sciendo
Control and Cybernetics
vol. **52** (2023) No. 3
pages: 269-289
DOI: 10.2478/candc-2023-0039

Empirical analysis of different hybridization strategies for solving systems of nonlinear equations*

by

Bishwesvar Pratap Singh, Marko M. Mäkelä and Yury Nikulin[1]

University of Turku, Department of Mathematics and Statistics,
20014 Vesilinnantie 5, Turku, Finland
[1]corresponding author: yurnik@utu.fi

Abstract: This paper considers various hybrid methods for solving a system of nonlinear equations. Extensive numerical experiments confirm the research hypothesis that one particular form of hybridization of the conjugate gradient algorithm with Newton's method outperforms other hybrid strategies considered in the paper.

Keywords: Newton's method, iterative algorithm, descent direction, conjugate gradient, nonlinear optimization

1. Introduction

The problem of solving a system of equations is central in the field of applied mathematics. Many robust and efficient techniques exist to solve systems being linear or differentiable. Much progress has been made in generalizing classical Newton's method to solve such systems, and these developed techniques are attractive since they maintain fast local convergence properties that are characteristics of Newton-based methods for smooth equations, see Atkinson (1978). Newton's method can converge rapidly once a good approximation is obtained that is near the true solution. However, determining good starting values is not always easy, and the method is comparatively expensive to employ (Burden, Faires, 2010).

One of the earliest alternative methods to the Newton-based method was the conjugate gradient (CG) method. This approach was first proposed by Hestenes and Stiefel in 1952 (Hestenes and Stiefel, 1952). The CG algorithm is based on a series of conjugate directions generated in such a way that for example in linear cases they are mutually orthogonal. This allows for efficient convergence towards

*Submitted: February 2024; Accepted: June 2024.

the solution of the linear system. However, it was not widely used until the early 1960s, when Fletcher and Reeves (1964) improved the algorithm. In this variant, the definition of conjugacy is slightly altered. Instead of seeking exact conjugacy, it allows for satisfaction of a more relaxed criterion for conjugacy. This improved convergence behaviour can be especially beneficial when dealing with ill-conditioned or nearly linearly dependent systems.

In general, systems of nonlinear equations can indirectly be solved as unconstrained optimization problems by transforming them into a single objective unconstrained optimization problem (see Taheri and Mammadov, 2012). In the late 1960s and early 1970s the subject became more popular when researchers began to explore the combination of different optimization techniques to improve performance. Such a combination is generally termed hybridization. The concept of hybridization involves integrating two or more optimization methods, algorithms, or strategies to create a new approach that takes advantage of the strengths of each component.

Powell (1970) proposed a hybrid method that combines the conjugate gradient method and the Newton-Raphson method. This hybrid method was used to solve unconstrained optimization problems and was shown to be more efficient than individual methods, particularly for solving nonlinear optimization problems with many variables. In recent years, researchers have continued developing new hybrid methods for solving unconstrained optimization problems (see Shi, 2000; Taheri and Mammadov, 2012).

In this paper, we are mainly focused on solving a system of nonlinear equations as an unconstrained optimization problem with the combination of the Gradient (G), Conjugate Gradient (CG), Newton's (N) and Quasi-Newton (QN) method. These methods are organized into two groups, one comprising Gradient and Conjugate Gradient methods, and the other including Newton's and Quasi-Newton methods. Our approach entails selecting one method from each group, resulting in four possible combinations for tackling the problem. The hybridization of the Conjugate Gradient and Newton's methods (CGN) gives rise to a powerful optimization algorithm that combines the strengths of both the Conjugate Gradient and Newton's methods. Particularly, it is useful for nonlinear optimization problems with many variables. However, it is computationally expensive for some problems due to the need of calculating the Hessian matrix at each iteration.

The paper is organized as follows. In Section 2, we convert a constraint satisfaction problem containing a system of nonlinear equations into an unconstrained optimization problem, and describe classical gradient-based iterative optimization approaches. Section 3 highlights the hybridization technique focusing on the conjugate gradient and Newton's methods combination. Section 4 lists all combinations of methods considered in our empirical study. Section 5

presents the benchmarks and results of numerical experiments. Section 6 uses rank analysis to find the best hybridization strategies. Section 7 checks the statistical validity of the conclusion made in the previous section. Section 8 summarizes information about convergence. Section 9 visualizes some of the most typical computational results obtained. The paper ends with the conclusion given in Section 10.

2. Preliminaries

2.1. The formulation

In the last decades, many constructive algorithms have been designed to solve systems of equations precisely, see, e.g., Buckley (1978) and Kelley (1995). Solving the system of nonlinear equations is a relatively challenging problem because of computational difficulties related to nonlinearity (Nedzhibov, 2008). It is well known that Newton's method is one of the most traditional methods for solving either one or a system of equations, but its convergence is very sensitive to the initial solution. To overcome the difficulties of finding a good quality initial solution and achieve faster convergence, many adaptive hybrid approaches have been proposed (see, in particular, Buckley, 1978; Taheri and Mammadov, 2012). Some of these algorithms are complicated and costly (high processing time, demanding computation power) but are still widely used because computing systems are becoming more and more computationally affordable nowadays.

We consider a system of nonlinear equations

$$\boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} F_1(\boldsymbol{x}) \\ F_2(\boldsymbol{x}) \\ \vdots \\ F_m(\boldsymbol{x}) \end{pmatrix} = \mathbf{0}, \tag{1}$$

where $F_1, F_2, \ldots, F_m : \mathbb{R}^n \to \mathbb{R}$ are nonlinear and twice continuously differentiable functions. Many optimization-based methods (see, e.g., Shi, 2000; Taheri and mammadov, 2012) have been developed for solving (1). In an optimization-based approach, the constraint satisfaction problem (1) can be reformulated as the following optimization problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) := \min_{\boldsymbol{x}\in\mathbb{R}^n} \frac{1}{2}\|\boldsymbol{F}(\boldsymbol{x})\|^2 \tag{2}$$

where $\|\cdot\|$ means Euclidean norm. If the minimum in (2) exists with $f(\boldsymbol{x}) = 0$, then it corresponds to the solution of (1). In this paper, we propose a hybrid algorithm to solve the system of nonlinear equations. The suggested

algorithm is a combination of the Newton method and the Conjugate Gradient method. We used the Newton method to increase the convergence rate and the Conjugate Gradient method to satisfy global convergence and shallow memory requirements. The step length is determined in both the Newton and Conjugate Gradient directions.

It is well known that, with a given starting point $\boldsymbol{x}_1$, optimization methods generate a sequence of estimates $\boldsymbol{x}_k$ at each iteration until a solution, featuring a tolerance less than a predetermined one, is reached. The main aspect of behaviour of the algorithm is that at every iteration k the currently established solution $\boldsymbol{x}_k$ moves steadily towards the neighbourhood of a local minimizer, while keeping certain that ultimately it satisfies the following stopping criterion

$$\|\boldsymbol{g}_k\| < \epsilon,$$

where $\boldsymbol{g}_k$ is the gradient $\nabla f(\boldsymbol{x}_k)$ and ϵ is a predefined positive tolerance parameter. Let $\boldsymbol{x}_k$ be the solution, $\boldsymbol{d}_k$ be the search direction, and λ_k be the step length at k-th iteration. At $(k+1)$-th iteration, the solution can be obtained by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k.$$

There have been multiple strategies proposed for moving from the current iterate to the next iterate, but two commonly accepted strategies are the so-called trust region and the line search. This paper uses only the line search strategy to choose with respect to a search direction $\boldsymbol{d}_k$, which produces a lower function value in a new iteration.

2.2. Search direction

The search direction $\boldsymbol{d}_k$ for the general gradient-based methods can be defined as

$$\boldsymbol{d}_k = -B_k^{-1}\boldsymbol{g}_k,$$

where B_k is a nonsingular and symmetric matrix. For Newton's method,

$$\boldsymbol{d}_k = -H_k^{-1}\boldsymbol{g}_k \tag{3}$$

where H_k is the exact Hessian of $\boldsymbol{F}$ at $\boldsymbol{x}_k$. For the Quasi-Newton method, B_k approximates Hessian, and H_k is updated at every iteration. In the Conjugate Gradient method

$$\boldsymbol{d}_k = \begin{cases} -\boldsymbol{g}_k, & k = 1 \\ -\boldsymbol{g}_k + \beta_k \boldsymbol{d}_{k-1}, & k \geq 2, \end{cases} \tag{4}$$

where β_k is the coefficient of the Conjugate Gradient method. In this paper, we are using the Fletcher-Reeves method (Fletcher and Reeves, 1964) to calculate β_k

$$\beta_k = \frac{\|\boldsymbol{g}_k\|^2}{\|\boldsymbol{g}_{k-1}\|^2}.$$

Generally, the search direction $\boldsymbol{d}_k$ needs to satisfy

$$\boldsymbol{g}_k^T \boldsymbol{d}_k < 0,$$

and then it ensures that $\boldsymbol{d}_k$ is the descent direction at $\boldsymbol{x}_k$.

2.3. Line search

Line search is an umbrella term, which refers to a class of one of the most effective conventional routines incorporated inside optimization methods. The line search tries to find the optimal step length λ_k along the search direction after this direction $\boldsymbol{d}_k$ is specified and fixed. The accuracy and speed of optimization methods depend on search direction $\boldsymbol{d}_k$ and step length λ_k. The direction vector $\boldsymbol{d}_k$ decides in which direction the search moves at the current iteration, and λ_k decides how far it moves along $\boldsymbol{d}_k$ in this iteration. There are two types of methods to find the step length - the exact line search and the inexact line search.

To find λ_k using the exact line search we need to solve

$$\min_{\lambda_k > 0} f(\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k).$$

On the other hand, choosing step length λ_k such that it decreases function value at the next iterate such that

$$f(\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k) < f(\boldsymbol{x}_k) \tag{5}$$

is called inexact line search. In multidimensional real-world problems, finding the exact step length is difficult, so inexact line search is quite popular. It is also computationally less expensive to perform.

Note that the decreasing property of function value, in other words (5), is not sufficient for the optimization method to converge to its solution, as sometimes the decrease in function value is not sufficient. Hence the idea of the inexact line search is to move sufficiently in the direction of $\boldsymbol{d}_k$. There is a variety of inexact methods to estimate a suitable step length λ_k such as Armijo's, Goldstein's and Wolfe-Powell's rules.

2.4. Wolfe-Powell rule

The Wolfe-Powell rule states that the step length λ_k along the direction $\boldsymbol{d}_k$, chosen at each iteration, must satisfy

$$f(\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) + \rho \lambda_k \boldsymbol{g}_k^T \boldsymbol{d}_k, \quad \rho \in (0, 1) \tag{6}$$

and

$$\boldsymbol{g}_{k+1}^T \boldsymbol{d}_k \geq \sigma \boldsymbol{g}_k^T \boldsymbol{d}_k, \quad \sigma \in (\rho, 1). \tag{7}$$

Condition (6) ensures that if the algorithm is taking larger steps, then it produces a larger decrease in function value, in simple words it keeps an upper limit on step length λ, while (7) makes sure that the algorithm does not take a too short step.

This work uses the perturbed Wolfe condition where we start with $\lambda = 1$ and decrease λ by some small quantity until the largest value that satisfies the condition (6) is found.

3. The CGN algorithm

This section presents a hybrid method, which consists of a combination of the Conjugate gradient (CG) and Newton's (N) methods. We denote the combination of the CG and N methods as CGN. Here and later on, $\boldsymbol{d}_{1,k}$ represents the Newton's direction (3) at $\boldsymbol{x}_k$, and $\boldsymbol{d}_{2,k}$ represents the conjugate gradient direction (4) at $\boldsymbol{x}_k$.

Here, δ_0, η, ρ, σ are parameters such that $0 < \delta_0 < 1$, $0 < \eta < 1$, $0 < \rho < \frac{1}{2}$, $\rho < \sigma < 1$. Furthermore, $\gamma_1 > 1$, $\gamma_2 > 1$, Λ_0 and b_i, $i = 1, 2, 3$, are positive constants.

In the CGN algorithm's Step 3, at any iteration $\boldsymbol{x}_k$, if $\det(H_k) = 0$, then Newton's direction is not computable because of singularity, we proceed only with conjugate gradient direction and step length λ_k, calculated at Step 9, and next iteration point $\boldsymbol{x}_{k+1}$ is calculated. When both directions are available, we calculate them and check if both simultaneously give a descent direction at Step 8. If the condition is satisfied, the process of hybridization starts, otherwise we continue in a conjugate gradient direction.

Algorithm 1: Conjugate Gradient Newton (CGN)

Input: Starting point $\boldsymbol{x}_1$, maximum number of iterations N, tolerance $\epsilon > 0$, initial values δ_0 and Λ_0 of modifiable parameters δ and Λ, fixed value parameters γ_1, γ_2, η, ρ, b_1, b_2, b_3, τ, T.

1 While $(k < N)$

2 If $\|\boldsymbol{g}_k\| < \epsilon$ then $\boldsymbol{x}_k$ is the solution and go to step 16.

3 If $\det(H_k) = 0$ compute direction $\boldsymbol{d}_{2,k} = -\boldsymbol{g}_k$ and go to step 9.

4 If $\det(H_k) \neq 0$ compute Newton's direction $\boldsymbol{d}_{1,k}$ that satisfies $H_k \boldsymbol{d}_{1,k} = -\boldsymbol{g}_k$ and conjugate gradient direction

$$\boldsymbol{d}_{2,k} = \begin{cases} -\boldsymbol{g}_k & k = 1 \\ -\boldsymbol{g}_k + \lambda_k \boldsymbol{d}_{2,k-1} & k \geq 2. \end{cases}$$

5 Set $\delta = \delta_0$ and $\Lambda = \Lambda_0$. If $|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})| > \gamma_1$ and $\|\boldsymbol{g}_k\| > \gamma_2$ set $\delta = b_2 \delta_0$ and go to step 8.

6 If $k = 1$ or if $\|\boldsymbol{g}_k\| \leq \|\boldsymbol{g}_{k-1}\|$ set $\bar{\boldsymbol{x}} = \boldsymbol{x}_k + \boldsymbol{d}_{1,k}$.

7 If $f(\bar{\boldsymbol{x}}) < f(\boldsymbol{x}_k)$ and $\|\nabla f(\bar{\boldsymbol{x}})\| < \eta \|\boldsymbol{g}_k\|$, set $\delta = b_1 \delta_0$.

8 If $\boldsymbol{d}_{1,k}^T \boldsymbol{d}_{2,k} \geq 0$ go to step 11.

9 Use the Wolfe-Powell rule (6)-(7) to get $\lambda_k > 0$ along direction $\boldsymbol{d}_k = \boldsymbol{d}_{2,k}$.

10 Calculate $\boldsymbol{s}_k = \lambda_k \boldsymbol{d}_k$ and go to step 14.

11 Calculate ξ such that

$$\xi = \begin{cases} \frac{1}{\Lambda + \|\boldsymbol{g}_k\|} & k = 1 \\ \frac{1}{\Lambda + |f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})|} & k \geq 2. \end{cases}$$

and set $\boldsymbol{d}_k(\xi) = (1 - \xi)\boldsymbol{d}_{2,k} + \xi \boldsymbol{d}_{1,k}$.

12 If $\boldsymbol{d}_k(\xi)^T \boldsymbol{d}_{2,k} < \delta \|\boldsymbol{d}_k(\xi)\| \cdot \|\boldsymbol{d}_{2,k}\|$ set $\Lambda = b_3 \Lambda$ and go to step 11.

13 (A) Use the Wolfe-Powell rule to get $\lambda_k > 0$ along direction $\boldsymbol{d}_k = \boldsymbol{d}_{2,k}$, set $\bar{\boldsymbol{s}}_k = \lambda_k (1 - \xi)\boldsymbol{d}_{2,k} + \xi \boldsymbol{d}_{1,k}$. If $f(\boldsymbol{x}_k + \bar{\boldsymbol{s}}_k) \leq f(\boldsymbol{x}_k) - \tau \|\bar{\boldsymbol{s}}_k\|$ and $\lambda_k \|\boldsymbol{d}_{2,k}\| \leq T \|\boldsymbol{d}_{1,k}\|$ set $\boldsymbol{s}_k = \bar{\boldsymbol{s}}_k$ otherwise $\boldsymbol{s}_k = \lambda_k \boldsymbol{d}_{2,k}$.
(B) Use the Wolfe-Powell rule to get $\lambda_k > 0$ along direction $\boldsymbol{d}_k = \boldsymbol{d}_k(\xi)$ and calculate $\boldsymbol{s}_k = \lambda_k \boldsymbol{d}_k$.

14 Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$, $k = k + 1$ and go to step 1.

15 OUTPUT ('Method failed to converge after N iterations')

16 Stop

In steps 11-12, first, we calculate a parameter ξ, which is the coefficient of the convex combination of $\boldsymbol{d}_{1,k}$ and $\boldsymbol{d}_{2,k}$, then we check that the hybrid direction deviates enough from the conjugate gradient direction, and if not, then we increase the preference of $\boldsymbol{d}_{1,k}$ in the convex combinations in step 11, where the hybrid direction is calculated. In Step 13, two different versions of calculating the next iteration point are considered. Step 13A determines step length λ_k in the conjugate gradient direction only. In step 13B, step length λ_k is determined in the convex combination of both conjugate gradient direction and Newton's direction.

Constant parameters γ_1 and γ_2 (Step 5) are threshold values of the vertical slope and gradient. The constant η defines a suitable portion of the gradient in Step 7. Constant parameters τ and T magnify the norm values of directional vectors in Step 13. Constant parameter ρ is used to specify the lower bound for the parameter σ used in the Wolfe-Powell rule (6)-(7).

Non-modifiable parameters b_1 (Step 7), b_2 (Step 5), and b_3 (Step 12) are positive constants offering the opportunity to specify the amount of contribution of the methods. More precisely, when the slope of the function is slight, the algorithm tends towards the Newton's method, otherwise the contribution of the conjugate gradient is increased.

4. Combinations of methods

In this work, we propose to consider other combinations of different methods. Expanding upon the combination methodology originally delineated by Taheri and Mammadov (2012), we have extended this framework to incorporate various combinations of the Gradient (G) method, Conjugate Gradient (CG) method, Newton's (N) method, and Quasi-Newton's (QN) method.

Table 1: Various hybridizations considered

Method Group 1	Method Group 2	Direction	Method name
Conjugate Gradient (CG)	Newton's (N)	Conjugate gradient direction (A)	CGN(A)
Conjugate Gradient (CG)	Newton's (N)	Hybrid direction (B)	CGN(B)
Conjugate Gradient (CG)	Quasi Newton's (QN)	Conjugate Gradient direction (A)	CGQN(A)
Conjugate Gradient (CG)	Quasi Newton's (QN)	Hybrid direction (B)	CGQN(B)
Gradient (G)	Newton's (N)	Gradient direction (A)	GN(A)
Gradient (G)	Newton's (N)	Hybrid direction (B)	GN(B)
Gradient (G)	Quasi Newton's (QN)	Gradient direction (A)	GQN(A)
Gradient (G)	Quasi Newton's (QN)	Hybrid direction (B)	GQN(B)

The GN method shares analogous steps with the CGN algorithm described earlier. However, in the GN method, at the fourth step of the Algorithm, the direction vector is chosen as antigradient, i.e. $\boldsymbol{d}_{2,k} = -\boldsymbol{g}_k$ for all values of k. In

other words, the hybrid direction $\boldsymbol{d}_k(\xi)$ is derived as a convex combination of the Gradient and Newton's directions.

Conversely, in the GQN method, the hybrid direction $\boldsymbol{d}_k(\xi)$ is also a convex combination of the Gradient and Newton's directions. However, in place of the Hessian matrix H_k, the method employs an approximated matrix B_k, which is iteratively updated using the BFGS (Broyden-Fletcher-Goddfarb-Shanno, Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) method.

In the case of the CGQN method, both directions $\boldsymbol{d}_{1,k}$ and $\boldsymbol{d}_{2,k}$ are involved. The hybrid direction $\boldsymbol{d}_k(\xi)$ is computed as a convex combination of the Conjugate Gradient and Newton's directions.

5. Numerical experiments and results

In this section, we show the computational performance of the above algorithms on a series of test problems, discussed in Taheri and Mammadov (2012) and Andrei (2008). Table 1 gives the basic problem description.

The calculations were carried out in MATLAB. For termination criteria, we are using the predefined tolerance $\|\boldsymbol{g}_k\| < \epsilon$ where $\epsilon = 10^{-6}$ or the user-defined maximum number of iterations ($N = 500$).

The user-defined parameters for the experiment are as follows: $\delta_0 = 0.001$, $\Lambda_0 = 1$, $\eta = 0.99$, $\rho = 0.001$, $\sigma = 0.9$, $b_1 = 0.9$, $b_2 = \frac{1}{b_1}$, $b_3 = 1.1$, $\gamma_1 = 2$, $\gamma_2 = 2$, $\tau = 10^{-10}$ and $T = 10^{10}$.

Table 2: Test problems after Taheri and Mammadov (2012) and Andrei (2008)

Problem	Function name	Dimension n	Number of functions m
P00	Linear functions	21	21
P01	Linear functions	21	21
P1	Helical Valley function	3	3
P2	Powell Singular function	4	4
P3	Wood function	4	6
P4	Watson function	6	31
P5	Variably dimensional function	n	n+2
P6	Discrete Boundary Value function	n	n
P7	Extended Rosenbrock function	n	n
P8	Trigonometric function	n	n
P9	Axis Parallel Hyper-Ellipsoid function	n	n
P10	Griewangk's function	n	n
P11	Sum of Different Power functions	n	n
P12	Ackley's function	n	n

In Table 3, we compare the numbers of iterations for GN, CGQN, CGN and GQN. We choose either G or CG methods as well as either N or QN methods for all possible combinations. They are kept in two groups because of their similarity. Each method has two different strategies for the hybridization, referred to as A and B, as this is shown in Table 1. In the algorithm version A, the step length λ_k is calculated only in $\boldsymbol{d}_{2,k}$ direction, while in version B, the step length λ_k is calculated in direction of $\boldsymbol{d}_k(\xi)$, which is a convex combination of directions $\boldsymbol{d}_{1,k}$ and $\boldsymbol{d}_{2,k}$.

For problems P1-P6, there is a fixed dimension, while for problems P7-P12, the dimension n is variable, so one can choose different numerical values of n. In our experiments, we set $n = 20, 50$, and 100. Each problem with a different value of n is considered to be a separate test problem. The number of iterations in bold font shows the minimum number of iterations for that test problem.

Table 3: Number of iterations after which the method converges

Problem	n	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
P00	21	14	**13**	*	*	15	22	*	41
P01	21	**3**	**3**	*	*	**3**	**3**	*	33
P1	3	**17**	55	*	*	19	28	31	41
P2	4	**20**	**20**	27	*	**20**	26	46	54
P3	4	29	47	*	*	**10**	63	*	88
P4	6	**11**	12	*	*	12	22	27	36
P5	20	8	10	12	10	**7**	9	9	9
	50	**9**	14	15	25	**9**	15	12	15
	100	11	18	12	26	**9**	11	**9**	11
P6	20	**2**	**2**	*	43	**2**	**2**	95	40
	50	**1**	**1**	*	*	**1**	**1**	*	106
	100	**1**	**1**	*	*	**1**	2	2	187
P7	20	27	68	*	*	**17**	48	*	140
	50	**20**	448	*	*	*	67	*	153
	100	**22**	35	*	*	*	65	*	172
P8	20	11	21	185	252	**10**	12	17	33
	50	22	39	*	*	*	**15**	19	44
	100	*	33	*	*	**20**	*	28	*
P9	20	18	19	47	55	**14**	**14**	116	81
	50	21	23	79	122	**15**	**15**	167	96
	100	125	**93**	178	166	*	*	136	350
P10	20	**9**	12	*	45	13	14	137	137
	50	17	34	*	115	**16**	**16**	195	211
	100	*	*	*	*	*	*	**239**	263
P11	20	20	*	**5**	8	14	*	86	138
	50	*	*	**5**	**5**	*	*	44	47
	100	*	*	**5**	**5**	*	*	35	83
P12	20	**4**	7	53	47	5	7	5	7
	50	**3**	5	34	43	**3**	5	5	5
	100	**4**	5	28	33	**4**	5	5	5

* means the method failed to converge after a specified number of iterations.

The number of iterations performed can depend on the design of the code, so we have also included the total number of function evaluations in Table 4

to analyze the efficiency of each algorithm more precisely. The total number of function evaluations originates from both iterative and step-length computations.

Table 4: Total number of function evaluations

Prb	n	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
P00	21	73	63	*	*	**56**	57	*	104
P01	21	12	**5**	*	*	9	**5**	*	73
P1	3	**84**	336	*	*	93	103	150	143
P2	4	78	**77**	107	*	77	79	205	142
P3	4	204	311	*	*	**51**	239	*	334
P4	6	54	59	*	*	**52**	73	132	100
P5	20	66	61	96	61	**60**	**60**	70	**60**
	50	95	115	153	218	**94**	99	132	99
	100	135	161	142	247	113	110	114	**107**
P6	20	5	4	*	103	**3**	4	318	94
	50	3	**2**	*	*	3	**2**	*	249
	100	3	**2**	*	*	3	**2**	*	449
P7	20	195	404	*	*	**89**	169	*	419
	50	**116**	9244	*	*	*	256	*	518
	100	**126**	182	*	*	*	250	*	648
P8	20	61	119	1112	1553	50	**45**	85	131
	50	188	206	*	*	*	**71**	114	208
	100	*	214	*	*	**140**	*	196	*
P9	20	54	57	105	157	**38**	**38**	242	172
	50	71	76	191	278	**45**	**45**	349	207
	100	307	**236**	446	388	*	*	320	725
P10	20	**26**	50	*	101	28	30	276	276
	50	96	178	*	427	**36**	**36**	395	427
	100	*	*	*	*	*	*	**491**	538
P11	20	43	*	**19**	22	31	*	175	279
	50	*	*	15	**11**	*	*	91	98
	100	*	*	15	**11**	*	*	74	172
P12	20	**16**	18	109	99	20	17	20	17
	50	**9**	11	71	92	**9**	11	15	11
	100	12	**11**	60	73	12	**11**	15	**11**

* means the method failed to converge after a specified number of iterations.

6. Analysis of the results

6.1. Evaluation of the results

In Table 3 we see that there are some places where the method given failed to converge (marked by *). In this situation, it is not informative to perform analysis directly on the number of iterations. Therefore, we introduce a special ranking system as summarized in Table 5.

The algorithm with the least number of iterations has the highest rank, starting with the lowest number 1. In case two or more algorithms have the same number of iterations, they receive the same ranking which is the mean of

Table 5: Ranks based on the number of iterations for the test problems

Prb	n	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
P00	21	2	**1**	11	11	3	4	9	5
P01	21	**2.5**	**2.5**	11	11	**2.5**	**2.5**	10	5
P1	3	**1**	6	11	10	2	3	4	5
P2	4	**2**	**2**	5	10	**2**	4	6	7
P3	4	2	3	11	11	**1**	4	11	5
P4	6	**1**	2.5	10	11	2.5	4	5	6
P5	20	2	6.5	8	6.5	**1**	4	4	4
	50	**1.5**	4	6	8	**1.5**	6	3	6
	100	4	7	6	8	**1.5**	4	**1.5**	4
P6	20	**2.5**	**2.5**	11	6	**2.5**	**2.5**	7	5
	50	**2.5**	**2.5**	11	11	**2.5**	**2.5**	10	5
	100	**1.5**	**1.5**	10	10	**1.5**	4	10	5
P7	20	2	4	11	11	**1**	3	11	5
	50	**1**	4	11	10	11	2	11	3
	100	**1**	2	11	11	11	3	11	4
P8	20	2	5	7	8	**1**	3	4	6
	50	3	4	10	10	10	**1**	2	5
	100	10	3	11	10	**1**	10	2	4
P9	20	3	4	5	6	**1.5**	**1.5**	8	7
	50	3	4	5	7	**1.5**	**1.5**	8	6
	100	2	**1**	5	4	10	10	3	6
P10	20	**1**	2	11	5	3	4	6.5	6.5
	50	3	4	11	5	**1.5**	**1.5**	6	7
	100	10	10	11	11	10	10	**1**	2
P11	20	4	9	**1**	2	3	10	5	6
	50	9	9	**1.5**	**1.5**	10	10	3	4
	100	9	9	**1.5**	**1.5**	10	10	3	4
P12	20	**1**	5	8	7	2.5	5	2.5	5
	50	**1.5**	4.5	7	8	**1.5**	4.5	4.5	4.5
	100	**1.5**	4.5	7	8	**1.5**	4.5	4.5	4.5

what they would have under ordinal ranking. By this way of ranking, we ensure that the sum of ranks remains the same as under ordinal ranking. Given that we evaluate altogether 8 methods, we assigned rank 9 to the methods when they converge within 750 iterations. Then, we assign rank 10 if the method solves the problem with tolerance reduced to 10^{-3}. If the method is not capable of finding an optimal solution either with an increased number of iterations or with a reduced value of tolerance, then we assign the lowest rank to it, in other words, the largest possible ranking value, 11.

6.2. Rank analysis

Table 6 presents the summary of ranking results, given before in Table 5.

Clearly, from Table 6 we can say that CGN(A) is the most efficient method with the least average number of iterations. From Table 4, we see that GQN(B)

Table 6: Weighted ranking averaged

Parameter	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
Average	**3.05**	4.30	8.10	7.88	3.82	4.63	5.88	5.05
Variance	7.33	6.11	10.72	9.05	13.59	8.69	10.68	**1.37**
STD	2.66	2.43	3.22	2.96	3.63	2.90	3.21	**1.15**

is the most consistent, since it solves almost all the problems except P8 when $n = 100$ and has the least variance and standard deviation values.

We can choose a weighted ranking for our conclusions, so if a method fails to solve even a small problem, when we deal with analogous problems of variable dimensions, it is penalized more. We chose three weighted iteration parameters (WIP)

$$\text{wip}_1 = 0.5,\ \text{wip}_2 = 0.3,\ \text{wip}_3 = 0.2$$

to get the final weighted iterative ranking (WIR) for a problem. Let us take CGN(A) for P5 from Table 5 and calculate the weighted rank

$$\text{WIR} = 0.5 \times 2 + 0.3 \times 1.5 + 0.2 \times 4 = 2.25.$$

In a similar way, we can calculate the weighted ranking for the number of function evaluations ranking (WFR) for P5 with weighted function parameters (WFP)

$$\text{wfp}_1 = 0.5,\ \text{wfp}_2 = 0.3,\ \text{wfp}_3 = 0.2,$$

and

$$\text{WFR} = 0.5 \times 6 + 0.3 \times 2 + 0.2 \times 5 = 4.6.$$

In the final Grand Rank (GR), we used the equal weight parameters wip $= 0.5$, wfp $= 0.5$ with WIR and WFR parameters (wip and wfp can be determined by users based on their priorities). Thus, we get

$$\text{GR} = 0.5 \times 2.25 + 0.5 \times 4.6 \approx 3.43.$$

From the final GR (Table 7), we can state that CGN(A) is the most reliable and efficient method in these test problems. In the final grand rank (GR) we used, as mentioned above, the equal weight parameters wip $= 0.5$, wfp $= 0.5$, but we decided to check the stability of results with the different convex combinations.

Now, let us take P5 data and generate a Grand Ranking table for all weight combinations from Table 8 using the formula for GR calculation as specified before.

Table 7: Equally weighted Grand Ranks for test problems

Prb	CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
P00	3	**2**	9.5	9.5	3	**2**	9	5
P01	3	**2**	9.5	9.5	**2**	3	8.5	5
P1	**1**	6	9.5	9	2	3	4.5	4.5
P2	2.5	1.75	5	9	**1.75**	4	6.5	6.5
P3	2	3.5	9.5	9.5	**1**	3.5	9.5	5
P4	**1.5**	2.75	9	9.5	1.75	4	5.5	5.5
P5	3.43	5.5	7.15	6.75	**1.58**	3.53	4.65	3.43
P6	3.03	**2.15**	7.75	6	2.28	2.4	8	5
P7	**1.75**	3.6	9.5	9.35	5.25	2.45	9.5	4.2
P8	3.95	4.3	8.1	8.5	3.65	3.1	**3**	5.3
P9	**2.8**	3.4	5	5.9	3	3	7	6.5
P10	**3.2**	4.5	9.5	6.13	3.5	4	5.25	5.68
P11	6.25	8.5	4.25	**1.63**	6	9	4	5
P12	**1.38**	4.18	7.5	7.5	2.6	3.8	4.63	3.8
Average	**2.77**	3.87	7.91	7.7	2.81	3.63	6.39	5.03

Table 8: Possible weightings for Grand Rank calculation

wip	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
wfp	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9

Next, we plot each algorithm Grand Ranking with all possible weights (Table 9) to analyze how the weights affect Grand Ranking.

Here in Fig. 1 we have the results for each method with a convex combination of weights. So, we can see that some lines intersect, which means that the ranking depends on wip and wfp values, but the dependencies are quite stable. In some cases, the line has a positive slope which means WFR is higher than WIR, and if the slope is negative, then WIR is higher than WFR.

7. Hypothesis testing

In this section, we check whether the methods have a statistically significant difference one from another. Let us state two mutually exclusive, Null and Alternative, hypotheses:

H_0 : Two methods have the same performance.
H_1 : Two methods perform differently.

The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used for checking the validity of a hypothesis since, because of data size, we can

Table 9: Grand Ranking with all possible weights

CGN(A)	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
2.48	5.78	7.03	7.15	1.31	4.38	3.49	4.36
2.72	5.71	7.06	7.05	1.38	4.17	3.78	4.13
2.95	5.64	7.09	6.95	1.44	3.95	4.07	3.89
3.19	5.57	7.12	6.85	1.51	3.74	4.36	3.66
3.43	5.5	7.15	6.75	1.58	3.53	4.65	3.43
3.66	5.43	7.18	6.65	1.64	3.31	4.94	3.19
3.89	5.36	7.21	6.55	1.70	3.09	5.23	2.95
4.13	5.29	7.24	6.45	1.77	2.88	5.52	2.72
4.36	5.22	7.27	6.35	1.83	2.66	5.81	2.48

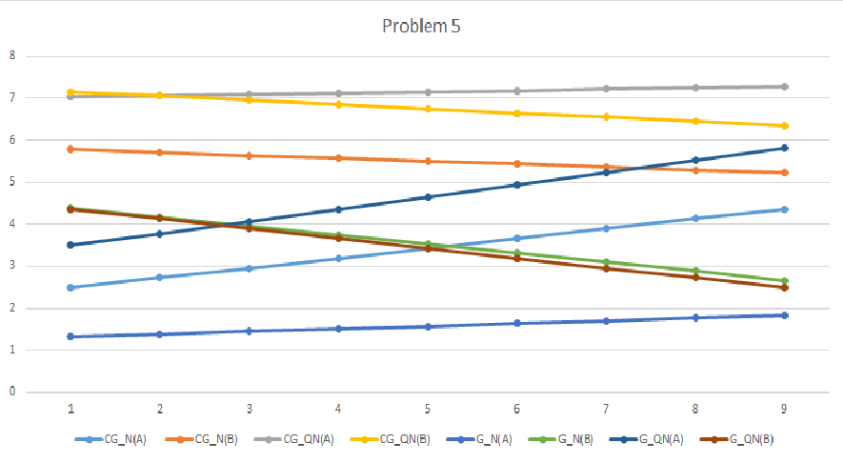

Figure 1: Convex weighted ranking

not prove the normality. Thus, the Wilcoxon test can be a good alternative to the conventional T-test.

From the Grand Rank Table 7, we can see that CGN(A) is the most reliable and efficient method so we test it against all the other methods.

The Wilcoxon Signed-Ranks Test critical value is 137 (since the number of test problems for rank calculation in Table 5 is 30), so at least one value, either positive sum or negative sum, has to be lower than the critical value.

In the case of GN(A) and GN(B), the null hypothesis cannot be rejected so there is no statistically significant reason to reject the fact that two algorithms may produce a similar outcome in terms of the number of iterations, number

Table 10: Hypothesis testing

Sum	CGN(B)	CGQN(A)	CGQN(B)	GN(A)	GN(B)	GQN(A)	GQN(B)
Pos. Sum	125	55	62	284	227	107	127.5
Neg. Sum	334	407	400	153	228	358	337.5

of function evaluations or ranks. But for all other cases, we can reject the null hypothesis so we can say that these algorithms are statistically different.

8. Convergence

Table 11 shows a summary of convergence for all algorithms from Tables 5 and 6. Table 6 shows the relaxed ranking which means the method converges to the solution with some relaxation. We performed two manipulations with respect to this ranking, first, we increased the number of iterations from 500 to 750 and if the method converges, then rank 9 is given, otherwise, we decrease the tolerance to 10^{-3}, and if the method converges, then rank 10 is given. If the method failed to converge in the case of both relaxations, then we give it rank 11. GQN(B) returns solutions for almost all problems except one, although the number of iterations and the total number of function evaluations are higher than in other algorithms. CGN(A) and CGN(B) are the most reliable and efficient methods as they are characterized by a lower number of iterations and a total number of function evaluations with more than 90% convergence and 100% relaxed convergence rates.

Table 11: Convergence percentage for test problems

Algorithm	Convergence	Relaxed Convergence	Non Convergence
CGN(A)	86.7	100	0
CGN(B)	86.7	100	0
CGQN(A)	46.7	56.7	43.3
CGQN(B)	53.3	73.3	26.7
GN(A)	76.7	93.3	6.7
GN(B)	80	100	0
GQN(A)	76.7	86.7	13.3
GQN(B)	96.7	100	0

9. Graphical output

In this section, we present a few examples of the graphical output to demonstrate the Algorithm 1 outcome. We found out that CGN method is most efficient and reliable on the test problems mentioned in Table 2. Let us analyze the graphical output for problem P2, which looks simple visually as we have four functions only.

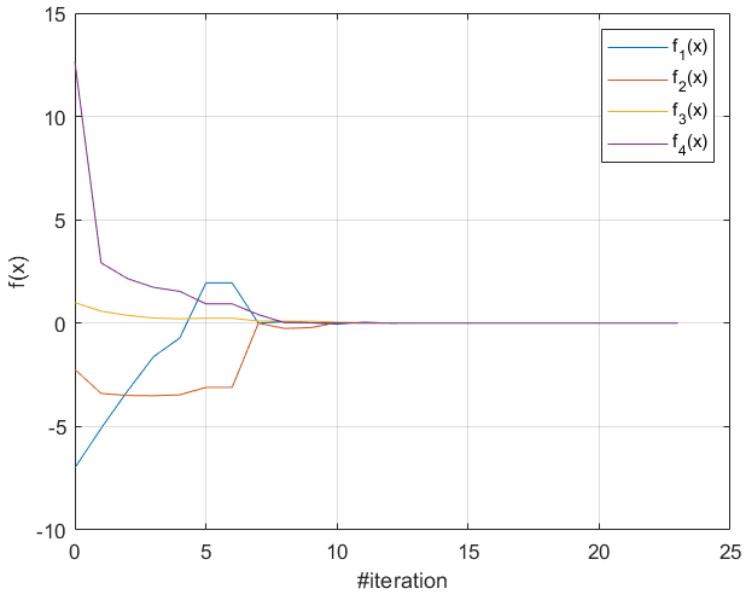

Figure 2: Function values vs. number of iterations for P2

In Fig. 2 each curved line shows how the function value changes at each iteration. In some cases, functions move towards the root from the first iteration on but for some, it takes a few iterations to converge. Figure 3 demonstrates combined error at each iteration converging towards the user-defined tolerance of 10^{-6}. Figure 4 shows the values of the variables at each iteration moving towards the root. In the depicted case, the P2 value of the root is zero. Now, we look at a problem with the specified problem dimension. The graphical output for P8 (with $n = 50$) is presented in Figs. 5 through 7. From there we see that the convergence for P8 is achieved much slower when compared to the case of P2.

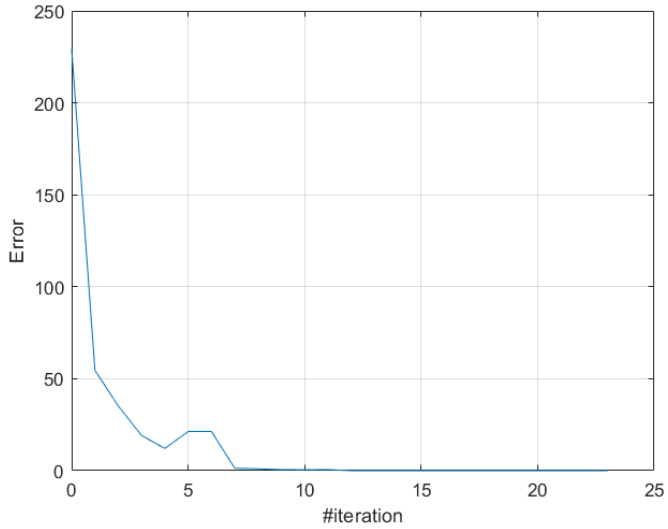

Figure 3: Error vs. number of iterations for P2

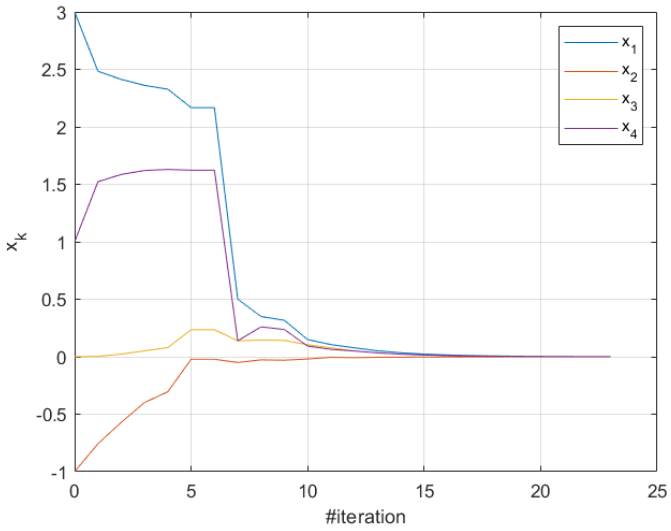

Figure 4: Root vs. number of iterations for P2

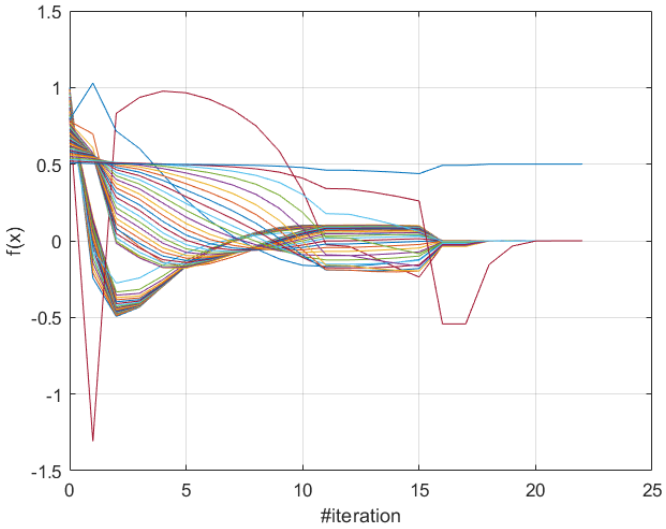

Figure 5: Values vs. number of iterations for P8

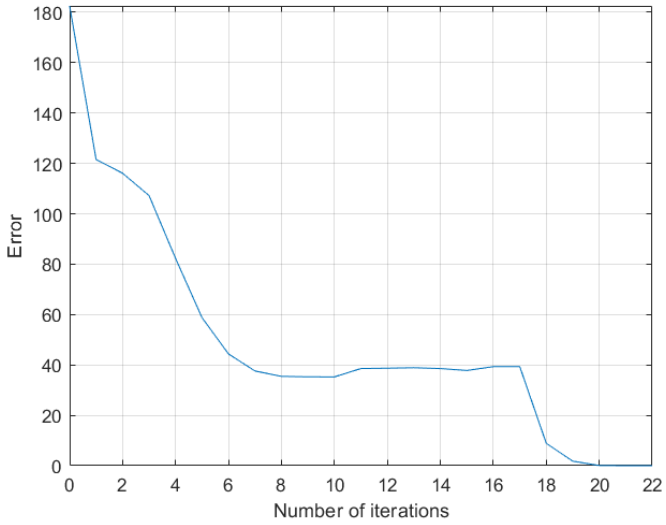

Figure 6: Error vs. number of iterations for P8

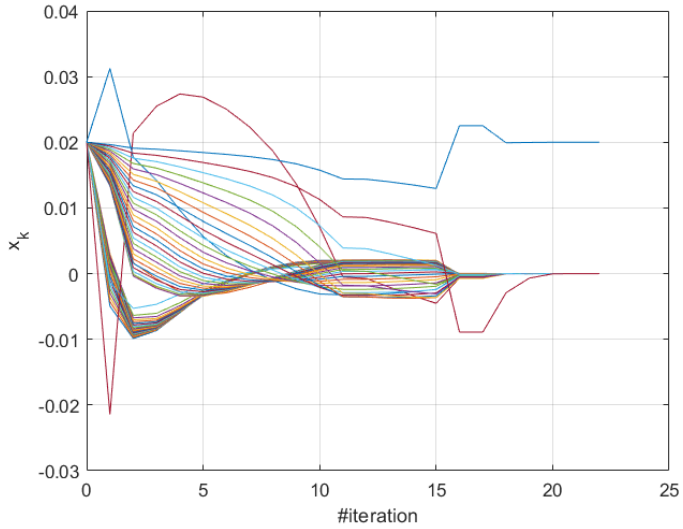

Figure 7: Root vs. number of iterations for P8

10. Conclusions

In this paper, generally following the hybridization approach of Taheri and Mammadov (2012), we considered a new hybridization variant (Conjugate Gradient Newton (CGN)). We compared it with several other possible combinations of methods (GN, GQN, CGQN). We empirically demonstrated that CGN(A) variant is, probably, the most reliable hybridization variant that in most cases outperforms (w.r.t. numbers of iterations and function evaluations) other hybridization variants. However, the original variant of hybridization from Taheri and Mammadov (2012) (variant GN(A)) stays second best in our analysis. Additional experiments are needed to ascertain the statistical significance of the difference between them.

References

Andrei, N. (2008) An Unconstrained Optimization Test Functions Collection, *Advanced Modeling and Optimization*, **10**, 147–161.

Atkinson, K. E. (1978) *An Introduction to Numerical Analysis. Nonlinear Systems of Equations.* John Wiley & Sons, Canada.

Broyden, C. G. (1970) "The convergence of a class of double-rank minimization algorithms", Journal of the Institute of Mathematics and Its Applications, **6**: 76–90, doi:10.1093/imamat/6.1.76

Buckley, A. (1978) A combined conjugate-gradient quasi-Newton minimization algorithm. *Mathematical Programming*, **15**, 200–210.

Burden, R. L. and Faires, J.D. (2010) *Numerical Analysis*, 9th Edition. Brooks Cole.

Hestenes, M. and Stiefel, E. (1952) Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, **49**, 409–436.

Fletcher, R. (1970) "A New Approach to Variable Metric Algorithms", Computer Journal, **13** (3): 317–322, doi:10.1093/comjnl/13.3.317

Fletcher, R. and Reeves, C. (1964) Function Minimization by Conjugate Gradients. *Computer Journal*, **7**, 149–154.

Goldfarb, D. (1970) "A Family of Variable Metric Updates Derived by Variational Means", Mathematics of Computation, **24** (109): 23–26, doi:10.1090/S0025-5718-1970-0258249-6

Kelley, C. (1987) *Solving Nonlinear Equations with Newton's Method*. Cambridge University Press.

Kelley, C. (1995) *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Philadelphia.

Nedzhibov, G. (2008) A family of multi-point iterative methods for solving systems of nonlinear equations. *Journal of Computational and Applied Mathematics*, **2**, 244-250.

Powell, M. J. D. (1970) A hybrid method for nonlinear equations. *Numerical Methods for Nonlinear Algebraic Equations*, ch. 6, 87–114.

Shanno, David F. (July 1970) "Conditioning of quasi-Newton methods for function minimization", Mathematics of Computation, **24** (111): 647–656, doi:10.1090/S0025-5718-1970-0274029-X, MR 0274029

Shi, Y. (2000) Globally Convergent Algorithms for Unconstrained Optimization. *Computational Optimization and Applications*, New York, **16**, 295–308.

Taheri, S. and Mammadov, M. (2012) Solving systems of nonlinear equations using a globally Convergent Optimization algorithm. *Global Journal of Technology & Optimization*, **3**, 132–138.