L-BFGS and Delayed Dynamical Systems Approach for Unconstrained Optimization

Xiaohui XIE

Abstract

The dynamical (or ode) systems approach for optimization problems has existed for two decades. The main feature of this approach is that a continuous path starting from the initial point can be generated and eventually the path will converge to the solution. This approach is quite different from conventional optimization methods where a sequence of points, or a discrete path, is generated. An advantage of the dynamical systems approach is that it is more suitable for large scale problems. Common examples of this approach are ode's based on the steepest descent direction or the Newton's direction. In this research we apply the L-BFGS scheme to the ode model, hopefully to improve on the rate of convergence over the steepest descent direction, but not to suffer from the large amount of computational work in the Newton's direction.

1 Problem background and introduction

This paper studies computational methods for a local or the global minimizer of an unconstrained optimization problem. Optimization problems are classified into:
(a) Unconstrained Problem:

$$\min_{x\in R^n} f(x) \qquad f: R^n \to R^1 \qquad (UP)$$

(b) Equality Constrained Problem:

$$\begin{aligned} &\min_{x\in R^n} f(x) \\ &\quad h(x) \quad = 0 \qquad h: R^n \to R^p \end{aligned}$$

(c) Inequality Constrained Problem:

$$\begin{aligned} &\min_{x\in R^n} f(x) \\ &\quad g(x) \quad \le 0 \qquad h: R^n \to R^p \end{aligned}$$

(d) General Constrained Problem:

$$\begin{aligned} &\min_{x\in R^n} f(x) \\ &\quad g(x) \quad \le 0 \\ &\quad h(x) \quad = 0 \end{aligned}$$

The motivation of unconstrained methods is to generate a sequence of points $\{x_k\}$ (x_0 given) such that (1) $f(x_k) > f(x_{k+1})$;(2) $\{x_k\}$ is convergent, and (3) the limit point of the sequence is a stationary point of (UP). Different methods advance from x_k to x_{k+1} differently. Well-known methods include the steepest descent method, Newton's method and quasi-Newton method. A common theme behind all these methods is to find a direction $p \in R^n$ so that there exists an $\bar{\epsilon} > 0$ such that

$$f(x + \epsilon p) < f(x) \qquad \forall \epsilon \in (0, \bar{\epsilon})$$

This direction is called a descent direction of $f(x)$ at x. Once we have found a descent direction, we may go along this direction to approach one more step toward the optimum solution.

The following paragraphs summarize the advantages and disadvantages of these methods.

1.1 Steepest descent method

Using directional derivative in multivariable calculus, it is clear that for (UP), p is a descent direction at $x \Leftrightarrow \nabla f(x)^T p < 0$. Hence $p = -\nabla f(x)$, or equivalently, $p = -\nabla f(x)/\|\nabla f(x)\|_2$ is obviously a descent direction for $f(x)$. This direction is called the steepest descent direction.

Method of Steepest Descent: At each iteration k: find the lowest point of f in the direction $-\nabla f(x_k)$ from x_k, i.e., find λ_k that solves

$$\min_{\lambda > 0} f(x_k - \lambda \nabla f(x_k))$$

Then $x_{k+1} = x_k - \lambda \nabla f(x_k)$.

Unfortunately, the steepest descent method converges only linearly, and sometimes very slowly linearly. In fact, if $\{x_k\}$, which is generated by the steepest descent method, converges to a local minimizer x^* where $\nabla^2 f(x^*)$ is positive definite (p.d.), and λ_{max} and λ_{min} are the largest and smallest eigenvalues of $\nabla^2 f(x^*)$, then one can show that $\{x_k\}$ satisfies

$$\lim_{k \to \infty} sup \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} \le c, \quad c = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}},$$

in a particular weighted l_2 norm, and the bound on c is tight for some starting x_0. This property indicates that the steepest descent method is q-linearly convergent. When λ_{max} and λ_{min} are far apart, then c is close to 1, and the convergence will be slow.

1.2 Newton's method

At point x_k, if $\nabla^2 f(x_k)$ is p.d., the function $f(x)$ can be approximated by a quadratic function based on the Taylor expansion:

$$\begin{aligned} f(x) \cong\ & f(x_k) + \nabla f(x_k)^T (x - x_k) \\ & + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k) \end{aligned} \qquad (1)$$

Then the minimizer of (1) is given by

$$\begin{aligned} \nabla f(x) &= 0 \\ &\Rightarrow \nabla f(x_k) + \nabla^2 f(x_k)(x - x_k) = 0 \\ &\Rightarrow x = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k), \end{aligned}$$

where $-[\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$, is called the Newton's direction. Then we define

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k),$$

and the resulting method of computing x_k is called the Newton's method.

Newton's method
Given x_0, compute

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k), \qquad k \leftarrow k + 1.$$

A key requirement for Newton's method is the p.d. of $\nabla^2 f(x_k)$. Descent directions guarantee that $f(x)$ can be further reduced and therefore they form the basis of some global methods. However, in some real applications, if the starting point is far away from the optimal solution, or the Hessian is not positive definite, Newton's direction is not adopted. Although Newton's method converges very fast ($\{x_k\}$ converges to x^* q-quadratically), the Hessian matrix is difficult to compute. So we would like to find more feasible methods with (A) no second-order information, i.e., no Hessian; and (B) fast convergence. A rule of thumb is that first-order information normally gives slow (linear) convergence, while second-order information normally gives fast (quadratic) convergence. Let us discuss several practical considerations. In general, the convergence is quadratic: the error is essentially squared at each step (that is, the number of accurate digits doubles in each step). There are some caveats, however. Firstly, Newton's method requires that the derivative be calculated directly. (If the derivative is approximated by the slope of a line through two points on the function, the secant method results; this can be more efficient depending on how one measures computational effort.) Secondly, if the initial value is too far from the true zero, Newton's method can fail to converge. Because of this, most practical implementations of Newton's method put an upper limit on the number of iterations and perhaps on the size of the iterates. Thirdly, if the root being sought has multiplicity greater than one, the convergence rate is merely linear (errors reduced by a constant factor at each step) unless special procedures are taken. Finding the inverse of the Hessian is an expensive operation. Therefore the descent direction $-[\nabla^2 f(x)]^{-1} \nabla f(x)$ is often solved approximately (but to great accuracy) using methods such as the conjugate gradient method. There also exist various quasi-Newton methods, where an approximation for the Hessian is used instead.

Table 1 compares the advantages and disadvantages between the steepest descent method and the Newton's method.

Table 1: **Comparison of the methods**

	Advantage	Disadvantage
Steepest descent method	Simple and inexpensive, guarantees descent	Slow convergence
Newton's method	Very fast convergence if applicable	Expensive, second-order information matrix inversion

1.3 Quasi-Newton method—BFGS

Instead of using the Hessian matrix, the quasi-Newton methods approximate it.

Quasi-Newton methods are based on Newton's method to find the stationary point of $f(x)$, where the gradient $\nabla f(x)$ is 0. In Quasi-Newton methods the Hessian matrices of second derivatives of $f(x)$ do not need to be computed. The Hessian is updated by analyzing successive gradient vectors instead. Quasi-Newton methods are a generalization of the secant method to find the root of the first derivative for multidimensional problems. In multi-dimensions the secant equation is under-determined, and quasi-Newton methods differ in how they constrain the solution, typically by adding a simple low-rank update to the current estimate of the Hessian.

In quasi-Newton methods, the inverse of the Hessian matrix is approximated in each iteration by a p.d. matrix, say H_k, where k is the iteration index. Thus, the k th iteration has the following basic structure:
(a) set $p_k = -H_k g_k, \quad (g_k = \nabla f(x_k))$,
(b) line search along p_k giving $x_{k+1} = x_k + \lambda_k p_k$,
(c) update H_k giving H_{k+1}.

The initial matrix H_0 can be any positive definite symmetric matrix, although in the absence of any better estimate, the choice $H_0 = I$ often suffices. Potential advantages of the method are:
(1) only first-order information is required;
(2) H_k being symmetric and p.d. implies the descent property; and
(3) $O(n^2)$ multiplications per iteration.

The most important quasi-Newton formula was suggested by Broyden, Fletcher, Goldfarb, and Shanno independently in 1970, and is subsequently known as the BFGS formula. It is used to solve an unconstrained nonlinear optimization problem.

$$H_{k+1}^{BFGS} = H_k + \left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k}\right) \frac{s_k s_k^T}{s_k^T y_k} - \left(\frac{s_k y_k^T H_k + H_k y_k s_k^T}{s_k^T y_k}\right) \quad (2)$$

where $s_k = x_{k+1} - x_k$,
$y_k = \nabla f(x_{k+1}) - \nabla f(x_k) = g_{k+1} - g_k$.

We have the following theorem.

Theorem 1.1 *If H_k^{BFGS} is a p.d. matrix, and $s_k^T y_k > 0$, then H_{k+1}^{BFGS} in (2) is also positive definite.*

Proof: For any $z \neq 0$, it is sufficient to prove that

$$z^T \left[H_k + \left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k}\right) \frac{s_k s_k^T}{s_k^T y_k} - \left(\frac{s_k y_k^T H_k + H_k y_k s_k^T}{s_k^T y_k}\right)\right] z > 0,$$

In the rest of the proof, the subscript k will be omitted. Since H is p.d., we can write $H = LL^T$, and let
$a = L^T z$ and $b = L^T y$, then

$$
\begin{aligned}
z^T H_{k+1} z &= z^T \left[H_k + \left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k} \right) \frac{s_k s_k^T}{s_k^T y_k} \right. \\
&\quad \left. - \left(\frac{s_k y_k^T H_k + H_k y_k s_k^T}{s_k^T y_k} \right) \right] z \\
&= z^T \left[H + \left(1 + \frac{y^T H y}{s^T y} \right) \frac{s s^T}{s^T y} - \left(\frac{s y^T H + H y s^T}{s^T y} \right) \right] z \\
&= a^T a + \frac{z^T b^T b s s^T z}{(s^T y)^2} + \frac{(z^T s)^2}{s^T y} - \frac{z^T s b^T a}{s^T y} - \frac{a^T b s^T z}{s^T y} \\
&= \left(a - \frac{z^T s b}{s^T y} \right)^T \left(a - \frac{z^T s b}{s^T y} \right) + \frac{(z^T s)^2}{s^T y} \\
&= \left\| a - \frac{z^T s b}{s^T y} \right\|^2 + \frac{(z^T s)^2}{s^T y} \\
&\geq 0
\end{aligned}
$$

If the norm above equals zero, i.e.,

$$\left\| a - \frac{z^T s b}{s^T y} \right\| = 0,$$

then we have

$$a = \frac{z^T s b}{s^T y}, \text{ or } L^T z = \frac{z^T s L^T y}{s^T y},$$

which means that $y \propto z$.

However, since $s^T y > 0$,

$$\frac{(z^T s)^2}{s^T y} > 0,$$

as $y \propto z$. Thus the theorem is proved. □

1.4 Limited-Memory Quasi-Newton Methods—L-BFGS

Limited-memory quasi-Newton methods are useful for solving large problems whose Hessian matrices cannot be computed at a reasonable cost or are not sparse. These methods maintain simple and compact approximations of Hessian matrices: instead of storing fully dense $n \times n$ approximations, they save only a few vectors of length n that represent the approximations implicitly. Despite these modest storage requirements, they often yield an acceptable rate of convergence. Various limited-memory methods have been proposed; we focus mainly on an algorithm known as L-BFGS, which, as its name suggests, is based on the BFGS updating formula. The main idea of this method is to use curvature information from only the most recent iterations to construct the Hessian approximation. Curvature information from earlier iterations, which is less likely to be relevant to the actual behavior of the Hessian at the current iteration, is discarded in the interest of saving storage.

As we have discussed in section 1.3, each step of the BFGS method has the form

$$x_{k+1} = x_k - \alpha_k H_k \nabla f_k,$$

where α_k is the step length and H_k is updated at every iteration by means of the formula

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \tag{3}$$

where

$$\rho_k = \frac{1}{y_k^T s_k}, \qquad V_k = I - \rho_k y_k s_k^T, \tag{4}$$

and

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla f_{k+1} - \nabla f_k. \tag{5}$$

Since the inverse Hessian approximation H_k will generally be dense, the cost of storing and manipulating it is prohibitive when the number of variables is large. To circumvent this problem, we store a modified version of H_k implicitly, by storing a certain number (say, m) of the vector pairs $\{s_i, y_i\}$ used in the formulas (3)-(5). The product $H_k \nabla f_k$ can be obtained by performing a sequence of inner products and vector summations involving ∇f_k and the pairs $\{s_i, y_i\}$. After the new iterate is computed, the oldest vector pair in the set of pairs $\{s_i, y_i\}$ is replaced by the new pair $\{s_k, y_k\}$ obtained from the current step (5).

We now describe the updating process in a little more detail. At iteration k, the current iterate is x_k and the set of vector pairs is given by $\{s_i, y_i\}$ for $i = k - m, \ldots, k - 1$. We first choose some initial

Hessian approximation H_0 (in contrast to the standard BFGS iteration, this initial approximation is allowed to vary from iteration to iteration) and find by repeated application of the formula (3) that the L-BFGS approximation H_{k+1} satisfies the following formula [12]:

In general, we have for $k+1 \leq m$ the usual BFGS updated

$$\begin{aligned} H_{k+1} &= V_k^T V_{k-1}^T \cdots V_0^T H_0 V_0 \cdots V_{k-1} V_k \\ & +V_k^T \cdots V_1^T \rho_0 s_0 s_0^T V_1 \cdots V_k \\ & \vdots \\ & +V_k^T V_{k-1}^T \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} v_k \\ & +V_k \rho_{k-1} s_{k-1} s_{k-1}^T V_k \\ & +\rho_k s_k s_k^T. \end{aligned} \tag{6}$$

For $k+1 > m$ we have the update

$$\begin{aligned} H_{k+1} &= V_k^T V_{k-1}^T \cdots V_{k-m+1}^T H_0 V_{k-m+1} \cdots V_{k-1} V_k \\ & +V_k^T \cdots V_{k-m+2}^T \rho_{k-m+1} s_{k-m+1} s_{k-m+1}^T \\ & \quad \cdot V_{k-m+2} \cdots V_k \\ & \vdots \\ & +V_k^T V_{k-1}^T \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} V_k \\ & +V_k \rho_{k-1} s_{k-1} s_{k-1}^T V_k \\ & +\rho_k s_k s_k^T. \end{aligned} \tag{7}$$

A method for choosing H_0 that has proved effective in practice is to set $H_0 = \gamma_k I$, where

$$\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}.$$

The strategy of keeping the m most recent correction pairs $\{s_i, y_i\}$ works well in practice; indeed no other strategy has yet proved to be consistently better. However, the main weakness of the L-BFGS method is that it converges slowly on ill-conditioned problems — specifically, on problems where the Hessian matrix contains a wide distribution of eigenvalues. On certain applications, the nonlinear conjugate methods are competitive with limited-memory quasi-Newton methods.

Algorithm (L-BFGS)

```
Choose starting point $x_0$, integer $m > 0$;
$k \leftarrow 0$;
**repeat**
  Choose $H_0$
  Compute $p_k \leftarrow -H_k \nabla f_k$
  Compute $x_{k+1} \leftarrow x_k + \alpha_k p_k$ where $\alpha_k$ is chosen
      to satisfy the Wolfe conditions;
  **if** $k > m$
      Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from
        storage;
  Compute and save
      $s_k \leftarrow x_{k+1} - x_k, y_k = \nabla f_{k+1} - \nabla f_k$;
  $k \leftarrow k + 1$;
**until convergence.**
```

2 Analysis for dynamical systems with time delay

2.1 Introduction of dynamical systems

For the easiness of reading, the (UP) problem is reproduced here:

$$\min_{x \in R^n} f(x) \qquad f : R^n \to R^1 \tag{8}$$

It is very important that the optimization problem (8) itself is posted in the continuous form, i.e., x can be changed continuously. In the literature, the necessary and sufficient conditions of a local optimum are also presented in the continuous form. Furthermore, almost all the theoretical study for problem (8) is in the continuous form. However, it is very interesting to say that when it comes down to the numerical solution of (8), most of the conventional methods, such as the gradient/ steepest descent method, Newton's method and quasi-Newton's method, are all addressed in the discrete form. This interesting situation is mainly due to the fact that the computer's computation can be only done discretely. However, is it possible to study both the optimization problem and the solution methods in its original form, i.e., continuous form? In this sense, we may use the dynamical system approach or neural network approach to solve the original optimization problem.

• Dynamical system approach. The essence of this approach is to convert problem (8) into a dynamical system or an ordinary differential equation (ode) so that the solution of problem (8) corresponds to a stable equilibrium point of this dynamical system.

• Neural network approach. The mathematical representation of neural network is an ordinary differential equation which is asymptotically stable at any isolated solution point. A companion of this neural network is an energy function which is a Lyapunov function. And as time evolves, the solution of the ode will converge to the optimum, and in this whole process, the energy function will decrease monotonically in time.

The following discussion reviews the research results in the dynamical system approach, and identifies the merits of this approach.

Consider the following simple dynamical system or ordinary differential equation

$$\frac{dx(t)}{dt} = p(x). \tag{9}$$

We first state some classical result on the existence and uniqueness of the solution, and some stability definitions for the dynamical system (9) [21,24].

Theorem 2.1 [24] *Assume that $p(x)$ is a continuous function from R^n to R^n. Then for arbitrary $t_0 \geq 0$ and $x_0 \in R^n$ there exists a local solution $x(t)$ satisfying $x(t_0) = x_0$, $t \in [t_0, \tau)$ to (9) for some $\tau > t_0$. If furthermore $p(x)$ is locally Lipschitz continuous at x_0, then the solution is unique, and if $p(x)$ is Lipschitz continuous in R^n then τ can be extended to ∞.*

Definition 2.2 *(Equilibrium point) A point $x^* \in R^n$ is called an equilibrium point of (9) if $p(x^*) = 0$.*

Definition 2.3 *(Stablility in the sense of Lyapunov) Let $x(t)$ be the solution of (9). An isolated equilibrium point x^* is Lyapunov stable if for any $x_0 = x(t_0)$ and any scalar $\epsilon > 0$, there exists a $\delta > 0$ such that if $\|x(t_0) - x^*\| < \delta$, then $\|x(t) - x^*\| < \epsilon$ for $t \geq t_0$.*

Definition 2.4 *(Convergence) Let $x(t)$ be the solution of (9). An isolated equilibrium point x^* is convergent if there exists a $\delta > 0$ such that if $\|x(t_0) - x^*\| < \delta$, $x(t) \to x^*$ as $t \to \infty$.*

A dynamical system or ode (9) arising from an optimization problem needs to have $p(x)$ being a descent direction for the objective function $f(x)$. Some well-known versions are:

Dynamical system based on the steepest descent direction:

$$\frac{dx(t)}{dt} = -\nabla f(x(t))$$

Dynamical system based on the Newton direction:

$$\frac{dx(t)}{dt} = -[\nabla^2 f(x(t))]^{-1} \nabla f(x(t)) \tag{10}$$

As in the discrete optimization methods in the previous chapter, the steepest descent direction has a slow convergence rate - meaning that it takes a very "large" value of t to approach the equilibrium point. The Newton direction has a much faster convergence rate, but the amount of work in evaluating the Jacobian is much greater.

Some other dynamical systems in the literature are:

$$\frac{dx(t)}{dt} = s(t) \cdot p(x(t)), \tag{11}$$

$$a(t) \cdot \frac{d^2x(t)}{dt^2} + b(t) \cdot B(x(t)) \cdot \frac{dx(t)}{dt} = p\left(x(t)\right), \tag{12}$$

where $p(x)$is a descent direction for $f(x)$, $B(x) \in R^{n \times n}$ is a positive definite matrix, $a(t)$ and $b(t)$ are scalar functions in t, and $s(t)$ is a positive scalar function in t and bounded above.

A major advantage of the dynamical systems approach is that very large problems can be solved [13]. No matter whether we use any of (9)-(12), existing ode methods are quite mature to tackle these problems. Solving systems with tens or hundreds of thousands of unknowns poses no problem to ode solvers. The problem size handled can be much larger than traditional methods described in Chapter 1, which are of order of magnitude in the thousands. The research in the ode approach is to find a "good" $p(x)$ in (9) that balances the convergence rate and the amount of work.

The dynamical systems approach normally consists of the following three steps:

(a) to establish an ode system;

(b) to study the convergence of the solution $x(t)$ of the ode as $t \to \infty$; and

(c) to solve the ode system numerically.

The convergence study of $x(t)$ as $t \to \infty$ and the stability of the corresponding dynamical system have mostly been addressed on a case by case base. No standard theory and/or methodology are given. This phenomenon certainly limits the systematic study of the dynamical system approach and its application potential as well. Two papers are worth mentioning, one by Tanabe [18] which used the stability theory of the dynamical system to study the ode system, and the other one by Yamashita [24] which employed Lyapunov's direct method to study the ode system.

Even though the solutions of ode systems are continuous, yet the actual computation has to be done discretely. In all the dynamical systems (10)-(12), the numerical solutions were mainly solved by either discrete optimization methods or finite difference methods.

In summary, the main attractiveness of this approach is its simplicity and its originality in pursuing the continuous form. Furthermore, there is not any restriction on the form of the objective function $f(x)$ in (8).

2.2 Delayed dynamical systems approach

As stated above, the steepest descent direction and the Newton direction of the dynamical systems approach both have their weakness. The main idea in this paper is to apply the theme of the L-BFGS algorithm in Chapter 1 to the dynamical systems approach, making it a bridge between the steepest descent direction and the Newton direction. The resulting dynamical system is a delayed ode, thus we call it the delayed dynamical systems approach.

The delayed dynamical systems approach solves the delayed ode:

$$\frac{dx(t)}{dt} = -H(x(t), x(t-\tau_1(t), \ldots, x(t-\tau_m(t)))) \cdot \nabla f(x(t)), \tag{13}$$

where H and $t-\tau_1(t), \ldots, t-\tau_m(t)$ are to be defined below.

As the delayed ode (13) is numerically solved, we compute approximations $x_0, x_1, \ldots, x_k, x_{k+1}, \ldots$ to $x(t)$ at time points $t_0, t_1, \ldots, t_k, t_{k+1}, \ldots$. We define $H = H_k$ to be a different function in the interval $(t_{k-1}, t_k]$ iteratively by:

Given $t_0, x(t_0) = x_0$, and an initial H_0, for $t_0 \le t$, we define

$$\begin{aligned} H(x(t), x(t_0)) &:= H_1(x(t), x(t_0)) \\ &:= V_0(t)^T H_0 V_0(t) + \rho_0(t) s_0(t) s_0(t)^T, \end{aligned}$$

where

$$s_0(t) = x(t) - x_0,$$

$$y_0(t) = \nabla f(x(t)) - \nabla f(x_0),$$

$$\rho_0(t) = 1/y_0(t)^T s_0(t),$$

$$V_0(t) = I - \rho_0(t) y_0(t) s_0(t)^T,$$

in the R.H.S. of (13) and determine a stepsize $h_1 = (t_1 - t_0)$ to compute, using some numerical ode method, an approximation x_1 to $x(t_1)$ at t_1. Then for $t_1 \le t$, we define

$$\begin{aligned} &H(x(t), x(t_1), x(t_0)) \\ &\quad := H_2(x(t), x(t_1), x(t_0)) \\ &\quad := V_1(t)^T V_0(t_1)^T H_0 V_0(t_1) V_1(t) \\ &\qquad + V_1(t)^T \rho_0(t_1) s_0(t_1) s_0(t_1)^T V_1(t) \\ &\qquad + \rho_1(t) s_1(t) s_1(t)^T \end{aligned}$$

where

$$s_1(t) = x(t) - x(t_1),$$

$$y_1(t) = \nabla f(x(t)) - \nabla f(x(t_1)),$$

$$\rho_1(t) = 1/y_1(t)^T s_1(t),$$

$$V_1(t) = I - \rho_1(t) y_1(t) s_1(t)^T,$$

in the R.H.S. of (13) and determine a stepsize $h_2 = (t_2 - t_1)$ to compute, using some numerical ode method, an approximation x_2 to $x(t_2)$ at t_2. Of course, computationally x_1 is used instead of $x(t_1)$.

This process is repeated until we have accepted x_{m-1} at t_{m-1}. Then for $t_{m-1} \le t$, we use

$$\begin{aligned}
&H(x(t), x(t_{m-1}), \ldots, x(t_1), x(t_0)) \\
&\quad := H_m(x(t), x(t_{m-1}), \ldots, x(t_1), x(t_0)) \\
&\quad := V_{m-1}(t)^T V_{m-2}(t_{m-1})^T \ldots V_1(t_2)^T V_0(t_1)^T \\
&\qquad \cdot H_0 V_0(t_1) V_1(t_2) \ldots V_{m-2}(t_{m-1}) V_{m-1}(t) \\
&\qquad + V_{m-1}(t)^T V_{m-2}(t_{m-1})^T \ldots V_1(t_2)^T \\
&\qquad \cdot \rho_0(t_1) s_0(t_1) s_0(t_1)^T V_1(t_2) \ldots V_{m-2}(t_{m-1}) \\
&\qquad \cdot V_{m-1}(t) \\
&\qquad + \ldots \\
&\qquad + V_{m-1}(t)^T \rho_{m-2}(t_{m-1}) s_{m-2}(t_{m-1}) \\
&\qquad \cdot s_{m-2}(t_{m-1})^T V_{m-1}(t) \\
&\qquad + \rho_{m-1}(t) s_{m-1}(t) s_{m-1}(t)^T \qquad (13A)
\end{aligned}$$

where

$$\begin{aligned}
s_{m-1}(t) &= x(t) - x(t_{m-1}), \\
y_{m-1}(t) &= \nabla f(x(t)) - \nabla f(x(t_{m-1})), \\
\rho_{m-1}(t) &= 1/y_{m-1}(t)^T s_{m-1}(t), \\
V_{m-1}(t) &= I - \rho_{m-1}(t) y_{m-1}(t) s_{m-1}(t)^T.
\end{aligned}$$

to compute x_m at t_m. Beyond this point we save only m previous values of x. The definition of H is now, for $m \le k$,
for $t_k \le t$,

$$\begin{aligned}
&H(x(t), x(t_k), \ldots, x(t_{k-m+2}), x(t_{k-m+1})) \\
&\quad := H_{k+1}(x(t), x(t_k), \ldots, x(t_{k-m+2}), x(t_{k-m+1})) \\
&\quad := V_k(t)^T V_{k-1}(t_k)^T \ldots V_{k-m+2}(t_{k-m+3})^T \\
&\qquad \cdot V_{k-m+1}(t_{k-m+2})^T H_0 V_{k-m+1}(t_{k-m+2}) \\
&\qquad \cdot V_{k-m+2}(t_{k-m+3}) \ldots V_{k-1}(t_k) V_k(t) \\
&\qquad + V_k(t)^T V_{k-1}(t_k)^T \ldots V_{k-m+2}(t_{k-m+3})^T \\
&\qquad \cdot \rho_{k-m+1}(t_{k-m+2}) s_{k-m+1}(t_{k-m+2}) \\
&\qquad \cdot s_{k-m+1}(t_{k-m+2})^T V_{k-m+2}(t_{k-m+3}) \ldots \\
&\qquad \cdot V_{k-1}(t_k) V_k(t) \\
&\qquad + \ldots \\
&\qquad + V_k(t)^T \rho_{k-1}(t_k) s_{k-1}(t_k) \\
&\qquad \cdot s_{k-1}(t_k)^T V_k(t) \\
&\qquad + \rho_k(t) s_k(t) s_k(t)^T \qquad (13B)
\end{aligned}$$

where

$$\begin{aligned}
s_k(t) &= x(t) - x(t_k), \\
y_k(t) &= \nabla f(x(t)) - \nabla f(x(t_k)), \\
\rho_k(t) &= 1/y_k(t)^T s_k(t), \\
V_k(t) &= I - \rho_k(t) y_k(t) s_k(t)^T.
\end{aligned}$$

It is obvious that the delayed ode (13) is a continuous version of the L-BFGS scheme. The $H = H_k$ in (13) attempts to approximate the inverse of the Jacobian in the Newton method. It is worth mentioning that the matrix H_k is never computed explicitly. We only need to compute the R.H.S. of (13), i.e., the product of H_k and a vector.

2.3 Uniqueness property of dynamical systems

The positive definite property of the dynamical system (13) is proved in Appendix I, hence we conclude that the solution exists. The next step is to show the Lipschitz continuity of this system, which implies that the solution is also unique.

2.3.1 Definition of Lipschitz continuity

Let $F : R^n \to R^m$ be a function, we say that F is **Lipschitz continuous** with **Lipschitz constant** L if there is a nonnegative constant L such that

$$\|F(x_1) - F(x_2)\| \le L\|x_1 - x_2\|$$

for all x_1 and x_2 in R^m.

2.3.2 Lipschitz continuity of method (13)

By Theorem 2.1, the solution of our new unconstrained optimization method (13) is unique if the right-handed-side of (13) is Lipschitz continuous. We firstly rewrite (13) as

$$\frac{dx(t)}{dt} = -H(x(t), x(t-\tau))\nabla f(x(t)) \; .$$

Let $u = x(t)$ and $w = x(t-\tau)$, then our aim is to prove that $-H(u, w)\nabla f(u)$ is Lipschitz continuous with respect to u and w. In other words, we want to prove that

$$\|H(u,w)\nabla f(u) - H(\bar{u},w)\nabla f(\bar{u})\| \leq L_1\|u-\bar{u}\|, \quad (14)$$

$$\|H(u,w)\nabla f(u) - H(u,\bar{w})\nabla f(u)\| \leq L_2\|w-\bar{w}\|. \quad (15)$$

This problem is a difficult one and the following theorem may give some hint to our goal:

Theorem 2.5 *(Kurdyka, subanalytic, semi-algebraic) Let F: $X \subset R^n \rightarrow R$ be a definable C^1-function such that*

$$\left|\partial F/\partial x_i\right| < M$$

for some M and each i.
Then there exist a finite partition of X and $C > 0$ such that on each piece, the restriction of F to this piece is C-Lipschitz.
Moreover, this finite partition only depends on X and not on F. (And C only depends on M and n.)

It is obvious that $H(u,w)\nabla f(u)$ is a column vector. In order to transform a vector into a scalar we can use $e_i^T H(u,w)\nabla f(u)$ and problem (14) and (15) can be converted to

$$\left|\frac{\partial}{\partial u}[e_i^T H(u,w)\nabla f(u)]\right| < M_1,$$

$$\left|\frac{\partial}{\partial w}[e_i^T H(u,w)\nabla f(u)]\right| < M_2$$

Equation (13B) shows that $H(u,w)$ depends in a complicated way on

$$y(u,w) = \nabla f(u) - \nabla f(w) \ ,$$

$$s(u,w) = u - w \ .$$

The resulting partial derivatives $\frac{\partial}{\partial w}[e_i^T H\nabla f(u)]$ contain so many fractions whose denominators all have terms $y^T s$ that the bound must be controlled properly because the denominators tend to 0. The main problem is therefore to make the numerators and denominators have the same order so that they canceled out each other.

The following lemma seems to play a critical role in this cancellation:

Lemma 2.6 *Let $F : R^n \rightarrow R^m$ be continuously differentiable in the open convex set $D \subset R^n$, $x \in D$, and let $J = \frac{\partial F}{\partial x}$ be Lipschitz continuous at x in the neighborhood D, using a vector norm and the induced matrix operator norm and the Liptschitz constant γ. Then, for any $x + p \in D$,*

$$\|F(x+p) - F(x) - J(x)p\| \leq \frac{\gamma}{2}\|p\|^2.$$

If we define $x = w, p = u - w = s, F(x) = \nabla f(x)$, then the lemma converts to

$$\|\nabla f(u) - \nabla f(w) - \nabla^2 f(w)(u-w)\| \leq \frac{\gamma}{2}\|u-w\|$$

Substituting y and s into the above inequality yields

$$\|y - \nabla^2 f(w)s\| \leq \frac{\gamma}{2}\|s\|.$$

3 Numerical testing

Based on the analysis in the previous sections, we hope to conclude that the continuous-time Limited Memory BFGS method has a better performance than other traditional methods. In this section we will show the computational results of several examples.

3.1 The test problems

We have tested method (13) on the following unconstrained optimization problems:
Problem 1: Extended Rosenbrock function

$$f(x) = \sum_{i=1}^{n}[100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2],$$
$$[x_0]_{2i-1} = -1.2, [x_0]_{2i} = 1, [x^*]_i = 1, f(x^*) = 0.$$

Problem 2: Penalty function I

$$f(x) = \sum_{i=1}^{n} 10^{-5}(x_i - 1)^2 + [(\sum_{i=1}^{n} x_i^2) - \frac{1}{4}]^2,$$
$$[x_0]_i = i.$$

Problem 3: Variable dimensioned function

$$f(x) = \sum_{i=1}^{n}(x_i - 1)^2 + [\sum_{i=1}^{n} i(x_i - 1)]^2,$$

$$+[\sum_{i=1}^{n} i(x_i - 1)]^4,$$

$$[x_0]_i = 1 - i/n, [x^*]_i = 1, f(x^*) = 0.$$

Problem 4: Linear function-rank 1

$$f(x) = \sum_{i=1}^{m}[i(\sum_{j=1}^{n} jx_j) - 1]^2, (m \geq n)$$

$$[x_0]_i = \frac{1}{i}, f(x^*) = \frac{m(m-1)}{2(2m+1)} \text{ at any point}$$

$$\text{where } \sum_{j=1}^{n} jx_j = \frac{3}{2m+1}.$$

3.2 Comparison between continuous-time L-BFGS and continuous-time steepest descent

Our main platform of numerical computation is Matlab. The Matlab library contains nonstiff ode solvers ode113, ode23, and ode45, and stiff ode solvers ode15s, ode23s, and ode23tb. There is also a nonstiff delayed differential solver ddesd. Nonstiff solvers are efficient for ode problems without a wide-spread spectrum of eigenvalues, whereas stiff solvers are good for problems with both large and small eigenvalues.

The difficulty with solving (13) is the time delay. In order to get familiar with the ode solvers, and eventually to take into account the delayed equation, we first test ode (and not dde) solvers on the above test problems.

In the first phase we start with solving a modified form of Problem 1:

Modified Extended Rosenbrock function:

$$f(x) = \sum_{i=1}^{n}[100(x_{2i} - x_{2i-1})^2 + (1 - x_{2i-1})^2],$$

$$[x_0]_{2i-1} = -1.2, [x_0]_{2i} = 1, [x^*]_i = 1, f(x^*) = 0.$$

The difference between the original Rosenbrock and the modified one is the square of x_{2i-1} in the first part of the right-handed-side.

We use the nonstiff ode solver ode113 on this problem with dimension $n = 100$ and tolerance $= 10^{-4}$. The result is given in the following table, where t denotes the iteration time, *value* denotes the computed optimal solution value, and *step* denotes the number of iterations.

Table 2: **Modified Rosenbrock Problem using ode113**

	t	value	step
L-BFGS	2	0	497
Steepest descent	23.2813	0.0006	53557

We can see that the continuous-time L-BFGS is obviously faster than the continuous-time steepest descent method in the value of t and number of integration steps.

Based on the performance in phase 1, we then move on to the second phase, where the delayed equation solver ddesd is used. This time we focus on all the test problems mentioned above and also use the original Extended Rosenbrock. However, the numerical results are not so good as we have expected – the continuous-time steepest descent method shows better performance than the continuous-time L-BFGS. After an analysis on the problems, it is found that many of them have a wide spectrum of eigenvalues. In other words, the problems are stiff.

In the third phase, the Matlab stiff ode solver ode15s is being used on the above four problems. Matlab does not have a stiff solver, and hence we are unable to take into account the effect of time delay. In the following tables, P denotes the problem number, N the dimension of variables. Table 4 to Table 6 give the performance on function value when the solution reach to the optimum value.

Table 3: **Comparison of the two methods for $m = 2$ on function value**

P	*N*	*Steepest descent*	*Limited-Memory BFGS*
1	10^3	0	5.8852×10^3
2	10^3	1.2030×10^5	9.7257×10^{-3}
3	10^3	5.2945×10^{-4}	42.307
4	10^3	6.0317	1.3153×10^{23}

Table 4: **Comparison of the two methods for $m = 4$ on function value**

P	N	*Steepest descent*	*Limited-Memory BFGS*
1	10^3	0	4.2692×10^3
2	10^3	1.5114×10^5	9.7304×10^3
3	10^3	0	2.9868×10^{-1}
4	10^3	6.0317	32.595

Table 5: **Comparison of the two methods for $m = 6$ on function value**

P	N	*Steepest descent*	*Limited-Memory BFGS*
1	10^3	0	1.1273×10^3
2	10^3	1.5068×10^5	9.7296×10^{-3}
3	10^3	7.1928×10^{-8}	1.0795×10^{-5}
4	10^3	6.0317	6.0317

Table 7 to Table 9 focus on the comparison for the norm of gradient as solution tends to the optimum value.

Table 6: **Comparison of the norm of gradient of the two methods for $m = 2$**

P	N	*Steepest descent*	*Limited-Memory BFGS*
1	1000	0	57.060
2	1000	1.4146×10^3	6.9366×10^{-5}
3	1000	4.6056×10	3.1173
4	1000	8.5570×10^{-7}	2.8544×10^{15}

Table 7: **Comparison of the norm of gradient of the two methods for $m = 4$**

P	N	*Steepest descent*	*Limited-Memory BFGS*
1	1000	0	6.5250
2	1000	1.6787×10^3	1.8992×10^{-4}
3	1000	0	8.1280×10
4	1000	2.1221×10^{-7}	1.9531×10^2

Table 8: **Comparison of the norm of gradient of the two methods for $m = 6$**

P	N	*Steepest descent*	*Limited-Memory BFGS*
1	1000	0	3.1648
2	1000	1.6749×10^3	1.7542×10^{-4}
3	1000	5.3639×10^{-1}	1.5238
4	1000	3.7336×10^{-7}	3.7253×10^{-5}

By comparing these tables one may find that the continuous-time L-BFGS performs better in Problem 2, while the continuous-time steepest descent method performs better in Problem 4. In Problems 1 and 3, there is not much difference.

In order to show the full potential of the continuous-time L-BFGS, it is our next goal to use a stiff dde solver, which will be performed in the next phase of this research.

3.3 A new code using Radar5

For stiff problems, including differential-algebraic and neutral delay equations with constant or state-dependent (eventually vanishing) delays, the code RADAR5 written by Ernest Hairer is more appropriate. This code uses collocation methods based on Radau nodes and solves dde's of the type

$$My'(t) = f\left(t, y(t), y(t - \alpha_1(t, y(t))), \ldots, y(t - \alpha_m(t, y(t)))\right)$$

$$y(t_0) = y_0,\ y(t) = g(t) \quad \text{for} \quad t < t_0,$$

where M is a constant $d \times d$ matrix and $\alpha_i(t, y(t)) < t$ for all $t \geq t_0$ and for all i. The value $g(t_0)$ may be different from y_0, allowing for a discontinuity at t_0. (Collocation methods have been proved to have excellent stability properties for delay equations.)

The code RADAR5 is written in ANSI Fortran-90 and is made up of several routines. Because our main test programs are written in MATLAB, we need to call FORTRAN from MATLAB. This is not an easy task and will be tackled in the next phase of our research.

4 Main stages of this research

(A) Prove that the function H in (13) is positive definite. (Done and shown in Appendix I).

(B) Prove that H is Lipschitz continuous. (Still ongoing)

(C) Show that the solution to (13) is asymptotically stable. (A simple consequence of (B).)

(D) Show that (13) has a better rate of convergence than the dynamical system based on the steepest descent direction.

(E) Perform numerical testing.

(F) Apply this new optimization method to practical problems.

APPENDIX I: To show that H in (13) is positive definite.

Without loss of ambiguity, in the subsequent proof, we drop the $t, t_0, t_1, \ldots, t_k, t_{k+1}$, ect. in $s_k(t), y_k(t), V_k(t)$, and so on below.

Property 1 If H_0 is positive definite, the matrix H defined by (13) is positive definite (provided that $y_i^T s_i > 0$ for all i).

Proof: We prove the result by induction. From the above discussion we know that (13), the continuous analog of the L-BFGS formula, has two cases. Hence our proof needs to cater for each of them.

For the first case $k+1 > m$, note that when $m = 1$

$$\begin{aligned} H_{k+1} &= V_k^T H_0 V_k + \rho_k s_k s_k^T \\ &= H_0 - \frac{H_0 y_k s_k^T}{y_k^T s_k} - \frac{s_k y_k^T H_0}{y_k^T s_k} \\ &\quad + \frac{s_k y_k^T H_0 y_k s_k^T}{(y_k^T s_k)^2} + \frac{s_k s_k^T}{y_k^T s_k} \end{aligned}$$

It is obvious that the proof of p.d. of this matrix is the same as that of Theorem 1 in section 1.3. Therefore, H_{k+1} is p.d. when $m = 1$.

Now suppose they are true for $m = l$, we show that they are true for $m = l+1$.

When $m = l$, we have (denoting H_{k+1} by H_{k+1}^l to emphasize $m = l$)

$$\begin{aligned} H_{k+1}^l &= V_k^T V_{k-1}^T \cdots V_{k-l+1}^T H_0 V_{k-l+1} \cdots V_{k-1} V_k \\ &\quad + \{V_k^T \cdots V_{k-l+2}^T \rho_{k-l+1} s_{k-l+1} s_{k-l+1}^T \\ &\qquad \cdot V_{k-l+2} \cdots V_k \\ &\quad + V_k^T \cdots V_{k-l+3}^T \rho_{k-l+2} s_{k-l+2} s_{k-l+2}^T \\ &\qquad \cdot V_{k-l+3} \cdots V_k \\ &\quad \vdots \\ &\quad + V_k^T V_{k-1}^T \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} V_k \\ &\quad + V_k \rho_{k-1} s_{k-1} s_{k-1}^T V_k \\ &\quad + \rho_k s_k s_k^T\}. \end{aligned}$$

being positive definite. (There are $l+1$ terms in H_{k+1}^l)

If $m = l+1$, from (13B)

$$\begin{aligned} H_{k+1}^{l+1} &= V_k^T V_{k-1}^T \cdots V_{k-l}^T H_0 V_{k-l} \cdots V_{k-1} V_k \\ &\quad + V_k^T \cdots V_{k-l+1}^T \rho_{k-l} s_{k-l} s_{k-l}^T \\ &\qquad \cdot V_{k-l+1} \cdots V_k \\ &\quad + \{V_k^T \cdots V_{k-l+2}^T \rho_{k-l+1} s_{k-l+1} s_{k-l+1}^T \\ &\qquad \cdot V_{k-l+2} \cdots V_k \\ &\quad + V_k^T \cdots V_{k-l+3}^T \rho_{k-l+2} s_{k-l+2} s_{k-l+2}^T \\ &\qquad \cdot V_{k-l+3} \cdots V_k \\ &\quad \vdots \\ &\quad + V_k^T V_{k-1}^T \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} V_k \\ &\quad + V_k \rho_{k-1} s_{k-1} s_{k-1}^T V_k \\ &\quad + \rho_k s_k s_k^T\}. \end{aligned}$$

(There are $l+2$ terms in H_{k+1}^{l+1}.)

Comparing these two equations we find that the terms in curly braces are the same, and let

$$\begin{aligned} (*) &= V_k^T \cdots V_{k-l+2}^T \rho_{k-l+1} s_{k-l+1} s_{k-l+1}^T \\ &\qquad \cdot V_{k-l+2} \cdots V_k \\ &\quad + V_k^T \cdots V_{k-l+3}^T \rho_{k-l+2} s_{k-l+2} s_{k-l+2}^T \\ &\qquad \cdot V_{k-l+3} \cdots V_k \\ &\quad \vdots \\ &\quad + V_k^T V_{k-1}^T \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} V_k \\ &\quad + V_k \rho_{k-1} s_{k-1} s_{k-1}^T V_k \\ &\quad + \rho_k s_k s_k^T \end{aligned}$$

Thus,

$$\begin{aligned} H_{k+1}^l &= V_k^T V_{k-1}^T \cdots V_{k-l+1}^T H_0 V_{k-l+1} \cdots V_{k-1} V_k \\ &\quad + (*) \end{aligned}$$

$$\begin{aligned} H_{k+1}^{l+1} &= V_k^T V_{k-1}^T \cdots V_{k-l}^T H_0 V_{k-l} \cdots V_{k-1} V_k \\ &\quad + V_k^T \cdots V_{k-l+1}^T \rho_{k-l} s_{k-l} s_{k-l}^T \\ &\qquad \cdot V_{k-l+1} \cdots V_k + (*) \\ &= V_k^T V_{k-1}^T \cdots V_{k-l+1}^T (V_{k-l}^T H_0 V_{k-l} + \rho_{k-l} \\ &\qquad \cdot s_{k-l} s_{k-l}^T) V_{k-l+1} \cdots V_{k-1} V_k + (*) \end{aligned}$$

Since we have assumed that H_{k+1}^l is p.d., if we try to prove that H_{k+1}^{l+1} is also p.d., we should prove that

$V_{k-l}^T H_0 V_{k-l} + \rho_{k-l} s_{k-l} s_{k-l}^T$ and H_0 have the same property, i.e., $V_{k-l}^T H_0 V_{k-l} + \rho_{k-l} s_{k-l} s_{k-l}^T$ is also p.d..

Now we move forward to prove that $V_{k-l}^T H_0 V_{k-l} + \rho_{k-l} s_{k-l} s_{k-l}^T$ is p.d..

From the proof before, we have the following conclusion:

Consider the formula

$$B = V_k^T A V_k + \rho_k s_k s_k^T \qquad k = 0, 1, \cdots$$

(The definition of V_k, ρ_k, s_k are the same as in the L-BFGS formula.) If we know A is a p.d. matrix, then B is also p.d..

In this sense, it is easy to know that $V_{k-l}^T H_0 V_{k-l} + \rho_{k-l} s_{k-l} s_{k-l}^T$ is p.d.. As we have assumed that H_{k+1}^l is p.d. when H_0 is p.d., we conclude that H_{k+1}^{l+1} is p.d..

For the second case $k + 1 \leq m$,

$$\begin{aligned} H_{k+1} = {} & V_k^T V_{k-1}^T \cdots V_0^T H_0 V_0 \cdots V_{k-1} V_k \\ & + V_k^T \cdots V_1^T \rho_0 s_0 s_0^T V_1 \cdots V_k \\ & \vdots \\ & + V_k^T V_{k-1}^T \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} v_k \\ & + V_k \rho_{k-1} s_{k-1} s_{k-1}^T V_k \\ & + \rho_k s_k s_k^T. \end{aligned}$$

We also use induction to prove H_{k+1} is p.d..

Firstly, $H_1 = V_0^T H_0 V_0 + \rho_0 s_0 s_0^T$ from above, so it is clearly that H_1 is p.d..

Secondly, assumed that H_k is p.d.. We are going to prove that H_{k+1} is also p.d.. We have assumed

$$\begin{aligned} H_k = {} & V_{k-1}^T V_{k-2}^T \cdots V_0^T H_0 V_0 \cdots V_{k-2} V_{k-1} \\ & + V_{k-1}^T \cdots V_1^T \rho_0 s_0 s_0^T V_1 \cdots V_{k-1} \\ & + V_{k-1}^T \cdots V_2^T \rho_1 s_1 s_1^T V_2 \cdots V_{k-1} \\ & \vdots \\ & + V_{k-1} \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} \\ & + \rho_{k-1} s_{k-1} s_{k-1}^T. \end{aligned}$$

is p.d.. Hence,

$$\begin{aligned} H_{k+1} = {} & V_k^T V_{k-1}^T \cdots V_0^T H_0 V_0 \cdots V_{k-1} V_k \\ & + V_k^T \cdots V_1^T \rho_0 s_0 s_0^T V_1 \cdots V_k \\ & \vdots \\ & + V_k^T V_{k-1}^T \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} v_k \\ & + V_k \rho_{k-1} s_{k-1} s_{k-1}^T V_k \\ & + \rho_k s_k s_k^T \\ = {} & V_k^T (V_{k-1}^T V_{k-2}^T \cdots V_0^T H_0 V_0 \cdots V_{k-2} V_{k-1} \\ & + V_{k-1}^T \cdots V_1^T \rho_0 s_0 s_0^T V_1 \cdots V_{k-1} \\ & + V_{k-1}^T \cdots V_2^T \rho_1 s_1 s_1^T V_2 \cdots V_{k-1} \\ & + \cdots + V_{k-1} \rho_{k-2} s_{k-2} s_{k-2}^T V_{k-1} \\ & + \rho_{k-1} s_{k-1} s_{k-1}^T) V_k + \rho_k s_k s_k^T \\ = {} & V_k^T H_k V_k + \rho_k s_k s_k^T. \end{aligned}$$

Therefore, we have proved that H_{k+1} is p.d. when $k + 1 \leq m$.

So we have proved the property for both cases $k + 1 \leq m$ and $k + 1 > m$. $\square$

The proof of the property above is part of our work on the delayed dynamical systems approach for unconstrained optimization. The requirement that $y_i^T s_i > 0$ for all i in Property 1 is not a major issue, because we work with a continuous ode and the numerical method can always be restarted.

References

[1] Aluffi-Pentini, F., Parisi, V. and Zirilli, F. Algorithm 617 DAFNE: A differential equations algorithm for nonlinear equations, ACM Trans. on Math. Software, 10 (3), 1984, 317-324.

[2] Boggs, P. T., The solution of nonlinear systems of equations by A-stable integration techniques, SIAM J. Numer. Anal. 8 (4), 1971, 767-785.

[3] Botsaris, C. A. and Jacobson, D. H., A Newton-type curvilinear search method for optimization, JMAA, 54, 1976, 217-229.

[4] Brown, A. A. and Bartholomew-Biggs, M. C., ODE versus SQP methods for constrained optimization, JOTA 62 (3), 1989, 371-386.

[5] Chen, Y. H. and Fang, S. C., Solving convex programming problem with equality constraints by neural networks, Computers Math. Appl. 36, 1998, 41-68.

[6] Chu, M. T., On the continuous realization of iterative processes, SIAM Review 30 (3), 1988, 375-387.

[7] Han, Q., Liao, L.-Z., Qi, H. and Qi, L., Stability analysis of gradient-based neural networks for optimization problems, J. Global Optim. 19 (4), 1962, 363-381.

[8] Hassan, N. and Rzymowski, W., An ordinary differential equation in nonlinear programming. Nonlinear Analysis, Theory, Method & Applications 15 (7), 1990, 597-599.

[9] aykin, S. S., Neural Networks: A Comprehensive Foundation, Prentice-Hall, Englewood Cliffs, NJ, 1994.

[10] He, B. S., Inexact implicit methods for monotone general variational inequalities, Mathematical Programming, 86 (1), 1999, 199-217.

[11] Incerti, S., Parisi, V. and Zirilli, F., A new method for solving nonlinear simultaneous equations, SIAM J. Numer. Anal. 16, 1979, 779-789.

[12] Jorge Nocedal, Updating Quasi-Newton Matrices With Limited Storage, Mathematics of Computation, Vol 35, No 151, July 1980, pp. 773-782.

[13] Liao, L.-Z. and Qi, H., A neural network for the linear complementarity problem, Math. Comput. Modelling 29 (3), 1999, 9-18.

[14] L.Z. Liao, L.Q. Qi, and H.W. Tam, A gradient-based continuous method for large-scale optimization problems, Journal of Global Optimization, Vol 31 , Apr 2005, pp. 271-286.

[15] Liao, L.-Z., Qi, H. and Qi, L., Solving nonlinear complementarity problems with neural networks: a reformulation method approach, JCAM 131 (12), 2001, 343-359.

[16] LI-ZHI LIAO, HOUDUO QI and LIQUN QI, (2004), Neurodynamical Optimization, Journal of Global Optimization 28: 175-195.

[17] Novakovi?c, Z. R., Solving systems of non-linear equations using the Lyapunov direct method, Computers Math. Applic. 20 (12), 1990, 19-23.

[18] Tanabe, K., A geometric method in nonlinear programming, JOTA 30 (2), 1980, 181-210.

[19] Tank, D. W. and Hopfield, J. J., Simple neural optimization networks: An A/D convert, signal decision circuit, and a linear programming circuit, IEEE Trans. Circuits Syst. 33, 1986, 533-541.

[20] Wilde, N. G., A note on a differential equation approach to nonlinear programming, Management Science 15 (11), 1969, 739-739.

[21] Williems, J. L., Stability Theory of Dynamical Systems, Nelson, 1970.

[22] Wu, X., Xia, Y., Li, J. and Chen, W. K., A high performance neural network for solving linear and quadratic programming problems, IEEE Trans. Neural Networks, 7, 1996, 643-651.

[23] Xia, Y., A new neural network for solving linear programming problems and its applications, IEEE Trans. Neural Networks 7, 1996, 525-529.

[24] Yamashita, H., A differential equation approach to nonlinear programming, Math. Prog. 18,1980, 155-168.

[25] Zabcayk, J ., Mathematical Control Theory: An Introduction, Birkhauser, Boston, 1992.