

The development of some rotationally invariant population based optimization methods

Marthinus Nicolaas Ras

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Mechanical) in the Faculty of Engineering at Stellenbosch University



Supervisor: Albert A. Groenwold

March 2013

DECLARATION

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Signature:

Date:

Abstract

In this study we consider the lack of rotational invariance of three different population based optimization methods, namely the particle swarm optimization (PSO) algorithm, the differential evolution (DE) algorithm and the continuous-parameter genetic algorithm (CPGA). We then propose rotationally invariant versions of these algorithms.

We start with the PSO. The so-called classical PSO algorithm is known to be *variant* under rotation, whereas the linear PSO is rotationally *invariant*. This invariance however, comes at the cost of lack of diversity, which renders the linear PSO inferior to the classical PSO.

The previously proposed so-called *diverse rotationally invariant* (DRI) PSO is an algorithm that aims to combine both diversity and invariance. This algorithm is rotationally invariant in a *stochastic sense* only. What is more, the formulation depends on the introduction of a random rotation matrix S , but invariance is only guaranteed for ‘small’ rotations in S . Herein, we propose a formulation which is diverse and *strictly invariant* under rotation, if still in a stochastic sense only. To do so, we depart with the linear PSO, and then we add a self-scaling random vector with a standard normal distribution, sampled uniformly from the surface of a n -dimensional unit sphere.

For the DE algorithm, we show that the classic *DE/rand/1/bin* algorithm, which uses constant mutation and standard crossover, is rotationally *variant*. We then study a previously proposed rotationally invariant DE formulation in which the crossover operation takes place in an orthogonal base constructed using Gram-Schmidt orthogonalization.

We propose two new formulations by firstly considering a very simple rotationally *invariant* formulation using constant mutation and whole arithmetic crossover. This rudimentary formulation performs badly, due to lack of diversity. We then introduce diversity into the formulation using two distinctly different strategies. The first adjusts the crossover step by perturbing the direction of the linear combination between the target vector and the mutant vector. This formulation is invariant in a stochastic sense only. We add a self-scaling random vector to the unaltered whole arithmetic crossover vector. This formulation is strictly invariant, if still in a stochastic sense only.

For the CPGA we show that a standard CPGA using blend crossover and standard mutation, is rotationally *variant*. To construct a rotationally invariant CPGA it is possible to modify the crossover operation to be rotationally invariant. This however, again results in loss of diversity. We introduce diversity in two ways: firstly using a modified mutation scheme, and secondly, following the same approach as in the PSO and the DE, by adding a self-scaling random vector to the offspring vector. This formulation is strictly invariant, albeit still in a stochastic sense only.

Numerical results are presented for the variant and invariant versions of the respective algorithms. The intention of this study is not the contribution of yet another competitive and/or superior popu-

lation based algorithm, but rather to present formulations that are both diverse and *invariant*, in the hope that this will stimulate additional future contributions, since rotational invariance in general is a desirable, salient feature for an optimization algorithm.

Opsomming

In hierdie studie bestudeer ons die gebrek aan rotasionele invariansie van drie verskillende populasie-gebaseerde optimeringsmetodes, met name die partikel-swerm optimerings (PSO) algoritme, die differensiële evolusie (DE) algoritme en die kontinue-parameter genetiese algoritme (KPGA). Ons stel dan rotasionele invariante weergawes van hierdie algoritmes voor.

Ons begin met die PSO. Die sogenaamde klassieke PSO algoritme is bekend dat dit variant is onder rotasie, terwyl die lineêre PSO rotasioneel invariant is. Hierdie invariansie lei tot 'n gebrek aan diversiteit in die algoritme, wat beteken dat die lineêre PSO minder goed presteer as die klassieke PSO.

Die voorheen voorgestelde sogenaamde *diverse rotasionele invariante* (DRI) PSO is 'n algoritme wat beoog om beide diversiteit en invariansie te kombineer. Hierdie algoritme is slegs rotasioneel invariant in 'n stogastiese sin. Boonop is die formulering afhanklik van 'n willekeurige rotasie matriks S , maar invariansie is net gewaarborg vir 'klein' rotasies in S . In hierdie studie stel ons 'n formulering voor wat divers is en *streng invariant* onder rotasie, selfs al is dit steeds net in 'n stogastiese sin. In hierdie formulering, vertrek ons met die lineêre PSO, en voeg dan 'n self-skalerende ewekansige vektor met 'n standaard normaalverdeling by, wat eenvormig van die oppervlakte van 'n n -dimensionele eenheid sfeer geneem word.

Vir die DE algoritme toon ons aan dat die klassieke *DE/rand/1/bin* algoritme, wat gebruik maak van konstante mutasie en standaard kruising rotasioneel *variant* is. Ons bestudeer dan 'n voorheen voorgestelde rotasionele invariante DE formulering waarin die kruisingsoperasie plaasvind in 'n ortogonale basis wat gekonstrueer word met behulp van die Gramm-Schmidt ortogonaliseringsproses.

Verder stel ons dan twee nuwe formulerings voor deur eerstens 'n baie eenvoudige rotasionele *invariante* formulering te oorweeg, wat konstante mutasie en volledige rekenkundige kruising gebruik. Hierdie elementêre formulering onderpresteer as gevolg van die afwesigheid van diversiteit. Ons voeg dan diversiteit by die formulering toe, deur gebruik te maak van twee afsonderlike strategieë. Die eerste verander die kruisings stap deur die rigting van die lineêre kombinasie tussen die teiken vektor en die mutasie vektor te perturbeer. Hierdie formulering is slegs invariant in 'n stogastiese sin. In die ander formulering, soos met die nuwe rotasionele invariante PSO, voeg ons bloot 'n self-skalerende ewekansige vektor by die onveranderde volledige rekenkundige kruisingsvektor. Hierdie formulering is *streng invariant* onder rotasie, selfs al is dit steeds net in 'n stogastiese sin.

Vir die KPGA wys ons dat die standaard KPGA wat gemengde kruising en standaard mutasies gebruik, rotasioneel *variant* is. Om 'n rotasionele invariante KPGA te konstrueer is dit moontlik om die kruisingsoperasie aan te pas. Dit veroorsaak weereens 'n verlies aan diversiteit. Ons

maak die algoritmes divers op twee verskillende maniere: eerstens deur gebruik te maak van 'n gewysigde mutasie skema, en tweedens deur die selfde aanslag te gebruik as in die PSO en die DE, deur 'n self-skalerende ewekansige vektor by die nageslag vektor te voeg. Hierdie formulering is *streng invariant* onder rotasie, selfs al is dit steeds net in 'n stogastiese sin.

Numeriese resultate word vir die variante en invariante weergawe van die onderskeie algoritmes verskaf.

Die doel van hierdie studie is nie die bydrae van bloot nog 'n kompetierend en/of beter populasie-gebaseerde optimeringsmetode nie, maar eerder om formuleringe voor te lê wat beide divers en *invariant* is, met die hoop dat dit in die toekoms bykomende bydraes sal stimuleer, omdat rotasionele invariansie in die algemeen 'n aantreklike, belangrike kenmerk is vir 'n optimerings algoritme.

Contents

Abstract	iii
Opsomming	v
List of Figures	x
List of Tables	xi
1 Introduction	1
2 Background	4
2.1 Concept of rotational invariance	4
2.2 Hadamard product	5
2.3 Problem formulation	5
2.4 Initialization	6
2.5 Averaging over the number of runs	6
3 PSO	8
3.1 Basic formulation of the PSO	9
3.1.1 Linear velocity update rule	9
3.1.2 Classical velocity update rule	9
3.2 Diverse rotationally invariant (DRI) velocity update rule	10
3.3 A strictly (stochastic) rotationally invariant (SRI) velocity update rule	10
3.3.1 Pseudocode	11
3.4 Numerical experiments	12
3.4.1 Scalability of algorithms SRI and DRI	14
3.5 Summary	16
4 DE	20

<i>CONTENTS</i>	viii
4.1 Differential evolution algorithm	21
4.1.1 Differential mutation	21
4.1.2 Crossover	22
4.1.3 Selection	24
4.2 Rotationally invariant differential evolution algorithms	24
4.2.1 Rotationally invariant crossover using Gram-Schmidt orthogonalization	25
4.2.2 Perturbation rotation invariant crossover (PRICO)	25
4.2.3 Strictly (stochastic) rotationally invariant crossover (SSRICO)	26
4.3 Numerical experiments	27
4.4 Summary	30
5 CPGA	45
5.1 Continuous-parameter genetic algorithm	45
5.1.1 Selection	46
5.1.2 Blend crossover (BLX- α)	46
5.1.3 Mutation scheme	47
5.1.4 Elitist strategy	48
5.2 Rotation invariant continuous-parameter genetic algorithm	48
5.2.1 Modified mutation scheme	48
5.2.2 Strictly (stochastically) rotationally invariant continuous-parameter genetic algorithm (CPGA(S))	48
5.3 Numerical experiments	50
5.4 Summary	54
6 Conclusion	68
References	69

List of Figures

1.1	Function value contour plot relating f and \hat{f}	1
2.1	Average function value as a function of the number of runs over which is averaged	7
3.1	Average function values for the Rosenbrock test function f_1 using PSO	14
3.2	Average function values for the Quadric test function f_2 using PSO	16
3.3	Average function values for the Ackley test function f_3 using PSO	17
3.4	Average function values for the Rastrigin test function f_4 using PSO	17
3.5	Average function values for the Griewank test function f_5 using PSO	18
3.6	Average function values for the Dixon-Price test function f_6 using PSO	18
3.7	CPU time of the DRI and SRI algorithms as a function of dimensionality	19
4.1	Average function values for the Rosenbrock test function f_1 using classical DE and GSRIDE	33
4.2	Average function values for the Rosenbrock test function f_1 using SRIDE and PRICO	34
4.3	Average function values for the Quadric test function f_2 using classical DE and GSRIDE	35
4.4	Average function values for the Quadric test function f_2 using SRIDE and PRICO	36
4.5	Average function values for the Ackley test function f_3 using classical DE and GSRIDE	37
4.6	Average function values for the Ackley test function f_3 using SRIDE and PRICO	38
4.7	Average function values for the Rastrigin test function f_4 using classical DE and GSRIDE	39
4.8	Average function values for the Rastrigin test function f_4 using SRIDE and PRICO	40
4.9	Average function values for the Griewank test function f_5 using classical DE and GSRIDE	41
4.10	Average function values for the Griewank test function f_5 using SRIDE and PRICO	42
4.11	Average function values for the Dixon-Price test function f_6 using classical DE and GSRIDE	43

LIST OF FIGURES

x

4.12	Average function values for the Dixon-Price test function f_6 using SRIDE and PRICO	44
5.1	Average function values for the Rosenbrock test function f_1 using SPGA and CPGA(M)	55
5.2	Average function values for the Rosenbrock test function f_1 using CPGA(S)	56
5.3	Average function values for the Quadric test function f_2 using SPGA and CPGA(M)	57
5.4	Linear average function values for the Quadric test function f_2 using CPGA(S)	58
5.5	Expansion of 100 runs for the Quadric test function f_2 , using CPGA(S)	58
5.6	Log average function values for the Quadric test function f_2 using CPGA(S)	59
5.7	Average function values for the Ackley test function f_3 using SPGA and CPGA(M)	60
5.8	Average function values for the Ackley test function f_3 using CPGA(S)	61
5.9	Average function values for the Rastrigin test function f_4 using SPGA and CPGA(M)	62
5.10	Average function values for the Rastrigin test function f_4 using CPGA(S)	63
5.11	Average function values for the Griewank test function f_5 using SPGA and CPGA(M)	64
5.12	Average function values for the Dixon-Price test function f_6 using CPGA(S)	65
5.13	Average function values for the Dixon-Price test function f_6 using SPGA and CPGA(M)	66
5.14	Average function values for the Dixon-Price test function f_6 using CPGA(S)	67

List of Tables

3.1	Constant inertia factor w at which the best average objective function value is obtained using PSO	15
4.1	Constant parameters at which the best average objective function value is obtained using DE	31
4.2	Average objective function values obtained using DE with random parameters . . .	32
5.1	Constant parameters at which the best average objective function value is obtained using CPGA	53

Chapter 1

Introduction

To set the scene, we depart with a verbatim setting from Wilke *et al.* [1]: Consider the scale and frame in which an optimization problem is defined. Figure 1.1 depicts two reference frames \mathbf{x} and $\hat{\mathbf{x}}$, related by a scale factor s , translation by a vector \mathbf{t} , and rotation by a proper orthogonal matrix \mathbf{Q} , i.e. $\hat{\mathbf{x}} = \mathbf{t} + s\mathbf{Q}\mathbf{x}$. A given function is expressed in these reference frames as $f(\hat{\mathbf{x}}) = f(\mathbf{t} + s\mathbf{Q}\mathbf{x})$. An alternative but equivalent interpretation of Figure 1.1 is that f and \hat{f} are two distinct functions described in the same reference frame \mathbf{x} , i.e. $f(\mathbf{x}) = \hat{f}(\mathbf{t} + s\mathbf{Q}\mathbf{x})$.

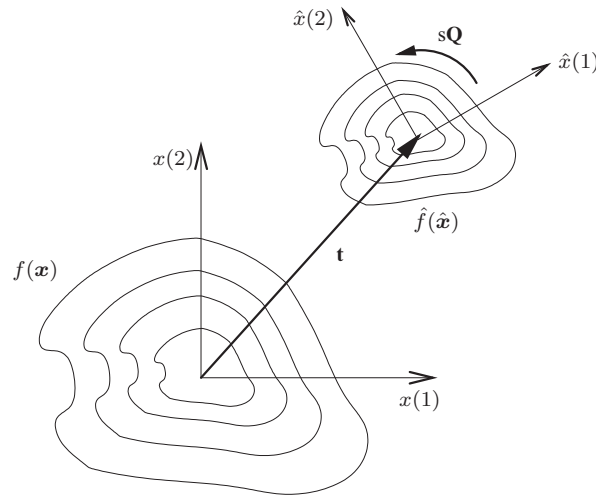


Figure 1.1: Function value contour plot. Two possible interpretations are that f and \hat{f} are distinct functions in the same reference frame, or that f is a single function expressed in two reference frames \mathbf{x} and $\hat{\mathbf{x}}$. s , \mathbf{t} and \mathbf{Q} denote scaling, translation and rotation respectively

Let us now consider ‘invariance’ in the context of optimization algorithms. ‘Scale invariance’ implies algorithm performance that is independent of uniform scaling of all variables. ‘Frame invariance’ on the other hand implies algorithm performance that is independent of frame translation and rotation.

Frame indifference (or objectivity), is well known in classical mechanics [2], where physical laws dictate that this principle must hold. However, no corresponding law in optimization theory re-

quires that an optimization algorithm must be frame invariant. It might therefore seem that frame indifference is merely an aesthetic requirement.

However, arguments in favor of frame invariant optimization algorithms do exist. Due to the lack of a physical justification, the arguments are necessarily based on a user's perspective. If a particular optimization algorithm is frame dependent, it follows that there exists a specific choice of reference frame in which the problem can be solved easier (i.e. requiring less iterations) or better (i.e. achieving a lower cost function value) as compared to some other reference frame. In general, the performance difference in different reference frames can not be quantified, and depends amongst others on the optimization problem and algorithm specifics. Since *a priori* knowledge of the optimal reference frame for a particular problem is seldom available, this places an additional burden on the analyst, which now has to consider solving the problem in a number of reference frames. (An exception being the specialized class of separable functions.) If the algorithm's frame dependency is severe, the analyst requires some external procedure to take the algorithm's frame dependency into account. A conceptual procedure is to recast the optimization problem to simultaneously solve for the reference frame and solution, but this renders the problem ill-posed.

An alternative phrasing of the above argument is as follows: A frame dependent algorithm implies a bias towards some particular reference frame (or frames). This in turn implies a bias towards some subclass of problems. If a problem is well suited to be solved in a particular reference frame, a frame dependent algorithm that is *aligned* with this specific reference frame will perform well. This however, implies that frame *dependent* optimization algorithms are *specialist* algorithms, tuned to perform well on a special subclass of problems. In contrast, frame *invariant* optimization algorithms are *general* algorithms, applicable to a larger class of problems.

But, how do we know if a problem is well suited to be solved in a particular reference frame? In general, we do not. Therefore, the choice of a frame dependent algorithm to solve a particular problem makes the *tacit assumption* that the problem's reference frame is well matched with the algorithm's reference frame bias. If the tacit assumption holds, good performance is expected, since the correct assumption implies additional problem information. Again, an example of practical relevance is the class of separable or decomposable optimization problems, where an n -dimensional problem is simply the sum of n 1-dimensional problems. An algorithm that independently searches along the coordinate axes, (which renders such an algorithm frame dependent), exploits this specific function characteristic. It is therefore expected that such an algorithm will be superior to its frame invariant counterpart.

To summarize, frame invariance of optimization algorithms is not a strict requirement. Frame invariance does however, provide a useful classification of optimization algorithms. A frame invariant algorithm requires less *a priori* knowledge (or tacit assumptions) of the optimization problem, as compared to a frame variant algorithm. This implies that a frame invariant algorithm will have either inferior or superior performance as compared to a frame variant algorithm, depending on the validity of the assumed information. In addition, frame invariance is desirable as it increases the predictive power of numerical performance results [3].

Even though frame invariance is not required, some classes of optimization algorithms do satisfy this principle. In classical gradient based optimization [4], the gradient vector (or some conjugate direction to the gradient), indicates some direction of improvement, even if this direction is not optimal. The gradient vector of any scalar function satisfies the transformation rules for chang-

ing both scale and reference frame. This (usually) renders classical gradient based optimization algorithms scale and reference frame invariant.

However, in so-called modern (stochastic) optimization procedures, like the population based methods we discuss herein, few algorithms satisfy the requirement of reference frame invariance. In this study we will introduce five new rotationally invariant algorithms.

The thesis is constructed as follows: we present some concepts crucial for the thesis in Chapter 2. In Chapter 3 we present a strictly stochastically rotationally invariant particle swarm optimization (PSO) formulation, in Chapter 4 we will study the rotational (in)variance of the differential evolution (DE) algorithm and in Chapter 5 we will study the rotational (in)variance of the continuous-parameter genetic algorithm (CPGA). Finally we conclude our study in Chapter 6.

Chapter 2

Background

In this chapter we present select concepts crucial to the rest of the thesis. In Section 2.1 we present the concept of rotational invariance, in Section 2.2 we present a convenient notation for use throughout this thesis. In Section 2.3 we present the formal optimization problem formulation, in Section 2.4 we discuss the initialization steps of the algorithms to follow, and in Section 2.5 we reflect on the number of runs used in the numerical experiments.

2.1 Concept of rotational invariance

Let us consider two related functions f and \hat{f} in the same Cartesian reference frame; again we depart with Wilke *et al.* [1]. The two functions are related through an arbitrary scale factor $s \in \mathbb{R}$, translation by an arbitrary vector $\mathbf{t} \in \mathbb{R}^n$, and rotation by an arbitrary proper orthogonal matrix \mathbf{Q} (henceforth denoted $\mathbf{Q} \in \text{Orth}^+$). Recall that a matrix \mathbf{Q} is orthogonal if and only if $\mathbf{Q}^T \mathbf{Q} = \mathbf{Q} \mathbf{Q}^T = \mathbf{I}$. A proper orthogonal matrix \mathbf{Q} has the additional property $\det(\mathbf{Q}) = +1$. By definition,

$$f(\mathbf{x}) = \hat{f}(\hat{\mathbf{x}}), \quad (2.1)$$

where \mathbf{x} and $\hat{\mathbf{x}}$ represent two design vectors. We denote any arbitrary scaled, translated and rotated quantity by a superscript caret (^).

It follows from vector analysis (e.g. see [5]) that the relation between the two vectors \mathbf{x} and $\hat{\mathbf{x}}$ is given by

$$\hat{\mathbf{x}} = s\mathbf{Q}\mathbf{x} + \mathbf{t}, \quad (2.2)$$

for any $s \in \mathbb{R}$, $\mathbf{t} \in \mathbb{R}^n$ and $\mathbf{Q} \in \text{Orth}^+$. Substituting (2.2) into (2.1) gives

$$f(\mathbf{x}) = \hat{f}(s\mathbf{Q}\mathbf{x} + \mathbf{t}). \quad (2.3)$$

Although (2.2) depicts a specific sequence of scaling, translation and rotation of \mathbf{x} , it is noted that it is in fact completely general, e.g.

$$\begin{aligned} \hat{\mathbf{x}} &= s\mathbf{Q}(\mathbf{x} + \mathbf{t}) \\ &= s\mathbf{Q}\mathbf{x} + s\mathbf{Q}\mathbf{t} \\ &= s\mathbf{Q}\mathbf{x} + \mathbf{c}, \end{aligned} \quad (2.4)$$

where $\mathbf{c} = s\mathbf{Q}\mathbf{t}$ now represents the translation vector.

Let us consider an additional vector transformation rule. Consider two arbitrary vectors \mathbf{x}_1 and \mathbf{x}_2 and the difference vector $\mathbf{v} = \mathbf{x}_1 - \mathbf{x}_2$. In addition, consider related vectors $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$ and $\hat{\mathbf{v}} = \hat{\mathbf{x}}_1 - \hat{\mathbf{x}}_2$. The relation between \mathbf{v} and $\hat{\mathbf{v}}$ is

$$\begin{aligned}\hat{\mathbf{v}} &= \hat{\mathbf{x}}_1 - \hat{\mathbf{x}}_2 \\ &= (s\mathbf{Q}\mathbf{x}_1 + \mathbf{t}) - (s\mathbf{Q}\mathbf{x}_2 + \mathbf{t}) \\ &= s\mathbf{Q}(\mathbf{x}_1 - \mathbf{x}_2) \\ &= s\mathbf{Q}\mathbf{v},\end{aligned}\tag{2.5}$$

for any $s \in \mathbb{R}$ and $\mathbf{Q} \in \text{Orth}^+$.

Any optimization algorithm is scale, translation and rotation invariant *if and only if* the transformation rules given by (2.2) and (2.5) are satisfied *for all* $\mathbf{Q} \in \text{Orth}^+$; the same optimal results are then obtained irrespective of the scale and reference frame used.

For a stochastic optimization procedure, we may choose to satisfy (2.2) and (2.5) such that the algorithm's performance is invariant of scale, translation and rotation in a *stochastic sense*. We will refer to this as 'stochastic frame invariance'.

In this study, we consider bounded, unconstrained functions f and \hat{f} , respectively defined over domains D and \hat{D} . $\mathbf{x} \in D$ and $\hat{\mathbf{x}} \in \hat{D}$ each represents a set of allowable design vectors, which are related through $\hat{D} = s\mathbf{Q}D + \mathbf{t}$.

2.2 Hadamard product

Consider two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, we note that the entry wise product (Hadamard product) [6] denoted by \circ in $\mathbf{c} = \mathbf{a} \circ \mathbf{b}$ is equivalent to the matrix vector multiplication

$$\mathbf{c} = \mathbf{A}\mathbf{b},\tag{2.6}$$

with the diagonal matrix \mathbf{A} given by $A(i, i) = a(i)$, $\forall i = 1, 2, \dots, n$. We make use of this result throughout our study in an interchangeable fashion in order to ease our theoretical study of frame (in)variance. Our notation is uppercase for diagonal matrix representation implying matrix multiplication and lowercase for vector representation together with the Hadamard product.

2.3 Problem formulation

For the sake of brevity, we restrict ourselves to the unconstrained or bounded constrained multimodal global optimization problem which we will define as follows: find the global minimum value $f(\mathbf{x}^*)$ of a given real-valued function $f : D \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, such that

$$f^* = f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in D,\tag{2.7}$$

where D is the allowable (bounded) search space. Since this problem is intractable, the aim is usually to find a suitably low approximation \bar{f} to f^* .

The allowable search space is described by lower and upper limits within which trial vectors must remain respectively given by \mathbf{x}_{LB} and \mathbf{x}_{UB} both in \mathbb{R}^n .

2.4 Initialization

All the algorithms to follow start with this step. For the sake of brevity we discuss it here instead of repeating this in each chapter.

The initial position of particle i is given by

$$\mathbf{x}_0^i = \mathbf{x}_{LB} + \mathbf{R}(\mathbf{x}_{UB} - \mathbf{x}_{LB}), \quad (2.8)$$

with \mathbf{R} a diagonal matrix with each diagonal entry a random scalar $\in [0, 1]$.

For an arbitrary rotated reference frame we obtain

$$\begin{aligned} \hat{\mathbf{x}}_0^i &= \hat{\mathbf{x}}_{LB} + \hat{\mathbf{R}}(\hat{\mathbf{x}}_{UB} - \hat{\mathbf{x}}_{LB}) \\ &= s\mathbf{Q}\mathbf{x}_{LB} + \mathbf{t} + \hat{\mathbf{R}}(s\mathbf{Q}\mathbf{x}_{UB} + \mathbf{t} - s\mathbf{Q}\mathbf{x}_{LB} - \mathbf{t}) \\ &= s\mathbf{Q}\mathbf{x}_{LB} + \mathbf{t} + \hat{\mathbf{R}}s\mathbf{Q}(\mathbf{x}_{UB} - \mathbf{x}_{LB}). \end{aligned} \quad (2.9)$$

The expected transformation for a frame invariant transformation would be

$$\begin{aligned} \hat{\mathbf{x}}_0^i &= s\mathbf{Q}\mathbf{x}_0^i + \mathbf{t} \\ &= s\mathbf{Q}(\mathbf{x}_{LB} + \mathbf{R}(\mathbf{x}_{UB} - \mathbf{x}_{LB})) + \mathbf{t} \\ &= s\mathbf{Q}\mathbf{x}_{LB} + \mathbf{t} + s\mathbf{Q}\mathbf{R}(\mathbf{x}_{UB} - \mathbf{x}_{LB}). \end{aligned} \quad (2.10)$$

In order to obtain a frame invariant transformation we need

$$\hat{\mathbf{R}}\mathbf{Q} = \mathbf{Q}\mathbf{R}, \quad (2.11)$$

which results in

$$\hat{\mathbf{R}} = \mathbf{Q}\mathbf{R}\mathbf{Q}^T, \quad \forall \mathbf{Q} \in \text{Orth}^+. \quad (2.12)$$

Since we consider frame invariance of algorithms and not of functions we need to rotate the hypercube together in space with a function according to (2.12) in order to quantify the frame (in)variance instead of the invariance of the cost function [7].

This implies that the random numbers should be generated in the rotated reference frame in order to recover frame invariance in an exact sense. This is in general not possible as we do not know \mathbf{Q} , as discussed in Chapter 1.

It is noted that when simple bounds are considered, $x_{LB} = -x_{UB}$ and equal for all the design variables, then the domain \mathbf{D} can be viewed as an n -dimensional cube. One could then redefine the domain to be an n -dimensional sphere with radius $r = \frac{1}{2}\|x_{UB} - x_{LB}\|$ with the origin as the centre point. Initialization by randomly generating points within this spherical domain would then be *strictly stochastically invariant*, in a very similar fashion to the SRI PSO, SSRICO and CPGA(S) formulations presented later in this study.

2.5 Averaging over the number of runs

In the numerical experiments performed in Sections 3.4, 4.3 and 5.3, it is required to average the results over a number of runs to gauge the average performance of the algorithms, since they are

stochastic. Figure 2.1 reveals that the average function value stabilizes as the number of runs over which is averaged, increases towards 150 runs. In this case we have used SCPGA (discussed in Section 5.1) for Problem 3 (presented in Sections 3.4, 4.3 and 5.3), unrotated, with $\alpha = 0.5$ and $P_m = 0.001$.

The function value stabilizes after say 30 runs, but is still affected by outliers at around 65 and 80. The effect of the outliers are neglectable for our purposes and 100 runs seems a reasonable number in the remainder of this study.

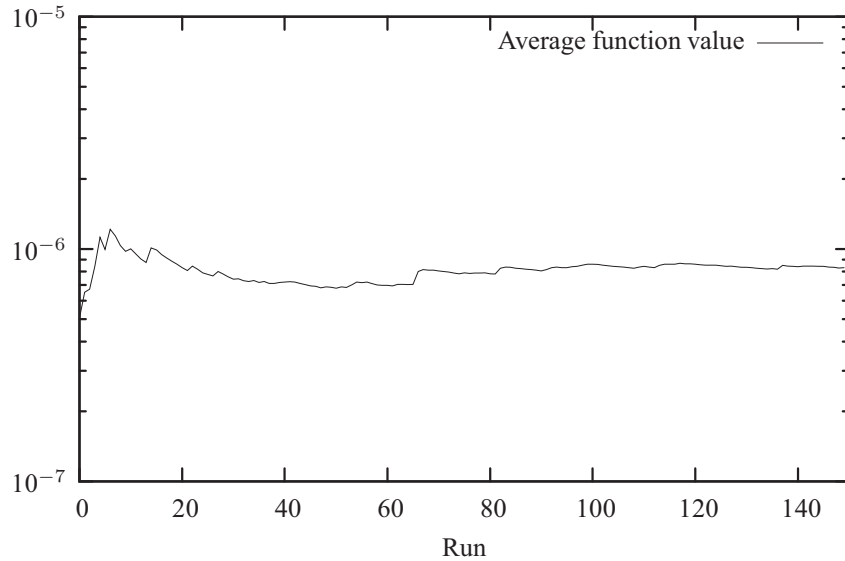


Figure 2.1: Average function value as a function of the number of runs over which is averaged

Chapter 3

A strictly stochastically rotationally invariant PSO formulation

The work presented in this chapter is reproduced from a paper titled “A strictly stochastically rotationally invariant PSO formulation” [8]. The paper is co-authored by Dr Daniel N. Wilke of the Mechanical and Aeronautical Engineering, University of Pretoria, Pretoria, South Africa, Prof. Albert A. Groenwold of the Department of Mechanical Engineering at the University of Stellenbosch, Stellenbosch, South Africa, and Dr S. Kok of Advanced Mathematical Modelling, CSIR Modelling and Digital Science, Pretoria, South Africa.

Previously Wilke *et al.* [1] proposed the so-called *diverse rotationally invariant* (DRI) PSO, an algorithm that aims to combine both diversity and invariance. This algorithm is rotationally invariant in a *stochastic sense* only. In this chapter we reflect on the invariance under rotation of the DRI PSO [1] and a newly proposed velocity update rule. To do so, we will study a ‘vanilla’ PSO, i.e. we do not implement any heuristics such as maximum velocity restriction, dynamic inertia adjustment, position restriction or local neighborhoods, etc., since this will merely confuse the issue we wish to address.

We propose an example of a velocity update rule that is both diverse and *strictly stochastically invariant* w.r.t. the reference frame used. The new velocity update rule simply adds a third term to the existing invariant search direction to create diversity. The proposed formulation is also strictly invariant under scale and translation.

This chapter is arranged as follows: in Section 3.1, we present a brief overview of the PSO, including the linear and classical velocity update rules in Sections 3.1.1 and 3.1.2, respectively. We discuss the previously proposed DRI PSO in Section 3.2. Our newly proposed velocity update rule, denoted the *strictly (stochastic) rotationally invariant* (SRI) PSO, is then presented and studied in Section 3.3. In Section 3.4, we present numerical results for a modest set of test problems; the intention being not to contribute yet another competitive and/or superior PSO variant, but to reflect on invariance itself. This includes a brief reflection on scalability of the computational effort required for algorithm SRI as compared to algorithm DRI. Finally, we summarize the finding of this chapter in Section 3.5.

3.1 Basic formulation of the PSO

Consider a swarm of p particles in an n -dimensional search space [1]. The first step of course is initialization, already discussed in Section 2.4. The position vector \mathbf{x}_k^i of each particle i is updated by

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \mathbf{v}_{k+1}^i, \quad (3.1)$$

where k is a unit pseudo time increment (iteration number). \mathbf{v}_{k+1}^i represents the velocity vector that is obtained from the velocity rule, given by

$$\mathbf{v}_{k+1}^i = w\mathbf{v}_k^i + \boldsymbol{\nu}_k^i, \quad (3.2)$$

where the inertia factor w is a real number, typically between 0.4 and 0.9 [9, 10, 11], and $\boldsymbol{\nu}_k^i$ is the stochastic ‘velocity’ vector.

3.1.1 Linear velocity update rule

The stochastic vector $\boldsymbol{\nu}_k^i$ for the linear velocity update rule [12] is given by

$$\boldsymbol{\nu}_k^i = c_1 r_{1k}^i (\mathbf{p}_k^i - \mathbf{x}_k^i) + c_2 r_{2k}^i (\mathbf{p}_k^g - \mathbf{x}_k^i), \quad (3.3)$$

with r_{1k}^i and r_{2k}^i random scalar numbers $\in [0, 1]$, and \mathbf{p}_k^i and \mathbf{p}_k^g respectively being the fittest positions found for particle i and the complete group or swarm up to iteration k . The linear velocity update rule is scale, translation and rotation invariant [1]. This means that the transformed (scaled, translated and rotated) stochastic vector $\hat{\boldsymbol{\nu}}_k^i$ is given by

$$\begin{aligned} \hat{\boldsymbol{\nu}}_k^i &= c_1 r_{1k}^i (\hat{\mathbf{p}}_k^i - \hat{\mathbf{x}}_k^i) + c_2 r_{2k}^i (\hat{\mathbf{p}}_k^g - \hat{\mathbf{x}}_k^i) \\ &= s\mathbf{Q}\boldsymbol{\nu}_k^i. \end{aligned} \quad (3.4)$$

The intrinsic properties of a vector are its magnitude and direction; these exist independent of a reference frame [5]. In the linear velocity update rule, only the vector magnitudes (which are invariant) are randomly scaled. Consequently, the linear velocity update rule demonstrates ‘magnitudal diversity’, but lacks ‘directional diversity’. This results in particle trajectories that collapse to lines [13].

3.1.2 Classical velocity update rule

For the classical velocity update rule [14, 15] the stochastic vector $\boldsymbol{\nu}_k^i$ is

$$\boldsymbol{\nu}_k^i = c_1 \boldsymbol{\Theta}_{1k}^i \circ (\mathbf{p}_k^i - \mathbf{x}_k^i) + c_2 \boldsymbol{\Theta}_{2k}^i \circ (\mathbf{p}_k^g - \mathbf{x}_k^i), \quad (3.5)$$

with $\boldsymbol{\Theta}_{1k}^i$ and $\boldsymbol{\Theta}_{2k}^i$ row vectors of random numbers $\in [0, 1]$. The classical velocity update rule has been shown to be rotationally *variant* [1]. This means that the transformed stochastic vector

$$\begin{aligned} \hat{\boldsymbol{\nu}}_k^i &= c_1 \hat{\boldsymbol{\Theta}}_{1k}^i \circ (\hat{\mathbf{p}}_k^i - \hat{\mathbf{x}}_k^i) + c_2 \hat{\boldsymbol{\Theta}}_{2k}^i \circ (\hat{\mathbf{p}}_k^g - \hat{\mathbf{x}}_k^i) \\ &\neq s\mathbf{Q}\boldsymbol{\nu}_k^i, \end{aligned} \quad (3.6)$$

and thus does not satisfy (2.5). However, the classical velocity update rule is strictly *invariant* under scaling and translation [1].

3.2 Diverse rotationally invariant (DRI) velocity update rule

Previously, a diverse rotationally invariant velocity update rule was proposed by Wilke et al. [1]. The formulation randomly scales the vector magnitudes of $(\mathbf{p}_k^i - \mathbf{x}_k^i)$ and $(\mathbf{p}_k^g - \mathbf{x}_k^i)$. In addition, it imposes ‘small’ perturbations of the vector directions $(\mathbf{p}_k^i - \mathbf{x}_k^i)$ and $(\mathbf{p}_k^g - \mathbf{x}_k^i)$. The vector directions are perturbed by multiplying each of the above vectors with an independent *random rotation matrix* \mathbf{S} . The random rotation matrices are constructed anew for each particle i and for every iteration k , i.e.

$$\mathbf{v}_k^i = c_1 r_{1k}^i \mathbf{S}_{1k}^i (\mathbf{p}_k^i - \mathbf{x}_k^i) + c_2 r_{2k}^i \mathbf{S}_{2k}^i (\mathbf{p}_k^g - \mathbf{x}_k^i), \quad (3.7)$$

with each \mathbf{S}_{lk}^i , $l = 1, 2$, a random rotation matrix of dimension $n \times n$.

Wilke et al. [1] relax the requirements for reference frame invariance by only requiring that the mean objective function value over a large number of runs is invariant of the reference frame, and they denote these algorithms to be rotationally invariant in a *stochastic sense*. Furthermore, they demonstrate that stochastic rotational invariance requires that the rotations are ‘small’. They then use the linear approximation to a rotation matrix as a computationally viable alternative to computing \mathbf{S}_{lk}^i for small rotations as

$$\mathbf{S}_k^i = \mathbf{I} + \mathbf{W}_k^i. \quad (3.8)$$

\mathbf{W} is a random skew matrix constructed as

$$\mathbf{W} = \frac{\tau\pi}{180} (\mathbf{A} - \mathbf{A}^\top), \quad (3.9)$$

with \mathbf{A} an $n \times n$ random matrix with each entry a uniform random number $\in [-0.5 \ 0.5]$, and τ a real scaling factor.

3.3 A strictly (stochastic) rotationally invariant (SRI) velocity update rule

As discussed in Section 3.1.1, the linear velocity update rule is rotationally *invariant*, but the particle trajectories collapse to lines. The advantage of directionally diverse (n -dimensional) particle search trajectories was quantified in [13]. On the other hand, the classical velocity update rule allows for particles to have directionally diverse search trajectories, but unfortunately this comes at the cost of rotational *variance*.

Based on the linear velocity update rule, we propose an alternative directionally diverse, rotationally invariant velocity update rule herein, denoted the strictly (stochastic) rotational invariant (SRI) velocity update rule. The proposed formulation simply adds a third term to the linear velocity update rule in order to introduce diversity. The new term is simply a scaled, normalized random vector with a *standard normal distribution*. This results in a point sampled *uniformly* on the surface of a n dimensional unit sphere [16].

The proposed formulation is

$$\mathbf{v}_k^i = c_1 r_{1k}^i (\mathbf{p}_k^i - \mathbf{x}_k^i) + c_2 r_{2k}^i (\mathbf{p}_k^g - \mathbf{x}_k^i) + c_s r_{3k}^i \|\mathbf{p}_k^i - \mathbf{p}_k^g\| \Phi_k^i, \quad (3.10)$$

with Φ_k^i representing a normalized vector with a standard distribution, r_{3k}^i a random scalar and c_s a scaling factor. We introduce $\|\mathbf{p}_k^i - \mathbf{p}_k^g\|$ to assist with convergence and termination, but alternative possibilities exist. The formulation can be abbreviated as

$$\boldsymbol{\nu}_k^i = \boldsymbol{\nu}_{\text{lin}_k^i} + c_s r_{3k}^i \|\mathbf{p}_k^i - \mathbf{p}_k^g\| \Phi_k^i, \quad (3.11)$$

with $\boldsymbol{\nu}_{\text{lin}_k^i}$ the linear velocity update rule given in (3.3).

The transformed stochastic vector $\hat{\boldsymbol{\nu}}_k^i$ is given by

$$\begin{aligned} \hat{\boldsymbol{\nu}}_k^i &= c_1 r_{1k}^i (\hat{\mathbf{p}}_k^i - \hat{\mathbf{x}}_k^i) + c_2 r_{2k}^i (\hat{\mathbf{p}}_k^g - \hat{\mathbf{x}}_k^i) + c_s r_{3k}^i \|\hat{\mathbf{p}}_k^i - \hat{\mathbf{p}}_k^g\| \Phi_k^i \\ &= \hat{\boldsymbol{\nu}}_{\text{lin}_k^i} + c_s r_{3k}^i \|s\mathbf{Q}\mathbf{p}_k^i + \mathbf{t} - s\mathbf{Q}\mathbf{p}_k^g - \mathbf{t}\| \Phi_k^i \\ &= s\mathbf{Q}\boldsymbol{\nu}_{\text{lin}_k^i} + c_s r_{3k}^i \|s\mathbf{Q}(\mathbf{p}_k^i - \mathbf{p}_k^g)\| \Phi_k^i \\ &= s(\mathbf{Q}\boldsymbol{\nu}_{\text{lin}_k^i} + c_s r_{3k}^i \|\mathbf{p}_k^i - \mathbf{p}_k^g\| \Phi_k^i). \end{aligned} \quad (3.12)$$

The required transformation is

$$\begin{aligned} \hat{\boldsymbol{\nu}}_k^i &= s\mathbf{Q}\boldsymbol{\nu}_k^i \\ &= s\mathbf{Q}(\boldsymbol{\nu}_{\text{lin}_k^i} + c_s r_{3k}^i \|\mathbf{p}_k^i - \mathbf{p}_k^g\| \Phi_k^i) \\ &= s(\mathbf{Q}\boldsymbol{\nu}_{\text{lin}_k^i} + c_s r_{3k}^i \|\mathbf{p}_k^i - \mathbf{p}_k^g\| \mathbf{Q}\Phi_k^i) \\ &= s(\mathbf{Q}\boldsymbol{\nu}_{\text{lin}_k^i} + c_s r_{3k}^i \|\mathbf{p}_k^i - \mathbf{p}_k^g\| \Psi_k^i), \end{aligned} \quad (3.13)$$

with $\Psi_k^i = \mathbf{Q}\Phi_k^i$ a normalized random vector with a standard normal distribution. A comparison of (3.12) and (3.13) reveals that the formulation is *strictly* rotationally invariant only if $\Psi_k^i = \Phi_k^i$. This is in general not true for uniquely generated random vectors, but does hold in an averaged stochastic sense, since both vectors are normalized random vectors with a standard normal distribution. This implies that the probability density function is merely rotated between the two reference frames and remains unchanged. Hence our introduction of the terminology that the stochastic vectors are rotationally invariant in a *strictly stochastic* sense.

The position update rule for \mathbf{x}_{k+1}^i is

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + w\mathbf{v}_k^i + \boldsymbol{\nu}_k^i, \quad (3.14)$$

and for $\hat{\mathbf{x}}_{k+1}^i$, it is

$$\begin{aligned} \hat{\mathbf{x}}_{k+1}^i &= \hat{\mathbf{x}}_k^i + w\hat{\mathbf{v}}_k^i + \hat{\boldsymbol{\nu}}_k^i \\ &= s\mathbf{Q}\mathbf{x}_k^i + \mathbf{t} + s\mathbf{Q}w\mathbf{v}_k^i + s\mathbf{Q}\boldsymbol{\nu}_k^i \\ &= s\mathbf{Q}(\mathbf{x}_k^i + w\mathbf{v}_k^i + \boldsymbol{\nu}_k^i) + \mathbf{t} \\ &= s\mathbf{Q}\mathbf{x}_{k+1}^i + \mathbf{t}. \end{aligned} \quad (3.15)$$

The construction of position update rule of the velocity update rule given in (3.15) satisfies the transformation rule given in (2.2), for all $\mathbf{Q} \in \text{Orth}^+$. The SRI velocity update rule is strictly scale, translation and rotation invariant, albeit that rotational invariance hold in a stochastic sense only.

3.3.1 Pseudocode

The pseudocode for the SRI PSO algorithm is given in Algorithm 1.

Algorithm 1: SRI PSO algorithm

```

1 Set:  $p, n, w, c_1, c_2, c_s, k_{max}$ 
2 Initialize:  $k = 0$ 
3 for  $i = 1, p$  do
4   Initialize:  $\mathbf{x}_0^i, \mathbf{v}_0^i$ 
5    $\mathbf{p}_0^i \leftarrow \mathbf{x}_0^i$ 
6 end
7  $\mathbf{p}_0^g \leftarrow \arg \min_i f(\mathbf{p}_0^i)$ 
8 repeat
9   for  $i = 1, p$  do
10    Determine:  $\|\mathbf{p}_k^i - \mathbf{p}_k^g\|$ 
11    update  $\nu_k^i$  using (3.10)
12     $\mathbf{v}_{k+1}^i \leftarrow w\mathbf{v}_k^i + \nu_k^i$ 
13     $\mathbf{x}_{k+1}^i \leftarrow \mathbf{x}_k^i + \mathbf{v}_{k+1}^i$ 
14    if  $f(\mathbf{x}_{k+1}^i) < f(\mathbf{p}_k^i)$  then
15       $\mathbf{p}_{k+1}^i \leftarrow \mathbf{x}_{k+1}^i$ 
16    else
17       $\mathbf{p}_{k+1}^i \leftarrow \mathbf{p}_k^i$ 
18    end
19  end
20   $\mathbf{p}_{k+1}^g \leftarrow \arg \min_i f(\mathbf{p}_{k+1}^i)$ 
21   $k \leftarrow k + 1$ 
22 until  $k = k_{max}$ 
23 end

```

3.4 Numerical experiments

We now compare the rotational invariance of the two rotationally invariant velocity update rules of the PSO we have considered. Again we use the approach used by Wilke *et al.* [1]. We use a popular test set in the unrotated reference frame $f(\mathbf{x})$, and an arbitrary rotated reference frame $f(\mathbf{Q}\mathbf{x})$ [17]. Here, \mathbf{Q} is a random, proper orthogonal transformation matrix, constructed as in [7]. (\mathbf{Q} is not to be confused with \mathbf{S} , introduced in previous sections.) The transformation matrix results in a pure rotation of each test function.

The aim is *not* an exhaustive determination of optimal algorithmic parameters, but a study of the relative performance of algorithm SRI as compared to algorithm DRI. We do so without confusing the issue using additional heuristics such as maximum velocity restriction, position restriction, craziness or dynamic inertia reduction, etc. Clearly however, incorporation of these heuristics (in an invariant manner) into algorithms SRI and DRI is deserved of future attention.

Rather arbitrarily, we select $c_1 = c_2 = 2$, a swarm size of $p = 20$ particles, initial velocities $\mathbf{v}_0^i = \mathbf{0}$, and a simple synchronous updating scheme [18]. For the SRI and DRI PSOs we select $c_s = 0.1$ and $\tau = 3$ respectively. The initial positions \mathbf{x}_0^i are generated randomly within the entire search space. Real variables are implemented using *double-precision floating-point* arithmetic according

to the *IEEE Standard for Binary Floating-Point Arithmetic*, commonly referred to as ‘IEEE 754’. We study the effect of the inertia constant w . To do this, we average the performance over 100 runs (each run terminates after 10000 iterations), with w kept constant. This procedure is repeated for w between 0 and 1, in increments of 0.1. For each of the 100 independent runs, a new random rotation matrix \mathbf{Q} is constructed, to ensure that there is no bias toward any particular reference frame.

We use the following six test functions that are popular in PSO research:

1. The Rosenbrock function (unimodal), n even

$$f_1(\mathbf{x}) = \sum_{i=1}^{\frac{n}{2}} \left(100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2 \right)$$

2. The Quadric function (unimodal)

$$f_2(\mathbf{x}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2$$

3. The Ackley function (multimodal)

$$f_3(\mathbf{x}) = -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right) + 20 + e$$

4. The generalized Rastrigin function (multimodal)

$$f_4(\mathbf{x}) = \sum_{i=1}^n \left(x_i^2 - 10 \cos(2\pi x_i) + 10 \right)$$

5. The generalized Griewank function (multimodal)

$$f_5(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos \left(\frac{x_i}{\sqrt{i}} \right) + 1$$

6. The Dixon-Price function (unimodal)

$$f_6(\mathbf{x}) = \sum_{i=2}^n i(2x_i^2 - x_{i-1})^2$$

All problems have dimensions $n = 30$, and each component of the initial positions are limited between ± 2.048 , ± 100 , ± 30 , ± 5.12 , ± 600 and ± 30 for the respective problems. The global minimum for all test problems is $f(\mathbf{x}^*) = 0$. Except for the Rosenbrock function f_1 , which has solution vector $\mathbf{x}^* = [1, 1, \dots, 1]^T$, the other test problems have the solution vector $\mathbf{x}^* = [0, 0, \dots, 0]^T$.

Depicted in Figures 3.1 through 3.6, are the mean objective function values after 10000 iterations averaged over 100 runs for both the unrotated and rotated functions, for the six test problems under consideration. The *rotational invariance* of the SRI and the *stochastic rotational invariance* of the DRI PSO are evident from the figures.

For the sake of clarity, an overview of the performance of the linear, the classical, the SRI and the DRI PSO algorithms is given in Table 3.1. The table summarizes the best function values obtained, together with the inertia factor at which the best function value is obtained after 10000 iterations. Results for both the unrotated and rotated test functions are given.

There is a significantly improved performance for almost all the rotated test functions with the SRI and DRI PSOs, which in turn perform comparable, possibly with algorithm DRI marginally superior for the arbitrary setting used in the algorithms.

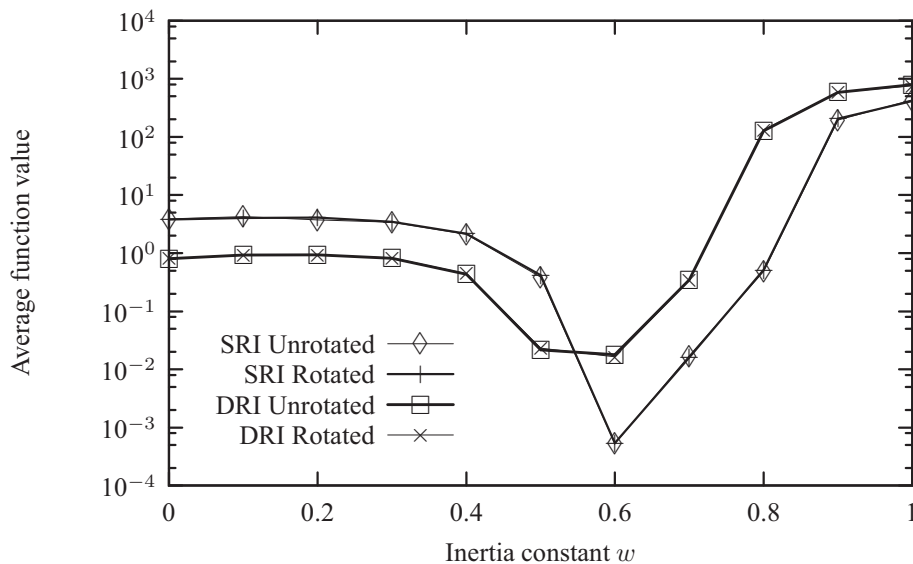


Figure 3.1: Average function values obtained after 10^4 iterations averaged over 100 runs for the rotated and unrotated Rosenbrock test function f_1

3.4.1 Scalability of algorithms SRI and DRI

To investigate the scalability of the computational effort required for algorithms SRI and DRI, we depict the required computational effort (in CPU seconds) to complete 50 runs of f_1 at $w = 0.5$ for a range of increasing dimensionality. The results are depicted in Figure 3.7.

The high computational demands for algorithm DRI are due to the computational expense asso-

Table 3.1: Constant inertia factor w at which the best average objective function value is obtained for the unrotated $f_{\text{ave}}^{\text{best}}|_{\text{U}}$ and rotated $f_{\text{ave}}^{\text{best}}|_{\text{R}}$ test functions

Function	w	$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
Linear velocity update rule			
f_1	0.8	55.8	54.0
f_2	0.8	4360	4520
f_3	0.8	11.6	12.0
f_4	0.8	151	157
f_5	0.8	31.6	31.9
f_6	0.8	4.28×10^5	4.44×10^5
Classical velocity update rule			
f_1	0.4	1.68	13.8
f_2	0.4	9.93×10^{-10}	4.43×10^{-8}
f_3	0.6	1.02×10^{-14}	2.34
f_4	0.6	41.2	133
f_5	0.6	1.75×10^{-2}	1.51×10^{-2}
f_6	0.5	5.26×10^{-1}	14.3
DRI velocity update rule			
f_1	0.6	1.79×10^{-2}	1.63×10^{-2}
f_2	0.5	6.64×10^{-43}	1.15×10^{-43}
f_3	0.7	3.52	3.37
f_4	0.6	74.2	73.6
f_5	0.3	1.02×10^{-2}	8.79×10^{-3}
f_6	0.6	3.77×10^{-1}	6.42×10^{-1}
SRI velocity update rule			
f_1	0.6	5.25×10^{-4}	5.41×10^{-4}
f_2	0.6	1.44×10^{-26}	1.83×10^{-26}
f_3	0.8	8.79	9.05
f_4	0.7	65.2	62.5
f_5	0.2	7.33×10^{-3}	9.86×10^{-3}
f_6	0.7	2.42×10^{-1}	3.37×10^{-1}

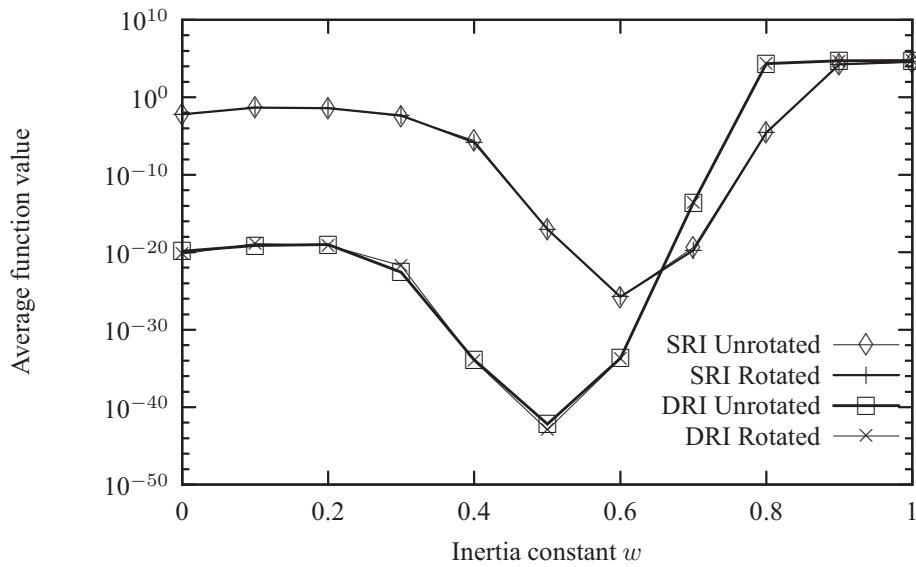


Figure 3.2: Average function values obtained after 10^4 iterations averaged over 100 runs for the rotated and unrotated Quadric test function f_2

ciated with constructing the skew matrix \mathbf{W} in (3.9), which requires the generation of $(n^2 - n)$ *uniform random numbers* per particle. Generating the random vector Φ_k^i in (3.10) for algorithm SRI requires n *normally distributed random numbers* per particle. The computational scaling associated with the two algorithms are of course dependent on the algorithms used to generate the different types of random numbers.

3.5 Summary

We have proposed a PSO formulation that is both *strictly stochastically* rotational invariant and diverse, by simply adding an additional term to the linear velocity update rule of the standard linear PSO. Numerically, the proposed formulation seems to compare well with algorithm DRI, a similarly diverse, and rotationally invariant PSO, albeit that the latter is invariant in a *stochastic* sense only.

It is noted that we made no attempt whatsoever to tune the scaling parameter c_s . Compared to algorithm DRI, the formulation proposed herein is an attractive alternative due to the lower computational expense of $\mathcal{O}(n)$ as compared to $\mathcal{O}(n^2)$, with respect to random number generation, albeit that these are different types of random numbers.

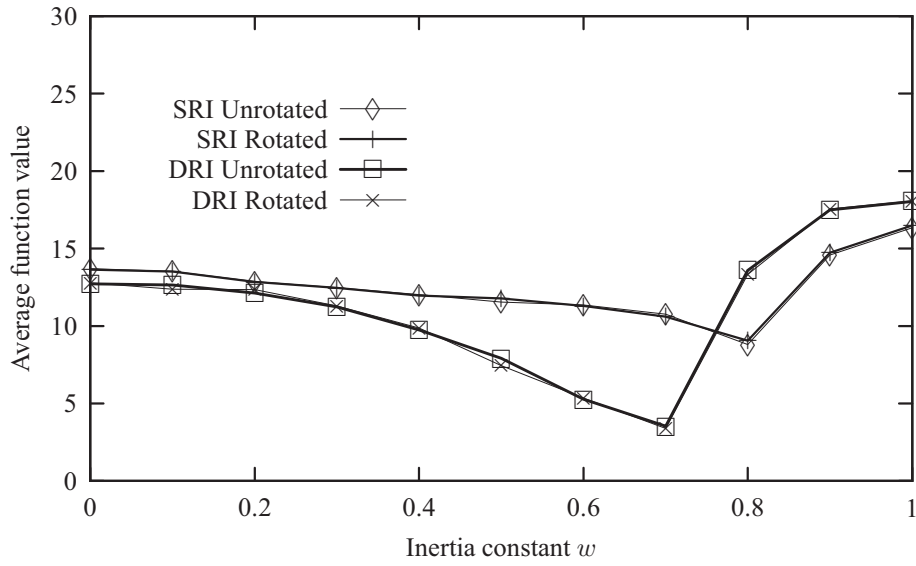


Figure 3.3: Average function values obtained after 10^4 iterations averaged over 100 runs for the rotated and unrotated Ackley test function f_3

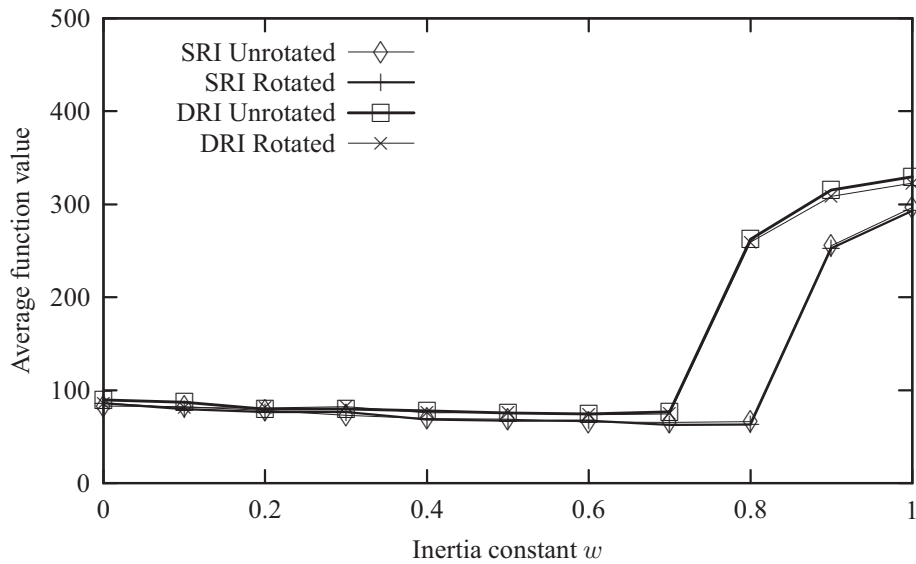


Figure 3.4: Average function values obtained after 10^4 iterations averaged over 100 runs for the rotated and unrotated Rastrigin test function f_4

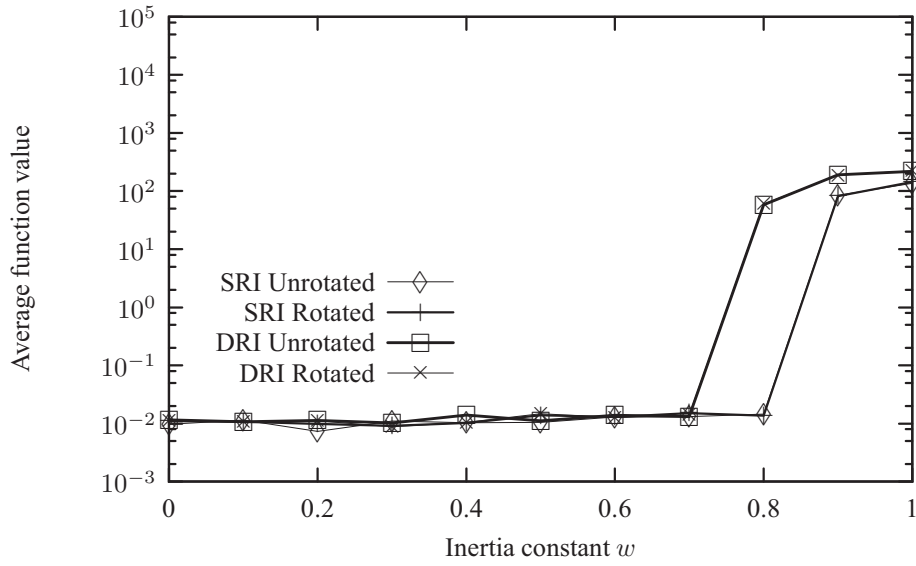


Figure 3.5: Average function values obtained after 10^4 iterations averaged over 100 runs for the rotated and unrotated Griewank test function f_5

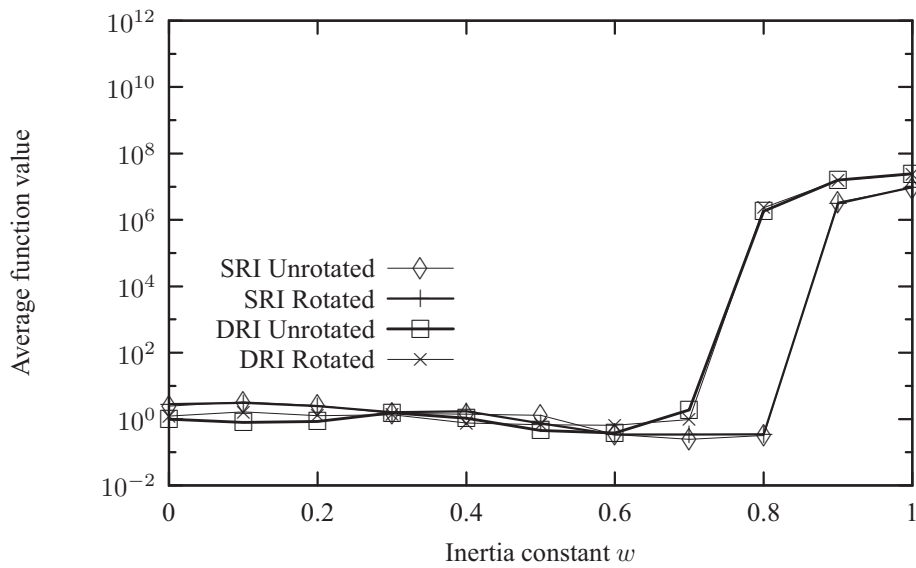


Figure 3.6: Average function values obtained after 10^4 iterations averaged over 100 runs for the rotated and unrotated Dixon-Price test function f_6

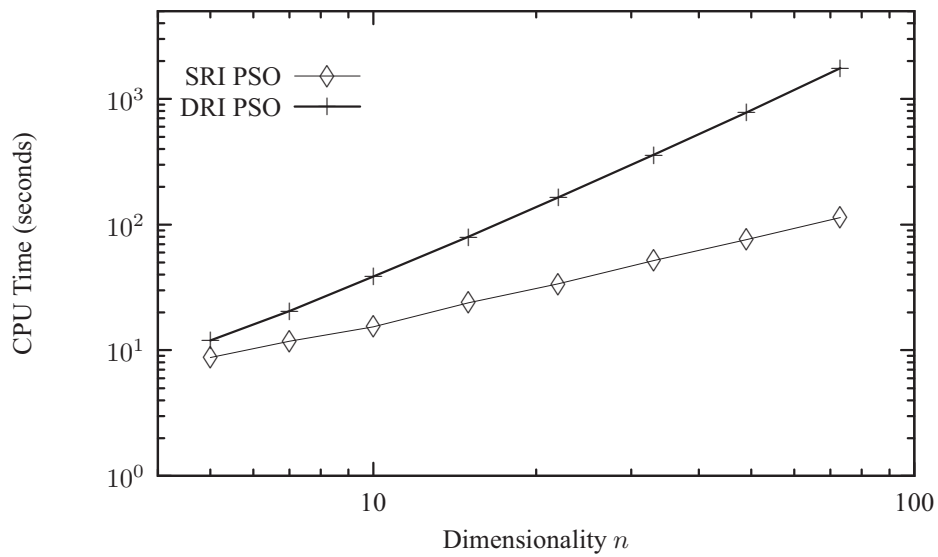


Figure 3.7: CPU time of the DRI and SRI algorithms as a function of dimensionality

Chapter 4

On the rotational (in)variance of the DE algorithm

The work presented in this chapter is reproduced from a paper titled “On the rotational variance of the DE algorithm” [19]. The paper is co-authored by Dr Daniel N. Wilke of the Mechanical and Aeronautical Engineering, University of Pretoria, Pretoria, South Africa, Prof. Albert A. Groenwold of the Department of Mechanical Engineering at the University of Stellenbosch, Stellenbosch, South Africa, and Dr S. Kok of Advanced Mathematical Modelling, CSIR Modelling and Digital Science, Pretoria, South Africa.

Previously Takahama and Sakai [20] proposed a rotationally invariant DE algorithm. In this chapter we introduce two new rotationally *invariant* DE algorithms, the intention again not being the contribution of yet another competitive and/or superior DE variant, but rather to present additional rotational invariant formulations to the existing formulations, in the hope that this will stimulate additional future contributions. Our formulations differ from that of Takahama and Sakai in that we modify the crossover operation to be rotationally *invariant* and diverse, whereas they construct an orthogonal base using Gramm-Schmidt orthogonalization, whereafter they perform a *variant* crossover operation in said orthogonal base. The orthogonal base may be viewed as an optimal frame for crossover.

For the two new algorithms we show competing performance against the standard *variant* DE algorithm on a test set in the unrotated and arbitrarily rotated reference frames.

This chapter is constructed as follows: we present the differential evolution (DE) algorithm used in this study in Section 4.1. In Section 4.2.1 we give an overview of the algorithm previously proposed by Takahama and Sakai. We present our first invariant differential evolution algorithm in Section 4.2.2 and the second in Section 4.2.3. In Section 4.3, we present numerical results for some popular test functions, in both the unrotated and rotated reference frames. Finally, we summarize the findings of this chapter in Section 4.4.

4.1 Differential evolution algorithm

We now consider some of the operations in DE algorithms and study the frame (in)variance of these operations.

The very first step of a DE algorithm is of course initialization, already discussed in Section 2.4. That is followed by differential mutation, crossover and selection; discussed in Sections 4.1.1, 4.1.2 and 4.1.3 respectively. The process continues until the maximum number of generations k_{max} is reached.

4.1.1 Differential mutation

Next, we consider three popular forms of differential mutation namely constant, dither and jitter mutation [21]. The theoretical analysis of constant and dither mutation is equivalent, we therefore consider them together and do a separate analysis for jitter mutation.

Constant and dither mutation

The population at the k^{th} generation is given by \mathbf{x}_k^i , $i = 1, \dots, p$ and $\mathbf{x} \in \mathbb{R}^n$. For every generation k and each target vector \mathbf{x}_k^i , $i = 1, \dots, p$, a mutant vector \mathbf{v}_{k+1}^i is generated using

$$\mathbf{v}_{k+1}^i = \mathbf{x}_k^{r_1} + F(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}), \quad (4.1)$$

with $F \in [0, 2]$ a real scalar; and $r_1, r_2, r_3 \in \{1, 2, \dots, p\}$ uniform random integers, such that $r_1 \neq r_2 \neq r_3 \neq i$.

An intuitive proof follows readily by considering that the magnitude of a vector is a frame invariant quantity. Therefore constant and dither mutation, which simply scale the length of the vectors, are independent of the coordinate system.

Formally, let us consider the transformed stochastic vector

$$\begin{aligned} \hat{\mathbf{v}}_{k+1}^i &= \hat{\mathbf{x}}_k^{r_1} + F(\hat{\mathbf{x}}_k^{r_2} - \hat{\mathbf{x}}_k^{r_3}) \\ &= (s\mathbf{Q}\mathbf{x}_k^{r_1} + \mathbf{t}) + F((s\mathbf{Q}\mathbf{x}_k^{r_2} + \mathbf{t}) - (s\mathbf{Q}\mathbf{x}_k^{r_3} + \mathbf{t})) \\ &= s\mathbf{Q}\mathbf{x}_k^{r_1} + \mathbf{t} + s\mathbf{Q}F(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}) \\ &= s\mathbf{Q}(\mathbf{x}_k^{r_1} + F(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3})) + \mathbf{t} \\ &= s\mathbf{Q}\mathbf{v}_{k+1}^i + \mathbf{t}. \end{aligned} \quad (4.2)$$

The mutant vector satisfy the transformation rules given in (2.2) and (2.5), for all $\mathbf{Q} \in \text{Orth}^+$. Hence, the constant and dither mutation operators velocity update rules are scale, translation and rotation invariant.

Jitter mutation

Jitter mutation makes use of a random vector \mathbf{F} where each component is a random number $\in [0, 2]$. Equivalently \mathbf{F} can be seen as a $n \times n$ matrix with independent random values on the

diagonal of the matrix while off diagonal entries are zero. We use this to conduct our analysis. The mutant vector is then given by

$$\mathbf{v}_{k+1}^i = \mathbf{x}_k^{r_1} + \mathbf{F}(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}). \quad (4.3)$$

Similarly, the transformed stochastic vector is

$$\begin{aligned} \hat{\mathbf{v}}_{k+1}^i &= \hat{\mathbf{x}}_k^{r_1} + \hat{\mathbf{F}}(\hat{\mathbf{x}}_k^{r_2} - \hat{\mathbf{x}}_k^{r_3}) \\ &= (s\mathbf{Q}\mathbf{x}_k^{r_1} + \mathbf{t}) + \hat{\mathbf{F}}((s\mathbf{Q}\mathbf{x}_k^{r_2} + \mathbf{t}) - (s\mathbf{Q}\mathbf{x}_k^{r_3} + \mathbf{t})) \\ &= s\mathbf{Q}\mathbf{x}_k^{r_1} + s\hat{\mathbf{F}}\mathbf{Q}(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}) + \mathbf{t}. \end{aligned} \quad (4.4)$$

By considering the transformation rules given in (2.2) and (2.5), the required transformation is

$$\begin{aligned} \hat{\mathbf{v}}_{k+1}^i &= s\mathbf{Q}\mathbf{v}_{k+1}^i + \mathbf{t} \\ &= s\mathbf{Q}(\mathbf{x}_k^{r_1} + \mathbf{F}(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3})) + \mathbf{t} \\ &= s\mathbf{Q}\mathbf{x}_k^{r_1} + s\mathbf{Q}\mathbf{F}(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}) + \mathbf{t}. \end{aligned} \quad (4.5)$$

Comparing the actual transformation in (4.4) to the required transformation in (4.5), rotation invariance requires

$$\mathbf{Q}\mathbf{F} = \hat{\mathbf{F}}\mathbf{Q} \rightarrow \hat{\mathbf{F}} = \mathbf{Q}\mathbf{F}\mathbf{Q}^T \quad \forall \mathbf{Q} \in \text{Orth}^+. \quad (4.6)$$

Since the relation between $\hat{\mathbf{F}}$ and \mathbf{F} has to hold $\forall \mathbf{Q} \in \text{Orth}^+$, the solution to (4.6) is

$$\hat{\mathbf{F}} = \mathbf{F} = a\mathbf{I}, \quad (4.7)$$

with a any real scalar value and \mathbf{I} the identity matrix. Thus, for strict invariance jitter mutation has to reduce to dither mutation.

Since jitter mutation scales on a components basis, additional diversity is introduced into the DE algorithm. Not only are the difference vectors' length changed, but also the direction of the vectors.

4.1.2 Crossover

Crossover is used to combine the target vector \mathbf{x}_k^i and the mutant vector \mathbf{v}_{k+1}^i to create the trial vector \mathbf{u}_{k+1}^i . We will look at two ways for doing this, namely whole arithmetic crossover and standard crossover.

Whole arithmetic crossover

Whole arithmetic crossover combines the target vector \mathbf{x}_k^i and the mutant vector \mathbf{v}_{k+1}^i to form the trial vector \mathbf{u}_{k+1}^i by applying

$$\mathbf{u}_{k+1}^i = (1 + r_b)\mathbf{v}_{k+1}^i + r_b\mathbf{x}_k^i, \quad (4.8)$$

with r_b a uniform random number $\in [0, 1]$ [22].

The transformed trial vector, $\hat{\mathbf{u}}_{k+1}^i$, is given by

$$\begin{aligned} \hat{\mathbf{u}}_{k+1}^i &= (1 + r_b)\hat{\mathbf{v}}_{k+1}^i + r_b\hat{\mathbf{x}}_k^i + \mathbf{t} \\ &= (1 + r_b)s\mathbf{Q}\mathbf{v}_{k+1}^i + r_bs\mathbf{Q}\mathbf{x}_k^i + \mathbf{t} \\ &= s\mathbf{Q}[(1 + r_b)\mathbf{v}_{k+1}^i + r_b\mathbf{x}_k^i] + \mathbf{t} \\ &= s\mathbf{Q}\mathbf{u}_{k+1}^i + \mathbf{t}. \end{aligned} \quad (4.9)$$

Hence, the whole arithmetic crossover operation satisfies the transformation rules (2.2) and (2.5), for all $Q \in \text{Orth}^+$ and is therefore scale, translation and rotation invariant.

Although this operation might be used in genetic algorithms it is rarely, if ever, used in DE algorithms. This is due to poor performance, as numerically illustrated later in this study. The poor performance is due to lack of diversity of the method.

Standard crossover

Standard crossover combines the target vector \mathbf{x}_k^i and the mutant vector \mathbf{v}_{k+1}^i to form the trial vector \mathbf{u}_{k+1}^i by applying the following crossover rule for each dimension $j = 1, 2, \dots, n$:

$$u_{k+1}^i(j) = \begin{cases} v_{k+1}^i(j) & \text{if } \zeta(j) \leq C_r \text{ or } j = r \\ x_k^i(j) & \text{if } \zeta(j) > C_r \text{ and } j \neq r, \end{cases} \quad (4.10)$$

with $\zeta \in [0, 1]$ a vector with each component a uniform random number and $C_r \in [0, 1]$ a real scalar value. r is a uniform randomly chosen integer value $\in \{1, 2, \dots, p\}$.

Equivalently, the standard crossover operations can be viewed as

$$\mathbf{u}_{k+1}^i = \mathbf{B}_{k+1}^i \mathbf{v}_{k+1}^i + \tilde{\mathbf{B}}_{k+1}^i \mathbf{x}_k^i, \quad (4.11)$$

with \mathbf{B}_{k+1}^i a diagonal matrix with 1 on the diagonal for $\zeta(j) \leq C_r$ or $j = r$, otherwise 0; and $\tilde{\mathbf{B}}_{k+1}^i$ the conjugate diagonal of \mathbf{B}_{k+1}^i . That is $\tilde{\mathbf{B}}_{k+1}^i + \mathbf{B}_{k+1}^i = \mathbf{I}$, with \mathbf{I} the identity matrix. The transformed trial vector is then

$$\begin{aligned} \hat{\mathbf{u}}_{k+1}^i &= \hat{\mathbf{B}}_{k+1}^i \hat{\mathbf{v}}_{k+1}^i + \hat{\tilde{\mathbf{B}}}_{k+1}^i \hat{\mathbf{x}}_k^i \\ &= \hat{\mathbf{B}}_{k+1}^i (s\mathbf{Q}\mathbf{v}_{k+1}^i + \mathbf{t}) + \hat{\tilde{\mathbf{B}}}_{k+1}^i (s\mathbf{Q}\mathbf{x}_k^i + \mathbf{t}) \\ &= s\hat{\mathbf{B}}_{k+1}^i \mathbf{Q}\mathbf{v}_{k+1}^i + \hat{\tilde{\mathbf{B}}}_{k+1}^i \mathbf{t} + s\hat{\tilde{\mathbf{B}}}_{k+1}^i \mathbf{Q}\mathbf{x}_k^i + \hat{\mathbf{B}}_{k+1}^i \mathbf{t} \\ &= s\hat{\mathbf{B}}_{k+1}^i \mathbf{Q}\mathbf{v}_{k+1}^i + s\hat{\tilde{\mathbf{B}}}_{k+1}^i \mathbf{Q}\mathbf{x}_k^i + (\hat{\mathbf{B}}_{k+1}^i + \hat{\tilde{\mathbf{B}}}_{k+1}^i) \mathbf{t} \\ &= s\hat{\mathbf{B}}_{k+1}^i \mathbf{Q}\mathbf{v}_{k+1}^i + s\hat{\tilde{\mathbf{B}}}_{k+1}^i \mathbf{Q}\mathbf{x}_k^i + \mathbf{I}\mathbf{t} \\ &= s(\hat{\mathbf{B}}_{k+1}^i \mathbf{Q}\mathbf{v}_{k+1}^i + \hat{\tilde{\mathbf{B}}}_{k+1}^i \mathbf{Q}\mathbf{x}_k^i) + \mathbf{t}. \end{aligned} \quad (4.12)$$

By considering the transformation rules given in (2.2) and (2.5), the required transformation for scale, translation and rotational invariance is

$$\begin{aligned} \hat{\mathbf{u}}_{k+1}^i &= s\mathbf{Q}\mathbf{u}_{k+1}^i + \mathbf{t} \\ &= s\mathbf{Q}(\mathbf{B}_{k+1}^i \mathbf{v}_{k+1}^i + \tilde{\mathbf{B}}_{k+1}^i \mathbf{x}_{k+1}^i) + \mathbf{t} \\ &= s(\mathbf{Q}\mathbf{B}_{k+1}^i \mathbf{v}_{k+1}^i + \mathbf{Q}\tilde{\mathbf{B}}_{k+1}^i \mathbf{x}_{k+1}^i) + \mathbf{t}. \end{aligned} \quad (4.13)$$

Comparing the actual transformation in (4.12) to the required transformation in (4.13), rotation invariance requires

$$\mathbf{Q}\mathbf{B}_{k+1}^i = \hat{\mathbf{B}}_{k+1}^i \mathbf{Q} \rightarrow \hat{\mathbf{B}}_{k+1}^i = \mathbf{Q}\mathbf{B}_{k+1}^i \mathbf{Q}^T \quad \forall \mathbf{Q} \in \text{Orth}^+. \quad (4.14)$$

Since the relation between $\hat{\mathbf{B}}_{k+1}^i$ and \mathbf{B}_{k+1}^i has to hold $\forall \mathbf{Q} \in \text{Orth}^+$, the solution to (4.14) is

$$\hat{\mathbf{B}}_{k+1}^i = \mathbf{B}_{k+1}^i = b\mathbf{I}, \quad (4.15)$$

with b any real scalar value. Similarly

$$\tilde{\mathbf{B}}_{k+1}^i = \tilde{\mathbf{B}}_{k+1}^i = (1 - b)\mathbf{I}. \quad (4.16)$$

Therefore, for strict invariance the standard crossover operations have to reduce to whole arithmetic crossover, with a constant term b instead of the random number, r_b .

From (4.12) and (4.13) it follows that the standard crossover operator is scale and translation invariant. It is however, evident that rotational invariance is sacrificed.

4.1.3 Selection

The new generation ($k + 1$) is selected by comparing the function value of the trial vector $f(\mathbf{u}_{k+1}^i)$ with it's corresponding target vector $f(\mathbf{x}_k^i)$. The vector with the smaller function value is selected. Selection is by definition scale and frame invariant as vectors are selected without performing any operations on them.

4.2 Rotationally invariant differential evolution algorithms

Using constant or dither mutation, the generation of the mutant vector \mathbf{v}_{k+1}^i in the DE algorithm is rotationally invariant. The lack of rotational invariance of the classic *DE/rand/1/bin* algorithm is a result of the commonly used standard crossover operator that independently generates the trial vector's components. This operator is also however, responsible for diversity in the classic DE algorithm, which is a direct result of the component operation to generate the trial vector \mathbf{u}_{k+1}^i . On the other hand, using constant or dither mutation and whole arithmetic crossover we are able to present a rotationally invariant algorithm. This, however, results in a poor performing algorithm due to the lack of diversity, as illustrated in Section 4.3.

Hence we only need to modify the crossover operation such that it is *rotationally invariant* and *diverse* in order to compute the trial vector \mathbf{u}_{k+1}^i .

In this section we will first look at a previously proposed rotationally invariant formulation, then we will propose a diverse DE algorithm, which is rotationally invariant in a *stochastic sense* only and a diverse DE algorithm that is *strictly rotationally invariant in a stochastic sense*. Stochastic rotational invariance will be understood to imply that the algorithm's performance is unaffected in a stochastic sense i.e. over a large number of runs. On the other hand, strict rotational invariance in a stochastic sense will imply that the probability density functions between references frames are only related by a rotation between rotated reference frames. It is important to emphasize that the point-wise invariance between sample points generated from these probability density functions does not hold although the probability density functions are invariant.

4.2.1 Rotationally invariant crossover using Gram-Schmidt orthogonalization

Previously, Takahama and Sakai [20] proposed a rotationally invariant crossover scheme. To understand their approach, we first express the standard crossover operation in terms of coordinate vectors as

$$\begin{aligned} \mathbf{y} &= \mathbf{v}_{k+1} - \mathbf{x}_{k+1}^i \\ \mathbf{u}_{k+1} &= \mathbf{x}^i + \sum_{k \in K} (\mathbf{y}, \mathbf{e}_k) \mathbf{e}_k, \end{aligned} \quad (4.17)$$

with $(\mathbf{y}, \mathbf{e}_k)$ the inner product of \mathbf{y} and \mathbf{e}_k , K the set of indexes of selected elements and \mathbf{e}_k a unit vector with the k -th element one and the other elements zero.

Using Gram-Schmidt orthogonalization Takahama and Sakai construct an orthogonal base $\mathbf{b}_1, \dots, \mathbf{b}_n$ using n of the target vectors \mathbf{x}_{k+1}^i for each iteration. This of course requires that $p \geq n$. The new base of \mathbf{b}_k 's is used in the crossover operation given in (4.17) instead of the \mathbf{e}_k 's. Thus,

$$\begin{aligned} \mathbf{y} &= \mathbf{v}_{k+1} - \mathbf{x}_{k+1}^i \\ \mathbf{u}_{k+1} &= \mathbf{x}^i + \sum_{k \in K} (\mathbf{y}, \mathbf{b}_k) \mathbf{b}_k. \end{aligned} \quad (4.18)$$

For this algorithm, it is noted that the construction of the orthogonal base requires significant computational effort, $\mathcal{O}(2n^3)$.

4.2.2 Perturbation rotation invariant crossover (PRICO)

For the perturbation rotationally invariant crossover (PRICO) we propose herein, we compute the trial vector by perturbing a linear uniform stochastic combination between \mathbf{x}_k^i and \mathbf{v}_{k+1}^i , similar to whole arithmetic crossover. This is done by first computing a difference vector \mathbf{d}_{k+1}^i that defines the linear combination

$$\mathbf{d}_{k+1}^i = \mathbf{v}_{k+1}^i - \mathbf{x}_k^i. \quad (4.19)$$

We then perturb the direction cosines of \mathbf{d}_{k+1}^i by first computing the unit vector \mathbf{e}_k^r and the norm $\|\mathbf{d}_{k+1}^i\|$ of the difference vector \mathbf{d}_{k+1}^i . Each component j of \mathbf{e}_k^r is then perturbed by applying

$$\epsilon_k^r(j) = \cos\left(\frac{\mu\pi}{180}(2\varrho_k^r(j) - 1) + \arccos(e_k^r(j))\right), \quad j = 1, 2, \dots, n, \quad (4.20)$$

with $\varrho_k^r(j)$ a uniform random scalar between 0 and 1. μ defines the limits of the direction cosine perturbations.

Finally, we compute the trial vector \mathbf{u}_{k+1}^i as

$$\mathbf{u}_{k+1}^i = \mathbf{x}_k^i + 2r\|\mathbf{d}_{k+1}^i\|\boldsymbol{\epsilon}_k^r, \quad (4.21)$$

with $r \in [0, 1]$ a uniform random number.

The perturbation of the direction cosines of \mathbf{d}_{k+1}^i are introduced in order to introduce diversity into the crossover operator. Although the direction cosine perturbations are not *strictly* rotationally invariant, it is sufficiently invariant in a *stochastic* sense for small perturbations ($\mu < 5$), as we will illustrate in Section 4.3.

Algorithm 2: PRICO algorithm

```

1 Set:  $p, n, \mu, F, k_{max}$ 
2 Initialize:  $k = 0$ 
3 for  $i = 1, p$  do
4   Initialize:  $\mathbf{x}_0^i$ 
5 end
6 repeat
7   for  $i = 1, p$  do
8     Generate:  $r_1, r_2, r_3$ 
9      $\mathbf{v}_{k+1}^i \leftarrow \mathbf{x}_k^{r_1} + F(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3})$ 
10     $\mathbf{d}_{k+1}^i \leftarrow \mathbf{v}_{k+1}^i - \mathbf{x}_k^i$ 
11    Determine:  $\|\mathbf{d}_{k+1}^i\|$ 
12     $\mathbf{e}_k^r \leftarrow \mathbf{d}_{k+1}^i / \|\mathbf{d}_{k+1}^i\|$ 
13    for  $j = 1, n$  do
14      update  $\epsilon_k^r(j)$  using (4.20)
15    end
16     $\mathbf{u}_{k+1}^i \leftarrow \mathbf{x}_k^i + 2r\|\mathbf{d}_{k+1}^i\|\mathbf{e}_k^r$ 
17    if  $f(\mathbf{u}_{k+1}^i) < f(\mathbf{x}_k^i)$  then
18       $\mathbf{x}_{k+1}^i \leftarrow \mathbf{u}_{k+1}^i$ 
19    else
20       $\mathbf{x}_{k+1}^i \leftarrow \mathbf{x}_k^i$ 
21    end
22  end
23   $k \leftarrow k + 1$ 
24 until  $k = k_{max}$ 
25 end

```

Pseudocode

The pseudocode for the PRICO algorithm is given in Algorithm 2.

4.2.3 Strictly (stochastic) rotationally invariant crossover (SSRICO)

Now, we propose a directionally diverse crossover operation, denoted SSRICO. The proposed formulation adds a term to the trail vector \mathbf{u}_{k+1}^i based on whole arithmetic crossover operation, in order to introduce diversity. The new term is simply a scaled, normalized random vector with a *standard normal distribution*. This results in a point sampled *uniformly* on the surface of a n -dimensional unit sphere [16].

The new trial vector \mathbf{u}_{k+1}^i is then formulated as

$$\mathbf{u}_{k+1}^i = \mathbf{u}_{k+1}^{*i} + c_s r_k^i \|\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}\| \Phi_k^i, \quad (4.22)$$

with Φ_k^i the normalized vector with a standard normal distribution, r_k^i a random scalar, c_s a scaling

factor and \mathbf{u}_{k+1}^{*i} the trail vector we get using whole arithmetic crossover, as given by (4.8). We introduce $\|\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}\|$ to ensure the additional term converges with the rest of the formulation.

The transformed trail vector is then given by

$$\begin{aligned}
\hat{\mathbf{u}}_{k+1}^i &= \hat{\mathbf{u}}_{k+1}^{*i} + c_s r_k^i \|\hat{\mathbf{x}}_k^{r_2} - \hat{\mathbf{x}}_k^{r_3}\| \Phi_k^i \\
&= s\mathbf{Q}\mathbf{u}_{k+1}^{*i} + \mathbf{t} + c_s r_k^i \|s\mathbf{Q}\mathbf{x}_k^{r_2} + \mathbf{t} - s\mathbf{Q}\mathbf{x}_k^{r_3} - \mathbf{t}\| \Phi_k^i \\
&= s\mathbf{Q}\mathbf{u}_{k+1}^{*i} + \mathbf{t} + s c_s r_k^i \|\mathbf{Q}(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3})\| \Phi_k^i \\
&= s\mathbf{Q}\mathbf{u}_{k+1}^{*i} + \mathbf{t} + s c_s r_k^i \|\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}\| \Phi_k^i.
\end{aligned} \tag{4.23}$$

The required transformation is

$$\begin{aligned}
\hat{\mathbf{u}}_{k+1}^i &= s\mathbf{Q}(\mathbf{u}_{k+1}^{*i} + c_s r_k^i \|\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}\| \Phi_k^i) + \mathbf{t} \\
&= s\mathbf{Q}\mathbf{u}_{k+1}^{*i} + s c_s r_k^i \|\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}\| \mathbf{Q}\Phi_k^i + \mathbf{t} \\
&= s\mathbf{Q}\mathbf{u}_{k+1}^{*i} + s c_s r_k^i \|\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}\| \Psi_k^i + \mathbf{t},
\end{aligned} \tag{4.24}$$

with $\Psi_k^i = \mathbf{Q}\Phi_k^i$ a normalized random vector with a standard normal distribution. A comparison of (4.23) and (4.24), reveals that the formulation is *strictly* rotationally invariant only if $\Psi_k^i = \Phi_k^i$. This is in general not true for uniquely generated random vectors, but does hold in an averaged stochastic sense, since both vectors are normalized random vectors with a standard normal distribution. This implies that the probability density function is merely rotated between the two reference frames and remains unchanged. Hence our introduction of the terminology that the stochastic vectors are rotationally invariant in a *strictly stochastic* sense.

Pseudocode

The pseudocode for the SSRICO algorithm is given in Algorithm 3.

4.3 Numerical experiments

We now perform an empirical study to quantify the (lack of) rotational invariance of the various DE formulations. Again we use the approach used by Wilke *et al.* [1]. We deliberately keep the algorithms simple and do not implement additional heuristics as to not confuse the issue of rotational invariance. We will be looking at the classical DE; a DE/rand/1/arth, henceforth referred to as the arithmetic crossover DE (ACDE); the classical DE using rotational invariant crossover based on Gram-Schmidt orthogonalization referred to as GSRIDE and the two proposed rotationally invariant crossover operation, namely SSRICO and PRICO, alongside constant mutation. Real variables are implemented using double-precision floating-point arithmetic. We choose a population size of $p = 30$ and conduct the study for various parameter settings of each algorithm. Each run consists of 210000 function evaluations (7000 iterations). All results presented are averaged over 100 runs.

We vary F between 0 and 1, in increments of 0.1. In the classical DE and GSRIDE we vary C_r from 0 to 0.9 in increments of 0.1, and for the PRICO we vary μ from 0.5 to 5 in increments of 0.5. For SSRICO we set $c = 0.1$.

We use the following six test functions that are popular in the literature:

Algorithm 3: SSRICO algorithm

```

1 Set:  $p, n, c_s, F, k_{max}$ 
2 Initialize:  $k = 0$ 
3 for  $i = 1, p$  do
4   Initialize:  $\mathbf{x}_0^i$ 
5 end
6 repeat
7   for  $i = 1, p$  do
8     Generate:  $r_1, r_2, r_3$ 
9      $\mathbf{v}_{k+1}^i \leftarrow \mathbf{x}_k^{r_1} + F(\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3})$ 
10    Determine:  $\|\mathbf{x}_k^{r_2} - \mathbf{x}_k^{r_3}\|$ 
11    update  $\mathbf{u}_{k+1}^i$  using (4.22)
12    if  $f(\mathbf{u}_{k+1}^i) < f(\mathbf{x}_k^i)$  then
13       $\mathbf{x}_{k+1}^i \leftarrow \mathbf{u}_{k+1}^i$ 
14    else
15       $\mathbf{x}_{k+1}^i \leftarrow \mathbf{x}_k^i$ 
16    end
17  end
18   $k \leftarrow k + 1$ 
19 until  $k = k_{max}$ 
20 end

```

1. The Rosenbrock function (unimodal), n even

$$f_1(\mathbf{x}) = \sum_{i=1}^{\frac{n}{2}} \left(100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2 \right)$$

2. The Quadric function (unimodal)

$$f_2(\mathbf{x}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2$$

3. The Ackley function (multimodal)

$$f_3(\mathbf{x}) = -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right) + 20 + e$$

4. The generalized Rastrigin function (multimodal)

$$f_4(\mathbf{x}) = \sum_{i=1}^n \left(x_i^2 - 10 \cos(2\pi x_i) + 10 \right)$$

5. The generalized Griewank function (multimodal)

$$f_5(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

6. The Dixon-Price function (unimodal)

$$f_6(\mathbf{x}) = \sum_{i=2}^n i(2x_i^2 - x_{i-1})^2$$

All problems have dimension $n = 30$, and each component of the initial positions are limited between ± 2.048 , ± 100 , ± 30 , ± 5.12 , ± 600 and ± 30 for the respective problems. The global minimum for all test problems is $f(\mathbf{x}^*) = 0$. Except for the Rosenbrock function f_1 , which has solution vector $\mathbf{x}^* = [1, 1, \dots, 1]^T$, all other test problems have the solution vector $\mathbf{x}^* = [0, 0, \dots, 0]^T$.

Some of the functions in our test set are decomposable [7], viz. the design variables are uncoupled. This implies that once an optimal value for a given design variable is obtained, it remains optimal, independent of the other design variables. This is similar to optimizing n 1-dimensional optimization problems, instead of 1 n -dimensional coupled optimization problem. We therefore study the test set in the unrotated or decomposable reference frame $f(\mathbf{x})$, as well as in an arbitrary rotated reference frame $f(\mathbf{Q}\mathbf{x})$, in which the design variables are coupled [17]. Here, \mathbf{Q} is a random, proper orthogonal transformation matrix, constructed as in [7]. The transformation matrix results in a pure rotation of each test function. For each of the 100 independent runs, a new random rotation matrix \mathbf{Q} is constructed, to ensure that there is no bias toward any particular reference frame.

Figures 4.1 through 4.12 depict the mean objective function values after 2.1×10^5 function evaluations (or 7000 iterations) averaged over 100 runs for both the various unrotated and rotated functions at various parameter values.

For the sake of clarity, an overview of the performance of the algorithms are given in Table 4.1. The table summarizes the best function values obtained, together with the parameter settings for which the algorithm obtained the best average best function value for the *unrotated* functions after 2.1×10^5 function evaluations (7000 iterations). The corresponding average best function values for the parameter settings of the rotated function is also given.

The figures demonstrate that Table 4.1 gives an unrealistic view of the algorithm, due to the dependence on parameters. To quantify some sense of robustness, we rerun the algorithms using all parameters as random numbers with reasonable bounds. For all algorithms F is a random number that varies between 0.4 and 1. For the Standard DE and GSRIDE, C_r varies between 0.2 and 0.8. For PRICO, μ varies between 2 and 5. The results are presented in Table 4.2.

The performance of the classical DE should be considered the benchmark for this work. The *rotational variance* of the classical DE is evident from Figures 4.1(a), 4.3(a), 4.5(a), 4.7(a), 4.9(a) and 4.11(a). Here each bar represent a different value of F and within each bar the value of C_r varies. One can also see that the classical DE is sensitive to parameters.

The *rotational invariance* of the GSRIDE is evident from Figures 4.1(b), 4.3(b), 4.5(b), 4.7(b), 4.9(b) and 4.11(b). Here each bar represent a different value of F and within each bar the value of

C_r varies. As one would expect from looking at the formulation the performance of the GSRIDE is, in general, similar to that of the unrotated classical DE.

PRICO performs well compared to the other formulations. The *rotational invariance* of PRICO is evident from Figures 4.2(b), 4.4(b), 4.6(b), 4.8(b), 4.10(b) and 4.12(b). Here each bar represent a different value of F and within each bar the value of μ varies.

The *rotational invariance* of the ACDE and SSRICO is evident from Figures 4.2(a), 4.4(a), 4.6(a), 4.8(a), 4.10(a) and 4.12(a). The poor performance of the ACDE is due to the lack of diversity in the formulation. The SSRICO introduces diversity to the ACDE formulation and yields competitive results.

4.4 Summary

In this chapter we have demonstrated the lack of frame *invariance* of the classical DE, which may result in severe performance loss for rotated functions.

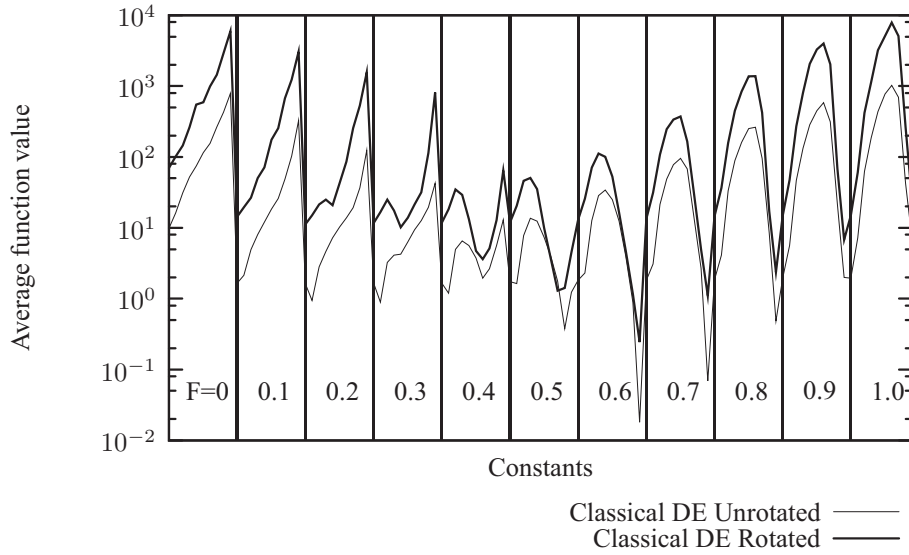
When constructing invariant DE algorithms, it is desirable to retain diversity in the algorithm. We have presented two rotationally *invariant* and diverse DE algorithms, namely the PRICO and SSRICO algorithm. We accomplish this in PRICO by merely perturbing the direction cosines of chosen difference vectors, and in SSRICO by constructing a scalable n -dimensional sphere around the trial vector.

Table 4.1: Constant parameters at which the best average objective function value is obtained for the unrotated $f_{\text{ave}}^{\text{best}}|_{\text{U}}$ and rotated $f_{\text{ave}}^{\text{best}}|_{\text{R}}$ test functions

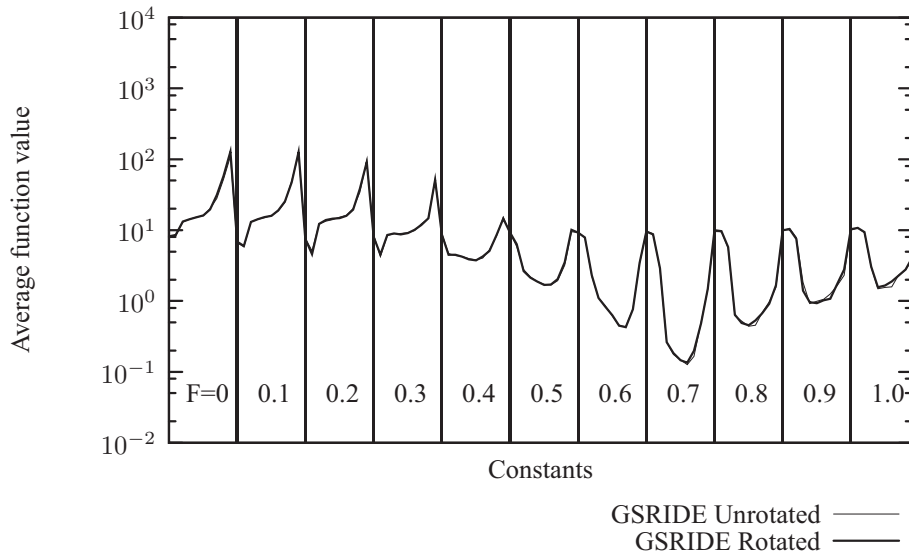
Classical DE				
Function	F	CR	$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
f_1	0.6	0.9	1.81×10^{-2}	2.44×10^{-1}
f_2	0.6	0.9	3.35×10^{-12}	5.86×10^{-11}
f_3	0.5	0.7	4.74×10^{-15}	14.8
f_4	0.5	0.1	0	98.5
f_5	0.3	0.0	1.11×10^{-17}	2.69×10^{-5}
f_6	0.3	0.0	5.15×10^{-4}	4.14
GSRIDE				
Function	F	CR	$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
f_1	0.7	0.6	1.26×10^{-1}	1.34×10^{-1}
f_2	0.6	0.3	1.02×10^{-16}	6.93×10^{-17}
f_3	1.0	0.2	7.62×10^{-1}	6.85×10^{-1}
f_4	0.3	0.5	19.8	19.9
f_5	0.5	0.2	7.28×10^{-3}	1.02×10^{-2}
f_6	1.0	0.4	5.68×10^{-2}	1.06×10^{-1}
PRICO				
Function	F	μ	$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
f_1	1.0	2.5	1.38×10^{-2}	1.30×10^{-2}
f_2	0.9	4.5	7.62×10^{-16}	7.38×10^{-16}
f_3	1.0	5.0	1.70	1.60
f_4	0.6	4.0	16.0	16.8
f_5	1.0	4.5	8.17×10^{-3}	1.04×10^{-2}
f_6	1.0	5.0	2.62×10^{-1}	3.54×10^{-1}
ACDE				
Function	F		$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
f_1	1.0		54.7	59.0
f_2	1.0		4.43×10^3	4.63×10^3
f_3	1.0		9.88	9.80
f_4	1.0		1.07×10^2	1.04×10^2
f_5	1.0		26.2	27.2
f_6	1.0		2.86×10^5	2.99×10^5
SSRICO				
Function	F		$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
f_1	1.0		3.36×10^{-1}	3.42×10^{-1}
f_2	1.0		5.05×10^{-14}	7.79×10^{-14}
f_3	1.0		9.31×10^{-3}	9.31×10^{-3}
f_4	0.4		17.7	19.0
f_5	1.0		9.12×10^{-4}	1.67×10^{-3}
f_6	0.8		1.97×10^{-1}	3.84×10^{-1}

Table 4.2: Average objective function values obtained for the unrotated $f_{ave|U}$ and rotated $f_{ave|R}$ test functions, using random parameters

Classical DE		
Function	$f_{ave U}$	$f_{ave R}$
f_1	32.3	26.8
f_2	4.30×10^3	2.73×10^3
f_3	6.412×10^{-15}	16.9
f_4	59.3	175
f_5	7.39×10^{-5}	7.39×10^{-5}
f_6	5.65×10^{-2}	6.46×10^{-1}
GSRIDE		
Function	$f_{ave U}$	$f_{ave R}$
f_1	6.43×10^{-2}	6.58×10^{-2}
f_2	9.66×10^{-9}	9.82×10^{-8}
f_3	2.00	1.92
f_4	119	124
f_5	1.27×10^{-2}	1.49×10^{-2}
f_6	5.70×10^{-1}	6.33×10^{-1}
PRICO		
Function	$f_{ave U}$	$f_{ave R}$
f_1	1.94×10^{-1}	2.00×10^{-1}
f_2	1.54×10^{-12}	2.98×10^{-12}
f_3	5.56	5.70
f_4	1.76×10^1	1.84×10^1
f_5	1.60×10^{-2}	1.47×10^{-2}
f_6	1.10	1.37
ACDE		
Function	$f_{ave U}$	$f_{ave R}$
f_1	79.0	79.2
f_2	6.03×10^3	5.48×10^3
f_3	10.8	10.5
f_4	131	131
f_5	34.1	33.4
f_6	5.20×10^5	4.48×10^5
SSRICO		
Function	$f_{ave U}$	$f_{ave R}$
f_1	9.31×10^{-1}	9.39×10^{-1}
f_2	6.40×10^{-10}	4.33×10^{-10}
f_3	2.00×10^{-1}	1.80×10^{-1}
f_4	104	107
f_5	4.70×10^{-3}	2.83×10^{-3}
f_6	4.06×10^{-1}	6.80×10^{-1}



(a) Classical DE

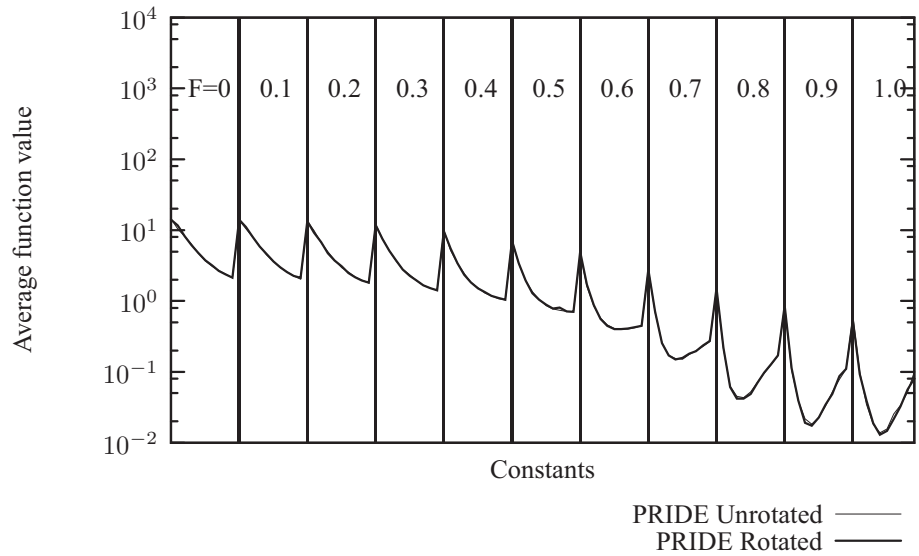


(b) GSRIDE

Figure 4.1: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Rosenbrock test function f_1

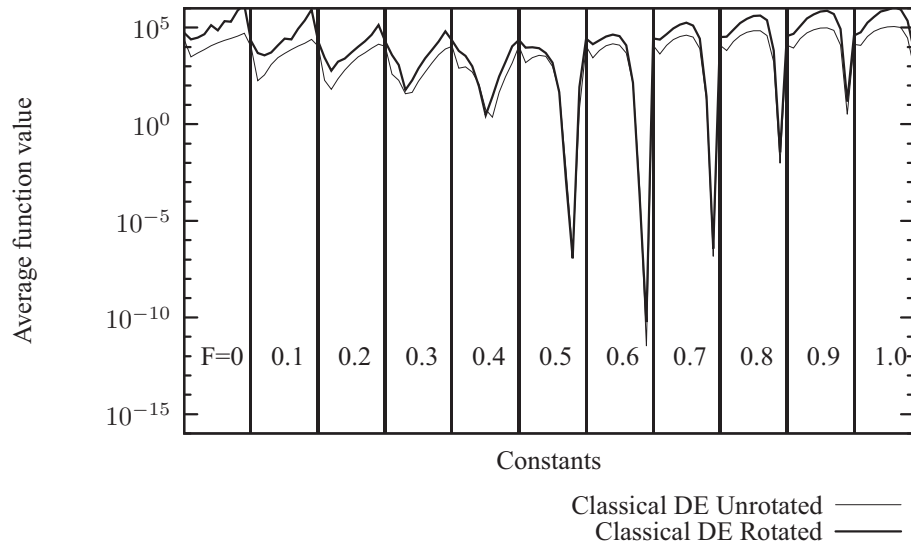


(a) SRIDE

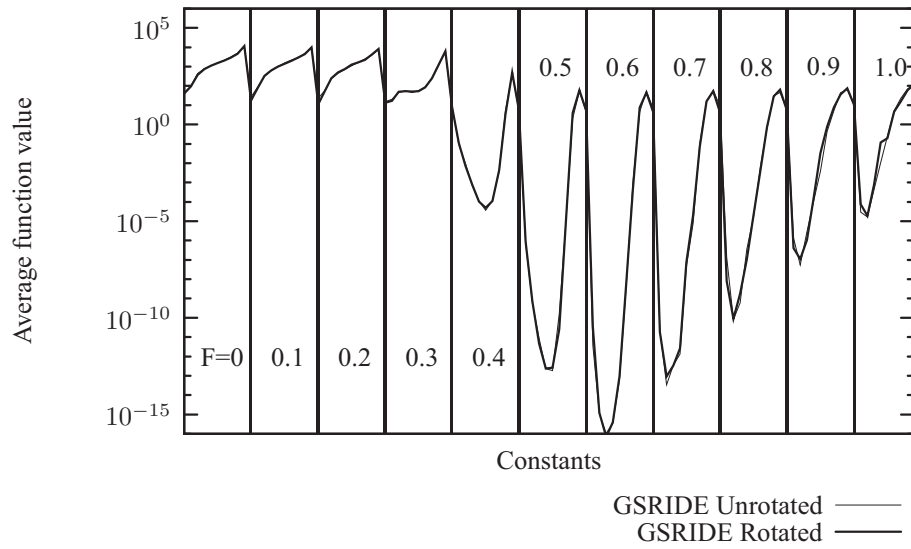


(b) PRICO

Figure 4.2: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Rosenbrock test function f_1

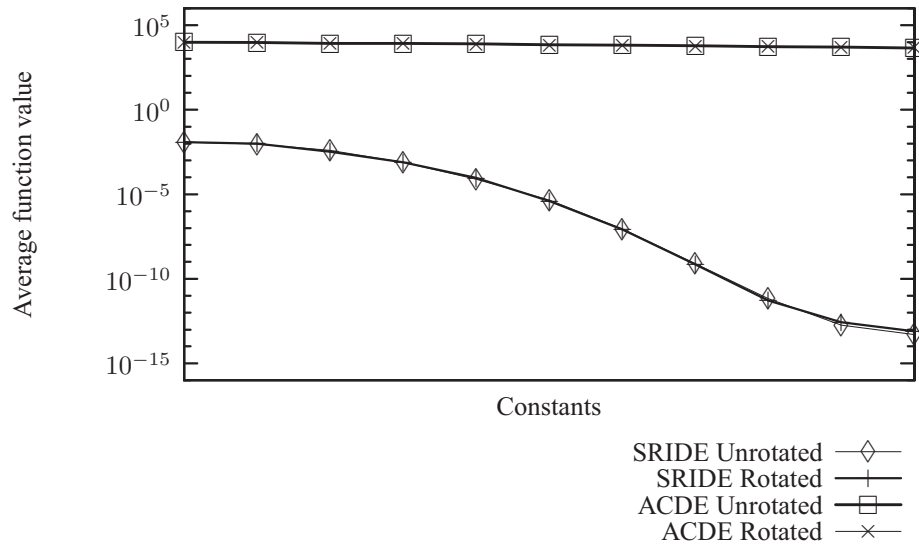


(a) Classical DE

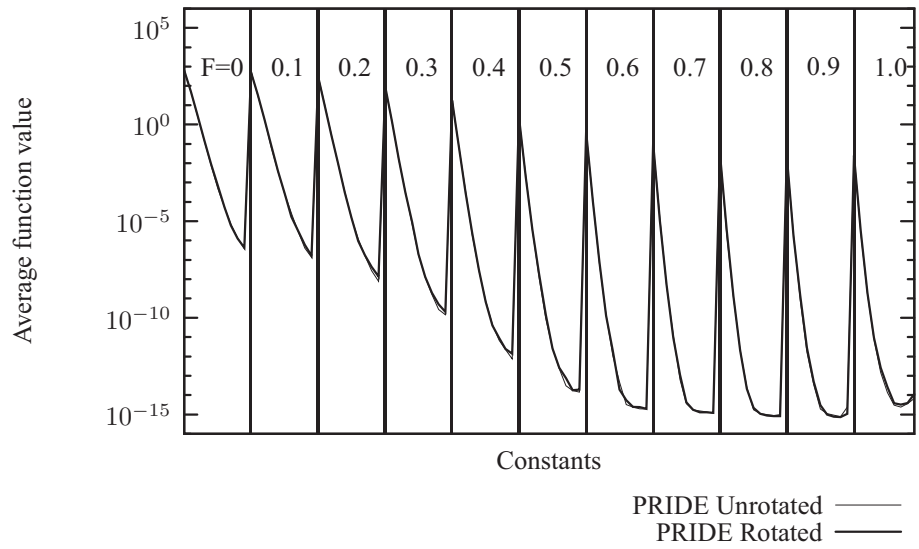


(b) GSRIDE

Figure 4.3: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Quadric test function f_2

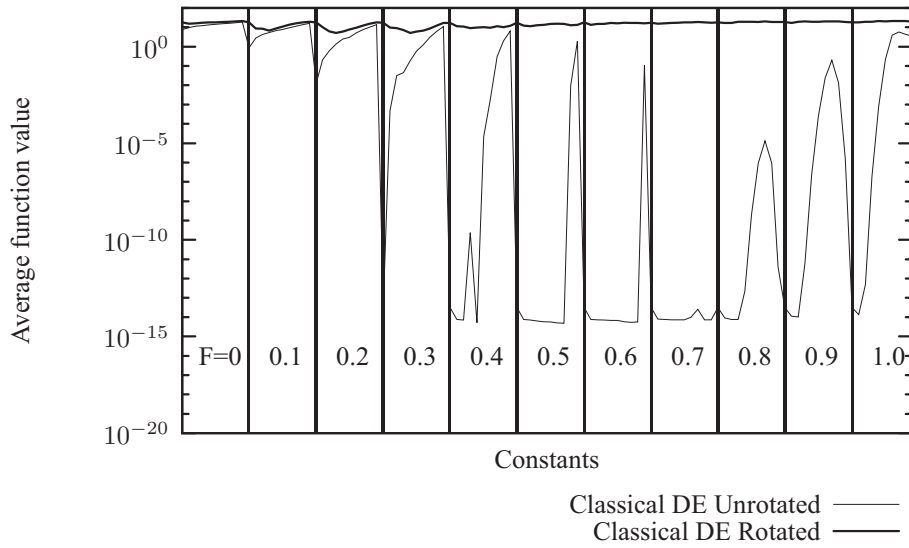


(a) ACDE and SSRICO

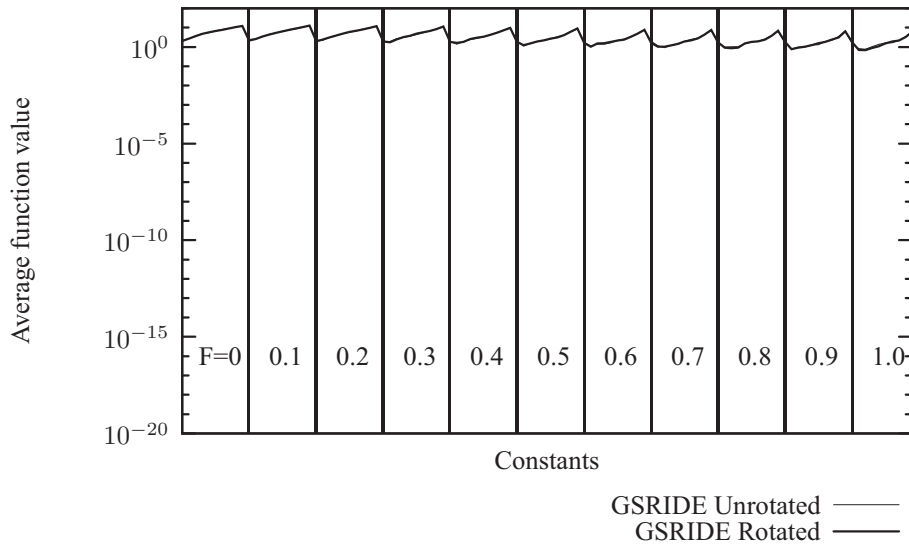


(b) PRICO

Figure 4.4: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Quadric test function f_2

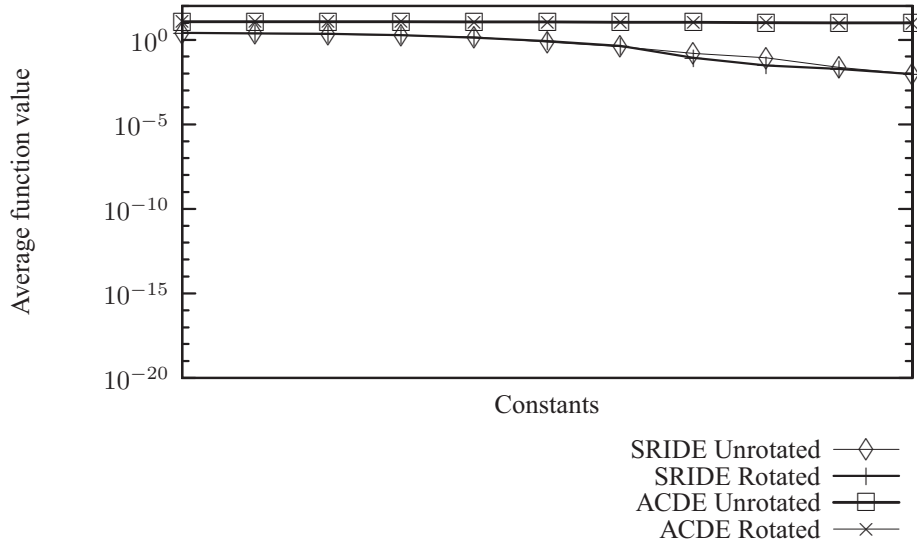


(a) Classical DE

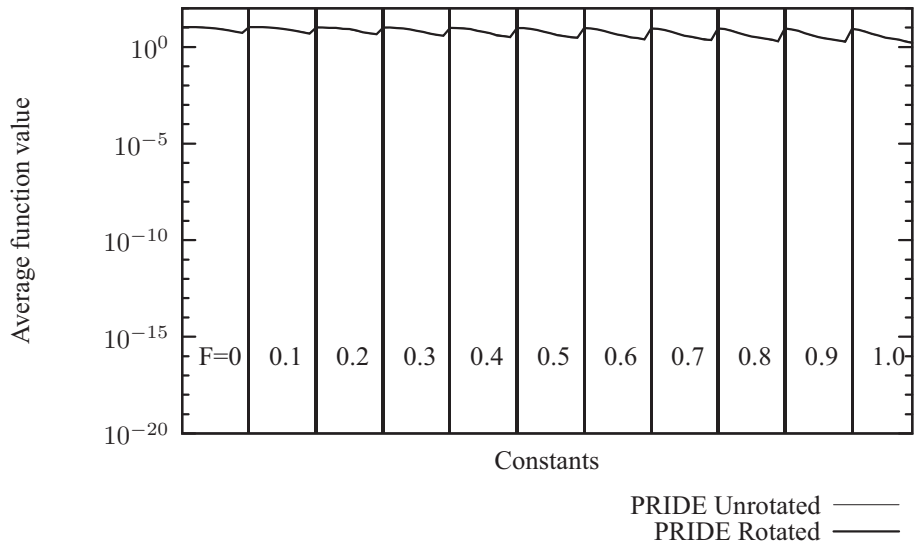


(b) GSRIDE

Figure 4.5: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Ackley test function f_3

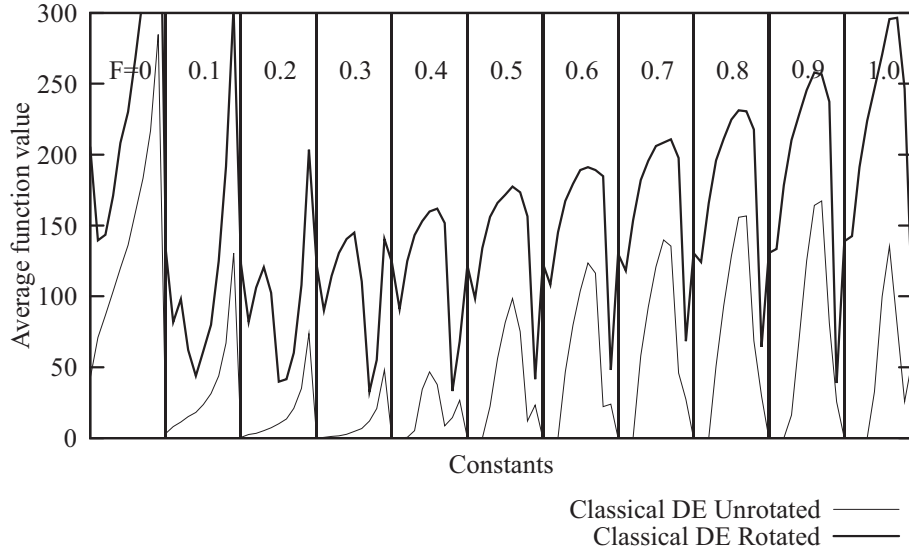


(a) ACDE and SSRICO

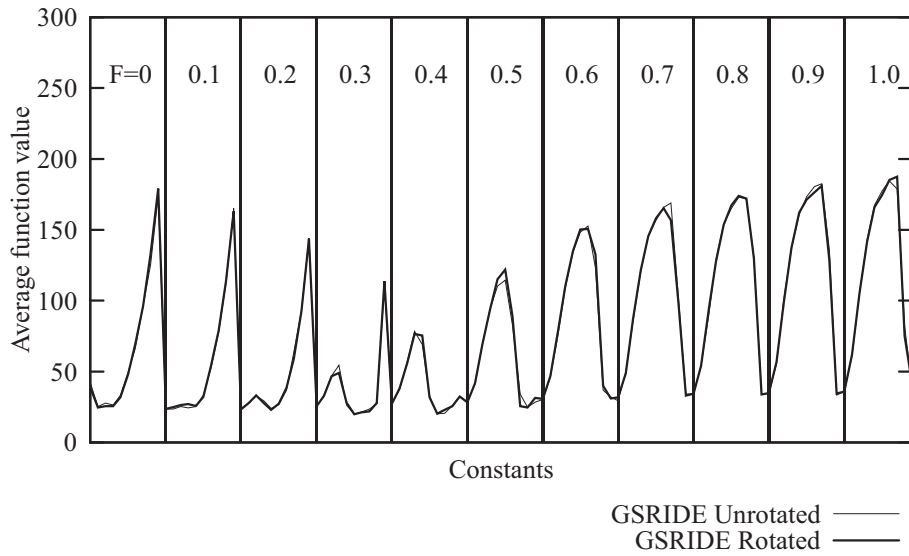


(b) PRICO

Figure 4.6: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Ackley test function f_3

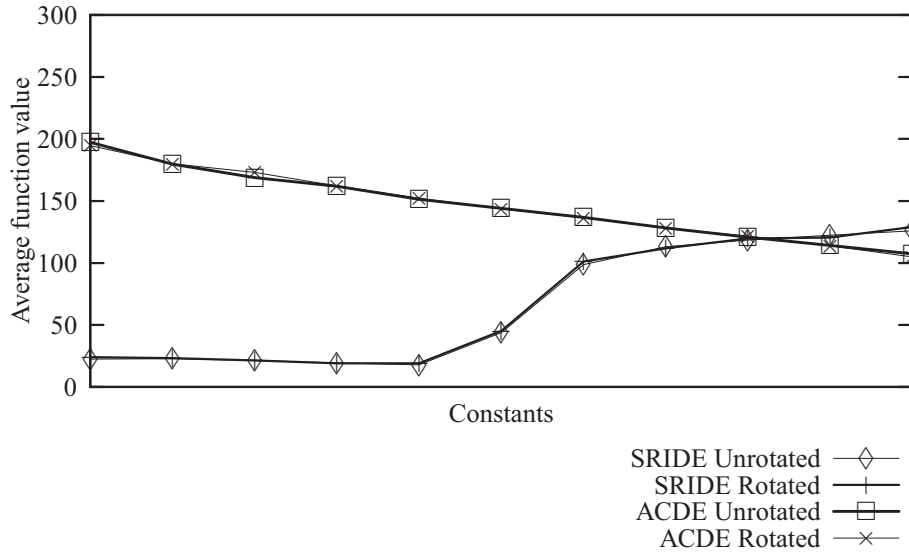


(a) Classical DE

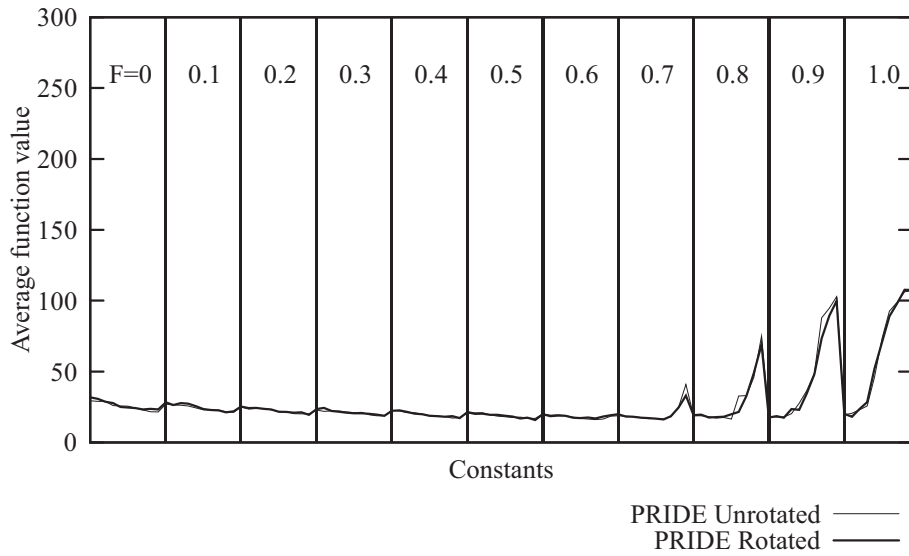


(b) GSRIDE

Figure 4.7: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Rastrigin test function f_4

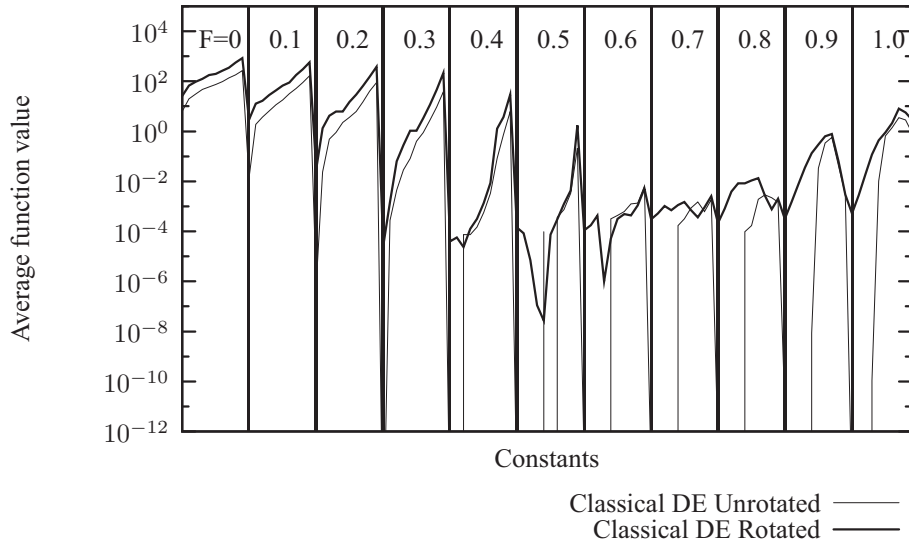


(a) ACDE and SSRICO

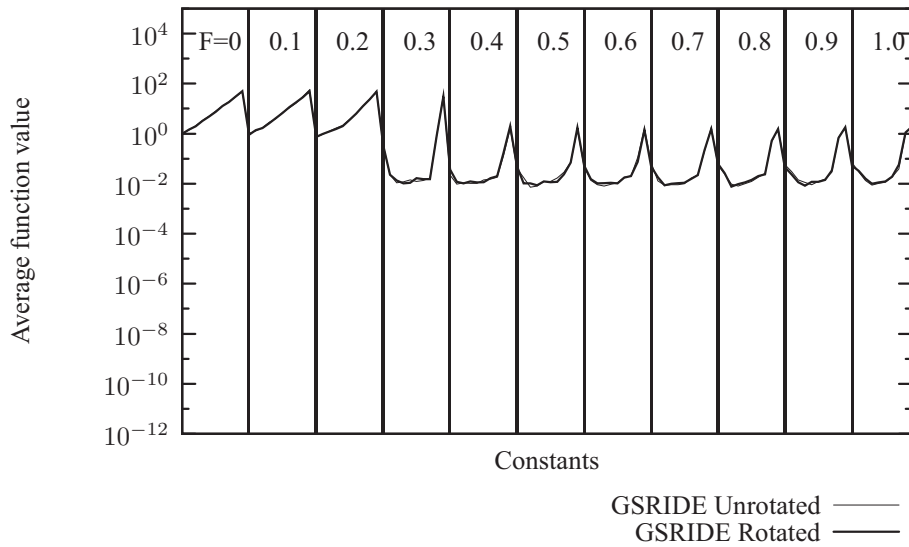


(b) PRICO

Figure 4.8: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Rastrigin test function f_4

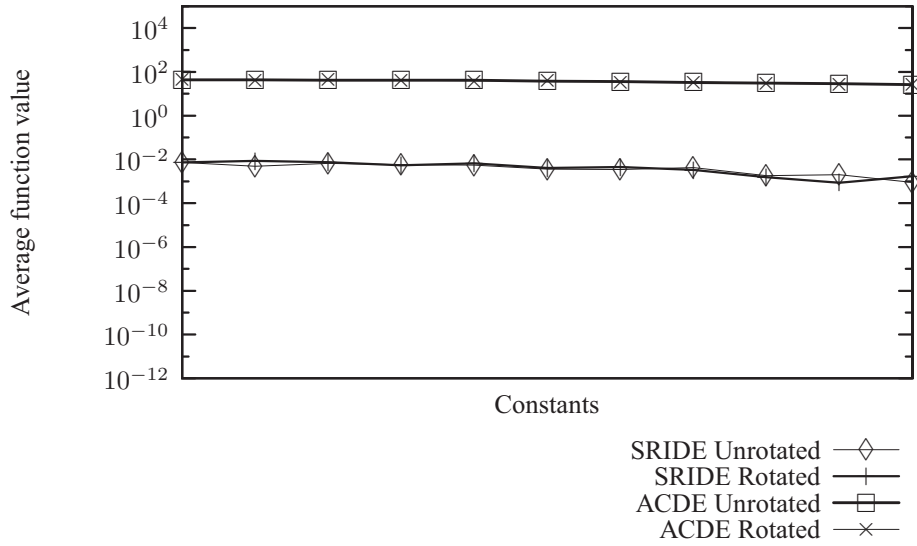


(a) Classical DE

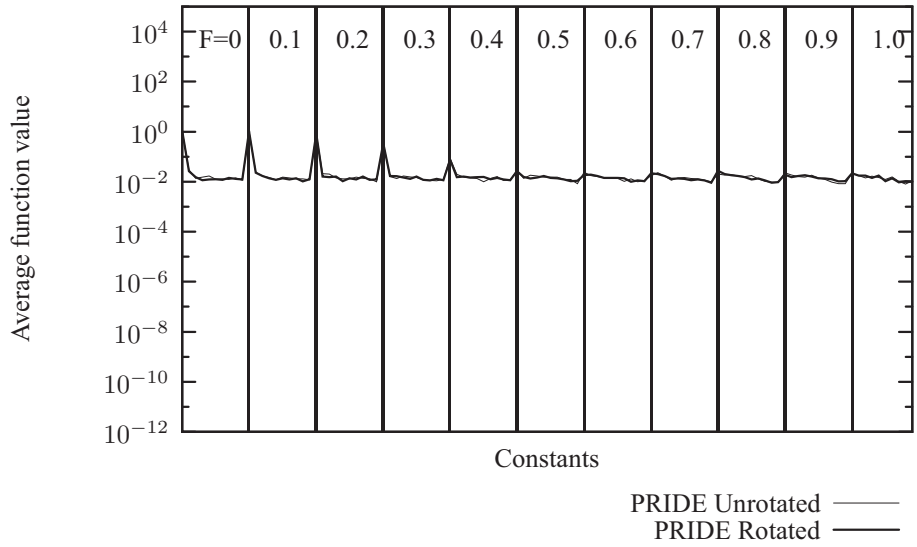


(b) GSRIDE

Figure 4.9: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Griewank test function f_5

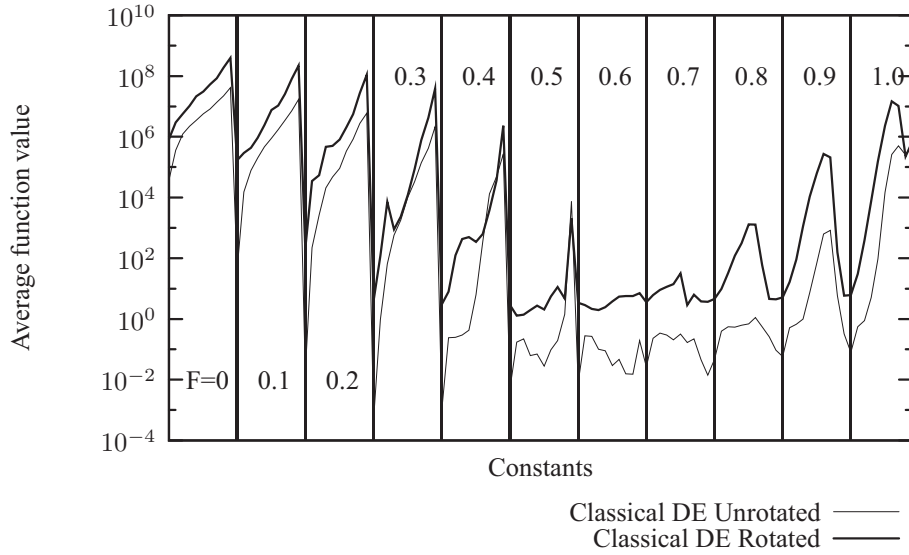


(a) ACDE and SSRICO

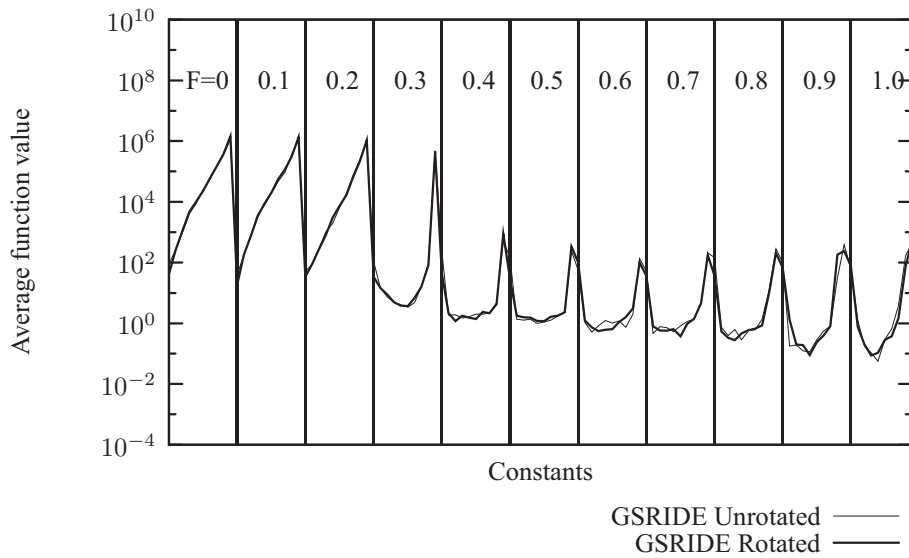


(b) PRICO

Figure 4.10: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Griewank test function f_5

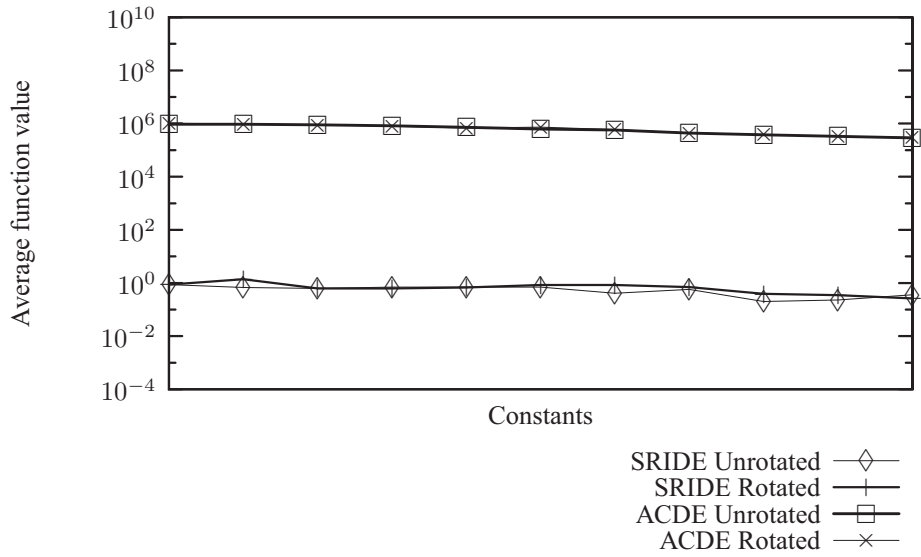


(a) Classical DE

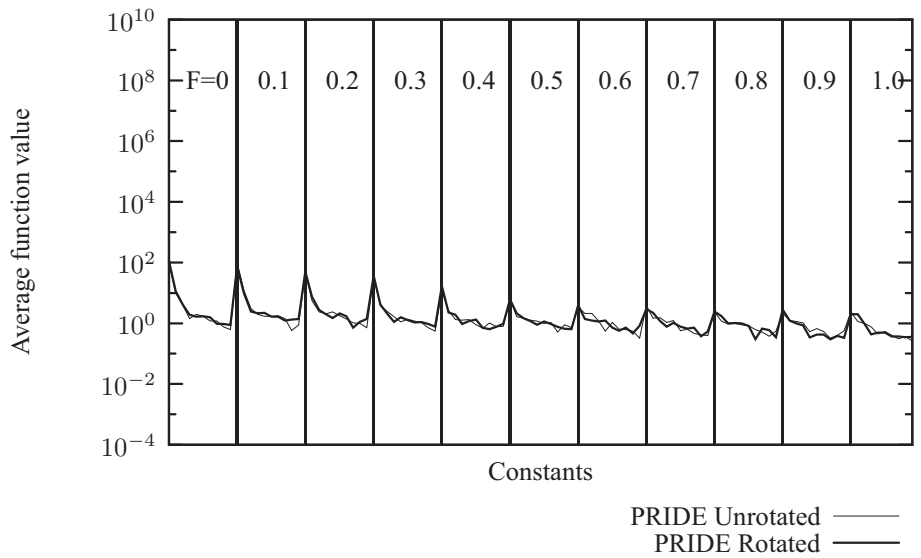


(b) GSRIDE

Figure 4.11: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Dixon-Price test function f_6



(a) ACDE and SSRICO



(b) PRICO

Figure 4.12: Average function value obtained after 7000 iterations averaged over 100 runs for the rotated and unrotated Dixon-Price test function f_6

Chapter 5

On rotationally (in)variant continuous-parameter genetic algorithms

The work presented in this chapter is reproduced from a paper titled “On rotationally invariant continuous-parameter genetic algorithms” [23]. The paper is co-authored by Dr Daniel N. Wilke of the Mechanical and Aeronautical Engineering, University of Pretoria, Pretoria, South Africa, Prof. Albert A. Groenwold of the Department of Mechanical Engineering at the University of Stellenbosch, Stellenbosch, South Africa, and Dr S. Kok of Advanced Mathematical Modelling, CSIR Modelling and Digital Science, Pretoria, South Africa.

In this chapter we introduce two *rotationally invariant* continuous-parameter genetic algorithms (CPGAs) for which we show competing performance against the standard *variant* CPGA on a test set in the unrotated and an arbitrarily rotated reference frame.

This chapter is constructed as follows: we present the CPGA used in this study in Section 5.1. We present our first invariant CPGA in Section 5.2.1 and the second in Section 5.2.2. In Section 5.3, we present numerical results for some popular test functions, in both the unrotated and arbitrary rotated reference frames. Finally, we summarize the findings of this chapter in Section 5.4.

5.1 Continuous-parameter genetic algorithm

Various CPGA formulations exist; in this section we describe the formulation of the CPGA used herein. We deliberately keep the algorithms simple and do not implement additional heuristics as to not confuse the issue of rotational invariance. Although crossover schemes exist that are rotationally invariant, e.g. linear crossover [24], they generally lack diversity [25]. This is due to the linear combination of two parent vectors, which results in a solution vector that remains in a bounded plane in n -dimensional space. We deliberately choose a rotationally variant CPGA in order to numerically demonstrate the severity in performance loss that an algorithm may experience when there is severe coupling between the design variables.

Our initial population of $p \geq 2$ strings are generated uniform randomly in the n dimensional search space, see Section 2.4. The population at the k^{th} generation is given by \mathbf{x}_k^i , $i = 1, \dots, p$, with $\mathbf{x} \in \mathbb{R}^n$. For every generation k a number of p strings are selected for the parent population \mathbf{z}_k^i , $i =$

$1, \dots, p$, using a binary tournament selection scheme, as described in Section 5.1.1 to follow. We then perform crossover by randomly selecting any two parents from the parent population and apply the blend crossover operator (BLX- α) to bear offspring \mathbf{y}_k^i , $i = 1, 2, \dots, p$, as outlined in Section 5.1.2 to follow. The offspring generation is then mutated according to Section 5.1.3 and an elitist strategy as described in Section 5.1.4 finally gives the population for the next generation. We continue this process until the maximum number of generations k_{max} is reached.

5.1.1 Selection

In the binary tournament selection, during each iteration k we select two members randomly from the population \mathbf{x}_k^i and select the fittest member of the two for the parent population. We continue until the parent population \mathbf{z}_k^i contains the same number of members as the initial population p .

Selection is by definition scale and frame invariant as vectors are selected without performing any operations on them.

5.1.2 Blend crossover (BLX- α)

Blend crossover [26] combines two randomly selected parent vectors $\mathbf{z}_k^a, \mathbf{z}_k^b \in \{\mathbf{z}_k^1, \mathbf{z}_k^2, \dots, \mathbf{z}_k^p\}$ to form the offspring vector \mathbf{y}_{k+1}^i by applying

$$\begin{aligned} \mathbf{y}_{k+1}^i &= \mathbf{z}_k^a - \alpha(\mathbf{z}_k^b - \mathbf{z}_k^a) + \mathbf{R}_k^i(\mathbf{z}_k^b + \alpha(\mathbf{z}_k^b - \mathbf{z}_k^a) - (\mathbf{z}_k^a - \alpha(\mathbf{z}_k^b - \mathbf{z}_k^a))) \\ &= \mathbf{z}_k^a - \alpha(\mathbf{z}_k^b - \mathbf{z}_k^a) + \mathbf{R}_k^i(\mathbf{z}_k^b - \mathbf{z}_k^a)(1 + 2\alpha), \end{aligned} \quad (5.1)$$

with α a user specified real parameter and \mathbf{R}_k^i a diagonal matrix ($n \times n$) with a uniform random number $\in [0, 1]$ on each diagonal entry. We continue until the offspring population contains the same number of members as the initial population.

The transformed offspring vector $\hat{\mathbf{y}}_{k+1}^i$ is given by

$$\begin{aligned} \hat{\mathbf{y}}_{k+1}^i &= \hat{\mathbf{z}}_k^a - \alpha(\hat{\mathbf{z}}_k^b - \hat{\mathbf{z}}_k^a) + \hat{\mathbf{R}}_k^i(\hat{\mathbf{z}}_k^b - \hat{\mathbf{z}}_k^a)(1 + 2\alpha) \\ &= (s\mathbf{Q}\mathbf{z}_k^a + \mathbf{t}) - \alpha((s\mathbf{Q}\mathbf{z}_k^b + \mathbf{t}) - (s\mathbf{Q}\mathbf{z}_k^a + \mathbf{t})) + \\ &\quad \hat{\mathbf{R}}_k^i((s\mathbf{Q}\mathbf{z}_k^b + \mathbf{t}) - (s\mathbf{Q}\mathbf{z}_k^a + \mathbf{t}))(1 + 2\alpha) \\ &= s\mathbf{Q}\mathbf{z}_k^a + \mathbf{t} - s\alpha(\mathbf{Q}\mathbf{z}_k^b - \mathbf{Q}\mathbf{z}_k^a) + s\hat{\mathbf{R}}_k^i(\mathbf{Q}\mathbf{z}_k^b - \mathbf{Q}\mathbf{z}_k^a)(1 + 2\alpha) \\ &= s\mathbf{Q}\mathbf{z}_k^a - s\alpha\mathbf{Q}(\mathbf{z}_k^b - \mathbf{z}_k^a) + s\hat{\mathbf{R}}_k^i\mathbf{Q}(\mathbf{z}_k^b - \mathbf{z}_k^a)(1 + 2\alpha) + \mathbf{t}. \end{aligned} \quad (5.2)$$

From (2.2) and (2.5), the required transformed offspring vector $\hat{\mathbf{y}}_{k+1}^i$ for rotational invariance is

$$\begin{aligned} \hat{\mathbf{y}}_{k+1}^i &= s\mathbf{Q}\mathbf{y}_{k+1}^i + \mathbf{t} \\ &= s\mathbf{Q}(\mathbf{z}_k^a - \alpha(\mathbf{z}_k^b - \mathbf{z}_k^a) + \mathbf{R}_k^i(\mathbf{z}_k^b - \mathbf{z}_k^a)(1 + 2\alpha)) + \mathbf{t} \\ &= s\mathbf{Q}\mathbf{z}_k^a - s\alpha\mathbf{Q}(\mathbf{z}_k^b - \mathbf{z}_k^a) + s\mathbf{Q}\mathbf{R}_k^i(\mathbf{z}_k^b - \mathbf{z}_k^a)(1 + 2\alpha) + \mathbf{t}. \end{aligned} \quad (5.3)$$

By comparing (5.2) and (5.3) we note that for rotational invariance we require

$$\hat{\mathbf{R}}_k^i\mathbf{Q}(\mathbf{z}_k^b - \mathbf{z}_k^a) = \mathbf{Q}\mathbf{R}_k^i(\mathbf{z}_k^b - \mathbf{z}_k^a), \quad (5.4)$$

which in turn implies

$$\hat{\mathbf{R}}_k^i \mathbf{Q} = \mathbf{Q} \mathbf{R}_k^i \Rightarrow \hat{\mathbf{R}}_k^i = \mathbf{Q} \mathbf{R}_k^i \mathbf{Q}^\top \quad \forall \mathbf{Q} \in \text{Orth}^+. \quad (5.5)$$

Since the relation between $\hat{\mathbf{R}}_k^i$ and \mathbf{R}_k^i has to hold $\forall \mathbf{Q} \in \text{Orth}^+$, (5.5) is satisfied when

$$\hat{\mathbf{R}}_k^i = \mathbf{R}_k^i = r \mathbf{I}, \quad (5.6)$$

with r a uniform random number $\in [0, 1]$. This implies that strict frame invariance only allows for scaling the magnitude of $(\mathbf{z}_k^b - \mathbf{z}_k^a)$.

5.1.3 Mutation scheme

We randomly mutate components of the newly formed offspring population \mathbf{y}_{k+1}^i , $i = 1, 2, \dots, p$. To do this we generate a uniform random scalar $\gamma \in [0, 1]$ for each j^{th} component and then apply the following:

$$x_{k+1}^i(j) = \begin{cases} y_{k+1}^i(j) & \gamma > P_m, \\ x_{LB}(j) + \delta(j)(x_{UB}(j) - x_{LB}(j)) & \gamma \leq P_m, \end{cases} \quad (5.7)$$

for $j = 1, 2, \dots, n$, with $\delta(j)$ a uniform random scalar $\in [0, 1]$ and P_m a user specified probability of mutation, usually $\in [0, 0.03]$. Mutation is disabled for $P_m = 0$, and the swarm is effectively reinitialized in the search space for $P_m = 1$, i.e. it becomes uniform random search.

Equivalently, (5.7) can be expressed as

$$\mathbf{x}_{k+1}^i = \mathbf{B}_{k+1}^i \mathbf{y}_{k+1}^i + \tilde{\mathbf{B}}_{k+1}^i (\mathbf{x}_{LB} + \Upsilon_{k+1}^i (\mathbf{x}_{UB} - \mathbf{x}_{LB})), \quad (5.8)$$

with Υ_{k+1}^i a $n \times n$ diagonal matrix with each diagonal entry a random number $\in [0, 1]$. \mathbf{B}_{k+1}^i a diagonal matrix with 1 on the diagonal for $\gamma > P_m$; otherwise 0. $\tilde{\mathbf{B}}_{k+1}^i$ is a diagonal matrix with the conjugate diagonal of \mathbf{B}_{k+1}^i i.e. $\tilde{\mathbf{B}}_{k+1}^i + \mathbf{B}_{k+1}^i = \mathbf{I}$ with \mathbf{I} the identity matrix.

The transformed position vector is

$$\begin{aligned} \hat{\mathbf{x}}_{k+1}^i &= \hat{\mathbf{B}}_{k+1}^i \hat{\mathbf{y}}_{k+1}^i + \hat{\tilde{\mathbf{B}}}_{k+1}^i (\hat{\mathbf{x}}_{LB} + \hat{\Upsilon}_{k+1}^i (\hat{\mathbf{x}}_{UB} - \hat{\mathbf{x}}_{LB})) \\ &= \hat{\mathbf{B}}_{k+1}^i (s \mathbf{Q} \mathbf{y}_{k+1}^i + \mathbf{t}) + \hat{\tilde{\mathbf{B}}}_{k+1}^i (s \mathbf{Q} \mathbf{x}_{LB} + \mathbf{t} + \hat{\Upsilon}_{k+1}^i (s \mathbf{Q} \mathbf{x}_{UB} + \mathbf{t} - s \mathbf{Q} \mathbf{x}_{LB} - \mathbf{t})) \\ &= s \hat{\mathbf{B}}_{k+1}^i \mathbf{Q} (\mathbf{y}_{k+1}^i + \mathbf{t}) + s \hat{\tilde{\mathbf{B}}}_{k+1}^i (\mathbf{Q} \mathbf{x}_{LB} + \mathbf{t} + \hat{\Upsilon}_{k+1}^i \mathbf{Q} (\mathbf{x}_{UB} - \mathbf{x}_{LB})). \end{aligned} \quad (5.9)$$

By considering the transformation rules in (2.2) and (2.5) the required transformed position vector is

$$\begin{aligned} \hat{\mathbf{x}}_{k+1}^i &= s \mathbf{Q} \mathbf{x}_{k+1}^i + \mathbf{t} \\ &= s \mathbf{Q} \mathbf{B}_{k+1}^i \mathbf{y}_{k+1}^i + s \mathbf{Q} \tilde{\mathbf{B}}_{k+1}^i (\mathbf{x}_{LB} + \Upsilon_{k+1}^i (\mathbf{x}_{UB} - \mathbf{x}_{LB})) + \mathbf{t}. \end{aligned} \quad (5.10)$$

From (5.9) and (5.10) it is clear that the mutation step is not rotationally invariant.

5.1.4 Elitist strategy

Lastly, we incorporate an elitist strategy by replacing the worst offspring vector (highest fitness value) with the string in the previous iteration with the lowest fitness value. This ensures that the best known solution is preserved and available for parent selection.

5.2 Rotation invariant continuous-parameter genetic algorithm

We will now present two rotationally invariant algorithms. Both these algorithms use a crossover scheme with magnitudal scaling of $(z_k^b - z_k^a)$ and we then wish to introduce diversity into the algorithms. The first algorithm uses a modified mutation scheme to introduce diversity. The second algorithm does away with mutation and introduces diversity by simply adding a random vector to the offspring vector.

5.2.1 Modified mutation scheme

In the first algorithm, denoted the CPGA(M), the mutation operator is modified by simply operating on a complete string instead of specific genes of a string only. This ensures that the intrinsic properties of the string vectors are preserved.

We therefore randomly select strings in the children population that are randomly placed in the feasible search space with a probability of P_m . Here P_m would be of much higher values than the traditional mutation scheme, typically $\in [0, 0.15]$.

This step, as with the initialization step, is only rotationally invariant if we generate the vector in a rotated frame. This is in general not possible. We will however see in Section 5.3 that the variance due to this step is negligible and we consider this algorithm *stochastically rotationally invariant*.

Pseudocode

The pseudocode for the CPGA(M) is given in Algorithm 4.

5.2.2 Strictly (stochastically) rotationally invariant continuous-parameter genetic algorithm (CPGA(S))

Next we propose a directionally diverse rotationally invariant algorithm denoted the CPGA(S). After the offspring vector \mathbf{y}_{k+1}^i is constructed using magnitudal scaling of $(z_k^b - z_k^a)$, a new term is added to the offspring vector to introduce diversity. The new term is simply a scaled, normalized random vector with a *standard normal distribution*. This results in a point sampled *uniformly* on the surface of a n dimensional unit sphere [16].

The new vector, \mathbf{x}_{k+1}^i is then formulated as

$$\mathbf{x}_{k+1}^i = \mathbf{y}_{k+1}^i + c_s r_k^i (1 + 2\alpha) \|z_k^b - z_k^a\| \Phi_k^i, \quad (5.11)$$

Algorithm 4: CPGA(M)

```

1 Set:  $p, n, \alpha, P_m, k_{max}$ 
2 Initialize:  $k = 0$ 
3 for  $i = 1, p$  do
4   Initialize:  $\mathbf{x}_0^i$ 
5 end
6 repeat
7   for  $i = 1, p$  do
8     Generate:  $r_1, r_2$ 
9     if  $f(\mathbf{x}_k^{r_1}) < f(\mathbf{x}_k^{r_2})$  then
10       $\mathbf{z}_k^i \leftarrow \mathbf{x}_k^{r_1}$ 
11    else
12       $\mathbf{z}_k^i \leftarrow \mathbf{x}_k^{r_2}$ 
13    end
14  end
15  for  $i = 1, p$  do
16    Generate:  $a, b$ 
17     $\mathbf{y}_{k+1}^i \leftarrow \mathbf{z}_k^a - \alpha(\mathbf{z}_k^b - \mathbf{z}_k^a) + r_k^i(\mathbf{z}_k^b - \mathbf{z}_k^a)(1 + 2\alpha)$ 
18    if  $\gamma > P_m$  then
19       $\mathbf{x}_{k+1}^i \leftarrow \mathbf{y}_{k+1}^i$ 
20    else
21      Reinitialize:  $\mathbf{x}_{k+1}^i$ 
22    end
23  end
24   $\arg \max_i f(\mathbf{x}_{k+1}^i) \leftarrow \arg \min_i f(\mathbf{x}_k^i)$ 
25   $k \leftarrow k + 1$ 
26 until  $k = k_{max}$ 
27 end

```

with Φ_k^i the normalized vector with a standard normal distribution, r_k^i a random scalar, c_s a scaling factor and \mathbf{y}_{k+1}^i the offspring vector we get using blend crossover, as given in (5.1). We introduce $(1 + 2\alpha)\|\mathbf{z}_k^b - \mathbf{z}_k^a\|$ to ensure the additional term converges as the population converges.

The transformed blended vector is then given by

$$\begin{aligned}
\hat{\mathbf{x}}_{k+1}^i &= \hat{\mathbf{y}}_{k+1}^i + c_s r_k^i (1 + 2\alpha) \|\hat{\mathbf{z}}_k^b - \hat{\mathbf{z}}_k^a\| \Phi_k^i \\
&= s\mathbf{Q}\mathbf{y}_{k+1}^i + \mathbf{t} + c_s r_k^i (1 + 2\alpha) \|s\mathbf{Q}\mathbf{z}_k^b + \mathbf{t} - s\mathbf{Q}\mathbf{z}_k^a - \mathbf{t}\| \Phi_k^i \\
&= s\mathbf{Q}\mathbf{y}_{k+1}^i + s c_s r_k^i (1 + 2\alpha) \|\mathbf{Q}(\mathbf{z}_k^b - \mathbf{z}_k^a)\| \Phi_k^i + \mathbf{t} \\
&= s\mathbf{Q}\mathbf{y}_{k+1}^i + s c_s r_k^i (1 + 2\alpha) \|\mathbf{z}_k^b - \mathbf{z}_k^a\| \Phi_k^i + \mathbf{t}.
\end{aligned} \tag{5.12}$$

The required transformation is

$$\begin{aligned}
\hat{\mathbf{x}}_{k+1}^i &= s\mathbf{Q}(\mathbf{y}_{k+1}^i + c_s r_k^i (1 + 2\alpha) \|\mathbf{z}_k^b - \mathbf{z}_k^a\| \Phi_k^i) + \mathbf{t} \\
&= s\mathbf{Q}\mathbf{y}_{k+1}^i + s c_s r_k^i (1 + 2\alpha) \|\mathbf{z}_k^b - \mathbf{z}_k^a\| \mathbf{Q}\Phi_k^i + \mathbf{t} \\
&= s\mathbf{Q}\mathbf{y}_{k+1}^i + s c_s r_k^i (1 + 2\alpha) \|\mathbf{z}_k^b - \mathbf{z}_k^a\| \Psi_k^i + \mathbf{t},
\end{aligned} \tag{5.13}$$

Algorithm 5: CPGA(S)

```

1 Set:  $p, n, \alpha, P_m, c_s, k_{max}$ 
2 Initialize:  $k = 0$ 
3 for  $i = 1, p$  do
4   Initialize:  $\mathbf{x}_0^i$ 
5 end
6 repeat
7   for  $i = 1, p$  do
8     generate  $r_1, r_2$ 
9     if  $f(\mathbf{x}_k^{r_1}) < f(\mathbf{x}_k^{r_2})$  then
10       $\mathbf{z}_k^i \leftarrow \mathbf{x}_k^{r_1}$ 
11    else
12       $\mathbf{z}_k^i \leftarrow \mathbf{x}_k^{r_2}$ 
13    end
14  end
15  for  $i = 1, p$  do
16    select:  $\mathbf{z}_k^a, \mathbf{z}_k^b$ 
17     $\mathbf{y}_{k+1}^i \leftarrow \mathbf{z}_k^a - \alpha(\mathbf{z}_k^b - \mathbf{z}_k^a) + r_k^i(\mathbf{z}_k^b - \mathbf{z}_k^a)(1 + 2\alpha)$ 
18    update  $\mathbf{x}_{k+1}^i$  using (5.11)
19  end
20   $\arg \max_i f(\mathbf{x}_{k+1}^i) \leftarrow \arg \min_i f(\mathbf{x}_k^i)$ 
21   $k \leftarrow k + 1$ 
22 until  $k = k_{max}$ 
23 end

```

with $\Psi_k^i = \mathbf{Q}\Phi_k^i$ a normalized random vector with a *standard normal distribution*. By comparing (5.12) and (5.13), it is clear that the formulation is only *strictly* rotationally invariant if $\Psi_k^i = \Phi_k^i$. This is in general not true for uniquely generated random vectors, but does hold in an averaged stochastic sense, since both vectors are normalized random vectors with a standard normal distribution. This implies that the probability density function is merely rotated between the two reference frames and remains unchanged. Hence our introduction of the terminology that the stochastic vectors are rotationally invariant in a *strictly stochastic* sense.

Pseudocode

The pseudocode for the CPGA(S) is given in Algorithm 5.

5.3 Numerical experiments

We now perform an empirical study to quantify the (lack of) rotation invariance of the standard CPGA we have presented in Section 5.1, henceforth referenced to as SCPGA, and the two proposed rotationally invariant CPGA implementations, namely CPGA(M) and CPGA(S). Again we use

the approach used by Wilke *et al.* [1]. Real variables are implemented using double-precision floating-point arithmetic. We choose a population size of $p = 30$ and conduct the study for various parameter settings of each algorithm. Each run consists of 300000 function evaluations (10000 iterations). All results presented are averaged over 100 runs.

For all formulations we vary α between 0 and 1 in steps of 0.1. For SCPGA we vary the mutation rate P_m between 0 and 0.01 in increments of 0.001. For CPGA(M) we vary the mutation rate P_m between 0 and 0.15 in increments of 0.005. For the CPGA(S) we vary the sphere scaling factor c_s from 0 to 0.4 in increments of 0.025.

We use the following six test functions that are popular in the literature:

1. The Rosenbrock function (unimodal), n even

$$f_1(\mathbf{x}) = \sum_{i=1}^{\frac{n}{2}} \left(100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2 \right)$$

2. The Quadric function (unimodal)

$$f_2(\mathbf{x}) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2$$

3. The Ackley function (multimodal)

$$f_3(\mathbf{x}) = -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right) + 20 + e$$

4. The generalized Rastrigin function (multimodal)

$$f_4(\mathbf{x}) = \sum_{i=1}^n \left(x_i^2 - 10 \cos(2\pi x_i) + 10 \right)$$

5. The generalized Griewank function (multimodal)

$$f_5(\mathbf{x}) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos \left(\frac{x_i}{\sqrt{i}} \right) + 1$$

6. The Dixon-Price function (unimodal)

$$f_6(\mathbf{x}) = \sum_{i=2}^n i(2x_i^2 - x_{i-1})^2$$

All problems have dimension $n = 30$, and each component of the initial positions are limited between ± 2.048 , ± 100 , ± 30 , ± 5.12 , ± 600 and ± 30 for the respective problems. The global minimum for all test problems is $f(\mathbf{x}^*) = 0$. Except for the Rosenbrock function f_1 , which has solution vector $\mathbf{x}^* = [1, 1, \dots, 1]^T$, all other test problems have the solution vector $\mathbf{x}^* = [0, 0, \dots, 0]^T$.

Some of the functions in our test set are decomposable [7], viz. the design variables are uncoupled. This implies that once an optimal value for a given design variable is obtained, it remains optimal, independent of the other design variables. This is similar to optimizing n 1-dimensional optimization problems, instead of 1 n -dimensional coupled optimization problem. We therefore study the test set in the unrotated or decomposable reference frame $f(\mathbf{x})$, as well as in an arbitrary rotated reference frame $f(\mathbf{Q}\mathbf{x})$, in which the design variables are coupled [17]. Here, \mathbf{Q} is a random, proper orthogonal transformation matrix, constructed as in [7]. The transformation matrix results in a pure rotation of each test function. For each of the 100 independent runs, a new random rotation matrix \mathbf{Q} is constructed, to ensure that there is no bias toward any particular reference frame.

Figures 5.1 through 5.14 depict the mean objective function values after 3×10^6 function evaluations (or 10000 iterations) averaged over 100 runs for both the various unrotated and rotated functions at various parameter values.

For the sake of clarity, an overview of the performance of the algorithms is given in Table 5.1. The table summarizes the best function values obtained, together with the parameter settings for which the algorithm obtained the best average best function value for the *unrotated* functions after 3×10^6 function evaluations (10000 iterations). The corresponding average best function values for the parameter settings of the rotated function are also given.

Table 5.1 reveals that CPGA(M) and CPGA(S) consistently outperform SCPGA in the rotated frame. Developing a multi species parallel CPGA with CPGA(M) and CPGA(S) seems a profitable endeavor for future study.

The performance of the SCPGA should be considered the benchmark for this work. The *rotational variance* of the SCPGA is evident from Figures 5.1(a), 5.3(a), 5.7(a), 5.9(a), 5.11(a) and 5.13(a). Here α is varied between each bar. It can be seen that the SCPGA is sensitive to its parameters.

From Figures 5.1(b), 5.3(b), 5.7(b), 5.9(b), 5.11(b) and 5.13(b) we note that although the CPGA(M) is not *analytically rotationally invariant*, it is *rotationally invariant in a stochastic sense*. This is because the algorithm does not search independently along the coordinate axes. This formulation is relatively insensitive to parameters, if one chooses the parameters within reasonable bounds.

The *rotational invariance* of the CPGA(S) is evident from Figures 5.2, 5.6, 5.8, 5.10, 5.12 and 5.14. Here, averaging the Quadric test function f_2 on a *linear scale* leads to a distorted view, which can be seen in Figure 5.4. This is due to single outliers in the runs, as can be seen in Figure 5.5, where the function values obtained over the course of 100 runs for particular set of parameters are shown. To avoid this we average the Quadric test function f_2 *logarithmically*. At higher values of α the formulation can give relatively competitive results.

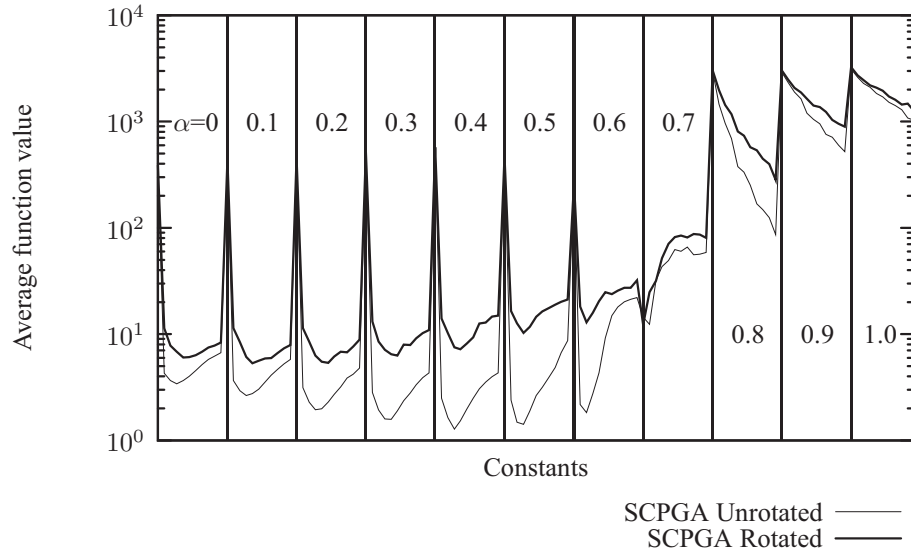
Table 5.1: Constant parameters at which the best average objective function value is obtained for the unrotated $f_{\text{ave}}^{\text{best}}|_{\text{U}}$ and rotated $f_{\text{ave}}^{\text{best}}|_{\text{R}}$ test functions

SCPGA				
Function	α	P_m	$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
f_1	0.4	0.003	1.27	7.53
f_2	0.5	0.003	5.91	2.061680×10^1
f_3	0.5	0.001	7.18×10^{-7}	8.27
f_4	0.5	0.001	5.43×10^{-10}	117
f_5	0.3	0.003	3.26×10^{-2}	2.16×10^{-2}
f_6	0.2	0.002	8.60×10^{-1}	4.14
CPGA(M)				
Function	α	P_m	$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
f_1	0.4	0.090	3.33	3.30
f_2	0.4	0.075	4.35×10^{-1}	6.64×10^{-1}
f_3	0.0	0.145	1.47	1.55
f_4	0.1	0.120	16.9	17.9
f_5	0.3	0.045	2.50×10^{-3}	1.15×10^{-2}
f_6	0.2	0.090	4.99×10^{-3}	6.18×10^{-2}
CPGA(S)				
Function	α	c_s	$f_{\text{ave}}^{\text{best}} _{\text{U}}$	$f_{\text{ave}}^{\text{best}} _{\text{R}}$
f_1	0.5	0.350	2.26	2.93
f_2	0.5	0.325	5.08×10^{-12}	3.15×10^{-12}
f_3	0.4	0.400	6.26	6.11
f_4	0.7	0.050	79.5	81.3
f_5	0.6	0.275	8.66×10^{-3}	1.04×10^{-2}
f_6	0.4	0.375	1.80	2.11

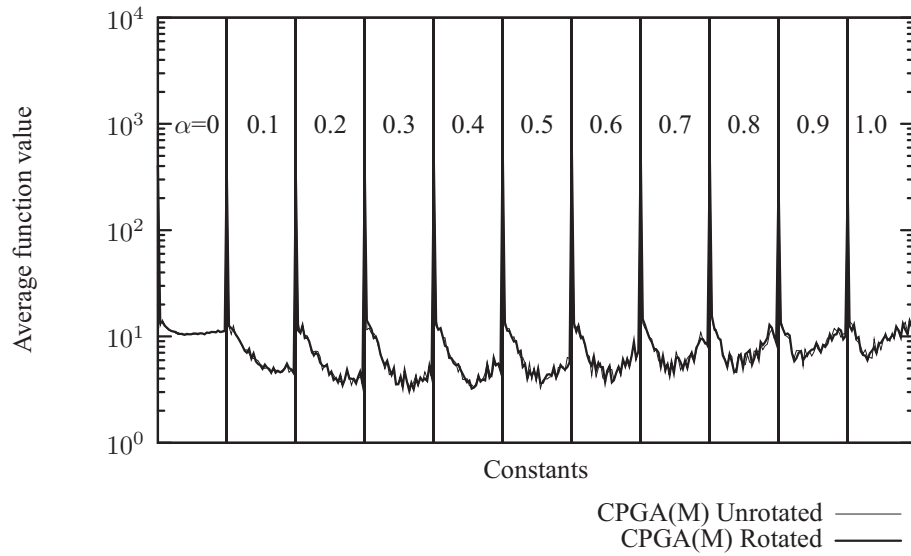
5.4 Summary

In this chapter we have shown, both analytically and numerically, the frame *variance* of the standard CPGA, which may result in severe performance loss for rotated functions.

When constructing invariant CPGAs, it is essential to retain diversity in the algorithm. When using a modified mutation scheme, we need a much higher mutation rate to achieve this. Alternatively we add diversity to the formulation by constructing a scalable n -dimensional sphere around the offspring vector.



(a) SCPGA



(b) CPGA(M)

Figure 5.1: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Rosenbrock test function f_1

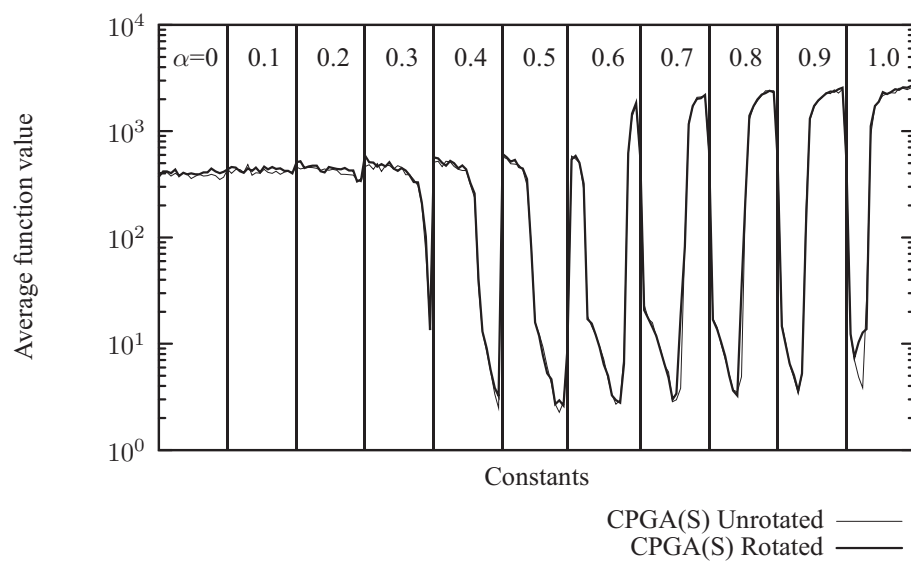
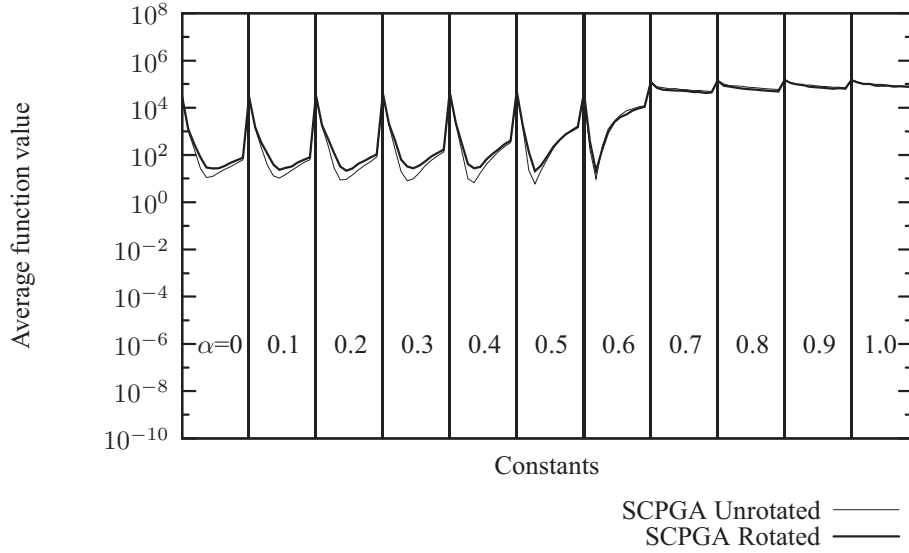
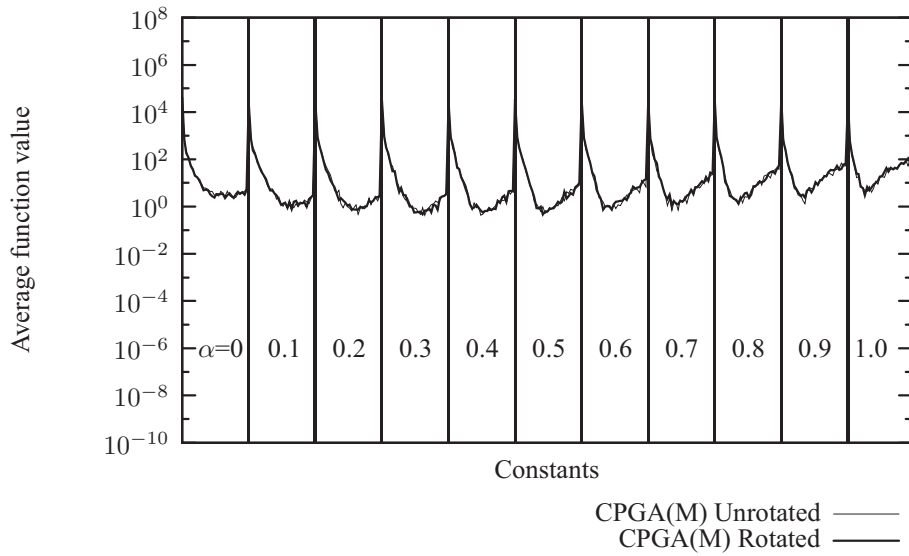


Figure 5.2: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Rosenbrock test function f_1 , using CPGA(S)



(a) SCPGA



(b) CPGA(M)

Figure 5.3: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Quadric test function f_2

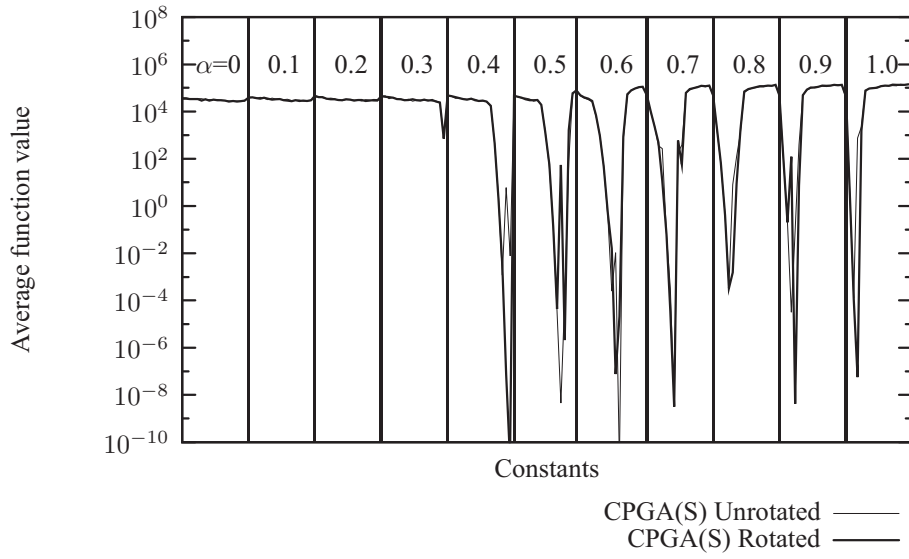


Figure 5.4: Average function value obtained after 10000 iterations *linearly* averaged over 100 runs for the rotated and unrotated Quadric test function f_2 , using CPGA(S)

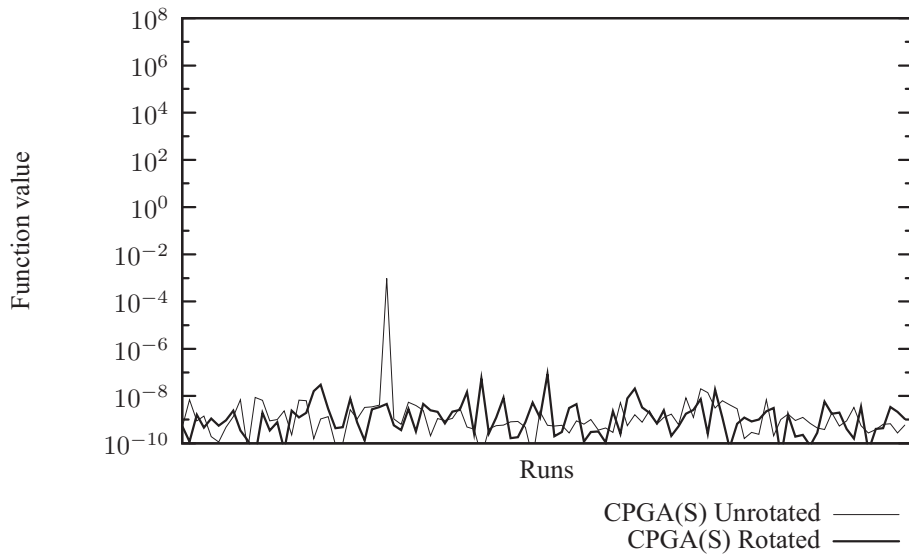


Figure 5.5: Expansion of 100 runs for the Quadric test function f_2 , using CPGA(S)

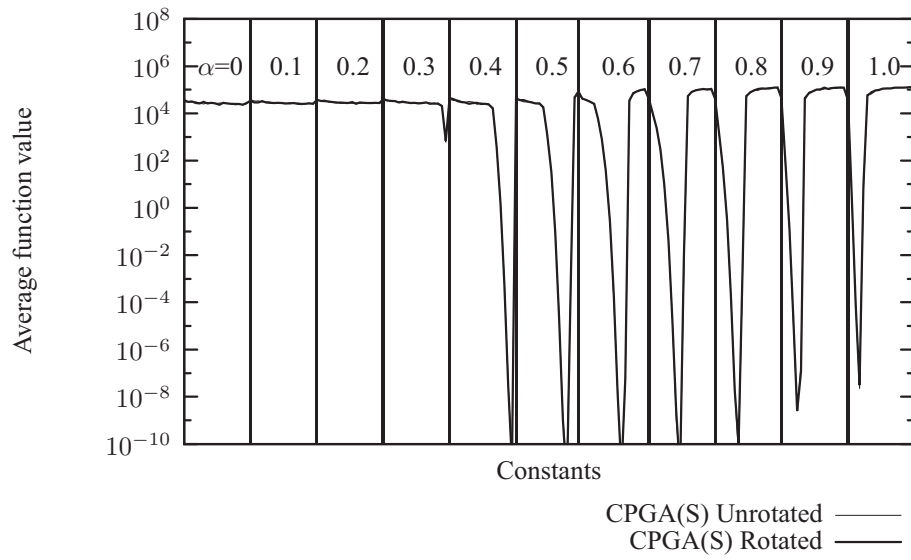
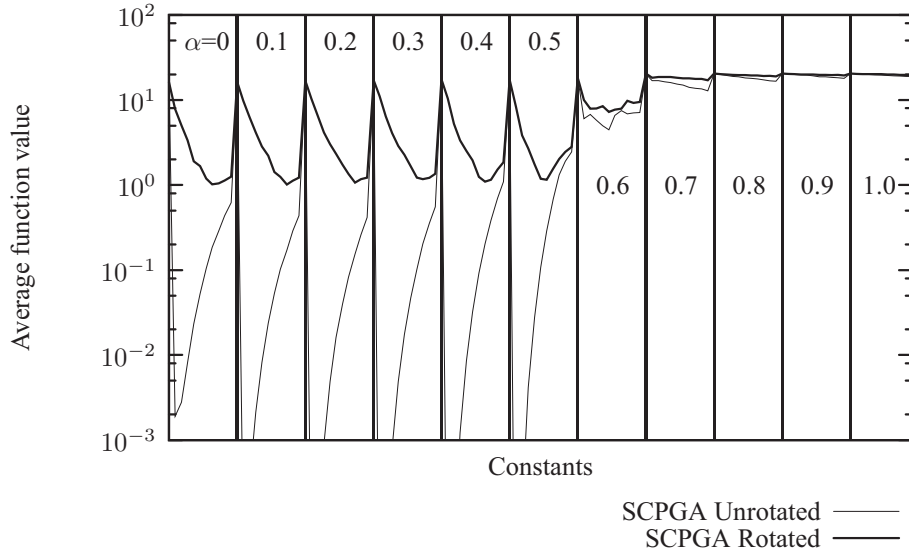
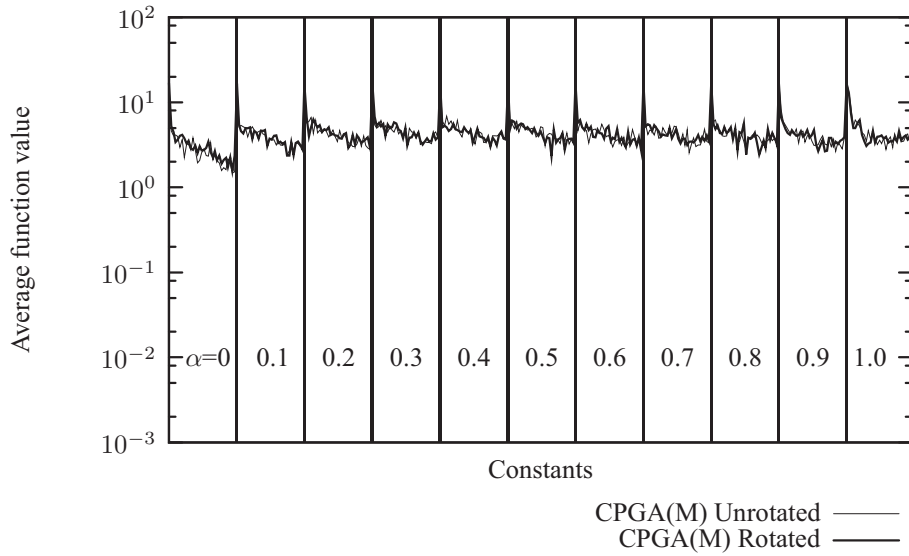


Figure 5.6: Average function value obtained after 10000 iterations *logarithmically* averaged over 100 runs for the rotated and unrotated Quadric test function f_2 , using CPGA(S)



(a) SCPGA



(b) CPGA(M)

Figure 5.7: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Ackley test function f_3

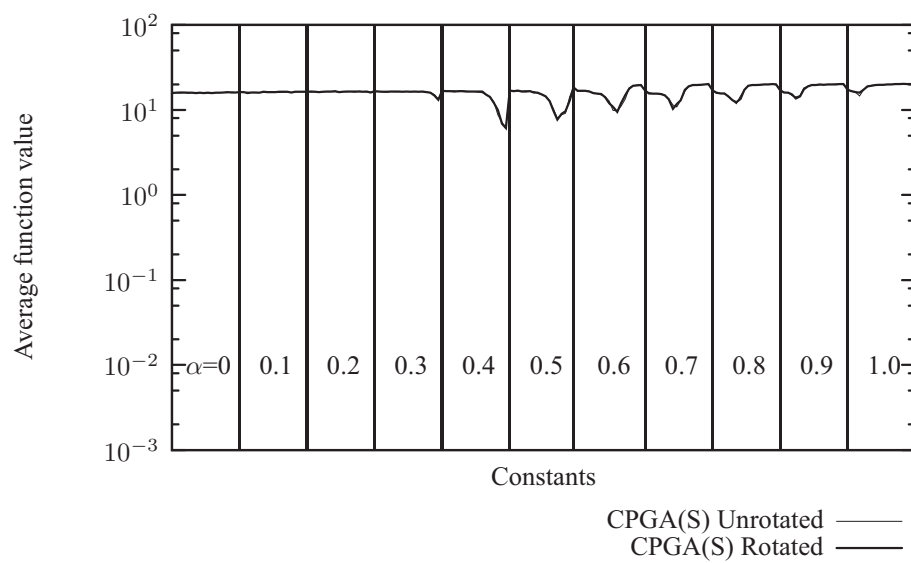
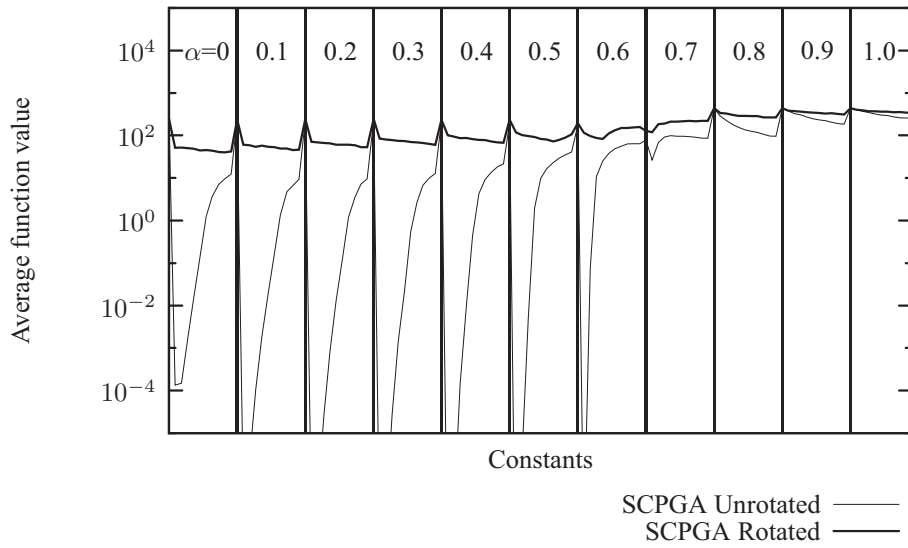
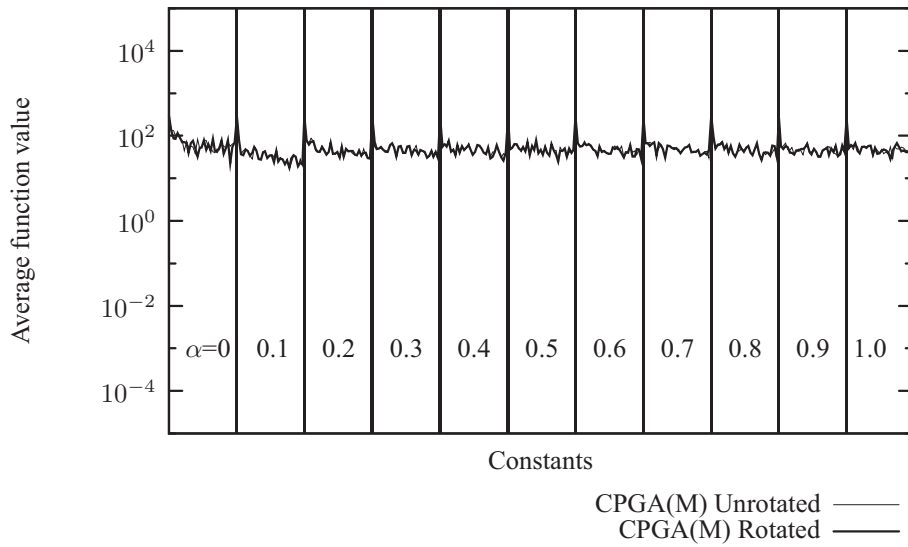


Figure 5.8: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Ackley test function f_3 , using CPGA(S)



(a) SCPGA



(b) CPGA(M)

Figure 5.9: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Rastrigin test function f_4

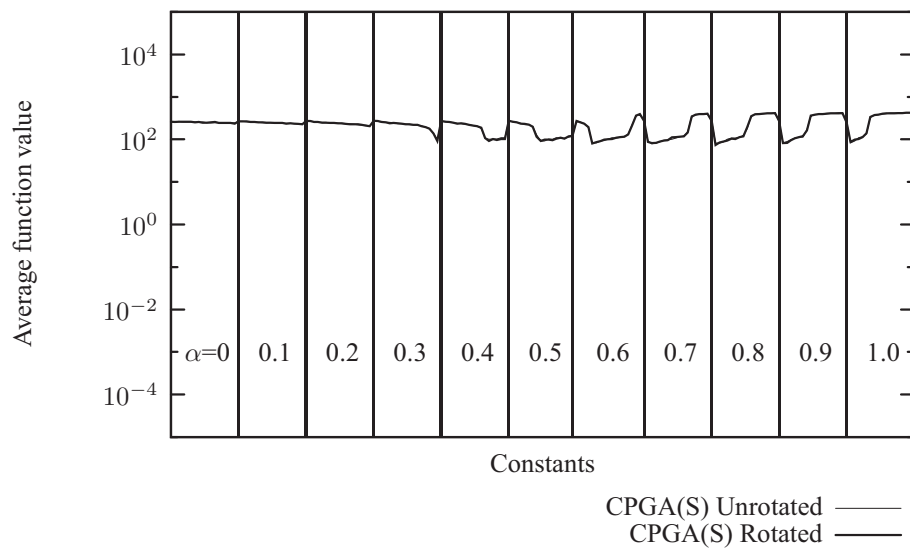
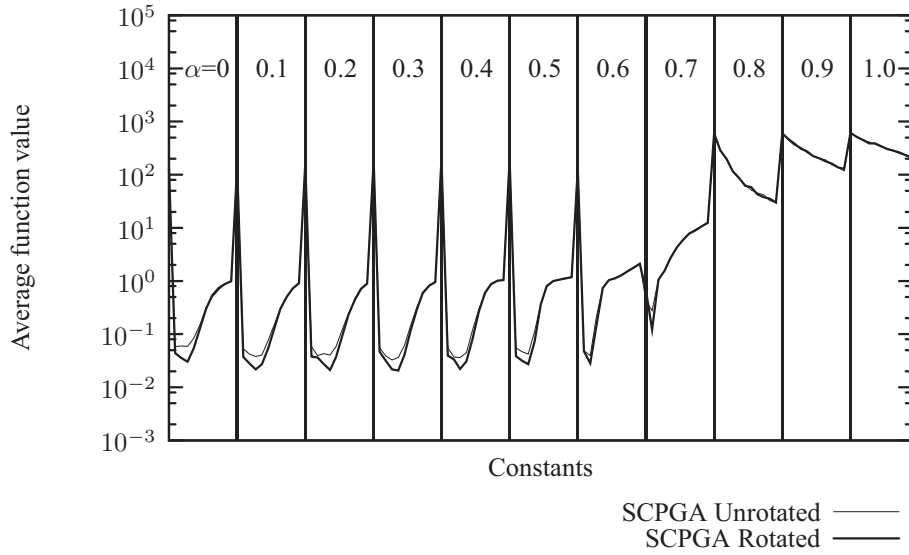
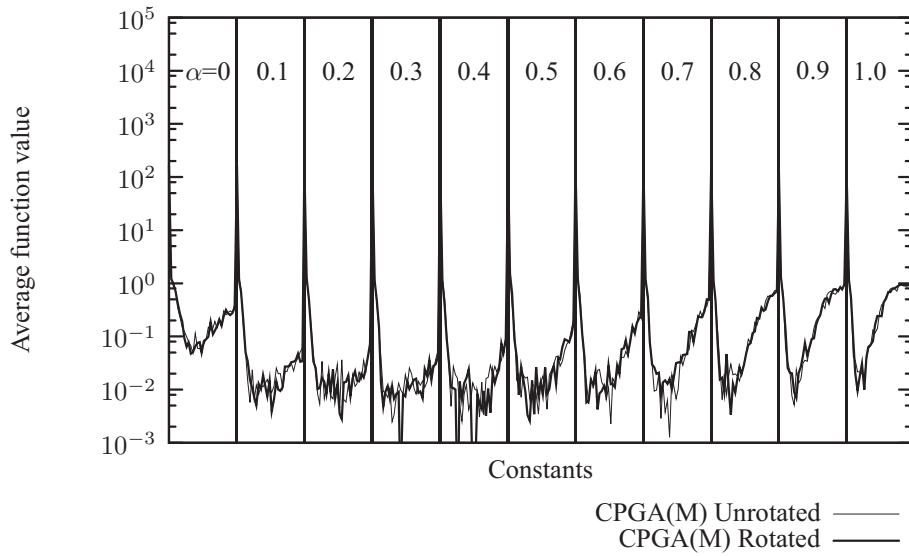


Figure 5.10: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Rastrigin test function f_4 , using CPGA(S)



(a) SCPGA



(b) CPGA(M)

Figure 5.11: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Griewank test function f_5

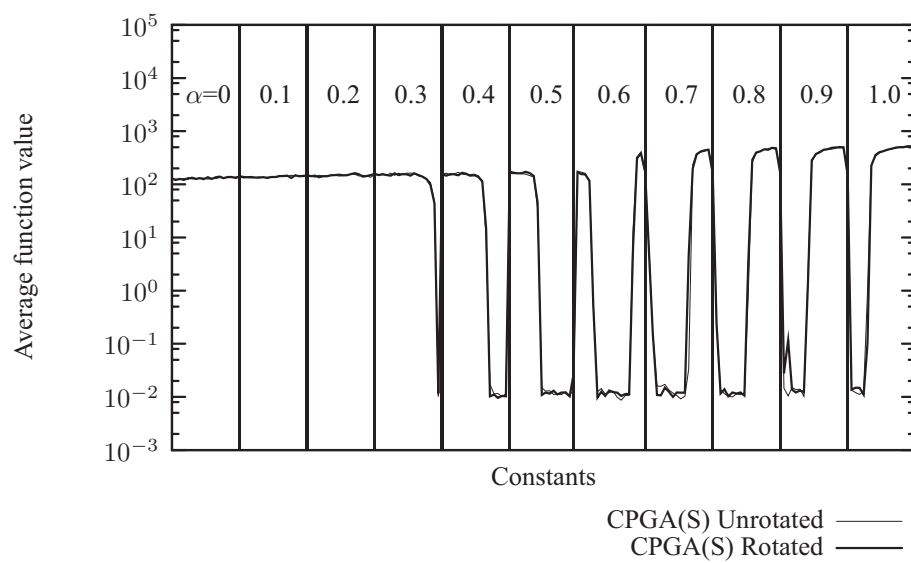
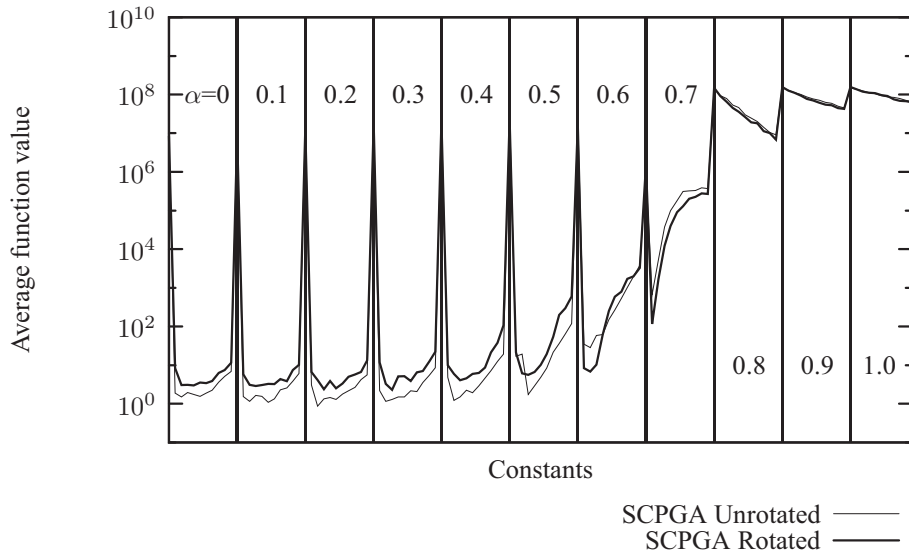
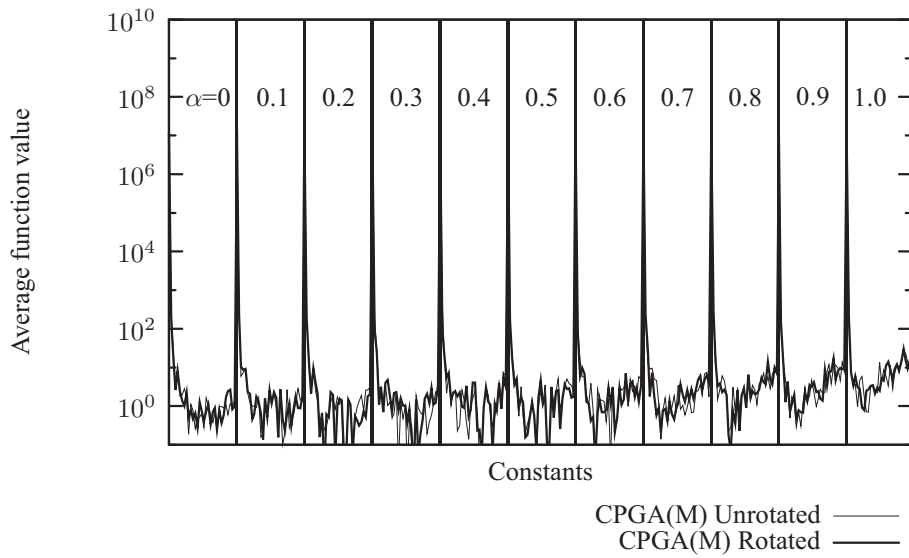


Figure 5.12: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Griewank test function f_5 , using CPGA(S)



(a) SCPGA



(b) CPGA(M)

Figure 5.13: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Dixon-Price test function f_6

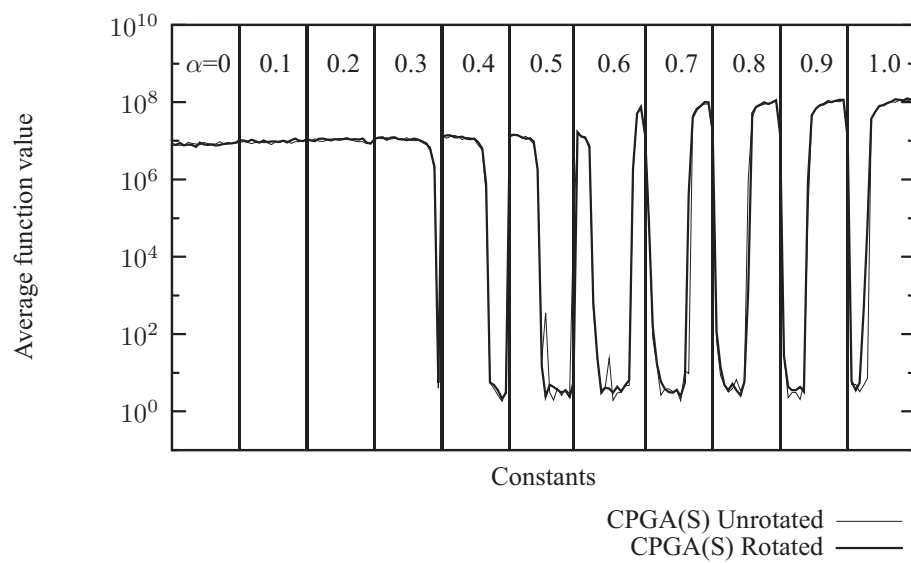


Figure 5.14: Average function value obtained after 10000 iterations averaged over 100 runs for the rotated and unrotated Dixon-Price test function f_6 , using CPGA(S)

Chapter 6

Conclusion

In this study we have considered the lack of rotational invariance of three different population based optimization methods, namely the particle swarm optimization (PSO) algorithm, the differential evolution (DE) algorithm and the continuous parameter genetic algorithm (CPGA). We have proposed a number of rotationally invariant versions of these algorithms.

When constructing invariant versions of the algorithms, it is essential to retain diversity. Our proposed PSO formulation do this by simply adding an additional term to the linear velocity update rule of the standard linear PSO.

Our proposed DE algorithms, namely the PRICO and the SSRICO algorithms, accomplish this in PRICO by merely perturbing the direction cosines of chosen difference vectors, and in SSRICO by constructing a scalable n -dimensional sphere around the trial vector.

CPGA(M), the first of our proposed invariant CPGAs, uses a modified mutation scheme, with a high mutation rate to add diversity. Our other proposed CPGA, denoted CPGA(S), adds diversity to the formulation by constructing a scalable n -dimensional sphere around the offspring vector.

The novel algorithms perform competitively, especially compared to rotated test functions using the standard algorithms, be it the classical PSO, the classical DE or the standard CPGA.

Nevertheless, the aim of this study is not to be competitive with well understood and researched algorithms, but to rather show that frame invariant algorithms can fairly easily be constructed, in the hope that this will stimulate additional future contributions, since rotational invariance in general is a desirable, salient feature for any optimization algorithm.

References

- [1] D.N. Wilke, S. Kok, and A.A. Groenwold. Comparison of linear and classical velocity update rules in particle swarm optimization: Notes on scale and frame invariance. *International Journal for Numerical Methods in Engineering*, 70:985–1008, 2007.
- [2] G.A. Holzapfel. *Nonlinear solid mechanics: A continuum approach for engineering*. Wiley, Chichester, 2001.
- [3] N. Hansen, R. Ros, N. Mauny, M. Schoenauer, and A. Auger. Impacts of invariance in search: When CMA-ES and PSO face ill-conditioned and non-separable problems. *Applied Soft Computing*, 11:5755–5769, 2011.
- [4] J.A. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer, New York, 2005.
- [5] M.R. Spiegel. *Theory and problems of vector analysis*. Schaum Publishing Co., New York, NY, USA, 1959.
- [6] R. A. Horn. *Topics in matrix analysis*. Cambridge University Press, New York, NY, USA, 1986.
- [7] R. Salomon. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. *BioSystems*, 39:3:263–278, 1996.
- [8] M. N. Ras, D. N. Wilke, A. A. Groenwold, and S. Kok. A strictly stochastically rotationally invariant PSO formulation. 2012. Submitted.
- [9] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *IEEE Conference on Evolutionary Computation*, pages 69–73, 1998.
- [10] Y. Shi and R.C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, pages 591–600, 1998.
- [11] R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *IEEE Congress on Evolutionary Computation*, pages 84–88, 2000.
- [12] U. Paquet and A.P. Engelbrecht. A new particle swarm optimiser for linearly constrained optimisation. In *IEEE Congress on Evolutionary Computation*, pages 227–233, 2003.

- [13] D.N. Wilke, S. Kok, and A.A. Groenwold. Comparison of linear and classical velocity update rules in particle swarm optimization: Notes on diversity. *International Journal for Numerical Methods in Engineering*, 70:962–984, 2007.
- [14] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *IEEE Conference on Neural Networks*, pages 1942–1948, 1995.
- [15] R.C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.
- [16] G. Marsaglia. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics*, 43(2):645–646, April 1972.
- [17] R. Salomon. Some comments on evolutionary algorithm theory. *Evolutionary Computation Journal*, 4:4:405–415, 1996.
- [18] A. Carlisle and G. Dozier. An off-the-shelf PSO: Workshop on particle swarm optimization. In *Purdue School of Engineering and Technology, IN, USA,*, 2001.
- [19] M. N. Ras, D. N. Wilke, A. A. Groenwold, and S. Kok. On the rotational variance of the DE algorithm. 2012. Submitted.
- [20] T. Takahama and S. Sakai. Solving nonlinear optimization problems by differential evolution with a rotation-invariant crossover operation using gram-schmidt process. In *Proceeding of the world congress on nature and biologically inspired computing (NaBIC2010)*, pages 533–540, 2010.
- [21] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer-Verlag, Berlin, Germany, 2005.
- [22] X. Yu and M. Gen. *Introduction to Evolutionary Algorithms*. Springer, 2010.
- [23] M. N. Ras, D. N. Wilke, A. A. Groenwold, and S. Kok. On rotationally invariant continuous-parameter genetic algorithms. 2012. Submitted.
- [24] A. H. Wright. Genetic algorithms for real parameter optimization. In Gregory J. Rawlins, editor, *Foundations of genetic algorithms*, pages 205–218. Morgan Kaufmann, San Mateo, CA, 1991.
- [25] I. Ono, H. Kita, and S. Kobayashi. A real-coded genetic algorithm using the unimodal normal distribution crossover. In Ashish Ghosh and Shigeyoshi Tsutsui, editors, *Advances in Evolutionary Computing*, Natural Computing, pages 213–237, New York, NY, USA, 2003. Springer-Verlag.
- [26] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In Darrell L. Whitley, editor, *Foundation of Genetic Algorithms 2*, pages 187–202, San Mateo, CA, 1993. Morgan Kaufmann.