Explicit Stabilised Gradient Descent for Faster Strongly Convex Optimisation

Armin Eftekhari  
Alan Turing Institute,  
London, UK  
eaeftekhari@turing.ac.uk

Bart Vandereycken  
Section of Mathematics,  
University of Geneva,  
Geneva, Switzerland  
bart.vandereycken@unige.ch

Gilles Vilmart  
Section of Mathematics,  
University of Geneva,  
Geneva, Switzerland  
Gilles.Vilmart@unige.ch

Konstantinos C. Zygalakis  
School of Mathematics,  
University of Edinburgh,  
Edinburgh, UK  
k.zygalakis@ed.ac.uk

Abstract

We introduce the explicit stabilised gradient descent method (ESGD) for strongly convex optimisation problems. This new algorithm is based on explicit stabilised integrators for stiff differential equations, a powerful class of numerical schemes to avoid the severe step size restriction faced by standard explicit integrators. For optimising quadratic and strongly convex functions, we prove that ESGD nearly achieves the optimal convergence rate of the conjugate gradient algorithm, and the suboptimality of ESGD diminishes as the condition number of the quadratic function worsens. We show that this optimal rate is obtained also for a partitioned variant of ESGD applied to perturbations of quadratic functions. In addition, numerical experiments on general strongly convex problems show that ESGD outperforms Nesterov’s accelerated gradient descent.

1 Introduction

Optimisation is at the heart of many applied mathematical and statistical problems, while its beauty lies in the simplicity of describing the problem in question. In this work, given a function \( f: \mathbb{R}^d \to \mathbb{R} \), we are interested in finding a minimiser \( x^* \in \mathbb{R}^d \) of the program

\[
\min_{x \in \mathbb{R}^d} f(x). 
\]  
(1)

We make the common assumption throughout that \( f \in \mathcal{F}_{\mu,L} \), namely the set of \( \mu \)-strongly convex differentiable functions that have \( L \)-Lipschitz continuous derivative [1]. Corresponding to \( f \) is its gradient flow, defined as

\[
\frac{dx}{dt} = -\nabla f(x), \quad x(0) = x_0 \in \mathbb{R}^d, 
\]  
(2)

where \( x_0 \) is its initialisation. It is easy to see that traversing the gradient flow always reduces the value of \( f \). Indeed, for any positive \( h \), it holds that

\[
f(x(h)) - f(x_0) = -\int_0^h \| \nabla f(x(t)) \|_2^2 \, dt. 
\]  
(3)

Preprint. May 21, 2018.
By discretising the gradient flow in (2), we can design various optimisation algorithms for Program (1). For example, by substituting in (2) the approximation
\[
\frac{dx}{dt} \approx \frac{x_{n+1} - x_n}{h}
\]
at \(x_n\), we obtain the gradient descent (GD), namely
\[
x_{n+1} = x_n - h \nabla f(x_n),
\]
where \(h > 0\) is the step size [1]. For this discretisation to remain stable, that is, for the GD update in (5) to remain close to the gradient flow and for the value of \(f\) to reduce in every iteration, the step size \(h\) must not be too large. Indeed, a well-known shortcoming of GD is that we must take \(h \leq 2/L\) to ensure stability, otherwise \(f\) might increase from one iteration to the next [1]. This drawback of GD has motivated a number of research works that use different discretisations of (2) to design better optimisation algorithms [2, 3, 4, 5, 6]. For example, substituting in (2) the approximation (4) at \(x_{n+1}\) instead of \(x_n\), we arrive at the update
\[
x_{n+1} = x_n - h \nabla f(x_{n+1}),
\]
which is known as the implicit Euler method in numerical analysis [7] because, as the name suggests, it involves solving (6) for \(x_{n+1}\). It is not difficult to see that, unlike GD, there is no limit on the step size \(h\) for the implicit Euler method, a property known in the numerical analysis jargon as A-stability [7]. Moreover, it is easy to verify that \(x_{n+1}\) in (6) is also the unique minimiser of the program
\[
\min_{x \in \mathbb{R}^d} hf(x) + \frac{1}{2} \|x - x_n\|^2_2,
\]
and in fact the map from \(x_n\) to \(x_{n+1}\) is known in the optimisation literature as the proximal map of the function \(hf\) [8]. Unfortunately, even if \(\nabla f\) is known explicitly, solving (6) for \(x_{n+1}\) or equivalently computing the proximal map is often just as hard as solving Program (1), with a few notable exceptions [8]. This setback severely limits the applicability of the proximal algorithm in (6) for solving Program (1).

**Contributions.** With this motivation, we propose the Explicit Stabilised Gradient Descent (ESGD) method for solving Program (1). ESGD offers the best of both worlds, namely the computational tractability of GD (explicit Euler method) and the stability of the proximal algorithm (implicit Euler method). Inspired by [2], ESGD uses explicit stabilised methods [9, 10, 11] to discretise the gradient flow (2). In numerical analysis, explicit stabilised methods provide a computationally efficient alternative to the implicit Euler method for stiff differential equations, where standard integrators face a severe step size restriction, in particular in high dimensions for spatial discretisations of diffusion PDEs, see the review [12]. Every iteration of ESGD consists of \(s\) internal stages, where each stage performs a simple GD-like update. Unlike GD however, ESGD does not decay monotonically along its internal stages, which allows it to take longer steps and travel faster along the gradient flow. After \(s\) internal stages, ESGD ensures that its new iterate is stable, namely the value of \(f\) indeed decreases after each iteration of ESGD.

The rest of this paper is organised as follows. Section 2 formally introduces ESGD, which is summarised in Algorithm 1 and accessible without reading the rest of this paper. In Section 3, we then quantify the performance of ESGD for solving strongly convex quadratic programs, while in Section 4 we introduce and study theoretically a partitioned variant of ESGD applied to perturbations of quadratic functions. Then in Section 5, we empirically compare ESGD with other first-order optimisation algorithms and conclude that ESGD improves over the state of the art in practice. This paper concludes with an overview of the remaining theoretical challenges.

## 2 Introducing Explicit Stabilised Gradient Descent

Let us start with the simple scalar program
\[
\min_{x \in \mathbb{R}} \frac{\lambda x^2}{2},
\]
where \(\lambda\) is positive, and consider the corresponding gradient flow
\[
\frac{dx}{dt} = -\lambda x, \quad x(0) = x_0 \in \mathbb{R},
\]
also known as the Dahlquist test equation [7]. It is obvious from (9) that \( \lim_{t \to \infty} x(t) = 0 \) and any sensible optimisation algorithm should provide iterates \( \{x_n\}_n \) with a similar property, that is,

\[
\lim_{n \to \infty} x_n = 0.
\]

For GD, which corresponds to the explicit Euler discretisation of the gradient flow, it easily follows from (2) that

\[
x_{n+1} = R_{gd}(z)x_n, \quad R_{gd}(z) = 1 + z, \quad z = -\lambda h,
\]

where \( R_{gd} \) is the stability polynomial. Hence, (10) holds if \( z \in \mathcal{D}_{gd} \), where the stability domain \( \mathcal{D}_{gd} \) of GD is defined as

\[
\mathcal{D}_{gd} = \{ z \in \mathbb{C}; |R_{gd}(z)| < 1 \}.
\]

That is, (10) holds if \( h \in (0, 2/\lambda) \), which imposes a severe limit on the time step \( h \) when \( \lambda \) is large. Beyond this limit, namely for larger step sizes, the iterates of GD might not necessarily reduce the value of \( f \) or, put differently, the explicit Euler method might no longer be a faithful discretisation of the gradient flow. At the other extreme, for the proximal algorithm which corresponds to the implicit Euler discretisation of the gradient flow in (9), it follows from (6) that

\[
x_{n+1} = R_{pa}(z)x_n, \quad R_{pa}(z) = \frac{1}{1 - z}, \quad z = -\lambda h,
\]

with the stability domain \( \mathcal{D}_{pa} = \{ z \in \mathbb{C}; |R_{pa}(z)| < 1 \} \). Therefore (10) holds for any positive step size \( h \). This property is known as A-stability of a numerical method [7]. Unfortunately, the proximal algorithm (implicit Euler method) is often computationally intractable particularly in higher dimensions. In numerical analysis, explicit stabilised methods for discretising the gradient flow offer the best of both worlds, as they are not only explicit and thus computationally tractable, but they also share some favourable stability properties of the implicit method. Our main contribution in this work is adapting these methods for optimisation, as detailed next.

For discretising the gradient flow (9), the key idea behind explicit stabilised methods is to relax the requirement that every step of explicit Euler method should remain stable, namely, faithful to the gradient flow. This relaxation in turn allows the explicit stabilised method to take longer steps and traverse the gradient flow faster. To be specific, applying any given explicit Runge-Kutta method with \( s \) stages (i.e. using \( s \) evaluations of \( \nabla f \) per step to (9) yields a recursion of the form

\[
x_{n+1} = R_s(z)x_n, \quad R_s(z) = 1 + z + a_2z^2 + \ldots + a_sz^s,
\]

with the corresponding stability domain \( \mathcal{D}_s = \{ z \in \mathbb{C}; |R_s(z)| < 1 \} \). We wish to choose \( \{a_j\}_{j=2}^s \) to maximise the step size \( h \) while ensuring that \( z = -h\lambda \) still remains in the stability domain \( \mathcal{D}_s \), namely for the update of the explicit stabilised method to remain stable. More formally, we wish to solve

\[
\max_{a_2, \ldots, a_s} l_s \quad \text{subject to} \quad |R_s(z)| \leq 1, \quad \forall z \in [-l_s, 0].
\]

The solution to (15) is \( l_s = 2\lambda^2 \) and, after substituting the optimal values for \( \{a_j\}_{j=1}^s \) in (14), we find that the unique corresponding \( R_s(z) \) is the shifted Chebyshev polynomial \( R_s(z) = T_s(1 + z/s^2) \) where \( T_s(\cos \theta) = \cos(s\theta) \) is the Chebyshev polynomial of the first kind with degree \( s \) [12]. As we can see in Figure 1, \( R_s(z) \) equi-oscillates between \(-1\) and \(1\) on \( z \in [-l_s, 0] \), see the red curve in this Figure 1(b). That is, after every \( s \) internal stages, the new iterate of the explicit stabilised method remains stable and faithful to (9), while travelling the most along the gradient flow.

Numerical stability is still an issue for the explicit stabilised method outlined above, particularly for the values of \( z = -\lambda h \) for which \( |R_s(z)| = 1 \). As seen in Figure 1(a), even the slightest numerical imperfection will land us outside of the stability domain \( \mathcal{D}_s \), which might make the algorithm unstable. In addition, for such values of \( z \), the new iterate \( x_{n+1} \) is not necessarily any closer to the minimiser, here the origin. Traditionally, for a damping factor \( \eta \), one therefore tightens the stability requirement to \( |R_s(z)| \leq \alpha_s(\eta) < 1 \) for every \( z \in [-l_s, -\delta_\eta] \). More specifically, for positive \( \eta \), we can take

\[
R_s(z) = \frac{T_s(\omega_0 + \omega_1 z)}{T_s(\omega_0)}, \quad \omega_0 = 1 + \frac{\eta}{s}, \quad \omega_1 = \frac{T_s(\omega_0)}{T_s'(\omega_0)},
\]

in which case \( R_s(z) \) oscillates between \(-\alpha_s(\eta)\) and \( \alpha_s(\eta) \) for every \( z \in [-l_s, -\delta_\eta] \), where

\[
\alpha_s(\eta) = \frac{1}{T_s(\omega_0)} < 1, \quad l_{s, \eta} = \frac{1 + \omega_0}{\omega_1}.
\]
3 Strongly Convex Quadratic Programming

Consider the Program (1) with

$$f(x) = \frac{1}{2} x^T A x - b^T x,$$

where $A \in \mathbb{R}^{d \times d}$ is a positive definite and symmetric matrix, and $b \in \mathbb{R}^d$. As the next result shows, that convergence of a Runge-Kutta method (such as GD, proximal algorithm) depends$^1$ on the eigenvalues of $A$.

**Proposition 1.** For solving Program (1) with $f$ in (19), consider an optimisation algorithm with the stability function $R$, for example $R = R_{gd}$ for GD or $R = R_{pa}$ for proximal algorithm, and let

---

$^1$The proof can be found in the supplementary material.
Algorithm 1  Explicit Stabilised Gradient Descent (ESGD) for solving Program (1)

**Input:** The gradient $\nabla f$ of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, Initialisation $x_0 \in \mathbb{R}^d$.

**Body:** Until a convergence criterion is met, repeat:
- Set $x^0_n = x_n$ and $x^1_n = x^0_n - h \mu_1 \nabla f(x_n)$.
- For $j \in \{2, \cdots, s\}$, repeat:
  \[
  x^j_n = -\mu_j h \nabla f(x^{j-1}_n) + \nu_j x^{j-1}_n - (\nu_j - 1) x^{j-2}_n,
  \]
  where
  \[
  \mu_1 = \frac{\omega_1}{\omega_0}, \quad \mu_j = \frac{2 \omega_1 T_{j-1}(\omega_0)}{T_j(\omega_0)}, \quad \nu_j = \frac{2 \omega_0 T_{j-1}(\omega_0)}{T_j(\omega_0)},
  \]
  \[
  \omega_0 = 1 + \frac{\eta}{s^2}, \quad \omega_1 = \frac{T_s(\omega_0)}{T_s'(\omega_0)},
  \]
  and $T_s$ is the Chebyshev polynomial of the first kind with degree $s$.
- Set $x_{n+1} = x^s_n$.

**Output:** Estimate $\hat{x} = x_{n+1}$ of a minimiser of Program (1).

\[
\{x_n\}_{n \geq 0} \text{ be the iterates of this algorithm. Also let } \mu_1 \leq \cdots \leq \mu_d = L \text{ denote the eigenvalues of } A \text{ in nondecreasing order. Then it holds that}
\]
\[
f(x_{n+1}) - f(x^*) \leq \max_{i \in [d]} R^2(\Delta t \lambda_i) (f(x_n) - f(x^*)),
\]
for every $n \geq 0$.

Let us next apply Proposition 1 to both GD and ESGD.

**Gradient descent.** In the case of GD, it is not difficult to verify that
\[
\max_{i \in [d]} R^2_{gd}(-\lambda_i h) = \begin{cases} R^2_{gd}(-\mu h), & \text{if } 0 < h \leq \frac{2}{\mu + L}, \\ R^2_{gd}(-Lh), & \text{if } h \geq \frac{2}{\mu + L}, \end{cases}
\]
where we used (11). It follows from (21) that we must take $h \in (0, 2/L)$ for GD to be stable, namely for GD to reduce the value of $f$. In addition, one obtains the best possible decay by choosing $h = 2/(\mu + L)$. More specifically, we have that
\[
\min_h \max_{i \in [d]} R^2_{gd}(-\lambda_i h) = R^2_{gd} \left( \frac{-2 \mu}{\mu + L} \right) = R^2_{gd} \left( \frac{-2L}{\mu + L} \right) = \left( \frac{\kappa - 1}{\kappa + 1} \right)^2,
\]
where $\kappa := L/\mu$, is the condition number of matrix $A$. That is, the best convergence rate for GD is predicted by Proposition 1 as
\[
f(x_{n+1}) - f(x^*) \leq \left( \frac{\kappa - 1}{\kappa + 1} \right)^2 (f(x_n) - f(x^*)),
\]
achieved with the step size $h = 2/(\mu + L)$. Remarkably, the same conclusion holds for any function in $F_{\mu, L}$, as discussed in [16].

**Explicit stabilised gradient descent.** In the case of Algorithm 1, there are three different interlinked parameters that need to be chosen, namely the step size $h$, the number of internal stages $s$, and the damping factor $\eta$. For fixed positive $\eta$, let us next select $h$ and $s$ such the numerator of (16), namely $|T_s(\omega_0 + \omega s)|$, is bounded by one. Equivalently, we take $h, s$ such that
\[
-1 \leq \omega_0 - \omega_1 Lh \leq \omega_0 - \omega_1 \mu h \leq 1.
\]
For an efficient algorithm, we choose the smallest step size $h$ and the smallest number $s$ of internal stages such that (24) holds. More specifically, (24) dictates that $\kappa \leq (1 + \omega_0)/(-1 + \omega_0) = 1 + 2s^2/\eta$, see (18). This in turn determines the parameters $h, s$ as
\[
h = \frac{\omega_0 - 1}{\omega_1 \mu}, \quad s = \left\lceil \sqrt{(\kappa - 1)\eta/2} \right\rceil.
\]
Figure 2: This figure considers the non-asymptotic scenario and plots, as a function of the condition number $\kappa$, the difference $c_{\text{esgd}}(\kappa) - c_{\text{opt}}(\kappa)$ of the (effective) convergence rate of ESGD compared to the optimal one, for many fixed values of the damping parameter $\eta$. In this plot, we observe that $c_{\text{esgd}}(\kappa) - c_{\text{opt}}(\kappa)$ decays as $C_0/\sqrt{\kappa}$ for $\kappa \to \infty$ and for fixed $\eta$, see the slope $-1/2$ in the plot. Numerical evaluations suggest that the constant $C_0 = O(1/\sqrt{\eta})$ becomes arbitrarily small as $\eta$ grows, which corroborates Proposition 2. For comparison, we also include the result $c_{\text{agd}}(\kappa) - c_{\text{opt}}(\kappa)$ for the optimal rate of the Nesterov's Accelerated Gradient Descent (AGD) given by $c_{\text{agd}} = (1 - 2/\sqrt{3\kappa + 1})^2$ [16]. Comparing the two algorithms in Section 5 we find that ESGD outperforms AGD, namely $c_{\text{esgd}} \leq c_{\text{agd}}$ for $\eta \geq \eta_0$, where $\eta_0 \approx 1.17$ is a moderate size constant.

Under (24) and using the definitions in (16,17), we find that $|R_{\text{esgd}}(-\lambda_i h)| \leq \alpha_s(\eta)$ for every $i \in [d]$. Then an immediate consequence of Proposition 1 is that

$$f(x_{n+1}) - f(x^*) \leq \alpha_s(\eta)^2(f(x_n) - f(x^*)).$$

(26)

Given that every iteration of ESGD consists of $s$ internal stages—hence, with the same cost as $s$ GD steps—it is natural to define the effective convergence rate of ESGD as

$$c_{\text{esgd}}(\kappa) := \alpha_s(\eta)^{2/s}.$$ 

(27)

The following result, proved in the supplementary material, evaluates the effective convergence rate of ESGD, as given by (27), at the limit of $\eta \to \infty$.

**Proposition 2.** With the choice of step size $h$ and number $s$ of internal stages in (25), the effective convergence rate $c_{\text{esgd}}(\kappa)$ of ESGD for solving Program (1) with $f$ given in (19) satisfies

$$\lim_{\eta \to \infty} c_{\text{esgd}}(\kappa) = c_{\text{opt}}(\kappa) + O\left(\kappa^{-3/2}\right), \quad c_{\text{opt}}(\kappa) := \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^2,$$

(28)

Above, $c_{\text{opt}}(\kappa)$ is the optimal convergence rate of a first-order algorithm, which is achieved by the conjugate gradient (CG) algorithm for quadratic $f$, see [1]. Put differently, Proposition 2 states that ESGD nearly achieves the optimal convergence rate in the limit of $\eta \to \infty$. Also note that remarkably the performance of ESGD relative to the conjugate gradient improves as the condition number of $f$ worsens, namely as $\kappa$ increases. The non-asymptotic behaviour of ESGD is numerically investigated in Figure 2, corroborating Proposition 2. As illustrated in Section 5, Algorithm 1 also applies to non-quadratic optimization problems.\(^2\)

4 Perturbation of a Quadratic Objective Function

Proposition 2 shows us that in the case of strongly convex quadratic problems one can recover the optimal convergence rate of the conjugate gradient for large values of the damping parameter $\eta$. Here we discuss a modification of the Algorithm 1 to a specific class of nonlinear problems for which we can prove the same convergence rate as in the quadratic case. More precisely, we will consider

$$f(x) = \frac{1}{2} x^T A x + g(x),$$

(29)

\(^2\)For a non-quadratic $f \in F_{\mu,L}$, the parameters $h$ and $s$ should be chosen again using (25), where $\kappa = L/\mu.$
where $A$ is positive definite matrix for which we know the smallest and largest eigenvalue, and $g$ is a $\beta$-smooth convex function [1]. Inspired by [17], we consider the modification$^3$ of Algorithm 1 specifically designed for the minimization of functions of the form (29). We call this method Partitioned Explicit Stabilised Gradient Descent (PESGD) and show in Theorem 1, proved in the supplementary material, that it matches the rate given by the analysis of quadratic problems. PESGD is a numerically stable implementation of the following update

$$x_{n+1} = R_s(-Ah)x_n - B_s(-Ah)\nabla g(x_n), \quad B_s(\zeta) = \frac{1-R_s(\zeta)}{\zeta}$$

(30)

where $R_s$ is the stability function given by equation (16) and $B_s$ is a related polynomial. It is worth noting that this method has only one evaluation of $\nabla g$ per step of the algorithm, which can be advantageous if the evaluations of $\nabla g$ are costly.

**Algorithm 2** Partitioned Explicit Stabilised Gradient Descent (PESGD) for solving Program (29)

**Input:** The gradient $\nabla g$ of a differentiable function $g : \mathbb{R}^d \to \mathbb{R}$, and the matrix $A \in \mathbb{R}^{d \times d}$. Initialisation $x_0 \in \mathbb{R}^d$.

**Body:** Until the convergence criterion is met. repeat:

- Set $x_n^0 = x_n$ and $x_n^1 = x_n^0 - \mu_1 h(Ax_n^0 + h\nabla g(x_n^0))$.

- For $j \in \{2, \cdots, s\}$, repeat:
  
  $$x_n^j = -\mu_j h(Ax_n^{j-1} + h\nabla g(x_n^0)) + \nu_j x_n^{j-1} - (\nu_j - 1)x_n^{j-2},$$

  where the constants $\mu_j, \nu_j$ are defined in (18).

- Set $x_{n+1} = x_n^s$.

**Output:** Estimate $\hat{x} = x_{n+1}$ of a minimiser of Program (1) for $f$ given by (29).

**Theorem 1.** Let $f$ be given by (29) where $A \in \mathbb{R}^{d \times d}$ is a positive definite matrix with largest and smallest eigenvalues $L$ and $\mu$, respectively, and condition number $\kappa = L/\mu$. Let $\gamma > 0$ and $g$ a $\beta$-smooth convex function for which $0 < \beta < C(\eta)\gamma/\mu$ where $C(\eta)$ is defined in (A.3) and the parameters of PESGD are chosen according to conditions (25). Then the iterates of PESGD satisfy

$$f(x_{n+1}) - f(x^*) \leq (1 + \gamma)^2\alpha_2^2(\eta)(f(x_n) - f(x^*))$$

(31)

Roughly speaking, Theorem 1 states that the effective convergence rate of PESGD matches that of PESGD in (26) up to a factor of $(1 + \gamma)^2$. Since $\lim_{\eta \to \infty}(1 + \gamma)^2 = 1$, the effective rate in (31) is equivalent to $c_{\exp}(\kappa)$ in the limit of a large condition number $\kappa$. Indeed, the numerical evidence in Figure 2 suggests that PESGD remains efficient for moderate values of the damping parameter $\eta$. In particular, for the value of $\eta_0 = 1.17$, we have $C(\eta_0) \approx 0.59$ when $\kappa \gg 1$.

## 5 Numerical Examples

We now illustrate the performance of ESGD (Algorithm 1) and PESGD (Algorithm 2) for solving Program (1) on the following test problems:

(a) Strongly convex quadratic programming: $f(x) = \frac{1}{2}x^T Ax - x^T b$, with $A \in \mathbb{R}^{d \times d}$ a random matrix drawn from the Wishart distribution. We used $d = 4800$. This problem is mildly ill-conditioned with $\kappa \approx 10^4$ and can also be solved by the conjugate gradient (CG) method.

(b) Regularised logistic regression: $f(x) = \sum_{i=1}^m \log(1 + \exp(-y_i\xi_i^T x)) + \frac{\tau}{2}\|x\|^2_2$, where $\Xi = [\xi_1, \cdots, \xi_m]^T \in \mathbb{R}^{m \times d}$ is the design matrix and $y \in \{-1, 1\}^d$. We have $f \in \mathcal{F}_{\mu,L}$ with $\mu = \tau$ and $L = \tau + \|\Xi\|^2_2/4$. As in [18], we used the Madelon UCI dataset with $d = 500$, $m = 2000$, and $\tau = 10^2$. This is a very poorly conditioned problem with $\kappa = L/\mu \approx 10^9$.

(c) Regression with (smoothed) elastic net regularisation: $f(x) = \frac{1}{2}\|Ax - b\|^2_2 + \lambda L_\tau(\|x\|_1) + \frac{\lambda}{2}\|x\|^2_2$, where $L_\tau(t)$ is the standard Huber loss function with parameter $\tau$ to smooth $|t|$ [19].

---

$^3$In the case where $g(x) = 0$ this algorithm coincides with Algorithm 1 for $\nabla f(x) = Ax$. 

7
We used $A \in \mathbb{R}^{m \times d}$ a random matrix drawn from the Gaussian distribution scaled by $1/\sqrt{d}$, $d = 3000$, $m = 900$, $\lambda = 0.2$, $\tau = 10^{-3}$, $\mu = 10^{-2}$. The objective function is in $\mathcal{F}_{\mu,L}$ with $L \approx (1 + \sqrt{m/d})^2 + \lambda/\tau + \mu$. The condition number is $\kappa \approx 10^4$.

(d) Nonlinear elliptic PDE: we consider a finite difference discretisation of the 1D integro-differential partial differential equation

$$
\frac{\partial^2 u(x)}{\partial x^2} = \int_0^1 \frac{u(s)}{(1 + |x - s|)^2} ds, \quad 0 \leq x \leq 1, \quad u(0) = 1, \quad u(1) = 0,
$$

which describes the stationary variant of a temperature profile of air near the ground [20]. Using a finite difference approximation $u(i\Delta x) \simeq U_i, i = 1, \ldots, d$ on a spatial mesh with size $\Delta x = 1/(d + 1)$, and using the trapezoidal quadrature for the integral, we obtain a problem of the form (29) where $A$ is the usual tridiagonal discrete Laplace matrix of size $d \times d$, and the entries of $\nabla g(U) \in \mathbb{R}^d$ are given by $\frac{\partial g(U)}{\partial U_i} = \frac{\Delta x}{2(1+i\Delta x)^2} + \sum_{j=1}^d \frac{\Delta x U_j^{(2)}}{(1+i\Delta x|j|)^2}$.

Problems (a)-(d) are depicted in Figure 3, where we also output the intermedia values $x^n_j$ in Algorithms 1 and 2 after every evaluation of the gradient of $f$. Figure 3 confirms that ESGD behaves like CG for quadratic problems for large values of damping $\eta$, while it outperforms AGD for moderate values of damping for more general strongly convex objective functions. Finally, in the case of nonlinear elliptic PDE (here $d = 500$), we see that the cheaper variant PESGD performs similarly to ESGD for the sets of parameters $\eta = 10$, $s = 715$ and $\eta = 1.17$, $s = 245$, while having a single evaluation of the gradient $\nabla g(x^n)$ per step, which corroborates Theorem 1.

6 Conclusion

In this paper, using ideas from numerical analysis of ODEs, we introduced a new class of optimisation algorithms for strongly convex functions based on explicit stabilised methods. These new methods, ESGD and PESGD, are as easy to implement as SD, and were shown to match the optimal convergence
rates of first order methods for certain subclasses of $F_{\mu,L}$. Our numerical experiments illustrate that this might be the case for all functions in $F_{\mu,L}$, and proving this is the subject of our current research efforts. In addition, there is a number of different interesting research avenues for this class of methods, including adjusting them to convex optimisation problems ($\mu=0$), as well as for adaptively choosing the time-step $h$ using local information to optimise their performance further. Furthermore, their adaptation to stochastic optimisation problems, where one replaces the full gradient of the function $f$ by a noisy version but cheaper version of it, is another interesting but challenging direction we aim to investigate further.

A Proof of the main results

In this section we discuss the proofs of the main results in the paper.

A.1 Proof of Proposition 1

We start our proof by noticing that the gradient flow (2) in the case of the quadratic function (19) becomes

$$\frac{dx}{dt} = -Ax + b.$$  

In addition since $A$ is positive definite and symmetric, there exists an orthogonal matrix $V$ such that

$$A = VDV^{-1}, \quad D = \text{diag}(\lambda_1, \ldots, \lambda_d), \quad \lambda_1 \leq \cdots \leq \lambda_d.$$  

If we now make the change of variables $y = V^{-1}x - D^{-1}V^{-1}b$ we obtain the following equation

$$\frac{dy}{dt} = -Dy,$$  

Hence in this coordinate system each coordinate is independent of the other, while the objective function can be written as

$$f(y) - f(y^*) = \frac{1}{2} \sum_{i=1}^{d} \lambda_i y_i^2, \quad y^* = 0.$$  

We can now write equation (A.1) in the vector form as

$$\frac{dy_i}{dt} = -\lambda_i y_i,$$  

and hence each coordinate satisfies independently the simple quadratic Program (19) and the application of Runge–Kutta method with stability function $R(z)$ gives $y_{n+1} = R(-\lambda_i h) y_n$. Hence

$$f(y_{n+1}) - f(y^*) = \sum_{i=1}^{d} \lambda_i [R(-h\lambda_i)]^{\frac{y_i^2}{2}} \leq \max_{i \in [d]} R^2(-\Delta t \lambda_i) (f(y_n) - f(y^*)), $$  

which completes the proof.

A.2 Proof of Proposition 2

Using (17) and properties of Chebyshev polynomials we have

$$\alpha_s(\eta) = \left[ \cosh \left( s \text{arcosh} \left( 1 + \frac{\eta}{s^2} \right) \right) \right]^{-1}. $$  

Using (25) and the estimate $\cosh(sx)^{-2/s} \rightarrow e^{-2x}$ for $s \rightarrow \infty$, we deduce

$$\lim_{\eta \rightarrow \infty} c_{\text{sgd}}(\kappa) = e^{-2 \text{arcosh}(1 + \frac{2}{9})} = \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^2 + O(\kappa^{-3/2}). $$  

A.3 Proof of Theorem 1

Our starting point in proving Theorem 1 would be equation (30). In particular, we will show that if we choose our parameters suitably then this scheme converges with the rate predicted by Theorem 1, and we will then show how Algorithm 2 corresponds to an implementation of this scheme.

We start the proof by noting that since $f$ in (29) is strongly convex there exists a unique minimizer $x^\star$ satisfying
\[
Ax^\star + \nabla g(x^\star) = 0
\]  
(A.2)

Thus
\[
\begin{align*}
x_{n+1} - x^\star &= R_s(-Ah)x_n - hB_s(-Ah)\nabla g(x_n) - x^\star \\
&= R_s(-Ah)(x_n - x^\star) - hB_s(-Ah)(\nabla g(x_n) - \nabla g(x^\star)) \\
&+ (R_s(-Ah) - I + hB_s(-Ah))x^\star 
\end{align*}
\]

where in the above identity we have used (A.2) multiplied on the left by $B_s(-Ah)$. Now, by definition of the matrix $B$ we have $R_s(-Ah) - I + hB_s(-Ah)A = 0$, and we obtain
\[
x_{n+1} - x^\star = R_s(-Ah)(x_n - x^\star) - hB_s(-Ah)(\nabla g(x_n) - \nabla g(x^\star))
\]

and hence
\[
\|x_{n+1} - x^\star\| \leq (\|R_s(-Ah)\| + h\|B_s(-Ah)\|\beta) \|x_n - x^\star\|.
\]

We now know from the analysis in the main text that if one is to choose parameters $h, s$ according to (25) then
\[
\|R(-Ah)\| = \alpha_s(\eta)
\]

and using the fact that $\|B_s(-Ah)\| \leq 1$, we see that
\[
\|x_{n+1} - x^\star\| \leq \left[\alpha_s(\eta) + \beta \left( \frac{\omega_0 - 1}{\omega_1 \mu} \right) \right] \|x_n - x^\star\|
\]

and hence
\[
\|x_{n+1} - x^\star\| \leq (1 + \gamma)\alpha_s(\eta)\|x_n - x^\star\|
\]

for
\[
0 < \beta < \beta^\star = \gamma \mu C(\eta)
\]

where we define
\[
C(\eta) = \frac{\omega_1 \alpha_s(\eta)}{\omega_0 - 1}.
\]

We have thus proved Theorem 1 for a numerical scheme of the form (30). It remains to prove that Algorithm 2 is a scheme equivalent to (30). Indeed, the following identity can be proved by induction on $j = 2, \ldots, s$, using $T_j(x) = 2xT_{j-1}(x) - T_{j-2}(x)$,
\[
x_n = R_{s,j}(-hA)x_n - hB_{s,j}(-hA)\nabla g(x_n)
\]

where $R_{s,j}(\xi) = T_j(\omega_0 + \omega_1 \xi)/T_j(\omega_0)$, $B_{s,j}(\xi) = (1 - R_{s,j}(\xi))/\xi$, and we obtain the result (30) by taking $j = s$.

References

[1] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 ed., 2014.

[2] D. Scieur, V. Roulet, F. R. Bach, and A. d’Aspremont, “Integration methods and optimization algorithms,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 4-9 December 2017, Long Beach, CA, USA, pp. 1109–1118, 2017.

[3] W. Su, S. Boyd, and E. J. Candès, “A differential equation for modeling nesterov’s accelerated gradient method: Theory and insights,” *Journal of Machine Learning Research*, vol. 17, no. 153, pp. 1–43, 2016.

[4] A. Wibisono, A. C. Wilson, and M. I. Jordan, “A variational perspective on accelerated methods in optimization,” *Proceedings of the National Academy of Sciences*, vol. 113, no. 47, pp. E7351–E7358, 2016.
[5] A. C. Wilson, B. Recht, and M. I. Jordan, “A Lyapunov analysis of momentum methods in optimization,” arXiv:1611.02635, 2016.

[6] M. Betancourt, M. I. Jordan, and A. C. Wilson, “On symplectic optimization,” arXiv:1802.03653, 2018.

[7] E. Hairer and G. Wanner, Solving ordinary differential equations II. Stiff and differential-algebraic problems. Berlin and Heidelberg: Springer-Verlag, 1996.

[8] N. Parikh and S. Boyd, “Proximal algorithms,” Found. Trends Optim., vol. 1, pp. 127–239, Jan. 2014.

[9] B. P. Sommeijer, L. F. Shampine, and J. G. Verwer, “RKC: an explicit solver for parabolic PDEs,” J. Comput. Appl. Math., vol. 88, pp. 316–326, 1998.

[10] A. Abdulle and A. Medovikov, “Second order Chebyshev methods based on orthogonal polynomials,” Numer Math., vol. 90, no. 1, pp. 1–18, 2001.

[11] A. Abdulle, “Fourth order Chebyshev methods with recurrence relation,” SIAM J. Sci. Comput., vol. 23, no. 6, pp. 2041–2054, 2002.

[12] A. Abdulle, “Explicit stabilized Runge-Kutta methods,” in Encyclopedia of Applied and Computational Mathematics, (Berlin), pp. 460–468, Björn Engquist (Ed.), Springer-Verlag, 2015.

[13] A. Abdulle and S. Cirilli, “S-ROCK: Chebyshev methods for stiff stochastic differential equations,” SIAM J. Sci. Comput., vol. 30, no. 2, pp. 997–1014, 2008.

[14] A. Abdulle, I. Almuslimani, and G. Vilmart, “Optimal explicit stabilized integrator of weak order one for stiff and ergodic stochastic differential equations,” To appear in SIAM/ASA Journal on Uncertainty Quantification (JUQ), 2018.

[15] P. J. van der Houwen and B. P. Sommeijer, “On the internal stability of explicit, m-stage Runge-Kutta methods for large m-values,” Z. Angew. Math. Mech., vol. 60, no. 10, pp. 479–485, 1980.

[16] L. Lessard, B. Recht, and A. Packard, “Analysis and design of optimization algorithms via integral quadratic constraints,” SIAM Journal on Optimization, vol. 26, no. 1, pp. 57–95, 2016.

[17] C. J. Zbinden, “Partitioned Runge-Kutta-Chebyshev methods for diffusion-advection-reaction problems,” SIAM J. Sci. Comput., vol. 33, no. 4, pp. 1707–1725, 2011.

[18] D. Scieur, A. d’Aspremont, and F. Bach, “Regularized nonlinear acceleration,” in NIPS ’16 Proceedings of the 30th International Conference on Neural Information Processing Systems, pp. 712–720, 2016.

[19] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” Journal of the Royal Statistical Society: Series B (Statistical Methodology), vol. 67, no. 2, pp. 301–320, 2005.

[20] A. S. Vasudeva Murthy and J. G. Verwer, “Solving parabolic integro-differential equations by an explicit integration method,” J. Comput. Appl. Math., vol. 39, no. 1, pp. 121–132, 1992.