PROBABILITY-DIRECTED PROBLEM OPTIMIZATION TECHNIQUE FOR SOLVING SYSTEMS OF LINEAR AND NON-LINEAR EQUATIONS

Muhammed J. Al-Muhammed

Faculty of Information Technology American University of Madaba, Madaba, Jordan

ABSTRACT

Although many methods have been proposed for solving linear or nonlinear systems of equations, there is always a pressing need for more effective and efficient methods. Good methods should produce solutions with high precision and speed. This paper proposed an innovative method for solving systems of linear and nonlinear equations. This method transforms the problem into an optimization problem and uses a probability guided search technique for solving this optimization problem, which is the solution for the system of equations. The transformation results in an aggregate violation function and a criterion function. The aggregation violation function is composed of the constraints that represent the equations and whose satisfaction is a solution for the system of equations. The criterion function intelligently guides the search for the solution to the aggregate violation function by determining when the constraints must be checked; thereby avoiding unnecessary, time-intensive checks for the constraints. Experiments conducted with our prototype implementation showed that our method is effective in finding solutions with high precision and efficient in terms of CPU time.

KEY WORDS

Solutions for systems of linear and non-linear equations, random-guided search, optimization problem, global minimum

1. INTRODUCTION

Systems of linear or non-linear equations are ubiquitous. They frequently result from modeling problems in many important domains such as engineering, robotics, business, and many more. Therefore, finding effective and efficient methods for finding simultaneous solutions for these equations is extremely and practically important to study the properties of the problems modeled by these equations.

The desirable properties of the solution methods are the effectiveness and efficiency. The method must be effective in producing a solution with high precision. It must also produce the solution for the problem in a minimum time and resources requirements.

Researchers have proposed many methods for solving systems of equations [1][7][8] (see [5] for a good discussion of these methods). These methods either solve the equations directly by
applying numerical methods [4][9][10][11][17][23] or solve them indirectly by transforming the system of equations into problem optimization [2][3][4][5][12][16][19][22].

This paper offers a technique for solving systems of linear or nonlinear equations. The technique indirectly solves a system of equations by solving an optimization problem. Each equation in the system is considered a constraint. These constraints are combined, in a way to be made precise later, into a non-negative function, called aggregate violation function. This function measures the collective amount of violation caused by some substitution to the parameters of the constraints. Thus, a substitution for which the aggregate violation function evaluates to zero is the solution for the system of equations because this substitution satisfies all the constraints (i.e. the equations).

The proposed technique goes even further by creating a function, called a criterion function, whose main objective is to identify whether some substitution is promising (results in a better value for the aggregate violation function) or not. The criterion function is created using the systems of equations in such a way that, as will show later, its computational demand is much less than the computational demand of the aggregate violation function. This function controls the process of evaluating the aggregate violation by permitting this evaluation at some substitution only when this substitution is promising. Our technique therefore saves the time required for blindly evaluating the aggregate violation function on every possible substitution and consequently increases its efficiency.

To solve the optimization problem, we propose a probability-guided algorithm that uses random numbers and biased mapping to quickly direct the search to the parts of the variables’ domains where the solution resides. The algorithm also makes use of the criterion function to intelligently direct the search in such a way that the method avoids unnecessary checks of the constraints.

The paper makes the following contributions. First, it offers an effective algorithm for problem optimization. Second, it offers effective transformation from systems of equations to optimization problem. This transformation produces a very rigorous aggregate violation function whose satisfaction results in high precision solutions of the equation systems. Third, the transformation produces a light-weighted criterion function that quickly determines if the aggregate violation function must be checked or not, thereby avoiding unnecessary time consuming checks of the aggregate violation.

We present our contribution as follows. Section 2 describes the problem formalization, which transforms the solution to systems of equations to the solution of problem optimization. In section 3, we discuss the proposed algorithm and its technical details. Section 4 presents our experimental evaluation of the proposed technique. We conclude and give directions for future work in sections 5.

2. THE PROBLEM FORMALIZATION

In this section we formalize the problem of solving a system of equations. We present some fundamental concepts in subsection 2.1 and show our formalization in subsection 2.2.

2.1 PRELIMINARIES

Consider the following system of equations:
We rewrite the system (1) as the following set of constraints

\[ C = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = b_1 
\]
\[ a_1^2 x_1 + a_2^2 x_2 + \ldots + a_n^2 x_n = b_2 
\]
\[ \ldots \]
\[ a_1^n x_1 + a_2^n x_2 + \ldots + a_n^n x_n = b_n \]  

(1)

The terms \( a_i^j \) and \( b_i (i, j=1, \ldots, n) \) are constants and \( x_i \)'s are variables. The variables \( x_i \)'s can be of any power (not necessarily linear) or even functions (sine, log, etc.). We represent them as in (2) just to simplify the presentation. A solution to the set of equations is a substitution \( X^* = (x_1^*, x_2^*, \ldots, x_n^*) \) for the variables \( x_i \)'s such that all of the constraints \( C \) hold.

We associate with the constraints (2) a function \( f(x_1, x_2, \ldots, x_n) \), which is defined by summing all the constraints \( C \) and taking the absolute value for the resulting sum. The variables \( x_i \)'s with the same index and power are combined together. Consider for instance the following system of equations.

\[ x_1 + 3x_2^2 + x_3 = 6 
\]
\[ 3x_1 + 2x_2 + x_3 = 7 
\]
\[ x_1^2 + x_2^2 + 2x_2 = 4 \]

We sum these three constraints to yield the following function \( f(x_1, x_2, x_3) \).

\[ f(x_1, x_2, x_3) = |4x_1 + x_1^2 + 2x_2 + 4x_2^2 + 4x_3 - 17| \]

Observe that the variables \( x_1 \) of powers 1 are summed to yield \( 4x_1 \) and the variable \( x_1 \) of power 2 are summed together and so on.

According to this, we create the function \( f(x_1, x_2, \ldots, x_n) \) from the equations (1) by adding together the variables with the same index and with the same power. The addition yields the function (3).

\[ f(x_1, x_2, \ldots, x_n) = |\sum_{i=1}^{n} a_i^1 x_1 + \sum_{i=1}^{n} a_i^2 x_2 + \ldots + \sum_{i=1}^{n} a_i^n x_n - \sum_{i=1}^{n} b_i| \]  

(3)

Let \( X^* = (x_1^*, x_2^*, \ldots, x_n^*) \) be a possible substitution. We define the degree of the violation caused by the substitution \( X^* \) to some constraint \( C^k \) by:

\[ \rho^k (X^*) = |a_1^k x_1^* + a_2^k x_2^* + \ldots + a_n^k x_n^* - b_k| \]
It should be clear that the degree of the violation is zero (i.e. no violation) when the substitution \( X \) satisfies the constraint \( C_k \). The degree of the violation is positive if \( X \) does not satisfy the constraint \( C_k \). Therefore, greater values of \( \rho_k(X_i) \) indicate a large violation of the substitution \( X_i \) to the constraint \( C_k \).

We in addition define the aggregate violation of the substitution \( X_i \) to all of the constraints \( C_i \) as follows

\[
E^c(X_i) = \sum_{k=1}^{n} \rho^k(X_i)
\]

That is, the aggregate violation of some substitution \( X_i \) is the sum of the violations of \( X_i \) over all the constraints. The aggregate violation is zero only if \( X_i \) satisfies all the constraints. Greater values of the aggregate violation \( E^c(X) \) indicate a greater violation for a constraint or more.

We emphasize that function \( f(x_1, x_2, \ldots, x_n) \) is completely different from the aggregate violation \( E^c(X) \). The following example shows the difference between them.

Example (1): consider the following two linear equations whose solution is \( X (1, 1) \).

\[
\begin{align*}
2x_1 + x_2 &= 3 \\
-x_1 + x_2 &= 0
\end{align*}
\]

By summing these two equations and take the absolute value we obtain the function \( f(x_1, x_2) = |x_1 + 2x_2 - 3| \). Let \( X(0, 1.5) \). It is clear that \( f(0, 1.5) \) equals to zero while \( E^c(X) \) is not equal to zero. That is, \( E^c(0, 1.5) = |(2(0) + 1.5 - 3)| + |-(0) + 1.5 - 0| = 3 \).

For most of the systems of equations (especially the systems of linear equations), there is a large number of common variables between the equations. For such systems the function \( f(x_1, x_2, \ldots, x_n) \) is likely to have fewer number of variable substitutions than the original constraints because the common variables are combined by the summation. The function \( f(x_1, x_2, \ldots, x_n) \) demands therefore less computation than the aggregate violation function does for such systems. On the other hand, even if there are few (or even none) common variables, the function \( f(.) \) still demands less computation than the aggregate violation. That is because it computes the absolute value only one time while the aggregate violation function computes the absolute value for each constraint.

Proposition (1): Let \( X_i, X_j \) betwo possible substitutions for the aggregate violation function. If \( f(X_i) > E^c(X_j) \), it cannot be the case that \( E^c(X_i) < E^c(X_j) \).This is represented mathematically as \( f(X_i) > E^c(X_j) \iff E^c(X_i) < E^c(X_j) \).

Proof

The proof of this proposition is straightforward and follows directly from the definition of the function \( f(.) \) and the aggregate violation function.

Suppose that \( f(X_i) > E^c(X_j) \). Since \( f(X_i) < E^c(X_i) \) based on the definition of both \( f(.) \) and \( E(.) \) and the properties of the absolute value, we conclude that \( E^c(X_i) > E^c(X_j) \).
Proposition (1) precisely defines the relationship between the aggregate violation \( E^c(.) \) and the function \( f(.) \). For any substitution \( X' \), if the value of the function \( f(X') \) is larger than the value of the aggregate violation at some previous substitution \( X \), the substitution \( X' \) will not satisfy the constraints better than \( X \); that is \( X' \) will not reduce the value of the aggregate violation. Given this, the function \( f(.) \) can be used as a precondition for checking the constraints at some \( X' \). Namely, the constraints should not be checked at any substitution \( X' \) for which proposition (1) holds. Observe if the substitution \( X' \) does not satisfy proposition (1) (i.e. \( f(X') \leq E^c(X') \)), it is not necessary that \( X' \) reduces the constraints violations (reduce the aggregate violation).

We therefore call the function \( f(.) \) a *criterion function*. It is called so because it guides the process of whether we should evaluate the aggregate violation or not at some substitution \( X' \).

**2.2 Problem Transformation**

Given the aggregate violation of the constraints \( E^c(X) \) and the criterion function \( f(X) \), we formalize the problem of solving the system (1) as follows. Rather than solving the system of equations per se, we indirectly solve it by solving the following optimization problem.

\[
\text{Minimize} \quad E^c(X) = \sum_{k=1}^{n} \rho^k (X')
\]

Guided by the criterion function:

\[
f(x_1, x_2, ..., x_n) = |\sum_{i=1}^{n} a_{i1} x_1 + \sum_{i=2}^{n} a_{i2} x_2 + ... + \sum_{i=n}^{n} a_{in} x_n - \sum_{i=1}^{n} b_i |
\]

(4)

According to the definition of the aggregate violation \( E^c(X') \), it is clear that the global minimum of \( E^c(X') \) is zero. Therefore, if \( E^c(X') = 0 \) for some substitution \( X' \), this substitution is the global minimum for the optimization problem (4). This substitution \( X' \) is also an exact solution for the system (1) because it violates no constraint (i.e. satisfies all the equations in system (1)). If, however, \( E^c(X') > 0 \), the substitution \( X' \) is not the global minimum for the optimization problem (4). This substitution \( X' \) is not an exact solution for the system (1) either due to the constraint violation (i.e. one or more of the equations in system (1) not satisfied). Furthermore, it can be easily shown that if the substitution \( X' \) is an exact solution for the system (1), it is not the global minimum for the optimization problem (4) either. According to this discussion, a substitution \( X' \) is a solution for the system (1) if and only if it is the global minimum for the optimization problem (4).

The criterion function \( f(x_1, x_2, ..., x_n) \) plays an extremely important role in the process of optimizing the problem (4); and equivalently finding a solution for the equations (1). It intelligently instructs the solver when the constraints must be checked for some substitution \( X' \). Blindly checking the constraints for every possible substitution \( X' \) has a negative impact on the performance of the solver since this check is computationally intensive especially if we have a large number of these constraints. Generally speaking, the criterion function \( f(.) \) greatly improves the solver’s performance by allowing this solver to check the constraints only when it should.
3. The Hit-Hit Moving Directives Algorithm

We propose an algorithm, called Hit-Hit Moving Directives (HHMD-3), for solving unconstrained optimization problem (4). HHMD-3 is a probability-directed search technique that finds the global optimum (minimum or maximum) for unconstrained optimization problems such as the one in (4). This algorithm utilizes sequences of random numbers to search for the global optimum within the domains of the variables. It defines three directives over the unity interval [0, 1] as shown in Figure 1. These directives are a principal directive and two marginal directives. The principal directive covers part of the unity interval, which we call the effective search interval (ESI). The marginal directives cover the remaining part of the unity interval, which we call the marginal search intervals (MSI). The principal directive is centered at the point Ci of the unity interval and has a radius qi. The two marginal directives (left and right) are defined in the rest of the interval with the radius (1 - 2qi). The role of these directives is to guide the search in the domains of the variables and quickly locate the global minimum for the problem. In particular, the principal directive biases the search to the parts of the domains where the global minimum is more likely to reside. The two marginal directives guarantee that if the global minimum is not within the effective search interval, the technique can escape being caught within this interval and therefore for missing the global minimum.

Figure 1: The principal directive (shaded) and two marginal directives on the sides.

The principal and marginal directives are fully defined by their parameters Hi and hi (heights) and the radius (qi). The heights are given by formulas (5).

\[
H_i = \frac{\alpha + 1}{2} - |\beta - q_i| \\
h_i = \frac{\beta - 1}{1 - 2(q_i - \varepsilon)} - |\beta - q_i|
\]  

(5)

Where \(\alpha\) and \(\beta\) belong to the interval [0, 1] and \(\alpha + \beta = 1\). The number \(\varepsilon\) is a sufficiently small real number (e.g. 1E-300), called the adjustment factor. Given that the value of \(q_i \in (0, 0.5]\), the role of the adjustment factor is to guarantee that the denominator of \(h_i\) will never be zero and therefore \(h_i\) becomes undefined.

Referring to formulas (5), we can easily observe that \(H_i^*(2q_i) = \alpha + |\beta - q_i|\) is the area of the principal directive. Likewise, \(h_i^*(1 - 2(q_i - \varepsilon)) = |\beta - q_i|\) is the area of the two marginal directives. This means that the area of the principal directive is at least \(\alpha\) while the area of the two marginal directives at most \(\beta\). It is clear that the sum of the areas of the three directives equals 1. Note that
we can increase the area of the principal directive by increasing \( \alpha \) and increase of the marginal directives by increasing \( \beta \). Additionally, decreasing the radius \( q \) enlarges the area of the principal directive and reduces the area of the two marginal directives.

**Definition (1)**

1) We call the part of the interval that is covered by the principal directive the effective search interval because it has the global optimum with a high probability.

2) For the purpose of this paper, we mean by "global optimum" the minimum value (zero) of the aggregate violation.

Let \( P^M \) be the probability that some random number \( \gamma \in [0, 1] \) belongs to the base of the principal directive and \( P^L \) and \( P^R \) be the probability that \( \gamma \) belongs to the bases of the left and right directive respectively. We define these probabilities by the following formulas. \( P^M = h_i^*(2q_i) \) and \( P^L = h_i^*(c_i-q_i) \) and \( P^R = h_i^*(1-c_i-q_i) \). Where \( c_i \) is the center of the principal directive and \( q_i \) is its radius.

Observe that definition (1) bases the probability on the area of the directive. That is, the larger the area of a directive is the larger the likelihood that \( \gamma \) will belong to the base of this directive. Since \( \alpha \) and \( \beta \) represent respectively the lower and upper bounds of respectively the principal and marginal directives areas, we can increase the probability of having some random number belong to the base of principal directive by increasing \( \alpha \). Since \( \alpha + \beta = 1 \), this means as the probability of having a random number belong the base of the principal directive increases the probability that this random number belongs to the bases of the marginal directives decreases. In other words, \( \alpha \) and \( \beta \) allow us to fully bias random numbers toward the bases principal or marginal directives.

The values for \( \alpha \) and \( \beta \) are determined based on the criteria of biasing the probability toward the principal directive without totally ignoring the marginal directives. Since the minimum bound of the principal directive is \( \alpha \), any value for \( \alpha > 0.5 \) would cause more bias toward the principal directive base (or the effective search interval). During our preliminary experiments to tune the values of \( \alpha \) and \( \beta \), we found that \( \alpha = 0.75 \) and \( \beta = 0.25 \) are very rational choices because they quickly derive the search to optimal solution.

Given the principal and marginal directives, we define the biased mapping between the interval \([0, 1]\) and the bases of the directives by the mapping rules in Figure 2. As the figure shows, mapping a random number \( \psi \in [0, 1] \) to the bases of the principal or marginal directives is actually proportional to the area of the directives. Specifically, the algorithm compares \( \psi \) to the areas of the left and right directives and maps \( \psi \) to base of either the left directive if the comparison in lines 2, 6 succeeds or to the base of the right directive if the comparison in lines 3, 5 succeeds. It maps \( \psi \) to the base of the principal directive in lines 4 or 7 otherwise.

Based on the discussion, it is clear that any random number \( \psi \) either hits the base of the principal or the bases of the marginal directives. That is why we called the algorithm hit-hit algorithm.
42 Computer Science & Information Technology (CS & IT)

(1) IF C_i < 0.5 THEN

(2) IF ψ_i < P_L THEN γ_i = ψ_i / h_i

(3) ELSE IF ψ_i < (P_L + P_R) THEN γ_i = ψ_i / h_i + 2q_i

(4) ELSE γ_i = \frac{2(ψ_i - (Li + R_i))}{H_i} + C_i - q_i

(5) ELSE IF ψ_i < P_R THEN γ_i = (ψ_i / h_i) + C_i + q_i

(6) ELSE IF ψ_i < (P_L + P_R) THEN γ_i = (ψ_i / h_i) + C_i + q_i - 1

(7) ELSE γ_i = \frac{2(ψ_i - (Li + R_i))}{H_i} + C_i - q_i

Figure 2: the rules of the biased mapping from [0, 1] to principal/marginal directives

3.1 THE TERMINATION CONDITIONS

Let \( x \) and \( y \) be the two substitutions of the variables \( x_1, x_2, \ldots, x_n \) in two consecutive rounds \( i \) and \( i+1 \). Suppose also that \( E_c(X_i) \) and \( E_c(X_{i+1}) \) are the best values for \( E_c \) in the rounds \( i \) and \( i+1 \) respectively. We define the termination conditions as follows.

\[
| E^c(X^{i+1}) - E^c(X^i) | < \delta
\]  

(6)

Where \( \delta \) is sufficiently small number (e.g. 1E−100). The termination condition means that the value of the aggregate violation does not change for two consecutive rounds \( i \) and \( i+1 \). If this condition holds, the search for the global minimum (zero) has reached an equilibrium point and no improvement can be achieved if the algorithm continues the search.

3.2 THE ALGORITHM TECHNICAL DETAILS

Figure 3 shows the technical details of the algorithm. The algorithm searches the domains of the parameters \( X \) for values that bring the aggregate violation function \( E^c(X) \) to its minimum (zero). It performs a number of rounds until the termination condition (6) holds. In any round \( j \), the algorithm conducts many experiments each of which consists of \( m \) steps. In each step, it generates \( n \) random numbers \( ψ \) in the interval [0, 1] using the computer built-in random generator and uses the biased mapping rules (Figure 2) to map each random number to the bases of one of the directives (principal or marginal). The mapping yields new biased random numbers \( γ \) (lines 9–11). The algorithm uses the biased mapping to focus most of the search in the principal directives since these directives cover parts of the domain in which the global minimum most likely resides. The biased mapping, however, does not ignore the marginal directives that cover the remaining parts of the domains. Thus, the biased mapping never causes the algorithm to miss the global minimum or get trapped in a local minimum.

The values \( γ \)'s are mapped to the actual domains ([a_i, b_i]) of the variables \( X \) using the formula in line 12. For each randomly created substitution for the variables \( X \), the algorithm computes the value of the criterion function (line 13). If the new substitution satisfies the condition in line 14, this substitution is promising and therefore, the algorithm computes the aggregate violation at this substitution (line 15). If the calculated value better than the previous minimum stored in \( VE^c \), the algorithm stores the new minimum along with the following fundamental information: the
substitution itself and the random values $\gamma_i$'s from which the substitution was generated (lines 17–19).

After performing $m$ steps, the algorithm reduces the radiuses of the principal directives using the reduction formula $\forall i q_i = q_i / d$, where $d > 1$ (lines 21-23). The algorithm performs another experiment if at least one of $q_i$'s is still greater than some pre-specified threshold $\varepsilon$ (typically $\varepsilon$ is less than $1E^{-60}$).

We make two important points regarding $q_i$ reduction formula and its effect on the algorithm convergence. First, the reduction factor $d$ can be theoretically any real number greater than 1. Greater values of $d$ cause the algorithm to converge faster because it quickly reduces the radiuses of the principal directives. We tested our algorithm for only four different values of $d$, namely 1.2, 1.5, 1.8, and 2. All these values cause the algorithm to converge quickly irrespective of the formula that defines the aggregate violation, although the time of convergence becomes relatively shorter as $d$ increased from 1.2 to 1.5 to 2. No tests were made for $d$ greater than 2. Secondly, referring to formula 5, it is clear that as the algorithm further reduces $q_i$, the areas of the principal directives increase and the areas of the marginal directives decrease. This means that more points are mapped to the bases of the principal directives and fewer points to the bases of the marginal directives. In other words, reducing $q_i$ plays the major role in greatly focusing the search to the principal directives (where the global minimum most likely is located). Therefore, the effective intervals are thoroughly searched because (1) their radiuses continuously reduce and (2) more points mapped to them.

\textsuperscript{1}The best value for $E^c$ in round $j$ is the one that yields the closest value to zero in this round.
FOR $i \leftarrow 1$ to $n$ DO 
  $C_i = 0.5$ /* initialize centers of principal directives */ 
  $q_i = C_i$ /* initialize radiuses of principal directives */ 
REPEAT
  WHILE ($q_i \geq \varepsilon$ for any $i$) DO
    FOR $i \leftarrow 1$ to $n$ DO /* $m$ steps in each experiment */
      Generate a random number $\psi_i \in [0, 1]$ /* Random numbers generated */
      $\gamma_i = \text{MAP}(\psi_i)$ using the logic in Figure 2
      $F = f(x_1^j, x_2^j, \ldots, x_n^j)$ /* compute the criterion function $f$ at $x_i^j$ */
      IF $F < C^c$ THEN
        $C^c = F$
      ENDIF
      $X^j_{\min} = (x_1^j, x_2^j, \ldots, x_n^j)$
      $Q^j_{\min} = (\gamma_1^j, \gamma_2^j, \ldots, \gamma_n^j)$
    ENDWHILE ($q_i > \varepsilon$, for any $i$)
    Compute $H_i$ and $h_i$ using formulas (5)
  ENDWHILE
FOR $i \leftarrow 1$ to $n$ DO
  $C_i = \gamma_i$
  IF $C_i < 0.5$ then $q_i = C_i$
  ELSE $q_i = 1 - C_i$
  $j = j + 1$
UNTIL Termination Condition (6) holds.
END ALGORITHM

Figure 3: the technical steps of the Hit-Hit moving directives algorithm (HHMD-3).

In any subsequent round $j$, the algorithm uses the information of the previous round $j-1$ to dynamically adjust the parameters of the directives. In particular, the algorithm moves the centers of the principal directives $C_i$'s to the values $\gamma_i$'s, which produced the best minimum in the previous round, and calculates the new radiuses $q_i$'s (lines 25–28). The algorithm moves the centers to these values because there is a high probability that the search finds values for the variables that improve the minimum of the aggregate violation in the vicinity of the random numbers $\gamma_i$'s.
4. PERFORMANCE ANALYSIS

We implemented our algorithm using JAVA programming language. We conducted many experiments using our prototype implementation to evaluate the effectiveness and efficiency of our algorithm. The experiments were conducted on a large number of systems of linear and non-linear equations obtained from benchmarks[5][14][15] and others. The hardware platform is Duo core processor laptop running at 1.7 GH with main memory of 2GB. The operating system is windows 7 (32 bits).

We start our analysis by studying the performance of our algorithm on samples of systems with few but challenging equations obtained from [16][17]. Table 1 shows these systems and the domains of their variables. It shows also the performance of our algorithm measured in terms of both the aggregate violation EC (i.e. the precision of the solutions) and the CPU time in milliseconds (ms).

<table>
<thead>
<tr>
<th>Equations</th>
<th>Domain</th>
<th>Aggregate violation EC</th>
<th>CPU Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e^{-i\theta} - 8x_1 \sin(x_2) = 0$</td>
<td>[-10, 10]</td>
<td>7.54E-7</td>
<td>377</td>
</tr>
<tr>
<td>$x_1 + x_2 -1 = 0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(x_1 - 1)^2 = 0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$3x_1 - \cos(x_2, x_3) - 0.5 = 0$</td>
<td>[-10, 10]</td>
<td>8.34E-8</td>
<td>689</td>
</tr>
<tr>
<td>$x_1^2 - 625x_1^2 - 0.25 = 0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$e^{-x_1x_2} + 20x_1 + \left(\frac{10\pi - 3}{3}\right) = 0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1^2 + x_2^2 - 5x_1x_2x_3 = 85$</td>
<td>[-10, 10]</td>
<td>2.19E-8</td>
<td>3036</td>
</tr>
<tr>
<td>$x_1^3 - x_2^3 + x_3 = 60$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1^4 + x_2^4 - x_2 = 2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$3x_1^3 + 2\sin(x_1) \cos(x_2) - 10.2 = 0$</td>
<td>[-10, 10]</td>
<td>6.35E-13</td>
<td>23</td>
</tr>
<tr>
<td>$2x_1^2 + \cos(x_1) \sin(x_2) - 17.3 = 0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1^3 + 3x_2 - x_3 + 10 = 0$</td>
<td>[-10, 10]</td>
<td>1.5E-12</td>
<td>273</td>
</tr>
<tr>
<td>$x_1^2 + x_2^2 + x_3^2 = 6$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1^4 - x_2^4 + x_3^2 = 2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\frac{0.25}{\pi} x_1 + 0.5x_1 - 0.5 \sin(x_2, x_3) = 0$</td>
<td>[-10, 10]</td>
<td>4.45E-13</td>
<td>51</td>
</tr>
<tr>
<td>$\frac{e^{-x_1x_2}}{\pi} - 2x_1 + (1 - \frac{0.25}{\pi})e^{x_1} = 0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4x_1^2 + 4x_1x_2 + 2x_2^2 = 14 = 0$</td>
<td>[-10, 10]</td>
<td>2.55E-11</td>
<td>12</td>
</tr>
<tr>
<td>$4x_1^2 + 4x_1x_3 + 2x_3^2 = 26x_3 - 22 = 0$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The performance of the algorithm presented in terms of aggregate violation (EC) and CPU time in milliseconds (ms).

According to the performance figures in the table, the algorithm performed really well. This is evident in both the aggregate violation (EC) and the CPU time. The highest aggregate violation is "7.5E-07". The rest of aggregate violations are much less. This means that we have solutions with really high precision. As the CPU time shows, the algorithm required roughly "0.6" second or less for almost all the cases except one case for which the algorithm required about 3 seconds. This problem took 3 seconds because it is known to be so hard to solve.
Table 2 presents benchmarks obtained from [5][14][15]. The table shows the name of the systems of equations (label), the equations themselves, the number of variables, and the domains of the variables. Note, for the economics modeling applications, we considered systems that consist of up to 1000 variables while other algorithms consider only up to 20.

<table>
<thead>
<tr>
<th>Label</th>
<th>System of Equations</th>
<th>Variables</th>
<th>domain</th>
</tr>
</thead>
</table>
| Benchmark i1 | \( x_1 - 0.25428722 - 0.18324757x_2x_3x_4 = 0 \) 
\( x_2 - 0.37842197 - 0.16275449x_3x_4x_5 = 0 \) 
\( x_3 - 0.27162577 - 0.16955071x_4x_5x_6 = 0 \) 
\( x_4 - 0.19807914 - 0.15585316x_5x_6x_7 = 0 \) 
\( x_5 - 0.44166728 - 0.19950920x_6x_7x_8 = 0 \) 
\( x_6 - 0.14654113 - 0.18922793x_7x_8x_9 = 0 \) 
\( x_7 - 0.42937161 - 0.21180486x_8x_9x_{10} = 0 \) 
\( x_8 - 0.07056438 - 0.17081208x_9x_{10}x_{11} = 0 \) 
\( x_9 - 0.34504906 - 0.19612740x_{10}x_{11}x_{12} = 0 \) 
\( x_{10} - 0.42651102 - 0.21466544x_{11}x_{12}x_{13} = 0 \) | 10 | [-2, 2] |
| Neurophysiology application | \( x_1^2 + x_2^2 = 1 \) 
\( x_2^2 + x_3^2 = 1 \) 
\( x_3x_4 + x_6x_7 = c1 \) 
\( x_4x_5 + x_8x_9 = c2 \) 
\( x_5x_6 + x_9x_{10}x_{11} = c3 \) 
\( x_6x_7 + x_8x_{12}x_{13} = c4 \) | 6 | [-10, 10] |
| Chemical equilibrium application | \( x_1 - 3x_3 = 0 \) 
\( 2x_1x_2 + x_1 + 3x_3^2 + 2R_1x_2^2 + R_3x_3x_4 + R_3x_4x_5 = 0 \) 
\( 2x_2x_3 + 2Rx_2^2 - 8x_5 + R_5x_3 + R_5x_4 = 0 \) 
\( 2x_3x_4 + 2Rx_3^2 - 4Rx_5 = 0 \) 
\( x_5 + 1 + R_5x_2^2 + 3x_3x_4^2 + R_5x_5 + R_5x_6 + x_4^2 - I + R_6x_5 + R_6x_6x_7 + R_6x_7x_8 = 0 \) 
\( R = 10, R_5 = 0.193, R_6 = 0.342957/40 \) 
\( R_7 = 0.030448/\sqrt{40}, R_8 = 0.0001799/40 \) 
\( R_9 = 0.0001255/\sqrt{40}, R_{10} = 0.00003846/40 \) | 5 | [-10, 10] |
| Combustion Application | \( x_2 + 2x_6 + x_8 + 2x_{10} = 10^{-5} \) 
\( x_3 + x_8 = 3 \cdot 10^{-6} \) 
\( x_1 + x_6 + 2x_8 + x_9 + x_{10} = 5 \cdot 10^{-5} \) 
\( x_4 + 2x_7 = 10^{-5} \) 
\( 0.0140487 \cdot 10^{-7}x_6 = x_1^2 \) 
\( 0.1010931 \cdot 10^{-5}x_6 = 2x_3^2 \) 
\( 0.0781078 \cdot 10^{-5}x_7 = 2x_4^2 \) 
\( 0.1362836 \cdot 10^{-6}x_8 = x_1x_2 \) 
\( 0.0194111 \cdot 10^{-7}x_9 = 2x_2^2 \) 
\( 0.0289326 \cdot 10^{-14}x_{10} = x_1x_2^2 \) | 10 | [-10, 10] |
### Computer Science & Information Technology (CS & IT) 47

**Table 2: Systems of equations benchmarks.**

<table>
<thead>
<tr>
<th>Benchmark i4</th>
<th>10</th>
<th>[-1, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1^2 - 0.25428722 - 0.18324757x_2x_3x_4 + x_3x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2^2 - 0.37842197 - 0.16275449x_4x_5x_6 + x_5x_6 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3^2 - 0.227162577 - 0.16955071x_1x_5x_6 + x_1x_5 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_4^2 - 0.19807914 - 0.15585316x_1x_4x_6 + x_1x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_5^2 - 0.44166728 - 0.19950720x_1x_3x_4 + x_1x_3 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_6^2 - 0.14654113 - 0.18022793x_1x_3x_5 + x_1x_3 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_i^2 - 0.42937161 - 0.21180486x_1x_i + x_1 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_i^2 - 0.42651102 - 0.21466544x_1x_i + x_1 = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark i5</th>
<th>10</th>
<th>[-1, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 - 0.24863995 - 0.19594124 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2 - 0.87528857 - 0.05612619 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3 - 0.23939835 - 0.20177810 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_4 - 0.47620128 - 0.16497518 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_5 - 0.24711044 - 0.20198178 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_6 - 0.33565227 - 0.15724045 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_7 - 0.13128974 - 0.12384342 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_8 - 0.45937304 - 0.18180253 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_9 - 0.46896600 - 0.21241045 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark i2</th>
<th>20</th>
<th>[-1, 2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 - 0.24863995 - 0.19594124 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_2 - 0.87528857 - 0.05612619 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3 - 0.23939835 - 0.20177810 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_4 - 0.47620128 - 0.16497518 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_5 - 0.24711044 - 0.20198178 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_6 - 0.33565227 - 0.15724045 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_7 - 0.13128974 - 0.12384342 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_8 - 0.45937304 - 0.18180253 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_9 - 0.46896600 - 0.21241045 x_1, x_2, x_3, x_4 = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benchmark i3</th>
<th>20</th>
<th>[-2, 2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>The same as Benchmark i2, but different interval for the variables.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Economics modeling application</th>
<th>20</th>
<th>[-10, 10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x_1 + \sum_{i=1}^{n+1} x_i x_i) x_i - c_i = 0, (1 \leq k \leq n - 1)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sum_{i=1}^{n+1} x_i + 1 = 0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Brown</th>
<th>5</th>
<th>[0, 3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2x_1 + x_2 + x_3 + x_4 + x_5 - 6 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1 + 2x_2 + x_3 + x_4 + x_5 + 6 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1 + x_2 + 2x_3 + x_4 + x_5 - 6 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1 + x_2 + x_3 + 2x_4 + x_5 - 6 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_1 + x_2 + x_3 + x_4 + 2x_5 - 6 = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solutions</td>
<td>Performance</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><strong>System</strong></td>
<td><strong>Variable values</strong></td>
<td><strong>EC</strong></td>
</tr>
</tbody>
</table>
| Benchmark -/1      | x0=0.2638927436500653  
x1=0.3803170663929820  
x2=0.2804250886510391  
x3=0.2132412203761027  
x4=0.44438239437353655  
x5=0.14944883430013256  
x6=0.4330924999205027  
x7=0.06746428319148512  
x8=0.3469624642044846  
x9=0.39763460651053917 | 0.0081 | 212 |
| Neurophysiology    | x0=0.773073454805388  
x1=0.5765805512504851  
x2 = 0.6343165089155978  
x3 = -0.8170403098469485  
x4=3.7481129311345285E-13  
x5=3.822719918389339E-12 | 1.79E−11 | 609 |
| Chemical equilibrium application | x0=0.050958721835229426  
x1=1.748330139399421  
x2=0.2715118958827709  
x3=-0.8464707181498703  
x4=0.03582437318962839 | 0.0325 | 670 |
| Combustion Application | x0=7.885401512197632E-6  
x1=-6.28722414571225E-7  
x2=1.7216365890249108E-5  
x3=-5.111503458721245E-6  
x4=7.830654080720478E-6  
x5=1.526760666106192E-5  
x6=7.555224350141998E-6  
x7=1.2783895904533438E-5  
x8=1.2757429127319142E-5  
x9=3.574492433600085E-6 | 2.66E−9 | 297 |
| Benchmark i4       | x0=-0.5077723973445423  
x1=0.615256474526175  
x2=-0.5279178631279993  
x3=0.44796092038237223  
x4=0.667271196118725  
x5=-0.38625078626268783  
x6=0.6572653126601917  
x7=-0.2712236420769336  
x8=0.5881911765836603  
x9=-0.653703315232845 | 0.00281 | 203 |
| Benchmark i5       | x0=-0.501902791893752  
x1=0.6146257966533131  
x2=-0.5293855157439512  
x3=0.44456590561843945  
x4=0.6644074507239301  
x5=0.39586363376169453  
x6=0.6550747173991653  
x7=-0.26249240257821429  
x8=0.5874639200667668  
x9=-0.6531478452401633 | 9.52E−5 | 190 |
| Benchmark i2       | x0=0.22065654006016544  
x1=0.6679777509246896  
x2=0.25308693662292403 | 0.0655 | 834 |
<p>| Benchmark i3 | 0.06911 | 452 |
| Economics modeling application(5) | 9.21E-13 | 97 |
| Economics modeling application(20) | 7.10E-15 | 163 |</p>
<table>
<thead>
<tr>
<th>Economics modeling application(30)</th>
<th>1.77E−15</th>
<th>249</th>
</tr>
</thead>
<tbody>
<tr>
<td>Economics modeling application(100)</td>
<td>1.77E−15</td>
<td>645</td>
</tr>
<tr>
<td>Economics modeling application(200)</td>
<td>7.68E−14</td>
<td>2,046</td>
</tr>
<tr>
<td>Economics modeling application(300)</td>
<td>3.36E−14</td>
<td>2,938</td>
</tr>
<tr>
<td>Economics modeling application(500)</td>
<td>3.16E−14</td>
<td>11,099</td>
</tr>
<tr>
<td>Economics modeling application(1000)</td>
<td>5.13E−14</td>
<td>32,971</td>
</tr>
<tr>
<td>Brown</td>
<td>5.11E−13</td>
<td>87</td>
</tr>
</tbody>
</table>

Table 3: The performance numbers of our algorithm for the benchmarks in Table 2.

Table 3 shows the performance of our algorithm for the systems in Table 2. The performance is presented in terms of values of the variables, the aggregate violations, and the CPU time in milliseconds. We did not show the values of the variables for the economics modeling application.
when the number of the variables exceeds 30. Basically the values for these variables are pretty the same as for 30 variables or fewer.

Based on the figures in Table 3, our algorithm produced solutions with high precision. The aggregate violations are $x \times 10^{-9}$, $x \times 10^{-11}$, $x \times 10^{-14}$, and $x \times 10^{-15}$ (where $x$ is a real number) for most of the systems. For some of the systems, the accuracy was ranging from $7.43 \times 10^{-4}$ (Benchmark i5) to $0.06911$ (Benchmark i3). It is worth mentioning that our algorithm achieved high precision for hard systems of equations: Chemical equilibrium application and Benchmark i2.

To further discuss the performance of our algorithm, we compared it with highly effective algorithms in literature. The comparison includes both the CPU time and the precision of the solution (in terms of aggregate violation). Table 4 compares our algorithm and the Modified Line Search (MLS) [4]. The entry "-" in some of the table’s cells means that there is no reported performance for the corresponding system of equations.

<table>
<thead>
<tr>
<th>Benchmarks (variables)</th>
<th>Our algorithm</th>
<th>MLS (Modified Line Search)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EC</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>Benchmark i1 (10)</td>
<td>0.0081</td>
<td>212</td>
</tr>
<tr>
<td>Benchmark i2 (20)</td>
<td>0.0655</td>
<td>834</td>
</tr>
<tr>
<td>Benchmark i3 (20)</td>
<td>0.06911</td>
<td>452</td>
</tr>
<tr>
<td>Benchmark i4 (10)</td>
<td>0.00281</td>
<td>203</td>
</tr>
<tr>
<td>Benchmark i5 (10)</td>
<td>9.52E-5</td>
<td>190</td>
</tr>
<tr>
<td>Neurophysiology application (6)</td>
<td>1.79E−11</td>
<td>609</td>
</tr>
<tr>
<td>Chemical equilibrium application (5)</td>
<td>0.0325</td>
<td>662</td>
</tr>
<tr>
<td>Combustion Application (10)</td>
<td>2.66E−9</td>
<td>297</td>
</tr>
<tr>
<td>Brown (5)</td>
<td>5.11E−13</td>
<td>87</td>
</tr>
<tr>
<td>Economics modeling application (5)</td>
<td>9.21E−13</td>
<td>97</td>
</tr>
<tr>
<td>Economics modeling application (10)</td>
<td>5.10E−15</td>
<td>132</td>
</tr>
<tr>
<td>Economics modeling application (20)</td>
<td>7.10E−15</td>
<td>163</td>
</tr>
<tr>
<td>Economics modeling application (30)</td>
<td>1.77E−15</td>
<td>249</td>
</tr>
<tr>
<td>Economics modeling application (100)</td>
<td>1.77E−15</td>
<td>645</td>
</tr>
<tr>
<td>Economics modeling application (200)</td>
<td>7.68E−14</td>
<td>2,046</td>
</tr>
<tr>
<td>Economics modeling application (300)</td>
<td>3.36E−14</td>
<td>2,938</td>
</tr>
<tr>
<td>Economics modeling application (500)</td>
<td>3.16E−14</td>
<td>11,099</td>
</tr>
<tr>
<td>Economics modeling application (1000)</td>
<td>5.13E−14</td>
<td>32,971</td>
</tr>
</tbody>
</table>

Table 4: The performance of our technique versus MLS.

Table 4 and Figure 4 indicate that our algorithm outperformed MLS in both solution precision and the CPU time. Figure 4(a) visually compares the errors in the solutions produced by our algorithm and MLS. Our algorithm clearly produced solutions with much smaller errors than MLS. Referring to Table 4, our algorithm produced a solution for the problem Benchmark i1 with an aggregate violation of 0.0081, while MLS produced a solution with accumulative error of 0.22084 for the same problem. Other examples in Table 4 show even much better precision in the solutions produced by our algorithm. For instance, our algorithm produced solutions for Combustion...
Application (10) and Economics modeling application (10) with aggregate violations of respectively 2.66E-9 and 5.10E-15 while MLS produced solutions with violations of respectively 0.01506 and 0.00294 for the same problems.

(a): Comparing the amount of error in solution of each of the problems. The higher the bar the larger the error and the worse the solution is.

(b): The time performance for our algorithm and MLS.

Figure 4: The plot of performance numbers of our technique versus MLS.

Figure 4(b) visually compares the timing numbers of our algorithm versus those of MLS. Generally speaking, our algorithm performed better than MLS. Our technique required much less time to produce solutions for all the problems than MLS did. Referring to Table 4, our algorithm required only 190 milliseconds to produce a solution for Benchmark i5 problem while MLS required 1000 milliseconds. Other numbers in the same table clearly show that our technique required less time in all the problems.

The authors in [6] proposed an effective algorithm called Algorithm 2.4. They compared their algorithm against a large number of algorithms and showed its superiority to the others. The comparison is based on the following two problems [6].
Problem (1)

\[
\begin{align*}
[E_i (x_2 \sin \psi_i - x_3) - F_i (x_2 \sin \phi_i - x_3)]^2 + [F_i (1 + x_2 \cos \phi_i) - E_i (x_2 \cos \psi_i - 1)]^2 - \\
[(1 + x_2 \cos \phi_i)(x_2 \sin \psi_i - x_3)x_1 - (x_2 \sin \phi_i - x_3)(x_2 \cos \psi_i - x_3)x_1]^2 &= 0 \\
\end{align*}
\]

where,

\[
E_i = x_2 \cos \theta_i - \cos \phi_0 - x_2 x_3 \sin \phi_i - \sin \phi_0 - (x_2 \sin \phi_i - x_3)x_1 \\
F_i = -x_2 \cos \psi_i - x_2 x_3 \sin \psi_i + x_2 \cos \psi_0 + x_i x_3 + (x_3 - x_1)x_2 \sin \psi_0
\]

(i = 1, 2, 3)

Where the values of \( \phi \) and \( \psi \) are given in the following table [6].

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \psi_i )</th>
<th>( \phi_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.3554170041747050114</td>
<td>1.7461756494150842271</td>
</tr>
<tr>
<td>1</td>
<td>1.7444828545735749268</td>
<td>2.0364691127919609051</td>
</tr>
<tr>
<td>2</td>
<td>2.06562342666945315689</td>
<td>2.339907786826597820</td>
</tr>
<tr>
<td>3</td>
<td>2.4600678478912500533</td>
<td>2.460067840968093455C</td>
</tr>
</tbody>
</table>

Problem (2)

The Combustion Application problem as defined in Table 2. They used the interval \([0, 1]\) for all the variables instead of \([-10, 10]\).

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Our algorithm</th>
<th>Algorithm 2.4 [8]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EC</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>Problem (1)</td>
<td>4.3E−72</td>
<td>62</td>
</tr>
<tr>
<td>Problem (2)</td>
<td>1.6E−52</td>
<td>211</td>
</tr>
</tbody>
</table>

Table 5: The performance of our algorithm versus Algorithm 2.4

As Table 5 shows, our algorithm outperformed Algorithm 2.4 in terms of CPU time. We used different measure to estimate the aggregate violation. Therefore, it is not possible to compare the precision.

Authors in [21] presented conjugate direction flower pollination algorithm (CDFPA) and compared the performance of this algorithm with other algorithms such as flower pollination algorithm (FPA) and conjugate direction (CD) method. Based on the reported results in [21], the CDFPA performed better than the others. Table 6 shows the cases and the performance of our algorithm versus CDFPA. The entry "." means no reported performance numbers and "\( \approx \)" means close but not equal to. It is clear from the table that our algorithm performed better than CDFPA. Our algorithm found the exact solution in all cases while CDFPA found the exact solution for only one case and approximated solution for the rest. No timing figures reported for CDFPA to compare against.

<table>
<thead>
<tr>
<th>Cases</th>
<th>Our algorithm</th>
<th>CDFPA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EC</td>
<td>CPU time (ms)</td>
</tr>
<tr>
<td>Case 1</td>
<td>( x_1 + 0.99x_2 = 1 ) ( 0.99x_1 + 0.98x_3 = 1 )</td>
<td>0</td>
</tr>
<tr>
<td>Case 2</td>
<td>( H \cdot x = b ) ( h_{ij} = 1/(i+j-1) )</td>
<td>0</td>
</tr>
</tbody>
</table>
Finally we compare our algorithm with the one proposed in [20]. Table 7 shows the systems of equations and the performance of our algorithm compared to that of the algorithm in [20]. Clearly our approach produced better precision. No timing numbers are reported in [20] to compare against.

### Table 6: the performance of our algorithm versus CDFPA

<table>
<thead>
<tr>
<th>Case 3</th>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$200x_1 + 101x_2 = 100$</td>
<td>$x_1^2 - 2x_2 + 1 + 3x_2 = -1$</td>
</tr>
<tr>
<td>$400x_1 + 201x_2 = -100$</td>
<td>$2x_1^2 - 3.9999x_1 + 6.0001x_2 = -1.9999$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>672</td>
<td>13615</td>
</tr>
<tr>
<td>$= 0$</td>
<td>$= 0$</td>
</tr>
</tbody>
</table>

Table 7: The performance of our algorithm versus [20].

<table>
<thead>
<tr>
<th>Equations</th>
<th>Interval</th>
<th>Error in solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2x_1 + x_2 + x_3 + x_4 + x_5 = 6 - 0$</td>
<td>$[-10, 10]$</td>
<td>Our method 4.5E-9 Algorithm[20] “MinError” 5.09E-05</td>
</tr>
<tr>
<td>$x_1 + x_2 + x_3 + x_4 + x_5 = 6 - 0$</td>
<td>$[-10, 10]$</td>
<td>5.09E-05</td>
</tr>
</tbody>
</table>
| $x_1
- x_2 + x_3 + x_4 + x_5 = 6 - 0$ | $[-10, 10]$ | 3.93E-07 |
| $x_1 \parallel x_2 \parallel x_3 \parallel 2x_4 \parallel x_5 = 6 - 0$ | $[-10, 10]$ | 3.1E-14 |
| $x_1x_2x_3x_4x_5 - 1 = 0$ | $[-10, 10]$ | 3.93E-07 |
| $x_1 + x_2 + x_3 + x_4 + x_5 = 6 - 0$ | $[-10, 10]$ | 3.1E-14 |
| $x_1
- x_2 + x_3 + x_4 + x_5 = 6 - 0$ | $[-10, 10]$ | 3.93E-07 |
| $x_1 \parallel x_2 \parallel x_3 \parallel 2x_4 \parallel x_5 = 6 - 0$ | $[-10, 10]$ | 3.1E-14 |
| $x_1x_2x_3x_4x_5 - 1 = 0$ | $[-10, 10]$ | 3.93E-07 |

### Note on the Constraint Violation Measure

Our technique outperformed other techniques in terms of time, precision (measured in terms of the aggregate violation EC), or both. Furthermore, we measure the aggregate violation as the sum
of absolute values of the violations in each individual constraint while others measure the violation in terms of square root of the sum of the square of the individual violation. Our measure of violation is therefore more rigorous than theirs. That is because it can be easily shown that

\[ \sum_{i=1}^{n} |\rho_i(X_i)| \geq \sqrt{\sum_{i=1}^{n} (f_i(X_i))^2} \forall X_i. \]

According to this, our technique would have shown even significantly higher precision if we used the same constraint violation measure as the others.

5. CONCLUSION AND FUTURE WORK

This paper proposed effective approach to find solutions for systems of linear and non-linear equations. Our approach transforms the problem of solving systems of equations into an optimization problem. Our transformation results in an aggregate violation function whose global minimum is the solution for the system of equations. The transformation defines also a criterion function that effectively determines when the aggregate function must be evaluated.

The paper proposed random-guided algorithm to find the solution for the optimization problem and therefore to the corresponding system of equations. The algorithm uses one principal and two marginal directives that effectively search for the global minimum of the optimization problem. These three directives are augmented with biased mapping rules that enable the algorithm to focus the search in the parts of the domain that most likely contain the global minimum without ignoring the other parts of the domains that may contain the minimum.

We conducted many experiments to evaluate our approach. The experiments showed that our proposed technique is very effective in finding solutions with high precision and high speed. We also compared our algorithm with state-of-art algorithms. Our algorithm has better performance in terms of both the solution precision and the CPU time.

We have two directions for future work. First, we would like to conduct more experiments to analyze the effect of tuning the parameters of the algorithm (\(\alpha, \beta, d\)) on its performance. Second, we would like to check the advantage of parallelizing the algorithm so that we can run more than one version of the algorithm using different centers for the directives.

REFERENCES


AUTHOR

Dr. Muhammed Jassem Al-Muhammed holds PhD in computer science, Brigham Young University, USA, 2007. I joined American university of Madaba (Jordan), faculty of information technology in 2013. Prior to this, I worked as a faculty member in Damascus University and international university for science and technology, Syria. Al-Muhammed has published many conference and journal papers. Most of his research interest is in computer security (including cryptography, access control, and so on), problem optimization, and semantic web. Al-Muhammed has also authored many books in different computer fields.