.

Now everything is in place to evaluate $\mathbf{c}_3 = -\frac{1}{6}J^{-1}(\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + 6\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_2)$. The full step will be corrected from $\mathbf{c}_1$ to $\mathbf{c}_1+\mathbf{c}_2+\mathbf{c}_3$. The evaluations required are shown in Figure 2.

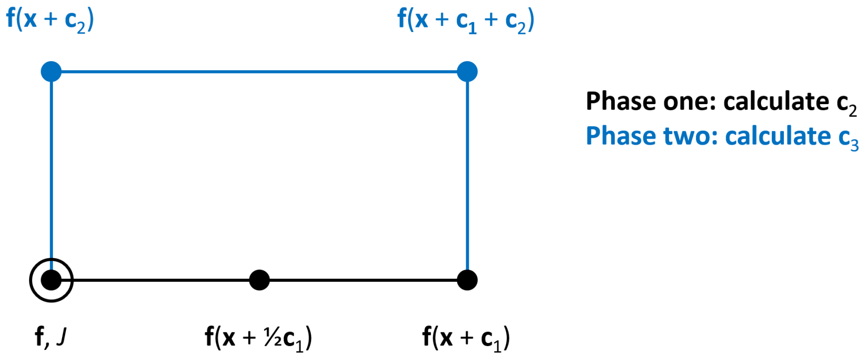

Figure 2: Finite difference stencil for calculating the third order correction $\mathbf{c}_3$ along with the second order correction $\mathbf{c}_2$ that is also required.

5.3. Fourth Order

For $n = 4$, the general formula gives

$$\mathbf{f}^{(4)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + 6\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1 2\mathbf{c}_2 + 4\mathbf{f}^{(2)}\mathbf{c}_1 6\mathbf{c}_3 + 3\mathbf{f}^{(2)}2\mathbf{c}_2 2\mathbf{c}_2 + \mathbf{f}^{(1)}24\mathbf{c}_4 = \mathbf{0}$$

$$\Rightarrow \qquad \mathbf{c}_4 = -\tfrac{1}{24}J^{-1}(\mathbf{f}^{(4)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + 12\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_2 + 24\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_3 + 12\mathbf{f}^{(2)}\mathbf{c}_2\mathbf{c}_2).$$

As expected, there are more higher-order and mixed derivatives. The double derivative $\mathbf{f}^{(2)}\mathbf{c}_2\mathbf{c}_2$ can take advantage of the Jacobian to eliminate a point from the stencil, just as previous unidirectional derivatives did. The direction $\mathbf{c}_3$ is now involved in the derivatives, so three evaluation phases are required. All errors have to be $O(\epsilon^5)$ including those of $\mathbf{c}_2$ and $\mathbf{c}_3$.

5.3.1. Phase One: Calculating $\mathbf{c}_2$

An additional stencil point $\mathbf{x} + \frac{3}{2}\mathbf{c}_1$ will be added to increase the order of accuracy in the $\mathbf{c}_1$ direction. Defining $\mathbf{f}_{nl}(\mathbf{x}+\mathbf{a}) = \mathbf{f}(\mathbf{x}+\mathbf{a}) - (\mathbf{f} + J\mathbf{a})$ as before,

$$\begin{aligned}
\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1 &\simeq 24\mathbf{f}_{nl}(\mathbf{x}+\tfrac{1}{2}\mathbf{c}_1) - 6\mathbf{f}_{nl}(\mathbf{x}+\mathbf{c}_1) + \tfrac{8}{9}\mathbf{f}(\mathbf{x}+\tfrac{3}{2}\mathbf{c}_1)\\
\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 &\simeq -120\mathbf{f}_{nl}(\mathbf{x}+\tfrac{1}{2}\mathbf{c}_1) + 48\mathbf{f}_{nl}(\mathbf{x}+\mathbf{c}_1) - 8\mathbf{f}(\mathbf{x}+\tfrac{3}{2}\mathbf{c}_1)\\
\mathbf{f}^{(4)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 &\simeq 192\mathbf{f}_{nl}(\mathbf{x}+\tfrac{1}{2}\mathbf{c}_1) - 96\mathbf{f}_{nl}(\mathbf{x}+\mathbf{c}_1) + \tfrac{64}{3}\mathbf{f}(\mathbf{x}+\tfrac{3}{2}\mathbf{c}_1),
\end{aligned}$$

all with errors $O(\epsilon^5)$. These formulae came from writing out the Taylor expansions and inverting the system of equations, which can also be done by inverting a matrix as the equations are linear in $\mathbf{f}$ and its derivatives. Calculate $\mathbf{c}_2 = -\frac{1}{2}J^{-1}\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_1$ using the first formula above.

5.3.2. Phase Two: Calculating $\mathbf{c}_3$

The mixed derivative $\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_2$ needs a grid with three points in the $\mathbf{c}_1$ direction and two in the $\mathbf{c}_2$ direction for a total of six. Many previous points can be re-used, with the only new point for fourth order in this phase being $\mathbf{x} + \frac{1}{2}\mathbf{c}_1 + \mathbf{c}_2$.

$$\begin{aligned}
\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_2 &\simeq (4\mathbf{f}(\mathbf{x}+\mathbf{c}_2) - 8\mathbf{f}(\mathbf{x}+\tfrac{1}{2}\mathbf{c}_1+\mathbf{c}_2) + 4\mathbf{f}(\mathbf{x}+\mathbf{c}_1+\mathbf{c}_2)) -\\
&\quad (4\mathbf{f} - 8\mathbf{f}(\mathbf{x}+\tfrac{1}{2}\mathbf{c}_1) + 4\mathbf{f}(\mathbf{x}+\mathbf{c}_1))\\
\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_2 &\simeq (-3\mathbf{f}(\mathbf{x}+\mathbf{c}_2) + 4\mathbf{f}(\mathbf{x}+\tfrac{1}{2}\mathbf{c}_1+\mathbf{c}_2) - \mathbf{f}(\mathbf{x}+\mathbf{c}_1+\mathbf{c}_2)) -\\
&\quad (-3\mathbf{f} + 4\mathbf{f}(\mathbf{x}+\tfrac{1}{2}\mathbf{c}_1) - \mathbf{f}(\mathbf{x}+\mathbf{c}_1))\\
\mathbf{f}^{(2)}\mathbf{c}_2\mathbf{c}_2 &\simeq 2\mathbf{f}_{nl}(\mathbf{x}+\mathbf{c}_2).
\end{aligned}$$

Again, all errors are $O(\epsilon^5)$ and $\mathbf{c}_3 = -\frac{1}{6}J^{-1}(\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + 6\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_2)$ can now be calculated to the required accuracy.

5.3.3. Phase Three: Calculating $\mathbf{c}_4$

This phase requires the single mixed derivative $\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_3$, which can be handled analogously to when $\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_2$ first appeared, by extending the stencil in the $\mathbf{c}_3$ direction with the two points $\mathbf{x} + \mathbf{c}_3$ and $\mathbf{x} + \mathbf{c}_1 + \mathbf{c}_3$.

$$\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_3 \simeq \mathbf{f}(\mathbf{x}+\mathbf{c}_1+\mathbf{c}_3) - \mathbf{f}(\mathbf{x}+\mathbf{c}_3) - (\mathbf{f}(\mathbf{x}+\mathbf{c}_1) - \mathbf{f}(\mathbf{x})).$$

Now all values are available to evaluate the fourth order correction $\mathbf{c}_4 = -\frac{1}{24}J^{-1}(\mathbf{f}^{(4)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1\mathbf{c}_1 + 12\mathbf{f}^{(3)}\mathbf{c}_1\mathbf{c}_1\mathbf{c}_2 + 24\mathbf{f}^{(2)}\mathbf{c}_1\mathbf{c}_3 + 12\mathbf{f}^{(2)}\mathbf{c}_2\mathbf{c}_2)$. The full step will be corrected from $\mathbf{c}_1$ to $\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \mathbf{c}_4$ and the evaluations required are shown in Figure 3.

It is clear that this process could be continued to even higher orders, although the stencils would require more and more points. Practically, automated selection of points and calculations of the stencil coefficients would also be required.

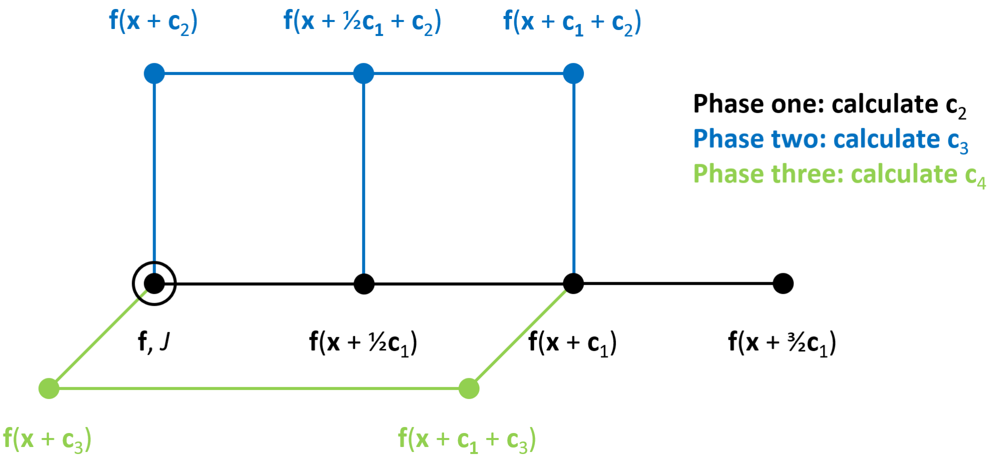

Figure 3: Finite difference stencil for calculating the second, third and fourth order corrections $\mathbf{c}_2 + \mathbf{c}_3 + \mathbf{c}_4$ at fourth order accuracy.

6. Numerical Test Problem

The higher-order algorithms were tested on a simple function to verify their performance. This function had to exhibit the 'narrow curving valleys' problem in its sum of squares, so a very anisotropic function $(x, y) \mapsto (x, Ky)$ for $K \gg 1$ was chosen and then some nonlinearity added in parameter space. The resulting function is

$$\mathbf{f}(x, y) = (x + y^2, K(y - x^2)).$$

Typical values of $K = 10^6$ were used and the iteration was started from the arbitrary point $(x, y) = (\pi, e)$, moving towards the minimum sum of squares at $\mathbf{f}(0, 0) = (0, 0)$.

Figure 4 shows the improved performance of the higher order corrected methods on the test problem with $K = 10^6$. The error norm in the plot is the value of $|\mathbf{f}(x, y)|$ after each iteration. There is a slow convergence region for $0.1 \leq |\mathbf{f}| \leq 10$ after which the algorithm converges rapidly. When the full Newton step using the linear model becomes a valid, convergence should be quadratic with the error norm roughly squaring on each iteration. This rapid convergence appears as the near-vertical descending lines on the graph for $|\mathbf{f}| < 0.1$.

6.1. Varying Valley Anisotropy

Varying K should show the relative performance of the different order algorithms as the valley gets narrower, while the curvature is kept constant. Table 1 shows the number of iterations required to converge as K is varied between 1 and 10^{12}.

A simplified model is that the valley has width $1/K$ while the n^{th} order method has error term $O(\epsilon^{n+1})$, so this error would push the proposed step out of the valley when $O(\epsilon^{n+1}) = 1/K$ or $\epsilon = O(K^{-\frac{1}{n+1}})$. This would give $O(1/\epsilon) = O(K^{\frac{1}{n+1}})$ steps to convergence.

Plotting the data on a log-log plot in Figure 5 reveals straight lines in parts of the data that suggest a power law relationship. For $K \geq 10^9$ there is an additional increase in convergence time, which may be from the limits of double precision used for the calculation. Taking the gradient through the last three available points with $K \leq 10^8$ gives power law exponents of $0.660, 0.392, 0.265, 0.203$ for the first through fourth order methods. This is somewhat similar to the simplified model's $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}$ although lower-order methods seem slower, particularly the first order.

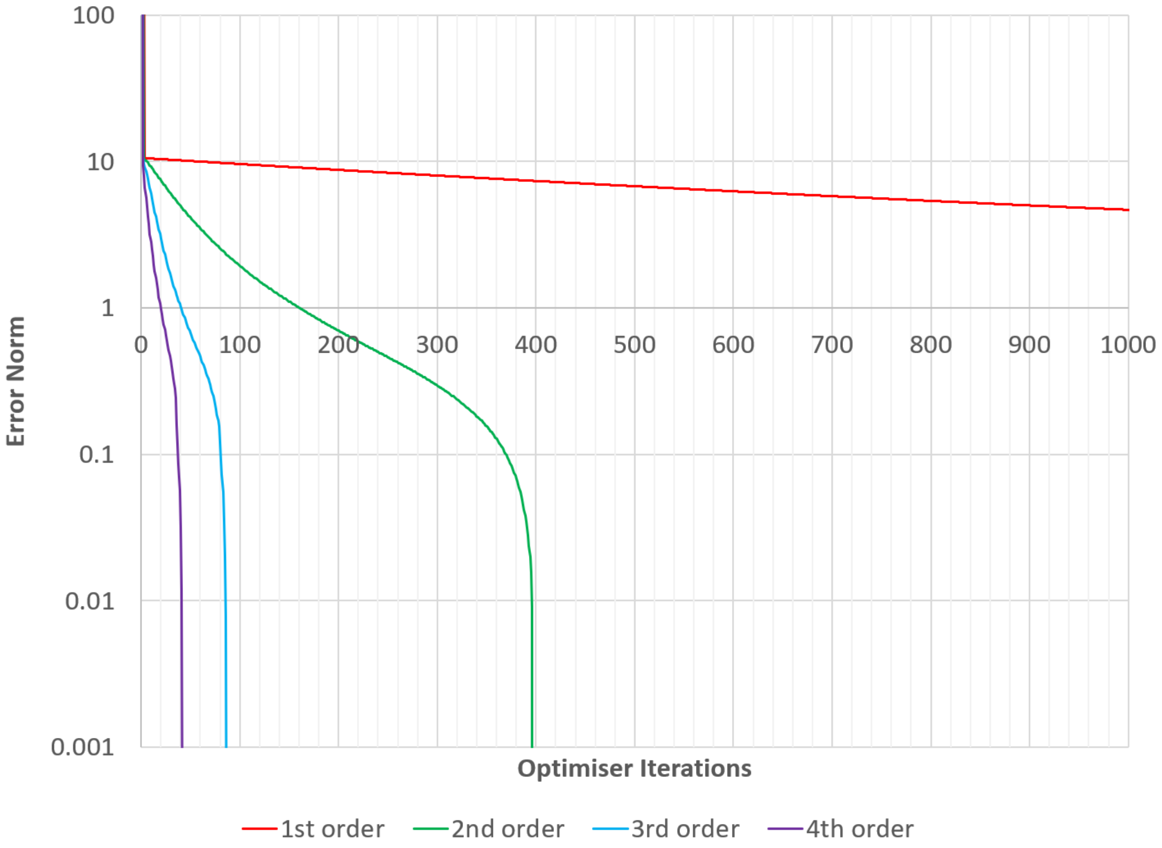

Figure 4: Performance of higher-order corrected Levenburg–Marquardt methods on a test problem.

Table 1: Test problem convergence times for different order methods as the anisotropy factor K is varied.

Anisotropy K	**1st order**	**2nd order**	**3rd order**	**4th order**
1	8	6	5	5
10	15	8	6	5
100	47	16	9	8
1000	196	30	18	11
10000	880	68	24	18
100000	4041	162	50	27
10^6	18733	397	88	43
10^7	>20000	971	166	70
10^8	>20000	2432	312	110
10^9	>20000	5828	631	243
10^{10}	>20000	>20000	2876	968
10^{11}	>20000	>20000	10886	2706
10^{12}	>20000	>20000	>20000	9159

6.2. Integration with Levenburg–Marquardt Method

These numerical experiments were performed with a Levenburg–Marquardt method enhanced with the higher-order corrections. Step length ϵ is controlled by choosing the damping parameter $\lambda \geq 0$ in the pseudo-inverse $[J^{-1}]_{LM(\lambda)} = (J^T J + \lambda I)^{-1} J^T$. The note [10] shows that $\lambda = 0$ gives the full Gauss–Newton step, while $\lambda \to \infty$ produces infinitesimal steepest gradient steps, with the values of λ in between producing optimal reductions in the linear model for a

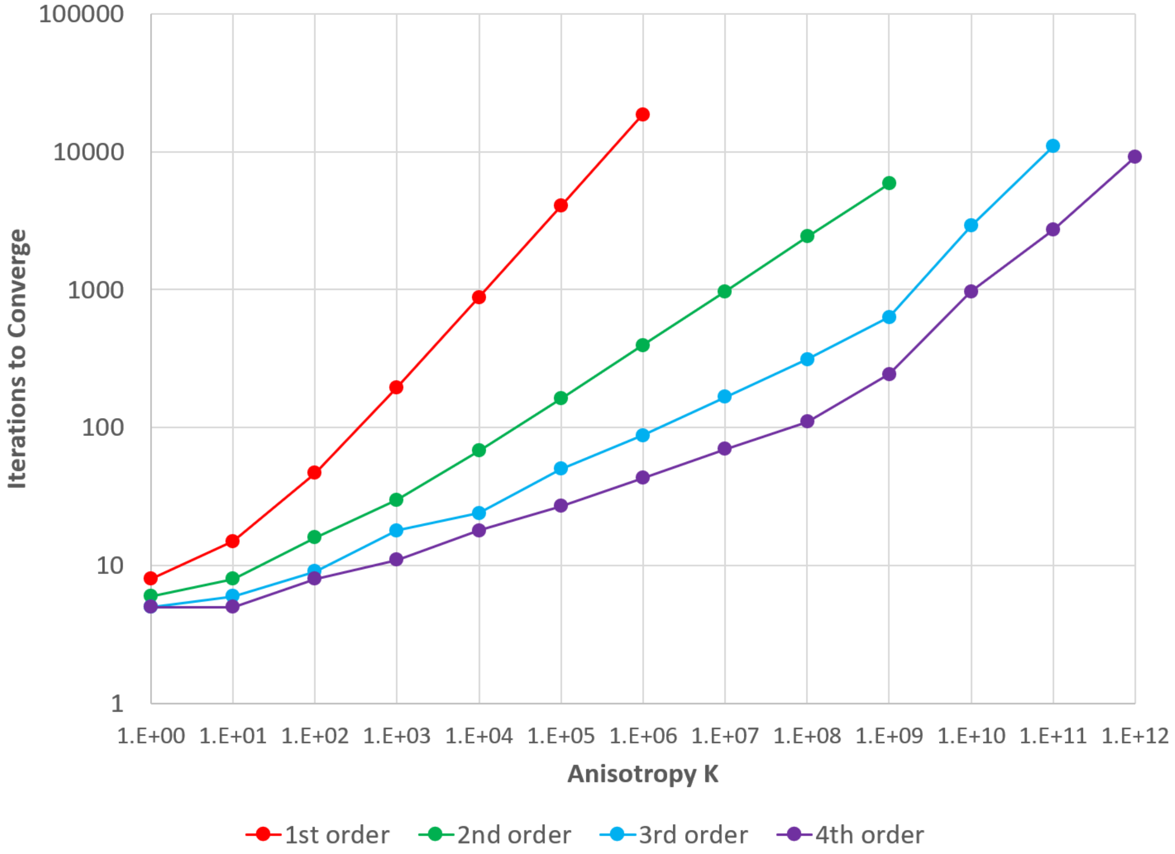

Figure 5: Test problem convergence times for different order methods as the anisotropy factor K is varied.

given step size.

A good choice of λ should be chosen at each step. In this study the values

$$\lambda_n = \lambda_{\text{old}} 10000^{(n/10)^3} \qquad \text{for} \quad -10 \leq n \leq 10,$$

where λ_{old} is the value from the previous step, are run in parallel and the one that produces the best reduction in $|\mathbf{f}|$ chosen. The initial step uses $\lambda_{\text{old}} = 1$.

The fact these steps are run in parallel means that for the higher order methods, many initial Levenburg–Marquardt steps $\mathbf{c}_1(\lambda_n)$ are calculated, each of which has higher order corrections $\mathbf{c}_i(\lambda_n)$. The function $\mathbf{f}$ is evaluated at all the corrected step points $\mathbf{x}_{\text{out}}(\lambda_n) = \sum_{i=0}^{order} \mathbf{c}_i(\lambda_n)$ and the one with lowest $|\mathbf{f}|$ and its corresponding λ_n is chosen.

The higher order methods enable longer steps and thus smaller values of λ to be used. The scheme above is somewhat wasteful by trying 21 values of λ each step, but on modern computers these can be parallelised, unlike the slow progress along the narrow optimisation valley, which is a serial calculation.

7. Performance on a Practical Problem

These algorithms have also been used on a more complex optimisation problem (which motivated their development). This problem has 180 parameters and 300 output variables and features levels of successively more difficult narrow curved valleys in its optimisation space.

The full details of this problem are not the point of this paper but a brief summary will be given here. An initial distribution of 100 Ca$^+$ ions is accelerated through a potential of 1 kV and given a $\pm 2\%$ energy chirp. It is transported through a curved electrostatic channel, where the electric field is produced by 15 rings of 12 configurable electrodes. Each electrode is modelled as a point charge and these 180 charges are the optimisation variables. The output vector whose norm should be minimised contains the (x, y, z) position coordinates of the 100 ions on exiting the channel (so 300 entries in all), with the bunch centroid subtracted. The ions do not interact in this model and their trajectories are calculated by the 4$^{\text{th}}$ order Runge–Kutta method [11, 12] with a timestep $\delta t = 10^{-7}$ seconds.

This problem is interesting because focussing the ions to a point in the linear dynamics approximation can be done with standard optics, but optical aberrations at higher order will remain, for example spherical aberration from large angle effects. These higher order aberrations can also be corrected by careful choice of the electrode voltages, although this gets more difficult the smaller the focal point becomes and the more aberrations have to be cancelled simultaneously.

The figure of merit is the RMS focal size of the ion bunch, which is $\frac{1}{\sqrt{100}}$ of the norm $|\mathbf{f}|$. Figure 6 shows how this is reduced by the Levenburg–Marquardt optimisation method with various levels of higher order correction.

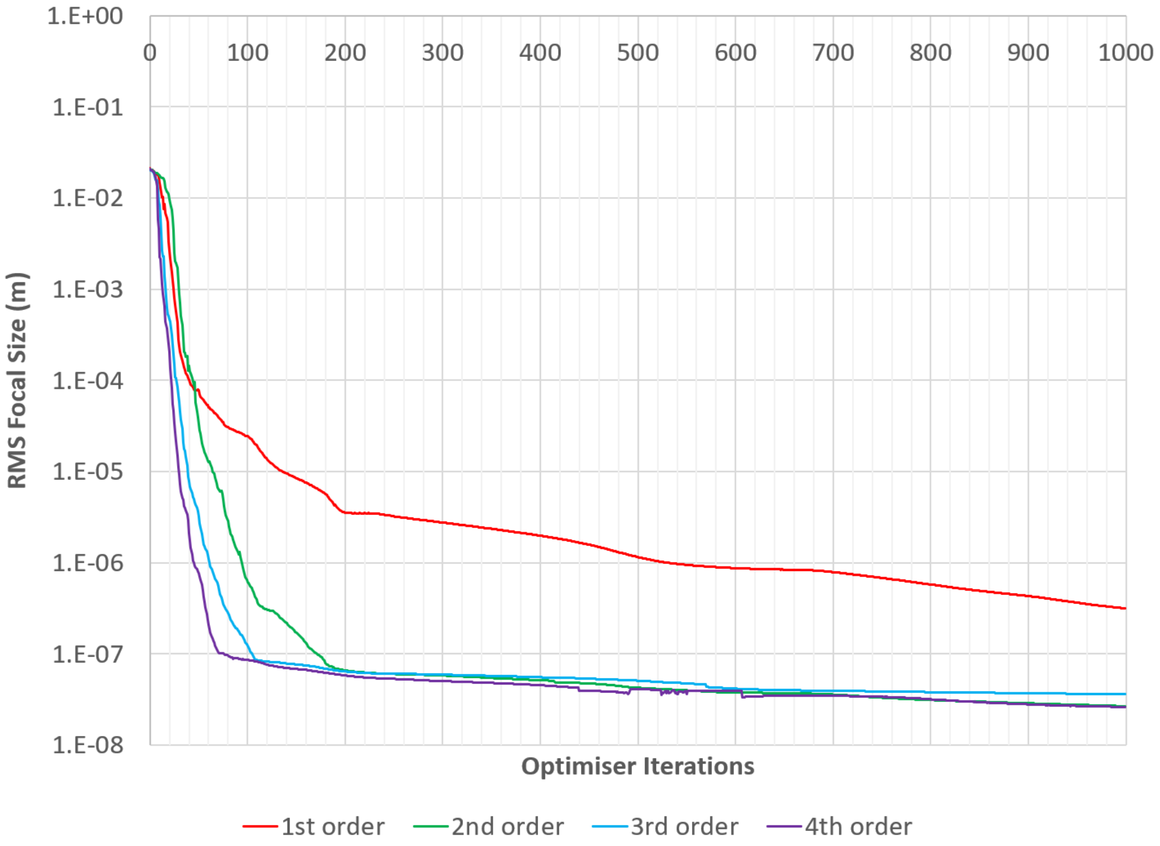

Figure 6: Performance of higher-order corrected Levenburg–Marquardt methods on a physics problem with 180 parameters and 300 output variables.

Figure 6 shows that higher order methods signficantly accelerate the optimisation progress while the ion bunch focal size is greater than 10^{-7} metres, with comparative performance analogous to that on the test problem in Figure 4. Once the focal size reduces below 10^{-7} metres, progress in the optimisation slows down greatly for all orders of method reaching this level. The reason for this slow-down is yet to be determined. Lack of numerical precision would produce a similar 'noise floor', although in this study care was taken to calculate the Jacobian J with automatic differentiation rather than the noisier finite difference schemes.

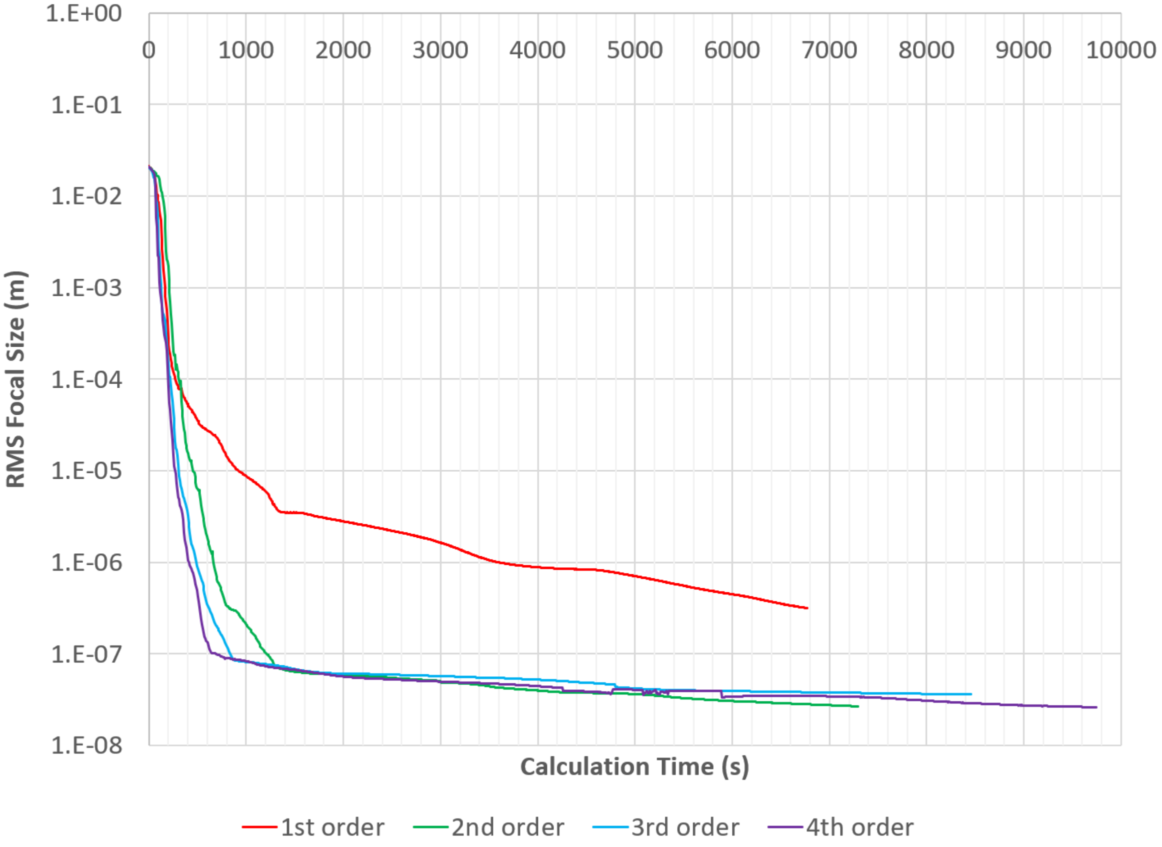

Figure 7: Calculation time vs. performance for higher-order methods on a physics problem with 180 parameters.

One potential concern with these higher-order methods is that the additional evaluations of $\mathbf{f}$ in the stencil will cost too much time to make the method worth using. To measure this effect, the optimised focal size is plotted as a function of calculation time in Figure 7. The differences in total execution time can be seen at the right-hand end of each line, which is after 1000 iterations. The fourth order method takes 44% longer per step but still manages to pull ahead of the other methods in real time before the 'floor' is reached. Table 2 gives the amount of time for each method to reach an RMS focal size of under 1 micron.

Table 2: Real time taken to reach a focal size of $< 10^{-6}$ m in the ion focussing problem with 180 parameters.

Method Order	Iterations	Calculation Time (s)
1	532	3604.863
2	94	665.812
3	62	495.010
4	46	418.301

8. Conclusion

Methods such as Levenburg–Marquardt are not only for curve fitting, but also powerful optimisers where the function to be minimised is a sum of squares. Like many optimisers, they can get stuck for long periods of time in 'curved narrow valleys'. This paper derives higher-order corrections beyond the already known second order [3, 4] that further accelerate the optimiser performance in these situations. A general formula for deriving the n^{th} order correction is given, with suggestions on how to build finite difference stencils to evaluate it.

These successful methods have been derived using the concept of a 'natural pathway' for the optimisation, which is an ordinary differential equation (ODE) that is meant to follow the valleys. The form of this ODE is chosen somewhat arbitrarily here but it appears to work well, perhaps because it is a continuous version of the optimiser's path in the limit where step size $\epsilon \to 0$. Using a higher-order step method on this ODE thus makes the optimiser behave 'as if' it had done a large number of very small steps.

This link with ODEs also suggests potential future work in applying well-known higher-order methods for ODEs such as Runge–Kutta [11, 12] or Bulirsch–Stoer [13] to difficult optimisation problems, as well as methods for stiff ODEs. In this paper the RK4 method was not preferred because it would have required four evaluations of the Jacobian, which is still much more expensive than the eight additional function points in the stencil of the fourth order method.

References

[1] "A Method for the Solution of Certain Non-Linear Problems in Least Squares", Kenneth Levenberg, Quarterly of Applied Mathematics **2** (2), pp.164–168 (1944) `doi:10.1090/qam/10666`

[2] "An Algorithm for Least-Squares Estimation of Nonlinear Parameters", Donald Marquardt, SIAM Journal on Applied Mathematics **11** (2), pp.431–441 (1963) `doi:10.1137/0111030`

[3] "Why are Nonlinear Fits to Data so Challenging?", Mark K. Transtrum, Benjamin B. Machta and James P. Sethna, Phys. Rev. Lett. **104**, 060201 (2010) `doi:10.1103/PhysRevLett.104.060201`

[4] "Geometry of nonlinear least squares with applications to sloppy models and optimization", Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna, Phys. Rev. E **83**, 036701 (2011) `doi:10.1103/PhysRevE.83.036701`

[5] "Improvements to the Levenberg-Marquardt algorithm for nonlinear least-squares minimization", Mark K. Transtruma and James P. Sethna, arXiv:1201.5885v1 [physics.data-an] (2012).

[6] "Geodesic acceleration and the small-curvature approximation for nonlinear least squares", Mark K. Transtruma and James P. Sethna, arXiv:1207.4999v1 [math.OC] (2012).

[7] "Du calcul des derivations" [On the calculus of derivatives] (in French), L.F.A. Arbogast, Strasbourg: Levrault, pp. xxiii+404 (1800). Entirely freely available from Google books `https://books.google.com/books?id=YoPq8uCy5Y8C`

[8] "Sullo sviluppo delle funzioni" [On the development of the functions] (in Italian), F. Faà di Bruno, Annali di Scienze Matematiche e Fisiche, **6**: 479–480, LCCN 06036680 (1855). Entirely freely available from Google books `https://books.google.com/books?id=ddE3AAAAMAAJ&pg=PA479`

[9] “Note sur une nouvelle formule de calcul differentiel” [On a new formula of differential calculus] (in French), F. Faà di Bruno, The Quarterly Journal of Pure and Applied Mathematics, **1**: 359–360 (1857). Entirely freely available from Google books `https://books.google.com/books?id=7BELAAAAYAAJ&pg=PA359`

[10] “Bounded Approximate Solutions of Linear Systems using SVD”, Stephen Brooks, tech note BNL-223624-2022-TECH, available from `https://technotes.bnl.gov/` or `https://stephenbrooks.org/ap/report/2015-3/svdboundedsolve.pdf` (2015).

[11] Runge, C., *Math. Ann.* **46**, p.167 (1895); and Kutta, M.W.Z., *für Math. u. Phys.* **46**, p.435 (1901).

[12] *Numerical Recipes in C*, W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vettering, chapter 15.1: Integration of Ordinary Differential Equations - Runge–Kutta Method.

[13] *Introduction to Numerical Analysis*, J. Stoer and R. Bulirsch, section 7.2.14, New York: Springer-Verlag (1980).