J Heuristics
DOI 10.1007/s10732-014-9245-2

An analysis of the velocity updating rule of the particle swarm optimization algorithm

Mohammad Reza Bonyadi · Zbigniew Michalewicz · Xiaodong Li

Received: 4 February 2013 / Revised: 8 March 2014 / Accepted: 13 March 2014

Abstract The particle swarm optimization algorithm includes three vectors associated with each particle: inertia, personal, and social influence vectors. The personal and social influence vectors are typically multiplied by random diagonal matrices (often referred to as random vectors) resulting in changes in their lengths and directions. This multiplication, in turn, influences the variation of the particles in the swarm. In this paper we examine several issues associated with the multiplication of personal and social influence vectors by such random matrices, these include: (1) Uncontrollable changes in the length and direction of these vectors resulting in delay in convergence or attraction to locations far from quality solutions in some situations (2) Weak direction alternation for the vectors that are aligned closely to coordinate axes resulting in preventing the swarm from further improvement in some situations, and (3) limitation in particle movement to one orthant resulting in premature convergence in some situations. To overcome these issues, we use randomly generated rotation matrices (rather than the random diagonal matrices) in the velocity updating

M. R. Bonyadi (✉) · Z. Michalewicz
School of Computer Science, The University of Adelaide, Rm 4.52, Adelaide, SA 5005, Australia
e-mail: mohammad.bonyadi@adelaide.edu.au

Z. Michalewicz
e-mail: zbyszek@cs.adelaide.edu.au

Z. Michalewicz
Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland

Z. Michalewicz
Polish-Japanese Institute of Information Technology, ul. Koszykowa 86, 02-008 Warsaw, Poland

X. Li
School of Computer Science & IT, RMIT University, Melbourne 3001, Australia
e-mail: xiaodong.li@rmit.edu.au

Published online: 01 April 2014

rule of the particle swarm optimizer. This approach makes it possible to control the impact of the random components (i.e. the random matrices) on the direction and length of personal and social influence vectors separately. As a result, all the above mentioned issues are effectively addressed. We propose to use the Euclidean rotation matrices for rotation because it preserves the length of the vectors during rotation, which makes it easier to control the effects of the randomness on the direction and length of vectors. The direction of the Euclidean matrices is generated randomly by a normal distribution. The mean and variance of the distribution are investigated in detail for different algorithms and different numbers of dimensions. Also, an adaptive approach for the variance of the normal distribution is proposed which is independent from the algorithm and the number of dimensions. The method is adjoined to several particle swarm optimization variants. It is tested on 18 standard optimization benchmark functions in 10, 30 and 60 dimensional spaces. Experimental results show that the proposed method can significantly improve the performance of several types of particle swarm optimization algorithms in terms of convergence speed and solution quality.

Keywords Continuous optimization · Swarm intelligence · Particle swarm optimization · Rotation matrices

Abbreviations

ABSQ	Average of Best Solution Qualities: the average of solution qualities over 100 runs of an algorithm when it is applied to a benchmark function
AIWPSO	Adaptive Inertia Weight PSO: one of the variants of PSO
CoPSO	Constriction coefficient PSO: one of the variants of PSO
Decreasing-IW	Decreasing Inertia Weight: one of the variants of PSO
FE	Function evaluation: the number of times that the under-process function is evaluated during the optimization process in one run
GCPSO	Guaranteed Convergence PSO: one the varients of PSO
Increasing-IW	Increasing inertia weight: one of the variants of PSO
PI/SI	Personal influence/social influence: personal and Social Influence (see Eq. 2)
pPSA	Perturbed particle swarm algorithm: one of the variants of PSO
PSO	Particle swarm optimization
Rndm	Diagonal Random Matrices: d*d diagonal matrices with randome values uniformly distributed in [0, 1].
Rotm	Rotation about the origin in all possible planes with predifined angles (α_{ij}) for each plane (see Eq. 10)
Rot(μ,σ)	Rotation about the origin in all possible planes with random angles for each plane normaly distributed ($\alpha_{ij} \sim \mathcal{N}(\mu, \sigma)$) (see Eq. 10)
Rotm	Rotation about the origin in all planes with random angles each of which generated according to normal distribution with appropriate variance (see Fig. 7) and μ=0
Stochastic-IW	Stochastic Inertia Weight PSO: one of the variants of PSO

StdPSO2006	Standard PSO proposed in 2006: one of the variants of PSO
WPSO	Wilke's PSO: one of the varients of PSO

1 Introduction

In this paper we modify the Particle Swarm Optimization (PSO) algorithm to enhance its performance for the following optimization problem:

$$\text{optimize } f(\vec{x}), \ \vec{x} = (x_1, \ldots, x_d) \tag{1a}$$

$$\text{subject to } \vec{l} \leq \vec{x} \leq \vec{u} \tag{1b}$$

$$\text{where } \vec{l} = (l_1, \ldots, l_d), \ \vec{u} = (u_1, \ldots, u_d) \tag{1c}$$

In the rest of the paper, without losing generality, we consider minimization problems only.

PSO is a stochastic population-based optimization method that was developed by Kennedy and Eberhart (1995). The algorithm has been successfully applied to many problems such as function optimization, artificial neural network training, pattern classification, and fuzzy system control (Engelbrecht 2005; Poli 2008), to name a few. Due to the ease of implementation and the fast convergence to acceptable solutions, PSO has received much more attention in recent years (Poli 2008). However, there are still many open issues in this algorithm, e.g. selection of the topology of the swarm, population sizing, and rotation variance (Engelbrecht 2005; Wilke 2005; Poli et al. 2007). Also, several studies have been conducted to analyze the behavior of particles with different coefficients (Shi and Eberhart 1998a,b; Zhang et al. 2005; Jiang et al. 2007) and different topologies (Mendes et al. 2004; Clerc 2006). Moreover, some researchers have analyzed the role of random matrices in the velocity updating rule (Secrest and Lamont 2003; Krohling 2004; Wilke 2005; Clerc 2006; Wilke et al. 2007a,b). This study presents a more in-depth analysis over these random matrices.

Four research contributions reported in this paper are:

(1) identification of several problems caused by multiplying personal and social influence vectors by two random matrices,
(2) presentation of a new approach based on Euclidean rotation matrices with random directions (generated by random normal distribution) for updating the velocity of particles to address these problems,
(3) detailed investigation of the parameters of the normal distribution used in the Euclidean rotation matrices, and
(4) experimentation with an adaptive approach that identifies values of the parameter of the normal distribution during the run. This adaptive approach is independent from the algorithm and the number of dimensions of the problem at hand.

The proposed Euclidean rotation is added to seven PSO variants and these extended algorithms are compared with their original versions on 18 standard optimization functions. Results of these experiments confirm that the proposed Euclidean rotation enhances the performance of the PSO methods for finding better solutions.

The rest of the paper is organized as follows. After a brief review of the basic components of PSO and some well-studied variants of PSO in Sect. 2, the effect of random matrices on the trajectory of particles is investigated in Sect. 3. It is shown that the random matrices presented in the velocity updating rule may have an uncontrollable impact on both the length and direction of the particles' movement. In Sect. 4 a new approach for updating the velocity is presented and the ability of this approach to address the identified problems is tested. This approach introduces an additional parameter σ that is responsible for controlling the direction of particles' movement. The role of this parameter is investigated in Sect. 5. Then the proposed method is incorporated into several PSO variants. These new versions of PSOs are compared with their standard instances. The comparison shows that the proposed approach enhances the performance of all tested PSO variants in the majority of cases. The last section concludes the paper by summing up the advantages and disadvantages of the proposed method.

2 Background

In this section, we present a brief background of PSO. In Sect. 2.1, the original PSO model and its early modifications are reviewed. In Sect. 2.2, several PSO variants that are used in our comparative study are described. These variants are selected because they are either frequently used in literature or recently proposed. In Sect. 2.3, several existing studies on the use of random matrices in PSO are reviewed.

2.1 Basics of PSO

Particle Swarm Optimization (PSO) was introduced by Kennedy and Eberhart (1995). The method processes a population (referred to as *swarm*) of $m \geq 2$ particles; each particle is defined by three d-dimensional vectors:

- *Position* ($\vec{x}_t^i$)—the position of the ith particle in the tth iteration that is used to evaluate the particle's quality.
- *Velocity* ($(\vec{v}_t^i)$)—the direction and magnitude of movement of the ith particle in the tth iteration.
- *Personal best* ($(\vec{p}_t^i)$)—the best position[1] that the ith particle has visited in its lifetime (up to the tth iteration). This vector serves as a memory for keeping knowledge of quality solutions (Eberhart and Kennedy 1995).

In the standard form of PSO (Kennedy and Eberhart 1995), the velocity vector for a particle i is updated according to three other vectors that are the personal best position of the ith particle ($(\vec{p}_t^i)$), the best position found over the whole swarm (known as $(\vec{g}_t^i)$), and the current position of the particle ($(\vec{x}_t^i)$). The formulation for updating the velocity and position of the particles in the original PSO is given as Kennedy and Eberhart (1995):

[1] In general, the personal best can be a *set* of best positions, but all PSO types listed in this paper use a single personal best.

$$\vec{V}_{t+1}^{\,i} = \vec{V}_t^{\,i} + \varphi_1 R_{1t} \underbrace{(\vec{p}_t^{\,i} - \vec{x}_t^{\,i})}_{Personal\ influence\ (PI)} + \varphi_2 R_{2t} \underbrace{(\vec{g}_t - \vec{x}_t^{\,i})}_{Social influence\ (SI)} \tag{2a}$$

$$\vec{x}_{t+1}^{\,i} = \vec{x}_t^{\,i} + \vec{V}_t^{\,i} \tag{2b}$$

In this equation, φ_1 and φ_2 are two real numbers known as acceleration coefficients,[2] and p'' and g'' are the personal best (of particle i) and the global best vector (the best personal best in the swarm), respectively, until iteration t (this model of PSO is usually known as the *global best model* because of the usage of the global best vector). Also, the role of $PI = \vec{p}_t^{\,i}$ (Personal Influence) and $SI = \vec{g}_t^{\,i}$ (Social Influence) vectors is to *attract* the particles to move toward known quality solutions, i.e. personal and global best. Moreover, R_{1t} and R_{2t} are two $d \times d$ diagonal matrices (Clerc 2006; Oca et al. 2009), where their elements are random numbers distributed uniformly ($\sim U(0, 1)$)[3] in [0, 1]. These matrices can be interpreted as *mappings* (any such mapping is called *Rndm* hereafter), which are applied in order to exploit new solutions around the global and personal best vectors and to keep the particles from moving exactly toward them; consequently, they diversify the particles for more effective searches. Also, these matrices are responsible for the direction diversity in the swarm (Wilke et al. 2007a,b). Note that matrices R_{1t} and R_{2t} are generated at each iteration for each particle independently. In Eberhart and Kennedy (1995), another model was introduced in which $\vec{g}_t^{\,i}$ was determined for each particle according to a sub-set of all particles rather than all particles in the swarm (this is called *local best model*).

Shi and Eberhart (1998a,b) introduced a new coefficient ω (*inertia weight*) in Eq. 2a to control the influence of the previous velocity value (this component, $\omega \vec{V}_t^{\,i}$, is called inertia) as follows:

$$\vec{V}_{t+1}^{\,i} = \omega \vec{V}_t^{\,i} + \varphi_1 R_{1t} \underbrace{(\vec{p}_t^{\,i} - \vec{x}_t^{\,i})}_{Personal influence} + \varphi_2 R_{2t} \underbrace{(\vec{g}_t - \vec{x}_t^{\,i})}_{Social influence} \tag{3}$$

where coefficient ω controls the influence of the previous velocity on movement (this is called InertiaPSO throughout the paper). One of the issues in InertiaPSO (as well as the original PSO) is that the velocity vector may grow to infinity if the value of ω, ϕ_1, and ϕ_2 is not set correctly. There are many strategies to prevent the velocity from growing to infinity (Helwig and Wanka 2007). One popular and simple strategy is to restrict the velocity to the range $[-\vec{V}_{max}, \vec{V}_{max}]$, where $\vec{V}_{max}$ is the maximum allowed velocity. Also, in many implementations of PSO, if a particle has left the search space, its objective value is not considered for updating personal and global best vectors. The growth of velocity was studied by Clerc and Kennedy (2002), Trelea (2003), Van den Bergh and Engelbrecht (2006), Poli (2009) through theoretical analysis of the

[2] These two coefficients control the effect of $\vec{p}_t^{\,i}$ and $\vec{g}_t$ on the movement of particles and they play an important role in the convergence of the algorithm. They are usually determined by a practitioner (Oca et al. 2009) or by the dynamic of the particles' movement (Clerc and Kennedy 2002).

[3] Alternatively, these two random matrices are often considered as two random vectors (Kennedy and Eberhart 1995). In this case, the multiplication of these random vectors by *PI* and *SI* is element-wise (Hadamard product).

dynamics of particles. It was found that the convergence behavior of PSO depends on the value of its coefficients (ω, ϕ_1, and ϕ_2). In fact, if these values are set correctly (within a specific range) then particles converge to a fixed point (shown to be the weighted average of $\vec{p}_t^i$ and $\vec{g}_t$) in the search space (Trelea 2003). Note that this fixed point is not guaranteed to be a quality point (Van den Bergh and Engelbrecht 2010).

In the remaining part of this section, some discussions of PSO variants (sub-section *B*) together with the analysis of random matrices (R_{1t} and R_{2t}) in PSO are provided (sub-section *C*).

2.2 PSO variants

There are numerous PSO variants based on different velocity/position updating rules (Liang et al. 2006; Ghosh et al. 2010), different topologies (Mendes et al. 2004; Pso 2006; Xinchao 2010), different parameter values (Clerc and Kennedy 2002; Ratnaweera et al. 2004), different hybridizations (Huang et al. 2010; Wang et al. 2011), and population sizing (Chen and Zhao 2009; Hsieh et al. 2009); all of these have improved the searching ability of the standard PSO. For example, an algorithm called Constriction coefficient PSO (CoPSO) was proposed (Clerc and Kennedy 2002), which defined a new form of the velocity updating rule:[4]

$$\vec{V}_{t+1}^i = \chi\left(\vec{V}_t^i + c_1 R_{1t}\left(\vec{p}_t^i - \vec{x}_t^i\right) + c_2 R_{2t}\left(\vec{g}_t - \vec{x}_t^i\right)\right) \tag{4}$$

The parameter χ was called the *constriction factor.* The authors demonstrated that tuning the values of χ, c_1 andc_2 can prevent the swarm from explosion[5] and can lead to better balance between exploration and exploitation within the search space. They proved that velocity does not grow to infinity if the parameters (χ,c_1 andc_2) satisfy the following equation:

$$\chi = \frac{2}{\left|2 - c - \sqrt{c^2 - 4c}\right|} \tag{5}$$

where $c = c_1 + c_2 > 4$.

Many researchers applied time-based (iteration-based) adaptation approaches to the parameters of PSO. For example, a Decreasing-IW PSO was proposed (Shi and Eberhart 1998a,b) where the value of inertia decreased along the iteration number of the algorithm. In contrast, another approach, named Increasing-IW PSO, was proposed (Zheng et al. 2003); this approach used the same equation with the maximum and minimum of inertia interchanged. In another time-varying approach, called Stochastic-IW PSO (Eberhart and Shi 2001), the value of inertia weight was considered as a uniform random number in the interval [0.5, 1). This interval for the random inertia weight was selected because the expected value of a uniform random number in this

[4] Note that this formulation (Eq. 4) is algebraically equivalent to the one in Eq. 3.

[5] This phenomenon (known also as "drunkard's walk") refers to the unlimited growth of velocity vectors and happens when the control parameters are defined randomly (Clerc and Kennedy 2002).

interval (0.75) is in a good agreement with the value 0.729 that is used frequently in PSO variants for inertia weight.

In Van den Bergh and Engelbrecht (2006) the convergence of CoPSO[6] was investigated and it was shown that, under some conditions, the global best position gets stuck and does not move (this phenomenon is called *stagnation*). In order to address this issue, the velocity updating rule for the global best was revised. This method was called Guaranteed Convergence PSO (GCPSO). The authors proposed the use of local search around the position of the global best particle and set the velocity vector for this particle accordingly. The method was applied to several test benchmarks and the results showed significant improvement compared with CoPSO.

In this paper, the PSO variant introduced in Pso (2006) is considered the standard one and is called StdPSO2006 from now on. This version differs from the version proposed in Shi and Eberhart (1998a,b) only in terms of its topology updating; the global best vector is defined for each particle according to a restricted number of particles (stored in a list called $link_i$ for each particle i) rather than all particles in the swarm (this vector is called the local best vector for particle i). The length of $link_i$ is equal for all i and is a parameter (shown by K) set by the user. At every iteration, if a better solution was not found by the swarm at the previous iteration then $link_i$ is updated for all particles. To update $link_i$ for particle i, K other particles are chosen randomly and stored in $link_i$. The best personal best among particles in the set $\{link_i \cup i\}$ is selected as the local best vector for particle i. The set $link_i$ is updated only if no better solution was found, otherwise, it remains the same. Note that, the set $link_i$ is selected randomly for each particle independently; thus, these sets (for all particles) might have overlap in one iteration while have no overlap (disjoint) in another.

In 2011 an adaptive approach, called the Adaptive Inertia Weight PSO (AIWPSO), was proposed (Nickabadi et al. 2011) where the value of inertia weight was adaptively changed during the search process. The authors proposed a linear relation between the inertia weight in the current iteration and the number of particles improved in the previous iteration. Their results showed a considerable improvement in solution qualities in comparison to other adaptive/time-varying strategies for controlling the inertia.

There are many other variants of PSO (Tu and Lu 2004; Poli et al. 2007) that have emerged over the past 15 years and it would be difficult to discuss all of them in detail. However, in order to have a comprehensive test environment, we confined our tests to nine PSO variants. The variants have been selected in this paper because:

- they have been used frequently in other papers as baselines for comparisons, or
- they have studied the rules of the random matrices in the velocity updating rule.

2.3 Investigation of random matrices

A study of the particles' movement was presented through visualization (Secrest and Lamont 2003). The authors claimed that in PSO (Eq. 3), $\phi_1 R_{1t} PI$ is a vector in the

[6] Note that the model of PSO in that paper was formulated the same as InertiaPSO but the values of parameters were selected according to the formulation in CoPSO.

direction of PI and $\phi_1 R_{1t}$ only influences the length of PI (and the same holds for SI). Consequently, the position of each particle at the $(t+1)$th iteration is somewhere (because of the stochastic nature of the velocity caused by *Rndm*) across a parallelogram defined by the current position, PI, and SI in the tth iteration. However, it is unclear whether the authors used random values or random matrices for the velocity updating rule, because the use of random values would result in getting the same value for all dimensions, so the direction diversity would be ignored.

Two issues related to the use of random matrices were detected in Clerc (2006). The first issue was that the summation of uniform distributions used in the velocity updating rule for *Rndm* mappings was no longer uniform. The second issue was that when personal best ($\vec{p}$) or global best ($\vec{g}$) share the same coordinate value with the position vector ($\vec{x}$) for particle i, PI (or SI) loses that dimension. In this case, the search process becomes a random line search in that dimension. In order to address these problems, several alternative distributions for R_{1t} and R_{2t} were proposed and compared (Clerc 2006).

A comparison between CoPSO and linear PSO (i.e. CoPSO where R_{1t} and R_{2t} are random numbers rather than random matrices) was conducted (Wilke et al. 2007a,b). This comparison showed that the trajectory of the movement of particles in the linear PSO turns into a line search at the very beginning of the search process. The authors claimed that, because of the directional diversity caused by random matrices (instead of random values in the linear PSO), CoPSO does not suffer from this problem. However, it was shown that, under some circumstances, the direction diversity might be lost in CoPSO as well (Van den Bergh and Engelbrecht 2010).

The performance of CoPSO was also investigated in cases where the search space was rotated (Wilke 2005). The authors found that CoPSO is sensitive to rotating the search space, i.e. CoPSO is a rotation variant algorithm. In order to implement a CoPSO model that is rotation invariant, they used random rotation matrices (rather than random diagonal matrices) in the velocity updating rule. They used an approximation idea (an exponential map) for generating the rotation matrices to reduce the computational expenses arising from matrix multiplication. In the exponential map, a rotation matrix M is generated by:

$$M = I + \sum_{i=1}^{max_i} \frac{1}{i!} \left(\frac{\alpha\pi}{180} \left(A - A^T \right) \right)^i \tag{6}$$

where A is a $d \times d$ matrix with elements generated randomly in the interval $[-0.5, 0.5]$, α is a scalar representing the rotation angle, and I is the identity matrix. The generated matrix M is a random rotation matrix with the angle α. It is clear in Eq. 6 that the matrix M becomes a more accurate estimation of a rotation matrix with the angle α when max_i is larger. In Wilke et al. (2007a,b) the value of max_i was set to 1, limiting the approximation to one term of the summation only. Thus, the approximation error of generating the random rotation matrix using the exponential map method grows with the rotation angle (α). Also, the impact of different values of α was not discussed and the value of this parameter was set to 3. Note that for larger values of max_i, the time complexity of generating rotation matrices grows rapidly. The time complexity for generating and multiplying the approximated rotation matrix (with max_i=1) into a vector is in $O(d^2)$, i.e. including generating the

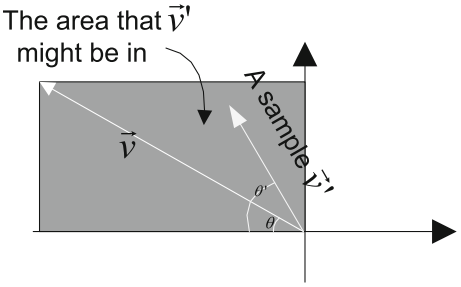


Fig. 1 The results of applying 5,000 random matrices R to vector $\vec{v}$. The *grey area* shows the area that the original vector $\vec{v}$ might be mapped to

matrix ($O(d^2)$) plus vector-matrix multiplication ($O(d^2)$). The rotation matrix was incorporated into CoPSO (the random diagonal matrices in CoPSO were replaced by the random rotation matrices) to test if it could improve the performance of the algorithm when the search space is rotated. The method (hereafter, we use acronym WPSO for Wilke's PSO) was applied to five standard optimization benchmark functions and its results were compared to those of the standard PSO (Wilke 2005). These results showed that the performance of WPSO was not influenced by rotating the search space.

3 Effects of random matrices on vectors

Consider a vector $\vec{v} = (x_0, y_0)'$ in 2-dimensional Cartesian coordinates[7] and consider a 2 dimensional mapping *Rndm* called $R(R = \begin{bmatrix} r_{11} & 0 \\ 0 & r_{22} \end{bmatrix}, r_{11}, r_{22} \sim U(0, 1))$. Also, assume that $\vec{v}^1$ is the result of applying the mapping $R$to $\vec{v}$, i.e. $\vec{v} = R \times \vec{v}(x_0, y_0)$. To investigate the effects of applying R to an arbitrary vector, we generated 5,000 random matrices R and applied them to a vector $\vec{v}$. Figure 1 shows the results of applying R.

All possible locations for the mapped vector $\vec{v}'$ have been represented by the grey area in Fig. 1. Clearly, the mapping R displays some properties when it is applied to an arbitrary vector $\vec{v}$, namely:

Property 1: R does not preserve the length of $\vec{v}$

Property 2: Applying R does not preserve the phase (direction in d dimensions) of $\vec{v}$

Property 3: As $\vec{v}$ becomes closer to one axis, the probability of θ and θ' (the angle between $\vec{v}$ and x axis and $\vec{v}'$ and x axis, respectively) becoming closer to each other increases rapidly; for a vector $\vec{v}$ that is exactly on one axis of the coordinates, no rotation takes place for any R[8]

Property 4: Applying R produces vector $\vec{v}$ in the same quadrant (orthants in multidimensional space) where $\vec{v}$ is in (as the random values are positive)

[7] For the sake of simplicity, the 2-dimensional space is considered for the presented examples. However, the results can be generalized to d-dimensional spaces.

[8] Note that this property holds for d-dimensions as well where θ and θ' are considered as angles between the vector $\vec{v}'$ and an arbitrary axis of the coordinate.

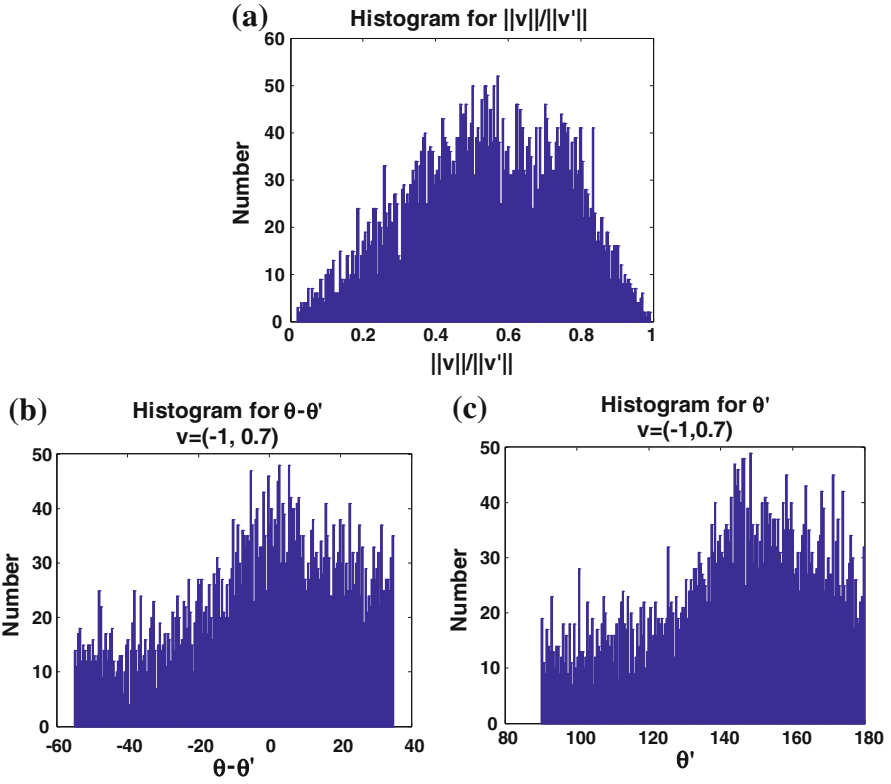


Fig. 2 The effect of 5,000 random mapping R on **a** the length of $\vec{v}$, **b** the changes of the direction of $\vec{v}$ **c** the direction of $\vec{v}$

Let us analyze the behavior of $\|\vec{v}'\|$ and $\theta - \theta\prime$ to visualize the effects of applying R to $\vec{v}$. To illustrate the listed properties, an arbitrary two-dimensional vector $\vec{v} = (-1, 0.7)$ chosen randomly only for experimental purposes, is mapped 5,000 times by randomly generated matrices R. The histograms of $\|\vec{v}'\|/\|\vec{v}\|$, $\theta\prime$, and $\theta - \theta\prime$ have been shown in Fig. 2a–c, respectively. Clearly, the length of $\vec{v}$ (Fig. 2a) is less than or equal to the length of v'' (Property 1). Considering $\frac{\|\vec{v}'\|}{\|\vec{v}\|}$ as a random variable, the probability distribution for this random variable is different with changes of the vector $\vec{v}$. As an example, $P(\frac{\|\vec{v}'\|}{\|\vec{v}\|} < 0.2) \approx 0.2$ for vector $\vec{v} = (-1, 0.007)$, while this value for vector $\vec{v} = (-1, 0.7)$ is around 0.06. Clearly, scaling the length of vector $\vec{v}$ by applying R is vector-dependent. Although the probability distribution for each element of $\vec{v}$ is uniform, there is no guarantee that the length of $\vec{v}$ has a uniform distribution. Thus, it is not possible to assume that applying R has a uniform effect on the length of any vector in the d-dimensional case, and this assumption (having uniform effect on a vector) is correct only in one-dimensional space. Our simulations showed that this distribution is close to uniform for the vectors that are closer to axes of the coordinates while it is not uniform otherwise.

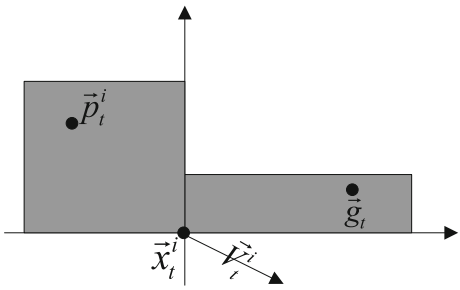


Fig. 3 The effect of *Rndm* on the *PI* and *SI* in PSO. The *grey areas* are the areas to which the personal and global best vectors might be mapped using the *Rndm*

Applying R changes the direction (Property 2) of the vector $\vec{v}$ (Fig. 2b). The distribution of this change is not the same for all vectors and it might be quite different for different vectors. Indeed, the $P(|\theta - \theta\prime| < \gamma)$ (probability of rotating the vector $\vec{v}$ less than γ degrees when it is mapped by an R) depends on the vector $\vec{v}$ and is not controllable.

For example, the value of $P\left(\left|\theta - \theta'\right| < 10°\right) \approx 0.31$ when $\vec{v} = (-1, 0.7)$, while this value is around 0.14 when $\vec{v}$ is $(-1, 0.07)$. Also, as the vector $\vec{v}$ becomes closer to the axes of the coordinates, the probability $P(\left|\theta - \theta'\right| \leq \gamma)$ is rapidly increased. For example, for $\vec{v}$, the probability $P(\left|\theta - \theta'\right| \leq 5°)$ is around 0.16 while this probability for v'' is around 0.997 (Property 3). This increment in probability continues until v'' is exactly on one of the axes. In this case, $P\left(\left|\theta - \theta'\right| \leq \varepsilon\right) = 1$ for any non-negative ε. This means no rotation can take place and these random mappings operate as a random scaling factor. Figure 2.c shows that the angle between the horizontal axis and all vectors $\vec{v}$ is in the interval $[90°, 180°]$. This shows that all vectors $\vec{v}$ are placed in the second quadrant, that is, where vector $\vec{v}$ is found (Property 4).

Now let us look at the general case of PSO. In the velocity updating rule (Eq. 3), the vectors *PI* and *SI* are mapped by two *Rndm*, R_{1t} and R_{2t}. Figure 3 shows possible results of such mapping in a particular instance (with $\phi_1 = \phi_2 = 1.49$).

In Fig. 3, the greyed rectangles represent the areas in which the mapped versions of *PI* and *SI* may be found. Clearly, by applying *Rndm* to *PI* and *SI*, the resulting vectors may be dramatically different from the original ones. So, "the knowledge of quality solutions" (Eberhart and Kennedy 1995) is radically affected and this may cause undesirable behaviors in particles' movement, especially in terms of exploitation of the area around the personal and global bests.

To summarize, we have identified three main problems which are present in the velocity updating rule of PSO when *Rndm* is used.

Problem 1: Uncontrollable changes of the length and direction of *PI* and *SI* (Properties 1 and 2) which results in inefficient exploitation for better solutions in the area around personal and global best.[9]

Problem 2: When *PI* and *SI* are close to an axis (assume that the position of the particle is in the center of the coordinates), the direction freedom offered by the random

[9] Note that, although the changes that result from applying *Rndms* may help the algorithm to jump out of local optima, the controllable changes are more desirable because the exploration/exploitation balance in the algorithm is adjustable.

mappings *Rndm* is very small and limited, hence the exploitation would not be very effective (Property 3).[10]

Problem 3: By applying any *Rndm* to *PI* and *SI*, there is no chance for them to be mapped to other quadrants (Property 4). In fact, the random matrices may confine the exploitation of personal and global bests to just one quadrant which may result in trapping the particles in that quadrant.

To illustrate the issue that Problem 2 may cause, consider the problem of finding the minimum of function $f(x, y) = x^2 + y^2$ (the optimum solution is at$(0, 0)$ and $f(0, 0) = 0$). We experimented with StdPSO2006 using four particles in order to find the optimum solution with the following initialization assumptions:

(1) The position of particles was initialized as a uniform random value in the interval [3.99, 4.01]. The values for the first dimension were randomly generated in the defined function's boundaries.
(2) The second dimension for the velocity vector for all particles was initialized close to 0 (a random value in the interval $[-0.01, 0.01]$).[11]

Our simulation showed that, in this situation, in 97 % of 2,000 runs (each run with four particles and 1,000 function evaluations), the best solution found by the algorithm was around the point$(0, 4)$. This shows that the value for the second dimension could not be changed by the algorithm, which confirms the second identified problem. In fact, in this situation the *Rndm* cannot maintain direction diversity of the movement of particles to guide them to explore the search space.

To illustrate the problem that might be caused by the Problem 3, assume that we want to find the minimum of the following 2-dimensional function:

$$f(x, y) = \frac{(x+1)(y+1)}{((x+1)^2+1)((y+1)^2+1)}, \quad x, y \in [-5, 5] \tag{7}$$

For this function, there is an asymptote with the value of zero when both x and y increased to positive infinity. However, this is actually a local optimum while the best point which minimizes $f(x, y)$ is $(x, y) = (-2, 0)$ with $f(-2, 0) = -0.25$. Consider that all particles are initialized with positive values for their positions and velocities. For the reasons given in the previous paragraph, i.e., the random matrices confine *PI* and *SI* to one quadrant, there is a small chance for the particles to find the optimal point and they can easily be trapped in the local optima. Our simulation showed that, by initializing the positions and velocities of four particles as positive numbers and using the StdPSO2006 for optimization in 25 iterations, the process converges to the local optima in 75 of 100 runs.

To sum up, it is apparent that the mapping *Rndm* significantly influences the search process. In the following section, an approach is proposed to overcome the abovementioned problems. As reported in Van den Bergh and Engelbrecht (2010), meaningful investigations of the exploration/exploitation abilities of PSO are not possible without

[10] A special case of this issue occurs when one of the elements of PI or SI is zero ($\vec{x}$ and $\vec{p}$ or $\vec{x}$ and $\vec{g}$ have equal values for one of their dimensions). In this case, the search becomes a random scale in that dimension (Clerc 2006; Spears et al. 2010; Van den Bergh and Engelbrecht 2010).

[11] Note that this situation may happen with non-zero probability.

considering all parts of the velocity updating equation. However, for simplicity (see also Spears et al. 2010), the inertia component can be excluded from the analysis of the behavior of particles when one of the main three problems (discussed earlier in this section) is studied. In fact, according to Eq. 3, there is no coefficient incorporated to $\omega\vec{v}_t$ for controlling/changing its direction (Problem 1). Thus, once the value for one of the dimensions of the inertia becomes close to zero, it cannot affect the bearing of movement in that dimension in the next iteration. By setting *SI* and *PI* (Problem 2) to small values for that dimension, no further changes in the bearing of movement in following iterations can take place (see earlier example for Problem 2). Also, as was exemplified, by initializing the velocity vector parallel to *SI* and *PI*, the inertia component cannot play an effective role during the following iterations (as was shown in the example for Problem 3).

4 Proposed approach

In order to address all issues identified in the previous section, a new form of the velocity updating rule is proposed; it uses a rotation mapping rather than a random mapping for *PI* and *SI* (similar to the WPSO approach, see Wilke 2005), but with some modifications). The new formula (based on the concept of Euclidean rotation) is defined for the velocity updating rule:

$$\vec{V}_{t+1}^{\,i} = \omega\vec{V}_t^{\,i} + \varphi_1 r_{1t}\, Rotm_t\,(\sigma_1)\left(\vec{p}_t^{\,i} - \vec{x}_t^{\,i}\right) + \varphi_2 r_{2t}\, Rotm_t\,(\sigma_2)\left(\vec{g}_t^{\,i} - \vec{x}_t^{\,i}\right) \tag{8}$$

where $Rotm_t(\sigma_1)$ and $Rotm_t(\sigma_2)$ are two rotation matrices (replacing the random matrices R_{1t} and R_{2t} that were used in earlier PSO variants) and r_{1t} and r_{2t} are two real random numbers, uniformly distributed in [0, 1], which are used to control the magnitude of movement.

The concept of rotation matrices is used to address the identified issues with *Rndm*s. A *rotation matrix* is a $d \times d$ orthogonal matrix where its determinant is equal to 1. In this paper we use the Euclidean rotation matrix defined as follows (Duffin and Barrett 1994):

$$Rot_{a,b}\,(\alpha) = \left\{ \begin{array}{l} r_{a,a} = r_{b,b} = \cos\,(\alpha) \\ r_{a,b} = -\sin\,(\alpha) \\ r_{i,j} | r_{b,a} = \sin\,(\alpha) \\ r_{j,j} = 1, \quad j \neq a,\, j \neq b \\ r_{i,j} = 0, \quad elsewhere \end{array} \right\} \tag{9}$$

As an example, in a two dimensional case, this matrix becomes $\begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$. Note that the Euclidean rotation matrix preserves the length of the vectors during rotation and only changes the direction of the vectors. The random components r_{1t} and r_{2t} are responsible for the changes of the length of the vectors in Eq. 8.

$Rot_{a,b}(\alpha)$ rotates a vector in the $x_a x_b$ plane by angle α about the origin while it preserves its length (Chow 2000). To rotate a vector in all possible planes, all

corresponding rotation matrices are multiplied by each other. We define $Rot\ (\mu, \sigma)$ as follows:

$$Rot\ (\mu, \sigma) = \prod_{1 \leq i < j \leq d} Rot_{i,j}\left(\alpha_{i,j}\right) \tag{10}$$

where $Rot(\mu, \sigma)$ is a rotation matrix which rotates a vector in all planes $x_i x_j$ about the origin with the angle $\alpha_{i,j}$ that is a normally distributed random number, with the mean μ, and standard deviation σ, i.e. $\{\alpha_{ij}, \forall i, j\} \sim \mathcal{N}(\mu, \sigma)$. In this paper we set $\mu = 0$, and by this setting, the direction changes caused by rotation are centralized around the direction of the original vector. We use also the following notation: $\text{Rotm}(\sigma) = \text{Rot}(0, \sigma)$. By multiplying a vector $\vec{v}$ with this rotation matrix, the vector is rotated around the origin while its length remains unchanged.

Note that Eq. 10 forces some computational overhead on the velocity calculations (see Eq. 11) arising from matrix multiplication. This overhead depends on the number of matrices (in our case, $d(d-1)/2$ which is the number of possible planes in a d dimensional space). In fact, the time complexity for generating $Rotm(\sigma)$ is $O(d^5)$. However, the implementation proposed in this paper reduces the time complexity to $O(d^2)$, which is the same as generating a d by d matrix (which is used in the standard PSO), while calculations are still accurate (see "Appendix III" for details on this implementation).

Because of this computational overhead, one can consider selecting only one random plane in each iteration for each particle and conducting the rotation only in that plane rather than all possible planes (it is called *RotmP* throughout the paper). This method is faster than *Rotm* in terms of computational complexity. In fact, the complexity of selecting one of the planes and rotation in that plane is $O(d)$. We compare these two approaches (*Rotm* and *RotmP*) in the experimental results section.

The proposed approach differs from WPSO in several aspects. First, WPSO uses an approximation method for generating rotation matrices, which causes inaccuracy in rotating the vectors, whereas the proposed approach can accurately generate the rotation matrices in the same order of complexity. Second, the exponential map method used in WPSO for generating rotation matrices does not preserve the length of movement (due to the usage of approximation) when rotating the vectors, whereas the proposed method keeps the length unchanged. Third, in WPSO, the error of the approximation of the rotation matrices grows with the angle of rotation, whereas the proposed method is accurate for any value of rotation angles (this has been pointed out as a drawback of the exponential map method for generating rotation matrices in Wilke et al. (2007a,b)).

As mentioned earlier, the lengths of the vectors which are mapped by $Rotm(\sigma)$ are not changed. Also, rotation of the vector is independent from the orientation of the coordinate axes. Based on the properties of a normal distribution, 99.7 % of changes in the vector's angles are in the interval $[-3\sigma, 3\sigma]$ in each plane. Indeed, the $P(|\theta - \theta'| < \gamma)$ (the probability of rotating the vector less than γ degrees) is the same for all vectors, and it is calculable and also controllable, by σ.

To exemplify these claims, consider the vector $\vec{v} = (-1, 0.7)$ from Sect. 3, i.e. $\vec{v}$.

Figure 4a shows vector $\vec{v}$ and all other vectors (the vectors in the grey area) are the result of applying $Rotm$ (4) to $\vec{v}$ in 5,000 observations (compare Fig. 1, 4a). Figure 4b

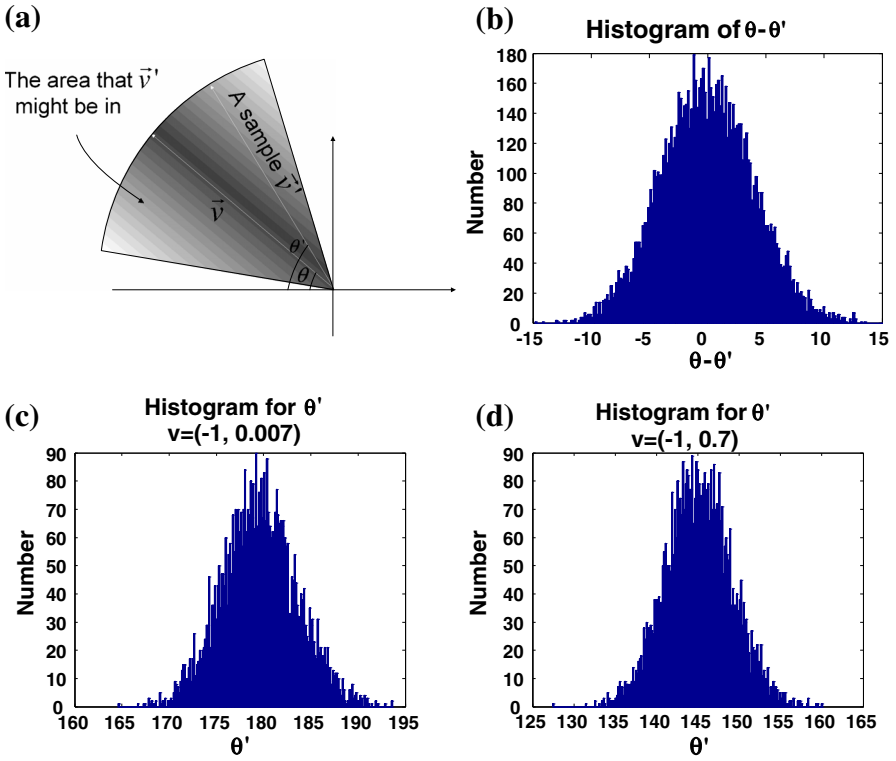


Fig. 4 The effects of 5,000 generated *Rotm(4)* on **a** vector $\vec{v} = (1, 0.7)$, **b** the changes of direction of $\vec{v} = (1, 0.7)$, **c** direction of v'', and **d** direction of $\vec{v} = (1, 0.007)$, $\vec{v} = (1, 0.7)$ is a random sample

shows the histogram for $\theta - \theta$', where θ and θ' are the angle between $\vec{v}$ and x axis and $\vec{v}$ and x axis, respectively. The figure indicates that the angle (direction) has been changed between $-15°$ and $15°$. Moreover, the rotation matrices have changed the angles more than 10 degrees in just 0.8 % of observations ($P(|\theta - \theta'| < 10)$ is 99.2 %). Our experiments showed that this value is the same for the vector $\vec{v} = (-1, 0.007)$, which was not the case when the *Rndm* mapping was used (see Fig. 2). Figure 4c shows that by applying the rotation matrix to a vector that is close to some coordinate axes, e.g. $\vec{v} = (-1, 0.007)$, the quadrant may be changed (the angle of the mapped version can be more than 180°). Figure 4d shows the histogram of θ' when 5,000 rotation matrices *Rotm(4)* were applied to the vector $\vec{v} = (-1, 0.7)$.

To summarize, the rotation matrix approach preserves the length of the vector $\vec{v}$ while it rotates $\vec{v}$ according to a predefined rotation angle. This angle can be a random variable normally distributed with a mean 0 and standard deviation σ (by setting μ=0, the mapped vectors will be distributed around the original vector following a Gaussian distribution).

The application of the mapping *Rotm*(σ) has the following properties:

Property 1: The lengths of *PI* and *SI* are not changed by Rotm(σ) and they are controlled by $\phi_1 r_{1t}$ and $\phi_2 r_{2t}$, respectively.

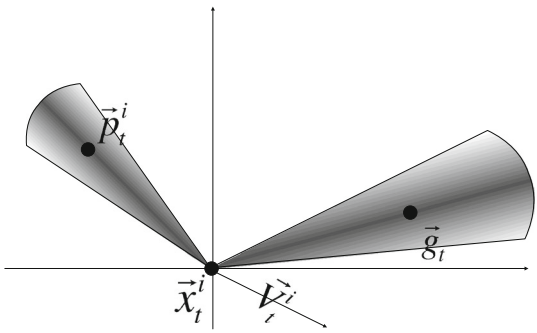


Fig. 5 The effect of *Rotm(4.6)* on the personal and global bests vectors in PSO. The *grey areas* are the areas to which the personal and global best vectors might be mapped to using this rotation

Property 2: Direction of *PI* and *SI* is controlled by σ.

Property 3: The rotation of a vector is independent of the orientation of the coordinate axes.

Property 4: The *Rotm*(σ) can rotate the *PI* and *SI* to a new quadrant.

In the velocity updating rule given by Eq. 8, the direction of each part is changed in all planes according to σ while the effects of the length of each part are controlled only by ϕ and r. Figure 5 shows an instance when *Rotm(4.6)* is used in the PSO updating rules (4.6 has been chosen only for the example purposes).

In Fig. 5, the shadowed areas are the possible areas to which the *PI* and *SI* might be mapped after applying *Rotm*(σ) (while $\phi_1 = \phi_2 = 1.49$ and $\sigma_1 = \sigma_2 = 4.6$). Also, the shaded spectrum of these areas shows the probability of mapping the original vector to the area, i.e., the darker the area is, the higher the probability that the mapped vector will be in that area.

In the rest of the paper we use a simpler version of Eq. 8, where $\sigma_1 = \sigma_2 = \sigma$:

$$\vec{V}_{t+1}^{\,i} = \omega \vec{V}_t^{\,i} + Rotm_t\,(\sigma)\left(\varphi_1 r_{1t}\left(\vec{p}_t^{\,i} - \vec{x}_t^{\,i}\right) + \varphi_2 r_{2t}\left(\vec{g}_t - \vec{x}_t^{\,i}\right)\right) \tag{11}$$

Note that this approach can be used in the velocity updating rule of other types of PSO as well, by simply replacing any *Rndm* with *Rotm*. Hereafter, any type of PSO (PSO*) that uses the approach of rotation matrices is called PSO*-*Rotm*, e.g., when the proposed approach is used in CoPSO we call the method CoPSO-*Rotm*.

To show that this approach can effectively address Problem 2 of the velocity updating rule identified in the previous section, we applied the StdPSO2006-*Rotm* (with $\sigma = 20$, which was determined experimentally) to function $f(x, y) = x^2 + y^2$. Also, we initialized all four particles through the same procedure described in the previous section. In this situation, the algorithm could find optimal solutions in 99 % of 2,000 runs; this shows that the algorithm can offer better direction diversity in comparison to the StdPSO2006.

Now, let us return to the optimization problem given by Eq. 7. StdPSO2006-*Rotm* (with $\sigma = 20$) was trapped in the local optima in only 23 out of 100 runs (as opposed to 75 in the original version of StdPSO2006). Also, by considering $\sigma = 28$, the algorithm failed in only 10 out of 100 runs (see Sect. 5.3 for the details of setting of the parameter σ).

Thus rotation matrices address all problems related to the velocity updating rule identified in the previous section; further, the introduction of rotation matrices makes PSO rotation invariant, as already reported in Wilke (2005).

In the following section, the new velocity updating rule based on $Rotm(\sigma)$ rather than *Rndm* is tested on several well-studied PSO variants and the results are reported. These experiments show that the proposed approach improves the overall ability of the PSO to explore better solutions in the search space. The only parameter that has been added to the algorithm is σ, and it is investigated in more detail. For this parameter, a sensitivity analysis is performed to set its value for different algorithms. Also, a simple adaptive method is proposed to make the algorithm less problem-dependent.

5 Experimental results

In the following four subsections we describe the experimental setup, discuss the benchmark functions selected for experimentation, provide a sensitivity study of the parameter σ, and report on experimental results.

5.1 Experimental setup

All variants were implemented under the MatlabR2011a environment. Due to the stochastic nature of the PSO-based methods, the reported results for each algorithm are the averages over 100 runs. We analyze the effect of the parameter σ on the performance of different PSO-based methods. After that, in order to show the efficiency of the proposed rotation matrix approach, three sets of comparisons are considered.

- In the first set of comparisons, the proposed approach (including *Rotm* and *RotmP*) is incorporated into CoPSO (i.e. CoPSO-*Rotm* and CoPSO-*RotmP*) and compared with two other methods CoPSO-*Rndm* (Clerc and Kennedy 2002), WPSO (Wilke 2005). The purpose of this test is to find which type of random matrices (Euclidean rotation in all planes, Euclidean rotation in one plane, uniform, or exponential map rotation) is more effective. Also, to have a fair comparison among these methods, in this test *Rotm* and *RotmP* are incorporated to CoPSO because WPSO is an extension of CoPSO. Moreover, the parameter σ for *Rotm* and *RotmP* is fixed to a single value as this parameter is fixed in WPSO as well.
- We continue the comparisons by incorporating *Rotm* into several well-known PSO algorithms, namely CoPSO, StdPSO2006, Decreasing-IW, Increasing-IW, Stochastic-IW, AIWPSO, and GCPSO (Shi and Eberhart 1998a,b; Eberhart and Shi 2001; Clerc and Kennedy 2002; Zheng et al. 2003; Pso 2006; Van den Bergh and Engelbrecht 2010; Nickabadi et al. 2011). They are then compared with their corresponding original versions. The purpose of this test is to show the effect of the usage of rotation matrices on well-known PSO algorithms. In this test, all parameters are kept the same as the original parameters for each type while the proposed approach is incorporated into these methods. These PSO algorithms were selected for comparison purposes because they provide quality results and they have been frequently used by other researchers in their experiments.

Table 1 The parameters used for the selected methods for comparison

Algorithm name	Velocity updating
CoPSO (Clerc and Kennedy 2002)	Equation 4, $c_1 = c_2 = 2.05$
Increase-IW (Zheng et al. 2003)	Equation 3, $\omega = 0.4 \rightarrow 0.9$, $\phi_1 = \phi_2 = 2$
Decrease-IW (Shi and Eberhart 1998a,b)	Equation 3, $\omega = 0.9 \rightarrow 0.4$, $\phi_1 = \phi_2 = 2$
Stochastic-IW (Eberhart and Shi 2001)	Equation 4, $\omega = [0.5, 1]$, $c_1 = c_2 = 2.05$
AIWPSO (Nickabadi et al. 2011)	Equation 3, $\omega = adaptive$, $\phi_1 = \phi_2 = 1.492$
StdPSO2006 (PSO 2006)	Equation 3, $\omega = \frac{1}{2\ln(2)}$, $\phi_1 = \phi_2 = 0.5 + \ln(2)$, K=3
GCPSO (Van den Bergh and Engelbrecht 2010)	Equation 4, $c_1 = c_2 = 2.05$
WPSO (Wilke et al. 2007a,b)	Equation 4, $c_1 = c_2 = 2.05$

These values have been extracted from the original papers

- In the last set of comparisons, the proposed adaptive approach is tested when it is used to set the value of σ in StdPSO2006-*Rotm* and AIWPSO-*Rotm*. The aim of this test is to evaluate the proposed adaptive approach for the variable σ.

These tests are conducted on several standard optimization functions that are introduced in Sect. 5.2. A performance metric called ABSQ (Average of Best Solution Quality) is reported for each test. ABSQ is the average of objective values that an algorithm achieves in a predefined number of function evaluations (FE) over 100 runs. The maximum allowed number of FEs is fixed (10^4 in all experiments, unless it is specified in the test), the algorithm is run, and the best found solution in this time frame is recorded as BSQ. This procedure is repeated 100 times and the average is reported as ABSQ. The parameters used for each method are listed in Table 1.

Note that the same parameters were used in both versions of each method (PSO* and PSO*-*Rotm*); the number of particles for all methods is set to 20 in all tests. Also, velocity clamping was used to limit the value of each dimension of the velocity vector in all experiments (the value for V_{max} for each dimension i was set to $\frac{\max([l_i|,|v_i|)}{2}$). Moreover, the objective value for a particle is not updated if the particle is in the infeasible region (out of the boundaries). This bound handling technique is known as Infinity bound handling technique (Helwig et al. 2013). Note that, the Infinity bound handling together with velocity clamping is frequently used in the literature, however, it is not the best possible choice for handling particles which are out of the boundaries, see Helwig et al. (2013) for detailed analysis.

In some tests, the percentage of improvement (impairment) has been reported. This percentage is calculated by the following formula:

$$\frac{z^* - z}{z^*} * 100 \tag{12}$$

where z^* and z are the averages of ABSQ over a set of functions for two under-comparison algorithms, respectively and $z^* > z$.

Table 2 Functions used for tests

Function name	F	Rotated F	Global optimum	$f(x)$	Search range
Rosenbrock	f_1	f_{10}	[1, 1, …, 1]	0	$[-2.048, 2.048]^d$
Rastrigin	f_2	f_{11}	[0, 0, …, 0]	0	$[-5.12, 5.12]^d$
Ackely	f_3	f_{12}	[0, 0, …, 0]	0	$[-32.768, 32.768]^d$
Weierstrass	f_4	f_{13}	[0, 0, …, 0]	0	$[-0.5, 0.5]^d$
Griewank	f_5	f_{14}	[0, 0, …, 0]	0	$[-600, 600]^d$
Sphere	f_6	f_{15}	[0, 0, …, 0]	0	$[-100, 100]^d$
Non-continuous Rastrigin	f_7	f_{16}	[0, 0, …, 0]	0	$[-5.12, 5.12]^d$
Quadratic (Hyper-Ellipsoid)	f_8	f_{17}	[0, 0, …, 0]	0	$[-100, 100]^d$
Generalized Penalized	f_9	f_{18}	[1, 1, …, 1]	0	$[-50, 50]^d$

5.2 Test functions

In order to compare the proposed approach to other methods, some benchmark optimization problems were used. This benchmark includes 9 well-studied functions selected from Yao et al. (1999), Tu and Lu (2004), Suganthan et al. (2005) and their rotated versions. The formula for all functions can be found in "Appendix I". Due to the fact that some of these test functions are solvable by d one-dimensional searches, Salomon's method (Salomon 1995) is applied to rotate them. This procedure is realized by generating an orthogonal matrix M that is multiplied by the input variables x to get the rotated variable $y = M * x$. The new variable y is used to determine the objective value for particles. This type of rotation increases the complexity of the function so they will not be solvable by a simple search.

Table 2 shows the bounds for all functions. In addition, the location for the best solution and the best objective value for all functions are reported in this table. f_{1-9} are the basis functions while f_{10-18} are their rotated versions (expressions for these functions can be found in "Appendix I").

The optimal solution for most of these functions is in the center of coordinates. To test the performance of the algorithms when the optimal solution is not on the center, we add a vector o to shift these functions so that their optimal solutions are not on the center. This shift only takes place for the last set of tests in this paper. The vector o was generated randomly for each function separately in such a way that the optimal solution is still in the search range. The vector o was generated once for each function and it was fixed for each function for all tests.

5.3 Parameter setting

The standard deviation of the rotation angle in the proposed rotation matrix (*Rotm*) is the only parameter that has been added to the PSO algorithms. In this section, a method for finding the best value for this parameter is discussed.

The setting of parameters is a challenging issue whenever algorithms are compared. Parameters for algorithms need to be either tuned for each algorithm or the use

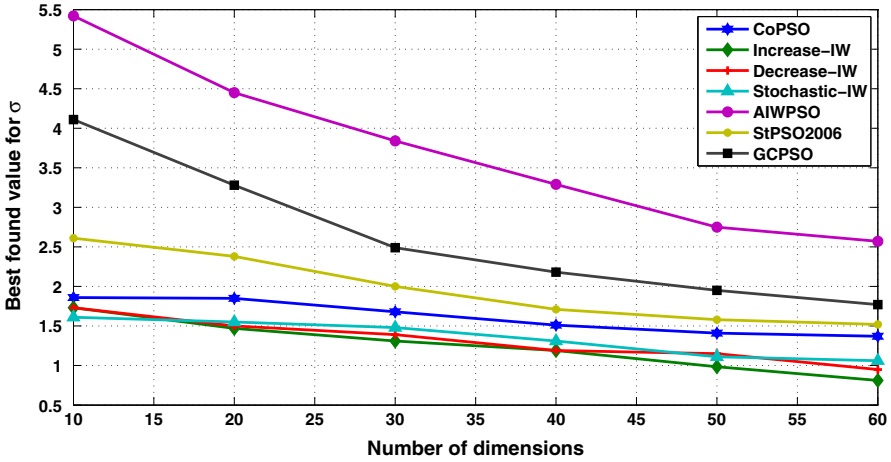


Fig. 6 Setting the parameter σ for seven various types of PSO

of identical parameters needs to be justified. Because seven PSO variants are considered for extension by using the proposed rotation matrices, the value of σ should be investigated for all of these algorithms independently. Also, several parameters such as population size remain the same across algorithms in all tests. Furthermore, since the focus of the paper is not the comparison of algorithms, but to show that different algorithms can benefit from the new rotation matrix, the individual parameter settings (apart from σ) for each algorithm don't matter as much.

Also, in order to have a valid setting procedure, two standard functions, a unimodal (Sphere) and a multimodal (Griewank) function were selected for this purpose. To set the parameter σ for each algorithm, we incorporated the proposed rotation matrix into the algorithm, fixed the number of dimensions and applied the method to both functions (Griewank and Sphere) to find the best performance for various values of σ.

Our investigations showed that the best values for σ are in the interval [0, 7] for all of these methods. Hence, we changed the value of σ in this interval with a step size 0.01 (701 steps overall). In each run, one of the algorithms was applied to the under-process function (two mentioned functions) 50 times (for a fixed number of dimensions, $701 \times 50 \times 2$ tests were done). This process was done for 10, 20, 30, 40, 50, and 60 dimensional problems. In total, for each algorithm, $701 \times 50 \times 2 \times 6$ tests were run to find the best value for σ. The results for all setting procedures for all methods are shown in Fig. 6. Note that because the best values for both functions were very close, the average of the objective values is considered to be the most likely determiner of the best value of σ. Figure 6 presents this average of best values for σ for seven PSO variants. It is obvious that the best value is different for different methods and for different numbers of dimensions. Hereafter, the selected values for σ in all tests are the ones that have been presented in Fig. 6.

One of the reasons behind the reduction of the best σ when increasing the number of dimensions can be related to the overall angle changes in the higher dimensions. In fact, by increasing the number of dimensions, the number of planes in the space is increased. As the rotation of the velocity vector is applied to each plane independently

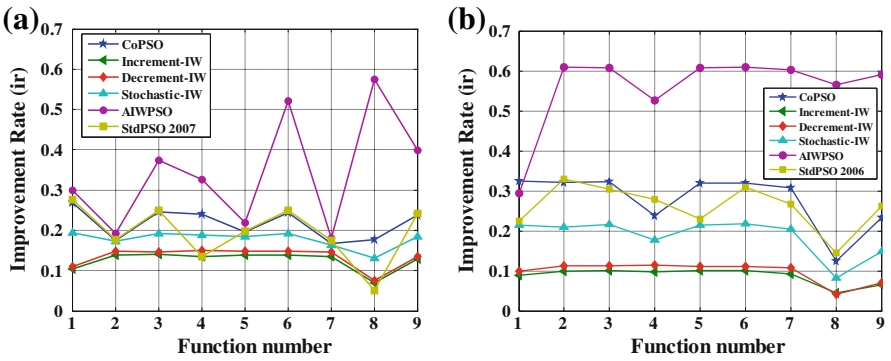


Fig. 7 The average improvement rate (ir) of six PSOs when they are applied to 9 standard benchmark functions in Table 2. **a** 10 dimensional cases, **b** 30 dimensional cases

(see Eq. 8), the overall applied rotation to this vector is increased by the number of dimensions. Thus, a smaller rotation is needed as the number of dimensions increases. Also, Fig. 6 indicates that the best values of σ for different algorithms are different, especially for AIWPSO. Although the reason behind this phenomenon is not clear, our experiments show that, in each algorithm, the best value of σ is correlated with the improvement rate of the particles (ir), defined as the number of improved particles divided by the population size. In order to find the relation between σ and ir, we recorded the average of ir in 10,000 function evaluations for each algorithm (six PSO variants) when they were applied to f_{1-9} (Table 2) over 100 runs for each function. Figure 7 shows the results.

Figure 7 The average improvement rate (ir) of six PSOs when they are applied to 9 standard benchmark functions in Table 2. (a) 10 dimensional cases, (b) 30 dimensional cases

A simple comparison between the improvement rate ir of the methods (Fig. 7) and the corresponding σ (Fig. 6) indicates a positive correlation between these variables. For example, the algorithm AIWPSO, which has the largest σ in Fig. 6, has also the largest value for ir on average (Fig. 7).

Based on the correlation between σ and ir and d, a simple adaptive approach (Eiben et al. 1999) is proposed for σ to make the algorithm independent of the problem and the number of dimensions. According to Figs. 6 and 7, we can assume that the value of σ is dependent on the number of dimensions d and the current improvement rate (ir_t).[12] Therefore, we can say that $\sigma_t = h(ir_t, d)$ where h is a function that relates the value of σ at iteration t to the number of dimensions (d) as well as ir at iteration t.

As reported in Fig. 6, appropriate values for σ become smaller as the number of dimensions grows. Hence, it is safe to assume that there is a negative correlation between these variables. Also, the appropriate value of σ becomes bigger when the used algorithms have a higher average value for ir (Fig. 7). Obviously, function h can be defined in many ways to reflect the discovered correlations. In this paper,

[12] Note that these are the parameters that have been considered in this paper and there might be some others that have an effect on the value of σ.

we defined h as follows:

$$h\left(ir_t, d\right) = \frac{\alpha * ir_t}{\sqrt{d}} + \beta \tag{13}$$

In this formula, α and β are two constants, which bound the value of σ to the interval$[\beta, \frac{\alpha}{\sqrt{d}} + \beta]$. By using this equation the correlations between σ and d as well as ir_t are taken into account. The value of β prevents σ from becoming 0. Also, the square root of d in the denominator of the formula reduces the speed of reduction by d(as was observed in Fig. 6).

Our experiments showed that $\alpha = 30$ and β=0.01 works appropriately for our cases. This adaptive approach is added to some PSO variants and its results are reported.

5.4 Results analysis

The algorithms are compared in terms of their ABSQ. To provide an accurate comparison among these algorithms, the Wilcoxon's test (Wilcoxon 1945) with 5 % level of significance is conducted. The p value of the test has been used in order to show the significance of the differences in results in each case. To make a fair comparison, all parameters/coefficients for all algorithms are the ones taken from their original versions (see Table 1), regardless of the random matrices they use.

In the first test, the proposed rotation matrices (both *Rotm* and *RotmP*) are compared with two other methods, CoPSO-*Rndm* and WPSO. Table 3 shows the results of applying these methods to eighteen 10-dimensional test functions. In order to have a fair comparison among these methods, *Rotm* and *RotmP* are incorporated into CoPSO and their results are compared with WPSO and CoPSO-*Rndm* (recall that WPSO is an extension of CoPSO). Moreover, the value of σ is kept constant ($\sigma = 2$) as WPSO used a constant value for σ. Also, the number of FEs was set to 200,000. Each algorithm has been compared with the others based on the Wilcoxon's test. The result of an algorithm that is significantly better than the result of another algorithm is indicated by the superscript letters R, P, N, or W for CoPSO-*Rotm*, CoPSO-*RotmP*, CoPSO-*Rndm*, and WPSO, respectively.

The performance of CoPSO-*Rotm* is significantly better than CoPSO-*RotmP* in 10 functions while it is significantly worse in 8 functions. Thus, it seems it is slightly beneficial to conduct rotation in all planes (*Rotm*) rather than only one plane (*RotmP*) randomly. Hence, from here on, we only experiment with *Rotm* approach to test if it can improve the performance of the other PSO variants. Also, CoPSO-*Rotm* is significantly better than WPSO in 16 functions out of 18. This shows that the proposed *Rotm* offers better performance compared to the exponential map approach proposed in Wilke et al. (2007a,b). In addition, CoPSO-*Rotm* performs better than CoPSO-*Rndm* in 14 functions out of all 18. Thus, it is clear that the proposed *Rotm* is more effective than other tested methods for generating random rotation matrices for PSO variants. Note that, because the parameters of WPSO have been taken from the literature and they have not been set for the used benchmark set, this comparison between different algorithms has limited validity. In order to have a fair comparison, we have set all parameters for both methods (rotation angle, acceleration coefficients, and inertia weight) to equal values and the base methods are also identical.

Table 3 Results from comparison between CoPSO-*Rotm* (the CoPSO that uses the rotation matrices), CoPSO-RotmP, WPSO, CoPSO-*Rndm*

Function	CoPSO-*Rotm*	CoPSO-*RotmP*	WPSO	CoPSO-*Rndm*
Non-rotated functions				
f_1	12.01	**0.1993**$^{\text{CWN}}$	11.9	9.93$^{\text{CW}}$
f_2	**3.55E−17**$^{\text{PWN}}$	9.68E−15	7.12E−15	8.39E−17$^{\text{PW}}$
f_3	**6.77E−15**$^{\text{PWN}}$	2.77E−14$^{\text{N}}$	5.45E−14$^{\text{N}}$	1.33E−13
f_4	19.12$^{\text{W}}$	4.898$^{\text{CWN}}$	21.04	19.28$^{\text{W}}$
f_5	**1.11E−16**$^{\text{PWN}}$	4.72E−16$^{\text{WN}}$	2.91E−15$^{\text{N}}$	8.21E−15
f_6	**4.35E−195**$^{\text{PWN}}$	2.3E−181$^{\text{WN}}$	3.75E−172$^{\text{N}}$	3.87E−109$^{\text{W}}$
f_7	**0.13914**$^{\text{PWN}}$	0.4770$^{\text{WN}}$	2.014	0.92$^{\text{W}}$
f_8	1.19E−43$^{\text{WN}}$	**1.9E−132**$^{\text{CWN}}$	1.01E−42$^{\text{N}}$	8.32E−18
f_9	13.07$^{\text{WN}}$	**2.952**$^{\text{CWN}}$	47.55	14.45$^{\text{W}}$
Rotated functions				
f_{10}	12.08$^{\text{N}}$	**0.1993**$^{\text{CWN}}$	11.71$^{\text{CN}}$	17.6
f_{11}	**8.88E−17**$^{\text{PWN}}$	1.81E−14	5.50E−15$^{\text{P}}$	4.27E−15
f_{12}	**5.06E−15**$^{\text{PWN}}$	3.36E−14	8.65E−14	9.25E−15$^{\text{PW}}$
f_{13}	20.08$^{\text{WN}}$	**4.827**$^{\text{CWN}}$	21.46	22.27
f_{14}	**1.31E−16**$^{\text{PWN}}$	4.22E−16$^{\text{WN}}$	3.35E−15$^{\text{N}}$	9.34E−15
f_{15}	**8.92E−196**$^{\text{PWN}}$	1.2E−184$^{\text{WN}}$	6.61E−172$^{\text{N}}$	2.49E−131
f_{16}	**0.1632**$^{\text{PWN}}$	0.2915$^{\text{W}}$	1.457	0.2347
f_{17}	**2.44E−44**$^{\text{WN}}$	8.3E−131$^{\text{CWN}}$	4.90E−43$^{\text{N}}$	1.66E−12
f_{18}	14.51$^{\text{W}}$	**5.61**$^{\text{CWN}}$	36.32	9.21$^{\text{CW}}$

This test was done with 20 particles and 200,000 FEs. The values are ABSQ. Superscripts show that the significance level (p value) of Wilcoxon's test between the two methods was less than 0.05 ($p < 0.05$). The best result among all three methods has been bolded

Several variants of PSO (Shi and Eberhart 1998a,b; Eberhart and Shi 2001; Clerc and Kennedy 2002; van den Bergh and Engelbrecht 2002; Zheng et al. 2003; Pso 2006; Nickabadi et al. 2011) were tested when they used random matrices (*Rndm*) or the proposed rotation matrices (*Rotm*(σ) with σ taken from Fig. 6) in their velocity updating rule (Eq. 11). Note that in each test the only change incorporated into each method was the replacement of the *Rndm* with *Rotm* while all other parameters (acceleration coefficients, inertia weight, population size, etc) have been left untouched.

Table 4 shows the results of applying StdPSO2006 to 18 benchmark problems (introduced in Table 2) in 10, 30, and 60 dimensions. The best results reported in Table 4 have been bolded. Those which were significantly better based on the Wilcoxon's test ($p < 0.05$) have also been marked by an asterisk. The StdPSO2006-*Rotm* performs significantly better in 14 out of 18 10-dimensional problems. Moreover, the algorithms perform almost the same for one problem (f_9) and, for the 3 remaining problems (f_4, f_{13}, and f_{18}), StdPSO2006 performs better. On the other hand, the performance of StdPSO2006-*Rotm* is significantly better than StdPSO2006 in *all* functions of 30 and

Table 4 Results of applying StdPSO2006 and StdPSO2006-*Rotm* to 18 benchmarks with a different number of dimensions

Function	10 Dimensions		30 Dimensions		60 Dimensions	
	Rndm	*Rotm*	*Rndm*	*Rotm*	*Rndm*	*Rotm*
Non-rotated functions						
f_1	4.68E+00	**2.99E+00***	3.39E+01	**2.82E+01***	8.99E+02	**6.00E+01***
f_2	7.174E−14	**0***	2.46E−02	**2.07E−06***	2.01E+01	**1.85E−02***
f_3	1.426E−06	**1.39E−09***	6.51E−01	**5.56E−03***	9.82E+00	**4.13E−01***
f_4	**3.82E−03***	2.5E−01	9.47E+00	**7.75E+00***	5.48E+01	**2.74E+01***
f_5	3.42E−13	**4.63E−18***	9.57E−02	**6.39E−06***	6.26E+01	**5.99E−02***
f_6	2.02E−11	**1.55E−17***	9.57E+00	**7.06E−04***	6.89E+03	**6.50E+00***
f_7	8.16E−14	**0***	7.91E−02	**1.38E−02***	2.37E+01	**2.82E−01***
f_8	3.15E+00	**7.20E−07***	2.05E+04	**5.51E+02***	1.56E+05	**1.17E+04***
f_9	**1.14E−04**	9.15E−4	2.90E+03	**1.31E−02***	2.85E+07	**5.14E+01***
Average	8.71E−01	**3.60E−01**	2.60E+03	**6.52E+01**	1.65E+06	**1.27E+03**
Rotated functions						
f_{10}	4.35E+00	**3.30E+00***	3.20E+01	**2.82E+01***	7.57E+02	**6.15E+01***
f_{11}	4.40E−14	**0***	2.72E−02	**2.20E−06***	2.05E+01	**1.99E−02***
f_{12}	1.46E−06	**1.35E−09***	6.33E−01	**5.83E−03***	1.01E+01	**4.25E−01***
f_{13}	**1.63E−02***	2.7E−01	1.83E+01	**7.26E+00***	6.52E+01	**2.77E+01***
f_{14}	1.87E−13	**2.31E−18***	1.05E−01	**7.65E−06***	6.68E+01	**6.88E−02***
f_{15}	2.32E−11	**2.13E−17***	1.31E+01	**7.26E−04***	7.59E+03	**6.68E+00***
f_{16}	1.25E−13	**0***	7.77E−02	**3.04E−02***	2.35E+01	**3.63E−01***
f_{17}	6.3E−01	**1.64E−06***	1.84E+04	**5.55E+02***	1.64E+05	**1.12E+04***
f_{18}	**4.96E−07***	1.60E−3	6.18E+01	**1.36E−02***	1.46E+07	**5.61E+01***
Average	5.55E−01	**3.98E−01**	2.06E+03	**6.56E+01**	3.18E+06	**1.32E+03**

The columns specified by *Rndm* are the results when this algorithm uses random matrices while *Rotm* are these results when this algorithm uses rotation matrices. The values are ABSQ. A Star shows the significance level (p value) of Wilcoxon's test was less than 0.05 ($p < 0.05$)

60 dimensions. This shows that the proposed rotation matrices are more effective for the problems with a larger number of dimensions.

Additionally, the average of solutions (average row in Table 4) is quite different for the rotated and non-rotated functions when the StdPSO2006 is used (36, 20 and 48 % differences for 10, 30, and 60 dimensional problems). However, these values are very close to each other when the StdPSO2006-*Rotm* is used (9, 0.6, and 3.7 % for 10, 30, and 60 dimensional problems, respectively). In fact, rotating the functions has a small effect on the performance of the StdPSO2006-*Rotm* whereas this effect is considerable when the StdPSO2006 is used. Finally, the averages of solutions for the StdPSO2006-*Rotm* over 10, 30, and 60 dimensional non-rotated (rotated) functions are 58 % (28 %), 97 % (96 %), and 99 % (99 %) better than that of the StdPSO2006, respectively. The optimization curves for some of these functions using StdPSO2006 and StdPSO2006-*Rotm* are available in "Appendix II".

Table 5 Results of applying CoPSO and CoPSO-*Rotm* to 18 benchmarks with different number of dimensions

Function	10 Dimensions		30 Dimensions		60 Dimensions	
	Rndm	*Rotm*	*Rndm*	*Rotm*	*Rndm*	*Rotm*
Non-rotated functions						
f_1	3.52E+00	**1.10E+00***	3.21E+01	**2.76E+01***	1.70E+02	**6.48E+01***
f_2	**0**	1.11E−16	3.83E−03	**2.08E−08***	1.65E+00	**2.22E−03***
f_3	1.91E−10	**2.33E−11***	4.94E−02	**5.55E−04***	3.27E+00	**1.41E−01***
f_4	**2.07E−01***	3.52E+00	**7.75E+00***	2.13E+01	**3.15E+01***	5.26E+01
f_5	**2.66E−17***	7.98E−17	3.84E−04	**8.95E−08***	4.31E+00	**7.27E−03***
f_6	3.32E−19	**3.69E−21***	4.55E−02	**7.45E−06***	5.25E+02	**7.74E−01***
f_7	**0**	4.44E−16	7.48E−01	**3.62E−01***	9.61E+00	**5.21E+00***
f_8	4.28E−06	**2.47E−10***	1.59E+03	**6.74E+01***	2.94E+04	**6.87E+03***
f_9	**1.47E−03***	5.18E−03	**3.54E+00***	2.18E+01	5.46E+03	**1.19E+02***
Average	**4.14E−01**	5.14E−01	1.81E+02	**1.54E+01**	3.96E+03	**7.91E+02**
Rotated functions						
f_{10}	3.03E+00	**1.07E+00***	2.92E+01	**2.83E+01***	1.47E+02	**6.48E+01***
f_{11}	**0**	5.55E−17	3.95E−04	**2.32E−08***	1.48E+00	**2.09E−03***
f_{12}	1.91E−10	**2.46E−11***	5.86E−02	**5.79E−04***	3.11E+00	**1.34E−01***
f_{13}	**1.96E+00***	3.32E+00	**1.76E+01***	2.04E+01	5.25E+01	**5.21E+01**
f_{14}	**3.70E−17***	8.67E−17	1.01E−03	**8.79E−08***	5.67E+00	**6.84E−03***
f_{15}	8.94E−19	**5.66E−21***	7.82E−02	**5.95E−06***	5.74E+02	**7.33E−01***
f_{16}	**0.00E+00***	8.28E−03	**2.31E−01**	5.49E−01	1.09E+01	**4.24E+00***
f_{17}	1.06E−06	**3.93E−10***	1.09E+03	**7.02E+01***	5.56E+04	**6.97E+03***
f_{18}	**1.12E−03**	1.03E−02	**1.76E+01**	2.04E+01	7.10E+03	**1.19E+02***
Average	5.55E−01	**4.90E−01**	1.28E+02	**1.55E+01**	7.05E+03	**8.01E+02**

The columns specified by *Rndm* are the results when this algorithm uses random matrices while *Rotm* are these results when this algorithm uses rotation matrices. The values are ABSQ. A Star shows that the p value of Wilcoxon's test was less than 0.05 ($p < 0.05$)

Table 5 reports the results of CoPSO when it uses random matrices versus rotation matrices in its velocity updating rule.[13] It is clear that CoPSO-*Rotm* performs significantly better than CoPSO for 8 out of 18 functions in 10 dimensional space. Also, for four functions (f_2, f_7, f_{11}, and f_{18}), the rotation matrices have no significant effect on CoPSO and both algorithms work almost the same as one another. CoPSO-*Rotm* performs worse than the CoPSO in terms of ABSQ in six functions. When it comes to 30 dimensional problems, the CoPSO-*Rotm* wins the challenge on 13 functions, and only loses the race on three of them.

Finally, in 60 dimensional problems, the effect of the proposed rotation matrix is more substantial and it makes CoPSO-*Rotm* significantly better than CoPSO in 16

[13] Note that the maximum number of FE in this test (Table 5) is 10,000 which is different from the one in Table 3, 200,000. Thus, the difference between the results in these two tables is expected.

Table 6 Results of applying AIWPSO and AIWPSO-*Rotm* to 18 benchmarks with different number of dimensions

Function	10 Dimensions		30 Dimensions		60 Dimensions	
	Rndm	*Rotm*	*Rndm*	*Rotm*	*Rndm*	*Rotm*
Non-rotated functions						
f_1	9.56E−01	**1.12E−01***	1.45E+01	**8.11E+00***	4.55E+01	**2.04E+01***
f_2	8.14E−16	**9.25E−17***	1.29E−03	**5.60E−14***	1.16E−01	**1.17E−06***
f_3	1.20E−14	**6.59E−15***	1.11E−01	**7.46E−07***	6.57E−01	**3.56E−03***
f_4	**5.42E−01***	7.57E−01	**3.56E+00***	5.14E+00	**8.45E+00***	1.06E+01
f_5	1.05E−16	**6.94E−17***	4.12E−03	**2.01E−13***	2.61E−01	**5.51E−06***
f_6	9.75E−33	**1.44E−39***	3.85E−01	**1.92E−11***	3.21E+01	**5.08E−04***
f_7	1.20E−14	**1.63E−15***	**1.97E−02***	3.34E−02	**1.54E−01***	3.56E−01
f_8	5.06E−02	**3.61E−15***	1.34E+03	**1.26E+01***	1.36E+04	**2.44E+03***
f_9	2.53E−02	**6.99E−03***	**9.62E−01***	9.93E−01	1.33E+01	**9.92E+00***
Average	1.75E−01	**9.74E−02**	1.51E+02	**2.99E+00**	1.53E+03	**2.76E+02**
Rotated functions						
f_{10}	2.42E+00	**2.13E+00***	**2.59E+01***	2.78E+01	5.98E+01	**5.81E+01***
f_{11}	1.55E−15	**5.55E−17***	8.02E−04	**7.47E−14***	9.91E−02	**1.88E−06***
f_{12}	1.07E−14	**7.66E−15***	1.01E−01	**9.41E−07***	7.78E−01	**3.39E−03***
f_{13}	1.09E+00	**6.02E−01***	5.82E+00	**5.19E+00***	1.45E+01	**1.34E+01**
f_{14}	4.29E−16	**6.48E−17**	1.76E−03	**3.03E−13***	2.53E−01	**6.87E−06***
f_{15}	1.24E−34	**2.03E−39***	1.23E−01	**1.60E−11***	4.80E+01	**5.58E−04***
f_{16}	6.22E−15	**5.92E−16***	**4.59E−02***	7.86E−02	1.36E−01	**1.19E−01***
f_{17}	3.14E−06	**7.88E−17***	7.53E+01	**4.48E+00***	2.89E+03	**1.62E+03**
f_{18}	4.77E−02	**1.12E−02***	3.80E+00	**1.45E+00***	**1.64E+01**	1.99E+01
Average	3.94E−01	**3.04E−01**	1.23E+01	**4.33E+00**	3.37E+02	**1.91E+02**

The columns specified by *Rndm* are the results when this algorithm uses random matrices while *Rotm* are these results when this algorithm uses rotation matrices

functions and almost the same as CoPSO for the two remaining functions. In addition, the table demonstrates that the performance of CoPSO-*Rotm* is not greatly affected when the functions are rotated (4.6, 0.6, and 1.2 % for 10, 30, and 60 dimension problems, respectively). However, this is not the case for CoPSO and the performance of the algorithm is significantly affected when the functions are rotated (25.4, 29.2, and 43.8 % worse for 10, 30, and 60 dimensional problems). Although the average of solutions found by CoPSO-*Rotm* for all 10 dimensional problems was 3 % worse than that of CoPSO, it was 90 % better than that of CoPSO in the 30 dimensional problems and 85 % better for the 60 dimensional problems, thereby showing that the proposed random rotation has improved the overall performance of CoPSO. The convergence curves for some of these functions using CoPSO and CoPSO-*Rotm* are available in "Appendix II".

Table 6 shows the results of applying the proposed random rotations (*Rotm*) to AIWPSO. It is clear that the proposed rotation matrix has improved the performance

of the method for 16 functions in 10 dimensional problems, but only in one case the performance was reduced. Also, in the 30 and 60 dimensional functions, the performance of the algorithm is significantly better for 13 functions. In AIWPSO-*Rotm*, the average of solutions has been less affected when the function's space is rotated (43 % affected for 10, 30, and 60 dimensional problems on average in comparison to 75 % for AIWPSO). The average improvement over all the 10, 30, and 60 dimensional problems was 29.4, 95.5 and 75 %, respectively. The convergence curves for some of these functions using AIWPSO and AIWPSO-*Rotm* are available in "Appendix II".

In Decreasing-IW method the proposed rotation matrix has improved the performance of the algorithm by 17, 89.5 and 99 % for 10, 30, and 60 dimensional problems, respectively (Table 7). In Increasing-IW (Stochastic-IW) method, although the proposed rotation causes 11 % (7.7 %) impairment in 10 dimensional problems, it offers 92.8 % (90.6 %) and 98.8 % (92 %) improvement for the 30 and 60 dimensional problems, respectively. In GCPSO, the percentage of improvement in 10, 30, and 60 dimensional problems were 14, 10, and 36 %, respectively.

Results indicate that the proposed Euclidean rotation can improve the performance of different PSO methods. Figure 8 shows the convergence curve of several PSO variants when they use *Rndm* or *Rotm* to optimize the function f_1 in 10D and 60D.

Figure 8 shows that if *Rndm* is used in the algorithms, the improvement of the objective value slows down towards the end of the optimization. However, this is not the case if *Rotm* is used in the same algorithm in the tested cases. It seems that towards the end of the optimization the swarm does not have sufficient diversity to exploit better solutions. However, it seems that when *Rotm* is used, the swarm is still diverse enough to exploit better solutions. This was actually expected as towards the end of the optimization, the particles are closer to each other and the velocity vectors become smaller. Thus, the probability of having the *PI* and *SI* closer to one of the axes is higher, which, based on the second identified issue discussed in Sect. 3, results in ineffective exploitation and causes impairment in the performance of the algorithm. However, when *Rotm* is used, this issue does not exist and particles are still diverse enough to find better solutions.

In the last test, the proposed adaptive approach for σ as well as the ability of the algorithm to deal with the problems with shifted optima is examined (the optimal point is not in the center of the coordinate). In these test functions, the locations of the optimal solutions of all test functions have been shifted by a vector o (see Sect. 5.1). Two PSO variants (StdPSO 2006, AIWPSO) were selected to run on these benchmarks when they use *Rndm* (their original version), *Rotm* with a static approach for σ, or *Rotm* with the adaptive approach. They were applied to each test case 100 times and their results are reported. Also, each algorithm was run for 10,000 FEs with 20 particles. The listed problems are in 10-dimensional space. Table 8 shows the results.

The results presented in Table refTab8 indicate that the proposed adaptive approach has improved the ability of the StdPSO2006-*Rotm* (compared to the static approach) in 13 functions out of 18 (in 7 functions the improvement was significant). This improvement takes place in 10 functions for the AIWPSO-*Rotm* (in four functions the improvements were significant). The results reported in the table also indicate that the proposed *Rotm* approach has improved the performance of the methods for dealing with the problems in which the location of global optima is not located in the center

Table 7 Results of comparing three algorithms decreasing-IW, increasing-IW, stochastic-IW, and GCPSO when they use random or rotation matrices

Report type	Number of dimensions	Space rotation*	Decreasing-IW	Decreasing-IW-*Rotm*	Increasing-IW	Increasing-IW-*Rotm*	Stochastic-IW	Stochastic-IW-*Rotm*	GCPSO	GCPSO-*Rotm*
Averages of ABSQs over all functions	10	N	6.39E−01	**6.30E−01**	**6.97E−01**	9.67E−01	**4.91E−01**	5.50E−01	4.84E−1	**4.31E−1**
		Y	8.86E−01	**6.26E−01**	9.77E−01	**9.26E−01**	7.01E−01	**5.50E−01**	6.58E−1	**5.50E−1**
	30	N	1.01E+03	**9.58E+01**	1.07E+03	**6.95E+01**	2.88E+02	**2.70E+01**	1.81E+2	**1.54E+2**
		Y	7.58E+02	**8.96E+01**	8.09E+02	**6.76E+01**	2.75E+02	**2.58E+01**	1.22E+2	**1.18E+2**
	60	N	2.75E+05	**1.61E+03**	1.13E+04	**8.01E+02**	1.13E+04	**8.01E+02**	3.55E+3	**3.22E+3**
		Y	7.05E+04	**1.60E+03**	1.36E+05	**9.80E+02**	8.67E+03	**7.81E+02**	8.42E+3	**4.49E+3**
Significantly better ABSQ	10	N	2	**7**	3	**6**	2	**6**	0	**3**
		Y	2	**7**	3	**6**	1	**6**	0	**3**
	30	N	2	**7**	2	**7**	2	**5**	0	**3**
		Y	3	**6**	2	**6**	2	**6**	1	**3**
	60	N	1	**7**	2	**7**	2	**7**	1	**5**
		Y	0	**9**	0	**8**	1	**7**	1	**3**

The results are the averages over 9 functions in each group (10, 30, and 60 dimensional non-rotated, and 10, 30, and 60 dimensional rotated). In the column "Space rotation", 'N' means non-rotated and 'Y' means rotated function

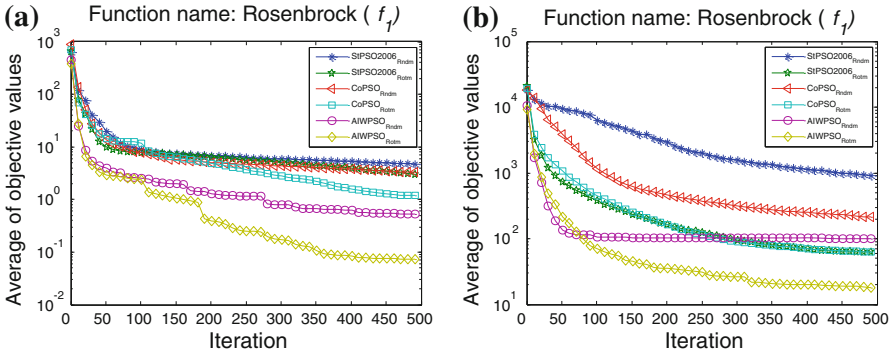


Fig. 8 Convergence curve for several PSO-based methods when they used *Rotm* or Rnmd and they are applied to **a** Rosenbrock (f_1) 10D, **b** Rosenbrock (f_1) 60D. The convergence curve for some other functions can be found in "Appendix II"

Table 8 Comparative results between StdPSO2006-*Rotm* and AIWPSO-*Rotm*, when they use the adaptive approach or static value for σ

Function name	StdPSO2006-*Rndm*	**StdPSO 2006-*Rotm***		AIWPSO-*Rndm*	**AIWPSO-*Rotm***	
		Static σ	Adaptive σ		Static σ	Adaptive σ
f_1	5.73E+00	**2.21E+00**^{RA}	**3.95E+00**^{R}	8.99E−01	1.27E−01^{R}	**1.07E−01**^{R}
f_2	6.194E−12	**0**^{R}	**0**^{R}	9.19E−16	3.41E−17^{R}	**5.36E−19**^{RS}
f_3	1.521E−05	8.99E−09^{R}	**1.03E−09**^{RS}	5.33E−14	**3.18E−15**^{R}	4.19E−15^{R}
f_4	**9.12E−03**^{SA}	1.15E−01	2.15E−01	**2.71E−01**^{S}	7.52E−01	3.52E−01^{S}
f_5	5.33E−13	**5.13E−18**^{R}	6.19E−18^{R}	2.19E−16	5.31E−17^{R}	**4.72E−17**^{R}
f_6	7.12E−11	3.51E−17^{R}	**8.15E−18**^{RS}	7.75E−34	5.92E−40^{R}	**5.55E−43**^{RS}
f_7	9.83E−14	**0**^{R}	**0**^{R}	3.49E−14	**2.57E−15**^{RA}	9.42E−15^{R}
f_8	6.19E+00	5.29E−07^{R}	**1.01E−07**^{RS}	4.97E−02	2.58E−15^{R}	**1.94E−15**^{R}
f_9	1.14E−04	3.86E−4	**1.87E−5**^{RS}	3.28E−02	**7.01E−03**^{R}	7.08E−03^{R}
f_{10}	4.39E+00	3.10E+00^{R}	**2.05E+00**^{RS}	3.51E+00	**1.19E+00**^{R}	1.35E+00^{R}
f_{11}	3.22E−14	**0**^{R}	**0**^{R}	2.75E−15	4.39E−17^{R}	**4.37E−18**^{RS}
f_{12}	5.49E−06	2.18E−09^{R}	**8.77E−10**^{RS}	1.91E−14	4.72E−15^{R}	**2.99E−15**^{R}
f_{13}	**1.71E−02**^{SA}	3.8E−01	3.77E−01	1.33E+00	5.76E−01^{R}	**1.39E−02**^{RS}
f_{14}	3.59E−13	**1.33E−18**^{R}	2.13E−18^{R}	1.69E−16	5.39E−17	**1.28E−17**^{R}
f_{15}	1.72E−11	1.17E−17^{R}	**1.01E−17**^{R}	2.98E−34	1.33E−39^{R}	**2.37E−39**^{R}
f_{16}	3.55E−13	**0**^{R}	**0**^{R}	5.04E−15	**2.67E−16**^{RA}	7.97E−16^{R}
f_{17}	5.29E−01	2.94E−06^{R}	**5.95E−07**^{RS}	7.19E−06	**3.21E−17**^{R}	4.01E−17^{R}
f_{18}	**5.26E−07**^{SA}	2.95E−3	3.15E−4^{R}	5.63E−02	**2.22E−02**^{R}	3.53E−02^{R}

The values are the ABSQ. A Star shows that the significance level (p value) of Wilcoxon's test between static and adaptive approach was less than 0.05 ($p < 0.05$). Also, the superscript s shows that the significance level between *Rndm* and *Rotm* with static σ is less than 0.05

of the coordinate. In fact, the StdPSO2006-*Rotm* with static σ performed significantly better than its original version in 14 functions over all 18 listed when the optimum solution is not in the center of the coordinates. Also, the AIWPSO-*Rotm* performed significantly better (on average) than their original versions in optimizing 16 functions over all 18 functions.

6 Conclusion and future work

The velocity updating rule of PSO (with the use of randomly generated diagonal matrices) can drastically impact the direction and length of generated vectors. This impact can be different for different vectors and, despite calculability, it is not controllable. Additional side-effects of multiplying personal and global best vectors by two random diagonal matrices were identified in this paper. These are: delay in convergence and attraction to inappropriate directions, dissimilar direction changes in different vectors, and limitation in particles' movement in the search space. In order to overcome these problems, the usage of Euclidean rotation matrices with random direction (generated by a normal distribution) rather than random diagonal matrices was suggested. According to the presented results, the proposed approach addressed these problems effectively. Also, as the Euclidean rotation preserves the length of the vectors and only affects their directions, it makes it easier to control the effects of randomness applied by the random matrices on the direction of the vectors. The parameters of the normal distribution (i.e. mean and variance σ) for the rotation direction were analyzed in detail through some experiments. A tuning procedure and an adaptive approach (Eiben et al. 1999) were considered for setting the value of the parameter σ. In the empirical analysis presented in this paper, the new approach was adjoined to seven variants of PSO and the performance was put under evaluation using 18 standard benchmark functions. Experiments showed that the proposed approach significantly improved the performance of these PSO variants. Although designing a rotation invariant PSO,(which is one of the challenges involved in the algorithm (Hansen et al. 2008)) was not the aim of this study, experiments suggested that the proposed approach helps the algorithm to be rotation invariant, as theoretically investigated in Wilke (2005). The presented approach (usage of random rotation matrices) is just one of many possible approaches that can be used for dealing with the identified problems. As an example, one possibility would be to perform all calculations for optimization in the hyper-spherical space (rather than Cartesian space) thereby making rotation calculations much easier. Also, the proposed rotation approach involved rotating the vectors in all possible planes, which is computationally expensive. An alternative approach was presented in this paper (called *RotmP*), in which only one of the planes was selected randomly and rotation was applied only to that plane. Although results showed that rotation in all possible planes is slightly beneficial, there is still a tradeoff between computational expenses and the quality of solutions. Thus, it is worthwhile to investigate *RotmP* in more detail and to apply that to large scale optimization problems. According to the experiments presented in this paper, it seems that one of the PSO variants (called AIWPSO) gains more benefit from the usage of rotation matrices. The reason behind this phenomenon is not clear; it might be related to the interactions of different com-

ponents or parameters of the algorithm. Hence, it is worthwhile to investigate this effect on different methods and to provide reasons as to why some methods are more and some are less compatible with rotation matrices. Moreover, it would be of great interest to evaluate the proposed approach in dynamic environments. We plan to apply the proposed velocity update equation to a set of dynamic test functions in a systematic way to understand the merits of the proposed approach in such environments.

Acknowledgments The authors would like to extend their great appreciation to the associate editor and anonymous reviewers for constructive comments that have helped us to improve the quality of the paper. Also, the authors extend thanks to Dr. Frank Neumann, Dr. Andrew Sutton, and Dr. Michael Kirley who provided us with excellent comments. This work was partially funded by the ARC Discovery Grants DP0985723, DP1096053, and DP130104395, as well as by Grant N N519 5788038 from the Polish Ministry of Science and Higher Education (MNiSW).

7 Appendix I

Appendix I provides the formula for all used benchmark functions. The variable *pd* represents the number of dimensions.

f_1: Rosenbrock's function

$$f(x) = \sum_{i=1}^{pd-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$$

f_2: Rastrigin's function

$$f(x) = \sum_{i=1}^{pd} (x_i^2 - 10\cos(2\pi x_i) + 10)$$

f_3: Ackley's function

$$f(x) = 20 + e - 20e^{-0.2\sqrt{\frac{\sum_{i=1}^{pd} x_i^2}{pd}}} - e^{\frac{\sum_{i=1}^{pd} \cos(2\pi x_i)}{pd}}$$

f_4: Weierstrass function

$$f(x) = \sum_{i=1}^{pd} (\sum_{k=0}^{k\max} [a^k \cos(2\pi b^k (x_i + 0.5))])$$
$$-pd \sum_{k=0}^{k\max} [a^k \cos(2\pi b^k 0.5)],$$
$$where_a = 0.5, b = 0.3, k\max = 20$$

f_5: Griewank function

$$f(x) = 1 + \frac{\sum_{i=1}^{pd} (x_i - 100)^2}{4000} - \prod_{i=1}^{pd} \cos(\frac{x_i - 100}{\sqrt{i}})$$

f_6: Sphere function

$$f(x) = \sum_{i=1}^{pd} x_i^2$$

f_7: Non-continuous Rastrigin's function

$$f(x) = \sum_{i=1}^{pd} (y_i^2 - 10\cos(2\pi y_i) + 10)$$

$$y_i = \begin{cases} x_i & |x_i| < \frac{1}{2} \\ \frac{round(2x_i)}{2} & |x_i| \geq \frac{1}{2} \end{cases}, \quad i = 1, 2, \ldots, pd$$

f_8: Quadratic function

$$f(x) = \sum_{i=1}^{pd} \left(\sum_{j=1}^{i} x_j \right)^2$$

f_9: Generalized penalized function

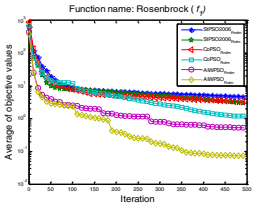
$$f(x) = 0.1 \left\{ \begin{array}{l} \sin^2(3\pi x_1) + \sum_{i=1}^{pd-1} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] + \\ (x_{pd} - 1)^2 [1 + \sin^2(2\pi x_{pd})] \end{array} \right\} + \sum_{i=1}^{pd} u(x_i, 5, 100, 4)$$

where

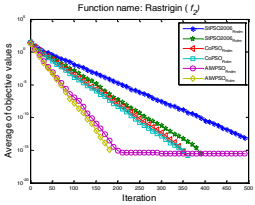
$$u(x_i a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \leq x_i \leq a \\ k(-x_i - a)^m & x_i < a \end{cases}$$

8 Appendix II

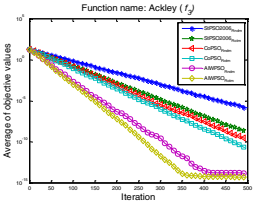
Appendix II displays optimization curves for some of the benchmark functions (f_1, f_2, f_3, f_5, f_6, f_8) using theStdPSO2006, theStdPSO2006-*Rotm*,the CoPSO, the CoPSO-*Rotm*, the AIWPSO, and the AIWPSO-*Rotm* in 10, 30, and 60 dimensional spaces.



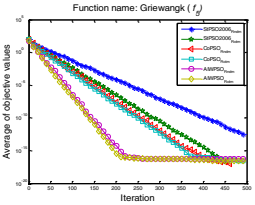
10D Rosenbrock



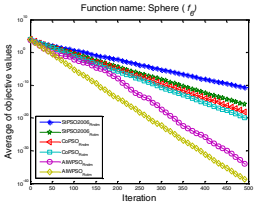
10D Rastrigin



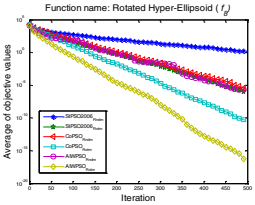
10D Ackely



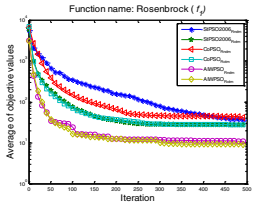
10D Griewank



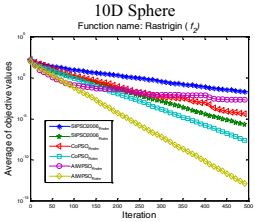
10D Sphere



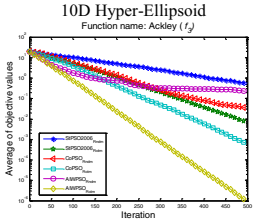
10D Hyper-Ellipsoid



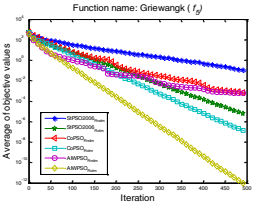
30D Rosenbrock



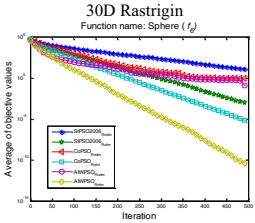
30D Rastrigin



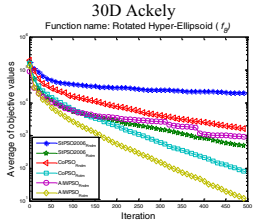
30D Ackely



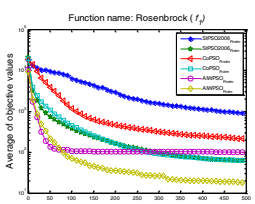
30D Griewank



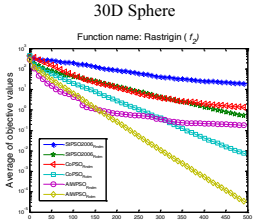
30D Sphere



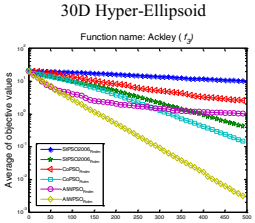
30D Hyper-Ellipsoid



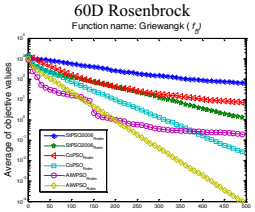
60D Rosenbrock



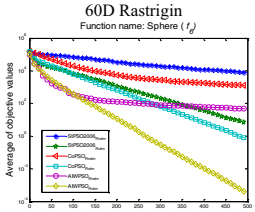
60D Rastrigin



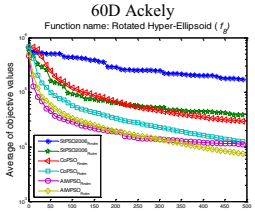
60D Ackely



60D Griewank



60D Sphere



60D Hyper-Ellipsoid

9 Appendix III

Appendix III presents the implementation details for calculating $Rotm(\sigma)$ as given in Eq. (10). Assume we need to multiply d-dimensional vector v by $Rotm(\sigma)$. According to Eq. (10), we have:

$$v \times Rotm(\sigma) = v \times Rot_{1,2}\left(\alpha_{1,2}\right) \times Rot_{1,3}\left(\alpha_{1,3}\right) \times \cdots \times Rot_{d-1,d}\left(\alpha_{d-1,d}\right)$$

The number of matrices on the right side of the equation is $d(d-1)/2$. However, according to Eq. (10), each matrix $Rot_{i,j}\left(\alpha_{i,j}\right)$ only contains 4 non-zero values that are in (i,i), (i,j), (j,i), and (j,j) positions of Rot. Consequently, multiplying v by Rot can be performed as follows:

$$v_k \times Rot_{i,j}\left(\alpha_{i,j}\right) = \begin{cases} v_k Rot_{i,i}\left(\alpha_{i,i}\right) + v_j Rot_{j,i}\left(\alpha_{j,i}\right) & if k = i \\ v_k Rot_{j,j}\left(\alpha_{j,j}\right) + v_i Rot_{i,j}\left(\alpha_{j,i}\right) & if k = j \\ v_k & otherwise \end{cases}$$

Clearly, multiplication of v_k and $Rot_{i,j}\left(\alpha_{i,j}\right)$ can be done in O(1). Also, this multiplication alters only two indexes in v. This formulation can be repeatedly applied to v for different matrices $Rot_{j,i}\left(\alpha_{j,i}\right)$. Because the number of these matrices is $d(d-1)/2$ and the multiplication is in O(1), this multiplication is done in $O(d^2)$.

References

Chen, D.B., Zhao, C.X.: Particle swarm optimization with adaptive population size and its application. Appl. Soft Comput. **9**(1), 39–48 (2009)

Chow, T.L.: Mathematical Methods for Physicists: A Concise Introduction. Cambridge University Press, Cambridge (2000)

Clerc, M.: (2006) Particle Swarm Optimization. Wiley, New York

Clerc, M., Kennedy, J.: The particle swarm—explosion, stability, and convergence in a multidimensional complex space. IEEE Trans. Evol. Comput. **6**(1), 58–73 (2002)

de Oca, M.A.M., Stutzle, T., Birattari, M., Dorigo, M.: Frankenstein's PSO: a composite particle swarm optimization algorithm. IEEE Trans. Evol. Comput. **13**(5), 1120–1132 (2009)

Duffin, K.L., Barrett, W.A.: Spiders: a new user interface for rotation and visualization of n-dimensional point sets. In: Proceedings of the Conference on Visualization, IEEE Computer Society Press, Los Alamitos, CA (1994)

Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: International Symposium on Micro Machine and Human Science, IEEE (1995)

Eberhart, R.C., Shi, Y.: Tracking and optimizing dynamic systems with particle swarms. In: Proceedings IEEE Congress Evolutionary Computation, IEEE (2001)

Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. **3**(2), 124–141 (1999)

Engelbrecht, A.P.: Fundamentals of Computational Swarm Intelligence. Wiley, Hoboken (2005)

Ghosh, S., Das, S., Kundu, D., Suresh, K., Panigrahi, B.K., Cui, Z.: An inertia-adaptive particle swarm system with particle mobility factor for improved global optimization. Neural Comput. Appl. **21**(2), 237–250 (2010)

Hansen, N., Ros, R., Mauny, N., Schoenausr, M, Auger, A.: PSO facing non-separable and ill-conditioned problems. In: INRIA (2008)

Helwig, S., Wanka, R.: Particle swarm optimization in high-dimensional bounded search spaces. In: Swarm Intelligence Symposium, IEEE (2007)

Helwig, S., Branke, J., Mostaghim, S.: Experimental analysis of bound handling techniques in particle swarm optimization. IEEE Trans. Evol. Comput. **17**(2), 259–271 (2013)
Hsieh, S.T., Sun, T.Y., Liu, C.C., Tsai, S.J.: Efficient population utilization strategy for particle swarm optimizer. IEEE Trans. Syst. Man Cybern. Part B-Cybern. **39**(2), 444–456 (2009)
Huang, H., Qin, H., Hao, Z., Lim, A.: Example-based learning particle swarm optimization for continuous optimization. Inf. Sci. (2010)
Jiang, M., Luo, Y.P., Yang, S.Y.: Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. Inf. Process. Lett. **102**(1), 8–16 (2007)
Kennedy, J., Eberhart, R.: Particle swarm optimization. In: International Conference on Neural Networks, IEEE (1995)
Krohling, R.A.: Gaussian Swarm: A Novel Particle Swarm Optimization Algorithm. IEEE (2004)
Liang, J.J., Qin, A.K., Suganthan, P.N., Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans. Evol. Comput. **10**(3), 281–295 (2006)
Mendes, R., Kennedy, J., Neves, J.: The fully informed particle swarm: simpler, maybe better. IEEE Trans. Evol. Comput. **8**(3), 204–210 (2004)
Nickabadi, A., Ebadzadeh, M.M., Safabakhsh, R.: A novel particle swarm optimization algorithm with adaptive inertia weight. Appl. Soft Comput. **11**(4), 3658–3670 (2011)
Poli, R.: Analysis of the publications on the applications of particle swarm optimisation. J. Artif. Evol. Appl. 1–10 (2008)
Poli, R.: Mean and variance of the sampling distribution of particle swarm optimizers during stagnation. IEEE Trans. Evol. Comput. **13**(4), 712–721 (2009)
Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization an overview. Swarm Intell. **1**(1), 33–57 (2007)
Pso, I.: PSO Source Code. http://particleswarm.info/Standard_PSO_2006.c (2006)
Ratnaweera, A., Halgamuge, S.K., Watson, H.C.: Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Trans. Evol. Comput. **8**(3), 240–255 (2004)
Salomon, R.: Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions—a survey of some theoretical and practical aspects of genetic algorithms. BioSystems **39**, 263–278 (1995)
Secrest, B.R., Lamont, G.B.: Visualizing particle swarm optimization-Gaussian particle swarm optimization. In: Swarm Intelligence Symposium, IEEE (2003)
Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: World Congress on Computational Intelligence, IEEE (1998a)
Shi, Y., Eberhart, R.: Parameter Selection in Particle Swarm Optimization. Evolutionary Programming VII. Springer, Berlin (1998b)
Spears, W.M., Green, D.T., Spears, D.F.: Biases in particle swarm optimization. Int. J. Swarm Intell. Res. **1**(2), 34–57 (2010)
Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y., Auger, A., Tiwari, S.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. In: KanGAL Report (2005)
Trelea, I.C.: The particle swarm optimization algorithm: convergence analysis and parameter selection. Inf. Process. Lett. **85**(6), 317–325 (2003)
Tu, Z., Lu, Y.: A robust stochastic genetic algorithm (StGA) for global numerical optimization. IEEE Trans. Evol. Comput. **8**(5), 456–470 (2004)
van den Bergh, F., Engelbrecht, A.: A new locally convergent particle swarm optimiser. In: Systems, Man and Cybernetics, Hammamet, Tunisia, IEEE (2002)
Van den Bergh, F., Engelbrecht, A.P.: A study of particle swarm optimization particle trajectories. Inf. Sci. **176**(8), 937–971 (2006)
Van den Bergh, F., Engelbrecht, A.P.: A convergence proof for the particle swarm optimiser. Fundamenta Informaticae **105**(4), 341–374 (2010)
Wang, Y., Li, B., Weise, T., Wang, J., Yuan, B., Tian, Q.: Self-adaptive learning based particle swarm optimization. Inf. Sci. **181**(20), 4515–4538 (2011)
Wilcoxon, F.: Individual comparisons by ranking methods. Biometr. Bull. **1**(6), 80–83 (1945)
Wilke, D.: Analysis of the Particle Swarm Optimization Algorithm. Master Thesis, University of Pretoria (2005)
Wilke, D.N., Kok, S., Groenwold, A.A.: Comparison of linear and classical velocity update rules in particle swarm optimization: notes on diversity. Int. J. Numer. Methods Eng. **70**(8), 962–984 (2007a)

Wilke, D.N., Kok, S., Groenwold, A.A.: Comparison of linear and classical velocity update rules in particle swarm optimization: notes on scale and frame invariance. Int. J. Numer. Methods Eng. **70**(8), 985–1008 (2007b)

Xinchao, Z.: A perturbed particle swarm algorithm for numerical optimization. Appl. Soft Comput. **10**(1), 119–124 (2010)

Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. IEEE Trans. Evol. Comput. **3**(2), 82–102 (1999)

Zhang, L.-P., Yu, H.-J., Hu, S.-X.: Optimal choice of parameters for particle swarm optimization. J. Zhejiang Univ. Sci. **6A**(6), 528–534 (2005)

Zheng, Y., Ma, L., Zhang, L., Qian, J.: Empirical study of particle swarm optimizer with an increasing inertia weight. In: Congress on Evolutionary Computation, IEEE (2003)