UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Global optimization on manifolds

Dissertation presented by
Francois BRANCALEONE

for obtaining the Master's degree in
Mathematical Engineering

Advisor
Pierre-Antoine ABSIL

Readers
Pierre-Yves GOUSENBOURGER, **Laurent JACQUES**

Academic year 2017-2018

Abstract

The problem of minimizing an objective function f on a Riemannian manifold has been a topic of much interest over the past few years due to many important applications, such as low-rank learning, shape analysis, image segmentation, matrix mean computation, and blind source separation. Several Riemannian optimization algorithms already exist that converge to local minima of f, but the research on Riemannian global optimization algorithms is still limited.

In contrast, global optimization on Euclidean spaces has been investigated for a long time and many algorithms have been proposed, e.g., Particle Swarm Optimization, Bacterial Foraging Optimization, Random Linkage method, Markov Chain Monte Carlo (MCMC) Simulated Annealing and Covariance Matrix Adaptation Evolution Strategy.

The purpose of this master's thesis is to investigate a few of these global optimization algorithms that are relevant on Euclidean spaces and then generalize at least two of them to the manifold setting. The generalization should address the algorithm descriptions and also their analysis. A package needs to be developed and tested on one or more applications. In this work, I investigated the performances of two new methods with their accelerated versions and compare their performances with the already implemented ones on the manifold setting.

Acknowledgements

This master's thesis represents a work at the end of my university career. It is the result of a work spread over a year and marks the passage between the title of academic and engineer.

The project was generated by experts in the domain of numerical analysis and manifolds. Pierre-Antoine Absil (Université catholique de Louvain) is an expert in optimization on matrix manifolds and his research area is numerical optimization, with particular interests in numerics on manifolds, linear and nonlinear programming algorithms, and biomedical applications. The expertise field of Pierre-Yves Gousenbourger (Université catholique de Louvain) in optimization, curve fitting interpolation and representation of physical problems on manifolds is closely related to the topic of this work. Laurent Jacques (Université catholique de Louvain) counts the representation of data on non-conventional spaces among his many fields of interests and contributed to decide the direction of this master's thesis. They all have set up this project and that's why I would like to express my gratitude to them for making this master's thesis possible.

First of all, I would like to thank Pierre-Antoine Absil for his support and availability throughout the academic year. His advice and experience were of inconceivable help for the production of this work and understanding important concepts on manifolds.

I would specially thanks Pierre-Yves Gousenbourger for his constant availability and all the time spent for me to discuss and orientate some of my ideas. His help was very useful to have a better comprehension of some methods applied on manifolds. His advices to have a good visualization of the results were very helpful. Thanks to him, I had also access to some of his practical code lines to visualize some of my results.

Many thanks goes to Guillaume Olikier as well for his availability to explain me the problem he is working on in the context of his doctoral thesis.

I'm also grateful to Laurent Jacques to be part of this project and be one of the reader.

Finally, I would like to thank anyone who has contributed from far or near to this master's thesis.

Contents

Notation conventions

Vectors and matrices

$\mathbb{R}$	Set of real numbers
$\mathbb{R}^+$	Set of positive real numbers
$\mathbb{R}^n$	Set of real column vectors of size n
$\mathbb{R}^{m\times n}$	Set of real matrices with m rows and n columns
$\mathbb{R}_*^{m\times n}$	Set of full-rank $m \times n$ matrices
I_n	Identity matrix of size n
$A^\top$	Transpose of the matrix A
$\mathrm{span}(A)$	Intersection of all subspaces containing the set of columns of A
$\mathrm{trace}(A)$	Trace of the matrix A (sum of diagonal entries)
$\mathrm{col}(A)$	Subspace spanned by the columns of A
$\mathrm{diag}(A)$	Extracts the diagonal entries of A, in a column vector
$\|A\|_F$	Frobenius norm of the matrix A, $\|A\|_F = \sqrt{\mathrm{trace}(A^\top A)}$
$A \otimes B$	Kronecker product of matrices A and B
$A \succeq 0$	Positive semi definite matrix
$\exp(A)$, $\log(A)$	Matrix exponential and logarithm

Sets and manifolds

$\mathcal{M}$	Smooth, finite-dimensional (usually Riemannian) manifolds
$\mathbb{S}^{n-1}$	The unit sphere $\mathbb{S}^{n-1} = \left\{x \in \mathbb{R}^n \middle\| x^\top x = 1\right\}$
$\mathrm{St}(n,p)$	The Stiefel manifold $\mathrm{St}(n,p) = \left\{X \in \mathbb{R}^{n\times p} \middle\| X^\top X = I_p\right\}$
$\mathrm{Gr}(n,p)$	The Grassmann manifold $\mathrm{Gr}(n,p) = \left\{\mathrm{span}(X) \middle\| X \in \mathbb{R}^{n\times p} \middle\| X^\top X = I_p\right\}$
$\mathrm{Obl}(n,p)$	Oblique manifold $\mathrm{Obl}(n,p) = \left\{X \in \mathbb{R}^{n\times p} \middle\| \mathrm{diag}(X^\top X) = I_p\right\}$
$\mathrm{Elip}(n,p)$	Elliptope manifold $\mathrm{Elip}(n,p) = \left\{Y \in \mathbb{R}^{n\times p} \middle\| Y = Y^\top \succeq 0, \mathrm{rank}(Y) = n, \mathrm{diag}(Y) = I_p\right\}$ where $Y = X^\top X$ and $X \in \mathrm{Obl}(n,p)$
$\mathrm{O}(n)$	The orthogonal group $\mathrm{O}(n) = \left\{X \in \mathbb{R}^{n\times n} \middle\| X^\top X = I_n\right\}$

Tools on manifolds

$T_x\mathcal{M}$	Tangent space at x that is a point on the manifold $\mathcal{M}$
M.`retr`(x, u, t)	Retraction operator. Computes the point you reach by following the tangent vector tu starting at x.
M.`proj`(x, u)	Projection operator. Computes the orthogonal projection of the vector u from the ambient or total space to the tangent space at x.
M.`log`(x, y)	Logarithhmic map. Computes a tangent vector at x pointing toward y.
M.`rand`()	Computes a random point on the manifold.
M.`transp`(x, y, u)	Computes a tangent vector at y that "looks like" the tangent vector u at x.

Miscellaneous

$x \sim y$	Equivalence relation evaluated for two objects x and y
$[x]$	Equivalence class of x for the equivalence relation $\sim$
$\|u\|_x$	Norm of the tangent vector u at x, $\|u\|_x = \sqrt{\langle u, u\rangle_x}$
$\mathrm{dist}(x, y)$	Riemannian (or geodesic) distance.
$\angle(A, B)$	Subspace angle between the matrices A and B.

Abbreviations

SA	Simulated Annealing method
SSA	Stochastic Simulated Annealing method
DE	Differential Evolution method
CDE	Chaotic Differential Evolution method
PSO	Particle Swarm Optimization method
AccSA	Accelerated Simulated Annealing method
AccDE	Accelerated Differential Evolution method
AccCDE	Accelerated Chaotic Differential Evolution method
TR	Trustregions method
CG	Conjugate Gradient method
i.i.d	Independent, identically distributed (random variables)
SVD	Single Value Decomposition

Introduction

This master's thesis is concerned with global optimization on manifolds, that is, on smooth nonlinear spaces. This topic has several applications such as shape analysis, independent component analysis, low-rank learning, image segmentation, matrix mean computation, and blind source separation.

When we talk about global optimization, we are wondering how we could find an estimation of the *optimal solution* of a certain problem. We are for example interested in the minimum (or maximum) solution of a certain objective function f that describes a specific problem.

When facing an estimation problem, we ask ourselves the following question; how can one design efficient algorithms to perform the estimation? Efficiency and performances of the algorithms can be assessed both in terms of required computational resources and in terms of estimation quality (relative error, distance to minimal solution...). Iterative algorithms may converge to two sorts of solutions: a *global* optimizer is an absolute best, whereas a *local* optimizer is only the best one in a neighbourhood around itself. Of course, global optimizers are always the target, but for some problems, it is overwhelmingly difficult to find them.

The topic of global optimization has long been a subject of high interest in applied mathematics, which results in several algorithms built throughout the years. Indeed, many algorithms were designed to estimate the minimal solution of global optimization problems on Euclidean spaces. L. M. Rios and V. Sahinidis proposed in [1] (2013) a large review of all the algorithms that are relevant for global optimization on Euclidean spaces. Furthermore, there exist many other papers and reference books (Locatelli and Schoen, 2013 [2]) that describe global optimization algorithms on Euclidean spaces. As you can see, there are a myriad of resources available concerning this topic.

Although many algorithms are proposed in the literature for Euclidean spaces, this is not the case for topological spaces such as manifolds. Indeed, the manifold setting has been a subject of increasing interest during the last few years and many searchers try to generalize well-known methods on manifolds (Liu *et al*, 2016 [3]). The reference book of Absil *et al* [4] (2008) describes precisely the notion of a manifold and defines practical tools that are necessary to generalize algorithms.

In order to better understand how a problem is treated with manifolds, we consider the following objective function $f : \mathbb{R}^3 \to \mathbb{R} : x \mapsto f(x)$ that we want to minimize under the constraint $\|x\|_2 = 1$. Note that the constraint translates the fact that the search space

is a sphere of radius 1. To make the immediate link, the space described by a sphere is a typical example of a manifold. Naturally, there exist many other and more complex forms of manifolds (like a torus or the Stiefel manifold) and some of them are described in this work. Generally speaking, a manifold is a topological space that is locally Euclidean (i.e., around every point, there is a neighborhood that is topologically the same as the open unit ball in $\mathbb{R}^n$).

Several algorithms have already been generalized to manifolds (like trustregions, conjugate gradient ...) and are implemented on the Matlab toolbox called Manopt that was built by Nicolas Boumal [5] in 2014. This toolbox was very useful throughout this master's thesis not only because it contains implementations of *old* algorithms already generalized on manifolds, but also because it contains the implementation of many manifolds and practical tools that were used in this work to implement *novel* algorithms. One can download this toolbox on *Manopt.org*. Even if Manopt is a large toolbox for global optimization problems on manifolds, there are still other interesting algorithms that are not generalized on the manifold setting. This was the starting point for my research.

This work focuses on the generalization on manifolds of novel methods that are relevant on the Euclidean space and have not yet been implemented onto Manopt. The generalization adresses both the algorithm's description and its analysis. At the beginning of the academic year, we decided to implement two promising novel methods (Simulated Annealing and Differential Evolution) and I managed to implement accelerated versions of these two methods. I developed a package that contains all the novel methods and that could be used as add-on to the original Manopt toolbox.

I applied the novel methods to specific problems and analyzed their performances compared to the old methods that were already available. It became apparent during my experiments that for some problems, the novel methods performed the same and sometimes even better than the methods that were already available. I showed that for a simple problem, the accelerated versions of the novel methods are able to switch from one local minimum to another better local minimum while the old methods got trapped in the first local minimum they met. Furthermore, I managed to approximate an important tool for the Stiefel manifold that was not available in the Manopt toolbox. It will be demonstrated later in this work that the approximation presented satisfactory results.

This master's thesis is structured as a report. The first chapter presents the novel methods I implemented; we show the algorithms and their analysis for Euclidean spaces. The background of manifolds necessary to understand the problem is introduced in the second chapter. The third chapter presents the results which are mainly illustrated with figures and plots. Finally, I will conclude by highlighting the main differences between the methods I developed in this work and the already available ones. I will also propose some ideas that could be worth exploring as opportunities for future research.

Chapter 1

The novel methods

In this first chapter, we present the novel optimization methods that are relevant on Euclidean spaces and that we generalized on manifolds. The goal of this first chapter is to give an overview of the methods and their working on Euclidean spaces. In the second chapter, we will show how these novel methods have been adapted to fit in on Manopt.

Global optimization on Euclidean spaces has been investigated for a long time and many algorithms have been proposed. More specifically, there exist a lot of derivative-free algorithms (=algorithms that do not require derivative information) that were applied for global optimization on Euclidean spaces. Fueled by a growing number of applications in science and engineering, the development of derivative-free optimization algorithms has been studied for a long time. The main resources that gave us a large review of some optimization methods are the overview paper [1] and the reference book [2]. After travelling a large amount of possible candidate methods to implement on manifolds, we selected two of them. These two methods seemed to be the most promising ones to apply on manifolds according to a discussion we had at the beginning of the academic year with the experienced reader and supervisors.

The first and the second section are dedicated to two derivative-free methods: Simulated Annealing (SA) and Differential Evolution (DE). In the third section, we present an accelerated version of the methods that includes a gradient method. This means that these accelerated methods are thus no longer derivative-free methods.

1.1 Simulated Annealing

The Simulated Annealing (SA) method is a probabilistic technique to approximate the global optimum of a given function. The notion of 'annealing' comes from a technique involving heating and controlled cooling of a material. In SA we keep a temperature variable to simulate this heating process. We initially set it high and then allow it to slowly 'cool' as the algorithm runs. While this temperature variable is high the algorithm will be allowed, with more frequency, to accept solutions that are worse than our current solution. This gives the algorithm the ability to jump out of any local optimums. As the

temperature is reduced, so is the chance of accepting worse solutions, therefore allowing the algorithm to gradually focus on a area of the search space in which hopefully, a close to optimum solution can be found. This gradual 'cooling' process is what makes the SA remarkably effective at finding a close to optimum solution when dealing with large problems which contain numerous local optima.

1.1.1 General description and pseudocode

At each step k, the SA method considers some neighbouring state $\hat{x}_k$ of the current state x_k, and probabilistically decides between moving the system to state $\hat{x}_k$ or staying in state x_k. The probability of making the transition from the current state x_k to a candidate new state $\hat{x}_k$ is specified by an acceptance probability function $P(x_k|\hat{x}_k, T_k)$, that depends on the energies $e_k = E(x_k)$ and $\hat{e}_k = E(\hat{x}_k)$ of the two states and on a global time-varying parameter T_k called the temperature. Here, we suppose $E : \Omega \subset \mathbb{R}^n \to \mathbb{R}$ to be a continuous, real-valued function where n gives thus dimension of the space and where Ω is non-empty and bounded in $\mathbb{R}^n$.

Consider now that we wish to find the global minimum of the function $E(x)$. We could imagine the acceptance probability function to be of the form:

$$P(\hat{x}_k|x_k, T_k) = \begin{cases} exp\left[-\frac{E(\hat{x}_k)-E(x_k)}{T_k}\right], & \text{if } E(\hat{x}_k) > E(x_k) \\ 1, & \text{if } E(\hat{x}_k) \leq E(x_k) \end{cases} \tag{1.1}$$

Note that when the neighbouring state $\hat{x}_k$ of the current state x_k returns a higher energy value (that is, when $E(\hat{x}_k) > E(x_k)$), the acceptance function $P(\hat{x}_k|x_k, T_k)$, returns a value in the range $]0, 1[$. If the candidate solution $\hat{x}$ has a lower or equal energy value than x_k (that is, when $E(\hat{x}_k) \leq E(x_k)$), we accept the neighbour solution $\hat{x}_k$ to be the following iterate with probability 1.

Consider now the case: $E(\hat{x}_k) > E(x_k)$. We are thus in the case $P(\hat{x}_k|x_k, T_k) = exp\left[-\frac{E(\hat{x}_k)-E(x_k)}{T_k}\right]$. In this case, we can decide to accept the neighbouring state $\hat{x}_k$ if the following condition is verified:

$$P(\hat{x}_k|x_k, T_k) > \text{rand}(0:1)$$

where rand$(0:1)$ generates a random value between 0 and 1. Remark that for high values of the temperature T_k, the value of $P(\hat{x}_k|x_k, T_k)$ is sensitive to be very close to 1 which means that the condition has more chance to be verified and thus the neighbouring state $\hat{x}_k$ has more chance to be accepted. On the other side, when the temperature T_k is very small, the value of $P(\hat{x}_k|x_k, T_k)$ is sensitive to be significantly lower than 1 which means that the condition has a lower chance to be verified and thus the neighbouring state $\hat{x}_k$ has lower chance to be accepted. This is why we choose T_k to be a 'cooling schedule' and

is such that $T_k \to 0$ as $k \to \infty$. This way, we have that $exp\left[-\frac{E(\hat{x}_k)-E(x_k)}{T_k}\right] \to 0$ as $k \to \infty$ where k denotes the number of iterations of the algorithm. By this, as the algorithm goes further (this means: k is increasing), the probability to choose a worse solution will become smaller and smaller.

We will now describe the algorithm of the SA method. We show firstly a basic implementation of the algorithm to have a global view of the procedure:

- First we need to set the initial temperature. The initial value of the temperature must be high enough to ensure that any new solution generated in a state transition would be accepted with a probability close to 1 at the beginning of the algorithm,
- Secondly, we create a random initial solution and fix a number of maximum cost evaluations,
- Then, we begin looping. One iteration here will be to select a neighbour candidate by making a small change to our current solution. We then decide whether to move to that neighbour solution or not. At each iteration, we decrease the temperature according to a certain cooling schedule,
- Finally, stop looping when the stopping criterion is met. Usually either the system has sufficiently cooled down or the maximum number of iterations has been reach.

The pseudo-code of the method is given by Algorithm 1.

Algorithm 1 Simulated Annealing algorithm

Input: Problem, objective function $E(x)$, accept. prob. function $P(\hat{x}|x,T)$, $max\_costevals$, max_iter_temp, max_temp, min_temp, the cooling schedule $T(k, max\_temp)$.
Output: x_{best}, E_{best}
$x_{current} \leftarrow$ **CreateInitialSol**(Problem);
$x_{best} \leftarrow x_{current}$;
$E_{best} \leftarrow E(x_{best})$;
$temp_{current} \leftarrow max\_temp$;
$costevals = 1$;
$k = 0$;
while $\text{temp}_{current} > min\_temp$ **and** $costevals < max\_costevals$ **do**
  for $i = 1, ..., max\_iter\_temp$ **do**
    $E_{current} \leftarrow E(x_{current})$;
    $x_i \leftarrow$ **CreateNeighbourSolution**(Problem,$\text{x}_{current}$);
    $E_i \leftarrow E(x_i)$;
    $costevals = costevals + 2$;
    if $E_i \leq E_{current}$ **then**
      $x_{current} \leftarrow x_i$;
      if $E_{current} \leq E_{best}$ **then**
        $x_{best} \leftarrow x_{current}$;
        $E_{best} \leftarrow E(x_{best})$;
      end if
    else if $P(x_i|x_{current}, temp_{current}) > rand(0:1)$ **then**
      $x_{current} \leftarrow x_i$;
    end if
  end for
  $k = k + 1$;
  $temp_{current} \leftarrow \text{T}(k, max\_temp)$;
  end while

The following section describes the design choices and discuss the parameters that appear in the SA algorithm.

1.1.2 Design choices and parameters

As one can observe in Algorithm 1, there are some parameters that we have to define and we have to make some design choices. Table 1.1 gives a review of the terms present in the SA algorithm.

Problem	Description of the space in which we are treating the problem. We suppose here that we are on the Euclidean space $\mathbb{R}^n$ but afterwards, we will be working on manifolds.
$E(x)$	Objective function: $E : \mathbb{R}^n \to \mathbb{R}$.
$P(x\|\hat{x}, T)$	Probability density function given by 1.1.
$max\_costevals$	maximum number of evaluations of the objective function allowed.
max_iter_temp	maximum number of iterations at each temperature allowed. Generally, $max\_iter\_temp << max\_costevals$.
max_temp	initial temperature for the cooling schedule.
min_temp	final temperature for the cooling schedule that is close to 0.
$T(k, max\_temp)$	Cooling schedule. This function returns the value of the current temperature ($temp_{current}$) depending on how far we are in the cooling schedule (given by k) and the initial maximum temperature (given by max_temp). Once $temp_{current} < min\_temp$ we stop the algorithm. As discussed above, the cooling schedule is such that $T_k \to 0$ as $k \to \infty$. We describe different cooling schedules in appendix A.1.
CreateInitialSol(Problem)	This function returns an initial solution depending on the Problem. For the Euclidean space, we could imagine to simply generate a random vector in $\mathbb{R}^n$.
$costevals$	Counts the number of function evaluations. Once we have that $costevals \geq max\_costevals$, we stop the algorithm.
CreateNeighbourSolution($x_{current}$)	This function creates a neighbour solution around the state $x_{current}$. There are many ways to do this. We could for example generate a random solution. The way we choose our neighbour solution will affect the the convergence of the algorithm. This is explained in appendix A.1.

Table 1.1: Review of the terms present in the SA algorithm

Concerning the convergence properties and cooling schedules of SA, we refer to appendix A.1.

1.1.3 Stochastic Simulated Annealing

The Stochastic Simulated Annealing (SSA) method is very similar to the SA method excepted that we don't generate a neighbour solution to $x_{current}$ the same way as in Algorithm 1. Indeed, the SSA algorithm modifies only one random *component* of the current solution $x_{current}$.

To summarize, the function:

$$x_i \leftarrow \underline{\textbf{CreateNeighbourSolution}}\textbf{(Problem, } x_{current}\textbf{)}$$

in Algorithm 1 now becomes:

$$\begin{gathered} j = \text{rand}(1 : D) \\ x_{current,j} \leftarrow \underline{\textbf{CreateRandomComponent}}\textbf{(Problem, } x_{current,j}\textbf{)} \\ x_i = x_{current} \end{gathered}$$

Where $x_{current,j}$ denotes the j-th component of the current solution $x_{current}$. The component j is chosen by selecting randomly an integer in $[1, D]$ where D denotes the dimension of the solution space. However, this SSA method cannot be applied for any global optimization problem on manifolds since changing the component of an element on a manifold may yield that the element is not on the manifold anymore. Anyway, in chapter 3 we will explicitly announce when we can use the SSA method and why.

1.2 Differential Evolution

Price and Storn developed Differential Evolution (DE) [6] to be a reliable and versatile function optimizer. The "DE community" has been growing since the early DE years of 1994-1996 and even more researchers are working on and with DE. Since then, DE has proven itself in competitions like the IEEE's International Contest on Evolutionary Optimization (ICEO) in 1997.
DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. This method is thus a population based optimization algorithm that does not require the gradient or any differential form of the objective function. It is related to sibling Evolutionary Algorithms such as the Genetic Algorithm, Evolutionary Programming, and Evolution Strategies, and has some similarities with Particle Swarm Optimization (PSO). The crucial idea behind DE is a scheme for generating trial parameter vectors. Basically, DE adds the weighted difference between two population vectors to a third vector. DE has proven to be a powerful tool for solving real-valued functions, due to its simplicity, robustness and better convergence rate compared with other evolutionary algorithms [7].

1.2.1 General description and pseudocode

The DE algorithm consists of four main steps that are described in Figure 1.1. We will describe each step individually.

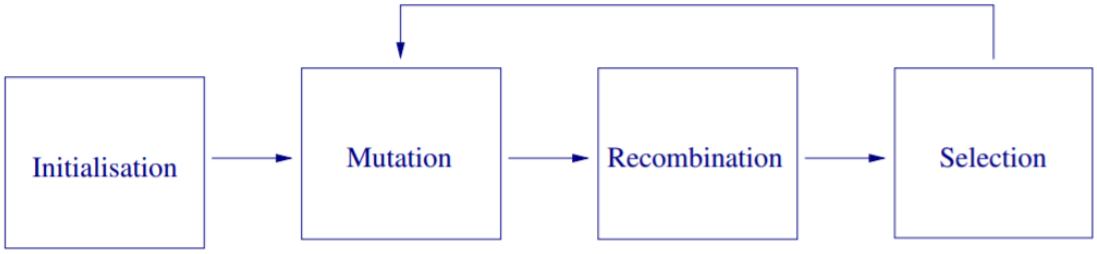


Figure 1.1: The main steps of the DE algorithm.

First, consider again an objective function $E : \Omega \subset \mathbb{R}^n \to \mathbb{R}$ whose search domain Ω is non-empty and bounded in $\mathbb{R}^n$. The DE algorithm starts with a population of vectors in Ω, let's say that the size of this population is N_p. We denote $X_0 = \left\{X_{0,j}\right\}_{j=1}^{N_p}$ the initial population. Let's say that each vector $X_{0,j} \in \Omega$ is called an *agent*. Consider that we are working on an Euclidean space of dimension $D \leq n$, then each agent is fully described by their $D-$components. In mathematical formula, we have thus that $X_{0,j} = X\left\{0, j, i\right\}_{i=1}^{D}$ and where $j = 1, ..., N_p$. To resume, let's denote $X_{k,j,i}$ the i-th *component* of the j-th *agent* of the k-th *population* (=the population at iterate k). We are now ready to start the four steps.

Initialization. DE begins with a randomly assigned population. That is, we choose randomly N_p agents in Ω and we construct by this a population X_0 of size N_p.

Mutation. The mutation step perturbs each agent with the scaled difference of two or more randomly selected agents. That is, for each agent of the population $X_{k,j}$ where $j = 1, ..., N_p$, we select randomly five agents among all agents of the population X_k, let's denote these five random agents by $X_{k,rand1}, X_{k,rand2}, ..., X_{k,rand5}$. At this stage we are going to construct a new population of N_p agents, let's denote this new population by $V_{k,j}$. Each agent of this new population is constructed according to a linear combination of agents $X_{k,rand1}, X_{k,rand2}, ..., X_{k,rand5}$ that are chosen randomly among the population. There exists many different schemes for this linear combination that are all described in Table 1.2. Note that the scalars λ and μ are parameters that we have to determine. According to [8, 9], we choose $\lambda \in (0.3, 0.9)$ and $\mu \in (0.3, 0.9)$. It has also been found that selecting λ from the interval $(0.5, 1)$ randomly for each difference vector, improves the convergence behaviour significantly, especially for noisy objective functions [10].

Name of the scheme	Mathematical formula
DE/rand/1	$V_{k,j} = X_{k,rand1} + \lambda(X_{k,rand2} - X_{k,rand3})$
DE/best/1	$V_{k,j} = X_{k,best} + \lambda(X_{k,rand2} - X_{k,rand3})$
DE/rand to best/1	$V_{k,j} = X_{k,best} + \lambda(X_{k,rand2} - X_{k,rand3}) + \mu(X_{k,best} - X_{k,rand1})$
DE/curr. to rand/1	$V_{k,j} = X_{k,j} + rand(X_{k,rand1} - X_{k,j}) + \lambda(X_{k,rand2} - X_{k,rand3})$
DE/curr. to best/1	$V_{k,j} = X_{k,j} + \lambda(X_{k,rand2} - X_{k,rand3}) + \mu(X_{k,best} - X_{k,j})$
DE/rand/2	$V_{k,j} = X_{k,rand1} + \lambda(X_{k,rand2} - X_{k,rand3}) + \lambda(X_{k,rand4} - X_{k,rand5})$

Table 1.2: Different schemes for the mutation step of the DE algorithm.

Recombination. We construct again a new population of N_p agents that is denoted by $U_{k,j}$ and that is constructed by copying components from the original and the mutative agents to enhance the diversity of the population. We then have the following notation:

$$U_{k,j,i} = \begin{cases} V_{k,j,i} & \text{if } (rand_{k,j,i} \leq CR \quad \text{or } i_{rand} = i) \\ X_{k,j,i} & \text{otherwise} \end{cases} \tag{1.2}$$
$$i = 1, ..., D; j = 1, ..., N_p$$

where $CR \in [0, 1]$ is the crossover rate and i_{rand} is a random index ranging in $[1, D]$ to ensure that $U_{k,j}$ gets at least one element from $V_{k,j}$. We say that the constructed agent $U_{k,j}$ is the competitor of the original agent $X_{k,j}$. From [10], we choose for the classical setting and impose $CR = 0.8$.

Selection. At this stage, we select the next iterate for each agent. For each agent $X_{k,j}$, we determine if its corresponding competitor $U_{k,j}$ is better or not. If it is, we assign the next iterate $X_{k+1,j}$ to $U_{k,j}$. If it isn't, we keep the agent $X_{k,j}$ for the next iteration. This gives the following notation:

$$X_{k+1,j} = \begin{cases} U_{k,j} & \text{if } E(U_{k,j}) \leq E(X_{k,j}) \\ X_{k,j} & \text{otherwise} \end{cases}$$
$$j = 1, ..., N_p$$

Note that throughout the algorithm, the size of the populations created at the mutation and recombination step have always the same size as the original population.

Termination condition. As for Algorithm 1, we could imagine that the termination criterion would be a maximum number of cost evaluations.

An illustration of these main steps for one agent is given by Figure 1.2. In this illustration, we consider an objective function $E : \mathbb{R}^2 \to \mathbb{R}$ that is represented by the red ellipsoid curves and has a global minimum in the middle of the smallest ellipsoid curve. We wish to find this global minimum with the DE algorithm. The steps for one agent are described in order from (a) to (e).

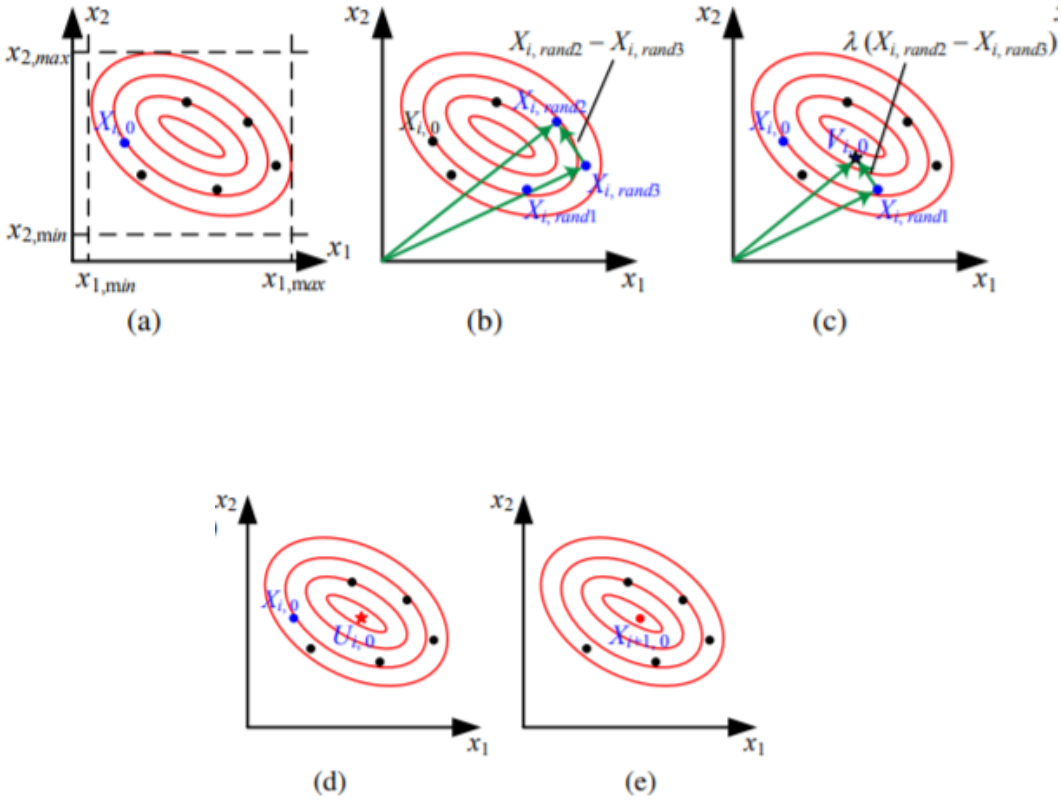


Figure 1.2: Illustration of the main steps of the DE method for one agent of the population i. (a) We are given the population i of $N_p = 6$ agents and we consider the agent $X_{i,0}$. (b) We select randomly three agents among the 6 agents. These agents are denoted by $X_{i,rand1}, X_{i,rand2}$ and $X_{i,rand3}$ (in blue). (c) According to the scheme DE/rand/1 given in Table 1.2, we construct a new agent $V_{i,0}$. (d) We do the recombination step described by the system 1.2 (here $D = 2$). We obtain the competitor agent $U_{i,0}$ (the red star) of the original agent $X_{i,0}$. (e) It seems that the competitor agent has smaller objective value, and thus we assign $X_{i+1,0}$ to $U_{i,0}$ (Figure in [3]).

The pseudocode of the DE algorithm is given by Algorithm 2.

Algorithm 2 Differential Evolution Algorithm

Input: Problem, objective function $E(x)$, $max\_costevals$, **MutationScheme()**, N_p, CR
Output: x_{best}, E_{best};
$X_0 \leftarrow$ **CreateInitialPop**$(Problem, N_p)$;
$costs\,\{j\} \leftarrow E(X_{0,j})$;
$[E_{best}, index] \leftarrow min(costs)$;
$x_{best} \leftarrow X_{0,index}$;
$costevals = N_p$;
$k = 0$;
while $costevals < max\_costevals$ **do**
  for j = 1, ..., N_p **do**
    $rand1 \leftarrow random(1 : N_p)$;
    $rand2 \leftarrow random(1 : N_p)$;
    $rand3 \leftarrow random(1 : N_p)$;
    $rand4 \leftarrow random(1 : N_p)$;
    $rand5 \leftarrow random(1 : N_p)$;
    $V_{k,j}$=**MutationScheme(**$X_{k,j}, X_{k,rand1}, X_{k,rand2}, X_{k,rand3}, X_{k,rand4}, X_{k,rand5}$**)**;
    $randi \leftarrow random(1, D)$;
    for i = 1, ..., D do
      $randnumber \leftarrow random(0, 1)$;
      if $randnumber \leq CR$ **or** $randi = i$ **then**
        $U_{k,j,i} \leftarrow V_{k,j,i}$;
      else
        $U_{k,j,i} \leftarrow X_{k,j,i}$;
      end if
    end for
    $U\_cost_{k,j} \leftarrow E(U_{k,j})$;
    $costevals = costevals + 1$;
    if $U\_cost_{k,j} < costs\,\{j\}$ **then**
      $X_{k+1,j} \leftarrow U_{k,j}$;
      $cost\,\{j\} \leftarrow U\_cost_{k,j}$;
      if $cost\,\{j\} < E_{best}$ **then**
        $X_{best} \leftarrow X_{k+1,j}$;
        $E_{best} \leftarrow cost\,\{j\}$;
      end if
    else
      $X_{k+1,j} \leftarrow X_{k,j}$
    end if
  end for
  $k = k + 1$;
end while

1.2.2 Design choices and parameters

As one can observe in Algorithm 2, there are some parameters that we have to define and we have to make some design choices. Table 1.3 gives a review of the terms present in the DE algorithm.

Problem	Description of the space in which we are treating the problem. We suppose here that we are on the Euclidean space $\mathbb{R}^n$ but afterwards, we will be working on manifolds.
$E(x)$	Objective function: $E : \mathbb{R}^n \to \mathbb{R}$.
$max\_costevals$	maximum number of evaluations of the objective function allowed.
MutationScheme()	One of the schemes described in Table 1.2.
CreateInitialPop$(Problem, N_p)$	This function returns an initial population of size N_p depending on the Problem. For the Euclidean space, we could imagine to simply generate N_p random vectors in $\mathbb{R}^n$.
$costevals$	Counts the number of function evaluations. Once we have that $costevals \geq max\_costevals$, we stop the algorithm.
CR	Crossover probability and has a value in $[0, 1]$. If the crossover probability is close to 1, we accept most of the components of the newly generated agent $V_{k,j}$ given to the competitor agent $U_{k,j}$.

Table 1.3: Review of the terms present in the DE algorithm

Concerning the convergence properties and cooling schedules of DE, we refer to appendix A.2.

1.2.3 Improved Differential Evolution

In this section, we present an improved version of the DE algorithm based on two strategies from two different papers. In [11], a novel method called *composite* DE is built and compared with the classical DE method, the difference is that we apply a specific strategy concerning the choice of the mutation scheme. In [12], a new algorithm is developed using the stochastic properties of chaotic systems to spread the individuals as much as possible and the global convergence is proved. We will firstly present the improvement with the *composite* DE. Secondly, we describe the improvement with chaotic systems. Finally, the final improved DE (we call it CDE) is obtained by combining these two improved methods.

***composite* DE**

As explained in the previous sections, the mutation schemes have a significant influence on the performance of DE. In [11], one studies whether the performance of DE can be improved by combining several mutation schemes with some suitable control parameters. The main step concerns the choice of the mutation scheme given by the function **MutationScheme()** in Algorithm 2. The improved method simply chooses randomly one of the mutation schemes in Table 1.2 for each agent. Experimental results show that the this improved method is very competitive and that its overall performance was better compared to the state-of-the-art DE variants cited in [11].

Chaotic strategy for DE

This improved method is proposed in order to spread the individuals in search spaces as much as possible. By examining the traditional DE, we find that there is no mechanism in the algorithm to extract and use global information about search space and the evolutionary scheme to repeat computation on some previously detected points. DE is in its infancy and can most probably be improved. Chaos is a kind of characteristic of nonlinear systems, which has been extensively studied and applied in many fields. In order to improve the global performance of DE, the chaotic mappings are used to initialize and reinitialize population so that search space information can be extracted and used. We present two chaotic systems namely logistic map [13] and mapping drawn from chaotic neuron [14] , which are defined in 1.3 and 1.4.

$$ch_{k+1} = \mu ch_k(1 - ch_k) \tag{1.3}$$

$$ch_{k+1} = \nu ch_k - 2\tanh(\gamma ch_k)\exp\big[-3(ch_k)^2\big] \tag{1.4}$$

where $k = 0, 1, 2, ...K$ is the iteration counter, K is the preset maximum number of chaotic iterations, μ and γ are the control parameters, $\nu \in (0, 1)$ is a damping factor and ch_k is a self feedback. It has been proven in [13] that when we set $\mu = 4$ and $ch_0 \notin \{0, 0.25, 0.5, 0.75, 1\}$ for equations 1.3 and 1.4 and such that we avoid little period $ch_k = ch_{k-i}, i = \{1, 2, 3, 4\}$ for 1.4, then the mapped variable in equations 1.3 or 1.4 can distribute in search space with ergodicity, randomness and irregularity. This way, if an algorithm needs the points to be distributed in the search space as much as possible, the two chaotic systems can meet this need. For facility, we choose the chaotic system described by equation 1.3.

We will now present the algorithm of the CDE (Chaotic Differential Evolution) method shown in Algorithm 4. The initial step in Algorithm 4 is to initialize a chaotic population of N_p points with Algorithm 3 which will ensures the individuals in the returned population to be spread in the search space as much as possible. We determine the best agent of this population and assign its objective value to E_{best} in Algorithm 4. We can now begin looping in Algorithm 4. The first step in the while loop is to execute the DE algorithm (described in Algorithm 2) with a fixed maximum amount of cost evaluations.

The solution returned by DE is given by x^{DE}. The second step is applied for local search if the gradient is available, if not, we skip this step. Suppose at this stage that the gradient is available and the called gradient method returns a local minimum x^* (if the gradient wasn't available, we would simply have had $x^* = x^{DE}$). The third step constructs a chaotic population X_{sub} of size $N_p/2$ by calling Algorithm 3. We evaluate the objective value of each agent of this subpopulation. The fifth step replaces the worst agents of the current population by all the agents of the subpopulation. By $worsthalf$ we mean the half part of the agents that generates the worst objective values. Finally, we replace the worst agent with the solution returned in step 2.2.

Algorithm 3 Chaotically initializing population

Input: Problem, N_p, K, x_{min}, x_{max}
Output: X_0
$j = 1$;
while $j \leq N_p$ **do**
$k = 0$;

Step 1. Randomly initialize variable $ch_0^i \in [0, 1]$ where $ch_0^i \notin \{0, 0.25, 0.5, 0.75, 1\}$ and for $i = 1, ..., n$.

while $k < K$ **do**

Step 2. Generate different chaotic variables ch_k^i for $i = 1, 2, ..., n$ according to formula 1.3.

$k = k + 1$;
end while

Step 3. Mapping the chaotic variables ch_k^i to feasible region according to equation:

$$X_{0,j,i} = x_{min,i} + ch_k^i(x_{max,i} - x_{min,i})$$

$j = j + 1$;
end while

Algorithm 4 Chaotic Differential Evolution

Input: Problem, objective function $E(x)$, $max\_costevals$, $max\_costevals\_DE$, N_p, K, x_{min} x_{max}.
Output: x_{best}, E_{best}
Step 1. Chaotically initialize the population X_0 by calling Algorithm 3 with the parameters given as input arguments.
$costs\,\{j\} \leftarrow E(X_{0,j})$;
$[E_{best}, index] \leftarrow min(Costs)$;
$x_{best} \leftarrow X_{0,index}$;
$k = 0$;
$X_{current} \leftarrow X_0$;
while $k \leq max\_costevals$ **do**

Step 2.1. Call the DE method with maximum cost evaluations given by $max\_costevals\_DE$. The solution returned is x^{DE}.

$k = k + max\_costevals\_DE$;

Step 2.2. If the gradient of the objective function is available, call a gradient method with starting point x^{DE}. Assume a local minimum x^* is found. If the gradient is not available, skip this step and pose $x^* = x^{DE}$.

Step 2.3. Use the chaotically initializing population method (Algorithm 3 with input $K, N_p/2, x_{min}$ and x_{max}) to generate a chaotic subpopulation X_{sub} of size $N_p/2$.

Step 2.4. Evaluate the costs of each agent of this sub population.

$costs_{sub}\,\{j\} \leftarrow E(X_{sub,j})$;
$k = k + N_p/2$;

Step 2.5. Use X_{sub} to replace the worst half part of agents of the current population, while the best half part of agents is kept steady in the population.

$X_{current,worsthalf} \leftarrow X_{sub,j}$;
$costs\,\{worsthalf\} \leftarrow costs_{sub}$;

Step 2.6. Replace the worst agent of the current population with x^*.

$[\sim, index\_max] = \max(costs\,\{j\})$;
$X_{current,index\_max} \leftarrow x^*$;
$costs\,\{index\_max\} \leftarrow E(x^*)$;
end while

Note that the algorithm is very different when the gradient is not available. Indeed, if the gradient is not available, we have no choice to skip Step 2.2. A potential way out to avoid to skip the Step 2.2 is to call a method that does not require any knowledge about the gradient of the objective function and realises local search. In [11], the Hook Jeeves pattern search method is proposed for step 2.2. However, this method seems very difficult

to generalize for manifolds and we left this for further work. Consider from now on that we denote the classical Chaotic Differential Evolution by CDE. CDE refers thus to Algorithm 4 without step 2.2 (no knowledge about the gradient is thus needed). We denote AccCDE as the accelerated Chaotic Differential Evolution and refers thus to Algorithm 4 and requires a knowledge of the gradient.

Concerning the parameters x_{min} and x_{max}, these parameters are dependent of the problem search space. For example, consider the domain of the objective function $f : [-2, 2] \in \mathbb{R}^2 \to \mathbb{R}$. The domain of the function f corresponds thus to a square delimited by the points $\big\{[-2, -2], [-2, 2], [2, 2], [2, -2]\big\}$. Then, a possible parameter setting could be $x_{min} = [-2, -2]$ and $x_{max} = [2, 2]$.
However, assigning numerical values to x_{min} and x_{max} when we are working with manifolds, may be very difficult as explained in the next chapters.

To conclude, we impose in Algorithm 4 that when DE is called, it is called with the option of composite DE, that is, choose randomly one of the mutation scheme shown in Table 1.2 for each agent. By mixing these two concepts, we obtain an improved version of DE.

1.3 Accelerated version of the methods

In this section, we present an accelerated version of the methods that is similar both for SA as for DE. The goal is to combine gradient methods with stochastic methods like SA and DE. We require the objective function to be differentiable since we need to have an expression of the gradient to apply this accelerated method (we are no longer in the derivative-free case).
The main difficulty when we are treating global optimization problems arises from the number of minima of the objective functions, which usually increases exponentially with the size of the system or the complexity of the problem. Many gradient methods that works very well in local search may then get trapped in a local minimum and never reach a better local minimum. This is where stochastic methods come in, they are able to perform global search and they may switch from one local minimum to a better local minimum. On the contrary, gradient method are not able to perform this switch...
The challenge now is thus to mix the efficiency of gradient methods in local search with the efficiency of stochastic methods in global search. We first introduce the stretching function technique. After that, we present the accelerated method. The main source for this section is the paper [15].

1.3.1 Stretching function technique

The "stretching" technique relies on the concept of transforming the objective function in a way that the knowledge of the previously detected minimizers is incorporated in its new

form. This technique consists of a two-phase transformation of the objective function. Firstly, we consider that we have found a local minimum denoted by x^* with a gradient method for example. The first phase of the transformation makes all the local minima with values higher than the local minimum x^* disappear by stretching the objective function $f(x)$ upwards. This first transformation is given by:

$$G(x) = E(x) + \lambda_1 \|x - x^*\| \big(\text{sign}(E(x) - E(x^*)) + 1\big) \tag{1.5}$$

where $E : \mathbb{R}^n \to \mathbb{R}$ is a real-valued objective function.
The parameter λ_1 controls the upward stretching of the objective function through the transformation $G(x)$. We set the value of λ_1 to 10^3 as in [15].

The second stage of the transformation stretches the neighborhood of x^* and changes the detected minimum to a maximum. However, the minima with lower values of the objective function remain unaffected by the transformation:

$$H(x) = G(x) + \frac{\lambda_2 \big(\text{sign}(E(x) - E(x^*)) + 1\big)}{2\tanh(\mu(G(x) - G(x^*)))} \tag{1.6}$$

The parameters λ_2 and μ determine the range of the effect and magnitude of the elevation, respectively. We set the value of λ_2 and μ to 1 and 10^{-8} respectively as in [15].

To illustrate the stretching technique, we consider the objective function $E : \mathbb{R}^2 \to \mathbb{R}$ as follows:

$$E(x_1, x_2) = \sum_{i=1}^{5} i cos[(i+1)x_1 + i] \sum_{j=1}^{5} i cos[(j+1)x_2 + j] + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2 + 200 \tag{1.7}$$

this is knwon as the Levy function No. 5 [16] and its domain is in the cube $[-2, 2]^2$. This function has a lot of local minima and the global minimum is at $x^*_{global} = [-1.425, -0.8]$ and his objective value is $E(x^*_{global}) = 13.2691$. We suppose that we have found a local minimum at $x^* = [0.3, -0.3]$ with $E(x^*) = 176.863$. An illustration of this objective function is given by Figure 1.3.

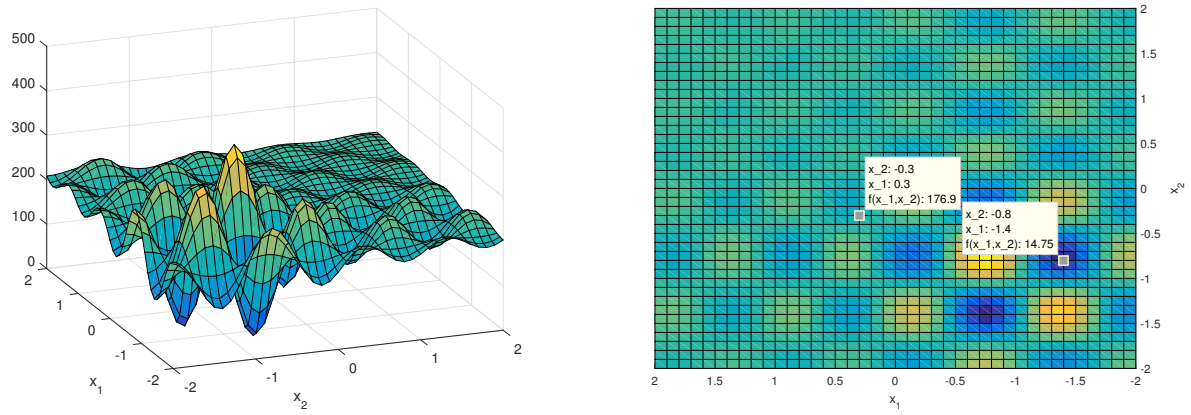


Figure 1.3: Illustration of the objective function given by equation 1.7. The global minimum is situated at $x^*_{global} = [-1.425, -0.8]$ and the current local minimum found so far by a gradient method is situated at $x^* = [0.3, -0.3]$.

After the transformation $G(x)$ given by equation 1.5, we obtain the modified objective function illustrated at Figure 1.4.

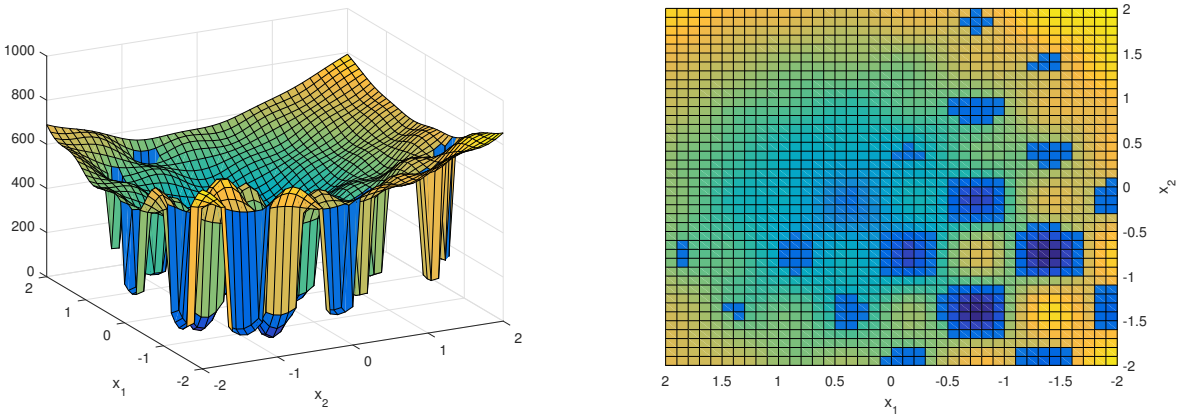


Figure 1.4: Illustration of the effect of the first transformation $G(x)$ given by equation 1.5. All minima with functional values higher than x^* are stretched upwards. One can clearly observe the effect of the transformation compared to Figure 1.3.

After the transformation $H(x)$ given by equation 1.6, we obtain the modified objective function illustrated at Figure 1.5.

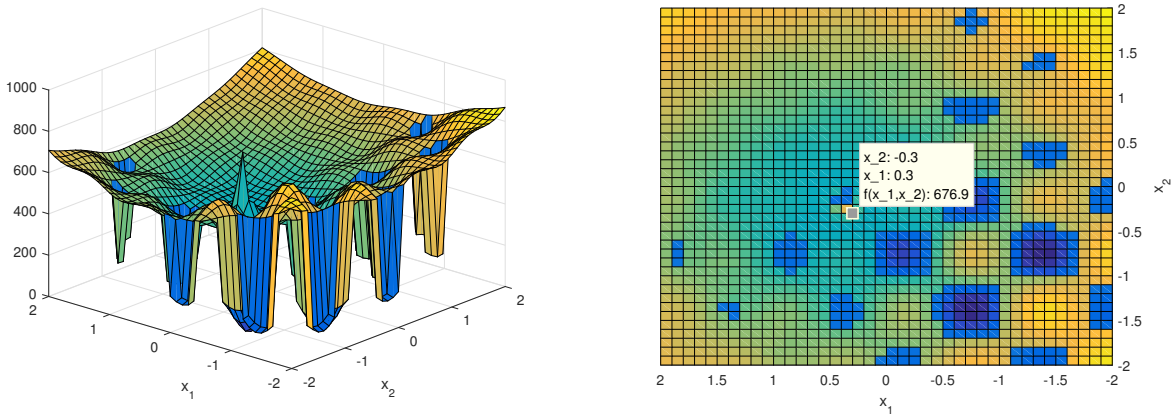


Figure 1.5: Illustration of the effect of the second transformation $H(x)$ given by equation 1.6. The current local minimum $x^* = [0.3, -0.3]$ is transformed to a maximizer, while the minima with lower function values remain unaffected.

1.3.2 General description and pseudocode

We will now present the accelerated method. As said before, we will mix an efficient method for local search like a gradient method with a stochastic method that is efficient for global search.

Gradient methods. Gradient based algorithms are local deterministic methods which can find a stationary point near the initial one efficiently. There are many efficient gradient descent methods for finding the local minima of functions like steepest descent method, Newton method, quasi-Newton method, trustregion method... Generally speaking, these methods converge faster and can obtain a solution with precision higher than stochastic approaches. However, the gradient method rely heavily on the initial point we give to the algorithm as we will show in chapter 3. It is hard to ensure convergence to the global minimum by simply using these methods. This is where stochastic methods come in.

In the accelerated method, we firstly use a gradient method to find a local minimum x^*. Secondly, the obtained local minimum x^* is used to construct the "stretching" function with the transformations $G(x)$ and $H(x)$ given by equations 1.5 and 1.6 respectively. Then, a stochastic method (SA or DE) is used with the "stretched" function $H(x)$ as objective function. The stochastic method is called iteratively until a better solution x^*_{better} is found. Finally, once the stochastic method finds x^*_{better}, we take x^*_{better} as initial point for the gradient method and repeat the above steps.

The pseudo code is given by Algorithm 5.

Algorithm 5 Accelerated method

Input: Problem, objective function $E(x)$, **GradMethod()**, **StochMethod()**
Output: x_{best}, E_{best};
$x_0 \leftarrow$ **CreateInitialSol**$(Problem)$;
$k = 0$;
Set $\bar{\epsilon}$ a random vector whose entries are in $[-1, 1]$ and the dimension of this vector are coherent with the dimension of x_0;
Phase1: (local search) $x^{(k*)}$=**GradMethod**(x^k);
Phase2: Construct the "stretching" function $H(x)$ with the local minimum $x^{(k*)}$;
Phase3: Perturb the solution x^{k*}:

$$\bar{x} = x^{(k*)} + \bar{\epsilon}$$

Phase4: (global search) Call the **StochMethod**$(\bar{x})$ with $H(x)$ as objective function and with starting point $\bar{x}$;
Phase5: Denote the solution $x^{(k+1)*}$ as the solution returned in **Phase4**;
if $E(x^{(k+1)*}) \leq E(x^{(k*)})$ **then**
  $x^{(k+1)} = x^{(k+1)*}$;
  $k = k + 1$;
  return to **Phase1**;
else
  $\bar{\epsilon} = 2\bar{\epsilon}$;
  return to **Phase3**;
end if
Phase5: If a certain stopping criterion is met, stop the algorithm and set $x_{best} = x^k$ and $E_{best} = E(x_{best})$.

Let's describe each phase of the algorithm one by one. Note that we suppose to have total knowledge about the gradient and stochastic method, that is, we know the parameter setting for each method. The initialization phase consists in creating a (random) initial solution x_0 based on the description of the problem.

Phase1. The first phase consists in applying a gradient method with starting point x^k in order to find a local minimum x^{k*}.

Phase2. In the second phase, we construct the stretching function $H(x)$ based on the local minimum x^{k*} with equation 1.6.

Phase3. Here, we perturb the solution with a random vector $\bar{\epsilon}$ coherent to the dimension of x^{k*} and we obtain $\bar{x}$.

Phase4. We call a stochastic method (SA or DE) on the function $H(x)$ with starting

point $\bar{x}$. Note that there is a little difference between calling the SA or the DE method:

- **Simulated Annealing:** we simply call Algorithm 1 in section 1.1 with starting point $\bar{x}$. This means that we impose at the beginning of Algorithm 1 that $x_{current} = \bar{x}$.
- **Differential evolution:** This stochastic method needs to have a population of vectors (or *agents*) as input argument as precised in section 1.2. This is why we are going to construct a population of random vectors around $\bar{x}$ and then call Algorithm 2 with this population.

Phase5. If the stochastic method can find a better solution than the one found by the gradient method, return to **Phase1.**. Otherwise, we increase the perturbation applied to the the local minimum returned by the gradient method by doubling $\bar{\epsilon}$. This step will help us to discover new horizons of the space domain of the objective function and we may encounter a better local minimum or even the global minimum.

Phase6. We have to impose some stopping criteria. Firstly, we have to impose a maximum number of iterations for the gradient method. Similarly, we have to impose a maximum number of function evaluations for the stochastic method. Secondly, we define a counter that counts at each time we call the gradient method how many iterations the gradient method needed. To this counter, we add also the number of function evaluations needed by the stochastic method. Finally, if this counter exceeds some value defined by the user, the algorithm stops.

We give an illustration of one iterate of the accelerated DE method at Figure 1.6. On the figure, we consider a continuous real valued function $f : \mathbb{R} \to \mathbb{R}$ (in blue) and we wish to find its global minimum with accelerated DE.

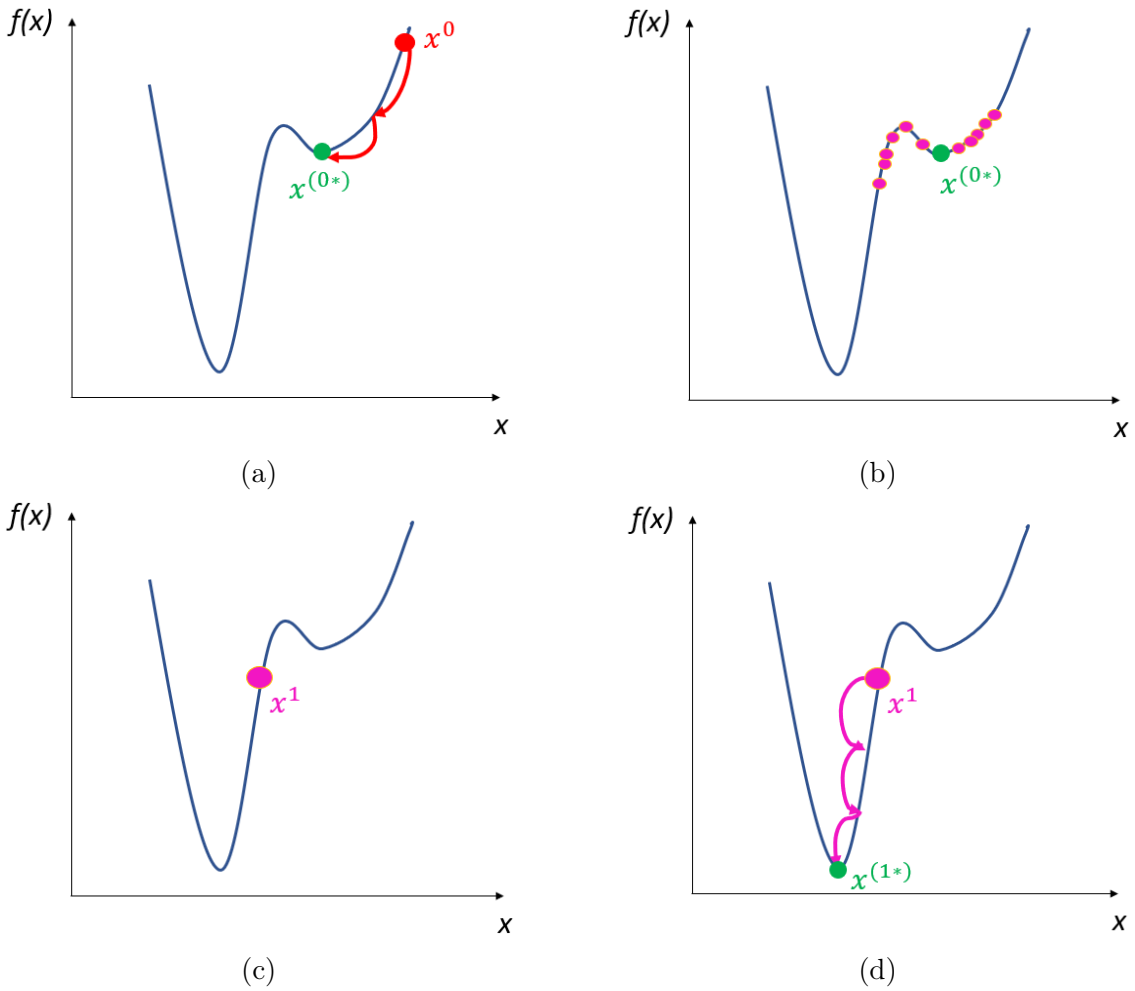


Figure 1.6: Illustration of the accelerated differential evolution algorithm. (a) We start the algorithm from x^0 (in red). We call the gradient method in **Phase1** and land on $x^{(0*)}$ (in green) which is a local minimum of the function $f(x)$. (b) We perturb the local minimum $x^{(0*)}$ as in **Phase3** and construct a population of points (in purple). This population of points is used as input argument for the DE method in **Phase4**. (c) The DE method returns x^1 (in purple) which, at first view, has a lower objective value compared to $x^{(0*)}$. (d) Since $f(x^1) < f(x^{(0*)})$, we return to **Phase1** to call again the gradient method and starting from x^1. The gradient method reaches the global minimum $x^{(1*)}$ (in green).

Chapter 2

Generalization of the novel methods on manifolds

Now that we got an overview of the novel methods in chapter 1, we have to generalize these methods on manifolds. Thanks to the Matlab toolbox for optimization on manifolds, also named Manopt, we had an overview of previously generalized methods on manifolds. Indeed, a lot of optimization methods that existed for a long time in the Euclidean space have already been generalized on manifolds like the trustregions, Steepest descent, Conjugate Gradient, BFGS, PSO and Neldermead method. Among these methods, PSO and Neldermead are the only ones that are derivative-free methods. Most of these methods were implemented on Manopt in 2012. Their implementation for generalization on manifolds are freely available on Manopt.org and that really helped a lot to get a first view on how Manopt works and how to implement the methods described in chapter 1.

This chapter deals with the main difficulties we had to challenge to implement the methods on manifolds. First, we will explain more in detail how manifolds are defined (and more specifically Riemannian manifolds). The second part will discuss some concepts on manifolds and the main difficulties that we faced for the implementation for each of the methods. Finally, we will introduce some additional improvements for the Stiefel manifold. Note that many concepts introduced in this chapter are directly taken from the book [4] that helps a lot to become familiar with the structure of manifolds. This chapter is therefore mainly a summary of notions that appears in [4] and that are useful for this master's thesis. The final step of the master thesis would be to add the implementations of these methods on Manopt to the ones that already have been generalized on manifolds. The main goal is thus to develop a package of the new methods and make it available on Manopt.

2.1 What are manifolds?

In mathematics, a n-dimensional manifold is a topological space[1] $\mathcal{M}$ for which every point $x \in \mathcal{M}$ has a neighbourhood homeomorphic to the Euclidean space $\mathbb{R}^n$. By homeomorphic we mean intrinsic topological equivalence. Two objects are homeomorphic if they can be deformed into each other by a continuous, invertible mapping. Thus, a square and a circle are homeomorphic to each other, but a sphere and a torus are not as illustrated in Figure 2.1.

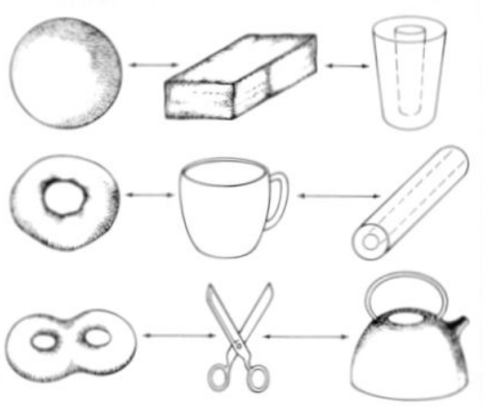


Figure 2.1: Illustration of homeomorphisms. By stretching and deforming, it is possible to turn a sphere into a cube or a beaker, but not into a torus. Sphere, ball, cube and beaker are all topologically equivalent. Figures having a single hole are all topologically equivalent to the doughnut-shaped torus. Another class contains two holes, and so on (Figure in [18]).

The concept of a manifold is central to many parts of geometry and modern mathematical physics because it allows complicated structures to be described and understood in terms of the simpler local topological properties of Euclidean space.

We firstly define the manifold in it's full generality. We then develop the notions of quotient manifold, tangent spaces and derivatives on manifolds. Finally, we introduce Riemannian manifolds to present some useful tools. Most of the definitions in this section come directly from [4, 5].

2.1.1 Definition of a manifold

A manifold is a topological space that locally resembles Euclidean space near each point. One-dimensional manifolds include lines and circles. Two-dimensional manifolds are also

[1] A topological space may be defined as a set of points, along with a set of neighbourhoods for each point, satisfying a set of axioms relating points and neighbourhoods. A more precise definition of a topological space can be found in [17].

called surfaces. Examples include the plane, the sphere, the torus ... which can all be embedded (formed without self-intersections) in three dimensional real space. The mathematical definition of a manifold relies on the concepts of charts and atlases.

Roughly speaking, a coordinate map, or a *chart*, of a manifold is an invertible map between a subset of the manifold and a simple space such that both the map and its inverse preserve the desired structure. For a topological manifold, the simple space is a subset of some Euclidean space $\mathbb{R}^n$ and interest focuses on the topological structure. This structure is preserved by homeomorphisms, invertible maps that are continuous in both directions. More mathematically, we define *charts* as follows:

Definition 1 (chart [4]). *Let $\mathcal{M}$ be a topological space and $\mathcal{U} \subseteq \mathcal{M}$ an open set. Let $\mathcal{V} \subseteq \mathbb{R}^n$ be open. A homeomorphism $\phi : \mathcal{U} \to \mathcal{V}, \phi(u) = (x_1(u), ..., x_n(u))$ is called a coordinate system on $\mathcal{U}$, and the functions $x_1, ...x_n$ the coordinate functions. The pair $(\mathcal{U}, \phi)$ is called a n-dimensional chart of the set $\mathcal{M}$.*

The interest of the notion of chart $(\mathcal{U}, \phi)$ is that it makes it possible to study objects associated with $\mathcal{U}$ by bringing them to the subset $\phi(\mathcal{U})$ of $\mathbb{R}^n$ as illustrated in Figure 2.2. Even though charts are a necessary ingredient to define manifolds, they are seldom used in practice when working on a specific manifold. The reason for this is that differential geometric tools are often coordinate-free. Coordinate-free means the choice of charts is irrelevant: only the ensuing manifold structure matters.

A specific collection of charts which covers a manifold is called an *atlas*. An atlas is not unique as all manifolds can be covered multiple ways using different combinations of charts.

Definition 2 (atlas [4]). *An atlas $\mathcal{A}$ on $\mathcal{M}$ is a collection of charts $\{(\mathcal{U}_\alpha, \phi_\alpha)\}$ such that $\mathcal{U}_\alpha$ cover $\mathcal{M}$ (that is $\bigcup_\alpha \mathcal{U}_\alpha = \mathcal{M}$) and for any pair α, β with $(\mathcal{U}_\alpha \cap \mathcal{U}_\beta) \neq \emptyset$, the homeomorphism $\phi_\beta \circ \phi_\alpha^{-1} : \phi_\alpha(\mathcal{U}_\alpha \cap \mathcal{U}_\beta) \to \phi_\beta(\mathcal{U}_\alpha \cap \mathcal{U}_\beta)$ which correspond to the coordinate transformation, is smooth [2] on its domain $\phi_\alpha(\mathcal{U}_\alpha \cap \mathcal{U}_\beta)$).*

Two atlases are said to be equivalent if their union is also an atlas. Figure 2.2 illustrates the concept of a chart and the coordinate transformation in Definition 2.

---

[2] We say that the mapping $\phi_\beta \circ \phi_\alpha^{-1} : \phi_\alpha(\mathcal{U}_\alpha \cap \mathcal{U}_\beta) \to \phi_\beta(\mathcal{U}_\alpha \cap \mathcal{U}_\beta)$ is smooth if $\phi_\beta \circ \phi_\alpha^{-1}$ is of class $\mathcal{C}^\infty$. ($\mathcal{C}^k$ means k-times continuously differentiable.)

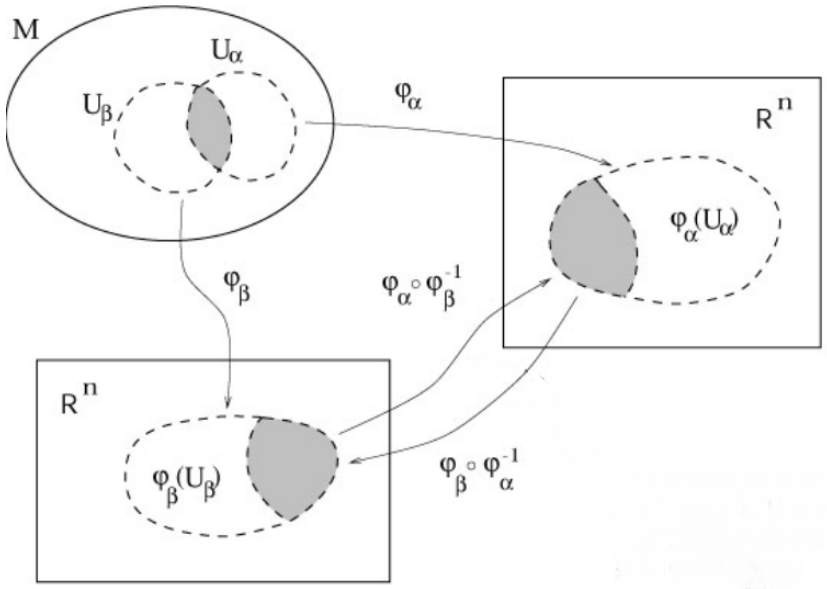


Figure 2.2: Illustration of the concept of a chart and the coordinate transformation. We consider a topological space $\mathcal{M}$ and $\mathcal{U}_\alpha, \mathcal{U}_\beta \subseteq \mathcal{M}$ two open sets such that $(\mathcal{U}_\alpha \cap \mathcal{U}_\beta) \neq \emptyset$. The pairs $(\mathcal{U}_\alpha, \phi_\alpha)$ and $(\mathcal{U}_\beta, \phi_\beta)$ are called charts of the set $\mathcal{M}$. The figure shows also two coordinate transformations $\phi_\beta \circ \phi_\alpha^{-1}$ and $\phi_\alpha \circ \phi_\beta^{-1}$ to go from one chart to the other. Suppose now that the open sets $\mathcal{U}_\alpha, \mathcal{U}_\beta$ would have covered the whole set $\mathcal{M}$, that is $\mathcal{U}_\alpha \cup \mathcal{U}_\beta = \mathcal{M}$ (which is not the case on the figure). Then, provided that the homeomorphisms $\phi_\beta \circ \phi_\alpha^{-1}$ and $\phi_\alpha \circ \phi_\beta^{-1}$ are smooth, we could have said that the collection of charts $\big\{(\mathcal{U}_\alpha, \phi_\alpha), (\mathcal{U}_\beta, \phi_\beta)\big\}$ is an atlas on $\mathcal{M}$. (Figure in [19], chapter on manifolds)

Let's define the concept of maximal atlas:

Definition 3 (maximal atlas [4])**.** *Given an atlas $\mathcal{A}$, then, the set of all charts $(\mathcal{U}, \phi)$ such that $\mathcal{A} \cup \big\{(\mathcal{U}, \phi)\big\}$ is also an atlas is called the maximal atlas (or complete atlas) generated by the atlas $\mathcal{A}$.*

Unlike an ordinary atlas, the maximal atlas of a given manifold is unique. Though it is useful for definitions, it is an abstract object and not used directly (e.g. in calculations).

We are now able to define a manifold in general:

Definition 4 (manifold [4])**.** *A n-dimensional manifold is a couple $\big\{(\mathcal{M}, \mathcal{A}^+)\big\}$, where $\mathcal{M}$ is a set and $\mathcal{A}^+$ is a maximal atlas of $\mathcal{M}$ into $\mathbb{R}^n$, such that the topology induced by*

$\mathcal{A}^+$ *is Hausdorff*[3] *and second-countable*[4].

To resume, given a manifold $\{(\mathcal{M}, \mathcal{A}^+)\}$, an atlas of the set $\mathcal{M}$ whose maximal atlas is $\mathcal{A}^+$ is called an atlas of the manifold $\{(\mathcal{M}, \mathcal{A}^+)\}$; a chart of the set $\mathcal{M}$ that belongs to $\mathcal{A}^+$ is called a chart of the manifold $\{(\mathcal{M}, \mathcal{A}^+)\}$.

Definition 5 (dimension)**.** *Given a manifold* $(\mathcal{M}, \mathcal{A}^+)$*, if all the charts of* $\mathcal{A}^+$ *have the same dimension* n*, then* $dim\mathcal{M} := n$ *is the dimension of the manifold.*

We introduce the concept of product manifold end embedded submanifold. These concepts will be useful for the further sections.

Product manifolds. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be manifolds of dimension d_1 and d_2, respectively. The set $\mathcal{M}_1 \times \mathcal{M}_2$ is defined as the set of pairs (x_1, x_2) , where x_1 is in $\mathcal{M}_1$ and x_2 is in $\mathcal{M}_2$. If $(\mathcal{U}_1, \phi_1)$ and $(\mathcal{U}_2, \phi_2)$ are charts of the manifolds $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively, then the mapping $\phi_1 \times \phi_2 : \mathcal{U}_1 \times \mathcal{U}_2 \to \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} : (x_1, x_2) \to (\phi_1(x_1), \phi_2(x_2))$ is a chart of the set $\mathcal{M}_1 \times \mathcal{M}_2$ which is called the *product* of the manifolds $\mathcal{M}_1$ and $\mathcal{M}_2$. Its manifold topology is equivalent to the product topology[5].

Embedded submanifold. Let $(\mathcal{N}, \mathcal{A})$ be a submanifold of $(\mathcal{M}, \mathcal{B})$. Since $\mathcal{M}$ and $\mathcal{M}$ are manifolds, they are also topological spaces with their manifold topology. If the manifold topology of $\mathcal{N}$ coincides with its subspace topology from the topological space $\mathcal{M}$, then $\mathcal{N}$ is called an *embedded submanifold* of the manifold $\mathcal{M}$. This definition of an embedded submanifold is rather formal, there exists some conditions in [4] to check whether a subset $\mathcal{N}$ of a manifold $\mathcal{M}$ is an embedded submanifold of $\mathcal{M}$ but we will not go further in detail.

---

[3] In topology, a Hausdorff space (or T_2 space) is a topological space in which distinct point have disjoint neighbourhoods. More precisely, suppose x, y two distinct points in a topological space $\mathcal{M}$. These two distinct points can be separated by neighbourhoods if there exists a neighbourhood $\mathcal{U}$ of x and a neighbourhood $\mathcal{V}$ of y such that $\mathcal{U} \cap \mathcal{V} = \emptyset$. We say that $\mathcal{M}$ is a Hausdorff space if all distinct points in $\mathcal{M}$ are pairwise neighborhood-separable. In fact, many spaces of use in analysis, such as topological groups and topological manifolds, have the Hausdorff condition explicitly stated in their definitions [17].

[4] In topology, a second-countable space, also called a completely separable space, is a topological space whose topology has a countable base. More explicitly, a topological space $\mathcal{T}$ is second-countable if there exists some countable collection $\mathcal{U} = \{U_i\}_{i=1}^{\infty}$ of open subsets of $\mathcal{T}$ such that any open subset of $\mathcal{T}$ can be written as a union of elements of some subfamily of $\mathcal{U}$ [17].

[5] By product topology, we mean the topology on the Cartesian product $\mathcal{M}_1 \times \mathcal{M}_2$ of two topological spaces whose open sets are the unions of subsets $\mathcal{U}_2 \times \mathcal{U}_2$, where $\mathcal{U}_1$ and $\mathcal{U}_2$ are open subsets of $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. For example, If one starts with the standard topology on the real line $\mathbb{R}$ and defines a topology on the product of n copies of $\mathbb{R}$ in this fashion, one obtains the ordinary Euclidean topology on $\mathbb{R}^n$ [4].

2.1.2 Quotient manifolds

We develop the theory of quotient manifolds because this concept will be used in chapter 3. It has several applications in matrix computations, most notably in algorithms that involve subspaces of $\mathbb{R}^n$. Some optimization problems may involve computations with subspaces. Usually, when we do computations with subspaces, this will be realized with the span of matrices that will represent the subspace. The difficulty is that for one given subspace, there are infinitely many matrices that represent the subspace. It is then desirable to partition the set of matrices into classes of "equivalent" elements that represent the same object. This leads to the concept of quotient spaces and quotient manifolds.

We firstly introduce the concept of immersion and submersion that will be useful to define a quotient manifold in a concise way.

Consider F a function from a manifold $\mathcal{M}_1$ of dimension d_1 into another manifold $\mathcal{M}_2$ of dimension d_2. Let x be a point on $\mathcal{M}_1$. Suppose we choose the charts $(\mathcal{M}_1, \phi_1)$ and $(\mathcal{M}_2, \phi_2)$ around x and $F(x)$. The coordinate representation of F is given by the function:

$$\hat{F} = \phi_2 \circ F \circ \phi_1^{-1} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$$

We say that F is *differentiable* or *smooth* at x if $\hat{F}$ is of class C^∞ at $\phi_1(x)$. The rank of F at x is the dimension of the range of $\mathrm{D}\hat{F}(\phi_1(x))[\cdot] : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ where $\mathrm{D}\hat{F}(\phi_1(x))[\cdot]$ denotes the *differential* of $\hat{F}$ at $\phi_1(x)$.

We say that the function F is called an *immersion* if its rank is equal to d_1 at each point of its domain (hence $d_1 \leq d_2$). If its rank is equal to d_2 at each point of its domain (hence $d_1 \geq d_2$), then it is called a *submersion*.

We now define the concept of *equivalence relation* $\sim$.

Definition 6 (equivalence relation [4])**.** *We say that a manifold $\mathcal{M}$ is equipped with an equivalence relation $\sim$ if the relation is*

- *reflexive: $x \sim x$ for all $x \in \mathcal{M}$.*
- *symmetric: $y \sim x$ if and only if $y \sim x$ for all $x, y \in \mathcal{M}$.*
- *transitive: if $x \sim y$ and $y \sim z$ then $x \sim z$ for all $x, y, z \in \mathcal{M}$.*

The set of all elements that are equivalent to a point x is called the equivalence class containing x and is thus defined as follows:

$$[x] := \{y \in \mathcal{M} : y \sim x\}$$

The set of all equivalence classes of $\sim$ in $\mathcal{M}$ is called the *quotient* of $\mathcal{M}$ by $\sim$ and is defined as follows:

$$\mathcal{M}/\sim := \big\{[x] : x \in \mathcal{M}\big\}$$

We are now able to define a *quotient manifold*:

Definition 7 (quotient manifold [4]). *Consider a manifold $(\mathcal{M}, \mathcal{A}^+)$ as defined in 2.1.1. The manifold $(\mathcal{M}/\sim, \mathcal{B}^+)$ where $\mathcal{B}^+$ is a manifold structure*[6] *on the set $\mathcal{M}/\sim$, is called a quotient manifold of $(\mathcal{M}, \mathcal{A}^+)$ if the natural projection* [7] *π is a submersion.*

Before illustrating the quotient manifold with an example, we describe some manifolds that are well known and implemented in Manopt.

Oblique manifold. The oblique manifold is a product of sphere. That is, $X \in \mathcal{M}$ if each column of X has unit 2-norm in $\mathbb{R}^n$. The manifold is described as follows:

$$\mathrm{Obl}(n,p) := \left\{ X \in \mathbb{R}^{n\times p} | \mathrm{diag}(X^T X) = I_p \right\}$$

In [5], it is shown how independent component analysis can be cast on this manifold as non-orthogonal joint diagonalization.

Fixed-rank elliptope. When only the product of $Y = X^T X$ matters (with X is an oblique manifold), matrices of the form QX are equivalent for all orthogonal matrix Q. Quotienting out this equivalence relation yields the fixed-rank elliptope:

$$\mathrm{Elip}(n,p) := \left\{ Y \in \mathbb{R}^{p\times p} | Y = Y^T \succeq 0, \mathrm{rank}(Y) = n, \mathrm{diag}(Y) = I_p \right\}$$

Stiefel manifold. This manifold is used for many applications like principal component analysis and dimension reduction [20], Lyapunov exponent computation [21], data analysis and image processing [22, 23] or blind source separation [24] and will appear in an optimization problem in chapter 3. Let $\mathrm{St}(n,p)$ where $p \leq n$ be The set of all $n \times p$ orthonormal matrices:

$$\mathrm{St}(n,p) := \left\{ X \in \mathbb{R}^{n\times p} | X^T X = I_p \right\}$$

where I_p denotes the $p \times p$ identity matrix. The set $\mathrm{St}(p,n)$ is called a *Stiefel manifold.*

Remark that for $p = 1$, the Stiefel manifold reduces to the unit sphere S^{n-1} in $\mathbb{R}^n$. The Stiefel manifold is also an embedded submanifold of $\mathbb{R}^{n\times p}$, its topology is the subset induced by $\mathbb{R}^{n\times p}$.

Grassmann manifold. Let $\mathrm{Grass}(p,n)$ denote the set of all p-dimensional subspaces of $\mathbb{R}^n$:

$$\mathrm{Grass}(n,p) := \left\{ \mathrm{span}(X) | X \in \mathbb{R}^{n\times p} : X^T X = I_p \right\}$$

[6] A manifold structure on $\mathcal{M}$ is a maximal atlas of a set $\mathcal{M}$ that induces a second-countable Hausdorff topology.

[7] The mapping $\pi : \mathcal{M} \to \mathcal{M}/\sim$ defined by $x \to [x]$ is called the natural projection.

We will now develop a correspondence between $\text{Grass}(p, n)$ and a quotient manifold of $\mathbb{R}^{n\times p}$.

Consider firstly the mapping:

$$\text{col} : \text{St}(n, p) \to \text{Grass}(n, p) : U \to \text{col}(U) = \text{the column space spanned by} \quad U$$

This mapping is a submersion. Note that the spanned column space is invariant by rotation of $\mathbb{R}^n$ (Leichtweiss, 1961). The submersion col induces an equivalence relation such that U and U' are equivalent if $\text{col}(U) = \text{col}(U')$, that is, if U and U' represent the same column space. Let

$$\text{O}(p) = \left\{ Q \in R^{p\times p} : Q^T Q = I_p \right\}$$

denote the set of $p \times p$ orthogonal matrices. Since U and U' are equivalent if and only if there exists some $Q \in \text{O}(p)$ such that $U' = UQ$, we say that $\text{Grass}(n, p)$ is a quotient of $\text{St}(n, p)$ by the action of $\text{O}(p)$:

$$\text{Grass}(n, p) = \text{St}(n, p)/\text{O}(r)$$

2.1.3 Tangent spaces and derivatives

We are now interested in the tangent spaces and derivative forms of a certain function f on a manifold. In particular, we still would like to "do calculus" on our manifold and have good notions of curves, tangent vectors, differential forms. The small drawback with the more general approach is that the definition of a tangent vector is more abstract. We can still define the notion of a curve on a manifold, but such a curve does not live in any given $\mathbb{R}^n$, so it is not possible to define tangent vectors in a simple-minded way using derivatives.

We firstly denote a curve on a manifold $\mathcal{M}$ as a smooth mapping $\gamma : \mathbb{R} \to \mathcal{M} : t \to \gamma(t)$. Suppose we want to have the expression of the tangent vector at $\gamma(t) \in \mathcal{M}$ of a certain smooth real-valued function f on $\mathcal{M}$. Note that the function $f \circ \gamma : \mathbb{R} \to \mathbb{R} : t \to f(\gamma(t))$ is a smooth function with a well-defined classical derivative. We deduce the following definition for a tangent vector:

Definition 8 (tangent vector [4])**.** *A tangent vector ξ_x to a manifold $\mathcal{M}$ at a point x is a mapping such that there exists a curve γ on $\mathcal{M}$ with $\gamma(0) = x$ that satisfies:*

$$\xi_x f := \left. \frac{df(\gamma(t))}{dt} \right|_{t=0}$$

where $f : \mathcal{M} \to \mathbb{R}$ is a smooth real-valued function. We say that such a curve γ realize the tangent vector ξ_x.

The *tangent space* to $\mathcal{M}$ at x, denoted by $T_x\mathcal{M}$ is the set of all tangent vectors to $\mathcal{M}$ a x and is illustrated on Figure 2.3.

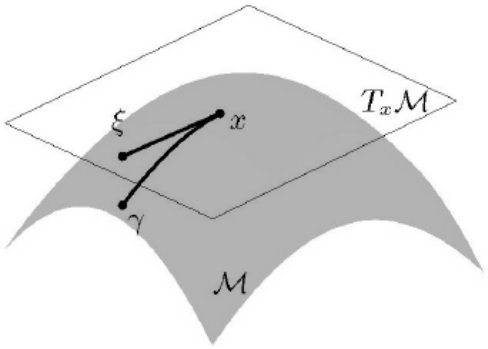


Figure 2.3: Illustration of the tangent space $T_x\mathcal{M}$ at x on a manifold $\mathcal{M}$. The vector ξ denotes the tangent vector realized by the curve $\gamma(t)$ (Figure in [25]).

Suppose now $F : \mathcal{M} \to \mathcal{N}$ be a smooth mapping between two manifolds $\mathcal{M}$ and $\mathcal{N}$. Let ξ_x be a tangent vector at a point x of $\mathcal{M}$. One can show that $\mathrm{D}F(x)[\xi_x]$ is a tangent vector at x on $\mathcal{N}$ ass illustrated on Figure 2.4 The tangent vector $\mathrm{D}F(x)[\xi_x]$ is realized by $F \circ \gamma$, where γ is any curve that realizes ξ_x. The mapping

$$\mathrm{D}F(x) : T_x\mathcal{M} \to T_{F(x)}\mathcal{N} : \xi \to \mathrm{D}F(x)[\xi]$$

is a linear mapping called the *differential* of F at x.

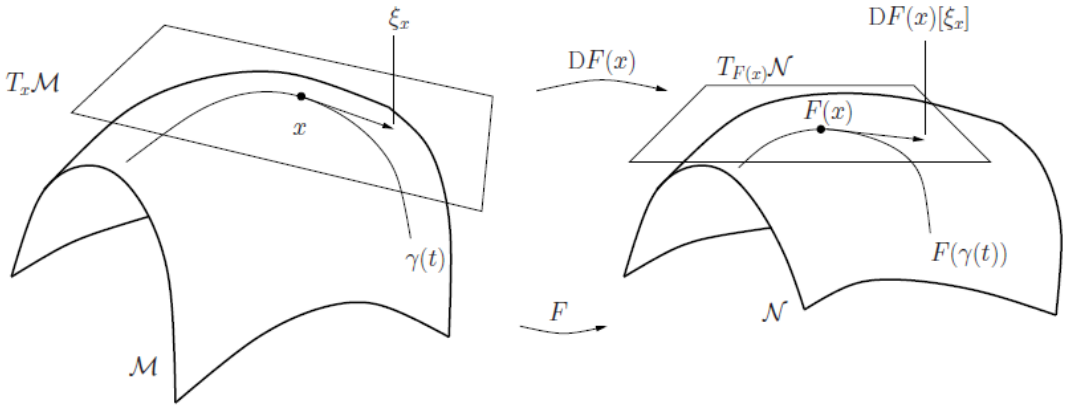


Figure 2.4: Differential mapping of F at x (Figure 3.5 in [4]).

To end this part, we refer to appendix B.1 that develops the equation of the tangent space for a sphere and a Stiefel manifold.

2.1.4 Riemannian manifolds

To measure distances and angles on manifolds, we refer to Riemannian manifolds. All differentiable manifolds can be given the structure of a Riemannian manifold. The Euclidean space itself carries a natural structure of Riemannian manifold. We will first introduce Riemannian metric and give a formal definition of Riemannian manifolds.

Definition 9 (metric [4])**.** *A Riemannian metric on a smooth manifold $\mathcal{M}$ is a 2-tensor field $g \in \mathcal{T}^2(\mathcal{M})$ that is symmetric ($g(X,Y) = g(Y,X)$), bilinear positive definite ($g(X,X) > 0$ if $X \neq 0$).*

A Riemannian metric thus determines an inner product on each tangent space $T_x\mathcal{M}$ which is typically written $g(\xi_x, \zeta_x) = \langle\, \xi_x, \zeta_x \rangle_x$ where $\xi_x, \zeta_x \in T_x\mathcal{M}$.

With this definition of a Riemannian metric, we can deduce the definition of a Riemannian manifold:

Definition 10 (Riemannian manifold [4])**.** *A manifold $\mathcal{M}$ whose tangent space at x $T_x\mathcal{M}$ is endowed with a smoothly varying inner product $\forall x \in \mathcal{M}$, is called a Riemannian manifold.*

To resume, a Riemannian manifold is thus a couple $(\mathcal{M}, g)$ where $\mathcal{M}$ is a manifold and g is a Riemannian metric on $\mathcal{M}$. Nevertheless, when the Riemannian metric is unimportant or clear from the context, we simply talk about "the Riemannian manifold $\mathcal{M}$". In the further sections and chapters, we will always consider that the concerned manifolds are Riemannian manifolds. The implementation for the inner product in Manopt is already done for each manifold that is represented in Manopt.

A Riemannian metric (tensor) makes it possible to define various geometric notions on a Riemannian manifold, such as a norm, angles, lengths of curves, areas (or volumes), curvature, gradients of functions and divergence of vector fields. Let's discuss some of these terms.

- The inner product $\langle\, \cdot, \cdot \rangle_x$ induces a norm on $T_x\mathcal{M}$,

$$\|\xi_x\|_x := \sqrt{\langle\, \xi_x, \xi_x \rangle_x}$$

  We can also define the angle between two nonzero vectors to be the unique $\theta \in [0, \pi]$ satisfying:

$$cos\theta = \langle\, \xi_x, \zeta_x \rangle_x / (\|\xi_x\| \|\zeta_x\|)$$

  We say that ξ_x and ζ_x are orthogonal if their angle is $\pi/2$, or equivalently if $\langle\, \xi_x, \zeta_x \rangle_x = 0$.

- The length of a curve $\gamma : [a, b] \to \mathcal{M}$ on a Riemannian manifold $(\mathcal{M}, g)$ is defined by:

$$L(\gamma) = \int_a^b \sqrt{g(\dot{\gamma}(t), \dot{\gamma}(t)}dt$$

- The Riemannian distance on a connected Riemannian manifold $(\mathcal{M}, g)$ is

$$\text{dist} : \mathcal{M} \times \mathcal{M} \to \mathbb{R} : \text{dist}(x, y) = \inf_\Gamma L(\gamma)$$

where Γ is the set of all curves in $\mathcal{M}$ joining points x and y.

2.2 The novel methods on manifolds

This section will present the adaptations of the methods presented in chapter 1 in order to be generalized on manifolds. All kind of manifold structures are already implemented in the matlab toolbox Manopt. The toolbox also provides a lot of tools on manifolds that were very useful. In this section, we will present some operators on manifolds that are used in the methods and well defined in Manopt.

2.2.1 Operators on manifolds

A lot of operators on manifolds are implemented in Manopt. The operators generally imply a relation between points on the manifold, between a point on a manifold and his tangent vector or even between vectors on the tangent space and a vector from the ambient or total space. We will describe the most important operators that appear in Manopt. Consider here that x, y are two points on the manifold $\mathcal{M}$. The notation of the operators in Manopt is M.`operator()` where M stands for the concerned manifold $\mathcal{M}$.

- **Metric.** (M.`inner`(x, u, v)) Computes the inner product of two vectors u, v on the tangent space at x: $g : T_x\mathcal{M} \times T_x\mathcal{M} \to \mathbb{R} : (u, v) \to \langle u, v\rangle_x$.

- **Norm.** (M.`norm`(x, u)) Computes the norm of the vector u on the tangent space at x: $T_x\mathcal{M} \to \mathbb{R} : u \to \sqrt{\langle u, u\rangle_x}$.

- **Distance.** (M.`dist`(x, y)) Computes the Riemannian distance between x and y as defined in 2.1.4: $\mathcal{M} \times \mathcal{M} \to \mathbb{R} : (x, y) \to$ M.`dist`(x, y).

- **Retraction.** (M.`retr`(x, u, t)) Computes the point you reach by following the vector tu ($t \in \mathbb{R}$) starting at x: $T_x\mathcal{M} \to \mathcal{M} : tu \to$ M.`retr`(x, u, t). This operator is one of the most important especially for optimization methods that are based on the steepest descent. Indeed, the simplest approach to optimizing a differentiable function is to continuously translate a test point $x(t)$ in the direction of steepest descent, -grad $f(x)$, on the constraint set until one reaches a point where the gradient

vanishes. This can be done with the retraction operator. More precisely, we define a retraction as follows:

Definition 11 (retraction [4])**.** *A retraction on a manifold $\mathcal{M}$ is a smooth mapping R from the tangent bundle $T\mathcal{M}$ onto $\mathcal{M}$ with the following properties. Let R_x denote the restriction of R to $T_x\mathcal{M}$.*

- *$R_x(0_x) = x$, where 0_x denotes the zero element of $T_x\mathcal{M}$.*
- *With the canonical identification $T_0xT_x\mathcal{M} \approx T_x\mathcal{M}$, R_x satisfies*

$$DR_x(0_x) = idT_x\mathcal{M}$$

where $idT_x\mathcal{M}$ denotes the identity mapping on $T_x\mathcal{M}$.

An illustration of a retraction on $\mathcal{M}$ is given in Figure 2.5. Note that on Manopt, the exponential map denoted by the operator M.`exp`(x, u, t), does exactly the same job as the retraction operator except that the retraction is a cheaper proxy for the exponential map.

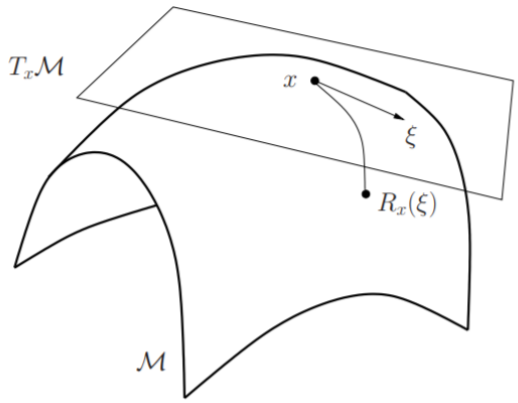


Figure 2.5: Illustration of a retraction. (Figure 4.1 in [4])

- **Tangent space projector.** (M.`proj`(x, u)) Computes the orthogonal projection of the vector u from the ambient or total space to the tangent space at x: $\mathbb{R}^n \to T_x\mathcal{M} : u \to \text{proj}(x, u)$.
- **Logarithmic map.** (M.`log`(x, y)) Computes a tangent vector at x pointing toward y: $\mathcal{M} \times \mathcal{M} \to T_x\mathcal{M} : (x, y) \to$ M.`log`(x, y). The computed tangent vector is in the same plane as the curve $\gamma(t)$ that realize the tangent vector and that links the two points x and y and is such that

$$\text{M.}\texttt{norm}(x, \text{M.}\texttt{log}(x, y)) = \text{M.}\texttt{dist}(x, y). \tag{2.1}$$

As is, this definition is not perfect. There might indeed be more than one eligible tangent vector at x pointing toward y and satisfying equation. For example, think of the sphere $\mathbb{S}^2$ and place x and y at the poles: for any vector $\xi \in T_x\mathbb{S}^2$ such that $\|\xi\| = \pi$, we have M.`retr`$\xi) = y$. For a more careful definition of the logarithm, see for example (do Carmo, 1992). As long as x and y are not "too far apart", this definition is satisfactory. Note that this operator does the inverse job compared with the retraction operator.

- **Random point.** (M.`rand()`) Computes a random point on the manifolds.
- **Vector transport.** (M.`transp`(x, y, u)) Computes a tangent vector at y that "looks like" the tangent vector u at x.

Most of these operators are defined for almost every manifold that appears in Manopt. However, for some manifolds like the Stiefel manifold for example, some operators are still not defined. In section 2.3, we will show an algorithm for completing a missing operator for the Stiefel manifold.

Concerning the convergence properties on manifolds, we refer to appendix B.2.

2.2.2 Adaptation of the algorithms

In this section we will describe what concretely changes in the algorithms described in chapter 1. Most of the changes implies the operators on manifolds defined in 2.2.1. Recall that we are dealing with a real-valued function $E : \mathcal{M} \to \mathbb{R}$.

Simulated Annealing. What changes in Algorithm 1, is the way we generate an initial solution and a neighbour solution to the current solution ($x_{current}$), described by the functions **CreateInitialSol**(Problem) and **CreateNeighbourSolution**($x_{current}$). We simply replace these functions by the random operator: **Problem.M.**`rand()`.

Differential Evolution. What changes in Algorithm 2, is again the way we generate the initial iterate **CreateInitialPop**(Problem, N_p). To construct an initial random population, we call N_p times the operator **Problem.M.**`rand()`.

The main adaptation we have to consider concerns the mutation scheme described by **MutationScheme**() (recall that there are different mutation schemes possible as given by Table 1.2). Indeed, for manifold settings, we can not "simply" add two points on the manifold and hoping that their addition will give again a point on the manifold. In the Euclidean space, this is not a problem since the addition or substraction of vectors in $\mathbb{R}^n$ is another vector in $\mathbb{R}^n$. For manifolds, we can not apply classical substractions and additions as in the Euclidean space so we have to completely redefine substractions and additions as mappings on manifolds.

In order to adapt the mutation schemes on a manifold $\mathcal{M}$, we have to think how to define an addition and a substraction between two points (or *agents*) on the manifold $\mathcal{M}$. We decided to represent the *substraction operator* and *addition operator* as in [3]. We consider two *agents* $x, y \in \mathcal{M}$, the curve $\gamma(x, y)$ between the two agents and finally the two operators defined as follows:

- *substraction operator.* We define the operator $\ominus(x, y) : \mathcal{M} \times \mathcal{M} \to T_x\mathcal{M}$ which takes two points x and y as input and produces a tangent vector at x. The magnitude of this tangent vector is the length of $\gamma(x, y)$ and its direction is the direction that points toward y. We can make the direct link with with the log operator defined in 2.2.1. Indeed, we have that:

$$\ominus(x, y) = \mathrm{M}.\texttt{log}(x, y) \tag{2.2}$$

  We have defined the substraction operator that can be applicable on Manopt.

- *addition operator.* We define the operator $\oplus(x, \vec{v}_y,) : \mathcal{M} \times T_y\mathcal{M} \to \mathcal{M}$ which takes a tangent vector $\vec{v}_y$ at y and a point x as input argument and produces a point $z \in \mathcal{M}$ which is computed as follows:

  – Step 1: parallel transport the tangent vector $\vec{v}_y$ from y to x along the curve $\gamma(x, y)$,
  – Step 2: return the point reached by following the vector $\vec{v}_y$ starting at x.

  We can make the direct link with the retraction and the transp operators defined in 2.2.1:

$$\oplus(x, \vec{v}_y) = \mathrm{M}.\texttt{retr}(x, \mathrm{M}.\texttt{trans}(y, x, \vec{v}_y)) \tag{2.3}$$

  Note that a random tangent vector at y can rapidly be computed as follows:

$$\vec{v}_y = \mathrm{M}.\texttt{proj}(y, u)$$

  where u is a random vector from the total or ambient space.

Table 1.2 from Chapter 1 now becomes Table 2.1.

Name of the scheme	Mathematical formula for manifolds
DE/rand/1	$V_{k,j} = X_{k,rand1} \oplus \lambda(X_{k,rand2} \ominus X_{k,rand3})$
DE/best/1	$V_{k,j} = X_{k,best} \oplus \lambda(X_{k,rand2} \ominus X_{k,rand3})$
DE/rand to best/1	$V_{k,j} = \big(X_{k,best} \oplus \lambda(X_{k,rand2} \ominus X_{k,rand3})\big) \oplus \mu(X_{k,best} \ominus X_{k,rand1})$
DE/curr. to rand/1	$V_{k,j} = \big(X_{k,j} \oplus rand(X_{k,rand1} \ominus X_{k,j})\big) \oplus \lambda(X_{k,rand2} \ominus X_{k,rand3})$
DE/curr. to best/1	$V_{k,j} = \big(X_{k,j} \oplus \lambda(X_{k,rand2} \ominus X_{k,rand3})\big) \oplus \mu(X_{k,best} \ominus X_{k,j})$
DE/rand/2	$V_{k,j} = \big(X_{k,rand1} \oplus \lambda(X_{k,rand2} \ominus X_{k,rand3})\big) \oplus \lambda(X_{k,rand4} \ominus X_{k,rand5})$

Table 2.1: Different schemes for the mutation step on manifolds. The operators $\oplus$ and $\ominus$ refers to operators on manifolds defined in equations 2.2 and 2.3.

The recombination step of the DE algorithm can not be applied on manifolds since modifying one component of an agent will have the consequence that the agent may not be on the manifold anymore. Think of a sphere manifold $\mathbb{S}^2$. An agent on this manifold may be the vector $x = [\sqrt{2}/2, 0, -\sqrt{2}/2]$ (since $\|x\| = 1$, we have that $x \in \mathbb{S}^2$). Suppose now that by the recombination step, the second component is modified and we obtain the vector $x' = [\sqrt{2}/2, 1, -\sqrt{2}/2]$. Remark that in this case $x' \notin \mathbb{S}^2$ since $\|x'\| \neq 1$.

However, we will see in chapter 3 that for specific optimization problems, we can actually include the recombination step in the DE algorithm. When the recombination is permitted for an optimization problem in chapter 3, we will explicitly announce it and shortly explain why this is possible.

2.3 Extension for the Stiefel manifold in Manopt

In section 2.2.1, we got an overview of all the available operators on Manopt. However, some of them are still not defined for some manifolds. More specifically, the log operator for the Stiefel manifold is still not implemented on Manopt. In this section, we present an iterative method called the shooting method that will do the same job as the log operator for a Stiefel manifold [26]. In chapter 3, we prove with an example that the shooting algorithm actually works and represent very well the function and meaning of the log operator. We begin this section by introducing the notion of geodesics that will be useful for the shooting algorithm. In the second part, we describe the algorithm and present its pseudocode.

2.3.1 Geodesics

In differential geometry, a geodesic is a generalization of the notion of a "straight line" to "curved spaces". The term "geodesic" comes from geodesy, the science of measuring the size and shape of Earth; in the original sense, a geodesic was the shortest route between two points on the Earth's surface, namely, a segment of a great circle. The term has been generalized to include measurements in much more general mathematical spaces; for example, in graph theory, one might consider a geodesic between two vertices/nodes of a graph.

Before getting to the mathematical expression of geodesics, let's define a concept that already has been used in the previous chapters. In 2.1.3 we defined formally the notion of curve on a manifold. Let's define the notions of *smooth curves* and *piecewise smoooth curves*.

Definition 12 (smooth curve [26])**.** *Given any Riemannian manifold, $\mathcal{M}$, a curve on $\mathcal{M}$ is a map, $\gamma : \mathbb{R} \to \mathcal{M}$. For a closed interval, $[a, b] \subseteq \mathbb{R}$, a map $\gamma : [a, b] \to \mathcal{M}$ is a smooth curve from $p = \gamma(a)$ to $q = \gamma(b)$ iff γ can be extended to a smooth curve $\tilde{\gamma} : (a - \epsilon, b + \epsilon) \to \mathcal{M}$, for some $\epsilon > 0$.*

Definition 13 (piecewise smooth curve [26]). *Given any two points, $p, q \in \mathcal{M}$, a continuous map, $\gamma : [a, b] \to \mathcal{M}$, is a piecewise smooth curve from p to q iff*

- *There is a sequence $a = t_0 < t_1 < ... < t_{k-1} < t_k = b$ of numbers, $t_i \in \mathbb{R}$, so that each map, $\gamma_i = \gamma[t_i, t_{i+1}]$, called a curve segment is a smooth curve, for $i = 0, ..., k-1$.*
- *$\gamma(a) = p$ and $\gamma(b) = q$.*

We can now introduce the definition of a *geodesic*:

Definition 14 (geodesic). *A curve $\gamma : \mathbb{R} \to \mathcal{M}$ is a geodesic if it has zero acceleration on all its domain.*

If the differential geometric properties of $\mathcal{M}$ are known, one can uniquely compute a geodesic $\alpha : [0, 1] \to \mathcal{M}$ when given a starting point $\alpha(0) \in \mathcal{M}$ and an initial velocity vector $\dot{\alpha}(0) \in T_{\alpha(0)}\mathcal{M}$, also termed a *shooting vector*. We denote $\alpha(1)$ as the point reached by following the direction $\dot{\alpha}(0)$ at $\alpha(0)$:

$$\alpha(1) = \mathrm{M.}\texttt{retr}(\alpha(0), \dot{\alpha}(0))$$

We will now derive the equation of the geodesic on a Stiefel manifold. To start the development, we consider the curve $\alpha : [0, 1] \to \mathcal{M}$. We begin with the condition that $\alpha(t)$ remains on the Stiefel manifold $\mathrm{St}(n, p)$ for every $t \in [0, 1]$, that is:

$$\alpha(t)\alpha(t)^\top = I_p$$

Taking two derivatives,

$$\alpha(t)^\top \ddot{\alpha}(t) + 2\dot{\alpha}(t)^\top \dot{\alpha}(t) + \ddot{\alpha}(t)^\top \alpha(t) = 0 \tag{2.4}$$

For an uniformly parameterized geodesic on a manifold embedded in an Euclidean space, the acceleration vector at each point along the geodesic is in the normal space [26]. This result implies that:

$$\ddot{\alpha}(t) + \alpha(t)S = 0 \tag{2.5}$$

for some symmetric matrix S. By substituting $\ddot{\alpha}(t) = -\alpha S$ into equation 2.4, we see that $S = \dot{\alpha}(t)^\top \dot{\alpha}(t)$. By replacing this expression of S in 2.5, we get the so called geodesic equation for a curve $\alpha(t)$:

$$\ddot{\alpha}(t) + \alpha(t)(\dot{\alpha}(t)^\top \dot{\alpha}(t)) = 0 \tag{2.6}$$

To resume, if a curve $\alpha(t)$ satisfies equation 2.6, then it is a geodesic curve on the Stiefel manifold. Edelman, Arias, and Smith [27] provide the following analytical solution to the geodesic equation. Given a point $\alpha(0) \in \mathrm{St}(n, p)$ and a shooting vector $\dot{\alpha}(0) \in T_{\alpha(0)}\mathrm{St}(n, p)$, then, for $t \in [0, 1]$

$$\alpha(t) = \begin{bmatrix} \alpha(0) & \dot{\alpha}(0) \end{bmatrix} \mathrm{expm}\left(t \begin{bmatrix} A & -S(0) \\ I & A \end{bmatrix} \right) I_{2p,p} \mathrm{expm}\left(-tA\right) \tag{2.7}$$

where $A = \alpha(0)^\top \dot{\alpha}(0)$, $S(0) = \dot{\alpha}(0)^\top \dot{\alpha}(0)$ and $I_{2p,p}$ is the $2p \times p$ matrix with the first p rows equal to the $p \times p$ identity matrix and the last p rows equal to the $p \times p$ zero matrix, and expm refers to the matrix exponential. We now have a precise description of a geodesic.

Note that we can use this equation to define the retraction operator. Indeed, suppose $\alpha(0) = U_1$, the shooting vector $\Delta \in T_{\alpha(0)}\mathrm{St}(n,p)$ and the point $\alpha(1) = U_2$. The retraction operator is defined as the point we reach on the manifold starting from a given initial point and a given direction (defined by the shooting vector). Therefore, we have :

$$U_2 = \mathrm{M.retr}(U_1, \Delta) = \begin{bmatrix} U_1 & \Delta \end{bmatrix} \mathrm{expm}\left(\begin{bmatrix} U_1^\top \Delta & -\Delta^\top \Delta \\ I & U_1^\top \Delta \end{bmatrix} \right) I_{2p,p} \mathrm{expm}\left(-U_1^\top \Delta \right)$$

2.3.2 General description and pseudocode of the shooting method

In [26] two numerical recipes for endpoint geodesics on a general Riemannian manifold $\mathcal{M}$ for use on the Stiefel manifold are proposed. We decided to implement one of these two methods, namely the *shooting method* because it is the more theoretically simplistic and widely known method. The main idea is to fix $\alpha(0) = U_1$ initialize a shooting vector $\Delta \in T_{\alpha(0)}\mathrm{St}(n,p)$, and iteratively update this shooting vector until the resulting shooting geodesic has its endpoint $\alpha(1)$ located within a certain error tolerance of U_2. The goal is thus to construct a geodesic between two points U_1 and U_2 on the manifold and obtain an expression of the final shooting vector illustrated on Figure 2.6.

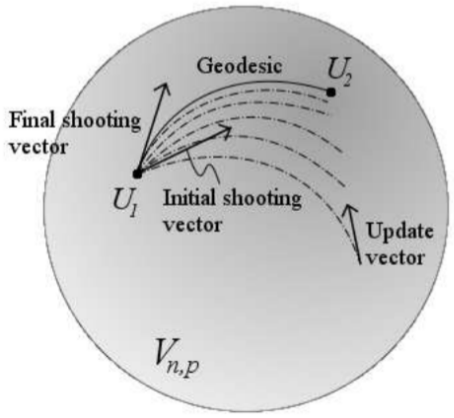


Figure 2.6: Illustration of the shooting method to construct a geodesic between U_1 and U_2: at first,we start at the lowest geodesic given by an initial shooting vector. As the algorithm goes further, the geodesic updates and we obtain a final shooting vector at U_1 that points toward U_2. $V_{n,p}$ refers to the Stiefel manifold embedded in the Euclidean space (Figure in [26]).

The pseudocode is given by Algorithm 6. The function $f(\alpha(0), \Delta, t_i)$ refers to equation 2.7 to construct a geodesic. The shooting vector is updated as long as the difference $\alpha(1) - U_2$ in the ambient space $\mathbb{R}^{n\times p}$ is greater than a convergence tolerance τ. This difference vector is then projected on the tangent space of $\alpha(1)$ and transported through the geodesic $\alpha(t)$. In order to maintain numerical stability, we carry out this parallel translation in T stages, where we sequentially translate w from $\alpha(t_i)$ to $\alpha(t_{i-1})$ for $i = T, T-1, ..., 1$. Now that w lies in the same vector space as Δ, we compute the update to the shooting vector as $\Delta \leftarrow \Delta - \delta w$ for some step size $\delta > 0$.

Algorithm 6 Shooting method for Stiefel manifold

Input: Problem, gradient step size δ, convergence tolerance τ, maximum numbers iterations J, T, uniform time partition $\{t_i = i/T; i = 1, 2, 3, ..., T\}$
Output: $[\Delta, \alpha(t)] \leftarrow Shoot(U_1, U_2)$
$\tilde{\Delta} \leftarrow U_2 - U_1$;
$\Delta \leftarrow$ Problem.M.proj$(U_1, \tilde{\Delta})$;
$\alpha(0) \leftarrow U_1$;
for $i = 1, ..., T$ **do**
  $\alpha(t_i) \leftarrow f(\alpha(0), \Delta, t_i)$;
end for
$j = 0$;
$\|w\| = \tau + 1$;
while $\|w\| > \tau$and$j < J$ **do**
  $\tilde{w} \leftarrow \alpha(1) - U_2$;
  $w \leftarrow$ Problem.M.proj$(\alpha(1), \tilde{w})$;
  for $i = T, T-1..., 1$ **do**
    $w \leftarrow$ Problem.M.transp$(\alpha(t_i), \alpha(t_{i-1}), w)$;
  end for
  $\Delta \leftarrow \Delta - \delta w$;
  for $i = 1, ..., T$ **do**
    $\alpha(t_i) \leftarrow f(\alpha(0), \Delta, t_i)$;
  end for
  $j = j + 1$;
end while

Generally, the parameters of the Shooting method are taken as follows: $\delta = 0.5, T = 10000, \tau = 10^{-6}$ and J is chosen depending on the problem. In chapter 3, an experiment is designed to ensure the good working of the shooting method and an analysis for the parameters of the methods is presented.

Chapter 3

Results of the novel methods on manifolds

This chapter presents the results and performances of the novel methods with graphs, illustrations and tables. The methods of chapter 1 are not only compared to each other, but also compared to methods that were already implemented on Manopt. We decided to compare our *derivative free* methods to the Particle Swarm Optimization (PSO) method. PSO is a population based method (just like DE) that has showed very interesting performances for global optimization problems and presents a good convergence when we are dealing with real-valued objective functions [28, 29]. PSO has been implemented on Manopt in 2012. For the methods that requires knowledge of the derivative of the objective function (like the accelerated version of the methods described in section 1.3), these ones were compared to the trustregion method that is present in Manopt since 2012. The trustregion method is the most trust-worthy in Manopt and is the first choice for smooth optimization. A precise description and implementation of the trustregion algorithm is detailed in [4, 5]. We refer to the sources cited if one wishes to have a deeper view on the PSO and trustregion method. The main tool used is thus Matlab and all kind of data (time computation, absolute error, relative error, objective value...) harvested during the algorithms are illustrated.

Each section of this chapter presents a specific problem and is composed of two parts; one part describes the problem, that is, a mathematical description of the objective function, a mathematical background to describe the utility of this objective function and some applications for this problem. The second part presents the results for the *derivative free* and gradient based methods. For this second part, we also briefly announce the chosen values for the parameters and the design choices. In total we have four problems. The fifth section shows the results obtained for the shooting method.

If one wishes to reproduce the results, we directly refer to the additional zip-file to this work (please, follow firstly the README).

Table C.1 in appendix C.1 represents the chosen *default* values for the parameters when we call the methods. If there is no specific precision mentioned concerning these parameters through this chapter, suppose then that these parameters values are used.

3.1 Sphere packing

As first problem, we consider the problem of placing p points $x_1, ..., x_p$ on a sphere $\mathbb{S}^{n-1} = \left\{x \in \mathbb{R}^n | x^\top x = 1\right\}$ such that the two closest points (w.r.t. the geodesic distance) are as far as possible. This problem, known as the Thomson or Tammes problem (or also as the sphere packing problem) is directly linked to that of placing as many points as possible on a sphere such that no two points are closer to each other than a given tolerance $\epsilon > 0$. Applications may be found in coding theory.

3.1.1 Mathematical description

Formally, the optimization problem is the following structured non-smooth problem:

$$\max_{x_1,...,x_p \in \mathbb{S}^{n-1}} \min_{1 \leq i < j \leq n} \text{dist}(x_i, x_j) \tag{3.1}$$

Suppose we define the distance between two points x_i and x_j on a sphere as the second norm of the difference of the two points:

$$\text{dist}(x_i, x_j) = \left\|x_i - x_j\right\|^2$$

We thus have that:

$$\begin{aligned} \left\|x_i - x_j\right\|^2 &= (x_i - x_j)^\top (x_i - x_j) \\ &= x_i^\top x_i - 2x_i^\top x_j + x_j^\top x_j \\ &= 1 - 2x_i^\top x_j + 1 \end{aligned}$$

We conclude thus that minimizing $\left\|x_i - x_j\right\|^2$ is equivalent to maximize the product $x_i^T x_j$.

Similarly to the previous reasoning, the angle α between two points x_i and x_j on the sphere is given by $\alpha = \arccos(x_i^\top x_j)$. The angle α can therefore give an indication of the geodesic distance between the two points, we note $\text{dist}(x_i, x_j) \propto \arccos(x_i^\top x_j)$. Since the function $\arccos(x)$ is a strictly decreasing function (see appendix C.2) of $x_i^T x_j$ on the domain $[-1, 1]$, we can conclude that maximizing $\text{dist}(x_i, x_j)$ is equivalent to maximize $x_i^\top x_j$.

With the two previous approaches, we can rewrite an equivalent formulation of the problem given by equation 3.1:

$$\min_{x_1,...,x_p \in \mathbb{S}^{n-1}} \max_{1 \leq i < j \leq p} x_i^T x_j \tag{3.2}$$

We now resort the classical log-sum-exp approximation of the max function [30], based on the following bounds. Let $y_1, ..., y_m \in \mathbb{R}$ and $y_{max} = \max_i y_i$. Then, since $y_i - y_{max} \leq 0$ for all i,

$$y_{max} \leq \epsilon \log\left(\sum_i^m \exp\left(\frac{y_i}{\epsilon}\right)\right) = \epsilon \log\left(\exp\left(\frac{y_{max}}{\epsilon}\right)\sum_i^m \exp\left(\frac{y_i}{\epsilon}\right)\right)$$
$$\leq \epsilon \log\left(m \exp\left(\frac{y_{max}}{\epsilon}\right)\right)$$
$$= y_{max} + \epsilon \log(m)$$

Thus, for a fixed value of $\epsilon > 0$, the following is a smooth approximation of the problem described in 3.2 which can be tackled using the optimization algorithms described in chapter 1:

$$\min_{x_1,...,x_p \in \mathbb{S}^{n-1}} f(x_1, ..., x_p) = \epsilon \log\left(\sum_{1 \leq i < j \leq p} \exp\left(\frac{x_i^T x_j}{\epsilon}\right)\right) \tag{3.3}$$

The gradient of this function is well-defined and we refer to the Matlab codes if one wishes to have a precise description of the gradient.

Let $X \in \mathbb{R}^{p \times n}$ such that $x_1, ..., x_p$ denote its (unit norm) rows. The matrix X assembles thus p points on the sphere. We write $f(x_1, ..., x_n) = f(X)$.

From the Phd thesis of Nicolas Boumal in 2016 [5], we deduce the following concerning the value for ϵ:
"After some simulations, we concluded that the value of ϵ should be as close to 0 as affordable. If it is too close to zero, optimization first becomes much slower, then simply doesn't work anymore becomes of floating point overflow errors (NaN's and Inf's start to appear). If it is too large, then log-sum-exp is a poor approximation of the max function, and the spread will be less uniform. An okay value seems to be 0.01 or 0.001 for example. Note that a better strategy than using a small epsilon straightaway is to reduce epsilon bit by bit and to warm-start subsequent optimization in that way. However, in our codes we set $\epsilon = 0.0015$."

We can treat this problem by working on the Oblique or Elliptope manifold. We refer to appendix C.3 for further details concerning the Oblique and Elliptope manifold.

We expect this problem to have many solutions since there exists a lot of different distributions of points on the sphere that will generate nearly the same objective function. Think for example of placing two points on a sphere such that they are as far as possible from each other; it is obvious that we have a large amount of possible dispositions of the two points on the sphere for this problem. Moreover, the Riemannian distance between these two points will be the same for each optimal disposition. We can link this reasoning with the fact that $f(X) = f(XQ)$ for any rotational orthogonal matrix Q.

3.1.2 Results

We begin this section with specifications for some of the novel methods:

- **SSA:** We are allowed to use the SSA algorithm. Indeed, since the input argument of the problem is a population of p-points on the sphere $\mathbb{S}^{n-1}$ (represented by a matrix X of size $p \times n$ as described in section 3.1.1), we could easily modify one random point among the p-points in the matrix X. Concretely, the so called random *component* in the description of SSA (see section 1.1.4) corresponds to an *agent* of the population X. That is, at each iterations of the SSA algorithm, we will only replace one agent of the population X by another agent generated randomly. On the other side, the SA algorithm modifies the whole population at each iteration.
- **DE:** The recombination step for the DE method is allowed. We apply the same reasoning as for SSA.

The minimal objective value of the minimization problem described by 3.3 is obtained with the TR method available in Manopt. Table 3.1 shows the objective values obtained for different parameters settings. The distributions obtained by TR are also showed in Figure 3.1.

(n, p)	$(3, 6)$	$(3, 16)$	$(3, 24)$	$(4, 16)$
f^*_{TR}	0.00373	0.6234	0.7292	0.4024

Table 3.1: Minimal objective values returned by the TR algorithm for the minimization problem described in 3.3. The pair (n, p) denotes respectively the dimension of the sphere $\mathbb{S}^{n-1}$ and the number of points on the sphere.

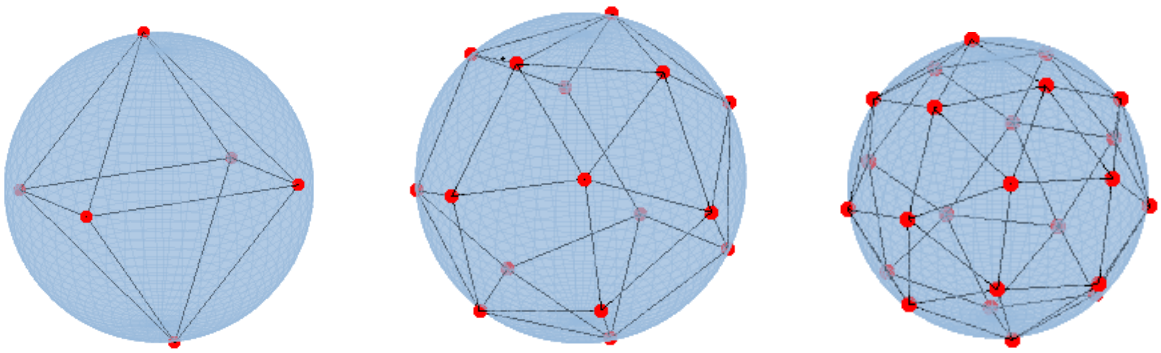


Figure 3.1: Distributions returned by the TR algorithm for $p = 6, 16$ and 24 points on the sphere.

In our results, we present the performances of our novel methods compared to the solution returned by TR (notably for the relative errors of the objective values).

Derivative free methods

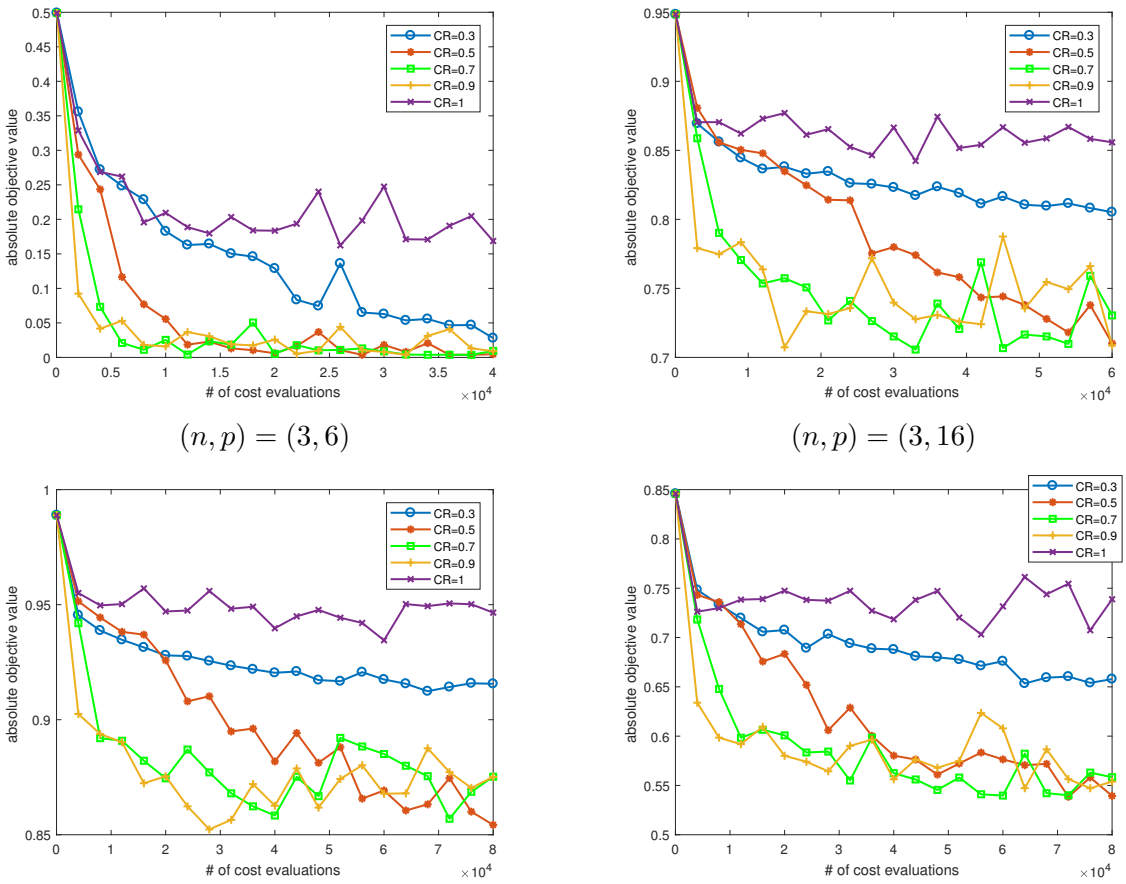
In Figure 3.2 and 3.3, we present the results concerning the parameters and design choices of DE. We show the evolution of the absolute objective value for different values of the crossover rate CR and for different schemes (see Table 1.2). We do this with different pairs (n, p) where n is the dimension of the sphere $\mathbb{S}^{n-1}$ and p is equal to the number of points on the manifold. Table 3.2 shows the computation times for the different schemes.

In Figure 3.4, we present the results for all the derivative-free novel methods and for different pairs (n, p) compared to the objective value returned by TR.

In appendix C.4, we compare the novel methods with PSO by means of differences of relative errors:

$$\nu_k = \frac{\xi_k - \xi^{PSO}}{\xi^{PSO}}$$

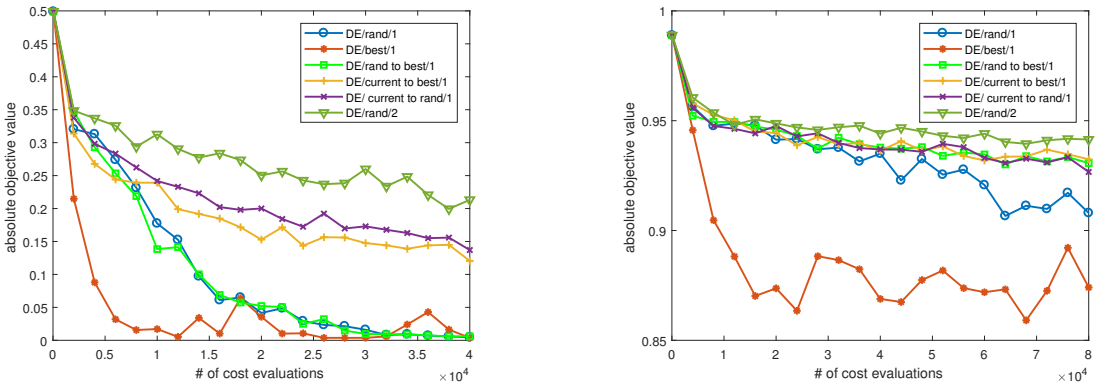
where ξ^{PSO} is the final relative error of PSO after a certain number of iterations and ξ_k is the relative error of a certain method at iterate k.



$(n, p) = (3, 6)$ $(n, p) = (3, 16)$

$(n, p) = (3, 24)$ $(n, p) = (4, 16)$

Figure 3.2: Parameter analysis of the crossover rate (CR) in DE with DE/best/1 as mutation scheme. The plots represents the evolution of the absolute objective value for different values of CR. The data points are the means of a sample of size 10. Globally, we observe that the best value for CR is 0.7 or 0.9.



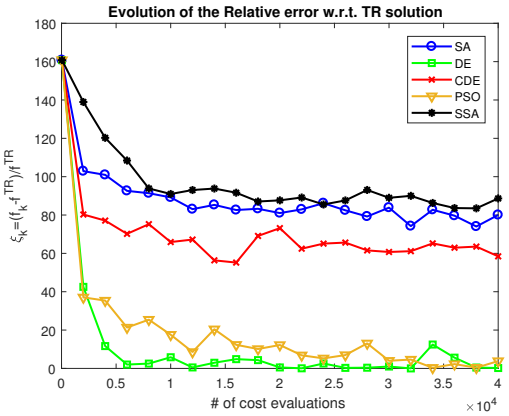
$(n, p) = (3, 6)$ $(n, p) = (3, 24)$

Figure 3.3: Parameter analysis of the different mutation schemes in DE with $CR = 0.7$. The plots represents the evolution of the absolute objective value for the different schemes. The data points are the means of a sample of size 10. Globally, we observe that the best mutation scheme is DE/best/1.

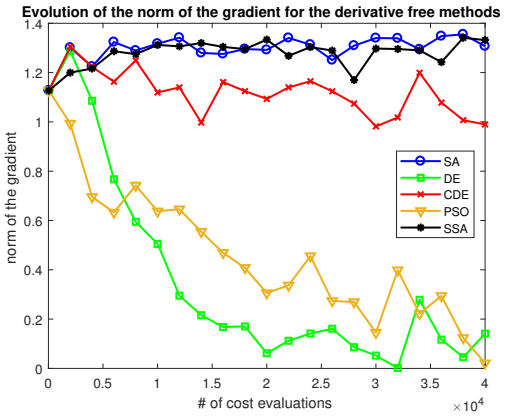
(n, p, max_costs)	$(3, 6, 40000)$	$(3, 16, 60000)$	$(3, 24, 80000)$	$(4, 16, 80000)$
DE/rand/1	14.7808 ($\pm$ 0.1251)	23.6674 ($\pm$ 0.0338)	34.7938 ($\pm$ 0.0485)	**31.5825**($\pm$ 0.0685)
DE/best/1	**14.7444** ($\pm$ 0.0853)	**23.6578** ($\pm$ 0.0468)	**34.8581** ($\pm$ 0.0329)	**31.5825** ($\pm$ 0.0588)
DE/rand-best/1	18.0307 ($\pm$ 0.0902)	30.1451 ($\pm$ 0.0220)	41.9795 ($\pm$ 0.4895)	40.2854 ($\pm$ 0.0386)
DE/curr.-best/1	18.0986 ($\pm$ 0.1046)	30.1753 ($\pm$ 0.0354)	41.8429 ($\pm$ 0.0357)	40.3235 ($\pm$ 0.0558)
DE/curr.-rand/1	18.1396 ($\pm$ 0.0596)	30.3423 ($\pm$ 0.0436)	42.0290 ($\pm$ 0.0448)	42.7178 ($\pm$ 3.3818)
DE/rand/2	18.0899 ($\pm$ 0.0873)	30.2984 ($\pm$ 0.0491)	41.8986 ($\pm$ 0.7049)	41.9146 ($\pm$ 0.1414)

Table 3.2: Average computation time (in seconds) for the different schemes of DE from samples of size 10. The pair (n, p, max_costs) corresponds respectively to the dimension of the sphere, the number of points on the sphere and the maximum number of cost evaluations allowed. The values in brackets are the standard deviations from the samples. The bolt numbers are the ones with the smallest computation time.

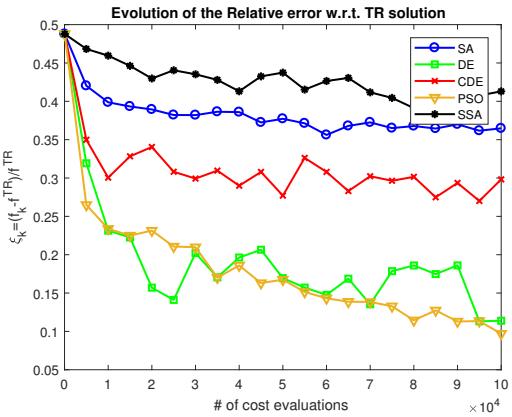
From the previous figures, we conclude that DE/best/1 is the best mutation scheme for this problem and we set $CR = 0.7$.



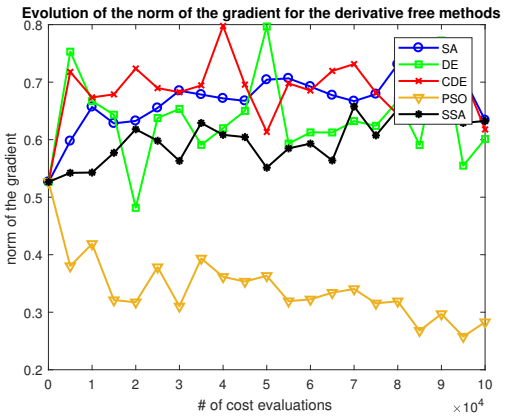
Relative error for $(n, p) = (3, 6)$



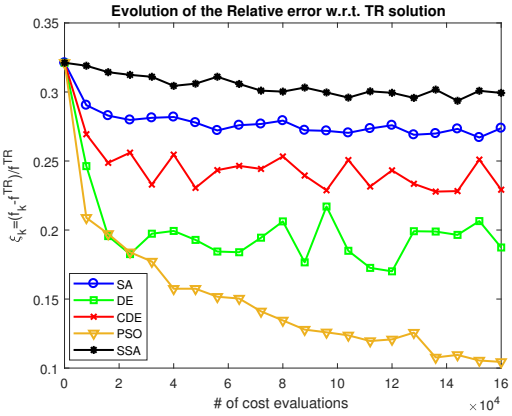
Norm of the gradient for $(n, p) = (3, 6)$



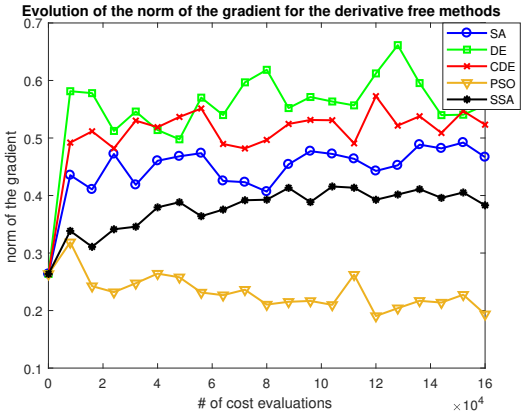
Relative error for $(n, p) = (3, 16)$



Norm of the gradient for $(n, p) = (3, 16)$



Relative error for $(n, p) = (3, 24)$



Norm of the gradient for $(n, p) = (3, 24)$

Figure 3.4: Evolution of the relative error $\xi_k = \frac{f_k - f^*}{f^*}$ and the norm of the gradient for the derivative-free methods. The data points are the means of a sample of size 10. When comparing the relative errors, we observe globally that PSO is the best method. Furthermore, the DE method is very close to the performance of PSO and is even better for a small amount of points on the sphere. The norm of the gradient is far from a zero value, this means that the stochastic methods didn't manage to reach exactly a local minimum.

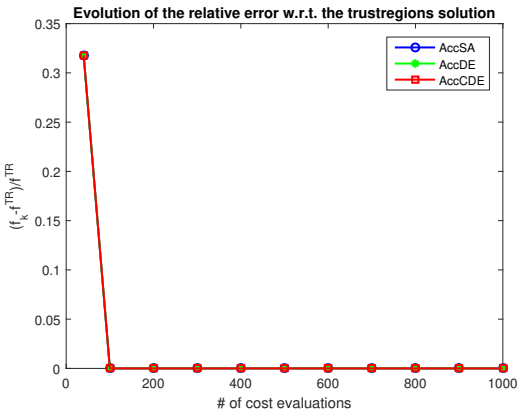
(n, p, max_costs)	$(3, 6, 40000)$	$(3, 16, 100000)$	$(3, 24, 160000)$	$(4, 16, 160000)$
SA	7.9017 ($\pm$ 0.0556)	22.8228 ($\pm$ 0.0230)	39.6396 ($\pm$ 0.0683)	35.5314 ($\pm$ 0.0715)
SSA	7.0613 ($\pm$ 0.0401)	21.1440 ($\pm$ 0.0273)	36.4205 ($\pm$ 0.0335)	33.3125 ($\pm$ 0.0208)
DE	14.746 ($\pm$ 0.0781)	42.9392 ($\pm$ 0.1333)	73.3879 ($\pm$ 0.3840)	67.6860 ($\pm$ 0.0583)
CDE	14.4486 ($\pm$ 0.0624)	42.4583 ($\pm$ 0.1029)	70.8583 ($\pm$ 0.1028)	66.8270 ($\pm$ 0.0686)
PSO	8.8534 ($\pm$ 0.0697)	24.5569 ($\pm$ 0.0347)	43.6502 ($\pm$ 0.0271)	39.0968 ($\pm$ 0.0546

Table 3.3: Average computation time (in seconds) for the methods from samples of size 10. The pair (n, p, max_costs) corresponds respectively to the dimension of the sphere, the number of points on the sphere and the maximum number of cost evaluations allowed. The values in brackets are the standard deviations from the samples. The red numbers are the greatest computation times.

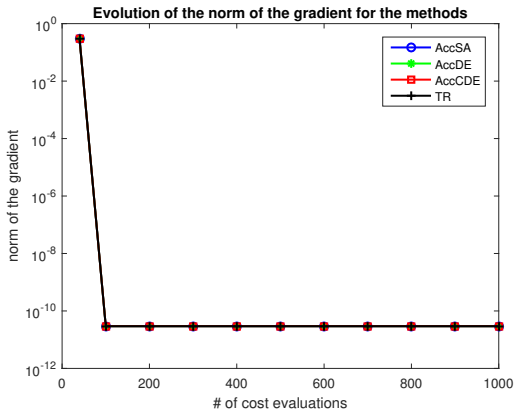
From the previous results, we conclude that the DE method is the closest to the performance of PSO for this problem. Among all our derivative-free novel methods, DE seems to be the most precise one but also the one that requires the most computational effort as shown in Table 3.3.

Accelerated methods

The accelerated novel methods use the TR algorithms to find a local solution. Since the TR algorithm converges in a few steps to the minimal objective value of this problem, the accelerated methods show all the same behaviour as illustrated in Figure 3.5 for $(n, p) = (3, 24)$. These methods are not worth for further analysis for this problem since they depend uniquely of the performance of the TR method.



Relative error for $(n, p) = (3, 24)$



Norm of the gradient for $(n, p) = (3, 24)$

Figure 3.5: Evolution of the relative error w.r.t. the trustregions solution: $\xi_k = \frac{f_k - f^{TR}}{f^{TR}}$ and norm of the gradient for all the accelerated methods from a sample of size 5. We observe that the accelerated methods cannot find a better solution than the one generated by TR. Indeed, the first step of these accelerated methods is to call a gradient method (local search) and the second step calls a stochastic method to discover a better local minimum (global search). Here, it seems that the TR method reaches immediately the global minimum, so the stochastic methods cannot find a better solution. These results are similar for other parameter settings for (n, p).

3.2 Levy function No 5

In this section, we present the performances of the novel methods on a 2-dimensional objective function that has the particularity to have some local minima on a well-known domain. This problem will allow us to have a good visualization of the performances of the methods since we can represent graphically the results obtained and we have knowledge of the search space and the exact location of the global minimum.

3.2.1 Mathematical description

We consider a lightly modified version of the Levy function No 5. The objective function $E : \mathbb{R}^2 \to \mathbb{R} : (x_1, x_2) \in [-1, 1]^2 \to E(x_1, x_2)$ is then given by:

$$E(x_1, x_2) = \sum_{i=1}^{4} icos[(i+1)x_1 + i] \sum_{j=1}^{4} icos[(j+1)x_2 + j] + 0.5\big((10x_1 + 1.45)^2 + (0.0002x_2 + 0.8)^2\big) + 100$$

The gradient of the objective function can be computed and we refer to appendix C.5 if one wishes to have a precise description of the gradient.

We modified the Levy function No 5 in order to obtain four observable local minima as illustrated in Figure 3.6.

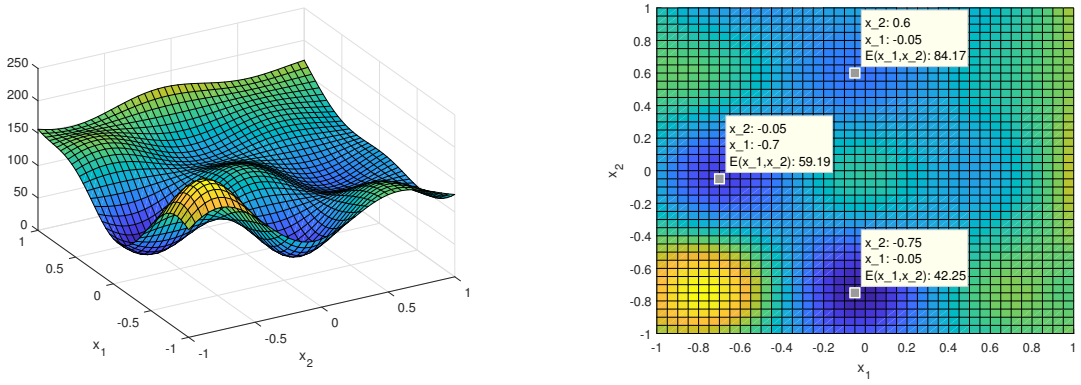


Figure 3.6: Modified Levy function with the three main local minima shown. The global minimum is at $(x_1, x_2) = (-0.05, -0.75)$ and its value is 42.25.

Our goal is thus to reach the global minimum of this modified Levy function. In addition, we will impose that the search space is constraint such that $\|x\| = 1$ where $x = (x_1, x_2) \in [-1, 1]^2$. Therefore, the set of feasible solutions is given in Figure 3.7 by the circle in red. Note that this constraint is equivalent to work on a sphere manifold $\mathbb{S}^1$. To resume, we look for the global minimum on the red circle according to the objective function $E(x_1, x_2)$:

$$\min_{x \in \mathbb{S}^1} E(x_1, x_2) \tag{3.4}$$

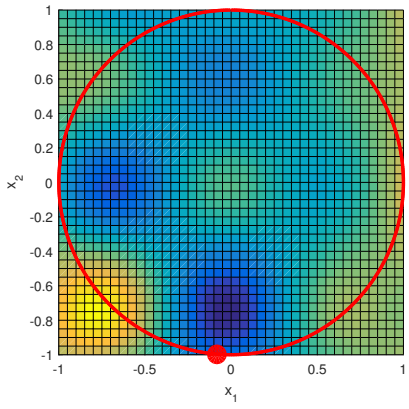


Figure 3.7: The set of feasible solutions is illustrated by the circle in red. The global minimum (the red point) is at $[x_1, x_2] = [-0.08175, -0.9967]$ and its value is 67.568.

Note that this problem is thus applied for a fixed dimension $n = 2$. In the results, the main concerns are about the starting point for the algorithms as will be illustrated.

3.2.2 Results

We begin this section with some specifications for some methods:

- **SSA.** We cannot use this method. Indeed, since modifying a component of a point on the circle $x \in \mathbb{S}^1$ may lead that this point is not on the circle anymore ($x_{new} \notin \mathbb{S}^1$) since the norm may not be equal to 1 anymore.
- **DE.** We cannot use the recombination step of the DE method anymore for the same reasons as for SSA.

Secondly, we introduce a parameter δ that will influence the initial distribution of the points on the circle given a starting point x_{start}. The We refer to appendix C.6 to visualize the effect of δ.

Note that we could also increase the number of agents to cover a bigger part of the search space. However, we will limit ourselves to ten agents for this problem to test the performances of the methods.

Although fixing a high value for δ will be a good idea to converge to the global minimum, we will do the opposite to test our methods. We will force the population based methods to start from a population of agents that are near to each other as illustrated for $\delta = 0.5$. This way, we will observe if the methods are able to converge to the global minimum starting from a "badly" distributed population. The SA and TR algorithm will start from the best agent of this badly distributed population.

What matters here, is that we have a good visualization and a precise description of the search space and we know where the global minimum is located so we can have meaningful analysis of the performances of the methods. This is particularly interesting because the conclusions we take at the end of this section, could be applied to problems where we don't have the knowledge of the global minimum or a visualization of the search space.

The results shown are averages taken from samples of size 100. As previously debated, we set $\delta = 0.5$.

Derivative free methods

We perform the same analysis as in 3.1 for the parameter CR and the best mutation scheme what concerns the DE method. It seems that the best value for CR is 0.7 and the best mutation scheme is DE/rand/2 (one can check the plots obtained in appendix C.7). However, since CDE is very similar to DE and presents a strong stochastic character, we will force the mutation scheme to be DE/best/1 for DE. This way, we have on one side

the DE method without any random interference and on the other side, we have the CDE method that randomly generates candidate solutions. We will find out if the stochastic character of the method is better than the deterministic character for this problem.

Secondly, we present the results when we start from $x_{start} = (0, 1)$ in Figure 3.8. The data points are the means of a sample of size 100. We say that a method *reaches* the global solution if the angle between the returned solution by the method and the global solution is smaller than $0.05°$. Figure 3.9 presents the histograms that indicates where the solutions returned by the methods are located on the circle from the global solution.

The Figures in appendix C.8 presents the same results but starting from $x_{start} = (-1, 0)$.

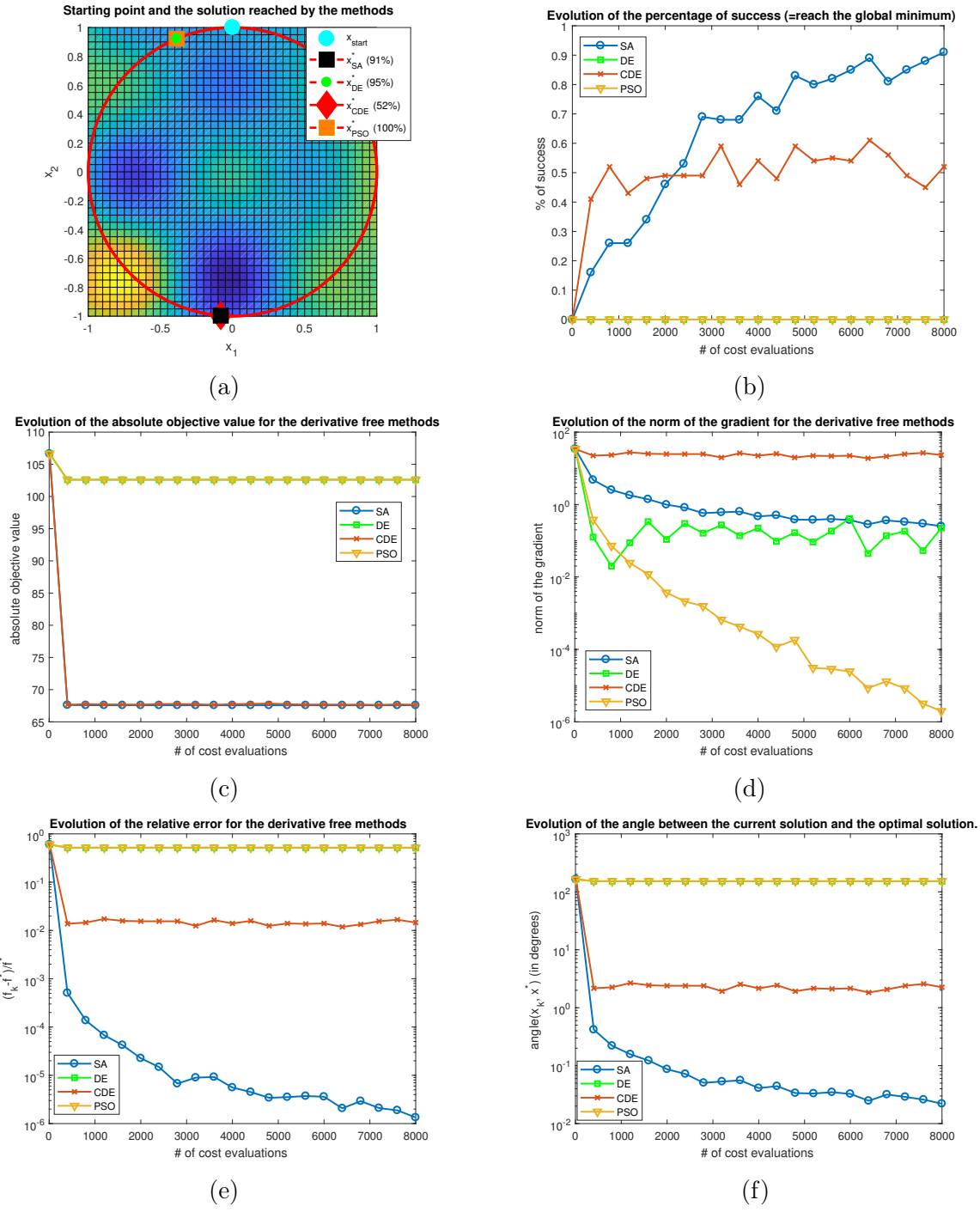


Figure 3.8: Obtained results for the derivative-free methods when we start from the point $x_{start} = (0, 1)$. Figure (a) illustrates where the solutions returned by the methods are situated on the circle after 8000 cost evaluations. The additional percentage in the legend indicates the probability we have to land on the point shown in Figure (a) after 8000 iterations. It seems that CDE and SA reach the global minimum while PSO and DE are "trapped" in a local minmimum. This is confirmed by Figure (b)-(c)-(d)-(e). Figure (f) describes where the iterates are located (on average) from the global solution.

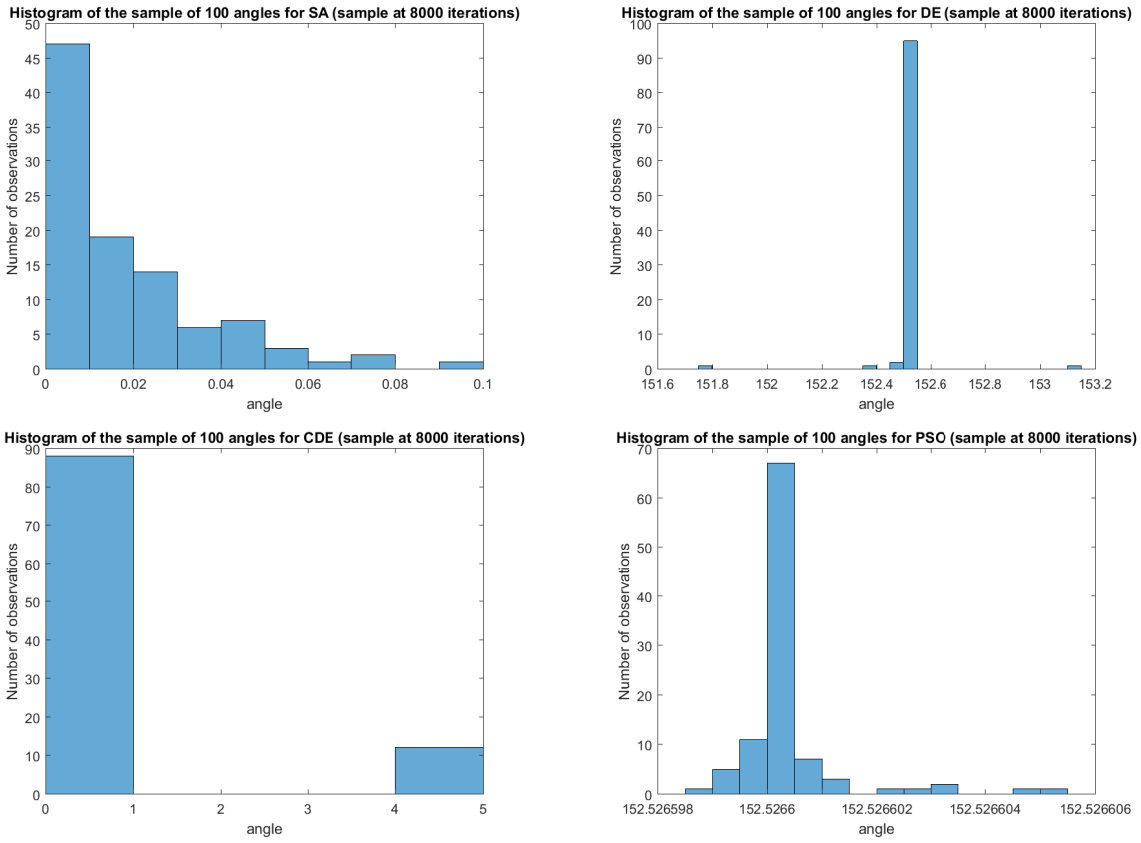


Figure 3.9: Histograms of the sample of 100 angles between the solution reached by the methods after 8000 cost evaluations and the global solution when we start from $x_{start} = (0, 1)$. We observe that the returned solution by SA is on average at an angle of almost 0° from the global solution. The returned solution of the DE and PSO method is at an angle of nearly 152° degrees of the global solution. The returned solution of the CDE can be situated at an angle of 4° or almost 0° degrees from the global solution.

From the previous results, we conclude that the DE and PSO can easily get 'stucked' in a local minimum. On the other hand, CDE and SA perform very well since they were able to reach the global minimum. The stochastic character of CDE and SA took over the deterministic character of PSO and DE (with DE/best/1).

Accelerated methods

We show the same results for the accelerated methods in Figure 3.10 where we start from $x_{start} = (-1, 0)$. In appendix C.9 we present the same results but starting from $x_{start} = (0, 1)$. Table 3.4 shows the average computation times after 8000 iterations.

x_{start}	$(0, 1)$	$(-1, 0)$
AccSA	8.3459 ($\pm$ 0.1380)	8.1582 ($\pm$ 0.1988)
AccDE	8.48 ($\pm$ 0.012)	8.8432 ($\pm$ 0.3466)
AccCDE	2.2625 ($\pm$ 0.02265)	2.5012 ($\pm$ 0.2458)
TR	0.0044 ($\pm$ 0.4330E-3)	0.004 ($\pm$ 0.3929E-3)

Table 3.4: Average computation time (in seconds) for the different method for 8000 iterations from samples of size 100. The pair (x_1, x_2) corresponds to the starting point for the methods. The values in brackets are the standard deviations from the samples.

For the case $x_{start} = (-1, 0)$ and the case $x_{start} = (0, 1)$, it is clear that the accelerated methods reach the global minimum while the TR got "trapped" in the first local minimum met. Furthermore, it seems that AccSA shows the best performances among our novel algorithms.

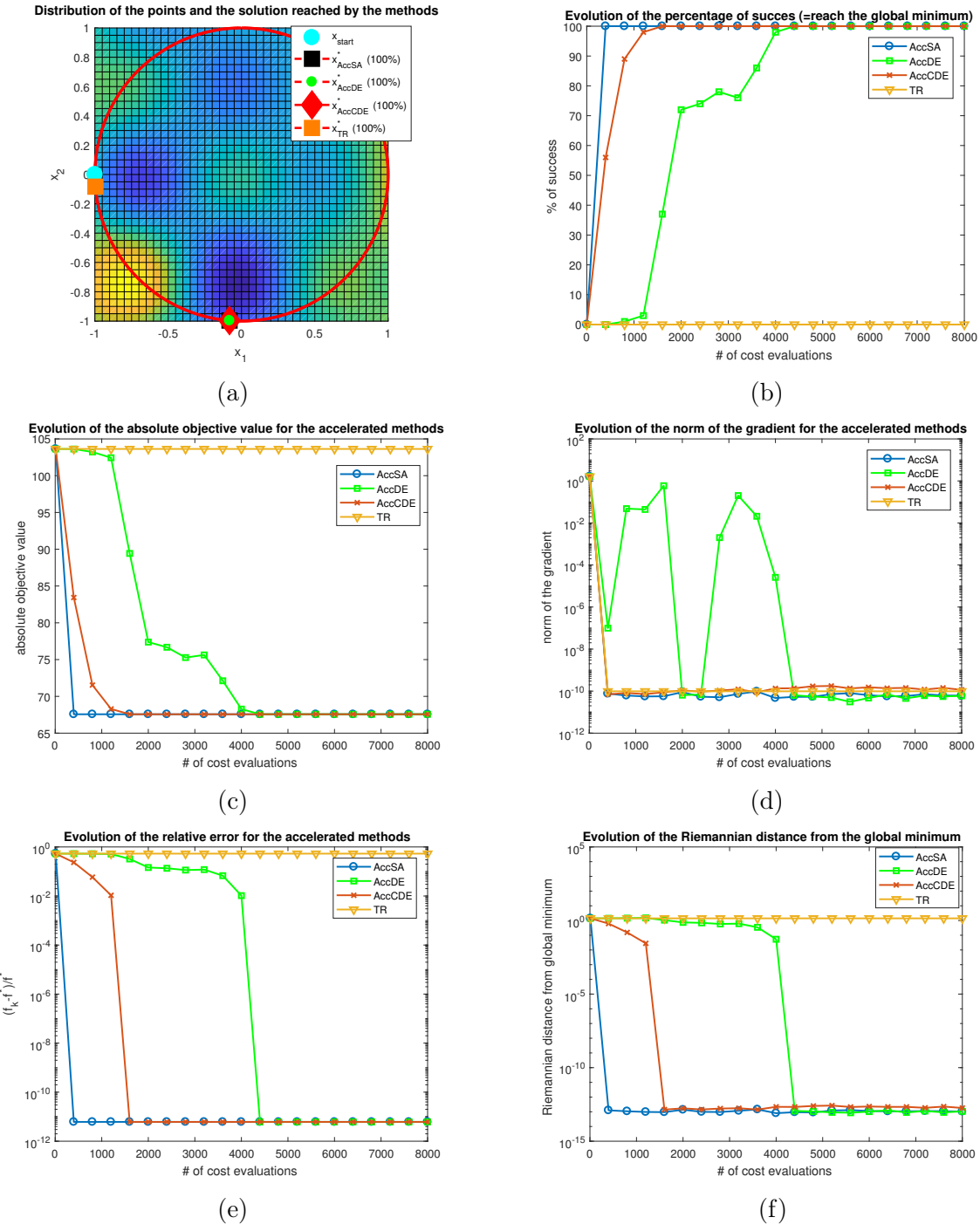


Figure 3.10: Obtained results for the gradient based methods when we start from the point $x_{start} = (-1, 0)$. Figure (a) illustrates where the solutions returned by the methods are situated on the circle after 8000 cost evaluations. The additional percentage in the legend indicates the probability we have to land on the point shown in Figure (a) after 8000 iterations. It seems that AccCDE, AccSA and AccDE reach the global minimum while TR is "trapped" in a local minimum. This is confirmed by Figure (b)-(c)-(d)-(e). Figure (f) describes where the iterates are located (on average) from the global solution.

3.3 Sphere fitting

We analyze the performances of the novel methods for the sphere fitting problem. The goal of this problem is to fit smooth paths to manifold-valued data points using piecewise functions. We approximate a function f on an interval $[a, b]$ by dividing this interval into a number of subintervals and to look for a piecewise approximation by polynomials of low degree. A large overview of piecewise-polynomial approximations on Euclidean spaces is covered in [31] (chapter 11).

To obtain an approximation of the smooth path, we minimize an objective function composed of a data-attachment term and a regularization term. In [32], an analysis is proposed for the quality of the path that is returned by a well-known fast algorithm [33]. The quality of this path is quantified by comparing it with the smooth path returned by the PSO method. It seems from the paper [32] that the fast method approaches the solution of the problem *"with a very satisfactory small relative error of 1% (compared to the PSO solution) of the cost value"*. In addition to PSO, we can compare the exactitude of the fast method with our novel methods.

This problem is motivated by several applications in engineering and the sciences, such as projected-based model order reduction of dynamical systems that depend on one parameter [34]. Some other applications like estimations of wind fields are also developed in [32].

3.3.1 Mathematical description

We use the same notations as in [32] and present the problem in the same way.

Consider n data points $\{d_0, ..., d_n\} \subset \mathcal{M}$ that take values on a manifold $\mathcal{M}$ and are associated with measurement parameters $t_0 \leq t_1 \leq ... \leq t_n$.

Bézier fitting
Consider firstly the trivial case where $\mathcal{M} = \mathbb{R}^d$ to define the Bézier curve. A *Bézier curve of degree K ($\in \mathbb{N}$)* is a function β parametrized by K *control points* $\{b_0, ..., b_K\} \in \mathbb{R}^d$ taking the form:

$$\beta_K(\cdot; b_0, ..., b_K) : [0, 1] \to \mathbb{R}^d, t \mapsto \sum_{j=0}^{K} b_j B_{jK}(t)$$

where $B_{jK}(t) = \binom{K}{j} t^j (1-t)^{K-j}$ are the Bernstein basis polynomials [35]. The first and the last control points are interpolated by construction while the position of the other control points model the shape of the curve. More specifically, the quadratic and cubic

Bézier curves are respectively

$$\beta_2(t; b_0, b_1, b_2) = b_0(1-t)^2 + 2b_1(1-t)t + b_2t^2$$
$$\beta_3(t; b_0, b_1, b_2, b_3) = b_0(1-t)^3 + 3b_1(1-t)^2t + 3b_2(1-t)t^2 + b_3t^3$$

One well-known way to generalize Bézier curves to a Riemannian manifold $\mathcal{M}$ is via the De Casteljau algorithm. This algorithm is generalized by Popiel and Noakes [36].

We can now consider a general manifold $\mathcal{M}$. As illustrated in Figure 3.11, the composite Bézier function $\mathcal{B} \in \mathcal{M}$ is a composition of n Bezier curves, *i.e.*,

$$\mathcal{B} : [0, n] \to \mathcal{M}, t \mapsto \beta^i(t-i)\text{on}[i, i=1]$$

where $i = 0, ..., n-1$ and where β^i defines a piece of $\mathcal{B}$ associated to the endpoints $\{p_i, p_{i+1}\} \subset \mathcal{M}$. The control points of the i^{th} and $(i+1)^{th}$ piece of $\mathcal{B}$ defined on the left and right of p_i are noted $\left\{b_i^-, b_i^+\right\} \subset \mathcal{M}$, $i = 1, ..., n-1$. The continuity of $\mathcal{B}$ is trivial as $\mathcal{B}(i) = \beta(i) = \beta^{i-1}(i) = p_i$.
"Differentiability is ensured by taking $p_1 = av\big[(b_1^-, b_1^+), (\frac{2}{5}, \frac{3}{5})\big], p_i = av\big[(b_i^-, b_i^+), (\frac{1}{2}, \frac{1}{2})\big]$ *(for* $i = 2, ..., n-2$*) and* $p_{n-1} = av\big[(b_{n-1}^-, b_{n-1}^+), (\frac{3}{5}, \frac{2}{5})\big]$ *[32]."*
A proof of these properties can be found in [37].

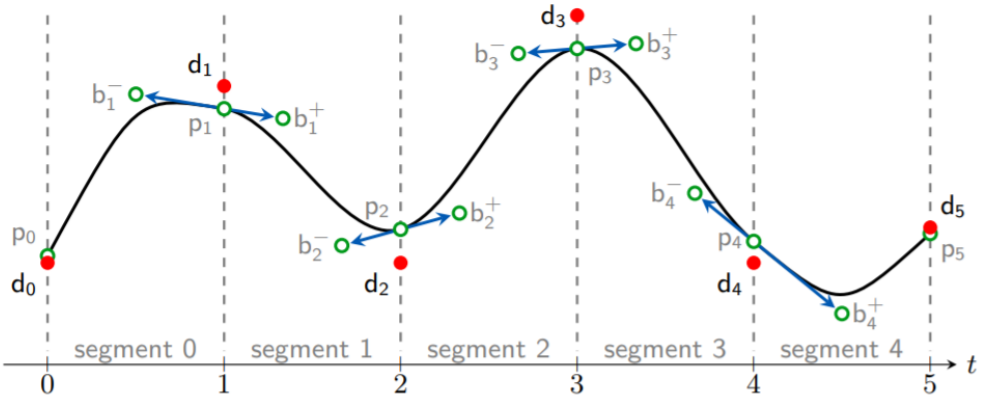


Figure 3.11: *"Schematic representation of the composite Bezier function* $\mathcal{B}(t)$*: the data points* d_i *are represented in red; the circled green ones are the control points. The first and last Bézier segments are quadratic Bézier functions while all the other segments are cubic Bézier functions [32]."*

We want thus to compute the composite Bézier curve that minimizes the mean square acceleration and its fidelity to data points. The problem is described by the minimization of the objective function f:

$$\min_{p_i, b_i^+, b_i^-} f(p_i, b_i^+, b_i^-) = \underset{p_i, b_i^+, b_i^-}{\text{minimize}} \underbrace{\sum_{i=0}^{n-1} \int_0^1 \left\| \ddot{\beta}^i(t) \right\|_{\mathcal{M}}^2 \mathrm{d}t}_{regularization} + \lambda \underbrace{\sum_{i=0}^{n} \text{dist}^2(p_i, d_i)}_{data-attachment}$$

where p_i, b_i^+ and b_i^- are thus points on the manifold $\mathcal{M}$ and the parameter $\lambda > 0$ adjusts the balance between data fidelity and the "smoothness" of $\mathcal{B}$. Note that this balance tends to the interpolation problem when $\lambda \to \infty$. Not also that what we enter as input argument for the objective function is a set of points on the manifold (similarly to the objective function for the sphere packing problem).

The data-attachment term is well defined on Riemannian manifolds (see the expression of $\text{dist}(\cdot,\cdot)$ in section 2.1.4). The regularization term hasn't been defined yet. Indeed, we are dealing with a second derivative on a manifold (mean square acceleration on manifolds). In [38], an approximation of this derivative form on manifolds is built based on a generalization of second order finite differences to manifolds with a certain stepsize $\Delta\tau$. The effect of the stepsize $\Delta\tau$ is analyzed in [32] and one concludes that a stepsize of $\Delta\tau = 10^{-2}$ leads to a relative error less than 1% for the problem. Without including any additional details, we will suppose that the second derivative $\ddot{\beta}^i(t)$ is well defined. The integration is replaced by a classical trapezoidal rule. For more information concerning the generalization of second order finite differences to manifolds and the trapezium rule, one can check Appendix C.10 which resumes some concepts from [38].

Unfortunately, an expression of the gradient for this objective function doesn't exists yet.

As said before, there exists an approximated solution of this minimization problem that is returned by a fast well known algorithm [33]. The advantage is that this approximated solution can be rapidly computed. It is computed in two steps.

Step1. Note that the objective function is a quadratic function in the $2n$ variables $p_0, (b_i^-, b_i^+)_{i=1}^{n-1}$ and p_n. In [33], it is shown how the solution reads $\mathrm{x} = Q(\lambda)\mathrm{d}$ or

$$x_j = \sum_{l=0}^{n} q_{jl}(\lambda) d_l \tag{3.5}$$

where $x = [x_0, x_1, ..., x_{2n-1}]^\top := \left[p_0, b_1^-, b_1^+, ..., b_{n-1}^+, p_n\right]^\top \in \mathbb{R}^{2n\times r}$ contains the $2n$ optimization variables, $d := [d_0, ..., d_n]^\top \in \mathbb{R}^{n+1\times r}$ contains the data points and finally $Q(\lambda) \in \mathbb{R}^{2n\times n+1}$ is a matrix of coefficient depending on λ.

Step2. We finally generalize equation 3.5 on a general manifold as in [32]:

$$x_j = \exp_{d_j^*}\left(\sum_{l=0}^{n} q_{jl}(\lambda)\mathrm{M.log}(d_j^*, d_l)\right) \tag{3.6}$$

where d_j^* is a reference point and we impose that $d_j^* = d_i$ when x_j is one of the control points b_i^-, b_i^+ or p_i. Finally, the smooth curve is reconstructed using the De Casteljau

algorithm [36].

The approximation of the smooth path is thus obtained by the points x_j in equation 3.6 and is very fast to compute. We can now analyze the performances of our novel methods compared with the fast algorithm.

3.3.2 Results

- We are allowed to use SSA for the same reasons as mentioned for the sphere packing problem. We ara also allowed to use the recombination step of the DE algorithm for the same reasons.
- Since the gradient for this objective function isn't available, we cannot use the accelerated methods.

Figure 3.12 show the results for six *geodesic* data points and $\lambda = 80$. Figure 3.13 shows the results obtained with six *random* data points and $\lambda = 80$. Figure 3.14 shows the same results but for $\lambda = 20$. We can see the difference between the two different values used for λ on the representation of the spheres. Indeed, for a smaller λ, the smooth curves are less "attached" to the data points.
Finally, we refer to appendix C.11 to visualize the results for eight random data points and $\lambda = 80$.

We observe that for geodesic data points, the three methods DE, CDE and PSO converge to the approximated solution while the achievements of SA and SSA are mediocre. For random data points, it seems that the PSO and the DE method are able to find a better solution than the approximated one. This may be due to the fact that for randomly placed points on the sphere the curves undergo a greater curvature, while for geodesic points the paths are less curved. We can thus conclude that the approximated solution presents a bad behaviour for more curved paths. On the other side, DE and PSO seems to be comfortable with curved paths; we can clearly see the difference with the approximated path on the illustrations of the spheres.

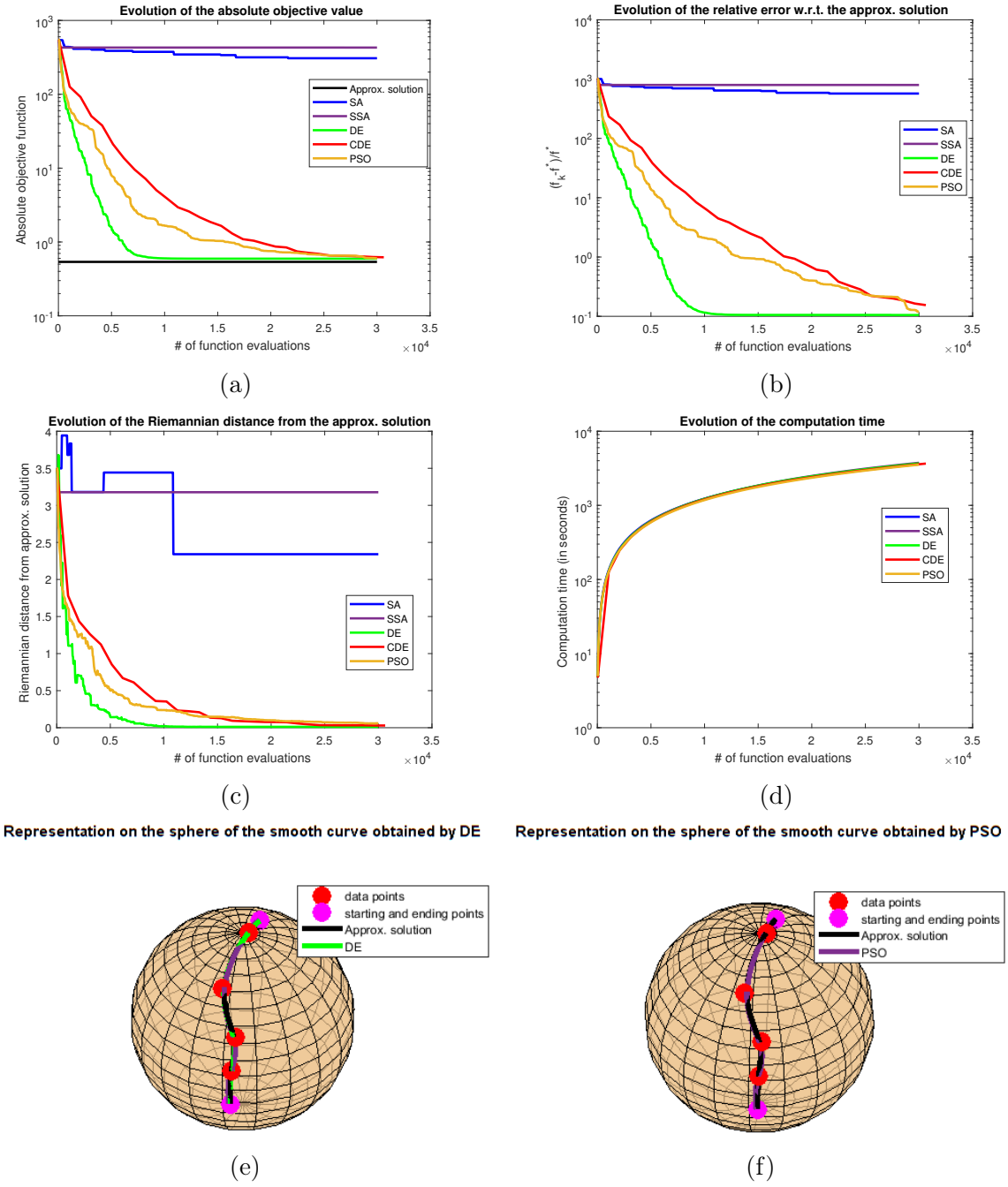


Figure 3.12: Results for the sphere fitting problem of six geodesic data points on the sphere and $\lambda = 80$. From Figure (a) and (b), we observe that DE, CDE and PSO are able close to reach the objective value f^* of the approx. solution. The relative error indicates that DE is is the closest to the objective value f^*. Figure (c) shows that the Riemannian distance between the solutions returned by DE, CDE and PSO is almost zeros from the approx. solution. Figure (e) and (f) illustrate the different paths for PSO and DE compared to the approx. solution. The difference of the paths are almost not visible since the paths are very close to each other.

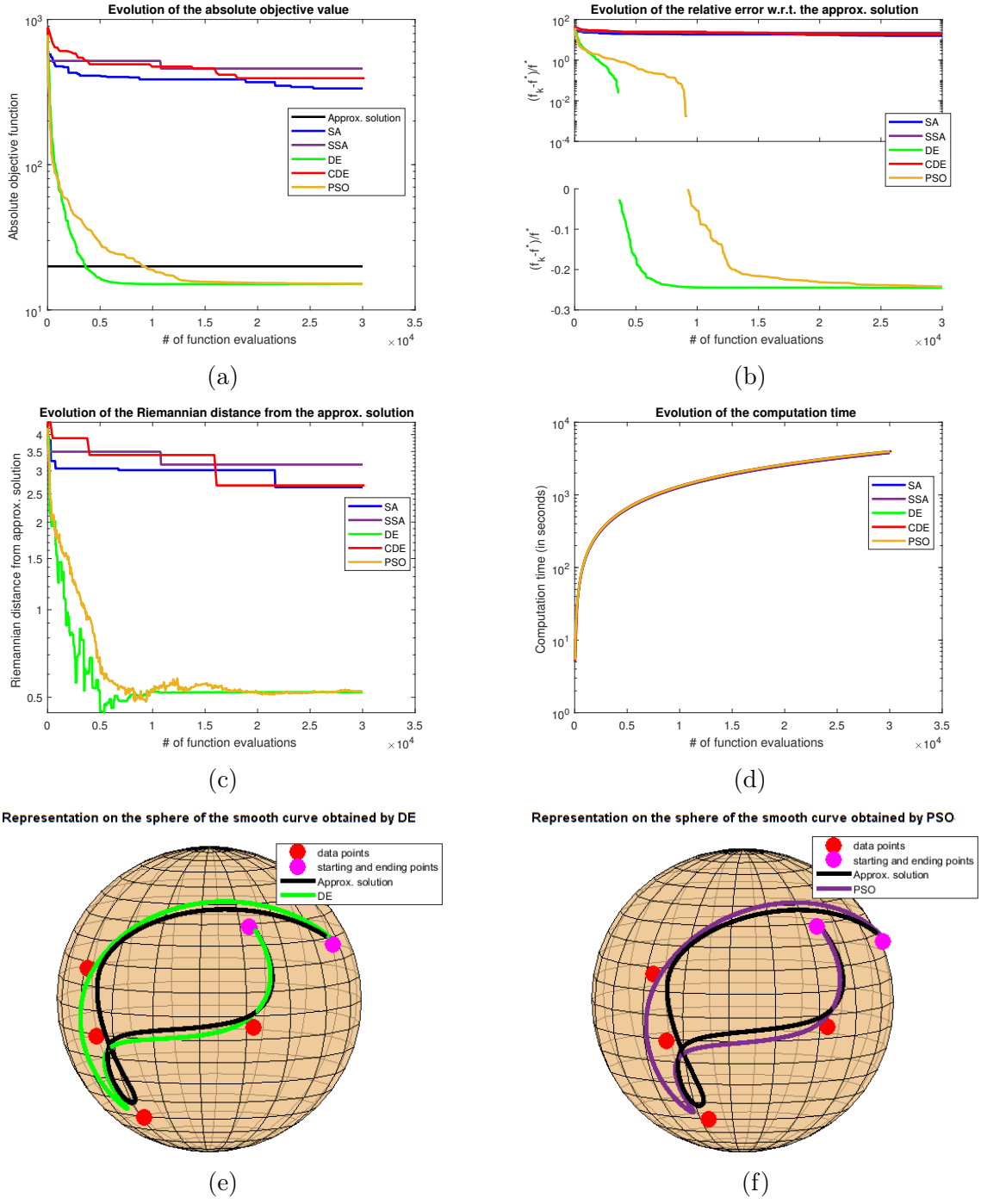


Figure 3.13: Results for the sphere fitting problem of six random data points on the sphere and $\lambda = 80$. From Figure (a) and (b), we observe that DE and PSO are able to reach a lower objective value than the objective value f^* of the approx. solution. The relative error indicates that the objective value of PSO and DE is almost 20% smaller than f^*. Figure (c) shows that the Riemannian distance between the solutions returned by PSO and DE are at a value of 0.5 from the approx. solution. Figure (e) and (f) illustrate the different paths for PSO and DE compared to the approx. solution.

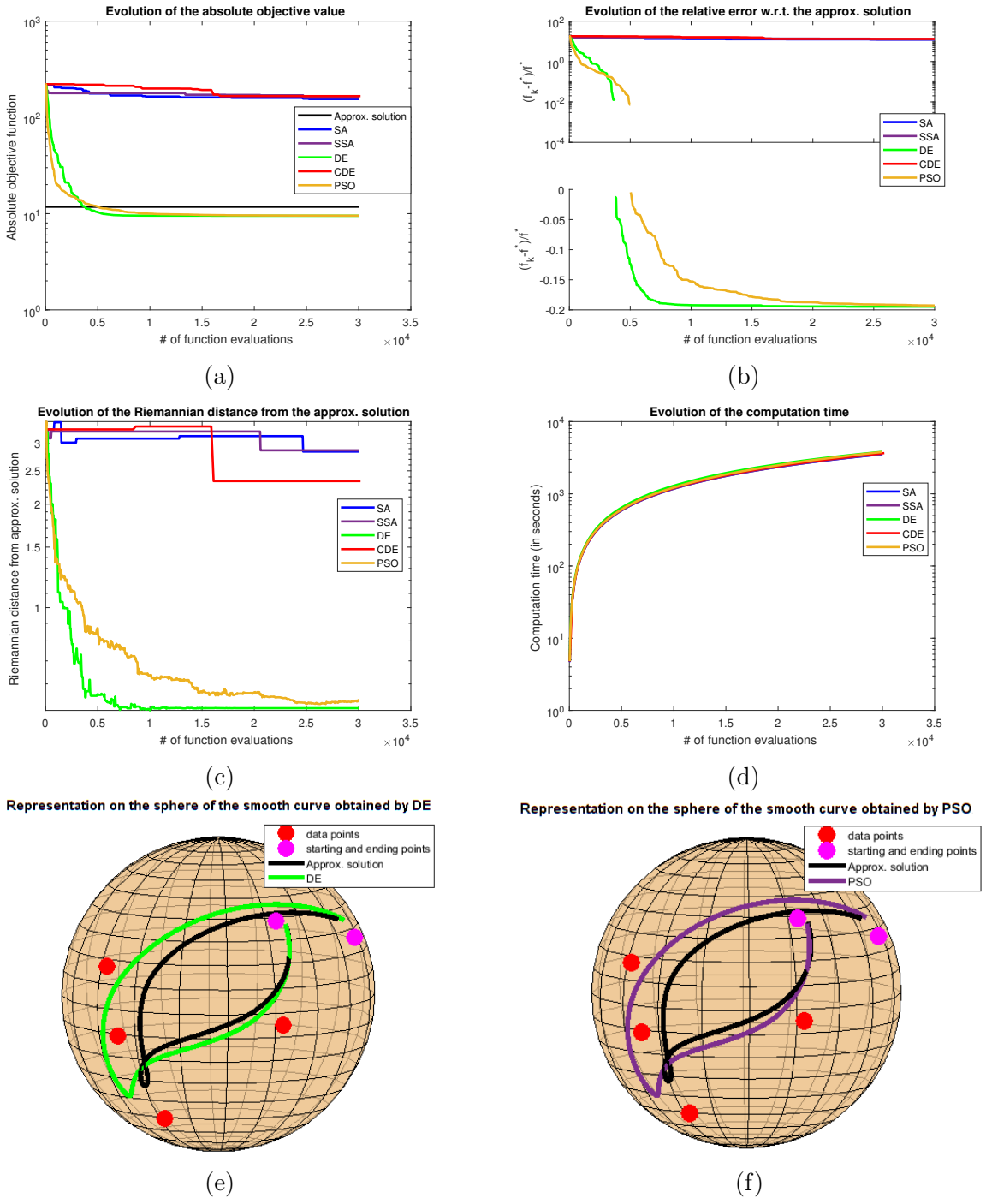


Figure 3.14: Results for the sphere fitting problem of six random data points on the sphere and $\lambda = 20$. From Figure (a) and (b), we observe that DE and PSO are able to reach a lower objective value than the objective value f^* of the approx. solution. The relative error indicates that the objective value of PSO and DE is almost 20% smaller than f^*. Figure (c) shows that the Riemannian distance between the solutions returned by PSO and DE are near the approx. solution. Figure (e) and (f) illustrate the different paths for PSO and DE compared to the approx. solution.

$(n, \lambda, max\_costs, R/G)$	$(6, 80, 30000, R)$	$(6, 20, 30000, R)$	$(8, 80, 35000, R)$	$(6, 80, 30000, G)$
SA	3530	3542	6490	3671
SSA	3550	3542	6491	3750
DE	3800	3819	6978	3680
CDE	3620	3640	6857	3590
PSO	3678	3785	6728	3531

Table 3.5: Average computation time (in seconds) for the methods from samples of size 4. The pair $(n, \lambda, max\_costs, R/G)$ corresponds respectively to the number of points on the sphere, the parameter λ for data-attachment, the maximum number of cost evaluations allowed and the nature of the points (R=random; G=Geodesic).

3.4 Best low multilinear rank approximation of higher-order tensors

This section deals with a particular Tucker compression problem. Formally, given a higher-order tensor, we are interested in its best low multilinear rank approximation. We firstly introduce the matrix case; in this case, the low rank approximation problem has only one local, hence global, minimum. It is known that the optimal solution for a matrix follows from the singular value decomposition (SVD) and is proven by the Eckart-Young-Mirsky theorem in [39].

For higher-order cases, the problem is by far more complex. Indeed, in the tensor problem, there exists local optima. The values of the cost function at different local minima can be very close to each other and on the other hand, the subspaces on which the original tensor can be computed can be very different [40]. If the subspaces are of importance, different local minima may yield very different results. In [40], it is shown with numerical examples that there exists a relation between the number of local minima that are found and the distribution of the higher-order singular values of the tensor. One concludes that if there is a gap between specific singular values, the problem seems to be easier and a fewer minima are found. The problem seems more difficult if the tensor has no distinct low multilinear rank structure.

In this section, we will observe if our novel methods are able to switch from an initial local minimum to another one. This task seems hard since it is known that the local minima may have objective values that are very close to each other as said previously.

3.4.1 Mathematical description

Matrix case

We want to solve the low rank approximation problem for a given matrix A and the desired rank k:

$$\begin{aligned} \underset{X \in \mathbb{R}^{m \times n}}{\text{minimize}} \quad & \|A - X\|_F \\ \text{subject to} \quad & rank(X) \leq k \end{aligned} \tag{3.7}$$

where $\|\cdot\|_F$ is the Frobenius norm.

The well-known optimal solution of this problem follows from the SVD. Let

$$A = U \Sigma V^\top$$

be the SVD of the matrix A where $U \in \mathbb{R}^{m \times m}$, $\Sigma \in \mathbb{R}^{m \times n}$ and $V \in \mathbb{R}^{n \times n}$. Note that the matrices U and V are unitary matrices and that the matrix Σ contains the singular values of matrix A on its diagonal.

We define the following notations:

$$U := \begin{bmatrix} U_1 & U_2 \end{bmatrix}, \qquad \Sigma := \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \qquad V := \begin{bmatrix} V_1 & V_2 \end{bmatrix}$$

where $\Sigma_1 \in \mathbb{R}^{k \times k}$, $U_1 \in \mathbb{R}^{m \times k}$ and $V_1 \in \mathbb{R}^{k \times n}$. Then, the rank-$k$ matrix obtained from the truncated SVD is given by:

$$\hat{A} = U_1 \Sigma_1 V_1^\top$$

As said previously, $X = \hat{A}$ is known to be the optimal solution to the problem 3.7. This has been proven by the Eckart-Young-Mirsky theorem in [39].

Higher-order case

Approximating a tensor by another one with low multilinear rank is often called Tucker compression. For simplicity, we consider third-order real-valued tensors. The cost function that has to be minimized is defined in the following way: given a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_2}$, its best rank-$(R_1, R_2, R_3)$ approximation $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ minimizes the least-squares function $f : \mathbb{R}^{I_1 \times I_2 \times I_3} \to \mathbb{R}$:

$$\begin{aligned} \underset{\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}}{\text{minimize}} \quad & \|\mathcal{A} - \mathcal{X}\|_F \\ \text{subject to} \quad & rank(\mathcal{X}) \leq (R_1, R_2, R_3) \end{aligned} \tag{3.8}$$

In practice, it is equivalent and more convenient to *maximize* the function (see [41]):

$$\begin{aligned} g :& St(I_1, R_1) \times St(I_2, R_2) \times St(I_3, R_3) \to \mathbb{R} \\ & (U, V, W) \mapsto \left\| \mathcal{A} \bullet_1 U^\top \bullet_2 V^\top \bullet_3 W^\top \right\| = \left\| U^\top A_{(1)} (V \otimes W) \right\| \end{aligned}$$

where the matrices U, V and W are orthonormal matrices, $St(I_n, R_n)$ stands for the set of column-wise orthonormal $(I_n \times R_n)$-matrices (Stiefel manifold). Note that $St(I_1, R_1) \times St(I_2, R_2) \times St(I_3, R_3)$ is a product manifold of three Stiefel manifolds. The concept of a product manifold is defined in section 2.1.1. The terms $\bullet_n$ stands for the mode-n product of a tensor with a matrix and is defined as follows:

Definition 15 (mode-n products [41])**.** *We denote the mode-n products of a tensor* $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ *with matrices* $M^{(n)} \in \mathbb{R}^{j_n \times I_n}$*,* $n = 1, 2, 3$ *by* $\mathcal{A} \bullet_n M^{(n)}$*, where*

$$\begin{aligned} \left(\mathcal{A} \bullet_1 M^{(1)} \right)_{j_1 i_2 i_3} &= \sum_{i_1} a_{i_1 i_2 i_3} m^{(1)}_{j_1 i_1} \\ \left(\mathcal{A} \bullet_2 M^{(2)} \right)_{i_1 j_2 i_3} &= \sum_{i_2} a_{i_1 i_2 i_3} m^{(2)}_{j_2 i_2} \\ \left(\mathcal{A} \bullet_3 M^{(3)} \right)_{i_1 i_2 j_3} &= \sum_{i_3} a_{i_1 i_2 j_3} m^{(3)}_{j_3 i_3} \end{aligned}$$

The term $A_{(1)}$ stands for the matrix representation of the tensor $\mathcal{A}$ and is defined as follows:

Definition 16 (matrix representations [41])**.** *The matrix representations* $A_{(n)}, n = 1, 2, 3$ *of a tensor* $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ *is given by:*

$$\left(A_{(1)}\right)_{i_1, (i_2 - 1) I_3 + i_3} = \left(A_{(2)}\right)_{i_2, (i_3 - 1) I_1 + i_1} = \left(A_{(2)}\right)_{i_3, (i_1 - 1) I_2 + i_2} = a_{i_1 i_2 i_3}$$

The Euclidean gradient of the function $g(U, V, W)$ is well defined and is given by:

$$\begin{aligned} \text{grad} g(U, V, W) &= (\text{grad}_U g, \text{grad}_V g, \text{grad}_W g) \\ &= (2A_{(1)}(V \otimes W)(V \otimes W)^\top (A_{(1)})^\top U, \\ & 2A_{(2)}(W \otimes U)(W \otimes U)^\top (A_{(2)})^\top U, \\ & 2A_{(3)}(U \otimes V)(U \otimes V)^\top (A_{(3)})^\top U) \end{aligned}$$

The equivalent formulation of the minimization problem described in 3.8 is thus given by the following maximization problem:

$$\begin{aligned} \underset{U,V,W}{\text{maximize}} \quad & \left\| U^T A_{(1)} (V \otimes W) \right\| \\ \text{subject to} \quad & U \in St(I_1, R_1) \\ \text{subject to} \quad & V \in St(I_2, R_2) \\ \text{subject to} \quad & W \in St(I_3, R_3) \end{aligned} \tag{3.9}$$

It has been showed that we can transform the solution of the maximization problem described in 3.9 onto a solution of the minimization described in 3.8 with the following equation [40]:

$$\mathcal{X} = \mathcal{A} \bullet_1 UU^T \bullet_2 VV^T \bullet_3 WW^T \tag{3.10}$$

The procedure is thus to firstly determine the best matrices U, V and W for the maximization problem in 3.9 and afterwards obtain an expression for $\mathcal{X}$ with equation 3.10.

We introduce the following *invariance property* for the objective function of problem 3.9:

$$g(U, V, W) = g(UQ^{(1)}, VQ^{(2)}, WQ^{(3)})$$

where $Q^{(i)} \in O_{R_i}, i = 1, 2, 3$ are orthogonal matrices. It is thus advisable to work on the manifold $\mathcal{M} = St(R_1, I_1)/O_{R_1} \times St(R_2, I_2)/O_{R_2} \times St(R_3, I_3)/O_{R_3}$ which can be seen as a product of three Grassmann manifolds since the Grassmann manifold is a quotient manifold of the Stiefel manifold $\mathrm{St}(n, p)$ by the action of $\mathrm{O}(p)$ as showed in section 2.1.2. By the invariance property, it is thus equivalent to work on the Grassmann manifold for the problem 3.9.

We decided to work with the Grassmann manifold. The column spaces of U, V and W are computed using iterative algorithms like our novel methods. The entries of the tensor $\mathcal{A}$ we wish to approximate are generated randomly from a uniform distribution in the interval $(0, 1)$.

3.4.2 Results

We apply our novel methods on the maximization problem described in 3.9. We begin with some comment for the methods:

- **SSA.** We cannot use this method. Modifying a component (or a line) of the matrices U, V or W may lead that this matrix is not orthogonal anymore.
- **DE.** We cannot us the recombination step in DE for the same reasons as for SSA.

Derivative-free methods

Firstly, we apply the TR algorithm on the maximization problem 3.9 with a tensor $\mathcal{A} \in \mathbb{R}^{10\times10\times10}$ whose entries are random entries in the interval $(0, 1)$ and the imposed multilinear rank is $(2, 2, 2)$. We start TR from five different starting matrices for U, V and W. We denote these five different starting matrices as $\left\{(U_i^{TrustR}, V_i^{TrustR}, W_i^{TrustR})\right\}_{i=1}^{5}$ and they are generated randomly. Tables 3.6, 3.7 and 3.8 show the objective values (in brackets) reached by TR from the five starting matrices and also the subspace angles between the different solutions reached by TR. Let's denote U_i^* the solution reached by TR starting from U_i^{TrustR}. Then, the entry (i, j) of Table 3.6 indicates the subspace angle

between the matrix U_i^* and the matrix U_j^* $(i, j = 1, 2, ...5)$. The objective values in red are the minimal objective values reached and in bold the maximal objective value reached. Since we are dealing with a maximization problem (problem 3.9), we want the objective value to be as great as possible.

	U_1^{TrustR} **(6353.6)**	U_2^{TrustR} (6350.7)	U_3^{TrustR} (6353.6)	U_4^{TrustR} (6352.1)	U_5^{TrustR} (6350.7)
U_1^{TrustR}	0	1.39	0	1.559	1.39
U_2^{TrustR}		0	1.39	1.129	0
U_3^{TrustR}			0	1.559	1.39
U_4^{TrustR}				0	1.129
U_5^{TrustR}					0

Table 3.6: Subspace angles between the different matrices U. The objective values reached by TR are in brackets. The best starting matrix is U_1^{TrustR} since the objective value reached by TR starting from this matrix is the greatest.

	V_1^{TrustR} **(6353.6)**	V_2^{TrustR} (6350.7)	V_3^{TrustR} (6353.6)	V_4^{TrustR} (6352.1)	V_5^{TrustR} (6350.7)
V_1^{TrustR}	0	1.564	0	1.415	1.564
V_2^{TrustR}		0	1.564	1.335	0
V_3^{TrustR}			0	1.415	1.564
V_4^{TrustR}				0	1.335
V_5^{TrustR}					0

Table 3.7: Subspace angles between the different matrices V.The objective values reached by TR are in brackets. The best starting matrix is V_1^{TrustR} since the objective value reached by TR starting from this matrix is the greatest.

	W_1^{TrustR} **(6353.6)**	W_2^{TrustR} (6350.7)	W_3^{TrustR} (6353.6)	W_4^{TrustR} (6352.1)	W_5^{TrustR} (6350.7)
W_1^{TrustR}	0	1.111	0	1.405	1.111
W_2^{TrustR}		0	1.111	1.164	0
W_3^{TrustR}			0	1.405	1.111
W_4^{TrustR}				0	1.164
W_5^{TrustR}					0

Table 3.8: Subspace angles between the different matrices W. The objective values reached by TR are in brackets. The best starting matrix is W_1^{TrustR} since the objective value reached by TR starting from this matrix is the greatest.

From Tables 3.6, 3.7 and 3.8, we conclude that the "worst" objective value reached for problem 3.9 is 6350.7 and that there exists (at least) three local minima for the problem.

We will now apply our methods starting from the "worst" matrices $(U_5^{TrustR}, V_5^{TrustR}, W_5^{TrustR})$ and observe if they are able to reach the objective value 6350.7 and why not reach an even better objective value (like 6353.6 or 6352.1). The population based methods start from a lightly perturbed population around $(U_5^{TrustR}, V_5^{TrustR}, W_5^{TrustR})$. We show the evolution of the objective values of the methods on Figure 3.15. We also compare the subspace angle of the matrices returned by the methods and the matrices returned by TR: (U_5^*, V_5^*, W_5^*).

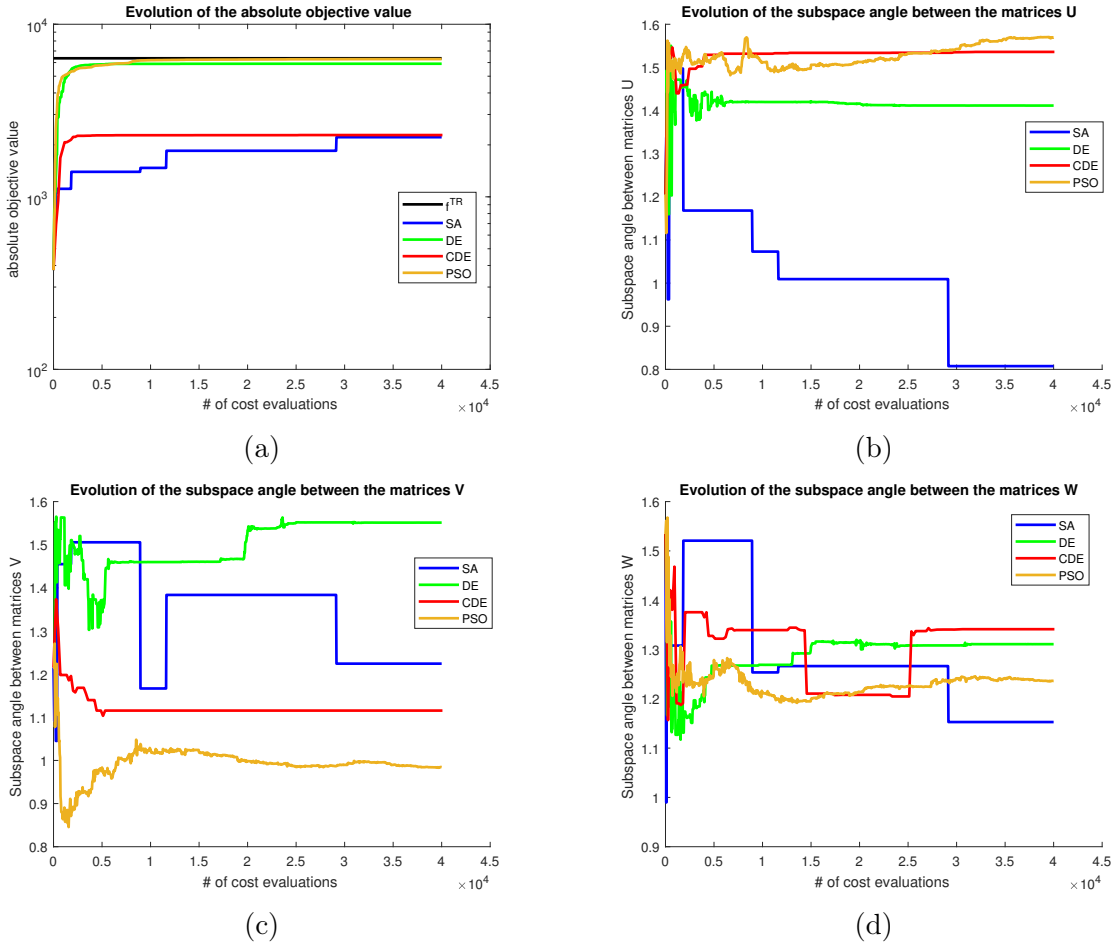


Figure 3.15: Low multilinear rank approximation of a tensor $\mathcal{A} \in \mathbb{R}^{10\times10\times10}$ whose entries are random entries in the interval $[0, 1]$ and the imposed multilinear rank is $(2, 2, 2)$. The methods starts from $(U_5^{TrustR}, V_5^{TrustR}, W_5^{TrustR})$ (see previous tables) and it seems on Figure (a) that only DE and PSO are able to reach the objective value of 6350.7. Even if DE and PSO are able to reach an objective value of 6350.7, it seems that the matrices are not close at all to the solution returned by TR as illustrated on Figure (b)-(c)-(d). Furthermore, the methods are not able to reach a better objective value than 6350.7.

Accelerated methods

The accelerated methods use the TR algorithm for local search. The performances of AccSA, AccDE and AccCDE depends thus mainly on the performance of the TR method. As seen previously for the derivative-free methods, the solution reached by TR is very dependent on the starting point. Again, we apply TR to the problem 3.9 and we start from different initial matrices. We obtain Table 3.9 for the matrix U. We refer to appendix C.12 for the tables concerning the matrices V and W. The worst objective value reached is in red.

	U_1^{TrustR} (5384.6)	U_2^{TrustR} (5381.3)	U_3^{TrustR} (5384.6)	U_4^{TrustR} (5384.6)	U_5^{TrustR} (5376.7)
U_1^{TrustR}	0	1.2929	0	0	1.2048
U_2^{TrustR}		0	1.2929	1.2929	1.2846
U_3^{TrustR}			0	0	1.2048
U_4^{TrustR}				0	1.2048
U_5^{TrustR}					0

Table 3.9: Subspace angles between the different matrices U. The objective values reached by TR are in brackets.

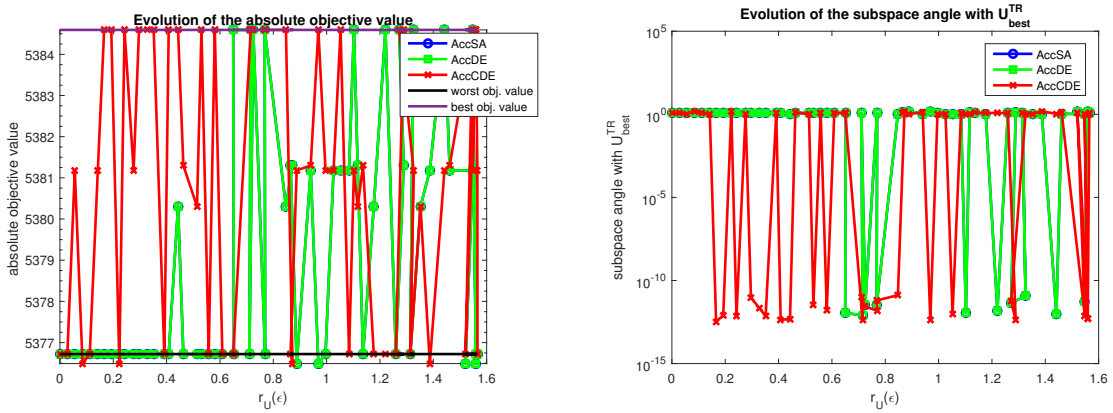
We will now let the accelerated methods start from a population around the central point $(U_5^{TrustR}, V_5^{TrustR}, W_5^{TrustR})$. The population is constructed by perturbing the central point in a random direction and with a parameter $\varepsilon > 0$. The greater the value of ε, the larger the population covers the search space. We define the maximal subspace angle between the solution returned by TR and the initial population as follows:

$$r_U(\epsilon) = \max_j \angle \left(U^{TR}, U_j(\varepsilon) \right)$$

where and $U_j(\varepsilon)$ denotes the population around a central point perturbed by a parameter ε ($j = 1, ..., N_p$). The symbol $\angle$ denotes the subspace angle.

Figure 3.16 shows the results obtained for the accelerated methods. We called the accelerated methods for different populations. The greater the value of $r_U(\varepsilon)$, the further the population is located from the worst solution (which has objective value 5376.7). Ideally, we would like the methods to converge to a higher objective value when $r_U(\epsilon)$ is equal to 1.2048 (as in Table 3.9). Unfortunately, this is not the case as illustrated on Figure 3.16. It seems that a small perturbation suffices to TR to converge to a higher objective value. However, the results seems to be very irregular when $r_U(\epsilon)$ increases.

We refer to appendix C.13 for the results obtained for V and W.



(a) (b)

Figure 3.16: Results for the accelerated methods applied on the Low rank approximation problem. It seems that we need only a small perturbation ε to reach the best known objective value. We cannot analyze the performances of the accelerated methods correctly since the TR method does all the job.

3.5 Shooting method

We will now present the results obtained for the Shooting method described in section 2.3. Recall that the Shooting method does the same job as the log operator for a Stiefel manifold. The method returns the geodesic $\alpha : [0, 1] \to \mathrm{St}(n, p)$ linking two points U_1 and U_2 and returns also the tangent vector Δ (or the *shooting* vector) in the tangent plane of U_1 and that is pointing toward U_2 (see Algorithm 6 in section 2.3.2). However, we don't have to forget that the Shooting method is an approximation of the log operator. We designed four experiments to have a better estimation of the error of the Shooting method. The experiments are similar to the ones realized in [26]. By default, we set the parameters of Algorithm 4 as follows: $\tau = 1E{-}4, T = 10000, J = 2000, n = 10, p = 3$ and $\delta = 0.8$.

Experiment1. We firstly compute a random point U_1 on the Stiefel manifold $\mathrm{St}(n, p)$ (that is, a matrix $U_1 \in \mathbb{R}^{n\times p}$ such that $U_1^T U_1 = I_p$). Then, we compute a random tangent vector Δ in the tangent plane $T_{U_1}\mathrm{St}(n, p)$. We produce the point U_2 by following the direction of Δ with starting point U_1 ($U_2 = \mathrm{M.}\texttt{retr}(U_1, \Delta)$). Finally, we measure the error with the expression $error(t_i) = \angle(\alpha_{sh}(t_i), \mathrm{M.}\texttt{retr}(U_1, \Delta, t_i))$ where $\alpha_{sh}(t)$ for $t \in [0, 1]$ is the geodesic returned by the shooting method. Note that $\alpha_{sh}(0) = U_1$ and we hope that $\alpha_{sh}(1) \approx U_2$. The results are shown in table 3.10 and Figure 3.17.

Experiment2. Figure 3.18 analyzes the algorithm convergence behavior for different values of the fixed step size δ. The curve plotted represents the average number of iterations needed to reach a convergence tolerance τ. The average is taken on a sample of size 10. Empirical evidence shows that the algorithm converge with the fewest number of

iterations when the step size is close to $\delta = 1.2$.

Experiment3. In the third experiment, we investigate the average computation time of the Shooting method under various conditions. First, we show how the average computation time varies with increasing values for n and with fixed parameters $p = 3$ and $T = 80$. Secondly, we show how the average computation time varies when increasing p with fixed parameters $n = 400$ and $T = 80$. Finally, we do the same with varying T and fixed parameters $n = 400$ and $p = 3$. We do this for different convergence tolerances τ and for the fixed parameters $J = 2000$ and $\delta = 0.8$. We refer to appendix C.14 that shows the results for this experiment.

Experiment4. The goal here is to test whether the shooting method implemented for the log operator of a Stiefel manifold works correctly. The log operator for the Grassmann manifold is well defined in Manopt. The experiment checks if the results for the Stiefel manifold are similar to the results for the Grassman manifold on the low multilinear rank approximation problem seen in section 3.4. Theoretically, the results must on average be similar due to the invariance property seen in section 3.4.1. This is a good way to analyze if the shooting method works well for the Stiefel manifold. We do this with methods that uses the log operator, like DE or PSO. The results are shown in Figure 3.19 for DE and the results for PSO are shown in appendix C.15.

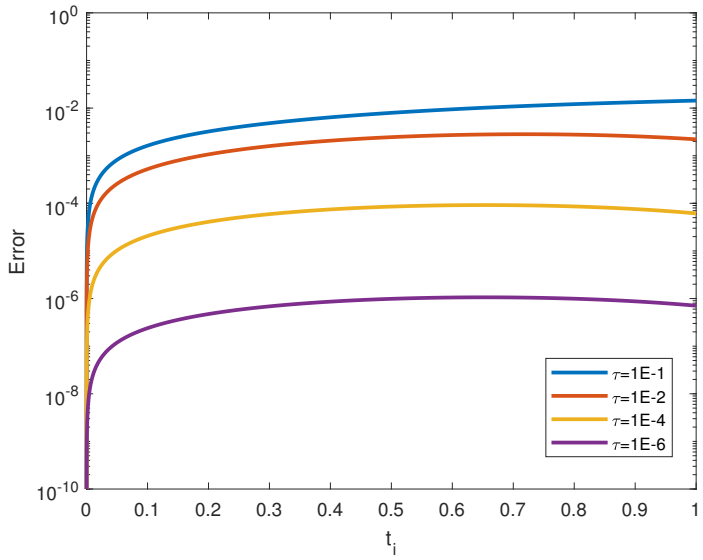


Figure 3.17: Experiment1: error of the shooting method at each t_i for different convergence tolerances τ. For a higher convergence tolerance in Algorithm 6, the error is smaller.

	Conv. tolerance τ	Mean error ($\pm$ standard deviation)
$T = 100$	$\tau = 10^{-2}$	0.002 ($\pm$ 9.61E-4)
	$\tau = 10^{-4}$	2.46E-5 ($\pm$ 1.12E-5)
	$\tau = 10^{-6}$	3.74E-7 ($\pm$ 1.65E-7)
$T = 1000$	$\tau = 10^{-2}$	0.0035 ($\pm$ 0.0015)
	$\tau = 10^{-4}$	6.043E-5 ($\pm$ 2.5E-5)
	$\tau = 10^{-6}$	4.54E-7 ($\pm$ 1.85E-7)
$T = 10000$	$\tau = 10^{-2}$	0.0020 ($\pm$ 8.59E-4)
	$\tau = 10^{-4}$	6.58E-5 ($\pm$ 2.63E-5)
	$\tau = 10^{-6}$	7.61E-7 ($\pm$ 3.03E-7)

Table 3.10: Results for the error of the shooting method for a Stiefel manifold $\mathrm{St}(n = 10, p = 3)$. The mean and the standard deviation are taken from the sample $\{error(t_i)\}_{i=1}^{T}$ where $t_i = i/T$. We can conclude that for a convergence tolerance 10^{-k} $(k = 1, 2, 4, 6)$, the expected error at a point t_i of the geodesic is maximum 10^{-k}.

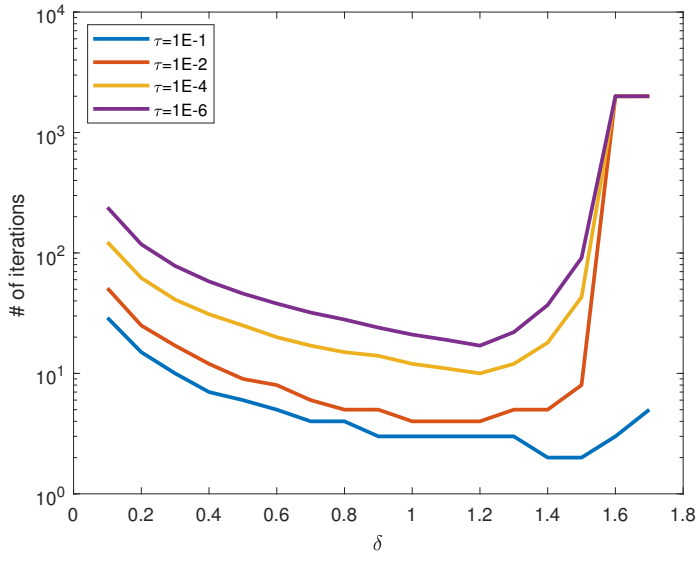
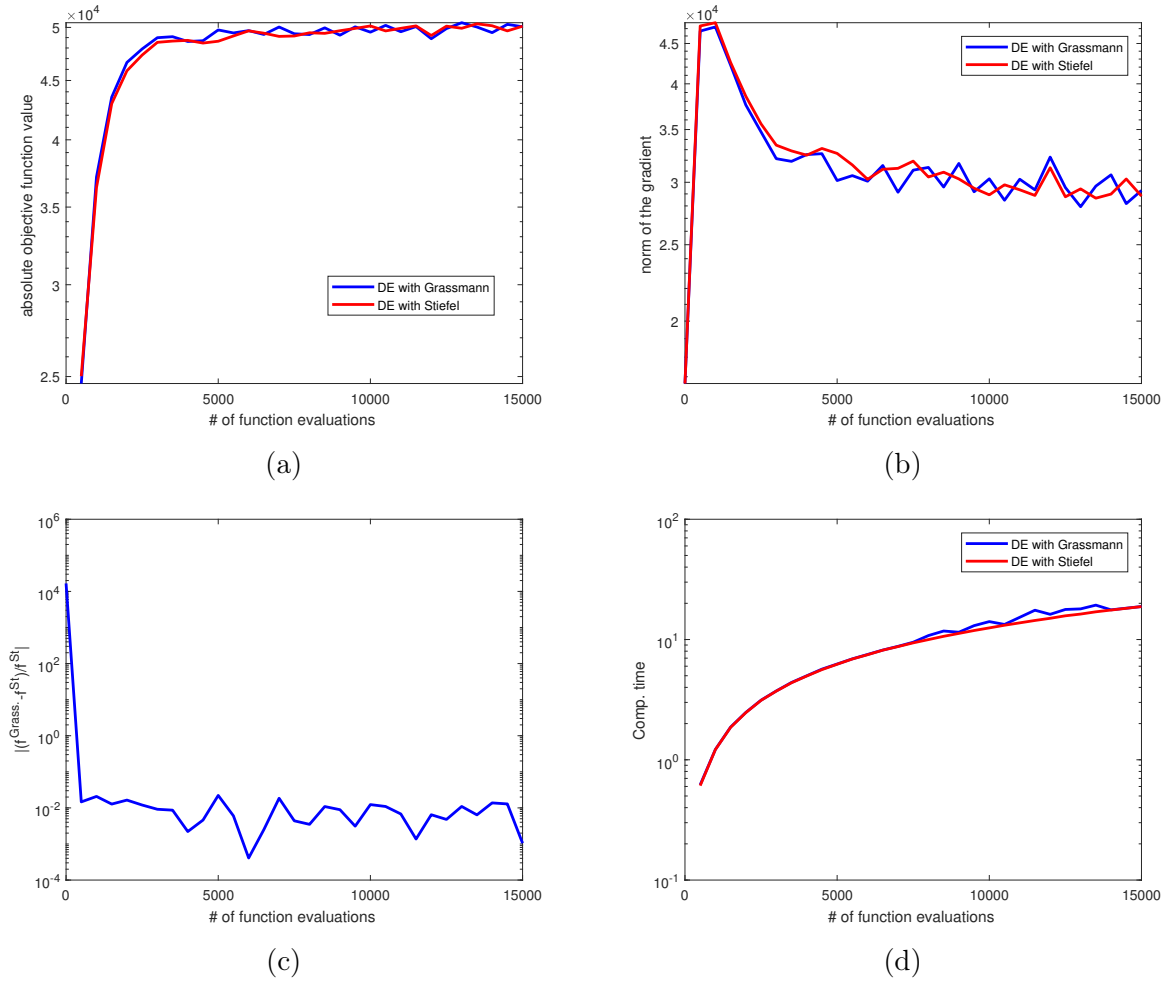


Figure 3.18: Experiment2: number of iterations needed to reach a convergence tolerance τ versus the step size δ. The algorithm converge with the fewest number of iterations when the step size is close to $\delta = 1.2$.



(a) (b) (c) (d)

Figure 3.19: Experiment 4: DE method applied for the low multilinear rank approximation problem on a tensor $\mathcal{A} \in \mathbb{R}^{10\times10\times10}$ of random entries in (0,1) and an imposed multilinear rank (2,2,2). Results are shown both for the Stiefel and Grassmann manifold (the sample is of size 5). It seems that the approximated log operator for the Stiefel manifold works correctly since the results are similar compared to the Grassmann manifold. Indeed, we nearly see the difference between the curves in Figure (a) and (d). The norms of the gradients are not that close to each other in Figure (b), this is probably due to the stochastic character of the methods. The relative error in Figure (c) defined by $\frac{f^{Grass}-f^{St}}{f^{Grass}}$ is generally close to zero.

Conclusion

The purpose of this master's thesis was to investigate a few global optimization algorithms that are relevant on the Euclidean space and generalize them to the manifold setting. I managed to implement the Simulated Annealing (SA) and the Differential Evolution (DE) algorithm with their accelerated versions (AccSA, AccDE and AccCDE) on the manifold setting. I analyzed their working on the Euclidean space in the first chapter. In the second chapter, we saw how these novel algorithms were adapted on manifolds and defined some useful tools available on Manopt. I applied the novel methods to four different problems in chapter 3 and deduce some conclusions.

It seems that the implementation of the novel methods in Matlab was a success since they showed coherent results in chapter 3. Depending on the problem, the novel methods showed interesting/disappointing results compared to the already generalized methods. I decided to compare the derivative-free methods (DE, SA, CDE) with the Particle Swarm Optimization method (PSO). The performances of the gradient based methods (AccSA, AccDE, AccCDE) were compared with the trustregions method (TR). For each problem, we deduced the following conclusions:

- *Problem 1: Sphere packing.* The results of the novel methods were quite disappointing for this problem. Indeed, it seems that PSO is the best method among all the derivative-free methods. The DE method is the closest to the performance of PSO and very similar for a small number of points to distribute on the sphere (see Figure 3.4).

- *Problem 2: Levy function.* What concerns the derivative-free algorithms, the performances of SA and CDE were clearly better than PSO. Indeed, it seems that PSO and DE got easily "trapped" in a local minimum while SA and CDE were able to jump out of this local minimum to find a better one (see Figure 3.8). The novel gradient based methods showed also satisfactory results since they were all able to reach the global minimum, while the TR method got trapped in the first local minimum met (see Figure 3.10).

- *Problem 3: Sphere fitting.* These results were very satisfactory for the DE method. Indeed, after a certain number of iterations, the DE method was able to find a better solution than the well-known approximated solution. I illustrated how DE managed to reach an objective value that is 20% better than the objective value

of the approximated solution when fitting a curve on random data points on the sphere (see Figure 3.13). The performances of PSO were similar to DE but never better than DE for this problem. However, the computation time for PSO and DE to generate a better solution takes a lot of time for this problem (about one hour for 6 points on the manifold), while the approximated solution is generated after a few seconds (see Table 3.5).

- *Problem 4: Low-rank multilinear approximation.* This problem is a very complex problem and is worth for further analysis. It seems that our derivative-free methods are able to reach a local minimum (see Figure 3.15). The accelerated methods depends heavily on the performance of TR and are thus not ideal to be analyzed on this probblem. From several sources, we can confirm that this problem presents a lot of local minima and it is very hard to switch from one minimum to the other with stochastic methods.

Generally, we can thus conclude that each novel derivative-free methods shows satisfactory results for one of the four problem. The most satisfactory result is probably the performance of DE on the sphere fitting problem. The interest of the accelerated methods is mainly showed for the Levy function where we observed that these methods were able to switch from a local minimum to another one. This is a real add-on for the Manopt toolbox that contains no accelerated methods that mix global and local search.

Finally, we also implemented an iterative algorithm to approximate the log operator for the Stiefel manifold. Results in chapter 3 show that the committed error is more than acceptable. We proved by a simple example applied on the low rank approximation problem that the shooting algorithm works correctly.

For further works, it would be interesting to have a concise description of the low rank approximation problem. An analysis of the complexity of the problem seems unavoidable in order to apply novel methods on this problem.

Concerning the sphere fitting problem, it would be interesting to analyze the effect of the curvature on the approximated solution. Indeed, it seems that for paths with a greater curvature, some novel methods are able to find a significantly better solution (about 20%) than the approximated solution. This may be due to the greater curvature of the paths imposed by the problem, but a further analysis could be worth interesting to describe the phenomenon.

I didn't manage to implement an improved version of SA. However, several resources passed throughout this work mention some improved SA method (like [42]).

Finally, other algorithms could be generalized to manifolds and it could be interesting to compare their performances on manifolds with the already generalized ones. As said in the introduction, a myriad of resources exists in the literature concerning algorithms for global optimization and some of them could be worth exploring on manifolds.

List of Figures

List of Tables

Appendices

Appendix A

Complements to chapter 1

A.1 Convergence and cooling schedules for SA

The main terms in Table 1.1 that will influence the convergence of the SA algorithm are the probability density function, the generative neighbour function **CreateNeighbourSolution(**$x_{current}$**)** (let's denote this generative function $g(x)$) and also the cooling schedule $T(k, max\_temp)$. In this subsection, we introduce the main schemes that will influence the convergence and that were summarized in [43].

A.1.1 Convergence

Boltzmann annealing. This annealing scheme was essentially introduced as a Monte Carlo importance-sampling technique for doing large-dimensional path integrals arising in statistical physics problems [44]. The algorithm chooses for the following generative function,

$$g(x) = (2\pi T)^{-n/2} exp\left[\frac{-\Delta x^2}{2T^2}\right]$$

where $\Delta x = x - x_0$ is the deviation of x from x_0, n is the dimension of the space and T the temperature.

Given $g(x)$, it has been proven [45] that it suffices to obtain a global minimum of the objective function $E(x)$ if the cooling schedule is decreasing logarithmically and not faster than the following equation:

$$T(k, max\_temp) = \frac{max\_temp}{\log(k)}$$

In [43], this is showed by demonstrating that any point in the space can be sampled infinitely often in annealing-time by showing that the products of probabilities of not generating a state x is zero.

Fast Annealing. It was noted that the Cauchy distribution has some definite advantages over the Boltzmann form [46]. The Cauchy distribution,

$$g(x) = \frac{T}{(\Delta x^2 + T^2)^{(n+1)/2}}$$

has a fatter tail than the Gaussian form of the Boltzmann distribution, permitting easier access to test local minima in the search for the desired global minimum. As for the Boltzmann annealing, it sufficies to have a cooling schedule not faster than

$$T(k, max\_temp) = \frac{max\_temp}{k}$$

This is again proved in [43] with the same principle used for the Boltzmann annealing.

A.1.2 Cooling schedules

The two previously presented cooling schedules for the Boltzmann and Fast annealing schemes ensure the convergence but require a prohibitive computing time, so it is necessary to consider faster methods of temperature decrease. We will now present some cooling schedules that are more practical.

Exponential multiplicative cooling. This scheme is used as reference in the comparison among the different cooling criteria [46]. The temperature decrease is made by multiplying the initial temperature by a factor α that decreases exponentially with respect to temperature cycle k:

$$T(k, max\_temp) = \alpha^k max\_temp$$

where $0 < \alpha < 1$.
The question now is, how can we determine the value of α? To do this, we have to impose some requirements and design a strategy. Our strategy in Algorithm 1 is to impose a maximum number of cost evaluations ($max\_costevals$) and a maximum number of iterations at each temperature (max_iter_temp) and observe the results returned by SA. We require that the temperature must be near zero at the end of the algorithm ($min\_temp \approx 0$). This, means that we have to impose that:

$$T(k\_final) = min\_temp$$

and note that $k\_final = max\_costevals/max\_iter$ in Algorithm 1. Since we imposed that max_iter is a divider of $max\_costevals$, we have that $k\_final \in \mathbb{N}_0$. We can now extract the value of α by solving the following equation:

$$T(k\_final) = \alpha^{k-final} max\_temp = min\_temp$$

and we obtain finally:

$$\alpha = \exp\left(\frac{\log\left(\frac{min\_temp}{max\_temp}\right)}{k\_final}\right)$$

Note that we verify that $\alpha < 1$ since $min\_temp << max\_temp$ and $k\_final > 0$.

Logarithmical multiplicative cooling. For this scheme, we incorporate a parameter β of cooling speeding-up that makes possible the use of this scheme in practice . Without the factor β, the scheme would require a prohibitive computing time. The temperature decrease is made multiplying the initial temperature by a factor that decreases in inverse proportion to the natural logarithm of temperature cycle k:

$$T(k, max\_temp) = \frac{max\_temp}{1 + \beta\log(1 + k)}$$

where $\beta > 1$.
Again, we impose the same strategy and requirements as for the Exponential multiplicative cooling to determine the value of β. We thus impose that:

$$\frac{max\_temp}{1 + \beta\log(1 + k\_final)} = min\_temp$$

and we obtain that:

$$\beta = \frac{\frac{max\_temp}{min\_temp} - 1}{\log(1 + k\_final)}$$

Since the fraction $\frac{max\_temp}{min\_temp}$ is near infinity, we can assure that $\beta > 1$.

Linear multiplicative cooling. The temperature decrease is made multiplying the initial temperature by a factor γ that decreases in inverse proportion to the temperature cycle k:

$$T(k, max\_temp) = \frac{max\_temp}{1 + \gamma k}$$

where $\gamma > 0$.

A.2 Convergence and mutation schemes for DE

The different mutation schemes in Table 1.2 may influence the convergence of DE. A convergence condition has been proposed to ensure the convergence in probability of DE [47]. We will firstly develop what we understand by convergence in probability and describe the convergence condition.

A.2.1 Convergence

The following definition of convergence, that is, convergence in probability, is used in [47].

Definition 17 (convergence in probability [47]). *Let $\{X_k, k = 0, 1, 2...\}$ be a population sequence generated by DE to solve the optimization problem of the same kind as the problem described in 1.2.1. Then DE converges to the global optimum, if and only if:*

$$\lim_{k\to\infty} P\left\{X_k \cap S_\delta^* \neq \emptyset\right\} = 1$$

where $P\{A\}$ denotes the probability of a certain event A and S_δ^* is a set defined as follows:

$$S_\delta^* = \left\{x | E(x) - E(x^*) < \delta\right\}$$

where δ is a small positive value, $E(x)$ is thus the objective function and x^* the optimal solution.

Let us give a sufficient condition for the convergence of DE.

Theorem 1 (sufficient condition [47]). *In the $t_k$target population X_{t_k}, there exists at least one individual x which corresponds to the trial individual y by a reproduction operator, such that*

$$P\{y \in S_\delta^*\} \geq \xi(t_k) > 0$$

and the series $\sum_{k=1}^{\infty} \xi(t_k)$ diverges; then DE converges to the optimal solution set S_δ^*.
Where $\{t_k = 1, 2, ...\}$ denotes any subsequence of natural number set, $P\left\{y \in S_\delta^*\right\}$ denotes the probability that y belongs to the optimal solution set S_δ^*, and $\xi(t_k)$ is a small positive value which may change as t_k.

A proof of Theorem 1 is given in [47]. To resume, DE can converge to the global optimal solution if its reproduction operation satisfies the sufficient conditions given in Theorem 1.

A.2.2 Mutation schemes

An additional mutation step to assist the classical DE was developed to ensure the convergence [47]. The additional mutation step is called the AsCo-mutation (Assist Convergence) operator and is defined as follow:

Definition 18 (AsCo-mutation). *AsCo-mutation is a mutation operator assisting the classical DE to converge. It satisfies the following conditions:*

- *To a certain sub-sequence $\left\{X(t_k), k = 1, 2, ...\right\}$ of population sequence $\left\{X(k), k = 1, 2, ...\right\}$, AsCo-mutation changes at least one individual in each $\left\{X(t_k), k = 1, 2, ...\right\}$ with a positive probability.*

- *Let $Y(t_k)$ denote the population generated by using AsCo-mutation; there exists at least on individual y in $Y(t_k)$ such that:*

$$P\{y \in S_\delta^*\} \geq \xi(t_k) > 0$$

and the series $\sum_{k=1}^{\infty} \xi(t_k)$ diverges.

When we say that the AsCo-mutation operator "assists" the classical DE, we mean that we have to add an additional step in Algorithm 2 (just after the mutation step) to satisfy the convergence condition as described in [47].

We will now introduce the most common mutation operators that are the Uniform mutation and Gaussian mutation. In [47], it is proven that these operators meet the definition of AsCo-mutation.

Uniform Mutation. Uniform mutation replaces the solution vector x with a uniformly distributed random vector w confined in the domain of the objective function. Each component of the vector w is a uniformly distributed random number between $[0, 1]$. The implementation of the Uniform mutation can be flexible. It is proven that the operators presented in table 1.2 generate individuals that are ergodic in the search space, just like Uniform mutation operator.

Gaussian Mutation. Gaussian mutation modifies all components of the solution vector x by adding a random noise:

$$\bar{x} = x + \nu$$

where ν is a vector of independent random Gaussian numbers with zero mean and standard deviations σ.

Since uniform and Gaussian mutation operators satisfy Definition 2, we can conclude the following theorem:

Theorem 2. *DE algorithm employing uniform mutation or Gaussian mutation operators converges in probability to the global optimum of the optimization problem.*

Appendix B

Complements to chapter 2

B.1 Examples of tangent spaces

Tangent space of a sphere. Suppose $t \to x(t)$ be a curve in the unit sphere S^{n-1}. Since $x(t) \in S^{n-1}$ for all t, we have that:

$$x^T(t)x(t) = 1$$

for all t. Differentiating this equation with respect to t yields:

$$\dot{x}^T x(t) + x^T x(t)\dot{x}(t) = 0 \tag{B.1}$$

Suppose a point y on the curve $x(t)$, and thus $y \in S^{n-1}$. Then, according to equation 2.1, we say that z is a tangent vector at y if:

$$\Big\{ z \in \mathbb{R}^n | y^T z = 0 \Big\}$$

This corresponds to the set of all vectors orthogonal to x in $\mathbb{R}^n$. We thus have that the tangent space of a sphere at y is defined as:

$$T_y S^{n-1} = \Big\{ z \in \mathbb{R}^n | y^T z = 0 \Big\}$$

More over, consider the function $F : \mathbb{R}^n \to \mathbb{R} : x \to x^T x - 1$. Remark that $S^{n-1} = \big\{ x \in \mathbb{R}^n | F(x) = 0 \big\}$, it follows that

$$T_x S^{n-1} = \ker(\mathrm{D}F(x)) = \Big\{ z \in \mathbb{R}^n | x^T z = 0 \Big\}$$

Tangent space of the Stiefel manifold. Recall that the Stiefel manifold is defined as follows:

$$\mathrm{St}(n,p) = \Big\{ X \in \mathbb{R}^{n \times p} | X^T X = I_p \Big\}$$

Suppose again that the curve $t \to X(t)$ be a curve in $\mathrm{St}(p,n)$, this means $X(t) \in \mathbb{R}^{n\times p}$ and

$$X^T(t)X(t) = I_p$$

for all t. By differentiating we have:

$$\dot{X}^T(t)X(t) + X^T(t) + \dot{X}(t) = 0$$

Suppose Y on the curve $X(t)$, and thus $Y \in \mathrm{St}(n,p)$. We say that Z is a tangent "matrix" at Y if:

$$\left\{ Z \in \mathbb{R}^{n\times p} | Y^T Z + Z^T Y = 0 \right\}$$

The tangent space of a Stiefel manifold at Y is thus defined as follows:

$$T_Y \mathrm{St}(n,p) = \left\{ Z \in \mathbb{R}^{n\times p} | Y^T Z + Z^T Y = 0 \right\}$$

B.2 Convergence on manifolds

The notion of convergence on manifolds is a straightforward generalization of the $\mathbb{R}^n$ case. An infinite sequence $\{x_k\}, k = 0, 1, ...$ of points of a manifold $\mathcal{M}$ is said to be *convergent* if there exists a chart $(\mathcal{U}, \phi)$ of $\mathcal{M}$, a point $x^* \in \mathcal{U}$, and a $K > 0$ such that x_k is in $\mathcal{U}$ for all $k \geq K$ and such that the sequence $\left\{\phi(x_k)\right\}, k = K, K+1, ...$ converges to $\phi(x^*)$.

Linear convergence. A sequence $\{x_k\}\, k = 0, 1, ...$ of points of $\mathbb{R}^n$ is said to converge linearly to a point x^* if there exists a constant $c \in (0,1)$ and an integer $K \geq 0$ such that, for all $k \in K$, it holds that $||x_{k+1} - x_*|| \leq c||x_k - x_*||$. If $\mathcal{M}$ is a Riemannian manifold, then the induced Riemannian distance makes it possible to define linear convergence as follows.

Definition 19 (linear convergence [4])**.** *Let $\mathcal{M}$ be a Riemannian manifold and let* dist *denote the Riemannian distance on $\mathcal{M}$. We say that a sequence $\{x_k\}\, k = 0, 1, ...$ converges linearly to a point $x^* \in \mathcal{M}$ if there exists a constant $c \in (0,1)$ and an integer $K \geq 0$ such that, for all $k \geq K$, it holds that*

$$\mathrm{dist}(x_{k+1}, x_*) \leq c\mathrm{dist}(x_k, x_*) \tag{B.2}$$

The limit:

$$\lim_{k\to\infty}\sup \frac{\mathrm{dist}(x_{K+1}, x_*)}{\mathrm{dist}(x_k, x_*)}$$

is called the linear convergence factor of the sequence. An iterative algorithm on $\mathcal{M}$ is said to converge linearly to a point x_* if there exists a neighbourhood $\mathcal{U}$ of x^* and a constant $c \in (0,1)$ such that, for every point $x_0 \in \mathcal{U}$, the sequence $\{x_k\}$ generated by the algorithm satisfies (2.3).

Superlinear convergence. In contrast to linear convergence, the notions of superlinear convergence and order of convergence can be defined on a manifold independently of any other structure.

Definition 20 (superlinear convergence [4])**.** *Let $\mathcal{M}$ be a manifold and let $\{x_k\}_{k=0,1,\ldots}$ be a sequence on $\mathcal{M}$ converging to x_*. Let $(\mathcal{U}, \phi)$ be a chart of $\mathcal{M}$ with $x \in \mathcal{U}$. If*

$$\lim_{k\to\infty} \frac{||\phi(x_{k+1}) - \phi(x_*)||}{||\phi(x_k) - \phi(x_*)||} = 0$$

then $\{x_k\}$ is said to converge superlinearly to x_*. If there exist constants $p > 0$, $c \geq 0$ and $K \geq 0$ such that for all $k \geq K$, there holds

$$||\phi(x_{k+1}) - \phi(x_*)|| \leq c||\phi(x_k) - \phi(x_*)||^p \tag{B.3}$$

then $\{x_k\}$ is said to converge to x_* with order at least p. An iterative algorithm on $\mathcal{M}$ is said to converge locally to a point x_* with order at least p if there exists a chart $(\mathcal{U}, \phi)$ at x^* and a constant $c > 0$ such that, for every initial point $x_0 \in \mathcal{U}$, the sequence $\{x_k\}$ generated by the algorithm satisfies (2.4). If $p = 2$, the convergence is quadratic.

Appendix C

Complements to chapter 3

C.1 Default parameters for the methods

Mode	Methods	Parameters			
Derivative-free	SA	max_temp	min_temp	iter_temp	
		2000	10E-7	40	
	SSA	max_temp	min_temp	iter_temp	
		2000	10E-7	40	
	DE (P)	CR	scheme		
		0.7	DE/best/1		
	CDE (P)	K	x_{min}	x_{min}	
		100	random	random	
	PSO (P)				
Gradient based	AccSA	disturbance	max_stoch	local	global
		0.1	1000	TR	SA
	AccDE (P)	disturbance	max_stoch	local	global
		0.1	1000	TR	DE
	AccCDE (P)	K	max_stoch	local	global
		100	1000	TR	DE
	TR				

Table C.1: Parameters for the methods. The (P) denotes the population based methods. The parameters of PSO and TR method are chosen according to the default parameters in Manopt.

C.2 arccos(x)

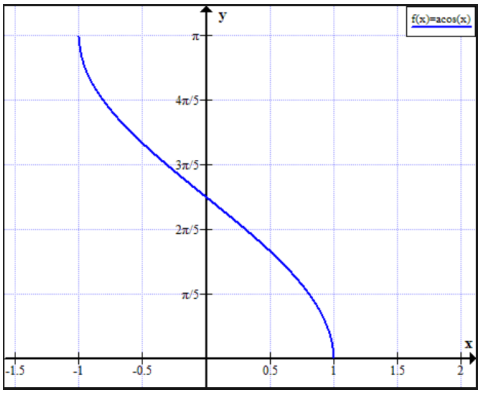


Figure C.1: Illustration of $f(x) = \arccos(x)$.

C.3 Sphere packing with the Oblique and Elliptope manifold

Furthermore, it is apparent that f is only a function of XX^T. It is important to understand that what we enter as input argument in the objective function given by 3.3 is a distribution of p-points whose dimension is n. These p-points are thus represented in a matrix X of size $p \times n$. The output argument is a real value. Note that $f(X) = f(XQ)$ for any orthogonal matrix $Q \in O(n)$. Indeed, applying a global rotation to the points on the sphere does not change the distances separating them.

Note that we can use two different manifolds for this problem. We can firstly represent the search space of this optimization problem with the oblique manifold. The minimization problem described in (3.3) becomes:

$$\min_{X \in \mathrm{Obl}(p,n)} f(X) = \epsilon \log \left(\sum_{1 \leq i < j \leq p} \exp \left(\frac{XX^T}{\epsilon} \right) \right) \tag{C.1}$$

where the matrix X represents thus p-points on the sphere $\mathbb{S}^{n-1}$.

Similarly, we can represent this optimization problem with fixed-rank elliptope manifold. The minimization problem described in 3.3 becomes finally:

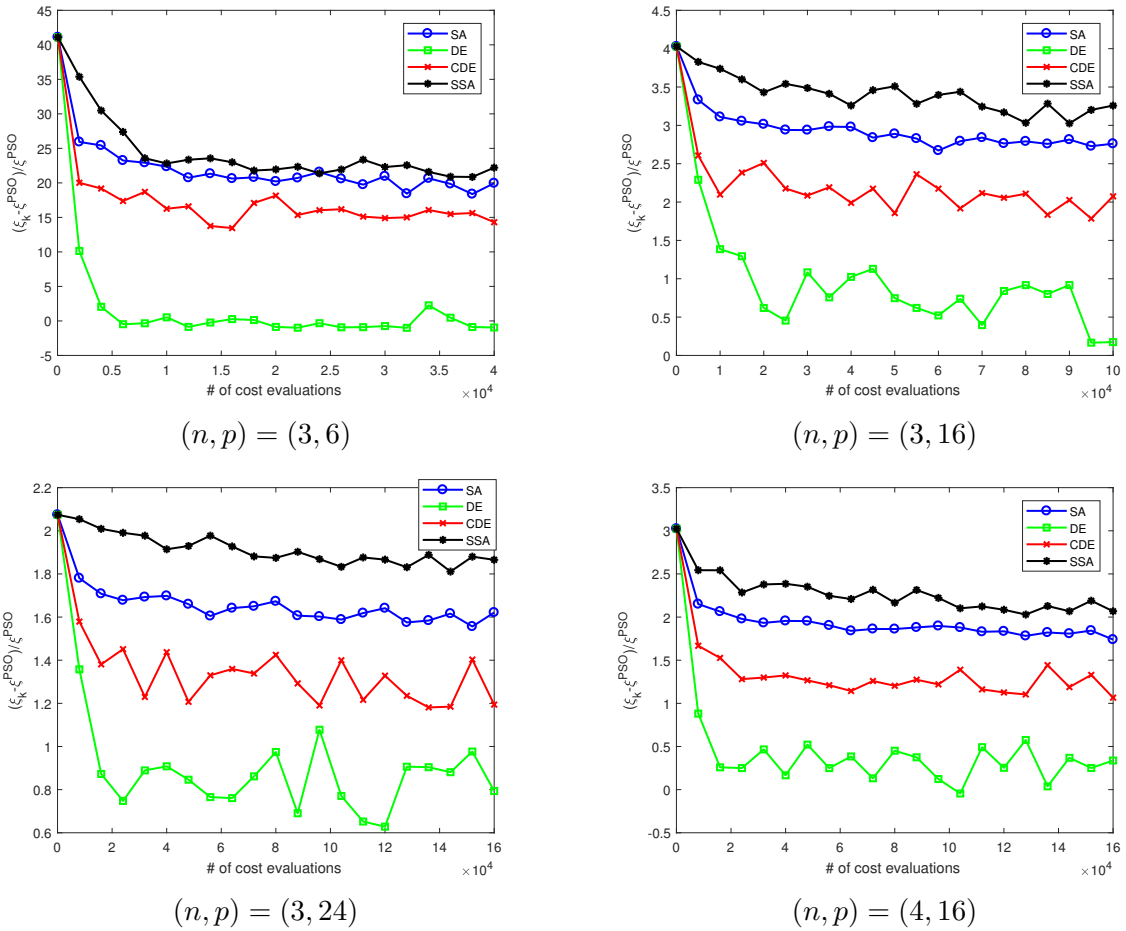
$$\min_{Y \in \mathrm{Elip}(p,n)} f(Y) = \epsilon \log \left(\sum_{1 \leq i < j \leq p} \exp \left(\frac{Y}{\epsilon} \right) \right) \tag{C.2}$$

where $Y = X^T X$ with the matrix X representing the p-points on the sphere $\mathbb{S}^{n-1}$.

We can thus pick one of the two manifolds. From [5] we get the following information: *"The elliptope factory quotients out the global rotation invariance of the problem, which is more natural but conceptually a bit more complicated."*

We thus decided to use the oblique manifold.

C.4 Comparison with PSO for the sphere packing problem



$(n, p) = (3, 6)$ $(n, p) = (3, 16)$

$(n, p) = (3, 24)$ $(n, p) = (4, 16)$

Figure C.2: Comparison of the novel methods with PSO by means of differences of relative errors. It seems that DE is the closest method to the performance of PSO.

C.5 Gradient of the modified Levy function

The modified Levy function described in subsection 3.2.1 is given by:

$$E(x_1, x_2) = \sum_{i=1}^{4} icos[(i+1)x_1 + i] \sum_{j=1}^{4} icos[(j+1)x_2 + j] + 0.5\big((10x_1 + 1.45)^2 + (0.0002x_2 + 0.8)^2\big) + 100$$

The gradient can be computed with the basic rules of partial derivatives:

$$\begin{pmatrix} \frac{\partial E}{\partial x_1} \\ \frac{\partial E}{\partial x_2} \end{pmatrix} = \begin{pmatrix} -\sum_{i=1}^{4} i(i+1)\sin[(i+1)x_1 + i] \sum_{j=1}^{4} i\cos[(j+1)x_2 + j] + 10(10x_1 + 1.45) \\ -\sum_{i=1}^{4} i(i+1)\sin[(i+1)x_2 + i] \sum_{j=1}^{4} i\cos[(j+1)x_2 + j] + 0.0002(0.0002x_1 + 0.8) \end{pmatrix}$$

C.6 Levy function: effect of the parameter δ.

Figure C.3 shows how the parameter δ influences the distribution.

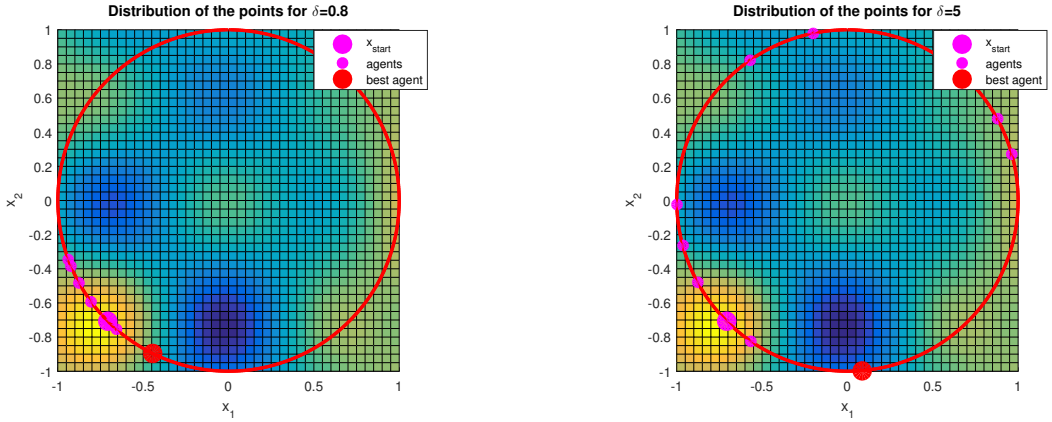


Figure C.3: Effect of the parameter δ. We see that the larger the value for δ, the more uniformly the points are distributed on the circle. The red point represents $x\_start = [-\sqrt{2}/2, -\sqrt{2}/2]$ and the points in magenta represent the *agents* of the initial distribution around x_{start} we enter as input argument for the population based methods (like DE, CDE and PSO). The big point is the agent with the lowest objective value from which we will let start the SA and TR algorithm. We conclude that a high value of δ is better in order to cover a big part of the search space and thus, if we are lucky, start the algorithms from a point that is near a local minimum (or even near the global minimum).

C.7 Levy function: plots for the mutation schemes and different CR

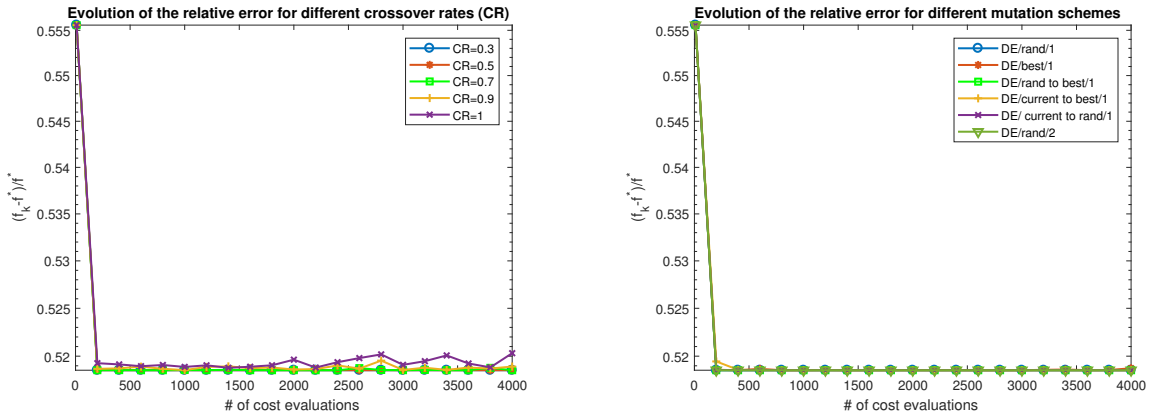


Figure C.4: Parameter analysis of the crossover rate (CR) and the mutation schemes in DE. We start from the point $x_{start} = (0, 1)$. The differences are tiny, so we choose $CR = 0.7$ and the scheme DE/best/1.

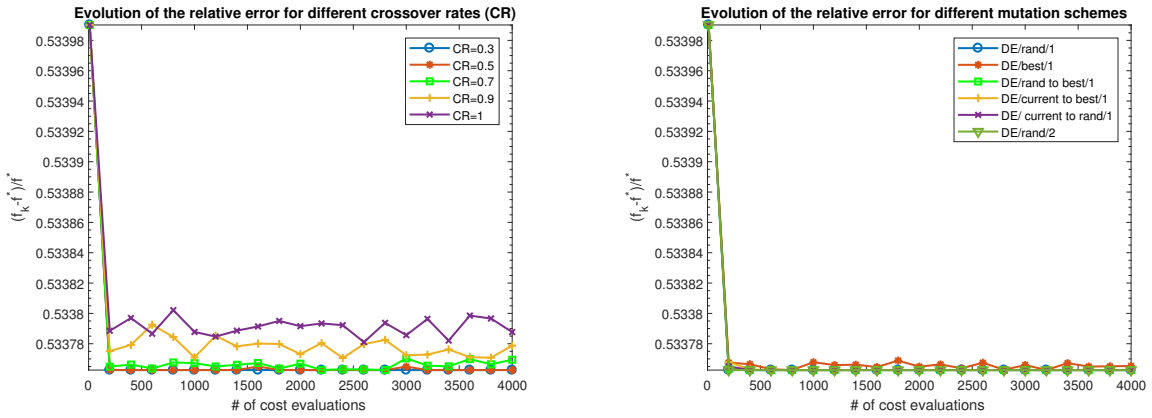


Figure C.5: Parameter analysis of the crossover rate (CR) and the mutation schemes in DE. We start from the point $x_{start} = (-1, 0)$. Even if the differences are so tiny, the best choice seems to be the setting $CR = 0.5$ and the scheme DE/rand/2.

C.8 Levy function: performances of the derivative-free methods for $x_{start} = (-1, 0)$

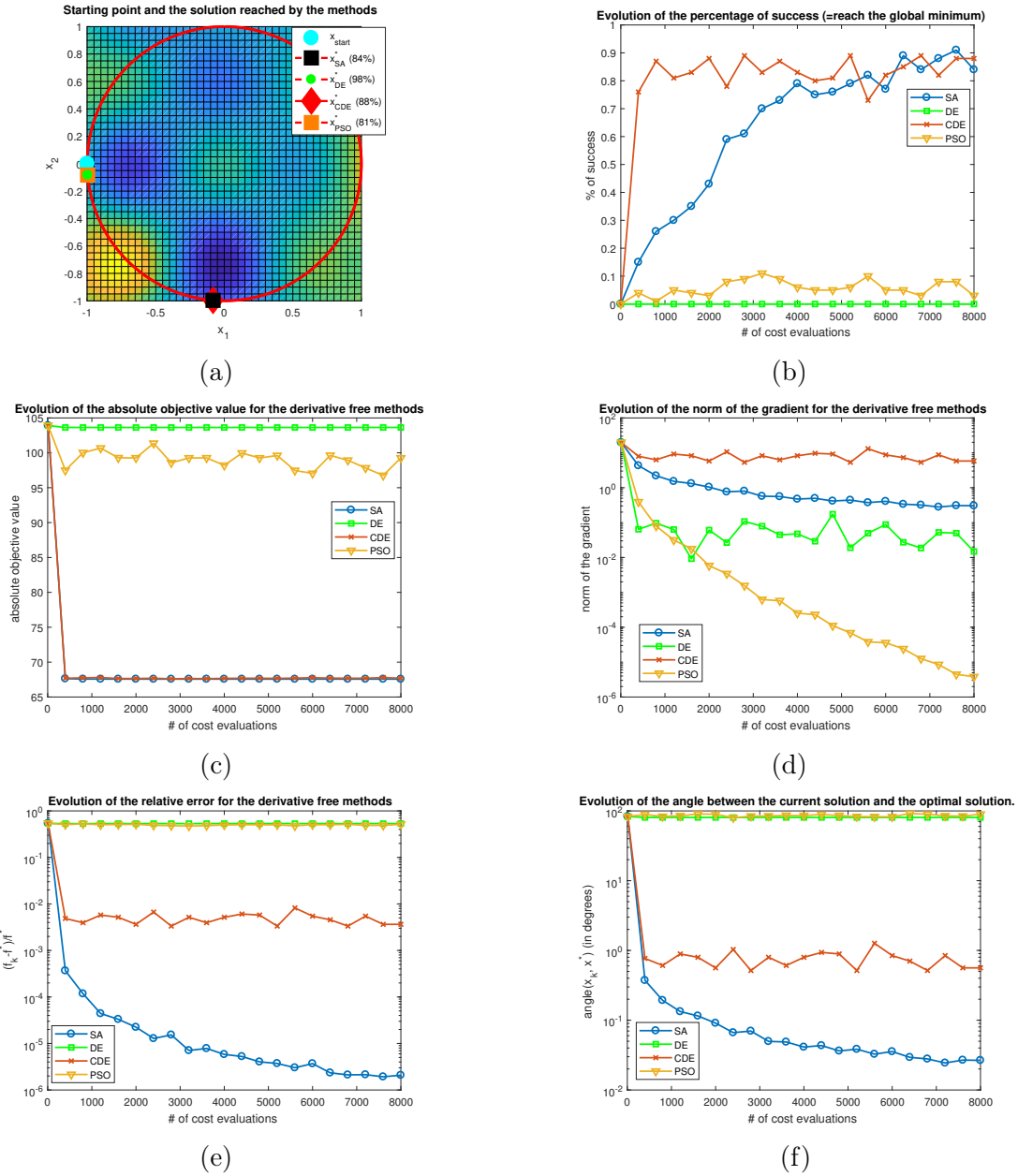


Figure C.6: Obtained results for the derivative-free methods when we start from the point $x_{start} = (-1, 0)$. Figure (a) illustrates where the solutions returned by the methods are situated on the circle after 8000 cost evaluations. The additional percentage in the legend indicates the probability we have to land on the point shown in Figure (a) after 8000 iterations. It seems that CDE and SA reach the global minimum while PSO and DE are "trapped" in a local minmimum. This is confirmed by Figure (b)-(c)-(d)-(e). Figure (f) describes where the iterates are located from the global solution.

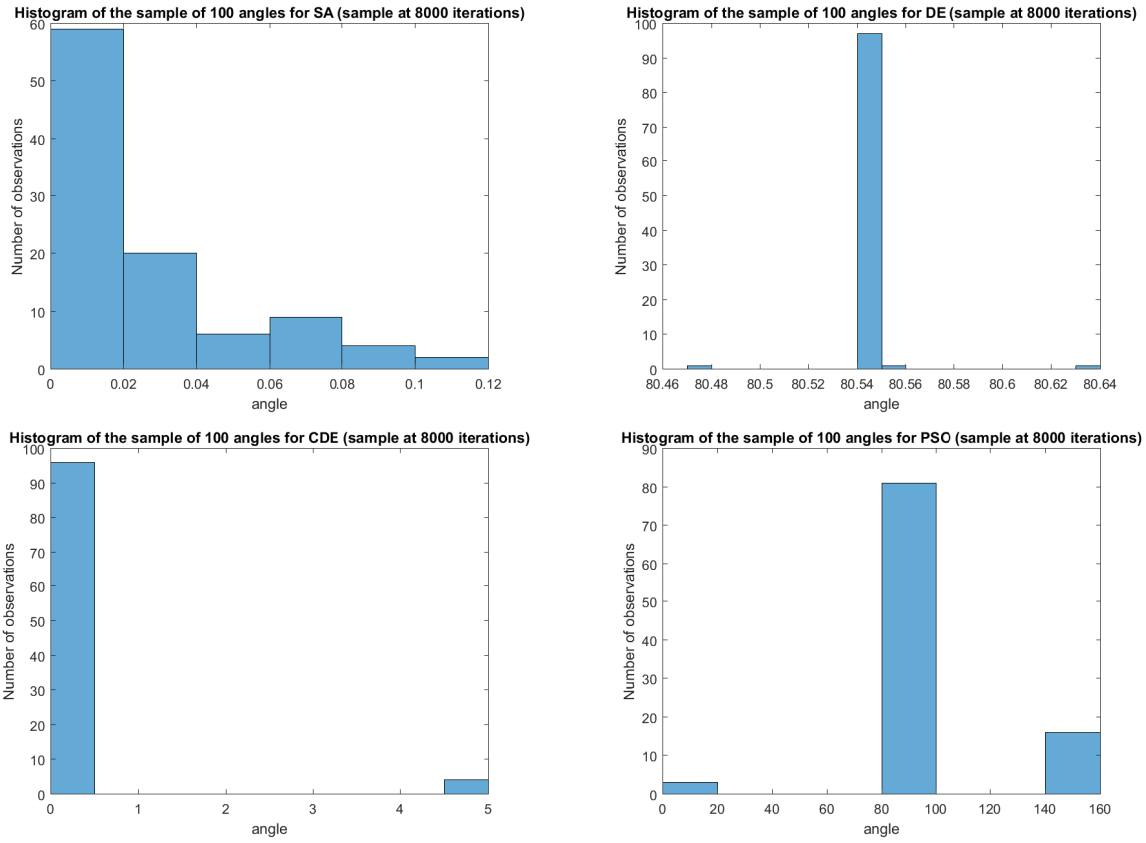


Figure C.7: Histograms of the sample of 100 angles between the solution reached by the methods after 8000 cost evaluations and the global solution when we start from $x_{start} = (-1, 0)$. We observe that the returned solution by SA is on average at an angle of almost 0° from the global solution. The returned solution of the DE and PSO method is at an angle of nearly 80° degrees of the global solution. The PSO method can also reach a solution that is situated at an angle of approximately 150° of the global solution. The returned solution of the CDE can be situated at an angle of 4.5° or almost 0° degrees from the global solution.

C.9 Levy functions: performances of the accelerated methods for $x_{start} = (0, 1)$

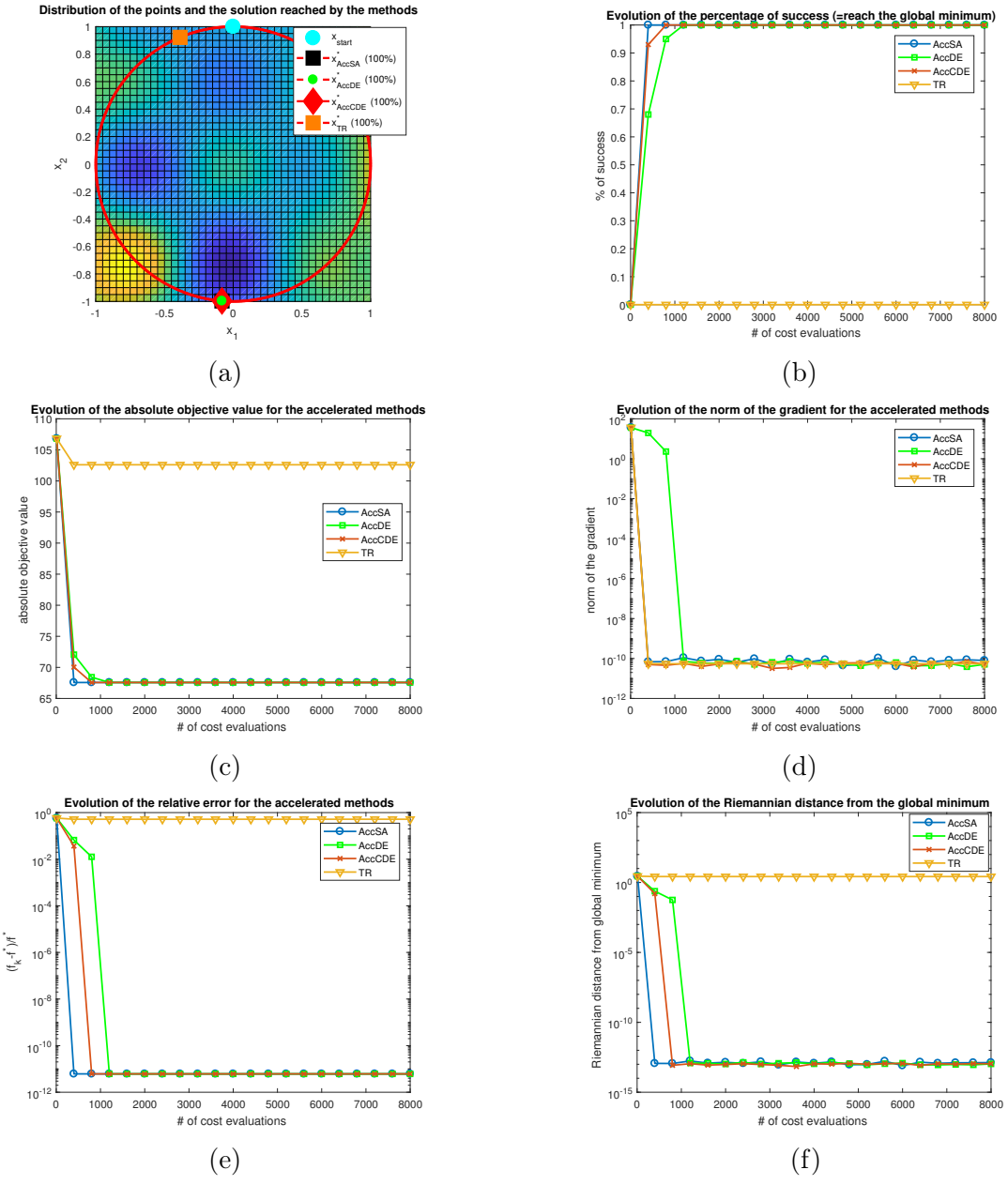


Figure C.8: Obtained results for the gradient based methods when we start from the point $x_{start} = (0, 1)$. Figure (a) illustrates where the solutions returned by the methods are situated on the circle after 8000 cost evaluations. The additional percentage in the legend indicates the probability we have to land on the point shown in Figure (a) after 8000 iterations. It seems that AccCDE, AccSA and AccDE reach the global minimum while TR is "trapped" in a local minimum. This is confirmed by Figure (b)-(c)-(d)-(e). Figure (f) describes where the iterates are located (on average) from the global solution.

C.10 Finite differences and trapezium rule

We define a curve $\gamma : [0,1] \mapsto \mathcal{M}$ on a manifold $\mathcal{M}$. Suppose the discretization times τ_i with spacing $\Delta\tau$. When $\mathcal{M}$ is a Euclidean space, a first order finite difference formula is given by:

$$v_i = \frac{\gamma_{i+1} - \gamma_i}{\Delta\tau}$$

As seen in chapter 2, we define the difference of two points on the manifold as follows: $\gamma_{i+1} - \gamma_i$ is a vector rooted at γ_i and pointing toward γ_{i+1}. We thus have by the logarithmic map associated to the manifold $\mathcal{M}$

$$v_i = \frac{\mathrm{M.}\texttt{log}(\gamma_i, \gamma_{i+1})}{\Delta\tau}$$

where v_i is a tangent vector in the tangent space $T_{\gamma_i}\mathcal{M}$ pointing toward γ_{i+1} and such that $\|v_i\| = \mathrm{dist}(\gamma_{i+1}, \gamma_i)/\Delta\tau$.

We use the same trick for second order derivatives. The classic formula is:

$$a_i = \frac{\gamma_{i-1} - 2\gamma_i + \gamma_{i+1}}{\Delta\tau^2} = \frac{(\gamma_{i-1} - \gamma_i) + (\gamma_{i+1} - \gamma_i)}{\Delta\tau^2}$$

and becomes

$$a_i = \frac{\mathrm{M.}\texttt{log}(\gamma_i, \gamma_{i-1}) + \mathrm{M.}\texttt{log}(\gamma_i, \gamma_{i+1})}{\Delta\tau^2}$$

The trapezium rule is a way of estimating the area under a curve. The trapezium rule works by splitting the area under a curve into a number of trapeziums, which we know the area of. We suppose here that the base of the trapeziums is equal to $\Delta\tau$.

With these two concepts, the regularization term of the objective function in section 3.3.1 becoms:

$$\underbrace{\sum_{i=0}^{n-1} \int_0^1 \left\| \ddot{\beta}^i(t) \right\|_{\mathcal{M}}^2 \mathrm{d}t}_{regularization} = \sum_{k=1}^{n-1} \Delta\tau \left\| \frac{\mathrm{M.}\texttt{log}\left(\mathcal{B}(t_{k-1}), \mathcal{B}(t_k)\right) + \mathrm{M.}\texttt{log}\left(\mathcal{B}(t_k), \mathcal{B}(t_{k+1})\right)}{\Delta\tau^2} \right\|_{\mathcal{M}}^2$$

where $\mathcal{B}$ is the composite Bézier function of the piecewise functions $\beta^i(t)$ where $i = 1, ..., n-1$ as defined in section 3.3.1.

C.11 Sphere fitting: 8 random points and $\lambda = 80$.

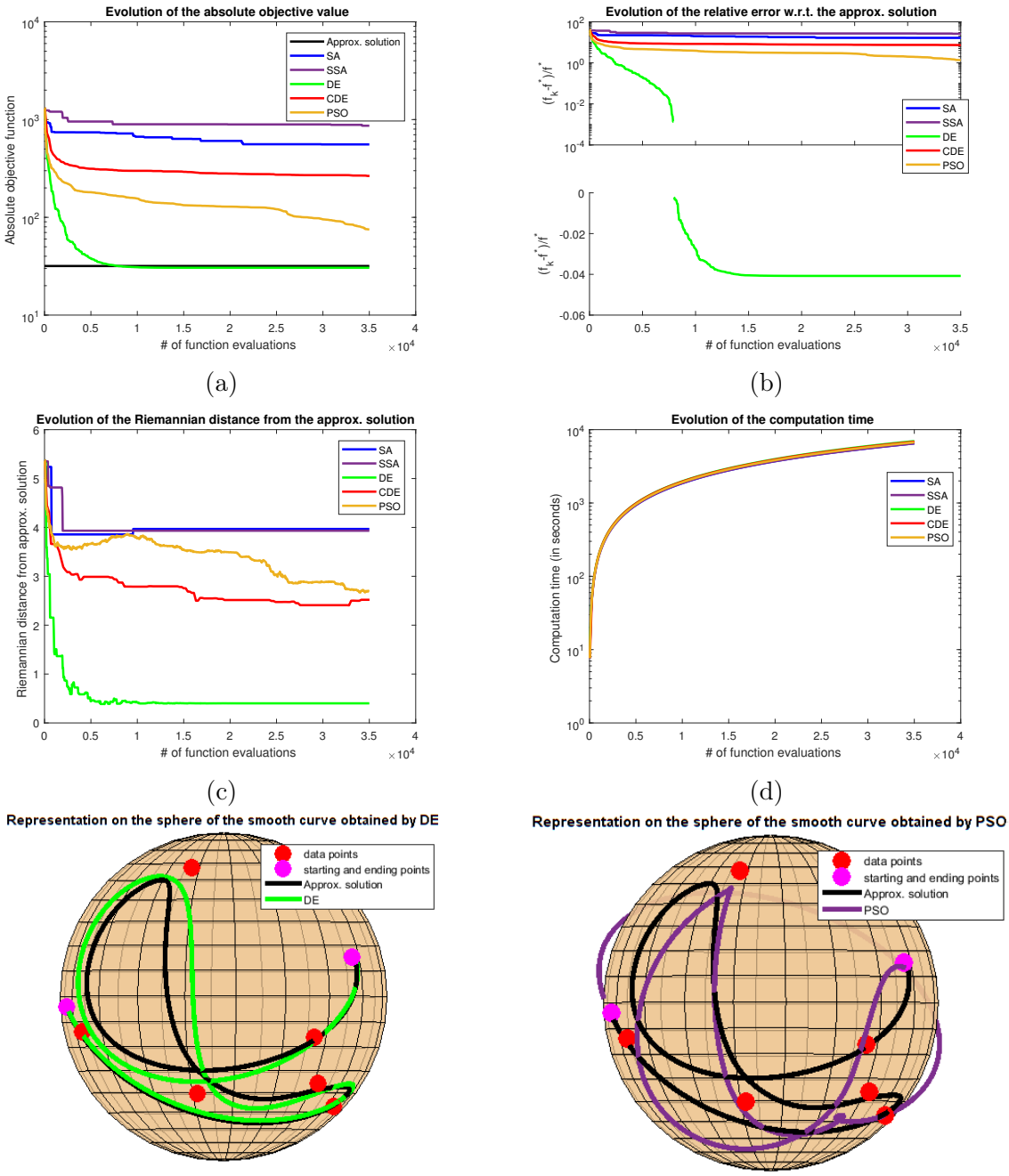


Figure C.9: Results for the sphere fitting problem of eight random data points on the sphere and $\lambda = 80$. From Figure (a) and (b), we observe that DE is able to reach a lower objective value than the objective value f^* of the approx. solution. The relative error indicates that the objective value of DE is almost 4% smaller than f^*. Figure (c) shows that the Riemannian distance between the solutions returned by DE are at a value of 0.6 from the approx. solution. Figure (e) and (f) illustrate the different paths for PSO and DE compared to the approx. solution.

C.12 Low rank approx. problem: tables for V and W

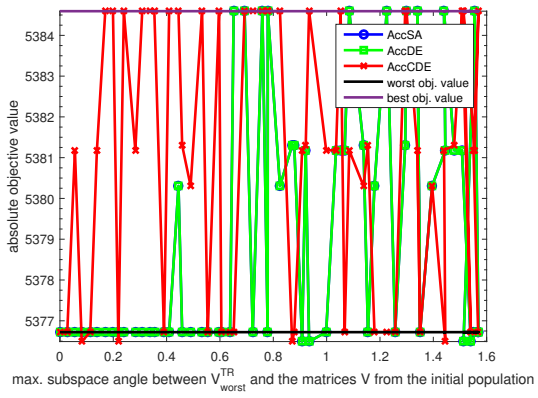
	V_1^{TrustR} (5384.6)	V_2^{TrustR} (5381.3)	V_3^{TrustR} (5384.6)	V_4^{TrustR} (5384.6)	V_5^{TrustR} (5376.7)
V_1^{TrustR}	0	1.334	0	0	1.3968
V_2^{TrustR}		0	1.334	1.334	1.1478
V_3^{TrustR}			0	0	1.3968
V_4^{TrustR}				0	1.3968
V_5^{TrustR}					0

Table C.2: Subspace angles between the different matrices V. The objective values reached by TR are in brackets.

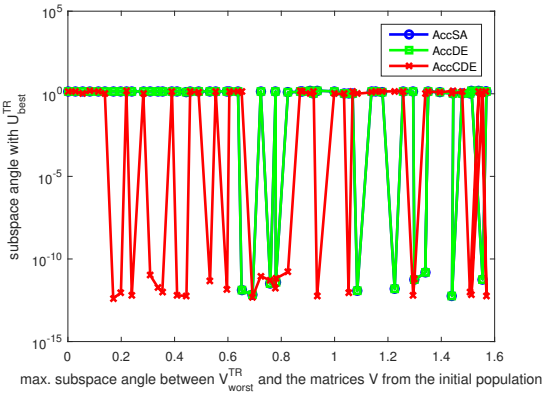
	W_1^{TrustR} (5384.6)	W_2^{TrustR} (5381.3)	W_3^{TrustR} (5384.6)	W_4^{TrustR} (5384.6)	W_5^{TrustR} (5376.7)
W_1^{TrustR}	0	1.4328	0	0	1.2041
W_2^{TrustR}		0	1.4328	1.4328	1.2661
W_3^{TrustR}			0	0	1.2041
W_4^{TrustR}				0	1.2041
W_5^{TrustR}					0

Table C.3: Subspace angles between the different matrices W. The objective values reached by TR are in brackets.

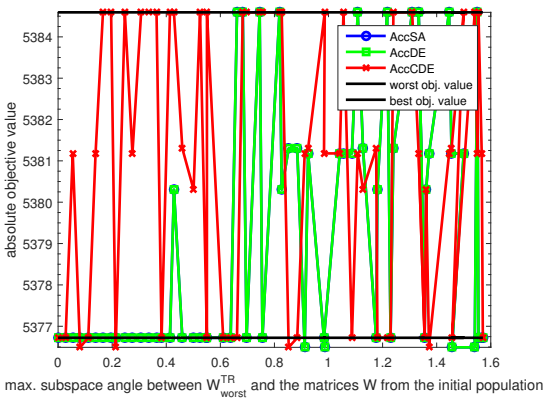
C.13 Low rank approx. problem:results for V and W



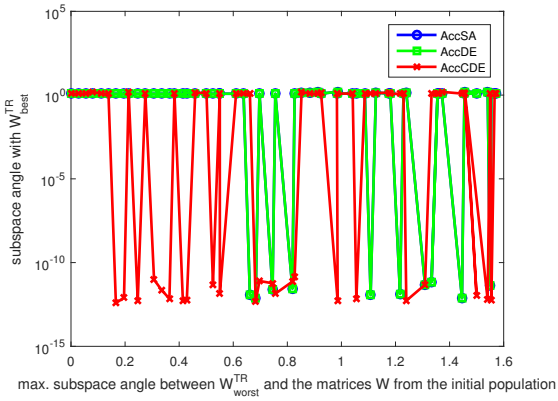
(a)



(b)

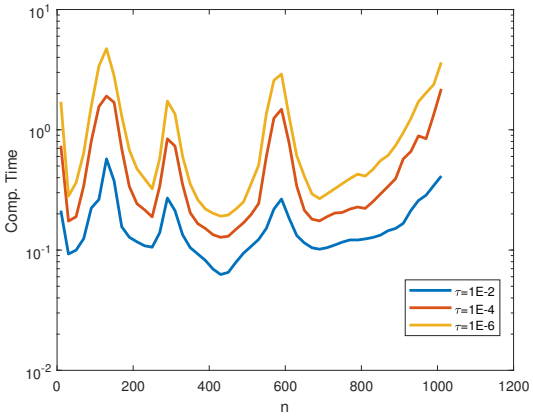


(c)

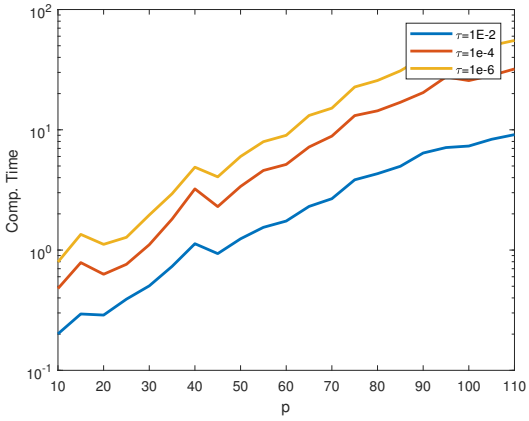


(d)

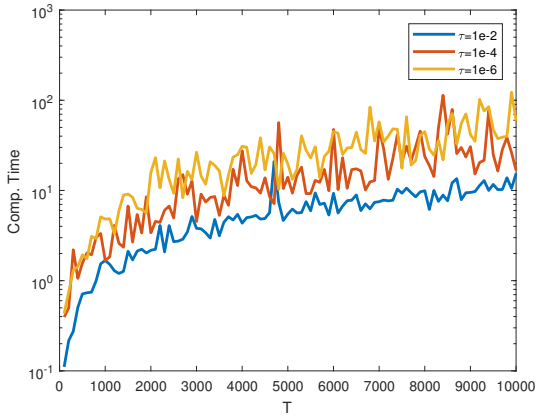
C.14 Shooting method: analysis of the parameters n, p and T



(a) Average computational time vs. n.



(b) Average computational time vs. p.



(c) Average computational time vs. T. Only one sample is represented.

Figure C.11: Experiment3: Average computation time for the shooting method. The average is taken on a sample of size 100 for n and p. As expected for increasing p and T, the computation time increases.

C.15 Shooting method: comparison with Stiefel and Grassmann manifold for PSO

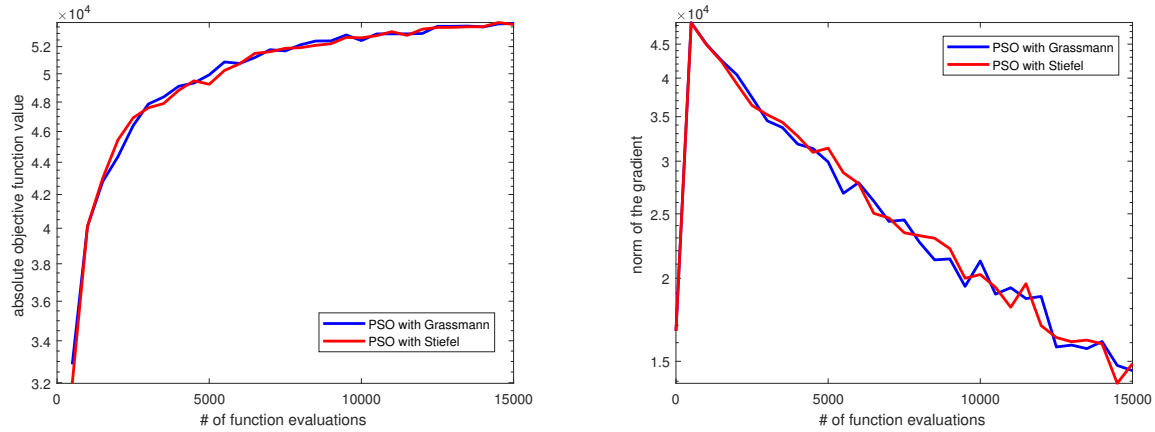


Figure C.12: Experiment 4: PSO method applied for the low multilinear rank approximation problem on a tensor $\mathcal{A} \in \mathbb{R}^{10\times10\times10}$ of random entries in [0,1] and an imposed multilinear rank (2,2,2). Results are shown both for the Stiefel and Grassmann manifold (the sample is of size 5). It seems that the approximated log operator for the Stiefel manifold works correctly since the results are similar compared to the Grassmann manifold. Indeed, we nearly see the difference between the curves in Figure (a) and (d). The norms of the gradients are not that close to each other in Figure (b), this is probably due to the stochastic character of the methods. The relative error in Figure (c) defined by $\frac{f^{Grass}-f^{St}}{f^{Grass}}$ is generally close to zero.

Bibliography

[1] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

[2] Marco Locatelli and Fabio Schoen. *Global optimization: theory, algorithms, and applications*, volume 15. Siam, 2013.

[3] Yong-Jin Liu, Chun-Xu Xu, Ran Yi, Dian Fan, and Ying He. Manifold differential evolution (mde): a global optimization method for geodesic centroidal voronoi tessellations on meshes. *ACM Transactions on Graphics (TOG)*, 35(6):243, 2016.

[4] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[5] Nicolas Boumal. *Optimization and estimation on manifolds.* PhD thesis, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2014.

[6] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[7] Jakob Vesterstrom and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1980–1987. IEEE, 2004.

[8] G Jeyakumar C Shanmugavelayutham. Convergence analysis of differential evolution variants on unconstrained global optimization functions. *arXiv preprint arXiv:1105.1901*, 2011.

[9] Efrñn Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A Coello Coello. A comparative study of differential evolution variants for global optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 485–492. ACM, 2006.

[10] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[11] Yong Wang, Zixing Cai, and Qingfu Zhang. Differential evolution with composite trial vector generation strategies and control parameters. *IEEE Transactions on Evolutionary Computation*, 15(1):55–66, 2011.

[12] Yong-Jun Wang and Jiang-She Zhang. Global optimization by an improved differential evolutionary algorithm. *Applied Mathematics and Computation*, 188(1):669–680, 2007.

[13] Robert M May. Simple mathematical models with very complicated dynamics. *Nature*, 261(5560):459, 1976.

[14] Yang Li-Jiang and Chen Tian-Lun. Application of chaos in genetic algorithms. *Communications in Theoretical Physics*, 38(2):168, 2002.

[15] Yong-Jun Wang and Jiang-She Zhang. An efficient algorithm for large scale global optimization of continuous functions. *Journal of Computational and Applied Mathematics*, 206(2):1015–1026, 2007.

[16] Nicolas Hadjisavvas and Panos M Pardalos. *Advances in Convex Analysis and Global Optimization: Honoring the Memory of C. Caratheodory (1873–1950)*, volume 54. Springer Science & Business Media, 2013.

[17] D. Martin Crossley. *Essential topolog*. Springer Undergraduate Mathematics Series, 2005.

[18] S. Caroll. Baking and maths, homeomorphisms of the torus, part 4 (topology of the identity component).

[19] S. Caroll. Lecture notes on general relativity. *Enrico Fermi Institute. University of Chicago*, 1997.

[20] Anuj Srivastava and Xiuwen Liu. Tools for application-driven linear dimension reduction. *Neurocomputing*, 67:136–160, 2005.

[21] Thomas J Bridges and Sebastian Reich. Computing lyapunov exponents on a stiefel manifold. *Physica D: Nonlinear Phenomena*, 156(3-4):219–238, 2001.

[22] Fabian J Theis, Thomas P Cason, and P-A Absil. Soft dimension reduction for ica by joint diagonalization on the stiefel manifold. In *International Conference on Independent Component Analysis and Signal Separation*, pages 354–361. Springer, 2009.

[23] Frank Tompkins and Patrick J Wolfe. Bayesian filtering on the stiefel manifold. In *Computational Advances in Multi-Sensor Adaptive Processing, 2007. CAMPSAP 2007. 2nd IEEE International Workshop on*, pages 261–264. IEEE, 2007.

[24] P.-A. Absil. Stiefel manifolds and their applications. 2009.

[25] Anders Eriksson and Anton Van Den Hengel. Optimization on the manifold of multiple homographies. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 242–249. IEEE, 2009.

[26] Darshan Bryner. Endpoint geodesics on the stiefel manifold embedded in euclidean space. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1139–1159, 2017.

[27] Alan Edelman, Tomás A Arias, and Steven T Smith. The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[28] Sylverin Kemmoé Tchomté and Michel Gourgand. Particle swarm optimization: A study of particle displacement for solving continuous and combinatorial optimization problems. *International Journal of Production Economics*, 121(1):57–67, 2009.

[29] James Blondin. Particle swarm optimization: A tutorial. *from site: http://cs. armstrong. edu/saad/csci8100/pso tutorial. pdf*, 2009.

[30] R. Eisele. The log-sum-exp trick in machine learning. *Systems Architect and DBA*, 2016.

[31] E. Süli and David Mayers. *An introduction to Numerical Analysis*. Cambridge University press, 2013.

[32] Pierre-Yves Gousenbourger, Laurent Jacques, and P-A Absil. Fast method to fit a $\mathcal{C}^1$ piecewise-bézier function to manifold-valued data points: How suboptimal is the curve obtained on the sphere $\mathbb{S}^2$? In *International Conference on Geometric Science of Information*, pages 595–603. Springer, 2017.

[33] Pierre-Yves Gousenbourger, Estelle Massart, Antoni Musolas, Pierre-Antoine Absil, Julien M Hendrickx, Laurent Jacques, and Youssef Marzouk. Piecewise-bézier c1 smoothing on manifolds with application to wind field estimation. *ESANN2017, to appear*, 2017.

[34] Lorenz Pyta and Dirk Abel. Interpolatory galerkin models for the navier-stokes-equations. *IFAC-PapersOnLine*, 49(8):204–209, 2016.

[35] Gerald E Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002.

[36] T. Popiel and L. Noakes. Bézier curves and $\mathcal{C}^2$ interpolation in riemannian manifolds. *Journal on Approximation Theory 148(2)*, 2007.

[37] P-A Absil, Pierre-Yves Gousenbourger, Paul Striewski, and Benedikt Wirth. Differentiable piecewise-bézier surfaces on riemannian manifolds. *SIAM Journal on Imaging Sciences*, 9(4):1788–1828, 2016.

[38] Nicolas Boumal and P-A Absil. A discrete regression method on manifolds and its application to data on so (n). *IFAC Proceedings Volumes*, 44(1):2284–2289, 2011.

[39] Gene H Golub, Alan Hoffman, and Gilbert W Stewart. A generalization of the eckart-young-mirsky matrix approximation theorem. *Linear Algebra and its applications*, 88:317–327, 1987.

[40] Mariya Ishteva, P-A Absil, Sabine Van Huffel, and Lieven De Lathauwer. Tucker compression and local optima. *Chemometrics and Intelligent Laboratory Systems*, 106(1):57–64, 2011.

[41] Mariya Ishteva, P-A Absil, Sabine Van Huffel, and Lieven De Lathauwer. Best low multilinear rank approximation of higher-order tensors, based on the riemannian trust-region scheme. *SIAM Journal on Matrix Analysis and Applications*, 32(1):115–135, 2011.

[42] Bo Yang, Liang Guang, Tero Säntti, and Juha Plosila. t (k)-sa: accelerated simulated annealing algorithm for application mapping on networks-on-chip. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 1191–1198. ACM, 2012.

[43] Lester Ingber. Very fast simulated re-annealing. *Mathematical and computer modelling*, 12(8):967–973, 1989.

[44] N. Metronolis. A. W. Rosenbluth. M. N. Rosenbluth. A. H. Teller and E. Teller. Ectuation of state calculations by fast computing machines. *J. them: Phys. 21*, 1960.

[45] DGeman SGeman. Stochastic relaxation, gibbs distribution and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 16:721, 1984.

[46] Harold Szu and Ralph Hartley. Fast simulated annealing. *Physics letters A*, 122(3-4):157–162, 1987.

[47] Zhongbo Hu, Shengwu Xiong, Qinghua Su, and Xiaowei Zhang. Sufficient conditions for global convergence of differential evolution algorithm. *Journal of Applied Mathematics*, 2013, 2013.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**