Conjugate direction particle swarm optimization solving systems of nonlinear equations

Yuanbin Mo\textsuperscript{a,*}, Hetong Liu\textsuperscript{b}, Qin Wang\textsuperscript{b}

\textsuperscript{a} School of Mathematics & Computer Sciences, Guangxi University for Nationalities, 530006, Nanning, China
\textsuperscript{b} Science College of China, Jiliang University, 310018, Hangzhou, China

\textbf{ARTICLE INFO}

\textbf{Keywords:}
Systems of nonlinear equations
Conjugate direction method
Particle swarm optimization

\textbf{ABSTRACT}

Solving systems of nonlinear equations is a difficult problem in numerical computation. For most numerical methods such as the Newton's method for solving systems of nonlinear equations, their convergence and performance characteristics can be highly sensitive to the initial guess of the solution supplied to the methods. However, it is difficult to select a good initial guess for most systems of nonlinear equations. Aiming to solve these problems, Conjugate Direction Particle Swarm Optimization (CDPSO) was put forward, which introduced conjugate direction method into Particle Swarm Optimization (PSO) in order to improve PSO, and enable PSO to effectively optimize high-dimensional optimization problem. In one optimization problem, when after some iterations PSO got trapped in local minima with local optimal solution $x^*$, conjugate direction method was applied with $x^*$ as a initial guess to optimize the problem to help PSO overcome local minima by changing high-dimension function optimization problem into low-dimensional function optimization problem. Because PSO is efficient in solving the low-dimension function optimization problem, PSO can efficiently optimize high-dimensional function optimization problem by this tactic. Since CDPSO has the advantages of Method of Conjugate Direction (CD) and Particle Swarm Optimization (PSO), it overcomes the inaccuracy of CD and PSO for solving systems of nonlinear equations. The numerical results showed that the approach was successful for solving systems of nonlinear equations.

\textsuperscript{*} The project was supported by Guangxi University for Nationalities Foundation (No: 200702YJ14).
* Corresponding author.

\textit{E-mail addresses:} moyuanbing\textsuperscript{a}@263.net (Y. Mo), liuht20071984\textsuperscript{b}@163.com (H. Liu).

\begin{thebibliography}{9}

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

Solving systems of nonlinear equations is a problem that we often face with in engineering field, such as in weather forecast, petroleum geological prospecting, computational mechanics and control field etc. It is difficult \cite{1} to solve a system of nonlinear equations, especially for higher-order nonlinear equations, which we haven’t got an efficient and reliable algorithm up to now, though lots of work have done in this area. The Newton’s method and its improved form are extensively used at present, but their convergence and performance characteristics can be highly sensitive to the initial guess of the solution supplied to the methods, and the algorithm would fail if the initial guess of the solution is improper. However, it is difficult to select a good initial guess for most systems of nonlinear equations. For this reason, it is necessary to find an efficient algorithm for systems of nonlinear equations. Let the form of systems of nonlinear equations be

$$\begin{align*}
\end{align*}$$

0898-1221/5 - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2008.10.005
\begin{align}
\begin{cases}
f_1(x_1, x_2, \ldots, x_n) &= 0 \\
f_2(x_1, x_2, \ldots, x_n) &= 0 \\
\vdots \\
f_n(x_1, x_2, \ldots, x_n) &= 0.
\end{cases}
\end{align}

(1)

With the development of the optimization algorithm, some people try to solve systems of nonlinear equations by optimization algorithm. In particular, the advance in evolution algorithm provides greater possibilities for optimization algorithm to solve systems of nonlinear equations; this method can overcome the difficulty of selecting a good initial guess. The system of nonlinear equations (1) is equal to the optimization problem (2).

$$\min f(\mathbf{x}) = \sqrt{\sum_{i=0}^{n} f_i^2(\mathbf{x})} \quad \mathbf{x} = (x_1, x_2, \ldots, x_n).$$

(2)

Generally, the optimization problem that transform from (1) is a high-dimensional optimization problem, and the common optimization algorithm is easy to be trapped into local minima in optimizing it, so the solution of the systems of nonlinear equations that we got would not be accurate. To get an accurate solution for a system of nonlinear equations, the optimization algorithm is required to optimize the high-dimension function problem effectively; we assume that the optimization algorithm is simple, clear and terse. Aiming at these problems, Conjugate Direction Particle Swarm Optimization (CDPSO) is put forward, which combine the advantages of Conjugate Direction method (CD) and Particle Swarm Optimization (PSO). The numerical computations showed that CDPSO could overcome the problem of CD and PSO of getting easily trapping into local minima. The method resulted in accurate solution when it was applied to solving systems of nonlinear equations.

2. Particle swarm optimization and it's analysis

2.1. Particle swarm optimization (PSO)

Particle swarm optimization (PSO) was proposed by Kennedy and Eberhart [2–4], and it is a population-based global optimization algorithm. The basic formulas are as follows:

$$\mathbf{v}_{id}(t + 1) = w \phi_1 \mathbf{v}_{id}(t) + c_1 \phi_2 (\mathbf{P}_{id}(t) - \mathbf{x}_{id}(t)) + c_2 \phi_3 (\mathbf{P}_{gd}(t) - \mathbf{x}_{id}(t))$$

(3)

$$\mathbf{x}_{id}(t + 1) = \mathbf{x}_{id}(t) + \mathbf{v}_{id}(t + 1)$$

(4)

where \( \mathbf{v} \) denotes particle velocity, \( \mathbf{x} \) denotes particle position, \( t \) denotes the number of iteration, \( \mathbf{x}_{id}(t) \) is \( i \)-th particle's \( t \)-th position, \( d \)-th component of \( i \)-th particle position, \( \mathbf{v}_{id}(t) \) is \( i \)-th iteration, \( d \)-th component of \( i \)-th particle velocity, \( \omega, c_1 \) and \( c_2 \) are three adjustable parameters, \( \phi_1, \phi_2 \) and \( \phi_3 \) are random number of (0, 1), \( \mathbf{P}_i(t) \) is the optimal position that \( i \)-th particle has found till \( t \)-th iteration and \( \mathbf{P}_{gd}(t) \) is \( d \)-th component of \( \mathbf{P}_i(t) \), \( \mathbf{P}_{gd}(t) \) is the best of \( \mathbf{P}_i(t) \). Since PSO is simple and easy to apply in reality, it achieved great advance, and was approved by international evolutionary computing field in short-term, and was applied extensively in many domain. At the same time PSO is easy to be trapped into local minima in optimizing high-dimensional function, lots of improved method was proposed. It mainly includes 1. To improve the control parameters, 2. Re-endow the particle’s state variable with new value, 3. Combine with another algorithm, 4. Apply new position and velocity alternate formula, and so on.

2.2. Particle swarm optimization analysis

In this paper, PSO property was analyzed in two regards as follows. Firstly, if there was only one particle and the formula (3) had not the term \( \omega \phi_1 \mathbf{v}_j(j) \), we could infer (5) from (3) and (4).

$$\mathbf{x}_i(j + 1) = \mathbf{x}_i(j) + (\eta_1 \phi_1 + \eta_2 \phi_2) (\mathbf{P}_{id}(j) - \mathbf{x}_i(j))$$

(5)

(5) showed that particle swarm optimization was an extension form of hill-climbing algorithm and the particle went towards optimization direction every time. Secondly, formula (3) showed that every particle's searching direction was the weighted average of \( \mathbf{P}_{id}(j) - \mathbf{x}_i(j) \) and \( \mathbf{P}_{gd}(j) - \mathbf{x}_i(j) \) and \( \mathbf{P}_{gd}(j) \) played an important role in searching direction. Since initial particle distributed evenly in the searching field and every particle searched in one part of the searching field, PSO was effective for the low-dimension function optimization problem. But for high dimension function optimization problem, if the algorithm was trapped into local minima and \( \mathbf{P}_{id}(j) \) did not change over several steps – due to the mutual restriction of each dimensional variable – it was not easy for the algorithm to escape from the local minima, so we could not find the solution. To show that, let \( \mathbf{P}_{id}(1) = \mathbf{P}_{id}(2) \), then:

$$\mathbf{x}_i(2) = \mathbf{x}_i(1) + (\eta_1 \phi_1 + \eta_2 \phi_2) (\mathbf{P}_{id}(1) - \mathbf{x}_i(1))$$

$$\mathbf{x}_i(3) = \mathbf{x}_i(2) + (\eta_1 \phi_1 + \eta_2 \phi_2) (\mathbf{P}_{id}(1) - \mathbf{x}_i(2))$$

$$= \mathbf{x}_i(1) + k (\mathbf{P}_{id}(1) - \mathbf{x}_i(1))$$
where \( k = 2(\eta_1 \phi_1 + \eta_2 \phi_2) - (\eta_1 \phi_1 + \eta_2 \phi_2)^2 \). It showed that if \( P_{gd}(j) \) didn’t change, then the searching direction did not change too, so \( P_{gd} \) could not get improvement and PSO could not jump over local minima. It was similar if the formula had the term \( \omega \phi_1 \psi_1(j) \), so it was necessary to introduce new tactics to improve PSO in optimizing high dimensional function optimization problem. It would improve PSO if we could find a method, which could improve \( P_{gd} \) when PSO was trapped into local minima.

3. Conjugate direction method

Conjugate direction (CD) method was proposed in 1964 by Powell [5]. For problem (2), the basic step of CD are as follows. Step 1. Randomly select one point \( A \) as initial position, and optimize problem (2) along the first coordinate axis. In this step, we optimize single dimensional function optimization once; let \( B \) denote the final solution, and let \( k = 0 \). Step 2. Let \( B \) be the initial position and then optimize the problem (2) along \( n \) coordinate axes one by one; each optimization solution is the initial position for the next optimization. After optimizing \( n \) times, we get position \( C \). It has been proved that direction of \( BC \) was conjugate with direction of the first coordinate axis. Letting \( C \) be the initial position and optimizing the problem (2) along \( BC \), we get the optimization solution \( E \). In this process of optimization, we did \((n + 1)\) times optimization calculation. Substituted direction of the first coordinate axis with \( BC \), we get new \( n \) directions. Let \( k = k + 1 \).

Step 3. Setting \( E \) as the initial position, we go to step 1 and repeat the loop until \( k = n \). Finally we get the optimization solution \( D \); the number of optimization calculations carried out was

\[
1 + (n + 1)(n - 1) = n^2.
\]

It has been proved that \( D \) is nearer the optimization solution of the problem (2) than \( A \), especially if \( f(x) \) of the problem (2) is the quadratic function; \( D \) must be the optimization solution of problem (2).

4. Conjugate direction particle swarm optimization (CDPSO)

4.1. Conjugate direction particle swarm optimization (CDPSO)

CD showed that it can get a better optimization solution than the initial solution. The better the initial solution, the better the optimization solution it would get, and the faster it would converge. But it is difficult to select a good initial solution. In general, initial solution is randomly given, so it would take a long time for CD if we want to get a good optimization solution and sometimes it would fail if initial solution was bad. PSO is a stochastic algorithm, and is analogous to hill-climbing algorithm and the initial particle is more than one particle, so the optimization value of the objective function will decline fast at first when PSO is applied in optimization. But with the development of the calculation, the decline velocity would reduce, and sometimes it gets trapped into local minima, especially if the objective optimization function is a high-dimensional function. To remedy this situation, conjugate direction particle swarm optimization (CDPSO) was proposed. When PSO gets trapped into local minima, we apply conjugate direction method with initial solution \( x^* \) by the property of conjugate direction method, then we get a new solution \( P_{gd}^* \) that is be better than \( P_{gd} \) so it could help PSO to jump over local minima. Substituted \( P_{gd} \) with \( P_{gd}^* \) PSO was applied again and into a new loop, and so on, until termination.

4.2. Trapping into local minima estimation for PSO

In the process of calculation, PSO was regarded as having been trapped into local minima when \( f(P_{gd}) \) did not change over several iterations.

4.3. Practical movement of conjugate direction particle swarm optimization

For problem (2), the main steps of CDPSO are shown as follows:

Step 1. Randomly given \( x_1, x_2, \ldots, x_m \), called initial particle swarm.

Step 2. Apply PSO to optimize the problem (2). If \( f(P_{gd}) \) does not change much over several iterations (generally it is 20 iterations), PSO is regarded as trapped into local minima, and \( x^* \) is the optimization solution at that time.

Step 3. Apply conjugate direction method to optimize problem (2) with initial solution \( x^* \). For the single dimension optimization problems that is brought out by using conjugate direction method, PSO is used to optimize it. The number of initial particle swarm need not be too large. Generally 50 is enough and so it is for the number of iteration. Let the final solution be \( x^{**} \).

Step 4. If the termination condition is satisfied, then stop, if not, substitute \( f(P_{gd}) \) with \( x^{**} \) then go to step 2.
Table 1
The optimization results of Rastrigin by PSO.

<table>
<thead>
<tr>
<th>x_i</th>
<th>1</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>3.0970</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
</tr>
<tr>
<td>x_2</td>
<td>2.8953</td>
<td>1.3899</td>
<td>1.9899</td>
<td>1.9899</td>
<td>1.9899</td>
<td>1.9899</td>
</tr>
<tr>
<td>x_3</td>
<td>1.0142</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
</tr>
<tr>
<td>x_4</td>
<td>1.0251</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
</tr>
<tr>
<td>x_5</td>
<td>0.1993</td>
<td>-0.9950</td>
<td>-0.9950</td>
<td>-0.9950</td>
<td>-0.9950</td>
<td>-0.9950</td>
</tr>
<tr>
<td>x_6</td>
<td>-1.1186</td>
<td>-0.9950</td>
<td>-0.9950</td>
<td>-0.9950</td>
<td>-0.9950</td>
<td>-0.9950</td>
</tr>
<tr>
<td>x_7</td>
<td>0.7129</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
<td>0.9950</td>
</tr>
<tr>
<td>x_8</td>
<td>-1.0702</td>
<td>-0.0000</td>
<td>-0.0000</td>
<td>-0.0000</td>
<td>-0.0000</td>
<td>-0.0000</td>
</tr>
<tr>
<td>x_9</td>
<td>-2.2848</td>
<td>-0.0000</td>
<td>-0.0000</td>
<td>-0.0000</td>
<td>-0.0000</td>
<td>-0.0000</td>
</tr>
<tr>
<td>x_{10}</td>
<td>-0.0676</td>
<td>-2.9849</td>
<td>-2.9849</td>
<td>-2.9849</td>
<td>-2.9849</td>
<td>-2.9849</td>
</tr>
</tbody>
</table>

Table 2
The optimization results of Rastrigin by PSO.

<table>
<thead>
<tr>
<th>x_i</th>
<th>1</th>
<th>200</th>
<th>400</th>
<th>600</th>
<th>800</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td>0.1431</td>
<td>-0.0001</td>
<td>-0.0007</td>
<td>0.0001</td>
<td>-0.0000</td>
<td>-0.0000</td>
</tr>
<tr>
<td>x_2</td>
<td>2.1983</td>
<td>-0.0000</td>
<td>-0.0000</td>
<td>0.0001</td>
<td>-0.0000</td>
<td>-0.0000</td>
</tr>
<tr>
<td>x_3</td>
<td>1.9401</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0001</td>
<td>-0.0000</td>
<td>-0.0000</td>
</tr>
<tr>
<td>x_4</td>
<td>-1.7080</td>
<td>-0.0002</td>
<td>-0.0001</td>
<td>0.0000</td>
<td>-0.0000</td>
<td>-0.0000</td>
</tr>
<tr>
<td>x_5</td>
<td>0.2261</td>
<td>-0.9950</td>
<td>-0.9692</td>
<td>-0.9948</td>
<td>0.0001</td>
<td>0.0001</td>
</tr>
<tr>
<td>x_6</td>
<td>0.9392</td>
<td>0.9950</td>
<td>0.9941</td>
<td>0.9949</td>
<td>0.9950</td>
<td>0.9949</td>
</tr>
<tr>
<td>x_7</td>
<td>-0.1129</td>
<td>0.9494</td>
<td>0.9494</td>
<td>0.9494</td>
<td>0.9494</td>
<td>0.9494</td>
</tr>
<tr>
<td>x_8</td>
<td>-0.1516</td>
<td>0.9950</td>
<td>0.9949</td>
<td>0.9949</td>
<td>0.9949</td>
<td>0.9949</td>
</tr>
<tr>
<td>x_9</td>
<td>-2.1893</td>
<td>-0.0001</td>
<td>0.0000</td>
<td>0.0001</td>
<td>-0.0000</td>
<td>-0.0000</td>
</tr>
<tr>
<td>x_{10}</td>
<td>4.9798</td>
<td>0.9950</td>
<td>0.0000</td>
<td>0.0001</td>
<td>-0.0000</td>
<td>-0.0000</td>
</tr>
</tbody>
</table>

Table 3
The optimization results of Griewank.

<table>
<thead>
<tr>
<th>x_i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSO</td>
<td>80.5684</td>
<td>28.3753</td>
<td>22.6039</td>
<td>47.5712</td>
<td>38.5074</td>
<td>18.7846</td>
<td>39.0690</td>
<td>41.6892</td>
<td>5.4496</td>
<td>33.8004</td>
</tr>
<tr>
<td>CDPSO</td>
<td>0.0288</td>
<td>0.0092</td>
<td>0.0093</td>
<td>0.0088</td>
<td>0.1120</td>
<td>0.2074</td>
<td>0.0705</td>
<td>0.0017</td>
<td>0.0284</td>
<td>0.0115</td>
</tr>
</tbody>
</table>

5. Experiment and results

Three benchmark functions are given to examine the performance of CDPSO in this part.

Test 1: 10 dimensions Rastrigin function

\[ f(x) = \sum_{i=1}^{10} [x_i^2 - 10 \cos(2\pi x_i) + 10] \ |x_i| \leq 5.2. \]

The best result is \( \min f(x) = f(0, 0, \ldots, 0) = 0 \). Apply CDPSO and PSO respectively to optimize it, and three adjustable parameters \( \omega_1 = 0.53, \eta_1 = 0.35, \eta_2 = 0.45 \), the number of initial particle \( k = 300 \) and iteration \( m = 1000 \). Final results are given in Tables 1 and 2.

Test 2: 30 dimensions Griewank function

\[ f(x) = \frac{1}{4000} \sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30} \cos(x_i/\sqrt{i}) + 1 \ |x_i| \leq 600. \]

The best result is \( \min f(x) = f(0, 0, \ldots, 0) = 0 \). With the same parameters, CDPSO and PSO was run 10 times, and the final optimal results of every time were given in Table 3.

Test 3: The Hartman’s Function

\[ f(x) = -\sum_{i=1}^{4} c_i \exp \left[ -\sum_{j=1}^{6} \alpha_i (x_j - p_{ij})^2 \right], \text{ where } 0 \leq x_j \leq 1, c = (1 \ 1.2 \ 3 \ 3.2), \]

\[ (p_{ij}) = \begin{pmatrix}
0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\
0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\
0.2348 & 0.1415 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\
0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381
\end{pmatrix} \]
Case study

Case 1 [6]
\[
\begin{align*}
A &= bh - (b - 2t)(h - 2t) \\
l_p &= bh^3/12 - (b - 2t)(h - 2t)^3/12 \\
&\text{where } A = 165, l_p = 9369, l_n = 6835 \\
f_1(x) &= x_2^2 + x_1^2 - 5x_1x_2x_3 - 85 \\
f_2(x) &= x_3^2 - x_2^2 - x_3 - 60 \\
f_3(x) &= x_1^2 + x_3^2 - x_2 - 2 \\
3 &\leq x_1 \leq 5, \quad 2 \leq x_2 \leq 4, \quad 0.5 \leq x_3 \leq 2 \\
x_1 + \frac{x_2x_3}{4} + 0.75 &= 0 \\
x_2 + 0.405 \exp 1 + x_1x_2 - 1.405 &= 0 \\
x_3 - \frac{x_2}{2} + 1.5 &= 0 \\
x_4 - 0.605 \exp 1 - x_3^2 - 0.395 &= 0 \\
x_5 - \frac{x_2}{2} + 1.5 &= 0 \\
x_6 - x_1x_5 &= 0 \\
f_4(x) &= (3 - 5x_1) + 1 - 2x_2 \\
f_5(x) &= (3 - 5x_1)x_i - x_{i-1} - 2x_{i+1} \quad (i = 2, \ldots, 9) \\
f_6(x) &= (3 - 5x_1)x_{10} + 1 - x_9.
\end{align*}
\]

For case 1, one solution got by CDPSO is \( x = (23.271482, 8.943089, 12.912774)^T \), and one solution for the case 4 is \( x = (0.915551, -0.222256, -0.414654, -0.440697, -0.439254, 0.420892, -0.354588, -0.135767, 0.427562, 0.752203)^T \). Crzyworzka didn't give solution for this case in [8]. Case 2, the solution is \( x = (4, 3, 1) \), and case 3, \( x = (-1, 1, -1, 1, 1) \).

The Newton’s method, CD, PSO and CDPSO were applied, respectively, to solve four problems. For PSO and CDPSO, \( k = 250, m = 300 \), the other parameters are same to test 1. Every algorithm was run 100 times, and the algorithm was regarded as successful if \( e = \max f_i(x) < 10^{-6} \). The successful times are given in Table 5.
7. Conclusions

Using PSO, it is easy to get trapped in the local minima when optimizing multi-dimensional functions. In this paper, we propose CDPSO to overcome that problem. CDPSO combine the advantages of Conjugate Direction Method (CD) and Particle Swarm Optimization (PSO). Experimental results illustrate CDPSO efficiency.

References