Accelerated Gradient Methods with Memory

R. Drummond and S. R. Duncan

Abstract—A set of accelerated first order algorithms with memory are proposed for minimising strongly convex functions. The algorithms are differentiated by their use of the iterate history for the gradient step. The increased convergence rate of the proposed algorithms comes at the cost of robustness, a problem that is resolved by a switching controller based upon adaptive restarting. Several numerical examples highlight the benefits of the proposed approach over the fast gradient method. For example, it is shown that these gradient based methods can minimise the Rosenbrock banana function to $7.58 \times 10^{-12}$ in 43 iterations from an initial condition of $(-1,1)$.

Index Terms—Optimisation algorithms, fast gradient method, absolute stability.

INTRODUCTION

This paper considers the minimisation of continuously differentiable functions $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ belonging to the class $\mathcal{S}_{\mu,L}^n$ of strongly convex and Lipschitz bounded functions, satisfying

\begin{align}
\langle f'(x) - f'(y), x - y \rangle &\geq \mu \|x - y\|^2, \\
\|f'(x) - f'(y)\| &\leq L \|x - y\|,
\end{align}

with $0 < \mu < L$. For this purpose, black-box gradient methods where the algorithms are only given the parameters $\mu$ and $L$ are developed. The main results of the paper are a set of gradient-based algorithms ($\Sigma_N$ in (\ref{eq:algo})) that use the iterate memory to accelerate the convergence rate towards the minimiser. The algorithms are parametrised for quadratic functions such that the error of the mode corresponding to the dominant eigenvalue of the Hessian $w_k^{(i)}$ converges to zero at the rate

$$\|w_k^{(i)}\|_2 = \left(1 - \frac{\mu}{L}\right)^k \|w_0^{(i)}\|_2$$

for generic $N \in \mathbb{N}$ as described in Section \ref{sec:algo}. This rate is faster than that of the fast gradient method for this mode when $N \geq 3$.

The paper is structured as follows. Section \ref{sec:algo} briefly describes the classical first-order methods of gradient descent and the fast gradient method. The proposed set of algorithms are introduced in Section \ref{sec:algo_with_memory} where their parameterisation and robustness for quadratic problems is also discussed. Section \ref{sec:algo_with_memory} generalises the analysis of these methods to strongly convex problems using absolute stability theory. A switching based control scheme, based upon adaptive restarting, is introduced in Section \ref{sec:switching_control} to robustify these algorithms and recover their acceleration. The performance of these robustified algorithms is examined in Section \ref{sec:experiments} via numerical examples.

NOTATION

The index notation $x_k^{(i)}$ indicates the $i$th element of the vector $x_k$ evaluated at time step $k$. Bracketed upper indices are used to relate to elements. $\mathcal{Z}$-transforms of signals will be denoted in capitals when clear from the context.

The fast gradient method is referred to as $\mathcal{FG}$ and gradient descent as $\mathcal{GD}$. In the latter stages of the paper, these algorithms will also be referred to as $\Sigma_1 = \mathcal{GD}$ and $\Sigma_2 = \mathcal{FG}$ so as to be consistent with the notation of the proposed algorithm set $\Sigma_N$. The reciprocal of the condition number of a function is denoted $\kappa = \mu/L$. The identity matrix of dimension $n$ is denoted $I_n$.

I. FIRST ORDER METHODS: GRADIENT DESCENT AND THE FAST GRADIENT METHOD

The use of first-order methods for minimising strongly convex functions $f(x) \in \mathcal{S}_{\mu,L}^n$ has recently seen revived interest due to their relative computational simplicity, making them ideal for problems with large numbers of decision variables. A typical first order method is initialised with an estimate $x_0 \in \mathbb{R}^n$ of the unique minimiser of the function $x^* \in \mathbb{R}^n$ and a set of parameters $\theta \in \mathbb{R}^N$ determined from the strong convexity and Lipschitz constants $\mu$ and $L$. The estimate $x_0$ is then updated iteratively by stepping in the direction of the gradient of the function, generating a sequence $x_k \in \mathbb{R}^n$.

The classical first-order method is the gradient descent ($\mathcal{GD}$) algorithm

$$x_{k+1} = x_k - \frac{1}{L} \nabla f(x_k)$$

which is posed here with a step-length of $1/L$. For functions in $\mathcal{S}_{\mu,L}^n$, this algorithm is guaranteed to converge monotonically towards the optimal value $x^*$ but only at a rate $\propto 1 - \mu/L$, which is too slow for many applications.

A major development in first-order methods was Nesterov’s fast gradient method ($\mathcal{FG}$) (or accelerated gradient method) of \cite[Chapter 2]{Nesterov}. Three versions of this algorithm were proposed in \cite[Chapter 2]{Nesterov} and in this paper, the simplest of these will be considered as it recovers the same convergence rate for functions in $\mathcal{S}_{\mu,L}^n$. This method is described by the iterate sequence

\begin{align}
x_{k+1} &= y_k - \frac{1}{L} \nabla f(y_k) \\
y_k &= (1 + \beta)x_k - \beta x_{k-1}
\end{align}

R. Drummond and S. R. Duncan are with the Department of Engineering Science, University of Oxford, 17 Parks Road, OX1 3PJ, Oxford, United Kingdom, Email: {ross.drummond,stephen.duncan}@eng.ox.ac.uk.
where a step length of $1/L$ has again been used and with the tuning parameter $\beta$ being determined from the reciprocal of the condition number $\kappa = \mu/L$ according to

$$\beta = \frac{1 - \sqrt{\kappa}}{1 + \sqrt{\kappa}} \quad (5)$$

At each iteration $k$, the $FG$ algorithm uses the stored iterate values $x_k$ and $x_{k-1}$ to generate a new point $y_k \in \mathbb{R}^n$ to take the gradient step from. The striking feature of this algorithm is that this rather simple augmentation of gradient descent can result in a dramatic speed up in the convergence rate to $\alpha 1 - (\mu/L)^{1/2}$ such that

$$\|x_k - x^*\|^2_2 \leq (1 - (\mu/L)^{1/2})\|x_0 - x^*\|^2_2. \quad (6)$$

Crucially, since the algorithm only uses addition and multiplication, this speed up is achieved without significant sacrifices on computational efficiency.

Even though the $FG$ algorithm was developed more than twenty years ago, the extent of the relative simple change of stepping from the point $y_k$ instead of $x_k$ is still being interpreted from a dynamical systems perspective. Using previous iterate information $x_{k-1}$ is said to introduce “momentum” or “inertia” into the algorithm [8], a term which has been interpreted in terms of both continuous [23] and discrete [19] time second order dynamical systems. It was from this perspective that the symplectic integrator schemes of [3, 29] were developed.

Another interpretation is obtained by considering the algorithms as the feedback interconnection of a linear system with a nonlinear function $u(y_k) = \nabla f(y[k])$ that is both static and sector bounded. Such an approach allows the tools of absolute stability theory [13, Chapter 6] to then be applied. This connection between algorithm design and absolute stability was pioneered in [15] which used the language of integral quadratic constraints (IQC) to obtain convergence rate bounds that only require solving low-dimensional semi-definite programs (SDP). Lyapunov functions for $FG$ were proposed in [30, 10, 25], with a similar structure to the classical Tsypkin functions of [26] and [8, 23, 5]. Exploiting this feedback interpretation, [27] proposed a parametrisation of $FG$ with the fastest known convergence rate, with this parametrisation obtained from considering the closed-loop behaviour. Other developments of the fast gradient method include using secant information [1], sum-of-squares programming to develop developments of the fast gradient method include using a similar approach. This section is concerned with algorithms where $N \geq 3$. The notation $\Sigma_N$ denotes an algorithm with memory $N$, with $\Sigma_1 = GD$ and $\Sigma_2 = FG$. The results of this section build upon the ideas presented in [6] for minimising the quadratic cost functions of model predictive control.

The algorithms $\Sigma_N$ [7] have a similar structure to $FG$, as they perform a gradient step from a point $y_k$ generated from $N$ past iterates. The difference between them comes from the increased number of past iterates used in $\Sigma_N$. It is not required that the algorithm parameters $\theta \in \Theta$ be positive, so the generated point $y_k$ may not lie within the convex hull of the previous iterates. This fact is used in Section V.C for the minimisation of non-convex functions.

A. Setting the parameters $\theta$

The $\Sigma_N$ algorithm parameters $\theta$ still need to be defined. These parameters are set by considering the minimisation of quadratic functions $f_Q(x) : \mathbb{R}^n \rightarrow \mathbb{R}$

$$f_Q(x) = \frac{1}{2} x^T H x + h^T x \quad (8)$$

belonging to the class $f_Q \in Q_{\mu,L}^n \subset S_{\mu,L}^n$ where the eigenvalues $\lambda_i$ of the Hessian $H$ are restricted to be positive and lie within the range $\lambda_i \in [\mu, L] \forall i = 1, \ldots, n$.

Considering the iterate update of $\Sigma_N$ in (7a) with $f() = f_Q()$, then by subtracting the minimiser $x^* = 0$ from both sides and adding $\frac{1}{L} \nabla f(x^*) = 0$ to the RHS generates the recursive error sequence

$$e_{k+1} = \sum_{j=0}^{N-1} \theta^{(j)} \left( 1 - \frac{1}{L} H \right) e_{k-j}. \quad (9)$$

of the fast gradient method, although these algorithms are not robust in the sense discussed in Section II.B. To recover performance, an adaptive restarting scheme is proposed and numerical examples showcase the benefits of the approach.

II. Main Results: Proposed Algorithm set

For a given $N \in \mathbb{N}$, the minimisation of functions $f(x) \in \mathcal{S}_{\mu,L}^n$ is considered using the $\Sigma_N$ algorithms

$$x_{k+1} = y_k - \frac{1}{L} \nabla f(y_k), \quad (7a)$$

$$y_k = \sum_{j=0}^{N-1} \theta^{(j)} x_{k-j}, \quad (7b)$$

whose parameters $\theta = [\theta_0, \ldots, \theta_{N-1}] \in \mathbb{R}^N$ satisfy the affine constraint

$$\theta \in \Theta := \left\{ \theta \in \mathbb{R}^N : \sum_{j=0}^{N-1} \theta^{(j)} = 1 \right\}. \quad (7c)$$

The constraint (7c) on the sum of the parameters allows the error $e_k = x_k - x^*$ to be expressed recursively. The parameter $N$ defines the “memory” of the algorithm, with $N = 1$ returning gradient descent $GD$ and $N = 2$ giving the structure of the fast gradient method $FG$. This section is concerned with algorithms where $N \geq 3$. The notation $\Sigma_N$ denotes an algorithm with memory $N$, with $\Sigma_1 = GD$ and $\Sigma_2 = FG$. The results of this section build upon the ideas presented in [6] for minimising the quadratic cost functions of model predictive control.

The algorithms $\Sigma_N$ [7] have a similar structure to $FG$, as they perform a gradient step from a point $y_k$ generated from $N$ past iterates. The difference between them comes from the increased number of past iterates used in $\Sigma_N$. It is not required that the algorithm parameters $\theta \in \Theta$ be positive, so the generated point $y_k$ may not lie within the convex hull of the previous iterates. This fact is used in Section V.C for the minimisation of non-convex functions.
Introducing the function \( m(s) : [\kappa,1] \to [0,1-\kappa] \),
\[
m(s) = 1 - s,
\]
then this error system (9) admits an eigen-decomposition
\[
w^{(i)}_{k+1} = \sum_{j=0}^{N-1} \theta^{(j)} m(\lambda_i/L) w^{(j)}_{k-j}, \quad i = 1, \ldots, n,
\]
with \( \lambda_i \) being the \( i \)th eigenvalue of the Hessian \( H \).

The parameters \( \theta \) are set according to the mode associated to the dominant eigenvalue of the Hessian, namely \( \lambda_{i_0} = \mu \). This mode evolves according to
\[
w^{(i_0)}_{k+1} = \sum_{j=0}^{N-1} \theta^{(j)} m(\mu) w^{(j)}_{k-j},
\]
and has characteristic equation
\[
p_N(r; m(\mu)) = r^N - \sum_{j=0}^{N-1} \theta^{(j)} m(\mu) r^{N-1-j} = 0.
\]

Denoting the roots of this polynomial as \( r_1, r_2, \ldots, r_N \), then the evolution of the state of the above linear system can be written
\[
w^{i_0}_k \propto c_1 r_1^k + c_2 r_2^k + \cdots + c_N r_N^k,
\]
with the roots \( r_j \) determining the rate of convergence. The problem is then to choose the parameters \( \theta \in \mathbb{R}^N \) (subject to the constraint (7c) that minimises the moduli of the roots \( r_j \) for \( j = 1, \ldots, N \).

Defining the root radius of a polynomial \( p_N(r) \)
\[
\rho(p_N) = \max \{|r| : p_N(r; m(\mu)) = 0, r \in \mathbb{C} \},
\]
with the polynomials \( p_N(r) \) contained within the set
\[
P = \left\{ r^N - \sum_{j=0}^{N-1} \theta^{(j)} m(\mu) r^{N-1-j} : \sum_{j=0}^{N-1} \theta^{(j)} = 1, \theta^{(j)} \in \mathbb{R} \right\},
\]
then this problem can be cast as that of finding \( \theta \in \Theta \) that transforms \( p_N \) into the optimal polynomial \( p^*_N \) minimising
\[
\rho^* := \rho(p^*_N) = \inf_{p_N \in P} \rho(p_N).
\]

In general, the problem of globally minimising the root radius of a polynomial is known to be both non-convex and not Lipschitz. However, in [4], it was shown that there exists an analytic solution to this problem if the coefficients of the polynomials \( p_N(r; m(\mu)) \) are subject to a single affine constraint, which is the class of polynomials considered here with the affine constraint given by (7c).

Theorem 6 from [4] states that if the parameters \( \theta \) satisfy the affine constraint (7c), and moreover are allowed to be complex \( \theta \in \mathbb{C}^N \), then the solution \( p^*_N(r) \) to the root radius optimisation (17) is the polynomial with a common real root \( \gamma \in \mathbb{R} \)
\[
p^*_N(r) = (r - \gamma)^N.
\]

Similarly, if the parameters satisfy the affine constraint (7c) but are restricted to be real \( \theta \in \mathbb{R}^N \), then Theorem 1 of [4] states that the optimal polynomial is instead
\[
p_N(r) = (r + \gamma)^{N-j}(r - \gamma)^j,
\]
for some integer \( j \) with \( 0 \leq j \leq N \).

These theorems form the basis for the parameter choice adopted here; choosing \( \theta \) such that \( p_N(r; m(\mu)) \) has a common root \( \gamma \). Even though this choice may not be optimal according to Theorem 1 of [4], it gives an analytic solution to the problem which is useful for determining the algorithm speed-up.

Proposition 1 (Choosing \( \theta \)): If such a choice exists, then the parameters \( \theta \in \mathbb{R}^N \) should be chosen such that the polynomial \( p_N(r; m(\mu)) \) has a common root at \( \gamma \in \mathbb{R} \). This common root is given by
\[
\gamma = 1 - \left( \frac{\mu}{L} \right)^{\frac{1}{j}}.
\]

Proof.
To compute the common root \( \gamma \) in (20), the parameters \( \theta \) of the polynomials \( p_N(r; m(\mu)) \) are related to its roots \( r_1, r_2, \ldots, r_N \) using Viète’s formulas [23]. These formulas are obtained from the elementary symmetric polynomials and state that for any polynomial such as \( p_N(r; m(\mu)) \) defined by real or complex coefficients \( \theta \), then the following equations are satisfied
\[
r_1 + r_2 + r_3 + \cdots + r_N = \theta_{N-1} m(\mu),
\]
\[
r_1 r_2 + r_1 r_3 + \cdots + r_{N-1} r_N = \theta_{N-2} m(\mu),
\]
\[
r_1 r_2 r_3 + \cdots + r_1 r_2 r_N + \cdots + r_{N-2} r_{N-1} r_N = \theta_{N-3} m(\mu),
\]
\[
\vdots
\]
\[
r_1 r_2 \cdots r_N = (-1)^{N+1} \theta_0 m(\mu).
\]

If \( \theta \) is chosen according to Proposition 1, then the polynomial \( p_N(r; m(\mu)) \) has a common root \( \gamma \), as in \( r_1 = r_2 = \cdots = \gamma \), and the Viète’s formulas collapse to
\[
N \gamma = \theta_{N-1} m(\mu),
\]
\[
\left( \begin{array}{c} N \end{array} \right) \gamma^2 = -\theta_{N-2} m(\mu),
\]
\[
\left( \begin{array}{c} N \end{array} \right) \gamma^3 = \theta_{N-3} m(\mu),
\]
\[
\vdots
\]
\[
\gamma^N = (-1)^{N+1} \theta_0 m(\mu).
\]

The constraint \( \sum_{j=0}^{N-1} \theta^{(j)} = 1 \) means that after summing these formulas, the common root satisfies the polynomial
\[
\gamma^N + \sum_{j=1}^{N-1} (-1)^{j+1} \left( \begin{array}{c} N \end{array} \right) \gamma^j = (-1)^{N+1} m(\mu).
\]

Alternatively, this polynomial can be written
\[
(\gamma - 1)^N = (-1)^N k,
\]

giving (20).

Note the similarity between this equal root condition and the critical damping analysis of the fast gradient method in [19]. In fact, with this condition, the fast gradient method is recovered with \( N = 2 \).
To this end, the absolute stability approach of [13] is adopted. In this setting, the gradient of the cost function is regarded as a nonlinear function $u(x) : \mathbb{R}^n \to \mathbb{R}^n$ that is static and sector bounded, satisfying the quadratic inequality
\[
\begin{bmatrix}
y_k - y^* \\ u_k - u^*
\end{bmatrix} \begin{bmatrix}
-2mL I_n & (L + m) I_n \\
(L + m) I_n & -2I_n
\end{bmatrix} \begin{bmatrix}
y_k - y^* \\ u_k - u^*
\end{bmatrix} \geq 0.
\]
If $\nabla f(x) \in S^m_{\mu,L}$, then $u(x) = \nabla f(x)$ satisfies these criteria [13]. One can apply a loop transformation [13] on the nonlinearity so as to normalise the sector from $[\mu, L]$ to $[0, 1]$ by writing the gradient as
\[
\nabla f(x) = (L - \mu) \left( \frac{\nabla f(x) - \mu x}{L - \mu} + \frac{\mu x}{L - \mu} \right) 
\] or
\[
\nabla f(x) = (L - \mu) \nabla f_{[0,1]}(x) + \mu x
\]
with the transformed nonlinearity $u_{[0,1]}(x) = \nabla f_{[0,1]}(x) = \frac{\nabla f(x) - \mu x}{L}$ now lying within the sector $[0, 1]$. This loop transformation allows the algorithms $\Sigma_N$ to be expressed as
\[
x_{k+1} = \sum_{j=0}^{N-1} \kappa \theta(j) x_{k-j} - \frac{L - \mu}{L} \nabla f_{[0,1]}(y_k).
\]
whose $Z$-transform, with $u_{[0,1]}(y_k) = \nabla f_{[0,1]}(y_k)$, is
\[
X(z) = - \left( \frac{L - \mu}{L} \right) \left( \sum_{j=0}^{N-1} \kappa \theta(j) z^{-j} I_n \right)^{-1} U_{[0,1]}(z),
\]
and
\[
Y(z) = \left( \sum_{j=0}^{N-1} \kappa \theta(j) z^{-j} I_n \right) X(z).
\]
By using dimension reduction [15], the transfer function of the system of interest is then reduced to
\[
\frac{Y(z)}{U_{[0,1]}(z)} = G_N(z) = - \left( \frac{L - \mu}{L} \right) \frac{\sum_{j=0}^{N-1} \kappa \theta(j) z^{-j}}{z + \sum_{j=0}^{N-1} \kappa \theta(j) z^{-j}}.
\]
This transfer function allows the algorithm to be expressed as the feedback interconnection of the linear system $G_N(z)$ with the nonlinearity $u_{[0,1]}(y_k) = \nabla f_{[0,1]}(y_k)$ as illustrated in Figure 2.

The stability of such feedback systems can be verified using standard techniques such as constructing Lyapunov functions (e.g., those proposed by Tsypkin [26] and Szegö [24]) and searching for Zames-Falb multipliers. These results give a framework by which one can compute many system properties, such as reachable sets, stability margins, exponential convergence rates and local stability regions [23] for instance. For the sake of brevity, the details of these methods are not included here. It is stressed that this analysis is applicable for all functions satisfying the sector conditions, generalising the analysis to the whole class of functions $f(x) \in S^m_{\mu,L}$.

The design of Proposition 5 can then be considered from the open loop perspective, as the parameters $\theta$ were set

![Figure 1: Variations of the root radius of $p_N(r; m(\lambda_i/L))$ as a function of $m(\lambda_i/L)$ when $\kappa = 0.01$. The root radius at $m(\lambda_i/L) = m(\kappa)$ dramatically drops according to (20) but increases elsewhere and in fact even exceeds one. This results in a divergence which is controlled using a switching controller.](image-url)
such that the poles of $G_N(z)$ were fixed to $\gamma$. Even though these poles are Schur, and hence $G_N(z)$ is stable, this does not guarantee the stability of the overall feedback loop. In fact, it is known that introducing feedback can in fact lead to instability, as reflected in the analysis of Section B. A more nuanced design criteria would then be posed in the $z$-domain and take into consideration the passivity of the feedback structure more explicitly, in a manner similar to [27]. This approach was not adopted here as analytic results were desired.

IV. Control by Restarting

To stabilise and recover the performance of $\Sigma_N$, the use of adaptive restarting [19] as a form of switching control is proposed. The typical notion of restarting an algorithm means to perform a gradient descent step at a given iteration to promote monotonic convergence and was initially applied after a pre-defined iteration number [17]. In [19], an improved adaptive restarting approach was proposed where the gradient descent step was only applied if the projected iterates caused a certain condition to trigger, with a typical trigger being that the projected iterate would increase the value of the function as in $f(x_{k+1}) \geq f(x_k)$. This is the trigger condition adopted here although others exist, such as the gradient condition of [19]. When such a condition is triggered then the algorithm switches to $\mathcal{GD}$, guaranteeing a decrease in the function value, before switching back to the non-monotonic algorithm (such as $\mathcal{FG}$) to take advantage of its faster convergence rate. Here, this notion is generalised to switch between the many algorithms in the set $\Omega = \{\Sigma_j : j = 1, \ldots, N\}$, not just between $\mathcal{GD}$ and $\mathcal{FG}$.

Denote $T_N(x_k, \ldots, x_{k+1-N})$ as the iterate update operator for $\Sigma_N$, such that (31) can be written more compactly as

$$x_{k+1} = T_N(x_k, \ldots, x_{k+1-N}). \tag{32}$$

With this notation the implementation of the fast gradient method with adaptive restarting can be expressed as Algorithm 1 (where a function based restart is used).

This adaptive approach is generalised in Algorithm 2 to allow the algorithms to switch between the whole set $\Omega$.

A second scheme that also guarantees monotonic convergence is to update each algorithm in the set $\Omega$ simultaneously and then to select the iterate that minimises the cost function the most. This is illustrated in the multi-legged algorithm $\Sigma^{ml}_N$ of Algorithm 3, so named because the algorithm tentatively steps from the point $x_k$ in various directions. Monotonic convergence is guaranteed with this scheme for $f(x) \in S^N_{\mu, L}$ by including the $\mathcal{GD} = \Sigma_1$ algorithm in $\Omega$.

---

**Algorithm 1 $\mathcal{FG}$ with Adaptive Restarting**

**Require:** $x_0 \in \mathbb{R}^n$, $x_{-1} = x_0$, $L$ and $\beta$.

**for** $k = 0, 1, 2, \ldots$ **do**

- $\hat{x}_{k+1} = T_2(x_k, x_{k-1})$
- **if** $f(\hat{x}_{k+1}) - f(x_k) > 0$ **then**
  - $x_{k+1} = T_1(x_k)$
- **else**
  - $x_{k+1} = \hat{x}_{k+1}$
**end if**

**end for**

**Algorithm 2 $\Sigma_N$ with Adaptive Restarting ($\Sigma^{ml}_N$)**

**Require:** $N \in \mathbb{N}$, $x_{1-N,0} \in \mathbb{R}^N$, $L$ and $\theta$.

**for** $k = 0, 1, 2, \ldots$ **do**

- $\hat{x}_{k+1} = T_N(x_k, x_{k-1}, \ldots, x_{k+1-N})$
- **if** $f(\hat{x}_{k+1}) - f(x_k) > 0$ **then**
  - $\hat{x}_{k+1} = T_{N-1}(x_k, x_{k-1}, \ldots, x_{k+2-N})$
- **if** $f(\hat{x}_{k+1}) - f(x_k) > 0$ **then**
  - $\hat{x}_{k+1} = T_{N-2}(x_k, x_{k-1}, \ldots, x_{k+3-N})$
- **end if**
- **end if**
- $x_{k+1} = \hat{x}_{k+1}$
**end if**

**end for**

**Algorithm 3 Multi-Legged Algorithm ($\Sigma^{ml}_N$)**

**Require:** $N \in \mathbb{N}$, $x_{1-N,0} \in \mathbb{R}^N$, $L$ and $\theta$.

**for** $k = 0, 1, 2, \ldots$ **do**

- $\hat{x}_{k+1,1} = T_1(x_k)$
- $\hat{x}_{k+1,2} = T_2(x_k, x_{k-1})$
- $\hat{x}_{k+1,3} = T_N(x_k, x_{k-1}, \ldots, x_{k+1-N})$

Define $X_{k+1} = \{\hat{x}_{k+1,1}, \ldots, \hat{x}_{k+1,N}\}$

$$x_{k+1} = \min_{x \in X_{k+1}} f(x).$$

**end for**

Controlling the algorithms by adaptive restarting in this way allows algorithms for which the operator $T_N$ is not be globally contracting to be considered. The use of such operators was found to speed up convergence (as discussed in the examples of Section V) and suggests connections.
to the fundamental limitations between controller performance and robustness.

The need to evaluate the function at each iteration increases the computational cost of running these algorithms, and this could be reduced by using the gradient based trigger condition. Memory requirements are also increased with these methods. But, as demonstrated in the following examples section, adaptive restarting brings both added performance and robustness over the fast gradient method, with robustness being highlighted as a necessary feature for the use of these algorithms in practice [2]. It is also noted that these approaches are highly parrelisable which should recover some computational speed in an efficient implementation.

V. Examples

This section evaluates the performance of the proposed first-order algorithms via several numerical examples.

A. Quadratic problems

1) Hessian eigenvalues clustered at L: We begin with the follow quadratic functions whose eigenvalues are clustered at the Lipschitz constant $L$

$$f_{ex:1}(x) = (x^{(1)})^2 + 1x + \sum_{j=0}^{n-2} (L-j)(x^{(j+2)})^2.$$  \hspace{1cm} (33)

The minimisation of this function with $\mu = 1$, $L = 10^4$ and $n = 10^3$ from an initial condition of $x_0 = 0$ using $\Sigma_{6}^{Rec}$, $\Sigma_{6}^{Re}$, $\Sigma_{6}^{m1}$ with adaptive restart and the multi-legged algorithm $\Sigma_{6}^{Nest}$ is shown in Figures 3a and 3b. As a reminder, $\Sigma_{6}^{Rec}$ corresponds to gradient descent, $\Sigma_{6}^{m1}$ is the multi-legged algorithm for $N = 6$ whilst $\Sigma_{6}^{Re}$ and $\Sigma_{6}^{Nest}$ are respectively the fast gradient method and the proposed algorithms with adaptive restart. Also plotted on this figure is the trace of the algorithm $\Sigma_6$ without adaptive restarting, denoted $\Sigma_6^{NR}$. Increasing the memory $N$ led to faster convergence speed, with $\Sigma_{6}^{Rec}$ being an order of magnitude faster than the fast gradient method. This was until the function values became small from which the function based adaptive restarting scheme struggled. Without restarting, the iterates of $\Sigma_6$ were actually divergent for this problem, highlighting the need for stabilisation by restarting. This simple example shows how the fast gradient method may not be “optimal” for many strongly convex problems and initial conditions.

2) A more even distribution of Hessian eigenvalues: In contrast, the following quadratic function

$$f_{ex:2}(x) = \frac{1}{2}x^T \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 1 & \ldots & 1 \\ 1 & 1 & 3 & \ldots & 1 \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 1 & 1 & \ldots & \ldots & n \end{bmatrix} x + \frac{1}{2} \begin{bmatrix} 2 \\ 3 \\ \vdots \\ n \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ \vdots \\ n \end{bmatrix} x.$$  \hspace{1cm} (34)

has a more even distribution of eigenvalues. The minimisation of this function with $n = 10^3$ is shown in Figures 3d and 3c with the fastest convergent rate once again being observed with $\Sigma_{6}^{m1}$. The condition number of this function is $1.37 \times 10^4$, but because the eigenvalues are spread more evenly in the range $[\mu, L]$, the performance improvement of the algorithms $\Sigma_{6}^{Re}$ were not as substantial as for $f_{ex:1}(x)$. However, $\Sigma_{6}^{m1}$ still obtains a significantly faster convergence rate than $f_{ex:2}(x)$. 

B. Nesterov’s counter-example

Any proposed first-order method that compares itself against the fast gradient method has to take into consideration that algorithm’s optimality. This label is discussed here.

The fast gradient method is referred to as an “optimal” algorithm [18] Chapter 2, with the justification being that there exists a function

$$f_{Nest}(x) = \frac{1}{2}\mu \|x\|^2 + \frac{\mu(Q_f - 1)}{4} \left\{ \frac{1}{2} (x^{(1)})^2 + \sum_{i=1}^{\infty} (x^{(i)} - x^{(i+1)})^2 - x^{(1)} \right\}.$$  \hspace{1cm} (35)

for which from the specific initial condition $x_0 = 0$, no algorithm can perform better. This “worst-case” function is defined for signals in a Hilbert space $x \in l_2$ and is strongly convex since

$$\mu I \leq f_{Nest}'(x) \leq \mu Q_f I.$$  \hspace{1cm} (36)

From the first order optimality condition

$$f_{Nest}'(x^*) = 0,$$  \hspace{1cm} (37)

it follows that any sequence that minimises this function must satisfy

$$\frac{Q_f + 1}{Q_f - 1} (x^*)^{(1)} - (x^*)^{(2)} = 1$$ \hspace{1cm} (38a)

$$(x^*)^{(k+1)} - 2 \frac{Q_f + 1}{Q_f - 1} (x^*)^{(k)} + x^*(k-1) = 0, \hspace{1cm} k = 2, 3, \ldots$$ \hspace{1cm} (38b)

This sequence can be regarded as a linear system with characteristic equation

$$q^2 - 2 \frac{Q_f + 1}{Q_f - 1} q + 1 = 0.$$ \hspace{1cm} (39)

The smallest root of this equation is $q = \sqrt{\frac{Q_f - 1}{Q_f + 1}}$ [18] Chapter 2 and so the optimal sequence then satisfies

$$(x^*)^{(k)} = q^k, \hspace{1cm} k = 1, 2, \ldots.$$ \hspace{1cm} (40)

hence

$$\|x_k - x^*\|^2 \geq \left( \frac{Q_f - 1}{Q_f + 1} \right)^{2k} \|x_0 - x^*\|^2.$$ \hspace{1cm} (41)

This lower bound for the error $\|x_k - x^*\|$ is the same as the upper bound from the fast gradient method given in (40). Hence, no algorithm can perform better for this particular function and initial condition, justifying its title of being an optimal method.
To compare the proposed algorithms against such a benchmark, the minimisation of a truncated version of this worst-case function
\[ F_{\text{Next}}(x) = \frac{1}{2} \mu \|x\|^2 + \frac{\mu (Q_f - 1)}{4} \left\{ \frac{1}{2} (x(1))^2 + \sum_{i=1}^{n} (x(i) - x(i+1))^2 \right\} - x(i) \] (42)
was considered with \( n = 10^3 \) and \( Q_f = 10^6 \) starting from the same initial condition \( x_0 = 0 \). The results of this minimisation are shown in Figure 4 where the rate \( \|x_k - x^*\|^2/\|x_0 - x^*\|^2 \) generated by the algorithm \( \Sigma^m_{\text{ml}} \) and the bound from \( \Sigma^m_{\text{ml}} \) are plotted. Notably, for the truncated function, \( \Sigma^m_{\text{ml}} \) could actually violate the bound. However, no claim is made here against the optimality of \( \mathcal{F} \mathcal{G} \) for \( f_{\text{wc}}(x) \). The violation was only achieved by truncating the function and sufficiently increasing the condition number \( Q_f \) such that, by a uniqueness of solution argument, the minimiser \( \tilde{x} \) is corrupted, making the bound in \( \Sigma^m_{\text{ml}} \) rather irrelevant for \( f_{\text{wc}}(x) \).

C. Non-convex functions

Due to applications in machine learning, recent results on accelerated gradient methods have focussed on functions that are only weakly convex or even non-convex \cite{12} \cite{11}. For such problems, gradient descent typically gets stuck in local minima/saddle points from which random perturbations in the gradient may have to be added to escape \cite{11}. It has also been noted that introducing momentum (via the fast gradient) can be beneficial for such problems, as the iterates can then overshoot the local minima \cite{12}.

This idea of adding momentum for non-convex problems is extended here to the accelerated \( \Sigma^m_{\text{ml}} \) method. No rigorous convergence analysis is provided as that goes beyond the scope of the paper. The justification of using \( \Sigma^m_{\text{ml}} \) for such problems is that the generated points \( y_k \) from which the gradient step is taken from may not lie within the convex hull of the iterate history. This allows the algorithm’s iterates to leave the local minima. In essence, the randomness of the stochastic gradient method is replaced by the jumps and local instabilities of the
Figure 4: Convergence of the truncated version of the Nesterov’s counter-example $f_{Nest}$ using $\Sigma^m_0$. The truncation allows the generated iterates to exceed the bound given in (41).

deterministic $\Sigma^m_N$ algorithm.

For these problems, the choice of $\mu$ and $L$ was found to significantly influence the convergence. Making $\mu/L$ too small meant that $\Sigma^m_N$ converged to a local minimum via gradient descent while making $\mu/L$ too large meant that even the gradient descent algorithm became divergent. Best performance was achieved when these parameters were set such that the resulting algorithms were on the boundary between contracting and diverging.

Two benchmark non-convex functions are examined; Rosenbrock’s banana function and the Rastrigin function. Again, it is highlighted that a deterministic gradient-based method was used for these minimisations.

1) Rosenbrock banana function: The Rosenbrock banana function $[21]$ with $n = 2$

$$f_{Ros}(x) = (1 - x^{(1)})^2 + 100(x^{(2)} - x^{(1)})^2$$

has a global minimiser at $(1,1)$ at the bottom of a valley. For gradient methods, converging to this valley is trivial but then travelling down to the global minimiser is exhaustive. Figure 5 shows the minimisation of this function using the algorithms $GD$, $FG$ and $\Sigma^m_N$ with $N = 3, \ldots , 9$, $\mu = 10^{-5}$ and $L = 0.9 \times 10^3$ from $x_0 = (-1,1)$. The figure shows the convergence of gradient descent being slow both to and in the valley, and furthermore, for this choice of $\mu$ and $L$, the fast gradient method was divergent. This illustrates the well-known lack of robustness of this algorithm $[2]$. The algorithms $\Sigma^m_N$ performed well for this function, with $\Sigma^m_0$ finding a function value of $7.58 \times 10^{-12}$ in 43 iterations (with this number of iterations not accounting for the preliminary steps taken by the algorithm in the restarting). This is lower than the function value of $1.35 \times 10^{-10}$ in 185 iterations of the Nelder-Mead method $[16]$.

2) Rastrigin function: A more challenging benchmark non-convex function is the Rastrigin function $[20]$

$$f_{Rast}(x) = 10n + \sum_{j=1}^{n} x_j^2 - 10\cos(2\pi x_j)$$

which has a global minima at the origin and many local minima. The minimisation of this function with $\mu = 1$, $L = 140$ and $n = 2$ is considered here. These parameters were chosen from considering the balance between algorithm convergence and ability to leave local minima, and, strikingly, this choice of $L$ is greater than the curvature at the minimiser. Figure 6a shows the trace of the iterates for this minimisation using $\Sigma^m_N$ from the initial conditions $x_0 = (5,5)$ and $x_0 = (-5,-3)$. From $x_0 = (5,5)$, a function value of $f_{Rast} = 10^{-6}$ was obtained after 463 iterations. However, the algorithm performed less well from $x_0 = (-5,-3)$, with the obtained functions values fluctuating around $f_{Rast} \approx 5$ after an initial decrease from $f_{Rast} = 34.4$. The local minima of the function can be clearly seen in this figure, with the iterates leaving the local dimples and fluctuating near the minimiser.

Figure 6b further evaluates the influence of the initial conditions for this minimisation of this function. The figure plots the log of the minimum value of the cost function obtained by the $\Sigma^m_0$ algorithm in 1000 iterations. Typically, for initial conditions on the 8 pointed star apparent in the figure, function values between $10^{-6}$ and $10^{-3}$ were found. However, everywhere else, function values only between $10^{-3}$ and $10^{-1}$ were obtained. In contrast, for this function, gradient descent got trapped in the local minima and the fast gradient method was unstable for this choice of $\mu$ and $L$ which were chosen to escape the local minima.
The iterates are able to leave the local minima before fluctuating against this mode, the algorithms lose robustness, which is recovered by the use of a switching controller. This switching controller uses adaptive restarting such that at each iteration, the algorithms switch between a set of potential iterates to recover monotonic convergence and the acceleration. Numerical examples showcase the benefits of the proposed approach, both for strongly convex and non-convex functions. For example, the algorithms were shown to find values between $10^{-1}$ and $10^{-3}$ of the Rastrigin function from generic initial conditions and values between $10^{-6}$ and $10^{-8}$ with initial conditions along several lines intersecting the global minimiser. Also, when applied to the Rosenbrock banana function, the proposed algorithms could find a function value of $7.58 \times 10^{-12}$ in 43 iterations from an initial condition of $x_0 = (-1,1)$. It is hoped that the proposed methods will encourage further interest in accelerating first order methods still further using the algorithm memory and to robustify them by closing feedback loops.

VI. CONCLUSIONS

A set of first-order algorithms was proposed for minimising strongly convex functions. The proposed algorithms have a similar structure to the fast gradient method but use an increased number of historical iterates, giving them memory. The algorithms are parameterised by considering quadratic functions such that the error associated to the dominant eigenvalue of the Hessian converges at the rate $1 - (\mu/L)^{\frac{1}{N}}$ for generic $N \in \mathbb{N}$. By parametrising against this mode, the algorithms lose robustness, which is recovered by the use of a switching controller. This switching controller uses adaptive restarting such that at each iteration, the algorithms switch between a set of potential iterates to recover monotonic convergence and the acceleration. Numerical examples showcase the benefits of the proposed approach, both for strongly convex and non-convex functions. For example, the algorithms were shown to find values between $10^{-1}$ and $10^{-3}$ of the Rastrigin function from generic initial conditions and values between $10^{-6}$ and $10^{-8}$ with initial conditions along several lines intersecting the global minimiser. Also, when applied to the Rosenbrock banana function, the proposed algorithms could find a function value of $7.58 \times 10^{-12}$ in 43 iterations from an initial condition of $x_0 = (-1,1)$. It is hoped that the proposed methods will encourage further interest in accelerating first order methods still further using the algorithm memory and to robustify them by closing feedback loops.

REFERENCES

[1] R. O. Alli-Oke and W. P. Heath, “A secant-based Nesterov method for convex functions,” Optimization Letters, vol. 11, no. 1, pp. 81–105, 2017.
[2] N. S. Aybat, A. Fallah, M. Gurbuzbalaban, and A. Ozdaglar, “Robust accelerated gradient method,” arXiv preprint arXiv:1805.10579, 2018.
[3] M. Betancourt, M. I. Jordan, and A. C. Wilson, “On symplectic optimization,” arXiv preprint arXiv:1802.03653, 2018.
[4] V. D. Blondel, M. Gurbuzbalaban, A. Megretski, and M. L. Overton, “Explicit solutions for root optimization of a polynomial family with one affine constraint,” IEEE Transactions on Automatic Control, vol. 57, no. 12, pp. 3078–3089, 2012.
[5] R. Drummond, S. R. Duncan, and G. Valmorbida, “The discrete time Popov criterion,” Technical report.
[6] R. Drummond, J. L. Jerez, and E. C. Kerrigan, “Gradient filter methods for predictive control with simple constraints,” in Proc. of the UKACC International Conference on Control. Loughborough, UK. IEEE, 2014, pp. 679–684.
[7] M. Fazlyab, M. Morari, and V. M. Preciado, “Design of first-order optimization algorithms via sum-of-squares programming,” arXiv preprint arXiv:1803.10928, 2018.
[8] G. Goh, “Why momentum really works,” Distill, vol. 2, no. 4, p. e6, 2017.
[9] W. M. Haddad and D. S. Bernstein, “Explicit construction of quadratic Lyapunov functions for the small gain, positivity, circle, and Popov theorems and their application to robust stability. Part II: Discrete-time theory,” International journal of robust and nonlinear control, vol. 4, no. 2, pp. 249–265, 1994.
[10] B. Hu and L. Lessard, “Dissipativity theory for Nesterov’s accelerated method,” arXiv preprint arXiv:1706.04381, 2017.
[11] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, “How to escape saddle points efficiently,” arXiv preprint arXiv:1708.08887, 2017.
[12] C. Jin, P. Netrapalli, and M. I. Jordan, “Accelerated gradient descent escapes saddle points faster than gradient descent,” arXiv preprint arXiv:1711.10456, 2017.
