# Action constrained quasi-Newton methods

R. M. Gower and J. Gondzio*

December 30, 2014

**Abstract**

At the heart of Newton based optimization methods is a sequence of symmetric linear systems. Each consecutive system in this sequence is similar to the next, so solving them separately is a waste of computational effort. Here we describe automatic preconditioning techniques for iterative methods for solving such sequences of systems by maintaining an estimate of the inverse system matrix. We update the estimate of the inverse system matrix with quasi-Newton type formulas based on what we call an *action constraint* instead of the secant equation. We implement the estimated inverses as preconditioners in a Newton-CG method and prove quadratic termination. Our implementation is the first parallel quasi-Newton preconditioners, in full and limited memory variants. Tests on logistic Support Vector Machine problems reveal that our method is very efficient, converging in wall clock time before a Newton-CG method without preconditioning. Further tests on a set of classic test problems reveal that the method is robust. The action constraint makes these updates flexible enough to mesh with trust-region and active set methods, a flexibility that is not present in classic quasi-Newton methods.

**Keywords**: quasi-Newton method, inexact Newton method, preconditioners, linear systems, conjugate gradients, balancing preconditioner.

## 1 Introduction

### 1.1 Motivation

Second order methods for unconstrained nonlinear optimization display several advantages: they deliver a high accuracy of computations and enjoy a fast (quadratic) local convergence. However, these benefits may sometimes come at too high a cost. Indeed, evaluating the full Hessian and solving equations with it is sometimes very expensive and occasionally prohibitive. Several approaches have been designed over the years to remove some of the drawbacks of the second order methods while preserving their main advantages. Those include the *inexact* Newton methods [11] and a family of *quasi-Newton* methods [5, 6, 14].

The inexact Newton method admits a (controlled) error in solving the Newton system and therefore allows to employ matrix-free iterative solvers that only apply the system matrix as an

*School of Mathematics and Maxwell Institute for Mathematical Sciences, The University of Edinburgh, corresponding author: gowerrobert@gmail.com

operator. These iterative methods only sample the action of the system operator, circumventing the cost of calculating the entire Hessian matrix. Quasi-Newton methods follow a completely different logic: they build an approximation of the inverse Hessian using low-rank updates derived from information on how the Hessian operates along a given direction.

The motivation behind this paper is to combine these two approaches: Use samples of the Hessian's action made available from an iterative solver to build an approximation to the inverse Hessian. This approximation is then used to precondition and solve the subsequent Newton system, and the process is repeated. The methods proposed in this paper and their analysis are based on the quasi-Newton literature.

The development of quasi-Newton methods was pioneered by Davidon in the late 50's [10] and culminated in the BFGS method, named to honour the independent developments of Broyden [5], Fletcher [13], Goldfarb [19] and Shanno[37] over the 60's and early 70's. Nowadays, these methods are frequently referred to as members of the Broyden family [5, 6, 14].

Quasi-Newton methods obtain/improve an estimate $G_{k+1} \in S^n$ of the Hessian matrix $\nabla^2 f_{k+1} := \nabla^2 f(x_{k+1})$, where $S^n$ is the set of symmetric matrices in $\mathbb{R}^{n\times n}$, $f \in C^2(\mathbb{R}^n)$ and $x_{k+1} \in \mathbb{R}^n$. Their input is a previous estimate $G_k$ and a desired *action* for the new estimate $G_{k+1} : \delta_k \to \gamma_k$, that is

$$\gamma_k = G_{k+1}\delta_k,$$

where $\delta_k = x_{k+1} - x_k$ and $\gamma_k = \nabla f_{k+1} - \nabla f_k$. From the fundamental theorem of calculus

$$\gamma_k = \left(\int_0^1 \nabla^2 f(x_k + t\delta_k)dt\right)\delta_k,$$

so $G_{k+1}$ has the same action as $\int_0^1 \nabla^2 f(x_k + t\delta_k)dt$ applied to $\delta_k$. Alternatively, to obtain an estimate of the (pseudo-)inverse Hessian, the action is inverted and imposed as $G_{k+1} : \gamma_k \to \delta_k$.

This setup can produce approximate Hessians (or their inverse) from any observed action, in particular, when samples of the Hessian's action $d \to \nabla^2 f_{k+1} d$, with $d \in \mathbb{R}^n$, are available. Though this limitation of incorporating only a 1-dimensional action is a hindrance when meshing quasi-Newton methods with inexact Newton methods because, in contrast, inexact solvers make available the sampled action of the Hessian on a *subspace* (most often with more dimensions than one). This mismatch has resulted in two strategies:

(i) Limit the inflow of new information to using only the action of the Hessian on a *single* direction per iteration [4, 3].

(ii) Use a basis for the subspace and associated Hessian's action, to sequentially update the approximation [28]. This is costly and *cannot be parallelized.*

We present a generalization of quasi-Newton methods which overcomes this drawback.

Instead of sampling the Hessian's action on a single direction, we sample it on a low dimensional subspace. This guarantees a much faster influx of information and produces better approximations. Using a set of directions at one time also allows us to perform updates that exploit block-matrix operations which can be executed in parallel.

Since the new methods exploit the Hessian's action along a set of directions, we call them the *quasi-Newton Action Constrained* methods, `quNac` for short.

The motivation to develop `quNac` comes from the need to solve large and difficult problems. Therefore all computational aspects of the method are taken into serious consideration. In particular, we embed `quNac` into a Newton-CG scheme. We discuss several variants of a possible implementation of `quNac` and provide preliminary computational results which demonstrate its efficiency on non-trivial medium scale problems.

The next section contains the problem formulation and introduces the notation used in the paper. From this initial motivation, we have broadened our scope to include preconditioning techniques for solving a sequence of (slowly) changing symmetric systems of equations as opposed to focusing on a sequence of Newton systems. Throughout the development we embrace two possible cases; when `quNac` approximations are developed either for estimating the system matrix or its inverse.

### 1.2 Background

Consider the problem of sequentially solving in $d_k \in \mathbb{R}^n$ the symmetric systems

$$Q_k d_k = b_k, \quad \text{for } k = 1, 2, \ldots, \tag{1}$$

where $Q_k \in S^n$ and $b_k \in \mathbb{R}^n$. Here the $Q_k$'s are "slowly changing" in the sense that $\|Q_{k+1} - Q_k\|$ is relatively small in some matrix norm. We make no assumption on the $\{b_k\}$ sequence. Such slowly changing *target matrices* $\{Q_k\}$ can arise from evaluating a continuous matrix field over neighboring points, such as is the case with the Hessian matrix in Newton type methods when step sizes are small. Sequences of symmetric systems also appear when solving nonlinear systems with the Newton-Raphson method and the Jacobian is symmetric, such as discretizations of the Nonlinear Schrödinger [39] and the complex Ginzburg-Landau equation [1].

Solving a single system in (1) through iterative methods involves calculating $Q_{k+1}\mathcal{S}_k$, the action of $Q_{k+1}$ over a low dimensional *sampling matrix* $\mathcal{S}_k \in \mathbb{R}^{n\times q}$, as opposed to requiring the entire matrix $Q_{k+1}$. This raises a question of how can one estimate the *target matrix* $Q_{k+1}$, or its inverse, from this *sampled action*.

Our strategy is to maintain an *estimate matrix* $G_k \in S^n$ of $Q_k$, and use the sampled action $\mathcal{S}_k \to Q_{k+1}\mathcal{S}_k$ to update $G_k$ and to produce a new estimate $G_{k+1} \in S^n$. To determine a unique $G_{k+1}$, and exploit that $\|Q_{k+1} - Q_k\|$ is small, we minimize $\|G_{k+1} - G_k\|$ subject to an *action* constraint

$$G_{k+1}\mathcal{S}_k = Q_{k+1}\mathcal{S}_k,$$

and a symmetry constraint

$$G_{k+1} = G_{k+1}^T.$$

This is known as the *least change* strategy in the quasi-Newton literature, first proposed by Greenstadt in 1969 [23]. We henceforth refer to the problem of determining $G_{k+1}$ under these constraints and the least change objective as the *least change problem*. As the constraint set

$\{G \in \mathbb{R}^{n\times n} \,|\, G = G^T,\, G\mathcal{S}_k = Q_{k+1}\mathcal{S}_k\}$ is a subspace of $\mathbb{R}^{n\times n}$, the resulting solution $G_{k+1}$ of the least change problem is a projection of $G_k$ onto this constraint set. This characterization as a projection is useful for including additional constraints as shown in the classic quasi-Newton setting by Dennis and Schnabel [12].

The sampled action also offers information on the (pseudo-)inverse of $Q_{k+1}$ when it exists as

$$Q_{k+1}^{-1}(Q_{k+1}\mathcal{S}_k) = \mathcal{S}_k.$$

Thus with an estimate $H_k \in \mathbb{R}^{n\times n}$ of the (pseudo-)inverse of $Q_k$, a new estimate can be obtained by minimizing the least change objective, imposing the following action constraint

$$H_{k+1}(Q_{k+1}\mathcal{S}_k) = \mathcal{S}_k,$$

and the symmetry constraint. We use the same technique to calculate the direct or inverse estimate, the difference being which action we impose, $Q_{k+1}\mathcal{S}_k \to \mathcal{S}_k$ or $\mathcal{S}_k \to Q_{k+1}\mathcal{S}_k$.

As our main application, we build estimates of inverse Hessian matrices to act as preconditioners in the Newton-CG method. In the unconstrained minimization of a function $f \in C^2(\mathbb{R}^n, \mathbb{R})$, given an initial $x_0 \in \mathbb{R}^n$, the Newton-CG method approximately solves systems,

$$\nabla^2 f_k d_k = -\nabla f_k,$$

using the Conjugate Gradient method [24], where $\nabla^2 f_k := \nabla^2 f(x_k)$ is the Hessian matrix and $\nabla f_k := \nabla f(x_k)$, the gradient evaluated at $x_k \in \mathbb{R}^n$. A line search is then performed to calculate a step size $a_k \in \mathbb{R}_+$ and iterate

$$x_{k+1} = x_k + a_k d_k.$$

In the Conjugate Gradient method, the action of the Hessian matrix is sampled on a low dimensional Krylov subspace. With this sampled action we construct an estimate $G_k$ that is used to precondition the next Newton system $H_k\nabla^2 f_{k+1} d_{k+1} = -H_k \nabla f_{k+1}$.

### 1.3 Format of the paper

After examining previous work and connections to our own in Section 1.4, in Section 2.1 we solve the least change problem with a weighted Frobenius norm. Then we explore properties of the approximation matrices, such as sufficient conditions on the sampling matrix and target matrix to ensure the quadratic hereditary property and positive definiteness, both important in the context of preconditioning and in nonlinear optimization. This is followed by Proposition 2.3 that shows when is the `quNac` update equivalent to applying a sequence of rank-2 updates. This is used to establish the connection between sequential BFGS and DFP updates and `quNac` updates.

We then specialize this updating scheme to Hessian matrices in Section 3 and develop a family of methods analogous to the Broyden family [5]. In Section 4 we recap the Preconditioned Conjugate Gradients (PCG) method, followed by Section 5 where we detail a preconditioned Newton-CG method which employs `quNac` in a full or limited memory variant that guarantee descent directions. We contrast our limited memory `quNac` implementation to Morales and Nocedal's L-BFGS

preconditioner [28], showing that the former is a parallel version of the latter. The quadratic hereditary of this Newton-PCG method is proved in Section 5.1, followed by promising numerical tests in Section 6, comparing the new method to Newton-CG, BFGS and L-BFGS on academic problems and regularized logistic regression problems with real data. Finally we summarize our findings in Section 7.

### 1.4 Prior work and Connections

A member of the `quNac` methods apparently first appeared in domain decomposition methods for solving PDE's [27] where it is referred to as a balancing preconditioner. The domain decomposition methods give rise to a single large linear system which is block structured. After solving systems defined by the individual blocks, often in the least-squares sense, the balancing preconditioner aggregates these solutions into a symmetric preconditioner for the original large system. Our results enrich the balancing preconditioners by showing that they are a projection of a first guess preconditioner (the Neumman-Neumman preconditioner) onto the space of matrices with desirable properties (symmetric and having the same action as the (pseudo-)inverse over the direct sum of the nullspaces of the block matrices). Furthermore, we show that the balancing preconditioner is but one of a family of preconditioners that have these properties.

The balancing preconditioner has been taken out of the PDE context and tested as a general purpose preconditioner for solving a single linear system and systems with changing right hand side by Gratton, Sartenaer and Ilunga [22]. Gratton *et al.* prove favourable spectral properties of the balancing preconditioner and study its relation to multiple BFGS updates. Our analysis of the quadratic hereditary property indicates how one might sequentially update a preconditioner using the balancing preconditioner formula, which in turn allows us to extend the method to solving sequences of linear systems where the system matrix also changes.

The problem of solving sequences of linear systems has also been addressed by recycling Krylov subspace methods [33, 16, 26] and in [18] when only the right-hand side changes. In these methods, a selected Krylov subspace is retained from a previous system solve that serves as an approximate eigenspace to improve the conditioning of the next system.

Alternatively, updating a factorized preconditioner is possible, such as partial LU decomposition for nonsymmetric systems [40] and constraint preconditioners [2].

Building a preconditioner through Frobenius norm probing [25] for a single linear system has a similar flavour to our preconditioning method, where $H_{k+1}$ is obtained by approximately minimizing $\|H_{k+1}Q_{k+1} - I\|_F$ subject to an additional action constraint that is incorporated into the objective function as a penalty. These aforementioned approaches, and addressed problems, are notably distinct from ours. Rather, our setup is heavily borrowed from that of quasi-Newton methods.

Schnabel [36] shows how to build estimate matrices that satisfy multiple secant equations, and in doing so, obtains generalizations of the Powell-Symmetric-Broyden (*PSB*), BFGS and DFP formulas. He then goes on to show that these generalizations are the solutions of the least change problem with a particular weighted Frobenius norm. By swapping multiple secant updates for an

action constraint, Schnabel's generalized BFGS and DFP are equivalent to our inverse and direct `quNac` method presented in Section 3.

The least change problem was first proposed and solved for the standard quasi-Newton updates [23, 19] but to the best of our knowledge this paper is the first that solves the problem with a general action constraint and for any positive definite weighting matrix in the Frobenius norm.

Outside of the preconditioning literature, our proposed matrix optimization problem has connections to low rank matrix completion [7]. With a previous estimate $G_k = 0$, one can view the action constraint as a sampling of the target matrix through its action on a subspace. The least change solution $G_{k+1}$ then leads to low rank solutions of at most three times the number of columns in the sampling matrix.

## 2 The quasi-Newton action constrained methods

### 2.1 The least change problem

We now deduce the solution to the least change problem for a general action constraint and weighted Frobenius norm. This includes and extends Schnabel's generalized BFGS, DFP and PSB methods [36].

Given an estimate matrix $G_k \in S^n$, our objective is to calculate an update matrix $E_k \in S^n$ such that $G_k + E_k$ is an estimate of the target matrix $Q_{k+1} \in S^n$. To ensure that the update matrix is the least change to $G_k$, it is obtained by minimizing a weighted Frobenius norm

$$\|\mathcal{W}_k^{-1/2} E_k \mathcal{W}_k^{-1/2}\|_F^2 := \mathrm{Tr}\left(\mathcal{W}_k^{-1} E_k \mathcal{W}_k^{-1} E_k^T\right), \tag{2}$$

where $\mathcal{W}_k \in S^n$ is a positive definite *weighting* matrix. To impose that $G_{k+1}$ remains symmetric, we use a symmetry constraint

$$E_k = E_k^T. \tag{3}$$

The action constraint is imposed as

$$E_k \mathcal{S}_k = (Q_{k+1} - G_k)\mathcal{S}_k, \tag{4}$$

where $\mathcal{S}_k \in \mathbb{R}^{n\times q}$, $q$ an integer considerably smaller than $n$ and $\mathcal{S}_k$ is full rank.

Dropping the iteration index $k$, collecting the objective function (2), symmetry constraint (3) and the action constraint (4) we have the *least change problem* that characterizes our update

$$\min_E \frac{1}{2}\mathrm{Tr}\left(\mathcal{W}^{-1} E \mathcal{W}^{-1} E^T\right) \tag{5}$$

$$E\mathcal{S} = R\mathcal{S} \tag{6}$$

$$E = E^T, \tag{7}$$

where $R \in S^n$ is a given symmetric matrix. We now deduce the solution to the least change problem which is one of the central results of this article. A key definition we repeatedly use is

$$\mathrm{proj}_{\mathcal{S}}^{\mathcal{W}} := \mathcal{S}(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1}\mathcal{S}^T,$$

thus $\text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}$ is an oblique projection onto the space spanned by the columns of $\mathcal{S}$. The following demonstration is not necessary for the development of the remainder of the article, and the reader may jump ahead to the solution (16).

The objective function of the least change problem (5) is a convex quadratic function of $E$ and the constraints are linear. Thus the solution is unique and characterized by the KKT conditions. The Lagrangian of our least change problem is given by

$$\Phi(E,\Lambda,\Gamma) = \frac{1}{2}\text{Tr}\left(\mathcal{W}^{-1}E\mathcal{W}^{-1}E^T\right) + \text{Tr}\left(\Lambda^T(E-R)\mathcal{S}\right) + \text{Tr}\left(\Gamma(E-E^T)\right),$$

where $\Lambda \in \mathbb{R}^{n\times q}$ and $\Gamma \in \mathbb{R}^{n\times n}$. Differentiating (for a comprehensive list of formulas on matrix differentiation please consult [35]) in $E$ we have

$$D_E\Phi(E,\Lambda,\Gamma) = \mathcal{W}^{-1}E\mathcal{W}^{-1} + \Lambda\mathcal{S}^T + \Gamma^T - \Gamma.$$

Setting $D_E\Phi(E,\Lambda,\Gamma)$ to zero and isolating $E$ gives

$$E = \mathcal{W}(\Gamma - \Lambda\mathcal{S}^T - \Gamma^T)\mathcal{W}. \tag{8}$$

Using the symmetry constraint (7) of $E$ we find that

$$\Gamma - \Gamma^T = \frac{1}{2}\left(\Lambda\mathcal{S}^T - \mathcal{S}\Lambda^T\right).$$

Substituting back into (8) gives

$$E = -\frac{1}{2}\mathcal{W}\left(\mathcal{S}\Lambda^T + \Lambda\mathcal{S}^T\right)\mathcal{W}. \tag{9}$$

The solution $E$ is now solely determined by $\Lambda\mathcal{S}^T$, and we focus on obtaining this matrix. Right multiplying by $\mathcal{S}$ and using the action constraint (6) then left multiplying by $\mathcal{W}^{-1}$ gives

$$\mathcal{W}^{-1}R\mathcal{S} = -\frac{1}{2}\left(\mathcal{S}\Lambda^T + \Lambda\mathcal{S}^T\right)\mathcal{W}\mathcal{S}. \tag{10}$$

If the columns of $\mathcal{S}$ are linearly independent then $\mathcal{S}^T\mathcal{W}\mathcal{S}$ is invertible. Isolating $\Lambda$

$$\Lambda = -\left(\mathcal{S}\Lambda^T\mathcal{W}\mathcal{S} + 2\mathcal{W}^{-1}R\mathcal{S}\right)(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1}. \tag{11}$$

Right multiplying by $\mathcal{S}^T$ we find that

$$\Lambda\mathcal{S}^T = -\left(\mathcal{S}\Lambda^T\mathcal{W} + 2\mathcal{W}^{-1}R\right)\text{proj}_{\mathcal{S}}^{\mathcal{W}}. \tag{12}$$

From (12) we see that $\Lambda\mathcal{S}^T$ is equal to an unknown matrix times the matrix $\text{proj}_{\mathcal{S}}^{\mathcal{W}}$. This is a fact we shall use later on in the demonstration. Left multiplying by $\mathcal{S}^T\mathcal{W}$ in (11), we get

$$\mathcal{S}^T\mathcal{W}\Lambda = -\mathcal{S}^T\mathcal{W}\left(\mathcal{S}\Lambda^T\mathcal{W}\mathcal{S} + 2\mathcal{W}^{-1}R\mathcal{S}\right)(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1},$$

transposing

$$\Lambda^T\mathcal{W}\mathcal{S} = -(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1}\left(\mathcal{S}^T\mathcal{W}\Lambda\mathcal{S}^T + 2\mathcal{S}^TR\mathcal{W}^{-1}\right)\mathcal{W}\mathcal{S}.$$

Substituting this into (11) we get

$$\begin{aligned}\Lambda &= \left(\mathcal{S}(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1}\left(\mathcal{S}^T\mathcal{W}\Lambda\mathcal{S}^T + 2\mathcal{S}^T R\mathcal{W}^{-1}\right)\mathcal{W}\mathcal{S} - 2\mathcal{W}^{-1}R\mathcal{S}\right)(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1} \\ &= \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}\Lambda + 2\left(\text{proj}_{\mathcal{S}}^{\mathcal{W}}R\mathcal{S} - \mathcal{W}^{-1}R\mathcal{S}\right)(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1}.\end{aligned}$$

Right multiplying by $\mathcal{S}^T$ and isolating $\Lambda\mathcal{S}^T$ gives

$$\begin{aligned}(I - \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W})\Lambda\mathcal{S}^T &= 2\left(\mathcal{S}(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1}\mathcal{S}^T R - \mathcal{W}^{-1}R\right)\mathcal{S}(\mathcal{S}^T\mathcal{W}\mathcal{S})^{-1}\mathcal{S}^T \\ &= -2\left(I - \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}\right)\mathcal{W}^{-1}R\text{proj}_{\mathcal{S}}^{\mathcal{W}}.\end{aligned}$$

The above gives the $\left(I - \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}\right)$ projection of $\Lambda\mathcal{S}^T$. It remains to find the $\text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}$ projection of $\Lambda\mathcal{S}^T$. Decomposing $\Lambda\mathcal{S}^T$ according to these projections we find

$$\Lambda\mathcal{S}^T = -2\left(I - \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}\right)\mathcal{W}^{-1}R\text{proj}_{\mathcal{S}}^{\mathcal{W}} + \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}\Lambda\mathcal{S}^T. \tag{13}$$

From (12) we know that there exists $\Psi \in \mathbb{R}^{n\times n}$ such that $\Lambda\mathcal{S}^T = \Psi\text{proj}_{\mathcal{S}}^{\mathcal{W}}$, thus

$$\Lambda\mathcal{S}^T = -2\left(I - \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}\right)\mathcal{W}^{-1}R\text{proj}_{\mathcal{S}}^{\mathcal{W}} + \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}\Psi\text{proj}_{\mathcal{S}}^{\mathcal{W}}. \tag{14}$$

Inserting (14) into (12), after some elimination, we find that

$$2\text{proj}_{\mathcal{S}}^{\mathcal{W}}R\text{proj}_{\mathcal{S}}^{\mathcal{W}} = -\text{proj}_{\mathcal{S}}^{\mathcal{W}}(\mathcal{W}\Psi + (\mathcal{W}\Psi)^T)\text{proj}_{\mathcal{S}}^{\mathcal{W}}.$$

The solution is $\Psi = -\mathcal{W}^{-1}R$, upto additions in the nullspace of $\mathcal{S}$. This reduces (13) to

$$\Lambda\mathcal{S}^T = \left(\text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W} - 2I\right)\mathcal{W}^{-1}R\text{proj}_{\mathcal{S}}^{\mathcal{W}}.$$

Inserting the above in (9) we obtain the solution

$$\begin{aligned}E &= -\frac{1}{2}\left(\left(\mathcal{W}\text{proj}_{\mathcal{S}}^{\mathcal{W}} - 2I\right)R\text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W} + \mathcal{W}\text{proj}_{\mathcal{S}}^{\mathcal{W}}R\left(\text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W} - 2I\right)\right) \\ &= \mathcal{W}\text{proj}_{\mathcal{S}}^{\mathcal{W}}R\left(I - \text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}\right) + R\text{proj}_{\mathcal{S}}^{\mathcal{W}}\mathcal{W}. \quad \blacksquare\end{aligned} \tag{15}$$

Picking up the iteration index $k$ again, identifying $R = Q_{k+1} - G_k$, the projection of $G_k$ onto the subspace of symmetric matrices that satisfy the action constraint is given by

$$\boxed{G_k + E_k = Q_{k+1} + \left(I - \mathcal{W}_k\text{proj}_{\mathcal{S}_k}^{\mathcal{W}_k}\right)(G_k - Q_{k+1})\left(I - \text{proj}_{\mathcal{S}_k}^{\mathcal{W}_k}\mathcal{W}_k\right)}, \tag{16}$$

which is a rank-$3q$ update applied to $G_k$ that only requires knowing $Q_{k+1}\mathcal{S}_k$ and $\mathcal{W}_k\mathcal{S}_k$. The updates (16) include generalization of quasi-Newton methods, analogous to Schnabel's generalization with an action constraint in the place of multiple secant equations. The generalized DFP and Powell-Symmetric-Broyden ($PSB$) method are recovered by substituting $\mathcal{W}_k = Q_k$ and $\mathcal{W}_k = I$, respectively. The generalized BFGS method for estimating the inverse target matrix is recovered by substituting $\mathcal{W}_k = Q_k$ and swapping the occurrences of $Q_k\mathcal{S}_k$ and $\mathcal{S}_k$, so that $Q_k\mathcal{S}_k \to \mathcal{S}_k$ is the imposed action constraint. Different from Schnabel's proof of the generalized BFGS updates, our solution does not assume that $G_k$ is invertible.

We now move on to sufficient conditions that guarantee the quadratic hereditary property and positive definiteness of the resulting approximation matrix.

## 2.2 The quadratic hereditary property

Iteratively updating an estimate $G_k$ using (16), we would like the estimate matrices to gradually converge to the target matrices. Though updating $G_k$ using (16) results in an estimate with the desired action, this update might have a destructive interference on the overall convergence. When the target matrices change little from one iteration to the next, the key to promoting convergence is guaranteeing that the new estimate $G_{k+1}$ inherits the action of the previous estimate $G_k$. In the Proposition below, we prove that this convergence occurs if the target matrix is constant for a number of iterations, say $\rho \in \mathbb{N}$ iterations.

For simplicity, assume that we have a sequence of full rank sampling matrices $\mathcal{S}_i \in \mathbb{R}^{n\times q_i}$ and $\rho, q_i \in \mathbb{N}$ for $i = 1, \ldots, \rho$ such that $\sum_{i=1}^{\rho} q_i = n$.

**Proposition 2.1 (Quadratic Hereditary)** *Let $G_0 \in S^n$ and $G_{k+1} = G_k + E_k$ defined by (16) with $Q_k = Q \in S^n$ and $\mathcal{W}_k \succ 0$ for $k = 0, \ldots, \rho$. If $\mathcal{S}_k^T \mathcal{W}_k \mathcal{S}_i = 0$ for every $i < k \leq \rho$ then*

$$G_{k+1}\mathcal{S}_i = Q\mathcal{S}_i, \quad \text{for } i \leq k \leq \rho, \tag{17}$$

*and $G_{\rho+1} = Q$.*

**Proof:** The proof is by induction on $k$ that (17) is true. For $k = 0$, our hypothesis becomes $G_1\mathcal{S}_0 = Q\mathcal{S}_0$ which is equivalent to the action constraint (4) with $k = 0$. Suppose our hypothesis is true for $k-1$ and let us analyse the $k$ case. For $i = k$, (17) is equivalent to the action constraint (4). For $i \leq k-1$, as $\mathcal{S}_k^T \mathcal{W}_k \mathcal{S}_i = 0$, we have

$$\text{proj}_{\mathcal{S}_k}^{\mathcal{W}_k} \mathcal{W}_k \mathcal{S}_i = 0.$$

Using (16) to substitute $G_{k+1}$, we have

$$\begin{aligned} G_{k+1}\mathcal{S}_i &= Q\mathcal{S}_i + \left(I - \mathcal{W}_k \text{proj}_{\mathcal{S}_k}^{\mathcal{W}_k}\right)(G_k - Q)\left(I - \text{proj}_{\mathcal{S}_k}^{\mathcal{W}_k}\mathcal{W}_k\right)\mathcal{S}_i \\ &= Q\mathcal{S}_i + \left(I - \mathcal{W}_k \text{proj}_{\mathcal{S}_k}^{\mathcal{W}_k}\right)(G_k - Q)\mathcal{S}_i \quad [\text{by induction } G_k\mathcal{S}_i = Q\mathcal{S}_i, \text{ for } i \leq k.] \\ &= Q\mathcal{S}_i. \end{aligned}$$

This concludes the induction.

To prove $G_{\rho+1} = Q$, we need to show that the horizontal concatenation

$$\mathcal{S}_{1:\rho} := [\mathcal{S}_1, \ldots, \mathcal{S}_\rho],$$

is nonsingular. To see this, let $\alpha_i \in \mathbb{R}^{q_i}$, for $i = 0, \ldots, \rho$ be such that

$$\sum_{i=0}^{\rho} \mathcal{S}_i \alpha_i = 0.$$

Left multiplying by $\alpha_\rho \mathcal{S}_\rho^T \mathcal{W}_\rho$ eliminates all terms except $\alpha_\rho^T \mathcal{S}_\rho^T \mathcal{W}_\rho \mathcal{S}_\rho \alpha_\rho = 0$, from which the positive definiteness of $\mathcal{W}_\rho$ and full rank of $\mathcal{S}_\rho$ implies that $\alpha_\rho = 0$. The same procedure with $\alpha_{\rho-1}\mathcal{S}_{\rho-1}^T \mathcal{W}_{\rho-1}$

shows that $\alpha_{\rho-1} = 0$ and so forth. Therefore, $\mathcal{S}_{1:\rho}$ has an inverse. By induction (17) is true for $k = \rho$, thus

$$G_{\rho+1}\mathcal{S}_{1:\rho} = Q\mathcal{S}_{1:\rho}.$$

Right multiplying the inverse of $\mathcal{S}_{1:\rho}$ on both sides shows that $G_{\rho+1} = Q$. ∎

To illustrate the proposition, consider the case where $W_k = I$ in (16) which is a generalization of the PSB method. If the sampling matrices $\mathcal{S}_i$ for $i = 0, \ldots, k$ have mutually orthogonal columns, then Proposition 2.1 states that by updating using the PSB method the resulting $G_{k+1}$ satisfies the quadratic Hereditary property. One way to achieve this would be to use residuals of a Krylov method to form the columns of the sampling matrices. Alternatively, if the weighting matrix satisfies the action constraint, then quadratic hereditary is guaranteed when the columns of the sampling matrix and resulting action matrix are orthogonal.

**Corollary 2.1** *If $\mathcal{S}_k^T Q\mathcal{S}_i = 0$ for $i < k$ and $\mathcal{W}_i\mathcal{S}_i = Q\mathcal{S}_i$ for $i \leq k$ then due to Proposition 2.1, the estimate matrix $G_{k+1}$ satisfies the quadratic Hereditary property.*

The equivalent statements and proofs when the inverse action constraint $Q\mathcal{S}_k \to \mathcal{S}_k$ is imposed follow verbatim by swapping the labels of sampling matrix $\mathcal{S}_k$ and the sampled action $Q\mathcal{S}_k$. For example, after this label swap, Corollary 2.1 remains true though the weighting matrix need satisfy $\mathcal{W}_i Q\mathcal{S}_i = \mathcal{S}_i$ and the resulting quadratic hereditary is $H_{k+1}Q\mathcal{S}_i = \mathcal{S}_i$ for $i \leq k$.

In the following section, we prove a sufficient condition for the solutions of the least change problem (16) to be positive definite.

### 2.3 Positive definiteness

To apply the approximation matrix as a preconditioner, certain solvers require that it be positive definite. Positive definiteness is also important in unconstrained minimization: when we replace the Hessian matrix by an estimate matrix and solve the resulting quasi-Newton system, the search direction is $d_k = -H_k\nabla f_k$. If $H_k$ is positive definite and we are not at a stationary point $\nabla f_k \neq 0$ then $d_k$ is guaranteed to be a descent direction as

$$-\nabla f_k^T d_k = \nabla f_k^T H_k \nabla f_k > 0.$$

The next Lemma and Proposition are the main tools for proving positive definiteness of approximation matrices.

**Lemma 2.1 (Action Constrained Positive Definite Matrix)** *Let $P, A, B \in \mathbb{R}^{n\times n}$ where $A$ and $B$ are positive definite over Range$(P) := \{Px \,|\, x \in \mathbb{R}^n\}$ and Range$(I - P)$ respectively, then the matrix*

$$G = P^T AP + (I - P^T)B(I - P),$$

*is positive definite.*

**Proof:** Let $x \in \mathbb{R}^n$, then

$$x^T G x = x^T P^T A P x + x^T (I-P)^T B (I-P) x \geq 0.$$

If $x^T G x = 0$ then $Px = 0$ and $(I-P)x = 0$ consequentially $x = Px + (I-P)x = 0$. ∎

With Lemma 2.1, we characterize when a subset of estimate matrices that result from (16) are positive definite, namely those with a weighting matrix that satisfies the action constraint $\mathcal{W}_k \mathcal{S}_k = Q_{k+1}\mathcal{S}_k$. With such a weighting matrix, the update (16) takes the form of the update (`quNac`), further down the page. Such a weighting matrix always exists when $\mathcal{S}_k^T Q_{k+1} \mathcal{S}_k$ is positive definite. To see this, let $P = \text{proj}_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1}$ and let

$$\mathcal{W}_k = Q_{k+1} P + (I-P)^T (I-P).$$

The projection matrix guarantees that $\mathcal{W}_k \mathcal{S}_k = Q_{k+1}\mathcal{S}_k$ and, by noting that $Q_{k+1}P = P^T Q_{k+1} P$, Lemma 2.1 guarantees that the matrix $\mathcal{W}_k$ is positive definite.

**Proposition 2.2 (Positive Definite quNac)** *If $G_0$ is positive definite and the product of the sampling matrix with the resulting action $\mathcal{S}_k^T Q_{k+1} \mathcal{S}_k$ is positive definite for $k = 0, \ldots, \rho \in \mathbb{N}$ and*

$$G_{k+1} = Q_{k+1} proj_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1} + \left(I - Q_{k+1} proj_{\mathcal{S}_k}^{Q_{k+1}}\right) G_k \left(I - proj_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1}\right), \qquad \texttt{(quNac)}$$

*then $G_k$ is positive definite for $k = 0, \ldots, \rho + 1$.*

**Proof:** By induction on $k$, suppose that $G_k$ is positive definite. The first term on the right hand side of (`quNac`) can be re-written as

$$Q_{k+1} \text{proj}_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1} = Q_{k+1} \text{proj}_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1} \text{proj}_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1}.$$

In the context of Lemma 2.1, let $P = \text{proj}_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1}$, $A = Q_{k+1}$ and $B = G_k$, and by noting that $\text{Range}(P) = \text{Range}(\mathcal{S}_k)$ then $G_{k+1}$ is positive definite. ∎

We call the estimates resulting from (`quNac`) the *quasi-Newton action constrained* estimates. Different from (16) which is a rank-$3q$ update, each quNac estimate is a rank-$2q$ update. Next we prove an essential Lemma used to connect quNac methods to the BFGS and DFP methods.

From this point on, we apply (`quNac`) as a function by explicitly referring to the previous estimate and desired action $G_{k+1} = \texttt{quNac}(G_k, \mathcal{S}_k \to Q_{k+1}\mathcal{S}_k)$. In particular, in order the estimate an inverse matrix, we apply the update $H_{k+1} = \texttt{quNac}(H_k, Q_{k+1}\mathcal{S}_k \to \mathcal{S}_k)$ where the order of the action constraint has been switched. Applying the positive definite Propositions to $H_{k+1}$ is simply a matter of switching the labels of $Q_{k+1}\mathcal{S}_k$ and $\mathcal{S}_k$ in the statements and proofs.

## 2.4 Unravelling quNac into sequential rank 2 updates

Under orthogonality conditions between the columns of the sampling matrix and associated action, the rank-$2q$ `quNac` update is equivalent to sequentially applying the quNac update built from the action on the $q$ individual columns of the sampling matrix. This has already been proved for the BFGS update in [22]. We call this *unravelling* the quNac update.

For this Proposition and henceforth, we say that $V, U \in \mathbb{R}^{n \times j}, j \in \mathbb{N}$, are $A-$orthogonal, for $A \in S^n$, when $V^T A U = U^T A V = 0$.

**Proposition 2.3 (Unraveling)** *If the columns of $\mathcal{S}_k := [s_1, \ldots, s_q]$ are $Q_{k+1}-$orthogonal, then $G_{k+1} = \texttt{quNac}(G_k, \mathcal{S}_k \to Q_{k+1}\mathcal{S}_k)$ is equal to $G_k^q$ where $G_k^1 := G_k$ and*

$$G_k^{i+1} = \texttt{quNac}(G_k^i, s_i \to Q_{k+1}s_i), \quad \textit{for } i = 1, \ldots q.$$

**Proof:** Borrowing Nocedal's notation [31] for multiple BFGS updates, multiple quNac updates applied to $G_k$ to obtain $G_k^q$ is equivalent to

$$\begin{aligned} G_k^q &= (V_1 \cdots V_q)^T G_k (V_1 \cdots V_q) \\ &\quad + (V_2 \cdots V_q)^T Q_{k+1} \text{proj}_{s_1}^{Q_{k+1}} Q_{k+1} (V_2 \cdots V_q) \\ &\quad + (V_3 \cdots V_q)^T Q_{k+1} \text{proj}_{s_2}^{Q_{k+1}} Q_{k+1} (V_3 \cdots V_q) \\ &\quad + \cdots \\ &\quad + Q_{k+1} \text{proj}_{s_q}^{Q_{k+1}} Q_{k+1}, \end{aligned} \tag{18}$$

where $V_i = I - \text{proj}_{s_i}^{Q_{k+1}} Q_{k+1}$ for $i = 1, \ldots, q$. As $s_i$ and $s_j$ are $Q_{k+1}-$orthogonal for $i \neq j$,

$$\begin{aligned} V_i V_j &= (I - \text{proj}_{s_i}^{Q_{k+1}} Q_{k+1})(I - \text{proj}_{s_j}^{Q_{k+1}} Q_{k+1}) \\ &= (I - \text{proj}_{s_j}^{Q_{k+1}} Q_{k+1} - \text{proj}_{s_i}^{Q_{k+1}} Q_{k+1}) \\ &= (I - \text{proj}_{[s_j, s_i]}^{Q_{k+1}} Q_{k+1}), \end{aligned}$$

where $[s_j, s_i]$ is the column concatenation of $s_j$ and $s_i$. This applied recursively yields

$$\begin{aligned} &(V_{i+1} \cdots V_q)^T Q_{k+1} \text{proj}_{s_i}^{Q_{k+1}} Q_{k+1} (V_{i+1} \cdots V_q) \\ &\qquad = \left(I - Q_{k+1} \text{proj}_{[s_{i+1}, \ldots, s_q]}^{Q_{k+1}}\right) Q_{k+1} \text{proj}_{s_i}^{Q_{k+1}} Q_{k+1} \left(I - \text{proj}_{[s_{i+1}, \ldots, s_q]}^{Q_{k+1}} Q_{k+1}\right) \\ &\qquad = Q_{k+1} \text{proj}_{s_i}^{Q_{k+1}} Q_{k+1}. \end{aligned}$$

These observations applied to (18) reveal

$$\begin{aligned} G_k^q &= Q_{k+1} \sum_{i=1}^{q} \text{proj}_{s_i}^{Q_{k+1}} Q_{k+1} + \left(I - Q_{k+1} \text{proj}_{[s_1, \ldots, s_q]}^{Q_{k+1}}\right) G_k \left(I - \text{proj}_{[s_1, \ldots, s_q]}^{Q_{k+1}} Q_{k+1}\right) \\ &= Q_{k+1} \text{proj}_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1} + \left(I - Q_{k+1} \text{proj}_{\mathcal{S}_k}^{Q_{k+1}}\right) G_k \left(I - \text{proj}_{\mathcal{S}_k}^{Q_{k+1}} Q_{k+1}\right) \end{aligned}$$

which is the quNac update $\texttt{quNac}(G_k, \mathcal{S}_k \to Q_{k+1}\mathcal{S}_k)$. ∎

Proposition 2.3 is used to bridge quNac updates with sequentially applying Broyden family updates. Next we determine two practical quNac methods that generalize the DFP and BFGS methods.

## 3 The `inverse` and `direct` quNac methods

Based on (`quNac`), we determine two methods for estimating the Hessian matrix $\nabla f_{k+1}$ and its (pseudo-) inverse. The least change objective in the quNac framework can be justified when $f$ is twice continuously differentiable, that is, $\nabla^2 f : x \to \nabla^2 f(x)$ is a continuous matrix field.

With a given estimate $G_k \approx \nabla^2 f_k$, we define the *direct* quNac update as $G_{k+1} = \texttt{quNac}(G_k, \mathcal{S}_k \ \to \ \nabla^2 f_{k+1}\, \mathcal{S}_k)$. Positive definiteness is guaranteed by Proposition 2.2 when $G_k \succ 0$ and when $\mathcal{S}_k^T \nabla^2 f_{k+1} \mathcal{S}_k \ \succ \ 0$. Using the Woodbury formula [41], in the Appendix 8 we show that much like the DFP method, one can update the inverse when $H_k = G_k^{-1}$ exists and work solely with $H_k$ through the formula

$$H_{k+1} = H_k + \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} - H_k \nabla^2 f_{k+1} \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1} H_k \nabla^2 f_{k+1}} \nabla^2 f_{k+1} H_k. \tag{19}$$

Alternatively, we can use the `quNac` update to estimate the inverse Hessian without the need to go through the Woodbury formula. To build an estimate matrix $H_{k+1} \in S^n$ of the inverse Hessian with the appropriate action $H_{k+1} : \nabla^2 f_{k+1} \mathcal{S}_k \to \mathcal{S}_k$, we simply invert the order of the arguments $\mathcal{S}_k$ and $\nabla^2 f_{k+1} \mathcal{S}_k$ in the `quNac` function so that $H_{k+1} = \texttt{quNac}(H_k, \nabla^2 f_{k+1} \mathcal{S}_k \to \mathcal{S}_k)$. This results in the *inverse* quNac update

$$H_{k+1} = \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} + \left( I - \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} \nabla^2 f_{k+1} \right) H_k \left( I - \nabla^2 f_{k+1} \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} \right). \tag{20}$$

In this inverse perspective, $\nabla^2 f_{k+1} \mathcal{S}_k$ is the sampling matrix and $\mathcal{S}_k$ the resulting action. Positive definiteness of $H_{k+1}$ follows by Proposition 2.2 when $H_k \succ 0$ and when the product of the sampling matrix and associated action is positive definite, that is, when $\mathcal{S}_k^T \nabla^2 f_{k+1} \mathcal{S}_k \succ 0$.

The BFGS and DFP methods are instances of the inverse and direct quNac, respectively. When $\mathcal{S}_k = s \in \mathbb{R}^n$ is comprised of a single column, then the inverse (direct) quNac update is equivalent to applying a BFGS (DFP) update with the action $\nabla^2 f_{k+1} s \to s$ $\left( s \to \nabla^2 f_{k+1} s \right)$ which can be re-written as

$$\begin{aligned} H_{k+1} &= \frac{s s^T}{s^T \nabla^2 f_{k+1} s} + \left( I - \frac{s s^T \nabla^2 f_{k+1}}{s^T \nabla^2 f_{k+1} s} \right) H_k \left( I - \frac{\nabla^2 f_{k+1} s s^T}{s^T \nabla^2 f_{k+1} s} \right) \\ &= \text{proj}_s^{\nabla^2 f_{k+1}} + \left( I - \text{proj}_s^{\nabla^2 f_{k+1}} \nabla^2 f_{k+1} \right) H_k \left( I - \nabla^2 f_{k+1} \text{proj}_s^{\nabla^2 f_{k+1}} \right). \end{aligned}$$

That is, applying the BFGS and DFP update using the pair $\delta_k, \gamma_k \in \mathbb{R}^n$ is equivalent to applying the update $\texttt{quNac}(H_k, \gamma_k \to \delta_k)$ and $\texttt{quNac}(G_k, \delta_k \to \gamma_k)$, respectively. Thus we can apply Propositions 2.2 and 2.1 to show that the resulting estimate is positive definite when $H_k \succ 0$, $\gamma_k^T \delta_k > 0$ and quadratic Hereditary holds when $\{\delta_1, \ldots, \delta_k\}$ are $Q-$orthogonal where $Q$ is the constant Hessian matrix. These sufficient conditions are well known for the BFGS and DFP methods, but it is nice to see how they are derived using the same tools for quNac methods.

Furthermore, when the columns of $\mathcal{S}_k$ are $\nabla^2 f_{k+1}-$orthogonal, then according to Proposition 2.3 applying the inverse (direct) quNac update is equivalent to sequentially applying BFGS (DFP) updates built from the $i$th column of $\mathcal{S}_k$ and $\nabla^2 f_{k+1} \mathcal{S}_k$, for $i = 1, \ldots, q$. We use this observation to implement a new *parallelizable* method for applying a L-BFGS preconditioner.

We now digress from the main flow of the article to show that, much like the Broyden family, the `direct` and `inverse` quNac methods can be combined to generate a family of methods.

### 3.1 A Family of quNac methods

We can update a given $H_k$ estimate using a combination

$$H_{k+1}^{\lambda} = \lambda_k H_{k+1}^D + (1 - \lambda_k) H_{k+1}^I,$$

where $H^I_{k+1}$ and $H^D_{k+1}$ are given by the inverse (20) and direct (19) estimate, respectively, and $\lambda_k \in [0, 1]$. Manipulating the formulas for $H^D_{k+1}$ and $H^I_{k+1}$ we find

$$\begin{aligned} H^\lambda_{k+1} &= H^I_{k+1} + \lambda_k \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} \nabla^2 f_{k+1} H_k \left( I - \nabla^2 f_{k+1} \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} \right) \qquad (21) \\ &\quad + \lambda H_k \nabla^2 f_{k+1} \left( \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} - \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1} H_k \nabla^2 f_{k+1}} \nabla^2 f_{k+1} H_k \right) \\ &= H^I_{k+1} - \lambda_k V_k V_k^T, \qquad (22) \end{aligned}$$

where

$$V_k = \left( \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} \nabla^2 f_{k+1} - I \right) H_k \nabla^2 f_{k+1} \mathcal{S}_k (\mathcal{S}_k^T \nabla^2 f_{k+1} H_k \nabla^2 f_{k+1} \mathcal{S}_k)^{-1/2} \in \mathbb{R}^{n \times q},$$

thus analogously to the Broyden family, each member of the quNac family is at most a rank-$q$ matrix in distance from each other. When $H^D_{k+1}$ and $H^I_{k+1}$ are positive definite, then so is $H^\lambda_{k+1}$ as it is a positive sum of two positive definite matrices.

The resulting $H^\lambda_{k+1}$ also satisfies the action constraint as

$$\left( \lambda_k H^D_k + (1 - \lambda_k) H^I_k \right) \nabla^2 f_{k+1} \mathcal{S}_k = \lambda \mathcal{S}_k + (1 - \lambda) \mathcal{S}_k = \mathcal{S}_k. \qquad (23)$$

When the quadratic Hereditary property holds for $H^D_{k+1}$ and $H^I_{k+1}$, it also holds for $H^\lambda_{k+1}$ using the same observation as in (23) though with $\mathcal{S}_i$ for $i = 1, \ldots, k$, in the place of $\mathcal{S}_k$.

To implementing a Newton-CG method with a `quNac` preconditioner we need the details of the PCG method. Readers familiar with the PCG method can jump to the Restarting Preconditioner Lemma 4.1.

# 4 Conjugate Gradients

The conjugate gradients method, developed by Magnus Hestenes and Eduard Stiefel [24], is an iterative method for finding the solution to

$$\min_x \phi(x) := \min_x \frac{1}{2} x^T Q x - x^T b, \qquad (24)$$

where $x, b \in \mathbb{R}^n$ and $Q \in S^n$ is a positive definite matrix which guarantees that the critical point defined by

$$\nabla \phi(x) = Qx - b = 0, \qquad (25)$$

is the unique solution. With a given $x_0 \in \mathbb{R}^n$, the method iteratively finds $x_k$, the minimum of $\phi(x)$ restricted to $x_0 \oplus \mathcal{K}_k$, where $\mathcal{K}_k = \text{span}\left\{ \nabla \phi(x_0), Q \nabla \phi(x_0), \ldots, Q^{k-1} \nabla \phi(x_0) \right\}$ is the $k$th *Krylov subspace*. This construction implies that if $v \in \mathcal{K}_k$ then $Qv \in \mathcal{K}_{k+1}$. The Krylov subspaces are nested, in that $\mathcal{K}_k \subset \mathcal{K}_{k+1}$, thus each $x_{k+1}$ tends to be an improvement over the previous $x_k$. As $x_k$ is a constrained optima, the gradient $r_k := \nabla \phi(x_k)$, which is the *residual* in equation (25) at $x_k$, is in $\mathcal{K}_k^\perp$, the orthogonal complement of $\mathcal{K}_k$.

The CG method searches the Krylov spaces by using $Q-$orthogonal directions, which are also known as the conjugate directions. The first conjugate direction is set to $p_0 := -r_0$. An exact line

search is then performed with $\alpha_0 := \arg\min\{\alpha \,|\, \phi(x_0+\alpha p_0)\}$ to obtain a new iterate $x_1 = x_0+\alpha_0 p_0$. For this reason $r_1$ is orthogonal to $\mathcal{K}_1 = \text{span}\{p_0\}$. Then recursively from $x_k$, a conjugate direction in $\mathcal{K}_{k+1}$ is determined by applying the Gram-Schmidt orthogonalization process with inner product $\langle\cdot,\cdot\rangle_Q$ to $-r_k$,

$$p_k = -r_k + \frac{\langle r_k, p_{k-1}\rangle_Q}{\langle p_{k-1}, p_{k-1}\rangle_Q} p_{k-1}. \tag{26}$$

Only the component of $r_k$ in the $p_{k-1}$ direction is removed as $r_k \in \mathcal{K}_k^{\perp} \subset (Q\mathcal{K}_{k-1})^{\perp}$ which guarantees that the inner product of $r_k$ with each $Qp_0, \ldots, Qp_{k-2}$ is zero. An exact line search over $p_k$ is then performed to find $x_{k+1}$

$$x_{k+1} = x_k + \alpha_k p_k, \tag{27}$$

where $\alpha_k = -\langle r_k, p_k\rangle / \langle p_k, p_k\rangle_Q$. Finally, as $\phi(x)$ is a quadratic function, the gradient can be calculated iteratively

$$r_{k+1} = r_k + \alpha_k Q p_k. \tag{28}$$

If a preconditioner $M \in S^n$ with $M \succ 0$ is used, in other words, if an equivalent positive definite system to $M^{-1}Qx = M^{-1}b$ is solved, then the Gram-Schmidt process is applied to the sequence $M^{-1}r_k$ instead of $r_k$ resulting in

$$p_0 = -M^{-1}r_0, \tag{29}$$

$$p_k = -M^{-1}r_k + \frac{\langle M^{-1}r_k, p_{k-1}\rangle_Q}{\langle p_{k-1}, p_{k-1}\rangle_Q} p_{k-1}, \quad k > 0. \tag{30}$$

Before moving on, we need a Lemma that is fundamental in proving the quadratic Hereditary property of our forthcoming Newton-PCG implementation. The Lemma establishes sufficient conditions on the preconditioner and a new starting point such that after stopping then starting the PCG method at this new point, the PCG method continues to build $Q$-orthogonal search directions.

**Lemma 4.1 (Restarting Preconditioner)** *Let $p_0 \ldots p_{k-1}$ be a set of $Q-$orthogonal directions. Let $\bar{x}_0 \in \mathbb{R}^n$ with gradient $\nabla f(\bar{x}_0)$ such that $p_j^T \nabla f(\bar{x}_0) = 0$, for $j = 1, \ldots, k-1$. Let $M \in \mathbb{R}^n$ be a symmetric positive definite matrix such that*

$$M^{-1}Qp_j = p_j, \quad \text{for } j = 1, \ldots, k-1. \tag{31}$$

*Then by executing $t$ iterations of the PCG method on the system $Qx = b$, where $k+t+1 \leq n$, with initial point $\bar{x}_0$ and $M^{-1}$ as a preconditioner, the conjugate directions calculated, namely $\bar{p}_0, \ldots, \bar{p}_t$, are such that*

$$\{p_0 \ldots p_{k-1}, \bar{p}_0, \ldots, \bar{p}_t\},$$

*is a $Q-$orthogonal set.*

**Proof:** Let $\bar{r}_0, \ldots, \bar{r}_t$ be the residual vectors associated with the conjugate directions $\bar{p}_0, \ldots, \bar{p}_t$, where $\bar{r}_0 := \nabla f(\bar{x}_0)$. We use induction on $t$, where our induction hypothesis is that $\bar{p}_i^T Q p_j = 0$ and $\bar{r}_i^T p_j = 0$ for $1 \leq j \leq k-1$ and $0 \leq i \leq t$. For $t = 0$, as $\bar{p}_0 = -M^{-1}\bar{r}_0$,

$$\begin{aligned}\bar{p}_0^T Q p_j &= -\bar{r}_0^T M^{-1} Q p_j \quad \text{(using (31))} \\ &= -\bar{r}_0^T p_j = 0, \quad \text{for } j = 1, \ldots, k-1.\end{aligned}$$

Supposing the induction hypothesis is true for $t-1$ and all $0 \leq j \leq k-1$, using (28) to calculate the next residual $\bar{r}_t$, then by induction

$$\begin{aligned}\bar{r}_t^T p_j &= \bar{r}_{t-1}^T p_j - \frac{\langle \bar{r}_{t-1}, \bar{p}_{t-1} \rangle}{\langle \bar{p}_{t-1}, \bar{p}_{t-1} \rangle_Q} \bar{p}_{t-1}^T Q p_j \\ &= \bar{r}_{t-1}^T p_j \\ &= 0.\end{aligned}$$

Using (30) to substitute $\bar{p}_t$

$$\begin{aligned}\bar{p}_t^T Q p_j &= -\bar{r}_t^T M^{-1} Q p_j + \frac{\langle M^{-1}\bar{r}_t, \bar{p}_{t-1} \rangle_Q}{\langle \bar{p}_{t-1}, \bar{p}_{t-1} \rangle_Q} \bar{p}_{t-1}^T Q p_j \\ &= -\bar{r}_t^T M^{-1} Q p_j \quad \text{(applying (31))} \\ &= -\bar{r}_t^T p_j \\ &= 0, \quad \text{for } j = 1, \ldots, k-1. \quad \blacksquare\end{aligned}$$

We refer to $\bar{x}_0$ and $M^{-1}$ of Lemma 4.1 as a restart point and restarting preconditioner, respectively.

For further reading on the Preconditioned Conjugate Gradients ($PCG$) method see [38] for a pedagogic explanation and [21] for a description that uses oblique projections.

# 5 Implementing a Newton-PCG quNac method

We use the `inverse` quNac formula (20) to update a preconditioner within a Newton-PCG method for finding local minima of $f \in C^2(\mathbb{R}^n)$, where $f$ is possibly non-convex, see Algorithm 5.1.

The inputs are an initial point $x_0$, initial estimate $H_0$ and `max_q`; the maximum number of columns allowed in $\mathcal{S}_k$ at each iteration $k$. In the first iteration, $k = 0$, the search direction $d_0 = -H_0 \nabla f_0$ is used. To determine $x_{k+1}$, a line search is used that first checks to see if $a_k = 1$ meets the line search criteria. In our implementation we use a sufficient descent criteria

$$f(x_k + a_k d_k) - f(x_k) \leq c_1 \alpha_k d_k^T \nabla f_k, \tag{32}$$

with $c = 10^{-4}$.

The PCG method Algorithm 5.2 is then called with $H_k$ as a preconditioner to approximately solve $\nabla^2 f_{k+1} d_{k+1} = -\nabla f_{k+1}$ with the number of iterations capped by `max_q`. Further limiting the number of PCG iterations is a tolerance

$$\texttt{PCG\_tol} = \min\left\{0.01, \|\nabla f(x_{k+1})\|^{1/2}\right\},$$

which corresponds to the "super-linear" choice in inexact Newton methods [11]. The conjugate directions calculated during the PCG execution, which we denote by $[p_{q(k)}, \ldots, p_{q(k+1)-1}]$ henceforth, are saved to form the columns of $\mathcal{S}_k$. Specifically, the columns of $\mathcal{S}_k$ are the $\nabla^2 f_{k+1}-$normalized conjugate directions

$$\mathcal{S}_k = \left[ \frac{p_{q(k)}}{\|p_{q(k)}\|_{\nabla^2 f_{k+1}}}, \ldots, \frac{p_{q(k+1)-1}}{\|p_{q(k+1)-1}\|_{\nabla^2 f_{k+1}}} \right]. \tag{33}$$

This normalization is done to simplify calculations, as with this choice $\text{proj}_{\mathcal{S}_k}^{\nabla^2 f_k} = \mathcal{S}_k \mathcal{S}_k^T$. So that the resulting estimate is positive definite, we only collect conjugate directions so long as negative curvature is not encountered, line 7 of Algorithm 5.2. This ensures that $\mathcal{S}_k^T \nabla^2 f_{k+1} \mathcal{S}_k \succ 0$. There is a safeguard for non-convex functions on line 8 of Algorithm 5.2. If negative curvature is encountered on the first PCG iteration, then the first conjugate direction $p_0 = -H_k \nabla f_{k+1}$ is returned as the search direction. Before moving onto the next iteration, the estimate matrix is updated by either a full or limited memory inverse quNac (20) update, detailed in Sections 5.1 and 5.2, respectively.

In line 6 of Algorithm 5.2, we need to calculate a Hessian-vector product. This can be done efficiently through reverse AD (*Automatic Differentiation*) [9]. Naturally there also exist problems and applications where fast Hessian-vector products are readily available, such as Fast-Fourier transform, Neural Networks [34] or obvious structure prevailing in the Hessian matrix. As a final option, the user would be required to write an efficient subroutine for calculating Hessian-vector products.

**Algorithm 5.1:** Newton-PCG quNac

**Input**: $H_0, x_0 \in \mathbb{R}$, `max_q` $\in \mathbb{N}$.

1 $k = 0, d_0 = -H_0 \nabla f_0$

2 **while** $|\nabla f_k|/|\nabla f_0| > \epsilon$ *or* $|\nabla f_k| > \epsilon$ **do**

3 Determine $a_k$ through a line-search on $\{a \,|\, x_k + a d_k\}$ starting with $a_k = 1$

4 $x_{k+1} = x_k + a_k d_k$

5 $[\mathcal{S}, \nabla^2 f_{k+1}\mathcal{S}, d_k] =$`PCG` $(\nabla^2 f_{k+1}, H_k, x_{k+1}, \max\_q,$ `PCG_tol`$)$

6 $H_{k+1} =$ `quNac`$(H_k, \nabla^2 f_{k+1}\mathcal{S} \to \mathcal{S})$, using Algorithm 5.3

7 $k = k + 1$

**Output**: $x_k$.

## 5.1 Full memory Inverse quNac

Both the limited and full memory variants of the inverse quNac update have been implemented in a way that promotes parallel linear algebra through Matrix multiplication. To derive these two variants, let $\underline{\mathcal{S}}_k = \nabla^2 f_{k+1}\mathcal{S}_k$ be the $n \times q$ matrix stored from executing PCG method in Algorithm 5.2. With the normalization (33) of $\mathcal{S}_k$, the inverse quNac update can be calculated by

$$\begin{aligned} E_k &= \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} + \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} \nabla^2 f_{k+1} H_k (\nabla^2 f_{k+1} \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} - I) - H_k \nabla^2 f_{k+1} \text{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} \\ &= \mathcal{S}_k \mathcal{S}_k^T + \mathcal{S}_k \underline{\mathcal{S}}_k^T H_k (\underline{\mathcal{S}}_k \mathcal{S}^T - I) - H_k \underline{\mathcal{S}}_k \mathcal{S}_k^T \\ &= \mathcal{S}_k \left( I_{p\times p} + \underline{\mathcal{S}}_k^T H_k \underline{\mathcal{S}}_k \right) \mathcal{S}_k^T - H_k \underline{\mathcal{S}}_k \mathcal{S}_k^T - \mathcal{S}_k \underline{\mathcal{S}}_k^T H_k. \end{aligned}$$

**Algorithm 5.2:** PCG$(A, M^{-1}, y_0, \max_q, \texttt{PCG_tol})$

$r_0 = \nabla f(y_0)$
$z_0 = M^{-1}(r_0)$
$p_0 = -z_0$
$y_0 = 0$
**for** $i = 0, \ldots, max\_q - 1$ **do**
  $c_i = \langle Ap_i, p_i \rangle$
  **if** $c_i \leq 0$ **then**
    **if** $i > 0$ **then break** **else** $y_0 = p_0$
  $\alpha_i = \dfrac{\langle r_i, z_i \rangle}{c_i}$
  $y_{i+1} = y_i + \alpha_i p_i$
  $r_{i+1} = r_i + \alpha_i A p_i$
  $z_{i+1} = M^{-1} r_{i+1}$
  $\beta_i = \dfrac{\langle r_{i+1}, z_{i+1} \rangle}{\langle r_i, z_i \rangle}$
  $p_{i+1} = -z_{i+1} + \beta_i p_i$
  **if** $\|r_{i+1}\|/\|r_0\| <$PCG_tol **then**
    $q = i + 1$
    **break**
$q = \min\{\max\_q, i\}$

**Output**: $\mathcal{S} = \left[c_0^{-1/2} p_0, \ldots, c_0^{-1/2} p_{q-1}\right], A\mathcal{S}, y_q.$

This has been coded in Algorithm 5.3 and costs $O(n^2 q)$ operations. Line 1 is the bottleneck as it involves a multiplication of a possibly dense $n \times n$ matrix with a $n \times q$ matrix. The cost of sequentially applying $q$ BFGS updates is also $O(n^2 q)$, the important difference is that Algorithm 5.3 can greatly benefit from multithreading and parallel linear algebra, while there is no obvious parallelism in applying BFGS updates. In fact, if $q$ processors are available in a shared memory architecture, then the wall clock time of Algorithm 5.3 is $O(n^2)$ plus additional overheads of the parallel paradigm (such as creating and destroying threads).

**Algorithm 5.3:** Scaled Inverse $\texttt{quNac}(H, \underline{\mathcal{S}} \to \mathcal{S})$ update

**Input**: $H \in \mathbb{R}^{n \times n}$ and $\mathcal{S}, \underline{\mathcal{S}} \in \mathbb{R}^{n \times q}$

1 $\underline{H} = H\underline{\mathcal{S}}$

2 $\overline{H} = \underline{H}\mathcal{S}^T$

3 $E = \mathcal{S}\left(I_{p\times p} + \underline{\mathcal{S}}^T \underline{H}\right)\mathcal{S}^T - \overline{H} - \overline{H}^T$

**Output**: $E$.

The next Corollary shows that when Algorithm 5.1 uses `quNac` updates, the resulting preconditioners satisfy the quadratic Hereditary property. Thus when Algorithm 5.1 is applied to convex quadratic problems, the method terminates after a total of $n$ inner steps of the PCG method.

Due to this following Corollary, we chose to update the preconditioner with all available conjugate directions. This is in contrast with the strategies mentioned in [28], where the last conjugate directions or a uniform sampling of conjugate directions are used to perform L-BFGS updates.

**Corollary 5.1 (Quadratic Hereditary for quNac Preconditioner)** *Assume Algorithm 5.1 is applied to a convex quadratic function $\phi(x)$ with $\nabla^2\phi(x) \equiv Q \in \mathbb{R}^{n\times n}$, and consider its kth major iteration, $k \geq 1$. Then $H_{k+1}Q\mathcal{S}_i = \mathcal{S}_i$ for $i = 0, \ldots, k$.*

**Proof:** We prove this using the Restarting Preconditioner Lemma 4.1 to show that $\{p_0, \ldots, p_{q(k+1)-1}\}$ is a $Q-$orthogonal set, then apply Corollary 2.1 and the comment after Corollary 2.1 to prove quadratic hereditary. The proof is by induction where our hypothesis is that the set $\{p_0, \ldots, p_{q(k)-1}\}$ is a $Q-$orthogonal set and $p_j^T \nabla\phi(x_k) = 0$ for all $0 \leq j \leq q(k-1) - 1$.

The base case of our induction is $k = 2$. The set of vectors $\{p_0, \ldots, p_{q(1)-1}\}$ calculated by the first PCG call are $Q-$orthogonal by construction. At iteration $k = 1$, as $x_1 + d_1$ is the minimum of the quadratic $\phi(x)$ over $x_1 \oplus \mathcal{K}_{q(1)-1}$, the step parameter $a_1 = 1$ is accepted. Therefore $x_2 = x_1 + d_1$, $\nabla\phi(x_2) \in \mathcal{K}_{q(1)-1}^{\perp}$ and $p_j^T \nabla\phi(x_2) = 0$, for $j = 0, \ldots q(1) - 1$. This proves, together with the action constraint $H_1 Q p_j = p_j$ for $j = 0, \ldots, q(1) - 1$, that $x_2$ and $H_1$ are a restarting point and a restarting preconditioner, respectively, and by Lemma 4.1 the set $\{p_0, \ldots, p_{q(1)-1}, p_{q(1)}, \ldots, p_{q(2)-1}\}$ is $Q-$orthogonal. This concludes the proof of our induction hypothesis for $k = 2$.

Suppose that $p_j^T \nabla\phi(x_k) = 0$ for all $0 \leq j \leq q(k-1)-1$ and $\{p_0, \ldots, p_{q(k)-1}\}$ are $Q-$orthogonal. This $Q-$orthogonality guarantees by Corollary 2.1 that $H_k$ satisfies the hereditary property $QH_k p_i = p_i$ for $i = 0, \ldots, q(k) - 1$.

At the $k$th iteration $a_k = 1$ is accepted as $x_k + d_k$ is the minimum of $x_k \oplus \text{span}\{p_{q(k-1)}, \ldots, p_{q(k)-1}\}$,

thus $p_j^T \nabla\phi(x_{k+1}) = 0$ for $q(k-1) \le j < q(k)$. For $j < q(k-1)$ we have

$$\begin{aligned} p_j^T \nabla\phi(x_{k+1}) &= p_j^T \left(\nabla\phi(x_k) + Qd_k\right) \\ &= p_j^T \left(\nabla\phi(x_k) + Q\left(\sum_{i=q(k-1)}^{q(k)-1} \alpha_i p_i\right)\right) \\ &= p_j^T \nabla\phi(x_k) + \sum_{i=q(k-1)}^{q(k)-1} \alpha_i \alpha_j p_j^T Q p_i \quad \text{(applying the induction hypothesis)} \\ &= 0 + 0. \end{aligned}$$

Thus $x_{k+1}$ and $H_k$ are a restarting point and a restarting preconditioner, respectively, and by Lemma 4.1 the vectors $\{p_0 \dots p_{q(k+1)-1}\}$ are $Q-$orthogonal, which concludes the induction. Finally, the columns of the sampling matrices are scalar multiples of the conjugate directions, thus Corollary 2.1 and the comment that follow it guarantees the quadratic hereditary of $H_{k+1}$ is Algorithm 5.1. ∎

### 5.2 Limited memory `quNac`

To implement a limited memory variant of the `inverse` quNac update (20), instead of updating $H_k$, in line 6 of Algorithm 5.1, we initiate $H_k = H_0^{k+1}$ which is a user specified initial estimate approximation (or simply the identity in the lack there of). Both $H_0^{k+1}$ and $H_{k+1}$ must be coded as operators acting on vectors in $\mathbb{R}^n$ instead of explicit matrices. In Algorithm 5.4 we show how to execute the operation $v \to H_0^{k+1}(v) + E_k(v)$ without the need to store a matrix. Let $\mathcal{S}_k$ and $\underline{\mathcal{S}}_k = \nabla^2 f_{k+1}\mathcal{S}_k$ be the $n \times q$ matrices stored from the previous PCG call. Then to calculate $H_{k+1}v$ we have

$$\begin{aligned} (H_0^{k+1} + E_k)v &= \mathrm{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} + (I - \mathrm{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}} \nabla^2 f_{k+1}) H_k (I - \nabla^2 f_{k+1} \mathrm{proj}_{\mathcal{S}_k}^{\nabla^2 f_{k+1}}) \\ &= \mathcal{S}_k \mathcal{S}_k^T v + (I - \mathcal{S}_k \underline{\mathcal{S}}_k^T) H_0^{k+1} (I - \underline{\mathcal{S}}_k \mathcal{S}_k^T) v \\ &= (H_0^{k+1}(v - \underline{\mathcal{S}}_k(\mathcal{S}_k^T v)) + \mathcal{S}_k \left( (\mathcal{S}_k^T v) - \underline{\mathcal{S}}_k^T \left( H_0^{k+1}(v - \underline{\mathcal{S}}_k(\mathcal{S}_k^T v)) \right) \right), \end{aligned} \tag{34}$$

which can be calculated efficiently by Algorithm 5.4. As the columns of $\mathcal{S}_k$ are $\nabla^2 f_{k+1}-$orthogonal, Proposition 2.3 proves that Algorithm 5.4 has the same result, in exact precision, as applying the L-BFGS two-loop recursion [31] to the columns of $\mathcal{S}_k$ and $\underline{\mathcal{S}}_k$. To compare the two methods for applying a preconditioner operator, we have placed the L-BFGS two-loop recursion and LquNac side-by-side in Figure 1. The only difference between them is that $\boxed{v}$ and $\boxed{r}$ in Algorithm 5.5 are replaced by a new variable $z$ in Algorithm 5.4. This small change removes the dependency between the two lines in each `for` loop in Algorithm 5.5 so that the loops can be calculated as matrix-vector products instead. Matrix-vector multiplications can be easily sped up through multithreading and shared memory parallelism, while the two `for` loops in Algorithm 5.5 are essentially sequential.

As of MATLAB version 7.4 (R2007a), MATLAB automatically multithreads matrix-vector multiplication, and tests on our quad-core Desktop comparing the time taken to perform a L-BFGS

Figure 1: Comparing the L-BFGS two-loop recursion with the parallel LquNac.

**Input** $H_0 : \mathbb{R}^n \to \mathbb{R}^n, \mathcal{S} = [s_1, \ldots, s_q], \underline{\mathcal{S}} = \left[\underline{s}_1 \ldots \underline{s}_q\right]$ and $v \in \mathbb{R}^n$.

**Algorithm 5.4:** LquNac

$v^{\mathcal{S}} \leftarrow \mathcal{S}^T v$;
$z \leftarrow v - \underline{\mathcal{S}} v^{\mathcal{S}}$;
$r \leftarrow H_0(z)$;

$r^{\underline{\mathcal{S}}} \leftarrow \underline{\mathcal{S}}^T r$;
$z \leftarrow r + \mathcal{S}(v^{\mathcal{S}} - r^{\underline{\mathcal{S}}})$ ;
**Output**: $z$

**Algorithm 5.5:** two-loop recursion

... **for** $i = 1, \ldots, q$ **do**
  $v_i^{\mathcal{S}} \leftarrow s_i^T v$;
  $\boxed{v} \leftarrow v - v_i^{\mathcal{S}} \underline{d}_i$;
$r \leftarrow H_0(v)$;
**for** $i = q, \ldots, 1$ **do**
  $r^{\underline{\mathcal{S}}} \leftarrow \underline{d}_i^T r$;
  $\boxed{r} \leftarrow r + s_i(v_i^{\mathcal{S}} - r^{\underline{\mathcal{S}}})$;
**Output**: $r$

two-loop recursion as compared to the LquNac update revealed that the speed-up can be more than four fold when there is sufficient number of columns in $\mathcal{S}_k$ and $\underline{\mathcal{S}}_k$, see Figure 2. This speed is specially important as applying this L-BFGS preconditioner is the bottle-neck in the PCG iteration. There are a number of outliers in Figure 2 that are difficult to investigate as multithreading is performed implicitly. To have finer control and better exploit this parallelism an explicit parallel paradigm needs to be implemented, something we leave for future work. Though we only consider this limited memory implementation that uses conjugate directions from the previous iteration, certainly other implementations are possible, for instance, by retaining conjugate directions from other iterations.

# 6 Numerical Tests

In our tests we compare five methods. The first two methods are the full and limited memory `inverse` quNac update detailed in Algorithms 5.1. We have labelled the two quNac methods by InverseQuNac and InverseLQuNac, when the full memory variant in Algorithm 5.3 and the limited variant in Algorithm 5.4 are used to update the estimate, respectively. The third method is Newton_CG implemented according to Algorithm 6.1 of [32] though with an additional maximum number of CG iterations set to the dimension $n$ of the problem. The last two approaches are the BFGS and L-BFGS [31] methods. To compare the methods, we embed them in the same line search framework with a sufficient descent criteria (32) that initially checks if $a_k = 1$ can be accepted. Though a line search that guarantees the Wolfe conditions is often advised for quasi-Newton methods, we found this to be inefficient when applied to non-convex functions, as an almost exhaustive search for correct parameter $a_k$ would often occur. The initial Hessian approximation was set to

$$H_0 = \frac{\nabla f_0^T \nabla f_0}{\nabla f_0^T \nabla^2 f_0 \nabla f_0} I.$$

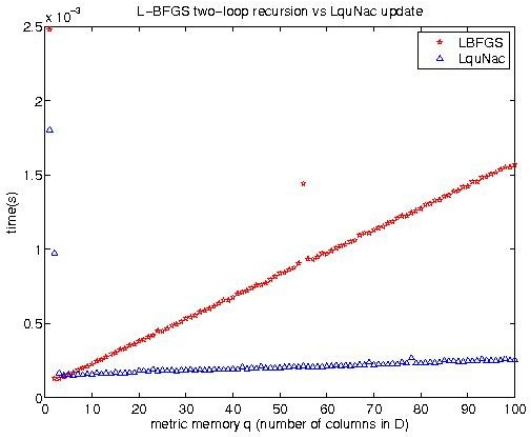



Figure 2: Time taken by Applying the L-BFGS two-loop recursion in Algorithm 5.5 with the LquNac update 5.4 where $\mathcal{S}, \underline{\mathcal{S}} \in \mathbb{R}^{500\times q}$ is randomly generated and $q$ is increased from 1 to 100 and $H_0 = I$.

In all the limited memory methods the maximum memory, `max_q` in the `quNac` methods, was set to 20.

Our MATLAB implementation "quNac" can be downloaded from the Edinburgh Research Group in Optimization website: `http://www.maths.ed.ac.uk/ERGO`. In this package one can test different line search criteria, including Wolfe-conditions, and different initial Hessian $H_0$ approximations.

We have run tests on a Desktop with 64bit quad-core Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz with 6MB cache size with a Scientific Linux release 6.4 (Carbon) operating system.

### 6.1 Linear SVM with logistic loss

Our first set of tests consists of convex Support Vector Machine (SVM) problems. SVMs have become a widely successful machine learning method for classification, and thanks to Chih-Chung Chang and Chih-Jen Lin LIBSVM collection [8], have readily available data sets. We have selected all data sets for binary classification with less than or equal to $50'000$ features (dimensions).

The linear binary SVM problem consists of finding a separating hyperplane $f_w(x) = \langle w, x\rangle$ with $w \in \mathbb{R}^n$ that is able to predict the classification of $x \in X \subset \mathbb{R}^n$, namely, $f_w(x) > 0$ and $f_w(x) \leq 0$ for the first and second class, respectively. To this end, known data pairs $(x^i, y_i)$ are collected where $x^i \in \mathbb{R}^n$ are *feature* vectors and $y_i \in \{-1, 1\}$ are *labels*, where $y_i$ indicates the class of $x^i$ for $i = 1, \ldots, m$. The linear classifier $w$ is then selected based on these data pairs by minimizing a loss function, where a popular choice [42] is the logistic loss function

$$L_w(y, X) = \sum_{i=1}^{m} \ln\left(1 + \exp(-y_i \left\langle x^i, w\right\rangle)\right).$$

We use one of two regularizers, the $\ell_2$ norm

$$R_2(w) = \|w\|_2^2,$$

or the *pseudo-Huber* norm

$$R_\mu(w) = \mu \sum_{i=1}^{n} \left( \sqrt{1 + \frac{x_i^2}{\mu^2}} - 1 \right),$$

where $\mu < 1$. The pseudo-Huber norm is an approximation to the $\ell_1$ norm as $\mu \to 0$, and has been shown to be successful in promoting sparsity in convex regularized problems [15]. The resulting unconstrained optimization problem is given by

$$\min_w L_w(y, X) + \lambda R_\mu(w),$$

where $\lambda$ is the regularizer parameter and has been set to $\lambda = 1$ in all our tests. Our interest was in encountering the unique solution to these convex problems thus we solved the SVM problem with a precision of $\epsilon = 10^{-7}$. We found through sampling a number of the problems that when increasing the precision, the solution would become increasingly sparse up to approximately $\epsilon = 10^{-7}$. Though optimizing to a high tolerance raises the question of over-fitting, this is not an issue here as the number of data points far exceeds the number of unknowns features, with the exception of the problem `colon-cancer` (62 data points and 2000 features) and `duke breast cancer` (44 data points and 7129 features).

In Tables 1 and 2 we have the run times of each method to reach the unique solution with a $\ell_2$ and pseudo-Huber regularizer, respectively. In each table, "ss" represents "small step", in that the method takes steps smaller than $\epsilon^2 = 10^{-14}$ before reaching the solution. While "TO" represents "Timeout" in that the method exceeded the maximum time allowed, which we set to 10min. Each row corresponds to a problem and the highlighted cells in the row indicate the smallest run time among all methods, while the boxed cell is the fastest among the limited memory methods. The last rows contain the standard deviation and average for each method across all solved problems, though as each method failed to solve a number of problems, these statistics have to be interpreted with care.

On the $\ell_2$ and pseudo-Huber regularized problems, InverseQuNac was the fastest method on most of the problems. Among the limited memory implementations, when tested on the $\ell_2$ regularized problems of Table 1, Newton-CG was the fastest on 23, InverseLQuNac was the fastest on 5 and L-BFGS was the fastest on 16 of the 44 problems tested. Though InverseLQuNac was the most robust, failing to converge on only one problem and with the lowest standard deviation and average. For the pseudo-Huber regularized problems of Table 2 the Newton-CG, InverseLQuNac and L-BFGS had the smallest run time on 11, 12 and 20 of the total 44 problems, respectively. The InverseLQuNac was the robust out of the limited memory methods, failing only to converge on 3 problems, while Newton-CG and L-BFGS failed on 8 and 6 problems, respectively.

With the pseudo-Huber regularizer, as the sparse solution is approached, the Hessian becomes ill-conditioned [15]. This affected the stability of Newton_CG method. The InverseQuNac and InverseLQuNac seemed to be the least affected by this ill-conditioning.

To appraise the rate of convergence of each method, in Figure 3 we have plotted the evolution of the error through time for each method applied to the `epsilon_normalized` problem. The `epsilon_normalized` problem is the most challenging of our SVM problems. Originating from the

| | # features | # data | InverseQuNac | inverseLQuNac | Newton_CG | BFGS | LBFGS |
|---|---|---|---|---|---|---|---|
| problem | | | Time(s) | Time(s) | Time(s) | Time(s) | Time(s) |
| a1a | 119 | 1605 | 0.90 | 0.22 | 0.17 | 1.74 | 0.38 |
| a2a | 119 | 2265 | 0.14 | 0.24 | 0.19 | 2.07 | 0.48 |
| a3a | 122 | 3185 | 0.16 | 0.31 | 0.27 | 2.69 | 0.58 |
| a4a | 122 | 4781 | 0.18 | 0.43 | 0.33 | 3.12 | 0.90 |
| a5a | 122 | 6414 | 0.25 | 0.52 | 0.45 | 4.20 | 1.09 |
| a6a | 122 | 11220 | 0.41 | 0.87 | 0.72 | 6.58 | 2.10 |
| a7a | 122 | 16100 | 0.60 | 1.32 | 1.23 | 9.71 | 3.49 |
| a8a | 123 | 22696 | 0.86 | 2.57 | 2.00 | 14.36 | 5.56 |
| a9a | 123 | 32561 | 1.31 | 4.13 | 3.46 | 21.89 | 9.48 |
| australian | 14 | 690 | 0.08 | 0.14 | 0.10 | 0.75 | 1.00 |
| australiansc | 14 | 690 | 0.05 | 0.07 | 0.06 | 0.21 | 0.12 |
| breast-cancer | 10 | 683 | 0.02 | 0.02 | 0.02 | 0.01 | 0.05 |
| breast-cancersc | 10 | 683 | 0.12 | 0.17 | 0.15 | 0.20 | 0.08 |
| cod-rna | 8 | 59535 | 0.91 | 1.63 | 1.99 | 8.09 | 8.73 |
| cod-rna.r | 8 | 157413 | 2.80 | 4.44 | 4.66 | 20.17 | 16.26 |
| colon-cancer | 2000 | 62 | 1.65 | 0.24 | 0.26 | 42.68 | 0.23 |
| covtype.binary | 54 | 581012 | 10.38 | 16.36 | 20.56 | 2.24 | 9.70 |
| covtype.binarysc | 54 | 581012 | 12.22 | 19.83 | 19.56 | 35.45 | 9.25 |
| diabetes | 8 | 768 | 0.03 | 0.04 | 0.32 | 0.20 | 0.18 |
| diabetessc | 8 | 768 | 0.03 | 0.04 | 0.04 | 0.13 | 0.05 |
| fourclass | 2 | 862 | 0.02 | 0.03 | 0.02 | 0.04 | 0.03 |
| fourclasssc | 2 | 862 | 0.02 | 0.02 | 0.02 | 0.03 | 0.02 |
| german.numer | 24 | 1000 | 0.06 | 0.12 | 0.12 | 0.99 | 2.31 |
| german.numersc | 24 | 1000 | 0.04 | 0.07 | 0.06 | 0.40 | 0.13 |
| gisettesc | 5000 | 6000 | 84.31 | 146.27 | 214.69 | TO | 161.39 |
| heart | 13 | 270 | 0.07 | 0.08 | 0.06 | 0.51 | 168.18 |
| heartsc | 13 | 270 | 0.02 | 0.04 | 0.04 | 0.15 | 0.06 |
| ionospheresc | 34 | 351 | 0.04 | 0.07 | 0.06 | 0.34 | 0.13 |
| liver-disorders | 6 | 345 | 0.05 | 0.07 | 0.06 | 0.08 | 0.05 |
| liver-disorderssc | 6 | 345 | 0.04 | 0.07 | 0.06 | 0.11 | 0.03 |
| mushrooms | 112 | 8124 | 0.18 | 0.24 | 0.24 | 0.76 | 0.17 |
| sonarsc | 60 | 208 | 0.04 | 0.08 | 0.07 | 0.25 | 0.61 |
| splice | 60 | 1000 | 0.05 | 0.09 | 0.09 | 0.46 | ss |
| splicesc | 60 | 1000 | 0.04 | 0.06 | 0.06 | 0.13 | 0.06 |
| svmguide1 | 4 | 3089 | TO | TO | TO | 0.09 | 0.10 |
| svmguide3 | 22 | 1243 | 0.04 | 0.07 | 0.06 | 0.40 | 0.21 |
| w1a | 300 | 2477 | 0.16 | 0.20 | 0.14 | 1.79 | 0.13 |
| w2a | 300 | 3470 | 0.17 | 0.25 | 0.20 | 2.28 | 0.17 |
| w3a | 300 | 4912 | 0.21 | 0.28 | 0.27 | 2.47 | 0.24 |
| w4a | 300 | 7366 | 0.25 | 0.37 | 0.34 | 3.14 | 0.32 |
| w5a | 300 | 9888 | 0.29 | 0.48 | 0.46 | 3.76 | 0.41 |
| w6a | 300 | 17188 | 0.54 | 0.89 | 0.77 | 5.87 | 0.73 |
| w7a | 300 | 24692 | 0.78 | 1.28 | 1.44 | 8.75 | 1.12 |
| w8a | 300 | 49749 | 1.73 | 3.10 | 3.50 | 19.74 | 2.73 |
| standard deviation | | | 12.94 | 22.42 | 32.78 | 9.44 | 34.89 |
| average | | | 2.84 | 4.83 | 6.50 | 5.33 | 9.51 |

Table 1: Binary classification with $\ell_2$ regularizer and $\epsilon = 10^{-7}$ and memory= 20. TO = TimeOut and ss = small step. The highlighted cells contain the fastest run time, while the boxed cells contain the fastest run time among the limited memory implementations

| | # features | # data | InverseQuNac | inverseLQuNac | Newton_CG | BFGS | LBFGS |
|---|---|---|---|---|---|---|---|
| problem | | | Time(s) | Time(s) | Time(s) | Time(s) | Time(s) |
| a1a | 119 | 1605 | 3.10 | 15.98 | 33.44 | 6.38 | ss |
| a2a | 119 | 2265 | 2.89 | 12.34 | 54.95 | 6.41 | ss |
| a3a | 122 | 3185 | 4.22 | 13.60 | 119.53 | 6.47 | 7.02 |
| a4a | 122 | 4781 | 4.03 | 38.26 | 176.66 | 9.05 | 6.46 |
| a5a | 122 | 6414 | 3.96 | 16.34 | 77.39 | 11.20 | 7.43 |
| a6a | 122 | 11220 | 6.45 | 18.89 | 118.58 | 17.13 | 10.00 |
| a7a | 122 | 16100 | 7.82 | 25.54 | 188.65 | 18.51 | 18.45 |
| a8a | 123 | 22696 | 8.26 | 20.89 | TO | 25.02 | 30.30 |
| a9a | 123 | 32561 | 12.14 | 27.44 | TO | 34.28 | 16.68 |
| australian | 14 | 690 | 0.10 | 0.14 | 0.12 | 0.80 | 0.93 |
| australiansc | 14 | 690 | 0.04 | 0.07 | 0.07 | 0.42 | 0.15 |
| breast-cancer | 10 | 683 | 0.02 | 0.02 | 0.02 | 0.01 | 0.05 |
| breast-cancersc | 10 | 683 | 0.36 | 0.94 | 1.61 | 0.40 | 0.23 |
| cod-rna | 8 | 59535 | 0.99 | 1.87 | 3.41 | 7.51 | 7.21 |
| cod-rna.r | 8 | 157413 | 2.35 | 3.96 | 5.05 | 17.08 | 13.44 |
| colon-cancer | 2000 | 62 | 58.73 | 26.77 | 319.28 | 261.38 | 436.45 |
| covtype.binary | 54 | 581012 | 9.24 | 14.40 | 18.32 | 1.95 | 8.57 |
| covtype.binarysc | 54 | 581012 | 563.51 | TO | TO | TO | 210.94 |
| diabetes | 8 | 768 | 0.04 | 0.05 | 0.36 | 0.21 | 0.19 |
| diabetessc | 8 | 768 | 0.06 | 0.10 | 0.12 | 0.25 | 0.12 |
| fourclass | 2 | 862 | 0.03 | 0.03 | 0.02 | 0.03 | 0.03 |
| fourclasssc | 2 | 862 | 0.02 | 0.03 | 0.03 | 0.05 | 0.03 |
| german.numer | 24 | 1000 | 0.08 | 0.18 | 0.16 | 1.02 | 2.48 |
| german.numersc | 24 | 1000 | 0.10 | 0.16 | 0.15 | 0.64 | 0.23 |
| gisettesc | 5000 | 6000 | TO | TO | TO | TO | TO |
| heart | 13 | 270 | 0.07 | 0.09 | 0.08 | 0.57 | 0.82 |
| heartsc | 13 | 270 | 0.08 | 0.16 | 0.15 | 0.36 | 0.19 |
| ionospheresc | 34 | 351 | 0.26 | 0.67 | 2.29 | 1.10 | ss |
| liver-disorders | 6 | 345 | 0.16 | 0.52 | 0.64 | 0.32 | 0.15 |
| liver-disorderssc | 6 | 345 | 0.20 | 1.50 | 0.88 | 0.33 | 0.11 |
| mushrooms | 112 | 8124 | 11.88 | 27.17 | 284.39 | 11.24 | 5.36 |
| sonarsc | 60 | 208 | 0.80 | 4.54 | 5.94 | ss | ss |
| splice | 60 | 1000 | 0.13 | 0.24 | 0.18 | 0.62 | 0.39 |
| splicesc | 60 | 1000 | 0.11 | 0.20 | 0.18 | 0.47 | 0.18 |
| svmguide1 | 4 | 3089 | TO | TO | TO | 0.42 | 0.17 |
| svmguide3 | 22 | 1243 | 0.78 | 127.34 | 5.86 | 1.79 | ss |
| w1a | 300 | 2477 | 9.62 | 30.97 | 469.14 | 23.72 | 46.92 |
| w2a | 300 | 3470 | 10.30 | 26.77 | 236.00 | 26.79 | 24.87 |
| w3a | 300 | 4912 | 15.43 | 52.27 | 458.37 | 32.16 | 25.74 |
| w4a | 300 | 7366 | 18.99 | 58.65 | 189.74 | 42.56 | 47.68 |
| w5a | 300 | 9888 | 23.30 | 44.99 | TO | 32.60 | 32.83 |
| w6a | 300 | 17188 | 23.28 | 38.59 | 355.64 | 48.79 | 46.40 |
| w7a | 300 | 24692 | 28.32 | 82.81 | TO | 81.88 | 61.41 |
| w8a | 300 | 49749 | 61.45 | 124.26 | TO | 147.90 | 74.72 |
| standard deviation | | | 86.82 | 30.85 | 137.86 | 47.13 | 77.27 |
| average | | | 21.28 | 20.97 | 86.87 | 21.46 | 30.14 |

Table 2: Binary classification with pseudo-Huber regularizer and $\epsilon = 10^{-7}$ and memory= 20. TO = TimeOut and ss = small step. The highlighted cells contain the fastest run time, while the boxed cells contain the fastest run time among the limited memory implementations

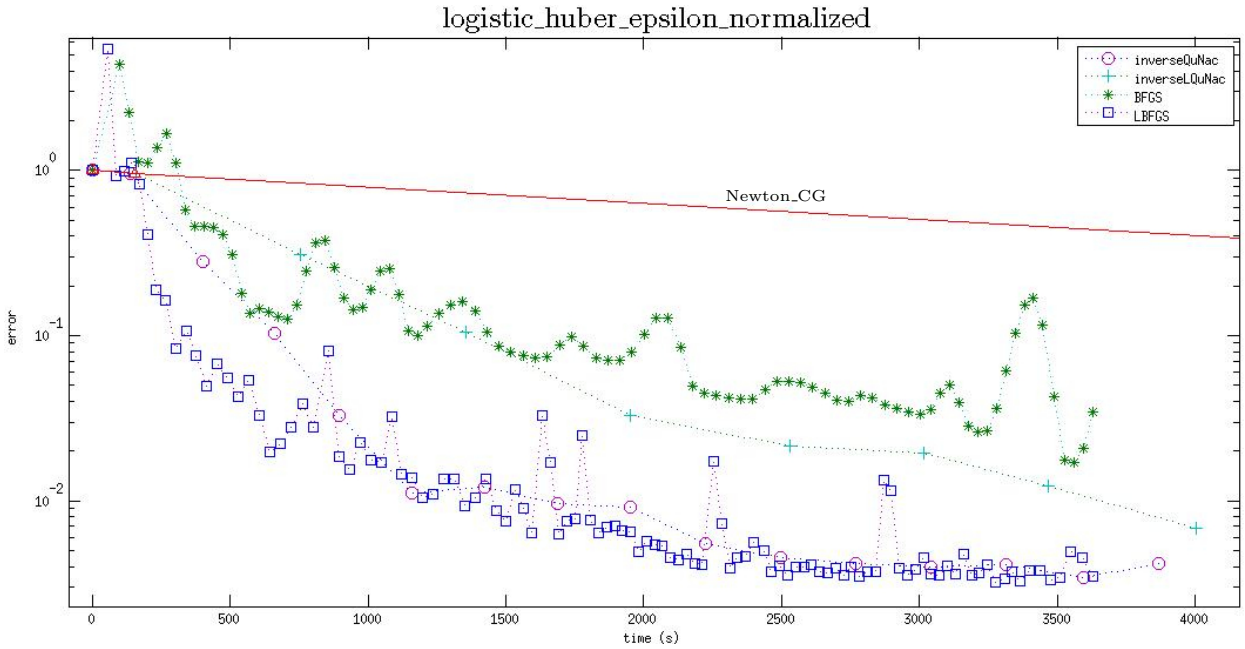



Figure 3: The `epsilon_normalized` problem with pseudo-Huber regularizer has 400,000 data points and 2000 features.

Pascal Large Scale Learning Challenge 2008[1], `epsilon_normalized` is very ill-conditioned. The L-BFGS and InversequNac enjoyed the fastest convergence, though the L-BFGS method suffered from some oscillation thus the quality of its solution depends on when the algorithm is terminated.

In Figure 4a we have plotted the evolution of the error through time for the full memory methods: InverseQuNac, BFGS, and Newton-CG, applied to `cod-rna.r` with an $\ell_2$ regularizer. In this plot, the InverseQuNac method converges first in just over 2 seconds followed by Newton_CG in 4 seconds. The BFGS method needs more than 16 seconds to converge.

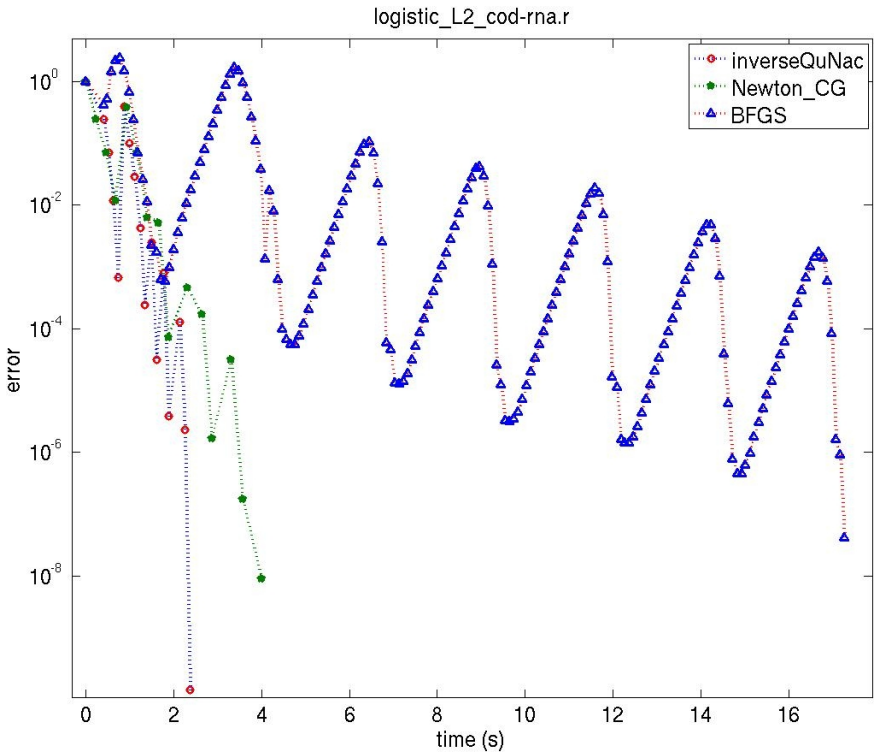
To not forget the benefits of limited memory implementations, we have tested two additional large-scale problems, `rcv1_train-binary` and `duke breast-cancer`, whose dimensions do not permit a full memory implementation. In Figures 4c and 4b we have plotted the evolution of the error through time for InverseLQuNac, Newton_CG and L-BFGS.

The three methods had similar results on the `rcv1_train-binary` though the L-BFGS converged first. While on the `duke breast-cancer`, the InverseLQuNac converged in just over 60 seconds, Newton-CG stagnated at a very high error of 0.4 and L-BFGS rapidly decreased the error initially, but stagnated at an error of $10^{-6}$.

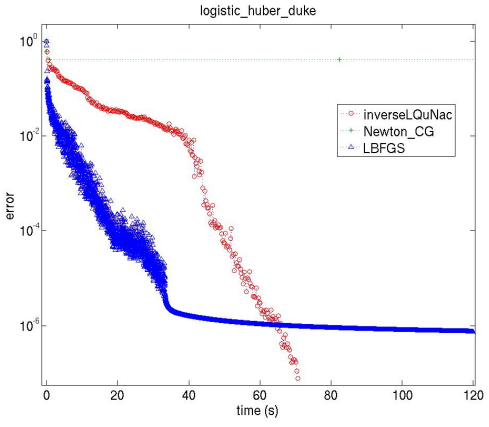
### 6.2 Classic Academic functions

We selected a number of academic unconstrained problems from [29] based solely on scalability of the function and availability of the MATLAB code, in that, together with their derivatives

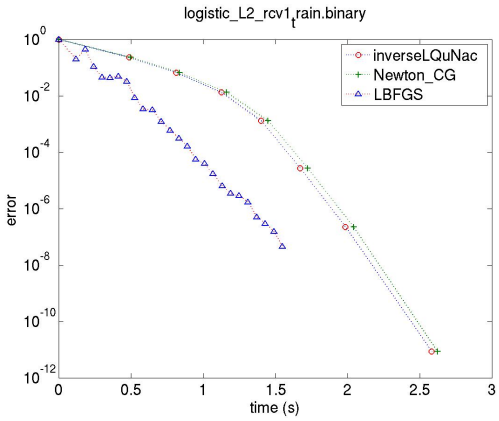
[1] `http://largescale.ml.tu-berlin.de/about/`




(a) The evolution of the error through time for each method applied to SVM with $\ell_2$ regularizer on the `cod-rna.r2` problem. The error is on a logarithmic scale.




(b) The `duke breast-cancer` problem with pseudo-Huber regularizer has 44 data points and 7129 features.




(c) The `rcv1_train-binary` problem with $\ell_2$ regularizer has 20242 data points and 47236 features.

Figure 4: The evolution of error through time for each limited memory method applied to SVM LR problem

| Problem | Description |
|---|---|
| The Watson function | quartic function |
| The Penalty Function #1 | quartic penalty function |
| The Penalty Function #2 | nonlinear penalty function |
| The Trigonometric Function | squared sum of trig. Functions |
| The Extended Rosenbrock parabolic valley #1 | indefinite Hessian matrix |
| The Extended Powell Singular Quartic | Singular Hessian matrix |
| The Chebyquad Function | quadrature of Chebyshev polynomials with no known solution |
| The Gregory and Karney Tridiagonal Matrix | Ill-conditioned positive definite quadratic |
| The Hilbert Matrix Function | Ill-conditioned positive definite quadratic |

Table 3: Unconstrained test set description

were readily coded thanks to John Burkardt (`http://people.sc.fsu.edu/~jburkardt/m_src/test_opt/test_opt.html`), see Table 3. Among these tests were two convex quadratic functions with ill-conditioned $Q \in \mathbb{R}^{n\times n}$ Hessian matrices; The Hilbert matrix $Q^H_{ij} = 2/(i+j-1)$ for $1 \leq i, j \leq n$ and the Gregory and Karney Tridiagonal Matrix where $Q_{11} = 4$, $Q_{12} = -2$, $Q_{ii} = 2$, $Q_{i(i+1)} = -2 = Q_{i(i-1)}$ for $i = 2, \ldots, n$.

Each test specifies an initial starting point from which we run each method until $\|\nabla f(x)\|/\|\nabla f(x_0)\| < \epsilon$, which we set to $\epsilon = 10^{-8}$, or until 10 minutes of time was exhausted. As a number of these problems were not convex, we employed a resetting and curvature criteria. Before taking a step in the $d_k$ direction, line 4 of Algorithm 5.1, we verify if

$$-\frac{\langle d_k, \nabla f_k\rangle}{\|d_k\|\|\nabla f_k\|} > \epsilon,$$

otherwise we reset the estimate $H_k = H_0$ and set $d_k = -H_0\nabla f_k$. As many of these test functions have indefinite Hessian matrices, we terminate the PCG method at line 7 of Algorithm 5.2 when negative curvature $\langle Ap_i, p_i\rangle < 0$ is encountered. If no direction of positive curvature is encountered, the estimate matrix is not updated, and we repeat the use of the previous estimate matrix $H_{k+1} = H_k$. This idea of repeating a previous estimate has been analysed in detail and tested in [17].

In Table 4 we report times taken to attain a stationary point for each method. The Newton_CG method was the fastest on 31 out of the 66 problems, while InverseQuNac, InverseLQuNac, BFGS and L-BFGS methods were the fastest on 15, 5, 7 and 8 problems, respectively. Comparing only the limited memory methods, Newton-CG, InverseLQuNac and L-BFGS methods were the fastest on 36, 20 and 8 problems, respectively. The InverseQuNac is the most stable, in that, it reached a stationary point on the largest number of problems; 65 out of 66. The results show that this particular adaptation of the quNac method for general non-convex functions was very robust.

| Problem | dimension | InverseQuNac | inverseLQuNac | Newton_CG | BFGS | LBFGS |
|---|---|---|---|---|---|---|
| The Penalty Function #2 | 100 | 0.16 | 0.18 | 0.24 | 0.59 | 0.06 |
| | 125 | 0.25 | 0.19 | 0.28 | 1.32 | 0.09 |
| | 150 | 0.33 | 0.26 | 0.38 | 2.35 | 0.12 |
| The Penalty Function #1 | 100 | 0.06 | 0.06 | 0.05 | 0.06 | 0.05 |
| | 200 | 0.08 | 0.06 | 0.05 | 0.11 | 0.05 |
| | 300 | 0.10 | 0.06 | 0.05 | 0.17 | 0.05 |
| | 400 | 0.12 | 0.06 | 0.05 | 0.23 | 0.05 |
| | 500 | 0.18 | 0.07 | 0.05 | 0.32 | 0.05 |
| | 600 | 0.22 | 0.06 | 0.05 | 0.39 | 0.05 |
| | 700 | 0.28 | 0.07 | 0.05 | 0.51 | 0.05 |
| | 800 | 0.35 | 0.07 | 0.05 | 0.64 | 0.05 |
| | 900 | 0.45 | 0.07 | 0.05 | 0.79 | 0.05 |
| | 1000 | 0.56 | 0.07 | 0.05 | 0.98 | 0.05 |
| Rosenbrock # 1 | 100 | 0.07 | 0.09 | 0.07 | 0.09 | 0.08 |
| | 200 | 0.09 | 0.10 | 0.07 | 0.19 | 0.09 |
| | 300 | 0.13 | 0.11 | 0.08 | 0.27 | 0.09 |
| | 400 | 0.17 | 0.11 | 0.08 | 0.39 | 0.10 |
| | 500 | 0.25 | 0.12 | 0.09 | 0.52 | 0.10 |
| | 600 | 0.31 | 0.12 | 0.09 | 0.68 | 0.11 |
| | 700 | 0.40 | 0.12 | 0.09 | 0.85 | 0.11 |
| | 800 | 0.50 | 0.12 | 0.10 | 1.05 | 0.11 |
| | 900 | 0.63 | 0.13 | 0.10 | 1.31 | 0.11 |
| | 1000 | 0.77 | 0.13 | 0.11 | 1.64 | 0.11 |
| The Extended Powell | 100 | 0.08 | 0.08 | 0.07 | 0.12 | 0.16 |
| | 200 | 0.09 | 0.08 | 0.08 | 0.28 | 0.11 |
| | 300 | 0.12 | 0.09 | 0.08 | 0.41 | 0.11 |
| | 400 | 0.15 | 0.09 | 0.09 | 0.58 | 0.30 |
| | 500 | 0.20 | 0.10 | 0.09 | 0.78 | 0.31 |
| | 600 | 0.28 | 0.10 | 0.10 | 1.00 | 0.32 |
| | 700 | 0.33 | 0.10 | 0.11 | 1.24 | 0.33 |
| | 800 | 0.42 | 0.10 | 0.11 | 1.52 | 0.33 |
| | 900 | 0.53 | 0.11 | 0.11 | 1.95 | 0.34 |
| | 1000 | 0.67 | 0.11 | 0.12 | 2.45 | 0.36 |
| The Watson function | 100 | 1.07 | 2.43 | 6.27 | 0.93 | TO |
| | 200 | 7.73 | 15.35 | 20.31 | 1.68 | TO |
| | 300 | 9.43 | 60.54 | 63.27 | 2.78 | TO |
| | 400 | 65.36 | 95.45 | 74.20 | 3.42 | TO |
| | 500 | 97.53 | 311.24 | 344.94 | 5.01 | TO |
| | 600 | 71.71 | 328.34 | 163.18 | 6.69 | TO |
| The Chebyquad Function | 10 | 0.28 | 0.42 | 0.45 | 0.20 | 0.60 |
| | 20 | 0.15 | 0.77 | 0.74 | ss | ss |
| | 30 | 0.81 | TO | 23.21 | ss | ss |
| Tridiagonal Matrix Function | 100 | 0.05 | 0.07 | 0.02 | ss | TO |
| | 200 | 0.08 | 0.14 | 0.05 | ss | TO |
| | 300 | 0.17 | 0.24 | 0.07 | ss | TO |
| | 400 | 0.27 | 0.35 | 0.10 | ss | TO |
| | 500 | 0.55 | 0.48 | 0.13 | ss | TO |
| | 600 | 0.75 | 0.57 | 0.17 | ss | TO |
| | 700 | 1.05 | 0.69 | 0.20 | ss | TO |
| | 800 | 1.42 | 0.82 | 0.24 | ss | TO |
| | 900 | 2.18 | 0.97 | 0.27 | ss | TO |
| | 1000 | 3.12 | 1.13 | 0.31 | ss | TO |
| The Hilbert Matrix Function | 100 | 0.03 | 0.04 | 0.05 | 0.30 | 19.71 |
| | 200 | 0.07 | 0.08 | 0.18 | 0.93 | 162.88 |
| | 300 | 0.12 | 0.29 | 0.47 | 2.15 | TO |
| | 400 | 0.21 | 0.54 | 0.69 | 3.46 | TO |
| | 500 | 0.34 | 0.83 | 1.70 | 6.07 | 549.74 |
| | 600 | 0.46 | 1.31 | 2.30 | 8.28 | 538.13 |
| | 700 | 0.60 | 1.78 | 3.15 | 11.06 | TO |
| | 800 | 0.79 | 2.28 | 4.53 | 14.16 | TO |
| | 900 | 1.03 | 2.97 | 5.17 | 17.84 | TO |
| | 1000 | 1.22 | 3.53 | 5.70 | 21.76 | TO |
| The Trigonoestimate Function | 100 | ss | 2.44 | 2.34 | ss | ss |
| | 200 | 0.61 | ss | ss | ss | ss |
| | 300 | 1.50 | ss | ss | ss | ss |
| | 400 | 1.70 | 23.45 | 17.36 | ss | ss |
| standard deviation | | 16.72 | 57.71 | 48.44 | 4.55 | 124.76 |

Table 4: Tests on Academic functions from Table 3 with $\epsilon = 10^{-8}$ and memory= 20. TO = TimeOut and ss = small step. The highlighted cells contain the fastest run time, while the boxed cells contain the fastest run time among the limited memory implementations

# 7 Conclusion

We have developed a family of updating schemes that generates a sequence of symmetric matrices which approximate a desired target sequence of symmetric matrices, where only the action of our target matrices on certain subspaces is known. Furthermore, the updates have small rank, with rank at most three times that of the given subspace dimension. This setup allows us to estimate the inverse of a matrix field, such as the inverse Hessian matrix, only by sampling its action and never explicitly calculating the inverse. Sufficient conditions for positive definiteness and the quadratic hereditary property of the estimates are established in this general setting.

The application we focus on is solving sequences of Newton systems; a common building block of many optimization methods. In this setting, we match the action of our estimate matrix to that of the Hessian (or inverse) on a Krylov basis of directions of positive curvature. This choice guarantees positive definiteness of the estimate matrices.

Additionally, we present an implementation for these methods in Algorithm 5.1 and a limited memory variant in Algorithm 5.4 in a Newton-CG framework. Both update variants exploit parallel linear algebra, essentially performing multiple BFGS updates in parallel. This is apparently the first such parallel implementations of BFGS and L-BFGS updates. Quadratic hereditary is proved for the full memory implementation. Tests of linear SVM problems with Logistic Loss and a regularizer have shown the `inverse` quNac method to be very promising, while our tests on Classic academic problems indicate that it is robust. Certainly more exhaustive tests are required.

The flexibility afforded by the action constraint could potentially be used to incorporate these methods into various optimization frameworks, such as active set methods where the sampling matrix is the basis of kernel of active linear constraints. Furthermore, using positive curvature is not the only possibility. Directions of negative curvature could be explored in a trust region model [20, 30].

**Acknowledgements and funding:** The authors would like to thank Felix Lieder for his suggestions on constructing positive definite matrices, and Artur Gower for proof reading the manuscript.

# References

[1] Igor S Aranson and Lorenz Kramer, *The world of the complex Ginzburg-Landau equation*, Reviews of Modern Physics, 74 (2002), pp. 99–143.

[2] Stefania Bellavia, Valentina De Simone, Benedetta Morini, and Daniela di Serafino, *On the update of constraint preconditioners for regularized KKT systems*, Optimization Online, (2014).

[3] L. Bergamaschi, R. Bru, A. Martínez, and M. Putti, *Quasi-Newton preconditioners for the inexact Newton method*, Electronic Transactions on Numerical Analysis, 23 (2006), pp. 76–87.

[4] E. G. Birgin and J. M. Martínez, *Structured minimal-memory inexact quasi-Newton method and secant preconditioners for augmented Lagrangian optimization*, Computational Optimization and Applications, 39 (2007), pp. 1–16.

[5] CG Broyden, *A class of methods for solving nonlinear simultaneous equations*, Mathematics of computation, 19 (1965), pp. 577–593.

[6] C. G. Broyden, *The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations*, J. Inst. Maths Applics, 76 (1970), pp. 76–90.

[7] Emmanuel J. Candès and Benjamin Recht, *Exact Matrix Completion via Convex Optimization*, Foundations of Computational Mathematics, 9 (2009), pp. 717–772.

[8] Chih-Chung Chang and Chih-Jen Lin, *Libsvm*, ACM Transactions on Intelligent Systems and Technology, 2 (2011), pp. 1–27.

[9] Bruce Christianson, *Automatic Hessians by reverse accumulation*, IMA J. Numer. Anal., 12 (1992), pp. 135–150.

[10] W. C Davidon, *Variable metric method for minimization*, tech. report, A.E.C. Research and Development Report, ANL-5990, 1959.

[11] Ron S. Dembo, Stanley C. Eisenstat, and Trond Steihaug, *Inexact Newton Methods*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 400–408.

[12] J. E. Jr. Dennis and R. B. Schnabel, *Least Change Secant Updates for Quasi-Newton Methods*, SIAM Review, 21 (1979), pp. 443–459.

[13] By R Fletcher and M J D Powell, *A rapidly convergent descent method for minimization*, The Computer Journal, 6 (1963), pp. 163—-168.

[14] Rodger Fletcher, *A new approach to variable metric algorithms*, The Computer Journal, 13 (1970), pp. 317–323.

[15] Kimon Fountoulakis and Jacek Gondzio, *A Second-Order Method for Strongly Convex l1-regularization Problems*, tech. report, Technical Report ERGO-13-011., 2013.

[16] André Gaul and Nico Schlömer, *Preconditioned Recycling Krylov subspace methods for self-adjoint problems*, ArXiv e-prints, (2012), pp. 1–28.

[17] Philip E Gill and Michael W Leonard, *Reduced-Hessian quasi-Newton methods for unconstrained optimization*, SIAM J. Optim., 12 (2001), pp. 209–237.

[18] L Giraud, S Gratton, and E Martin, *Incremental spectral preconditioners for sequences of linear systems*, Applied Numerical Mathematics, 57 (2007), pp. 1164–1180.

[19] Donald Goldfarb, *A Family of Variable-Metric Methods Derived by Variational Means*, Mathematics of Computation, 24 (1970), p. 23.

[20] N. I. M. Gould, S. Lucidi, M. Roma, and Ph. Toint, *Exploiting negative curvature directions in linesearch methods for unconstrained optimization*, Optimization Methods and Software, 14 (2000), pp. 75–98.

[21] R. M. Gower, *Conjugate Gradients: The short and painful explanation with oblique projections*, tech. report, University of Edinburgh, Maxwell Institute for Mathematical Sciences, 2014.

[22] S Gratton, A Sartenaer, and J Tshimanga, *On a class of limited memory preconditioners for large scale linear systems with multiple right-hand sides*, SIAM Journal on Optimization, 21 (2011), pp. 912–935.

[23] By J Greenstadt, *Variations on Variable-Metric Methods*, Mathematics of Computation, 24 (1969), pp. 1–22.

[24] M. R. Hestenes and E. Stiefel, *Methods of Conjugate Gradients for Solving Linear Systems*, Journal of research of the National Bureau of Standards, 49 (1952).

[25] Thomas Huckle and Alexander Kallischko, *Frobenius Norm Minimization and Probing for Preconditioning*, International Journal of Computer Mathematics, 00 (2007), pp. 1–31.

[26] D. Loghin, D. Ruiz, and A. Touhami, *Adaptive preconditioners for nonlinear systems of equations*, Journal of Computational and Applied Mathematics, 189 (2006), pp. 362–374.

[27] J Mandel, *Balancing domain decomposition*, Communications on Numerical Methods in Engineering, 9 (1993), pp. 233–241.

[28] José Luis Morales and Jorge Nocedal, *Automatic Preconditioning by Limited Memory Quasi-Newton Updating*, SIAM Journal on Optimization, 10 (2000), pp. 1079–1096.

[29] JJ Moré, BS Garbow, and KE Hillstrom, *Testing unconstrained optimization software*, ACM Transactions on Mathematical ..., 7 (1981), pp. 17–41.

[30] JJ Moré and DC Sorensen, *On the use of directions of negative curvature in a modified Newton method*, Mathematical Programming, 16 (1979), pp. 1–20.

[31] Jorge Nocedal, *Updating Quasi-Newton Matrices with Limited Storage*, Mathematics of Computation, 35 (1980), p. 773.

[32] J Nocedal and S J Wright, *Numerical Optimization*, vol. 43 of Springer Series in Operations Research, Springer, 1999.

[33] Michael L. Parks, Eric de Sturler, Greg Mackey, Duane D. Johnson, and Spandan Maiti, *Recycling Krylov Subspaces for Sequences of Linear Systems*, SIAM Journal on Scientific Computing, 28 (2006), pp. 1651–1674.

[34] Barak A Pearlmutter, *Fast exact multiplication by the Hessian*, Tech. Report January, CSETech. Paper 286., 1993.

[35] K. B. Petersen and M. S. Pedersen, *The Matrix Cookbook*, tech. report, Technical University of Denmark, 2012.

[36] RB Schnabel, *Quasi-Newton Methods Using Multiple Secant Equations; CU-CS-247-83*, tech. report, Computer Science Technical Reports Computer. University of Colorado, Boulder, Boulder, 1983.

[37] D F Shanno, *Conditioning of Quasi-Newton Methods for Function Minimization*, Mathematics of Computation, 24 (1971), pp. 647–656.

[38] Jonathan Richard Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, tech. report, School of Computer Science Carnegie Mellon University, 1994.

[39] C Sulem and P L Sulem, *The Nonlinear Schrödinger Equation: Self-Focusing and Wave Collapse*, no. v. 139 in Applied Mathematical Sciences, Springer, 1999.

[40] Jurjen Duintjer Tebbens, *Efficient Preconditioning of sequences of nonsymmetric linear systems*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1918–1941.

[41] Max A Woodbury, *Inverting modified matrices*, tech. report, Rep. no. 42, Statistical Research Group, Princeton University, 1950.

[42] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin, *Recent Advances of Large-Scale Linear Classification*, Proceedings of the IEEE, 100 (2012), pp. 2584–2603.

# 8 Appendix: Updating the Inverse with the Direct approach

Dispensing the iteration subscript $k$, to find the inverse $(G+E)^{-1}$ when a `direct` quNac update $\texttt{quNac}(G, \mathcal{S} \ \rightarrow \ Q \ \mathcal{S})$. is applied to $G$, we use the Woodbury formula [41]

$$(G+E)^{-1} = G^{-1} - G^{-1}U(I+VG^{-1}U)^{-1}VG^{-1},$$

where $G, E \in \mathbb{R}^{n\times n}$ and $E = UV$ with $U, V^T \in \mathbb{R}^{n\times q}$. First we express the `direct` quNac update as two rank-$p$ updates $G + E^1 + E^2$ where

$$(G+E) = G + \underbrace{(Q-G)\text{proj}_{\mathcal{S}}^{Q}Q}_{E^1} - \underbrace{Q\text{proj}_{\mathcal{S}}^{Q}G\left(I - \text{proj}_{\mathcal{S}}^{Q}Q\right)}_{E^2}, \tag{35}$$

The first $E^1$ can be split up as $E^1 = U^1V^1$ with

$$U^1 = (Q-G)D, \quad V^1 = (\mathcal{S}^TQ\mathcal{S})^{-1}\mathcal{S}^TQ.$$

Applying the Woodbury formula where $H \equiv G^{-1}$ we get

$$\begin{aligned}
(G+E^1)^{-1} &= H - H(Q-G)\mathcal{S}\left(I + (\mathcal{S}^TQ\mathcal{S})^{-1}\mathcal{S}^TQH(Q-G)\mathcal{S}\right)^{-1}(\mathcal{S}^TQ\mathcal{S})^{-1}\mathcal{S}^TQH \\
&= H - H(Q-G)\mathcal{S}\left((\mathcal{S}^TQ\mathcal{S})^{-1}\mathcal{S}^TQHQ\mathcal{S}\right)^{-1}(\mathcal{S}^TQ\mathcal{S})^{-1}\mathcal{S}^TQH \\
&= H - H(Q-G)\mathcal{S}\left(\mathcal{S}^TQHQ\mathcal{S}\right)^{-1}\mathcal{S}^TQH \\
&= H - H(Q-G)\text{proj}_{\mathcal{S}}^{QHQ}QH.
\end{aligned}$$

The second update can be split up as $E^2 = U^2V^2$ with

$$U^2 = -Q\mathcal{S}(\mathcal{S}^TQ\mathcal{S})^{-1} = (V^1)^T, \quad V^2 = \mathcal{S}^TG\left(I - \text{proj}_{\mathcal{S}}^{Q}Q\right).$$

If we let $\bar{H} = (G+E^1)^{-1}$, then applying the Woodbury formula again

$$\begin{aligned}
&((G+E^1)+E^2)^{-1} = \bar{H} \\
&+ \underbrace{\bar{H}Q\mathcal{S}(\mathcal{S}^TQ\mathcal{S})^{-1}}_{\mathbb{I}}\left(\underbrace{I - \mathcal{S}^TG\left(I - \text{proj}_{\mathcal{S}}^{Q}Q\right)\bar{H}Q\mathcal{S}(\mathcal{S}^TQ\mathcal{S})^{-1}}_{\mathbb{III}}\right)^{-1}\underbrace{\mathcal{S}^TG\left(I - \text{proj}_{\mathcal{S}}^{Q}Q\right)\bar{H}}_{\mathbb{IIII}}.
\end{aligned}$$

When substituting in $\bar{H}$, simplifications arise such as

$$\begin{aligned}
\bar{H}Q\mathcal{S} &= \left(H - H(Q-G)\text{proj}_{\mathcal{S}}^{QHQ}QH\right)Q\mathcal{S} \\
&= (HQ\mathcal{S} - H(Q-G)\mathcal{S}) \\
&= \mathcal{S}.
\end{aligned}$$

Thus

$$\mathbb{I} = \bar{H}Q\mathcal{S}(\mathcal{S}^TQ\mathcal{S})^{-1} = \mathcal{S}(\mathcal{S}^TQ\mathcal{S})^{-1},$$

and

$$\begin{aligned}\mathbb{III} &= I - \mathcal{S}^T G\left(I - \text{proj}_{\mathcal{S}}^{Q} Q\right)\bar{H}Q\mathcal{S}(\mathcal{S}^T Q\mathcal{S})^{-1} \\ &= I - \mathcal{S}^T G\left(I - \text{proj}_{\mathcal{S}}^{Q} Q\right)D(\mathcal{S}^T Q\mathcal{S})^{-1} \\ &= I - \mathcal{S}^T G(\mathcal{S} - \mathcal{S})(\mathcal{S}^T Q\mathcal{S})^{-1} = I.\end{aligned}$$

For the final part, take note that

$$\begin{aligned}\mathcal{S}^T Q\bar{H} &= \mathcal{S}^T Q\left(H - H(Q-G)\text{proj}_{\mathcal{S}}^{QHQ} QH\right) \\ &= \mathcal{S}^T QH - \mathcal{S}^T Q(HQ - I)\text{proj}_{\mathcal{S}}^{QHQ} QH \\ &= \mathcal{S}^T QH + \mathcal{S}^T Q\text{proj}_{\mathcal{S}}^{QHQ} QH - \mathcal{S}^T QH \\ &= \mathcal{S}^T Q\text{proj}_{\mathcal{S}}^{QHQ} QH.\end{aligned}$$

Furthermore

$$\begin{aligned}\mathcal{S}^T G\bar{H} &= \mathcal{S}^T G\left(H - H(Q-G)\text{proj}_{\mathcal{S}}^{QHQ} QH\right) \\ &= \mathcal{S}^T\left(I + (G-Q)\text{proj}_{\mathcal{S}}^{QHQ} QH\right)\end{aligned}$$

Thus

$$\begin{aligned}\mathbb{IIII} &= \mathcal{S}^T G\left(I - \text{proj}_{\mathcal{S}}^{Q} Q\right)\bar{H} \\ &= \mathcal{S}^T G\bar{H} - \mathcal{S}^T GD(\mathcal{S}^T Q\mathcal{S})^{-1}\mathcal{S}^T Q\bar{H} \\ &= \mathcal{S}^T G\bar{H} - \mathcal{S}^T G\text{proj}_{\mathcal{S}}^{QHQ} QH \\ &= \mathcal{S}^T\left(I + (G-Q)\text{proj}_{\mathcal{S}}^{QHQ} QH\right) - \mathcal{S}^T G\text{proj}_{\mathcal{S}}^{QHQ} QH \\ &= \mathcal{S}^T\left(I - Q\text{proj}_{\mathcal{S}}^{QHQ} QH\right).\end{aligned}$$

Bringing all this together yields

$$\begin{aligned}(G+E)^{-1} &= \overbrace{H - H(Q-G)\text{proj}_{\mathcal{S}}^{QHQ} QH}^{\bar{H}} + \overbrace{\text{proj}_{\mathcal{S}}^{Q}\left(I - Q\text{proj}_{\mathcal{S}}^{QHQ} QH\right)}^{\mathbb{I}\cdot\mathbb{III}\cdot\mathbb{IIII}} \\ &= H - (HQ - I)\text{proj}_{\mathcal{S}}^{QHQ} QH + \text{proj}_{\mathcal{S}}^{Q} - \text{proj}_{\mathcal{S}}^{QHQ} QH \\ &= H + \text{proj}_{\mathcal{S}}^{Q} - HQ\text{proj}_{\mathcal{S}}^{QHQ} QH.\end{aligned}$$

With indices

$$(G_k + E_k)^{-1} = H_k + \text{proj}_{\mathcal{S}_k}^{Q_{k+1}} - H_k Q_{k+1}\text{proj}_{\mathcal{S}_k}^{Q_{k+1}H_k Q_{k+1}} Q_{k+1} H_k. \tag{36}$$