Particle swarm optimization algorithms, analysis and computing

van Koot, K.A.A.M.

Award date:
2014

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Author
K.A.A.M. (Kenneth) van Koot

Supervisor
Prof. dr. ir. B. (Barry) Koren

Date
August 18, 2014

Particle Swarm Optimization Algorithms, Analysis and Computing.

Abstract

In this thesis the concept of particle swarm optimization is presented. First the origin of the particle swarm algorithm will be outlined and the particle swarm optimization algorithm will be analysed. Therefore a small one-dimensional test problem is used. From this test problem guidelines for the selection of the parameters in the particle swarm optimization algorithm will be given. The parameter selection criteria are applied on two more optimization problems, which are finding the maximum value of a specific function and the brachistochrone curve problem. The result and the algorithm's performance on these optimization problems will be discussed. The results show that a good parameter selection will enhance the use of the particle swarm optimization algorithm. For the purpose of a quick search towards the region of the global optimum of the problem another parameter selection is recommended than for the extensive search to the exact position of the global optimum.

Acknowledgments

This thesis is the result of my graduation project in completion of the Master of Science program Industrial and Applied Mathematics with specialization Computational Science and Engineering at the Eindhoven University of Technology. This work concludes all the years that I have spent at this university. Numerous people have contributed in various ways to help me finish this project, for which I would like to express my deep gratitude.

First I would like to acknowledge my supervisor Barry Koren, who has helped my decide on a topic for my graduation project. I would like to thank Barry for all the useful feedback and support he gave me during this project.

Furthermore I would like to thank all my close friends that I have made over the years of my study. Without their presence and support the time spent during my study would not have been as fun. Also I would like to thank my fellow students that spent their study time with me at GEWIS. The fellow bartenders and friends from dispuut GELIMBO have ensured some wonderful memories to which we can look back with joy.

Finally I am very grateful to my family. Their unlimited care and support kept me going through all these years. Claire thanks for all your love and support, without it the final years of my study would not have been as wonderful.

Kenneth van Koot,
July 2014

Contents

Chapter 1

Introduction

1.1 Problem description

Particle swarm optimization, PSO, is an optimization method that has been developed recently. The particle swarm optimization algorithm has ties with generic algorithms and originates from studies that were monitoring the behavior of birds in a flock. It is an iterative method that uses randomness on its way to find the global optimum of a given problem. The algorithm consists only of simple calculations and is therefore very straightforward to implement. It uses a set of particles, called swarm, that, at convergence, will represent valid solutions to the optimization problem. Each variable in the problem that has to be optimized can be seen as a position value. Each particle will be a point in an n-dimensional space. For each particle in the swarm the goal function of the optimization problem is calculated. Each particle's positions and the values where its goal function has the best value will be stored. Also the position and the value where the goal function has the best value over all the particles will be stored.
Now the way the particle swarm optimization algorithm works is that all the particles will move through the n-dimensional space. Each new position represents a new solution to the optimization problem. After each iteration step of the algorithm the goal function will again be calculated for all the particles and compared with the stored values. If the new position of a particle leads to a better value of the goal function the stored position and value of that particular particle will be updated. Also the overall best value and position is updated when necessary. The way the particles will move through the n-dimensional search space is that the particles will be drawn towards these stored position values. On its way the PSO algorithm finds better solutions to the optimization problem and converges towards the globally best solution. Every particle is attracted by its own best position so far and by the globally best position found so far. Furthermore each particle will have an inertia term that ensures the particles to sample the search space properly.
In general, a PSO algorithm has three problem dependent parameters. There is one parameter for the inertia term and two parameters for the attraction terms. These attraction terms are also each weighted by a random number. These random numbers ensure an

improved exploring behavior of all the particles. Without it the algorithm becomes deterministic and the final solution of the particle swarm algorithm will just be a linear combination of the initial solutions.
The choice of these parameters will influence the performance of the algorithm. They will influence the search behavior of the algorithm, from a coarse exploration of the search space to a detailed exploitation of it. This will have an effect on the accuracy of the final solution found by the algorithm and on the convergence speed of the algorithm. How to choose these parameters wisely will be analysed for several problems in this thesis.

1.2 Thesis outline

This chapter serves as an introduction to the particle swarm optimization method. A description of the origin and the build up of the particle swarm optimization algorithm is given in chapter 2. In chapter 3 the particle swarm optimization algorithm is analysed and some parameter selection guidelines will be derived. The performance of the algorithm on the brachistochrone curve problem will be analysed in chapter 4. This thesis is concluded in chapter 5.

Chapter 2

Historical overview

Particle swarm optimization can be used as a method for optimization of continuous non-linear functions. First the origin and the development of the algorithm are described. The algorithm is based on a simplified social model that has ties with swarming theory and has been described first by Kennedy and Eberhart [1].
A number of scientists were interested in the movement of birds in a flock and have created some computer simulations to describe the process. The models they used relied on manipulation of inter-individual distances. They described the flocking behavior as a function of the birds' effort to maintain an optimal distance between themselves and their neighbours. The assumption was made that the social sharing of information offers an evolutionary advantage. This hypothesis was fundamental to the development of particle swarm optimization.

The development of the particle swarm optimization algorithm is best described by explaining its conceptual development. The algorithm began as simulations of simplified social models, like bird flocking. The original intent of those simulations was to graphically simulate the sudden movements of birds in a flock. A first simulation relied on nearest neighbour velocity matching and, so called, craziness. A group of particles, birds, was randomly distributed with an initial position and an initial velocity. At each iteration step for every particle its nearest neighbour was determined and then the velocity of that particle was assigned to the particle that was considered. This method resulted in a rather good model for the description of the movement, however the flock quickly got a uniform unchanging direction. In order to get a more accurate behavior a stochastic variable, craziness, was added to the model. At each iteration step the variable changes the velocity of some randomly chosen particles. This causes some more variation into the system of particles, which leads to a more realistic movement of all the particles.
One of the drawbacks of this first model was that the variation was random and artificial. Therefore in a following simulation a dynamic force was introduced. The simulation used the nearest neighbour velocity matching, but instead of the craziness variable the particles were attracted by some selected sink point. This caused the particles to move around in a realistic way until they eventually set in the sink point.

The addition of the sink point led to some new insights into the way birds fly in a flock. In the simulation the sink point was known to all particles, but in real life birds will land anywhere where they can find food. In their search for food, birds in a flock are capable to benefit from knowledge that other individuals have gathered. This social sharing of information gives them an evolutionary advantage to find the best food sources. Therefore the model was adapted to try to simulate this behavior.
This new simulation created an initial set of particles on a plane. The plane represents the search space where the birds can find food. This food source was set in a certain fixed position in the plane. At each iteration step every particle calculated its present position in the plane and its distance to the food source. Every particle stored its shortest distance to the food source and the corresponding position in the plane in a variable called p_1. Now at each iteration step of the algorithm the velocity of each particle was adjusted in a simple manner. If the particle was to the right of its own p_1, then its velocity in the x-direction was adjusted by a negative random amount and if it was to the left the velocity was adjusted by a positive random amount. Thus: $v_x = v_x \pm r_1 b_1$, where v_x is the velocity in the x-direction, r_1 is a random number and b_1 is a parameter of the system. Similarly, if the particle was above or below its own p_1, then its velocity in the y-direction was adjusted by a negative or positive amount respectively. The parameter b_1 enables the user to influence the behavior of the algorithm. The user can adapt the sensitivity of the algorithm by choosing different values for b_1. Large values cause the velocity to change rapidly, while small values will lead to a more modest change in the velocity of the particle.
In order to introduce the sharing of information into the algorithm the variable p_2 was introduced. This variable represents the globally best position and corresponding distance found by any particle so far. It is known to all particles in the swarm. The velocities of all particles were adjusted by the influence of this variable in the same way as before. So if a particle was to the right of the p_2 its velocity in the x-direction was adjusted negatively and likewise for the other cases. Thus $v_{x,y} = v_{x,y} \pm r_2 b_2$, where r_2 is a random number not necessarily the same as r_1 and b_2 is a parameter of the system. The simulations performed with this algorithm showed good results. When the parameters b_1 and b_2 were chosen large all the particles rapidly approached the given food source. When the parameters were chosen small the particles circled around the food source approaching it in a realistic manner and finally all particles set in the food source.
In the algorithm both variables p_1 and p_2 play an important role. The variable p_1 and parameter b_1 introduce each particle's own experience into the system in the way that each particle tends to return to the position that most satisfies it so far. On the other hand the variable p_2 and parameter b_2 introduce the social behavior into the system which each particle wants to adhere to. Some further simulations of the algorithm showed that a large value for b_1 relative to b_2 led to endless wandering of single particles through the search space and that the reverse led to particles setting prematurely in local optima. The algorithm performed best when both parameters were chosen approximately equal. Both parameters were set to 2 in this algorithm, giving the stochastic factors a mean of 1. It was noted though that further research needed to be done to determine the optimal values for each particular problem.

A further improvement of the algorithm was to adjust the velocities in a different way. Instead of adjusting the velocities purely based on the position of the particle, the velocities were also adjusted according to the distance the particle has to the best positions. This led to the following formula for the velocity:

$$v_{k+1} = v_k + 2r_1(p_1 - x_k) + 2r_2(p_2 - x_k), \tag{2.1}$$

where x_k is the current position of the particle. In this formula both the velocity and the position of the particle can be considered as vectors or as scalars depending on the problem. Remark that the name velocity may be a bit misleading; in the formula above it is presented as a displacement. But since the displacement is taken over unit time steps it is preferred to name the parameter velocity. The new velocity of a particle is adjusted according to its previous velocity and the distances of its current position to its own best position and the group's best position. The particles' new positions would be updated via the following equation:

$$x_{k+1} = x_k + v_{k+1}. \tag{2.2}$$

During the development of the algorithm Kennedy and Eberhart [1] made use of five basic principles of swarm intelligence which were formulated by Millonas [2]. First is the proximity principle: the algorithm should be able to carry out simple space and time computations. Second is the quality principle: the particle population should be able to respond to quality factors in the environment. Third is the principle of diverse response: the particle population should be able to make a broad search covering the entire search space. Fourth is the principle of stability: the particle population should not change its mode of behavior every time the environment changes. Fifth is the principle of adaptability: the particle population should be able to change behavior when it is worth the computational price.
At first glance some principles seem to contradict each other. However, the particle swarm optimization algorithm given in (2.1) and (2.2) adheres to all the principles.
It consists of only simple calculations in a two-dimensional space for a discrete time space. The algorithm responds to the quality factors p_1 and p_2 and they also ensure a broad search through the entire space. The algorithm changes its mode of behavior only when p_2 changes, satisfying the fourth and fifth principle.

The particle swarm optimization algorithm was tested on various problems and proved to be an effective way to find an optimal solution. Sometimes the algorithm outperformed a gradient-based algorithm and even for the extremely nonlinear functions, such as Schaffer's F6 function, the algorithm found the global optimum each time.
Kennedy and Eberhart introduced the particle optimization method [1]. Later they proposed an algorithm that implemented the method in a slightly different way [3]. The original algorithm is globally oriented while the new algorithm is locally oriented. Next the locally oriented algorithm will be described and compared with the original algorithm.

The steps of the original algorithm are:

1. Initialize a set of particles with random positions and velocities,
2. Evaluate the problem-dependent minimization function,
3. Compare current position and value with p_2 and each particle's own p_1 and update these if required,
4. Change velocities via: $v_{k+1} = v_k + b_1 r_1(p_1 - x_k) + b_2 r_2(p_2 - x_k)$,
5. Update positions and loop to step 2 until a criterion is met.

This algorithm is globally oriented since the variable p_2 is known to all particles. In real life it is not realistic that all birds will immediately know all information of other birds in the flock. So it might not be realistic that all particles will immediately know the globally best solution found by a particular particle. Therefore there was the desire to design a locally oriented algorithm.
In this algorithm the particles will only know the information of a few of their nearest neighbours instead of the information of the entire group. In the algorithm the variable p_2 is replaced by the variable ${p_2}^l$. This variable represents the best position and corresponding value found by any particle in the neighbourhood of the considered particle. The size of the neighbourhood can vary. For example when the number of neighbours considered is two, the considered particle will get the information of its nearest neighbour to its left and right, while in a neighbourhood of size six, the considered particle will get the information of its six nearest neighbours.
When testing this algorithm the simulations showed that the model with a neighbourhood of size two would not get trapped in a local optimum. This was due to the fact that blocks of neighbours would separate and explore different regions, making it a more flexible search. However, for finding the optimal solution this algorithm required more iteration steps than the global algorithm. The simulations for the model with a neighbourhood of size six showed that the particles might get trapped in a local optimum. This is because of the stronger interconnection between the particles. The particles would get trapped in a local optimum less often than the global algorithm, but it required more iteration steps in finding the global optimum. Thus the algorithm with size two neighbourhoods is most resistant to local optima at the cost of more iteration steps. While expanding the size of the neighbourhood would reduce the number of iteration steps needed, it would introduce the chance in getting trapped in a local optimum just like in the original algorithm.
Another modification to the algorithm was made when Shi and Eberhart [4] did some further analysis on the algorithm. They had a closer look at the role of the different terms in the righthand side of equation (2.1). The first term is the previous velocity of the particle and the second and third term are the contributions to the change in velocity by the particle position. Without these latter terms the particles would keep moving in the same direction without a change in their velocities. In this case the algorithm will only find the global optimum if by coincidence it is lying on the path of one of the particles.

On the other hand if the first term of the equation is ignored and the velocities are only determined by the current position of the particle and the best known positions, then the velocity is memoryless. The particle with the best solution so far will not move at all, while the other particles will move to the average of their own best position and the globally best. The only time the algorithm will change is if a particle finds a new globally best position. Then that particle will stop moving and the other particles move towards a new average. So without their previous velocity all particles will contract towards the globally best position. A consequence of this is that the search space in which the particles move shrinks through the iteration steps of the algorithm and the global optimum can only be found if it is within the initial search space. So this algorithm becomes locally oriented and is very dependent on the initial position of the particles.
When the first term is added the current velocity of the particles is taken into account in the equation. Therefore the particles are able to explore new area and expand the initial search space. This leads to a more global search of the algorithm.

Now both the locally and globally oriented search have their own benefits in solving different problems. There is a tradeoff between the local and global search. In order to have a different balance between both searches an inertia weight, a, is brought into the equation. This a balances the role of the global and local search. It can be a positive constant, a positive linear function or even a nonlinear function. The equations to update the velocity and position become:

$$v_{k+1} = av_k + b_1 r_1(p_1 - x) + b_2 r_2(p_2 - x), \tag{2.3}$$

$$x_{k+1} = x_k + v_{k+1}. \tag{2.4}$$

To investigate the influence of the inertia weight Shi and Eberhart ran some simulations with the new algorithm. They used the optimization problem of finding the maximum of Schaffer's F6 function. This optimization problem is later also used in this thesis.

In their simulations the inertia weight, a, differed and all other parameters were fixed. They recorded the average number of iteration steps for the algorithm to find the global optimum. The maximum number of iteration steps in their test problem was fixed, if more iteration steps were required they claimed that the algorithm failed to find the optimum. The simulations show that for small inertia weights, $a < 0.8$, the particles tend to move quickly together and when the algorithm finds the global optimum it finds it fast. Thus the algorithm behaves more like a local search algorithm. If the global optimum is within the initial search space the algorithm will find it quickly. If the global optimum is not within the search space of the initial particles then the global optimum will not be found, because due to the inertia term the velocity of the particles will quickly tend to zero. On the other hand for large inertia weights, $a > 1.2$, the algorithm is more like a global search method and will exploit new area. Therefore the algorithm needs more iteration steps to find the global optimum and sometimes it will not find the global optimum at all.

Now for medium inertia weights, $0.8 < a < 1.2$, the algorithm has the best chance to find the global optimum within a moderate number of iteration steps. The simulations show that the algorithm with an a in this range will have less chance of failing to find the global optimum and find it within a reasonable number of iteration steps.
From these simulations it can be seen that the bigger the inertia weight is, the less dependent the solution is on the initial population and the more capable to explore new search areas. In general it is a good idea that an optimization algorithm has the ability to explore new area in the beginning to find a good seed and then refine the search more locally to find the exact global optimum. That is why it seems useful to make the inertia weight a time dependent parameter. When this was tested in the simulations it was found that this gave the best performance for the algorithm. The global optimum was always found in a relatively low number of iteration steps.
Now the question can be raised what exactly is a low number of iteration steps. To answer the question some convergence criterion is needed. Therefore in [5] some more enhancements to the particle swarm algorithm are presented. These include the introduction of a convergence criterion for the number of iteration steps.
A convergence criterion is needed to avoid any unnecessary function evaluations after the optimal solution is found. This convergence criterion should ideally not have any problem specific parameters. The convergence criterion that is introduced is very basic. It states that the maximum change in the objective function is recorded for a specified number of consecutive iteration steps. If for that number of iteration steps the maximum change in the objective function is smaller than a given threshold value, then convergence is assumed.

Furthermore the remark can be made that for solving continuous problems one could also make use of a gradient-based algorithm. These algorithms have proven to be very efficient. In comparison the particle swarm optimization algorithm generally requires many more function evaluations but it also has some advantages. For example the PSO algorithm requires only simple calculations which makes it easy to program, but most of all it does not require continuous differentiability in the problem to find a solution. This is an advantage over gradient-based algorithm that do need continuous differentiability in the problem. Another big advantage of the PSO algorithm over gradient-based algorithms is the ability to find the global optimum of a problem. The gradient-based algorithms are much more likely to get stuck in local optima. How to deal with constrained and discontinuous problems is also described in [5].
The PSO algorithm described so far is only defined for unconstrained problems. Since there are many problems where constraints play a role, the capability of dealing with constrained problems is added to the algorithm, [5]. This is done by making use of a penalty function which looks like:

$$\tilde{f}(x) = f(x) + \alpha \sum_{i=1}^{m} \max[0, g_i(x)]^2. \tag{2.5}$$

Here $\tilde{f}(x)$ is the new objective function, $f(x)$ is the original objective function, α is a large penalty parameter, m is the number of constraints and $g_i(x)$ is the set of all constraints.

If a particle violates a constraint the function value of that particular g is larger than zero, causing the objective function to increase by the penalty parameter. So for that particular particle that violates a constraint, the objective function can no longer be an optimal solution.
Instead of drifting further away from the optimal solution it is preferable that the violated particle moves in a direction that would reduce the objective function and that it moves back to the feasible region of the search space. This can be achieved by resetting the velocity of the violated particle at iteration step k to zero, so that the velocity at iteration step $k+1$ becomes:

$$v_{k+1} = b_1 r_1 (p_1 - x) + b_2 r_2 (p_2 - x). \tag{2.6}$$

In this way the particle is only influenced by the best known positions. In most cases this causes the particle to move back towards a feasible region of the search space.
As said before the PSO algorithm is able to find the optimal solution for continuous problems very accurately. Although for continuous problems one could also use a gradient-based algorithm. However, the PSO algorithm can also be applied to problems with discrete and discontinuous functions and variables, problems that are not suitable to gradient-based algorithms. The way to deal with discrete functions is straightforward. The position of each particle is adapted to represent a discrete point by rounding each position coordinate to its closest discrete value. This makes sure that after each iteration the objective function is located only on discrete points.

Now the development of the PSO algorithm and some extensions to it are described. The PSO algorithm that will be used throughout this thesis is a very simple one without most of these extensions.
The algorithm will be globally oriented, all the particles in the swarm will have the information of all the other particles. So the variable p_2 will be known to all particles. Although this topology of the particles may cause the algorithm to get trapped in a local optimum, this algorithm will be easier to implement and also its computational time will be shorter than is the case for the locally oriented algorithm.
Furthermore, the parameter a for the inertia term will be constant. Although a time dependent a led to better results in the simulations carried out by Shi and Eberhardt, the way to make the parameter time dependent can be very problem specific. Therefore the choice is made to let the parameter a be fixed in order to get a good comparison between the simulations that will be carried out in this thesis. With the same reasoning the maximum number of iteration steps will also be fixed. A convergence criterion can also be problem specific and therefore the maximum number of iteration steps is fixed. In the simulations it will be recorded how many iteration steps will be needed to find the global optimum.
Finally, the optimization problems that will be used in this thesis will be continuous, unconstrained problems. Therefore it is not necessary to include a penalty function into the algorithm. The goal of this thesis is to see how the tuning parameters can be chosen wisely so that the PSO algorithm will work efficiently. For this purpose it is sufficient to use continuous, unconstrained optimization problems.

Chapter 3

Particle swarm algorithm analysis

3.1 Convergence analysis and parameter selection

Up till now the particle swarm optimization algorithm has been described by its evolutionary steps. The algorithm has been built step by step and extensions of the algorithm have been discussed.
The algorithm has a number of tuning parameters that influence its performance as a trade-off between exploration and exploitation. The exploration ability of the algorithm ensures that all regions of the search space are tested in order to locate a good candidate optimum. While the exploitation ability of the algorithm ensures that the search space around the candidate solution is thoroughly tested in order to accurately locate the optimum. So in order to make the performance of the algorithm optimal a good understanding is needed in how to choose the parameters. For example the user can favor a thorough search in the entire search space for a robust location of the optimum at the cost of a large number of evaluations, or favor a quick convergence with the risk of not finding the global optimum. The role of the tuning parameters will be described and some guidelines for the parameter selection are formulated, [6].
The PSO algorithm used here looks as follows:

$$v_{k+1} = av_k + b_1 r_1(p_1 - x_k) + b_2 r_2(p_2 - x_k), \tag{3.1}$$
$$x_{k+1} = cx_k + dv_{k+1}. \tag{3.2}$$

At iteration step k, the velocity, v_k, of the considered particle is updated by its current velocity weighted by an inertia weight, a, and updated by the best positions found so far, where p_1 stands for the particle's own best position and p_2 stands for the globally best position of the entire swarm. The influence of the attraction to the best positions is determined by the coefficients b_1 and b_2. The position of the particle, x_k, is updated by its current position and the updated velocity, v_{k+1}, weighted respectively by coefficients c and d. Furthermore there are random numbers, r_1, r_2, which ensure that a good exploration of the search space is possible. Usually they are uniformly chosen in the range $[0, 1]$.

To get a clear insight in the way the PSO algorithm works it is best that for now the randomness is stripped from the algorithm. The random numbers are set to their expected value, so $r_1 = r_2 = \frac{1}{2}$.
For the purpose of analysing the algorithm, it is rewritten by setting:

$$b = \frac{b_1 + b_2}{2}, \tag{3.3}$$

$$p = \frac{b_1}{b_1 + b_2} p_1 + \frac{b_2}{b_1 + b_2} p_2. \tag{3.4}$$

The new attraction parameter b is thus the average of the parameters b_1 and b_2, and the new best point p is given by the weighted average of p_1 and p_2. The PSO algorithm now becomes:

$$v_{k+1} = av_k + b(p - x_k), \tag{3.5}$$

$$x_{k+1} = cx_k + dv_{k+1}. \tag{3.6}$$

This algorithm consists of four tuning parameters a, b, c, d. In [6] it is described that only two are useful, the other two can, without loss of generality, be fixed.
To see this the eigenvalues of the system are analysed. For this purpose, equation (3.5) is substituted into equation (3.6) and the set of equations is written in matrix form as follows:

$$y_{k+1} = Ay_k + Bp, \qquad \text{where,}$$
$$y_k = \begin{bmatrix} v_k \\ x_k \end{bmatrix}, \quad A = \begin{bmatrix} a & -b \\ ad & c - bd \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b \\ bd \end{bmatrix}. \tag{3.7}$$

The eigenvalues of the matrix A are the solutions of the equation:

$$\det(A - \lambda I) = \begin{vmatrix} a - \lambda & -b \\ ad & c - bd - \lambda \end{vmatrix} \tag{3.8}$$

$$= \lambda^2 + (bd - a - c)\lambda + ac = 0, \tag{3.9}$$

so the eigenvalues are

$$\lambda_{1,2} = \frac{-(bd - a - c) \pm \sqrt{(bd - a - c)^2 - 4ac}}{2}. \tag{3.10}$$

The equation for the eigenvalues does not use the parameters b and d individually, they only appear in the product bd. So without loss of generality one of the parameters can be fixed. For convenience, $d = 1$ is chosen.
With the choice $d = 1$ and a suitably chosen b the PSO algorithm will produce the same sequence of particle positions as before. Only the v_k will be different, but this has no impact on the optimization algorithm since the objective function only depends on the particle positions.

Furthermore, the eigenvalues are symmetric in the parameters a and c. To see that one of these can be eliminated the requirement is needed that the particles converge to the optimal position:

$$\lim_{k\to\infty} x_k = p. \tag{3.11}$$

For this purpose, the second equation of the system (3.7) is rewritten:

$$\begin{aligned} x_{k+1} &= cx_k + adv_k + bd(p - x_k), && (3.12)\\ &= cx_k + ax_k - acx_{k-1} + bdp - bdx_k, \quad \text{since } dv_k = x_k - cx_{k-1} && (3.13)\\ &= (c + a - bd)x_k - acx_{k-1} + bdp. && (3.14) \end{aligned}$$

The requirement (3.11) is substituted into equation (3.14) and for $k \to \infty$ this leads to:

$$\begin{aligned} x_{k+1} - (c + a - bd)x_k + acx_{k-1} &= bdp, && (3.15)\\ p - (c + a - bd)p + acp &= bdp, && (3.16)\\ (1 - c - a + ac)p &= 0, && (3.17)\\ (1 - a)(1 - c)p &= 0. && (3.18) \end{aligned}$$

Since p is essentially non-zero, the necessary condition is:

$$(1 - a)(1 - c) = 0. \tag{3.19}$$

The eigenvalues are symmetric in a and c, so the cases $a = 1$ or $c = 1$ are equivalent as far as the particle positions are considered. For convenience, the case $c = 1$ is chosen.
The choices $c = d = 1$ ensure that the velocity becomes a true velocity, in the sense that the velocity is the difference between two successive particle positions over a unit timestep, from equation (3.6) it follows:

$$v_{k+1} = x_{k+1} - x_k. \tag{3.20}$$

The parameter selection described above leads to the following PSO algorithm:

$$\begin{aligned} v_{k+1} &= av_k + b(p - x_k), && (3.21)\\ x_{k+1} &= x_k + v_{k+1}, && (3.22) \end{aligned}$$

yielding the following simple matrix form:

$$\begin{aligned} &y_{k+1} = Ay_k + Bp, \qquad \text{where,}\\ &y_k = \begin{bmatrix} v_k \\ x_k \end{bmatrix}, \quad A = \begin{bmatrix} a & -b \\ a & 1 - b \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b \\ b \end{bmatrix}. \end{aligned} \tag{3.23}$$

Important for the PSO algorithm to work is that the particles will converge to the global optimum. This global optimum should be an equilibrium point of the system. For any equilibrium point the condition $y_{k+1}^{\text{eq}} = y_k^{\text{eq}} = y^{\text{eq}}$ has to hold, i.e.

$$\begin{aligned} y^{\text{eq}} &= Ay^{\text{eq}} + Bp, && (3.24)\\ (I-A)y^{\text{eq}} &= Bp, && (3.25)\\ \begin{bmatrix} 1-a & b \\ -a & b \end{bmatrix} y^{\text{eq}} &= \begin{bmatrix} b \\ b \end{bmatrix} p. && (3.26) \end{aligned}$$

From this it is easy to see that

$$y^{eq} = \begin{bmatrix} 0 \\ p \end{bmatrix}, \tag{3.27}$$

so $v^{eq} = 0$ and $x^{eq} = p$. This is exactly as expected, since at equilibrium the particle should have zero velocity and its position should be p, the best solution found so far.
In order for the system to have an equilibrium the matrix $I - A$ should not be inconsistent, so the eigenvalues of the matrix should not be equal to zero. The eigenvalues of the matrix $I - A$ are the solutions of the equation:

$$\begin{aligned} \det(I - A - \mu I) &= \begin{vmatrix} 1-a-\mu & b \\ -a & b-\mu \end{vmatrix} && (3.28)\\ &= \mu^2 + (a-b-1)\mu + b = 0. && (3.29) \end{aligned}$$

So the eigenvalues are

$$\mu_{1,2} = \frac{-(a-b-1) \pm \sqrt{(a-b-1)^2 - 4b}}{2}. \tag{3.30}$$

To look for $\mu_1 = 0$ and/or $\mu_2 = 0$, the case $\mu_{1,2} \in \mathbb{R}$ is to be dealt with only, i.e. $(a-b-1)^2 - 4b \geq 0$. Then for $\mu_{1,2} = 0$ it holds

$$\begin{aligned} (a-b-1)^2 &= (a-b-1)^2 - 4b, && (3.31)\\ b &= 0. && (3.32) \end{aligned}$$

So, an equilibrium point exists if $b \neq 0$.
The initial particles are not at equilibrium. As said before it is important for the PSO algorithm to determine whether the particles will convergence to an equilibrium and if so how the particles will move through the search space. It is desirable that the particle swarm will cover most of the search space before setting at the globally best position. From the theory of linear systems it can be derived that in order to say something about the convergence behavior of the particles, the eigenvalues of the matrix A play an important role. The eigenvalues of the matrix A are

$$\lambda_{1,2} = \frac{1 + a - b \pm \sqrt{(b-a-1)^2 - 4a}}{2}. \tag{3.33}$$

The necessary and sufficient condition for the PSO algorithm to converge is that both eigenvalues have magnitude less than 1. When this condition is fulfilled the particles will eventually settle at the equilibrium and the PSO algorithm will converge. From the condition a set of inequalities will be derived for which the PSO algorithm will converge. The cases $\lambda_{1,2} \in \mathbb{R}$ and $\lambda_{1,2} \in \mathbb{C}$ will be dealt with separately:

- $\lambda_{1,2} \in \mathbb{R} : (b-a-1)^2 - 4a \geq 0$.
  Convergence requirement:

$$\begin{aligned} -1 < \quad & \frac{1+a-b\pm\sqrt{(b-a-1)^2-4a}}{2} \quad < 1, && (3.34)\\ -3-a+b < \quad & \pm\sqrt{(b-a-1)^2-4a} \quad < 1-a+b, && (3.35)\\ (-3-a+b)^2 \overset{(i)}{>} \quad & (b-a-1)^2-4a \quad \overset{(ii)}{<} (1-a+b)^2. && (3.36) \end{aligned}$$

  From (3.36)(i) it is derived that:

$$\begin{aligned} 9-6(b-a)+(b-a)^2 &> (b-a)^2-2(b-a)+1-4a, && (3.37)\\ -4(b-a) &> -8-4a, && (3.38)\\ b-a &< 2+a, && (3.39)\\ 0 &< 2+2a-b. && (3.40) \end{aligned}$$

  From (3.36)(ii) it is derived that:

$$\begin{aligned} (b-a)^2-2(b-a)+1-4a &< 1+2(b-a)+(b-a)^2, && (3.41)\\ -4(b-a) &< 4a, && (3.42)\\ b &> 0. && (3.43) \end{aligned}$$

- $\lambda_{1,2} \in \mathbb{C} : (b-a-1)^2 - 4a < 0$
  Convergence requirement:

$$\begin{aligned} \left|\frac{1+a-b\pm i\sqrt{-(b-a-1)^2+4a}}{2}\right| &< 1, && (3.44)\\ (1+a-b)^2-(b-a-1)^2+4a &< 4, && (3.45)\\ a &< 1. && (3.46) \end{aligned}$$

The set of inequalities for which the PSO will converge is given by:

$$a < 1, \qquad b > 0, \qquad 2a-b+2 > 0. \tag{3.47}$$

So for any initial position and velocity of a particle in the search space, the particle will converge to the global optimum if and only if the tuning parameters a and b are chosen such that the inequalities hold.

To investigate if the initial particles will cover the entire search space before settling in the global optimum the movement of the particles is analysed. Again from the theory of linear systems the behavior of the particles is derived.
From the magnitude of the eigenvalues it can be derived how fast the particle will converge. For instance when the eigenvalues have values close to 1 the convergence of the particle will be slow, while if an eigenvalue has a value of 0 the particle will converge fast. When one of the eigenvalues has a value that is equal to 1 the particle will not converge to the global optimum. To further investigate the convergence behavior of the particles the following properties are considered.
The particles will oscillate harmonically around the global optimum if both eigenvalues of the matrix A are complex. The eigenvalues are complex valued if the following inequality holds:

$$(b-a)^2 - 2(b-a) + 1 - 4a < 0. \tag{3.48}$$

It is also possible that the particles will have a zigzagging behavior towards the global optimum. A zigzagging behavior means that the direction in which the particles travel will change for each iteration step. This is the case when at least one eigenvalue of the matrix A has a negative real part, so if $\mathrm{Re}(\lambda_{1,2}) \in \mathbb{R}^-$. Again the cases $\lambda_{1,2} \in \mathbb{R}$ and $\lambda_{1,2} \in \mathbb{C}$ are dealt with separately:

- If $\lambda_{1,2} \in \mathbb{R}$, then it has to hold that:

$$1 + a - b \pm \sqrt{(b-a-1)^2 - 4a} < 0, \tag{3.49}$$
$$(1 + a - b)^2 < (b-a-1)^2 - 4a, \tag{3.50}$$
$$a < 0. \tag{3.51}$$

- If $\lambda_{1,2} \in \mathbb{C}$, then it has to hold that:

$$1 + a - b < 0. \tag{3.52}$$

This leads to the inequalities:

$$a < 0, \qquad a - b + 1 < 0. \tag{3.53}$$

Above some guidelines for choosing the free parameters a and b have been derived. These guidelines will now be applied to a test problem.

3.2 Parameter selection for one-dimensional test problem

Next several cases are described for different choices of the tuning parameters a and b. In [6] examples of the behavior of particle motions are given. For various choices of the tuning parameters a and b, the movement of a particle is shown. The simulations show the behavior of a single particle in a PSO algorithm for which the attraction point p is fixed. The PSO algorithm was performed with the parameters:

$$x_0 = 2, \qquad v_0 = -0.1, \qquad p = 0, \qquad m = 50 \text{ iteration steps.} \tag{3.54}$$

This corresponds with the situation of a single particle with initial position in $x = 2$ and initial velocity $v = -0.1$. This single particle is attracted by the global optimum which is fixed on $x = 0$. The particle swarm algorithm will move the particle throughout the search space and, if convergent, will settle it in the global optimum. The movement of the particle will be shown in a plot for different choices of the parameters.
First some cases are considered for which the chosen parameters are not optimal. For these cases the particle does not converge towards the global optimum or its convergence behavior is not desirable otherwise.
For instance when the parameters are chosen as $a = 0.9$ and $b = 0$. The parameter b is now chosen such that the system (3.26) has no unique solution, since $\mu_{1,2} = 0$. Therefore the algorithm will not converge to the global optimum. This parameter choice corresponds with the case, as described before in [4], where the new velocity of a particle is only determined by its current velocity and not influenced by the best known positions. The particle will move in the direction of its initial velocity and will not converge to p. This is because the velocity of the particle eventually goes to 0 due to the inertia weight a. The result of this can be seen in figure 3.1(a).
The result corresponds with what can be expected if the eigenvalues $\lambda_{1,2}$ for this case are analysed. In table 3.1 the eigenvalues are given for several choices of the parameters a and b. So for this case, where $a = 0.9$ and $b = 0$, the eigenvalues $\lambda_{1,2}$ are real, so no harmonic oscillations will occur. The eigenvalues have no negative real part, so a zigzagging behavior of the particle will not occur. Since $b \not> 0$, from the eigenvalue analysis, it is also expected that the particle will not converge to the global optimum. For this case one eigenvalue is equal to 1, so the particle will not converge towards the global optimum.
The other case described in [4] is the case in which the new velocity of a particle is only determined by the current and best known position and not by its current velocity. The corresponding parameters are chosen as $a = 0$ and $b = 0.5$. Again the eigenvalues $\lambda_{1,2}$ are real and positive, so the particle will not oscillate or zigzag, see table 3.1. For this choice of parameters all requirements for the particle to converge are met, so the particle will converge. Since one of the eigenvalues $\lambda_{1,2}$ has magnitude 0 the convergence will be fast. The result is that the particle will directly converge to the global optimum without further exploring the search space, see figure 3.1(b). This behavior may not be desirable because the particle will not reach the global optimum if it lies outside the initial search space.

An even more extreme choice for the parameters is $a = 0$ and $b = 1$. For this case both eigenvalues $\lambda_{1,2}$ are equal to zero and the particle will directly move towards the global optimum. After one iteration step the particle has settled in the global optimum, which also can be concluded if one writes out the particle swarm algorithm for this choice of parameters. The velocity in the first iteration step will be equal to $v_1 = p - x_0$ and the next particle position will be $x_1 = x_0 + v_1 = p$. So the next velocity is $v_2 = p - p = 0$ and thus the particle will be fixed in the attraction point p for all the iteration steps, see figure 3.1(c).
In the next three cases the inequalities for the algorithm to converge are violated. In figure 3.1(d) it is shown what happens when the parameter $b < 0$. The particle is now repelled from the global optimum instead of attracted and the particle will diverge from the global optimum. This diverging behavior can be explained since one of the eigenvalues is larger than 1.
In another case the parameters $a = 1,\ b = 0.5$ are chosen. For this case the eigenvalues $\lambda_{1,2}$ are complex and of magnitude 1, so the particle will oscillate harmonically without damping or amplification, see figure 3.1(e). The particle will not settle in a certain point; it will keep its momentum, causing it to endlessly oscillate around the global optimum.
Lastly the parameters are chosen as $a = 0.5$ and $b = 3$. In table 3.1 it can be seen that the eigenvalues $\lambda_{1,2}$ are real and negative. So the particle will not oscillate harmonically, but will have a zigzagging behavior instead. Furthermore the eigenvalue conditions for the particle to converge are violated for this choice of parameters. The magnitude of one eigenvalue in this case is equal to 1. So, again the particle will not settle in the global optimum. The particle will keep zigzagging around the global optimum, see figure 3.1(f).

(a, b)	(λ_1, λ_2)
$(0.9, 0)$	$(1, 0.9)$
$(0, 0.5)$	$(\frac{1}{2}, 0)$
$(0, 1)$	$(0, 0)$
$(0.5, -0.01)$	$(1.0196\ldots, 0.4904\ldots)$
$(1, 0.5)$	$(\frac{3}{4} + \frac{\sqrt{7}}{4}i, \frac{3}{4} - \frac{\sqrt{7}}{4}i) \Rightarrow \lvert\lambda_{1,2}\rvert = 1$
$(0.5, 3)$	$(-1, -\frac{1}{2})$

Table 3.1: Eigenvalues λ_1 and λ_2 for several choices of parameters a and b.

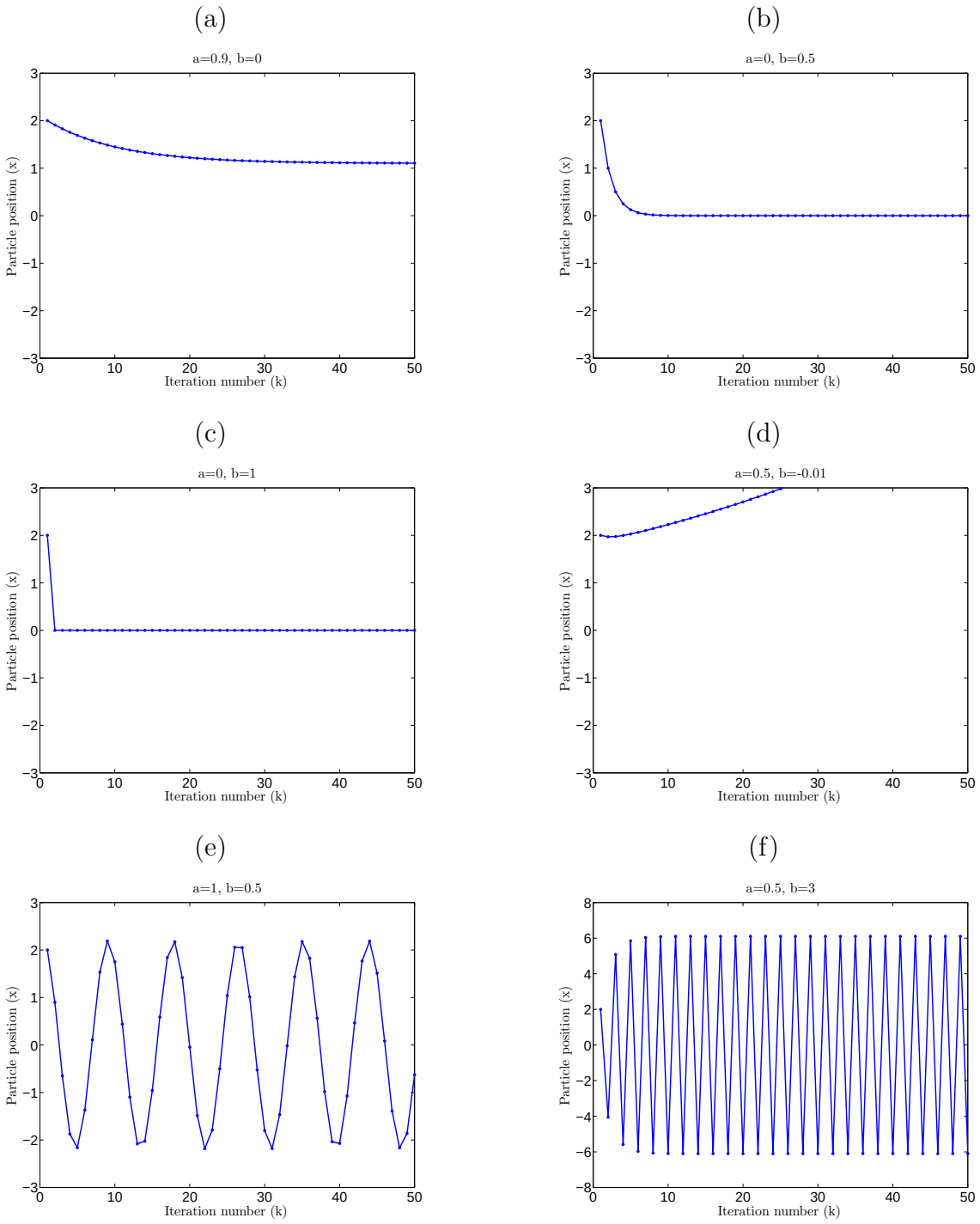


Figure 3.1: Examples of particle movement for several choices of the parameters a and b.

Next some cases are described for which the selected parameters satisfy the inequalities to guarantee convergence. So the particle will converge to the global optimum for all cases.
In figure 3.2(a) slowly convergent harmonic oscillations are shown and in figure 3.2(b) it is shown that the oscillations decay quicker and the particle converges faster. In the first case the parameters are chosen as $a = 0.9$ and $b = 0.1$ and for the other case the parameters are chosen as $a = 0.7$ and $b = 0.3$. For both cases the eigenvalues $\lambda_{1,2}$ are complex and have a positive real part, see table 3.2. So indeed harmonic oscillations are expected. The magnitude of the eigenvalues in the first case is larger than the magnitude of the eigenvalues in the second case. This indicates that the convergence of the particle will be faster for the second set of parameters. In the first case the particle samples the search space relatively well, the exploration and exploitation are balanced in that case. While in the other case the exploitation of the optimum is favored compared to the exploration of the search space. So both sets of parameters have their own advantages and it will depend on the considered problem which one is favorable.
For the next choice of parameters, $a = 0.9$ and $b = 3$, the eigenvalues $\lambda_{1,2}$ are complex and have a negative real part, see table 3.2. The particle will harmonically oscillate and zigzag while it converges towards the global optimum. In figure 3.2(c) these harmonic oscillations combined with zigzagging are shown. Now the search space is thoroughly explored at the cost of slow convergence. The magnitude of the eigenvalues for this case is close to 1. So this case is useful for a full exploration of the search space to get a good idea where the global optimum might be.
If the parameters are chosen as $a = 0.1$ and $b = 0.1$ the eigenvalues $\lambda_{1,2}$ are real and positive. So the particle will not oscillate or zigzag. In figure 3.2(d) this non-oscillatory convergence is given. One eigenvalue for this case is close to 1 while the other eigenvalue is close to 0, so the convergence of the particle in this case will be moderate. As said before, in general, this non-oscillatory behavior is not desirable for optimization since it only samples half of the search space. However, in some special cases it might be very useful, for example when it is known from the problem that the solution can not have negative values.
Finally the parameters are chosen as $a = 0.1$ and $b = 2.1$ and $a = -0.7$ and $b = 0.5$. For both cases the eigenvalues are real, see table 3.2. For the first case both eigenvalues are negative resulting in a symmetric zigzagging behavior of the particle. In the other case the eigenvalues have opposite signs, causing the particle to have an asymmetric zigzagging behavior. In figure 3.2(e) symmetric zigzagging convergence is shown and in figure 3.2(f) asymmetric zigzagging is shown.

(a, b)	(λ_1, λ_2)
$(0.9, 0.1)$	$(0.9 + 0.3i, 0.9 - 0.3i) \Rightarrow \lvert\lambda_{1,2}\rvert^2 = 0.9$
$(0.7, 0.3)$	$(0.7 + 0.4583\ldots i, 0.7 - 0.4583\ldots i) \Rightarrow \lvert\lambda_{1,2}\rvert^2 = 0.7$
$(0.9, 3.0)$	$(-1.1 + 0.7730\ldots i, -1.1 - 0.7730\ldots i) \Rightarrow \lvert\lambda_{1,2}\rvert^2 = 0.9$
$(0.1, 0.1)$	$(0.8873\ldots, 0.1127\ldots)$
$(0.1, 2.1)$	$(-0.1127\ldots, -0.8873\ldots)$
$(-0.7, 0.5)$	$(0.7426\ldots, -0.9426\ldots)$

Table 3.2: Eigenvalues λ_1 and λ_2 for several choices of parameters a and b.

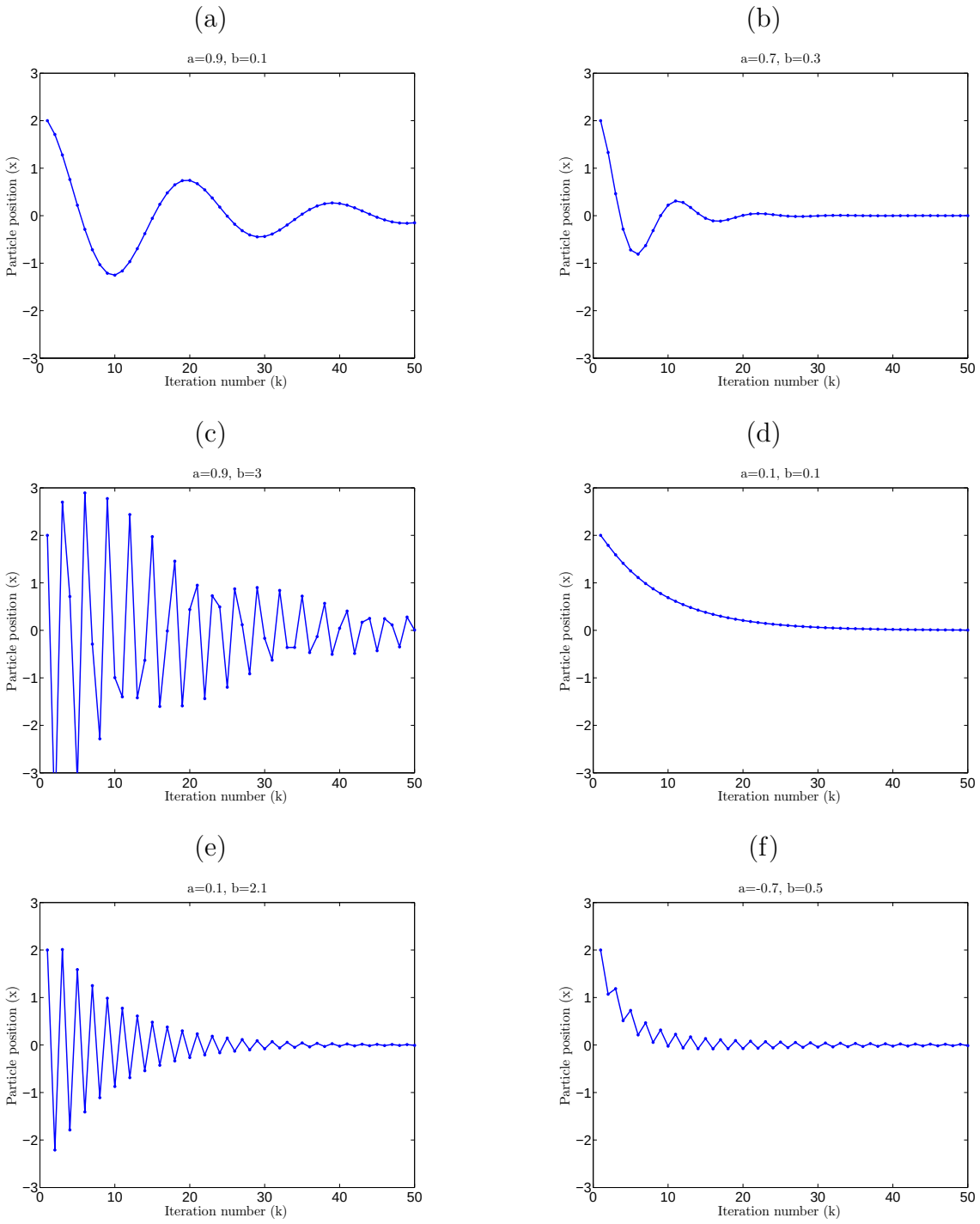


Figure 3.2: Examples of particle movement for several choices of the parameters a and b.

The foregoing convergence and existence analyses have been done for the PSO algorithm without randomness. This leads to some selection criteria for choosing the tuning parameters in such a way that they fit to the problem that has to be optimized. These considerations will also hold if randomness is included into the PSO algorithm. In general the random numbers will enhance the zigzagging behavior of the particles, but at the cost of a slower convergence. So randomness improves the exploration of the search space and prevents the particles setting in a non optimal point. When random numbers are included, the equivalent attraction point p changes from (3.4) to:

$$p = \frac{b_1 r_1}{b_1 r_1 + b_2 r_2} p_1 + \frac{b_2 r_2}{b_1 r_1 + b_2 r_2} p_2. \tag{3.55}$$

From this it can be seen that even when for a number of iteration steps the best known positions do not change, so p_1 and p_2 remain constant, the attraction point may still change due to the influence of the random numbers. This explains why the PSO algorithm is less vulnerable for setting in a non-optimal solution. In the long run it is expected that $p_1 = p_2$ holds, then the attraction point p will not change and all particles will settle in that point. In the next section the PSO algorithm is applied to another test problem and randomness is introduced into the algorithm.

3.3 Parameter selection on Schaffer's F6 function

To investigate the role of randomness another optimization problem is introduced. The goal of this optimization problem is to find the global maximum of a given function. The function that will be used is known as Schaffer's F6 function and is given by

$$f(x, y) = \frac{1}{2} + \frac{\sin^2(\sqrt{x^2 + y^2}) - \frac{1}{2}}{(1 + 0.001(x^2 + y^2))^2}. \tag{3.56}$$

To simplify the optimization problem a one-dimensional cross section of this function is selected. For $y = 0$ the function becomes

$$f(x) = \frac{1}{2} + \frac{\sin^2 \sqrt{x^2} - \frac{1}{2}}{(1 + 0.001x^2)^2}. \tag{3.57}$$

The function is symmetric, so for optimization purposes only the positive x-axis is considered. In figure 3.3 the function is given, where x is in the domain $(0, 100)$.
From the figure it is clear that the function has many local maxima and one global maximum. Therefore the function is ideal to test the PSO algorithm. The global optimum is located at $x = 1.5692\ldots$ and the corresponding function value is equal to $(f(1.5692\ldots)) = 0.9975\ldots$.

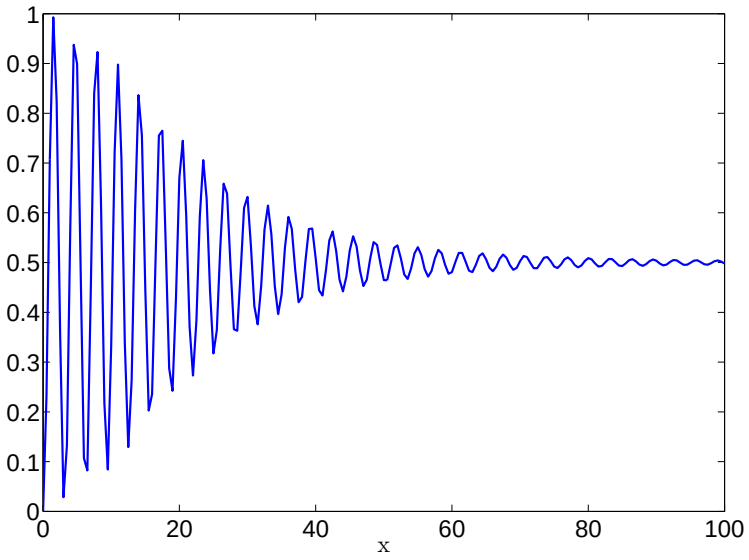


Figure 3.3: Schaffer's F6 function for y=0

The particle swarm optimization algorithm that is used for this optimization problem is the following:

$$v_{k+1} = av_k + b_1 r_1(p_1 - x_k) + b_2 r_2(p_2 - x_k), \tag{3.58}$$

$$x_{k+1} = x_k + v_{k+1}. \tag{3.59}$$

Here a, b_1 and b_2 are the tuning parameters and r_1 and r_2 are random numbers. These random numbers will be different for each new iteration step of the algorithm. They will be uniformly generated from the range $(0, 1)$, using the *Matlab* command *rand()*.
The PSO algorithm needs an initial set of particles. The number of particles in the swarm can influence the performance of the PSO algorithm. For larger swarms the PSO algorithm is in general more likely to find the global optimum. But this may come with an increase in the time needed to run the PSO algorithm. For each particle the algorithm needs to run through all the iteration steps which may cost extra computational time.
The distribution of the particles will influence the performance of the PSO algorithm. If the particles in the initial set are positioned close to each other the algorithm might not have an optimal performance. For instance if all initial particles are close to a local optimum it is likely that the PSO algorithm will settle in that optimum. Ideally the initial particles are distributed uniformly over the domain. In the next chapter the influence of the initial swarm on the PSO algorithm performance will be discussed in more detail. For now the number of initial particles in the swarm will be fixed and the assumption is made that the particles are distributed well over the domain.

To find the global optimum of Schaffer's F6 function an initial swarm of five particles is selected. For a swarm of five particles the algorithm seems to find the global optimum in most of the cases for all the different choices of the tuning parameters, while the computational time of the algorithm is moderate.
The five initial particles for the PSO algorithm on Schaffer's F6 function are five x-values with their corresponding function value. These initial function values will be stored in the variable p_1. This variable stores the particles' own best position found so far. In other words the variable stores the maximum values that each particle has attained so far. The variable p_2 stores the global maximum found by any particle so far.
The initial points are located at $x = (0, 20, 40, 60, 80)$, this ensures a good distribution over the search space, see figure 3.4.

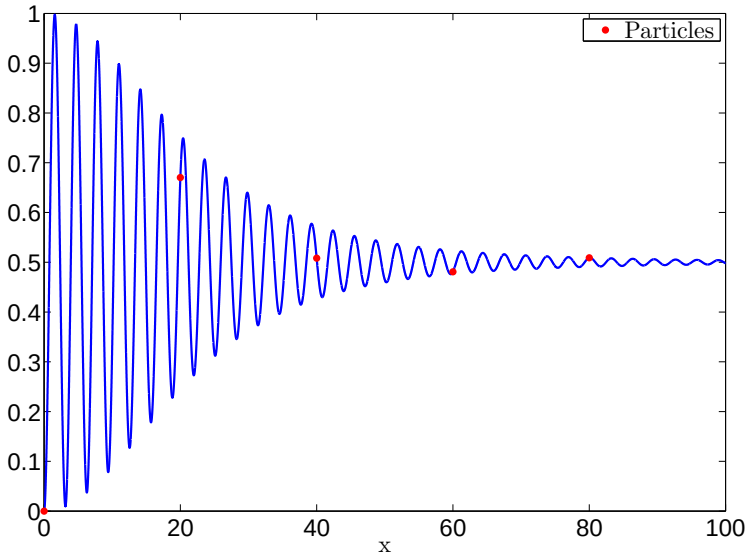


Figure 3.4: Initial positions of the particle in the swarm

Before the performance of the PSO algorithm is analysed for different choices of the tuning parameters a, b_1 and b_2, first some examples will be given to see how the particle swarm optimization algorithm works.
For each iteration step of the PSO algorithm the position of each particle in the swarm is determined. The function value for this new position of a particle is compared with its best position found so far. If the new position has a higher function value than the value stored in p_1, then the variable p_1 is updated. Also the variable p_2 will be updated if the maximum function value of p_1 changes. In figure 3.5 the particle positions on Schaffer's F6 function are shown after several iteration steps. In the figure the position of each particle's own maximum attained function value can be seen and also the global optimum found so far is plotted.

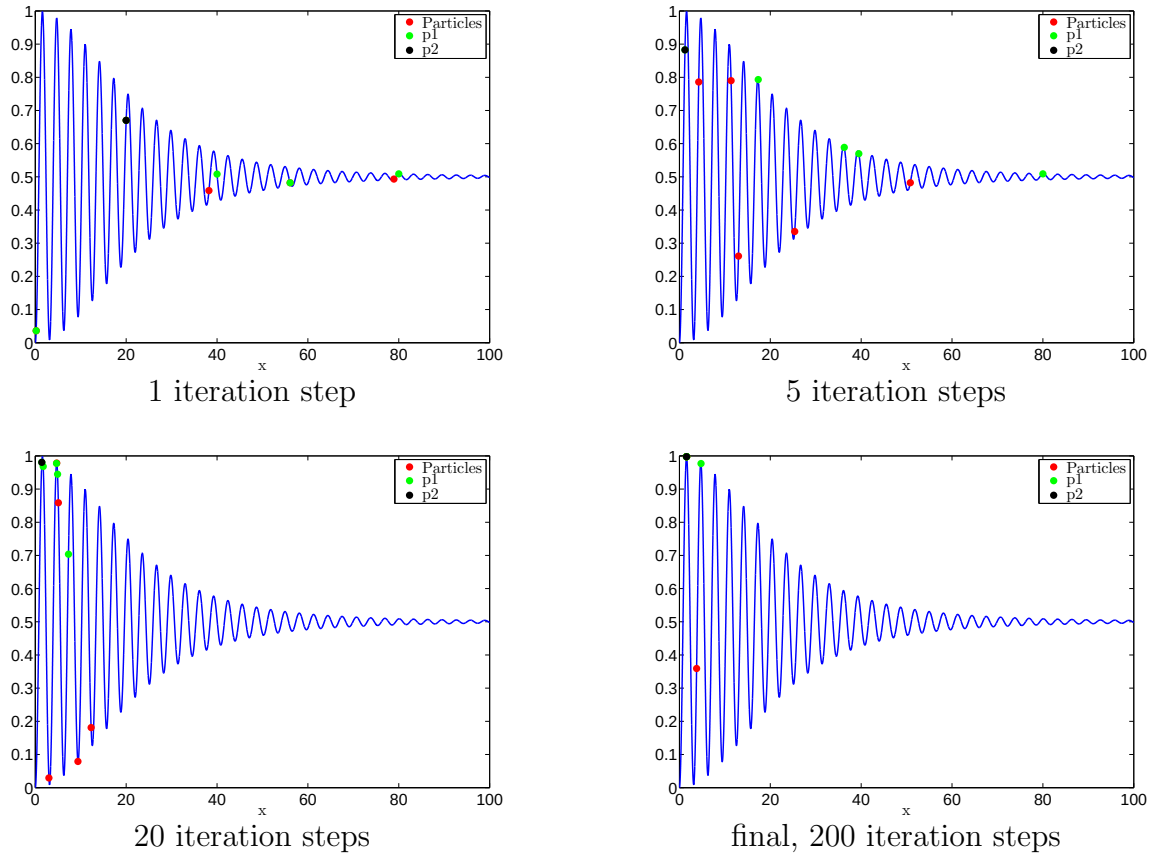


Figure 3.5: PSO algorithm after several iteration steps

For this example an arbitrary choice is made for the tuning parameters a, b_1 and b_2. The figure shows that the global optimum converges towards the global maximum of Schaffer's F6 function.
It may occur that the particle's current position is the particle's best position found so far, so these points will coincide in the figure. This is the case for example after 1 iteration step of the considered PSO algorithm, see figure 3.5. Here three particles have reached a position with a higher function value than in their initial position.
For this choice of the tuning parameters the global maximum is found within the number of iteration steps that is allowed, as can be seen in the last figure of 3.5. In this last figure it can also be seen that not all particles did settle in the global optimum. There is one particle that has not reached the global optimum. This particle is located between the position of the global optimum and the particle's own best position found so far. The fact that not all particles will reach the global optimum within the number of iteration steps allowed is not a problem. Since the goal of the PSO algorithm is to find the global maximum of Schaffer's F6 function it would be sufficient if only one particle ever reaches the global optimum. Although for a good exploitation around the global optimum it is preferable

that more particles will settle in the global optimum. This ensures a more accurate search around the global optimum.

Next the performance of the PSO algorithm is analysed for several different choices of the tuning parameters. The maximum number of iteration steps for the PSO algorithm to find the global maximum of Schaffer's F6 function is set to 200. For all choices of the tuning parameters the PSO algorithm is simulated 50 times. This is done to get a mean performance of the algorithm because of the random numbers. To keep track of the performance of the algorithm the number of iteration steps needed to find the global optimum is recorded. In appendix A.1 the number of iteration steps needed is shown in a table for several choices of the tuning parameters for all 50 simulation runs.
From this table two variables are extracted to analyse the performance of the PSO algorithm. First the number of times the global optimum is not found within the allowed number of iteration steps is recorded. Further, if the global optimum is found the mean number of iteration steps needed to find the global optimum is recorded. These results are given in table 3.3.

For the first six cases the parameters are chosen equal to the parameters that were used in the test problem in the previous section. So the values for the parameters b_1 and b_2 are kept equal in these cases. There is one exception, in case (c) the parameters are chosen as $b_1 = b_2 = 2$. This choice is made since the PSO algorithm often breaks down if the parameters are chosen equal to 3. The behavior also occurs when the parameters are chosen equal to 2 but less often, see the table in appendix A.1. The breakdown of the PSO algorithm can be explained by the introduction of the random numbers into the PSO algorithm. If for the first few iteration steps the selected random numbers are large, close to 1, it might happen that some particles will have a strong exploring behavior and move outside the domain of the function. Since there is no constraint in this PSO algorithm to tell the particles to move back to the feasible region of the search space the algorithm will break down.
In the latter two cases the tuning parameters b_1 and b_2 are chosen to be different. The first case corresponds with the situation where the particles in the swarm are more drawn to their own best position known so far. The second case corresponds with the situation where the particles are more drawn towards the global optimum.

Case	Parameters (a, b_1, b_2)	Number of failures	Mean number of iteration steps
a	$(0.9, 0.1, 0.1)$	5	113
b	$(0.7, 0.3, 0.3)$	27	34
c	$(0.9, 2, 2)$	48	132
d	$(0.1, 0.1, 0.1)$	50	–
e	$(0.1, 2, 2)$	8	83
f	$(-0.7, 0.5, 0.5)$	36	86
g	$(0.9, 0.9, 0.1)$	13	128
h	$(0.9, 0.1, 0.9)$	5	146

Table 3.3: Properties for several choices of parameters a, b_1 and b_2.

In the following, comparisons are made between the performance of the PSO algorithm in the previous one-dimensional test problem and the current optimization problem.
In table 3.3 it is shown that for case (a) the PSO algorithm is very likely to find the maximum value of Schaffer's F6 function. The mean number of iteration steps needed to find the optimum is 113. This result corresponds with the result for the one-dimensional test problem found in the previous section. Also there, the algorithm slowly converged towards the global optimum but the search space was thoroughly explored.
For case (b) the algorithm finds the global optimum in roughly half the number of simulation runs only. But if the global optimum is found, it is found within a small number of iteration steps. This result also corresponds with that of the previous test problem. There it also appeared that the convergence was faster than for case (a) but that the search space was explored less well, causing the algorithm to settle in a local optimum.
In the previous test problem it was shown that for case (c) the search space is explored very thoroughly but that the number of iteration steps needed to find the global optimum is large. This behavior can also be seen for the current optimization problem. Only two times the global optimum is found within the selected number of iteration steps. It is plausible that for case (c) the global optimum will eventually be found, but at the cost of a long computational time.
In table 3.3 it is shown that for case (d) the global optimum is never found. This was to be expected since for the previous test problem it was shown that the PSO algorithm converges very fast towards the global optimum in the algorithm, the variable p_2. However, the variable p_2 is a local optimum of Schaffer's F6 function. Therefore the global optimum of Schaffer's F6 function is not found for this selection of parameters.
For case (e) it is shown that the global optimum is found in most of the simulation runs and due to the zigzagging behavior of the particles the optimum is found within a moderate number of iteration steps. For case (f) it can be seen that the algorithm fails to find the global optimum in more than half of the number of simulation runs. However, if the global optimum is found it will be found within a moderate number of iteration steps.
Finally the PSO algorithm performance is analysed for the cases where the tuning parameters b_1 and b_2 are different. The parameter a is in these cases chosen the same as in case (a). This choice is made since the algorithm performed rather well for this case, the algorithm found the exact solution for almost every simulation run within a moderate number of iteration steps.
For both cases it is shown that the number of times the PSO algorithm fails to find the optimum is almost equal to the number of failures in case (a). However, for both cases it appears that the number of iteration steps needed to find the global optimum is higher than the number of iteration steps in case (a). This can be explained by the fact that the particles in the PSO algorithm will now be attracted more towards either their own best position known so far, the variable p_1, or the globally best position so far, the variable p_2. This causes the PSO algorithm to use more iteration steps to find the global optimum of the function, since the particles are likely to explore local optima before converging towards the global optimum.

Chapter 4

Particle swarm algorithm on brachistochrone problem

The brachistochrone problem has been an important problem for the development of mathematics in the seventeenth century. It marks the beginning of variational calculus. The brachistochrone problem was first formulated by Johann Bernoulli in an article in *Acta Eruditorum* in June 1696 . The name of the problem comes from the Greek words "brachus" meaning short and "chronos" meaning time. Bernoulli stated the problem in the following way: Given two points A and B in a vertical plane, where B is lower than A. What is the curve traced out by a moving point that starts in A and only under the influence of gravity reaches B in the shortest time?

4.1 Bernoulli's solution to brachistochrone problem

Bernoulli published the article on the brachistochrone problem to challenge some of the other great mathematicians of that time to come up with a solution. The problem was solved by Bernoulli and also by Leibniz, Newton and his brother Jacob Bernoulli. Johann Bernoulli thought he was the first that formulated the problem but Leibniz told him that already Galileo Galilei claimed he had found a solution, but this solution was incorrect. Galilei reasoned that the curve with the shortest time had to be an arc. In this section Johann Bernoulli's solution to the problem is given, [7].
In his solution Bernoulli used some known results and combined it with a new idea he came up with. One of the known results he used was a result that Galilei had found. He found out that a particle at rest falling under the influence of gravity has a velocity proportional to $\sqrt{h}$ when it has fallen over a distance h. The new idea of Bernoulli that led him to the solution of the brachistochrone problem was the following. He did not consider a particle falling but he considered a ray of light shining through a medium with variable density. According to Fermat a ray of light will always follow the shortest way through a medium. So Bernoulli reasoned that if the medium would have a density at height x corresponding with the known gravitational velocity of a particle at height x the ray of light would follow a brachistochrone curve.

In his solution Bernoulli made use of figure 4.1, which is explained as follows.

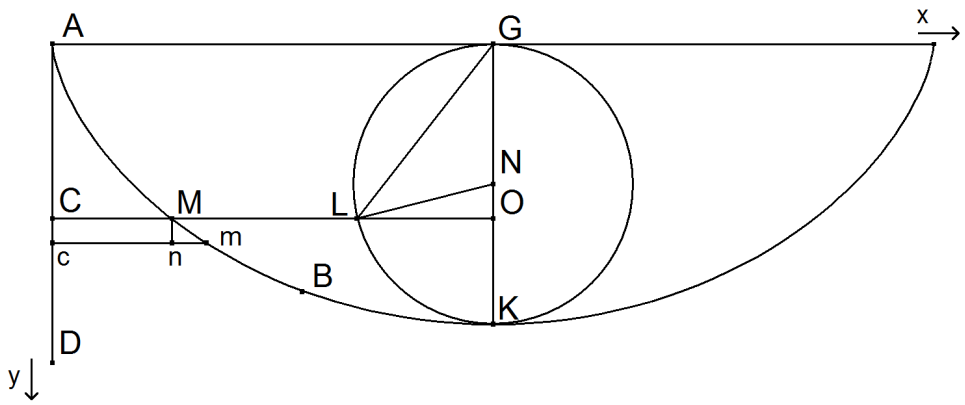


Figure 4.1: Main figure in Bernoulli's solution

- A and B are the given points in the problem

- AMB is the brachistochrone curve that has to be found

- AG is the horizontal line through A

- ACD is the vertical line through A

- $CM = x$ and $AC = y$

- CM is the horizontal line through M

- cm is the horizontal line that lies infinitesimally below CM

- n is the point on cm directly below M

- $\mathrm{d}x = nm, \mathrm{d}y = Mn, \mathrm{d}s = Mm$

- $GLKG$ is the generating circle of the cycloid AMB

- K is the point where the cycloid has a horizontal tangent

- N is the center of the circle and $GK = a$ is the diameter

- L lies on the circle and on CM

- O lies on CM and on GK

Suppose i is the angle of incidence and r is the angle of refraction at the point M. And suppose also an infinitesimal triangle above M with sides $\mathrm{d}x'$, $\mathrm{d}y'$ and $\mathrm{d}s'$. Assume the velocity above M is v_1 and the velocity below M is v_2. Then by geometry

$$\sin i = \frac{\mathrm{d}x'}{\mathrm{d}s'} \quad \text{and} \quad \sin r = \frac{\mathrm{d}x}{\mathrm{d}s}, \tag{4.1}$$

and by Snell's law

$$\frac{\sin i}{\sin r} = \frac{v_1}{v_2}. \tag{4.2}$$

So it follows that

$$\frac{1}{v_1}\frac{\mathrm{d}x'}{\mathrm{d}s'} = \frac{1}{v_2}\frac{\mathrm{d}x}{\mathrm{d}s}. \tag{4.3}$$

The variable v can be assumed constant over an infinitesimal time. Therefore the quantity $\frac{1}{v}\frac{\mathrm{d}x}{\mathrm{d}s}$ will not change in an infinitesimal time, so it is also constant for finite time. Now Bernoulli concludes that for all points M of the brachistochrone it holds that:

$$\frac{\mathrm{d}x}{\mathrm{d}s} = cv, \tag{4.4}$$

for some constant $c \in \mathbb{R}$. This gives a differential equation for the brachistochrone curve. Since

$$(\mathrm{d}s)^2 = (\mathrm{d}x)^2 + (\mathrm{d}y)^2 \qquad \text{and} \qquad v = b\sqrt{y}, \quad \text{for appropriate } b \in \mathbb{R}, \tag{4.5}$$

differential equation (4.4) can be rewritten as:

$$\frac{(\mathrm{d}y)^2}{(\mathrm{d}x)^2} = \frac{(\mathrm{d}s)^2}{(\mathrm{d}x)^2} - 1 = \frac{1}{c^2v^2} - 1 = \frac{a}{y} - 1 = \frac{a-y}{y}, \tag{4.6}$$

where $a = \frac{1}{(cb)^2}$. Here Bernoulli used the known result from Galilei. This finally results in

$$\frac{\mathrm{d}x}{\mathrm{d}y} = \sqrt{\frac{y}{a-y}}. \tag{4.7}$$

The curve that fits to this differential equation is a cycloid. A cycloid is the curve traced by a fixed point on the edge of a circle that rolls along a straight line. The next step in Bernoulli's proof is to prove that the cycloid produced by a circle with diameter a fulfills this differential equation (4.7). In figure 4.1 this cycloid is given by AMB and is produced by the circle that has diameter $GK = a$. Next the equation (4.7) is written as

$$\frac{\mathrm{d}x}{\mathrm{d}y} = \sqrt{\frac{y}{a-y}} = \frac{y}{\sqrt{ay-y^2}} = \frac{1}{2}\frac{a}{\sqrt{ay-y^2}} - \frac{1}{2}\frac{a-2y}{\sqrt{ay-y^2}}. \tag{4.8}$$

The equation is integrated to get

$$x = I_1 - I_2, \quad \text{where} \quad I_1 = \int \frac{1}{2}\frac{a}{\sqrt{ay-y^2}}\mathrm{d}y, \quad I_2 = \int \frac{1}{2}\frac{a-2y}{\sqrt{ay-y^2}}\mathrm{d}y. \tag{4.9}$$

It is easy to see that $I_2 = \sqrt{ay - y^2}$. So

$$(I_2)^2 = y(a - y) = GO \cdot OK, \qquad \text{since } AC = GO = y. \tag{4.10}$$

Now LO is the altitude of the right triangle GLK, so by the geometric mean theorem it follows that:

$$I_2 = \sqrt{GO \cdot OK} = LO. \tag{4.11}$$

To determine I_1 the triangle LON and a part of the circle around L is used, see figure 4.2.

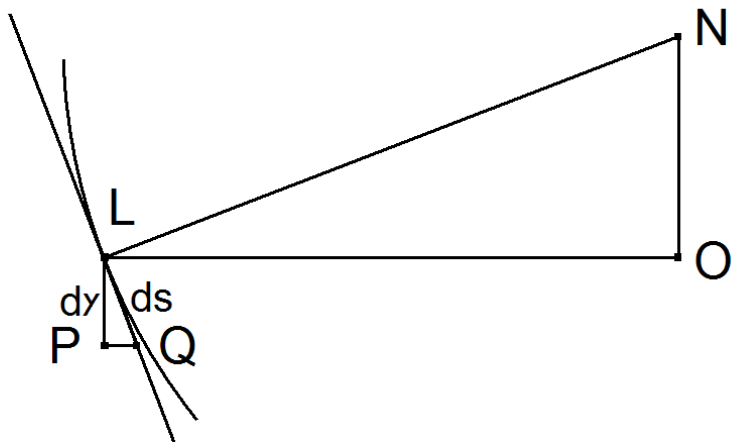


Figure 4.2: $\triangle LON$ and arc in neighbourhood of L

The sides of the triangle are

$$LN = \frac{1}{2}a, \quad NO = GO - GN = AC - GN = y - \frac{1}{2}a, \quad LO = \sqrt{ay - y^2}. \tag{4.12}$$

The infinitesimal triangle LPQ has sides $LP = \mathrm{d}y$ and $LQ = \mathrm{d}s$, which is the hypotenuse. For this infinitesimal triangle $\mathrm{d}s$ coincides with the part of the arc. Now $\triangle LPQ$ is similar to $\triangle LON$, since the tangent LQ is perpendicular to the radius LN and LP is perpendicular to LO. It follows that $\angle(PLQ)$ is equal to $\angle(NLO)$. So

$$\frac{\mathrm{d}s}{\mathrm{d}y} = \frac{LN}{LO}, \tag{4.13}$$

$$\frac{\mathrm{d}s}{\mathrm{d}y} = \frac{\frac{1}{2}a}{\sqrt{ay - y^2}}, \tag{4.14}$$

$$\mathrm{d}s = \frac{\frac{1}{2}a}{\sqrt{ay - y^2}}\mathrm{d}y. \tag{4.15}$$

Integrating this equation gives

$$\text{arc length } GL = \int \mathrm{d}s = \int \frac{\frac{1}{2}a}{\sqrt{ay - y^2}}\mathrm{d}y = I_1. \tag{4.16}$$

Substituting equation (4.11) and (4.16) into equation (4.9) gives

$$CM = \text{arc } GL - LO, \tag{4.17}$$
$$CM + LO = \text{arc } GL. \tag{4.18}$$

So finally

$$ML = AG - (CM + LO) = \text{arc } GK - \text{arc } GL = \text{arc } LK. \tag{4.19}$$

The generating circle goes through the point M. The displacement needed to get the diameter of the circle from that position to the position in GK is equal to ML. This displacement corresponds with rolling the circle over a part of the arc equal to LK. A defining property of a cycloid is that the horizontal displacement of its generating circle equals the part of the arc over which the circle rolls. This is exactly what Bernoulli derived from the differential equation (4.7). So he proved that a circle with diameter a produces a cycloid that goes through A and B.
Finally Bernoulli showed how to construct the cycloid that starts in A and goes through B. Therefore he used figure 4.3.

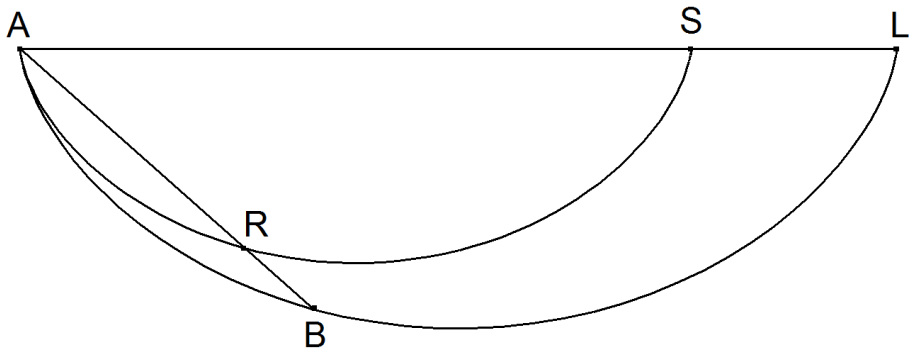


Figure 4.3: Uniqueness of cycloids

Bernoulli first draws another random cycloid ARS that starts in A and is produced by a circle with diameter d. Next he draws the line between A and B and marks the intersection of the line AB with the cycloid as the point R. Now the diameter of the circle that generates the cycloid ABL is proportional to the diameter of the circle that generates the cycloid ARS as $AB : AR$. So the circle that produces the cycloid ABL has diameter $\frac{AB}{AR}d$.

4.2 Analytical solution to brachistochrone problem

Bernoulli uses for his solution mainly geometric reasoning to proof that the solution to the brachistochrone problem is a cycloid. In this section a more modern solution is given. It uses some results known from variational calculus.
For the brachistochrone problem the energy conservation law is applied to the moving particle. So in all the points along the curve the potential energy plus the kinetic energy should be constant. Now assume that point A has height h and the moving particle has fallen to a height y, from this it is derived that

$$\begin{aligned} mgh &= \frac{1}{2}mv^2 + mg(h-y), && (4.20)\\ \frac{1}{2}mv^2 &= mgy, && (4.21)\\ v &= \sqrt{2gy}. && (4.22) \end{aligned}$$

Furthermore for the velocity of the moving particle it holds that

$$v = \frac{\mathrm{d}s}{\mathrm{d}t} = \frac{\sqrt{\mathrm{d}x^2 + \mathrm{d}y^2}}{\mathrm{d}t} = \frac{\sqrt{1 + (\frac{\mathrm{d}y}{\mathrm{d}x})^2}}{\mathrm{d}t}\,\mathrm{d}x. \tag{4.23}$$

Combining the results of equations (4.22) and (4.23) gives the following result

$$\mathrm{d}t = \sqrt{\frac{1 + (\frac{\mathrm{d}y}{\mathrm{d}x})^2}{2gy}}\,\mathrm{d}x. \tag{4.24}$$

Integrating this equation gives the time it takes the particle to travel from point A to B over the curve y

$$t = \int_{x_A}^{x_B} \sqrt{\frac{1 + (\frac{\mathrm{d}y}{\mathrm{d}x})^2}{2gy}}\,\mathrm{d}x. \tag{4.25}$$

The goal is to minimize t so that the curve from A to B will be the shortest path between the two points. To minimize t the Euler-Lagrange equation is used.
The Euler-Lagrange equation states that if S is defined by an integral of the form

$$S = \int L(x, y, \frac{\mathrm{d}y}{\mathrm{d}x})\,\mathrm{d}x, \tag{4.26}$$

then S has a stationary value if the following equation is satisfied:

$$\frac{\partial L}{\partial y} - \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\partial L}{\partial y_x}\right) = 0, \tag{4.27}$$

where $y_x = \frac{\mathrm{d}y}{\mathrm{d}x}$.
For equation (4.25) it holds

$$L(x, y, y_x) = \sqrt{\frac{1 + (y_x)^2}{2gy}}. \tag{4.28}$$

Note that $L(x, y, y_x)$ according to (4.28) does not explicitly depend on x; $\frac{\partial L}{\partial x} = 0$. Therefore the Beltrami identity can be applied. First the Beltrami identity is derived.
Consider

$$\frac{\mathrm{d}L}{\mathrm{d}x} = \frac{\partial L}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x} + \frac{\partial L}{\partial y_x}\frac{\mathrm{d}^2y}{\mathrm{d}x^2} + \frac{\partial L}{\partial x}, \tag{4.29}$$

$$\frac{\partial L}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}L}{\mathrm{d}x} - \frac{\partial L}{\partial y_x}\frac{\mathrm{d}^2y}{\mathrm{d}x^2} - \frac{\partial L}{\partial x}. \tag{4.30}$$

The Euler-Lagrange equation (4.27) is multiplied by $\frac{\mathrm{d}y}{\mathrm{d}x}$, this gives

$$\frac{\partial L}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x} - \frac{\mathrm{d}y}{\mathrm{d}x}\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\partial L}{\partial y_x}\right) = 0. \tag{4.31}$$

Now substitution of equation (4.30) into equation (4.31) gives

$$\frac{\mathrm{d}L}{\mathrm{d}x} - \frac{\partial L}{\partial y_x}\frac{\mathrm{d}^2y}{\mathrm{d}x^2} - \frac{\partial L}{\partial x} - \frac{\mathrm{d}y}{\mathrm{d}x}\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\partial L}{\partial y_x}\right) = 0, \tag{4.32}$$

$$-\frac{\partial L}{\partial x} + \frac{\mathrm{d}}{\mathrm{d}x}\left(L - \frac{\mathrm{d}y}{\mathrm{d}x}\frac{\partial L}{\partial y_x}\right) = 0. \tag{4.33}$$

Since $\frac{\partial L}{\partial x} = 0$, this finally leads to the Beltrami identity

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(L - \frac{\mathrm{d}y}{\mathrm{d}x}\frac{\partial L}{\partial y_x}\right) = 0, \tag{4.34}$$

$$L - \frac{\mathrm{d}y}{\mathrm{d}x}\frac{\partial L}{\partial y_x} = C. \tag{4.35}$$

Now the Beltrami identity is applied to the problem therefore $\frac{\partial L}{\partial y_x}$ is computed

$$\frac{\partial L}{\partial y_x} = \frac{y_x}{\sqrt{2gy(1 + (y_x)^2)}}. \tag{4.36}$$

Substituting this into the Beltrami identity (4.35) leads to

$$C = \sqrt{\frac{1 + (y_x)^2}{2gy}} - \frac{(y_x)^2}{\sqrt{2gy(1 + (y_x)^2)}}, \tag{4.37}$$

$$= \frac{1 + (y_x)^2 - (y_x)^2}{\sqrt{2gy(1 + (y_x)^2)}}, \tag{4.38}$$

$$= \frac{1}{\sqrt{2gy(1 + (y_x)^2)}}. \tag{4.39}$$

Squaring both sides of the last equation and rearranging leads to the following differential equation

$$\frac{1}{y(1+(y_x)^2)} = 2C^2 g, \tag{4.40}$$

$$y(1+(y_x)^2) = \frac{1}{2C^2 g} =: K, \tag{4.41}$$

$$(y_x)^2 = \frac{K}{y} - 1, \tag{4.42}$$

$$y_x = \sqrt{\frac{K-y}{y}}. \tag{4.43}$$

The solution to this differential equation will give the curve for which the moving particle will travel in the shortest time from point A to point B. To solve the differential equation it is written as

$$\mathrm{d}y = \sqrt{\frac{K-y}{y}}\,\mathrm{d}x, \tag{4.44}$$

$$x = \int \sqrt{\frac{y}{K-y}}\,\mathrm{d}y. \tag{4.45}$$

In order to integrate this equation the following substitution is used

$$y = \frac{K}{2}(1-\cos\theta) = K\sin^2\frac{\theta}{2}, \tag{4.46}$$

where θ is equal to the angle between the y-axis and the tangent of the curve in the point (x, y).
Then

$$x = \int \sqrt{\frac{\sin^2\frac{\theta}{2}}{1-\sin^2\frac{\theta}{2}}}\,K\sin\frac{\theta}{2}\cos\frac{\theta}{2}\,\mathrm{d}\theta, \tag{4.47}$$

$$= \int \frac{\sin\frac{\theta}{2}}{\cos\frac{\theta}{2}}\,K\sin\frac{\theta}{2}\cos\frac{\theta}{2}\,\mathrm{d}\theta, \tag{4.48}$$

$$= \int K\sin^2\frac{\theta}{2}\,\mathrm{d}\theta, \tag{4.49}$$

$$= \frac{K}{2}(\theta-\sin\theta) + D. \tag{4.50}$$

The integration constant D is determined by the initial condition $x = 0, \theta = 0$, thus $D = 0$. Eventually the solution to the brachistochrone problem is given by the parametric equations

$$x = \frac{K}{2}(\theta-\sin\theta), \tag{4.51}$$

$$y = \frac{K}{2}(1-\cos\theta). \tag{4.52}$$

Finally the constant K can be determined by the requirement that the particle has to go through the point $B = (x_B, y_B)$. This requirement leads to a system of equations that can be easily solved

$$x_B = \frac{K}{2}(\theta_B - \sin\theta_B), \tag{4.53}$$

$$y_B = \frac{K}{2}(1 - \cos\theta_B). \tag{4.54}$$

Now the exact solution to the brachistochrone problem is fully known.
For this solution the time it takes a particle to travel from the point A to the point B can be calculated as follows

$$t = \int_{x_A}^{x_B} \sqrt{\frac{1 + (\frac{\mathrm{d}y}{\mathrm{d}x})^2}{2gy}}\,\mathrm{d}x \tag{4.55}$$

$$= \int_{\theta_A}^{\theta_B} \sqrt{\frac{1 + \left(\frac{\sin\theta}{1-\cos\theta}\right)^2}{gK(1-\cos\theta)}}\,\frac{K}{2}(1-\cos\theta)\,\mathrm{d}\theta \tag{4.56}$$

$$= \frac{1}{2}\sqrt{\frac{K}{g}} \int_{\theta_A}^{\theta_B} \sqrt{\left(1 + \left(\frac{\sin\theta}{1-\cos\theta}\right)^2\right)(1-\cos\theta)}\,\mathrm{d}\theta \tag{4.57}$$

$$= \frac{1}{2}\sqrt{\frac{K}{g}} \int_{\theta_A}^{\theta_B} \sqrt{1 - \cos\theta + \frac{\sin^2\theta}{1-\cos\theta}}\,\mathrm{d}\theta \tag{4.58}$$

$$= \frac{1}{2}\sqrt{\frac{K}{g}} \int_{\theta_A}^{\theta_B} \sqrt{\frac{(1-\cos\theta)^2 + \sin^2\theta}{1-\cos\theta}}\,\mathrm{d}\theta \tag{4.59}$$

$$= \frac{1}{2}\sqrt{\frac{K}{g}} \int_{\theta_A}^{\theta_B} \sqrt{\frac{2 - 2\cos\theta}{1-\cos\theta}}\,\mathrm{d}\theta \tag{4.60}$$

$$= \frac{1}{2}\sqrt{\frac{K}{g}} \int_{\theta_A}^{\theta_B} \sqrt{2}\,\mathrm{d}\theta \tag{4.61}$$

$$= \sqrt{\frac{K}{2g}}(\theta_B - \theta_A). \tag{4.62}$$

Here K is a known constant, g is the gravitational constant and $\theta_A,\ \theta_B$ can be determined from

$$x_i = \frac{K}{2}(\theta_i - \sin\theta_i), \qquad \text{for } i = A, B. \tag{4.63}$$

4.3 Discretization of the brachistochrone problem

To apply the particle swarm optimization algorithm to the brachistochrone problem first the problem needs to be discretized. Again the goal is to find the fastest path between the two given points A and B. In the previous section it was shown that the time it takes the particle to travel from point A to point B is given by (4.25). As quadrature formula to numerically integrate (4.25) the midpoint rule will be used. For this purpose, the x-axis is discretized into n equidistant intervals with size $\Delta x = \frac{x_B - x_A}{n}$. The x-coordinates of the interval boundaries are given by:

$$x_i = x_A + i\Delta x, \qquad \text{for } i = 0, 1, \ldots, n. \tag{4.64}$$

For each x_i there is a corresponding y_i. Now the solution and its derivative in the midpoints $i + \frac{1}{2}$ of the intervals are discretized as

$$y_{i+\frac{1}{2}} \to \frac{y_{i+1} + y_i}{2}, \tag{4.65}$$

$$\left(\frac{\mathrm{d}y}{\mathrm{d}x}\right)_{i+\frac{1}{2}} \to \frac{y_{i+1} - y_i}{\Delta x}. \tag{4.66}$$

Hence, the quadrature formula can be written out as:

$$t = \sum_{i=0}^{n-1} \sqrt{\frac{1 + \left(\frac{y_{i+1} - y_i}{\Delta x}\right)^2}{g(y_{i+1} + y_i)}} \Delta x. \tag{4.67}$$

This formula is used as the minimization function for the PSO algorithm. But before the PSO algorithm is applied the minimization function is examined around the point A.
The minimization function described above can be applied to any path that leads from the point A to the point B. So let us first take a look at the discretization of the exact solution found in the previous section. The exact solution is a cycloid that starts in the point A; $(x, y) = (0, 0)$. In figure 4.3 it can be seen that the derivative in A might be infinite. Therefore the derivative around the point A is derived. The exact solution is given by (4.51) and (4.52).
Now the derivative is given by:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\frac{\mathrm{d}y}{\mathrm{d}\theta}}{\frac{\mathrm{d}x}{\mathrm{d}\theta}} = \frac{\sin\theta}{1 - \cos\theta}. \tag{4.68}$$

In the neighbourhood of point A the parameter θ will be small; for the derivative near A it follows that:

$$\lim_{\theta \to 0} \frac{\mathrm{d}y}{\mathrm{d}x} = \lim_{\theta \to 0} \frac{\sin\theta}{1 - \cos\theta} = \infty. \tag{4.69}$$

Hence, the integrand in (4.25) behaves singularly. Although this could be easily avoided, the singular behavior is even strengthened by the integrand's zero denominator in point A. The singularity can not be resolved sufficiently accurately by the current discretization. Since our interest is in investigating the PSO algorithm, we refrain from curing the discretization near the singularity. Instead, the exact solution will be used near A. I.e., the positions of the points outside the neighbourhood of A will be optimized, not those in this neighbourhood.

4.4 Numerical solution with PSO

In this section the brachistochrone problem that was posed by Johann Bernoulli will be solved using the particle swarm optimization algorithm. The goal of this optimization is to find the path that will take the shortest time for a particle to travel from the point A to the point B. The points A and B are given and it is assumed the particle will fall under gravitational force, g. The minimization function for the traveling time t is given by (4.67).
The PSO algorithm that will be used to minimize the given function is (2.3)-(2.4). The random numbers r_1 and r_2 will be different for all particles in the swarm for each new iteration step of the algorithm. They will be uniformly generated from the range $(0, 1)$, using the *Matlab* command *rand()*.
For the particle swarm optimization algorithm an initial swarm of particles is needed. For this optimization problem the particles will be different paths that go through the points A and B.
There are many parameters that influence the performance of the PSO algorithm. For instance, the initial swarm of paths will influence the performance. Not only the number of paths in the initial swarm will have an effect, also the placement and the shape of the paths will have an effect on the performance of the PSO algorithm. It is desirable that the paths will be placed well distributed over the entire search space. If for instance all initial paths will be placed say above the exact solution of the brachistochrone problem, the PSO algorithm is unlikely to ever find the exact solution. Concerning the shape of the initial paths, the paths can be nice smooth differential functions that go through the points A and B, but it is also possible to select discontinuous paths or paths with many oscillations.

4.4.1 Parameter selection

The role the aforementioned parameters will play in the performance of the PSO algorithm will be investigated later. For now the performance of the PSO algorithm is analysed for the different choices of the tuning parameters a, b_1 and b_2. Therefore an initial swarm of eight paths is chosen. These paths will be well distributed over the entire search space.

The starting point A and ending point B will be fixed and are chosen as, $A = (0, 2)$ and $B = (2, 0)$. As initial paths the following function types are selected.

$$y = \begin{cases} C\sqrt{x} + D \\ Cx + D \\ C(x - x_B)^2 + D \\ C(x - x_B)^{50} + D \\ C \\ Cx + \cos\left(\frac{\pi}{L}(x - x_A)\right) + D \\ \frac{C}{x} + D \\ C(x - \pi)^2 + D \end{cases} \tag{4.70}$$

In the section 4.3 it was decided that because of the singularity in point A, within a neighbourhood of the point A the exact solution is used for all the initial paths, to overcome expected large discretization errors around the point A.
The constants C and D are determined such that the paths will connect to the exact solution near the point A and will go through the point B. The constant L is selected in such a way that the cosine function has two periods between the points A and B. The constant function connects to the exact solution at the boundary of the neighbourhood around the point A. To let the constant function go through the point B the function makes a jump at $x = x_B$. Furthermore for the first quadratic function and the function of 50th degree the vertex of the function is in the point B, while for the second quadratic function the vertex is in the point $x = \pi$. This leads to the set of initial paths given in figure 4.4.
Before the PSO algorithm is applied to the initial swarm of paths, the time it takes for a particle to travel from the point A to the point B is calculated for all the initial paths. For calculation purposes the gravitational force is set to $g = 1$. The traveling times are also given in figure 4.4. In the initial step of the PSO algorithm each path's current shape is set as p_1 and the path with the shortest traveling time is set as p_2. So in this example the square root function is the best initial solution and set equal to p_2.

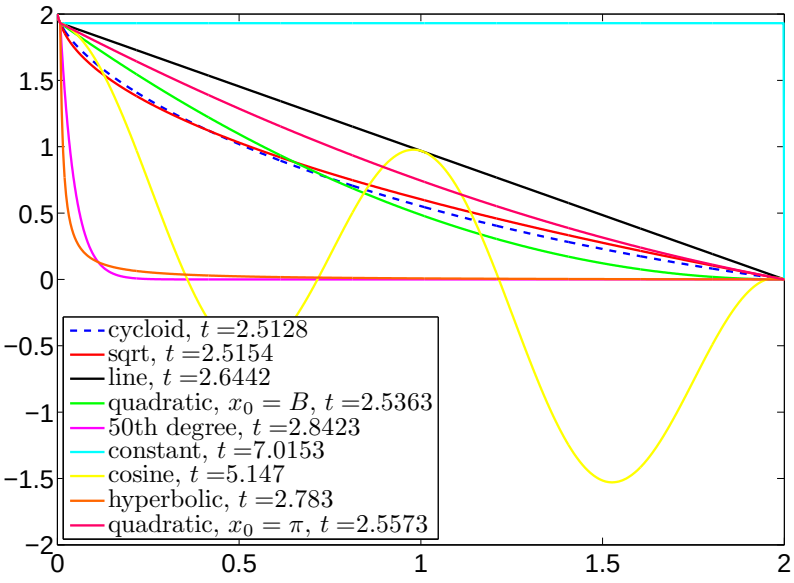


Figure 4.4: Initial paths and exact solution for the brachistochrone problem

Now the PSO algorithm is applied. After each iteration step the paths will have a new position and for this position the traveling time is computed. This time is compared with each path's own shortest traveling time that is stored in t_{p_1}. If the traveling time of the new path is shorter than the value t_{p_1} then the variable p_1 is updated. The new shape of the path will be set as p_1 and its traveling time is set as t_{p_1}. Also the variable p_2 will be updated if a new path will have a shorter traveling time than the shortest traveling time of all paths so far, which is stored in t_{p_2}.

In figure 4.5 the positions of the different paths are given after several iteration steps of the algorithm. In the figure it can be seen that after one iteration step the positions of the paths have not changed much. After some more iteration steps it becomes clear that the initial paths will converge towards the exact solution of the brachistochrone problem. The cycloid that represents the exact solution is given in the figure as the dashed line. For this arbitrary choice of the parameters a, b_1 and b_2 the PSO algorithm finds a good approximation of the exact solution within the maximum number of iteration steps. In the last frame of figure 4.5 all paths coincide by very good approximation with the exact solution. For all the paths the traveling time is now by good approximation equal to the traveling time of the exact solution.

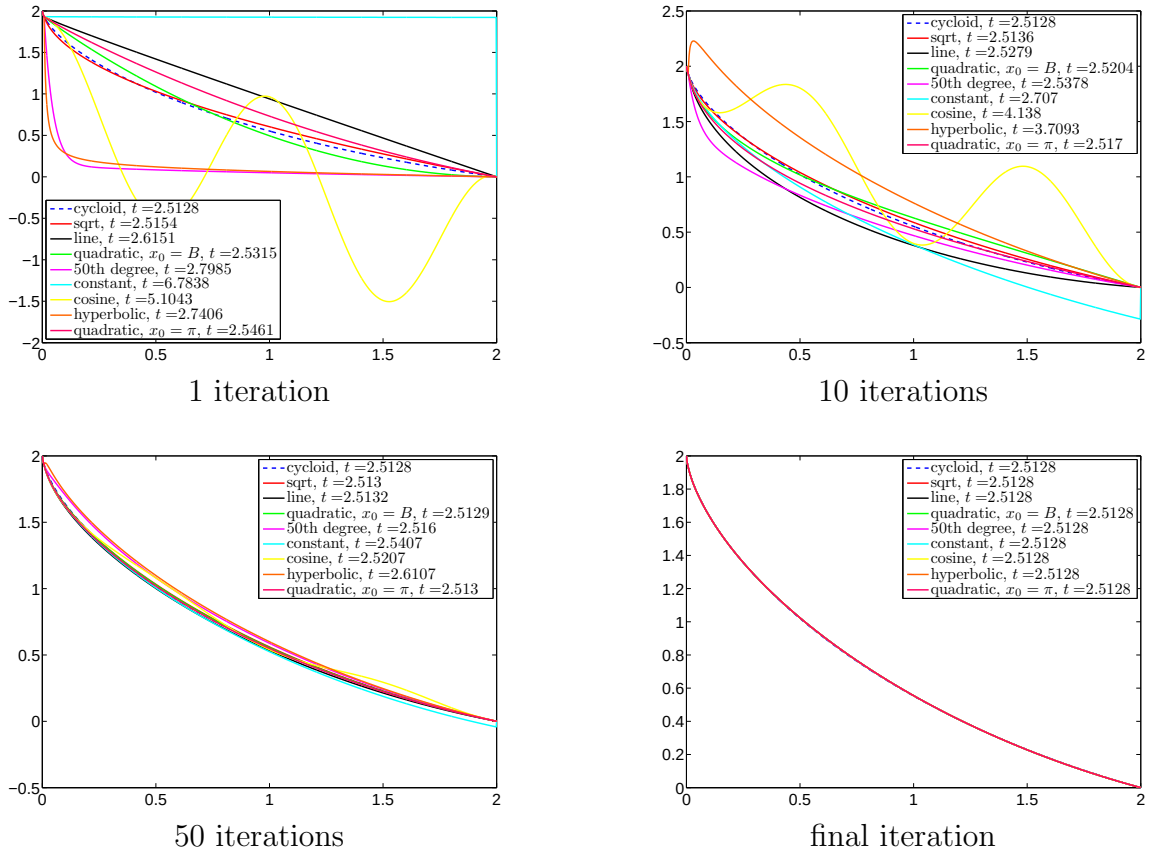


Figure 4.5: PSO algorithm after several iteration steps

Not for all choices of the parameters a, b_1 and b_2 the global optimum found by the PSO algorithm will be a good approximation to the exact solution.
Next the performance of the PSO algorithm is analysed for several different choices of the tuning parameters. The maximum number of iteration steps for the PSO algorithm is set to 1000. For all choices of the tuning parameters the PSO algorithm is simulated 50 times, to get a good mean performance of the algorithm. To keep track of the performance of the algorithm a number of variables is analysed. These variables are: the number of times the global optimum p_2 is improved, the difference in traveling time between the exact solution and the best approximation of the PSO algorithm and the 2-norm of the difference between the exact solution and the best approximation.
The number of times the parameter p_2 is improved gives an indication of how fast the PSO algorithm converges towards the best solution. The difference in traveling time and the norm indicate how well the exact solution is approximated by the PSO algorithm.
In appendix B.1 the plots of these variables are given. For each choice of the tuning parameters the time difference, the norm and the number of improvements are given for all 50 simulation runs, see figure B.1. Since there is a dependency between the norm and

the time difference, they are given in the same figure and the number of improvements is given separately. The dependence between the norm and the time difference arises from the fact that if the time difference of a solution is larger it is also likely that the norm for that solution is larger. However, it not needs to be, it is possible that the norm for a particular solution is smaller than for another solution while that solution has the smaller time difference.
To analyse the performance of the PSO algorithm the mean of these variables is recorded. For each choice of the tuning parameters the mean norm, the mean time difference and the mean number of improvements is given in table 4.1.
The same tuning parameters are chosen as for the optimization problem on Schaffer's F6 function that is described before. For the parameter choices in case (c) it is possible that the PSO algorithm will break down. If in the first iteration steps the random numbers are large than it is possible that a path will move outside the physical domain of the problem. The physical domain being the space below the height of the starting point A. Outside the physical domain the path will have a shape such that a particle following that path, will not reach the point B under gravity. This causes the path to have a complex valued traveling time, which leads to breakdown of the algorithm. If for the first iteration steps of the PSO algorithm the random numbers are moderate than this behavior will not occur.
The same breakdown behavior occurred for case (c) for the optimization of Schaffer's F6 function. Therefore case (c) is included twice in table 4.1. Case (c_1) represents the mean values over all 50 simulation runs of the PSO algorithm, while case (c_2) represents the mean values over the cases where the PSO algorithm did not break down.

Case	Parameters (a, b_1, b_2)	Mean norm	Mean time difference	Mean number of improvements of p_2
a	$(0.9, 0.1, 0.1)$	0.1016	3.560×10^{-5}	555
b	$(0.7, 0.3, 0.3)$	0.1722	8.256×10^{-5}	190
c_1	$(0.9, 2, 2)$	2.295×10^{12}	–	29
c_2	$(0.9, 2, 2)$	0.3647	2.639×10^{-4}	10
d	$(0.1, 0.1, 0.1)$	0.4615	3.206×10^{-4}	276
e	$(0.1, 2, 2)$	0.5103	5.115×10^{-4}	302
f	$(-0.7, 0.5, 0.5)$	0.2522	1.536×10^{-4}	226
g	$(0.9, 0.9, 0.1)$	0.1244	5.732×10^{-5}	809
h	$(0.9, 0.1, 0.9)$	0.0398	1.156×10^{-5}	530

Table 4.1: Performance properties for several choices of parameters a, b_1 and b_2.

Now the results for the different choices of the tuning parameters are analysed and compared with the previous result for the test problem and the problem of Schaffer's F6 function. For case (a) the PSO algorithm approximates the exact solution quite well. The mean norm and mean time difference for this case are the smallest of all cases where the parameters b_1 and b_2 are chosen equal, see table 4.1.
In figure 4.6 an example is given of the approximation that is found by the PSO algorithm. All the paths in the swarm have settled by very good approximations on the exact solution and have practically the same traveling time.

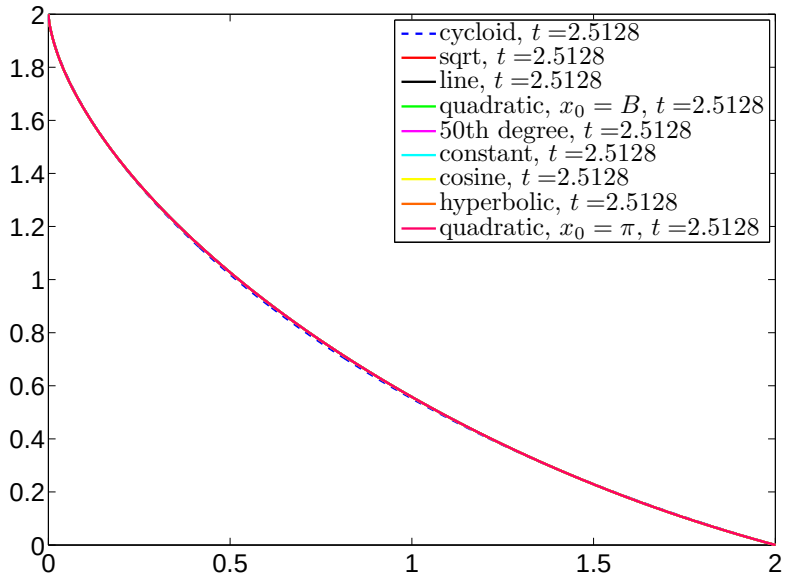


Figure 4.6: PSO algorithm, best approximations for case (a)

For the parameter choices in case (a) the best approximations, p_2, are on average updated 558 times. That is roughly half of the maximum number of iteration steps. This indicates that if the tuning parameters are selected as in case (a) the PSO algorithm slowly converges towards the globally best solution. This result corresponds with the results for the test problem and the optimization of Schaffer's F6 function. There also for the parameter selection of case (a), the PSO algorithm slowly converged towards the globally best solution.

If a faster convergence of the algorithm is needed than it might be better to select the parameters as in case (b). For the brachistochrone problem this case will also find a good approximation of the exact solution. Although this approximation is somewhat less accurate than the approximation found in case (a), the algorithm converges much faster. The mean number of updates of the best approximation of the algorithm is less than 200. So

for case (b) the algorithm converges faster but settles on a somewhat less accurate approximation to the exact solution. This behavior of the algorithm corresponds with the other two test problems considered before. There also the PSO algorithm converged more quickly, but sometimes setting on a non optimal solution.

For case (c) the algorithm can break down. The reason why this might happen has been discussed before in section 3.3. Now the performance of the PSO algorithm is investigated for the simulation runs where it did not break down. From the total number of 50 simulation runs, in 18 simulation runs the PSO algorithm did not break down, meaning the algorithm found a valid approximation. In table 4.1 it can be seen that this approximation is found very fast. Within a few iteration steps the algorithm reaches its best approximation. This approximation is not as good as for the cases (a) and (b) though. But since the algorithm finds this approximation very fast, the parameter selection of case (c) might be a good choice to explore a large initial search space. For this selection of the parameters one gets a good idea of the region where the exact solution of the brachistochrone problem is. So one could exploit this region further with another parameter selection so that the exact solution can be approximated more accurately.

For case (d) it was shown for the test problem and for the problem on Schaffer's F6 function that the algorithm converged fast towards the best approximation found so far without further exploring the search space. For the problem on Schaffer's F6 function this resulted in the situation that the PSO algorithm could not find the global maximum of the function in any of the simulation runs. This was due to the fact that the global optimum of the function was located at the edge of the search space, while the PSO algorithm converged towards a maximum value more in the middle of the search space. Now for the brachistochrone problem the results are better. The algorithm finds a decent approximation of the exact solution. The reason that the PSO algorithm will find a decent approximation in this case is that the exact solution to the problem lies in the middle of the search space. So for the parameter choices of case (d) the PSO algorithm still converges steadily towards the best approximation found so far without further exploring the search space. The fact that the algorithm converges steadily can be seen by the number of times the best approximation is updated. This happens in roughly a quarter of the number of iteration steps allowed. This shows that the algorithm has a rather fast convergence towards its best approximation.

For the parameter selection of case (e) it was seen in the test problem that the PSO algorithm will have a zigzagging behavior when it converges towards its best approximation. Now for the brachistochrone problem this zigzagging behavior of the algorithm leads to the fact that the performance of the PSO algorithm can differ between simulation runs. For some simulation runs the algorithm finds a good approximation to the exact solution within a small number of iteration steps. While for other simulation runs the approximation to the exact solution might be worse and the algorithm needs significantly more iteration steps to find it. This large spread in the performance of the PSO algorithm can be seen in appendix B.1 in figure (e). On average the performance of the PSO algorithm

is comparable with the performance of the algorithm in case (d). The mean number of updates of the best approximation is moderate and the exact solution is approximated about as well as in case (d). However, these approximations are less good than the approximations found in cases (a) and (b).

A better approximation to the exact solution is found for the parameter selection of case (f), see table 4.1. The reason that the approximations found for this case of parameter selection are rather good is that the PSO algorithm has a small zigzagging behavior while it converges steadily towards the best approximation. This behavior has been shown for the test problem.

Finally there are the two cases of the parameter selection where the parameters b_1 and b_2 are chosen to be different. The performance of the PSO algorithm for these two cases is very good. For both cases the PSO algorithm finds very good approximations to the exact solution, with the mean norm and the mean time difference being very small. The mean number of updates of the best approximation is rather high though for the parameter selection of case (g). In that case the best approximation is updated in almost every iteration run of the algorithm. So the final approximation might be very good, the PSO algorithm will only converge very slowly towards it.

For case (h) the performance of the algorithm is the best of all the cases that are considered. The mean norm and time difference are very small, so one can say that the exact solution will be found by the PSO algorithm for this parameter selection. The number of updates of the best approximation is also moderate in this case. The reason why the algorithm performs so well in this case is that the tuning parameter b_2 is chosen larger than the tuning parameter b_1. The parameter b_2 corresponds with the globally best approximation that the PSO algorithm uses. For this choice the PSO algorithm converges faster towards the best approximation and will exploit the search space around it more thoroughly.

4.4.2 Number of initial paths

As said before there are many variables that can influence the performance of the PSO algorithm. In this section the role of the number of initial paths in the swarm will be investigated. For this analysis some sets of initial paths are used. These sets make use of a basis function that will be the same for all initial paths that are in a set. The function that will be used is the following

$$y = Cx^m + D \qquad \text{for } m \in \mathbb{Q}. \tag{4.71}$$

Again, the constants C and D will be determined such that the initial paths will connect to the exact solution at the edge of the neighbourhood around the point A and such that the paths will go through the point B.

The number of paths in a set will vary but to keep the area of the search space equal, the smallest and largest number of m is the same in all different sets. The number of particles in the sets that will be considered will be equal to $2n+1$, where $n = 1, 2, 4, 8, 16$. The numbers m in a set are chosen in such a way that the initial paths will be distributed well over the search space. To obtain this good distribution n numbers are chosen as $m_1, m_2, \ldots, m_n \in \mathbb{N}$. Then the other n numbers are chosen as $\frac{1}{m_1}, \frac{1}{m_2}, \ldots, \frac{1}{m_n}$ and finally one m will be equal to 1 in all sets. Figure 4.7 shows how the different initial sets of paths are distributed over the search space. Here the points A and B are chosen the same as before, so $A = (0, 2)$ and $B = (2, 0)$. In the plots the exact solution and its traveling time is also given.

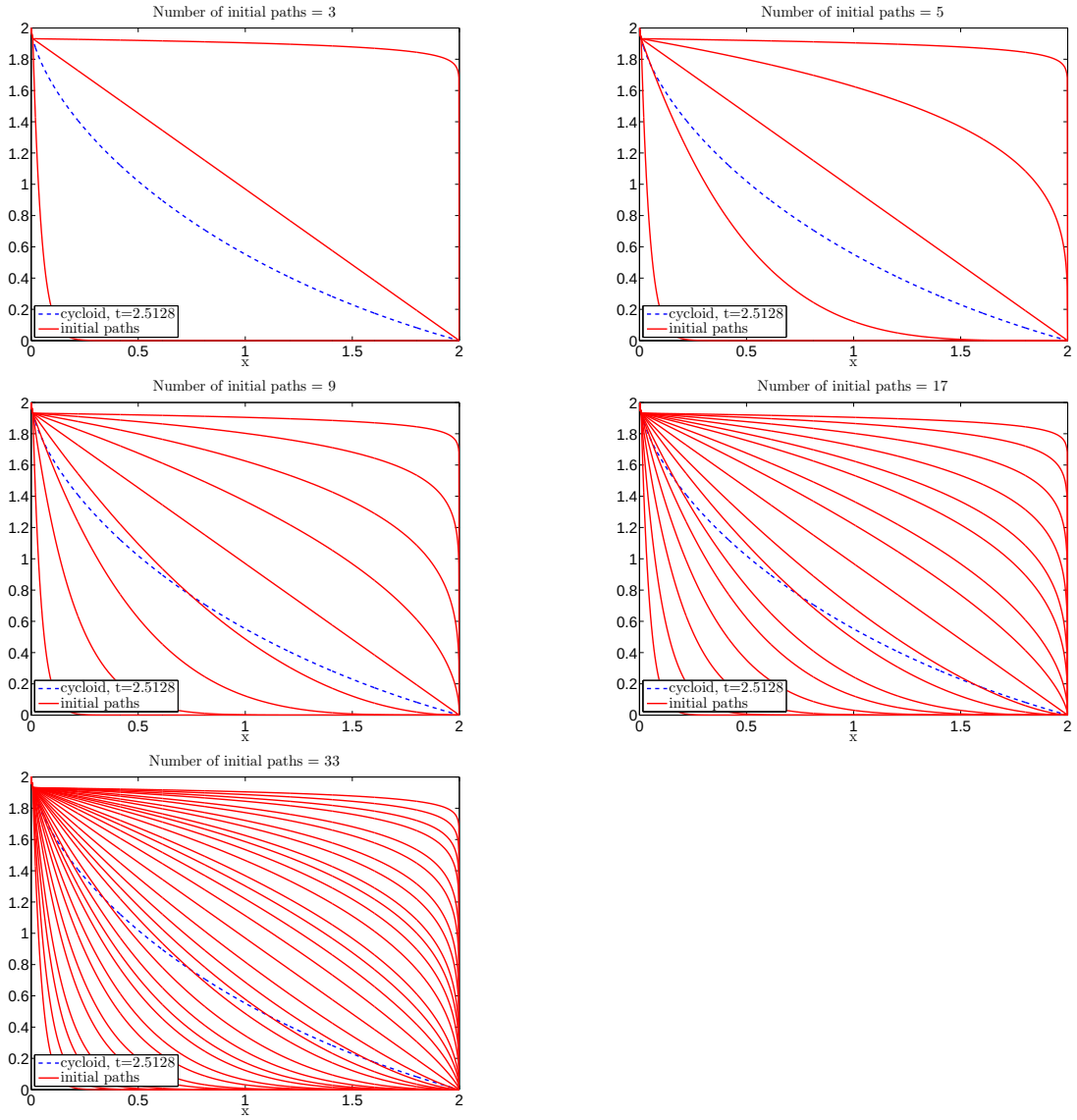


Figure 4.7: Initial paths for different numbers of initial paths

Now the PSO algorithm will be applied with these sets of initial paths. The goal is to find the exact solution to the brachistochrone problem. For this purpose, again the PSO algorithm is simulated 50 times and, again, the maximum number of iteration steps per simulation is equal to 1000. The parameter selection that is used for these simulation runs is that of case (a), so $a = 0.9,\ b_1 = 0.1,\ b_2 = 0.1$. This parameter selection is chosen, because in the previous section it was shown that the PSO algorithm performs very well for this choice of parameters.
To keep track of the performance of the algorithm again the same three variables are used as before. So the difference of the norm and the difference of the traveling time between the best approximation and the exact solution is recorded, as well as the number of times the best approximation of the algorithm is updated. In appendix B.2 the plots of the variables are given. For each number of initial paths in the swarm the time difference, the norm and the number of improvements are given for all 50 simulation runs, see figure B.2.
To analyse the performance of the PSO algorithm the mean of these variables is again recorded. For each number of initial paths the mean norm, the mean time difference and the mean number of improvements is given in table 4.2.

Case	Number of initial paths	Mean norm	Mean time difference	Mean number of improvements of p_2
a	$m = 3$	4.903	0.0400	199
b	$m = 5$	0.3781	7.009×10^{-4}	547
c	$m = 9$	0.2591	4.520×10^{-4}	631
d	$m = 17$	0.1875	3.340×10^{-4}	731
e	$m = 33$	0.1345	1.954×10^{-4}	826

Table 4.2: Performance properties for several numbers of initial paths, $a = 0.9,\ b_1, b_2 = 0.1$.

The results in this table show that for the PSO algorithm where 3 initial paths are used, the exact solution to the brachistochrone problem will not be found. So an initial swarm of three paths is too small to perform a proper optimization for the brachistochrone problem. For the other numbers of initial paths, the PSO algorithm will find a good approximation to the exact solution. Now if these results are compared it shows that the approximation becomes more accurate if the swarm is extended. Each doubling in the number of initial paths leads to an even more accurate solution of the optimization problem. These doublings of number of initial paths lead to a decrease of roughly 30% for the mean norm and mean time difference.
Each doubling in the number of initial paths also leads to an increase in the mean number of improvements needed to find the best approximation. Moreover, the computational time of the algorithm will increase drastically. Each doubling in number of initial paths leads to a doubling in computational time of the algorithm. So there is a trade-off. For a quick but less accurate search towards the exact solution a low number of initial paths in the swarm is favorable, while for a more accurate approximation of the exact solution a higher number of initial paths is recommended, at the cost of a larger computational time of the algorithm.

4.4.3 Distribution of initial paths

Another fact that influences the performance of the PSO algorithm is the distribution of the initial paths over the search space. In this section a brief example will be given of how the algorithm will perform for different distributions of the initial paths. For these paths the same basis function, (4.71), is used as in the previous section. The number of initial paths is chosen to be equal to 9. Furthermore, all the data will be the same as in the previous section. So $A = (0, 2)$, $B = (2, 0)$, $a = 0.9$, $b_1 = 0.1$, $b_2 = 0.1$, maximum number of iteration steps is 1000 and number of simulation runs is 50.
The initial paths will be distributed in three qualitatively different ways over the search space. First the initial paths will be distributed uniformly over the entire search space. For the other two cases the initial paths will be distributed over the upper part respectively the lower part of the search space. In figure 4.8 the distribution of the initial paths can be seen. Also the exact solution and its traveling time is given.

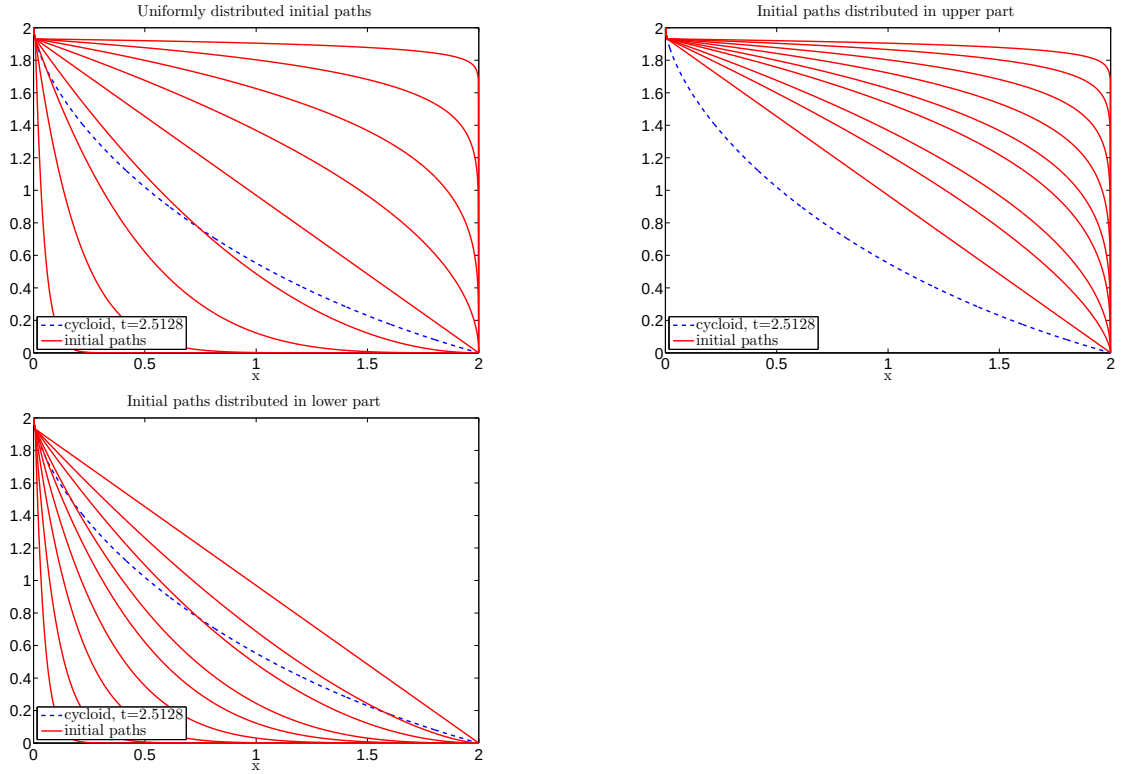


Figure 4.8: Initial paths distributed differently over the search space

Now the PSO algorithm is applied to the latter two sets of initial paths. The set in which the initial paths are distributed uniformly over the search space is equal to that in the previous section, where the number of initial paths was equal to 9. So for this set the same data will be used. Again for comparing the performance of the algorithm the difference in the norm and traveling time are recorded, as well as the number of improvements of the best approximation. These results are given in appendix B.3 in figure B.3.

To investigate the performance of the PSO algorithm the mean of these variables is recorded. For each different distribution of the initial paths the mean norm, the mean time difference and the mean number of improvements is given in table 4.3.

Distribution	Mean norm	Mean time difference	Mean number of improvements of p_2
Uniform	0.2591	4.520×10^{-4}	631
Upper part	6.765	0.0744	728
Lower part	0.1620	2.993×10^{-4}	616

Table 4.3: Performance properties for different distributions of initial paths, $a = 0.9,\ b_1 = b_2 = 0.1$.

The results in the table show that if the initial paths are all distributed in the upper part of the search space, then the exact solution to the brachistochrone problem will not be found. Further, if the initial paths are all distributed in the lower part of the search space, the algorithm performs slightly better than for the case where all initial paths are distributed uniformly over the entire search space. This result was to be expected. If the initial paths are all distributed in the upper part of the search space, then the exact solution to the brachistochrone problem lies outside the region that is spanned by the initial paths. So unless the paths in the algorithm will have a strong exploring behavior the exact solution will not be found. For the parameter selection $a = 0.9,\ b_1 = b_2 = 0.1$ the paths will not have a strong exploring behavior. They will converge to a solution that is within their initial search region. The fact that the algorithm performs better when the initial paths are distributed in the lower part of the search space can also be explained. For this case more initial paths are closer to the exact solution than is the case if the initial paths are distributed uniformly over the search space. Therefore the exact solution will be localized faster and thus the region around the exact solution will be better exploited.

4.4.4 Shape of initial paths

The last fact that will be discussed that has an effect on the performance of the algorithm is the shape of the initial paths. For this analysis no new runs of the PSO algorithm have to be performed. The influence of the shape of the initial paths will be discussed on the basis of known results.

To get an idea of the influence of the shape of the initial paths on the performance of the algorithm two results from the previous sections will be compared. The first result that is used is the result obtained by the PSO algorithm with the parameter selection case (a), given in section 4.4.1. This result will be compared with the result obtained by the PSO algorithm with the number of initial paths equal to 9, as presented in section 4.4.2.

To obtain the first result eight initial paths that all have a qualitatively different basis function have been used. The shapes of these initial paths differs strongly. Some paths are constant on a large part of the domain while another path has some oscillations, see figure 4.4. For the second result, nine initial paths have been used, which all have the same basis function. Therefore their shapes are more identical, see figure 4.7(c). In table 4.4 the results for these two different set of initial paths are repeated.

	Mean norm	Mean time difference	Mean number of improvements of p_2
Varying shapes	0.1071	3.924×10^{-5}	558
Similar shapes	0.2591	4.520×10^{-4}	631

Table 4.4: Performance properties for different shapes of initial paths.

From these results it can be concluded that it is wise to give the initial paths shapes that are qualitatively more different, since for that case the algorithm outperforms the case where the shapes of the initial paths are more similar.

Chapter 5

Conclusion

5.1 Main achievements

In this master thesis the concept of particle swarm optimization has been described. The origin of the method was described and the terms in the algorithm were explained. The purpose of the latter was to understand how the particle swarm optimization algorithm works and what the role of each individual term in the algorithm is. Next a convergence analysis was made of the particle swarm optimization algorithm, for which the randomness was stripped from the algorithm. The analysis showed that the addition of the tuning parameters c and d into the algorithm is unnecessary. The analysis also led to some guidelines for the values of the remaining tuning parameters a, b_1 and b_2.
The guidelines for choosing these tuning parameters were tested on three optimization problems. The performance of the particle swarm optimization algorithm on these optimization problems was monitored for different choices of these parameters. This led to the conclusion that for some parameter selections the particle swarm optimization algorithm performs better than for some other parameter selections. For instance, for the parameter selections in the cases (a) and (b) the particle swarm algorithm approximates the exact solution to the optimization problem more accurately. This does not mean that the other cases are useless. Some of these cases will lead to a significant shorter computational time, for example case (c). This might be favored for a quick but less accurate approximation to the exact solution. In combination with a more accurate search on the region of this first approximation, the particle swarm algorithm might also give some good results.
Lastly some other facts that will influence the performance of the particle swarm optimization algorithm were investigated. From this it is concluded that the particle swarm optimization works best if the number of initial paths is chosen not too large, and that these initial paths should preferably have qualitatively different shapes and be well distributed over the search space. For initial swarms that fulfill these recommendations the particle swarm optimization algorithm will most likely find a good approximation to the optimization problem that is considered.

5.2 Suggestions for further research

In this chapter some ideas for further research will be discussed. Some of these ideas have already been touched upon in this thesis. Further research might give some new insights in the role the ideas might have on the particle swarm optimization algorithm.

Tuning parameter a. For all the optimization problems considered in this thesis the parameter a is chosen to be fixed for all iteration steps of the algorithm. A possible modification of the particle swarm optimization algorithm is that the parameter a will differ over the iteration steps. The parameter can be decreased linearly, as introduced in [4]. Starting with large a the algorithm will have a more exploring behavior in the beginning. Then slowly decreasing a, the algorithm will get a more exploiting behavior towards the end. Also the parameter a can be decreased with a fraction, if for a certain number of iteration steps no improvement has been made. This was introduced in [8]. The parameter a can also be changed in a problem-independent way, as proposed in [5]. The latter method is based on the coefficient of variation of the goal function values. The parameter a will be changed if the coefficient of variation falls below a certain threshold value.

Tuning parameters b_1, b_2. The tuning parameters b_1 and b_2 are introduced to influence the attraction of the particles towards the best position. The attraction towards each particle's own best position is influenced by b_1 and the attraction towards the globally best position is influenced by b_2. For the optimization problems in this thesis the parameters b_1 and b_2 are chosen equal for most of the parameter selection cases. There are only two parameter selection cases described where the parameters are chosen unequal. In one case the parameter b_1 is chosen significantly larger than b_2 and in the other case it is the other way around. Both these cases are performed for the same choice of the parameter a, namely $a = 0.9$. Testing more parameter selection cases where the parameters b_1 and b_2 are chosen not equal for different choices of the parameter a might result in new insights in what role the individual parameters b_1 and b_2 have on the particle swarm algorithm.

Tuning parameters c, d. The tuning parameters c and d are introduced as tuning parameters for the function to update the position of the particles in the swarm. The origin of this addition arises in [6]. In the same article an analysis is performed on the particle swarm optimization algorithm. For this purpose, the random numbers where chosen to be fixed. This made the algorithm deterministic and some analysis on the behavior of the algorithm was performed. It led to the conclusion that the addition of the parameters c and d to the function to update the particle position is unnecessary. In the particle swarm optimization algorithms used in this thesis these parameters are therefore by definition set equal to 1. But since randomness is included in these algorithms there might be an advantage in choosing these parameters differently.

Number of particles in the swarm. The number of particles in the swarm has a great influence on the performance of the algorithm. For the brachistochrone curve problem the influence of the number of particles in the swarm has been investigated. It was found that for small numbers the algorithm was not able to accurately approximate the exact solution to the optimization problem. For larger numbers there was some gain in the algorithm's performance, but then the computational time of the algorithm became significantly larger. So there seems to be an optimal number for the particles in the swarm such that the algorithm will accurately find a solution to the optimization problem without iterating unnecessary particles. It would be nice if the number of particles to be used in the swarm can be determined in advanced for every optimization problem.

Discontinuous optimization problems. The optimization problems that are presented in this thesis are all continuous optimization problems. For continuous optimization problems other optimization algorithms exist that can find the exact solution to the problem. These other optimization methods may be gradient-based. The great advantage of the particle swarm optimization method is that it can also solve discontinuous optimization problems. How the particle swarm optimization is applied on such discontinuous optimization problems seems straightforward. The particles in the swarm will be discretized so that they are only positioned on discrete points. Then the particle swarm optimization algorithm can be applied in the same way as for continuous problem, but further study may give new insights in the way the particle swarm optimization algorithm can handle discontinuous problems.

Topology of the swarm. The particle swarm optimization algorithm that is used in this thesis shares the information between the particles in the same way. All particles in the swarm will know the information of all the other particles in the swarm. If a particle finds a new globally best position this information will be known immediately to all other particles. A modification to this global sharing of information is that the swarm will have a different topology. For instance each individual particle may only know the information of a certain number of particles that are within a neighbourhood around it, as proposed in [3]. This might benefit the exploration behavior of the algorithm, since it is now possible that groups of particles will explore different regions of the search space before convergence towards the globally best solution.

Random numbers. An important realization for a good performance of the particle swarm optimization is the addition of randomness to the algorithm. The randomness strengthens the dynamic behavior of the particle swarm and enhances the ability to find a solution to the optimization problem. In the particle swarm optimization algorithms that are used in this thesis the random numbers r_1 and r_2 are chosen uniformly in the domain $(0, 1)$. For each iteration step of the algorithm and for each particle these random numbers are picked again. This leads to an algorithm where randomness is present in every iteration step for every particle, which makes it challenging to do a convergence analysis for the algorithm. A modification might be to pick the random numbers in a different way.

For instance the random numbers may be picked per iteration step but kept the same for all particles during that iteration step. Or the random numbers may be chosen equal for several successive iteration steps. This may make a convergence analysis of the algorithm more doable. To get an idea of the influence of these different random number selections further study is required.

Craziness. A way to include more randomness into the particle swarm optimization method is via the variable craziness, which is presented in [1]. This variable selects some particles and resets there position to a new randomly chosen position in the search space. The selection of these particles and the moment when to reset their position depends on the behavior of the algorithm. If the behavior of the swarm becomes too uniform and the particles are all located within a small neighbourhood of each other it might be wise to reset some of these particles. This might enhance the exploration behavior of the particle swarm optimization algorithm. Whether the performance of the algorithm will benefit from this addition may be further investigated.

Appendix A

PSO algorithm performance for Schaffer's F6 function

In this appendix the performance of the particle swarm optimization algorithm can be found for the optimization of Schaffer's F6 function. In the table the number of times the algorithm found the exact solution is shown for all the different parameter selection cases. An entry of **fail** in the table means that the PSO algorithm did not find the exact solution within the maximum number of iteration steps. An entry of **break** in the table means that the PSO algorithm did break down for that parameter selection.

a	b	c	d	e	f	g	h
151	28	**fail**	**fail**	27	47	192	172
96	35	**fail**	**fail**	88	132	134	165
fail	**fail**	**fail**	**fail**	**fail**	**fail**	**fail**	132
89	53	**fail**	**fail**	152	**fail**	184	**fail**
133	**fail**	**fail**	**fail**	197	**fail**	113	148
128	**fail**	**fail**	**fail**	**fail**	**fail**	109	129
70	33	142	**fail**	15	**fail**	151	120
134	24	**fail**	**fail**	**break**	**fail**	**fail**	192
61	**fail**	**fail**	**fail**	197	**fail**	**fail**	**fail**
107	**fail**	**fail**	**fail**	**fail**	**fail**	158	116
112	**fail**	**fail**	**fail**	**fail**	63	110	154
139	17	**fail**	**fail**	72	**fail**	**fail**	186
149	30	**fail**	**fail**	**fail**	**fail**	100	150
152	**fail**	**fail**	**fail**	116	127	163	121
97	**fail**	**fail**	**fail**	**break**	**fail**	0	157
132	**fail**	**fail**	**fail**	49	**fail**	194	182
98	42	**fail**	**fail**	**break**	61	186	143
101	**fail**	**fail**	**fail**	93	**fail**	71	141
fail	**fail**	**fail**	**fail**	196	**fail**	94	174
130	**fail**	**fail**	**fail**	43	**fail**	49	138
142	**fail**	**fail**	**fail**	123	51	**fail**	162
31	**fail**	**fail**	**fail**	**fail**	127	**fail**	29
100	32	122	**fail**	38	**fail**	120	161
148	**fail**	**fail**	**fail**	72	**fail**	87	194
105	**fail**	**fail**	**fail**	47	65	167	158
77	41	**fail**	**fail**	46	96	129	149
123	30	**fail**	**fail**	65	**fail**	**fail**	129
127	20	**fail**	**fail**	197	**fail**	161	121
117	**fail**	**fail**	**fail**	75	**fail**	98	181
128	26	**fail**	**fail**	16	45	39	173
109	40	**fail**	**fail**	197	**fail**	134	112
90	32	**fail**	**fail**	**fail**	**fail**	176	124
92	39	**fail**	**fail**	26	**fail**	**fail**	153
120	**fail**	**fail**	**fail**	47	**fail**	**fail**	**fail**
98	**fail**	**fail**	**fail**	49	**fail**	**fail**	**fail**
fail	43	**fail**	**fail**	**fail**	**fail**	128	104
128	32	**fail**	**fail**	19	**fail**	83	**fail**
109	**fail**	**fail**	**fail**	60	**fail**	197	138
133	45	**fail**	**fail**	**break**	67	181	189
146	**fail**	**fail**	**fail**	30	**fail**	113	139
37	**fail**	**fail**	**fail**	**break**	**fail**	192	123
143	**fail**	**fail**	**fail**	51	**fail**	154	159
128	28	**fail**	**fail**	36	148	134	154
93	**fail**	**fail**	**fail**	37	90	39	141
115	**fail**	**fail**	**fail**	197	**fail**	110	129
92	43	**fail**	**fail**	77	**fail**	**fail**	163
fail	34	**fail**	**fail**	23	**fail**	**fail**	101
fail	**fail**	**fail**	**fail**	17	**fail**	56	141
168	**fail**	**fail**	**fail**	**break**	**fail**	**fail**	131
120	30	**fail**	**fail**	193	87	88	191

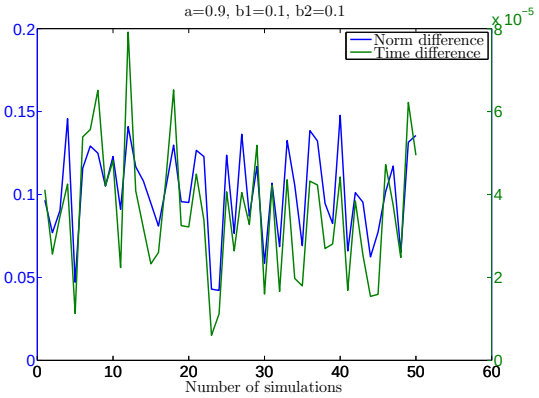
Table A.1: Number of iteration steps needed for finding global optimum using different parameter settings

Appendix B

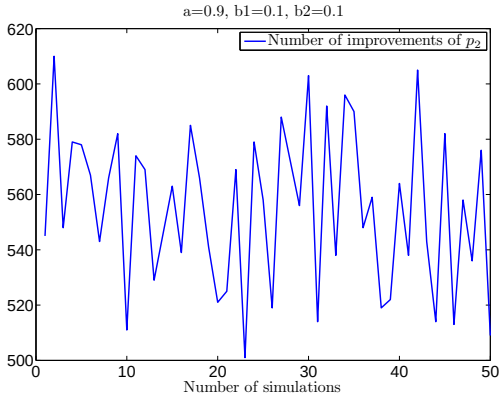
PSO algorithm performance for brachistochrone problem

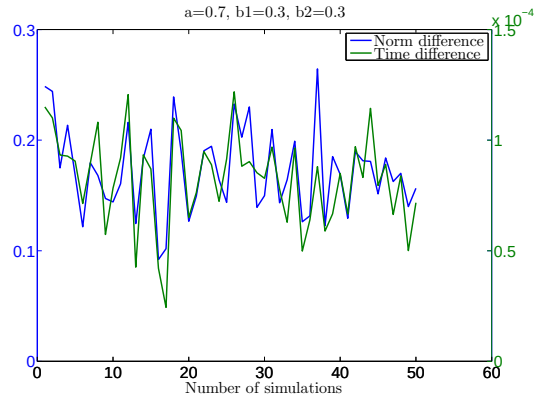
B.1 Parameter selection

In this appendix the performance of the particle swarm optimization algorithm can be found for the optimization of the brachistochrone problem for all the different parameter selection cases. In the figure the frames on the left show the difference in the norm and the difference in traveling time between the exact solution and the best approximation for each simulation run. The frames on the right show the number of improvements of the best approximation, p_2, for each simulation run.

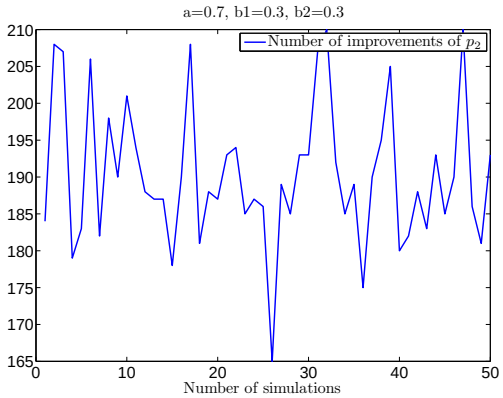

a=0.9, b1=0.1, b2=0.1
Norm difference
Time difference
x 10^-5
Number of simulations

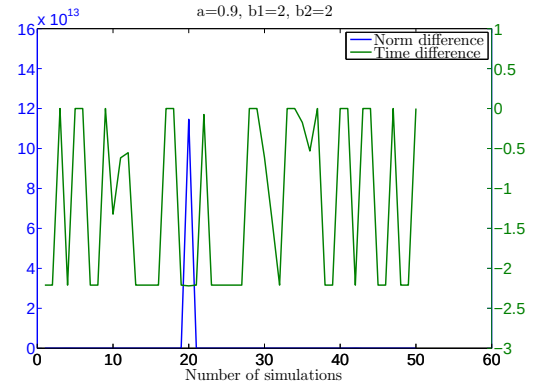
(a)


a=0.9, b1=0.1, b2=0.1
Number of improvements of p_2
Number of simulations

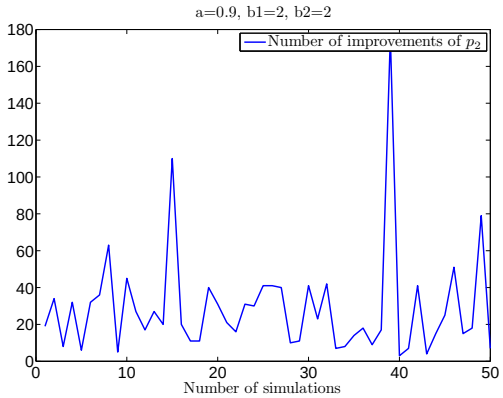

a=0.7, b1=0.3, b2=0.3
Norm difference
Time difference
x 10^-4
Number of simulations

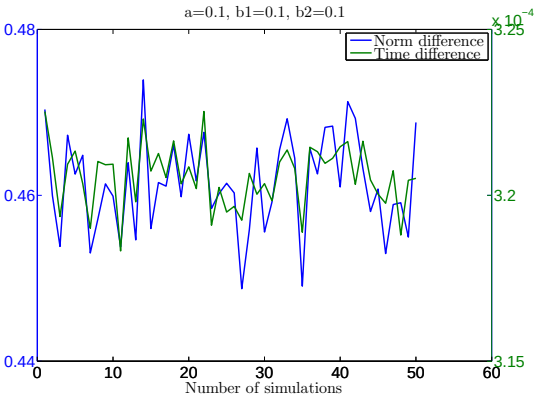
(b)


a=0.7, b1=0.3, b2=0.3
Number of improvements of p_2
Number of simulations

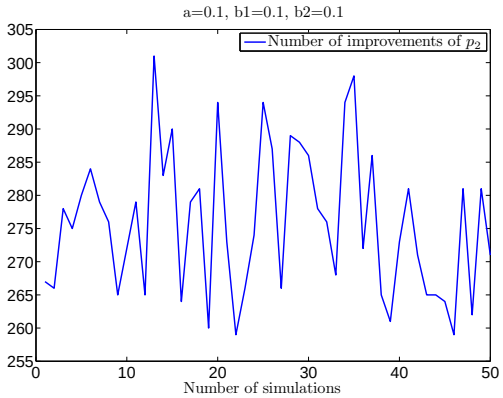

a=0.9, b1=2, b2=2
x 10^13
Norm difference
Time difference
Number of simulations

(c)


a=0.9, b1=2, b2=2
Number of improvements of p_2
Number of simulations


a=0.1, b1=0.1, b2=0.1
Norm difference
Time difference
x 10^-4
Number of simulations

(d)


a=0.1, b1=0.1, b2=0.1
Number of improvements of p_2
Number of simulations

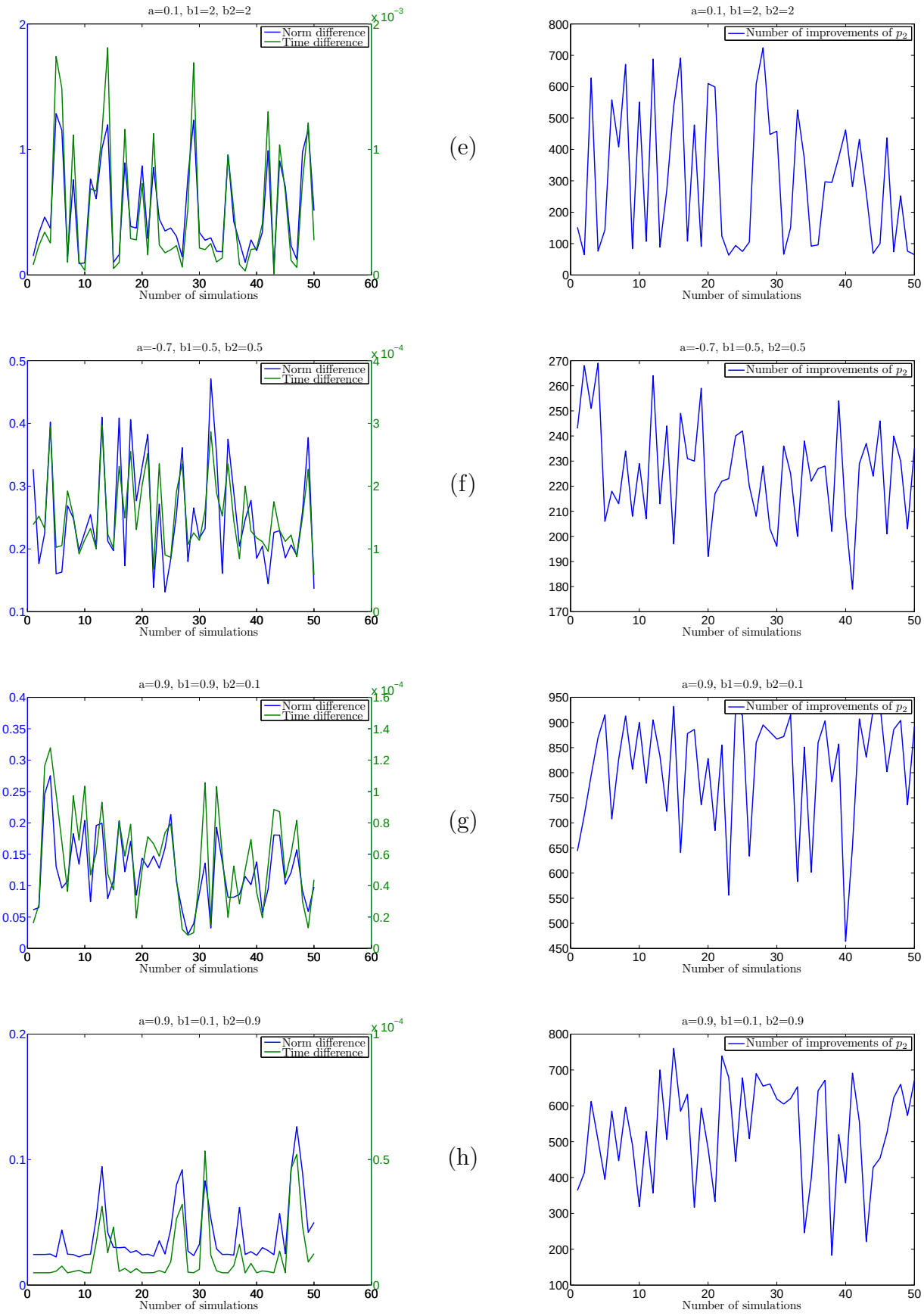
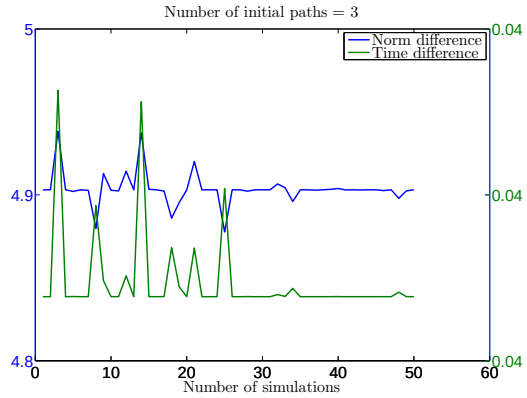


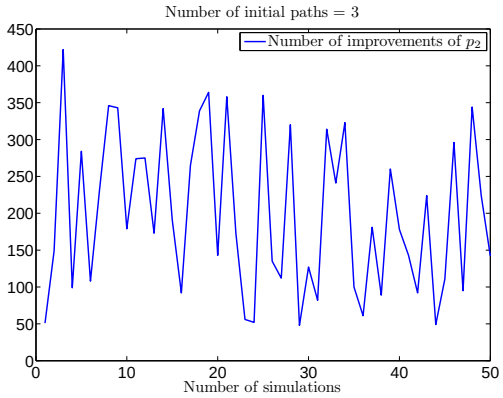
Figure B.1: Convergence behavior of particle swarm optimization algorithm, for several choices of the parameters a and b.

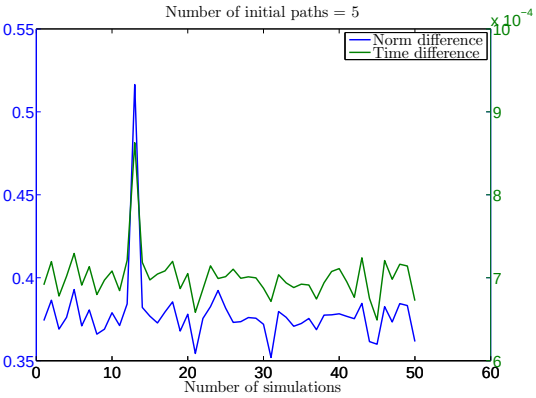
B.2 Number of initial paths

In this appendix the performance of the particle swarm optimization algorithm can be found for the optimization of the brachistochrone problem for the different cases of the number of initial paths.

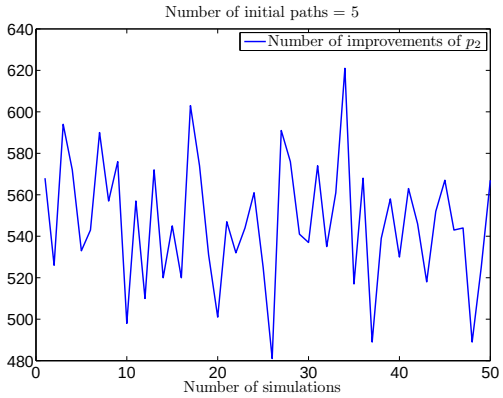


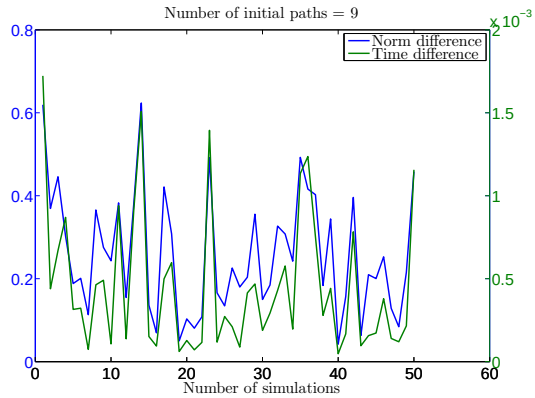
(a)



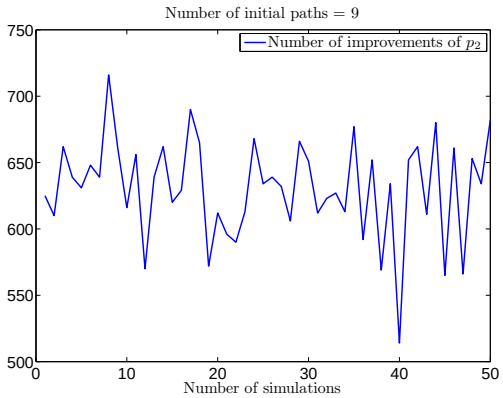


(b)





(c)



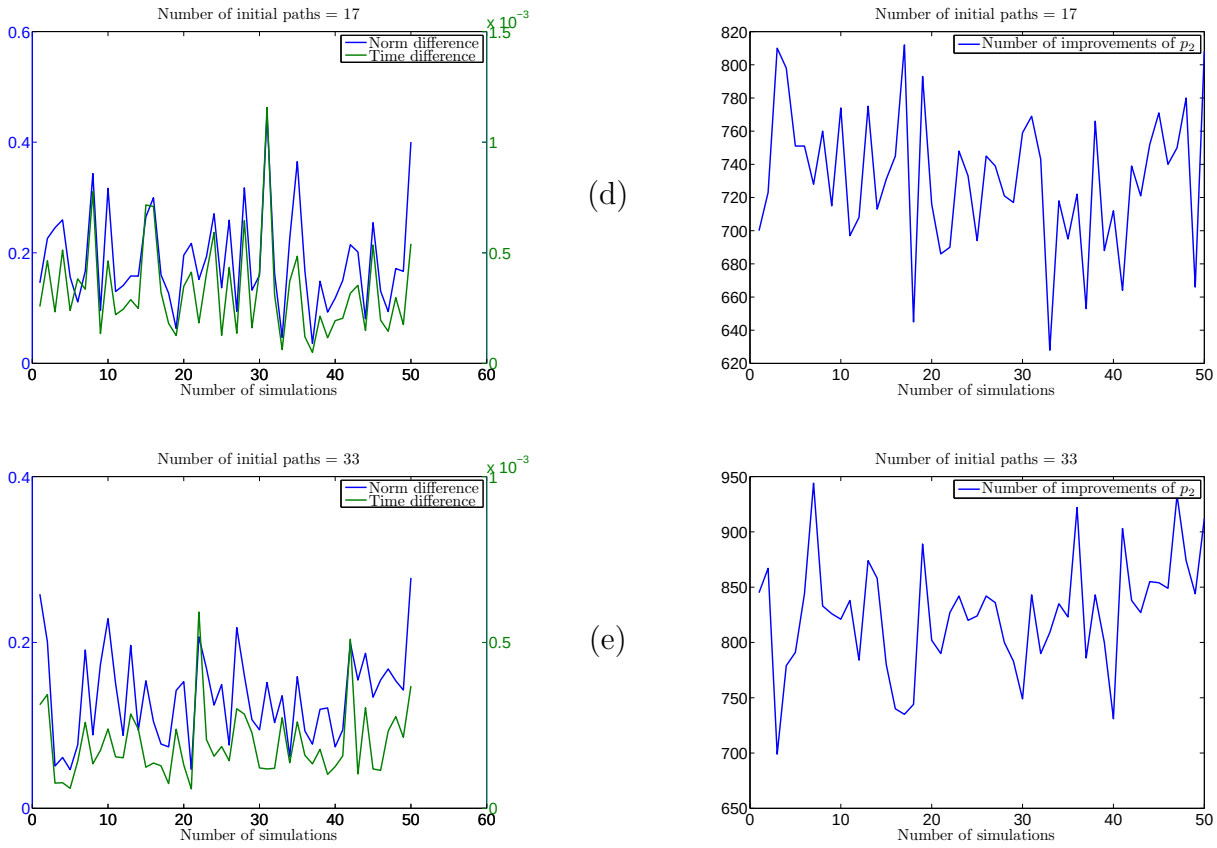


Figure B.2: Convergence behavior of particle swarm optimization algorithm, for several number of initial paths.

B.3 Distribution of initial paths

In this appendix the performance of the particle swarm optimization algorithm can be found for the optimization of the brachistochrone problem for the different distributions of the initial paths.

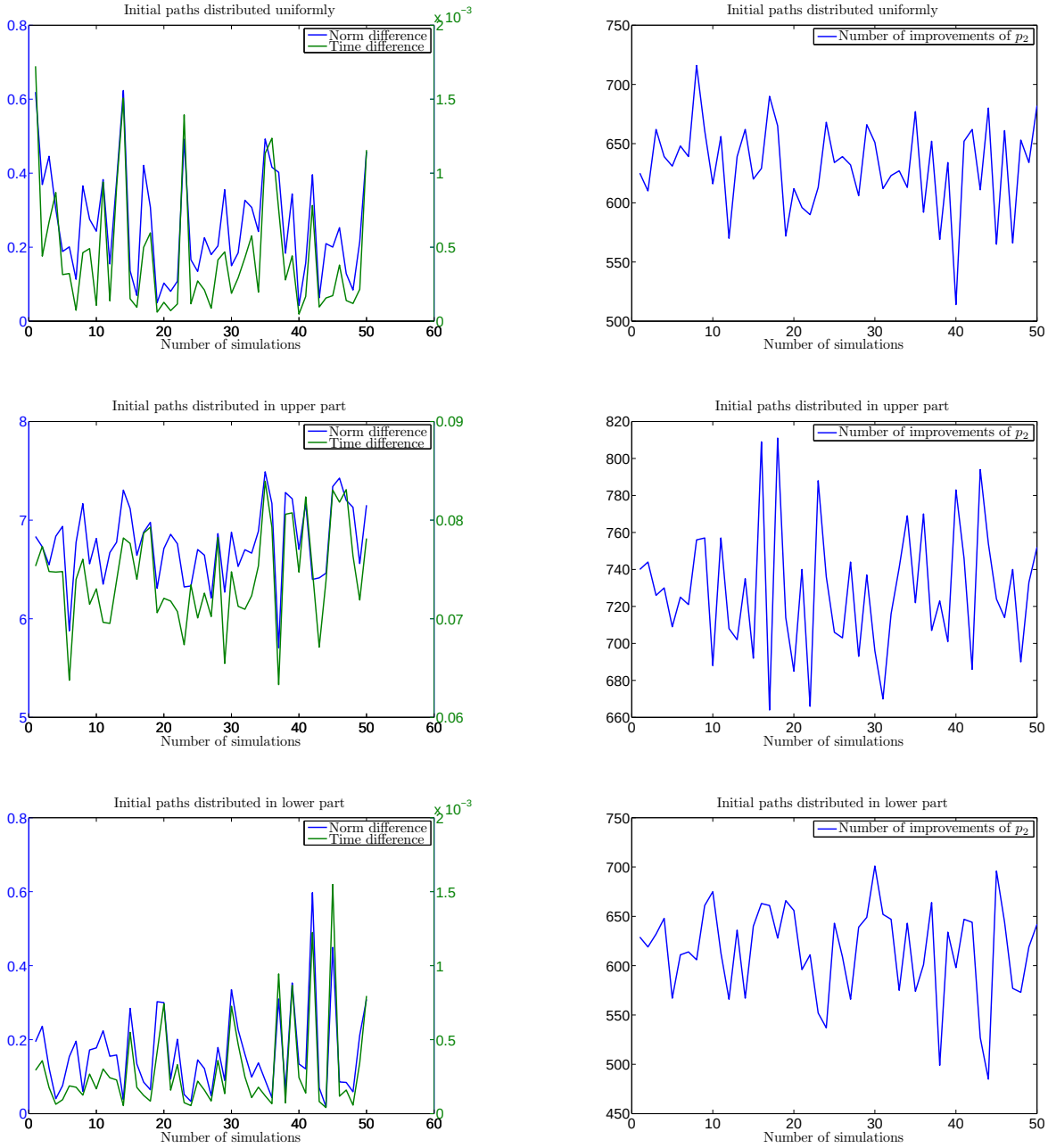


Figure B.3: Convergence behavior of particle swarm optimization algorithm, for different distributions of initial paths.

Bibliography

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization", *Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia*, 1995, pg. 1942-1948.

[2] M.M. Millonas, "Swarms, phase transitions and collective intelligence", *Artificial Life III*, Addison Wesley, Reading, Massachusetts, 1994.

[3] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory", *Proceedings of the Sixth International Symposium on Micro Machines and Human Science, Nagoya, Japan*, 1994, pg. 39-43.

[4] Y. Shi and R. Eberhart, "A modified particle swarm optimizer", *Proceedings of the IEEE International Conference on Evolutionary Computation, Anchorage, Alaska*, 1998, pg. 69-73.

[5] G. Venter and J. Sobieszczanski-Sobieski, "Particle swarm optimization", *Structures, Structural Dynamics, and Materials Conference, Denver, Colorado*, 2002, pg. 1-9.

[6] I.C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection", *Information Processing Letters*, Volume 85, 2003, pg. 317-325.

[7] J.A. van Maanen, "Het brachystochroonprobleem", *Een complexe grootheid, leven en werk van Johann Bernoulli*, Epsilon Uitgaven, Utrecht, The Netherlands, 1995.

[8] P.C. Fourie and A.A. Groenwold, "Particle swarms in size and shape optimization", *Proceedings of the International Workshop on Multidisciplinary Design Optimization, Pretoria, South Africa*, 2000, pg. 97-106.