Investigation of quasi-Newton methods for unconstrained optimization

Yang Ding
Enkeleida Lushi
Qingguo Li

Simon Fraser University
8888 University Drive
Burnaby, B.C., V5A 1S6
Canada

Abstract

In this paper, we investigate quasi-Newton methods for solving unconstrained optimization problems. We consider four different quasi-Newton update formulas, namely, BFGS, DFP, SR1 and PSB. Line search and trust region strategies are used in the algorithms to find the step length at each iteration. The algorithms are tested on 30 benchmark problems and comparisons are made between different updating methods and also the step update strategies. Conclusions are drawn upon the obtained results. For nonlinear equations, we considered Broyden's method and test three algorithms on the set of benchmark problems.

0.1 Introduction

The most well-known minimization technique for unconstrained problems is Newton's Method. It is effective, robust and quadratically convergent. In each iteration, the step update is: $x_{k+1} = x_k - (\nabla^2 f_k)^{-1}\nabla f_k$. The second derivative need to be calculated analytically and supplied to the algorithm by the user. However, Newton's method has the disadvantage of being computationally expensive. The inverse of the Hessian has to be calculated in every iteration, and that is rather costly. Moreover, in some applications, the second derivatives may be unavailable. One fix to the problem is to use a finite difference approximation to the Hessian. (See Chapter 7 of [10].) The other fix, which is more widely used, is quasi-Newton Methods, where approximate Hessian or inverse Hessian updates are updated in each iteration, while the gradients are supplied.

We consider solving the nonlinear unconstrained minimization problem

$$\min f(x), x \in R^n \tag{1}$$

The general structure of quasi-Newton method can be summarized as follows

Given x_0 and initial Hessian approximation B_0(inverse Hessian approximation H_0);
$k \to 0$;
For $k = 0, 1, 2, ...$
Evaluate gradient g_k.
Calculate the update vector s_k by line search or trust region methods.

$$x_{k+1} \leftarrow x_k + s_k;$$
$$y_k \leftarrow g_{k+1} - g_k;$$

Update B_{k+1} or H_{k+1} according to the quasi-Newton formulas.
End(for)

The basic requirement for the updating formula is that the *secant condition* is satisfied in each iteration, i.e.,

$$B_{k+1}s_k = y_k \tag{2}$$

A variety of formulas have been found to update B_k that satisfy (2). In this study, we focus on four popular quasi-Newton methods, namely BFGS, DFP, PSB and SR1. We also discuss Broyden's method for nonlinear equations. The performances of these methods are compared to each other. We also compare the line search and trust region strategies to find the update step.

The report is organized as follows: Section 2 outlines the four different update formulas and their properties; Section 3 discusses the line search and trust region strategies and the algorithms; Section 4 presents Broyden's method for nonlinear equations, Section 4 compares the methods and presents the numerical results, and we conclude our report in Section 5.

0.2 Quasi-Newton Formulas for Optimization

The basic idea behind the quas-Newton formulas is to update B_{k+1} to B_k in some computational cheap ways while ensure the *secant condition* (2). And the computation of the update should be relative cheap.

0.2.1 BFGS

The BFGS as the most popular quasi-Newton method. It is well-liked for it's robustness and for its self-correcting properties, as well as for the superlinear convergence it achieves. The inverese Hessian update H_{k+1} is obtained by solving the problem:

$$\min_H \|H - H_k\|, \quad s.t. \quad H = H^T, Hy_k = s_k \tag{3}$$

The solution of (3) gives the inverse Hessian update

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T \tag{4}$$

where

$$\rho_k = \frac{1}{y_k^T s_k}$$

The Hessian update can be calculated by using the Sherman-Morrison-Woodbury formula[7] as

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{5}$$

Good initial approximations of the (inverse) Hessian are often used to boost the performance of the method. The choices usually use preconditioning or scaling as follows:

- The identity matrix I, or some multiple of it, $H_0 = \beta I$
- Some finite difference approximation at x_0
- A scaled version of the identity, $H_0 = diag(s_1, s_2, \cdots, s_n)$, where s_i are scales.
- H_0 is rescaled before H_1 is computed as $H_0 = (y_0^T s_0 / y_0^T y_0) I$ which ensures that H_{k+1} doesn't get very large.

BFGS is the most effective quasi-Newton correction. If H_k is positive definite, then so is H_{k+1}, hence if H_0 is chosen positive definite, the rest of the H_k will be positive definite. Also, BFGS has self-correcting properties: if H_k incorrectly approximates the curvature of the objective function and this estimate slows down the iteration, then the (inverse) Hessian approximation will tend to correct itself in the next few steps. The self-correcting properties stand and fall with the quality of the Wolfe Line Search, as they ensure the model carries appropriate curvature information. The Line search should always try $\alpha = 1$ first, as this step will be accepted and produce superlinear convergence.

There are situations when this updating formula produces bad results.That happens when $y_k^T s_k \approx 0$ and H_k gets large. The commonly used fix is the rescaling of the initial (inverse) Hessian approximation. BFGS is also sensitive to round-off error and inaccurate line searches. It can fail for general nonlinear problems and it can get stuck on a saddle point.

The Hessian approximation B_k should not be used directly, because it needs to solve the system $B_k p_k = -g_k$. Instead, a Cholesky factorization of the matrix should be used, as it lowers the cost to as much as the version of the algorithm where the inverse Hessian is used.

0.2.2 DFP

The DFP update was first proposed by Davidon, and popularized by Fletcher and Powell. It comes from solving following problem:

$$\min_B \|B - B_k\|, \quad s.t. \quad B = B^T, Bs_k = y_k \tag{6}$$

The solution of (6) gives the Hessian update

$$B_{k+1} = (I - \gamma_k y_k s_k^T) B_k (I - \gamma_k s_k y_k^T) + \gamma_k y_k y_k^T \tag{7}$$

where

$$\gamma_k = \frac{1}{y_k^T s_k}$$

by using the Sherman-Morrison-Woodbury formula[7], we get the inverse Hessian update as

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k} \tag{8}$$

DFP is a method very similar to BFGS. When it was discovered, it revolutionized the field of non-linear optimization.

This formula, like BFGS, is a rank 2 formula update and it has nice properties as well, however it is not as fast. It is less effective than BFGS at self-correcting of the Hessians. Likewise, DFP could fail for general nonlinear problems, it can stop at a saddle point, it is sensitive to inaccurate line searches and it's Hessian updates are sensitive to round-off errors and other inaccuracies. Scaling and preconditioning exist to boost the performance of the method. DFP has also an interesting property that, for a quadratic objective function, it generates the directions of the conjugate gradient while constructing the inverse Hessian. Something to note is that the DFP update is the dual of the BFGS update.

M.J.D. Powell analyzed the performance of BFGS and DFP algorithm on a very simple objective function of two variables. Through studying the eigenvalues of the Hessian matrix,he found out that the DFP algorithm can be highly inefficient at correcting erroneously large eigenvalues of matrices, which may be the main reason in general for the observed superiority of the BFGS formula[6].

0.2.3 PSB

The PSB (Powell-symmetric-Broyden) update comes from the solution of the following problem:

$$\min_B \|B - B_k\|_F, \quad s.t. \quad Bs_k = y_k, (B - B_k) = (B - B_k)^T \tag{9}$$

The solution of (9) gives the Hessian update of PSB

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T + s_k (y_k - B_k s_k)^T}{s_k^T s_k} - \frac{s_k^T (y_k - B_k s_k) s_k s_k^T}{(s_k^T s_k)^2} \tag{10}$$

By using the Sherman-Morrison-Woodbury formula[7], we obtain the following inverse Hessian update for PSB:

$$H_{k+1} = H_k + \frac{(H_k - H_k y_k) y_k^T + y_k (s_k - H_k y_k)^T}{y_k^T y_k} - \frac{y_k^T (s_k - H_k y_k) y_k y_k^T}{(y_k^T y_k)^2} \tag{11}$$

PSB is a rank 2 symmetric update and closely related to the Broyden's update for non-linear equation. Some of its more notable properties are superlinear convergence. However,

it can converge locally to any isolated zero. The (inverse) Hessian updates are symmetric, but B_{k+1} is not necessarily positive definite even if B_k is positive definite. Like BFGS and DFP, it it can suffer from round-off error and inaccurate line searches.

0.2.4 SR1

The SR1(*symmetric-rank-1*) update comes from solving σ, ν from the general form

$$B_{k+1} = B_k + \sigma\nu\nu^T, \quad \text{s.t.} \quad B_{k+1}s_k = y_k \tag{12}$$

The solution of the problem gives the update the approximated Hessian matrix,

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \tag{13}$$

By using the Sherman-Morrison-Woodbury formula[7], we obtain the following inversed Hessian update for SR1

$$H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k} \tag{14}$$

A significant difference between BFGS and SR1 updates is that BFGS guarantees to produce a positive definite B_{k+1} if B_k is positive definite and $s_k y_k > 0$, while SR1 does not have this property. It has been proved that trust region SR1 has an n+1 step q-superlinear rate of convergence[5].

We are interested in the SR1 formula for the following reasons:

- The matrices generated are very good approximations to the (inverse) Hessian matrices, often better than BFGS.

- A simple safeguard prevents the breakdown of the method and the numerical instabilities.

- Sometimes when it is not possible to impose curvature conditions, indefinite Hessian approximations are desirable, as long as they reflect the real Hessian.

The drawback of the method is that sometimes $(s_k - H_k y_k)^T y_k \approx 0$ and there may not be a symmetric rank one formula that satisfies the secant condition. Hence instabilities and breakdown may occur. The fix is usually by skipping the update if the above quantity is small, and this has no negative effect on the iterations.

0.2.5 Convergence of matrix B_k

Convergence on the Hessian approximation matrix B_k to the real Hessian $\nabla^2 f(x)$ has also been studied[2][4][3]. The following common assumptions made are listed as follows.

(A.1) $f(x)$ is twice continously differentiable

(A.2) $\nabla^2 f(x)$ is Lipschitz continuous

(A.3) The sequence $\{x_k\}$ converges to some finite limit point x^*

(A.4) The sequence $\{s_k\}$ is uniformly linearly independent.

For BFGS and DFP, Ge and Powell shows that under the assumption (A.1) and (A.2), if $\nabla^2 f(x^*)$ is positive definite, and there exists a sequence $\{x_k\}$ converges to the limit x^*, then B_k will converge to $\nabla^2 f(x^*)$[3].

For PSB, Powell proved that under the assumption (A.1-4), if there exists a sequence $\{x_k\}$ converges to the limit x^*, then the sequence B_k will converge to $\nabla^2 f(x^*)$[4].

For SR1, it is shown that under (A.1-4), the then the sequence B_k will converge to $\nabla^2 f(x^*)$, and the convergence rate depends on the convergence rate of x_k [2].

0.3 Line Search Vs. Trust Region quasi-Newton Algorithms

To compute the new step update in quasi-Newton Methods we use either the Line Search or the Trust Region strategies. These two strategies have different properties and are best used with specific Hessian or Inverse Hessian updates.

0.3.1 Line search

The most commonly used Line Search Method is to find the step length that satisfies the (strong) Wolfe conditions (15) . The Wolfe Line Search conditions ensure that the gradients are sampled at points where the model captures important curvature information. The Line Search will need the Hessian approximations to be positive definite.

$$\begin{aligned} f(x_k + \alpha_k p_k) &\leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k \\ |\nabla f(x_k + \alpha_k p_k)^T p_k| &\leq c_2 |\nabla f_k^T p_k| \end{aligned} \tag{15}$$

The general structure of quasi-Newton algorithm with line search can be summarized as follows

Algorithm 1. Quasi-Newton algorithm with line search

Given starting point x_0, Hessian approximation B_0, convergence tolerance $\epsilon > 0$;

$k \leftarrow 0$;

While $\|\nabla\| > \epsilon$;

Compute a search direction

$$p_k = -(B_k)^{-1} g_k; \quad \text{or} \quad p_k = -H_k g_k \tag{16}$$

Choose α_k using line search along p_K;

Define

$$\begin{aligned} x_{k+1} &\leftarrow x_k + \alpha_k p_k; \\ s_k &\leftarrow x_{k+1} - x_k; \\ y_k &\leftarrow \nabla f_{k+1} - \nabla f_k; \end{aligned}$$

if the safeguard condition holds

Update B_{k+1} according to a quasi-Newton formula;

else

$$B_{k+1} = B_k$$

End(while)

There are many ways to do a line search in a given direction to find an acceptable step length α_k. Some require the Wolfe conditions to be satisfied, others require the Goldstein conditions. There are also derivative free line searches, like the Fibonacci or the golden section search. For this project, we coded two line searche methods that strive to maintain the strong Wolfe conditions, but that use different interpolations to find a trial step length in the subroutine **zoom**(refer to Chp. 3.4 in [7] for details). The first one uses bisection and the other uses a cubic interpolation.

0.3.2 Trust region

The trust region method attempts to search for an accpetable step in a neighborhood around x_k instead of searching in one direction as done in line search algorithms. The Trust Method is prefered because it does not require the Hessian approximations modified to be sufficiently positive definite. This suits very well rank one update methods, like SR1, where the Hessian approximations might not maintain positive definitiveness.

For completeness, we list the trust region based quasi-Newton algorithm as follows[5][7]

Algorithm 2. Quasi-Newton algorithm with trust region

Given starting point x_0, trust-region radius Δ_0, convergence tolerance $\epsilon > 0$, parameter $\eta \in (0, 10^{-3})$ and $r \in (0, 1)$;

$k \leftarrow 0$;

While $\|\nabla\| > \epsilon$;

Compute s_k by solving the subproblem

$$\min_s \nabla f_k^T s + \frac{1}{2} s^T B_k s \quad \text{subject to} \quad \|s\| \leq \Delta_k \tag{17}$$

Compute

$$y_k = \nabla f(x_k + s_k) - \nabla f_k;$$
$$\textbf{ared} = f_k - f(x_k + s_k)$$
$$\textbf{pred} = -\left(\nabla f_k^T s_k + \tfrac{1}{2} s_k^T B_k s_k\right)$$

If ared/pred$> \eta$

$$x_{k+1} = x_k + s_k$$

else

$$x_{k+1} = x_k$$

end(if)

if ared/pred> 0.75

if $\|s_k\| \leq 0.8\Delta_k$

$$\Delta_{k+1} = \Delta_k$$

else

$$\Delta_{k+1} = 2\Delta_k$$

end(if)

elseif $0.1 \leq$ **ared/pred** ≤ 0.75

$$\Delta_{k+1} = \Delta_k$$

else

$$\Delta_{k+1} = 0.5\Delta_k$$

end(if)

if safeguard condition holds

Compute B_{k+1} (even if $x_{k+1} = x_k$)

else

$$B_{k+1} = B_k$$

end(if)

$k \leftarrow k + 1$

end(while)

One feature of this implementation of trust region method is that it updates the Hessian approximation B_k at all steps, including the reject steps.

0.4 Quasi-Newton Methods for Nonlinear Equations

In some application, instead of optimizing a given objective function, we find values of variables in a model that statisfies a nonlinear equation. A nonlinear equation can be represented as:

$$r(x) = 0 \tag{18}$$

where $r : \mathbf{R}^n \rightarrow \mathbf{R}^n$ is a vector function, i.e.:

$$r(x) = \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_n(x) \end{bmatrix} \tag{19}$$

When the Jacobian $J(x)$ is available, Newton's method can be used to solve the equation (19). Secant methods or quasi-Newton methods can be implimented for solving nonlinear equations under the situation that the $J(x)$ is not avaliable or expensive to compute. In this method, B_k is updated to mimic the behavior of the true Jacobian $J(x)$, and used to compute a search direction p_k. The most successful update formula, proposed in 1965 by C. Broyden, is

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)s_k^T}{s_k^T s_k} \tag{20}$$

where B_k is the Jacobian approxiamtion at the iteration k, $s_k = x_{k+1} - x_k$, $y_k = r(x_{k+1}) - r(x_k)$.

It has been proved in [1] that the Broyden update makes the smallest possible change to the Jacobian in Euclidean norm $\|B_k - B_{k+1}\|$ that satisfies the following Secant equation

$$y_k = B_{k+1}s_k \tag{21}$$

The line search based quasi-Newton algorithm can be summarized as follows:

Algorithm 3. Quasi-Newton algorithm with line search for nonlinear equation

Given x_0 and B_0;
For $k = 0, 1, 2, ...$
Caculate a search direction p_k as

$$p_k = -(B_k)^{-1}r(x_k); \tag{22}$$

Choose α_k by line search along p_K;

$$\begin{gathered} x_{k+1} \leftarrow x_k + \alpha_k p_k; \\ s_k \leftarrow x_{k+1} - x_k; \\ y_k \leftarrow r(x_{k+1}) - r(x_k); \end{gathered}$$

Update B_{k+1} as (20);
End(for)
In order to use line search, we need to define a merit function as

$$f(x) = \frac{1}{2}\|r(x)\|^2 \tag{23}$$

to measure whether a new iterate is better or worse than the current iterate. Here we can choose a different norm as merit function. The gradient of the $f(x)$ is $\nabla f(x) = J(x)^T r(x)$, which is needed for performing a line search satisfying the Wolfe conditions. However, the requirement for the gradient is inappropriate for quasi-Newton methods. Backtracking line search, which enforces the sufficient decrease condition, can be used, but no global convergence result is found on Broyden's method with backtracking. Meanwhile, Griewank proposed a derivative free line search method for the nonlinear equation[8]. It is shown that if the Broyden's method with the proposed line search method is applied to nonlinear equations with uniformly nonsingular Jacobian matrices and the Jacobian matrices are Lipschitz continuous, then the method converges to the unique solution of the nonlinear equation. In the method, he defines a ratio $q_k(\alpha_k)$ as:

$$q_k(\alpha_k) = -\frac{r(x_k)^T(r(x_k + \alpha_k p_k) - r(x_k))}{\|r(x_k + \alpha_k p_k) - r(x_k)\|^2} \tag{24}$$

and accepts the steplength α_k if following inequlity holds:

$$q_k(\alpha_k) \geq 1/2 + \epsilon$$

where $\epsilon \in (0, 1/6)$ is a constant.

The inequlity implies that

$$\|r(x_{k+1})\| < \|r(x_k)\|$$

Meanwhile, the trust region method can also be used as a global strategy for Broyden's method. The performance ratio ρ_k of actual predicted reduction is defined as:

$$\rho_k = -\frac{\|r(x_k)\|^2 - \|r(x_k + p_k)\|^2}{\|r(x_k\|^2 - \|r(x_k) + J(x_k)p_k\|^2} \tag{25}$$

By considering this performance ratio, the trust region algorithm listed in [7] can be implemented.

We implemented Broyden's method with the three following step searches:
- Backtracking line search
- Derivative free line search
- Trust Region

The numerical results on these three algorithms can be found in next section.

0.5 Numerical Results

In this section, we test different quasi-Newton algorithms on a set of benchmark problems, taken from [9]. They are widely used in testing unconstrained optimization solvers.

0.5.1 Performance evaluation method

In practice, most comparisons between algorithms involve displaying a table of the performance of each solver on a set of problems. The interpretation of the results often gives a lot of disagreement. In this project, instead we use *performance profiles* to evaluate and compare the performance of the proposed quasi-Newton methods. The performance profile for a solver is the cumulative distribution function for a performance metric.The *performance profile* was proposed in [10] as a tool for benchmarking optimization software.

Here we explain something about the *performance profiles*. Assume there are n_s solvers running on n_p problem, for each problem p and solver s, we define the $t_{s,p}$ as the number of function evaluations (or other performance measure) required to solve the problem p. Define the performance ratio as

$$r_{s,p} = \frac{t_{s,p}}{min\{t_{s,p} : s \in \mathcal{S}\}} \tag{26}$$

if solver s fail in solving the problem p, the ratio is set to some sufficiently large number $r_M \geq r_{s,p}$.

The performance profile function is defined as

$$\rho_s(\tau) = \frac{1}{n_p} size\{p : 1 \leq p \leq n_p, log(r_s, p) \leq \tau\} \tag{27}$$

where $\rho_s(\tau)$ defines the probability for solver s that the performance ratio $r_{s,p}$ is within a factor τ of the best possible ratio. Note that $\rho_s(\tau) \in [0, 1]$ and the inequity $\rho_s(\tau_1) < \rho_g(\tau_1)$ indicates that the solver g performs better than solver s at τ_1.

In the test, the results are generated by running a quasi-Newton solver on the set of problems and recording the information such as the number of function evaluations and computing time, for example. Then we plot the *performance profiles* for the solvers and compare.

Function Number	Function (() refer to problem number in [9]) Name	Number of Functions	Number of Variables
1	Beale function(5)	3	2
2	Rosenbrock function (1)	2	2
3	Wood function (14)	6	4
4	Brown badly scaled function(4)	3	2
5	Powell badly scaled function(3)	2	2
6	Linear function - rank 1(33)	$m \geq n$	n
7	Helical valley function (7)	3	3
8	Gaussian function (9)	15	3
9	Box three-dimensional function (12)	$m \geq n$	3
10	Biggs EXP6 function (18)	$m \geq n$	6
11	Variably Dimensioned function(25)	$m = n + 2$	n
12	Watson function (20)	$m = 31$	$2 \leq n \leq 31$
13	Penalty I function(23)	$m = n + 1$	n
14	Penalty II function(24)	$m = 2n$	n
15	Brown and Dennis function (16)	$m \geq n$	4
16	Gulf research and development function(11)	$3 \leq m \leq 100$	3
17	Trigonometric function(26)	$m = n$	n
18	Extended Rosenbrock function(21)	$m = n$	n even
19	Extended Powell Singular function(22)	$m = n$	multiple of 4
20	Osborne 1 function (17)	33	5
21	Osborne 2 function (19)	65	11
22	Bard function (8)	15	3
23	Broyden tridiagonal function(30)	$m = n$	n
24	Broyden banded function (31)	$m = n$	n
25	Freudenstein and Roth function(2)	2	2
26	Jenrich and Sampson function (6)	$m \geq n$	2
27	Kowalik and Osborne function (15)	11	4
28	Discrete boundary value function(28)	$m = n$	n
29	Linear function - rank 1(34)	$m \geq n$	n
30	Meyer function(10)	16	3

Table 1: Testing functions for quasi-Newton methods

0.5.2 Comparison of quasi-Newton algorithms for unconstrained optimization

Incorporating our two the line searches and trust region algorithms into the four quasi-Newton we are considering, we obtain 24 quasi-Newton algorithms which are listed in the following table:

	BFGS		SR1		DFP		PSB	
	Hessian	Inverse Hessian	Hessian	Inverse Hessian	Hessian	Inverse Hessian	Hessian	Inverse Hessian
Line Search Bisection	LBFGS B	LIBFGS B	LSR1 B	LISR1 B	LDFP B	LIDFP B	LPSB B	LIPSB B
Line Search Cubic	LBFGS C	LIBFGS C	LSR1 C	LISR1 C	LDFP C	LIDFP C	LPSB C	LIPSB B
Trust Region	TBFGS	TIBFGS	TSR1	TISR1	TDFP	TIDFP	TPSB	TIPSB

Table 2: Quasi-Newton algorithms

The parameters we use for the line search methods are: $c_1 = 10^{-4}, c_2 = 0.9, \epsilon = 10^{-4}$. The parameters for trust region methods are: $\delta_0 = 0.2, \epsilon = 10^{-4}, \eta = 10^{-2}, r = 10^{-8}$.

In the first set of numerical experiments, we compare the four different quasi-Newton updates (BFGS, SR1, DFP, PSB) with each other for line search and trust region step finding strategies.

We compare the running time and plot the *performance profiles*. The results are shown in Fig. 1- 6. For the line search with bisection step finding way, we find that whether Hessian or inverse Hessian updates are used, the performance of BFGS is superior to the other methods'. In general the rank 2 updates do better than SR1.

For the the line search with cubic interpolation way, we notice similar results. BFGS still performs best, and DFP performs quite well compared to the other.

For the trust region way, BFGS and SR1 outperform the other two methods, just as shown in the literature. The good performance of SR1 in this case could be explained because the trust region way to find the step length is well-suited to it, since the Hessians need not be positive definite .

0.5.3 Line search vs. trust region

In the following experiments, we investigate how each of the four updates performs with two different ways to compute the step length: the line search method or the trust region method. For this comparison, we consider only the line search with bisection. The results are shown in Fig.7-10.

We notice from the plots that there is a clear discrepancy for between the two strategies for the quasi-Newton updates BFGS and SR1 , as trust region strategy performs better than the line search strategy for the benchmark set of problems. DFP and PSB updates perform more or less the same with both strategies.

We also compare trust region based BFGS and SR1 methods with MATLAB's minimization toolbox function **fminunc**, which uses a line search based BFGS. The result is shown in Fig.11. We notice the trust region based methods perform better for the testing problems. Tables of the number of function evaluation and the CPU time are attached in the appendix. The results show that BGFS and SR1 solve almost all the problems, while the MATLAB function fails on three of them.

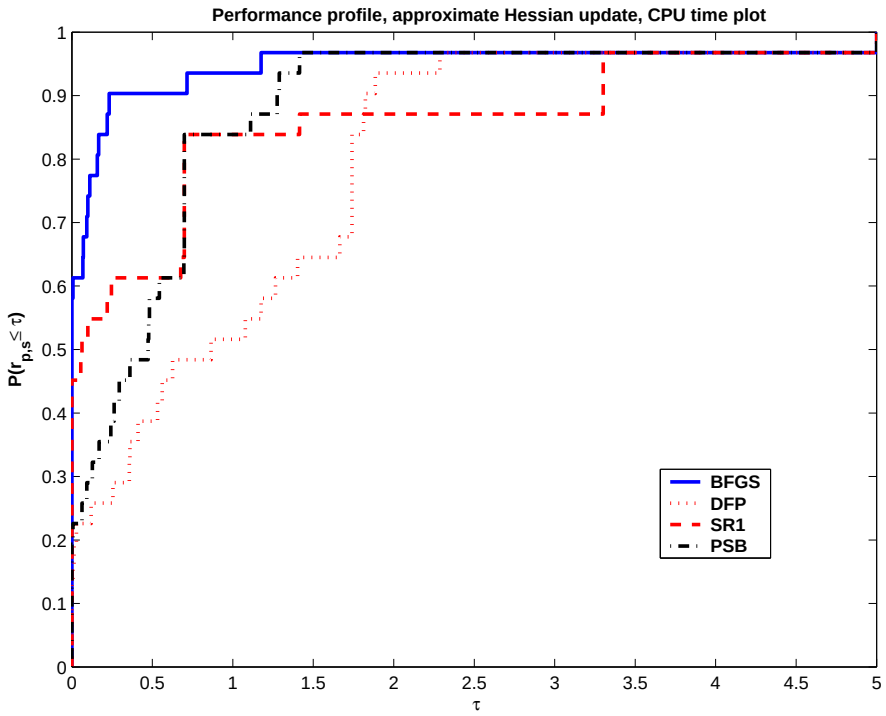

Figure 1: Performance profile for Hessian update with bisection line search method

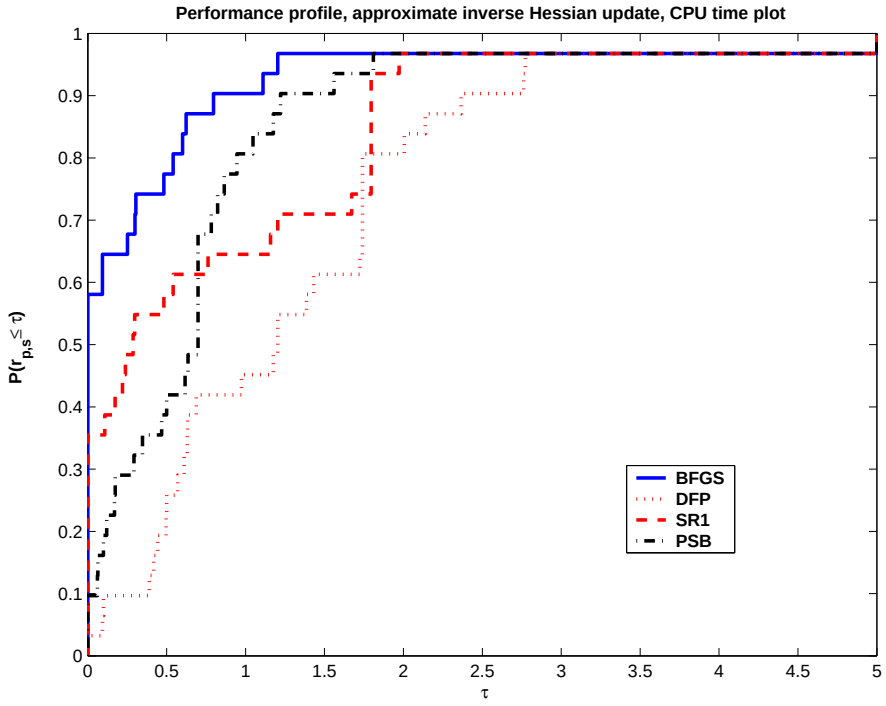

Figure 2: Performance profile for inverse Hessian update with bisection line search method

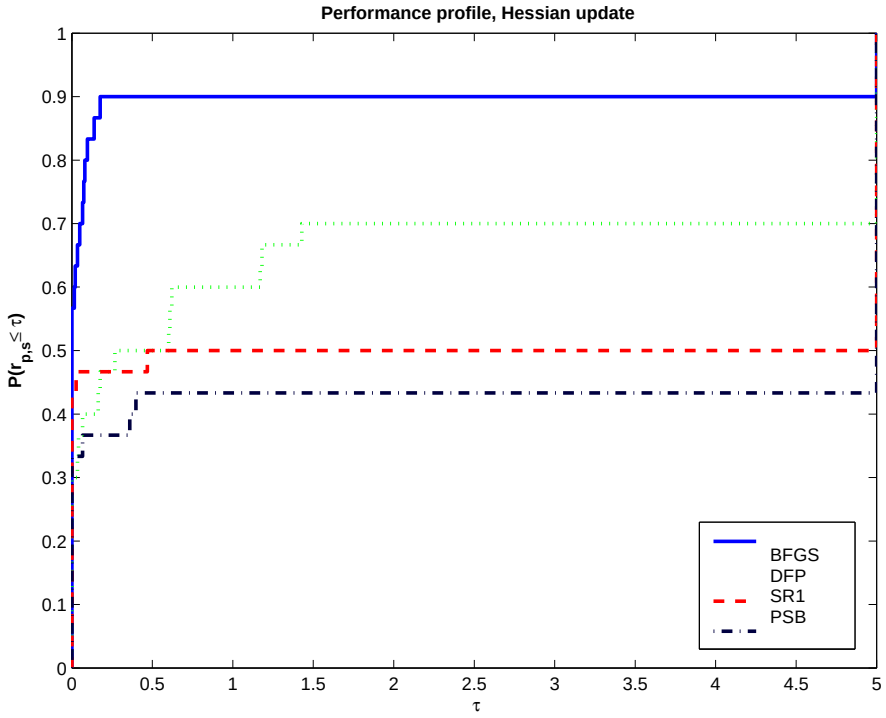

Figure 3: Performance profile for Hessian update with cubic line search method

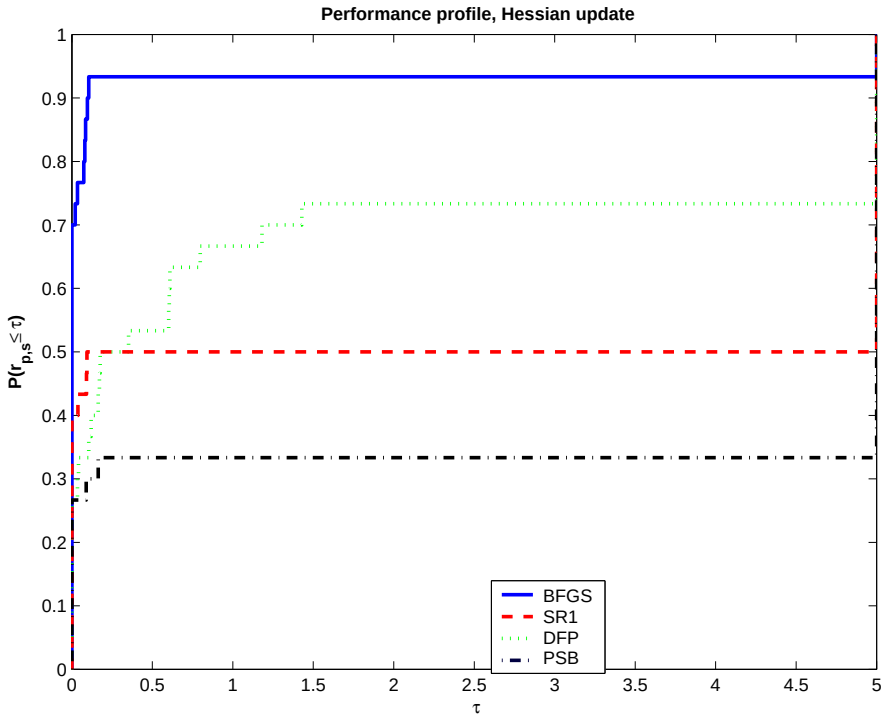

Figure 4: Performance profile for inverse Hessian update with cubic line search method

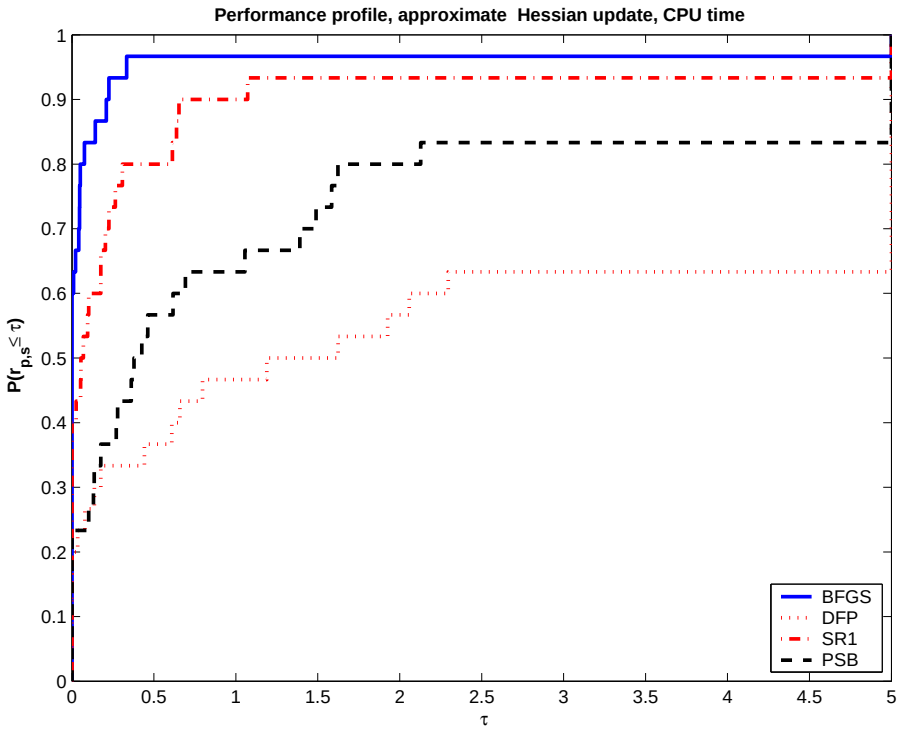

Figure 5: Performance profile for Hessian update with Trust region

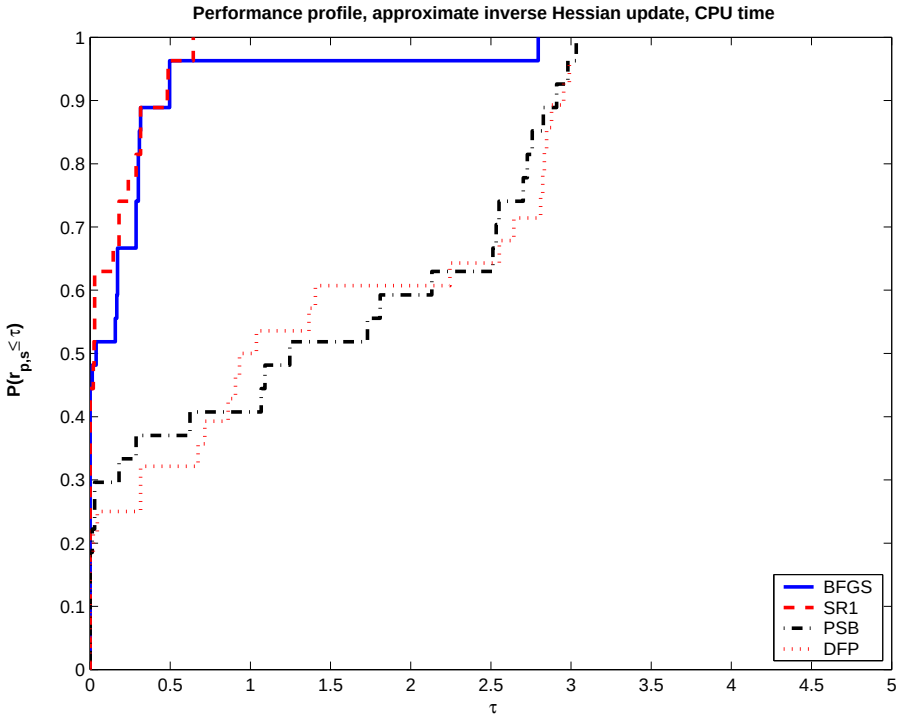

Figure 6: Performance profile for inverse Hessian update with Trust region

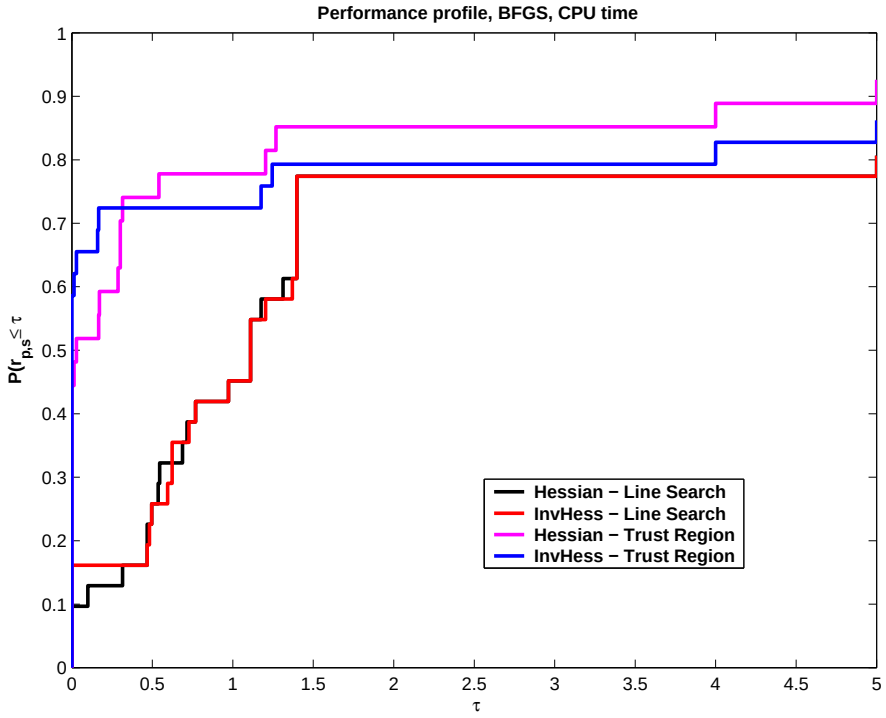

Figure 7: Performance profile for BFGS with line search and Trust region

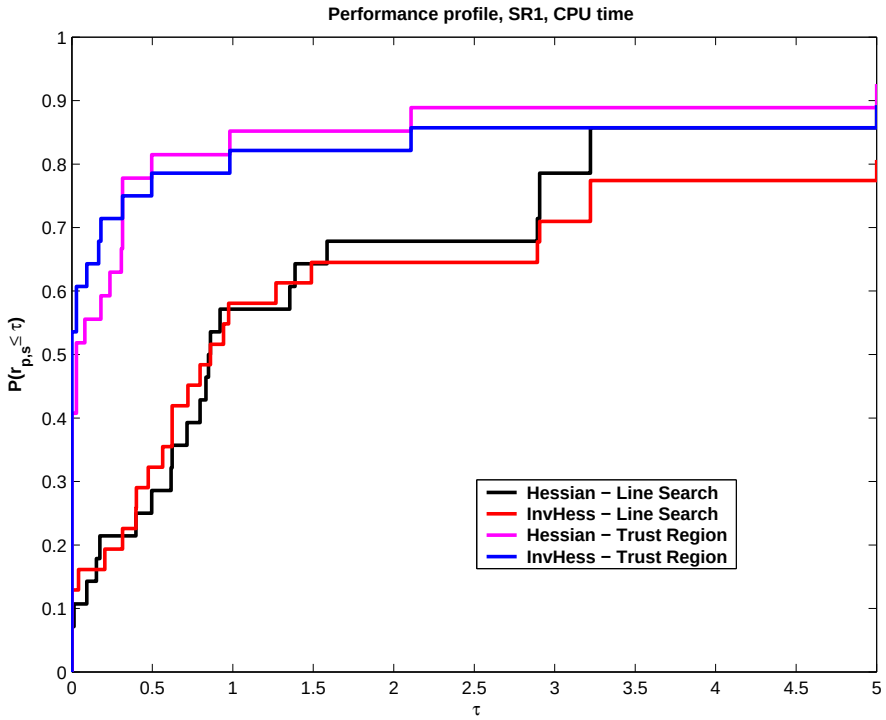

Figure 8: Performance profile for SR1 with line search and Trust region

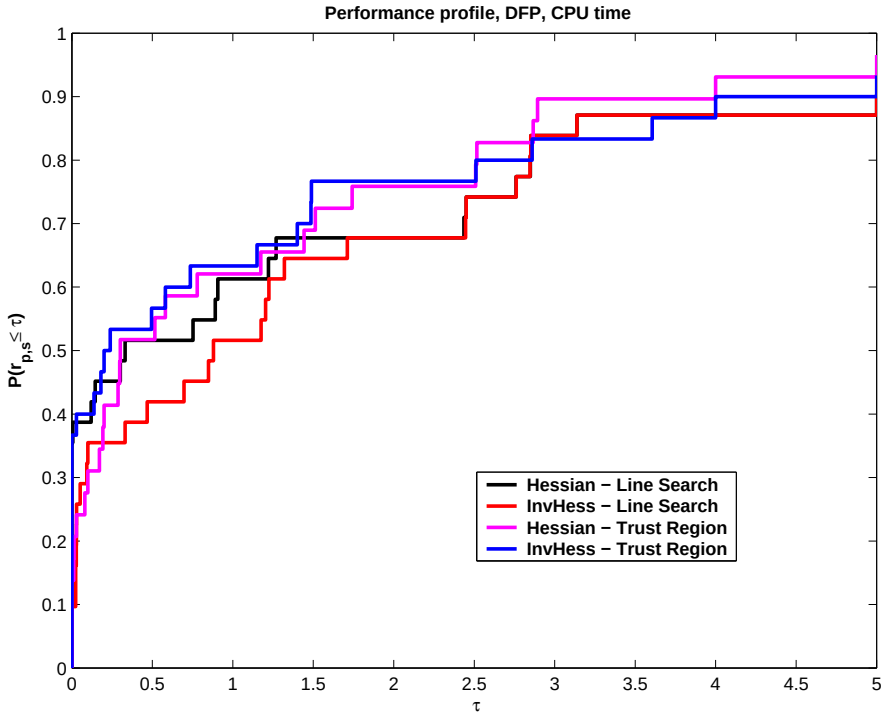

Figure 9: Performance profile for DFP with line search and Trust region

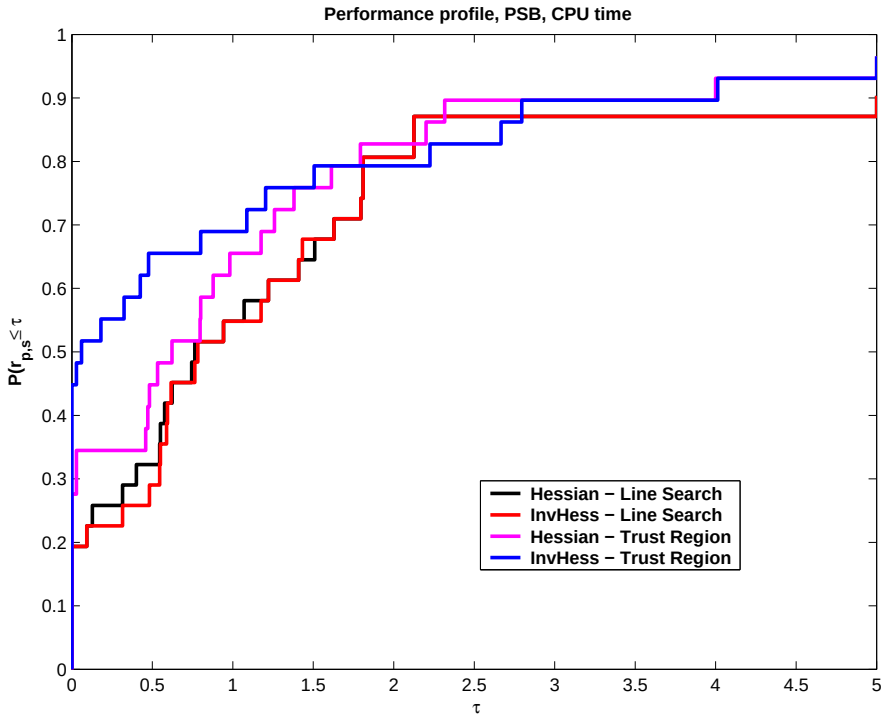

Figure 10: Performance profile for PSB with line search and Trust region

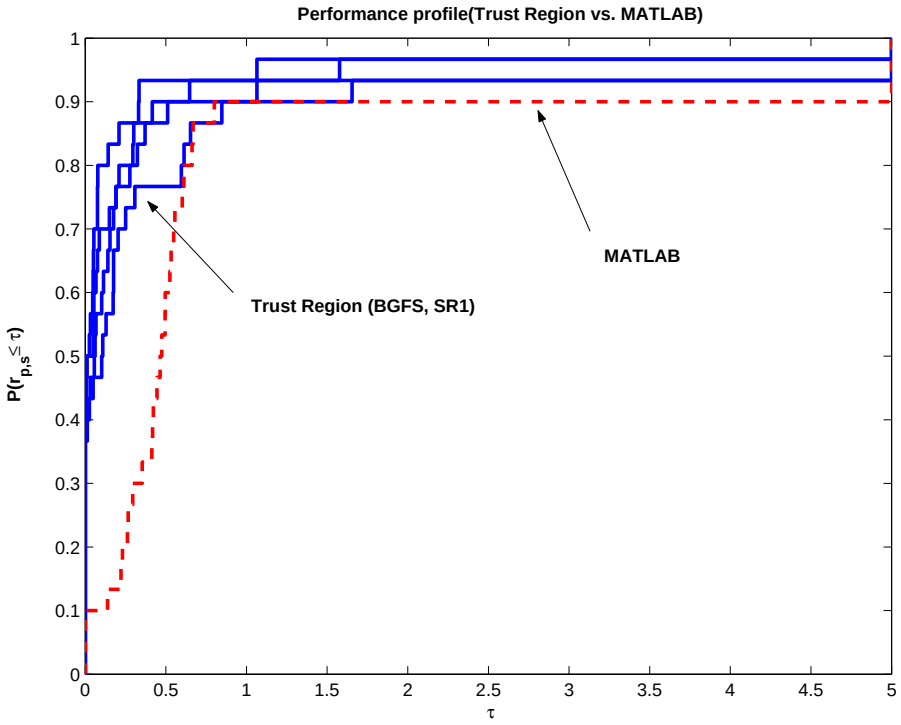

Figure 11: Performance profile for Trust region(BFGS, SR1) and MATLAB

0.5.4 Convergence of the Hessian approximations

In order to verify the covergence results of the approximated Hessian B_k to $\nabla^2 f(x^*)$, we run the trust region based quasi-Newton algorithms on the Rosenbrock function, and record the Frobenius norm $n_b = \|B_k - \nabla^2 f(x^*)\|$ for each iteration. The results, shown in Fig. 12, do not show convergence of B_k. We assume this happens because the assumption (A.4) may not be satisfied for this problem. We also try the SR1 on a quadratic function. We do not discover the convergece of B_k in this case either, as the serach directions s_k turn out to be linearly dependent. However the convergence of the BFGS, SR1, DFP and PSB to the solution x^* does not seem to be affected.

0.5.5 System of nonlinear equations

We test the quasi-Newton solvers for nonlinear equations on 12 problems taken from the benchmark list of problems, and the profiles are shown in Fig.13. The results tell us that the derivative free line search gives the best results, and simple backtracking is better than trust region. The backtracking fails on 3 problems, derivative free line search fails on 2, and trust region fails on 5 problems. Detailed results are listed in table 3,

0.6 Conclusions and Further Work

Based on our numerical results we conclude:

- The BFGS method is superior to the others, especially when a Wolfe Conditions based line search is used to compute the step length.

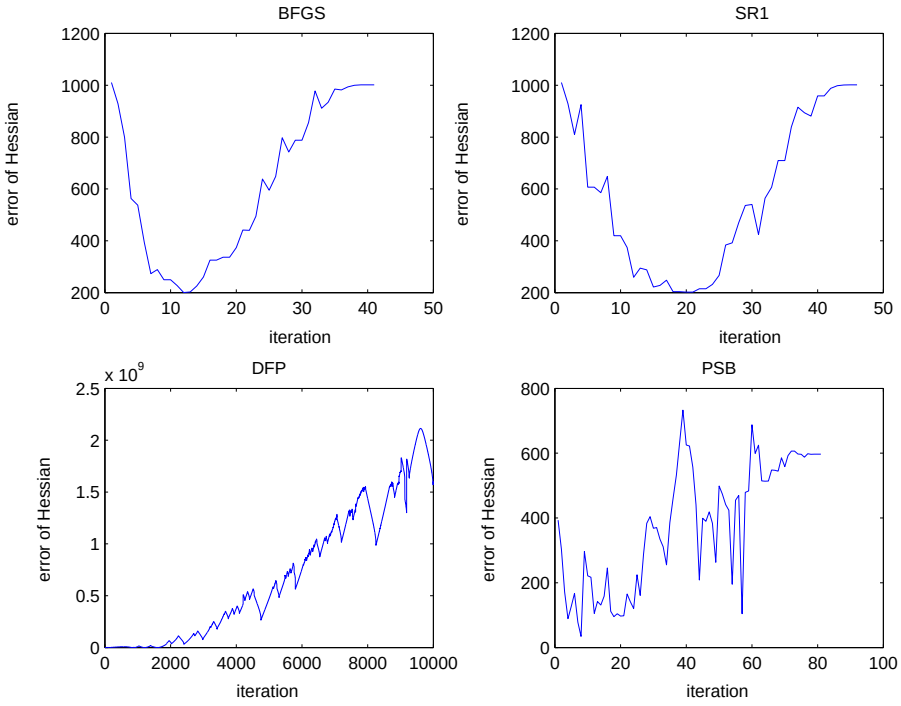

Figure 12: Hessian approximation error

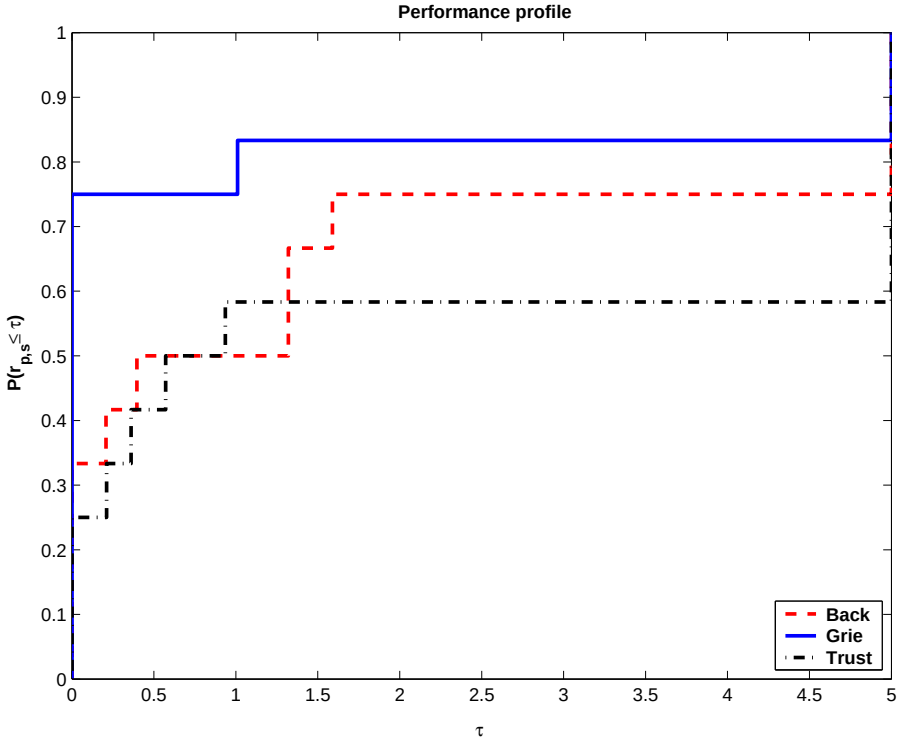

Figure 13: Performance profile for nonlinear equations

Function	Backtracking		Griewank		Trust Region	
Number	itrn.	CPU	itrn.	CPU	itrn.	CPU
1	230	20	11	5	95	96
4	10000	10000	34	15	10000	10000
5	10000	10000	78	33	10000	10000
6	10000	10000	3	2	10000	10000
7	162	23	666	255	65	66
12	1299	37	10000	10000	805	806
17	1048	8	10000	10000	27	28
18	230	20	11	5	10000	10000
19	37	19	37	19	10000	10000
23	17	9	17	9	39	40
24	35	18	35	18	57	58
28	11	6	11	6	41	42

Table 3: Results on solving nonlinear equations

- SR1 performs as good as BFGS when trust region is used.

- PSB and DFP perform well, although they are slower than BFGS and SR1. They could solve some problems from the benchmark set that the later could not, hence they should not be discarded as possible solvers on random problems.

- The trust region based methods give better convergence than line search methods for the benchmark problems. However, line search methods perform better in solving nonlinear equations. Hence the decision to use one or the other rests with the user and should be dependent on the problem to be solved.

- Some algorithms failed on some of the problems and some others got stuck on saddle points. This tells us it is better to design algorithms based on the specifics of the problem in question, as understanding of the structure of the problem is rather important.

- For the line search based BFGS and DFP methods, we should always try initial step length $\alpha_0 = 1$, as it will be accepted.

- In general, to solve some unconstrained minimization problem, it is a good idea to try BFGS first.

Further work we could have done:

- We could have tried more quasi-Newton update formulas from the Broyden class to get a better idea of these methods.

- We could have tried more line search strategies, like the golden section, Fibonacci search or quadratic interpolation Wolfe search or even Goldstein Conditions based line searches, and compared which performs better.

- We could have tested the algorithms on more general problems. The benchmark set of problems are quite a selected set, hence they might not give a good idea how the methods might work on a random problem.

- Better choices could have been used for the initial approximations of the Hessians, scaled versions of it could have avoided later breakdowns.

- We could have decreased the computational time for the Hessian update based methods, by using Cholesky factorization to get faster inversions of the matrices.

- We could have tried different parameters to see if they make it easier to find solutions

to the problems.

- We could have looked up other alternatives for updating B_{k+1} other than updates that come from $\min \|B - B_k\|$, if there is any.

Bibliography

[1] J.E. Dennis and R.B. Schnabel, Numerical methods for unconstrained optimization, Prentice-Hall, Englewood Cliffs, NJ, 1983

[2] A. R. Conn, N.I.M. Gould, P.L. Toint, Convergence of quasi-Newton matrices generated by the symmetric rank one update, Mathematical programming, 50:177-195, 1991.

[3] R.P. Ge and M.J.D. Powell, The convergence of variable metric matrices in the unconstrained optimization, Methmatical programming, 27:123-143, 1983.

[4] M.J.D. Powell,, A new algorithm for unconstrained optimization, In J.B. Rosen, *et. al* eds., Nonlinear programming, Academic press, New York, 1970

[5] R.H. Byrd, H.F. Khalfan, and R.B. Schnabel, Analysis of a symmetric rank-one trust region method, SIAM J. Optimization, 6(4):1025-1039, 1996

[6] M.J.D. Powell., How bad are the BFGS and DFP methds when the objective function is quadratic? Math. Programming 34:118-122, 1990

[7] Jorge Nocedal. Stephen J. Wright, Numerical optimization. New York : Springer, c1999

[8] A. Griewank, The 'global' convergence of Broyden-like methods with a suitable line search, J. of. Austral. Meth. Soc. Ser. B, 28:75-92, 1986.

[9] J.J. More, B. S. Garbow, and K.E. Hillstorm, Testing Unconstrained optimization software, ACM trans. Mathematical Software, 7(1):17-41, 1981.

[10] E.D. Dolan and J.J. More, Benchmarking optimization software with performance profiles, Mathematical Programing, 91(2):201-203, 2002.

[11] A. R. Conn, N.I.M. Gould, P.L. Toint, Testing a class of methods for solving minimization problems with simple bounds on the variables, Methematics of computation, 50(182), 399-430, 1988

[12] D.Pu., The convergence of DFP algorithm without exact linear search, J. Optim. Theory Appl. 112:187-211, 2002

[13] D.Pu.and W. Tian, the revised DFP algorithm without exact linear search, J. Computational and applied mathmatics, 154: 319-339, 2003

0.7 Appendix

The results for each of the trust region based methods are as follows:

Function	BFGS		SR1		DFP		PSB	
Number	itrn.	CPU	itrn.	CPU	itrn.	CPU	itrn.	CPU
1	15	0.343	19	0.344	18	0.344	19	0.365
2	47	0.016	54	0.016	10001	8.03	46	0.028
3	52	0.031	234	0.078	10001	10.935	215	0.085
4	41	0.016	168	0.047	4685	2.296	1266	0.435
5	238	0.671	438	1.796	10001	23.714	9143	7.211
6	6	0.031	6	0.032	6	0.031	6	0.028
7	30	0.016	45	0.016	5933	4.062	69	0.042
8	3	0	3	0	3	0	3	0
9	34	0.01	51	0.015	10001	10.232	46	0.015
10	41	0.015	19	0.016	10001	14.044	26	0.028
11	17	0.015	17	0	17	0	17	0.014
12	50	0.078	42	0.062	10001	42.632	5640	16.445
13	58	0.047	226	0.125	52	0.031	10001	20.83
14	33	0.015	35	0.015	10001	21.089	375	0.197
15	30	0.015	27	0.016	37	0.016	133	0.126
16	34	0.047	21	0.031	10001	15.091	21	0.028
17	33	0.015	34	0.016	83	0.047	30	0.014
18	106	0.063	178	0.078	10001	20.731	2605	2.165
19	61	0.031	76	0.032	382	0.203	2560	2.333
20	60	0.032	57	0.031	10001	14.779	10001	13.845
21	70	0.094	10001	83.355	2942	4.531	10001	26.86
22	20	0.015	32	0.014	309	0.141	38	0.028
23	37	0.015	260	0.098	22	0.015	41	0.014
24	32	0.016	65	0.028	48	0.031	10001	82.11
25	12	0	12	0	13	0	32	0.015
26	18	0.016	16	0	65	0.032	24	0.015
27	31	0.016	35	0.014	2617	1.5	90	0.06
28	25	0.016	18	0.014	82	0.047	43	0.03
29	5	0	5	0	5	0	5	0
30	10001	122.57	10001	58.639	10001	16.717	10001	10.419

Table 4: The number of iterations and CPU time for trust region based Hessian update methods

Function	BFGS		SR1		DFP		PSB	
Number	itrn.	CPU	itrn.	CPU	itrn.	CPU	itrn.	CPU
1	15	0.374	19	0.359	18	0.343	23	0.359
2	42	0.031	47	0.016	10001	8.544	969	0.407
3	53	0.031	231	0.094	10001	11.011	572	0.265
4	41	0.015	81	0.046	10001	16.18	56	0.031
5	261	0.812	360	0.859	10000	10	10001	8.856
6	7	0.046	7	0.031	7	0.032	7	0.031
7	30	0.015	45	0.031	1911	0.968	123	0.078
8	3	0	3	0	3	0	3	0
9	34	0.031	47	0.031	10001	10.574	1340	0.719
10	40	0.031	19	0.016	10001	13.025	10001	14.401
11	17	0.031	17	0.016	17	0.016	17	0.015
12	47	0.094	42	0.047	10001	44.934	9926	45.888
13	58	0.046	189	0.141	52	0.032	10001	22.522
14	86	0.062	38	0.031	10001	20.82	10001	21.148
15	33	0.047	28	0.015	34	0.016	10000	10
16	34	0.046	21	0.047	10001	15.619	21	0.031
17	33	0.015	34	0.016	83	0.063	165	0.109
18	100	0.063	189	0.109	10001	20.492	425	0.297
19	72	0.047	51	0.032	1884	1.718	4300	5.623
20	60	0.063	53	0.031	10001	17.743	10001	13.635
21	65	0.079	74	0.11	1041	1.39	10001	28.208
22	20	0.016	31	0.016	309	0.187	10001	10.34
23	38	0.047	48	0.031	23	0.015	46	0.031
24	36	0.032	64	0.047	36	0.031	32	0.031
25	12	0	12	0	13	0	27	0.016
26	10000	10	16	0.031	65	0.031	17	0.016
27	31	0.015	37	0.016	2978	2.031	10001	11.339
28	28	0.031	18	0.031	82	0.047	369	0.25
29	5	0	5	0	5	0	5	0
30	10000	10	10000	10	10000	10	10001	11.105

Table 5: The number of iterations and CPU time for trust region based inverse Hessian update methods